DMI COLLEGE OF ENGINEERING

PALANCHUR, CHENNAI – 123.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CS1204 – DATASCIENCE ESSENTIALS

CS1204 DATA SCIENCE ESSENTIALS L T P C
3 0 2 4

COURSE OBJECTIVES:

- To understand the data science fundamentals, techniques and processes of data science
- To apply descriptive data analytics
- To learn to describe the relationship between data.
- To understand inferential data analytics
- To utilize the Python libraries for Data Wrangling.
- To present and interpret data using visualization libraries in Python

UNIT I INTRODUCTION 9

Data Science: Benefits and uses – facets of data - Data Science Process: Overview – Defining research goals – Retrieving data – Data preparation - Exploratory Data analysis (EDA) fundamentals – Significance of EDA – Making sense of data – Comparing EDA with classical and Bayesian analysis – Software tools for EDA - Visual Aids for EDA – managing data – cleaning and sampling for modeling and validation – Basic Statistical descriptions of Data

UNIT II DESCRIBING DATA 9

Types of Data - Types of Variables -Describing Data with Tables and Graphs –Describing Data with Averages - Frequency distributions – Outliers –Interpreting Distributions – graphs – averages - Describing Variability – interquartile range – variability for qualitative and ranked data - Normal Distributions and Standard (z) Scores

UNIT III DESCRIBING RELATIONSHIPS AND INFERENTIAL ANALYTICS 9

Correlation –Scatter plots –correlation coefficient for quantitative data –computational formula for correlation coefficient – Regression –regression line –least squares regression line – Standard error of estimate – interpretation of r2 –multiple regression equations –regression towards the mean - Hypothesis testing

UNIT IV PYTHON LIBRARIES FOR DATA WRANGLING 9

Basics of Numpy arrays –aggregations –computations on arrays –comparisons, masks, boolean logic – fancy indexing – structured arrays – Data manipulation with Pandas – data indexing and selection – operating on data – missing data – Hierarchical indexing – combining datasets – aggregation and grouping – pivot tables

UNIT V DATA VISUALIZATION 9

Importing Matplotlib – Line plots – Scatter plots – visualizing errors – density and contour plots – Histograms – text and annotation – customization – three-dimensional plotting - Geographic Data with Basemap - Visualization with Seaborn - multiple plots in one window – exporting graph using graphics parameters - Case studies - Create a dashboard using any visualization tool (Tableau public / Power BI).

TEXT BOOKS

1. Robert S. Witte and John S. Witte, ―Statistics‖, Eleventh Edition, Wiley Publications, 2017.
2. Jake VanderPlas, ―Python Data Science Handbook‖, O'Reilly, 2016.

UNIT I
INTRODUCTION

Data Science: Benefits and uses – facets of data - Data Science Process: Overview – Defining research goals – Retrieving data – Data preparation - Exploratory Data analysis (EDA) fundamentals – Significance of EDA – Making sense of data – Comparing EDA with classical and Bayesian analysis – Software tools for EDA - Visual Aids for EDA – managing data – cleaning and sampling for modeling and validation – Basic Statistical descriptions of Data

Data

In computing, data is information that has been translated into a form that is efficient for movement or processing

DATA SCIENCE

Data science is the areas of study which involves extracting insights from vast amounts of data using various scientific methods, algorithm and Process.

Bigdata

Data science focuses on processing the huge volume of heterogeneous data known as Bigdata.

- The characteristics of big data are often referred to as the three Vs:
  - Volume—How much data is there?
  - Variety—How diverse are different types of data?
  - Velocity—At what speed is new data generated?
- Fourth V:
- Veracity: How accurate is the data?

Data science is an evolutionary extension of statistics capable of dealing with the massive amounts of data produced today.

1.1 BENEFITS AND USES OF DATA SCIENCE

- Data science and big data are used almost everywhere in both commercial and non- commercial settings.
- Commercial companies in almost every industry use data science and big data to gain insights into their customers, processes, staff, completion, and products.
- Many companies use data science to offer customers a better user experience.
  - Eg: Google AdSense, which collects data from internet users so relevant commercial messages can be matched to the person browsing the internet
  - MaxPoint - example of real-time personalized advertising.
- Human resource professionals:
  - people analytics and text mining to screen candidates,
  - monitor the mood of employees, and
  - study informal networks among coworkers
- Financial institutions use data science:
  - to predict stock markets, determine the risk of lending money, and
  - learn how to attract new clients for their services
- Governmental organizations:
  - internal data scientists to discover valuable information,
  - share their data with the public
  - Eg: Data.gov is but one example; it's the home of the US Government's open data.

- organizations collected 5 billion data records from widespread applications such as Google Maps, Angry Birds, email, and text messages, among many other data sources.

- Nongovernmental organizations:
  - World Wildlife Fund (WWF), for instance, employs data scientists to increase the effectiveness of their fundraising efforts.
  - Eg: DataKind is one such data scientist group that devotes its time to the benefit of mankind.
- Universities:
  - Use data science in their research but also to enhance the study experience of their students.
  - massive open online courses (MOOC) produces a lot of data, which allows universities to study how this type of learning can complement traditional classes.
  - Eg: Coursera, Udacity, and edX

1.2 FACETS OF DATA

In data science and big data, it has many different types of data, and each of them tends to require different tools and techniques. The main categories of data are these:

- Structured
- Unstructured
- Natural language
- Machine-generated
- Graph-based
- Audio, video, and images
- Streaming

Structured data

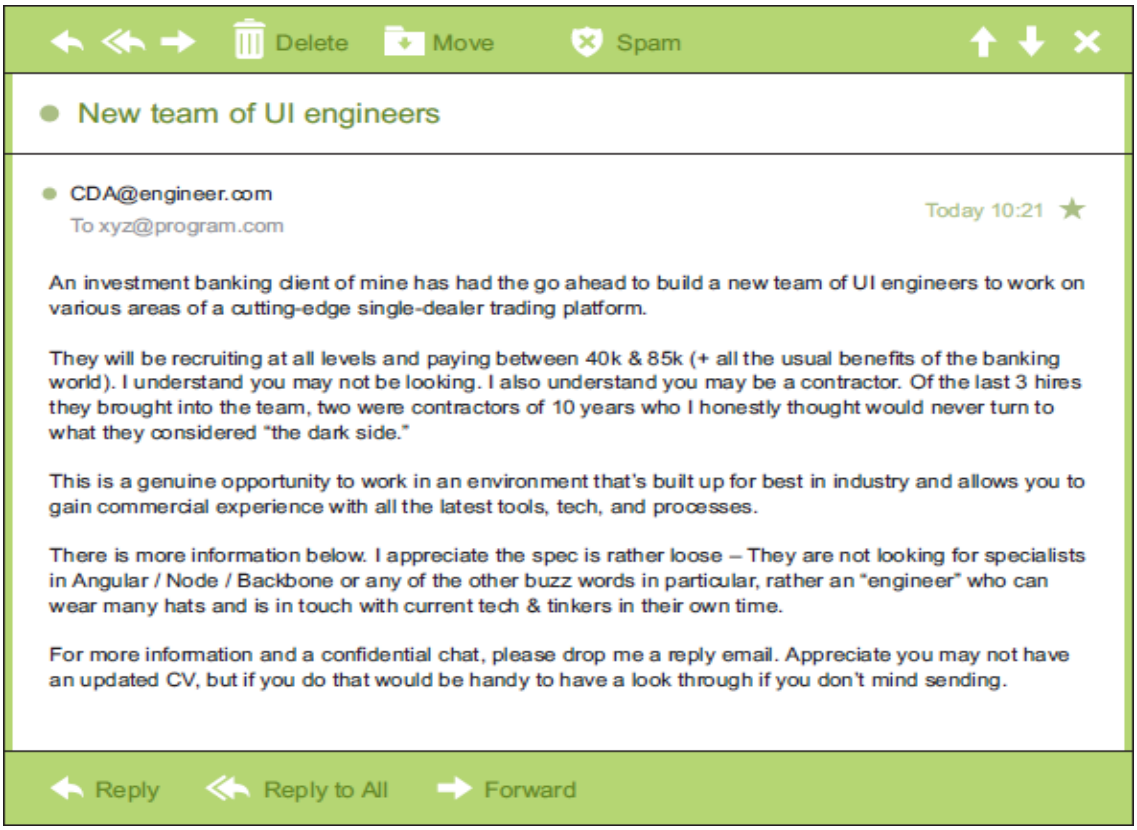
- Structured data is data that depends on a data model and resides in a fixed field within a record. As such, it's often easy to store structured data in tables within databases or Excel files

- SQL, or Structured Query Language, is the preferred way to manage and query data that resides in databases.

	Indicator ID	Dimension List	Timeframe	Numeric Value	Missing Value Flag	Confidence Inte
2	214390830	Total (Age-adjusted)	2008	74.6%		73.8%
3	214390833	Aged 18-44 years	2008	59.4%		58.0%
4	214390831	Aged 18-24 years	2008	37.4%		34.6%
5	214390832	Aged 25-44 years	2008	66.9%		65.5%
6	214390836	Aged 45-64 years	2008	88.6%		87.7%
7	214390834	Aged 45-54 years	2008	86.3%		85.1%
8	214390835	Aged 55-64 years	2008	91.5%		90.4%
9	214390840	Aged 65 years and over	2008	94.6%		93.8%
10	214390837	Aged 65-74 years	2008	93.6%		92.4%
11	214390838	Aged 75-84 years	2008	95.6%		94.4%
12	214390839	Aged 85 years and over	2008	96.0%		94.0%
13	214390841	Male (Age-adjusted)	2008	72.2%		71.1%
14	214390842	Female (Age-adjusted)	2008	76.8%		75.9%
15	214390843	White only (Age-adjusted)	2008	73.8%		72.9%
16	214390844	Black or African American only (Age-adjusted)	2008	77.0%		75.0%
17	214390845	American Indian or Alaska Native only (Age-adjusted)	2008	66.5%		57.1%
18	214390846	Asian only (Age-adjusted)	2008	80.5%		77.7%
19	214390847	Native Hawaiian or Other Pacific Islander only (Age-adjusted)	2008	DSU		
20	214390848	2 or more races (Age-adjusted)	2008	75.6%		69.6%

Unstructured data

Unstructured data is data that isn't easy to fit into a data model because the content is context-specific or varying. One example of unstructured data is email



Natural language

- Natural language is a special type of unstructured data; it's challenging to process because it requires knowledge of specific data science techniques and linguistics.
- The natural language processing community has had success in entity recognition, topic recognition, summarization, text completion, and sentiment analysis, but models trained in one domain don't generalize well to other domains.
- Even state-of-the-art techniques aren't able to decipher the meaning of every piece of text.

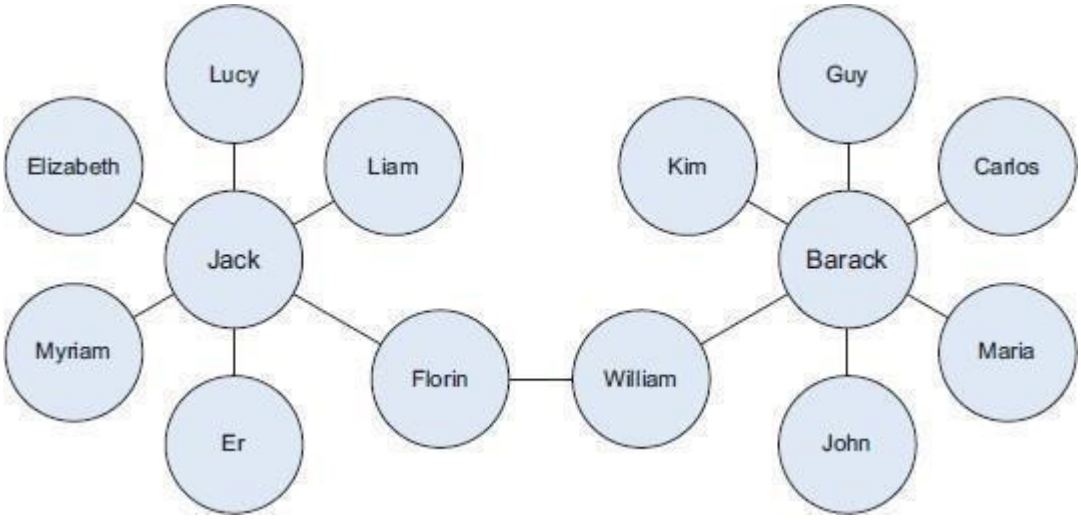
Machine-generated data

- Machine-generated data is information that's automatically created by a computer, process, application, or other machine without human intervention.
- Machine-generated data is becoming a major data resource and will continue to do so.
- The analysis of machine data relies on highly scalable tools, due to its high volume and speed. Examples of machine data are web server logs, call detail records, network event logs, and telemetry.

```
CSIPERF:TXCOMMIT;313236
2014-11-28 11:36:13, Info                  CSI    00000153 Creating NT transaction (seq
69), objectname [6]"(null)"
2014-11-28 11:36:13, Info                  CSI    00000154 Created NT transaction (seq 69)
result 0x00000000, handle @0x4e54
2014-11-28 11:36:13, Info                  CSI    00000155@2014/11/28:10:36:13.471
Beginning NT transaction commit...
2014-11-28 11:36:13, Info                  CSI    00000156@2014/11/28:10:36:13.705 CSI perf
```

Graph based or network - data

- ―Graph data‖ can be a confusing term because any data can be shown in a graph
- Graph or network data is, in short, data that focuses on the relationship or adjacency of objects.
- The graph structures use nodes, edges, and properties to represent and store graphical data.
- Graph-based data is a natural way to represent social networks, and its structure allows you to calculate specific metrics such as the influence of a person and the shortest path between two people.
- Graph-based data can be found on many social media websites.
- Eg: LinkedIn, Twitter, movie interests on Netflix



Audio, image, and video

- Audio, image, and video are data types that pose specific challenges to a data scientist.
- Recognizing objects in pictures, turn out to be challenging for computers.
- Major League Baseball Advanced Media - video capture to approximately 7 TB per
- game for the purpose of live, in-game analytics.
- High-speed cameras at stadiums will capture ball and athlete movements to calculate in real time.
- DeepMind succeeded at creating an algorithm that's capable of learning how to play video games.
- This algorithm takes the video screen as input and learns to interpret everything via a complex process of deep learning.
- Google – Artificial Intelligence Development plans

Streaming data

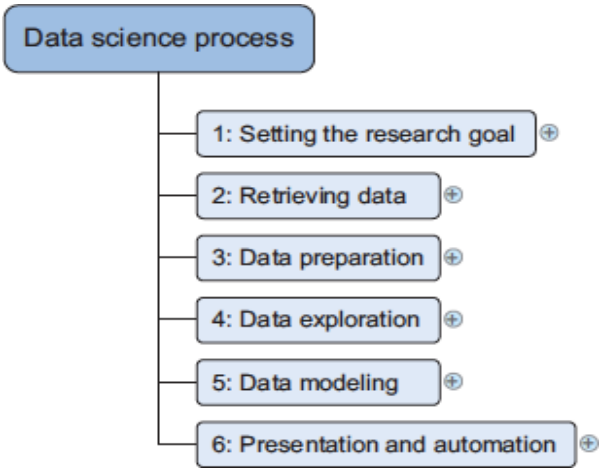
- The data flows into the system when an event happens instead of being loaded into a data store in a batch.
- Examples are the ―What's trending‖ on Twitter, live sporting or music events, and the stock market.

1.3 DATA SCIENCE PROCESS

Overview of the data science process

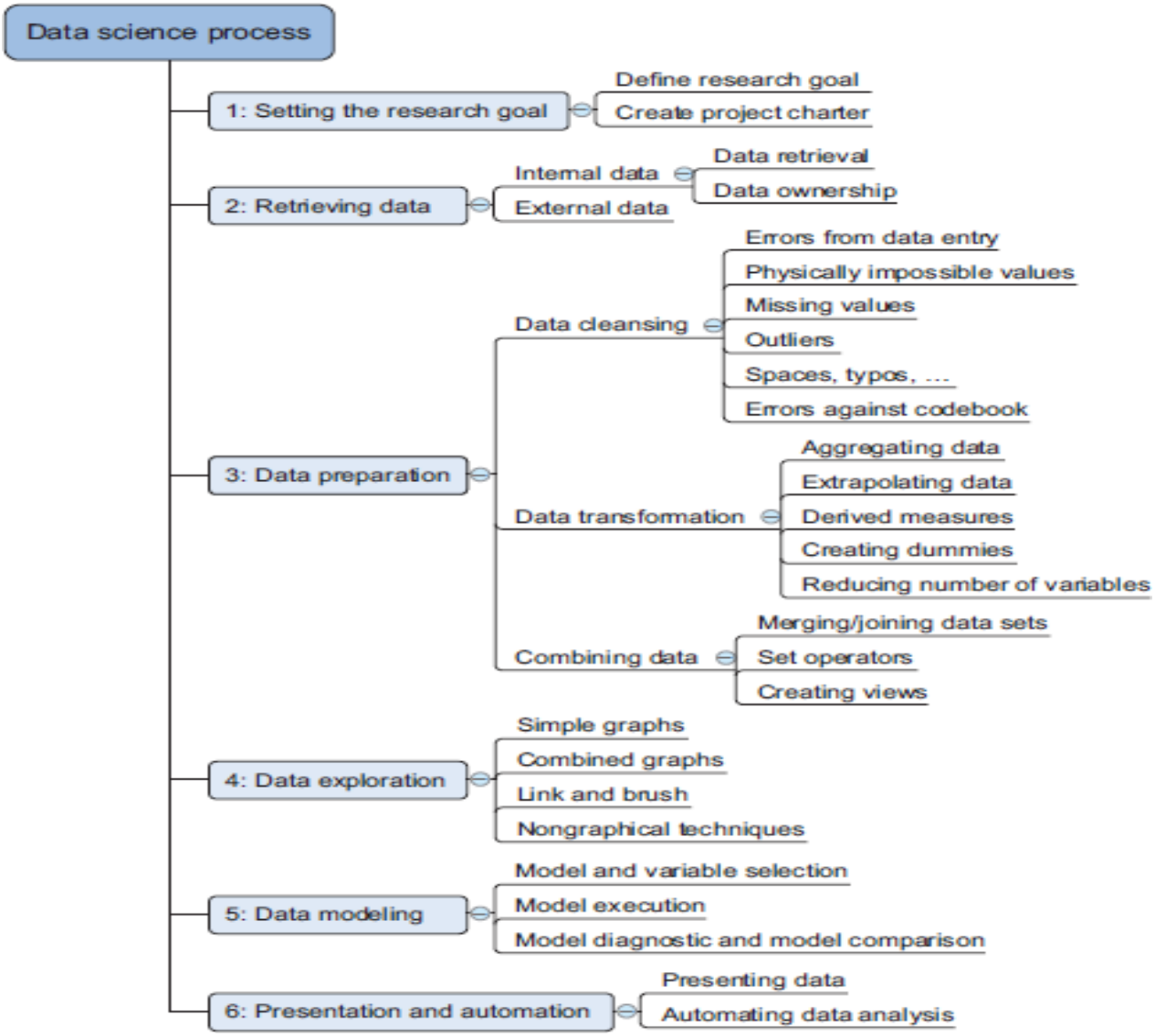
The typical data science process consists of six steps

- Setting the research goal
- Retrieving data
- Data preparation
- Data exploration
- Data modeling or model building
- Presentation and automation



Overview of the data science process:

- A structured data science approach helps you maximize your chances of success in a data science project at the lowest cost.
- The first step of this process is setting a research goal.
- The main purpose here is to make sure all the stakeholders understand the what, how, and why of the project.
- Draw the result in a project charter.



1. The first step of this process is setting a research goal. The main purpose here is making sure all the stakeholders understand the what, how, and why of the project. In every serious project this will result in a project charter.
2. The second phase is data retrieval. The data is required for analysis, so this step includes finding suitable data and getting access to the data from the data owner. The result is data in its raw form, which probably needs transformation before it becomes usable.
3. The third step is data preparation. This includes transforming the data from a raw form into data that is directly usable in the models. To achieve this, detect and correct different kinds of errors in the data, combine data from different data sources and transform it. After completing this step, we can progress to data visualization and modeling.
4. The fourth step is data exploration. The goal of this step is to gain a deep understanding of the data, look for patterns, correlations, deviations based on visual and descriptive techniques. The insights gained from this phase will enable us to start modeling.
5. The fifth step is data modeling. It is now that you attempt to gain the insights or make the predictions stated in your project charter.
6. The last step of the data science model is presenting the results and automating the analysis, if needed. One goal of a project is to change a process and/or make better decisions. The importance of this step is more apparent in projects on a strategic and tactical level. Certain projects require performing the business process over and over again, so automating the project will save time.

1.4 DEFINING RESEARCH GOALS

A project starts by understanding the what, the why, and the how of our project. The outcome should be a clear research goal, a good understanding of the context, well-defined deliverables, and a plan of action with a timetable. This information is then best placed in a project charter.

Spend time understanding the goals and context of your research

- An essential outcome is the research goal that states the purpose of our assignment in a clear and focused manner.
- Understanding the business goals and context is critical for project success.
- Continue asking questions and devising examples until you grasp the exact business expectations, identify how your project fits in the bigger picture, appreciate how your research is going to change the business, and understand how they'll use your results

Create a project charter

A project charter requires teamwork, and input covers at least the following:

- A clear research goal
- The project mission and context
- How you're going to perform your analysis
- What resources you expect to use
- Proof that it's an achievable project, or proof of concepts
- Deliverables and a measure of success
- A timeline

1.5 RETRIEVING DATA

- The next step in data science is to retrieve the required data. Sometimes there is need to go into the field and design a data collection process yourself, but most of the time you won't be involved in this step.
- Many companies will have already collected and stored the data, and what they don't have can often be bought from third parties.

- More and more organizations are making even high-quality data freely available for public and commercial use.
- Data can be stored in many forms, ranging from simple text files to tables in a database. The objective now is acquiring all the data to need.

Start with data stored within the company (Internal data)

- Most companies have a program for maintaining key data, so much of the cleaning work may already be done. This data can be stored in official data repositories such as databases, data marts, data warehouses, and data lakes maintained by a team of IT professionals.
- Data warehouses and data marts are home to preprocessed data, data lakes contain data in its natural or raw format.
- Finding data even within your own company can sometimes be a challenge. As companies grow, their data becomes scattered around many places. the data may be dispersed as people change positions and leave the company.
- Getting access to data is another difficult task. Organizations understand the value and sensitivity of data and often have policies in place so everyone has access to what they need and nothing more.
- These policies translate into physical and digital barriers called Chinese walls. These ―walls‖ are mandatory and well-regulated for customer data in most countries.

External Data

- If data isn't available inside your organization, look outside your organizations. Companies provide data so that you, in turn, can enrich their services and ecosystem. Such is the case with Twitter, LinkedIn, and Facebook.
- More and more governments and organizations share their data for free with the world.
- A list of open data providers that should get you started.

Open data site	Description
Data.gov	The home of the US Government's open data
https://open-data.europa.eu/	The home of the European Commission's open data
Freebase.org	An open database that retrieves its information from sites like Wikipedia, MusicBrains, and the SEC archive
Data.worldbank.org	Open data initiative from the World Bank
Aiddata.org	Open data for international development
Open.fda.gov	Open data from the US Food and Drug Administration

Do data quality checks to prevent problems

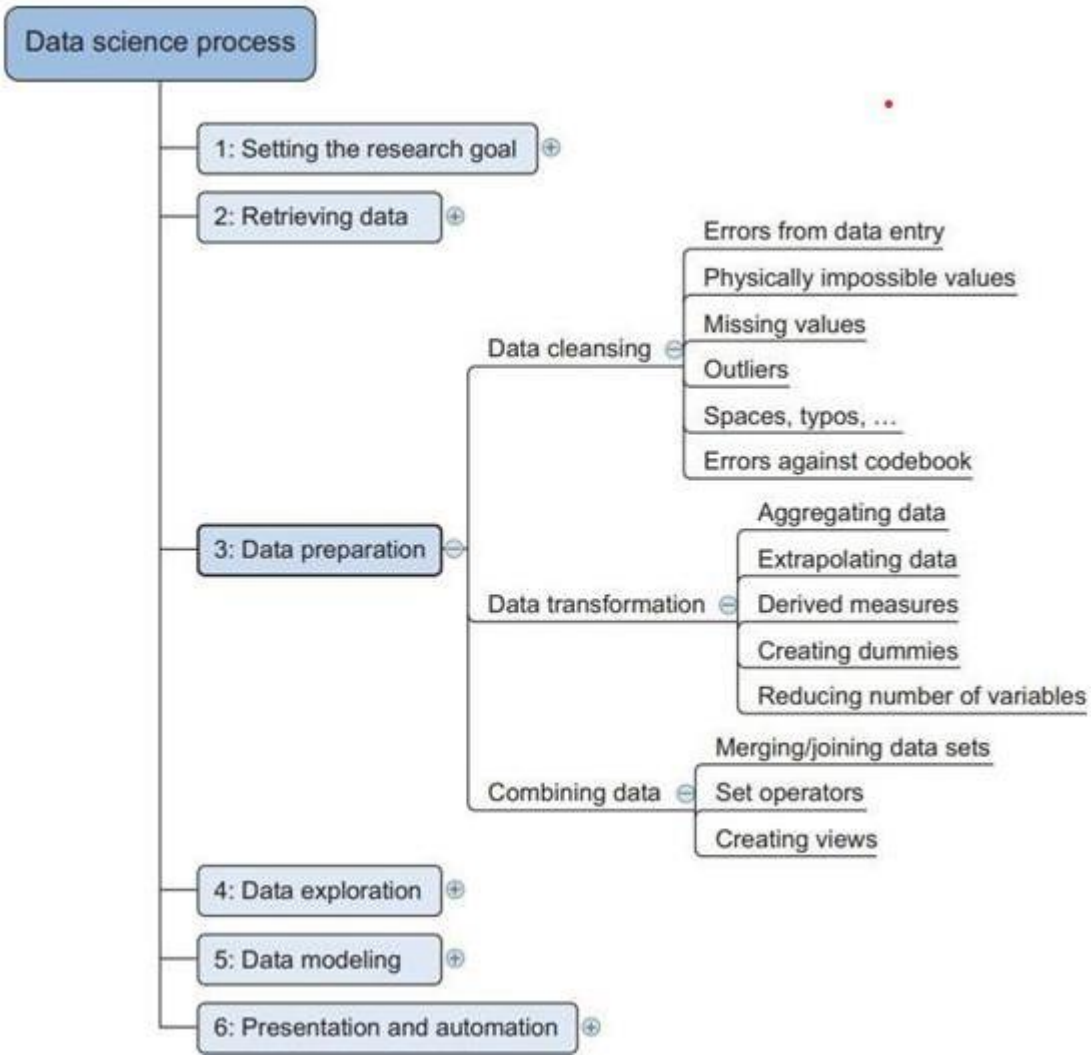
A good portion of the project time should be spent on doing data correction and cleansing. The retrieval of data is the first time we'll inspect the data in the data science process. Most of the errors encountered during the data gathering phase are easy to spot.

The data is investigated during the import, data preparation and exploratory phases. The difference is in the goal and the depth of the investigation. During data retrieval, check to see if the data is equal to the data in the source document and to see if we have the right data types.

1.6 DATA PREPARATION (CLEANSING, INTEGRATING, TRANSFORMING DATA)

The data received from the data retrieval phase is likely to be ―a diamond in the rough.‖ so the task is to prepare it for use in the modeling and reporting phase. The model needs the data in specific format, so data transformation will always come into play.

It is good to correct data errors as early on in the process as possible. Figure shows the most common actions to take during the data cleansing, integration, and transformation phase.



Cleansing data

Data cleansing is a sub process of the data science process that focuses on removing errors in your data so your data becomes a true and consistent representation of the processes it originates from.

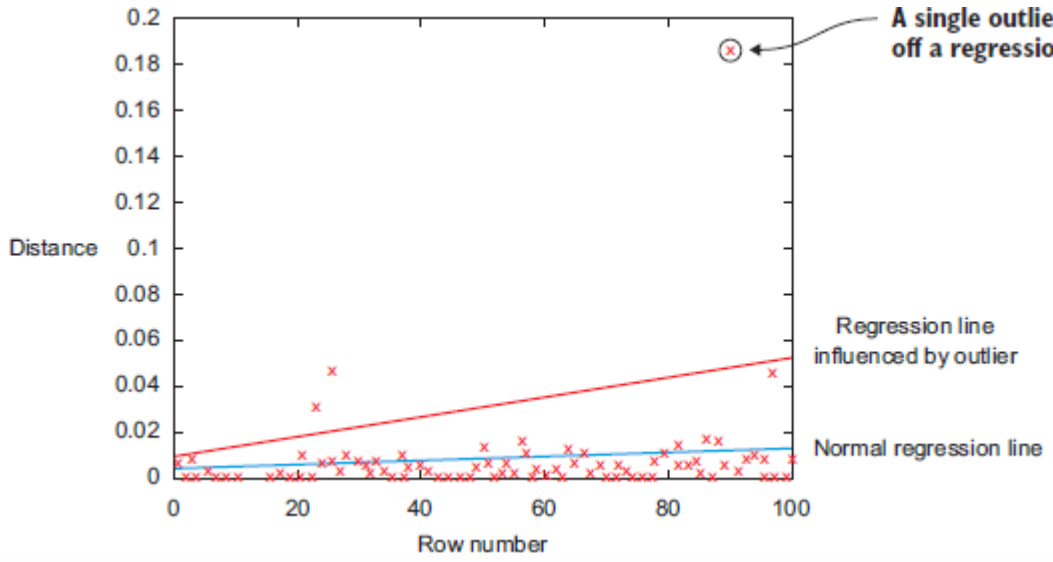
- The first type is the interpretation error, such as when you take the value in your data for granted, like saying that a person's age is greater than 300 years.
- The second type of error points to inconsistencies between data sources or against your company's standardized values.

An example of this class of errors is putting ―Female‖ in one table and ―F‖ in another when they represent the same thing: that the person is female.

Overview of common errors

General solution	
Try to fix the problem early in the data acquisition chain or else fix it in the program.	
Error description	**Possible solution**
Errors pointing to false values within one data set	
Mistakes during data entry	Manual overrules
Redundant white space	Use string functions
Impossible values	Manual overrules
Missing values	Remove observation or value
Outliers	Validate and, if erroneous, treat as missing value (remove or insert)
Errors pointing to inconsistencies between data sets	
Deviations from a code book	Match on keys or else use manual overrules
Different units of measurement	Recalculate
Different levels of aggregation	Bring to same level of measurement by aggregation or extrapolation

Sometimes use more advanced methods, such as simple modeling, to find and identify data errors; diagnostic plots can be especially insightful. For example, in figure we use a measure to identify data points that seem out of place. We do a regression to get acquainted with the data and detect the influence of individual observations on the regression line.



The encircled point influences the model heavily and is worth investigating because it can point to a region where you don't have enough data or might indicate an error in the data, but it also can be a valid data point.

Data Entry Errors

- Data collection and data entry are error-prone processes. They often require human intervention, and introduce an error into the chain.
- Data collected by machines or computers isn't free from errors. Errors can arise from human sloppiness, whereas others are due to machine or hardware failure.
- Detecting data errors when the variables study doesn't have many classes can be done by tabulating the data with counts.
- When a variable that can take only two values have: —Good‖ and —Bad‖, we can create a frequency table and see if those are truly the only two values present. In table the values —Godo‖ and —Bade‖ point out something went wrong in at least 16 cases.

Value	Count
Good	1598647
Bad	1354468
Godo	15
Bade	1

Most errors of this type are easy to fix with simple assignment statements and if-thenelse rules:
f x == —Godo‖:
x = —Good‖
if x == —Bade‖:
x = —Bad‖

Redundant Whitespace

- Whitespaces tend to be hard to detect but cause errors like other redundant characters would.
- The whitespace cause the miss match in the string such as —FR ‖ – —FR‖, dropping the observations that couldn't be matched.

Fixing redundant whitespaces is luckily easy enough in most programming languages. They all provide string functions that will remove the leading and trailing whitespaces. For instance, in Python we can use the strip() function to remove leading and trailing spaces.

Fixing Capital Letter Mismatches

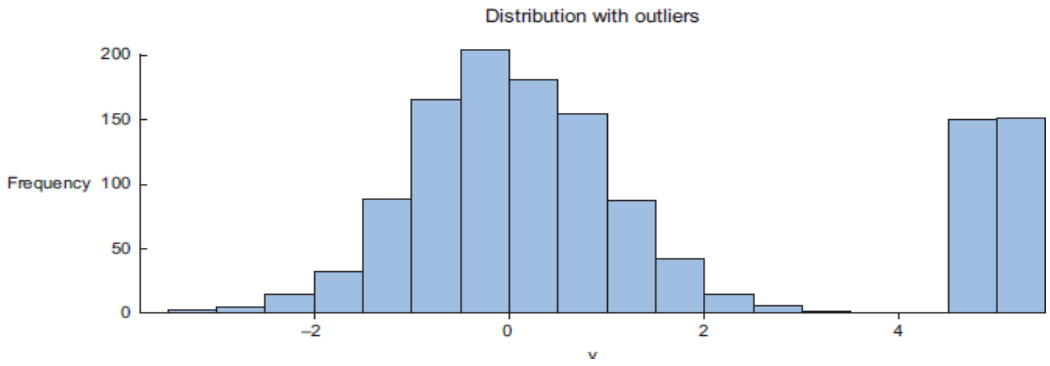
Capital letter mismatches are common. Most programming languages make a distinction between —Brazil‖ and —brazil‖.
In this case you can solve the problem by applying a function that returns both strings in lowercase, such as
.lower() in Python. —Brazil‖.lower() == —brazil‖.lower() should result in true.

Impossible Values and Sanity Checks

Check the value against physically or theoretically impossible values such as people taller than 3 meters or someone with an age of 299 years. Sanity checks can be directly expressed with rules:
check = 0 <= age <= 120

Outliers

An outlier is an observation that seems to be distant from other observations or, more specifically, one observation that follows a different logic or generative process than the other observations. The easiest way to find outliers is to use a plot or a table with the minimum and maximum values.
The plot on the top shows no outliers, whereas the plot on the bottom shows possible outliers on the upper side when a normal distribution is expected.



Dealing with Missing Values

Missing values aren't necessarily wrong, but you still need to handle them separately; certain modeling techniques can't handle missing values. They might be an indicator that something went wrong in your data collection or that an error happened in the ETL process. Common techniques data scientists use are listed in table

Integrating data

Your data comes from several different places, and in this substep we focus on integrating these different sources. Data varies in size, type, and structure, ranging from databases and Excel files to text documents.

The Different Ways of Combining Data

Perform two operations to combine information from different data sets.

- Joining
- Appending or stacking

Joining Tables

- Joining tables allows combining the information of one observation found in one table with the information that you find in another table. The focus is on enriching a single observation.
- Let's say that the first table contains information about the purchases of a customer and the other table contains information about the region where your customer lives.
- Joining the tables allows combining the information, as shown in figure.

Client	Item	Month
John Doe	Coca-Cola	January
Jackie Qi	Pepsi-Cola	January

Client	Region
John Doe	NY
Jackie Qi	NC

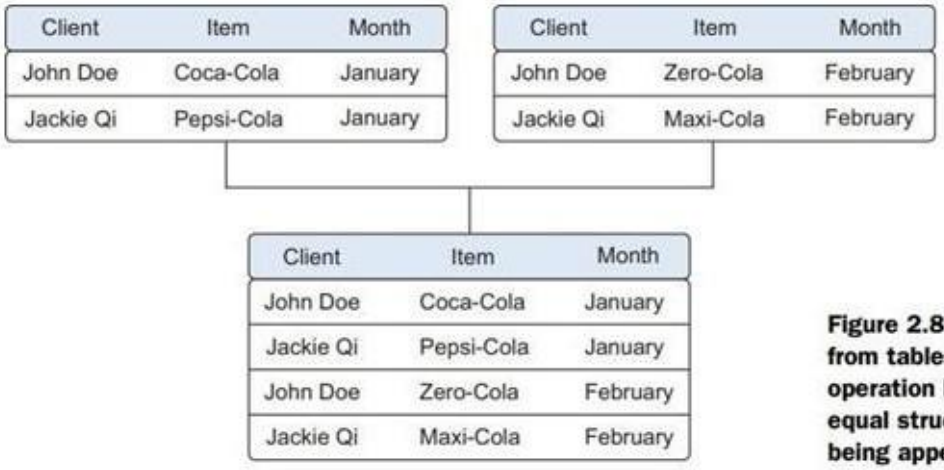
Client	Item	Month	Region
John Doe	Coca-Cola	January	NY
Jackie Qi	Pepsi-Cola	January	NC

To join tables, we use variables that represent the same object in both tables, such as a date, a country name, or a Social Security number. These common fields are known as keys. When these keys also uniquely define the records in the table they are called primary keys.
The number of resulting rows in the output table depends on the exact join type that can use

Appending Tables

Appending or stacking tables is effectively adding observations from one table to another table.

Appending or stacking tables is effectively adding observations from one table to another table. Figure shows an example of appending tables. One table contains the observations from the month January and the second table contains observations from the month February. The result of appending these tables is a larger one with the observations from January as well as February.



Client	Item	Month
John Doe	Coca-Cola	January
Jackie Qi	Pepsi-Cola	January

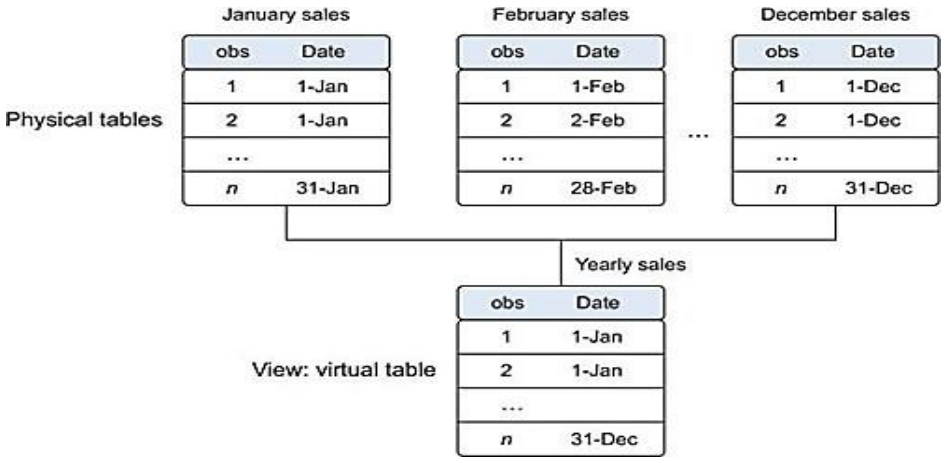
Client	Item	Month
John Doe	Zero-Cola	February
Jackie Qi	Maxi-Cola	February

Client	Item	Month
John Doe	Coca-Cola	January
Jackie Qi	Pepsi-Cola	January
John Doe	Zero-Cola	February
Jackie Qi	Maxi-Cola	February

Figure 2.8 Appending data from tables is a common operation but requires an equal structure in the tables being appended.

Using views to simulate data joins and appends

- To avoid duplication of data, we virtually combine data with views. The problem is that when we duplicate the data, more storage space is needed. In case of less data, that may not cause problems. But if every table consists of terabytes of data, then it becomes problematic to duplicate the data. For this reason, the concept of a view was invented.
- A view behaves as if we are working on a table, but this table is nothing but a virtual layer that ombines the tables. Figure shows how the sales data from the different months is combined virtually into a yearly sales table instead of duplicating the data.
- Views do come with a drawback, however. While a table join is only performed once, the join that creates the view is recreated every time it's queried, using more processing power than a pre- calculated table would have

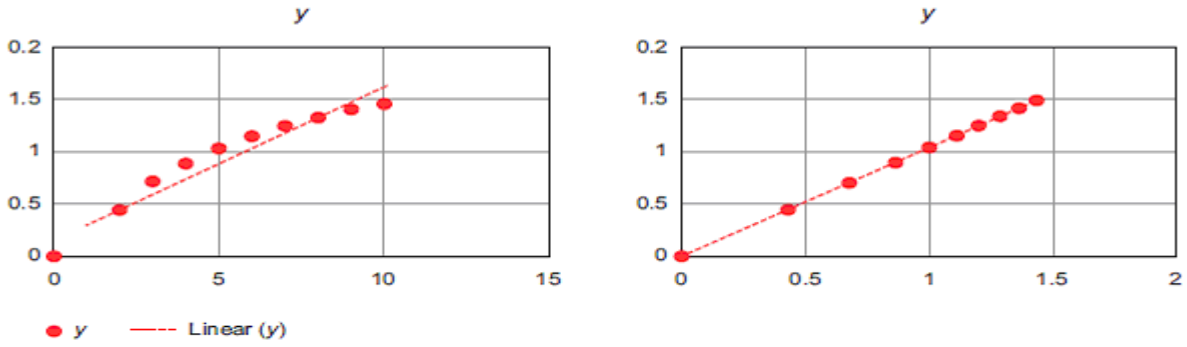


Transforming data

Certain models require their data to be in a certain shape. Transforming your data so it takes a suitable form for data modeling.

Relationships between an input variable and an output variable aren't always linear. Take, for instance, a relationship of the form $y = ae^{bx}$. Taking the log of the independent variables simplifies the estimation problem dramatically. Transforming the input variables greatly simplifies the estimation problem. Other times you might want to combine two variables into a new variable.

x	1	2	3	4	5	6	7	8	9	10
log(x)	0.00	0.43	0.68	0.86	1.00	1.11	1.21	1.29	1.37	1.43
y	0.00	0.44	0.69	0.87	1.02	1.11	1.24	1.32	1.38	1.46



Transforming x to log x makes the relationship between x and y linear (right), compared with the non-log x (left).

Reducing the Number of Variables

- Having too many variables in your model makes the model difficult to handle, and certain techniques don't perform well when you overload them with too many input variables. For instance, all the techniques based on a Euclidean distance perform well only up to 10 variables.
- Data scientists use special methods to reduce the number of variables but retain the maximum amount of data.

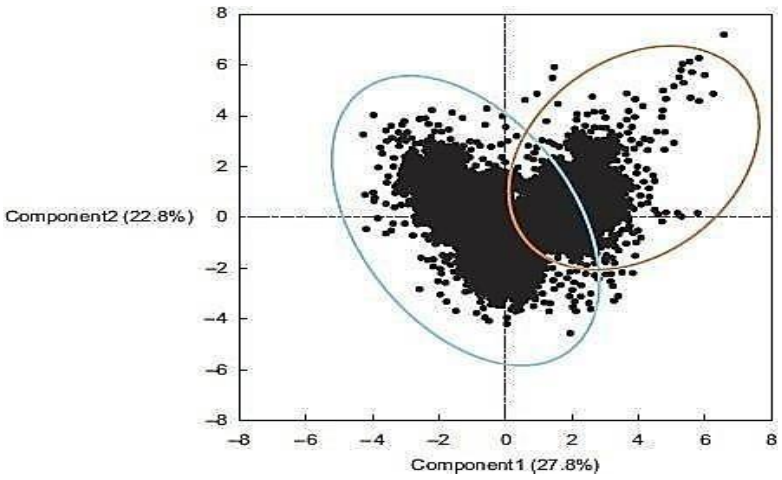


Figure shows how reducing the number of variables makes it easier to understand the key values. It also shows how two variables account for 50.6% of the variation within the data set (component1 = 27.8% + component2 = 22.8%). These variables, called ―component1‖ and ―component2,‖ are both combinations of the original variables. They're the principal components of the underlying data structure

Turning Variables into Dummies

- Dummy variables can only take two values: true(1) or false(0). They're used to indicate the absence of a categorical effect that may explain the observation.
- In this case you'll make separate columns for the classes stored in one variable and indicate it with 1 if the class is present and 0 otherwise.
- An example is turning one column named Weekdays into the columns Monday through Sunday. You use an indicator to show if the observation was on a Monday; you put 1 on Monday and 0 elsewhere.
- Turning variables into dummies is a technique that's used in modeling and is popular with, but not exclusive to, economists.

Figure. Turning variables into dummies is a data transformation that breaks a variable that has multiple classes into multiple variables, each having only two possible values: 0 or 1

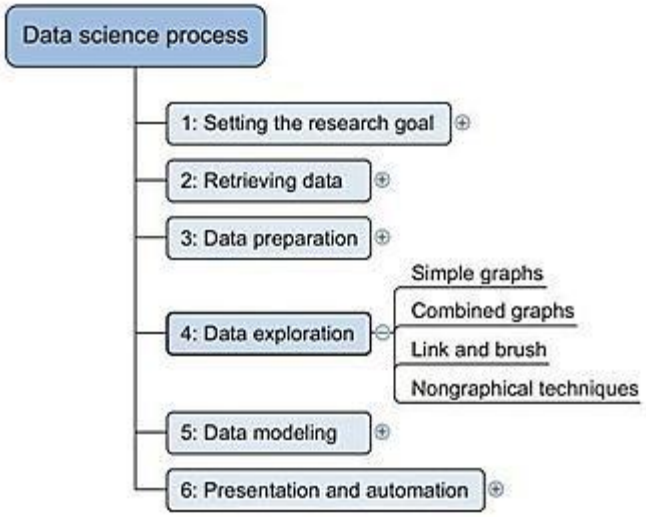
Customer	Year	Gender	Sales
1	2015	F	10
2	2015	M	8
1	2016	F	11
3	2016	M	12
4	2017	F	14
3	2017	M	13

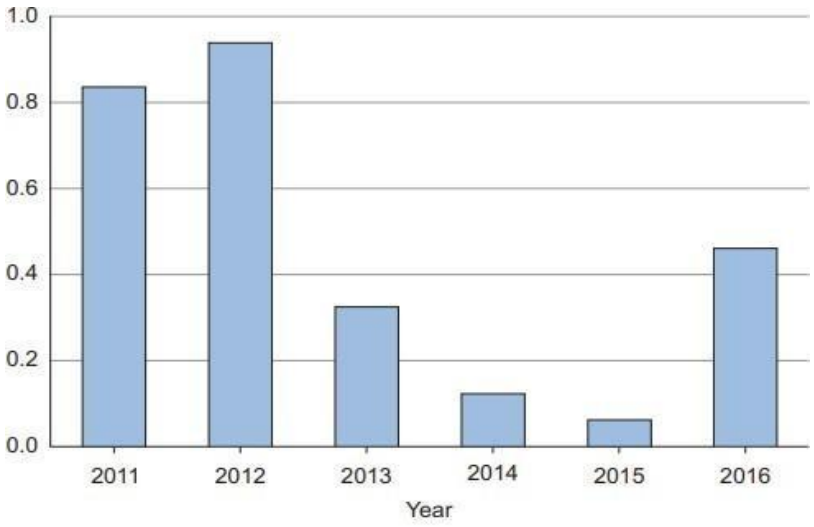
M F

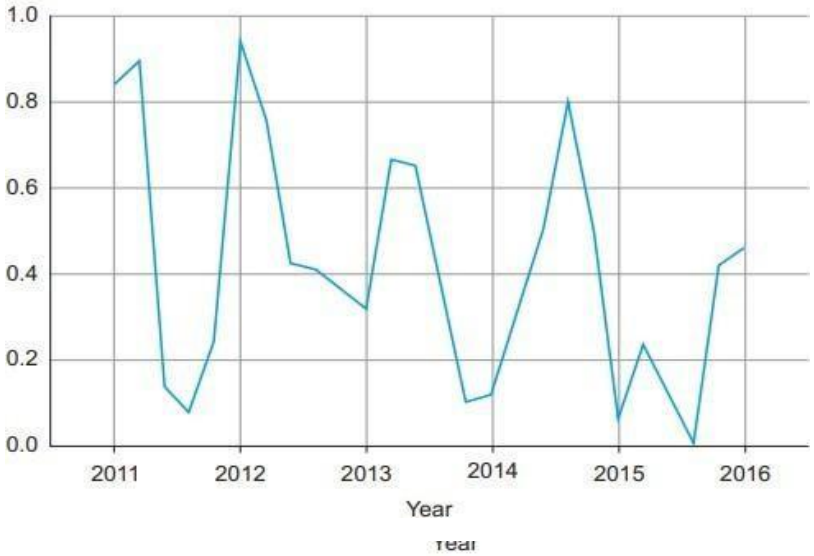
Customer	Year	Sales	Male	Female
1	2015	10	0	1
1	2016	11	0	1
2	2015	8	1	0
3	2016	12	1	0
3	2017	13	1	0
4	2017	14	0	1

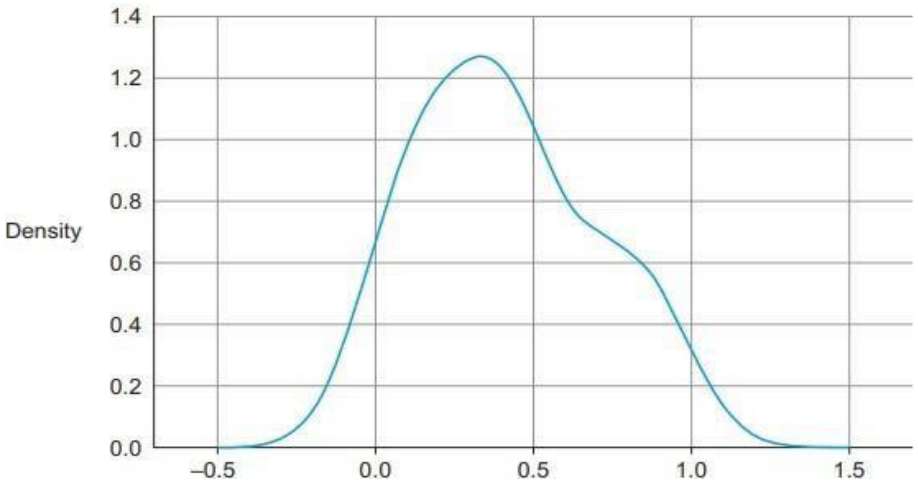
1.7 EXPLORATORY DATA ANALYSIS (EDA)

- Information becomes much easier to grasp when shown in a picture, therefore you mainly use graphical techniques to gain an understanding of your data and the interactions between variables. This phase is about exploring data, discovering anomalies missed before and to fix them.
- The visualization techniques used in this phase range from simple line graphs or histograms as shown in figure, to more complex diagrams such as Sankey and network graphs. Sometimes it's useful to compose a composite graph from simple graphs to get even more insight into the data. Other times the graphs can be animated or made interactive to make it easier.

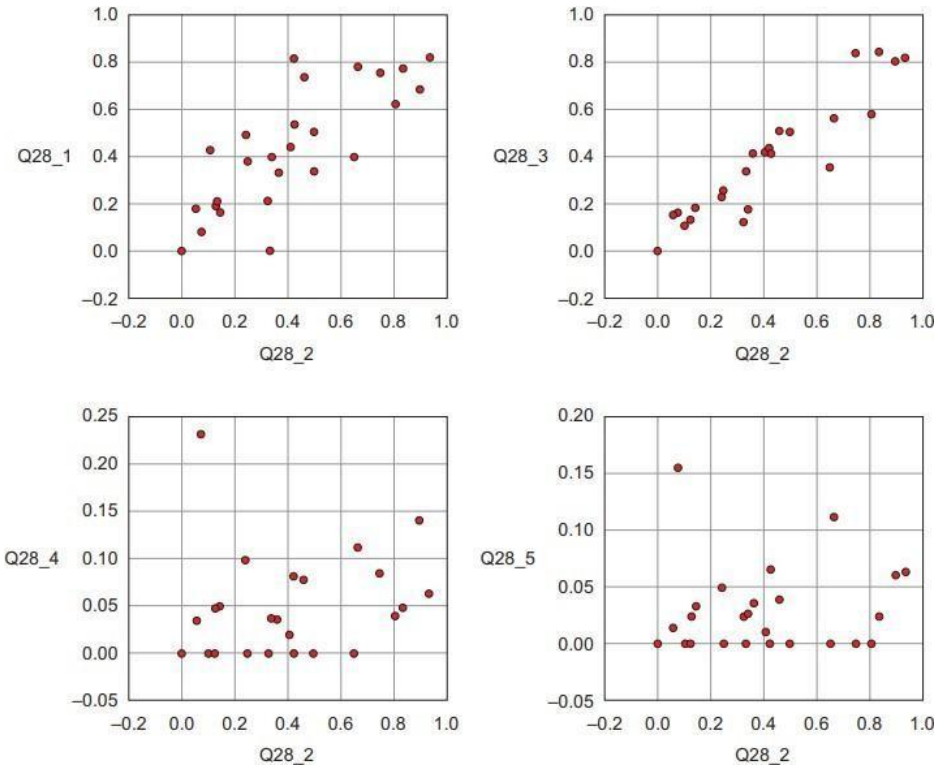


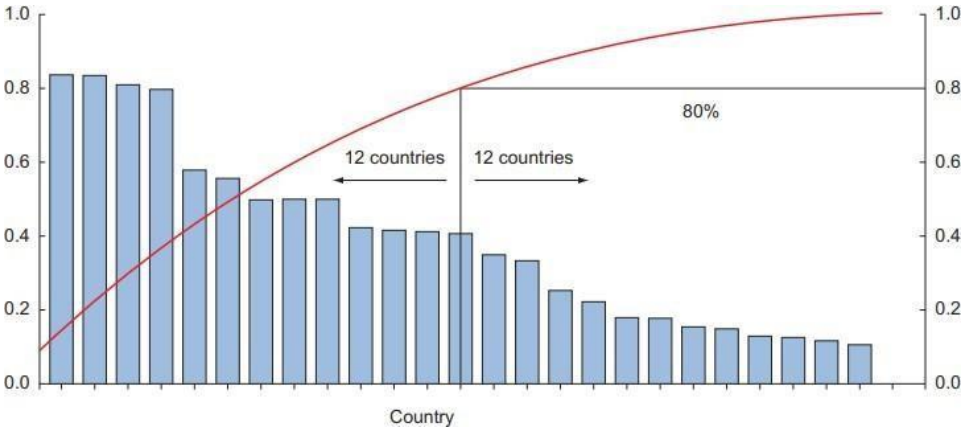






These plots can be combined to provide even more insight, as shown in figure. Overlaying several plots is common practice


Q28_1
Q28_2
Q28_3
Q28_2
Q28_4
Q28_2
Q28_5
Q28_2


80%
12 countries
12 countries
Country

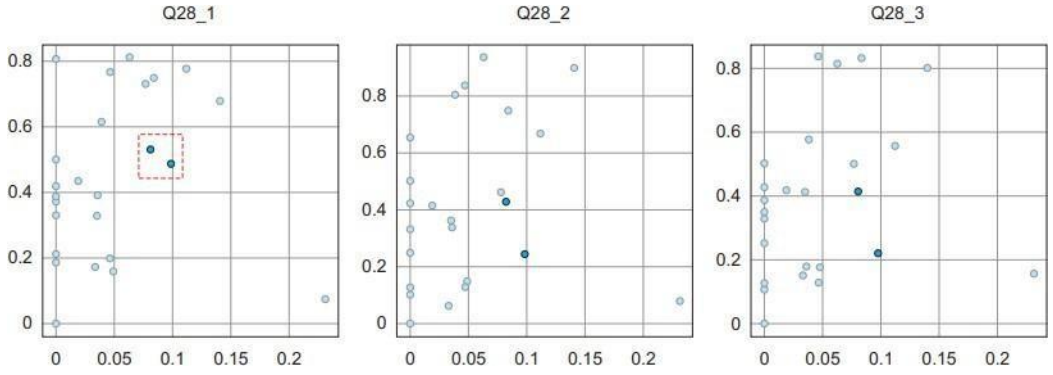

Q28_1
Q28_2
Q28_3

Figure shows another technique: brushing and linking. With brushing and linking we combine and link different graphs and tables (or views) so changes in one graph are automatically transferred to the other graphs. This interactive exploration of data facilitates the discovery of new insights.

Figure shows the average score per country for questions. Not only does this indicate a high correlation between the answers, but it's easy to see that when we select several points on a subplot, the points will correspond to similar points on the other graphs. In this case the selected points on the left graph correspond to points on the middle and right graphs, although they correspond better in the middle and right graphs.

Two other important graphs are the histogram shown in figure and the boxplot shown in figure. In a histogram a variable is cut into discrete categories and the number of occurrences in each category are summed up and shown in the graph. The boxplot, on the other hand, doesn't show how many observations are present but does offer an impression of the distribution within categories. It can show the maximum, minimum, median, and other characterizing measures at the same time

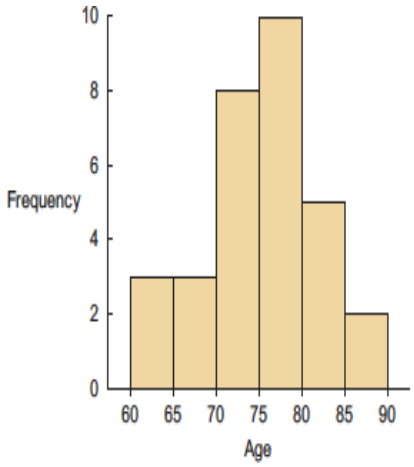


Figure 2.19 Example histogram: the number of people in the age-groups of 5-year intervals

Tabulation, clustering, and other modeling techniques can also be a part of exploratory analysis.

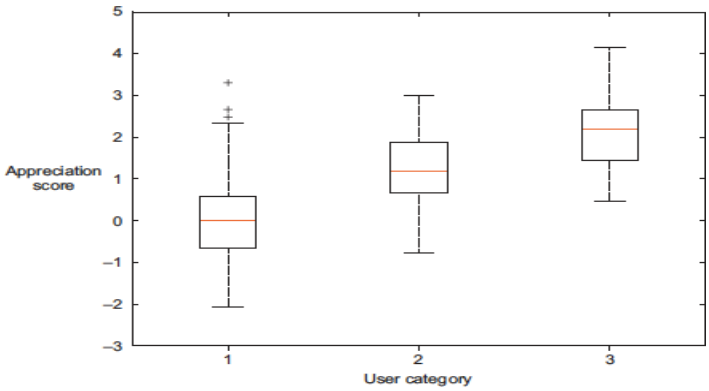


Figure 2.20 Example boxplot: each user category has a distribution of the appreciation each has for a certain picture on a photography website.

1.8 SIGNIFICANCE OF EDA

➢ Different fields of science, economics, engineering, and marketing accumulate and store data primarily in electronic databases. Appropriate and well-established decisions should be made using data collected.

➢ It is practically impossible to make sense of datasets containing more than a handful of data points without the help of computer programs. To make sure of the insights provided by the collected data and to make further decisions, data mining is performed which includes distinct analysis processes.

➢ Exploratory data analysis is the key and first exercise in data mining. It allows us to visualize data to understand it as well as to create hypotheses (ideas) for further analysis. The exploratory analysis centers around creating a synopsis of data or insights for the next steps in a data mining project. EDA actually reveals the ground truth about the content without making any underlying assumptions.

➢ Hence, data scientists use this process to actually understand what type of modeling and hypotheses can be created. Key components of exploratory data analysis include summarizing data, statistical analysis, and visualization of data.

➢ Python provides expert tools for exploratory analysis

- Pandas for summarizing
- Scipy, along with others, for statistical analysis
- Matplotlib and Plotly for visualizations

Steps in EDA

The four different steps involved in exploratory data analysis are,

1. Problem Definition
2. Data Preparation
3. Data Analysis
4. Development and Representation of the Results

1. Problem Definition

• It is essential to define business problem to be solved before trying to extract useful insight from the data.

• The problem definition works as the driving force for a data analysis plan execution. The main tasks involved in problem definition are

o defining the main objective of the analysis
o defining the main deliverables
o outlining the main roles and responsibilities
o obtaining the current status of the data
o defining the timetable, and performing cost/benefit analysis

• Based on the problem definition, an execution plan can be created.

2. Data Preparation

• This step involves methods for preparing the dataset before actual analysis. This step involves

o defining the sources of data
o defining data schemas and tables
o understanding the main characteristics of the data
o cleaning the dataset
o deleting non-relevant datasets
o transforming the data
o dividing the data into required chunks for analysis

3. Data analysis

➢ This is one of the most crucial steps that deals with descriptive statistics and analysis of the data
➢ The main tasks involve
➢ summarizing the data
➢ finding the hidden correlation
➢ relationships among the data
➢ developing predictive models
➢ evaluating the models
➢ calculating the accuracies

Some of the techniques used for data summarization are

- Summary Tables
- Graphs
- Descriptive Statistics
- Inferential Statistics

- Correlation Statistics
- Searching
- Grouping
- Mathematical Models

4. Development and representation of the results

• This step involves presenting the dataset to the target audience in the form of graphs, summary tables, maps, and diagrams.

• This is also an essential step as the result analyzed from the dataset should be interpretable by the business stakeholders, which is one of the major goals of EDA.

• Most of the graphical analysis techniques include

- scattering plots
- character plots
- histograms
- box plots
- residual plots
- mean plots

1.9 MAKING SENSE OF DATA

➢ It is crucial to identify the type of data under analysis. Different disciplines store different kinds of data for different purposes.

➢ Example: medical researchers store patients' data, universities store students' and teachers' data, and real estate industries storehouse and building datasets.

➢ A dataset contains many observations about a particular object. For instance, a dataset about patients in a hospital can contain many observations. A patient can be described by a

- patient identifier (ID)
- name
- address
- weight

➢ Each of these features that describes a patient is a variable.

- date of birth
- address
- email
- gender

PATIENT_ID = 1001
Name = Yoshmi Mukhiya
Address = Mannsverk 61, 5094, Bergen,
Norway Date of birth = 10th July 2018
Email =
yoshmimukhiya@gmail.com
Weight = 10
Gender = Female

These datasets are stored in hospitals and are presented for analysis

.Most of this data is stored in some sort of database management system in tables/schema.

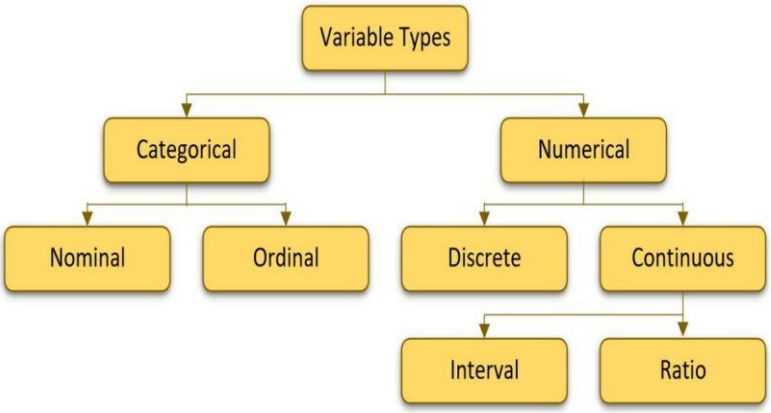
Table for storing patient Information

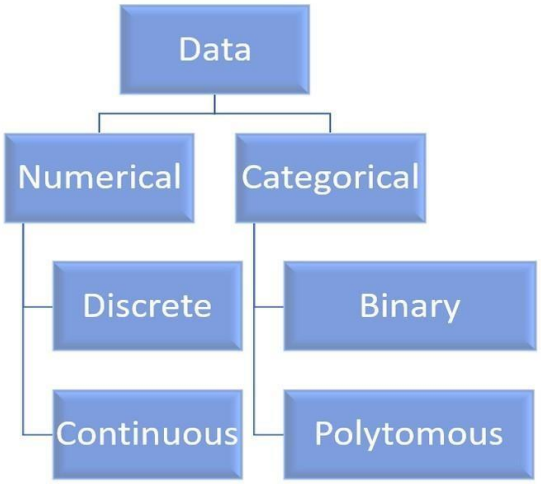
PATIENT_ID	NAME	ADDRESS	DOB	EMAIL	Gender	WEIGHT
001	Suresh Kumar Mukhiya	Mannsverk, 61	30.12.1989	skmu@hvl.no	Male	68
002	Yoshmi Mukhiya	Mannsverk 61, 5094, Bergen	10.07.2018	yoshmimukhiya@gmail.com	Female	1
003	Anju Mukhiya	Mannsverk 61, 5094, Bergen	10.12.1997	anjumukhiya@gmail.com	Female	24
004	Asha Gaire	Butwal, Nepal	30.11.1990	aasha.gaire@gmail.com	Female	23
005	Ola Nordmann	Danmark, Sweden	12.12.1789	ola@gmail.com	Male	75

➢ The table contains five observations (001, 002, 003, 004, 005).
➢ Each observation describes variables (PatientID, name, address, dob, email, gender, and weight).

Types of datasets

➢ Most datasets broadly fall into two groups—Numerical Data and Categorical Data.





Numerical data

➢ This data has a sense of measurement involved in it

➢ For example, a person's age, height, weight, blood pressure, heart rate, temperature, number of teeth, number of bones, and the number of family members.

➢ This data is often referred to as quantitative data in statistics.

➢ The numerical dataset can be either discrete or continuous types.

Discrete data

➢ This is data that is countable and its values can be listed.

➢ For example, if we flip a coin, the number of heads in 200 coin flips can take values from 0 to 200 cases.

➢ A variable that represents a discrete dataset is referred to as a discrete variable.

➢ The discrete variable takes a fixed number of distinct values.

Example:

o The Country variable can have values such as Nepal, India, Norway, and Japan.

o The Rank variable of a student in a classroom can take values from 1, 2, 3, 4, 5, and so on.

Continuous data

➢ A variable that can have an infinite number of numerical values within a specific range is classified as continuous data.

➢ A variable describing continuous data is a continuous variable.

➢ Continuous data can follow an interval measure of scale or ratio measure of scale

Example:

o The temperature of a city

o The weight variable is a continuous variable

Categorical data

➢ This type of data represents the characteristics of an object

➢ Examples: gender, marital status, type of address, or categories of the movies.

➢ This data is often referred to as qualitative datasets in statistics.

Examples of categorical data

o Gender (Male, Female, Other, or Unknown)

o Marital Status (Annulled, Divorced, Interlocutory, Legally Separated, Married, Polygamous, Never Married, Domestic Partner, Unmarried, Widowed, or Unknown)

o Movie genres (Action, Adventure, Comedy, Crime, Drama, Fantasy, Historical, Horror, Mystery, Philosophical, Political, Romance, Saga, Satire, Science Fiction, Social, Thriller, Urban, or Western)

o Blood type (A, B, AB, or O)

o Types of drugs (Stimulants, Depressants, Hallucinogens, Dissociatives, Opioids, Inhalants, or Cannabis)

➢ A variable describing categorical data is referred to as a categorical variable.

➢ These types of variables can have a limited number of values.

Types of categorical variables

Binary categorical variable

➢ This type of variable can take exactly two values

➢ Also referred to as a dichotomous variable.

➢ Example: while creating an experiment, the result is either success or failure.

Polytomous variables

➢ This type can take more than two possible values.

➢ Example: marital status can have several values, such as divorced, legally separated, married, never married, unmarried, widowed, etc.

➢ Most of the categorical dataset follows either nominal or ordinal measurement scales.

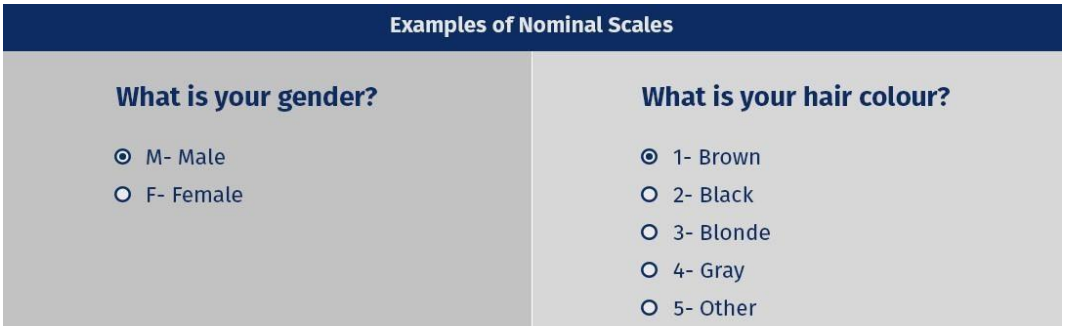
Measurement scales

➢ There are four different types of measurement scales in statistics:

- Nominal,
- Ordinal,
- Interval and
- Ratio.

Nominal Scale:

- A nominal scale associates numbers with variables for naming or labeling the object. It is the most basic of the four levels of measurement. This is a method of measuring the objects or events into a discrete category. It assigns a number to an object only for the identification of the object. So it is a categorical data or qualitative data.
- The numbers are only used for labeling variables and without any quantitative value. The scales are referred to as labels.



- The languages that are spoken in a particular country
- Parts of speech in grammar (noun, pronoun, adjective, and so on)

➢ Nominal scales are considered qualitative scales and the measurements that are taken using qualitative scales are considered qualitative data.

➢ Using numbers as labels have no concrete numerical value or meaning.

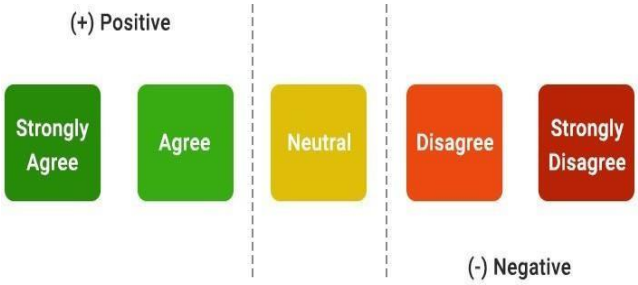
➢ No form of arithmetic calculation can be made on nominal measures.

Ordinal Scale:

- It establishes the rank between the variables of a scale but not the difference value between the variables. The ordinal scale is the next level of data measurement scale. Here the ―Ordinal‖ is the indication of ―Order‖.
- Ordinal measurement assigns a numerical value to the variables based on their relative ranking. In a nominal scale, there is no predefined order for arranging the data.

➢ The main difference in the ordinal and nominal scale is the order.

Example of ordinal scale using the Likert scale:



➢ The answer to the question is scaled down to five different ordinal values, Strongly Agree, Agree, Neutral, Disagree, and Strongly Disagree.

➢ These Scales are referred to as the Likert scale.

More examples of the Likert scale:

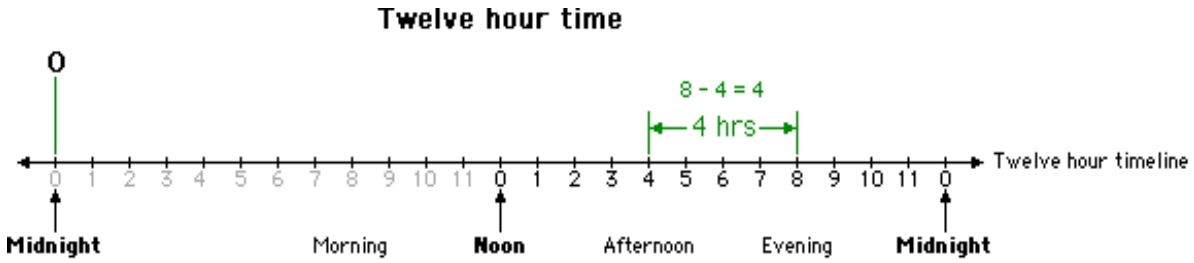
How do you feel today?	How satisfied are you with our service?
● 1 - Very Unhappy	● 1 - Very Unsatisfied
○ 2 - Unhappy	○ 2 - Somewhat Unsatisfied
○ 3 - OK	○ 3 - Neutral
○ 4 - Happy	○ 4 - Somewhat Satisfied
○ 5 - Very Happy	○ 5 - Very Satisfied

Interval Scale (Ranking Scale) :

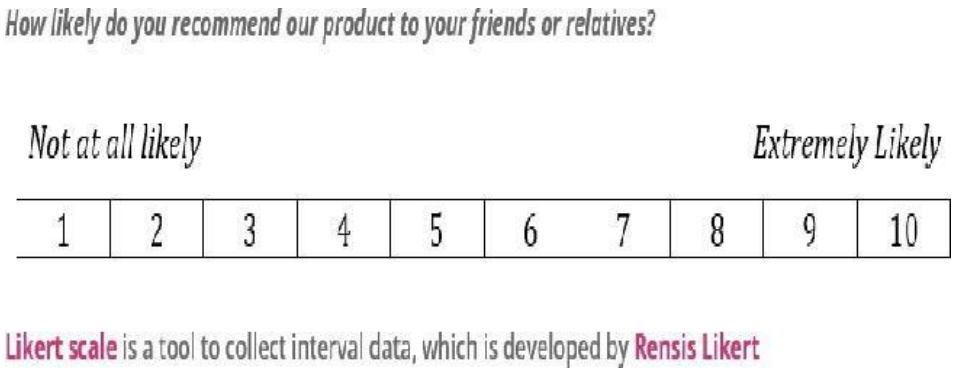
Along with establishing a rank and name of variables, the scale also makes known the difference between the two variables. The only drawback is that there is no fixed start point of the scale, i.e., the actual zero value is absent. The distance between any two adjacent attributes is called an interval, and intervals are always equal.

Example1:

o The measure of central tendencies—mean, median, mode, and standard deviations.



Example2:

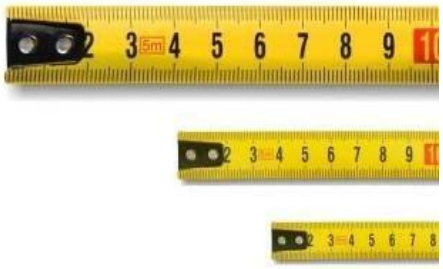


Ratio Scale:

Ratio scale is the most advanced measurement scale, which has variables that are labeled in order and have a calculated difference between variables. This scale has a fixed starting point, i.e., the actual zero

value is present. Ratio scale is purely quantitative. Among the four levels of measurement, ratio scale is the most precise.
Examples of ratio scales are age, wight, height, income, distance etc.

Examples: the measure of energy, mass, length, duration, electrical energy and volume.



Summary of the data types and scale measures:

Provides:	Nominal	Ordinal	Interval	Ratio
The "order"of values is known		✓	✓	✓
"Counts," aka "Frequency of Distribution"	✓	✓	✓	✓
Mode	✓	✓	✓	✓
Median		✓	✓	✓
Mean			✓	✓
Can quantify the difference between each value			✓	✓
Can add or subtract values			✓	✓
Can multiple and divide values				✓
Has "true zero"				✓

1.10 COMPARING EDA WITH CLASSICAL AND BAYESIAN ANALYSIS

Several approaches to data analysis

- ➢ Classical data analysis
- ➢ Exploratory data analysis approach
- ➢ Bayesian data analysis approach

Classical data analysis

➢ This approach includes the problem definition and data collection step followed by model development, which is followed by analysis and result communication.

Exploratory data analysis approach

➢ This approach follows the same approach as classical data analysis except for the model imposition and the data analysis steps are swapped.

➢ The main focus is on the data, its structure, outliers, models, and visualizations.

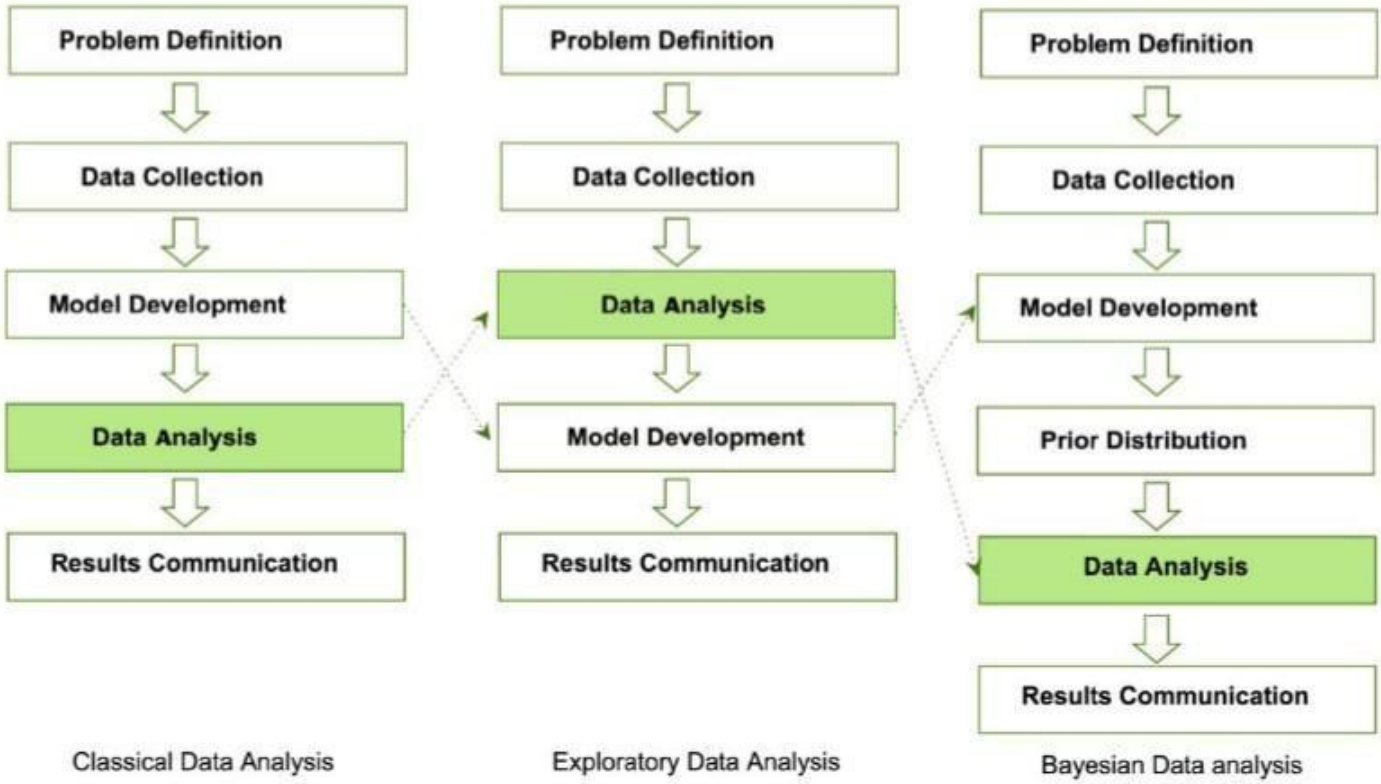
➢ EDA does not impose any deterministic or probabilistic models on the data.

Bayesian data analysis approach

➢ This approach incorporates prior probability distribution knowledge into the analysis steps.

➢ Prior probability distribution of any quantity expresses the belief about that particular quantity before considering some evidence.

Three different approaches for data analysis



1.11 SOFTWARE TOOLS AVAILABLE FOR EDA

➢ Python

• an open-source programming language widely used in data analysis, data mining, and data science

An interpreted, object-oriented programming language for rapid application development. Python and EDA can be used to identify missing values in a data set and to handle missing values for machine learning.

➢ R programming language

• an open-source programming language widely used in statistical computation and graphical data analysis

➢ Weka

• an open-source data mining package that involves several EDA tools and algorithms. It Proides algorithms for Data preprocessing, Classification, Regression, Clustering, Association rules and Visualization.

➢ KNIME

• an open-source tool for data analysis and is based on Eclipse

➢ Excel/Spreadsheet

• It supports important features like summarizing data, visualizing data, data wrangling etc.. Microsoft Excel is paid but there are various other spreadsheet tools like open office, google docs are open source.

➢ Tableau

• Data Visualization software which allows exploring the data using Charts.

Python tools and packages

Python programming	Fundamental concepts of variables, string, and data types Conditionals and functions Sequences, collections, and iterations Working with files Object-oriented programming
NumPy	Create arrays with NumPy, copy arrays, and divide arrays Perform different operations on NumPy arrays Understand array selections, advanced indexing, and expanding Working with multi-dimensional arrays Linear algebraic functions and built-in NumPy functions
pandas	Understand and create `DataFrame` objects Subsetting data and indexing data Arithmetic functions, and mapping with pandas Managing index Building style for visual analysis
Matplotlib	Loading linear datasets Adjusting axes, grids, labels, titles, and legends Saving plots
SciPy	Importing the package Using statistical packages from SciPy Performing descriptive statistics Inference and data analysis

NumPy

➢ NumPy is Numerical Python which is a Python library. NumPy is used for working with arrays.

➢ It also has functions for working in domain of linear algebra, fourier transform, and matrices.

Pandas

➢ Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data.

SciPy

➢ SciPy stands for Scientific Python which is a scientific computation library that uses NumPy.

➢ It provides more utility functions for optimization, stats and signal processing. Like NumPy, SciPy is open source so we can use it freely. SciPy has optimized and added functions that are frequently used in NumPy and Data Science.

Matplotlib

➢ Matplotlib is a low-level graph plotting library in python that serves as a visualization utility.

➢ It provides a huge library of customizable plots used to create professional reporting applications, interactive analytical applications, complex dashboard applications, web/GUI applications etc.,

1.12 VISUAL AIDS FOR EDA

Data scientists extract knowledge fromthe data and to present the data to stakeholders. Visual aids are very useful tools Presenting results to stakeholders. Different types of techniques that can be used in the visualization of data.

- Univariate analysis (Used for univariate data - the data containing one variable)
  Univariate plots show the frequency or the distribution shape of a variable. Below ar visual tools used to

analyze univariate data. Histograms are two-dimensional plots in which the x-axis divides into a range o numerical bins or time intervals. The y-axis shows the frequency values, which are counts of occurrences of values for each bin

- Bar chart
- Scatterplot
- Histograms
- Line Chart
- Pie chart
- Box Plot
- Stacked area plot (Line Chart+Bar Chart)
- Table chart
- Probability Distribution Plots

Example:

Heights (in cm)	164	167.3	170	174.2	178	180	186

- Bivariate Plots (Used for bivariate data - the data containing two variables)

- Bar Chart
- Scatter Plot
- Box Plot
- Polar chart
- Lollipop chart
- Pie chart
- Density Plot
- Contour Plot

Example:

TEMPERATURE(IN CELSIUS)	ICE CREAM SALES
20	2000
25	2500
35	5000
43	7800

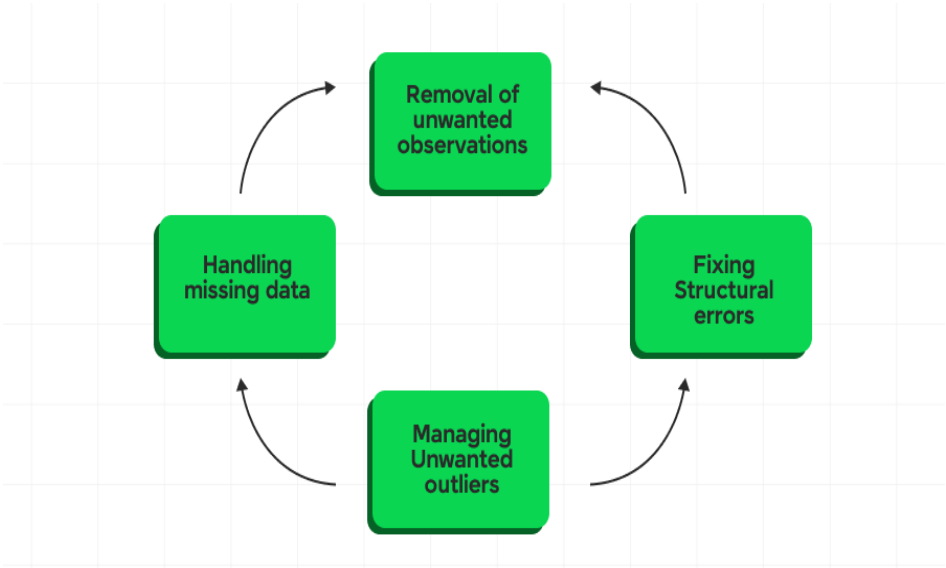
- Multivariate Plots (Used for Multivariate data - the data containing more than two variables) Multivariate data contains high dimensionality data and has applications in deep learning such as visualizing natural language such as text or images.
  - Scatter Plot
  - PCA-Principal Component analysis
    Heatmap
    Example

Household ID	Household Size	Annual Income	Number of Pets
1	2	$37,000	0
2	4	$49,000	0
3	4	$58,000	1
4	1	$68,000	3
5	3	$61,000	2
6	5	$64,000	2
7	6	$79,000	1
8	4	$89,000	1
9	7	$104,000	1
10	2	$95,000	0

Univariate	Bivariate	Multivariate
It only summarize single variable at a time.	It only summarize two variables	It only summarize more than 2 variables.
It does not deal with causes and relationships.	It does deal with causes and relationships and analysis is done.	It does not deal with causes and relationships and analysis is done.
It does not contain any dependent variable.	It does contain only one dependent variable.	It is similar to bivariate but it contains more than 2 variables.
The main purpose is to describe.	The main purpose is to explain.	The main purpose is to study the relationship among them.
The example of a univariate can be height.	The example of bivariate can be temperature and ice sales in summer vacation.	correlation between the amount of time spent on social media and an employee's productivity

1.13 DATA CLEANING

Data cleaning is the process of identifying and correcting or removing errors, inconsistencies, and inaccuracies **in datasets**. It involves:



- Removing duplicate or irrelevant observations
- Fixing structural errors
- Handling missing data
- Filtering out outliers
- Standardizing data formats
- Correcting typos or formatting issues
- This process is important for ensuring data quality and reliability in analysis and decision-making.

Step-by-Step Data Cleaning Process

While every dataset is unique and may require specific approaches, following a structured process can help ensure thorough and effective data cleaning. Here's a step-by-step guide to data cleaning:

Step 1: Understand the Data
Before getting into cleaning, it's important to understand your data:

- Review the data dictionary or metadata if available.
- Examine the structure of the dataset (rows, columns, data types).
- Understand the context and source of the data.
- Identify the key variables for your analysis.

Step 2: Make a Copy of the Raw Data
Always preserve the original dataset:

- Create a working copy of the raw data.
- This allows you to revert changes if needed and maintains data provenance.

Step 3: Perform Initial Data Exploration
Get a feel for the dataset:

- Use descriptive statistics (mean, median, standard deviation, etc.).
- Visualize distributions with histograms, box plots, or scatter plots.
- Check for obvious anomalies or patterns.

Step 4: Check Data Types and Structures
Ensure data is in the correct format:

- Verify that each column has the appropriate data type (numeric, categorical, datetime, etc.).
- Check if date formats are consistent.
- Identify any columns that may need to be split or combined.

Step 5: Handle Missing Data
Address null or empty values:

- Quantify the amount of missing data for each variable.
- Determine if data is missing completely at random (MCAR), missing at random (MAR), or missing not at random (MNAR).
- Decide on an appropriate strategy (deletion, imputation, etc.) based on the nature of the missingness and its potential impact on your analysis.

Step 6: Remove Duplicates
Identify and handle duplicate entries:

- Determine what constitutes a duplicate in your dataset.

- Use appropriate methods to identify duplicates (exact or fuzzy matching).
- Decide whether to remove duplicates or flag them for further investigation.

Step 7: Handle Outliers

Identify and address extreme values:

- Use statistical methods or visualization to detect outliers.
- Investigate the cause of outliers (data entry errors, genuine anomalies, etc.).
- Decide on an appropriate treatment (removal, capping, transformation, etc.) based on the nature of the outlier and its relevance to your analysis.

Step 8: Standardize and Normalize

Ensure consistency across the dataset:

- Standardize units of measurement.
- Normalize text data (e.g., consistent capitalization, handling of special characters).
- Encode categorical variables consistently.

Step 9: Correct Invalid Values

Address values that don't make logical sense:

- Check for values outside of possible ranges (e.g., negative ages, future dates).
- Correct obvious data entry errors.
- Flag or remove values that violate business rules or logical constraints.

Step 10: Handle Structural Errors

Address issues with the dataset's structure:

- Reshape data if necessary (e.g., from wide to long format or vice versa).
- Split or combine columns as needed.
- Ensure consistent naming conventions for variables.

Step 11: Validate and Cross-Check

Ensure the cleaning process hasn't introduced new errors:

- Cross-validate cleaned data against the original dataset.
- Check if summary statistics make sense after cleaning.
- Verify that relationships between variables are preserved or explainable.

Step 12: Handle Special Cases

Address any domain-specific issues:

- Apply any specific business rules or constraints.
- Handle industry-specific data quirks (e.g., financial data rounding, scientific notation).

Step 13: Document the Cleaning Process

Maintain a clear record of all cleaning steps:

- Document each transformation applied to the data.
- Note any assumptions made during the cleaning process.
- Record the rationale behind significant cleaning decisions.

Step 14: Create Automated Cleaning Scripts

If possible, automate the cleaning process:

- Write scripts or use workflow tools to automate repetitive cleaning tasks.
- This ensures reproducibility and makes it easier to apply the same cleaning process to future datasets.

Step 15: Perform a Final Review

Take a step back and review the entire cleaned dataset:

- Check if the cleaned data meets the requirements for your intended analysis.
- Verify that all identified issues have been addressed.
- Ensure the cleaning process hasn't introduced bias or significantly altered the fundamental characteristics of the dataset.

By following these steps, you can approach data cleaning in a systematic and thorough manner.

Data Cleaning in Data Science: Ensuring Quality for Meaningful Insights

Let's learn data cleaning, and explore common issues, techniques, tools, best practices, and challenges. Read Data Science vs Data Analytics if you're deciding between these two dynamic fields and want to make an informed career choice.

Common Data Issues

Before we explore the solutions, it's important to understand the problems. Here are some of the most common data issues that necessitate cleaning:

a) **Missing Data**: One of the most prevalent issues in datasets is missing values. This can occur due to data entry errors, system malfunctions, or simply because the information wasn't available at the time of collection. Missing data can significantly skew analyses and lead to incorrect conclusions if not handled properly.

b) **Duplicate Data**: Duplicate records can inflate dataset size and lead to biased analysis results. They often occur due to manual data entry errors, system glitches, or when merging data from multiple sources.

c) **Inconsistent Formatting**: Inconsistencies in data format can make analysis challenging. For example, dates might be recorded in different formats (MM/DD/YYYY vs. DD/MM/YYYY), or names might be entered with varying capitalization or order (John Doe vs. Doe, John).

d) **Typos and Spelling Errors**: Human error in data entry can lead to typos and misspellings. While seemingly minor, these errors can cause significant problems, especially when dealing with categorical data or when using the data for text analysis.

e) **Outliers**: Outliers are data points that significantly differ from other observations. While sometimes outliers represent genuine anomalies that require investigation, they can also be the result of measurement or data entry errors.

f) **Inconsistent Units**: When working with numerical data, inconsistent units can lead to severe misinterpretations. For instance, mixing metric and imperial measurements or using different currencies without proper conversion.

g) **Structural Errors**: These occur when there are issues with data labeling, categorization, or the overall structure of the dataset. For example, having inconsistent category names or mislabeled variables.

h) **Encoded or Garbled Data**: Sometimes data can become corrupted during transfer or storage, resulting in unreadable or nonsensical values. This is particularly common with special characters or when dealing with different character encodings.

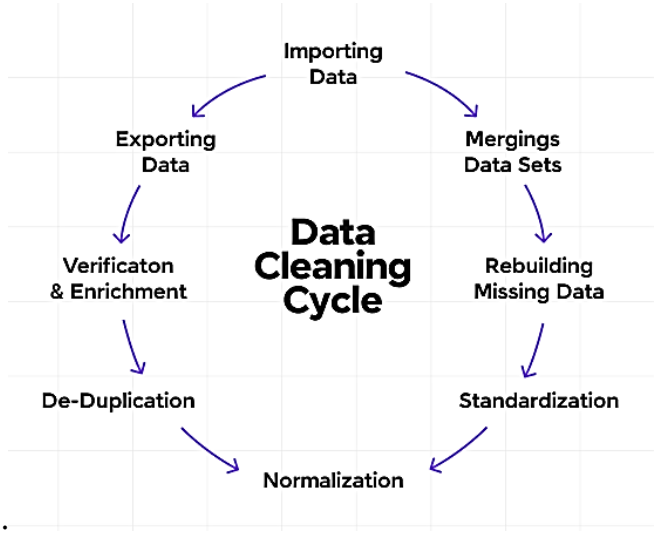
i) **Inconsistent Naming Conventions**: Variables or categories might be named inconsistently across different parts of a dataset or across multiple datasets that need to be merged.

j) **Data Type Mismatches**: This occurs when data is stored in an inappropriate format. For example, storing numerical data as text, which can prevent mathematical operations.

k) **Truncated Data**: Sometimes data can be cut off or truncated due to system limitations or data transfer issues. This is particularly problematic with long text fields or large numerical values.

l) **Unnecessary Metadata**: Datasets might include extraneous information that isn't relevant to the analysis at hand, such as automatically-generated timestamps or system-specific identifiers.

m) **Inconsistent Aggregation**: When working with data that has been aggregated or summarized, there might be inconsistencies in how the aggregation was performed across different subsets of the data.



.

Data Cleaning Techniques and Tools

Once you've identified the issues in your dataset, the next step is to apply appropriate cleaning techniques. Here's an overview of common data cleaning techniques and some popular tools used in the process:

a) Handling Missing Data

- **Deletion**: Remove rows or columns with missing data. This is simple but can lead to significant data loss.
- **Imputation**: Fill in missing values using statistical methods. Common approaches include:
  - Mean/Median/Mode imputation
  - Regression imputation
  - Multiple imputation
- **Using a dedicated category**: For categorical data, treating ―missing‖ as its own category.
- **Advanced techniques**: Using machine learning algorithms like K-Nearest Neighbors or Random Forests for prediction-based imputation. Read Data Science vs. Machine Learning to master the key differences and elevate your coding skills.

b) Dealing with Duplicates

- **Exact matching**: Identify and remove identical rows.
- **Fuzzy matching**: Use algorithms to identify near-duplicate entries, accounting for minor differences in spelling or formatting.
- **Composite key matching**: Create a unique identifier using multiple columns to identify duplicates.

c) Standardizing Formats

- **Regular expressions**: Use regex patterns to identify and correct inconsistent formatting.
- **Parsing libraries**: Utilize specialized libraries for parsing dates, addresses, or other structured data.
- **Lookup tables**: Create standardized mappings for common variations of the same data.

d) Correcting Typos and Misspellings

- **Spell-checking algorithms**: Use built-in or custom dictionaries to identify and correct misspellings.
- **Fuzzy string matching**: Employ algorithms like Levenshtein distance or Soundex to match similar strings.
- **Manual correction**: For critical or high-impact data, manual review and correction may be necessary.

e) Handling Outliers

- **Statistical methods**: Use techniques like z-score or Interquartile Range (IQR) to identify outliers.
- **Visualization**: Employ scatter plots, box plots, or histograms to visually identify outliers.
- **Domain expertise**: Consult subject matter experts to determine if outliers are genuine anomalies or errors.
- **Winsorization**: Cap extreme values at a specified percentile of the data.

Popular Tools for Data Cleaning

1. Python Libraries
   - **Pandas**: Offers a wide range of data manipulation and cleaning functions.
   - **NumPy**: Useful for numerical operations and handling missing data.
   - **Scikit-learn**: Provides imputation methods and other preprocessing techniques.
   - **Fuzzywuzzy**: Specializes in fuzzy string matching.
2. **R Packages**
   - **tidyr**: Part of the tidyverse, focused on tidying data.
   - **dplyr**: Offers a grammar for data manipulation.
   - **stringr**: Provides tools for string manipulation.
   - **mice**: Specializes in multiple imputations for missing data.
3. **SQL**
   - While primarily a query language, SQL can be used for various data cleaning tasks, especially when working with large datasets in databases.
4. **OpenRefine**
   - A powerful tool for working with messy data, offering features like clustering for cleaning text data.
5. **Trifacta Wrangler**
   - Provides a user-friendly interface for data cleaning and transformation tasks.
6. **Talend Data Preparation**
   - Offers both open-source and enterprise versions for data cleaning and preparation.
7. **Excel**
   - While not suitable for large datasets, Excel's built-in functions and Power Query can be useful for smaller-scale data cleaning tasks.
8. **Databricks**
   - A unified analytics platform that integrates with various big data tools, useful for cleaning large-scale datasets.
9. Google Cloud Dataprep
   - A cloud-based service for visually exploring, cleaning, and preparing data for analysis.
10. **Alteryx**
    - Provides a workflow-based approach to data cleaning and preparation, with a strong focus on spatial data.

1.14 BASIC STATISTICAL DESCRIPTIONS OF DATA

- For data preprocessing to be successful, it is essential to have an overall picture of our data. Basic statistical descriptions can be used to identify properties of the data and highlight which data values should be treated as noise or outliers.
- Basic statistical descriptions can be used to identify properties of the data and highlight which data values should be treated as noise or outliers.
- For data preprocessing tasks, we want to learn about data characteristics regarding both central tendency and dispersion of the data.
- Measures of central tendency include mean, median, mode and midrange.
- Measures of data dispersion include quartiles, interquartile range (IQR) and variance.

• These descriptive statistics are of great help in understanding the distribution of the data.

Measuring the Central Tendency

• We look at various ways to measure the central tendency of data, include: Mean, Weighted mean, Trimmed mean, Median, Mode and Midrange.

1. Mean :

• The mean of a data set is the average of all the data values. The sample mean x is the point estimator of the population mean μ.

$$\text{Sample mean } \bar{x} = \frac{\text{Sum of the values of then observations}}{\text{Number of observations in the sample}} = \frac{\sum x_i}{n}$$

$$\text{Population mean } \mu = \frac{\text{Sum of the values of the N observations}}{\text{Number of observations in the population}} = \frac{\sum x_i}{n}$$

2. Median :

Sum of the values of then observations Number of observations in the sample

Sum of the values of the N observations Number of observations in the population

• The median of a data set is the value in the middle when the data items are arranged in ascending order. Whenever a data set has extreme values, the median is the preferred measure of central location.

• The median is the measure of location most often reported for annual income and property value data. A few extremely large incomes of property values can inflate the mean.

• For an off number of observations:

7 observations= 26, 18, 27, 12, 14, 29, 19.

Numbers in ascending order = 12, 14, 18, 19, 26, 27, 29

• The median is the middle value.

Median=19

• For an even number of observations :

8 observations = 26 18 29 12 14 27 30 19

Numbers in ascending order =12, 14, 18, 19, 26, 27, 29, 30

The median is the average of the middle two values.

3. Mode:

• The mode of a data set is the value that occurs with greatest frequency. The greatest frequency can occur at two or more different values. If the data have exactly two modes, the data have exactly two modes, the data are bimodal. If the data have more than two modes, the data are multimodal.

• **Weighted mean**: Sometimes, each value in a set may be associated with a weight, the weights reflect the significance, importance or occurrence frequency attached to their respective values.

• **Trimmed mean**: A major problem with the mean is its sensitivity to extreme (e.g., outlier) values. Even a small number of extreme values can corrupt the mean. The trimmed mean is the mean obtained after cutting off values at the high and low extremes.

• For example, we can sort the values and remove the top and bottom 2 % before computing the mean. We should avoid trimming too large a portion (such as 20 %) at both ends as this can result in the loss of valuable information.

• **Holistic measure** is a measure that must be computed on the entire data set as a whole. It cannot be computed by partitioning the given data into subsets and merging the values obtained for the measure in each subset.

Measuring the Dispersion of Data

• An **outlier** is an observation that lies an abnormal distance from other values in a random sample from a population.

- **First quartile (Q_1):** The first quartile is the value, where 25% of the values are smaller than Q_1 and 75% are larger.
- **Third quartile (Q_3):** The third quartile is the value, where 75 % of the values are smaller than Q_3 and 25% are larger.
- The **box plot** is a useful graphical display for describing the behavior of the data in the middle as well as at the ends of the distributions. The box plot uses the median and the lower and upper quartiles. If the lower quartile is Q_1 and the upper quartile is Q_3, then the difference (Q_3 - Q_1) is called the interquartile range or IQ.
- **Range**: Difference between highest and lowest observed values

Variance :

- The variance is a measure of variability that utilizes all the data. It is based on the difference between the value of each observation (x;) and the mean (x) for a sample, u for a population).
- The variance is the average of the squared between each data value and the mean.

$$\text{Sample variance} : S^2 = \frac{\sum (x_i - \bar{x})^2}{n-1}$$

$$\text{Population variance} : \sigma^2 = \frac{\sum (x_i - \mu)^2}{N}$$

Standard Deviation :

- The standard deviation of a data set is the positive square root of the variance. It is measured in the same in the same units as the data, making it more easily interpreted than the variance.
- The standard deviation is computed as follows:

$$\text{Population standard deviation} = \sigma = \sqrt{\sigma^2} = \sqrt{\frac{\sum (x - \mu)^2}{N}}$$

$$\text{Sample standard deviation} = S = \sqrt{S^2} = \sqrt{\frac{\sum (x - \bar{x})^2}{n-1}}$$

Difference between Standard Deviation and Variance

Sr. No.	Standard Deviation	Variance
1.	Standard deviation is a measure of dispersion of the values of a data set from their mean.	It is the statistical measure of how far the numbers are spread in a data set from their average.
2.	It is a common term in statistical theory to calculate central tendency	Variance is primarily used for statistical probability distribution to measure volatility from the mean.
3.	It measures the absolute variability of the dispersion.	It helps determine the size of the data spread.
4.	It is calculated by taking the square root of the variance.	It is calculated by taking the average of the squared deviation of each value in the data set from the mean.
5.	The standard deviation is symbolized by the Greek letter sigma "σ" as in lower case sigma.	The notation for the variance of a variable is "σ^2" sigma squared.
6.	$\sigma = \sqrt{\sum}(x - M)^2/n$ where M = mean, x = a values in a data set and n = number of values.	$\sigma^2 = \sum (x - M)^2/n$ where M = mean, x = each value in the data set, n = number of values in the data set.
7.	Used in finance sector as a measure of market and security volatility.	Used in asset allocation.

Graphic Displays of Basic Statistical Descriptions

• There are many types of graphs for the display of data summaries and distributions, such as Bar charts, Pie charts, Line graphs, Boxplot, Histograms, Quantile plots and Scatter plots.

1. Scatter diagram

• Also called scatter plot, X-Y graph.

• While working with statistical data it is often observed that there are connections between sets of data. For example the mass and height of persons are related, the taller the person the greater his/her mass.

• To find out whether or not two sets of data are connected scatter diagrams can be used. Scatter diagram shows the relationship between children's age and height.

• A scatter diagram is a tool for analyzing relationship between two variables. One variable is plotted on the horizontal axis and the other is plotted on the vertical axis.

- The pattern of their intersecting points can graphically show relationship patterns. Commonly a scatter diagram is used to prove or disprove cause-and-effect relationships.
- While scatter diagram shows relationships, it does not by itself prove that one variable causes other. In addition to showing possible cause and effect relationships, a scatter diagram can show that two variables are from a common cause that is unknown or that one variable can be used as a surrogate for the other.

2. Histogram

- A histogram is used to summarize discrete or continuous data. In a histogram, the data are grouped into ranges (e.g. 10-19, 20-29) and then plotted as connected bars. Each bar represents a range of data.
- To construct a histogram from a continuous variable you first need to split the data into intervals, called bins. Each bin contains the number of occurrences of scores in the data set that are contained within that bin.
- The width of each bar is proportional to the width of each category and the height is proportional to the frequency or percentage of that category.

3. Line graphs

- It is also called stick graphs. It gives relationships between variables.
- Line graphs are usually used to show time series data that is how one or more variables vary over a continuous period of time. They can also be used to compare two different variables over time.
- Typical examples of the types of data that can be presented using line graphs are monthly rainfall and annual unemployment rates.
- Line graphs are particularly useful for identifying patterns and trends in the data such as seasonal effects, large changes and turning points. Fig. 1.12.1 show line graph.

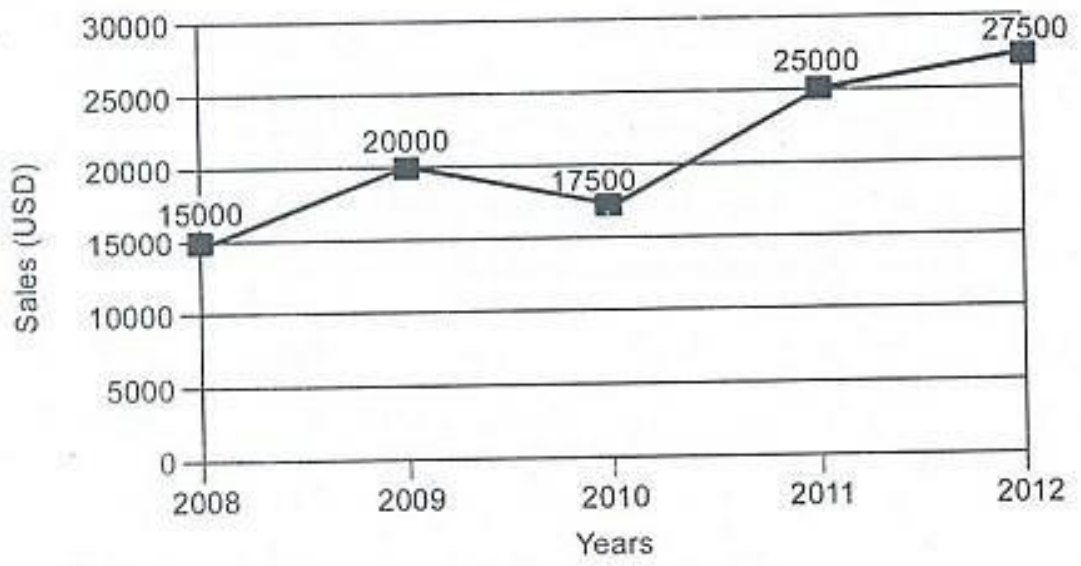


Fig. 1.12.1 : Line graph

As well as time series data, line graphs can also be appropriate for displaying data that are measured over other continuous variables such as distance.

- For example, a line graph could be used to show how pollution levels vary with increasing distance from a source or how the level of a chemical varies with depth of soil.
- In a line graph the x-axis represents the continuous variable (for example year or distance from the initial measurement) whilst the y-axis has a scale and indicated the measurement.

- Several data series can be plotted on the same line chart and this is particularly useful for analysing and comparing the trends in different datasets.
- Line graph is often used to visualize rate of change of a quantity. It is more useful when the given data has peaks and valleys. Line graphs are very simple to draw and quite convenient to interpret.

4. Pie charts

- A type of graph is which a circle is divided into sectors that each represents a proportion of whole. Each sector shows the relative size of each value.

- A pie chart displays data, information and statistics in an easy to read "pie slice" format with varying slice sizes telling how much of one data element exists.
- Pie chart is also known as circle graph. The bigger the slice, the more of that particular data was gathered. The main use of a pie chart is to show comparisons. Fig. 1.12.2 shows pie chart.

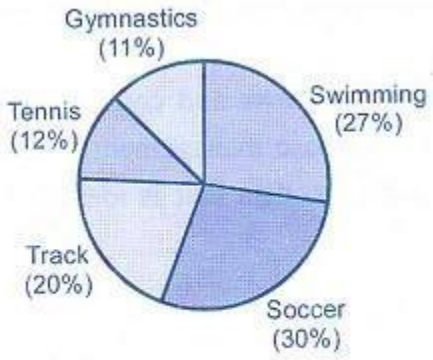


Fig. 1.12.2 : Pie chart

- Various applications of pie charts can be found in business, school and at home. For business pie charts can be used to show the success or failure of certain products or services.
- At school, pie chart applications include showing how much time is allotted to each subject. At home pie charts can be useful to see expenditure of monthly income in different needs.
- Reading of pie chart is as easy figuring out which slice of an actual pie is the biggest.

Limitation of pie chart:

- It is difficult to tell the difference between estimates of similar size.

UNIT II

DESCRIBING DATA

Types of Data - Types of Variables -Describing Data with Tables and Graphs –Describing Data with Averages - Frequency distributions – Outliers –Interpreting Distributions – graphs – averages - Describing Variability – interquartile range – variability for qualitative and ranked data - Normal Distributions and Standard (z) Scores

Statistics

Statistics **is the** process of collecting data, evaluating data, and summarizing it into a mathematical form.

There are two methods of analyzing data in mathematical statistics that are used on a large scale:

- Descriptive Statistics
- Inferential Statistics

Descriptive Statistics

The descriptive method of statistics is used to **describe the data collected and summarize the data** and its **properties** using the **measures of central tendencies and the measures of dispersion.**

Example

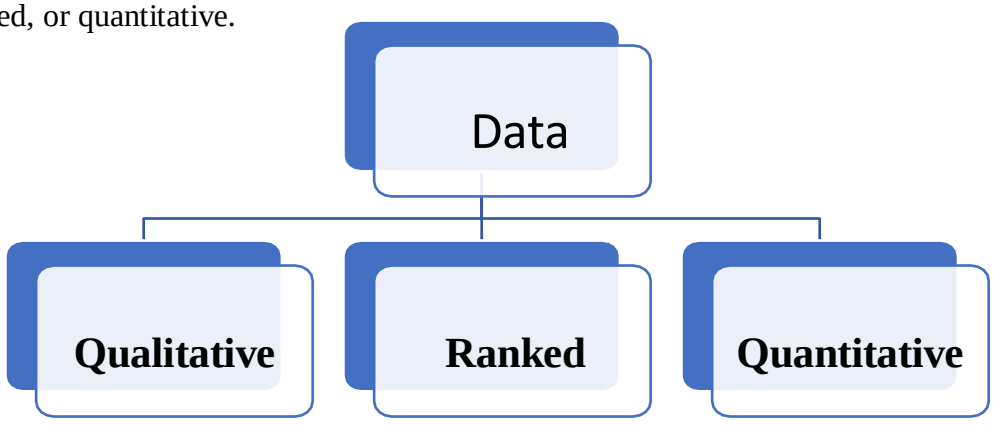
- ❖ A graph showing the annual change in global temperature during the last 30 years
- ❖ A report that describes the average difference in grade point average (GPA) between college students who regularly drink alcoholic beverages and those who don't.

Inferential Statistics

- ❖ Inferential statistics can be defined as a field of statistics that uses analytical tools for **drawing conclusions about a population** by examining random samples.
- ❖ The goal of inferential statistics is to make generalizations about a population. In inferential statistics, a statistic is taken from the sample data (e.g., the sample mean) that used to make inferences about the population parameter (e.g., the population mean).
- ❖ Statistics also provides tools—a variety of tests and estimates—for generalizing beyond collections of actual observations. This more advanced area is known as inferential statistics.

2.1 THREE TYPES OF DATA

- Data is a collection of actual observations or scores in a survey or an experiment.
- The precise form of statistical analysis often depends on whether data are qualitative, ranked, or quantitative.



- **Qualitative data** consist of words (Yes or No), letters (Y or N), or numerical codes (0 or 1) that represent a class or category.

Table 1.2
QUALITATIVE DATA: "DO YOU HAVE A FACEBOOK PROFILE?" YES (Y) OR NO (N) REPLIES OF STATISTICS STUDENTS

Y	Y	Y	N	N	Y	Y	Y
Y	Y	Y	N	N	Y	Y	Y
N	Y	N	Y	Y	Y	Y	Y
Y	Y	N	Y	N	Y	N	Y
Y	N	Y	N	N	Y	Y	Y
Y	Y	N	Y	Y	Y	Y	Y
N	N	N	N	Y	N	N	Y
Y	Y	Y	Y	Y	N	Y	N
Y	Y	Y	Y	N	N	Y	Y
N	Y	N	N	Y	Y	Y	Y
	Y	Y	N				

- **Ranked data** consist of numbers (1st, 2nd, . . . 40th place) that represent relative standing within a group.
- **Quantitative data** consist of numbers (weights of 238, 170, . . . 185 lbs) that represent an amount or a count. To determine the type of data, focus on a single observation in any collection of observations

Table 1.1
QUANTITATIVE DATA: WEIGHTS (IN POUNDS) OF MALE STATISTICS STUDENTS

160	168	133	170	150	165	158	165
193	169	245	160	152	190	179	157
226	160	170	180	150	156	190	156
157	163	152	158	225	135	165	135
180	172	160	170	145	185	152	
205	151	220	166	152	159	156	
165	157	190	206	172	175	154	

2.2 TYPES OF VARIABLES

A **variable** is a characteristic or property that can take on different values.

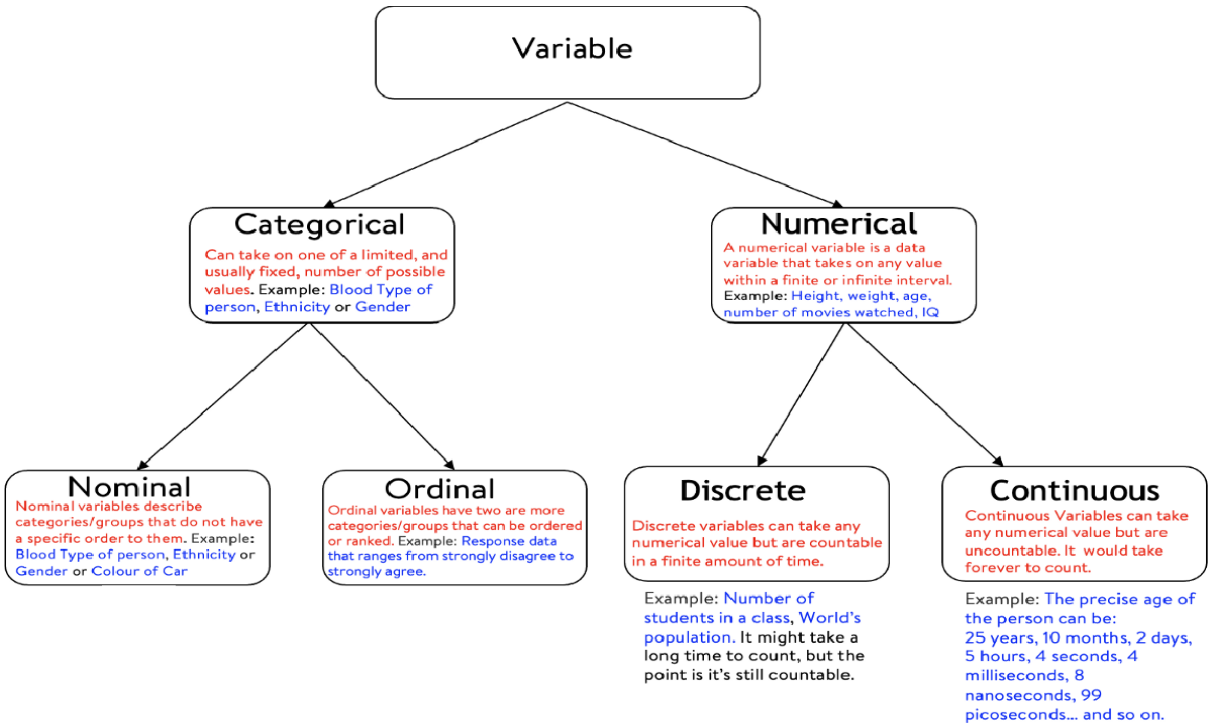
- The weights can be described not only as quantitative data but also as observations for a quantitative variable, since the various weights take on different numerical values.
- By the same token, the replies can be described as observations for a qualitative variable, since the replies to the Facebook profile question take on different values of either Yes or No.
- Given this perspective, any single observation can be described as a constant, since it takes on only one value.

Discrete and Continuous Variables

Quantitative variables can be further distinguished as **discrete or continuous.**

A **discrete variable** consists of isolated numbers separated by gaps.

Discrete variables can only assume specific values that you cannot subdivide. Typically, you count discrete values, and the results are integers.



Examples

- Counts- such as the number of children in a family. (1, 2, 3, etc., but never 1.5)
- These variables cannot have fractional or decimal values. You can have 20 or 21 cats, but not 20.5
- The number of heads in a sequence of coin tosses.
- The result of rolling a die.
- The number of patients in a hospital.
- The population of a country.

While discrete variables have no decimal places, the average of these values can be fractional. For example, families can have only a discrete number of children: 1, 2, 3, etc. However, the average number of children per family can be 2.2.

A **continuous variable** consists of numbers whose values, at least in theory, have no restrictions.

Continuous variables can assume any numeric value and can be meaningfully split into smaller parts. Consequently, they have valid fractional and decimal values. In fact, continuous variables have an infinite number of potential values between any two points. Generally, you measure them using a scale.

Examples of continuous variables include weight, height, length, time, and temperature.
Durations, such as the reaction times of grade school children to a fire alarm; and standardized test scores, such as those on the Scholastic Aptitude Test (SAT).

Independent and Dependent Variables Independent Variable
In an experiment, an **independent variable** *is the treatment manipulated by the investigator.*

- Independent variables (IVs) are the ones that you include in the model to explain or predict changes in the dependent variable.
- Independent indicates that they stand alone and other variables in the model do not influence them.
- Independent variables are also known as predictors, factors, treatment variables, explanatory variables, input variables, x-variables, and right-hand variables—because they appear on the right side of the equals sign in a regression equation.

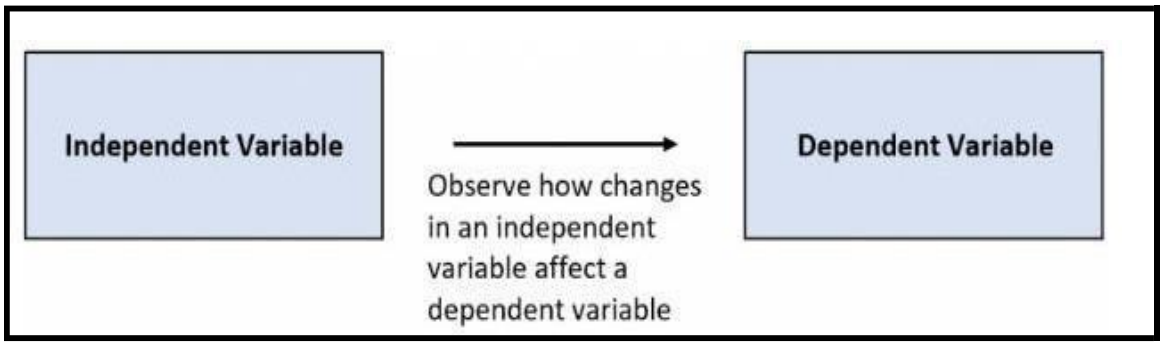
- It is a variable that stands alone and isn't changed by the other variables you are trying to measure.

For example, someone's age might be an independent variable. Other factors (such as what they eat, how much they go to school, how much television they watch)

The impartial creation of distinct groups, which differ only in terms of the independent variable, has a most desirable consequence. Once the data have been collected, any difference between the groups can be interpreted as being *caused* by the independent variable.

Dependent Variable

When a variable is believed to have been influenced by the independent variable, it is called a **dependent variable.** In an experimental setting, the dependent variable is measured, counted, or recorded by the investigator.

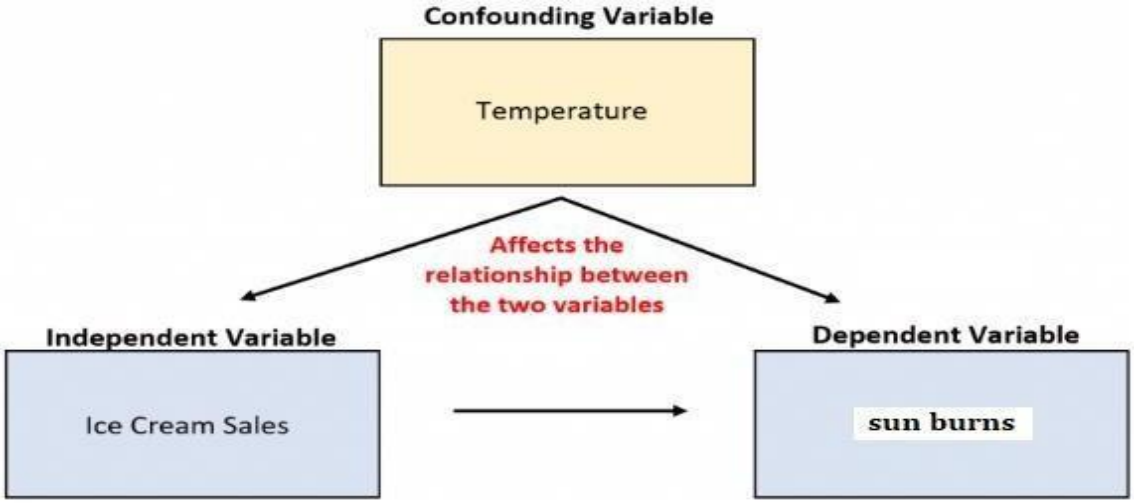


- The dependent variable (DV) is what you want to use the model to explain or predict. The values of this variable depend on other variables.
- It's also known as the response variable, outcome variable, and left-hand variable. Graphs place dependent variables on the vertical, or Y, axis.
- A dependent variable is exactly what it sounds like. It is something that depends on other factors.

For example the blood sugar test depends on what food you ate, at which time ate etc.
Unlike the independent variable, the dependent variable isn't manipulated by the investigator. Instead, it represents an outcome: the data produced by the experiment.

Confounding Variable

An uncontrolled variable that compromises the interpretation of a study is known as a **confounding variable.** Sometimes a confounding variable occurs because it's impossible to assign subjects randomly to different conditions.



2.3 DESCRIBING DATA WITH TABLES AND GRAPHS

TABLES (FREQUENCY DISTRIBUTIONS)

- A frequency distribution is a collection of observations produced by sorting observations into classes and showing their frequency (f) of occurrence in each class.
- To organize the weights of the male statistics students listed in Table 1.1. First, arrange a column of consecutive numbers, beginning with the lightest weight
  (133) at the bottom and ending with the heaviest weight (245) at the top.
- A short vertical stroke or tally next to a number each time its value appears in the original set of data; once this process has been completed, substitute for each tally count a number indicating the frequency (f) of occurrence of each weight.
- When observations are sorted into classes of single values, as in Table 2.1, the result is referred to as a frequency distribution for ungrouped data.
- The frequency distribution shown in Table 2.1 is only partially displayed becausethere are more than 100 possible values between the largest and smallest observations.

Table 2.1 FREQUENCY DISTRIBUTION (UNGROUPED DATA)

WEIGHT	f
245	1
244	0
243	0
242	0
*	
*	
*	
161	0
160	4
159	1
158	2
157	3
*	
*	
*	
136	0
135	2
134	0
133	1
Total	53

Grouped Data

When observations are sorted into classes of more than one value, as in Table 2.2,the result is referred to as a frequency distribution for grouped data.

- Data are grouped into class intervals with 10 possible values each.
- The bottom class includes the smallest observation (133), and the top classincludes the largest observation (245).
- The distance between bottom and top is occupied by an orderly series of classes.
- The frequency (f) column shows the frequency of observations in each class and,at the bottom, the total number of observations in all classes.

Table 2.2 FREQUENCY DISTRIBUTION (GROUPED DATA)

WEIGHT	f
240–249	1
230–239	0
220–229	3
210–219	0
200–209	2
190–199	4
180–189	3
170–179	7
160–169	12
150–159	17
140–149	1
130–139	3
Total	53

Gaps between Classes:

- The size of the gap should always equal one unit of measurement.
- It should always equal the smallest possible difference between scores within a particular set of data.
- Since the gap is never bigger than one unit of measurement, no score can fall intothe gap.

How Many Classes?

- Classes should not be too large and not too high.

When There Are Either Many or Few Observations:

- Grouping of classes can be 10, the recommended number of classes, as recommended.

Real Limits of Class Intervals

- The real limits are located at the midpoint of the gap between adjacent tabled boundaries; that is, one-half of one unit of measurement below the lower tabled boundary and one-half of one unit of measurement above the upper tabled boundary.

Eg: The real limits for 140–149 in Table 2.2 are 139.5 (140 minus one-half of the unit of measurement of 1) and 149.5 (149 plus one-half of the unit of measurement of 1), and the actual width of the class interval would be 10 (from 149.5 139.5 = 10).

Table 2.3 FREQUENCY DISTRIBUTION WITH TOO MANY INTERVALS

WEIGHT	*f*
245–249	1
240–244	0
235–239	0
230–234	0
225–229	2
220–224	1
215–219	0
210–214	0
205–209	2
200–204	0
195–199	0
190–194	4
185–189	1
180–184	2
175–179	2
170–174	5
165–169	7
160–164	5
155–159	9
150–154	8
145–149	1
140–144	0
135–139	2
130–134	1
Total	53

Table 2.4 FREQUENCY DISTRIBUTION WITH TOO FEW INTERVALS

WEIGHT	*f*
200–249	6
150–199	43
100–149	4
Total	53

GUIDELINES

Essential:

1. Each observation should be included in one, and only one, class.

Example: 130–139, 140–149, 150–159, etc.

2. List all classes, even those with zero frequencies.

Example: Listed in Table 2.2 is the class 210–219 and its frequency of zero.

3. All classes should have equal intervals.

Example: 130–139, 140–149, 150–159, etc. It would be incorrect to use 130–139, 140–159, etc.,

Optional:

4. All classes should have both an upper boundary and a lower boundary.

Example: 240–249. Less preferred would be 240–above, in which no maximum value can be assigned to observations in this class.

5. Select the class interval from convenient numbers, such as 1, 2, 3, . . . 10, particularly 5 and 10 or multiples of 5 and 10.

Example: 130–139, 140–149, in which the class interval of 10 is a convenient number.

6. The lower boundary of each class interval should be a multiple of the class interval. Example: 130–139, 140–149, in which the lower boundaries of 130, 140, are multiples of10, the class interval.

7. Aim for a total of approximately 10 classes. Example: The distribution in Table 2.2 uses 12 classes.

CONSTRUCTING FREQUENCY DISTRIBUTIONS

1. Find the range
2. Find the class interval required to span the range by dividing the range by the desired number of classes
3. Round off to the nearest convenient interval
4. Determine where the lowest class should begin.
5. Determine where the lowest class should end.
6. Working upward, list as many equivalent classes as are required to include the largest observation.
7. Indicate with a tally the class in which each observation falls.
8. Replace the tally count for each class with a number—the frequency (f)—and showthe total of all frequencies.
9. Supply headings for both columns and a title for the table.

Problems:

Progress Check *2.2 The IQ scores for a group of 35 high school dropouts are as follows:

91	85	84	79	80
87	96	75	86	104
95	71	105	90	77
123	80	100	93	108
98	69	99	95	90
110	109	94	100	103
112	90	90	98	89

(a) Construct a frequency distribution for grouped data.

(b) Specify the *real* limits for the lowest class interval in this frequency distribution.

2.2 (a) Calculating the class width,

$$\frac{123-69}{10}=\frac{54}{10}=5.4$$

Round off to a convenient number, such as 5.

IQ	TALLY*	f
120–124	/	1
115–119		0
110–114	//	2
105–109	///	3
100–104	𝍸	4
95–99	𝍸 /	6
90–94	//// //	7
85–89	////	4
80–84	///	3
75–79	///	3
70–74	/	1
65–69	/	1
	Total	35

**Tally column usually is omitted from the finished table.*

(b) 64.5–69.5

Progress Check *2.3 What are some possible poor features of the following frequency distribution?

ESTIMATED WEEKLY TV VIEWING TIME (HRS) FOR 250 SIXTH GRADERS

VIEWING TIME	f
35–above	2
30–34	5
25–30	29
20–22	60
15–19	60
10–14	34
5–9	31
0–4	29
Total	250

2.3 Not all observations can be assigned to one and only one class (because of gap between 20–22 and 25–30 and overlap between 25–30 and 30–34). All classes are not equal in width (25–30 versus 30–34). All classes do not have both boundaries (35–above).

2.4 OUTLIERS

- The appearance of one or more very extreme scores are called outliers.
- Ex: A GPA of 0.06, an IQ of 170, summer wages of $62,000

Check for Accuracy

- Whenever you encounter an outrageously extreme value, such as a GPA of 0.06, attempt to verify its accuracy.
- If the outlier survives an accuracy check, it should be treated as a legitimate score.

Might Exclude from Summaries

- We might choose to segregate (but not to suppress!) an outlier from any summaryof the data.
- We might use various numerical summaries, such as the median and interquartile range, to that ignore extreme scores, including outliers.

Might Enhance Understanding

- A valid outlier can be viewed as the product of special circumstances, it might helpyou to understand the data.

Progress Check *2.4 Identify any outliers in each of the following sets of data collected from nine college students.

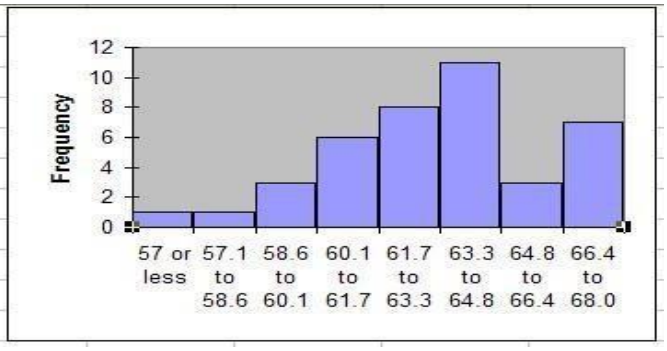
SUMMER INCOME	AGE	FAMILY SIZE	GPA
$6,450	20	2	2.30
$4,820	19	4	4.00
$5,650	61	3	3.56
$1,720	32	6	2.89
$600	19	18	2.15
$0	22	2	3.01
$3,482	23	6	3.09
$25,700	27	3	3.50
$8,548	21	4	3.20

2.4 Outliers are a summer income of $25,700; an age of 61; and a family size of 18. No outliers for GPA.

2.5 RELATIVE FREQUENCY DISTRIBUTIONS

Relative frequency distributions shows the frequency of each class as a part or fraction of the total frequency for the entire distribution.
Note that in the right column, the frequencies (counts) have been turned into relative frequencies (percents).
Count the total number of items. In this chart the total is 40.
Divide the count (the frequency) by the total number. For example, 1/40 = .025 or 3/40 = .075

Height	*Frequency*	*Rel. Freq.*
57 or less	1	0.025
57.1 to 58.6	1	0.025
58.6 to 60.1	3	0.075
60.1 to 61.7	6	0.15
61.7 to 63.3	8	0.2
63.3 to 64.8	11	0.275
64.8 to 66.4	3	0.075
66.4 to 68.0	7	0.175
Total	*40*	*1*



Progress Check *2.5

GRE scores for a group of graduate school applicants are distributed as follows

GRE	*f*
725–749	1
700–724	3
675–699	14
650–674	30
625–649	34
600–624	42
575–599	30
550–574	27
525–549	13
500–524	4
475–499	2
Total	200

GRE	RELATIVE *f*
725–749	.01
700–724	.02
675–699	.07
650–674	.15
625–649	.17
600–624	.21
575–599	.15
550–574	.14
525–549	.07*
500–524	.02
475–499	.01
Totals	1.02

**From 13/200 = .065, which rounds to .07.*

Convert to a relative frequency distribution. When calculating proportions, round numbers to two digits to the right of the decimal point, using the rounding procedure specified in Section A.7 of Appendix A.

Cumulative Frequency Distributions

A frequency distribution showing the total number of observations in each class and all lower-ranked classes. **Cumulative percentages** are often referred to as ***percentile ranks.***

Constructing Cumulative Frequency Distributions

Table 2.6
CUMULATIVE FREQUENCY DISTRIBUTION

WEIGHT	f	CUMULATIVE f	CUMULATIVE PERCENT
240–249	1	53	100
230–239	0	52	98
220–229	3	52	98
210–219	0	49	92
200–209	2	49	92
190–199	4	47	89
180–189	3	43	81
170–179	7	40	75
160–169	12	33	62
150–159	17	21	40
140–149	1	4	8
130–139	3	3	6
Total	53		

Progress Check *2.5 GRE scores for a group of graduate school applicants are distributed as follows:

GRE	f
725–749	1
700–724	3
675–699	14
650–674	30
625–649	34
600–624	42
575–599	30
550–574	27
525–549	13
500–524	4
475–499	2
Total	200

Convert the distribution of GRE scores shown in Question 2.5 to a cumulative frequency distribution.

Convert the distribution of GRE scores obtained in Question 2.6(a) to a cumulative percent frequency distribution

Answer

GRE	(a) CUMULATIVE f	(b) CUMULATIVE PERCENT(%)
725–749	200	100
700–724	199	100
675–699	196	98
650–674	182	91
625–649	152	76
600–624	118	59
575–599	76	38
550–574	46	23
525–549	19	10
500–524	6	3
475–499	2	1

Percentile Ranks

The percentile rank of a score indicates the percentage of scores in the entire distribution with similar or smaller values than that score

Thus a weight has a percentile rank of 80 if equal or lighter weights constitute 80 percent of the entire distribution

- Data set:
  - 2,2,3,4,5,5,5,6,7,8,8,8,8,8,9,9,10,11,11,12
  - What is the percentile ranking of "10"?
  - Percentile rank of x= $\frac{\#\ of\ values\ below}{n} * 100$

Percentile rank of '10' = $\frac{16}{20} * 100 = 80\%$

Progress Check *2.7

Referring to Table 2.6, find the approximate percentile rank of any weight in the class 200– 209

Answer

Table 2.6
CUMULATIVE FREQUENCY DISTRIBUTION

WEIGHT	*f*	CUMULATIVE *f*	CUMULATIVE PERCENT
240–249	1	53	100
230–239	0	52	98
220–229	3	52	98
210–219	0	49	92
200–209	2	49	92
190–199	4	47	89
180–189	3	43	81
170–179	7	40	75
160–169	12	33	62
150–159	17	21	40
140–149	1	4	8
130–139	3	3	6
Total	53		

The approximate percentile rank for weights between 200 and 209 lbs is 92 (because 92 is the cumulative percent for this interval).

Approximate Percentile Ranks Ungrouped Data[exact percentile]

FREQUENCY DISTRIBUTIONS FOR QUALITATIVE (NOMINAL) DATA

- When, among a set of observations, any single observation is a word, letter, or numerical code, the data are qualitative.
- Determine the frequency with which observations occupy each class, and report these frequencies.
- This frequency distribution reveals that Yes replies are approximately twice as prevalent as No replies.

Table 2.7
FACEBOOK PROFILE SURVEY

Response	*f*
Yes	56
No	27
Total	83

Ordered Qualitative Data

- Whether Yes is listed above or below No in Table 2.7.
- When, however, qualitative data have an ordinal level of measurement because observations can be ordered from least to most, that order should be preserved inthe frequency table.

Relative and Cumulative Distributions for Qualitative Data

- Frequency distributions for qualitative variables can always be converted into relative frequency distributions.

That a captain has an approximate percentile rank of 63 among officers since 62.5(or 63) is the cumulative percent for this class.

Table 2.8
RANKS OF OFFICERS IN THE U.S. ARMY (PROJECTED 2016)

RANK	*f*	PROPORTION	CUMULATIVE PERCENT
General	311	.004*	100.0
Colonel	13,156	.167	99.6
Major	16,108	.204	82.9
Captain	29,169	.370	62.5
Lieutenant	20,083	.255	25.5
Total	78,827		

Problem

Progress Check *2.8 Movie ratings reflect ordinal measurement because they can be ordered from most to least restrictive: NC-17, R, PG-13, PG, and G. The ratings of some films shown recently in San Francisco are as follows:

PG	PG	PG	PG-13	G
G	PG-13	R	PG	PG
R	PG	R	PG	R
NC-17	NC-17	PG	G	PG-13

(a) Construct a frequency distribution.

(b) Convert to relative frequencies, expressed as percentages.

(c) Construct a cumulative frequency distribution.

(d) Find the *approximate* percentile rank for those films with a PG rating.

2.8

MOVIE RATINGS	(a) *f*	(b) RELATIVE *f*(%)	(c) CUMULATIVE *f*
NC-17	2	10	20
R	4	20	18
PG-13	3	15	14
PG	8	40	11
G	3	15	3
Totals	20	100%	

(d) Percentile rank for films with a PG rating is 55 (from $\frac{11}{20}$ multiplied by 100).

2.7 GRAPHS

Data can be described clearly and concisely with the aid of a well-constructed frequency distribution. And data can often be described even more vividly by converting frequency distributions into graphs.

GRAPHS FOR QUANTITATIVE DATA

Histograms

A bar-type graph for quantitative data. The common boundaries between adjacent bars emphasize the continuity of the data, as with continuous variables.

A histogram is a display of statistical information that uses rectangles to show the frequency of data items in successive numerical intervals of equal size.

Important features of histograms

- Equal units along the horizontal axis (the X axis, or abscissa) reflect the various class intervals of the frequency distribution.
- Equal units along the vertical axis (the Y axis, or ordinate) reflect increases in frequency. (The units along the vertical axis do not have to be the same width as those along the horizontal axis.)
- The intersection of the two axes defines the origin at which both numerical scales equal 0.
- Numerical scales always increase from left to right along the horizontal axis and from bottom to top along the vertical axis
- The body of the histogram consists of a series of bars whose heights reflect the frequencies for the various classes.
- The adjacent bars in histograms have common boundaries that emphasize the continuity of quantitative data for continuous variables.
- The introduction of gaps between adjacent bars would suggest an artificial disruption in the data more appropriate for discrete quantitative variables or for qualitative variables.

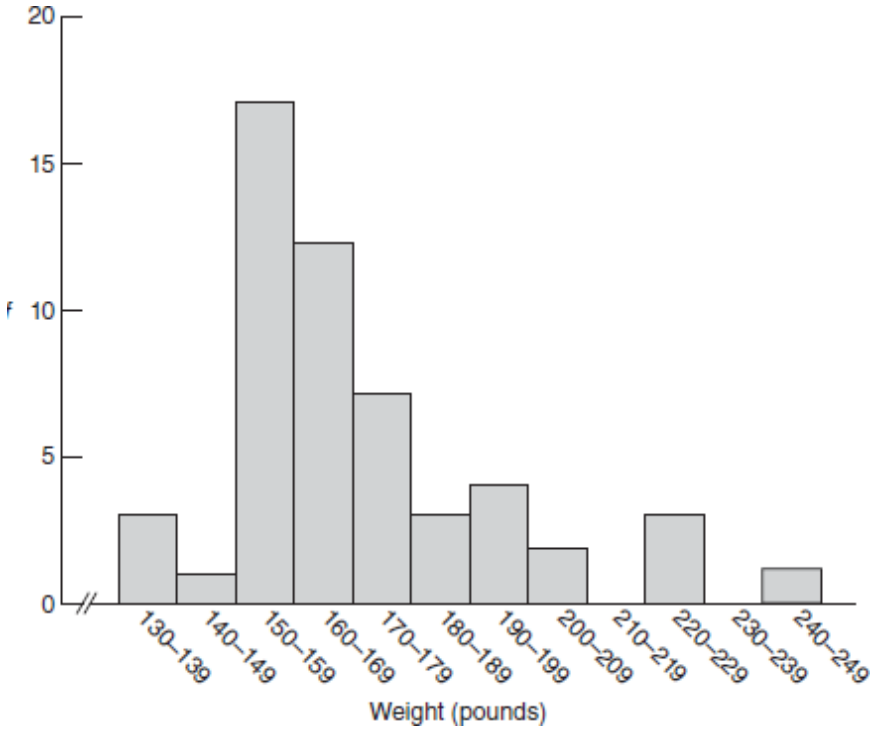


Figure: Histogram

Frequency Polygon

An important variation on a histogram is the frequency polygon, or line graph. Frequency polygons may be constructed directly from frequency distributions.

Step-by-step transformation of a histogram into a frequency polygon

A. This panel shows the histogram for the weight distribution.

B. Place dots at the midpoints of each bar top or, in the absence of bar tops, at midpoints for classes on the horizontal axis, and connect them with straight lines.

C. c. Anchor the frequency polygon to the horizontal axis. First, extend the upper tail to the midpoint of the first unoccupied class on the upper flank of the histogram. Then extend the lower tail to the midpoint of the first unoccupied class on the lower flank of the histogram. Now all of the area under the frequency polygon is enclosed completely.

D. Finally, erase all of the histogram bars, leaving only the frequency polygon.

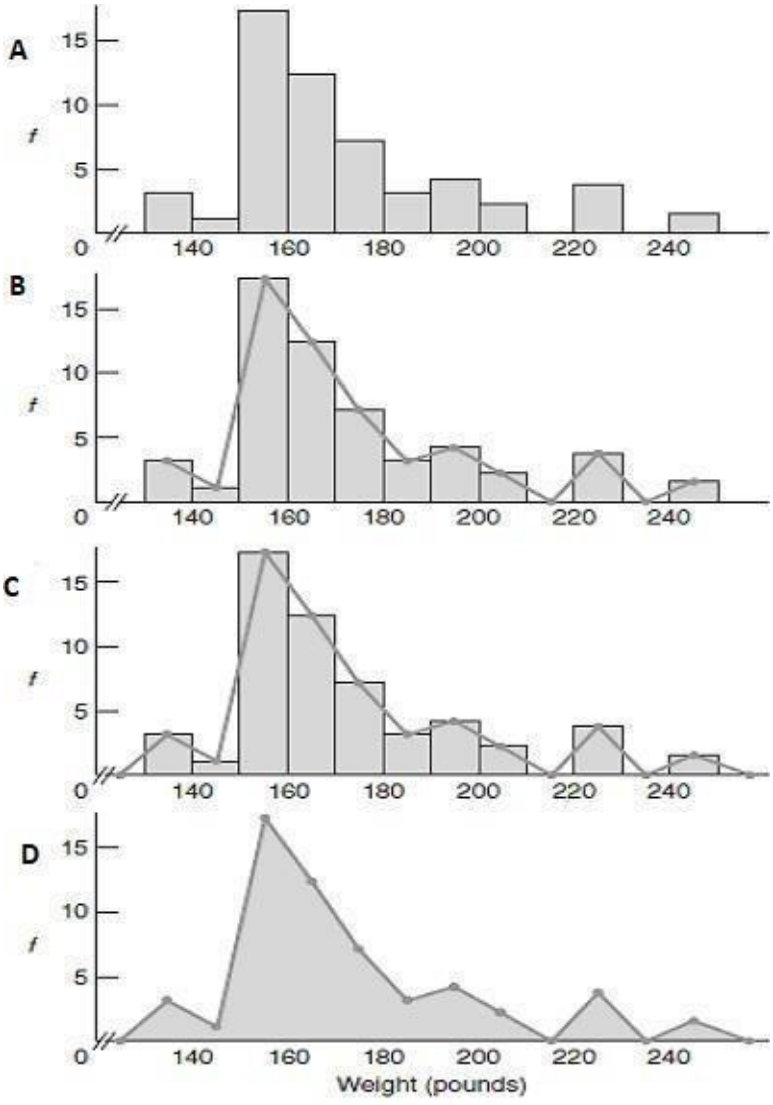


FIGURE
Transition from histogram to frequency polygon.

Stem and Leaf Displays

Another technique for summarizing quantitative data is a stem and leaf display. Stem and leaf displays are ideal for summarizing distributions, such as that for weight data, without destroying the identities of individual observations.

Constructing Stem and Leaf Display

The leftmost panel of table re-creates the weights.

To construct the stem and leaf display for the table given below, first note that, when counting by tens, the weights range from the 130s to the 240s.

Arrange a column of numbers, the stems, beginning with 13 (representing the 130s) and ending with 24 (representing the 240s). Draw a vertical line to separate the stems, which represent multiples of 10, from the space to be occupied by the leaves, which represent multiples of 1.

For example

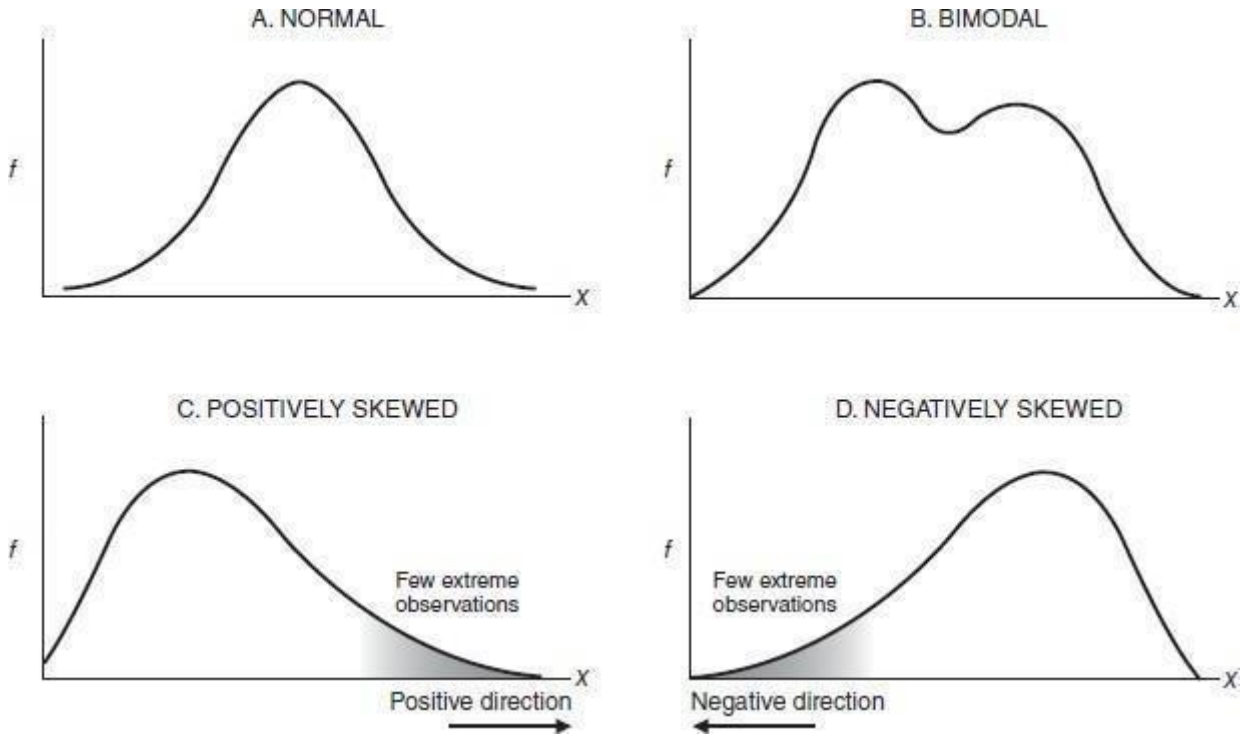
Enter each raw score into the stem and leaf display. As suggested by the shaded coding in Table 2.9, the first raw score of 160 reappears as a leaf of 0 on a stem of 16. The next raw score of 193 reappears as a leaf of 3 on a stem of 19, and the third raw score of 226 reappears as a leaf of 6 on a stem of 22, and so on, until each raw score reappears as a leaf on its appropriate stem.

Table 2.9
CONSTRUCTING STEM AND LEAF DISPLAY FROM WEIGHTS OF MALE STATISTICS STUDENTS

RAW SCORES					STEM AND LEAF DISPLAY
160	165	135	175		
193	168	245	165	13	3 5 5
226	169	170	185	14	5
152	160	156	154	15	2 7 1 7 8 0 2 0 2 6 9 8 2 6 4 7 6
180	170	160	179	16	0 3 5 8 9 0 0 0 6 5 5 5
205	150	225	165	17	2 0 0 0 2 5 9
163	152	190	206	18	0 0 5
157	160	159	165	19	3 0 0 0
151	190	172	157	20	5 6
157	150	190	156	21	
220	133	166	135	22	6 0 5
145	180	158		23	
158	152	152		24	5
172	170	156			

TYPICAL SHAPES

Whether expressed as a histogram, a frequency polygon, or a stem and leaf display, an important characteristic of a frequency distribution is its shape. Below figure shows some of the more typical shapes for smoothed frequency polygons (which ignore the inevitable irregularities of real data).



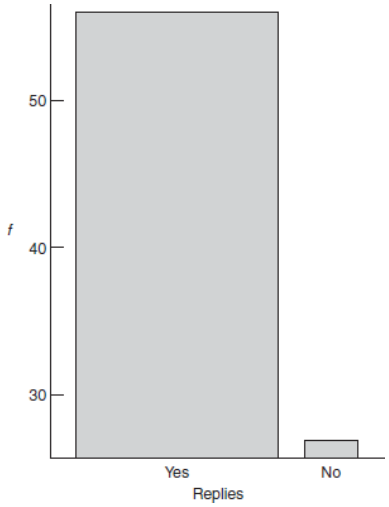
A GRAPH FOR QUALITATIVE (NOMINAL) DATA

- As with histograms, equal segments along the horizontal axis are allocated to the different words or classes that appear in the frequency distribution for qualitative data. Likewise, equal segments along the vertical axis reflect increases in frequency. The body of the bar graph consists of a series of bars whose heights reflect the frequencies for the various words or classes.
- A person's answer to the question ―Do you have a Facebook profile?‖ is either Yes or No, not some impossible intermediate value, such as 40 percent Yes and 60 percent No.
- Gaps are placed between adjacent bars of bar graphs to emphasize the discontinuous nature of qualitative data.

MISLEADING GRAPHS

Graphs can be constructed in an unscrupulous manner to support a particular point of view.

Popular sayings says, including ―Numbers don't lie, but statisticians do‖ and ―There are three kinds of lies—



lies, damned lies, and statistics.‖

STEPS

CONSTRUCTING GRAPHS

1. ***Decide on the appropriate type of graph,*** recalling that histograms and frequency polygons are appropriate for quantitative data, while bar graphs are appropriate for qualitative data and also are sometimes used with discrete quantitative data.
2. ***Draw the horizontal axis, then the vertical axis,*** remembering that the vertical axis should be about as tall as the horizontal axis is wide.
3. ***Identify the string of class intervals that eventually will be superimposed on the horizontal axis.*** For qualitative data or ungrouped quantitative data, this is easy—just use the classes suggested by the data. For grouped quantitative data, proceed as if you were creating a set of class intervals for a frequency distribution. (See the box "Constructing Frequency Distributions" on page 27.)
4. ***Superimpose the string of class intervals (with gaps for bar graphs) along the entire length of the horizontal axis.*** For histograms and frequency polygons, be prepared for some trial and error—use a pencil! Do not use a string of empty class intervals to bridge a sizable gap between the origin of 0 and the smallest class interval. Instead, use wiggly lines to signal a break in scale, then begin with the smallest class interval. Also, do not clutter the horizontal scale with excessive numbers—use just a few convenient numbers.
5. ***Along the entire length of the vertical axis, superimpose a progression of convenient numbers,*** beginning at the bottom with 0 and ending at the top with a number as large as or slightly larger than the maximum observed frequency. If there is a considerable gap between the origin of 0 and the smallest observed frequency, use wiggly lines to signal a break in scale.
6. Using the scaled axes, ***construct bars (or dots and lines) to reflect the frequency of observations within each class interval.*** For frequency polygons, dots should be located above the midpoints of class intervals, and both tails of the graph should be anchored to the horizontal axis, as described under "Frequency Polygons" in Section 2.8.
7. ***Supply labels for both axes and a title (or even an explanatory sentence) for the graph.***

2.8 DESCRIBING DATA WITH AVERAGES

MODE

The mode reflects the value of the most frequently occurring score. In other words

A mode is defined as the value that has a higher frequency in a given set of values. It is the value that appears the most number of times.

Example:

In the given set of data: 2, 4, 5, 5, 6, 7, the mode of the data set is 5 since it has appeared in the set twice.

Types of Modes

Bimodal, Trimodal & Multimodal (More than one mode)

- When there are two modes in a data set, then the set is called **bimodal**

For example, The mode of Set A = {2,2,2,3,4,4,5,5,5} is 2 and 5, because both 2 and 5 is repeated three times in the given set.

- When there are three modes in a data set, then the set is called **trimodal**

For example, the mode of set A = {2,2,2,3,4,4,5,5,5,7,8,8,8} is 2, 5 and 8

- When there are four or more modes in a data set, then the set is called **multimodal**

Example: The following table represents the number of wickets taken by a bowler in 10 matches. Find the mode of the given set of data.

Match No.	1	2	3	4	5	6	7	8	9	10
No. of Wickets	2	1	1	3	2	3	2	2	4	1

It can be seen that 2 wickets were taken by the bowler frequently in different matches. Hence, the mode of the given data is 2.

MEDIAN

The median reflects the middle value when observations are ordered from least to most.
The median splits a set of ordered observations into two equal parts, the upper and lower halves.

Finding the Median

- Order scores from least to most.
- If the total number of observation given is odd, then the formula to calculate the median is: Median = $\{(n+1)/2\}^{th}$ term / observation
- If the total number of observation is even, then the median formula is: Median = $1/2[(n/2)^{th}$ term + $\{(n/2)+1\}^{th}$ term]

Example 1:

Find the median of the following:

4, 17, 77, 25, 22, 23, 92, 82, 40, 24, 14, 12, 67, 23, 29

Solution:
n= 15
When we put those numbers in the order we have:

4, 12, 14, 17, 22, 23, 23, 24, 25, 29, 40, 67, 77, 82, 92,

Median = $\{(n+1)/2\}^{th}$ term
= (15+1)/2
=8
The 8^{th} term in the list is 24
The median value of this set of numbers is 24.

Example 2:
Find the median of the following:
9,7,2,11,18,12,6,4

Solution n=8
When we put those numbers in the order we have:
2, 4, 6, 7, 9,11, 12, 18

Median = $1/2[(n/2)^{th}$ term + $\{(n/2)+1\}^{th}$ term]

= ½ [(8/2) term + ((8/2)+1)term]
=$1/2[4^{th}$ term+5^{th} term] (in our list 4^{th} term is 7 and 5^{th} term is 9)
= ½[7+9]
=1/2(16)
=8
The median value of this set of numbers is 8.

MEAN
The mean is found by adding all scores and then dividing by the number of scores.

Mean is the average of the given numbers and is calculated by dividing the sum of given numbers by the total number of numbers.

$$Mean = \frac{sum\ of\ all\ scores}{number\ of\ scores}$$

Types of means
- Sample mean
- Population mean

Sample Mean
The sample mean is a central tendency measure. The arithmetic average is computed using samples or random values taken from the population. It is evaluated as the sum of all the sample variables divided by the total number of variables.

SAMPLE MEAN

$$\bar{X} = \frac{\sum X}{n}$$

Population Mean
The population mean can be calculated by the sum of all values in the given data/population divided by a total number of values in the given data/population.

POPULATION MEAN

$$\mu = \frac{\sum X}{N}$$

AVERAGES FOR QUALITATIVE AND RANKED DATA

Mode
he mode always can be used with qualitative data.

Median
The median can be used whenever it is possible to order qualitative data from least to most because the level of measurement is ordinal.

2.9 DESCRIBING VARIABILITY

RANGE
The range is the difference between the largest and smallest scores.
The range in statistics for a given data set is the difference between the highest and lowest values. For example, if the given data set is {2,5,8,10,3}, then the range will be 10 – 2 = 8.
Example 1: Find the range of given observations: 32, 41, 28, 54, 35, 26, 23, 33, 38, 40. Solution: Let us first arrange the given values in ascending order.
23, 26, 28, 32, 33, 35, 38, 40, 41, 54
Since 23 is the lowest value and 54 is the highest value, therefore, the range of the observations will be;
Range (X) = Max (X) – Min (X)
= 54 – 23
= 31

VARIANCE
Variance is a measure of how data points differ from the mean. A variance is a measure of how far a set of data (numbers) are spread out from their mean (average) value.
Formula
$\sigma = \Sigma(x-\mu)^2$ or
Variance = (Standard deviation)^2= σ^2 $=> \sigma^2 = \Sigma(x-\mu)^2 /n$

the values of all scores must be added and then divided by the total number of scores. Example
X = 5, 8, 6, 10, 12, 9, 11, 10, 12, 7
Solution
Mean = sum (x)/ n n= 10
sum (x) = 5+8+6+10+12+9+11+10+12+ 7
= 90
Mean=> μ = 90 / 10 = 9 Deviation from mean
$x-\mu$ = -4, -1, -3, 1, 3, 0, 2,1,3,-2

$(x-\mu)^2$ = 16,1,9,1,9,0,4,1,9,4

$\Sigma(x-\mu)^2$ = 16+1+9+1+9+0+4+1+9+4
=54

$\sigma^2 = \Sigma(x-\mu)^2 /n$

=54/10
= 5.4

STANDARD DEVIATION

The standard deviation, the square root of the mean of all squared deviations from the mean, that is,
Standard deviation = √variance

Standard Deviation: A rough measure of the average (or standard) amount by which scores deviate

Standard Deviation: A Measure of Distance

The mean is a measure of position, but the standard deviation is a measure f distance (on either side of the mean of the distribution).

Sum of Squares (SS)

Calculating the standard deviation requires that we obtain first a value for the variance. However, calculating the variance requires, in turn, that we obtain the sum of the squared deviation scores.

The sum of squared deviation scores or more simply the sum of squares, symbolized by SS

SUM OF SQUARES (*SS*) FOR POPULATION (DEFINITION FORMULA)

$$SS = \sum (X - \mu)^2 \qquad (4.1)$$

—The sum of squares equals the sum of all squared deviation scores.‖ You can reconstruct this formula by remembering the following three steps:

1. Subtract the population mean, μ, from each original score, X, to obtain a deviation score, X − μ.
2. Square each deviation score, (X − μ)2, to eliminate negative signs.
3. Sum all squared deviation scores, Σ (X − μ)2.

VARIANCE FOR POPULATION

$$\sigma^2 = \frac{SS}{N} \qquad (4.5)$$

STANDARD DEVIATION FOR POPULATION

$$\sigma = \sqrt{\sigma^2} = \sqrt{\frac{SS}{N}} \qquad (4.6)$$

Table 4.1
CALCULATION OF POPULATION STANDARD DEVIATION σ (DEFINITION FORMULA)

A. COMPUTATION SEQUENCE

Assign a value to N 1 representing the number of X scores
Sum all X scores 2
Obtain the mean of these scores 3
Subtract the mean from each X score to obtain a deviation score 4
Square each deviation score 5
Sum all squared deviation scores to obtain the sum of squares 6
Substitute numbers into the formula to obtain population variance, σ^2 7
Take the square root of σ^2 to obtain the population standard deviation, σ 8

B. DATA AND COMPUTATIONS

X	4 $X-\mu$	5 $(X-\mu)^2$
13	3	9
10	0	0
11	1	1
7	–3	9
9	–1	1
11	1	1
9	–1	1

1 $N = 7$ 2 $\Sigma X = 70$ 6 $SS = \Sigma (X-\mu)^2 = 22$

3 $\mu = \frac{70}{7} = 10$

7 $\sigma^2 = \frac{SS}{N} = \frac{22}{7} = 3.14$ 8 $\sigma = \sqrt{\frac{SS}{N}} = \sqrt{\frac{22}{7}} = \sqrt{3.14} = 1.77$

Sum of Squares Formulas for Sample

Sample notation can be substituted for population notation in the above two formulas without causing any essential changes:

SUM OF SQUARES (*SS*) FOR SAMPLE (DEFINITION FORMULA)

$$SS = \Sigma(X - \overline{X})^2 \qquad (4.3)$$

VARIANCE FOR SAMPLE

$$s^2 = \frac{SS}{n-1} \qquad (4.7)$$

STANDARD DEVIATION FOR SAMPLE

$$s = \sqrt{s^2} = \sqrt{\frac{SS}{n-1}} \qquad (4.8)$$

Table 4.3
CALCULATION OF SAMPLE STANDARD DEVIATION (S)
(DEFINITION FORMULA)

A. COMPUTATION SEQUENCE

Assign a value to n **1** representing the number of X scores
Sum all X scores **2**
Obtain the mean of these scores **3**
Subtract the mean from each X score to obtain a deviation score **4**
Square each deviation score **5**
Sum all squared deviation scores to obtain the sum of squares **6**
Substitute numbers into the formula to obtain the sample variance, s^2 **7**
Take the square root of s^2 to obtain the sample standard deviation, s **8**

B. DATA AND COMPUTATIONS

X	**4** $X-\bar{X}$	**5** $(X-\bar{X})^2$
7	4	16
3	0	0
1	−2	4
0	−3	9
4	1	1

1 $n=5$ **2** $\Sigma X=15$ **6** $SS=\Sigma(X-\bar{X})^2=30$

3 $\bar{X}=\frac{15}{5}=3$

7 $s^2=\frac{SS}{n-1}=\frac{30}{4}=7.50$ **8** $s=\sqrt{\frac{SS}{n-1}}=\sqrt{\frac{30}{4}}=\sqrt{7.50}=2.74$

Table 4.5
TWO ESTIMATES OF POPULATION VARIABILITY

WHEN μ IS UNKNOWN ($\bar{X}=3$)

X	$X-\bar{X}$	$(X-\bar{X})^2$
7	$7-3=4$	16
3	$3-3=0$	0
1	$1-3=-2$	4
0	$0-3=-3$	9
4	$4-3=1$	1

$\Sigma(X-\bar{X})=0$ $\Sigma(X-\bar{X})^2=30$

$df=n-1=5-1=4$

$s^2(df=n-1)=\frac{\Sigma(X-\bar{X})^2}{n-1}=\frac{30}{4}=7.50$

WHEN μ IS KNOWN ($\mu=2$)

X	$X-\mu$	$(X-\mu)^2$
7	$7-2=5$	25
3	$3-2=1$	1
1	$1-2=-1$	1
0	$0-2=-2$	4
4	$4-2=2$	4

$\Sigma(X-\mu)=5$ $\Sigma(X-\mu)^2=35$

$df=n=5$

$s^2(df=n)=\frac{\Sigma(X-\mu)^2}{n}=\frac{35}{5}=7.00$

DEGREES OF FREEDOM (df)

- Degrees of freedom (df) refers to the number of values that are free to vary, given one or more mathematical restrictions, in a sample being used to estimate a population characteristic.
- Degrees of freedom are the number of independent variables that can be estimated in a statistical analysis. These values of these variables are without constraint, although the values do impost restrictions on other variables if the data set is to comply with estimate parameters.
- Degrees of Freedom (df) The number of values free to vary, given one or more mathematical restrictions.

Formula
Degree of freedom **df = n-1**

Example
Consider a data set consists of five positive integers. The sum of the five integers must be the multiple of 6. The values are randomly selected as 3, 8, 5, and 4.
The sum of this for values is 20. So we have to choose the fifth integer to make the sum divisible by 6. Therefore the fifth element is 10.

The number of degrees of Degrees of Freedom (df) The number of values free to vary, given one or more mathematical restrictions. Freedom—in the numerator, as in the formulas for s2 and s. In fact, we can use degrees of freedom to rewrite the formulas for the sample variance and standard deviation:

VARIANCE FOR SAMPLE

$$s^2 = \frac{SS}{n-1} = \frac{SS}{df} \quad (4.9)$$

STANDARD DEVIATION FOR SAMPLE

$$s = \sqrt{\frac{SS}{n-1}} = \sqrt{\frac{SS}{df}} \quad (4.10)$$

2.10 INTERQUARTILE RANGE (IQR)

The interquartile range (IQR), is simply the range for the middle 50 percent of the scores. More specifically, the IQR equals the distance between the third quartile (or 75th percentile) and the first quartile (or 25th percentile), that is, after the highest quarter (or top 25 percent) and the lowest quarter (or bottom 25 percent) have been trimmed from the original set of scores. Since most distributions are spread more widely in their extremities than their middle, the IQR tends to be less than half the size of the range.

Simply, The IQR describes the middle 50% of values when ordered from lowest to highest. To find the interquartile range (IQR), first find the median (middle value) of the lower and upper half of the data. These values are quartile 1 (Q1) and quartile 3 (Q3). The IQR is the difference between Q3 and Q1.

Table 4.6
CALCULATION OF THE IQR

A. INSTRUCTIONS

1. Order scores from least to most.
2. To determine how far to penetrate the set of ordered scores, begin at either end, then add 1 to the total number of scores and divide by 4. If necessary, round the result to the nearest whole number.
3. Beginning with the largest score, count the requisite number of steps (calculated in step 2) into the ordered scores to find the location of the third quartile.
4. The third quartile equals the value of the score at this location.
5. Beginning with the smallest score, again count the requisite number of steps into the ordered scores to find the location of the first quartile.
6. The first quartile equals the value of the score at this location.
7. The IQR equals the third quartile minus the first quartile.

B. EXAMPLE

1. 7, 9, 9, 10, 11, 11, 13
2. $(7 + 1)/4 = 2$
3. 7, 9, 9, 10, 11, 11, 13
   2 1
4. third quartile = 11
5. 7, 9, 9, 10, 11, 11, 13
   1 2
6. first quartile = 9
7. IQR = 11 − 9 = 2

2.11 NORMAL DISTRIBUTIONS AND STANDARD (Z) SCORES

THE NORMAL CURVE

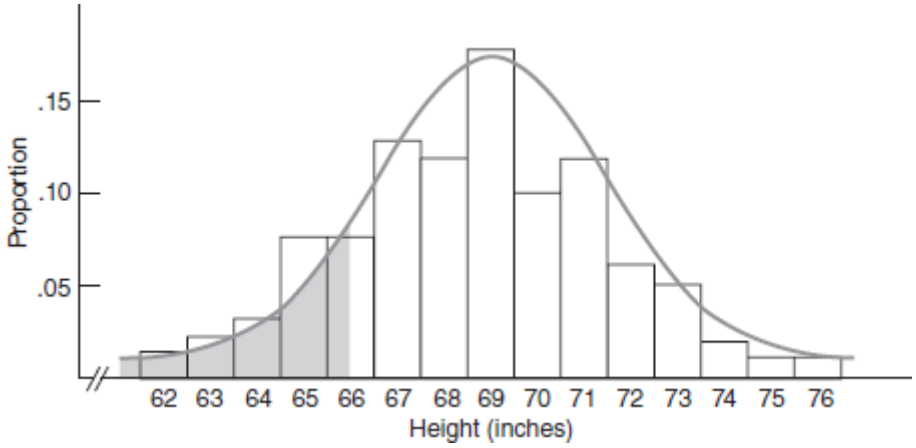
The normal distribution is a continuous probability distribution that is symmetrical on both sides of the mean, so the right side of the center is a mirror image of the left side.

Properties of the Normal Curve

- The normal curve is a theoretical curve defined for a continuous variable, as described in Section 1.6, and noted for its symmetrical bell-shaped form, as revealed in below figure
- Because the normal curve is symmetrical, its lower half is the mirror image of its upper half.
- The normal curve peaks above a point midway along the horizontal spread and then tapers off gradually in either direction from the peak (without actually touching the horizontal axis, since, in theory, the tails of a normal curve extend infinitely far).
- The values of the mean, median (or 50th percentile), and mode, located at a point midway along the horizontal spread, are the same for the normal curve.

Properties of a normal distribution

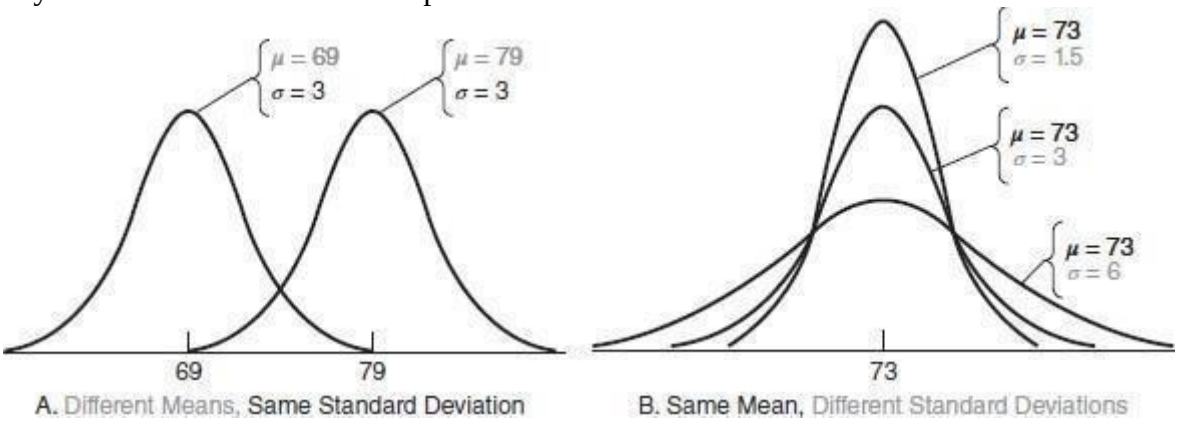
- The mean, mode and median are all equal.
- The curve is symmetric at the center (i.e. around the mean, μ).
- Exactly half of the values are to the left of center and exactly half the values are to the right.
- The total area under the curve is 1.



Different Normal Curves

As a theoretical exercise, it is instructive to note the various types of normal curves that are produced by an arbitrary change in the value of either the mean (μ) or the standard deviation (σ).

Obvious differences in appearance among normal curves are less important than you might suspect. Because of their common mathematical origin, every normal curve can be interpreted in exactly the same way once any distance from the mean is expressed in standard deviation units.



z SCORES

A z score is a unit-free, standardized score that, regardless of the original units of measurement, indicates how many standard deviations a score is above or below the mean of its distribution.

A z score can be defined as a measure of the number of standard deviations by which a score is below or above the mean of a distribution. In other words, it is used to determine the distance of a score from the mean. If the z score is positive it indicates that the score is above the mean. If it is negative then the score will be below the mean. However, if the z score is 0 it denotes that the data point is the same as the mean.

To obtain a z score, express any original score, whether measured in inches, milliseconds, dollars, IQ points, etc., as a deviation from its mean (by subtracting its mean) and then split this deviation into standard deviation units (by dividing by its standard deviation),

$$z = \frac{X - \mu}{\sigma}$$

Where X is the original score and μ and σ are the mean and the standard deviation, respectively, for the normal distribution of the original scores. Since identical units of measurement appear in both the numerator

and denominator of the ratio for z, the original units of measurement cancel each other and the z score emerges as a unit-free or standardized number, often referred to as a standard score.

A z score consists of two parts:
1. A positive or negative sign indicating whether it's above or below the mean; and
2. A number indicating the size of its deviation from the mean in standard deviation units.

Converting to z Scores Example

Suppose on a GRE test a score of 1100 is obtained. The mean score for the GRE test is 1026 and the population standard deviation is 209. In order to find how well a person scored with respect to the score of an average test taker, the z score will have to be determined.

The steps to calculate the z score are as follows:

- Step 1: Write the value of the raw score in the z score equation. $z = (1100-\mu)/\sigma$
- Step 2: Write the mean and standard deviation of the population in the z score formula. $z = (1100-1026)/209$
- Step 3: Perform the calculations to get the required z score. $z = 0.345$
- Step 4: A z score table can be used to find the percentage of test-takers that are below the score of the person. Using the first two digits of the z score, determine the row containing these digits of the z table. Now using the 2nd digit after the decimal, find the corresponding column. The intersection of this row and column will give a value. As shown below, this value will be 0.6368 for the given example.
- Step 5: Use the value from step 5 and multiply it by 100 to get the required percentage. 0.6368 * 100 = 63.68%. This shows that 63.68% of test-takers scores are lesser than the given raw score.

STANDARD NORMAL CURVE

If the original distribution approximates a normal curve, then the shift to standard or z scores will always produce a new distribution that approximates the standard normal curve. This is the one normal curve for which a table is actually available**.**

Although there is an infinite number of different normal curves, each with its own mean and standard deviation, there is only one standard normal curve, with a mean of 0 and a standard deviation of 1.

For a standard normal curve

$$\text{Mean of } z = \frac{X-\mu}{\sigma} = \frac{\mu-\mu}{\sigma} = \frac{0}{\sigma} = 0$$

Mean = 0

Standard deviation = 1

$$\text{Standard deviation of } z = \frac{X-\mu}{\sigma} = \frac{\mu+1\sigma-\mu}{\sigma} = \frac{1\sigma}{\sigma} = 1$$

Standard Normal Table

The standard normal table consists of columns of z scores coordinated with columns of proportions

Using the Top Legend of the Table

Notice that columns are arranged in sets of three, designated as A, B, and C in the legend at the top of the table. When using the top legend, all entries refer to the upper half of the standard normal curve. The entries in column A are z scores, beginning with 0.00 and ending with 4.00

Given a z score of zero or more, columns B and C indicate how the z score splits the area in the upper half of the normal curve. As suggested by the shading in the top legend, column B indicates the proportion of area between the mean and the z score, and column C indicates the proportion of area beyond the z score, in the upper tail of the standard normal curve.

Using the Bottom Legend of the Table

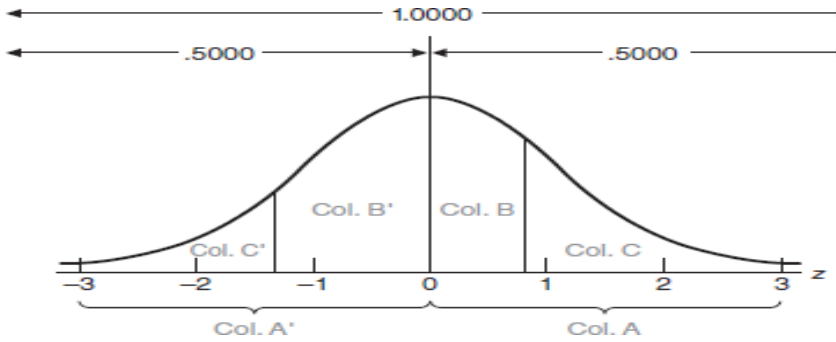
Now the columns are designated as A′, B′, and C′ in the legend at the bottom of the table. When using the bottom legend, all entries refer to the lower half of the standard normal curve.

A negative z score, columns B′ and C′ indicate how that z score splits the lower half of the normal curve. As suggested by the shading in the bottom legend of the table, column B′ indicates the proportion of area between the mean and the negative z score, and column C′ indicates the proportion of area beyond the negative z score, in the lower tail of the standard normal curve.

TABLE 5.1
PROPORTIONS (OF AREAS) UNDER THE STANDARD NORMAL CURVE FOR VALUES OF *z* (FROM TABLE A OF APPENDIX C)

A z	B	C	A z	B	C	A z	B	C
0.00	.0000	.5000	0.40	.1554	.3446	0.80	.2881	.2119
0.01	.0040	.4960	0.41	.1591	.3409	0.81	.2910	.2090
•	•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•	•
						0.99	.3389	.1611
						1.00	.3413	.1587
•	•	•	•	•	•	1.01	.3438	.1562
•	•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•	•
0.38	.1480	.3520	0.78	.2823	.2711	1.18	.3810	.1190
0.39	.1517	.3483	0.79	.2852	.2148	1.19	.3830	.1170

−z A′	B′	C′	−z A′	B′	C′	−z A′	B′	C′



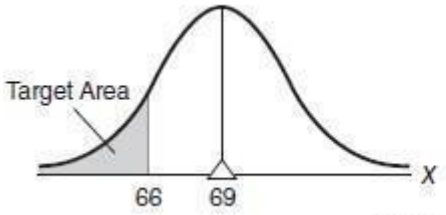
FINDING PROPORTIONS

Finding Proportions for One Score

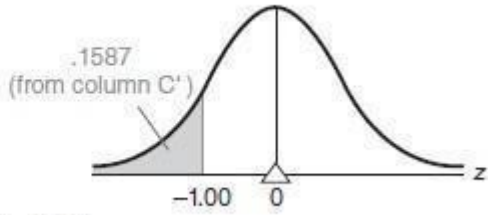
- Sketch a normal curve and shade in the target area,
- Plan your solution according to the normal table.
- Convert X to z.

$$z = \frac{X - \mu}{\sigma}$$

Find: Proportion Below 66



Solution:



Answer: .1587

- Find the target area.

Finding Proportions between Two Scores

- Sketch a normal curve and shade in the target area, (example, find proportion between 245 to 255)
- Plan your solution according to the normal table.
- Convert X to z by expressing 255 as

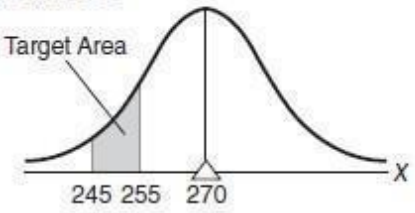
$$z = \frac{255 - 270}{15} = \frac{-15}{15} = -1.00$$

and by expressing 245 as

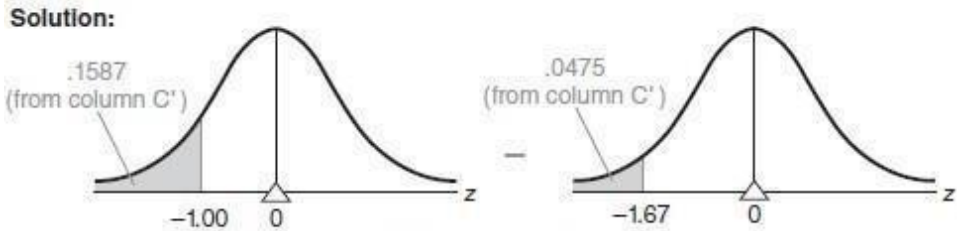
$$z = \frac{245 - 270}{15} = \frac{-25}{15} = -1.67$$

- Find the target area.

Find: Proportion Between 245 and 255



Solution:



Answer: .1587
– .0475
.1112

FINDING SCORES

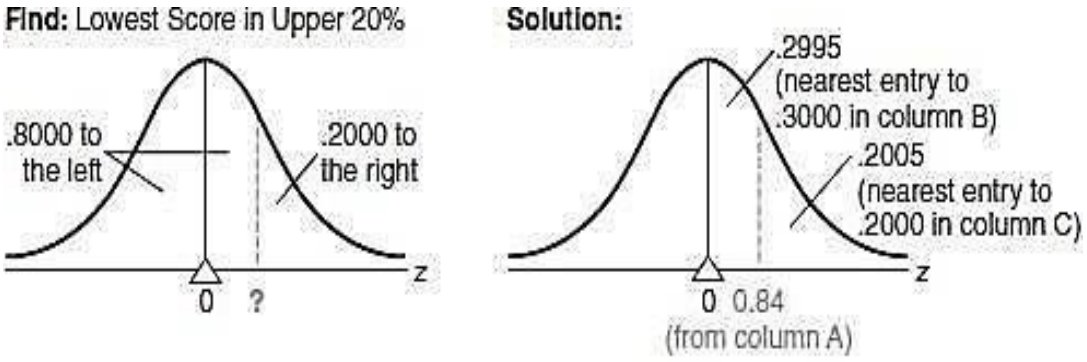
So far, we have concentrated on normal curve problems for which Table A must be consulted to find the unknown proportion (of area) associated with some known score or pair of known scores

Now we will concentrate on the opposite type of normal curve problem for which Table A must be consulted to find the unknown score or scores associated with some known proportion.

For this type of problem requires that we reverse our use of Table A by entering proportions in columns B, C, B′, or C′ and finding z scores listed in columns A or A′.

Finding One Score

- Sketch a normal curve and, on the correct side of the mean, draw a line representing the target score, as in figure



Answer: $X = \mu + (z)(\sigma)$
$= 230 + (0.84)(50)$
$= 230 + 42$
$= 272$

It's often helpful to visualize the target score as splitting the total area into two sectors—one to the left of (below) the target score and one to the right of (above) the target score

- Plan your solution according to the normal table.

In problems of this type, you must plan how to find the z score for the target score. Because the target score is on the right side of the mean, concentrate on the area in the upper half of the normal curve, as described in columns B and C.

- Find z.
- Convert z to the target score.

When converting z scores to original scores, you will probably find it more efficient to use the following equation

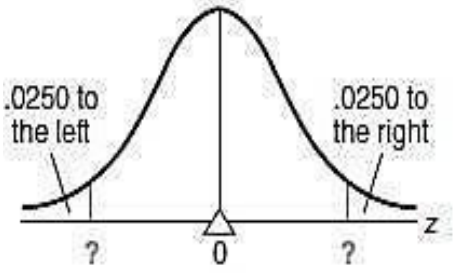
CONVERTING z SCORE TO ORIGINAL SCORE

$$X = \mu + (z)(\sigma)$$

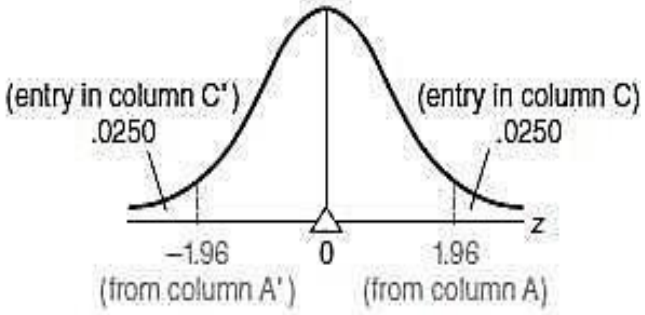
Finding Two Scores

- Sketch a normal curve. On either side of the mean, draw two lines representing the two target scores, as in figure

Find: Pairs of Scores for the Extreme 2.5%



Solution:



Answer: $X_{min} = \mu + (z)(\sigma)$
$= 22 + (-1.96)(4)$
$= 22 - 7.84$
$= 14.16$

Answer: $X_{max} = \mu + (z)(\sigma)$
$= 22 + (1.96)(4)$
$= 22 + 7.84$
$= 29.84$

- Plan your solution according to the normal table.
- Find z.
- Convert z to the target score.

Points to Remember

1. range = largest value – smallest value in a list

2. class interval = range / desired no of classes

3. relative frequency = frequency (f)/ε(f)

4. Cumulative frequency - *add to the frequency of each class the sum of the frequencies of all classes ranked below it.*

5. Cumulative percentage = (f/cumulative f)*100

6. Histograms

7. Construction of frequency polygon

8. Stem and leaf display

9. Mode - *The value of the most frequent score.*

10. For odd no of terms **Median = {(n+1)/2}th term / observation**. For even no of terms ***Median = 1/2[(n/2)th term + {(n/2)+1}th term]***

11. Mean = sum of all scores / number of scores

SAMPLE MEAN $\bar{X} = \frac{\sum X}{n}$ **POPULATION MEAN** $\mu = \frac{\sum X}{N}$

Variance $\sigma = \Sigma(x-\mu)^2$ or

Variance = (Standard deviation)^2= σ^2 $=> \sigma^2 = \Sigma(x-\mu)^2 /n$

12. Range (X) = Max (X) – Min (X)

STANDARD DEVIATION FOR POPULATION

$$\sigma = \sqrt{\sigma^2} = \sqrt{\frac{SS}{N}} \quad (4.6)$$

SUM OF SQUARES (SS) FOR POPULATION (DEFINITION FORMULA)

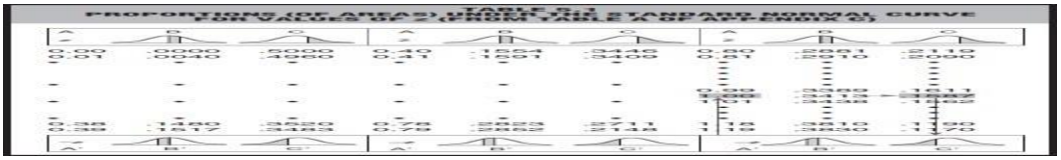
$$SS = \sum (X - \mu)^2 \quad (4.1)$$

VARIANCE FOR SAMPLE

$$s^2 = \frac{SS}{n-1} \quad (4.7)$$

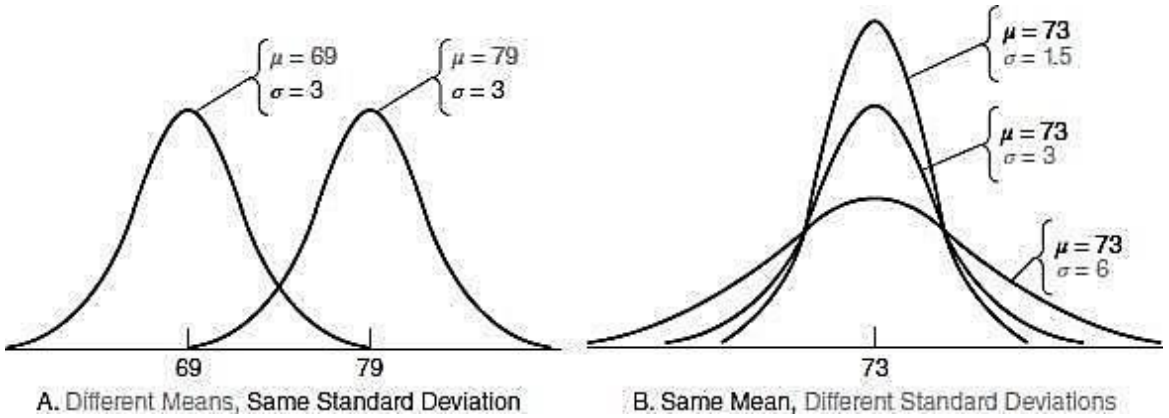
STANDARD DEVIATION FOR SAMPLE

$$s = \sqrt{s^2} = \sqrt{\frac{SS}{n-1}} \quad (4.8)$$



13. Degree of freedom **df = n-1**

14. Types of normal curve



A. Different Means, Same Standard Deviation

B. Same Mean, Different Standard Deviations

15. z – score

z SCORE

$$z = \frac{X - \mu}{\sigma} \quad (5.1)$$

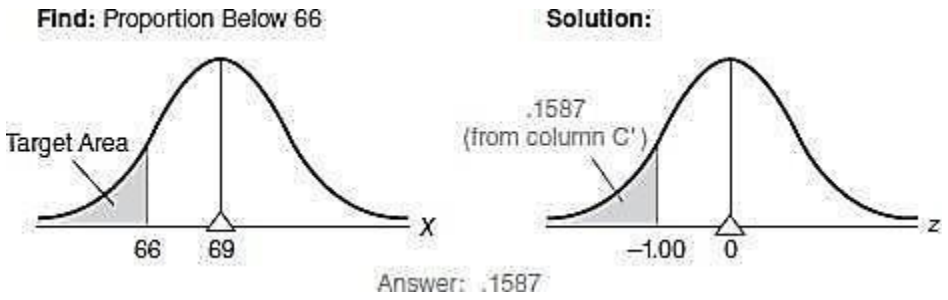
16. Standard normal curve; **mean = 0, standard deviation = 1**

17. Finding proportion

$$z = \frac{X - \mu}{\sigma}$$

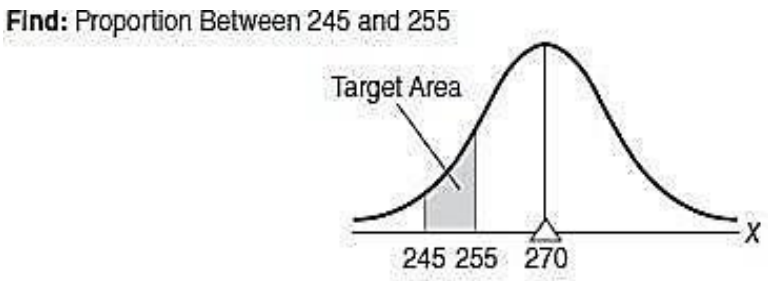
18. Finding proportion

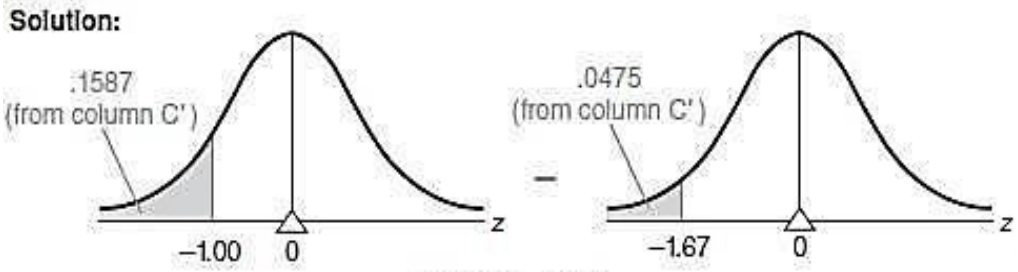
1. For one score



Answer: .1587

2. For between two score





Answer: .1587
−.0475
.1112

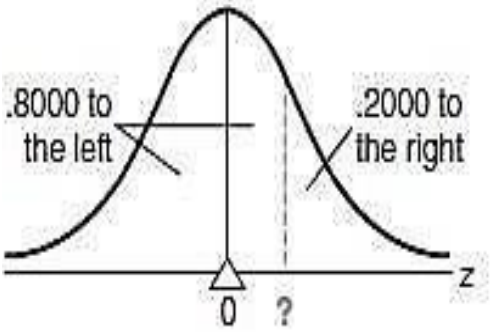
19. Finding scores

CONVERTING z SCORE TO ORIGINAL SCORE

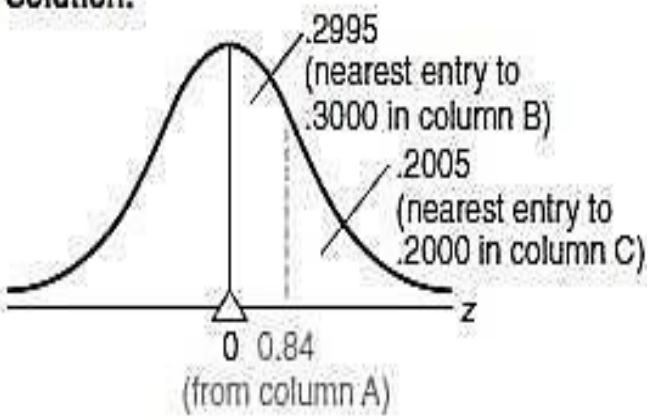
$$X = \mu + (z)(\sigma)$$

20. Finding scores – one score

Find: Lowest Score in Upper 20%



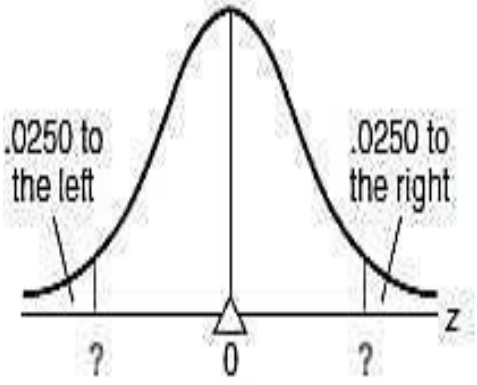
Solution:



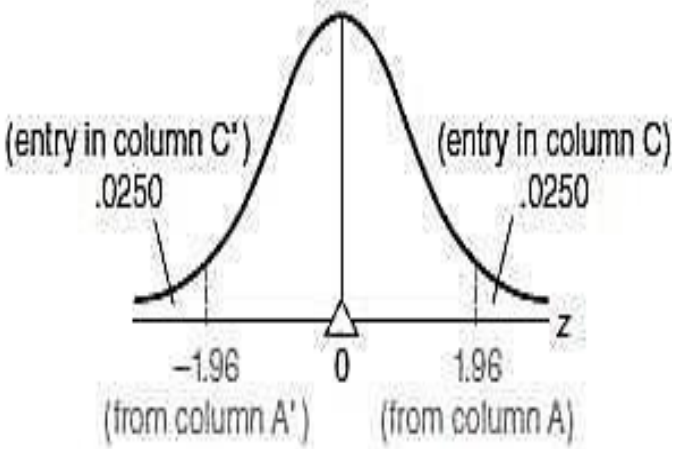
Answer: $X = \mu + (z)(\sigma)$
$= 230 + (0.84)(50)$
$= 230 + 42$
$= 272$

Two scores

Find: Pairs of Scores for the Extreme 2.5%



Solution:



Answer: $X_{min} = \mu + (z)(\sigma)$
$= 22 + (-1.96)(4)$
$= 22 - 7.84$
$= 14.16$

Answer: $X_{max} = \mu + (z)(\sigma)$
$= 22 + (1.96)(4)$
$= 22 + 7.84$
$= 29.84$

UNIT III

DESCRIBING RELATIONSHIPS AND INFERENTIAL ANALYTICS

Correlation –Scatter plots –correlation coefficient for quantitative data –computational formula for correlation coefficient – Regression –regression line –least squares regression line – Standard error of estimate – interpretation of r2 –multiple regression equations –regression towards the mean - Hypothesis testing

3.1 CORRELATION

Correlation refers to a process for establishing the relationships between two variables. You learned a way to get a general idea about whether or not two variables are related, is to plot them on a "scatter plot". While there are many measures of association for variables which are measured at the ordinal or higher level of measurement, correlation is the most commonly used approach.

Types of Correlation

- **Positive Correlation** – when the values of the two variables move in the same direction so that an increase/decrease in the value of one variable is followed by an increase/decrease in the value of the other variable.
- **Negative Correlation** – when the values of the two variables move in the opposite direction so that an increase/decrease in the value of one variable is followed by decrease/increase in the value of the other variable.
- **No Correlation** – when there is no linear dependence or no relation between the two variables.

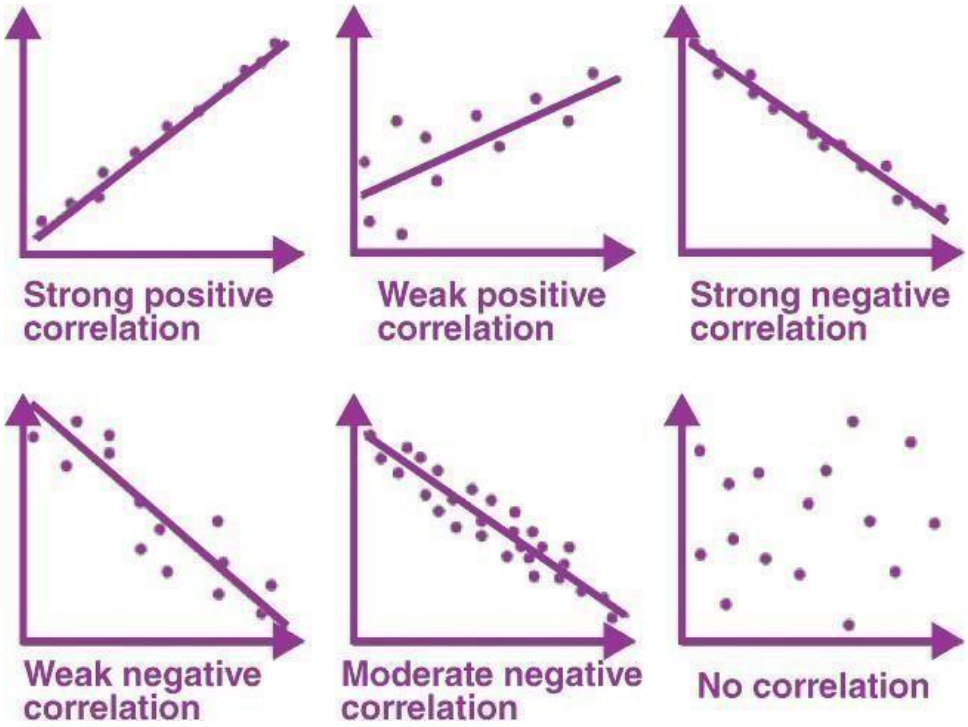


Table 6.2 THREE TYPES OF RELATIONSHIPS

A. POSITIVE RELATIONSHIP

FRIEND	SENT	RECEIVED
Doris	13	14
Steve	9	18
Mike	7	12
Andrea	5	10
John	1	6

B. NEGATIVE RELATIONSHIP

FRIEND	SENT	RECEIVED
Doris	13	6
Steve	9	10
Mike	7	14
Andrea	5	12
John	1	18

C. LITTLE OR NO RELATIONSHIP

FRIEND	SENT	RECEIVED
Doris	13	10
Steve	9	18
Mike	7	12
Andrea	5	6
John	1	14

3.2 SCATTERPLOTS

A scatter plot is a graph containing a cluster of dots that represents all pairs of scores. In other words Scatter plots are the graphs that present the relationship between two variables in a data-set. It represents data points on a two-dimensional plane or on a Cartesian system.

Construction of scatter plots

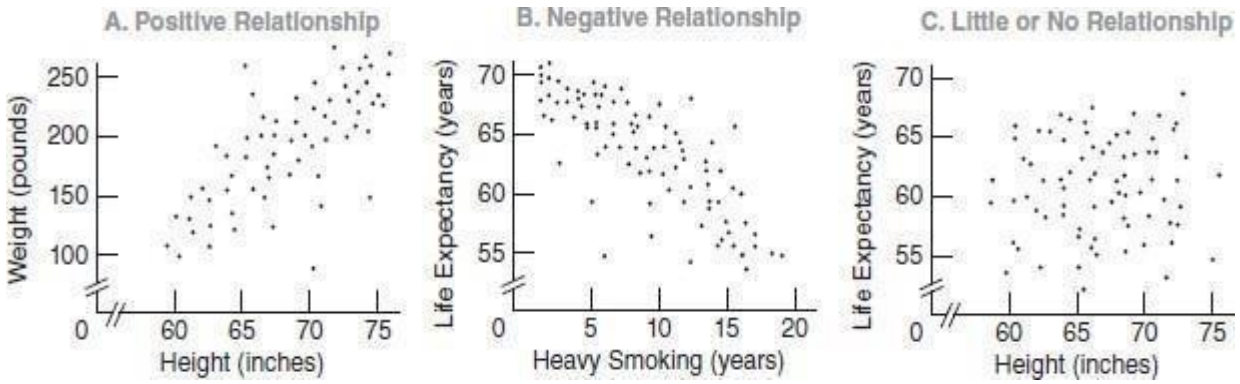
- The independent variable or attribute is plotted on the X-axis.
- The dependent variable is plotted on the Y-axis.
- Use each pair of scores to locate a dot within the scatter plot

Positive, Negative, or Little or No Relationship?

The first step is to note the tilt or slope, if any, of a dot cluster.
A dot cluster that has a slope from the lower left to the upper right, as in panel A of below figure reflects a
positive relationship.

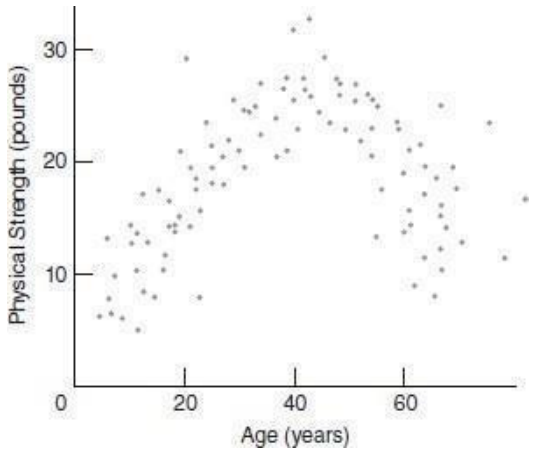
A dot cluster that has a slope from the upper left to the lower right, as in panel B of below figure reflects a
negative relationship.

A dot cluster that lacks any apparent slope, as in panel C of below figure reflects **little or no relationship.**



Perfect Relationship
A dot cluster that equals (rather than merely approximates) a straight line reflects a perfect relationship between two variables.

Curvilinear Relationship
The previous discussion assumes that a dot cluster approximates a straight line and, therefore, reflects a **linear relationship.** But this is not always the case. Sometimes a dot cluster approximates a bent or curved line, as in below figure, and therefore reflects a **curvilinear relationship.**



3.3 A CORRELATION COEFFICIENT FOR QUANTITATIVE DATA : r

The correlation coefficient, r, is a summary measure that describes the extent of the statistical relationship between two interval or ratio level variables.

Properties of r

- The correlation coefficient is scaled so that it is always between -1 and +1.
- When r is close to 0 this means that there is little relationship between the variables and the farther away from 0 r is, in either the positive or negative direction, the greater the relationship between the two variables.
- The sign of r indicates the type of linear relationship, whether positive or negative.
- The numerical value of r, without regard to sign, indicates the strength of the linear relationship.
- A number with a plus sign (or no sign) indicates a positive relationship, and a number with a minus sign indicates a negative relationship

3.4 COMPUTATION FORMULA FOR r

Calculate a value for r by using the following computation formula:

CORRELATION COEFFICIENT (COMPUTATION FORMULA)

$$r = \frac{SP_{xy}}{\sqrt{SS_x SS_y}}$$

Where the two sum of squares terms in the denominator are defined as

$$SS_x = \Sigma\left(X - \bar{X}\right)^2 = \Sigma X^2 - \frac{\left(\Sigma X\right)^2}{n}$$

$$SS_y = \Sigma\left(Y - \bar{Y}\right)^2 = \Sigma Y^2 - \frac{\left(\Sigma Y\right)^2}{n}$$

The sum of the products term in the numerator, SPxy, is defined in below formula

$$SP_{xy} = \Sigma\left(X - \bar{X}\right)\left(Y - \bar{Y}\right) = \Sigma XY - \frac{\left(\Sigma X\right)\left(\Sigma Y\right)}{n}$$

Or the formula is written as

$$r = \frac{n(\sum xy) - (\sum x)(\sum y)}{\sqrt{[n\sum x^2 - (\sum x)^2][n\sum y^2 - (\sum y)^2]}}$$

Where n = Number of Information

Σx = Total of the First Variable Value Σy = Total of the Second Variable Value

Σxy = Sum of the Product of first & Second Value Σx^2 = Sum of the Squares of the First Value

Σy^2 = Sum of the Squares of the Second Value

Table 6.3
CALCULATION OF *r*: COMPUTATION FORMULA

A. COMPUTATIONAL SEQUENCE

Assign a value to *n* (1), representing the number of pairs of scores.
Sum all scores for *X* (2) and for *Y* (3).
Find the product of each pair of *X* and *Y* scores (4), one at a time, then add all of these products (5).
Square each *X* score (6), one at a time, then add all squared *X* scores (7).
Square each *Y* score (8), one at a time, then add all squared *Y* scores (9).
Substitute numbers into formulas (10) and solve for SP_{xy}, SS_x, and SS_y
Substitute into formula (11) and solve for *r*.

B. DATA AND COMPUTATIONS

	CARDS		4	6	8
FRIEND	**SENT, *X***	**RECEIVED, *Y***	***XY***	***X*²**	***Y*²**
Doris	13	14	182	169	196
Steve	9	18	162	81	324
Mike	7	12	84	49	144
Andrea	5	10	50	25	100
John	1	6	6	1	36

1 $n = 5$ 2 $\Sigma X = 35$ 3 $\Sigma Y = 60$ 5 $\Sigma XY = 484$ 7 $\Sigma X^2 = 325$ 9 $\Sigma Y^2 = 800$

10 $$SP_{xy} = \Sigma XY - \frac{(\Sigma X)(\Sigma Y)}{n} = 484 - \frac{(35)(60)}{5} = 484 - 420 = 64$$

$$SS_x = \Sigma X^2 - \frac{(\Sigma X)^2}{n} = 325 - \frac{(35)^2}{5} = 325 - 245 = 80$$

$$SS_y = \Sigma Y^2 - \frac{(\Sigma Y)^2}{n} = 800 - \frac{(60)^2}{5} = 800 - 720 = 80$$

11 $$r = \frac{SP_{xy}}{\sqrt{SS_x SS_y}} = \frac{64}{\sqrt{(80)(80)}} = \frac{64}{80} = .80$$

3.5 REGRESSION

A regression is a statistical technique that relates a dependent variable to one or more independent (explanatory) variables. A regression model is able to show whether changes observed in the dependent variable are associated with changes in one or more of the explanatory variables.
Regression captures the correlation between variables observed in a data set, and quantifies whether those correlations are statistically significant or not.

A Regression Line

a regression line is a line that best describes the behaviour of a set of data. In other words, it‟s a line that best fits the trend of a given data.
The purpose of the line is to describe the interrelation of a dependent variable (Y variable) with one or many independent variables (X variable). By using the equation obtained from the regression line an analyst can forecast future behaviours of the dependent variable by inputting different values for the independent ones.

Types of regression

The two basic types of regression are

- Simple linear regression

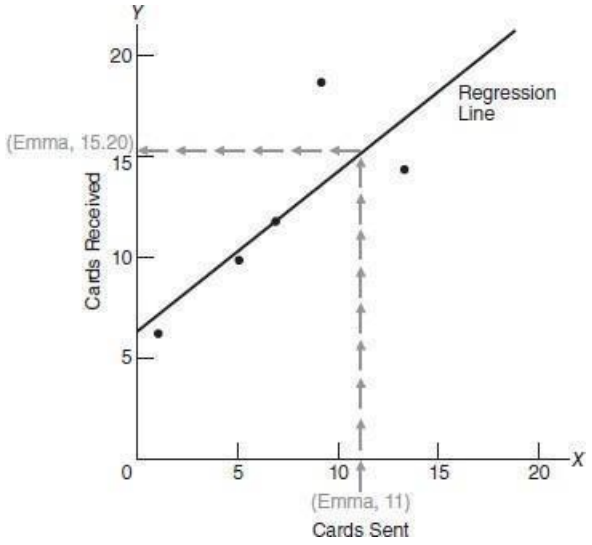
Simple linear regression uses one independent variable to explain or predict the outcome of the dependent variable Y

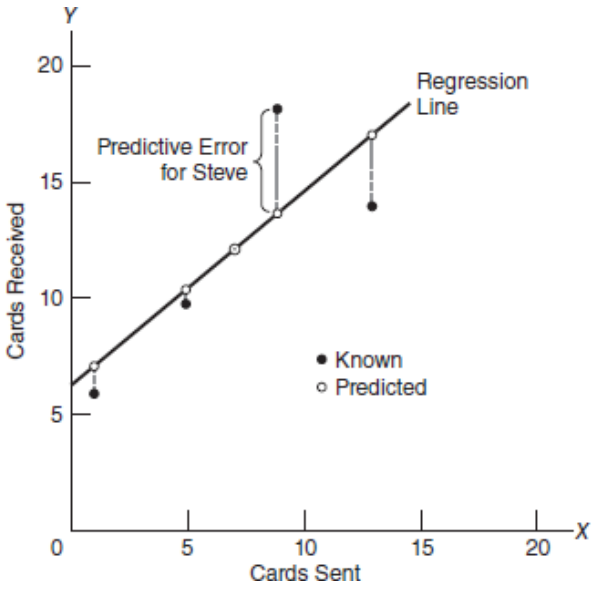
- Multiple linear regression

Multiple linear regressions use two or more independent variables to predict the outcome

Predictive Errors

Prediction error refers to the difference between the predicted values made by some model and the actual values.





3.6 LEAST SQUARES REGRESSION LINE

The placement of the regression line minimizes not the total predictive error but the total squared predictive error, that is, the total for all squared predictive errors. When located in this fashion, the regression line is often referred to as the least squares regression line.

The Least Squares Regression Line is the line that minimizes the sum of the residuals squared. The residual is the vertical distance between the observed point and the predicted point, and it is calculated by subtracting ˆy from y.

Formula

y‟‟ = bx+a b – slope , a – y intercept

$$b = \frac{N\Sigma(xy) - \Sigma x \Sigma y}{2 \quad 2}$$

$$b = \frac{\Sigma y - m \Sigma x}{N}$$

Example

"x"	"y"
2	4
3	5
5	7
7	10
9	15

Step 1: For each (x,y) calculate x^2 and xy:

X	**y**	**x^2**	**xy**
2	4	4	8
3	5	9	15
5	7	25	35
7	10	49	70
9	15	81	135

Step 2: Sum x, y, x2 and xy (gives us Σx, Σy, Σx2 and Σxy):

Σx: 26 Σy: 41 Σx^2: 168 Σxy: 263

Step 3: Calculate Slope b

$$b = \frac{N\Sigma(xy) - \Sigma x \Sigma y}{N\Sigma(x^2) - (\Sigma x)^2}$$

$$= \frac{5 \times 263 - 26 \times 41}{5 \times 168 - 262}$$

$$= \frac{1315 - 1066}{840 - 676}$$

$$= \frac{249}{164}$$

b = 1.5183.

Step 4: Calculate Intercept a $a = \frac{\Sigma y - b \Sigma x}{N}$

$$= \frac{41 - 1.5183 \times 26}{5}$$

a = 0.3049.

Step 5: $y^{\wedge} = bx+a$

$y^{\wedge} = 1.518x + 0.305$

x	y	y = 1.518x + 0.305	Error
2	4	3.34	−0.66
3	5	4.86	−0.14
5	7	7.89	0.89
7	10	10.93	0.93
9	15	13.97	−1.03

To predict the y value we can assume any value for x. Assume x = 8.
Then y = 1.518 x 8 + 0.305

= 12.45

3.7 STANDARD ERROR OF ESTIMATE ,s y | x

The standard error of the estimate is a measure of the accuracy of predictions. The regression line is the line that minimizes the sum of squared deviations of prediction (also called the sum of squares error), and the standard error of the estimate is the square root of the average squared deviation.

The standard error of estimate and symbolized as **s y | x**, this estimate of predictive error complies with the general format for any sample standard deviation, that is, the square root of a sum of squares term divided by its degrees of freedom.

$$SS_{y|x} = \sum(Y - Y')^2$$

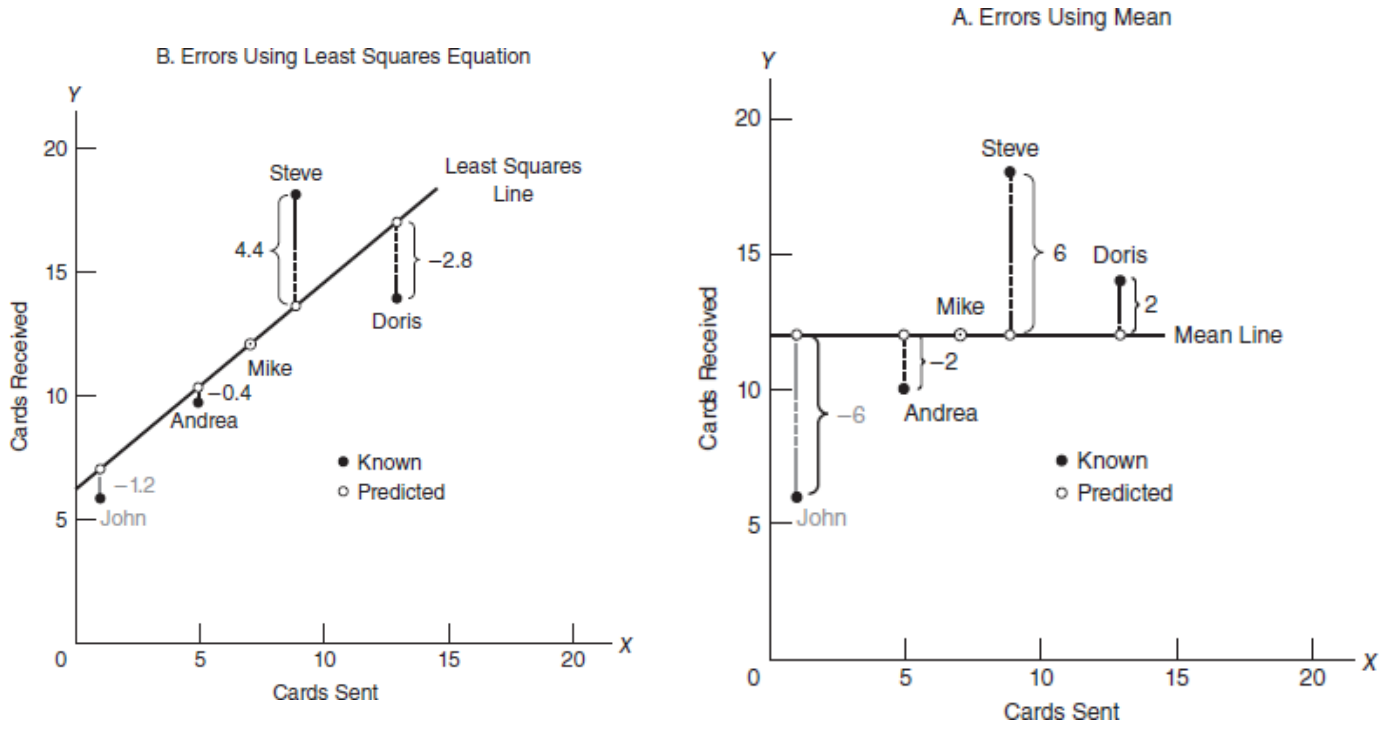


Fig. Predictive errors for five friends

Example

Calculate the standard error of estimate for the given X and Y values. X = 1,2,3,4,5 Y=2,4,5,4,5

Solution
Create five columns labeled x, y, y‟, y – y‟, (y – y‟)2 and N=5

x	y	x^2	xy	Y‟= bx+a	y-y‟	(y– y‟)^2
1	2	1	2	2.8	-0.8	0.64
2	4	4	8	3.4	0.6	0.36
3	5	9	15	4.0	1	1
4	4	16	16	4.6	-0.6	0.36
5	5	25	25	5.2	-0.2	0.04
Σx:15	**Σy:20**	**Σx^2:55**	**Σxy:66**			**Σ(y – y')^2 = 2.4**

Note: for finding b value we have to find xy and x^2, so add xy and x^2 column in table

$b = \frac{N\Sigma(xy) - \Sigma x \Sigma y}{N\Sigma(x^2) - (\Sigma x)^2}$

$b = \frac{5(66)-15x20}{5(55)-(15)^2}$

$= \frac{330 - 300}{275-225}$

b= 30/50 = 0.6

$a = \frac{\Sigma y - b \Sigma x}{N}$

$= \frac{20 - (0.6 \times 15)}{5}$

$= \frac{20 - 11}{5}$

a= 9/5 = 2.2

SSy/x = √((y-y')^2 / n-2)

=√(2.4/3)

SSy/x = 0.894

3.8 INTERPRETATION OF r 2

R-Squared (R^2 or the coefficient of determination) is a statistical measure in a regression model that determines the proportion of variance in the dependent variable that can be explained by the independent variable. In other words, r-squared shows how well the data fit the regression model (the goodness of fit).

R-squared can take any values between 0 to 1. Although the statistical measure provides some useful insights regarding the regression model, the user should not rely only on the measure in the assessment of a statistical model.

In addition, it does not indicate the correctness of the regression model. Therefore, the user should always draw conclusions about the model by analyzing r-squared together with the other variables in a statistical model.
The most common interpretation of r-squared is how well the regression model explains observed data.

$$r^2 = \frac{SS_{Y'}}{SS_Y} = \frac{SS_Y - SS_{Y|X}}{SS_Y}$$

$$SS_{y'} = \sum (Y' - \overline{Y})^2$$

3.9 MULTIPLE REGRESSION EQUATIONS

Multiple regression is a statistical technique applied on datasets dedicated to draw out a relationship between one response or dependent variable and multiple independent variables.
Multiple regression works by considering the values of the available multiple independent variables and predicting the value of one dependent variable.

Example:
A researcher decides to study students‟ performance from a school over a period of time. He observed that as the lectures proceed to operate online, the performance of students started to decline as well. The parameters for the dependent variable “decrease in performance” are various independent variables like “lack of attention, more internet addiction, neglecting studies” and much more.

Formula to find multiple regression
y = b1x1 + b2x2 + … bnxn + a

3.10 REGRESSION TOWARD THE MEAN

Regression toward the mean refers to a tendency for scores, particularly extreme scores, to shrink toward the mean.
In statistics, regression toward the mean (also called reversion to the mean, and reversion to mediocrity) is a concept that refers to the fact that if one sample of a random variable is extreme, the next sampling of the same random variable is likely to be closer to its mean.

Example
A military commander has two units return, one with 20% casualties and another with 50% casualties. He praises the first and berates the second. The next time, the two units return with the opposite results. From this experience, he “learns” that praise weakens performance and berating increases performance.

The Regression Fallacy
The regression fallacy is committed whenever regression toward the mean is interpreted as a real, rather than a chance, effect.

The regression fallacy can be avoided by splitting the subset of extreme observations into two groups

Table 7.4
REGRESSION TOWARD THE MEAN: BATTING AVERAGES OF TOP 10 HITTERS IN MAJOR LEAGUE BASEBALL DURING 2014 AND HOW THEY FARED DURING 2015

	BATTING AVERAGES*		
TOP 10 HITTERS (2014)	2014	2015	REGRESS TOWARD MEAN?
1. J. Alture	.341	.313	Yes
2. V. Martinez	.335	.282	Yes
3. M. Brantley	.327	.310	Yes
4. A. Beltre	.324	.287	Yes
5. J. Abreu	.317	.290	Yes
6. R. Cano	.314	.287	Yes
7. A. McCutchen	.314	.292	Yes
8. M. Cabrera	.313	.338	No
9. B. Posey	.311	.318	No
10. B. Revere	.306	.306	No

3.11 HYPOTHESIS TESTING

Hypothesis is usually considered as the principal instrument in research. The main goal in many research studies is to check whether the data collected support certain statements or predictions. A statistical hypothesis is an assertion or conjecture concerning one or more populations. Test of hypothesis is a process of testing of the significance regarding the parameters of the population on the basis of sample drawn from it. Thus, it is also termed as "Test of Significance".

In short, hypothesis testing enables us to make probability statements about population parameter. The hypothesis may not be proved absolutely, but in practice it is accepted if it has withstood a critical testing.

Points to be considered while formulating Hypothesis

- Hypothesis should be clear and precise.
- Hypothesis should be capable of being tested.
- Hypothesis should state relationship between variables.
- Hypothesis should be limited in scope and must be specific.
- Hypothesis should be stated as far as possible in most simple terms so that the same is easily understandable by all concerned.
- Hypothesis should be amenable to testing within a reasonable time.
- Hypothesis must explain empirical reference.

Types of Hypothesis:

There are two types of hypothesis, i.e., Research Hypothesis and Statistical Hypothesis

1. **Research Hypothesis:** A research hypothesis is a tentative solution for the problem being investigated. It is the supposition that motivates the researcher to accomplish future course of action. In research, the researcher determines whether or not their supposition can be supported through scientific

investigation.

2. **Statistical Hypothesis:** Statistical hypothesis is a statement about the population which we want to verify on the basis of sample taken from population. Statistical hypothesis is stated in such a way that they may be evaluated by appropriate statistical techniques.

Types of Statistical Hypotheses

There are two types of statistical hypotheses:

1. **Null Hypothesis (H0)** – A statistical hypothesis that states that there is no difference between a parameter and a specific value, or that there is no difference between two parameters.

2. **Alternative Hypothesis (H1 or Ha)** – A statistical hypothesis that states the existence of a difference between a parameter and a specific value, or states that there is a difference between two parameters. Alternative hypothesis is created in a negative meaning of the null hypothesis.

Suppose we want to test the hypothesis that the population mean (μ) is equal to the hypothesised mean (μ_{H0}) = 100. Then we would say that the null hypothesis is that the population mean is equal to the hypothesised mean 100 and symbolically we can express as:

$H0: \mu = \mu_{H0} = 100$

If our sample results do not support this null hypothesis, we should conclude that something else is true. What we conclude rejecting the null hypothesis is known as alternative hypothesis. In other words, the set of alternatives to the null hypothesis is referred to as the alternative hypothesis. If we accept H0, then we are rejecting H1 and if we reject H0, then we are accepting H1. For $H0: \mu = \mu_{Ho} = 100$, we may consider three possible alternative hypotheses as follows:

Alternative hypothesis	To be read as follows
$H1: \mu \neq \mu Ho$	(The alternative hypothesis is that the population n is not equal to 100 i.e., it may be more or less than
$H1: \mu > \mu Ho$	(The alternative hypothesis is that the population n is greater than 100)
$H1: \mu < \mu Ho$	(The alternative hypothesis is that the population n is less than 100)

The null hypothesis and the alternative hypothesis are chosen before the sample is drawn (the researcher must avoid the error of deriving hypotheses from the data that he/she collects and then testing the hypotheses from the same data). In the choice of null hypothesis, the following considerations are usually kept in view:

1. Alternative hypothesis is usually the one which one wishes to prove and the null hypothesis is the one which one wishes to disprove. Thus, a null hypothesis represents the hypothesis we are trying to reject, and alternative hypothesis represents all other possibilities.

2. Null hypotheses should always be specific hypothesis i.e., it should not state about or approximately a certain value.

3. In testing hypothesis**,** there are two possible outcomes:

- Reject H_0 and accept H_1 because of sufficient evidence in the sample in favour of H_1;
- Do not reject H_0 because of insufficient evidence to support H_1.

BASIC CONCEPTS CONCERNING TESTING OF HYPOTHESES

1. **The level of significance:** This is a very important concept in the context of hypothesis testing. It is always some percentage (usually 5%) which should be chosen with great care, thought and reason. In case we take the significance level at 5 per cent, then this implies that H_0 will be rejected when the sampling result (i.e., observed evidence) has a less than 0.05 probability of occurring if H_0 is true. In other words, the 5 per cent level of significance means that researcher is willing to take as much as a 5 per cent risk of rejecting the null hypothesis when it (H_0) happens to be true. Thus, the significance level is the maximum value of the probability of rejecting H_0 when it is true and is usually determined in advance before testing the hypothesis.

2. **Decision rule or Test of Hypothesis:** A decision rule is a procedure that the researcher uses to decide whether to accept or reject the null hypothesis. The decision rule is a statement that tells under what circumstances to reject the null hypothesis. The decision rule is based on specific values of the test statistic (e.g., reject H_0 if Calculated value > table value at the same level of significance)

3. **Types of Error:** In the context of testing of hypotheses, there are basically two types of errors we can make.

a. Type 1 error: To reject the null hypothesis when it is true is to make what is known as a type I error. The level at which a result is declared significant is known as the type I error rate, often denoted by α.

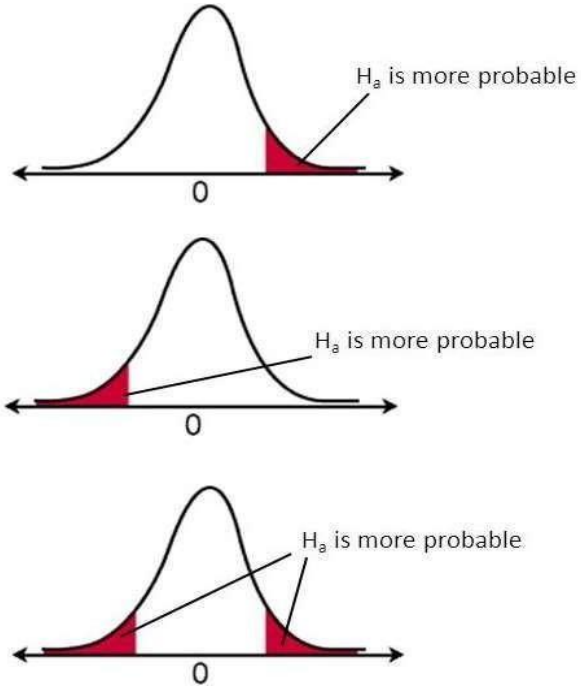
b. Type II error: If we do not reject the null hypothesis when in fact there is a difference between the groups, we make what is known as a type II error. The type II error rate is often denoted as β.

In a tabular form the said two errors can be presented as follows:

Particulars	Decision	
	Accept H0	**Reject H0**
H0 (True)	Correct Decision	Type I error (α error)
H0 (False)	Type II error (β error)	Correct decision

4. **One- tailed and Two-tailed Tests:** A test of statistical hypothesis, where the region of rejection is on only one side of the sampling distribution, is called a one tailed test. For example, suppose the null hypothesis states that the mean is less than or equal to 10. The alternative hypothesis would be that the mean is greater than 10. The region of rejection would consist of a range of numbers located on the right side of sampling distribution i.e., a set of numbers greater than 10.

A test of statistical hypothesis, where the region of rejection is on both sides of the sampling distribution, is called a two-tailed test. For example, suppose the null hypothesis states that the mean is equal to 10. The alternative hypothesis would be that the mean is less than 10 or greater than 10. The region of rejection would consist of a range of numbers located on both sides of sampling distribution; i.e., the region of rejection would consist partly of numbers that were less than 10 and partly of numbers that were greater than 10.



Right-tail test

$H_a: \mu > \text{value}$

Left-tail test

$H_a: \mu < \text{value}$

Two-tail test

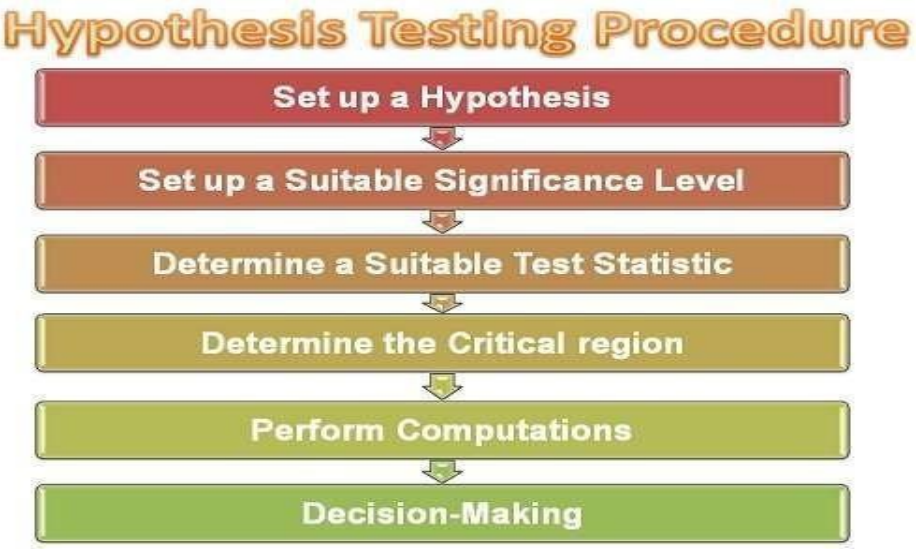
$H_a: \mu \neq \text{value}$

Procedure of Hypothesis Testing

Procedure for hypothesis testing refers to all those steps that we undertake for making a choice between the two actions i.e., rejection and acceptance of a null hypothesis. The various steps involved in hypothesis testing are stated below:

1. **Making a formal statement:** The step consists in making a formal statement of the null hypothesis (H0) and also of the alternative hypothesis (Ha or H1). This means that hypotheses should be clearly stated, considering the nature of the research problem.
2. **Selecting a significance level:** The hypotheses are tested on a pre-determined level of significance and as such the same should be specified. Generally, in practice, either 5% level or 1% level is adopted for the purpose.
3. **Deciding the distribution to use:** After deciding the level of significance, the next step in hypothesis testing is to determine the appropriate sampling distribution. The choice generally remains between normal distribution and the t-distribution.
4. **Selecting a random sample and computing an appropriate value:** Another step is to select a random sample(s) and compute an appropriate value from the sample data concerning the test statistic utilizing the relevant distribution. In other words, draw a sample to furnish empirical data.
5. **Calculation of the probability:** One has then to calculate the probability that the sample result would diverge as widely as it has from expectations, if the null hypothesis were in fact true.
6. **Comparing the probability and Decision making:** Yet another step consists in comparing the probability thus calculated with the specified value for α, the significance level. If the calculated probability is equal to or smaller than the α value in case of one-tailed test (and α /2 in case of two-tailed test), then reject the null hypothesis (i.e., accept the alternative hypothesis), but if the calculated probability is greater, then accept the null hypothesis.

The above stated general procedure for hypothesis testing can also be depicted in the form of a cart flow-



Tests of Hypotheses

Hypothesis testing determines the validity of the assumption (technically described as null hypothesis) with a view to choose between two conflicting hypotheses about the value of a population parameter. Hypothesis testing helps to decide on the basis of a sample data, whether a hypothesis about the population is likely to be true or false. Statisticians have developed several tests of hypotheses (also known as the tests of significance) for the purpose of testing of hypotheses which can be classified as:

a) Parametric tests or standard tests of hypotheses; and
b) Non-parametric tests or distribution-free test of hypotheses.

Parametric tests usually assume certain properties of the parent population from which we draw samples. Assumptions like observations come from a normal population, sample size is large, assumptions about the population parameters like mean, variance, etc., must hold good before parametric tests can be used. But there are situations when the researcher cannot or does not want to make such assumptions. In such situations we use statistical methods for testing hypotheses which are called non-parametric tests because such tests do not depend on any assumption about the parameters of the parent population. Besides, most non-parametric tests assume only nominal or ordinal data, whereas parametric tests require measurement equivalent to at least an interval scale. As a result, non-parametric tests need more observations than parametric tests to achieve the same size of Type I and Type II errors.

IMPORTANT PARAMETRIC TESTS

The important parametric tests are: (1) *z*-test; (2) *t*-test; and (3) *F*-test. All these tests are based on the assumption of normality i.e., the source of data is considered to be normally distributed.

1. **z- test**: It is based on the normal probability distribution and is used for judging the significance of several statistical measures, particularly the mean. This is a most frequently used test in research studies. This test is used even when binomial distribution or *t*-distribution is applicable on the presumption that such a distribution tends to approximate normal distribution as „*n*‟ becomes larger. *z*-test is generally used for comparing the mean of a sample to some hypothesised mean for the population in case of large sample, or when population variance is known. *z*-test is also used for judging he significance of difference between means of two independent samples incase of large samples, or when population variance is known. *z*-test is also used for comparing the sample proportion to a theoretical value of population proportion or for judging the difference in proportions of two independent samples when *n* happens to be large. Besides, this test may be used for judging the significance of median, mode, coefficient of correlation and several other measures.

2. **t- test:** It is based on t-distribution and is considered an appropriate test for judging the significance of a sample mean or for judging the significance of difference between the means of two

samples in case of small sample(s) when population variance is not known (in which case we use variance of the sample as an estimate of the population variance). In case two samples are related, we use paired t-test (or what is known as difference test) for judging the significance of the mean of difference between the two related samples. It can also be used for judging the significance of the coefficients of simple and partial correlations.

3. **F-test**: It is based on *F*-distribution and is used to compare the variance of the two-independent samples. This test is also used in the context of analysis of variance (ANOVA) for judging the significance of more than two sample means at one and the same time. It is also used for judging the significance of multiple correlation coefficients.

Non parametric Tests

Non parametric tests are used when the data isn't normal. Therefore, the key is to figure out if you have normally distributed data. The only non-parametric test you are likely to come across in elementary stats is the chi-square test. However, there are several others. For example: the Kruskal Willis test is the non-parametric alternative to the One-way ANOVA and the Mann Whitney is the non- parametric alternative to the two-sample t test.

Illustration1:

A sample of 400 male students is found to have a mean height 67.47 inches. Can it be reasonably regarded as a sample from a large population with mean height 67.39 inches and standard deviation 1.30 inches? Test at 5% level of significance.

Solution: Taking the null hypothesis that the mean height of the population is equal to 67.39 inches, we can write:

H0: μ H0 = 67.39"" H1: μ H0 $\neq$ 67.39""

and the given information as $\bar{X}$ = 67.47"", σp = 1.30″ , n = 400.

Assuming the population to be normal, we can work out the test statistic z as under:

Z-TEST

- Formula to find the value of Z (z-test) Is:

$$Z = \frac{\bar{x} - \mu_0}{\sigma / \sqrt{n}}$$

- $\bar{x}$ = mean of sample
- μ_0 = mean of population
- σ = standard deviation of population
- n = no. of observations

Hence, $Z = \frac{67.47–67.39}{1.30/\sqrt{400}} = \frac{0.08}{0.065} = 1.231$

As Ha is two-sided in the given question, we shall be applying a two-tailed test for determining the rejection regions at 5% level of significance which comes to as under, using normal curve area table:

R: | z | > 1.96

The observed value of z is 1.231 which is in the acceptance region since R: | z | > 1.96 and thus, H0 is accepted. We may conclude that the given sample (with mean height = 67.47") can be regarded to have been taken from a population with mean height 67.39" and standard deviation 1.30" at 5% level of significance.

Illustration 2.

Suppose we are interested in a population of 20 industrial units of the same size, all of which are experiencing excessive labour turnover problems. The past records show that the mean of the distribution of annual turnover is 320 employees, with a standard deviation of 75 employees. A sample of 5 of these industrial units is taken at random which gives a mean of annual turnover as 300 employees. Is the sample mean consistent with the population mean? Test at 5% level.

Solution: Taking the null hypothesis that the population mean is 320 employees, we can write:

$$H_0 : \mu_{H_0} = 320 \text{ employees}$$

$$H_a : \mu_{H_0} \neq 320 \text{ employees}$$

and the given information as under:

$$\bar{X} = 300 \text{ employees}, \ \sigma_p = 75 \text{ employees}$$

$$n = 5; \ N = 20$$

Assuming the population to be normal, we can work out the test statistic z as under:

$$z^* = \frac{\bar{X} - \mu_{H_0}}{\sigma_p/\sqrt{n} \times \sqrt{(N-n)/(N-1)}}$$

$$= \frac{300 - 320}{75/\sqrt{5} \times \sqrt{(20-5)/(20-1)}} = -\frac{20}{(33.54)\ (.888)}$$

$$= -0.67$$

As H_a is two-sided in the given question, we shall apply a two-tailed test for determining the rejection regions at 5% level of significance which comes to as under, using normal curve area table:

$$R : |z| > 1.96$$

The observed value of z is –0.67 which is in the acceptance region since $R : |z| > 1.96$ and thus, H_0 is accepted and we may conclude that the sample mean is consistent with population mean i.e., the population mean 320 is supported by sample results.

Illustration: 3

Raju Restaurant near the railway station at Falna has been having average sales of 500 tea cups per day. Because of the development of bus stand nearby, it expects to increase its sales. During the first 12 days after the start of the bus stand, the daily sales were as under:

550, 570, 490, 615, 505, 580, 570, 460, 600, 580, 530, 526

On the basis of this sample information, can one conclude that Raju Restaurant's sales have increased? Use 5 per cent level of significance.

Solution: Taking the null hypothesis that sales average 500 tea cups per day and they have not increased unless proved, we can write:

$H_0 : \mu = 500$ cups per day

$H_a : \mu > 500$ (as we want to conclude that sales have increased).

As the sample size is small and the population standard deviation is not known, we shall use t-test assuming normal population and shall work out the test statistic t as:

$$t = \frac{\bar{X} - \mu}{\sigma_s/\sqrt{n}}$$

(To find $\bar{X}$ and σ_s we make the following computations:)

$$\therefore \quad \bar{X} = \frac{\sum X_i}{n} = \frac{6576}{12} = 548$$

and
$$\sigma_s = \sqrt{\frac{\sum (X_i - \bar{X})^2}{n-1}} = \sqrt{\frac{23978}{12-1}} = 46.68$$

Hence,
$$t = \frac{548 - 500}{46.68/\sqrt{12}} = \frac{48}{13.49} = 3.558$$

Degree of freedom = $n - 1 = 12 - 1 = 11$

As H_a is one-sided, we shall determine the rejection region applying one-tailed test (in the right tail because H_a is of more than type) at 5 per cent level of significance and it comes to as under, using table of *t*-distribution for 11 degrees of freedom:

$$R : t > 1.796$$

The observed value of *t* is 3.558 which is in the rejection region and thus H_0 is rejected at 5 per cent level of significance and we can conclude that the sample data indicate that Raju restaurant's sales have increased.

S. No.	X_i	$(X_i - \bar{X})$	$(X_i - \bar{X})^2$
1	550	2	4
2	570	22	484
3	490	–58	3364
4	615	67	4489
5	505	–43	1849
6	580	32	1024
7	570	22	484
8	460	–88	7744
9	600	52	2704
10	580	32	1024
11	530	–18	324
12	526	–22	484
$n = 10$	$\sum X_i = 6576$		$\sum (X_i - \bar{X})^2 = 23978$

Illustration: 4

Sample of sales in similar shops in two towns are taken for a new product with the following results:

Town	Mean sales	Variance	Size of sample
A	57	5.3	5
B	61	4.8	7

Is there any evidence of difference in sales in the two towns? Use 5 per cent level of significance for testing this difference between the means of two samples.

Solution: Taking the null hypothesis that the means of two populations do not differ we can write:

$$H_0 : \mu_1 = \mu_2$$
$$H_a : \mu_1 \neq \mu_2$$

and the given information as follows:

Sample from town A as sample one	$\bar{X}_1 = 57$	$\sigma_{s_1}^2 = 5.3$	$n_1 = 5$
Sample from town B As sample two	$\bar{X}_2 = 61$	$\sigma_{s_2}^2 = 4.8$	$n_2 = 7$

Since in the given question variances of the population are not known and the size of samples is small, we shall use *t*-test for difference in means, assuming the populations to be normal and can work out the test statistic *t* as under:

$$t = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{(n_1-1)\sigma_{s_1}^2 + (n_2-1)\sigma_{s_2}^2}{n_1+n_2-2}} \times \sqrt{\frac{1}{n_1}+\frac{1}{n_2}}}$$

with d.f. $= (n_1 + n_2 - 2)$

$$= \frac{57-61}{\sqrt{\frac{4(5.3)+6(4.8)}{5+7-2}} \times \sqrt{\frac{1}{5}+\frac{1}{7}}} = -3.053$$

Degrees of freedom $= (n_1 + n_2 - 2) = 5 + 7 - 2 = 10$

As H_a is two-sided, we shall apply a two-tailed test for determining the rejection regions at 5 per cent level which come to as under, using table of *t*-distribution for 10 degrees of freedom:

$$R : |t| > 2.228$$

The observed value of *t* is – 3.053 which falls in the rejection region and thus, we reject H_0 and conclude that the difference in sales in the two towns is significant at 5 per cent level.

Limitations of the Test of Hypotheses

- Test do not explain the reasons as to why does the difference exist, say between the means of the two samples. They simply indicate whether the difference is due to fluctuations of sampling or because of other reasons but the tests do not tell us as to which is/are the other reason(s) causing the difference.
- Results of significance tests are based on probabilities and as such cannot be expressed with full certainty.
- Statistical inferences based on the significance tests cannot be said to be entirely correct evidences concerning the truth of the hypotheses.

UNIT IV

PYTHON LIBRARIES FOR DATA WRANGLING

Basics of Numpy arrays –aggregations –computations on arrays –comparisons, masks, boolean logic – fancy indexing – structured arrays – Data manipulation with Pandas – data indexing and selection – operating on data – missing data – Hierarchical indexing – combining datasets – aggregation and grouping – pivot tables

4.1 BASICS OF NUMPY ARRAYS

NumPy (short for Numerical Python) provides an efficient interface to store and operate on dense data buffers. NumPy arrays are like Python's built-in list type, but NumPy arrays provide much more efficient storage and data operations as the arrays grow larger in size.

Attributes of arrays
Determining the size, shape, memory consumption, and data types of arrays
Indexing of arrays
Getting and setting the value of individual array elements
Slicing of arrays
Getting and setting smaller subarrays within a larger array
Reshaping of arrays
Changing the shape of a given array
Joining and splitting of arrays
Combining multiple arrays into one, and splitting one array into many

NumPy Array Attributes

- ndim (the number of dimensions),
- shape (the size of each dimension)
- size (the total size of the array)

Example

```
np.random.seed(0) # seed for reproducibility
x1 = np.random.randint(10, size=6) # One-dimensional array
x2 = np.random.randint(10, size=(3, 4)) # Two-dimensional array
x3 = np.random.randint(10, size=(3, 4, 5)) # Three-dimensional array
print("x3 ndim: ", x3.ndim) print("x3 shape:", x3.shape) print("x3 size: ", x3.size)
print("dtype:", x3.dtype)
print("itemsize:", x3.itemsize, "bytes") print("nbytes:", x3.nbytes, "bytes")
```

Array Indexing:

- Accessing Single Elements

Accessing Single Elements

- Indexing in NumPy will feel quite familiar like list indexing,
- In a one-dimensional array, you can access the ith value (counting from zero) by specifying the desired index in square brackets, just as with Python lists
- To index from the end of the array, you can use negative indices
- In a multidimensional array, you access items using a comma-separated tuple of indices
- Unlike Python lists, NumPy arrays have a fixed type. This means, for example, that if you attempt to insert a floating-point value to an integer array, the value will be silently truncated.

Array Slicing: Accessing Subarrays

Just as we can use square brackets to access individual array elements, we can also use them to access subarrays with the slice notation, marked by the colon (:) character.
The NumPy slicing syntax follows that of the standard Python list; to access a slice of an array x, use this:

x[start:stop:step]
start – starting array index
stop – array index to stop (last value will not be considered) step – terms has to be printed from start to stop
Default to the values start=0, stop=size of dimension, step=1.

Example
x = np.arange(10) x
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

x[:5] *# prints first five elements*
array([0, 1, 2, 3, 4])

x[5:] *# elements after index 5*
array([5, 6, 7, 8, 9])

x[4:7] *# middle subarray(from 4th index to 6th index)*
array([4, 5, 6])

While using negative indices the defaults for start and stop are swapped. This becomes a convenient way to reverse an array
x[::-1] *# all elements, reversed*
array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])

x[5::-2] *# reversed every other from index 5*
array([5, 3, 1])

Multidimensional sub arrays

Multidimensional slices work in the same way, with multiple slices separated by commas. For example:
x2
array([[12, 5, 2, 4],
[7, 6, 8, 8],
[1, 6, 7, 7]])
x2[:2, :3] # two rows, three columns array([[12, 5, 2],
[7, 6, 8]])
x2[:3, ::2] # all rows, every other column(every second column) array([[12, 2],
[7, 8],
[1, 7]])
Finally, sub array dimensions can even be reversed together
x2[::-1, ::-1]
array([[7, 7, 6, 1],
[8, 8, 6, 7],
[4, 2, 5, 12]])

Reshaping of Arrays

The most flexible way of doing this is with the **reshape()** method. For example, if you want to put the numbers 1 through 9 in a 3×3 grid, you can do the following

```
grid = np.arange(1, 10).reshape((3, 3)) print(grid)
[[1 2 3]
[4 5 6]
[7 8 9]]
```

Array Concatenation and Splitting

Concatenation of arrays

Concatenation, or joining of two arrays in NumPy, is primarily accomplished through the routines **np.concatenate, np.vstack,** and **np.hstack.** np.concatenate takes a tuple or list of arrays as its first argument.

```
x = np.array([1, 2, 3])
y = np.array([3, 2, 1]) np.concatenate([x, y])

array([1, 2, 3, 3, 2, 1])
```

You can also concatenate more than two arrays at once

```
z = [99, 99, 99]
print(np.concatenate([x, y, z])) [ 1 2 3 3 2 1 99 99 99]
```

np.concatenate can also be used for two-dimensional arrays

```
grid = np.array([[1, 2, 3],
[4, 5, 6]])
np.concatenate([grid, grid])

array([[1, 2, 3],
[4, 5, 6],
[1, 2, 3],
[4, 5, 6]])
```

Concatenate along the second axis (zero-indexed)

```
np.concatenate([grid, grid], axis=1)
array([[1, 2, 3, 1, 2, 3],
[4, 5, 6, 4, 5, 6]])
```

np.vstack (vertical stack) functions

```
x = np.array([1, 2, 3])
grid = np.array([[9, 8, 7],
[6, 5, 4]])
p.vstack([x, grid])
array([[1, 2, 3],
[9, 8, 7],
[6, 5, 4]])
```

np.hstack (horizontal stack) functions

```
y = np.array([[99],
[99]])
np.hstack([grid, y])
```

```
array([[ 9, 8, 7, 99],
[ 6, 5, 4, 99]])
```

Splitting of arrays

The opposite of concatenation is splitting, which is implemented by the functions **np.split, np.hsplit**, and **np.vsplit**. For each of these, we can pass a list of indices giving the split points

```
x = [1, 2, 3, 99, 99, 3, 2, 1]
x1, x2, x3 = np.split(x, [3, 5]) print(x1, x2, x3)
```

```
[1 2 3] [99 99] [3 2 1]
```

Notice that N split points lead to N + 1 subarrays. The related functions np.hsplit and np.vsplit are similar

```
grid = np.arange(16).reshape((4, 4)) grid
array([[ 0, 1, 2, 3],
[ 4, 5, 6, 7],
[ 8, 9, 10, 11],
[12, 13, 14, 15]])

upper, lower = np.vsplit(grid, [2])
print(upper) print(lower)
```

```
[[0 1 2 3]
[4 5 6 7]]
[[ 8 9 10 11]
[12 13 14 15]]

left, right = np.hsplit(grid, [2])
print(left) print(right)
```

```
[[ 0 1]
[ 4 5]
[ 8 9]
[12 13]]
[[ 2 3]
[ 6 7]
[10 11]
[14 15]]
```

Computation on NumPy Arrays: Universal Functions

Introducing UFuncs

NumPy provides a convenient interface into just this kind of statically typed, compiled routine. This is known as a vectorized operation.

Vectorized operations in NumPy are implemented via ufuncs, whose main purpose is to quickly execute repeated operations on values in NumPy arrays. Ufuncs are extremely flexible—before we saw an operation between a scalar and an array, but we can also operate between two arrays

Exploring NumPy's UFuncs

Ufuncs exist in two flavors: unary ufuncs, which operate on a single input, and binary ufuncs, which operate on two inputs. We'll see examples of both these types of functions here.

Array arithmetic

NumPy‘s ufuncs make use of Python‘s native arithmetic operators. The standard addition, subtraction, multiplication, and division can all be used.

x = np.arange(4) **print**("x =", x) **print**("x + 5 =", x + 5) **print**("x - 5 =", x - 5) **print**("x * 2 =", x * 2)

Operator	Equivalent ufunc	Description
+	np.add	Addition (e.g., 1 + 1 = 2)
-	np.subtract	Subtraction (e.g., 3 - 2 = 1)
-	np.negative	Unary negation (e.g., -2)
*	np.multiply	Multiplication (e.g., 2 * 3 = 6)
/	np.divide	Division (e.g., 3 / 2 = 1.5)
/ /	np.floor_divide	Floor division (e.g., 3 // 2 = 1)
* *	np.power	Exponentiation (e.g., 2 ** 3 = 8)
%	np.mod	Modulus/remainder (e.g., 9 % 4 = 1)

Absolute value

Just as NumPy understands Python‘s built-in arithmetic operators, it also understands Python‘s built-in absolute value function.

- np.abs()
- np.absolute()

x = np.array([-2, -1, 0, 1, 2]) abs(x)

array([2, 1, 0, 1, 2])

The corresponding NumPy ufunc is np.absolute, which is also available under the alias np.abs
np.absolute(x) array([2, 1, 0, 1, 2])

np.abs(x)
array([2, 1, 0, 1, 2])

Trigonometric functions

NumPy provides a large number of useful ufuncs, and some of the most useful for the data scientist are the trigonometric functions.

- np.sin()
- np.cos()
- np.tan()

inverse trigonometric functions

- np.arcsin()
- np.arccos()
- np.arctan()

Defining an array of angles: theta = np.linspace(0, np.pi, 3)
Compute some trigonometric functions like
print("theta = ", theta) **print**("sin(theta) = ", np.sin(theta)) **print**("cos(theta) = ", np.cos(theta))
print("tan(theta) = ", np.tan(theta))

Exponents and logarithms

Another common type of operation available in a NumPy ufunc are the exponentials.

- np.exp(x) – calculate exponent of all elements in the input array ie e^x (e=2.7182)
- np.exp2(x) – calculate 2**x for all x being the array elements
- np.power(x,y) – calculates the power as x^y

```
x = [1, 2, 3]
print("x =", x)
print("e^x =", np.exp(x))
print("2^x =", np.exp2(x))
print("3^x =", np.power(3, x))
```

The inverse of the exponentials, the logarithms, are also available. The basic np.log gives the natural logarithm; if you prefer to compute the base-2 logarithm or the base-10 logarithm as .

- np.log(x) - is a mathematical function that helps user to calculate Natural logarithm of x where x belongs to all the input array elements
- np.log2(x) - to calculate Base-2 logarithm of x
- np.log10(x) - to calculate Base-10 logarithm of x

```
x = [1, 2, 4, 10]
print("x =", x)
print("ln(x) =", np.log(x))
print("log2(x) =", np.log2(x))
print("log10(x) =", np.log10(x))
```

Specialized ufuncs

NumPy has many more ufuncs available like

- Hyperbolic trig functions,
- Bitwise arithmetic,
- Comparison operators,
- Conversions from radians to degrees,
- Rounding and remainders, and much more

More specialized and obscure ufuncs is the submodule scipy.special. If you want to compute some obscure mathematical function on your data, chances are it is implemented in scipy.special.

- Gamma function

Advanced Ufunc Features Specifying output

Rather than creating a temporary array, you can use this to write computation results directly to the memory location where you'd like them to be. For all ufuncs, you can do this using the out argument of the function.

```
x = np.arange(5) y = np.empty(5)
np.multiply(x, 10, out=y)


print(y)
```

```
[ 0. 10. 20. 30. 40.]
```

4.2 AGGREGATES

To reduce an array with a particular operation, we can use the reduce method of any ufunc. A reduce repeatedly applies a given operation to the elements of an array until only a single result remains.

x = np.arange(1, 6) np.add.reduce(x)
Similarly, calling reduce on the multiply ufunc results in the product of all array elements
np.multiply.reduce(x)
120
If we'd like to store all the intermediate results of the computation, we can instead use Accumulate
np.add.accumulate(x) array([1, 3, 6, 10, 15])

Outer products
ufunc can compute the output of all pairs of two different inputs using the outer method. This allows you, in one line, to do things like create a multiplication table.

x = np.arange(1, 6) np.multiply.outer(x, x)

array([[1, 2, 3, 4, 5],
[2, 4, 6, 8, 10],
[3, 6, 9, 12, 15],
[4, 8, 12, 16, 20],
[5, 10, 15, 20, 25]])

Aggregations: Min, Max, and Everything in Between
Minimum and Maximum

```
[[ 0.8967576  0.03783739   0.75952519   0.06682827]
[ 0.8354065                0.19544769   0.43447084]
[ 0.66859307  0.15038721   0.37911423   0.6687194 ]]
```

Python has built-in min and max functions, used to find the minimum value and maximum value of any given array.

For min, max, sum, and several other NumPy aggregates, a shorter syntax is to use methods of the array object itself.

- np.min() – finds the minimum (smallest) value in the array
- np.max() – finds the maximum (largest) value in the array Example

x=[1,2,3,4]
np.min(x) 1
np.max(x) 4

Multidimensional aggregates
One common type of aggregation operation is an aggregate along a row or column.
By default, each NumPy aggregation function will return the aggregate over the entire array. ie. If we use the np.sum() it will calculates the sum of all elements of the array.
Example
m = np.random.random((3, 4))
print(M)

M.sum() 6.0850555667307118

Aggregation functions take an additional argument specifying the axis along which the aggregate is computed. The axis normally takes either 0 or 1. if the axis = 0 then it runs along with columns, if axis =1 it runs along with rows.

Example
We can find the minimum value within each column by specifying axis=0

M.min(axis=0)
array([0.66859307, 0.03783739, 0.19544769, 0.06682827])

Similarly, we can find the maximum value within each row M.max(axis=1)
array([0.8967576 , 0.99196818, 0.6687194])

Other aggregation functions
NumPy provides many other aggregation functions most aggregates have a NaN-safe counterpart that computes the result while ignoring missing values, which are marked by the special IEEE floating-point NaN value.

Function Name	NaN-safe Version	Description
np.sum	np.nansum	Compute sum of elements
np.prod	np.nanprod	Compute product of elements
np.mean	np.nanmean	Compute median of elements
np.std	np.nanstd	Compute standard deviation
np.var	np.nanvar	Compute variance
np.min	np.nanmin	Find minimum value
np.max	np.nanmax	Find maximum value
np.argmin	np.nanargmin	Find index of minimum value
np.argmax	np.nanargmax	Find index of maximum value
np.median	np.nanmedian	Compute median of elements
np.percentile	np.nanpercentile	Compute rank-based statistics of elements
np.any	N/A	Evaluate whether any elements are true
np.all	N/A	Evaluate whether all elements are true

4.3 COMPUTATION ON ARRAYS: Broadcasting

Broadcasting is simply a set of rules for applying binary ufuncs (addition, subtraction, multiplication, etc.) on arrays of different sizes.
For arrays of the same size, binary operations are performed on an element-by-element basis.
a = np.array([0, 1, 2])
b = np.array([5, 5, 5]) a + b

array([5, 6, 7])
Broadcasting allows these types of binary operations to be performed on arrays of different sizes. a + 5
array([5, 6, 7])

We can think of this as an operation that stretches or duplicates the value 5 into the array [5, 5, 5], and adds the results. The advantage of NumPy‘s broadcasting is that this duplication of values does not

actually take place.

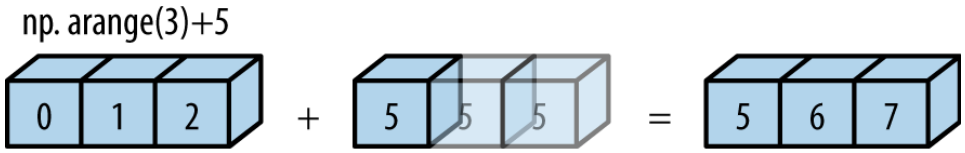
We can similarly extend this to arrays of higher dimension. Observe the result when we add a one-dimensional array to a two-dimensional array.

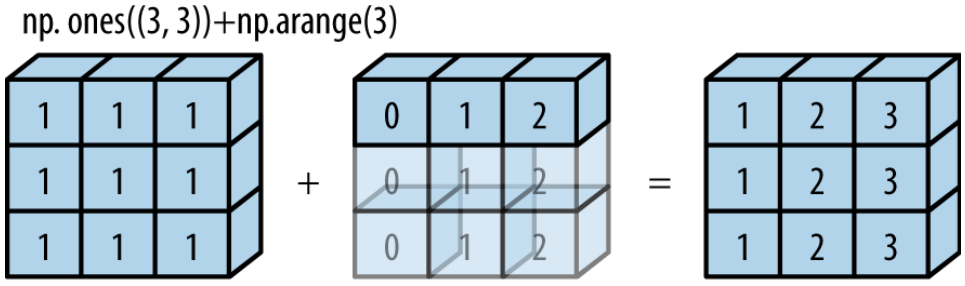
Example
M = np.ones((3, 3)) M

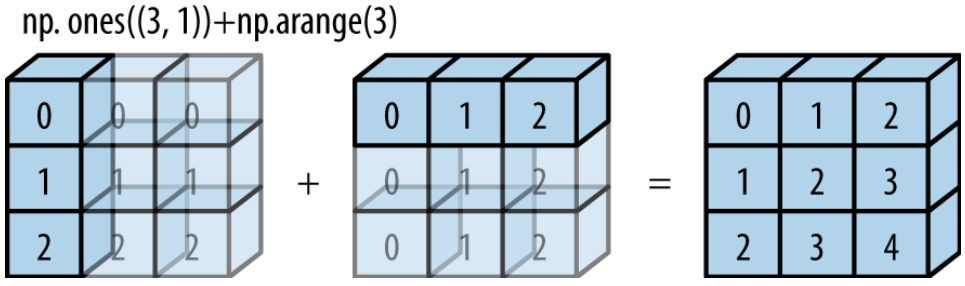
array([[1., 1., 1.],
[1., 1., 1.],
[1., 1., 1.]])

M + a
array([[1., 2., 3.],
[1., 2., 3.],
[1., 2., 3.]])
Here the one-dimensional array a is stretched, or broadcast, across the second dimension in order to match the shape of M.

Just as before we stretched or broadcasted one value to match the shape of the other, here we‘ve stretched both a and b to match a common shape, and the result is a two dimensional array.







The light boxes represent the broadcasted values: again, this extra memory is not actually allocated in the course of the operation, but it can be useful conceptually to imagine that it is.

Rules of Broadcasting

Broadcasting in NumPy follows a strict set of rules to determine the interaction between the two arrays.

- Rule 1: If the two arrays differ in their number of dimensions, the shape of the one with fewer dimensions is padded with ones on its leading (left) side.
- Rule 2: If the shape of the two arrays does not match in any dimension, the array with shape equal to 1 in that dimension is stretched to match the other shape.
- Rule 3: If in any dimension the sizes disagree and neither is equal to 1, an error is raised.

Broadcasting example 1

Let‘s look at adding a two-dimensional array to a one-dimensional array:

M = np.ones((2, 3)) a = np.arange(3)

Let‘s consider an operation on these two arrays. The shapes of the arrays are:
M.shape = (2, 3)
a.shape = (3,)
We see by rule 1 that the array a has fewer dimensions, so we pad it on the left with ones:
M.shape -> (2, 3)
a.shape -> (1, 3)
By rule 2, we now see that the first dimension disagrees, so we stretch this dimension to match:
M.shape -> (2, 3)
a.shape -> (2, 3)
The shapes match, and we see that the final shape will be (2, 3):
M + a
array([[1., 2., 3.],
[1., 2., 3.]])

Broadcasting example 2

Let‘s take a look at an example where both arrays need to be broadcast:
a = np.arange(3).reshape((3, 1))
b = np.arange(3)

Again, we‘ll start by writing out the shape of the arrays: a.shape = (3, 1)
b.shape = (3,)

Rule 1 says we must pad the shape of b with ones: a.shape -> (3, 1)
b.shape -> (1, 3)

And rule 2 tells us that we upgrade each of these ones to match the corresponding size of the other array:
a.shape -> (3, 3)
b.shape -> (3, 3)

Because the result matches, these shapes are compatible. We can see this here: a + b
array([[0, 1, 2],
[1, 2, 3],
[2, 3, 4]])

4.4 COMPARISONS, MASKS, AND BOOLEAN LOGIC

Comparison Operators as ufuncs.
We saw that using +, -, *, /, and others on arrays leads to element-wise operations. NumPy also implements comparison operators such as < (less than) and > (greater than) as element-wise ufuncs. The result of these comparison operators is always an array with a Boolean data type. All six of the standard comparison operations are available:

x = np.array([1, 2, 3, 4, 5]) x < 3 *# less than*

array([True, True, False, False, False], dtype=bool) x > 3 *# greater than*
array([False, False, False, True, True], dtype=bool)
x <= 3 *# less than or equal*
array([True, True, True, False, False], dtype=bool)
x >= 3 *# greater than or equal*
array([False, False, True, True, True], dtype=bool)
x != 3 *# not equal*
array([True, True, False, True, True], dtype=bool)
x == 3 *# equal*
array([False, False, True, False, False], dtype=bool)

Operator	Equivalent ufunc
==	np.equal
!=	np.not_equal
<	np.less
<=	np.less_equal
>	np.greater
>=	np.greater_equal

Just as in the case of arithmetic ufuncs, these will work on arrays of any size and shape. Here is a two- dimensional example

rng = np.random.RandomState(0) x = rng.randint(10, size=(3, 4))
x

array([[5, 0, 3, 3],
[7, 9, 3, 5],
[2, 4, 7, 6]])

x < 6

array([[True, True, True, True], [False, False, True, True],
[True, True, False, False]], dtype=bool)

The result is a Boolean array, and NumPy provides a number of straightforward patterns for working with these Boolean results.

Working with Boolean Arrays

- np.count_nonzero()
- np.sum()
- np.sum(x , axis)
- np.any()
- np.all()
- np.all(x , axis)

Boolean operators

Operator	Equivalent ufunc
&	np.bitwise_and
|	np.bitwise_or
^	np.bitwise_xor
~	np.bitwise_not

Example

np.sum((inches > 0.5) & (inches < 1)) inches > (0.5 & inches) < 1
np.sum(~((inches <= 0.5) | (inches >= 1)))

Boolean Arrays as Masks

A more powerful pattern is to use Boolean arrays as masks, to select particular subsets of the data themselves. Returning to our x array from before, suppose we want an array of all values in the array that are less than, say, 5
We can obtain a Boolean array for this condition easily, as we've already seen Example
x
array([[5, 0, 3, 3],
[7, 9, 3, 5],
[2, 4, 7, 6]])

x < 5
array([[False, True, True, True], [False, False, True, False],
[True, True, False, False]], dtype=bool)

Masking operation

To select these values from the array, we can simply index on this Boolean array; this is known as a masking operation.
x[x < 5]

array([0, 3, 3, 3, 2, 4])
What is returned is a one-dimensional array filled with all the values that meet this condition; in other words, all the values in positions at which the mask array is True.

4.5 FANCY INDEXING

Fancy indexing is like the simple indexing we've already seen, but we pass arrays of indices in place of single scalars. This allows us to very quickly access and modify complicated subsets of an array's values.

Exploring Fancy Indexing

Fancy indexing is conceptually simple: it means passing an array of indices to access multiple array

elements at once.

Types of fancy indexing.

- Indexing / accessing more values
- Array of indices
- In multi dimensional
- Standard indexing

Example

import numpy as np

rand = np.random.RandomState(42) x = rand.randint(100, size=10) **print**(x)

[51 92 14 71 60 20 82 86 74 74]

Indexing / accessing more values

Suppose we want to access three different elements. We could do it like this: [x[3], x[7], x[2]]

[71, 86, 14]

Array of indices

We can pass a single list or array of indices to obtain the same result. ind = [3, 7, 4]
x[ind]

array([71, 86, 60])

In multi dimensional

Fancy indexing also works in multiple dimensions. Consider the following array.
X = np.arange(12).reshape((3, 4)) X

array([[0, 1, 2, 3],
[4, 5, 6, 7],
[8, 9, 10, 11]])

Standard indexing

Like with standard indexing, the first index refers to the row, and the second to the column. row = np.array([0, 1, 2])
col = np.array([2, 1, 3])
X[row, col] array([2, 5, 11])

Combined Indexing

For even more powerful operations, fancy indexing can be combined with the other indexing schemes we‘ve seen.
Example array
print(X)
[[0 1 2 3]
[4 5 6 7]
[8 9 10 11]]

- Combine fancy and simple indices

X[2, [2, 0, 1]]
array([10, 8, 9])

- Combine fancy indexing with slicing

```
X[1:, [2, 0, 1]]
array([[ 6, 4, 5],
[10, 8, 9]])
```

- Combine fancy indexing with masking mask = np.array([1, 0, 1, 0], dtype=bool) X[row[:, np.newaxis], mask]

```
array([[ 0, 2],
[ 4, 6],
[ 8, 10]])
```

Modifying Values with Fancy Indexing

Just as fancy indexing can be used to access parts of an array, it can also be used to modify parts of an array. Change some value in an array

Modify particular element by index

For example, imagine we have an array of indices and we'd like to set the corresponding items in an array to some value.

```
x = np.arange(10)
i = np.array([2, 1, 8, 4])
x[i] = 99
print(x)
```

[0 99 99 3 99 5 6 7 99 9]

Using assignment operator

We can use any assignment-type operator for this. For example

```
x[i] -= 10
print(x)
```

[0 89 89 3 89 5 6 7 89 9]

Using at()

Use the at() method of ufuncs for other behavior of modifications.

```
x = np.zeros(10) np.add.at(x, i, 1) print(x)
```

[0. 0. 1. 2. 3. 0. 0. 0. 0. 0.]

Sorting Arrays

Sorting in NumPy: np.sort and np.argsort

Python has built-in sort and sorted functions to work with lists, we won't discuss them here because NumPy's np.sort function turns out to be much more efficient and useful for our purposes. By default np.sort uses an O[*N* log *N]*, quicksort algorithm, though mergesort and heapsort are also available. For most applications, the default quicksort is more than sufficient.

Sorting without modifying the input.

To return a sorted version of the array without modifying the input, you can use np.sort

```
x = np.array([2, 1, 4, 3, 5]) np.sort(x)
```

array([1, 2, 3, 4, 5])

Returns sorted indices

A related function is argsort, which instead returns the *indices* of the sorted elements

```
x = np.array([2, 1, 4, 3, 5]) i = np.argsort(x)
print(i)
```

[1 0 3 2 4]

Sorting along rows or columns

A useful feature of NumPy's sorting algorithms is the ability to sort along specific rows or columns of a multidimensional array using the axis argument. For example

```
rand = np.random.RandomState(42) X = rand.randint(0, 10, (4, 6)) print(X)
```

[[6 3 7 4 6 9]
[2 6 7 4 3 7]
[7 2 5 4 1 7]
[5 1 4 0 9 5]]
np.sort(X, axis=0) array([[2, 1, 4, 0, 1, 5],
[5, 2, 5, 4, 3, 7],
[6, 3, 7, 4, 6, 7],
[7, 6, 7, 4, 9, 9]])
np.sort(X, axis=1) array([[3, 4, 6, 6, 7, 9],
[2, 3, 4, 6, 7, 7],
[1, 2, 4, 5, 7, 7],
[0, 1, 4, 5, 5, 9]])

Partial Sorts: Partitioning

Sometimes we're not interested in sorting the entire array, but simply want to find the K smallest values in the array. NumPy provides this in the np.partition function. np.partition takes an array and a number K; the result is a new array with the smallest K values to the left of the partition, and the remaining values to the right, in arbitrary order

```
x = np.array([7, 2, 3, 1, 6, 5, 4])
np.partition(x, 3)
```

array([2, 1, 3, 4, 6, 5, 7])

Note that the first three values in the resulting array are the three smallest in the array, and the remaining array positions contain the remaining values. Within the two partitions, the elements have arbitrary order.

Partitioning in multidimensional array

Similarly to sorting, we can partition along an arbitrary axis of a multidimensional array.

```
np.partition(X, 2, axis=1)
```

array([[3, 4, 6, 7, 6, 9],
[2, 3, 4, 7, 6, 7],
[1, 2, 4, 5, 7, 7],
[0, 1, 4, 5, 9, 5]])

4.6 STRUCTURED ARRAYS

This section demonstrates the use of NumPy‘s structured arrays and record arrays, which provide efficient storage for compound, heterogeneous data.

NumPy data types

Character	Description	Example
'b'	Byte	np.dtype('b')
'i'	Signed integer	np.dtype('i4') == np.int32
'u'	Unsigned integer	np.dtype('u1') == np.uint8
'f'	Floating point	np.dtype('f8') == np.int64
'c'	Complex floating point	np.dtype('c16') == np.complex128
'S', 'a'	string	np.dtype('S5')
'U'	Unicode string	np.dtype('U') == np.str_ 'V' Raw data (void)

np.dtype('V') == np.void

Consider if we have several categories of data on a number of people (say, name, age, and weight), and we‘d like to store these values for use in a Python program. It would be possible to store these in three separate arrays.

name = ['Alice', 'Bob', 'Cathy', 'Doug'] age = [25, 45, 37, 19]
weight = [55.0, 85.5, 68.0, 61.5]

Creating structured array

NumPy can handle this through structured arrays, which are arrays with compound data types. create a structured array using a compound data type specification as follows.

```
data = np.zeros(4, dtype={'names':('name', 'age', 'weight'),
                          'formats':('U10', 'i4', 'f8')})
print(data.dtype)
```

[('name', '<U10'), ('age', '<i4'), ('weight', '<f8')] U10 - Unicode string of maximum length 10 i4 - 4-byte (i.e., 32 bit) integer
f8 - 8-byte (i.e., 64 bit) float

Now we can fill the array with our lists of values

```
data['name'] = name data['age'] = age data['weight'] = weight print(data)
```

[('Alice', 25, 55.0) ('Bob', 45, 85.5) ('Cathy', 37, 68.0)('Doug', 19, 61.5)]

Refer values through index or name

The handy thing with structured arrays is that you can now refer to values either by index or by name.

```
data['name']# by name
```

array(['Alice', 'Bob', 'Cathy', 'Doug'],dtype='<U10')

```
data[0]# by index
```

('Alice', 25, 55.0)

Using Boolean masking

This allows to do some more sophisticated operations such as filtering on any fields.

```
data[data['age'] < 30]['name']
```

array(['Alice', 'Doug'],dtype='<U10')

Creating Structured Arrays Dictionary method

np.dtype({'names':('name', 'age', 'weight'),
'formats':('U10', 'i4', 'f8')}) dtype([('name', '<U10'), ('age', '<i4'), ('weight', '<f8')])

Numerical types can be specified with Python types

np.dtype({'names':('name', 'age', 'weight'),
'formats':((np.str_, 10), int, np.float32)}) dtype([('name', '<U10'), ('age', '<i8'), ('weight', '<f4')])

List of tuples

np.dtype([('name', 'S10'), ('age', 'i4'), ('weight', 'f8')])

dtype([('name', 'S10'), ('age', '<i4'), ('weight', '<f8')])

Specify the types alone

np.dtype('S10,i4,f8')
dtype([('f0', 'S10'), ('f1', '<i4'), ('f2', '<f8')])

4.7 DATA MANIPULATION WITH PANDAS

Pandas is a newer package built on top of NumPy, and provides an efficient implementation of a DataFrame. DataFrames are essentially multidimensional arrays with attached row and column labels, and often with heterogeneous types and/or missing data.

Pandas, and in particular its Series and DataFrame objects, builds on the NumPy array structure and provides efficient access to these sorts of ―data munging‖ tasks that occupy much of a data scientist‘s time.

Here we will focus on the mechanics of using Series, DataFrame, and related structures effectively.

Introducing Pandas Objects

Pandas objects can be thought of as enhanced verions of NumPy structured arrays in which the rows and columns are identified with labels rather than simple integer indices.

Pandas provide a host of useful tools, methods, and functionality on top of the basic data structures. Three fundamental Pandas data structures: the Series, DataFrame, and Index

The Pandas Series Object

A Pandas Series is a one-dimensional array of indexed data. It can be created from a list or array as follows:

data = pd.Series([0.25, 0.5, 0.75, 1.0]) data

0 0.25
1 0.50
2 0.75
3 1.00
dtype: float64

- ***Finding values***

The values are simply a familiar NumPy array
data.values

array([0.25, 0.5 , 0.75, 1.])

- ***Finding index***

The index is an array-like object of type pd.Index
data.index

RangeIndex(start=0, stop=4, step=1)

- ***Access by index***

Like with a NumPy array, data can be accessed by the associated index via the familiar Python square-bracket notation
data[1] 0.5
data[1:3]

1 0.50
2 0.75
dtype: float64

Series as generalized NumPy array

the NumPy array has an implicitly defined integer index used to access the values, the Pandas Series has an explicitly defined index associated with the values.
This explicit index definition gives the Series object additional capabilities. For example, the index need not be an integer, but can consist of values of any desired type.
For example, if we wish, we can use strings as an index.

Strings as an index

data = pd.Series([0.25, 0.5, 0.75, 1.0],
index=['a', 'b', 'c', 'd']) data

a 0.25
b 0.50
c 0.75
d 1.00
dtype: float64

Noncontiguous or non sequential indices.

data = pd.Series([0.25, 0.5, 0.75, 1.0],
index=[2, 5, 3, 7]) data

2 0.25
5 0.50
3 0.75
7 1.00
dtype: float64

Series as specialized dictionary

A dictionary is a structure that maps arbitrary keys to a set of arbitrary values, and a Series is a structure that maps typed keys to a set of typed values.
just as the type-specific compiled code behind a NumPy array makes it more efficient than a Python list for certain operations, the type information of a Pandas Series makes it much more efficient than Python dictionaries for certain operations.

We can make the Series-as-dictionary analogy even more clear by constructing a Series object directly from a Python dictionary.
For example

sub1={_sai‘:90,‘ram‘:85,‘kasim‘:92,‘tamil‘:89} mark=pd.Series(sub1)

mark

sai	90
ram	85
kasim	92
tamil	89

dtype: int64

Dictionary-style item access

Mark[_ram‘]
85

Array-style slicing

Mark[_sai‘:‘kasim‘]
sai 90
ram 85
kasim 92

Constructing Series objects

- ***List or NumPy array***

pd.Series([2, 4, 6])

0 2
1 4
2 6
dtype: int64

- ***Repeated to fill the specified index***

pd.Series(5, index=[100, 200, 300])

100 5
200 5
300 5
dtype: int64

- ***Data can be a dictionary, in which index defaults to the sorted dictionary keys***

pd.Series({2:'a', 1:'b', 3:'c'})

1 b
2 a
3 c
dtype: object

- ***The index can be explicitly set if a different result is preferred***

pd.Series({2:'a', 1:'b', 3:'c'}, index=[3, 2]) 3 c
2 a
dtype: object

The Pandas DataFrame Object

The fundamental structure in Pandas is the DataFrame. The DataFrame can be thought of either as a generalization of a NumPy array, or as a specialization of a Python dictionary.

DataFrame as a generalized NumPy array

A DataFrame is an analog of a two-dimensional array with both flexible row indices and flexible column names. Just as you might think of a two-dimensional array as an ordered sequence of aligned one-dimensional columns, you can think of a DataFrame as a sequence of aligned Series objects. Here, by ―aligned‖ we mean that they share the same index.

To demonstrate this, let‘s first construct a new Series listing the marks of subject2.

```
sub2={'sai':91,'ram':95,'kasim':89,'tamil':90}
```

We can use a dictionary to construct a single two-dimensional object containing this information.

```
result=pd.DataFrame({'DS':sub1,'FDS':sub2})  result
```

		DS	FDS
Sai		90	91
Ram		85	95
Kasim	92	89	
Tamil	89	90	

DataFrame has an index attribute

Like the Series object, the DataFrame has an index attribute that gives access to the index labels

```
result.index
```

```
Index(['sai', 'ram', 'kasim', 'tamil'], dtype='object')
```

DataFrame has a columns attribute.

The DataFrame has a columns attribute, which is an Index object holding the column labels.

```
result.columns
```

```
Index(['DS', 'FDS'], dtype='object')
```

DataFrame as specialized dictionary

We can also think of a DataFrame as a specialization of a dictionary. Where a dictionary maps a key to a value, a DataFrame maps a column name to a Series of column data.

```
result['DS']
```

```
sai 90
ram 85
kasim 92
tamil 89
Name: DS, dtype: int64
```

Note

In a two-dimensional NumPy array, data[0] will return the first row. For a DataFrame, data['col0'] will return the first column. Because of this, it is probably better to think about DataFrames as generalized dictionaries rather than generalized arrays, though both ways of looking at the situation can be useful.

Constructing DataFrame objects

A Pandas DataFrame can be constructed in a variety of ways. Here we‘ll give several examples.

- **From a single Series object.**
- **From a list of dicts.**

- **From a dictionary of Series objects.**
- **From a two-dimensional NumPy array.**
- **From a NumPy structured array.**

From a single Series object.

A DataFrame is a collection of Series objects, and a single column DataFrame can be constructed from a single Series.
sub1=pd.Series({'sai':90,'ram':85,'kasim':92,'tamil':89}) pd.DataFrame(sub1,columns=['DS'])

	DS
Sai	90
Ram	85
Kasim	92
Tamil	89

From a list of dicts.
Any list of dictionaries can be made into a DataFrame. We'll use a simple list comprehension to create some data
data = [{'a': i, 'b': 2 * i} **for** i **in** range(3)] pd.DataFrame(data)

a b
0 0 0
1 1 2
2 2 4
Even if some keys in the dictionary are missing, Pandas will fill them in with NaN (i.e.,—not a number‖) values.
pd.DataFrame([{'a': 1, 'b': 2}, {'b': 3, 'c': 4}]) a b c
0 1.0 2 NaN
1 NaN 3 4.0

From a dictionary of Series objects.

As we saw before, a DataFrame can be constructed from a dictionary of Series objects as well.
pd.DataFrame({'DS':sub1,'FDS':sub2})

		DS	FDS
sai		90	91
ram		85	95
kasim	92	89	
tamil	89	90	

From a two-dimensional NumPy array.
Given a two-dimensional array of data, we can create a DataFrame with any specified column and index names. If omitted, an integer index will be used for each.
pd.DataFrame(np.random.rand(3, 2), columns=['food', 'water'],
index=['a', 'b', 'c'])

food water
a 0.865257 0.213169
b 0.442759 0.108267
c 0.047110 0.905718

From a NumPy structured array.
A Pandas DataFrame operates much like a structured array, and can be created directly.
A = np.zeros(3, dtype=[('A', 'i8'), ('B', 'f8')]) A

array([(0, 0.0), (0, 0.0), (0, 0.0)],
dtype=[('A', '<i8'), ('B', '<f8')])
pd.DataFrame(A) A B
0 0 0.0
1 0 0.0
2 0 0.0

The Pandas Index Object
We have seen here that both the Series and DataFrame objects contain an explicit index that lets you reference and modify data. This Index object is an interesting structure in itself, and it can be thought of either as an immutable array or as an ordered set.
ind = pd.Index([2, 3, 5, 7, 11]) ind

Int64Index([2, 3, 5, 7, 11], dtype='int64')

- ***Index as immutable array***

The Index object in many ways operates like an array. For example, we can use standard Python indexing notation to retrieve values or slices.

ind[1] 3
ind[::2]
Int64Index([2, 5, 11], dtype='int64')

- ***Index as ordered set***

Pandas objects are designed to facilitate operations such as joins across datasets, which depend on many aspects of set arithmetic.
The Index object follows many of the conventions used by Python‘s built-in set data structure, so that unions, intersections, differences, and other combinations can be computed in a familiar way.

indA = pd.Index([1, 3, 5, 7, 9])
indB = pd.Index([2, 3, 5, 7, 11]) indA & indB *# intersection*

Int64Index([3, 5, 7], dtype='int64') indA | indB *# union*
Int64Index([1, 2, 3, 5, 7, 9, 11], dtype='int64')

indA ^ indB *# symmetric difference*

Int64Index([1, 2, 9, 11], dtype='int64')

4.8 DATA INDEXING AND SELECTION

Data Selection in Series

A Series object acts in many ways like a one dimensional NumPy array, and in many ways like a standard Python dictionary. It will help us to understand the patterns of data indexing and selection in these arrays.

- Series as dictionary
- Series as one-dimensional array
- Indexers: loc, iloc, and ix

- ***Series as dictionary***

Like a dictionary, the Series object provides a mapping from a collection of keys to a collection of values.

```
data = pd.Series([0.25, 0.5, 0.75, 1.0],
index=['a', 'b', 'c', 'd']) data
```

```
a 0.25
b 0.50
c 0.75
d 1.00
dtype: float64
```

```
data['b']
```

```
0.5
```

Examine the keys/indices and values

We can also use dictionary-like Python expressions and methods to examine the keys/indices and values

```
'a' in data
True
data.keys()
Index(['a', 'b', 'c', 'd'], dtype='object')
```

```
list(data.items())
[('a', 0.25), ('b', 0.5), ('c', 0.75), ('d', 1.0)]
```

Modifying series object

Series objects can even be modified with a dictionary-like syntax. Just as you can extend a dictionary by assigning to a new key, you can extend a Series by assigning to a new index value.

```
data['e'] = 1.25 data
```

```
a 0.25
b 0.50
c 0.75
d 1.00
e 1.25
dtype: float64
```

- ***Series as one-dimensional array***

A Series builds on this dictionary-like interface and provides array-style item selection via the same basic mechanisms as NumPy arrays—that is, slices, masking, and fancy indexing.

Slicing by explicit index
data['a':'c']

a 0.25
b 0.50
c 0.75
dtype: float64

Slicing by implicit integer index
data[0:2]

a 0.25
b 0.50
dtype: float64

Masking
data[(data > 0.3) & (data < 0.8)]

b 0.50

c 0.75

dtype: float64

Fancy indexing
data[['a', 'e']]

a 0.25
e 1.25
dtype: float64

- ***Indexers: loc, iloc, and ix***

Pandas provides some special indexer attributes that explicitly expose certain indexing schemes. These are not functional methods, but attributes that expose a particular slicing interface to the data in the Series.
data = pd.Series(['a', 'b', 'c'], index=[1, 3, 5]) data

1 a
3 b
5 c
dtype: object

loc - the loc attribute allows indexing and slicing that always references the explicit index.
data.loc[1] 'a'

data.loc[1:3] 1 a
3 b
dtype: object
iloc - The iloc attribute allows indexing and slicing that always references the implicit Python-style index.

data.iloc[1] 'b'

data.iloc[1:3] 3 b
5 c
dtype: object

ix- ix is a hybrid of the two, and for Series objects is equivalent to standard [ ]-based indexing.

Data Selection in DataFrame

- **DataFrame as a dictionary**
- **DataFrame as two-dimensional array**
- **Additional indexing conventions**

DataFrame as a dictionary

The first analogy we will consider is the DataFrame as a dictionary of related Series objects.

The individual Series that make up the columns of the DataFrame can be accessed via dictionary-style indexing of the column name.

Dictionary-style indexing of the column name. result=pd.DataFrame({'DS':sub1,'FDS':sub2}) result[_DS‘]

	DS
Sai	90
Ram	85
Kasim	92
Tamil	89

Attribute-style access with column names that are strings
result.DS

	DS
Sai	90
Ram	85
Kasim	92
Tamil	89

Comparing attribute style and dictionary style accesses
result.DS is result[_DS‘]

True
Modify the object
Like with the Series objects this dictionary-style syntax can also be used to modify the object, in this case to add a new column:

result[_TOTAL‘]=result[_DS‘]+result[_FDS‘] result

		DS	FDS	TOTAL
Sai		90	91	181
Ram		85	95	180
Kasim	92	89	181	
Tamil	89	90	179	

DataFrame as two-dimensional array

- **Transpose**

We can transpose the full DataFrame to swap rows and columns.
result.T

	sai	ram	kasim	tamil
DS	90	85	92	89
FDS	91	95	89	90
TOTAL 181	180	181	179	
				28

Pandas again uses the loc, iloc, and ix indexers mentioned earlier. Using the iloc indexer, we can index the underlying array as if it is a simple NumPy array (using the implicit Python-style index), but the DataFrame index and column labels are maintained in the result

- ***loc***

result.loc[: _ram‘, : _FDS‘]

	DS	FDS
Sai	90	91
Ram	85	95

- ***iloc***

result.iloc[:2, :2]

	DS	FDS
sai	90	91
ram	85	95

- ***ix***

result.ix[:2, :‘FDS‘]

	DS	FDS
Sai	90	91
Ram	85	95

- ***Masking and Fancy indexing***

In the loc indexer we can combine masking and fancy indexing as in the following:
result.loc[result.total>180,[_DS‘, _FDS‘]]

	DS	FDS
sai	90	91
kasim 92	89	

- ***Modifying values***

Indexing conventions may also be used to set or modify values; this is done in the standard way that you might be accustomed to from working with NumPy.

result.iloc[1,1] =70

		DS	FDS	TOTAL
Sai		90	91	181
Ram		85	**70**	180
Kasim	92	89	181	
Tamil	89	90	179	

Additional indexing conventions Slicing row wise

result['sai':'kasim']

		DS	FDS	TOTAL
Sai		90	91	181
Ram		85	**70**	180
Kasim	92	89	181	

Such slices can also refer to rows by number rather than by index:
result[1:3]

	DS	FDS	OTAL
ram	85	**70**	80
kasim 92	89	181	

Masking row wise
result[result.total>180]

	DS	FDS	TOTAL
Sai	90	91	181
kasim 92	89	181	

4.9 OPERATING ON DATA IN PANDAS

Pandas inherits much of this functionality from NumPy, and the ufuncs. So Pandas having the ability to perform quick element-wise operations, both with basic arithmetic (addition, subtraction, multiplication,

etc.) and with more sophisticated operations (trigonometric functions, exponential and logarithmic functions, etc.).

For unary operations like negation and trigonometric functions, these ufuncs will preserve index and column labels in the output.

For binary operations such as addition and multiplication, Pandas will automatically align indices when passing the objects to the ufunc.

Here we are going to see how the universal functions are working in series and DataFrames by

- **Index preservation**
- **Index alignment**

Index Preservation

Pandas is designed to work with NumPy, any NumPy ufunc will work on Pandas Series and DataFrame objects. We can use all arithmetic and special universal functions as in NumPy on pandas. In outputs the index will preserved (maintained) as shown below.

For series

```
x=pd.Series([1,2,3,4])  x

0 1
1 2
2 3
3 4
dtype: int64
```

For DataFrame

```
df=pd.DataFrame(np.random.randint(0,10,(3,4)),
                                        columns=['a','b','c','d'])
```

a	b	c
1	4	1
8	4	0
7	7	7

For universal function. (here we use exponent as example)

Ufuncs for series

```
np.exp(ser)

0 8103.083928
1 54.598150
2 403.428793
3 20.085537
dtype: float64
```

Ufuncs for Data Frame

```
np.exp(df)
```

	A	b	c	d
0	2.718282	54.598150	2.718282	54.598150
1	2980.957987	54.598150	1.000000	54.598150
2	1096.633158	1096.633158	1096.633158	7.389056

Index Alignment

Pandas will align indices in the process of performing the operation. This is very convenient when you are working with incomplete data, as we'll.

Index alignment in Series

suppose we are combining two different data sources, then the index will aligned accordingly.

```
x=pd.Series([2,4,6],index=[1,3,5])
y=pd.Series([1,3,5,7],index=[1,2,3,4])  x+y
```

```
1 3.0
2   NaN
3   9.0
4   NaN
5   NaN
dtype: float64
```

The resulting array contains the union of indices of the two input arrays, which we could determine using standard Python set arithmetic on these indices.

Any item for which one or the other does not have an entry is marked with NaN, or ―Not a Number,‖ which is how Pandas marks as missing data.

Fill value in missing data (fill_value)

If using NaN values is not the desired behavior, we can modify the fill value using appropriate object methods in place of the operators.

```
x.add(y,fill_value=0)
```

```
1 3.0
2 3.0
3 9.0
4 7.0
5 6.0
dtype: float64
```

Index alignment in DataFrame

A similar type of alignment takes place for both columns and indices when you are performing operations on DataFrames.

A = pd.DataFrame(rng.randint(0, 20, (2, 2)),columns=list('AB')) A

A B
0 1 11
1 5 1

B = pd.DataFrame(rng.randint(0, 10, (3, 3)), columns=list('BAC'))
B

B A C
0 4 0 9
1 5 8 0
2 9 2 6

A + B

A B C
0 1.0 15.0 NaN
1 13.0 6.0 NaN
2 NaN NaN NaN

Notice that indices are aligned correctly irrespective of their order in the two objects, and indices in the result are sorted. As was the case with Series, we can use the associated object's arithmetic method and pass any desired fill_value to be used in place of missing entries. Here we'll fill with the mean of all values in A.

fill = A.stack().mean() A.add(B, fill_value=fill)

A B C
0 1.0 15.0 13.5
1 13.0 6.0 4.5
2 6.5 13.5 10.5

Mapping between Python operators and Pandas methods. Python operator Pandas method(s)

+	add()
-	sub(), subtract()
*	mul(), multiply()
/	truediv(), div(), divide()
//	floordiv()
%	mod()
**	pow()

Operations between Data Frame and Series

When you are performing operations between a DataFrame and a Series, the index and column alignment is similarly maintained. Operations between a DataFrame and a Series are similar to operations between a two- dimensional and one-dimensional NumPy array.

A = rng.randint(10, size=(3, 4)) A
array([[3, 8, 2, 4],
[2, 6, 4, 8],
[6, 1, 3, 8]])

```
A - A[0]
array([[ 0, 0, 0, 0],
[-1, -2, 2, 4],
[ 3, -7, 1, 4]])
```

4.10 HANDLING MISSING DATA

A number of schemes have been developed to indicate the presence of missing data in a table or DataFrame. Generally, they revolve around one of two strategies: using a **mask** that globally indicates missing values, or choosing a **sentinel value** that indicates a missing entry.
In the masking approach, the mask might be an entirely separate Boolean array, or it may involve appropriation of one bit in the data representation to locally indicate the null status of a value.

In the sentinel approach, the sentinel value could be some data-specific convention, such as indicating a missing integer value with –9999 or some rare bit pattern, or it could be a more global convention, such as indicating a missing floating-point value with NaN (Not a Number), a special value which is part of the IEEE floating-point specification.

Missing Data in Pandas

The way in which Pandas handles missing values is constrained by its NumPy package, which does not have a built-in notion of NA values for non floating- point data types.

NumPy supports fourteen basic integer types once you account for available precisions, signedness, and endianness of the encoding. Reserving a specific bit pattern in all available NumPy types would lead to an unwieldy amount of overhead in special-casing various operations for various types, likely even requiring a new fork of the NumPy package.

Pandas chose to use sentinels for missing data, and further chose to use two already-existing Python null values: the special floatingpoint NaN value, and the Python None object. This choice has some side effects, as we will see, but in practice ends up being a good compromise in most cases of interest.

None: Pythonic missing data

The first sentinel value used by Pandas is None, a Python singleton object that is often used for missing data in Python code. Because None is a Python object, it cannot be used in any arbitrary NumPy/Pandas array, but only in arrays with data type 'object' (i.e., arrays of Python objects)

This dtype=object means that the best common type representation NumPy could infer for the contents of the array is that they are Python objects.

NaN: Missing numerical data

NaN is a special floating-point value recognized by all systems that use the standard IEEE floating-point representation.

```
vals2 = np.array([1, np.nan, 3, 4]) vals2.dtype
```

```
dtype('float64')
```

You should be aware that NaN is a bit like a data virus—it infects any other object it touches. Regardless of the operation, the result of arithmetic with NaN will be another NaN

```
1 + np.nan nan
0 * np.nan
```

Nan

NaN and None in Pandas

NaN and None both have their place, and Pandas is built to handle the two of them nearly interchangeably.

pd.Series([1, np.nan, 2, None])

0 1.0
1 NaN
2 2.0
3 NaN

dtype: float64

For types that don‘t have an available sentinel value, Pandas automatically type-casts when NA values are present. For example, if we set a value in an integer array to np.nan, it will automatically be upcast to a floating- point type to accommodate the NA

x = pd.Series(range(2), dtype=int) x

0 0
1 1

dtype: int64 x[0] = None x

0 NaN
1 1.0

dtype: float64

Notice that in addition to casting the integer array to floating point, Pandas automatically converts the None to a NaN value.

Pandas handling of NAs by type

Typeclass	Conversion when storing NAs	NA sentinel value
floating	No change	np.nan
object	No change	None or np.nan
integer	Cast to float64	np.nan
boolean	Cast to object	None or np.nan

Note : In Pandas, string data is always stored with an object dtype**.**

Operating on Null Values

there are several useful methods for detecting, removing, and replacing null values in Pandas data structures. They are:

- isnull() - Generate a Boolean mask indicating missing values
- notnull() - Opposite of isnull()
- dropna() - Return a filtered version of the data
- fillna() - Return a copy of the data with missing values filled or imputed

Detecting null values

Pandas data structures have two useful methods for detecting null data: isnull() and notnull(). isnull()
data = pd.Series([1, np.nan, 'hello', None]) data.isnull()

0 False
1 True
2 False

3 True dtype: bool

notnull()
data.notnull()

0 True
1 False True

3 False dtype: bool

Dropping null values

dropna()
data.dropna()

0 1
2 hello dtype: object

Dropping null values in dataframe

df = pd.DataFrame([[1, np.nan, 2], [2, 3, 5],
[np.nan, 4, 6]]) Df

0 1 2
0 1.0 NaN 2
1 2.0 3.0 5
2 NaN 4.0 6

df.dropna()

0 1 2
1 2.0 3.0 5

Drop values in column or row
We can drop NA values along a different axis; axis=1 drops all columns containing a null value.

df.dropna(axis='columns')

0 2
1 5
2 6

Rows or columns having all null values
You can also specify how='all', which will only drop rows/columns that are all null values.

df[3] = np.nan df

0 1 2 3
0 1.0 NaN 2 NaN
1 2.0 3.0 5 NaN
2 NaN 4.0 6 NaN

df.dropna(axis='columns', how='all') 0 1 2
0 1.0 NaN 2
1 2.0 3.0 5
2 NaN 4.0 6

Specific no of null values (thresh)

the thresh parameter lets you specify a minimum number of non-null values for the row/column to be kept

df.dropna(axis='rows', thresh=3)

0 1 2 3
1 2.0 3.0 5 NaN

Filling null values

Sometimes rather than dropping NA values, you'd rather replace them with a valid value. This value might be a single number like zero, or it might be some sort of imputation or interpolation from the good values. You could do this in-place using the isnull() method as a mask, but because it is such a common operation Pandas provides the fillna() method, which returns a copy of the array with the null values replaced.

data = pd.Series([1, np.nan, 2, None, 3], index=list('abcde')) data

a 1.0
b NaN
c 2.0
d NaN

Fill with single value

We can fill NA entries with a single value, such as zero

data.fillna(0)

a 1.0
b 0.0
c 2.0
d 0.0
e 3.0
dtype: float64

Fill with previous value

We can specify a forward-fill to propagate the previous value forward

data.fillna(method='ffill')

a 1.0
b 1.0
c 2.0
d 2.0
e 3.0
dtype: float64

Fill with next value
We can specify a back-fill to propagate the next values backward.

data.fillna(method='bfill')

a 1.0
b 2.0
c 2.0
d 3.0
e 3.0
dtype: float64

4.11 HIERARCHICAL INDEXING

Up to this point we've been focused primarily on one-dimensional and twodimensional data, stored in Pandas Series and DataFrame objects, respectively. Often it is useful to go beyond this and store higher-dimensional data—that is, data indexed by more than one or two keys.
Pandas does provide Panel and Panel4D objects that natively handle three-dimensional and four-dimensional, a far more common pattern in practice is to make use of hierarchical indexing (also known as multi-indexing) to incorporate multiple index levels within a single index.

In this way, higher-dimensional data can be compactly represented within the familiar one-dimensional Series and two-dimensional DataFrame objects.

Here we'll explore the direct creation of MultiIndex objects; considerations around indexing, slicing, and computing statistics across multiply indexed data; and useful routines for converting between simple and hierarchically indexed representations of your data.

A Multiply Indexed Series

Pandas MultiIndex
Pandas provides a better way. Our tuple-based indexing is essentially a rudimentary multi-index, and the Pandas MultiIndex type gives us the type of operations we wish to have. We can create a multi-index from the tuples as follows

index = [('California', 2000), ('California', 2010),
('New York', 2000), ('New York', 2010),
('Texas', 2000), ('Texas', 2010)]
populations = [33871648, 37253956,
18976457, 19378102,
20851820, 25145561]
pop = pd.Series(populations, index=index) pop

(California, 2000) 33871648
(California, 2010) 37253956
(New York, 2000) 18976457
(New York, 2010) 19378102
(Texas, 2000) 20851820

```
#creating multi index
index = pd.MultiIndex.from_tuples(index) index

MultiIndex(levels=[['California','New York','Texas'],[2000, 2010]],
labels=[[0, 0, 1, 1, 2, 2], [0, 1, 0, 1, 0, 1]])
```

Hierarchical representation of the data

```
pop = pop.reindex(index) pop

California 2000 33871648
2010 37253956
New York  2000 18976457
2010 19378102
Texas          2000 20851820
2010 25145561
dtype: int64
```

Here the first two columns of the Series representation show the multiple index values, while the third column shows the data.

Access all data with second index

```
pop[:, 2010]

California 37253956
New York 19378102
Texas 25145561 dtype: int64
```

MultiIndex as extra dimension

we could easily have stored the same data using a simple DataFrame with index and column labels. The
unstack() method will quickly convert a multiplyindexed Series into a conventionally indexed DataFrame.

```
pop_df = pop.unstack() pop_df

2000  2010
California 33871648 37253956
New York              18976457 19378102
Texas          20851820 25145561
```

The **stack()** method provides the opposite operation.

```
pop_df.stack()

California 2000 33871648
2010 37253956
New York 2000 18976457
2010 19378102
Texas 2000 20851820
2010 25145561
dtype: int64
```

Add a new column in multi dimensional data frame.

```
pop_df = pd.DataFrame({'total': pop,
'under18': [9267089, 9284094,
4687374, 4318033,
5906301, 6879014]})
pop_df
```

	total	under18
California	2000	33871648 9267089
	2010	37253956 9284094
New York	2000	18976457 4687374
	2010	19378102 4318033
Texas	2000	20851820 5906301

2010 25145561 6879014

Universal functions

All the ufuncs and other functionality work with hierarchical indices.

```
f_u18 = pop_df['under18'] / pop_df['total'] f_u18.unstack()
```

	2000	2010
California	0.273 594	0.249211
New York	0.247010	0.222831
Texas	0.283251	0.273568

Methods of Multi Index Creation

To construct a multiply indexed Series or DataFrame is to simply pass a list of two or more index arrays to the constructor.

```
df = pd.DataFrame(np.random.rand(4, 2), index=[['a', 'a', 'b', 'b'], [1, 2, 1, 2]],
columns=['data1', 'data2']) df
```

data1 data2
a 1 0.554233 0.356072
2 0.925244 0.219474
b 1 0.441759 0.610054
2 0.171495 0.886688

if you pass a dictionary with appropriate tuples as keys, Pandas will automatically recognize this and use a MultiIndex by default.

```
data = { ('California', 2000): 33871648,
('California', 2010): 37253956,
('Texas', 2000): 20851820,
('Texas', 2010): 25145561,
('New York', 2000): 18976457,

('New York', 2010): 19378102}
pd.Series(data)
```

California	2000 33871648
	2010 37253956
New York	2000 18976457
	2010 19378102
Texas	2000 20851820
	2010 25145561
dtype: int64	

Explicit MultiIndex constructors

You can construct the MultiIndex from a simple list of arrays, giving the index values within each level.
pd.MultiIndex.from_arrays([['a', 'a', 'b', 'b'], [1, 2, 1, 2]])

MultiIndex(levels=[['a', 'b'], [1, 2]],
labels=[[0, 0, 1, 1], [0, 1, 0, 1]])

Multi index from a list of tuples,
pd.MultiIndex.from_tuples([('a', 1), ('a', 2), ('b', 1), ('b', 2)])

MultiIndex(levels=[['a', 'b'], [1, 2]],
labels=[[0, 0, 1, 1], [0, 1, 0, 1]])

Multi index from Cartesian product.
pd.MultiIndex.from_product([['a', 'b'], [1, 2]])

MultiIndex(levels=[['a', 'b'], [1, 2]],
labels=[[0, 0, 1, 1], [0, 1, 0, 1]])

MultiIndex level names

It is convenient to name the levels of the MultiIndex. You can accomplish this by passing the names argument to any of the above MultiIndex constructors, or by setting the names attribute of the index after the fact.
pop.index.names = ['state', 'year'] pop

State	year
California	2000 33871648
	2010 37253956
New York	2000 18976457
	2010 19378102
Texas	2000 20851820
	2010 25145561
dtype: int64	

MultiIndex for columns

In a DataFrame, the rows and columns are completely symmetric, and just as the rows can have multiple levels of indices, the columns can have multiple levels as well.

 # hierarchical indices and columns

```
index = pd.MultiIndex.from_product([[2013, 2014], [1, 2]], names=['year', 'visit'])
columns = pd.MultiIndex.from_product([['Bob', 'Guido', 'Sue'], ['HR', 'Temp']], names=['subject', 'type'])
```

 # mock some data

```
data = np.round(np.random.randn(4, 6), 1)
data[:, ::2] *= 10
data += 37
```

 # create the DataFrame

health_data = pd.DataFrame(data, index=index, columns=columns) health_data

subject	Bob		Guido		Sue	
type	HR	Temp	HR	Temp	HR	Temp
year visit						
2013 1	31.0	38.7	32.0	36.7	35.0	37.2
2	44.0	37.7	50.0	35.0	29.0	36.7
2014 1	30.0	37.4	39.0	37.8	61.0	36.9
2	47.0	37.8	48.0	37.3	51.0	36.5

Indexing and Slicing a MultiIndex

Indexing and slicing on a MultiIndex is designed to be intuitive, and it helps if you think about the indices as added dimensions. We'll first look at indexing multiply indexed Series, and then multiply indexed DataFrames.

Multiply indexed Series

Pop

state	year	
California	2000	33871648
	2010	37253956
New York	2000	18976457
	2010	19378102
Texas	2000	20851820
	2010	25145561

dtype: int64

- Access single elements

We can access single elements by indexing with multiple terms

```
pop['California', 2000]
```

33871648

- Partial indexing

The MultiIndex also supports partial indexing, or indexing just one of the levels in the index

```
pop['California']
```

year
2000 33871648
2010 37253956
dtype: int64

- Partial slicing

Partial slicing is available as well, as long as the MultiIndex is sorted.
pop.loc['California':'New York']

State	year
California	2000 33871648
	2010 37253956
New York	2000 18976457
	2010 19378102
dtype: int64	

- Sorted indices

With sorted indices, we can perform partial indexing on lower levels by passing an empty slice in the first index
pop[:, 2000]

State	
California	33871648
New York	18976457
Texas	20851820
dtype: int64	

- Other types of indexing and selection Selection based on Boolean masks

pop[pop > 22000000]

state year
California 2000 33871648
2010 37253956
Texas 2010 25145561
dtype: int64

Selection based on fancy indexing
pop[['California', 'Texas']]

state year
California 2000 33871648
2010 37253956
Texas 2000 20851820
2010 25145561
dtype: int64

Rearranging Multi-Indices

We saw a brief example of this in the stack() and unstack() methods, but there are many more ways to finely control the rearrangement of data between hierarchical indices and columns, and we‘ll explore them here.

- ***Sorted and unsorted indices***

We'll start by creating some simple multiply indexed data where the indices are *not lexographically sorted*:

index = pd.MultiIndex.from_product([['a', 'c', 'b'], [1, 2]]) data = pd.Series(np.random.rand(6), index=index) data.index.names = ['char', 'int']
data
char int

A	1	0.003001
	2	0.164974
C	1	0.741650

Pandas provides a number of convenience routines to perform this type of sorting; examples are the sort_index() and sortlevel() methods of the DataFrame. We'll use the simplest, sort_index(), here:

data = data.sort_index() data

char int

A	1	0.003001
	2	0.164974
B	1	0.001693
	2	0.526226
C	1	0.741650
	2	0.569264

dtype: float64

With the index sorted in this way, partial slicing will work as expected:
data['a':'b'] char int

a	1	0.003001
	2	0.164974
b	1	0.001693
	2	0.526226

dtype: float64

- ***Stacking and unstacking indices***

it is possible to convert a dataset from a stacked multi-index to a simple two-dimensional representation, optionally specifying the level to use.
pop.unstack(level=0)

state California New York Texas year
2000 33871648 18976457 20851820
2010 37253956 19378102 25145561
pop.unstack(level=1)

year	2000
state	
California	33871648 37253956
New York	18976457 19378102
Texas	20851820 25145561

The opposite of unstack() is stack(), which here can be used to recover the original series:
pop.unstack().stack()

state year	
California	2000 33871648
	2010 37253956
New York	2000 18976457
	2010 19378102
Texas	2000 20851820

2010 25145561
dtype: int64

- ***Index setting and resetting***

Another way to rearrange hierarchical data is to turn the index labels into columns; this can be accomplished with the reset_index method. Calling this on the population dictionary will result in a DataFrame with a state and year column holding the information that was formerly in the index. For clarity, we can optionally specify the name of the data for the column representation.
pop_flat = pop.reset_index(name='population') pop_flat

state	year population
0 California	2000 33871648
1 California	2010 37253956
2 New York	2000 18976457
3 New York	2010 19378102
4 Texas	2000 20851820
5 Texas	2010 25145561

Data Aggregations on Multi-Indices

We‘ve previously seen that Pandas has built-in data aggregation methods, such as mean(), sum(), and max(). For hierarchically indexed data, these can be passed a level parameter that controls which subset of the data the aggregate is computed on.
For example, let‘s return to our health data: (you can create your own data frame / series)
health_data

subject	Bob		Guido Sue	
type	Temp HR	HR	Temp HR	Temp

year visit
2013 1 31.0 38.7 32.0 36.7 35.0 37.2
2 44.0 37.7 50.0 35.0 29.0 36.7

2014 1 30.0 37.4 39.0 37.8 61.0 36.9
2 47.0 37.8 48.0 37.3 51.0 36.5

Calculate the average as follows
data_mean = health_data.mean(level='year') data_mean

subject Bob Guido Sue
type HR Temp HR Temp HR Temp year
2013 37.5 38.2 41.0 35.85 32.0 3695
2014 38.5 37.6 43.5 37.55 56.0 36.70
By further making use of the axis keyword, we can take the mean among levels on the columns as well:
data_mean.mean(axis=1, level='type') type HR Temp
year
2013 36.833333 37.000000
2014 46.000000 37.283333

4.12 COMBINING DATASETS
Concat and Append

Simple Concatenation with pd.concat
Pandas has a function, pd.concat(), which has a similar syntax to np.concatenate but contains a number of options that we'll discuss momentarily
pd.concat() can be used for a simple concatenation of Series or DataFrame objects, just as np.concatenate() can be used for simple concatenations of arrays
ser1 = pd.Series(['A', 'B', 'C'], index=[1, 2, 3])
ser2 = pd.Series(['D', 'E', 'F'], index=[4, 5, 6]) pd.concat([ser1, ser2])

1 A
2 B
3 C
4 D
5 E
6 F
dtype: object

Concatenation in data frame.
df1 = make_df('AB', [1, 2])
df2 = make_df('AB', [3, 4])
print(df1); **print**(df2); **print**(pd.concat([df1, df2]))

df3 df4 pd.concat([df3, df4], axis='col')

A B C D A B C D
0 A0 B0 0 C0 D0 0 A0 B0 C0 D0
1 A1 B1 1 C1 D1 1 A1 B1 C1 D1

Duplicate indices
One important difference between np.concatenate and pd.concat is that Pandas concatenation preserves indices, even if the result will have duplicate indices! Consider this simple example.

x = make_df('AB', [0, 1])
y = make_df('AB', [2, 3])

y.index = x.index *# make duplicate indices!*
print(x); **print**(y); **print**(pd.concat([x, y]))

x y pd.concat([x, y])

A B A B A B

0 A0 B0 0 A2 B2 0 A0 B0

1 A1 B1 1 A3 B3 1 A1 B1

0 A2 B2

1 A3 B3

The append() method

Series and DataFrame objects have an append method that can accomplish the same thing in fewer keystrokes. For example, rather than calling pd.concat([df1, df2]), you can simply call df1.append(df2):
print(df1); **print**(df2); **print**(df1.append(df2))

df1		df2	df1.append(df2)
A B	A B	A B	

1 A1 B1 3 A3 B3 1 A1 B1

2 A2 B2 4 A4 B4 2 A2 B2

3 A3 B3

4 A4 B4

Merge and Join

One essential feature offered by Pandas is its high-performance, in-memory join and merge operations.

Categories of Joins

- One-to-one joins
- Many-to-one joins
- Many-to-many joins

One – to – one joins

The simplest type of merge expression is the one-to-one join, which is in many ways very similar to the column-wise concatenation.

df1 = pd.DataFrame({'employee': ['Bob', 'Jake', 'Lisa', 'Sue'],
'group': ['Accounting', 'Engineering', 'Engineering', 'HR']})

df2 = pd.DataFrame({'employee': ['Lisa', 'Bob', 'Jake', 'Sue'], 'hire_date': [2004, 2008, 2012, 2014]})

print(df1); **print**(df2)

d df2

f

1 emplo employee hire_date

yee group

0 Bob Accounting 0 Lisa 2004

1	Jake	Engineering	1 Bob	2008
2	Lisa	Engineering	2 Jake	2012
3	Sue	HR		3 Sue 2014

To combine this information into a single DataFrame, we can use the pd.merge() function

df3 = pd.merge(df1, df2) df3

	employee	group	hire_date
0	Bob	Accounting	2008
1	Jake	Engineering	2012
2	Lisa	Engineering	2004
3	Sue	HR	2014

Many-to-one joins

Many-to-one joins are joins in which one of the two key columns contains duplicate entries. For the many-to- one case, the resulting DataFrame will preserve those duplicate entries as appropriate.

```
df4 = pd.DataFrame({'group': ['Accounting', 'Engineering', 'HR'],
                                    'supervisor': ['Carly', 'Guido', 'Steve']})
pd.merge(df3, df4)
```

	Employee		group		hire_date supervisor	
0	Bob		Accounting		2008 Carly	
1	Jake	Engineering		2012	Guido	
2	Lisa	Engineering		2004	Guido	
3	Sue		HR		2014	Steve

The resulting DataFrame has an additional column with the ―supervisor‖ information, where the information is repeated in one or more locations as required by the inputs.

Many-to-many joins

Many-to-many joins are a bit confusing conceptually, but are nevertheless well defined. If the key column in both the left and right array contains duplicates, then the result is a many-to-many merge. This will be perhaps most clear with a concrete example.

```
df5 = pd.DataFrame({'group': ['Accounting', 'Accounting', 'Engineering', 'Engineering', 'HR', 'HR'],
'skills': ['math', 'spreadsheets', 'coding', 'linux', 'spreadsheets', 'organization']})
pd.merge(df1, df5)
```

	employee	**Group**	**skills**
0	Bob	Accounting	math
1	Bob	Accounting	spreadsheets
2	Jake	Engineering	coding
3	Jake	Engineering	Linux
4	Lisa	Engineering	coding
5	Lisa	Engineering	Linux
6	Sue	HR	spreadsheets
7	Sue	HR	organization

4.13 AGGREGATION AND GROUPING

Computing aggregations like sum(), mean(), median(), min(), and max(), in which a single number gives insight into the nature of a potentially large dataset.

Simple Aggregation in Pandas

As with a one dimensional NumPy array, for a Pandas Series the aggregates return a single value.

```
rng = np.random.RandomState(42) ser = pd.Series(rng.rand(5))
ser

0 0.374540
1 0.950714
2 0.731994
3 0.598658
4 0.156019
dtype: float64

ser.sum() 2.8119254917081569
```

SUM

Mean

ser.mean() 0.56238509834163142

The same operations also performed in DataFrame

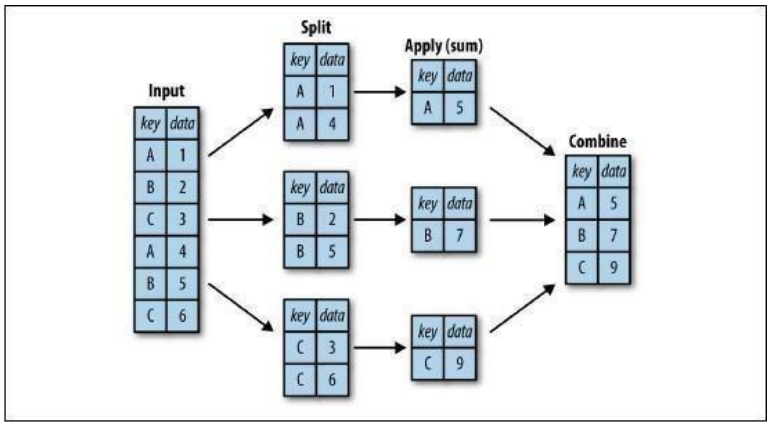
Listing of Pandas aggregation methods Aggregation		***Description***
count()	Total number of items	
first(), last()	First and last item mean(), median()	Mean and median
min(), max()	Minimum and maximum	
std(), var()	Standard deviation and variance	
mad()	Mean absolute deviation	
prod()	Product of all items	
sum()	Sum of all items	

GroupBy: Split, Apply, Combine

Simple aggregations can give you a flavor of your dataset, but often we would prefer to aggregate conditionally on some label or index: this is implemented in the socalled groupby operation. The name ―group by‖ comes from a command in the SQL database language, but it is perhaps more illuminative to think of it in the terms first coined by Hadley Wickham of Rstats fame: split, apply, combine.

- The split step involves breaking up and grouping a DataFrame depending on the value of the specified key.
- The apply step involves computing some function, usually an aggregate, transformation, or filtering, within the individual groups.

- The combine step merges the results of these operations into an output array.



Example

df = pd.DataFrame({'key': ['A', 'B', 'C', 'A', 'B', 'C'],
'data': range(6)}, columns=['key', 'data'])
Df

key data

A
B
C
A
B
C

The GroupBy object

The GroupBy object is a very flexible abstraction. The most important operations made available by a GroupBy are aggregate, filter, transform, and apply.

Groupby supports the basic operations like.

- Column indexing.
- Iteration over groups.
- Dispatch methods.
- Aggregate, filter, transform, apply

Column indexing.

The GroupBy object supports column indexing in the same way as the DataFrame, and returns a modified GroupBy object. For example
df=pd.read_csv('D:\iris.csv') df.groupby('variety')

<pandas.core.groupby.generic.DataFrameGroupByobjectat 0x0000023BAADE84C0>

df.groupby(' variety)[' petal.length"]

<pandas.core.groupby.generic.SeriesGroupByobject at 0x0000023BAADE8490>

df.groupby(' variety ')[—petal.length"].sum()

variety Setosa 73.1
Versicolor 213.0
Virginica 277.6
Name: petal.length, dtype: float64

Iteration over groups.

The GroupBy object supports direct iteration over the groups, returning each group as a Series or DataFrame.

This can be useful for doing certain things manually, though it is often much faster to use the built-in apply functionality, which we will discuss momentarily.

Dispatch methods.

Through some Python class magic, any method not explicitly implemented by the GroupBy object will be passed through and called on the groups, whether they are DataFrame or Series objects. For example, you can use the describe() method of DataFrames to perform a set of aggregations that describe each group in the data.

Example

df.groupby('variety')['petal.length'].describe().unstack()

variety

count	Setosa	50.000000
	Versicolor	50.000000
	Virginica	50.000000
mean	Setosa	1.462000
	Versicolor	4.260000
	Virginica	5.552000
Std	Setosa	0.173664
	Versicolor	0.469911
	Virginica	0.551895
Min	Setosa	1.000000
	Versicolor	3.000000
	Virginica	4.500000
25%	Setosa	1.400000
	Versicolor	4.000000
	Virginica	5.100000
50%	Setosa	1.500000
	Versicolor	4.350000
	Virginica	5.550000
75%	Setosa	1.575000
	Versicolor	4.600000
	Virginica	5.875000
max	Setosa	1.900000
	Versicolor	5.100000
	Virginica	6.900000
dtype:	float64	

Aggregate, filter, transform,and apply

```
rng = np.random.RandomState(0)
df = pd.DataFrame({'key': ['A', 'B', 'C', 'A', 'B', 'C'],

'data1': range(6),
'data2': rng.randint(0, 10, 6)},
columns = ['key', 'data1', 'data2']) df
```

key data1 data2

0	A	0	5
1	B	1	0
2	C	2	3
3	A	3	3
4	B	4	7
5	C	5	9

Aggregation.

We‘re now familiar with GroupBy aggregations with sum(), median(), and the like, but the aggregate() method allows for even more flexibility. It can take a string, a function, or a list thereof, and compute all the aggregates at once. Here is a quick example combining all these:

```
df.groupby('key').aggregate(['min', np.median, max]) data1 data2
```

min median max min median max

key						
A	0	1.5	3	3	4.0	5
B	1	2.5	4	0	3.5	7
C	2	3.5	5	3	6.0	9

Filtering.

A filtering operation allows you to drop data based on the group properties. For example, we might want to keep all groups in which the standard deviation is larger than some critical value.
The filter() function should return a Boolean value specifying whether the group passes the filtering.

Transformation.

While aggregation must return a reduced version of the data, transformation can return some transformed version of the full data to recombine. For such a transformation, the output is the same shape as the input. A common example is to center the data by subtracting the group-wise mean:

```
df.groupby('key').transform(lambda x: x - x.mean())
```

	data1	data2
0	-1.5	1.0
1	-1.5	-3.5
2	-1.5	-3.0
3	1.5	-1.0
4	1.5	3.5
5	1.5	3.0

The apply() method.

The apply() method lets you apply an arbitrary function to the group results. The function should take a DataFrame, and return either a Pandas object (e.g., DataFrame, Series) or a scalar; the combine operation will be tailored to the type of output returned.

4.13 PIVOT TABLES

A pivot table is a similar operation that is commonly seen in spreadsheets and other programs that operate on tabular data. The pivot table takes simple column wise data as input, and groups the entries into a two- dimensional table that provides a multidimensional summarization of the data. The difference between pivot tables and GroupBy can sometimes cause confusion; it helps me to think of pivot tables as essentially a multidimensional version of GroupBy aggregation. That is, you split apply-combine, but both the split and the combine happen across not a one-dimensional index, but across a two-dimensional grid.

Pivot Table Creation

import numpy as np import pandas as pd
df=pd.read_csv('D:\diabetes.csv') df.pivot_table('preg',index='age',columns='Class').sample(10)

#here diabetes data set has large no of rows so we use sample()

Class	tested_negative	tested_positive
Age		
63	5.500000	NaN
28	3.440000	2.000000
61	7.000000	4.000000
69	5.000000	NaN
45	7.285714	7.375000
62	6.500000	1.000000
53	2.000000	6.250000
68	8.000000	NaN
23	1.516129	1.857143

Class	tested_negative	tested_positive
Age		
52	13.000000	3.428571

UNIT V

DATA VISUALIZATION

Importing Matplotlib – Line plots – Scatter plots – visualizing errors – density and contour plots – Histograms – text and annotation – customization – three-dimensional plotting - Geographic Data with Basemap - Visualization with Seaborn - multiple plots in one window – exporting graph using graphics parameters – Overview of Power BI).

Simple Line Plots

The simplest of all plots is the visualization of a single function y = f x . Here we will take a first look at creating a simple plot of this type.
The figure (an instance of the class plt.Figure) can be thought of as a single container that contains all the objects representing axes, graphics, text, and labels.
The axes (an instance of the class plt.Axes) is what we see above: a bounding box with ticks and labels, which will eventually contain the plot elements that make up our visualization.

Line Colors and Styles

- The first adjustment you might wish to make to a plot is to control the line colors and styles.
- To adjust the color, you can use the color keyword, which accepts a string argument representing virtually any imaginable color. The color can be specified in a variety of ways
- If no color is specified, Matplotlib will automatically cycle through a set of default colors for multiple lines

Different forms of color representation.
specify color by name - color='blue'
short color code (rgbcmyk) - color='g'
Grayscale between 0 and 1 - color='0.75'
Hex code (RRGGBB from 00 to FF) - color='#FFDD44' RGB tuple, values 0 and 1 - color=(1.0,0.2,0.3) all HTML color names supported - color='chartreuse'

- We can adjust the line style using the linestyle keyword.

Different line styles
linestyle='solid' linestyle='dashed' linestyle='dashdot' linestyle='dotted'

Short assignment
linestyle='-' # solid linestyle='--' # dashed linestyle='-.' # dashdot linestyle=':' # dotted

- linestyle and color codes can be combined into a single nonkeyword argument to the plt.plot() function

plt.plot(x, x + 0, '-g') # solid green plt.plot(x, x + 1, '--c') # dashed cyan plt.plot(x, x + 2, '-.k') # dashdot black plt.plot(x, x + 3, ':r'); # dotted red

Axes Limits

- The most basic way to adjust axis limits is to use the plt.xlim() and plt.ylim() methods Example

plt.xlim(10, 0)
plt.ylim(1.2, -1.2);

- The plt.axis() method allows you to set the x and y limits with a single call, by passing a list that specifies [xmin, xmax, ymin, ymax]

plt.axis([-1, 11, -1.5, 1.5]);

- Aspect ratio equal is used to represent one unit in x is equal to one unit in y. plt.axis('equal')

Labeling Plots

The labeling of plots includes titles, axis labels, and simple legends. Title - plt.title()
Label - plt.xlabel()
plt.ylabel()
Legend - plt.legend()

Example programs Line color

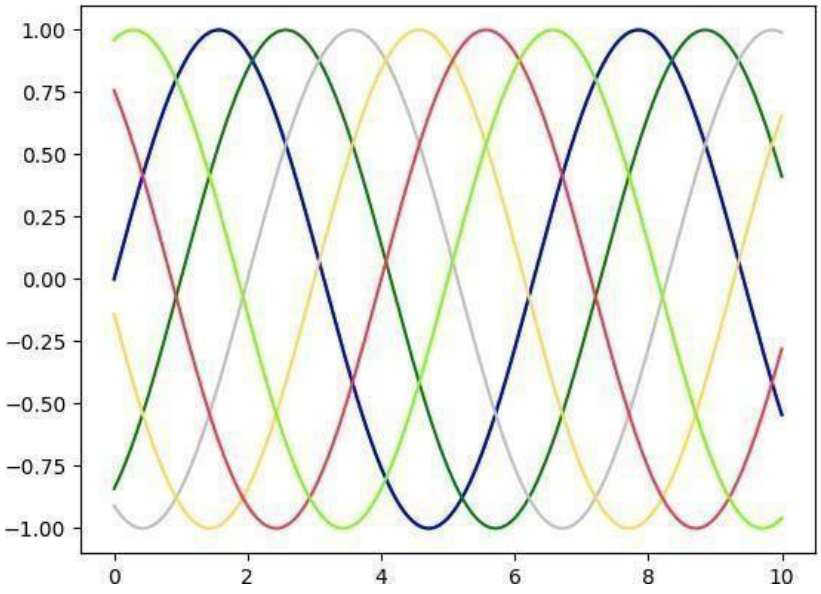
import matplotlib.pyplot as plt import numpy as np

fig = plt.figure() ax = plt.axes()

x = np.linspace(0, 10, 1000) ax.plot(x, np.sin(x));

plt.plot(x, np.sin(x - 0), color='blue') # specify color by name plt.plot(x, np.sin(x - 1), color='g') # short color code (rgbcmyk) plt.plot(x, np.sin(x - 2), color='0.75') # Grayscale between 0 and 1

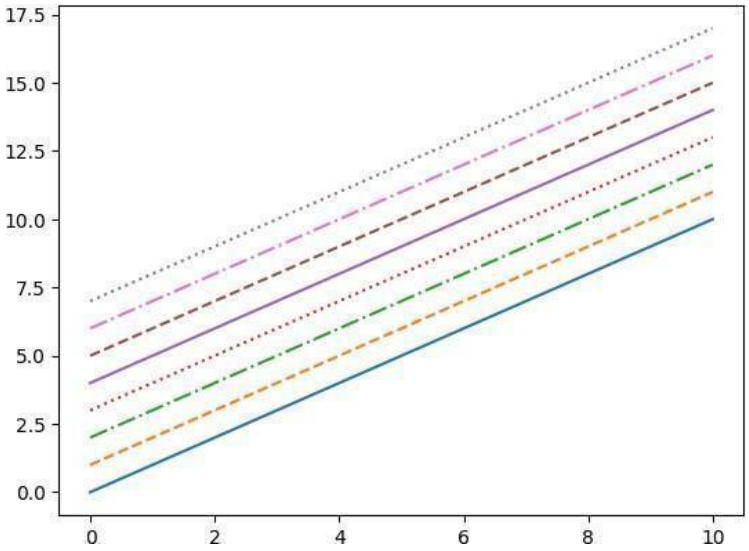
plt.plot(x, np.sin(x - 3), color='#FFDD44') # Hex code (RRGGBB from 00 to FF) plt.plot(x, np.sin(x - 4), color=(1.0,0.2,0.3)) # RGB tuple, values 0 and 1 plt.plot(x, np.sin(x - 5), color='chartreuse');# all HTML color names supported



Line style

import matplotlib.pyplot as plt

import numpy as np fig = plt.figure()
ax = plt.axes()
x = np.linspace(0, 10, 1000) plt.plot(x, x + 0, linestyle='solid') plt.plot(x, x + 1, linestyle='dashed') plt.plot(x, x + 2, linestyle='dashdot') plt.plot(x, x + 3, linestyle='dotted');
# For short, you can use the following codes: plt.plot(x, x + 4, linestyle='-') # solid plt.plot(x, x + 5, linestyle='--') # dashed plt.plot(x, x + 6, linestyle='-.') # dashdot plt.plot(x, x + 7, linestyle=':'); # dotted



Another commonly used plot type is the simple scatter plot, a close cousin of the line plot. Instead of points being joined by line segments, here the points are represented individually with a dot, circle, or other shape.
Syntax
plt.plot(x, y, 'type of symbol ', color);

Example
plt.plot(x, y, 'o', color='black');

- The third argument in the function call is a character that represents the type of symbol used for the plotting. Just as you can specify options such as '-' and '--' to control the line style, the marker style has its own set of short string codes.

Example

- Various symbols used to specify ['o', '.', ',', 'x', '+', 'v', '^', '<', '>', 's', 'd']
- Short hand assignment of line, symbol and color also allowed.

plt.plot(x, y, '-ok');

- Additional arguments in plt.plot()

We can specify some other parameters related with scatter plot which makes it more attractive. They are color, marker size, linewidth, marker face color, marker edge color, marker edge width, etc
Example
plt.plot(x, y, '-p', color='gray', markersize=15, linewidth=4, markerfacecolor='white', markeredgecolor='gray', markeredgewidth=2) plt.ylim(-1.2, 1.2);

Scatter Plots with plt.scatter

- A second, more powerful method of creating scatter plots is the plt.scatter function, which can be used very similarly to the plt.plot function

plt.scatter(x, y, marker='o');

- The primary difference of plt.scatter from plt.plot is that it can be used to create scatter plots where the properties of each individual point (size, face color, edge color, etc.) can be individually controlled or mapped to data.
- Notice that the color argument is automatically mapped to a color scale (shown here by the colorbar() command), and the size argument is given in pixels.
- Cmap – color map used in scatter plot gives different color combinations.

Perceptually Uniform Sequential

```
['viridis', 'plasma', 'inferno', 'magma']
```

Sequential

```
['Greys', 'Purples', 'Blues', 'Greens', 'Oranges', 'Reds', 'YlOrBr', 'YlOrRd',
'OrRd', 'PuRd', 'RdPu', 'BuPu', 'GnBu', 'PuBu', 'YlGnBu', 'PuBuGn', 'BuGn',
```

Diverging

```
['PiYG', 'PRGn', 'BrBG', 'PuOr', 'RdGy', 'RdBu', 'RdYlBu', 'RdYlGn', 'Spectral'
'coolwarm', 'bwr', 'seismic']
```

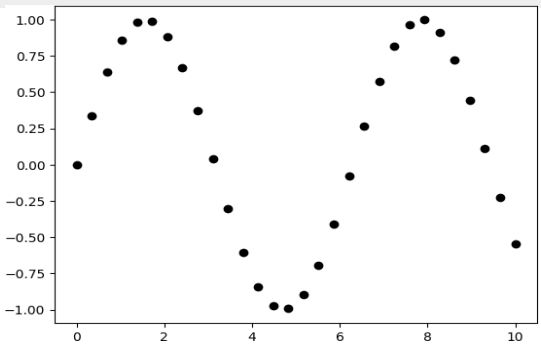
Qualitative

```
['Pastel1', 'Pastel2', 'Paired', 'Accent', 'Dark2', 'Set1', 'Set2', 'Set3',
'tab10', 'tab20', 'tab20b', 'tab20c']
```

Miscellaneous

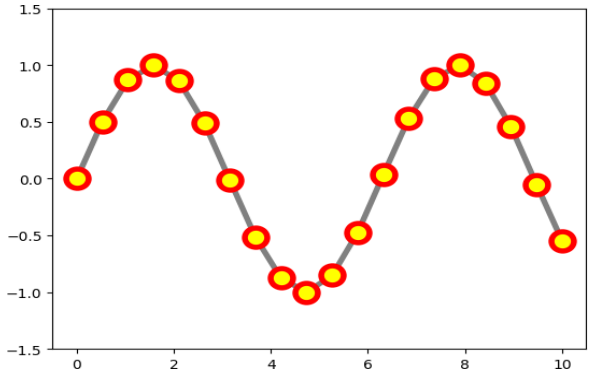
Example programs. Simple scatter plot.

```
import numpy as np
import matplotlib.pyplot as plt x = np.linspace(0, 10, 30)
y = np.sin(x)
plt.plot(x, y, 'o', color='black');
```



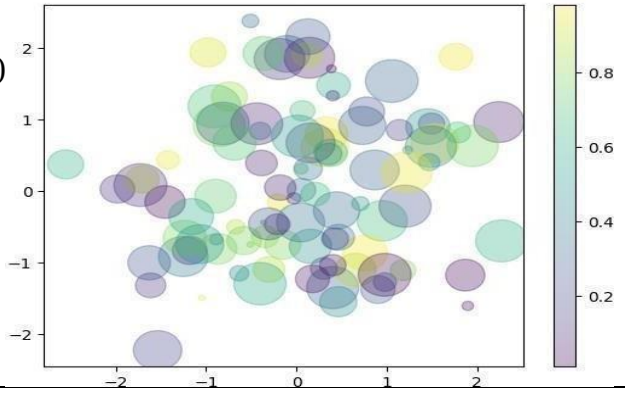
Scatter plot with edge color, face color, size, and width of marker. (Scatter plot with line)

```
import numpy as np
import matplotlib.pyplot as plt x = np.linspace(0, 10, 20)
y = np.sin(x)
plt.plot(x, y, '-o', color='gray', markersize=15, linewidth=4,
markerfacecolor='yellow', markeredgecolor='red',
markeredgewidth=4) plt.ylim(-1.5, 1.5);
```



Scatter plot with random colors, size and transparency

```
import numpy as np
import matplotlib.pyplot as plt rng = np.random.RandomState(0)
 x = rng.randn(100)
y = rng.randn(100) colors = rng.rand(100)
sizes = 1000 * rng.rand(100)
plt.scatter(x, y, c=colors, s=sizes, alpha=0.3,
map='viridis') plt.colorbar()
```



Visualizing Errors

For any scientific measurement, accurate accounting for errors is nearly as important, if not more important, than accurate reporting of the number itself. For example, imagine that I am using some astrophysical observations to estimate the Hubble Constant, the local measurement of the expansion rate of the Universe.
In visualization of data and results, showing these errors effectively can make a plot convey much more complete information.

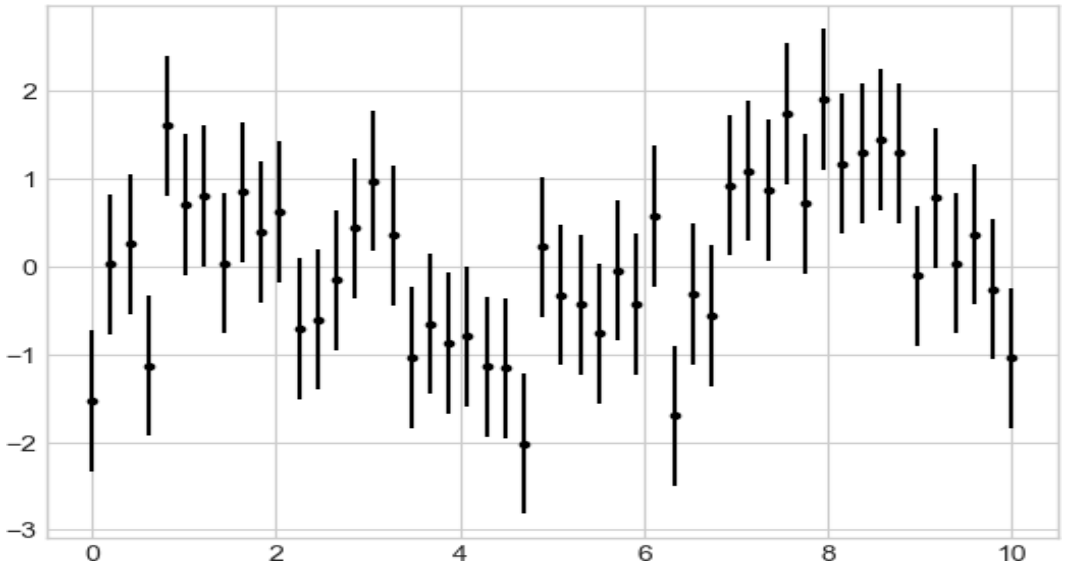
Types of errors

- Basic Errorbars
- Continuous Errors

Basic Errorbars

A basic errorbar can be created with a single Matplotlib function call.
import matplotlib.pyplot as plt plt.style.use('seaborn-whitegrid') **import numpy as np**

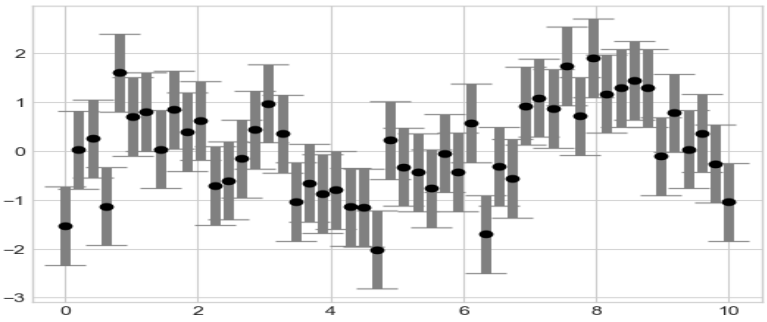
x = np.linspace(0, 10, 50)
dy = 0.8
y = np.sin(x) + dy * np.random.randn(50) plt.errorbar(x, y, yerr=dy, fmt='.k');



Here the fmt is a format code controlling the appearance of lines and points, and has the same syntax as the shorthand used in plt.plot()

- In addition to these basic options, the errorbar function has many options to fine tune the outputs. Using these additional options you can easily customize the aesthetics of your errorbar plot.

plt.errorbar(x, y, yerr=dy, fmt='o', color='black',ecolor='lightgray', elinewidth=3, capsize=0);



Continuous Errors

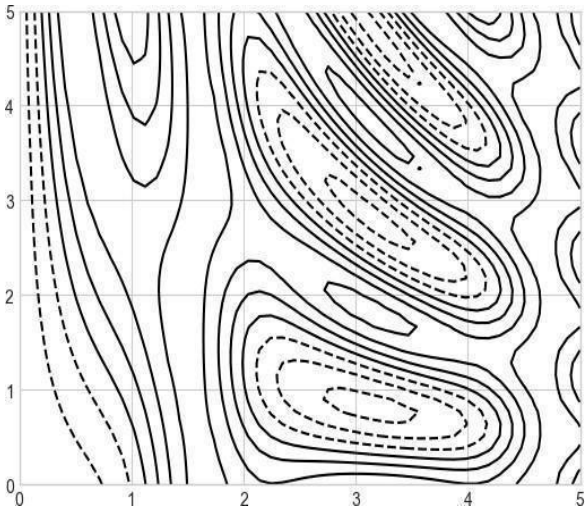
- In some situations it is desirable to show errorbars on continuous quantities. Though Matplotlib does not have a built-in convenience routine for this type of application, it's relatively easy to combine primitives like plt.plot and plt.fill_between for a useful result.
- Here we'll perform a simple Gaussian process regression (GPR), using the Scikit-Learn API. This is a

method of fitting a very flexible nonparametric function to data with a continuous measure of the uncertainty.

Density and Contour Plots

To display three-dimensional data in two dimensions using contours or color-coded regions. There are three Matplotlib functions that can be helpful for this task:

- plt.contour for contour plots,
- plt.contourf for filled contour plots, and
- plt.imshow for showing images.



Visualizing a Three-Dimensional Function

A contour plot can be created with the plt.contour function. I

t takes three arguments:

- a grid of x values,
- a grid of y values, and
- a grid of z values.

The x and y values represent positions on the plot, and the z values will be represented by the contour levels.

The way to prepare such data is to use the np.meshgrid function, which builds two-dimensional grids from one- dimensional arrays:

Example

def f(x, y):

return np.sin(x) ** 10 + np.cos(10 + y * x) * np.cos(x) x = np.linspace(0, 5, 50)

y = np.linspace(0, 5, 40) X, Y = np.meshgrid(x, y) Z = f(X, Y)

plt.contour(X, Y, Z, colors='black');

- Notice that by default when a single color is used, negative values are represented by dashed lines, and positive values by solid lines.
- Alternatively, you can color-code the lines by specifying a colormap with the cmap argument.
- We'll also specify that we want more lines to be drawn—20 equally spaced intervals within the data range.

plt.contour(X, Y, Z, 20, cmap='RdGy');

- One potential issue with this plot is that it is a bit ―splotchy.‖ That is, the color steps are discrete rather than continuous, which is not always what is desired.
- You could remedy this by setting the number of contours to a very high number, but this results in a rather inefficient plot: Matplotlib must render a new polygon for each step in the level.
- A better way to handle this is to use the plt.imshow() function, which interprets a two-dimensional grid of data as an image.

There are a few potential gotchas with imshow().

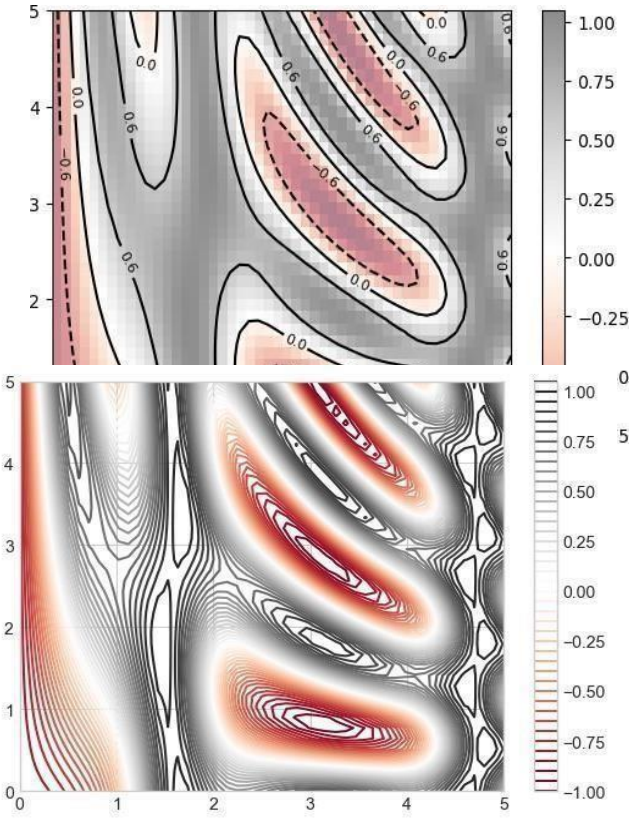
- plt.imshow() doesn't accept an x and y grid, so you must manually specify the extent [xmin, xmax, ymin, ymax] of the image on the plot.
- plt.imshow() by default follows the standard image array definition where the origin is in the upper left, not in the lower left as in most contour plots. This must be changed when showing gridded data.
- plt.imshow() will automatically adjust the axis aspect ratio to match the input data; you can change this by setting, for example, plt.axis(aspect='image') to make x and y units match.

Finally, it can sometimes be useful to combine contour plots and image plots. we'll use a partially transparent background image (with transparency set via the alpha parameter) and over-plot contours with labels on the contours themselves (using the plt.clabel() function):

contours = plt.contour(X, Y, Z, 3, colors='black') plt.clabel(contours, inline=True, fontsize=8) plt.imshow(Z, extent=[0, 5, 0, 5], origin='lower', cmap='RdGy', alpha=0.5)
plt.colorbar();

Example Program

```
import numpy as np

import matplotlib.pyplot as pltdef f(x, y):

return np.sin(x) ** 10 + np.cos(10 + y * x) * np.cos(x)

x = np.linspace(0, 5, 50)

y = np.linspace(0, 5, 40) X,

Y = np.meshgrid(x, y) Z = f(X, Y)

plt.imshow(Z, extent=[0, 10, 0, 10],

origin='lower', cmap='RdGy') plt.colorbar()
```



Histograms

- Histogram is the simple plot to represent the large data set. A histogram is a graph showing frequency distributions. It is a graph showing the number of observations within each given interval.

Parameters

- plt.hist( ) is used to plot histogram. The hist() function will use an array of numbers to create a histogram, the array is sent into the function as an argument.
- bins - A histogram displays numerical data by grouping data into "bins" of equal width. Each bin is plotted as a bar whose height corresponds to how many data points are in that bin. Bins are also sometimes called "intervals", "classes", or "buckets".
- normed - Histogram normalization is a technique to distribute the frequencies of the histogram over a wider range than the current range.
- x - (n,) array or sequence of (n,) arrays Input values, this takes either a single array or a sequence of arrays which are not required to be of the same length.
- histtype - {'bar', 'barstacked', 'step', 'stepfilled'}, optional The type of histogram to draw.
- 'bar' is a traditional bar-type histogram. If multiple data are given the bars are arranged side by side.
- 'barstacked' is a bar-type histogram where multiple data are stacked on top of each other.
- 'step' generates a lineplot that is by default unfilled.
- 'stepfilled' generates a lineplot that is by default filled. Default is 'bar'
- align - {'left', 'mid', 'right'}, optional Controls how the histogram is plotted.
- 'left': bars are centered on the left bin edges.
- 'mid': bars are centered between the bin edges.
- 'right': bars are centered on the right bin edges. Default is 'mid'
- orientation - {'horizontal', 'vertical'}, optional

If 'horizontal', barh will be used for bar-type histograms and the bottom kwarg will be the left edges.

- color - color or array_like of colors or None, optional

Color spec or sequence of color specs, one per dataset. Default (None) uses the standard line color sequence.

Default is None

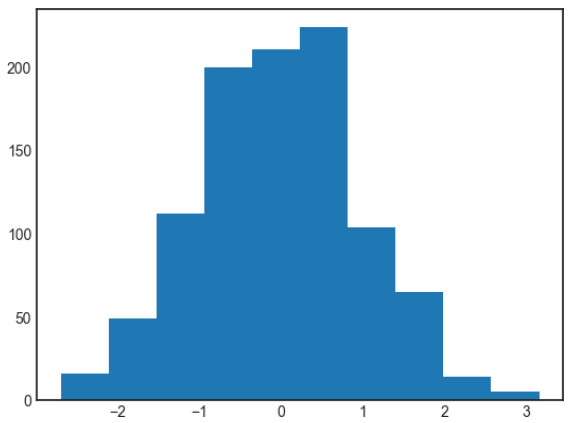
- label - str or None, optional. Default is None

Other parameter

- **kwargs - Patch properties, it allows us to pass a variable number of keyword arguments to a python function. ** denotes this type of function.

Example

```
import numpy as np
import matplotlib.pyplot as plt

plt.style.use('seaborn-white') data = np.random.randn(1000)

plt.hist(data);
```

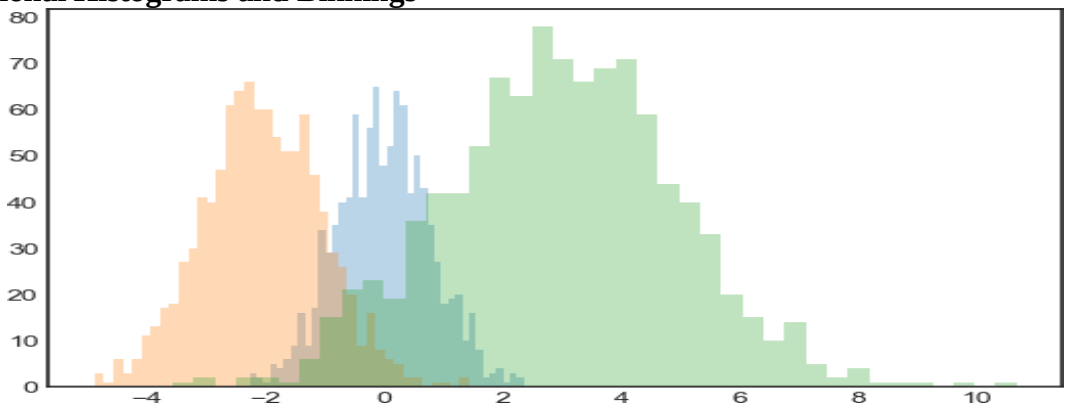


The hist() function has many options to tune both the calculation and the display; here's an example of a more customized histogram.
plt.hist(data, bins=30, alpha=0.5,histtype='stepfilled', color='steelblue',edgecolor='none');

The plt.hist docstring has more information on other customization options available. I find this combination of histtype='stepfilled' along with some transparency alpha to be very useful when comparing histograms of several distributions

x1 = np.random.normal(0, 0.8, 1000)
x2 = np.random.normal(-2, 1, 1000)
x3 = np.random.normal(3, 2, 1000)
kwargs = dict(histtype='stepfilled', alpha=0.3, bins=40) plt.hist(x1, **kwargs)
plt.hist(x2, **kwargs) plt.hist(x3, **kwargs);

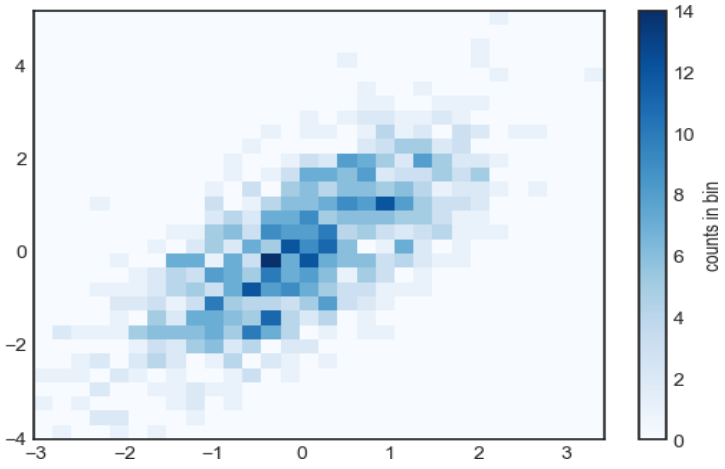
Two-Dimensional Histograms and Binnings



- We can create histograms in two dimensions by dividing points among two dimensional bins.
- We would define x and y values. Here for example We'll start by defining some data—an x and y array drawn from a multivariate Gaussian distribution:
- Simple way to plot a two-dimensional histogram is to use Matplotlib's plt.hist2d() function

Example

mean = [0, 0]
cov = [[1, 1], [1, 2]]
x, y = np.random.multivariate_normal(mean, cov, 1000).T plt.hist2d(x, y, bins=30, cmap='Blues')
cb = plt.colorbar() cb.set_label('counts in bin')



Legends

Plot legends give meaning to a visualization, assigning labels to the various plot elements. We previously saw how to create a simple legend; here we'll take a look at customizing the placement and aesthetics of the legend in Matplotlib.

Plot legends give meaning to a visualization, assigning labels to the various plot elements. We previously saw how to create a simple legend; here we'll take a look at customizing the placement and aesthetics of the legend in Matplotlib

```
plt.plot(x, np.sin(x), '-b', label='Sine')
plt.plot(x, np.cos(x), '--r', label='Cosine') plt.legend();
```

Customizing Plot Legends

Location and turn off the frame - We can specify the location and turn off the frame. By the parameter loc and framon.

```
ax.legend(loc='upper left', frameon=False) fig
```

Number of columns - We can use the ncol command to specify the number of columns in the legend.

```
ax.legend(frameon=False, loc='lower center', ncol=2) fig
```

Rounded box, shadow and frame transparency

We can use a rounded box (fancybox) or add a shadow, change the transparency (alpha value) of the frame, or change the padding around the text.

```
ax.legend(fancybox=True, framealpha=1, shadow=True, borderpad=1) fig
```

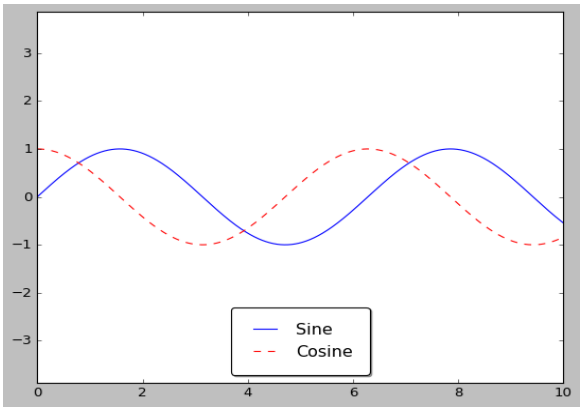
Choosing Elements for the Legend

- The legend includes all labeled elements by default. We can change which elements and labels appear in the legend by using the objects returned by plot commands.
- The plt.plot() command is able to create multiple lines at once, and returns a list of created line instances.

Passing any of these to plt.legend() will tell it which to identify, along with the labels we'd like to specify

```
y = np.sin(x[:, np.newaxis] + np.pi * np.arange(0, 2, 0.5))
lines = plt.plot(x, y)  plt.legend(lines[:2],['first','second']);
```

```
# Applying label individually. plt.plot(x, y[:, 0], label='first')
plt.plot(x, y[:, 1], label='second')
plt.plot(x, y[:, 2:]) plt.legend(framealpha=1, frameon=True);
```



Multiple legends

It is only possible to create a single legend for the entire plot. If you try to create a second legend using plt.legend() or ax.legend(), it will simply override the first one. We can work around this by creating a new legend artist from scratch, and then using the lower-level ax.add_artist() method to manually add the second artist to the plot

Example

```
import matplotlib.pyplot as plt plt.style.use('classic')
import numpy as np
x = np.linspace(0, 10, 1000)
ax.legend(loc='lower center', frameon=True, shadow=True,borderpad=1,fancybox=True) fig
```

Color Bars

In Matplotlib, a color bar is a separate axes that can provide a key for the meaning of colors in a plot. For continuous labels based on the color of points, lines, or regions, a labeled color bar can be a great tool. The simplest colorbar can be created with the plt.colorbar() function.

Customizing Colorbars Choosing color map.

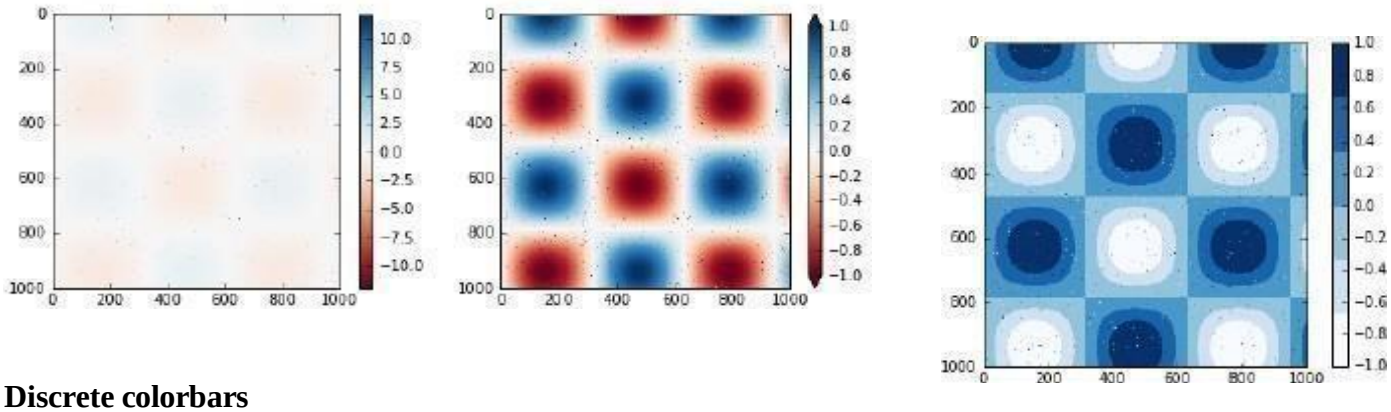
We can specify the colormap using the cmap argument to the plotting function that is creating the visualization. Broadly, we can know three different categories of colormaps:

- **Sequential colormaps** - These consist of one continuous sequence of colors (e.g., binary or viridis).
- **Divergent colormaps** - These usually contain two distinct colors, which show positive and negative deviations from a mean (e.g., RdBu or PuOr).
- **Qualitative colormaps** - These mix colors with no particular sequence (e.g., rainbow or jet).

Color limits and extensions

- Matplotlib allows for a large range of colorbar customization. The colorbar itself is simply an instance of plt.Axes, so all of the axes and tick formatting tricks we've learned are applicable.
- We can narrow the color limits and indicate the out-of-bounds values with a triangular arrow at the top and bottom by setting the extend property.

plt.subplot(1, 2, 2) plt.imshow(I, cmap='RdBu') plt.colorbar(extend='both') plt.clim(-1, 1);



Discrete colorbars

Colormaps are by default continuous, but sometimes you'd like to represent discrete values. The easiest way to do this is to use the plt.cm.get_cmap() function, and pass the name of a suitable colormap along with the number of desired bins.

plt.imshow(I, cmap=plt.cm.get_cmap('Blues', 6)) plt.colorbar()
plt.clim(-1, 1);

Subplots

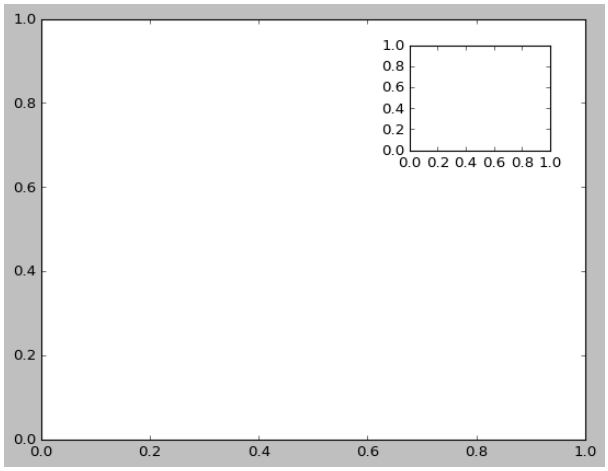
- Matplotlib has the concept of subplots: groups of smaller axes that can exist together within a single figure.
- These subplots might be insets, grids of plots, or other more complicated layouts.
- We'll explore four routines for creating subplots in Matplotlib.
- plt.axes: Subplots by Hand
- plt.subplot: Simple Grids of Subplots
- plt.subplots: The Whole Grid in One Go
- plt.GridSpec: More Complicated Arrangements

plt.axes: Subplots by Hand

- The most basic method of creating an axes is to use the plt.axes function. As we've seen previously, by default this creates a standard axes object that fills the entire figure.
- plt.axes also takes an optional argument that is a list of four numbers in the figure coordinate system.
- These numbers represent [bottom, left, width,height] in the figure coordinate system, which ranges from 0 at the bottom left of the figure to 1 at the top right of the figure.

For example,
we might create an inset axes at the top-right corner of another axes by setting the x and y position to 0.65 (that is, starting at 65% of the width and 65% of the height of the figure) and the x and y extents to 0.2 (that is, the size of the axes is 20% of the width and 20% of the height of the figure).

```
import matplotlib.pyplot as plt import numpy as np
ax1 = plt.axes() # standard axes ax2 = plt.axes([0.65, 0.65, 0.2, 0.2])
```



Vertical sub plot

The equivalent of plt.axes() command within the object-oriented interface is ig.add_axes().
Let's use this to create two vertically stacked axes.

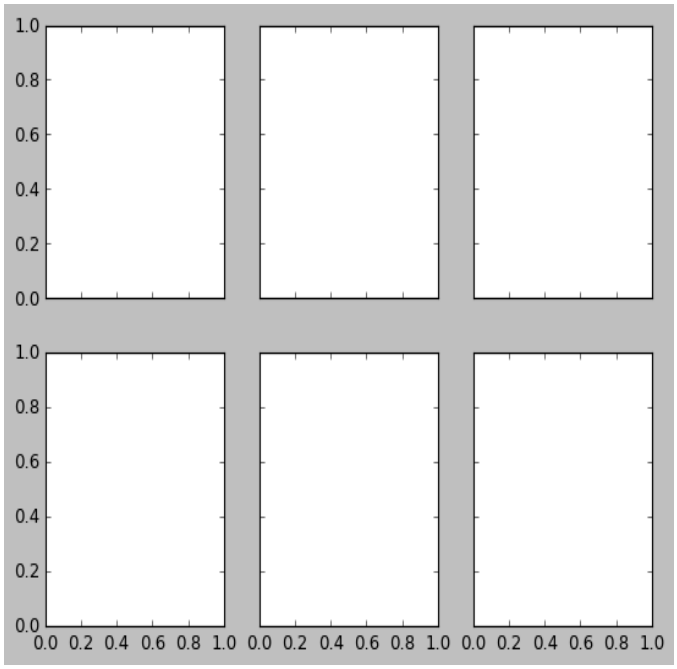
```
fig = plt.figure()

ax1 = fig.add_axes([0.1, 0.5, 0.8, 0.4],
xticklabels=[], ylim=(-1.2, 1.2))
ax2 = fig.add_axes([0.1, 0.1, 0.8, 0.4],
ylim=(-1.2, 1.2))
x = np.linspace(0, 10) ax1.plot(np.sin(x))
ax2.plot(np.cos(x));
```

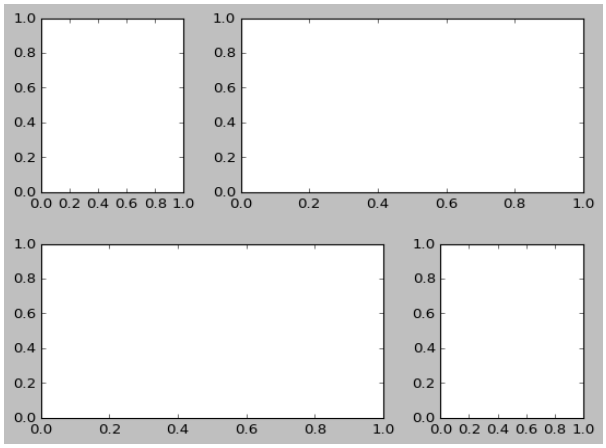
- We now have two axes (the top with no tick labels) that are just touching: the bottom of the upper panel (at position 0.5) matches the top of the lower panel (at position 0.1+ 0.4).
- If the axis value is changed in second plot both the plots are separated with each other, example ax2 = fig.add_axes([0.1, 0.01, 0.8, 0.4

plt.subplot: Simple Grids of Subplots

- Matplotlib has several convenience routines to align columns or rows of subplots.
- The lowest level of these is plt.subplot(), which creates a single subplot within a grid.
- This command takes three integer arguments—the number of rows, the number of columns, and the index of the plot to be created in this scheme, which runs from the upper left to the bottom right

```
for i in range(1, 7): plt.subplot(2, 3, i)
plt.text(0.5, 0.5, str((2, 3, i)), fontsize=18, ha='center')
```





plt.subplots: The Whole Grid in One Go

- The approach just described can become quite tedious when you're creating a large grid of subplots, especially if you'd like to hide the x- and y-
axis labels on the inner plots.
- For this purpose, plt.subplots() is the easier tool to use (note the s at the end of subplots).
- Rather than creating a single subplot, this function creates a full grid of subplots in a single line, returning them in a NumPy array.
- The arguments are the number of rows and number of columns, along with optional keywords sharex and sharey, which allow you to specify the relationships between different axes.
- Here we'll create a 2×3 grid of subplots, where all axes in the same row share their y- axis scale, and all axes in the same column share their x-axis scale

fig, ax = plt.subplots(2, 3, sharex='col', sharey='row')

Note that by specifying sharex and sharey, we've automatically removed inner labels on the grid to make the plot cleaner.

plt.GridSpec: More Complicated Arrangements

To go beyond a regular grid to subplots that span multiple rows and columns, plt.GridSpec() is the best tool. The plt.GridSpec() object does not create a plot by itself; it is simply a convenient interface that is recognized by the plt.subplot() command.

For example, a gridspec for a grid of two rows and three columns with some specified width and height space looks like this:

grid = plt.GridSpec(2, 3, wspace=0.4, hspace=0.3) From this we can specify subplot locations and extents

```
plt.subplot(grid[0, 0])
plt.subplot(grid[0, 1:])
plt.subplot(grid[1, :2])
plt.subplot(grid[1, 2]);
```

Text and Annotation

- The most basic types of annotations we will use are axes labels and titles, here we will see some more visualization and annotation information's.
- Text annotation can be done manually with the plt.text/ax.text command, which will place text at a particular x/y value.
- The ax.text method takes an x position, a y position, a string, and then optional keywords specifying the color, size, style, alignment, and other properties of the text. Here we used ha='right' and ha='center', where ha is short for horizontal alignment.

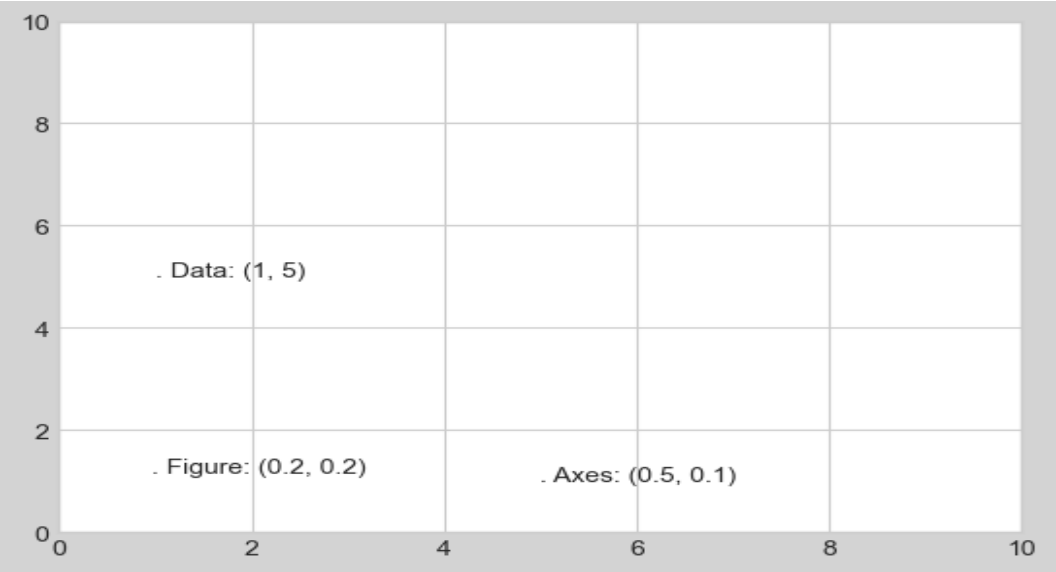
Transforms and Text Position

- We anchored our text annotations to data locations. Sometimes it's preferable to anchor the text to a position on the axes or figure, independent of the data. In Matplotlib, we do this by modifying the transform.
- Any graphics display framework needs some scheme for translating between coordinate systems.
- Mathematically, such coordinate transformations are relatively straightforward, and Matplotlib has a well- developed set of tools that it uses internally to perform them (the tools can be explored in the matplotlib.transforms submodule).
- There are three predefined transforms that can be useful in this situation.

- ax.transData - Transform associated with data coordinates
- ax.transAxes - Transform associated with the axes (in units of axes dimensions)
- fig.transFigure - Transform associated with the figure (in units of figure dimensions)

EXAMPLLE

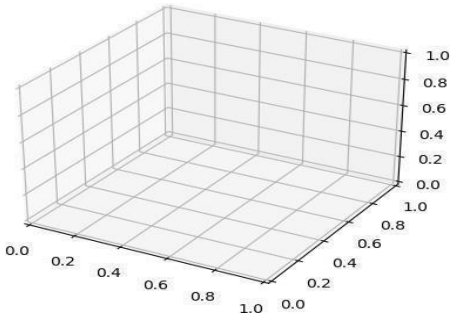
```
import matplotlib.pyplot as plt import matplotlib as mpl plt.style.use('seaborn-whitegrid') import numpy as np
import pandas as pd
fig, ax = plt.subplots(facecolor='lightgray') ax.axis([0, 10, 0, 10])
# transform=ax.transData is the default, but we'll specify it anyway ax.text(1, 5, ". Data: (1, 5)",
transform=ax.transData)
ax.text(0.5, 0.1, ". Axes: (0.5, 0.1)", transform=ax.transAxes)
ax.text(0.2, 0.2, ". Figure: (0.2, 0.2)", transform=fig.transFigure);
```



Three-Dimensional Plotting in Matplotlib

We enable three-dimensional plots by importing the mplot3d toolkit, included with the main Matplotlib installation.

```
import numpy as np
import matplotlib.pyplot as plt from mpl_toolkits
import mplot3d fig = plt.figure()
ax = plt.axes(projection='3d')
```



With this 3D axes enabled, we can now plot a variety of three-dimensional plot types.

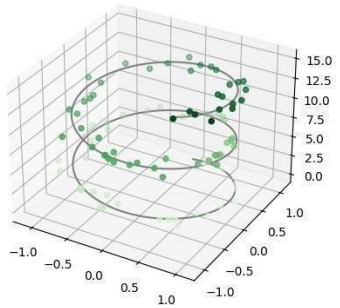
Three-Dimensional Points and Lines

The most basic three-dimensional plot is a line or scatter plot created from sets of (x, y, z) triples.

In analogy with the more common two-dimensional plots discussed earlier, we can create these using the ax.plot3D

and ax.scatter3D functions

```
import numpy as np
import matplotlib.pyplot as plt from mpl_toolkits import mplot3d ax = plt.axes(projection='3d')
# Data for a three-dimensional line zline = np.linspace(0, 15, 1000) xline = np.sin(zline)
yline = np.cos(zline) ax.plot3D(xline, yline, zline, 'gray')

# Data for three-dimensional scattered points zdata =
15 * np.random.random(100)
xdata = np.sin(zdata) + 0.1 * np.random.randn(100)
ydata = np.cos(zdata) + 0.1 * np.random.randn(100)
```



Note that by default, the text is aligned above and to the left of the specified coordinates; here the "." at the beginning of each string will approximately mark the given coordinate location.

The transData coordinates give the usual data coordinates associated with the x- and y-axis labels. The transAxes coordinates give the location from the bottom-left corner of the axes (here the white box) as a fraction of the axes size.

The transfigure coordinates are similar, but specify the position from the bottom left of the figure (here the gray box) as a fraction of the figure size.
Notice now that if we change the axes limits, it is only the transData coordinates that will be affected, while the others remain stationary.

Arrows and Annotation

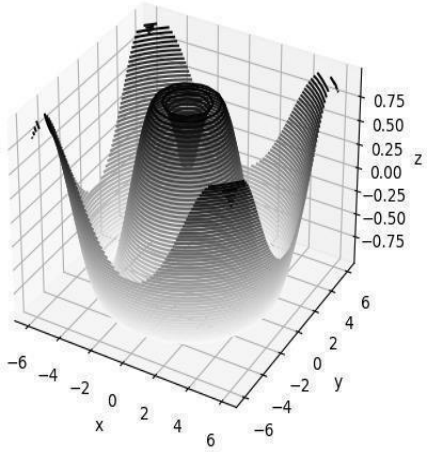
- Along with tick marks and text, another useful annotation mark is the simple arrow.
- Drawing arrows in Matplotlib is not much harder because there is a plt.arrow() function available.
- The arrows it creates are SVG (scalable vector graphics)objects that will be subject to the varying aspect ratio of your plots, and the result is rarely what the user intended.
- The arrow style is controlled through the arrowprops dictionary, which has numerous options available.ax.scatter3D(xdata, ydata, zdata, c=zdata, cmap='Greens'); plt.show()

Notice that by default, the scatter points have their transparency adjusted to give a sense of depth on the page.

Three-Dimensional Contour Plots

- mplot3d contains tools to create three-dimensional relief plots using the same inputs.
- Like two-dimensional ax.contour plots, ax.contour3D requires all the input data to be in the form of two-dimensional regular grids, with the Z data evaluated at each point.
- Here we'll show a three-dimensional contour diagram of a three dimensional sinusoidal function

```
import numpy as np
import matplotlib.pyplot as plt from mpl_toolkits
 import mplot3d def f(x, y):
return np.sin(np.sqrt(x ** 2 + y ** 2)) x = np.linspace(-6, 6, 30)
y = np.linspace(-6, 6, 30) X, Y = np.meshgrid(x, y) Z = f(X, Y)
fig = plt.figure()
ax = plt.axes(projection='3d') ax.contour3D(X, Y, Z, 50,
cmap='binary') ax.set_xlabel('x')
ax.set_ylabel('y') ax.set_zlabel('z') plt.show()
```



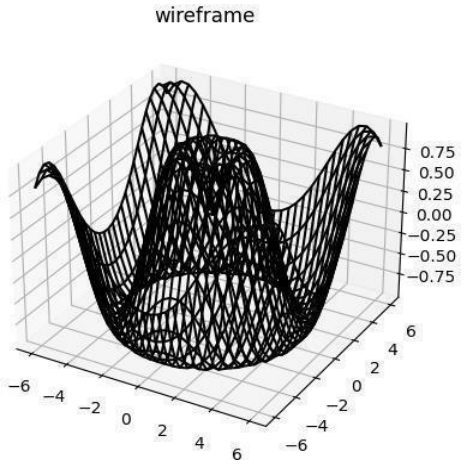
Sometimes the default viewing angle is not optimal, in which case we can use the view_init method to set the elevation and azimuthal angles.

ax.view_init(60, 35) fig

Wire frames and Surface Plots

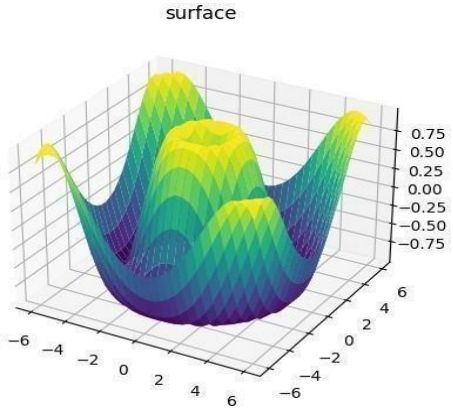
- Two other types of three-dimensional plots that work on gridded data are wireframes and surface plots.
- These take a grid of values and project it onto the specified threedimensional surface, and can make the resulting three-dimensional forms quite easy to visualize.

```
import numpy as np
import matplotlib.pyplot as plt from mpl_toolkits
 import mplot3d fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_wireframe(X, Y, Z, color='black')
ax.set_title('wireframe');
plt.show()
```



- A surface plot is like a wireframe plot, but each face of the wireframe is a filled polygon.
- Adding a colormap to the filled polygons can aid perception of the topology of the surface being visualized

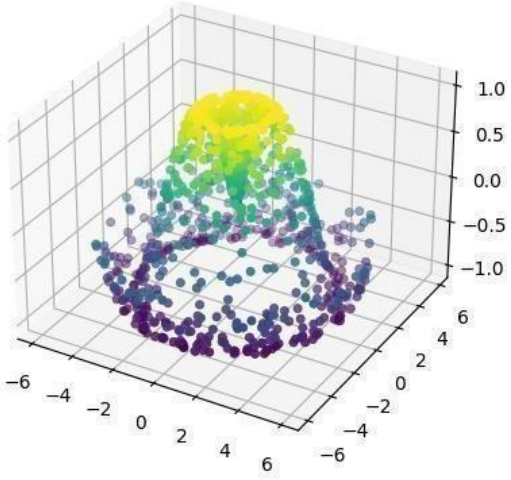
```
import numpy as np
import matplotlib.pyplot as plt from mpl_toolkits
import mplot3d ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
cmap='viridis', edgecolor='none') ax.set_title('surface')
plt.show()
```



Surface Triangulations

- For some applications, the evenly sampled grids required by the preceding routines are overly restrictive and inconvenient.
- In these situations, the triangulation-based plots can be very useful.

```
import numpy as np
import matplotlib.pyplot as plt from mpl_toolkits
import mplot3d
theta = 2 * np.pi * np.random.random(1000) r = 6 * np.random.
random(1000)
x = np.ravel(r * np.sin(theta))
y = np.ravel(r * np.cos(theta))
z = f(x, y)
ax = plt.axes(projection='3d')
ax.scatter(x, y, z, c=z, cmap='viridis', linewidth=0.5)
```



Geographic Data with Basemap

- One common type of visualization in data science is that of geographic data.
- Matplotlib's main tool for this type of visualization is the Basemap toolkit, which is one of several Matplotlib toolkits that live under the mpl_toolkits namespace.
- Basemap is a useful tool for Python users to have in their virtual toolbelts
- Installation of Basemap. Once you have the Basemap toolkit installed and imported, geographic plots also require the PIL package in Python 2, or the pillow package in Python 3.

```
import numpy as np
import matplotlib.pyplot as plt

from mpl_toolkits.basemap import Basemap plt.figure(figsize=(8, 8))

m = Basemap(projection='ortho', resolution=None, lat_0=50, lon_0=-100)
m.bluemarble(scale=0.5);
```

- Matplotlib axes that understands spherical coordinates and allows us to easily over-plot data on the map
- We'll use an etopo image (which shows topographical features both on land and under the ocean) as the map background

Program to display particular area of the map with latitude and longitude lines

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.basemap
import Basemap from itertools import chain
fig = plt.figure(figsize=(8, 8))
m = Basemap(projection='lcc', resolution=None,
width=8E6, height=8E6,
lat_0=45, lon_0=-100,) m.etopo(scale=0.5, alpha=0.5)
def draw_map(m, scale=0.2):
# draw a shaded-relief image m.shadedrelief(scale=scale)
# lats and longs are returned as a dictionary
lats = m.drawparallels(np.linspace(-90, 90, 13))
lons = m.drawmeridians(np.linspace(-180, 180, 13))
# keys contain the plt.Line2D instances
lat_lines = chain(*(tup[1][0] for tup in lats.items()))
lon_lines = chain(*(tup[1][0] for tup in lons.items()))
all_lines = chain(lat_lines, lon_lines)
# cycle through these lines and set the desired style for line in all_lines:
line.set(linestyle='-', alpha=0.3, color='r')
```

Map Projections

The Basemap package implements several dozen such projections, all referenced by a short format code. Here we'll briefly demonstrate some of the more common ones.

- Cylindrical projections
- Pseudo-cylindrical projections
- Perspective projections
- Conic projections

Cylindrical projection

- The simplest of map projections are cylindrical projections, in which lines of constant latitude and longitude are mapped to horizontal and vertical lines, respectively.

- This type of mapping represents equatorial regions quite well, but results in extreme distortions near the poles.
- The spacing of latitude lines varies between different cylindrical projections, leading to different conservation properties, and different distortion near the poles.
- Other cylindrical projections are the Mercator (projection='merc') and the cylindrical equal-area (projection='cea') projections.
- The additional arguments to Basemap for this view specify the latitude (lat) and longitude (lon) of the lower-left corner (llcrnr) and upper-right corner (urcrnr) for the desired map, in units of degrees.

import numpy as np

import matplotlib.pyplot as plt

from mpl_toolkits.basemap import Basemap

fig = plt.figure(figsize=(8, 6), edgecolor='w')
m = Basemap(projection='cyl', resolution=None, llcrnrlat=-90, urcrnrlat=90,
llcrnrlon=-180, urcrnrlon=180,) draw_map(m)

Pseudo-cylindrical projections

- Pseudo-cylindrical projections relax the requirement that meridians (lines of constant longitude) remain vertical; this can give better properties near the poles of the projection.
- The Mollweide projection (projection='moll') is one common example of this, in which all meridians are elliptical arcs
- It is constructed so as to
- preserve area across the map: though there are distortions near the poles, the area of small patches reflects the true area.
- Other pseudo-cylindrical projections are the sinusoidal (projection='sinu') and Robinson (projection='robin') projections.
- The extra arguments to Basemap here refer to the central latitude (lat_0) and longitude (lon_0) for the desired map.

import numpy as np
import matplotlib.pyplot as plt

from mpl_toolkits.basemap import Basemap fig = plt.figure(figsize=(8, 6), edgecolor='w')

m = Basemap(projection='moll', resolution=None, lat_0=0, lon_0=0)
draw_map(m)

Perspective projections

- Perspective projections are constructed using a particular choice of perspective point, similar to if you photographed the Earth from a particular point in space (a point which, for some projections, technically lies within the Earth!).
- One common example is the orthographic projection (projection='ortho'), which shows one side of the globe as seen from a viewer at a very long distance.
- Thus, it can show only half the globe at a time.
- Other perspective-based projections include the gnomonic projection (projection='gnom') and stereographic projection (projection='stere').
- These are often the most useful for showing small portions of the map.

import numpy as np
import matplotlib.pyplot as plt

from mpl_toolkits.basemap

import Basemap fig = plt.figure(figsize=(8, 8))

m = Basemap(projection='ortho', resolution=None, lat_0=50, lon_0=0)
draw_map(m);

Conic projections

- A conic projection projects the map onto a single cone, which is then unrolled.
- This can lead to very good local properties, but regions far from the focus point of the cone may become very distorted.
- One example of this is the Lambert conformal conic projection (projection='lcc').
- It projects the map onto a cone arranged in such a way that two standard parallels (specified in Basemap by lat_1 and lat_2) have well-represented distances, with scale decreasing between them and increasing outside of them.
- Other useful conic projections are the equidistant conic (projection='eqdc') and the Albers equal-area (projection='aea') projection

import numpy as np
import matplotlib.pyplot as plt

from mpl_toolkits.basemap

import Basemap fig = plt.figure(figsize=(8, 8))

m = Basemap(projection='lcc', resolution=None,

lon_0=0, lat_0=50, lat_1=45, lat_2=55, width=1.6E7, height=1.2E7) draw_map(m)

Drawing a Map Background

The Basemap package contains a range of useful functions for drawing borders of physical features like continents, oceans, lakes, and rivers, as well as political boundaries such as countries and US states and counties.

The following are some of the available drawing functions that you may wish to explore using IPython's help features:

- Physical boundaries and bodies of water drawcoastlines() - Draw continental coast lines

drawlsmask() - Draw a mask between the land and sea, for use with projecting images on one or the other drawmapboundary() - Draw the map boundary, including the fill color for oceans
drawrivers() - Draw rivers on the map
fillcontinents() - Fill the continents with a given color; optionally fill lakes with another color

- Political boundaries

drawcountries() - Draw country boundaries drawstates() - Draw US state boundaries drawcounties() - Draw US county boundaries

- Map features

drawgreatcircle() - Draw a great circle between two points drawparallels() - Draw lines of constant latitude drawmeridians() - Draw lines of constant longitude drawmapscale() - Draw a linear scale on the map

- Whole-globe images

bluemarble() - Project NASA's blue marble image onto the map shadedrelief() - Project a shaded relief image onto the map etopo() - Draw an etopo relief image onto the map warpimage() - Project a user-provided image onto the map

Plotting Data on Maps

- The Basemap toolkit is the ability to over-plot a variety of data onto a map background.
- There are many map-specific functions available as methods of the Basemap instance. Some of these map-specific methods are:

contour()/contourf() - Draw contour lines or filled contours imshow() - Draw an image
pcolor()/pcolormesh() - Draw a pseudocolor plot for irregular/regular meshes plot() - Draw lines and/or markers

scatter() - Draw points with markers quiver() - Draw vectors
barbs() - Draw wind barbs drawgreatcircle() - Draw a great circle

Visualization with Seaborn

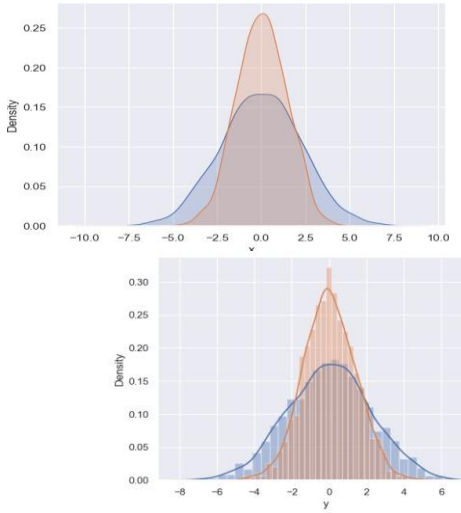
The main idea of Seaborn is that it provides high-level commands to create a variety of plot types useful for statistical data exploration, and even some statistical model fitting.

Histograms, KDE, and densities

- In statistical data visualization, all you want is to plot histograms and joint distributions of variables. We have seen that this is relatively straightforward in Matplotlib
- Rather than a histogram, we can get a smooth estimate of the distribution using a kernel density estimation, which Seaborn does with sns.kdeplot

import pandas as pd import seaborn as sns
data = np.random.multivariate_normal([0, 0], [[5, 2], [2, 2]], size=2000)
data = pd.DataFrame(data, columns=['x', 'y'])
for col **in** 'xy':

sns.kdeplot(data[col], shade=True)



- Histograms and KDE can be combined using distplot

sns.distplot(data['x'])
sns.distplot(data['y']);

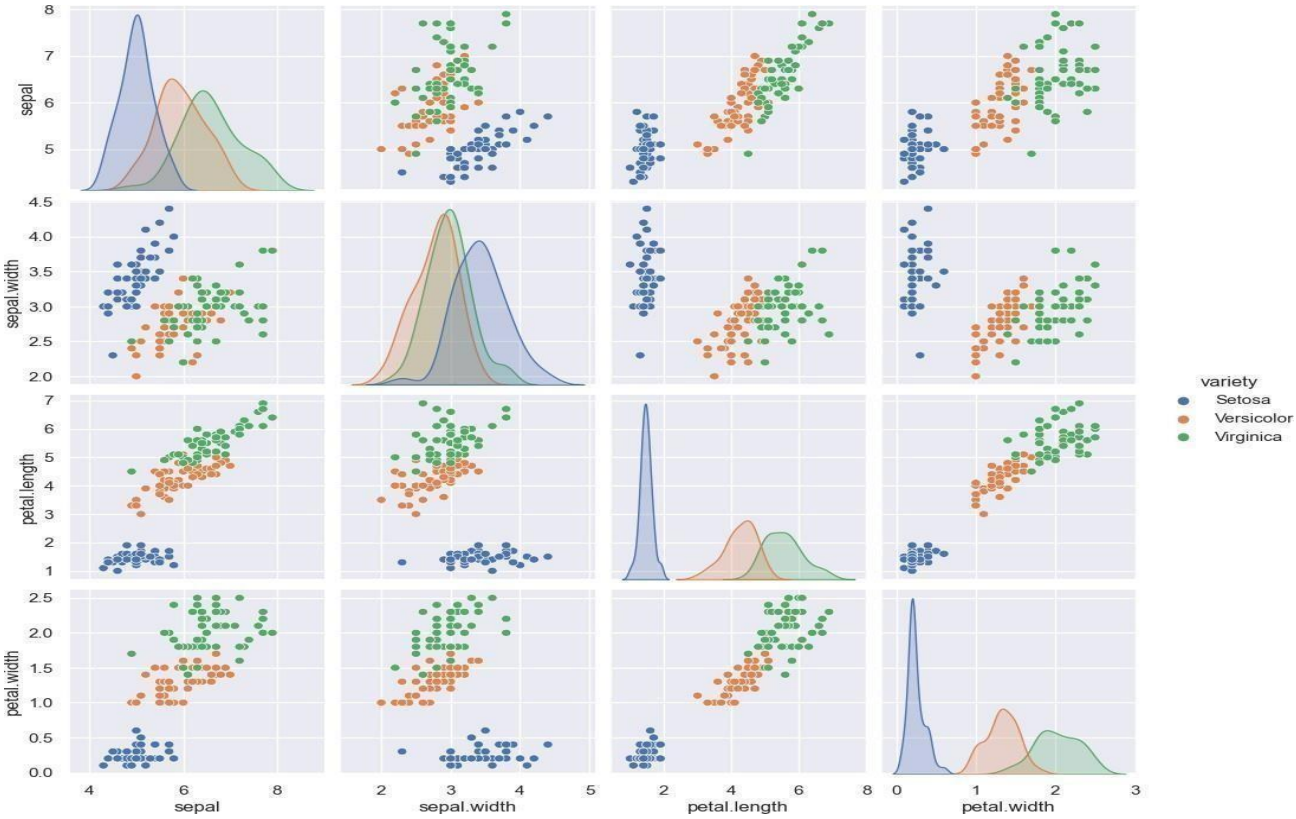
- If we pass the full two-dimensional dataset to kdeplot, we will get a two-dimensional visualization of the data.
- We can see the joint distribution and the marginal distributions together using sns.jointplot.

Pair plots

When you generalize joint plots to datasets of larger dimensions, you end up with pair plots. This is very useful for exploring correlations between multidimensional data, when you'd like to plot all pairs of values against each other.

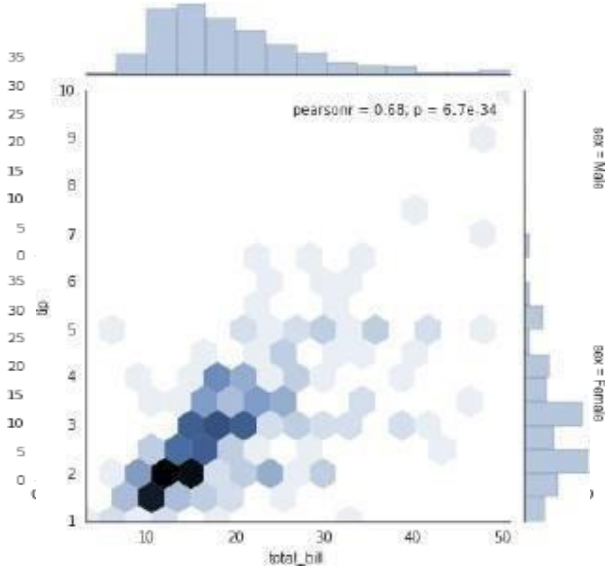
We'll demo this with the Iris dataset, which lists measurements of petals and sepals of three iris species:

import seaborn as sns
iris = sns.load_dataset("iris") sns.pairplot(iris, hue='species', size=2.5);



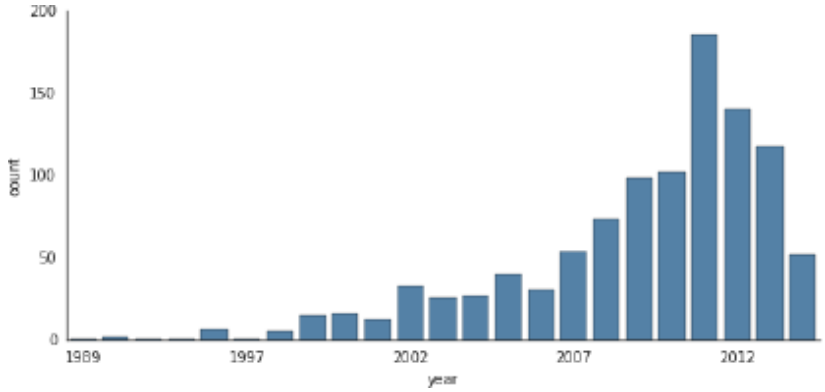
Faceted histograms

- Sometimes the best way to view data is via histograms of subsets. Seaborn's FacetGrid makes this extremely simple.
- We'll take a look at some data that shows the amount that restaurant staff receive in tips based on various indicator data



Bar plots

Time series can be plotted with sns.factorplot.



Multiple plots in one window

Multiple plots in one window, often referred to as subplots, allow for the display of several distinct graphs within a single figure. This is particularly useful for comparing different datasets, visualizing various aspects of the same data, or presenting related information in a consolidated view.

How it works:

Instead of creating separate windows for each plot, a single figure is divided into a grid of smaller plotting areas. Each of these areas can then host an individual plot.

Key concepts:

- **Figure:** The overall window or canvas where all the plots are displayed.
- **Subplot/Axes:** Individual plotting areas within the figure, each capable of holding a distinct plot (e.g., a line plot, scatter plot, bar chart).

- **Grid Layout:** The arrangement of subplots within the figure, typically defined by the number of rows and columns.

Module Used

- **lattice:** The lattice package uses grid package to provide better relationships between the data. It is an add-on package for the implementation of Trellis graphics (graphics that shows relationships between variables conditioned together).
- **gridExtra:** This package provides multiple functions that define various grids to be used to arrange multiple plots.

Both of these packages have to be installed on the system explicitly. To begin with, we need to set up a dataframe, that contains data to be plotted on the x and y-axis and also another column that can be used to differentiate between lattices. For each unique value of this column, a different lattice plot will be generated.

Individual lattice plots are created via xyplot() function, this function produces a scatter plot. For each row, a certain condition decides in which lattice it should be placed.

Syntax:

xyplot(formula, data, ..)

Parameter:

- *formula: To specify certain condition*
- *data: dataframe to be plotted*

Once all the lattice plots are ready it is time to arrange them in one window for this arrange() function of the gridExtra package is employed.

arrange() function in R Language is used for reordering of table rows with the help of column names as expression passed to the function.

Syntax: *arrange(x, expr)*

Parameters:

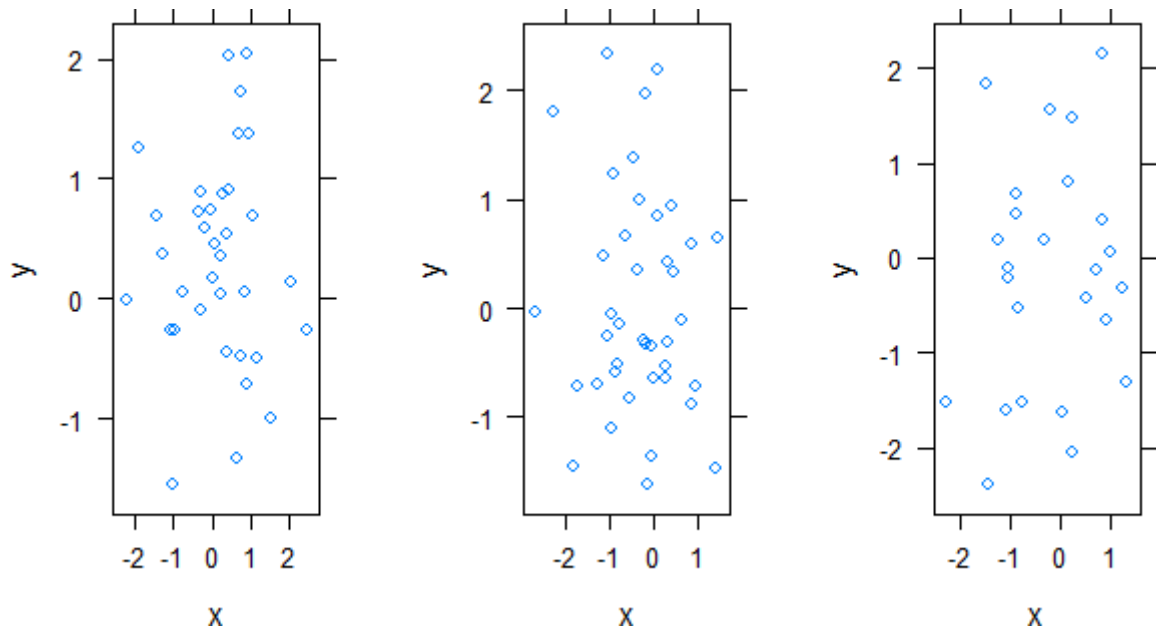
x: *data set to be reordered*

expr: *logical expression with column name*

Example:

```
library(lattice)
library(gridExtra)

df<-data.frame(x = rnorm(100),
               y = rnorm(100),
               z = c(rep("A", 35),
                     rep("B", 40),
                     rep("C", 25))
)

df

lat1 <- xyplot(y~x,df[df$z == 'A',])
lat2 <- xyplot(y~x,df[df$z == 'B',])
lat3 <- xyplot(y~x,df[df$z == 'C',])

grid.arrange(lat1, lat2, lat3, ncol = 3)
```

Output:



OVERVIEW OF POWER BI

Microsoft Power BI is a collection of apps, software services and connectors that come together to turn unrelated data into visually impressive and interactive insights. Power BI can work with simple data sources like Microsoft Excel and complicated ones like cloud-based or on-premises hybrid Data warehouses. Power BI has the capabilities to easily connect to your data sources, visualise and share and publish your findings with anyone and everyone.

Power BI is simple and fast enough to connect to an Excel workbook or a local database. It can also be robust and enterprise-grade, ready for extensive modeling and real time analytics. This means it can be used in a variety of environments from a personal report and visualisation tool to the analytics and decision engine behind group projects, divisions, or entire corporations.

As Power BI is a Microsoft product and has built in connections to Excel, there are many functions that will be familiar to an Excel user.

1.2 The parts of Power BI

Power BI constitutes of a Microsoft Windows desktop application called Power BI Desktop, an online SaaS (Software as a Service) called Power BI Service and a mobile Power BI apps that can be accessed from Windows phones and tablets, and also available on Apple iOS and Google Android devices.

These three elements— **Desktop**, the **Service**, and **Mobile** apps - are the backbone of the Power BI system and lets users create, share and consume the actionable insights in the most effective way.

1.3 Use of Power BI and roles

The use of Power BI could depend on the role that you are in. For example: if you are the stakeholder of a project, then you might want to use **Power BI Service** or the Mobile **app** to have a glance at how the business is performing. But on the other hand, if you are a developer, you would be using **Power BI Desktop** extensively to publish Power BI desktop reports to the Power BI Service.

In the upcoming modules we would be discussing about these three components - **Desktop**, **Service** and **Mobile** apps - in more detail.

1.4 Power BI flow

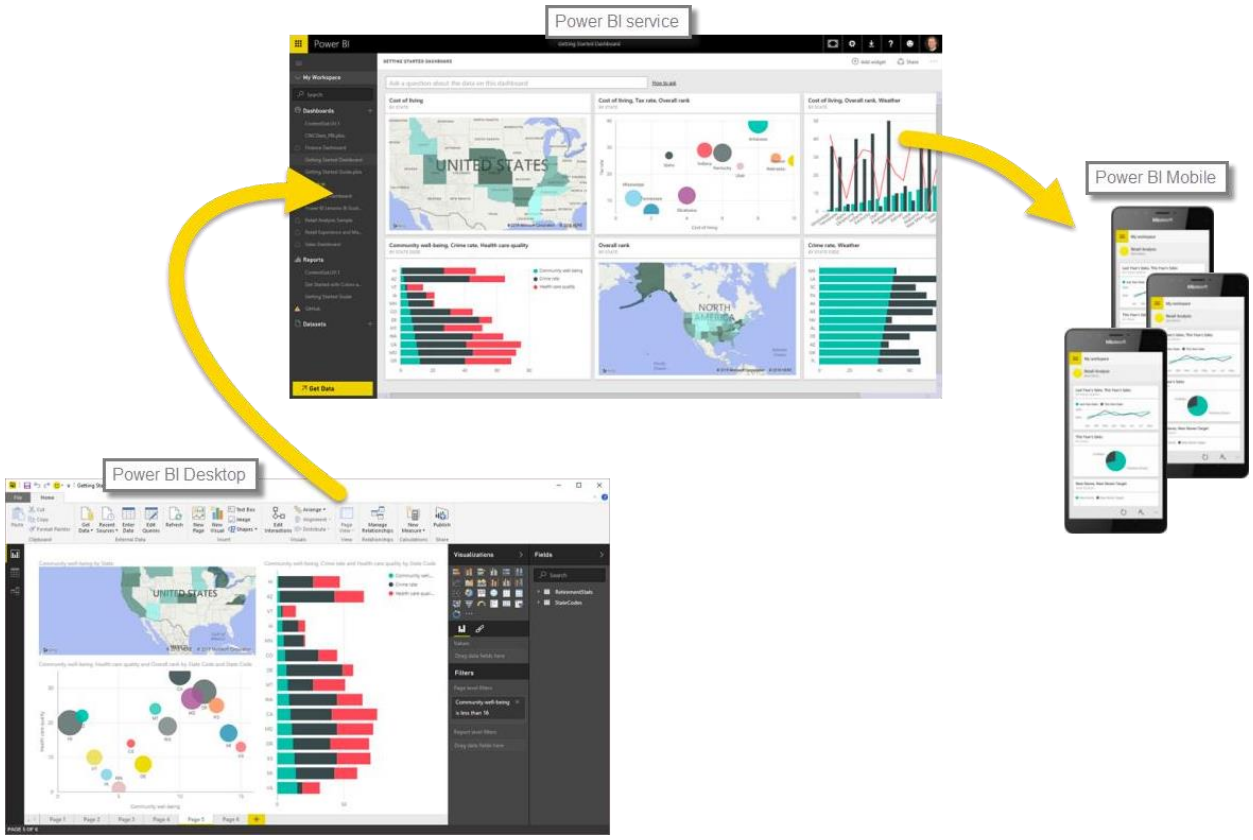
Generally, the flow starts at the Power BI Desktop, where a report is created. This created report can be published to the Power BI Service and finally shared so that the users can use it from the Mobile apps.

This is the most common approach for sharing reports. There are other approaches but we will stick to this flow for this entire tutorial to help learn the different aspects of Power BI.

1.5 Use Power BI

The **common** flow of activity in Power BI looks like this:

1. Bring data into Power BI Desktop, and create a report.
2. Publish to the Power BI service, where you can create new visualizations or build dashboards.
3. Share dashboards with others, especially people who are on the go.
4. View and interact with shared dashboards and reports in Power BI Mobile apps.



Depending on the user role, the user might spend most of the time in one of the three components than the other.

1.6 Building blocks of Power BI

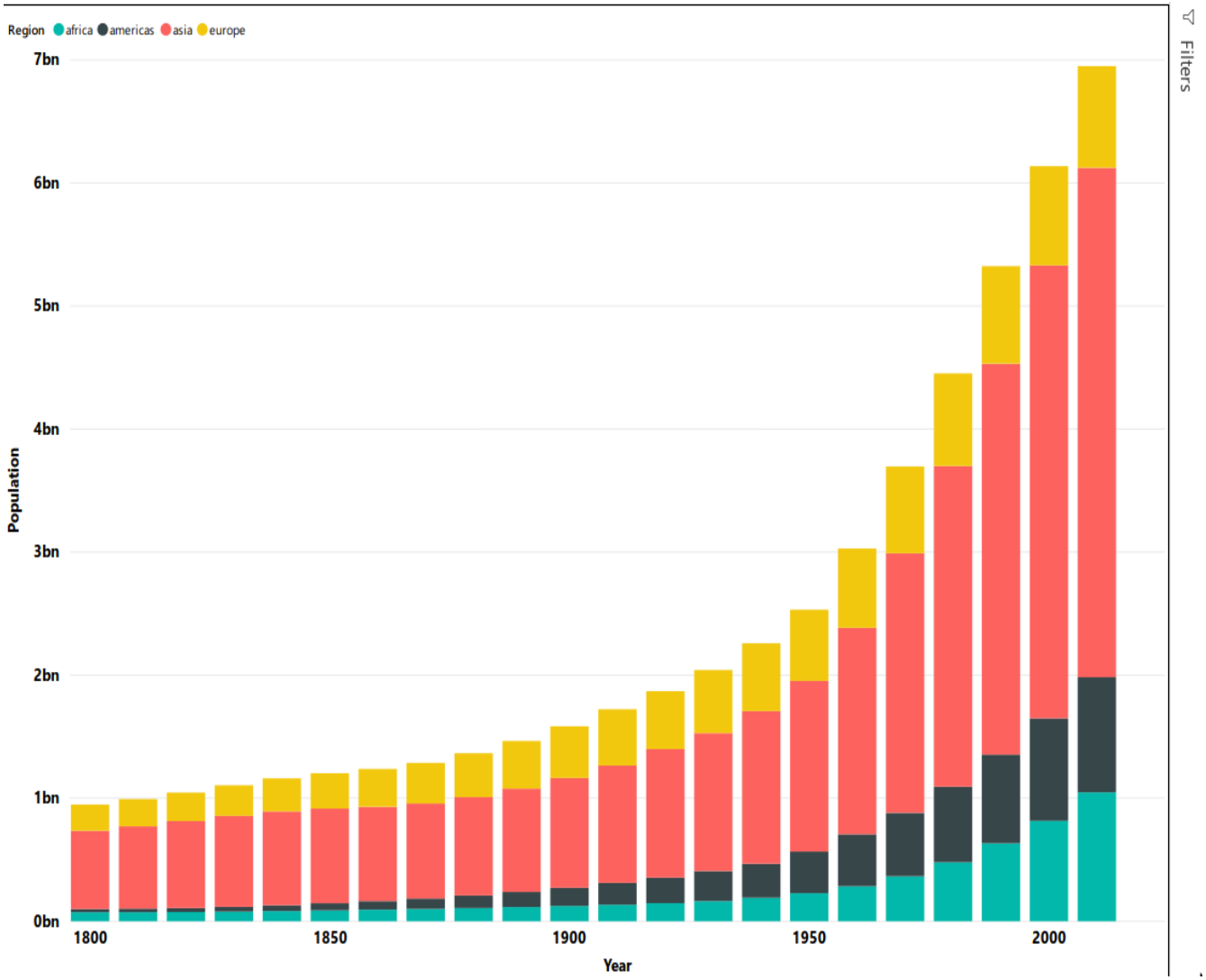
The basic building blocks in Power BI are:

- Visualizations

- Datasets
- Reports
- Dashboards
- Tiles

1.6.1 Visualizations

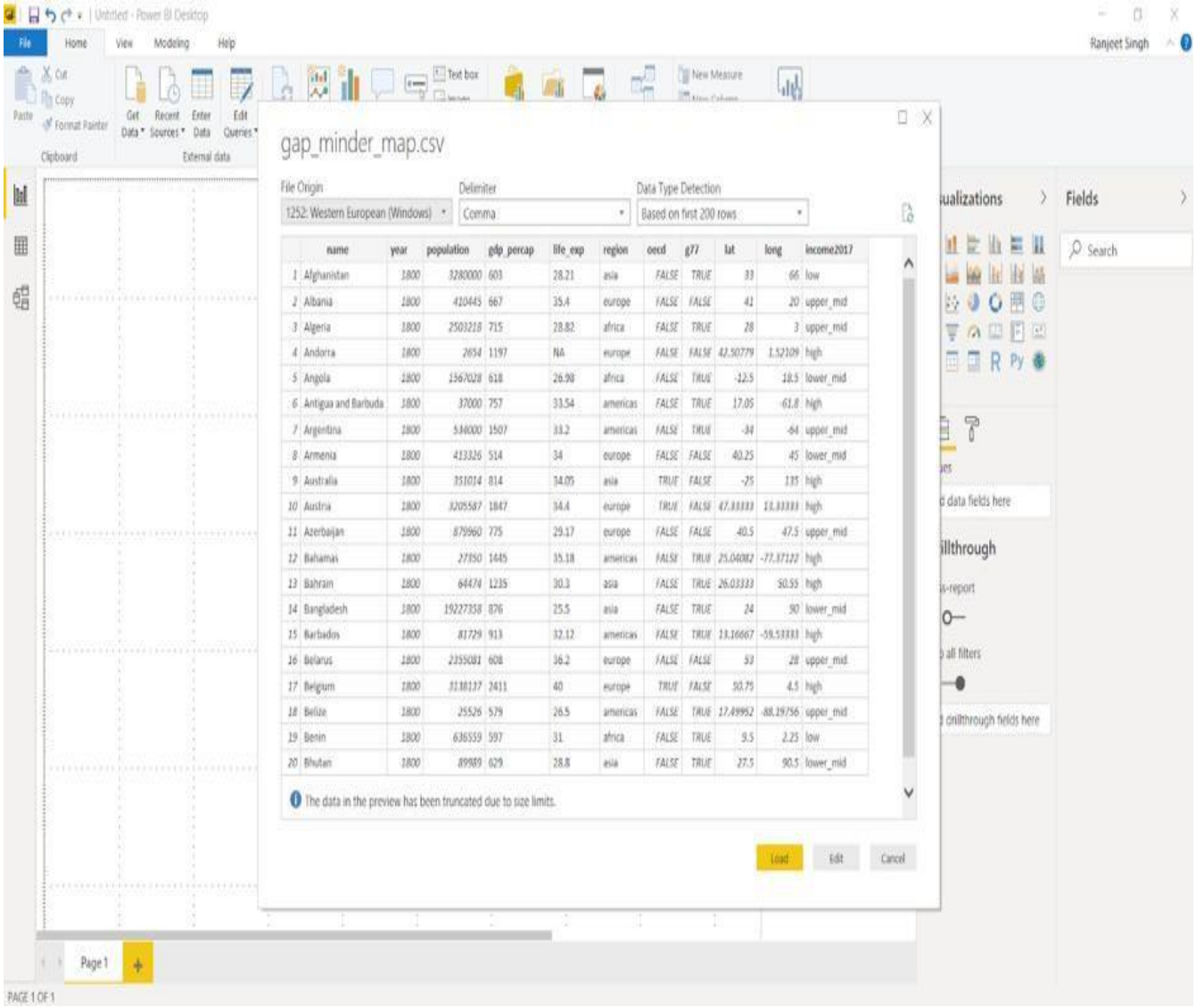
A visualization is a representation of data in a visual format. It could be a line chart, a bar graph, a color coded map or any visual way to present the data.



Visualizations can be a simple number representing a significant calculation or it could be more complex like multiple charts showing the proportion of users participating in a survey. The main idea of visualisation is to show the data in a way that tells the story that is lying underneath it. Like the saying goes: a picture says a thousand words.

1.6.2 Datasets

A **dataset** is a collection of data that Power BI uses to create its visualizations. You can have a simple dataset that's based on a single table from a Microsoft Excel workbook, similar to what's shown in the following image.



Dataset can also be a combination of many different sources, which can be filtered using Power BI and combined into one to use.

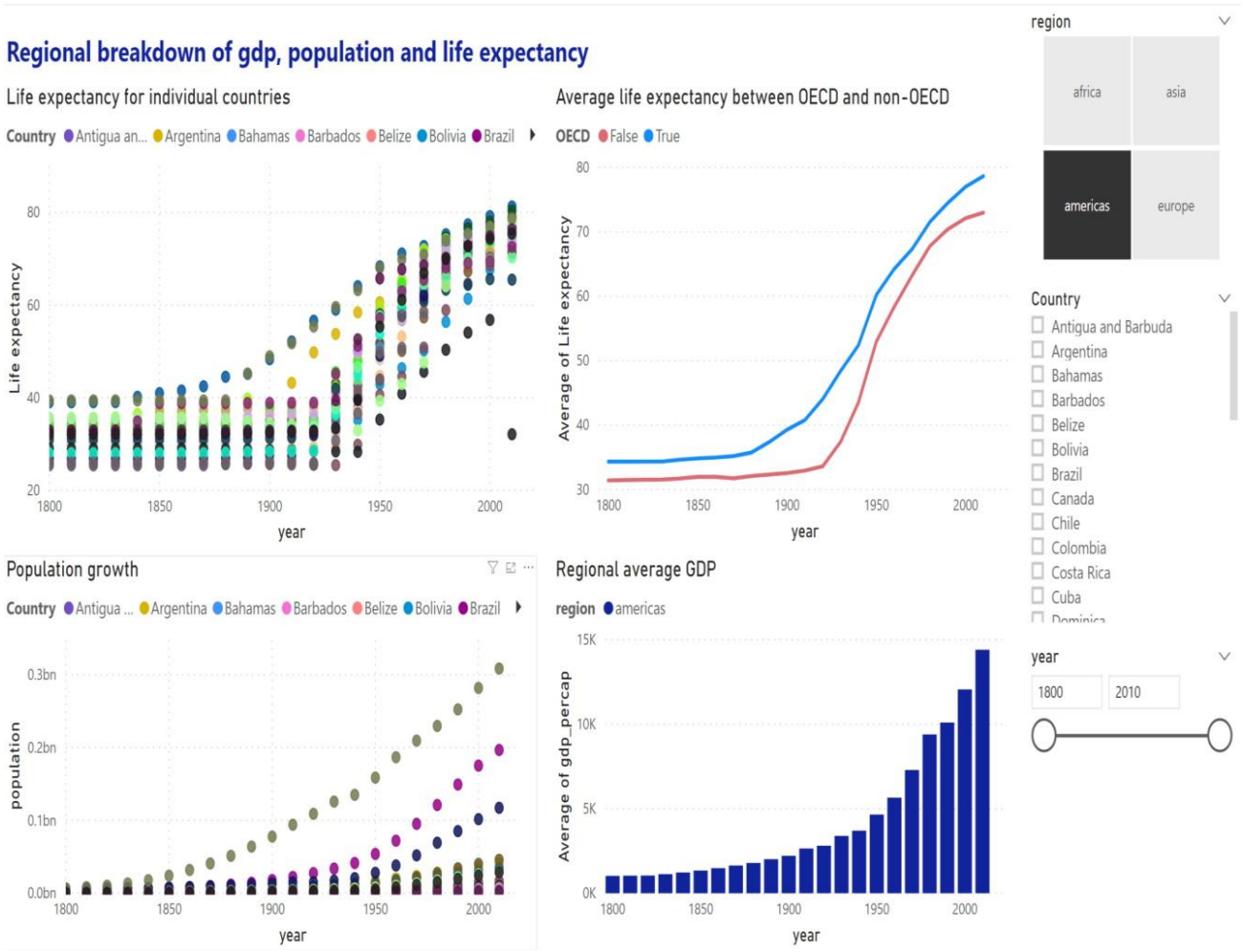
For example: One data source contains countries and locations in the form of latitude and longitude. Another data source contains demographics of these countries like population and GDP. Power BI can combine these two data sources into one dataset which can be used for visualizations.

An important feature of Power BI is the ability to connect to various data sources using its connectors. Whether the data you want is in Excel or a Microsoft SQL Server database, in Azure or Oracle, or in a service like Facebook, Salesforce, or MailChimp, Power BI has built-in data connectors that let you easily connect to that data, filter it if necessary, and bring it into your dataset.

After you have a dataset, you can begin creating visualizations that show different portions of it in different ways, and gain insights based on what you see. That is where reports come in.

1.6.3 Reports

In Power BI, a **Report** is a collection of visualizations that appear together on one or more pages. A report in Power BI is a collection of items that are related to each other. We will be working with the gapminder data to create the report below that looks at the GDP, population and life expectancy by global regions.



Reports let us create and structure visualizations on pages based on the way the we want to tell the story.

1.6.4 Dashboards

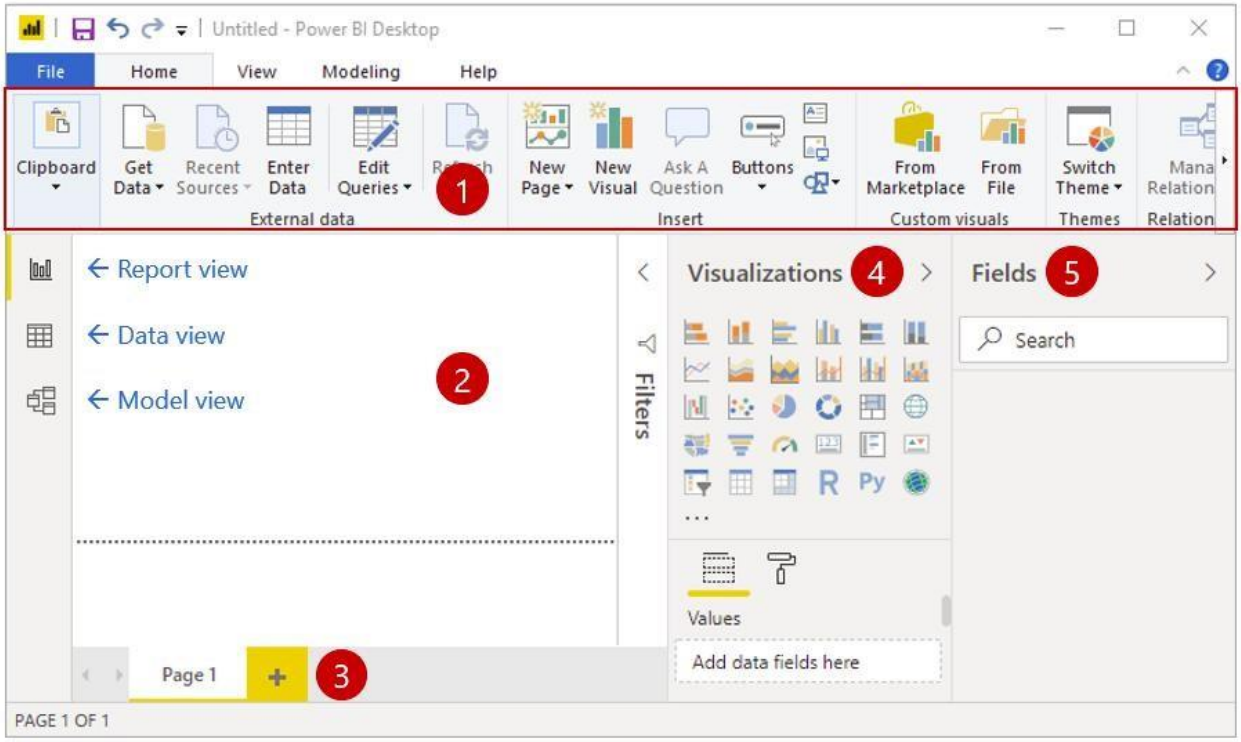
A Power BI dashboard is a collection of visuals from a single page that you can share with others. Often it is a selected group of visuals that provide quick insight into the data or story you are trying to present.

A dashboard must fit on a single page, often called a canvas (the canvas is the blank backdrop in Power BI Desktop or the service, where you put visualizations). Think of it like the canvas that an artist or painter uses — a workspace where you create, combine, and rework interesting and compelling visuals. You can share dashboards with other users or groups, who can then interact with your dashboards when they're in the Power BI service or on their mobile device.

1.7 Power BI Services

1.7.1 Overview of Power BI Desktop

Power BI Desktop is a free application for PCs that lets you gather, transform, and visualize your data. In this module, you'll learn how to find and collect data from different sources and how to clean or transform it. You'll also learn tricks to make data-gathering easier. Power BI Desktop and the Power BI Service work together. You can create your reports and dashboards in Power BI Desktop, and then publish them to the Power BI Service for others to consume.



1. **Ribbon** - Displays common tasks that are associated with reports and visualizations.
2. **Report view, or canvas** - Where visualizations are created and arranged. You can switch between **Report**, **Data**, and **Model** views by selecting the icons in the left column.
3. **Pages tab** - Located along the bottom of the page, this area is where you would select or add a report page.
4. **Visualizations pane** - Where you can change visualizations, customize colors or axes, apply filters, drag fields, and more.
5. **Fields pane** - Where query elements and filters can be dragged onto the **Report** view or dragged to the **Filters** area of the Visualizations pane.

Working in Power BI desktop and creating visuals will be the focus of todays session.

1.7.2 Power BI Service

Power BI Service is the online component of Power BI where you can publish your dashboards and reports. You can also view other dashboard and reports that have been shared with you. Monash staff members have access to this, all you need to do is log in with you Monash account at Power BI service. We will cover this in more detail later in the workshop. One thing to be aware of is that once the report is published, the report and underlying data will be stored on Microsoft servers and is not private. Be aware of any privacy or confidentiality issues with your data and we suggest using another approach or tool if cannot anonymise your data.

1.8 Power BI licence

We will be covering the free version of Power BI desktop in this session. This version allows you to use all the functionalities of the desktop version such as loading, analysing and visualising data. You can create as many visuals and reports as you would like with the free license. However publishing and sharing your reports and visuals online with another person or publically requires a Power BI license. This means that you can still share your report and data files between users and computers through other means, like a shared drive, and only require one user with a Power BI license to publish and share the final report online. Please contact the Monash Business Intelligence team for more information on licensing.