

## PYTHON – INSTALLATION

### Contents

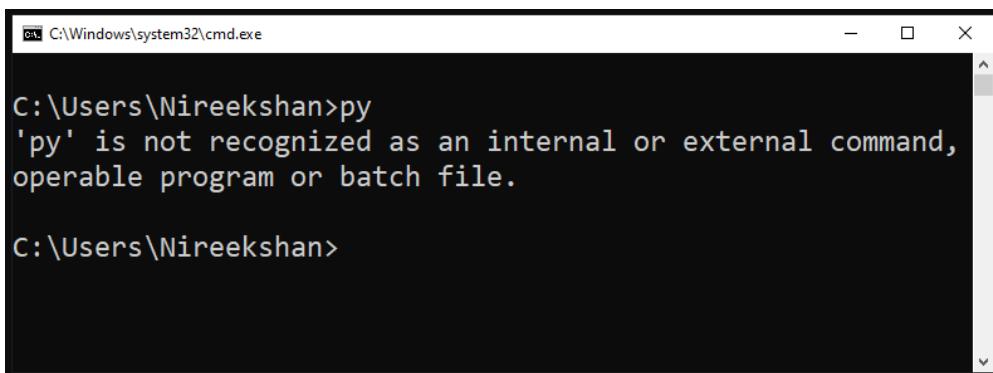
<b>1. Checking python software installed or not in laptop/computer .....</b>	<b>2</b>
<b>2. Download python software .....</b>	<b>2</b>
<b>3. Important step while installing software .....</b>	<b>2</b>
<b>4. Check the python version.....</b>	<b>5</b>

## Data Science – Python Installation

### PYTHON – INSTALLATION

#### 1. Checking python software installed or not in laptop/computer

- ✓ We can check python software is installed or not in laptop/computer.
- ✓ Open a command prompt in your laptop/computer.
- ✓ Type **py** command in command prompt and enter then check it.



C:\Windows\system32\cmd.exe  
C:\Users\Nireekshan>py  
'py' is not recognized as an internal or external command,  
operable program or batch file.  
C:\Users\Nireekshan>

- ✓ If python is not installed then we will get response as above screenshot.

#### 2. Download python software

- ✓ We can download python software from official website, we can find url in below,

<https://www.python.org/downloads/>

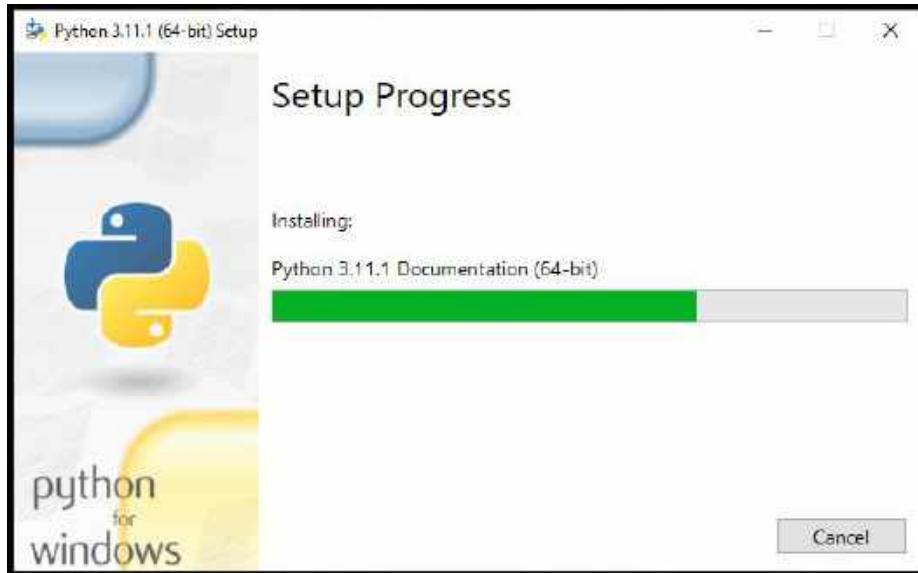
#### 3. Important step while installing software

- ✓ Once python software downloaded then we can install by clicking the software.
- ✓ While installing we should check the checkbox of,
  - Add python 3.11.1 to PATH
- ✓ You can follow the below screenshot

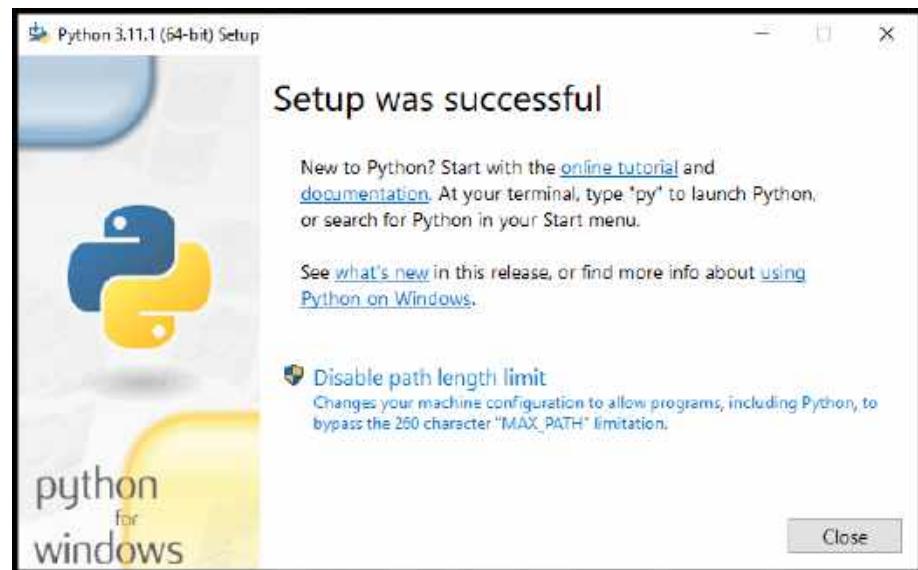
## Data Science – Python Installation



## Data Science – Python Installation

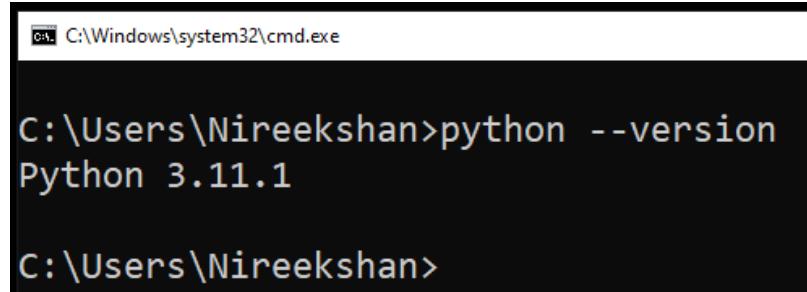


- ✓ Once we check the checkbox then we need to click on **Install Now**.
- ✓ Finally we will get, **Setup was successful** screen



### 4. Check the python version

- ✓ Open a **NEW** command prompt in your laptop/computer.
- ✓ Type **python --version** command in command prompt and check it.



```
C:\Windows\system32\cmd.exe
C:\Users\Nireekshan>python --version
Python 3.11.1
C:\Users\Nireekshan>
```

# Data Science - Fundamentals

---

## 1. DATA SCIENCE FUNDAMENTALS

### Table of Contents

<b>1. Data Introduction .....</b>	2
<b>2. What is Data? .....</b>	2
<b>3. Information.....</b>	2
<b>4. Data Processing .....</b>	2
<b>5. Data, Information Examples .....</b>	3
5.1 Example.....	3
5.2 Example.....	3
<b>6. What is Data Science? .....</b>	4
6.1 Definition 1.....	4
6.2 Definition 2.....	4
6.3 Scientist.....	4
6.4 Nature of Scientist .....	4
6.5 Process behind the scientist .....	4
<b>7. BigData.....</b>	6
<b>8. Data Measurement table .....</b>	6
<b>9. Types of BigData.....</b>	7
9.1 Structured .....	7
9.2 Semi structured.....	7
9.3 Unstructured data.....	7
<b>10. V specialty in BigData.....</b>	9
10.1 Volume .....	9
10.2 Variety .....	9
10.3 Velocity .....	9
<b>11. Data Analytics.....</b>	10
<b>12. Different types of Analytics .....</b>	10
12.1 Descriptive Analytics.....	10
12.2 Diagnostic Analytics .....	10
12.3 Predictive Analytics.....	10
12.4 Prescriptive Analytics .....	11

# Data Science - Fundamentals

## 1. DATA SCIENCE FUNDAMENTALS

### 1. Data Introduction

- ✓ Currently we are living in the data world.
- ✓ Everyone is communicating by using devices and social networks, due to this huge amount of data is generating.
- ✓ All applications are generating data.
  - Ecommerce applications
  - Banking applications
  - Social network etc.

### 2. What is Data?

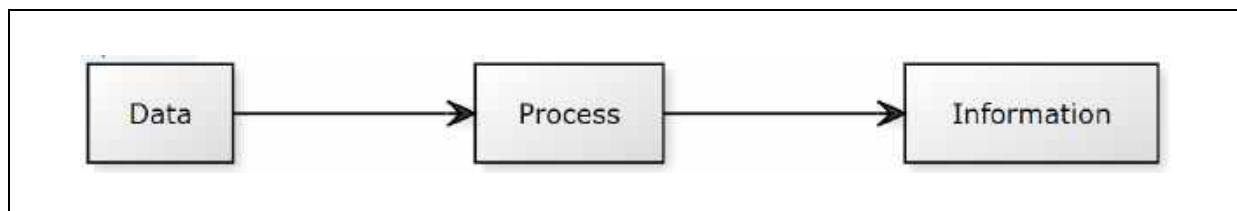
- ✓ Data is a collection of Facts.
- ✓ Facts,
  - Numbers
  - Alphabets
  - Alphanumeric
  - Symbols
  - Images
  - Audio
  - Video & etc

### 3. Information

- ✓ Data we may not understand clearly.
- ✓ The processed data is called as information.

### 4. Data Processing

- ✓ Converting the raw data into meaningful information



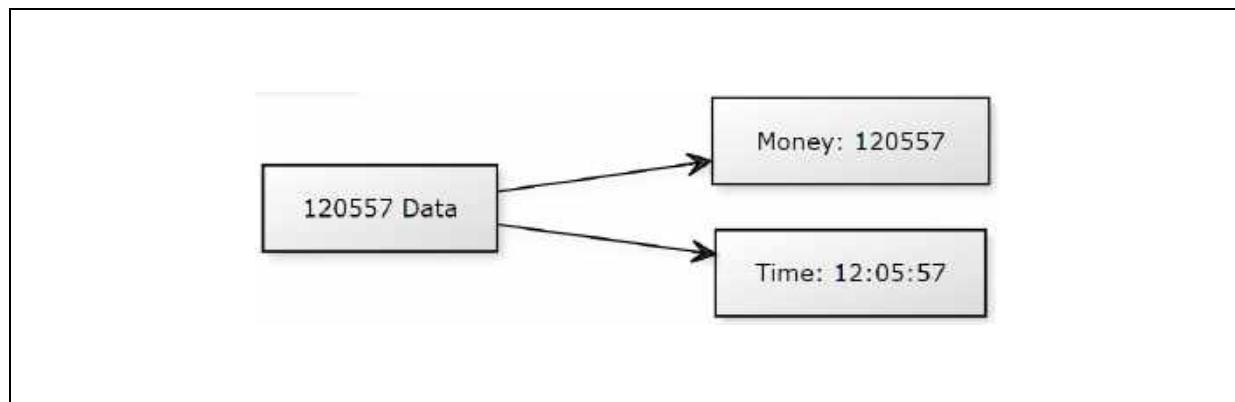
## Data Science - Fundamentals

### 5. Data, Information Examples

#### 5.1 Example

Data	Information
Daniel, Data Scientist, age is sweet sixteen, Bengaluru, India	Name : Daniel Profession : Data Scientist Age : 16 Location : Bengaluru Country : India

#### 5.2 Example



# Data Science - Fundamentals

---

## 6. What is Data Science?

### 6.1 Definition 1

- ✓ Data science is a combination of scientific methods, algorithms to extract knowledge and insights from data.

### 6.2 Definition 2

- ✓ Data science is all about how we take data,
  - Process data
  - Understand the past/present conditions
  - Decision making process
  - Predicting the future results
  - Create new industries/products condition

### 6.3 Scientist

- ✓ Scientist is a person who is having expert knowledge on specific scenarios.

### 6.4 Nature of Scientist

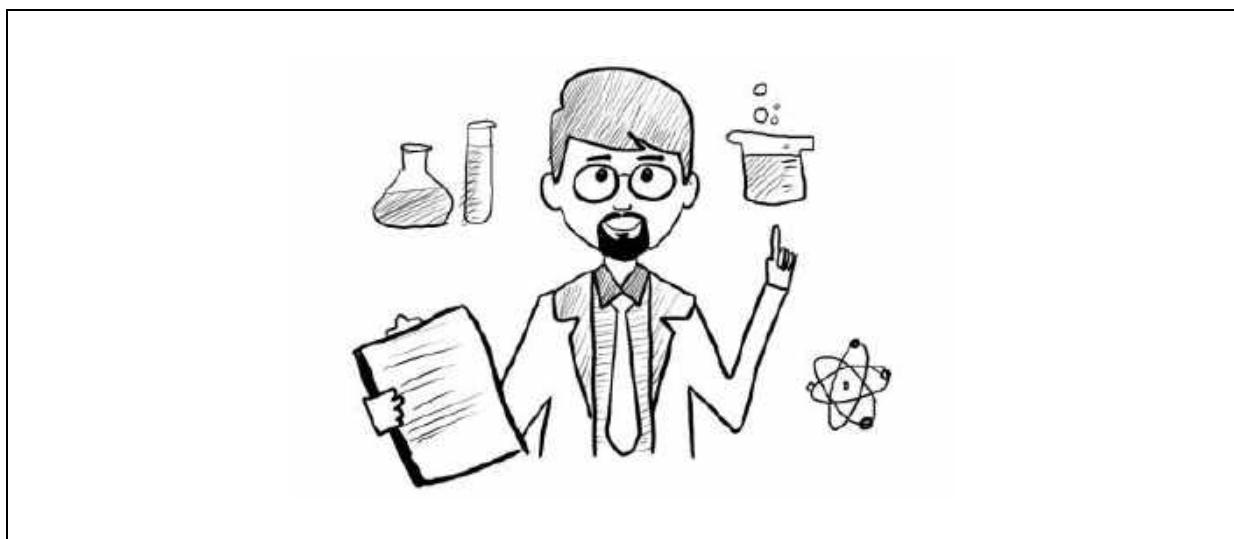
- ✓ A scientist used to conduct more research to achieve results.

### 6.5 Process behind the scientist

- ✓ In simple words a scientist is someone who,
  - Systematically gathers information
  - Doing research
  - Getting evidence
  - Test the evidence
  - Gaining knowledge
  - Understanding the knowledge.
  - Sharing knowledge

## Data Science - Fundamentals

---



**7. BigData**

- ✓ Extremely large data sets is called as a BigData.

**8. Data Measurement table**

<b>Data Measurement Chart</b>		
(Types of Various Units of Memory)		
Bit	----	Single Binary Digit (1 or 0)
Byte	----	8 Bits
Kilo Byte	(KB)	1,024 Bytes
Mega Byte	(MB)	1,024 Kilobytes
Giga Byte	(GB)	1,024 Megabytes
Tera Byte	(TB)	1,024 Gigabytes
Peta Byte	(PB)	1,024 Terabytes
Exa Byte	(EB)	1,024 Petabytes
Zetta Byte	(ZB)	1,024 Exabytes
Yotta Byte	(YB)	1,024 Zettabyte

## 9. Types of BigData

- ✓ There are mainly 3 types of big data.
  - Structured
  - Semi-structured
  - Unstructured

### 9.1 Structured

- ✓ Structured data will be stored in table formats like rows and columns.
- ✓ It is very easy to understand.
- ✓ Examples like, a table in database, excel file, csv file.

### 9.2 Semi structured

- ✓ Semi structured may not in proper format
- ✓ This type of data having some properties or tags.
- ✓ Example
  - Xml file
  - JSON file

### 9.3 Unstructured data

- ✓ Unstructured data have no proper format.
- ✓ In this world mostly unstructured data exists.
- ✓ Example
  - log file
  - audio, video
  - e-mail & etc

## Data Science - Fundamentals

Structured data				Semi-structured data	Unstructured data
1	John	18	B.Sc.	<pre>&lt;University&gt;   &lt;Student ID="1"&gt;     &lt;Name&gt;John&lt;/Name&gt;     &lt;Age&gt;18&lt;/Age&gt;     &lt;Degree&gt;B.Sc.&lt;/Degree&gt;   &lt;/Student&gt;   &lt;Student ID="2"&gt;     &lt;Name&gt;David&lt;/Name&gt;     &lt;Age&gt;31&lt;/Age&gt;     &lt;Degree&gt;Ph.D. &lt;/Degree&gt;   &lt;/Student&gt;   ... &lt;/University&gt;</pre>	The university has 5600 students. John's ID is number 1, he is 18 years old and already holds a B.Sc. degree. David's ID is number 2, he is 31 years old and holds a Ph.D. degree. Robert's ID is number 3, he is 51 years old and also holds the same degree as David, a Ph.D. degree.

## Data Science - Fundamentals

### 10. V specialty in BigData

#### 10.1 Volume

- ✓ Volume of the data means, size of the data.
- ✓ Based on size only we can measure as the data is in big data or not.

#### 10.2 Variety

- ✓ Variety of data means, the generated data may exist in different formats like
- ✓ It can be,
  - Structured
  - Semi-structured
  - Unstructured.

#### 10.3 Velocity

- ✓ Velocity of data means, speed of the data generating and processing from the source points.

#### In short

- Volume = Size
- Variety = Types
- Velocity = Speed

## 11. Data Analytics

- ✓ All companies are analysing big data to improve the business.
- ✓ From the result of analysis, companies used to reach customers to understand them clearly.

### 11.1 Every company's target

- ✓ Ultimate target of any company is, wanted to stand in best position, and should be competitive with other companies.

## 12. Different types of Analytics

- ✓ There are mainly four types of analytics,
  1. Descriptive Analytics
  2. Diagnostic Analytics
  3. Predictive Analytics
  4. Prescriptive Analytics

### 12.1 Descriptive Analytics

- ✓ Descriptive analytics helps to understand like **what is happening?**
- ✓ Descriptive analytics provide the information about happened situations.
- ✓ Examples
  - Amazon offered one product then how the customers are buying that specific product.

### 12.2 Diagnostic Analytics

- ✓ Diagnostic analytics helps to understand like **why did it happen?**
- ✓ It is used to do a deeper understanding about the data to find out the cause of events and behaviours.

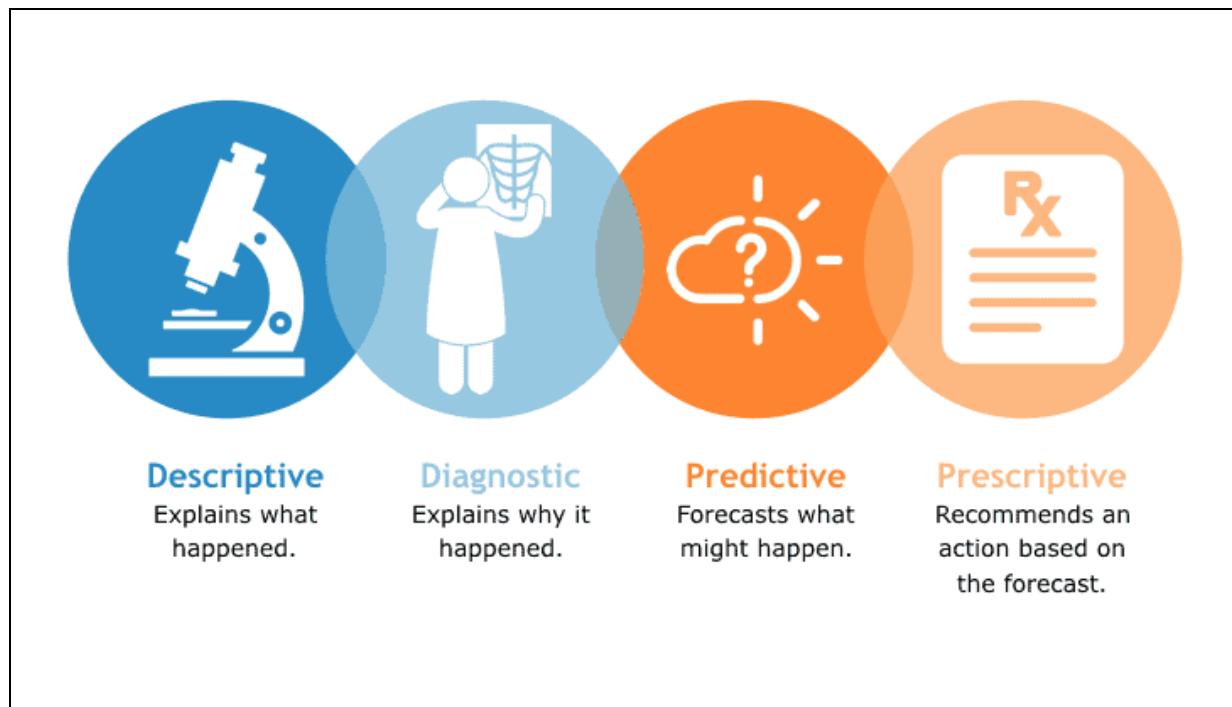
### 12.3 Predictive Analytics

- ✓ Predictive analytics helps to understand like **what will happen in future?**
- ✓ It is used to understand the future expectations, trends and provide the information about what will be happen in future.

## Data Science - Fundamentals

### 12.4 Prescriptive Analytics

- ✓ Prescriptive analytics helps to understand like **what is the next best action?**
- ✓ It is used to apply the best decision actions for improvement.



### 1. PYTHON – INTRODUCTION

#### Table of Contents

<b>1. What is Python?.....</b>	<b>2</b>
<b>2. Python programming applications .....</b>	<b>2</b>
<b>3. History of Python? .....</b>	<b>3</b>
<b>4. Why the name was python?.....</b>	<b>3</b>
<b>5. Python logo.....</b>	<b>4</b>
<b>6. Current python version .....</b>	<b>4</b>
<b>7. Python supports.....</b>	<b>4</b>

## 1. PYTHON INTRODUCTION

### 1. What is Python?

- ✓ Python is a general purpose and high-level programming language.
- ✓ General purpose means, currently all companies are using python programming language to develop, test and deploy the software applications.
- ✓ High level programming means, it's a human readable language and easy to understand.

### 2. Python programming applications

- ✓ By using python programming we can develop,

Type of application	Purpose of the application
1. Standalone application	<ul style="list-style-type: none"> <li>✓ An application which needs to install on every machine to work with that application.</li> </ul>
2. Web applications	<ul style="list-style-type: none"> <li>✓ An application which follows client-server architecture.</li> </ul>
3. Database applications	<ul style="list-style-type: none"> <li>✓ An application which perform curd (create, update, retrieve and delete) operations in database.</li> </ul>
4. Bigdata applications	<ul style="list-style-type: none"> <li>✓ An application which can process the BigData. (like pyspark)</li> </ul>
5. Machine learning	<ul style="list-style-type: none"> <li>✓ An application which enables computers to learn automatically from past data.</li> </ul>

### Make a note

- ✓ By using python we can even implement many of other applications too.

### 3. History of Python?

- ✓ Python was created by **Guido Van Rossum** in the year of 1991.
- ✓ Python is open source software means we can download freely from [www.python.org](http://www.python.org) website and customise the code as well.



**Guido Van Rossum**

### 4. Why the name was python?

- ✓ A TV show Monty Python's Flying Circus was very much popular fun show in 1970's.
- ✓ So, Guido likes this show and given this name to his programming language.

### 5. Python logo



### 6. Current python version

- ✓ While preparing this document the current python version is,
  - Python 3.11.1

### 7. Python supports

- ✓ Python supports Functional and Object oriented programming approach
  - Python = Functional Programming + Object Oriented Programming

## 2. PYTHON - KEYWORDS

### Contents

<b>1. What is a keyword in python? .....</b>	<b>2</b>
<b>2. How many keywords in PYTHON-3.11.1 version?.....</b>	<b>2</b>
<b>3. All keywords in one simple table .....</b>	<b>2</b>

# Data Science – Python Keywords

---

## 2. PYTHON KEYWORDS

### 1. What is a keyword in python?

- ✓ The word which is reserved to do a specific purpose, that word is called as keyword or reserved word.

### 2. How many keywords in PYTHON-3.11.1 version?

- ✓ There are total **35** keywords are available in python **3.11.1** version.
- ✓ All keywords list we can check from below website url
  - [https://docs.python.org/3.11/reference/lexical\\_analysis.html#keywords](https://docs.python.org/3.11/reference/lexical_analysis.html#keywords)

### 3. All keywords in one simple table

- ✓ All keywords we can see in below table.

S.No	1	2	3	4	5	6	7	8	9	10	11
Flow control Variable Function/Method Object Oriented Module Logical Checking Deleting Builtin constants Exception handling Asynchronous											
1	if	global	def	class	import	and	is	del	True	try	async
2	else	nonlocal	with		from	or	in		False	except	await
3	elif		lambda		as	not			None	finally	
4	for		yield							raise	
5	while		return							assert	
6	break										
7	continue										
8	pass										
Total	8	2	5	1	3	3	2	1	3	5	2

### Make a note:

- ✓ All keywords in Python contain only alphabets.
- ✓ These are in lowercase except 3 keywords,
  - True
  - False
  - None

### 3. PYTHON – HELLO WORLD PROGRAM

#### Table of Contents

<b>1. Ways to write python program .....</b>	<b>2</b>
<b>2. Python program execution steps .....</b>	<b>2</b>
<b>3. Understanding first python program .....</b>	<b>2</b>

## Data Science – Python Hello World Program

---

### 3. PYTHON – HELLO WORLD PROGRAM

#### 1. Ways to write python program

- ✓ We can write the python programs in many ways like,
  - By using any notepad, Notepad++, Edit plus, IDLE (Integrated Development Environment), jupyter, PyCharm IDE, Anaconda, Visual Studio, spyder, atom & etc.
- ✓ Initially the best practice is, using notepad to write python program.

#### 2. Python program execution steps

- ✓ Open a new notepad and **write** python program.
- ✓ We can **save** this program with **.py** or **.python** extension.
- ✓ **Run** or execute the program.
- ✓ Finally, we will get **output**.

<b>Program Name</b>	A program to print Hello World
	demo.py

```
print("Hello World")
```

<b>Run</b>	py demo.py
------------	------------

<b>Output</b>	Hello World
---------------	-------------

#### 3. Understanding first python program

- ✓ `print("Hello World")`,
  - `print()` is a predefined function in python.
  - The purpose of `print()` is to display the output.
  - "Hello World" it is a string in python
  - We will learn more about **function** and **string** in upcoming chapters

## Data Science – Python Hello World Practice

---

**Program Name** Run and check the output  
demo1.py

```
print("Hello World")
```

**Output**

**Program Name** Run and check the output  
demo2.py

```
print("Hello World")
print("How are you doing")
```

**Output**

**Program Name** Run and check the output  
demo3.py

```
print("Hello World")
print("How are you doing")
    print("This is Daniel")
```

**Output**

## Data Science – Python Hello World Practice

---

**Program Name** Run and check the output  
demo4.py

```
print()
```

**Output**

**Program Name** Run and check the output  
demo5.py

```
print("Hello World")
```

**Output**

**Program Name** Run and check the output  
demo6.py

```
print("This is 10 $ dollars")
```

**Output**

## Data Science – Python Hello World Practice

---

**Program Name** Run and check the output  
demo7.py

```
print(111)
```

**Output**

**Program Name** Run and check the output  
demo8.py

```
print("Hello World")
    print("How are you doing")
print("This is Daniel")
```

**Output**

**Program Name** Run and check the output  
demo9.py

```
Print("Hello World")
```

**Output**

## Data Science – Python Hello World Practice

---

**Program Name** Run and check the output  
demo10.py

```
PRINT("Hello World")
```

**Output**

**Program Name** Run and check the output  
demo11.py

```
print("Hello World")
```

**Output**

**Program Name** Run and check the output  
demo12.py

```
print("ello orld")
```

**Output**

## Data Science – Python Hello World Practice

---

**Program Name** Run and check the output  
demo13.py

```
print("Hello World")
```

**Output**

## 4. PYTHON – NAMING CONVENTIONS

### Table of Contents

<b>1. Identifier.....</b>	2
<b>2. Why should we follow naming conventions? .....</b>	2
<b>3. Points to follow for identifiers in Python.....</b>	3
# Point 1 .....	3
# Point 2 .....	4
# Point 3 .....	5
# Point 4 .....	6
# Point 5 .....	6
Examples.....	7
<b>Common error.....</b>	7
<b>4. Python identifiers table.....</b>	8
<b>5. Comments in program.....</b>	10
Purpose of comments.....	10
1. Singe line comments.....	10
2. Multi line comments.....	11

### 4. PYTHON - NAMING CONVENTIONS

#### 1. Identifier

- ✓ A name in a python program is called **identifier**.
- ✓ This name can be,

- Package name
- Module name
- Variable name
- Function name
- Class name
- Method name

- ✓ Python creator made some suggestions to the programmers regarding how to write identifiers in a program.

#### 2. Why should we follow naming conventions?

- ✓ While writing the program if we follow the naming conventions then the written code is,
  - Easy to understand.
  - Easy to read.
  - Easy to debug.

## Data Science – Python Naming Conventions

### 3. Points to follow for identifiers in Python

- ✓ We need to follow few points to define an identifiers,

#### # Point 1

- ✓ While writing an identifier we can use,
  - Alphabets, either upper case or lower case
  - Numbers from 0 to 9
  - Underscore symbol (\_)
- ✓ If we are using any other symbol then we will get syntax error.

**Program Name** Creating a valid identifier  
demo1.py

```
student_id = 101
print(student_id)
```

**Output**  
101

**Program Name** Creating an invalid identifier  
demo2.py

```
$student_id = 101
print($student_id)
```

**Error**

SyntaxError: invalid syntax

## Data Science – Python Naming Conventions

### # Point 2

- ✓ We can write an identifier with number but identifier should not start with digit.

**Program Name** Creating a valid identifier  
demo3.py

```
student_id123 = 101
print(student_id123)
```

**Output**  
101

**Program Name** Creating an invalid identifier  
demo4.py

```
123student_id = 101
print(123student_id)
```

**Error**  
**SyntaxError:** invalid decimal literal

## Data Science – Python Naming Conventions

### # Point 3

- ✓ Identifiers are case sensitive.

**Program Name** Creating a valid identifier  
demo5.py

```
value = 10
print(value)
```

**Error**  
10

**Program Name** Identifier is a case sensitive  
demo5.py

```
value = 10
print(VALUE)
```

**Error**  
**NameError:** name 'VALUE' is not defined

## Data Science – Python Naming Conventions

### # Point 4

- ✓ We cannot use keywords as identifiers.

**Program Name** We should not use keywords to create an identifiers  
demo6.py

```
if = 10
print(if)
```

**Error** **SyntaxError:** invalid syntax

### # Point 5

- ✓ Spaces are not allowed in between the identifier.

**Program Name** Spaces not allowed between identifier  
demo7.py

```
student id = 101
print(student id)
```

**Error** **SyntaxError:** invalid syntax

## Data Science – Python Naming Conventions

### Examples

- ✓ 435student # invalid
- ✓ student564 # valid
- ✓ student565info # valid
- ✓ \$student # invalid
- ✓ \_student\_info # valid
- ✓ class # invalid
- ✓ def # invalid

### Common error

- ✓ **SyntaxError**: invalid syntax

## Data Science – Python Naming Conventions

---

### 4. Python identifiers table

- ✓ This table we can understand while studying upcoming topics.

Identifier	Conventions to follow for identifiers
1. class	<ul style="list-style-type: none"> <li>✓ In python, a class name should start with upper case and remaining letters are in lower case.</li> <li>✓ If name having multiple words, then every nested word should start with upper case letter.</li> </ul> <p style="margin-left: 40px;">○ Example: StudentInfo</p> <ul style="list-style-type: none"> <li>✓ Info: This rule is applicable for classes created by users only; the <b>in-built</b> class names used all are in lower-case.</li> </ul>
2. package	
3. module	
4. variable	<ul style="list-style-type: none"> <li>✓ Names should be in lower case.</li> <li>✓ If name having multiple words, then separating words with underscore (_) is good practice.</li> </ul> <p style="margin-left: 40px;">○ Example: student_id</p>
5. function	
6. method	
7. Non-public instance variables	<ul style="list-style-type: none"> <li>✓ Non-public instance variables should begin with underscore (_), we can say private data.</li> </ul> <p style="margin-left: 40px;">○ Example: _balance</p>
8. constants	<ul style="list-style-type: none"> <li>✓ Names should be in upper case.</li> <li>✓ If name having multiple words, then separating words with underscore (_) is good practice.</li> </ul>

## Data Science – Python Naming Conventions

	<ul style="list-style-type: none"><li>○ Example: IN_PROGRESS</li></ul>
9. Non-accessible entities	<ul style="list-style-type: none"><li>✓ Few variables, class constructors (topic in object oriented programming) names having two underscores symbols starting and ending</li></ul> <ul style="list-style-type: none"><li>○ Example: __init__(self)</li></ul>

### 5. Comments in program

- ✓ There are two types of comments

1. Single line comments
2. Multi line comments

#### Purpose of comments

- ✓ Comments are useful to describe about the code in an easy way.
- ✓ Python ignores comments while running the program.

#### 1. Singe line comments

- ✓ By using single line comment, we can comment only a single line.
- ✓ To comment single line, we need to use hash symbol #

<b>Program Name</b>	A program with single line comment demo8.py
	#This is Basic program in python print("Welcome to python programming")
<b>output</b>	Welcome to python programming

## Data Science – Python Naming Conventions

### 2. Multi line comments

- ✓ By using multi line comment we can comment multiple lines.
- ✓ To comment multiple lines, we need to use triple double quotes symbol.

**Program Name** A program with multi line comments  
demo9.py

```
"""Author Daniel  
Project Python project  
Location Bengaluru"""
```

```
print("Welcome to python programming")
```

**output**  
Welcome to python programming

### 5. PYTHON - VARIABLES

#### Contents

<b>1. Variable .....</b>	<b>2</b>
<b>2. Purpose of the variable .....</b>	<b>2</b>
<b>3. Properties of variable.....</b>	<b>2</b>
<b>4. Naming convention to a variable .....</b>	<b>2</b>
<b>5. Creating a variable .....</b>	<b>2</b>
<b>6. Creating multiple variables in single line .....</b>	<b>5</b>
<b>7. Single value for multiple variables.....</b>	<b>6</b>
<b>8. Variable re-initialization.....</b>	<b>7</b>

## Data Science – Python Variables

### 5. PYTHON - VARIABLES

#### 1. Variable

- ✓ Variable is a reserved memory location to store values.

#### 2. Purpose of the variable

- ✓ The purpose of variable is to represent the **data**
- ✓ Data means a collection of **facts**
- ✓ Facts can be,
  - Alphabets.
  - Numbers.
  - Alphanumeric
  - Symbols

#### 3. Properties of variable

- ✓ Every variable has its own properties,
- ✓ Every variable can contain,
  - Name
  - Type
  - Value

#### 4. Naming convention to a variable

- ✓ Name should start with lower case.
- ✓ If name having multiple words then separating every word with underscore symbol is a good practice

#### 5. Creating a variable

- ✓ We need to follow below syntax to create a variable.

##### Syntax

```
name_of_the_variable = value
```

## Data Science – Python Variables

**Program Name** Creating a single variable  
demo1.py

```
age = 16  
print(age)
```

**output**  
16

**Program Name** Creating two variables  
demo2.py

```
name = "Daniel"  
age = 16  
  
print(name)  
print(age)
```

**output**  
Daniel  
16

**Program Name** Creating two variables and using one print  
demo3.py

```
name = " Daniel"  
age = 16  
  
print(name, age)
```

**output**  
Daniel 16

## Data Science – Python Variables

**Program Name** Creating a variable with meaningful text message  
demo4.py

```
name = "Daniel"  
  
print("I am", name)
```

**output**  
I am Daniel

### Important note

- ✓ Python having dynamic data type nature.
- ✓ It means, while creating variable we no need to provide the data type explicitly in python.
- ✓ Regarding data type we will learn in Data type chapter.

## 6. Creating multiple variables in single line

- ✓ We can creating multiple variables in single line
- ✓ # Rule
  - There should be same number on the left and right-hand sides.  
Otherwise we will get error.

**Program Name** Creating multiple variables in single line  
demo5.py

a, b, c = 1, 2, 3

```
print(a)  
print(b)  
print(c)
```

**output**

```
1  
2  
3
```

## Data Science – Python Variables

### 7. Single value for multiple variables

- ✓ We can assign a single value to multiple variables simultaneously.

**Program Name** Assign single value to multiple variables  
demo6.py

```
a = b = c = 3
```

```
print(a)  
print(b)  
print(c)
```

**Output**

```
3  
3  
3
```

## Data Science – Python Variables

### 8. Variable re-initialization

- ✓ Based on requirement we can re-initialize existing variable value.
- ✓ In variable re-initialization old values will be replaced with new values.

<b>Program Name</b>	Creating a variable demo7.py
	<pre>sal = 10000 print("My salary is:", sal)</pre>
<b>Output</b>	My salary is: 10000

<b>Program Name</b>	Re-initialising variable demo8.py
	<pre>sal = 10000 sal = 12000  print("My salary is:", sal)</pre>
<b>Output</b>	My salary is: 12000

## 6. PYTHON – DATA TYPES

### Table of Contents

<b>1. What is a Data Type in python?.....</b>	<b>2</b>
<b>2. type(p) function.....</b>	<b>3</b>
<b>3. Different types of data types .....</b>	<b>4</b>
3.1 Built-in data types: .....	4
1. Numeric types.....	5
2. bool data type (boolean data type).....	7
3. None data type .....	8
4. Sequences in Python .....	9
4.1 string data type .....	10
4.2 list data structure .....	11
4.3 tuple data structure .....	11
4.4 set data structure .....	12
4.5 dictionary data structure .....	12
4. 6 range data type .....	13
3.2 User defined data types .....	15

## Data Science – Python Data Types

### 6. PYTHON – DATA TYPES

#### 1. What is a Data Type in python?

- ✓ A data type represents the type of the data stored into a variable or memory.

**Program Name** print different kinds of variables  
demo1.py

```
emp_id = 1  
name = "Daniel"  
salary = 10000.56
```

```
print("My employee id is: ", emp_id)  
print("My name is: ", name)  
print("My salary is: ", salary)
```

#### Output

```
My employee id is: 1  
My name is: Daniel  
My salary is: 10000.56
```

#### Note

- ✓ By using type(p) function we can check the data type of each variable.

### 2. type(p) function

- ✓ type(p) is predefined function in python.
- ✓ By using this we can check the type of the variables.

**Program**

**Name** demo2.py

```
emp_id = 1
name = "Daniel"
salary = 10000.56

print("My employee id is: ", emp_id)
print("My name is: ", name)
print("My salary is: ", salary)
print()
print("emp_id type is: ", type(emp_id))
print("name type is: ", type(name))
print("salary type is: ", type(salary))
```

**Output**

```
My employee id is: 1
My name is: Daniel
My salary is: 10000.56
```

```
emp_id type is: <class 'int'>
name type is: <class 'str'>
salary type is: <class 'float'>
```

### 3. Different types of data types

- ✓ There are two type of data types.
  1. Built-in data types
  2. User defined data types

#### 3.1 Built-in data types:

- ✓ The data types which are already existing in python are called built-in data types.
  1. Numeric types
    - int
    - float
  2. bool (boolean type)
  3. None
  4. Sequence
    - str
    - list
    - tuple
    - set
    - dict
    - range

## Data Science – Python Data Types

---

### 1. Numeric types

- ✓ The numeric types represent numbers, these are divided into three types,
  1. int
  2. float
  3. complex

#### 1.1 int data type

- ✓ The int data type represents a number without decimal values.
- ✓ In python there is no limit for int data type.
- ✓ It can store very large values conveniently.

**Program Name** To print integer value  
demo3.py

```
a = 20
print(a)
print(type(a))
```

**output**

```
20
<class 'int'>
```

**Program Name** int data type can store bigger values too  
demo4.py

```
b = 99999999999999999999
print(b)
print(type(b))
```

**output**

```
99999999999999999999
<class 'int'>
```

### 1. 2. float data type

- ✓ The float data type represents a number with decimal values.

**Program Name** To print float value and data type  
demo5.py

```
salary = 10000.56
print(salary)
print(type(salary))
```

**Output**

```
10000.56
<class 'float'>
```

## 2. bool data type (boolean data type)

- ✓ bool data type represents boolean values in python.
- ✓ bool data type having only two values those are,
  - **True**
  - **False**

### Make a note

- ✓ Python internally represents,
  - **True** as 1
  - **False** as 0

**Program Name**      boolean values  
demo6.py

a = **True**  
b = **False**

print(a)  
print(b)

**output**  
True  
False

### 3. None data type

- ✓ **None** data type represents an object that does not contain any value.
- ✓ If any object having no value, then we can assign that object with **None** data type.

**Program Name** None data type  
demo7.py

```
a = None
print(a)
print(type(a))
```

**output**
None
<class 'NoneType'>

#### Note

- ✓ A function and method can return **None** data type.
- ✓ This point we will understand more in functions and oops chapters.

### 4. Sequences in Python

- ✓ Sequence means an object.
- ✓ Sequence object can store a group of values,
  1. string
  2. list
  3. tuple
  4. set
  5. dict
  6. range

#### Make a note

- ✓ Regarding sequences like **string**, **list**, **tuple**, **set** and **dict** we will discuss in upcoming chapters
  - Python String chapter
  - Python List Data Structure chapter
  - Python Tuple Data Structure chapter
  - Python Set Data Structure chapter
  - Python Dictionary Data Structure chapter

### 4.1 string data type

- ✓ A group of characters enclosed within single quotes or double quotes or triple quotes is called as string.

**Program Name** Creating a string  
demo8.py

```
name1 = 'Daniel'  
name2 = "Daniel"  
name3 = '''Daniel'''  
name4 = """Daniel"""  
  
print(name1)  
print(name2)  
print(name3)  
print(name4)  
print(type(name1))  
print(type(name2))  
print(type(name3))  
print(type(name4))
```

#### Output

```
Daniel  
Daniel  
Daniel  
Daniel  
<class 'str'>  
<class 'str'>  
<class 'str'>  
<class 'str'>
```

## 4.2 list data structure

- ✓ We can create list data structure by using square brackets []
- ✓ list can store a group of values.

**Program Name** Creating a list data structure  
demo9.py

```
values = [10, 20, 30, 40]
print(values)
print(type(values))
```

**Output**

```
[10, 20, 30, 40]
<class 'list'>
```

## 4.3 tuple data structure

- ✓ We can create tuple data structure by using parenthesis symbol ()
- ✓ tuple can store a group of values.

**Program Name** Creating a tuple data structure  
demo10.py

```
values = (10, 20, 30, 40)
print(values)
print(type(values))
```

**Output**

```
(10, 20, 30, 40)
<class 'tuple'>
```

## Data Science – Python Data Types

---

### 4.4 set data structure

- ✓ We can create set data structure by using curly braces {}
- ✓ set can store a group of values.

**Program Name** Creating a set data structure  
demo11.py

```
values = {10, 20, 30, 40}
print(values)
print(type(values))
```

**Output**

```
{40, 10, 20, 30}
<class 'set'>
```

### 4.5 dictionary data structure

- ✓ We can create dictionary data structure by using curly braces {}
- ✓ Dictionary can store a group of values in the form of key value pair.

**Program Name** Creating a dictionary data structure  
demo12.py

```
details = {1: "Jeswanth", 2: "Kumari", 3: "Prasad", 4: "Daniel"}
print(details)
print(type(details))
```

**Output**

```
{1: 'Jeswanth', 2: 'Kumari', 3: 'Prasad', 4: 'Daniel'}
<class 'dict'>
```

### 4. 6 range data type

- ✓ range is a data type in python.
- ✓ Generally, range means a group of values from starting to ending.

#### Creating range of values

- ✓ We can create range of values by using range(p) predefined function

#### 1. range(p) function

- ✓ As discussed, we can create range of values by using range(p) function.
- ✓ Here p should be integer, otherwise we will get error.

**Program Name** creating a range of values 0 to 4  
demo13.py

```
a = range(5)
print(a)
print(type(a))
```

**Output**
range(0, 5)
<class 'range'>

#### Note:

- ✓ If we provide range(5), then range object holds the values from 0 to 4

### 2. range(start, end) function

- ✓ As discussed we can create range of values by using range(start, end) function.
- ✓ Here start means starting value and end means till to end-1 value

**Program Name** creating a range of values 1 to 9  
demo14.py

```
a = range(1, 10)  
print(a)
```

**Output**  
range(1, 10)

#### Note:

- ✓ If we provide range(1, 10), then range object holds the values from 1 to 9

### Accessing range values by using for loop

- ✓ We can access range values by using for loop.

**Program Name** Access elements from range data type  
demo15.py

```
r = range(10, 15)
```

```
for value in r:  
    print(value)
```

**output**

```
10  
11  
12  
13  
14
```

### 3.2 User defined data types

- ✓ Data types which are created by programmer.
- ✓ The datatype which are created by the programmers are called ‘user-defined’ data types, example is class, module, array etc.
- ✓ We will discuss about in OOPS chapter.

## Data Science – Python Variables Practice

**Question 1** Is Python case sensitive when dealing with identifiers?

**Answer** Yes

**Question 2** What is the maximum possible length of an identifier?

**Answer** There is no length limit for identifier

**Program** Printing name, id, salary

**Name** demo1.py

```
name = "Daniel"  
emp_id = 11  
emp_salary = 1000.123  
  
print(name)  
print(emp_id)  
print(emp_salary)
```

**Output**

## Data Science – Python Variables Practice

**Program Name** Printing name, id, salary with meaningful info  
demo2.py

```
name = "Daniel"  
emp_id = 11  
emp_salary = 1000.123  
  
print("Emp name is: ",name)  
print("Emp id is: ",emp_id)  
print("Emp salary is: ",emp_salary)
```

**Output**

**Program Name** below variable names are valid or invalid  
demo3.py

```
a=1  
_a = 2      # single underscore  
__a = 3     # two underscore  
__a__ = 4   # two underscores starting and ending of the  
variable  
  
print(a)  
print(_a)  
print(__a)  
print(__a__)
```

**Output**

## Data Science – Python Variables Practice

**Program Name** below variable names are valid or invalid  
demo4.py

```
name = "Daniel"  
name_1='Abhishek'  
_ = 'Ramesh'  
  
print(name)  
print(name_1)  
print(_)
```

**Output**

**Program Name** below variable names are valid or invalid  
demo5.py

```
name = "Daniel"  
name_1 = "Naresh"  
1_name = "Srihari"  
_ = "Ramesh"  
  
print(name)  
print(name_1)  
print(1_name)  
print(_)
```

**Output**

## Data Science – Python Variables Practice

---

**Program Name** below variable names are valid or invalid  
demo6.py

```
name = "Daniel"  
name_1 = "Abhishek"  
$_name = "Srihari"  
_ = "Ramesh"
```

```
print(name)  
print(name_1)  
print(1_name)  
print(_)
```

**Output**

**Program Name** below variable name is valid or invalid  
demo7.py

```
abc = 1,000,000  
print(abc)
```

**Output**

## Data Science – Python Variables Practice

**Program Name** below variable name is valid or invalid  
demo8.py

```
abc = 1,000,000
print(abc)
print(type(abc))
```

**Output**

**Program Name** below variable names are valid or invalid  
demo9.py

```
a b c = 1000 2000 3000
print(a b c)
```

**Output**

**Program Name** below variable names are valid or invalid  
demo10.py

```
a b c = 1000 2000 3000
print(a b c)
print(type(a b c))
```

**Output**

## Data Science – Python Variables Practice

---

**Program Name** below variable names are valid or invalid  
demo11.py

```
a, b, c = 1000, 2000, 3000  
print(a, b, c)
```

**Output**

**Program Name** below variable names are valid or invalid  
demo12.py

```
a, b, c = 1000, 2000, 3000  
print(a, b, c)  
print(type(a, b, c))
```

**Output**

## Data Science – Python Variables Practice

---

**Program Name** below variable names are valid or invalid  
demo13.py

```
a, b, c = 1000, 2000, 3000
print(a, b, c)
```

```
print(type(a))
print(type(b))
print(type(c))
```

**Output**

**Program Name** below variable name is valid or invalid  
demo14.py

```
a_b_c = 1,2,3
print(a_b_c)
```

**Output**

## Data Science – Python Variables Practice

**Program** below variable name is valid or invalid

**Name** demo15.py

```
a_b_c = 1,2,3  
print(a_b_c)  
  
print(type(a_b_c))
```

**Output**

**Program** below program valid or invalid

**Name** demo16.py

```
10  
print("hello")
```

**Output**

**Program** below program valid or invalid

**Name** demo17.py

```
a  
print("hello")
```

**Output**

## Data Science – Python Variables Practice

### Invalid cases for variables

#### 1. While creating a variable,

- o Variable name should be written in left hand side.
- o Value should be written in right hand side
- o Otherwise we will get syntax error.

**Program Name** Creating variable in wrong direction  
demo18.py

```
10 = emp_id  
print(emp_id)
```

**Error**  
`SyntaxError: can't assign to literal`

#### 2. variables names,

- o should not give as keywords names, otherwise we will get error

**Program Name** keywords we should not use as variable names  
demo19.py

```
if = 10  
print(if)
```

**Error**  
`SyntaxError: invalid syntax`

## Data Science – Python Variables Practice

---

3. Variables name without value is invalid.

**Program Name** Variable without value is invalid.

demo20.py

```
a  
print(a)
```

**Error**

NameError: name 'a' is not defined

## 7. PYTHON – OPERATORS

### Table of Contents

<b>1. Operator .....</b>	<b>2</b>
<b>2. Type of operators .....</b>	<b>2</b>
1. Arithmetic Operators: (+, -, *, /, %, **, //).....	3
2. Assignment operator: (=, +=, -=, *=, /=, %=, **=, //=) .....	5
3. Unary minus operator: (-) .....	6
4. Relational operators (>, >=, <, <=, ==, !=).....	7
5. Logical operators (and, or, not) .....	9
6. Membership operators(in, not in) .....	11

## Data Science – Python Operators

---

### 7. PYTHON – OPERATORS

#### 1. Operator

- ✓ An operator is a symbol that performs an operation.
- ✓ An operator acts on some variables, those variables are called as operands.
- ✓ If an operator acts on a single operand, then it is called as **unary** operator
- ✓ If an operator acts on 2 operands, then it is called as **binary** operator.
- ✓ If an operator acts on 3 operands, then it is called as **ternary** operator.

**Program Name** operator and operands  
demo1.py

```
a = 10
b = 20
c = a + b
print(c)
```

**output**  
30

- ✓ Here a, b and c are called as operands.
- ✓ + symbol is called as operator.

#### 2. Type of operators

✓ Arithmetic Operators:	+, -, *, /, %, **, //
✓ Assignment Operator	=
✓ Unary minus Operator	-
✓ Relational Operator	>, <, >=, <=, ==, !=
✓ Logical Operators	and, or, not
✓ Membership operators	in, not in

## Data Science – Python Operators

---

Make a note:

- ✓ Python does not have **increment** and **decrement** operators.

### 1. Arithmetic Operators: (+, -, \*, /, %, \*\*, //)

- ✓ These operators will do their usual job, so please don't expect any surprises.

Assume that,

```
a = 13
b = 5
```

Operator	Meaning	Example	Result
+	Addition	$a + b$	18
-	Subtraction	$a - b$	8
*	Multiplication	$a * b$	65
/	Division	$a / b$	2.6
%	Modulus (Remainder of division)	$a \% b$	3
**	Exponent operator (exponential power value)	$a ** b$	371293

## Data Science – Python Operators

//	Integer division (gives only integer quotient)	a // b	2

### Make a note

- ✓ Division operator / always performs floating point arithmetic, so it returns float values.
- ✓ Floor division (//) can perform both floating point and integral as well,
  - If values are int type, then result is int type.
  - If at least one value is float type, then result is float type.

**Program Name** Arithmetic Operators  
demo2.py

```
a = 13
b = 5
print(a+b)
print(a-b)
print(a*b)
print(a/b)
print(a%b)
print(a**b)
print(a//b)
```

### Output

```
18
8
65
2.6
3
371293
2
```

## Data Science – Python Operators

---

### 2. Assignment operator: (=, +=, -=, \*=, /=, %=, \*\*=, //=)

- ✓ By using these operators, we can assign values to variables.

Assume that,

`a = 13`

Operator	Example	Equals to	Result
=	<code>a = 13</code>	<code>a = 13</code>	13
+=	<code>a += 6</code>	<code>a = a + 6</code>	19
-=	<code>a -= 6</code>	<code>a = a - 6</code>	7

**Program Name** Assignment operator  
demo3.py

```
a = 13
print(a)
```

```
a += 5
print(a)
```

**Output**

```
13
18
```

## Data Science – Python Operators

### 3. Unary minus operator: (-)

- ✓ Symbol of unary minus operator is -

**Program Name**      Unary minus operator  
demo4.py

```
a = 10
print(a)
print(-a)
```

**Output**

```
10
-10
```

## Data Science – Python Operators

---

### 4. Relational operators ( $>$ , $\geq$ , $<$ , $\leq$ , $\text{==}$ , $\text{!=}$ )

- ✓ By using these operators, we can create **simple conditions**.
- ✓ These operators are used to **compare two values**.
- ✓ These operators' returns result as **Boolean type** either True or False.

Assume that,

$a = 13$

$b = 5$

Operator	Example	Result
$>$	$a > b$	True
$\geq$	$a \geq b$	True
$<$	$a < b$	False
$\leq$	$a \leq b$	False
$\text{==}$	$a \text{ == } b$	False
$\text{!=}$	$a \text{ != } b$	True

## Data Science – Python Operators

**Program Name** Relational operators  
demo4.py

```
a = 13
b = 5

print(a > b)
print(a >= b)
print(a < b)
print(a <= b)
print(a == b)
print(a != b)
```

**Output**

```
True
True
False
False
False
True
```

## Data Science – Python Operators

---

### 5. Logical operators (and, or, not)

- ✓ In python there are three logical operators those are,
  - **and**
  - **or**
  - **not**
- ✓ Logical operators are useful to create **compound conditions**.
- ✓ Compound condition is a **combination** of more than one simple condition.
- ✓ Each simple condition brings the Boolean result finally the total compound condition evaluates either **True** or **False**.

#### For Boolean types behaviour

- ✓ **and**
  - If both arguments are True, then only result is True
- ✓ **or**
  - If at least one argument is True, then result is True
- ✓ **not**
  - complement

**Program Name**      Logical operators.  
demo5.py

```
a = True
b = False

print(a and a)
print(a or b)
print(not a)
```

**Output**

```
True
True
False
```

## Data Science – Python Operators

---

**Program Name** Logical operators  
demo7.py

```
a = 1
b = 2
c = 3
print((a>b) and (b>c))
print((a<b) and (b<c))
```

**Output**

```
False
True
```

## 6. Membership operators (in, not in)

- ✓ There are two membership operators,
  - **in**
  - **not in**
- ✓ Membership operators are useful to check whether the given value is available in sequence or not. (It may be string, list, set, tuple and dict)

### The **in** operator

- ✓ **in** operator returns **True**, if element is found in the collection or sequences.
- ✓ **in** operator returns **False**, if element is not found in the collection or sequences.

### The **not in** operator

- ✓ **not in** operator returns **True**, if an element is not found in the sequence.
- ✓ **not in** operator returns **False**, if an element is found in the sequence.

**Program Name**      in, not in operators  
demo8.py

values = [10, 20, 30]

```
print(20 in values)
print(22 in values)
```

**output**

```
True
False
```

## Data Science – Python Operators

**Program**

in, not in operators

**Name**

demo9.py

```
text = "Welcome to python programming"
```

```
print("Welcome" in text)
print("welcome" in text)
print("nireekshan" in text)
print("Daniel" not in text)
```

**output**

True

False

False

True

### 8. PYTHON – INPUT AND OUTPUT

#### Table of Contents

<b>1. Input and output .....</b>	<b>2</b>
<b>2. Hard coding.....</b>	<b>2</b>
<b>3. input(p) function .....</b>	<b>3</b>
<b>4. Convert from string to int .....</b>	<b>5</b>
<b>5. Convert from string type to other type .....</b>	<b>7</b>
<b>6. Convert float data type value into int data type.....</b>	<b>9</b>
<b>7. List out few type conversion functions in python.....</b>	<b>12</b>

## 8. Input and Output

### 1. Input and output

- ✓ Input represents data given to the program.
- ✓ Output represents the result of the program.

### 2. Hard coding

- ✓ Till now we have executed examples by hard coding values to variables
- ✓ In this chapter we will learn how to take values at run time.

<p><b>Program</b></p> <p>Name</p>	Hard coding the values demo1.py
	age = 16 print(age)
<p><b>output</b></p>	16

- ✓ Based on requirement we can take values at runtime or dynamically as well.

Please enter the age: 16
Entered value is: 16

## Data Science – Python Input and Output

---

### 3. **input(p)** function

- ✓ **input(p)** is a predefined function.
- ✓ This function accepts input from the keyboard.
- ✓ This function takes a value from keyboard and returns it as a string type.
- ✓ Based on requirement we can convert from string to other types.

**Program Name** Printing name by taking value at run time  
demo2.py

```
name = input("Enter the name:")
print("You entered name as:", name)
```

**Run** py demo2.py

**Output**

```
Enter the name: Daniel
You entered name as: Daniel
```

**Program Name** Printing name and age by taking value at run time  
demo3.py

```
name = input("Enter the name: ")
age = input("Enter the age: ")

print("You entered name as: ", name)
print("You entered age as: ", age)
```

**Run** py demo3.py

**Output**

```
Enter the name: Prasad
Enter the age: 16
```

## Data Science – Python Input and Output

```
You entered name as: Prasad  
You entered age as: 16
```

**Program Name** Checking return type value for `input()` function  
`demo4.py`

```
value = input("Enter the value ")  
print("Entered value as: ", value)  
print("type is: ", type(value))
```

**Run** `py demo4.py`

**Output**  
Enter the value: **Daniel**  
Entered value as: **Daniel**  
type is: <class 'str'>

**Run** `py demo4.py`

**Output**  
Enter the value:**123**  
Entered value as: **123**  
type is: <class 'str'>

**Run** `py demo4.py`

**Output**  
Enter the value:**123.456**  
Entered value as: **123.456**  
type is: <class 'str'>

### 4. Convert from string to int

- ✓ We can convert the string value into int value.
- ✓ int(p) is a predefined function
- ✓ This function converts to int data type.

**Program Name** Converting from string to int data type  
demo5.py

```
a = "123"  
print(a)  
print(type(a))
```

```
b = int(a)  
print(b)  
print(type(b))
```

#### Output

```
123  
<class 'str'>  
123  
<class 'int'>
```

## Data Science – Python Input and Output

**Program Name** Converting from string to int data type  
demo7.py

```
a = input("Enter a value:")
print("Your value is:", a)
print("data type is:", type(a))

b = int(a)
print("After converting the value is:", b)
print("data type is:", type(b))
```

**Run** py demo5.py

**Output**

```
Enter a value:123
Your value is: 123
data type is: <class 'str'>
After converting the value is: 123
data type is: <class 'int'>
```

### 5. Convert from string type to other type

- ✓ We can convert the string value into float value.
- ✓ float(p) is a predefined function
- ✓ This function converts to float data type.

**Program Name** Converting from string to float data type  
demo8.py

```
a = "123.99"  
print(a)  
print(type(a))
```

```
b = float(a)  
print(b)  
print(type(b))
```

#### Output

```
123.99  
<class 'str'>  
123.99  
<class 'float'>
```

## Data Science – Python Input and Output

**Program Name** Converting from string to float data type  
demo9.py

```
a = input("Enter a value:")
print("Your value is:", a)
print("data type is:", type(a))

b = float(a)
print("After converting the value is:", b)
print("data type is:", type(b))
```

**Run** py demo9.py

**Output**

```
Enter a value:123.99
Your value is: 123.99
data type is: <class 'str'>
After converting the value is: 123.99
data type is: <class 'float'>
```

### 6. Convert float data type value into int data type

- ✓ `int(p)` is predefined function in python.
- ✓ This function converts value into int data type

**Program**

Converting from float to int data type

**Name**

`demo10.py`

```
a = 10000.45
```

```
print(a)
```

```
print(type(a))
```

```
b = int(a)
```

```
print(b)
```

```
print(type(b))
```

**Output**

```
10000.45
```

```
<class 'float'>
```

```
10000
```

```
<class 'int'>
```

## Data Science – Python Input and Output

**Program Name** + operator concatenates/joins two string values  
demo11.py

```
p1 = input("Enter First product price:")  
p2 = input("Enter Second product price:")
```

```
print("Total price is:", p1+p2)
```

**Run** py demo11.py

**Output**

```
Enter First product price:111  
Enter Second product price:222  
Total price is: 111222
```

## Data Science – Python Input and Output

**Program Name** Converting string to int and perform + operator  
demo12.py

```
p1 = input("Enter First product price:")  
p2 = input("Enter Second product price:")  
  
a = int(p1)  
b = int(p2)  
  
print("Total price is:", a+b)
```

**Run** py demo12.py

**Output**

```
Enter First product price:111  
Enter Second product price:222  
Total price is: 333
```

### 7. List out few type conversion functions in python

1. int(p) – converts other data type into integer type
2. float(p) – converts other data type into float type
3. str(p) – converts other data type into a string.
4. bool(p) – converts other data type into boolean type
5. list(p) – converts sequence to list
6. tuple(p) – converts sequence to a tuple.
7. set(p) – converts sequence to set.
8. dict(p) – converts a tuple of order (key, value) into a dictionary.

## 9. PYTHON – FLOW CONTROL

### Table of Contents

<b>1. Flow control.....</b>	2
<b>2. Types of the execution .....</b>	2
1. Sequential .....	2
2. Conditional.....	2
3. Looping.....	3
<b>3. Sequential statements .....</b>	3
<b>4. Indentation.....</b>	4
4.1 IndentationError .....	4
<b>5. Conditional or Decision-making statements .....</b>	5
5.1 if statement.....	5
5.2 if else statement .....	7
5.3 if elif else statement .....	9
<b>6. Looping .....</b>	11
6.1 for loop.....	11
6.2 while loop.....	12
<b>7. break statement.....</b>	14
<b>8. continue statement.....</b>	16
<b>9. pass statement .....</b>	17

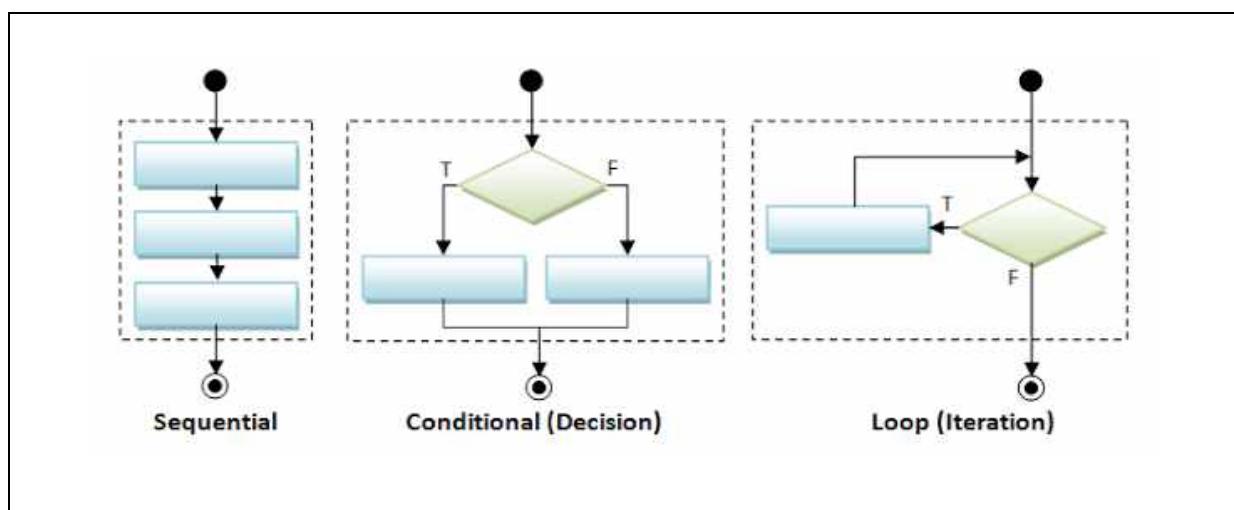
## 9. PYTHON – FLOW CONTROL

### 1. Flow control

- ✓ The order of statements execution is called as flow of control.

### 2. Types of the execution

- ✓ Based on requirement the programs statements can execute in different ways like sequentially, conditionally and repeatedly etc.
- ✓ In any programming language, statements will be executed mainly in three ways,
  - Sequential execution.
  - Conditional execution.
  - Looping execution.



### 1. Sequential

- ✓ Statements execute from top to bottom, means one by one sequentially.
- ✓ By using sequential statement, we can develop only simple programs.

### 2. Conditional

- ✓ Based on conditions, statements used to execute.
- ✓ Conditional statements are useful to develop better and complex programs.

## Data Science – Python Flow Control

### 3. Looping

- ✓ Based on conditions, statements used to execute randomly and repeatedly.
- ✓ Looping execution is useful to develop better and complex programs.

### 3. Sequential statements

**Program Name** sequential statement: executes from top to bottom  
demo1.py

```
print("one")
print("two")
print("three")
```

#### output

```
one
two
three
```

### 2. Conditional or Decision-making statements

- if - valid combination
- if else - valid combination
- if elif else - valid combination
- if elif elif else - valid combination

### 3. Looping

- ✓ for loop
- ✓ while loop

#### 4. Other keywords

- ✓ break
- ✓ continue
- ✓ pass

#### 4. Indentation

- ✓ Python uses indentation to indicate a block of code.
- ✓ Indentation refers to adding white space before a statement to a particular block of code.
- ✓ Python uses 4 spaces as indentation by default.
  - However, the number of spaces is up to you, but a minimum of 1 space has to be used.

##### 4.1 IndentationError

- ✓ If we didn't follow the proper indentation then we will get error as **IndentationError**: expected an indented block

## 5. Conditional or Decision-making statements

### 5.1 if statement

#### syntax

```
if condition:  
    if block statements  
  
    out of if block statements
```

- ✓ **if** is a keyword in python
- ✓ **if** statement contains an expression/condition/value.
- ✓ As per the syntax colon (:) is mandatory otherwise it throws syntax **error**.
- ✓ After **if** statement we need to follow indentation otherwise it throws **IndentationError**.
- ✓ Condition gives the result as a bool type, means either **True** or **False**.



- ✓ If the condition result is **True**, then **if** block statements will be executed
- ✓ If the condition result is **False**, then **if** block statements won't execute.

## Data Science – Python Flow Control

**Program Name** Executing if block statements by using **if** statement  
demo2.py

```
x = 1
y = 1

print("x==y value is: ", (x==y))
if x == y:
    print("if block statements executed")
```

**output**
x==y value is: **True**
if block statements executed

**Program Name** Executing out of if block statements  
demo3.py

```
x = 1
y = 2

print("x==y value is: ", (x==y))
if x == y:
    print("if block statements executed")
print("out of if block statements")
```

**output**
x==y value is: **False**
out of if block statements

## 5.2 if else statement

### syntax

```
if condition:  
    if block statements1  
  
else:  
    else block statements2
```

- ✓ **if** and **else** are keywords in python
- ✓ **if** statement contains an expression/condition.
- ✓ As per the syntax colon (:) is mandatory otherwise it throws syntax **error**.
- ✓ After **if** and **else** statements we need to follow indentation otherwise it throws **IndentationError**.
- ✓ Condition gives the result as bool type, means either **True** or **False**



- ✓ If the condition result is **True**, then **if** block statements will be executed
- ✓ If the condition result is **False**, then **else** block statements will be executed.

## Data Science – Python Flow Control

**Program Name** Executing if block statements by using **if else** statement  
demo4.py

```
x = 1
y = 1

print("x==y value is: ", (x==y))

if x == y:
    print("if block statements executed")
else:
    print("else block statements executed")
```

**output**
x==y value is: **True**
if block statements executed

**Program Name** printing else block statements  
demo5.py

```
x = 1
y = 2

print("x==y value is: ", (x == y))

if x == y:
    print("if block statements executed")
else:
    print("else block statements executed")
```

**output**
x==y value is: **False**
else block statements executed

### 5.3 if elif else statement

#### syntax

```
if condition1:  
    if block statements  
  
elif condition2:  
    elif block1statements  
  
elif condition3:  
    elif block2statements  
  
else:  
    else block statements
```

- ✓ **if, elif and else** are keywords in python
- ✓ **if** statement contains an expression/condition.
- ✓ As per the syntax colon (:) is mandatory otherwise it throws error.
- ✓ After **if, elif** and **else** statements we need to follow indentation otherwise it throws IndentationError.
- ✓ Condition gives the result as bool type, means either **True** or **False**



- ✓ If the condition result is **True**, then any matched **if** or **elif** block statements will execute.
- ✓ If all **if** and **elif** conditions results are **False**, then **else** block statements will execute.

**Make a note**

- ✓ Here, else part is an optional

**Program Name** printing corresponding value by using if, elif, else statements  
demo6.py

```
print("Please enter the values from 0 to 4")
x = int(input("Enter a number: "))

if x == 0:
    print("You entered:", x)

elif x == 1:
    print("You entered:", x)

elif x == 2:
    print("You entered:", x)

elif x == 3:
    print("You entered:", x)

elif x == 4:
    print("You entered:", x)

else:
    print("Beyond the range than specified")
```

**output**

```
Enter a number: 1
You entered: 1

Enter a number: 100
Beyond the range than specified
```

## 6. Looping

- ✓ If we want to execute a group of statements in multiple times, then we should go for looping kind of execution.
  - for loop
  - while loop

### 6.1 for loop

- ✓ **for** is a keyword in python
- ✓ Basically, **for** loop is used to get or iterate elements one by one from sequence like string, list, tuple, etc...
- ✓ While iterating elements from sequence we can perform operations on every element.

#### Syntax

```
for variable in sequence:  
    statements
```

**Program Name** Using for loop printing value from list  
demo7.py

```
values = [10, 20, 30, "Daniel"]  
for value in values:  
    print(value)
```

#### output

```
10  
20  
30  
Daniel
```

## 6.2 while loop

- ✓ **while** is a keyword in python
- ✓ If we want to execute a group of statements repeatedly until the condition reaches to **False**, then we should go for **while** loop.

### Syntax

```

Initialization
while condition:
    while block statements
    increment/decrement
  
```

- ✓ **while** loop contains an expression/condition.
- ✓ As per the syntax colon (:) is mandatory otherwise it throws syntax error.
- ✓ After **while** loop we need to follow indentation otherwise it throws IndentationError.
- ✓ Condition gives the result as bool type, means either **True** or **False**



- ✓ If the condition result is **True**, then while loop executes till the condition reaches to **False**.
- ✓ If the condition result is **False**, then **while** loop execution terminates.

### Conclusion

- ✓ Till condition is **True** the **while** loop statements will be executed.
- ✓ If the condition reaches to **False**, then **while** loop terminate the execution.

## Data Science – Python Flow Control

---

**Program Name** Printing numbers from 1 to 5 by using while loop  
demo8.py

```
x = 1
while x <= 5:
    print(x)
    x = x+1

print("End")
```

**output**

```
1
2
3
4
5
End
```

## 7. break statement

- ✓ **break** is a keyword in python
- ✓ The **break** statement can be used inside the loops.
- ✓ By using break we can break the execution based on some condition.
- ✓ Generally, **break** statement is used to terminate **for** and **while** loops.

**Program Name** while loop without break  
demo9.py

```
x = 1
while x<=10:
    print("x=", x)
    x = x + 1

print("out of loop")
```

**output**

```
x= 1
x= 2
x= 3
x= 4
x= 5
x= 6
x= 7
x= 8
x= 9
x= 10
out of loop
```

## Data Science – Python Flow Control

**Program Name** printing just 1 to 5 by using while loop and break  
demo10.py

```
x = 1
while x <= 10:
    print("x=", x)
    x = x+1
    if x == 5:
        break

print("out of the loop")
```

**output**

```
x= 1
x= 2
x= 3
x= 4
out of the loop
```

**Program Name** break without loop  
demo11.py

```
x = 1
if x <= 10:
    print(x)
    break
```

**output**

SyntaxError: 'break' outside loop

## 8. continue statement

- ✓ **continue** is a keyword in python
- ✓ We can use **continue** statement to skip current iteration and continue next iteration.

**Program Name** continue statement  
demo12.py

```
cart = [10, 20, 500, 700, 50, 60]
for item in cart:
    if item == 500:
        continue
    print(item)
```

**output**

```
10
20
700
50
60
```

**Program Name** continue without loop  
demo13.py

```
x = 1
if x <= 10:
    continue

print(x)
```

**output**

SyntaxError: 'continue' not properly in loop

## 9. pass statement

- ✓ **pass** is keyword in python.
- ✓ The pass statement is used as a placeholder for future code.
- ✓ It is useful as a placeholder when a statement is required syntactically, but no code needs to be executed.
- ✓ pass is a null operation, when it is executed, nothing happens.
- ✓ We can define an empty function, class, method with pass statement.

**Program Name** Function without pass statement  
demo14.py

```
def upcoming_sales():

    upcoming_sales()
```

**output**

IndentationError: expected an indented block

**Program Name** Function with pass statement  
demo15.py

```
def upcoming_sales():
    pass
```

upcoming\_sales()

**output**

### Make a note

- ✓ We can even define a method or block of code with pass statement.

**10. Python - String****Table of Contents**

<b>1. Gentle reminder .....</b>	2
<b>2. What is a string? .....</b>	2
2.1. Definition 1.....	2
2.2. Definition 2.....	2
2.3. String is more popular.....	2
<b>3. Creating string .....</b>	3
3.1. When should we go for triple single and triple double quotes? .....	5
<b>4. Mutable .....</b>	8
<b>5. Immutable .....</b>	8
<b>6. Strings are immutable.....</b>	8
<b>7. Mathematical operators on string objects.....</b>	10
7.1. Addition (+) operator with string.....	10
7.2. Multiplication (*) operator with string .....	10
<b>8. Length of the string .....</b>	11
<b>9. Membership operators (in, not in) .....</b>	12
9.1. in operator .....	12
9.2. not in operator.....	13
<b>10. Methods in str class.....</b>	14
11.1. upper() method.....	17
11.2. lower() method .....	18
11.3. strip() method .....	19
11.4. count(p) method .....	20
11.5. replace(p1, p2) method .....	21
11.6. split(p) method .....	23

## 10. Python - String

### 1. Gentle reminder

- ✓ We already learnt first Hello world program in python.
- ✓ In that program we just print a group of characters by using print(p) function.
- ✓ That group of characters are called as a string.

<b>Program Name</b>	printing Welcome to python programming demo1.py
<b>Output</b>	print("Hello world")  Hello world

### 2. What is a string?

#### 2.1. Definition 1

- ✓ A group of characters enclosed within single or double or triple quotes is called as string.

#### 2.2. Definition 2

- ✓ We can say string is a sequential collection of characters.

#### 2.3. String is more popular

- ✓ In any kind of programming language, mostly usage data type is string.

## Data Science - Python String

### 3. Creating string

Syntax 1 With single quotes

```
name1 = 'Daniel'
```

Syntax 2 With double quotes

```
name2 = "Daniel"
```

Syntax 3 With triple single quotes

```
name3 = '''Daniel'''
```

Syntax 4 With triple double quotes

```
name4 = """Daniel"""
```

## Data Science - Python String

**Program Name** Creating string by using all possibilities  
demo2.py

```
name1= 'Daniel'  
name2 = "Prasad"  
name3 = '''Mouli'''  
name4 = """Veeru"""\n  
print(name1, "name is created by using single quotes")  
print(name2, "name is created by using double quotes")  
print(name3, "name is created by using triple single quotes")  
print(name4, "name is created by using triple double quotes")
```

**output**

Daniel name is created by using single quotes  
Prasad name is created by using double quotes  
Mouli name is created by using triple single quotes  
Veeru name is created by using triple double quotes

### Make a note

- ✓ Generally, to create a string mostly used syntax is double quotes syntax.

### 3.1. When should we go for triple single and triple double quotes?

- ✓ If you want to create multiple lines of string, then triple single or triple double quotes are the best to use.

**Program Name** printing Employee information  
demo3.py

```
loc1 = '''TCS company
          White Field
          Bangalore'''
```

```
loc2 = """TCS company
          Bangalore
          ITPL tech park"""
```

```
print(loc1)
print(loc2)
```

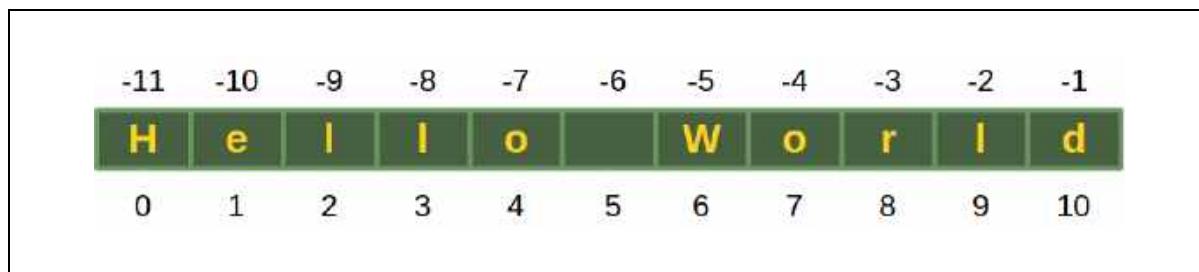
**output**

```
TCS company
White Field
Bangalore
```

```
TCS company
Bangalore
ITPL tech park
```

## Data Science - Python String

### Diagram representation



**Program Name** Accessing string by using index  
demo4.py

```
wish = "Hello World"
```

```
print(wish[0])
print(wish[1])
```

### Output

```
H
e
```

**Program Name** Accessing string by using slicing  
demo5.py

```
wish = "Hello World"
```

```
print(wish[0:7])
```

### Output

```
Hello W
```

## Data Science - Python String

**Program Name** Accessing string by using for loop  
demo6.py

```
wish = "Hello World"
```

```
for char in wish:  
    print(char)
```

**Output**

```
H  
e  
l  
l  
o
```

```
W  
o  
r  
l  
d
```

### 4. Mutable

- ✓ Once if we create an object then the state of existing object can be change/modify/update.
- ✓ This behaviour is called as mutability.

### 5. Immutable

- ✓ Once if we create an object then the state of existing object cannot be change/modify/update.
- ✓ This behaviour is called as immutability.

### 6. Strings are immutable

- ✓ String having immutable nature.
- ✓ Once we create a string object then we cannot change or modify the existing object.

**Program Name** Printing name and first index in string  
demo7.py

```
name = "Daniel"  
print(name)  
print(name[0])
```

**output**

```
Daniel  
D
```

## Data Science - Python String

**Program Name** string having immutable nature  
demo8.py

```
name = "Daniel"
```

```
print(name)
print(name[0])
name[0] = "X"
```

**output**

```
Daniel
```

```
D
```

```
TypeError: 'str' object does not support item assignment
```

## 7. Mathematical operators on string objects

- ✓ We can perform two mathematical operators on string.
- ✓ Those operators are,
  - Addition (+) operator.
  - Multiplication (\*) operator.

### 7.1. Addition (+) operator with string

- ✓ The + operator works like concatenation or joins the strings.

<b>Program Name</b>	+ works as concatenation operator demo9.py
	a = "Python" b = "Programming" print(a+b)
<b>output</b>	PythonProgramming

### 7.2. Multiplication (\*) operator with string

- ✓ This operator works with string to do repetition.

<b>Program Name</b>	* operator works as repetition in strings demo10.py
	course="Python" print(course*3)
<b>output</b>	PythonPythonPython

## Data Science - Python String

### 8. Length of the string

- ✓ We can find number of characters in string by using len() function

**Program Name** length of the string  
demo11.py

```
course = "Python"  
print(len(course))
```

**Output**

```
6
```

## 9. Membership operators (in, not in)

### Definition 1

- ✓ We can check, if a string or character is a member of string or not by using **in** and **not in** operators

### Definition 2

- ✓ We can check, if string is a substring of main string or not by using **in** and **not in** operators.

#### 9.1. in operator

- ✓ **in** operator returns **True**, if the string or character found in the main string.

<b>Program</b>	in operator
<b>Name</b>	demo12.py
	<pre>print('p' in 'python') print('z' in 'python') print('on' in 'python') print('pa' in 'python')</pre>

<b>output</b>	
	True
	False
	True
	False

## Data Science - Python String

### 9.2. not in operator

- ✓ The **not in** operator returns opposite result of **in** operator.
- ✓ **not in** operator returns True, if the string or character not found in the main string.

**Program** not in operator

**Name** demo13.py

```
print('b' not in 'apple')
```

**output**

```
True
```

## Data Science - Python String

### 10. Methods in str class

- ✓ As discussed, str is a predefined class.
- ✓ So, str class can contain methods because methods can be created inside of class only.
- ✓ We can check these methods by using `dir(parameter1)` predefined function.
  
- ✓ So, internally str class contains two types of methods,
  - With underscore symbol methods.
    - We no need to focus
  - Without underscore symbol methods.
    - We need to focus much on these

**Program Name** Printing str class methods by using `dir(str)` function  
**demo14.py**

```
print(dir(str))
```

**output**

[

```
__ge__', '__getattribute__', '__getitem__', '__getnewargs__',
'__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__',
'__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', 'capitalize',
```

### Important methods

'count', 'upper', 'lower', 'replace', 'split', 'strip',

]

### Important point

- ✓ As per object-oriented principle,
  - If we want to access instance methods, then we should access by using object name.
- ✓ So, all str class methods we can access by using str object

### Important methods in str class

- ✓ upper()
- ✓ lower()
- ✓ strip()
- ✓ count(p)
- ✓ replace(p1, p2)
- ✓ split(p)

### 11.1. upper() method

- ✓ upper() is a pre-defined method in str class
- ✓ This method we should access by using string object.
- ✓ This method converts lower case letters into upper case letters

**Program Name** Converting from lowercase to uppercase  
demo15.py

```
name = "daniel"  
print("Before converting: ", name)  
print("After converting: ", name.upper())
```

**Output**  
Before converting: daniel  
After converting: DANIEL

## Data Science - Python String

### 11.2. lower() method

- ✓ lower() is a pre-defined method in str class
- ✓ This method we should access by using string object.
- ✓ This method converts upper case letters into lower case letters

**Program Name** Converting from uppercase to lowercase  
demo16.py

```
name = "DANIEL"
print("Before converting: ", name)
print("After converting: ", name.lower())
```

**Output**  
Before converting: DANIEL  
After converting: daniel

## Data Science - Python String

### 11.3. strip() method

- ✓ strip() is a pre-defined method in str class
- ✓ This method we should access by using string object.
- ✓ This method removes left and right side spaces of string

#### Make a note

- ✓ This method will not remove the spaces which are available middle of string object.

**Program** removing spaces in starting and ending of the string

**Name** demo17.py

```
course = "Python      "
print("with spaces course length is: ",len(course))
x = course.strip()
print("after removing spaces, course length is: ",len(x))
```

#### Output

```
with spaces course length is: 18
after removing spaces, course length is: 6
```

**11.4. count(p) method**

- ✓ count(p) is a pre-defined method in str class
- ✓ This method we should access by using string object.
- ✓ By using count() method we can find the number of occurrences of substring present in the string

**Program Name** counting sub string by using count() method  
demo18.py

```
s="Python programming language, Python is easy"  
print(s.count("Python"))  
print(s.count("Hello"))
```

**output**

```
2  
0
```

## Data Science - Python String

### 11.5. replace(p1, p2) method

- ✓ replace(p1, p2) is a pre-defined method in str class
- ✓ This method we should access by using string object.
- ✓ We can replace old string with new string by using replace(p1, p2) method.

**Program Name** replacing string by using replace method  
demo19.py

```
s1 = "Java programming language"  
s2 = s1.replace("Java", "Python")  
  
print(s1)  
print(s2)
```

**output**  
Java programming language  
Python programming language

### Replace method returns new string object

- ✓ As we know string is immutable, so replace(p1, p2) method never perform changes on the existing string object.
- ✓ replace(p1, p2) method creates new string object, we can check this by using id(p) function

**Program Name** replacing string by using replace(p1, p2) method  
demo20.py

```
s1 = "Java programming language"
s2 = s1.replace("Java", "Python")

print(s1)
print(s2)

print(id(s1))
print(id(s2))
```

**output**

```
Java programming language
Python programming language
49044256
48674600
```

### String objects are immutable, Is replace(p1, p2) method will modify the string objects?

- ✓ Once we create a string object, we cannot change or modify the existing string object.
- ✓ This behaviour is called as immutability.
- ✓ If we are trying to change or modify the existing string object, then with those changes a new string object will be created.
- ✓ So, replace(p1, p2) method will create new string object with the modifications.

**Splitting of Strings:****11.6. split(p) method**

- ✓ `split(p)` is a pre-defined method in `str` class
- ✓ This method we should access by using string object.
- ✓ The default separator is space.
- ✓ We can split the given string according to specified separator by using `split(p)` method.
- ✓ `split(p)` method returns list.

**Program** splitting string by using `split()` method

**Name** demo21.py

```
s = "Python programming language"  
n = s.split()
```

```
print("Before splitting:", s)  
print("After splitting: ", n)
```

**output**

```
Before splitting: Python programming language  
After splitting: ['Python', 'programming', 'language']
```

## Data Science - Python String

**Program Name** splitting string by using split(p) method  
demo22.py

```
s = "This is, Python programming, language "
n = s.split(",")
```

```
print("Before splitting:", s)
print("After splitting: ", n)
```

**output**

```
Before splitting: This is, Python programming, language
After splitting: ['This is', ' Python programming', ' language']
```

## Data Science – Python – Practice - Flow Control

---

**Program Name** Please execute the program and check the output  
demo1.py

a = 33  
b = 200

```
if b > a:  
    print("b is greater than a")
```

**Output**

**Program Name** Please execute the program and check the output  
demo2.py

a = 33  
b = 200

```
if b > a:  
    print("b is greater than a")
```

**Output**

## Data Science – Python – Practice - Flow Control

**Program Name** Please execute the program and check the output  
demo3.py

a = 33  
b = 33

```
if b > a:  
    print("b is greater than a")  
  
elif a == b:  
    print("a and b are equal")
```

**Output**

**Program Name** Please execute the program and check the output  
demo4.py

a = 200  
b = 33

```
if b > a:  
    print("b is greater than a")  
  
elif a == b:  
    print("a and b are equal")  
  
else:  
    print("a is greater than b")
```

**Output**

## Data Science – Python – Practice - Flow Control

**Program Name** Please execute the program and check the output  
demo5.py

```
a = 200  
b = 33
```

```
if b > a:  
    print("b is greater than a")  
  
else:  
    print("b is not greater than a")
```

**Output**

**Program Name** Please execute the program and check the output  
demo6.py

```
a = 200  
b = 33
```

```
if a > b: print("a is greater than b")  
  
print("A") if a > b else print("B")
```

**Output**

## Data Science – Python – Practice - Flow Control

**Program Name** Please execute the program and check the output  
demo7.py

```
a = 200  
b = 33
```

```
if a > b: print("a is greater than b")  
  
print("A") if a > b else print("B")
```

**Output**

**Program Name** Please execute the program and check the output  
demo8.py

```
flag = True
```

```
if flag==True:  
    print("Welcome")  
    print("To")  
    print("Python programming language" )
```

**Output**

## Data Science – Python – Practice - Flow Control

**Program Name** Please execute the program and check the output  
demo9.py

flag = **False**

```
if flag==True:  
    print("Welcome")  
    print("To")  
    print("Python programming language" )
```

**Output**

**Program Name** Please execute the program and check the output  
demo10.py

i = 20;

```
if (i < 15):  
    print ("i is smaller than 15")  
    print ("i'm in if Block")
```

```
else:  
    print ("i is greater than to 15")  
    print ("i'm in else Block")
```

```
print ("i'm not in if and not in else Block")
```

**Output**

## Data Science – Python – Practice - Flow Control

**Program Name** Please execute the program and check the output  
demo11.py

```
i = 10
if (i == 10):

    if (i < 15):
        print ("i is smaller than 15")

    if (i < 12):
        print ("i is smaller than 12 too")

    else:
        print ("i is greater than 15")
```

**Output**

**Program Name** Please execute the program and check the output  
demo12.py

```
x = int(input("Enter x value :"))
y = int(input("Enter y value :"))
z = int(input("Enter z value :"))

if x>y and x>z:
    print("x is greater than remaining two values")

elif y>z:
    print("y is greater than remaining two values")

else:
    print("z is greater than remaining two values ")
```

**Output**

## Data Science – Python – Practice - Flow Control

**Program Name** Calculating the marks and check the output  
demo13.py

```
math = int(input("Enter your maths marks:"))
sci = int(input("Enter your science marks:"))
kan = int(input("Enter your Kannada marks:"))

a = (math + sci +kan)/3

i = int(a)

print("The total average 3 subjects are :",i,"%")

if i>90 and i<=100:
    print("A+ bro")

elif i>80 and i<=90:
    print("A bro")

elif i>60 and i<=80:
    print("B+ bro")

elif i>50 and i<=60:
    print("B bro")

elif i>=30 and i<=35:
    print("C+ bro just pass")

elif i<35:
    print("Sorry boss, you failed")
```

**Output**

## Data Science – Python – Practice - Flow Control

**Program Name**

Please execute the program and check the output  
demo14.py

```
print("Welcome to DANIEL Airport")
weight = float(input("How many pounds does your suitcase
weigh?"))

if weight > 50:
    print("There is a $25 charge for luggage and already
deducted from your credit card, don't get surprise.")

print("Thank you and Enjoy the Journey Boss.")
```

**Output**

**Program Name**

Please execute the program and check the output  
demo15.py

```
balance = -5
```

```
if balance < 0:
    print("Balance is in negative mode please add funds now
or you will be charged a penalty.")
```

**Output**

## Data Science – Python – Practice - Flow Control

**Program Name** Please execute the program and check the output  
demo16.py

```
num = 22
if num % 2 == 0:
    print("Even Number")
else:
    print("Odd Number")
```

**Output**

**Program Name** Please execute the program and check the output  
demo17.py

```
x=10

print(isinstance(x, int))
print(isinstance(x, str))
```

**Output**

**Program Name** Please execute the program and check the output  
demo18.py

```
name="Daniel"

print(isinstance(name, str))
print(isinstance(name, int))
```

**Output**

## Data Science – Python – Practice - Flow Control

**Program Name** Please execute the program and check the output  
demo19.py

x=10

```
if isinstance(x, int):
    print("integer data type")

else:
    print("Its other type")
```

**Output**

**Program Name** Please execute the program and check the output  
demo20.py

x=10

```
if isinstance(x, str):
    print("string data type")

else:
    print("It is an integer type")
```

**Output**

## Data Science – Python – Practice - Flow Control

**Program Name** Please execute the program and check the output  
demo21.py

```
name="daniel "

if isinstance(name, str):
    print("string data type")

else:
    print("It is other type")
```

**Output**

**Program Name** Please execute the program and check the output  
demo22.py

```
name="Daniel"

if isinstance(xyz, int):
    print("integer data type")

else:
    print("It is string data type")
```

**Output**

## Data Science – Python – Practice - Flow Control

**Program Name**

Please execute the program and check the output  
demo23.py

```
answer = input("Is Python teaching by Daniel, Yes or Not >> ")

if answer == "Yes":
    print("Welcome to Python sessions by Daniel, please don
't sleep")

else :
    print("I guess you are not student of Daniel.")

print("Thanks!")
```

**Output**

## Data Science – Python – Practice - Flow Control

**Program Name**

Please execute the program and check the output  
demo24.py

```
print("welcome to AiNextGen company...")  
  
allowed_users = ['daniel', 'chiru', 'balayya']  
username = input("What is your login name? : ")  
  
if username in allowed_users:  
    print("Access granted, Enjoy Guru :)")  
  
else:  
    print("Sorry Boss, access denied: Contact Admin team.  
Namaskaar")
```

**Output**

## Data Science – Python – Practice - Flow Control

**Program Name**

Please execute the program and check the output  
demo25.py

```
number = 4
guess = int(input('Guess number between 1 to 10 : '))

if guess == number:
    print("Congratulations Dude, you guessed it but no prizes
        :) ")

elif guess < number:
    print('No, it is a little higher than guessed number')

else:
    print('No, it is a little lower than guessed number')
```

**Output**

## Data Science – Python – Practice - Flow Control

**Program Name** Please execute the program and check the output demo26.py

```
x = int(input("What is the time?"))

if x < 10:
    print("Good morning")

elif x<12:
    print("Soon time for lunch")

elif x<18:
    print("Good day")

elif x<22:
    print("Good evening")

elif x>=22 and x<=24:
    print("Good night")

else:
    print("Dude, please Buy clock and learn timings, Good day")
```

**Output**

## Data Science – Python – Practice - Flow Control

**Program Name**

Please execute the program and check the output  
demo27.py

```
marks=int(input("Enter marks: "))

if marks >= 90:
    print("A grade")

elif marks >=80:
    print("B grade")

elif marks >=70:
    print("C grade")

elif marks >= 65:
    print("D grade")

elif marks >= 55:
    print("E grade")

elif marks >= 35:
    print("Congrats for great achievement, hope you are from
Back bench")

else:
    print("Congratulation Dude, you have successfully failed")
```

**Output**

## Data Science – Python – Practice - Flow Control

**Program Name** Please execute the program and check the output  
demo28.py

```
marks=int(input("Enter marks: "))

if marks >= 90:
    if marks > 96:
        print("A++")

    elif marks > 93 and marks <= 96:
        print("A+")

    elif marks >= 90:
        print("A grade")

elif marks >=80:
    print("B grade")

elif marks >=70:
    print("C grade")

elif marks >= 65:
    print("D grade")

elif marks >= 55:
    print("E grade")

elif marks >= 35:
    print("Congrats for great achievement, hope you are from
Back bench")

else:
    print("Congratulation Dude, you have successfully failed")
```

**Output**

## Data Science – Python – Practice - Flow Control

---

**Program Name** Please execute the program and check the output  
demo29.py

```
while True:  
    n = input('Please enter 4 to quit application: ')  
    if n == '4':  
        break  
  
    print("Thank you")
```

**Output**

## Data Science – Python – Practice - Flow Control

**Program Name** Please execute the program and check the output  
demo30.py

```
number = 4
running = True

while running:
    guess = int(input('Please guess number between 1 to 10 : '))
    if guess == number:
        print('Congratulations Dude, you guessed it but no prizes :)')
        running = False
    elif guess < number:
        print('No, it is a little higher than that.')
    else:
        print('No, it is a little lower than that.')

else:
    print('The while loop is over.')
```

**Output**

## Data Science – Python – Practice - Flow Control

**Program Name** Please execute the program and check the output  
demo31.py

```
for x in range(1, 10):
    if (x%2==0):
        print(x)
```

**Output**

**Program Name** Please execute the program and check the output  
demo31.py

```
for x in range(1, 20):
    if (x%2==0) and (x%4==0):
        print(x)
```

**Output**

**Program Name** Please execute the program and check the output  
demo32.py

```
data = [12, 45.678, 1+2j, True, 'daniel', [1, 2, 3]]
```

```
for item in data:
    print("Type of ",item, " is ", type(item))
```

**Output**

## Data Science – Python – Practice - Flow Control

### Program

Accepts a sequence of lines and print those lines by using list, when entered blank line then program terminates

Name demo33.py

```
lines = []
print(lines)

while True:
    l = input()

    if l:
        lines.append(l)
    else:
        break

print(lines)
```

### Output

## Data Science – Python – Practice - Flow Control

**Program Name** Please execute the program and check the output  
demo34.py

```
s = input("Enter string, mixed with numbers: ")  
l = 0  
for c in s:  
    if c.isalpha():  
        l = l+1  
  
print("Characters count from entered text:", l)
```

**Output**

**Program Name** Please execute the program and check the output  
demo35.py

```
s = input("Enter string, mixed with numbers: ")  
l = 0  
for c in s:  
    if c.isdigit():  
        l = l+1  
  
print("Numbers count from entered text:", l)
```

**Output**

## Data Science – Python – Practice - Flow Control

**Program Name** Please execute the program and check the output  
demo36.py

```
I = input("Enter any alphabet letter: ")

if I in ('a', 'e', 'i', 'o', 'u'):
    print(I, " is a vowel.")

else:
    print(I, " is a consonant.")
```

**Output**

**Program Name** Please execute the program and check the output  
demo37.py

```
I = input("Enter any alphabet letter: ")

if len(I)==1:

    if I in ('a', 'e', 'i', 'o', 'u'):
        print(I, " is a vowel.")

    else:
        print(I, " is a consonant.")

else:
    print("Invalid input")
```

**Output**

## Data Science – Python – Practice - Flow Control

**Program Name**

Please execute the program and check the output  
demo38.py

```
print("List of months: January, February, March, April, May,  
June, July, August, September, October, November, December")  
  
month_name = input("Enter name of Month: ")  
  
if month_name == "February":  
    print("No. of days: 28/29 days")  
  
elif month_name in ["April", "June", "September", "November"]:  
    print("No. of days: 30 days")  
  
elif month_name in ["January", "March", "May", "July",  
"August", "October", "December"]:  
    print("No. of days: 31 day")  
  
else:  
    print("Any month name like", month_name, "?")
```

**Output**

## Data Science – Python – Practice - Flow Control

**Program Name** Please execute the program and check the output  
demo39.py

```
n = int(input("Enter a number for table: "))

for i in range(1,11):
    print(n, 'x', i,'=',n*i)
```

**Output**

**Program Name** Please execute the program and check the output  
demo40.py

```
x = ['Daniel', 'abhishek']
for i in x:
    print(i)
```

**Output**

## Data Science – Python – Practice - Flow Control

**Program Name** Please execute the program and check the output  
demo41.py

```
x = ['Daniel', 'abhishek']
for i in x:
    print(i.upper())
```

**Output**

**Program Name** Please execute the program and check the output  
demo42.py

```
x = ['DANIEL', 'ABHISHEK']
for i in x:
    print(i)
```

**Output**

**Program Name** Please execute the program and check the output  
demo43.py

```
x = ['DANIEL', 'ABHISHEK']
for i in x:
    print(i.lower())
```

**Output**

## Data Science – Python – Practice - Flow Control

**Program Name** Please execute the program and check the output  
demo44.py

```
x = [1, 2, 3]
for i in x:
    print(i.upper())
```

**Output**

**Program Name** Please execute the program and check the output  
demo45.py

```
i = 1
while i<=20:
    if i%2 == 0:
        break
    print(i)
    i += 2
```

**Output**

## Data Science – Python – Practice - Flow Control

**Program Name** Please execute the program and check the output  
demo46.py

```
i = 1
while i<=20:
    if i%3 == 0:
        break
    print(i)
    i += 2
```

**Output**

**Program Name** Please execute the program and check the output  
demo47.py

```
i = 1
while False:
    if i%3 == 0:
        break
    print(i)
    i += 2
```

**Output**

## Data Science – Python – Practice - Flow Control

**Program Name** Please execute the program and check the output  
demo48.py

```
True = False
while True:
    print(True)
    break
```

**Output**

**Program Name** Please execute the program and check the output  
demo49.py

```
i = 0
while i < 5:
    print(i)
    i += 1
    if i == 3:
        break
else:
    print(0)
```

**Output**

## Data Science – Python – Practice - Flow Control

**Program Name** Please execute the program and check the output  
demo50.py

```
i = 0
while i < 3:
    print(i)
    i += 1
else:
    print(0)
```

**Output**

**Program Name** Please execute the program and check the output  
demo51.py

```
x = "python"
for i in x:
    print(i)
```

**Output**

## Data Science – Python – Practice - Flow Control

---

**Program Name** Please execute the program and check the output  
demo52.py

```
x = "python"  
for i in x:  
    print(i, end= "")
```

**Output**

**Program Name** Please execute the program and check the output  
demo53.py

```
x = "python"  
for i in x:  
    print(i, end= " ")
```

**Output**

## Data Science – Python – Practice - Flow Control

**Program Name** Please execute the program and check the output  
demo54.py

```
x = "python"  
for i in x:  
    print(i+"Z")
```

**Output**

**Program Name** Please execute the program and check the output  
demo55.py

```
x = "python"  
if i in x:  
    print(i)
```

**Output**

## Data Science – Python – Practice - Flow Control

**Program Name** Please execute the program and check the output  
demo56.py

```
x = "python"  
while i in x:  
    print(i)
```

**Output**

**Program Name** Please execute the program and check the output  
demo57.py

```
x = "python"  
i="a"  
while i in x:  
    print(i)
```

**Output**

**Program Name** Please execute the program and check the output  
demo58.py

```
x = 'abcd'  
for i in range(10):  
    print(i)
```

**Output**

## Data Science – Python – Practice - Flow Control

**Program Name** Please execute the program and check the output  
demo59.py

```
x = 'abcd'  
for i in range(x):  
    print(i)
```

**Output**

**Program Name** Please execute the program and check the output  
demo60.py

```
x = 'abcd'  
for i in range(len(x)):  
    print(i)
```

**Output**

**Program Name** Please execute the program and check the output  
demo61.py

```
x = 'abcd'  
for i in range(x):  
    print(i)
```

**Output**

## Data Science – Python – Practice - Flow Control

**Program Name** Please execute the program and check the output  
demo62.py

```
x = 'abcd'  
for i in range(len(x)):  
    print(i.upper())
```

**Output**

**Program Name** Please execute the program and check the output  
demo63.py

```
x = 'abcd'  
for i in range(len(x)):  
    print(x[i])
```

**Output**

**Program Name** Please execute the program and check the output  
demo64.py

```
x = 'abcd'  
for i in range(len(x)):  
    print(i[x])
```

**Output**

## Data Science – Python – Practice - Flow Control

**Program Name** Please execute the program and check the output  
demo65.py

```
x = 123
for i in x:
    print(i)
```

**Output**

**Program Name** Please execute the program and check the output  
demo66.py

```
for i in range(0):
    print(i)
```

**Output**

**Program Name** Please execute the program and check the output  
demo67.py

```
for i in range(2):
    print(i)
```

**Output**

## Data Science – Python – Practice - Flow Control

**Program Name** Please execute the program and check the output  
demo68.py

```
for i in range(2.0):  
    print(i)
```

**Output**

**Program Name** Please execute the program and check the output  
demo69.py

```
for i in range(int(2.0)):  
    print(i)
```

**Output**

**Program Name** Please execute the program and check the output  
demo70.py

```
for i in "abcd":  
    print(i)
```

**Output**

## Data Science – Python – Practice - Flow Control

**Program Name** Please execute the program and check the output  
demo71.py

```
for i in " ":
    print(i)
```

**Output**

**Program Name** Please execute the program and check the output  
demo72.py

```
x = 2
for i in range(x):
    x += 1
    print (x)
```

**Output**

**Program Name** Please execute the program and check the output  
demo73.py

```
x = 2
for i in range(x):
    x -= 2
    print (x)
```

**Output**

## Data Science – Python – Practice - Flow Control

**Program Name** Please execute the program and check the output  
demo74.py

```
for i in range(10):
    if i == 5:
        break
    else:
        print(i)
else:
    print("no value")
```

**Output**

**Program Name** Please execute the program and check the output  
demo75.py

```
for i in range(5):
    if i == 5:
        break
    else:
        print(i)
else:
    print("no value")
```

**Output**

## Data Science – Python – Practice - Flow Control

---

**Program Name** Please execute the program and check the output  
demo76.py

```
text = "my name is python"  
for i in text:  
    print (i, end=",")
```

**Output**

**11. Python - Functions - Part - 1****Table of Contents**

<b>1. Function .....</b>	2
<b>2. Types of function.....</b>	2
2.1. Pre-defined or built-in functions.....	2
2.2. User Defined Functions:.....	3
<b>3. Function related terminology .....</b>	3
<b>4. Function definition .....</b>	4
4.1. Creating a function.....	4
4.2. Calling a function .....	6
<b>6. A Function can call other function.....</b>	10
<b>5. Based on Parameters: Functions are two types.....</b>	12
5.1. Function without parameters .....	12
5.2. Function with parameters.....	13
<b>7. return keyword in python.....</b>	17
7.1. Function without return.....	18
7.2. Function with return .....	19
7.3. return vs None .....	25
7.4. A function can return multiple values .....	26

**11. Python - Functions - Part - 1****1. Function**

- ✓ A function can contain group of statements which performs the task.

**Advantages**

- ✓ Maintaining the code is an easy way.
- ✓ Code reusability.

**Make a note**

- ✓ `print()` is predefined function in python which prints output on the console.

**2. Types of function**

- ✓ There are two types of functions,
  - Pre-defined or built-in functions
  - User-defined functions

**2.1. Pre-defined or built-in functions**

- ✓ The functions which are already existing in python are called as predefined function

**Examples**

- `print(p)`
- `type(p)`
- `input(p)`

### 2.2. User Defined Functions:

- ✓ Based on requirement a programmer can create his own function, these functions are called as user defined functions.
- ✓ So, practically we will see how to define and use, user defined functions.

### 3. Function related terminology

- ✓ If we want to understand function concept in better way then we need to focus on function related terminology,
  - **def** keyword
  - name of the function
  - parenthesis ()
  - parameters (if required)
  - colon symbol:
  - function body
  - **return** type (optional)

## Data Science - Python Functions - Part - 1

---

### 4. Function definition

- ✓ A function can contains group of statements.
- ✓ The purpose of function is to perform an operation.
- ✓ A function can contain mainly two parts,
  1. Creating a function
  2. Calling a function

#### 4.1. Creating a function

- ✓ Very first step is we need to create a function.
- ✓ We need to use **def** keyword to create a function.
- ✓ After **def** keyword we should write name of the function.
- After function name, we should write parenthesis **()**
  - This parenthesis may contain parameters.
    - Parameters are just like variables which receive the values
    - If function having parameters, then we need to provide the values while calling.
    - We will learn more in parameterized function
- After parenthesis we should write colon : **symbol**
- After : symbol in next line we should provide indentation
- ✓ Function body.
  - Actual logic contains by function body
  - This function body helps to perform the operation.
- ✓ Before closing the function, function may contain return type.

#### Syntax

```
def functionname():
    """ doc string"""
    Body of the function to perform operation
```

## Data Science - Python Functions - Part - 1

### A naming convention to define a function

- ✓ As discussed in Naming convention chapter, function name should be in lower case.
- ✓ If name having multiple words, then separating words with underscore (\_) symbol is a good practice.

**Program Name** Creating a function  
demo1.py

```
# function creation
def display():
    print("Welcome to function")
```

**output**

### Make a note

- ✓ When we execute above program, then function body not executed.
- ✓ To execute function body, we need to call the function.

## Data Science - Python Functions - Part - 1

### 4.2. Calling a function

- ✓ After function is created then we need to call that function to execute the function body.
- ✓ While calling the function, function name should be match otherwise we will get error.

<b>Program Name</b>	Create and call user defined function demo2.py
	# function creation <b>def</b> display(): print("Welcome to function concept")
	# function calling display()
<b>output</b>	Welcome to function concept

## Data Science - Python Functions - Part - 1

**Program Name** Create and call user defined function  
demo3.py

```
# function creation
def display():
    print("Welcome to function concept")

# function calling
display()
display()
```

**output**

```
Welcome to function concept
Welcome to function concept
```

**Program Name** Create and call user defined function  
demo4.py

```
# function creation
def display():
    print("Welcome to function concept")

# function calling
details()
```

**output**

```
NameError: name 'details' is not defined
```

## Data Science - Python Functions - Part - 1

### Question

- Can i create more than one function in a single python program?

### Answer

- Yes, we can
- Based on requirement we can create any number of functions.

**Program Name** Creating two functions and calling those functions  
demo5.py

```
def first():
    print("This is first function")

def second():
    print("This is second function")

first()
second()
```

**output**
This is first function
This is second function

## Data Science - Python Functions - Part - 1

---

**Program Name** Creating two functions and calling those functions  
demo6.py

```
def first():
    print("This is first function")

def second():
    print("This is second function")

second()
first()
```

**output**

```
This is second function
This is first function
```

## Data Science - Python Functions - Part - 1

### 6. A Function can call other function

- ✓ Based on requirement a function can call another function as well.
- ✓ We can call a function inside another function.

#### Syntax

```
def first_function():
    body of the first function
    we can call the secondfunction
```

```
def second_function():
    body of the second function
```

first function calling

**Program Name** Creating two functions  
demo7.py

```
def m1():
    print("first function")

def m2():
    print("second function")
```

```
m1()
m2()
```

**output**

```
first function
second function
```

## Data Science - Python Functions - Part - 1

---

**Program Name** One function can call another function  
demo8.py

```
def m1():
    print("first function")
    m2()

def m2():
    print("second function")

m1()
```

**output**

```
first function
second function
```

## 5. Based on Parameters: Functions are two types

- ✓ Based on parameters, functions can be divided into two types,
  - Function without parameters (**or**) No parameterised function
  - Function with parameters (**or**) Parameterised function

### 5.1. Function without parameters

- ✓ If a function having no parameters then that function is called as, No parameterized function

#### Syntax

```
def nameofthefunction():
    body of the function to perform operations

function calling
```

<b>Program Name</b>	Function which having no parameters demo9.py
	# defining a function <b>def</b> display(): print("Welcome to function which having no parameters")
<b>output</b>	# calling function display()
	Welcome to function which having no parameters

## 5.2. Function with parameters

- ✓ If a function having parameters then that function called as parameterised function

### Why function having parameters?

- ✓ Function parameters help to process the function operation.
- ✓ When we pass parameters then,
  - Function capture parameters values
  - These values perform the operations.
  - Finally it brings the result.

#### Syntax

```
def functionname(parameter1, parameter2, ...):
    body of the function
function calling
```

**Program Name** One parameterized function  
demo10.py

```
def testing(a):
    print("one parameterised function:", a)

testing(10)
```

**output**  
one parameterised function: 10

## Data Science - Python Functions - Part - 1

**Program Name** One parameterized function  
demo11.py

```
def testing(a):
    print("one parameterised function:", a)

testing(10.56)
```

**output**  
one parameterised function: 10.56

**Program Name** One parameterized function  
demo12.py

```
def testing(a):
    print("one parameterised function:", a)

testing("Daniel")
```

**output**  
one parameterised function: Daniel

## Data Science - Python Functions - Part - 1

**Program Name** One parameterized function  
demo13.py

```
def testing(a):  
    print("one parameterised function:", a)
```

```
x = input("Enter a value:")  
testing(x)
```

**output**

```
Enter a value: 10  
one parameterised function: 10
```

**Program Name** Two parameterized function  
demo14.py

```
def testing(a, b):  
    print("two parameterised function:", a, b)
```

```
testing(10, 20)
```

**output**

```
two parameterised function: 10 20
```

## Data Science - Python Functions - Part - 1

---

**Program Name** Function performing addition operation  
demo15.py

```
def addition(a, b):
    print("Addition of two values=", (a+b))

addition(10, 20)
```

**output**  
Addition of two values =30

### 7. return keyword in python

- ✓ Based on return statement, functions can be divided into two types,
  - Function without return statement
  - Function with return statement
- ✓ return is a keyword in python.
- ✓ We should use return statement with function or method, otherwise we will get error.

**Program Name** return outside of function which is invalid  
demo16.py

```
print('Hello')  
return 100
```

**output**  
SyntaxError: 'return' outside function

### 7.1. Function without return

- ✓ If a function cannot contains return statement then that function is called as a function without return statement.
- ✓ It's not mandatory to write return statement to a function.
- ✓ A function without return statement is valid.

**Program Name** Function displaying information  
demo17.py

```
def balance():
    print("My bank balance is: ")
```

```
balance()
```

**output**  
My bank balance is:

## Data Science - Python Functions - Part - 1

### 7.2. Function with return

- ✓ Based on requirement we can write return statement to a function.
- ✓ A function with return statement is valid.

#### Syntax

```
def nameofthefunction():
    body of the function
    return result
```

**Program Name**      Function with return statement displaying information  
                          demo18.py

```
def balance():
    print("My bank balance is: ")
    return 100
```

```
balance()
```

**output**      My bank balance is:

#### Note

- ✓ If a function contains return statement then that function calling we need to assign to a variable.
- ✓ Daniel why we need to assign to a variable?
- ✓ Yes, i will explain please wait in another five minutes, then you can understand.

## Data Science - Python Functions - Part - 1

---

**Program Name** Function with return statement  
demo19.py

```
def balance():
    print("My bank balance is: ")
    return 100

b = balance()
print(b)
```

**output**  
My bank balance is:  
100

## Data Science - Python Functions - Part - 1

### Why we need to assign a function calling to a variable?

- ✓ If we assign function calling to a variable then that variable holding the variable value.
- ✓ That variable we can use further in our program.

**Program Name** Function with return statement  
demo20.py

```
def balance():
    return 100

b = balance()

if b==0:
    print("Balance is: ", b)

elif b<=0:
    print("Balance is: ", b, " negative please deposit")

else:
    print("Balance is: ", b)
```

**output**  
Balance is: 100

## Data Science - Python Functions - Part - 1

**Program Name** Function with return statement  
demo21.py

```
def balance():
    return 0

b = balance()

if b == 0:
    print("Balance is: ", b)

elif b<=0:
    print("Balance is: ", b, " negative please deposit")

else:
    print("Balance is: ", b)
```

**output**  
Balance is: 0

## Data Science - Python Functions - Part - 1

**Program Name** Function with return statement  
demo22.py

```
def balance():
    return -50

b = balance()

if b == 0:
    print("Balance is: ", b)

elif b <= 0:
    print("Balance is: ", b, "it is negative please deposit")

else:
    print("Balance is: ", b)
```

**output**  
Balance is: -50 it is negative please deposit

## Data Science - Python Functions - Part - 1

### Note

- ✓ Below program also valid but not recommended.

**Program** Function with return statement

**Name** demo23.py

```
def balance():
    print("My bank balance is: ")
    return 100

print(balance())
```

**output**

```
My bank balance is:
100
```

### Note

- ✓ A method can return a value as well.
- ✓ We will learn this concept again in OOPS chapter.

## Data Science - Python Functions - Part - 1

### 7.3. return vs None

- ✓ If any function is not return anything, then by default that function returns **None** data type.
- ✓ We can also say as, if we are not writing return statement, then default return value is None

**Program Name** function returning the None value  
demo24.py

```
def m1():
    print("This function is returning nothing")

x = m1()
print(x)
```

**output**

```
This function is returning nothing
None
```

**7.4. A function can return multiple values**

- ✓ In python, a function can return multiple values.
- ✓ Based on requirement a function can return multiple values.
  - If function is returning two values then while function calling we need to assign to two variables
  - If function is returning three values then while function calling we need to assign to three variables.
  - If function is returning more than one values, while calling function if we assign with one variable then all values will be stored in tuple.

**Syntax**

```
def name_of_the_function():
    body of the function
    return value1, value2, value3,...,valueN
```

## Data Science - Python Functions - Part - 1

---

**Program Name** Define a function which can return multiple values  
demo25.py

```
def m1():
    a = 10
    b = 11
    return a, b

x, y = m1()

print("first value is:", x)
print("second value is:", y)
```

**output**

```
first value is: 10
second value is: 11
```

## Data Science - Python Functions - Part - 1

**Program Name** Define a function which can return multiple values  
demo26.py

```
def m1():
    a = 10
    b = 11
    c = 12

    return a, b, c

x, y, z = m1()

print("first value is:", x)
print("second value is:", y)
print("third value is:", z)
```

**output**

first value is: 10  
second value is: 11  
third value is: 12

## Data Science - Python Functions - Part - 1

---

**Program Name** Define a function which can return multiple values  
demo27.py

```
def m1():
    a = 10
    b = 11
    c = 12
    return a, b, c

x = m1()
print("all values:", x)
```

**output**  
all values: (10, 20, 30)

## Data Science - Python Functions - Part - 1

---

**Program Name** A function with parameters and return type.  
demo28.py

```
def add(x, y, z):
    result = x+y+z
    return result

r = add(10, 20, 30)

print("Addition of values:", r)
```

**output**  
Addition of values: 60

## Data Science – Python Practice - String

**Program Name** Please execute the program and check the output  
demo1.py

```
s="hello"  
print(s)  
print(type(s))
```

**Output**

**Program Name** Please execute the program and check the output  
demo2.py

```
s="hello"  
print(length(s))
```

**Output**

**Program Name** Please execute the program and check the output  
demo3.py

```
s="hello"  
print(len(s))
```

**Output**

## Data Science – Python Practice - String

**Program Name** Please execute the program and check the output  
demo4.py

```
s1="hello"  
s2="hello"  
print(id(s1))  
print(id(s2))
```

**Output**

**Program Name** Please execute the program and check the output  
demo5.py

```
s1="hello"  
s2="hello"  
print(s1 is s2)
```

**Output**

**Program Name** Please execute the program and check the output  
demo6.py

```
s1="hello"  
s2="hello"  
s3="hi"  
print(s1 is s2)  
print(s2 is s3)
```

**Output**

## Data Science – Python Practice - String

**Program Name** Please execute the program and check the output  
demo7.py

```
s1="hello"  
s2="hello"  
s3="hi"  
print(s1 == s2)  
print(s2 == s3)
```

**Output**

**Program Name** Please execute the program and check the output  
demo8.py

```
print("a"+"bc")
```

**Output**

**Program Name** Please execute the program and check the output  
demo9.py

```
print("abcd"[2:])
```

**Output**

## Data Science – Python Practice - String

**Program Name** Please execute the program and check the output  
demo10.py

```
str1 = 'hello'  
str2 = ','  
str3 = 'world'  
  
print(str1[-1:])
```

**Output**

**Program Name** Please execute the program and check the output  
demo11.py

```
print(r"\nhello")
```

**Output**

**Program Name** Please execute the program and check the output  
demo12.py

```
print('new' 'line')
```

**Output**

## Data Science – Python Practice - String

**Program Name** Please execute the program and check the output  
demo13.py

```
str1="helloworld"  
  
print(str1[::-1])
```

**Output**

**Program Name** Please execute the program and check the output  
demo14.py

```
example = "snow world"  
example[3] = 's'  
print(example)
```

**Output**

**Program Name** Please execute the program and check the output  
demo15.py

```
l = [1, 2, 3]  
print(min(l))  
print(max(l))
```

**Output**

## Data Science – Python Practice - String

**Program Name** Please execute the program and check the output  
demo16.py

```
q="what are you"  
print(min(q))  
print(max(q))
```

**Output**

**Program Name** Please execute the program and check the output  
demo17.py

```
example="hello"  
print(example.count("l"))
```

**Output**

**Program Name** Please execute the program and check the output  
demo18.py

```
example = "helle"  
print(example.find("e"))
```

**Output**

## Data Science – Python Practice - String

**Program Name** Please execute the program and check the output  
demo19.py

```
example = "helle"  
print(example.rfind("e"))
```

**Output**

**Program Name** Please execute the program and check the output  
demo20.py

```
example="helloworld"  
print(example[::-1])
```

**Output**

**Program Name** Please execute the program and check the output  
demo21.py

```
str1="restart"  
  
char = str1[0]  
str1 = str1.replace(char, '$')  
print(str1)
```

**Output**

## Data Science – Python Practice - String

**Program Name** Please execute the program and check the output  
demo22.py

```
str1="restart"  
  
char = str1[0]  
str1 = str1.replace(char, '$')  
str2 = char + str1[1:]  
  
print(str2)
```

**Output**

**Program Name** Please execute the program and check the output  
demo23.py

```
example="helloworld"  
print(example[::-1].startswith("d"))
```

**Output**

**Program Name** Please execute the program and check the output  
demo24.py

```
print("hello\example\test.txt")
```

**Output**

## Data Science – Python Practice - String

**Program Name** Please execute the program and check the output  
demo25.py

```
print("hello\\example\\test.txt")
```

**Output**

**Program Name** Please execute the program and check the output  
demo26.py

```
print("hello\"example\"test.txt")
```

**Output**

**Program Name** Please execute the program and check the output  
demo27.py

```
print("hello"\example"\test.txt")
```

**Output**

**Program Name** Please execute the program and check the output  
demo28.py

```
print("hello"+1+2+3)
```

**Output**

## Data Science – Python Practice - String

**Program Name** Please execute the program and check the output  
demo29.py

```
print("D", end = ' ')
print("C", end = ' ')
print("B", end = ' ')
print("A", end = ' ')
```

**Output**

**Program Name** Please execute the program and check the output  
demo30.py

```
print("Hello".replace("l", "e"))
```

**Output**

**Program Name** Please execute the program and check the output  
demo31.py

```
print("abc DEF".capitalize())
```

**Output**

**Program Name** Please execute the program and check the output  
demo32.py

```
print("abcdef".upper())
```

**Output**

## Data Science – Python Practice - String

**Program Name** Please execute the program and check the output  
demo33.py

```
print("ABCDEF".upper())
```

**Output**

**Program Name** Please execute the program and check the output  
demo34.py

```
print("ABCDEFG".lower())
```

**Output**

**Program Name** Please execute the program and check the output  
demo35.py

```
print("abcdef".center())
```

**Output**

**Program Name** Please execute the program and check the output  
demo36.py

```
print("xyzxyzxyzxy".count('yy'))
```

**Output**

## Data Science – Python Practice - String

**Program Name** Please execute the program and check the output  
demo37.py

```
print("xyzxyzxyz".count('yy', 1))
```

**Output**

**Program Name** Please execute the program and check the output  
demo38.py

```
print("xyzxyzxyz".count('yy', 4))
```

**Output**

**Program Name** Please execute the program and check the output  
demo39.py

```
print("xyzxyzxyz".endswith("yyy"))
```

**Output**

**Program Name** Please execute the program and check the output  
demo40.py

```
print("ab\lcd\tef")
```

**Output**

## Data Science – Python Practice - String

**Program Name** Please execute the program and check the output  
demo41.py

```
print("a\nb")
```

**Output**

**Program Name** Please execute the program and check the output  
demo42.py

```
print("abcdef".find("cd"))
```

**Output**

**Program Name** Please execute the program and check the output  
demo43.py

```
print("ccdcddcd".find("c"))
```

**Output**

## Data Science – Python Practice - String

---

**Program Name** Please execute the program and check the output  
demo44.py

```
print("Hello {0} and {1}".format('foo', 'bin'))
```

**Output**

**Program Name** Please execute the program and check the output  
demo45.py

```
print("Hello {1} and {0}".format('bin', 'foo'))
```

**Output**

**Program Name** Please execute the program and check the output  
demo46.py

```
print("Hello {} and {}".format('foo', 'bin'))
```

**Output**

## Data Science – Python Practice - String

**Program Name** Please execute the program and check the output  
demo47.py

```
print("Hello {name1} and {name2}".format('foo', 'bin'))
```

**Output**

**Program Name** Please execute the program and check the output  
demo48.py

```
print("Hello {name1} and  
{name2}".format(name1='foo',name2='bin'))
```

**Output**

**Program Name** Please execute the program and check the output  
demo49.py

```
print('The sum of {0} and {1} is {2}'.format(2, 10, 12))
```

**Output**

## Data Science – Python Practice - String

**Program Name** Please execute the program and check the output  
demo50.py

```
print('ab12'.isalnum())
```

**Output**

**Program Name** Please execute the program and check the output  
demo51.py

```
print('ab,12'.isalnum())
```

**Output**

**Program Name** Please execute the program and check the output  
demo52.py

```
print('ab'.isalpha())
```

**Output**

**Program Name** Please execute the program and check the output  
demo53.py

```
print('a B'.isalpha())
```

**Output**

## Data Science – Python Practice - String

**Program Name** Please execute the program and check the output  
demo54.py

```
print('0xa'.isdigit())
```

**Output**

**Program Name** Please execute the program and check the output  
demo55.py

```
print('.isdigit())
```

**Output**

**Program Name** Please execute the program and check the output  
demo56.py

```
print('my_string'.isidentifier())
```

**Output**

## Data Science – Python Practice - String

**Program Name** Please execute the program and check the output  
demo57.py

```
print('$my_string'.isidentifier())
```

**Output**

**Program Name** Please execute the program and check the output  
demo58.py

```
print('abc'.islower())
```

**Output**

**Program Name** Please execute the program and check the output  
demo59.py

```
print('a@ 1.'.islower())
```

**Output**

**Program Name** Please execute the program and check the output  
demo60.py

```
print('11'.isnumeric())
```

**Output**

## Data Science – Python Practice - String

**Program Name** Please execute the program and check the output  
demo61.py

```
print('HelloWorld'.istitle())
```

**Output**

**Program Name** Please execute the program and check the output  
demo62.py

```
print('Hello World'.istitle())
```

**Output**

**Program Name** Please execute the program and check the output  
demo63.py

```
s1=""" hello  
      hi  
      how are you"""  
print(s1)
```

**Output**

## Data Science – Python Practice - String

**Program Name** Please execute the program and check the output  
demo64.py

```
s1=""" hello  
      hi  
      how are you"""  
print(s1.strip())
```

**Output**

**Program Name** Please execute the program and check the output  
demo65.py

```
print('abcdef'.partition('cd'))
```

**Output**

**Program Name** Please execute the program and check the output  
demo66.py

```
print('abcdefcdgh'.partition('cd'))
```

**Output**

## Data Science – Python Practice - String

**Program Name** Please execute the program and check the output  
demo67.py

```
print('abcd'.partition('cd'))
```

**Output**

**Program Name** Please execute the program and check the output  
demo68.py

```
print('abcdef12'.replace('cd', '12'))
```

**Output**

**Program Name** Please execute the program and check the output  
demo69.py

```
print('abef'.replace('cd', '12'))
```

**Output**

**Program Name** Please execute the program and check the output  
demo70.py

```
print('abcdefghijklm'.split('cd'))
```

**Output**

## Data Science – Python Practice - String

**Program Name** Please execute the program and check the output  
demo71.py

```
print('Ab!2'.swapcase())
```

**Output**

**Program Name** Please execute the program and check the output  
demo72.py

```
print('ab cd ef'.title())
```

**Output**

**Program Name** Please execute the program and check the output  
demo73.py

```
print('ab cd-ef'.title())
```

**Output**

## 1. Accessing string characters

- ✓ We can access string characters by using,

- Indexing
- Slicing

### 1.1. Indexing in string

- ✓ Indexing means a position of string's characters where it stores.
- ✓ We need to use square brackets [] to access the string index.
- ✓ String indexing result is string type.

### Python support two types of indexes

- ✓ Positive index
- ✓ Negative index

### 1.2. Positive index

- ✓ The position of string characters can be positive index from **left to right** direction (we can say forward direction).
- ✓ In this way, the starting position is **0**(zero)

### 1.3. Negative index

- ✓ The position of string characters can be negative index from **right to left** direction (we can say backward direction).
- ✓ In this way, the starting position is **-1**(minus one)
  - Please don't think too much like **-0** (minus zero), there is no minus zero boss.

## Data Science – Python Practice - String

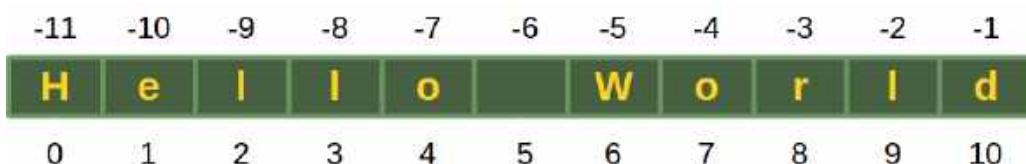
### Internal representation

**Program Name** Printing text message  
demo74.py

```
wish = "Hello World"  
print(wish)
```

**output**  
Hello World

### Diagram representation



**Program Name** Accessing string index by using integer values.  
demo75.py

```
wish = "Hello World"  
  
print(wish[0])  
print(wish[1])
```

**output**  
H  
e

**Program Name** Accessing string index by using negative values.  
demo76.py

```
wish = "Hello World"
```

```
print(wish[-1])
print(wish[-2])
```

**output**

```
d
l
```

#### 1.4. Slicing in string

- ✓ A substring of a string is called as a slice.
- ✓ A slice can represent a part of string from string or a piece of string.
- ✓ String slicing result is string type.
- ✓ We need to use square brackets [] in slicing.

**Syntax 1:**

```
nameofthestring [start: stop]
```

- ✓ **start**
  - It indicates the index where slice can start.
  - Default value is 0
- ✓ **stop**
  - It indicates the index where slice can end.
  - Default value is max allowed index of list i.e. length of the string

## Data Science – Python Practice - String

---

### Syntax2:

nameofthestring [start : stop :step]

#### ✓ **start**

- It indicates the index where slice can start.
- Default value is 0

#### ✓ **stop**

- It indicates the index where slice can end.
- Default value is max allowed index of list i.e. length of the string

#### ✓ **Step size**

- Increment value.
- Default value is 1

## 2. String several cases in slicing

### Make a note

- ✓ If we are not specifying begin index, then it will consider from beginning of the string.
- ✓ If we are not specifying end index, then it will consider up to end of the string.
- ✓ The default value for step is 1

```
wish = "Hello World"
```

- ✓ wish[:] => accessing from 0<sup>th</sup> to last
- ✓ wish[::] => accessing from 0<sup>th</sup> to last
- ✓ wish[2::] => accessing from 2<sup>nd</sup> position to till last.
- ✓ wish[4::] => accessing from 4<sup>th</sup> position to till last.
- ✓ wish[:8:] => accessing from 0<sup>th</sup> to 7<sup>th</sup> in steps of 1
- ✓ wish[0:9:1] => accessing string from 0<sup>th</sup> to 8<sup>th</sup> means (9-1) element.

**Program Name** slice operator several use cases  
demo77.py

```
wish = "Hello World"

print(wish[:])
print(wish[::])
print(wish[2::])
print(wish[4::])
print(wish[:8:])
print(wish[0:9:1])
```

**Output**

## Data Science – Python Practice - String

---

```
Hello World  
Hello World  
llo World  
o World  
Hello Wo  
Hello Wor
```

**12. Python - Functions - Part 2****Table of Contents**

<b>1. Formal and actual arguments .....</b>	2
<b>2. Types of arguments .....</b>	3
2.1. Positional arguments .....	4
2.2. Keyword arguments.....	6
2.3. Default arguments .....	8
2.4. * args or variable length arguments .....	10
<b>3. Anonymous functions or Lambdas .....</b>	13
3.1. map(p1, p2) function .....	17
3.2. filter(p1, p2) function.....	18
3.3. reduce(p1, p2) function .....	19

## 12. Python - Functions - Part 2

### 1. Formal and actual arguments

- ✓ When a function is defined it may have some parameters.
- ✓ These parameters receive the values.
  - Parameters are called as a '**formal arguments**'
  - When we call the function, we should pass values or data to the function.
    - These values are called as '**actual arguments**'

**Program Name**      Formal and actual arguments  
demo1.py

```
def add(a, b):  
    c = a + b  
    print(c)  
  
# call the function  
  
x = 10  
y = 15  
add(x, y)
```

**output**  
25

- ✓ **a** and **b** called as formal arguments
- ✓ **x** and **y** called actual arguments

### 2. Types of arguments

In python there are 4 types of actual arguments are existing,

1. positional arguments
2. keyword arguments
3. default arguments
4. variable length arguments (\*args)

## Data Science - Python Functions - Part 2

### 2.1. Positional arguments

- ✓ These are the arguments passed to a function in correct positional order.
- ✓ The number of arguments and position of arguments should be matched, otherwise we will get error.

**Program Name** Positional arguments  
demo2.py

```
def sub(x, y):  
    print(x-y)
```

# calling function

```
sub(20, 10)
```

**output**  
10

### Make a note

- ✓ If we change the number of arguments, then we will get error.
- ✓ This function accepts two arguments then if we are trying to provide three values then we will get error.

## Data Science - Python Functions - Part 2

Program Error: Positional arguments  
Name demo3.py

```
def sub(x, y):
    print(x-y)

# calling function

sub(10, 20, 30)
```

output

TypeError: sub() takes 2 positional arguments but 3 were given

## Data Science - Python Functions - Part 2

### 2.2. Keyword arguments

- ✓ Keyword arguments are arguments that recognize the parameters by the name of the parameters.
- ✓ We can use an identifier (name of the variable) to provide values to the function parameters.

**Program Name** keyword arguments  
demo4.py

```
def cart(product, price):
    print("Product is :" , product)
    print("cost is :" , price)

cart(product = "bangles", price = 200000)
```

**output**

Product is: bangles  
cost is: 200000

**Program Name** keyword arguments  
demo5.py

```
def cart(product, price):
    print("Product is :" , product)
    print("cost is :" , price)

cart(product = "handbag ", price = 100000)
```

**output**

Product is: handbag  
cost is: 100000

## Data Science - Python Functions - Part 2

---

**Program Name** keyword arguments  
demo6.py

```
def cart(product, price):
    print("Product is:" , product)
    print("cost is:" , price)

cart(price = 1200, product = "shirt")
```

**output**
Product is: shirt
cost is: 1200

## Data Science - Python Functions - Part 2

### 2.3. Default arguments

- ✓ During function definition we can provide default values to function parameters
- ✓ While creating a function, we can provide values to the function parameters.

**Program Name** Default arguments  
demo7.py

```
def cart(product, price = 40.0):
    print("Product is :" , product)
    print("cost is :" , price)

# calling function

cart(product = "pen")
```

**output**  
Product is: pen  
Cost is : 40.0

**Program Name** Default arguments  
demo8.py

```
def cart(product, price = 40.0):
    print("product is :", product)
    print("cost is :", price)

cart(product = "handbag", price = 10000)
```

**output**  
Product is: handbag  
Cost is : 10000

## Data Science - Python Functions - Part 2

**Program Name** Default arguments  
demo9.py

```
def cart(product, price = 40.0):
    print("Product is:", product)
    print("cost is:", price)

cart(price = 500, product = "shirt")
```

**output**  
Product is: shirt  
Cost is : 500

### Make a note

- ✓ If we are not passing any value, then only default value will be considered.

## 2.4. \* args or variable length arguments

- ✓ \*args also called as variable length arguments.
- ✓ Sometimes our requirement can be like, need to provide more values to function parameter to process the result. Here we can use \*args concept.
- ✓ It represents single star symbol before the argument name.
- ✓ Internally the provided values will be stored in a tuple.

### Few points

- ✓ If we define **one** parameterised function then during function calling we need to pass **one** value.
- ✓ If we define **two** parameterised function then during function calling we need to pass **two** values, if we pass more or less than two values then we will get error.

#### Syntax

```
def nameofthefunction(*x):  
    body of the function
```

- **\*x** is variable length argument
- ✓ Now we can pass any number of values to this **\*x**.
- ✓ Internally the provided values will be represented in **tuple**.

## Data Science - Python Functions - Part 2

**Program Name** Variable length argument  
demo10.py

```
def m(x):  
    print(x)
```

```
m(10)
```

**output**  
10

**Program Name** Variable length argument  
demo11.py

```
def m(x):  
    print(x)
```

```
m(10, 20)
```

**output**  
TypeError: m() takes 1 positional argument but 2 were given

**Program Name** Variable length argument  
demo12.py

```
def m(*x):  
    print(x)
```

```
m(10)
```

**output**  
(10)

## Data Science - Python Functions - Part 2

**Program Name** Variable length argument  
demo13.py

```
def m(*x):  
    print(x)
```

```
m(10, 20)
```

**output**  
(10, 20)

**Program Name** Variable length argument  
demo14.py

```
def m(*x):  
    print(x)
```

```
m(10, 20, 30)
```

**output**  
(10, 20, 30)

### 3. Anonymous functions or Lambdas

- ✓ Generally we can create normal function by using def keyword
- ✓ lambda is a keyword in python.
- ✓ By using **lambda** keyword we can create lambda function.
- ✓ Lambda function also called as anonymous function.
- ✓ A function without a name is called as anonymous function.
- ✓ Lambda function will process the input and return the result.

#### Lambda function syntax

```
lambda argument_list: expression
```

#### Advantage

- ✓ By using Lambda Functions, we can write very concise code so that readability of the program will be improved.

**Program**      anonymous function  
**Name**           demo16.py

```
s = lambda a: a*a

x = s(4)
print(x)
```

#### output

16

- ✓ Here because of lambda keyword it creates anonymous function.

## Data Science - Python Functions - Part 2

### A simple difference between normal and lambda functions

**Program Name** To find square by using a normal function  
demo17.py

```
def square(t):
    return t*t

s=square(2)
print(s)
```

**output**  
4

**Program Name** To find sum of two values by using normal function  
demo18.py

```
def add(x, y):
    return x + y

b = add(2, 3)
print(b)
```

**output**  
5

## Data Science - Python Functions - Part 2

**Program Name** To find sum of two values by using anonymous function demo19.py

```
add = lambda x, y: x+y  
  
result = add(1, 2)  
print("The sum of value is:", result)
```

**output**

```
3
```

### Make notes

- ✓ Lambda Function internally returns expression value and we **no need to write return statement** explicitly.

### Where lambda function fits exactly?

- ✓ Sometimes we can pass function as argument to another function. In such cases lambda functions are best choice.
- ✓ We can use lambda functions very commonly with filter(),map() and reduce() functions, these functions expect function as argument.

### Lambda functions

- ✓ map(p1, p2) function
- ✓ filter(p1, p2) function
- ✓ reduce(p1, p2) function

### 3.1. map(p1, p2) function

- ✓ map(fun, iterable) is a predefined function in python.
- ✓ This function takes iterable and apply logic on every item and returns new iterable.

#### Syntax

```
map(function, sequence)
```

**Program Name** Find square by using map function

**Name** demo20.py

```
without_gst_cost = [100, 200, 300, 400]
with_gst_cost = map(lambda x: x+10, without_gst_cost)

x = list(with_gst_cost)
print("Without GST items costs: ", without_gst_cost)
print("With GST items costs: ", x)
```

#### output

```
Without GST items costs: [100, 200, 300, 400]
With GST items costs: [110, 210, 310, 410]
```

## Data Science - Python Functions - Part 2

### 3.2. filter(p1, p2) function

- ✓ filter(fun, iterable) is a predefined function in python.
- ✓ This function takes iterable and apply filtering logic on every item and returns new iterable.

#### Syntax

```
filter(function, sequence)
```

**Program Name** example by using filter function  
demo21.py

```
items_cost = [999, 888, 1100, 1200, 1300, 777]
gt_thousand = filter(lambda x : x>1000, items_cost)

x = list(gt_thousand)
print("Eligible for discount:", x)
```

**output**  
Eligible for discount: [1100, 1200, 1300]

## Data Science - Python Functions - Part 2

### 3.3. reduce(p1, p2) function

- ✓ `reduce(fun, iterable)` is a predefined function existing in `functools` module.
- ✓ This function apply a function of two arguments cumulatively to the items of sequence, from left to right.
- ✓ Finally this function reduce the sequence to a single value.

#### Syntax

```
reduce(function, sequence)
```

#### Program

Name      reduce function

demo22.py

```
from functools import reduce
```

```
items_cost = [111, 222, 333, 444]
```

```
total_cost = reduce(lambda x, y : x+y, items_cost)
```

```
print(total_cost)
```

#### output

```
1110
```

- ✓ For example, `reduce(lambda x, y: x+y, [1, 2, 3, 4, 5])` calculates  $((((1+2)+3)+4)+5)$

**13. Python - Module****Table of Contents**

<b>1. Modules.....</b>	<b>2</b>
<b>2. import module.....</b>	<b>3</b>
<b>3. Renaming or aliasing a module.....</b>	<b>4</b>
<b>4. from and import keywords .....</b>	<b>5</b>
<b>5. import * (star symbol).....</b>	<b>6</b>
<b>6. Member aliasing.....</b>	<b>6</b>

## 13. Python - Module

### 1. Modules

- ✓ In python a module means, a saved python file.
- ✓ This file can contain a group of classes, methods, functions and variables.
- ✓ Every Python file mean **.py** or **.python** extension file is called as a module.

```
Program    module program
Name      additionmultiplication.py

x = 10

def addition(a, b):
    print("Sum of two values: ", (a+b))

def multiplication(a, b):
    print("Multiplication of two values: ", (a*b))
```

- ✓ Now **additionmultiplication.py** file is a module.
- ✓ **additionmultiplication.py** module contains one variable and two functions.

## 2. import module

- ✓ **import** is a keyword in python.
- ✓ By using import keyword we can import (get) modules in our program.
- ✓ Once we imported the module then we can use members (variables, functions & etc) of module.

**Program Name** importing **additionmultiplication** module and calling members  
demo1.py

```
import additionmultiplication

print(additionmultiplication.x)
additionmultiplication.addition(1, 2)
additionmultiplication.multiplication(2, 3)
```

**output**

```
10
Sum of two values: 3
Multiplication of two values: 6
```

### Make a note:

- ✓ Whenever we are using a module in our program, for that module compiled file will be generated and stored in the hard disk permanently.

### 3. Renaming or aliasing a module.

- ✓ **as** is a keyword in python
- ✓ By using **as** keyword we can rename/alias existing module.

#### Syntax

```
import additionmultiplication as admul
```

- ✓ Here **additionmultiplication** is module name and alias name is **admul**
- ✓ We can access members by using alias name **admul**

#### Program

#### Name

```
importing module
```

```
demo2.py
```

```
import additionmultiplication as admul
```

```
print(admul.x)
```

```
admul.addition(1,2)
```

```
admul.multiplication(3, 4)
```

#### output

```
10
```

```
Sum of two values: 3
```

```
Multiplication of two values: 12
```

#### 4. from and import keywords

- ✓ **from** is keyword in python
- ✓ We can import some specific members of module by using **from** keyword.
- ✓ The main advantage of **from** keyword is we can access members directly without using module name.

**Program Name** from and import keywords  
demo3.py

```
from additionmultiplication import x, addition
print(x)
addition(10,20)
```

**output**

```
10
Sum of two values: 30
```

**Program Name** **NameError:** name 'multiplication' is not defined  
demo4.py

```
from additionmultiplication import x, addition
print(x)
multiplication(10,20)
```

**Error**

```
10
NameError: name 'multiplication' is not defined
```

## Data Science - Python - Module

### 5. import \* (star symbol)

- ✓ We can use \* symbol to import all members of a module.
- ✓ We can import all members of a module as by using import \* (symbol)

<p><b>Program Name</b></p> <p>importing by using *</p> <p>demo5.py</p>	<pre>from additionmultiplication import *  print(x) addition(10, 20) multiplication(10, 20)</pre>
<p><b>output</b></p>	<pre>10 Sum of two values: 30 Multiplication of two values: 200</pre>

### 6. Member aliasing

- ✓ We can give alias name to the members of a module

<p><b>Program Name</b></p> <p>member aliasing</p> <p>demo6.py</p>	<pre>from additionmultiplication import x as y, addition as add  print(y) add(10, 20)</pre>
<p><b>output</b></p>	<pre>10 Sum of two values: 30</pre>

### 14. Python Package

#### Contents

1. What is a package?.....	2
2. __init__.py file.....	2
3. Advantage.....	2

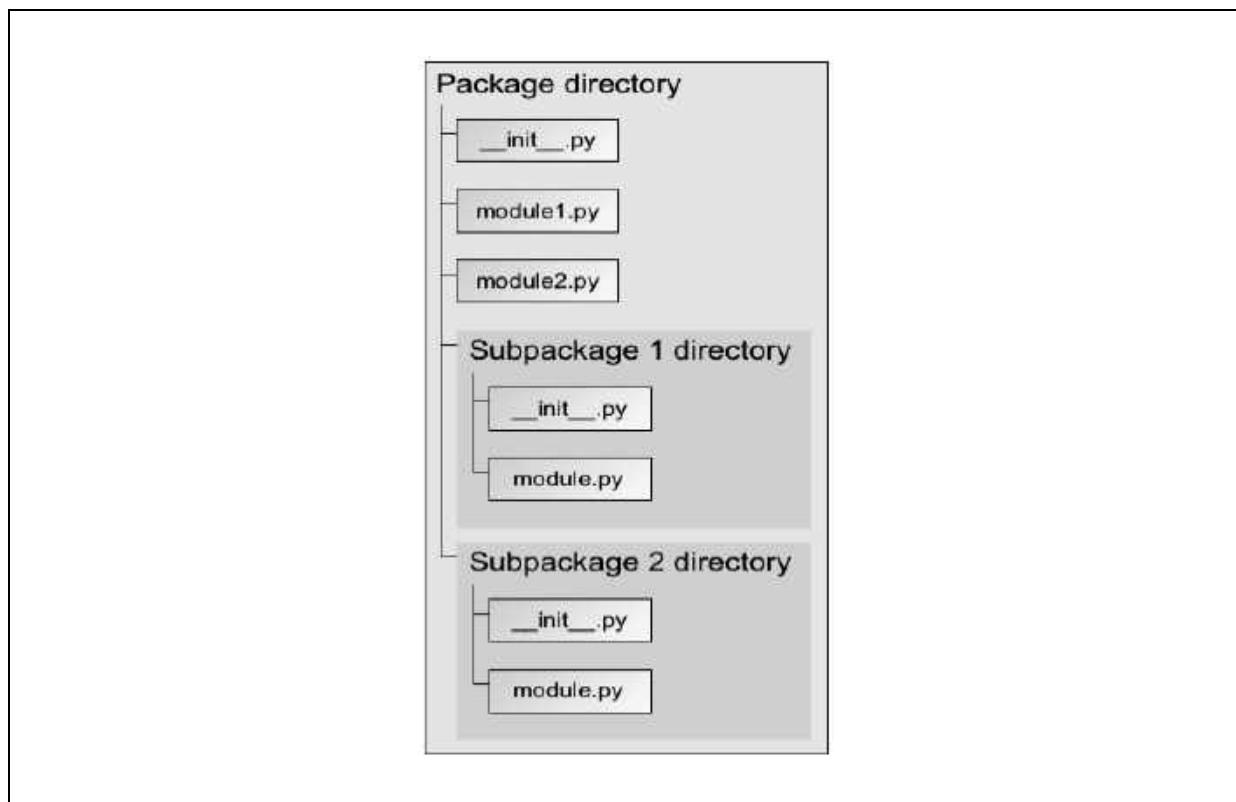
## 14. Python Package

### 1. What is a package?

- ✓ A package is nothing but folder or directory which represents collection of python modules(programs)

### 2. `__init__.py` file

- ✓ Any folder or directory contains `__init__.py` file, is considered as a Python package.
- ✓ `__init__.py` can be empty file.
- ✓ A package can contain sub packages also.



### 3. Advantage

- ✓ We can resolve naming conflicts.
- ✓ We can identify our components uniquely.
- ✓ It improves the modularity of the application.

## Data Science – Python Package

### Example1

```
|  
|---demo1.py  
|  
|---demo2.py  
|  
|---- __init__.py  
|  
|---pack1  
|  
|---- test1.py  
|  
|---- __init__.py
```

**Program** Creating \_\_init\_\_.py file  
**Name** \_\_init\_\_.py

Empty file

**Program** executing a package  
**Name** test1.py

```
def m1():  
    print("Hello this is test1 present in pack1")
```

## Data Science – Python Package

---

**Program Name** executing a package  
demo1.py

```
import pack1.test1  
pack1.test1.m1()
```

**output**  
Hello this is test1 present in pack1

**Program Name** executing a package  
demo2.py

```
from pack1.test1 import m1  
m1()
```

**output**  
Hello this is test1 present in pack1

## Data Science – Python Package

### Example2

```
|---demo3.py  
|  
|-----__init__.py  
|  
|---maindir  
|  
|-----test2.py  
|  
|-----__init__.py  
|  
|----- subdir  
|  
|-----test3.py  
|  
|-----__init__.py
```

**Program Name** package \_\_init\_\_.py

Empty file

**Program Name** executing a package test2.py

```
def m2():  
    print("This is test2 present in maindir")
```

## Data Science – Python Package

**Program Name** executing a package  
test3.py

```
def m3():
    print("This is test3 present in maindir.subdir")
```

**Program Name** executing a package  
demo3.py

```
from maindir.test2 import m2
from maindir.subdir.test3 import m3
```

```
m2()
m3()
```

**output**

```
This is test2 present in maindir
This is test3 present in maindir.subdir
```

### Make a note

- ✓ Summary diagram of library, packages, modules which contains functions, classes and variables.
  - Library - A group of packages
  - Package - A group of modules
  - Modules - A group of variables functions and classes.

**15. Python - List Data Structure****Table of Contents**

<b>1. Why should we learn about data structures?</b> .....	2
<b>2. Python Data structures .....</b>	3
<b>3. Sequence of elements.....</b>	3
<b>4. list data structure .....</b>	4
<b>5. Creating list.....</b>	6
5.1. Creating empty list .....	6
5.2. Creating list with elements .....	6
5.3. Creating list by using list(p) predefined function.....	9
<b>6. list having mutable nature .....</b>	10
<b>7. Accessing elements from list .....</b>	11
7.1. By using index .....	11
7.2. Slicing .....	15
7.3. Accessing list by using for loop .....	17
<b>8. len(p) function.....</b>	18
<b>9. Methods in list data structure .....</b>	19
9.1. count(p)method .....	21
9.2. append(p)method.....	22
9.3. insert(p1, p2) method:.....	23
9.4. remove(p) method:.....	24
9.5. reverse():.....	25
9.6. sort() method:.....	26
<b>10. Mathematical + and * operators .....</b>	27
10.1. Concatenation operator +.....	27
10.2 Repetition operator * .....	27
<b>11. Membership operators .....</b>	28
<b>12. list comprehension .....</b>	29

## 15. Python - List Data Structure

### 1. Why should we learn about data structures?

- ✓ The common requirement in any real time project is like, creating, updating, retrieving, and deleting elements.
- ✓ Few more real times operations are below,
  - Storing
  - Searching
  - Retrieving
  - Deleting
  - Processing
  - Duplicate
  - Ordered
  - Unordered
  - Size
  - Capacity
  - Sorting
  - Un-sorting
  - Random access
  - Keys
  - Values
  - Key – value pairs
- ✓ So, to understand above operations where to use and how to use then we need to learn about data structures.

### 2. Python Data structures

- ✓ If you wanted to store a group of individual objects in a single entity, then you should go for data structures.

### 3. Sequence of elements

- ✓ Data structure also called as sequence.
- ✓ A sequence is a datatype that can contains a group of elements.
- ✓ The purpose of any sequence is, to store and process a group of elements.
- ✓ In python, strings, lists, tuples, set and dictionaries are very important sequence datatype.

## Data Science - Python List Data Structure

---

### 4. list data structure

- ✓ We can **create** list by using,
    - square brackets [] symbols
    - list() predefined function.
  
  - ✓ A list can **store** group of objects or elements.
    - A list can store **same** (Homogeneous) type of elements.
    - A list can store **different** type (Heterogeneous) of elements.
  
  - ✓ A list **size** will increase dynamically.
  
  - ✓ In list **insertion** order is preserved or **fixed**.
    - If we insert elements into 10, 20, 30 then output also will display as 10, 20, 30 then this is called as insertion order is preserved or fixed
    - Example
      - Input                      => [10, 20, 30]
      - Output                      => [10, 20, 30]
  
  - ✓ **Duplicate** elements are allowed.
  - ✓ List having **mutable** nature.
    - Mutable means once we create a list object then we can change or modify the content of list object.
  
  - ✓ Store elements by using **index**.
    - A list data structure supports both positive and negative indexes.
    - Positive index means from left to right
    - Negative index means right to left
- 

**Note:**

- ✓ Inside list every object can be separated by comma separator.
-

### Make a note

- ✓ list is a predefined class in python.
- ✓ Once if we create list object means internally object is creating for list class.
- ✓ What is a class how class works, we will learn in OOPs chapter.

## Data Science - Python List Data Structure

### 5. Creating list

- ✓ We can create list by using square brackets []
- ✓ Inside list, elements will be separated by comma separator

#### 5.1. Creating empty list

- ✓ An empty list is valid

**Program Name** Creating empty list  
demo1.py

```
a = []
print(a)
print(type(a))
```

**Output**
[]  
<class 'list'>

#### 5.2. Creating list with elements

- ✓ We can create list directly with elements.

**Program Name** Creating list with same type of elements  
demo2.py

```
numbers = [10, 20, 30, 40]
print(numbers)
```

**Output**
[10, 20, 30, 40]

## Data Science - Python List Data Structure

**Program Name** Creating list with same type of elements with **duplicates**  
demo3.py

```
numbers = [10, 20, 30, 40, 10, 20, 30, 40]
print(numbers)
```

**Output**  
[10, 20, 30, 40, 10, 20, 30, 40]

**Program Name** creating list with same type of elements  
demo4.py

```
names = ["Daniel", "Prasad", "Ramesh", "Daniel"]
print(names)
```

**Output**  
["Daniel", "Prasad", "Ramesh", "Daniel"]

**Program Name** Creating list with **different** type of elements  
demo5.py

```
student_info = ["Daniel", 10, 35.9]
print(student_info)
```

**Output**  
["Daniel", 10, 35.9]

## Data Science - Python List Data Structure

---

### Note

- ✓ Observe the above programs output,
  - Order is preserved
  - Duplicates are allowed.

### 5.3. Creating list by using list(p) predefined function

- ✓ list(p) is a predefined function in python.
- ✓ By using this function we can create list.
- ✓ list(p) function takes only one parameter.
- ✓ This parameter should be sequence (range, list, set, tuple, etc...) object otherwise we will get error.

<b>Program Name</b>	Creating list by using list(p) function demo6.py
	<pre>r = range(0, 10) a = list(r) print(a)</pre>
<b>output</b>	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

## 6. list having mutable nature

- ✓ Once we created a list object then we can change or modify the elements in the existing list object.
- ✓ So, list having mutable nature.

**Program Name** list having mutable nature  
demo7.py

```
a = [1, 2, 3, 4, 5]
print(a)
print("Before modifying a[0] : ", a[0])

a[0] = 20
print("After modifying a[0] : ", a[0])
print(a)
```

**output**

```
[1, 2, 3, 4, 5]
Before modifying a[0] : 1
After modifying a[0] : 20
[20, 2, 3, 4, 5]
```

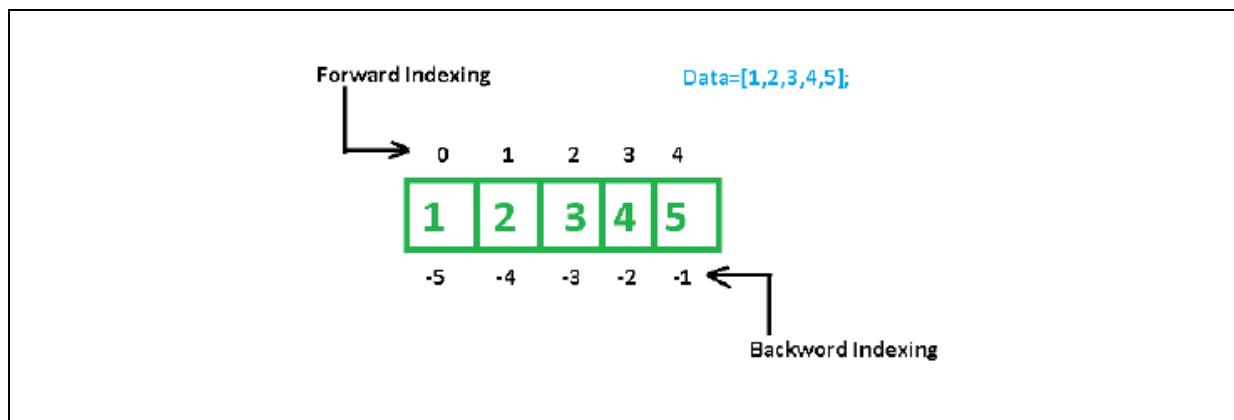
## 7. Accessing elements from list

- ✓ We can access elements from list by using,

1. Index.
2. Slice operator.
3. loops

### 7.1. By using index

- ✓ **index** represents accessing the elements by their position numbers in the list.



- ✓ **Indexing** represents accessing the elements by their position numbers in the list.
- ✓ Index starts from **0** onwards.
- ✓ List supports both positive and negative indexes.
  - Positive index represents from left to right direction
  - Negative index represents from right to left.
- ✓ If we are trying to access beyond the range of list index, then we will get error like **IndexError**.

## Data Science - Python List Data Structure

**Program Name** list indexing  
demo8.py

```
names = ["Daniel", "Prasad", "Ramesh"]

print(names)

print(names[0])
print(names[1])
print(names[2])

print(type(names))
```

**output**

```
['Daniel', 'Prasad', 'Ramesh']
Daniel
Prasad
Ramesh
<class 'list'>
```

## Data Science - Python List Data Structure

**Program Name** list indexing  
demo9.py

```
names = ["Daniel", "Prasad", "Ramesh"]

print(names)

print(names[0])
print(names[1])
print(names[2])

print(type(names))

print(type(names[0]))
print(type(names[1]))
print(type(names[2]))
```

**output**

```
['Daniel', 'Prasad', 'Ramesh']
Daniel
Prasad
Ramesh
<class 'list'>
<class 'str'>
<class 'str'>
<class 'str'>
```

## Data Science - Python List Data Structure

---

**Program Name**    **IndexError**: list index out of range  
demo10.py

```
names = ["Daniel", "Prasad", "Ramesh"]

print(names)
print(names[30])
```

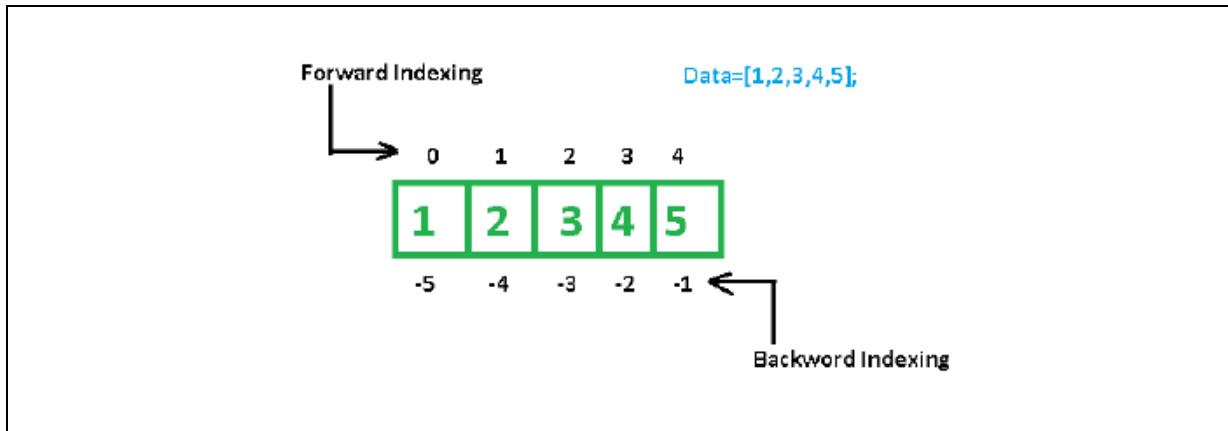
**output**

```
['Daniel', 'Prasad', 'Ramesh']
IndexError: list index out of range
```

## Data Science - Python List Data Structure

### 7.2. Slicing

- ✓ Slicing represents extracting a piece of the list from already created list



### Syntax

`[start: stop: stepsize]`

#### ✓ start

- It indicates the index where slice can start.
- Default value is 0

#### ✓ stop

- It indicates the index where slice can end.
- Default value is max allowed index of list i.e. length of the list

#### ✓ Step size

- Increment value.
- Default value is 1

## Data Science - Python List Data Structure

---

**Program Name** Slice example  
demo11.py

```
n = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(n)
print(n[:])
print(n[::])
print(n[0:5:])
```

**output**

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5]
```

### 7.3. Accessing list by using for loop

- ✓ We can access elements from list by using **for** loop.

**Program Name** accessing elements from list by using for loop  
demo12.py

```
values = [100, 200, 300, 400]
```

```
for value in values:  
    print(value)
```

**output**

```
100  
200  
300  
400
```

## Data Science - Python List Data Structure

### 8. len(p) function

- ✓ By using len(p) predefined function we can find the length of list.
- ✓ This function returns the number of elements present in the list.

**Program Name** To find length of list  
demo13.py

```
values = [10, 20, 30, 40, 50]  
print(len(values))
```

**Output**  
5

## 9. Methods in list data structure

- ✓ As discussed, list is a predefined class.
- ✓ So, list class can contain methods because methods can be created inside of class only.
- ✓ We can check these methods by using `dir(parameter1)` predefined function.
  
- ✓ So, internally list class contains two types of methods,
  - With underscore symbol methods.
    - We no need to focus
  - Without underscore symbol methods.
    - We need to focus much on these

**Program Name** Printing list data structure methods by using `dir(list)` function  
**demo14.py**

```
print(dir(list))
```

**output**

```
[  
  '__add__', ..... , '__subclasshook__',
```

**Important methods**

```
'append', 'count', 'insert', 'remove', 'reverse', 'sort'
```

```
]
```

### Important point

- ✓ As per object-oriented principle,
  - If we want to access instance methods, then we should access by using object name.
- ✓ So, all list methods we can access by using list object.

### Important methods in list

- ✓ count(p) method
- ✓ append(p) method
- ✓ insert() method
- ✓ remove() method
- ✓ reverse() method
- ✓ sort() method

### 9.1. count(p)method

- ✓ count(p) is a predefined method in list class
- ✓ We should access this method by using list object.
- ✓ This method returns the number of occurrences of specific value in the list.

**Program Name** To find count of specific value in list  
demo15.py

```
n = [1, 2, 3, 4, 5, 5, 5, 3]
print(n.count(5))
print(n.count(2))
```

**output**

```
3
1
```

## 9.2. append(p)method

- ✓ append(p) is a predefined method in list class
- ✓ We should access this method by using list object.
- ✓ This method adds object or element to the existing list object.
- ✓ This method will add the object to list at the end of the list.

**Program Name** appending elements into list  
demo16.py

```
a = []
a.append(10)
a.append(20)
a.append(30)
print(a)
```

**output**  
[10, 20, 30]

**Program Name** appending elements into list  
demo17.py

```
a = [10, 20, "Daniel"]

a.append("Naresh")
a.append("Veeru")
print(a)
```

**output**  
[10, 20, "Daniel", "Naresh", "Veeru"]

**9.3. insert(p1, p2) method:**

- ✓ insert(p1, p2) is a predefined method in list class.
- ✓ We should access this method by using list object.
- ✓ By using this method we can insert value at specific position in list.

**Program Name** inserting elements into list  
demo18.py

```
a = [10, 20, 30, 40, 50]
```

```
a.insert(0, 76)  
print(a)
```

**output**

```
[76, 10, 20, 30, 40, 50]
```

append(element)	insert(index, element)
✓ This method adds element at last position.	✓ This method adds element at specific index position.

## Data Science - Python List Data Structure

### 9.4. remove(p) method:

- ✓ remove(p) is a predefined method in list class
- ✓ We should access this method by using list object.
- ✓ By using this method we can remove value from list.

**Program Name** Removing element from list  
demo19.py

```
a = [10, 20, 30]
```

```
a.remove(10)  
print(a)
```

**output**

```
[20, 30]
```

### Ordering elements of List:

#### 9.5. reverse():

- ✓ reverse() is a predefined method in list class
- ✓ We should access this method by using list object.
- ✓ By using this method we can reverse values in list.

**Program** reverse of the list

**Name** demo20.py

```
a = [10, 20, 30, 40]
```

```
print(a)
a.reverse()
print(a)
```

**output**

```
[10, 20, 30, 40]
[40, 30, 20, 10]
```

### 9.6. sort() method:

- ✓ sort() is a predefined method in list class
- ✓ We should access this method by using list object.
- ✓ By default insertion order is fixed.
- ✓ By using this method we can sort values in list.
  - For numbers the order is ascending order.
  - For strings the order is alphabetical order

**Program Name** sorting the numbers and names  
demo21.py

```
a = [10, 40, 50, 20, 30]
a.sort()
print(a)
```

```
b = ['Daniel', 'Ramesh', 'Arjun']
b.sort()
print(b)
```

#### output

```
[10, 20, 30, 40, 50]
['Arjun', 'Daniel', 'Ramesh']
```

## 10. Mathematical + and \* operators

### 10.1. Concatenation operator +

- ✓ '+' operator concatenate two list objects to join them and returns single list.

**Program Name** + operator concatenates the lists  
demo22.py

```
a = [10, 20, 30]  
b = [40, 50, 60]  
c = a + b
```

```
print(c)
```

**output**  
[10, 20, 30, 40, 50, 60]

### 10.2 Repetition operator \*

- ✓ '\*' operator works to repetition of elements in the list.

**Program Name** \* operator repetition the lists  
demo23.py

```
a = [10, 20, 30]
```

```
print(a)  
print(a*2)
```

**output**  
[10, 20, 30]  
[10, 20, 30, 10, 20, 30]

## 11. Membership operators

- ✓ We can check if the element is a member of a list or not by using membership operators those are,
  - **in** operator
  - **not in** operator
- ✓ If the element is member of list, then **in** operator returns True otherwise False.
- ✓ If the element is not in the list, then **not in** operator returns True otherwise False

**Program Name**      Membership operators  
                          demo24.py

```
a = [10, 20, 30, 40, 50]

print(20 in a)          # True
print(20 not in a)      # False

print(90 in a)          # False
print(90 not in a)      # True
```

**output**

```
True
False
False
True
```

## 12. list comprehension

- ✓ List comprehensions represents creating new lists from Iterable object like a list, set, tuple, dictionary and range.
- ✓ List comprehension takes input as iterable, we can apply conditional logic on every item and returns new list.
- ✓ List comprehensions code is very concise way.

### Syntax

```
list = [expression for item1 in iterable1 if statement]
```

- ✓ Here Iterable represents a list, set, tuple, dictionary or range object.
- ✓ The result of list comprehension is new list based on the applying conditions.

<b>Program Name</b>	list comprehension example demo25.py
	<pre>values = [10, 20, 30] result = [value+2 for value in values]  print(values) print(result)</pre>
<b>output</b>	<pre>[10, 20, 30] [12, 22, 32]</pre>

## Data Science - Python List Data Structure

**Program Name** list comprehension example

**Name** demo26.py

```
values = [10, 20, 30]  
result = [value*3 for value in values]
```

```
print(values)  
print(result)
```

**output**

```
[10, 20, 30]  
[30, 60, 90]
```

**Program Name** list comprehension example

**Name** demo27.py

```
values = [10, 20, 30, 40, 50, 60, 70, 80, 90]  
result = [value for value in values if value <= 50]
```

```
print(values)  
print(result)
```

**output**

```
[10, 20, 30, 40, 50, 60, 70, 80, 90]  
[10, 20, 30, 40, 50]
```

## Data Science - Python List Data Structure

**Program Name** square numbers from 1 to 10 by using list comprehension  
demo28.py

```
values = range(1, 11)
squares = [value*2 for value in values]
print(squares)
```

**output**  
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

**16. Python – Tuple Data Structure****Table of Contents**

<b>1. Tuple data Structure.....</b>	2
<b>2. When should we go for tuple data structure? .....</b>	3
<b>3. Syntax Surprise 1: Single value tuple.....</b>	4
<b>4. Syntax Surprise 2. Parenthesis is optional for tuple .....</b>	5
<b>5. Different ways to create a tuple .....</b>	6
<b>6. Accessing elements of tuple: .....</b>	8
6.1 Index.....	8
6.2. Slice operator:.....	9
<b>7. Tuple vs immutability:.....</b>	10
<b>8. Mathematical operators on tuple: .....</b>	11
8.1. Concatenation operator (+): .....	11
8.2 Multiplication operator (*) .....	12
<b>9. len(p) function .....</b>	12
<b>10. Method in tuple data structure .....</b>	13
10.1. count(p) method .....	14
10.2. index(p) method .....	15
<b>12. Differences between List and Tuple: .....</b>	16
<b>13. Can I add elements to this tuple t = (11, 22, [33, 44], 55, 66)? .....</b>	17

## 16. Python – Tuple Data Structure

### 1. Tuple data Structure

- ✓ We can **create** tuple data structure by using,
  - Parenthesis () symbol.
  - Predefined tuple(p) function.
  
- ✓ A tuple can **store** group of objects or elements.
  - A tuple can store **same** (Homogeneous) type of elements.
  - A tuple can store **different** (Heterogeneous) type of elements.
  
- ✓ In tuple insertion **order** is preserved or **fixed**.
  - If we insert elements into 10, 20, 30 then output also will display as 10, 20, 30 then this is called as insertion order is preserved or fixed
  - Example
    - Input      => (10, 20, 30)
    - Output     => (10, 20, 30)
  
- ✓ **Duplicate** elements are **allowed**.
- ✓ Tuple having **immutable** nature.
  - Immutable means once we create a tuple object then we cannot change or modify the content of tuple object.
  
- ✓ Store elements by using **index**.
  - A tuple data structure supports both positive and negative indexes.
  - Positive index means from left to right
  - Negative index means right to left

#### Note:

- ✓ tuple is a predefined class in python.
- ✓ Once if we create tuple object means internally object is creating for tuple class.

## Data Science – Python Tuple Data Structure

### Note:

- ✓ Inside tuple every object can be separated by comma separator.

### 2. When should we go for tuple data structure?

- ✓ If we are going to define a data which never change over all the period, then we should go for tuple data structure.

### Example:

1. Week days names
2. Month names
3. Year names

**Program Name** Tuple having same type of objects  
demo1.py

```
employee_ids = (10, 20, 30, 40, 50)
print(employee_ids)
print(type(employee_ids))
```

**Output**
(10, 20, 30, 40, 50)
<class 'tuple'>

## Data Science – Python Tuple Data Structure

### 3. Syntax Surprise 1: Single value tuple

- ✓ If tuple having only one object, then that object should end with comma separator otherwise python internally not considered as it is tuple.

**Program Name** A single value with tuple syntax, but it's not tuple  
demo2.py

```
number = (9)

print(number)
print(type(number))
```

**Output**  
(9)  
<class 'int'>

**Program Name** A single value with tuple syntax, but it's not tuple  
demo3.py

```
name = ("Daniel")

print(name)
print(type(name))
```

**Output**  
Daniel  
<class 'str'>

## Data Science – Python Tuple Data Structure

**Program Name** Tuple single value ends with comma separator then it's tuple  
demo4.py

```
name = ("Daniel", )  
print(name)  
print(type(name))
```

**Output**  
('Daniel')  
<class 'tuple'>

### 4. Syntax Surprise 2. Parenthesis is optional for tuple

- ✓ While creating a tuple parenthesis is optional

**Program Name** Parenthesis symbol is optional while creating tuple  
demo5.py

```
emp_ids = 10, 20, 30, 40  
print(emp_ids)
```

**output**  
(10, 20, 30, 40)

## Data Science – Python Tuple Data Structure

### 5. Different ways to create a tuple

#### 1. Empty tuple

- ✓ We can create an empty tuple by using empty parenthesis.

```
Program    empty tuple
Name      demo6.py

emp_id = ()
print(emp_id)
print(type(emp_id))

output
()
<class 'tuple'>
```

#### 2. Tuple with group of values

- ✓ Tuple can contain group of objects; those objects can be same type or different type.

```
Program    Tuple example
Name      demo7.py

emp_id = (11, 12, 13)
std_id = 120, 130, 140
print(emp_id)
print(std_id)

output
(11, 12, 13)
(120, 130, 140)
```

## Data Science – Python Tuple Data Structure

**Program Name** Tuple example  
demo8.py

```
t = (11, 12, 13, "daniel")
print(t)
```

**output**

```
(11, 12, 13, "daniel")
```

### 3. By using tuple(p) function

- ✓ We can create tuple by using tuple(p) function.

**Program Name** Creating tuple by using tuple function  
demo9.py

```
a = [11, 22, 33]
t = tuple(a)
print(t)
```

**output**

```
(11, 22, 33)
```

## Data Science – Python Tuple Data Structure

### 6. Accessing elements of tuple:

- ✓ We can access tuple elements by using,
  - Index
  - Slice operator

#### 6.1 Index

- ✓ Index means position where element stores

**Program Name** Accessing tuple by using index  
demo10.py

```
t = (10, 20, 30, 40, 50, 60)
```

```
print(t[0])      #    10
print(t[-1])     #    60
```

#### Output

```
10
60
```

## Data Science – Python Tuple Data Structure

### 6.2. Slice operator:

- ✓ A group of objects from starting point to ending point

**Program Name** Accessing tuple by using slice  
demo11.py

```
t = (10, 20, 30, 40, 50, 60)
```

```
print(t[2:5])
print(t[2:100])
print(t[::-2])
```

**Output**

```
30, 40, 50)
(30, 40, 50, 60)
(10, 30, 50)
```

## Data Science – Python Tuple Data Structure

### 7. Tuple vs immutability:

- ✓ Tuple having immutable nature.
- ✓ If we create a tuple then we cannot modify the elements of existing tuple.

<b>Program Name</b>	Prove tuple having immutable nature demo12.py
	<pre>t = (10, 20, 30, 40) print(t[1]) t[1] = 70</pre>
<b>Output</b>	20 <b>TypeError:</b> 'tuple' object does not support item assignment

### 8. Mathematical operators on tuple:

- ✓ We can apply plus (+) and Multiplication (\*) operators on tuple.
- ✓ + Operator works as concatenation.
- ✓ \* Operator works as multiplication.

#### 8.1. Concatenation operator (+):

- ✓ + operator concatenates two tuples and returns single tuple

**Program Name** Concatenation operator on tuple  
demo13.py

```
t1 = (10,20,30)  
t2 = (40,50,60)  
t3 = t1 + t2  
  
print(t3)
```

**Output**

```
(10, 20, 30, 40, 50, 60)
```

## Data Science – Python Tuple Data Structure

### 8.2 Multiplication operator (\*)

- ✓ Multiplication operator works as repetition operator

**Program Name** Repetition operator on tuple  
demo14.py

```
t1 = (10,20,30)
t2 = t1*3
print(t2)
```

**Output**  
(10, 20, 30, 10, 20, 30, 10, 20, 30)

### 9. len(p) function

- ✓ To return number of elements present in the tuple

**Program Name** len(p) function  
demo15.py

```
t = (10,20,30,40)
print(len(t))
```

**Output**

4

## 10. Method in tuple data structure

- ✓ As discussed, tuple is a predefined class.
- ✓ So, tuple class can contain methods because methods can be created inside of class only.
- ✓ We can check these methods by using `dir(p)` predefined function.
  
- ✓ So, internally tuple class contains two types of methods,
  - With underscore symbol methods.
    - We no need to focus
  - Without underscore symbol methods.
    - We need to focus much on these

**Program Name** Printing tuple data structure methods by using `dir(p)` function  
`demo16.py`

```
print(dir(tuple))
```

**output**

```
[  
    '__add__', ...., '__subclasshook__',  
  
    'count', 'index'  
]
```

### Important point

- ✓ As per object-oriented principle,
  - If we want to access instance method then we should access by using object name.
- ✓ So, all tuple methods we can access by using tuple object.

### Methods in tuple

1. count(parameter1) method
2. index(parameter1) method

#### 10.1. count(p) method

- ✓ count(p) is a method, we should access this method by using tuple object.
- ✓ This method returns the number of occurrences of specified item in the tuple

<b>Program Name</b>	count(p) method demo17.py
	<pre>t = (10, 20, 10, 10, 20) print(t.count(10))</pre>
<b>output</b>	3

## Data Science – Python Tuple Data Structure

### 10.2. index(p) method

- ✓ returns index of first occurrence of the given element.
- ✓ If the specified element is not available, then we will get **ValueError**.

**Program Name** index(p) method  
demo18.py

```
t = (10, 20, 30)  
print(t.index(30))
```

**Output**  
2

**Program Name** index(p) method  
demo19.py

```
t = (10, 20, 30)  
print(t.index(88))
```

**Output**  
**ValueError**: tuple.index(x): x not in tuple

## Data Science – Python Tuple Data Structure

---

### 12. Differences between List and Tuple:

- ✓ List and Tuple are exactly same except small difference:
  - List objects are mutable
  - Tuple objects are immutable.
- ✓ In both cases,
  - Insertion order is preserved.
  - Duplicate objects are allowed
  - Heterogeneous objects are allowed
  - Index and slicing are supported.

<b>List</b>	<b>Tuple</b>
<ul style="list-style-type: none"> <li>✓ List is a Group of Comma separated Values within Square Brackets and Square Brackets are mandatory.</li> <li>✓ Example: <code>i = [10, 20, 30, 40]</code></li> </ul>	<ul style="list-style-type: none"> <li>✓ Tuple is a Group of Comma separated Values within Parenthesis and Parenthesis are optional.</li> <li>✓ Example: <code>t = (10, 20, 30, 40)</code></li> <li>✓ Example: <code>t = 10, 20, 30, 40</code></li> </ul>
<ul style="list-style-type: none"> <li>✓ List Objects are Mutable i.e. once we create List object we can perform any changes in that Object.</li> <li>✓ Example: <code>i[1] = 70</code></li> </ul>	<ul style="list-style-type: none"> <li>✓ Tuple Objects are Immutable i.e. once we create Tuple object we cannot change its content.</li> <li>✓ Example: <code>t[1] = 70</code></li> <li>✓ <b>TypeError</b>: tuple object does not support item assignment.</li> </ul>
<ul style="list-style-type: none"> <li>✓ If the Content is not fixed and keep on changing, then we should go for List.</li> </ul>	<ul style="list-style-type: none"> <li>✓ If the content is fixed and never changes then we should go for Tuple.</li> </ul>

## Data Science – Python Tuple Data Structure

### 13. Can I add elements to this tuple $t = (11, 22, [33, 44], 55, 66)$ ?

- ✓ Yes we can add elements to list in tuple.
- ✓ In second index position list is available, to that we can add

**Program Name** tuple data structure can store any data  
demo23.py

```
t = (11, 22, [33, 44], 55, 66)

t[2].append(77)
print(t)
```

**output**  
(11, 22, [33, 44, 77], 55, 66)

**17. PYTHON - SET DATA STRUCTURE****Table of Contents**

<b>1. Set data structure .....</b>	<b>2</b>
<b>2. When should we go for set? .....</b>	<b>3</b>
<b>3. Creating set by using same type of elements.....</b>	<b>3</b>
<b>4. Creating set by using different type of elements.....</b>	<b>4</b>
<b>5. Creating set by range(p) type of elements.....</b>	<b>4</b>
<b>6. Creating set by using set(p) predefined function.....</b>	<b>5</b>
<b>7. Empty set .....</b>	<b>6</b>
<b>8. len(p) function.....</b>	<b>7</b>
<b>9. Methods in set data structure .....</b>	<b>8</b>
9.1. add(p) method: .....	9
9.2. remove(p) method .....	10
9.3. clear() method .....	10
<b>10. Membership operators: (in and not in) .....</b>	<b>11</b>
<b>11. set comprehensions .....</b>	<b>12</b>
<b>12. Remove duplicates in list elements .....</b>	<b>12</b>

**17. PYTHON - SET DATA STRUCTURE****1. Set data structure**

- ✓ We can **create** set by using,
  - Curly braces {} symbols
  - set() predefined function.
  
- ✓ A set can **store group** of objects or elements.
  - A set can store **same** (Homogeneous) type of elements.
  - A set can store **different** (Heterogeneous)type of elements.
  
- ✓ A set size will **increase** dynamically.
  
- ✓ In set insertion order is not preserved means **not fixed**.
  - If we insert elements into 10, 20, 30, 40 then there is no guarantee for output as 10, 20, 30, 40
  - Example
    - Input           =>   {10, 20, 30, 40}
    - Output         =>   {20, 40, 10, 30}
  
- ✓ Duplicate elements are **not allowed**.
- ✓ Set having **mutable** nature.
  - Mutable means once we create a set object then we can change or modify the content of set object.
  
- ✓ Set data structure **cannot** store the elements in index order.

**Note:**

- ✓ set is a predefined class in python.
- ✓ Once if we create set object means internally object is creating for set class.

**Note:**

- ✓ Inside set every object can be separated by comma separator.

**2. When should we go for set?**

- ✓ If we want to represent a group of **unique** values as a single entity, then we should go for set.
- ✓ set cannot store duplicate elements.
- ✓ Insertion order is not preserved.

**Note**

- ✓ We can create set by using curly braces {} and all elements separated by comma separator in set.

**3. Creating set by using same type of elements**

**Program Name** creating same type of elements by using set  
demo1.py

```
s = {10, 20, 30, 40}  
print(s)  
print(type(s))
```

**Output**  
{40, 10, 20, 30}  
<class 'set'>

## Data Science – Python Set Data Structure

### 4. Creating set by using different type of elements

**Program Name** creating different type of elements by using set  
demo2.py

```
s = {10, "Daniel", 30.9, "Prasad", 40}  
print(s)  
print(type(s))
```

**Output**  
{40, 10, 'Daniel', 'Prasad', 30.9}  
<class 'set'>

### 5. Creating set by range(p) type of elements

**Program Name** creating set by using range(p)  
demo3.py

```
s = set(range(5))  
print(s)
```

**output**  
{0, 1, 2, 3, 4}

## Data Science – Python Set Data Structure

---

**Program Name** set not allowed duplicates  
demo4.py

```
s ={10, 20, 30, 40, 10, 10}
print(s)
print(type(s))
```

**Output**

```
{40, 10, 20, 30}
<class 'set'>
```

### Make a note

- ✓ Observe the above programs output,
  - Order is not preserved
  - Duplicates are not allowed.

### 6. Creating set by using set(p) predefined function

- ✓ We can create set by using set(p) function.
- ✓ set(p) predefined function will take only one parameter.
- ✓ This parameter should be sequence (range, list, set, tuple, etc...) object otherwise we will get error.

**Program Name** creating set by using set(parameter1) function  
demo5.py

```
r = range(0, 10)
l = set(r)
print(l)
```

**output**

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

## Data Science – Python Set Data Structure

### 7. Empty set

- ✓ We can create empty set.
- ✓ While creating empty set compulsory we should use **set()** function.
- ✓ If we didn't use set function, then it treats as dictionary instead of set.

**Program Name** demo6.py

Empty curly braces is not a set, it considers as a dictionary

```
s = {}  
print(s)  
print(type(s))
```

**Output**

```
{}  
<class 'dict'>
```

**Program Name** demo7.py

Using **set()** function to create empty set

```
s = set()  
print(s)  
print(type(s))
```

**Output**

```
set()  
<class 'set'>
```

## Data Science – Python Set Data Structure

### 8. len(p) function

- ✓ By using len(p) predefined function we can find the length of set.
- ✓ This function returns the number of elements present in the set.

**Program Name** To find length of set  
demo8.py

```
n = {10, 20, 30, 40, 50}  
print(len(n))
```

**Output**  
5

**Program Name** To find length of set  
demo9.py

```
n = {10, 20, 30, 40, 50, 10, 10, 10}  
print(len(n))
```

**Output**  
5

## 9. Methods in set data structure

- ✓ As discussed, set is a predefined class.
- ✓ So, set class can contain methods because methods can be created inside of class only.
- ✓ We can check these methods by using `dir(parameter1)` predefined function.
  
- ✓ So, internally set class contains two types of methods,
  - With underscore symbol methods.
    - We no need to focus
  - Without underscore symbol methods.
    - We need to focus much on these

**Program Name** Printing set data structure methods by using `dir(set)` function  
`demo10.py`

```
print(dir(set))
```

**output**

```
[  
'__and__', ..... '__subclasshook__', '__xor__',
```

**Important methods**

```
'add', 'clear', 'remove',
```

```
]
```

**Important point**

- ✓ As per object-oriented principle,
  - If we want to access instance method, then we should access by using object name.
- ✓ So, all set methods we can access by using set object.

**Important methods in set:**

- ✓ add(p) method
- ✓ remove(p) method
- ✓ clear() method

**9.1. add(p) method:**

- ✓ add(p) is a method, we should access this method by using set object.
- ✓ This method adds element to the set.

**Program Name** add(p) function can add element to the set  
demo11.py

```
s = {10, 20, 30}  
s.add(40)  
print(s)
```

**Output**  
{40, 10, 20, 30}

## Data Science – Python Set Data Structure

---

### 9.2. remove(p) method

- ✓ remove(p) is a method in set class, we should access this method by using set object.
- ✓ This method removes specified element from the set.
- ✓ If the specified element not present in the set, then we will get **KeyError**

**Program Name** remove(p) method in set  
demo12.py

```
s = {40, 10, 30, 20}
s.remove(30)
print(s)
```

**Output**  
{40, 10, 20}

### 9.3. clear() method

- ✓ clear() is a method in set class, we should access this method by using set object.
- ✓ This method removes all elements from the set.

**Program Name** clear() method in set  
demo13.py

```
s = {10,20,30}
print(s)
s.clear()
print(s)
```

**Output**  
{10, 20, 30}  
set()

## 10. Membership operators: (in and not in)

- ✓ By using these operators, we can find the specified element is exists in set or not

Program      in operator  
Name          demo14.py

```
s = {1, 2, 3, 'daniel'}
```

```
print(s)
print(1 in s)
print('z' in s)
```

### Output

```
{1, 2, 3, 'daniel'}
True
False
```

## Data Science – Python Set Data Structure

---

### 11. set comprehensions

- ✓ set comprehensions represents creating new set from Iterable object like a list, set, tuple, dictionary and range.
- ✓ set comprehensions code is very concise way.

**Program Name** set comprehension  
demo15.py

```
s = {x*x for x in range(5)}
print(s)
```

**output**  
{0, 1, 4, 9, 16}

### 12. Remove duplicates in list elements

- ✓ We can remove duplicates elements which are exists in list by passing list as parameter to set function

**Program Name** removing duplicates from list  
demo18.py

```
a = [10, 20, 30, 10, 20, 40]
s = set(a)
print(s)
```

**Output**  
{40, 10, 20, 30}

**18. PYTHON - DICTIONARY DATA STRUCTURE****Table of Contents**

<b>1. Dictionary data structure .....</b>	2
<b>2. When should we go for dictionary?.....</b>	3
<b>3. Creating dictionary .....</b>	4
<b>4. Empty dictionary .....</b>	4
<b>5. We can add key-value pairs to empty dictionary.....</b>	5
<b>6. Access values by using keys from dictionary.....</b>	6
<b>7. Update dictionary.....</b>	7
7.1.Case 1.....	7
7.2.Case 2 .....	7
<b>8. Removing or deleting elements from dictionary .....</b>	9
<b>9. len(p) function .....</b>	11
<b>10. Methods in dict class data structure .....</b>	12
10.1. clear() method .....	14
10.2. keys() method .....	15
10.3. values().....	15
10.4. items() .....	16
<b>11. Dictionary Comprehension: .....</b>	17

**18. PYTHON - DICTIONARY DATA STRUCTURE****1. Dictionary data structure**

- ✓ If we want to represent group of **individual objects** as a single entity then we should go for below data structures,
  - list
  - set
  - tuple
  
- ✓ If we want to represent a group of objects as **key-value pairs** then we should go for,
  - dict or dictionary
  
- ✓ We can create dict by using,
  - **Curly braces {} symbols**
  - **dict()** predefined function.
  
- ✓ Dictionary data structure contains **key, value pairs**.
- ✓ **key-value**
  - In dictionary key, value pairs are **separated** by colon : symbol
  - One key-value pair is called as **item**.
  - In dictionary every item is separated by **comma symbol**.
  - In dictionary **duplicate keys** are **not allowed**.
  - In dictionary **duplicate values** can be **allowed**.
  - A dictionary keys and values can store **same** (Homogeneous) type of elements.
  - A dictionary keys and values can store **different** (Heterogeneous) type of elements.
  
- ✓ In dictionary insertion order is not preserved means **not fixed**.
- ✓ Dictionary **size** will **increase** dynamically.
- ✓ Dictionary object having **mutable** nature.
- ✓ Dictionary data structure **cannot** store the elements in **index** order.
  - Indexing and slicing concepts are not applicable

## Data Science – Python Dictionary Data Structure

### Note:

- ✓ dict is a predefined class in python.
- ✓ Once if we create dictionary object means internally object is creating for dict class.

### 2. When should we go for dictionary?

- ✓ If we want to represent a group of objects as **key-value** pairs then we should go for,
  - dict or dictionary

#### Symbols to create data structure for specific data structure?

- ✓ To create **list**, we need to use **square bracket symbols** : []
- ✓ To create **tuple**, we need to use **parenthesis symbols** : ()
- ✓ To create **set** we need to use **curly braces** with values : {}
- ✓ So, to create **dict** we need to use **curly braces** : {}

### Create dictionary

#### Syntax

```
d = { key1 : value1, key2 : value2 }
```

## Data Science – Python Dictionary Data Structure

### 3. Creating dictionary

- ✓ We can create dictionary with key, value pairs.

**Program Name**

creating dictionary

**Name** demo1.py

```
d = {10: "Ramesh", 20: "Arjun", 30: "Daniel"}  
print(d)
```

**output**

```
{10: "Ramesh", 20: "Arjun", 30: "Daniel"}
```

### 4. Empty dictionary

- ✓ We can create empty dictionary.

**Program Name**

creating empty dictionary

**Name** demo2.py

```
d = {}  
print(d)  
print(type(d))
```

**output**

```
{}  
<class 'dict'>
```

## 5. We can add key-value pairs to empty dictionary

- ✓ As we know we can create an empty dictionary.
- ✓ For that empty dictionary we can add key, value pairs.

**Program Name** creating empty dictionary and adding elements  
demo3.py

```
d = []
d[10] = "Ramesh"
d[20] = "Arjun"
d[30] = "Daniel"
print(d)
```

**output**  
{10: 'Ramesh', 20: 'Arjun', 30: 'Daniel'}

## Data Science – Python Dictionary Data Structure

### 6. Access values by using keys from dictionary

- ✓ We can access dictionary values by using keys
- ✓ Keys play main role to access the data.

**Program Name** Accessing dictionary values by using keys  
demo4.py

```
d = {10: "Ramesh", 20: "Arjun", 30: "Daniel"}  
  
print(d[10])  
print(d[20])  
print(d[30])
```

**output**  
Ramesh  
Arjun  
Daniel

**Program Name** Accessing key and value from dictionary using for loop  
demo5.py

```
d = {10: "Ramesh", 20: "Arjun", 30: "Daniel"}  
  
for k in d:  
    print(k, d[k])
```

**output**  
10 Ramesh  
20 Arjun  
30 Daniel

## 7. Update dictionary

- ✓ We can update the key in dictionary.

### Syntax

```
d[key] = value
```

### 7.1.Case 1

- ✓ While updating the key in dictionary, if key is not available then a new key will be added at the end of the dictionary with specified value.

### 7.2.Case 2

- ✓ While updating the key in dictionary, if key is already existing then old value will be replaced with new value.

**Program Name**

**Case 1:** Adding key-value pair to dictionary  
demo6.py

```
d = {10: "Ramesh", 20: "Arjun", 30: "Daniel"}
print("Old dictionary: ",d)

d[99] = "John"
print("Added key-value 99:John pair to dictionary: ", d)
```

### output

Old dictionary: {10: 'Ramesh', 20: 'Arjun', 30: 'Daniel'}

Added key-value 99:John pair to dictionary:  
{10: 'Ramesh', 20: 'Arjun', 30: 'Daniel', 99: 'John'}

## Data Science – Python Dictionary Data Structure

---

**Program Name** Case 2: Updating key-value pair in dictionary  
demo7.py

```
d = {10: 'Ramesh', 20: 'Arjun', 30: 'Daniel'}
print("Old dictionary:", d)
```

```
d[30] = 'Chandhu'
print("Updated dictionary 3:Chandhu :", d)
```

**output**

```
Old dictionary: {10: 'Ramesh', 20: 'Arjun', 30: 'Daniel'}
Updated dictionary 3:Chandhu : {10: 'Ramesh', 20: 'Arjun', 30:
'Chandhu'}
```

## 8. Removing or deleting elements from dictionary

- ✓ By using `del` keyword, we can remove the keys
- ✓ By using `clear()` method we can clear the objects in dictionary

### 8.1. By using `del` keyword

#### Syntax

```
del d[key]
```

- ✓ As per the syntax, it deletes entry associated with the specified key.
- ✓ If the key is not available, then we will get `KeyError`

**Program Name** Deleting key in dictionary  
demo8.py

```
d = {10: "Ramesh", 20: "Arjun", 30: "Daniel"}
print("Before deleting key from dictionary: ", d)
del d[10]
print("After deleting key from dictionary: ", d)
```

**output**  
Before deleting key from dictionary:  
`{10: 'Ramesh', 20: 'Arjun', 30: 'Daniel'}`

After deleting key from dictionary:  
`{20: 'Arjun', 30: 'Daniel'}`

## Data Science – Python Dictionary Data Structure

We can delete total dictionary object

### Syntax

```
del nameofthedictionary
```

- ✓ It can delete the total dictionary object.
- ✓ Once it deletes then we cannot access the dictionary.

**Program Name** Delete key in dictionary

demo9.py

```
d = {10: "Ramesh", 20: "Arjun", 30: "Daniel"}  
print(d)
```

```
del d  
print(d)
```

### output

```
{10: 'Ramesh', 20: 'Arjun', 30: 'Daniel'}  
NameError: name 'd' is not defined
```

## Data Science – Python Dictionary Data Structure

### 9. len(p) function

- ✓ This function returns the number of items in the dictionary

**Program** Finding length of dictionary

**Name** demo10.py

```
d = {100: "Ramesh", 200: "Arjun"}  
print("length of dictionary is: ",len(d))
```

**output**

```
length of dictionary is:2
```

## 10. Methods in dict class data structure

- ✓ As discussed dict is a predefined class.
- ✓ So, dict class can contain methods because methods can be created inside of class only.
- ✓ We can check these method by using `dir(parameter1)` predefined function.
  
- ✓ So, internally dict class contains two types of methods,
  - With underscore symbol methods.
    - We no need to focus
  - Without underscore symbol methods.
    - We need to focus much on these

**Program Name** Printing dict data structure methods by using `dir(dict)` function  
`demo11.py`

```
print(dir(dict))
```

**output**

```
[
```

```
'__class__', .....'__subclasshook__',
```

**Important methods**

```
'clear', 'items', 'keys', 'values'
```

```
]
```

## Data Science – Python Dictionary Data Structure

---

### Important point

- ✓ As per object-oriented principle,
  - If we want to access instance methods, then we should access by using object name.
- ✓ So, all dict methods we can access by using dict object.

### Important methods

- ✓ clear() method
- ✓ keys() method
- ✓ values() method
- ✓ items() method

## Data Science – Python Dictionary Data Structure

### 10.1. clear() method

- ✓ clear() is a method in dict class, we should access this method by using dictionary object.
- ✓ This method removes all entries in dictionary.
- ✓ After deleting all entries, it just keeps empty dictionary

**Program Name** removing dictionary object by using clear() method  
demo12.py

```
d = {10: "Ramesh", 20: "Arjun", 30:"Daniel"}  
print(d)  
d.clear()  
print("After cleared entries in dictionary: ", d)
```

**output**  
{10: 'Ramesh', 20: 'Arjun', 30: 'Daniel'}  
After cleared entries in dictionary: {}

## Data Science – Python Dictionary Data Structure

---

### 10.2. keys() method

- ✓ keys() is a method in dict class, we should access this method by using dict object.
- ✓ This method returns all keys associated with dictionary

**Program Name** keys() method  
demo13.py

```
d = {100: "Ramesh", 200: "Daniel", 300: "Mohan"}
print(d)
print(d.keys())
```

#### Output

```
{100: 'Ramesh', 200: 'Daniel', 300: 'Mohan'}
dict_keys([100, 200, 300])
```

### 10.3. values()

- ✓ values() is a method in dict class, we should access this method by using dict object.
- ✓ This method returns all values associated with the dictionary

**Program Name** values() method  
demo14.py

```
d = {100: "Ramesh", 200: "Daniel", 300: "Mohan"}
print(d)
print(d.values())
```

#### Output

```
{100: 'Ramesh', 200: 'Daniel', 300: 'Mohan'}
dict_values(['Ramesh', 'Daniel', 'Mohan'])
```

## Data Science – Python Dictionary Data Structure

### 10.4. items()

- ✓ items() is a method in dict class, we should access this method by using dict object.
- ✓ By using this method we can access keys and values from the dictionary.

**Program Name** Accessing key and value from dictionary using items() method  
demo15.py

```
d = {10: "Ramesh", 20: "Arjun", 30: "Daniel"}
```

```
for k, v in d.items():
    print(k, v)
```

**output**

```
10 Ramesh
20 Arjun
30 Daniel
```

## Data Science – Python Dictionary Data Structure

### 11. Dictionary Comprehension:

- ✓ A dictionary comprehension represents creating new dictionary from Iterable object like a list, set, tuple, dictionary and range.
- ✓ Dictionary comprehensions code is very concise way.

**Program Name**      Dictionary comprehension  
                  demo15.py

```
squares = {a: a*a for a in range(1,6)}
print(squares)
```

**Output**
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

**19. PYTHON – OBJECT ORIENTED PROGRAMMING****Table of Contents**

<b>1. Is Python follows Functional approach or Object-oriented approach?</b> .....	2
<b>2. Features of Object-Oriented Programming System</b> .....	2
<b>3. class:</b> .....	3
3.1. Def1:.....	3
3.2. Def2:.....	3
<b>4. How to define or create a class?</b> .....	4
<b>5. Brief discussion about class</b> .....	4
<b>6. object</b> .....	7
6.1. Why should we create an object?.....	7
6.2. What is an object? .....	7
6.3. Syntax to create an object .....	8
<b>7. Constructor</b> .....	10
7.1. What is the main purpose of constructor?.....	10
7.2. When constructor will be executed? .....	11
7.3. How many times Constructor will executes?.....	11
7.4. Types of constructors.....	12
7.5. Constructor without parameters .....	12
7.6. Creating parameterised constructor.....	13
<b>8. Difference between method and constructor</b> .....	17
<b>9. Instance variables:</b> .....	18
9.1. What is instance variable? .....	18
9.2. Separate copy instance variable for every object.....	18
9.3. Declaring & accessing instance variables.....	19
<b>10. Instance methods</b> .....	20
<b>11. self pre-defined variable</b> .....	21

**19. PYTHON – OBJECT ORIENTED PROGRAMMING**

- ✓ Object-Oriented Programming is a methodology to design software by using classes and objects.
- ✓ It simplifies the software development and maintenance by providing the below features,

**1. Is Python follows Functional approach or Object-oriented approach?**

- ✓ Python supports both functional and object-oriented programming.

**2. Features of Object-Oriented Programming System**

- ✓ class
- ✓ object
- ✓ constructor
- ✓ Inheritance & etc...

## 3. class:

### 3.1. Def1:

- ✓ A class is a model for creating an object and it does not exist physically.

### 3.2. Def2:

- ✓ A class is a specification (idea/plan/theory) of properties and actions of objects.

#### Syntax

```
class NameOfTheClass:  
    1. constructor  
    2. properties (attributes)  
    3. actions (behaviour)
```

- ✓ We can create class by using **class** keyword.
- ✓ class can contain,

- constructor
- properties
- actions

- ✓ Properties also called as variables.
- ✓ Actions also called as methods.

#### 4. How to define or create a class?

- ✓ A python class may contain the below things,

##### Syntax

```
class NameOfTheClass:  
    """ documentation string ""  
  
    1. Constructor  
  
    2. Variables  
        1. instance variables  
  
    3. Methods  
        1. instance methods
```

#### 5. Brief discussion about class

- ✓ We can create class by using **class** keyword.
- ✓ class keyword follows the **name of the class**.
- ✓ After name of the class we should give **colon :** symbol.
- ✓ After : colon symbol in next line we should provide the **indentation**, otherwise we will get error.
- ✓ class can contain,
  - **Constructor** are used for initialization purpose
  - **Variables** are used to represent the data.
    - **instance** variables
  - **Methods** are used to represent actions.
    - **instance** methods

### Class naming convention

- ✓ While writing a class we need to follow the naming convention to meet real time standards,
  - class names should start with upper case and remaining letters are in lower case.
    - **Example:** Student
  - If name having multiple words, then every inner word should start with upper case letter.
    - **Example:** StudentInfo

### Note

- ✓ Documentation string represents description of the class. Within the class doc string is always optional.

<b>Program Name</b>	Define a class demo1.py
	<code>class Employee:     def display(self):         print("Hello My name is Daniel")</code>
<b>output</b>	

### Make a note

- ✓ In above program, when we run then we will not get any output because we didn't call display method
- ✓ Above program Employee represents a **class** which is defined by developer.
- ✓ Developer defined only one method as display(self)
- ✓ Method we can define by using **def** keyword.
- ✓ Methods means it's just like a functions to perform an operations

## Data Science – Python Object Oriented Programming

---

### Kind info:

- ✓ Writing a class is not enough; we should know how to use the variables and methods.

**So,**

- ✓ We need to create an object to access instance data(variables/methods) of a class.

## 6. object

### 6.1. Why should we create an object?

- ✓ As per requirement we used to define variables and methods in a class.
- ✓ These variables and methods hold the data or values.
- ✓ When we create an object for a class, then only data will be stored for the data members of a class.

### 6.2. What is an object?

#### Definition 1:

- ✓ Instance of a class is known as an object.
  - Instance is a mechanism to allocate enough memory space for data members of a class.

#### Definition 2:

- ✓ Grouped item is known as an object.
  - Grouped item is a variable which stores more than one value.

#### Definition 3:

- ✓ Real world entities are called as objects.

#### Make some notes

- ✓ An object exists physically in this world, but class does not exist.

## Data Science – Python Object Oriented Programming

### 6.3. Syntax to create an object

#### Syntax

```
nameoftheobject = nameoftheclass()
```

#### Example

```
emp = Employee()
```

#### Program

#### Name

Creating a class and object

demo2.py

```
class Employee:  
    def display(self):  
        print("Hello my name is Daniel")
```

```
emp = Employee()  
emp.display()
```

#### output

Hello my name is Daniel

## Data Science – Python Object Oriented Programming

**Program Name** Creating a class and object  
demo2.py

```
class Employee:  
    def display(self):  
        print("Hello my name is Daniel")  
  
    def teaching(self):  
        print("I like teaching")  
  
emp = Employee()  
  
emp.display()  
emp.teaching()
```

**output**  
Hello my name is Daniel  
I like teaching

### Make a note

- ✓ We can create object for class.
- ✓ In the above example **emp** is object name.
  - emp is just like a variable
- ✓ above example, **display(self)** is instance method.
  - To access instance method, we should create an object
  - So, we are accessing instance methods by using object name

## 7. Constructor

- ✓ Constructor is a special kind of method in python.
- ✓ So, we can create constructor by using **def** keyword
- ✓ The name of the constructor should be **\_\_init\_\_(self)**
  - Two underscore symbols before and after init with self as parameter
- ✓ self should be first parameter in constructor,

### Syntax

```
class NameOfTheClass:  
    def __init__(self):  
        body of the constructor
```

### 7.1. What is the main purpose of constructor?

- ✓ The main purpose of constructor is to initialize instance variables.

## Data Science – Python Object Oriented Programming

### 7.2. When constructor will be executed?

- ✓ Constructor will be executed automatically at the time of object creation.

**Program Name** Creating a constructor  
demo3.py

```
class Employee:  
    def __init__(self):  
        print("constructor is executed")  
  
emp = Employee()
```

**output**  
constructor is executed

### 7.3. How many times Constructor will executes?

- ✓ If we create object in two times then constructor will execute two times.

**Program Name** Creating a constructor  
demo3.py

```
class Employee:  
    def __init__(self):  
        print("constructor is executed")  
  
emp1 = Employee()  
emp2 = Employee()
```

**output**  
constructor is executed  
constructor is executed

#### 7.4. Types of constructors

- ✓ Based on parameters constructors can be divided into two types,
  1. Constructor without parameters
  2. Constructor with parameters

#### 7.5. Constructor without parameters

- ✓ If constructor having no parameters, then at least it should contain `self` as one parameter.

##### Syntax

```
class NameOfTheClass:  
    def __init__(self):  
        body of the constructor
```

**Program Name** Creating a constructor  
demo3.py

```
class Employee:  
    def __init__(self):  
        print("constructor is executed")  
  
emp = Employee()
```

**output**  
constructor is executed

## 7.6. Parameterised constructor

- ✓ Based on requirement constructor can contain any number of parameters.

## 7.6. Creating parameterised constructor

- ✓ By default, first parameter should be `self` to constructor.
- ✓ Constructor can contain more parameters along with `self`
- ✓ If constructor having more parameters, then the first parameter should be `self` and remaining parameters will be next.

### Syntax

```
class NameOfTheClass:
    def __init__(self, parameter1, parameter2):
        body of the constructor
```

### Note: One parameterised constructor

**Program Name** One parameterised constructor  
demo6.py

```
class Employee:
    def __init__(self, number):
        self.number= number
        print("Employee id is: ", self.number)

    e1 = Employee(1)
    e2 = Employee(2)
    e3 = Employee(3)
```

**output**

```
Employee id is: 1
Employee id is: 2
Employee id is: 3
```

**Note: One parameterised constructor**

- ✓ If constructor having one parameter, then during object creation we need to pass one value.

**Can i write a constructor and an instance method in a single program?**

- ✓ Yes we can write constructor and instance method both in single program.
- ✓ Here constructor purpose is to initialize instance variables, and method purpose is to perform operations.

**Two parameterised constructor**

**Program Name** One parameterised constructor and instance method  
demo7.py

```
class Employee:
    def __init__(self, number):
        self.number = number

    def display(self):
        print("Employee id is:", self.number)

e1 = Employee(1)
e2 = Employee(2)
e3 = Employee(3)

e1.display()
e2.display()
e3.display()
```

**output**

```
Employee id is: 1
Employee id is: 2
Employee id is: 3
```

## Data Science – Python Object Oriented Programming

---

### Note: Access instance variable in instance method

- ✓ Inside instance method we can access instance variables by using self.

### Two parameterised constructor

**Program Name** Two parameterised constructor and instance method demo8.py

```
class Employee:
    def __init__(self, number, name):
        self.number = number
        self.name = name

    def display(self):
        print("Hello my id is :", self.number)
        print("My name is :", self.name)
```

```
e1=Employee(1, 'Daniel')
e1.display()
```

```
e2=Employee(2, 'Arjun')
e2.display()
```

### Output

```
Hello my id is: 1
My name is: Daniel
```

```
Hello my id is: 2
My name is: Arjun
```

### Note: Two parameterised constructor

- ✓ If constructor having two parameters, then during object creation we need to pass two values

**Three parameterised constructor**

**Program Name** Three parameterised constructor and instance method  
demo9.py

```
class Employee:  
    def __init__(self, number, name, age):  
        self.number = number  
        self.name = name  
        self.age = age  
  
    def display(self):  
        print("Hello my id is :", self.number)  
        print("My name is :", self.name)  
        print("My age is sweet :", self.age)  
  
e1=Employee(1, 'Daniel', 16)  
e1.display()  
  
e2=Employee(2, 'Arjun', 17)  
e2.display()  
  
e3=Employee(3, 'Prasad', 18)  
e3.display()
```

**Output**

Hello my id is: 1  
My name is: Daniel  
My age is sweet: 16

Hello my id is: 2  
My name is: Arjun  
My age is sweet: 17

Hello my id is :3  
My name is: Prasad  
My age is sweet: 18

**Note: Three parameterised constructor**

- ✓ If constructor having three parameters, then during object creation we need to pass three values.

**8. Difference between method and constructor**

<b>Method</b>	<b>Constructor</b>
✓ Methods are used to perform operations or actions	✓ Constructors are used to initialize the instance variables.
✓ Method name can be any name.	✓ Constructor name should be <code>__init__(self)</code>
✓ Methods we should call explicitly to execute	✓ Constructor automatically executed at the time of object creation.

## 9. Instance variables:

### 9.1. What is instance variable?

- ✓ If the value of a variable is changing from object to object such type of variables is called as instance variables.

### 9.2. Separate copy instance variable for every object

- ✓ For every object a separate copy of instance variables will be created.

**Program Name** Instance variables  
demo10.py

```
class Student:
    def __init__(self, name, number):
        self.name=name
        self.number=number

    s1 = Student('Daniel', 101)
    s2 = Student('Prasad', 102)

    print("Studen1 info:")
    print("Name: ", s1.name)
    print("Id : ", s1.number)

    print("Studen2 info:")
    print("Name: ", s2.name)
    print("Id : ", s2.number)
```

### Output

Studen1 info:

Name: Daniel

Id: 101

Studen2 info:

Name: Prasad

Id: 102

### 9.3. Declaring & accessing instance variables

- ✓ We can declare instance variables inside constructor
- ✓ We can access instance variables by using object name

**Program Name** Initializing instance variables inside Constructor  
demo11.py

```
class Employee:  
    def __init__(self):  
        self.eno = 10  
        self.ename = "Daniel"  
        self.esal = 10000  
  
    emp = Employee()  
  
    print("Employee number:", emp.eno)  
    print("Employee name:", emp.ename)  
    print("Employee salary:", emp.esal)
```

**output**  
Employee number: 10  
Employee name : Daniel  
Employee salary : 10000

## 10. Instance methods

- ✓ Instance methods are methods which act upon the instance variables of the class.
- ✓ Instance methods are bound with instances or objects, that's why called as instance methods.
- ✓ The first parameter for instance methods is **self** variable.
- ✓ Along with **self** variable it can contains other variables as well.

<b>Program Name</b>	Instance methods demo13.py
<pre>class Demo:     def __init__(self, a):         self.a=a      def m(self):         print(self.a)  d=Demo(10) d.m()</pre>	

<b>Output</b>	10
---------------	----

### 11. self pre-defined variable

- ✓ self is a predefined variable in python, this variable belongs to current class object.
  - self variable we can use to create below things,
    - Constructor
    - Instance variable
    - Instance methods
- ✓ Constructor
  - By using self, we can initialize the instance variables inside constructor `__init__(self)`
- ✓ Instance variable
  - By using self, we can declare and access instance variables,
- ✓ Instance methods
  - By using self, we can create instance methods.

## Contents

<b>1. Pandas.....</b>	2
<b>2. Main usage .....</b>	2
<b>3. Creator.....</b>	2
<b>4. Data Analysis .....</b>	2
<b>5. Key Features .....</b>	3
<b>6. Open source.....</b>	3
<b>7. Environment setup/pandas installation.....</b>	3
<b>8. pip command in python .....</b>	3
<b>9. Check installed python library.....</b>	4
<b>10. Pandas Data Structures.....</b>	4
10.1. Series.....	5
10.2. DataFrame.....	6
10.3. Panel.....	7
<b>11. Importing pandas .....</b>	9
<b>12. Aliasing or renaming pandas.....</b>	9
12.1. Why aliasing Boss?.....	9
12.2. Famous Alias name to pandas .....	9
12.3. Specialty of alias name as pd .....	10

### 1. PANDAS - INTRODUCTION

#### 1. Pandas

- ✓ Pandas is an open-source Python Library.
- ✓ Pandas have powerful data structures.

#### 2. Main usage

- ✓ Main usage of pandas is analysing the data.
- ✓ By using pandas we can do below things easily,
  - Data loading
  - Data preparation
  - Data manipulation
  - Data analysis & etc

#### 3. Creator

- ✓ Pandas created by Wes McKinney in the year of 2008

#### 4. Data Analysis

- ✓ It is a process of cleaning, transforming, and modeling data to discover useful information for business decision-making.
- ✓ The purpose of Data Analysis is to extract useful information from data and taking the decision based upon the data analysis.
- ✓ Data Analysis became very easy after pandas was introduced to this world.

### 5. Key Features

- ✓ It's very easy to load different file formats of data.
- ✓ Handling missing data.
- ✓ Reshaping the data.
- ✓ Grouping the data
- ✓ Indexing and label-based slicing.
- ✓ Easily insert and delete columns from data
- ✓ Operations like aggregation and transformations.
- ✓ High performance merging and joining of data and many of others boss.

### 6. Open source

- ✓ Pandas is an open source means it's free.

### 7. Environment setup/pandas installation

- ✓ Open command prompt and execute below command

```
pip install pandas
```

### 8. pip command in python

- ✓ pip stands for **python installer package**
- ✓ Pip is a package management system.
- ✓ It is used to install and manage software packages.
  - pip install package\_name
  - pip install pandas

## Data Science – Pandas Introduction

### 9. Check installed python library.

- ✓ We can check installed python library by using below command.

```
pip show pandas
```

### 10. Pandas Data Structures

- ✓ One of the important features in pandas is data structures.
- ✓ There are mainly 3 types of data structures in pandas

Data Structure	Dimentionality	
1. Series	1D	Column
2. DataFrame	2D	Rows & Columns
3. Panel	3D	Group of Dataframes

## Data Science – Pandas Introduction

### 10.1. Series

- ✓ A Series is similar to a single column of data.

#### Series

apples	
0	3
1	2
2	0
3	1

#### Series

oranges	
0	0
1	3
2	7
3	2

## 10.2. DataFrame

- ✓ A DataFrame is similar to a table which contains rows and columns of data.

### DataFrame

	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

Organization   Salesperson Name   Sales

0	Google	Sam	200
1	Google	Charlie	120
2	Salesforce	Ralph	125
3	Salesforce	Emily	250
4	Adobe	Rosalynn	150
5	Adobe	Chelsea	500

## Data Science – Pandas Introduction

### 10.3. Panel

- ✓ A Panel is a group of DataFrames of data.

The diagram illustrates a Panel as a collection of DataFrames. It shows three nested rectangular boxes, each representing a DataFrame. The innermost box contains four rows of data with columns 'age', 'name', and 'teacher'. The middle box contains three rows of data with the same columns. The outermost box contains two rows of data with the same columns. This visual representation indicates that a Panel contains multiple DataFrames, each with its own set of rows and columns.

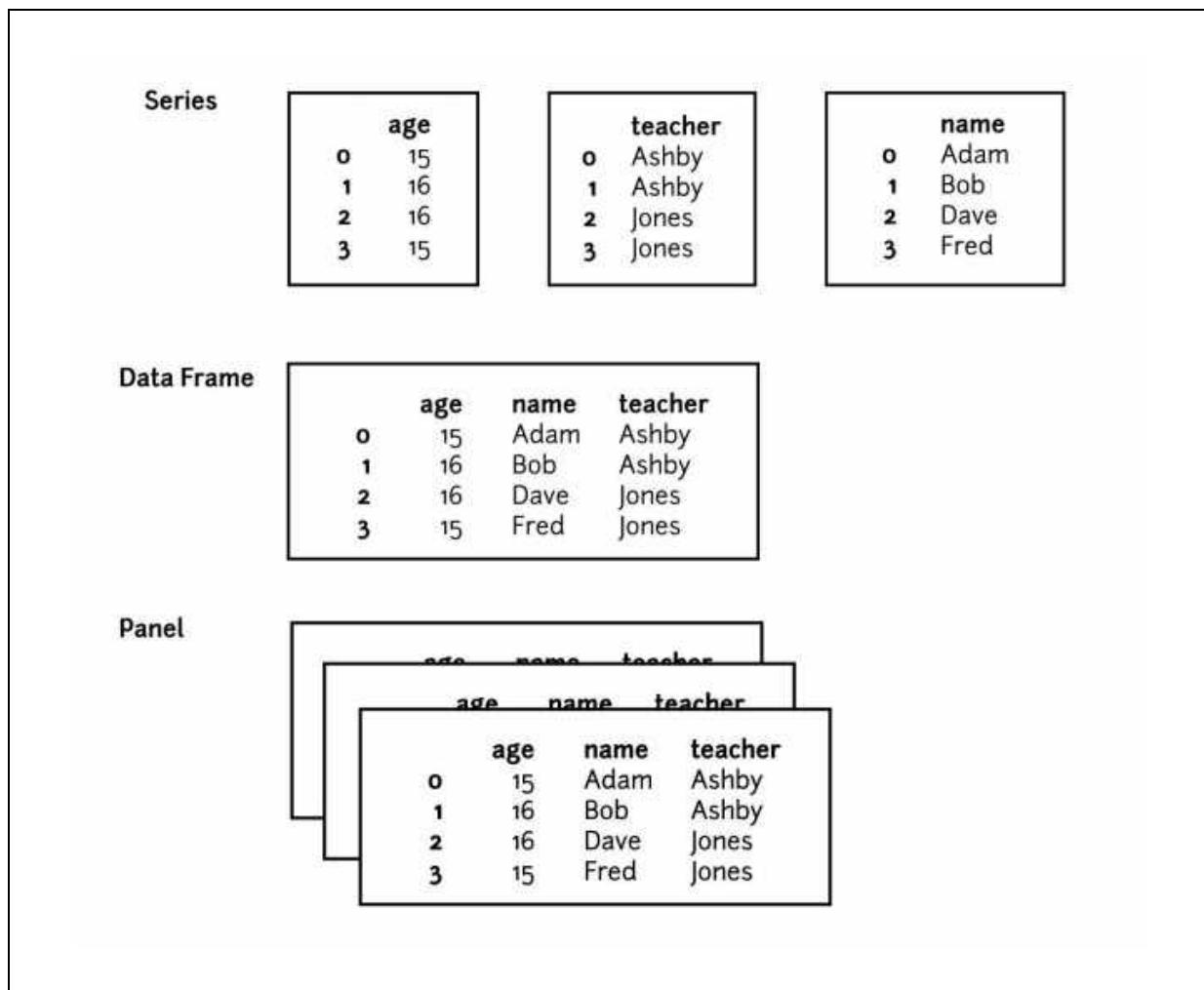
	age	name	teacher
0	15	Adam	Ashby
1	16	Bob	Ashby
2	16	Dave	Jones
3	15	Fred	Jones

### Mostly used

- ✓ The most widely used data structures are the Series and the DataFrame.
- ✓ Series and DataFrames deal with array data and tabular data respectively

## Data Science – Pandas Introduction

---



### Relationship...

- ✓ Series having single column
- ✓ DataFrame can have multiple Series
- ✓ Panel has multiple DataFrames

## 11. Importing pandas

- ✓ We can import pandas package by using `import` keyword

Program Name      importing pandas  
demo1.py

```
import pandas  
  
print("pandas imported successfully")
```

Output

```
pandas imported successfully
```

## 12. Aliasing or renaming pandas

- ✓ We can alias or rename pandas name.
- ✓ As per python syntax we need to use `as` keyword to alias pandas name

### 12.1. Why aliasing Boss?

- ✓ Let me give one best example.
- ✓ My name is: **Kasagani N Daniel**, you can call by Daniel instead of full name for simplicity

### 12.2. Famous Alias name to pandas

- ✓ We can give alias name to pandas.
- ✓ Note this name can be any name but the famous alias name is `pd`

**Program Name** aliasing pandas name as pd  
demo2.py

```
import pandas as pd

print("pandas imported successfully")
print("pandas renamed to pd")
print("Now onwards we can use pd instead of pandas name")
```

**Output**

```
pandas imported successfully
pandas renamed to pd
Now onwards we can use pd instead of pandas name
```

### 12.3. Specialty of alias name as pd

- ✓ Here in our example we have given alias name as pd.
- ✓ We can provide any name as alias name.
- ✓ Real time developers habituated to give alias name as pd.
- ✓ So, its **universal** alias name

## Data Science – Pandas - Series

---

### Contents

<b>1. Series.....</b>	2
<b>2. Creating Series.....</b>	2
2.1. Empty Series object.....	3
2.2. Creating Series by using list .....	4
2.3. Create a Series from array .....	8
2.4. Creating Series by column from DataFrame.....	12
<b>3. Index in Series .....</b>	13
3.1. What is index?.....	13
3.2. Index default value.....	13
<b>4. Accessing values in Series .....</b>	18

**2. PANDAS – SERIES****1. Series**

- ✓ The Pandas Series is a one-dimensional labeled array.
- ✓ Series can store same and different types of the data.
- ✓ Series stores data in sequential order.
- ✓ Series is like a, one column information.

**Series is a pre-defined class**

- ✓ Technically speaking **Series** is a pre-defined class in pandas library.

**2. Creating Series**

- ✓ We can create Series in different ways,
  - Empty series
  - By using list
  - By using an array
  - By accessing single column from DataFrame.

## Data Science – Pandas - Series

### 2.1. Empty Series object

- ✓ We need to create object to Series pre-defined class.

**Program Name** creating empty Series object  
demo1.py

```
import pandas as pd

s = pd.Series()
print(s)
print(type(s))
```

**Output**  
Series([], dtype: float64)  
<class 'pandas.core.series.Series'>

## Data Science – Pandas - Series

### 2.2. Creating Series by using list

- ✓ We can create Series by using list.

**Program Name** creating Series object using list  
demo2.py

```
import pandas as pd

m = [56, 45, 35, 41, 44, 60]
s = pd.Series(m)
print(s)
```

**Output**

```
0    56
1    45
2    35
3    41
4    44
5    60
dtype: int64
```

## Data Science – Pandas - Series

**Program Name** creating Series object using list, assigning a name  
demo3.py

```
import pandas as pd

m = [56, 45, 35, 41, 44, 60]
s = pd.Series(m, name = "marks")
print(s)
```

**Output**

```
0    56
1    45
2    35
3    41
4    44
5    60
Name: marks, dtype: int64
```

## Data Science – Pandas - Series

**Program Name** creating Series object using list, assigning a name  
demo4.py

```
import pandas as pd

n = ["Prasad", "Daniel", "Samuel", "Jeswanth"]
s = pd.Series(n, name = "students")
print(s)
```

**Output**

```
0      Prasad
1      Daniel
2      Samuel
3      Jeswanth
Name: students, dtype: object
```

## Data Science – Pandas - Series

**Program Name** creating Series object by using range and list.  
demo5.py

```
import pandas as pd

r = range(100)
a = list(r)
s = pd.Series(a)

print(s)
```

**Output**

```
0      0
1      1
2      2
3      3
4      4
       ..
95     95
96     96
97     97
98     98
99     99
Length: 100, dtype: int64
```

## Data Science – Pandas - Series

### 2.3. Create a Series from array

- ✓ We can pass array as an argument to the Series.
- ✓ By default, index is assigned to every element.

Program creating ndarray  
Name demo6.py

```
import pandas as pd
import numpy as np

values = [10, 20, 30, 40]
data = np.array(values)

print(data)
print(type(data))
```

Output

```
[10 20 30 40]
<class 'numpy.ndarray'>
```

## Data Science – Pandas - Series

**Program Name** creating ndarray and passing argument to the Series  
demo7.py

```
import pandas as pd
import numpy as np

values = [10, 20, 30, 40]
data = np.array(values)

s = pd.Series(data)
print(s)
```

**Output**

```
0    10
1    20
2    30
3    40
dtype: int32
```

## Data Science – Pandas - Series

**Program Name** Creating Series using ndarray  
demo8.py

```
import pandas as pd
import numpy as np

values = ['a', 'b', 'c', 'd']
data = np.array(values)

s = pd.Series(data)
print(s)
```

**Output**

```
0    a
1    b
2    c
3    d
dtype: object
```

## Data Science – Pandas - Series

**Program Name** Creating Series using ndarray  
demo9.py

```
import pandas as pd
import numpy as np

values = ['Vinay', 'Daniel', 'Veeru', 'Arjun']
data = np.array(values)
s = pd.Series(data)
print(s)
```

**Output**

```
0      Vinay
1      Daniel
2      Veeru
3      Arjun
dtype: object
```

### 2.4. Creating Series by column from DataFrame

- ✓ If we select single column from DataFrame then it returns Series object.
- ✓ This point we will learn during DataFrame chapter, thanks for understanding.

## Data Science – Pandas - Series

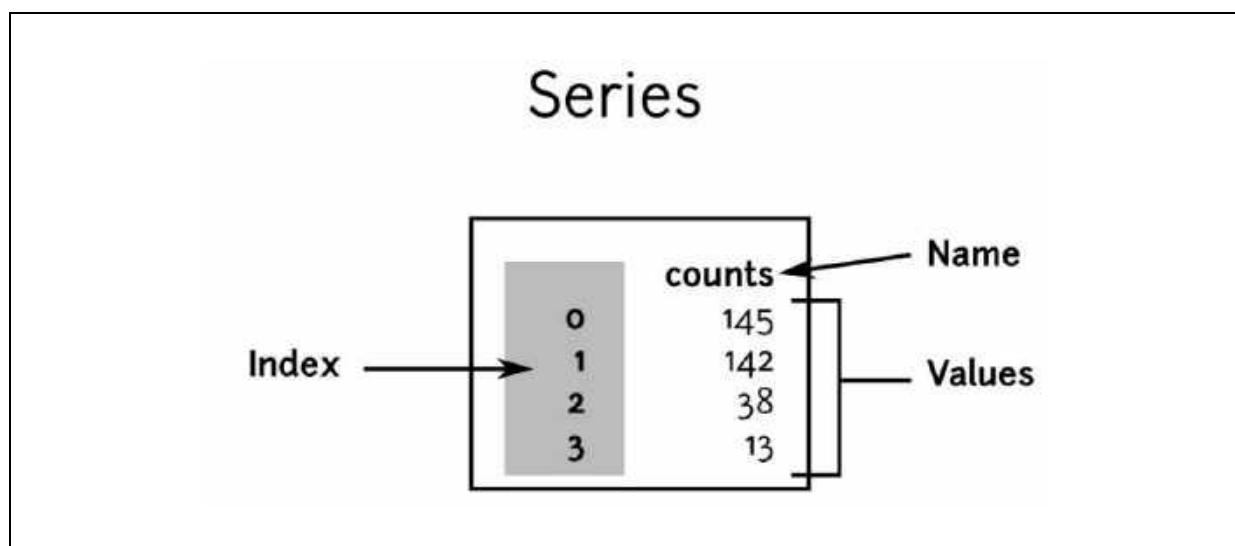
### 3. Index in Series

#### 3.1. What is index?

- ✓ Index means, the position of value where it stores.
- ✓ The index is a core feature in pandas.
- ✓ By default, index is assigned to every value.
- ✓ From the output, the left most column is the index column.
- ✓ The generic name for an index is an axis.

#### 3.2. Index default value

- ✓ The default values for an index are integers.
- ✓ The index starts from 0, 1, 2, 3, etc.
- ✓ Based on requirement we can customise this index
- ✓ These are called as axis labels



## Data Science – Pandas - Series

Program Name      Creating Series  
demo10.py

```
import pandas as pd

v = [145, 142, 38, 13]
s = pd.Series(v)
print(s)
```

Output

```
0    145
1    142
2     38
3     13
dtype: int64
```

## Data Science – Pandas - Series

**Program Name** Creating Series object and giving name  
demo11.py

```
import pandas as pd

v = [145, 142, 38, 13]
s = pd.Series(v, name = 'counts')
print(s)
```

**Output**

```
0    145
1    142
2     38
3     13
Name: counts, dtype: int64
```

## Data Science – Pandas - Series

**Program Name** Creating Series object and giving name and index  
demo12.py

```
import pandas as pd

v = [145, 142, 38, 13]
i = [10, 20, 30, 40]

s = pd.Series(v, name = 'counts', index = i )
print(s)
```

**Output**

```
10    145
20    142
30     38
40     13
Name: counts, dtype: int64
```

## Data Science – Pandas - Series

**Program Name** Creating Series object and giving name and index  
demo13.py

```
import pandas as pd

prices = [1000, 2000, 3000, 4000]
products = ["Nokia", "Samsung", "Oppo", "iPhone 6"]

s = pd.Series(prices, name = 'mobiles', index = products )
print(s)
```

**Output**

```
Nokia      1000
Samsung    2000
Oppo       3000
iPhone 6   4000
Name: mobiles, dtype: int64
```

## Data Science – Pandas - Series

### 4. Accessing values in Series

- ✓ We can access series values by using index

**Program Name** Creating Series and accessing values  
demo14.py

```
import pandas as pd

v = [56, 45, 35, 41, 44, 60]
s = pd.Series(v, name = "marks")

print(s)
print()
print(s[0])
print(s[1])
```

#### Output

```
0    56
1    45
2    35
3    41
4    44
5    60
Name: marks, dtype: int64

56
45
```

## Data Science – Pandas - Series

**Program Name** Creating Series and accessing values  
demo15.py

```
import pandas as pd

prices = [1000, 2000, 3000, 4000]
products = ["Nokia", "Samsung", "Oppo", "iPhone 6"]

s = pd.Series(prices, name = 'mobiles', index = products )

print(s)
print()
print(s["Nokia"])
print(s["Samsung"])
```

**Output**

```
Nokia      1000
Samsung    2000
Oppo       3000
iPhone 6   4000
Name: mobiles, dtype: int64

1000
2000
```

## Data Science – Pandas – NaN Value

---

### Contents

<b>1. NaN .....</b>	2
<b>2. None, NaN, nan and null.....</b>	2
<b>3. While loading any file.....</b>	3

### 3. PANDAS – NaN Value

#### 1. NaN Value

- ✓ The full form of NaN is **Not a Number**
- ✓ The purpose of NaN is, to represent the missing values in data.
- ✓ The data type of NaN is **float**.
- ✓ During data analysis we need to handle these NaN values.
- ✓ We will learn **more** about NaN in **12<sup>th</sup>** chapter which is handling missing values chapter.

#### 2. None, NaN, nan and null

- ✓ None, NaN, nan, and null are synonyms.
- ✓ These all are referring to empty or missing data found in a Series, DataFrame.

**Program Name** Creating Series with NaN values  
demo1.py

```
import pandas as pd
import numpy as np

marks = [36, 70, np.nan, 60]
s = pd.Series(marks)
print(s)
```

#### Output

```
0    36.0
1    70.0
2    NaN
3    60.0
dtype: float64
```

## Data Science – Pandas – NaN Value

**Program Name** Creating Series with NaN values  
demo2.py

```
import pandas as pd
import numpy as np

names = ["Daniel", "Ranjan", "Swathi", np.nan]
s = pd.Series(names)
print(s)
```

**Output**

```
0      Daniel
1      Ranjan
2      Swathi
3      NaN
dtype: object
```

### 3. While loading any file

- ✓ While loading any csv/excel/json file, if columns having missing values, then pandas consider those values as **NaN**.
- ✓ We will learn same point during loading files.

## Data Science – Pandas – Series - Attributes

---

### Contents

<b>1. Series Attributes.....</b>	2
1.1. values .....	2
1.2. index.....	4
1.3. dtypes.....	7
1.4. size .....	10

## 4. PANDAS – SERIES - ATTRIBUTES

### 1. Series Attributes

- ✓ Series is a predefined class.
- ✓ Series having different attributes.
- ✓ These attributes return information about the object.

#### 1.1. values

- ✓ values is predefined attribute in Series class.
- ✓ We can access values attribute by using series object.
- ✓ This attribute returns the group of values as an array.

**Program Name** Accessing values attribute.  
demo1.py

```
import pandas as pd

marks = [56, 45, 35, 41, 44, 60]
s = pd.Series(marks)

print(s)
print(s.values)
```

#### Output

```
0    56
1    45
2    35
3    41
4    44
5    60
dtype: int64
[56 45 35 41 44 60]
```

## Data Science – Pandas – Series - Attributes

**Program Name** Accessing values attribute.  
demo2.py

```
import pandas as pd

names = ["Bharath", "Daniel", "Nireekshan"]
s = pd.Series(names)

print(s)
print(s.values)
```

**Output**

```
0      Bharath
1      Daniel
2    Nireekshan
dtype: object
['Bharath' 'Daniel' 'Nireekshan']
```

## Data Science – Pandas – Series - Attributes

### 1.2. index

- ✓ index is predefined attribute in Series class.
- ✓ We can access index attribute by using series object.
- ✓ This attribute returns the index range like, RangeIndex(start=0, stop=6, step=1)

**Program Name** Accessing index attribute.

**demo3.py**

```
import pandas as pd

marks = [56, 45, 35, 41, 44, 60]
s = pd.Series(marks)

print(s)
print(s.index)
```

**Output**

```
0    56
1    45
2    35
3    41
4    44
5    60
dtype: int64
RangeIndex(start=0, stop=6, step=1)
```

## Data Science – Pandas – Series - Attributes

**Program Name** Accessing index attribute.  
demo4.py

```
import pandas as pd

marks = [56, 45, 35, 41, 44, 60]
i = [11, 12, 13, 14, 15, 16]
s = pd.Series(marks, index = i)

print(s)
print(s.index)
```

**Output**

```
11    56
12    45
13    35
14    41
15    44
16    60
dtype: int64
Int64Index([11, 12, 13, 14, 15, 16], dtype='int64')
```

## Data Science – Pandas – Series - Attributes

**Program Name** Accessing index attribute.  
demo5.py

```
import pandas as pd

marks = [56, 45, 35, 41, 44, 60]
i = ['a', 'b', 'c', 'd', 'e', 'f']
s = pd.Series(marks, index = i)

print(s)
print(s.index)
```

**Output**

```
a    56
b    45
c    35
d    41
e    44
f    60
dtype: int64
Index(['a', 'b', 'c', 'd', 'e', 'f'], dtype='object')
```

## Data Science – Pandas – Series - Attributes

### 1.3. dtypes

- ✓ dtypes is predefined attribute in Series class.
- ✓ We can access dtypes attribute by using series object.
- ✓ This attribute returns data type of series column.

**Program Name** Accessing dtypes attribute.  
demo6.py

```
import pandas as pd

marks = [56, 45, 35, 41, 44, 60]
s = pd.Series(marks)

print(s)
print(s.dtype)
```

#### Output

```
0    56
1    45
2    35
3    41
4    44
5    60
dtype: int64
int64
```

## Data Science – Pandas – Series - Attributes

**Program Name** Accessing dtypes attribute.  
demo7.py

```
import pandas as pd

salaries = [1000.23, 1100.45, 8889.7, 999.87]
s = pd.Series(salaries)

print(s)
print(s.dtypes)
```

**Output**

```
0    1000.23
1    1100.45
2    8889.70
3    999.87
dtype: float64
float64
```

## Data Science – Pandas – Series - Attributes

**Program Name** Accessing dtypes attribute.  
demo8.py

```
import pandas as pd

names = ["Daniel", "Abhinav", "Dinesh", "Akshitha"]
s = pd.Series(names)

print(s)
print(s.dtypes)
```

**Output**

```
0      Daniel
1      Abhinav
2      Dinesh
3      Akshitha
dtype: object
object
```

## Data Science – Pandas – Series - Attributes

### 1.4. size

- ✓ size is predefined attribute in Series class.
- ✓ We can access size attribute by using series object.
- ✓ This attribute returns number of values in series.

**Program Name** Accessing size attribute.  
demo9.py

```
import pandas as pd

marks = [56, 45, 35, 41, 44, 60]
s = pd.Series(marks)

print(s)
print(s.size)
```

### Output

```
0    56
1    45
2    35
3    41
4    44
5    60
dtype: int64
6
```

## Data Science – Pandas - Series- Methods

---

### Contents

<b>1. Series Methods.....</b>	<b>2</b>
1.1. head() .....	2
1.2. tail().....	4
1.3. sum() .....	5
1.4. count().....	7
1.5. mean() .....	8
1.6. describe() .....	9
1.7. unique().....	11
1.8. nunique().....	12

## 5. PANDAS – SERIES – METHODS

### 1. Series Methods

- ✓ Series is a predefined class.
- ✓ Series class having different methods
- ✓ These methods perform operations on Series of values.

#### 1.1. head()

- ✓ head() is predefined method in Series class.
- ✓ We can access head() method by using series object.
- ✓ This method returns first five values from the series

**Program Name** Accessing first five values from series  
demo1.py

```
import pandas as pd

marks = [56, 45, 35, 41, 60, 57, 56, 56]
s = pd.Series(marks)

print(s)
print(s.head())
```

**Output**

## Data Science – Pandas - Series- Methods

---

```
0    56
1    45
2    35
3    41
4    60
5    57
6    56
7    56
dtype: int64
0    56
1    45
2    35
3    41
4    60
dtype: int64
```

### 1.2. tail()

- ✓ tail() is predefined method in Series class.
- ✓ We can access tail() method by using series object.
- ✓ This method returns last five values from the series

**Program Name** Accessing last five values from series  
demo2.py

```
import pandas as pd

marks = [56, 45, 35, 41, 60, 57, 56, 56]
s = pd.Series(marks)

print(s)
print(s.tail())
```

#### Output

```
0    56
1    45
2    35
3    41
4    60
5    57
6    56
7    56
dtype: int64
3    41
4    60
5    57
6    56
7    56
dtype: int64
```

### 1.3. sum()

- ✓ sum() is predefined method in Series class.
- ✓ We can access sum() method by using series object.
- ✓ This method returns the sum of all values.

**Program Name** Get the sum of series of values.  
demo3.py

```
import pandas as pd

sales = [56, 45, 35, 41, 44, 60]
s = pd.Series(sales)

print(s)
print(s.sum())
```

#### Output

```
0    56
1    45
2    35
3    41
4    44
5    60
dtype: int64
281
```

## Data Science – Pandas - Series- Methods

**Program Name** Get the sum of series of values.  
demo4.py

```
import pandas as pd
import numpy as np

marks = [56, 45, 35, 41, np.nan, 60, np.nan]
s = pd.Series(marks)

print(s)
print(s.sum())
```

### Output

```
0    56.0
1    45.0
2    35.0
3    41.0
4    NaN
5    60.0
6    NaN
dtype: float64
237.0
```

### 1.4. count()

- ✓ count() is predefined method in Series class.
- ✓ We can access count() method by using series object.
- ✓ This method returns the number of non-NAN/null values.

**Program Name** Get the number of non-NaN values  
demo5.py

```
import pandas as pd
import numpy as np

marks = [56, 45, 35, 41, np.nan, 60, np.nan]
s = pd.Series(marks)

print(s)
print(s.count())
```

### Output

```
0    56.0
1    45.0
2    35.0
3    41.0
4    NaN
5    60.0
6    NaN
dtype: float64
5
```

### 1.5. mean()

- ✓ mean() is predefined method in Series class.
- ✓ We can access mean() method by using series object.
- ✓ This method returns the mean of series of values.

**Program Name** Get the mean of series values  
demo6.py

```
import pandas as pd

sales = [10, 20, 30, 40, 50]
s = pd.Series(sales)

print(s)
print(s.mean())
```

#### Output

```
0    10
1    20
2    30
3    40
4    50
dtype: int64
30.0
```

## Data Science – Pandas - Series- Methods

### 1.6. describe()

- ✓ describe() is predefined method in Series class.
- ✓ We can access describe() method by using series object.
- ✓ This method returns the below values,
  - count
  - mean
  - std
  - min
  - 25%
  - 50%
  - 75%
  - max

**Program Name**      describe() method  
demo7.py

```
import pandas as pd

sales = [56, 45, 35, 41, 60]
s = pd.Series(sales)

print(s)
print(s.describe())
```

**Output**

## Data Science – Pandas - Series- Methods

```
0      56  
1      45  
2      35  
3      41  
4      60  
dtype: int64  
count      5.000000  
mean      47.400000  
std       10.406729  
min      35.000000  
25%      41.000000  
50%      45.000000  
75%      56.000000  
max      60.000000  
dtype: float64
```

### 1.7. unique()

- ✓ unique() is predefined method in Series class.
- ✓ We can access unique() method by using series object.
- ✓ This method returns unique values from the series.

**Program Name** Get unique values from the series  
demo8.py

```
import pandas as pd

marks = [56, 45, 35, 41, 60, 57, 56, 56]
s = pd.Series(marks)

print(s)
print(s.unique())
```

#### Output

```
0    56
1    45
2    35
3    41
4    60
5    57
6    56
7    56
dtype: int64
[56 45 35 41 60 57]
```

### 1.8. nunique()

- ✓ nunique() is predefined method in Series class.
- ✓ We can access nunique() method by using series object.
- ✓ This method returns number of unique values from the series.

**Program Name** Get the number of unique values from the series  
demo9.py

```
import pandas as pd

marks = [56, 45, 35, 41, 60, 57, 56, 56]
s = pd.Series(marks)

print(s)
print(s.nunique())
```

#### Output

```
0    56
1    45
2    35
3    41
4    60
5    57
6    56
7    56
dtype: int64
6
```

## Contents

<b>1. DataFrame .....</b>	2
1.1. DataFrame is a pre-defined class .....	2
<b>2. Create DataFrame .....</b>	4
2.1. Create an Empty DataFrame .....	5
2.2. Create a DataFrame by using list .....	6
2.3. Creating a DataFrame by using list of lists .....	9
2.4. Creating a DataFrame by using dictionary .....	12
2.5. Creating DataFrame by loading the files.....	16

## 6. PANDAS – DATAFRAME – INTRODUCTION

### 1. DataFrame

- ✓ A Data frame is a two-dimensional data structure.
- ✓ Data frame is just like a table.
- ✓ Data frame contains rows and columns.

#### 1.1. DataFrame is a pre-defined class

- ✓ DataFrame is a pre-defined class in pandas library.

#### Example

Emp_No	Name	Salary
101	Ranjan	10000
102	Akshay	20000
103	Daniel	30000
104	Veeru	40000

**Columns and Rows are**

✓ Columns are

- First column name is : Emp\_No
- Second column name is : Name
- Third column name is : Salary

✓ Rows are

- First row data is : 101 Abhi 10000
- Second row data is : 102 Akshay 20000
- Third row data is : 103 Daniel 10000
- Forth row data is : 104 Veeru 10000

### 2. Create DataFrame

- ✓ DataFrame is a predefined class in pandas.
- ✓ We can create DataFrame in different ways like below,
  - Empty DataFrame
  - By using single list
  - By using nested list
  - By using dictionary
  - with another DataFrame
  - Loading files(real time approach)

#### Generally

- ✓ In real time when we load existing file then it returns DataFrame

## 2.1. Create an Empty DataFrame

- ✓ We can create an empty DataFrame

**Program Name** Creating empty DataFrame  
demo1.py

```
import pandas as pd

df = pd.DataFrame()
print(df)
print(type(df))
```

**Output**

```
Empty DataFrame
Columns: []
Index: []
```

```
<class 'pandas.core.frame.DataFrame'>
```

## Data Science - Pandas - DataFrame Introduction

### 2.2. Create a DataFrame by using list

- ✓ We can create DataFrame by using single list.
  - If we are using single list then it's a single column DataFrame
  - If we are using list of lists(nested lists) then it's multiple columns DataFrame

**Program Name** Creating DataFrame by using single list  
demo2.py

```
import pandas as pd

a = [10, 20, 30, 40, 50, 60, 70, 80, 90]
df = pd.DataFrame(a)

print(df)
print(type(df))
```

**Output**

```
      0
0    10
1    20
2    30
3    40
4    50
5    60
6    70
7    80
8    90
<class 'pandas.core.frame.DataFrame'>
```

#### Note

- ✓ From the output, DataFrame created with single column.
- ✓ Here column name is Zero, we can customise this as well.

## Data Science - Pandas - DataFrame Introduction

### Note on index

- ✓ If no index is passed, then by default, index will be range(n), where n is the array length

**Program Name** Creating DataFrame by using single list  
demo3.py

```
import pandas as pd

names = ["Ranjan", "Sagar", "Daniel", "Prasad", "Kumari",
         "Pravallika", "Arjun", "Akshay"]
df = pd.DataFrame(names)

print(df)
```

### Output

```
          0
0      Ranjan
1      Sagar
2     Daniel
3     Prasad
4     Kumari
5  Pravallika
6     Arjun
7     Akshay
```

## Data Science - Pandas - DataFrame Introduction

**Program Name** Creating single column DataFrame and checking length  
demo4.py

```
import pandas as pd

names = ["Ranjan", "Sagar", "Daniel", "Prasad", "Kumari",
"Pravallika", "Arjun", "Akshay"]
df = pd.DataFrame(names)

print(df)
print()
print("The length is:", len(df))
```

**Output**

```
          0
0      Ranjan
1      Sagar
2     Daniel
3     Prasad
4     Kumari
5  Pravallika
6     Arjun
7     Akshay

The length is: 8
```

### 2.3. Creating a DataFrame by using list of lists

- ✓ We can create DataFrame with list of lists (nested list).
- ✓ If we are using list of lists then it create a DataFrame with multiple columns.

## Data Science - Pandas - DataFrame Introduction

**Program Name** Creating DataFrame by using list of lists  
demo5.py

```
import pandas as pd

details = [
    ["Ranjan", 11],
    ["Sagar", 12],
    ["Daniel", 13],
    ["Prasad", 14],
    ["Kumari", 15],
    ["Pravallika", 16],
    ["Arjun", 17],
    ["Akshay", 18]
]

df = pd.DataFrame(details)

print(df)
```

### Output

	0	1
0	Ranjan	11
1	Sagar	12
2	Daniel	13
3	Prasad	14
4	Kumari	15
5	Pravallika	16
6	Arjun	17
7	Akshay	18

### Note

- ✓ From the output, DataFrame created with two columns.
- ✓ Here column names are 0 and 1 and we can customise this as well.

## Data Science - Pandas - DataFrame Introduction

---

### 2.3.1. Giving column names to DataFrame

- ✓ We can give column names to DataFrame.

**Program Name** Creating DataFrame and giving names to columns  
demo6.py

```
import pandas as pd

details = [
    ["Sagar", 20, 10000],
    ["Daniel", 16, 20000],
    ["Veeru", 24, 30000],
    ["Raju", 25, 40000],
    ["Kiran", 26, 50000],
    ["Kedar", 27, 60000],
    ["Reena", 28, 70000],
    ["Karthik", 29, 80000],
    ["Satish", 30, 90000]
]

cols = ["Name", "Age", "Salary"]

df = pd.DataFrame(details, columns = cols)

print(df)
```

**Output**

	Name	Age	Salary
0	Sagar	20	10000
1	Daniel	16	20000
2	Veeru	24	30000
3	Raju	25	40000
4	Kiran	26	50000
5	Kedar	27	60000
6	Reena	28	70000
7	Karthik	29	80000
8	Satish	30	90000

## Data Science - Pandas - DataFrame Introduction

---

### 2.4. Creating a DataFrame by using dictionary

- ✓ We can create DataFrame by using dictionary
- ✓ If we are using list of lists then it create a DataFrame with multiple columns.

**Program Name** Creating DataFrame by using dictionary  
demo8.py

```
import pandas as pd

details = {
    "Name": ["Daniel", "Abhi", "Veeru", "Raju", "Kiran",
    "Kedar", "Reena", "Karthik", "Satish"],

    "Age": [20, 21, 23, 24, 25, 26, 27, 28, 29]
}

df = pd.DataFrame(details)

print(df)
```

#### Output

	Name	Age
0	Daniel	20
1	Abhi	21
2	Veeru	23
3	Raju	24
4	Kiran	25
5	Kedar	26
6	Reena	27
7	Karthik	28
8	Satish	29

#### Note

- ✓ In above example Name and age considered as column names

### 2.4.1. We can customize the index values

- ✓ By default index value start from 0
- ✓ We can customise the index values in DataFrame.
- ✓ If index is passed, then the length of the index should equal to the length of the DataFrame.

## Data Science - Pandas - DataFrame Introduction

**Program Name** Creating DataFrame and giving index  
demo9.py

```
import pandas as pd

details = [
    ["Sagar", 20, 10000],
    ["Daniel", 16, 20000],
    ["Veeru", 24, 30000],
    ["Raju", 25, 40000],
    ["Kiran", 26, 50000],
    ["Kedar", 27, 60000],
    ["Reena", 28, 70000],
    ["Karthik", 29, 80000],
    ["Satish", 30, 90000]
]

c = ["Name", "Age", "Salary"]
i = [11, 22, 33, 44, 55, 66, 77, 88, 99]

df = pd.DataFrame(details, columns = c, index = i)

print(df)
```

### Output

	Name	Age	Salary
11	Sagar	20	10000
22	Daniel	16	20000
33	Veeru	24	30000
44	Raju	25	40000
55	Kiran	26	50000
66	Kedar	27	60000
77	Reena	28	70000
88	Karthik	29	80000
99	Satish	30	90000

## Data Science - Pandas - DataFrame Introduction

---

**Program Name** Creating DataFrame and giving index  
demo10.py

```
import pandas as pd

details = [
    ["Sagar", 20, 10000],
    ["Daniel", 16, 20000],
    ["Veeru", 24, 30000],
    ["Raju", 25, 40000],
    ["Kiran", 26, 50000],
    ["Kedar", 27, 60000],
    ["Reena", 28, 70000],
    ["Karthik", 29, 80000],
    ["Satish", 30, 90000]
]

c = ["Name", "Age", "Salary"]
i = ["Row1", "Row2", "Row4", "Row5", "Row6", "Row7", "Row8",
     "Row9", "Row10"]

df = pd.DataFrame(details, columns = c, index = i)

print(df)
```

### Output

	Name	Age	Salary
Row1	Sagar	20	10000
Row2	Daniel	16	20000
Row4	Veeru	24	30000
Row5	Raju	25	40000
Row6	Kiran	26	50000
Row7	Kedar	27	60000
Row8	Reena	28	70000
Row9	Karthik	29	80000
Row10	Satish	30	90000

**2.5. Creating DataFrame by loading the files.**

- ✓ We can create DataFrame by loading files like csv, json etc.
- ✓ This we will learn more on 8<sup>th</sup> chapter.

## Data Science – Pandas – Loading Different Files

---

### Contents

<b>1. Loading files .....</b>	2
<b>2. Loading files .....</b>	2
2.1. csv file.....	3
2.2. json file.....	7
2.3. excel file .....	8
2.4. TSV file.....	10
2.5. Table from webpage .....	11

### 7. PANDAS – LOADING DIFFERENT FILES

#### 1. Loading files

- ✓ We can load different files in pandas.
- ✓ Whenever we are loading the files then pandas return a **DataFrame**.
- ✓ Once after DataFrame is returned then based on requirement we can perform several operations.

#### 2. Loading files

- ✓ Csv file
- ✓ Json file
- ✓ Excel file
- ✓ Tsv file
- ✓ Table from webpage

## Data Science – Pandas – Loading Different Files

---

### 2.1. csv file

- ✓ Csv file stands for comma separated file.
- ✓ A csv file contains data like rows and columns.
- ✓ Whenever we are loading csv file then pandas returns DataFrame.
- ✓ We can load csv file by using **read\_csv("file name with path")** function

#### Syntax

```
pd.read_csv("file name with path")
```

**Program** Loading csv file

**Name** demo1.py

**Input file** sales1.csv

```
import pandas as pd
```

```
df = pd.read_csv("sales1.csv")
print(df)
```

#### Output

	Order ID	Customer Name	Product	Quantity
0	166837	Veeru	34in Ultrawide Monitor	2
1	166838	Tarun	Samsung m10	3
2	166839	Kedar	20in Monitor	1
3	166840	Lavanya	iPhone 11	3
4	166841	Venu	Macbook Pro Laptop	2
..	...	...	...	...
595	167403	Balaji	Macbook Pro Laptop	1
596	167404	Lavanya	ThinkPad Laptop	1
597	167405	Venu	Flatscreen TV	1
598	167406	Siddhu	Samsung m20	2
599	167407	Tarun	LG Washing Machine	1
[600 rows x 4 columns]				

## Data Science – Pandas – Loading Different Files

### Note

- ✓ Above program, sales1.csv file should exist in current directory otherwise we will get an error like **FileNotFoundException**
- ✓ Let's load file from the folder in this case we need to provide file name with folder.

**Program Name** Loading csv file  
**Input file** demo2.py  
If file name not exist then we will get an error

```
import pandas as pd

df = pd.read_csv("sales123.csv")
print(df)
```

### Output

**FileNotFoundException:** No such file or directory: 'sales123.csv'

## Data Science – Pandas – Loading Different Files

**Program Name** Loading csv file from **files** folder  
**Input file** demo3.py  
files\sales1.csv

```
import pandas as pd

df = pd.read_csv('files\sales1.csv')
print(df)
```

### Output

	Order ID	Customer Name	Product	Quantity
0	166837	Veeru	34in Ultrawide Monitor	2
1	166838	Tarun	Samsung m10	3
2	166839	Kedar	20in Monitor	1
3	166840	Lavanya	iPhone 11	3
4	166841	Venu	Macbook Pro Laptop	2
..	...	...	...	...
595	167403	Balaji	Macbook Pro Laptop	1
596	167404	Lavanya	ThinkPad Laptop	1
597	167405	Venu	Flatscreen TV	1
598	167406	Siddhu	Samsung m20	2
599	167407	Tarun	LG Washing Machine	1

[600 rows x 4 columns]

## Data Science – Pandas – Loading Different Files

**Program Name** Loading csv file from D drive  
**Input file** demo4.py  
D:\sales1.csv

```
import pandas as pd

df = pd.read_csv('D:\\sales1.csv')
print(df)
```

### Output

	Order ID	Customer Name	Product	Quantity
0	166837	Veeru	34in Ultrawide Monitor	2
1	166838	Tarun	Samsung m10	3
2	166839	Kedar	20in Monitor	1
3	166840	Lavanya	iPhone 11	3
4	166841	Venu	Macbook Pro Laptop	2
..	...	...	...	...
595	167403	Balaji	Macbook Pro Laptop	1
596	167404	Lavanya	ThinkPad Laptop	1
597	167405	Venu	Flatscreen TV	1
598	167406	Siddhu	Samsung m20	2
599	167407	Tarun	LG Washing Machine	1

[600 rows x 4 columns]

## Data Science – Pandas – Loading Different Files

### 2.2. json file

- ✓ JSON stands for **JavaScript Object Notation**
- ✓ In json file data is like key-value pairs and arrays.
- ✓ It is commonly used for transmitting data in web applications
- ✓ Whenever we are loading json file then pandas returns DataFrame.
- ✓ We can load csv file by using **read\_json("file name with path")** function

#### Syntax

```
pd.read_json("files name with path")
```

**Program Name** Loading json file  
**Input file** demo5.py  
**sales1.json**

```
import pandas as pd

df = pd.read_json("sales1.json")
print(df)
```

#### Output

	order_id	cust_name	product	quantity
0	16278939	Lavanya	ThinkPad Laptop	2
1	16278966	Kedar	Flatscreen TV	1
2	16278993	Jaya Chandra	Macbook Pro Laptop	2
3	16279020	Mallikarjun	iPhone 11	3
4	16279047	Shahid	LG Washing Machine	1
..	...	...	...	...
495	16292304	Sagar	iPhone 7	1
496	16292331	Chaithanya	Samsung m20	1
497	16292358	Siddhu	LG Dryer	2
498	16292385	Siddhu	AA Batteries (4-pack)	1
499	16292412	Sagar	iPhone 11	2

[500 rows x 4 columns]

## Data Science – Pandas – Loading Different Files

### 2.3. excel file

- ✓ Excel is a spreadsheet from Microsoft.
- ✓ It is using mainly to store business applications data
- ✓ Whenever we are loading excel file then pandas returns DataFrame.
- ✓ We can load csv file by using **read\_excel("file name with path")** function

**Run below command**

```
pip install xlrd
```

**Syntax**

```
pd.read_excel("files name with path")
```

## Data Science – Pandas – Loading Different Files

Program Loading excel file

Name demo6.py

Input file sales1.xlsx

```
import pandas as pd

df = pd.read_excel("sales1.xlsx")
print(df)
```

Output

```
   Order ID Custer Name          Product  Quantity
0      166837    Veeru  34in Ultrawide Monitor      2
1      166838    Tarun        Samsung m10       3
2      166839    Kedar  20in Monitor       1
3      166840   Lavanya        iPhone 11       3
4      166841    Venu  Macbook Pro Laptop      2
..        ...
595     167403  Balaji  Macbook Pro Laptop      1
596     167404   Lavanya  ThinkPad Laptop      1
597     167405    Venu    Flatscreen TV       1
598     167406   Siddhu  Samsung m20       2
599     167407    Tarun  LG Washing Machine      1

[600 rows x 4 columns]
```

## Data Science – Pandas – Loading Different Files

---

### 2.4. TSV file

- ✓ TSV stands for **Tab Separated File**
- ✓ Whenever we are loading tsv file then pandas returns DataFrame.
- ✓ We can load csv file by using **read\_table("file name with path")** function

#### Syntax

```
pd.read_table(path and file name)
```

**Program Name** Loading tsv file

**Name** demo7.py

**Input file** sales1.tsv

```
import pandas as pd

df = pd.read_table("sales1.tsv")
print(df)
```

#### Output

	Order ID	Custer Name	Product	Quantity
0	166837	Veeru	34in Ultrawide Monitor	2
1	166838	Tarun	Samsung m10	3
2	166839	Kedar	20in Monitor	1
3	166840	Lavanya	iPhone 11	3
4	166841	Venu	Macbook Pro Laptop	2
..	...	...	...	...
595	167403	Balaji	Macbook Pro Laptop	1
596	167404	Lavanya	ThinkPad Laptop	1
597	167405	Venu	Flatscreen TV	1
598	167406	Siddhu	Samsung m20	2
599	167407	Tarun	LG Washing Machine	1
[600 rows x 4 columns]				

## Data Science – Pandas – Loading Different Files

### 2.5. Table from webpage

- ✓ We can load table from webpage.
- ✓ Whenever we are loading table from webpage then it pandas returns DataFrame.
- ✓ We can load csv file by using `read_html("url")` function

#### Syntax

```
pd.read_html("url")
```

**Program Name** Loading table from website

**Name** demo8.py

**Input** Loading from web page

```
import pandas as pd
```

```
url = 'https://en.wikipedia.org/wiki/The_World%27s_Billionaires'
df_list = pd.read_html(url)
```

```
print(df_list[2])
```

#### Output

No.		Name	Net worth (USD)	Age	Nationality	Source(s) of wealth
0	1	Elon Musk	\$219 billion	50	United States	Tesla, SpaceX
1	2	Jeff Bezos	\$177 billion	58	United States	Amazon
2	3	Bernard Arnault & family	\$158 billion	73	France	LVMH
3	4	Bill Gates	\$129 billion	66	United States	Microsoft
4	5	Warren Buffett	\$118 billion	91	United States	Berkshire Hathaway
5	6	Larry Page	\$111 billion	49	United States	Alphabet Inc.
6	7	Sergey Brin	\$107 billion	48	United States	Alphabet Inc.
7	8	Larry Ellison	\$106 billion	77	United States	Oracle Corporation
8	9	Steve Ballmer	\$91.4 billion	66	United States	Microsoft
9	10	Mukesh Ambani	\$90.7 billion	64	India	Reliance Industries

## Data Science – Pandas – DataFrame – Attributes

---

### Contents

<b>1. DataFrame Attributes .....</b>	2
1.1. Length of DataFrame .....	2
1.2. columns.....	3
1.3. shape.....	4
1.4. shape[0] .....	5
1.5. shape[1] .....	6
1.6. size .....	7
1.7. dtypes.....	9
1.8. empty .....	10
1.9. index.....	12
1.10. values .....	13
1.11. T.....	14

## 8. PANDAS – DATAFRAME – ATTRIBUTES

### 1. DataFrame Attributes

- ✓ DataFrame is a predefined class.
- ✓ DataFrame having different attributes.
- ✓ These attributes return information about the DataFrame object.

#### 1.1. Length of DataFrame

- ✓ We can check length of DataFrame by using len(p) function
- ✓ This function returns the total number of rows from the DataFrame

**Program** Checking total number of rows/length from the DataFrame  
**Name** demo1.py  
**Input file** sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")
print("Total number of rows in DataFrame:", len(df))
```

#### Output

Total number of rows in DataFrame: 600

## Data Science – Pandas – DataFrame – Attributes

### 1.2. columns

- ✓ `columns` is predefined attribute in `DataFrame` class.
- ✓ We can access `columns` attribute by using `DataFrame` object.
- ✓ This attribute returns all column names from the `DataFrame`

**Program** Accessing columns attribute from `DataFrame`.

**Name** demo2.py

**Input file** sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")
print(df.columns)
```

**Output**

```
Index(['Order ID', 'Customer Name', 'Product', 'Quantity'],
      dtype='object')
```

## Data Science – Pandas – DataFrame – Attributes

### 1.3. shape

- ✓ shape is predefined attribute in DataFrame class.
- ✓ We can access shape attribute by using DataFrame object.
- ✓ This attribute returns the total number of rows and columns in tuple format.
  - From the tuple, first value represents total number of rows
  - From the tuple, second value represents total number of columns

**Program Name** Accessing shape attribute from DataFrame.

**Input file** demo3.py

sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")
print(df.shape)
```

**Output**

(600, 4)

### 1.4. shape[0]

- ✓ shape is predefined attribute in DataFrame class.
- ✓ We can access shape attribute by using DataFrame object.
- ✓ This attribute returns the total number of rows and columns in tuple
- ✓ Shape[0] returns total number for rows from the DataFrame

**Program Name** Accessing shape attribute and checking total number of rows

**Input file** demo4.py  
sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")
print(df.shape[0])
```

**Output**

600

### 1.5. shape[1]

- ✓ shape is predefined attribute in DataFrame class.
- ✓ We can access shape attribute by using DataFrame object.
- ✓ This attribute returns the total number of rows and columns in tuple
- ✓ Shape[1] returns total number for columns from the DataFrame

**Program Name** Accessing shape attribute and checking total number of columns

**Input file** demo5.py  
sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")
print(df.shape[1])
```

**Output**

4

## Data Science – Pandas – DataFrame – Attributes

### 1.6. size

- ✓ size is predefined attribute in DataFrame class.
- ✓ We can access size attribute by using DataFrame object.
- ✓ This attribute returns the total number of elements/values in DataFrame

**Program** Accessing total number of elements/values from DataFrame.

**Name** demo6.py

**Input file** sales1.csv

```
import pandas as pd  
  
df = pd.read_csv("sales1.csv")  
print(df.size)
```

**Output**

```
2400
```

#### Make a note:

- ✓ size = Number of rows X Number of columns
- ✓ size = Row\_count X Column\_count

## Data Science – Pandas – DataFrame – Attributes

**Program Name** Accessing total number of elements/values from DataFrame.  
**Input file** demo7.py  
sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")
print("Number of elements in DataFrame:", df.size)
print("Number of elements in DataFrame:",
      df.shape[0]*df.shape[1])
```

**Output**

```
Number of elements in DataFrame: 2400
Number of elements in DataFrame: 2400
```

## Data Science – Pandas – DataFrame – Attributes

### 1.7. dtypes

- ✓ dtypes is predefined attribute in DataFrame class.
- ✓ We can access dtypes attribute by using DataFrame object.
- ✓ This attribute returns the datatype of each column.

**Program** Checking all columns datatype from DataFrame

**Name** demo8.py

**Input file** sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")
print(df.dtypes)
```

**Output**

```
Order ID      int64
Customer Name    object
Product        object
Quantity       int64
dtype: object
```

### 1.8. empty

- ✓ empty is predefined attribute in DataFrame class.
- ✓ We can access empty attribute by using DataFrame object.
- ✓ This attribute check DataFrame is empty or not,
  - If DataFrame is empty then it returns **True** other **False**.

**Program Name** Checking DataFrame empty or not

**Name** demo9.py

**Input file** sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")
print(df.empty)
```

**Output**

False

## Data Science – Pandas – DataFrame – Attributes

**Program Name** Checking DataFrame empty or not

**Name** demo10.py

**Input file** sales1.csv

```
import pandas as pd

df = pd.DataFrame()

print(df)
print()
print(df.empty)
```

**Output**

```
Empty DataFrame
Columns: []
Index: []

True
```

## Data Science – Pandas – DataFrame – Attributes

### 1.9. index

- ✓ index is predefined attribute in DataFrame class.
- ✓ We can access index attribute by using DataFrame object.
- ✓ This attribute return index start and end value from the DataFrame.

**Program Name** Accessing index attribute from DataFrame

**Input file** demo11.py

sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")
print(df.index)
```

**Output**

```
RangeIndex(start=0, stop=600, step=1)
```

### 1.10. values

- ✓ values is predefined attribute in DataFrame class.
- ✓ We can access values attribute by using DataFrame object.
- ✓ This attribute return values of DataFrame,
  - Each row values in one array from starting to last row.

**Program** Accessing values attribute from DataFrame

**Name** demo12.py

**Input file** sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")
print(df.values)
```

**Output**

```
[[166837 'Veeru' '34in Ultrawide Monitor' 2]
 [166838 'Tarun' 'Samsung m10' 3]
 [166839 'Kedar' '20in Monitor' 1]
 ...
 [167405 'Venu' 'Flatscreen TV' 1]
 [167406 'Siddhu' 'Samsung m20' 2]
 [167407 'Tarun' 'LG Washing Machine' 1]]
```

## Data Science – Pandas – DataFrame – Attributes

### 1.11. T

- ✓ T is predefined attribute in DataFrame class.
- ✓ We can access T attribute by using DataFrame object.
- ✓ T means its transpose, it returns **rows as columns** and **columns as rows**

**Program Name** Converting rows as column and columns as rows from DataFrame demo13.py

```
import pandas as pd

details = [["Sagar", 20, 10000], ["Daniel", 16, 20000], ["Veeru", 24, 30000], ["Raju", 25, 40000], ["Kiran", 26, 50000], ["Kedar", 27, 60000], ["Reena", 28, 70000]]

df = pd.DataFrame(details, columns = ["Name", "Age", "Salary"])

print(df)
print()
print(df.T)
```

### Output

	Name	Age	Salary
0	Sagar	20	10000
1	Daniel	16	20000
2	Veeru	24	30000
3	Raju	25	40000
4	Kiran	26	50000
5	Kedar	27	60000
6	Reena	28	70000

	0	1	2	3	4	5	6
Name	Sagar	Daniel	Veeru	Raju	Kiran	Kedar	Reena
Age	20	16	24	25	26	27	28
Salary	10000	20000	30000	40000	50000	60000	70000

# Data Science – Pandas – DataFrame – Methods

---

## Contents

<b>1. DataFrame Methods .....</b>	2
1.1. head() .....	2
1.2. tail() .....	3
1.3. info().....	4
1.4. count().....	6
1.5. describe() .....	8
1.6. nunique().....	9
<b>2. Accessing single column From DataFrame .....</b>	10
<b>3. Rearranging columns in DataFrame .....</b>	14

## 9. PANDAS – DATAFRAME – METHODS

### 1. DataFrame Methods

- ✓ DataFrame is a predefined class.
- ✓ DataFrame having different methods.
- ✓ These methods perform an operation on DataFrame and returns result

#### 1.1. head()

- ✓ head() is predefined method in DataFrame class.
- ✓ We can access head() method by using DataFrame object only.
- ✓ This method returns first five rows from the DataFrame

**Program** Accessing first five rows from DataFrame

**Name** demo1.py

**Input file** sales1.csv

```
import pandas as pd

df1 = pd.read_csv("sales1.csv")
df2 = df1.head()

print(df2)
```

**Output**

	Order ID	Customer Name	Product	Quantity
0	166837	Veeru	34in Ultrawide Monitor	2
1	166838	Tarun	Samsung m10	3
2	166839	Kedar	20in Monitor	1
3	166840	Lavanya	iPhone 11	3
4	166841	Venu	Macbook Pro Laptop	2

## Data Science – Pandas – DataFrame – Methods

### 1.2. tail()

- ✓ tail() is predefined method in DataFrame class.
- ✓ We can access tail() method by using DataFrame object only.
- ✓ This method returns last five rows from the DataFrame

**Program** Accessing last five rows from DataFrame

**Name** demo2.py

**Input file** sales1.csv

```
import pandas as pd

df1 = pd.read_csv("sales1.csv")
df2 = df1.tail()

print(df2)
```

**Output**

	Order ID	Customer Name	Product	Quantity
595	167403	Balaji	Macbook Pro Laptop	1
596	167404	Lavanya	ThinkPad Laptop	1
597	167405	Venu	Flatscreen TV	1
598	167406	Siddhu	Samsung m20	2
599	167407	Tarun	LG Washing Machine	1

### 1.3. info()

- ✓ info() is predefined method in DataFrame class.
- ✓ We can access info() method by using DataFrame object only.
- ✓ This method returns below information about DataFrame,
  - Type of object
  - Range of object
  - Number of columns
  - Number of rows
  - The data type of each column
  - Number of data types
  - Total memory usage

**Program Name** Accessing info() method from DataFrame  
**Input file** demo3.py  
 sales1.csv

```
import pandas as pd

df1 = pd.read_csv("sales1.csv")
df1.info()
```

### Output

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 600 entries, 0 to 599
Data columns (total 4 columns):
 #   Column           Non-Null Count  Dtype  
---  --  
 0   Order ID        600 non-null    int64  
 1   Customer Name  600 non-null    object  
 2   Product         600 non-null    object  
 3   Quantity        600 non-null    int64  
dtypes: int64(2), object(2)
memory usage: 18.9+ KB
```

## Data Science – Pandas – DataFrame – Methods

**Program Name** Accessing info() method from DataFrame  
**Input file** demo4.py  
sales1\_with\_nan.csv

```
import pandas as pd

df1 = pd.read_csv("sales1_with_nan.csv")
df1.info()
```

### Output

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 600 entries, 0 to 599
Data columns (total 4 columns):
 #   Column           Non-Null Count  Dtype  
---  --  
 0   Order ID        600 non-null    int64  
 1   Customer Name  599 non-null    object  
 2   Product         600 non-null    object  
 3   Quantity        598 non-null    float64 
dtypes: float64(1), int64(1), object(2)
memory usage: 18.9+ KB
```

## Data Science – Pandas – DataFrame – Methods

### 1.4. count()

- ✓ count() is predefined method in DataFrame class.
- ✓ We can access count() method by using DataFrame object only.
- ✓ This method returns number of non-null values from each column.

**Program** Accessing count() method from DataFrame

**Name** demo5.py

**Input file** sales1.csv

```
import pandas as pd

df1 = pd.read_csv("sales1.csv")
c = df1.count()
print(c)
```

**Output**

```
Order ID      600
Customer Name 600
Product        600
Quantity       600
dtype: int64
```

## Data Science – Pandas – DataFrame – Methods

**Program Name** Accessing count() method from DataFrame

**Input file** demo6.py

**Output file** sales1\_with\_nan.csv

```
import pandas as pd

df1 = pd.read_csv("sales1_with_nan.csv")
c = df1.count()

print(c)
```

**Output**

```
Order ID      600
Customer Name 599
Product       600
Quantity      598
dtype: int64
```

## Data Science – Pandas – DataFrame – Methods

---

### 1.5. describe()

- ✓ `describe()` is predefined method in `DataFrame` class.
- ✓ We can access `describe()` method by using `DataFrame` object.
- ✓ This method returns the below values,
  - `count`
  - `mean`
  - `std`
  - `min`
  - `25%`
  - `50%`
  - `75%`
  - `max`

**Program Name** Accessing `describe()` method from `DataFrame`

**Name** demo7.py

**Input file** sales1\_with\_nan.csv

```
import pandas as pd

df1 = pd.read_csv("sales1.csv")
dsc = df1.describe()

print(dsc)
```

**Output**

	Order ID	Quantity
count	600.000000	600.000000
mean	167122.751667	1.481667
std	164.948568	0.683454
min	166837.000000	1.000000
25%	166980.750000	1.000000
50%	167120.500000	1.000000
75%	167266.250000	2.000000
max	167409.000000	3.000000

## Data Science – Pandas – DataFrame – Methods

### 1.6. nunique()

- ✓ nunique() is predefined method in DataFrame class.
- ✓ We can access nunique() method by using DataFrame object only.
- ✓ This method returns number of unique values from the DataFrame.

**Program Name** Get the number of nunique values from the DataFrame  
demo8.py

```
import pandas as pd

df1 = pd.read_csv("sales1.csv")
nu = df1.nunique()

print(nu)
```

#### Output

```
Order ID      573
Customer Name  23
Product        21
Quantity        3
dtype: int64
```

## 2. Accessing single column From DataFrame

- ✓ We can access columns from the DataFrame,
  - If we access single column then it returns the Series
  - If we access two column then it returns the DataFrame with two columns

**Program Name** Accessing single column from the DataFrame  
**Input file** demo9.py  
sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")

print(df.Product)
```

### Output

```
0      34in Ultrawide Monitor
1                      Samsung m10
2                  20in Monitor
3                  iPhone 11
4      Macbook Pro Laptop
...
595      Macbook Pro Laptop
596      ThinkPad Laptop
597      Flatscreen TV
598      Samsung m20
599      LG Washing Machine
Name: Product, Length: 600, dtype: object
```

## Data Science – Pandas – DataFrame – Methods

**Program Name** Accessing single column from the DataFrame  
**Input file** demo10.py  
sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")

print(df["Product"])
```

### Output

```
0      34in Ultrawide Monitor
1                  Samsung m10
2                  20in Monitor
3                  iPhone 11
4      Macbook Pro Laptop
       ...
595     Macbook Pro Laptop
596     ThinkPad Laptop
597     Flatscreen TV
598     Samsung m20
599     LG Washing Machine
Name: Product, Length: 600, dtype: object
```

## Data Science – Pandas – DataFrame – Methods

**Program Name** Accessing two column from the DataFrame  
**Input file** demo11.py  
sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")

print(df[["Customer Name", "Product"]])
```

### Output

```
      Customer Name          Product
0           Veeru  34in Ultrawide Monitor
1           Tarun           Samsung m10
2           Kedar        20in Monitor
3           Lavanya         iPhone 11
4           Venu     Macbook Pro Laptop
..            ...
595          Balaji     Macbook Pro Laptop
596          Lavanya    ThinkPad Laptop
597          Venu       Flatscreen TV
598          Siddhu        Samsung m20
599          Tarun      LG Washing Machine

[600 rows x 2 columns]
```

## Data Science – Pandas – DataFrame – Methods

---

**Program Name** Accessing single column from the DataFrame, applying sum  
**Input file** demo12.py  
sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")
total = df['Quantity'].sum()

print(total)
```

**Output**

889

### 3. Rearranging columns in DataFrame

- ✓ We can rearrange columns in DataFrame
- ✓ We can customise the order of columns in DataFrame

**Program Name** Creating DataFrame by loading csv file  
**Name** demo13.py  
**Input file** sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")

print(df)
```

#### Output

	Order ID	Customer Name	Product	Quantity
0	166837	Veeru	34in Ultrawide Monitor	2
1	166838	Tarun	Samsung m10	3
2	166839	Kedar	20in Monitor	1
3	166840	Lavanya	iPhone 11	3
4	166841	Venu	Macbook Pro Laptop	2
..	...	...	...	...
595	167403	Balaji	Macbook Pro Laptop	1
596	167404	Lavanya	ThinkPad Laptop	1
597	167405	Venu	Flatscreen TV	1
598	167406	Siddhu	Samsung m20	2
599	167407	Tarun	LG Washing Machine	1
[600 rows x 4 columns]				

## Data Science – Pandas – DataFrame – Methods

**Program Name** Rearranging columns in DataFrame  
**Input file** demo14.py  
sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")
df = df[["Product", "Customer Name", "Quantity", "Order ID"]]

print(df)
```

### Output

	Product	Customer Name	Quantity	Order ID
0	34in Ultrawide Monitor	Veeru	2	166837
1	Samsung m10	Tarun	3	166838
2	20in Monitor	Kedar	1	166839
3	iPhone 11	Lavanya	3	166840
4	Macbook Pro Laptop	Venu	2	166841
..	...	...	...	...
595	Macbook Pro Laptop	Balaji	1	167403
596	ThinkPad Laptop	Lavanya	1	167404
597	Flatscreen TV	Venu	1	167405
598	Samsung m20	Siddhu	2	167406
599	LG Washing Machine	Tarun	1	167407

[600 rows x 4 columns]

**10. Pandas – DataFrame – Rename column & index****Contents**

1. rename(p) method - Changing single column name .....	2
2. rename(p) method - Changing multiple column names .....	5
3. columns attribute - Changing multiple column names.....	8
4. rename(p) method - Changing index in DataFrame.....	10
5. index attribute - Changing multiple column names.....	12
6. Converting column names from lower case to upper case .....	15

## 10. Pandas – DataFrame – Rename column & index

- ✓ Sometimes we need to process column names and index.
- ✓ Based on requirement we can rename/change/modify/update column names and index.

### 1. rename(p) method - Changing single column name

- ✓ rename(p) is a predefined method in DataFrame
- ✓ We can access rename() method by using DataFrame object.
- ✓ By using this method we can change single column name in DataFrame.

**Program Name** Creating DataFrame and checking column names  
**Input file** demo1.py  
**sales3.csv**

```
import pandas as pd

df = pd.read_csv("sales3.csv")
print(df.head())
print()
print(df.columns)
```

#### Output

ord id	cust name	cust id	prod name	prod cost
0 192837	Veeru	3	LG Mobile	65999
1 192838	Neelima	19	Apple iPad 10.2-inch	63000
2 192839	Balaji	12	34in Ultrawide Monitor	75999
3 192840	Shahid	20	iPhone 11	60000
4 192841	Vinay	10	Bose SoundSport Headphones	69999

```
Index(['ord id', 'cust name', 'cust id', 'prod name', 'prod cost'], dtype='object')
```

## Data Science – Pandas – DataFrame – Rename column, index

### Note

- ✓ We need to use python dictionary to specify old column as key and new column as value.
- ✓ Whenever changing column then key should be match with column name in DataFrame otherwise changes will not reflects.

**Program Name** Renaming single column in DataFrame  
**Input file** demo2.py  
sales3.csv

```
import pandas as pd

df1 = pd.read_csv("sales3.csv")
d = {
    "ord id": "Order Id"
}

df2 = df1.rename(columns = d)

print(df1.head())
print()
print(df2.head())
```

### Output

	ord id	cust name	cust id	prod name	prod cost
0	192837	Veeru	3	LG Mobile	65999
1	192838	Neelima	19	Apple iPad 10.2-inch	63000
2	192839	Balaji	12	34in Ultrawide Monitor	75999
3	192840	Shahid	20	iPhone 11	60000
4	192841	Vinay	10	Bose SoundSport Headphones	69999

	Order Id	cust name	cust id	prod name	prod cost
0	192837	Veeru	3	LG Mobile	65999
1	192838	Neelima	19	Apple iPad 10.2-inch	63000
2	192839	Balaji	12	34in Ultrawide Monitor	75999
3	192840	Shahid	20	iPhone 11	60000
4	192841	Vinay	10	Bose SoundSport Headphones	69999

## Data Science – Pandas – DataFrame – Rename column, index

### Note

- ✓ Whenever changing column then key should be match with column name in DataFrame otherwise changes will not reflects.

**Program** Renaming single column in DataFrame

**Name** demo3.py

**Input file** sales3.csv

```
import pandas as pd

df1 = pd.read_csv("sales3.csv")

d = {
    "ord id": "Order Id"
}

df2 = df1.rename(columns = d)

print(df1.head())
print()
print(df2.head())
```

### Output

	ord id	cust name	cust id	prod name	prod cost
0	192837	Veeru	3	LG Mobile	65999
1	192838	Neelima	19	Apple iPad 10.2-inch	63000
2	192839	Balaji	12	34in Ultrawide Monitor	75999
3	192840	Shahid	20	iPhone 11	60000
4	192841	Vinay	10	Bose SoundSport Headphones	69999

	ord id	cust name	cust id	prod name	prod cost
0	192837	Veeru	3	LG Mobile	65999
1	192838	Neelima	19	Apple iPad 10.2-inch	63000
2	192839	Balaji	12	34in Ultrawide Monitor	75999
3	192840	Shahid	20	iPhone 11	60000
4	192841	Vinay	10	Bose SoundSport Headphones	69999

## 2. rename(p) method - Changing multiple column names

- ✓ rename(p) is a predefined method in DataFrame
- ✓ We can access rename() method by using DataFrame object.
- ✓ We can even change multiple column names by using rename(p) method.
- ✓ We need to use python dictionary to specify old column as key and new column as value.

**Program Name** Renaming multiple column in DataFrame  
**Input file** demo4.py  
**sales3.csv**

```
import pandas as pd

df1 = pd.read_csv("sales3.csv")

d = {
    'ord id': 'Order Id',
    'cust name': 'Customer Name',
    'cust id': 'Customer Id',
    'prod name': 'Product Name',
    'prod cost': 'Product Cost'
}

df2 = df1.rename(columns = d)

print(df1.head())
print()
print(df2.head())
```

**Output**

## Data Science – Pandas – DataFrame – Rename column, index

	ord id	cust name	cust id	prod name	prod cost
Order Id	Customer Name	Customer Id	Product Name	Product Cost	
0	192837	Veeru	3	LG Mobile	65999
1	192838	Neelima	19	Apple iPad 10.2-inch	63000
2	192839	Balaji	12	34in Ultrawide Monitor	75999
3	192840	Shahid	20	iPhone 11	60000
4	192841	Vinay	10	Bose SoundSport Headphones	69999

## Data Science – Pandas – DataFrame – Rename column, index

**Program Name** Renaming multiple column in DataFrame  
**Input file** demo5.py  
sales3.csv

```
import pandas as pd

df1 = pd.read_csv("sales3.csv")

d = {
    "ord id": "order_id",
    "cust name": "customer_name",
    "cust id": "customer_id",
    "prod name": "product_name",
    "prod cost": "product_cost"
}

df2 = df1.rename(columns = d)

print(df1.head())
print()
print(df2.head())
```

### Output

	ord id	cust name	cust id	prod name	prod cost
0	192837	Veeru	3	LG Mobile	65999
1	192838	Neelima	19	Apple iPad 10.2-inch	63000
2	192839	Balaji	12	34in Ultrawide Monitor	75999
3	192840	Shahid	20	iPhone 11	60000
4	192841	Vinay	10	Bose SoundSport Headphones	69999
	order_id	customer_name	customer_id	product_name	product_cost
0	192837	Veeru	3	LG Mobile	65999
1	192838	Neelima	19	Apple iPad 10.2-inch	63000
2	192839	Balaji	12	34in Ultrawide Monitor	75999
3	192840	Shahid	20	iPhone 11	60000
4	192841	Vinay	10	Bose SoundSport Headphones	69999

## Data Science – Pandas – DataFrame – Rename column, index

### 3. columns attribute - Changing multiple column names

- ✓ columns is predefined attribute in DataFrame class.
- ✓ We can access columns attribute by using DataFrame object.
- ✓ By using columns attribute we can even change multiple column names.

**Program Name** Renaming multiple column in DataFrame

**Input file** demo6.py

**sales3.csv**

```
import pandas as pd

df1 = pd.read_csv("sales3.csv")

print(df1.head())

df1.columns = [
    "order_id",
    "customer_name",
    "customer_id",
    "product_name",
    "product_cost"
]

print()
print(df1.head())
```

### Output

	ord id	cust name	cust id	prod name	prod cost
0	192837	Veeru	3	LG Mobile	65999
1	192838	Neelima	19	Apple iPad 10.2-inch	63000
2	192839	Balaji	12	34in Ultrawide Monitor	75999
3	192840	Shahid	20	iPhone 11	60000
4	192841	Vinay	10	Bose SoundSport Headphones	69999
	order_id	customer_name	customer_id	product_name	product_cost
0	192837	Veeru	3	LG Mobile	65999
1	192838	Neelima	19	Apple iPad 10.2-inch	63000
2	192839	Balaji	12	34in Ultrawide Monitor	75999
3	192840	Shahid	20	iPhone 11	60000
4	192841	Vinay	10	Bose SoundSport Headphones	69999

## Data Science – Pandas – DataFrame – Rename column, index

### Note

- ✓ While using columns attributes to change column names then Number of DataFrame column names should be match with existing columns otherwise we will get an error.

**Program** Renaming multiple column in DataFrame  
**Name** demo7.py  
**Input file** sales3.csv

```
import pandas as pd

df1 = pd.read_csv("sales3.csv")

print(df1.head())

df1.columns = [
    "order_id",
    "customer_name",
    "customer_id",
    "product_name",
    "product_cost",
    "total"
]

print()
print(df1.head())
```

### Output

**ValueError:** Length mismatch: Expected axis has 5 elements, new values have 6 elements

**4. rename(p) method - Changing index in DataFrame**

- ✓ rename(p) is a predefined method in DataFrame
- ✓ We can access rename() method by using DataFrame object.
- ✓ Based on requirement we can change the index in DataFrame.
- ✓ By using rename(p) we can change the DataFrame index

**Program Name** Creating a sample Dataframe  
demo8.py

```
import pandas as pd

d = {
    "order_id": [11, 21, 31],
    "customer_name": ["Prasad", "Daniel", "Jeswanth"],
    "product": ["iPhone", "hTC", "macbook"]
}

df1 = pd.DataFrame(d)
print(df1)
```

**Output**

	order_id	customer_name	product
0	11	Prasad	iPhone 11
1	21	Daniel	hTC
2	31	Jeswanth	macbook

## Data Science – Pandas – DataFrame – Rename column, index

**Program Name** Changing index in DataFrame  
demo9.py

```
import pandas as pd

d = {
    "order_id": [11, 21, 31],
    "customer_name": ["Prasad", "Daniel", "Jeswanth"],
    "product": ["iPhone", "hTC", "macbook"]
}

i = {0: 77, 1: 88, 2: 99}

df1 = pd.DataFrame(d)
df2 = df1.rename(index = i)

print(df1)
print()
print(df2)
```

**Output**

```
   order_id  customer_name   product
0         11           Prasad     iPhone
1         21           Daniel        hTC
2         31       Jeswanth    macbook

   order_id  customer_name   product
77        11           Prasad     iPhone
88        21           Daniel        hTC
99        31       Jeswanth    macbook
```

## 5. index attribute - Changing multiple column names

- ✓ index is predefined attribute in DataFrame class.
- ✓ We can access index attribute by using DataFrame object.
- ✓ By using index attribute we can even change index in DataFrame

**Program Name**      Changing index in DataFrame, using axis parameter  
**demo10.py**

```
import pandas as pd

d = {
    "order_id": [11, 21, 31],
    "customer_name": ["Prasad", "Daniel", "Jeswanth"],
    "product": ["iPhone", "hTC", "macbook"]
}

df1 = pd.DataFrame(d)
print(df1)

df1.index = [77, 88, 99]

print()
print(df1)
```

### Output

	order_id	customer_name	product
0	11	Prasad	iPhone
1	21	Daniel	hTC
2	31	Jeswanth	macbook

	order_id	customer_name	product
77	11	Prasad	iPhone
88	21	Daniel	hTC
99	31	Jeswanth	macbook

## Data Science – Pandas – DataFrame – Rename column, index

**Program Name** Changing index in DataFrame, using index attribute.  
**Input file** demo11.py  
sales31.csv

```
import pandas as pd

df1 = pd.read_csv("sales31.csv")
print(df1)

df1.index = range(10, 20)

print()
print(df1)
```

### Output

	ord id	cust name	cust id	prod name	prod cost
0	192837	Veeru	3	LG Mobile	65999
1	192838	Neelima	19	Apple iPad 10.2-inch	63000
2	192839	Balaji	12	34in Ultrawide Monitor	75999
3	192840	Shahid	20	iPhone 11	60000
4	192841	Vinay	10	Bose SoundSport Headphones	69999
5	192842	Vinay	10	20in Monitor	65999
6	192843	Balaji	12	iPhone 8	55999
7	192844	Nireekshan	1	27in 4K Gaming Monitor	55999
8	192845	Venu	23	iPhone 9	55999
9	192846	Veeru	3	20in Monitor	63000
	ord id	cust name	cust id	prod name	prod cost
10	192837	Veeru	3	LG Mobile	65999
11	192838	Neelima	19	Apple iPad 10.2-inch	63000
12	192839	Balaji	12	34in Ultrawide Monitor	75999
13	192840	Shahid	20	iPhone 11	60000
14	192841	Vinay	10	Bose SoundSport Headphones	69999
15	192842	Vinay	10	20in Monitor	65999
16	192843	Balaji	12	iPhone 8	55999
17	192844	Nireekshan	1	27in 4K Gaming Monitor	55999
18	192845	Venu	23	iPhone 9	55999
19	192846	Veeru	3	20in Monitor	63000

## Data Science – Pandas – DataFrame – Rename column, index

**Program Name** Changing columns and index in a Dataframe  
demo12.py

```
import pandas as pd

d = {
    "Ord Id": [11, 21, 31],
    "Customer Name": ["Prasad", "Daniel", "Jeswanth"],
    "Product": ["iPhone", "hTC", "macbook"]
}

df1 = pd.DataFrame(d)
print(df1)

df1.index = [333, 444, 555]
df1.columns = ["order_id", "customer_name", "product"]

print()
print(df1)
```

**Output**

```
      Ord Id Customer Name  Product
0          11        Prasad   iPhone
1          21        Daniel     hTC
2          31      Jeswanth  macbook

      order_id customer_name  product
333           11        Prasad   iPhone
444           21        Daniel     hTC
555           31      Jeswanth  macbook
```

## Data Science – Pandas – DataFrame – Rename column, index

### 6. Converting column names from lower case to upper case

- ✓ Based on requirement we can convert DataFrame column names from lower case to upper case.

**Program Name** demo13.py  
**Input file** sales3.csv

```
import pandas as pd

df1 = pd.read_csv("sales3.csv")

print(df1.head())

df1.columns = df1.columns.str.upper()

print()
print(df1.head())
```

#### Output

	ord id	cust name	cust id	prod name	prod cost
0	192837	Veeru	3	LG Mobile	65999
1	192838	Neelima	19	Apple iPad 10.2-inch	63000
2	192839	Balaji	12	34in Ultrawide Monitor	75999
3	192840	Shahid	20	iPhone 11	60000
4	192841	Vinay	10	Bose SoundSport Headphones	69999

	ORD ID	CUST NAME	CUST ID	PROD NAME	PROD COST
0	192837	Veeru	3	LG Mobile	65999
1	192838	Neelima	19	Apple iPad 10.2-inch	63000
2	192839	Balaji	12	34in Ultrawide Monitor	75999
3	192840	Shahid	20	iPhone 11	60000
4	192841	Vinay	10	Bose SoundSport Headphones	69999

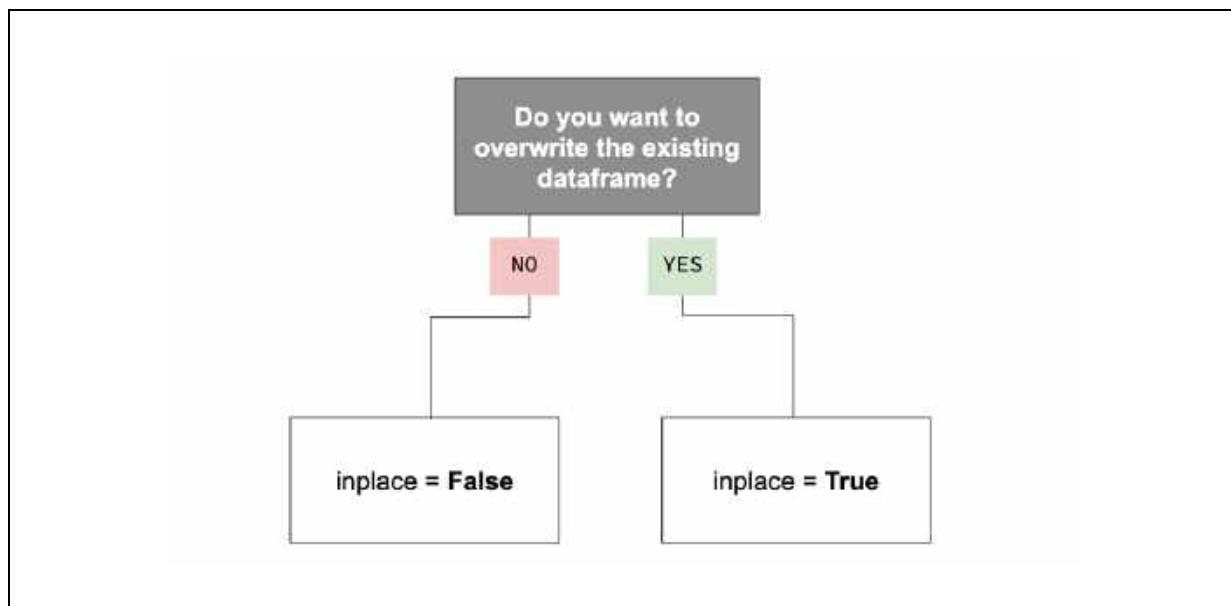
**11. PANDAS – INPLACE PARAMETER****Contents**

<b>1. The inplace parameter .....</b>	<b>2</b>
<b>2. rename(p) method – without inplace parameter.....</b>	<b>3</b>
<b>3. rename(p) method – with inplace parameter.....</b>	<b>4</b>

## 11. PANDAS – INPLACE PARAMETER

### 1. The inplace parameter

- ✓ inplace is a parameter for DataFrame methods like,
  - rename(p, inplace)
  - drop(p, inplace),
  - sort\_values(p, inplace),
  - set\_index(p, inplace) etc
- ✓ This inplace parameter accepts two values,
  - inplace = **False** means it cannot overwrite the existing dataframe
  - inplace = **True** means it can overwrite the existing dataframe.



### Simple words

- ✓ In-place means that you should update the original object instead of creating new object

## Data Science – PANDAS – INPLACE PARAMETER

### 2. rename(p) method – without inplace parameter

- ✓ rename(p) is a predefined method in DataFrame
- ✓ We can access rename() method by using DataFrame object.
- ✓ We can even change multiple column names by using rename(p) method.

**Program** rename(p) method - without inplace parameter

**Name** demo1.py

**Input file** sales3.csv

```
import pandas as pd

df1 = pd.read_csv("sales3.csv")

d = {"ord id": "order_id", "cust name": "customer_name", "cust id": "customer_id", "prod name": "product_name", "prod cost": "product_cost"}

print(df1.head())
df1.rename(columns = d)
print()
print(df1.head())
```

### Output

	ord id	cust name	cust id	prod name	prod cost
0	192837	Veeru	3	LG Mobile	65999
1	192838	Neelima	19	Apple iPad 10.2-inch	63000
2	192839	Balaji	12	34in Ultrawide Monitor	75999
3	192840	Shahid	20	iPhone 11	60000
4	192841	Vinay	10	Bose SoundSport Headphones	69999

	ord id	cust name	cust id	prod name	prod cost
0	192837	Veeru	3	LG Mobile	65999
1	192838	Neelima	19	Apple iPad 10.2-inch	63000
2	192839	Balaji	12	34in Ultrawide Monitor	75999
3	192840	Shahid	20	iPhone 11	60000
4	192841	Vinay	10	Bose SoundSport Headphones	69999

### 3. rename(p) method – with inplace parameter

- ✓ rename(p, inplace = **True**) is a predefined method in DataFrame.
- ✓ We can provide inplace parameter to rename(p, inplace = **True**) method
- ✓ If we provide inplace parameter then directly changes will be apply on original DataFrame.
- ✓ Whenever if we apply inplace = **True** then, it should update the original DataFrame instead of creating new DataFrame.

**Program** rename(p) method - with inplace parameter

**Name** demo2.py

**Input file** sales3.csv

```
import pandas as pd

df1 = pd.read_csv("sales3.csv")

d = {"ord id": "order_id", "cust name": "customer_name", "cust id": "customer_id", "prod name": "product_name", "prod cost": "product_cost"}

print(df1.head())
df1.rename(columns = d, inplace = True)
print()
print(df1.head())
```

**Output**

	ord id	cust name	cust id	prod name	prod cost
0	192837	Veeru	3	LG Mobile	65999
1	192838	Neelima	19	Apple iPad 10.2-inch	63000
2	192839	Balaji	12	34in Ultrawide Monitor	75999
3	192840	Shahid	20	iPhone 11	60000
4	192841	Vinay	10	Bose SoundSport Headphones	69999
	order_id	customer_name	customer_id	product_name	product_cost
0	192837	Veeru	3	LG Mobile	65999
1	192838	Neelima	19	Apple iPad 10.2-inch	63000
2	192839	Balaji	12	34in Ultrawide Monitor	75999
3	192840	Shahid	20	iPhone 11	60000
4	192841	Vinay	10	Bose SoundSport Headphones	69999

## Data Science - Pandas – iLOC and LOC

---

### 13. PANDAS – iLOC AND LOC

#### Contents

1. Selecting single column.....	2
2. Selecting multiple columns .....	4
3. Selecting specific column values.....	6
4. iloc and loc indexers.....	9
5. iloc[] indexer.....	10
5.1. iloc[] syntax .....	10
5.2. Creating a DataFrame .....	11
5.3. Selecting single column.....	12
5.4. Selecting multiple rows and columns .....	19
6. loc[] indexer.....	24
6.1. loc[] - Selecting rows by label/index .....	25
6.2. Creating a dataframe .....	25
6.3. Boolean / Logical indexing .....	35

### 13. PANDAS – iLOC AND LOC

#### 1. Selecting single column

- ✓ Based on requirement we can select columns from the DataFrame.
  - If we select a single column then it returns the Series

**Program Name** Selecting single column from the DataFrame  
**Input file** demo1.py  
sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")
print(df.Product)
```

#### Output

```
0      34in Ultrawide Monitor
1                      Samsung m10
2                  20in Monitor
3                  iPhone 11
4      Macbook Pro Laptop
        ...
595     Macbook Pro Laptop
596     ThinkPad Laptop
597     Flatscreen TV
598     Samsung m20
599     LG Washing Machine
Name: Product, Length: 600, dtype: object
```

## Data Science - Pandas – iLOC and LOC

**Program Name** Selecting single column from the DataFrame  
**Input file** demo2.py  
sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")
s = df["Product"]
print(s)
```

### Output

```
0      34in Ultrawide Monitor
1                  Samsung m10
2                  20in Monitor
3                  iPhone 11
4      Macbook Pro Laptop
       ...
595     Macbook Pro Laptop
596     ThinkPad Laptop
597     Flatscreen TV
598     Samsung m20
599     LG Washing Machine
Name: Product, Length: 600, dtype: object
```

## 2. Selecting multiple columns

- ✓ Based on requirement we can select columns from the DataFrame.
  - If we select more than one column then it returns the DataFrame.

**Program** Selecting multiple columns from the DataFrame  
**Name** demo3.py  
**Input file** sales1.csv

```
import pandas as pd

df1 = pd.read_csv("sales1.csv")
cols = ["Customer Name", "Product"]
df2 = df1[cols]

print(df2)
```

### Output

	Customer Name	Product
0	Veeru	34in Ultrawide Monitor
1	Tarun	Samsung m10
2	Kedar	20in Monitor
3	Lavanya	iPhone 11
4	Venu	Macbook Pro Laptop
..	...	...
595	Balaji	Macbook Pro Laptop
596	Lavanya	ThinkPad Laptop
597	Venu	Flatscreen TV
598	Siddhu	Samsung m20
599	Tarun	LG Washing Machine
[600 rows x 2 columns]		

## Data Science - Pandas – iLOC and LOC

**Program Name** Selecting single column from the DataFrame, apply total  
**Input file** demo4.py  
sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")
total = df['Quantity'].sum()
print(total)
```

**Output**

889

### 3. Selecting specific column values

- ✓ Based on requirement we can select specific column values from the DataFrame
  - By using Boolean condition we can select the data from DataFrame.

**Program Name** Creating a DataFrame by loading csv file  
**Input file** demo5.py  
 sales1.csv

```
import pandas as pd

df1 = pd.read_csv("sales1.csv")

print(df1)
```

#### Output

	Order ID	Customer Name	Product	Quantity
0	166837	Veeru	34in Ultrawide Monitor	2
1	166838	Tarun	Samsung m10	3
2	166839	Kedar	20in Monitor	1
3	166840	Lavanya	iPhone 11	3
4	166841	Venu	Macbook Pro Laptop	2
..	..	..	..	..
595	167403	Balaji	Macbook Pro Laptop	1
596	167404	Lavanya	ThinkPad Laptop	1
597	167405	Venu	Flatscreen TV	1
598	167406	Siddhu	Samsung m20	2
599	167407	Tarun	LG Washing Machine	1
[600 rows x 4 columns]				

## Data Science - Pandas – iLOC and LOC

**Program Name** Select specific customer from existing DataFrame  
**Input file** demo6.py  
sales1.csv

```
import pandas as pd

df1 = pd.read_csv("sales1.csv")

cust_name = df1["Customer Name"] == "Veeru"

print(df1[cust_name])
```

### Output

	Order ID	Customer Name	Product	Quantity
0	166837	Veeru	34in Ultrawide Monitor	2
25	166861	Veeru	Wired Headphones	1
34	166870	Veeru	Wired Headphones	2
61	166897	Veeru	USB-C Charging Cable	2
65	166901	Veeru	Samsung m10	2
77	166913	Veeru	iPhone 9	1
85	166920	Veeru	iPhone 9	2
88	166923	Veeru	27in FHD Monitor	1
113	166948	Veeru	Samsung m10	1
117	166952	Veeru	LG Washing Machine	2
129	166964	Veeru	Samsung m10	1
130	166965	Veeru	Bose SoundSport Headphones	1
134	166968	Veeru	iPhone 7	2
190	167019	Veeru	Macbook Pro Laptop	2
283	167105	Veeru	LG Washing Machine	1
315	167135	Veeru	LG Washing Machine	1
322	167142	Veeru	iPhone 7s	1
326	167146	Veeru	Samsung m20	2

## Data Science - Pandas – iLOC and LOC

---

**Program Name** Select specific customer from existing DataFrame  
**Input file** demo7.py  
sales1.csv

```
import pandas as pd

df1 = pd.read_csv("sales1.csv")

prod_name = df1["Product"] == "Macbook Pro Laptop"

print(df1[prod_name])
```

### Output

	Order ID	Customer Name	Product	Quantity
4	166841	Venu	Macbook Pro Laptop	2
11	166848	Karteek	Macbook Pro Laptop	1
15	166851	Jaya Chandra	Macbook Pro Laptop	1
71	166907	Karteek	Macbook Pro Laptop	1
122	166957	Harsha	Macbook Pro Laptop	2
136	166970	Tarun	Macbook Pro Laptop	2
156	166986	Pradhan	Macbook Pro Laptop	1
184	167013	Sumanth	Macbook Pro Laptop	1
190	167019	Veeru	Macbook Pro Laptop	2
192	167021	Mallikarjun	Macbook Pro Laptop	1
199	167027	Balaji	Macbook Pro Laptop	1
225	167050	Venu	Macbook Pro Laptop	2
242	167067	Madhurima	Macbook Pro Laptop	2
251	167074	Lavanya	Macbook Pro Laptop	1
261	167084	Shahid	Macbook Pro Laptop	1
285	167107	Jaya Chandra	Macbook Pro Laptop	1
302	167123	Karteek	Macbook Pro Laptop	1
310	167130	Partha	Macbook Pro Laptop	2
332	167152	Venki	Macbook Pro Laptop	1
348	167167	Kedar	Macbook Pro Laptop	2
353	167172	Balaji	Macbook Pro Laptop	1
357	167176	Shahid	Macbook Pro Laptop	2

### 4. iloc and loc indexers

- ✓ We can select columns from the DataFrame by using iloc and loc.
- ✓ **iloc** and **loc** are called as indexers in DataFrame
- ✓ By using these indexers we can get,
  - Rows and columns of DataFrame
  - Slice of DataFrame

## 5. iloc[] indexer

- ✓ The iloc is used for indexed-based selection method.
- ✓ We have to pass only integer index to select specific row/column.
- ✓ By using this we can get rows or columns at particular positions in the index (so it only takes integers).
- ✓ This does not include the last element in DataFrame

### 5.1. iloc[] syntax

#### iloc[] syntax

- ✓ `df.iloc[<row selection>]`

#### iloc[] syntax

- ✓ `df.iloc[<row selection>, <column selection>]`

#### Note on syntax

- ✓ So, according to the syntax there are two arguments,
  - Row selection
  - Column selection

## Data Science - Pandas – iLOC and LOC

### 5.2. Creating a DataFrame

- ✓ Whenever we load a csv file then pandas returns the DataFrame

**Program Name** Loading csv file by using pandas  
**Input file** demo8.py  
sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
print(df1)
```

#### Output

	Order id	Customer name	Customer id	Product name	Product cost
0	192837	Partha	8	Apple iPad 10.2-inch	59000
1	192838	Vinay	10	Flatscreen TV	65999
2	192839	Lavanya	16	20in Monitor	75000
3	192840	Mohan	24	Bose SoundSport Headphones	55000
4	192841	Kedar	2	Google Phone	59000
..	...	...	...	...	...
895	193732	Balaji	12	Samsung Galaxy S20	60000
896	193733	Neelima	19	27in 4K Gaming Monitor	75999
897	193734	Siddhu	18	Google Phone	69999
898	193735	Vinay	10	20in Monitor	69999
899	193736	Madhurima	7	27in FHD Monitor	55999

[900 rows x 5 columns]

### 5.3. Selecting single column

- ✓ We can select single or multiple rows by using iloc indexer.
  - If we select **one row or column** then it returns a Series.

Program      First **row** of the dataframe  
Name           demo9.py  
Input file    sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
s = df1.iloc[0]

print(s)
```

#### Output

Order id	192837
Customer name	Partha
Customer id	8
Product name	Apple iPad 10.2-inch
Product cost	59000
Name:	0, dtype: object

## Data Science - Pandas – iLOC and LOC

**Program Name** Second row of the dataframe  
demo10.py

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
s = df1.iloc[1]

print(s)
```

**Output**

```
Order id          192838
Customer name    Vinay
Customer id       10
Product name     Flatscreen TV
Product cost      65999
Name: 1, dtype: object
```

## Data Science - Pandas – iLOC and LOC

**Program** Last row of the dataframe

**Name** demo11.py

**Input file** sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
s = df1.iloc[-1]

print(s)
```

**Output**

```
Order id          193736
Customer name    Madhurima
Customer id       7
Product name     27in FHD Monitor
Product cost      55999
Name: 899, dtype: object
```

## Data Science - Pandas – iLOC and LOC

**Program** First **row** of the dataframe

**Name** demo12.py

**Input file** sales2.csv

```
import pandas as pd

df = pd.read_csv('sales2.csv')
s = df.iloc["Order id"]

print(s)
```

**Output**

**TypeError:** Cannot index by location index with a non-integer key

## Data Science - Pandas – iLOC and LOC

**Program** First **column** of the dataframe

**Name** demo13.py

**Input file** sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
df2 = df1.iloc[:, 0]

print(df2)
```

**Output**

```
0      192837
1      192838
2      192839
3      192840
4      192841
...
895    193732
896    193733
897    193734
898    193735
899    193736
Name: Order id, Length: 900, dtype: int64
```

## Data Science - Pandas – iLOC and LOC

**Program Name** Second **column** of the dataframe  
**Input file** demo14.py  
sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
df2 = df1.iloc[:, 1]

print(df2)
```

### Output

```
0      Partha
1      Vinay
2    Lavanya
3      Mohan
4      Kedar
...
895    Balaji
896  Neelima
897    Siddhu
898    Vinay
899  Madhurima
Name: Customer name, Length: 900, dtype: object
```

## Data Science - Pandas – iLOC and LOC

**Program** Last column of the dataframe

**Name** demo15.py

**Input file** sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
df2 = df1.iloc[:, -1]

print(df2)
```

**Output**

```
0      59000
1      65999
2      75000
3      55000
4      59000
...
895    60000
896    75999
897    69999
898    69999
899    55999
Name: Product cost, Length: 900, dtype: int64
```

#### 5.4. Selecting multiple rows and columns

- ✓ We can select single or multiple rows by using iloc indexer.
  - If we select **multiple rows and columns** by using iloc indexer then returns a DataFrame.

**Program** First five rows from dataframe

**Name** demo16.py

**Input file** sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
df2 = df1.iloc[0:5]

print(df2)
```

**Output**

	Order id	Customer name	Customer id	Product name	Product cost
0	192837	Partha	8	Apple iPad 10.2-inch	59000
1	192838	Vinay	10	Flatscreen TV	65999
2	192839	Lavanya	16	20in Monitor	75000
3	192840	Mohan	24	Bose SoundSport Headphones	55000
4	192841	Kedar	2	Google Phone	59000

## Data Science - Pandas – iLOC and LOC

**Program Name** First two columns of the dataframe with all rows  
**Input file** demo17.py  
sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
df2 = df1.iloc[:, 0:2]

print(df2)
```

### Output

```
      Order id Customer name
0      192837      Partha
1      192838      Vinay
2      192839      Lavanya
3      192840      Mohan
4      192841      Kedar
..        ...
895    193732      Balaji
896    193733      Neelima
897    193734      Siddhu
898    193735      Vinay
899    193736      Madhurima

[900 rows x 2 columns]
```

## Data Science - Pandas – iLOC and LOC

**Program Name** First three columns of the dataframe with all rows  
**Input file** demo18.py  
sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
df2 = df1.iloc[:, 0:3]

print(df2)
```

### Output

```
   Order id Customer name Customer id
0      192837        Partha             8
1      192838         Vinay            10
2      192839       Lavanya            16
3      192840        Mohan            24
4      192841        Kedar              2
..        ...
895     193732       Balaji            12
896     193733     Neelima            19
897     193734       Siddhu            18
898     193735         Vinay            10
899     193736    Madhurima             7

[900 rows x 3 columns]
```

## Data Science - Pandas – iLOC and LOC

---

**Program Name** first 5 rows and first 2 columns of the DataFrame  
**Input file** demo19.py  
sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
df2 = df1.iloc[0:5, 0:2]

print(df1)
print()
print(df2)
```

### Output

	Order id	Customer name	Customer id	Product name	Product cost
0	192837	Partha	8	Apple iPad 10.2-inch	59000
1	192838	Vinay	10	Flatscreen TV	65999
2	192839	Lavanya	16	20in Monitor	75000
3	192840	Mohan	24	Bose SoundSport Headphones	55000
4	192841	Kedar	2	Google Phone	59000
..	...	...	...	...	...
895	193732	Balaji	12	Samsung Galaxy S20	60000
896	193733	Neelima	19	27in 4K Gaming Monitor	75999
897	193734	Siddhu	18	Google Phone	69999
898	193735	Vinay	10	20in Monitor	69999
899	193736	Madhurima	7	27in FHD Monitor	55999

[900 rows x 5 columns]

	Order id	Customer name
0	192837	Partha
1	192838	Vinay
2	192839	Lavanya
3	192840	Mohan
4	192841	Kedar

## Data Science - Pandas – iLOC and LOC

**Program Name** first 5 rows and first 3 columns of the DataFrame  
**Input file** demo20.py  
sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
df2 = df1.iloc[0:5, 0:3]

print(df1)
print()
print(df2)
```

### Output

	Order id	Customer name	...	Product name	Product cost
0	192837	Partha	...	Apple iPad 10.2-inch	59000
1	192838	Vinay	...	Flatscreen TV	65999
2	192839	Lavanya	...	20in Monitor	75000
3	192840	Mohan	...	Bose SoundSport Headphones	55000
4	192841	Kedar	...	Google Phone	59000
..	...	...	...	...	...
895	193732	Balaji	...	Samsung Galaxy S20	60000
896	193733	Neelima	...	27in 4K Gaming Monitor	75999
897	193734	Siddhu	...	Google Phone	69999
898	193735	Vinay	...	20in Monitor	69999
899	193736	Madhurima	...	27in FHD Monitor	55999

[900 rows x 5 columns]

	Order id	Customer name	Customer id
0	192837	Partha	8
1	192838	Vinay	10
2	192839	Lavanya	16
3	192840	Mohan	24
4	192841	Kedar	2

### 6. loc[] indexer

- ✓ This indexer used to get rows or columns with particular labels from the index.
- ✓ The loc indexer is label based data selection means we have to pass the name of the row or column which we want to select.
- ✓ The loc indexer can also do boolean selection
- ✓ This includes the last element in DataFrame

#### Usage

- ✓ We can use loc[] indexer for two purposes,
  - Selecting rows by label/index
  - Selecting rows with a boolean/conditional lookup

#### Syntax

- ✓ `df.loc[<row selection>, <column selection>]`

### 6.1. loc[] - Selecting rows by label/index

- ✓ We can set index for the DataFrame by using set\_index(p) method.
- ✓ The loc[] indexer directly selects based on index values of any rows.
  - For example, if we set index as Product then we can select the specific product directly.

### 6.2. Creating a dataframe

- ✓ If we load a csv file in pandas then it returns DataFrame

**Program** Loading csv file

**Name** demo21.py

**Input file** sales2.csv

```
import pandas as pd

df1 = pd.read_csv("sales2.csv")
print(df1)
```

**Output**

	Order id	Customer name	Customer id	Product name	Product cost
0	192837	Partha	8	Apple iPad 10.2-inch	59000
1	192838	Vinay	10	Flatscreen TV	65999
2	192839	Lavanya	16	20in Monitor	75000
3	192840	Mohan	24	Bose SoundSport Headphones	55000
4	192841	Kedar	2	Google Phone	59000
..	..	..	..	..	..
895	193732	Balaji	12	Samsung Galaxy S20	60000
896	193733	Neelima	19	27in 4K Gaming Monitor	75999
897	193734	Sidhu	18	Google Phone	69999
898	193735	Vinay	10	20in Monitor	69999
899	193736	Madhurima	7	27in FHD Monitor	55999
[900 rows x 5 columns]					

## Data Science - Pandas – iLOC and LOC

---

**Program Name** Setting index as a Product name and selecting DataFrame  
**Input file** demo22.py  
sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
df1.set_index("Product name", inplace = True)

print(df1)
```

### Output

	Order id	Customer name	Customer id	Product cost
Product name				
Apple iPad 10.2-inch	192837	Partha	8	59000
Flatscreen TV	192838	Vinay	10	65999
20in Monitor	192839	Lavanya	16	75000
Bose SoundSport Headphones	192840	Mohan	24	55000
Google Phone	192841	Kedar	2	59000
...	...	...	...	...
Samsung Galaxy S20	193732	Balaji	12	60000
27in 4K Gaming Monitor	193733	Neelima	19	75999
Google Phone	193734	Siddhu	18	69999
20in Monitor	193735	Vinay	10	69999
27in FHD Monitor	193736	Madhurima	7	55999
[900 rows x 4 columns]				

## Data Science - Pandas – iLOC and LOC

---

**Program Name** Setting index as a Product name and selecting Product  
**demo23.py**  
**Input file** sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
df1.set_index("Product name", inplace =True)

a = 'iPhone 9'
df2 = df1.loc[a]
print(df2.head(10))
```

### Output

Product name	Order id	Customer name	Customer id	Product cost
iPhone 9	192861	Sagar	6	60000
iPhone 9	192867	Shafi	3	65000
iPhone 9	192891	Sagar	6	50000
iPhone 9	192902	Balaji	12	69999
iPhone 9	192914	Jaya Chandra	21	59000
iPhone 9	192921	Venki	15	50000
iPhone 9	192939	Tarun	11	55999
iPhone 9	192969	Siddhu	18	65000
iPhone 9	192984	Neelima	19	69000
iPhone 9	192998	Vinay	10	75999

## Data Science - Pandas – iLOC and LOC

---

**Program Name** Setting index as a Product name and selecting Product  
**Input file** demo24.py  
sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
df1.set_index("Product name", inplace = True)

a = 'ThinkPad Laptop'
df2 = df1.loc[a]
print(df2.head(10))
```

### Output

Product name	Order id	Customer name	Customer id	Product cost
ThinkPad Laptop	192859	Vijay	9	75000
ThinkPad Laptop	192862	Harsha	5	55000
ThinkPad Laptop	192868	Balaji	12	75999
ThinkPad Laptop	192888	Jaya Chandra	21	75000
ThinkPad Laptop	192915	Balaji	12	50000
ThinkPad Laptop	192974	Siddhu	18	69000
ThinkPad Laptop	192983	Sagar	6	63999
ThinkPad Laptop	192989	Venki	15	65000
ThinkPad Laptop	192999	Mallikarjun	13	69999
ThinkPad Laptop	193014	Sumanth	22	55999

## Data Science - Pandas – iLOC and LOC

---

**Program Name** Setting index as a Product name and selecting two products  
**Input file** demo25.py  
sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
df1.set_index("Product name", inplace = True)

a = ['iPhone 9', 'iPhone 11']
df2 = df1.loc[a]
print(df2)
```

### Output

Product name	Order id	Customer name	Customer id	Product cost
iPhone 9	192861	Sagar	6	60000
iPhone 9	192867	Shafi	3	65000
iPhone 9	192891	Sagar	6	50000
iPhone 9	192902	Balaji	12	69999
iPhone 9	192914	Jaya Chandra	21	59000
...	...	...	...	...
iPhone 11	193615	Venki	15	69999
iPhone 11	193700	Venki	15	63999
iPhone 11	193706	Madhurima	7	60000
iPhone 11	193716	Harsha	5	69000
iPhone 11	193727	Nireekshan	1	59000

[90 rows x 4 columns]

## Data Science - Pandas – iLOC and LOC

**Program Name** Setting index as a Product name and selecting two products  
**Input file** demo26.py  
sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
df1.set_index("Product name", inplace = True)

a = ['ThinkPad Laptop', '27in FHD Monitor']
df2 = df1.loc[a]
print(df2)
```

### Output

	Order id	Customer name	Customer id	Product cost
Product name				
ThinkPad Laptop	192859	Vijay	9	75000
ThinkPad Laptop	192862	Harsha	5	55000
ThinkPad Laptop	192868	Balaji	12	75999
ThinkPad Laptop	192888	Jaya Chandra	21	75000
ThinkPad Laptop	192915	Balaji	12	50000
...	...	...	...	...
27in FHD Monitor	193587	Jaya Chandra	21	59000
27in FHD Monitor	193631	Karteek	4	55000
27in FHD Monitor	193679	Jaya Chandra	21	63000
27in FHD Monitor	193680	Madhurima	7	63000
27in FHD Monitor	193736	Madhurima	7	55999

[71 rows x 4 columns]

## Data Science - Pandas – iLOC and LOC

---

### Program

Setting index as a Product name and selecting a couple of product names with Product cost and Customer id

Name demo27.py  
Input file sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
df1.set_index("Product name", inplace = True)

a = ['iPhone 9', 'ThinkPad Laptop']
b = ['Product cost', 'Customer id']

df2 = df1.loc[a, b]
print(df2)
```

### Output

	Product name	cost	Customer id
iPhone 9	60000	6	
iPhone 9	65000	3	
iPhone 9	50000	6	
iPhone 9	69999	12	
iPhone 9	59000	21	
...	...	...	
ThinkPad Laptop	63000	17	
ThinkPad Laptop	50000	18	
ThinkPad Laptop	63000	13	
ThinkPad Laptop	65999	15	
ThinkPad Laptop	59000	17	
[82 rows x 2 columns]			

**Program**

Setting index as a Product name and selecting a couple of product names with Product cost and Customer name

**Name** demo28.py  
**Input file** sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
df1.set_index("Product name", inplace = True)

a = ['iPhone 9', 'ThinkPad Laptop']
b = ['Product cost', 'Customer name']

df2 = df1.loc[a, b]

print(df2)
```

**Output**

	Product cost	Customer name
Product name		
iPhone 9	60000	Sagar
iPhone 9	65000	Shafi
iPhone 9	50000	Sagar
iPhone 9	69999	Balaji
iPhone 9	59000	Jaya Chandra
...	...	...
ThinkPad Laptop	63000	Pradhan
ThinkPad Laptop	50000	Siddhu
ThinkPad Laptop	63000	Mallikarjun
ThinkPad Laptop	65999	Venki
ThinkPad Laptop	59000	Pradhan

[82 rows x 2 columns]

## Data Science - Pandas – iLOC and LOC

### Program

Setting index as a Product name and selecting a couple of product names with all columns between Order id and Product cost

Name demo29.py  
 Input file sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
df1.set_index("Product name", inplace = True)

a = ['iPhone 8', 'Google Phone']
df2 = df1.loc[a, 'Order id' : 'Product cost']

print(df2)
```

### Output

	Order id	Customer name	Customer id	Product cost
Product name				
iPhone 8	192865	Siddhu	18	59000
iPhone 8	192874	Sumanth	22	69999
iPhone 8	192878	Jaya Chandra	21	69000
iPhone 8	192881	Lavanya	16	65000
iPhone 8	192897	Chaithanya	14	69999
...	...	...	...	...
Google Phone	193666	Karteek	4	50000
Google Phone	193672	Karteek	4	75000
Google Phone	193725	Mallikarjun	13	75999
Google Phone	193729	Chaithanya	14	65000
Google Phone	193734	Siddhu	18	69999
[93 rows x 4 columns]				

**Program**

Setting index as a Product name and selecting a couple of product names with all columns between Order id and Customer id

**Name** demo30.py  
**Input file** sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
df1.set_index("Product name", inplace = True)

a = ['iPhone 8', 'Google Phone']
df2 = df1.loc[a, 'Order id' : 'Customer id']

print(df2)
```

**Output**

Product name	Order id	Customer name	Customer id
iPhone 8	192865	Siddhu	18
iPhone 8	192874	Sumanth	22
iPhone 8	192878	Jaya Chandra	21
iPhone 8	192881	Lavanya	16
iPhone 8	192897	Chaithanya	14
...	...	...	...
Google Phone	193666	Karteek	4
Google Phone	193672	Karteek	4
Google Phone	193725	Mallikarjun	13
Google Phone	193729	Chaithanya	14
Google Phone	193734	Siddhu	18
[93 rows x 3 columns]			

### 6.3. Boolean / Logical indexing

- ✓ Based on conditions we can get rows and columns from Dataframe
- ✓ This is the most commonly used data analysis
- ✓ In most use cases, you will make selections based on the values of different columns in your data set.

#### Program

Select rows with specific product with all columns between Order id and Product cost

Name demo31.py  
Input file sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')

a = df1['Product name'] == 'LG Washing Machine'

df2 = df1.loc[a]

print(df2.head())
```

#### Output

	Order id	Customer name	Customer id	Product name	Product cost
5	192842	Karteek	4	LG Washing Machine	75000
70	192907	Balaji	12	LG Washing Machine	65999
111	192948	Sagar	6	LG Washing Machine	63000
117	192954	Sagar	6	LG Washing Machine	61000
173	193010	Madhurima	7	LG Washing Machine	65000

## Data Science - Pandas – iLOC and LOC

### Program

Select rows with specific product with all columns between Order id and Product cost

Name demo32.py  
Input file sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')

a = df1['Product name'] == 'LG Washing Machine'

df2 = df1.loc[a, 'Order id' : 'Product cost']

print(df2.head())
```

### Output

	Order id	Customer name	Customer id	Product name	Product cost
5	192842	Karteek	4	LG Washing Machine	75000
70	192907	Balaji	12	LG Washing Machine	65999
111	192948	Sagar	6	LG Washing Machine	63000
117	192954	Sagar	6	LG Washing Machine	61000
173	193010	Madhurima	7	LG Washing Machine	65000

## Data Science - Pandas – iLOC and LOC

### Program

Select rows with specific Customer with Product name and Product cost columns

Name demo33.py  
Input file sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')

a = df1['Customer name'] == 'Sagar'

df2 = df1.loc[a, 'Product name' : 'Product cost']

print(df2.head())
```

### Output

	Product name	Product cost
24	iPhone 9	60000
50	Google Phone	60000
54	iPhone 9	50000
111	LG Washing Machine	63000
117	LG Washing Machine	61000

## Data Science – Pandas – DataFrame - Filtering

---

### 14. PANDAS – DataFrame – Filtering

#### Contents

<b>1. Filtering</b> .....	2
1.1. Filtering Examples .....	2
1.2. We can filter data by using .....	3
1.3. Creating a DataFrame .....	3
<b>2. By using relational operators</b> .....	4
<b>3. Filtering by using loc and iloc</b> .....	9
3.1. Rows position and column name.....	11
3.2. Selecting multiple values of a column .....	14
3.3. isin() method.....	15
3.4. unique() method .....	17
<b>4. Select Non-Missing Data from DataFrame</b> .....	19

**14. PANDAS – DataFrame - Filtering****1. Filtering**

- ✓ Filtering the data in DataFrame is very common requirement.
- ✓ It is very important step in Data Analysis project
- ✓ Based on condition we can filter the data from DataFrame

**1.1. Filtering Examples**

- ✓ Banking
  - Select all the active customers whose accounts were opened after 1st January 2020
  - Get the details of all the customers who made more than 300 transactions in the last 6 months
- ✓ Organization
  - Fetch information of employees who spent more than 3 years in the organization and received highest rating in the past 2 years
- ✓ Telecom
  - Analyse complaints data and identify customers who filed more than 5 complaints in the last 1 year
- ✓ General
  - Extract details of metro cities where per capita income is greater than 40K dollars
- ✓ Many more...

## Data Science – Pandas – DataFrame - Filtering

### 1.2. We can filter data by using

- ✓ By using relational operators.
  - Single condition
  - Multiple condition
- ✓ By using loc & iloc indexers

### 1.3. Creating a DataFrame

- ✓ We can create DataFrame by loading csv file.

**Program Name** Loading csv file  
**Input file** demo1.py  
 sales4.csv

```
import pandas as pd

df = pd.read_csv("sales4.csv")

print(df)
```

#### Output

	Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost
0	1023	Venki	15	27in FHD Monitor	59000
1	1024	Chaithanya	14	iPhone 11	69000
2	1025	Shahid	28	Bose SoundSport Headphones	65999
3	1026	Veeru	3	Apple iPad 10.2-inch	63999
4	1027	Venu	23	Google Phone	63999
...	...	...	...	...	...
299995	301018	Karteek	4	Apple iPad 10.2-inch	51999
299996	301019	Veeru	3	Macbook Pro Laptop	51999
299997	301020	Harsha	5	LG Washing Machine	65000
299998	301021	Nireekshan	1	LG Mobile	60000
299999	301022	Pradhan	17	34in Ultrawide Monitor	55000

[300000 rows x 5 columns]

## Data Science – Pandas – DataFrame - Filtering

---

### 2. By using relational operators

- ✓ By using relational operators we can apply the filter on dataframe.

**Program Name** Filtering DataFrame by using relational operator: Single condition  
**demo2.py**  
**Input file** sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")

con1 = df1['Product_Cost'] > 65000
df2 = df1[con1]

print(df2)
```

### Output

	Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost
1	1024	Chaitanya	14	iPhone 11	69000
2	1025	Shahid	20	Bose SoundSport Headphones	65999
11	1034	Tarun	11	Samsung Galaxy S20	69999
13	1036	Chaitanya	14	LG ThinQ Refrigerator	69999
17	1040	Sumanth	22	iPhone 8	65999
...	...	...	...	...	...
299982	301005	Tarun	11	Samsung Galaxy S20	69000
299983	301006	Shafi	25	27in 4K Gaming Monitor	69000
299985	301008	Venki	15	Apple Airpods Headphones	65999
299988	301011	Venki	15	Google Phone	65999
299990	301013	Balaji	12	iPhone 11	65999

[112280 rows x 5 columns]

## Data Science – Pandas – DataFrame - Filtering

---

**Program Name** Filtering DataFrame by using relational operator: Single condition  
**demo3.py**  
**Input file** sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")

con1 = df1['Product_Cost'] > 70000
df2 = df1[con1]

print(df2)
```

### Output

	Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost
25	1048	Partha	8	Samsung Galaxy S9 Plus	75000
28	1051	Lavanya	16	LG Washing Machine	75000
42	1065	Madhurima	7	20in Monitor	75999
45	1068	Vinay	10	Samsung Galaxy S9 Plus	75999
53	1076	Tarun	11	LG Washing Machine	75999
...	...	...	...	...	...
299955	300978	Siddhu	18	34in Ultrawide Monitor	75000
299963	300986	Shahid	20	34in Ultrawide Monitor	75999
299964	300987	Harsha	5	Macbook Pro Laptop	75000
299965	300988	Madhurima	7	Flatscreen TV	75000
299973	300996	Vinay	10	Samsung Galaxy S9 Plus	75999

[37401 rows x 5 columns]

## Data Science – Pandas – DataFrame - Filtering

---

### Program

Filtering DataFrame by using relational operator: Multiple conditions

Name demo4.py  
Input file sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")

con1 = df1['Product_Cost'] > 50000
con2 = df1['Product_Cost'] < 60000

df2 = df1[con1 & con2]

print(df2)
```

### Output

	Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost
0	1023	Venki	15	27in FHD Monitor	59000
8	1031	Kedar	2	20in Monitor	55999
12	1035	Kedar	2	Google Phone	55999
14	1037	Nireekshan	1	Apple Airpods Headphones	55999
22	1045	Lavanya	16	iPhone 7s	51999
...	...	...	...	...	...
299993	301016	Siddhu	18	iPhone 9	59000
299994	301017	Kedar	2	ThinkPad Laptop	55999
299995	301018	Karteek	4	Apple iPad 10.2-inch	51999
299996	301019	Veeru	3	Macbook Pro Laptop	51999
299999	301022	Pradhan	17	34in Ultrawide Monitor	55000

[74794 rows x 5 columns]

## Data Science – Pandas – DataFrame - Filtering

---

### Program

Filtering DataFrame by using relational operator: Multiple conditions

Name demo5.py  
Input file sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")

con1 = df1.Product_Name == "iPhone 11"
con2 = df1.Customer_Name == "Nireekshan"

df2 = df1[con1 & con2]

print(df2)
```

### Output

	Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost
821	1844	Nireekshan	1	iPhone 11	63999
1086	2109	Nireekshan	1	iPhone 11	65000
1529	2552	Nireekshan	1	iPhone 11	55000
1539	2562	Nireekshan	1	iPhone 11	61000
1676	2699	Nireekshan	1	iPhone 11	65000
...	...	...	...	...	...
296768	297791	Nireekshan	1	iPhone 11	55000
297335	298358	Nireekshan	1	iPhone 11	63999
297717	298740	Nireekshan	1	iPhone 11	50000
297766	298789	Nireekshan	1	iPhone 11	55999
298524	299547	Nireekshan	1	iPhone 11	65999
[581 rows x 5 columns]					

## Data Science – Pandas – DataFrame - Filtering

---

### Program

Filtering DataFrame by using relational operator: Multiple conditions

Name demo6.py  
Input file sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")

con1 = df1.Product_Name == "iPhone 11"
con2 = df1.Customer_Name == "Shahid"

df2 = df1[con1 & con2]

print(df2)
```

### Output

	Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost
26	1049	Shahid	20	iPhone 11	65999
783	1806	Shahid	20	iPhone 11	69000
1260	2283	Shahid	20	iPhone 11	69000
1834	2857	Shahid	20	iPhone 11	65999
1848	2871	Shahid	20	iPhone 11	69999
...	...	...	...	...	...
298667	299690	Shahid	20	iPhone 11	65999
298969	299992	Shahid	20	iPhone 11	69000
299206	300229	Shahid	20	iPhone 11	65000
299691	300714	Shahid	20	iPhone 11	63999
299950	300973	Shahid	20	iPhone 11	75000

[594 rows x 5 columns]

## Data Science – Pandas – DataFrame - Filtering

---

### 3. Filtering by using loc and iloc

- ✓ We can filter the dataframe by using loc and iloc indexers as well

**Program Name** Filtering DataFrame by using loc indexer  
**Input file** demo7.py  
sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")

con1 = df1.Product_Name == "iPhone 11"
con2 = df1.Customer_Name == "Shahid"

df2 = df1.loc[con1 & con2]

print(df2)
```

### Output

	Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost
26	1049	Shahid	20	iPhone 11	65999
783	1806	Shahid	20	iPhone 11	69000
1260	2283	Shahid	20	iPhone 11	69000
1834	2857	Shahid	20	iPhone 11	65999
1848	2871	Shahid	20	iPhone 11	69999
...	...	...	...	...	...
298667	299690	Shahid	20	iPhone 11	65999
298969	299992	Shahid	20	iPhone 11	69000
299206	300229	Shahid	20	iPhone 11	65000
299691	300714	Shahid	20	iPhone 11	63999
299950	300973	Shahid	20	iPhone 11	75000

[594 rows x 5 columns]

## Data Science – Pandas – DataFrame - Filtering

**Program Name** Filtering DataFrame by using iloc indexer  
**Input file** demo8.py  
sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")

df2 = df1.iloc[:5, ]

print(df2)
```

### Output

	Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost
0	1023	Venki	15	27in FHD Monitor	59000
1	1024	Chaithanya	14	iPhone 11	69000
2	1025	Shahid	20	Bose SoundSport Headphones	65999
3	1026	Veeru	3	Apple iPad 10.2-inch	63999
4	1027	Venu	23	Google Phone	63999

## Data Science – Pandas – DataFrame - Filtering

---

### 3.1. Rows position and column name

- ✓ We can even select the dataframe by providing rows position and column name

**Program Name** Filtering DataFrame by using loc indexer  
**demo9.py**  
**Input file** sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")

rows = df1.index[0:]
cols = ["Product_Name", "Customer_Id"]

df2 = df1.loc[rows, cols]

print(df2)
```

### Output

	Product_Name	Customer_Id
0	27in FHD Monitor	15
1	iPhone 11	14
2	Bose SoundSport Headphones	20
3	Apple iPad 10.2-inch	3
4	Google Phone	23
...	...	...
299995	Apple iPad 10.2-inch	4
299996	Macbook Pro Laptop	3
299997	LG Washing Machine	5
299998	LG Mobile	1
299999	34in Ultrawide Monitor	17
[300000 rows x 2 columns]		

## Data Science – Pandas – DataFrame - Filtering

**Program Name** Filtering DataFrame by using loc indexer  
**Input file** demo10.py  
sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")

rows = df1.index[0:4]
cols = ["Product_Name", "Customer_Id"]

df2 = df1.loc[rows, cols]

print(df2)
```

### Output

	Product_Name	Customer_Id
0	27in FHD Monitor	15
1	iPhone 11	14
2	Bose SoundSport Headphones	20
3	Apple iPad 10.2-inch	3

## Data Science – Pandas – DataFrame - Filtering

**Program Name** Filtering DataFrame by using loc indexer  
**Input file** demo11.py  
sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")

rows = df1.index[5:]
cols = ["Product_Name", "Customer_Id"]

df2 = df1.loc[rows, cols]

print(df2)
```

### Output

```
          Product_Name Customer_Id
5      Samsung Galaxy S9 Plus        6
6                  iPhone 11       23
7      27in FHD Monitor        25
8      20in Monitor            2
9      LG Washing Machine       19
...
299995    Apple iPad 10.2-inch        4
299996      Macbook Pro Laptop        3
299997      LG Washing Machine        5
299998          LG Mobile            1
299999  34in Ultrawide Monitor       17

[299995 rows x 2 columns]
```

## Data Science – Pandas – DataFrame - Filtering

### 3.2. Selecting multiple values of a column

- ✓ We can filter dataframe by providing multiple values of a column

**Program Name** Filtering DataFrame by using loc indexer  
**demo12.py**  
**Input file** sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")

a = df1.Product_Name == "LG Washing Machine"
b = df1.Customer_Id == 1
c = a | b

df2 = df1.loc[c]

print(df2)
```

### Output

	Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost
9	1032	Neelima	19	LG Washing Machine	63000
14	1037	Nireekshan	1	Apple Airpods Headphones	55999
24	1047	Shafi	25	LG Washing Machine	63999
28	1051	Lavanya	16	LG Washing Machine	75000
39	1062	Tarun	11	LG Washing Machine	61000
...	...	...	...	...	...
299936	300959	Partha	8	LG Washing Machine	65999
299937	300960	Nireekshan	1	27in FHD Monitor	60000
299970	300993	Nireekshan	1	iPhone 8	60000
299997	301020	Harsha	5	LG Washing Machine	65000
299998	301021	Nireekshan	1	LG Mobile	60000

[26278 rows x 5 columns]

## Data Science – Pandas – DataFrame - Filtering

### 3.3. isin() method

- ✓ isin() is predefined method in Series class.
- ✓ We should access this method by using Series object.
- ✓ By using this method we can select data from DataFrame

**Program** Filtering DataFrame by using loc isin() method

**Name** demo13.py

**Input file** sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")

a = ["Macbook Pro Laptop"]

b = df1.Product_Name.isin(a)

df2 = df1[b]

print(df2)
```

### Output

	Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost
23	1046	Madhurima	7	Macbook Pro Laptop	50000
49	1072	Balaji	12	Macbook Pro Laptop	61000
56	1079	Tarun	11	Macbook Pro Laptop	75000
60	1083	Tarun	11	Macbook Pro Laptop	65999
85	1108	Vijay	9	Macbook Pro Laptop	59000
...	...	...	...	...	...
299906	300929	Partha	8	Macbook Pro Laptop	51999
299907	300930	Neelima	19	Macbook Pro Laptop	63000
299964	300987	Harsha	5	Macbook Pro Laptop	75000
299974	300997	Chaithanya	14	Macbook Pro Laptop	65999
299996	301019	Veenu	3	Macbook Pro Laptop	51999
[15144 rows x 5 columns]					

## Data Science – Pandas – DataFrame - Filtering

**Program Name** Filtering DataFrame by using loc isin() method  
**demo14.py**  
**Input file** sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")

a = ["34in Ultrawide Monitor", "Macbook Pro Laptop"]

b = df1.Product_Name.isin(a)

df2 = df1[b]

print(df2)
```

### Output

	Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost
23	1046	Madhurima	7	Macbook Pro Laptop	50000
49	1072	Balaji	12	Macbook Pro Laptop	61000
56	1079	Tarun	11	Macbook Pro Laptop	75000
60	1083	Tarun	11	Macbook Pro Laptop	65999
71	1094	Shahid	20	34in Ultrawide Monitor	61000
...	...	...	...	...	...
299964	300987	Harsha	5	Macbook Pro Laptop	75000
299969	300992	Tarun	11	34in Ultrawide Monitor	65999
299974	300997	Chaithanya	14	Macbook Pro Laptop	65999
299996	301019	Veeru	3	Macbook Pro Laptop	51999
299999	301022	Pradhan	17	34in Ultrawide Monitor	55000

[30299 rows x 5 columns]

## Data Science – Pandas – DataFrame - Filtering

### 3.4. unique() function

- ✓ unique() is predefined function in pandas.
- ✓ We should access this function by using pandas module.
- ✓ This function returns the unique values from the column.

**Program** Selecting unique column values

**Name** demo15.py

**Input file** sales4.csv

```
import pandas as pd

df = pd.read_csv("sales4.csv")

a = pd.unique(df.Product_Name)

print(a)
print(len(a))
```

### Output

```
['27in FHD Monitor' 'iPhone 11' 'Bose SoundSport Headphones'
 'Apple iPad 10.2-inch' 'Google Phone' 'Samsung Galaxy S9 Plus'
 '20in Monitor' 'LG Washing Machine' 'iPhone 7s' 'Samsung Galaxy S20'
 'LG ThinQ Refrigerator' 'Apple Airpods Headphones' 'iPhone 8'
 'Macbook Pro Laptop' 'LG Mobile' 'ThinkPad Laptop' 'Flatscreen TV'
 '34in Ultrawide Monitor' 'iPhone 9' '27in 4K Gaming Monitor']
20
```

## Data Science – Pandas – DataFrame - Filtering

**Program Name** Selecting unique column values  
**Input file** demo16.py  
sales4.csv

```
import pandas as pd

df = pd.read_csv("sales4.csv")

a = pd.unique(df.Customer_Name)

print(a)
print(len(a))
```

### Output

```
['Venki' 'Chaithanya' 'Shahid' 'Veeru' 'Venu' 'Daniel' 'Shafi' 'Kedar'
 'Neelima' 'Vijay' 'Tarun' 'Nireekshan' 'Karteek' 'Sumanth' 'Mallikarjun'
 'Vinay' 'Lavanya' 'Madhurima' 'Partha' 'Siddhu' 'Jaya Chandra' 'Balaji'
 'Pradhan' 'Harsha' 'Mohan']
25
```

#### 4. Select Non-Missing Data from DataFrame

##### notnull() method

- ✓ notnull() is predefined method in Series class.
- ✓ We should access this method by using Series object
- ✓ By using this function we can select the DataFrame which having non NaN values

**Program Name** Creating a DataFrame  
demo17.py

```
import pandas as pd
import numpy as np

data = [
    ['Shahid', 21, 40000],
    ['Nireekshan', 22, 20000],
    ['Veeru', 45, 90000],
    ['Sumanth', 20, 95000],
    [np.nan, 2, 99000],
    ['Prasad', 1, 41000]
]

c = ['Name', 'Age', 'Salary']

df1 = pd.DataFrame(data, columns = c)

print(df1)
```

##### Output

	Name	Age	Salary
0	Shahid	21	40000
1	Nireekshan	22	20000
2	Veeru	45	90000
3	Sumanth	20	95000
4	NaN	2	99000
5	Prasad	1	41000

## Data Science – Pandas – DataFrame - Filtering

---

**Program Name**      notnull() method  
demo18.py

```
import pandas as pd
import numpy as np

data = [
    ['Shahid', 21, 40000],
    ['Nireekshan', 22, 20000],
    ['Veeru', 45, 90000],
    ['Sumanth', 20, 95000],
    [np.nan, 2, 99000],
    ['Prasad', 1, 41000]
]

c = ['Name', 'Age', 'Salary']

df1 = pd.DataFrame(data, columns = c)

d = df1.Name.notnull()

df2 = df1[d]

print(df1)
print()
print(df2)
```

### Output

	Name	Age	Salary
0	Shahid	21	40000
1	Nireekshan	22	20000
2	Veeru	45	90000
3	Sumanth	20	95000
4	NaN	2	99000
5	Prasad	1	41000

	Name	Age	Salary
0	Shahid	21	40000
1	Nireekshan	22	20000
2	Veeru	45	90000
3	Sumanth	20	95000
5	Prasad	1	41000

**15. Pandas – DataFrame – Sorting****Contents**

<b>1. Sorting.....</b>	<b>2</b>
1.1. sort_values(by="column name") .....	3
1.2. sort_index().....	8

## 15. Pandas – DataFrame – Sorting

### 1. Sorting

- ✓ DataFrame contains group of rows, column, index and values.
- ✓ Based on requirement we can applying sorting on column name, index etc.

#### Creating a dataframe:

- ✓ If we load a csv file in pandas then it returns dataframe

**Program** Loading csv file  
**Name** demo1.py  
**Input file** sales4.csv

```

import pandas as pd

df = pd.read_csv("sales4.csv")
print(df)
  
```

#### Output

	Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost
0	1023	Venki	15	27in FHD Monitor	59000
1	1024	Chaitanya	14	iPhone 11	59000
2	1025	Shahid	20	Bose SoundSport Headphones	65999
3	1026	Veeru	3	Apple iPad 10.2-inch	63999
4	1027	Venu	23	Google Phone	63999
...	...	...	...	...	...
299995	301018	Karteek	4	Apple iPad 10.2-inch	51999
299996	301019	Veeru	3	Macbook Pro Laptop	51999
299997	301020	Harsha	5	LG Washing Machine	65000
299998	301021	Nireekshan	1	LG Mobile	50000
299999	301022	Pradhan	17	34in Ultrawide Monitor	55000
[300000 rows x 5 columns]					

## Data Science – Pandas – DataFrame - Sorting

---

### 1.1. sort\_values(by="column name")

- ✓ sort\_values(p) is predefined method in DataFrame class.
- ✓ We should access this method by using DataFrame object.
- ✓ This method sort the values based on column which we specified.
  - Number default sorting is ascending order
  - String default sorting is alphabetical order

**Program**      Sorting dataframe by using sort\_values(p) method  
**Name**           demo2.py  
**Input file**       sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")
df2 = df1.sort_values(by = "Product_Cost")

print(df2)
```

### Output

Order_Id	Customer_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost
52994	54917	Sumanth	22	Samsung Galaxy S9 Plus	50000
122987	124010	Karteek	4	27in FHD Monitor	50000
16661	17684	Venu	23	Google Phone	50000
122986	124009	Harsha	5	iPhone 9	50000
122974	123997	Lavanya	16	34in Ultrawide Monitor	50000
...	...	...	...	...	...
273569	274592	Jaya Chandra	21	Macbook Pro Laptop	75999
273565	274588	Partha	8	Apple Airpods Headphones	75999
171470	172493	Veeru	3	27in FHD Monitor	75999
76102	77125	Lavanya	16	Samsung Galaxy S20	75999
76472	77495	Nireekshan	1	Samsung Galaxy S20	75999

[300000 rows x 5 columns]

## Data Science – Pandas – DataFrame - Sorting

---

**Program Name**      Sorting dataframe by using sort\_values(p) method  
**Input file**      demo3.py  
sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")
df2 = df1.sort_values(by = "Customer_Id")

print(df2)
```

### Output

	Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost
160540	161563	Nireekshan	1	27in FHD Monitor	75000
158658	159681	Nireekshan	1	Macbook Pro Laptop	63999
266895	267918	Nireekshan	1	LG Washing Machine	59000
266908	267931	Nireekshan	1	Google Phone	55000
183395	184418	Nireekshan	1	LG ThinQ Refrigerator	69000
...	...	...	...	...	...
104589	105612	Shafi	25	LG ThinQ Refrigerator	69999
194864	195887	Shafi	25	iPhone 11	63999
223127	224158	Shafi	25	Samsung Galaxy S20	75999
249626	250649	Shafi	25	Macbook Pro Laptop	61000
109653	110676	Shafi	25	Samsung Galaxy S9 Plus	65999
[300000 rows x 5 columns]					

## Data Science – Pandas – DataFrame - Sorting

---

**Program**      Sorting dataframe by using sort\_values(p) method

**Name**           demo4.py

**Input file**    sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")
df2 = df1.sort_values(by = "Customer_Id", ascending = False)

print(df2)
```

### Output

	Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost
225837	226860	Shafi	25	Apple iPad 10.2-inch	69999
58299	59322	shafi	25	ThinkPad Laptop	63999
122755	123778	Shafi	25	Samsung Galaxy S20	61000
220596	221619	Shafi	25	Bose SoundSport Headphones	69999
9252	10275	Shafi	25	Flatscreen TV	60000
...	...	...	...	...	...
191789	192812	Nireekshan	1	Flatscreen TV	69999
146543	147566	Nireekshan	1	iPhone 7s	60000
283770	284793	Nireekshan	1	Apple Airpods Headphones	69000
79317	80340	Nireekshan	1	20in Monitor	69000
78470	79493	Nireekshan	1	iPhone 7s	61000
[300000 rows x 5 columns]					

## Data Science – Pandas – DataFrame - Sorting

---

**Program Name** Sorting dataframe by using sort\_values(p) method

**Input file** demo5.py

sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")
df2 = df1.sort_values(by = "Customer_Id", ascending = 0)

print(df2)
```

### Output

	Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost
225837	226860	Shafi	25	Apple iPad 10.2-inch	69999
58299	59322	Shafi	25	ThinkPad Laptop	63999
122755	123778	Shafi	25	Samsung Galaxy S20	61000
220596	221619	Shafi	25	Bose SoundSport Headphones	69999
9252	10275	Shafi	25	Flatscreen TV	60000
...	...	...	...	...	...
191789	192812	Nireekshan	1	Flatscreen TV	69999
146543	147566	Nireekshan	1	iPhone 7s	60000
283770	284793	Nireekshan	1	Apple Airpods Headphones	69000
79317	80340	Nireekshan	1	20in Monitor	69000
78470	79493	Nireekshan	1	iPhone 7s	61000
[300000 rows x 5 columns]					

## Data Science – Pandas – DataFrame - Sorting

**Program Name** Sorting dataframe by using sort\_values() method  
**Input file** demo6.py  
sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")
df2 = df1.sort_values(by = "Customer_Name")

print(df2)
```

### Output

```
   Order_Id Customer_Name Customer_Id      Product_Name Product_Cost
136196    137219       Balaji        12          LG Mobile     60000
128754    129777       Balaji        12      Samsung Galaxy S20     61000
292113    293136       Balaji        12          LG Mobile     65999
128753    129776       Balaji        12  LG ThinQ Refrigerator     65999
73777     74800        Balaji        12    Apple iPad 10.2-inch     51999
...
280203    281226       Vinay         10          iPhone 8     65999
155180    156203       Vinay         10  Samsung Galaxy S9 Plus     65000
82486     83509        Vinay         10  LG ThinQ Refrigerator     59000
26986     28009        Vinay         10          iPhone 11     59000
186628    187651       Vinay         10  LG ThinQ Refrigerator     51999
[300000 rows x 5 columns]
```

## Data Science – Pandas – DataFrame - Sorting

---

### 1.2. sort\_index()

- ✓ sort\_index() is predefined method in DataFrame class.
- ✓ We should access this method by using DataFrame object.
- ✓ This method sort the indexes in DataFrame

**Program Name**      Sorting dataframe by using sort\_index()  
demo7.py

```
import pandas as pd

d = {
    'Order id': [11, 21, 31],
    'Customer name': ['Kedar', 'Nireekshan', 'Daniel'],
    'Product': ['iPhone 11','hTC', 'macbook']
}

i = [555, 444, 333]

df1 = pd.DataFrame(d, index = i)
df2 = df1.sort_index()

print(df1)
print()
print(df2)
```

### Output

	Order id	Customer name	Product
555	11	Kedar	iPhone 11
444	21	Nireekshan	hTC
333	31	Daniel	macbook

	Order id	Customer name	Product
333	31	Daniel	macbook
444	21	Nireekshan	hTC
555	11	Kedar	iPhone 11

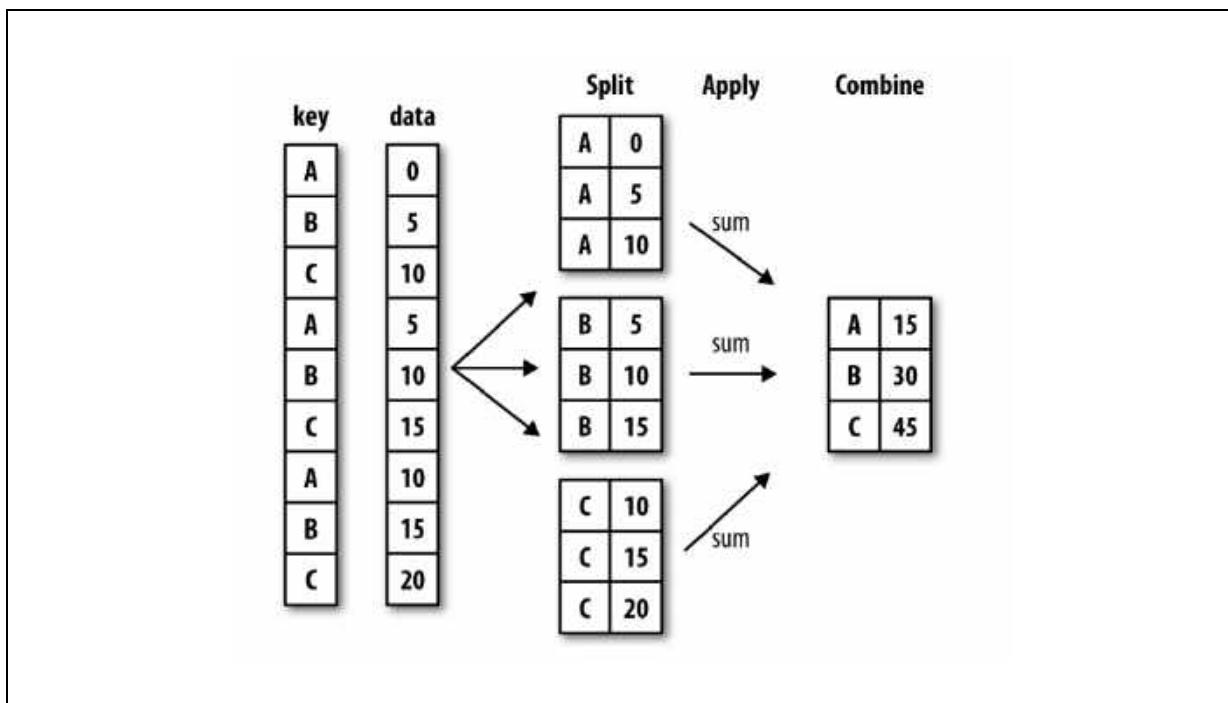
**16. Pandas – DataFrame – Groupby****Contents**

<b>1. Groupby Introduction .....</b>	<b>2</b>
<b>2. groupby(p) .....</b>	<b>3</b>

## 16. Pandas – DataFrame - Groupby

### 1. Groupby Introduction

- ✓ Groupby is very common and important operations in Data analysis,
- ✓ There are mainly 3 steps in Groupby operation,
  - Splitting the data into groups based on some criteria.
  - Applying operations to each group independently.
  - Combining the results.



- ✓ For example,
  - If we apply groupby on Product\_Name then it groups the data into based on name of the product.
- ✓ The groupby(p) method returns a GroupBy object.

## Data Science – Pandas – DataFrame - Groupby

### 2. groupby(p)

- ✓ groupby(p) is predefined method in DataFrame.
- ✓ We should access this method by using DataFrame object.
- ✓ This method returns GroupBy object.

**Program Name** Creating products DataFrame  
demo1.py

```
import pandas as pd

d = {
    "Product": ["Samsung", "Nokia", "Samsung", "Motorola",
    "Nokia", "Samsung", "Samsung"],

    "Orders": [2, 4, 3, 4, 6, 7, 3]
}

df1 = pd.DataFrame(d)

print(df1)
```

### Output

	Product	Orders
0	Samsung	2
1	Nokia	4
2	Samsung	3
3	Motorola	4
4	Nokia	6
5	Samsung	7
6	Samsung	3

## Data Science – Pandas – DataFrame - Groupby

**Program Name**

Creating products DataFrame, applying groupby  
demo2.py

```
import pandas as pd

d = {
    "Product": ["Samsung", "Nokia", "Samsung", "Motorola",
    "Nokia", "Samsung", "Samsung"],

    "Orders": [2, 4, 3, 4, 6, 7, 3]
}

df1 = pd.DataFrame(d)

grouped = df1.groupby(["Product"])
result = grouped.sum()

print(df1)
print()
print(result)
```

**Output**

```
      Product  Orders
0     Samsung      2
1       Nokia      4
2     Samsung      3
3  Motorola      4
4       Nokia      6
5     Samsung      7
6     Samsung      3

      Orders
Product
Motorola      4
Nokia        10
Samsung      15
```

## Data Science – Pandas – DataFrame - Groupby

**Program Name** Get the number of product on dates  
**Input file** demo3.py  
sales5.py

```
import pandas as pd

df1 = pd.read_csv("sales5.csv")
grouped = df1.groupby(["Mail_Id"])
result = grouped.size()

print(result)
```

### Output

```
Mail_Id
daniel@gmail.com      3
kedar@gmail.com       3
nirekshan@gmail.com   4
partha@gmail.com      1
prasad@gmail.com      3
shahid@gmail.com      1
dtype: int64
```

## Data Science – Pandas – DataFrame - Groupby

**Program Name** Get the number of product on count  
**Input file** demo4.py  
sales5.py

```
import pandas as pd

df1 = pd.read_csv("sales5.csv")
grouped = df1.groupby(["Product_Name"])
result = grouped.size()

print(result)
```

### Output

```
Product_Name
Kindle Paper White      3
LG Washing Machine     1
Samsung                 3
Sofa set                1
hTC mobile               2
iPad                     1
iPhone 8                  3
iPhone 9                  1
dtype: int64
```

## Data Science – Pandas – DataFrame - Groupby

**Program Name** How many products sold out on each month?  
**Input file** demo5.py  
sales5.py

```
import pandas as pd

df1 = pd.read_csv("sales5.csv")

cols = ['Date', 'Product_Name']
grouped = df1.groupby(cols)['Date']
result = grouped.count()

print(result)
```

### Output

```
Date      Product_Name
09/01/2019  iPad           1
09/02/2019  LG Washing Machine  1
09/06/2019  Kindle Paper White  2
              Sofa set          1
              iPhone 8          1
10/31/2019  Kindle Paper White  1
11/02/2019  Samsung          2
              hTC mobile         1
11/06/2019  Samsung          1
              iPhone 8          2
              iPhone 9          1
11/10/2019  hTC mobile         1
Name: Date, dtype: int64
```

## Data Science – Pandas – DataFrame - Groupby

**Program Name** Product wise each month sales  
**Input file** demo6.py  
sales5.py

```
import pandas as pd

df1 = pd.read_csv("sales5.csv")

cols = ['Product_Name', 'Date']
grouped = df1.groupby(cols)['Date']
result = grouped.count()

print(result)
```

### Output

```
Product_Name      Date
Kindle Paper White 09/06/2019    2
                  10/31/2019    1
LG Washing Machine 09/02/2019    1
Samsung           11/02/2019    2
                  11/06/2019    1
Sofa set           09/06/2019    1
hTC mobile         11/02/2019    1
                  11/10/2019    1
iPad               09/01/2019    1
iPhone 8            09/06/2019    1
                  11/06/2019    2
iPhone 9             11/06/2019    1
Name: Date, dtype: int64
```

## Data Science – Pandas – DataFrame - Groupby

**Program Name** Customer wise each month sales  
**Input file** demo7.py  
sales5.py

```
import pandas as pd

df1 = pd.read_csv("sales5.csv")

cols = ['Mail_Id', 'Date']
grouped = df1.groupby(cols)['Mail_Id']
result = grouped.count()

print(result)
```

### Output

```
Mail_Id          Date
daniel@gmail.com 09/06/2019    2
                  10/31/2019   1
kedar@gmail.com  09/06/2019    1
                  11/06/2019   2
nirekshan@gmail.com 09/06/2019  1
                      11/02/2019  2
                      11/06/2019  1
partha@gmail.com  11/06/2019  1
prasad@gmail.com  09/02/2019  1
                      11/02/2019  1
                      11/10/2019  1
shahid@gmail.com  09/01/2019  1
Name: Mail_Id, dtype: int64
```

## Data Science – Pandas – DataFrame - Groupby

**Program Name** Customer wise product details  
**Input file** demo8.py  
sales5.py

```
import pandas as pd

df1 = pd.read_csv("sales5.csv")

col = ['Mail_Id', 'Product_Name']
grouped = df1.groupby(col)['Product_Cost']
result = grouped.sum()

print(result)
```

### Output

```
Mail_Id          Product_Name
Daniel@gmail.com Kindle Paper White    20000
                  Sofa set            50000
kedar@gmail.com   iPhone 8           60000
nirekshan@gmail.com Kindle Paper White    20000
                      Samsung          30000
partha@gmail.com  iPhone 9           30000
prasad@gmail.com  LG Washing Machine  25000
                      hTC mobile        30000
shahid@gmail.com  iPad              70000
Name: Product_Cost, dtype: int64
```

## Data Science – Pandas – DataFrame - Groupby

---

**Program Name** Customer wise product details  
**demo9.py**  
**Input file** sales5.py

```
import pandas as pd

df1 = pd.read_csv("sales5.csv")

col = ['Mail_Id', 'Product_Name']
grouped = df1.groupby(col, as_index = False)[['Product_Cost']]
result = grouped.sum()

print(result)
```

### Output

	Mail_Id	Product_Name	Product_Cost
0	Daniel@gmail.com	Kindle Paper White	20000
1	Daniel@gmail.com	Sofa set	50000
2	kedar@gmail.com	iPhone 8	60000
3	nirekshan@gmail.com	Kindle Paper White	20000
4	nirekshan@gmail.com	Samsung	30000
5	partha@gmail.com	iPhone 9	30000
6	prasad@gmail.com	LG Washing Machine	25000
7	prasad@gmail.com	hTC mobile	30000
8	shahid@gmail.com	iPad	70000

## Data Science – Pandas – DataFrame - Groupby

**Program** Customer wise product sales

**Name** demo10.py

**Input file** sales5.py

```
import pandas as pd

df1 = pd.read_csv("sales5.csv")
col = ['Mail_Id']
grouped = df1.groupby(col)['Product_Cost']
result = grouped.sum()

print(result)
```

**Output**

```
Mail_Id
Daniel@gmail.com      70000
kedar@gmail.com       60000
nirekshan@gmail.com   50000
partha@gmail.com      30000
prasad@gmail.com      55000
shahid@gmail.com      70000
Name: Product_Cost, dtype: int64
```

## Data Science – Pandas – DataFrame - Groupby

**Program Name** Product wise whole sales  
**Input file** demo11.py  
sales5.py

```
import pandas as pd

df1 = pd.read_csv("sales5.csv")
col = ['Product_Name']
grouped = df1.groupby(col)['Product_Cost']
result = grouped.sum()

print(result)
```

### Output

```
Product_Name
Kindle Paper White      40000
LG Washing Machine     25000
Samsung                 30000
Sofa set                50000
hTC mobile               30000
iPad                      70000
iPhone 8                  60000
iPhone 9                  30000
Name: Product_Cost, dtype: int64
```

## Data Science – Pandas – DataFrame - Groupby

**Program Name** Day wise product sales

**Name** demo12.py

**Input file** sales5.py

```
import pandas as pd

df1 = pd.read_csv("sales5.csv")
col = ['Date']
grouped = df1.groupby(col)['Product_Cost']
result = grouped.sum()

print(result)
```

**Output**

```
Date
09/01/2019      70000
09/02/2019      25000
09/06/2019      100000
10/31/2019      10000
11/02/2019      35000
11/06/2019      80000
11/10/2019      15000
Name: Product_Cost, dtype: int64
```

## Data Science – Pandas – DataFrame - Groupby

Program      Describe method

Name          demo13.py

Input file    sales5.py

```
import pandas as pd

df1 = pd.read_csv("sales5.csv")
df2 = df1['Product_Cost'].describe()

print(df2)
```

Output

```
count      15.000000
mean      22333.333333
std       16888.993316
min      10000.000000
25%      10000.000000
50%      20000.000000
75%      22500.000000
max      70000.000000
Name: Product_Cost, dtype: float64
```

## Data Science – Pandas – DataFrame - Groupby

---

**Program Name** Applying a single function to columns in groups  
**Input file** demo14.py  
sales5.py

```
import pandas as pd

df1 = pd.read_csv("sales5.csv")

d = {
    'Product_Cost' : sum
}

cols = ['Date', 'Product_Name']
grouped = df1.groupby(cols)
result = grouped.agg(d)

print(result)
```

### Output

Date	Product_Name	Product_Cost
09/01/2019	iPad	70000
09/02/2019	LG Washing Machine	25000
09/06/2019	Kindle Paper White	30000
	Sofa set	50000
	iPhone 8	20000
10/31/2019	Kindle Paper White	10000
11/02/2019	Samsung	20000
	hTC mobile	15000
11/06/2019	Samsung	10000
	iPhone 8	40000
	iPhone 9	30000
11/10/2019	hTC mobile	15000

## Data Science – Pandas – DataFrame - Groupby

---

**Program Name** Applying a single function to columns in groups  
**Input file** demo15.py  
sales5.py

```
import pandas as pd

df1 = pd.read_csv("sales5.csv")

d = {
    'Product_Cost': sum,
    'Product_Name': "count",
}

cols = ['Date', 'Product_Name']

grouped = df1.groupby(cols)
result = grouped.agg(d)

print(result)
```

### Output

Date	Product_Name	Product_Cost	Product_Name
09/01/2019	iPad	70000	1
09/02/2019	LG Washing Machine	25000	1
09/06/2019	Kindle Paper White	30000	2
	Sofa set	50000	1
	iPhone 8	20000	1
10/31/2019	Kindle Paper White	10000	1
11/02/2019	Samsung	20000	2
	hTC mobile	15000	1
11/06/2019	Samsung	10000	1
	iPhone 8	40000	2
	iPhone 9	30000	1
11/10/2019	hTC mobile	15000	1

## Data Science – Pandas – DataFrame - Groupby

**Program Name** Applying a single function to columns in groups  
**Input file** demo16.py  
sales5.py

```
import pandas as pd

df1 = pd.read_csv('sales5.csv')

d = {
    'Product_Cost': [min, max, sum]
}

col = ['Product_Cost']
grouped = df1.groupby(col)
result = grouped.agg(d)

print(result)
```

### Output

	Product_Cost	Product_Cost		
		min	max	sum
Product_Cost	10000	10000	10000	50000
	15000	15000	15000	30000
	20000	20000	20000	80000
	25000	25000	25000	25000
	30000	30000	30000	30000
	50000	50000	50000	50000
	70000	70000	70000	70000

**17. Pandas – DataFrame - Merging or Joining****Contents**

<b>1. Introduction.....</b>	<b>2</b>
<b>2. merge(p1, p2, p3, p4) function .....</b>	<b>2</b>
<b>3. Types of joins.....</b>	<b>2</b>
3.1. Inner join .....	3
3.2. Left join .....	8
3.3. Right join .....	11
3.4. Outer Merge / Full outer join.....	14
<b>4. Other type of joins .....</b>	<b>17</b>
4.1. One to one .....	18
4.2. Many to one.....	20
4.3. Many to many .....	22

## 17. Pandas – DataFrame - Merging or Joining

### 1. Introduction

- ✓ By using pandas we can perform join operations as well.
- ✓ This is same as join operations in database.
- ✓ Here we are performing join in between the DataFrames
- ✓ Joining is the process of bringing two DataFrames into one DataFrame based on common attributes in columns

### 2. `merge(p1, p2, p3, p4)` function

- ✓ `merge(p1, p2, p3, p4)` is a predefined function in pandas.
  - `pd.merge(df1, df2, on = "column", how = "type of join")`
- ✓ We should access this function by using pandas library
- ✓ By using this function we can perform join operations over DataFrames

### 'how' Argument

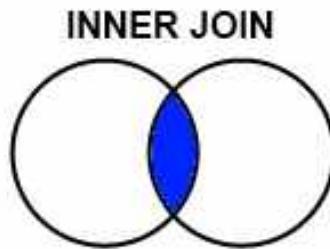
- ✓ The how argument helps to specify the type of join.

### 3. Types of joins

- ✓ Inner Merge / Inner join
- ✓ Left Merge / Left outer join
- ✓ Right Merge / Right outer join
- ✓ Outer Merge / Full outer join

### 3.1. Inner join

- ✓ We can perform inner join on DataFrames.
- ✓ This inner join is a kind of intersection means, it keeps the common data.



**merge(df1, df2, on = "column", how = "type")**

- ✓ `merge(df1, df2, on = "column", how = "type" )` is predefined function in pandas.
- ✓ To perform inner join, we need to pass how value as "inner" as per the syntax

#### Syntax

```
pd.merge(df1, df2, on = "column", how = "inner")
```

## Data Science – Pandas – DataFrame – Merging or Joining

**Program Name** Creating two DataFrames  
demo1.py

```
import pandas as pd

d1 = {
    "Id": [1, 2, 3, 4, 5, 6],
    "Name": ["Pradhan", "Venu", "Madhurima", "Nireekshan",
    "Shafi", "Veeru"],
    "Subject": ["English", "Java", "Html", "Python", "C", "dot
    net"]
}

d2 = {
    "Id": [11, 12, 13, 14, 15, 16],
    "Name": ["Srinu", "Sumanth", "Neelima", "Daniel", "Arjun",
    "Veeru"],
    "Subject": ["Java", "Html", "Cpp", "Python", "C", "dot net"]
}

df1 = pd.DataFrame(d1)
df2 = pd.DataFrame(d2)

print(df1)
print()
print(df2)
```

## Data Science – Pandas – DataFrame – Merging or Joining

### Output

```
      Id      Name  Subject
0    1    Pradhan   English
1    2      Venu       Java
2    3  Madhurima     Html
3    4  Nireekshan   Python
4    5      Shafi        C
5    6      Veeru  dot net

      Id      Name  Subject
0   11      Srinu       Java
1   12  Sumanth     Html
2   13  Neelima      Cpp
3   14    Daniel   Python
4   15    Arjun        C
5   16      Veeru  dot net
```

## Data Science – Pandas – DataFrame – Merging or Joining

Program

Inner Join

Name

demo2.py

```
import pandas as pd

d1 = {
    "Id": [1, 2, 3, 4, 5, 6],
    "Name": ["Pradhan", "Venu", "Madhurima", "Nireekshan",
              "Shafi", "Veeru"],
    "Subject": ["English", "Java", "Html", "Python", "C", "dot
                net"]
}

d2 = {
    "Id": [11, 12, 13, 14, 15, 16],
    "Name": ["Srinu", "Sumanth", "Neelima", "Daniel", "Arjun",
              "Veeru"],
    "Subject": ["Java", "Html", "Cpp", "Python", "C", "dot net"]
}

df1 = pd.DataFrame(d1)
df2 = pd.DataFrame(d2)

inn_join = pd.merge(df1, df2, on = "Subject", how = "inner")

print(df1)
print()
print(df2)
print()
print(inn_join)
```

## Data Science – Pandas – DataFrame – Merging or Joining

### Output

	Id	Name	Subject
0	1	Pradhan	English
1	2	Venu	Java
2	3	Madhurima	Html
3	4	Nireekshan	Python
4	5	Shafi	C
5	6	Veeru	dot net

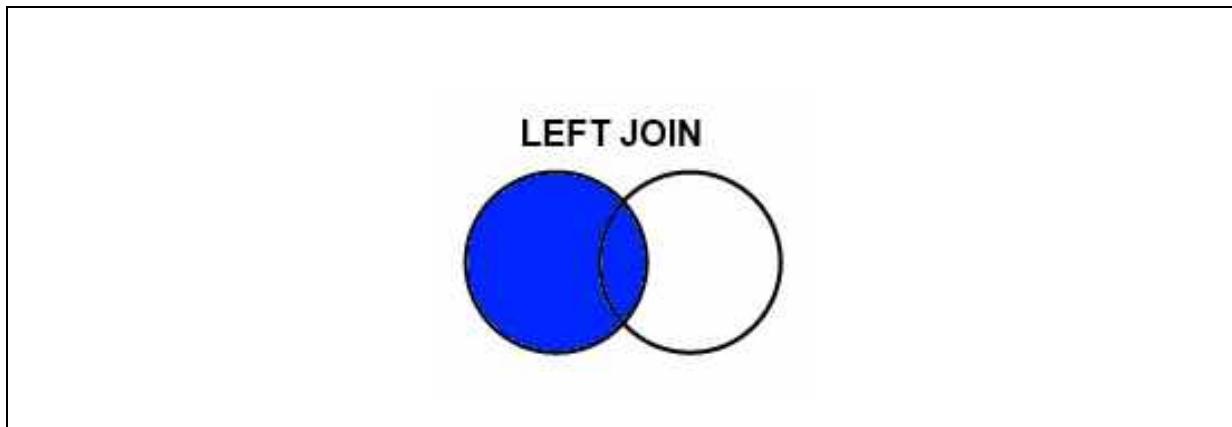
	Id	Name	Subject
0	11	Srinu	Java
1	12	Sumanth	Html
2	13	Neelima	Cpp
3	14	Daniel	Python
4	15	Arjun	C
5	16	Veeru	dot net

	Id_x	Name_x	Subject	Id_y	Name_y
0	2	Venu	Java	11	Srinu
1	3	Madhurima	Html	12	Sumanth
2	4	Nireekshan	Python	14	Daniel
3	5	Shafi	C	15	Arjun
4	6	Veeru	dot net	16	Veeru

### 3.2. Left join

- ✓ Keep every row in the left dataframe.
- ✓ Where there are missing values of the "on" variable in the right dataframe filled with NaN values in the result.



## Data Science – Pandas – DataFrame – Merging or Joining

Program

Left Join

Name

demo3.py

```
import pandas as pd

d1 = {
    "Id": [1, 2, 3, 4, 5, 6],
    "Name": ["Pradhan", "Venu", "Madhurima", "Nireekshan",
              "Shafi", "Veetu"],
    "Subject": ["English", "Java", "Html", "Python", "C", "dot
                net"]
}

d2 = {
    "Id": [11, 12, 13, 14, 15, 16],
    "Name": ["Srinu", "Sumanth", "Neelima", "Daniel", "Arjun",
              "Veetu"],
    "Subject": ["Java", "Html", "Cpp", "Python", "C", "dot net"]
}

df1 = pd.DataFrame(d1)
df2 = pd.DataFrame(d2)

left_join = pd.merge(df1, df2, on = "Subject", how = "left")

print(df1)
print()
print(df2)
print()
print(left_join)
```

## Data Science – Pandas – DataFrame – Merging or Joining

### Output

	Id	Name	Subject
0	1	Pradhan	English
1	2	Venu	Java
2	3	Madhurima	Html
3	4	Nireekshan	Python
4	5	Shafi	C
5	6	Veeru	dot net

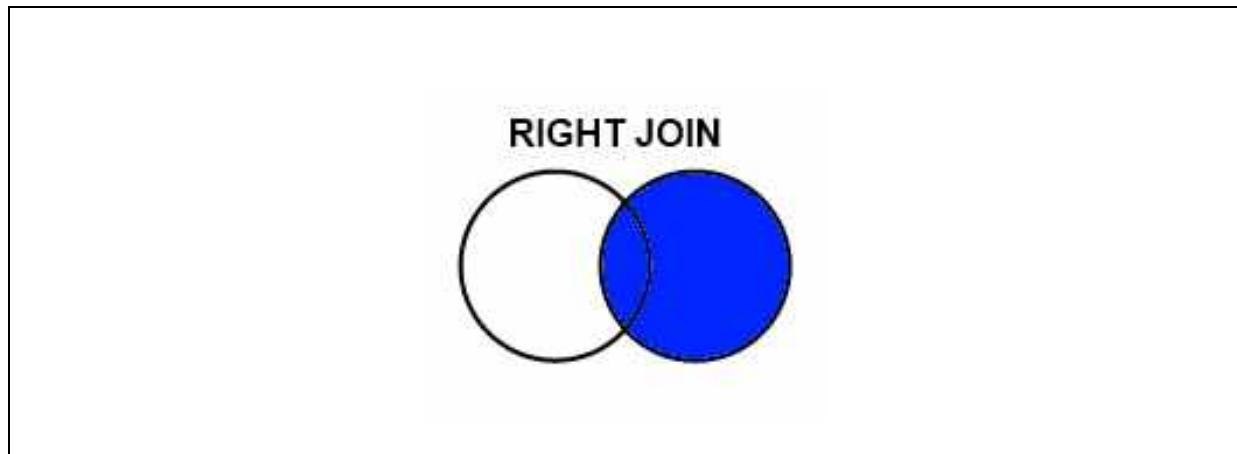
	Id	Name	Subject
0	11	Srinu	Java
1	12	Sumanth	Html
2	13	Neelima	Cpp
3	14	Daniel	Python
4	15	Arjun	C
5	16	Veeru	dot net

	Id_x	Name_x	Subject	Id_y	Name_y
0	1	Pradhan	English	NaN	NaN
1	2	Venu	Java	11.0	Srinu
2	3	Madhurima	Html	12.0	Sumanth
3	4	Nireekshan	Python	14.0	Daniel
4	5	Shafi	C	15.0	Arjun
5	6	Veeru	dot net	16.0	Veeru

### 3.3. Right join

- ✓ Keep every row in the right dataframe.
- ✓ Where there are missing values of the "on" variable in the left column filled with NaN values in the result.



## Data Science – Pandas – DataFrame – Merging or Joining

Program

Name demo4.py

```
import pandas as pd

d1 = {
    "Id": [1, 2, 3, 4, 5, 6],
    "Name": ["Pradhan", "Venu", "Madhurima", "Nireekshan",
              "Shafi", "Veetu"],
    "Subject": ["English", "Java", "Html", "Python", "C", "dot
                net"]
}

d2 = {
    "Id": [11, 12, 13, 14, 15, 16],
    "Name": ["Srinu", "Sumanth", "Neelima", "Daniel", "Arjun",
              "Veetu"],
    "Subject": ["Java", "Html", "Cpp", "Python", "C", "dot net"]
}

df1 = pd.DataFrame(d1)
df2 = pd.DataFrame(d2)

right_join = pd.merge(df1, df2, on = "Subject", how = "right")

print(df1)
print()
print(df2)
print()
print(right_join)
```

## Data Science – Pandas – DataFrame – Merging or Joining

### Output

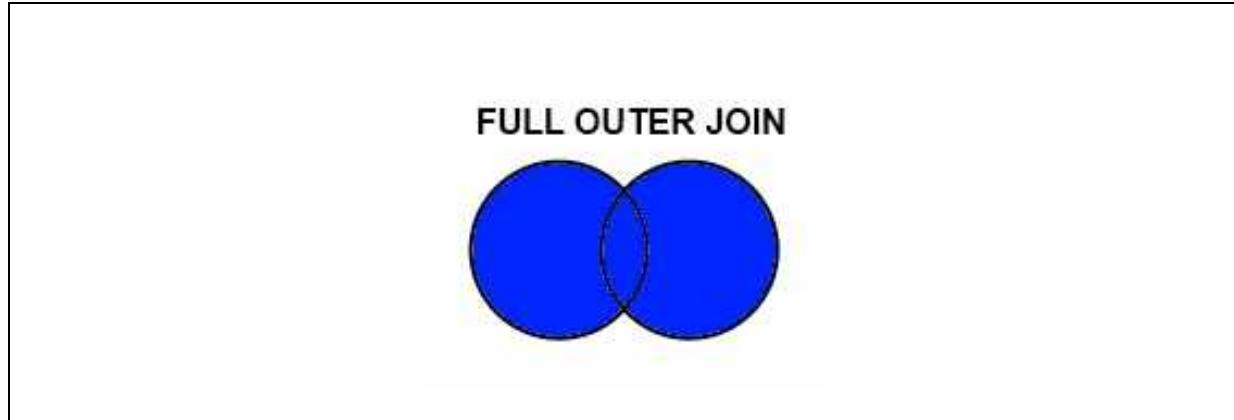
```
      Id      Name  Subject
0    1    Pradhan  English
1    2      Venu      Java
2    3  Madhurima     Html
3    4  Nireekshan   Python
4    5      Shafi        C
5    6      Veeru  dot net

      Id      Name  Subject
0   11    Srinu      Java
1   12  Sumanth     Html
2   13  Neelima      Cpp
3   14  Daniel   Python
4   15    Arjun        C
5   16    Veeru  dot net

  Id_x      Name_x  Subject  Id_y      Name_y
0  2.0        Venu      Java    11    Srinu
1  3.0  Madhurima     Html    12  Sumanth
2  NaN        NaN      Cpp    13  Neelima
3  4.0  Nireekshan   Python    14  Daniel
4  5.0      Shafi        C    15    Arjun
5  6.0      Veeru  dot net    16    Veeru
```

### 3.4. Outer Merge / Full outer join

- ✓ A full outer join returns all the rows from the left and right DataFrames.
- ✓ Where there is no common data there it will be filled with NaN values.



## Data Science – Pandas – DataFrame – Merging or Joining

Program

Name Outer Join

demo5.py

```
import pandas as pd

d1 = {
    "Id": [1, 2, 3, 4, 5, 6],
    "Name": ["Pradhan", "Venu", "Madhurima", "Nireekshan",
              "Shafi", "Veetu"],
    "Subject": ["English", "Java", "Html", "Python", "C", "dot
                net"]
}

d2 = {
    "Id": [11, 12, 13, 14, 15, 16],
    "Name": ["Srinu", "Sumanth", "Neelima", "Daniel", "Arjun",
              "Veetu"],
    "Subject": ["Java", "Html", "Cpp", "Python", "C", "dot net"]
}

df1 = pd.DataFrame(d1)
df2 = pd.DataFrame(d2)

outer_join = pd.merge(df1, df2, on = "Subject", how = "outer")

print(df1)
print()
print(df2)
print()
print(outer_join)
```

## Data Science – Pandas – DataFrame – Merging or Joining

### Output

	Id	Name	Subject
0	1	Pradhan	English
1	2	Venu	Java
2	3	Madhurima	Html
3	4	Nireekshan	Python
4	5	Shafi	C
5	6	Veeru	dot net

	Id	Name	Subject
0	11	Srinu	Java
1	12	Sumanth	Html
2	13	Neelima	Cpp
3	14	Daniel	Python
4	15	Arjun	C
5	16	Veeru	dot net

	Id_x	Name_x	Subject	Id_y	Name_y
0	1.0	Pradhan	English	NaN	NaN
1	2.0	Venu	Java	11.0	Srinu
2	3.0	Madhurima	Html	12.0	Sumanth
3	4.0	Nireekshan	Python	14.0	Daniel
4	5.0	Shafi	C	15.0	Arjun
5	6.0	Veeru	dot net	16.0	Veeru
6	NaN	NaN	Cpp	13.0	Neelima

## Data Science – Pandas – DataFrame – Merging or Joining

---

### 4. Other type of joins

- ✓ One to one
- ✓ Many to one
- ✓ Many to many

**Program Name** Creating DataFrames  
demo6.py

```
import pandas as pd

d1 = {
    "Employee": ["Nireekshan", "Veeru", "Lavanya", "Pradhan"],
    "Group": ["Development", "Testing", "Testing", "HR"]
}

d2 = {
    "Employee": ["Lavanya", "Nireekshan", "Veeru", "Pradhan"],
    "Hire_date": [2010, 2012, 2014, 2016]
}

df1 = pd.DataFrame(d1)
df2 = pd.DataFrame(d2)

print(df1)
print()
print(df2)
```

### Output

	Employee	Group
0	Nireekshan	Development
1	Veeru	Testing
2	Lavanya	Testing
3	Pradhan	HR

	Employee	Hire_date
0	Lavanya	2010
1	Nireekshan	2012
2	Veeru	2014
3	Pradhan	2016

#### 4.1. One to one

- ✓ This is very simple join and similar to the column-wise concatenation

<b>Program Name</b>	Creating DataFrames demo7.py
---------------------	---------------------------------

```
import pandas as pd

d1 = {
    "Employee": ["Nireekshan", "Veeru", "Lavanya", "Pradhan"],
    "Group": ["Development", "Testing", "Testing", "HR"]
}

d2 = {
    "Employee": ["Lavanya", "Nireekshan", "Veeru", "Pradhan"],
    "Hire_date": [2010, 2012, 2014, 2016]
}

df1 = pd.DataFrame(d1)
df2 = pd.DataFrame(d2)

one_one = pd.merge(df1, df2)

print(df1)
print()
print(df2)
print()
print(one_one)
```

## Data Science – Pandas – DataFrame – Merging or Joining

### Output

```
      Employee      Group
0   Nireekshan  Development
1       Veeru        Testing
2     Lavanya        Testing
3     Pradhan          HR

      Employee  Hire_date
0     Lavanya      2010
1  Nireekshan      2012
2       Veeru      2014
3     Pradhan      2016

      Employee      Group  Hire_date
0   Nireekshan  Development      2012
1       Veeru        Testing      2014
2     Lavanya        Testing      2010
3     Pradhan          HR        2016
```

## 4.2. Many to one

- ✓ Many-to-one joins are joins in which one of the two key columns contains duplicate entries

**Program Name**

Many to one

demo8.py

```
import pandas as pd

d1 = {
    "Employee": ["Nireekshan", "Veeru", "Lavanya", "Pradhan"],
    "Group": ["Development", "Testing", "Testing", "HR"]
}

d2 = {
    "Employee": ["Lavanya", "Nireekshan", "Veeru", "Pradhan"],
    "Hire_date": [2010, 2012, 2014, 2016]
}

d3 = {
    "Group": ["Testing", "Development", "HR"],
    "supervisor": ["Shafi", "Daniel", "Neelima"]
}

df1 = pd.DataFrame(d1)
df2 = pd.DataFrame(d2)
df3 = pd.DataFrame(d3)

one_one = pd.merge(df1, df2)

many_one = pd.merge(one_one, df3)

print(df1)
print()
print(df2)
print()
print(many_one)
```

## Data Science – Pandas – DataFrame – Merging or Joining

### Output

```
      Employee      Group
0  Nireekshan  Development
1      Veeru        Testing
2    Lavanya        Testing
3     Pradhan          HR

      Employee  Hire_date
0      Lavanya      2010
1  Nireekshan      2012
2      Veeru      2014
3     Pradhan      2016

      Employee      Group  Hire_date supervisor
0  Nireekshan  Development      2012      Daniel
1      Veeru        Testing      2014       Shafi
2    Lavanya        Testing      2010       Shafi
3     Pradhan          HR      2016   Neelima
```

### 4.3. Many to many

- ✓ If the key column in both the left and right DataFrame contains duplicates, then the result is a many-to-many merge

Program

Many to many

Name

demo9.py

```
import pandas as pd

d1 = {
    "Employee": ["Nireekshan", "Veeru", "Lavanya", "Pradhan"],
    "Group": ["Development", "Testing", "Testing", "HR"]
}

d2 = {
    "Group": ["Testing", "Testing", "Development",
              "Development", "HR", "HR"],
    "Skills": ["Manual", "Automation", "Coding", "Logical",
               "Spreadsheets", "Organization"]
}

df1 = pd.DataFrame(d1)
df2 = pd.DataFrame(d2)

many_many = pd.merge(df1, df2)

print(df1)
print()
print(df2)
print()
print(many_many)
```

## Data Science – Pandas – DataFrame – Merging or Joining

### Output

```
      Employee      Group
0  Nireekshan  Development
1      Veeru      Testing
2    Lavanya      Testing
3    Pradhan          HR

      Group      Skills
0  Testing      Manual
1  Testing  Automation
2 Development      Coding
3 Development      Logical
4          HR  Spreadsheets
5          HR  Organization

      Employee      Group      Skills
0  Nireekshan  Development      Coding
1  Nireekshan  Development      Logical
2      Veeru      Testing      Manual
3      Veeru      Testing  Automation
4    Lavanya      Testing      Manual
5    Lavanya      Testing  Automation
6    Pradhan          HR  Spreadsheets
7    Pradhan          HR  Organization
```

## Data Science – Pandas – DataFrame – Merging or Joining

### Based on column wise

- ✓ We can also do merge DataFrames based on columns wise as well

```
Program      Creating DataFrames
Name        demo10.py

import pandas as pd

d1 = {
    "Id": [1, 2, 3, 4, 5, 6],
    "Name": ["Pradhan", "Venu", "Madhurima", "Nireekshan",
              "Shafi", "Veeru"],
    "Subject": ["English", "Java", "Html", "Python", "C", "dot
                net"]
}

d2 = {
    "Id": [11, 12, 13, 14, 15, 16],
    "Name": ["Srinu", "Sumanth", "Neelima", "Daniel", "Arjun",
              "Veeru"],
    "Subject": ["Java", "Html", "Cpp", "Python", "C", "dot net"]
}

df1 = pd.DataFrame(d1)
df2 = pd.DataFrame(d2)

print(df1)
print()
print(df2)
```

## Data Science – Pandas – DataFrame – Merging or Joining

### Output

```
      Id      Name  Subject
0    1    Pradhan  English
1    2      Venu     Java
2    3  Madhurima    Html
3    4  Nireekshan  Python
4    5      Shafi       C
5    6      Veeru   dot net

      Id      Name  Subject
0   11     Srinu     Java
1   12  Sumanth    Html
2   13  Neelima     Cpp
3   14    Daniel  Python
4   15    Arjun       C
5   16      Veeru   dot net
```

## Data Science – Pandas – DataFrame – Merging or Joining

**Program Name** Merging two DataFrames based on column  
demo11.py

```
import pandas as pd

d1 = {
    "Id": [1, 2, 3, 4, 5, 6],
    "Name": ["Pradhan", "Venu", "Madhurima", "Nireekshan",
    "Shafi", "Veeru"],
    "Subject": ["English", "Java", "Html", "Python", "C", "dot
    net"]
}

d2 = {
    "Id": [11, 12, 13, 14, 15, 16],
    "Name": ["Srinu", "Sumanth", "Neelima", "Daniel", "Arjun",
    "Veeru"],
    "Subject": ["Java", "Html", "Cpp", "Python", "C", "dot net"]
}

df1 = pd.DataFrame(d1)
df2 = pd.DataFrame(d2)

result = pd.merge(df1, df2, on = 'Subject')

print(df1)
print()
print(df2)
print()
print(result)
```

## Data Science – Pandas – DataFrame – Merging or Joining

### Output

```
      Id      Name  Subject
0    1    Pradhan   English
1    2      Venu      Java
2    3  Madhurima     Html
3    4  Nireekshan   Python
4    5      Shafi        C
5    6      Veeru  dot net

      Id      Name  Subject
0   11    Srinu      Java
1   12  Sumanth     Html
2   13  Neelima      Cpp
3   14   Daniel   Python
4   15    Arjun        C
5   16    Veeru  dot net

  Id_x      Name_x  Subject  Id_y      Name_y
0    2      Venu      Java   11    Srinu
1    3  Madhurima     Html   12  Sumanth
2    4  Nireekshan   Python   14   Daniel
3    5      Shafi        C   15    Arjun
4    6      Veeru  dot net   16    Veeru
```

## Data Science – Pandas – DataFrame – Merging or Joining

**Program Name** Merging two DataFrames with multiple columns  
demo12.py

```
import pandas as pd

d1 = {
    "Id": [1, 2, 3, 4, 5, 6],
    "Name": ["Pradhan", "Venu", "Madhurima", "Nireekshan",
    "Shafi", "Veeru"],
    "Subject": ["English", "Java", "Html", "Python", "C", "dot
    net"]
}

d2 = {
    "Id": [11, 12, 13, 14, 15, 16],
    "Name": ["Srinu", "Sumanth", "Neelima", "Daniel", "Arjun",
    "Veeru"],
    "Subject": ["Java", "Html", "Cpp", "Python", "C", "dot net"]
}

df1 = pd.DataFrame(d1)
df2 = pd.DataFrame(d2)

result = pd.merge(df1, df2, on=["Name", "Subject"])

print(df1)
print()
print(df2)
print()
print(result)
```

## Data Science – Pandas – DataFrame – Merging or Joining

### Output

```
      Id      Name  Subject
0    1    Pradhan   English
1    2      Venu      Java
2    3  Madhurima     Html
3    4  Nireekshan   Python
4    5      Shafi        C
5    6      Veeru  dot net

      Id      Name  Subject
0   11     Srinu      Java
1   12  Sumanth     Html
2   13  Neelima      Cpp
3   14    Daniel   Python
4   15    Arjun        C
5   16      Veeru  dot net

  Id_x      Name  Subject  Id_y
0    6    Veeru  dot net     16
```

**18. Pandas – DataFrame – Concat****Contents**

1. Introduction .....	2
2. concat(p) .....	2

## 18. Pandas – DataFrame – Concat

### 1. Introduction

- ✓ Based on requirement we can add(concatenate) one DataFrame with another DataFrame.

### 2. concat(p)

- ✓ concat(p) is predefined function in pandas package.
- ✓ We should access this function by using pandas library name
- ✓ This function concatenate one DataFrame to another DataFrame

**Program Name** Creating DataFrames  
demo1.py

```
import pandas as pd

d1 = [[11, 22], [33, 44]]
d2 = [[55, 66], [77, 88]]

c1 = ["A", "B"]

df1 = pd.DataFrame(d1, columns = c1)
df2 = pd.DataFrame(d2, columns = c1)

print(df1)
print()
print(df2)
```

### Output

	A	B
0	11	22
1	33	44

	A	B
0	55	66
1	77	88

## Data Science – Pandas – DataFrame – Concat

---

**Program Name** Concatenating one DataFrame to another DataFrame  
demo2.py

```
import pandas as pd

d1 = [[11, 22], [33, 44]]
d2 = [[55, 66], [77, 88]]

c1 = ["A", "B"]

df1 = pd.DataFrame(d1, columns = c1)
df2 = pd.DataFrame(d2, columns = c1)

result = [df1, df2]

df3 = pd.concat(result)

print(df1)
print()
print(df2)
print()
print(df3)
```

## Data Science – Pandas – DataFrame – Concat

### Output

```
      A    B
0   11   22
1   33   44

      A    B
0   55   66
1   77   88

      A    B
0   11   22
1   33   44
0   55   66
1   77   88
```

## Data Science – Pandas – DataFrame – Concat

---

**Program Name** Concatenating one DataFrame to another DataFrame  
demo3.py

```
import pandas as pd

d1 = [[11, 22], [33, 44]]
d2 = [[55, 66], [77, 88]]

c1 = ["A", "B"]

df1 = pd.DataFrame(d1, columns = c1)
df2 = pd.DataFrame(d2, columns = c1)

result = [df1, df2]

df3 = pd.concat(result, ignore_index = True)

print(df1)
print()
print(df2)
print()
print(df3)
```

## Data Science – Pandas – DataFrame – Concat

### Output

```
      A    B  
0   11   22  
1   33   44  
  
      A    B  
0   55   66  
1   77   88  
  
      A    B  
0   11   22  
1   33   44  
2   55   66  
3   77   88
```

## Data Science – Pandas – DataFrame – Concat

---

**Program Name** Concatenating one DataFrame to another DataFrame  
demo4.py

```
import pandas as pd

d1 = [[11, 22], [33, 44]]
d2 = [[55, 66], [77, 88], [67, 99]]

c1 = ["A", "B"]

df1 = pd.DataFrame(d1, columns = c1)
df2 = pd.DataFrame(d2, columns = c1)

result = [df1, df2]

df3 = pd.concat(result)

print(df1)
print()
print(df2)
print()
print(df3)
```

## Data Science – Pandas – DataFrame – Concat

---

### Output

```
      A    B
0   11   22
1   33   44

      A    B
0   55   66
1   77   88
2   67   99

      A    B
0   11   22
1   33   44
0   55   66
1   77   88
2   67   99
```

## Data Science – Pandas – DataFrame – Concat

---

**Program Name** Concatenating one DataFrame to another DataFrame  
demo5.py

```
import pandas as pd

d1 = [[11, 22], [33, 44]]
d2 = [[55, 66], [77, 88], [67, 99]]

c1 = ["A", "B"]

df1 = pd.DataFrame(d1, columns = c1)
df2 = pd.DataFrame(d2, columns = c1)

result = [df1, df2]

df3 = pd.concat(result, ignore_index = True)

print(df1)
print()
print(df2)
print()
print(df3)
```

## Data Science – Pandas – DataFrame – Concat

### Output

```
      A    B
0   11   22
1   33   44

      A    B
0   55   66
1   77   88
2   67   99

      A    B
0   11   22
1   33   44
2   55   66
3   77   88
4   67   99
```

## Data Science – Pandas – DataFrame – Concat

**Program Name** Concatenating one DataFrame to another DataFrame  
demo6.py

```
import pandas as pd

d1 = [[11, 22], [33, 44]]
d2 = [[55, 66], [77, 88]]

c1 = ["A", "B"]
c2 = ["X", "Y"]

df1 = pd.DataFrame(d1, columns = c1)
df2 = pd.DataFrame(d2, columns = c2)

print(df1)
print()
print(df2)
```

### Output

	A	B
0	11	22
1	33	44
	X	Y
0	55	66
1	77	88

## Data Science – Pandas – DataFrame – Concat

**Program Name** Concatenating one DataFrame to another DataFrame  
demo7.py

```
import pandas as pd

d1 = [[11, 22], [33, 44]]
d2 = [[55, 66], [77, 88]]

c1 = ["A", "B"]
c2 = ["X", "Y"]

df1 = pd.DataFrame(d1, columns = c1)
df2 = pd.DataFrame(d2, columns = c2)

result = [df1, df2]

df3 = pd.concat(result, axis = 1)

print(df1)
print()
print(df2)
print()
print(df3)
```

### Output

```
A   B
0  11  22
1  33  44

      X   Y
0  55  66
1  77  88

      A   B   X   Y
0  11  22  55  66
1  33  44  77  88
```

## Data Science – Pandas – DataFrame – Concat

**Program Name** Concatenating one DataFrame to another DataFrame  
demo8.py

```
import pandas as pd

d1 = [[11, 22], [33, 44]]
d2 = [[55, 66], [77, 88], [65, 99]]

c1 = ["A", "B"]
c2 = ["X", "Y"]

df1 = pd.DataFrame(d1, columns = c1)
df2 = pd.DataFrame(d2, columns = c2)

print(df1)
print()
print(df2)
```

### Output

	A	B
0	11	22
1	33	44

	X	Y
0	55	66
1	77	88
2	65	99

## Data Science – Pandas – DataFrame – Concat

---

**Program Name** Concatenating one DataFrame to another DataFrame  
demo9.py

```
import pandas as pd

d1 = [[11, 22], [33, 44]]
d2 = [[55, 66], [77, 88], [65, 99]]

c1 = ["A", "B"]
c2 = ["X", "Y"]

df1 = pd.DataFrame(d1, columns = c1)
df2 = pd.DataFrame(d2, columns = c2)

result = [df1, df2]

df3 = pd.concat(result, axis = 1)

print(df1)
print()
print(df2)
print()
print(df3)
```

## Data Science – Pandas – DataFrame – Concat

### Output

```
      A    B
0   11   22
1   33   44

      X    Y
0   55   66
1   77   88
2   65   99

      A    B    X    Y
0   11.0  22.0  55   66
1   33.0  44.0  77   88
2     NaN   NaN   65   99
```

## Data Science – Pandas DataFrame – Adding, dropping cols/rows

---

### 19. Pandas – DataFrame - Adding, dropping columns & rows

#### Contents

1. Adding column to DataFrame .....	2
2. Dropping columns from DataFrame .....	6
3. Dropping rows from DataFrame .....	8

## Data Science – Pandas DataFrame – Adding, dropping cols/rows

### 19. Pandas – DataFrame - Adding, dropping columns & rows

#### 1. Adding column to DataFrame

- ✓ Based on requirement we can add column to existing DataFrame

**Program** Creating DataFrame  
**Name** demo1.py  
**Input file** sales8.csv

```
import pandas as pd

df = pd.read_csv("sales8.csv")

print(df.head(5))
```

#### Output

	Order Id	Customer name	Customer Id	Product name	Product cost	Quantity
0	192837	Veeru	3	LG Mobile	65999	1
1	192838	Neelima	19	Apple iPad 10.2-inch	63000	2
2	192839	Balaji	12	34in Ultrawide Monitor	75999	1
3	192840	Shahid	20	iPhone 11	60000	2
4	192841	Vinay	10	Bose SoundSport Headphones	69999	3

## Data Science – Pandas DataFrame – Adding, dropping cols/rows

**Program Name** Adding Status column to DataFrame  
**Name** demo6.py  
**Input file** sales8.csv

```
import pandas as pd

df = pd.read_csv("sales8.csv")

df['Status'] = "Delivered"
print(df.head())
```

### Output

	Order Id	Customer name	Customer Id	Product name	Product cost	Quantity	Status
0	192837	Veeru	3	LG Mobile	65999	1	Delivered
1	192838	Neelima	19	Apple iPad 10.2-inch	63000	2	Delivered
2	192839	Balaji	12	34in Ultrawide Monitor	75999	1	Delivered
3	192840	Shahid	20	iPhone 11	60000	2	Delivered
4	192841	Vinay	10	Bose SoundSport Headphones	69999	3	Delivered

**Program Name** Adding Total cost column to DataFrame  
**Name** demo2.py  
**Input file** sales8.csv

```
import pandas as pd

df = pd.read_csv("sales8.csv")

df["Total Cost"] = df['Product cost']*df['Quantity']

print(df.head(5))
```

### Output

	Order Id	Customer name	Customer Id	Product name	Product cost	Quantity	Total Cost
0	192837	Veeru	3	LG Mobile	65999	1	65999
1	192838	Neelima	19	Apple iPad 10.2-inch	63000	2	126000
2	192839	Balaji	12	34in Ultrawide Monitor	75999	1	75999
3	192840	Shahid	20	iPhone 11	60000	2	120000
4	192841	Vinay	10	Bose SoundSport Headphones	69999	3	209997

## Data Science – Pandas DataFrame – Adding, dropping cols/rows

---

**Program Name** Adding Total cost column to DataFrame by using apply method  
**Input file** demo3.py  
sales8.csv

```
import pandas as pd

df = pd.read_csv('sales8.csv')

def total(df):
    t = df['Product cost'] * df['Quantity']
    return t

df['Total cost'] = df.apply(total, axis = 1)

print(df.head())
```

### Output

	Order Id	Customer name	Customer Id	Product name	Product cost	Quantity	Total cost
0	192837	Veeru	3	LG Mobile	65999	1	65999
1	192838	Neelima	19	Apple iPad 10.2-inch	63000	2	126000
2	192839	Balaji	12	34in Ultrawide Monitor	75999	1	75999
3	192840	Shahid	20	iPhone 11	60000	2	120000
4	192841	Vinay	10	Bose SoundSport Headphones	69999	3	209997

## Data Science – Pandas DataFrame – Adding, dropping cols/rows

**Program Name** Adding total cost column to DataFrame in a specific position  
**Input file** demo5.py  
sales8.csv

```
import pandas as pd

df = pd.read_csv("sales8.csv")

print(df.head())

new = df['Product cost'] * df['Quantity']
df.insert(5,"Total Cost", new)

print()
print(df.head(5))
```

### Output

	Order Id	Customer name	Customer Id	Product name	Product cost	Quantity
0	192837	Veeru	3	LG Mobile	65999	1
1	192838	Neelima	19	Apple iPad 10.2-inch	63800	2
2	192839	Balaji	12	34in Ultrawide Monitor	75999	1
3	192840	Shahid	20	iPhone 11	60000	2
4	192841	Vinay	18	Bose SoundSport Headphones	69999	3

	Order Id	Customer name	Customer Id	Product name	Product cost	Total Cost	Quantity
0	192837	Veeru	3	LG Mobile	65999	65999	1
1	192838	Neelima	19	Apple iPad 10.2-inch	63800	126000	2
2	192839	Balaji	12	34in Ultrawide Monitor	75999	75999	1
3	192840	Shahid	20	iPhone 11	60000	120000	2
4	192841	Vinay	18	Bose SoundSport Headphones	69999	209997	3

## Data Science – Pandas DataFrame – Adding, dropping cols/rows

### 2. Dropping columns from DataFrame

- ✓ Based on requirement we can drop column from existing DataFrame

**Program Name** Dropping single columns  
**Input file** demo7.py  
sales8.csv

```
import pandas as pd

df1 = pd.read_csv("sales8.csv")

df2 = df1.drop(columns = 'Customer name')

print(df1.head())
print()
print(df2.head())
```

#### Output

	Order Id	Customer name	Customer Id	Product name	Product cost	Quantity
0	192837	Veeru	3	LG Mobile	65999	1
1	192838	Neelima	19	Apple iPad 10.2-inch	63000	2
2	192839	Balaji	12	34in Ultrawide Monitor	75999	1
3	192840	Shahid	20	iPhone 11	60000	2
4	192841	Vinay	10	Bose SoundSport Headphones	69999	3

	Order Id	Customer Id	Product name	Product cost	Quantity
0	192837	3	LG Mobile	65999	1
1	192838	19	Apple iPad 10.2-inch	63000	2
2	192839	12	34in Ultrawide Monitor	75999	1
3	192840	20	iPhone 11	60000	2
4	192841	10	Bose SoundSport Headphones	69999	3

## Data Science – Pandas DataFrame – Adding, dropping cols/rows

---

**Program Name** Dropping multiple columns  
**Input file** demo8.py  
sales8.csv

```
import pandas as pd

df1 = pd.read_csv("sales8.csv")

df2 = df1.drop(['Customer name', 'Product name'], axis = 1)

print(df1.head())
print()
print(df2.head())
```

### Output

	Order Id	Customer name	Customer Id	Product name	Product cost	Quantity
0	192837	Veetu	3	LG Mobile	65999	1
1	192838	Neelima	19	Apple iPad 10.2-inch	63000	2
2	192839	Balaji	12	34in Ultrawide Monitor	75999	1
3	192840	Shahid	20	iPhone 11	60000	2
4	192841	Vinay	10	Bose SoundSport Headphones	69999	3
	Order Id	Customer Id	Product cost	Quantity		
0	192837	3	65999	1		
1	192838	19	63000	2		
2	192839	12	75999	1		
3	192840	20	60000	2		
4	192841	10	69999	3		

## Data Science – Pandas DataFrame – Adding, dropping cols/rows

### 3. Dropping rows from DataFrame

- ✓ We can drop the rows from DataFrame by using drop method

**Program Name** Dropping single row  
**demo9.py**  
**Input file** sales8.csv

```
import pandas as pd

df1 = pd.read_csv("sales8.csv")

df2 = df1.drop(3, axis = 0)

print(df1.head())
print()
print(df2.head())
```

#### Output

	Order Id	Customer name	Customer Id	Product name	Product cost	Quantity
0	192837	Veeru	3	LG Mobile	65999	1
1	192838	Neelima	19	Apple iPad 10.2-inch	63000	2
2	192839	Balaji	12	34in Ultrawide Monitor	75999	1
3	192840	Shahid	20	iPhone 11	60000	2
4	192841	Vinay	18	Base Soundsport Headphones	69999	3

	Order Id	Customer name	Customer Id	Product name	Product cost	Quantity
0	192837	Veeru	3	LG Mobile	65999	1
1	192838	Neelima	19	Apple iPad 10.2-inch	63000	2
2	192839	Balaji	12	34in Ultrawide Monitor	75999	1
4	192841	Vinay	18	Base Soundsport Headphones	69999	3
5	192842	Vinay	18	20in Monitor	65999	2

## Data Science – Pandas DataFrame – Adding, dropping cols/rows

---

**Program Name** Dropping multiple rows  
**Input file** demo10.py  
sales8.csv

```
import pandas as pd

df1 = pd.read_csv("sales8.csv")
df2 = df1.drop([1, 2], axis = 0)

print(df1.head())
print()
print(df2.head())
```

### Output

	Order Id	Customer name	Customer Id	Product name	Product cost	Quantity
0	192837	Veeru	3	LG Mobile	65999	1
1	192838	Neelima	19	Apple iPad 10.2-inch	63000	2
2	192839	Balaji	12	34in Ultrawide Monitor	75999	1
3	192840	Shahid	20	iPhone 11	60000	2
4	192841	Vinay	10	Bose SoundSport Headphones	69999	3

	Order Id	Customer name	Customer Id	Product name	Product cost	Quantity
0	192837	Veeru	3	LG Mobile	65999	1
3	192840	Shahid	20	iPhone 11	60000	2
4	192841	Vinay	10	Bose SoundSport Headphones	69999	3
5	192842	Vinay	10	20in Monitor	65999	2
6	192843	Balaji	12	iPhone 8	55999	3

## 20. Pandas – DataFrame - Date and time

### Contents

<b>1. Date data type .....</b>	2
<b>2. Converting object data type into date data type.....</b>	4
2.1. <code>to_datetime(p)</code> .....	4
2.2. <code>astype (p)</code> .....	5
<b>3. Format parameter .....</b>	6
<b>4. NaT values .....</b>	14
<b>5. Selecting from start to end date values .....</b>	18
<b>6. Access specific dates like last 20 days or 2 months or 2 years records .....</b>	19
<b>7. Extract year, month, day from Date column.....</b>	24
<b>8. Encoding Days of the Week .....</b>	26

## 20. Pandas – DataFrame - Date and time

### 1. Date data type

- ✓ Whenever we load csv file, if that file contains any column having date values then by default pandas will consider that column as object
- ✓ We can provide `parse_dates = ['name of the column1', 'name of the column2']` then pandas considered those columns as datetime data type

**Program** Creating DataFrames

**Name** demo1.py

**File name** sales7\_dates.csv

```
import pandas as pd

df = pd.read_csv('sales7_dates.csv')

print(df.head())
print()
print(df.dtypes)
```

**Output**

Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost	Pur_Date
0	1023	Venki	27in FHD Monitor	59000	1/1/2019 0:00
1	1024	Chaitanya	iPhone 11	69000	1/1/2019 1:00
2	1025	Shahid	Bose SoundSport Headphones	65999	1/1/2019 2:00
3	1026	Veeru	Apple iPad 10.2-inch	63999	1/1/2019 3:00
4	1027	Venu	Google Phone	63999	1/1/2019 4:00

```
Order_Id      int64
Customer_Name  object
Customer_Id    int64
Product_Name   object
Product_Cost   int64
Pur_Date       object
dtype: object
```

## Data Science – Pandas – DataFrame – Date & Time

**Program Name** Loading csv file by using parse\_dates parameter

**Name** demo2.py

**File name** sales7\_dates.csv

```
import pandas as pd

df = pd.read_csv('sales7_dates.csv', parse_dates = ['Pur_Date'])

print(df.head())
print()
print(df.dtypes)
```

### Output

```
   Order_Id Customer_Name Customer_Id      Product_Name Product_Cost      Pur_Date
0     1023        Venki          15    27in FHD Monitor     59000 2019-01-01 00:00:00
1     1024    Chaithanya          14           iPhone 11     69000 2019-01-01 01:00:00
2     1025       Shahid          20  Bose SoundSport Headphones     65999 2019-01-01 02:00:00
3     1026       Veeru           3      Apple iPad 10.2-inch     63999 2019-01-01 03:00:00
4     1027        Venu          23        Google Phone     63999 2019-01-01 04:00:00

Order_Id          int64
Customer_Name    object
Customer_Id       int64
Product_Name      object
Product_Cost      int64
Pur_Date         datetime64[ns]
dtype: object
```

## 2. Converting object data type into date data type

- ✓ We can convert from object data type into datetime data type explicitly.
- ✓ By using,
  - `to_datetime(p)` function
  - `astype(p)` method in Series

### 2.1. `to_datetime(p)`

- ✓ `to_datetime(p)` is predefined function in pandas
- ✓ This function we should access by using pandas library.
- ✓ This function convert from object data type into date data type.

**Program Name** Loading csv file and converting Pur\_Date column into Date format  
**File name** demo3.py  
**File name** sales7\_dates.csv

```
import pandas as pd

df = pd.read_csv('sales7_dates.csv')

df['Pur_Date'] = pd.to_datetime(df['Pur_Date'])

print(df.head())
print()
print(df.dtypes)
```

### Output

Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost	Pur_Date
0	1023	Venki	15	27in FHD Monitor	59000 2019-01-01 00:00:00
1	1024	Chaitanya	14	iPhone 11	69000 2019-01-01 01:00:00
2	1025	Shahid	20	Bose SoundSport Headphones	65999 2019-01-01 02:00:00
3	1026	Veeru	3	Apple iPad 10.2-inch	63999 2019-01-01 03:00:00
4	1027	Venu	23	Google Phone	63999 2019-01-01 04:00:00

Order\_Id int64  
Customer\_Name object  
Customer\_Id int64  
Product\_Name object  
Product\_Cost int64  
Pur\_Date datetime64[ns]  
dtype: object

## Data Science – Pandas – DataFrame – Date & Time

### 2.2. astype (p)

- ✓ astype(p) is predefined method Series class
- ✓ This method we should access by using Series object only
- ✓ This method convert from object data type into datetime data type.

**Program Name** Loading csv file and converting Pur\_Date column into Date format  
**File name** demo4.py  
**sales7\_dates.csv**

```
import pandas as pd

df = pd.read_csv('sales7_dates.csv')

df['Pur_Date'] = df['Pur_Date'].astype('datetime64[ns]')

print(df.head())
print()
print(df.dtypes)
```

### Output

Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost	Pur_Date
0	Venki	15	27in FHD Monitor	59000	2019-01-01 00:00:00
1	Chaithanya	14	iPhone 11	69000	2019-01-01 01:00:00
2	Shahid	20	Bose SoundSport Headphones	65999	2019-01-01 02:00:00
3	Veeru	3	Apple iPad 10.2-inch	63999	2019-01-01 03:00:00
4	Venu	23	Google Phone	63999	2019-01-01 04:00:00

```
Order_Id          int64
Customer_Name    object
Customer_Id      int64
Product_Name     object
Product_Cost     int64
Pur_Date         datetime64[ns]
dtype: object
```

### 3. Format parameter

- ✓ We can represent the date formats in different ways,
  - March 23rd, 2015 as "03-23-15" or "3|23|2015" and etc
- ✓ So, we can use the format parameter to specify the exact format of the string.

Code	Description	Example
%Y	Full year	2001
%m	Month w/ zero padding	04
%d	Day of the month w/ zero padding	09
%I	Hour (12hr clock) w/ zero padding	02
%p	AM or PM	AM
%M	Minute w/ zero padding	05
%S	Second w/ zero padding	09

## Data Science – Pandas – DataFrame – Date & Time

**Program Name** Creating a DataFrame  
demo5.py

```
import pandas as pd

data = {
    'Product': ['Samsung', 'iPhone', 'Motorola'],
    'Status': ['Success', 'Success', 'Failed'],
    'Cost': [10000, 50000, 15000],
    'PurDate': ['20190902','20190913','20190921'],
}

df = pd.DataFrame(data)

print(df.head())
print()
print(df.dtypes)
```

**Output**

```
   Product    Status    Cost    PurDate
0  Samsung  Success  10000  20190902
1    iPhone  Success  50000  20190913
2  Motorola    Failed  15000  20190921

Product      object
Status       object
Cost        int64
PurDate     object
dtype: object
```

## Data Science – Pandas – DataFrame – Date & Time

**Program Name** Converting Date with specific format  
demo6.py

```
import pandas as pd

data = {
    'Product': ['Samsung', 'iPhone', 'Motorola'],
    'Status': ['Success', 'Success', 'Failed'],
    'Cost': [10000, 50000, 15000],
    'PurDate': ['20190902','20190913','20190921'],
}

df = pd.DataFrame(data)
df['PurDate'] = pd.to_datetime(df['PurDate'])

print(df.head())
print()
print(df.dtypes)
```

### Output

```
      Product    Status    Cost    PurDate
0    Samsung  Success  10000  2019-09-02
1      iPhone  Success  50000  2019-09-13
2  Motorola    Failed  15000  2019-09-21

Product          object
Status           object
Cost            int64
PurDate    datetime64[ns]
dtype: object
```

## Data Science – Pandas – DataFrame – Date & Time

**Program Name** Converting Date with specific format  
demo7.py

```
import pandas as pd

data = {
    "Product": ["Samsung", "iPhone", "Motorola"],
    "Status": ["Success", "Success", "Failed"],
    "Cost": [10000, 50000, 15000],
    "PurDate": ['02092019','13092019','21092019'],
}

df = pd.DataFrame(data)

df['PurDate'] = pd.to_datetime(df['PurDate'], format = '%d%m%Y')

print(df.head())
print()
print(df.dtypes)
```

**Output**

```
   Product    Status    Cost    PurDate
0  Samsung  Success  10000  2019-09-02
1    iPhone  Success  50000  2019-09-13
2  Motorola    Failed  15000  2019-09-21

Product          object
Status           object
Cost            int64
PurDate      datetime64[ns]
dtype: object
```

## Data Science – Pandas – DataFrame – Date & Time

**Program Name** Converting Date with specific format  
demo8.py

```
import pandas as pd

data = {
    "Product": ["Samsung", "iPhone", "Motorola"],
    "Status": ["Success", "Success", "Failed"],
    "Cost": [10000, 50000, 15000],
    "PurDate": ['02Sep2019','13Sep2019','21Sep2019'],
}

df = pd.DataFrame(data)

df['PurDate'] = pd.to_datetime(df['PurDate'])

print(df.head())
print()
print(df.dtypes)
```

### Output

```
   Product    Status    Cost      PurDate
0  Samsung  Success  10000  2019-09-02
1    iPhone  Success  50000  2019-09-13
2  Motorola    Failed  15000  2019-09-21

Product          object
Status           object
Cost            int64
PurDate    datetime64[ns]
dtype: object
```

## Data Science – Pandas – DataFrame – Date & Time

**Program Name** Converting Date with specific format  
demo9.py

```
import pandas as pd

data = {
    "Product": ["Samsung", "iPhone", "Motorola"],
    "Status": ["Success", "Success", "Failed"],
    "Cost": [10000, 50000, 15000],
    "PurDate": ['02Sep2019','13Sep2019','21Sep2019'],
}

df = pd.DataFrame(data)

df['PurDate'] = pd.to_datetime(df['PurDate'], format = '%d%b%Y')

print(df.head())
print()
print(df.dtypes)
```

### Output

```
      Product    Status   Cost     PurDate
0    Samsung  Success  10000  2019-09-02
1      iPhone  Success  50000  2019-09-13
2  Motorola    Failed  15000  2019-09-21

Product          object
Status           object
Cost            int64
PurDate    datetime64[ns]
dtype: object
```

## Data Science – Pandas – DataFrame – Date & Time

**Program Name** Converting Date with specific format  
demo10.py

```
import pandas as pd

data = {
    "Product": ["Samsung", "iPhone", "Motorola"],
    "Status": ["Success", "Success", "Failed"],
    "Cost": [10000, 50000, 15000],
    "PurDate": ['02-Sep-2019','13-Sep-2019','21-Sep-2019'],
}

df = pd.DataFrame(data)

df['PurDate'] = pd.to_datetime(df['PurDate'])

print(df.head())
print()
print(df.dtypes)
```

### Output

```
      Product    Status    Cost    PurDate
0    Samsung  Success  10000  2019-09-02
1     iPhone  Success  50000  2019-09-13
2  Motorola    Failed  15000  2019-09-21

Product           object
Status            object
Cost             int64
PurDate   datetime64[ns]
dtype: object
```

## Data Science – Pandas – DataFrame – Date & Time

**Program Name** Converting Date with specific format  
demo11.py

```
import pandas as pd

data = {
    "Product": ["Samsung", "iPhone", "Motorola"],
    "Status": ["Success", "Success", "Failed"],
    "Cost": [10000, 50000, 15000],
    "PurDate":
    ['20190902093000','20190913093000','20190921200000'],
}

df = pd.DataFrame(data)

df['PurDate'] = pd.to_datetime(df['PurDate'])

print(df.head())
print()
print(df.dtypes)
```

### Output

```
      Product    Status    Cost        PurDate
0   Samsung  Success  10000  2019-09-02 09:30:00
1     iPhone  Success  50000  2019-09-13 09:30:00
2  Motorola    Failed  15000  2019-09-21 20:00:00

Product           object
Status            object
Cost             int64
PurDate       datetime64[ns]
dtype: object
```

#### 4. NaT values

- ✓ Date column may contains missing values in pandas DataFrame.
- ✓ If Date column contains missing values then while converting into Date data type then we will get an error.
- ✓ By using errors = "coerce" keyword argument we can solve this problem.
- ✓ This argument converts Date column missing values into NaT (Not a Time) values.
  - Coerce errors i.e. convert un parse able date into **NaT** (Not a Time)

<b>Program Name</b>	Creating DataFrame demo12.py
---------------------	---------------------------------

```
import pandas as pd

data = {
    "Product": ["Samsung", "iPhone", "Motorola"],
    "Status": ["Success", "Success", "Failed"],
    "Cost": [10000, 50000, 15000],
    "PurDate": ['02-Sep-2019','Here date is missing','21-Sep-2019']
}

df = pd.DataFrame(data)

print(df.head())
print()
print(df.dtypes)
```

## Data Science – Pandas – DataFrame – Date & Time

### Output

```
      Product    Status    Cost          PurDate
0   Samsung   Success  10000  02-Sep-2019
1     iPhone   Success  50000  Here date is missing
2  Motorola   Failed  15000  21-Sep-2019

Product      object
Status       object
Cost        int64
PurDate     object
dtype: object
```

## Data Science – Pandas – DataFrame – Date & Time

**Program Name** Converting Date with specific format: **Error**  
demo13.py

```
import pandas as pd

data = {
    "Product": ["Samsung", "iPhone", "Motorola"],
    "Status": ["Success", "Success", "Failed"],
    "Cost": [10000, 50000, 15000],
    "PurDate": ['02-Sep-2019','Here date is missing','21-Sep-2019']
}

df = pd.DataFrame(data)

df['PurDate'] = pd.to_datetime(df['PurDate'])

print(df.head())
print()
print(df.dtypes)
```

### Output

TypeError: invalid string coercion to datetime for "Here date is missing" at position 1

## Data Science – Pandas – DataFrame – Date & Time

Program Name

to\_datetime(p) function

demo14.py

```
import pandas as pd

data = {
    "Product": ["Samsung", "iPhone", "Motorola"],
    "Status": ["Success", "Success", "Failed"],
    "Cost": [10000, 50000, 15000],
    "PurDate": ['02-Sep-2019','Here date is missing','21-Sep-2019']
}

df = pd.DataFrame(data)

df['PurDate'] = pd.to_datetime(df['PurDate'], errors="coerce")

print(df.head())
print()
print(df.dtypes)
```

### Output

```
   Product    Status    Cost      PurDate
0  Samsung  Success  10000  2019-09-02
1    iPhone  Success  50000        NaT
2  Motorola    Failed  15000  2019-09-21

Product          object
Status           object
Cost            int64
PurDate    datetime64[ns]
dtype: object
```

## 5. Selecting from start to end date values

- ✓ Based on requirement we can select specific dates, like
  - Start date to end date

**Program** Selecting Dataframe in between the dates  
**Name** demo15.py  
**File Name** sales7\_dates.csv

```
import pandas as pd

df = pd.read_csv('sales7_dates.csv', parse_dates=['Pur_Date'])

start = df['Pur_Date'] > '2019-1-1 01:00:00'
end = df['Pur_Date'] < '2019-1-1 05:00:00'

result = df[start & end]

print(result)
```

### Output

	Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost	Pur_Date
2	1025	Shahid	20	Bose SoundSport Headphones	65999	2019-01-01 02:00:00
3	1026	Veeru	3	Apple iPad 10.2-inch	63999	2019-01-01 03:00:00
4	1027	Venu	23	Google Phone	63999	2019-01-01 04:00:00

## 6. Access specific dates like last 20 days or 2 months or 2 years records

- ✓ Based on requirement we can get last 10 days, 20 days, 40 days, 1 month, 2 months, 3 months, 1 year and 2 years date data as well.
- ✓ We can also sort the dataframe by using `sort_values()`

**Program** Creating DataFrame

**Name** demo16.py

**File name** sales7\_dates.csv

```
import pandas as pd

df = pd.read_csv('sales7_dates.csv', parse_dates = ['Pur_Date'])

print(df.head())
```

### Output

	Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost	Pur_Date
0	1023	Venki	15	27in FHD Monitor	59000	2019-01-01 00:00:00
1	1024	Chaitanya	14	iPhone 11	69000	2019-01-01 01:00:00
2	1025	Shahid	20	Bose SoundSport Headphones	65999	2019-01-01 02:00:00
3	1026	Veeru	3	Apple iPad 10.2-inch	63999	2019-01-01 03:00:00
4	1027	Venu	23	Google Phone	63999	2019-01-01 04:00:00

## Data Science – Pandas – DataFrame – Date & Time

---

**Program** Accessing last 10 days records

**Name** demo17.py

**File name** sales7\_dates.csv

```
import pandas as pd

df = pd.read_csv('sales7_dates.csv', parse_dates = ['Pur_Date'])

new_df = df.set_index("Pur_Date")
days_10 = new_df.last("10D")

print(days_10)
```

**Output**

Pur_Date	Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost
2020-02-11 15:00:00	10782	Harsha	5	Google Phone	75000
2020-02-11 16:00:00	10783	Veeru	3	Google Phone	61000
2020-02-11 17:00:00	10784	Vinay	10	34in Ultrawide Monitor	69999
2020-02-11 18:00:00	10785	Jaya Chandra	21	ThinkPad Laptop	50000
2020-02-11 19:00:00	10786	Shahid	20	iPhone 11	51999
...	...	...	...	...	...
2020-02-21 10:00:00	11017	Kartek	4	28in Monitor	51999
2020-02-21 11:00:00	11018	Vinay	10	Apple Airpods Headphones	75999
2020-02-21 12:00:00	11019	Neelima	19	LG ThinQ Refrigerator	75999
2020-02-21 13:00:00	11020	Siddhu	18	iPhone 7s	65999
2020-02-21 14:00:00	11021	Tarun	11	34in Ultrawide Monitor	55000

[240 rows x 5 columns]

## Data Science – Pandas – DataFrame – Date & Time

---

**Program Name** Accessing last 40 days records  
**File name** demo18.py  
**sales7\_dates.csv**

```
import pandas as pd

df = pd.read_csv('sales7_dates.csv', parse_dates = ['Pur_Date'])

new_df = df.set_index("Pur_Date")
days_40 = new_df.last("40D")

print(days_40)
```

### Output

Pur_Date	Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost
2020-01-12 15:00:00	10052	Madhurima	7	Google Phone	51999
2020-01-12 16:00:00	10053	Pradhan	17	ThinkPad Laptop	63000
2020-01-12 17:00:00	10054	Venu	23	LG ThinQ Refrigerator	69000
2020-01-12 18:00:00	10055	Venki	15	ThinkPad Laptop	60000
2020-01-12 19:00:00	10056	Balaji	12	Flatscreen TV	65000
...	...	...	...	...	...
2020-02-21 10:00:00	11017	Karteek	4	20in Monitor	51999
2020-02-21 11:00:00	11018	Vinay	10	Apple Airpods Headphones	75999
2020-02-21 12:00:00	11019	Neelima	19	LG ThinQ Refrigerator	75999
2020-02-21 13:00:00	11020	Siddhu	18	iPhone 7s	65999
2020-02-21 14:00:00	11021	Tarun	11	34in Ultrawide Monitor	55000

[960 rows x 5 columns]

## Data Science – Pandas – DataFrame – Date & Time

---

**Program Name** Accessing last 1 month records  
**File name** demo19.py  
**sales7\_dates.csv**

```
import pandas as pd

df = pd.read_csv('sales7_dates.csv', parse_dates = ['Pur_Date'])

new_df = df.set_index("Pur_Date")
month_1 = new_df.last("1M")

print(month_1)
```

### Output

Pur_Date	Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost
2020-01-31 15:00:00	10518	Pradhan	17	LG ThinQ Refrigerator	65999
2020-01-31 16:00:00	10519	Shafi	25	Samsung Galaxy S20	63000
2020-01-31 17:00:00	10520	Veeru	3	34in Ultrawide Monitor	59000
2020-01-31 18:00:00	10521	Balaji	12	Macbook Pro Laptop	50000
2020-01-31 19:00:00	10522	Pradhan	17	iPhone 7s	51999
...	...	...	...	...	...
2020-02-21 10:00:00	11017	Karteek	4	20in Monitor	51999
2020-02-21 11:00:00	11018	Vinay	10	Apple Airpods Headphones	75999
2020-02-21 12:00:00	11019	Neelima	19	LG ThinQ Refrigerator	75999
2020-02-21 13:00:00	11020	Siddhu	18	iPhone 7s	65999
2020-02-21 14:00:00	11021	Tarun	11	34in Ultrawide Monitor	55000

[504 rows x 5 columns]

## Data Science – Pandas – DataFrame – Date & Time

---

**Program Name** Accessing last 1 year records  
**File name** demo20.py  
**File name** sales7\_dates.csv

```
import pandas as pd

df = pd.read_csv('sales7_dates.csv', parse_dates = ['Pur_Date'])

new_df = df.set_index("Pur_Date")
year_1 = new_df.last("1Y")

print(year_1)
```

### Output

Pur_Date	Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost
2019-12-31 15:00:00	9774	Veeru	3	27in FHD Monitor	65999
2019-12-31 16:00:00	9775	Shafi	25	27in 4K Gaming Monitor	65999
2019-12-31 17:00:00	9776	Shahid	20	Samsung Galaxy S20	50000
2019-12-31 18:00:00	9777	Venki	15	27in 4K Gaming Monitor	50000
2019-12-31 19:00:00	9778	Sumanth	22	Google Phone	65999
...	...	...	...	...	...
2020-02-21 10:00:00	11017	Karteek	4	20in Monitor	51999
2020-02-21 11:00:00	11018	Vinay	10	Apple Airpods Headphones	75999
2020-02-21 12:00:00	11019	Neelima	19	LG ThinQ Refrigerator	75999
2020-02-21 13:00:00	11020	Siddhu	18	iPhone 7s	65999
2020-02-21 14:00:00	11021	Tarun	11	34in Ultrawide Monitor	55000

[1248 rows x 5 columns]

## 7. Extract year, month, day from Date column

- ✓ Based on requirement we can get year, month, day, hour, minute from Date column.
- ✓ Sometimes it can be useful to break up a column of dates into components.

**Program Name** demo21.py  
**File name** sales7\_dates.csv

```
import pandas as pd

df = pd.read_csv('sales7_dates.csv', parse_dates = ['Pur_Date'])

df['year'] = df['Pur_Date'].dt.year
df['month'] = df['Pur_Date'].dt.month
df['day'] = df['Pur_Date'].dt.day

print(df.head())
```

### Output

	Order_Id	Customer_Name	Customer_Id	...	year	month	day
0	1023	Venki	15	...	2019	1	1
1	1024	Chaithanya	14	...	2019	1	1
2	1025	Shahid	20	...	2019	1	1
3	1026	Veeru	3	...	2019	1	1
4	1027	Venu	23	...	2019	1	1

[5 rows x 9 columns]

## Data Science – Pandas – DataFrame – Date & Time

---

**Program**      Breaking up the Date column value into multiple features

**Name**            demo22.py

**File name**    sales7\_dates.csv

```
import pandas as pd

df = pd.read_csv('sales7_dates.csv', parse_dates = ['Pur_Date'])

df['year'] = df['Pur_Date'].dt.year
df['month'] = df['Pur_Date'].dt.month
df['day'] = df['Pur_Date'].dt.day
df['hour'] = df['Pur_Date'].dt.hour
df['minute'] = df['Pur_Date'].dt.minute

print(df.head())
```

### Output

	Order_Id	Customer_Name	Customer_Id	Product_Name	...	month	day	hour	minute
0	1023	Venki	15	27in FHD Monitor	...	1	1	0	0
1	1024	Chaithanya	14	iPhone 11	...	1	1	1	0
2	1025	Shahid	20	Bose SoundSport Headphones	...	1	1	2	0
3	1026	Veeru	3	Apple iPad 10.2-inch	...	1	1	3	0
4	1027	Venu	23	Google Phone	...	1	1	4	0

[5 rows x 11 columns]

## 8. Encoding Days of the Week

- ✓ We can get the day of the week for each date by using pandas
  - Knowing the days names will helpful to understand the business flow, like we can compare total sales on specific day for the past three years.

**Program Name**

Encoding Days of the Week  
demo23.py

```
import pandas as pd

data = {
    'Product': ['Samsung', 'iPhone', 'Motorola'],
    'Status': ['Success', 'Success', 'Failed'],
    'Cost': [10000, 50000, 15000],
    'PurDate': ['2018-01-01','2018-01-02','2018-01-03'],
}

df = pd.DataFrame(data)
df['PurDate'] = pd.to_datetime(df['PurDate'])

print(df.head())
print()
print(df["PurDate"].dt.day_name())
```

**Output**

	Product	Status	Cost	PurDate
0	Samsung	Success	10000	2018-01-01
1	iPhone	Success	50000	2018-01-02
2	Motorola	Failed	15000	2018-01-03
0				Monday
1				Tuesday
2				Wednesday

Name: PurDate, dtype: object

## Data Science – Pandas – DataFrame – Date & Time

---

### Note

- ✓ The day of the week with Monday = 0, .... Sunday = 6

**Program Name** Encoding Days of the Week  
demo24.py

```
import pandas as pd

data = {
    'Product': ['Samsung', 'iPhone', 'Motorola'],
    'Status': ['Success', 'Success', 'Failed'],
    'Cost': [10000, 50000, 15000],
    'PurDate': ['2018-01-01','2018-01-02','2018-01-03'],
}

df = pd.DataFrame(data)
df['PurDate'] = pd.to_datetime(df['PurDate'])

print(df.head())
print()
print(df['PurDate'].dt.weekday)
```

### Output

	Product	Status	Cost	PurDate
0	Samsung	Success	10000	2018-01-01
1	iPhone	Success	50000	2018-01-02
2	Motorola	Failed	15000	2018-01-03
0	0			
1	1			
2	2			

Name: PurDate, dtype: int64

**21. Pandas – DataFrame - Concatenate Multiple csv files****Contents**

<b>1. Real time requirement .....</b>	<b>2</b>
<b>2. Loading csv files from all files .....</b>	<b>2</b>
<b>3. os module .....</b>	<b>2</b>
<b>4. listdir(p) function.....</b>	<b>3</b>
<b>5. filter(p1, p2) function .....</b>	<b>4</b>
<b>6. Concatenating all csv file.....</b>	<b>5</b>

**21. Pandas – DataFrame - Concatenate Multiple csv files****1. Real time requirement**

- ✓ Generally total data will be stored with multiple files
  - Yearly data: Jan sales, Feb sales, ....., Dec sales
- ✓ So, we need to concatenate all these monthly sales to bring year sales right

**2. Loading csv files from all files**

- ✓ The very first step is we need to access all files from specific folder.
- ✓ From that folder we need to capture only csv files.

**3. os module**

- ✓ os is a predefined module in python.
- ✓ By using this module we can load all files from the folder.

#### 4. **listdir(p)** function

- ✓ `listdir(p)` is a predefined function in `os` module
- ✓ This function we should access with `os` module name.
- ✓ By using this function we can get all file names from folder.
- ✓ This function returns all file names in list.

**Program Name** Accessing all files from the folder  
**Input file** demo1.py **daniel/jan\_sales.csv,....,dec\_sales.csv [15 files]**

```
import os

path = "./daniel"

all_files = os.listdir(path)

print(all_files)
```

#### Output

```
['apr_sales.csv', 'aug_sales.csv', 'dec_sales.csv', 'feb_sales.csv',
'fun.jpg', 'jan_sales.csv', 'jul_sales.csv', 'jun_sales.csv',
'mar_sales.csv', 'may_sales.csv', 'nov_sales.csv', 'oct_sales.csv',
'progress.txt', 'sep_sales.csv', 'status.xlsx']
```

## Data Science – Pandas – DataFrame – Concatenate Multiple Files

---

### 5. filter(p1, p2) function

- ✓ filter(p1, p2) is a predefined function in python
- ✓ We can access this function directly.
- ✓ By using this function we can apply Boolean logic and get results accordingly.

**Program Name** Accessing only csv files from folder  
**Input file** demo2.py  
**demo2.py**

```
import os

path = "./daniel"

all_files = os.listdir(path)

f = filter(lambda name: name.endswith('.csv'), all_files)
csv_files = list(f)

print(all_files)
print()
print(csv_files)
```

#### Output

```
['apr_sales.csv', 'aug_sales.csv', 'dec_sales.csv', 'feb_sales.csv',
'fun.jpg', 'jan_sales.csv', 'jul_sales.csv', 'jun_sales.csv',
'mar_sales.csv', 'may_sales.csv', 'nov_sales.csv', 'oct_sales.csv',
'progress.txt', 'sep_sales.csv', 'status.xlsx']
```

```
['apr_sales.csv', 'aug_sales.csv', 'dec_sales.csv', 'feb_sales.csv',
'jan_sales.csv', 'jul_sales.csv', 'jun_sales.csv', 'mar_sales.csv',
'may_sales.csv', 'nov_sales.csv', 'oct_sales.csv', 'sep_sales.csv']
```

## Data Science – Pandas – DataFrame – Concatenate Multiple Files

### 6. Concatenating all csv file

- ✓ Once we loaded all csv file then we can concatenate all csv file.
- ✓ Based on requirement by using pandas we can concatenate all csv files into one csv file

**Program Name** Concatenating all csv files  
**Input file** demo3.py  
**demo3.py** daniel/jan\_sales.csv,....,dec\_sales.csv [12 files]

```
import os
import glob
import pandas as pd

p = '.\daniel'

files = os.path.join(p, "*.csv")
csv_files = glob.glob(files)

result = (pd.read_csv(every) for every in csv_files)

df = pd.concat(result, ignore_index = True)

print(df)
df.to_csv("year.csv", index = False)
```

### Output

	Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost	Date
0	1320	Venu	23	LG ThinQ Refrigerator	55000	4/11/2019 9:00
1	1321	Jaya Chandra	21	Bose SoundSport Headphones	63000	4/13/2019 10:00
2	1322	Mallikarjun	13	Apple Airpods Headphones	69999	4/13/2019 11:00
3	1323	Siddhu	18	Samsung Galaxy S9 Plus	55000	4/14/2019 12:00
4	1324	Daniel	6	iPhone 8	55000	4/15/2019 13:00
..	...	...	...	...	...	...
103	1819	Daniel	6	LG Washing Machine	50000	9/3/2019 4:00
104	1820	Neelima	19	20in Monitor	55000	9/3/2019 5:00
105	1821	Karteek	4	20in Monitor	50000	9/3/2019 6:00
106	1822	Jaya Chandra	21	LG ThinQ Refrigerator	75999	9/3/2019 7:00
107	1823	Chaithanya	14	27in FHD Monitor	55000	9/3/2019 8:00
[108 rows x 6 columns]						

## Data Science – Pandas – DataFrame – Concatenate Multiple Files

**Program Name** Loading year.csv files

**Input file** demo4.py

year.csv

```
import pandas as pd

df = pd.read_csv("year.csv", parse_dates = ["Date"])

print(df)
```

**Output**

	Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost	Date
0	1320	Venu	23	LG ThinQ Refrigerator	55000	2019-04-11 09:00:00
1	1321	Jaya Chandra	21	Base SoundSport Headphones	63000	2019-04-13 10:00:00
2	1322	Mallikarjun	13	Apple Airpods Headphones	69999	2019-04-13 11:00:00
3	1323	Siddhu	18	Samsung Galaxy S9 Plus	55000	2019-04-14 12:00:00
4	1324	Daniel	6	iPhone 8	55000	2019-04-15 13:00:00

## Data Science – Data Analysis - EDA - Project

---

### DATA ANALYSIS - EDA – PROJECT

#### Contents

<b>1. EDA.....</b>	<b>2</b>
<b>2. Real time data .....</b>	<b>2</b>
<b>3. Results from EDA.....</b>	<b>3</b>

**DATA ANALYSIS - EDA – PROJECT****1. EDA**

- ✓ The full form of EDA means Explanatory Data Analysis.
- ✓ It is a process of performing initial investigations on data.
- ✓ By doing this kind of analysis, it helps us to discover patterns, standards.
- ✓ It's clearly explains the summary of statistics and graphical representations

**2. Real time data**

- ✓ e-Commerce domain
  - The dataset consists of transactional data with customers in different countries who make purchases from an online retail company based in the United Kingdom (UK).
  - Company: UK-based and online retail
  - Products for selling: Mainly all-occasion gifts
  - Customers: Most are wholesalers (local or international)
  - Transactions Period Data : From 1st Dec 2010 to 9th Dec 2011 (One year)



### 3. Results from EDA

- ✓ Highest number of orders from specific country
- ✓ Highest money spent on purchase specific country
- ✓ Top 5 countries place the highest number of orders
- ✓ Top 5 countries spent the money on purchase products
- ✓ Highest sales on specific Month
- ✓ There are no transactions on between dates.
- ✓ Sales increase days
- ✓ Sales decrease days
- ✓ Highest number of orders hours and etc

#### Dataset explanation

- ✓ InvoiceNo (invoice\_num) : A number assigned to each transaction
- ✓ StockCode (stock\_code) : Product code
- ✓ Description (description) : Product name
- ✓ Quantity (quantity) : Number of products purchased for each transaction
- ✓ InvoiceDate (invoice\_date) : Timestamp for each transaction
- ✓ UnitPrice (unit\_price) : Product price per unit
- ✓ CustomerID (cust\_id) : Unique identifier each customer
- ✓ Country (country) : Country name

**1. DATA VISUALIZATION PART – 1****Contents**

<b>1. Data Visualization.....</b>	<b>2</b>
1.1. Example in words .....	3
<b>2. Common data visualization techniques .....</b>	<b>3</b>
<b>3. Advantages .....</b>	<b>3</b>
<b>4. Few examples .....</b>	<b>4</b>
<b>5. Matplotlib.....</b>	<b>7</b>
<b>6. Line chart .....</b>	<b>8</b>
<b>7. Bar Chart .....</b>	<b>14</b>
<b>8. Histogram .....</b>	<b>18</b>
<b>9. Pie Chart .....</b>	<b>19</b>
<b>10. Scatter Plot .....</b>	<b>22</b>
<b>11. Box Plots .....</b>	<b>24</b>
<b>12. Heatmap .....</b>	<b>26</b>

**1. DATA VISUALIZATION PART – 1****1. Data Visualization**

- ✓ **Data Visualization** is the process of converting raw information (text, numbers, or symbols) into a graphical representation.
- ✓ If we visualize the data then it is very easy to understand.

**Best quote**

- ✓ A picture gives more meaningful information than thousand words

### 1.1. Example in words

- ✓ Reaching to target



### 2. Common data visualization techniques

- ✓ Bar charts
- ✓ Pie charts
- ✓ Line graphs
- ✓ Box plot
- ✓ Scatter plot & etc

### 3. Advantages

- ✓ To identify **trends**, such as whether sales increasing or decreasing.
- ✓ To identify **patterns**, such as during weekend more sales.
- ✓ To identify **relationships**, such as if we study more hours then we will get good marks.
- ✓ To identify **frequency**, such as how often a product is purchased in a specific area & etc

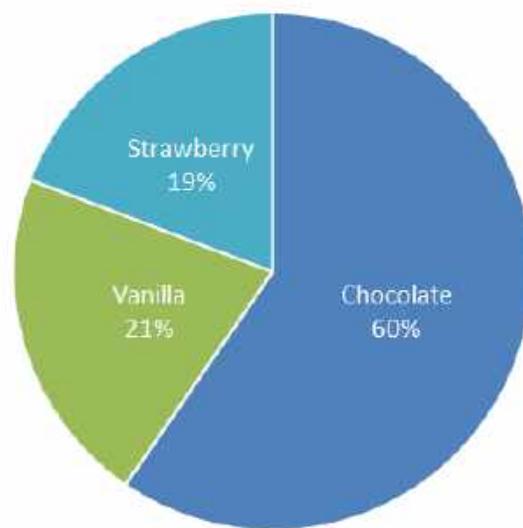
#### 4. Few examples

What's your favorite ice cream flavor?

Flavor	Count
Chocolate	62
Vanilla	22
Strawberry	20

What's your favorite ice cream flavor?

Based on 104 survey responses

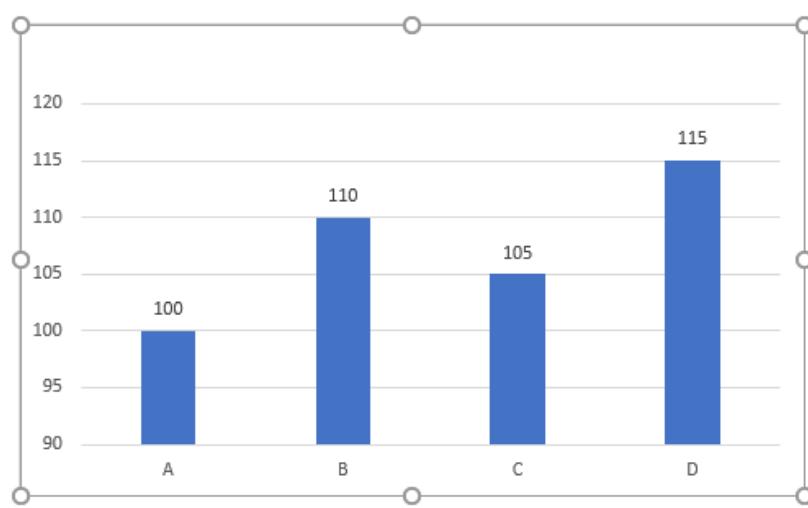


## Data Science – Data Visualization

Bar chart



Group	Value
A	100
B	110
C	105
D	115



## Data Science – Data Visualization

	Actual	Forecast
Jan	100K	
Feb	115K	
Mar	121K	
Apr	150K	
May	137K	
Jun	152K	152K
Jul		167K
Aug		184K
Sep		202K
Oct		223K
Nov		245K
Dec		269K

**Line Chart**



### 5. Matplotlib

- ✓ Matplotlib is the most popular plotting library in python.
- ✓ Using matplotlib we can plot the data.

#### Environment

- ✓ We can install this library by using pip command.

##### matplotlib installation

```
pip install matplotlib
```

## 6. Line chart

- ✓ A line chart or line graph is a type of chart which displays information as a series of data points connected by straight line
- ✓ A line chart is often used to visualize a trend in data over intervals of time.

**Program Name** Create a simple line chart  
demo1.py

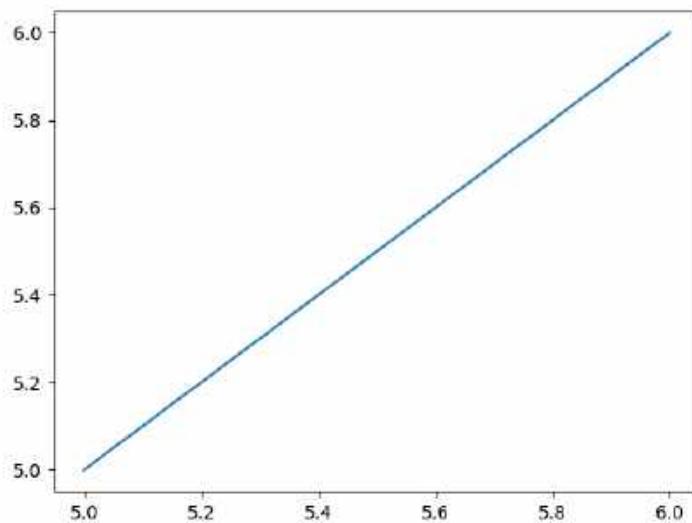
```
import matplotlib.pyplot as plt

x = [5, 6]
y = [5, 6]

plt.plot(x, y)

plt.show()
```

### Output



**Program Name** Create a simple line chart  
demo2.py

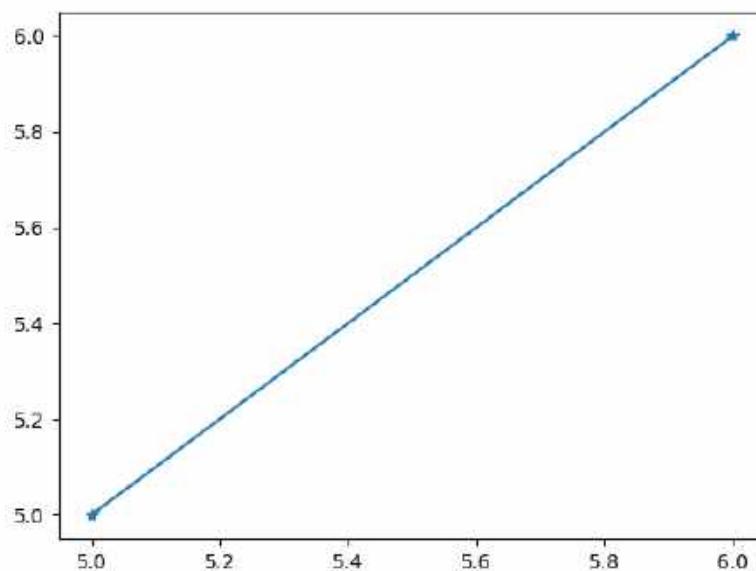
```
import matplotlib.pyplot as plt

x = [5, 6]
y = [5, 6]

plt.plot(x, y, marker='*')

plt.show()
```

**Output**



## Data Science – Data Visualization

**Program Name** Create a simple line chart  
demo3.py

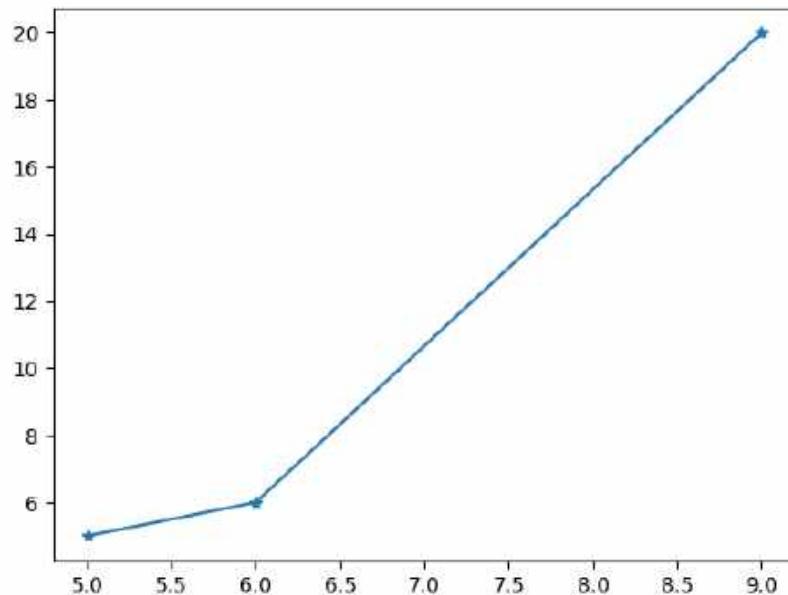
```
import matplotlib.pyplot as plt

x = [5, 6, 9]
y = [5, 6, 20]

plt.plot(x, y, marker='*')

plt.show()
```

**Output**



**Program Name** Create a simple line chart and title  
demo4.py

```
import matplotlib.pyplot as plt

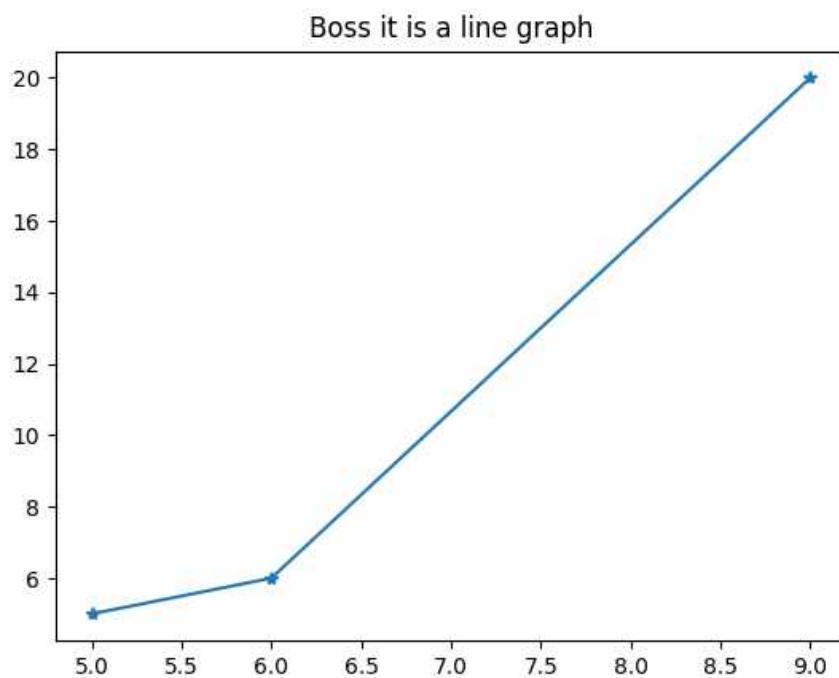
x = [5, 6, 9]
y = [5, 6, 20]

plt.title("Boss it is a line graph")

plt.plot(x, y, marker='*')

plt.show()
```

**Output**



## Data Science – Data Visualization

### 6.1. Labelling the axes

- ✓ We can label **x axis** and **y axis** by using xlabel and ylabel

**Program Name** Create a simple line chart and giving title and labelling  
demo5.py

```
import matplotlib.pyplot as plt

x = [5, 6, 9]
y = [5, 6, 20]

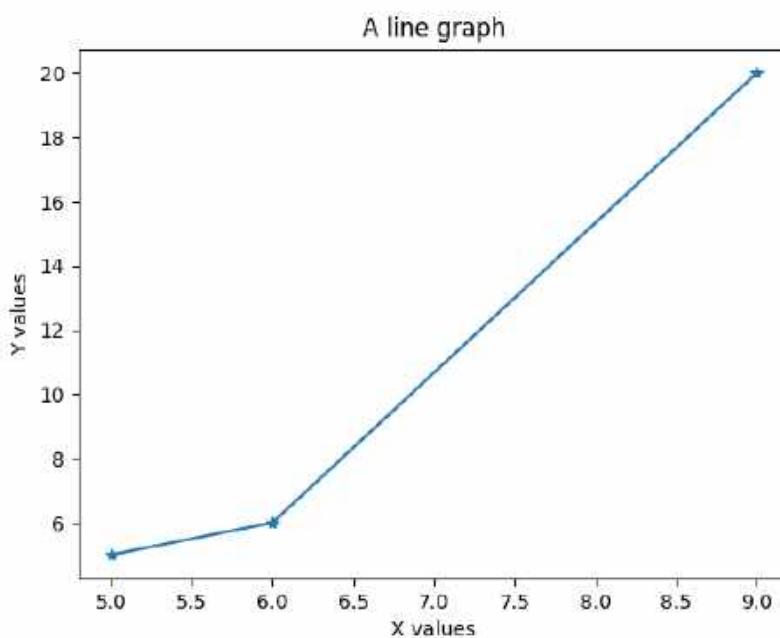
plt.title("A line graph")

plt.xlabel("X values")
plt.ylabel("Y values")

plt.plot(x, y, marker = '*')

plt.show()
```

#### Output



## Data Science – Data Visualization

**Program Name** Create two lines in single chart  
demo6.py

```
import matplotlib.pyplot as plt

x = [5, 6, 9]
y = [5, 6, 20]
p = [10, 20, 25]

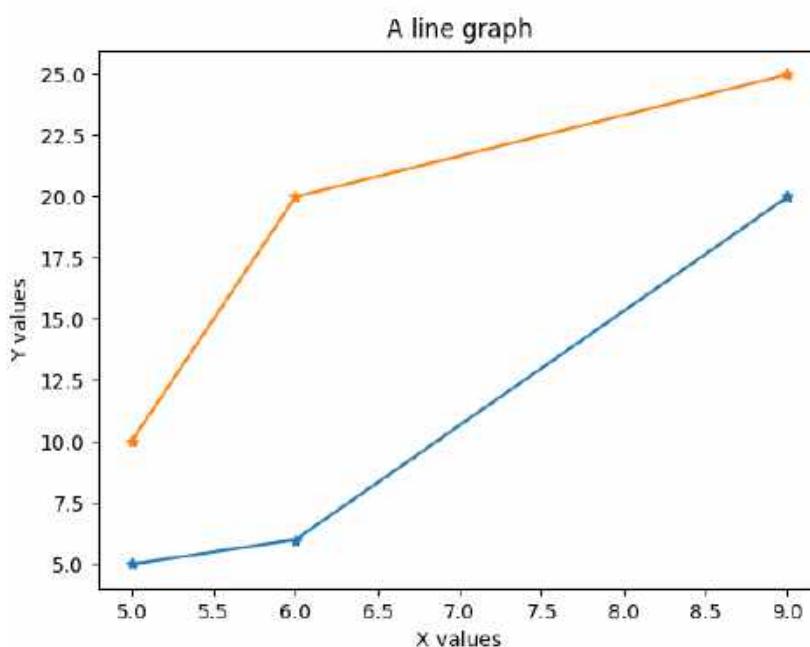
plt.title("A line graph")

plt.xlabel("X values")
plt.ylabel("Y values")

plt.plot(x, y, marker = '*')
plt.plot(x, p, marker = '*')

plt.show()
```

**Output**



## 7. Bar Chart

- ✓ The bar graph is the graphical representation of categorical data.

**Program Name** Creating bar chart  
demo7.py

```
import matplotlib.pyplot as plt

months = ["Jan", "Feb", "Mar", "Apr", "May", "June", "July", "Aug",
          "Sep", "Oct", "Nov", "Dec"]
sales = [23, 45, 56, 78, 213, 45, 78, 89, 99, 100, 101, 130]

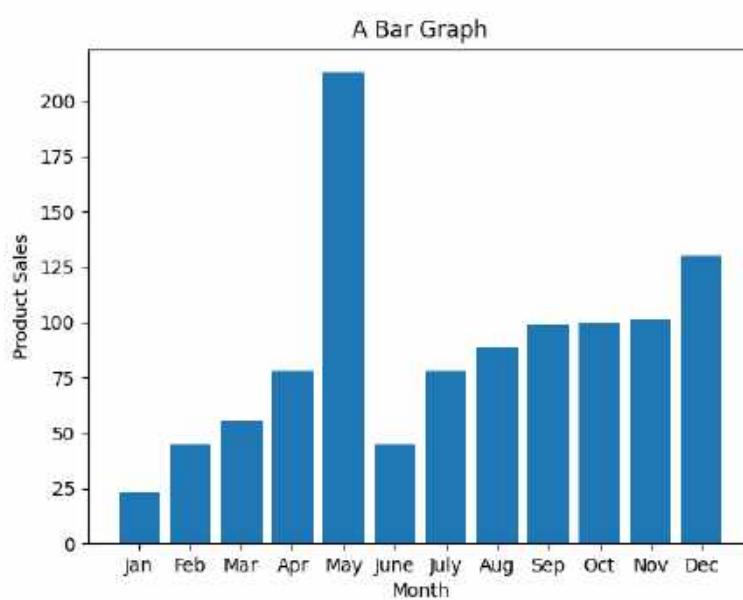
plt.title('A Bar Graph')

plt.xlabel('Month')
plt.ylabel('Product Sales')

plt.bar(months, sales)

plt.show()
```

### Output



## Data Science – Data Visualization

**Program Name** Creating horizontal bar chart  
demo8.py

```
import matplotlib.pyplot as plt

months = ["Jan", "Feb", "Mar", "Apr", "May", "June", "July", "Aug",
"Sep", "Oct", "Nov", "Dec"]
sales = [23, 45, 56, 78, 213, 45, 78, 89, 99, 100, 101, 130]

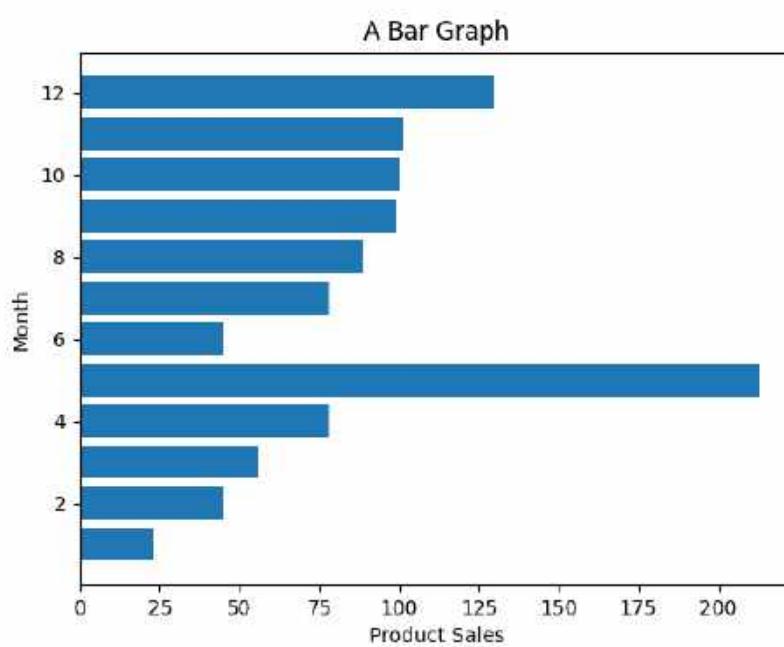
plt.title('A Bar Graph')

plt.xlabel('Product Sales')
plt.ylabel('Month')

plt.barh(months, sales)

plt.show()
```

**Output**



## Data Science – Data Visualization

**Program Name** Creating horizontal bar chart

**File name** demo9.py

**sales11.csv**

```
import matplotlib.pyplot as plt
import pandas as pd

df = pd.read_csv("sales11.csv")

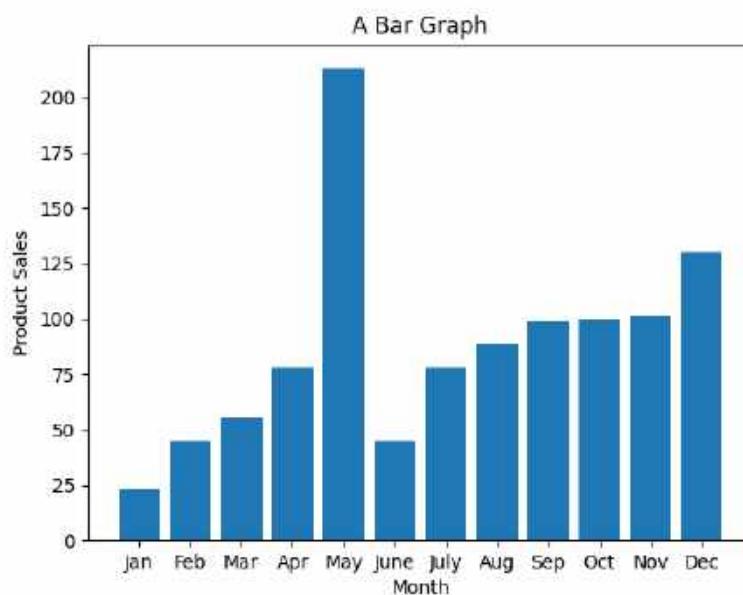
plt.title('A Bar Graph')

plt.xlabel('Month')
plt.ylabel('Product Sales')

plt.bar(df.month, df.sales)

plt.show()
```

**Output**



## Data Science – Data Visualization

**Program Name** Creating bar chart  
demo10.py

```
import matplotlib.pyplot as plt

months = ["Jan", "Feb", "Mar", "Apr", "May", "June", "July", "Aug",
"Sep", "Oct", "Nov", "Dec"]
sales = [23, 45, 56, 78, 213, 45, 78, 89, 99, 100, 101, 130]

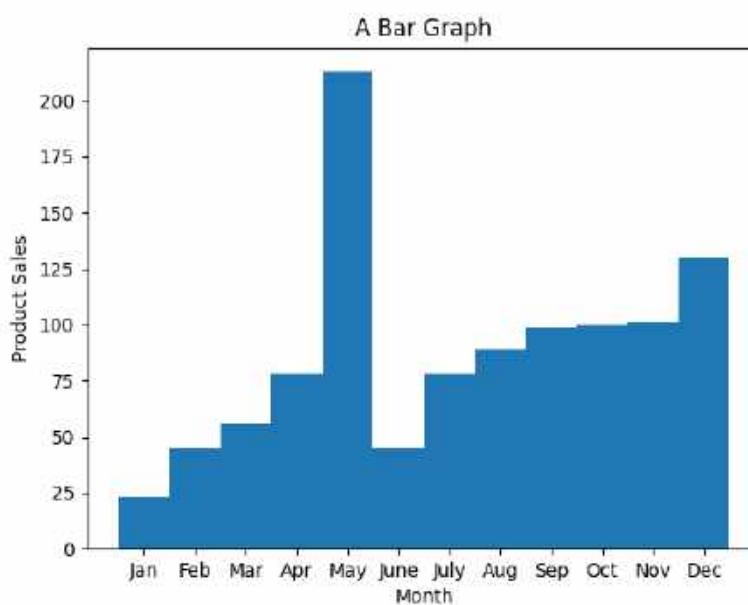
plt.title('A Bar Graph')

plt.xlabel('Month')
plt.ylabel('Product Sales')

plt.bar(months, sales, width = 1.0)

plt.show()
```

**Output**



## 8. Histogram

- ✓ A histogram is the graphical representation of quantitative data.
- ✓ This displays the frequency/count of numerical data in bars.

**Program Name**

Creating histogram  
demo11.py

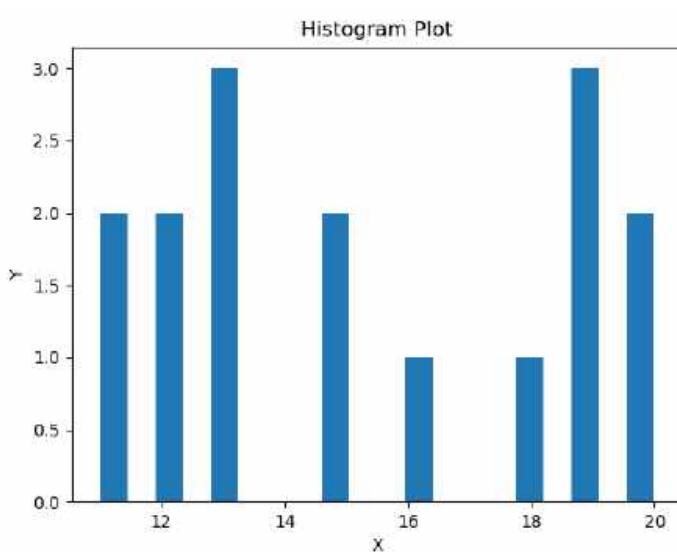
```
import matplotlib.pyplot as plt

data = [12, 15, 13, 20, 19, 20, 11, 19, 11, 12, 19, 13, 15, 16, 18, 13]

plt.xlabel("X")
plt.ylabel("Y")
plt.title("Histogram Plot")

plt.hist(data, bins = 20)
plt.show()
```

### Output



## 9. Pie Chart

- ✓ This is a circular plot that has been divided into slices displaying numerical proportions.
- ✓ Every slice in the pie chart shows the proportion of the element to the whole.
- ✓ A large category means that it will occupy a larger portion of the pie chart.

**Program Name** Creating pie chart  
demo12.py

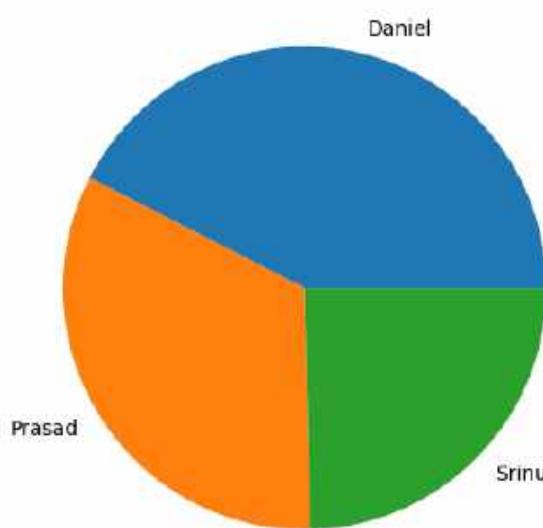
```
import matplotlib.pyplot as plt

students = ['Daniel', 'Prasad', 'Srinu']
points = [62, 48, 36]

plt.pie(points, labels = students)

plt.axis('equal')
plt.show()
```

### Output



## Data Science – Data Visualization

**Program Name** Creating pie chart  
demo13.py

```
import matplotlib.pyplot as plt

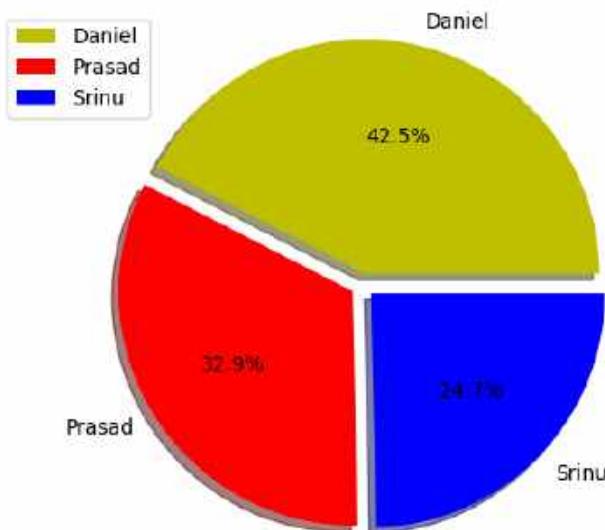
students = ['Daniel', 'Prasad', 'Srinu']
points = [62, 48, 36]

c = ['y', 'r', 'b']

plt.pie(points, labels = students, colors = c , shadow = True,
explode = (0.05, 0.05, 0.05), autopct = '%1.1f%%')

plt.axis('equal')
plt.legend()
plt.show()
```

### Output



### 9.1. Attributes

- ✓ The first parameter to the function is the list of numbers for every category.
  - labels attribute:
    - A list of categories separated by commas is then passed as the argument to labels attribute.
  - colors attribute:
    - To provide the color for every category.
    - To create shadows around the various categories in pie chart.
    - To split each slice of the pie chart into its own.

## 10. Scatter Plot

- ✓ In scatter plot each value in the data set is represented by a dot.
- ✓ By using this plot we can understand the relationship between two variables.

**Program Name** Creating Scatter plot  
demo14.py

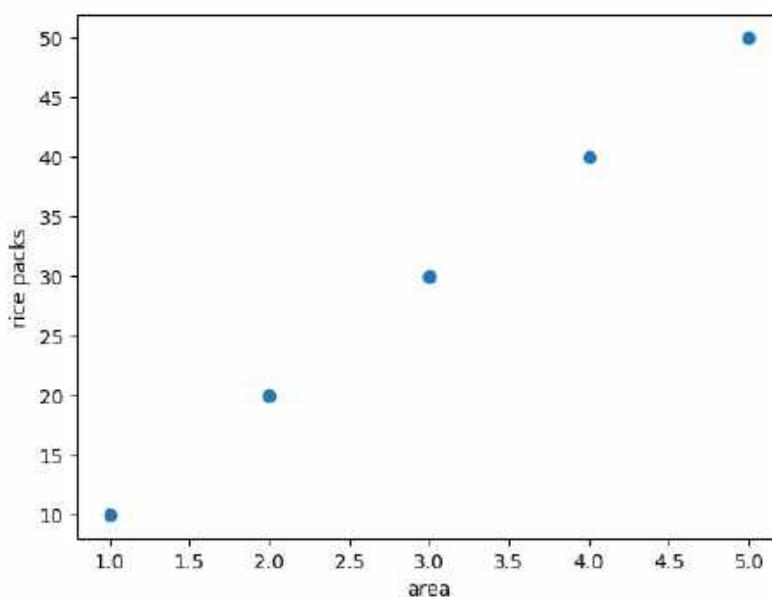
```
import matplotlib.pyplot as plt

area = [1, 2, 3, 4, 5]
rice_packs = [10, 20, 30, 40, 50]

plt.xlabel('area')
plt.ylabel('rice packs')

plt.scatter(area, rice_packs)
plt.show()
```

### Output



**Program Name** Creating Scatter plot  
demo15.py

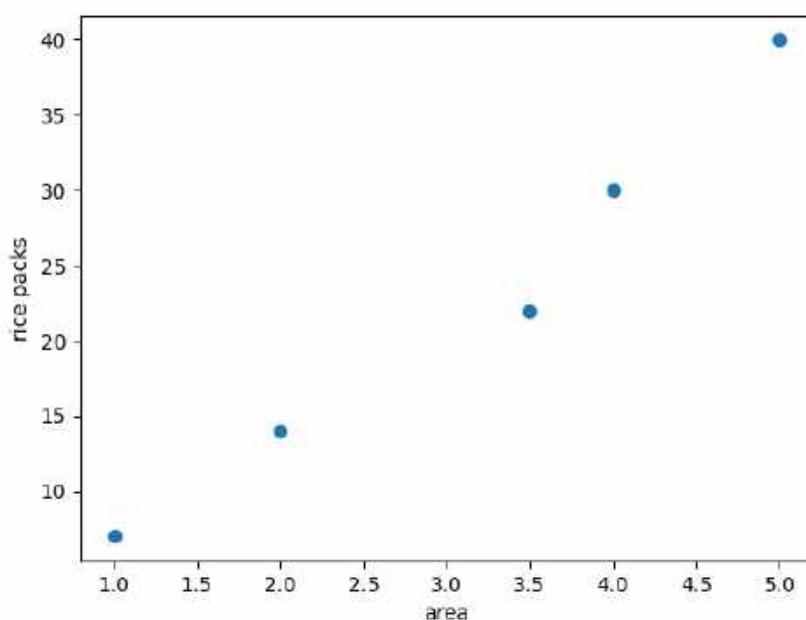
```
import matplotlib.pyplot as plt

area = [1, 2, 3.5, 4, 5]
rice_packs = [7, 14, 22, 30, 40]

plt.xlabel('area')
plt.ylabel('rice packs')

plt.scatter(area, rice_packs)
plt.show()
```

**Output**



## 11. Box Plots

- ✓ Box plots help us measure how well data in a dataset is distributed.
- ✓ The graph shows the maximum, minimum, median, first quartile and third quartiles of the dataset.

### 11.1. Use Box plots

- ✓ Use a boxplot when you need to get the overall statistical information about the data distribution.
- ✓ It is a good tool for detecting outliers in a dataset.

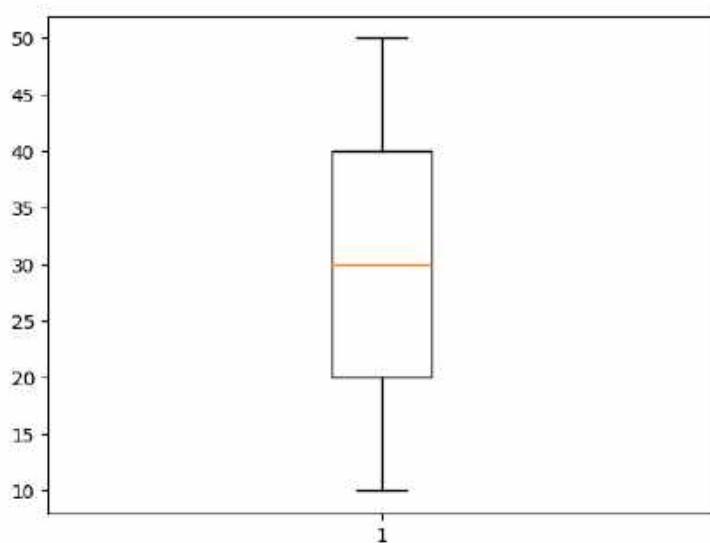
**Program Name** Creating box plot  
demo16.py

```
import matplotlib.pyplot as plt

data = [10, 20, 30, 40, 50]

plt.boxplot(data)
plt.show()
```

#### Output



## 11.2. Box plot explanation

- ✓ The line dividing the box into two shows the median of the data.
- ✓ The end of the box represents the upper quartile (75%) while the start of the box represents the lower quartile (25%).
- ✓ The part between the upper quartile and the lower quartile is known as the Inter Quartile Range (IQR) and helps in approximating 50% of the middle data.

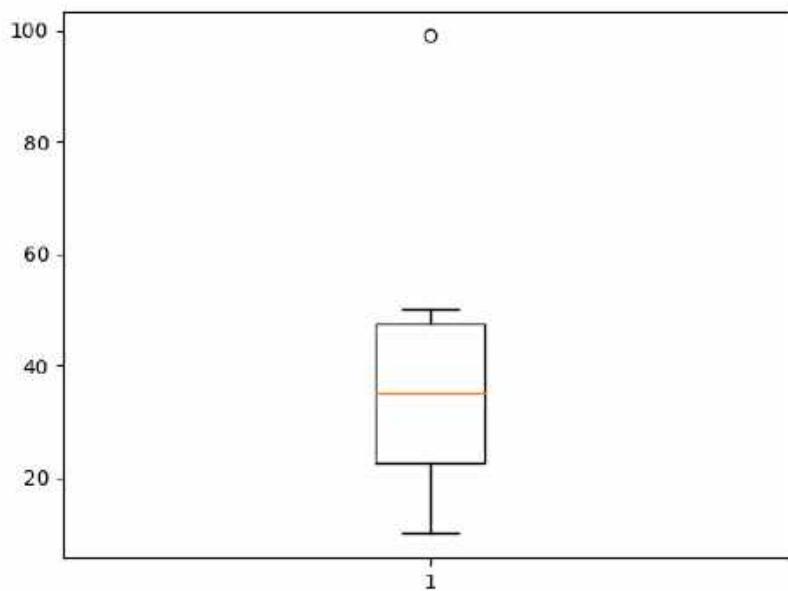
**Program Name** Creating box plot with outlier  
demo17.py

```
import matplotlib.pyplot as plt

data = [10, 20, 30, 40, 50, 90]

plt.boxplot(data)
plt.show()
```

### Output



## 12. Heatmap

- ✓ A heatmap is a method of data visualization that plots data by replacing numbers with colours.
- ✓ If it is representing with color then it is very easy to understand patterns between different values in the dataset.
- ✓ It is used to visualize data in a two-dimensional format as a coloured map so that different colour variations represent different patterns between features.

### 12.1. How to understand?

- ✓ A heatmap visualizes the relationship between features as a colour palette.
- ✓ While analysing a heatmap, always remember that **dark shades** represent a **high degree** of linear relationship between features and **light shades** represent a **low degree** of linear relationship between features.

## Data Science – Data Visualization

**Program Name** Creating box plot  
demo18.py

```
import matplotlib.pyplot as plt
import pandas as pd

d = {
    "Apple": [10, 20, 30, 40],
    "Orange": [7, 14, 21, 28],
    "Banana": [55, 15, 8, 12],
    "Pear": [15, 14, 1, 8]
}

i = ['Basket1', 'Basket2', 'Basket3', 'Basket4']

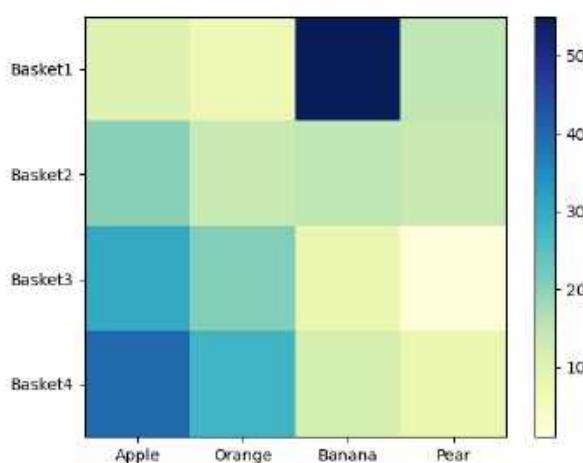
df = pd.DataFrame(d, index = i)

plt.imshow(df, cmap = "YIGnBu")
plt.colorbar()

plt.xticks(range(len(df)), df.columns)
plt.yticks(range(len(df)), df.index)

plt.show()
```

### Output



# Data Visualization Session at GRIET

---

## Data Visualization

### Contents

<b>1. Data Introduction .....</b>	4
<b>2. What is Data? .....</b>	4
<b>3. Data Visualization.....</b>	5
1.1. Example in words .....	5
<b>4. Advantages .....</b>	5
4.1. Common data visualization techniques .....	6
4.2. With data visualization.....	6
<b>5. Real time Examples .....</b>	7
5.1. Data & Visual: Netflix Subscribers.....	7
5.2. Data & Visual: Top Social Media Usage Statistics 2024 .....	9
5.3. Data & Visual: Top ChatGPT Statistics (2024) .....	10
<b>6. Process behind the Data Visualization .....</b>	14
6.1. Data collection .....	14
6.2. Data cleaning.....	15
6.3. Data analysis .....	16
6.4. Chose the right visualization .....	17
6.5. Creating visual representation.....	18
6.6. Review and Iterative .....	18
<b>7. Types of Data Visualization.....</b>	19
7.1. Basic Charts.....	19
7.2. Statistical Visualizations .....	20
7.3. Advanced Charts .....	21
7.4. Interactive Visualizations .....	21
7.5. Textual Visualizations.....	21
<b>8. Gestalt principles for Data Visualization .....</b>	22
8.1. Proximity .....	22
8.2. Similarity .....	22
8.3. Closure .....	22
8.4. Continuity.....	22
<b>9. Visualization reference model .....</b>	23
9.1. Data Layer .....	23
9.2. Processing Layer.....	23

## Data Visualization Session at GRIET

---

9.3. Visualization Layer .....	24
9.4. Interaction Layer .....	24
9.5. Presentation Layer .....	24
9.6. Feedback Layer .....	25
9.7. Deployment Layer .....	25
<b>10. Data visualizations by the number of variables .....</b>	<b>26</b>
10.1. Univariate Visualizations.....	26
10.2. Bivariate Visualizations .....	26
10.3. Multivariate Visualizations.....	27
<b>11. Matplotlib .....</b>	<b>28</b>
<b>12. Line chart .....</b>	<b>29</b>
12.1. Labelling the axes.....	33
<b>13. Bar Chart .....</b>	<b>35</b>
<b>14. Histogram .....</b>	<b>39</b>
<b>15. Pie Chart .....</b>	<b>40</b>
15.1. Attributes .....	42
<b>16. Scatter Plot .....</b>	<b>43</b>
<b>17. Box Plots .....</b>	<b>45</b>
17.1. Use Box plots.....	45
17.2. Box plot explanation .....	46
<b>18. Heatmap .....</b>	<b>47</b>
18.1. How to understand? .....	47

## Data Visualization Session at GRIET

### Data Visualization



## Data Visualization Session at GRIET

---

### Data Visualization



### 1. Data Introduction

- ✓ Currently we are all living in the data world.
- ✓ Everyone is communicating by using devices and social networks, due to this huge amount of data is generating.
- ✓ All applications are generating data.
  - Ecommerce applications.
  - Banking applications.
  - Social network etc.

### 2. What is Data?

- ✓ Data is a collection of Facts.
- ✓ Facts can be,
  - Numbers
  - Alphabets
  - Alphanumeric
  - Symbols
  - Images
  - Audio
  - Video & etc

## Data Visualization Session at GRIET

### 3. Data Visualization

- ✓ **Data Visualization** is the process of converting data into a graphical representation.
- ✓ If we visualize the data then it is very easy to understand.

#### Best quote

- ✓ A picture gives more meaningful information than thousand words

#### 1.1. Example in words

- ✓ Reaching to target



### 4. Advantages

- ✓ To identify **trends**, such as whether sales increasing or decreasing.
- ✓ To identify **patterns**, such as during weekend more sales.
- ✓ To identify **relationships**, such as if we study more hours then we will get good marks.
- ✓ To identify **frequency**, such as how often a product is purchased in a specific area & etc

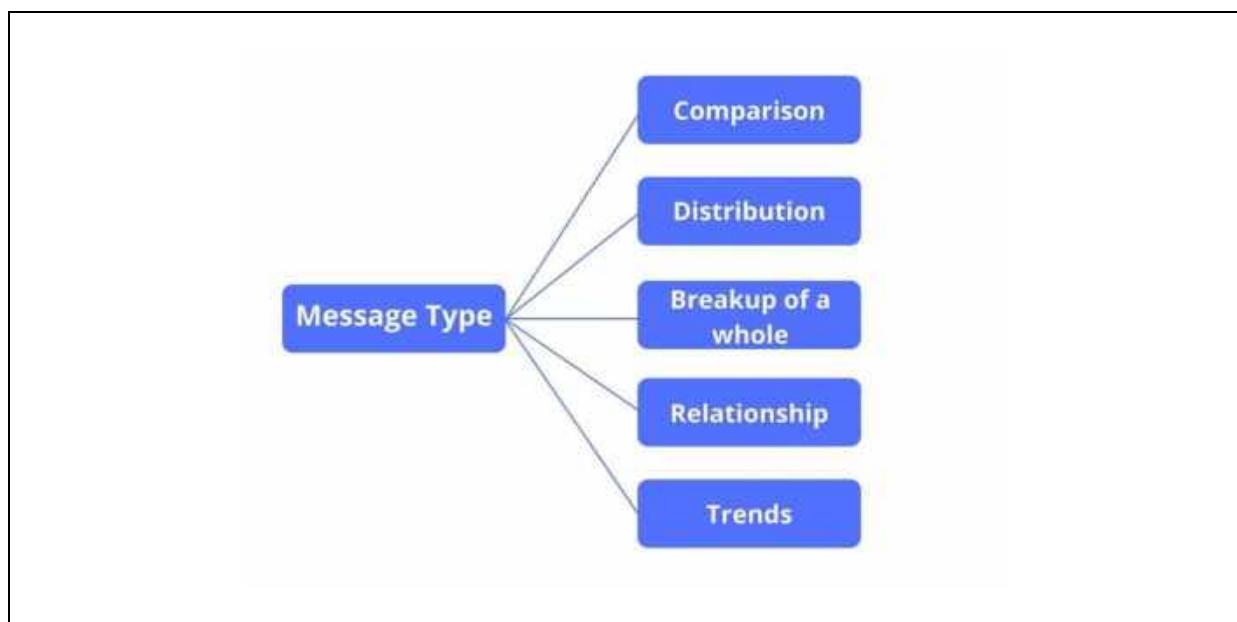
## Data Visualization Session at GRIET

### 4.1. Common data visualization techniques

- ✓ Bar charts
- ✓ Pie charts
- ✓ Line graphs
- ✓ Box plot
- ✓ Scatter plot & etc

### 4.2. With data visualization

- ✓ We are sharing message/information to end users.



## Data Visualization Session at GRIET

---

### 5. Real time Examples

#### 5.1. Data & Visual: Netflix Subscribers

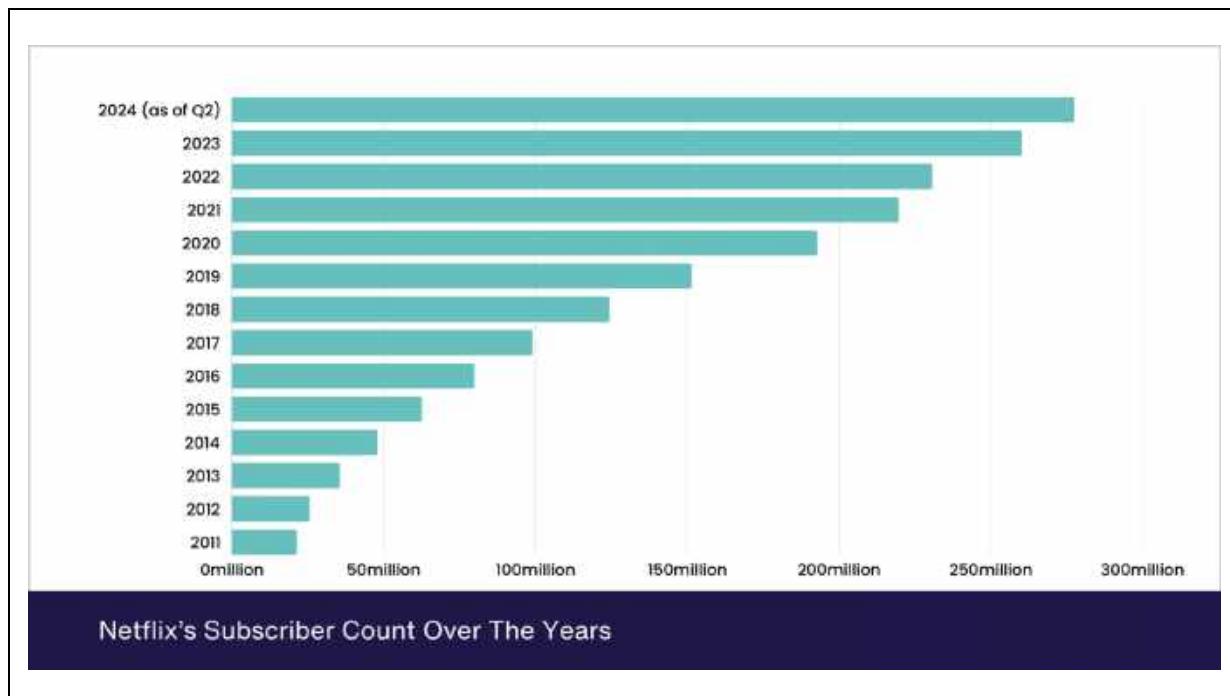
Ref Link: <https://www.demandsage.com/netflix-subscribers/>

#### Top Netflix Statistics At A Glance

- Netflix has 277.65 million subscribers as of 2024.
- Netflix generated \$18.93 billion in revenue in the first half of 2023.
- Women make up 51%, while Males make up 49% of all Netflix users.
- Netflix is preferred by 47% of Americans over other streaming platforms and is responsible for 8.4% of the screen time in the country.
- Around 65% of Netflix consumers are from outside of the United States of America & Canada.
- Netflix customers spend 62.1 minutes each day on average consuming content.

Year	Netflix Subscribers
2024 (as of Q2)	277.65 million
2023	260.28 million
2022	230.7 million
2021	219.7 million
2020	192.9 million
2019	151.5 million
2018	124.3 million
2017	99 million
2016	79.9 million
2015	62.7 million
2014	47.9 million
2013	35.6 million
2012	25.7 million
2011	21.5 million

## Data Visualization Session at GRIET



## Data Visualization Session at GRIET

---

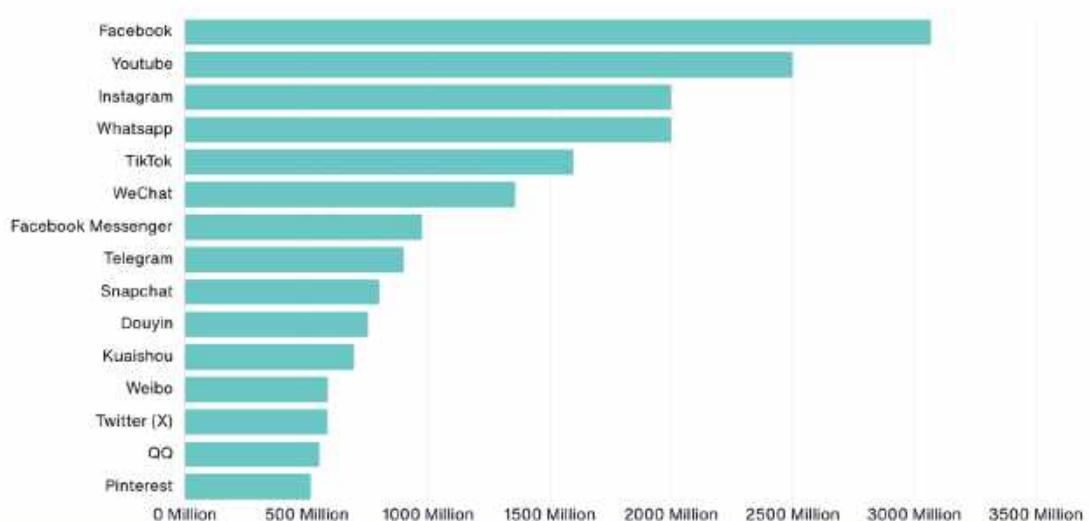
### 5.2. Data & Visual: Top Social Media Usage Statistics 2024

Ref Link: <https://www.demandsage.com/social-media-users/>

#### Top Social Media Usage Statistics 2024

- There are 5.17 billion social media users globally.
- 68% of the people in the United States use social media, approximately 308 million people.
- Facebook is the biggest social media platform, with over 3.07 billion users.
- A typical social media user interacts with 6.7 social media platforms.
- On average, users spend 2 hours and 20 minutes daily on Social media platforms.
- China has the highest number of social media users, with 1.07 billion users in the country.

#### Most Popular Social Media Platforms



Biggest Social Media Platforms Globally

## Data Visualization Session at GRIET

### 5.3. Data & Visual: Top ChatGPT Statistics (2024)

Ref Link: <https://www.demandsage.com/chatgpt-statistics/>

#### Top ChatGPT Statistics (2024)

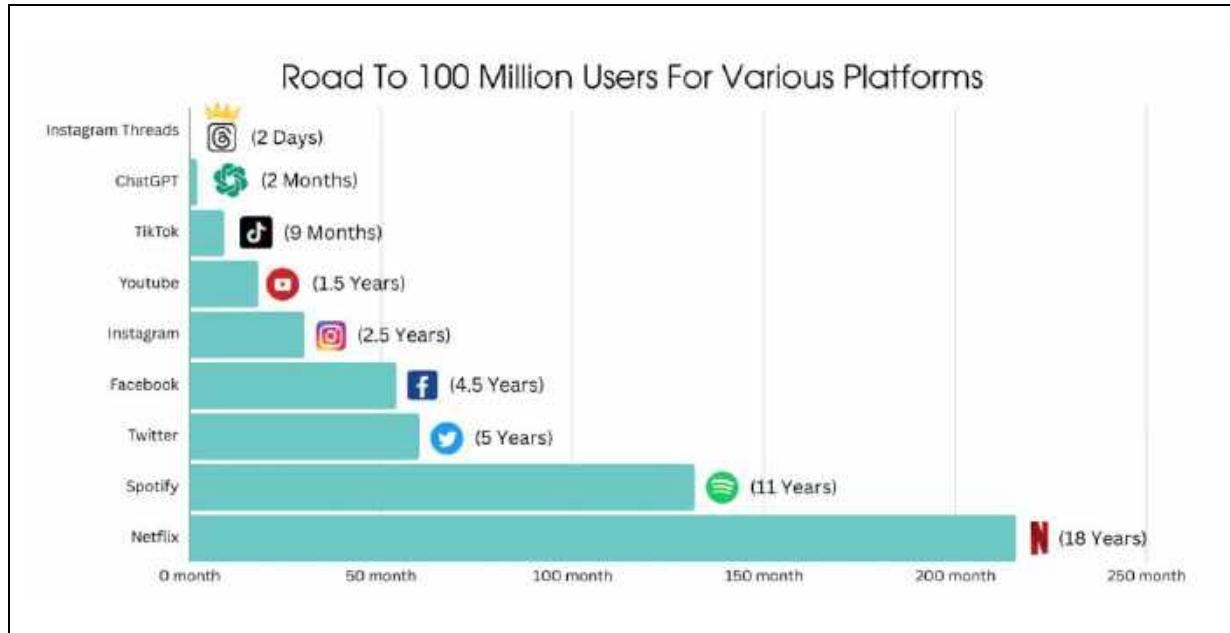
- ChatGPT has over 200 million weekly active users as of September 2024.
- Around 77.2 million monthly active users in the US.
- ChatGPT Plus is used by 7.7 million people worldwide.
- ChatGPT reached 1 million users in just five days after its launch.
- More than 92% of Fortune 500 companies are using ChatGPT.
- ChatGPT is forecasted to generate a revenue of \$1 billion in 2024.
- OpenAI spends approximately \$700,000 every day to operate ChatGPT.
- ChatGPT gets over 1.54 billion page visits every month on average.

#### ChatGPT User Demographics

ChatGPT Gender Split



## Data Visualization Session at GRIET

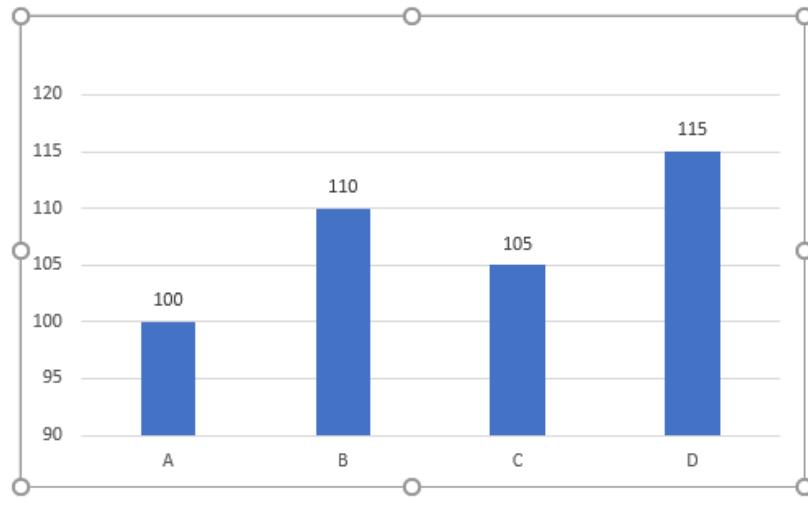


## Data Visualization Session at GRIET

**Bar chart**



Group	Value
A	100
B	110
C	105
D	115



## Data Visualization Session at GRIET

### Sales Data

	Actual	Forecast
Jan	100K	
Feb	115K	
Mar	121K	
Apr	150K	
May	137K	
Jun	152K	152K
Jul		167K
Aug		184K
Sep		202K
Oct		223K
Nov		245K
Dec		269K

### Line Chart



## Data Visualization Session at GRIET

### 6. Process behind the Data Visualization

#### Steps

- ✓ Data collection
- ✓ Data cleaning
- ✓ Data analysis
- ✓ Chose the right visualization
- ✓ Creating visual representation
- ✓ Review and Iterative

#### 6.1. Data collection

- ✓ Gather/collect the relevant data from difference sources.



## Data Visualization Session at GRIET

### 6.2. Data cleaning

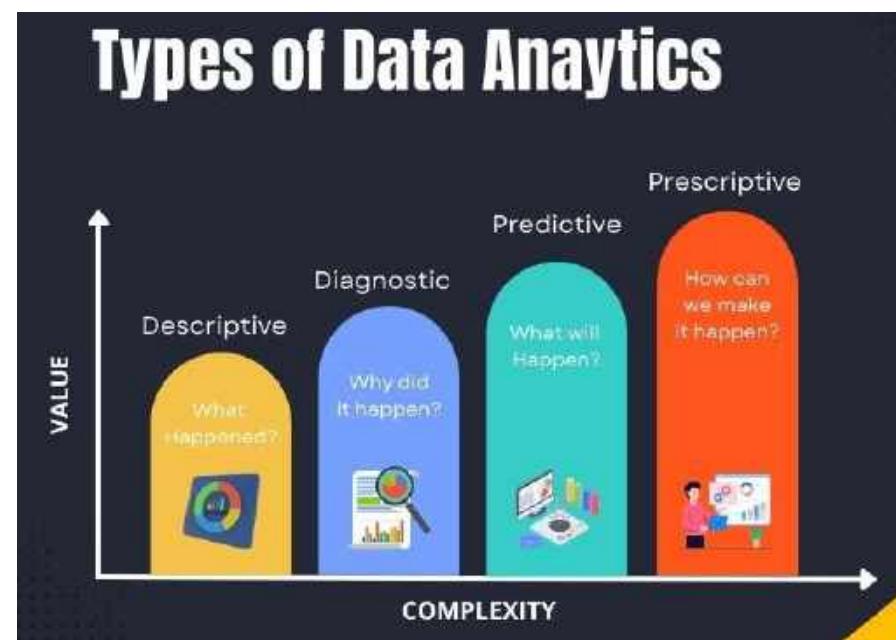
- ✓ Ensuring accuracy and consistency.



## Data Visualization Session at GRIET

### 6.3. Data analysis

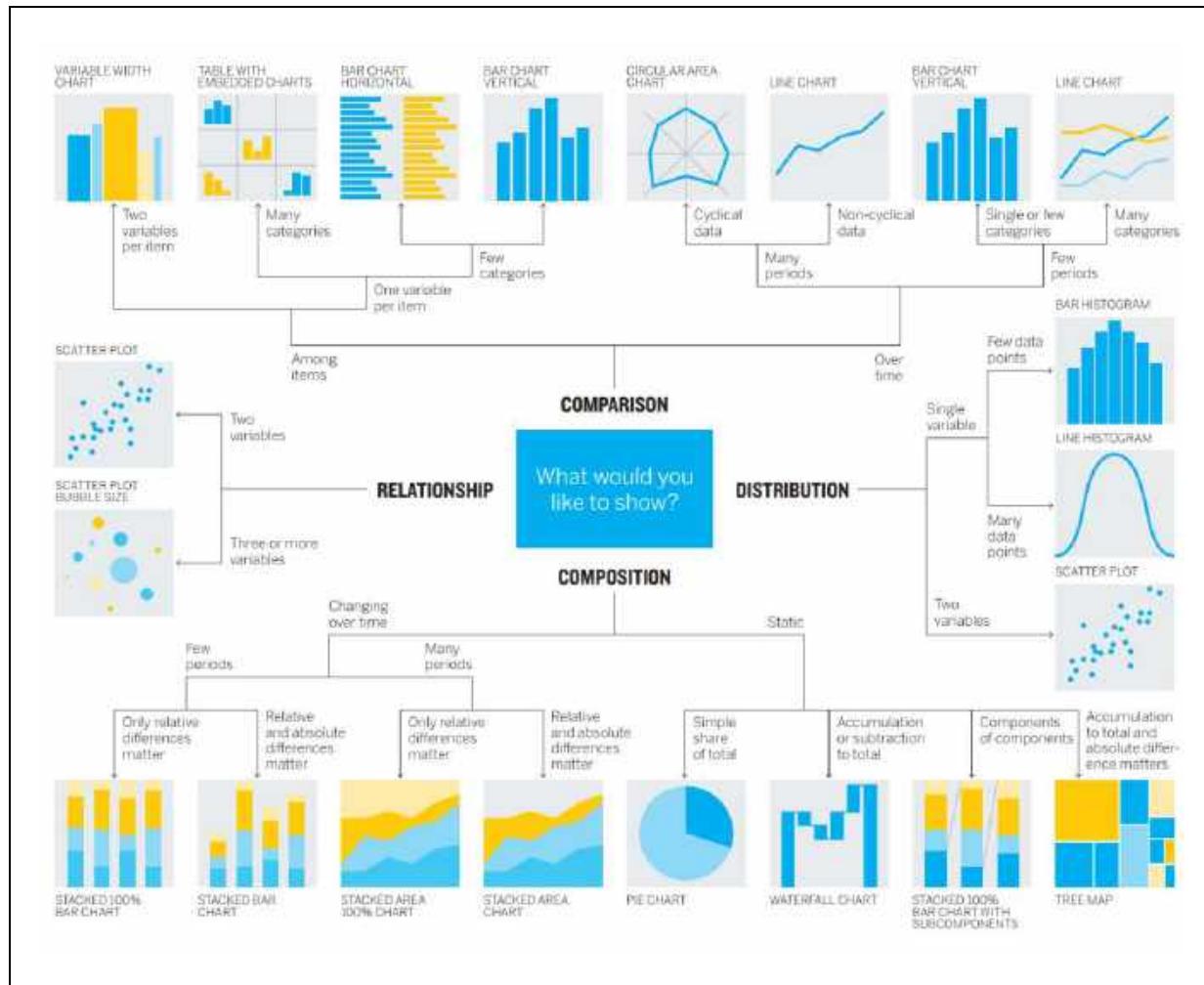
- ✓ Exploring data to identify trends and patterns.



## Data Visualization Session at GRIET

### 6.4. Choose the right visualization

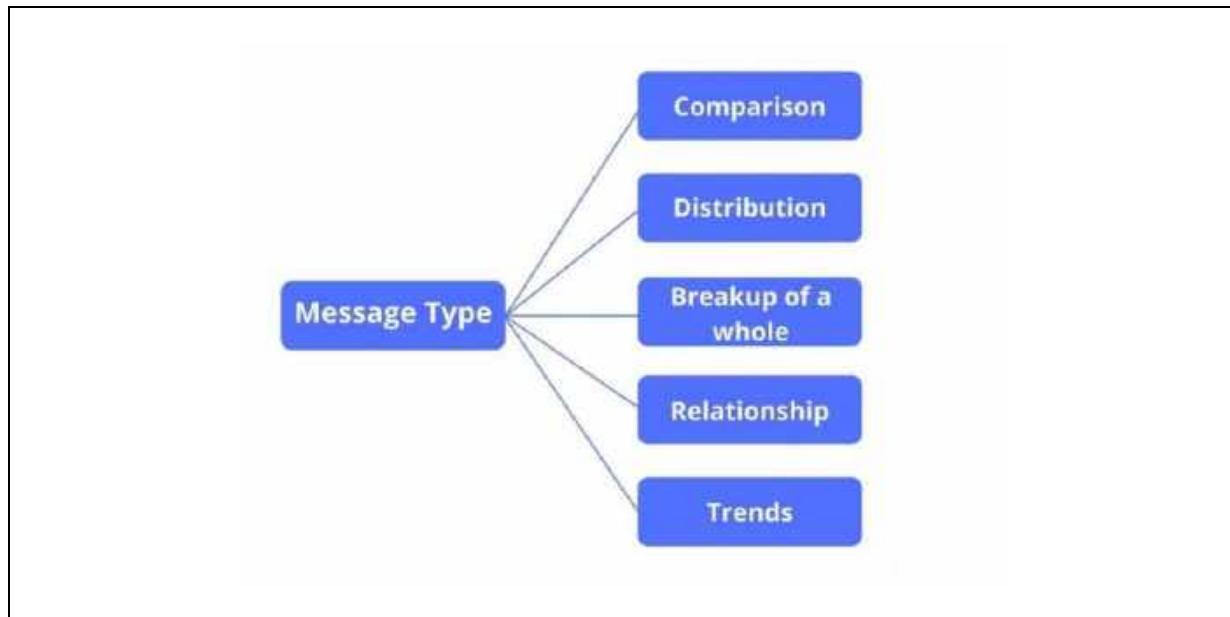
- ✓ Selecting appropriate charts, graphs, or maps based on requirement.



## Data Visualization Session at GRIET

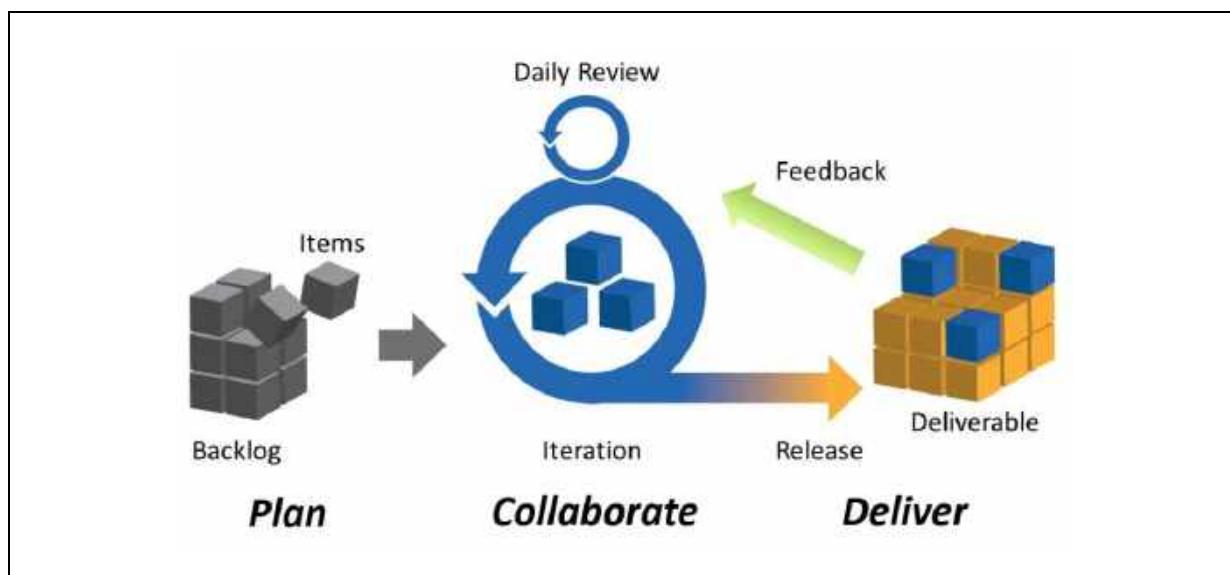
### 6.5. Creating visual representation

- ✓ Create data visualization to share information effectively



### 6.6. Review and Iterative

- ✓ Testing the visualization for clarity and effectiveness, making adjustments if needed.

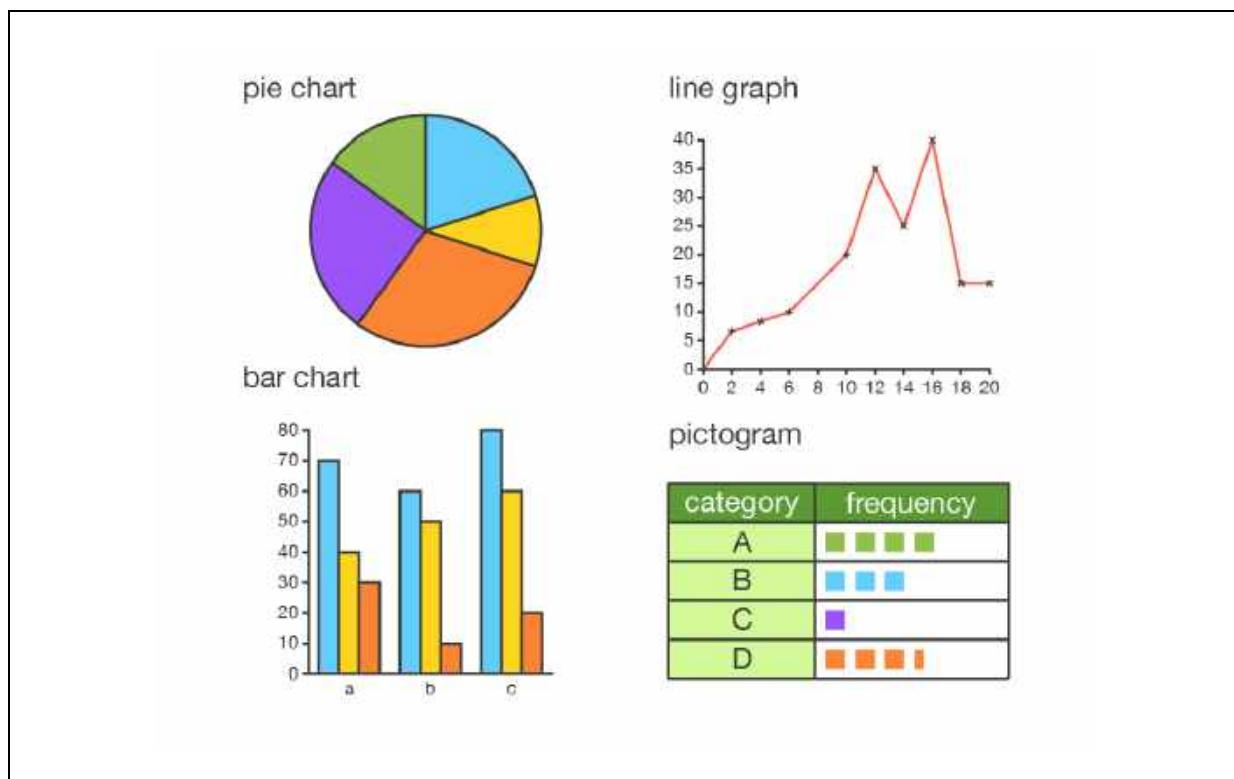


## Data Visualization Session at GRIET

### 7. Types of Data Visualization

#### 7.1. Basic Charts

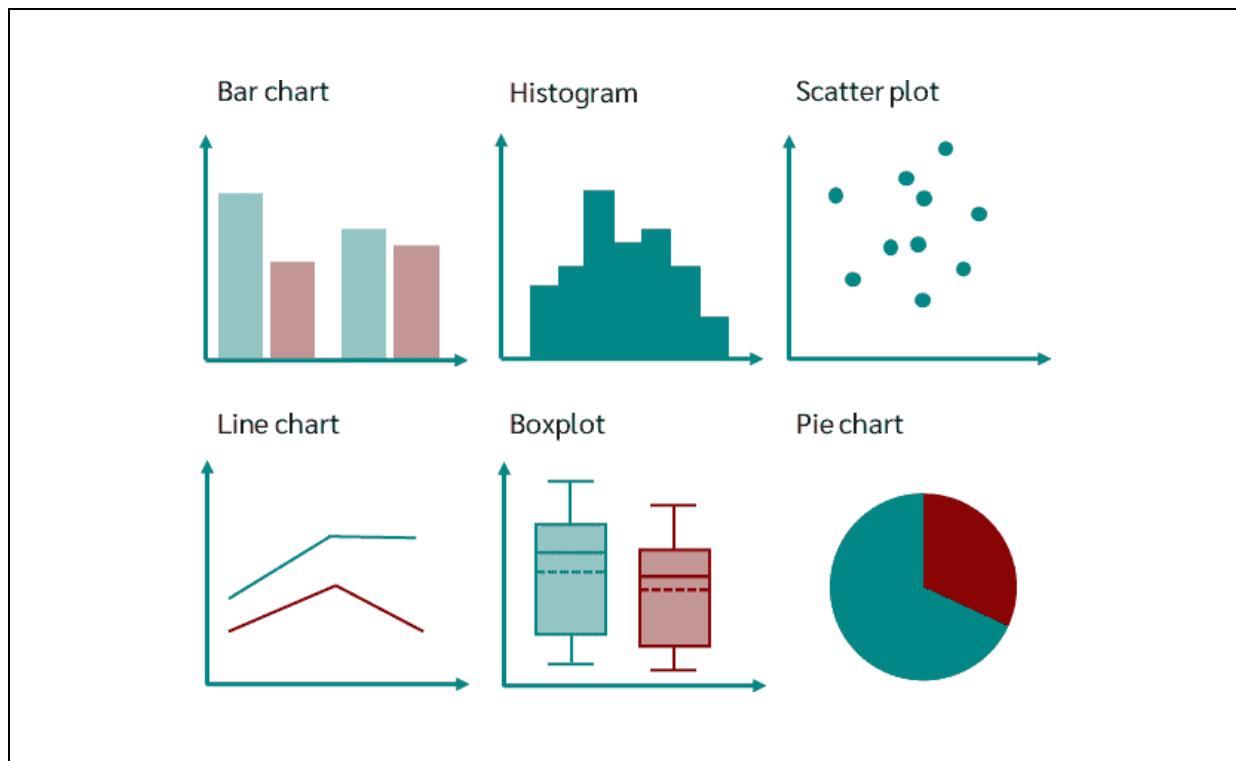
- ✓ **Bar Chart:** Compares quantities across different categories.
  - **Column Chart:** Vertical version of the bar chart.
- ✓ **Line Chart:** Shows trends over time or continuous data.
- ✓ **Pie Chart:** Displays proportions of a whole; best for limited categories.



## Data Visualization Session at GRIET

### 7.2. Statistical Visualizations

- ✓ **Histogram:** Represents the distribution of numerical data.
- ✓ **Box Plot:** Summarizes data distribution.
- ✓ **Scatter Plot:** Shows the relationship between two variables.



## Data Visualization Session at GRIET

---

### 7.3. Advanced Charts

- ✓ **Heatmap:** Uses color to represent data values in a matrix format.
- ✓ **Bubble Chart:** A scatter plot with an added dimension represented by the size of the bubbles.

### 7.4. Interactive Visualizations

- ✓ **Dashboards:** Combines multiple visualizations to provide an overview of key metrics.
- ✓ **Data Explorer:** Allows users to interact with data, filtering and adjusting parameters.

### 7.5. Textual Visualizations

- ✓ **Word Cloud:** Displays text data, highlighting frequently used terms.
- ✓ **Tag Cloud:** Similar to a word cloud, often used for categorizing data.

## Data Visualization Session at GRIET

---

### 8. Gestalt principles for Data Visualization

#### 8.1. Proximity

- ✓ Grouping related items together.
  - Example: In a scatter plot, showing sales data for different products, grouping all electronics together.

#### 8.2. Similarity

- ✓ Using similar shapes or colours to indicate relationships.
  - Example: Using same colors for regions in a bar chart (e.g., blue for East, green for West) helps viewers easily compare sales.

#### 8.3. Closure

- ✓ Completing incomplete shapes to create a whole.
  - Example: A line graph, showing monthly temperature changes can use dotted lines for predictions, allowing viewers to intuitively connect the dots and grasp trends.

#### 8.4. Continuity

- ✓ Following lines and patterns to guide the viewer's eye.
  - Example: A line graph, a stock price graph clearly shows trends, allowing viewers to easily spot increases and decreases over time.

## Data Visualization Session at GRIET

---

### 9. Visualization reference model

- ✓ A visualization reference model works as a framework for understanding the components and process in Data Visualization.

#### Steps

- ✓ Data Collection
- ✓ Data Processing
- ✓ Visualization Design
- ✓ User Interaction
- ✓ Presentation
- ✓ Feedback
- ✓ Deployment

#### 9.1. Data Layer

- ✓ **Data Sources**
  - Identify where the data is coming from (databases, APIs, spreadsheets).
- ✓ **Data Preparation**
  - Data Cleaning, transforming, and aggregating data to ensure it's ready for visualization.

#### 9.2. Processing Layer

- ✓ **Data Analysis:**
  - Apply statistical methods or algorithms to extract insights from the data.
- ✓ **Data Reduction:**
  - Simplifying data by selecting key variables or filtering out noise to focus on important information.

## Data Visualization Session at GRIET

---

### 9.3. Visualization Layer

✓ **Visual Encoding:**

- Choosing how to represent data visually (e.g., using colors, shapes, sizes).

✓ **Chart Types:**

- Selecting appropriate visualization types based on the data and insights (e.g., bar charts, line graphs, scatter plots).

✓ **Design Principles:**

- Applying best practices for layout, color schemes, labelling, and accessibility.

### 9.4. Interaction Layer

✓ **Interactivity:**

- Incorporating features like tooltips, filters, and zooming to allow users to explore data dynamically.

✓ **User Experience:**

- Ensuring that the interaction is intuitive and enhances the understanding of the data.

### 9.5. Presentation Layer

✓ **Contextual Information:**

- Providing background, legends, and annotations to help interpret the visualizations.

✓ **Storytelling:**

- Structuring the visualizations to convey a narrative and guide the viewer through the insights.

## Data Visualization Session at GRIET

---

### 9.6. Feedback Layer

- ✓ **User Testing:**
  - Gathering input from users to assess clarity, effectiveness, and engagement.
  
- ✓ **Iteration:**
  - Refining the visualizations based on feedback to improve understanding and impact.

### 9.7. Deployment Layer

- ✓ **Distribution:**
  - Sharing the visualizations through dashboards, reports, or web applications.
  
- ✓ **Maintenance:**
  - Regularly updating the visualizations to reflect new data and insights.

## Data Visualization Session at GRIET

---

### 10. Data visualizations by the number of variables

- ✓ We can divide the data visualization based on the number of variables.

#### Types

- ✓ Univariate Visualizations
- ✓ Bivariate Visualizations
- ✓ Multivariate Visualizations

#### 10.1. Univariate Visualizations

- ✓ These visualizations focus on a single variable, allowing you to explore its distribution and key statistics.
  - **Histograms:** Show the distribution of a continuous variable.
  - **Bar Charts:** Represent count of categories in a categorical variable.
  - **Box Plots:** Summarize the distribution, median, quartiles, and outliers of a single variable.
  - **Pie Charts:** Explains the proportions of categories in a categorical variable.
  - **Density Plots:** Display the distribution of a continuous variable in a smoothed format.

#### 10.2. Bivariate Visualizations

- ✓ These visualizations explore the relationship between two variables, helping to identify correlations or patterns.
  - **Scatter Plots:** Show the relationship between two continuous variables.
  - **Line Graphs:** By using this we can display how one variable changes in relation to another variable
  - **Grouped Bar Charts:** Compare the values of a categorical variable across different groups.
  - **Heatmaps:** Represent the intensity of a variable across two dimensions (e.g., correlation matrices).
  - **Bubble Charts:** Extend scatter plots by adding a third variable represented by the size of the bubbles.

## Data Visualization Session at GRIET

---

### 10.3. Multivariate Visualizations

- ✓ These involve three or more variables and can include
  - **3D Scatter Plots:** Visualize relationships among three continuous variables.
  - **Parallel Coordinates:** By using this we can understand how several variables relate to one another.
  - **Facet Grids:** Display multiple plots in a grid, each representing a subset of data based on one or more categorical variables.

## Data Visualization Session at GRIET

### 11. Matplotlib

- ✓ Matplotlib is a powerful and widely-used plotting library for Python
- ✓ Using matplotlib we can plot the data.

#### Environment

- ✓ We can install this library by using pip command.

#### matplotlib installation

```
pip install matplotlib
```

## Data Visualization Session at GRIET

### 12. Line chart

- ✓ A line chart or line graph is a type of chart which displays information as a series of data points connected by straight line
- ✓ A line chart is often used to visualize a trend in data over intervals of time.

**Program Name** Create a simple line chart  
demo1.py

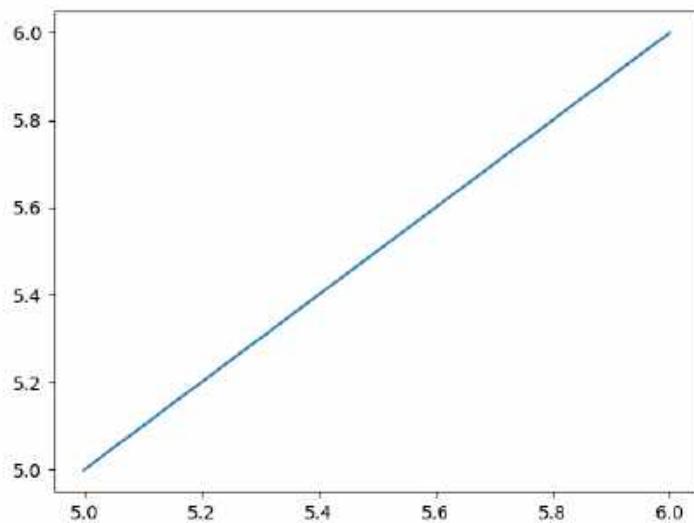
```
import matplotlib.pyplot as plt

x = [5, 6]
y = [5, 6]

plt.plot(x, y)

plt.show()
```

#### Output



## Data Visualization Session at GRIET

**Program Name** Create a simple line chart  
demo2.py

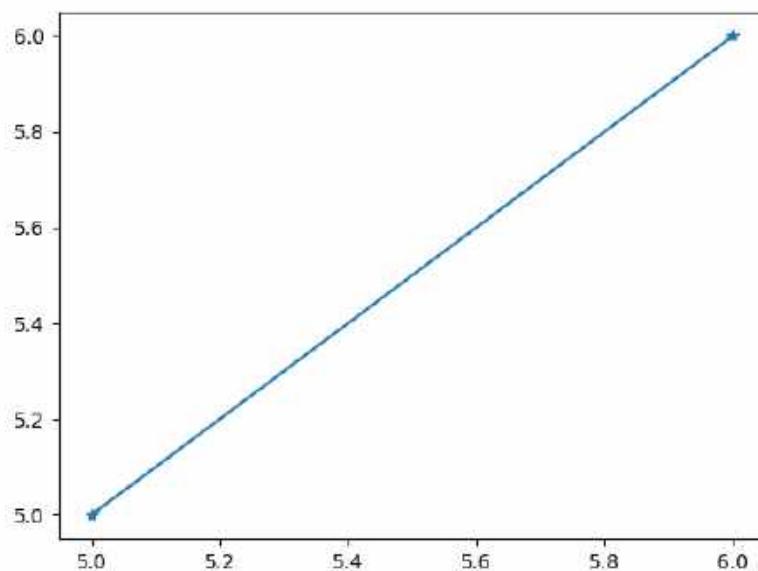
```
import matplotlib.pyplot as plt

x = [5, 6]
y = [5, 6]

plt.plot(x, y, marker='*')

plt.show()
```

**Output**



## Data Visualization Session at GRIET

**Program Name** Create a simple line chart  
demo3.py

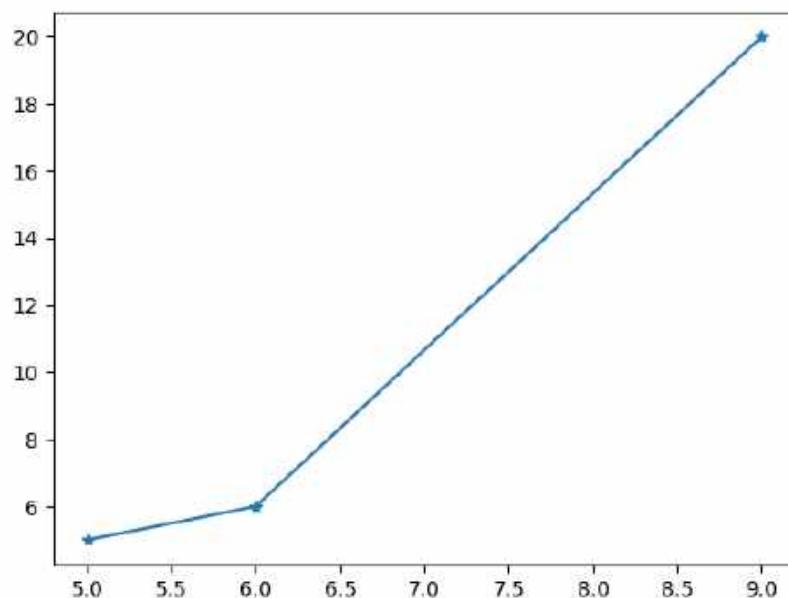
```
import matplotlib.pyplot as plt

x = [5, 6, 9]
y = [5, 6, 20]

plt.plot(x, y, marker='*')

plt.show()
```

**Output**



## Data Visualization Session at GRIET

**Program Name** Create a simple line chart and title  
demo4.py

```
import matplotlib.pyplot as plt

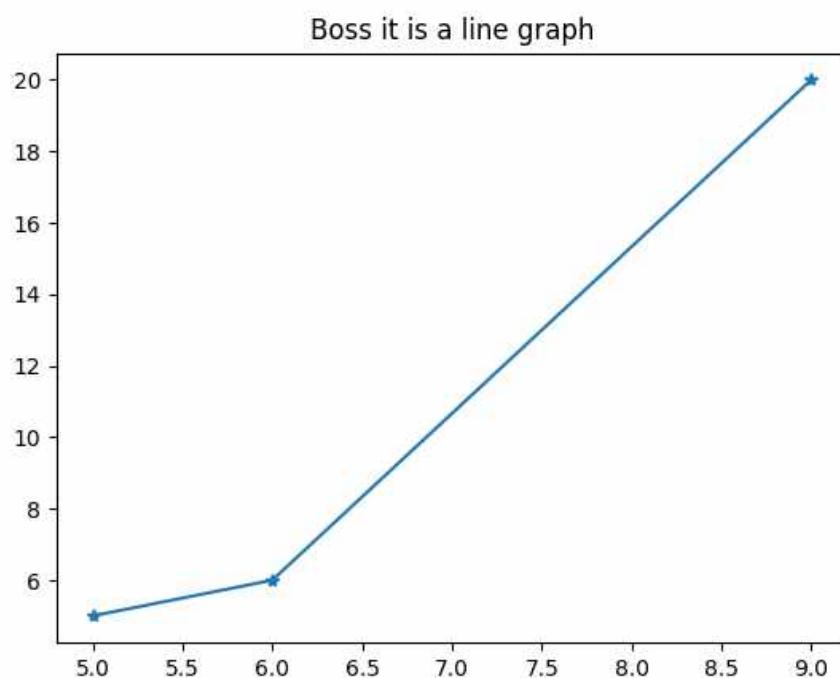
x = [5, 6, 9]
y = [5, 6, 20]

plt.title("Boss it is a line graph")

plt.plot(x, y, marker='*')

plt.show()
```

**Output**



## Data Visualization Session at GRIET

### 12.1. Labelling the axes

- ✓ We can label **x axis** and **y axis** by using xlabel and ylabel

**Program Name** Create a simple line chart and giving title and labelling  
demo5.py

```
import matplotlib.pyplot as plt

x = [5, 6, 9]
y = [5, 6, 20]

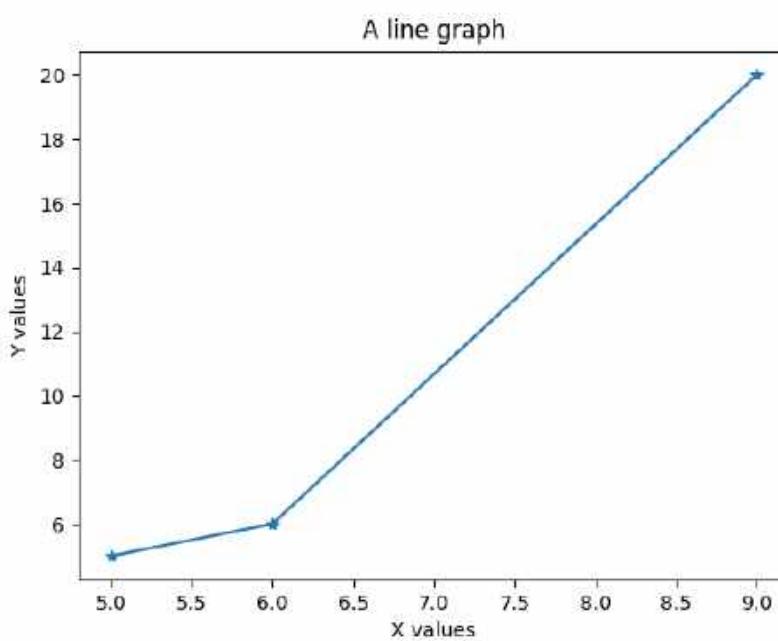
plt.title("A line graph")

plt.xlabel("X values")
plt.ylabel("Y values")

plt.plot(x, y, marker = '*')

plt.show()
```

#### Output



## Data Visualization Session at GRIET

**Program Name** Create two lines in single chart  
demo6.py

```
import matplotlib.pyplot as plt

x = [5, 6, 9]
y = [5, 6, 20]
p = [10, 20, 25]

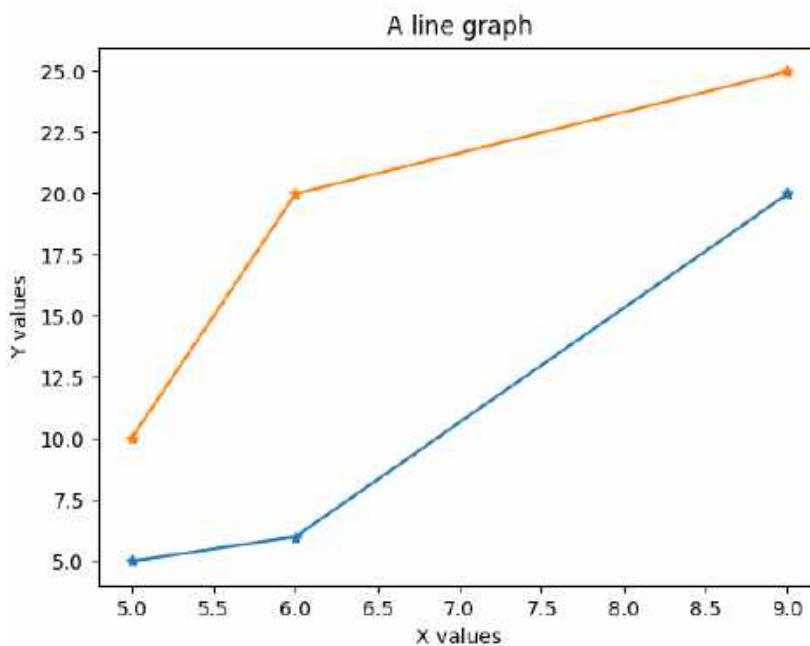
plt.title("A line graph")

plt.xlabel("X values")
plt.ylabel("Y values")

plt.plot(x, y, marker = '*')
plt.plot(x, p, marker = '*')

plt.show()
```

**Output**



## Data Visualization Session at GRIET

### 13. Bar Chart

- ✓ The bar graph is the graphical representation of categorical data.

**Program Name** Creating bar chart  
demo7.py

```
import matplotlib.pyplot as plt

months = ["Jan", "Feb", "Mar", "Apr", "May", "June", "July", "Aug",
          "Sep", "Oct", "Nov", "Dec"]
sales = [23, 45, 56, 78, 213, 45, 78, 89, 99, 100, 101, 130]

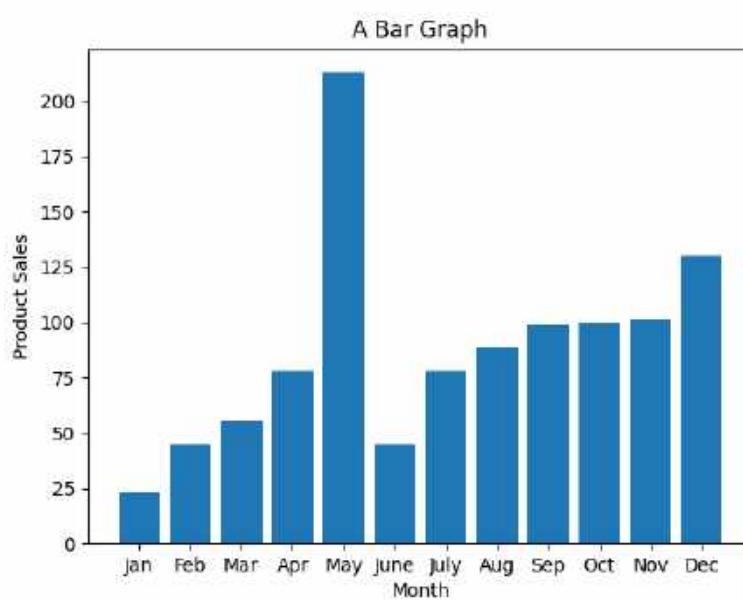
plt.title('A Bar Graph')

plt.xlabel('Month')
plt.ylabel('Product Sales')

plt.bar(months, sales)

plt.show()
```

#### Output



## Data Visualization Session at GRIET

**Program Name** Creating horizontal bar chart  
demo8.py

```
import matplotlib.pyplot as plt

months = ["Jan", "Feb", "Mar", "Apr", "May", "June", "July", "Aug",
          "Sep", "Oct", "Nov", "Dec"]
sales = [23, 45, 56, 78, 213, 45, 78, 89, 99, 100, 101, 130]

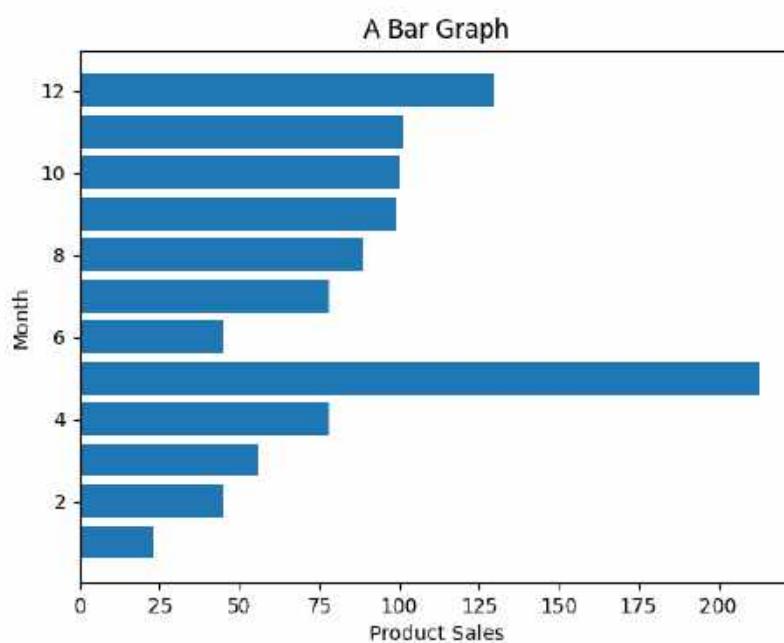
plt.title('A Bar Graph')

plt.xlabel('Product Sales')
plt.ylabel('Month')

plt.barh(months, sales)

plt.show()
```

### Output



## Data Visualization Session at GRIET

**Program Name** Creating horizontal bar chart  
**File name** demo9.py  
sales11.csv

```
import matplotlib.pyplot as plt
import pandas as pd

df = pd.read_csv("sales11.csv")

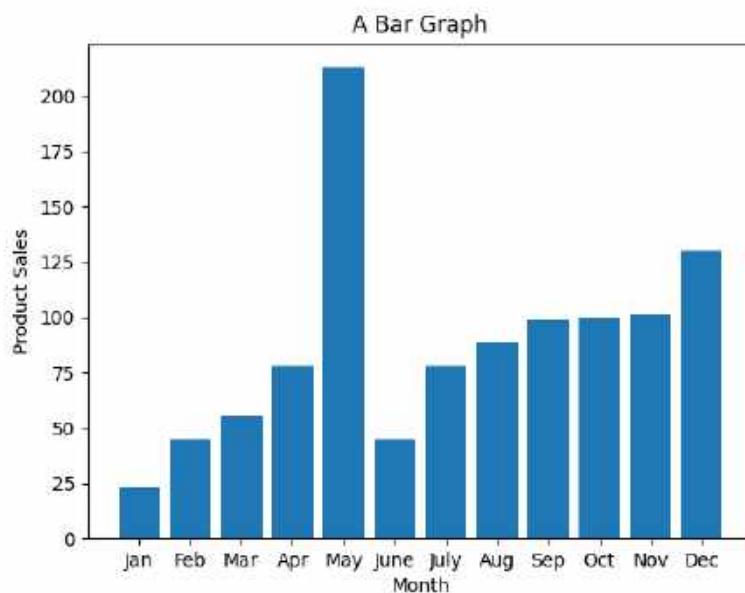
plt.title('A Bar Graph')

plt.xlabel('Month')
plt.ylabel('Product Sales')

plt.bar(df.month, df.sales)

plt.show()
```

### Output



## Data Visualization Session at GRIET

**Program Name** Creating bar chart  
demo10.py

```
import matplotlib.pyplot as plt

months = ["Jan", "Feb", "Mar", "Apr", "May", "June", "July", "Aug",
          "Sep", "Oct", "Nov", "Dec"]
sales = [23, 45, 56, 78, 213, 45, 78, 89, 99, 100, 101, 130]

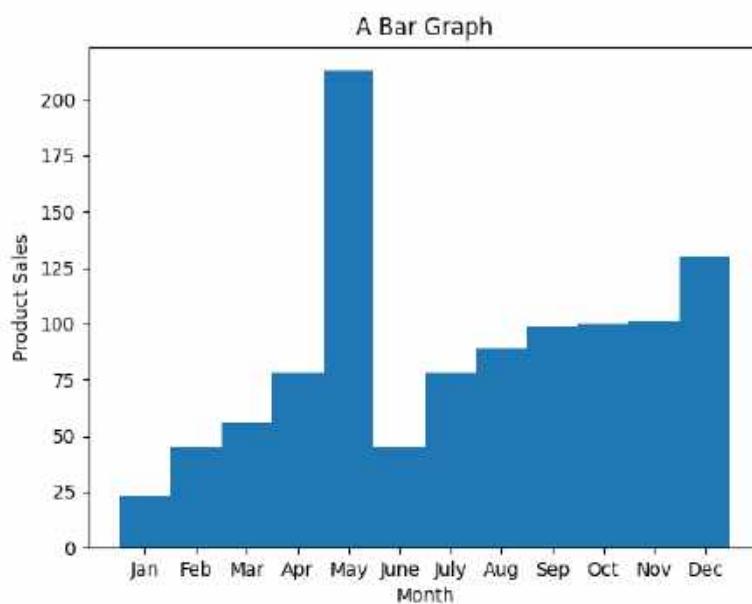
plt.title('A Bar Graph')

plt.xlabel('Month')
plt.ylabel('Product Sales')

plt.bar(months, sales, width = 1.0)

plt.show()
```

### Output



## Data Visualization Session at GRIET

### 14. Histogram

- ✓ A histogram is the graphical representation of quantitative data.
- ✓ This displays the frequency/count of numerical data in bars.

**Program**

Creating histogram

**Name**

demo11.py

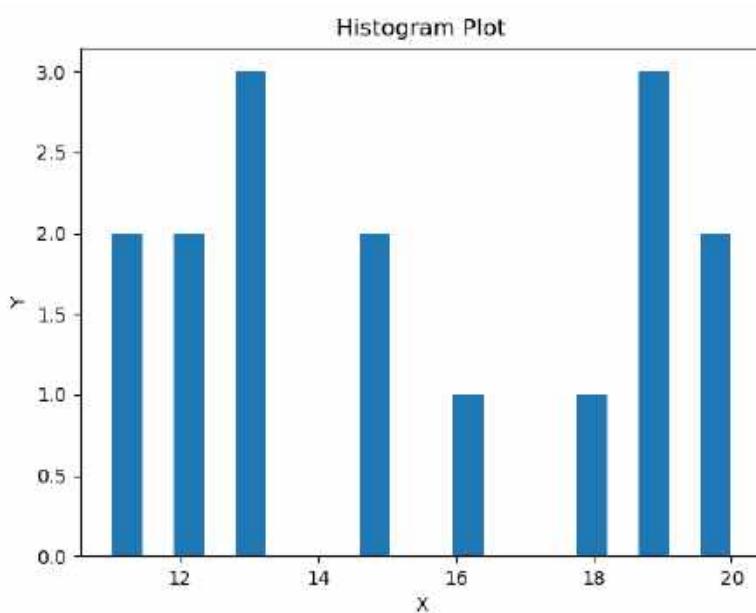
```
import matplotlib.pyplot as plt

data = [12, 15, 13, 20, 19, 20, 11, 19, 11, 12, 19, 13, 15, 16, 18, 13]

plt.xlabel("X")
plt.ylabel("Y")
plt.title("Histogram Plot")

plt.hist(data, bins = 20)
plt.show()
```

**Output**



## Data Visualization Session at GRIET

### 15. Pie Chart

- ✓ This is a circular plot that has been divided into slices displaying numerical proportions.
- ✓ Every slice in the pie chart shows the proportion of the element to the whole.
- ✓ A large category means that it will occupy a larger portion of the pie chart.

**Program Name** Creating pie chart  
demo12.py

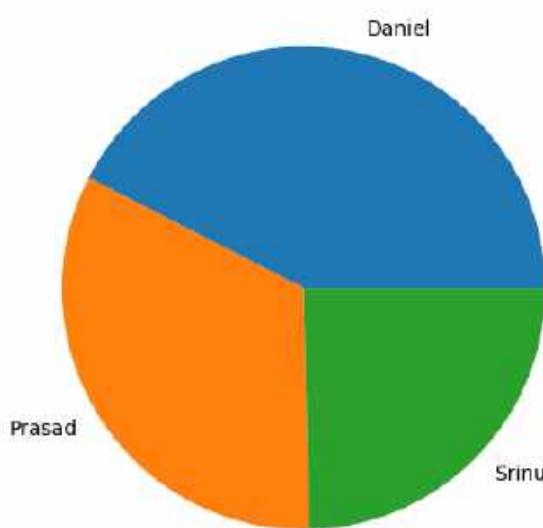
```
import matplotlib.pyplot as plt

students = ['Daniel', 'Prasad', 'Srinu']
points = [62, 48, 36]

plt.pie(points, labels = students)

plt.axis('equal')
plt.show()
```

#### Output



## Data Visualization Session at GRIET

Program

Creating pie chart

Name demo13.py

```
import matplotlib.pyplot as plt

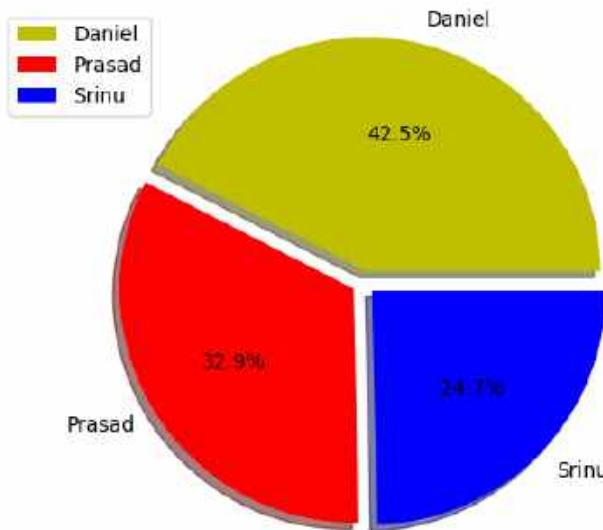
students = ['Daniel', 'Prasad', 'Srinu']
points = [62, 48, 36]

c = ['y', 'r', 'b']

plt.pie(points, labels = students, colors = c , shadow = True,
explode = (0.05, 0.05, 0.05), autopct = '%1.1f%%')

plt.axis('equal')
plt.legend()
plt.show()
```

Output



## Data Visualization Session at GRIET

---

### 15.1. Attributes

- ✓ The first parameter to the function is the list of numbers for every category.
  - labels attribute:
    - A list of categories separated by commas is then passed as the argument to labels attribute.
  - colors attribute:
    - To provide the color for every category.
  - To create shadows around the various categories in pie chart.
  - To split each slice of the pie chart into its own.

## Data Visualization Session at GRIET

### 16. Scatter Plot

- ✓ In scatter plot each value in the data set is represented by a dot.
- ✓ By using this plot we can understand the relationship between two variables.

**Program Name** Creating Scatter plot  
demo14.py

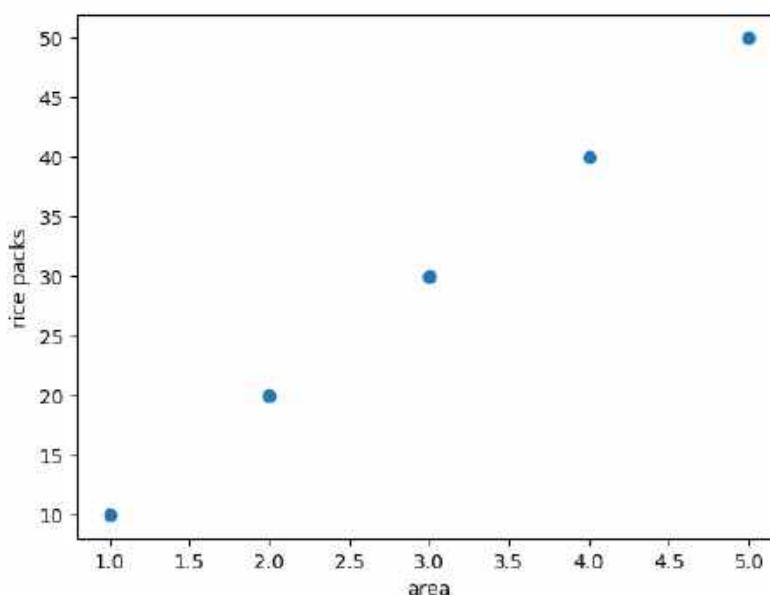
```
import matplotlib.pyplot as plt

area = [1, 2, 3, 4, 5]
rice_packs = [10, 20, 30, 40, 50]

plt.xlabel('area')
plt.ylabel('rice packs')

plt.scatter(area, rice_packs)
plt.show()
```

#### Output



## Data Visualization Session at GRIET

**Program Name** Creating Scatter plot  
demo15.py

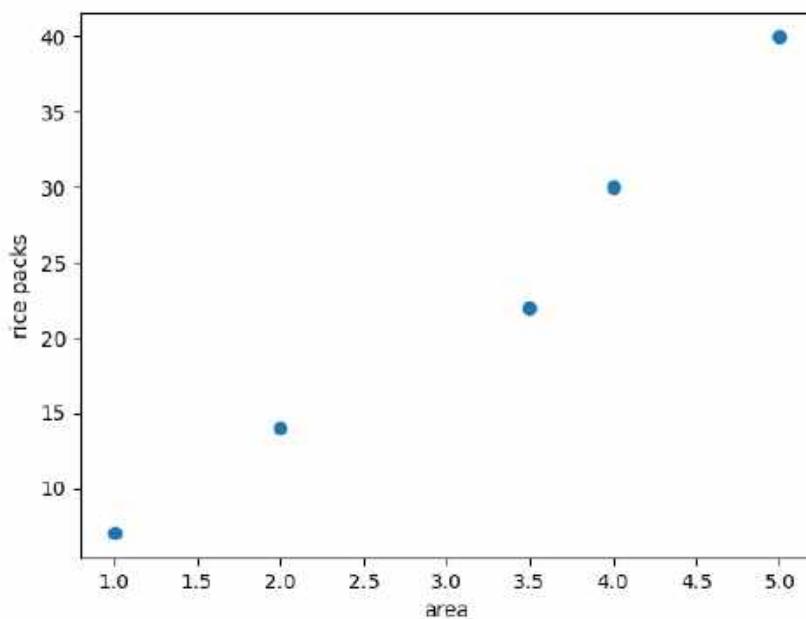
```
import matplotlib.pyplot as plt

area = [1, 2, 3.5, 4, 5]
rice_packs = [7, 14, 22, 30, 40]

plt.xlabel('area')
plt.ylabel('rice packs')

plt.scatter(area, rice_packs)
plt.show()
```

### Output



## Data Visualization Session at GRIET

### 17. Box Plots

- ✓ Box plots help us measure how well data in a dataset is distributed.
- ✓ The graph shows the maximum, minimum, median, first quartile and third quartiles of the dataset.

#### 17.1. Use Box plots

- ✓ Use a boxplot when you need to get the overall statistical information about the data distribution.
- ✓ It is a good tool for detecting outliers in a dataset.

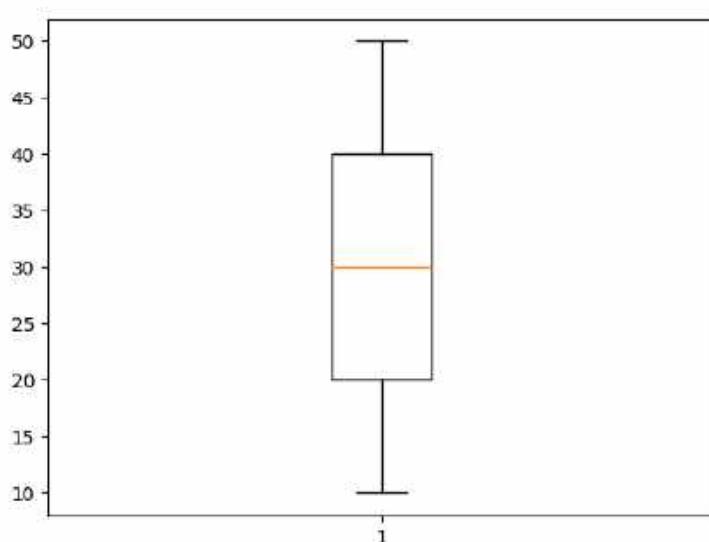
**Program Name** Creating box plot  
demo16.py

```
import matplotlib.pyplot as plt

data = [10, 20, 30, 40, 50]

plt.boxplot(data)
plt.show()
```

#### Output



## Data Visualization Session at GRIET

### 17.2. Box plot explanation

- ✓ The line dividing the box into two shows the median of the data.
- ✓ The end of the box represents the upper quartile (75%) while the start of the box represents the lower quartile (25%).
- ✓ The part between the upper quartile and the lower quartile is known as the Inter Quartile Range (IQR) and helps in approximating 50% of the middle data.

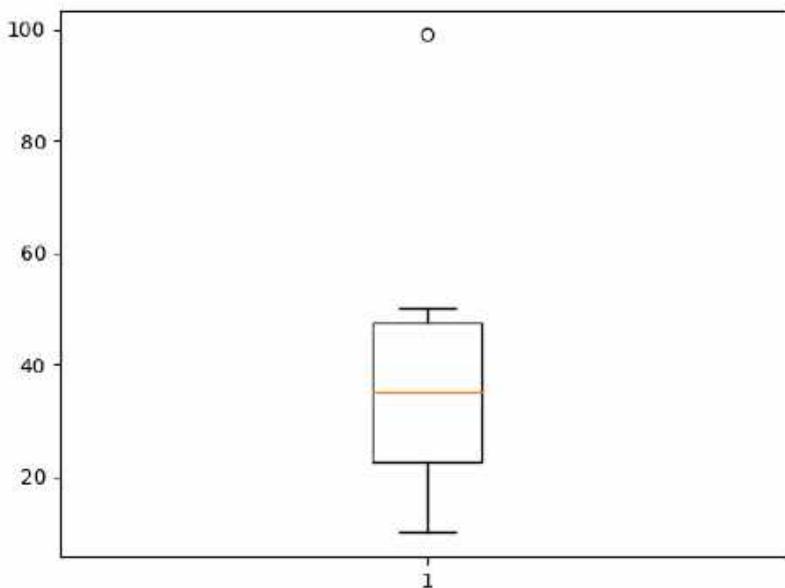
**Program Name** Creating box plot with outlier  
demo17.py

```
import matplotlib.pyplot as plt

data = [10, 20, 30, 40, 50, 90]

plt.boxplot(data)
plt.show()
```

#### Output



## Data Visualization Session at GRIET

---

### 18. Heatmap

- ✓ A heatmap is a method of data visualization that plots data by replacing numbers with colours.
- ✓ If it is representing with color then it is very easy to understand patterns between different values in the dataset.
- ✓ It is used to visualize data in a two-dimensional format as a coloured map so that different colour variations represent different patterns between features.

#### 18.1. How to understand?

- ✓ A heatmap visualizes the relationship between features as a colour palette.
- ✓ While analysing a heatmap, always remember that **dark shades** represent a **high degree** of linear relationship between features and **light shades** represent a **low degree** of linear relationship between features.

## Data Visualization Session at GRIET

**Program Name** Creating box plot  
demo18.py

```
import matplotlib.pyplot as plt
import pandas as pd

d = {
    "Apple": [10, 20, 30, 40],
    "Orange": [7, 14, 21, 28],
    "Banana": [55, 15, 8, 12],
    "Pear": [15, 14, 1, 8]
}

i = ['Basket1', 'Basket2', 'Basket3', 'Basket4']

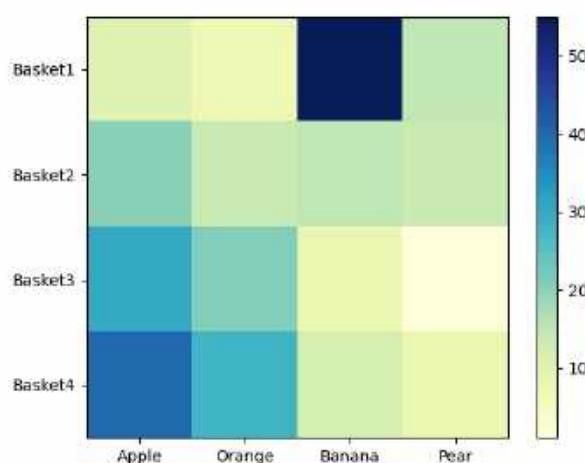
df = pd.DataFrame(d, index = i)

plt.imshow(df, cmap = "YIGnBu")
plt.colorbar()

plt.xticks(range(len(df)), df.columns)
plt.yticks(range(len(df)), df.index)

plt.show()
```

### Output



## 2. DATA VISUALIZATION – PART - 2

### Contents

<b>1. Titanic Introduction.....</b>	<b>2</b>
<b>2. Seaborn library .....</b>	<b>2</b>
2.1. Environment.....	2
<b>3. Titanic data set understanding .....</b>	<b>4</b>
<b>4. Data Analysis .....</b>	<b>8</b>
<b>5. Bar Plot .....</b>	<b>9</b>
<b>6. Get a count of the number of survivors .....</b>	<b>10</b>
<b>7. Count Plot.....</b>	<b>11</b>
<b>9. Plot the survival rate of each class .....</b>	<b>12</b>
<b>10. Let's understand the survival rate by gender and class.....</b>	<b>13</b>
<b>11. Let's understand the survival rate by gender, age and class.....</b>	<b>14</b>
<b>12. Dist Plot .....</b>	<b>15</b>
<b>13. Box Plot.....</b>	<b>16</b>
<b>14. Violin Plot .....</b>	<b>19</b>
<b>15. Word Cloud.....</b>	<b>21</b>
<b>16. Sunburst plot .....</b>	<b>23</b>

## 2. DATA VISUALIZATION – PART - 2

### 1. Titanic Introduction

- ✓ The Titanic was known as the unsinkable ship and was the largest, most luxurious passenger ship.
- ✓ Sadly, the British ocean liner sank on April 15, 1912, killing over many people while just few people got survived.
- ✓ Let's do analyse titanic dataset

### 2. Seaborn library

- ✓ Seaborn is advanced data visualization library.
- ✓ By using this we can visualize the data.

#### 2.1. Environment

- ✓ We can install this library by using pip command.

##### Seaborn installation

```
pip install seaborn
```

## Data Science – Data Visualization

**Program Name** Loading titanic dataset  
demo1.py

```
import seaborn as sns

df = sns.load_dataset('titanic')
print(df.head())
```

**Output**

```
   survived  pclass    sex   age  sibsp  parch ... who  adult_male   deck  embark_town  alive  alone
0         0       3  male  22.0      1      0 ... man     True   NaN  Southampton    no  False
1         1       1 female  38.0      1      0 ... woman   False     C  Cherbourg   yes  False
2         1       3 female  26.0      0      0 ... woman   False   NaN  Southampton   yes  True
3         1       1 female  35.0      1      0 ... woman   False     C  Southampton   yes  False
4         0       3  male  35.0      0      0 ... man     True   NaN  Southampton    no  True

[5 rows x 15 columns]
```

### 3. Titanic data set understanding

- ✓ Let's understand the titanic dataset.
- ✓ Data Set Column Descriptions
  - pclass: Passenger Class (1 = 1st; 2 = 2nd; 3 = 3rd)
  - survived: Survival (0 = No; 1 = Yes)
  - name: Name
  - sex: type of gender
  - age: Age
  - sibsp: Number of siblings/spouses aboard
  - parch: Number of parents/children aboard
  - fare: Passenger fare (British pound)
  - embarked: Port of embarkation (C = Cherbourg; Q = Queenstown; S = Southampton)
  - adult\_male: A male 18 or older (0 = No, 1=Yes)
  - deck: Deck of the ship
  - who: man (18+), woman (18+), child (<18)
  - alive: Yes, no
  - embarked\_town: Port of embarkation (Cherbourg, Queenstown, Southampton)
  - class: Passenger class (1st; 2nd; 3rd)
  - alone: 1 = alone, 0 = not alone ( you have at least 1 sibling, spouse, parent or child on board)

## Data Science – Data Visualization

**Program Name** Number of rows and columns  
demo2.py

```
import seaborn as sns

df = sns.load_dataset('titanic')
print(df.shape)
```

**Output**

(891, 15)

**Program Name** Display the columns  
demo3.py

```
import seaborn as sns

df = sns.load_dataset('titanic')
print(df.columns)
```

**Output**

Index(['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'fare',  
'embarked', 'class', 'who', 'adult\_male', 'deck', 'embark\_town',  
'alive', 'alone'], dtype='object')

## Data Science – Data Visualization

**Program Name** DataFrame information  
demo4.py

```
import seaborn as sns

df = sns.load_dataset('titanic')
df.info()
```

**Output**

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   survived    891 non-null    int64  
 1   pclass      891 non-null    int64  
 2   sex         891 non-null    object  
 3   age         714 non-null    float64 
 4   sibsp       891 non-null    int64  
 5   parch       891 non-null    int64  
 6   fare         891 non-null    float64 
 7   embarked    889 non-null    object  
 8   class        891 non-null    category
 9   who          891 non-null    object  
 10  adult_male  891 non-null    bool   
 11  deck         203 non-null    category
 12  embark_town 889 non-null    object  
 13  alive        891 non-null    object  
 14  alone        891 non-null    bool  
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.7+ KB
```

## Data Science – Data Visualization

**Program Name** unique values for sex(gender) column  
demo5.py

```
import seaborn as sns
import pandas as pd

df = sns.load_dataset('titanic')
result = df['sex'].unique()
print(result)
```

**Output**

```
['male' 'female']
```

## 4. Data Analysis

- ✓ From the data **max price/fare** a passenger paid for a ticket in this data set was 512.3292 British pounds, and the **minimum price/fare** was 0 British pounds.
- ✓ There is missing data for age column.
- ✓ The **mean** age is 29.699 and the oldest passenger in this data set was 80 years old, while the youngest was only .42 years old (about 5 months).

**Program Name**      describe() method  
demo6.py

```
import seaborn as sns

df = sns.load_dataset('titanic')
print(df.describe())
```

### Output

	survived	pclass	age	sibsp	parch	fare
count	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

## 5. Bar Plot

- ✓ A bar plot shows the **mean** value of every value in a categorical column.

**Program Name** Creating bar plot  
demo7.py

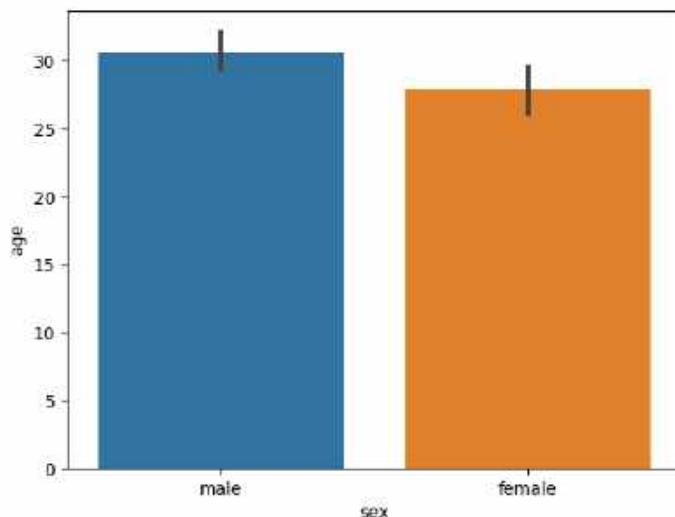
```
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')

df = sns.load_dataset('titanic')
sns.barplot(x = 'sex', y = 'age', data = df)

plt.show()
```

**Output**



- ✓ The plot clearly shows that the **average age** for all **male passengers** is above **30** while the average age of the **female passengers** is between 25 and 30.

## 6. Get a count of the number of survivors

- ✓ 0 represents not survived
- ✓ 1 means survived.

**Program Name** Get a count of the number of survivors  
demo8.py

```
import seaborn as sns

df = sns.load_dataset('titanic')
print(df['survived'].value_counts())
```

**Output**

```
0    549
1    342
Name: survived, dtype: int64
```

## 7. Count Plot

- ✓ This type of plot is similar to the bar plot, it displays the count of categories in a specific column.
- ✓ By using we can calculate the total number or count of survived and not survived.

**Program Name** Get a count of the number of survivors  
demo9.py

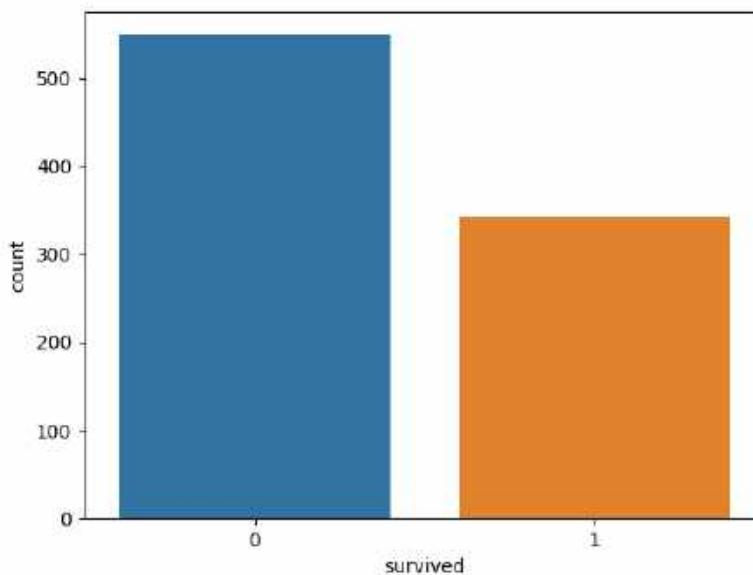
```
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')

df = sns.load_dataset('titanic')
sns.countplot(x = "survived", data = df)

plt.show()
```

**Output**



## 9. Plot the survival rate of each class

- ✓ A little over 60% of the passengers in first class survived. Less than 30% of passengers in third class survived.
- ✓ That means less than half of the passengers in third class survived, compared to the passengers in first class.

**Program Name** Plot the survival rate of each class  
demo10.py

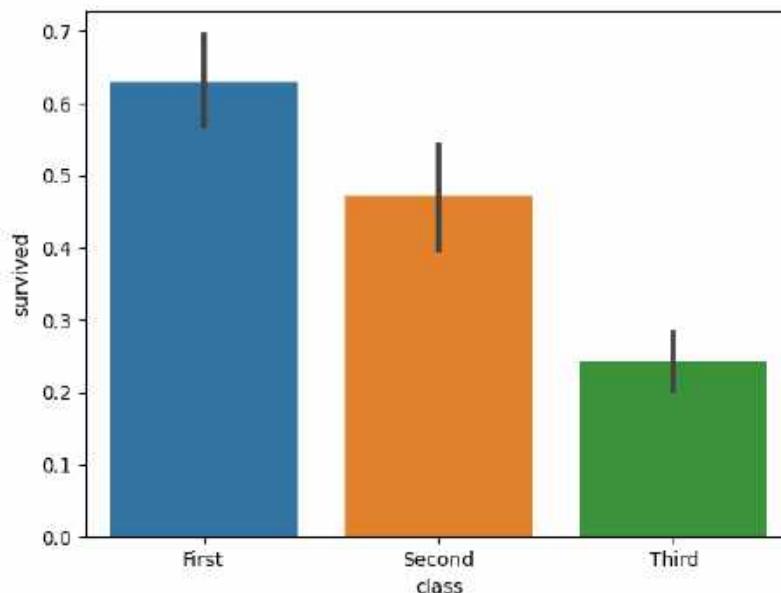
```
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')

df = sns.load_dataset('titanic')
sns.barplot(x = 'class', y = 'survived', data = df)

plt.show()
```

**Output**



**10. Let's understand the survival rate by gender and class.**

- ✓ From the pivot table below, we see that females in first class had a survival rate of about 96.8%, meaning the majority of them survived.
- ✓ Males in third class had the lowest survival rate at about 13.54%, meaning the majority of them did not survive.

**Program Name** Plot the survival rate of each class  
demo11.py

```
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')

df = sns.load_dataset('titanic')

result = df.pivot_table('survived', index = 'sex', columns = 'class')

print(result)
```

**Output**

class	First	Second	Third
sex			
female	0.968085	0.921053	0.500000
male	0.368852	0.157407	0.135447

**11. Let's understand the survival rate by gender, age and class.**

**Program Name** Plot the survival rate by gender, age and class  
demo12.py

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

df = sns.load_dataset('titanic')

diff_ages = pd.cut(df['age'], [0, 18, 80])

result = df.pivot_table('survived', ['sex', diff_ages], 'class')

print(result)
```

**Output**

class		First	Second	Third
sex	age			
female	(0, 18]	0.909091	1.000000	0.511628
	(18, 80]	0.972973	0.900000	0.423729
male	(0, 18]	0.800000	0.600000	0.215686
	(18, 80]	0.375000	0.071429	0.133663

## 12. Dist Plot

- ✓ To create distribution plot we need to call distplot(p) function.
- ✓ This will create histogram distribution of a dataset for a column.
- ✓ We can plot the price of the ticket for every passenger

**Program Name** Finding Most of the tickets, using dist plot  
demo13.py

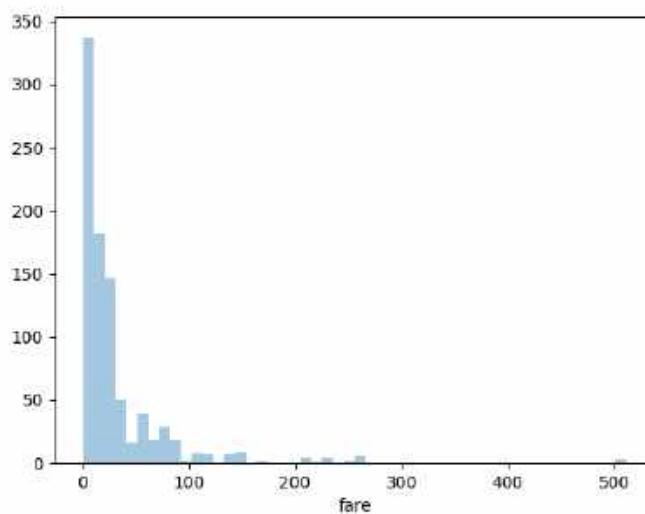
```
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')

df = sns.load_dataset('titanic')
sns.distplot(df['fare'], kde = False)

plt.show()
```

### Output



- ✓ The above plot shows that most of the tickets have been sold between 0 and 50 dollars.

### 13. Box Plot

- ✓ The box plot is used to display the distribution of the categorical data in the form of quartiles like Q1, Q2, Q3 and Q3.
- ✓ The center of the box shows the median value.
- ✓ Now let's plot a box plot that displays the distribution for the age with respect to each gender.

**Program Name** Creating a boxplot with sex column(gender) survived  
demo14.py

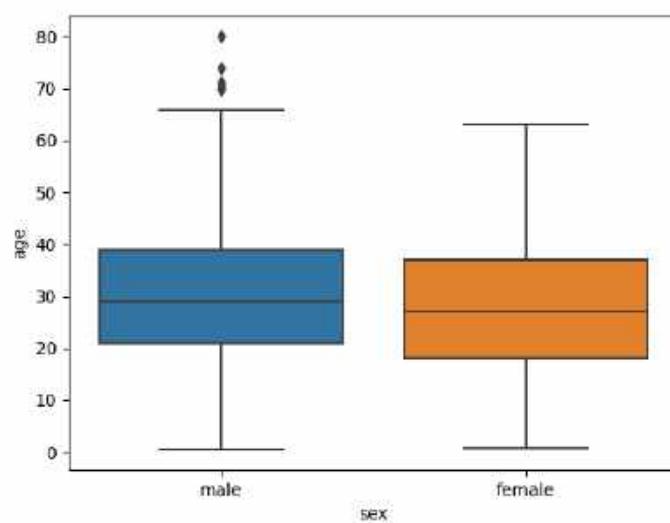
```
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')

df = sns.load_dataset('titanic')
sns.boxplot(x = 'sex', y = 'age', data = df)

plt.show()
```

#### Output



- ✓ The first quartile-Q1 starts at around 3 and ends at 22 which mean that 25% of the passengers are aged between 5 and 22.
- ✓ The second quartile-Q2 starts at around 23 and ends at around 28 which mean that 25% of the passengers are aged between 23 and 28.
- ✓ Similarly, the third quartile-Q3 starts and ends between 29 and 38, hence 25% passengers are aged within this range and finally the fourth or last quartile—Q4 starts at 39 and ends around 76.
- ✓ The part between the upper quartile and the lower quartile is known as the **Inter Quartile Range (IQR)** and helps in approximating 50% of the middle data.

## Data Science – Data Visualization

---

**Program Name** Creating a boxplot with survived  
demo15.py

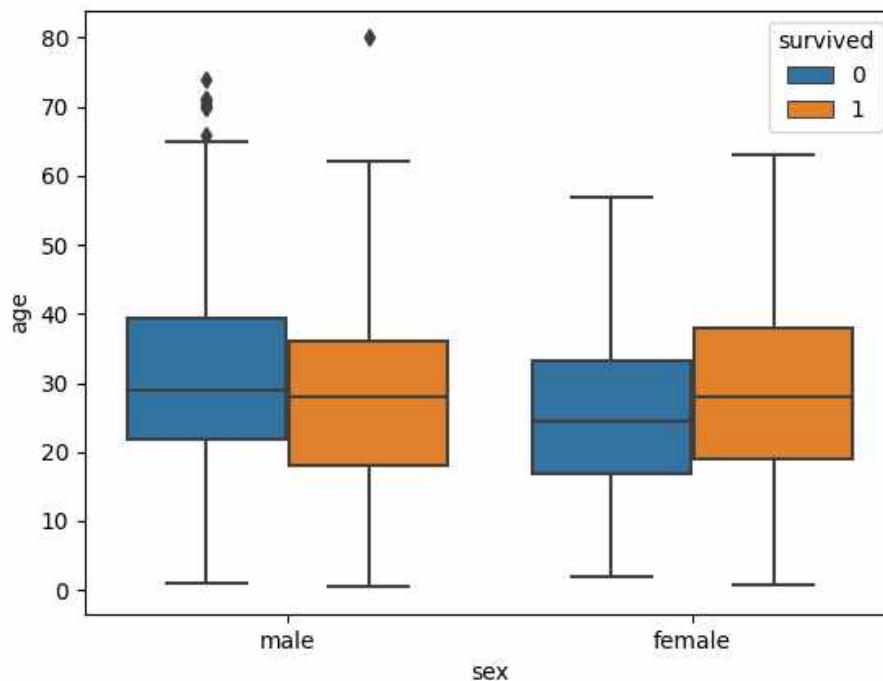
```
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')

df = sns.load_dataset('titanic')
sns.boxplot(x = 'sex', y = 'age', data = df, hue = "survived")

plt.show()
```

**Output**



- ✓ Other than the information about the age of the passengers, the above plot also shows the distribution of passengers who survived.
- ✓ The plot shows that most young males survived compared to females.

#### 14. Violin Plot

- ✓ This type of plot is the same as the box plot, but with a violin plot, we can display all components corresponding to a data point.

Program Name      Creating violin plot  
demo16.py

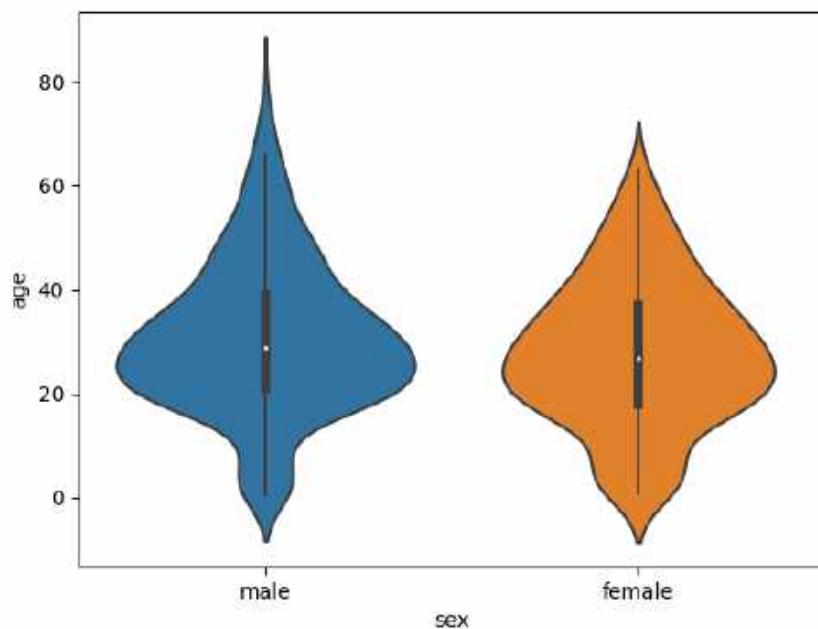
```
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')

df = sns.load_dataset('titanic')
sns.violinplot(x = 'sex', y = 'age', data = df)

plt.show()
```

Output



## Data Science – Data Visualization

**Program Name** Creating violin plot with survived  
demo17.py

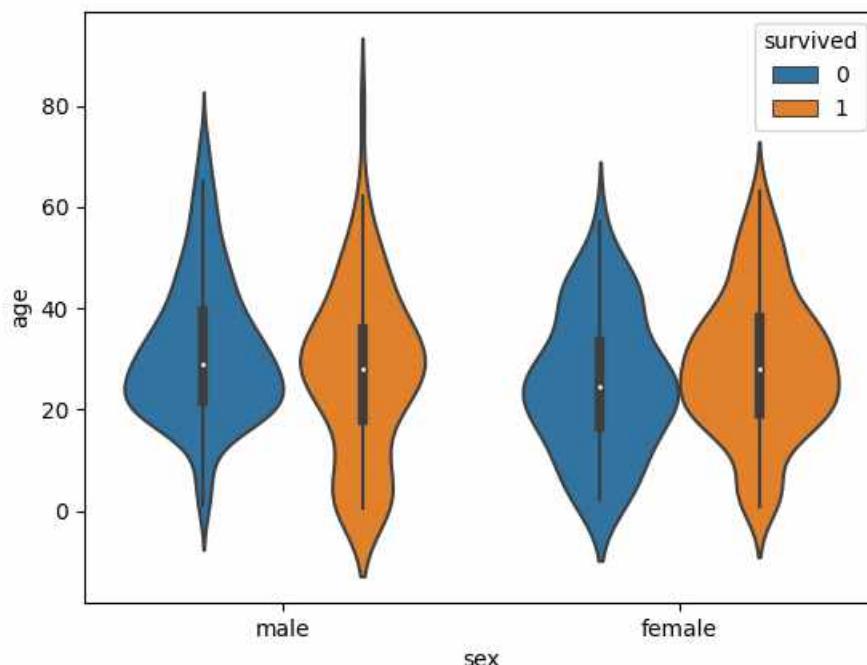
```
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')

df = sns.load_dataset('titanic')
sns.violinplot(x = 'sex', y = 'age', data = df, hue = 'survived')

plt.show()
```

**Output**



## 15. Word Cloud

- ✓ A word cloud is a data visualization technique.
- ✓ This technique displays most used words in large font and the least used words in small font.
- ✓ It helps to get an idea about your text data,

### We need to install

```
pip install wordcloud
```

Program Name Wordcloud example  
demo18.py

```
import matplotlib.pyplot as plt
from wordcloud import WordCloud

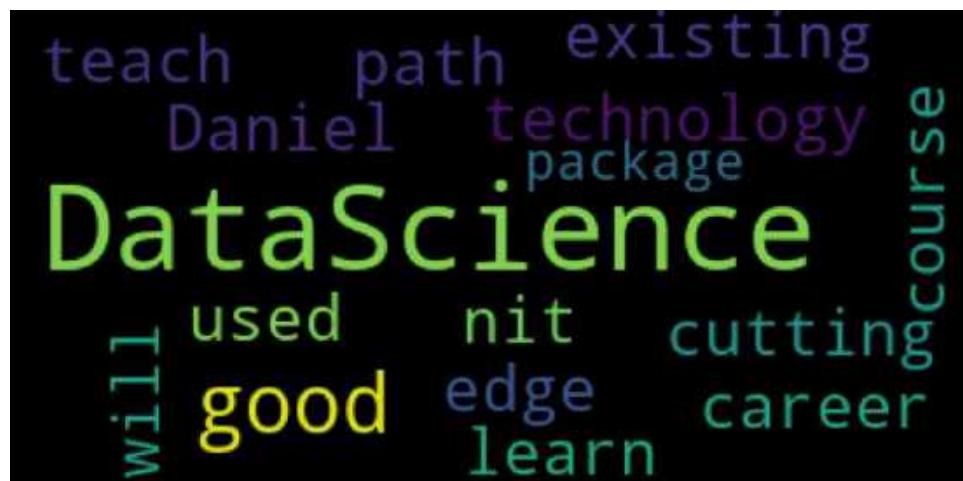
text = "DataScience having good career path, DataScience is
cutting edge technology, DataScience course is existing in nit,
Daniel used to teach DataScience, if we learn DataScience then we
will get good package"

wc = WordCloud()
wc.generate(text)

plt.figure(figsize = (12, 12))
plt.imshow(wc)

plt.axis('off')
plt.show()
```

Output



## 16. Sunburst plot

- ✓ A sunburst plot is a very popular data visualization technique used to visualize hierarchical data.
- ✓ In every level of the hierarchy is represented by a ring or circle.
- ✓ Whereas the innermost circle or ring is the highest level of the hierarchy.

### We need to install

```
pip install plotly
```

## Data Science – Data Visualization

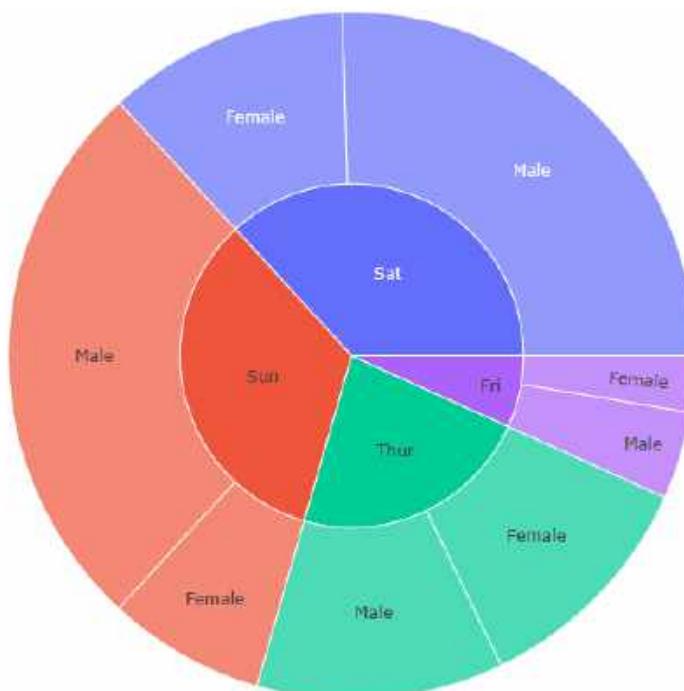
**Program Name** Sunburst plot example  
demo19.py

```
import plotly.express as px

data = px.data.tips()
figure = px.sunburst(data, path = ["day", "sex"], values =
"total_bill")

figure.show()
```

**Output**



### 1. Numpy – Introduction

#### Table of Contents

1. Numpy .....	2
2. What is an array? .....	2
3. Why to use numpy array? .....	2
4. Open source.....	2
5. numpy – Installation .....	2
6. pip command in python .....	3
7. Check installed python library. ....	3
8. import numpy package .....	3
9. ModuleNotFoundError.....	4
10. Famous Alias name to numpy.....	4

### 1. Numpy – Introduction

#### 1. Numpy

- ✓ NumPy is the fundamental package for scientific computing in Python.
- ✓ It is a Python library that provides multidimensional array object.
- ✓ The full form of **numpy** is 'Numerical Python'.
- ✓ Numpy was created by *Travis Oliphant*

#### 2. What is an array?

- ✓ Array is an object which stores a group of values.
- ✓ Also called as, ordered collection of values.
- ✓ Array can store same type of values.

#### 3. Why to use numpy array?

- ✓ Python lists are a bit slow in process.
- ✓ Numpy arrays are faster than python list.
- ✓ The array object in numpy is called as ndarray.

#### 4. Open source

- ✓ Numpy is an open source means it's free.

#### 5. numpy – Installation

- ✓ By default numpy will not be available with python installation.
- ✓ Explicitly we need to install numpy package.
- ✓ Run below command in command prompt.

```
pip install numpy
```

### 6. pip command in python

- ✓ pip stands for **python installer package**
- ✓ Pip is a package management system.
- ✓ It is used to install and manage software packages.
  - pip install package\_name
  - pip install numpy

### 7. Check installed python library.

- ✓ We can check installed python library by using below command.

```
pip show numpy
```

### 8. import numpy package

- ✓ We can import numpy package by using import.

<b>Program Name</b>	importing numpy package demo1.py
	<b>import</b> numpy print('numpy imported successfully')

#### Output

```
numpy imported successfully
```

## Data Science – Numpy Introduction

---

### 9. ModuleNotFoundError

- ✓ If numpy is not installed then we will get below error.

**Program Name** checking numpy installed or not  
demo2.py

```
import numpy
print('numpy imported successfully')
```

**Output**

```
Traceback (most recent call last):
File "demo2.py", line 1, in <module>
    import numpy
ModuleNotFoundError: No module named 'numpy'
```

### 10. Famous Alias name to numpy

- ✓ We can give alias name to numpy.
- ✓ Note this name can be any name but the famous alias name is np

**Program Name** alias name to numpy  
demo3.py

```
import numpy as np
print('numpy imported successfully')
print('Alias name given to numpy as np')
```

**Output**

```
numpy imported successfully
Alias name given to numpy as np
```

## 2. Numpy – Fundamentals

### Contents

1. Creating numpy array.....	2
2. numpy.ndim .....	2
3. Indexing and Slicing.....	6
4. Creating a array with all zeros .....	10
5. Creating a array with all ones .....	11

## 2. Numpy – Fundamentals

### 1. Creating numpy array

- ✓ We can create numpy array by using array(p) function.
- ✓ Internally it creates object to ndarray.
- ✓ We can pass list, tuple etc as a parameter to the array(p) function.
- ✓ Having same type of values is recommended.

### 2. numpy.ndim

- ✓ Ndim is predefined variable in numpy
- ✓ By using this we can check the array dimensions.

**Program Name** Creating numpy array with single value  
demo1.py

```
import numpy as np

age = 44
value = np.array(age)

print(value)
print(type(value))
print(value.ndim)
```

### Output

```
44
<class 'numpy.ndarray'>
0
```

## Data Science – Numpy Fundamentals

**Program Name** Creating numpy array with group of values  
demo2.py

```
import numpy as np

details = [10, 20, 30, 40, 50]
sales = np.array(details)

print(sales)
print(type(sales))
print(sales.ndim)
```

**Output**

```
[10 20 30 40 50]
<class 'numpy.ndarray'>
1
```

**Program Name** Creating numpy array with group of values  
demo3.py

```
import numpy as np

details = [[10, 20], [30, 40]]
sales = np.array(details)

print(sales)
print(type(sales))
print(sales.ndim)
```

**Output**

```
[[10 20]
 [30 40]]
<class 'numpy.ndarray'>
2
```

**Program Name** Creating numpy array with group of values  
demo4.py

```
import numpy as np

details = [[10, 20], [30, 40], [50, 60]]
sales = np.array(details)
print(sales)
print(type(sales))
print(sales.ndim)
```

**Output**

```
[[10 20]
 [30 40]
 [50 60]]
<class 'numpy.ndarray'>
2
```

### 3. Indexing and Slicing

- ✓ We can access numpy array values by using indexing and slicing
- ✓ Numpy array having indexing nature.
- ✓ Numpy array index start with 0.
  - First element stores in 0<sup>th</sup> index
  - Second element stores in 1<sup>st</sup> index etc
- ✓ By using slicing we can access piece of array from the main array.

**Program Name** Accessing numpy array by using indexing  
demo5.py

```
import numpy as np

details = [10, 20, 30, 40, 50]
sales = np.array(details)
print(sales)
print(sales[0])
print(sales[1])
print(sales[2])
```

**Output**

```
10
20
30
```

## Data Science – Numpy Fundamentals

**Program Name** Accessing numpy array by using indexing  
demo6.py

```
import numpy as np

details = [10, 20, 30, 40, 50]
sales = np.array(details)

print(sales)
print(sales[2:])
```

**Output**  
[30, 40, 50]

## Data Science – Numpy Fundamentals

**Program Name** Creating matrix and selecting elements  
demo7.py

```
import numpy as np

matrix = np.array([[10, 20, 30], [40, 50, 60], [70, 80, 90]])

print(matrix)

print(matrix[0,0])
print(matrix[0,1])
print(matrix[0,2])

print(matrix[1,0])
print(matrix[1,1])
print(matrix[1,2])

print(matrix[2,0])
print(matrix[2,1])
print(matrix[2,2])
```

**Output**

```
[[10 20 30]
 [40 50 60]
 [70 80 90]]
10
20
30
40
50
60
70
80
90
```

**IndexError**

- ✓ If we try to access value with out of bounds of index then we will get IndexError.

**Program Name** Accessing numpy array value  
demo8.py

```
import numpy as np

details = [10, 20, 30, 40, 50]
sales = np.array(details)

print(sales)
print(sales[22])
```

**Output**

IndexError: index 22 is out of bounds for axis 0 with size 5

#### 4. Creating a array with all zeros

- ✓ We can create array with all zeros by using numpy.zeros() function

**Program Name** Creating numpy array with group of values  
demo9.py

```
import numpy as np

sales = np.zeros(5)

print(sales)
print(type(sales))
```

**Output**

```
[0. 0. 0. 0. 0.]
<class 'numpy.ndarray'>
```

## 5. Creating a array with all ones

- ✓ We can create array with all ones by using numpy.ones() function

**Program Name** Creating numpy array with group of values  
demo10.py

```
import numpy as np

sales = np.ones(5)

print(sales)
print(type(sales))
```

**Output**

```
[1. 1. 1. 1. 1.]
<class 'numpy.ndarray'>
```

## Data Science – Numpy Attributes

---

### 3. NUMPY – ATTRIBUTES

#### Contents

1. Numpy Array Attributes.....	2
2. shape attribute .....	2
3. ndim attribute .....	3
4. arrayobject.T .....	5

## Data Science – Numpy Attributes

### 3. NUMPY – ATTRIBUTES

#### 1. Numpy Array Attributes

- ✓ Numpy array having predefined attributes to help us understand the essentials functionality.

#### 2. shape attribute

- ✓ shape is a predefined attribute in numpy array.
- ✓ We should access this shape attribute by using numpy array object.
- ✓ By using this we can check number of rows and columns in an array.
- ✓ Shape attribute returns the tuple as number of rows and columns.

**Program Name** Creating numpy array with group of values  
demo2.py

```
import numpy as np

details = [10, 20, 30], [40, 50, 60]
sales = np.array(details)
print(sales)
print(sales.shape)
```

#### Output

```
[[10 20 30]
 [40 50 60]]
(2, 3)
```

## Data Science – Numpy Attributes

### 3. ndim attribute

- ✓ ndim is a predefined attribute in numpy array.
- ✓ We should access this ndim attribute by using numpy array object
- ✓ By using this we can check the dimensions of an array

**Program Name** Creating numpy array, check with ndim attribute  
demo2.py

```
import numpy as np

details = [10, 20, 30, 40, 50]
sales = np.array(details)
print(sales)
print(sales.ndim)
```

**Output**

```
[10 20 30 40 50]
1
```

## Data Science – Numpy Attributes

**Program Name** Creating numpy array, check with ndim attribute  
demo3.py

```
import numpy as np

details = [[10, 20], [30, 40]]
sales = np.array(details)
print(sales)
print(sales.ndim)
```

**Output**

```
[[10 20]
 [30 40]]
2
```

**Program Name** Creating numpy array with group of values  
demo3.py

```
import numpy as np

details = [[10, 20], [30, 40], [50, 60]]
sales = np.array(details)
print(sales)
print(type(sales))
print(sales.ndim)
```

**Output**

```
[[10 20]
 [30 40]
 [50 60]]
<class 'numpy.ndarray'>
2
```

## Data Science – Numpy Attributes

### 4. arrayobject.T

- ✓ T is a predefined attribute in numpy array.
- ✓ We should access this T attribute by using numpy array object
- ✓ By using this we can transpose the array means it converts rows as columns and columns as rows.

Program Name      T attribute  
demo2.py

```
import numpy as np

details = [[10, 20, 30], [40, 50, 60]]
sales = np.array(details)
print(sales)
print()
print(sales.T)
```

Output

```
[[10 20 30]
 [40 50 60]]

[[10 40]
 [20 50]
 [30 60]]
```

## Data Science – Numpy Attributes

Program Name T attribute  
demo3.py

```
import numpy as np

details = [[10, 20], [30, 40]]
sales = np.array(details)
print(sales)
print()
print(sales.T)
```

Output

```
[[10 20]
 [30 40]]

[[10 30]
 [20 40]]
```

## Data Science – Numpy Important Methods

---

### 4. NUMPY – IMPORTANT METHODS

#### Contents

<b>1. Numpy Array Methods.....</b>	<b>2</b>
<b>2. min() method.....</b>	<b>2</b>
<b>3. max() method.....</b>	<b>3</b>
<b>4. sum() method .....</b>	<b>4</b>
<b>5. reshape() method.....</b>	<b>5</b>
<b>6. count_nonzero(p) function .....</b>	<b>8</b>
<b>7. sort() method .....</b>	<b>9</b>
<b>8. flatten() method.....</b>	<b>10</b>
<b>9. adding value to array of values.....</b>	<b>11</b>
<b>10. Diagonal of a Matrix.....</b>	<b>12</b>
<b>11. Trace of a Matrix .....</b>	<b>13</b>
<b>12. Adding and Subtracting Matrices .....</b>	<b>15</b>

## Data Science – Numpy Important Methods

### 4. NUMPY – IMPORTANT METHODS

#### 1. Numpy Array Methods

- ✓ Numpy array having predefined methods to perform different operations over array.

#### 2. min() method

- ✓ min() is a predefined method in numpy array.
- ✓ We should access this min() method by using numpy array object
- ✓ By using this we can check minimum value from the array.

Program      min() method  
Name          demo1.py

```
import numpy as np

details = [[10, 20, 30], [40, 50, 60]]
sales = np.array(details)
print(sales)
print(sales.min())
```

Output

10

## Data Science – Numpy Important Methods

### 3. max() method

- ✓ max() is a predefined method in numpy array.
- ✓ We should access this max() method by using numpy array object
- ✓ By using this we can check maximum value from the array.

Program      max() method

Name            demo2.py

```
import numpy as np

details = [[10, 20, 30], [40, 50, 60]]
sales = np.array(details)
print(sales)
print(sales.max())
```

Output

60

## Data Science – Numpy Important Methods

### 4. sum() method

- ✓ sum() is a predefined method in numpy array.
- ✓ We should access this method by using numpy array object
- ✓ By using this we can get sum of all values from array.

Program      sum() method  
Name          demo3.py

```
import numpy as np

details = [[10, 20, 30], [40, 50, 60]]
sales = np.array(details)
print(sales)
print()
print(sales.sum())
```

Output

```
[[10 20 30]
 [40 50 60]]

210
```

## Data Science – Numpy Important Methods

### 5. reshape() method

- ✓ reshape() is a predefined method in numpy array.
- ✓ We should access this method by using numpy array object
- ✓ By using this we can change the shape of an array.

Program      reshape() method  
Name          demo4.py

```
import numpy as np

details = [[10, 20, 30], [40, 50, 60]]
sales = np.array(details)
print(sales)
print()
print(sales.reshape(3, 2))
```

Output

```
[[10 20 30]
 [40 50 60]]

[[10 20]
 [30 40]
 [50 60]]
```

## Data Science – Numpy Important Methods

**Program Name** reshape() method  
demo5.py

```
import numpy as np

details = [[10, 20, 30], [40, 50, 60]]
sales = np.array(details)
print(sales)
print()
print(sales.reshape(1, 6))
```

**Output**

```
[[10 20 30]
 [40 50 60]]

[[10 20 30 40 50 60]]
```

## Data Science – Numpy Important Methods

**Program Name** reshape() method  
demo6.py

```
import numpy as np

details = [[10, 20, 30], [40, 50, 60]]
sales = np.array(details)
print(sales)
print()
print(sales.reshape(6, 1))
```

**Output**

```
[[10 20 30]
 [40 50 60]]

[[10]
 [20]
 [30]
 [40]
 [50]
 [60]]
```

## Data Science – Numpy Important Methods

### 6. count\_nonzero(p) function

- ✓ count\_nonzero(p) is a predefined function in numpy array.
- ✓ We should access this function by using numpy.
- ✓ By using this we can get non zero values from numpy

Program      count\_nonzero(p) function

Name           demo7.py

```
import numpy as np

details = [[10, 20, 30], [40, 50, 60]]
sales = np.array(details)
print(sales)
print()
print(np.count_nonzero(sales))
```

Output

```
[[10  0  30]
 [40  50  0]]

4
```

## Data Science – Numpy Important Methods

### 7. sort() method

- ✓ sort() is a predefined method in numpy array.
- ✓ We should access this method by using numpy array object
- ✓ By using this we can sort values in array.

Program      sort() method  
Name          demo8.py

```
import numpy as np

details = [[55, 13, 12], [99, 2, 1]]
sales = np.array(details)
print(sales)
sales.sort()

print()
print(sales)
```

Output

```
[[55 13 12]
 [99 2 1]]

[[12 13 55]
 [ 1  2 99]]
```

## Data Science – Numpy Important Methods

### 8. flatten() method

- ✓ flatten() is a predefined method in numpy array.
- ✓ We should access this method by using numpy array object
- ✓ This method keeps all values in one dimension array.

Program      flatten() method  
Name          demo9.py

```
import numpy as np

details = [[10, 20, 30], [40, 50, 60]]
sales = np.array(details)
print(sales)
print()
print(sales.flatten())
```

Output

```
[[10 20 30]
 [40 50 60]]

[[10 20]
 [30 40]
 [50 60]]
```

## Data Science – Numpy Important Methods

### 9. adding value to array of values

- ✓ Based on requirement we can add value to array of values.

**Program Name** Adding value to array of values  
demo10.py

```
import numpy as np

details = [[10, 20, 30], [40, 50, 60]]
sales = np.array(details)
print(sales)
print()
print(sales + 2)
```

**Output**

```
[[10 20 30]
 [40 50 60]]

[[12 22 32]
 [42 52 62]]
```

## Data Science – Numpy Important Methods

### 10. Diagonal of a Matrix

- ✓ Diagonal elements of a matrix.

**Program Name** Diagonal matrix

**Name** demo11.py

```
import numpy as np

matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

print(matrix)
print()
print(matrix.diagonal())
```

**Output**

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]

[1 5 9]
```

## Data Science – Numpy Important Methods

### 11. Trace of a Matrix

- ✓ The trace of a matrix is the sum of the diagonal elements.

**Program Name** Trace of the matrix  
demo12.py

```
import numpy as np

matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

print(matrix)
print()
print(matrix.trace())
```

**Output**

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]

15
```

## Data Science – Numpy Important Methods

**Program Name** Trace of the matrix  
demo13.py

```
import numpy as np

matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

print(matrix)
print()
print(sum(matrix.diagonal()))
```

**Output**

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]

15
```

## Data Science – Numpy Important Methods

### 12. Adding and Subtracting Matrices

- ✓ We can add & subtract two matrices.
- ✓ We need to call add and subtract functions

**Program Name** Adding two matrices  
demo14.py

```
import numpy as np

matrix_a = np.array([[1, 1, 1], [1, 1, 1], [1, 1, 2]])
matrix_b = np.array([[1, 3, 1], [1, 3, 1], [1, 3, 8]])

print(matrix_a)
print()
print(matrix_b)
print()
print(np.add(matrix_a, matrix_b))
```

**Output**

```
[[1 1 1]
 [1 1 1]
 [1 1 2]]

[[1 3 1]
 [1 3 1]
 [1 3 8]]

[[ 2  4  2]
 [ 2  4  2]
 [ 2  4 10]]
```

## Data Science – Numpy Important Methods

**Program Name** Subtracting two matrices  
demo15.py

```
import numpy as np

matrix_a = np.array([[1, 1, 1], [1, 1, 1], [1, 1, 2]])
matrix_b = np.array([[1, 3, 1], [1, 3, 1], [1, 3, 8]])

print(matrix_a)
print()
print(matrix_b)
print()
print(np.subtract(matrix_a, matrix_b))
```

**Output**

```
[[ 0 -2  0]
 [ 0 -2  0]
 [ 0 -2 -6]]
```

## Data Science – Numpy Important Methods

**Program Name** Adding two matrices  
demo16.py

```
import numpy as np

matrix_a = np.array([[1, 1, 1], [1, 1, 1], [1, 1, 2]])
matrix_b = np.array([[1, 3, 1], [1, 3, 1], [1, 3, 8]])

print(matrix_a + matrix_b)
```

**Output**

```
[[ 2  4  2]
 [ 2  4  2]
 [ 2  4 10]]
```

**Program Name** Subtracting two matrices  
demo17.py

```
import numpy as np

matrix_a = np.array([[1, 1, 1], [1, 1, 1], [1, 1, 2]])
matrix_b = np.array([[1, 3, 1], [1, 3, 1], [1, 3, 8]])

print(matrix_a - matrix_b)
```

**Output**

```
[[ 0 -2  0]
 [ 0 -2  0]
 [ 0 -2 -6]]
```

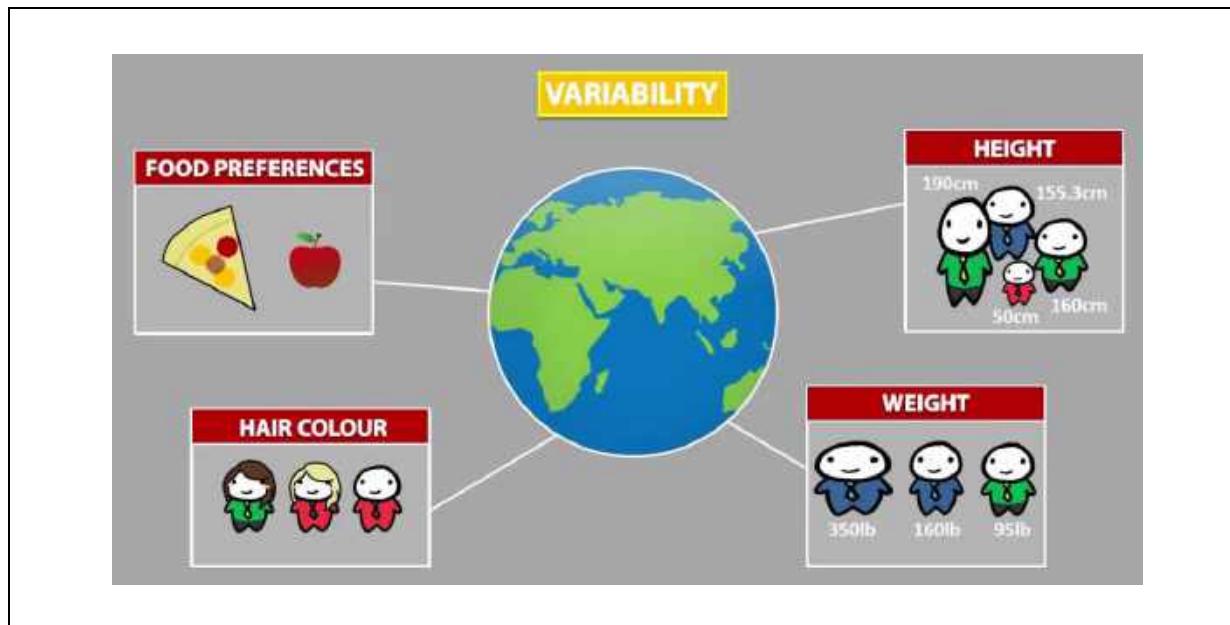
**1. Maths - Statistics - Part - 1****Contents**

<b>1. Statistics.....</b>	2
<b>2. Types of statistics .....</b>	3
2.1. Inferential statistics.....	3
2.2. Descriptive statistics .....	3
<b>3. Population .....</b>	4
<b>4. Sample .....</b>	5
<b>5. Variable.....</b>	6
<b>7. Types of variables.....</b>	9
<b>8. Types of Categorical variable.....</b>	11
8.1. Ordinal.....	11
8.2. Nominal.....	11
<b>9. Types of Quantitative variables.....</b>	12
9.1. Discrete variables.....	12
9.2. Continuous variables.....	12

## 1. Maths - Statistics - Part - 1

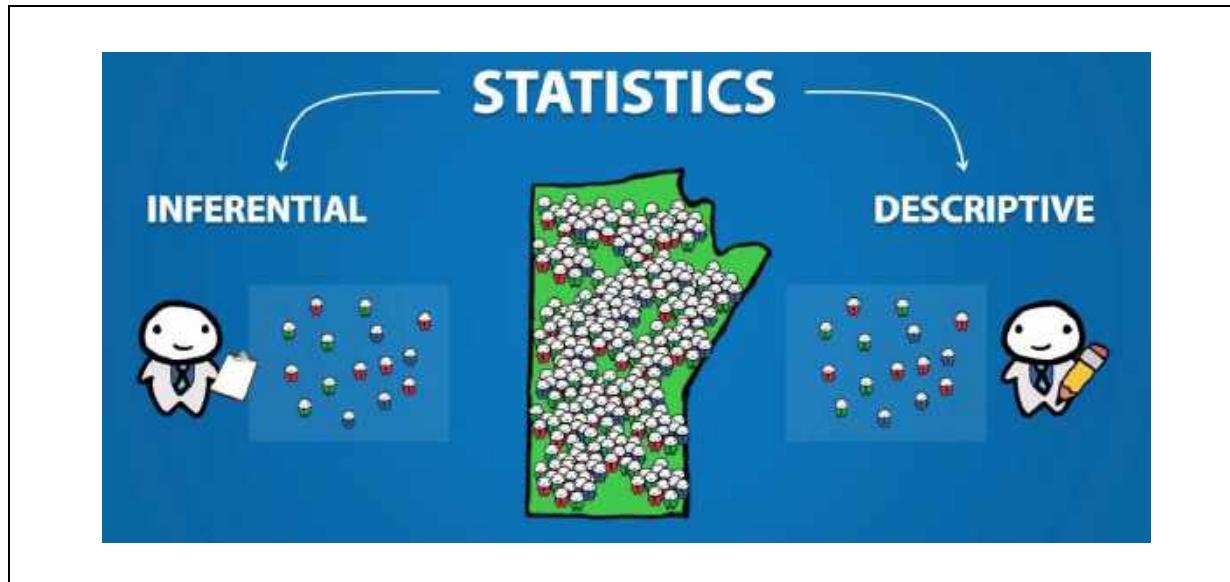
### 1. Statistics

- ✓ Statistics is the branch of mathematics dealing with the,
  - Data collection
  - Data Analysis,
  - Interpretation
  - Data presentation
  - Organizing the numerical data.



## 2. Types of statistics

- ✓ Inferential statistics
- ✓ Descriptive statistics



### 2.1. Inferential statistics

- ✓ Many times, a collection of the entire data is impossible.
- ✓ Hence a subset of the data points is collected.
- ✓ From the subset we can get conclusions about the entire population.
- ✓ This is called as inferential statistics.

### 2.2. Descriptive statistics

- ✓ These are used to summarize data, such as the mean, standard deviation & etc

#### Kind Info

- ✓ While applying statistics on data we can find underlying hidden relationships in between the variables

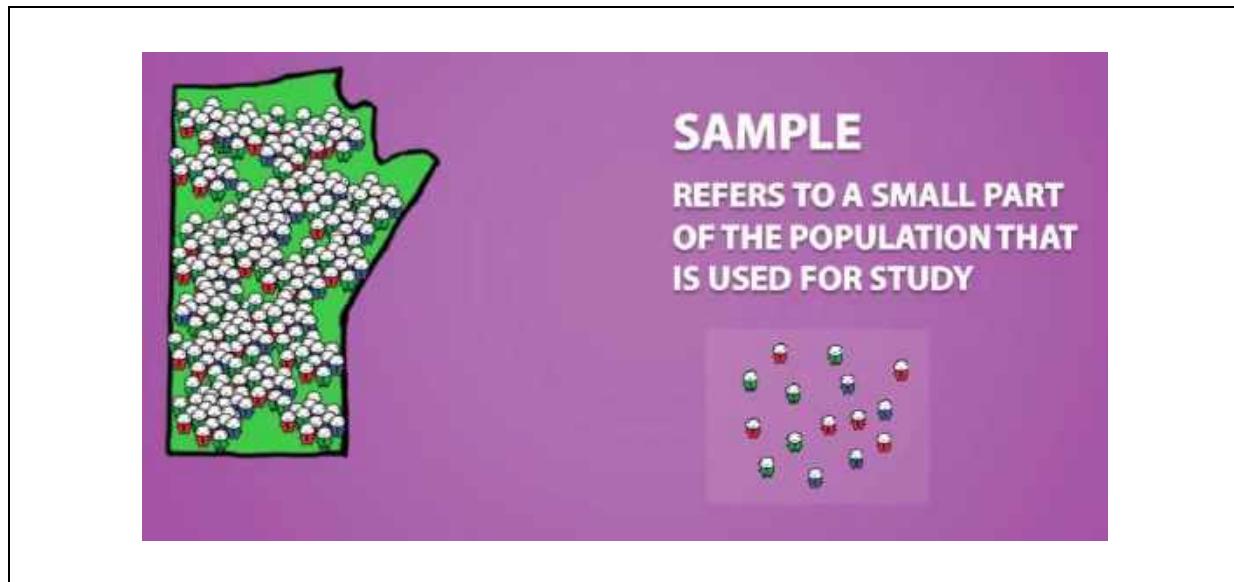
### 3. Population

- ✓ This point speaks about totality means all values or complete list of observations
- ✓ All the data points about the subject under study
  - It can be people, vehicles, sales, cats, houses & etc



#### 4. Sample

- ✓ A sample is a subset of a population, usually a small portion of the population that is being analysed.



#### Info

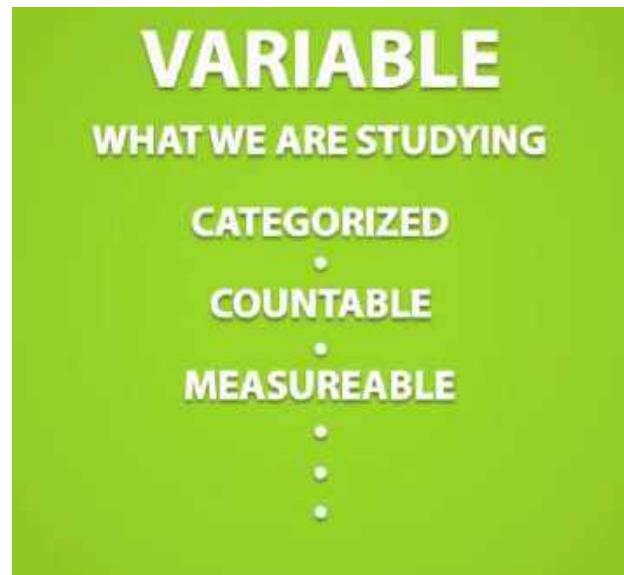
- ✓ Usually, it is expensive to perform an analysis on an entire population.
- ✓ So, by analysing sample helps to draw the conclusions about a population.

## 5. Variable

- ✓ In statistics what we are examining is a **variable**

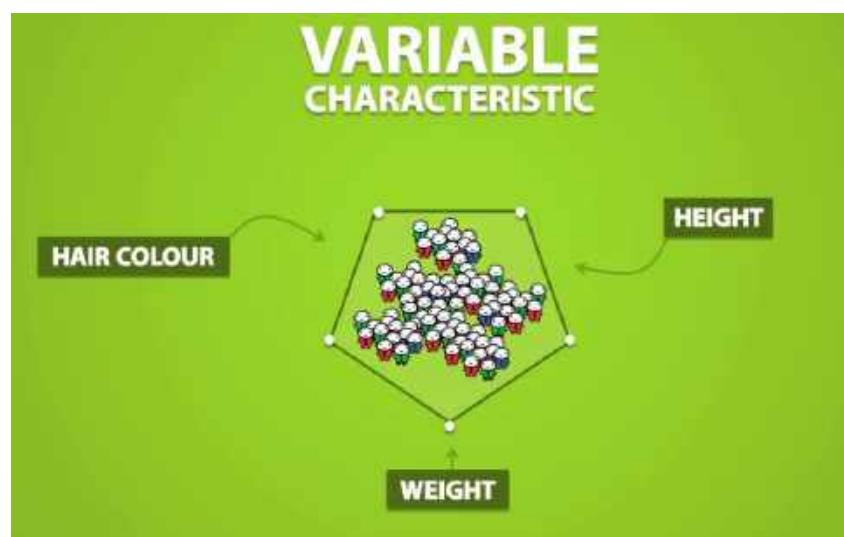


- ✓ So, a variable is measurable, countable, categorized...



## Data Science – Maths – Part - 1

- ✓ People have different kind of heights, weights, and colors
- ✓ These are all variables



## 6. Variables means

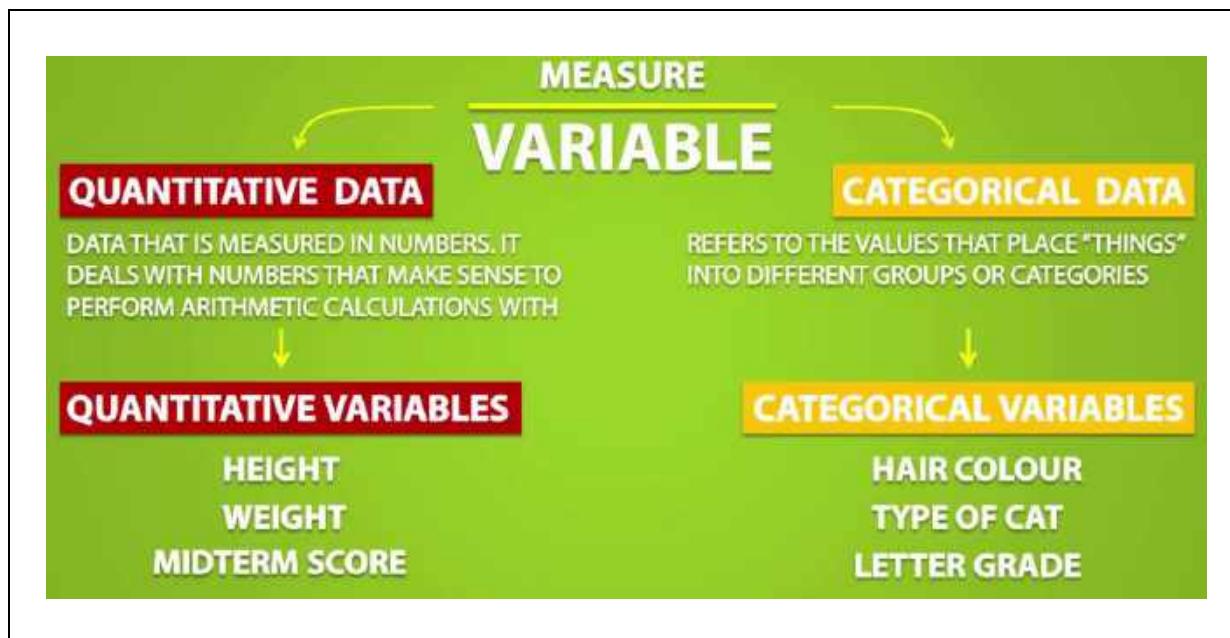
- ✓ Variable represent the characteristic representation of study.
- ✓ A variable is an attribute
- ✓ A variable has a value
- ✓ A variable can store any type of data

### Example

- ✓ Person\_name = Daniel
- ✓ Person\_age = 16
- ✓ Person\_type = male
- ✓ Person\_salary = 16000
- ✓ Person\_height = 5.9

## 7. Types of variables

- ✓ Quantitative
  - Measured in numbers
- ✓ Categorical variable or Qualitative
  - Different group of categories



### 7.1. Quantitative variables

- ✓ Quantitative variables are numeric.
- ✓ We can do some kind of arithmetic calculations
- ✓ They represent a measurable quantity.
- ✓ Example
  - Height
  - Weight
  - Midterm score
  - Population of a city.
  - Salary and etc

### 7.2. Categorical or Qualitative variables

- ✓ Categorical variables stores values which are names or labels
- ✓ Example
  - Type of person : male or female
  - Review about food: good, bad, ok
  - Color of the bike : red, green, blue

## 8. Types of Categorical variable

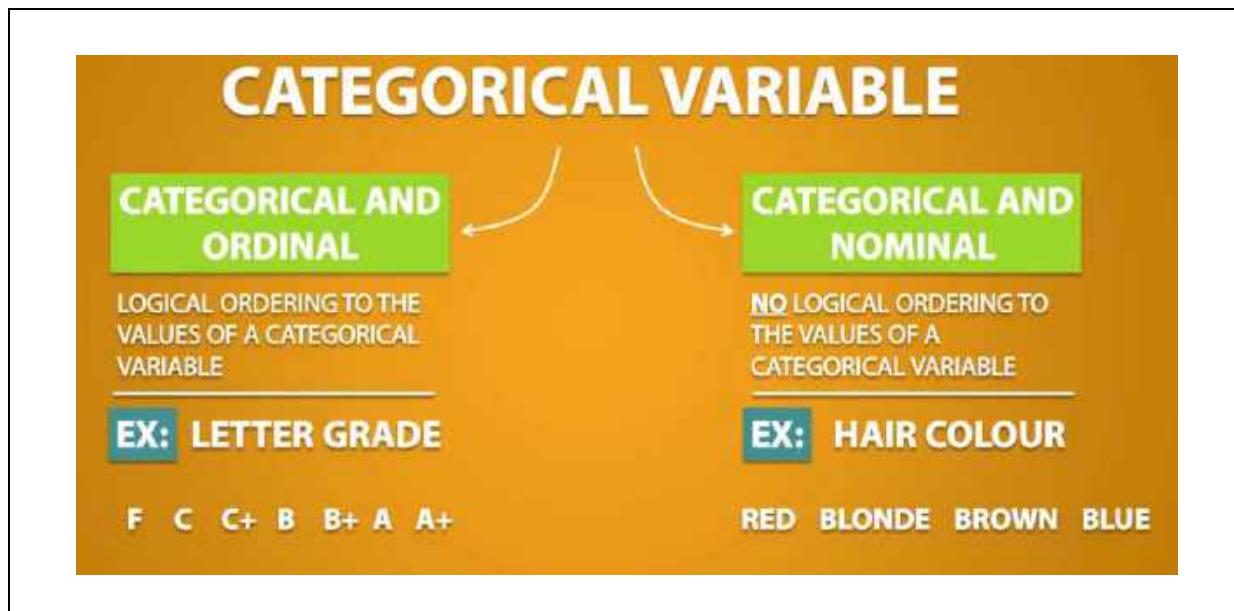
- ✓ There are two types of categorical variables
  - Ordinal
  - Nominal

### 8.1. Ordinal

- ✓ Logical order is possible to do analysis.
- ✓ We can logical order according to some dictionary format
- ✓ After this we can do some analysis
  - Example : Grades in exams

### 8.2. Nominal

- ✓ There is no logical ordering with respect to the actual values
  - Example : Hair color



## 9. Types of Quantitative variables

- ✓ Discrete variables
- ✓ Continuous variables

### 9.1. Discrete variables

- ✓ Discrete variables are like whole numbers.
- ✓ Example:
  - Number pets own
  - Number of people in a family
  - Number of bikes or cars

### 9.2. Continuous variables

- ✓ Continuous variables are like normal numbers includes floating point numbers.
- ✓ Example:
  - Weight
  - Salary
  - Bank balance



**2. Maths - Statistics – PART – 2****Contents**

1. Usage of mode, median, mean, range & standard deviation? .....	2
2. Measures of centre .....	3
3. Mode.....	4
4. Median.....	5
6. Mean.....	11
7. Measures of spread.....	14
8. Range .....	15
9. Standard Deviation.....	16
10. Variance .....	26

**2. Maths - Statistics – PART – 2**

**MODE      MEDIAN      MEAN  
RANGE      STANDARD DEVIATION**

**1. Usage of mode, median, mean, range & standard deviation?**

- ✓ Identifying and describing like how the dataset got distributed
- ✓ These mode, median, mean, range and standard deviation gives the numerical information and distribution about the dataset
- ✓ These also explains about
  - Measures of centre
  - Measures of spread



## 2. Measures of centre

- ✓ Mode
- ✓ Median
- ✓ Mean



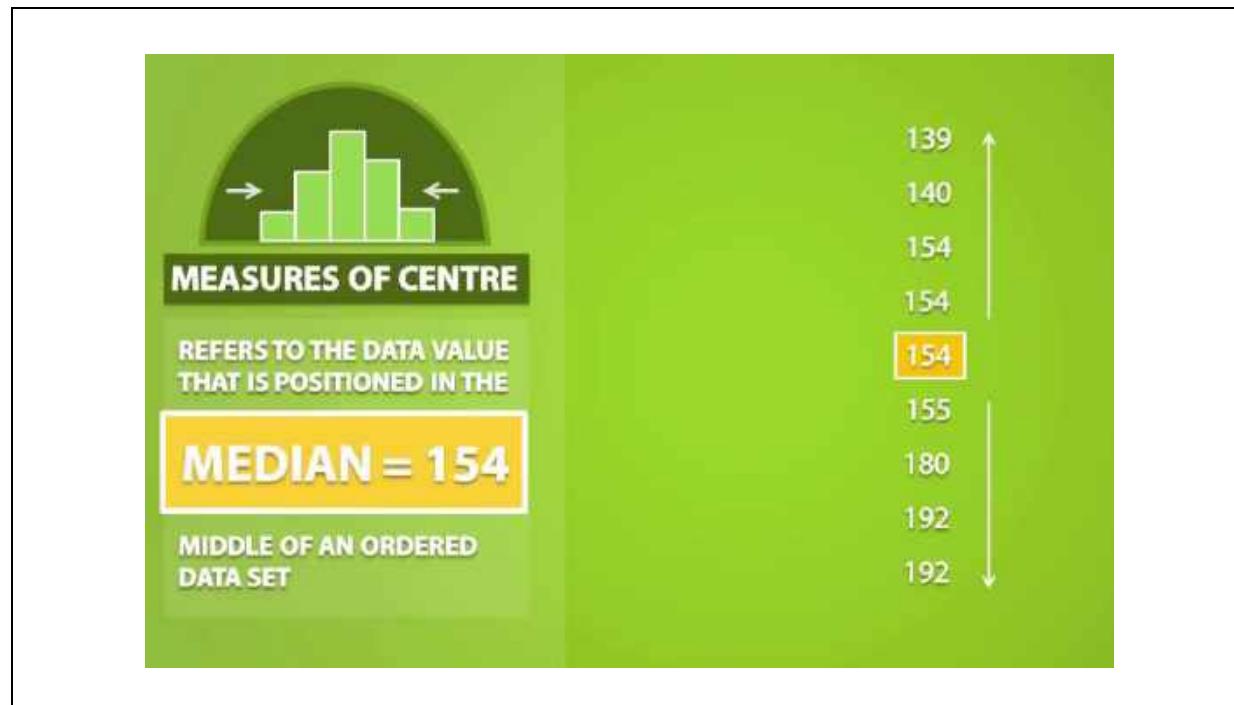
### 3. Mode

- ✓ Value which is most frequently observed
- ✓ Suppose we have taken random people heights and displayed as below
- ✓ Here 153 value is repeated in 3 times



#### 4. Median

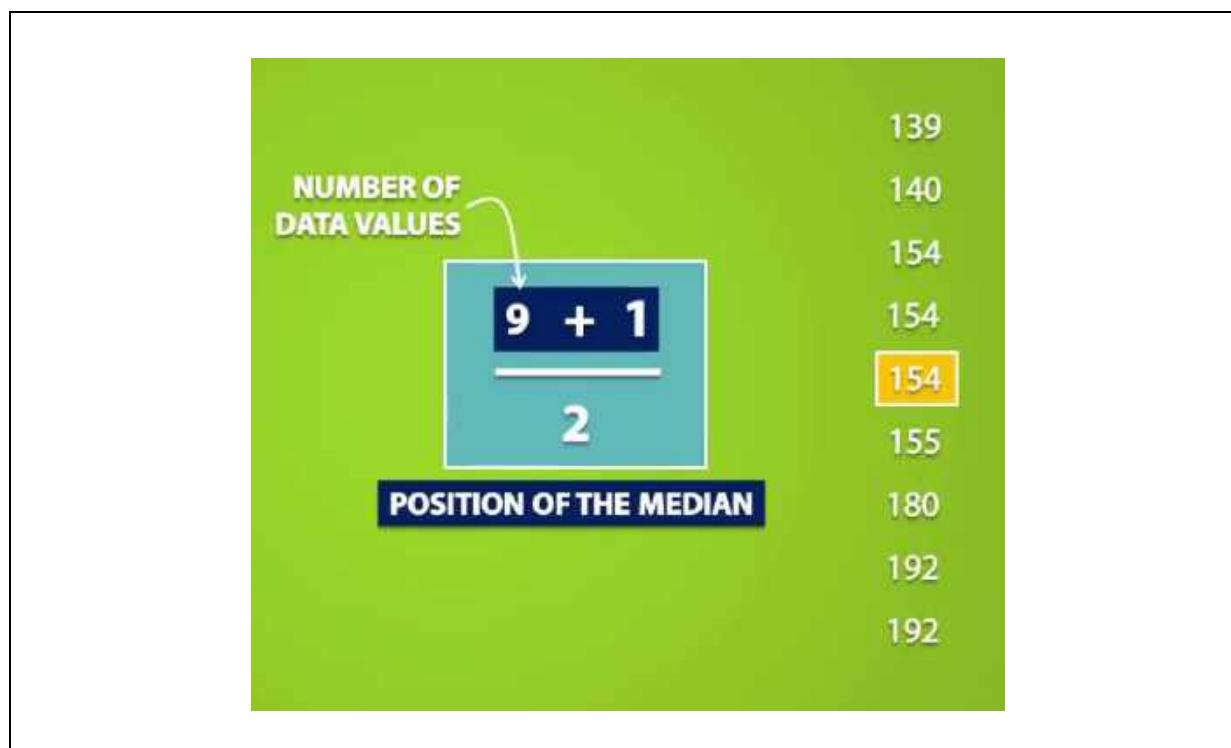
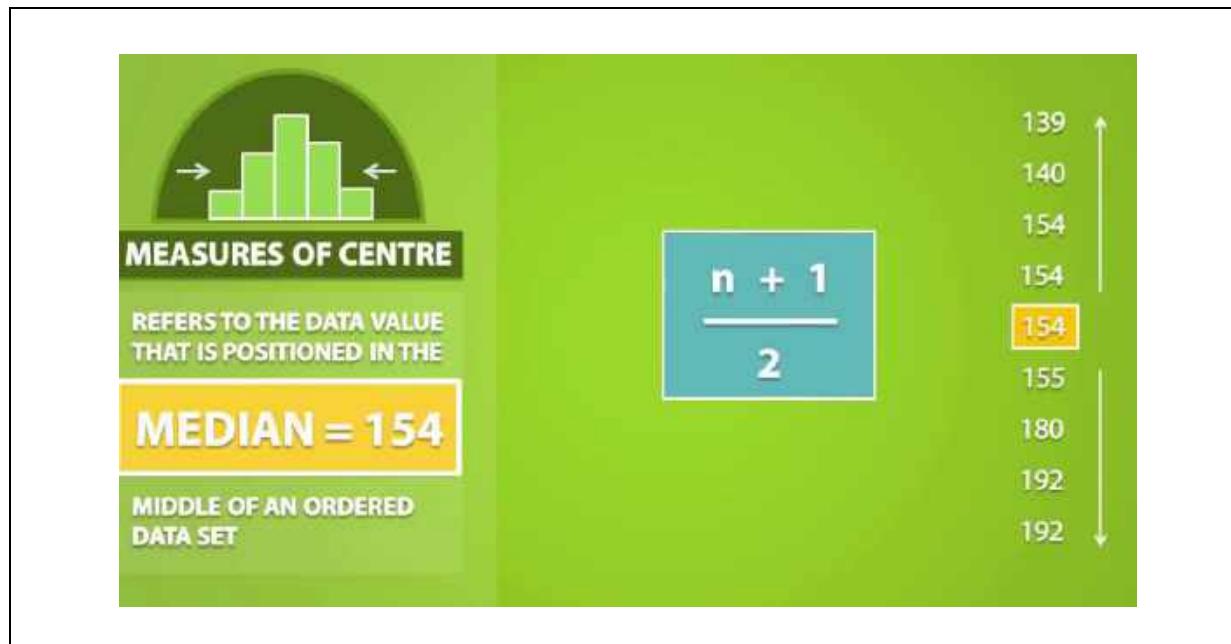
- ✓ Value that is positioned in the middle of an **ordered dataset**
- ✓ First we need to keep the data into an order
- ✓ We usually order the dataset into smallest to largest



## Data Science – Maths – Part - 2

### Special cases

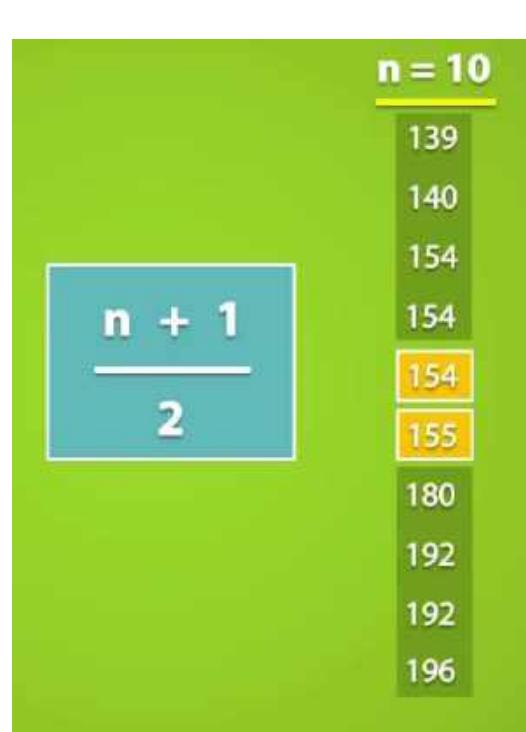
- ✓ If the dataset is extremely large then it might helpful use the below formula

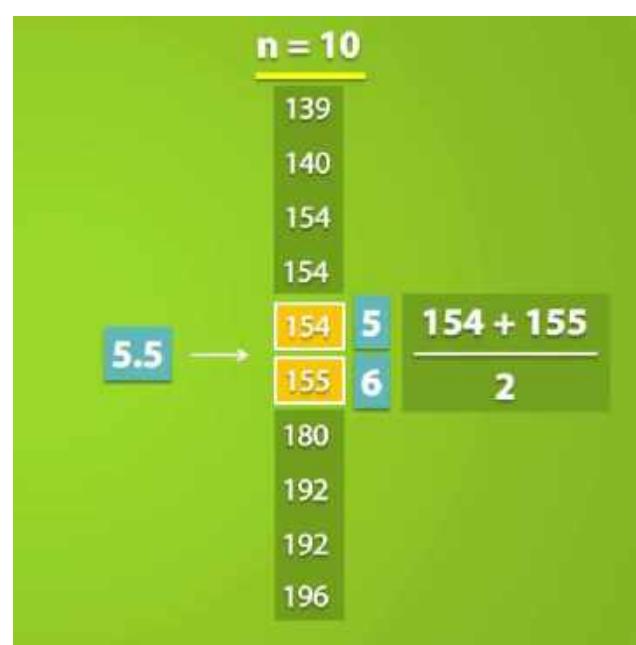


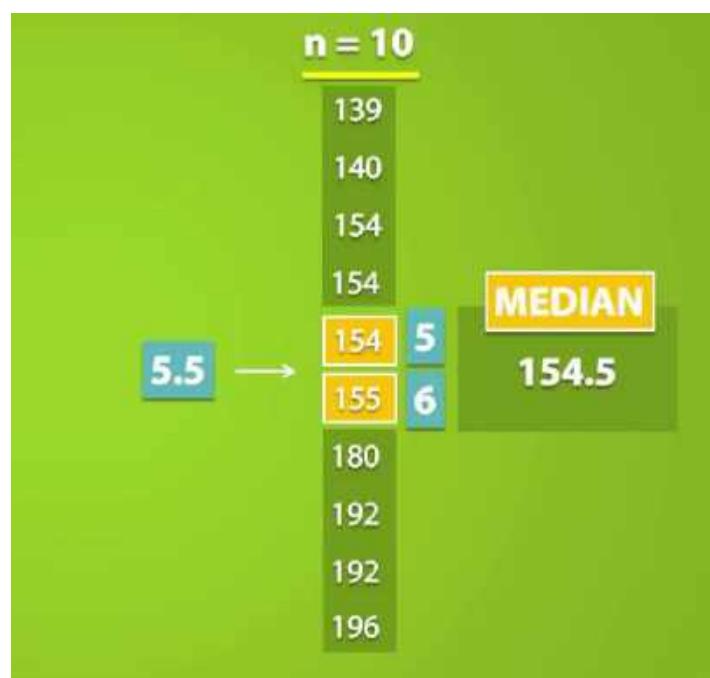


## Data Science – Maths – Part - 2

- ✓ If number of values are in odd or even number then we do have some special scenarios to find out the median value







## 6. Mean

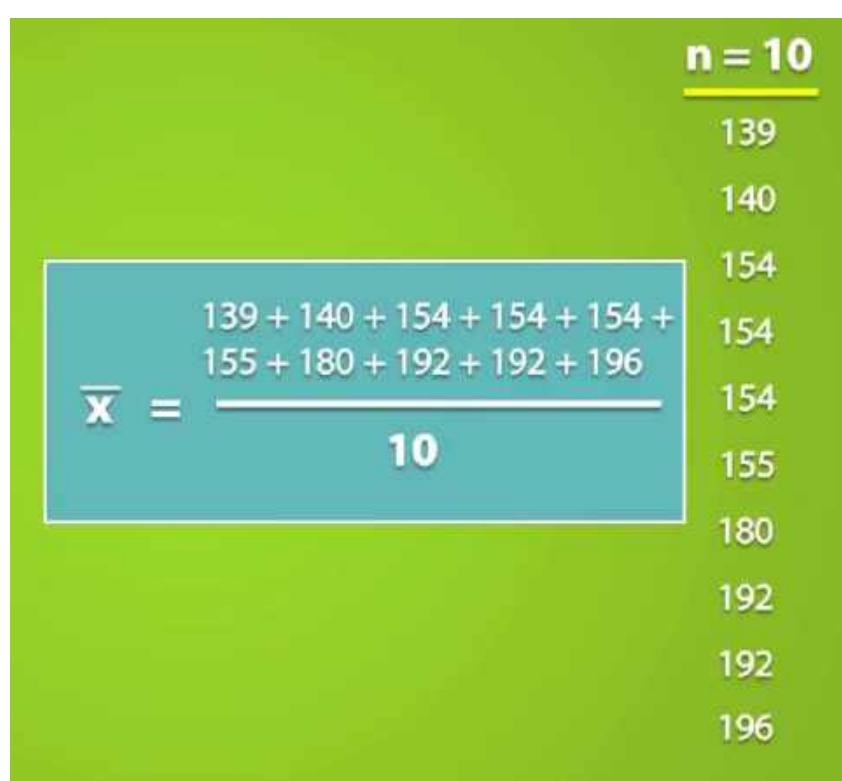
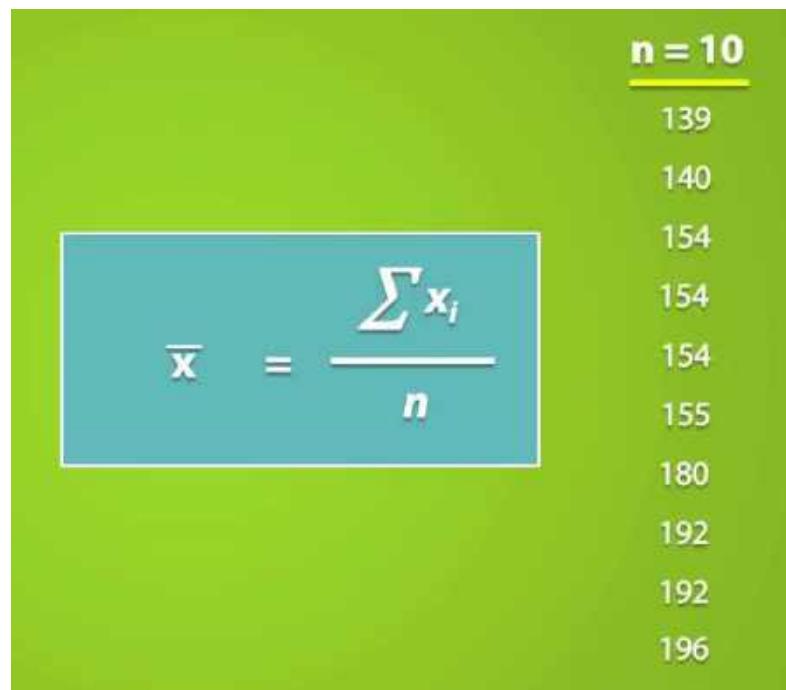
- ✓ The mean is just another name of average
- ✓ Below is the formula which indicates mean of total values

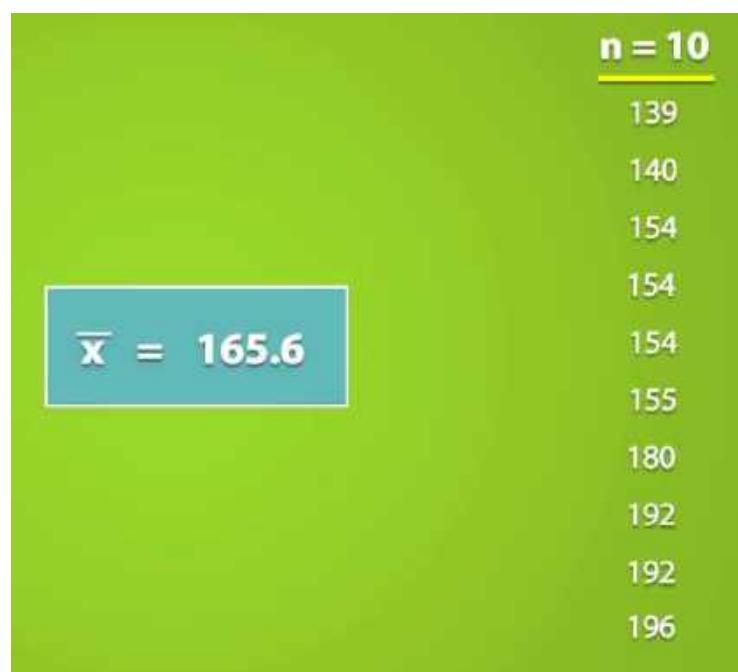
The diagram illustrates the formula for Mean. It features a central blue box containing the equation  $\text{MEAN} = \frac{\sum x_i}{n}$ . To the right of this equation, the words "SUMMATION OF ALL DATA VALUES" are stacked vertically, followed by "DIVIDED BY" and "TOTAL NUMBER OF DATA VALUES". The entire formula is set against a green background.

## Sample mean

- ✓ Below is the formula which indicates mean of sample values

The diagram illustrates the formula for Sample Mean. It features a central blue box containing the equation  $\bar{x} = \frac{\sum x_i}{n}$ . The background is white.



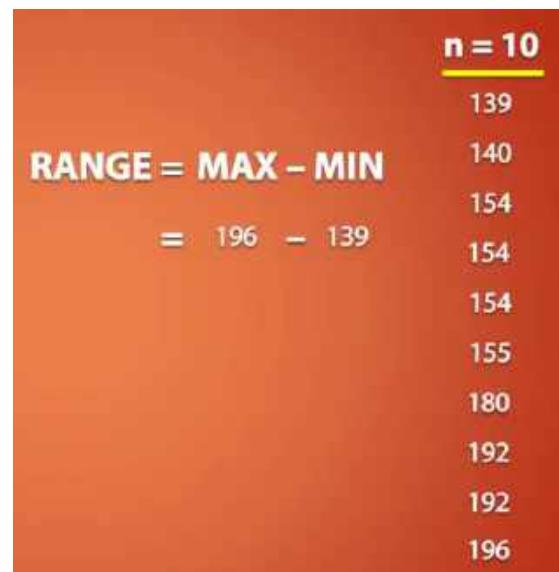


## 7. Measures of spread

- ✓ Range
- ✓ Standard deviation

### 8. Range

- ✓ Range means difference in between minimum value and maximum value
- ✓ It explains about the data is in between min and max values



## 9. Standard Deviation

- ✓ The Standard Deviation is a measure of how spread out numbers.
- ✓ Formula is very simple, It is the square root of the Variance

### STANDARD DEVIATION

$$s = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n - 1}}$$

10  
12  
16  
19  
20

### STANDARD DEVIATION

$$s = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n - 1}}$$

$x_i$	$x_i - \bar{x}$	$(x_i - \bar{x})^2$
10		
12		
16		
19		
20		

$$s = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n - 1}}$$

## Data Science – Maths – Part - 2

---

$x_i$	$x_i - \bar{x}$	$(x_i - \bar{x})^2$
10		
12		
16		
19		
20		
<hr/>		

$$s = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n-1}}$$

$$\bar{x} = \frac{\sum x_i}{n}$$

$x_i$	$x_i - \bar{x}$	$(x_i - \bar{x})^2$
10		
12		
16		
19		
20		
<hr/>		

$$s = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n-1}}$$

$$\bar{x} = \frac{10 + 12 + 16 + 19 + 20}{5}$$

$x_i$	$x_i - \bar{x}$	$(x_i - \bar{x})^2$
10		
12		
16		
19		
20		

$\bar{x} = 15.4$

$$s = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n-1}}$$

$x_i$	$x_i - \bar{x}$	$(x_i - \bar{x})^2$
10	10 – 15.4	
12	12 – 15.4	
16	16 – 15.4	
19	19 – 15.4	
20	20 – 15.4	

$\bar{x} = 15.4$

$$s = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n-1}}$$

## Data Science – Maths – Part - 2

---

$x_i$	$x_i - \bar{x}$	$(x_i - \bar{x})^2$
10	-5.4	
12	-3.4	
16	0.6	
19	3.6	
20	4.6	
$\bar{x} = 15.4$		

$$s = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n - 1}}$$

$x_i$	$x_i - \bar{x}$	$(x_i - \bar{x})^2$
10	-5.4	( -5.4 ) <sup>2</sup>
12	-3.4	( -3.4 ) <sup>2</sup>
16	0.6	( 0.6 ) <sup>2</sup>
19	3.6	( 3.6 ) <sup>2</sup>
20	4.6	( 4.6 ) <sup>2</sup>

$$s = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n - 1}}$$

## Data Science – Maths – Part - 2

---

$x_i$	$x_i - \bar{x}$	$(x_i - \bar{x})^2$
10	-5.4	29.16
12	-3.4	11.56
16	0.6	0.36
19	3.6	12.96
20	4.6	21.16

$$s = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n - 1}}$$

$x_i$	$x_i - \bar{x}$	$(x_i - \bar{x})^2$
10	-5.4	29.16
12	-3.4	11.56
16	0.6	0.36
19	3.6	12.96
20	4.6	21.16

$$s = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n - 1}}$$

**SUM = 75.2**

## Data Science – Maths – Part - 2

---

$X_i$	$X_i - \bar{X}$	$(X_i - \bar{X})^2$
10	-5.4	29.16
12	-3.4	11.56
16	0.6	0.36
19	3.6	12.96
20	4.6	21.16

$$s = \sqrt{\frac{75.2}{n - 1}}$$

$X_i$	$X_i - \bar{X}$	$(X_i - \bar{X})^2$
10	-5.4	29.16
12	-3.4	11.56
16	0.6	0.36
19	3.6	12.96
20	4.6	21.16

$$s = \sqrt{\frac{75.2}{5 - 1}}$$

## Data Science – Maths – Part - 2

---

$x_i$	$x_i - \bar{x}$	$(x_i - \bar{x})^2$
10	-5.4	29.16
12	-3.4	11.56
16	0.6	0.36
19	3.6	12.96
20	4.6	21.16

$$s = \sqrt{\frac{75.2}{4}}$$

$x_i$	$x_i - \bar{x}$	$(x_i - \bar{x})^2$
10	-5.4	29.16
12	-3.4	11.56
16	0.6	0.36
19	3.6	12.96
20	4.6	21.16

$$s = \sqrt{18.8}$$

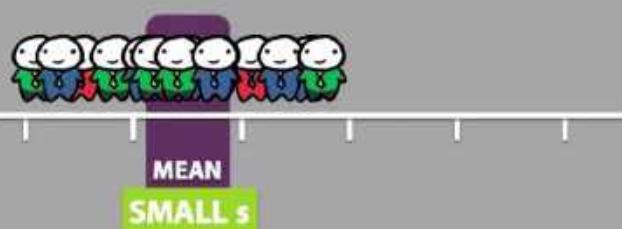
$x_i$	$x_i - \bar{x}$	$(x_i - \bar{x})^2$	$s = 4.336$
10	-5.4	29.16	
12	-3.4	11.56	
16	0.6	0.36	
19	3.6	12.96	
20	4.6	21.16	

**WHAT DOES THE  
STANDARD DEVIATION  
EVEN TELL US?**

**STANDARD DEVIATION**  
**HOW CLOSE THE VALUES IN A DATA SET**  
**ARE TO THE MEAN**

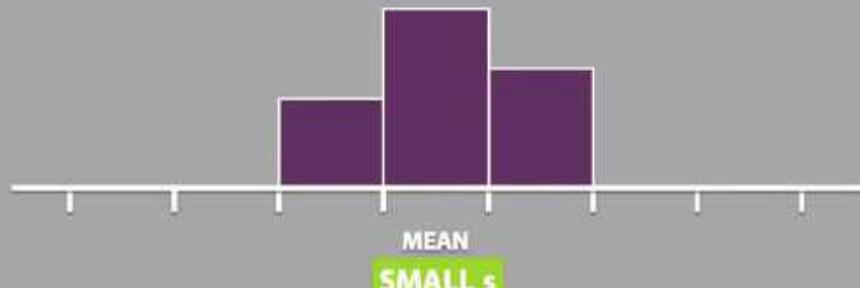
## STANDARD DEVIATION

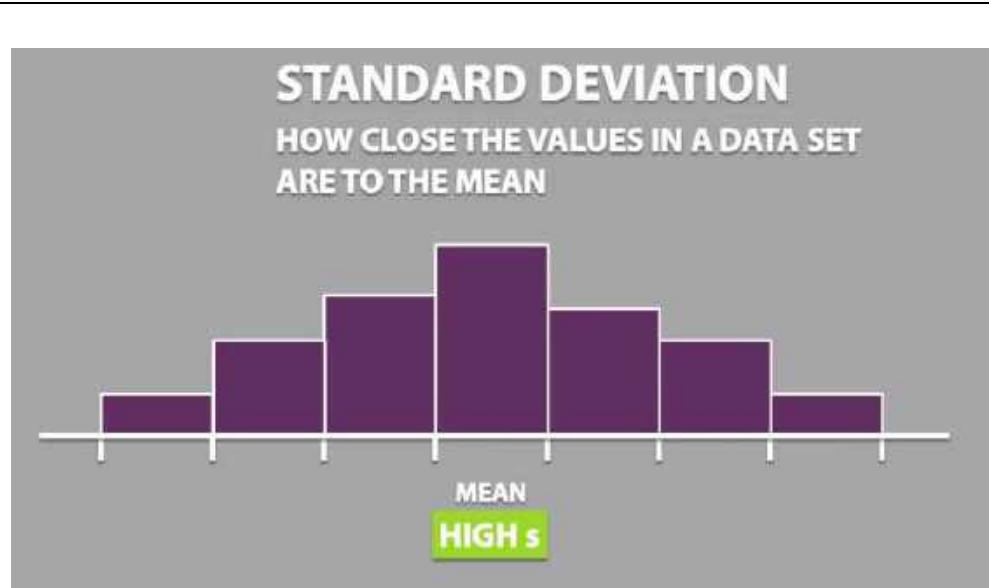
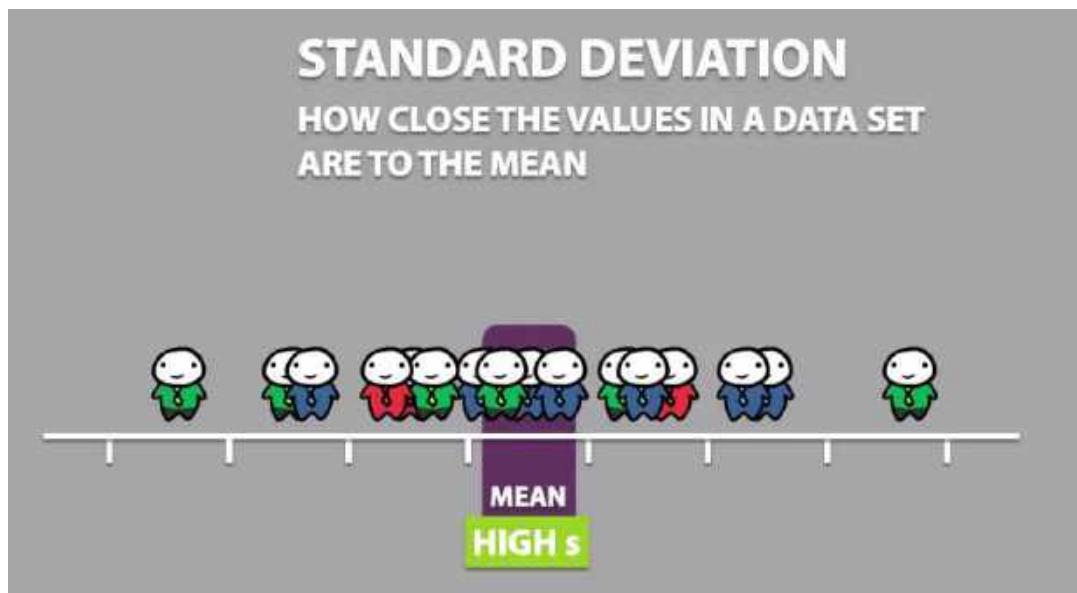
HOW CLOSE THE VALUES IN A DATA SET  
ARE TO THE MEAN



## STANDARD DEVIATION

HOW CLOSE THE VALUES IN A DATA SET  
ARE TO THE MEAN





## 10. Variance

- ✓ The average of squared differences from the mean.
- ✓ Variance is the average of squared differences from the mean
- ✓ By using this we can find how far the data points in a population are from the population mean.

<p style="text-align: center;"><b>VARIANCE</b></p> $s^2 = \frac{\sum (x_i - \bar{x})^2}{n - 1}$	<p style="text-align: center;"><b>STANDARD DEVIATION</b></p> $s = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n - 1}}$
---	--

<p style="text-align: center;"><b>SAMPLE VARIANCE</b></p> $s^2 = \frac{\sum (x_i - \bar{x})^2}{n - 1}$	<p style="text-align: center;"><b>SAMPLE STANDARD DEVIATION</b></p> $s = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n - 1}}$
--	---

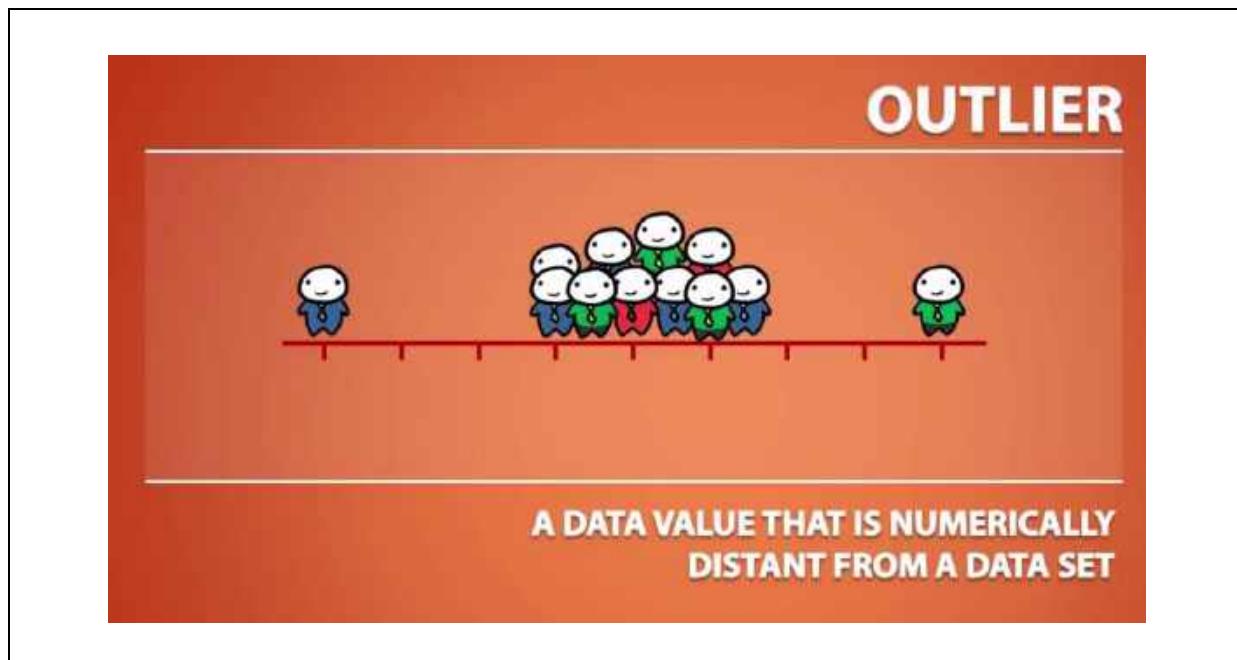
**3. Maths - Statistics – PART – 3****Contents**

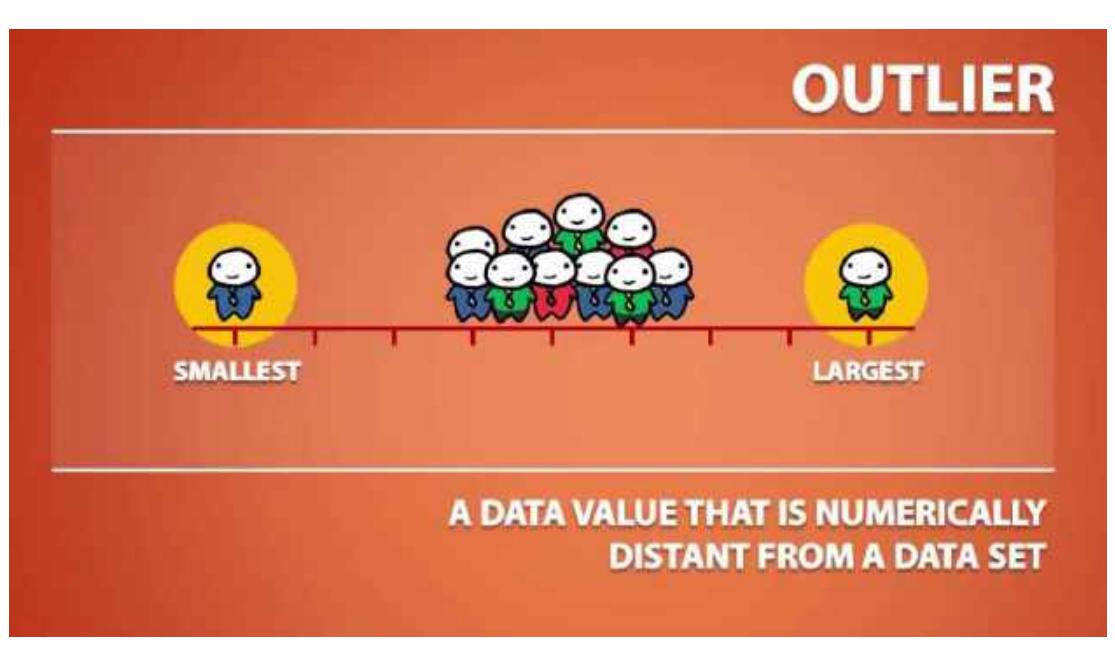
<b>1. What is an outlier? .....</b>	<b>2</b>
<b>2. Surprising...!!!.....</b>	<b>5</b>
<b>3. Checking mean, median, mode &amp; Range.....</b>	<b>9</b>

### 3. Maths - Statistics – PART – 3

#### 1. What is an outlier?

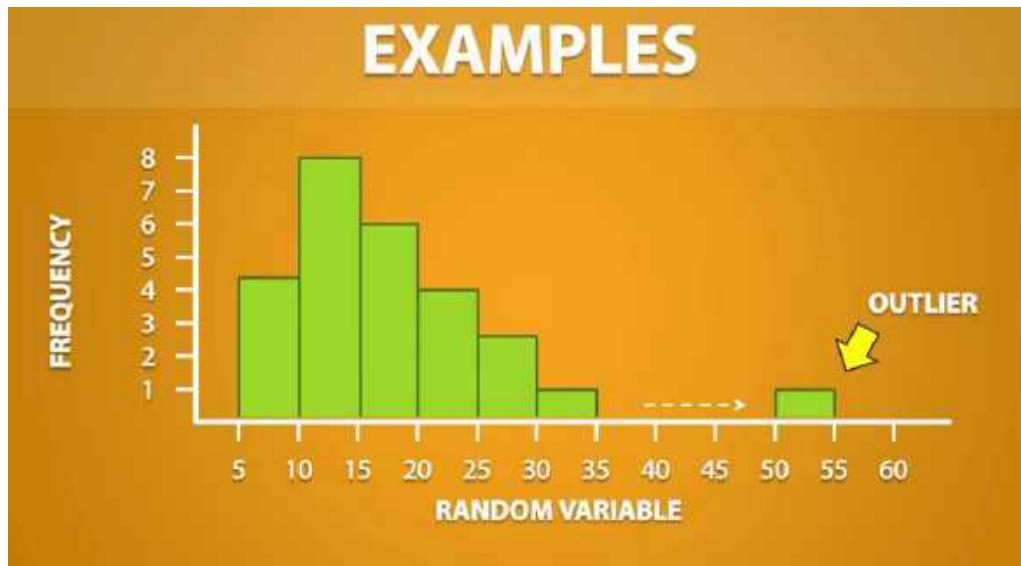
- ✓ An outlier is defined as a value which are very far from dataset
- ✓ An outlier is a data point that falls outside from main data points
- ✓ It can be largest value in dataset, smallest value in dataset





## Examples

- ✓ Observe the below histogram, a point is far from value

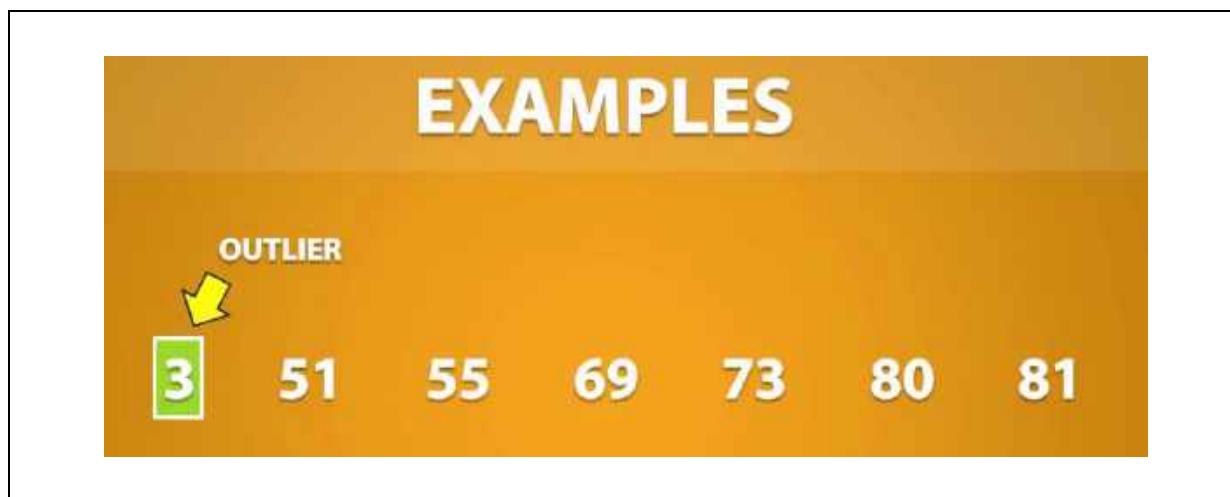


## Data Science – Maths – Part - 3

- ✓ Below example, 9000 is very larger value than other values

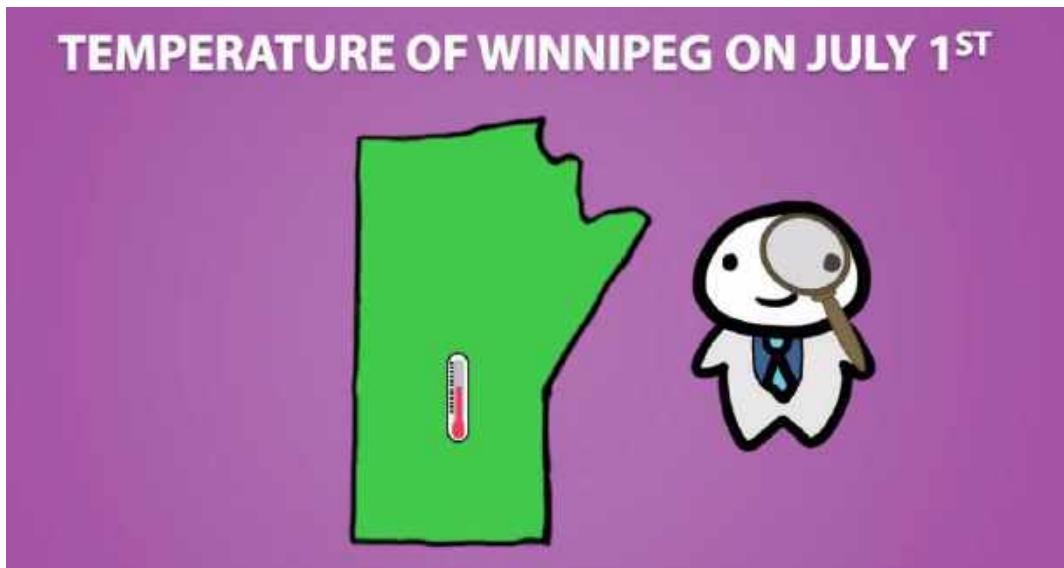


- ✓ Below example, 3 is very smaller value than other values



## 2. Surprising...!!!

- ✓ Outliers are data points but these are typical and surprising.
- ✓ These effects the measures of center and spread
- ✓ Observe the below example



**TEMPERATURE OF WINNIPEG ON JULY 1<sup>ST</sup>**

<u>YEAR</u>	<u>TEMPERATURE</u>
2015	26.0 °C
2014	15.0 °C
2013	20.5 °C
2012	31.0 °C
2011	-350.0 °C OUTLIER
2010	31.0 °C
2009	30.5 °C

### THE MEAN IS AFFECTED BY THE PRESENCE OF OUTLIERS

$$\bar{x} = \frac{\sum x_i}{n}$$

26.0 °C

15.0 °C

20.5 °C

31.0 °C

-350.0 °C

31.0 °C

30.5 °C

$$\bar{x} = \frac{26 + 15 + 20.5 + 31 + (-350) + 31 + 30.5}{7}$$

26.0 °C

15.0 °C

20.5 °C

31.0 °C

-350.0 °C

31.0 °C

30.5 °C



So

**THE MEAN IS AFFECTED BY THE  
PRESENCE OF OUTLIERS**

## Data Science – Maths – Part - 3

---

### 3. Checking mean, median, mode & Range

- ✓ Calculate mean, median, mode and Range for a dataset

<b>CALCULATIONS</b>			
DATA SET	MEASURE	WITH OUTLIER	WITHOUT OUTLIER
26.0 °C	MEAN	– 28	25.667
15.0 °C	MEDIAN		
20.5 °C	MODE		
31.0 °C	RANGE		
<b>OUTLIER</b> <b>-350.0 °C</b>			
31.0 °C			
30.5 °C			

<b>CALCULATIONS</b>			
DATA SET	MEASURE	WITH OUTLIER	WITHOUT OUTLIER
<b>OUTLIER</b> <b>-350.0 °C</b>	MEAN	– 28	25.667
15.0 °C	MEDIAN		
20.5 °C	MODE		
26.0 °C	RANGE		
30.5 °C			
31.0 °C			
31.0 °C			

## Data Science – Maths – Part - 3

---

CALCULATIONS			
OUTLIER <b>-350.0 °C</b>	MEASURE	WITH OUTLIER	WITHOUT OUTLIER
	<b>MEAN</b>	– 28	<b>25.667</b>
	<b>MEDIAN</b>		
	<b>MODE</b>		
	<b>RANGE</b>		

CALCULATIONS			
OUTLIER <b>-350.0 °C</b>	MEASURE	WITH OUTLIER	WITHOUT OUTLIER
	<b>MEAN</b>	– 28	<b>25.667</b>
	<b>MEDIAN</b>	26	<b>28.25</b>
	<b>MODE</b>		
	<b>RANGE</b>		

## Data Science – Maths – Part - 3

---

CALCULATIONS				
OUTLIER	DATA SET	MEASURE	WITH OUTLIER	WITHOUT OUTLIER
	<b>-350.0 °C</b>	MEAN	- 28	<b>25.667</b>
	<b>15.0 °C</b>	MEDIAN	26	<b>28.25</b>
	<b>20.5 °C</b>	MODE	31	<b>31</b>
	<b>26.0 °C</b>	RANGE		
	<b>30.5 °C</b>			
	<b>31.0 °C</b>			
	<b>31.0 °C</b>			

CALCULATIONS				
OUTLIER	DATA SET	MEASURE	WITH OUTLIER	WITHOUT OUTLIER
	<b>-350.0 °C</b>	MEAN	- 28	<b>25.667</b>
	<b>15.0 °C</b>	MEDIAN	26	<b>28.25</b>
	<b>20.5 °C</b>	MODE	31	<b>31</b>
	<b>26.0 °C</b>	RANGE	381	<b>16</b>
	<b>30.5 °C</b>			
	<b>31.0 °C</b>			
	<b>31.0 °C</b>			

## Data Science – Maths – Part - 3

---

- ✓ Is outlier affects the calculations: Yes then observe the below table

RESPONSE TO AN OUTLIER	MEASURE	WITH OUTLIER	WITHOUT OUTLIER
AFFECTED	MEAN	- 28	<b>25.667</b>
	MEDIAN	26	<b>28.25</b>
	MODE	31	<b>31</b>
	RANGE	381	<b>16</b>

RESPONSE TO AN OUTLIER	MEASURE	WITH OUTLIER	WITHOUT OUTLIER
AFFECTED	MEAN	- 28	<b>25.667</b>
RESISTANT	MEDIAN	26	<b>28.25</b>
RESISTANT	MODE	31	<b>31</b>
	RANGE	381	<b>16</b>

## Data Science – Maths – Part - 3

---

RESPONSE TO AN OUTLIER	MEASURE	WITH OUTLIER	WITHOUT OUTLIER
AFFECTED	MEAN	- 28	25.667
RESISTANT	MEDIAN	26	28.25
RESISTANT	MODE	31	31
AFFECTED	RANGE	381	16

  $R = \text{MAXIMUM} - \text{MINIMUM}$

RESPONSE TO AN OUTLIER	MEASURE	WITH OUTLIER	WITHOUT OUTLIER
AFFECTED	MEAN	- 28	25.667
RESISTANT	MEDIAN	26	28.25
RESISTANT	MODE	31	31
AFFECTED	RANGE	381	16
	STANDARD DEVIATION		

## Data Science – Maths – Part - 3

RESPONSE TO AN OUTLIER	MEASURE	WITH OUTLIER	WITHOUT OUTLIER
AFFECTED	MEAN	- 28	25.667
RESISTANT	MEDIAN	26	28.25
RESISTANT	MODE	31	31
AFFECTED	RANGE	381	16
STANDARD DEVIATION		$s = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n-1}}$	

RESPONSE TO AN OUTLIER	MEASURE	WITH OUTLIER	WITHOUT OUTLIER
AFFECTED	MEAN	- 28	25.667
RESISTANT	MEDIAN	26	28.25
RESISTANT	MODE	31	31
AFFECTED	RANGE	381	16
STANDARD DEVIATION		$s = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n-1}}$	

## Data Science – Maths – Part - 3

---

RESPONSE TO AN OUTLIER	MEASURE	WITH OUTLIER	WITHOUT OUTLIER
AFFECTED	MEAN	- 28	25.667
RESISTANT	MEDIAN	26	28.25
RESISTANT	MODE	31	31
AFFECTED	RANGE	381	16
AFFECTED	STANDARD DEVIATION	$s = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n - 1}}$	

**3. Maths - Statistics – PART – 4****Contents**

1. What is five number summary?	2
2. Explanation	3
3. 4 equal quarters	5
4. 1st Quartile	7
5. 3rd Quartile	9
6. Smallest and largest	10
7. Box plot	11
8. Whiskers	13
9. Interquartile range	14
10. $IQR = Q3 - Q1$	15
11. Outliers (modified box plot)	16
12. Outlier recognizing	17
13. Checking outliers	18

**4. Maths - Statistics – PART – 4****1. What is five number summary?**

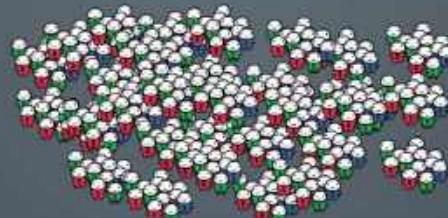
- ✓ The five number summary gives a way to describe the distribution

**FIVE NUMBER SUMMARY**  
GIVES US A WAY TO DESCRIBE A DISTRIBUTION  
USING ONLY **FIVE** NUMBERS

**MINIMUM    1<sup>ST</sup> QUARTILE    MEDIAN    3<sup>RD</sup> QUARTILE    MAXIMUM**

**FIVE NUMBER SUMMARY**  
GIVES US A WAY TO DESCRIBE A DISTRIBUTION  
USING ONLY **FIVE** NUMBERS

**FIVE NUMBER SUMMARY**  
**MINIMUM    1<sup>ST</sup> QUARTILE    MEDIAN    3<sup>RD</sup> QUARTILE    MAXIMUM**



## 2. Explanation

- ✓ So, if we took the data then we can find these five numbers in that data
- ✓ **Minimum** is **smallest** value in a dataset
- ✓ **Maximum** is **largest** value in a dataset
- ✓ **Median** is **middle** data value
  - It is the point 50% data values is **below** the median and 50% data values is **above** the median



- ✓ The median of **bottom half** is called as **1<sup>st</sup> Quartile**
  - It is the position 25% of the data values are below this and 75% values are above the point
  - The **1<sup>st</sup> Quartile** is essentially as median of the median



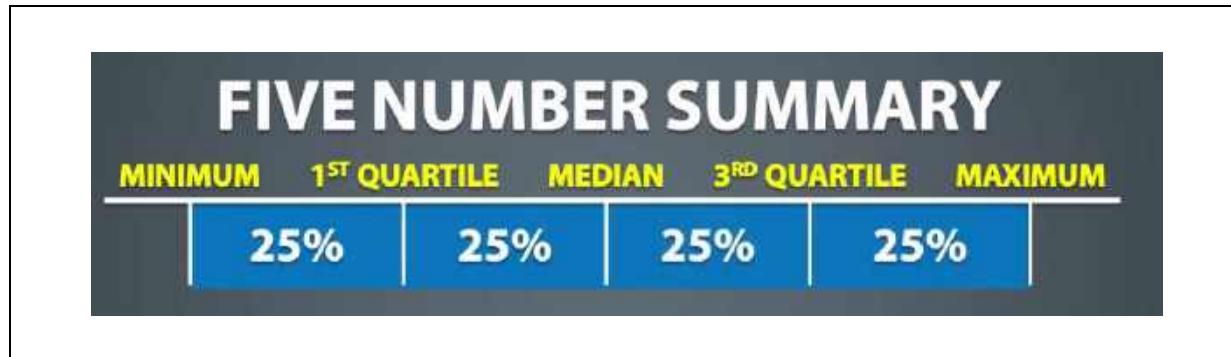
## Data Science – Maths – Part - 4

- ✓ The median of top half is called as 3<sup>rd</sup> Quartile
  - It is the position 75% of the data values are below this and 25% values are above the point



### 3. 4 equal quarters

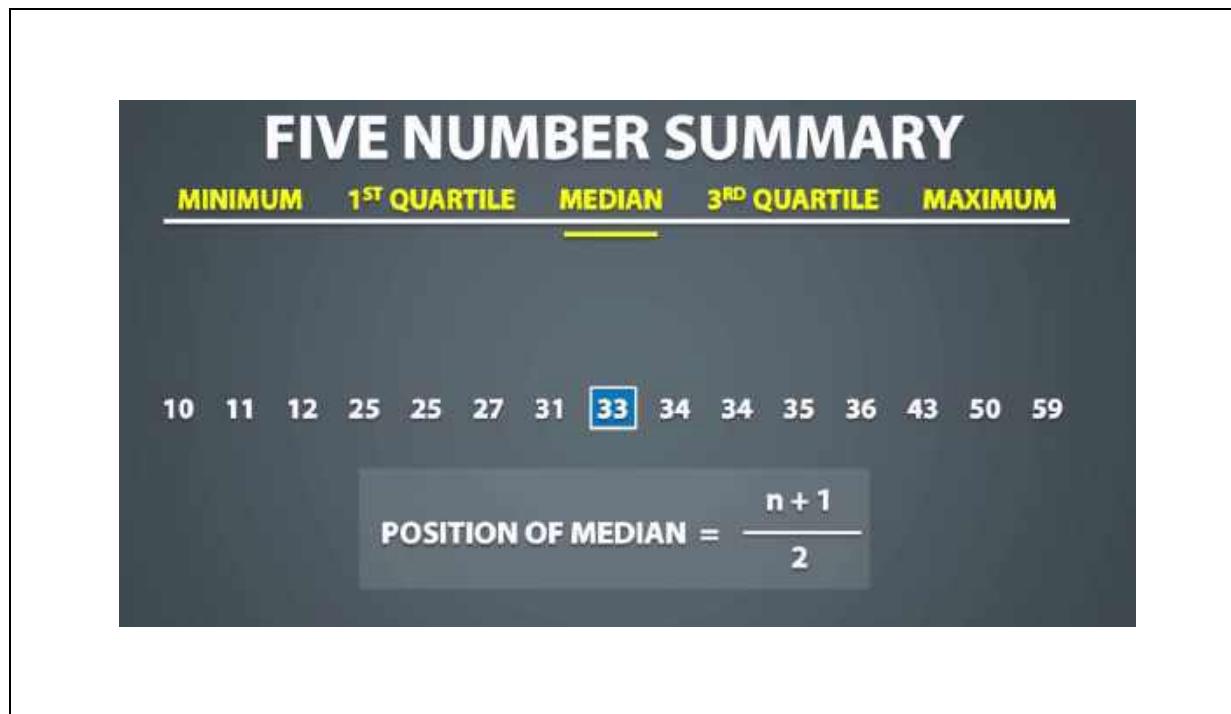
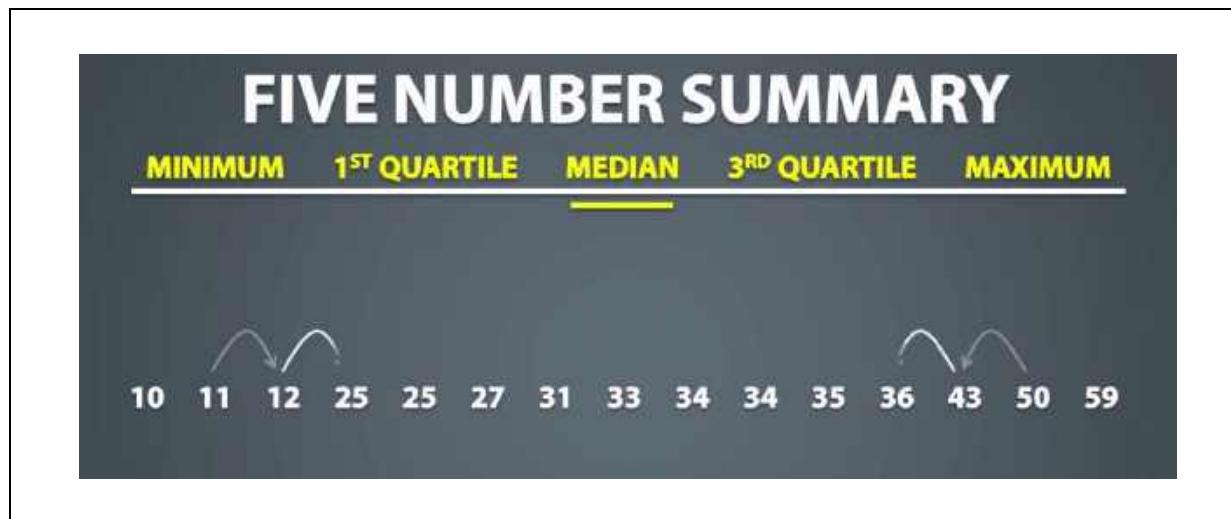
- ✓ The five number summary divides the data into equal quarters



- ✓ Let's take one example to determine the five number summary

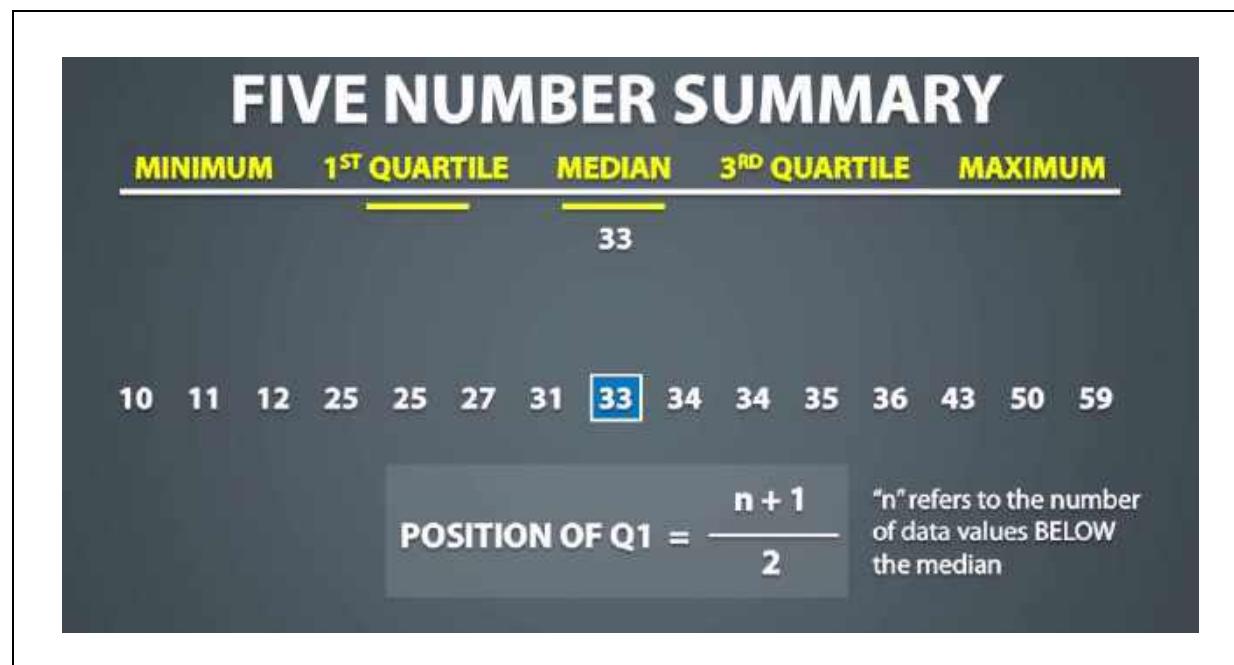


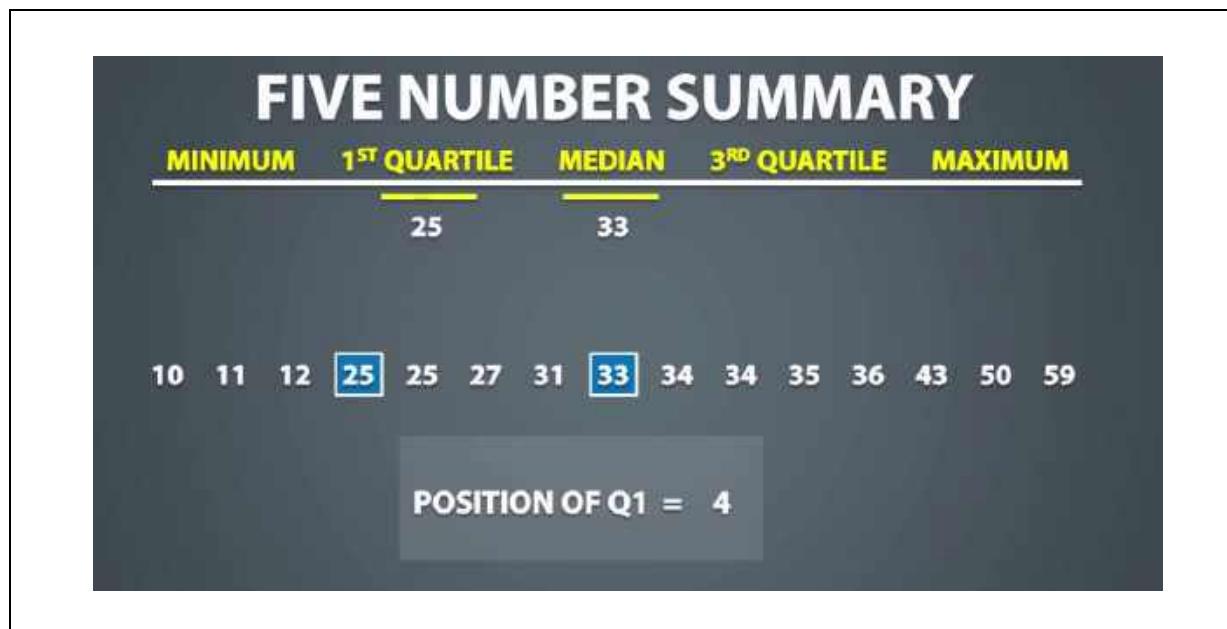
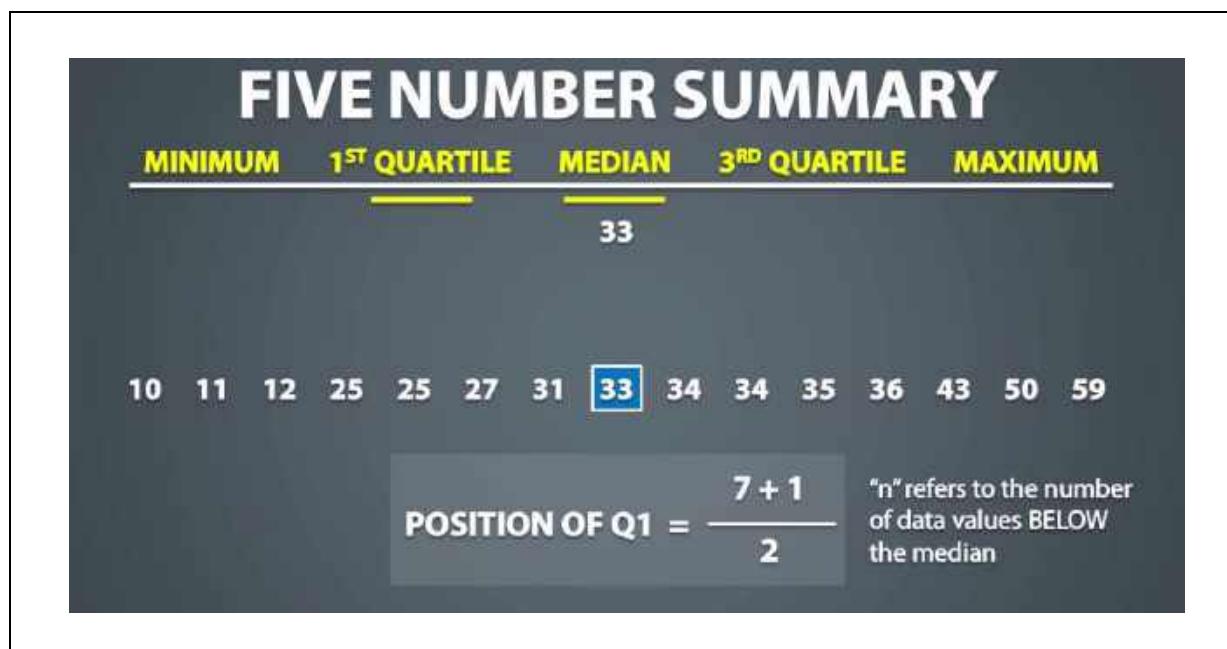
**Median:** Middle of the value



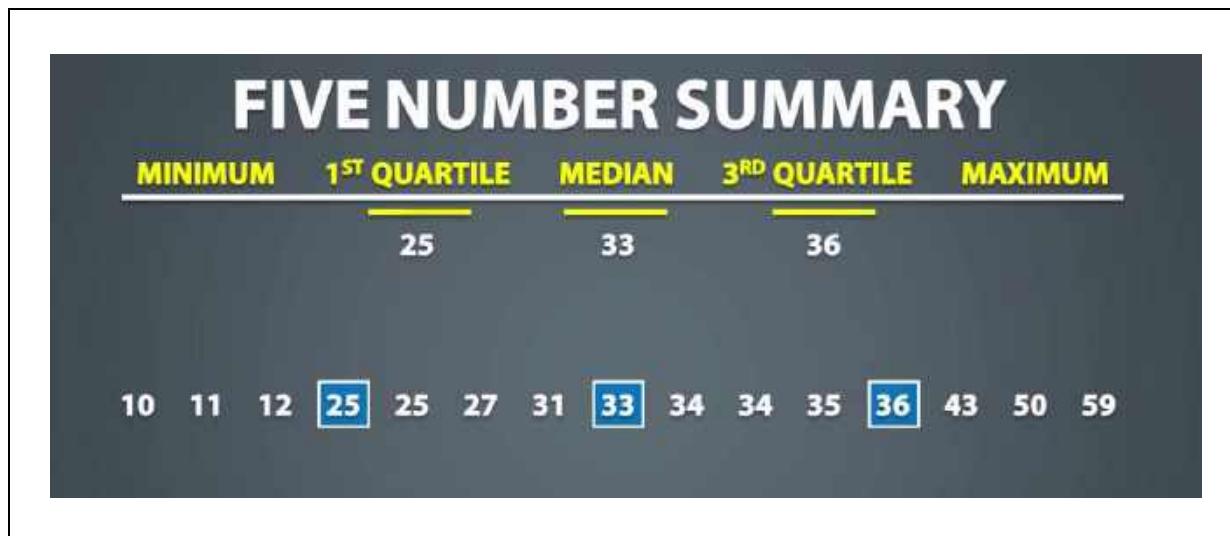
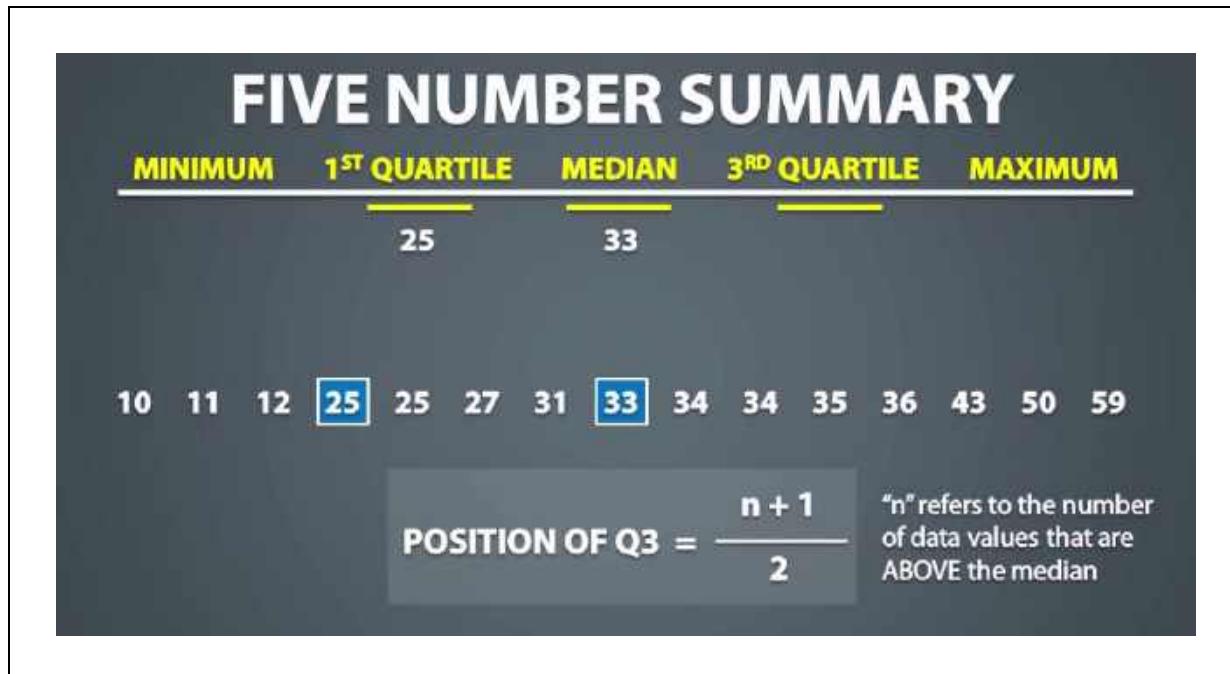


#### 4. 1st Quartile

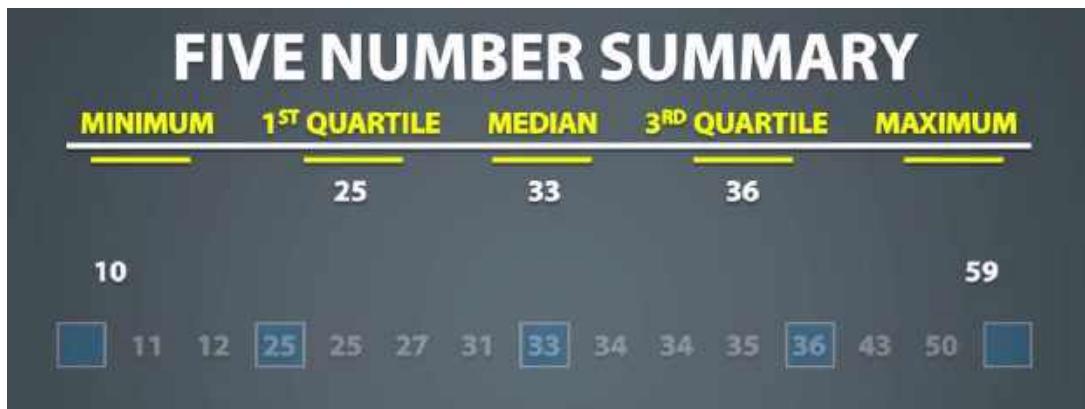




### 5. 3rd Quartile

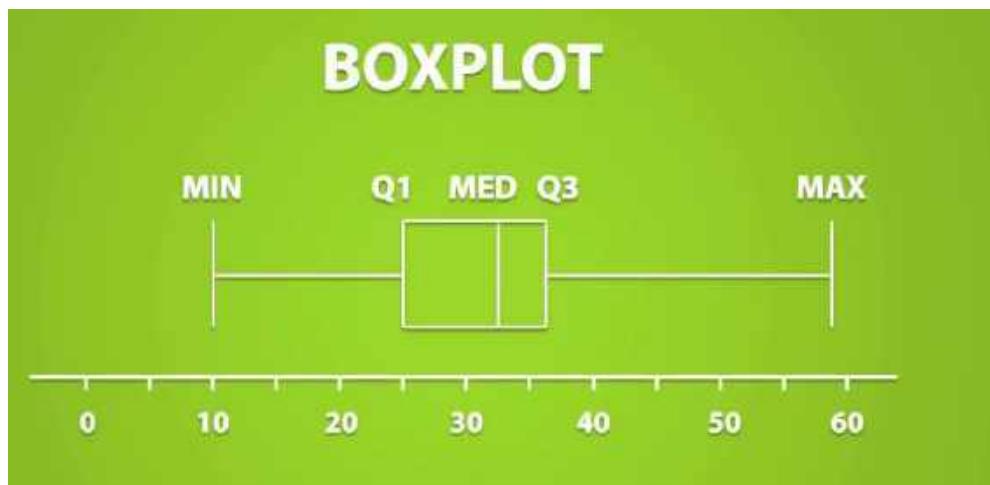


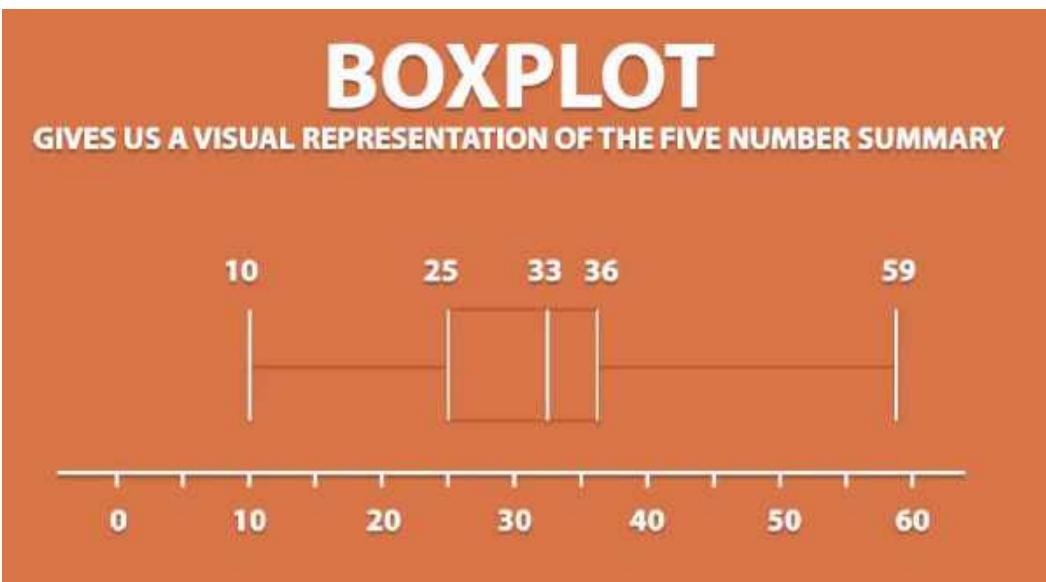
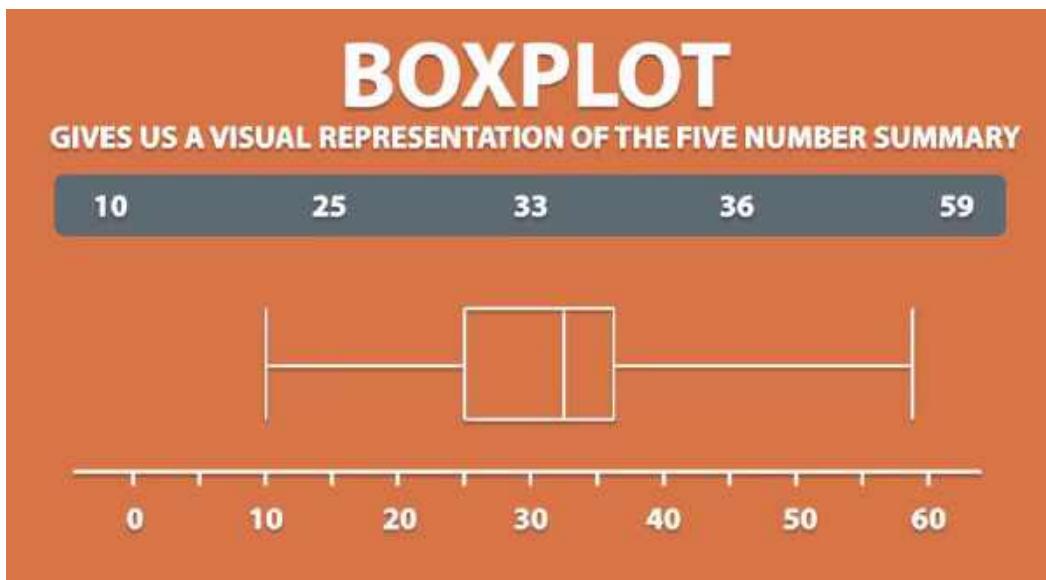
## 6. Smallest and largest



## 7. Box plot

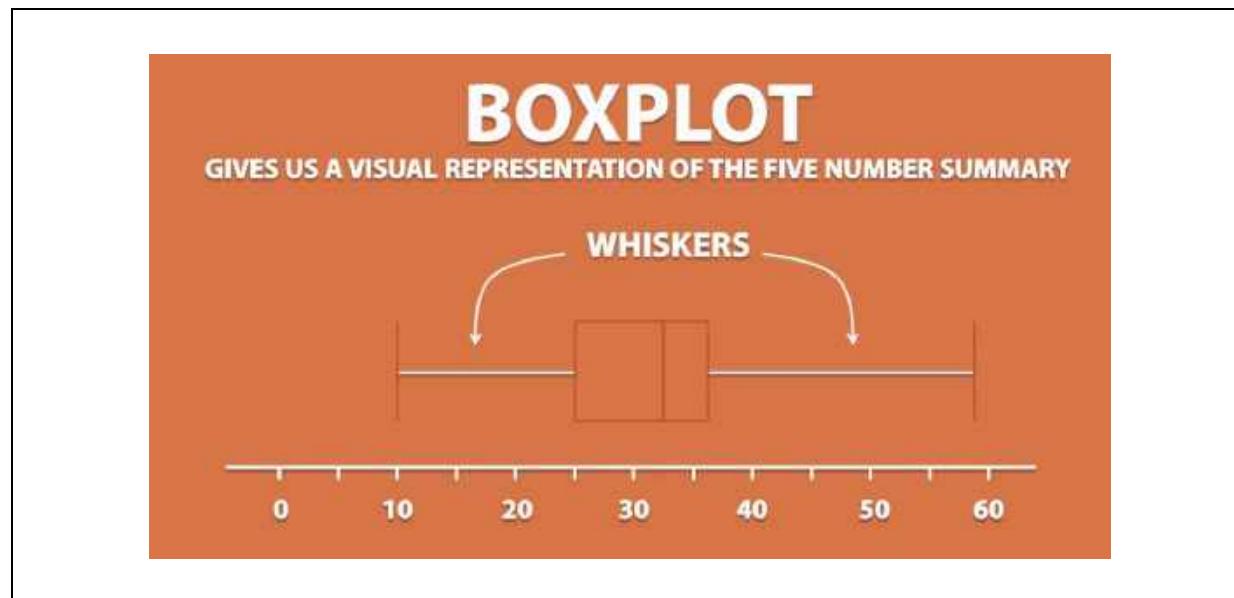
- ✓ We can take above five values summary and create a box plot
- ✓ A box plot gives us visual representation of the five numbers summary





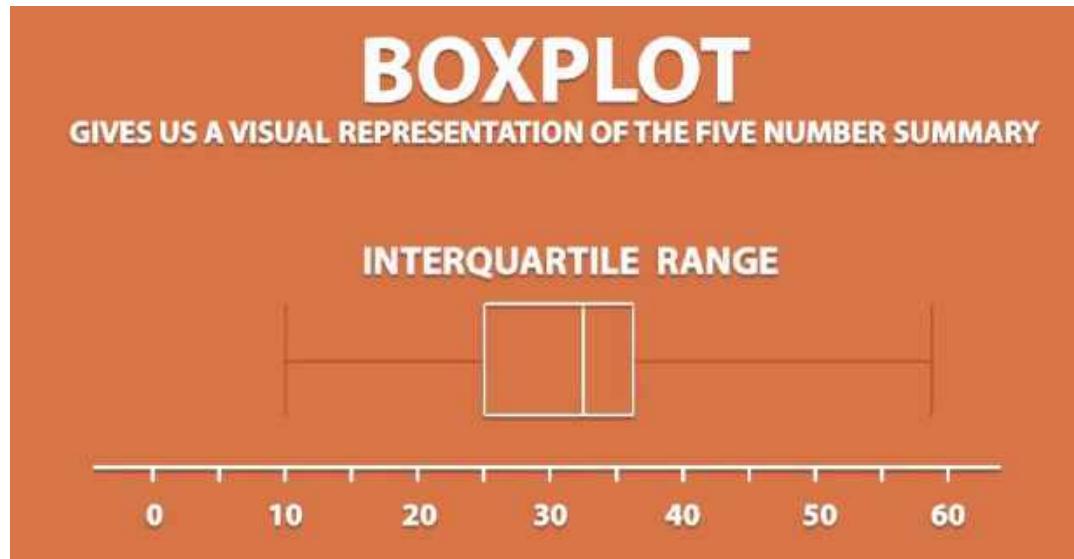
### 8. Whiskers

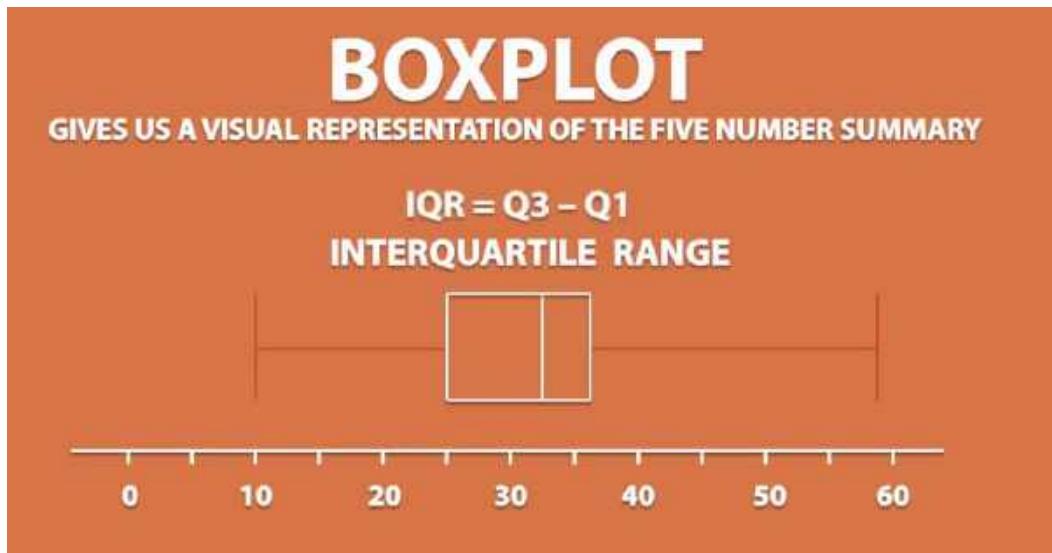
- ✓ The horizontal line that extends out from the box is called as whisker



## 9. Interquartile range

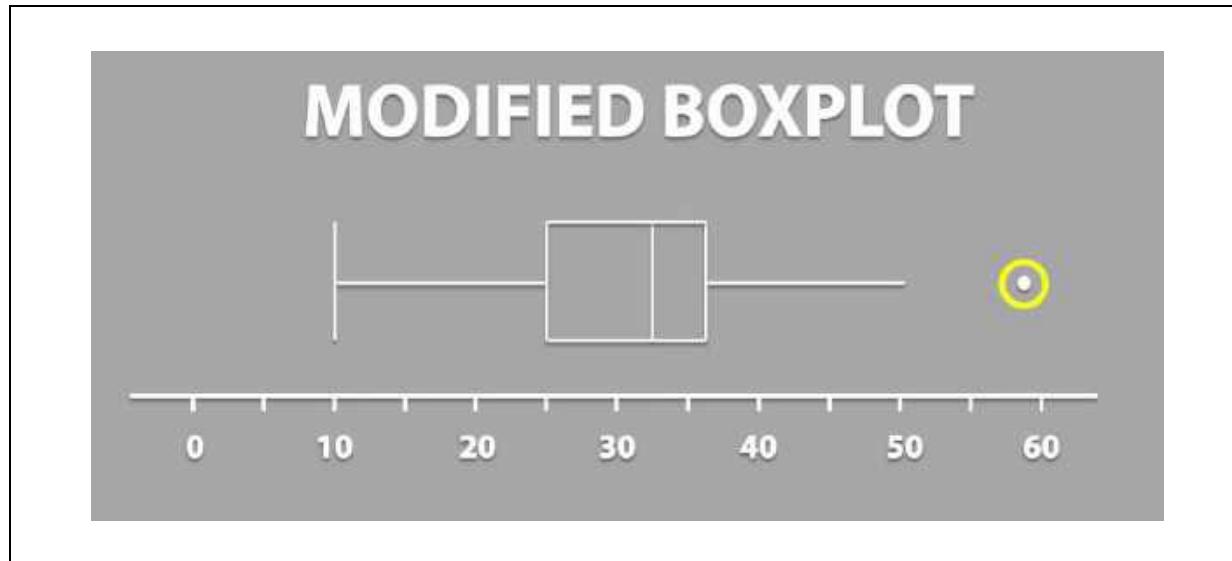
- ✓ The actual box is called as Interquartile range



**10. IQR = Q3 – Q1**

### 11. Outliers (modified box plot)

- ✓ A box plot with outlier is called as modified box plot



## 12. Outlier recognizing

A DATA VALUE IS CONSIDERED TO BE AN OUTLIER IF..

DATA VALUE



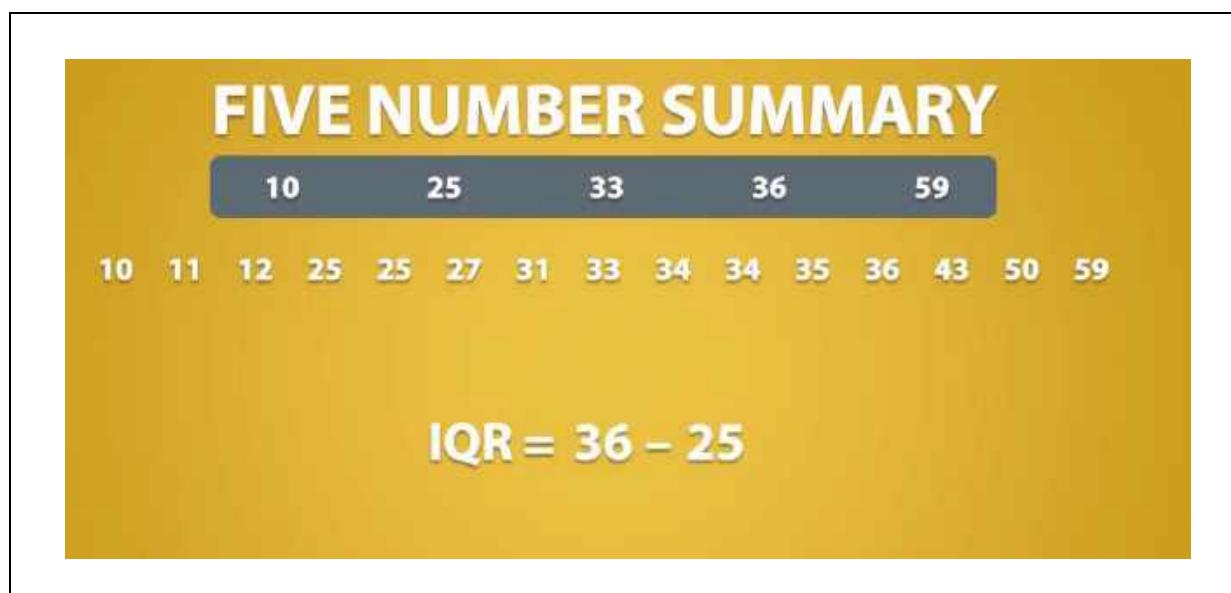
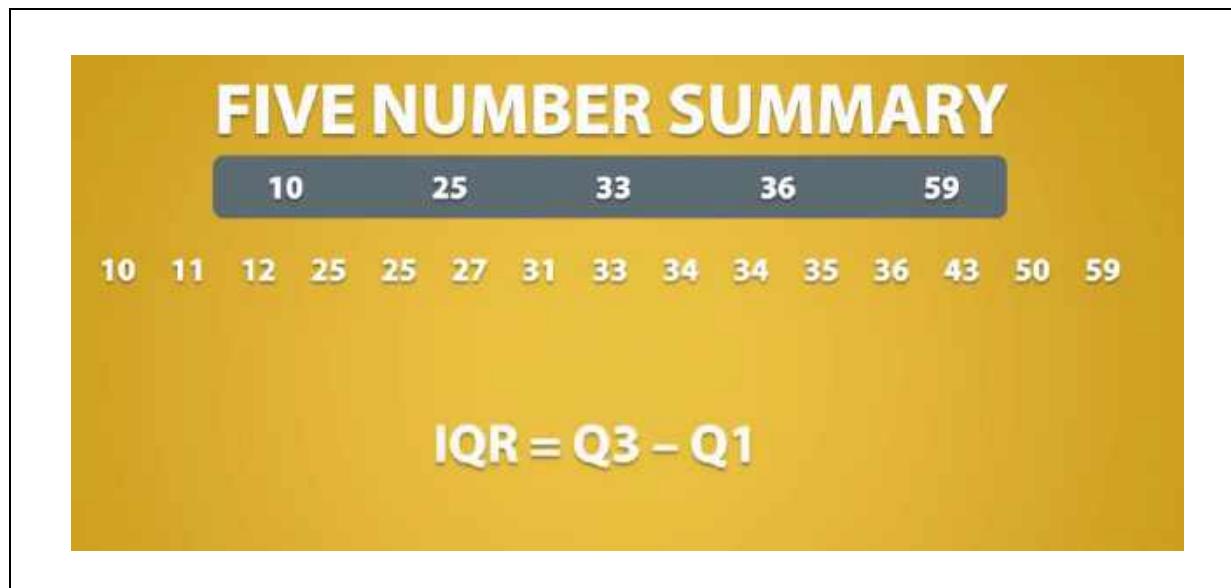
$Q1 - 1.5(IQR)$

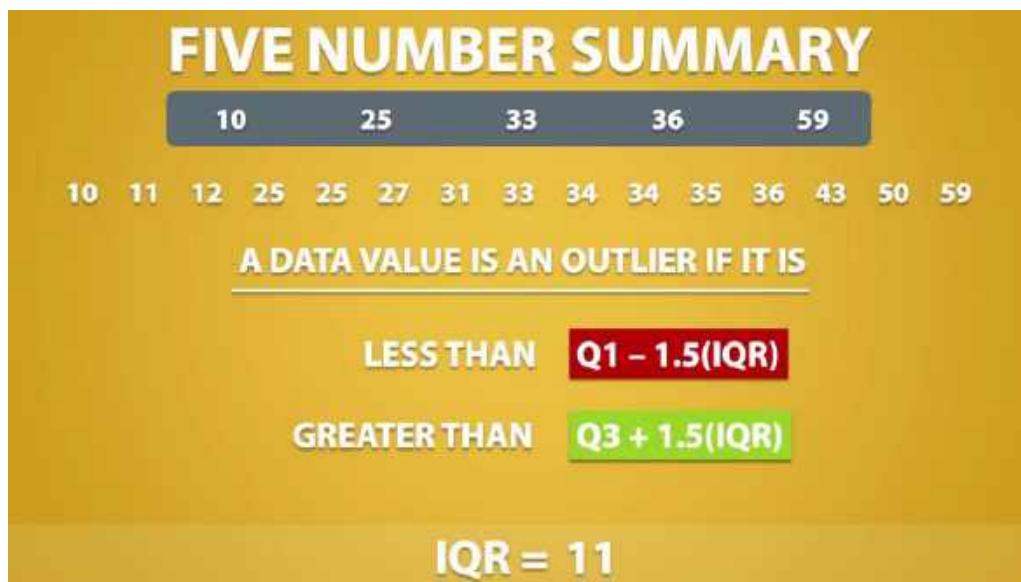
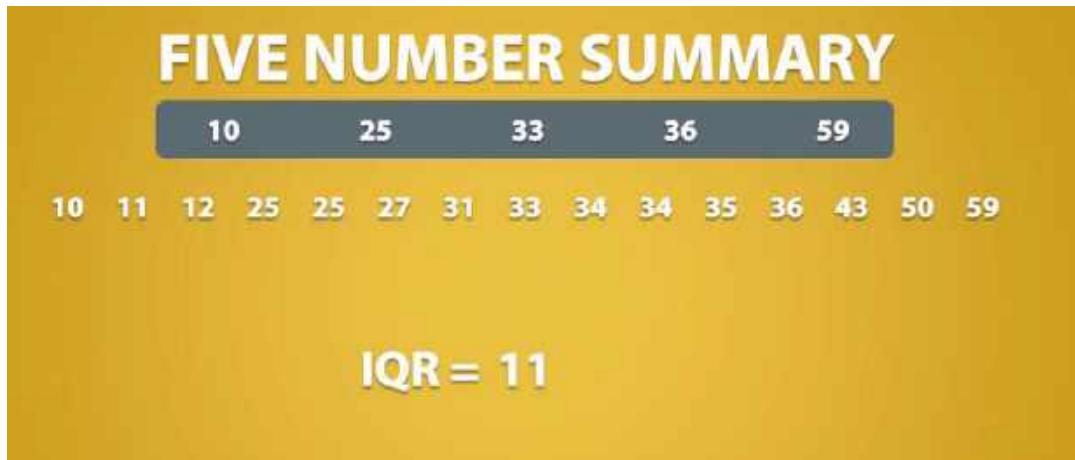
OR

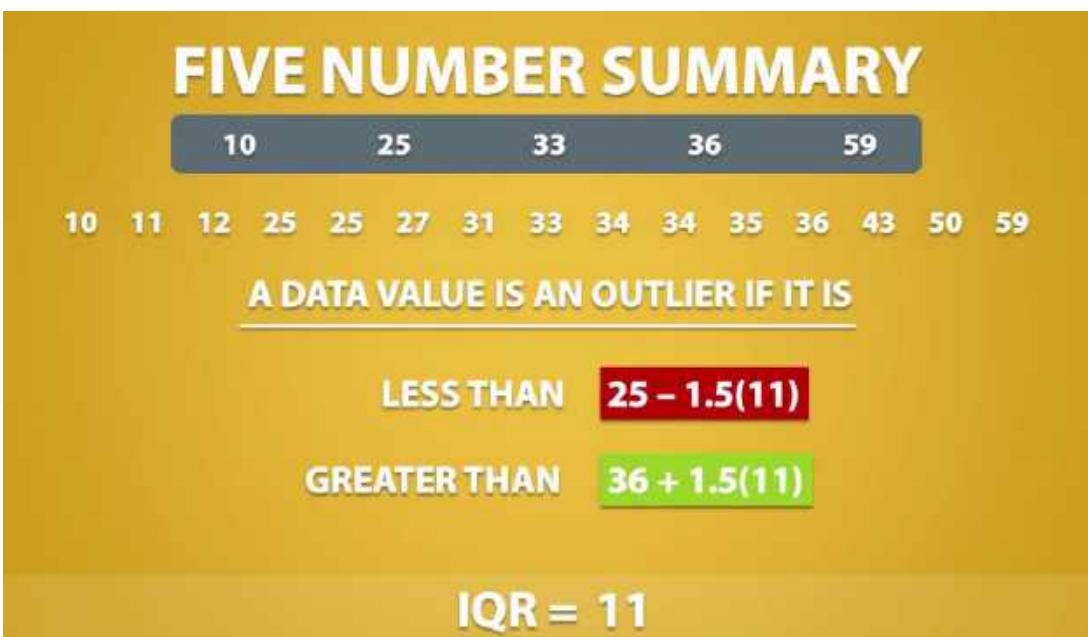
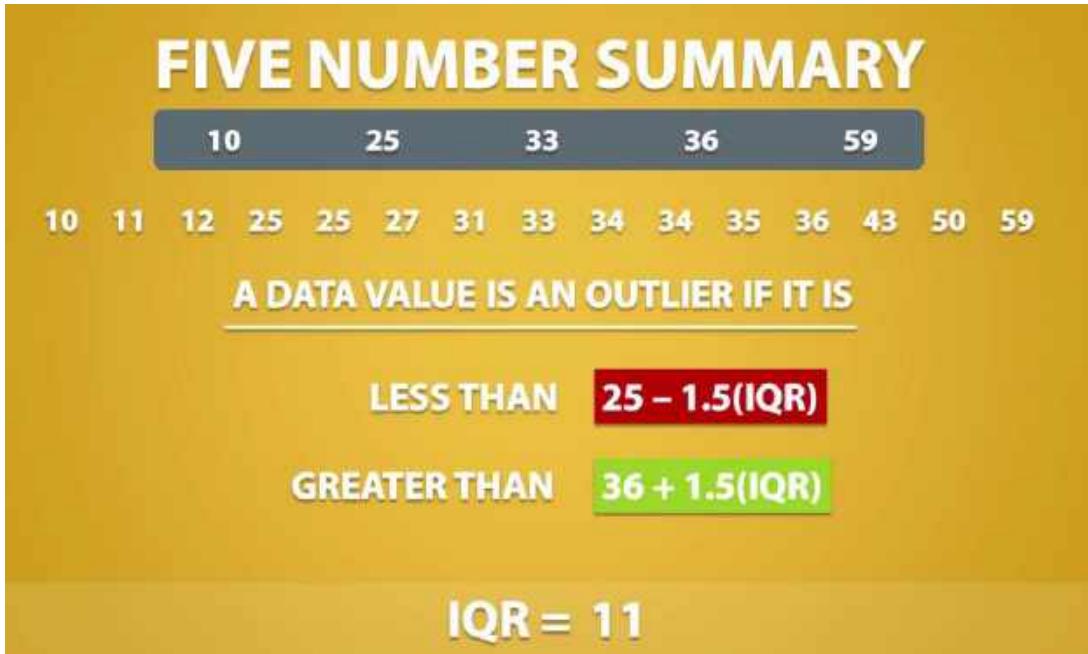
DATA VALUE

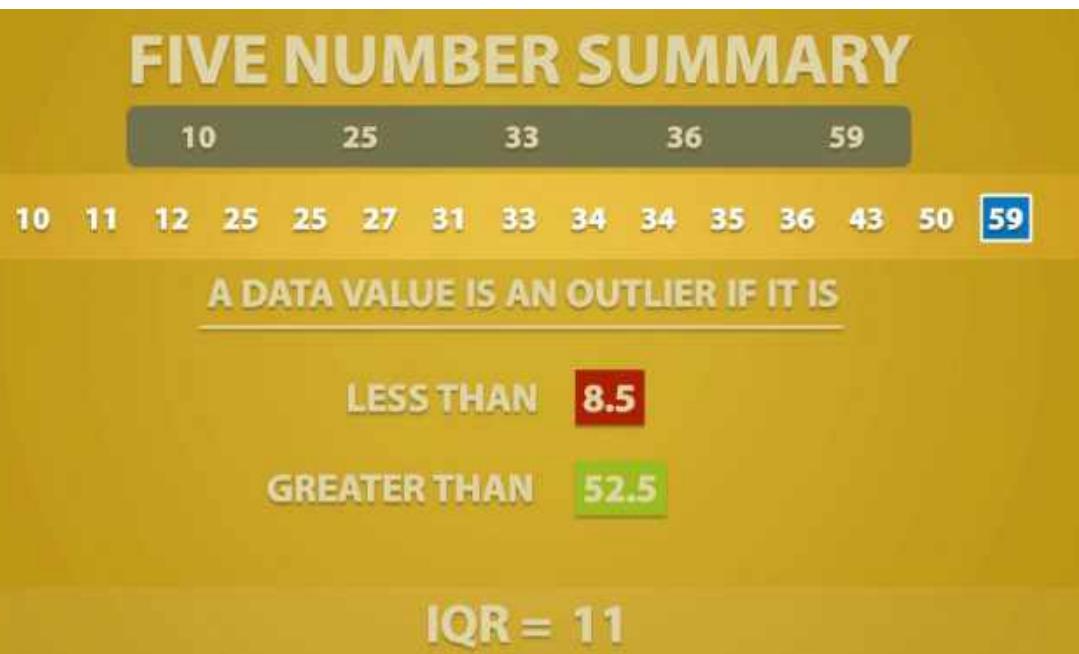
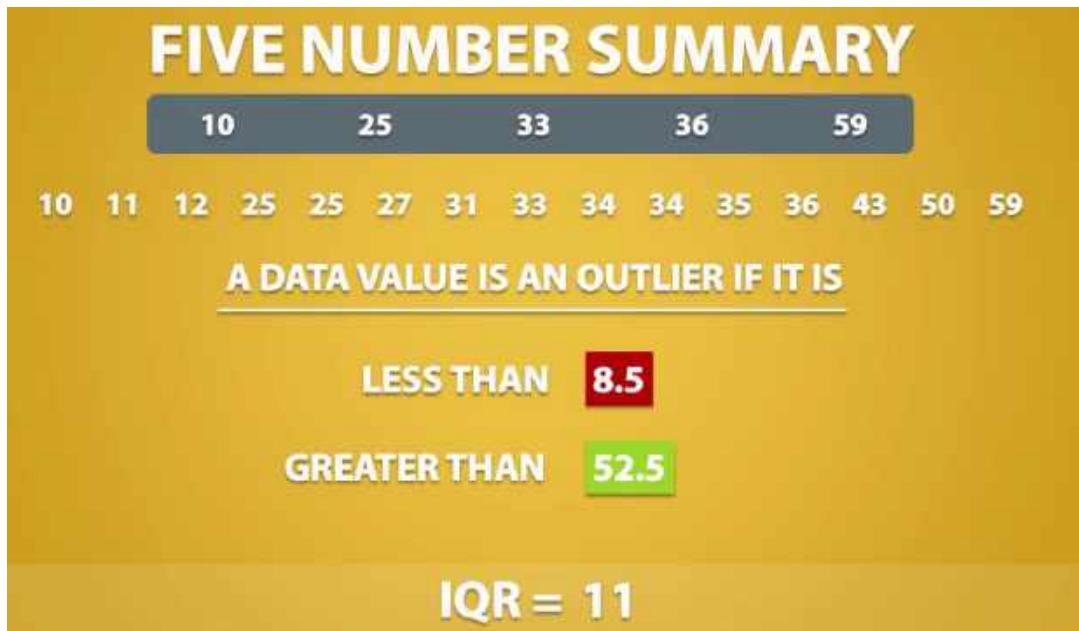


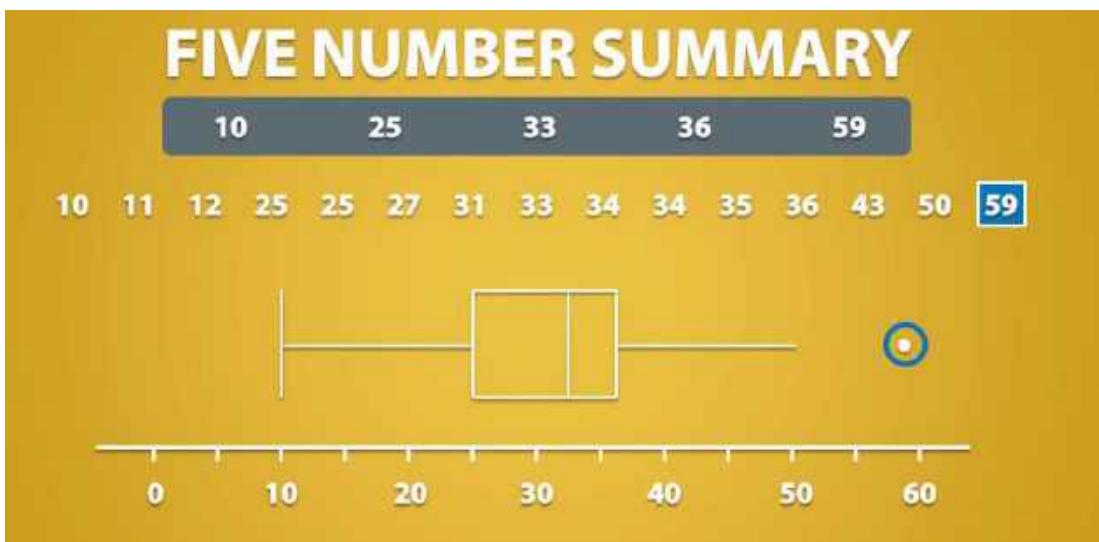
$Q3 + 1.5(IQR)$

**13. Checking outliers**









**5. Maths - Statistics – PART – 5****Contents**

<b>1. Symmetry and Skewness .....</b>	<b>2</b>
<b>2. Symmetric distribution .....</b>	<b>2</b>
<b>3. Distribution skewed or asymmetric distribution .....</b>	<b>3</b>
<b>4. Types of Skewness.....</b>	<b>4</b>
<b>5. Skewness to the LEFT .....</b>	<b>5</b>
<b>6. Skewness to the RIGHT .....</b>	<b>5</b>
<b>7. Boxplot.....</b>	<b>6</b>
<b>8. Boxplot skewness.....</b>	<b>6</b>
<b>9. A strategy to find skewness in boxplot .....</b>	<b>8</b>
9.1. Case 1:.....	8
9.2. Case 2:.....	9
9.3. Case 3:.....	10

## 5. Maths - Statistics – PART – 5

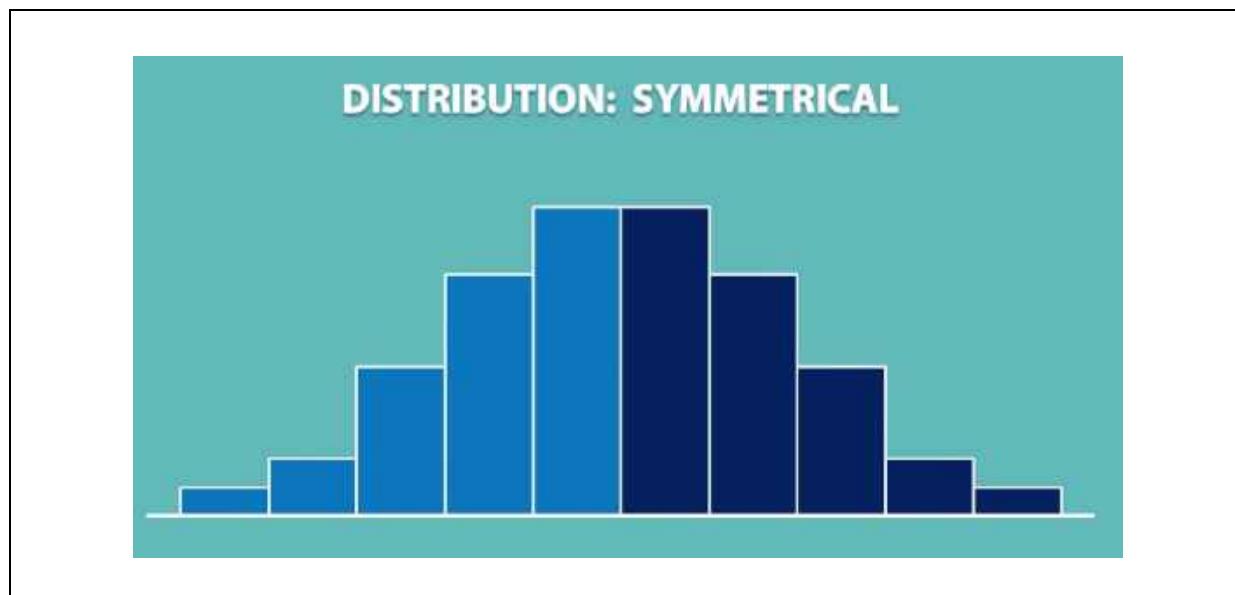
### 1. Symmetry and Skewness

- ✓ This concept explains about shape of a distribution



### 2. Symmetric distribution

- ✓ A distribution is called as symmetric, if it can be divided into two equal sizes of the same shape.
- ✓ Below histogram explains about symmetric distribution



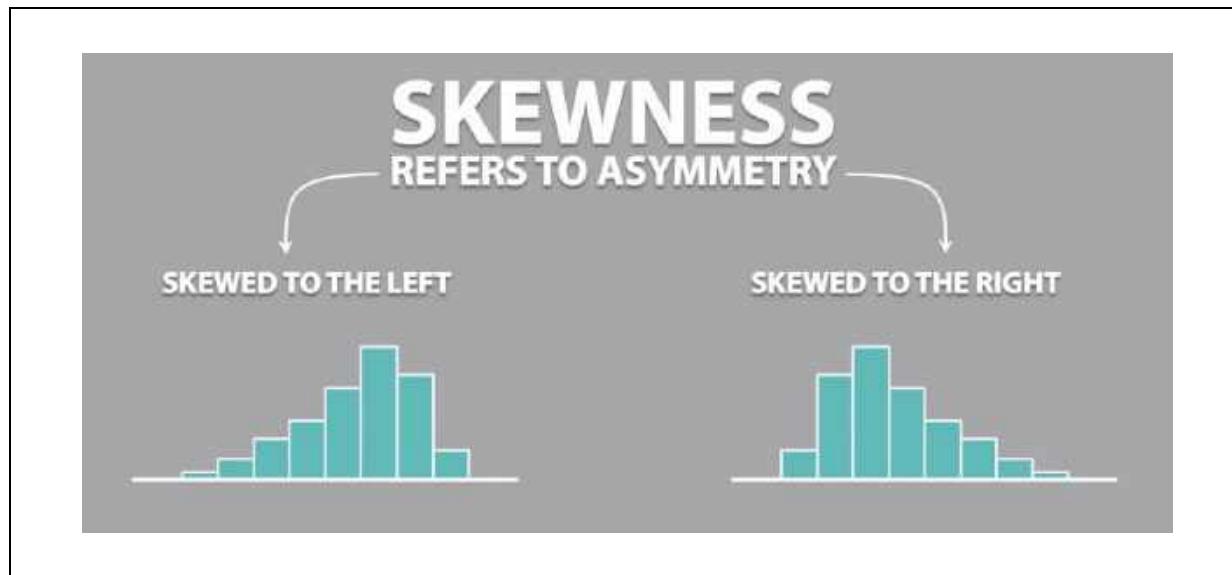
### 3. Distribution skewed or asymmetric distribution

- ✓ A distribution is called as skewed, if it cannot be divided into equal sizes.
- ✓ It's also called as asymmetric distribution
- ✓ Skewness refers to the asymmetry



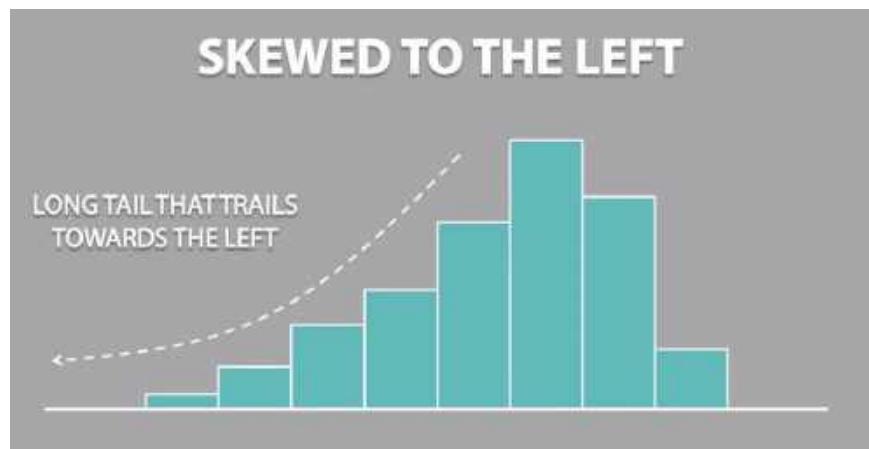
#### 4. Types of Skewness

- ✓ We understand the skewness based on direction which the data points clustered
- ✓ There are two types
  - Skewness to the LEFT
  - Skewness to the RIGHT



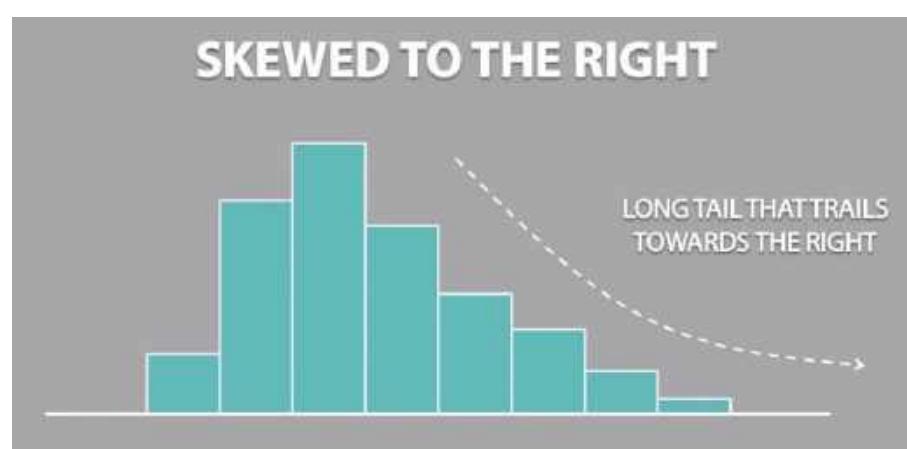
### 5. Skewness to the LEFT

- ✓ If the data is clustered at left hand side then it is called as skewness to the LEFT



### 6. Skewness to the RIGHT

- ✓ If the data is clustered at right hand side then it is called as skewness to the RIGHT

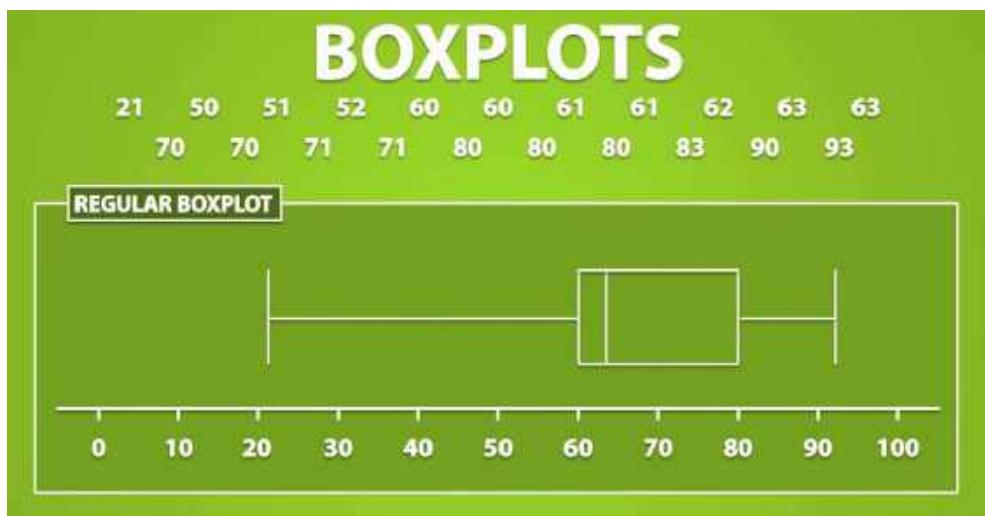


## 7. Boxplot

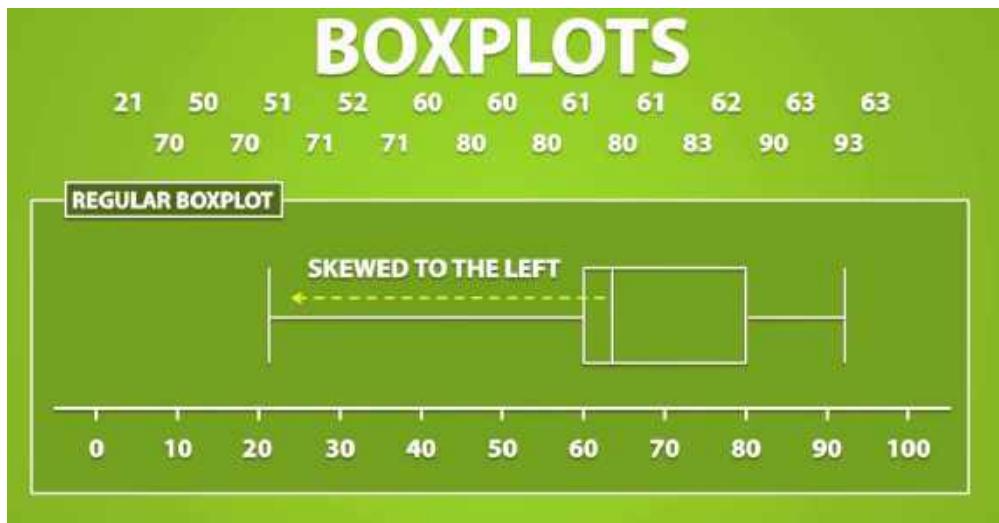
- ✓ We can determine skewness into the box plot
- ✓ The presences of outliers may effect to determine skewness

## 8. Boxplot skewness

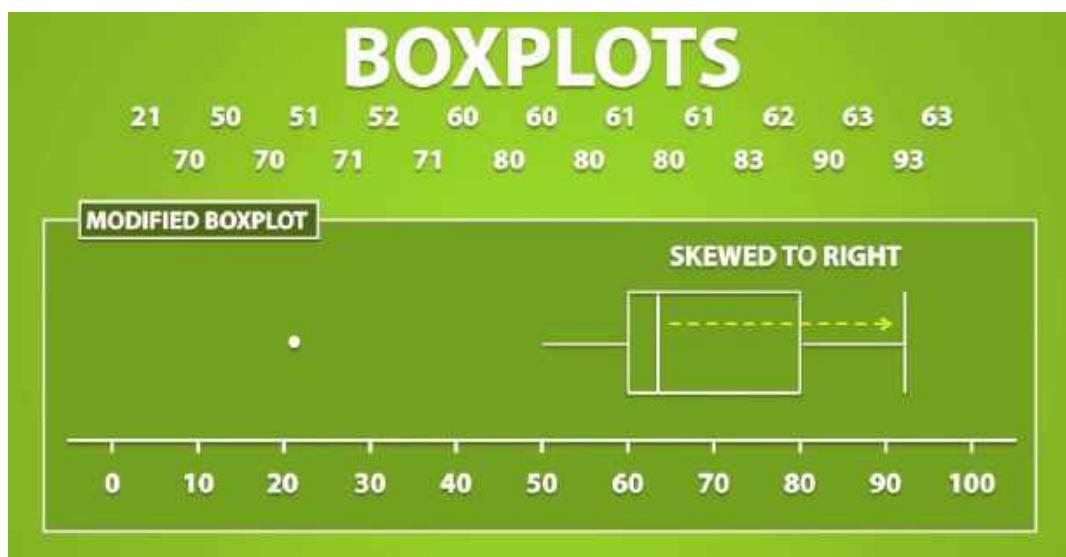
- ✓ When we construct the boxplot for the below data then we can draw below one



- ✓ According the boxplot we may think that this distribution is skewed to the left.



- ✓ But when we converted into modified box plot(because of outlier) then its directing to right side

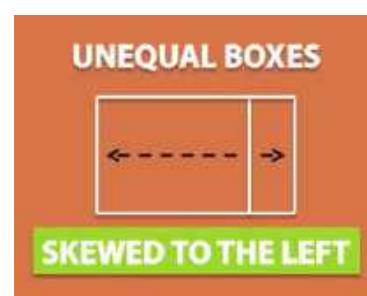


## 9. A strategy to find skewness in boxplot

- ✓ If we have unequal boxes, the side of the box is larger than that determines the skew

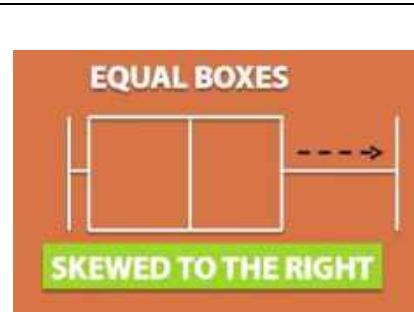
### 9.1. Case 1:

- ✓ If left side of the box larger than the right side, so its skewed to the left



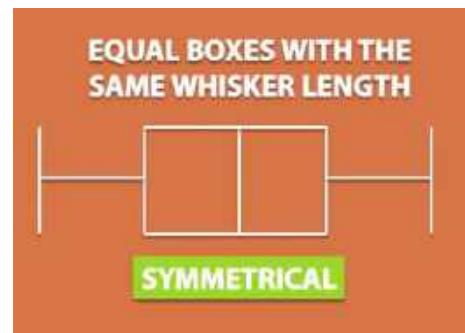
**9.2. Case 2:**

- ✓ If the boxes are equal in size then we need to consider the whisker size to determine this skew
- ✓ The larger whisker determines the skew
- ✓ In below case it's in right

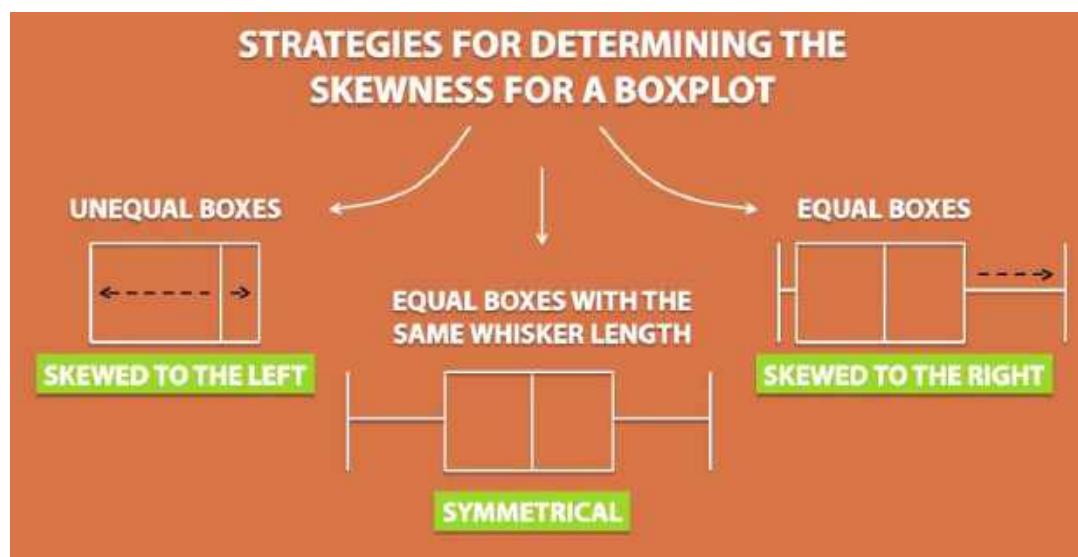


### 9.3. Case 3:

- ✓ If the boxes are equal with same whisker length then the distribution is said to be symmetrical



### A summary



**6. Maths - Statistics – PART – 6****Contents**

<b>1. Explanatory variable .....</b>	3
<b>2. Response variable .....</b>	3
<b>3. Scatter plot .....</b>	4
<b>4. Example .....</b>	4
<b>5. Correlation.....</b>	6
<b>6. Correlation – Direction &amp; Strength.....</b>	7
6.1. $r = 0$ .....	8
6.2. If $r$ value is towards to +1 or -1 .....	9
<b>7. Calculate correlation .....</b>	10

**6. Maths - Statistics – PART – 6**

- ✓ This concept explains about how two variables are related each other

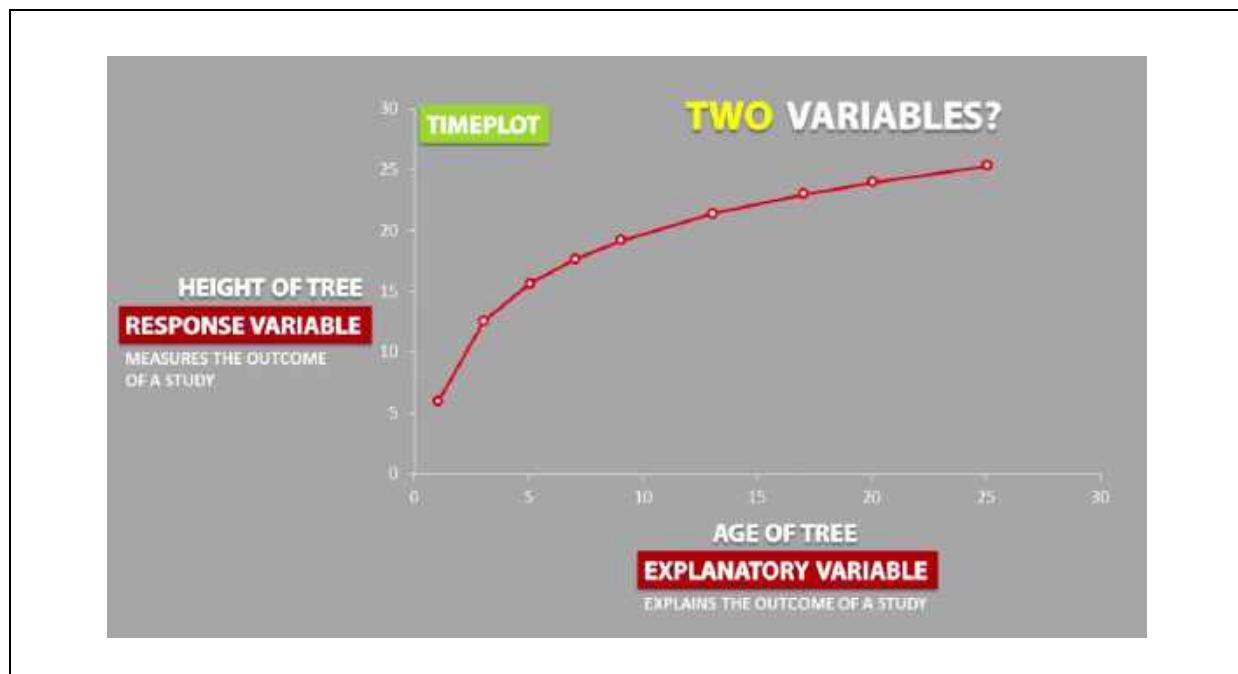


## 1. Explanatory variable

- ✓ This variables explains the outcome of the study
- ✓ Example is Age:
  - As we are reaching older then the taller will be increase till to certain level
  - **Age** is explains about **height**
- ✓ It is also called as Independent variable

## 2. Response variable

- ✓ This variables measures the outcome of the study
- ✓ It is also called as Dependent variable

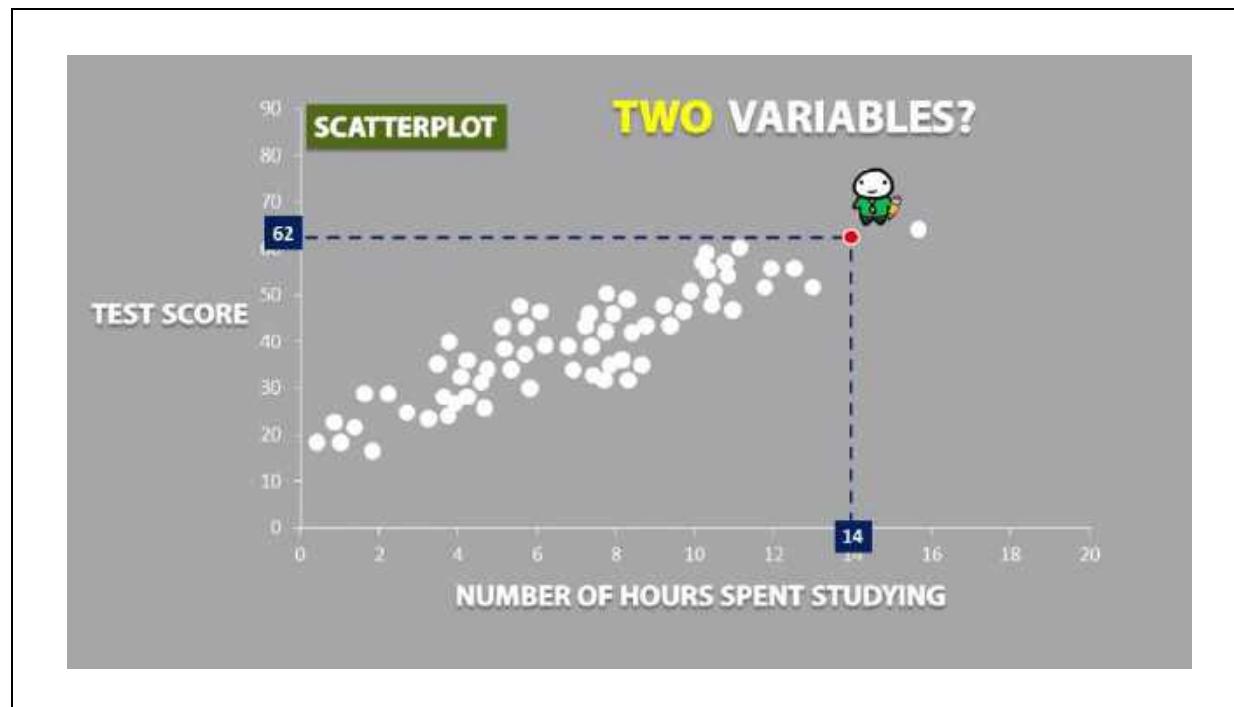


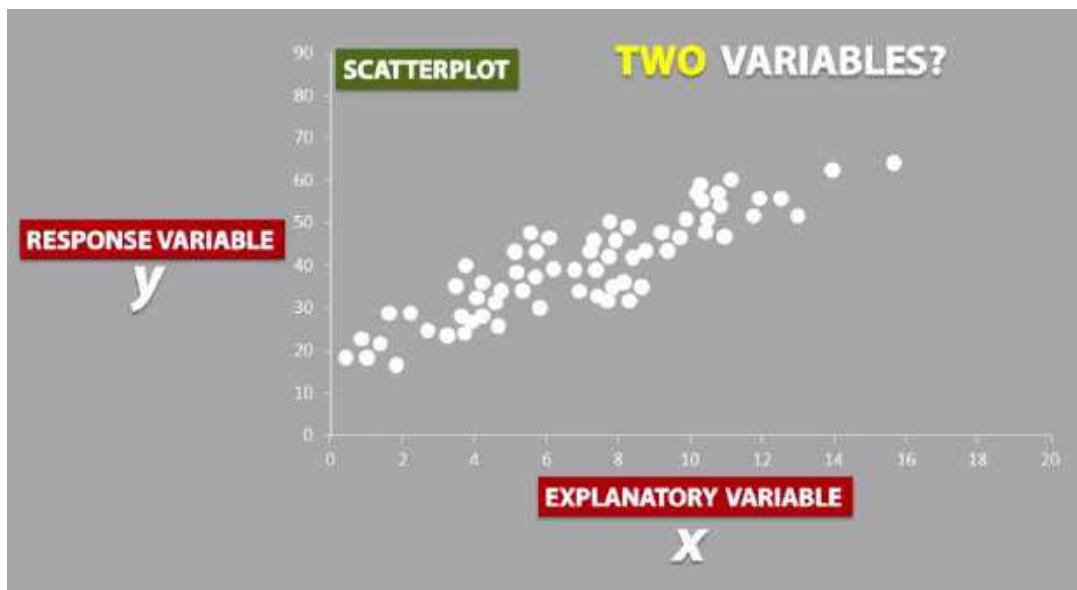
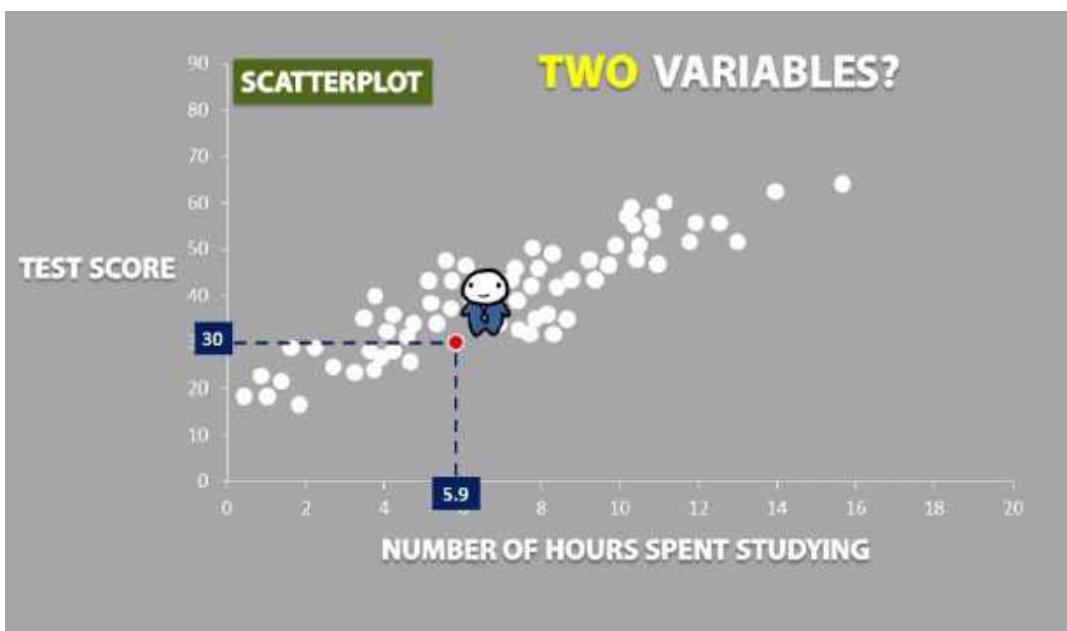
### 3. Scatter plot

- ✓ Scatter plot is the good example which explains about one variable growth/down based on other variable

### 4. Example

- ✓ Scatter plot is the good example which explains about one variable growth/down based on other variable



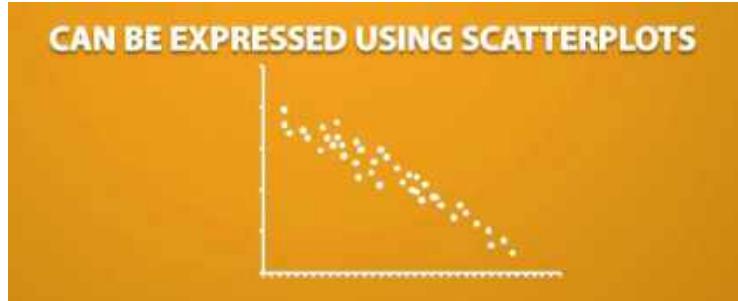


## 5. Correlation

- ✓ It explains about the **direction** and **strength** of the linear relationship shared between two quantitative variables
- ✓ It is denoted as **r**

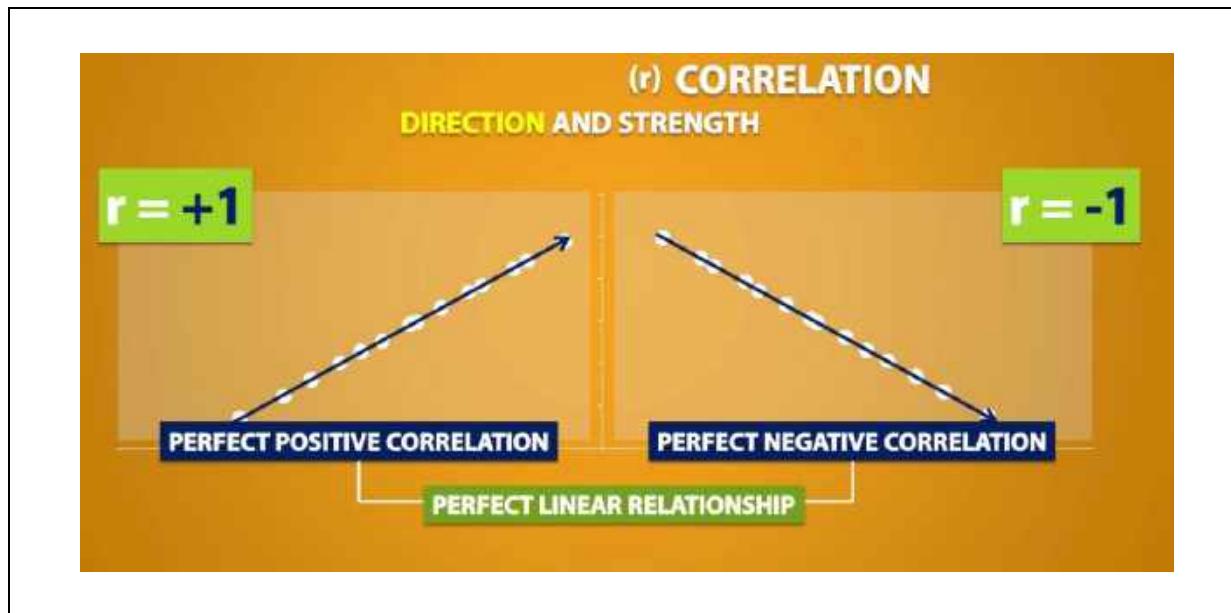
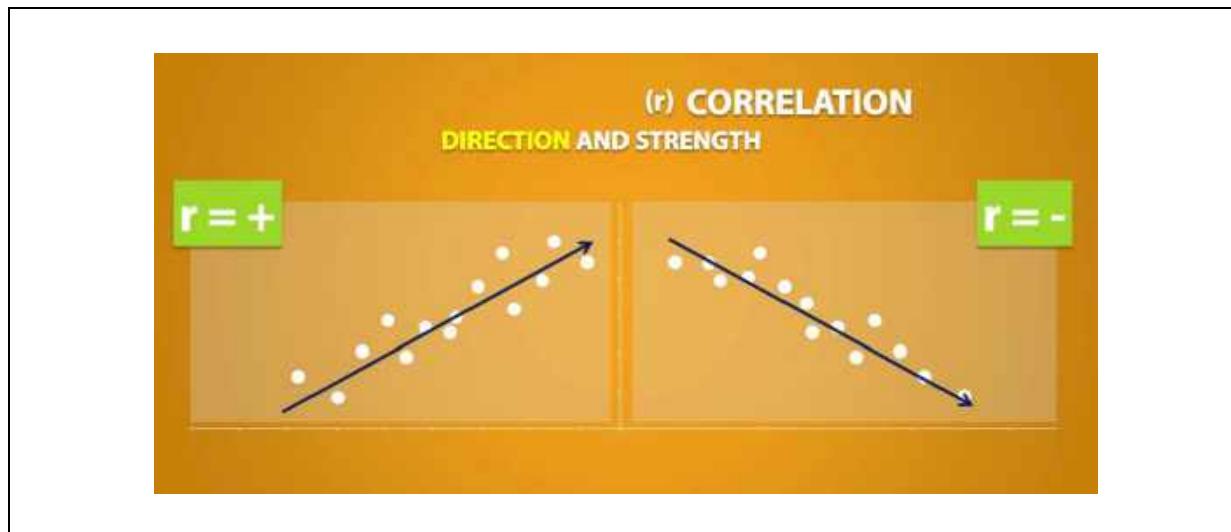


- ✓ Correlation can be expressed using scatter plots



## 6. Correlation – Direction & Strength

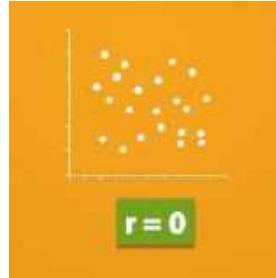
- ✓ Correlation speaks about the direction or slope of set of data points
- ✓ It explains about the direction can be upwards or downwards
  - If upwards then correlation is positive
  - If downwards then correlation is negative



- ✓ Correlation measures the strength of the linear relationship
- ✓ So, correlation values can be between **+1** and negative **-1**
- ✓ The strength of the linear relationship increased as  $r$  got close to positive **+1** or negative **-1**

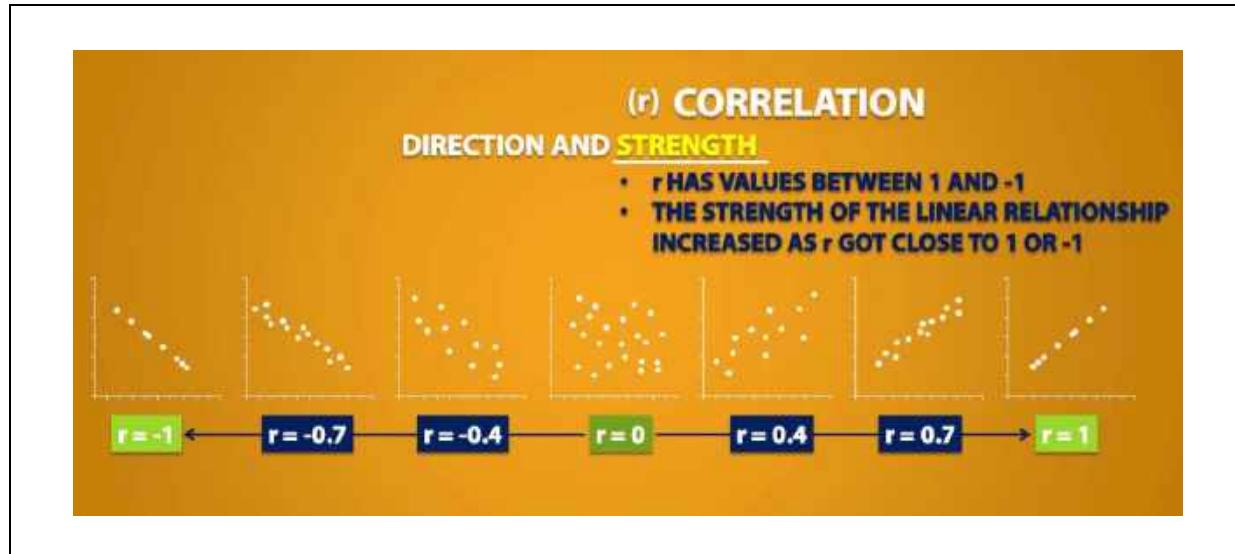
**6.1.  $r = 0$** 

- ✓ There is no correlation
- ✓ There is no linear relationship in between



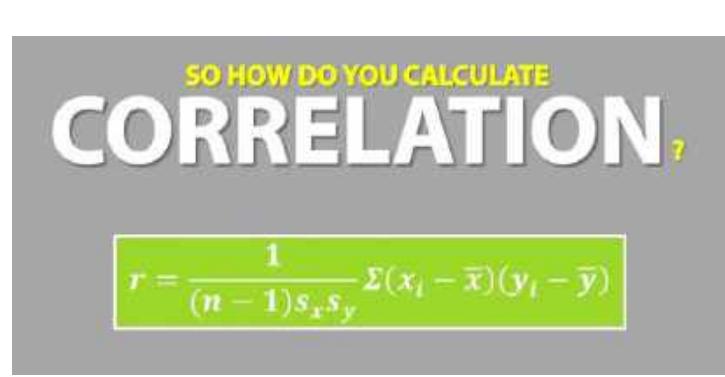
### 6.2. If r value is towards to +1 or -1

- ✓ In this case the linear relationship get stronger
- ✓ If r value gets close to +1 or -1 then the relationship is very stronger



## 7. Calculate correlation

- ✓ We can calculate correlation by using formula



## Data Science – Maths – Part - 6

---

### Example



**A TEACHER WANTS TO DETERMINE THE CORRELATION BETWEEN THE NUMBER OF HOURS SPENT STUDYING AND TEST SCORES.**

STUDENT NAME	NUMBER OF HOURS SPENT STUDYING	TEST SCORE ( out of 100 )
JOHN	13	53
ALLIE	15	69
MARK	7	92
SAMANTHA	3	10
JESSICA	10	85
JOSEPH	27	99



**A TEACHER WANTS TO DETERMINE THE CORRELATION BETWEEN THE NUMBER OF HOURS SPENT STUDYING AND TEST SCORES.**

STUDENT NAME	$x_i$	$y_i$
JOHN	13	53
ALLIE	15	69
MARK	7	92
SAMANTHA	3	10
JESSICA	10	85
JOSEPH	27	99

## Data Science – Maths – Part - 6

---

$$r = \frac{1}{(n-1)s_x s_y} \sum (x_i - \bar{x})(y_i - \bar{y})$$

$x_i$	$y_i$	$(x_i - \bar{x})$	$(y_i - \bar{y})$	$(x_i - \bar{x})(y_i - \bar{y})$
13	53			
15	69			
7	92			
3	10			
10	85			
27	99			

$$r = \frac{1}{(n-1)s_x s_y} \sum (x_i - \bar{x})(y_i - \bar{y})$$

$x_i$	$y_i$	$(x_i - \bar{x})$	$(y_i - \bar{y})$	$(x_i - \bar{x})(y_i - \bar{y})$
13	53			
15	69			
7	92			
3	10			
10	85			
27	99			

$\bar{x} = 12.5$

$\bar{y} = 68$

## Data Science – Maths – Part - 6

---


$$r = \frac{1}{(n-1)s_x s_y} \sum (x_i - \bar{x})(y_i - \bar{y})$$

$x_i$	$y_i$	$(x_i - \bar{x})$	$(y_i - \bar{y})$	$(x_i - \bar{x})(y_i - \bar{y})$
13	53	13 – 12.5		
15	69			
7	92			
3	10			
10	85			
27	99			
$\bar{x} = 12.5$		$\bar{y} = 68$		

$$r = \frac{1}{(n-1)s_x s_y} \sum (x_i - \bar{x})(y_i - \bar{y})$$

$x_i$	$y_i$	$(x_i - \bar{x})$	$(y_i - \bar{y})$	$(x_i - \bar{x})(y_i - \bar{y})$
13	53	0.5		
15	69	15 – 12.5		
7	92			
3	10			
10	85			
27	99			
$\bar{x} = 12.5$		$\bar{y} = 68$		

## Data Science – Maths – Part - 6

---

$$r = \frac{1}{(n-1)s_x s_y} \sum (x_i - \bar{x})(y_i - \bar{y})$$

$x_i$	$y_i$	$(x_i - \bar{x})$	$(y_i - \bar{y})$	$(x_i - \bar{x})(y_i - \bar{y})$
13	53	0.5		
15	69	2.5		
7	92	-5.5		
3	10	-9.5		
10	85	-2.5		
27	99	14.5		
$\bar{x} = 12.5$		$\bar{y} = 68$		

$$r = \frac{1}{(n-1)s_x s_y} \sum (x_i - \bar{x})(y_i - \bar{y})$$

$x_i$	$y_i$	$(x_i - \bar{x})$	$(y_i - \bar{y})$	$(x_i - \bar{x})(y_i - \bar{y})$
13	53	0.5	53 - 68	
15	69	2.5		
7	92	-5.5		
3	10	-9.5		
10	85	-2.5		
27	99	14.5		
$\bar{x} = 12.5$		$\bar{y} = 68$		

## Data Science – Maths – Part - 6

---


$$r = \frac{1}{(n-1)s_x s_y} \sum (x_i - \bar{x})(y_i - \bar{y})$$

$x_i$	$y_i$	$(x_i - \bar{x})$	$(y_i - \bar{y})$	$(x_i - \bar{x})(y_i - \bar{y})$
13	53	0.5	-15	
15	69	2.5	1	
7	92	-5.5	24	
3	10	-9.5	-58	
10	85	-2.5	17	
27	99	14.5	31	
$\bar{x} = 12.5$		$\bar{y} = 68$		

$$r = \frac{1}{(n-1)s_x s_y} \sum (x_i - \bar{x})(y_i - \bar{y})$$

$x_i$	$y_i$	$(x_i - \bar{x})$	$(y_i - \bar{y})$	$(x_i - \bar{x})(y_i - \bar{y})$
13	53	0.5	-15	( 0.5 )( -15 )
15	69	2.5	1	
7	92	-5.5	24	
3	10	-9.5	-58	
10	85	-2.5	17	
27	99	14.5	31	
$\bar{x} = 12.5$		$\bar{y} = 68$		

## Data Science – Maths – Part - 6

---

$$r = \frac{1}{(n-1)s_x s_y} \sum (x_i - \bar{x})(y_i - \bar{y})$$

$x_i$	$y_i$	$(x_i - \bar{x})$	$(y_i - \bar{y})$	$(x_i - \bar{x})(y_i - \bar{y})$
13	53	0.5	-15	( 0.5 )( -15 )
15	69	2.5	1	( 2.5 )( 1 )
7	92	-5.5	24	
3	10	-9.5	-58	
10	85	-2.5	17	
27	99	14.5	31	
$\bar{x} = 12.5$		$\bar{y} = 68$		

$$r = \frac{1}{(n-1)s_x s_y} \sum (x_i - \bar{x})(y_i - \bar{y})$$

$x_i$	$y_i$	$(x_i - \bar{x})$	$(y_i - \bar{y})$	$(x_i - \bar{x})(y_i - \bar{y})$
13	53	0.5	-15	-7.5
15	69	2.5	1	2.5
7	92	-5.5	24	-132
3	10	-9.5	-58	551
10	85	-2.5	17	-42.5
27	99	14.5	31	449.5
$\bar{x} = 12.5$		$\bar{y} = 68$		

## Data Science – Maths – Part - 6

---

$$r = \frac{1}{(n-1)s_x s_y} \sum (x_i - \bar{x})(y_i - \bar{y})$$

$x_i$	$y_i$	$(x_i - \bar{x})$	$(y_i - \bar{y})$	$(x_i - \bar{x})(y_i - \bar{y})$
13	53	0.5	-15	-7.5
15	69	2.5	1	2.5
7	92	-5.5	24	-132
3	10	-9.5	-58	551
10	85	-2.5	17	-42.5
27	99	14.5	31	449.5
$\bar{x} = 12.5$		$\bar{y} = 68$		<b>SUM = 821</b>

$$r = \frac{1}{(n-1)s_x s_y} \left[ \boxed{821} \right]$$

$x_i$	$y_i$	$(x_i - \bar{x})$	$(y_i - \bar{y})$	$(x_i - \bar{x})(y_i - \bar{y})$
13	53	0.5	-15	-7.5
15	69	2.5	1	2.5
7	92	-5.5	24	-132
3	10	-9.5	-58	551
10	85	-2.5	17	-42.5
27	99	14.5	31	449.5
$\bar{x} = 12.5$		$\bar{y} = 68$		<b>SUM = 821</b>

## Data Science – Maths – Part - 6

---


$$r = \frac{1}{(6-1)s_x s_y} \left[ \begin{array}{c} 821 \end{array} \right]$$

$x_i$	$y_i$	$(x_i - \bar{x})$	$(y_i - \bar{y})$	$(x_i - \bar{x})(y_i - \bar{y})$
13	53	0.5	-15	-7.5
15	69	2.5	1	2.5
7	92	-5.5	24	-132
3	10	-9.5	-58	551
10	85	-2.5	17	-42.5
27	99	14.5	31	449.5
$\bar{x} = 12.5$		$\bar{y} = 68$		<b>SUM = 821</b>

$$r = \frac{1}{(6-1)s_x s_y} \left[ \begin{array}{c} 821 \end{array} \right]$$

$x_i$	$y_i$	$(x_i - \bar{x})$	$(y_i - \bar{y})$	$(x_i - \bar{x})(y_i - \bar{y})$
13	53	0.5	-15	-7.5
15	69	2.5	1	2.5
7	92	-5.5	24	-132
3	10	-9.5	-58	551
10	85	-2.5	17	-42.5
27	99	14.5	31	449.5
$\bar{x} = 12.5$		$\bar{y} = 68$		<b>SUM = 821</b>
$s_x = 8.28$		$s_y = 32.91$		

## Data Science – Maths – Part - 6

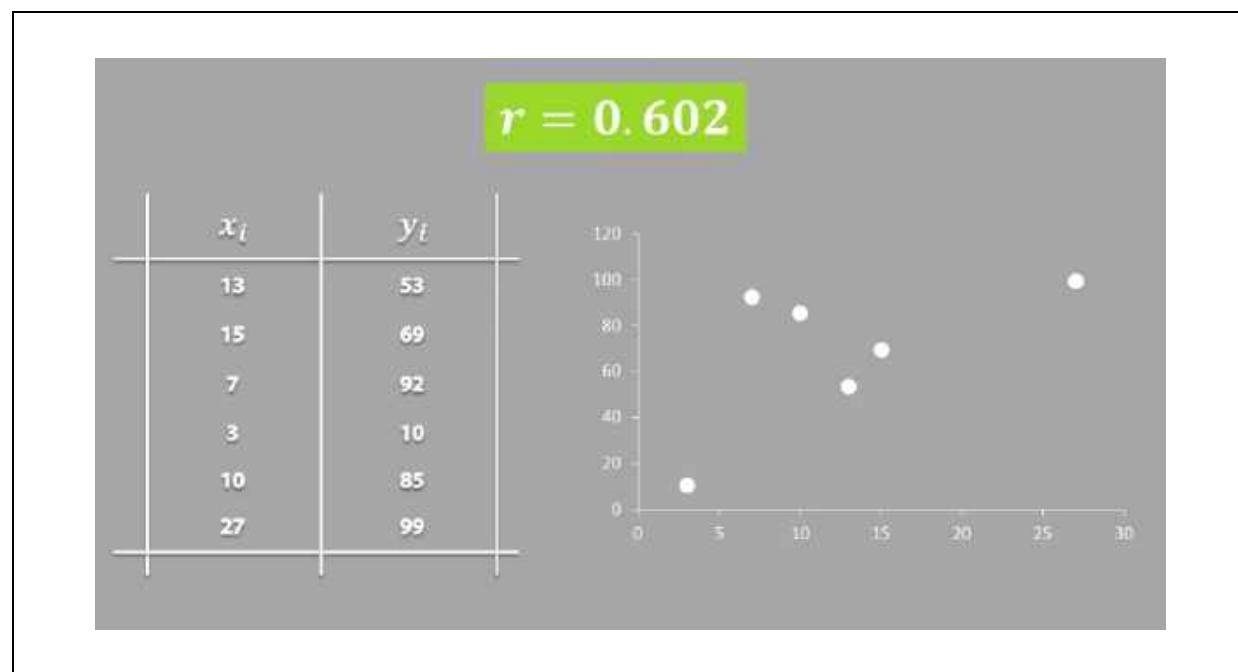
---

$r = \frac{1}{(6 - 1)(8.28)(32.91)} \left[ 821 \right]$				
$x_i$	$y_i$	$(x_i - \bar{x})$	$(y_i - \bar{y})$	$(x_i - \bar{x})(y_i - \bar{y})$
13	53	0.5	-15	-7.5
15	69	2.5	1	2.5
7	92	-5.5	24	-132
3	10	-9.5	-58	551
10	85	-2.5	17	-42.5
27	99	14.5	31	449.5
$\bar{x} = 12.5$		$\bar{y} = 68$	$\text{SUM} = 821$	
$s_x = 8.28$		$s_y = 32.91$		

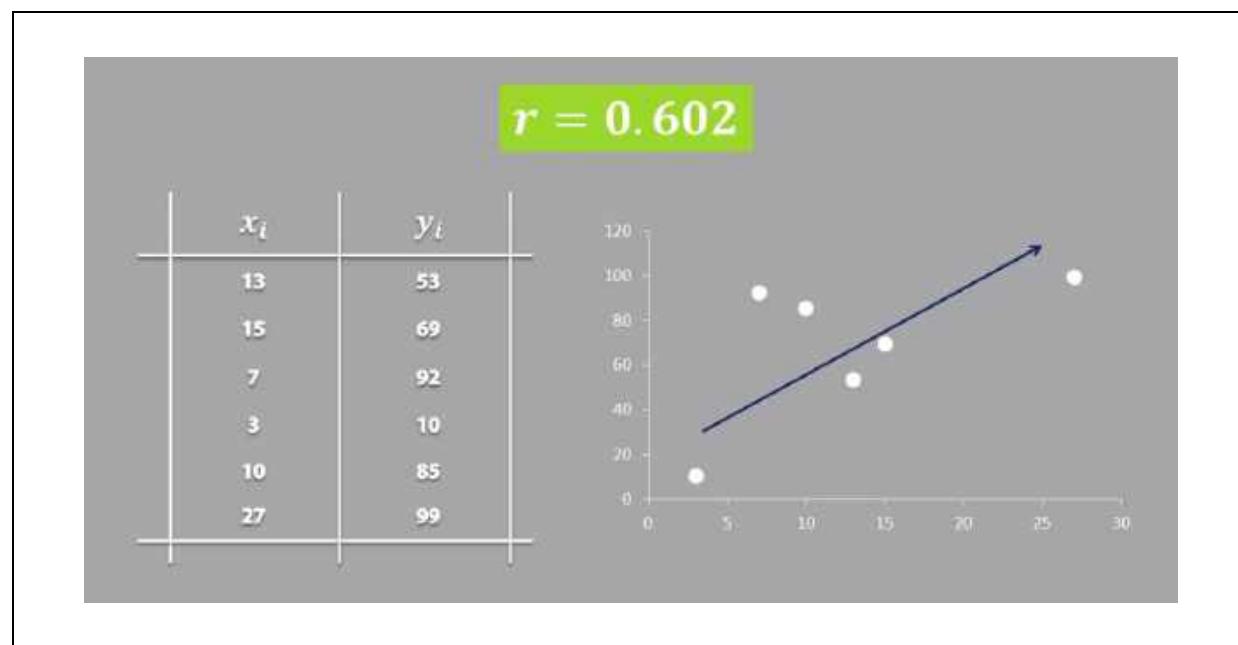
$r = 0.602$				
$x_i$	$y_i$	$(x_i - \bar{x})$	$(y_i - \bar{y})$	$(x_i - \bar{x})(y_i - \bar{y})$
13	53	0.5	-15	-7.5
15	69	2.5	1	2.5
7	92	-5.5	24	-132
3	10	-9.5	-58	551
10	85	-2.5	17	-42.5
27	99	14.5	31	449.5
$\bar{x} = 12.5$		$\bar{y} = 68$	$\text{SUM} = 821$	
$s_x = 8.28$		$s_y = 32.91$		

## Data Science – Maths – Part - 6

---



- ✓ So, here its explains the correlation is +0.6, its upwards direction



**7. Maths - Statistics – PART – 7****Contents**

<b>1. Correlation.....</b>	<b>2</b>
<b>2. Regression .....</b>	<b>3</b>
<b>3. Regression line .....</b>	<b>4</b>
3.1. Example: Regression line .....	5
<b>4.1. Regression line formula .....</b>	<b>6</b>
<b>5. R – SQUARED .....</b>	<b>20</b>

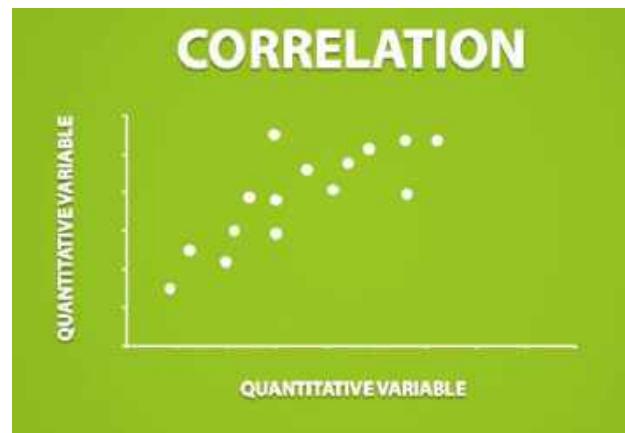
## 7. Maths - Statistics – PART – 7

- ✓ In this chapter we will discuss about Regression and R-Squared

- REGRESSION
- R-SQUARED

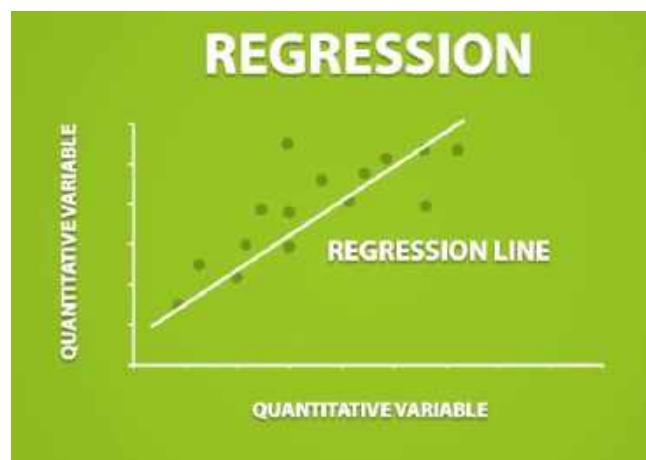
### 1. Correlation

- ✓ Correlation explains about how we measure the **direction** and **strength** of linear relationship in between quantitative variables



## 2. Regression

- ✓ Regression explains about how we can draw line in between the points
- ✓ This line represents about the pattern of the data
- ✓ This line is called as regression line



### 3. Regression line

- ✓ Regression line predicts the change in **Y** when **X** increases by one unit
- ✓ Here change in Y can be increase/decrease

## REGRESSION LINE

PREDICTS THE CHANGE IN "Y" WHEN "X" INCREASES BY ONE UNIT

## REGRESSION LINE

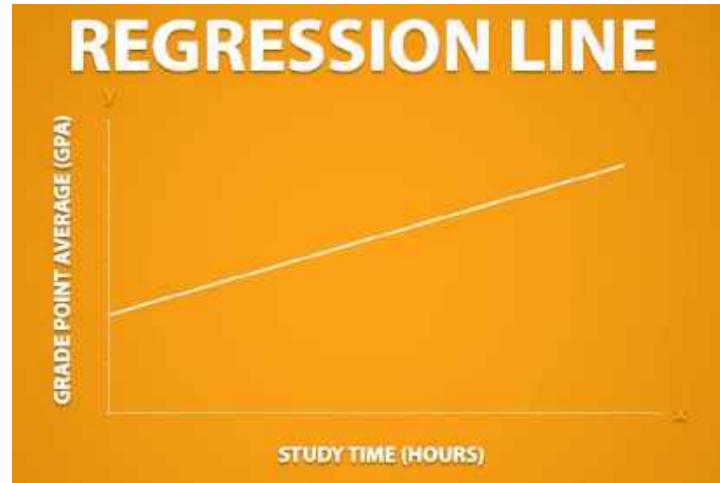
PREDICTS THE INCREASE IN "Y" WHEN "X" INCREASES BY ONE UNIT

## REGRESSION LINE

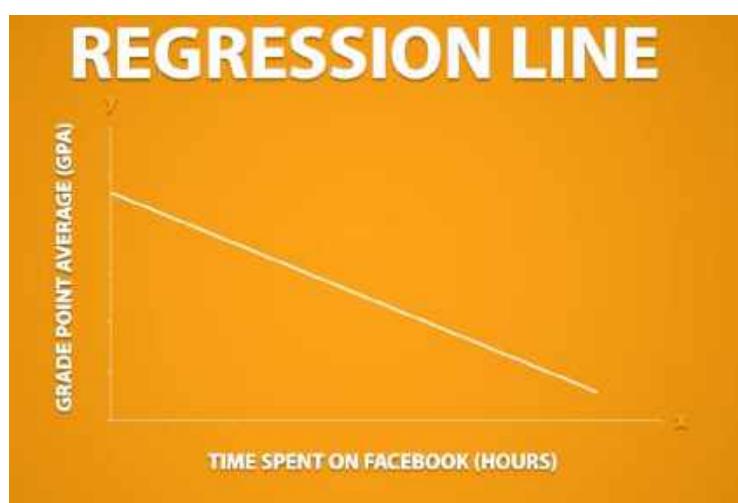
PREDICTS THE DECREASE IN "Y" WHEN "X" INCREASES BY ONE UNIT

### 3.1. Example: Regression line

- ✓ If we study time (hours) on X axis and grade on Y axis then we should access some positive relationship in between these two variables.
- ✓ Generally speaking the more you study then we will get good grade ☺



- ✓ Instead of studying if we spend more time on Facebook then we will get negative relationship



#### 4.1. Regression line formula

## REGRESSION LINE

$$\hat{y} = b_0 + b_1 x$$

## REGRESSION LINE

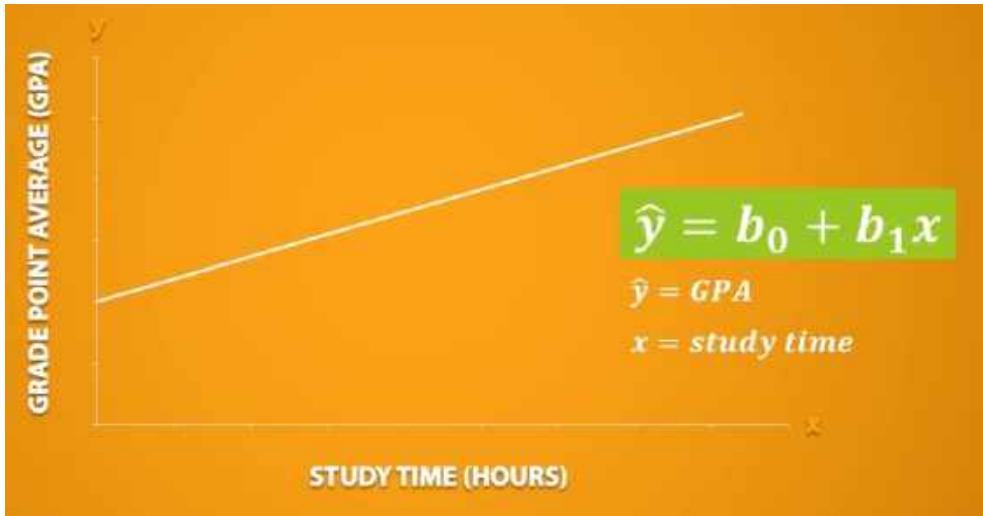
$$\hat{y} = b_0 + b_1 x$$

PREDICTED  
VALUE OF Y

Y INTERCEPT

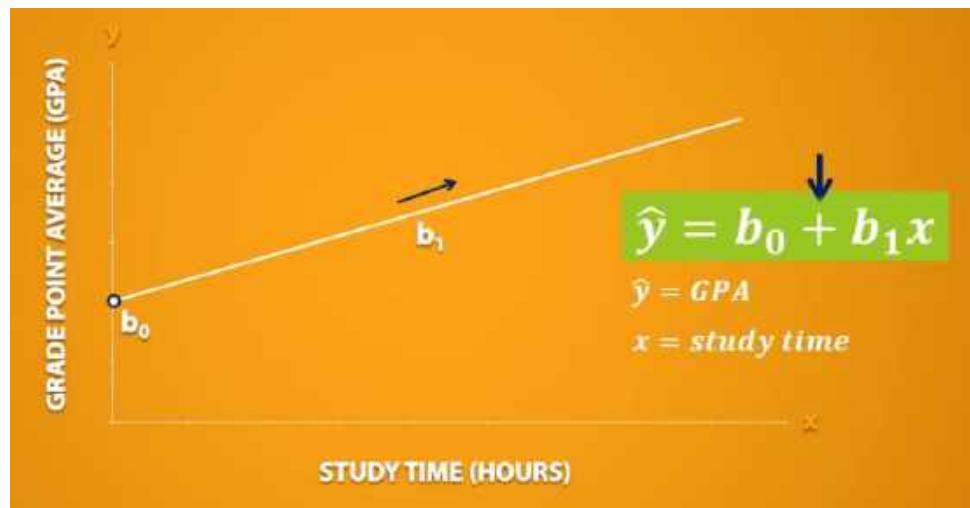
SLOPE

ANY VALUE  
OF X



Data Science – Maths – Part - 7

---



$$\hat{y} = b_0 + b_1 x$$
$$b_0 = \bar{y} - b_1 \bar{x}$$

$$\hat{y} = b_0 + b_1 x$$
$$b_0 = \bar{y} - b_1 \bar{x}$$
$$b_1 = r \frac{s_y}{s_x}$$

## Data Science – Maths – Part - 7

### Scenario



SUPPOSE A RESEARCHER WANTS TO PREDICT A STUDENT'S GPA FROM THE AMOUNT OF TIME THEY STUDY EACH WEEK



SUPPOSE A RESEARCHER WANTS TO PREDICT A STUDENT'S GPA FROM THE AMOUNT OF TIME THEY STUDY EACH WEEK

STUDENT	STUDY TIME	GPA
GEORGE	1	2.0
VANESSA	2	1.5
GATSBY	3	2.5
SOPHIA	5	3.5
EMMA	6	3.0
MELISSA	8	4.0
PATRICK	10	4.5

## Data Science – Maths – Part - 7



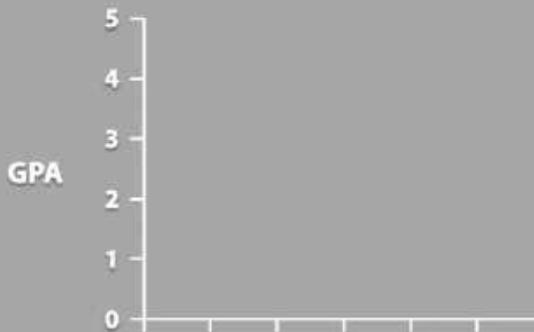
SUPPOSE A RESEARCHER WANTS TO PREDICT A STUDENT'S GPA FROM THE AMOUNT OF TIME THEY STUDY EACH WEEK



STUDY TIME	GPA
1	2.0
2	1.5
3	2.5
5	3.5
6	3.0
8	4.0
10	4.5



SUPPOSE A RESEARCHER WANTS TO PREDICT A STUDENT'S GPA FROM THE AMOUNT OF TIME THEY STUDY EACH WEEK

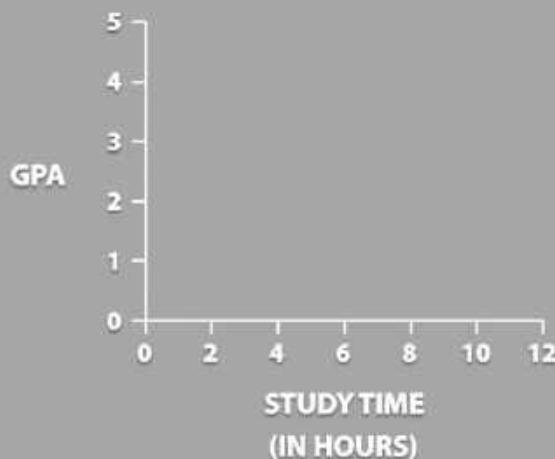


STUDY TIME	$y_i$
1	2.0
2	1.5
3	2.5
5	3.5
6	3.0
8	4.0
10	4.5

## Data Science – Maths – Part - 7



SUPPOSE A RESEARCHER WANTS TO PREDICT A STUDENT'S GPA  
FROM THE AMOUNT OF TIME THEY STUDY EACH WEEK

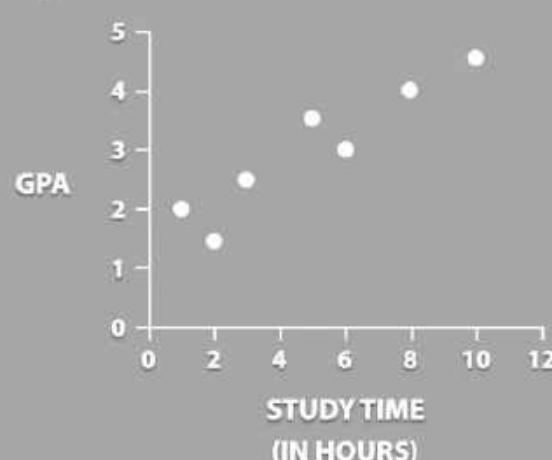


$x_i$	$y_i$
1	2.0
2	1.5
3	2.5
5	3.5
6	3.0
8	4.0
10	4.5

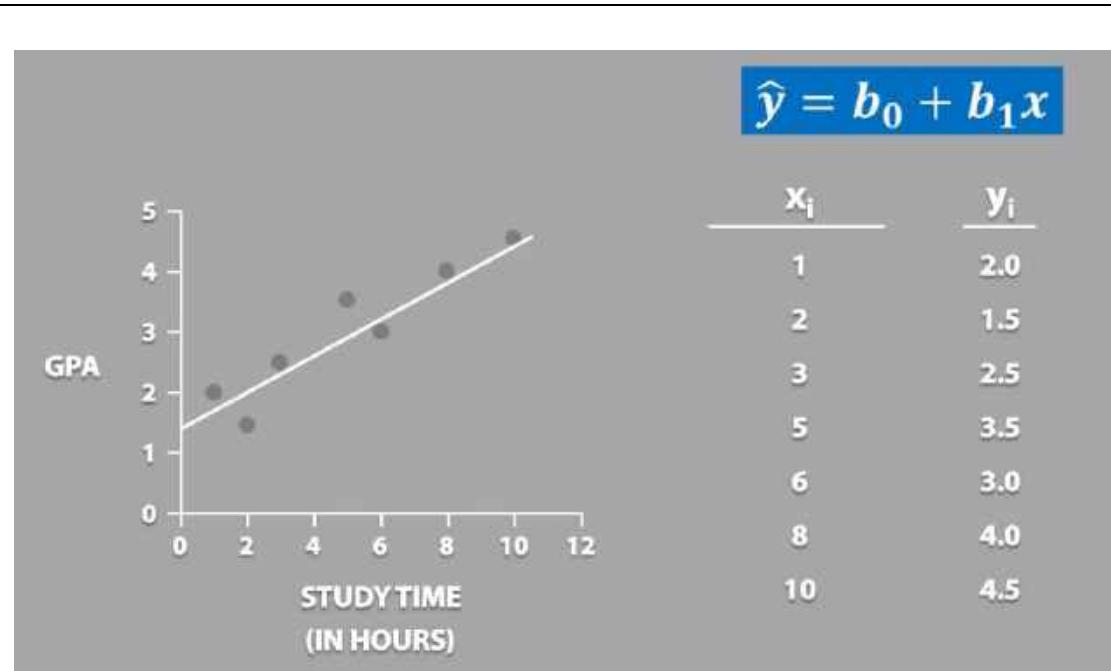
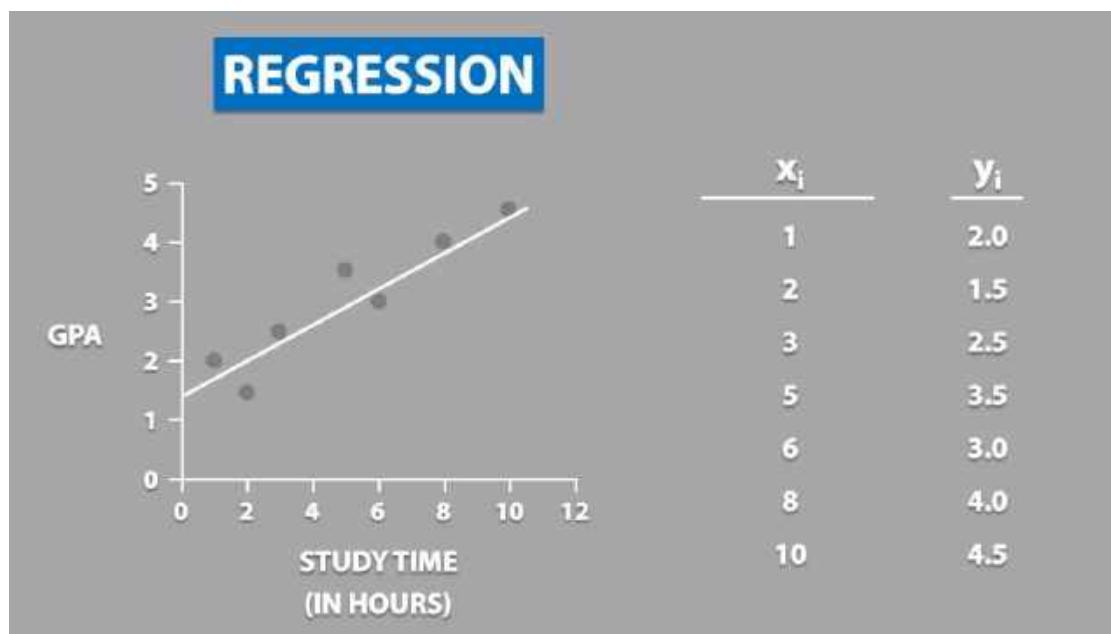
- ✓ Let's use scatter plot to plot this data



SUPPOSE A RESEARCHER WANTS TO PREDICT A STUDENT'S GPA  
FROM THE AMOUNT OF TIME THEY STUDY EACH WEEK

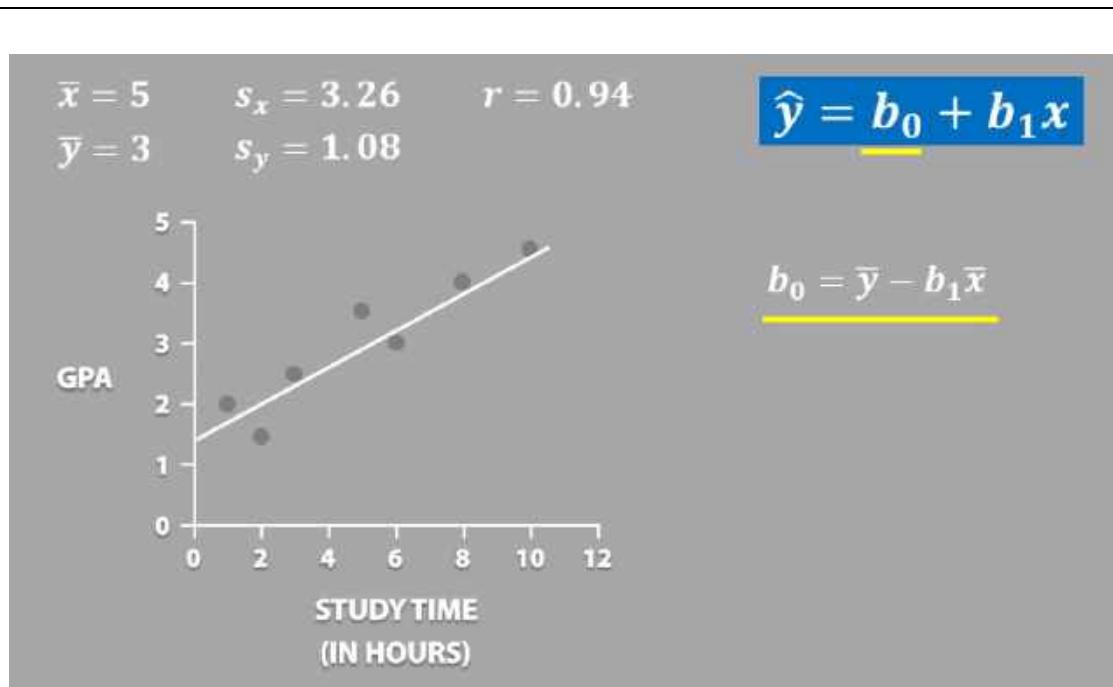
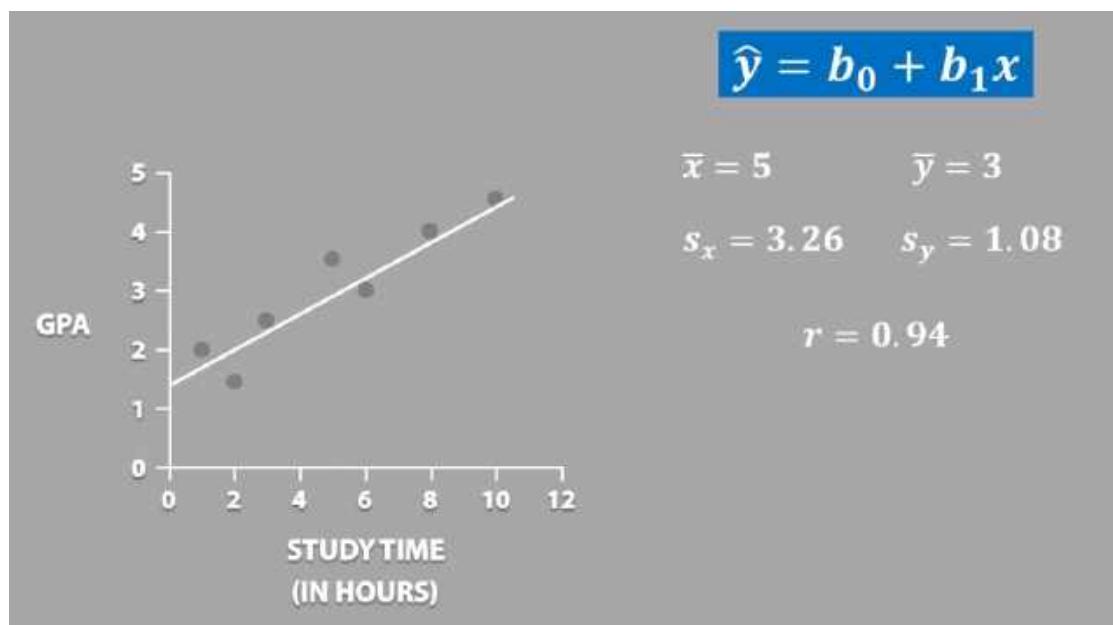


$x_i$	$y_i$
1	2.0
2	1.5
3	2.5
5	3.5
6	3.0
8	4.0
10	4.5



## Data Science – Maths – Part - 7

---



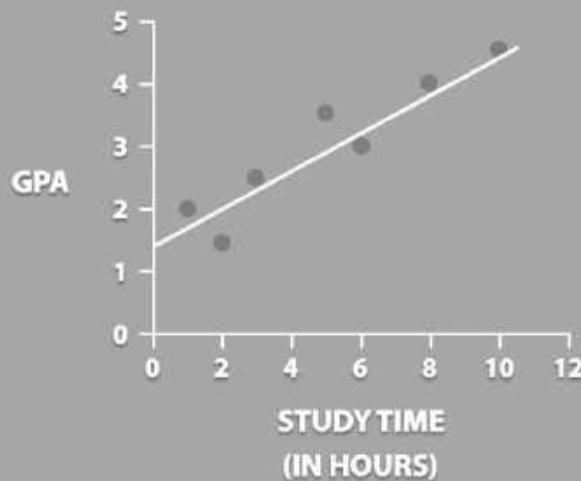
## Data Science – Maths – Part - 7

---

$$\bar{x} = 5 \quad s_x = 3.26 \quad r = 0.94$$

$$\bar{y} = 3 \quad s_y = 1.08$$

$$\hat{y} = b_0 + b_1 x$$



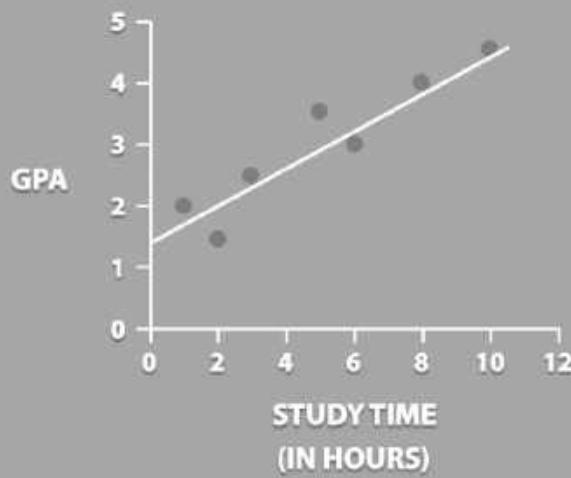
$$b_0 = \bar{y} - b_1 \bar{x}$$

$$b_1 = r \frac{s_y}{s_x}$$

$$\bar{x} = 5 \quad s_x = 3.26 \quad r = 0.94$$

$$\bar{y} = 3 \quad s_y = 1.08$$

$$\hat{y} = b_0 + b_1 x$$



$$b_0 = \bar{y} - b_1 \bar{x}$$

$$\Rightarrow b_1 = 0.311$$

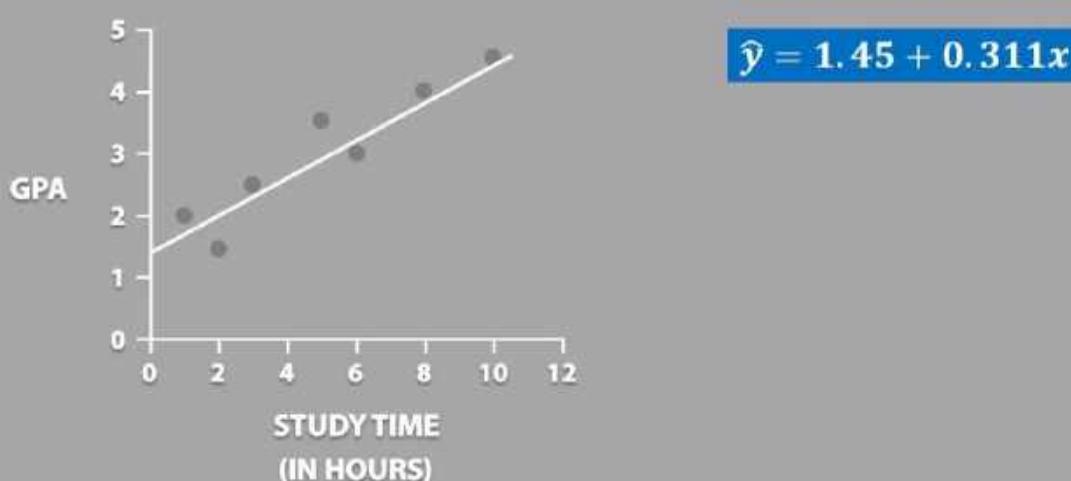
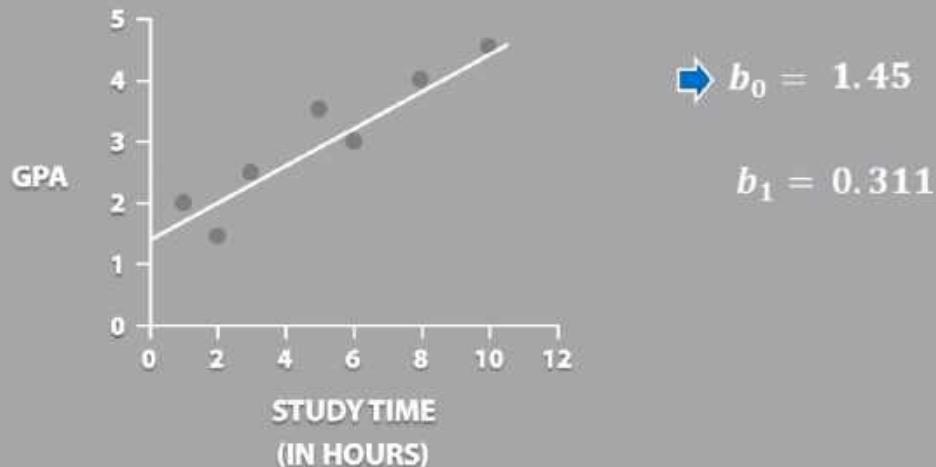
## Data Science – Maths – Part - 7

---

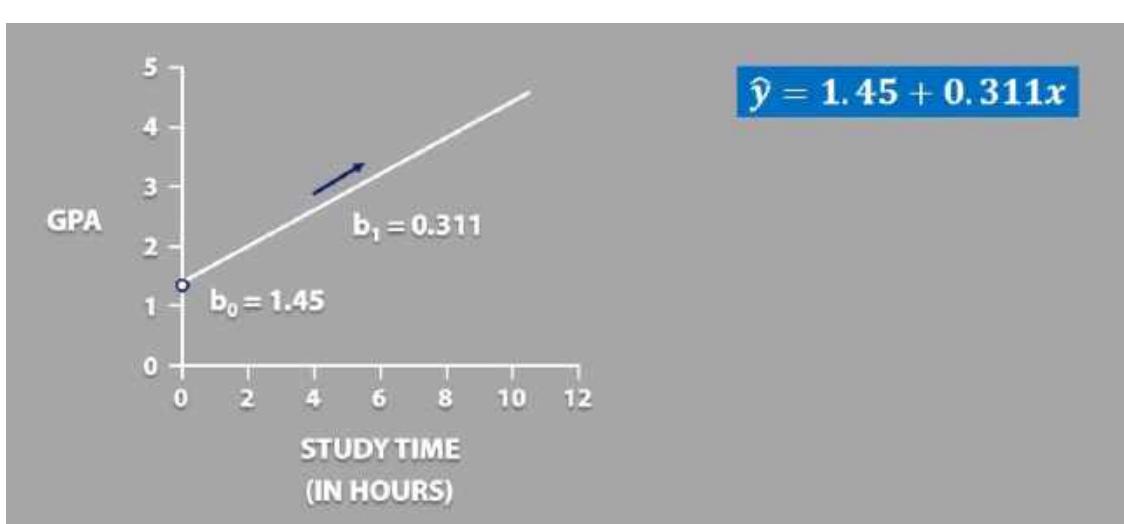
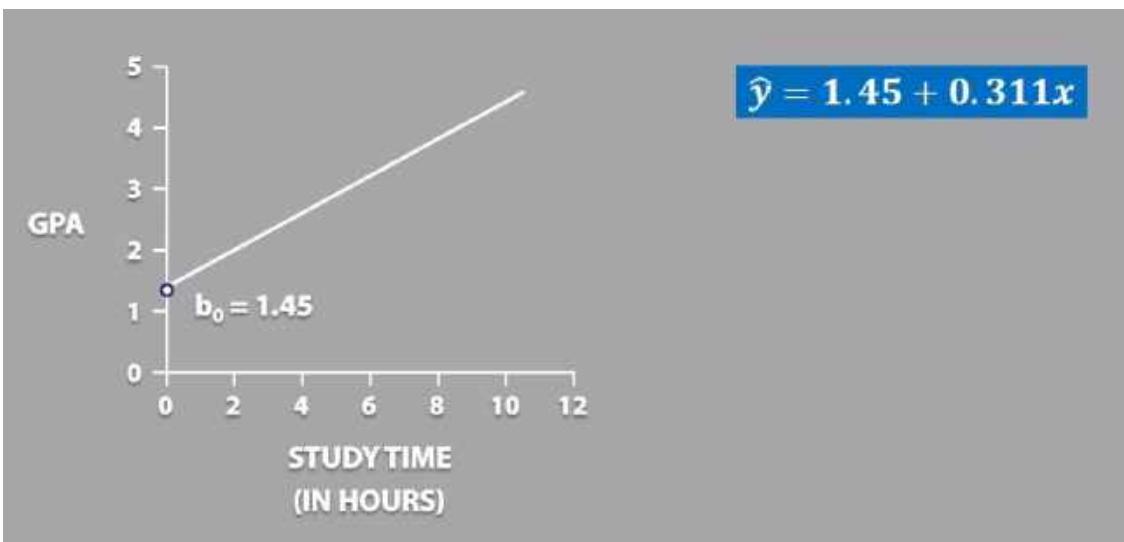
$$\bar{x} = 5 \quad s_x = 3.26 \quad r = 0.94$$

$$\bar{y} = 3 \quad s_y = 1.08$$

$$\hat{y} = b_0 + b_1 x$$



## Data Science – Maths – Part - 7

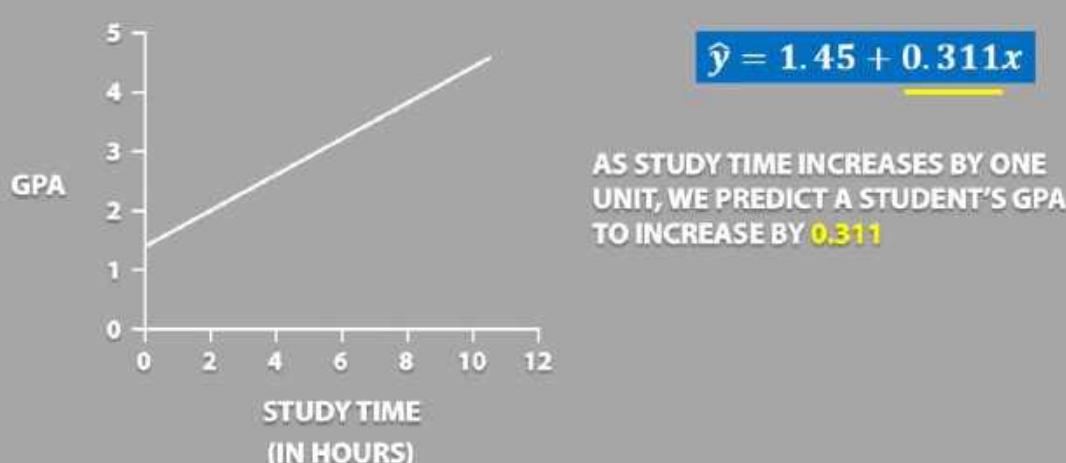


## Data Science – Maths – Part - 7

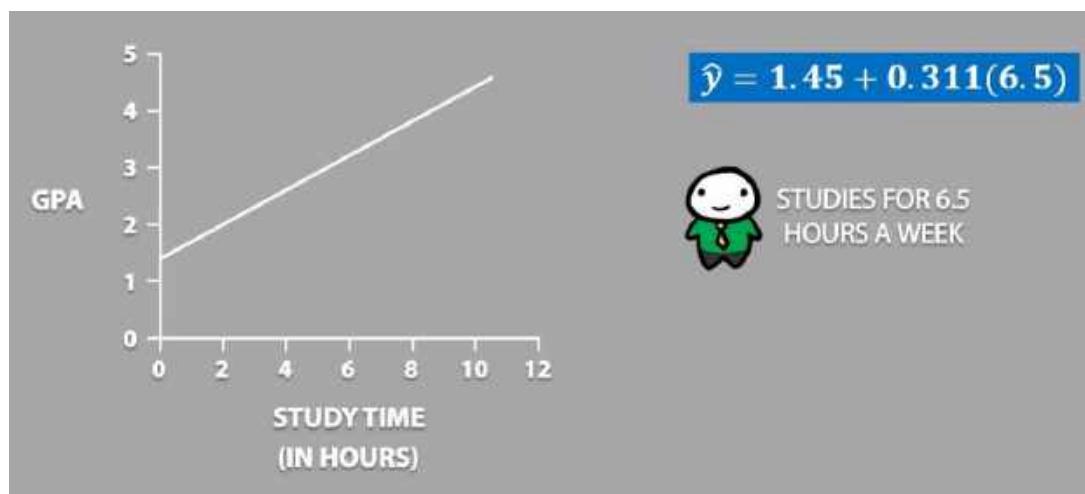
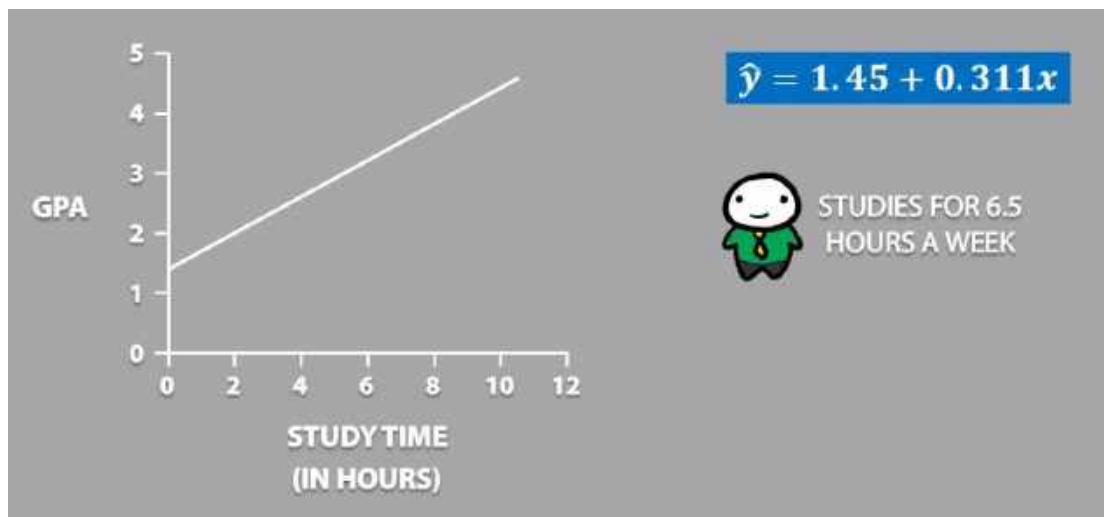
THE LINE OF LEAST SQUARES REGRESSION

$$\hat{y} = b_0 + b_1x$$

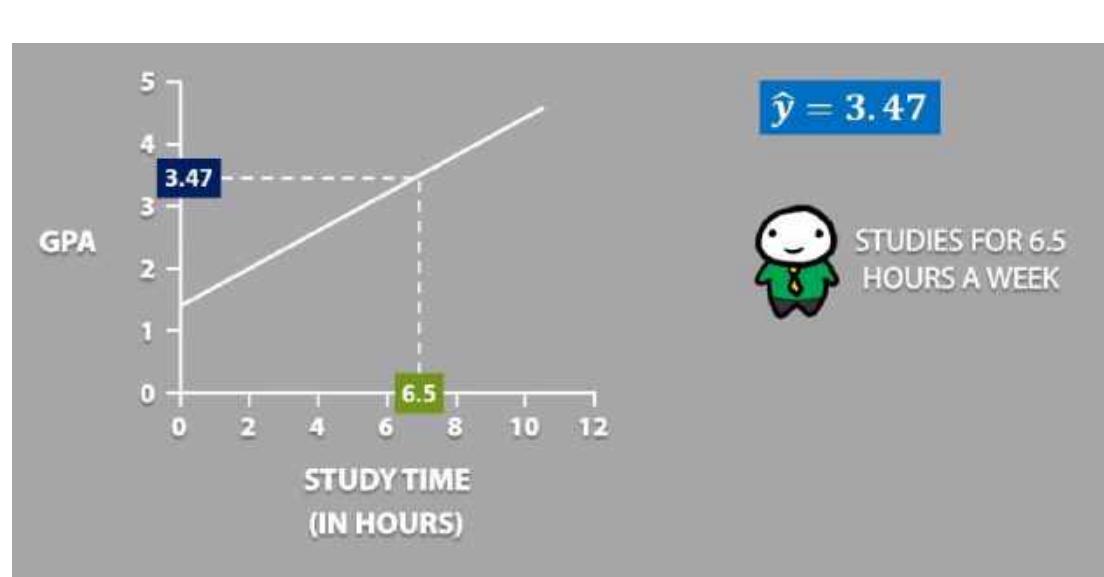
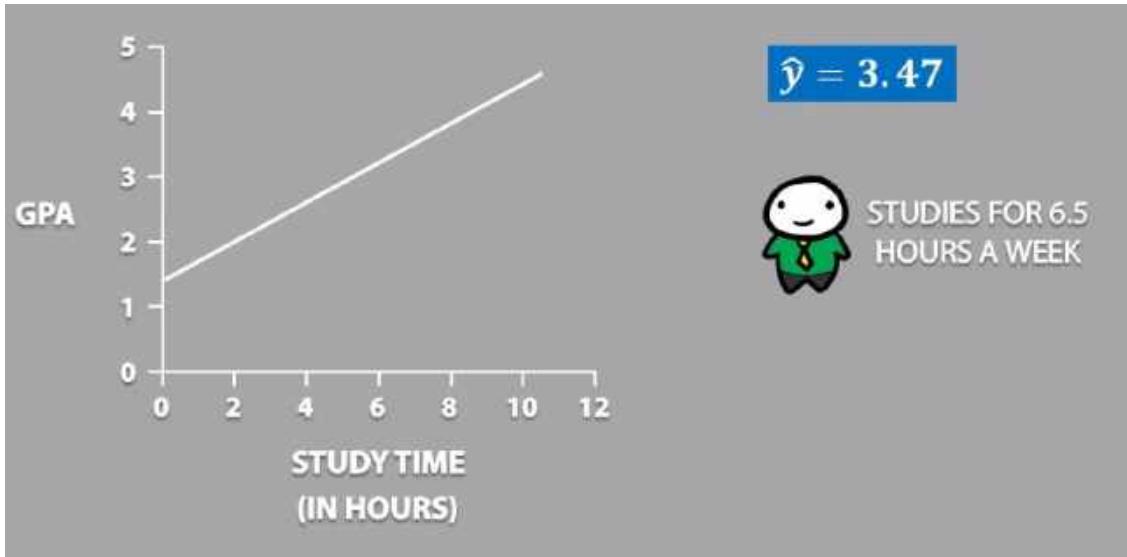
THE SLOPE OF A REGRESSION LINE PREDICTS THE CHANGE IN "Y" WHEN "X" INCREASES BY ONE UNIT



## Data Science – Maths – Part - 7



## Data Science – Maths – Part - 7



**5. R – SQUARED****R-SQUARED****R-SQUARED =  $r^2$** **R-SQUARED =  $r^2$**   
**=  $r \times r$**

$r$ 

HAS VALUES BETWEEN -1 AND 1

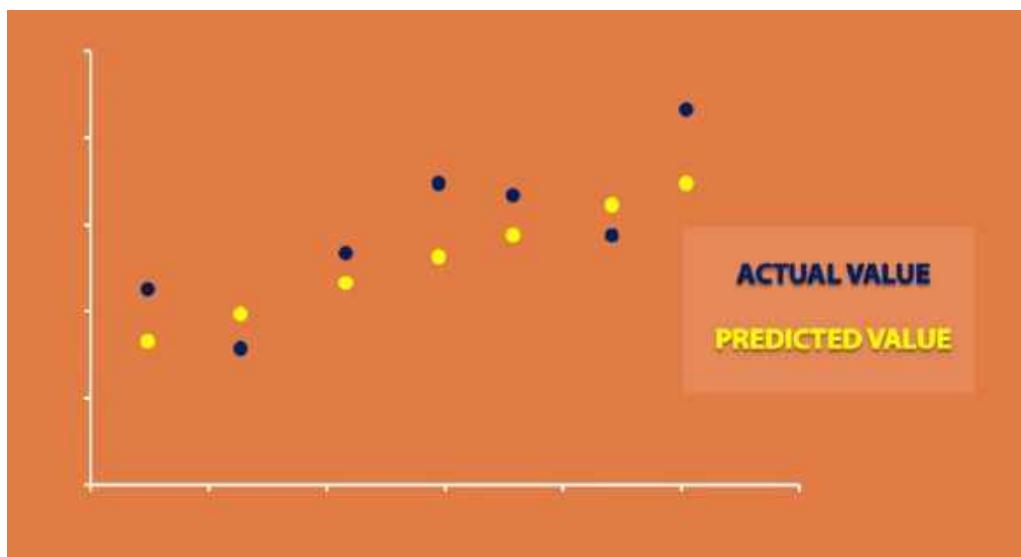
MEASURES THE LINEAR  
RELATIONSHIP BETWEEN TWO  
QUANTITATIVE VARIABLES WITH  
RESPECT TO DIRECTION AND  
STRENGTH

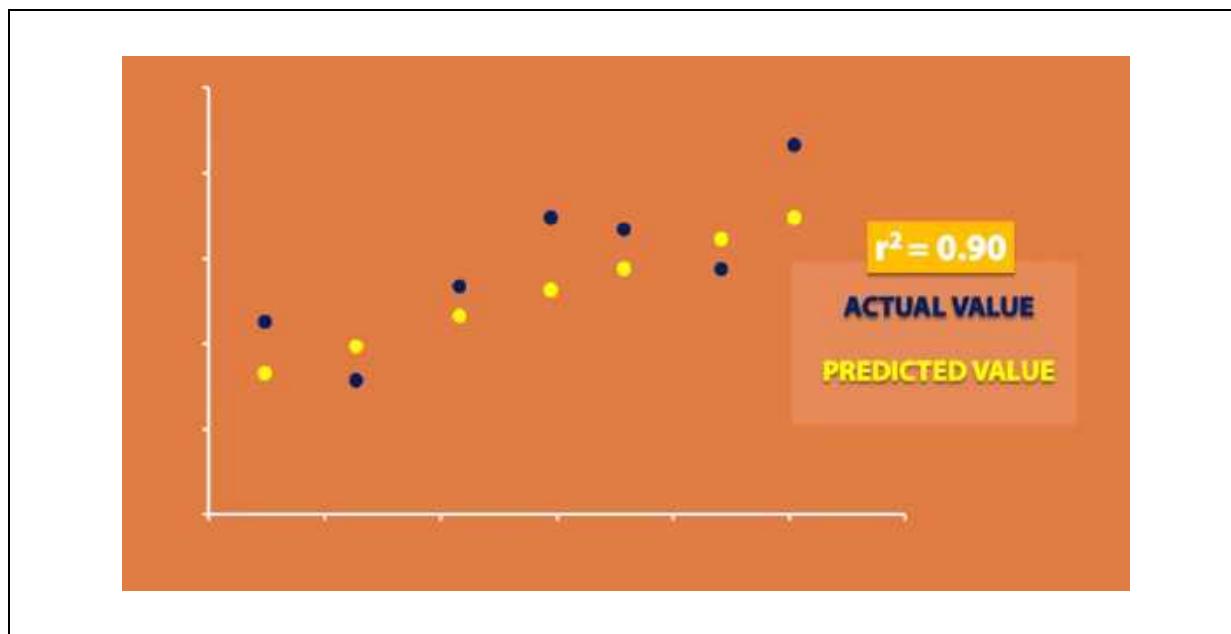
 $r^2$ 

HAS VALUES BETWEEN 0 AND 1

IS A MEASURE OF HOW CLOSE  
EACH DATA POINT FITS TO THE  
REGRESSION LINE

TELLS US HOW WELL THE  
REGRESSION LINE PREDICTS  
ACTUAL VALUES

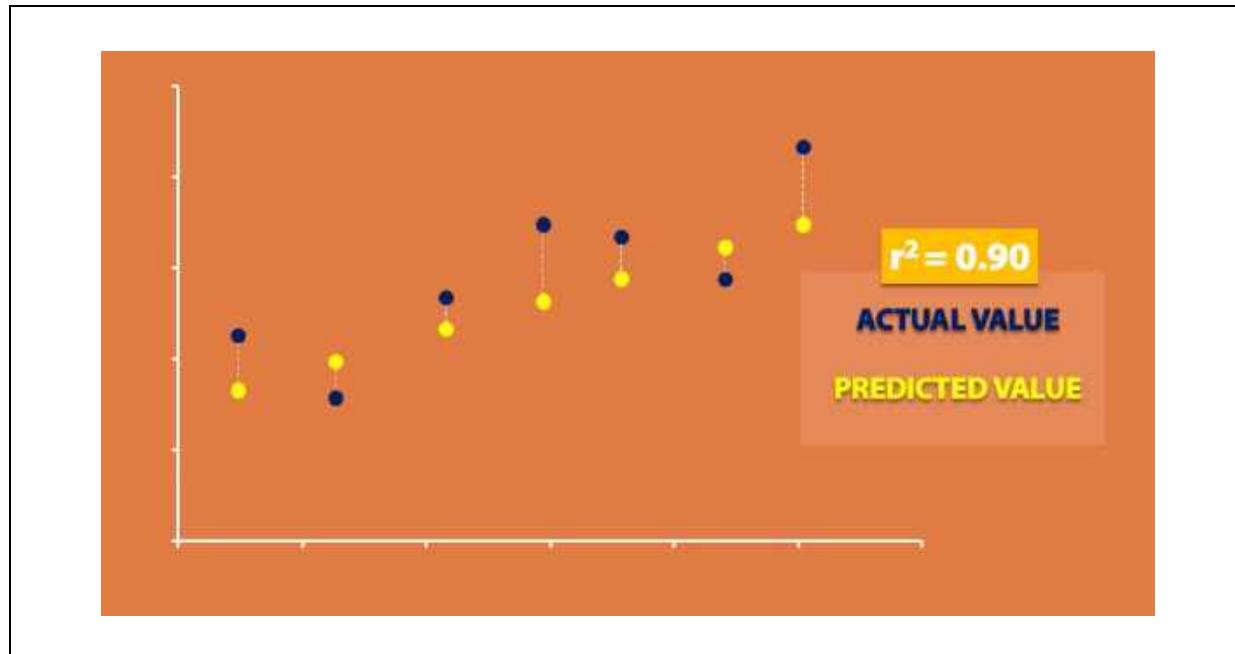




## Data Science – Maths – Part - 7

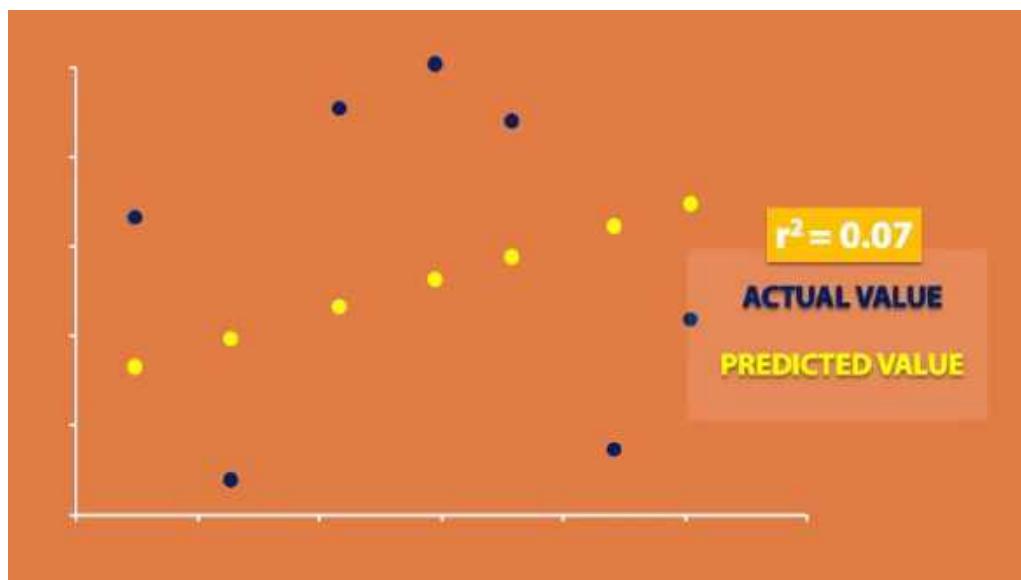
### Case 1: r squared value is high

- ✓ A high value of r square value explains the actual values and predicted values are close together
- ✓ We can clearly see the actual values and predicted values having very less distance in between them



**Case 2:** r squared value is low

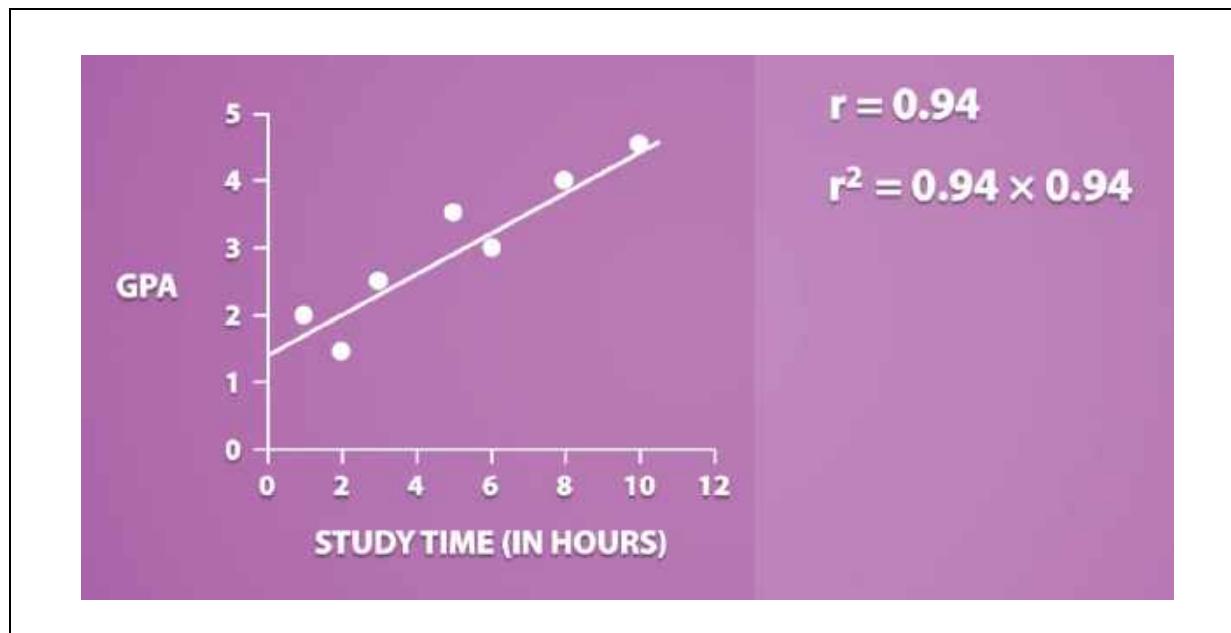
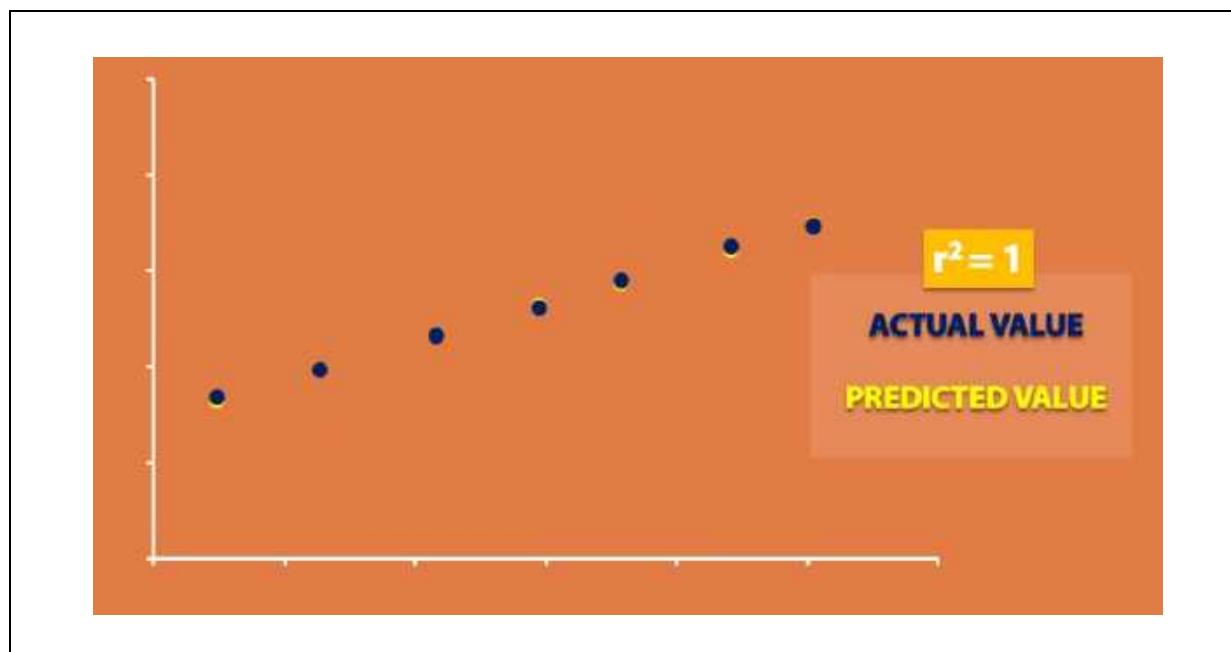
- ✓ A low value of r square value explains the regression line which the data points are does not fit the well
- ✓ We can clearly see the actual values are predicted values having large distance in between them

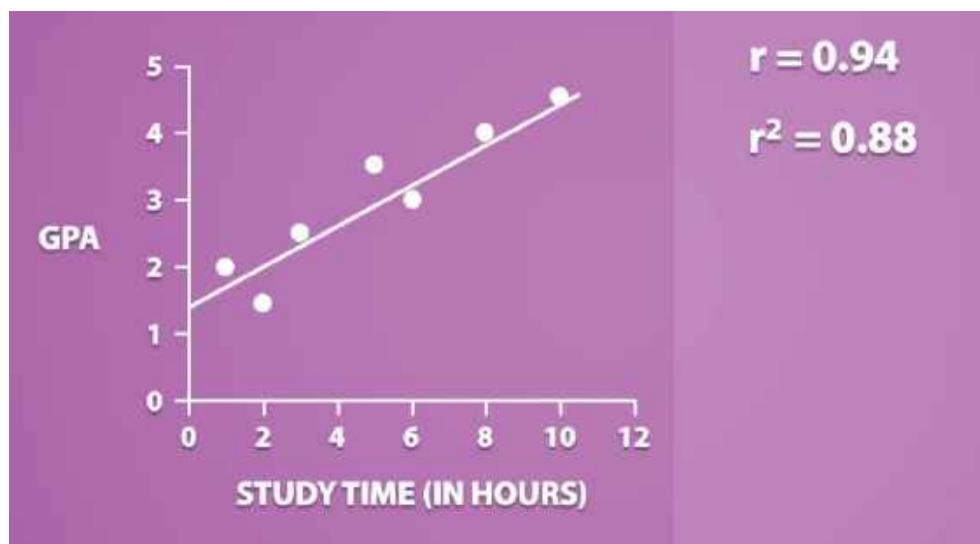


## Data Science – Maths – Part - 7

**Case 3:** r squared value is equals to one

- ✓ This means that we can predict value of y for any given value of x





**8. Maths - Statistics – PART – 8****Contents**

<b>1. Residual .....</b>	2
<b>2. Formula.....</b>	3

## 8. Maths - Statistics – PART – 8

- ✓ In this chapter we will discuss about residuals

# RESIDUAL

### 1. Residual

- ✓ It explains about how far the distance in between the predicted value from the actual value
- ✓ It basically tells the error in prediction

# RESIDUAL

HOW FAR OFF THE **PREDICTED VALUE** IS FROM THE **ACTUAL VALUE**  
TELLS US THE ERROR IN A PREDICTION

## 2. Formula

- ✓ The formula is, actual value of Y minus predicted value of Y

**RESIDUAL = ACTUAL VALUE OF Y – PREDICTED VALUE OF Y**

**RESIDUAL =  $y_i - \hat{y}$**

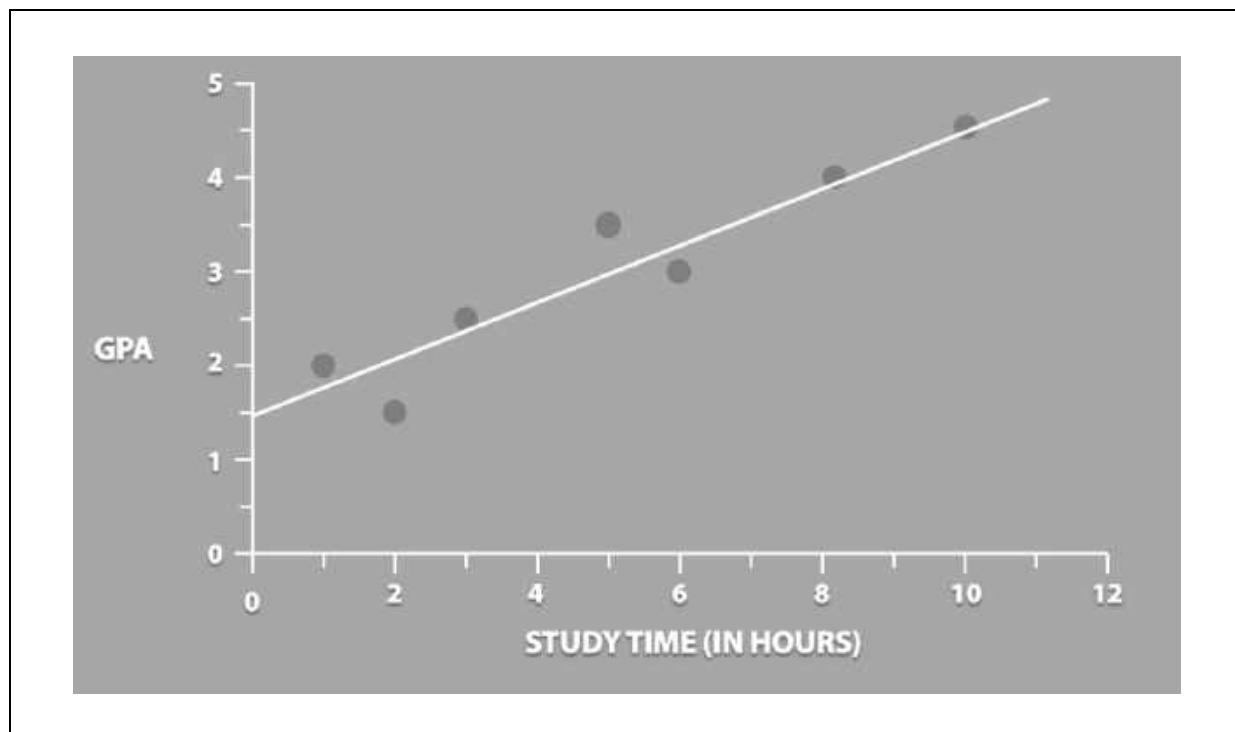
**RESIDUAL =  $y_i - \hat{y}$**

ACTUAL  
VALUE OF Y

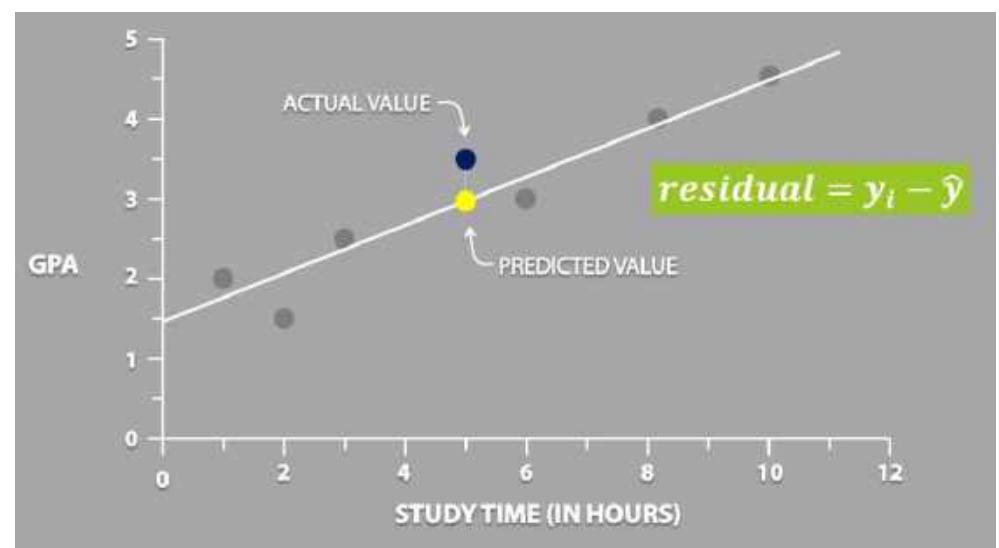
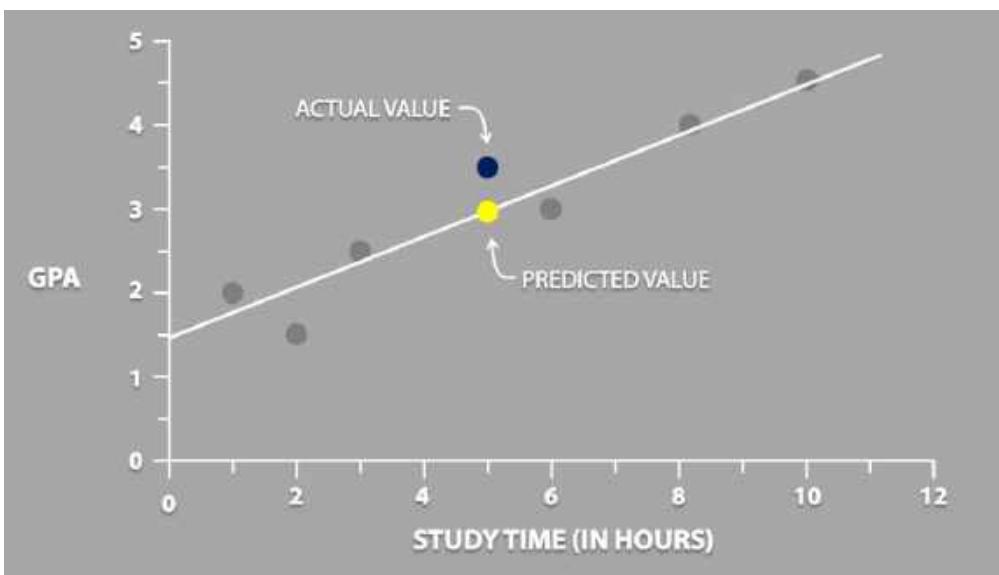
PREDICTED  
VALUE OF Y

## Data Science – Maths – Part - 8

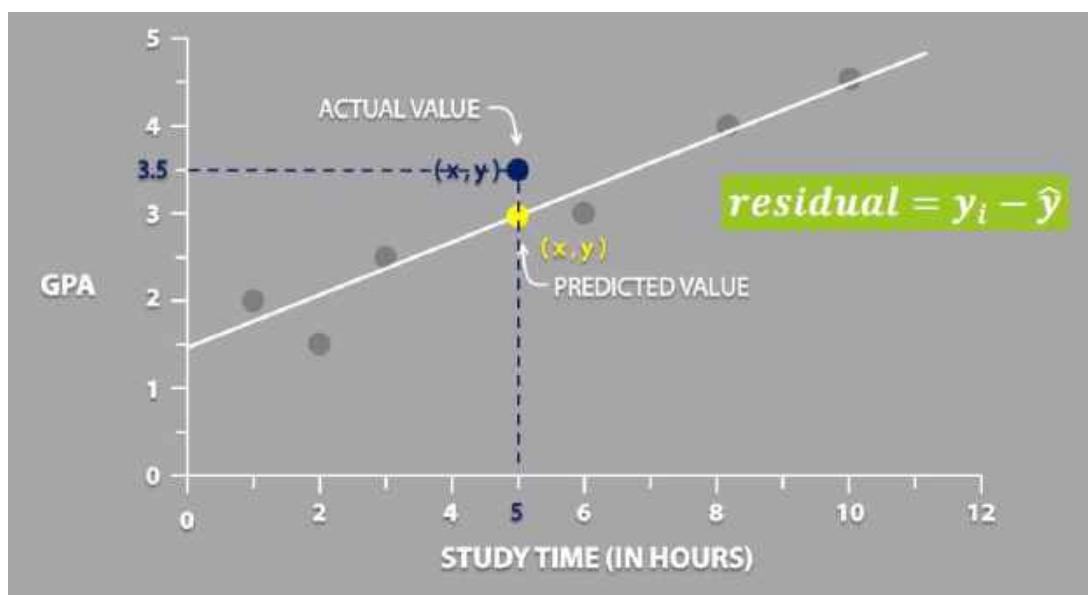
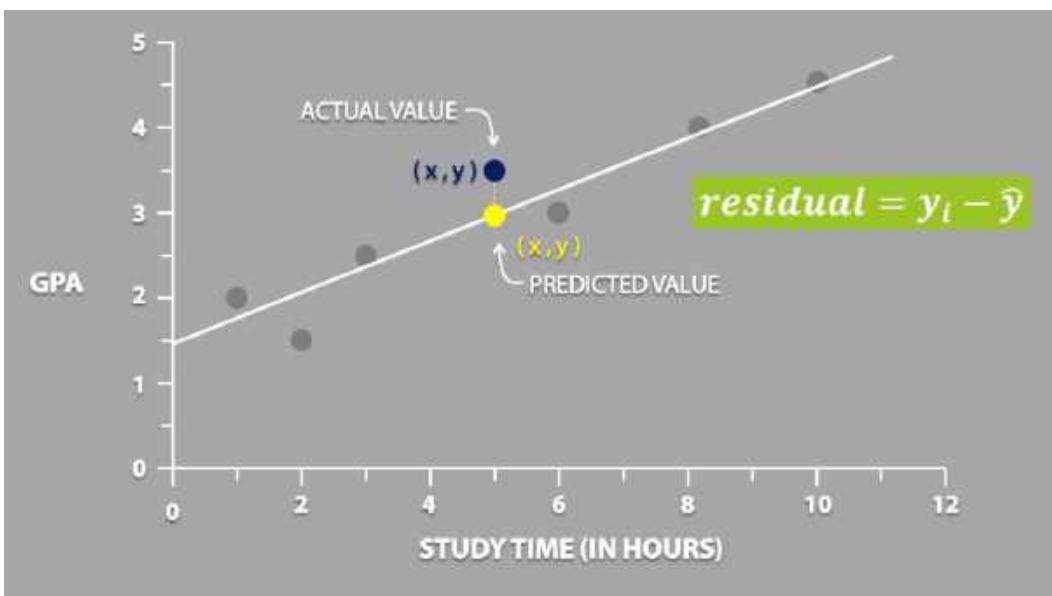
- ✓ From the previous example, study time on X axis and gpa on Y axis



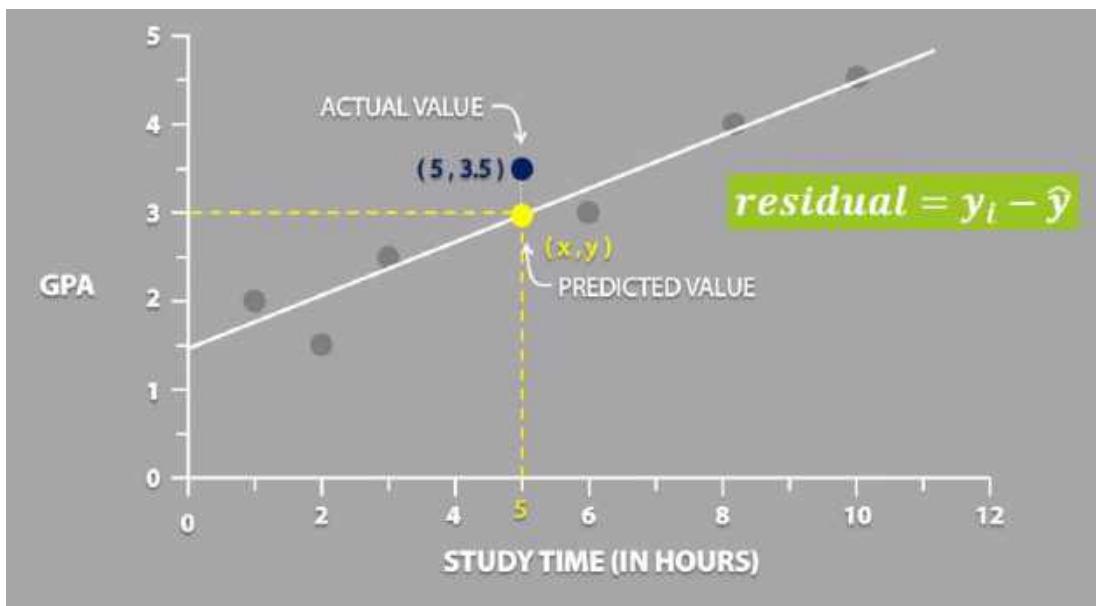
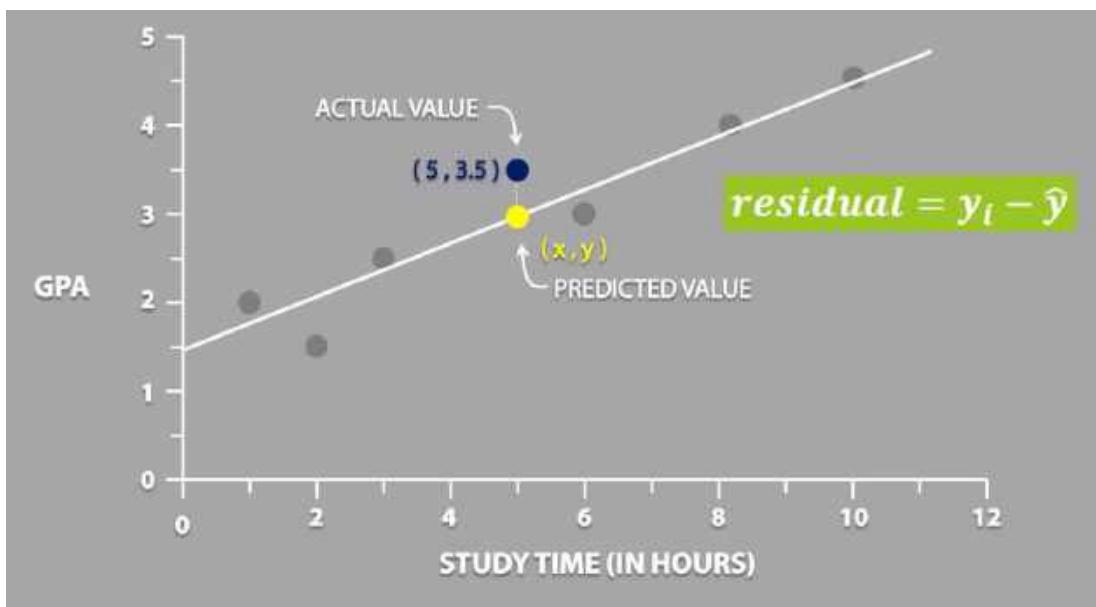
## Data Science – Maths – Part - 8



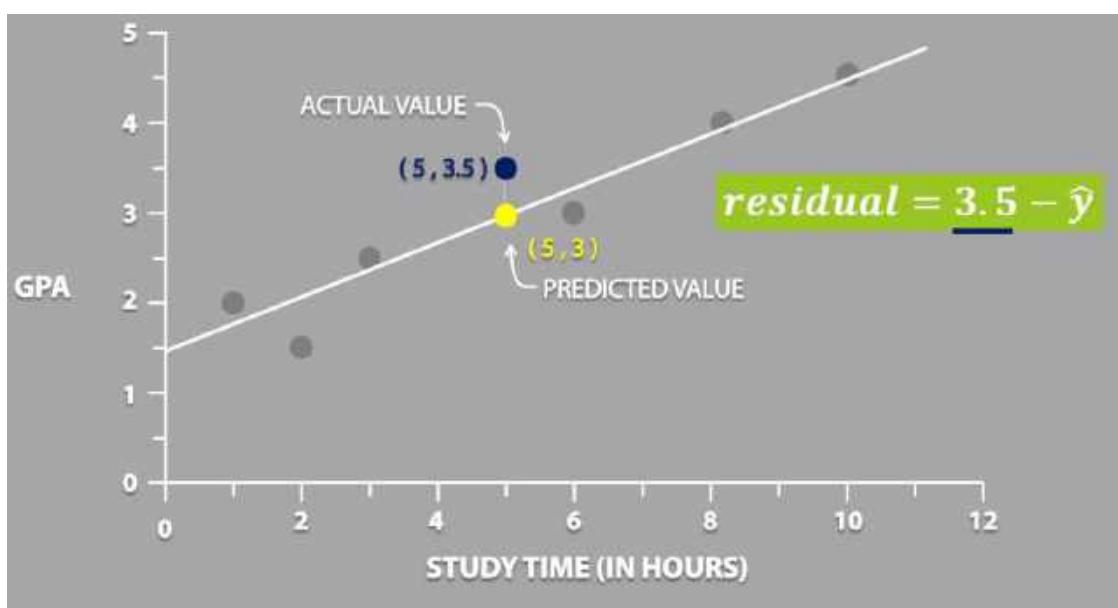
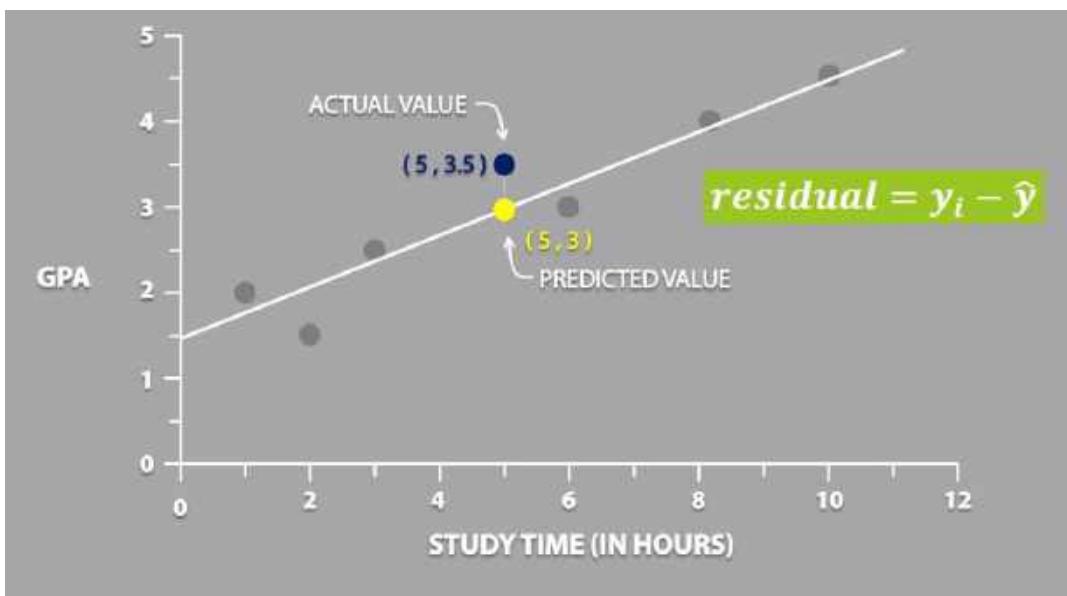
## Data Science – Maths – Part - 8



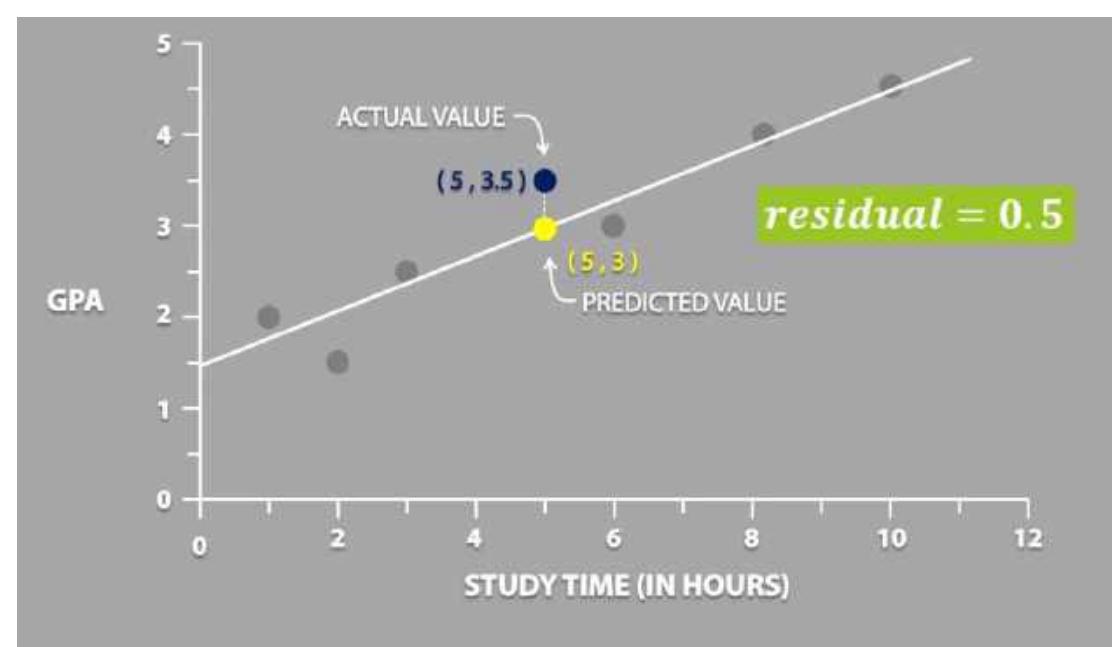
## Data Science – Maths – Part - 8



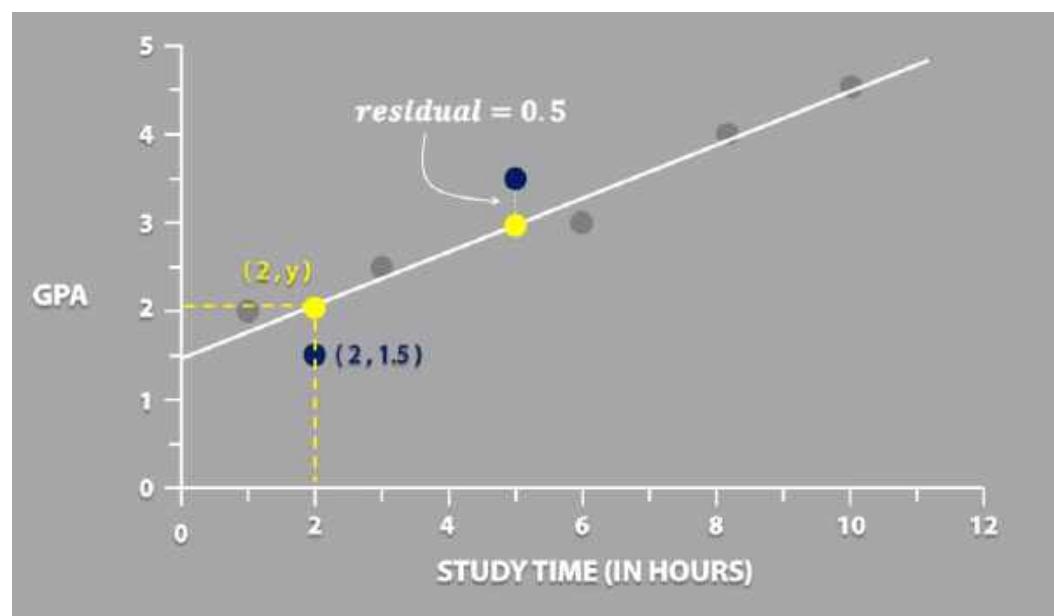
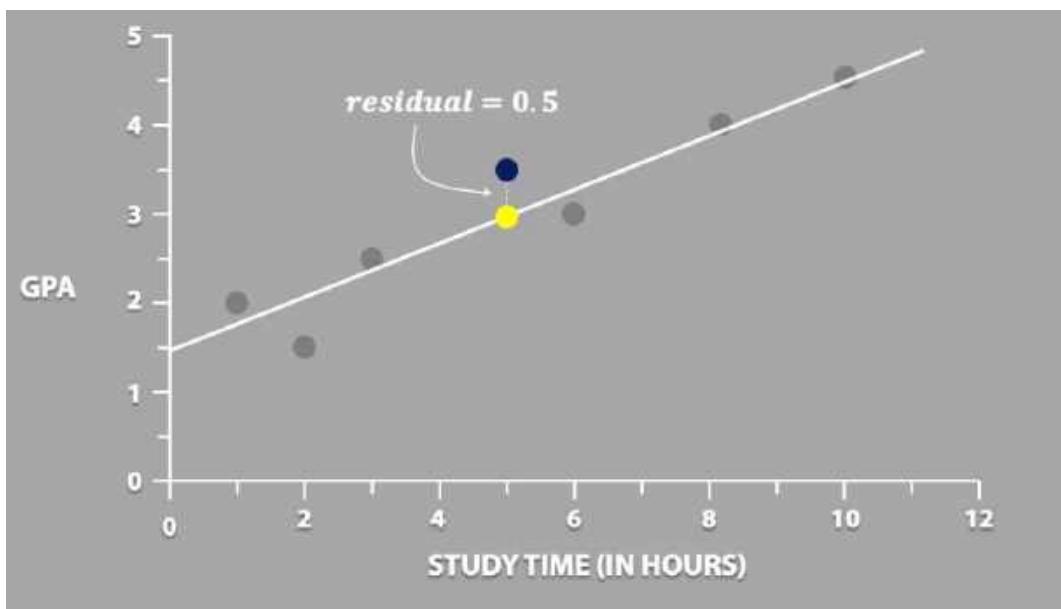
## Data Science – Maths – Part - 8



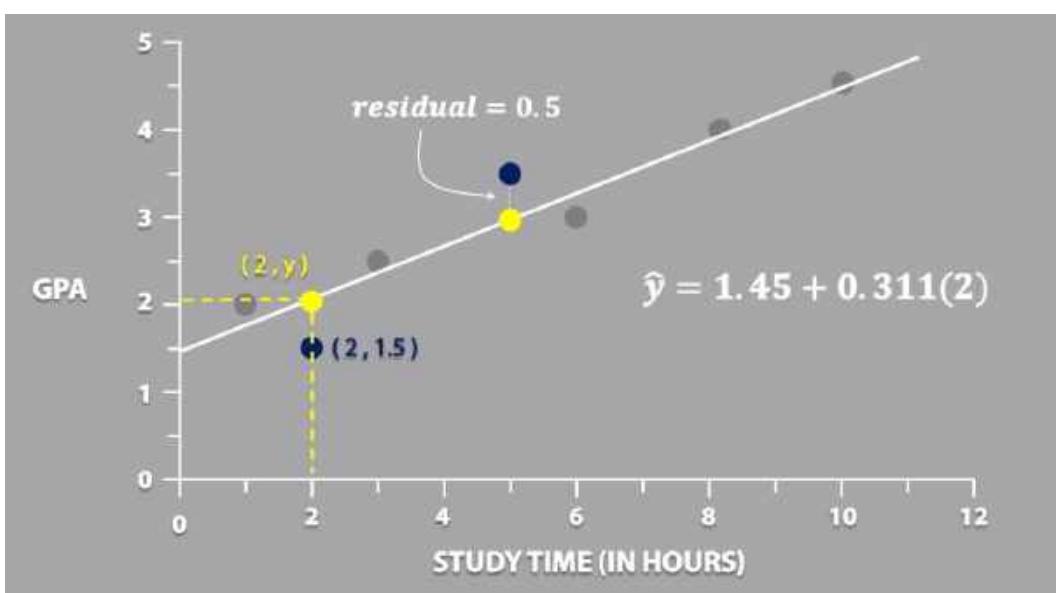
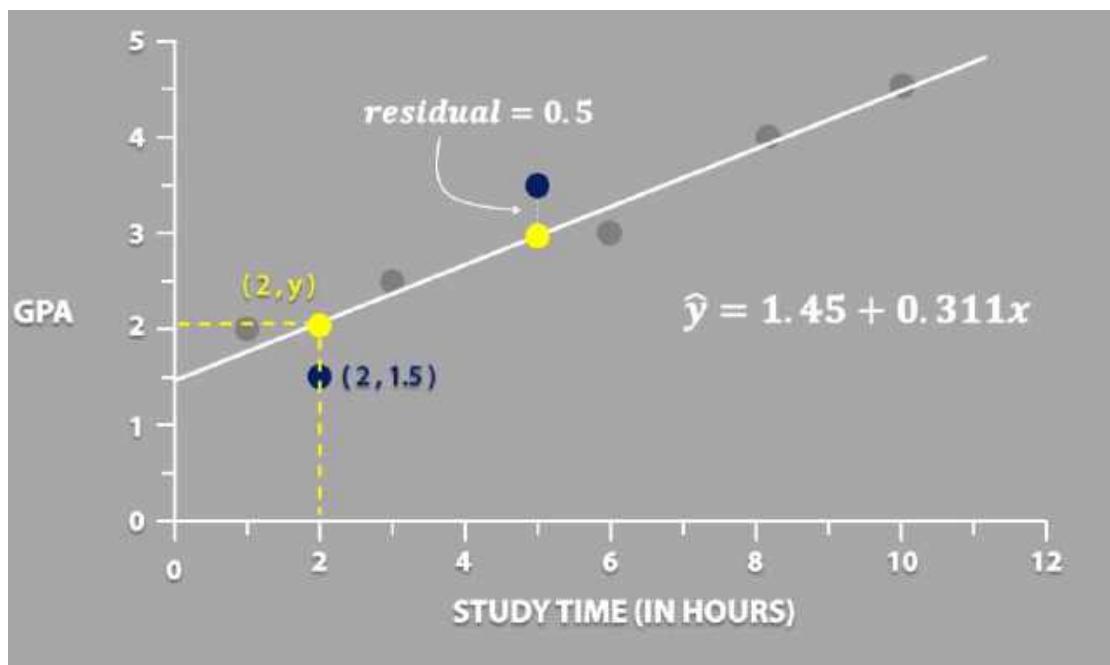
## Data Science – Maths – Part - 8



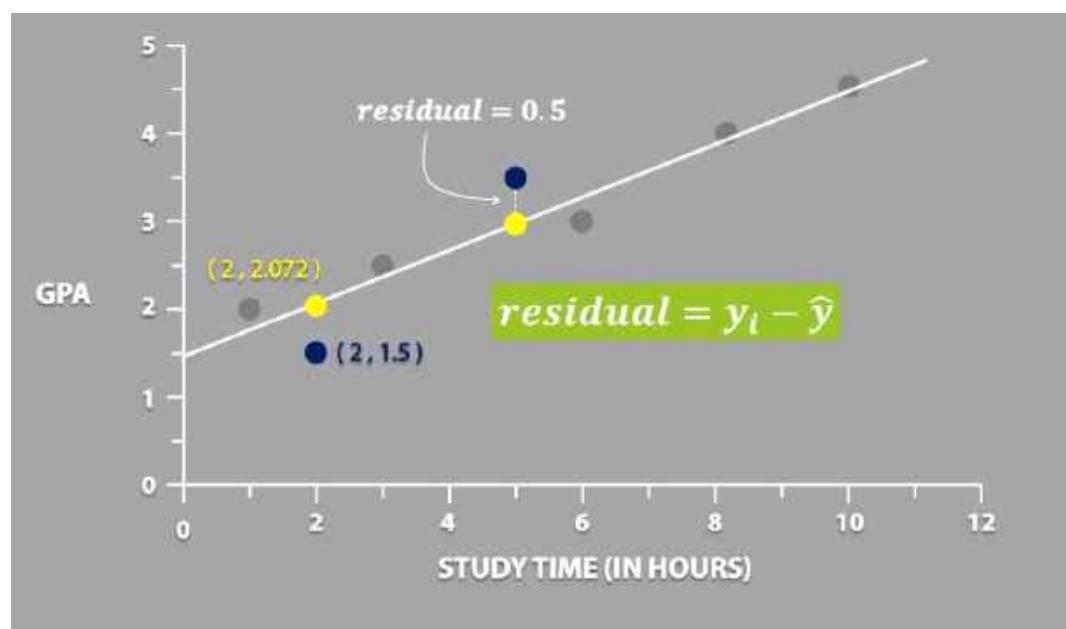
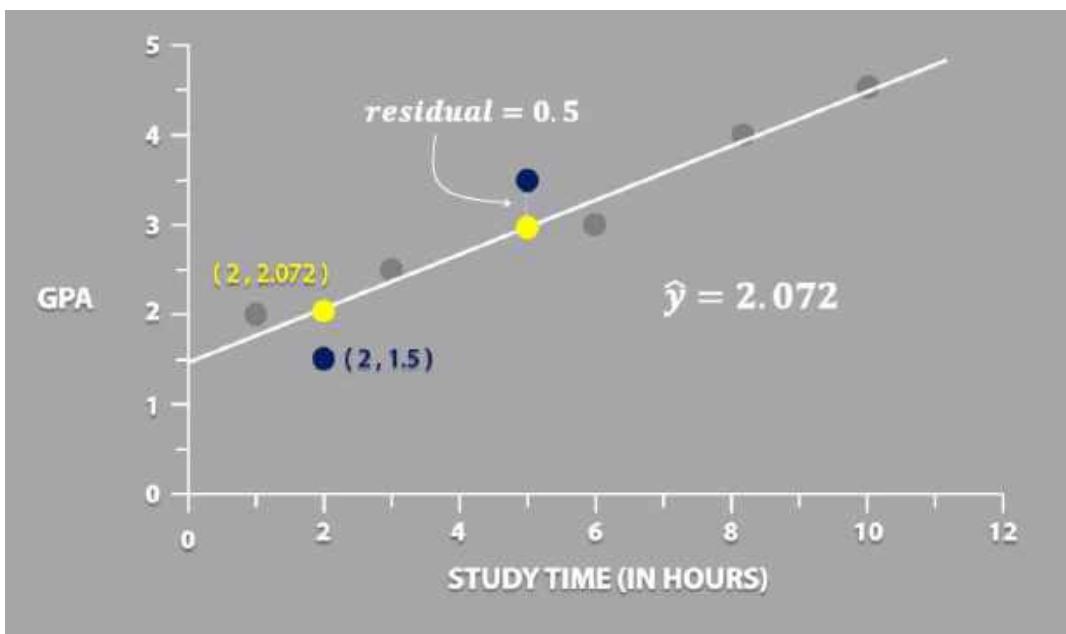
## Data Science – Maths – Part - 8



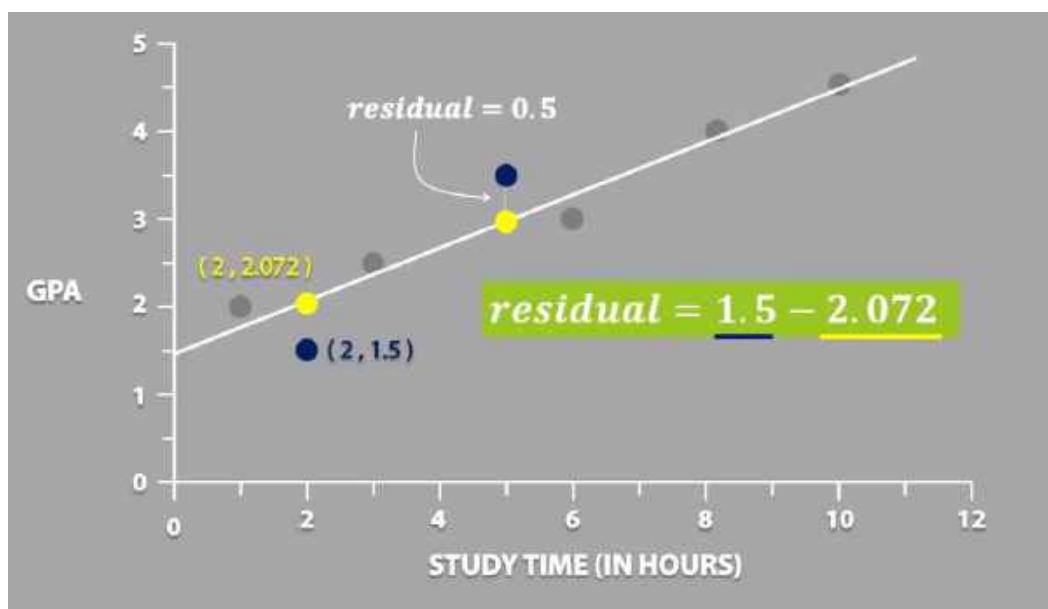
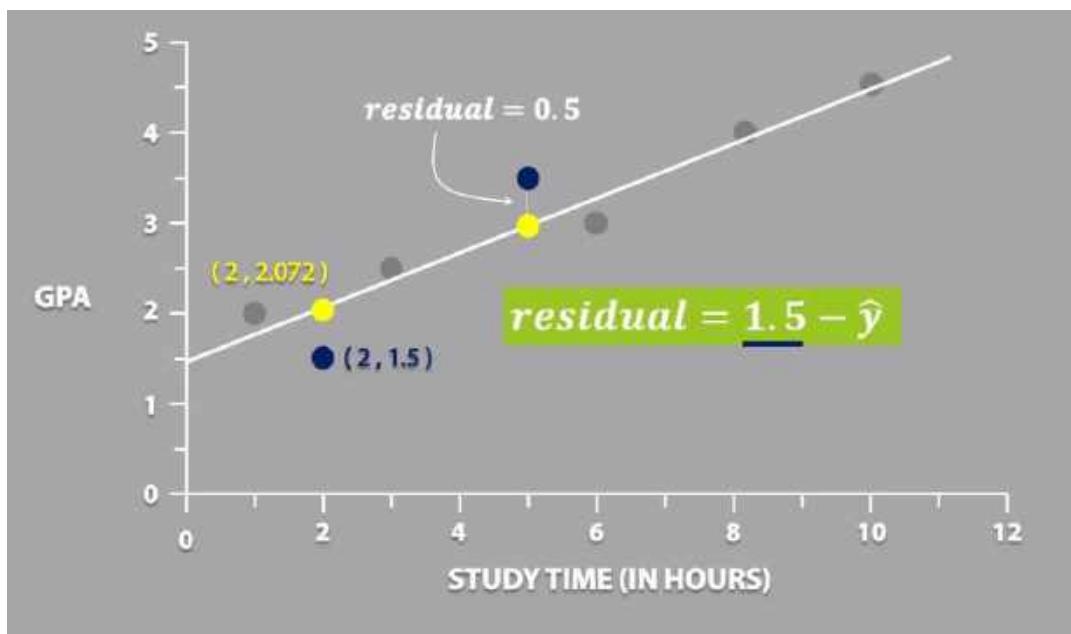
## Data Science – Maths – Part - 8



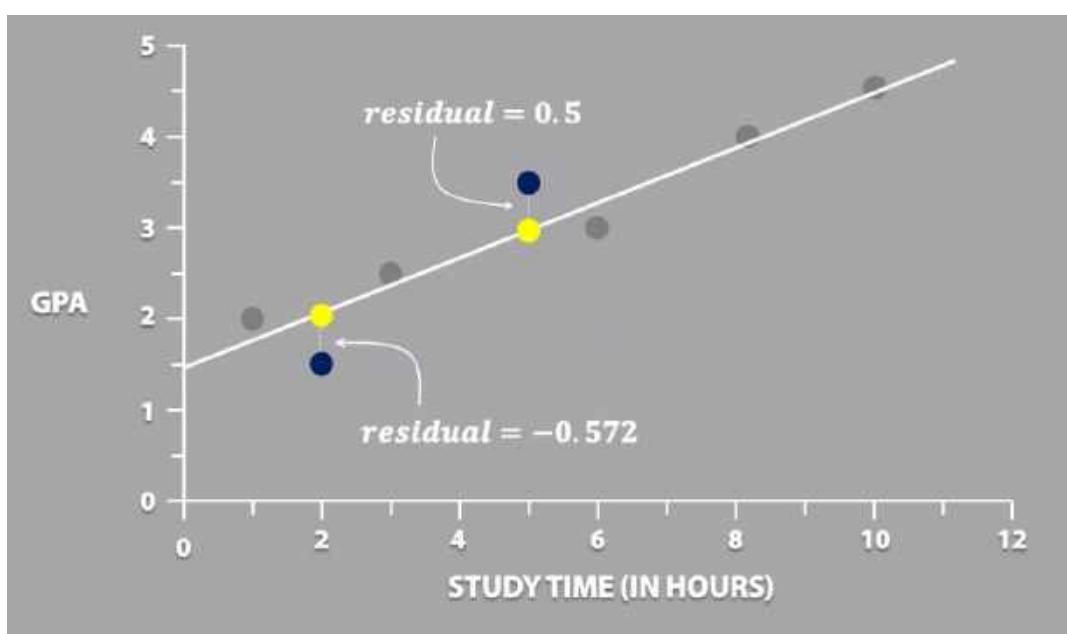
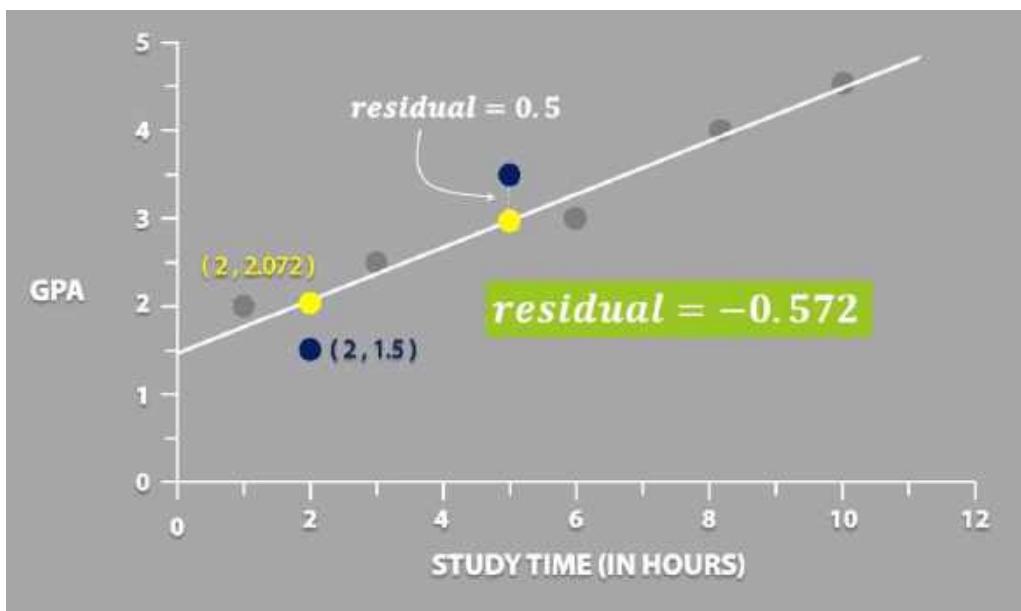
## Data Science – Maths – Part - 8

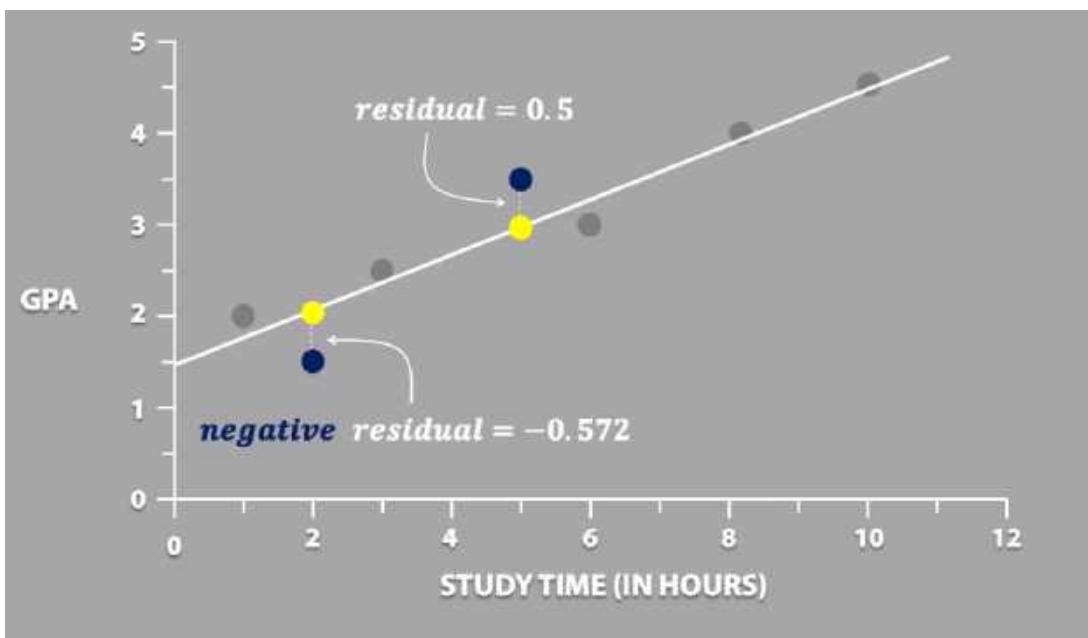


## Data Science – Maths – Part - 8



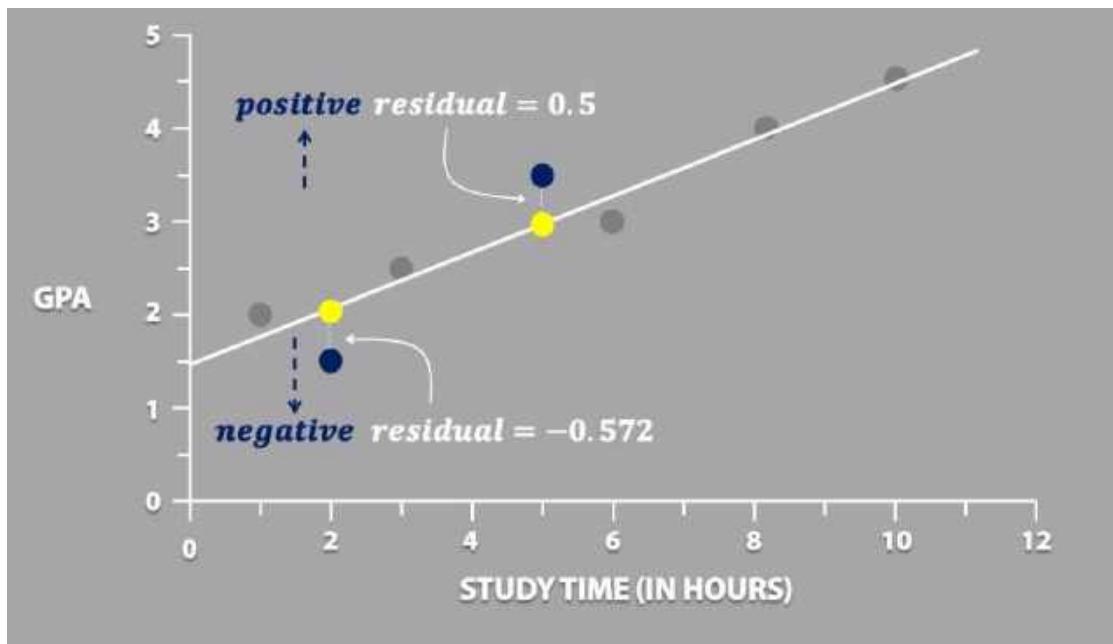
## Data Science – Maths – Part - 8





## Data Science – Maths – Part - 8

- ✓ A negative residual value means, it falls below the regression line
- ✓ A positive residual value means, the actual value located into above regression line



**10. Maths - Matrix****Contents**

<b>1. Matrix .....</b>	<b>2</b>
<b>2. Rows and columns.....</b>	<b>2</b>
<b>3. Scalar, Vector and Matrix.....</b>	<b>3</b>
<b>4. Adding matrix .....</b>	<b>4</b>
<b>5. Negative matrix .....</b>	<b>5</b>
<b>6. Subtracting matrix.....</b>	<b>6</b>
<b>7. Multiply by a Constant .....</b>	<b>7</b>
<b>8. Multiply Matrices .....</b>	<b>8</b>
<b>9. A real time example .....</b>	<b>10</b>
<b>10. Types of Matrix.....</b>	<b>13</b>
<b>11. Transpose matrix.....</b>	<b>14</b>

## 10. Maths - Matrix

### 1. Matrix

- ✓ A Matrix is an array of numbers:

$$\begin{bmatrix} 6 & 4 & 24 \\ 1 & -9 & 8 \end{bmatrix}$$

A Matrix

(This one has 2 Rows and 3 Columns)

### 2. Rows and columns

#### Rows and Columns

To show how many rows and columns a matrix has we often write **rows×columns**.

Example: This matrix is **2×3** (2 rows by 3 columns):

$$\begin{bmatrix} 6 & 4 & 24 \\ 1 & -9 & 8 \end{bmatrix}$$

### 3. Scalar, Vector and Matrix

#### Scalars, Vectors and Matrices

And when we include **matrices**, we get this interesting pattern:

- A **scalar** is a number, like 3, -5, 0.368, etc,
- A **vector** is a **list** of numbers (can be in a row or column),
- A **matrix** is an **array** of numbers (one or more rows, one or more columns).

Scalar	Vector	Matrix
24	$\begin{bmatrix} 2 & -8 & 7 \end{bmatrix}$ <small>row or column</small> $\begin{bmatrix} 2 \\ -8 \\ 7 \end{bmatrix}$	$\begin{bmatrix} 6 & 4 & 24 \\ 1 & -9 & 8 \end{bmatrix}$ <small>row(s) x column(s)</small>

In fact a **vector is also a matrix!** Because a matrix can have just one row or one column.

So the rules that work for matrices also work for vectors.

#### 4. Adding matrix

- ✓ We can add two matrices
- ✓ The two matrices must be the same size, i.e. the rows must match in size, and the columns must match in size.
- ✓ Example: a matrix with **3 rows and 5 columns** can be added to another matrix of **3 rows and 5 columns**.
- ✓ But it could not be added to a matrix with 3 rows and 4 columns (the columns don't match in size)

$$\begin{bmatrix} 3 & 8 \\ 4 & 6 \end{bmatrix} + \begin{bmatrix} 4 & 0 \\ 1 & -9 \end{bmatrix} = \begin{bmatrix} 7 & 8 \\ 5 & -3 \end{bmatrix}$$

3+4=7      8+0=8  
4+1=5      6-9=-3

These are the calculations:

## 5. Negative matrix

- ✓ The negative of a matrix is also simple:

$$- \begin{bmatrix} 2 & -4 \\ 7 & 10 \end{bmatrix} = \begin{bmatrix} -2 & 4 \\ -7 & -10 \end{bmatrix}$$

A diagram showing the calculation of the negative of a 2x2 matrix. A yellow circle contains a minus sign (-). A yellow arrow labeled  $-(2) = -2$  points from the minus sign to the first element of the matrix. Another yellow arrow labeled  $-(7) = -7$  points from the minus sign to the second element of the matrix.

These are the calculations:

$-(2) = -2$	$-(-4) = +4$
$-(7) = -7$	$-(10) = -10$

## 6. Subtracting matrix

- ✓ To subtract two matrices: subtract the numbers in the matching positions:
- ✓ Note: subtracting is actually defined as the addition of a negative matrix:  
 $A + (-B)$

$$\begin{bmatrix} 3 & 8 \\ 4 & 6 \end{bmatrix} - \begin{bmatrix} 4 & 0 \\ 1 & -9 \end{bmatrix} = \begin{bmatrix} -1 & 8 \\ 3 & 15 \end{bmatrix}$$

These are the calculations:

$$3-4=-1 \quad 8-0=8$$

$$4-1=3 \quad 6-(-9)=15$$

## 7. Multiply by a Constant

- ✓ We can multiply a matrix by a constant (*the value 2 in this case*):
- ✓ We call the constant a scalar, so officially this is called "scalar multiplication".

$$2 \times \begin{bmatrix} 4 & 0 \\ 1 & -9 \end{bmatrix} = \begin{bmatrix} 8 & 0 \\ 2 & -18 \end{bmatrix}$$

These are the calculations:

$$2 \times 4 = 8 \quad 2 \times 0 = 0$$

$$2 \times 1 = 2 \quad 2 \times -9 = -18$$

## Data Science – Maths – Part - 10

### 8. Multiply Matrices

- ✓ Multiplying a matrix by another matrix we need to do the "dot product" of rows and columns

To work out the answer for the **1st row** and **1st column**:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 \end{bmatrix}$$

"Dot Product"

The "Dot Product" is where we **multiply matching members**, then sum up:

$$(1, 2, 3) \bullet (7, 9, 11) = 1 \times 7 + 2 \times 9 + 3 \times 11 \\ = 58$$

We match the **1st** members (1 and 7), multiply them, likewise for the **2nd** members (2 and 9) and the **3rd** members (3 and 11), and finally sum them up.

Want to see another example? Here it is for the **1st row** and **2nd column**:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & 64 \end{bmatrix}$$

$$(1, 2, 3) \bullet (8, 10, 12) = 1 \times 8 + 2 \times 10 + 3 \times 12 \\ = 64$$

We can do the same thing for the **2nd row** and **1st column**:

$$(4, 5, 6) \bullet (7, 9, 11) = 4 \times 7 + 5 \times 9 + 6 \times 11 \\ = 139$$

And for the **2nd row** and **2nd column**:

$$(4, 5, 6) \bullet (8, 10, 12) = 4 \times 8 + 5 \times 10 + 6 \times 12 \\ = 154$$

## Data Science – Maths – Part - 10

And we get;

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & 64 \\ 139 & 154 \end{bmatrix} \checkmark$$

DONE!

## Data Science – Maths – Part - 10

### 9. A real time example

Example: The local shop sells 3 types of pies.

- Apple pies cost \$3 each
- Cherry pies cost \$4 each
- Blueberry pies cost \$2 each

And this is how many they sold in 4 days:

	Mon	Tue	Wed	Thu
Apple	13	9	7	15
Cherry	8	7	4	6
Blueberry	6	4	0	3

Now think about this ... the **value of sales** for Monday is calculated this way:

$$\begin{aligned} &\rightarrow \text{Apple pie value} + \text{Cherry pie value} + \text{Blueberry pie value} \\ &\rightarrow \$3 \times 13 + \$4 \times 8 + \$2 \times 6 = \$83 \end{aligned}$$

So it is, in fact, the "dot product" of prices and how many were sold:

$$\begin{aligned} (\$3, \$4, \$2) \cdot (13, 8, 6) &= \$3 \times 13 + \$4 \times 8 + \$2 \times 6 \\ &= \$83 \end{aligned}$$

We **match** the price to how many sold, **multiply** each, then **sum** the result.

## Data Science – Maths – Part - 10

In other words:

- The sales for Monday were: Apple pies:  $\$3 \times 13 = \$39$ , Cherry pies:  $\$4 \times 8 = \$32$ , and Blueberry pies:  $\$2 \times 6 = \$12$ . Together that is  $\$39 + \$32 + \$12 = \$83$
- And for Tuesday:  $\$3 \times 9 + \$4 \times 7 + \$2 \times 4 = \$63$
- And for Wednesday:  $\$3 \times 7 + \$4 \times 4 + \$2 \times 0 = \$37$
- And for Thursday:  $\$3 \times 15 + \$4 \times 6 + \$2 \times 3 = \$75$

So it is important to match each price to each quantity.

Now you know why we use the "dot product".

And here is the full result in Matrix form:

$$\begin{bmatrix} \$3 & \$4 & \$2 \end{bmatrix} \times \begin{bmatrix} 13 & 9 & 7 & 15 \\ 8 & 7 & 4 & 6 \\ 6 & 4 & 0 & 3 \end{bmatrix} = \begin{bmatrix} \$83 & \$63 & \$37 & \$75 \end{bmatrix}$$

$\$3 \times 13 + \$4 \times 8 + \$2 \times 6$

They sold  **$\$83$**  worth of pies on Monday,  **$\$63$**  on Tuesday, etc.

## Examples

*In General:*

To multiply an  $m \times n$  matrix by an  $n \times p$  matrix, the  $n$ s must be the same, and the result is an  $m \times p$  matrix.

$$m \times n \times n \times p \rightarrow m \times p$$

So ... multiplying a  $1 \times 3$  by a  $3 \times 1$  gets a  $1 \times 1$  result:

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} = [1 \times 4 + 2 \times 5 + 3 \times 6] = [32]$$

But multiplying a  $3 \times 1$  by a  $1 \times 3$  gets a  $3 \times 3$  result:

$$\begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 4 \times 1 & 4 \times 2 & 4 \times 3 \\ 5 \times 1 & 5 \times 2 & 5 \times 3 \\ 6 \times 1 & 6 \times 2 & 6 \times 3 \end{bmatrix} = \begin{bmatrix} 4 & 8 & 12 \\ 5 & 10 & 15 \\ 6 & 12 & 18 \end{bmatrix}$$

## 10. Types of Matrix

A **Matrix** is an array of numbers:

$$\begin{bmatrix} 6 & 4 & 24 \\ 1 & -9 & 8 \end{bmatrix}$$

A Matrix

(This one has 2 Rows and 3 Columns)

The **Main Diagonal** starts at the top left and goes down to the right:

$$\begin{bmatrix} 7 & 6 & 4 \\ 4 & 2 & -2 \\ 3 & 0 & 9 \end{bmatrix}$$

## 11. Transpose matrix

- ✓ To "transpose" a matrix, swap the rows and columns.
- ✓ We put a "T" in the top right-hand corner to mean transpose:

$$\begin{bmatrix} 6 & 4 & 24 \\ 1 & -9 & 8 \end{bmatrix}^T = \begin{bmatrix} 6 & 1 \\ 4 & -9 \\ 24 & 8 \end{bmatrix}$$

## Square

A **square** matrix has the same number of rows as columns.

$$\begin{bmatrix} 2 & 0 \\ 1 & 8 \end{bmatrix}$$

A square matrix (2 rows, 2 columns)

$$\begin{bmatrix} 6 & 4 & 24 \\ 1 & -9 & 8 \\ 3 & 0 & 7 \end{bmatrix}$$

Also a square matrix (3 rows, 3 columns)

## Identity Matrix

An **Identity Matrix** has **1s** on the main diagonal and **0s** everywhere else:

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

A  $3 \times 3$  Identity Matrix

- It is square (same number of rows as columns)
- It can be large or small ( $2 \times 2$ ,  $100 \times 100$ , ... whatever)
- Its symbol is the capital letter **I**

It is the matrix equivalent of the number "1", when we multiply with it the original is unchanged:

$$\mathbf{A} \times \mathbf{I} = \mathbf{A}$$

$$\mathbf{I} \times \mathbf{A} = \mathbf{A}$$

## Diagonal Matrix

A diagonal matrix has zero anywhere not on the main diagonal:

$$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

A diagonal matrix

### Scalar Matrix

A scalar matrix has all main diagonal entries the same, with zero everywhere else:

$$\begin{bmatrix} 5 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 5 \end{bmatrix}$$

A scalar matrix

### Zero Matrix (Null Matrix)

Zeros just everywhere:

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Zero matrix

**1. Feature Engineering****Contents**

<b>1. Feature scaling.....</b>	2
<b>2. Data pre-processing .....</b>	2
<b>3. Why Data Pre-processing in Machine Learning?.....</b>	2
<b>4. What type of data we need to handle?.....</b>	2
<b>5. Numerical data .....</b>	3
<b>6. To handle Numerical data.....</b>	4
<b>7. To handle Categorical data .....</b>	5
<b>8. scikit-learn installation.....</b>	6
<b>9. Label Encoder .....</b>	7
9.1. LabelEncoder class .....	7
9.2. fit_transform(p) method.....	7
<b>10. MinMaxScaler.....</b>	9
<b>11. Transforming Features .....</b>	13
<b>12. Handling outlier.....</b>	14
<b>13. Impute Missing values .....</b>	18
<b>14. Impute missing numeric values .....</b>	19

### 1. Feature Engineering

#### 1. Feature scaling

- ✓ Feature Scaling is a technique to standardize the independent features present in the data in a fixed range.
- ✓ It is performed during the data pre-processing.

#### 2. Data pre-processing

- ✓ In machine learning data pre-processing is an important step
- ✓ The purpose of this technique is cleaning and organizing the raw data to make it suitable for building and training machine learning models

#### 3. Why Data Pre-processing in Machine Learning?

- ✓ Typically, real-world data is incomplete, inconsistent, inaccurate (contains errors or outliers), and often lacks specific attribute values/trends.
- ✓ This is where data pre-processing enters the scenario – it helps to **clean**, **format**, and **organize** the raw data, thereby making it ready to use the data for Machine Learning models.

#### 4. What type of data we need to handle?

- ✓ Two types of data we need to handle
  - Numerical data
  - Categorical data

### 5. Numerical data

- ✓ Quantitative data is the measurement of something monthly sales, or student scores etc.
- ✓ The natural way to represent these quantities is numerically (e.g., 29 students, \$529,392 in sales).
- ✓ So we need to understand how to transforming raw numerical data into features, then we need to use this feature during machine learning

## 6. To handle Numerical data

Technique	Purpose
✓ LabelEncoder	✓ To convert all character/categorical variables to be numeric.
✓ StandardScaler	✓ To transform a feature which is <b>mean to 0</b> and <b>standard deviation to 1</b>
✓ Transforming Features	✓ We can transform features
✓ Handling outlier	✓ To handle outlier
✓ Impute Missing Values	✓ To impute missing value with strategy

## 7. To handle Categorical data

Technique	Purpose
✓ Encoding nominal categories	✓ To do one hot encoding
✓ Encoding ordinal categories	✓ Ordinal categorical
✓ Imputing categorical missing values	✓ Imputing categorical missing values with most frequent strategy

### 8. scikit-learn installation

- ✓ To execute all these examples we need to install scikit-learn library.

```
pip install scikit-learn
```

## 9. Label Encoder

- ✓ By using this we can convert all character/categorical variables to be numeric.

### 9.1. LabelEncoder class

- ✓ **LabelEncoder** is predefined class in **sklearn.preprocessing** package
- ✓ We need to import this class from **sklearn.preprocessing** package
- ✓ Once we imported then we need to **create an object** o **LabelEncoder** class.

### 9.2. fit\_transform(p) method

- ✓ **fit\_transform(p)** is predefined method in **LabelEncoder** class.
- ✓ We should access this method by using **LabelEncoder** object only.
- ✓ This method converts categorical variables into numerical values.

**Program**

LabelEncoder

**Name**

demo1.py

```
from sklearn.preprocessing import LabelEncoder
import pandas as pd

d = {
    "Company": ["Google", "Twitter", "Google", "LinkedIn"],
    "Role": ["Data Scientist", "Sales manager", "HR", "HR"]
}

df = pd.DataFrame(d)
label_encoder = LabelEncoder()

df["Company_n"] = label_encoder.fit_transform(df['Company'])
df["Role_n"] = label_encoder.fit_transform(df['Role'])

print()
print(df)
```

**Output**

	Company	Role	Company_n	Role_n
0	Google	Data Scientist	0	0
1	Twitter	Sales manager	2	2
2	Google		0	1
3	LinkedIn		1	1

## 10. MinMaxScaler

**MIN MAX SCALING**

Rescales feature values to between 0 and 1

Original value      Minimum value in feature

Rescaled value      Maximum value in feature

$$X'_i = \frac{X_i - \min(x)}{\max(x) - \min(x)}$$

- ✓ Min-max scaling is a common feature pre-processing technique which results in scaled data values that fall in the range 0 and 1.
  - 0 is minimum value
  - 1 is maximum value
  
- ✓ If we **rescale** the value of numeric feature then it is in between two values.

Program Name MinMaxScaler  
demo2.py

```
import pandas as pd

d = {
    "x" : [10, 20, 30, 40, 50],
    "y" : [25, 50, 75, 100, 125]
}

df = pd.DataFrame(d)
print(df)
```

Output

	x	y
0	10	25
1	20	50
2	30	75
3	40	100
4	50	125

Program Name MinMaxScaler: single column  
demo3.py

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

d = {
    "x" : [10, 20, 30, 40, 50],
    "y" : [25, 50, 75, 100, 125]
}

df = pd.DataFrame(d)
mm_scale = MinMaxScaler(feature_range = (0, 1))

print(df)

one_col = mm_scale.fit_transform(df[["x"]])

print(one_col)
```

Output

```
x      y
0  10   25
1  20   50
2  30   75
3  40  100
4  50  125

[[0.  ]
 [0.25]
 [0.5 ]
 [0.75]
 [1.  ]]
```

Program Name    MinMaxScaler: two columns  
demo4.py

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

d = {
    "x" : [10, 20, 30, 40, 50],
    "y" : [25, 50, 75, 100, 125]
}

df = pd.DataFrame(d)
mm_scale = MinMaxScaler(feature_range = (0, 1))

print(df)

df[["x", "y"]] = mm_scale.fit_transform(df[["x", "y"]])

print(df)
```

Output

	x	y
0	10	25
1	20	50
2	30	75
3	40	100
4	50	125

	x	y
0	0.00	0.00
1	0.25	0.25
2	0.50	0.50
3	0.75	0.75
4	1.00	1.00

## 11. Transforming Features

- ✓ By using **FunctionTransformer** we can transform the features.

Program      FunctionTransformer  
Name          demo5.py

```
import numpy as np
from sklearn.preprocessing import FunctionTransformer

a = [[10, 20], [30, 40], [50, 60]]

f = np.array(a)

def add_ten(x):
    return x + 10

obj = FunctionTransformer(add_ten)

result = obj.transform(f)

print(f)
print()
print(result)
```

### Output

```
[[10 20]
 [30 40]
 [50 60]]

[[20 30]
 [40 50]
 [60 70]]
```

## 12. Handling outlier

# HANDLING OUTLIERS

1. If due to an error: drop, mark as missing value, mark as possible error.
2. If a legitimate but extreme value: decide if it is genuinely a member of the population we are trying to address with our model.

- ✓ Outlier means a large value compared with other values
- ✓ Some values are also out of the range of the feature, so they are also considered as outliers.
- ✓ Outliers affect our model's efficiency because it influences the model very much.
- ✓ Three ways to handle these,
  - Drop outlier or filter outlier
  - Marking them using boolean condition
  - Transform into feature

## Data Science – Feature Engineering

**Program Name** A dataframe with outliers  
demo6.py

```
import numpy as np
import pandas as pd

houses = pd.DataFrame()

houses['Price'] = [534433, 392333, 293222, 4322032]
houses['rooms'] = [2, 3, 2, 116]
houses['Square_Feet'] = [1500, 2500, 1500, 48000]

print(houses)
```

**Output**

	Price	rooms	Square_Feet
0	534433	2	1500
1	392333	3	2500
2	293222	2	1500
3	4322032	116	48000

## Data Science – Feature Engineering

**Program Name** Filtering outlier  
demo7.py

```
import numpy as np
import pandas as pd

houses = pd.DataFrame()

houses['Price'] = [534433, 392333, 293222, 4322032]
houses['rooms'] = [2, 3, 2, 116]
houses['Square_Feet'] = [1500, 2500, 1500, 48000]

con1 = houses['rooms'] < 20
new = houses[con1]

print(new)
```

**Output**

	Price	rooms	Square_Feet
0	534433	2	1500
1	392333	3	2500
2	293222	2	1500

**Program Name** Mark them as outliers  
demo8.py

```
import numpy as np
import pandas as pd

houses = pd.DataFrame()

houses['Price'] = [534433, 392333, 293222, 4322032]
houses['rooms'] = [2, 3, 2, 116]
houses['Square_Feet'] = [1500, 2500, 1500, 48000]

houses["Outlier"] = np.where(houses["rooms"] < 20, 0, 1)

print(houses)
```

**Output**

	Price	rooms	Square_Feet	Outlier
0	534433	2	1500	0
1	392333	3	2500	0
2	293222	2	1500	0
3	4322032	116	48000	1

### 13. Impute Missing values

- ✓ We can impute missing values with different strategy.
- ✓ There are two types of missing values
  - Numeric missing values
  - Categorical missing values

## IMPUTING MISSING VALUES

1. If quantitative, replace with an average value.
2. If qualitative, replace with most common value.
3. Use a model to predict the missing values. For example, K-nearest neighbors.

ChrisAlben

### 14. Impute missing numeric values

- ✓ Missing numeric values we can impute with different strategy
  - mean
  - median
  - most\_frequent
  - constant

## Data Science – Feature Engineering

**Program Name** Creating DataFrame  
demo9.py

```
import numpy as np
import pandas as pd

students = [
    [85, 'M', 'verygood'],
    [95, 'F', 'excellent'],
    [60, None, 'good'],
    [np.NaN, 'M', 'average'],
    [70, 'M', 'good'],
    [np.NaN, None, 'verygood'],
    [60, 'F', 'verygood'],
    [98, 'M', 'excellent']
]

cols = ['marks', 'gender', 'result']
df = pd.DataFrame(students, columns = cols)
print(df)
```

**output**

	marks	gender	result
0	85.0	M	verygood
1	95.0	F	excellent
2	60.0	None	good
3	NaN	M	average
4	70.0	M	good
5	NaN	None	verygood
6	60.0	F	verygood
7	98.0	M	excellent

## Data Science – Feature Engineering

**Program Name** Imputing missing numeric values with mean strategy  
demo10.py

```
import numpy as np
from sklearn.impute import SimpleImputer
import pandas as pd

students = [
    [85, 'M', 'verygood'],
    [95, 'F', 'excellent'],
    [60, None, 'good'],
    [np.NaN, 'M', 'average'],
    [70, 'M', 'good'],
    [np.NaN, None, 'verygood'],
    [60, 'F', 'verygood'],
    [98, 'M', 'excellent']
]

cols = ['marks', 'gender', 'result']
df = pd.DataFrame(students, columns = cols)

print(df)

imputer = SimpleImputer(missing_values = np.nan, strategy =
'mean')

result = df['marks'].values.reshape(-1, 1)

df.marks = imputer.fit_transform(result)

print()
print(df)
```

output

	marks	gender	result
0	85.0	M	verygood
1	95.0	F	excellent
2	60.0	None	good
3	NaN	M	average
4	70.0	M	good
5	NaN	None	verygood
6	60.0	F	verygood
7	98.0	M	excellent

	marks	gender	result
0	85.0	M	verygood
1	95.0	F	excellent
2	60.0	None	good
3	78.0	M	average
4	70.0	M	good
5	78.0	None	verygood
6	60.0	F	verygood
7	98.0	M	excellent

## Data Science – Feature Engineering

**Program Name** Imputing missing numeric values with median strategy  
demo11.py

```
import numpy as np
from sklearn.impute import SimpleImputer
import pandas as pd

students = [
    [85, 'M', 'verygood'],
    [95, 'F', 'excellent'],
    [60, None, 'good'],
    [np.NaN, 'M', 'average'],
    [70, 'M', 'good'],
    [np.NaN, None, 'verygood'],
    [60, 'F', 'verygood']
]

cols = ['marks', 'gender', 'result']
df = pd.DataFrame(students, columns = cols)

print(df)

imputer = SimpleImputer(missing_values = np.nan, strategy = 'median')

result = df['marks'].values.reshape(-1, 1)

df.marks = imputer.fit_transform(result)

print()
print(df)
```

output

	marks	gender	result
0	85.0	M	verygood
1	95.0	F	excellent
2	60.0	None	good
3	NaN	M	average
4	70.0	M	good
5	NaN	None	verygood
6	60.0	F	verygood

	marks	gender	result
0	85.0	M	verygood
1	95.0	F	excellent
2	60.0	None	good
3	70.0	M	average
4	70.0	M	good
5	70.0	None	verygood
6	60.0	F	verygood

## Data Science – Feature Engineering

**Program Name** Imputing missing numeric values with `most_frequent` strategy  
`demo12.py`

```
import numpy as np
from sklearn.impute import SimpleImputer
import pandas as pd

students = [
    [85, 'M', 'verygood'],
    [95, 'F', 'excellent'],
    [60, None, 'good'],
    [np.NaN, 'M', 'average'],
    [70, 'M', 'good'],
    [np.NaN, None, 'verygood'],
    [60, 'F', 'verygood'],
    [98, 'M', 'excellent']
]

cols = ['marks', 'gender', 'result']
df = pd.DataFrame(students, columns = cols)

imputer = SimpleImputer(missing_values = np.nan, strategy = 'most_frequent')

result = df['marks'].values.reshape(-1, 1)

df.marks = imputer.fit_transform(result)

print(df)
```

output

	marks	gender	result
0	85.0	M	verygood
1	95.0	F	excellent
2	60.0	None	good
3	60.0	M	average
4	70.0	M	good
5	60.0	None	verygood
6	60.0	F	verygood
7	98.0	M	excellent

## Data Science – Feature Engineering

**Program Name** Imputing missing numeric values with constant strategy  
demo13.py

```
import numpy as np
from sklearn.impute import SimpleImputer
import pandas as pd

students = [
    [85, 'M', 'verygood'],
    [95, 'F', 'excellent'],
    [60, None, 'good'],
    [np.NaN, 'M', 'average'],
    [70, 'M', 'good'],
    [np.NaN, None, 'verygood'],
    [60, 'F', 'verygood'],
    [98, 'M', 'excellent']
]

cols = ['marks', 'gender', 'result']
df = pd.DataFrame(students, columns = cols)

print(df)

imputer = SimpleImputer(missing_values = np.nan, strategy =
'constant', fill_value = 80)

result = df['marks'].values.reshape(-1, 1)

df.marks = imputer.fit_transform(result)

print()
print(df)
```

output

	marks	gender	result
0	85.0	M	verygood
1	95.0	F	excellent
2	60.0	None	good
3	NaN	M	average
4	70.0	M	good
5	NaN	None	verygood
6	60.0	F	verygood
7	98.0	M	excellent

	marks	gender	result
0	85.0	M	verygood
1	95.0	F	excellent
2	60.0	None	good
3	80.0	M	average
4	70.0	M	good
5	80.0	None	verygood
6	60.0	F	verygood
7	98.0	M	excellent

# Data Science – Feature Engineering

---

## 2. Feature Engineering

### Contents

<b>1. Handling Categorical Data.....</b>	<b>2</b>
<b>2. Encoding Categorical Data .....</b>	<b>3</b>
2.1. Ordinal encoding.....	4
2.2. One hot encoding.....	6
2.3. Dummy variable encoding .....	9
2.4. Imputing Missing Class Values .....	11

## 2. Feature Engineering

### 1. Handling Categorical Data

#### **Categorical data**

- ✓ Categorical data are variables that contain label values rather than numeric values.

#### **Types of categorical data**

- ✓ Nominal Variable
- ✓ Ordinal Variable

#### **Nominal Variable**

- ✓ The variables which are having **no-order** those are called as **Nominal Variable**.
- ✓ Examples:

- Pet variables values : cat, dog
- Color variables values : blue, green, red

#### **Ordinal Variable**

- ✓ The variables which are having an **order** those are called as **ordinal Variable**.
  - ✓ Examples:
- 
- Score variables values : low, medium, high

#### **Kind note**

- ✓ In real time mostly we do have nominal variable scenarios.
- ✓ So, please understand the below scenarios

### 2. Encoding Categorical Data

- ✓ There are 3 ways to convert categorical variables to numerical values.
  - Ordinal encoding
  - One hot encoding
  - Dummy variable encoding

## 2.1. Ordinal encoding

✓ In ordinal encoding every nominal value is assigned an integer value.

✓ Example

- blue : 0
- green : 1
- red : 2

Program Name      Ordinal encoding  
demo1.py

```
from numpy import asarray
from sklearn.preprocessing import OrdinalEncoder

data = asarray([['blue'], ['green'], ['red']])

encoder = OrdinalEncoder()
result = encoder.fit_transform(data)

print(data)
print(result)
```

Output

```
[['blue']
 ['green']
 ['red']]
```

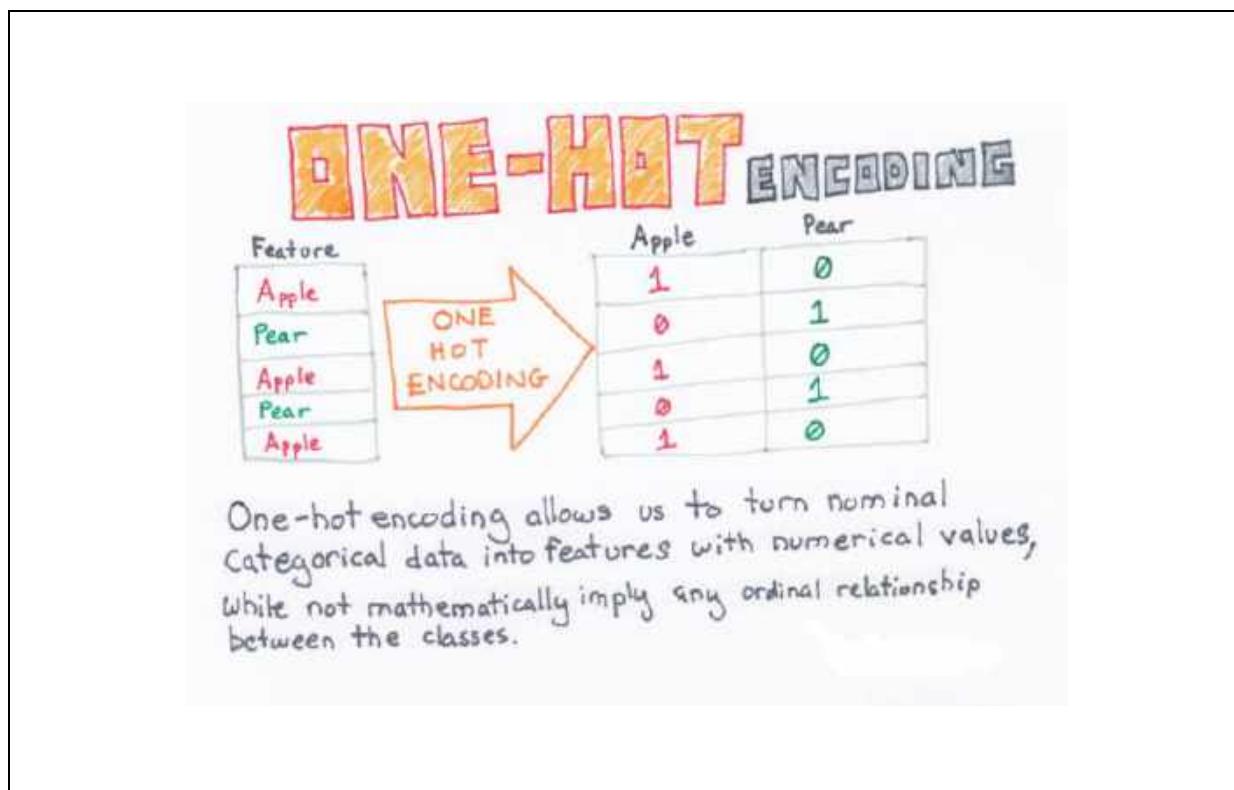
```
[[0.]
 [1.]
 [2.]]
```

### Problem with ordinal encoding

- ✓ If we have applied ordinal encoding on nominal values then it will be an order and having relationship but actually there is no relationship in between the nominal variables.
- ✓ Machine learning algorithm understands like there is an order in between nominal values.
- ✓ So it causes a problem like machine learning algorithm will produce poor performance.
- ✓ We can solve this problem by using **one hot encoding**.

## 2.2. One hot encoding

- ✓ For **nominal** values integer encoding may not be enough and even it is misleading the model.
- ✓ Here one hot encoding helps, it is technique where each of the nominal variables will be represented with binary values.



- ✓ Example

- blue :      1      0      0
- green :     0      1      0
- red :       0      0      1

## Data Science – Feature Engineering

Program Name One hot encoding  
demo2.py

```
from numpy import asarray
from sklearn.preprocessing import OneHotEncoder

a = [['apple'], ['peer'], ['apple'], ['peer'], ['apple']]
data = asarray(a)

encoder = OneHotEncoder(sparse = False)
onehot = encoder.fit_transform(data)

print(data)
print()
print(onehot)
```

output

```
[[ 'apple' ]
 [ 'peer' ]
 [ 'apple' ]
 [ 'peer' ]
 [ 'apple' ]]

[[1. 0.]
 [0. 1.]
 [1. 0.]
 [0. 1.]
 [1. 0.]]
```

Program Name One hot encoding  
demo3.py

```
from numpy import asarray
from sklearn.preprocessing import OneHotEncoder

data = asarray([['blue'], ['green'], ['red']])
encoder = OneHotEncoder(sparse = False)
onehot = encoder.fit_transform(data)

print(data)
print(onehot)
```

Output

```
[['blue']
 ['green']
 ['red']]

[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

### 2.3. Dummy variable encoding

- ✓ The one hot encoding creates one binary variable for each category.
- ✓ The problem is that this representation includes **redundancy**.
- ✓ For example, if we know that [1, 0, 0] represents for **first value** and [0, 1, 0] represents for **second value** then we don't need another binary variable to represent **third value**, instead we could use 0 values alone like [0, 0].

#### One hot encoding example

- ✓ Example

- blue : 1 0 0
- green : 0 1 0
- red : 0 0 1

#### Dummy variable encoding example

- ✓ Example

- blue : 0 0
- green : 1 0
- red : 0 1

#### Conclusion

- ✓ If we drop first column from the result of one hot encoding then we will get dummy variable encoding

## Data Science – Feature Engineering

Program

Name demo4.py

```
from numpy import asarray
from sklearn.preprocessing import OneHotEncoder

data = asarray([['blue'], ['green'], ['red']])
encoder = OneHotEncoder(drop = 'first', sparse = False)

onehot = encoder.fit_transform(data)

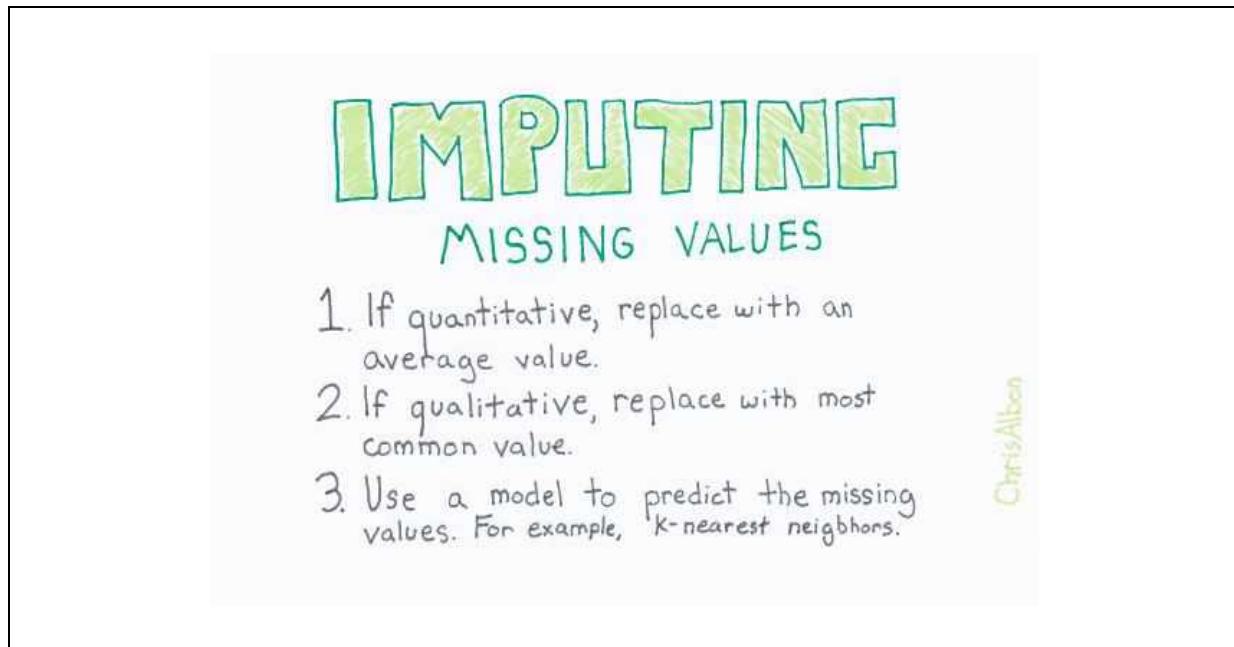
print(data)
print(onehot)
```

Output

```
[['blue']
 ['green']
 ['red']]

[[0. 0.]
 [1. 0.]
 [0. 1.]]
```

## 2.4. Imputing Missing Class Values



- ✓ Categorical feature may have missing values
- ✓ These we can impute with most frequent strategy

## Data Science – Feature Engineering

**Program Name** Imputing categorical values with most frequent strategy  
demo5.py

```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer

students = [
    [85, 'M', 'verygood'],
    [95, 'F', 'excellent'],
    [75, np.NaN, 'good'],
    [np.NaN, 'M', 'average'],
    [70, 'M', 'good'],
    [np.NaN, np.NaN, 'verygood'],
    [92, 'F', 'verygood'],
    [98, 'M', 'excellent']
]

cols = ['marks', 'gender', 'result']
df = pd.DataFrame(students, columns = cols)

print(df)

imputer = SimpleImputer(missing_values = np.NaN,
strategy='most_frequent')

result = df['gender'].values.reshape(-1, 1)

df.gender = imputer.fit_transform(result)

print()
print(df)
```

output

```
    marks gender      result
0    85.0      M  verygood
1    95.0      F  excellent
2    75.0     NaN      good
3    NaN      M   average
4    70.0      M      good
5    NaN     NaN  verygood
6    92.0      F  verygood
7    98.0      M  excellent
```

```
    marks gender      result
0    85.0      M  verygood
1    95.0      F  excellent
2    75.0      M      good
3    NaN      M   average
4    70.0      M      good
5    NaN      M  verygood
6    92.0      F  verygood
7    98.0      M  excellent
```

# Data Science – Feature Engineering

---

## 2. Feature Engineering

### Contents

<b>1. Handling Categorical Data.....</b>	<b>2</b>
<b>2. Encoding Categorical Data .....</b>	<b>3</b>
2.1. Ordinal encoding.....	4
2.2. One hot encoding.....	6
2.3. Dummy variable encoding .....	9
2.4. Imputing Missing Class Values .....	11

## 2. Feature Engineering

### 1. Handling Categorical Data

#### **Categorical data**

- ✓ Categorical data are variables that contain label values rather than numeric values.

#### **Types of categorical data**

- ✓ Nominal Variable
- ✓ Ordinal Variable

#### **Nominal Variable**

- ✓ The variables which are having **no-order** those are called as **Nominal Variable**.
- ✓ Examples:

- Pet variables values : cat, dog
- Color variables values : blue, green, red

#### **Ordinal Variable**

- ✓ The variables which are having an **order** those are called as **ordinal Variable**.
  - ✓ Examples:
- 
- Score variables values : low, medium, high

#### **Kind note**

- ✓ In real time mostly we do have nominal variable scenarios.
- ✓ So, please understand the below scenarios

### 2. Encoding Categorical Data

- ✓ There are 3 ways to convert categorical variables to numerical values.
  - Ordinal encoding
  - One hot encoding
  - Dummy variable encoding

## 2.1. Ordinal encoding

✓ In ordinal encoding every nominal value is assigned an integer value.

✓ Example

- blue : 0
- green : 1
- red : 2

Program Name      Ordinal encoding  
demo1.py

```
from numpy import asarray
from sklearn.preprocessing import OrdinalEncoder

data = asarray([['blue'], ['green'], ['red']])

encoder = OrdinalEncoder()
result = encoder.fit_transform(data)

print(data)
print(result)
```

Output

```
[['blue']
 ['green']
 ['red']]
```

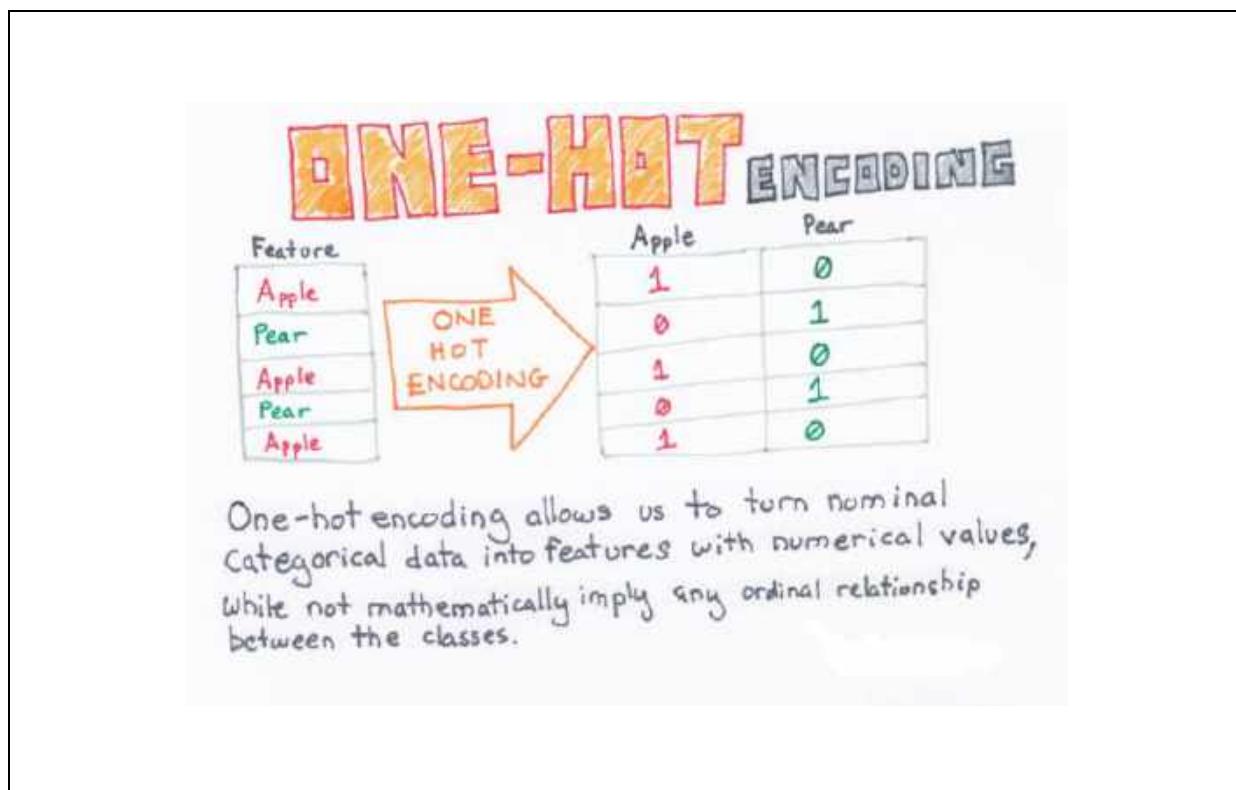
```
[[0.]
 [1.]
 [2.]]
```

### Problem with ordinal encoding

- ✓ If we have applied ordinal encoding on nominal values then it will be an order and having relationship but actually there is no relationship in between the nominal variables.
- ✓ Machine learning algorithm understands like there is an order in between nominal values.
- ✓ So it causes a problem like machine learning algorithm will produce poor performance.
- ✓ We can solve this problem by using **one hot encoding**.

## 2.2. One hot encoding

- ✓ For **nominal** values integer encoding may not be enough and even it is misleading the model.
- ✓ Here one hot encoding helps, it is technique where each of the nominal variables will be represented with binary values.



- ✓ Example

- blue :      1      0      0
- green :     0      1      0
- red :       0      0      1

Program Name One hot encoding  
demo2.py

```
from numpy import asarray
from sklearn.preprocessing import OneHotEncoder

a = [['apple'], ['peer'], ['apple'], ['peer'], ['apple']]
data = asarray(a)

encoder = OneHotEncoder(sparse = False)
onehot = encoder.fit_transform(data)

print(data)
print()
print(onehot)
```

output

```
[[ 'apple' ]
 [ 'peer' ]
 [ 'apple' ]
 [ 'peer' ]
 [ 'apple' ]]

[[1. 0.]
 [0. 1.]
 [1. 0.]
 [0. 1.]
 [1. 0.]]
```

Program Name One hot encoding  
demo3.py

```
from numpy import asarray
from sklearn.preprocessing import OneHotEncoder

data = asarray([['blue'], ['green'], ['red']])
encoder = OneHotEncoder(sparse = False)
onehot = encoder.fit_transform(data)

print(data)
print(onehot)
```

Output

```
[['blue']
 ['green']
 ['red']]

[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

### 2.3. Dummy variable encoding

- ✓ The one hot encoding creates one binary variable for each category.
- ✓ The problem is that this representation includes **redundancy**.
- ✓ For example, if we know that [1, 0, 0] represents for **first value** and [0, 1, 0] represents for **second value** then we don't need another binary variable to represent **third value**, instead we could use 0 values alone like [0, 0].

#### One hot encoding example

- ✓ Example

- blue : 1 0 0
- green : 0 1 0
- red : 0 0 1

#### Dummy variable encoding example

- ✓ Example

- blue : 0 0
- green : 1 0
- red : 0 1

#### Conclusion

- ✓ If we drop first column from the result of one hot encoding then we will get dummy variable encoding

**Program****Name** demo4.py

```
from numpy import asarray
from sklearn.preprocessing import OneHotEncoder

data = asarray([['blue'], ['green'], ['red']])
encoder = OneHotEncoder(drop = 'first', sparse = False)

onehot = encoder.fit_transform(data)

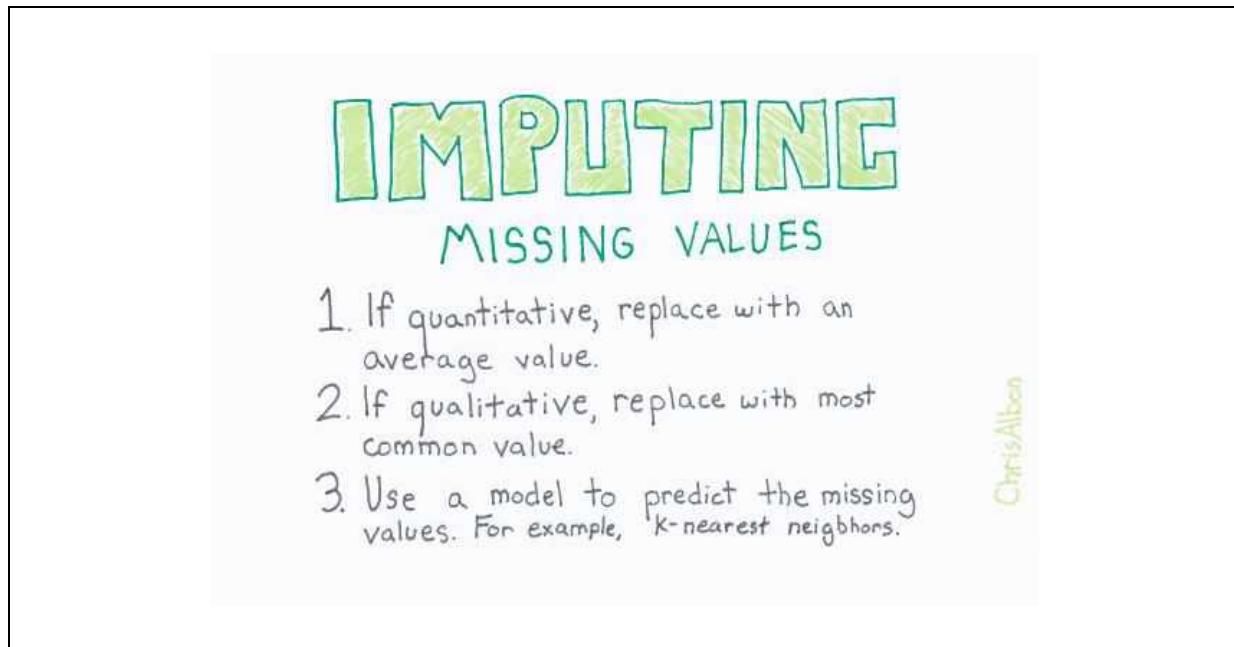
print(data)
print(onehot)
```

**Output**

```
[['blue']
 ['green']
 ['red']]
```

```
[[0. 0.]
 [1. 0.]
 [0. 1.]]
```

## 2.4. Imputing Missing Class Values



- ✓ Categorical feature may have missing values
- ✓ These we can impute with most frequent strategy

## Data Science – Feature Engineering

**Program Name** Imputing categorical values with most frequent strategy  
demo5.py

```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer

students = [
    [85, 'M', 'verygood'],
    [95, 'F', 'excellent'],
    [75, np.NaN, 'good'],
    [np.NaN, 'M', 'average'],
    [70, 'M', 'good'],
    [np.NaN, np.NaN, 'verygood'],
    [92, 'F', 'verygood'],
    [98, 'M', 'excellent']
]

cols = ['marks', 'gender', 'result']
df = pd.DataFrame(students, columns = cols)

print(df)

imputer = SimpleImputer(missing_values = np.NaN,
strategy='most_frequent')

result = df['gender'].values.reshape(-1, 1)

df.gender = imputer.fit_transform(result)

print()
print(df)
```

output

	marks	gender	result
0	85.0	M	verygood
1	95.0	F	excellent
2	75.0	NaN	good
3	NaN	M	average
4	70.0	M	good
5	NaN	NaN	verygood
6	92.0	F	verygood
7	98.0	M	excellent

	marks	gender	result
0	85.0	M	verygood
1	95.0	F	excellent
2	75.0	M	good
3	NaN	M	average
4	70.0	M	good
5	NaN	M	verygood
6	92.0	F	verygood
7	98.0	M	excellent



---

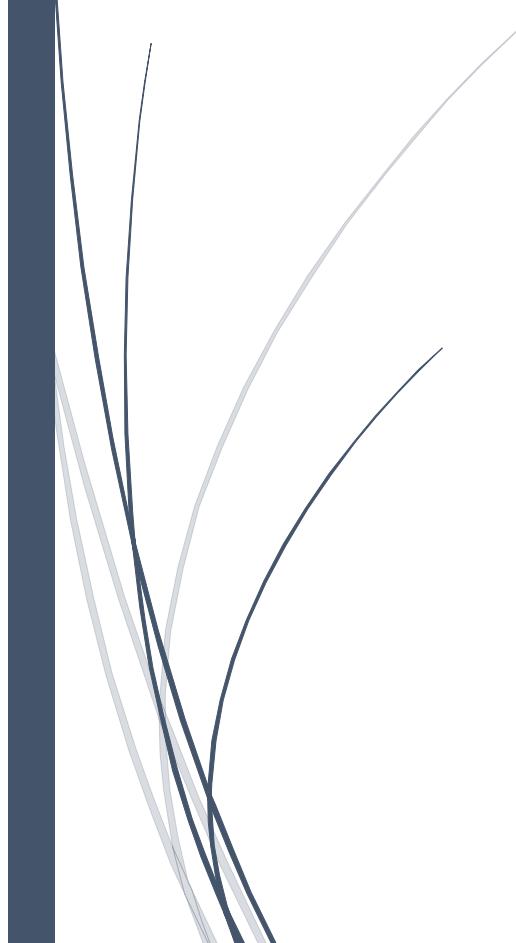
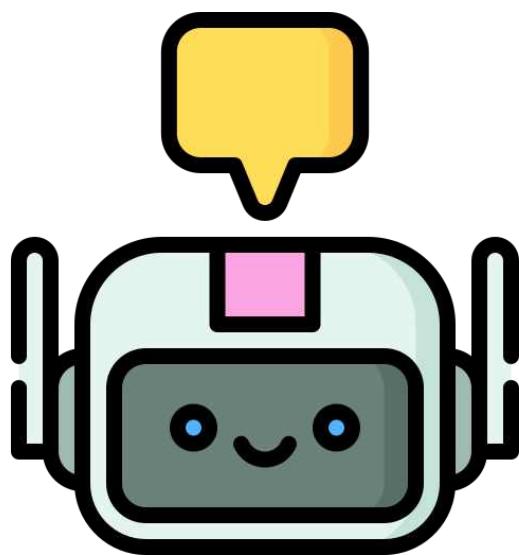
## AI Agents

---

### AI Agents Part 1 – Tutorial

---

Daniel



# AI Agents Part - 1

---

## AI Agents

<b>1. What is "AI" in AI Agent? .....</b>	<b>3</b>
<b>2. What is "Agent" in AI Agent?.....</b>	<b>3</b>
<b>3. What is AI Agent?.....</b>	<b>3</b>
<b>4. Real-world AI agent examples .....</b>	<b>4</b>
4.1. Self-Driving Cars (Tesla, Waymo).....	4
4.2. Chatbots in Banking (HDFC's EVA, SBI's SIA).....	4
4.3. Stock Trading Bots (Zerodha's Streak, Robinhood's AI Trading).....	4
4.4. Healthcare AI (IBM Watson, Google DeepMind).....	4
4.5. Fraud Detection in Banks (PayPal, Mastercard AI) .....	5
<b>5. Key Characteristics of AI Agents .....</b>	<b>5</b>
<b>6. AI Agent Categories Based on Functionality .....</b>	<b>5</b>
6.1. Reactive Agents.....	5
6.2. Deliberative Agents.....	5
6.3. Learning Agents .....	5
6.4. Multi-Agent Systems.....	5
<b>7. The Need for AI Agents .....</b>	<b>6</b>
7.1. Software development perspective.....	6
7.2. Autonomous system perspective .....	15
<b>6. Core Components of AI Agents.....</b>	<b>18</b>
6.1. Role-Playing.....	19
6.2. Focus .....	20
6.3. Tools.....	21
6.4. Cooperation .....	22
6.5. Guardrails.....	23
6.6. Memory.....	24
<b>7. Crewai Introduction .....</b>	<b>25</b>
<b>8. ollama Introduction.....</b>	<b>26</b>
<b>9. Creating AI Agents .....</b>	<b>27</b>
<b>10. Environment Setup.....</b>	<b>27</b>
<b>11. Creating Agents in CrewAI .....</b>	<b>29</b>
<b>12. Single-Agent vs. Multi-Agent Workflow in CrewAI.....</b>	<b>29</b>

## AI Agents Part - 1

---

12.1. Single-Agent Workflow .....	29
12.2. Multi-Agent Workflow .....	29
<b>13. AI Agents technical flow.....</b>	<b>30</b>
<b>14. Hello World Example: AI Research &amp; Writing with CrewAI .....</b>	<b>30</b>
14.1. AI Agents technical flow for Simple Greeting AI using CrewAI.....	30
<b>15. Use Case 1: Automated Content Creation using CrewAI .....</b>	<b>33</b>
15.1. AI Agents technical flow for Automated content creation using CrewAI .....	33
<b>16. Use Case 2: Automated Healthcare AI Summary with CrewAI.....</b>	<b>41</b>
16.1. AI Agents technical flow for healthcare content creation using CrewAI.....	41
<b>17. Use Case 3: Multi-Agent Research Assistant with CrewAI.....</b>	<b>47</b>
17.1. AI Agents technical flow for Multi-Agent Research Assistant with CrewAI.....	47

## AI Agents Part - 1

### AI Agents

#### 1. What is "AI" in AI Agent?

- ✓ AI refers to machines or software that can perform tasks that typically require human intelligence.
- ✓ These tasks include learning, reasoning, problem-solving, understanding language, and recognizing patterns.

#### 2. What is "Agent" in AI Agent?

- ✓ An agent is something that can sense its environment, make decisions, and take actions to achieve a goal.
- ✓ An AI agent is a system that perceives the world (through sensors, data, or inputs), processes that information, and takes actions accordingly.

#### 3. What is AI Agent?

- ✓ An AI agent is a smart system that interacts with its surroundings, learns from experience, and makes decisions to complete tasks efficiently.
- ✓ They can be software-based (like virtual assistants) or embodied in hardware (like robots).

**AI Agent = AI + Agent**

## AI Agents Part - 1

---

### 4. Real-world AI agent examples

- ✓ AI agents are widely used across industries to improve efficiency and automate tasks. Some key examples included here.

#### 4.1. Self-Driving Cars (Tesla, Waymo)

- ✓ **AI Role:** Detects roads, traffic signals, pedestrians, and other vehicles using sensors and cameras.
- ✓ **Agent Behavior:** Makes driving decisions like stopping, turning, or changing lanes safely.

#### 4.2. Chatbots in Banking (HDFC's EVA, SBI's SIA)

- ✓ **AI Role:** Understands customer queries about balances, transactions, and loans.
- ✓ **Agent Behavior:** Responds with relevant answers or directs users to human support.

#### 4.3. Stock Trading Bots (Zerodha's Streak, Robinhood's AI Trading)

- ✓ **AI Role:** Analyses stock market trends, news, and data.
- ✓ **Agent Behavior:** Buys or sells stocks based on predicted profitability.

#### 4.4. Healthcare AI (IBM Watson, Google DeepMind)

- ✓ **AI Role:** Reads medical reports, diagnoses diseases, and suggests treatments.
- ✓ **Agent Behavior:** Assists doctors in decision-making for better patient care.

## AI Agents Part - 1

---

### 4.5. Fraud Detection in Banks (PayPal, Mastercard AI)

- ✓ **AI Role:** Scans millions of transactions in real-time to find fraud patterns.
- ✓ **Agent Behavior:** Flags suspicious activity and prevents fraudulent transactions.

## 5. Key Characteristics of AI Agents

- ✓ **Autonomy** : Operate independently.
- ✓ **Perception** : Collect data from inputs.
- ✓ **Decision-making** : Analyze and act.
- ✓ **Action** : Execute tasks like responding or controlling systems.
- ✓ **Adaptability** : Improve over time.

## 6. AI Agent Categories Based on Functionality

### 6.1. Reactive Agents

- ✓ Respond to inputs without memory (e.g., rule-based chatbots).

### 6.2. Deliberative Agents

- ✓ Use reasoning and planning to make decisions (e.g., self-driving car AI).

### 6.3. Learning Agents

- ✓ Improve through experience (e.g., recommendation systems).

### 6.4. Multi-Agent Systems

- ✓ Multiple AI agents working together (e.g., swarm robotics).

## AI Agents Part - 1

---

### 7. The Need for AI Agents

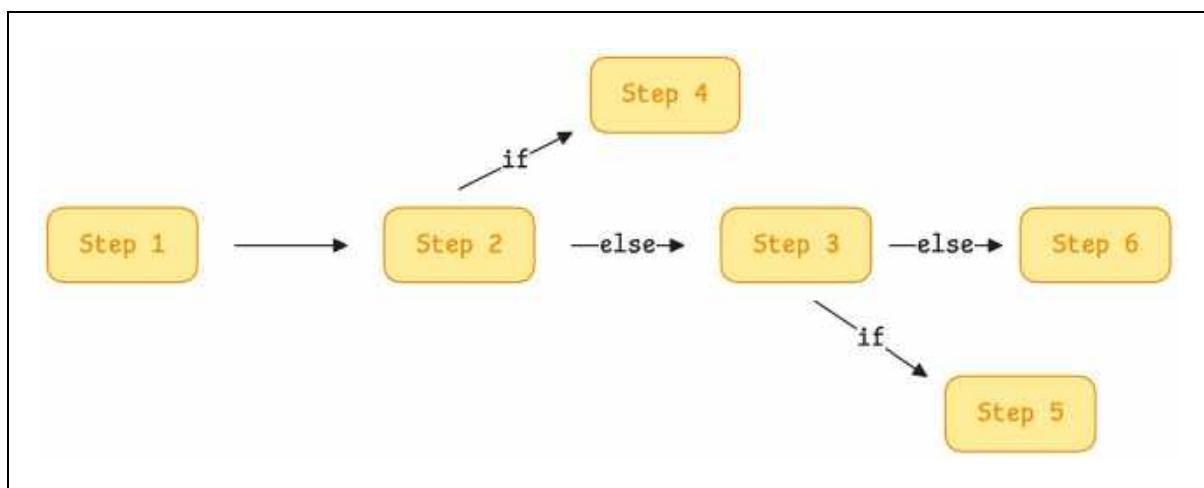
- ✓ There are several key perspectives to understanding the motivation behind AI agents:
  1. Software Development Perspective
  2. Autonomous Perspective

#### 7.1. Software development perspective

- ✓ Think about traditional software applications, they operate based on strict, predefined rules.
  - If a program needs to complete a task, every step must be explicitly defined.



- As new scenarios arise, developers constantly need to add more conditions and logic.



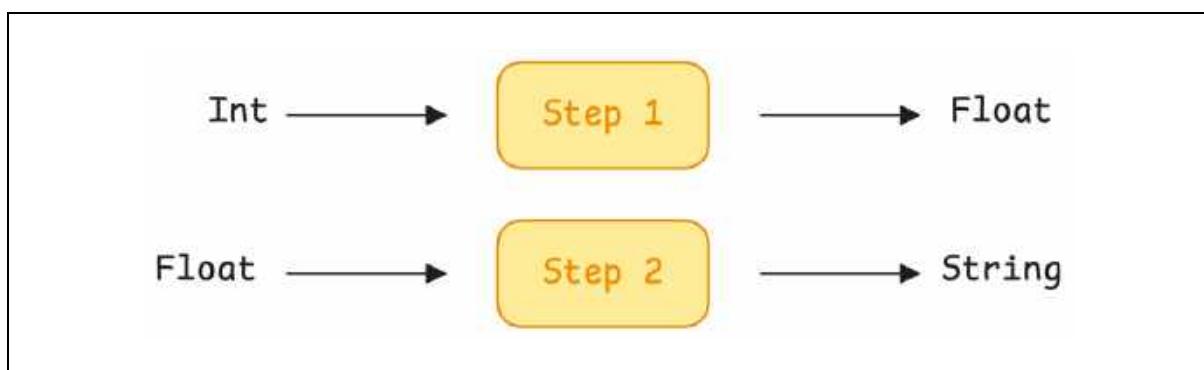
- Over time, the system becomes more difficult to scale and maintain.

## AI Agents Part - 1

---

### Traditional automation

- ✓ In other words, traditional automation follows strict predefined logic:
  - If condition A is met, do X.
  - If condition B is met, do Y.
  - Otherwise, do Z.
- ✓ Inputs have a predefined data type (text, number, etc.).
- ✓ Transformations on the input are mostly fixed.
- ✓ Output types are also fixed.



### When to Stick with Traditional Automation?

- ✓ This isn't a criticism of traditional automation, and we're not discouraging it.
- ✓ If it works for your needs, stick with it.
- ✓ There's no need to build AI agents unless truly necessary.

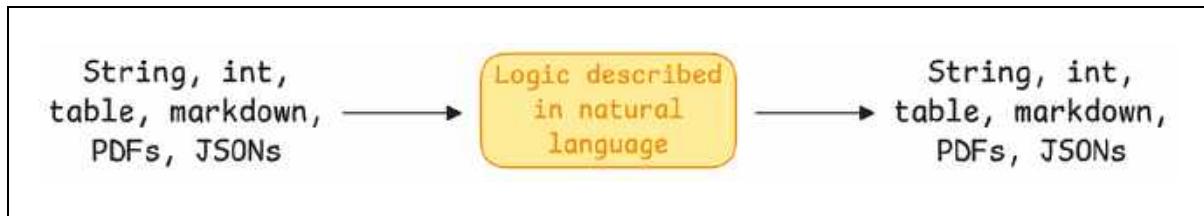
### Why AI Agents Are Different?

- ✓ No need for explicit instructions.
- ✓ Gather information dynamically.
- ✓ Use reasoning for ambiguous problems.
- ✓ Collaborate with other agents to complete tasks.
- ✓ Leverage external tools for real-time decisions.

## AI Agents Part - 1

### How AI Agents Offer More Flexibility?

- ✓ Inputs are not limited to a specific data type.
- ✓ They can be text, numbers, PDFs, tables, markdown, JSON, and more.



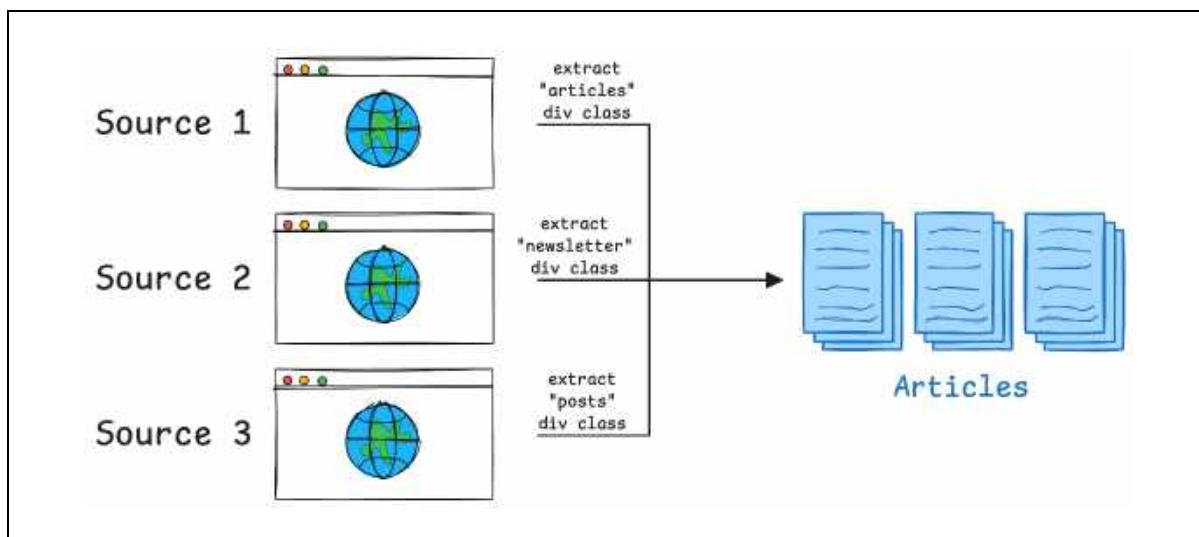
- ✓ Transformations are flexible and depend on what you ask the LLM to do.
- ✓ Outputs can vary, tokens, lists, structured data, JSON, code, and more.

## AI Agents Part - 1

---

### Traditional Approach: Manual Effort Required

- ✓ Imagine managing a news aggregation platform that collects articles from various sources.
- ✓ Traditionally, we need to manually extract and process data from different sources



- ✓ Write **scripts** to scrape multiple websites.
- ✓ Filter and categorize articles using hardcoded rules.
- ✓ Manually verify whether an article is relevant.

### Limitation

- ✓ If a website changes its layout, your scraper breaks.
- ✓ If new topics arise, you must update the filtering logic manually.

## AI Agents Part - 1

---

### AI Agent Approach: Automation & Intelligence

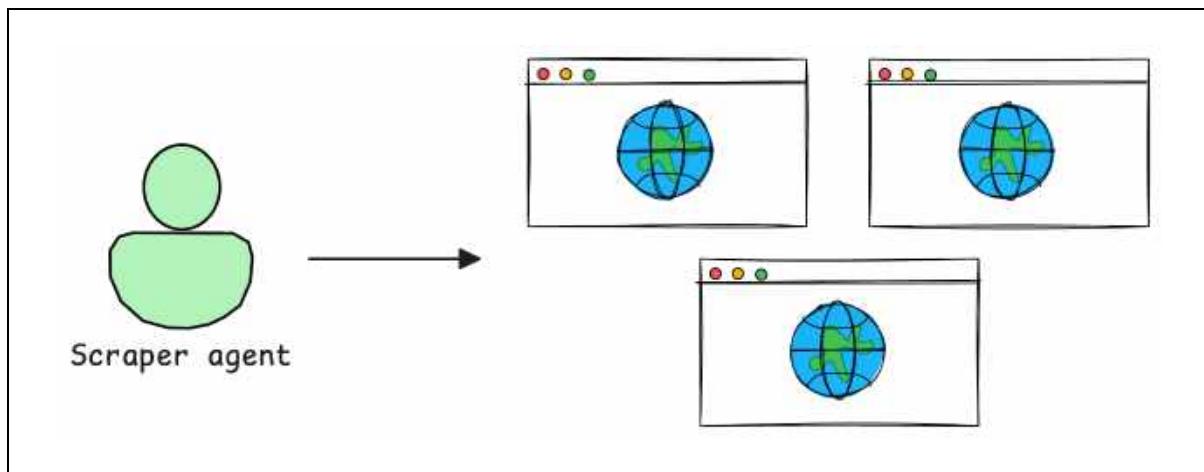
#### How AI Agents Work Together?

- ✓ AI agents collaborate to automate the entire news processing workflow from data collection to verification and summarization.
- ✓ Each agent plays a specific role in ensuring accuracy, relevance, and efficiency:
  1. A Web Scraper Agent
  2. A Topic Analysis Agent
  3. A Fact-Checking Agent
  4. A Summarization Agent

## AI Agents Part - 1

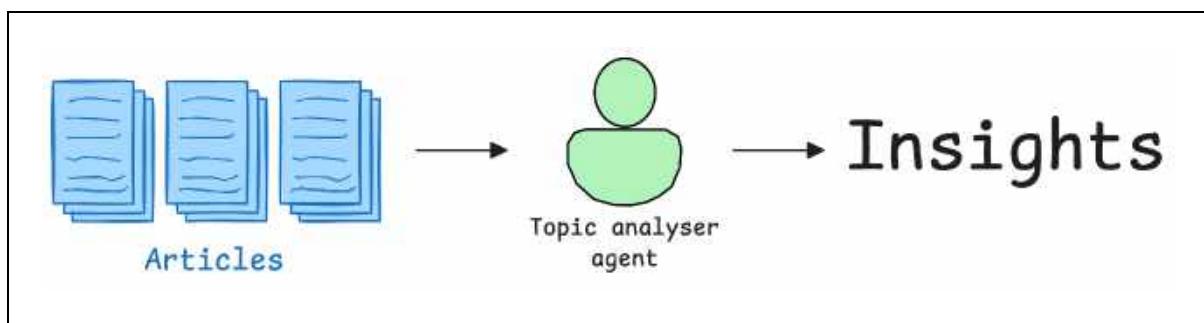
### 1. A Web Scraper Agent

- ✓ Automatically discovers new sources.
- ✓ Adapts to changes in web page structures without manual intervention.



### 2. A Topic Analysis Agent

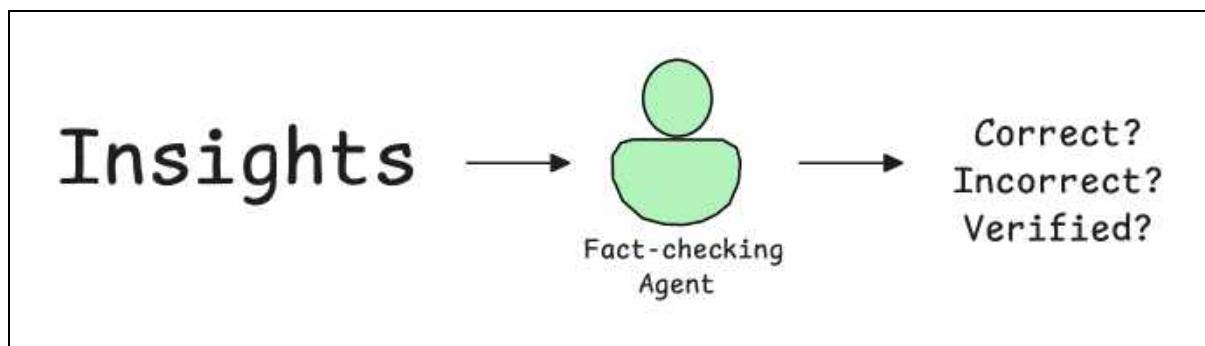
- ✓ Analyzes articles in real time.
- ✓ Detects emerging trends.
- ✓ Classifies content efficiently.



## AI Agents Part - 1

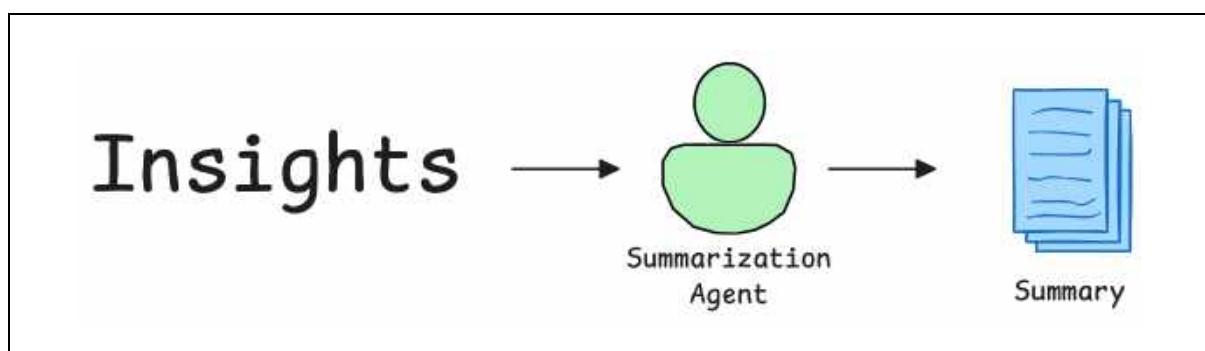
### 3. A Fact-Checking Agent

- ✓ It ensures article credibility by:
  - Cross-referencing external data sources.
  - Verifying claims for accuracy.
  - Detecting misinformation before publication.



### 4. A Summarization Agent

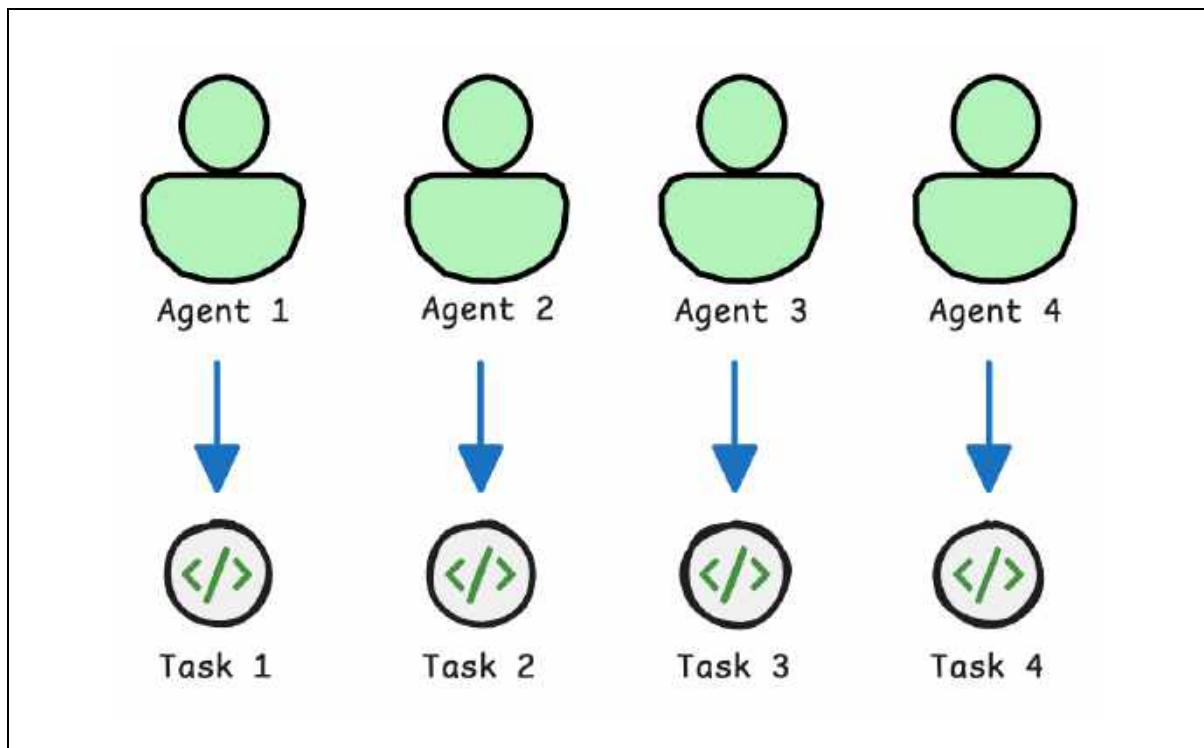
- ✓ Extracts key points from articles.
- ✓ Generates concise, easy-to-read summaries.



## AI Agents Part - 1

### Multi-agent framework

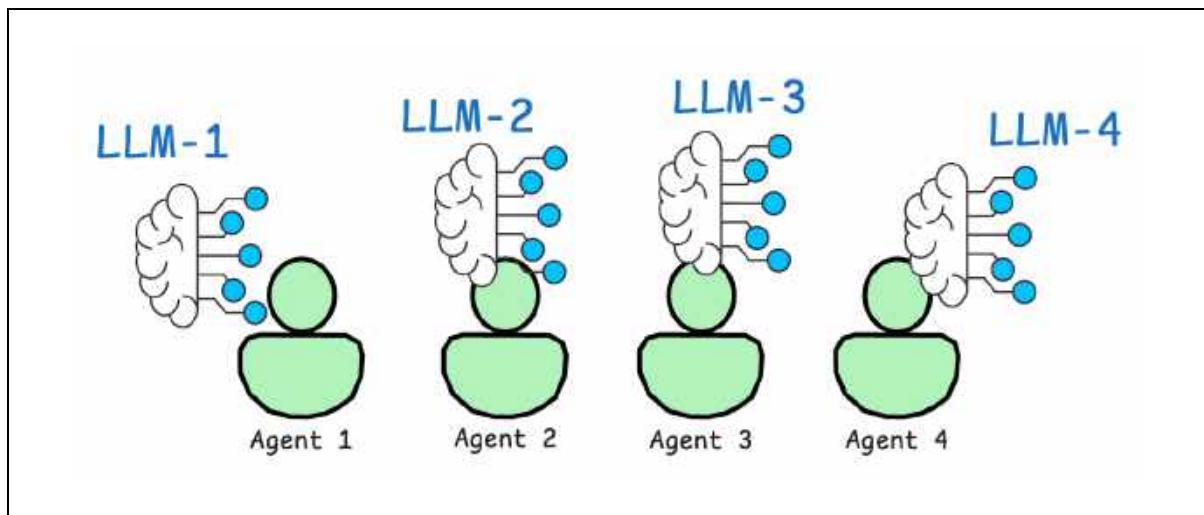
- ✓ Multi-agent means, it's a system where multiple AI agents work together.
- ✓ Each agent has a specific role but collaborates to achieve a common goal.



## AI Agents Part - 1

### Optimizing Performance with Multiple LLMs

- ✓ Each agent uses a specialized LLM for its task.
- ✓ Selecting the best LLM enhances efficiency and accuracy.



### AI Agents: Learning and Adapting

- ✓ AI agents continuously learn, adapt, and optimize.
- ✓ They update automatically without manual intervention.
- ✓ Beyond fixed rules, they collaborate and solve problems autonomously.

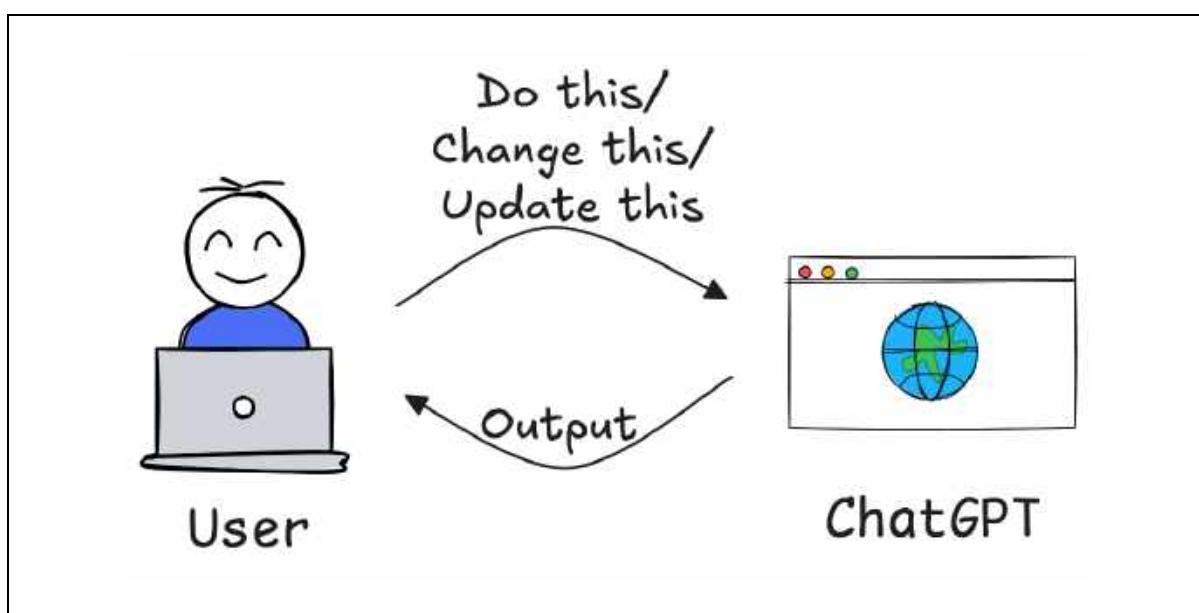
## AI Agents Part - 1

---

### 7.2. Autonomous system perspective

#### Limitations of Traditional LLM Interactions

- ✓ Users must refine outputs manually.
- ✓ The process is iterative and time-consuming:
  - Ask a question.
  - Review the response.
  - Adjust the prompt.
  - Repeat until satisfied



#### The Challenge of Using a Standard LLM for Research

- ✓ Imagine you need a report on the latest AI research trends. With a standard LLM, you would:
  - Ask for a summary of recent AI papers.
  - Review the response and realize you need sources.
  - Request citations and get a list of papers.
  - Notice outdated sources and refine your query.
  - Repeat the process until you get a useful report.
- ✓ This iterative process takes time and effort, requiring you to make decisions at every step.

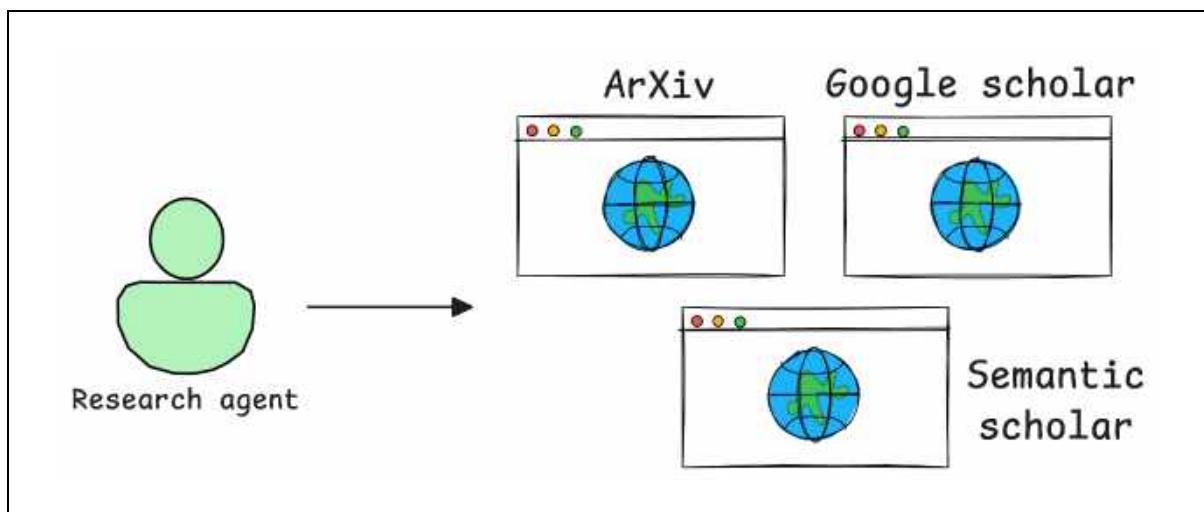
## AI Agents Part - 1

### AI Agents in Research

- ✓ Automate data gathering, analysis, and verification.
- ✓ Reduce the need for constant user input.
- ✓ Enhance efficiency and accuracy in research.

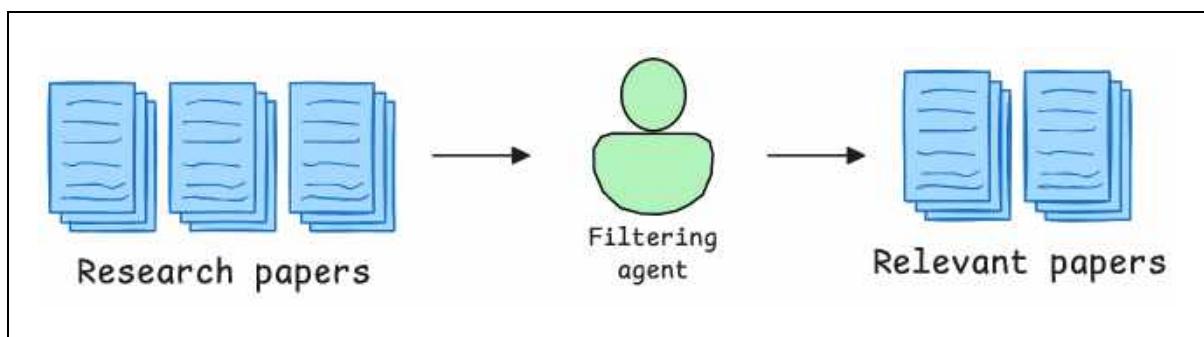
### 1. A Research Agent

- ✓ Searches for AI research papers automatically.
- ✓ Retrieves data from sources like arXiv, Semantic Scholar, or Google Scholar.
- ✓ Ensures access to the latest credible information.



### 2. A Filtering Agent

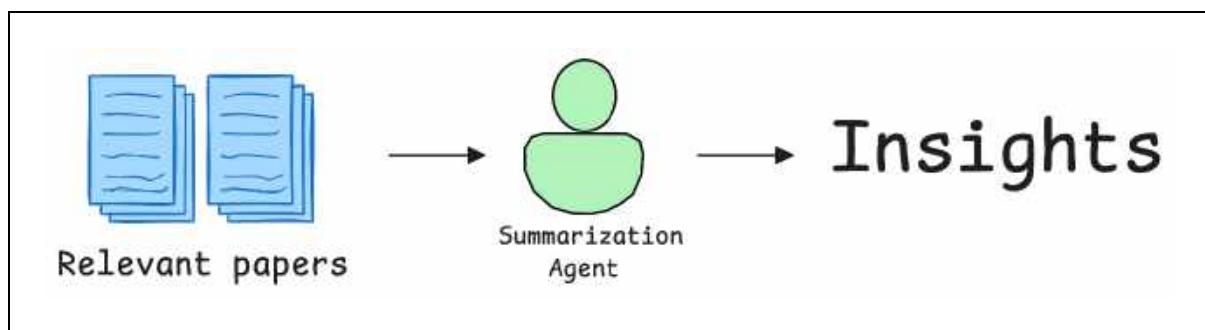
- ✓ Analyzes retrieved papers for relevance.
- ✓ Filters based on citation count, publication date, and keywords.
- ✓ Ensures only high-quality research is considered.



## AI Agents Part - 1

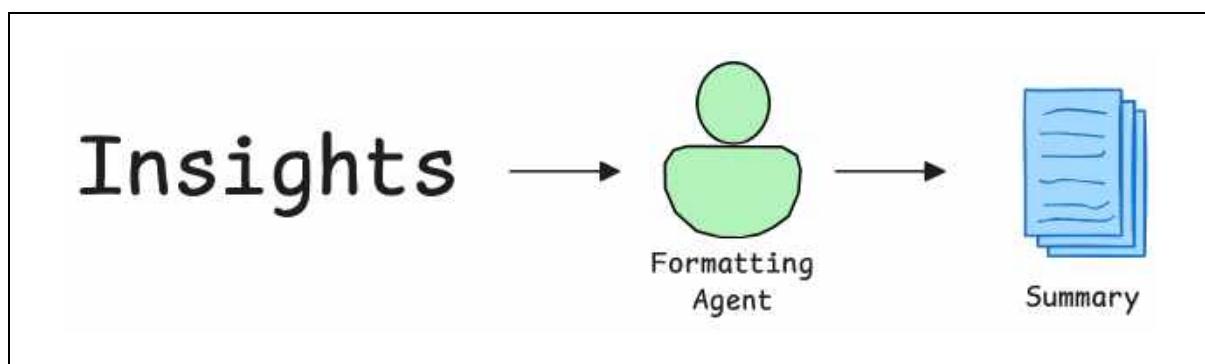
### 3. A Summarization Agent

- ✓ Extracts key insights from research papers.
- ✓ Compiles information into a concise, easy-to-read report.
- ✓ Saves time by presenting only the most relevant details.



### 4. A Formatting Agent

- ✓ Organizes the final report for clarity.
- ✓ Ensures a professional and structured layout.
- ✓ Enhances readability and presentation.



## AI Agents Part - 1

---

### 6. Core Components of AI Agents

- ✓ AI agents' reason, plan, and act autonomously.
- ✓ Six key building blocks ensure reliability and intelligence.
- ✓ Essential for real-world effectiveness.

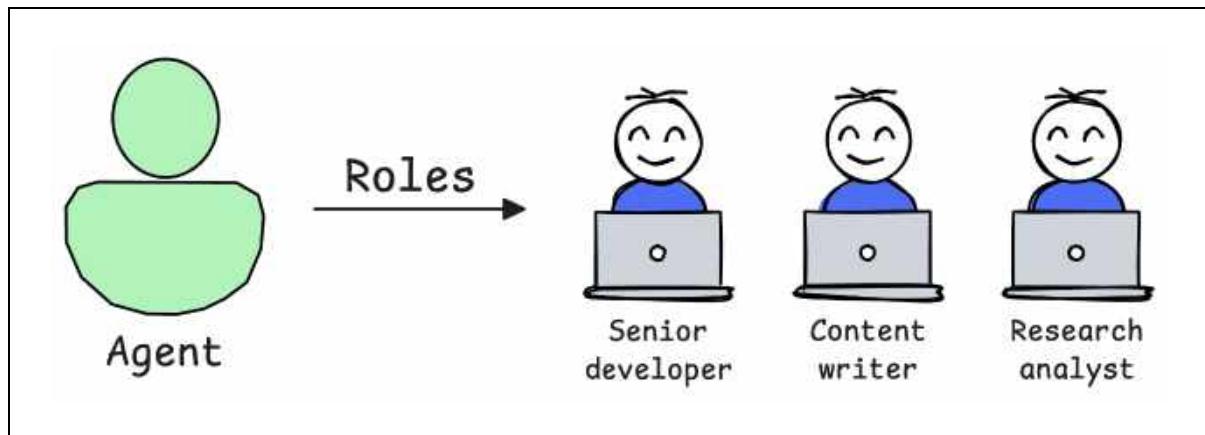
### Key Elements of AI Agents

1. Role-playing
2. Focus
3. Tools
4. Cooperation
5. Guardrails
6. Memory

## AI Agents Part - 1

### 6.1. Role-Playing

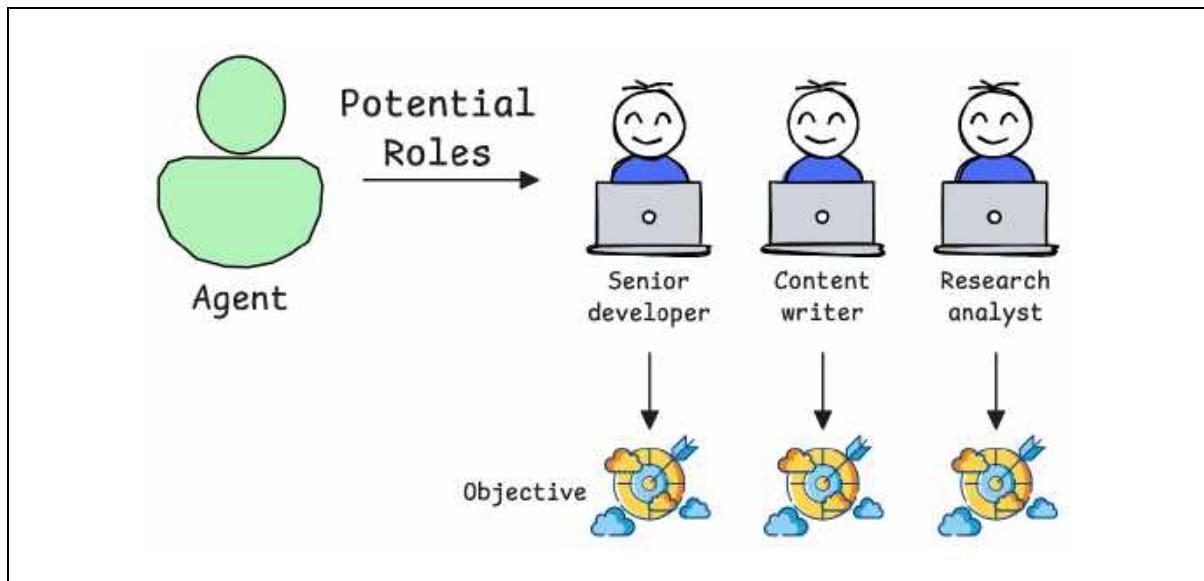
- ✓ Specific roles enhance task specialization.
- ✓ Defined roles improve structured and relevant responses.
- ✓ Clear identity and objectives ensure better-aligned outputs.



## AI Agents Part - 1

### 6.2. Focus

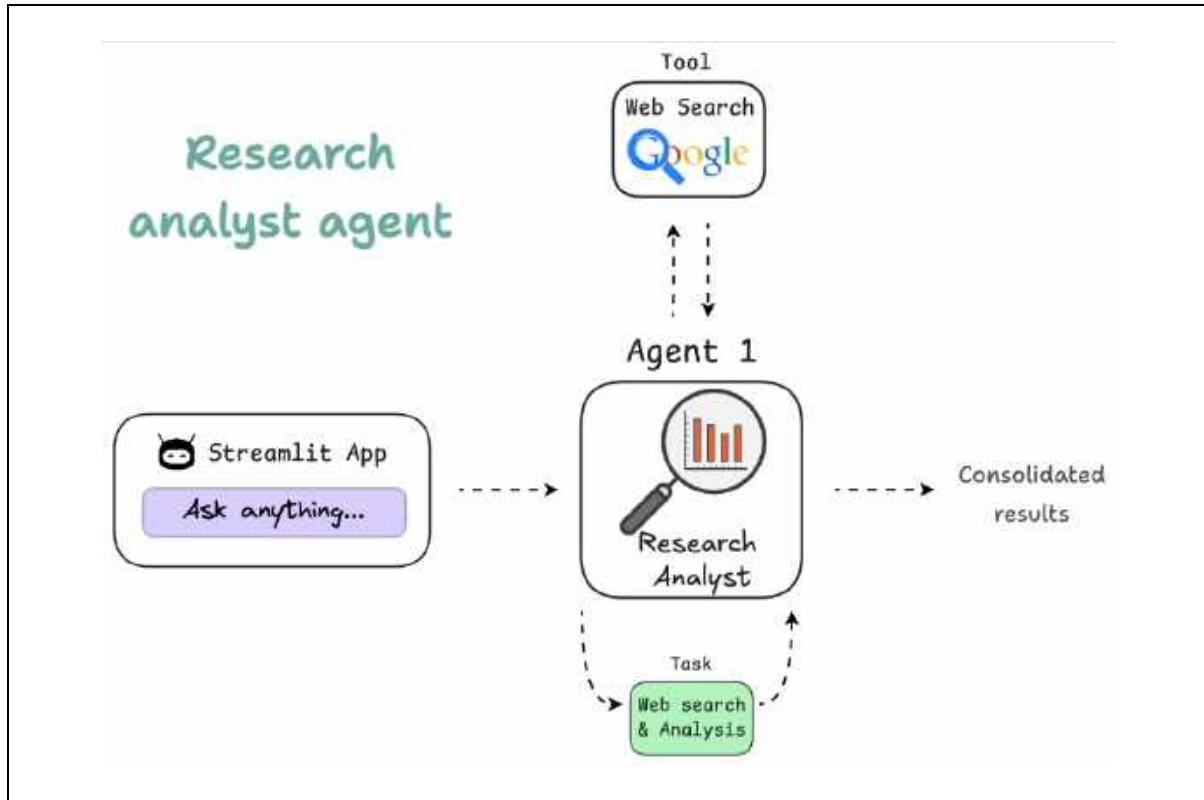
- ✓ Ensures agents stay aligned with their objectives.
- ✓ Minimizes errors and improves accuracy.
- ✓ More data isn't always better—structured input leads to better results.



## AI Agents Part - 1

### 6.3. Tools

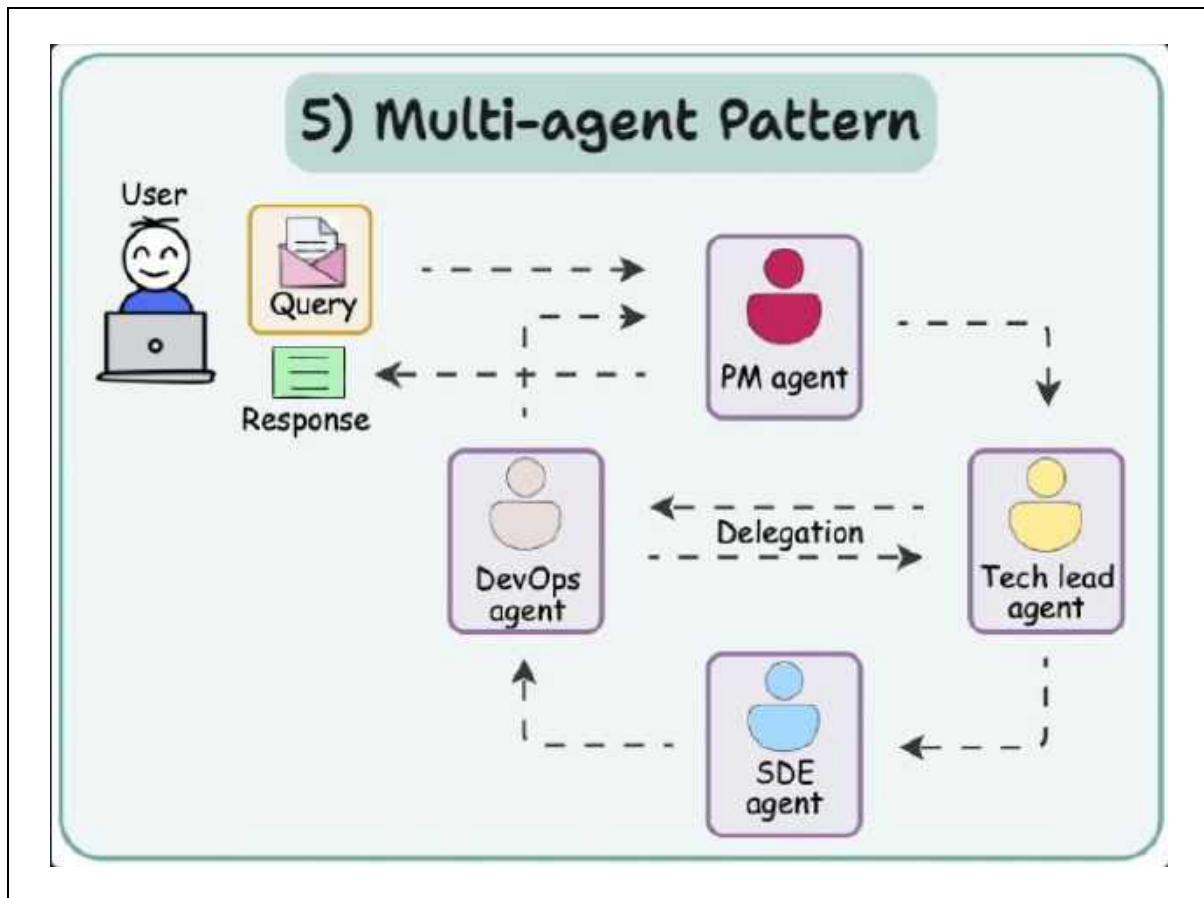
- ✓ Enhance AI agents beyond text-based reasoning.
- ✓ Enable real-time data retrieval and system interactions.



## AI Agents Part - 1

### 6.4. Cooperation

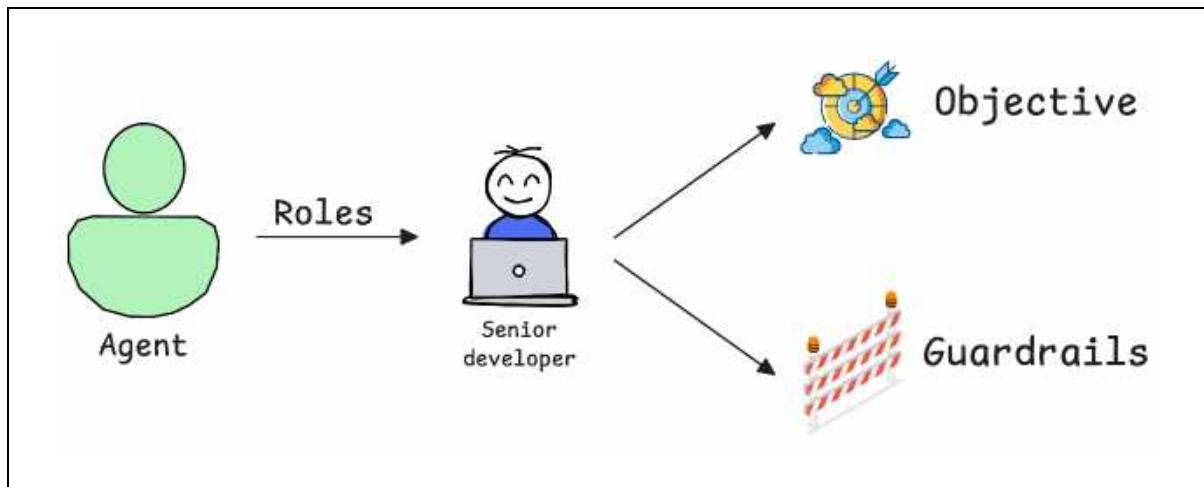
- ✓ Enabling Multi-Agent Collaboration
- ✓ Enables AI agents to work together efficiently.
- ✓ Enhances performance through collaboration and feedback exchange.



## AI Agents Part - 1

### 6.5. Guardrails

- ✓ Keeping AI Agents Reliable
- ✓ Implement safety measures for controlled behavior.
- ✓ Prevent false information, infinite loops, and unreliable decisions.



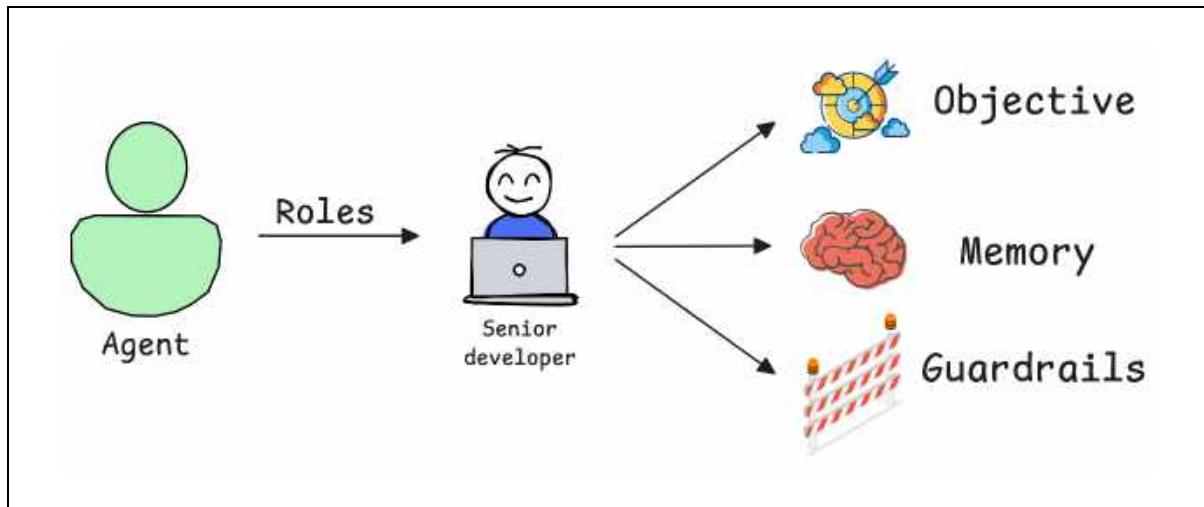
### Examples of Effective Guardrails

- ✓ **Limit Tool Usage:** Prevents excessive API calls or irrelevant queries.
- ✓ **Validation Checkpoints:** Ensures outputs meet quality standards before proceeding.
- ✓ **Fallback Mechanisms:** Enables human or agent intervention when needed.

## AI Agents Part - 1

### 6.6. Memory

- ✓ **Without memory:** Agents reset every session, leading to repetitive interactions.
- ✓ **With memory:** Agents retain context, refine responses, and improve decision-making over time.



## AI Agents Part - 1

---

### 7. CrewAI Introduction

- ✓ It's a python framework for building multi-agent AI systems.
- ✓ By using this we can create specialized agents with unique roles, collaborate, specialize, and automate tasks.

### How CrewAI Enhances AI Agents

- ✓ **Multi-Agent Collaboration:** Agents work together efficiently.
- ✓ **Specialization:** Assigns specific roles for better performance.
- ✓ **Task Automation:** Reduces manual intervention.
- ✓ **Memory & Context:** Enables continuous learning.
- ✓ **Tool Integration:** Uses external tools for enhanced capabilities.
- ✓ **Accuracy & Efficiency:** Delivers structured and reliable outputs.

### Adaptive, goal-driven AI agents

- ✓ With CrewAI, we can efficiently design **adaptive, goal-driven AI agents** that collaborate and execute complex tasks independently.

## AI Agents Part - 1

---

### 8. ollama Introduction

- ✓ Ollama enables running and customizing LLMs locally.
- ✓ Avoids cloud dependency by supporting open-source AI models.
- ✓ Optimized for efficient local deployment.

### How Ollama Enhances AI Agents?

- ✓ **Local Execution:** No cloud dependency, ensuring privacy.
- ✓ **Optimized Performance:** Faster inference with minimal resources.
- ✓ **Seamless Model Management:** Load, switch, and fine-tune easily.
- ✓ **Lightweight & Scalable:** Works on both edge devices and enterprises.
- ✓ **AI Agent Compatibility:** Supports autonomous agents with specialized models.

## AI Agents Part - 1

### 9. Creating AI Agents

- ✓ Now, let's move to implementation! Using CrewAI, an open-source framework, we can:
  - **Assign Roles:** Specialize agents for specific tasks.
  - **Set Goals:** Define clear objectives for each agent.
  - **Integrate Tools:** Equip agents with the right resources.
  - **Use Any LLM:** Seamlessly work with various language models.

### 10. Environment Setup

- ✓ To get started, install CrewAI as follows:

- pip install crewai
  - pip install crewai-tools

### Choosing the Right LLM for CrewAI

- ✓ Ollama serves LLMs locally for seamless performance.
- ✓ CrewAI supports multiple providers, including:
  - OpenAI
  - Gemini
  - Groq
  - Azure
  - Fireworks AI
  - Cerebras
  - SambaNova & more!
- ✓ This flexibility lets you pick the best model for your needs.

## AI Agents Part - 1

### Setting Up OpenAI with CrewAI

- ✓ To configure OpenAI as your LLM provider, create a `.env` file in your project's directory and add your API key like this:

```
○ OPENAI_API_KEY = your-api-key-here
```

- ✓ Make sure to replace `your-api-key-here` with your actual OpenAI API key.
- ✓ This allows CrewAI to authenticate and interact with OpenAI's models seamlessly.

### Installing and Running Ollama

- ✓ Download Ollama: Get it from official website <https://ollama.com>
- ✓ Install – Follow the on-screen setup instructions.
- ✓ Verify Installation run: `ollama --version`
- ✓ Download a Model: `ollama pull llama3.2`
- ✓ Run Locally: `ollama run llama3.2`
- ✓ Ready to Use – Serve LLMs locally without external APIs. Now, let's build our AI agents!

## AI Agents Part - 1

---

### 11. Creating Agents in CrewAI

- ✓ In CrewAI, Agents are autonomous entities with:
  - **Role:** Defines function (e.g., "Senior Technical Writer").
  - **Goal:** Specifies objective (e.g., "Write a publication-ready article").
  - **Backstory:** Adds expertise for better interactions.
- ✓ These attributes enable agents to perform tasks efficiently and independently.

### 12. Single-Agent vs. Multi-Agent Workflow in CrewAI

#### 12.1. Single-Agent Workflow

- ✓ Only one agent is assigned a task.
- ✓ Handles the entire process independently.
- ✓ Simple, efficient, and best for focused tasks (e.g., writing an article).

#### 12.2. Multi-Agent Workflow

- ✓ Multiple agents collaborate on different tasks.
- ✓ Workflows are divided (e.g., research, writing, review).
- ✓ Ideal for complex projects requiring diverse expertise.

## AI Agents Part - 1

---

### 13. AI Agents technical flow

- ✓ **Step 1:** Import Libraries.
- ✓ **Step 2:** Configure the LLM
- ✓ **Step 3:** Create Agents:
  - **Agent 1**
  - **Agent 2, etc**
- ✓ **Step 4:** Define Tasks
- ✓ **Step 5:** Create a Crew
- ✓ **Step 6:** Kickoff the Crew
- ✓ **Step 7:** Check the Response

### 14. Hello World Example: AI Research & Writing with CrewAI

- ✓ Simple AI workflow using CrewAI to generate a friendly greeting.
- ✓ We are using **a single-agent** workflow.

#### 14.1. AI Agents technical flow for Simple Greeting AI using CrewAI

- ✓ **Step 1:** Import Libraries, Essential for using CrewAI and LLMs.
- ✓ **Step 2:** Configure the LLM, Define the Ollama LLaMA 3.2 language model.
- ✓ **Step 3:** Create Agent, Define its role, goal, and backstory.
  - **Agent 1:** Friendly AI, Responds with a simple greeting.
- ✓ **Step 4:** Define Task, Assign a specific task to the agent.
  - **Task:** Say "Hello, World!"
- ✓ **Step 5:** Create a Crew, Group the agent and assign the task.
- ✓ **Step 6:** Kickoff the Crew, Execute the workflow.
- ✓ **Step 7:** Check the Response, Validate and display the greeting.

## AI Agents Part - 1

**Program Name** Steps from 1 to 7, Simple Greeting AI using CrewAI  
greetings\_demo.py

```
print("Step 1: Importing the libraries")

from crewai import Agent
from crewai import Task
from crewai import Crew
from crewai import LLM

print("Step 2: Configure the llm")

llm = LLM(
    model = "ollama/llama3.2",
    base_url = "http://localhost:11434"
)

print("Step 3: Create Agent")

simple_agent = Agent(
    role = "Friendly AI",
    goal = "Respond with a simple greeting.",
    backstory = "You are a helpful AI that loves saying hello.",
    verbose = True
)

print("Step 4: Define Tasks")

greeting_task = Task(
    description = "Say 'Hello, World!'",  

    agent = simple_agent,  

    expected_output = "A friendly greeting."
)
```

## AI Agents Part - 1

```
print("Step 5: Create a Crew")

crew = Crew(
    agents = [simple_agent],
    tasks = [greeting_task],
    verbose = True
)

print("Step 6: Kickoff the Crew")

response = crew.kickoff()

print("Step 7: Check the Response")

print(response)
```

### Output

**Step 1:** Importing the libraries  
**Step 2:** Configure the llm  
**Step 3:** Create Agent  
**Step 4:** Define Tasks  
**Step 5:** Create a Crew  
**Step 6:** Kickoff the Crew  
**Step 7:** Check the Response

Hello, World!

## AI Agents Part - 1

---

### 15. Use Case 1: Automated Content Creation using CrewAI

- ✓ Uses CrewAI to generate structured content.
- ✓ Senior Technical Writer (Ollama LLaMA 3.2) writes an engaging article on AI Agents.
- ✓ Ensures accuracy, efficiency, and scalability.
- ✓ We are using a **single-agent** workflow.

#### 15.1. AI Agents technical flow for Automated content creation using CrewAI

- ✓ **Step 1:** Import Libraries - Essential for using CrewAI and LLMs.
- ✓ **Step 2:** Configure the LLM - Define the language model before creating agents.
- ✓ **Step 3:** Create Agents, Define their role, goal, and backstory.
  - Agent 1: Senior Technical Writer
- ✓ **Step 4:** Define Tasks, Assign specific tasks to each agent.
- ✓ **Step 5:** Create a Crew, assign tasks.
- ✓ **Step 6:** Kickoff the Crew, Execute the workflow.
- ✓ **Step 7:** Check the Response, Validate and analyze the output.

## AI Agents Part - 1

**Program Name** Steps from 1 to 7, Automated Content Creation with CrewAI  
senior\_tech\_writer.py

```
print("Step 1: Importing the libraries")
```

```
from crewai import Agent
from crewai import Task
from crewai import Crew
from crewai import LLM
```

```
print("Step 2: Configure the llm")
```

```
llm = LLM(
    model = "ollama/llama3.2",
    base_url = "http://localhost:11434"
)
```

```
print("Step 3: Create Agent")
```

```
senior_technical_writer = Agent(
    role = "Senior Technical Writer",
    goal = """Craft clear, engaging, and well-structured
technical content based on research findings""",
    backstory = """You are an experienced technical writer
with expertise in simplifying complex concepts,
structuring content for readability, and ensuring accuracy
in documentation""",
    llm = llm,
    verbose = True
)
```

## AI Agents Part - 1

```
print("Step 4: Create Task")

writing_task = Task(
    description = """Write a well-structured, engaging,
    and technically accurate article
    on {topic}.""",
    agent = senior_technical_writer,
    expected_output = """A polished, detailed, and easy-to-
    read article on the given topic."""
)

print("Step 5: Create a Crew ")

crew = Crew(
    agents = [senior_technical_writer],
    tasks = [writing_task],
    verbose = True
)

print("Step 6: Run the Crew")

response = crew.kickoff(inputs = {"topic" : "AI Agents"})

print("Step 7: Check the Response")

print(response)
```

## AI Agents Part - 1

### Output

- Step 1:** Importing the libraries
- Step 2:** Configure the llm
- Step 3:** Create Agent
- Step 4:** Create Task
- Step 5:** Create a Crew
- Step 6:** Run the Crew
- Step 7:** Check the Response

### AI Agents: The Future of Intelligent Systems

Artificial intelligence (AI) has revolutionized numerous industries, transforming the way we live and work. At the heart of this technological advancement are AI agents, intelligent systems that can perform tasks autonomously, making decisions based on complex algorithms and machine learning models. In this article, we will delve into the world of AI agents, exploring their capabilities, types, and potential applications.

### Understanding AI Agents

AI agents are software-based entities that operate independently, executing tasks without human intervention. They use machine learning techniques to analyze data, identify patterns, and make predictions or decisions. AI agents can be classified into two primary categories: rule-based and machine learning-based systems.

#### Rule-Based Systems

---

Rule-based systems rely on pre-defined rules and decision-making processes to guide their actions. These systems are typically used in applications where a high degree of accuracy is required, such as in healthcare and finance. Rule-based AI

## AI Agents Part - 1

agents can be programmed to recognize specific patterns or anomalies, allowing them to make predictions or recommendations.

### Machine Learning-Based Systems

---

Machine learning-based systems, on the other hand, use machine learning algorithms to analyze data and improve their performance over time. These systems are more flexible than rule-based systems, as they can adapt to changing circumstances and learn from experience. Machine learning-based AI agents are commonly used in applications such as natural language processing and computer vision.

### Types of AI Agents

---

AI agents can be categorized into several types based on their capabilities and functionality:

1. **Simple Reflex Agents**: These agents respond to a specific set of input actions, using pre-defined rules to guide their behavior.
2. **Model-Based Reflex Agents**: These agents use knowledge-based systems to reason about the world, making decisions based on a combination of rules and machine learning models.
3. **Utility Function Agents**: These agents use utility functions to evaluate the consequences of their actions, optimizing their performance based on a set of goals or objectives.

### Applications of AI Agents

---

## AI Agents Part - 1

AI agents have numerous applications across various industries, including:

1. **Virtual Assistants**: AI agents are used in virtual assistants such as Siri and Alexa to provide personalized recommendations and perform tasks.
2. **Robotics**: AI agents control robots, allowing them to navigate complex environments and interact with humans.
3. **Healthcare**: AI agents are used in medical diagnosis and treatment, analysing patient data and providing insights to healthcare professionals.
4. **Finance**: AI agents are used in trading and investment, analysing market trends and making predictions.

### Advantages of AI Agents

---

AI agents offer several advantages over traditional systems, including:

1. **Increased Efficiency**: AI agents can automate tasks, reducing the need for human intervention and increasing productivity.
2. **Improved Accuracy**: AI agents use machine learning algorithms to analyze data, reducing errors and improving accuracy.
3. **Personalization**: AI agents can provide personalized recommendations and services, enhancing the user experience.

### Challenges and Limitations of AI Agents

---

While AI agents offer numerous benefits, they also pose several challenges and limitations, including:

## AI Agents Part - 1

1. **\*\*Explainability\*\*:** AI agents can be difficult to understand, making it challenging to explain their decision-making processes.
2. **\*\*Bias\*\*:** AI agents can inherit biases from the data used to train them, leading to unfair outcomes.
3. **\*\*Security\*\*:** AI agents can pose security risks if not designed with proper safeguards.

### Conclusion

---

AI agents are intelligent systems that have revolutionized numerous industries, transforming the way we live and work. With their capabilities in machine learning, natural language processing, and computer vision, AI agents offer numerous advantages over traditional systems. However, they also pose several challenges and limitations, including explainability, bias, and security concerns. As AI continues to evolve, it is essential to address these challenges and ensure that AI agents are designed with proper safeguards to ensure their safe and responsible deployment.

### Final Thoughts

---

The future of AI agents holds much promise, with potential applications in numerous industries. However, as we continue to develop and deploy AI systems, it is crucial to prioritize transparency, accountability, and fairness. By doing so, we can unlock the full potential of AI agents, creating a brighter future for humanity.

In conclusion, AI agents are intelligent systems that have transformed numerous industries. With their capabilities in machine learning, natural language processing, and computer vision, they offer numerous advantages over traditional

## AI Agents Part - 1

systems. However, it is essential to address the challenges and limitations associated with AI agents, ensuring that they are designed with proper safeguards to ensure their safe and responsible deployment.

As we move forward, it is crucial to prioritize transparency, accountability, and fairness in the development and deployment of AI agents. By doing so, we can unlock the full potential of these systems, creating a brighter future for humanity.

### Enhancing Performance with Stronger LLMs

- ✓ Weaker LLMs may produce less accurate responses.
- ✓ Upgrade to DeepSeek or GPT-4 for better accuracy & efficiency.
- ✓ Model switching steps were covered earlier.

## AI Agents Part - 1

---

### 16. Use Case 2: Automated Healthcare AI Summary with CrewAI

- ✓ Automates research and writing on AI in healthcare.
- ✓ **Multiple agents** gather insights and creates a structured summary.
- ✓ Ensures efficiency, accuracy, and time-saving.

#### 16.1. AI Agents technical flow for healthcare content creation using CrewAI

- ✓ **Step 1:** Import Libraries, Essential for using CrewAI and LLMs.
- ✓ **Step 2:** Configure the LLM, Define the language model before creating agents.
- ✓ **Step 3:** Create Agents, Define their role, goal, and backstory.
  - **Agent 1:** Researcher, Gathers information.
  - **Agent 2:** Writer, Summarizes research findings.
- ✓ **Step 4:** Define Tasks, Assign specific tasks to each agent.
- ✓ **Step 5:** Create a Crew, group agents and assign tasks.
- ✓ **Step 6:** Kickoff the Crew, Execute the workflow.
- ✓ **Step 7:** Check the Response – Validate and analyze the output.

## AI Agents Part - 1

**Program** Step from 1 to 7: Healthcare use case

**Name** health\_care\_demo.py

```
print("Step 1: Importing the libraries")
```

```
from crewai import Agent
from crewai import Task
from crewai import Crew
from crewai import LLM
```

```
print("Step 2: Configure the llm")
```

```
llm = LLM(
    model = "ollama/llama3.2",
    base_url = "http://localhost:11434"
)
```

```
print("Step 3.1: Create Researcher Agent")
```

```
researcher = Agent(
    role = "Researcher",
    goal = "Gather information about a given topic.",
    backstory = "A detail-oriented AI skilled in finding
    accurate information.",
    verbose = True,
    allow_delegation = False
)
```

## AI Agents Part - 1

```
print("Step 3.2: Create Writer Agent")

writer = Agent(
    role = "Writer",
    goal = "Summarize information into a concise and
    readable format.",
    backstory = "A creative AI that enjoys writing clear and
    engaging summaries.",
    verbose = True,
    allow_delegation = False
)

print("Step 4.1: Create Research Task")

research_task = Task(
    description = "Find key details about the benefits of AI in
    healthcare.",
    agent = researcher,
    expected_output = "A list of key benefits of AI in
    healthcare."
)

print("Step 4.2: Create Writing Task")

writing_task = Task(
    description = "Summarize the research findings into a
    short article.",
    agent = writer,
    expected_output = "A well-structured summary of AI's
    benefits in healthcare."
)
```

## AI Agents Part - 1

```
print("Step 5: Create a Crew")

crew = Crew(
    agents = [researcher, writer],
    tasks = [research_task, writing_task],
    verbose = True
)

print("Step 6: Run the Crew")

response = crew.kickoff()

print("Step 7: Check the Response")

print(response)
```

### Output

**Step 1:** Importing the libraries  
**Step 2:** Configure the llm  
**Step 3.1:** Create Researcher Agent  
**Step 3.2:** Create Writer Agent  
**Step 4.1:** Create Research Task  
**Step 4.2:** Create Writing Task  
**Step 5:** Create a Crew  
**Step 6:** Run the Crew  
**Step 7:** Check the Response

### \*\*The Transformative Benefits of AI in Healthcare\*\*

Artificial Intelligence (AI) is revolutionizing the healthcare sector through a range of innovative applications that enhance diagnostic accuracy, personalize medicine, and streamline operations. Here are some key benefits of AI in healthcare that demonstrate its transformative potential:

## AI Agents Part - 1

1. **Enhanced Diagnostic Accuracy**: AI algorithms, especially those utilizing deep learning techniques, surpass traditional diagnostic methods by providing more accurate analyses of medical images, such as X-rays, MRIs, and CT scans. Research indicates that AI can often match or even outperform human radiologists in identifying diseases, including cancer.
2. **Personalized Medicine**: By analysing extensive patient data, AI helps in crafting personalized treatment plans. It predicts individual responses to various therapies based on genetic profiles, lifestyle factors, and disease characteristics, resulting in more tailored and effective healthcare.
3. **Predictive Analytics**: AI's ability to process large datasets enables it to recognize patterns and forecast patient outcomes. Machine learning models can predict disease outbreaks or assess which patients are at a higher risk of readmission, facilitating timely preventive care.
4. **Operational Efficiency**: AI enhances healthcare operations through automation of routine tasks such as patient scheduling, flow management, and billing processes. This efficiency not only streamlines healthcare delivery but also alleviates administrative workloads on healthcare staff.
5. **Drug Discovery and Development**: The integration of AI in drug discovery accelerates the identification of potential drug candidates and assesses their efficacy by analysing biological data. This significantly reduces the time and financial investment needed to bring new medications to market.
6. **Remote Monitoring and Telemedicine**: AI-powered tools allow for continuous health monitoring through wearable devices. These applications enable healthcare providers to track

## AI Agents Part - 1

vital signs and address emerging health issues early on, particularly beneficial for managing chronic illnesses.

7. **\*\*Enhanced Patient Engagement\*\*:** Virtual assistants and chatbots driven by AI offer patients immediate access to information and support, fostering better communication and engagement with healthcare providers. This assists patients in managing conditions effectively and adhering to treatment plans.
8. **\*\*Integration of Data from Multiple Sources\*\*:** AI systems excel at synthesizing health data from diverse sources like electronic health records (EHRs), genetic data, and wearables. This comprehensive analysis equips healthcare professionals with a holistic view for informed decision-making.
9. **\*\*Efficiency in Clinical Trials\*\*:** AI optimizes various aspects of clinical trials, including design, patient recruitment, and monitoring, thereby expediting the overall process. It helps in identifying suitable candidates and predicting responses, which enhances the likelihood of trial success.
10. **\*\*Supporting Healthcare Professionals\*\*:** By helping with administrative tasks and processing complex medical data, AI supports healthcare providers in focusing more on their patients. It offers evidence-based recommendations that improve the quality of care.

In summary, the applications of AI in healthcare showcase its incredible ability to improve diagnostic processes, enhance patient outcomes, and create operational efficiencies. As AI continues to evolve, it holds the promise of transforming healthcare delivery into a more accurate, efficient, and patient-centered system.

## AI Agents Part - 1

---

### 17. Use Case 3: Multi-Agent Research Assistant with CrewAI

- ✓ Uses CrewAI to automate and enhance research tasks.
- ✓ **Multiple Agents**
  - Research Agent gathers real-time, relevant information.
  - Summarization Agent structures findings into a clear summary.
  - Fact-Checking Agent verifies accuracy for reliability.
  - Improves accuracy, efficiency, and scalability compared to a single-agent system

#### 17.1. AI Agents technical flow for Multi-Agent Research Assistant with CrewAI

- ✓ **Step 1:** Import Libraries – Load CrewAI, dotenv, and SerperDevTool.
- ✓ **Step 2:** Load Environment & Enable Web Search – Use `load_dotenv()` and SerperDevTool for real-time search.
- ✓ **Step 3:** Create Agents – Define roles and goals.
  - **Step 3.1:** Research Agent – Gathers the latest information using Serper Dev Tool.
  - **Step 3.2:** Summarization Agent – Extracts and summarizes key insights.
  - **Step 3.3:** Fact-Checking Agent – Verifies and corrects inaccuracies.
- ✓ **Step 4:** Define Tasks – Assign tasks to agents.
  - **Step 4.1:** Research Task – Use SerperDevTool to gather insights.
  - **Step 4.2:** Summarization Task – Convert findings into a structured summary.
  - **Step 4.3:** Fact-Checking Task – Validate and correct the summary.
- ✓ **Step 5:** Create a Crew – Group agents, assign tasks, and set sequential execution.
- ✓ **Step 6:** Run the Crew – Execute the workflow with an input topic.
- ✓ **Step 7:** Check the Response – Display the fact-checked, verified summary.

## AI Agents Part - 1

### Enabling Web Search

- ✓ Integrate Serper.dev for real-time web searches.
  - Get a free API key from Serper.dev.
  - Set the API key in the .env file for authentication.
- ✓ Next step: Define agents and their tasks!

```
OPENAI_API_KEY="sk-38381...."
```

```
SERPER_API_KEY="96f29..."
```

## AI Agents Part - 1

---

**Program Name** Steps from 1 to 7, Multi-Agent Research Assistant with CrewAI  
**Name** multi\_agent\_demo.py

```

print("Step 1: Importing the libraries")

from crewai import Agent, Task
from dotenv import load_dotenv
from crewai import Crew, Process

from crewai_tools import SerperDevTool


print("Step Special 1: Load Environment & Enable Web Search –
      Use load_dotenv() and SerperDevTool for real-time search.")

load_dotenv()

serper_dev_tool = SerperDevTool()


print("Step 2: Configure the llm: Using default LLM now")


print("Step 3.1: Research Agent ")

research_agent = Agent(
    role = "Internet Researcher",
    goal = "Find the most relevant and recent information
          about a given topic.",
    backstory = """You are a skilled researcher, adept at
                  navigating the internet and gathering high-quality,
                  reliable information.""",
    tools = [serper_dev_tool],
    verbose = True
)

```

## AI Agents Part - 1

```
print("Step 3.2: Summarization Agent")

summarizer_agent = Agent(
    role = "Content Summarizer",
    goal = "Condense the key insights from research into a
short and informative summary.",
    backstory = """You are an expert in distilling complex
information into concise, easy-to-read summaries.""",
    verbose = True
)

print("Step 3.3: Fact-Checking Agent")

fact_checker_agent = Agent(
    role = "Fact-Checking Specialist",
    goal = "Verify the accuracy of information and remove
any misleading or false claims.",
    backstory = """You are an investigative journalist with a
knack for validating facts,
ensuring that only accurate information is published."""",
    tools = [serper_dev_tool],
    verbose = True
)
```

## AI Agents Part - 1

```
print("Step 4.1: Research Task")

research_task = Task(
    description = """Use the SerperDevTool to search for the
    most relevant and recent data about {topic}."""
    "Extract the key insights from multiple sources.",
    agent = research_agent,
    tools = [serper_dev_tool],
    expected_output = "A detailed research report with key
    insights and source references."
)

print("Step 4.2: Summarization Task")

summarization_task = Task(
    description = "Summarize the research report into a
    concise and informative paragraph. "
    "Ensure clarity, coherence, and completeness.",
    agent = summarizer_agent,
    expected_output = "A well-structured summary with the
    most important insights."
)

print("Step 4.3: Fact-Checking Task")

fact_checking_task = Task(
    description = "Verify the summarized information for
    accuracy using the SerperDevTool. "
    "Cross-check facts with reliable sources and correct any
    errors.",
    agent = fact_checker_agent,
    tools = [serper_dev_tool],
    expected_output = "A fact-checked, verified summary of
    the research topic."
)
```

## AI Agents Part - 1

```
print("Step 5: Create a Crew")

research_crew = Crew(
    agents = [research_agent, summarizer_agent,
              fact_checker_agent],
    tasks = [research_task, summarization_task,
              fact_checking_task],
    process = Process.sequential,
    verbose = True
)

print("Step 6: Run the Crew")

result = research_crew.kickoff(inputs={"topic": "The impact of AI
on job markets"})

print("Step 7: Check the Response")

print("\nFinal Verified Summary:\n", result)
```

### Output

- Step 1:** Importing the libraries
- Step Special 1:** Load Environment & Enable Web Search – Use `load_dotenv()` and `SerperDevTool` for real-time search
- Step 2:** Configure the llm: Using default LLM now
- Step 3.1:** Research Agent
- Step 3.2:** Summarization Agent
- Step 3.3:** Fact-Checking Agent
- Step 4.1:** Research Task
- Step 4.2:** Summarization Task
- Step 4.3:** Fact-Checking Task
- Step 5:** Create a Crew
- Step 6:** Execute the Crew

## AI Agents Part - 1

### Step 7: Check the Response

#### Final Verified Summary:

The impact of Artificial Intelligence (AI) on job markets is significant, reshaping job roles and enhancing productivity. With AI automating traditional positions, particularly in technical fields, new job categories are emerging, forecasted to create 20 to 50 million jobs globally. AI is expected to boost workplace efficiency by up to 40%. However, this transition will vary across industries, with notable increases in AI job listings and demand for AI-literate professionals. Policymakers are urged to implement proactive measures, as AI could affect 40% of jobs worldwide, necessitating a fair transition into an AI-driven economy. Education will play a crucial role in preparing the workforce, emphasizing specialized skills in technology and AI. Overall, while AI presents challenges, it also offers opportunities for economic growth and workforce development if adequately managed.

## 1. Data Science – Machine Learning – Introduction

### Contents

<b>1. Machine learning.....</b>	<b>2</b>
<b>2. Is machine learning hard? .....</b>	<b>2</b>
<b>3. Program .....</b>	<b>2</b>
<b>4. What is an algorithm? .....</b>	<b>2</b>
<b>5. What is machine learning algorithm?.....</b>	<b>3</b>
<b>6. Diff b/n machine learning algorithm and normal algorithm? .....</b>	<b>3</b>
<b>7. Why Machine learning? .....</b>	<b>3</b>
<b>8. Machine learning definitions .....</b>	<b>4</b>
<b>9. Artificial intelligence vs Machine Learning vs Deep Learning.....</b>	<b>5</b>
9.1. Machine learning .....	6
9.2. Deep learning.....	7
9.3. Artificial intelligence .....	8
<b>10. Is computer identifying pictures on image like cat/dog? .....</b>	<b>9</b>
<b>11. Is human identifying picture on image like cat/dog? .....</b>	<b>9</b>
<b>12. Human vs computer .....</b>	<b>10</b>
<b>13. How a machine can take the decisions?.....</b>	<b>11</b>
<b>14. Past couple of decades.....</b>	<b>11</b>
<b>15. Human's follows this formula.....</b>	<b>11</b>
<b>16. Generally.....</b>	<b>12</b>
<b>17. How do machines think?.....</b>	<b>13</b>
<b>18. Real time examples .....</b>	<b>14</b>
<b>19. Gmail application .....</b>	<b>14</b>
<b>20. Banking loan application / credit card .....</b>	<b>15</b>
<b>21. Where machine learning helps? .....</b>	<b>16</b>
<b>22. Few of machine learning applications .....</b>	<b>16</b>
<b>23. Why machine learning required? .....</b>	<b>16</b>
<b>24. Technically speaking.....</b>	<b>17</b>

**1. Data Science – Machine Learning – Introduction****1. Machine learning**

- ✓ It is a technique which enables computers to learn automatically from past data.
- ✓ It is an approach to train the models.
- ✓ It is helpful to predict the future values.

**2. Is machine learning hard?**

- ✓ No, never
- ✓ Machine learning is very easy.
- ✓ It requires imagination, creativity, and a visual mind.
- ✓ Key point is, we should learn how to play with data and applying logic on data.
- ✓ This material helps how to do all these things.

**3. Program**

- ✓ A program is a group of instructions to perform the task; computer can execute these instructions to finish the task.
- ✓ This is very good approach for simple task

**4. What is an algorithm?**

- ✓ An algorithm is program which having group of instructions to perform a task.
  - Input + Logic = Output
  - Data + Program = Result

### 5. What is machine learning algorithm?

- ✓ Machine Learning algorithm is an application.
- ✓ It **learns** knowledge (patterns) automatically **from the data** without being explicitly programmed.
  - Data + Result = Program(Model)

### 6. Diff b/n machine learning algorithm and normal algorithm?

- ✓ Machine Learning algorithm learns knowledge (patterns) from the Data.
- ✓ Normal algorithm cannot learn anything on its own.

### 7. Why Machine learning?

- ✓ Currently every company is generating huge amount of the data.
- ✓ So, the volume of data is increasing on every day.
- ✓ Machine learning algorithms can extract information from data.
- ✓ The major use cases of Machine learning,
  - Creating models.
  - Getting deep insights & etc.
- ✓ Machine learning is going to be the center point of all fields.

## 8. Machine learning definitions



“ Machine learning is the study of computer algorithms that allow computer programs to automatically improve through experience.

~ Tom Mitchell,  
Machine Learning, McGraw Hill, 1997

Carnegie Mellon University  
Machine Learning

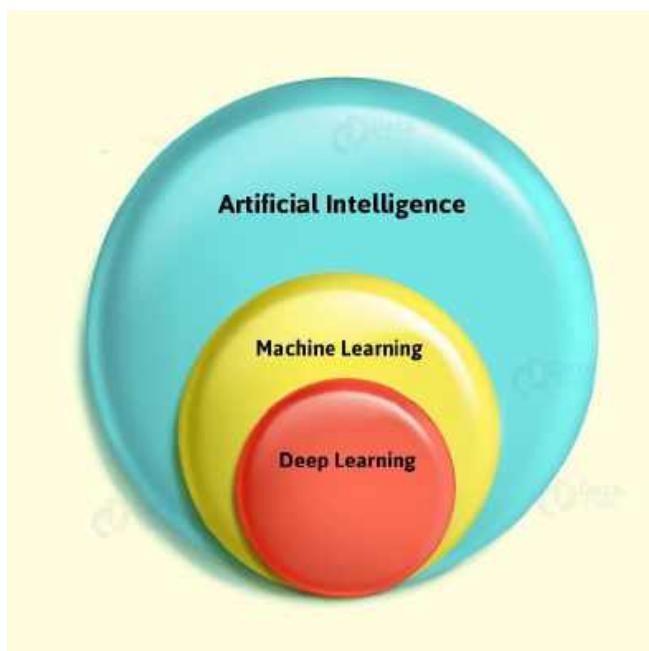
### Arthur Samuel definition

- ✓ Machine learning term was introduced by Arthur Samuel in the year of 1959 and he defined in this way,
  - Machine learning is,
    - Enables a machine to learn automatically from data,
    - Improve performance from experiences,
    - Predict things without being explicitly programmed

### Daniel's definition

- ✓ Teaching to computers how to learn by using data and experience, rather than by instructions.

## 9. Artificial intelligence vs Machine Learning vs Deep Learning



### 9.1. Machine learning

- ✓ Machine learning is a part of artificial intelligence
- ✓ Machine Learning is a technique of understand the data, learn from data and then apply what they have learnt to take next decision.

#### Examples

- ✓ Amazon using machine learning to give better product choice recommendations to their costumers based on their preferences.
- ✓ Netflix uses machine learning to give better suggestions to their users of the TV series or movie or shows that they would like to watch & many more

### 9.2. Deep learning

- ✓ Deep learning is actually a subset of machine learning.
- ✓ The main difference between deep and machine learning is, machine learning models works better but the model still needs some guidance.
- ✓ If a machine learning model returns an inaccurate prediction then the programmer needs to fix that problem explicitly.
- ✓ In the case of deep learning, the model does it by itself.

#### Example

- ✓ Automatic car driving system is a good example of deep learning.

### 9.3. Artificial intelligence

- ✓ AI is an ability of computer program to function like a human brain.
- ✓ Machine learning and deep learning are the subsets of AI
- ✓ The MOTO of AI is to replicate a human brain, the way a human brain thinks, works and functions.
- ✓ Currently AI is not yet fully implemented but we are very close to establish that too.

#### Example

- ✓ Sophia, the most advanced AI model present today.

### 10. Is computer identifying pictures on image like cat/dog?

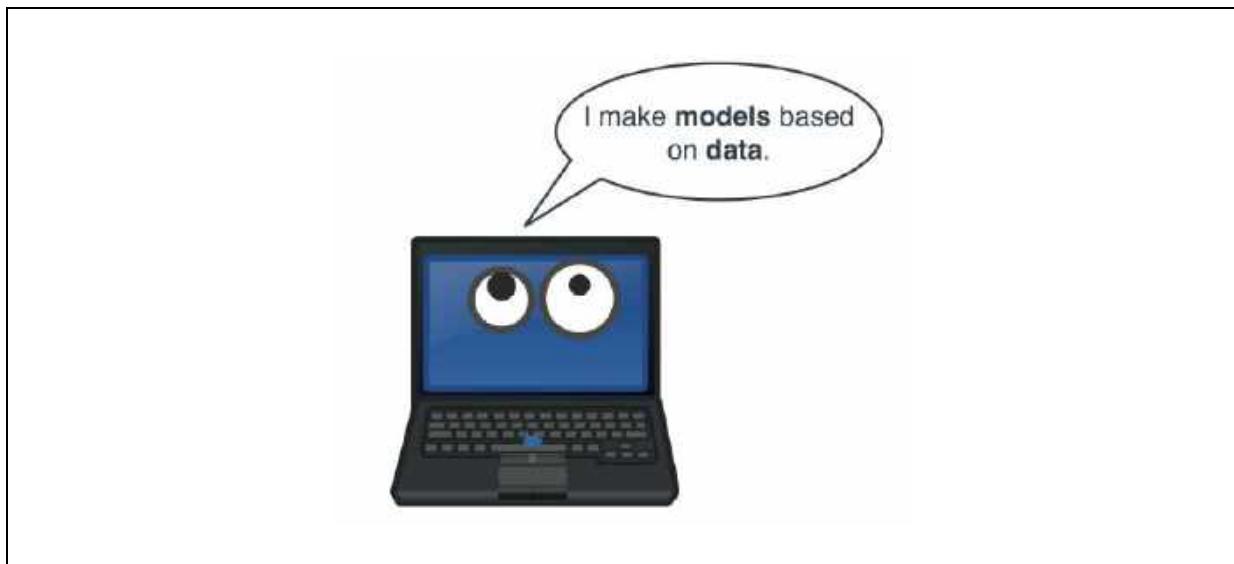
- ✓ Yes and it is really amazing task.
- ✓ If we giving lot of images to computer then computer can try to learn about the attributes to recognize images
- ✓ This is called as machine learning.
  - It is something like teaching to computer how to think like a human

### 11. Is human identifying picture on image like cat/dog?

- ✓ Absolutely YES right.
- ✓ During my childhood parents/teachers taught like how to recognize a cat/dog/tiger/lion/etc
- ✓ So, in real life, if we see cat/dog/tiger/lion (sorry tiger/lion I have seen in Zoo but not directly) then we can easy to recognize without much effort.
- ✓ So, a human can take the decision based on experience.

## 12. Human vs computer

- ✓ Humans take the **decisions** based on the **experience**
- ✓ Computer can create **models** by using data



**13. How a machine can take the decisions?**

- ✓ First we need to understand human decision making process before understanding about how a computer takes the decision.

**14. Past couple of decades**

- ✓ From past two decades most of the companies are digitalized.

**15. Human's follows this formula...**

- ✓ All humans used to follow some patterns or formulas to take the next best action.

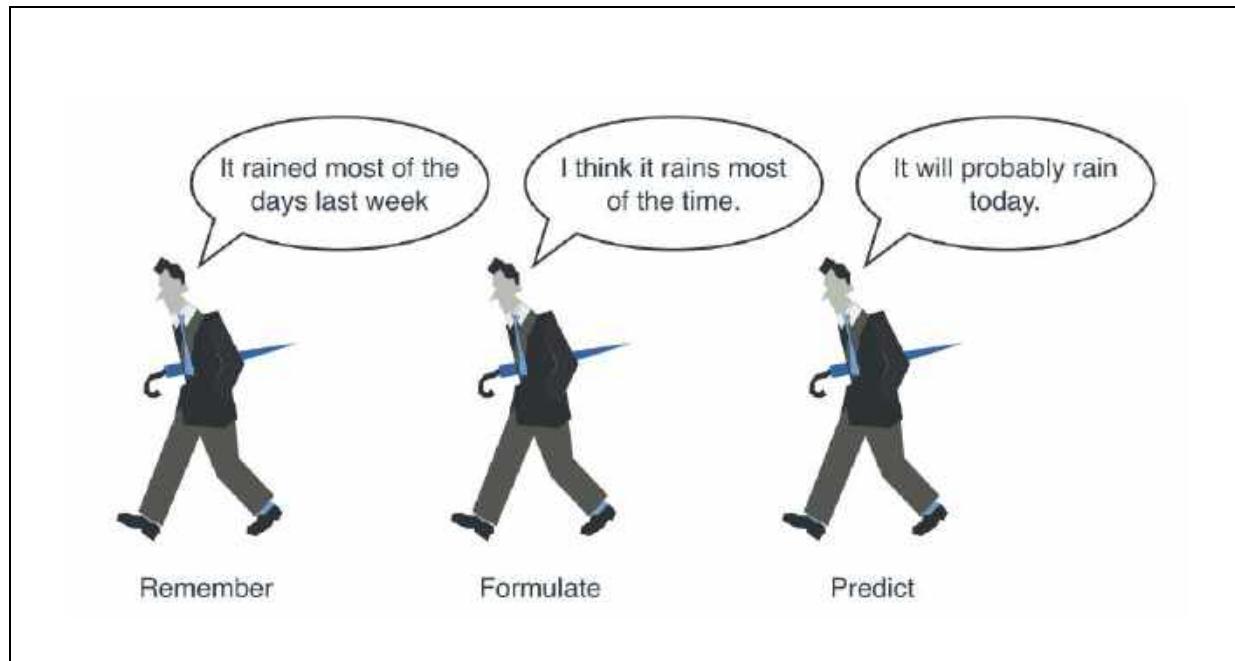
- Remember
  - Formulate
  - Predict

## 16. Generally

- ✓ We **remember** past situations that were similar.
- ✓ We **formulate** a general rule.
- ✓ We use this rule to **predict** what will happen if we take a certain decision.

### Example

- ✓ For supposed to answer for a question like, "**Will it rain today? Or not**", then we will follow one **process** to answer this question right.
- ✓ We may be right or wrong, but at least, we are trying to make an accurate prediction.



### 17. How do machines think?

- ✓ The goal is to make the computer think the way we think, namely, how we use the remember formulate-predict framework.
- ✓ Here are the points what the computer can understand the points

#### Steps

- ✓ Remember
  - Look at the given **HUGE** data.
- ✓ Formulate
  - Go through rules and formulas, and check which one fits for the data
- ✓ Predict
  - Use the rule to make predictions about future data.

### 18. Real time examples

- ✓ Gmail application
- ✓ Banking loan application or credit card eligibility & etc

### 19. Gmail application

- ✓ *Gmail* is the very well-known example.
- ✓ Hope we are all known about Spam folder too
- ✓ **Spam**
  - It is the common term used for junk or unwanted email, such as chain letters, promotions, and so on
- ✓ **Ham**
  - It is the common term used for non-spam mail means useful mail
- ✓ In this scenario machine learning algorithm is working to filter spam or not
- ✓ Machine learning algorithm,
  - Understand the previous mails(data)
  - Based on the previous mail creates some models.
  - These models applies on upcoming mails and predict whether it is spam or not

### 20. Banking loan application / credit card

- ✓ Banking loan application, this is the very common application hope everyone knows
- ✓ Once we applied for loan by submitting required documents(data) like pay slip, past 6 months banking statement and etc
- ✓ Then banking guys will run one application over the given data.
- ✓ This is Machine learning algorithm
- ✓ This algorithm passes through the data and predicts the weather the loan/credit card sanctioned or not.

#### Note

- ✓ Many other applications are using machine learning
- ✓ I would like to say, Machine learning is everywhere 😊

**21. Where machine learning helps?**

- ✓ From past two decades most of the companies are digitalized.
- ✓ So, here data is generated more and more.
- ✓ So, in this case machine learning helps us to develop predictive models and automate several things.
- ✓ Assuming that we had past couple of year's **amazon** transactions data.
- ✓ Some questions:
  - Wanted to know how the business was in last couple of years
  - Wanted to take best decisions to improve the business and etc
- ✓ To answer above questions, we need to,
  - Gather the data
  - Process the data
  - Take the decisions.

**22. Few of machine learning applications**

- ✓ Text processing
- ✓ Speech Recognition
- ✓ Traffic prediction
- ✓ Product recommendations
- ✓ Self-driving cars
- ✓ Email spam and malware filtering
- ✓ Online fraud detection
- ✓ Weather forecasting and prediction & etc

**23. Why machine learning required?**

- ✓ As a human we cannot access huge amount of data manually to process.
- ✓ We can train machine learning algorithms by providing huge amount of data.
- ✓ Machine learning algorithm travel through this data in order to learn about data, it create the model and predict results automatically for new data

## 24. Technically speaking...

- ✓ Technically speaking,
  - WITH THE help of historical data
  - MACHINE LEANRING algorithms
  - BUILD a mathematical MODEL
  - that helps in making PREDICTIONS
  - Without being EXPLICITELY PROGRAMMER

**1. Data Science – Machine Learning – Introduction****Contents**

<b>1. Machine learning.....</b>	<b>2</b>
<b>2. Is machine learning hard? .....</b>	<b>2</b>
<b>3. Program .....</b>	<b>2</b>
<b>4. What is an algorithm? .....</b>	<b>2</b>
<b>5. What is machine learning algorithm?.....</b>	<b>3</b>
<b>6. Diff b/n machine learning algorithm and normal algorithm? .....</b>	<b>3</b>
<b>7. Why Machine learning? .....</b>	<b>3</b>
<b>8. Machine learning definitions .....</b>	<b>4</b>
<b>9. Artificial intelligence vs Machine Learning vs Deep Learning.....</b>	<b>5</b>
9.1. Machine learning .....	6
9.2. Deep learning.....	7
9.3. Artificial intelligence .....	8
<b>10. Is computer identifying pictures on image like cat/dog? .....</b>	<b>9</b>
<b>11. Is human identifying picture on image like cat/dog? .....</b>	<b>9</b>
<b>12. Human vs computer .....</b>	<b>10</b>
<b>13. How a machine can take the decisions?.....</b>	<b>11</b>
<b>14. Past couple of decades.....</b>	<b>11</b>
<b>15. Human's follows this formula.....</b>	<b>11</b>
<b>16. Generally.....</b>	<b>12</b>
<b>17. How do machines think?.....</b>	<b>13</b>
<b>18. Real time examples .....</b>	<b>14</b>
<b>19. Gmail application .....</b>	<b>14</b>
<b>20. Banking loan application / credit card .....</b>	<b>15</b>
<b>21. Where machine learning helps? .....</b>	<b>16</b>
<b>22. Few of machine learning applications.....</b>	<b>16</b>
<b>23. Why machine learning required? .....</b>	<b>16</b>
<b>24. Technically speaking.....</b>	<b>17</b>

**1. Data Science – Machine Learning – Introduction****1. Machine learning**

- ✓ It is a technique which enables computers to learn automatically from past data.
- ✓ It is an approach to train the models.
- ✓ It is helpful to predict the future values.

**2. Is machine learning hard?**

- ✓ No, never
- ✓ Machine learning is very easy.
- ✓ It requires imagination, creativity, and a visual mind.
- ✓ Key point is, we should learn how to play with data and applying logic on data.
- ✓ This material helps how to do all these things.

**3. Program**

- ✓ A program is a group of instructions to perform the task; computer can execute these instructions to finish the task.
- ✓ This is very good approach for simple task

**4. What is an algorithm?**

- ✓ An algorithm is program which having group of instructions to perform a task.
  - Input + Logic = Output
  - Data + Program = Result

### 5. What is machine learning algorithm?

- ✓ Machine Learning algorithm is an application.
- ✓ It **learns** knowledge (patterns) automatically **from the data** without being explicitly programmed.
  - Data + Result = Program(Model)

### 6. Diff b/n machine learning algorithm and normal algorithm?

- ✓ Machine Learning algorithm learns knowledge (patterns) from the Data.
- ✓ Normal algorithm cannot learn anything on its own.

### 7. Why Machine learning?

- ✓ Currently every company is generating huge amount of the data.
- ✓ So, the volume of data is increasing on every day.
- ✓ Machine learning algorithms can extract information from data.
- ✓ The major use cases of Machine learning,
  - Creating models.
  - Getting deep insights & etc.
- ✓ Machine learning is going to be the center point of all fields.

## 8. Machine learning definitions



“ Machine learning is the study of computer algorithms that allow computer programs to automatically improve through experience.

~ Tom Mitchell,  
Machine Learning, McGraw Hill, 1997

Carnegie Mellon University  
Machine Learning

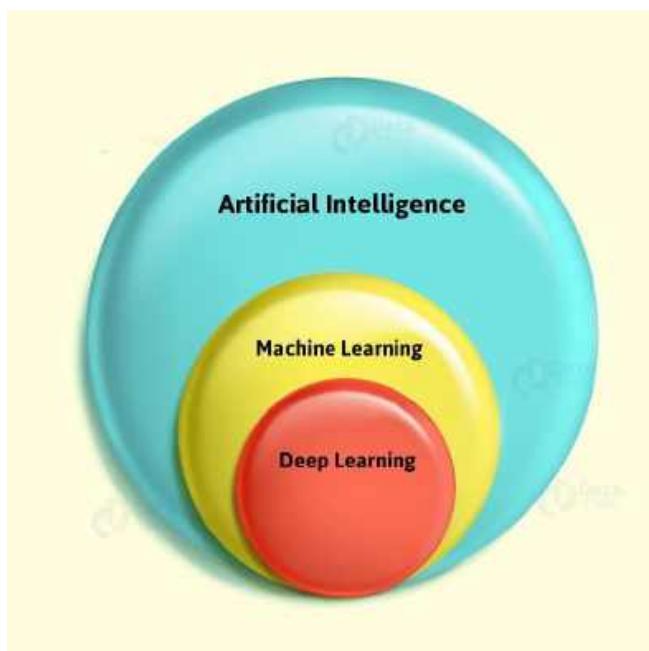
### Arthur Samuel definition

- ✓ Machine learning term was introduced by Arthur Samuel in the year of 1959 and he defined it this way,
  - Machine learning is,
    - Enables a machine to learn automatically from data,
    - Improve performance from experiences,
    - Predict things without being explicitly programmed

### Daniel's definition

- ✓ Teaching computers how to learn by using data and experience, rather than by instructions.

## 9. Artificial intelligence vs Machine Learning vs Deep Learning



### 9.1. Machine learning

- ✓ Machine learning is a part of artificial intelligence
- ✓ Machine Learning is a technique of understand the data, learn from data and then apply what they have learnt to take next decision.

#### Examples

- ✓ Amazon using machine learning to give better product choice recommendations to their costumers based on their preferences.
- ✓ Netflix uses machine learning to give better suggestions to their users of the TV series or movie or shows that they would like to watch & many more

### 9.2. Deep learning

- ✓ Deep learning is actually a subset of machine learning.
- ✓ The main difference between deep and machine learning is, machine learning models works better but the model still needs some guidance.
- ✓ If a machine learning model returns an inaccurate prediction then the programmer needs to fix that problem explicitly.
- ✓ In the case of deep learning, the model does it by itself.

#### Example

- ✓ Automatic car driving system is a good example of deep learning.

### 9.3. Artificial intelligence

- ✓ AI is an ability of computer program to function like a human brain.
- ✓ Machine learning and deep learning are the subsets of AI
- ✓ The MOTO of AI is to replicate a human brain, the way a human brain thinks, works and functions.
- ✓ Currently AI is not yet fully implemented but we are very close to establish that too.

#### Example

- ✓ Sophia, the most advanced AI model present today.

### 10. Is computer identifying pictures on image like cat/dog?

- ✓ Yes and it is really amazing task.
- ✓ If we giving lot of images to computer then computer can try to learn about the attributes to recognize images
- ✓ This is called as machine learning.
  - It is something like teaching to computer how to think like a human

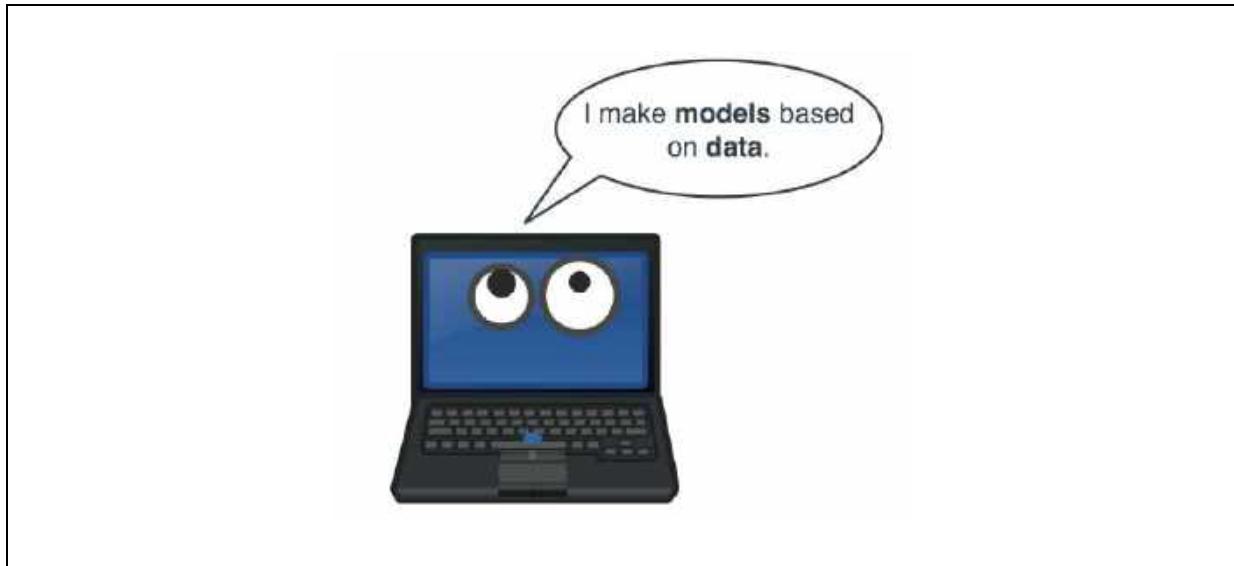
### 11. Is human identifying picture on image like cat/dog?

- ✓ Absolutely YES right.
- ✓ During my childhood parents/teachers taught like how to recognize a cat/dog/tiger/lion/etc
- ✓ So, in real life, if we see cat/dog/tiger/lion (sorry tiger/lion I have seen in Zoo but not directly) then we can easy to recognize without much effort.
- ✓ So, a human can take the decision based on experience.

## Data Science – Machine Learning - Introduction

### 12. Human vs computer

- ✓ Humans take the **decisions** based on the **experience**
- ✓ Computer can create **models** by using data



### 13. How a machine can take the decisions?

- ✓ First we need to understand human decision making process before understanding about how a computer takes the decision.

### 14. Past couple of decades

- ✓ From past two decades most of the companies are digitalized.

### 15. Human's follows this formula...

- ✓ All humans used to follow some patterns or formulas to take the next best action.

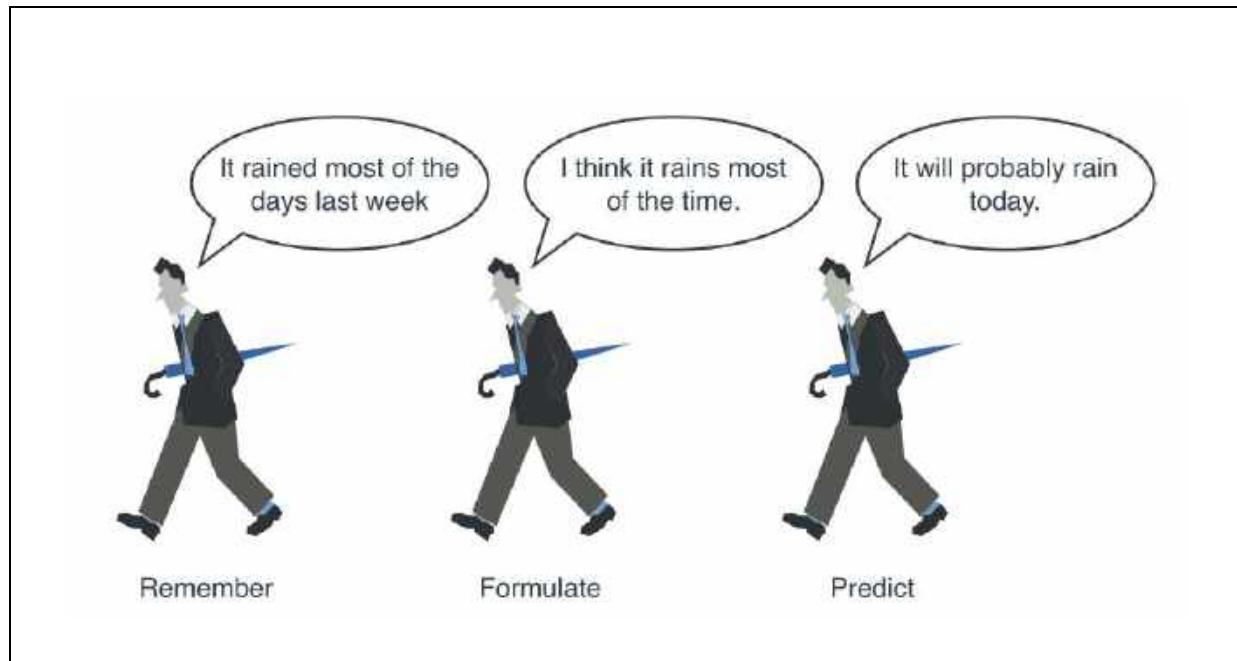
- Remember
  - Formulate
  - Predict

## 16. Generally

- ✓ We **remember** past situations that were similar.
- ✓ We **formulate** a general rule.
- ✓ We use this rule to **predict** what will happen if we take a certain decision.

### Example

- ✓ For supposed to answer for a question like, "**Will it rain today? Or not**", then we will follow one **process** to answer this question right.
- ✓ We may be right or wrong, but at least, we are trying to make an accurate prediction.



### 17. How do machines think?

- ✓ The goal is to make the computer think the way we think, namely, how we use the remember formulate-predict framework.
- ✓ Here are the points what the computer can understand the points

#### Steps

- ✓ Remember
  - Look at the given **HUGE** data.
- ✓ Formulate
  - Go through rules and formulas, and check which one fits for the data
- ✓ Predict
  - Use the rule to make predictions about future data.

### 18. Real time examples

- ✓ Gmail application
- ✓ Banking loan application or credit card eligibility & etc

### 19. Gmail application

- ✓ *Gmail* is the very well-known example.
- ✓ Hope we are all known about Spam folder too
- ✓ **Spam**
  - It is the common term used for junk or unwanted email, such as chain letters, promotions, and so on
- ✓ **Ham**
  - It is the common term used for non-spam mail means useful mail
- ✓ In this scenario machine learning algorithm is working to filter spam or not
- ✓ Machine learning algorithm,
  - Understand the previous mails(data)
  - Based on the previous mail creates some models.
  - These models applies on upcoming mails and predict whether it is spam or not

### 20. Banking loan application / credit card

- ✓ Banking loan application, this is the very common application hope everyone knows
- ✓ Once we applied for loan by submitting required documents(data) like pay slip, past 6 months banking statement and etc
- ✓ Then banking guys will run one application over the given data.
- ✓ This is Machine learning algorithm
- ✓ This algorithm passes through the data and predicts the weather the loan/credit card sanctioned or not.

#### Note

- ✓ Many other applications are using machine learning
- ✓ I would like to say, Machine learning is everywhere 😊

### 21. Where machine learning helps?

- ✓ From past two decades most of the companies are digitalized.
- ✓ So, here data is generated more and more.
- ✓ So, in this case machine learning helps us to develop predictive models and automate several things.
- ✓ Assuming that we had past couple of year's **amazon** transactions data.
- ✓ Some questions:
  - Wanted to know how the business was in last couple of years
  - Wanted to take best decisions to improve the business and etc
- ✓ To answer above questions, we need to,
  - Gather the data
  - Process the data
  - Take the decisions.

### 22. Few of machine learning applications

- ✓ Text processing
- ✓ Speech Recognition
- ✓ Traffic prediction
- ✓ Product recommendations
- ✓ Self-driving cars
- ✓ Email spam and malware filtering
- ✓ Online fraud detection
- ✓ Weather forecasting and prediction & etc

### 23. Why machine learning required?

- ✓ As a human we cannot access huge amount of data manually to process.
- ✓ We can train machine learning algorithms by providing huge amount of data.
- ✓ Machine learning algorithm travel through this data in order to learn about data, it create the model and predict results automatically for new data

## 24. Technically speaking...

- ✓ Technically speaking,
  - WITH THE help of historical data
  - MACHINE LEANRING algorithms
  - BUILD a mathematical MODEL
  - that helps in making PREDICTIONS
  - Without being EXPLICITELY PROGRAMMER

**2. Data Science – Machine Learning – Terminology****Contents**

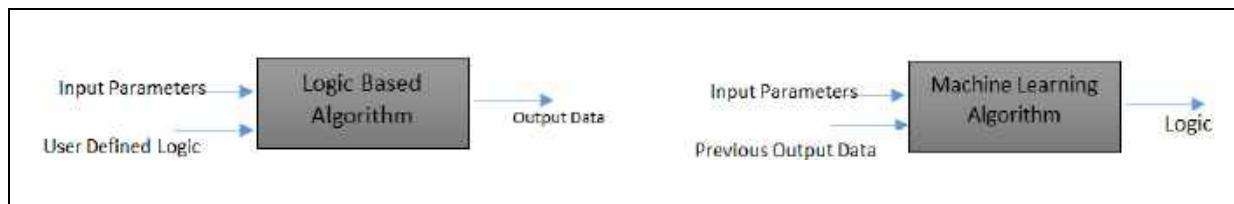
<b>1. Machine learning.....</b>	<b>2</b>
<b>2. Model.....</b>	<b>2</b>
<b>3. Train .....</b>	<b>3</b>
<b>4. How to train?.....</b>	<b>3</b>
<b>5. Model learns.....</b>	<b>3</b>
<b>6. Pattern .....</b>	<b>3</b>
<b>7. Before trained .....</b>	<b>3</b>
<b>8. After trained .....</b>	<b>3</b>
<b>9. Test the model.....</b>	<b>3</b>
<b>10. Model deployment.....</b>	<b>4</b>
<b>11. Towards target value or classifier.....</b>	<b>4</b>
<b>12. General example .....</b>	<b>4</b>
<b>13. Smart application .....</b>	<b>4</b>
<b>14. Normal application.....</b>	<b>5</b>
<b>15. Smart application .....</b>	<b>6</b>
<b>16. Use case .....</b>	<b>6</b>
<b>17. Accuracy testing .....</b>	<b>7</b>

## Data Science – Machine Learning - Terminology

### 2. Data Science – Machine Learning – Terminology

#### 1. Machine learning

- ✓ Machine learning is a technique which enables computers to learn automatically from past data.
- ✓ Machine Learning is an approach to train the **models**.
- ✓ Machine learning is helpful to predict the future values.



#### 2. Model

- ✓ In simple terms, a model is a representation of reality.
- ✓ A model is created to do a specific purpose

#### Example

- ✓ General example, a world map is a model for this physical world or universe.
  - The creator of world map have done deep analysis, understands, visualized with simple representation.

#### In terms of Machine learning,

- ✓ A model can be a,
  - A piece of code or program
  - A mathematical formula
  - A program + mathematical formula

## Data Science – Machine Learning - Terminology

---

### 3. Train

- ✓ We need to **train** the model

### 4. How to train?

- ✓ We need to use **data** to train the model

### 5. Model learns

- ✓ Model will learn **pattern** from data.

### 6. Pattern

- ✓ Patterns means a simple knowledge gained during training.
- ✓ So, model learnt patterns from data during training

### 7. Before trained

- ✓ Before training model, model doesn't know about the patterns.

### 8. After trained

- ✓ After training model, model knows about the patterns.

### 9. Test the model

- ✓ Once after model got trained then that model has to be tested to **check the accuracy**.
- ✓ If accuracy is good than the model works well

### 10. Model deployment

- ✓ Finally we need to deploy knowledge acquired by the model
- ✓ These models also called as model fit

### 11. Towards target value or classifier

- ✓ Models learnt patterns from data towards target value.
- ✓ So, let's try to understand what it means.

### 12. General example

- ✓ Assuming that one of my friend is introducing his friend as
  - His name is Prasad working in X Company and having 3 years of experience.
- ✓ When I heard about his experience I can guess/predict his salary (target value).
- ✓ So, our brain is already trained to guess some attributes from past data(old friends)
- ✓ So, we have patterns towards to target value

### 13. Smart application

- ✓ Once accuracy satisfied then we need to deploy the model.
- ✓ Once model deployed then that application will become as smart application

### 14. Normal application

- ✓ Assuming that we have created an application by using Java/dotnet/Python.
- ✓ It's just called as normal software application because this application is just works on instruction based.
- ✓ Instruct based means,
  - If x value is greater than y value, then prints x value
  - If x value is less than y value, then prints y value

#### Make a note

- ✓ Normal application cannot think like human to take decisions
- ✓ Normal applications cannot take own decisions

### 15. Smart application

- ✓ A smart application is an application which thinks like a human to take the decisions.
- ✓ Smart application is intelligence based.

### 16. Use case

- ✓ Whenever you send email automatically Gmail application is predicting whether it is a spam or not spam.
- ✓ That means Gmail application is well trained and captures the patterns about mail type like,
  - Spam mail
  - Promotion mail
  - Social media mail & etc
- ✓ So, Gmail application knows patterns about to recognize the type of mail.

## 17. Accuracy testing

- ✓ Before deploying the model, it needs to be tested in lot of dimensions.
- ✓ If model is getting good predictions then we need to deploy.

### Process in machine learning

- ✓ We need to **train** the model.
- ✓ During training model acquires the **knowledge**.
- ✓ This knowledge is called as **model fit**
- ✓ Once after model fit produced we need to **test** to check the **accuracy**.
- ✓ If satisfied with accuracy then we need to **deploy** the model

### Data is important

- ✓ We need to train the model with data
- ✓ During this training model will go through the data and understand the patterns about data.
- ✓ These patterns helps to predict the result on **new data**

**3. Data Science – Machine Learning – Data & ML Algorithm****Contents**

<b>1. Data.....</b>	<b>2</b>
<b>2. Data in table .....</b>	<b>3</b>
3. Row .....	3
4. Column .....	3
5. Cell.....	3
<b>6. Statistical Learning Perspective .....</b>	<b>4</b>
6.1. Example.....	4
<b>7. Why Machine learning algorithms learn this function?.....</b>	<b>6</b>
<b>8. Terminology in statistics, .....</b>	<b>7</b>
<b>9. Computer Science Perspective.....</b>	<b>8</b>
<b>10. Models and Algorithms.....</b>	<b>9</b>

**3. Data Science – Machine Learning – Data & ML Algorithm****1. Data**

- ✓ It is a collection of facts.
- ✓ Facts mean,
  - Alphabets
  - Numbers
  - Alphanumeric
  - Symbol

◆	A	B	C	D
1		Column 1	Column 2	Column 3
2	Row 1	2.2	2.3	1
3	Row 2	2.3	2.6	0
4	Row 3	2.1	2	1
5				

## 2. Data in table

- ✓ Generally, tables having data like rows, columns and cells
- ✓ Tabular data is very easy to understand.

## 3. Row

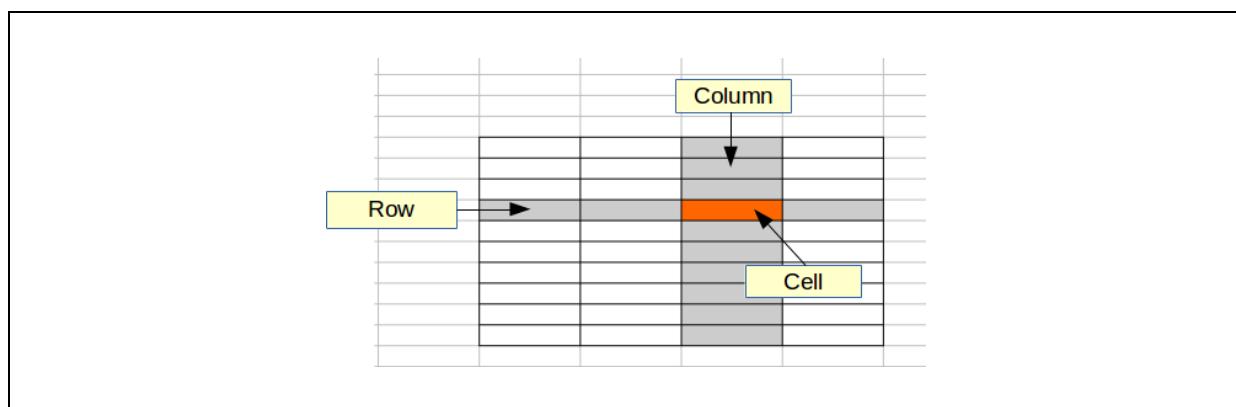
- ✓ A row describes a single entity or observation.
- ✓ A row is also called a record.

## 4. Column

- ✓ A column is a vertical group of cells within a table.
- ✓ A column having same type of values.
- ✓ In column we can store the data like weights, heights and prices etc info.

## 5. Cell

- ✓ A cell is a single value in a row and column.
- ✓ It may be a real value (1.5) an integer (2) or a category (red).



## 6. Statistical Learning Perspective

- ✓ In statistical learning perspective data means,
  - It is the context of a **hypothetical function** ( $f$ ) that the machine learning algorithm is **trying to learn**.

### 6.1. Example

- ✓ Assuming that, there is a farmer in village and initially he had 1 acre land.
- ✓ While forming rice it get yields as below,

Area	Rice packets
1	10
2	20
3	30

If former having 4 acres land then how many rice packets it yields?

**Answer**

- ✓ A non-technical guy also can answer as, we will get **40** packets for **4** acres of land.

Area	Rice packets
1	10
2	20
3	30
4	?

**Simple calculation**

- ✓ Formula is,

○ 1 acre	$=>$	$1 \times 10$	$=$	10
○ 2 acre	$=>$	$2 \times 10$	$=$	20
○ 3 acre	$=>$	$3 \times 10$	$=$	30
○ 4 acre	$=>$	<b>4 x 10</b>	$=$	<b>40</b>

- ✓ Simple Formula is,
  - $\text{Rice\_yield} = \text{land\_size} \times 10$

## 7. Why Machine learning algorithms learn this function?

- ✓ To predict output for the given input.

Output =  $f(\text{Input})$

- ✓ Columns are called as input variables.
- ✓ Predicted result is called as output variable or response variable.

Output Variable =  $f(\text{Input Variables})$

◆	A	B	C
1	X1	X2	Y
2	2.2	2.3	1
3	2.3	2.6	0
4	2.1	2	1
5			

- ✓ Input data may have more than one input variable.
- ✓ The group of input variables are called as input vector.

Output Variable =  $f(\text{Input Vector})$

### 8. Terminology in statistics,

- ✓ Input variables are called as Independent variables.
- ✓ Output variable are called as the Dependent variable.
- ✓ Here the output is dependent on a function of input.

Dependent Variable =  $f(\text{Independent Variables})$

- ✓ Input variable representing with  $X$
- ✓ Output variable representing with  $y$

$Y = f(X)$

- ✓ If we have multiple input variables then it's represent as input vector  $x_1, x_2, x_3$  for the data

## 9. Computer Science Perspective

- ✓ A row means it's an entity/observation-instance/object in a table.
- ✓ A Column is called as an attribute.
- ✓ During modeling and predictions,
  - Input is called as input attribute.
  - Output is called as output attributes.

Output Attribute = Program(Input Attributes)

◆	A	B	C	D
1		Attribute 1	Attribute 2	Output Attribute
2	Instance 1	2.2	2.3	1
3	Instance 2	2.3	2.6	0
4	Instance 3	2.1	2	1
5				

Output = Program(Input Features)

Prediction output = Program(Instance)

### 10. Models and Algorithms

- ✓ A model is a specific representation learned **from data** and the algorithm as the process of learning it.

Model = Algorithm(Data)

## Data Science – Machine Learning – Learning Function

---

### 4. Data Science – Machine Learning – Learning Function

#### Contents

1. Learning a Function .....	2
2. Purpose of learning-function .....	2
3. Machine learning algorithms .....	3

## Data Science – Machine Learning – Learning Function

### 4. Data Science – Machine Learning – Learning Function

#### 1. Learning a Function

- ✓ Machine learning algorithms are described as,
  - Learning a target function ( $f$ ) which maps input variables ( $X$ ) to output variable ( $Y$ )

$$\text{Output} = f(\text{Input})$$

$$Y = f(X)$$

- ✓ This function also having some error.
- ✓ This error is independent to the input data

$$Y = f(X) + \text{error}$$

#### 2. Purpose of learning-function

- ✓ The purpose of this learning-function is to make predictions.
- ✓ The function which is having less error, that function will make the accurate predictions.
- ✓ This is called predictive modeling.

### 3. Machine learning algorithms

- ✓ Machine learning algorithms are techniques for estimating the target function ( $f$ ).
- ✓ This function helps to predict the **output variable (Y)** for the given **input variables (X)**.
- ✓ Different machine learning algorithms having different assumptions about the form of the function, such as whether it is linear or nonlinear.

# Data Science – Machine Learning – Types of the Models

---

## 5. Data Science – Machine Learning – Types of the Models

### Contents

<b>1. Feature and label.....</b>	2
1.1. Features .....	2
1.2. Label .....	3
2. Label Example .....	3
<b>3. Labelled and unlabeled data.....</b>	4
3.1. Labelled Data: .....	4
3.2. Unlabeled Data .....	5
<b>4. Supervised learning definition.....</b>	6
<b>5. Unsupervised learning definition .....</b>	7
<b>6. Supervised learning example.....</b>	8
<b>7. Remember – Formulate - Predict. ....</b>	9
<b>8. Types of Supervised learning models.....</b>	11
8.1. Regression models .....	12
8.2. Classification models.....	12
<b>9. Unsupervised learning example .....</b>	13
<b>10. Types of unsupervised learning models.....</b>	14
10.1. Clustering .....	15
10.2. Dimensionality reduction.....	16

## 5. Data Science – Machine Learning – Types of the Models

- ✓ In Machine learning there are mainly 3 types of models exists,

- ✓ Supervised learning
- ✓ Unsupervised learning
- ✓ Reinforcement learning

### Best tip

- ✓ Before understanding the types of algorithms let's try to understand terminology

## 1. Feature and label

### 1.1. Features

- ✓ Features are simply the columns of the table
- ✓ These features describes the about the data
- ✓ In the given data, Age, Gender, Experience and Salary are features

Age	Gender	Experience	Salary
20	M	4	40000
24	F	5	50000

### 1.2. Label

- ✓ The output we will get after training the model is called as a **Label**
  
- ✓ Requirement
  - Suppose I wanted to predict the salary who had 6 years of experience
  - We prepared a model and that model predicted the salary.
  - Here salary is called as a **label**
  
- ✓ Simple
  - We are trying to predict a feature based on the others, that feature is the label.

### 2. Label Example

- ✓ If we are trying to predict the **type of pet** for example cat or dog based on information then that is the **label**.
- ✓ If we are trying to predict if the pet is **sick or healthy** based on symptoms and other information, then that is the **label**.
- ✓ If we are trying to predict the **age of the pet**, then the age is the **label**.

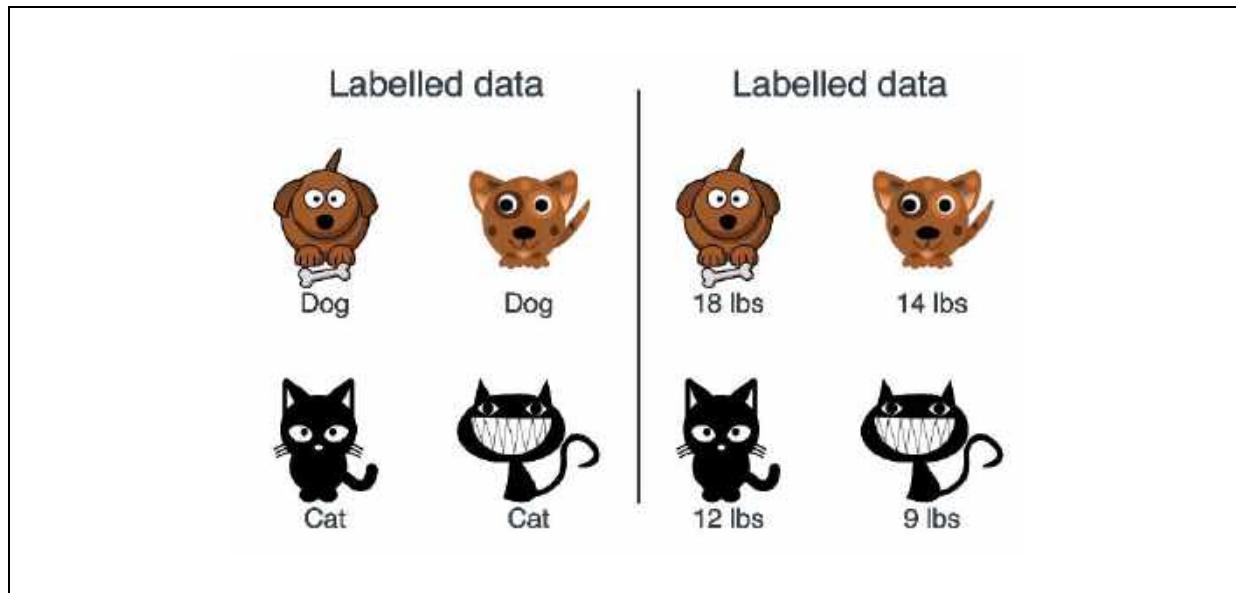
## Data Science – Machine Learning – Types of the Models

### 3. Labelled and unlabeled data

- ✓ According to the label, data is divided into two types
  - Labelled data
  - Unlabeled data

#### 3.1. Labelled Data:

- ✓ Labelled data comes with a tag or label, like a name, a type, or a number.



### 3.2. Unlabeled Data

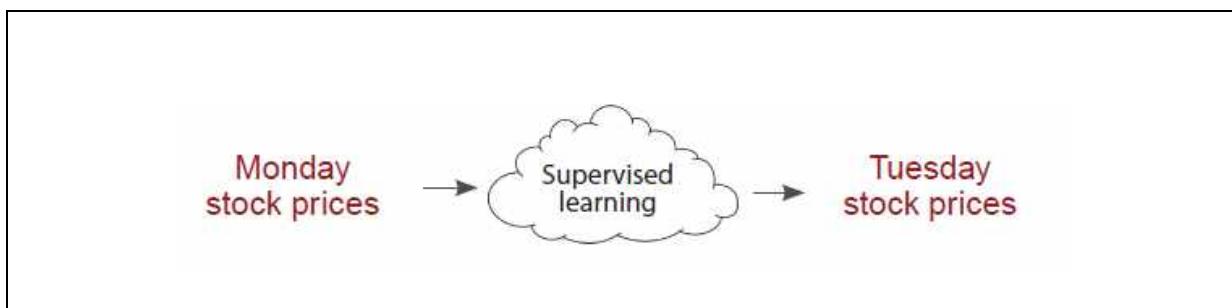
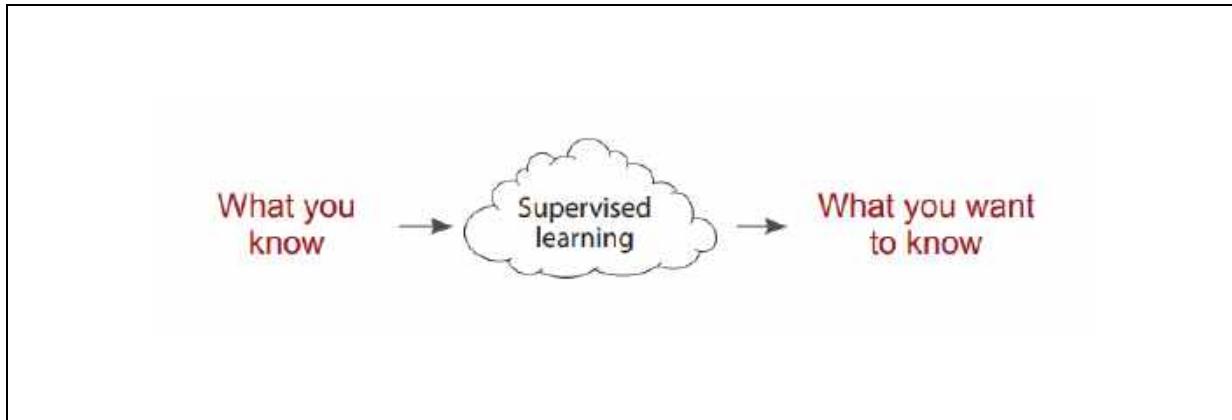
- ✓ Unlabeled data have no tag or label

Unlabelled data



#### 4. Supervised learning definition

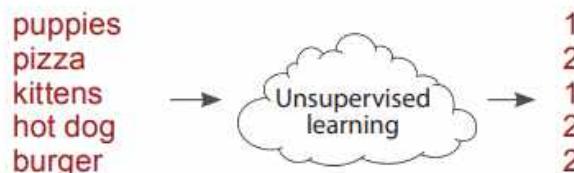
- ✓ In supervised learning, we will train the models by using input **features** and **labels**



## Data Science – Machine Learning – Types of the Models

### 5. Unsupervised learning definition

- ✓ In unsupervised learning, we will train the models by using only input features and there is no labels



#### Unsupervised machine learning

Unsupervised learning groups your data.

## 6. Supervised learning example

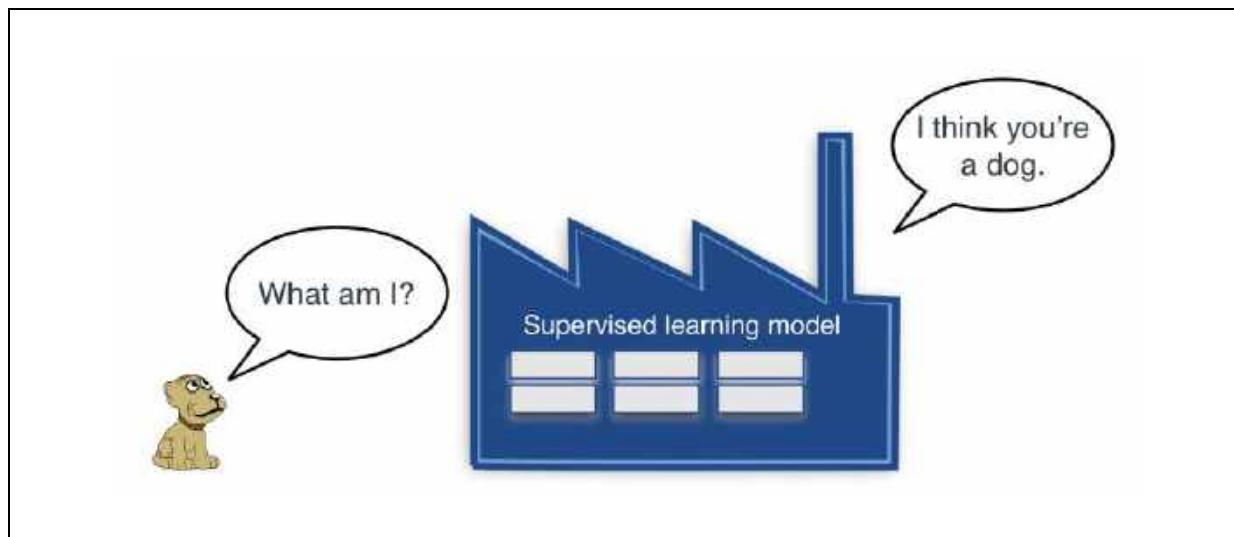
- ✓ Now a days it is very commonly using for all applications
- ✓ Supervised learning = **features + label**.

### Examples

- ✓ Image recognition,
- ✓ Text processing,
- ✓ Recommendation systems & many more.

### Scenario

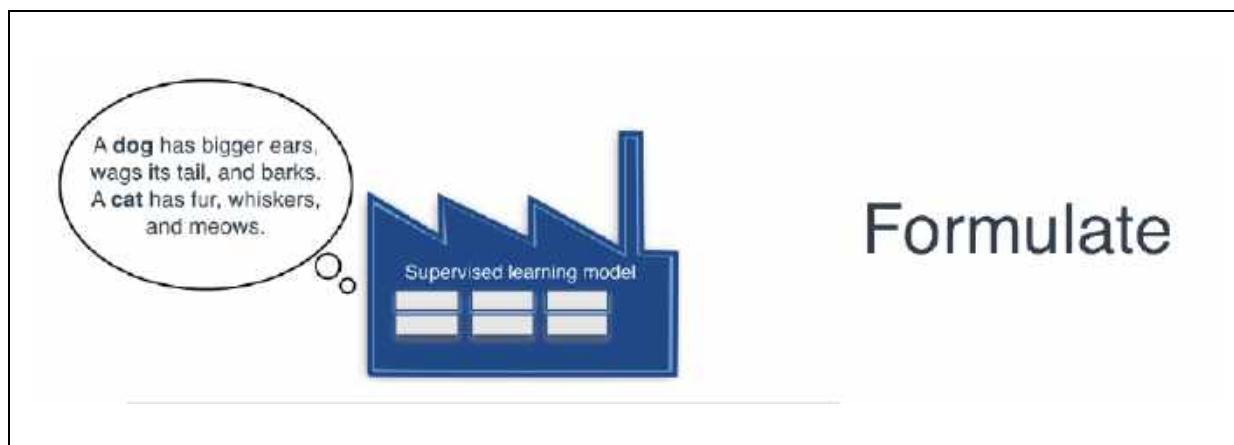
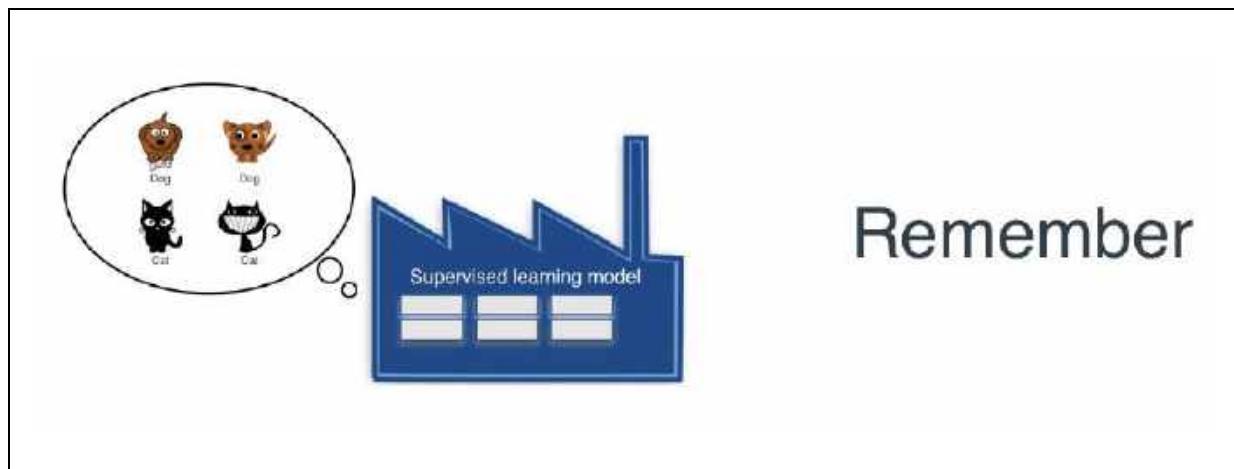
- ✓ In previous images, we have an images data about dogs and cats.
- ✓ **Labels** in the image are '**dog**' and '**cat**'.
- ✓ The machine learning model use previous data in order to predict the label of **new data points**.



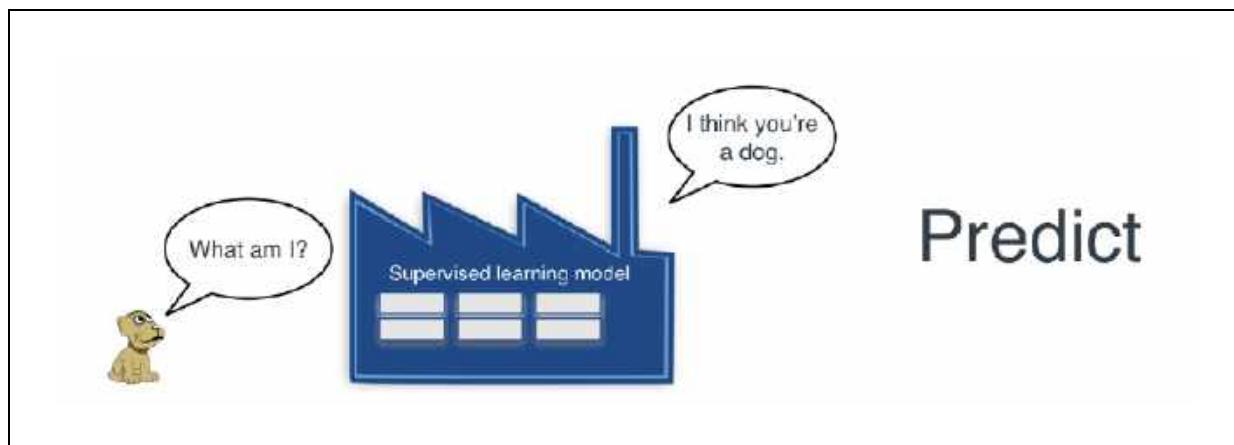
## Data Science – Machine Learning – Types of the Models

### 7. Remember – Formulate - Predict.

- ✓ Supervised learning works in remember – formulate – predict
- ✓ The model first remembers the dataset of dogs and cats.
- ✓ Then model formulates a model for **what is a dog** and **what is a cat**.
- ✓ Whenever a **new image comes** in, the model makes a prediction about what the label of the image weather cat or dot



## Data Science – Machine Learning – Types of the Models



**8. Types of Supervised learning models**

- ✓ Supervised learning models are divided into two types
  - Regression models
  - Classification models

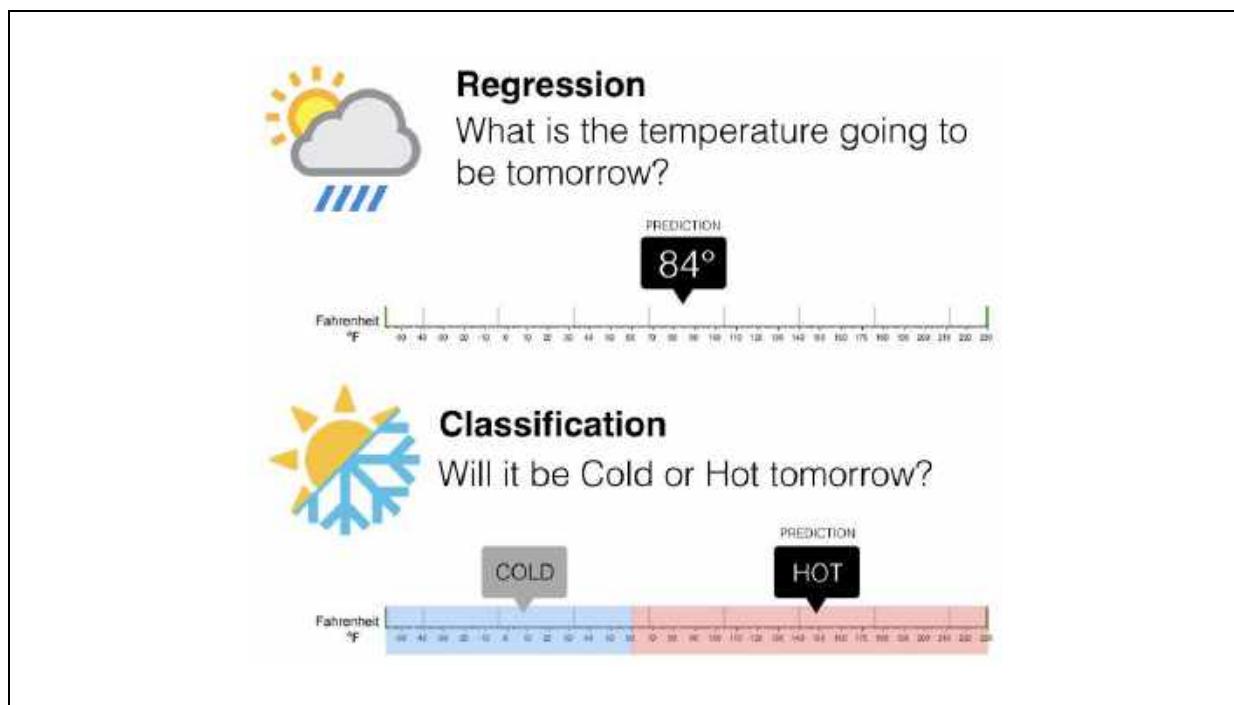
## Data Science – Machine Learning – Types of the Models

### 8.1. Regression models

- ✓ Regression models used to predict a **number**
- ✓ The output of a regression model is a **continuous**, since the prediction can be any real value.
- ✓ Examples:
  - Weight of the animal
  - Employee salary
  - Students marks
  - Stock market
  - Number of sales
  - Predicting price of house & etc

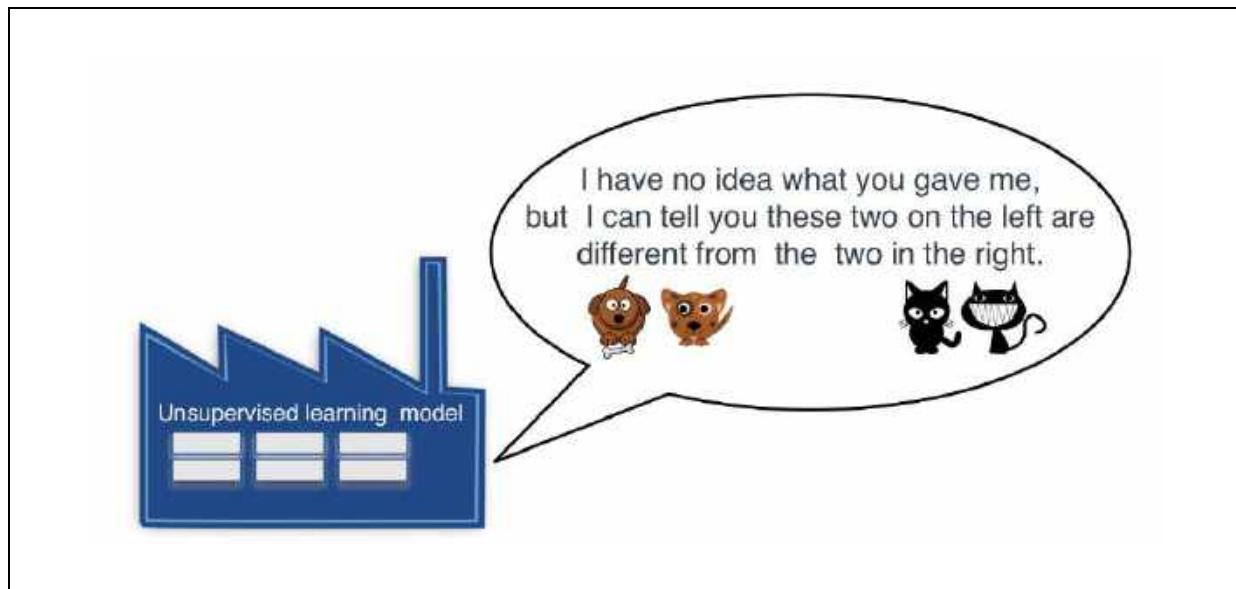
### 8.2. Classification models

- ✓ These are the types of models that predict the **class or state**.
- ✓ Examples
  - Type of animal (cat or dog),
  - Type of human being means male or female,
  - Biryani taste: good or bad or not good
  - Mail id spam or ham



## 9. Unsupervised learning example

- ✓ In unsupervised learning, we will train the models by using only input features but there are no labels
- ✓ Unsupervised learning is grouping the data based on similarities



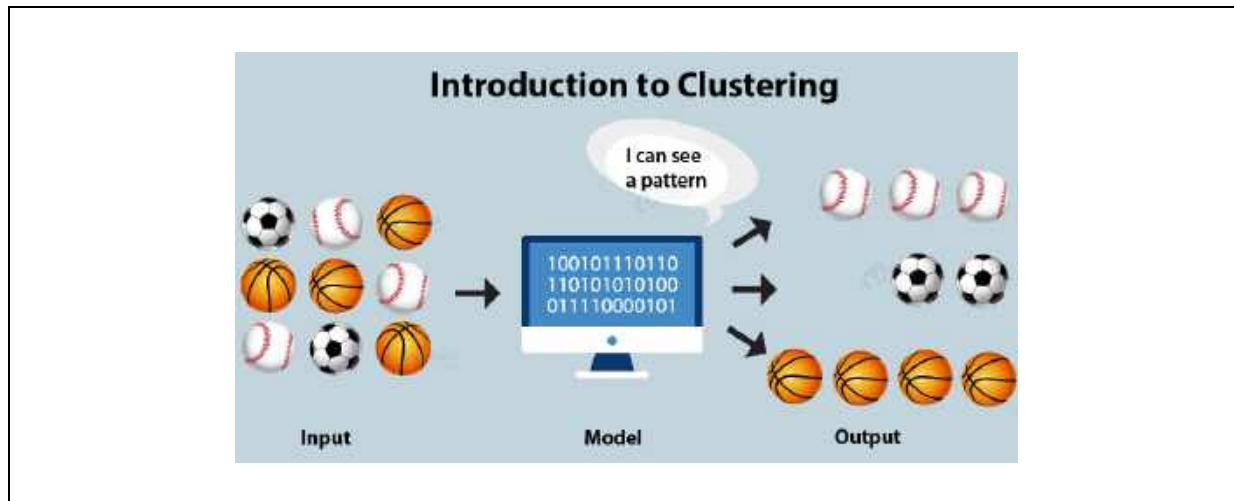
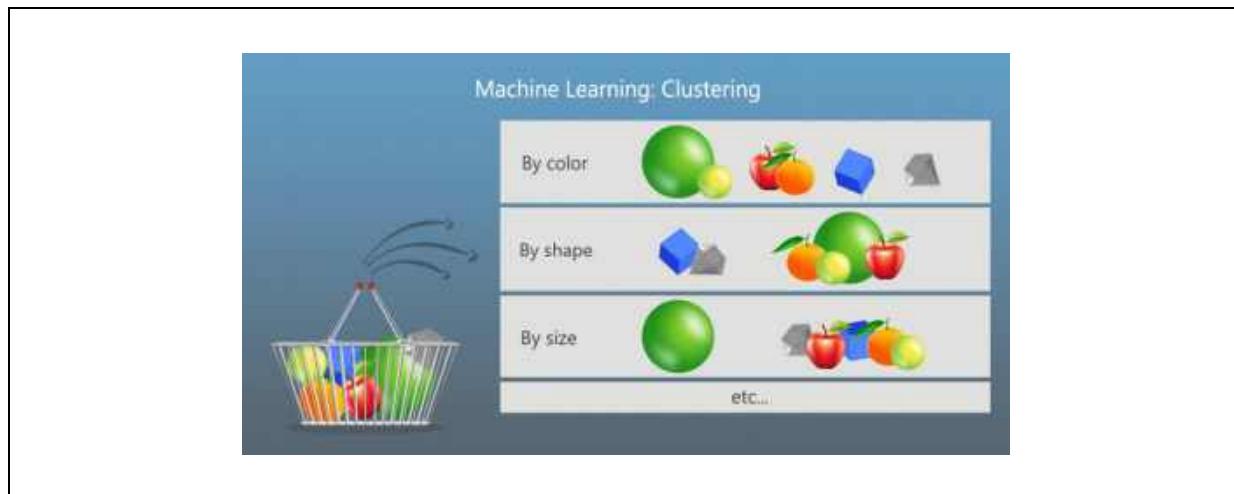
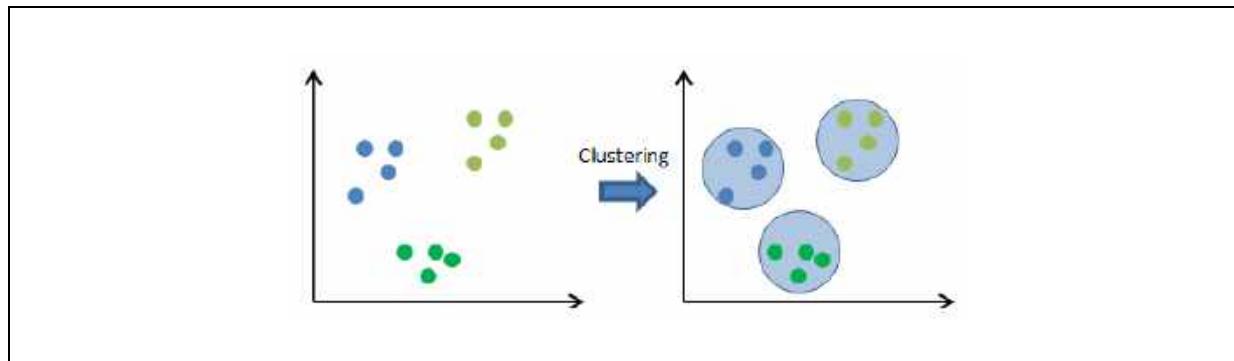
### 10. Types of unsupervised learning models

- ✓ Unsupervised learning models are divided into two types
  - Clustering
  - Dimensionality reduction

## Data Science – Machine Learning – Types of the Models

### 10.1. Clustering

- ✓ This is the task of grouping our data into clusters based on similarity.



### 10.2. Dimensionality reduction

- ✓ This is the task of simplifying our data and describing it with fewer features, without losing much generality.
- ✓ The dimensionality reduction algorithms will find ways that group them, losing as little information as possible.

**6. Data Science – Machine Learning – Life Cycle****Contents**

<b>1. Machine learning life cycle .....</b>	<b>2</b>
1.1. Data Collection.....	3
1.2. Data Preparation.....	3
1.3. Data Wrangling .....	3
1.4. Train the model.....	3
1.5. Test the model .....	4
1.6. Model deployment .....	4

**6. Data Science – Machine Learning – Life Cycle****1. Machine learning life cycle**

- ✓ There are mainly six steps involved here,
  - Data collection
  - Data preparation
  - Data Wrangling
  - Train the model
  - Test the model
  - Model deployment

## Data Science – Machine Learning – Life Cycle

---

### 1.1. Data Collection

- ✓ Data Gathering is the first step in machine learning life cycle.
- ✓ In this step, we need to identify the different data sources.
- ✓ Data can be collected from different sources such as files, database, web & etc.

### 1.2. Data Preparation

- ✓ During data preparation we should understand about the data format, quality of data & etc.
- ✓ A better understanding of data leads to an effective outcome.
- ✓ We will find correlations, general trends, and outliers.

### 1.3. Data Wrangling

- ✓ Data wrangling is the process of cleaning and converting raw data into a useable format.
- ✓ It is the process of,
  - Cleaning the data
  - Missing Values
  - Duplicate data
  - Invalid data
  - Selecting the variable to use,
- ✓ It is not necessary that data we have collected is always of our use as some of the data may not be useful.

### 1.4. Train the model

- ✓ During training, the model can learn different patterns and rules.
- ✓ We need to use train dataset to train the model.

### 1.5. Test the model

- ✓ Once model trained then testing is required to check the accuracy.
- ✓ We need to use test dataset to test the model.

### 1.6. Model deployment

- ✓ Once model trained and tested, if model is producing good results then we can deploy the model in the real time.

**7. Data Science – Machine Learning – Train & Test Datasets****Contents**

<b>1. Types of Datasets in machine learning.....</b>	<b>2</b>
<b>2. Train dataset.....</b>	<b>3</b>
<b>3. Test dataset .....</b>	<b>3</b>
<b>4. How to decide size of these 3 sets?.....</b>	<b>3</b>
<b>5. Creating array .....</b>	<b>4</b>
<b>6. train_test_split(p) function.....</b>	<b>5</b>
<b>7. train_test_split(p, random_state = 0) function.....</b>	<b>18</b>

## 7. Data Science – Machine Learning – Train & Test Datasets

### 1. Types of Datasets in machine learning

- ✓ There are mainly 3 types of datasets used in Machine learning,
  - Train dataset
  - Test dataset
  - Validation dataset

#### Note

- ✓ Here validation dataset is an optional but training and test datasets are mandatory



### 2. Train dataset

- ✓ Train dataset is used to train the models.
- ✓ This is the part of dataset which is used to train the model.
- ✓ Typically, training set contains about 60-70% of total dataset.
- ✓ First step is, model should get train with training dataset, during training model learns the parameters and underlying concepts from dataset.

### 3. Test dataset

- ✓ Test dataset is used to test the models.
- ✓ Once model training is done then we need to test the model with test dataset.
- ✓ During testing the model we will understand about model performance either good or not.
- ✓ Size of Test set is about 15-30% of total dataset.

### 4. How to decide size of these 3 sets?

- ✓ There is no thumb rule about choosing the size of these sets, but according to the experts' 70-30 or 60-40 is a good size for train and test set respectively.

## Data Science – Machine Learning – Train & Test Datasets

### 5. Creating array

- ✓ We can create an array and split that array

Program      Creating an array

Name          demo1.py

```
import numpy as np  
  
dataset = np.arange(10)  
  
print(dataset)
```

Output

```
[0 1 2 3 4 5 6 7 8 9]
```

## 6. `train_test_split(p)` function

- ✓ `train_test_split(p)` is a predefined function in `sklearn.model_selection` package
- ✓ We need to access this function from `sklearn` package.
- ✓ By using this function we can split the dataset into train dataset and test dataset.

**Program Name** Creating an array and splitting dataset  
**demo2.py**

```
import numpy as np
from sklearn.model_selection import train_test_split

dataset = np.arange(10)

result = train_test_split(dataset)

print(dataset)
print(result)
```

### Output

```
C:\Users\Nireekshan\Desktop\PROGRAMS>py demo1.py
[0 1 2 3 4 5 6 7 8 9]
[array([0, 1, 8, 9, 2, 5, 7]), array([3, 6, 4])]

C:\Users\Nireekshan\Desktop\PROGRAMS>py demo1.py
[0 1 2 3 4 5 6 7 8 9]
[array([3, 2, 6, 4, 1, 5, 9]), array([7, 8, 0])]

C:\Users\Nireekshan\Desktop\PROGRAMS>py demo1.py
[0 1 2 3 4 5 6 7 8 9]
[array([7, 8, 4, 3, 0, 6, 2]), array([5, 1, 9])]
```

## Data Science – Machine Learning – Train & Test Datasets

**Program Name** Creating an array and splitting  
demo3.py

```
import numpy as np
from sklearn.model_selection import train_test_split

dataset = np.arange(10)

X_train, X_test = train_test_split(dataset)

print(dataset)
print()
print(X_train)
print(X_test)
```

**Output**

```
[0 1 2 3 4 5 6 7 8 9]
[0 6 5 1 4 9 7]
[8 3 2]
```

## Data Science – Machine Learning – Train & Test Datasets

**Program Name** Creating an array and splitting demo4.py

```
import numpy as np
from sklearn.model_selection import train_test_split

dataset = np.arange(10)

X_train, X_test = train_test_split(dataset, test_size = 4)

print(dataset)
print()
print(X_train)
print(X_test)
```

**Output**

```
[0 1 2 3 4 5 6 7 8 9]
[6 5 7 9 3 1]
[2 0 4 8]
```

## Data Science – Machine Learning – Train & Test Datasets

**Program Name** Creating an array and splitting demo5.py

```
import numpy as np
from sklearn.model_selection import train_test_split

dataset = np.arange(10)

X_train, X_test = train_test_split(dataset, train_size = 6)

print(dataset)
print()
print(X_train)
print(X_test)
```

**Output**

```
[0 1 2 3 4 5 6 7 8 9]
[1 9 4 3 2 5]
[7 8 6 0]
```

## Data Science – Machine Learning – Train & Test Datasets

**Program Name** Creating an array and splitting demo6.py

```
import numpy as np
from sklearn.model_selection import train_test_split

dataset = np.arange(10)

X_train, X_test = train_test_split(dataset, test_size = 0.4)

print(dataset)
print()
print(X_train)
print(X_test)
```

**Output**

```
[0 1 2 3 4 5 6 7 8 9]
[9 5 2 0 8 7]
[1 4 3 6]
```

## Data Science – Machine Learning – Train & Test Datasets

**Program Name** Creating an array and splitting demo7.py

```
import numpy as np
from sklearn.model_selection import train_test_split

dataset = np.arange(10)

X_train, X_test = train_test_split(dataset, train_size = 0.6)

print(dataset)
print()
print(X_train)
print(X_test)
```

**Output**

```
[0 1 2 3 4 5 6 7 8 9]
[9 6 4 8 3 5]
[2 1 0 7]
```

## Data Science – Machine Learning – Train & Test Datasets

**Program Name** Creating an array and splitting demo8.py

```
import numpy as np
from sklearn.model_selection import train_test_split

dataset = np.arange(10)

X_train, X_test = train_test_split(dataset, train_size = 4, test_size = 4)

print(dataset)
print()
print(X_train)
print(X_test)
```

**Output**

```
[0 1 2 3 4 5 6 7 8 9]
[0 9 3 6]
[4 7 2 5]
```

## Data Science – Machine Learning – Train & Test Datasets

**Program Name** Creating an array and splitting demo9.py

```
import numpy as np
from sklearn.model_selection import train_test_split

dataset = np.arange(10)

X_train, X_test = train_test_split(dataset, train_size = 4, test_size = 10)

print(dataset)
print()
print(X_train)
print(X_test)
```

**Output**

```
ValueError: test_size=10 should be either positive and smaller than the number of samples 10 or a float in the (0, 1) range
```

## Data Science – Machine Learning – Train & Test Datasets

**Program Name** Creating an array and splitting  
demo10.py

```
import numpy as np

dataset = np.arange(20)

print(dataset)
```

**Output**

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
```

**Program Name** Creating an array and splitting  
demo11.py

```
import numpy as np

dataset = np.arange(20).reshape(2, 10)

print(dataset)
```

**Output**

```
[[ 0  1  2  3  4  5  6  7  8  9]
 [10 11 12 13 14 15 16 17 18 19]]
```

## Data Science – Machine Learning – Train & Test Datasets

**Program Name** Creating an array and splitting  
demo12.py

```
import numpy as np

dataset = np.arange(20).reshape(2, 10).T

print(dataset)
```

**Output**

```
[[ 0 10]
 [ 1 11]
 [ 2 12]
 [ 3 13]
 [ 4 14]
 [ 5 15]
 [ 6 16]
 [ 7 17]
 [ 8 18]
 [ 9 19]]
```

## Data Science – Machine Learning – Train & Test Datasets

**Program Name** Creating an array and splitting  
demo13.py

```
import numpy as np

X = np.arange(20).reshape(2, 10).T

y = np.arange(10)

print(X)
print()
print(y)
```

**Output**

```
[[ 0 10]
 [ 1 11]
 [ 2 12]
 [ 3 13]
 [ 4 14]
 [ 5 15]
 [ 6 16]
 [ 7 17]
 [ 8 18]
 [ 9 19]]

[0 1 2 3 4 5 6 7 8 9]
```

## Data Science – Machine Learning – Train & Test Datasets

**Program Name** Creating an array and splitting  
demo14.py

```
import numpy as np

X = np.arange(20).reshape(2, 10).T

y = np.arange(10)

print(X)
print()
print(y)
```

**Output**

```
[[ 0 10]
 [ 1 11]
 [ 2 12]
 [ 3 13]
 [ 4 14]
 [ 5 15]
 [ 6 16]
 [ 7 17]
 [ 8 18]
 [ 9 19]]

[0 1 2 3 4 5 6 7 8 9]
```

## Data Science – Machine Learning – Train & Test Datasets

**Program Name** Creating an array and splitting demo15.py

```
import numpy as np
from sklearn.model_selection import train_test_split

X = np.arange(20).reshape(2, 10).T

y = np.arange(10)

X_train, X_test, y_train, y_test = train_test_split(X, y)

print(X_train)
print()
print(X_test)
print()
print(y_train)
print()
print(y_test)
```

### Output

```
[[ 9 19]
 [ 6 16]
 [ 2 12]
 [ 4 14]
 [ 3 13]
 [ 8 18]
 [ 7 17]]

[[ 5 15]
 [ 1 11]
 [ 0 10]]

[9 6 2 4 3 8 7]

[5 1 0]
```

**7. `train_test_split(p, random_state = 0)` function**

- ✓ `train_test_split(p, random_state = 0)` is a predefined function in `sklearn.model_selection` package
- ✓ We need to access this function from `sklearn` package.
- ✓ By using this function we can split the dataset into train dataset and test dataset.
- ✓ We will get the same train and test datasets across different executions.

## Data Science – Machine Learning – Train & Test Datasets

**Program Name** Creating an array and splitting  
demo16.py

```
import numpy as np
from sklearn.model_selection import train_test_split

X = np.arange(20).reshape(2, 10).T

y = np.arange(10)

X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state = 0)

print(X_train)
print()
print(X_test)
print()
print(y_train)
print()
print(y_test)
```

**Output**

```
[[ 9 19]
 [ 1 11]
 [ 6 16]
 [ 7 17]
 [ 3 13]
 [ 0 10]
 [ 5 15]]

[[ 2 12]
 [ 8 18]
 [ 4 14]]

[9 1 6 7 3 0 5]

[2 8 4]
```

**8. Data Science – Machine Learning – R value****Contents**

<b>1. Regression .....</b>	<b>2</b>
<b>2. A line .....</b>	<b>2</b>
<b>3. The goal .....</b>	<b>3</b>
<b>4. Can we use regression in everywhere? .....</b>	<b>4</b>
<b>5. r value .....</b>	<b>4</b>
<b>6. r value range.....</b>	<b>4</b>
<b>7. Calculate r value .....</b>	<b>5</b>

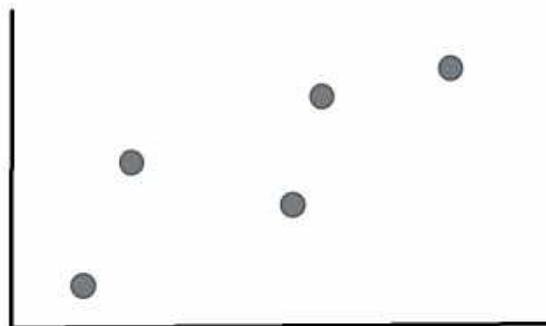
## 8. Data Science – Machine Learning – R value

### 1. Regression

- ✓ By using regression we can find the relationship between variables.
- ✓ Once we understand the relationship in between the variables then we can predict the future outcomes

### 2. A line

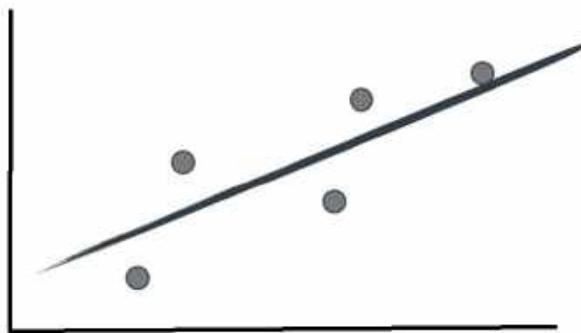
- ✓ If two variables having relationship then if we draw this relationship in a two dimensional then we get a straight line.
- ✓ The picture of linear regression is simple.
- ✓ Let us say we have some points, a line will travel in between these points



## Data Science – Machine Learning – R value

### 3. The goal

- ✓ The goal of linear regression is to draw the best fitted line.
- ✓ Best fitted line means that the line which passes as close as possible to these points.



### 4. Can we use regression in everywhere?

- ✓ If there is a relationship in between the variables then only we can use linear regression algorithm.
- ✓ If there is no relationship in between the variables then we cannot use linear regression algorithm.

### 5. r value

- ✓ r value explains about how variables are related each other.
- ✓ This is very important step to recognise relationship in between values of x-axis and y-axis.

### 6. r value range

- ✓ The r values range, from -1 to 1.
- ✓ While calculating if we get r value as near to **-1** or **1** then those variables are **strongly related** each other.
- ✓ While calculating if we get r value as **0** then it confirms that there is **no relationship** in between the variables.

## Data Science – Machine Learning – R value

### 7. Calculate r value

- ✓ By using python scipy module we can calculate r value easily

**Program Name** Plotting x and y values  
demo1.py

```
import pandas as pd
from scipy import stats
import matplotlib.pyplot as plt

d = {
    "area": [1, 2, 3, 4],
    "rice_yield": [10, 20, 30, 40]
}

df = pd.DataFrame(d)

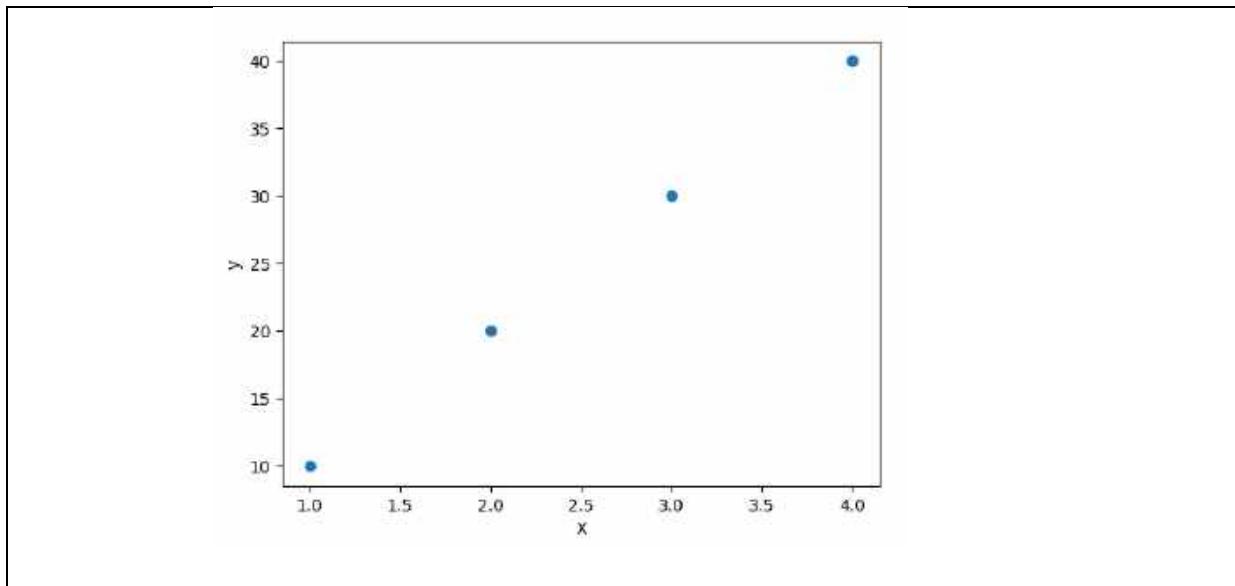
X = df.area.values
y = df.rice_yield.values

plt.xlabel("X")
plt.ylabel("y")

plt.scatter(X, y)
plt.show()
```

**Output**

## Data Science – Machine Learning – R value



## Data Science – Machine Learning – R value

**Program Name** Calculating r value  
demo2.py

```
import pandas as pd

d = {
    "area": [1, 2, 3, 4],
    "rice_yield": [10, 20, 30, 40]
}

df = pd.DataFrame(d)

r_value = df.corr()["rice_yield"]

print(r_value)
```

**Output**

1.0

### Note:

- ✓ The result 1.0
- ✓ So we can confirm that there is strong a relationship.
- ✓ So we can apply linear regression algorithm on top of these variables for future prediction

## Data Science – Machine Learning – R value

**Program Name** Calculating r value  
demo3.py

```
import pandas as pd

d = {
    "a": [600, 3000, 2, 3600, 4],
    "b": [550000, 565000, 610000, 680000, 725000]
}

df = pd.DataFrame(d)

r_value = df.corr()["b"]

print(r_value)
```

### Output

-0.06533879637370224

### Note:

- ✓ The result -0.06533879637370224 this value is very near to 0
- ✓ So we can confirm that there is no relationship.
- ✓ So we cannot apply linear regression algorithm on top of these variables, even if we apply then we will get wrong prediction results

## 9. Data Science – Machine Learning – Simple Linear Regression

### Contents

<b>1. Regression .....</b>	2
<b>2. A line .....</b>	2
<b>3. The goal .....</b>	3
<b>4. Linear Regression .....</b>	4
<b>5. Types of linear regression .....</b>	4
<b>6. Simple linear regression.....</b>	4
<b>7. Multiple linear regression.....</b>	4
<b>8. Simple linear regression example.....</b>	5
8.1. Problem statement .....	5
8.2. The solution .....	5
<b>9. Machine learning Terminology .....</b>	6
9.1. Features .....	6
9.2. Label or target.....	6
9.3. Models .....	6
9.4. Prediction.....	6
9.5. Formula .....	6
<b>10. Intercept and coefficient.....</b>	17
<b>11. <math>Y = m * X + b</math> (m is coefficient and b is intercept) .....</b>	18
<b>12. Best fitted line .....</b>	19
<b>13. Predicting a group of home prices.....</b>	21

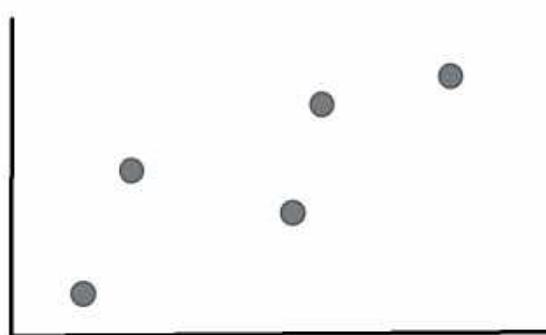
## 9. Data Science – Machine Learning – Simple Linear Regression

### 1. Regression

- ✓ Regression analysis is used to explain the relationship between two variables.
- ✓ Also called as it's a relationship in between dependent variable and one or more independent variables.

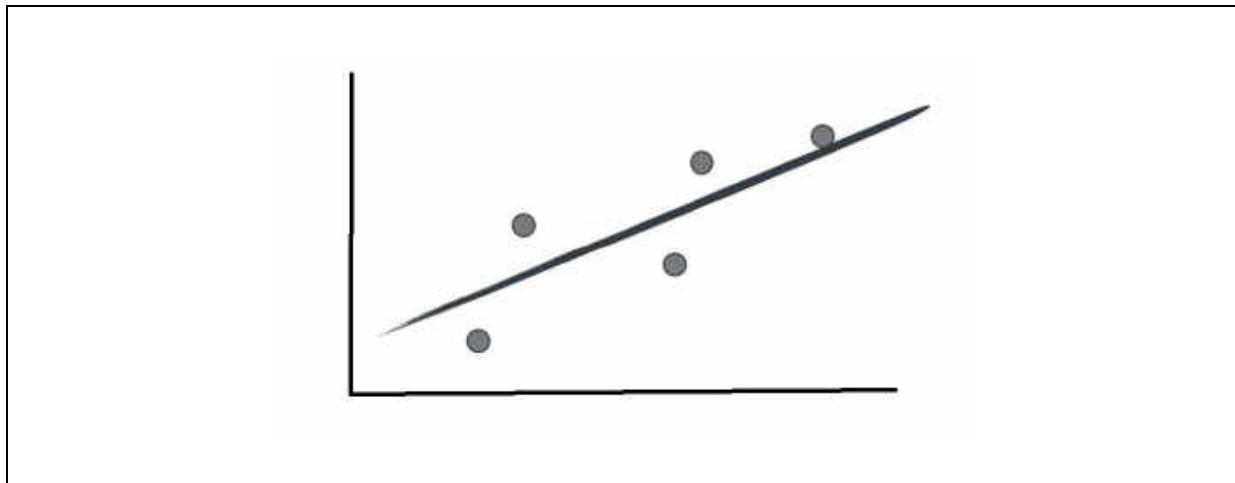
### 2. A line

- ✓ If two variables having relationship then if we draw this relationship in a two dimensional then we get a straight line.
- ✓ The picture of linear regression is simple.
- ✓ Let us say we have some points, a line will travel in between these points



### 3. The goal

- ✓ The goal of linear regression is to draw the best fitted line.
- ✓ Best fitted line means that the line which passes as close as possible to these points.



#### 4. Linear Regression

- ✓ This is a technique and it explains the relationship between the dependent variable and independent variables

#### 5. Types of linear regression

- ✓ There are two types of linear regression
  - Simple linear regression
  - Multiple linear regression

#### 6. Simple linear regression

- ✓ When you have only 1 independent variable and 1 dependent variable, it is called simple linear regression.

#### 7. Multiple linear regression

- ✓ When you have 2 or more independent variable and 1 dependent variable, it is called multiple linear regression.

## 8. Simple linear regression example

- ✓ When you have only 1 independent variable and 1 dependent variable, it is called simple linear regression.

### 8.1. Problem statement

- ✓ Assuming that we are planning to buy a new house and need to predict the price of a house

### 8.2. The solution

- ✓ While buying house first we need to check the area of the house

area	price
2600	550000
3000	565000
3200	610000
3600	680000
4000	725000

## 9. Machine learning Terminology

### 9.1. Features

- ✓ From the given problem the feature is **area** and **price**

### 9.2. Label or target

- ✓ Price of the house

### 9.3. Models

- ✓ A machine learning model is simply a rule, or a formula, which predicts a label from the features.
- ✓ In this case, the model is the equation we found for the price.

### 9.4. Prediction

- ✓ The prediction is simply the output of the model.
- ✓ If the model gives the result as “Hey Guru I think the house with 36000 area is going to cost \$300”, then the prediction is 300.

### 9.5. Formula

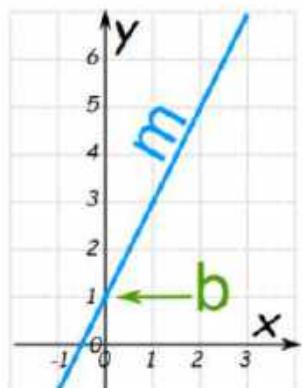
- ✓ Home price =  $m * (\text{area}) + b$

### Reminder

- ✓ Once please walk through our maths regression chapter ([Chapter 7. Statistics - PART - 7 - Regression](#)) which we have already discussed, thanks

Data Science – Machine Learning – Simple Linear Regression

---



**price =  $m * \text{area} + b$**

$$y = mx + b$$

Slope (or Gradient)      Y Intercept

## Data Science – Machine Learning – Simple Linear Regression

**Program Name** Loading house prices dataset  
demo1.py

```
import pandas as pd

df = pd.read_csv("homeprices.csv")

print(df.head())
```

**Output**

	area	price
0	2600	550000
1	3000	565000
2	3200	610000
3	3600	680000
4	4000	725000

## Data Science – Machine Learning – Simple Linear Regression

**Program Name** Creating scatter plot using matplotlib  
demo2.py

```
import pandas as pd
import matplotlib.pyplot as plt

df=pd.read_csv("homeprices.csv")

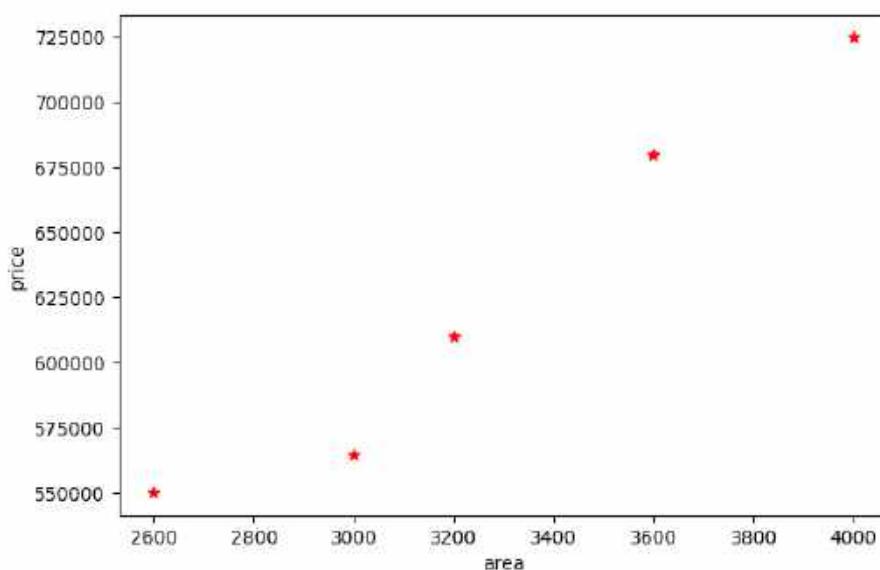
# plotting the dataset

plt.xlabel('area')
plt.ylabel('price')

plt.scatter(df.area, df.price, color = 'red', marker = '*')

plt.show()
```

### Output



## Data Science – Machine Learning – Simple Linear Regression

**Program Name** Loading the data set  
demo3a.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

df = pd.read_csv("homeprices.csv")
new_df = df.drop('price', axis = 'columns')

print(df)
print()
print(new_df)
```

### Output

```
    area    price
0  2600  550000
1  3000  565000
2  3200  610000
3  3600  680000
4  4000  725000

    area
0  2600
1  3000
2  3200
3  3600
4  4000
```

## Data Science – Machine Learning – Simple Linear Regression

**Program Name** Loading the data set  
demo3b.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

df = pd.read_csv("homeprices.csv")
new_df = df.drop('price', axis = 'columns')

print(new_df.values)
print()
print(df.price.values)
```

**Output**

```
[ [ 2600]
  [ 3000]
  [ 3200]
  [ 3600]
  [ 4000]]

[ 550000  565000  610000  680000  725000]
```

## Data Science – Machine Learning – Simple Linear Regression

**Program Name** Creating LinearRegression object  
demo3.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

df = pd.read_csv("homeprices.csv")
new_df = df.drop('price', axis = 'columns')

# Training the Algorithm
reg = LinearRegression()
reg.fit(new_df.values, df.price.values)

print("Training the Algorithm")
```

**Output**

Training the Algorithm

## Data Science – Machine Learning – Simple Linear Regression

**Program Name** Predict price of a home with area = 3300 sqr ft  
demo4.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

df=pd.read_csv("homeprices.csv")
new_df = df.drop('price', axis='columns')

reg = LinearRegression()
reg.fit(new_df.values, df.price.values)

# Making Predictions
print(reg.predict([[3300]]))
```

**Output**

```
[628715.75342466]
```

## Data Science – Machine Learning – Simple Linear Regression

**Program Name** Predict price of a home with area = 5000 sqr ft  
demo5.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

df=pd.read_csv("homeprices.csv")
new_df = df.drop('price', axis='columns')

reg = LinearRegression()
reg.fit(new_df.values, df.price.values)

# Making Predictions
print(reg.predict([[5000]]))
```

**Output**

```
[859554.79452055]
```

## Data Science – Machine Learning – Simple Linear Regression

**Program Name** Capture the coefficient from regression  
demo6.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

df=pd.read_csv("homeprices.csv")
new_df = df.drop('price', axis='columns')

reg = LinearRegression()
reg.fit(new_df.values, df.price.values)

print(reg.coef_)
```

**Output**

```
[135.78767123]
```

## Data Science – Machine Learning – Simple Linear Regression

**Program Name** Capture the intercept from regression  
demo7.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

df=pd.read_csv("homeprices.csv")
new_df = df.drop('price', axis='columns')

reg = LinearRegression()
reg.fit(new_df.values, df.price.values)

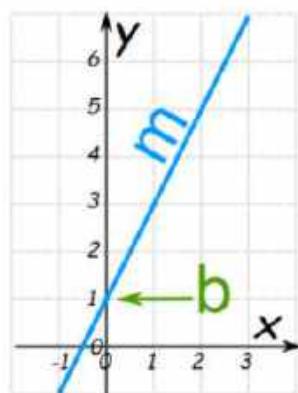
print(reg.intercept_)
```

**Output**

180616.43835616432

## 10. Intercept and coefficient

- ✓ Intercept = 180616.43835616432
- ✓ Coefficient = 135.78767123



$$\text{price} = m * \text{area} + b$$

$$y = mx + b$$

Slope (or Gradient)      Y Intercept

**11.  $Y = m * X + b$  (m is coefficient and b is intercept)**

- ✓ Let's calculate the above formula.
- ✓ In the given formula m is coefficient and b is intercept
- ✓  $Y = m * X + b$
- ✓  $Y = 135.78767123 * 3300 + 180616.43835616432$
- ✓  $Y = 628715.75342466$
- ✓ Awesome....!!!!

## 12. Best fitted line

- ✓ Let's calculate the above formula.
- ✓ We can draw a line

**Program  
Name**

Drawing a best fitted line  
demo8.py

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

df=pd.read_csv("homeprices.csv")

new_df = df.drop('price', axis='columns')

reg = LinearRegression()
reg.fit(new_df.values, df.price.values)

plt.xlabel('area')
plt.ylabel('price')

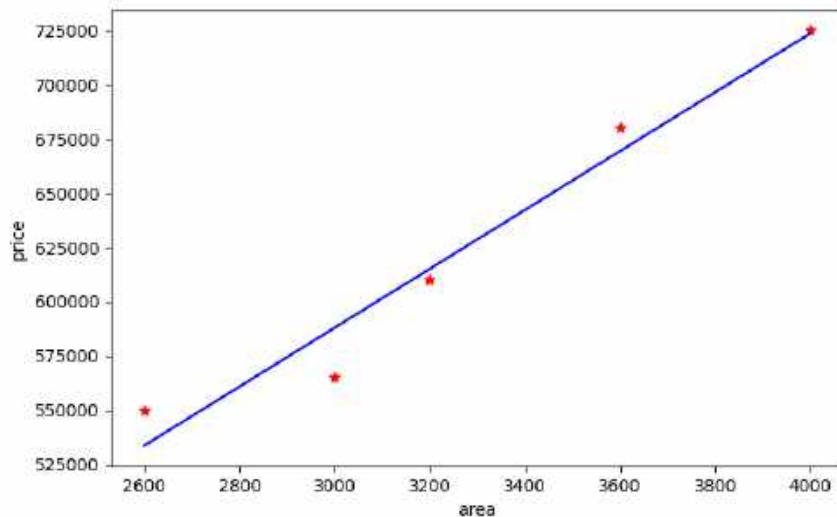
plt.scatter(df.area.values, df.price.values, color = 'red', marker = '*')

plt.plot(df.area.values, reg.predict(df[['area']].values), color = 'blue')

plt.show()
```

## Data Science – Machine Learning – Simple Linear Regression

Output



### 13. Predicting a group of home prices

- ✓ By using above model we can predict the group of home prices as well

**Program Name** Loading a group of house areas  
demo9.py

```
import pandas as pd
from sklearn import linear_model
from sklearn.linear_model import LinearRegression

df = pd.read_csv("homeprices.csv")
new_df = df.drop('price', axis='columns')

reg = LinearRegression()
reg.fit(new_df.values, df.price.values)

area_df = pd.read_csv("areas.csv")
print(area_df)
```

#### Output

	area
0	1000
1	1500
2	2300
3	3540
4	4120
5	4560
6	5490
7	3460
8	4750
9	2300
10	9000
11	8600
12	7100

## Data Science – Machine Learning – Simple Linear Regression

**Program Name** Predicting a group of home prices  
demo10.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

df = pd.read_csv("homeprices.csv")
new_df = df.drop('price', axis='columns')

reg = LinearRegression()
reg.fit(new_df.values, df.price.values)

area_df = pd.read_csv("areas.csv")

prices = reg.predict(area_df.values)
print(prices)
```

### Output

```
[ 316404.10958904  384297.94520548  492928.08219178  661304.79452055
 740061.64383562  799808.21917808  926090.75342466  650441.78082192
 825607.87671233  492928.08219178  1402705.47945205  1348390.4109589
 1144708.90410959]
```

## Data Science – Machine Learning – Simple Linear Regression

**Program Name** Create a csv file with predictions  
demo11.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

df = pd.read_csv("homeprices.csv")
new_df = df.drop('price', axis='columns')

reg = LinearRegression()
reg.fit(new_df.values, df.price.values)

area_df = pd.read_csv("areas.csv")
p = reg.predict(area_df.values)

area_df['prices'] = p
area_df.to_csv('output.csv')
print("Please check in current directory for output.csv")
```

### Output

Please check in current directory for output.csv

## Data Science – Machine Learning –Linear Regression Example

---

### 9.1. Data Science – Machine Learning – Linear Regression Example

#### Contents

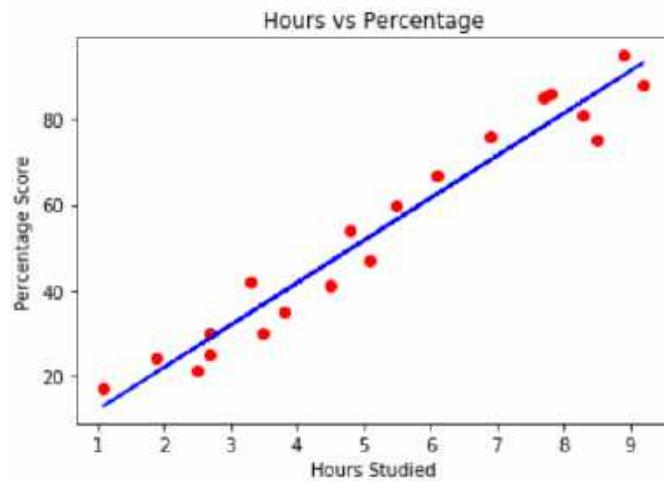
1. Scenario .....	2
2. Maths behind .....	3
3. Loading dataset .....	4
4. Plotting the data.....	5
5. Intercept & coefficient .....	10
6. Important info .....	12
7. Making Predictions.....	12
8. Evaluating the Algorithm .....	15
9. Evaluation metrics.....	15

## Data Science – Machine Learning –Linear Regression Example

### 9.1. Data Science – Machine Learning – Linear Regression Example

#### 1. Scenario

- ✓ Let's find the relationship in between **marks** and **number of study hours**
- ✓ We want to find out the **marks** for given **number of study hours** to a student
- ✓ If we plot the independent variable (hours) on the x-axis and dependent variable (percentage) on the y-axis then linear regression gives us a straight line that best fits the data points.



## 2. Maths behind

- ✓ We know that the equation of a straight line is basically
  - $y = mx + b$
- ✓ Where **b** is the **intercept** and **m** is the **slope** of the line.
- ✓ So basically, the linear regression algorithm gives us the most optimal value for the **intercept** and the **slope**.
- ✓ The y and x variables remain the same
- ✓ There can be multiple straight lines depending upon the values of intercept and slope.
- ✓ Basically what the linear regression algorithm does is it fits multiple lines on the data points and returns the line that results in the least error.

## Data Science – Machine Learning –Linear Regression Example

### 3. Loading dataset

- ✓ Once we have dataset then we need to load by using pandas
- ✓ If we load dataset then it returns the DataFrame

Program Name Loading student\_scores dataset  
demo1.py

```
import pandas as pd

df = pd.read_csv('student_scores.csv')

print(df.head())
```

Output

	Hours	Scores
0	2.5	21
1	5.1	47
2	3.2	27
3	8.5	75
4	3.5	30

## Data Science – Machine Learning –Linear Regression Example

### 4. Plotting the data

- ✓ Let's plot our data points to see the relationship between the data.

Program Name      Plotting the dataset  
demo2.py

```
import pandas as pd
import matplotlib.pyplot as plt

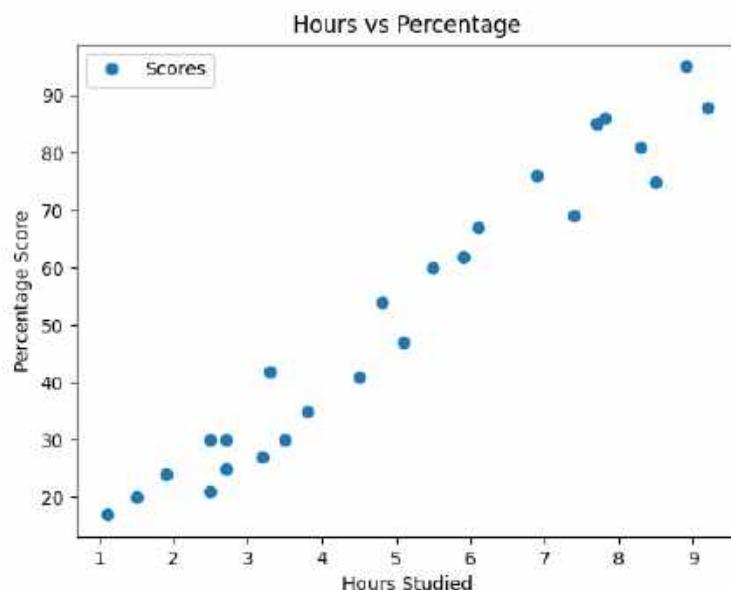
df = pd.read_csv('student_scores.csv')

df.plot(x='Hours', y='Scores', style='o')

plt.title('Hours vs Percentage')
plt.xlabel('Hours Studied')
plt.ylabel('Percentage Score')

plt.show()
```

### Output



## Data Science – Machine Learning –Linear Regression Example

**Program Name** Preparing the data  
demo3.py

```
import pandas as pd

df = pd.read_csv('student_scores.csv')

X = df.iloc[:, :-1].values
y = df.iloc[:, 1].values

print(X)
print(y)
```

**Output**

```
[[2.5]
 [5.1]
 [3.2]
 [8.5]
 [3.5]
 [1.5]
 [9.2]
 [5.5]
 [8.3]
 [2.7]
 [7.7]
 [5.9]
 [4.5]
 [3.3]
 [1.1]
 [8.9]
 [2.5]
 [1.9]
 [6.1]
 [7.4]
 [2.7]
 [4.8]
 [3.8]
 [6.9]
 [7.8]]
[21 47 27 75 30 20 88 60 81 25 85 62 41 42 17 95 30 24 67 69 30 54 35 76
86]
```

## Data Science – Machine Learning –Linear Regression Example

**Program Name** Splitting the data  
demo4.py

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

df = pd.read_csv('student_scores.csv')

X = df.iloc[:, :-1].values
y = df.iloc[:, 1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)

print("X_train")
print(X_train)

print()
print("X_test")
print(X_test)

print()
print("Y_train")
print(y_train)

print()
print("y_test")
print(y_test)
```

## Data Science – Machine Learning –Linear Regression Example

### Output

```
X_train
[[3.8]
 [1.9]
 [7.8]
 [6.9]
 [1.1]
 [5.1]
 [7.7]
 [3.3]
 [8.3]
 [9.2]
 [6.1]
 [3.5]
 [2.7]
 [5.5]
 [2.7]
 [8.5]
 [2.5]
 [4.8]
 [8.9]
 [4.5]]

X_test
[[1.5]
 [3.2]
 [7.4]
 [2.5]
 [5.9]]

Y_train
[35 24 86 76 17 47 85 42 81 88 67 30 25 60 30 75 21 54 95 41]

y_test
[20 27 69 30 62]
```

## Data Science – Machine Learning –Linear Regression Example

**Program Name** Training the model  
demo5.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

df = pd.read_csv('student_scores.csv')

X = df.iloc[:, :-1].values
y = df.iloc[:, 1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=0)

regressor = LinearRegression()
regressor.fit(X_train, y_train)

print("Model got trained with data")
```

**Output**

Model got trained with data

## Data Science – Machine Learning –Linear Regression Example

### 5. Intercept & coefficient

- ✓ In the theory section we said that linear regression model basically finds the best value for the intercept and slope, which results in a line that best fits the data.
- ✓ We can get the values of the intercept and slop from linear regression

Program

Name demo6.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

df = pd.read_csv('student_scores.csv')

X = df.iloc[:, :-1].values
y = df.iloc[:, 1].values

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)

regressor = LinearRegression()
regressor.fit(X_train, y_train)

print(regressor.intercept_)
```

Output

2.018160041434669

## Data Science – Machine Learning –Linear Regression Example

**Program Name**

Getting coefficient from created model

demo7.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

df = pd.read_csv('student_scores.csv')

X = df.iloc[:, :-1].values
y = df.iloc[:, 1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)

regressor = LinearRegression()
regressor.fit(X_train, y_train)

print(regressor.coef_)
```

**Output**

[9.91065648]

## Data Science – Machine Learning –Linear Regression Example

---

### 6. Important info

- ✓ This means that for every one unit of change in hours studied, the change in the score is about 9.91%.
- ✓ In simpler words, if a student studies **one hour more** than they previously studied for an exam, they can expect to achieve an increase of **9.91%** in the **score** achieved by the student previously.

### 7. Making Predictions

- ✓ Now successfully we have trained our algorithm.
- ✓ So, it's time to make some predictions.
- ✓ We need to use our test data and see how accurately our algorithm predicts the percentage score.

## Data Science – Machine Learning –Linear Regression Example

---

**Program Name** Making predictions  
demo8.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

df = pd.read_csv('student_scores.csv')

X = df.iloc[:, :-1].values
y = df.iloc[:, 1].values

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)

regressor = LinearRegression()
regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)
print(y_pred)
```

### Output

```
[16.88414476 33.73226078 75.357018  26.79480124
 60.49103328]
```

### Great

- ✓ The y\_pred is a numpy array that contains all the predicted values for the input values in the X\_test series.

### Comparison

- ✓ To compare the actual output values for X\_test with the predicted values.

## Data Science – Machine Learning –Linear Regression Example

**Program Name** Comparing the predicted result  
demo9.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

df = pd.read_csv('student_scores.csv')

X = df.iloc[:, :-1].values
y = df.iloc[:, 1].values

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)

regressor = LinearRegression()
regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)

d = {'Actual': y_test, 'Predicted': y_pred}

compare_df = pd.DataFrame(d)
print(compare_df)
```

### Output

	Actual	Predicted
0	20	16.884145
1	27	33.732261
2	69	75.357018
3	30	26.794801
4	62	60.491033

- ✓ Though our model is not very precise, the predicted percentages are close to the actual ones.

### 8. Evaluating the Algorithm

- ✓ We need to evaluate the performance of algorithm.
- ✓ This step is particularly important to compare how well different algorithms perform on a particular dataset.
- ✓ For regression algorithms there are three evaluation metrics are commonly used

### 9. Evaluation metrics

- ✓ Mean Absolute Error (**MAE**)
- ✓ Mean Squared Error (**MSE**)
- ✓ Root Mean Squared Error (**RMSE**)

## Data Science – Machine Learning –Linear Regression Example

### Mean Absolute Error (MAE)

- ✓ Mean Absolute Error (**MAE**) is the mean of the absolute value of the errors.

### Mean Squared Error (MSE)

- ✓ Mean Squared Error (**MSE**) is the mean of the squared errors.

### Root Mean Squared Error (RMSE)

- ✓ Root Mean Squared Error (**RMSE**) is the square root of the mean of the squared errors

$$MAE = \frac{1}{n} \sum \left| y - \hat{y} \right|$$

Divide by the total number of data points  
 Predicted output value  
 Actual output value  
 Sum of  
 The absolute value of the residual

$$MSE = \frac{1}{n} \sum \underbrace{\left( y - \hat{y} \right)^2}_{\text{The square of the difference between actual and predicted}}$$

## Data Science – Machine Learning –Linear Regression Example

### We no need to calculate manually

- ✓ We don't have to perform these calculations manually.
- ✓ The scikit-Learn library comes with pre-built functions that can be used to find out these values for us.

<b>Program Name</b>	Loading student_scores dataset demo10.py
<pre>import pandas as pd import numpy as np from sklearn.model_selection import train_test_split from sklearn.linear_model import LinearRegression from sklearn import metrics  df = pd.read_csv('student_scores.csv')  X = df.iloc[:, :-1].values y = df.iloc[:, 1].values  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)  regressor = LinearRegression() regressor.fit(X_train, y_train)  y_pred = regressor.predict(X_test)  mae = metrics.mean_absolute_error(y_test, y_pred) mse = metrics.mean_squared_error(y_test, y_pred) rmse = np.sqrt(metrics.mean_squared_error(y_test, y_pred))  print('Mean Absolute Error:', mae) print('Mean Squared Error:', mse) print('Root Mean Squared Error:', rmse)</pre>	

## Data Science – Machine Learning –Linear Regression Example

### Output

Mean Absolute Error: 4.18385989900298

Mean Squared Error: 21.598769307217413

Root Mean Squared Error: 4.647447612100368

## Data Science – Machine Learning –Linear Regression Example

### 9.2. Data Science – Machine Learning – Linear Regression Example

Program Name Loading salary dataset  
demo1.py

```
import pandas as pd

dataset = pd.read_csv('Salary_Data.csv')

print(dataset.head())
```

Output

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

## Data Science – Machine Learning –Linear Regression Example

**Program Name** Preparing the data  
demo2.py

```
import pandas as pd

dataset = pd.read_csv('Salary_Data.csv')

X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values

print(X)
print(y)
```

### Output

```
[[ 1.1]
 [ 1.3]
 [ 1.5]
 [ 2. ]
 [ 2.2]
 [ 2.9]
 [ 3. ]
 [ 3.2]
 [ 3.2]
 [ 3.7]
 [ 3.9]
 [ 4. ]
 [ 4. ]
 [ 4.1]
 [ 4.5]
 [ 4.9]
 [ 5.1]
 [ 5.3]
 [ 5.9]
 [ 6. ]
 [ 6.8]
 [ 7.1]
 [ 7.9]
 [ 8.2]
 [ 8.7]
 [ 9. ]
 [ 9.5]
 [ 9.6]
 [10.3]
 [10.5]]
[ 39343.  46205.  37731.  43525.  39891.  56642.  60150.  54445.  64445.
 57189.  63218.  55794.  56957.  57081.  61111.  67938.  66029.  83088.
 81363.  93940.  91738.  98273.  101302.  113812.  109431.  105582.  116969.
 112635. 122391. 121872.]
```

## Data Science – Machine Learning –Linear Regression Example

**Program Name** Splitting the dataset  
demo3.py

```
import pandas as pd
from sklearn.model_selection import train_test_split

dataset = pd.read_csv('Salary_Data.csv')

X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, random_state = 0)

print("X_train")
print(X_train)
print()

print("X_test")
print(X_test)
print()

print("y_train")
print(y_train)
print()

print("y_test")
print(y_test)
```

## Data Science – Machine Learning –Linear Regression Example

### Output

```
X_train
[[ 2.9]
 [ 5.1]
 [ 3.2]
 [ 4.5]
 [ 8.2]
 [ 6.8]
 [ 1.3]
 [10.5]
 [ 3. ]
 [ 2.2]
 [ 5.9]
 [ 6.]
 [ 3.7]
 [ 3.2]
 [ 9.]
 [ 2.]
 [ 1.1]
 [ 7.1]
 [ 4.9]
 [ 4. ]]

X_test
[[ 1.5]
 [10.3]
 [ 4.1]
 [ 3.9]
 [ 9.5]
 [ 8.7]
 [ 9.6]
 [ 4.]
 [ 5.3]
 [ 7.9]]]

y_train
[ 56642.  66029.  64445.  61111.  113812.  91738.  46205.  121872.  60150.
 39891.  81363.  93940.  57189.  54445.  105582.  43525.  39343.  98273.
 67938.  56957.]]

y_test
[ 37731.  122391.  57081.  63218.  116969.  109431.  112635.  55794.  83088.
 101302.]
```

## Data Science – Machine Learning –Linear Regression Example

**Program Name** Training the model  
demo4.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

dataset = pd.read_csv('Salary_Data.csv')

X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
1/3, random_state = 0)

print("Training the model")

regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

**Output**

Training the model

## Data Science – Machine Learning –Linear Regression Example

**Program Name** Predicting the salaries  
demo5.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

dataset = pd.read_csv('Salary_Data.csv')

X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
1/3, random_state = 0)

regressor = LinearRegression()
regressor.fit(X_train, y_train)

print("Predicting the salaries")

y_pred = regressor.predict(X_test)

print()
print(y_pred)
```

**Output**

```
Predicting the salaries
[ 40835.10590871 123079.39940819 65134.55626083 63265.36777221
 115602.64545369 108125.8914992 116537.23969801 64199.96201652
 76349.68719258 100649.1375447 ]
```

## Data Science – Machine Learning –Linear Regression Example

Program  
Name

Plotting training dataset  
demo6.py

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

dataset = pd.read_csv('Salary_Data.csv')

X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
1/3, random_state = 0)

regressor = LinearRegression()
regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)

print("Visualizing Training dataset results ")

plt.scatter(X_train, y_train, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')

plt.title('Salary vs Experience {Training set}')
plt.xlabel('Years of experience')
plt.ylabel('Salary')

plt.show()
```

## Data Science – Machine Learning –Linear Regression Example

### Output

Visualizing Training dataset results



## Data Science – Machine Learning –Linear Regression Example

**Program Name** Plotting test dataset demo7.py

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

dataset = pd.read_csv('Salary_Data.csv')

X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, random_state = 0)

regressor = LinearRegression()
regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)

print("Visualizing Training dataset results ")

plt.scatter(X_test, y_test, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')

plt.title('Salary vs Experience {Test set}')
plt.xlabel('Years of experience')
plt.ylabel('Salary')

plt.show()
```

## Data Science – Machine Learning –Linear Regression Example

### Output

Visualizing Training dataset results



**10. Data Science – Machine Learning – Polynomial Features****Contents**

<b>1. Polynomial Features for machine learning .....</b>	<b>2</b>
<b>2. Need of Polynomial Features.....</b>	<b>2</b>
<b>3. Equations .....</b>	<b>3</b>
<b>4. Lets create Polynomial features .....</b>	<b>19</b>

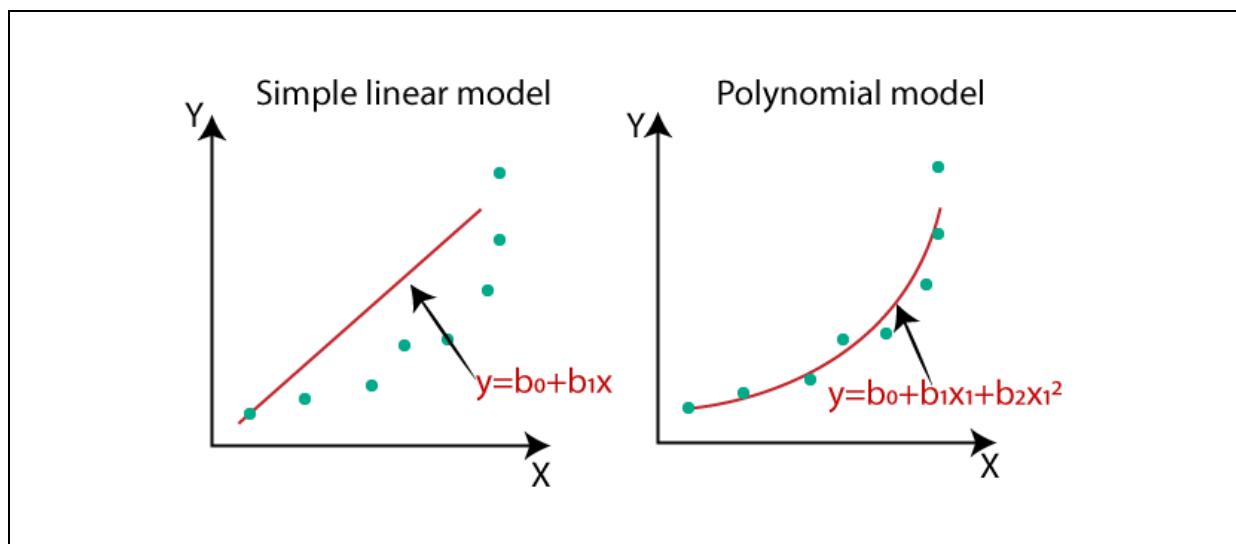
## 10. Data Science – Machine Learning – Polynomial Features

### 1. Polynomial Features for machine learning

- ✓ As such, polynomial features are a type of feature engineering means creating new input features based on the existing features.
- ✓ For example, if a dataset had one input feature X, then a polynomial feature would be the addition of a new feature (column) where values were calculated by squaring the values in X, e.g.  $X^2$ .
- ✓ This process can be repeated for each input variable in the dataset, creating a transformed version of each.
- ✓ The "degree" of the polynomial is used to control the number of features added.

### 2. Need of Polynomial Features

- ✓ If we apply a linear model on a linear dataset, then it provides us a good result as we have seen in Simple Linear Regression.
- ✓ If we apply the same model without any modification on a non-linear dataset, then it will produce wrong results
- ✓ Due to this,
  - The error rate will be high etc



## Data Science – Machine Learning – Polynomial Features

### 3. Equations

Simple Linear Regression equation

✓  $y = b_0 + b_1 x$

Multiple Linear Regression equation

✓  $y = b_0 + b_1 x_1 + b_2 x_2 + b_3 x_3 + \dots + b_n x_n$

Polynomial Regression equation:  $y = b_0 + b_1 x + b_2 x^2 + b_3 x^3 + \dots + b_n x^n$

## Data Science – Machine Learning – Polynomial Features

---

**Program Name** Creating an array  
demo1.py

```
from numpy import asarray  
  
data = asarray([[2], [3], [4]])  
print(data)
```

**Output**

```
[[2]  
[3]  
[4]]
```

## Data Science – Machine Learning – Polynomial Features

**Program Name** Creating feature from existing feature  
demo2.py

```
from numpy import asarray
from sklearn.preprocessing import PolynomialFeatures

data1 = asarray([[2],[3],[4]])

trans = PolynomialFeatures(degree = 1)
data2 = trans.fit_transform(data1)

print(data1)
print()
print(data2)
```

**Output**

```
[[2]
 [3]
 [4]]

[[1. 2.]
 [1. 3.]
 [1. 4.]]
```

## Data Science – Machine Learning – Polynomial Features

**Program Name** Creating feature from existing feature  
demo3.py

```
from numpy import asarray
from sklearn.preprocessing import PolynomialFeatures

data1 = asarray([[2],[3],[4]])

trans = PolynomialFeatures(degree = 2)
data2 = trans.fit_transform(data1)

print(data1)
print()
print(data2)
```

**Output**

```
[[2]
 [3]
 [4]]

[[ 1.  2.  4.]
 [ 1.  3.  9.]
 [ 1.  4. 16.]]
```

## Data Science – Machine Learning – Polynomial Features

**Program Name** Creating feature from existing feature  
demo4.py

```
from numpy import asarray
from sklearn.preprocessing import PolynomialFeatures

data1 = asarray([[2],[3],[4]])

trans = PolynomialFeatures(degree = 3)
data2 = trans.fit_transform(data1)

print(data1)
print()
print(data2)
```

**Output**

```
[[2]
 [3]
 [4]]

[[ 1.  2.  4.  8.]
 [ 1.  3.  9. 27.]
 [ 1.  4. 16. 64.]]
```

## Data Science – Machine Learning – Polynomial Features

**Program Name** Creating feature from existing feature  
demo5.py

```
from numpy import asarray

data1 = asarray([[2, 3],[4, 5],[6, 7]])

print(data1)
```

**Output**

```
[[2 3]
 [4 5]
 [6 7]]
```

## Data Science – Machine Learning – Polynomial Features

**Program Name** Creating feature from existing feature  
demo6.py

```
from numpy import asarray
from sklearn.preprocessing import PolynomialFeatures

data1 = asarray([[2, 3],[4, 5],[6, 7]])

trans = PolynomialFeatures(degree = 1)
data2 = trans.fit_transform(data1)

print(data1)
print()
print(data2)
```

**Output**

```
[[2 3]
 [4 5]
 [6 7]]

[[1.  2.  3.]
 [1.  4.  5.]
 [1.  6.  7.]]
```

## Data Science – Machine Learning – Polynomial Features

**Program Name** Creating feature from existing feature  
demo7.py

```
from numpy import asarray
from sklearn.preprocessing import PolynomialFeatures

data1 = asarray([[2, 3],[4, 5],[6, 7]])

trans = PolynomialFeatures(degree = 2)
data2 = trans.fit_transform(data1)

print(data1)
print()
print(data2)
```

**Output**

```
[[2 3]
 [4 5]
 [6 7]]

[[ 1.  2.  3.  4.  6.  9.]
 [ 1.  4.  5. 16. 20. 25.]
 [ 1.  6.  7. 36. 42. 49.]]
```

## Data Science – Machine Learning – Polynomial Features

**Program Name** Creating feature from existing feature  
demo8.py

```
from numpy import asarray
from sklearn.preprocessing import PolynomialFeatures

data1 = asarray([[2, 3],[4, 5],[6, 7]])

trans = PolynomialFeatures(degree = 3)
data2 = trans.fit_transform(data1)

print(data1)
print()
print(data2)
```

**Output**

```
[[2 3]
 [4 5]
 [6 7]]

[[ 1.   2.   3.   4.   6.   9.   8.  12.  18.  27.]
 [ 1.   4.   5.  16.  20.  25.  64.  80. 100. 125.]
 [ 1.   6.   7.  36.  42.  49. 216. 252. 294. 343.]]
```

## Data Science – Machine Learning – Polynomial Features

Program Name Loading dataset  
demo9.py

```
import pandas as pd

df = pd.read_csv("poly_dataset.csv")

print(df)
```

### Output

	Position	Level	Salary
0	Business Analyst	1	45000
1	Junior Consultant	2	50000
2	Senior Consultant	3	60000
3	Manager	4	80000
4	Country Manager	5	110000
5	Region Manager	6	150000
6	Partner	7	200000
7	Senior Partner	8	300000
8	C-level	9	500000
9	CEO	10	1000000

## Data Science – Machine Learning – Polynomial Features

**Program Name** Data preparation  
demo10.py

```
import pandas as pd

df = pd.read_csv("poly_dataset.csv")

# Data preparation
X = df.iloc[:, 1:2].values
y = df.iloc[:, 2].values

print(X)
print()
print(y)
```

**Output**

```
[[ 1]
 [ 2]
 [ 3]
 [ 4]
 [ 5]
 [ 6]
 [ 7]
 [ 8]
 [ 9]
 [10]]

[ 45000   50000   60000   80000  110000  150000  200000  300000  500000
 1000000]
```

## Data Science – Machine Learning – Polynomial Features

**Program Name** Plotting the dataset  
demo11.py

```
import pandas as pd
import matplotlib.pyplot as plt

df=pd.read_csv("poly_dataset.csv")

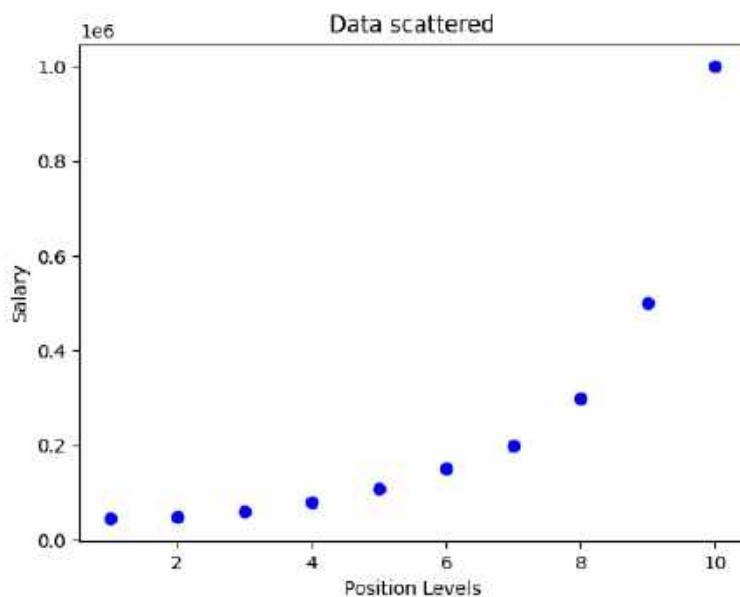
X = df.iloc[:, 1:2].values
y = df.iloc[:, 2].values

plt.scatter(X, y, color="blue")

plt.title("Data scattered")
plt.xlabel("Position Levels")
plt.ylabel("Salary")

plt.show()
```

### Output



## Data Science – Machine Learning – Polynomial Features

**Program Name** Model training  
demo12.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

# Loading the dataset
df = pd.read_csv("poly_dataset.csv")

# Data preparation
X = df.iloc[:, 1:2].values
y = df.iloc[:, 2].values

# Model training
lin_regs= LinearRegression()
lin_regs.fit(X, y)

print("Model got trained")
```

**Output**

Model got trained

## Data Science – Machine Learning – Polynomial Features

**Program Name** Model prediction  
demo13.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

# Loading the dataset
df = pd.read_csv("poly_dataset.csv")

# Data preparation
X = df.iloc[:, 1:2].values
y = df.iloc[:, 2].values

# Model training
lin_regs= LinearRegression()
lin_regs.fit(X, y)

print("Model got trained")
print(lin_regs.predict([[6.5]]))
```

**Output**

```
[330378.78787879]
```

## Data Science – Machine Learning – Polynomial Features

**Program Name** Plotting the dataset  
demo14.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Loading the dataset
df=pd.read_csv("poly_dataset.csv")

# Data preparation
X = df.iloc[:, 1:2].values
y = df.iloc[:, 2].values

# Model training
lin_regs= LinearRegression()
lin_regs.fit(X, y)

plt.scatter(X, y, color="blue")

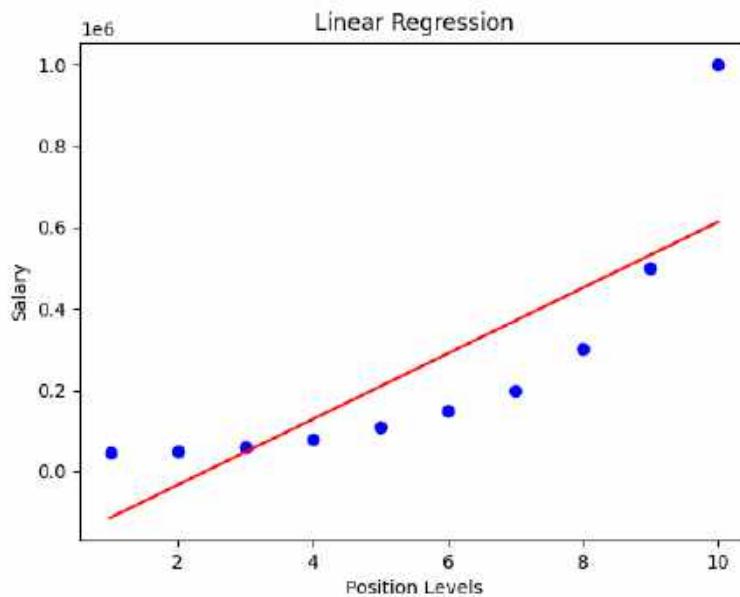
plt.plot(X, lin_regs.predict(X), color = "red")

plt.title("Linear Regression")
plt.xlabel("Position Levels")
plt.ylabel("Salary")

plt.show()
```

## Data Science – Machine Learning – Polynomial Features

Output



#### 4. Lets create Polynomial features

- ✓ We need to use PolynomialFeatures class to get polynomial features.

Program

Name

Fitting the Polynomial regression to the dataset

demo15.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

# Loading the dataset
df=pd.read_csv("poly_dataset.csv")

# Data preparation
X = df.iloc[:, 1:2].values
y = df.iloc[:, 2].values

poly_regs= PolynomialFeatures(degree = 1)

x_poly= poly_regs.fit_transform(X)

print(x_poly)
```

## Data Science – Machine Learning – Polynomial Features

### Output

```
          Position  Level   Salary
0  Business Analyst    1    45000
1  Junior Consultant   2    50000
2  Senior Consultant   3    60000
3        Manager        4    80000
4  Country Manager     5   110000
5  Region Manager      6   150000
6        Partner        7   200000
7  Senior Partner      8   300000
8        C-level         9   500000
9        CEO            10  1000000
[[ 1.  1.]
 [ 1.  2.]
 [ 1.  3.]
 [ 1.  4.]
 [ 1.  5.]
 [ 1.  6.]
 [ 1.  7.]
 [ 1.  8.]
 [ 1.  9.]
 [ 1. 10.]]
```

## Data Science – Machine Learning – Polynomial Features

**Program Name** Fitting the Polynomial regression to the dataset  
demo16.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

# Loading the dataset
df=pd.read_csv("poly_dataset.csv")

# Data preparation
X = df.iloc[:, 1:2].values
y = df.iloc[:, 2].values

poly_regs= PolynomialFeatures(degree = 2)

x_poly= poly_regs.fit_transform(X)

print(x_poly)
```

**Output**

	Position	Level	Salary
0	Business Analyst	1	45000
1	Junior Consultant	2	50000
2	Senior Consultant	3	60000
3	Manager	4	80000
4	Country Manager	5	110000
5	Region Manager	6	150000
6	Partner	7	200000
7	Senior Partner	8	300000
8	C-level	9	500000
9	CEO	10	1000000

```
[[ 1.  1.  1.]
 [ 1.  2.  4.]
 [ 1.  3.  9.]
 [ 1.  4. 16.]
 [ 1.  5. 25.]
 [ 1.  6. 36.]
 [ 1.  7. 49.]
 [ 1.  8. 64.]
 [ 1.  9. 81.]
 [ 1. 10. 100.]]
```

## Data Science – Machine Learning – Polynomial Features

**Program Name** Fitting the Polynomial regression to the dataset  
demo17.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

# Loading the dataset
df=pd.read_csv("poly_dataset.csv")

# Data preparation
X = df.iloc[:, 1:2].values
y = df.iloc[:, 2].values

poly_regs= PolynomialFeatures(degree = 2)
x_poly= poly_regs.fit_transform(X)

model = LinearRegression()
model.fit(x_poly, y)

print("Fitting the Polynomial regression to the dataset ")
```

**Output**

Fitting the Polynomial regression to the dataset

## Data Science – Machine Learning – Polynomial Features

**Program Name** Plotting Polynomial Regression features  
demo18.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
import matplotlib.pyplot as plt

# Loading the dataset
df=pd.read_csv("poly_dataset.csv")

# Data preparation
X = df.iloc[:, 1:2].values
y = df.iloc[:, 2].values

#Fitting the Polynomial regression to the dataset
poly_regs= PolynomialFeatures(degree = 2)
x_poly = poly_regs.fit_transform(X)
model =LinearRegression()
model.fit(x_poly, y)

# Plotting Polynomial Regression

plt.scatter(X, y, color="blue")

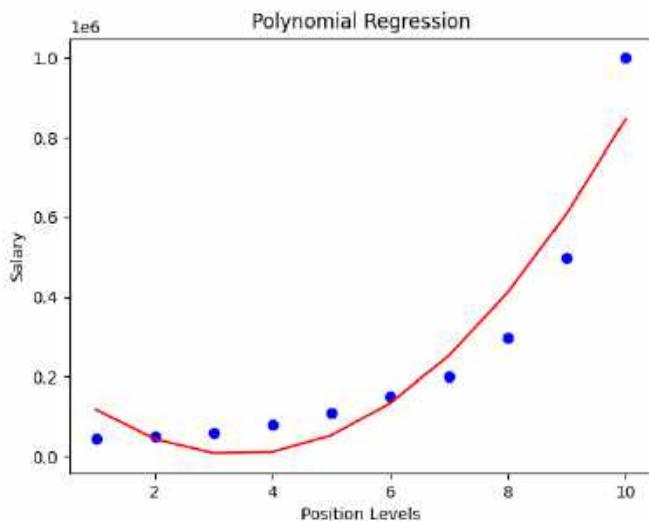
plt.plot(X, model.predict(x_poly), color="red")

plt.title("Polynomial Regression")
plt.xlabel("Position Levels")
plt.ylabel("Salary")

plt.show()
```

## Data Science – Machine Learning – Polynomial Features

Output



## Data Science – Machine Learning – Polynomial Features

**Program Name** Plotting Polynomial Regression features  
demo19.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
import matplotlib.pyplot as plt

# Loading the dataset
df=pd.read_csv("poly_dataset.csv")

# Data preparation
X = df.iloc[:, 1:2].values
y = df.iloc[:, 2].values

#Fitting the Polynomial regression to the dataset
poly_regs= PolynomialFeatures(degree = 3)
x_poly = poly_regs.fit_transform(X)
model =LinearRegression()
model.fit(x_poly, y)

# Plotting Polynomial Regression

plt.scatter(X, y, color="blue")

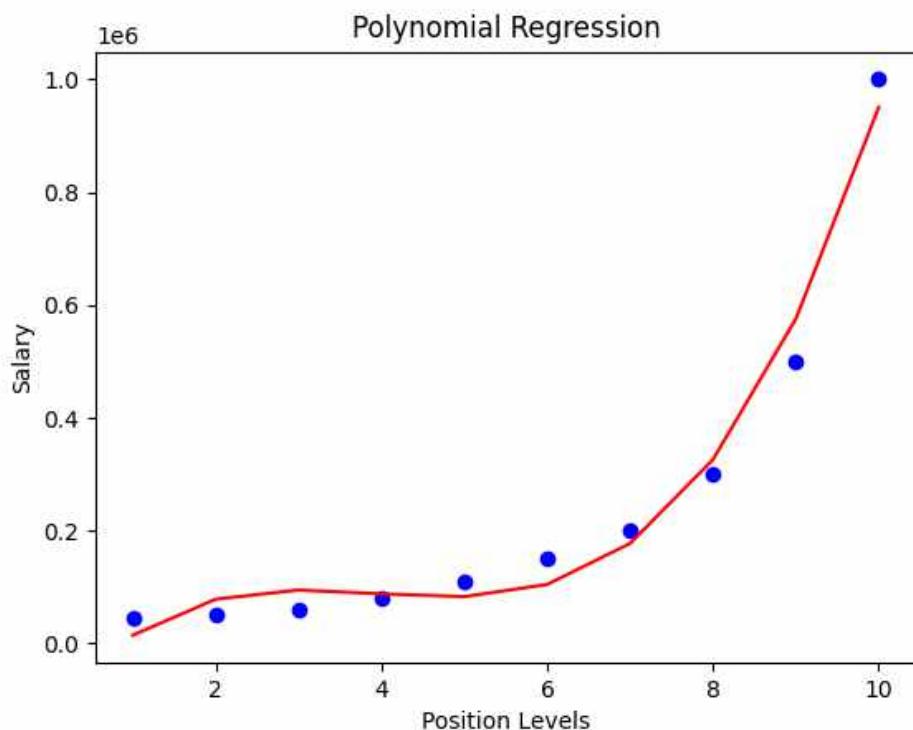
plt.plot(X, model.predict(x_poly), color="red")

plt.title("Polynomial Regression")
plt.xlabel("Position Levels")
plt.ylabel("Salary")

plt.show()
```

## Data Science – Machine Learning – Polynomial Features

Output



## Data Science – Machine Learning – Polynomial Features

**Program Name** Plotting Polynomial Regression features  
demo20.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
import matplotlib.pyplot as plt

# Loading the dataset
df=pd.read_csv("poly_dataset.csv")

# Data preparation
X = df.iloc[:, 1:2].values
y = df.iloc[:, 2].values

#Fitting the Polynomial regression to the dataset
poly_regs= PolynomialFeatures(degree = 4)
x_poly = poly_regs.fit_transform(X)
model =LinearRegression()
model.fit(x_poly, y)

# Plotting Polynomial Regression

plt.scatter(X, y, color="blue")

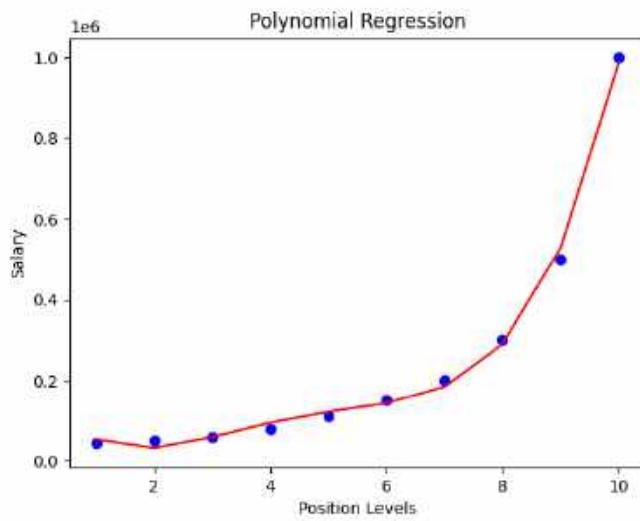
plt.plot(X, model.predict(x_poly), color="red")

plt.title("Polynomial Regression")
plt.xlabel("Position Levels")
plt.ylabel("Salary")

plt.show()
```

## Data Science – Machine Learning – Polynomial Features

Output



## Data Science – Machine Learning – Polynomial Features

**Program**

**Name** Predicting result

demo21.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
import matplotlib.pyplot as plt

# Loading the dataset
df=pd.read_csv("poly_dataset.csv")

# Data preparation
X = df.iloc[:, 1:2].values
y = df.iloc[:, 2].values

#Fitting the Polynomial regression to the dataset
poly_regs= PolynomialFeatures(degree = 4)
x_poly= poly_regs.fit_transform(X)
model =LinearRegression()
model.fit(x_poly, y)

# Prediction with Polynomial Regression
poly_pred = model.predict(poly_regs.fit_transform([[6.5]]))
print(poly_pred)
```

**Output**

[158862.45265155]

## Data Science – Machine Learning – Polynomial Features

---

### Note

- ✓ LinearRegression predicted output is : [330378.78787879]
- ✓ Polynomial Regression predicted output is : [158862.45265155]

### Conclusion

- ✓ Polynomial Regression predicted output is the **accurate** one according to the discussion

**11. Data Science – Machine Learning – Multiple Linear Regression****Contents**

<b>1. Multiple Linear Regression .....</b>	2
<b>2. Problem statement .....</b>	2
<b>3. Dataset.....</b>	3
<b>4. Machine learning Terminology .....</b>	4
4.1. Features and label.....	4
4.2. Models .....	4
4.3. Prediction.....	4
4.4. Formula .....	5

## 11. Data Science – Machine Learning – Multiple Linear Regression

### 1. Multiple Linear Regression

- ✓ Multiple Linear Regression explains the relationship between a single dependent continuous variable and more than one independent variable.

### 2. Problem statement

- ✓ Assuming that we are planning to buy a new house and need to predict the price of a house.
- ✓ Here price depends on area (square feet), bed rooms and age of the home (in years).
- ✓ Given these prices we have to predict prices of new homes based on area, bed rooms and age
- ✓ Given these home prices find out price of a home that has,
  - 3000 sqr ft area, 3 bedrooms, 40 year old
  - 2500 sqr ft area, 4 bedrooms, 5 year old

## Data Science – Machine Learning – Multiple Linear Regression

---

### 3. Dataset

- ✓ homeprices1.csv is dataset we are using in this example
- ✓ This dataset contains columns as,
  - Area
  - Bedrooms
  - Age
  - Price

Area	Bedrooms	Age	price
2600	3	20	550000
3000	4	15	565000
3200		18	610000
3600	3	30	595000
4000	5	8	760000
4100	6	8	810000

## 4. Machine learning Terminology

### 4.1. Features and label

- ✓ Here **area**, **bedrooms**, **age** are called independent variables or features whereas price is a dependant variable

### 4.2. Models

- ✓ A machine learning model is simply a rule, or a formula, which predicts a label from the features.
- ✓ In this case, the model is the equation we found for the price.

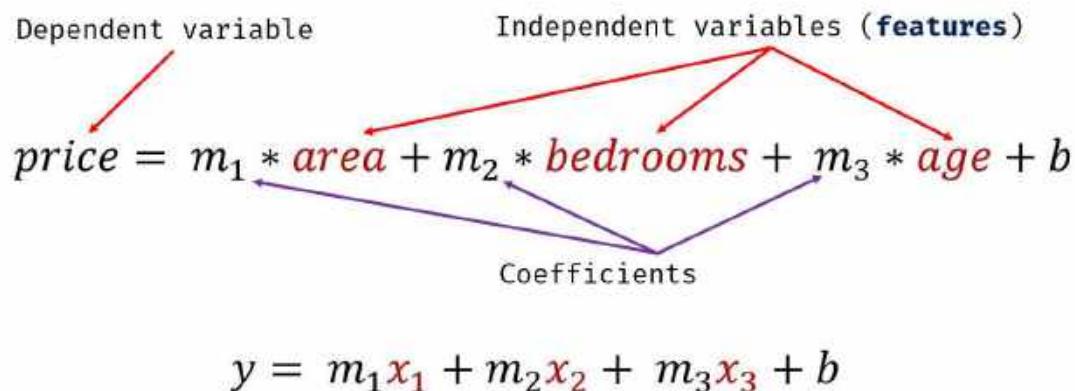
### 4.3. Prediction

- ✓ The prediction is simply the output of the model.

#### 4.4. Formula

$$y = m_1 x_1 + m_2 x_2 + m_3 x_3 + b$$

$$price = m_1 * area + m_2 * bedrooms + m_3 * age + b$$



## Data Science – Machine Learning – Multiple Linear Regression

**Program Name** Loading house prices dataset  
demo1.py

```
import pandas as pd

# Loading the dataset

df = pd.read_csv("homeprices1.csv")

print(df)
```

**Output**

	area	bedrooms	age	price
0	2600	3.0	20	550000
1	3000	4.0	15	565000
2	3200	NaN	18	610000
3	3600	3.0	30	595000
4	4000	5.0	8	760000
5	4100	6.0	8	810000

## Data Science – Machine Learning – Multiple Linear Regression

---

**Program Name** Data pre-processing – Finding mean of bedrooms column  
demo2.py

```
import pandas as pd

df = pd.read_csv("homeprices1.csv")

# Mean of the bedrooms
print("Mean of the bedrooms")
print(df.bedrooms.median())
```

**Output**

```
Mean of the bedrooms
4.0
```

## Data Science – Machine Learning – Multiple Linear Regression

**Program Name** Data pre-processing - Fill NA values with median value of a column  
demo3.py

```
import pandas as pd

df = pd.read_csv("homeprices1.csv")

# Data loading
print("Filling missing value with mean\n")

# Data preprocessing

m = df.bedrooms.median()
df.bedrooms = df.bedrooms.fillna(m)
print(df)
```

**Output**

```
Filling missing value with mean

      area  bedrooms   age   price
0    2600        3.0    20  550000
1    3000        4.0    15  565000
2    3200        4.0    18  610000
3    3600        3.0    30  595000
4    4000        5.0     8  760000
5    4100        6.0     8  810000
```

## Data Science – Machine Learning – Multiple Linear Regression

**Program Name** Model training  
demo4.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

# Data loading
df = pd.read_csv("homeprices1.csv")

# Data preprocessing
m = df.bedrooms.median()
df.bedrooms = df.bedrooms.fillna(m)
a = df.drop('price', axis = 'columns')

# Model training
reg = LinearRegression()
reg.fit(a.values, df.price)
print("Model trained")
```

**Output**  
Model trained

## Data Science – Machine Learning – Multiple Linear Regression

**Program Name**

Finding intercept

demo5.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

# Data loading
df=pd.read_csv("homeprices1.csv")

# Data preprocessing
m = df.bedrooms.median()
df.bedrooms = df.bedrooms.fillna(m)
a = df.drop('price', axis = 'columns')

# Model training
reg = LinearRegression()
reg.fit(a.values, df.price)

print("Intercept is:")
print(reg.intercept_)
```

**Output**

```
Intercept is:
221323.00186540408
```

## Data Science – Machine Learning – Multiple Linear Regression

**Program Name** Finding coefficients  
demo6.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

# Data loading
df = pd.read_csv("homeprices1.csv")

# Data preprocessing
m = df.bedrooms.median()
df.bedrooms = df.bedrooms.fillna(m)
a = df.drop('price', axis = 'columns')

# Model training
reg = LinearRegression()
reg.fit(a.vlaues, df.price)

print("Coefficients are:")
print(reg.coef_)
```

### Output

Coefficients are:  
[ 112.06244194 23388.88007794 -3231.71790863]

## Data Science – Machine Learning – Multiple Linear Regression

**Program Name** price of home with 3000 sqr ft area, 3 bedrooms, 40 year old  
demo7.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

# Data loading
df = pd.read_csv("homeprices1.csv")

# Data preprocessing
m = df.bedrooms.median()
df.bedrooms = df.bedrooms.fillna(m)
a = df.drop('price', axis='columns')

# Model training
reg = LinearRegression()
reg.fit(a.values, df.price)

# Prediction
print("price of home with 3000 sqr ft area, 3 bedrooms, 40 year old")
print(reg.predict([[3000, 3, 40]]))
```

### Output

price of home with 3000 sqr ft area, 3 bedrooms, 40 year old  
[498408.25158031]

## Data Science – Machine Learning – Multiple Linear Regression

**Program Name** price of home with 3000 sqr ft area, 3 bedrooms, 40 year old  
demo8.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

# Data loading
df = pd.read_csv("homeprices1.csv")

# Data preprocessing
m = df.bedrooms.median()
df.bedrooms = df.bedrooms.fillna(m)
a = df.drop('price', axis = 'columns')

# Model training
reg = LinearRegression()
reg.fit(a.values, df.price)

# Prediction
print("price of home with 3000 sqr ft area, 3 bedrooms, 40 year old")

b = 112.06244194*3000 + 23388.88007794*3 + -
3231.71790863*40 + 221323.00186540384
print(b)
```

### Output

```
price of home with 3000 sqr ft area, 3 bedrooms, 40 year old
[498408.25158031]
```

## Data Science – Machine Learning – Multiple Linear Regression

**Program Name** price of home with 2500 sqr ft area, 4 bedrooms, 5 year old  
demo9.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

# Data loading
df = pd.read_csv("homeprices1.csv")

# Data preprocessing
m = df.bedrooms.median()
df.bedrooms = df.bedrooms.fillna(m)
a = df.drop('price', axis = 'columns')

# Model training
reg = LinearRegression()
reg.fit(a.values, df.price)

# Prediction
print("price of home with 2500 sqr ft area, 4 bedrooms, 5 year old")

print(reg.predict([[2500, 4, 5]]))
```

### Output

price of home with 2500 sqr ft area, 4 bedrooms, 5 year old  
[578876.03748933]

## Data Science – Machine Learning – Pickling and Unpickling

---

### 12. Data Science – Machine Learning – Pickling and Unpickling

#### Contents

1. Pickling.....	2
2. Unpickling .....	2

**12. Data Science – Machine Learning – Pickling and Unpickling**

- ✓ Based on requirement we can write total state of object to the file and we can read total object information from the file.

**1. Pickling**

- ✓ The process of writing state of object to the file is called as **pickling**.
- ✓ We can implement pickling and unpickling by using python **pickle module**
- ✓ We can do pickling by using `dump(object, file)` predefined function in pickle module

▪ `pickle.dump(object, file)`

**2. Unpickling**

- ✓ The process of reading state of an object from the file is called **unpickling**.
- ✓ We can do unpickling by using `load(file)` predefined function in pickle module

▪ `obj = pickle.load(file)`

## Data Science – Machine Learning – Pickling and Unpickling

---

**Program Name** Pickling and unpickling example  
demo1.py

```
import pickle

class Employee:
    def __init__(self, eno, ename, esal):
        self.eno = eno
        self.ename = ename
        self.esal = esal

    def display(self):
        print("Number is:", self.eno)
        print("Name is:", self.ename)
        print("Salary is:", self.esal)

with open("emp.dat", "wb") as f:
    e = Employee(100, "Daniel", 1000)
    pickle.dump(e, f)
    print("Pickling of Employee Object completed...\n")

with open("emp.dat", "rb") as f:
    obj = pickle.load(f)
    print("Printing Employee Information after unpickling")
    obj.display()
```

### Output

Pickling of Employee Object completed...

Printing Employee Information after unpickling  
 Number is: 100  
 Name is: Daniel  
 Salary is: 1000

## Data Science – Machine Learning – Saving model, Joblib & Pickling

---

### 13. Data Science – Machine Learning – Saving model, Joblib & Pickling

#### Contents

1. Save model .....	2
2. Pickling.....	3

**13. Data Science – Machine Learning – Saving model, Joblib & Pickling****1. Save model**

- ✓ Once the model got created then we can save that model.
- ✓ There are two ways to save the model
  - By using python File IO pickle concept
  - By using Joblib library

**Program Name** Predict price of a home with area = 5000 sqr ft  
demo1.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

df = pd.read_csv("homeprices.csv")
new_df = df.drop('price', axis='columns')

reg = LinearRegression()
reg.fit(new_df, df.price)

# Predictions
print(reg.predict([[5000]]))
```

**Output**

```
[859554.79452055]
```

## Data Science – Machine Learning – Saving model, Joblib & Pickling

### 2. Pickling

- ✓ The process of writing state of object to the file is called as **pickling**.

**Program Name** Save the model into pickle file  
demo2.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression
import pickle

df = pd.read_csv("homeprices.csv")
new_df = df.drop('price', axis='columns')

model = LinearRegression()
model.fit(new_df.values, df.price.values)

with open('model_pickle', 'wb') as file:
    pickle.dump(model, file)

with open('model_pickle', 'rb') as file:
    model1 = pickle.load(file)
    print(model1.predict([[5000]]))
```

**Output**

[859554.79452055]

## Data Science – Machine Learning – Saving model, Joblib & Pickling

**Program Name** Doing prediction by using saved model  
demo3.py

```
import pickle

with open('model_pickle', 'rb') as file:
    model1 = pickle.load(file)
    print(model1.predict([[5000]]))
```

**Output**

```
[859554.79452055]
```

**Program Name** Doing prediction by using saved model  
demo4.py

```
import pickle

with open('model_pickle', 'rb') as file:
    model1 = pickle.load(file)
    print(model1.predict([[6000]]))
```

**Output**

```
[995342.46575342]
```

## Data Science – Machine Learning – Saving model, Joblib & Pickling

**Program Name** Save Trained Model Using Joblib  
demo5.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression
import joblib

df = pd.read_csv("homeprices.csv")
new_df = df.drop('price', axis='columns')

model = LinearRegression()
model.fit(new_df.values, df.price.values)

joblib.dump(model, 'model_joblib')

print("Model got saved")
```

**Output**

Model got saved

## Data Science – Machine Learning – Saving model, Joblib & Pickling

**Program Name** Do prediction with saved model  
demo6.py

```
import joblib

mj = joblib.load('model_joblib')

print(mj.predict([[5000]]))
```

**Output**

```
[859554.79452055]
```

**14. Data Science – Machine Learning – Cost function****Contents**

<b>1. Cost Functions in Machine Learning .....</b>	2
<b>2. Why Cost Functions in Machine Learning .....</b>	2
<b>3. How cost function works?.....</b>	2
<b>4. Goal of training.....</b>	2
<b>5. Useful sources .....</b>	3

## 14. Data Science – Machine Learning – Cost function

### 1. Cost Functions in Machine Learning

- ✓ Cost functions are also called as Loss functions in machine learning.
- ✓ These are **optimization** techniques.
- ✓ There are many cost functions in machine learning
- ✓ There are different cost functions for regression and classification.

### 2. Why Cost Functions in Machine Learning

- ✓ During the training phase, the model assumes the initial weights randomly and tries to make a prediction on training data.
- ✓ So here, how the model gets to know how much "far" it was from the prediction?
- ✓ Model needs this information so that it can adjust the weight accordingly (using gradient descent) in the next iteration on training data.

### 3. How cost function works?

- ✓ Cost function is a function that takes both **predicted outputs** and **actual outputs** from model.
- ✓ Then it calculates how much error in the model to predict the good results.
- ✓ In next step the model tries to adjust this error for the next iteration on training data to reach optimization level

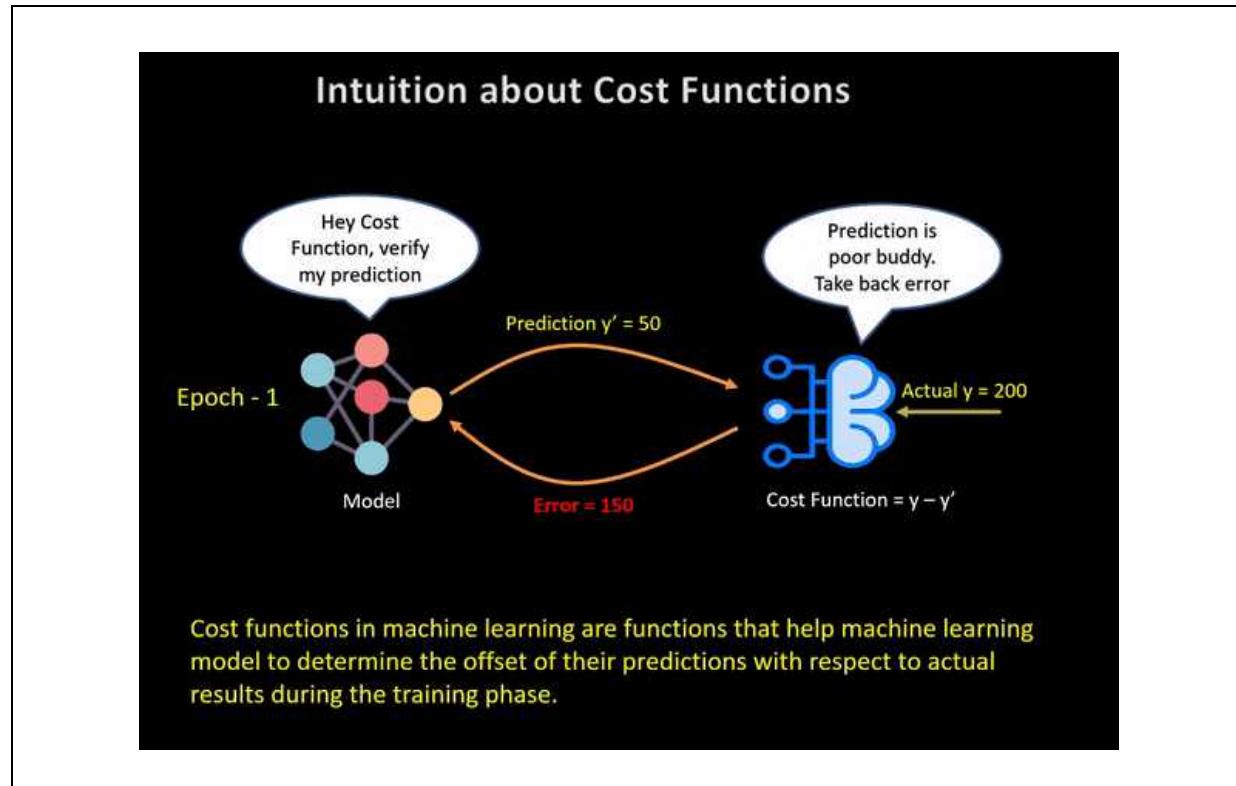
### 4. Goal of training

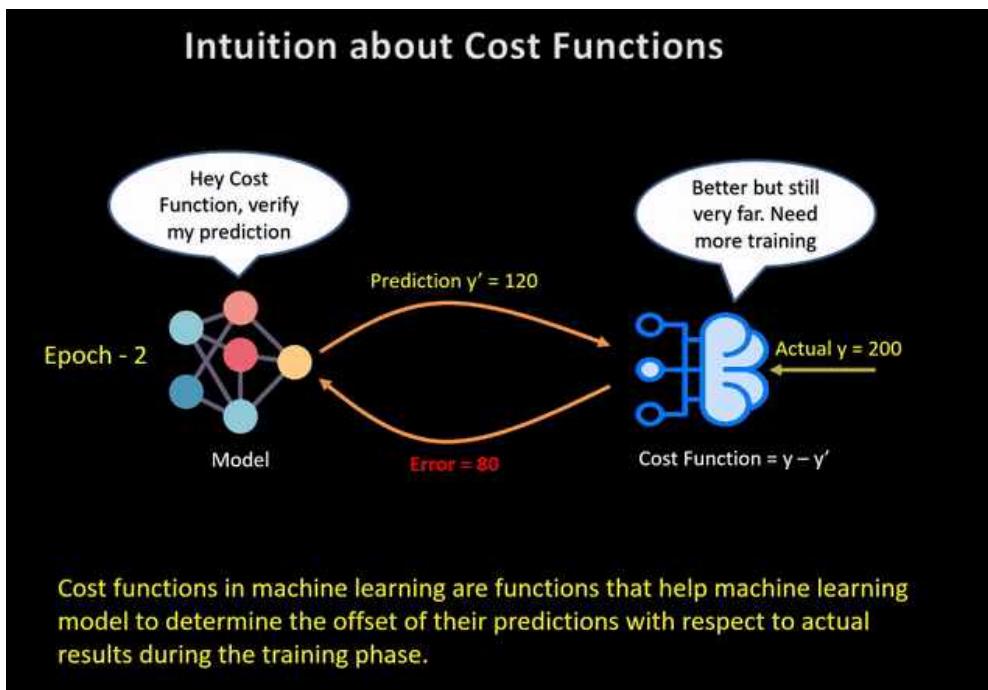
- ✓ The goal of the training phase is to come up with weights that minimize the **error in every iteration**.
- ✓ This way of training is required in machine learning to optimize model

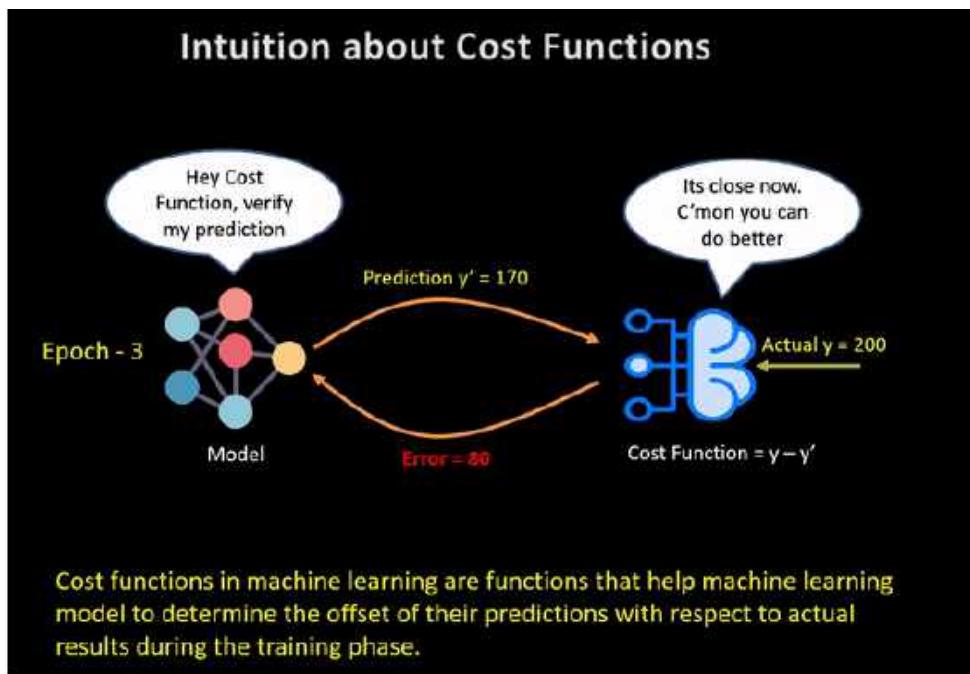
## Data Science – Machine Learning – Cost function

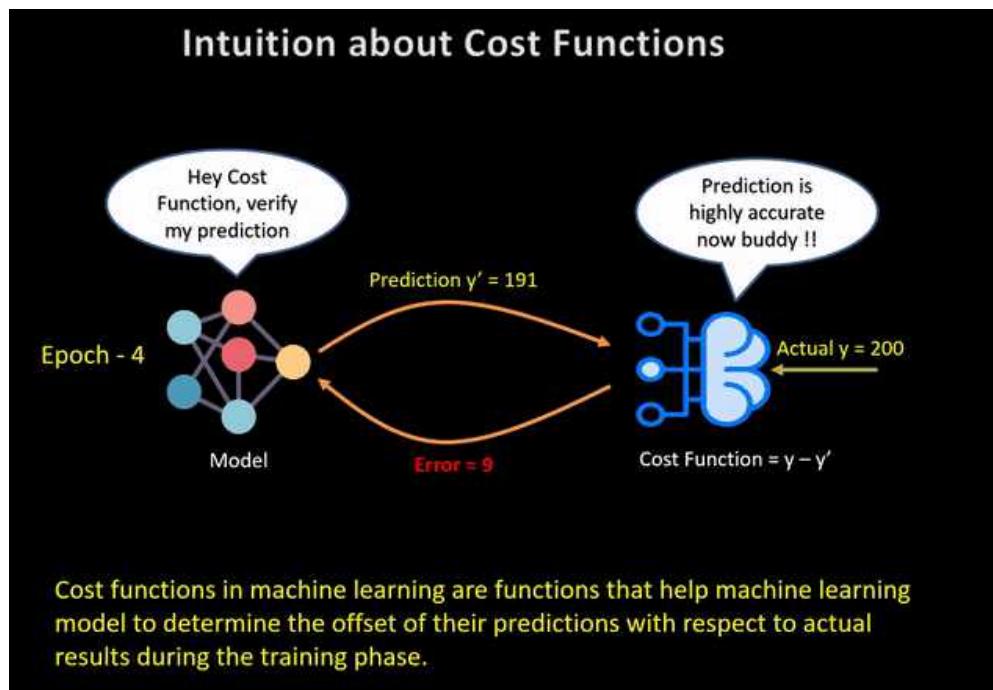
### 5. Useful sources

- ✓ These images are highly recommended to practice









**15. Data Science – Machine Learning – Regression Cost functions****Contents**

<b>1. Cost Functions for Regression.....</b>	<b>2</b>
<b>2. Types of regression metrics .....</b>	<b>2</b>
<b>3. Distance Based Error .....</b>	<b>3</b>
<b>4. Mean Squared Error .....</b>	<b>4</b>
<b>5. Root Mean Squared Error .....</b>	<b>6</b>
<b>6. Mean Absolute Error.....</b>	<b>7</b>
<b>7. Formulas .....</b>	<b>8</b>

## 15. Data Science – Machine Learning – Regression Cost functions

### 1. Cost Functions for Regression

- ✓ In regression, the model predicts an output value for each training data during the training phase.
- ✓ The cost functions for regression are calculated on **distance-based error**.

### 2. Types of regression metrics

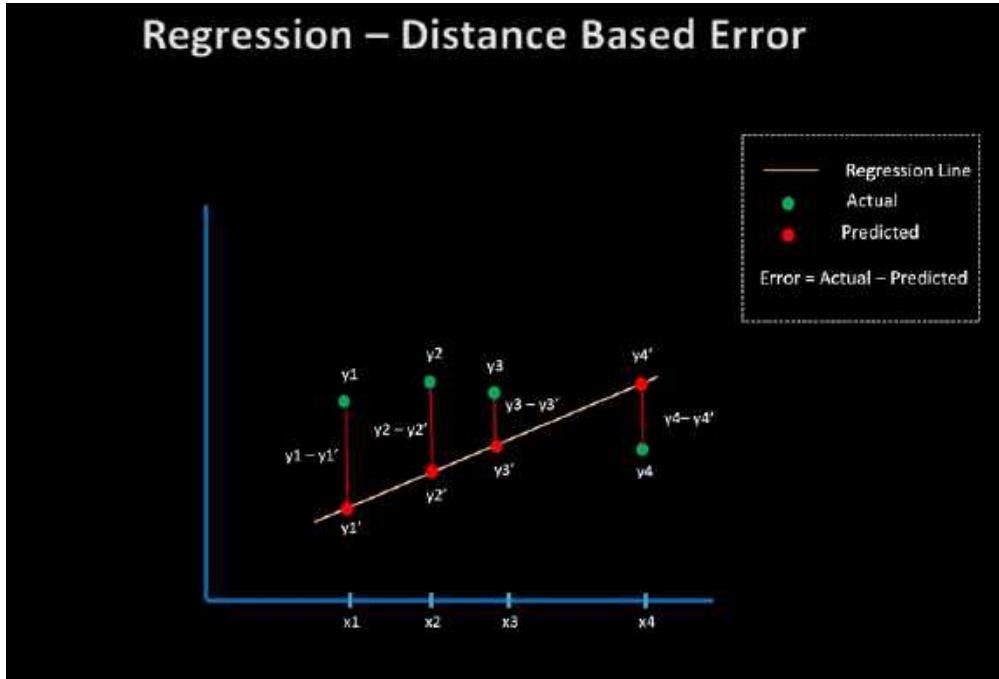
- ✓ There are three metrics in regression,
  - Mean Squared Error (MSE).
  - Root Mean Squared Error (RMSE)
  - Mean Absolute Error (MAE)

### 3. Distance Based Error

- Assuming that for a given set of input data, the **actual** output was  $y$  and our regression model **predicts  $y'$**  then the error in prediction is calculated as

■  $\text{Error} = y - y'$

- This also known as **distance-based error** and it forms the basis of cost functions that are used in regression models.



#### 4. Mean Squared Error

- ✓ The MSE is calculated as the **mean of the squared differences between predicted and expected target values** in a dataset.
- ✓ This value never be negative, since we are always squaring the errors.

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

## Data Science – Machine Learning – Regression Cost functions

**Program Name** Means Squared Error  
demo1.py

```
from sklearn.metrics import mean_squared_error

expected = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
predicted = [1.0, 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0.0]

mse_value = mean_squared_error(expected, predicted)

print(mse_value)
```

**Output**

```
0.3500000000000003
```

## 5. Root Mean Squared Error

- ✓ The Root Mean Squared Error, or RMSE, is an extension of the mean squared error.
  - $\text{RMSE} = \sqrt{\text{MSE}}$

<p><b>Program Name</b></p>	Root Means Squared Error demo2.py
	<pre>from sklearn.metrics import mean_squared_error  expected = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0] predicted = [1.0, 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0.0]  rmse_value = mean_squared_error(expected, predicted, squared = False)  print(rmse_value)</pre>
<b>Output</b>	0.5916079783099616

## 6. Mean Absolute Error

- ✓ MAE, average absolute difference between predicted and actual values.

**Program Name** Means Absolute Error  
demo3.py

```
from sklearn.metrics import mean_absolute_error

expected = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
predicted = [1.0, 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0.0]

mae_value = mean_absolute_error(expected, predicted)

print(mae_value)
```

**Output**

0.5

## 7. Formulas

---

Mean squared error

$$\text{MSE} = \frac{1}{n} \sum_{t=1}^n e_t^2$$

---

Root mean squared error

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{t=1}^n e_t^2}$$

---

Mean absolute error

$$\text{MAE} = \frac{1}{n} \sum_{t=1}^n |e_t|$$

---

**16. Data Science – ML – Dummy Variable & OneHotEncoding****Contents**

<b>1. Dataset : homeprices2.csv .....</b>	<b>2</b>
<b>2. Categorical data.....</b>	<b>3</b>
<b>3. How to handle text data in town column .....</b>	<b>4</b>
<b>4. Model may behaves like below, .....</b>	<b>4</b>
<b>5. Dummy variables.....</b>	<b>5</b>

## 16. Data Science – ML – Dummy Variable & OneHotEncoding

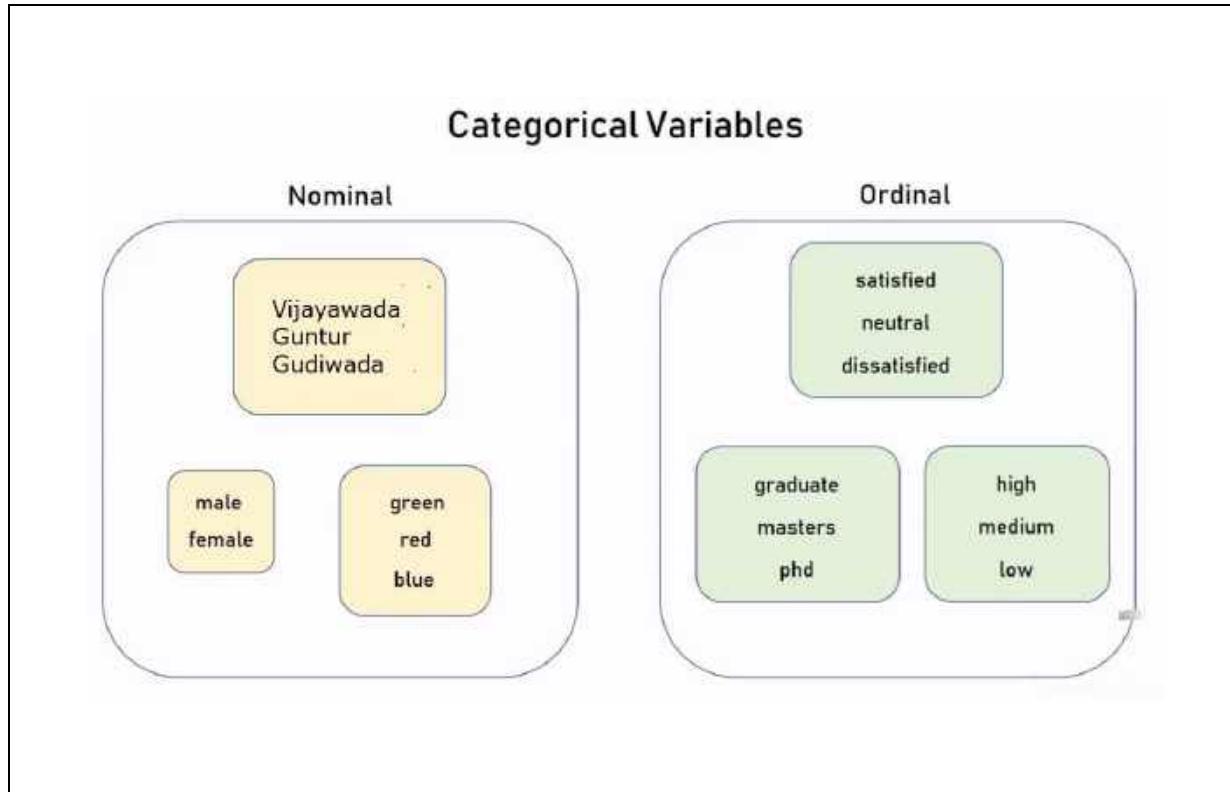
### 1. Dataset : homeprices2.csv

town	area	price
Vijayawada	2600	550000
Vijayawada	3000	565000
Vijayawada	3200	610000
Vijayawada	3600	680000
Vijayawada	4000	725000
Guntur	2600	585000
Guntur	2800	615000
Guntur	3300	650000
Guntur	3600	710000
Gudiwada	2600	575000
Gudiwada	2900	600000
Gudiwada	3100	620000
Gudiwada	3600	695000

### Prediction

- ✓ With 3400 sqr ft area in Guntur
- ✓ With 2800 sqr ft area in Gudiwada

## 2. Categorical data



**3. How to handle text data in town column**

- ✓ Here town column having text data
- ✓ How to handle text data in numeric model?

**4. Model may behaves like below,**

- ✓ We can convert text data into number and can proceed to the next step.
  - Vijayawada - 1
  - Guntur - 2
  - Gudiwada - 3
- ✓ If we are giving data to the model then model assumes that the order like 1, 2, 3 and follow the below approach
  - Vijayawada < Guntur < Gudiwada
    - Or
  - Vijayawada + Guntur = Gudiwada

## 5. Dummy variables

- ✓ So, in this scenario we need to create dummy variable

Town	Area	Price	Gudiwada	Guntur	Vijayawada
<b>Gudiwada</b>	2600	575000	1	0	0
Gudiwada	2900	600000	1	0	0
Gudiwada	3100	620000	1	0	0
Gudiwada	3600	695000	1	0	0
<b>Guntur</b>	2600	585000	0	1	0
Guntur	2800	615000	0	1	0
Guntur	3300	650000	0	1	0
Guntur	3600	710000	0	1	0
<b>Vijayawada</b>	2600	550000	0	0	1
Vijayawada	3000	565000	0	0	1
Vijayawada	3200	610000	0	0	1
Vijayawada	3600	680000	0	0	1
Vijayawada	4000	725000	0	0	1

## Data Science – ML – Dummy Variable & OneHotEncoding

Program Name Loading dataset  
demo1.py

```
import pandas as pd

df = pd.read_csv("homeprices2.csv")

print(df)
```

Output

	town	area	price
0	Vijayawada	2600	550000
1	Vijayawada	3000	565000
2	Vijayawada	3200	610000
3	Vijayawada	3600	680000
4	Vijayawada	4000	725000
5	Guntur	2600	585000
6	Guntur	2800	615000
7	Guntur	3300	650000
8	Guntur	3600	710000
9	Gudiwada	2600	575000
10	Gudiwada	2900	600000
11	Gudiwada	3100	620000
12	Gudiwada	3600	695000

## Data Science – ML – Dummy Variable & OneHotEncoding

**Program Name** Creating dummy variables by using pandas  
demo2.py

```
import pandas as pd

df = pd.read_csv("homeprices2.csv")
dummies = pd.get_dummies(df.town)

print(dummies)
```

**Output**

	Gudiwada	Guntur	Vijayawada
0	0	0	1
1	0	0	1
2	0	0	1
3	0	0	1
4	0	0	1
5	0	1	0
6	0	1	0
7	0	1	0
8	0	1	0
9	1	0	0
10	1	0	0
11	1	0	0
12	1	0	0

## Data Science – ML – Dummy Variable & OneHotEncoding

---

**Program Name** Creating dummy variables and adding to the dataframe  
demo3.py

```
import pandas as pd

df = pd.read_csv("homeprices2.csv")
dummies = pd.get_dummies(df.town)
merged = pd.concat([df, dummies], axis='columns')

print(merged)
```

### Output

	town	area	price	Gudiwada	Guntur	Vijayawada
0	Vijayawada	2600	550000	0	0	1
1	Vijayawada	3000	565000	0	0	1
2	Vijayawada	3200	610000	0	0	1
3	Vijayawada	3600	680000	0	0	1
4	Vijayawada	4000	725000	0	0	1
5	Guntur	2600	585000	0	1	0
6	Guntur	2800	615000	0	1	0
7	Guntur	3300	650000	0	1	0
8	Guntur	3600	710000	0	1	0
9	Gudiwada	2600	575000	1	0	0
10	Gudiwada	2900	600000	1	0	0
11	Gudiwada	3100	620000	1	0	0
12	Gudiwada	3600	695000	1	0	0

## Data Science – ML – Dummy Variable & OneHotEncoding

**Program Name** Creating dummy variables and preparing the dataframe demo4.py

```
import pandas as pd

df = pd.read_csv("homeprices2.csv")
dummies = pd.get_dummies(df.town)

merged = pd.concat([df, dummies], axis='columns')

final = merged.drop(['town'], axis='columns')

print(final)
```

**Output**

	area	price	Gudiwada	Guntur	Vijayawada
0	2600	550000	0	0	1
1	3000	565000	0	0	1
2	3200	610000	0	0	1
3	3600	680000	0	0	1
4	4000	725000	0	0	1
5	2600	585000	0	1	0
6	2800	615000	0	1	0
7	3300	650000	0	1	0
8	3600	710000	0	1	0
9	2600	575000	1	0	0
10	2900	600000	1	0	0
11	3100	620000	1	0	0
12	3600	695000	1	0	0

## Data Science – ML – Dummy Variable & OneHotEncoding

Program Name

Preparing X and y

demo5.py

```
import pandas as pd

df = pd.read_csv("homeprices2.csv")
dummies = pd.get_dummies(df.town)
merged = pd.concat([df, dummies], axis='columns')
final = merged.drop(['town'], axis='columns')

X = final.drop('price', axis='columns')
y = final.price

print(X)
print(y)
```

## Data Science – ML – Dummy Variable & OneHotEncoding

### Output

```
area  Gudiwada  Guntur  Vijayawada
0    2600      0       0       1
1    3000      0       0       1
2    3200      0       0       1
3    3600      0       0       1
4    4000      0       0       1
5    2600      0       1       0
6    2800      0       1       0
7    3300      0       1       0
8    3600      0       1       0
9    2600      1       0       0
10   2900      1       0       0
11   3100      1       0       0
12   3600      1       0       0
0    550000
1    565000
2    610000
3    680000
4    725000
5    585000
6    615000
7    650000
8    710000
9    575000
10   600000
11   620000
12   695000
Name: price, dtype: int64
```

## Data Science – ML – Dummy Variable & OneHotEncoding

**Program Name** Creating a model  
demo6.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

df = pd.read_csv("homeprices2.csv")
dummies = pd.get_dummies(df.town)
merged = pd.concat([df, dummies], axis='columns')
final = merged.drop(['town'], axis='columns')

X = final.drop('price', axis='columns')
y = final.price

model = LinearRegression()
model.fit(X, y)
print("Model got trained")
```

**Output**

Model got trained

## Data Science – ML – Dummy Variable & OneHotEncoding

**Program Name** Predicting the house prices  
demo7.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

df=pd.read_csv("homeprices2.csv")
dummies = pd.get_dummies(df.town)
merged = pd.concat([df, dummies],axis='columns')
final = merged.drop(['town'], axis='columns')

X = final.drop('price', axis='columns')
y = final.price

model = LinearRegression()
model.fit(X, y)
print(model.predict(X))
```

**Output**

```
[539709.73984091 590468.71640508 615848.20468716 666607.18125133
 717366.1578155 579723.71533005 605103.20361214 668551.92431735
 706621.15674047 565396.15136531 603465.38378844 628844.87207052
 692293.59277574]
```

## Data Science – ML – Dummy Variable & OneHotEncoding

**Program Name** Checking the score  
demo8.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

df=pd.read_csv("homeprices2.csv")
dummies = pd.get_dummies(df.town)
merged = pd.concat([df, dummies],axis='columns')
final = merged.drop(['town'], axis='columns')

X = final.drop('price', axis='columns')
y = final.price

model = LinearRegression()
model.fit(X, y)
print(model.score(X, y))
```

**Output**

0.9573929037221873

## Data Science – ML – Dummy Variable & OneHotEncoding

**Program Name** Predicting house price in Vijayawada  
demo9.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

df=pd.read_csv("homeprices2.csv")
dummies = pd.get_dummies(df.town)
merged = pd.concat([df, dummies],axis='columns')
final = merged.drop(['town'], axis='columns')

X = final.drop('price', axis='columns')
y = final.price

model = LinearRegression()
model.fit(X, y)
print(model.predict([[3400, 0, 0, 1]]))
```

**Output**

```
[641227.69296925]
```

## Data Science – ML – Dummy Variable & OneHotEncoding

**Program Name** Predicting house price in Guntur  
demo10.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

df=pd.read_csv("homeprices2.csv")
dummies = pd.get_dummies(df.town)
merged = pd.concat([df, dummies],axis='columns')
final = merged.drop(['town'], axis='columns')

X = final.drop('price', axis='columns')
y = final.price

model = LinearRegression()
model.fit(X, y)
print(model.predict([[3400, 0, 1, 0]]))
```

**Output**

```
[681241.66845839]
```

## Data Science – ML – Dummy Variable & OneHotEncoding

**Program Name** Predicting house price in Gudiwada  
demo11.py

```
import pandas as pd
from sklearn.linear_model import LinearRegression

df=pd.read_csv("homeprices2.csv")
dummies = pd.get_dummies(df.town)
merged = pd.concat([df, dummies],axis='columns')
final = merged.drop(['town'], axis='columns')

X = final.drop('price', axis='columns')
y = final.price

model = LinearRegression()
model.fit(X, y)
print(model.predict([[3400, 1, 0, 0]]))
```

**Output**

[666914.10449365]

**16. Data Science – Machine Learning – Gradient Descent****Contents**

<b>1. Gradient Descent.....</b>	<b>2</b>
<b>2. How Gradient Descent works .....</b>	<b>2</b>
<b>3. Convergence .....</b>	<b>3</b>
<b>4. Process behind the gradient .....</b>	<b>3</b>
<b>5. Steps .....</b>	<b>5</b>
<b>6. Follow below images.....</b>	<b>7</b>
<b>7. Fixed steps .....</b>	<b>11</b>
<b>8. Learning rate: Reaching minimum error .....</b>	<b>12</b>

## Data Science – Machine Learning – Gradient Descent

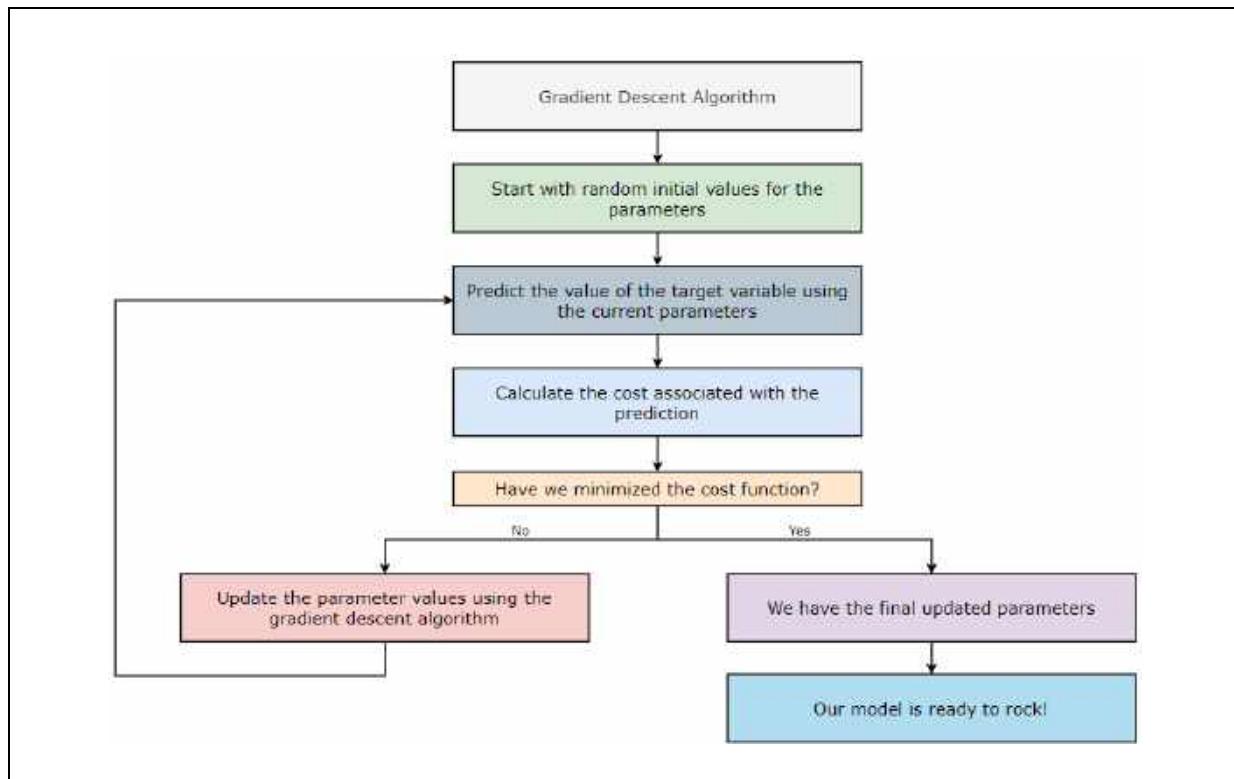
### 16. Data Science – Machine Learning – Gradient Descent

#### 1. Gradient Descent

- ✓ Gradient descent is an optimization algorithm
- ✓ This algorithm find the values of parameters (coefficients) of a function ( $f$ ) that minimizes a cost function (cost).

#### 2. How Gradient Descent works

- ✓ 1. Start with random initial values for the parameters.
- ✓ 2. Predict the value of the target variable using the current parameters.
- ✓ 3. Calculate the cost associated with the prediction.
- ✓ 4. Is cost minimized?
  - If yes, then go to step no - 6.
  - If no, then go to step no - 5.
- ✓ 5. Update the parameter values using the gradient descent algorithm and return to step no - 2
- ✓ 6. We have our final updated parameters.
- ✓ 7. Our model is ready.



## Data Science – Machine Learning – Gradient Descent

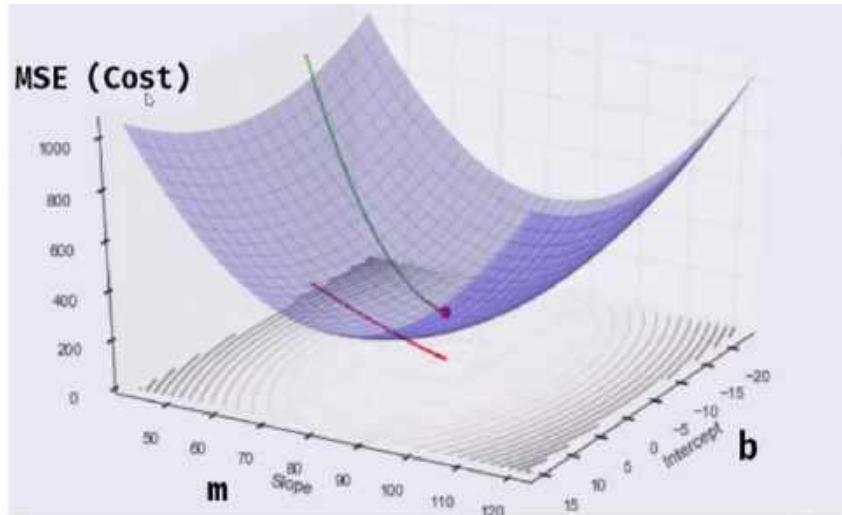
---

### 3. Convergence

- ✓ Reaching a point in which gradient descent makes very small changes in your objective function is called convergence.
- ✓ It doesn't mean that, it has reached the optimal result but it is somewhat near to that.

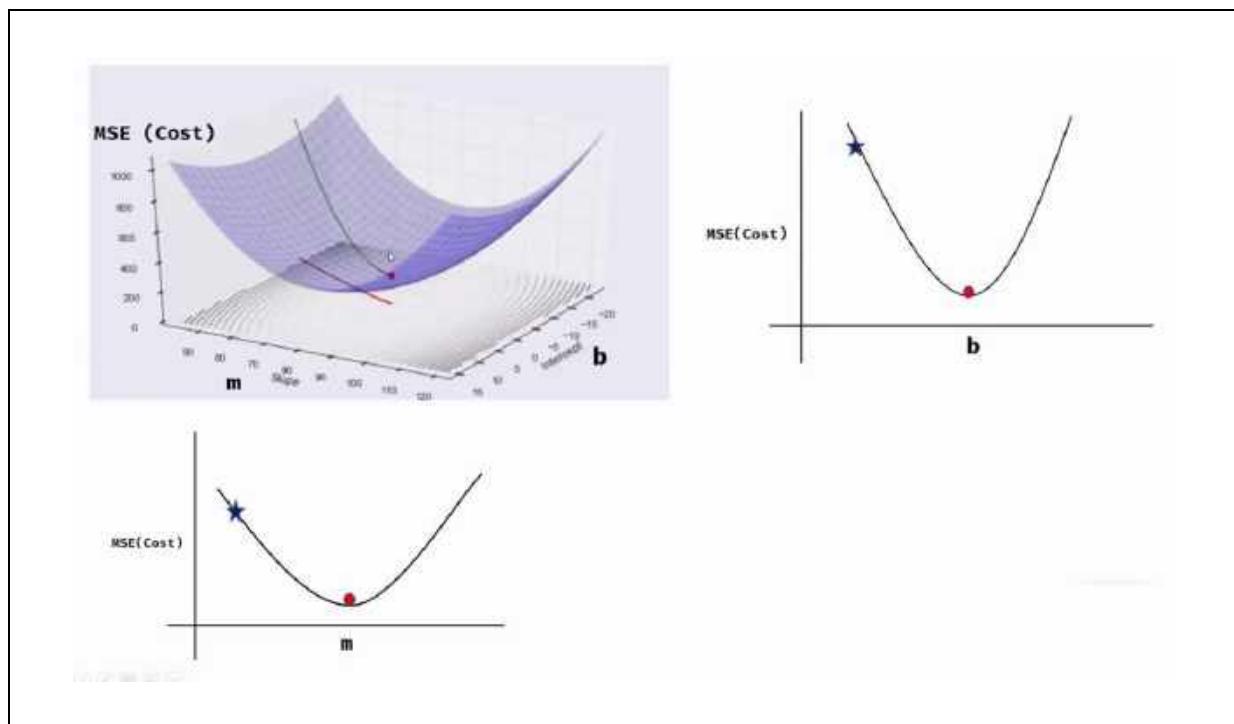
### 4. Process behind the gradient

- ✓ Gradient descent is the algorithm that finds best fit line for given training data set.
- ✓ If we calculate for every value of  $m$  and  $b$  we will get **MSE**
- ✓ If we plot  $m$  and  $b$  against MSE then we will get below image



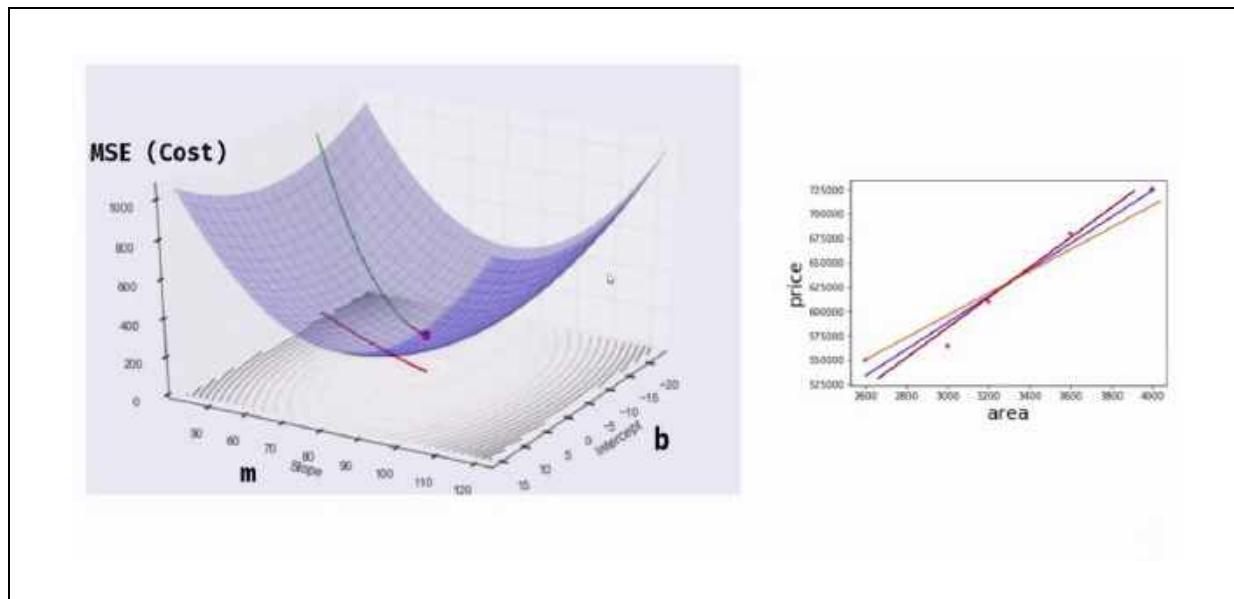
- ✓ Values
  - MSE values 0, 200, 400, 600, 800, 1000 etc
  - m values 50, 60, 70, 80, 90, 100, 110, 120 etc
  - b values 15, 10, 5, 0, -5, -10, -15 etc
- ✓ Here we will get an image which is like a bowl.
- ✓ We need to start some value for  $m$  and  $b$ , usually let's start from **zero** value

## Data Science – Machine Learning – Gradient Descent

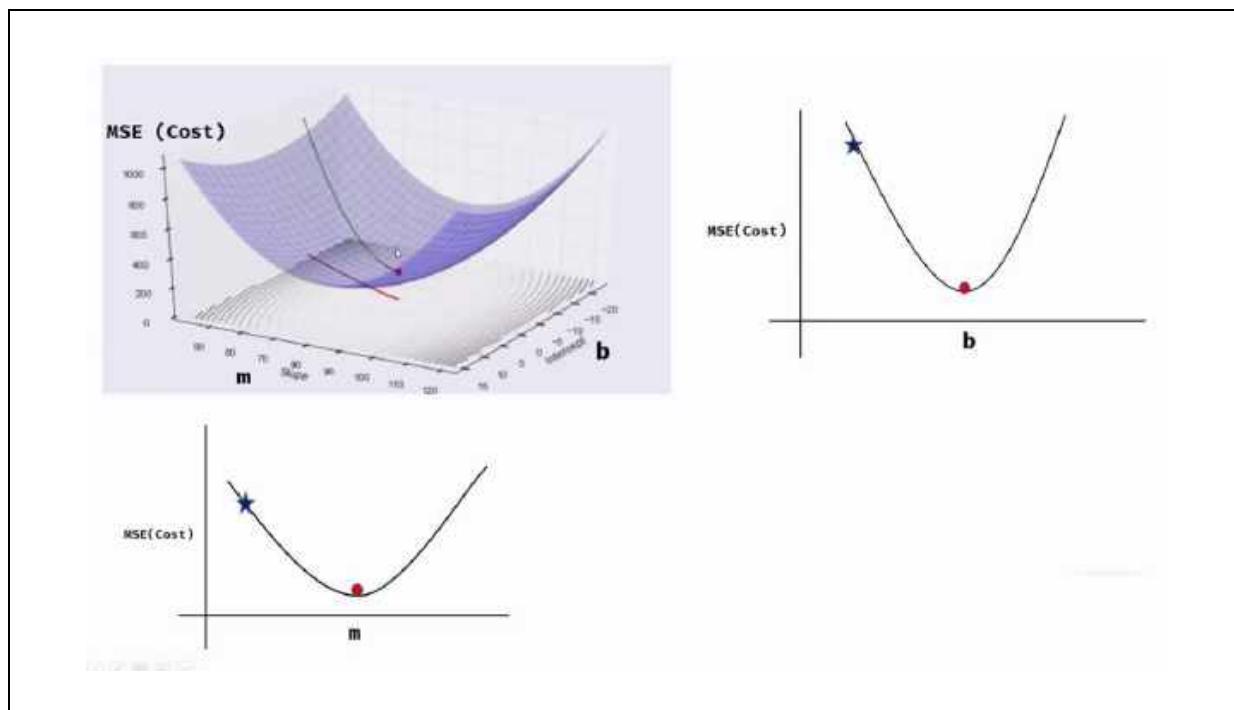


## 5. Steps

- ✓ Initially lets starts the point with  $m = 0$  and  $b = 0$
- ✓ Assuming that according to the diagram mse value is 1000 at  $m$ ,  $b$  equals to zero
- ✓ Now just tune the values of  $m$  and  $b$  with some amount
- ✓ Now check mse value and the error will get reduce
- ✓ So, we need to keep on doing this process until to reach error level to minima(minimum)
- ✓ Once it reach the minima then we got our answer
- ✓ Here we need to use that  $m$  and  $b$  values, in the prediction function

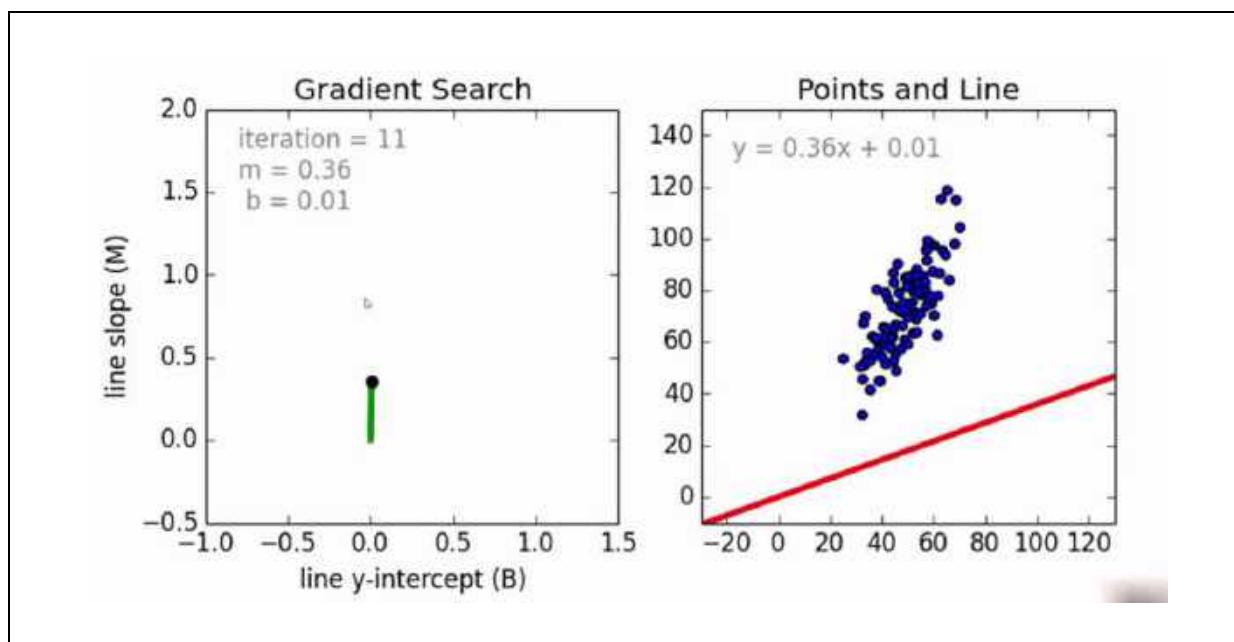
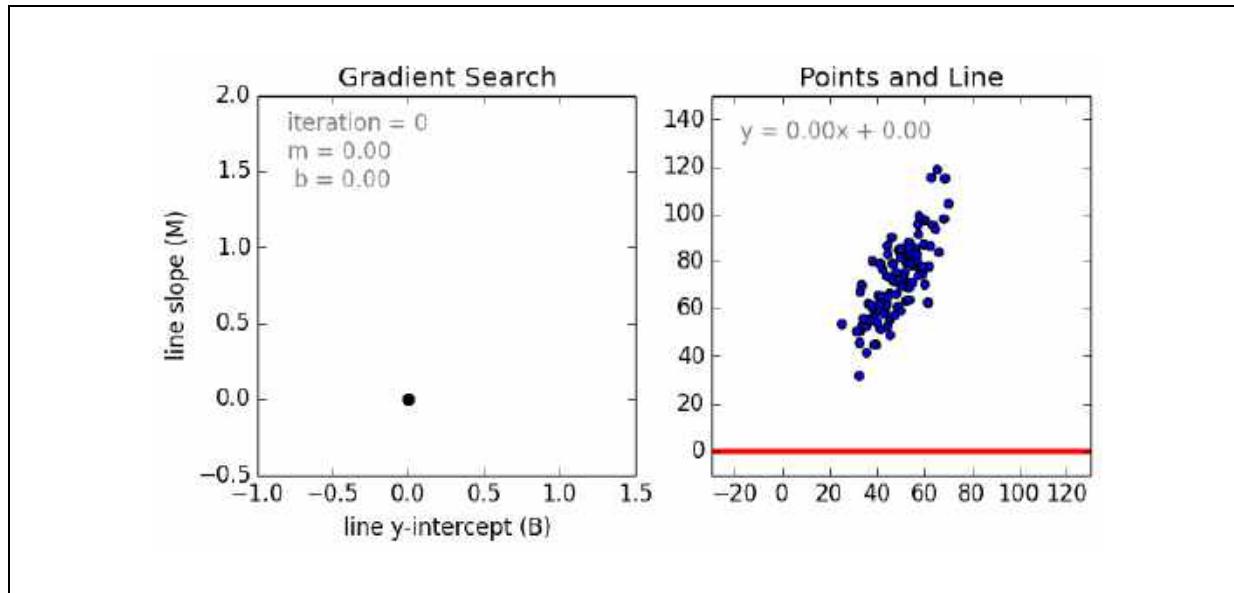


## Data Science – Machine Learning – Gradient Descent

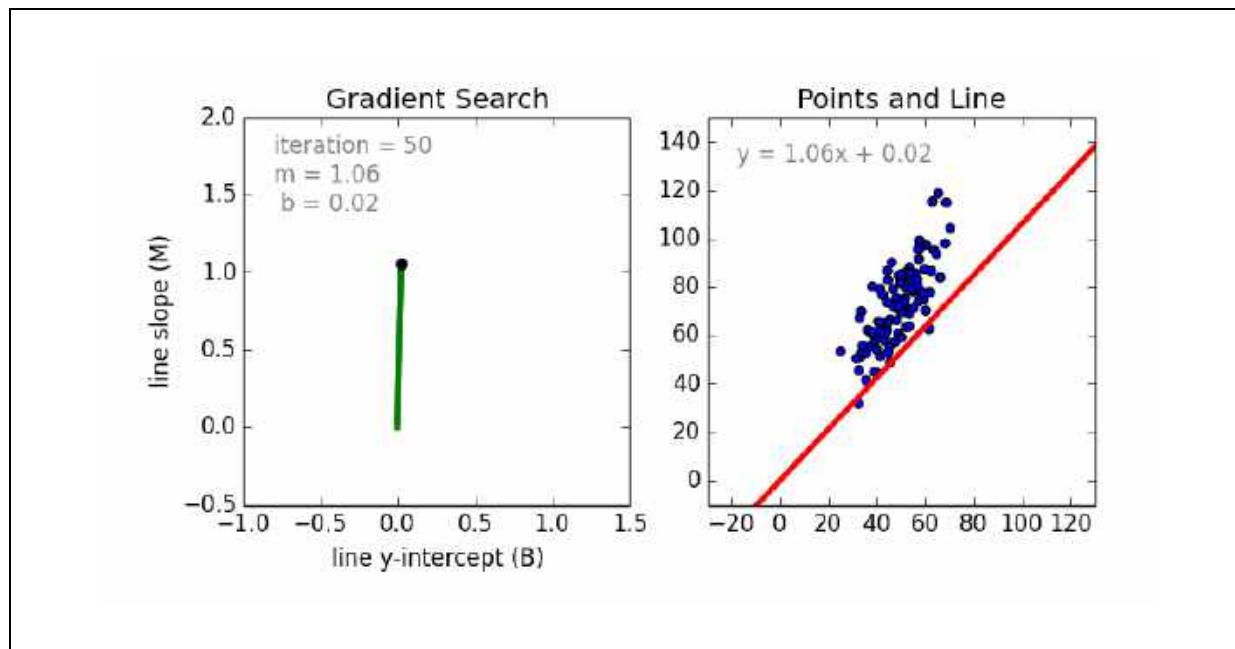
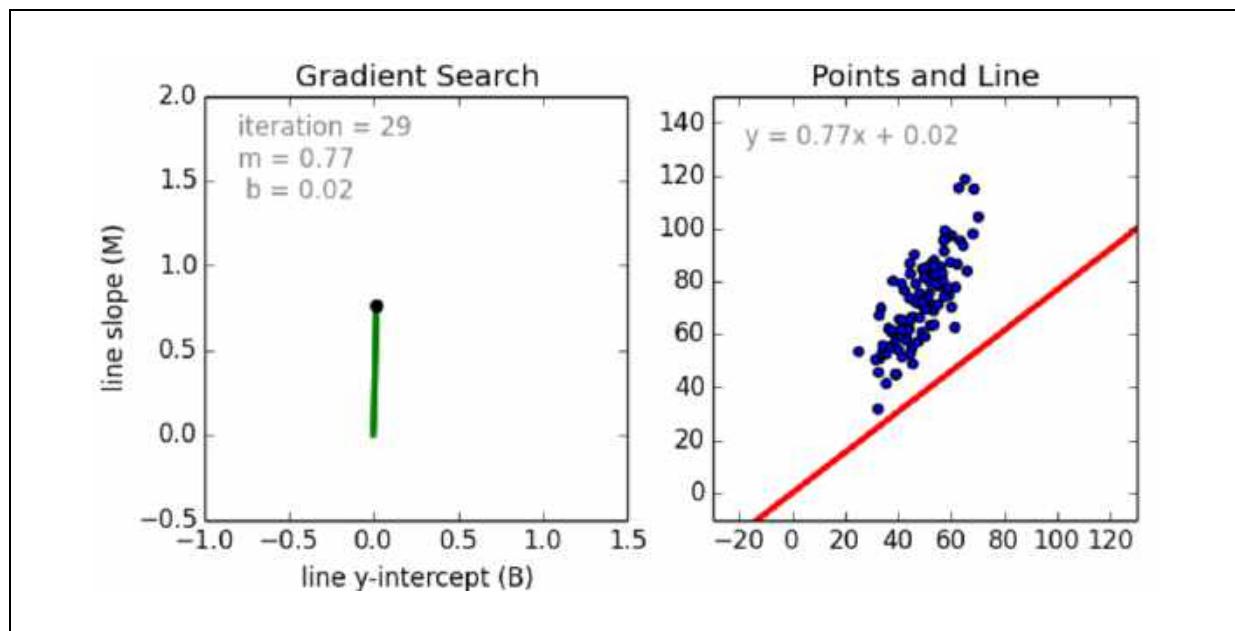


**6. Follow below images**

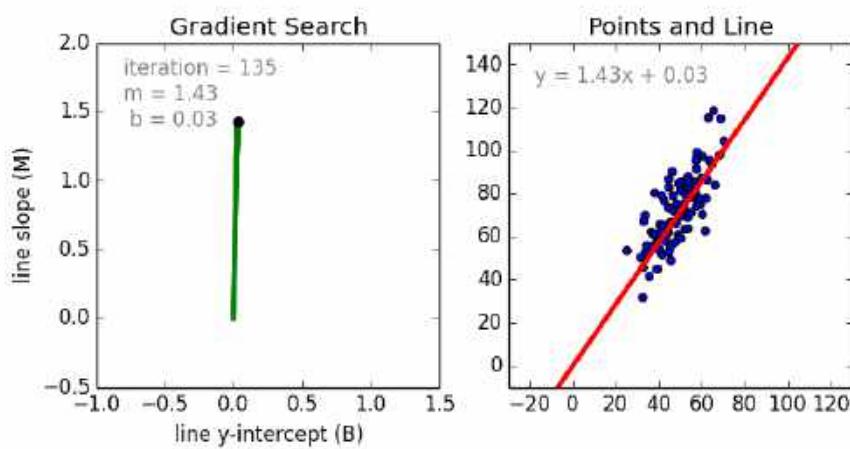
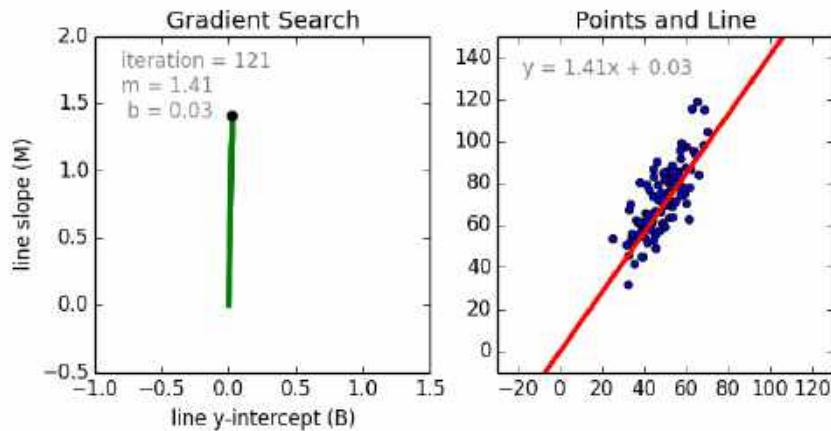
- ✓ Just try to understand below images for better understanding



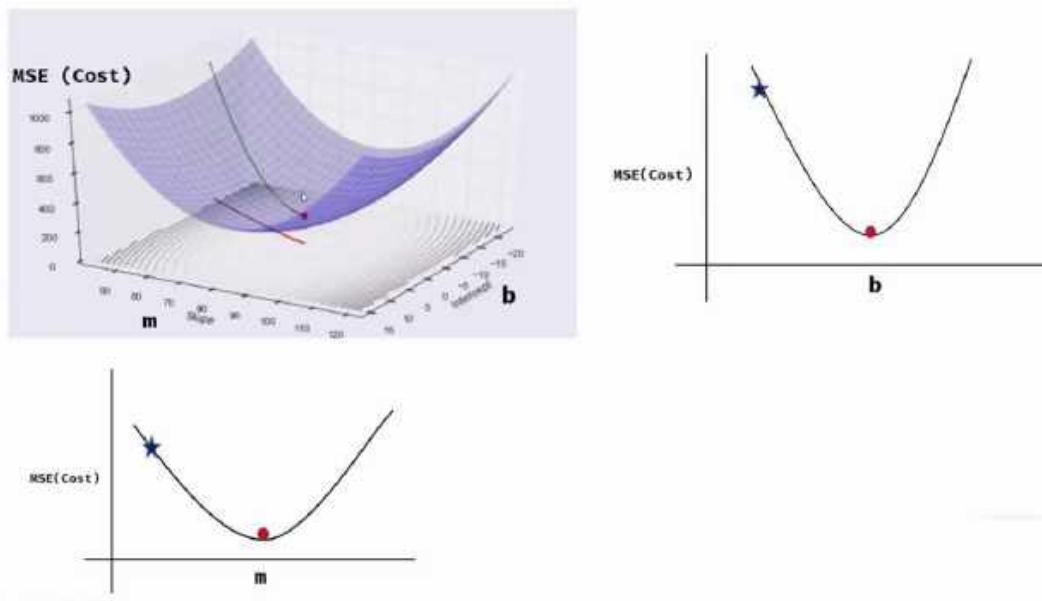
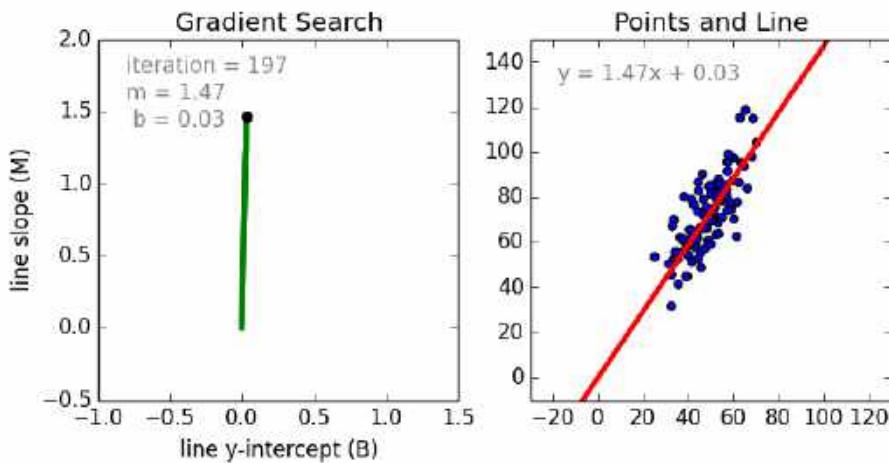
## Data Science – Machine Learning – Gradient Descent



## Data Science – Machine Learning – Gradient Descent

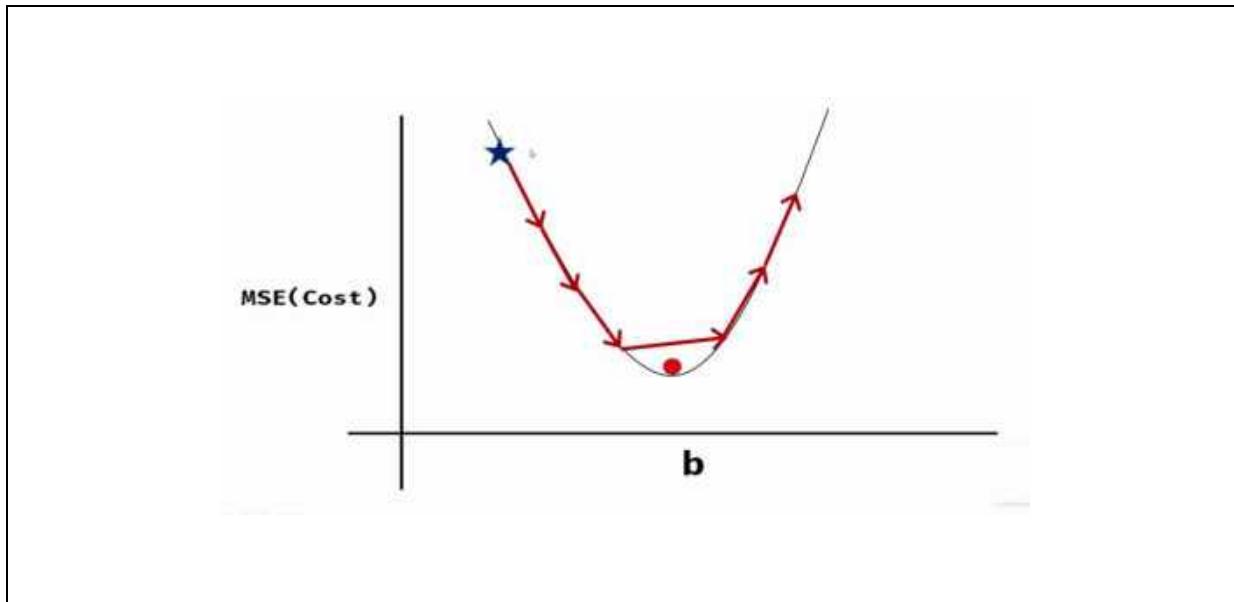


## Data Science – Machine Learning – Gradient Descent



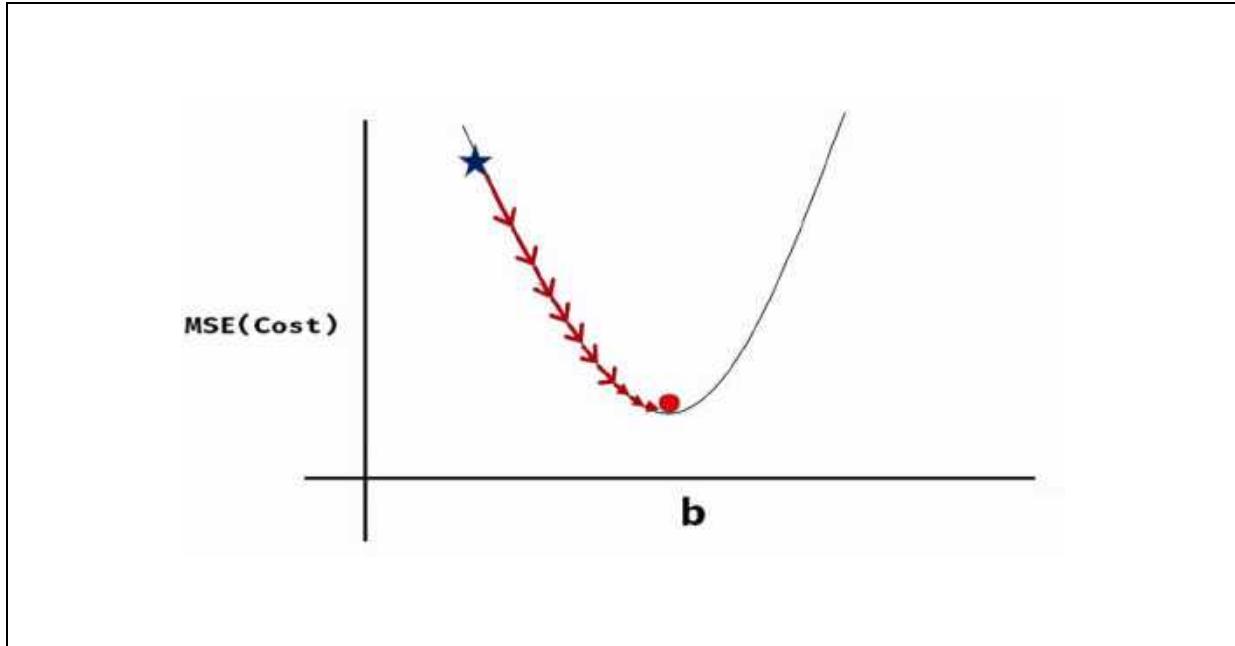
## 7. Fixed steps

- ✓ If we follow the fixed steps then we will get the below image
- ✓ By following this approach we may miss the global minima value.
- ✓ So, this approach is not going to work in well

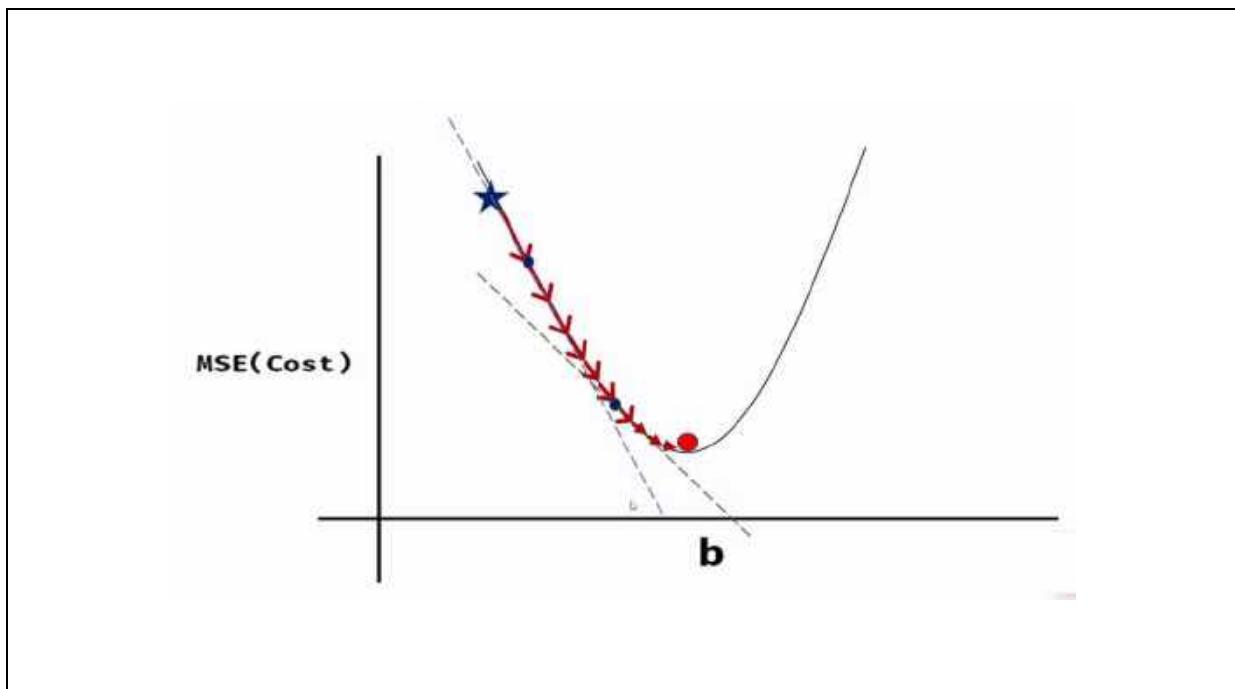


### 8. Learning rate: Reaching minimum error

- ✓ This approach seems to be good to catch minima value.
- ✓ The learning rate is a tuning parameter in an optimization algorithm.
- ✓ This determines the step size at each iteration while moving toward a minimum of a loss function.

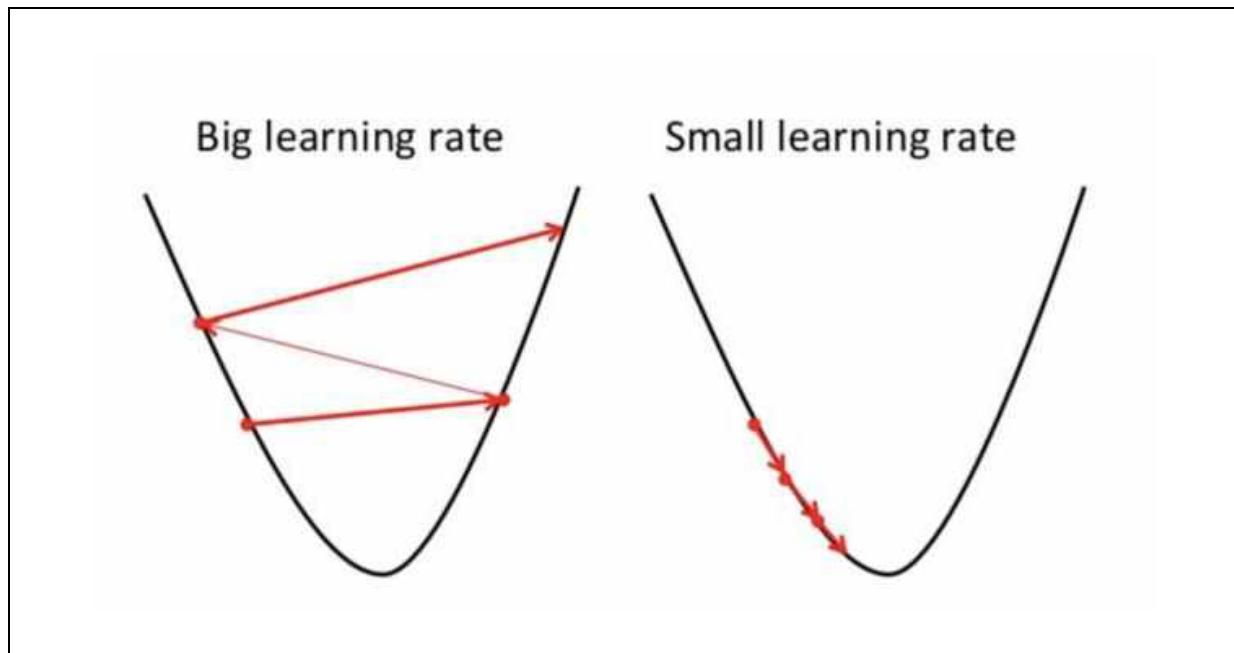


- ✓ At each point if we calculate the slope then we can reach the minimum value



### 9. Small and large learning rates

- ✓ A small leaning rate may lead to the model to take some time to learn
- ✓ A large learning rate will make the model converge as our pointer will shoot and we'll not be able to get to minima.



**18. Data Science – Machine Learning – Logistic Regression****Contents**

<b>1. Logistic Regression .....</b>	<b>2</b>
<b>2. Types of logistic regression .....</b>	<b>2</b>
<b>3. Binary classification.....</b>	<b>2</b>
<b>4. Multiclass classification .....</b>	<b>2</b>
<b>5. Data set.....</b>	<b>3</b>
<b>6. Problem statement .....</b>	<b>3</b>
<b>7. Logistic function or Sigmoid.....</b>	<b>4</b>

**18. Data Science – Machine Learning – Logistic Regression****1. Logistic Regression**

- ✓ Logistic regression comes under supervised Learning.
- ✓ It is a technique that is used to solve for classification problems.
- ✓ It is used for predicting the categorical dependent variable using a given set of independent variables.
- ✓ Examples
  - Email spam or not
  - Customer will buy product or not

**2. Types of logistic regression**

- ✓ Binary classification
  - This is having two classes
- ✓ Multiclass classification
  - This is having more than two classes

**3. Binary classification**

- ✓ In binary classification, there can be only two possible types of the dependent variables, such as,
  - 0 or 1
  - Pass or Fail
  - Yes or No etc.

**4. Multiclass classification**

- ✓ In multiclass classification, there can be 3 or more possible unordered types of the dependent variable, such as,
  - Ok, good, best
  - Cat, dot, sheep etc

## Data Science – Machine Learning – Logistic Regression

---

### 5. Data set

- ✓ Its insurance dataset
  - ZERO means didn't buy the insurance
  - ONE means will buy the insurance
- ✓ We can understand one pattern here like, young people not buying the insurance
- ✓ Whereas person age increasing then that person more likely to buy the insurance

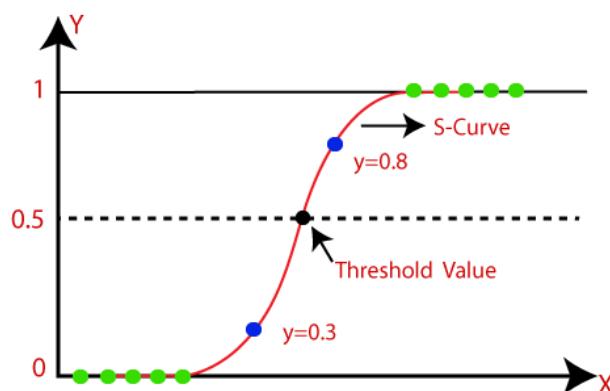
### 6. Problem statement

- ✓ Based on the age, we wanted to predict for persons will chose insurance or not.

age	Insurance status
22	0
25	0
47	1
52	0
46	1
56	1
55	0
60	1
62	1
61	1

## 7. Logistic function or Sigmoid

- ✓ The logistic function, also called the sigmoid function.
- ✓ It maps any real value into another value within a range of 0 and 1.
- ✓ The value of the logistic regression must be between 0 and 1.
- ✓ In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1.



### Formula

$$\text{sigmoid}(z) = \frac{1}{1 + e^{-z}}$$

$e$  = Euler's number  $\sim 2.71828$

Sigmoid function converts input into range 0 to 1

## Data Science – Machine Learning – Logistic Regression

**Program Name** Loading insurance dataset  
demo1.py

```
import pandas as pd

# Loading the dataset
df = pd.read_csv("insurance_data.csv")

print(df.head(10))
```

**Output**

	age	bought_insurance
0	22	0
1	25	0
2	47	1
3	52	0
4	46	1
5	56	1
6	55	0
7	60	1
8	62	1
9	61	1

## Data Science – Machine Learning – Logistic Regression

**Program Name** Plotting the dataset  
demo2.py

```
import pandas as pd
from matplotlib import pyplot as plt

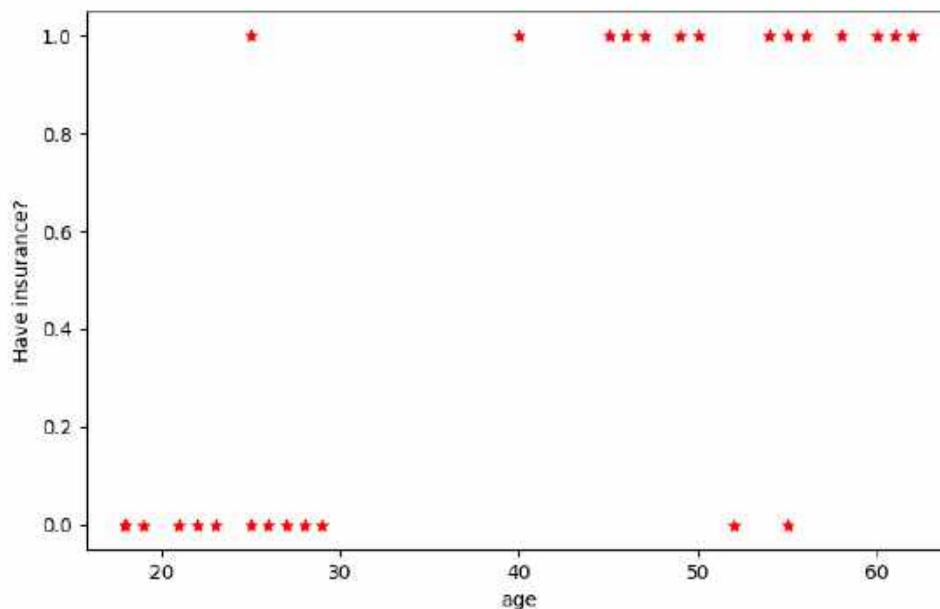
# Loading the dataset
df = pd.read_csv("insurance_data.csv")

# plotting the data
plt.scatter(df.age, df.bought_insurance, marker = '*', color = 'red')

plt.xlabel('age')
plt.ylabel('Have insurance?')

plt.show()
```

### Output



## Data Science – Machine Learning – Logistic Regression

**Program Name** Splitting the dataset  
demo3.py

```
import pandas as pd
from sklearn.model_selection import train_test_split

# Loading the dataset
df = pd.read_csv("insurance_data.csv")
X = df[['age']]
y = df.bought_insurance

# Splitting the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_state=52)

print("X_train", '\n')
print(X_train, '\n')
```

## Data Science – Machine Learning – Logistic Regression

### Output

```
X_train  
  
      age  
14    49  
4     46  
12    27  
2     47  
8     62  
6     55  
19   18  
26   23  
24   50  
20   21  
15   55  
16   25  
1    25  
17   58  
3    52  
25   54  
10   18  
5    56  
0    22  
22   40  
23   45  
13   29  
11   28  
21   26  
  
X_test  
  
      age  
7    60  
9    61  
18   19
```

## Data Science – Machine Learning – Logistic Regression

**Program Name** Splitting the dataset  
demo4.py

```
import pandas as pd
from sklearn.model_selection import train_test_split

# Loading the dataset
df = pd.read_csv("insurance_data.csv")

X = df[['age']]
y = df.bought_insurance

# Splitting the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_state = 52)

print("y_train", '\n')
print(y_train,'n')
```

**Output**

```
y_train  
14      1  
4       1  
12      0  
2       1  
8       1  
6       0  
19      0  
26      0  
24      1  
20      0  
15      1  
16      1  
1       0  
17      1  
3       0  
25      1  
10      0  
5       1  
0       0  
22      1  
23      1  
13      0  
11      0  
21      0  
Name: bought_insurance, dtype: int64  
  
y_test  
7       1  
9       1  
18      0  
Name: bought_insurance, dtype: int64
```

## Data Science – Machine Learning – Logistic Regression

**Program Name** Training the model  
demo5.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

# Loading the dataset
df = pd.read_csv("insurance_data.csv")

X = df[['age']]
y = df.bought_insurance

# Splitting the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_state = 52)

# Creating and training the model
model = LogisticRegression()
model.fit(X_train, y_train)

print("Model got trained")
```

**Output**

Model got trained

## Data Science – Machine Learning – Logistic Regression

### Program

Name demo6.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

# Loading the dataset
df = pd.read_csv("insurance_data.csv")

X = df[['age']]
y = df.bought_insurance

# Splitting the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.1, random_state = 52)

# Creating and training the model
model = LogisticRegression()
model.fit(X_train, y_train)

# Prediction
print(model.predict([[50]]))
print(model.predict([[25]]))
```

### Output

```
[1]
[0]
```

## Data Science – Machine Learning – Logistic Regression

**Program Name** Prediction the result  
demo7.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

# Loading the dataset
df = pd.read_csv("insurance_data.csv")

X = df[['age']]
y = df.bought_insurance

# Splitting the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_state = 52)

# Creating and training the model
model = LogisticRegression()
model.fit(X_train, y_train)

# Prediction
y_predicted = model.predict(X_test)

print("X_test data is \n")
print(X_test, '\n')

print("Prediction for X_test data \n")
print(y_predicted)
```

### Output

```
X_test data is

      age
7      60
9      61
18     19

Prediction for X_test data

[1 1 0]
```

### Prediction

- ✓ The one who has 19 years age he will not buy the insurance
- ✓ Both who has 60 and 61 years age persons will buy the insurance

## Data Science – Machine Learning – Logistic Regression

Program

Name Prediction score

demo8.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

# Loading the dataset
df=pd.read_csv("insurance_data.csv")

X = df[['age']]
y = df.bought_insurance

# Splitting the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.1, random_state = 52)

# Creating and training the model
model = LogisticRegression()
model.fit(X_train, y_train)

# Prediction score
print(model.score(X_test, y_test))
```

Output

1.0

## Data Science – Machine Learning – Logistic Regression

**Program Name** Prediction probability  
demo9.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

# Loading the dataset
df = pd.read_csv("insurance_data.csv")

X = df[['age']]
y = df.bought_insurance

# Splitting the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_state = 52)

# Creating and training the model
model = LogisticRegression()
model.fit(X_train, y_train)

print("X_test")
print(X_test)

print()

# Prediction probability
print(model.predict_proba(X_test))
```

**Output**

```
X_test
    age
7      60
9      61
18     19

[[0.06266647  0.93733353]
 [0.05553734  0.94446266]
 [0.92804604  0.07195396]]
```

**19. Data Science – ML – Logistic Regression – Multi class classification****Contents**

<b>1. Logistic Regression .....</b>	<b>2</b>
<b>2. Types of logistic regression .....</b>	<b>2</b>
<b>3. Binary classification examples .....</b>	<b>2</b>
<b>4. Multiclass classification examples.....</b>	<b>2</b>
<b>5. Practical example .....</b>	<b>3</b>
<b>6. Problem .....</b>	<b>4</b>
<b>7. Load dataset .....</b>	<b>5</b>

## Data Science – ML– Logistic Regression – Multiclass classification

---

### 19. Data Science – ML – Logistic Regression – Multi class classification

#### 1. Logistic Regression

- ✓ Logistic regression comes under supervised Learning.
- ✓ It is a technique that is used to solve for classification problems.
- ✓ It is used for predicting the categorical dependent variable using a given set of independent variables.

#### 2. Types of logistic regression

- ✓ Binary classification
  - This is having two classes
- ✓ Multiclass classification
  - This is having more than two classes

#### 3. Binary classification examples

- ✓ In binary classification, there can be only two possible types of the dependent variables, such as,
  - 0 or 1
  - Pass or Fail
  - Yes or No etc.

#### 4. Multiclass classification examples

- ✓ In multiclass classification, there can be 3 or more possible unordered types of the dependent variable, such as,
  - Ok, good, best
  - Cat, dot, sheep etc

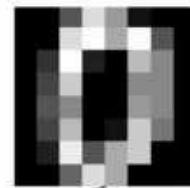
**5. Practical example**

- ✓ Identify hand written digits
  - If image having ZERO number then output should be Zero.
  - If image having ONE number then output should be ONE.
  - If image having TWO number then output should be TWO.
  - ....
  - If image having NINE number then output should be NINE.

## 6. Problem

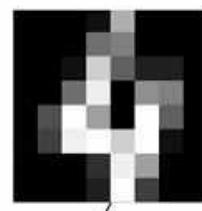
- ✓ Recognising the number.

Identify hand written digits recognition



0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---

Identify hand written digits recognition



0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---

## Data Science – ML– Logistic Regression – Multiclass classification

### 7. Load dataset

- ✓ Sklearn library comes up with built-in datasets.
- ✓ One of the dataset names is called as digits dataset.

**Program Name** Loading digits dataset  
demo1.py

```
from sklearn.datasets import load_digits

# Loading the dataset
digits = load_digits()

print(len(digits.data))
```

**Output**  
1797

**Program Name** Loading digits dataset and checking attributes  
demo2.py

```
from sklearn.datasets import load_digits

# Loading the dataset
digits = load_digits()

print(dir(digits))
```

**Output**  
['DESCR', 'data', 'feature\_names', 'frame', 'images', 'target',  
'target\_names']

## Data Science – ML– Logistic Regression – Multiclass classification

**Program Name** Loading digits dataset and accessing DESCR attribute  
demo3.py

```
from sklearn.datasets import load_digits

# Loading the dataset
digits = load_digits()

print(digits.DESCR)
```

### Output

```
C:\Users\Nireekshan\Desktop\PROGRAMS>py test.py
.. _digits_dataset:

Optical recognition of handwritten digits dataset
-----
**Data Set Characteristics:**

:Number of Instances: 1797
:Number of Attributes: 64
:Attribute Information: 8x8 image of integer pixels in the range 0..16.
:Missing Attribute Values: None
:Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)
:Date: July; 1998

This is a copy of the test set of the UCI ML hand-written digits datasets
https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits

The data set contains images of hand-written digits: 10 classes where
each class refers to a digit.
```

## Data Science – ML– Logistic Regression – Multiclass classification

**Program Name** Loading digits dataset and accessing data attribute  
demo4.py

```
from sklearn.datasets import load_digits

# Loading the dataset
digits = load_digits()

print(digits.data)
```

**Output**

```
[[ 0.  0.  5. ...  0.  0.  0.]
 [ 0.  0.  0. ... 10.  0.  0.]
 [ 0.  0.  0. ... 16.  9.  0.]
 ...
 [ 0.  0.  1. ...  6.  0.  0.]
 [ 0.  0.  2. ... 12.  0.  0.]
 [ 0.  0. 10. ... 12.  1.  0.]]
```

## Data Science – ML– Logistic Regression – Multiclass classification

---

**Program Name** Loading digits dataset, checking the first value  
demo5.py

```
from sklearn.datasets import load_digits

# Loading the dataset
digits = load_digits()

print(digits.data[0])
```

**Output**

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

**Program Name** Loading digits dataset, checking the second value  
demo6.py

```
from sklearn.datasets import load_digits

# Loading the dataset
digits = load_digits()

print(digits.data[1])
```

**Output**

```
[ 0.  0.  0. 12. 13.  5.  0.  0.  0.  0. 11. 16.  9.  0.  0.  0.  0.
 3. 15. 16.  6.  0.  0.  0.  7. 15. 16. 16.  2.  0.  0.  0.  0.  1. 16.
16.  3.  0.  0.  0.  1. 16. 16.  6.  0.  0.  0.  0.  1. 16. 16.  6.
 0.  0.  0.  0. 11. 16. 10.  0.  0.]
```

## Data Science – ML– Logistic Regression – Multiclass classification

**Program Name** Loading digits dataset and accessing images attribute  
demo7.py

```
from sklearn.datasets import load_digits

# Loading the dataset
digits = load_digits()

print(digits.images)
```

**Output**

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

[[ 0.  0.  0. ... 5.  0.  0.]
 [ 0.  0.  0. ... 9.  0.  0.]
 [ 0.  0.  3. ... 6.  0.  0.]
 ...
 [ 0.  0.  1. ... 6.  0.  0.]
 [ 0.  0.  1. ... 6.  0.  0.]
 [ 0.  0.  0. ... 10. 0.  0.]]]

[[ 0.  0.  0. ... 12. 0.  0.]
 [ 0.  0.  3. ... 14. 0.  0.]
 [ 0.  0.  8. ... 16. 0.  0.]
 ...]
```

## Data Science – ML– Logistic Regression – Multiclass classification

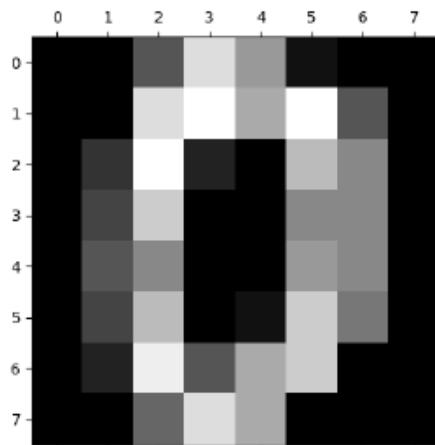
**Program Name** Loading digits dataset, displaying image  
demo8.py

```
from sklearn.datasets import load_digits
import matplotlib.pyplot as plt

# Loading the dataset
digits = load_digits()

plt.matshow(digits.images[0])
plt.gray()
plt.show()
```

**Output**



## Data Science – ML– Logistic Regression – Multiclass classification

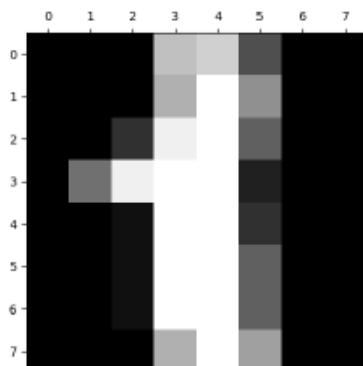
**Program Name** Loading digits dataset, displaying image  
demo9.py

```
from sklearn.datasets import load_digits
import matplotlib.pyplot as plt

# Loading the dataset
digits = load_digits()

plt.matshow(digits.images[1])
plt.gray()
plt.show()
```

**Output**



## Data Science – ML – Logistic Regression – Multiclass classification

**Program Name** Loading digits dataset, displaying image  
demo10.py

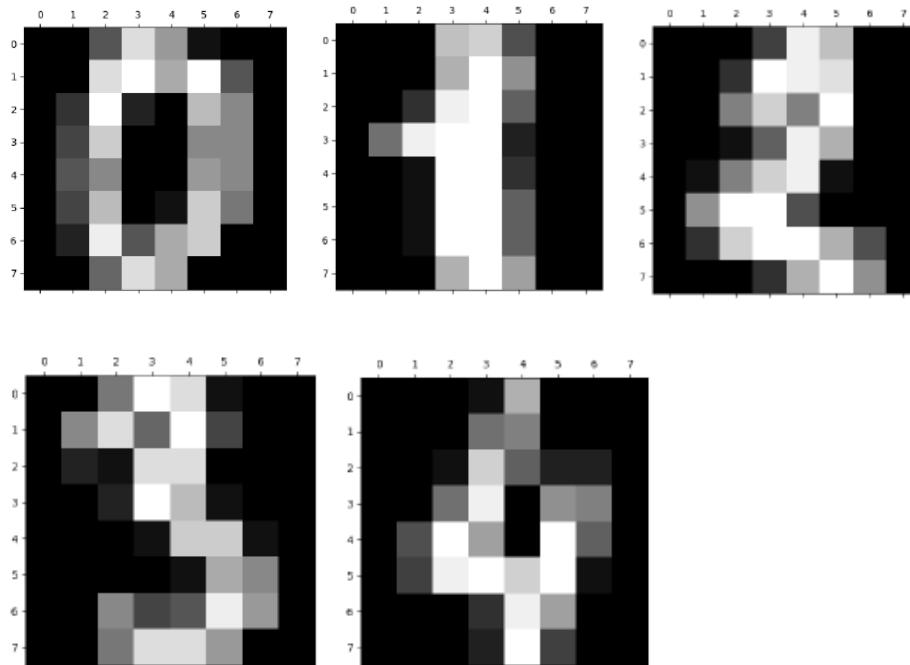
```
from sklearn.datasets import load_digits
import matplotlib.pyplot as plt

# Loading the dataset
digits = load_digits()

plt.gray()

for i in range(5):
    plt.matshow(digits.images[i])
    plt.show()
```

**Output**



## Data Science – ML– Logistic Regression – Multiclass classification

**Program Name** Loading digits dataset and accessing target attribute  
demo11.py

```
from sklearn.datasets import load_digits

# Loading the dataset
digits = load_digits()

print(digits.target)
```

**Output**

```
[0 1 2 ... 8 9 8]
```

**Program Name** Loading digits dataset and accessing target attribute  
demo12.py

```
from sklearn.datasets import load_digits

# Loading the dataset
digits = load_digits()

print(digits.target[0:5])
```

**Output**

```
[0 1 2 3 4]
```

## Data Science – ML– Logistic Regression – Multiclass classification

**Program Name** Splitting the dataset  
demo13.py

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split

# Loading the dataset
digits = load_digits()

print("Splitting the dataset")
X_train, X_test, y_train, y_test = train_test_split(digits.data,
digits.target, test_size = 0.2)
```

**Output**

```
Splitting the dataset
```

## Data Science – ML– Logistic Regression – Multiclass classification

**Program Name** Model creation  
demo14.py

```
from sklearn.datasets import load_digits
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Loading the dataset
digits = load_digits()

X_train, X_test, y_train, y_test = train_test_split(digits.data,
digits.target, test_size=0.2)

print("Model creation")
model = LogisticRegression(solver = 'lbfgs', max_iter = 3000)

model.fit(X_train, y_train)
```

### Output

Model creation

### Note

- ✓ lbfgs stand for: "Limited-memory Broyden–Fletcher–Goldfarb–Shanno Algorithm".
- ✓ It is one of the solvers' algorithms provided by Scikit-Learn Library.

## Data Science – ML– Logistic Regression – Multiclass classification

Program Name

Model score  
demo15.py

```
from sklearn.datasets import load_digits
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Loading the dataset
digits = load_digits()

X_train, X_test, y_train, y_test = train_test_split(digits.data,
digits.target, test_size=0.2)

model = LogisticRegression(solver = 'lbfgs', max_iter = 3000)

model.fit(X_train, y_train)
print(model.score(X_test, y_test))
```

Output

0.9611111111111111

## Data Science – ML– Logistic Regression – Multiclass classification

**Program Name**

Model prediction

demo16.py

```
from sklearn.datasets import load_digits
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Loading the dataset
digits = load_digits()

X_train, X_test, y_train, y_test = train_test_split(digits.data,
digits.target, test_size = 0.2)

model = LogisticRegression(solver = 'lbfgs', max_iter = 3000)

model.fit(X_train, y_train)

print(model.predict([digits.data[6]]))
```

**Output**

[6]

## Data Science – ML– Logistic Regression – Multiclass classification

**Program**

**Name** demo17.py

```
from sklearn.datasets import load_digits
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Loading the dataset
digits = load_digits()

X_train, X_test, y_train, y_test = train_test_split(digits.data,
digits.target, test_size=0.2)

model = LogisticRegression(solver = 'lbfgs', max_iter = 3000)

model.fit(X_train, y_train)

print(model.predict([digits.data[9]]))
```

**Output**

[9]

## Data Science – ML– Logistic Regression – Multiclass classification

Program

Name demo18.py

```
from sklearn.datasets import load_digits
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Loading the dataset
digits = load_digits()

X_train, X_test, y_train, y_test = train_test_split(digits.data,
digits.target, test_size=0.2)

model = LogisticRegression(solver = 'lbfgs', max_iter = 3000)

model.fit(X_train, y_train)

print(model.predict([digits.data[20]]))
```

Output

[0]

## Data Science – ML– Logistic Regression – Multiclass classification

Program

Name demo19.py

```
from sklearn.datasets import load_digits
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Loading the dataset
digits = load_digits()

X_train, X_test, y_train, y_test = train_test_split(digits.data,
digits.target, test_size=0.2)

model = LogisticRegression(solver = 'lbfgs', max_iter = 3000)

model.fit(X_train, y_train)

print(model.predict(digits.data[0:5]))
```

Output

```
[0 1 2 3 4]
```

**20. Data Science – Machine Learning – Decision Tree****Contents**

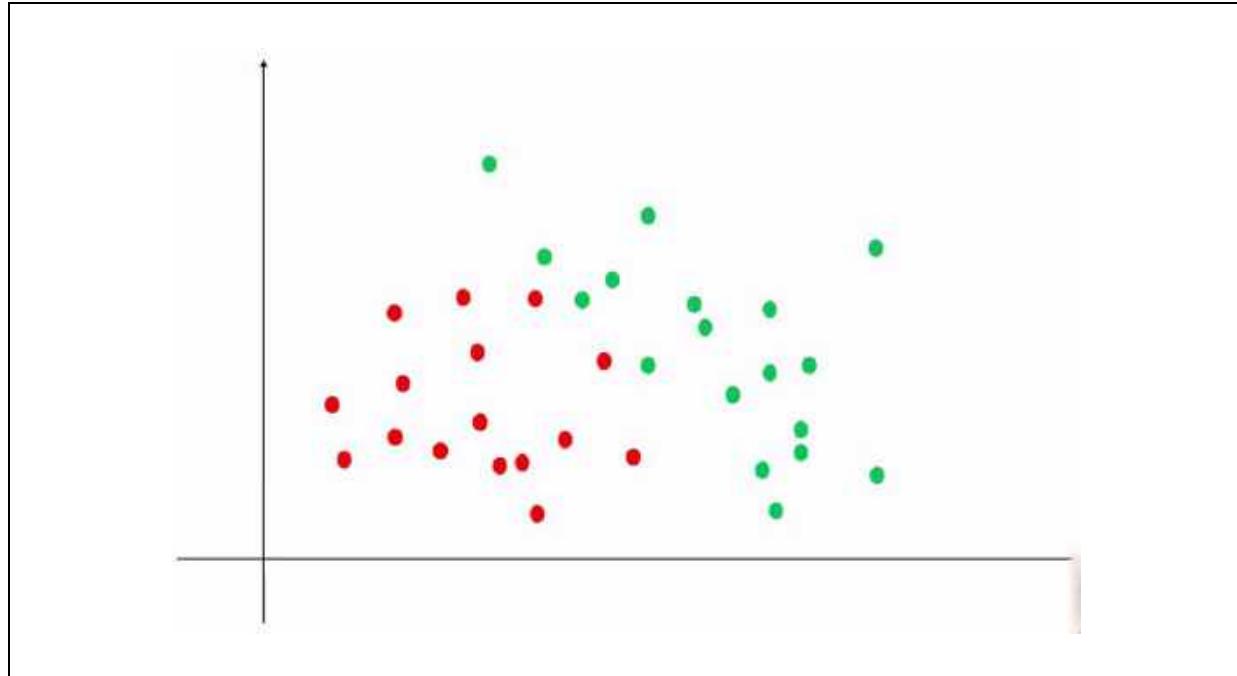
<b>1. If dataset is like below example1 .....</b>	<b>2</b>
<b>2. If dataset is as below example2.....</b>	<b>3</b>
<b>3. Decision Tree Classification Algorithm .....</b>	<b>4</b>
3.1. Decision Node .....	4
3.2. Leaf nodes.....	4
<b>4. CART algorithm.....</b>	<b>5</b>
<b>5. Why use Decision Trees?.....</b>	<b>5</b>
<b>6. Decision Tree Terminologies.....</b>	<b>6</b>
6.1. Root Node .....	6
6.2. Leaf Node .....	6
6.3. Splitting .....	6
6.4. Branch/Sub Tree .....	6
6.5. Pruning.....	6
6.6. Parent/Child node.....	6
<b>7. How does the Decision Tree algorithm Work?.....</b>	<b>7</b>
7.1. Example.....	8
<b>8. Problem .....</b>	<b>9</b>
<b>9. DecisionTreeClassifier class .....</b>	<b>17</b>
9.1. fit(X_train, y_train) method.....	18
9.2. predict(p) method.....	20

## Data Science – Machine Learning – Decision Tree

### 20. Data Science – Machine Learning – Decision Tree

#### 1. If dataset is like below example1

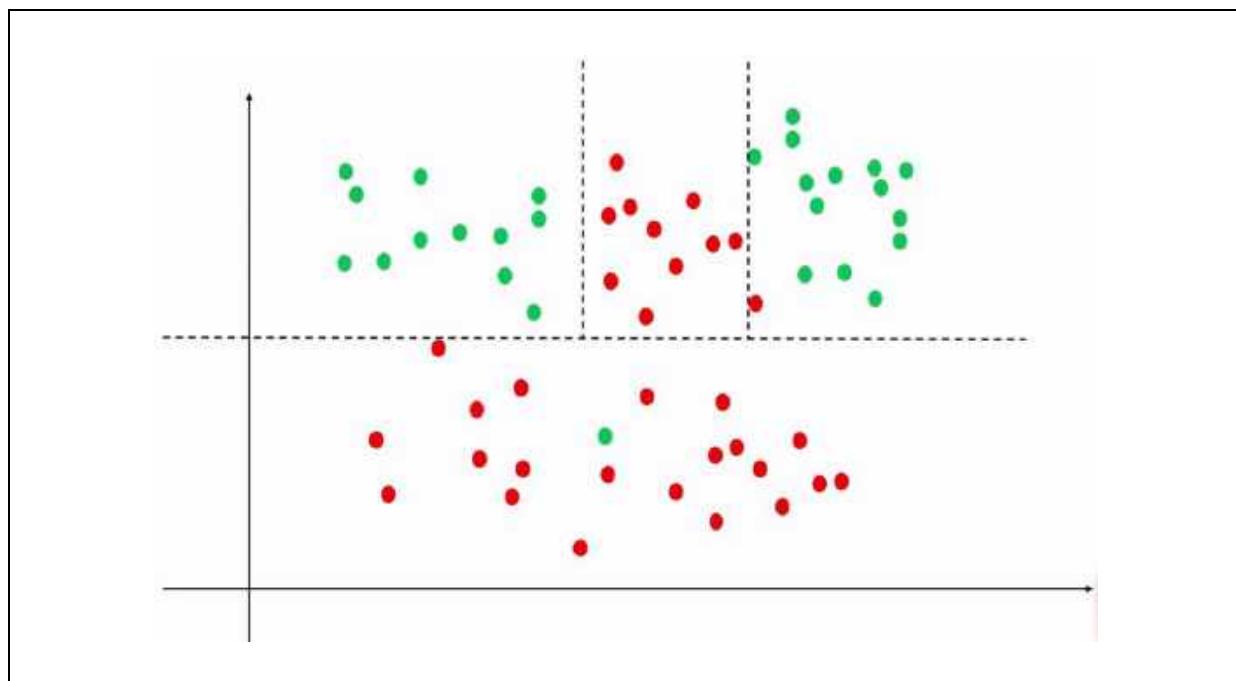
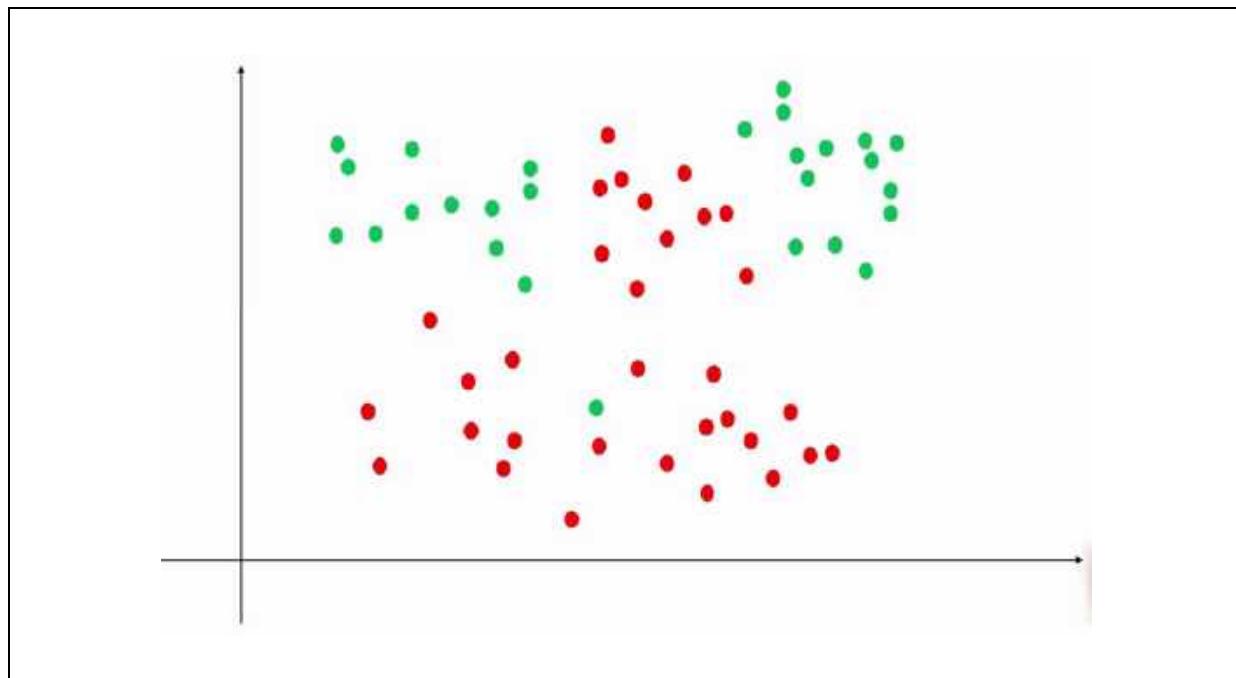
- ✓ We can draw a good separation line by using regression algorithm



## Data Science – Machine Learning – Decision Tree

### 2. If dataset is as below example2

- ✓ In the given below scenario, dataset is very complex.
- ✓ Then a single line may not fit for the given dataset
- ✓ Here decision tree comes into the picture



### 3. Decision Tree Classification Algorithm

- ✓ Decision Tree is a supervised learning technique.
- ✓ It can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems.
- ✓ It is called a decision tree because similar to a tree structure like it starts with the root node and expands on further branches and constructs a tree-like structure.
- ✓ In a Decision tree there are two nodes,
  - Decision Node
  - Leaf Node.

#### 3.1. Decision Node

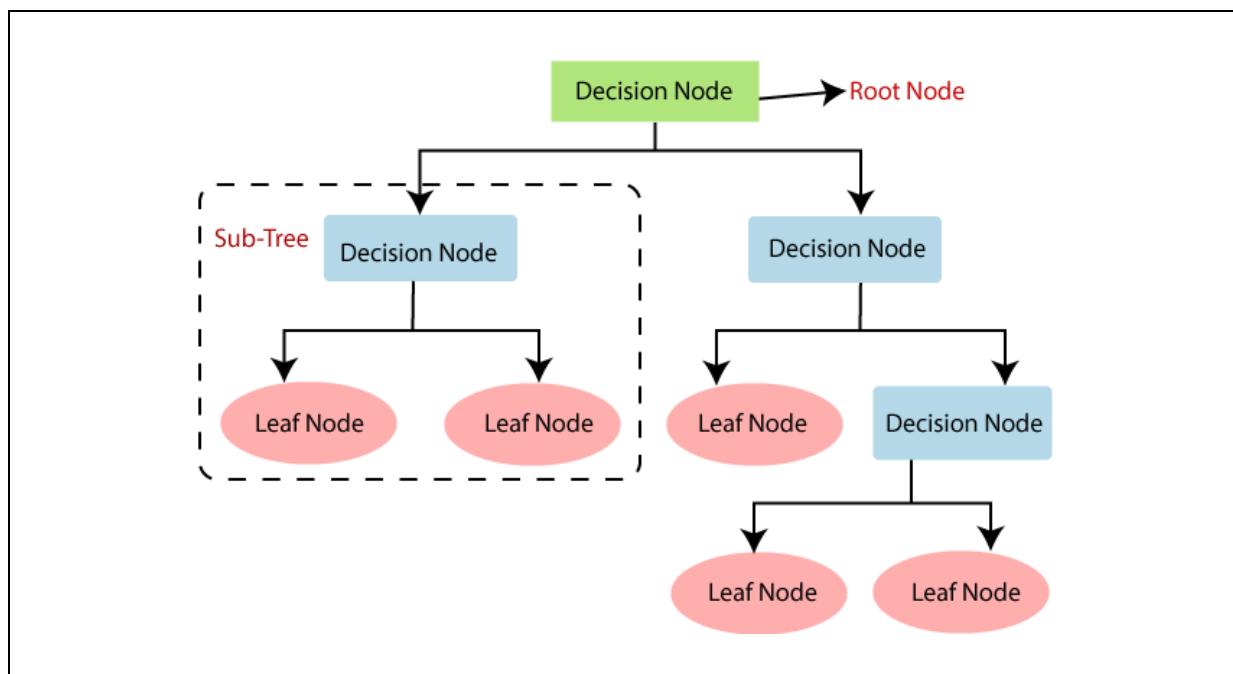
- ✓ Decision nodes are used to make any decision and have multiple branches.

#### 3.2. Leaf nodes

- ✓ Leaf nodes are the output of those decisions and do not contain any further branches.

#### 4. CART algorithm

- ✓ In order to build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm.
- ✓ A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.



#### 5. Why use Decision Trees?

- ✓ Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.
- ✓ The logic behind the decision tree can be easily understood because it shows a tree-like structure.

## 6. Decision Tree Terminologies

### 6.1. Root Node

- ✓ Root node is from where the decision tree starts.
- ✓ It represents the entire dataset, which further gets divided into two or more homogeneous sets.

### 6.2. Leaf Node

- ✓ Leaf nodes are the final output node.
- ✓ The tree cannot be segregated further after getting a leaf node.

### 6.3. Splitting

- ✓ Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.

### 6.4. Branch/Sub Tree

- ✓ A tree formed by splitting the tree.

### 6.5. Pruning

- ✓ Pruning is the process of removing the unwanted branches from the tree.

### 6.6. Parent/Child node

- ✓ The root node of the tree is called the parent node, and other nodes are called the child nodes.

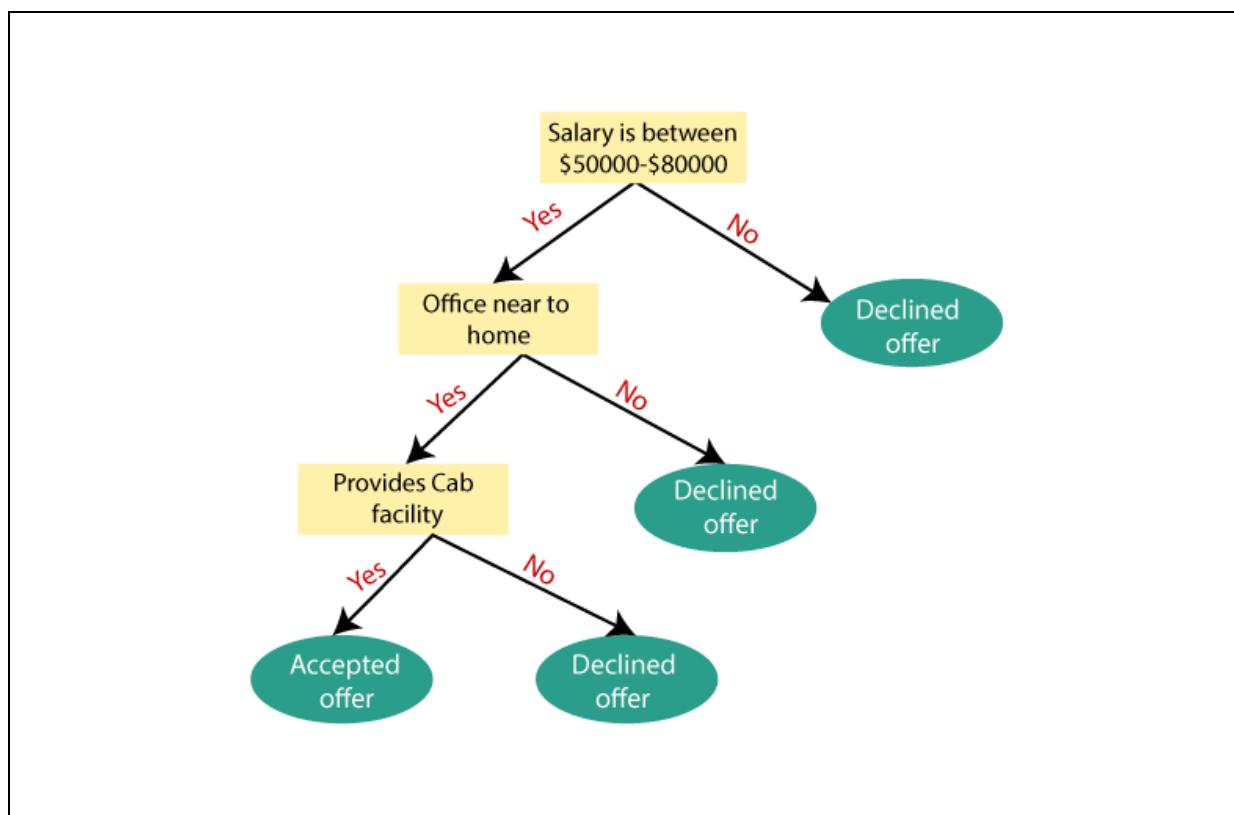
### 7. How does the Decision Tree algorithm Work?

- ✓ Step-1: Begin the tree with the root node, which contains the complete dataset.
- ✓ Step-2: Find the best attribute in the dataset using Attribute Selection Measure (ASM).
- ✓ Step-3: Divide the dataset into subsets that contains possible values for the best attributes.
- ✓ Step-4: Generate the decision tree node, which contains the best attribute.
- ✓ Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

## Data Science – Machine Learning – Decision Tree

### 7.1. Example

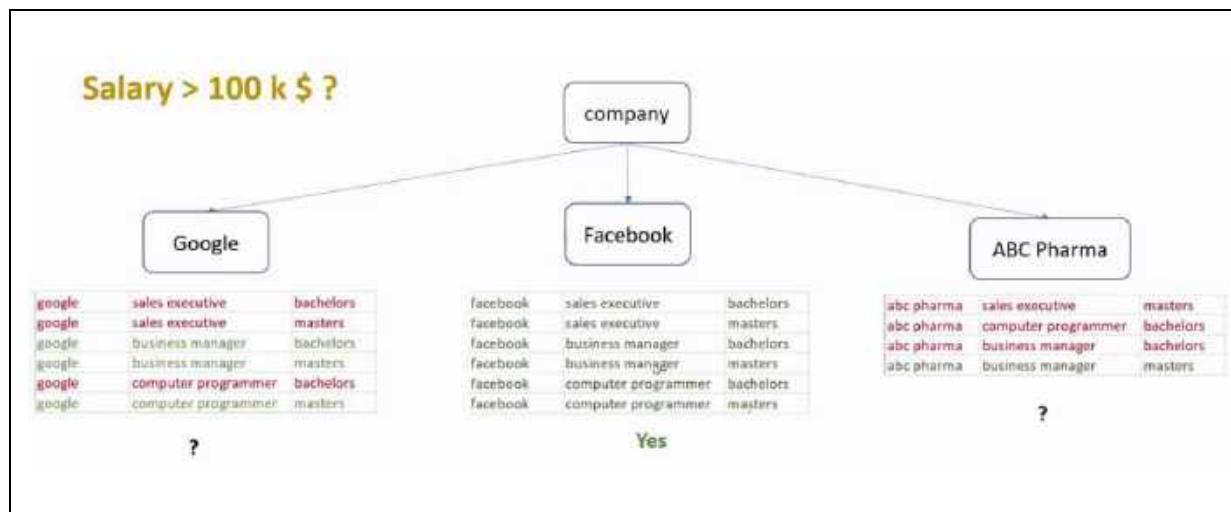
- ✓ Suppose there is a candidate who has a job offer and wants to decide whether he should accept the offer or Not.
- ✓ So, to solve this problem, the decision tree starts with the root node (Salary attribute by ASM).
- ✓ The root node splits further into the next decision node (distance from the office) and one leaf node based on the corresponding labels.
- ✓ The next decision node further gets split into one decision node (Cab facility) and one leaf node.
- ✓ Finally, the decision node splits into two leaf nodes (Accepted offers and Declined offer).



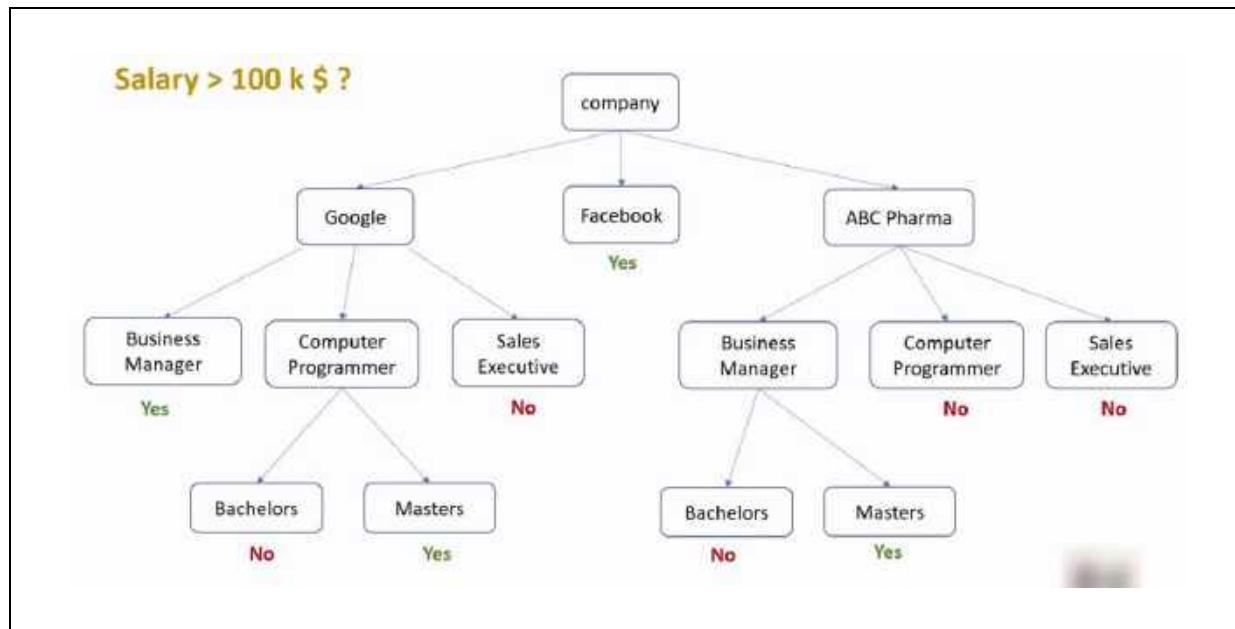
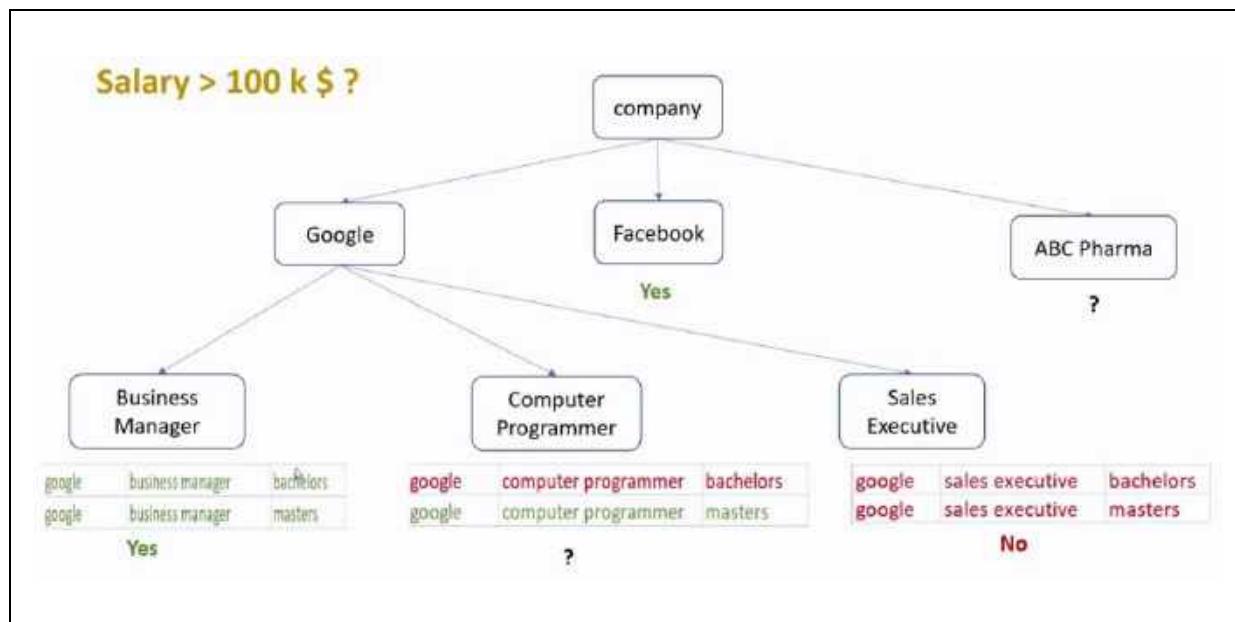
## Data Science – Machine Learning – Decision Tree

### 8. Problem

Company	Job	Degree	Salary_more_then_100k
google	sales executive	bachelors	0
google	sales executive	masters	0
google	business manager	bachelors	1
google	business manager	masters	1
google	computer programmer	bachelors	0
google	computer programmer	masters	1
abc pharma	sales executive	masters	0
abc pharma	computer programmer	bachelors	0
abc pharma	business manager	bachelors	0
abc pharma	business manager	masters	1
facebook	sales executive	bachelors	1
facebook	sales executive	masters	1
facebook	business manager	bachelors	1
facebook	business manager	masters	1
facebook	computer programmer	bachelors	1
facebook	computer programmer	masters	1



## Data Science – Machine Learning – Decision Tree



## Data Science – Machine Learning – Decision Tree

Program Name Loading dataset  
demo1.py

```
import pandas as pd

df = pd.read_csv("salaries.csv")
print(df)
```

### Output

	company	job	degree	salary_more_then_100k
0	google	sales executive	bachelors	0
1	google	sales executive	masters	0
2	google	business manager	bachelors	1
3	google	business manager	masters	1
4	google	computer programmer	bachelors	0
5	google	computer programmer	masters	1
6	abc pharma	sales executive	masters	0
7	abc pharma	computer programmer	bachelors	0
8	abc pharma	business manager	bachelors	0
9	abc pharma	business manager	masters	1
10	facebook	sales executive	bachelors	1
11	facebook	sales executive	masters	1
12	facebook	business manager	bachelors	1
13	facebook	business manager	masters	1
14	facebook	computer programmer	bachelors	1
15	facebook	computer programmer	masters	1

## Data Science – Machine Learning – Decision Tree

Program Name

Preparing input and target

demo2.py

```
import pandas as pd

df = pd.read_csv("salaries.csv")

inputs = df.drop('salary_more_then_100k', axis = 'columns')
target = df['salary_more_then_100k']

print("Input")
print(inputs.head())

print("Target")
print(target.head())
```

Output

```
Input
   company           job      degree
0  google    sales executive bachelors
1  google    sales executive    masters
2  google  business manager bachelors
3  google  business manager    masters
4  google  computer programmer bachelors
Target
0    0
1    0
2    1
3    1
4    0
Name: salary_more_then_100k, dtype: int64
```

## Data Science – Machine Learning – Decision Tree

---

**Program Name** Transforming input  
demo3.py

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

df = pd.read_csv("salaries.csv")

inputs = df.drop('salary_more_then_100k',axis='columns')
target = df['salary_more_then_100k']

le_company = LabelEncoder()

inputs['company_n'] =
    le_company.fit_transform(inputs['company'])

print(inputs)
```

### Output

	company	job	degree	company_n
0	google	sales executive	bachelors	2
1	google	sales executive	masters	2
2	google	business manager	bachelors	2
3	google	business manager	masters	2
4	google	computer programmer	bachelors	2
5	google	computer programmer	masters	2
6	abc pharma	sales executive	masters	0
7	abc pharma	computer programmer	bachelors	0
8	abc pharma	business manager	bachelors	0
9	abc pharma	business manager	masters	0
10	facebook	sales executive	bachelors	1
11	facebook	sales executive	masters	1
12	facebook	business manager	bachelors	1
13	facebook	business manager	masters	1
14	facebook	computer programmer	bachelors	1
15	facebook	computer programmer	masters	1

## Data Science – Machine Learning – Decision Tree

---

**Program Name** Transforming input  
demo4.py

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

df = pd.read_csv("salaries.csv")

inputs = df.drop('salary_more_then_100k',axis='columns')
target = df['salary_more_then_100k']

le_company = LabelEncoder()

inputs['company_n'] =
le_company.fit_transform(inputs['company'])
inputs['job_n'] = le_company.fit_transform(inputs['job'])

print(inputs)
```

### Output

	company	job	degree	company_n	job_n
0	google	sales executive	bachelors	2	2
1	google	sales executive	masters	2	2
2	google	business manager	bachelors	2	0
3	google	business manager	masters	2	0
4	google	computer programmer	bachelors	2	1
5	google	computer programmer	masters	2	1
6	abc pharma	sales executive	masters	0	2
7	abc pharma	computer programmer	bachelors	0	1
8	abc pharma	business manager	bachelors	0	0
9	abc pharma	business manager	masters	0	0
10	facebook	sales executive	bachelors	1	2
11	facebook	sales executive	masters	1	2
12	facebook	business manager	bachelors	1	0
13	facebook	business manager	masters	1	0
14	facebook	computer programmer	bachelors	1	1
15	facebook	computer programmer	masters	1	1

## Data Science – Machine Learning – Decision Tree

---

**Program Name** Transforming input  
demo5.py

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

df = pd.read_csv("salaries.csv")
inputs = df.drop('salary_more_then_100k',axis='columns')
target = df['salary_more_then_100k']

le_company = LabelEncoder()

inputs['company_n'] =
    le_company.fit_transform(inputs['company'])
inputs['job_n'] = le_company.fit_transform(inputs['job'])
inputs['degree_n'] = le_company.fit_transform(inputs['degree'])

print(inputs)
```

### Output

	company	job	degree	company_n	job_n	degree_n
0	google	sales executive	bachelors	2	2	0
1	google	sales executive	masters	2	2	1
2	google	business manager	bachelors	2	0	0
3	google	business manager	masters	2	0	1
4	google	computer programmer	bachelors	2	1	0
5	google	computer programmer	masters	2	1	1
6	abc pharma	sales executive	masters	0	2	1
7	abc pharma	computer programmer	bachelors	0	1	0
8	abc pharma	business manager	bachelors	0	0	0
9	abc pharma	business manager	masters	0	0	1
10	facebook	sales executive	bachelors	1	2	0
11	facebook	sales executive	masters	1	2	1
12	facebook	business manager	bachelors	1	0	0
13	facebook	business manager	masters	1	0	1
14	facebook	computer programmer	bachelors	1	1	0
15	facebook	computer programmer	masters	1	1	1

## Data Science – Machine Learning – Decision Tree

---

**Program Name** Transforming input  
demo6.py

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

df = pd.read_csv("salaries.csv")
inputs = df.drop('salary_more_then_100k',axis='columns')
target = df['salary_more_then_100k']

le_company = LabelEncoder()

inputs['company_n'] =
    le_company.fit_transform(inputs['company'])
inputs['job_n'] = le_company.fit_transform(inputs['job'])
inputs['degree_n'] = le_company.fit_transform(inputs['degree'])

inputs_n = inputs.drop(['company', 'job', 'degree'], axis='columns')

print(inputs_n)
```

### Output

	company_n	job_n	degree_n
0	2	2	0
1	2	2	1
2	2	0	0
3	2	0	1
4	2	1	0
5	2	1	1
6	0	2	1
7	0	1	0
8	0	0	0
9	0	0	1
10	1	2	0
11	1	2	1
12	1	0	0
13	1	0	1
14	1	1	0
15	1	1	1

## 9. DecisionTreeClassifier class

- ✓ **DecisionTreeClassifier** is predefined class in **sklearn.tree** package
- ✓ We need to import this class from **sklearn.tree** package
- ✓ Once we imported then we need to **create an object** to **DecisionTreeClassifier** class.

**Program**

**Name** Model creation

demo7.py

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier

df = pd.read_csv("salaries.csv")
inputs = df.drop('salary_more_then_100k',axis='columns')
target = df['salary_more_then_100k']

le_company = LabelEncoder()

inputs['company_n'] =
    le_company.fit_transform(inputs['company'])
inputs['job_n'] = le_company.fit_transform(inputs['job'])
inputs['degree_n'] = le_company.fit_transform(inputs['degree'])

inputs_n = inputs.drop(['company', 'job', 'degree'], axis='columns')

model = DecisionTreeClassifier()
print("DecisionTreeClassifier object created")
```

**Output**

DecisionTreeClassifier object created

## Data Science – Machine Learning – Decision Tree

---

### 9.1. fit(X\_train, y\_train) method

- ✓ fit(X\_train, y\_train) is predefined method in DecisionTreeClassifier class.
- ✓ We should access this method by using DecisionTreeClassifier object only.
- ✓ By using this method we need to train the model.

**Program**

**Name** Model creation

demo8.py

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier

df = pd.read_csv("salaries.csv")
inputs = df.drop('salary_more_then_100k',axis='columns')
target = df['salary_more_then_100k']

le_company = LabelEncoder()

inputs['company_n'] =
    le_company.fit_transform(inputs['company'])
inputs['job_n'] = le_company.fit_transform(inputs['job'])
inputs['degree_n'] = le_company.fit_transform(inputs['degree'])

inputs_n = inputs.drop(['company', 'job', 'degree'], axis='columns')

model = DecisionTreeClassifier()
model.fit(inputs_n.values, target)

print("Model got trained")
```

**Output**

Model got trained

## Data Science – Machine Learning – Decision Tree

Program

Name Model score

demo9.py

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier

df = pd.read_csv("salaries.csv")
inputs = df.drop('salary_more_then_100k',axis='columns')
target = df['salary_more_then_100k']

le_company = LabelEncoder()

inputs['company_n'] =
    le_company.fit_transform(inputs['company'])
inputs['job_n'] = le_company.fit_transform(inputs['job'])
inputs['degree_n'] = le_company.fit_transform(inputs['degree'])

inputs_n = inputs.drop(['company', 'job', 'degree'], axis='columns')

print("Model got trained")
model = DecisionTreeClassifier()
model.fit(inputs_n.values, target)

print(model.score(inputs_n, target))
```

Output

1.0

## Data Science – Machine Learning – Decision Tree

---

### 9.2. predict(p) method

- ✓ predict(p) is predefined method in DecisionTreeClassifier class.
- ✓ We should access this method by using DecisionTreeClassifier object only.
- ✓ By using this method we can predict the results.

**Program**

**Name**

Model prediction

demo10.py

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier

df = pd.read_csv("salaries.csv")
inputs = df.drop('salary_more_then_100k',axis='columns')
target = df['salary_more_then_100k']

le_company = LabelEncoder()

inputs['company_n'] =
    le_company.fit_transform(inputs['company'])
inputs['job_n'] = le_company.fit_transform(inputs['job'])
inputs['degree_n'] = le_company.fit_transform(inputs['degree'])

inputs_n = inputs.drop(['company', 'job', 'degree'], axis='columns')

print("Model got trained")
model = DecisionTreeClassifier()
model.fit(inputs_n.values, target)

print(model.predict([[2, 1, 0]]))
```

**Output**

[0]

## Data Science – Machine Learning – Decision Tree

Program Name

Model prediction

demo11.py

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier

df = pd.read_csv("salaries.csv")
inputs = df.drop('salary_more_then_100k',axis='columns')
target = df['salary_more_then_100k']

le_company = LabelEncoder()

inputs['company_n'] =
    le_company.fit_transform(inputs['company'])
inputs['job_n'] = le_company.fit_transform(inputs['job'])
inputs['degree_n'] = le_company.fit_transform(inputs['degree'])

inputs_n = inputs.drop(['company', 'job', 'degree'], axis='columns')

print("Model got trained")
model = DecisionTreeClassifier()
model.fit(inputs_n.values, target)

print(model.predict([[2, 1, 1]]))
```

Output

[1]

# Data Science – Machine Learning – Confusion Matrix

---

## 21. Data Science – Machine Learning – Confusion Matrix

### Contents

<b>1. Introduction</b> .....	2
<b>2. Confusion Matrix</b> .....	2
<b>3. Name of the each combination</b> .....	3
<b>4. Table1</b> .....	4
<b>5. Scenario</b> .....	5
5.1. True positive .....	6
5.2. True Negative.....	6
5.3. False Positive.....	7
5.4. False Negative.....	7
<b>6. Type I and Type II errors</b> .....	8
<b>7. Confusion Matrix</b> .....	9
<b>8. Table2</b> .....	9
<b>9. Confusion matrix example</b> .....	10
<b>10. Performance Metrics</b> .....	11
<b>11. Accuracy</b> .....	12
<b>12. Accuracy can be misleading</b> .....	13
<b>13. Recall or Sensitivity or True Positive Rate</b> .....	14
<b>14. Specificity or True Negative Rate</b> .....	15
<b>15. Precision</b> .....	16
<b>16. F1 Score</b> .....	18

## Data Science – Machine Learning – Confusion Matrix

### 21. Data Science – Machine Learning – Confusion Matrix

#### 1. Introduction

- ✓ Confusion matrix is a method to explain the results of the classification model.

#### 2. Confusion Matrix

- ✓ In binary classification, the outcomes are,
  - True
  - False
- ✓ To make it more clear let us consider an example of a binary classifier that scans the **MRI** images and **predicts whether a person has cancer or not**.
- ✓ The outcomes predicted by classifier and the actual outcomes can only have the following four combinations

Predicted	Actual
Cancer = Yes	Cancer = Yes
Cancer = Yes	Cancer = No
Cancer = No	Cancer = Yes
Cancer = No	Cancer = No

## Data Science – Machine Learning – Confusion Matrix

---

### 3. Name of the each combination

- ✓ Now let's understand clearly by comparing each other

Name	Predicted	Actual
True Positive	Cancer = Yes	Cancer = Yes
False Positive	Cancer = Yes	Cancer = No
False Negative	Cancer = No	Cancer = Yes
True Negative	Cancer = No	Cancer = No

## Data Science – Machine Learning – Confusion Matrix

---

### 4. Table1

<b>Name</b>	<b>Description</b>
<b>1. True Positive</b>	<ul style="list-style-type: none"> <li>✓ The model predicts that the patient is <b>having</b> cancer (positive) and indeed he <b>has</b> cancer (true prediction).</li> </ul>
<b>2. False Positive</b>	<ul style="list-style-type: none"> <li>✓ The model predicts that the patient is <b>having</b> cancer (positive) but actually he is <b>not having</b> cancer (false prediction)</li> </ul>
<b>3. False Negative</b>	<ul style="list-style-type: none"> <li>✓ The model predicts that the patient is <b>not having</b> cancer (negative) but he actually <b>has</b> cancer (false prediction).</li> </ul>
<b>4. True Negative</b>	<ul style="list-style-type: none"> <li>✓ The model predicts that the patient is <b>not having</b> cancer (negative) and indeed he does <b>not have</b> cancer (true prediction).</li> </ul>

## Data Science – Machine Learning – Confusion Matrix

---

### 5. Scenario

- ✓ Now consider the above classification (pregnant or not pregnant) carried out by a machine learning algorithm. The output of the machine learning algorithm can be mapped to one of the following categories.

## Data Science – Machine Learning – Confusion Matrix

### 5.1. True positive

- ✓ A person who is actually pregnant (positive) and classified as pregnant (positive). This is called TRUE POSITIVE (TP).



### 5.2. True Negative

- ✓ A person who is actually not pregnant (negative) and classified as not pregnant (negative). This is called TRUE NEGATIVE (TN).



## Data Science – Machine Learning – Confusion Matrix

### 5.3. False Positive

- ✓ A person who is actually not pregnant (negative) and classified as pregnant (positive). This is called FALSE POSITIVE (FP).



### 5.4. False Negative

- ✓ A person who is actually pregnant (positive) and classified as not pregnant (negative). This is called FALSE NEGATIVE (FN).



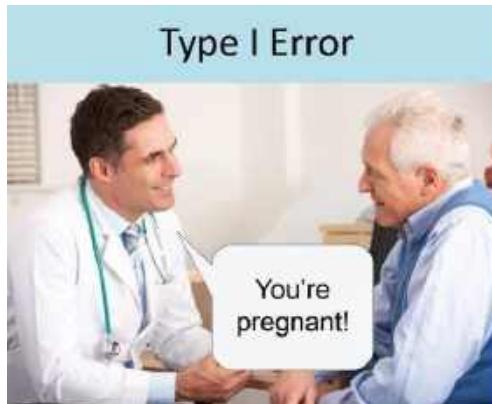
## 6. Type I and Type II errors

### Type I error

- ✓ Type I error also called as False Positive.
- ✓ Type I error occurs if you reject the null hypothesis when it was true.

### Type II error

- ✓ Type II error also called as False Negative
- ✓ Type II error occurs if you accept the null hypothesis when it was false.



## Data Science – Machine Learning – Confusion Matrix

---

### 7. Confusion Matrix

- ✓ We can represent above discussion by using matrix also

	Actual Cancer = Yes	Actual Cancer = No
Predicted Cancer = Yes	True Positive (TP)	False Positive (FP)
Predicted Cancer = No	False Negative (FN)	True Negative (TN)

### 8. Table2

- ✓ True Positive = No. of True Positives from total predictions
- ✓ False Positive = No. of False Positives from total predictions
- ✓ False Negative = No. of False Negatives from total predictions
- ✓ True Negative = No. of True Negatives from total predictions
- ✓ Total number of prediction =  $TP + FP + FN + TN$

## Data Science – Machine Learning – Confusion Matrix

### 9. Confusion matrix example

- ✓ Let's try to represent this confusion matrix with real numbers.

		Actual Cancer = Yes	Actual Cancer = No
Predicted Cancer = Yes	True Positive	False Positive	
	57	14	
Predicted Cancer = No	False Negative	True Negative	
	23	171	

- 57 = No. of True Positives from total predictions
- 14 = No. of False Positives from total predictions
- 23 = No. of False Negatives from total predictions
- 171 = No. of True Negatives from total predictions
- $57+14+23+171 = 265$  = Total no. of predictions

### 10. Performance Metrics

- ✓ If we have confusion matrix then we can use it to calculate various performance metrics to measure your classification models.

## Data Science – Machine Learning – Confusion Matrix

### 11. Accuracy

- ✓ Accuracy is a measure of the fraction of times the model predicted correctly (both true positive and true negative) out of total no. of predictions.

		Actual Cancer = Yes	Actual Cancer = No
Predicted Cancer = Yes	True Positive	False Positive 14	
	57		
Predicted Cancer = No	False Negative 23	True Negative 171	

$$\text{Accuracy} = \frac{TP+TN}{\text{Total Predictions}}$$

- ✓ This is the simple and very intuitive metrics.
- ✓ Let us consider the accuracy of the classifier in our example above.

$$\text{Accuracy} = \frac{57+171}{265} = 0.86$$

- ✓ So this means our model can classify cancer and non-cancer cases with 86% accuracy.

## Data Science – Machine Learning – Confusion Matrix

### 12. Accuracy can be misleading

- ✓ We felt like accuracy looks good to measure the performance of the model, but there is a catch!!
- ✓ Assume that your machine model trains badly and only identifies negative scenarios and miss to identify true positive completely. (This happens when there is a class imbalance in training data.)

	Actual Cancer = Yes	Actual Cancer = No
Predicted Cancer = Yes	True Positive 0	False Positive 0
Predicted Cancer = No	False Negative 80	True Negative 185

- ✓ So in our example, say the classifier completely misses to predict any case of cancer and marks all cases as non-cancer, this is how the confusion matrix will look like.
- ✓ So the accuracy of our classifier will be

$$\text{Accuracy} = \frac{185}{265} = 0.70$$

- ✓ So here model could not predict a single cancer case but we got accuracy is 70% which is not so good😊.
- ✓ So, only believing Accuracy is not good way
- ✓ This is why we have several other metrics to measure the performances of the machine learning model.

## Data Science – Machine Learning – Confusion Matrix

### 13. Recall or Sensitivity or True Positive Rate

- ✓ Recall is the fraction of times the model predicts positive cases correctly from the total number of actual positive cases.

$$\text{Recall} = \frac{TP}{TP+FN}$$

	Actual Cancer = Yes	Actual Cancer = No
Predicted Cancer = Yes	True Positive 57	False Positive 14
Predicted Cancer = No	False Negative 23	True Negative 171

- ✓ Let's understand, TP + FN is the total number of actual positive cases that the classifier should have identified.
- ✓ In our example, TP = 57 cancer cases classifier predicted correctly.
- ✓ FN = 23 cancer cases our classifier missed.
- ✓ SO, the total cancer cases were TP + FN = 57+23 = 80.

$$\text{Recall} = \frac{57}{57+23} = \frac{57}{80} = 0.71$$

- ✓ So this means our model has 71% recall capability to predict cancer cases from total actual cancer cases.

## Data Science – Machine Learning – Confusion Matrix

### 14. Specificity or True Negative Rate

- ✓ Specificity is the fraction of times the classifier predicts negatives cases correctly from the total number of actual negative cases.
- ✓ This metrics is the opposite of Recall/Sensitivity that we understood above.

	Actual Cancer = Yes	Actual Cancer = No
Predicted Cancer = Yes	True Positive 57	False Positive 14
Predicted Cancer = No	False Negative 23	True Negative 171

$$\text{Specificity} = \frac{TN}{TN+FP}$$

- ✓ Here  $TN+FP$  is the total number of actual negative cases.
- ✓ In our example,  $TN = 171$  non-cancer cases were predicted correctly by the classifier and  $FP = 14$  non-cancer cases were incorrectly identified as cancer.
- ✓ So, the total non-cancer cases were  $TN+FP = 171+14 = 185$ .

$$\text{Specificity} = \frac{171}{171+14} = \frac{171}{185} = 0.92$$

- ✓ So this means our model has 92% capability to identify negative cases from total number of actual negative cases.

## Data Science – Machine Learning – Confusion Matrix

### 15. Precision

- ✓ Precision calculates the fraction of times the model predicts positive cases correctly from the total number of positive cases it predicted.

$$\text{Precision} = \frac{TP}{TP+FP}$$

- ✓ Let's try to understand TP+FP is the total number of cases predicted by the classifier as positive.
- ✓ TP is the total number of cases that are actually positive.

		Actual Cancer = Yes	Actual Cancer = No
Predicted Cancer = Yes	True Positive	False Positive 14	
	57		
Predicted Cancer = No	False Negative 23	True Negative 171	

- ✓ In our example, the classifier predicted  $TP+FP = 71$  cases of cancer.
- ✓ Out of these 71 cases, only  $TP = 57$  cases were correct.

$$\text{Precision} = \frac{57}{57+14} = \frac{57}{71} = 0.80$$

## Data Science – Machine Learning – Confusion Matrix

---

- ✓ So this means our classifier has a precision or correctness of 80% when it predicts cases as cancer.

### 16. F1 Score

- ✓ So, Recall and precision are the best way to choose.
- ✓ Instead of using many metrics, Can't we chose one metric?
- ✓ Yes we can, here F1 score helps actually
- ✓ The F1 score is a measure of a model's accuracy.

$$\text{F1 Score} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

- ✓ Using F1 scores we can now evaluate and compare the performance of the models easily now.

**22. Data Science – Machine Learning – Bias Variance Trade Off****Contents**

<b>1. Bias.....</b>	<b>2</b>
<b>2. Variance.....</b>	<b>2</b>
<b>3. Remember .....</b>	<b>2</b>
<b>4. The goal of supervised algorithm .....</b>	<b>2</b>
<b>5. Error .....</b>	<b>2</b>
<b>6. Prediction error .....</b>	<b>2</b>
<b>7. Bias Error .....</b>	<b>3</b>
<b>8. Low Bias.....</b>	<b>3</b>
<b>9. High-Bias.....</b>	<b>3</b>
<b>10. Examples.....</b>	<b>3</b>
<b>11. Variance Error.....</b>	<b>4</b>
<b>12. Low variance:.....</b>	<b>4</b>
<b>13. High variance .....</b>	<b>4</b>
<b>14. Examples.....</b>	<b>4</b>
<b>15. Bias-Variance Trade-Off .....</b>	<b>5</b>
<b>16. Configuring the bias-variance trade-off for specific algorithms.....</b>	<b>5</b>
<b>17. The relationship b/w bias and variance .....</b>	<b>5</b>
<b>18. In reality.....</b>	<b>5</b>
<b>19. Bias and variance.....</b>	<b>6</b>

## Data Science – Machine Learning – Bias Variance Trade Off

---

### 22. Data Science – Machine Learning – Bias Variance Trade Off

#### 1. Bias

- ✓ Bias is the error or difference between prediction results and actual values.

#### 2. Variance

- ✓ Variance is the error that occurs due to small changes in the training set.

#### 3. Remember

- ✓ The bias and variance provide the information to understand the performance of machine learning algorithms while prediction.

#### 4. The goal of supervised algorithm

- ✓ In supervised learning technique, an algorithm learns a model from training data.
- ✓ The goal of any supervised learning algorithm is,
  - Find the best mapping function/target function ( $f$ ) for the output variable ( $Y$ ) for the given input data ( $X$ ).

#### 5. Error

- ✓ Error means some wrong calculation.
- ✓ Error can be small error or big error.
- ✓ Small error can be acceptable and big error should be minimize

#### 6. Prediction error

- ✓ Prediction error means, we made some prediction but there are some errors are existing.
- ✓ The prediction error for any machine learning algorithm can be divided into two parts
  - Bias Error
  - Variance Error

## Data Science – Machine Learning – Bias Variance Trade Off

---

### 7. Bias Error

- ✓ Bias are the simplifying assumptions made by a model to make the target function easy to learn.

### 8. Low Bias

- ✓ Low bias value suggests **more assumptions** about the form of the target function.

### 9. High-Bias

- ✓ High bias value suggests **fewer assumptions** about the form of the target function.

### 10. Examples

- ✓ Low-bias machine learning algorithms are,

- Decision Trees
- k-Nearest Neighbors
- Support Vector Machines.

- ✓ High-bias machine learning algorithms are,

- Linear Regression
- Linear Discriminant Analysis
- Logistic Regression.

## Data Science – Machine Learning – Bias Variance Trade Off

---

### 11. Variance Error

- ✓ Variance is the error that occurs due to small changes in the training set.
- ✓ The target function is estimated from the training data by a machine learning algorithm, we can expect the algorithm may have some variance.
- ✓ Ideally, it should not change too much from one training dataset to the other.

### 12. Low variance:

- ✓ Low variance value suggests small changes to the estimate of the target function with changes to the training dataset.

### 13. High variance

- ✓ High variance value suggests large changes to the estimate of the target function with changes to the training dataset.

### 14. Examples

- ✓ Low-variance machine learning algorithms are,
  - Linear Regression
  - Linear Discriminant Analysis
  - Logistic Regression.
- ✓ High-variance machine learning algorithms are,
  - Decision Trees
  - k-Nearest Neighbors
  - Support Vector Machines.

## Data Science – Machine Learning – Bias Variance Trade Off

---

### 15. Bias-Variance Trade-Off

- ✓ The goal of any supervised machine learning algorithm is to achieve the sweet spot of **low bias and low variance**.
- ✓ It means, the algorithm should achieve good prediction performance.

#### Note 1

- ✓ Parametric or linear machine learning algorithms often have a **high bias** but a **low variance**.
- ✓ Nonparametric or nonlinear machine learning algorithms often have a **low bias** but a **high variance**.

### 16. Configuring the bias-variance trade-off for specific algorithms

- ✓ Below are two examples of configuring the bias-variance trade-off for specific algorithms
  - The **k-nearest neighbour's algorithm** has low bias and high variance, but the trade-off can be changed by increasing the value of k which increases the number of neighbors that contribute to the prediction and in turn increases the bias of the model.
  - The **support vector machine algorithm** has low bias and high variance, but the trade-off can be changed by increasing the C parameter that influences the number of violations of the margin allowed in the training data which increases the bias but decreases the variance.

### 17. The relationship b/w bias and variance

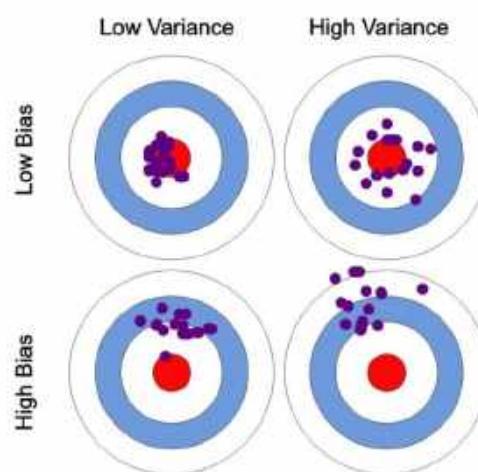
- ✓ Increasing the bias will decrease the variance.
- ✓ Increasing the variance will decrease the bias.

### 18. In reality

- ✓ In reality we cannot calculate the real bias and variance error terms because we do not know the actual underlying target function.

## Data Science – Machine Learning – Bias Variance Trade Off

### 19. Bias and variance



#### Error

- ✓ The error is calculated as the difference between predicted and actual value.

#### Low bias and low variance

- ✓ This is also called as Bias-Variance Tradeoff.
- ✓ If low bias and low variance then the error is very less.
- ✓ So, in this scenario the model is very accurate

#### Low bias and high variance

- ✓ So, in this scenario the model having lower accuracy.

#### High bias and low variance

- ✓ So, in this scenario the model having lower accuracy.

#### High bias and High variance

- ✓ So, in this scenario the model having lower accuracy.

## Data Science – Machine Learning – Bias Variance Trade Off

pip install mlxtend

**Program Name** Calculate bias and variance values  
demo1.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from mlxtend.evaluate import bias_variance_decomp

df = pd.read_csv('student_scores.csv')

X = df.iloc[:, :-1].values
y = df.iloc[:, 1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)

model = LinearRegression()

mse, bias, var = bias_variance_decomp(model, X_train, y_train,
X_test, y_test, loss='mse', num_rounds = 200, random_seed = 1)

print('MSE:', mse)
print('Bias:', bias)
print('Variance:', var)
```

**Output**

```
MSE: 25.994907724912153
Bias: 22.41211837451609
Variance: 3.5827893503960495
```

- ✓ In this case, we can see that the model has a high bias and a low variance.

**23. Data Science – Machine Learning – Random Forest Algorithm****Contents**

<b>1. Random Forest .....</b>	<b>2</b>
<b>2. Ensemble learning .....</b>	<b>2</b>
<b>3. Process behind in Random Forest .....</b>	<b>2</b>
<b>5. Why use Random Forest? .....</b>	<b>3</b>
<b>6. How does Random Forest algorithm work?.....</b>	<b>3</b>
<b>7. Steps in Random Forest .....</b>	<b>4</b>
<b>8. Use case .....</b>	<b>5</b>

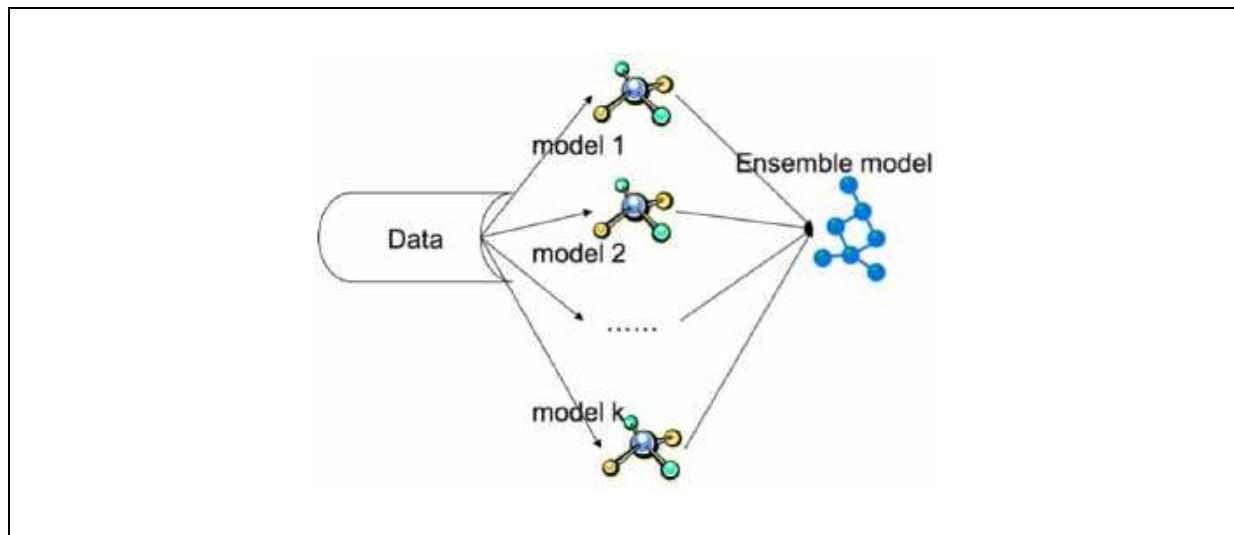
## 23. Data Science – Machine Learning – Random Forest Algorithm

### 1. Random Forest

- ✓ Random Forest is supervised learning technique.
- ✓ It can be used for both Classification and Regression problems in ML.
- ✓ Random forest is the concept of ensemble learning.

### 2. Ensemble learning

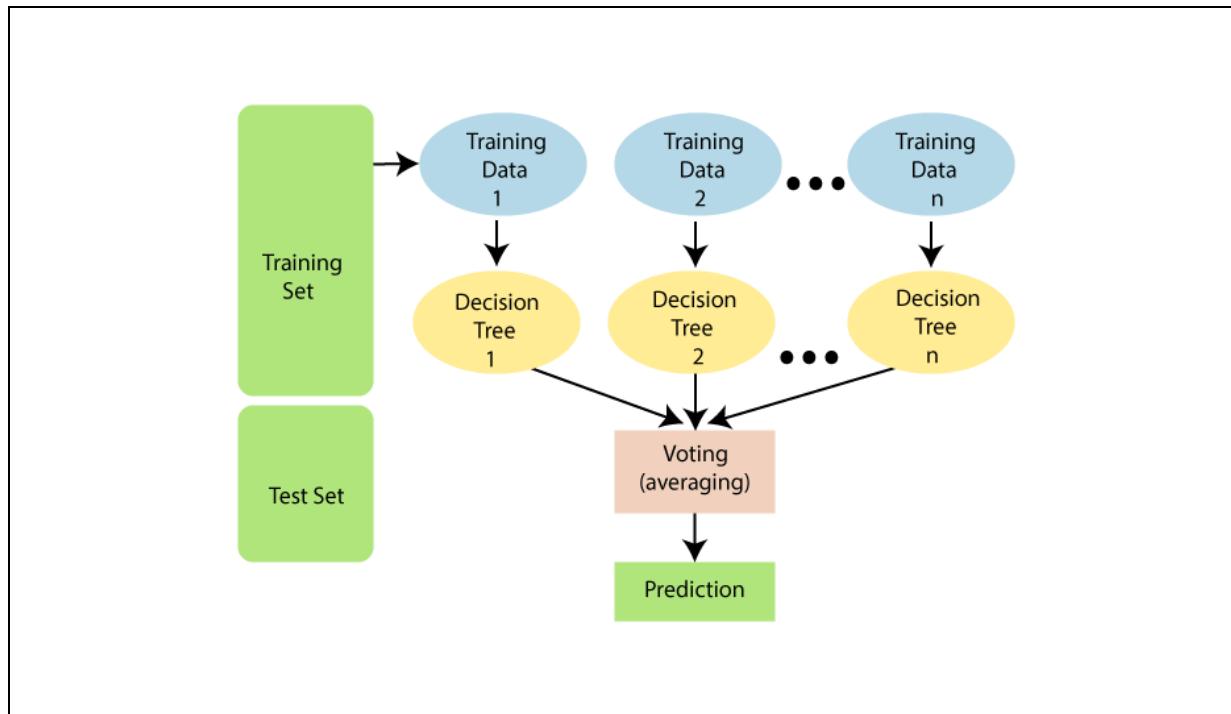
- ✓ This is the process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.



### 3. Process behind in Random Forest

- ✓ Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset.
- ✓ It takes the average of all decision trees to improve the predictive accuracy of that dataset.
  - Instead of believing on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.
- ✓ The greater number of trees in the forest leads to higher accuracy.

## Data Science – Machine Learning – Random Forest Algorithm



### 5. Why use Random Forest?

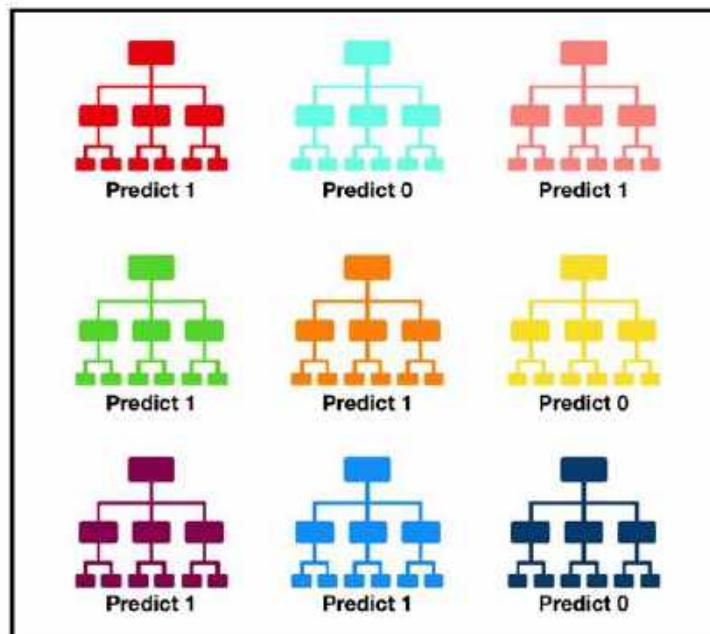
- ✓ It takes less training time as compared to other algorithms.
- ✓ It predicts output with high accuracy, even for the large dataset it runs efficiently.
- ✓ It can also maintain accuracy when a large proportion of data is missing.

### 6. How does Random Forest algorithm work?

- ✓ Random Forest works in two-phase,
  - First it creates the random forest by combining N decision tree.
  - Second is to make predictions for each tree created in the first phase.

## 7. Steps in Random Forest

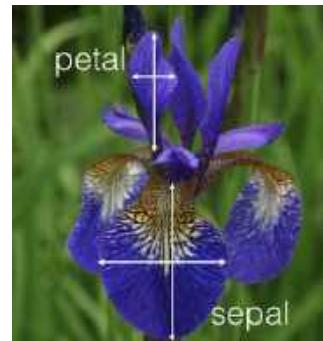
- ✓ Step-1: Select random K data points from the training set.
- ✓ Step-2: Build the decision trees associated with the selected data points
- ✓ Step-3: Choose the number N for decision trees that you want to build.
- ✓ Step-4: Repeat Step 1 & 2.
- ✓ Step-5: For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.



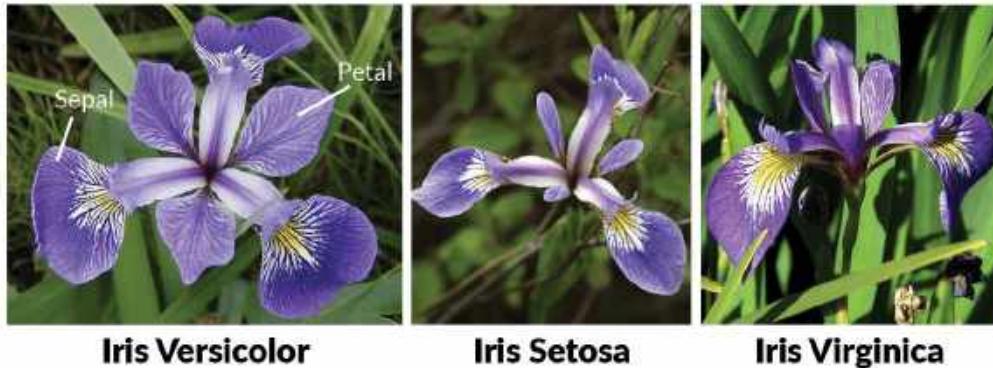
## Data Science – Machine Learning – Random Forest Algorithm

### 8. Use case

- ✓ Assuming that Abhi had a hobby which is interested in distinguishing the species of some iris flowers that he has found
- ✓ He has collected some measurements associated with each iris, which are:
  - The **length** and **width** of the **petals**
  - The **length** and **width** of the **sepals**, all measured in centimetres.
- ✓ He also has the measurements of some irises that have been previously identified to the species
  - setosa,
  - versicolor
  - virginica
- ✓ The goal is to create a machine learning model that can learn from the measurements of these irises whose species are already known.
- ✓ So that we can predict the species for the new irises that she has found.



## Data Science – Machine Learning – Random Forest Algorithm



### Flower codes

- ✓ Setosa - 0
- ✓ Versicolor - 1
- ✓ Virginica - 2

## Data Science – Machine Learning – Random Forest Algorithm

**Program Name** Loading iris dataset  
demo1.py

```
from sklearn.datasets import load_iris

iris = load_iris()

print(dir(iris))
```

**Output**

```
['DESCR', 'data', 'feature_names', 'filename', 'frame', 'target',
'target_names']
```

**Program Name** Displaying feature names  
demo2.py

```
from sklearn.datasets import load_iris

iris = load_iris()

print(iris.feature_names)
```

**Output**

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal
width (cm)']
```

## Data Science – Machine Learning – Random Forest Algorithm

---

**Program Name** Displaying target names  
demo3.py

```
from sklearn.datasets import load_iris

iris = load_iris()

print(iris.target_names)
```

**Output**

```
['setosa' 'versicolor' 'virginica']
```

## Data Science – Machine Learning – Random Forest Algorithm

**Program Name** Displaying data  
demo4.py

```
from sklearn.datasets import load_iris

iris = load_iris()

print(iris.data)
```

**Output**

```
[[5.1 3.5 1.4 0.2]
 [4.9 3. 1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5. 3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5. 3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3. 1.4 0.1]
 [4.3 3. 1.1 0.1]]
```

## Data Science – Machine Learning – Random Forest Algorithm

---

**Program Name** Length of the data  
demo5.py

```
from sklearn.datasets import load_iris

iris = load_iris()

print(len(iris.data))
```

**Output**  
150

## Data Science – Machine Learning – Random Forest Algorithm

**Program Name** Create a Dataframe by using data and features  
demo6.py

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

print(df)
```

### Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
..	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

[150 rows x 4 columns]

## Data Science – Machine Learning – Random Forest Algorithm

**Program Name** Adding target column to the dataframe  
demo7.py

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns = iris.feature_names)
df['target'] = iris.target

print(df.head())
```

### Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

## Data Science – Machine Learning – Random Forest Algorithm

**Program Name** Displaying target == 0 flowers  
demo8.py

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns = iris.feature_names)
df['target'] = iris.target

print(df[df.target == 0].head())
```

### Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

## Data Science – Machine Learning – Random Forest Algorithm

**Program Name** Displaying length of the target == 0 flowers  
demo9.py

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns = iris.feature_names)
df['target'] = iris.target

print(len(df[df.target == 0]))
```

**Output**

50

## Data Science – Machine Learning – Random Forest Algorithm

**Program Name** Displaying target == 1 flowers  
demo10.py

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

print(df[df.target == 1].head())
```

**Output**

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
50	7.0	3.2	4.7	1.4	1
51	6.4	3.2	4.5	1.5	1
52	6.9	3.1	4.9	1.5	1
53	5.5	2.3	4.0	1.3	1
54	6.5	2.8	4.6	1.5	1

## Data Science – Machine Learning – Random Forest Algorithm

**Program Name** Displaying length of the target == 1 flowers  
demo11.py

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

print(len(df[df.target==1]))
```

**Output**

50

## Data Science – Machine Learning – Random Forest Algorithm

**Program Name** Displaying target == 2 flowers  
demo12.py

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns = iris.feature_names)
df['target'] = iris.target

print(df[df.target==2].head())
```

### Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
100	6.3	3.3	6.0	2.5	2
101	5.8	2.7	5.1	1.9	2
102	7.1	3.0	5.9	2.1	2
103	6.3	2.9	5.6	1.8	2
104	6.5	3.0	5.8	2.2	2

## Data Science – Machine Learning – Random Forest Algorithm

**Program Name** Displaying length of the target == 2 flowers  
demo13.py

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns = iris.feature_names)
df['target'] = iris.target

print(len(df[df.target == 2]))
```

**Output**

50

## Data Science – Machine Learning – Random Forest Algorithm

---

**Program Name** Displaying the flower names  
demo14.py

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

a = lambda x: iris.target_names[x]
df['flower_name'] = df.target.apply(a)
print(df)
```

### Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa
..	...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2	virginica
146	6.3	2.5	5.0	1.9	2	virginica
147	6.5	3.0	5.2	2.0	2	virginica
148	6.2	3.4	5.4	2.3	2	virginica
149	5.9	3.0	5.1	1.8	2	virginica

[150 rows x 6 columns]

## Data Science – Machine Learning – Random Forest Algorithm

Program Name All setosa flowers  
demo15.py

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
a = lambda x: iris.target_names[x]
df['flower_name'] = df.target.apply(a)

setosa_50 = df[:50]
print(setosa_50.head())
```

### Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa

## Data Science – Machine Learning – Random Forest Algorithm

**Program Name** All versicolor flowers  
demo16.py

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
a = lambda x: iris.target_names[x]
df['flower_name'] = df.target.apply(a)

versicolor_50 = df[50:100]
print(versicolor_50.head())
```

### Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
50	7.0	3.2	4.7	1.4	1	versicolor
51	6.4	3.2	4.5	1.5	1	versicolor
52	6.9	3.1	4.9	1.5	1	versicolor
53	5.5	2.3	4.8	1.3	1	versicolor
54	6.5	2.8	4.6	1.5	1	versicolor

## Data Science – Machine Learning – Random Forest Algorithm

**Program Name** All virginica flowers  
demo17.py

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
a = lambda x: iris.target_names[x]
df['flower_name'] = df.target.apply(a)

verginica_50 = df[100:]
print(verginica_50.head())
```

### Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
100	6.3	3.3	6.0	2.5	2	virginica
101	5.8	2.7	5.1	1.9	2	virginica
102	7.1	3.0	5.9	2.1	2	virginica
103	6.3	2.9	5.6	1.8	2	virginica
104	6.5	3.0	5.8	2.2	2	virginica

## Data Science – Machine Learning – Random Forest Algorithm

**Program Name** Splitting the data  
demo18.py

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
a = lambda x: iris.target_names[x]
df['flower_name'] = df.target.apply(a)

X = df.drop(['target', 'flower_name'], axis = 'columns')
y = df.target

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)

print("Splitting the data")
```

**Output**

Splitting the data

## Data Science – Machine Learning – Random Forest Algorithm

Program Name Model training  
demo19.py

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
a = lambda x: iris.target_names[x]
df['flower_name'] = df.target.apply(a)

X = df.drop(['target', 'flower_name'], axis='columns')
y = df.target

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)

# Train Using RandomForestClassifier

model = RandomForestClassifier(n_estimators=40)
model.fit(X_train, y_train)
print("Model got trained")
```

### Output

Model got trained

## Data Science – Machine Learning – Random Forest Algorithm

Program Name Model score  
demo20.py

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
a = lambda x: iris.target_names[x]
df['flower_name'] = df.target.apply(a)

X = df.drop(['target', 'flower_name'], axis='columns')
y = df.target

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)

model = RandomForestClassifier(n_estimators=40)
model.fit(X_train, y_train)
print(model.score(X_test, y_test))
```

Output

0.9666666666666667

## Data Science – Machine Learning – Random Forest Algorithm

---

### Note

- ✓ `n_estimators=40` parameter defines the number of trees in the random forest.

## Data Science – Machine Learning – Random Forest Algorithm

Program Name

Model prediction

demo21.py

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
a = lambda x: iris.target_names[x]
df['flower_name'] = df.target.apply(a)

X = df.drop(['target', 'flower_name'], axis='columns')
y = df.target

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)

model = RandomForestClassifier(n_estimators = 40)

model.fit(X_train.values, y_train)

print(model.predict([[4.8,3.0,1.5,0.3]]))
```

Output

[0]

## 24. Data Science – Machine Learning – K Fold Cross Validation

### Contents

1. K-fold cross validation.....	2
2. Ways to training the model .....	2
3. Scenario 1 .....	3
4. Scenario 2 .....	4
5. Limitation.....	5
6. K fold cross validation .....	5
7. Iteration 1:.....	5
8. Iteration 2:.....	6
9. Last iteration: .....	7
10. Average of scores .....	7

## Data Science – Machine Learning – K Fold Cross Validation

### 24. Data Science – Machine Learning – K Fold Cross Validation

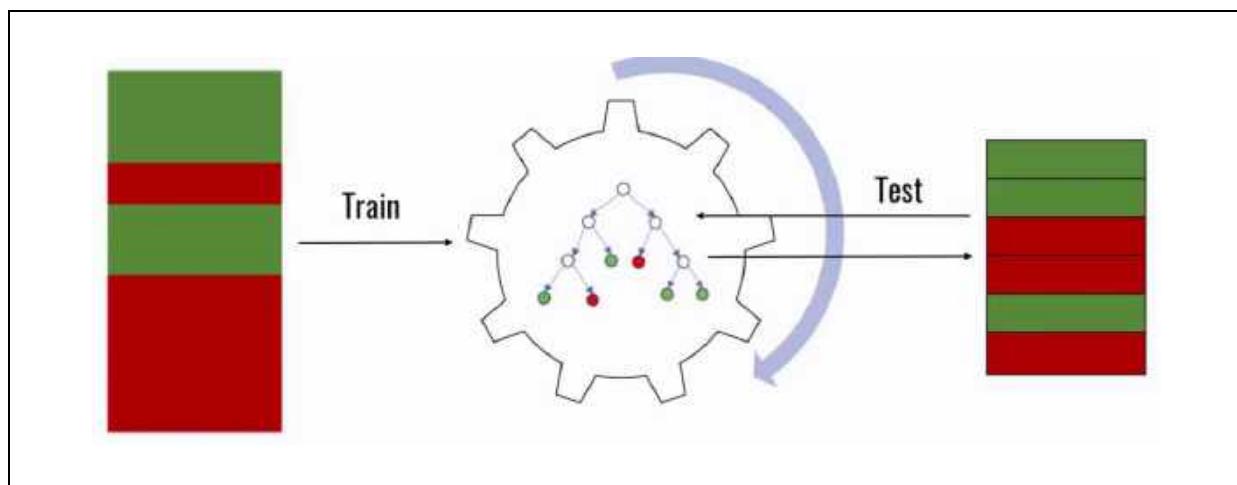
- ✓ To solve a problem we do have different machine learning algorithm for same problem
- ✓ So, we need to understand clearly which model is the best to use

#### 1. K-fold cross validation

- ✓ Cross-validation is a technique, it evaluate the model performance.

#### 2. Ways to training the model

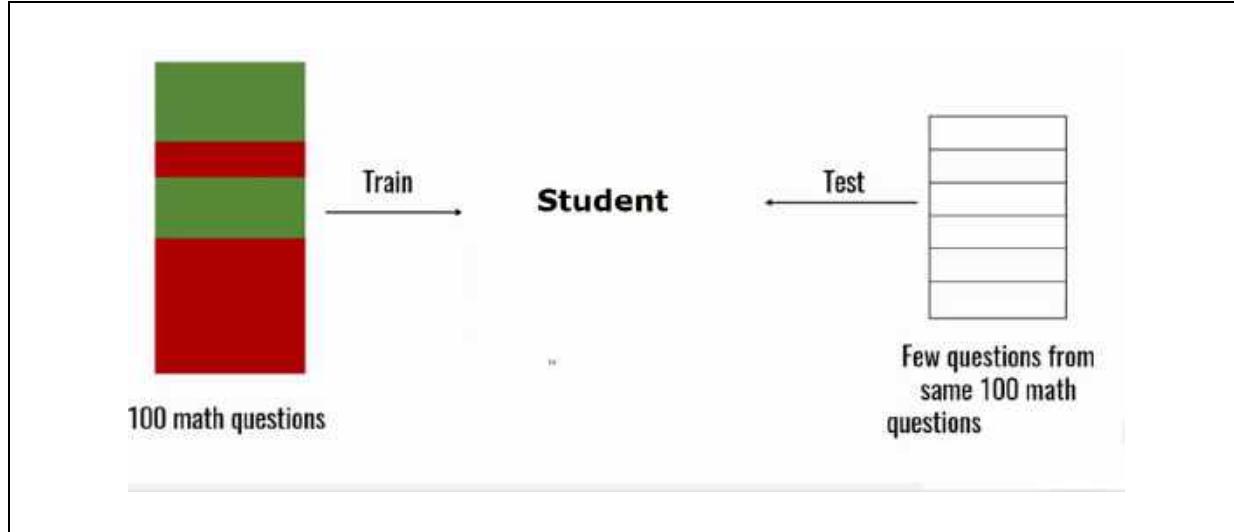
- ✓ So far we learned to spilt the data into train and test datasets
- ✓ Once the model got trained then we need to test the model



## Data Science – Machine Learning – K Fold Cross Validation

### 3. Scenario 1

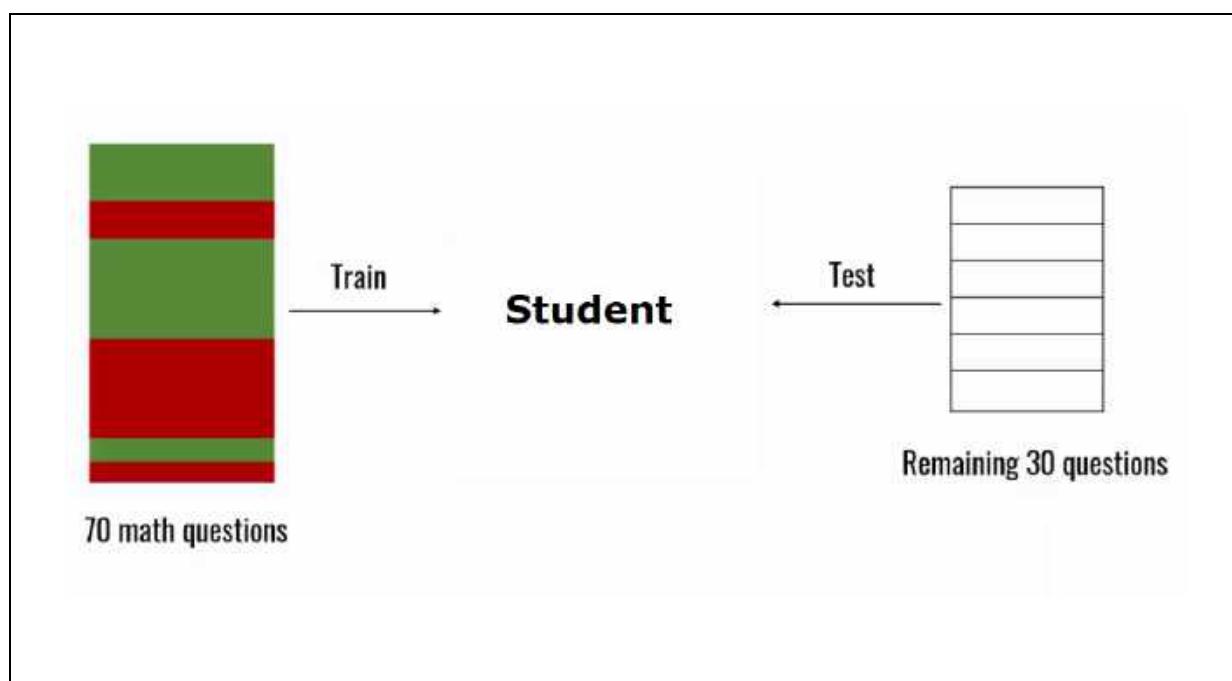
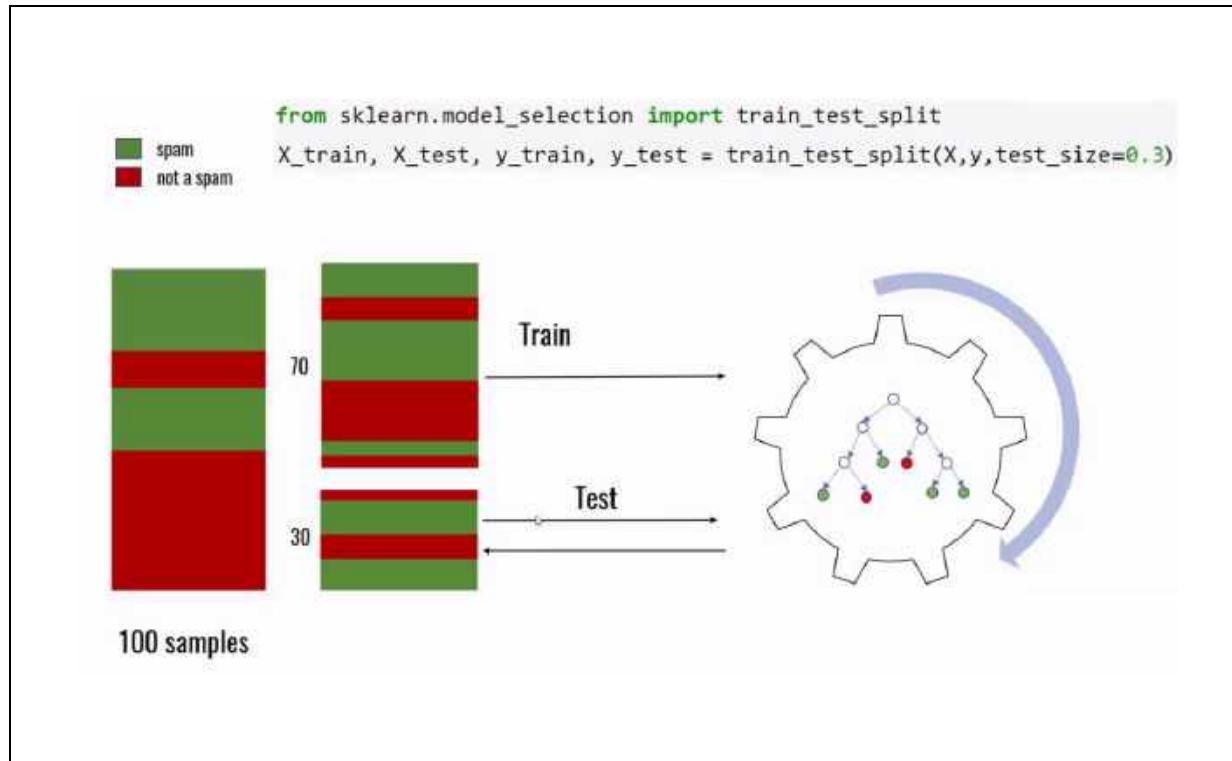
- ✓ Use all dataset to train and test the model



## Data Science – Machine Learning – K Fold Cross Validation

### 4. Scenario 2

- ✓ Split available dataset into training and testing to test the model



## Data Science – Machine Learning – K Fold Cross Validation

### 5. Limitation

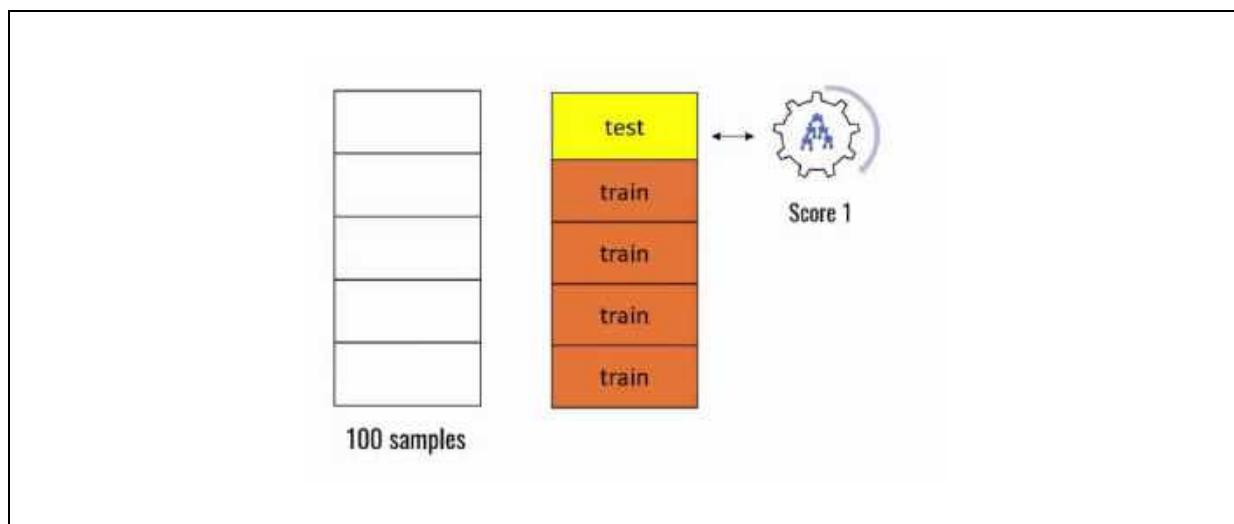
- ✓ During testing the model with test dataset, model may fail in few scenarios because model need to face new scenarios right

### 6. K fold cross validation

- ✓ Cross-validation is a technique, it evaluate the model performance.
- ✓ We used to divide 100 samples into folds, each contains 20 samples
- ✓ Then now we can start iterations to test the model in different ways

### 7. Iteration 1

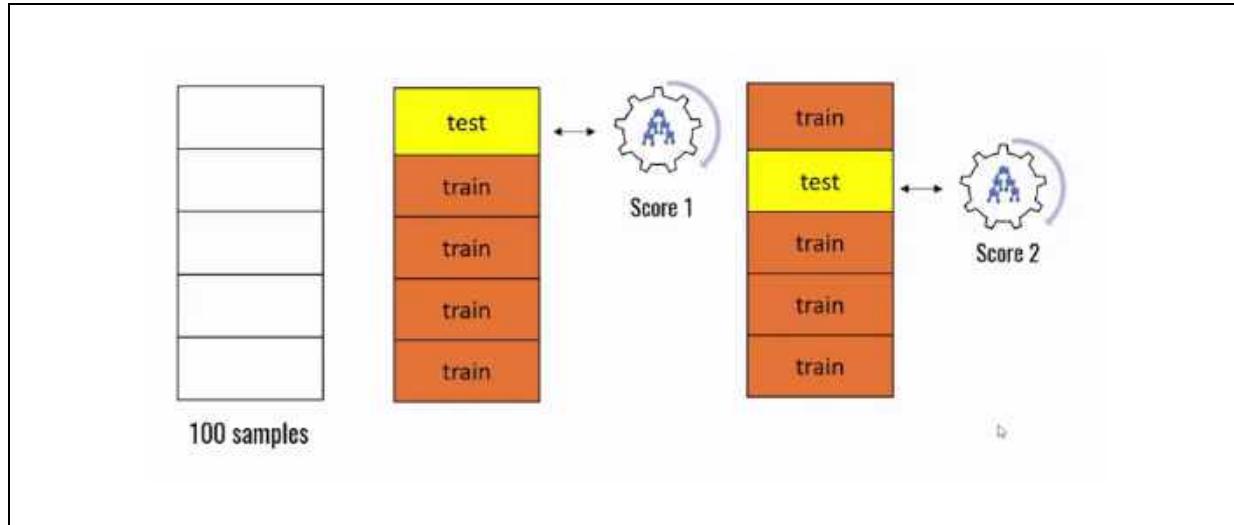
- ✓ Use the first fold to test the model
- ✓ Use the remaining folds to training



## Data Science – Machine Learning – K Fold Cross Validation

### 8. Iteration 2

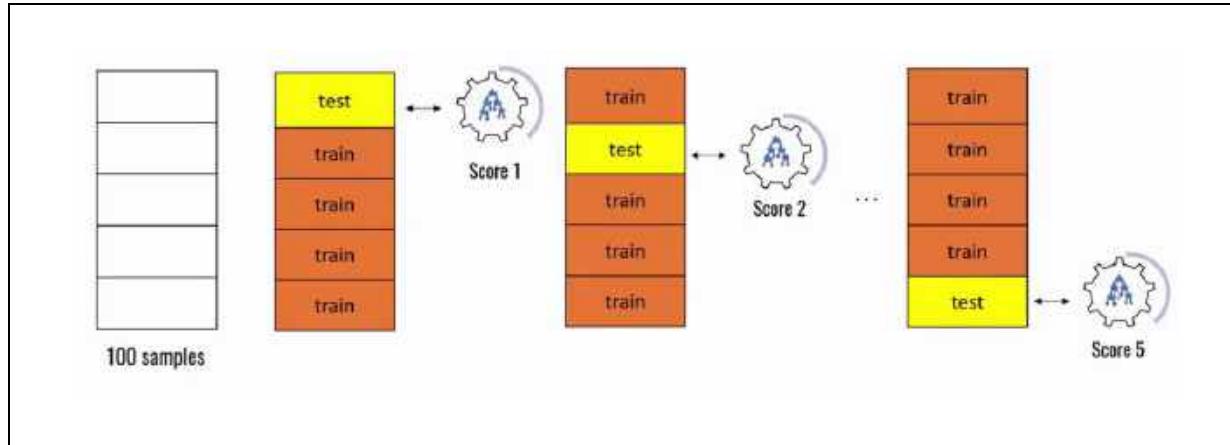
- ✓ Use the second fold to test the model
- ✓ Use the remaining folds to training



## Data Science – Machine Learning – K Fold Cross Validation

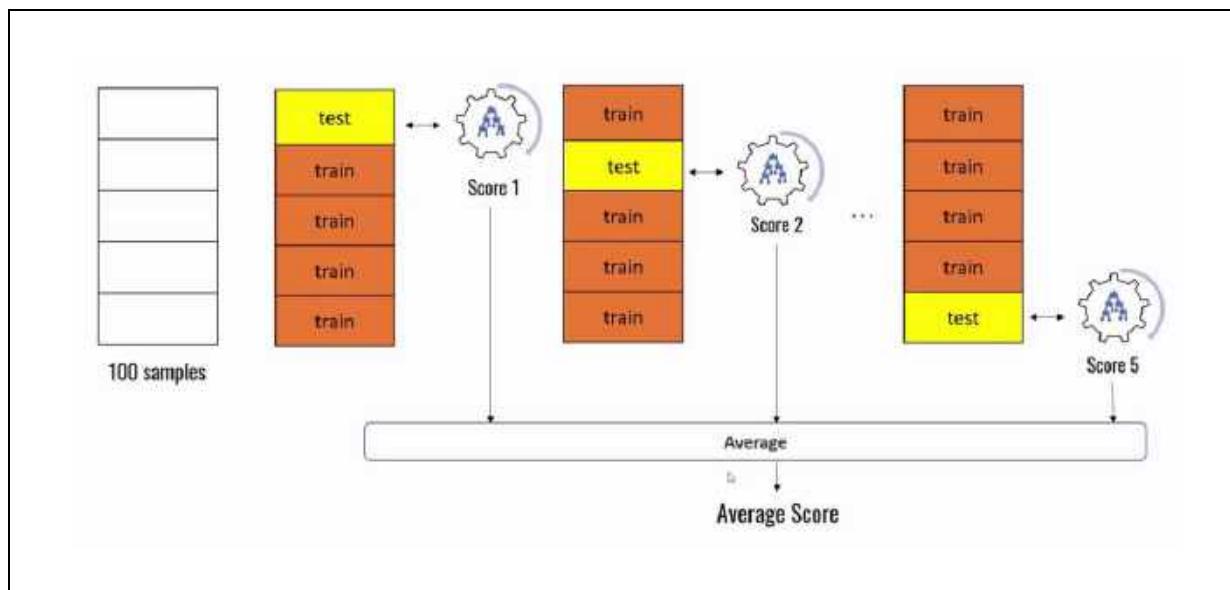
### 9. Last iteration

- ✓ Repeat the process till to the last fold



### 10. Average of scores

- ✓ This technique is good to calculate the average of all iterations scores



## Data Science – Machine Learning – K Fold Cross Validation

**Program Name**      Dataset is loading  
demo1.py

```
from sklearn.datasets import load_digits  
  
digits = load_digits()  
  
print("Dataset is loading")
```

**Output**                  Dataset is loading

**Program Name**      Splitting the data into train and test datasets  
demo2.py

```
from sklearn.datasets import load_digits  
from sklearn.model_selection import train_test_split  
  
digits = load_digits()  
X_train, X_test, y_train, y_test = train_test_split(digits.data,  
digits.target, test_size = 0.3)  
  
print("Splitting the data into train and test")
```

**Output**                  Splitting the data into train and test

## Data Science – Machine Learning – K Fold Cross Validation

**Program Name**

Applying logistic regression

demo3.py

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

digits = load_digits()
X_train, X_test, y_train, y_test = train_test_split(digits.data,
digits.target, test_size=0.3)

lr = LogisticRegression(solver = 'lbfgs', max_iter = 3000)
lr.fit(X_train, y_train)
print(lr.score(X_test, y_test))
```

**Output**

0.95

## Data Science – Machine Learning – K Fold Cross Validation

**Program Name** Applying SVM algorithm  
demo4.py

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

digits = load_digits()
X_train, X_test, y_train, y_test = train_test_split(digits.data,
digits.target, test_size=0.3)

svm = SVC()
svm.fit(X_train, y_train)

print(svm.score(X_test, y_test))
```

**Output**  
0.9907407407407407

## Data Science – Machine Learning – K Fold Cross Validation

**Program Name** Applying Random forest algorithm  
demo5.py

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

digits = load_digits()
X_train, X_test, y_train, y_test = train_test_split(digits.data,
digits.target, test_size=0.3)

rf = RandomForestClassifier(n_estimators=40)
rf.fit(X_train, y_train)

print(rf.score(X_test, y_test))
```

**Output**  
0.975925925925926

## Data Science – Machine Learning – K Fold Cross Validation

**Program Name** K fold cross validation  
demo6.py

```
from sklearn.model_selection import KFold  
  
kf = KFold(n_splits=3)  
  
print(kf)
```

**Output**  
KFold(n\_splits=3, random\_state=None, shuffle=False)

**Program Name** K fold cross validation: Example  
demo7.py

```
from sklearn.model_selection import KFold  
  
kf = KFold(n_splits=3)  
  
for train_index, test_index in kf.split([1,2,3,4,5,6,7,8,9]):  
    print(train_index, test_index)
```

**Output**

```
[3 4 5 6 7 8] [0 1 2]  
[0 1 2 6 7 8] [3 4 5]  
[0 1 2 3 4 5] [6 7 8]
```

## Data Science – Machine Learning – K Fold Cross Validation

**Program Name** Logistic regression model performance using cross\_val\_score  
demo8.py

```
from sklearn.model_selection import cross_val_score
from sklearn.datasets import load_digits
from sklearn.linear_model import LogisticRegression

digits = load_digits()

a = LogisticRegression(solver = 'lbfgs', max_iter = 5000)
scores = cross_val_score(a, digits.data, digits.target, cv=3)

print(scores)
```

**Output**

```
[0.92153589 0.94156928 0.91652755]
```

## Data Science – Machine Learning – K Fold Cross Validation

**Program Name** SVM model performance using cross\_val\_score  
demo9.py

```
from sklearn.model_selection import cross_val_score
from sklearn.datasets import load_digits
from sklearn.svm import SVC

digits = load_digits()

b = SVC()
scores = cross_val_score(b, digits.data, digits.target, cv=3)

print(scores)
```

**Output**  
[0.92153589 0.94156928 0.91652755]

## Data Science – Machine Learning – K Fold Cross Validation

**Program Name** Random forest model performance using cross\_val\_score  
demo10.py

```
from sklearn.model_selection import cross_val_score
from sklearn.datasets import load_digits
from sklearn.ensemble import RandomForestClassifier

digits = load_digits()

c = RandomForestClassifier(n_estimators=40)
scores = cross_val_score(c, digits.data, digits.target, cv=3)

print(scores)
```

**Output**  
[0.93823038 0.94156928 0.92821369]

## Data Science – Machine Learning – K Fold Cross Validation

**Program Name**

Checking average of all model scores

demo11.py

```
from sklearn.model_selection import cross_val_score
from sklearn.datasets import load_digits
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
import numpy as np

digits = load_digits()

a = LogisticRegression(solver = 'lbfgs', max_iter = 5000)
b = SVC()
c = RandomForestClassifier(n_estimators=40)

scores1 = cross_val_score(a, digits.data, digits.target, cv=3)
scores2 = cross_val_score(b, digits.data, digits.target, cv=3)
scores3 = cross_val_score(c, digits.data, digits.target, cv=3)

print(np.average(scores1))
print(np.average(scores2))
print(np.average(scores3))
```

**Output**

```
0.9265442404006677
0.9699499165275459
0.9315525876460767
```

## 25. Data Science – Machine Learning – Support Vector Machine

### Contents

1. Support Vector Machine .....	2
2. Decision boundary.....	2
3. Why SVM name is SVM.....	2
4. Usage .....	2
5. Types of SVM .....	2
6. Linear SVM.....	3
7. Non-linear SVM .....	3
8. 2 features forms straight line & 3 features forms plane.....	3
9. How does SVM works?.....	4
10. Hyperplane and Support vectors.....	5
11. Use case .....	6

**25. Data Science – Machine Learning – Support Vector Machine****1. Support Vector Machine**

- ✓ Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms.
- ✓ It is used for Classification as well as Regression problems.
- ✓ Mostly used for Classification problems in Machine Learning.

**2. Decision boundary**

- ✓ The goal of the SVM algorithm is to create the best line or decision boundary that can separate n-dimensional space into classes.
- ✓ This best decision boundary is called a hyperplane.

**3. Why SVM name is SVM**

- ✓ SVM chooses the extreme points/vectors that help in creating the hyperplane.
- ✓ These extreme cases are called as support vectors; hence this algorithm is called as Support Vector Machine.

**4. Usage**

- ✓ SVM algorithm can be used for,
  - Face detection
  - Image classification
  - Text categorization

**5. Types of SVM**

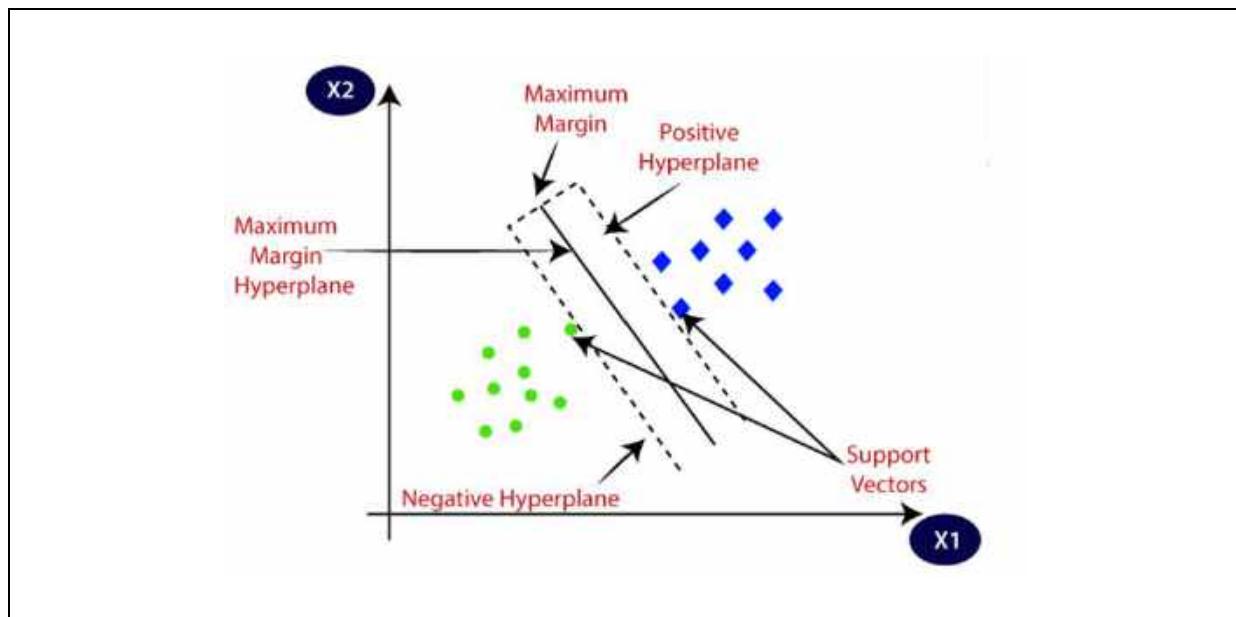
- ✓ Linear SVM
- ✓ Non-linear SVM

## 6. Linear SVM

- ✓ If the dataset can be classified into two classes by using a single straight line, then that is called as linearly separable data, and classifier is used called as Linear SVM classifier.

## 7. Non-linear SVM

- ✓ If the data set cannot be classified by using a straight line, then that is called as non-linear separable data, and classifier is used called as Non-linear SVM classifier.



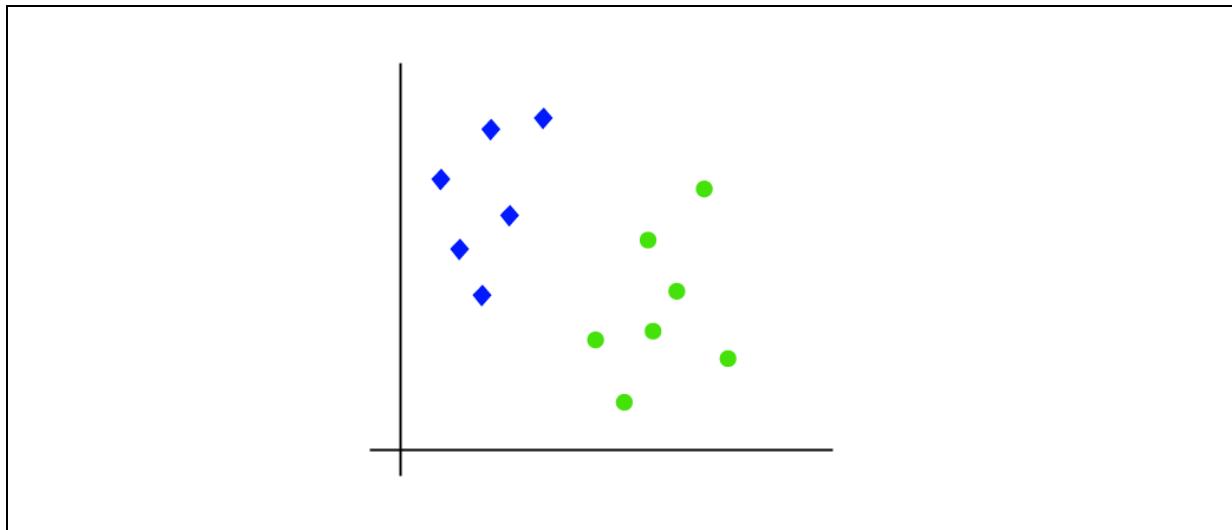
## 8. 2 features forms straight line & 3 features forms plane

- ✓ The dimensions of the hyperplane depend on the features present in the dataset,
  - If there are **2 features** then hyperplane will be a **straight line**
  - If there are **3 features** then hyperplane will be a **2-dimension plane**

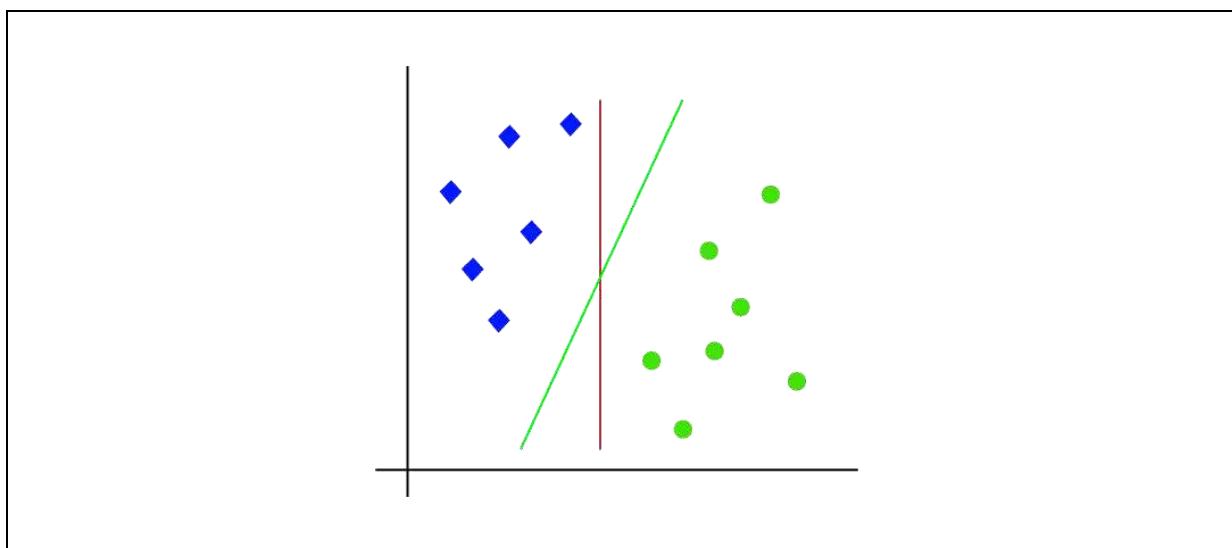
## 9. How does SVM works?

### Linear SVM

- ✓ Assuming that we have a dataset that has two tags (green and blue), and the dataset has two features  $x_1$  and  $x_2$ .
- ✓ We want a classifier that can classify the pair  $(x_1, x_2)$  of coordinates in either green or blue.

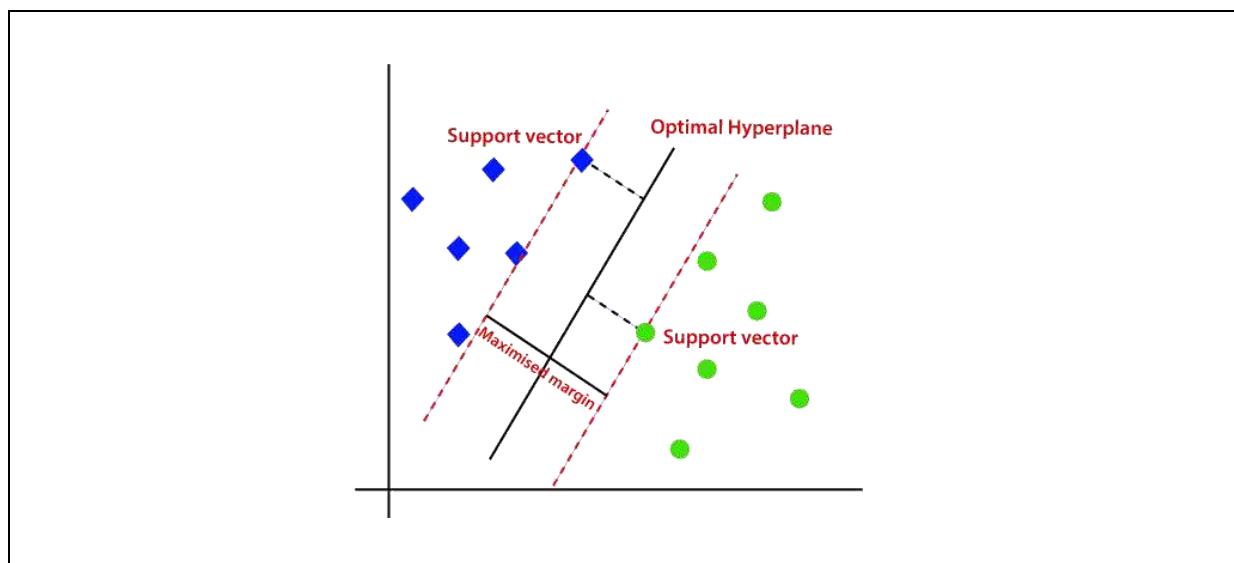


- ✓ Though it is 2-d space we can use straight line to separate these classes
- ✓ There can be multiple lines that can separate these classes too



## 10. Hyperplane and Support vectors

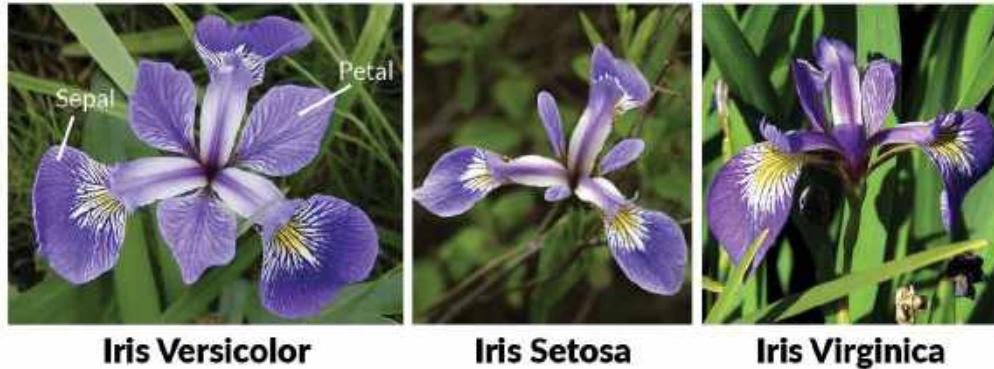
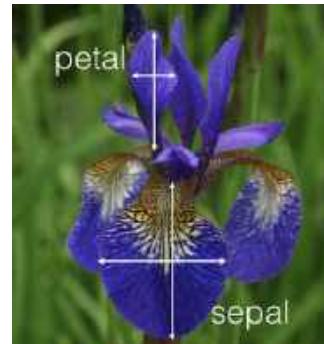
- ✓ SVM algorithm helps to find the best line or decision boundary, this best boundary or region is called as a **hyperplane**.
- ✓ SVM algorithm finds the closest point of the lines from both the classes; these points are called **support vectors**.
- ✓ The **distance between** the vectors and the hyperplane is called as **margin**.
- ✓ The **goal** of SVM is to maximize this margin.
- ✓ The hyperplane with maximum margin is called the **optimal hyperplane**.



## 11. Use case

- ✓ Assuming that Abhi had a hobby which is interested in distinguishing the species of some iris flowers that he has found
- ✓ He has collected some measurements associated with each iris, which are:
  - The **length** and **width** of the **petals**
  - The **length** and **width** of the **sepals**, all measured in centimetres.
- ✓ She also has the measurements of some irises that have been previously identified to the species
  - **setosa**,
  - **versicolor**
  - **virginica**
- ✓ The goal is to create a machine learning model that can learn from the measurements of these irises whose species are already known.
- ✓ So that we can predict the species for the new irises that she has found.

## Data Science – Machine Learning – Support Vector Machine



### Flower codes

- ✓ Setosa - 0
- ✓ Versicolor - 1
- ✓ Virginica - 2

## Data Science – Machine Learning – Support Vector Machine

**Program Name** Loading iris dataset  
demo1.py

```
from sklearn.datasets import load_iris

iris = load_iris()

print(dir(iris))
```

**Output**

```
['DESCR', 'data', 'feature_names', 'filename', 'frame', 'target',
'target_names']
```

**Program Name** Displaying feature names  
demo2.py

```
from sklearn.datasets import load_iris

iris = load_iris()

print(iris.feature_names)
```

**Output**

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal
width (cm)']
```

## Data Science – Machine Learning – Support Vector Machine

---

**Program Name** Displaying target names  
demo3.py

```
from sklearn.datasets import load_iris

iris = load_iris()

print(iris.target_names)
```

**Output**

```
['setosa' 'versicolor' 'virginica']
```

## Data Science – Machine Learning – Support Vector Machine

Program Name      Displaying data  
demo4.py

```
from sklearn.datasets import load_iris

iris = load_iris()

print(iris.data)
```

### Output

```
[[5.1 3.5 1.4 0.2]
 [4.9 3. 1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5. 3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5. 3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3. 1.4 0.1]
 [4.3 3. 1.1 0.1]]
```

## Data Science – Machine Learning – Support Vector Machine

---

**Program Name** Length of the data  
demo5.py

```
from sklearn.datasets import load_iris

iris = load_iris()

print(len(iris.data))
```

**Output**  
150

## Data Science – Machine Learning – Support Vector Machine

**Program Name** Create a Dataframe by using data and features  
demo6.py

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

print(df)
```

### Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
..	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

[150 rows x 4 columns]

## Data Science – Machine Learning – Support Vector Machine

**Program Name** Adding target column to the dataframe  
demo7.py

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

print(df.head())
```

**Output**

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

## Data Science – Machine Learning – Support Vector Machine

**Program Name** Displaying target == 0 flowers  
demo8.py

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

print(df[df.target==0].head())
```

**Output**

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

## Data Science – Machine Learning – Support Vector Machine

**Program Name** Displaying length of the target == 0 flowers  
demo9.py

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

print(len(df[df.target==0]))
```

**Output**

50

## Data Science – Machine Learning – Support Vector Machine

**Program Name** Displaying target == 1 flowers  
demo10.py

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

print(df[df.target==1].head())
```

**Output**

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
50	7.0	3.2	4.7	1.4	1
51	6.4	3.2	4.5	1.5	1
52	6.9	3.1	4.9	1.5	1
53	5.5	2.3	4.0	1.3	1
54	6.5	2.8	4.6	1.5	1

## Data Science – Machine Learning – Support Vector Machine

**Program Name** Displaying length of the target == 0 flowers  
demo11.py

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

print(len(df[df.target==1]))
```

**Output**

50

## Data Science – Machine Learning – Support Vector Machine

**Program Name** Displaying target == 2 flowers  
demo12.py

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

print(df[df.target==2].head())
```

### Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
100	6.3	3.3	6.0	2.5	2
101	5.8	2.7	5.1	1.9	2
102	7.1	3.0	5.9	2.1	2
103	6.3	2.9	5.6	1.8	2
104	6.5	3.0	5.8	2.2	2

## Data Science – Machine Learning – Support Vector Machine

**Program Name** Displaying length of the target == 2 flowers  
demo13.py

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

print(len(df[df.target==2]))
```

**Output**

50

## Data Science – Machine Learning – Support Vector Machine

---

**Program Name** Displaying the flower names  
demo14.py

```

import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

df['flower_name'] = df.target.apply(lambda x: iris.target_names[x])
print(df)

```

**Output**

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa
..	...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2	virginica
146	6.3	2.5	5.0	1.9	2	virginica
147	6.5	3.0	5.2	2.0	2	virginica
148	6.2	3.4	5.4	2.3	2	virginica
149	5.9	3.0	5.1	1.8	2	virginica

[150 rows x 6 columns]

## Data Science – Machine Learning – Support Vector Machine

**Program Name** All setosa flowers  
demo15.py

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

setosa_50 = df[:50]
print(setosa_50.head())
```

### Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa

## Data Science – Machine Learning – Support Vector Machine

**Program Name** All versicolor flowers  
demo16.py

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

versicolor_50 = df[50:100]
print(versicolor_50.head())
```

### Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
50	7.0	3.2	4.7	1.4	1	versicolor
51	6.4	3.2	4.5	1.5	1	versicolor
52	6.9	3.1	4.9	1.5	1	versicolor
53	5.5	2.3	4.8	1.3	1	versicolor
54	6.5	2.8	4.6	1.5	1	versicolor

## Data Science – Machine Learning – Support Vector Machine

---

**Program Name** All virginica flowers  
demo17.py

```

import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

virginica_50 = df[100:]
print(virginica_50.head())

```

### Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
100	6.3	3.3	6.0	2.5	2	virginica
101	5.8	2.7	5.1	1.9	2	virginica
102	7.1	3.0	5.9	2.1	2	virginica
103	6.3	2.9	5.6	1.8	2	virginica
104	6.5	3.0	5.8	2.2	2	virginica

## Data Science – Machine Learning – Support Vector Machine

Program Name

All types of flowers  
demo18.py

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

df0 = df[:50]
df1 = df[50:100]
df2 = df[100:]

print("All types of flowers")
```

Output

All types of flowers

## Data Science – Machine Learning – Support Vector Machine

Program  
Name

Plotting  
demo19.py

```
import pandas as pd
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

df0 = df[:50]
df1 = df[50:100]
df2 = df[100:]

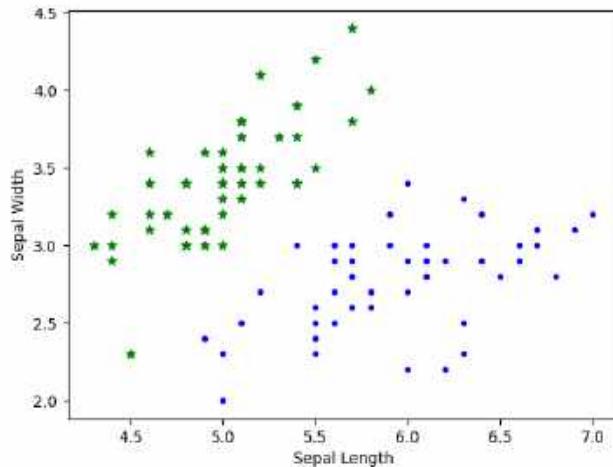
# Sepal length vs Sepal Width (Setosa vs Versicolor)

plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')

plt.scatter(df0['sepal length (cm)'], df0['sepal width (cm)'],
color="green", marker='*')

plt.scatter(df1['sepal length (cm)'], df1['sepal width (cm)'],
color="blue", marker='.')

plt.show()
```

**Output**

## Data Science – Machine Learning – Support Vector Machine

Program  
Name

Plotting  
demo20.py

```
import pandas as pd
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

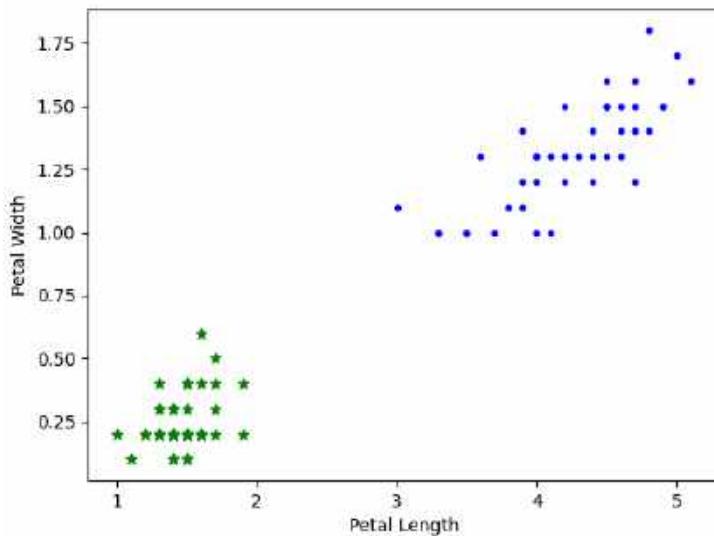
df0 = df[:50]
df1 = df[50:100]
df2 = df[100:]

# Petal length vs Petal Width (Setosa vs Versicolor)

plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.scatter(df0['petal length (cm)'], df0['petal width (cm)'],
color="green", marker='*')
plt.scatter(df1['petal length (cm)'], df1['petal width (cm)'],
color="blue", marker='.')

plt.show()
```

## Output



## Data Science – Machine Learning – Support Vector Machine

**Program Name** Splitting the data  
demo21.py

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

X = df.drop(['target', 'flower_name'], axis='columns')
y = df.target

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)

print("Splitting the data")
```

**Output**

Splitting the data

## Data Science – Machine Learning – Support Vector Machine

**Program Name** Model training  
demo22.py

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

X = df.drop(['target', 'flower_name'], axis='columns')
y = df.target

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)

# Train Using Support Vector Machine (SVM)

model = SVC()
model.fit(X_train, y_train)
print("Model got trained")
```

### Output

Model got trained

## Data Science – Machine Learning – Support Vector Machine

Program  
Name

Model score  
demo23.py

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

X = df.drop(['target', 'flower_name'], axis='columns')
y = df.target

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)

# Train Using Support Vector Machine (SVM)

model = SVC()
model.fit(X_train, y_train)

print(model.score(X_test, y_test))
```

Output

0.9666666666666667

## Data Science – Machine Learning – Support Vector Machine

Program Name

Model prediction  
demo24.py

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

X = df.drop(['target', 'flower_name'], axis='columns')
y = df.target

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)

# Train Using Support Vector Machine (SVM)

model = SVC()
model.fit(X_train, y_train)

print(model.predict([[4.8,3.0,1.5,0.3]]))
```

Output

[0]



## Data Science – Machine Learning – Underfitting and Overfitting

---

### 26. Data Science – Machine Learning – Underfitting and Overfitting

#### Contents

1. Overfitting & Underfitting .....	2
2. Noise .....	2
3. Bias.....	2
4. Variance .....	2
5. Overfitting .....	3
6. Underfitting .....	4
7. Good fit model.....	4
8. Good example .....	5

**26. Data Science – Machine Learning – Underfitting and Overfitting****1. Overfitting & Underfitting**

- ✓ The main goal of every machine learning model is to **generalize** well.
- ✓ A generalized model provides a suitable output on unknown dataset
  - This means after providing training on the dataset, it can produce reliable and accurate output.
- ✓ Overfitting and Underfitting are the two main problems that occur in machine learning, because of these ML model performances may impact to reduce.

**2. Noise**

- ✓ Noise means irrelevant data; it reduces the performance of the model.

**3. Bias**

- ✓ Difference between the predicted values and the actual values.

**4. Variance**

- ✓ Machine learning model performs well with the training dataset, but does not perform well with the test dataset.

## Data Science – Machine Learning – Underfitting and Overfitting

### 5. Overfitting

- ✓ Overfitting is the scenario where a machine learning model cannot generalize or fit well on unseen dataset.
- ✓ The over fitted model has **low bias** and **high variance**.



#### Note:

- ✓ We can avoid the Overfitting in Model by using cross-Validation, Training with more data, removing features, regularization, Ensemble

## Data Science – Machine Learning – Underfitting and Overfitting

---

### 6. Underfitting

- ✓ During training if model unable to learn properly then this is called as Underfitting.
- ✓ So it reduces the accuracy and produces unreliable predictions.
- ✓ The under fitted model has **high bias** and **low variance**.

### 7. Good fit model

- ✓ If model predicts well on training dataset and unseen dataset then it's called as good fit model

## Data Science – Machine Learning – Underfitting and Overfitting

---

### 8. Good example

- ✓ Assuming that there are three students have prepared for a mathematics examination.
  
- ✓ First student:
  - Prepared only Addition operations and skipped other math operations from textbook[X]
  
- ✓ Second student
  - Prepared all math operations from textbook[X]
  
- ✓ Third student
  - Prepared all math operations from textbook[X].
  - Practiced more on new topics from other math text books[Y, Z]

### During exam

- ✓ First student:
  - He can answer only for addition related questions.
  
- ✓ Second student
  - He can answer to the questions which are from only textbook[X]
  
- ✓ Third student
  - He can answer to the questions which are from textbook[X, Y, Z]

### Comparison

✓ First student	-	Under fitting
✓ Second student	-	Over fitting
✓ Third student	-	Good fit

## 27. Data Science – Machine Learning – Lasso & Ridge Regression

### Contents

1. Linear Regression .....	2
2. Lasso Regression.....	2
3. Coefficients can be large .....	2
4. L1 penalty .....	2
5. L1 Regularization .....	3
6. How to avoid overfitting issue? .....	3
7. L1 Regularization and L2 Regularization.....	3
8. Dataset Details: Melbourne house sale price .....	6
9. Ridge Regression .....	34
10. Coefficients can be large .....	34
11. L2 penalty .....	34

## 27. Data Science – Machine Learning – Lasso & Ridge Regression

### 1. Linear Regression

- ✓ Linear Regression is a standard algorithm for regression and it is used to explain the relationship between two variables.
- ✓ Also called as it's a relationship between dependent variable and one or more independent variables.

### 2. Lasso Regression

- ✓ In linear regression with a single input variable, this relationship is a line, and with **higher dimensions**, this relationship can be hyperplane that connects the input variables to the target variable.
- ✓ The coefficients of the model are found via an optimization process which minimizes error.

### 3. Coefficients can be large

- ✓ A problem with linear regression, the estimated coefficients of the model can become large and may become model unstable
- ✓ One approach to address the stability of regression models is to change the loss function to include additional costs for a model that has large coefficients.

### 4. L1 penalty

- ✓ **Lasso Regression** is a popular type of regularized linear regression that includes an L1 penalty.
- ✓ It penalizes a model based on the **sum of the absolute coefficient values**. This is called the L1 penalty.
- ✓ An L1 penalty minimizes the size of all coefficients and some coefficients to be minimized to the value zero, effectively removing input features from the model.

## 5. L1 Regularization

- ✓ A regression model that uses **L1** regularization technique is called **Lasso** Regression.
- ✓ Lasso full form is, Least Absolute Shrinkage and Selection Operator

- ✓ Minimization objective = LS Obj +  $\alpha * (\text{sum of the absolute value of coefficients})$

## 6. How to avoid overfitting issue?

- ✓ Regularization is an important concept that is used to avoid overfitting of the data,
- ✓ We can address this issue by using L1 and L2 Regularization
- ✓ Regularization is implemented by adding a “penalty” term to the best fit derived from the trained data, to achieve a *lesser variance*

## 7. L1 Regularization and L2 Regularization

- ✓ When you have a large number of features in your dataset, some of the Regularization techniques used to address over-fitting.

## Data Science – Machine Learning – Lasso & Ridge Regression

**Program Name** Importing required libraries  
demo1.py

```
import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

print("Importing required libraries")
```

**Output**

Importing required libraries

## Data Science – Machine Learning – Lasso & Ridge Regression

Program Name Loading the dataset  
demo2.py

```
import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')
print(dataset.head())
```

### Output

```
Suburb      Address  Rooms  ... Longitude    Regionname Propertycount
0 Abbotsford  68 Studley St    2  ...  144.9958 Northern Metropolitan  4019.0
1 Abbotsford  85 Turner St    2  ...  144.9984 Northern Metropolitan  4019.0
2 Abbotsford  25 Bloomberg St  2  ...  144.9934 Northern Metropolitan  4019.0
3 Abbotsford  18/659 Victoria St  3  ...  145.8116 Northern Metropolitan  4019.0
4 Abbotsford  5 Charles St     3  ...  144.9944 Northern Metropolitan  4019.0

[5 rows x 21 columns]
```

## 8. Dataset Details: Melbourne house sale price

- ✓ The dataset is about the housing market in Melbourne and contains information about the house sale price
- ✓ Notes on Specific Variables
  - Rooms: Number of rooms
  - Price: Price in dollars
  - Method: S - property sold; SP - property sold prior; PI - property passed in; PN - sold prior not disclosed; SN - sold not disclosed; NB - no bid; VB - vendor bid; W - withdrawn prior to auction; SA - sold after auction; SS - sold after auction price not disclosed. N/A - price or highest bid not available.
  - Type: br - bedroom(s); h - house, cottage, villa, semi, terrace; u - unit, duplex; t - townhouse; dev site - development site; o res - other residential.
  - SellerG: Real Estate Agent
  - Date: Date sold
  - Distance: Distance from CBD
  - Region name: General Region (West, North West, North, North east ...etc)
  - Property count: Number of properties that exist in the suburb (an outlying district of a city, especially a residential one.).
  - Bedroom2 : Scrapped # of Bedrooms (from different source)
  - Bathroom: Number of Bathrooms
  - Car: Number of cars/pots
  - Landsize: Land Size
  - BuildingArea: Building Size
  - Council Area: Governing council for the area

## Data Science – Machine Learning – Lasso & Ridge Regression

**Program Name** Rows and columns in DataFrame  
demo3.py

```
import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')
print("Rows and columns:", dataset.shape)
```

**Output**

Rows and columns: (34857, 21)

## Data Science – Machine Learning – Lasso & Ridge Regression

Program Name Unique values  
demo4.py

```
import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')
print(dataset.nunique())
```

### Output

```
Suburb          351
Address         34009
Rooms           12
Type            3
Price           2871
Method          9
SellerG         388
Date             78
Distance        215
Postcode        211
Bedroom2        15
Bathroom        11
Car              15
Landsize        1684
BuildingArea    740
YearBuilt       160
CouncilArea     33
Latitude        13402
Longitude       14524
Regionname      8
Propertycount   342
dtype: int64
```

## Data Science – Machine Learning – Lasso & Ridge Regression

---

**Program Name** Get the required columns  
demo5.py

```
import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')
print(dataset.shape)

cols_to_use = ['Suburb', 'Rooms', 'Type', 'Method', 'SellerG',
               'Regionname', 'Propertycount', 'Distance', 'CouncilArea',
               'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'Price']

dataset = dataset[cols_to_use]

print(dataset.head())
```

### Output

	Suburb	Rooms	Type	Method	SellerG	...	Bathroom	Car	Landsize	BuildingArea	Price
0	Abbotsford	2	h	SS	Jellis	...	1.0	1.0	126.0	NaN	NaN
1	Abbotsford	2	h	S	Biggin	...	1.0	1.0	202.0	NaN	1480000.0
2	Abbotsford	2	h	S	Biggin	...	1.0	0.0	156.0	79.0	1035000.0
3	Abbotsford	3	u	VB	Rounds	...	2.0	1.0	0.0	NaN	NaN
4	Abbotsford	3	h	SP	Biggin	...	2.0	0.0	134.0	150.0	1465000.0

[5 rows x 15 columns]

## Data Science – Machine Learning – Lasso & Ridge Regression

**Program Name** Rows and columns in DataFrame  
demo6.py

```
import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')

cols_to_use = ['Suburb', 'Rooms', 'Type', 'Method', 'SellerG',
'Regionname', 'Propertycount', 'Distance', 'CouncilArea',
'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'Price']

dataset = dataset[cols_to_use]
print("Rows and columns:", dataset.shape)
```

**Output**

Rows and columns: (34857, 15)

## Data Science – Machine Learning – Lasso & Ridge Regression

**Program Name** Checking NaN values  
demo7.py

```
import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')

cols_to_use = ['Suburb', 'Rooms', 'Type', 'Method', 'SellerG',
'Regionname', 'Propertycount', 'Distance', 'CouncilArea',
'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'Price']

dataset = dataset[cols_to_use]

print(dataset.isna().sum())
```

**Output**

```
Suburb          0
Rooms           0
Type            0
Method          0
SellerG         0
Regionname      3
Propertycount   3
Distance        1
CouncilArea     3
Bedroom2        8217
Bathroom        8226
Car              8728
Landsize        11810
BuildingArea    21115
Price            7610
dtype: int64
```

## Data Science – Machine Learning – Lasso & Ridge Regression

---

**Program Name** Few of the columns filling with zero  
demo8.py

```

import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')

cols_to_use = ['Suburb', 'Rooms', 'Type', 'Method', 'SellerG',
'Regionname', 'Propertycount', 'Distance', 'CouncilArea',
'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'Price']

dataset = dataset[cols_to_use]

cols_to_fill_zero = ['Propertycount', 'Distance', 'Bedroom2',
'Bathroom', 'Car']
dataset[cols_to_fill_zero] = dataset[cols_to_fill_zero].fillna(0)

print(dataset.isna().sum())

```

### Output

Suburb	0
Rooms	0
Type	0
Method	0
SellerG	0
Regionname	3
Propertycount	0
Distance	0
CouncilArea	3
Bedroom2	0
Bathroom	0
Car	0
Landsize	11810
BuildingArea	21115
Price	7610
dtype:	int64

## Data Science – Machine Learning – Lasso & Ridge Regression

**Program Name** Filling Landsize and BuildingArea columns with mean value  
demo9.py

```
import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')

cols_to_use = ['Suburb', 'Rooms', 'Type', 'Method', 'SellerG',
'Regionname', 'Propertycount', 'Distance', 'CouncilArea',
'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'Price']

dataset = dataset[cols_to_use]

cols_to_fill_zero = ['Propertycount', 'Distance', 'Bedroom2',
'Bathroom', 'Car']
dataset[cols_to_fill_zero] = dataset[cols_to_fill_zero].fillna(0)

dataset['Landsize'] =
dataset['Landsize'].fillna(dataset.Landsize.mean())

dataset['BuildingArea'] =
dataset['BuildingArea'].fillna(dataset.BuildingArea.mean())

print(dataset.isna().sum())
```

**Output**

```
Suburb          0
Rooms           0
Type            0
Method          0
SellerG         0
Regionname      3
Propertycount   0
Distance        0
CouncilArea     3
Bedroom2        0
Bathroom        0
Car              0
Landsize         0
BuildingArea    0
Price           7610
dtype: int64
```

## Data Science – Machine Learning – Lasso & Ridge Regression

Program Name

Dropping NaN values

demo10.py

```
import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')

cols_to_use = ['Suburb', 'Rooms', 'Type', 'Method', 'SellerG',
'Regionname', 'Propertycount', 'Distance', 'CouncilArea',
'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'Price']

dataset = dataset[cols_to_use]

cols_to_fill_zero = ['Propertycount', 'Distance', 'Bedroom2',
'Bathroom', 'Car']
dataset[cols_to_fill_zero] = dataset[cols_to_fill_zero].fillna(0)

dataset['Landsize'] =
dataset['Landsize'].fillna(dataset.Landsize.mean())

dataset['BuildingArea'] =
dataset['BuildingArea'].fillna(dataset.BuildingArea.mean())

dataset.dropna(inplace = True)

print(dataset.isna().sum())
```

## Data Science – Machine Learning – Lasso & Ridge Regression

### Output

```
Suburb      0
Rooms       0
Type        0
Method      0
SellerG     0
Regionname  0
Propertycount 0
Distance    0
CouncilArea 0
Bedroom2    0
Bathroom    0
Car         0
Landsize    0
BuildingArea 0
Price       0
dtype: int64
```

## Data Science – Machine Learning – Lasso & Ridge Regression

**Program Name** Creating dummy variables for characters data  
demo11.py

```
import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')

cols_to_use = ['Suburb', 'Rooms', 'Type', 'Method', 'SellerG',
'Regionname', 'Propertycount', 'Distance', 'CouncilArea',
'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'Price']

dataset = dataset[cols_to_use]

cols_to_fill_zero = ['Propertycount', 'Distance', 'Bedroom2',
'Bathroom', 'Car']
dataset[cols_to_fill_zero] = dataset[cols_to_fill_zero].fillna(0)

dataset['Landsize'] =
dataset['Landsize'].fillna(dataset.Landsize.mean())

dataset['BuildingArea'] =
dataset['BuildingArea'].fillna(dataset.BuildingArea.mean())

dataset.dropna(inplace = True)

dataset = pd.get_dummies(dataset, drop_first = True)

print(dataset.head())
```

## Data Science – Machine Learning – Lasso & Ridge Regression

### Output

```
Rooms  Propertycount ... CouncilArea_Yarra City Council CouncilArea_Yarra Ranges Shire Council
1      2          4019.0 ...
2      2          4019.0 ...
4      3          4019.0 ...
5      3          4019.0 ...
6      4          4019.0 ...
[5 rows x 745 columns]
```

## Data Science – Machine Learning – Lasso & Ridge Regression

---

**Program Name** Creating features and labels  
demo12.py

```

import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')

cols_to_use = ['Suburb', 'Rooms', 'Type', 'Method', 'SellerG',
'Regionname', 'Propertycount', 'Distance', 'CouncilArea',
'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'Price']

dataset = dataset[cols_to_use]

cols_to_fill_zero = ['Propertycount', 'Distance', 'Bedroom2',
'Bathroom', 'Car']
dataset[cols_to_fill_zero] = dataset[cols_to_fill_zero].fillna(0)

dataset['Landsize'] =
dataset['Landsize'].fillna(dataset.Landsize.mean())

dataset['BuildingArea'] =
dataset['BuildingArea'].fillna(dataset.BuildingArea.mean())

dataset.dropna(inplace = True)

dataset = pd.get_dummies(dataset, drop_first = True)

X = dataset.drop('Price', axis = 1)
y = dataset['Price']

print("Created features and labels")

```

**Output**

Created features and labels

## Data Science – Machine Learning – Lasso & Ridge Regression

**Program Name** Splitting training and testing datasets  
demo13.py

```
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')

cols_to_use = ['Suburb', 'Rooms', 'Type', 'Method', 'SellerG',
'Regionname', 'Propertycount', 'Distance', 'CouncilArea',
'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'Price']

dataset = dataset[cols_to_use]

cols_to_fill_zero = ['Propertycount', 'Distance', 'Bedroom2',
'Bathroom', 'Car']
dataset[cols_to_fill_zero] = dataset[cols_to_fill_zero].fillna(0)

dataset['Landsize'] =
dataset['Landsize'].fillna(dataset.Landsize.mean())

dataset['BuildingArea'] =
dataset['BuildingArea'].fillna(dataset.BuildingArea.mean())

dataset.dropna(inplace = True)

dataset = pd.get_dummies(dataset, drop_first = True)

X = dataset.drop('Price', axis = 1)
y = dataset['Price']
```

## Data Science – Machine Learning – Lasso & Ridge Regression

```
train_X, test_X, train_y, test_y = train_test_split(X, y, test_size =  
0.3, random_state=2)  
  
print("Splitting train and test datasets")
```

### Output

Splitting train and test datasets

**Program Name** Creating LinearRegression model and training demo14.py

```

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')

cols_to_use = ['Suburb', 'Rooms', 'Type', 'Method', 'SellerG',
'Regionname', 'Propertycount', 'Distance', 'CouncilArea',
'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'Price']

dataset = dataset[cols_to_use]

cols_to_fill_zero = ['Propertycount', 'Distance', 'Bedroom2',
'Bathroom', 'Car']
dataset[cols_to_fill_zero] = dataset[cols_to_fill_zero].fillna(0)

dataset['Landsize'] =
dataset['Landsize'].fillna(dataset.Landsize.mean())

dataset['BuildingArea'] =
dataset['BuildingArea'].fillna(dataset.BuildingArea.mean())

dataset.dropna(inplace = True)

dataset = pd.get_dummies(dataset, drop_first = True)

X = dataset.drop('Price', axis = 1)
y = dataset['Price']

train_X, test_X, train_y, test_y = train_test_split(X, y, test_size =

```

## Data Science – Machine Learning – Lasso & Ridge Regression

```
0.3, random_state=2)

reg = LinearRegression()

print("Created LinearRegression model and training")

reg.fit(train_X, train_y)
```

### Output

Created LinearRegression model

## Data Science – Machine Learning – Lasso & Ridge Regression

**Program Name** Creating LinearRegression model  
demo15.py

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')

cols_to_use = ['Suburb', 'Rooms', 'Type', 'Method', 'SellerG',
'Regionname', 'Propertycount', 'Distance', 'CouncilArea',
'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'Price']

dataset = dataset[cols_to_use]

cols_to_fill_zero = ['Propertycount', 'Distance', 'Bedroom2',
'Bathroom', 'Car']
dataset[cols_to_fill_zero] = dataset[cols_to_fill_zero].fillna(0)

dataset['Landsize'] =
dataset['Landsize'].fillna(dataset.Landsize.mean())

dataset['BuildingArea'] =
dataset['BuildingArea'].fillna(dataset.BuildingArea.mean())

dataset.dropna(inplace = True)

dataset = pd.get_dummies(dataset, drop_first = True)

X = dataset.drop('Price', axis = 1)
y = dataset['Price']
```

## Data Science – Machine Learning – Lasso & Ridge Regression

```
train_X, test_X, train_y, test_y = train_test_split(X, y, test_size =  
0.3, random_state=2)  
  
reg = LinearRegression()  
  
reg.fit(train_X, train_y)  
  
print("Training LinearRegression model")
```

### Output

Training LinearRegression model

**Program Name** Linear Regression: Training, training dataset score  
demo16.py

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')

cols_to_use = ['Suburb', 'Rooms', 'Type', 'Method', 'SellerG',
'Regionname', 'Propertycount', 'Distance', 'CouncilArea',
'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'Price']

dataset = dataset[cols_to_use]

cols_to_fill_zero = ['Propertycount', 'Distance', 'Bedroom2',
'Bathroom', 'Car']
dataset[cols_to_fill_zero] = dataset[cols_to_fill_zero].fillna(0)

dataset['Landsize'] =
dataset['Landsize'].fillna(dataset.Landsize.mean())

dataset['BuildingArea'] =
dataset['BuildingArea'].fillna(dataset.BuildingArea.mean())

dataset.dropna(inplace = True)

dataset = pd.get_dummies(dataset, drop_first = True)

X = dataset.drop('Price', axis = 1)
y = dataset['Price']
```

## Data Science – Machine Learning – Lasso & Ridge Regression

```
train_X, test_X, train_y, test_y = train_test_split(X, y, test_size =  
0.3, random_state=2)  
  
reg = LinearRegression()  
reg.fit(train_X, train_y)  
  
print("Training dataset score is:")  
print(reg.score(train_X, train_y))
```

### Output

```
Training dataset score is:  
0.6827792395792723
```

**Program Name** Linear Regression: Training, test dataset score  
demo17.py

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')

cols_to_use = ['Suburb', 'Rooms', 'Type', 'Method', 'SellerG',
'Regionname', 'Propertycount', 'Distance', 'CouncilArea',
'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'Price']

dataset = dataset[cols_to_use]

cols_to_fill_zero = ['Propertycount', 'Distance', 'Bedroom2',
'Bathroom', 'Car']
dataset[cols_to_fill_zero] = dataset[cols_to_fill_zero].fillna(0)

dataset['Landsize'] =
dataset['Landsize'].fillna(dataset.Landsize.mean())

dataset['BuildingArea'] =
dataset['BuildingArea'].fillna(dataset.BuildingArea.mean())

dataset.dropna(inplace = True)

dataset = pd.get_dummies(dataset, drop_first = True)

X = dataset.drop('Price', axis = 1)
y = dataset['Price']
```

## Data Science – Machine Learning – Lasso & Ridge Regression

```
train_X, test_X, train_y, test_y = train_test_split(X, y, test_size =  
0.3, random_state=2)  
  
reg = LinearRegression()  
reg.fit(train_X, train_y)  
  
print("Creating Linear Regression model and training with test  
dataset:")  
print(reg.score(test_X, test_y))
```

### Output

```
Creating Linear Regression model and training with test dataset  
0.1385368316157145
```

### Note

- ✓ If training score is very good and test score is very low then it called as over fit

## Data Science – Machine Learning – Lasso & Ridge Regression

Program Name

Lasso Regression

demo18.py

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso

import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')

cols_to_use = ['Suburb', 'Rooms', 'Type', 'Method', 'SellerG',
'Regionname', 'Propertycount', 'Distance', 'CouncilArea',
'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'Price']

dataset = dataset[cols_to_use]

cols_to_fill_zero = ['Propertycount', 'Distance', 'Bedroom2',
'Bathroom', 'Car']
dataset[cols_to_fill_zero] = dataset[cols_to_fill_zero].fillna(0)

dataset['Landsize'] =
dataset['Landsize'].fillna(dataset.Landsize.mean())

dataset['BuildingArea'] =
dataset['BuildingArea'].fillna(dataset.BuildingArea.mean())

dataset.dropna(inplace = True)

dataset = pd.get_dummies(dataset, drop_first = True)

X = dataset.drop('Price', axis = 1)
y = dataset['Price']
```

## Data Science – Machine Learning – Lasso & Ridge Regression

```
train_X, test_X, train_y, test_y = train_test_split(X, y, test_size =  
0.3, random_state=2)  
  
lasso_reg = Lasso(alpha = 50, max_iter = 100, tol = 0.1)  
lasso_reg.fit(train_X, train_y)  
  
print("Creating Lasso Regression model and training with train  
dataset")  
print(lasso_reg.score(train_X, train_y))
```

### Output

```
Creating Lasso Regression model and training with train dataset  
0.6766985624766824
```

**Program Name** Lasso Regression: Training, testing dataset score  
demo19.py

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso

import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')

cols_to_use = ['Suburb', 'Rooms', 'Type', 'Method', 'SellerG',
'Regionname', 'Propertycount', 'Distance', 'CouncilArea',
'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'Price']

dataset = dataset[cols_to_use]

cols_to_fill_zero = ['Propertycount', 'Distance', 'Bedroom2',
'Bathroom', 'Car']
dataset[cols_to_fill_zero] = dataset[cols_to_fill_zero].fillna(0)

dataset['Landsize'] =
dataset['Landsize'].fillna(dataset.Landsize.mean())

dataset['BuildingArea'] =
dataset['BuildingArea'].fillna(dataset.BuildingArea.mean())

dataset.dropna(inplace = True)

dataset = pd.get_dummies(dataset, drop_first = True)

X = dataset.drop('Price', axis = 1)
y = dataset['Price']
```

## Data Science – Machine Learning – Lasso & Ridge Regression

```
train_X, test_X, train_y, test_y = train_test_split(X, y, test_size =  
0.3, random_state=2)  
  
lasso_reg = Lasso(alpha = 50, max_iter = 100, tol = 0.1)  
lasso_reg.fit(train_X, train_y)  
  
print("Creating Lasso Regression model checking test dataset  
score")  
print(lasso_reg.score(test_X, test_y))
```

### Output

```
Creating Lasso Regression model checking test dataset score  
0.6636111369404489
```

## 9. Ridge Regression

- ✓ In linear regression with a single input variable, this relationship is a line, and with **higher dimensions**, this relationship can be hyperplane that connects the input variables to the target variable.
- ✓ The coefficients of the model are found via an optimization process which minimize error.

## 10. Coefficients can be large

- ✓ A problem with linear regression, the estimated coefficients of the model can become large and may become model unstable
- ✓ One approach to address the stability of regression models is to change the loss function to include additional costs for a model that has large coefficients.

## 11. L2 penalty

- ✓ **Ridge Regression** is a popular type of regularized linear regression that includes an **L2** penalty.
- ✓ It penalize a model based on **sum of the squared coefficient value**. This is called the L2 penalty.
- ✓ An L2 penalty minimizes the size of all coefficients and some coefficients to be minimized to the value zero, effectively removing input features from the model.

- ✓ Minimization objective = LS Obj +  $\alpha * (\text{sum of square of coefficients})$

## Data Science – Machine Learning – Lasso & Ridge Regression

**Program Name** Ridge Regression: Training, training dataset score  
demo20.py

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge

import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')

cols_to_use = ['Suburb', 'Rooms', 'Type', 'Method', 'SellerG',
'Regionname', 'Propertycount', 'Distance', 'CouncilArea',
'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'Price']

dataset = dataset[cols_to_use]

cols_to_fill_zero = ['Propertycount', 'Distance', 'Bedroom2',
'Bathroom', 'Car']
dataset[cols_to_fill_zero] = dataset[cols_to_fill_zero].fillna(0)

dataset['Landsize'] =
dataset['Landsize'].fillna(dataset.Landsize.mean())

dataset['BuildingArea'] =
dataset['BuildingArea'].fillna(dataset.BuildingArea.mean())

dataset.dropna(inplace = True)

dataset = pd.get_dummies(dataset, drop_first = True)

X = dataset.drop('Price', axis = 1)
y = dataset['Price']
```

## Data Science – Machine Learning – Lasso & Ridge Regression

```
train_X, test_X, train_y, test_y = train_test_split(X, y, test_size =  
0.3, random_state=2)  
  
ridge_reg = Ridge(alpha = 50, max_iter = 100, tol = 0.1)  
ridge_reg.fit(train_X, train_y)  
  
print("Ridge Regression model score with train dataset:")  
print(ridge_reg.score(train_X, train_y))
```

### Output

```
Ridge Regression model score with train dataset:  
0.6670848945194958
```

## Data Science – Machine Learning – Lasso & Ridge Regression

**Program Name** Ridge Regression: Training, testing dataset score  
demo21.py

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge

import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')

dataset = pd.read_csv('Melbourne_housing_FULL.csv')

cols_to_use = ['Suburb', 'Rooms', 'Type', 'Method', 'SellerG',
'Regionname', 'Propertycount', 'Distance', 'CouncilArea',
'Bedroom2', 'Bathroom', 'Car', 'Landsize', 'BuildingArea', 'Price']

dataset = dataset[cols_to_use]

cols_to_fill_zero = ['Propertycount', 'Distance', 'Bedroom2',
'Bathroom', 'Car']
dataset[cols_to_fill_zero] = dataset[cols_to_fill_zero].fillna(0)

dataset['Landsize'] =
dataset['Landsize'].fillna(dataset.Landsize.mean())

dataset['BuildingArea'] =
dataset['BuildingArea'].fillna(dataset.BuildingArea.mean())

dataset.dropna(inplace = True)

dataset = pd.get_dummies(dataset, drop_first = True)

X = dataset.drop('Price', axis = 1)
y = dataset['Price']
```

## Data Science – Machine Learning – Lasso & Ridge Regression

```
train_X, test_X, train_y, test_y = train_test_split(X, y, test_size =  
0.3, random_state=2)  
  
ridge_reg = Ridge(alpha=50, max_iter=100, tol=0.1)  
ridge_reg.fit(train_X, train_y)  
  
print("Ridge Regression model score with test dataset:")  
print(ridge_reg.score(test_X, test_y))
```

### Output

```
Ridge Regression model score with test dataset:  
0.6670848945194958
```

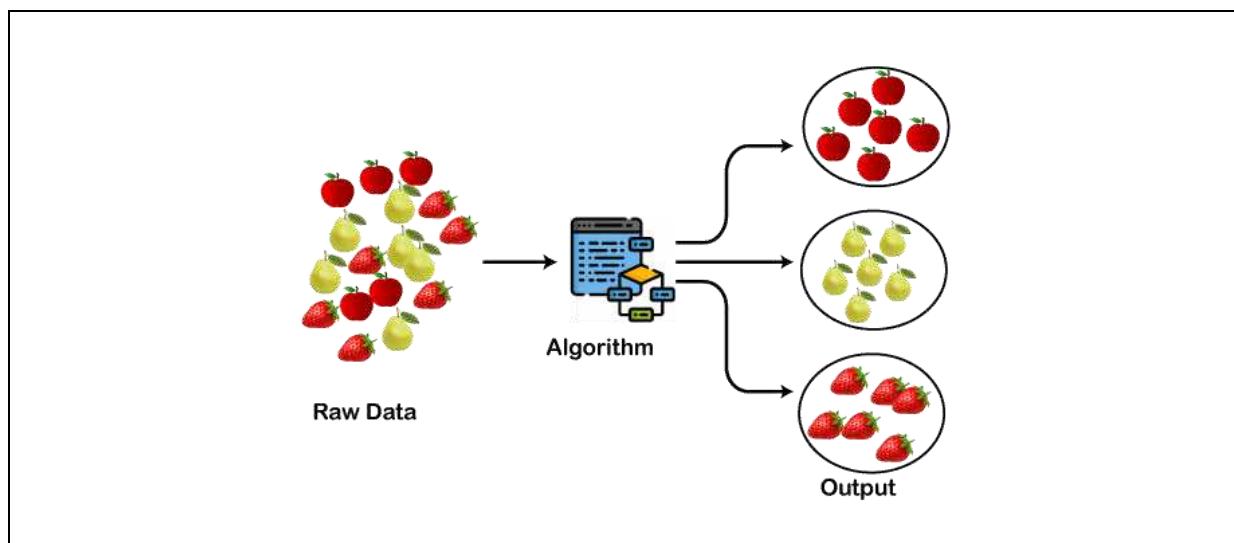
**28. Data Science – Machine Learning – K – Means Clustering****Contents**

<b>1. Clustering .....</b>	<b>2</b>
<b>2. K means clustering algorithm .....</b>	<b>2</b>
<b>3. Steps in K-Means Algorithm .....</b>	<b>3</b>
<b>4. Scenario .....</b>	<b>4</b>
<b>5. How to determine the correct number of clusters? .....</b>	<b>8</b>
<b>6. Elbow Method .....</b>	<b>10</b>

## 28. Data Science – Machine Learning – K – Means Clustering

### 1. Clustering

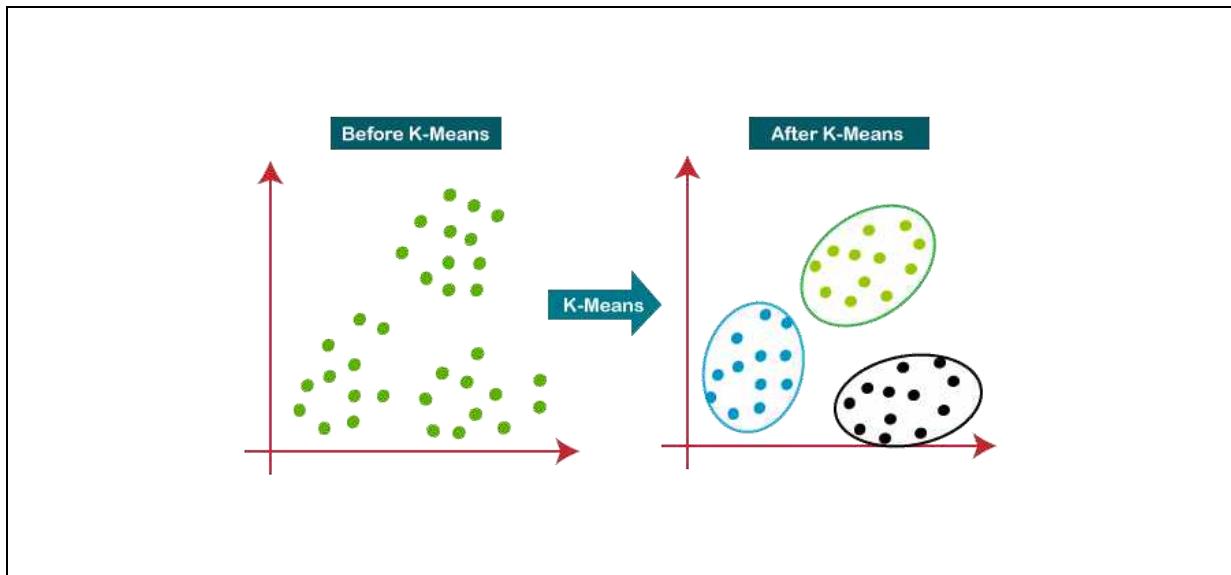
- ✓ Clustering or cluster analysis is a machine learning technique.
- ✓ It groups the unlabelled dataset.
- ✓ It is a way of grouping the data points into different clusters based on similarities.



### 2. K means clustering algorithm

- ✓ K-Means Clustering is an Unsupervised Learning algorithm.
- ✓ This algorithm groups the unlabeled dataset into different clusters based on similar properties.
- ✓ Here K defines the number of pre-defined clusters that need to be created in the process,
  - If  $K = 2$  then there will be two clusters,
  - If  $K = 3$  then there will be three clusters etc.
- ✓ It is a centroid-based algorithm, where each cluster is associated with a centroid.

## Data Science – Machine Learning – K – Means Clustering



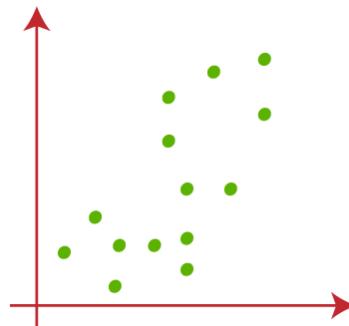
### 3. Steps in K-Means Algorithm

- ✓ Step-1: Select the number K to decide the number of clusters.
- ✓ Step-2: Select random K points or centroids.
- ✓ Step-3: Assign each data point to their closest centroid, which will form K clusters.
- ✓ Step-4: Calculate the variance and place a new centroid of each cluster.
- ✓ Step-5: Repeat the initial 3 steps, which mean reassign each data point to the new closest centroid of each cluster.
- ✓ Step-6: If any reassignment occurs, then go to step-4 else FINISH.
- ✓ Step-7: The model is ready.

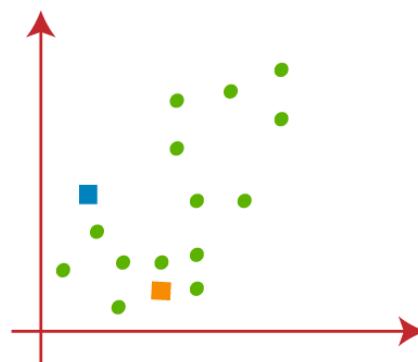
## Data Science – Machine Learning – K – Means Clustering

### 4. Scenario

- ✓ Assuming that we have scattered two variables in x and y axis

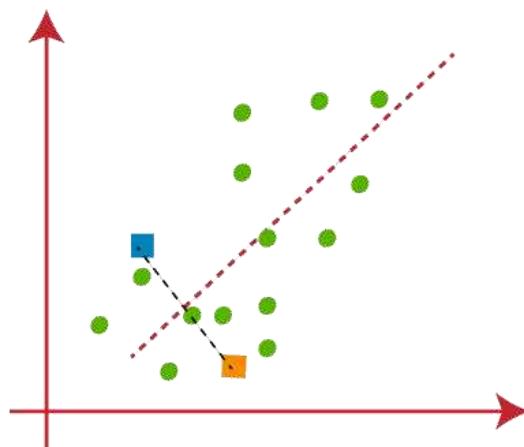


- ✓ Let's take some random k points or centroid to form the cluster.
- ✓ These points can be either the points from the dataset or any other point.
- ✓ So, here we are selecting the below two points as k points, which are not the part of our dataset.

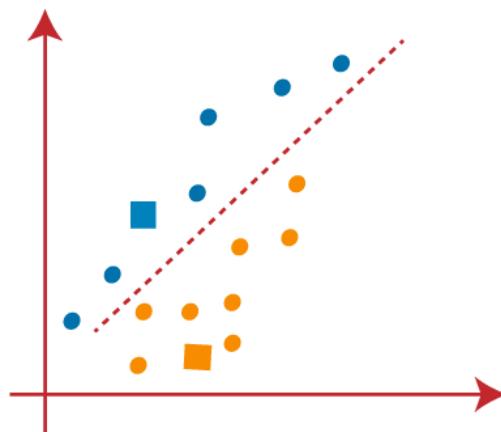


- ✓ Now we will assign each data point of the scatter plot to its closest K-point or centroid.
- ✓ Let's compute the distance between two points.
- ✓ So, we will draw a median between both the centroids.

## Data Science – Machine Learning – K – Means Clustering

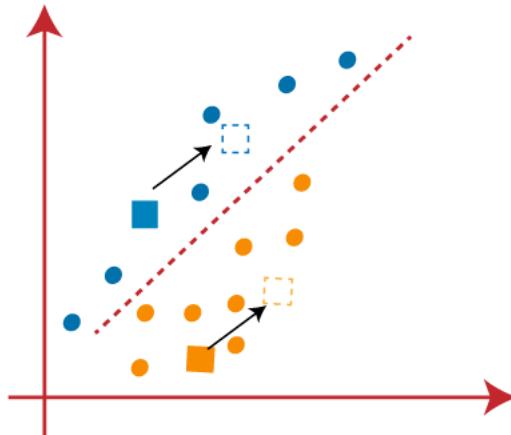


- ✓ From the above image, it is clear that points left side of the line is near to the K1 or blue centroid.
- ✓ Points to the right of the line are close to the yellow centroid.



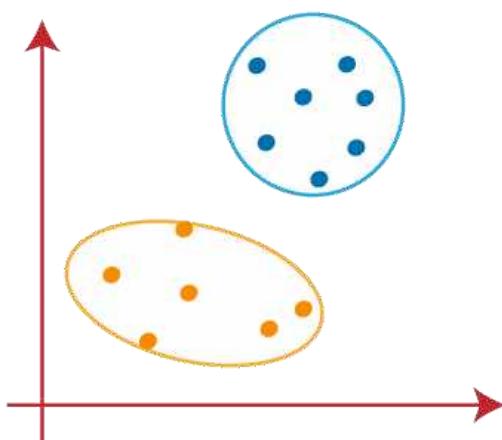
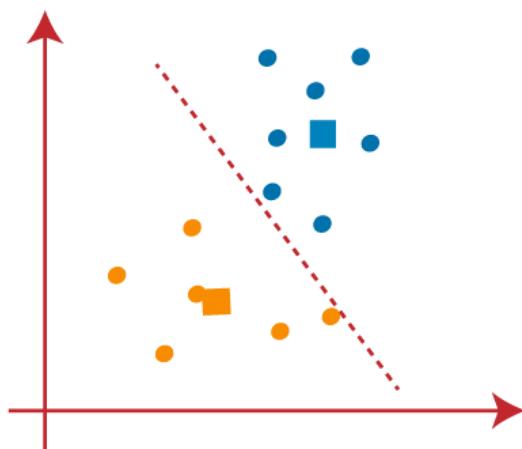
- ✓ As we need to find the closest cluster, so we will repeat the process by choosing a new centroid.
- ✓ To choose the new centroids, we will compute the center of gravity of these centroids, and will find new centroids.

## Data Science – Machine Learning – K – Means Clustering



- ✓ As we got the new centroids so again will draw the median line and reassign the data points.

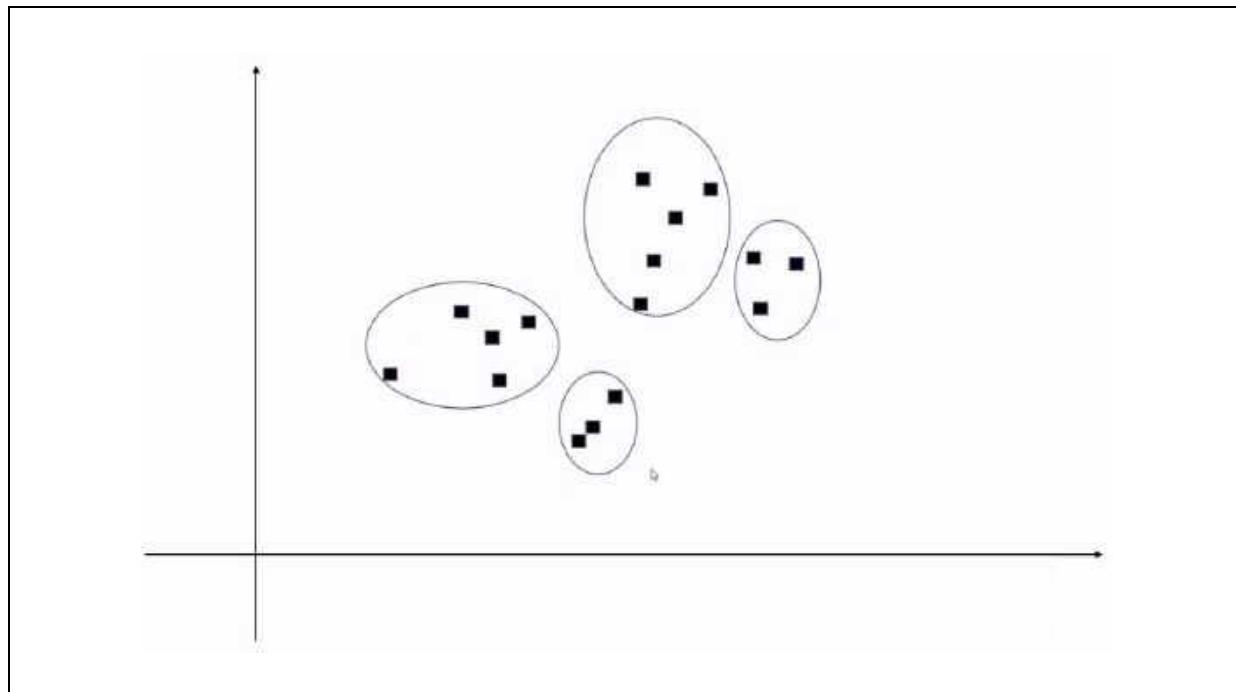
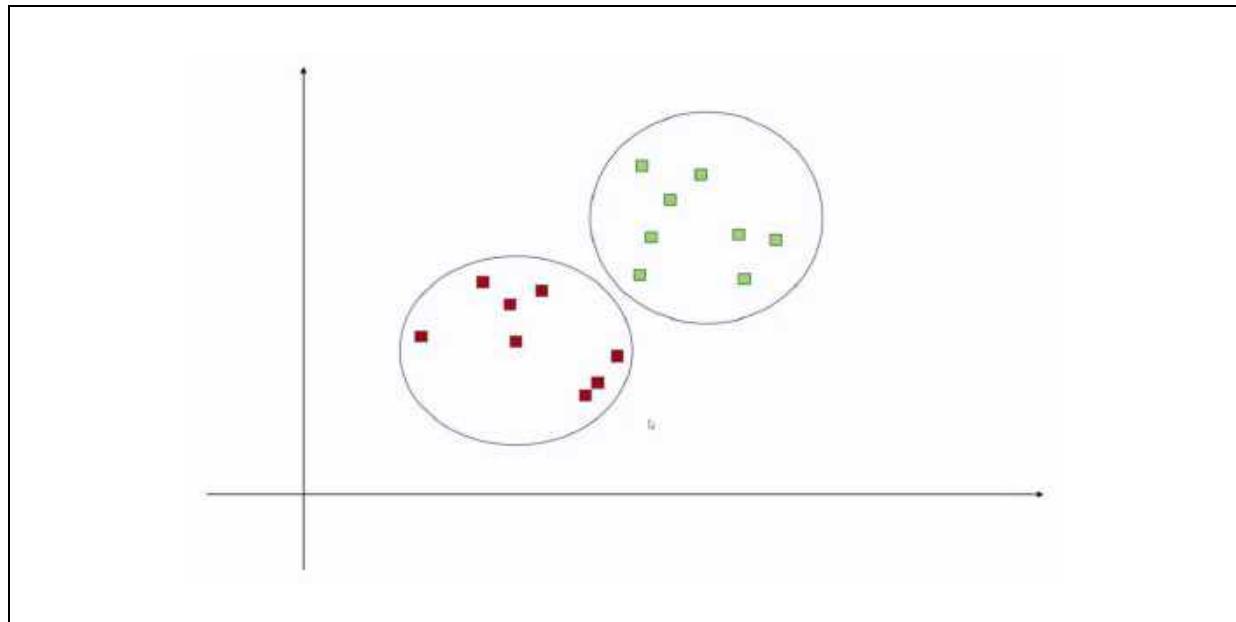
## Data Science – Machine Learning – K – Means Clustering



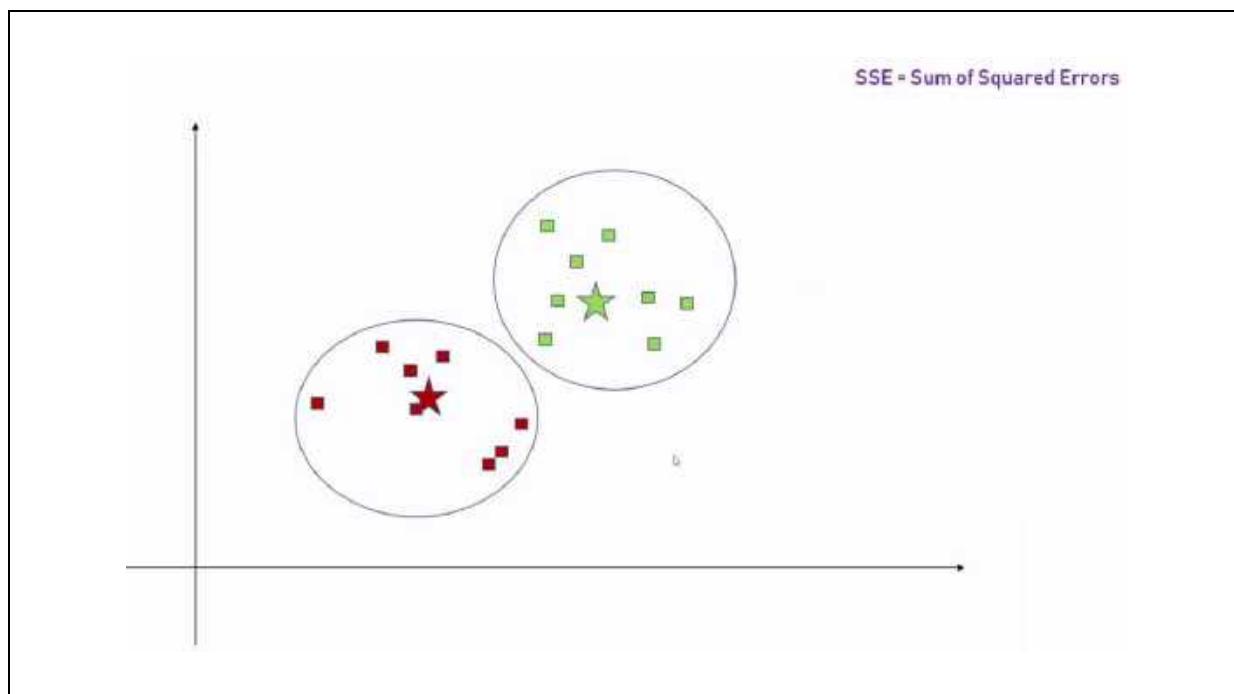
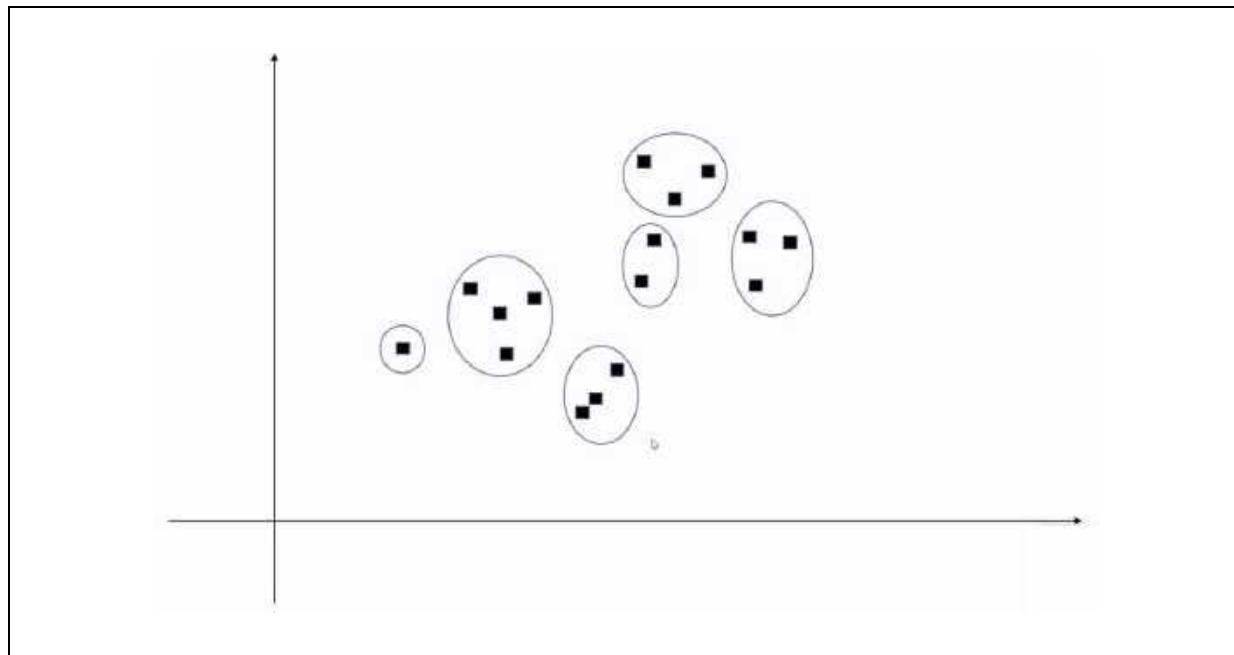
## Data Science – Machine Learning – K – Means Clustering

### 5. How to determine the correct number of clusters?

- ✓ Let's take few scenarios



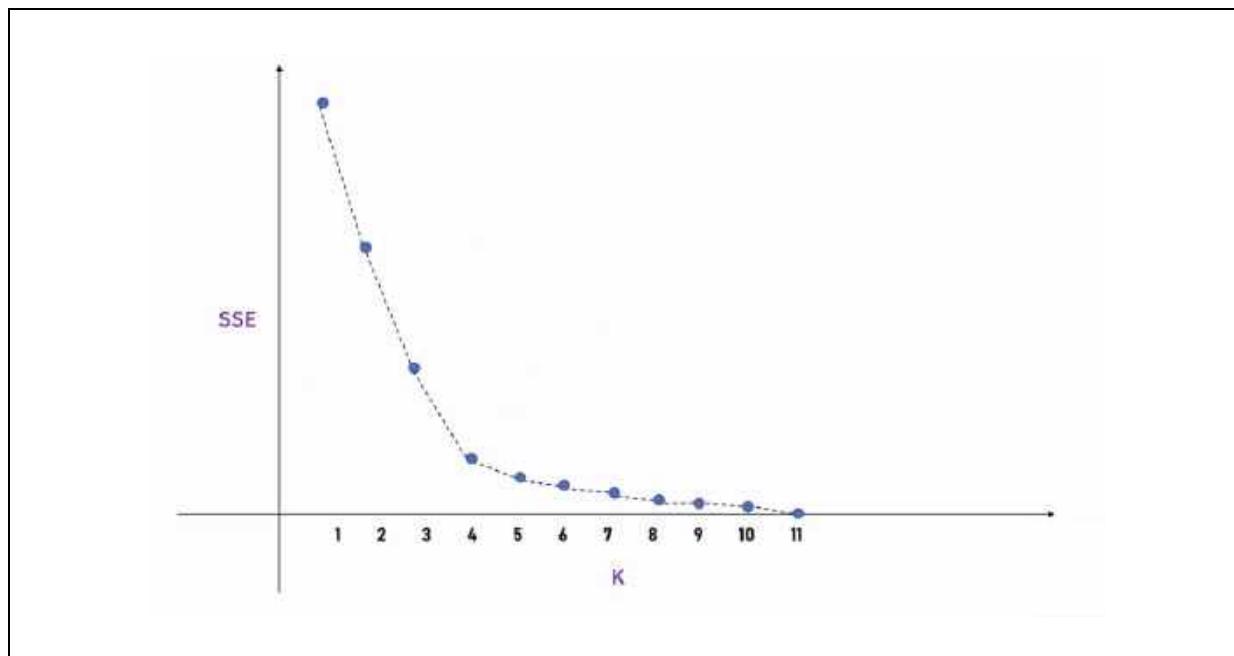
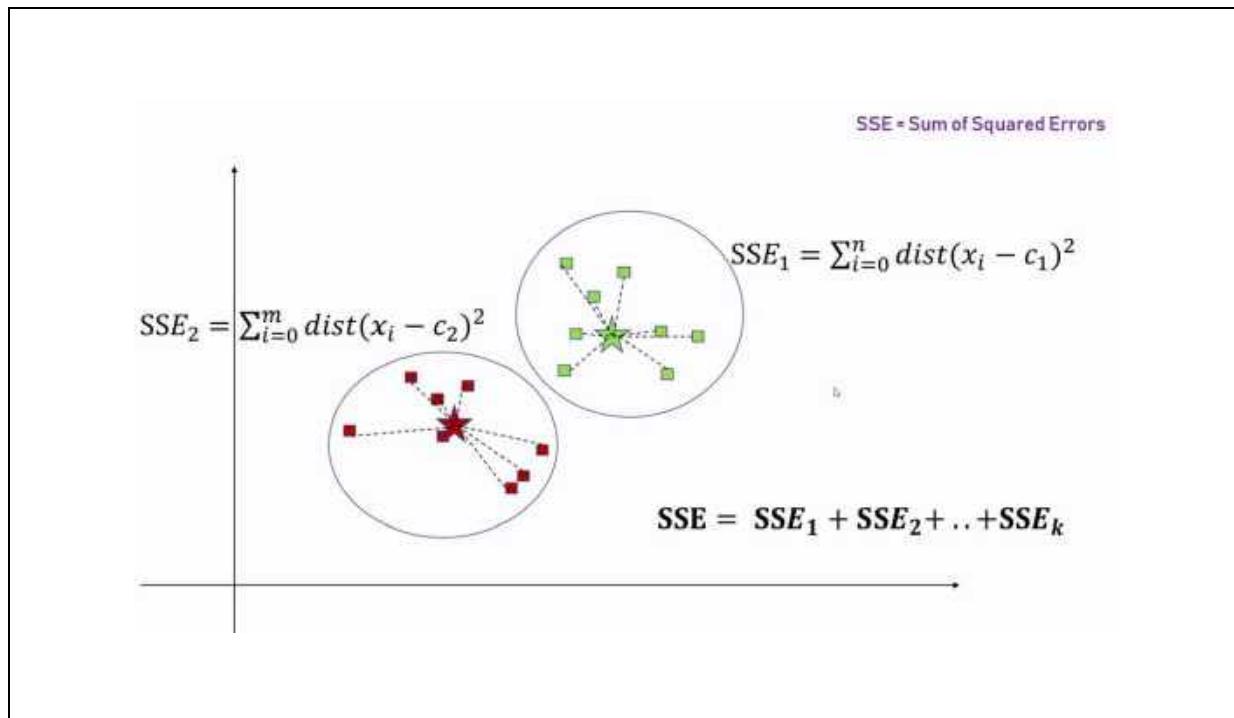
## Data Science – Machine Learning – K – Means Clustering



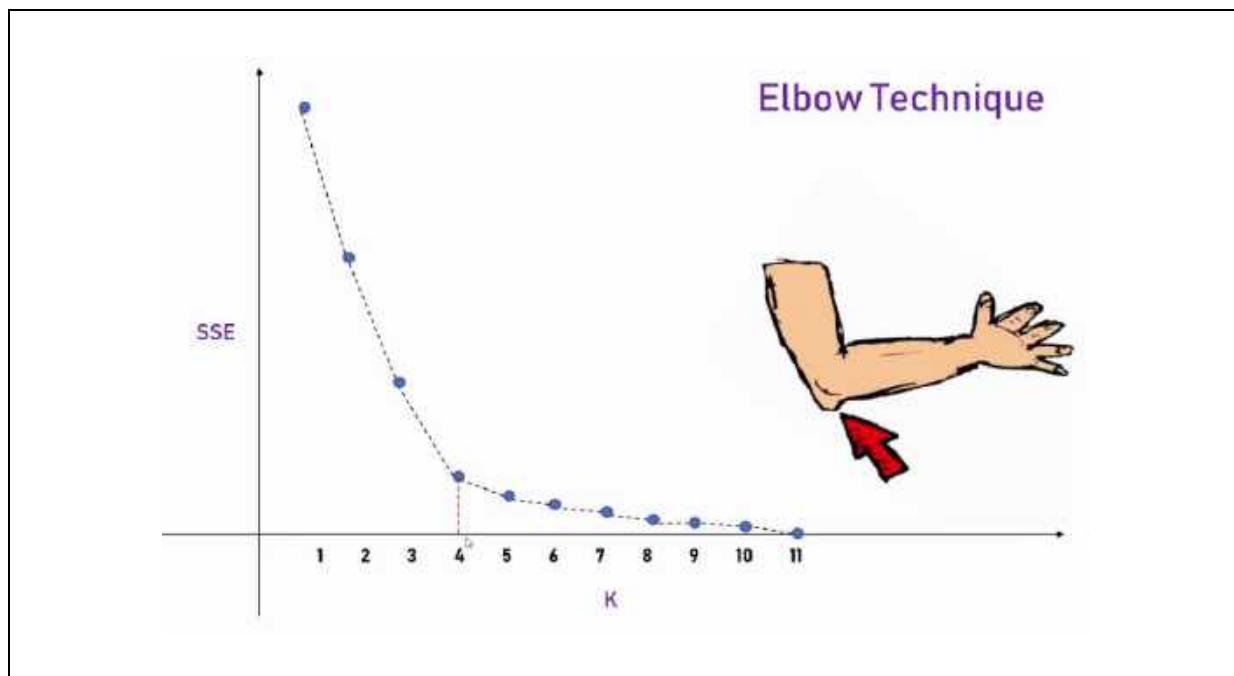
## Data Science – Machine Learning – K – Means Clustering

### 6. Elbow Method

- ✓ The Elbow method is one of the most popular ways to find the optimal number of clusters.
- ✓ This method uses the concept of Cluster Sum of Squares.
- ✓ It creates the total variations within a cluster.



## Data Science – Machine Learning – K – Means Clustering



## Data Science – Machine Learning – K – Means Clustering

**Program Name** Loading the dataset  
demo1.py

```
import pandas as pd

df = pd.read_csv("income.csv")

print(df.head())
```

**Output**

	Name	Age	Income(\$)
0	Rob	27	70000
1	Michael	29	90000
2	Mohan	29	61000
3	Ismail	28	60000
4	Kory	42	150000

## Data Science – Machine Learning – K – Means Clustering

**Program Name** Plotting the data  
demo2.py

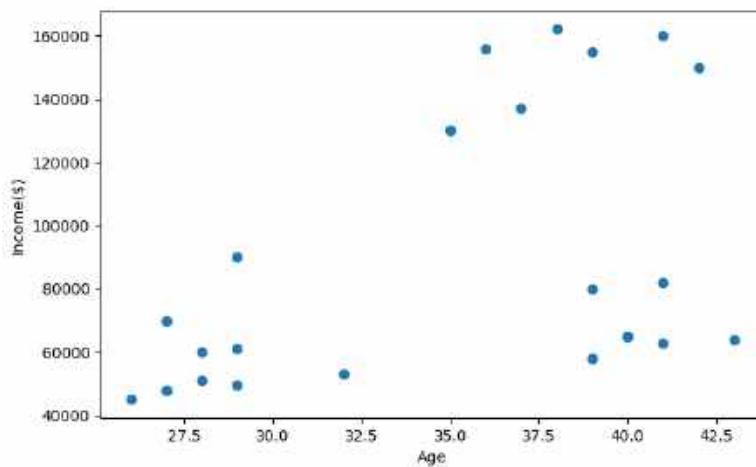
```
import pandas as pd
from matplotlib import pyplot as plt

df = pd.read_csv("income.csv")

plt.scatter(df.Age, df['Income($)'])
plt.xlabel('Age')
plt.ylabel('Income($)')

plt.show()
```

**Output**



## Data Science – Machine Learning – K – Means Clustering

**Program Name** Creating clusters  
demo3.py

```
import pandas as pd
from sklearn.cluster import KMeans

df = pd.read_csv("income.csv")

km = KMeans(n_clusters = 3)
y_predicted = km.fit_predict(df[['Age', 'Income($)']])
print(y_predicted)
```

**Output**

```
[1 1 2 2 0 0 0 0 0 0 2 2 2 2 2 2 2 1 1 2]
```

## Data Science – Machine Learning – K – Means Clustering

**Program Name** Predicting the cluster  
demo4.py

```
import pandas as pd
from sklearn.cluster import KMeans

df = pd.read_csv("income.csv")

km = KMeans(n_clusters = 3)

y_predicted = km.fit_predict(df[['Age', 'Income($)']])
df['cluster']=y_predicted

print(df.head())
```

**Output**

	Name	Age	Income(\$)	cluster
0	Rob	27	70000	2
1	Michael	29	90000	2
2	Mohan	29	61000	0
3	Ismail	28	60000	0
4	Kory	42	150000	1

## Data Science – Machine Learning – K – Means Clustering

Program Name Cluster distance  
demo5.py

```
import pandas as pd
from sklearn.cluster import KMeans

df = pd.read_csv("income.csv")

km = KMeans(n_clusters = 3)

y_predicted = km.fit_predict(df[['Age', 'Income($)']])
df['cluster']=y_predicted

print(km.cluster_centers_)
```

Output

```
[[3.29090909e+01 5.61363636e+04]
 [3.82857143e+01 1.50000000e+05]
 [3.40000000e+01 8.05000000e+04]]
```

## Data Science – Machine Learning – K – Means Clustering

Program

Plotting the clusters

Name

demo6.py

```
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.cluster import KMeans

df = pd.read_csv("income.csv")

km = KMeans(n_clusters = 3)

y_predicted = km.fit_predict(df[['Age', 'Income($)']])
df['cluster'] = y_predicted

df1 = df[df.cluster==0]
df2 = df[df.cluster==1]
df3 = df[df.cluster==2]

plt.scatter(df1.Age, df1['Income($)', color='green'])
plt.scatter(df2.Age, df2['Income($)', color='red'])
plt.scatter(df3.Age, df3['Income($)', color='black'])

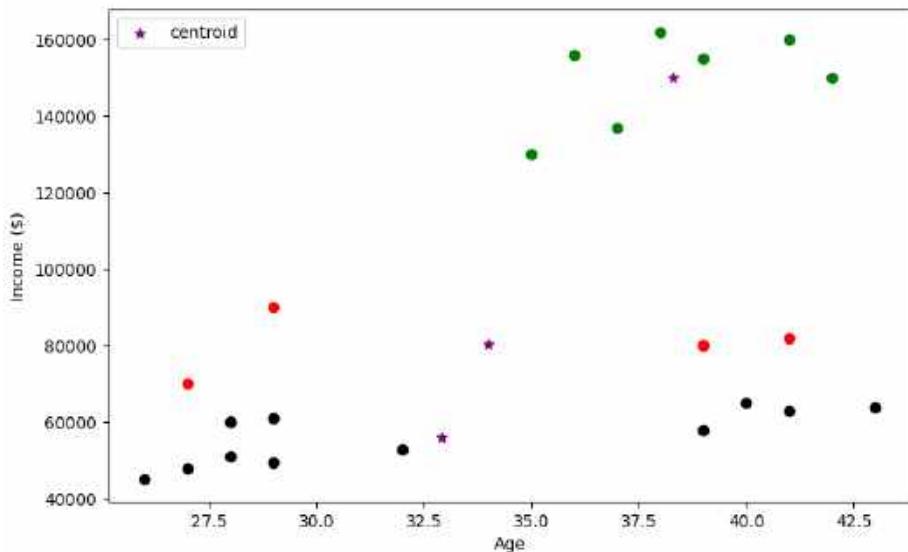
plt.scatter(km.cluster_centers_[:, 0], km.cluster_centers_[:, 1],
            color = 'purple', marker='*', label='centroid')

plt.xlabel('Age')
plt.ylabel('Income ($)')
plt.legend()

plt.show()
```

## Data Science – Machine Learning – K – Means Clustering

### Output



## Data Science – Machine Learning – K – Means Clustering

**Program Name** Features scaling  
demo7.py

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

df = pd.read_csv("income.csv")

scaler = MinMaxScaler()

scaler.fit(df[['Income($)']])
df['Income($)'] = scaler.transform(df[['Income($)']])

scaler.fit(df[['Age']])
df['Age'] = scaler.transform(df[['Age']])

print(df.head())
```

**Output**

	Name	Age	Income(\$)
0	Rob	0.058824	0.213675
1	Michael	0.176471	0.384615
2	Mohan	0.176471	0.136752
3	Ismail	0.117647	0.128205
4	Kory	0.941176	0.897436

## Data Science – Machine Learning – K – Means Clustering

**Program Name** Plotting after features scaling  
demo8.py

```
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.preprocessing import MinMaxScaler

df = pd.read_csv("income.csv")

scaler = MinMaxScaler()

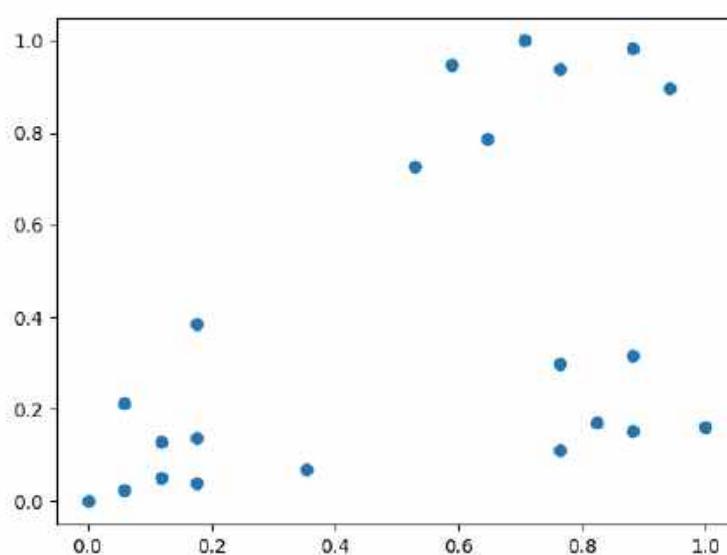
scaler.fit(df[['Income($)']])
df['Income($)'] = scaler.transform(df[['Income($)']])

scaler.fit(df[['Age']])
df['Age'] = scaler.transform(df[['Age']])

plt.scatter(df.Age, df['Income($)'])

plt.show()
```

### Output



## Data Science – Machine Learning – K – Means Clustering

Program Name Prediction  
demo9.py

```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler

df = pd.read_csv("income.csv")

scaler = MinMaxScaler()

scaler.fit(df[['Income($)']])
df['Income($)'] = scaler.transform(df[['Income($)']])

scaler.fit(df[['Age']])
df['Age'] = scaler.transform(df[['Age']])

km = KMeans(n_clusters = 3)
y_predicted = km.fit_predict(df[['Age', 'Income($)']])
print(y_predicted)
```

Output

```
[1 1 1 1 2 2 2 2 2 2 1 1 1 1 1 0 0 0 0 0]
```

## Data Science – Machine Learning – K – Means Clustering

---

**Program Name** Prediction  
demo10.py

```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler

df = pd.read_csv("income.csv")

scaler = MinMaxScaler()

scaler.fit(df[['Income($)']])
df['Income($)'] = scaler.transform(df[['Income($)']])

scaler.fit(df[['Age']])
df['Age'] = scaler.transform(df[['Age']])

km = KMeans(n_clusters=3)
y_predicted = km.fit_predict(df[['Age', 'Income($)']])

df['cluster'] = y_predicted
print(df.head())
```

**Output**

	Name	Age	Income(\$)	cluster
0	Rob	0.058824	0.213675	1
1	Michael	0.176471	0.384615	1
2	Mohan	0.176471	0.136752	1
3	Ismail	0.117647	0.128205	1
4	Kory	0.941176	0.897436	0

## Data Science – Machine Learning – K – Means Clustering

Program Name Cluster distance  
demo11.py

```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler

df = pd.read_csv("income.csv")

scaler = MinMaxScaler()

scaler.fit(df[['Income($)']])
df['Income($)'] = scaler.transform(df[['Income($)']])

scaler.fit(df[['Age']])
df['Age'] = scaler.transform(df[['Age']])

km = KMeans(n_clusters=3)
y_predicted = km.fit_predict(df[['Age', 'Income($)']])

df['cluster']=y_predicted
print(km.cluster_centers_)
```

Output

```
[[0.1372549  0.11633428]
 [0.72268908 0.8974359 ]
 [0.85294118 0.2022792 ]]
```

## Data Science – Machine Learning – K – Means Clustering

---

**Program Name**

Plotting the clusters

demo12.py

```

import pandas as pd
from sklearn.cluster import KMeans
from matplotlib import pyplot as plt
from sklearn.preprocessing import MinMaxScaler

df = pd.read_csv("income.csv")

scaler = MinMaxScaler()

scaler.fit(df[['Income($)']])
df['Income($)'] = scaler.transform(df[['Income($)']])

scaler.fit(df[['Age']])
df['Age'] = scaler.transform(df[['Age']])

km = KMeans(n_clusters=3)
y_predicted = km.fit_predict(df[['Age', 'Income($)']])

df['cluster']=y_predicted

df1 = df[df.cluster==0]
df2 = df[df.cluster==1]
df3 = df[df.cluster==2]

plt.scatter(df1.Age,df1['Income($)'),color='green')
plt.scatter(df2.Age,df2['Income($)'),color='red')
plt.scatter(df3.Age,df3['Income($)'),color='black')

plt.scatter(km.cluster_centers_[:, 0], km.cluster_centers_[:, 1],
           color='purple', marker='*',label='centroid')

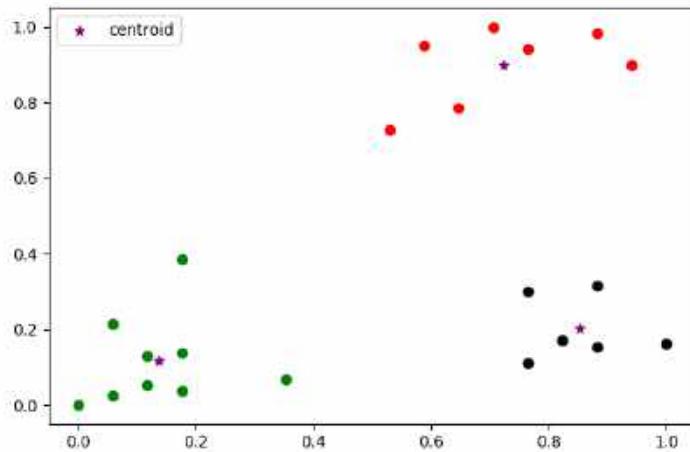
plt.legend()

plt.show()

```

## Data Science – Machine Learning – K – Means Clustering

### Output



## Data Science – Machine Learning – K – Means Clustering

Program Name      Elbow method  
demo13.py

```
import pandas as pd
from sklearn.cluster import KMeans
from matplotlib import pyplot as plt
from sklearn.preprocessing import MinMaxScaler

df = pd.read_csv("income.csv")

scaler = MinMaxScaler()

scaler.fit(df[['Income($)']])
df['Income($)'] = scaler.transform(df[['Income($)']])

scaler.fit(df[['Age']])
df['Age'] = scaler.transform(df[['Age']])

sse = []
k_rng = range(1,10)

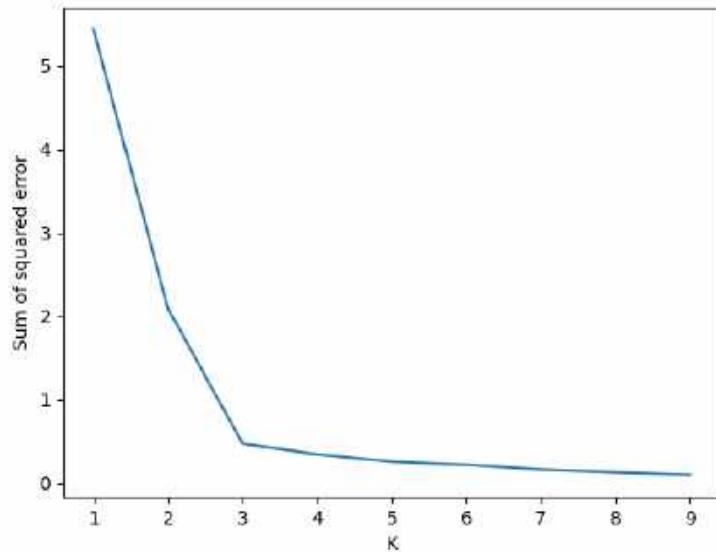
for k in k_rng:
    km = KMeans(n_clusters=k)
    km.fit(df[['Age', 'Income($)']])
    sse.append(km.inertia_)

plt.xlabel('K')
plt.ylabel('Sum of squared error')
plt.plot(k_rng, sse)

plt.show()
```

## Data Science – Machine Learning – K – Means Clustering

Output



**29. Data Science – Machine Learning – K Nearest Neighbor****Contents**

<b>1. K-Nearest Neighbor Algorithm .....</b>	<b>2</b>
<b>2. How it works?.....</b>	<b>2</b>
<b>3. Scenario .....</b>	<b>3</b>
<b>4. Use case .....</b>	<b>5</b>

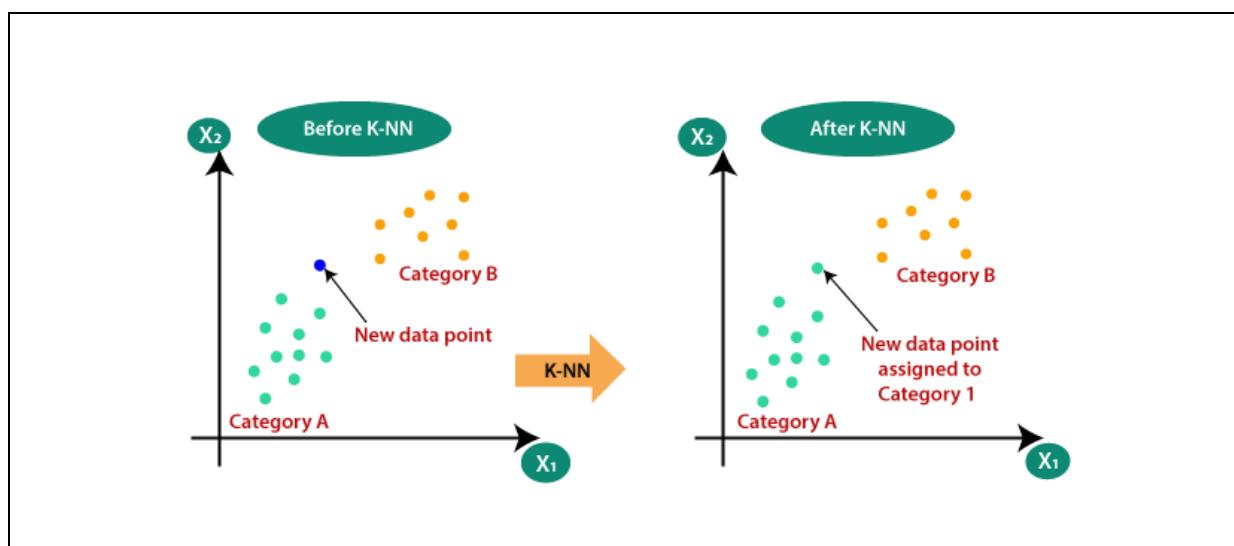
## 29. Data Science – Machine Learning – K Nearest Neighbor

### 1. K-Nearest Neighbor Algorithm

- ✓ K-Nearest Neighbor is a Supervised Learning technique.
- ✓ K-NN algorithm follow one basic rule that is, similar things are near to each other.
- ✓ It is also called a lazy learner algorithm because it does not learn from the training set immediately.
  - At the time of training phase this algorithm just stores the dataset
  - Whenever we get new data point then it classifies the category

### 2. How it works?

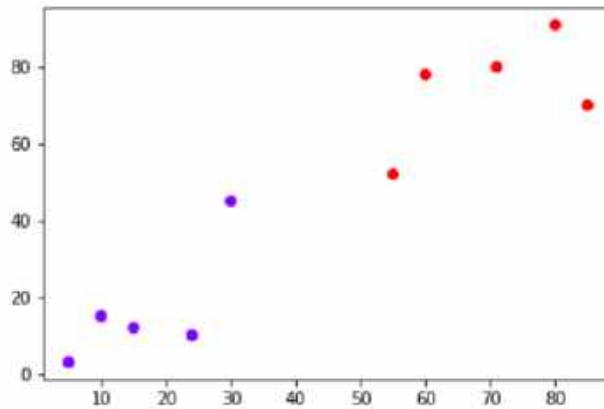
- ✓ It simply calculates the distance of a new data point to all other training data points.
- ✓ The distance can be of any type e.g. Euclidean or Manhattan etc.
- ✓ It selects the K-nearest data points.
- ✓ Finally it assigns the data point to the class to which the majority of the K data points belong.



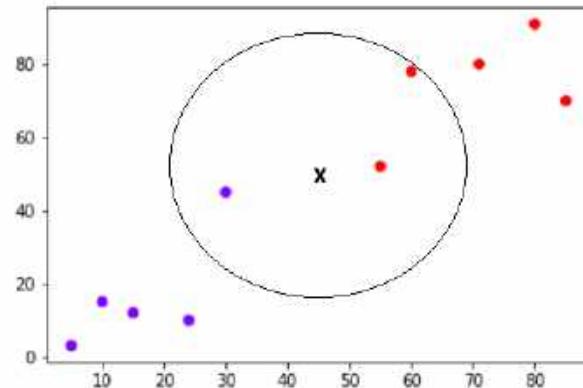
## Data Science – Machine Learning – K Nearest Neighbor

### 3. Scenario

- ✓ Suppose you have a dataset with two variables, which when plotted, looks like the one in the following figure.



- ✓ Our task is to classify a new data point with 'X' into "Blue" class or "Red" class.
- ✓ Suppose the value of K is 3.
- ✓ The KNN algorithm starts by calculating the distance of point X from all the points.
- ✓ It then finds the 3 nearest points with least distance to point X.
- ✓ This is shown in the figure below; the three nearest points have been encircled.



## Data Science – Machine Learning – K Nearest Neighbor

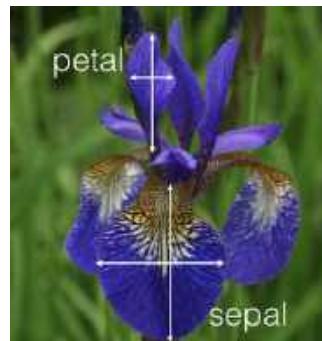
---

- ✓ The final step of the KNN algorithm is to assign new point to the class to which majority of the three nearest points belong.
- ✓ From the above image we can see that the two of the three nearest points belong to the class "Red" while one belongs to the class "Blue".
- ✓ Therefore the new data point will be classified as "Red".

## Data Science – Machine Learning – K Nearest Neighbor

### 4. Use case

- ✓ Assuming that Abhi had a hobby which is interested in distinguishing the species of some iris flowers that he has found
- ✓ He has collected some measurements associated with each iris, which are:
  - The **length** and **width** of the **petals**
  - The **length** and **width** of the **sepals**, all measured in centimetres.
- ✓ She also has the measurements of some irises that have been previously identified to the species
  - setosa,
  - versicolor
  - virginica
- ✓ The goal is to create a machine learning model that can learn from the measurements of these irises whose species are already known.
- ✓ So that we can predict the species for the new irises that she has found.



Iris Versicolor



Iris Setosa



Iris Virginica

## Data Science – Machine Learning – K Nearest Neighbor

---

### Flower codes

- ✓ Setosa - 0
- ✓ Versicolor - 1
- ✓ Virginica - 2

## Data Science – Machine Learning – K Nearest Neighbor

---

**Program Name** Loading iris dataset  
demo1.py

```
from sklearn.datasets import load_iris

iris = load_iris()

print(dir(iris))
```

**Output**

```
['DESCR', 'data', 'feature_names', 'filename', 'frame', 'target',
'target_names']
```

## Data Science – Machine Learning – K Nearest Neighbor

**Program Name** Displaying feature names  
demo2.py

```
from sklearn.datasets import load_iris

iris = load_iris()

print(iris.feature_names)
```

**Output**

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

**Program Name** Displaying target names  
demo3.py

```
from sklearn.datasets import load_iris

iris = load_iris()

print(iris.target_names)
```

**Output**

```
['setosa' 'versicolor' 'virginica']
```

## Data Science – Machine Learning – K Nearest Neighbor

**Program Name** Displaying data  
demo4.py

```
from sklearn.datasets import load_iris

iris = load_iris()

print(iris.data)
```

**Output**

```
[[5.1 3.5 1.4 0.2]
 [4.9 3. 1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5. 3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5. 3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3. 1.4 0.1]
 [4.3 3. 1.1 0.1]]
```

## Data Science – Machine Learning – K Nearest Neighbor

---

**Program Name** Length of the data  
demo5.py

```
from sklearn.datasets import load_iris

iris = load_iris()

print(len(iris.data))
```

**Output**  
150

## Data Science – Machine Learning – K Nearest Neighbor

**Program Name** Create a Dataframe by using data and features  
demo6.py

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

print(df)
```

### Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
..	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

[150 rows x 4 columns]

## Data Science – Machine Learning – K Nearest Neighbor

**Program Name** Adding target column to the dataframe  
demo7.py

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

print(df.head())
```

### Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

## Data Science – Machine Learning – K Nearest Neighbor

**Program Name** Displaying target == 0 flowers  
demo8.py

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

print(df[df.target==0].head())
```

**Output**

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

## Data Science – Machine Learning – K Nearest Neighbor

**Program Name** Displaying length of the target == 0 flowers  
demo9.py

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

print(len(df[df.target==0]))
```

**Output**

50

## Data Science – Machine Learning – K Nearest Neighbor

**Program Name** Displaying target == 1 flowers  
demo10.py

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

print(df[df.target==1].head())
```

**Output**

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
50	7.0	3.2	4.7	1.4	1
51	6.4	3.2	4.5	1.5	1
52	6.9	3.1	4.9	1.5	1
53	5.5	2.3	4.0	1.3	1
54	6.5	2.8	4.6	1.5	1

## Data Science – Machine Learning – K Nearest Neighbor

**Program Name** Displaying length of the target == 0 flowers  
demo11.py

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

print(len(df[df.target==1]))
```

**Output**

50

## Data Science – Machine Learning – K Nearest Neighbor

**Program Name** Displaying target == 2 flowers  
demo12.py

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

print(df[df.target==2].head())
```

### Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
100	6.3	3.3	6.0	2.5	2
101	5.8	2.7	5.1	1.9	2
102	7.1	3.0	5.9	2.1	2
103	6.3	2.9	5.6	1.8	2
104	6.5	3.0	5.8	2.2	2

## Data Science – Machine Learning – K Nearest Neighbor

**Program Name** Displaying length of the target == 2 flowers  
demo13.py

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

print(len(df[df.target==2]))
```

**Output**

50

## Data Science – Machine Learning – K Nearest Neighbor

**Program Name** Displaying the flower names  
demo14.py

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

df['flower_name'] = df.target.apply(lambda x: iris.target_names[x])
print(df)
```

### Output

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  target  flower_name
0           5.1          3.5            1.4            0.2      0    setosa
1           4.9          3.0            1.4            0.2      0    setosa
2           4.7          3.2            1.3            0.2      0    setosa
3           4.6          3.1            1.5            0.2      0    setosa
4           5.0          3.6            1.4            0.2      0    setosa
..          ...
145          6.7          3.0            5.2            2.3      2  virginica
146          6.3          2.5            5.0            1.9      2  virginica
147          6.5          3.0            5.2            2.0      2  virginica
148          6.2          3.4            5.4            2.3      2  virginica
149          5.9          3.0            5.1            1.8      2  virginica
[150 rows x 6 columns]
```

## Data Science – Machine Learning – K Nearest Neighbor

**Program Name** All setosa flowers  
demo15.py

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

setosa_50 = df[:50]
print(setosa_50.head())
```

### Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa

## Data Science – Machine Learning – K Nearest Neighbor

**Program Name** All versicolor flowers  
demo16.py

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

versicolor_50 = df[50:100]
print(versicolor_50.head())
```

### Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
50	7.0	3.2	4.7	1.4	1	versicolor
51	6.4	3.2	4.5	1.5	1	versicolor
52	6.9	3.1	4.9	1.5	1	versicolor
53	5.5	2.3	4.8	1.3	1	versicolor
54	6.5	2.8	4.6	1.5	1	versicolor

## Data Science – Machine Learning – K Nearest Neighbor

**Program Name** All virginica flowers  
demo17.py

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

verginica_50 = df[100:]
print(verginica_50.head())
```

### Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
100	6.3	3.3	6.0	2.5	2	virginica
101	5.8	2.7	5.1	1.9	2	virginica
102	7.1	3.0	5.9	2.1	2	virginica
103	6.3	2.9	5.6	1.8	2	virginica
104	6.5	3.0	5.8	2.2	2	virginica

## Data Science – Machine Learning – K Nearest Neighbor

**Program Name** Splitting the data  
demo18.py

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

X = df.drop(['target', 'flower_name'], axis='columns')
y = df.target

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)

print("Splitting the data")
```

**Output**

Splitting the data

## Data Science – Machine Learning – K Nearest Neighbor

**Program Name** Model training  
demo19.py

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

X = df.drop(['target', 'flower_name'], axis='columns')
y = df.target

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)

# Train Using K Neighbor classifier

classifier = KNeighborsClassifier(n_neighbors = 5)
classifier.fit(X_train, y_train)

print('Model got trained')
```

### Output

Model got trained

## Data Science – Machine Learning – K Nearest Neighbor

Program Name Model score  
demo20.py

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

X = df.drop(['target', 'flower_name'], axis='columns')
y = df.target

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)

classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)

print(classifier.score(X_test, y_test))
```

### Output

0.9666666666666667

## Data Science – Machine Learning – K Nearest Neighbor

Program

Name demo21.py

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

X = df.drop(['target', 'flower_name'], axis='columns')
y = df.target

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)

classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)

print(classifier.predict([[4.8, 3.0, 1.5, 0.3]]))
```

Output

[0]

## Data Science – Machine Learning – K Nearest Neighbor

Program Name

Model prediction

demo22.py

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

X = df.drop(['target', 'flower_name'], axis='columns')
y = df.target

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)

classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

print(y_pred)
```

Output

```
[0 1 0 1 2 1 0 1 2 0 0 2 1 0 1 0 0 0 0 1 1 0 2 2 0 2 0 0 1 0]
```

## Data Science – Machine Learning – K Nearest Neighbor

Program Name

Model evaluation  
demo22.py

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report,
confusion_matrix

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target
df['flower_name'] = df.target.apply(lambda x:
iris.target_names[x])

X = df.drop(['target', 'flower_name'], axis='columns')
y = df.target

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)

classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

## Data Science – Machine Learning – K Nearest Neighbor

### Output

[[13 0 0]				
[ 0 7 2]				
[ 0 0 8]]				
	precision	recall	f1-score	support
0	1.00	1.00	1.00	13
1	1.00	0.78	0.88	9
2	0.80	1.00	0.89	8
accuracy			0.93	30
macro avg	0.93	0.93	0.92	30
weighted avg	0.95	0.93	0.93	30

**30. Data Science – Machine Learning – Naïve Bayes Classifier****Contents**

1. Naive Bayes Classifier.....	2
2. Why is it called Naïve Bayes?.....	2
3. Bayes theorem.....	3
4. Scenario .....	4
5. Conditional probability .....	6
6. Use case .....	9
7. Math problem and solution.....	10
8. Use cases.....	12

## 30. Data Science – Machine Learning – Naïve Bayes Classifier

### 1. Naïve Bayes Classifier

- ✓ Naïve Bayes algorithm is a supervised learning algorithm.
- ✓ This is based on Bayes theorem and used for solving classification problems.
- ✓ It is mainly used in text classification.
  - Examples like spam filtration, Sentimental analysis, and classifying articles etc.

### 2. Why is it called Naïve Bayes?

- ✓ **Naive:**
  - It is called Naive because it assumes that the occurrence of a certain feature is independent of the other features.
  - Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple.
  - Hence each feature individually contributes to identify that it is an apple without depending on each other.
- ✓ **Bayes:**
  - It is called Bayes because it depends on the principle of Bayes' Theorem.

### 3. Bayes theorem

- ✓ Bayes' theorem is also known as Bayes' Rule or Bayes' law.
- ✓ It is used to determine the probability of a hypothesis with prior knowledge.
- ✓ It depends on the conditional probability.



Thomas Bayes

$$P(A/B) = \frac{P(B/A) * P(A)}{P(B)}$$

#### 4. Scenario

- ✓ When we flip a coin, the probability of getting head or tail is  $1/2$ , because there are two possibilities of outcomes
- ✓ So the chance of getting head or tail is 50%



$$p(\text{head}) = 1/2$$

Pick a random card, what is the probability of getting a queen?



4 queens, 52 total cards

$$P(\text{queen}) = 4/52 = 1/13$$

## 5. Conditional probability

Pick a random card, you know it is a diamond. Now what is the probability of that card being a queen?



Total diamonds = 13

Queen = 1



Total diamonds = 13

Queen = 1

$P(\text{queen/diamond}) = 1/13$

### Conditional Probability



$P(\text{queen/diamond}) = 1/13$

$P(A/B) = \text{Probability of event A knowing}$   
 $\text{that event B has already occurred}$

## Data Science – Machine Learning – Naïve Bayes Classifier



Thomas Bayes

$$P(A/B) = \frac{P(B/A) * P(A)}{P(B)}$$

$$P(\text{queen/diamond}) = \frac{P(\text{diamond/queen}) * P(\text{queen})}{P(\text{diamond})}$$
$$\begin{aligned} P(\text{diamond/queen}) &= 1/4 &= \frac{1/4 * 1/13}{1/4} \\ P(\text{queen}) &= 1/13 & \\ P(\text{diamond}) &= 1/4 &= 1/13 \end{aligned}$$

## Data Science – Machine Learning – Naïve Bayes Classifier

### 6. Use case

Passenger Id	Name	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Survived
1	Braund, Mr. Owen Harris	3	male	22	1	0	21171	7.25	S		0
2	Cumings, Mrs. John Bradley	1	female	38	1	0	17599	71.2833	C85	C	1
3	Heikkinen, Miss. Laina	3	female	26	0	0	3101282	7.925		S	1
4	Futrelle, Mrs. Jacques Heath	1	female	35	1	0	113803	53.1	C123	S	1
5	Allen, Mr. William Henry	3	male	35	0	0	373450	8.05		S	0
6	Moran, Mr. James	3	male		0	0	330877	8.4583	Q		0
7	McCarthy, Mr. Timothy J	1	male	54	0	0	17463	51.8625	E46	S	0
8	Palsson, Master. Gosta Leonard	3	male	2	3	1	349909	21.075		S	0
9	Johnson, Mrs. Oscar	3	female	27	0	2	347742	11.1333		S	1
10	Nasser, Mrs. Nicholas	2	female	14	1	0	237736	30.0708	C		1

Passenger Id	Name	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Survived
1	Braund, Mr. Owen Harris	3	male	22	1	0	21171	7.25	S		0
2	Cumings, Mrs. John Bradley	1	female	38	1	0	17599	71.2833	C85	C	1
3	Heikkinen, Miss. Laina	3	female	26	0	0	3101282	7.925		S	1
4	Futrelle, Mrs. Jacques Heath	1	female	35	1	0	113803	53.1	C123	S	1
5	Allen, Mr. William Henry	3	male	35	0	0	373450	8.05		S	0
6	Moran, Mr. James	3	male		0	0	330877	8.4583	Q		0
7	McCarthy, Mr. Timothy J	1	male	54	0	0	17463	51.8625	E46	S	0
8	Palsson, Master. Gosta Leonard	3	male	2	3	1	349909	21.075		S	0
9	Johnson, Mrs. Oscar	3	female	27	0	2	347742	11.1333		S	1
10	Nasser, Mrs. Nicholas	2	female	14	1	0	237736	30.0708	C		1

$$P\left(\frac{\text{Survived}}{\text{Male} \& \text{Class} \& \text{Age} \& \text{Cabin} \& \text{Fare}}\right)$$

Make a naïve assumption that features such as male, class, age , cabin, fare etc. are independent of each other

## 7. Math problem and solution

### Problem

- ✓ If the weather is sunny, then the Player should play or not?

### Solution

- ✓ To solve this, first consider the below dataset

	Outlook	Play
0	Rainy	Yes
1	Sunny	Yes
2	Overcast	Yes
3	Overcast	Yes
4	Sunny	No
5	Rainy	Yes
6	Sunny	Yes
7	Overcast	Yes
8	Rainy	No
9	Sunny	No
10	Sunny	Yes
11	Rainy	No
12	Overcast	Yes
13	Overcast	Yes

Frequency table for the Weather Conditions:

Weather	Yes	No
Overcast	5	0
Rainy	2	2
Sunny	3	2
Total	10	5

## Data Science – Machine Learning – Naïve Bayes Classifier

Likelihood table weather condition:

Weather	No	Yes	
Overcast	0	5	5/14= 0.35
Rainy	2	2	4/14=0.29
Sunny	2	3	5/14=0.35
All	4/14=0.29	10/14=0.71	

Applying Bayes' theorem:

$$P(\text{Yes}|\text{Sunny}) = P(\text{Sunny}|\text{Yes}) * P(\text{Yes}) / P(\text{Sunny})$$

$$P(\text{Sunny}|\text{Yes}) = 3/10 = 0.3$$

$$P(\text{Sunny}) = 0.35$$

$$P(\text{Yes}) = 0.71$$

$$\text{So } P(\text{Yes}|\text{Sunny}) = 0.3 * 0.71 / 0.35 = 0.60$$

$$P(\text{No}|\text{Sunny}) = P(\text{Sunny}|\text{No}) * P(\text{No}) / P(\text{Sunny})$$

$$P(\text{Sunny}|\text{No}) = 2/4 = 0.5$$

$$P(\text{No}) = 0.29$$

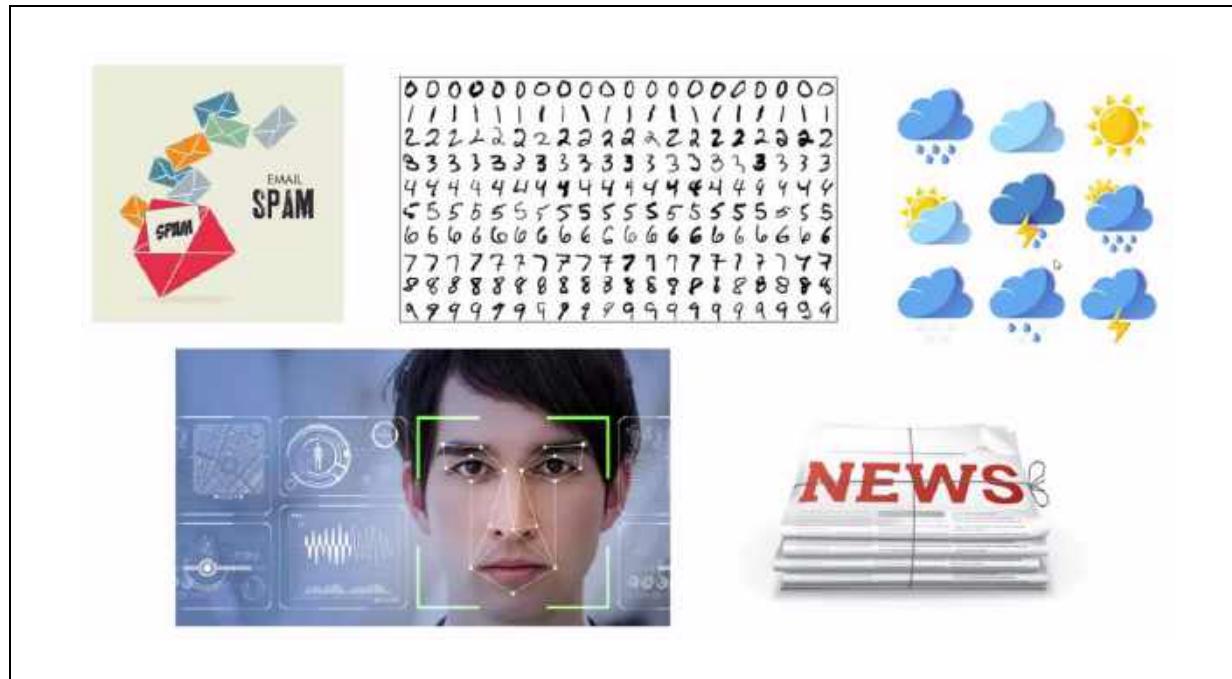
$$P(\text{Sunny}) = 0.35$$

$$\text{So } P(\text{No}|\text{Sunny}) = 0.5 * 0.29 / 0.35 = 0.41$$

So as we can see from the above calculation that  $P(\text{Yes}|\text{Sunny}) > P(\text{No}|\text{Sunny})$

Hence on a Sunny day, Player can play the game.

## 8. Use cases



## Data Science – Machine Learning – Naïve Bayes Classifier

**Program Name** Loading the dataset  
demo1.py

```
import pandas as pd

df = pd.read_csv("titanic.csv")

print(df.head())
```

**Output**

	Pclass	Sex	Age	Fare	Survived
0	3	male	22.0	7.2500	0
1	1	female	38.0	71.2833	1
2	3	female	26.0	7.9250	1
3	1	female	35.0	53.1000	1
4	3	male	35.0	8.0500	0

## Data Science – Machine Learning – Naïve Bayes Classifier

**Program Name** Preparing input  
demo2.py

```
import pandas as pd

df = pd.read_csv('titanic.csv', usecols = ['Pclass', 'Sex', 'Age', 'Fare',
'Survived'])
inputs = df.drop('Survived', axis='columns')

print(inputs.head())
```

**Output**

	Pclass	Sex	Age	Fare
0	3	male	22.0	7.2500
1	1	female	38.0	71.2833
2	3	female	26.0	7.9250
3	1	female	35.0	53.1000
4	3	male	35.0	8.0500

## Data Science – Machine Learning – Naïve Bayes Classifier

**Program Name** Preparing target  
demo3.py

```
import pandas as pd

df = pd.read_csv('titanic.csv', usecols = ['Pclass', 'Sex', 'Age', 'Fare',
'Survived'])
inputs = df.drop('Survived', axis='columns')

target = df.Survived

print(target)
```

**Output**

```
0      0
1      1
2      1
3      1
4      0
 ..
886    0
887    1
888    0
889    1
890    0
Name: Survived, Length: 891, dtype: int64
```

## Data Science – Machine Learning – Naïve Bayes Classifier

Program Name

Creating dummy variables

demo4.py

```
import pandas as pd

df = pd.read_csv('titanic.csv', usecols = ['Pclass', 'Sex', 'Age', 'Fare',
'Survived'])
inputs = df.drop('Survived', axis = 'columns')
target = df.Survived

dummies = pd.get_dummies(inputs.Sex, dtype = int)

print(dummies.head())
```

Output

	female	male
0	0	1
1	1	0
2	1	0
3	1	0
4	0	1

## Data Science – Machine Learning – Naïve Bayes Classifier

**Program Name** Concatenating the dataframe  
demo5.py

```
import pandas as pd

df = pd.read_csv('titanic.csv', usecols = ['Pclass', 'Sex', 'Age', 'Fare',
'Survived'])

inputs = df.drop('Survived', axis='columns')
target = df.Survived
dummies = pd.get_dummies(inputs.Sex, dtype = int)

inputs = pd.concat([inputs, dummies], axis = 'columns')

print(inputs.head())
```

**Output**

	Pclass	Sex	Age	Fare	female	male
0	3	male	22.0	7.2500	0	1
1	1	female	38.0	71.2833	1	0
2	3	female	26.0	7.9250	1	0
3	1	female	35.0	53.1000	1	0
4	3	male	35.0	8.0500	0	1

## Data Science – Machine Learning – Naïve Bayes Classifier

**Program Name** Dropping unnecessary column  
demo6.py

```
import pandas as pd

df = pd.read_csv('titanic.csv', usecols = ['Pclass', 'Sex', 'Age', 'Fare',
'Survived'])

inputs = df.drop('Survived', axis='columns')
target = df.Survived
dummies = pd.get_dummies(inputs.Sex, dtype = int)
inputs = pd.concat([inputs, dummies], axis = 'columns')

inputs.drop(['Sex', 'male'], axis = 'columns', inplace = True)

print(inputs.head())
```

### Output

	Pclass	Age	Fare	female
0	3	22.0	7.2500	0
1	1	38.0	71.2833	1
2	3	26.0	7.9250	1
3	1	35.0	53.1000	1
4	3	35.0	8.0500	0

## Data Science – Machine Learning – Naïve Bayes Classifier

Program  
Name

Checking empty values  
demo7.py

```
import pandas as pd

df = pd.read_csv('titanic.csv', usecols = ['Pclass', 'Sex', 'Age', 'Fare',
'Survived'])

inputs = df.drop('Survived', axis = 'columns')
target = df.Survived
dummies = pd.get_dummies(inputs.Sex, dtype = int)
inputs = pd.concat([inputs, dummies], axis = 'columns')
inputs.drop(['Sex', 'male'], axis='columns', inplace = True)

print(inputs.columns[inputs.isna().any()])
```

Output

```
Index(['Age'], dtype='object')
```

## Data Science – Machine Learning – Naïve Bayes Classifier

**Program Name** Checking empty values  
demo8.py

```
import pandas as pd

df = pd.read_csv('titanic.csv', usecols = ['Pclass', 'Sex', 'Age', 'Fare',
'Survived'])

inputs = df.drop('Survived', axis='columns')
target = df.Survived
dummies = pd.get_dummies(inputs.Sex, dtype = int)

inputs = pd.concat([inputs, dummies], axis = 'columns')

inputs.drop(['Sex', 'male'],axis='columns', inplace=True)

print(inputs.Age)
```

### Output

```
0      22.0
1      38.0
2      26.0
3      35.0
4      35.0
...
886    27.0
887    19.0
888    NaN
889    26.0
890    32.0
Name: Age, Length: 891, dtype: float64
```

## Data Science – Machine Learning – Naïve Bayes Classifier

---

**Program Name** Filling Age Nan values with mean  
demo9.py

```
import pandas as pd

df = pd.read_csv('titanic.csv', usecols = ['Pclass', 'Sex', 'Age', 'Fare',
'Survived'])

inputs = df.drop('Survived', axis='columns')
target = df.Survived
dummies = pd.get_dummies(inputs.Sex, dtype = int)
inputs = pd.concat([inputs, dummies], axis = 'columns')
inputs.drop(['Sex', 'male'],axis='columns', inplace=True)
inputs.Age = inputs.Age.fillna(inputs.Age.mean())

print(inputs)
```

### Output

	Pclass	Age	Fare	female
0	3	22.000000	7.2500	0
1	1	38.000000	71.2833	1
2	3	26.000000	7.9250	1
3	1	35.000000	53.1000	1
4	3	35.000000	8.0500	0
..	...	...	...	...
886	2	27.000000	13.0000	0
887	1	19.000000	30.0000	1
888	3	29.699118	23.4500	1
889	1	26.000000	30.0000	0
890	3	32.000000	7.7500	0

[891 rows x 4 columns]

## Data Science – Machine Learning – Naïve Bayes Classifier

### Program

Name demo10.py

```
import pandas as pd
from sklearn.model_selection import train_test_split

df = pd.read_csv('titanic.csv', usecols = ['Pclass', 'Sex', 'Age', 'Fare',
'Survived'])

inputs = df.drop('Survived', axis='columns')
target = df.Survived
dummies = pd.get_dummies(inputs.Sex, dtype = int)
inputs = pd.concat([inputs, dummies], axis = 'columns')
inputs.drop(['Sex', 'male'],axis='columns', inplace=True)
inputs.Age = inputs.Age.fillna(inputs.Age.mean())

X_train, X_test, y_train, y_test = train_test_split(inputs, target,
test_size=0.3)

print("Splitting the dataset")
```

### Output

Splitting the dataset

## Data Science – Machine Learning – Naïve Bayes Classifier

**Program Name** Creating model  
demo11.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB

df = pd.read_csv('titanic.csv', usecols = ['Pclass', 'Sex', 'Age', 'Fare',
'Survived'])

inputs = df.drop('Survived', axis='columns')
target = df.Survived
dummies = pd.get_dummies(inputs.Sex, dtype = int)
inputs = pd.concat([inputs, dummies], axis = 'columns')
inputs.drop(['Sex', 'male'],axis='columns', inplace=True)
inputs.Age = inputs.Age.fillna(inputs.Age.mean())
X_train, X_test, y_train, y_test = train_test_split(inputs, target,
test_size=0.3)

model = GaussianNB()
model.fit(X_train, y_train)

print("Model got trained")
```

**Output**  
Model got trained

## Data Science – Machine Learning – Naïve Bayes Classifier

---

**Program Name** Prediction demo12.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB

df = pd.read_csv('titanic.csv', usecols = ['Pclass', 'Sex', 'Age', 'Fare',
'Survived'])

inputs = df.drop('Survived', axis='columns')
target = df.Survived
dummies = pd.get_dummies(inputs.Sex, dtype = int)
inputs = pd.concat([inputs, dummies], axis = 'columns')
inputs.drop(['Sex', 'male'],axis='columns', inplace=True)
inputs.Age = inputs.Age.fillna(inputs.Age.mean())
X_train, X_test, y_train, y_test = train_test_split(inputs, target,
test_size=0.3)

model = GaussianNB()
model.fit(X_train, y_train)

print(X_test[0:5])
print(model.predict(X_test[0:5]))
```

**Output**

	Pclass	Age	Fare	female
445	1	4.00	81.8583	0
831	2	0.83	18.7500	0
294	3	24.00	7.8958	0
736	3	48.00	34.3750	1
525	3	40.50	7.7500	0
	[1 0 0 1 0]			

## Data Science – Machine Learning – Naïve Bayes Classifier

---

**Program Name** Prediction probability  
demo13.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB

df = pd.read_csv('titanic.csv', usecols = ['Pclass', 'Sex', 'Age', 'Fare',
'Survived'])

inputs = df.drop('Survived', axis='columns')
target = df.Survived
dummies = pd.get_dummies(inputs.Sex, dtype = int)
inputs = pd.concat([inputs, dummies], axis = 'columns')
inputs.drop(['Sex', 'male'],axis='columns', inplace=True)
inputs.Age = inputs.Age.fillna(inputs.Age.mean())

X_train, X_test, y_train, y_test = train_test_split(inputs, target,
test_size=0.3)

model = GaussianNB()
model.fit(X_train, y_train)

print(model.predict(X_test[0:5]))
print(model.predict_proba(X_test[:10]))
```

### Output

```
[0 1 1 1 0]
[[9.36162489e-01 6.38375110e-02]
 [4.15057870e-01 5.84942130e-01]
 [1.59954910e-10 1.00000000e+00]
 [2.61425161e-01 7.38574839e-01]
 [9.38899635e-01 6.11003653e-02]]
```

## Data Science – Machine Learning – Naïve Bayes Classifier

---

**Program Name**

Cross validation score

demo14.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import cross_val_score

df = pd.read_csv('titanic.csv', usecols = ['Pclass', 'Sex', 'Age', 'Fare',
'Survived'])

inputs = df.drop('Survived', axis='columns')
target = df.Survived
dummies = pd.get_dummies(inputs.Sex, dtype = int)
inputs = pd.concat([inputs, dummies], axis = 'columns')
inputs.drop(['Sex', 'male'],axis='columns', inplace=True)
inputs.Age = inputs.Age.fillna(inputs.Age.mean())

X_train, X_test, y_train, y_test = train_test_split(inputs, target,
test_size=0.3)

model = GaussianNB()
model.fit(X_train, y_train)

model.predict(X_test[0:5])

print(cross_val_score(GaussianNB(),X_train, y_train, cv=5))
```

**Output**

```
[ 0.768      0.792      0.776      0.80645161  0.75806452 ]
```

**31. Data Science – Machine Learning – GridSearchCV****Contents**

<b>1. Introduction</b> .....	2
<b>2. Hyper parameter tuning</b> .....	3
<b>3. Problem and solution</b> .....	4
<b>4. Hyper parameter tuning</b> .....	5
<b>5. Kernel Functions</b> .....	6
<b>6. Ways to tune parameters</b> .....	8
<b>7. RandomizedSearchCV</b> .....	18

**31. Data Science – Machine Learning – GridSearchCV****1. Introduction**

- ✓ While building a Machine learning model we always define two things that are,
  - Model parameters
  - Model hyperparameters of a predictive algorithm.
- ✓ Model parameters are the ones that are an internal part of the model and their value is computed automatically.
  - Support vector machine.
- ✓ Hyperparameters are the ones that can be manipulated by the programmer to improve the performance of the model like the learning rate.
- ✓ Different parts of a hyperparameter approaches,
  - GridSearchCV
  - RandomizedSearchCV

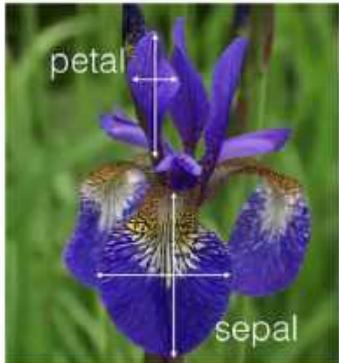
## 2. Hyper parameter tuning

- ✓ Hyperparameter tuning is the process of tuning the parameters while building machine learning models.
- ✓ These parameters are defined by programmer wish; machine learning algorithms never learn these parameters.
- ✓ If parameters are tuned then we will get good performance to the model
- ✓ We define the hyperparameter as shown below for the random forest classifier model.
- ✓ These parameters are tuned randomly and results are checked.

```
o RandomForestRegressor(bootstrap=True, ccp_alpha=0.0,
criterion='mse', max_depth=None, max_features='auto',
max_leaf_nodes=None, max_samples=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=None,
oob_score=False, random_state=None, verbose=0,
warm_start=False)
```

### 3. Problem and solution

#### sklearn iris flower dataset



features				target label
sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	flower
4.6	3.2	1.4	0.2	setosa
5.3	3.7	1.5	0.2	setosa
5.0	3.3	1.4	0.2	setosa
7.0	3.2	4.7	1.4	versicolor
6.4	3.2	4.5	1.5	versicolor

- ✓ Above problem we can solve by using different algorithms
  - LogisticRegression
  - SVM
  - Decision Tree
  - RandomForest
  - NaiveBayes
- ✓ If we are trying to use SVM then it having different parameters

## Data Science – Machine Learning – GridSearchCV

### 4. Hyper parameter tuning

- ✓ The process of choosing parameters called as hyper parameter tuning

SVM

example    `model = svm.SVC(kernel='rbf',C=30,gamma='auto')`

Parameter	Values
Kernel	'rbf', 'linear', 'poly'
C	Integer
Gamma	float

## 5. Kernel Functions

- ✓ SVM algorithms use a set of mathematical functions that are defined as the kernel.
  - The function of kernel is to take data as input and transform it into the required form.
  - For example linear, nonlinear, polynomial, radial basis function (RBF), and sigmoid.



## Data Science – Machine Learning – GridSearchCV

---

**Program Name** Loading iris dataset  
demo1.py

```
from sklearn import datasets
import pandas as pd

iris = datasets.load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['flower'] = iris.target
df['flower'] = df['flower'].apply(lambda x: iris.target_names[x])

print(df)
```

### Output

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	flower
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
..	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

[150 rows x 5 columns]

## Data Science – Machine Learning – GridSearchCV

---

### 6. Ways to tune parameters

- ✓ Approach 1: Use train\_test\_split and manually tune parameters by trial and error
- ✓ Approach 2: Use K Fold Cross validation
- ✓ Approach 3: Use GridSearchCV

**Program**      Approach 1: Use train\_test\_split and manually tune parameters by trial and error  
**Name**           demo2.py

```
from sklearn import svm, datasets
import pandas as pd
from sklearn.model_selection import train_test_split

iris = datasets.load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['flower'] = iris.target
df['flower'] = df['flower'].apply(lambda x: iris.target_names[x])

X_train, X_test, y_train, y_test = train_test_split(iris.data,
iris.target, test_size = 0.3)

model = svm.SVC(kernel = 'rbf', C = 30, gamma = 'auto')
model.fit(X_train, y_train)

print(model.score(X_test, y_test))
```

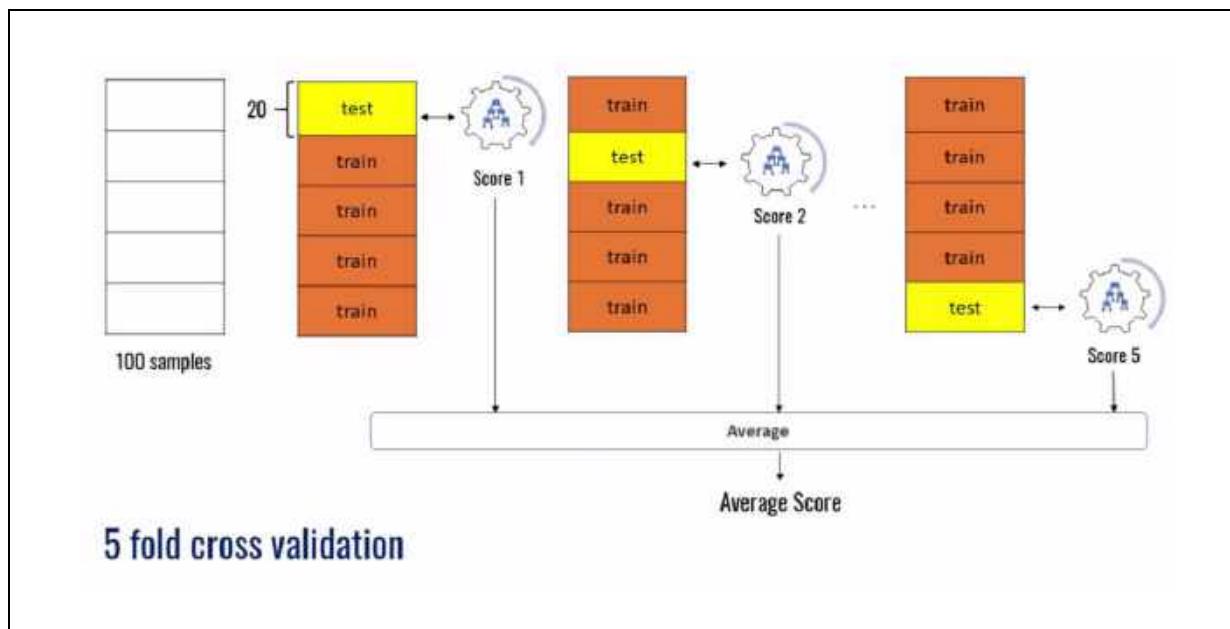
### Output

```
C:\Users\admin\Desktop>py demo.py
0.9111111111111111

C:\Users\admin\Desktop>py demo.py
0.9777777777777777

C:\Users\admin\Desktop>py demo.py
0.9555555555555556
```

## Data Science – Machine Learning – GridSearchCV



## Data Science – Machine Learning – GridSearchCV

Program Name

Approach 2: Use K Fold Cross validation

demo3.py

```
from sklearn import svm, datasets
import pandas as pd
from sklearn.model_selection import cross_val_score

iris = datasets.load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['flower'] = iris.target
df['flower'] = df['flower'].apply(lambda x: iris.target_names[x])

sc1 = cross_val_score(svm.SVC(kernel='linear', C=10,
gamma='auto'), iris.data, iris.target, cv=5)

print(sc1)
```

Output

```
[1.          1.          0.9         0.96666667 1.        ]
```

## Data Science – Machine Learning – GridSearchCV

**Program Name**

Approach 2: Use K Fold Cross validation

demo4.py

```
from sklearn import svm, datasets
import pandas as pd
from sklearn.model_selection import cross_val_score

iris = datasets.load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['flower'] = iris.target
df['flower'] = df['flower'].apply(lambda x: iris.target_names[x])

sc2 = cross_val_score(svm.SVC(kernel='rbf', C=10, gamma='auto'),
iris.data, iris.target, cv=5)

print(sc2)
```

**Output**

```
[ 0.96666667 1.           0.96666667 0.96666667 1.           ]
```

## Data Science – Machine Learning – GridSearchCV

Program Name

Approach 2: Use K Fold Cross validation  
demo5.py

```
from sklearn import svm, datasets
import pandas as pd
from sklearn.model_selection import cross_val_score

iris = datasets.load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['flower'] = iris.target
df['flower'] = df['flower'].apply(lambda x: iris.target_names[x])

sc3 = cross_val_score(svm.SVC(kernel='rbf', C=20, gamma='auto'),
iris.data, iris.target, cv=5)

print(sc3)
```

Output

```
[ 0.96666667  1.          0.9          0.96666667  1.          ]
```

## Data Science – Machine Learning – GridSearchCV

---

**Program Name** Approach: Use GridSearchCV  
demo6.py

```
from sklearn import svm, datasets
import pandas as pd
from sklearn.model_selection import GridSearchCV

iris = datasets.load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['flower'] = iris.target
df['flower'] = df['flower'].apply(lambda x: iris.target_names[x])

d = {
    'C': [1, 10, 20],
    'kernel': ['rbf', 'linear']
}

clf = GridSearchCV(svm.SVC(gamma='auto'), d, cv = 5,
return_train_score = False)

clf.fit(iris.data, iris.target)

print(clf.cv_results_)
```

### Output

```
{'mean_fit_time': array([0.00124702, 0.          , 0.00060325, 0.00035907, 0.00067601,
 0.00019941]), 'std_fit_time': array([0.00203303, 0.          , 0.00120649, 0.00048875, 0.00093714,
 0.00039883]), 'mean_score_time': array([0.00019922, 0.00120854, 0.00019917, 0.00100141, 0.
 0.0012084]), 'std_score_time': array([0.00039845, 0.00195683, 0.00039835, 0.00200281, 0.
 0.00195764]), 'param_C': masked_array(data=[1, 1, 10, 10, 20, 20],
 mask=[False, False, False, False, False, False],
 fill_value='?'),
 dtype=object), 'param_kernel': masked_array(data=['rbf', 'linear', 'rbf', 'linear', 'rbf',
 'linear'],
 mask=[False, False, False, False, False, False],
 fill_value='?'),
 dtype=object), 'params': [{ 'C': 1, 'kernel': 'rbf'}, { 'C': 1, 'kernel': 'linear'}, { 'C': 10,
 'kernel': 'rbf'}, { 'C': 10, 'kernel': 'linear'}, { 'C': 20, 'kernel': 'rbf'}, { 'C': 20, 'kernel': 'lin
 ear'}], 'split0_test_score': array([0.96666667, 0.96666667, 0.96666667, 1.          , 0.96666667,
 1.          ]), 'split1_test_score': array([1., 1., 1., 1., 1., 1.]), 'split2_test_score': array([
 0.96666667, 0.96666667, 0.96666667, 0.9       , 0.9       ,
 0.9       ]), 'split3_test_score': array([0.96666667, 0.96666667, 0.96666667, 0.96666667, 0.9666
 6667, 0.96666667] )}
```

## Data Science – Machine Learning – GridSearchCV

---

**Program Name** Approach: Use GridSearchCV, creating DataFrame with results  
demo7.py

```
from sklearn import svm, datasets
import pandas as pd
from sklearn.model_selection import GridSearchCV

iris = datasets.load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['flower'] = iris.target
df['flower'] = df['flower'].apply(lambda x: iris.target_names[x])

d = {
    'C': [1, 10, 20],
    'kernel': ['rbf', 'linear']
}

clf = GridSearchCV(svm.SVC(gamma = 'auto'), d, cv = 5,
return_train_score = False)

clf.fit(iris.data, iris.target)

df = pd.DataFrame(clf.cv_results_)

print(df)
```

### Output

	mean_fit_time	std_fit_time	mean_score_time	...	mean_test_score	std_test_score	rank_test_score
0	0.000758	0.001515	0.001209	...	0.980000	0.016330	1
1	0.001207	0.001955	0.000000	...	0.980000	0.016330	1
2	0.000199	0.000399	0.000000	...	0.980000	0.016330	1
3	0.000419	0.000536	0.000000	...	0.973333	0.038873	4
4	0.001052	0.002105	0.000000	...	0.966667	0.036515	5
5	0.000197	0.000393	0.001008	...	0.966667	0.042164	6

[6 rows x 15 columns]

## Data Science – Machine Learning – GridSearchCV

---

**Program Name** Approach: Use GridSearchCV, creating DataFrame with results  
demo8.py

```
from sklearn import svm, datasets
import pandas as pd
from sklearn.model_selection import GridSearchCV

iris = datasets.load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['flower'] = iris.target
df['flower'] = df['flower'].apply(lambda x: iris.target_names[x])

d = {
    'C': [1,10,20],
    'kernel': ['rbf', 'linear']
}

clf = GridSearchCV(svm.SVC(gamma='auto'), d, cv=5,
return_train_score = False)

clf.fit(iris.data, iris.target)

df = pd.DataFrame(clf.cv_results_)

result = df[['param_C', 'param_kernel', 'mean_test_score']]

print(result)
```

### Output

	param_C	param_kernel	mean_test_score
0	1	rbf	0.980000
1	1	linear	0.980000
2	10	rbf	0.980000
3	10	linear	0.973333
4	20	rbf	0.966667
5	20	linear	0.966667

## Data Science – Machine Learning – GridSearchCV

**Program Name** Approach: Use GridSearchCV.  
demo9.py

```
from sklearn import svm, datasets
import pandas as pd
from sklearn.model_selection import GridSearchCV

iris = datasets.load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['flower'] = iris.target
df['flower'] = df['flower'].apply(lambda x: iris.target_names[x])

d = {
    'C': [1,10,20],
    'kernel': ['rbf', 'linear']
}

clf = GridSearchCV(svm.SVC(gamma='auto'), d, cv=5,
return_train_score = False)

clf.fit(iris.data, iris.target)

df = pd.DataFrame(clf.cv_results_)

print(clf.best_params_)
```

**Output**

```
{'C': 1, 'kernel': 'rbf'}
```

## Data Science – Machine Learning – GridSearchCV

**Program Name** Approach: Use GridSearchCV.  
demo10.py

```
from sklearn import svm, datasets
import pandas as pd
from sklearn.model_selection import GridSearchCV

iris = datasets.load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['flower'] = iris.target
df['flower'] = df['flower'].apply(lambda x: iris.target_names[x])

d = {
    'C': [1,10,20],
    'kernel': ['rbf', 'linear']
}

clf = GridSearchCV(svm.SVC(gamma='auto'), d, cv=5,
return_train_score = False)

clf.fit(iris.data, iris.target)

df = pd.DataFrame(clf.cv_results_)

print(clf.best_score_)
```

**Output**

```
0.9800000000000001
```

### 7. RandomizedSearchCV

- ✓ Use RandomizedSearchCV to reduce number of iterations and with random combination of parameters.
- ✓ This is useful when you have too many parameters to try and your training time is longer. It helps reduce the cost of computation

## Data Science – Machine Learning – GridSearchCV

---

**Program Name** Approach: Use RandomizedSearchCV  
demo11.py

```

from sklearn import svm, datasets
import pandas as pd
from sklearn.model_selection import RandomizedSearchCV

iris = datasets.load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['flower'] = iris.target
df['flower'] = df['flower'].apply(lambda x: iris.target_names[x])

d = {
    'C': [1, 10, 20],
    'kernel': ['rbf', 'linear']
}

rs = RandomizedSearchCV(svm.SVC(gamma = 'auto'), d, cv = 5,
                        return_train_score = False,
                        n_iter=2
)

rs.fit(iris.data, iris.target)
cols = ['param_C', 'param_kernel', 'mean_test_score']
df = pd.DataFrame(rs.cv_results_)[cols]

print(df)

```

### Output

	param_C	param_kernel	mean_test_score
0	10	rbf	0.980000
1	20	rbf	0.966667

**32. Data Science – Machine Learning – XGBoost****Contents**

<b>1. Introduction</b> .....	2
<b>2. Why Use XGBoost?</b> .....	2
<b>3. Install XGBoost</b> .....	2
<b>4. Dataset explanation</b> .....	3
<b>5. Input and output from the Dataset</b> .....	4
5.1. Input Variables (X):.....	4
5.2. Output Variables (y):.....	4
<b>6. Label Encoding</b> .....	15
<b>7. Feature Importance with XGBoost and Feature Selection</b> .....	24

**32. Data Science – Machine Learning – XGBoost****1. Introduction**

- ✓ XGBoost stands for eXtreme Gradient Boosting.
- ✓ XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework.
- ✓ It combines the predictions from two or more models.
- ✓ So that it get the better performance.

**2. Why Use XGBoost?**

- ✓ The two reasons to use XGBoost are also the two goals of the project:
  - Execution Speed.
  - Model Performance.

**3. Install XGBoost**

- ✓ pip install xgboost

## Data Science – Machine Learning – XGBoost

---

### 4. Dataset explanation

- ✓ We are going to work with **pima-indians-diabetes.csv**
- ✓ This Dataset is related to health care domain.
- ✓ Pima Indians are a Native American group that lives in Mexico and Arizona, USA.
- ✓ It describes patient medical record data for Pima Indians and whether they had a diabetes within five years.
- ✓ The Pima Indian Diabetes dataset consisting of Pima Indian females 21 years and older is a popular benchmark dataset.
- ✓ It is a binary classification problem (onset of diabetes as 1 or not as 0).
- ✓ All of the input variables that describe each patient are numerical.

Feature	Description	Data type	Range
Preg	Number of times pregnant	Numeric	[0, 17]
Gluc	Plasma glucose concentration at 2 Hours in an oral glucose tolerance test (GTIT)	Numeric	[0, 199]
BP	Diastolic Blood Pressure (mm Hg)	Numeric	[0, 122]
Skin	Triceps skin fold thickness (mm)	Numeric	[0, 99]
Insulin	2-Hour Serum insulin ( $\mu$ h/ml)	Numeric	[0, 846]
BMI	Body mass index [weight in kg/(Height in m)]	Numeric	[0, 67.1]
DPF	Diabetes pedigree function	Numeric	[0.078, 2.42]
Age	Age (years)	Numeric	[21, 81]
Outcome	Binary value indicating non-diabetic /diabetic	Factor	[0,1]

## 5. Input and output from the Dataset

### 5.1. Input Variables (X):

- ✓ 1. Number of times pregnant
- ✓ 2. Plasma glucose concentration at 2 hours in an oral glucose tolerance test
- ✓ 3. Diastolic blood pressure (mm Hg)
- ✓ 4. Triceps skin fold thickness (mm)
- ✓ 5. 2-hour serum insulin ( $\mu$ IU/ml)
- ✓ 6. Body mass index (weight in kg/(height in m))
- ✓ 7. Diabetes pedigree function
- ✓ 8. Age (years)

### 5.2. Output Variables (y):

- ✓ 1. Class variable (0 or 1)

## Data Science – Machine Learning – XGBoost

**Program Name** Loading csv file  
**demo1.py**  
**Input file** pima-indians-diabetes.csv

```
from numpy import loadtxt

dataset = loadtxt('pima-indians-diabetes.csv', delimiter = ',')

print(dataset)
```

### Output

```
[[ 6.    148.    72.    ...   0.627   50.    1.    ]
 [ 1.    85.    66.    ...   0.351   31.    0.    ]
 [ 8.    183.    64.    ...   0.672   32.    1.    ]
 ...
 [ 5.    121.    72.    ...   0.245   30.    0.    ]
 [ 1.    126.    60.    ...   0.349   47.    1.    ]
 [ 1.    93.    70.    ...   0.315   23.    0.    ]]
```

## Data Science – Machine Learning – XGBoost

**Program** Preparing input (X) and output (y) variables

**Name** demo2.py

**Input file** pima-indians-diabetes.csv

```
from numpy import loadtxt

dataset = loadtxt('pima-indians-diabetes.csv', delimiter = ',')

X = dataset[:, 0:8]
y = dataset[:, 8]

print("Input and output")
```

**Output**

Input and output

## Data Science – Machine Learning – XGBoost

**Program Name** Splitting data into train and test datasets  
**Input file** demo3.py  
pima-indians-diabetes.csv

```
from sklearn.model_selection import train_test_split
from numpy import loadtxt

dataset = loadtxt('pima-indians-diabetes.csv', delimiter = ',')

X = dataset[:, 0:8]
y = dataset[:, 8]

seed = 7

test_size = 0.33

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size = test_size,
    random_state = seed
)

print("Train and Test dataset")
```

### Output

Train and Test dataset

## Data Science – Machine Learning – XGBoost

**Program Name** Model creation  
**demo4.py**  
**Input file** pima-indians-diabetes.csv

```
from sklearn.model_selection import train_test_split
from numpy import loadtxt
from xgboost import XGBClassifier

dataset = loadtxt('pima-indians-diabetes.csv', delimiter = ',')

X = dataset[:, 0:8]
y = dataset[:, 8]

seed = 7

test_size = 0.33

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size = test_size,
    random_state = seed
)

model = XGBClassifier()
print("Model created")
```

### Output

Model created

## Data Science – Machine Learning – XGBoost

**Program Name** Train the model  
**Name** demo5.py  
**Input file** pima-indians-diabetes.csv

```
from sklearn.model_selection import train_test_split
from numpy import loadtxt
from xgboost import XGBClassifier

dataset = loadtxt('pima-indians-diabetes.csv', delimiter = ',')

X = dataset[:, 0:8]
y = dataset[:, 8]

seed = 7

test_size = 0.33

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size = test_size,
    random_state = seed
)

model = XGBClassifier()
model.fit(X_train, y_train)
print("Model trained")
```

### Output

Model created

## Data Science – Machine Learning – XGBoost

**Program** Model prediction  
**Name** demo6.py  
**Input file** pima-indians-diabetes.csv

```
from sklearn.model_selection import train_test_split
from numpy import loadtxt
from xgboost import XGBClassifier

dataset = loadtxt('pima-indians-diabetes.csv', delimiter = ',')

X = dataset[:, 0:8]
y = dataset[:, 8]

seed = 7

test_size = 0.33

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size = test_size,
    random_state = seed
)

model = XGBClassifier()
model.fit(X_train, y_train)

predictions = model.predict(X_test)

print(X_test)
print(predictions)
```

## Data Science – Machine Learning – XGBoost

### Output

```
[[ 1.    90.    62.    ...   27.2    0.58   24.   ]
 [ 7.   181.    84.    ...   35.9    0.586   51.   ]
 [ 13.   152.    90.    ...   26.8    0.731   43.   ]
 ...
 [ 4.   118.    70.    ...   44.5    0.904   26.   ]
 [ 7.   152.    88.    ...   50.    0.337   36.   ]
 [ 7.   168.    88.    ...   38.2    0.787   40.   ]]
[[0 1 1 0 1 1 0 0 1 0 1 0 1 1 0 0 0 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 0
 0 1 0 1 1 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 1 1 1 0 0
 0 0 0 0 0 0 0 0 0 1 1 1 0 1 0 1 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0
 1 0 0 1 1 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0
 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 1 0 1 0 0 1 0 1 0 1 0 1 0 0 0 1
 1 0 0 0 1 1 0 1 1 0 0 1 0 0 1 1 1 0 1 0 0 0 1 1 0 0 1 0 0 1 0 1 0 0 1 0 1
 1 0 1 1 0 1 1 0 1 1 0 0 0 0 1 1 0 0 1 0 0 1 0 1 0 0 0 0 1 0 1 0 0 0 1 0 1]
```

## Data Science – Machine Learning – XGBoost

**Program** Model prediction  
**Name** demo7.py  
**Input file** pima-indians-diabetes.csv

```
from sklearn.model_selection import train_test_split
from numpy import loadtxt
from xgboost import XGBClassifier

dataset = loadtxt('pima-indians-diabetes.csv', delimiter = ',')

X = dataset[:, 0:8]
y = dataset[:, 8]

seed = 7

test_size = 0.33

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size = test_size,
    random_state = seed
)

model = XGBClassifier()
model.fit(X_train, y_train)

predictions = model.predict(X_test)

print(X_test)
print(predictions)
```

## Output

```
[[ 1.    90.    62.    ...  27.2    0.58   24.   ]
 [ 7.   181.    84.    ...  35.9    0.586   51.   ]
 [ 13.   152.    90.    ...  26.8    0.731   43.   ]
 ...
 [ 4.   118.    70.    ...  44.5    0.904   26.   ]
 [ 7.   152.    88.    ...  50.    0.337   36.   ]
 [ 7.   168.    88.    ...  38.2    0.787   40.   ]]
[[0 1 1 0 1 1 0 0 1 0 1 0 1 1 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 1 0
0 1 0 1 1 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 1 1 0 0
0 0 0 0 0 0 0 0 0 1 1 1 0 1 0 1 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0
1 0 0 1 1 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0
1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 1 0 1 0 0 1 0 1 0 1 0 1 0 0 0 1
1 0 0 0 1 1 0 1 1 0 0 1 0 0 1 1 1 0 1 0 0 0 1 1 0 0 1 0 0 1 0 1 0 0 1 0 0 1
1 0 1 1 0 1 1 0 1 1 0 0 0 0 1 1 0 0 1 0 0 1 0 1 0 0 0 0 1 0 1 0 0 0 1 0 1]
```

## Data Science – Machine Learning – XGBoost

**Program Name** Check model accuracy  
**Input file** demo8.py  
pima-indians-diabetes.csv

```
from sklearn.model_selection import train_test_split
from numpy import loadtxt
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score

dataset = loadtxt('pima-indians-diabetes.csv', delimiter = ',')

X = dataset[:, 0:8]
y = dataset[:, 8]

seed = 7

test_size = 0.33

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size = test_size,
    random_state = seed
)

model = XGBClassifier()
model.fit(X_train, y_train)

predictions = model.predict(X_test)

accuracy = accuracy_score(y_test, predictions)
print(accuracy)
```

### Output

0.740157

## 6. Label Encoding

- ✓ The iris flowers classification problem is an example of a problem that has a string class value.
- ✓ XGBoost cannot model this problem because output variables are strings, we need to convert output variables be numeric.
- ✓ We can easily convert the string values to integer values using the LabelEncoder.
- ✓ The three class values
  - (Iris-setosa, Iris-versicolor, Iris-virginica) are mapped to the integer values (0, 1, 2).

**Program Name** Loading csv file

**Name** demo1.py

**Input file** iris.csv

```
from pandas import read_csv

data = read_csv('iris.csv', header = None)
dataset = data.values

print(dataset[0:5])
```

**Output**

```
[[5.1 3.5 1.4 0.2 'Iris-setosa']
 [4.9 3.0 1.4 0.2 'Iris-setosa']
 [4.7 3.2 1.3 0.2 'Iris-setosa']
 [4.6 3.1 1.5 0.2 'Iris-setosa']
 [5.0 3.6 1.4 0.2 'Iris-setosa']]
```

## Data Science – Machine Learning – XGBoost

Program Data preparation  
Name demo2.py  
Input file iris.csv

```
from pandas import read_csv

data = read_csv('iris.csv', header = None)
dataset = data.values

X = dataset[:,0:4]
Y = dataset[:,4]

print(X[0:5])
print()
print(Y[0:5])
```

### Output

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.0 1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.0 3.6 1.4 0.2]]

['Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa']
```

Program	Encoding flower names
Name	demo3.py
Input file	iris.csv

```
from pandas import read_csv
from sklearn.preprocessing import LabelEncoder

data = read_csv('iris.csv', header = None)
dataset = data.values

X = dataset[:,0:4]
Y = dataset[:,4]

label_encoder = LabelEncoder()
label_encoder = label_encoder.fit(Y)
label_encoded_y = label_encoder.transform(Y)

print(label_encoded_y)
```

## Output

## Data Science – Machine Learning – XGBoost

**Program Name** Splitting the data  
**Input file** demo4.py  
iris.csv

```
from pandas import read_csv
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

data = read_csv('iris.csv', header = None)
dataset = data.values

X = dataset[:,0:4]
Y = dataset[:,4]

label_encoder = LabelEncoder()
label_encoder = label_encoder.fit(Y)
label_encoded_y = label_encoder.transform(Y)

seed = 7
test_size = 0.33

X_train, X_test, y_train, y_test = train_test_split(X,
label_encoded_y, test_size=test_size, random_state = seed)

print("Data splitting done")
```

**Output**

Data splitting done

## Data Science – Machine Learning – XGBoost

Program Model creation  
Name demo5.py  
Input file iris.csv

```
from pandas import read_csv
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier

data = read_csv('iris.csv', header = None)
dataset = data.values

X = dataset[:,0:4]
Y = dataset[:,4]

label_encoder = LabelEncoder()
label_encoder = label_encoder.fit(Y)
label_encoded_y = label_encoder.transform(Y)

seed = 7
test_size = 0.33

X_train, X_test, y_train, y_test = train_test_split(X,
label_encoded_y, test_size=test_size, random_state = seed)

model = XGBClassifier()

print("Model created")
```

### Output

Model created

## Data Science – Machine Learning – XGBoost

Program Name Model training

Name demo6.py

Input file iris.csv

```
from pandas import read_csv
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier

data = read_csv('iris.csv', header = None)
dataset = data.values

X = dataset[:,0:4]
Y = dataset[:,4]

label_encoder = LabelEncoder()
label_encoder = label_encoder.fit(Y)
label_encoded_y = label_encoder.transform(Y)

seed = 7
test_size = 0.33

X_train, X_test, y_train, y_test = train_test_split(X,
label_encoded_y, test_size=test_size, random_state = seed)

model = XGBClassifier()
model.fit(X_train, y_train)

print("Model trained")
```

### Output

Model trained

## Data Science – Machine Learning – XGBoost

Program      Prediction  
Name          demo7.py  
Input file    iris.csv

```
from pandas import read_csv
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from xgboost import XGBClassifier

data = read_csv('iris.csv', header = None)
dataset = data.values

X = dataset[:,0:4]
Y = dataset[:,4]

label_encoder = LabelEncoder()
label_encoder = label_encoder.fit(Y)
label_encoded_y = label_encoder.transform(Y)

seed = 7
test_size = 0.33

X_train, X_test, y_train, y_test = train_test_split(X,
label_encoded_y, test_size=test_size, random_state = seed)

model = XGBClassifier()
model.fit(X_train, y_train)

predictions = model.predict(X_test)

print(X_test[0:5])
print()
print(predictions[0:5])
```

## Data Science – Machine Learning – XGBoost

### Output

```
[[5.9 3.0 5.1 1.8]
 [5.4 3.0 4.5 1.5]
 [5.0 3.5 1.3 0.3]
 [5.6 3.0 4.5 1.5]
 [4.9 2.5 4.5 1.7]]

[2 1 0 1 1]
```

## Data Science – Machine Learning – XGBoost

---

**Program Name** Evaluate predictions  
**Input file** demo8.py  
 iris.csv

```

from pandas import read_csv
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from xgboost import XGBClassifier

data = read_csv('iris.csv', header = None)
dataset = data.values

X = dataset[:,0:4]
Y = dataset[:,4]

label_encoder = LabelEncoder()
label_encoder = label_encoder.fit(Y)
label_encoded_y = label_encoder.transform(Y)

seed = 7
test_size = 0.33

X_train, X_test, y_train, y_test = train_test_split(X,
label_encoded_y, test_size=test_size, random_state = seed)

model = XGBClassifier()
model.fit(X_train, y_train)

predictions = model.predict(X_test)

accuracy = accuracy_score(y_test, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
  
```

**Output**

Accuracy: 92.00%

## 7. Feature Importance with XGBoost and Feature Selection

- ✓ XGBoost is helping to get important features from the existing features
- ✓ We can see like every features how much score it having.

**Program Name** Feature Importance with XGBoost

**Name** demo1.py

**Input file** pima-indians-diabetes.csv

```
from numpy import loadtxt
from xgboost import XGBClassifier
from matplotlib import pyplot

dataset = loadtxt('pima-indians-diabetes.csv', delimiter = ',')

X = dataset[:, 0:8]
y = dataset[:, 8]

model = XGBClassifier()

model.fit(X, y)

print(model.feature_importances_)
```

**Output**

```
[0.10621195 0.24240209 0.08803371 0.07818192 0.10381888
 0.14867325 0.10059208 0.13208608]
```

## Data Science – Machine Learning – XGBoost

**Program Name** Plotting the Feature Importance with XGBoost  
**Input file** demo2.py  
pima-indians-diabetes.csv

```
from numpy import loadtxt
from xgboost import XGBClassifier
from matplotlib import pyplot

dataset = loadtxt('pima-indians-diabetes.csv', delimiter = ',')

X = dataset[:, 0:8]
y = dataset[:, 8]

model = XGBClassifier()

model.fit(X, y)

print(model.feature_importances_)

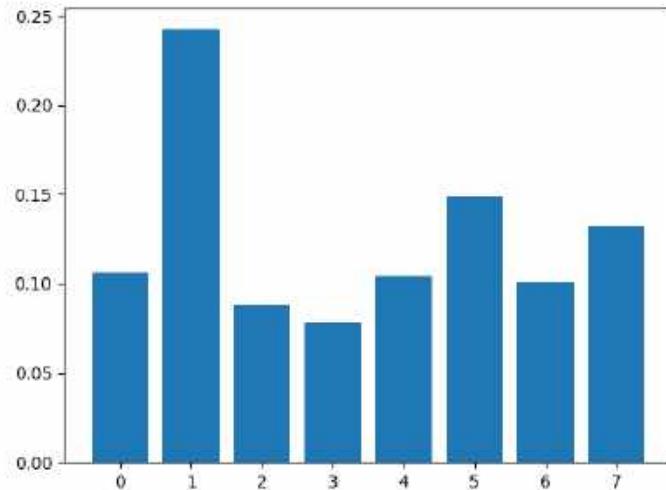
r = range(len(model.feature_importances_))
f_imp = model.feature_importances_

pyplot.bar(r, f_imp)
pyplot.show()
```

## Data Science – Machine Learning – XGBoost

### Output

```
[0.10621195 0.24240209 0.08803371 0.07818192 0.10381888  
 0.14867325 0.10059208 0.13208608]
```



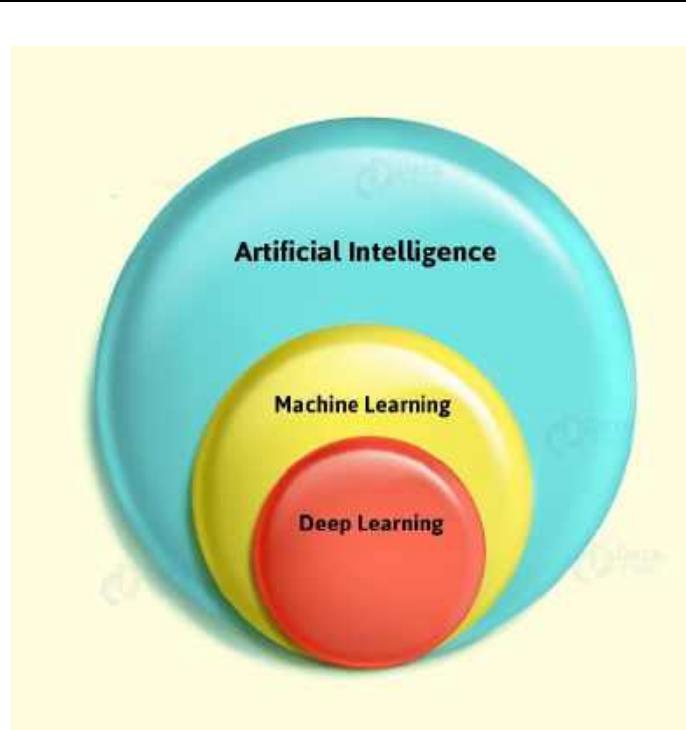
## 1. Deep learning – Introduction

### Contents

<b>1. Artificial intelligence vs Machine Learning vs Deep Learning.....</b>	2
1.1. Machine learning .....	2
1.2. Deep learning.....	3
1.3. Artificial intelligence .....	3
<b>2. Different models .....</b>	4
<b>3. Let's understand Deep learning.....</b>	5
<b>4. Differences between Deep Learning and Machine Learning.....</b>	6
4.1. Data .....	6
4.2. Computational Hardware CPU, GPU.....	6
4.3. CPU.....	6
4.4. GPU .....	7
<b>5. Feature selection.....</b>	8
<b>6. Performance .....</b>	8
<b>7. Neural networks.....</b>	9
<b>8. Types of neural networks.....</b>	10
8.1. Deep learning applications .....	10

## 1. Deep learning – Introduction

### 1. Artificial intelligence vs Machine Learning vs Deep Learning



#### 1.1. Machine learning

- ✓ Machine learning is a part of artificial intelligence
- ✓ Machine Learning is a technique to learn from the data and apply the prediction on new data

#### Examples

- ✓ Amazon using machine learning to give better product choice recommendations to their customers based on their preferences.
- ✓ Netflix uses machine learning to give better suggestions to their users of the TV series or movie or shows that they would like to watch & many more

## Data Science – Deep Learning - Introduction

---

### 1.2. Deep learning

- ✓ Deep learning is a subset of machine learning.
- ✓ The main difference between deep and machine learning is, machine learning models works better but the model still needs some guidance.
- ✓ If a machine learning model returns an inaccurate prediction then the programmer needs to fix that problem explicitly.
- ✓ In the case of deep learning, the model does it by itself.

#### Example

- ✓ Automatic car driving system is a good example of deep learning.

### 1.3. Artificial intelligence

- ✓ AI means the ability of computer program to function like a human brain.
- ✓ Machine learning and deep learning are the subsets of AI
- ✓ The MOTO of AI is to replicate a human brain, the way a human brain thinks, works and functions.
- ✓ Currently AI is not yet fully implemented but we are very near to establish that too.

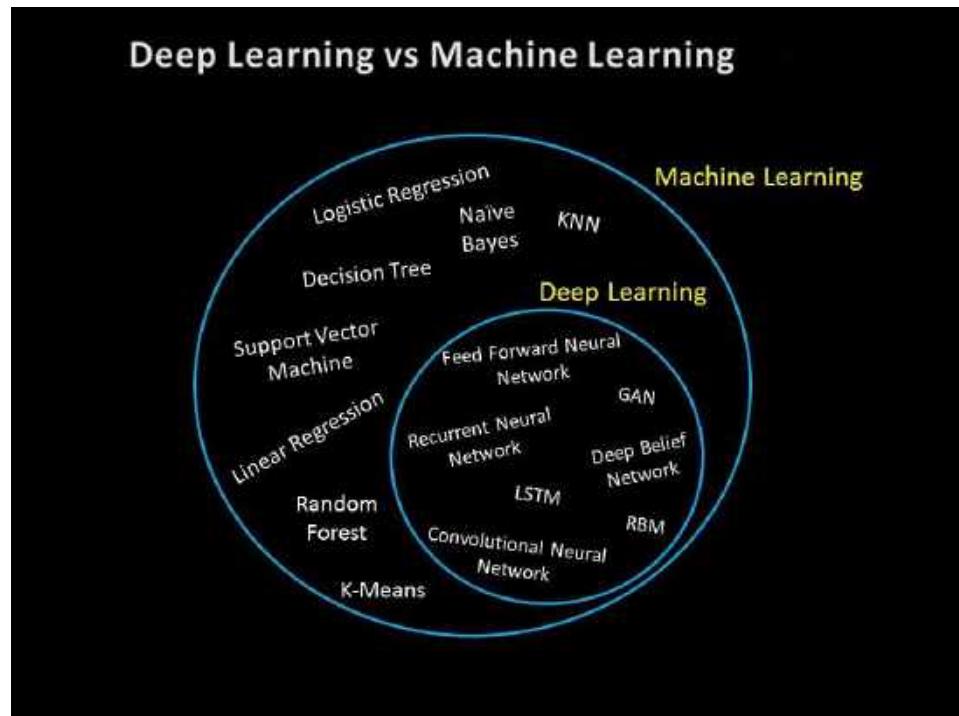
#### Example

- ✓ Sophia, the most advanced AI model present today.

## Data Science – Deep Learning - Introduction

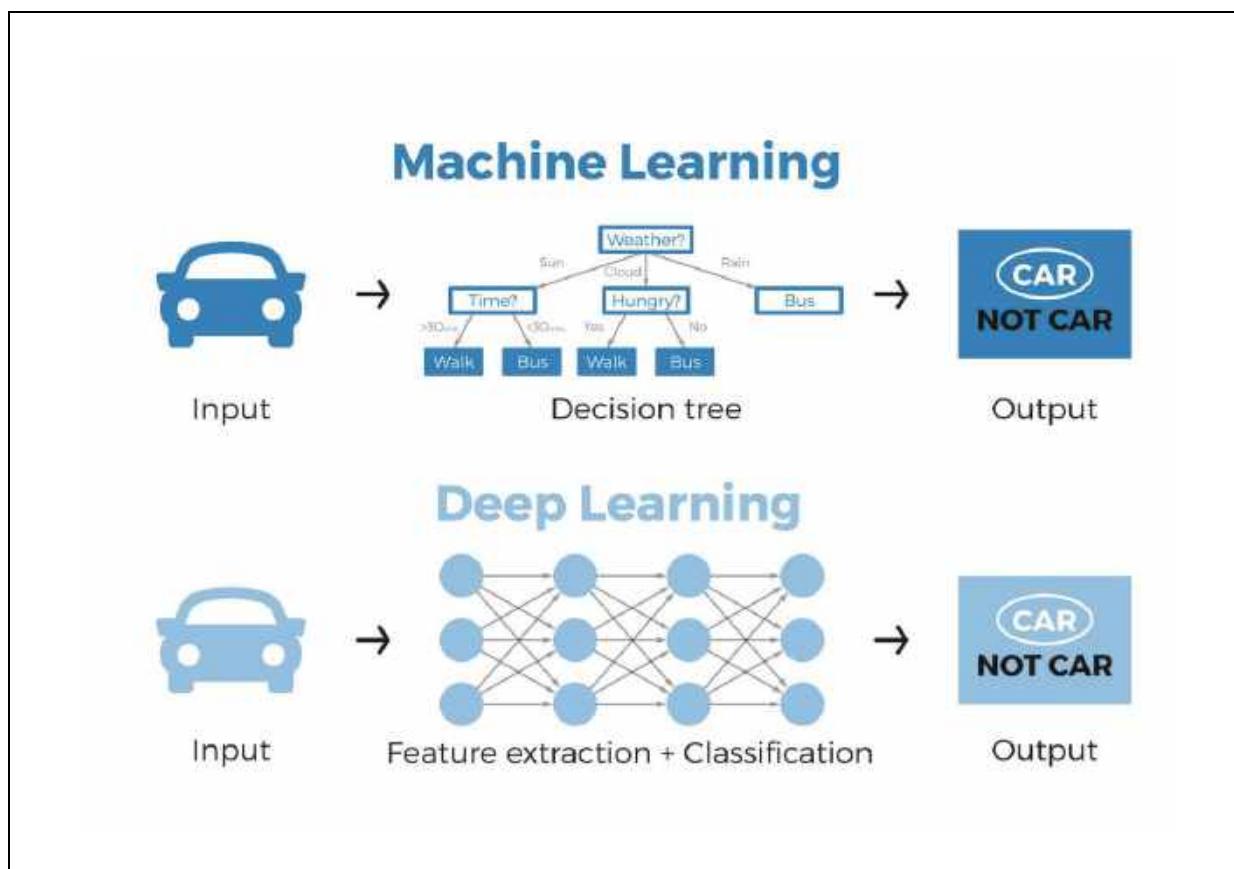
### 2. Different models

- ✓ In ML we have worked with different models like linear regression, logistic regression, KNN, Decision Tree & etc



### 3. Let's understand Deep learning

- ✓ Deep learning is a subset of machine learning mainly using Artificial Neural Networks, which are inspired by the human brain.
- ✓ Deep learning is able to capture required features automatically and solving the problems.
- ✓ DL is the technique that comes closest to the way humans learn.
- ✓ Deep learning methods use neural network architecture.
- ✓ That is why deep learning is often referred to as "deep neural networks"



### 4. Differences between Deep Learning and Machine Learning

#### 4.1. Data

- ✓ Deep learning models works with very huge data
- ✓ The more data you give to neural network, it will show much greater accuracy than other machine learning models.

#### 4.2. Computational Hardware CPU, GPU

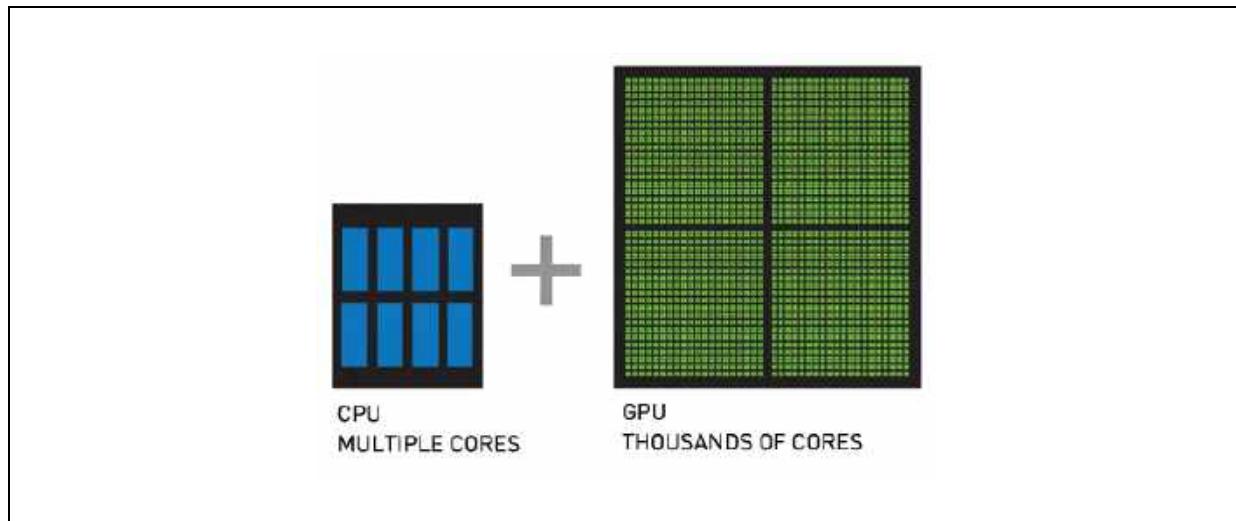
- ✓ ML models works with CPU[Central Processing Units]
- ✓ DL models works with GPU[Graphical Processing Units] and TPS[Tensor Processing Units]
  - These are highly advanced processing systems

#### 4.3. CPU

- ✓ It is Central Processing Unit
- ✓ It is a brain to the computer
- ✓ Processing serial instructions
- ✓ Good at speed during processing
- ✓ Consumes more memory during processing
- ✓ Companies like Intel, ARM and AMD produce the CPU and companies

#### 4.4. GPU

- ✓ It is Graphical Processing Unit
- ✓ Processing parallel instructions
- ✓ Having very good speed compare to CPU
- ✓ Consumes very less memory
- ✓ Companies like Nvidia, AMD's ATI produce the GPU



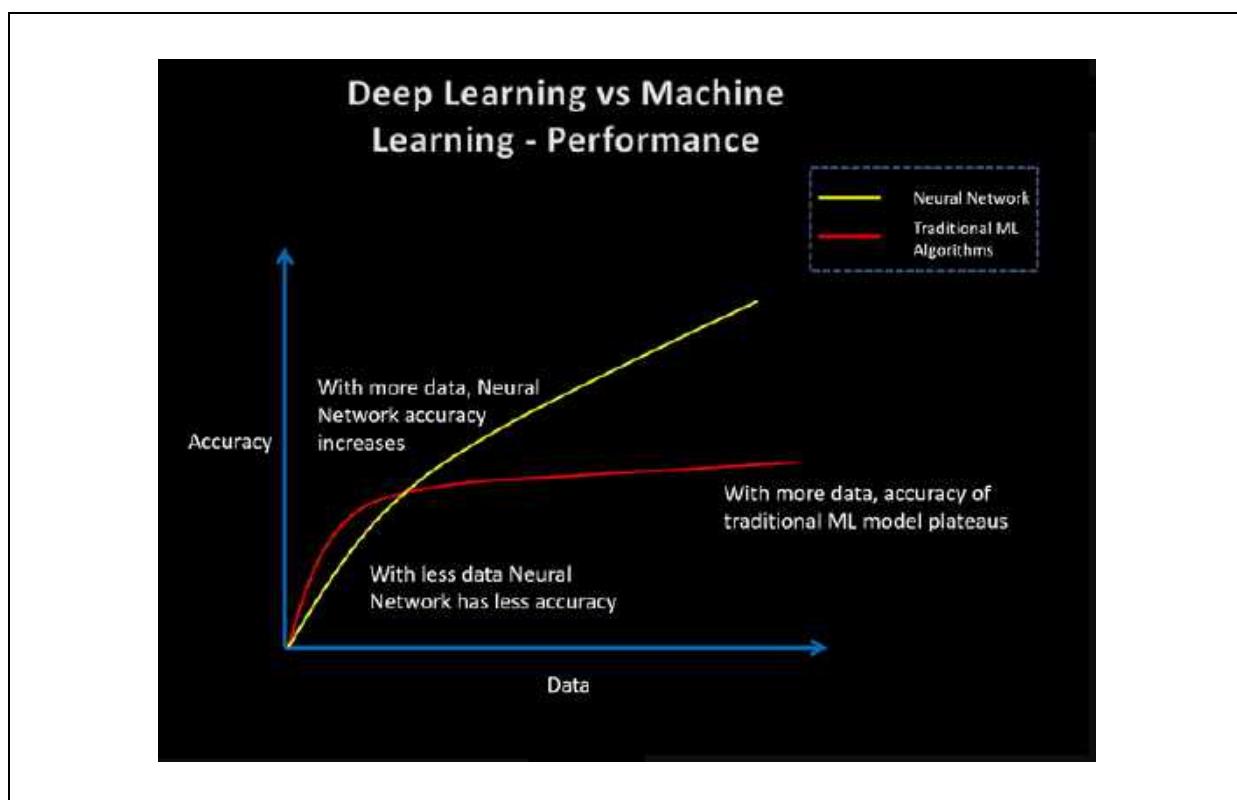
## Data Science – Deep Learning - Introduction

### 5. Feature selection

- ✓ In deep learning neural networks take care of the feature selection automatically.
- ✓ It decides a particular feature is important or not, and reduces the corresponding weights to almost zero.
- ✓ In machine learning algorithm, feature selection plays an important role and we need to do keep focus on this

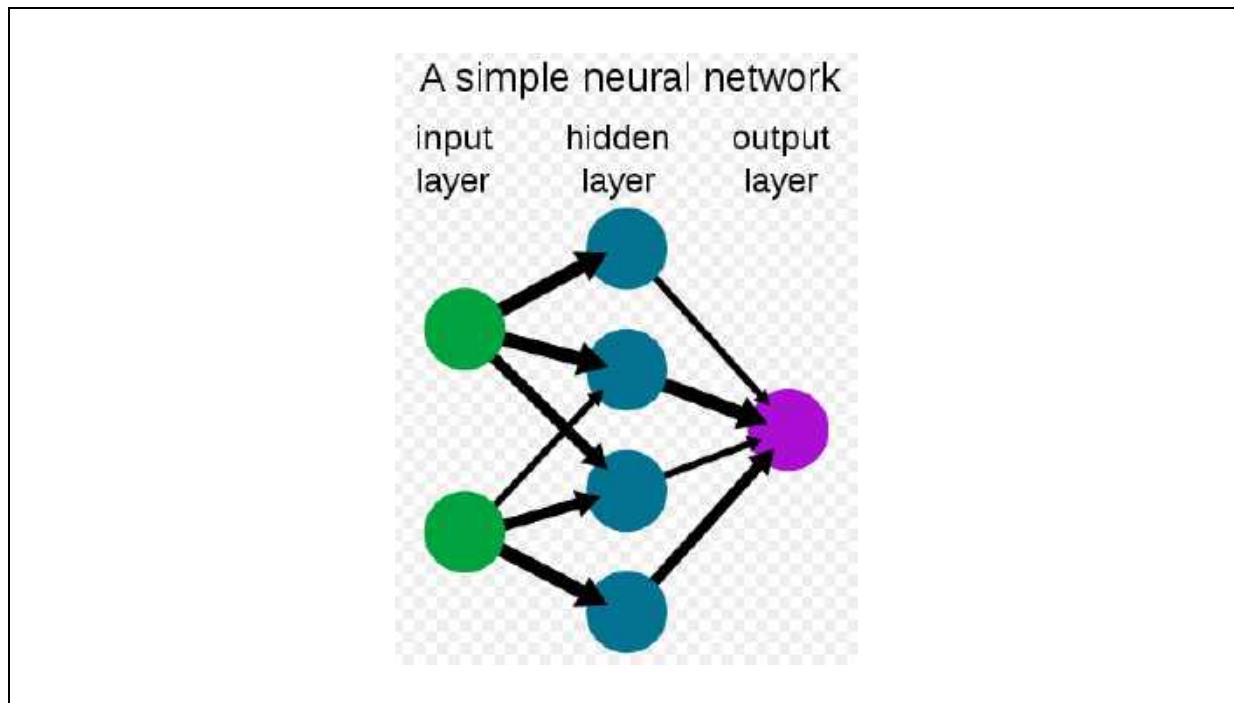
### 6. Performance

- ✓ Deep learning gives very good performance compare to machine learning



## 7. Neural networks

- ✓ Deep learning is implemented with the help of Neural Networks.
- ✓ A neural network is a network or circuit of neurons to solve the problem.
- ✓ The idea behind Neural Network is the biological neurons, which is nothing but a brain cell.
- ✓ Neural networks may have multiple hidden layers.



### 8. Types of neural networks

- ✓ There are mainly 3 types of neural networks in deep learning
  - Artificial Neural Networks (ANN) or Feed-Forward Neural network
  - Convolution Neural Networks (CNN)
  - Recurrent Neural Networks (RNN)

#### 8.1. Deep learning applications

- ✓ Self Driving Cars
- ✓ Fraud News Detection
- ✓ Natural Language Processing
- ✓ Virtual Assistants
- ✓ Entertainment
- ✓ Visual Recognition
- ✓ Fraud Detection
- ✓ Healthcare
- ✓ Language Translations & etc

## Data Science – Deep Learning - Libraries

---

### 2. Deep Learning – Libraries

#### Contents

1. Tensorflow.....	2
2. Pytorch.....	3
3. Keras .....	4
4. Steps to create deep learning models with Keras.....	5
4.1. Define your model. ....	5
4.2. Compile your model.....	5
4.3. Fit your model.....	5
4.4. Make predictions .....	5

## **2. Deep Learning – Libraries**

### **1. Tensorflow**

- ✓ Tensorflow is a deep learning framework
- ✓ This library was created by Google in the year of 2015 and it is open source.
- ✓ TensorFlow is the most famous deep learning library.
- ✓ It is entirely based on Python programming language and use for numerical computation and data flow, which makes machine learning faster and easier.
- ✓ TensorFlow is based on graph computation

#### **Logo**



#### **Tensorflow installation**

```
pip install tensorflow
```

#### **Update**

- ✓ Tensorflow 2<sup>nd</sup> version onwards keras library was in built
- ✓ So, we no need to install keras separately

## Data Science – Deep Learning - Libraries

---

### 2. Pytorch

- ✓ Pytorch is a deep learning framework
- ✓ Pytorch is a deep learning library created by Facebook in the year of 2016 and it is an open source

#### Logo



#### Note

- ✓ CNTK is another deep learning framework created by Microsoft but not that much popular compare to tensorflow and pytorch

### 3. Keras

- ✓ Keras is an open-source high-level Neural Network library, which was written in Python, is capable enough to run on Theano, TensorFlow, or CNTK.
- ✓ It was developed by one of the Google engineers, Francois Chollet.
- ✓ It is user-friendly, extensible, and faster experimentation with deep neural networks.
- ✓ It supports Convolutional Networks and Recurrent Networks individually also their combination.



## 4. Steps to create deep learning models with Keras

### 4.1. Define your model.

- ✓ Create a Sequential model and add configured layers.

### 4.2. Compile your model.

- ✓ Specify loss function and optimizers and call the compile() method on the model

### 4.3. Fit your model.

- ✓ Train the model on a sample of data by calling the fit () method on the model.

### 4.4. Make predictions

- ✓ Use the model to generate predictions on new data by calling the methods such as evaluate() or predict() on the model

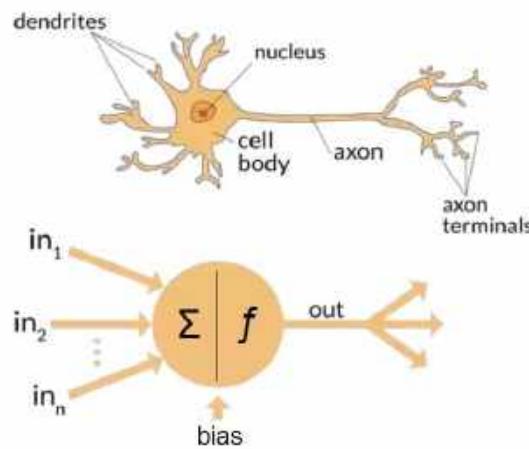
**3. Deep Learning – Terminology****Contents**

<b>1. Neuron .....</b>	<b>2</b>
<b>2. MLP (Multi-Layer Perceptrons).....</b>	<b>3</b>
<b>3. Neural network .....</b>	<b>4</b>
<b>4. Input, Hidden layers &amp; Output.....</b>	<b>5</b>
<b>5. Weights .....</b>	<b>6</b>
<b>6. Bias.....</b>	<b>7</b>
<b>7. Activation Function .....</b>	<b>8</b>
<b>8. Types of activation functions.....</b>	<b>9</b>
<b>9. Forward Propagation .....</b>	<b>10</b>
<b>10. Back propagation .....</b>	<b>11</b>
<b>11. Cost Function.....</b>	<b>12</b>
<b>12. Gradient Descent.....</b>	<b>13</b>
<b>13. Learning Rate.....</b>	<b>13</b>
<b>14. Batches .....</b>	<b>14</b>
<b>15. Epochs.....</b>	<b>14</b>

### 3. Deep Learning – Terminology

#### 1. Neuron

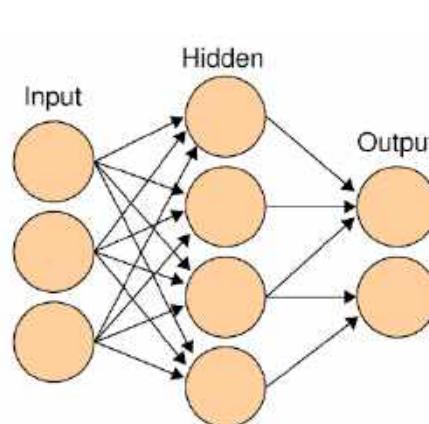
- ✓ Neuron forms the basic element of our brain.
- ✓ A group of neurons used to create neural network.
- ✓ A neuron receives an input, processes it and generates an output.



## Data Science – Deep Learning - Terminology

### 2. MLP (Multi-Layer Perceptrons)

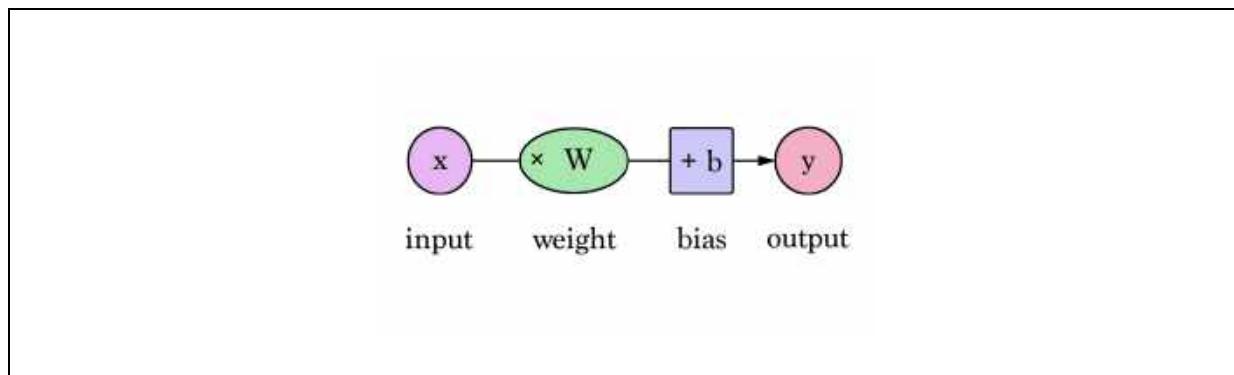
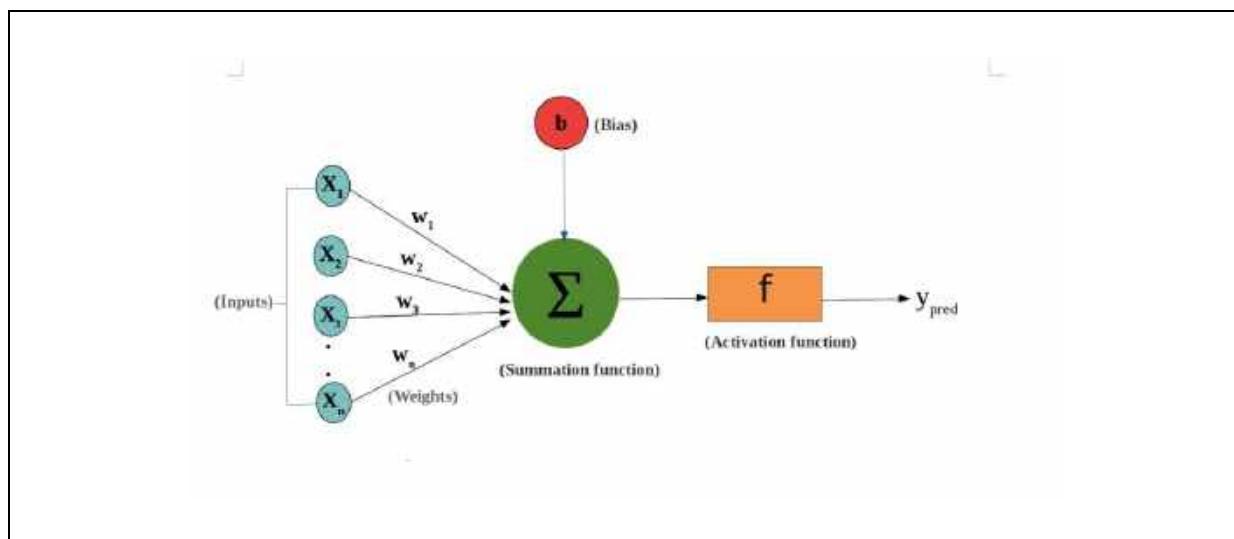
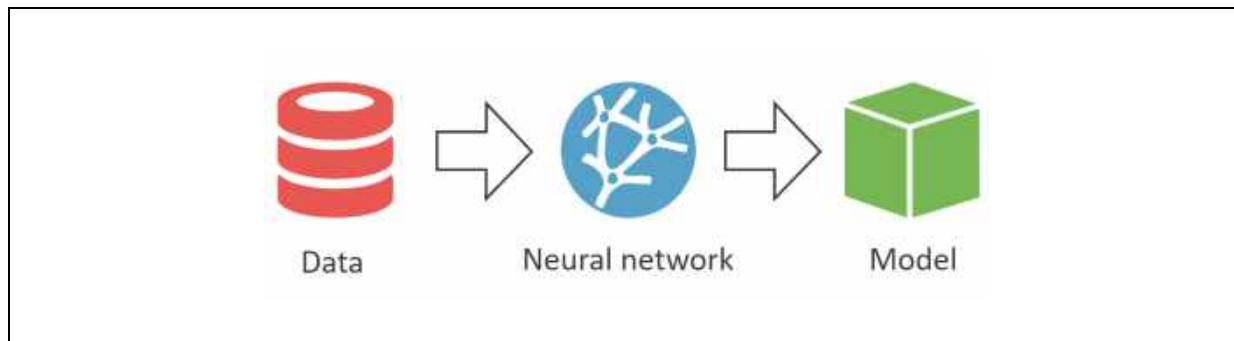
- ✓ A single neuron may not perform the complex tasks.
- ✓ So, it's required to use group of neurons to perform a complex task.
- ✓ In simple network we do have like,
  - Input layer.
  - Hidden layer.
  - Output layer.
- ✓ Each layer has multiple neurons.
- ✓ All neurons in each layer are connected to all the neurons in the next layer.



## Data Science – Deep Learning - Terminology

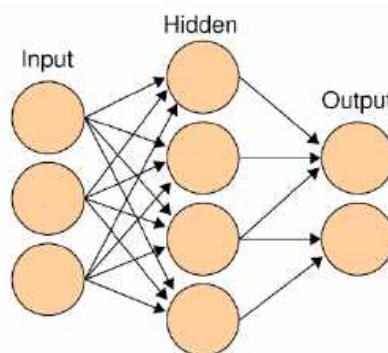
### 3. Neural network

- ✓ Neural Network is the backbone of deep learning.
- ✓ A Neural Network is combinations of basic Neurons also called as Perceptrons.
- ✓ The goal of a neural network is to find the mapping function.
  - Neurons having **weights** and **bias** which is updated during training.



#### 4. Input, Hidden layers & Output

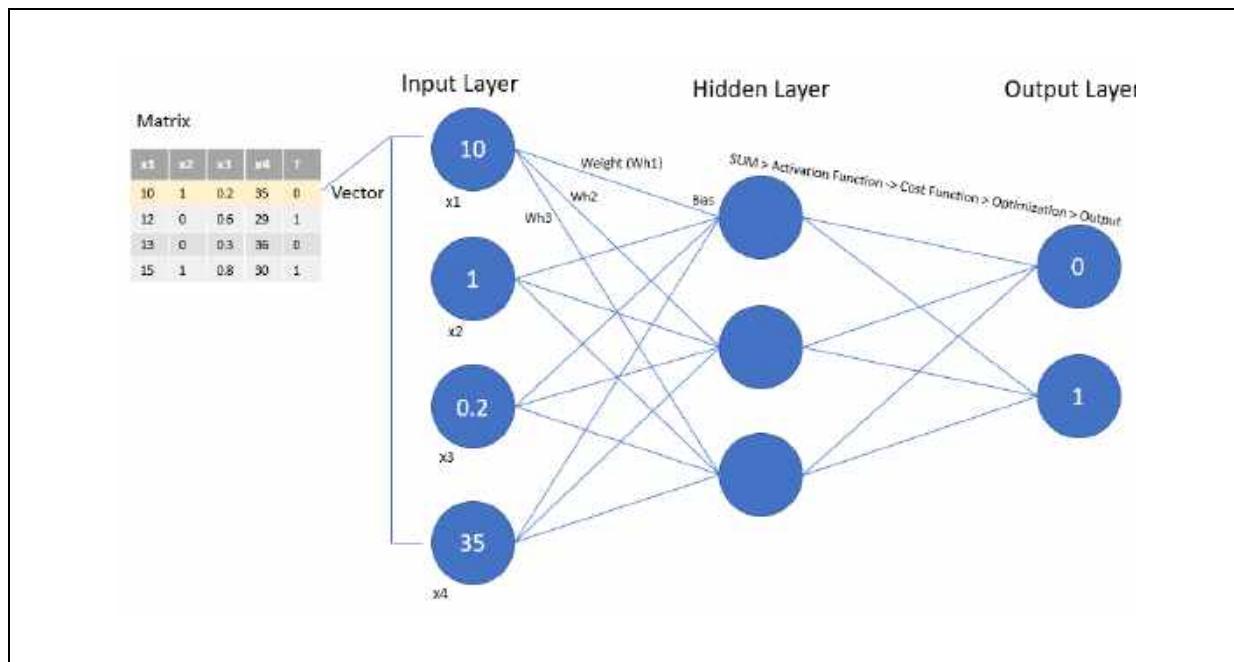
- ✓ Input layer receives the input
- ✓ The processing layers are the hidden layers within the network.
  - These layers perform specific tasks on the incoming data.
  - These layers can pass result to the next layers
- ✓ Output layer generates the output
- ✓ Input and output layers are visible but hidden layers are hidden



## Data Science – Deep Learning - Terminology

### 5. Weights

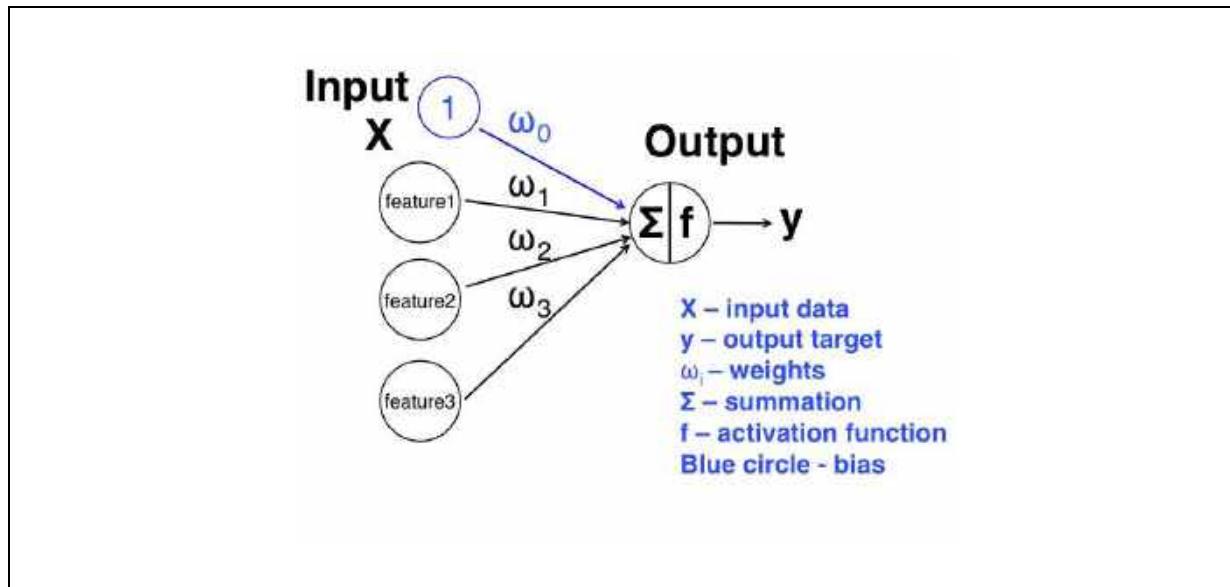
- ✓ When input enters into the neuron, it is multiplied by a weight.
- ✓ Assuming that, if a neuron has two inputs, then each input have separated weights.
- ✓ Here weights will be initialized randomly and these weights are updated during the model training.
  - Let's assume the input is **a** value and weight is **W1**.
  - Then after passing through the node the input becomes  **$a * W1$**



## Data Science – Deep Learning - Terminology

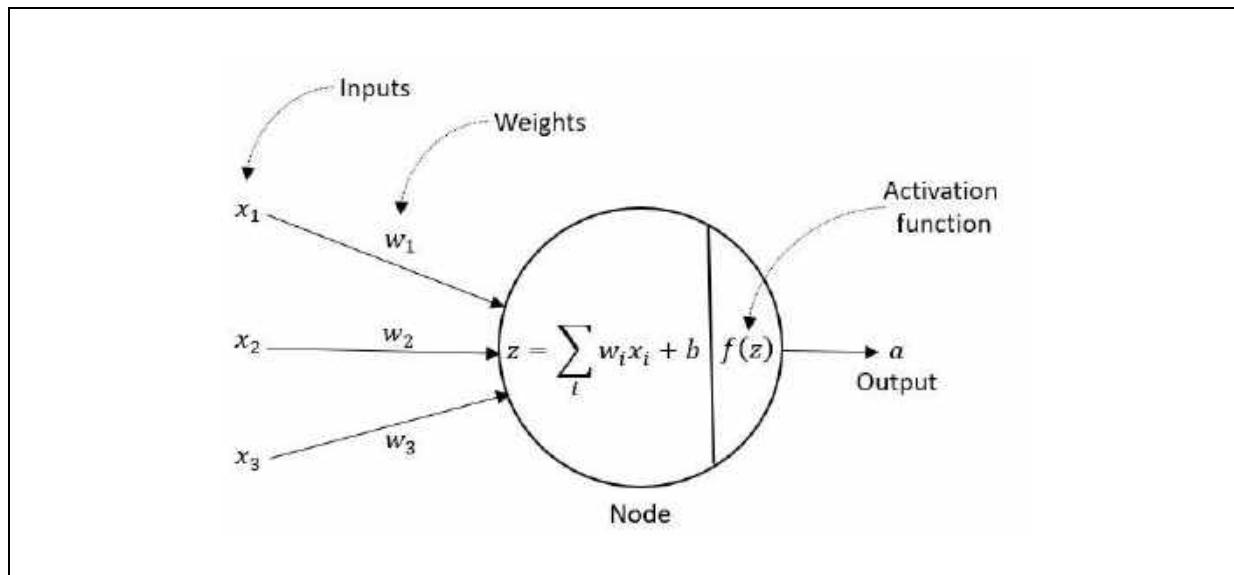
### 6. Bias

- ✓ In addition to the weights, bias also added to the input.
- ✓ After adding the bias, the result would look like,  $a * W1 + \text{bias}$ .



## 7. Activation Function

- ✓ The activation function translates the input signals to output signals.
- ✓ After applying activation function then its looks like,
  - $f(a^*W_1+b)$
  - Here  $f()$  is the activation function.
- ✓ The activation function puts a nonlinear transformation to the linear combination



### 8. Types of activation functions

- ✓ Sigmoid
- ✓ Linear
- ✓ Tanh or hyperbolic tangent
- ✓ ReLU(Rectified Linear Units)
- ✓ Softmax

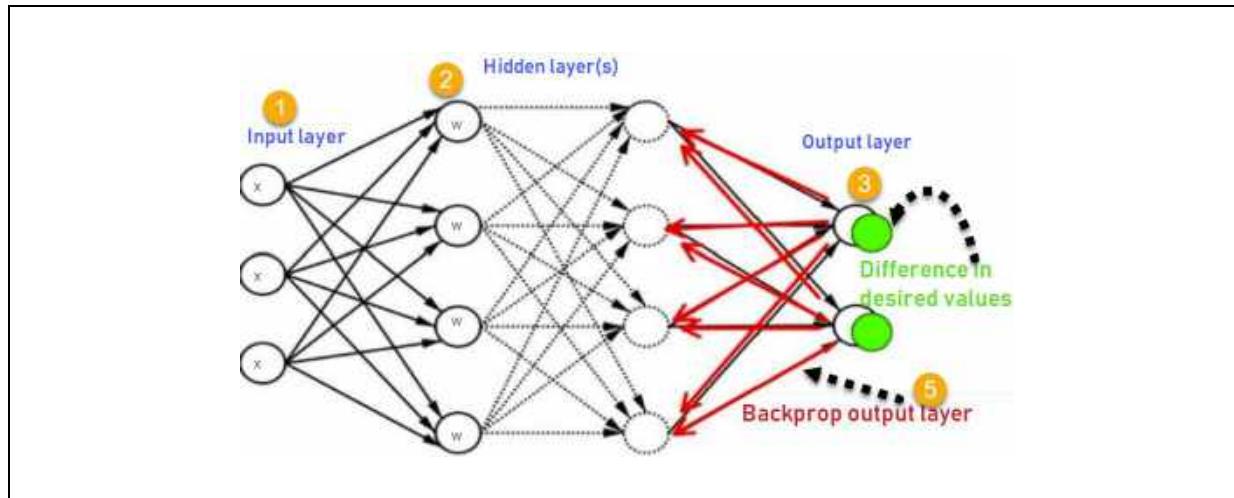
## 9. Forward Propagation

- ✓ In forward propagation, the information will be travelled into forward direction.
- ✓ The input layer provides input to the hidden layers and then the output is generated.
- ✓ In forward propagation input will not be travelled to backward direction.



## 10. Back propagation

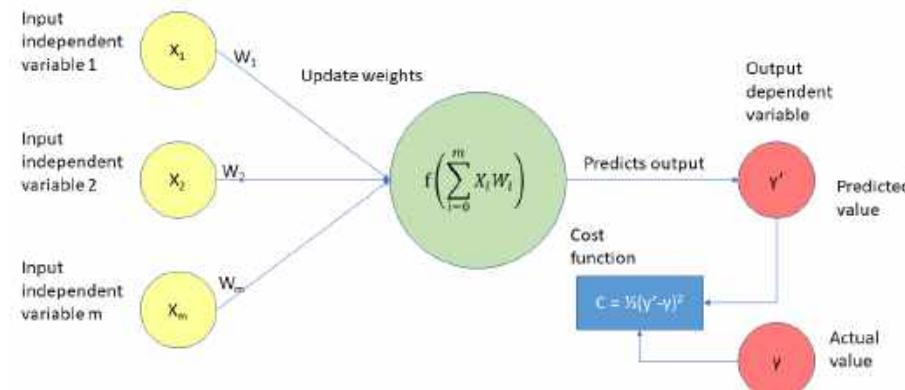
- ✓ During training, the network will get the results.
- ✓ These results will be compared with actual outputs by using loss/cost function.
- ✓ During comparing it will get error.
- ✓ To minimize this error internally weights supposed to be adjusted.
- ✓ So here back propagation helps to adjust the error.
- ✓ Back propagation means the,
  - The inputs results + error will travel in backward direction to adjust the weights



## Data Science – Deep Learning - Terminology

### 11. Cost Function

- ✓ When we create a network, the network tries to predict the output as close as possible to the actual value.
- ✓ We can measure this accuracy of the network by using the cost/loss function.
- ✓ The goal of running network is,
  - Increase our prediction accuracy
  - Reduce the error.
  - Minimizing the cost function.

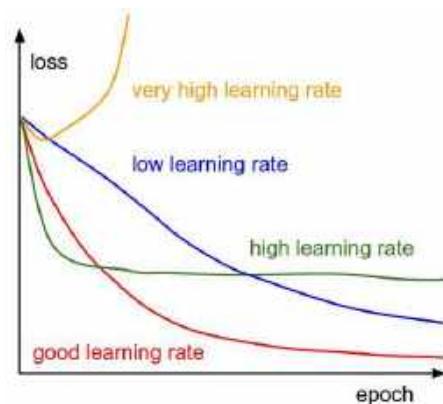


## 12. Gradient Descent

- ✓ Gradient descent is an optimization algorithm for minimizing the cost.

## 13. Learning Rate

- ✓ The learning rate is kind of hyper parameter to minimize the cost function in every iteration.
- ✓ We should choose the learning rate very carefully.
- ✓ If learning rate is large then it may miss minimum error point.
- ✓ If learning rate is very small then it takes long time to reach minimum error.
- ✓ So, optimize value is required.



## Data Science – Deep Learning - Terminology

---

### 14. Batches

- ✓ While training a neural network, instead of sending the entire input in one time, generally it divides into several chunks of equal size randomly.
- ✓ It would be really good practice to train the model with batch of data instead of entire data.

### 15. Epochs

- ✓ The training of the neural network with all the training data for one cycle.

**4. Deep Learning – Multilayer Perceptrons Models in Keras****Contents**

<b>1. Create model (Neural Network Model) by using Keras .....</b>	<b>3</b>
1.1. Model layers.....	4
1.2. Important properties to layers .....	5
1.2.1. Weight Initialization .....	5
1.2.2. Activation Function .....	6
<b>2. Model Compilation.....</b>	<b>7</b>
<b>3. Model Training .....</b>	<b>8</b>
<b>4. Model Prediction.....</b>	<b>9</b>

**4. Deep Learning – Multilayer Perceptrons Models in Keras****Steps to create MLP model in keras**

- ✓ Model creation (Neural Network Model) by using Keras
- ✓ Compile the model
- ✓ Model training
- ✓ Model prediction

**Behind the steps**

- ✓ Model creation
  - Model Layers
    - Weights initialization
    - Activation function
- ✓ Compile the model.
  - Optimization
  - loss function
  - metrics
- ✓ Model training
  - Epochs
  - Batch size
- ✓ Model predictions

**1. Create model (Neural Network Model) by using Keras**

- ✓ Sequential is a predefined class in keras package
- ✓ By using this we can create a model.

```
from tensorflow.keras.models import Sequential  
  
model = Sequential()
```

### 1.1. Model layers

- ✓ Once model created then we need to add layers to model.
  - Creating layers means, create object to Dense class
- ✓ We need to specify number of features to input layer.
  - Below example, 8 means its features

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()

layers_hidden_input = Dense(16, input_shape = (8,))

model.add(layers_hidden_input)
```

## 1.2. Important properties to layers

- ✓ Main properties,
  - Weight initialization.
  - Activation functions.

### 1.2.1. Weight Initialization

- ✓ By using `kernel_initializer`
  - `random_uniform`, `random_normal`, `zero`

#### Example

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()

layers_hidden_input = Dense(16, input_shape = (8, ),
                           kernel_initializer = "random_uniform")

model.add(layers_hidden_input)
```

## Data Science – DL – Multilayer Perceptrons Model in Keras

### 1.2.2. Activation Function

- ✓ We need to use activation functions like,
  - softmax
  - rectified linear (ReLU),
  - tanh,
  - sigmoid

#### Example

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()

layers_hidden_input = Dense(16, input_shape = (8, ),
kernel_initializer = "random_uniform", activation = "relu")

model.add(layers_hidden_input)
```

## 2. Model Compilation

- ✓ Once model created then we need to compile the model (Neural Network Model).
- ✓ During this step TensorFlow converts the model into a graph so the training can be carried out efficiently.
- ✓ We can compile by calling `compile()` method
- ✓ Important attributes in `compile()` method,
  - Model optimizer
  - Loss function
  - Metrics

### Example

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()

layers_hidden_input = Dense(16, input_shape = (8, ),
kernel_initializer = "random_uniform", activation = "relu")

model.add(layers_hidden_input)

model.compile(optimizer = ..., loss = ..., metrics = ...)
```

### 3. Model Training

- ✓ Once model created and compiled then next step is to train the model.
- ✓ We can train the model by using fit(...) method

#### Example

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()

layers_hidden_input = Dense(16, input_shape = (8, ),
kernel_initializer = "random_uniform", activation = "relu")

model.add(layers_hidden_input)

model.compile(optimizer = ..., loss = ..., metrics = ...)

model.fit(X, y, epochs = ..., batch_size =...)
```

#### 4. Model Prediction

- ✓ We model training is done then we can predict with new data.
  - `model.predict(X)`: To generate network output for the input data
  - `model.evaluate(X, y)`: To calculate the loss values for the input data

#### Example

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()

layers_hidden_input = Dense(16, input_shape = (8, ),
kernel_initializer = "random_uniform", activation = "relu")

model.add(layers_hidden_input)

model.compile(optimizer = ..., loss = ..., metrics = ...)

model.fit(X, y, epochs = ..., batch_size =...)

model.predict(X)
```

## 5. Deep Learning – First Neural Network with Keras

### Contents

<b>1. Implementing first neural network using keras.....</b>	<b>2</b>
<b>2. Dataset explanation .....</b>	<b>3</b>
<b>3. Input and output from the Dataset .....</b>	<b>4</b>
3.1. Input Variables (X):.....	4
3.2. Output Variables (y):.....	4
<b>4. Create the model.....</b>	<b>7</b>
4.1. Input shape and activation functions .....	8
<b>5. Compile the keras model .....</b>	<b>9</b>
5.1. Loss function, optimiser and metrics.....	10
<b>6. Fit the model .....</b>	<b>11</b>
<b>7. Evaluate the model .....</b>	<b>14</b>
<b>8. Prediction .....</b>	<b>17</b>
<b>9. verbose = 0 .....</b>	<b>20</b>
<b>10. Model summary .....</b>	<b>23</b>
<b>11. Image of the model .....</b>	<b>26</b>

**5. Deep Learning – First Neural Network with Keras****1. Implementing first neural network using keras**

- ✓ Importing the libraries
- ✓ Loading Dataset
- ✓ Data preparation
- ✓ Splitting the dataset
- ✓ Model creation
- ✓ Model compilation
- ✓ Model training
- ✓ Prediction

## 2. Dataset explanation

- ✓ We are going to work with **pima-indians-diabetes.csv**
- ✓ This Dataset is related to health care domain.
- ✓ Pima Indians are a Native American group that lives in Mexico and Arizona, USA.
- ✓ It describes patient medical record data for Pima Indians and whether they had a diabetes within five years.
- ✓ The Pima Indian Diabetes dataset consisting of Pima Indian females 21 years and older is a popular benchmark dataset.
- ✓ It is a binary classification problem (onset of diabetes as 1 or not as 0).
- ✓ All of the input variables that describe each patient are numerical.

Feature	Description	Data type	Range
Preg	Number of times pregnant	Numeric	[0, 17]
Gluc	Plasma glucose concentration at 2 Hours in an oral glucose tolerance test (GTIT)	Numeric	[0, 199]
BP	Diastolic Blood Pressure (mm Hg)	Numeric	[0, 122]
Skin	Triceps skin fold thickness (mm)	Numeric	[0, 99]
Insulin	2-Hour Serum insulin ( $\mu$ h/ml)	Numeric	[0, 846]
BMI	Body mass index [weight in kg/(Height in m)]	Numeric	[0, 67.1]
DPF	Diabetes pedigree function	Numeric	[0.078, 2.42]
Age	Age (years)	Numeric	[21, 81]
Outcome	Binary value indicating non-diabetic /diabetic	Factor	[0,1]

### 3. Input and output from the Dataset

#### 3.1. Input Variables (X):

- ✓ 1. Number of times pregnant
- ✓ 2. Plasma glucose concentration at 2 hours in an oral glucose tolerance test
- ✓ 3. Diastolic blood pressure (mm Hg)
- ✓ 4. Triceps skin fold thickness (mm)
- ✓ 5. 2-hour serum insulin ( $\mu$ IU/ml)
- ✓ 6. Body mass index (weight in kg/(height in m))
- ✓ 7. Diabetes pedigree function
- ✓ 8. Age (years)

#### 3.2. Output Variables (y):

- ✓ 1. Class variable (0 or 1)

## Data Science – DL – First Neural Network with Keras

**Program Name** Loading csv file  
**demo1.py**  
**Input file** pima-indians-diabetes.csv

```
from numpy import loadtxt

dataset = loadtxt('pima-indians-diabetes.csv', delimiter = ',')

print(dataset)
```

### Output

```
[[ 6.    148.    72.    ...   0.627   50.    1.    ]
 [ 1.    85.    66.    ...   0.351   31.    0.    ]
 [ 8.    183.    64.    ...   0.672   32.    1.    ]
 ...
 [ 5.    121.    72.    ...   0.245   30.    0.    ]
 [ 1.    126.    60.    ...   0.349   47.    1.    ]
 [ 1.    93.    70.    ...   0.315   23.    0.    ]]
```

## Data Science – DL – First Neural Network with Keras

**Program** Split into input (X) and output (y) variables

**Name** demo2.py

**Input file** pima-indians-diabetes.csv

```
from numpy import loadtxt

dataset = loadtxt('pima-indians-diabetes.csv', delimiter = ',')

X = dataset[:, 0:8]
y = dataset[:, 8]

print("Splitting data done")
```

**Output**

```
Splitting data done
```

## Data Science – DL – First Neural Network with Keras

### 4. Create the model

- ✓ First step is, we need to create the model by using Sequential class
- ✓ Once model created then we need to add layers to the model

**Program** Model creation

**Name** demo4.py

**Input file** pima-indians-diabetes.csv

```
from numpy import loadtxt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

dataset = loadtxt('pima-indians-diabetes.csv', delimiter = ',')

X = dataset[:, 0:8]
y = dataset[:, 8]

model = Sequential()

layer1 = Dense(12, input_shape = (8, ), activation = 'relu')
layer2 = Dense(8, activation = 'relu')
layer3 = Dense(1, activation = 'sigmoid')

model.add(layer1)
model.add(layer2)
model.add(layer3)

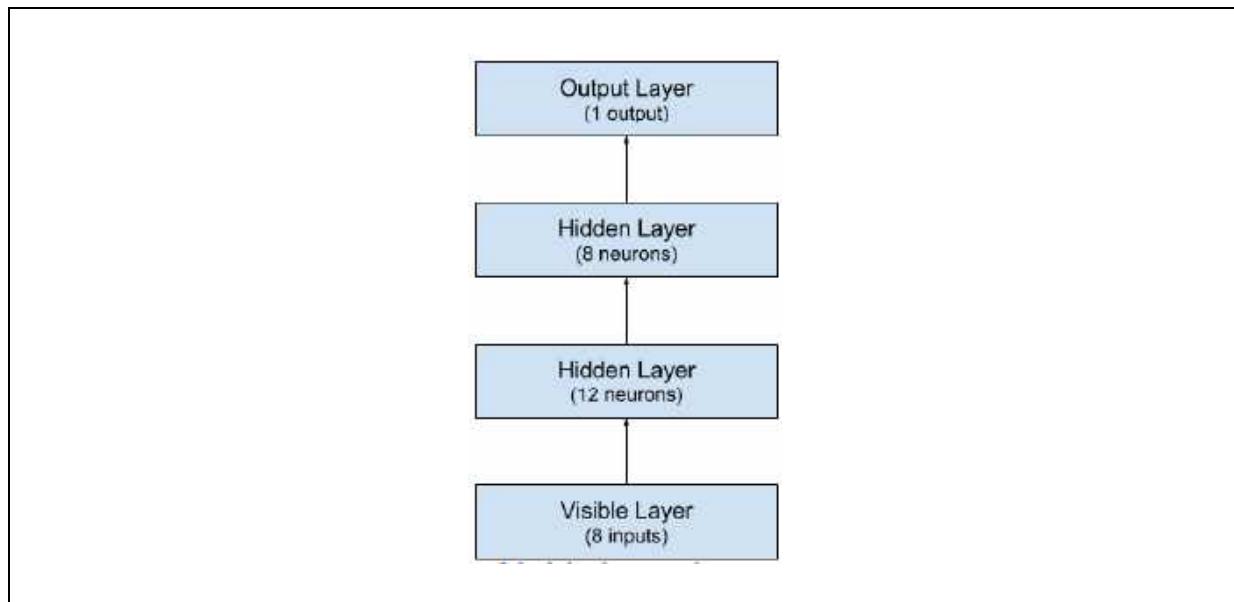
print("Model created")
```

**Output**

Model created

#### 4.1. Input shape and activation functions

- ✓ Fully connected layers are defined using the `Dense` class.
  - ✓ The model expects rows of data with **8** features (the `input_shape = (8,)` argument).
  - ✓ The first hidden layer has **12** nodes and uses the `relu` activation function.
  - ✓ The second hidden layer has **8** nodes and uses the `relu` activation function.
  - ✓ The output layer has **1** node and uses the sigmoid activation function.
- ✓ **The line of code**, `Dense` layer is doing two things, creating first hidden and input layers.



## Data Science – DL – First Neural Network with Keras

---

### 5. Compile the keras model

- ✓ Once the model created then we need to compile the model

<b>Program Name</b> <b>Input file</b>	Compile the keras model demo5.py pima-indians-diabetes.csv
--	--

```

# importing required libraries
from numpy import loadtxt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# load the dataset
dataset = loadtxt('pima-indians-diabetes.csv', delimiter = ',')

# split into input (X) and output (y) variables
X = dataset[:, 0:8]
y = dataset[:, 8]

# define the keras model
model = Sequential()

layer1 = Dense(12, input_shape = (8, ), activation = 'relu')
layer2 = Dense(8, activation = 'relu')
layer3 = Dense(1, activation = 'sigmoid')

model.add(layer1)
model.add(layer2)
model.add(layer3)

model.compile(loss = 'binary_crossentropy', optimizer = 'adam',
metrics = ['accuracy'])

print("Model compiled")

```

**Output**

Model created

### 5.1. Loss function, optimiser and metrics

- ✓ We need to provide loss function to evaluate a set of weights.
- ✓ The optimizer is used to search through different weights for the network.
- ✓ This loss is for a binary classification problems and is defined in Keras as “binary\_crossentropy”.
- ✓ We have given optimizer value as adam.
  - This is more efficient gradient descent algorithm.

## 6. Fit the model

- ✓ Once model compiled then we need to train the model
- ✓ By using fit(...) method we can train the model.

Program

Fit the model

Name

demo6.py

Input file

pima-indians-diabetes.csv

```
print("Topic: First DL Example")
print()
```

```
print("Step 1: Importing libraries")
```

```
from numpy import loadtxt
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
```

```
import warnings
warnings.filterwarnings("ignore")
```

```
print("Step 2: Loading the dataset")
```

```
dataset = loadtxt(
    'pima-indians-diabetes.csv',
    delimiter=','
)
```

```
print("Step 3: Data preparation")
```

```
X = dataset[:, 0:8]
y = dataset[:, 8]
```

## Data Science – DL – First Neural Network with Keras

```
print("Step 4: Splitting the dataset: Optional")

print("Step 5: Model(Neural Network) creation")

model = Sequential()

print("Step 5.1: Creating layers and add to the model")

layers12 = Dense(12, input_shape = (8, ), activation = 'relu')
layer3 = Dense(8, activation = 'relu')
layer4 = Dense(1, activation = 'sigmoid')

model.add(layers12)
model.add(layer3)
model.add(layer4)

print("Step 6: Model compilation")

model.compile(
    loss = 'binary_crossentropy',
    optimizer = 'adam',
    metrics = ['accuracy']
)

print("Step 7: Model training")

model.fit(
    X,
    y,
    epochs = 150,
    batch_size = 10,
)
```

## Data Science – DL – First Neural Network with Keras

### Output

```
[1m77/77-[0m-[32m-[0m-[37m-[0m-[1m0s-[0m 2ms/step - accuracy: 0.7211 - loss: 0.5256
Epoch 147/150
-[1m77/77-[0m-[32m-[0m-[37m-[0m-[1m0s-[0m 3ms/step - accuracy: 0.7631 - loss: 0.5104
Epoch 148/150
-[1m77/77-[0m-[32m-[0m-[37m-[0m-[1m0s-[0m 2ms/step - accuracy: 0.7820 - loss: 0.4735
Epoch 149/150
-[1m77/77-[0m-[32m-[0m-[37m-[0m-[1m0s-[0m 3ms/step - accuracy: 0.7645 - loss: 0.4969
Epoch 150/150
-[1m77/77-[0m-[32m-[0m-[37m-[0m-[1m0s-[0m 2ms/step - accuracy: 0.7715 - loss: 0.4861
-[1m24/24-[0m-[32m-[0m-[37m-[0m-[1m0s-[0m 2ms/step - accuracy: 0.7010 - loss: 0.5844
```

## 7. Evaluate the model

- ✓ Once training is done then we need to evaluate the performance of the network.
- ✓ By using evaluate() method we can evaluate.
- ✓ This method return two values,
  - The first will be the loss of the model on the dataset.
  - The second will be the accuracy of the model on the dataset.

**Program Name**

Fit the model

**Name**

demo7.py

**Input file**

pima-indians-diabetes.csv

```

print("Topic: First DL Example")
print()

print("Step 1: Importing libraries")

from numpy import loadtxt
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential

import warnings
warnings.filterwarnings("ignore")

print("Step 2: Loading the dataset")

dataset = loadtxt(
    'pima-indians-diabetes.csv',
    delimiter=','
)

print("Step 3: Data preparation")

X = dataset[:, 0:8]
y = dataset[:, 8]

```

## Data Science – DL – First Neural Network with Keras

```
print("Step 4: Splitting the dataset: Optional")

print("Step 5: Model(Neural Network) creation")

model = Sequential()

print("Step 5.1: Creating layers and add to the model")

layers12 = Dense(12, input_shape = (8, ), activation = 'relu')
layer3 = Dense(8, activation = 'relu')
layer4 = Dense(1, activation = 'sigmoid')

model.add(layers12)
model.add(layer3)
model.add(layer4)

print("Step 6: Model compilation")

model.compile(
    loss = 'binary_crossentropy',
    optimizer = 'adam',
    metrics = ['accuracy']
)

print("Step 7: Model training")

model.fit(
    X,
    y,
    epochs = 150,
    batch_size = 10,
)

# Evaluate the keras model

_, accuracy = model.evaluate(X, y)
print("Accuracy is:", accuracy*100)
```

## Data Science – DL – First Neural Network with Keras

### Output

```
Epoch 147/158
-[1m77/77-[0m +[32m-[0m-[37m-[0m +[1m0s-[0m 1ms/step - accuracy: 0.7728 - loss: 0.4662
Epoch 148/158
-[1m77/77-[0m +[32m-[0m-[37m-[0m +[1m0s-[0m 1ms/step - accuracy: 0.8018 - loss: 0.4593
Epoch 149/158
-[1m77/77-[0m +[32m-[0m-[37m-[0m +[1m0s-[0m 1ms/step - accuracy: 0.7617 - loss: 0.4846
Epoch 150/158
-[1m77/77-[0m +[32m-[0m-[37m-[0m +[1m0s-[0m 1ms/step - accuracy: 0.7996 - loss: 0.4454
-[1m24/24-[0m +[32m-[0m-[37m-[0m +[1m0s-[0m 679us/step - accuracy: 0.7554 - loss: 0.4755
Accuracy is: 78.25520634651184
```

## 8. Prediction

- ✓ By using predict(...) method we can do the prediction.
- ✓ We are using sigmoid activation function on the output layer.
  - The predictions will be a probability in the range between 0 and 1

Program

Model prediction

Name

demo8.py

Input file

pima-indians-diabetes.csv

```
print("Topic: First DL Example")
print()

print("Step 1: Importing libraries")

import numpy as np
from numpy import loadtxt
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential

import warnings
warnings.filterwarnings("ignore")

print("Step 2: Loading the dataset")

dataset = loadtxt(
    'pima-indians-diabetes.csv',
    delimiter = ','
)

print("Step 3: Data preparation")

X = dataset[:, 0:8]
y = dataset[:, 8]
```

## Data Science – DL – First Neural Network with Keras

```
print("Step 4: Splitting the dataset: Optional")

print("Step 5: Model(Neural Network) creation")

model = Sequential()

print("Step 5.1: Creating layers and add to the model")

layers12 = Dense(12, input_shape = (8, ), activation = 'relu')
layer3 = Dense(8, activation = 'relu')
layer4 = Dense(1, activation = 'sigmoid')

model.add(layers12)
model.add(layer3)
model.add(layer4)

print("Step 6: Model compilation")

model.compile(
    loss = 'binary_crossentropy',
    optimizer = 'adam',
    metrics = ['accuracy']
)

print("Step 7: Model training")

model.fit(
    X,
    y,
    epochs = 150,
    batch_size = 10,
)
```

## Data Science – DL – First Neural Network with Keras

```

print("Step 8: Prediction")

ip = [[6, 148, 72, 35, 0, 33.6, 0.627, 50]]

input_data = np.array(ip)
result = model.predict(input_data)

print()
print(result)

```

### Output

```

Epoch 148/150
-[1m77/77+[0m +[32m-----+ [0m+[37m+[0m +[1m0s+[0m 1ms/step - accuracy: 0.7216 - loss: 0.5420
Epoch 149/150
-[1m77/77+[0m +[32m-----+ [0m+[37m+[0m +[1m0s+[0m 2ms/step - accuracy: 0.7353 - loss: 0.5331
Epoch 150/150
-[1m77/77+[0m +[32m-----+ [0m+[37m+[0m +[1m0s+[0m 1ms/step - accuracy: 0.6918 - loss: 0.5576
Step 8: Prediction
-[1m1/1+[0m +[32m-----+ [0m+[37m+[0m +[1m0s+[0m 28ms/step
[[0.7929609]]

```

### Note

- ✓ Here model done prediction for first five rows compared to the expected class value.
- ✓ You can see that most rows are correctly predicted.
- ✓ We got 79.2% accuracy with good model performance.

## Data Science – DL – First Neural Network with Keras

### 9. verbose = 0

- ✓ If we provide verbose = 0 then progress bar will be not displayed.

Program	Model prediction
Name	demo9.py
Input file	pima-indians-diabetes.csv

```
print("Topic: First DL Example")
print()

print("Step 1: Importing libraries")

from numpy import loadtxt
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential

import warnings
warnings.filterwarnings("ignore")

print("Step 2: Loading the dataset")

dataset = loadtxt(
    'pima-indians-diabetes.csv',
    delimiter=','
)

print("Step 3: Data preparation")

X = dataset[:, 0:8]
y = dataset[:, 8]

print("Step 4: Splitting the dataset: Optional")
```

## Data Science – DL – First Neural Network with Keras

```
print("Step 5: Model(Neural Network) creation")

model = Sequential()

print("Step 5.1: Creating layers and add to the model")

layers12 = Dense(12, input_shape = (8, ), activation = 'relu')
layer3 = Dense(8, activation = 'relu')
layer4 = Dense(1, activation = 'sigmoid')

model.add(layers12)
model.add(layer3)
model.add(layer4)

print("Step 6: Model compilation")

model.compile(
    loss = 'binary_crossentropy',
    optimizer = 'adam',
    metrics = ['accuracy']
)

print("Step 7: Model training")

model.fit(
    X,
    y,
    epochs = 150,
    batch_size = 10,
    verbose = 0
)

# Evaluate the keras model
_, accuracy = model.evaluate(X, y)
print("Accuracy is:", accuracy*100)
```

## Output

## 10. Model summary

- ✓ By using summary method, we can check the summary of the model.

Program Name	Model summary
Input file	demo10.py
	pima-indians-diabetes.csv

```
print("Topic: First DL Example")
print()

print("Step 1: Importing libraries")

from numpy import loadtxt
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential

import warnings
warnings.filterwarnings("ignore")

print("Step 2: Loading the dataset")

dataset = loadtxt(
    'pima-indians-diabetes.csv',
    delimiter = ','
)

print("Step 3: Data preparation")

X = dataset[:, 0:8]
y = dataset[:, 8]

print("Step 4: Splitting the dataset: Optional")
```

## Data Science – DL – First Neural Network with Keras

```
print("Step 5: Model(Neural Network) creation")

model = Sequential()

print("Step 5.1: Creating layers and add to the model")

layers12 = Dense(12, input_shape = (8, ), activation = 'relu')
layer3 = Dense(8, activation = 'relu')
layer4 = Dense(1, activation = 'sigmoid')

model.add(layers12)
model.add(layer3)
model.add(layer4)

print("Step 6: Model compilation")

model.compile(
    loss = 'binary_crossentropy',
    optimizer = 'adam',
    metrics = ['accuracy']
)

print("Step 7: Model training")

model.fit(
    X,
    y,
    epochs = 150,
    batch_size = 10,
    verbose = 0
)

# Model Summary

model.summary()
```

## Data Science – DL – First Neural Network with Keras

### Output

```
Step 6: Model compilation
Step 7: Model training
Model: "sequential"

Layer (type)          Output Shape        Param #
dense (Dense)         (None, 12)           108
dense_1 (Dense)       (None, 8)            104
dense_2 (Dense)       (None, 1)             9

Total params: 685 (2.60 KB)
Trainable params: 221 (884.00 B)
Non-trainable params: 0 (0.00 B)
Optimizer params: 444 (1.74 KB)
```

## 11. Image of the model

- ✓ We can get the image of the model

Program	Image of the model
Name	demo11.py
Input file	pima-indians-diabetes.csv

```
# importing required libraries
from numpy import loadtxt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# load the dataset
dataset = loadtxt('pima-indians-diabetes.csv', delimiter = ',')

# split into input (X) and output (y) variables
X = dataset[:, 0:8]
y = dataset[:, 8]

# define the keras model

model = Sequential()

layer1 = Dense(12, input_shape = (8, ), activation = 'relu')
layer2 = Dense(8, activation = 'relu')
layer3 = Dense(1, activation = 'sigmoid')

model.add(layer1)
model.add(layer2)
model.add(layer3)

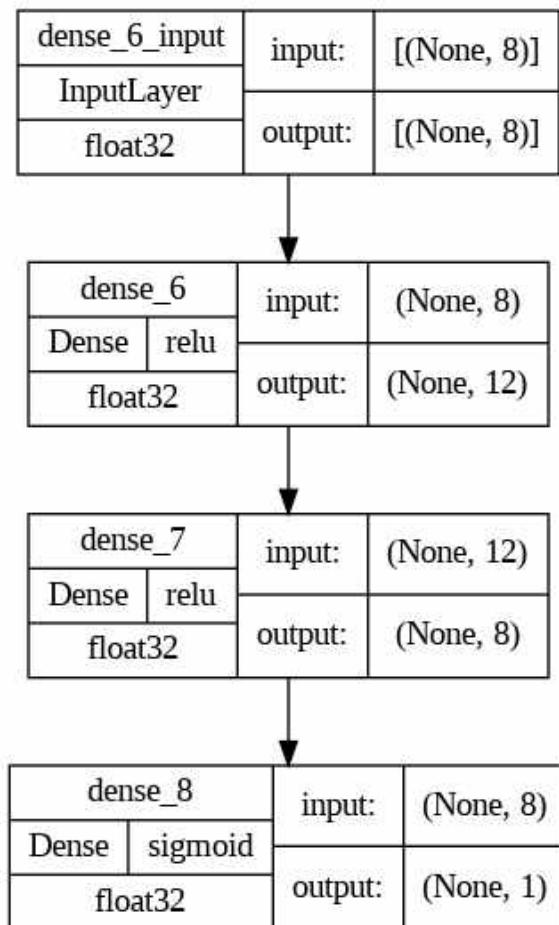
# compile the keras model
model.compile(loss = 'binary_crossentropy', optimizer = 'adam',
metrics = ['accuracy'])

# fit the keras model on the dataset
model.fit(X, y, epochs = 150, batch_size = 10, verbose = 0)
```

## Data Science – DL – First Neural Network with Keras

```
from tensorflow.keras.utils import plot_model  
  
plot_model(model, to_file = 'model.png', show_shapes = True,  
show_dtype = True, show_layer_names = True, expand_nested =  
True, show_layer_activations = True)
```

### Output



## 6. Deep Learning – Evaluate Model Performance

### Contents

1. Model evaluation .....	2
2. Data splitting - Automatic Verification Dataset .....	3
3. Data splitting - Use a Manual Verification Dataset.....	7
4. Manual k-Fold Cross-Validation.....	11

**6. Deep Learning – Evaluate Model Performance****1. Model evaluation**

- ✓ We can evaluate the model by using below ways,
  - Data Splitting
    - Use an automatic verification dataset
    - Use a manual verification dataset
  - Manual k-Fold Cross-Validation

## 2. Data splitting - Automatic Verification Dataset

- ✓ Once model compiling is done then we need to train the model.
- ✓ validation\_split keyword argument helps in automation verification dataset.
- ✓ We can train the model by using fit() method
  - For fit(validation\_split = 0.33) method we can provide validation\_split = 0.33 as keyword argument.
- ✓ We can give 0.2 or 0.33 for 20% or 33% of your training data held back for validation.

## Data Science – DL – Evaluate Model Performance

**Program Name** Model evaluation training  
**Input file** demo6.py  
pima-indians-diabetes.csv

```
print("Topic: First DL Example")
print()

print("Step 1: Importing libraries")

from numpy import loadtxt
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential

import warnings
warnings.filterwarnings("ignore")

print("Step 2: Loading the dataset")

dataset = loadtxt(
    'pima-indians-diabetes.csv',
    delimiter = ','
)

print("Step 3: Data preparation")

X = dataset[:, 0:8]
y = dataset[:, 8]

print("Step 4: Splitting the dataset: Optional")
```

## Data Science – DL – Evaluate Model Performance

```
print("Step 5: Model(Neural Network) creation")

model = Sequential()

print("Step 5.1: Creating layers and add to the model")

layers12 = Dense(12, input_shape = (8, ), activation = 'relu')
layer3 = Dense(8, activation = 'relu')
layer4 = Dense(1, activation = 'sigmoid')

model.add(layers12)
model.add(layer3)
model.add(layer4)

print("Step 6: Model compilation")

model.compile(
    loss = 'binary_crossentropy',
    optimizer = 'adam',
    metrics = ['accuracy']
)

print("Step 7: Model training")

model.fit(
    X,
    y,
    validation_split = 0.33,
    epochs = 150,
    batch_size = 10,
)
```

## Data Science – DL – Evaluate Model Performance

### Output

```
Epoch 147/150
[1m52s/52s | 0m + [32m] [0m + [1m0s+[0m 2ms/step - accuracy: 0.7713 - loss: 0.4595 - val_accuracy:
0.7841 - val_loss: 0.5430
Epoch 148/150
[1m52s/52s | 0m + [32m] [0m + [1m0s+[0m 2ms/step - accuracy: 0.7874 - loss: 0.4698 - val_accuracy:
0.7285 - val_loss: 0.5800
Epoch 149/150
[1m52s/52s | 0m + [32m] [0m + [1m0s+[0m 2ms/step - accuracy: 0.7521 - loss: 0.4930 - val_accuracy:
0.7528 - val_loss: 0.5419
Epoch 150/150
[1m52s/52s | 0m + [32m] [0m + [1m0s+[0m 2ms/step - accuracy: 0.7752 - loss: 0.4899 - val_accuracy:
0.7088 - val_loss: 0.5881
```

### Note

- ✓ Running the example, you can see that the verbose output on each epoch shows the loss and accuracy on both the training dataset and the validation dataset.

### 3. Data splitting - Use a Manual Verification Dataset

- ✓ train\_test\_split() function helps in manual verification of the dataset.
- ✓ Let me reminder we already worked with train\_test\_split() function in machine learning.

## Data Science – DL – Evaluate Model Performance

**Program Name** Model evaluation training  
**demo6.py**  
**Input file** pima-indians-diabetes.csv

```
print("Topic: First DL Example")
print()

print("Step 1: Importing libraries")

from numpy import loadtxt
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
from sklearn.model_selection import train_test_split

import warnings
warnings.filterwarnings("ignore")

print("Step 2: Loading the dataset")

dataset = loadtxt(
    'pima-indians-diabetes.csv',
    delimiter = ','
)

print("Step 3: Data preparation")

X = dataset[:, 0:8]
y = dataset[:, 8]
```

## Data Science – DL – Evaluate Model Performance

```
print("Step 4: Splitting the dataset: Optional")

X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size = 0.33,
    random_state = 7
)

print("Step 5: Model(Neural Network) creation")

model = Sequential()

print("Step 5.1: Creating layers and add to the model")

layers12 = Dense(12, input_shape = (8, ), activation = 'relu')
layer3 = Dense(8, activation = 'relu')
layer4 = Dense(1, activation = 'sigmoid')

model.add(layers12)
model.add(layer3)
model.add(layer4)

print("Step 6: Model compilation")

model.compile(
    loss = 'binary_crossentropy',
    optimizer = 'adam',
    metrics = ['accuracy']
)
```

## Data Science – DL – Evaluate Model Performance

```
print("Step 7: Model training")  
  
model.fit(  
    X_train,  
    y_train,  
    validation_data = (X_test, y_test),  
    epochs = 150,  
    batch_size = 10  
)
```

### Output

```
Epoch 148/150  
+1m52s/52s [0m +[32n—————] 0m+37m-[0m+[1m0s+[0m 2ms/step - accuracy: 0.7712 - loss: 0.4884 - val_accuracy:  
0.7283 - val_loss: 0.5412  
Epoch 149/150  
+1m52s/52s [0m +[32n—————] 0m+37m-[0m+[1m0s+[0m 2ms/step - accuracy: 0.7858 - loss: 0.4621 - val_accuracy:  
0.7638 - val_loss: 0.5658  
Epoch 150/150  
+1m52s/52s [0m +[32n—————] 0m+37m-[0m+[1m0s+[0m 2ms/step - accuracy: 0.7714 - loss: 0.4710 - val_accuracy:  
0.7362 - val_loss: 0.5260
```

#### 4. Manual k-Fold Cross-Validation

- ✓ By using k fold cross validation we can evaluate the model.
- ✓ Let me remind, we already worked with k fold cross validation in machine learning.
- ✓ It provides very good performance on the model.
- ✓ It splits the dataset into k subsets and used to apply training and validate on subsets.
- ✓ Finally it used to get average score from all models.

## Data Science – DL – Evaluate Model Performance

**Program Name** K fold cross validation  
**Input file** demo3.py  
pima-indians-diabetes.csv

```
# importing required libraries
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import StratifiedKFold
import numpy as np

import warnings
warnings.filterwarnings("ignore")

# load the dataset
dataset = np.loadtxt('pima-indians-diabetes.csv', delimiter = ',')

# split into input (X) and output (y) variables
X = dataset[:, 0:8]
y = dataset[:, 8]

# define 10-fold cross validation test harness
kfold = StratifiedKFold(n_splits = 10, shuffle = True, random_state = 7)

cvscores = []

for train, test in kfold.split(X, y):
    # create model
    model = Sequential()

    model.add(Dense(12, input_shape=(8,), activation='relu'))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))

    # Compile model
    model.compile(loss='binary_crossentropy',
```

## Data Science – DL – Evaluate Model Performance

```

optimizer='adam', metrics = ['accuracy'])

# Fit the model
model.fit(X[train], y[train], epochs=150, batch_size=10,
verbose=0)

# evaluate the model
scores = model.evaluate(X[test], y[test])

print("Accuracy:", scores[1]*100)

cvscores.append(scores[1] * 100)

print("Mean Accuracy", np.mean(cvscores))

```

### Output

```

('Accuracy:', 59.740257263183594)
+ [1m3/3+[0m +[32m
+ [0m-[37ms-[0m +[1m0s+[0m 0s/step - accuracy: 0.7113 - loss: 0.5755
('Accuracy:', 68.83116960525513)
+ [1m3/3+[0m +[32m
+ [0m-[37ms-[0m +[1m0s+[0m 0s/step - accuracy: 0.7152 - loss: 0.5587
('Accuracy:', 72.72727489471436)
+ [1m3/3+[0m +[32m
+ [0m-[37ms-[0m +[1m0s+[0m 0s/step - accuracy: 0.7212 - loss: 0.6130
('Accuracy:', 72.36841917937964)
+ [1m3/3+[0m +[32m
+ [0m-[37ms-[0m +[1m0s+[0m 0s/step - accuracy: 0.7644 - loss: 0.4472
('Accuracy:', 76.31579041481018)
Mean Accuracy 71.36192798614582

```

## Data Science – DL – Save the model

---

### 7. Deep Learning – Save the model

#### Contents

1. Save the model .....	2
-------------------------	---

## Data Science – DL – Save the model

---

### 7. Deep Learning – Save the model

#### 1. Save the model

- ✓ Based on requirement we can save the model
- ✓ Generally, in deep learning process,
  - model will be saved into json file
  - model weights will be saved into HDF5(Hierarchical Data Format)
- ✓ HDF is more convenient for storing large arrays of real values, as we have in the weights of neural networks.

## Data Science – DL – Save the model

Program  
Name  
Input file

Save the model  
demo1.py  
pima-indians-diabetes.csv

```
# importing required libraries
from numpy import loadtxt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from keras.models import model_from_json

# load the dataset
dataset = loadtxt('pima-indians-diabetes.csv', delimiter = ',')

# split into input (X) and output (y) variables
X = dataset[:, 0:8]
y = dataset[:, 8]

# define the keras model

model = Sequential()

model.add(Dense(12, input_shape = (8, ), activation = 'relu'))
model.add(Dense(8, activation = 'relu'))
model.add(Dense(1, activation = 'sigmoid'))

# compile the keras model
model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = ['accuracy'])

model.fit(X, y, epochs = 150, batch_size = 10)

model_json = model.to_json()

# saving the model
with open("model.json", "w") as json_file:
    json_file.write(model_json)

# saving the model weights
model.save_weights("model.weights.h5")

print("Saved model to disk")

print("Done")
```

## Data Science – DL – Save the model

### Output

```
Epoch 147/150
77/77 [=====] - 0s 1ms/step - loss: 0.5480 - accuracy: 0.7201
Epoch 148/150
77/77 [=====] - 0s 1ms/step - loss: 0.5500 - accuracy: 0.7227
Epoch 149/150
77/77 [=====] - 0s 1ms/step - loss: 0.5493 - accuracy: 0.7292
Epoch 150/150
77/77 [=====] - 0s 1ms/step - loss: 0.5509 - accuracy: 0.7279
Saved model to disk
Done
```

### model.json

- ✓ The json format of the model looks like the following:

```
{
  "class_name": "Sequential",
  "config": {
    "name": "sequential",
    "layers": [
      {
        "class_name": "InputLayer",
        "config": {
          "batch_input_shape": [
            null,
            8
          ],
          "dtype": "float32",
          "sparse": false,
          "ragged": false,
          "name": "dense_input"
        }
      },
      {
        "class_name": "Dense",
        "config": {
          "name": "dense",
          "trainable": true,
          "batch_input_shape": [
            null,
            8
          ],
          "dtype": "float32",
          "units": 12,
        }
      }
    ]
  }
}
```

## Data Science – DL – Save the model

```
"activation": "relu",
"use_bias": true,
"kernel_initializer": {
  "class_name": "GlorotUniform",
  "config": {
    "seed": null
  }
},
"bias_initializer": {
  "class_name": "Zeros",
  "config": {}
},
"kernel_regularizer": null,
"bias_regularizer": null,
"activity_regularizer": null,
"kernel_constraint": null,
"bias_constraint": null
},
{
  "class_name": "Dense",
  "config": {
    "name": "dense_1",
    "trainable": true,
    "dtype": "float32",
    "units": 8,
    "activation": "relu",
    "use_bias": true,
    "kernel_initializer": {
      "class_name": "GlorotUniform",
      "config": {
        "seed": null
      }
    },
    "bias_initializer": {
      "class_name": "Zeros",
      "config": {}
    },
    "kernel_regularizer": null,
    "bias_regularizer": null,
    "activity_regularizer": null,
    "kernel_constraint": null,
    "bias_constraint": null
  }
},
{
  "class_name": "Dense",
```

## Data Science – DL – Save the model

```
"config": {  
    "name": "dense_2",  
    "trainable": true,  
    "dtype": "float32",  
    "units": 1,  
    "activation": "sigmoid",  
    "use_bias": true,  
    "kernel_initializer": {  
        "class_name": "GlorotUniform",  
        "config": {  
            "seed": null  
        }  
    },  
    "bias_initializer": {  
        "class_name": "Zeros",  
        "config": {}  
    },  
    "kernel_regularizer": null,  
    "bias_regularizer": null,  
    "activity_regularizer": null,  
    "kernel_constraint": null,  
    "bias_constraint": null  
},  
}  
]  
},  
"keras_version": "2.8.0",  
"backend": "tensorflow"  
}
```

## Data Science – DL – Save the model

**Program** Loading the model from json file  
**Name** demo2.py  
**Input file** pima-indians-diabetes.csv

```
# importing required libraries
from keras.models import model_from_json
import numpy as np

# load the dataset
dataset = np.loadtxt('pima-indians-diabetes.csv', delimiter = ',')

# split into input (X) and output (y) variables
X = dataset[:, 0:8]
y = dataset[:, 8]

json_file = open('model.json' , 'r')

model_j = json_file.read()
model = model_from_json(model_j)
model.load_weights("model.weights.h5")
print("Loaded model from disk")

model.compile(loss = 'binary_crossentropy' , optimizer = 'rmsprop' ,
metrics=['accuracy'])
score = model.evaluate(X, y)

print(score)
```

**Output**

```
Loaded model from disk
24/24 [=====] - 0s 924us/step - loss: 0.4692 - accuracy: 0.782
[0.46923649311065674, 0.7825520634651184]
```

## Data Science – DL – Best Model With Check Point

---

### 8. Deep Learning – Best Model with Check Pointing

#### Contents

1. Check point .....	2
----------------------	---

## Data Science – DL – Best Model With Check Point

---

### 8. Deep Learning – Best Model with Check Point

#### 1. Check point

- ✓ Deep learning models can take hours, days or even weeks to train and if a training run is stopped unexpectedly, you can lose a lot of work.
- ✓ Now we need to learn how to how we can checkpoint the deep learning models during training.
- ✓ The checkpoint captures the weights of the model.
- ✓ These weights can be used to make predictions for ongoing training.

## Data Science – DL – Best Model With Check Point

---

Program Name Input file	Checkpoint Neural Network Model Improvements demo1.py pima-indians-diabetes.csv
----------------------------	---

```

# importing required libraries
from numpy import loadtxt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from keras.callbacks import ModelCheckpoint

# load the dataset
dataset = loadtxt('pima-indians-diabetes.csv', delimiter = ',')

# split into input (X) and output (y) variables
X = dataset[:, 0:8]
y = dataset[:, 8]

# define the keras model

model = Sequential()

model.add(Dense(12, input_shape = (8, ), activation = 'relu'))
model.add(Dense(8, activation = 'relu'))
model.add(Dense(1, activation = 'sigmoid'))

# compile the keras model
model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = ['accuracy'])

# checkpoint
filepath = "weights-improvement-{epoch:02d}-{val_accuracy:.2f}.keras"

checkpoint = ModelCheckpoint(filepath, monitor= 'val_accuracy' , verbose=1,
save_best_only = True, mode= 'max' )

callbacks_list = [checkpoint]

model.fit(X, y, validation_split=0.33, epochs=150, batch_size=10, callbacks =
callbacks_list, verbose=0)

print("Done")

```

## Data Science – DL – Best Model With Check Point

---

### Output

.....

.....

Epoch 102: val\_accuracy did not improve from 0.74016

Epoch 103: val\_accuracy did not improve from 0.74016

Epoch 104: val\_accuracy improved from 0.74016 to 0.74803, saving model to weights-improvement-104-0.75.hdf5

Epoch 105: val\_accuracy did not improve from 0.74803

Epoch 106: val\_accuracy did not improve from 0.74803

Epoch 107: val\_accuracy did not improve from 0.74803

Epoch 108: val\_accuracy did not improve from 0.74803

## Data Science – DL – Visualize model training history

---

### 9. Deep Learning – Visualize model training history

#### Contents

1. Visualize the model accuracy and loss .....	2
--	---

## Data Science – DL – Visualize model training history

---

### 9. Deep Learning – Visualize model training history

#### 1. Visualize the model accuracy and loss

- ✓ We can create plots from the collected history data.
- ✓ We can plot the neural network to model for the Pima Indians onset of diabetes binary classification problem
- ✓ The example collects the history and create two charts
  - A plot of accuracy on the training and validation datasets over training epochs
  - A plot of loss on the training and validation datasets over training epochs

## Data Science – DL – Visualize model training history

---

Program	Visualize model training history
Name	demo1.py
Input file	pima-indians-diabetes.csv

```

# importing required libraries
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import matplotlib.pyplot as plt
import numpy as np

# load pima Indians dataset
dataset = np.loadtxt("pima-indians-diabetes.csv", delimiter=",")

# split into input (X) and output (Y) variables
X = dataset[:,0:8]
Y = dataset[:,8]

# create model
model = Sequential()

model.add(Dense(12, input_shape=(8,), activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])

# Fit the model
history = model.fit(X, Y, validation_split=0.33, epochs=150, batch_size=10,
verbose=0)

# list all data in history
print(history.history.keys())

# summarize history for ACCURACY
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])

plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

```

## Data Science – DL – Visualize model training history

```
plt.show()

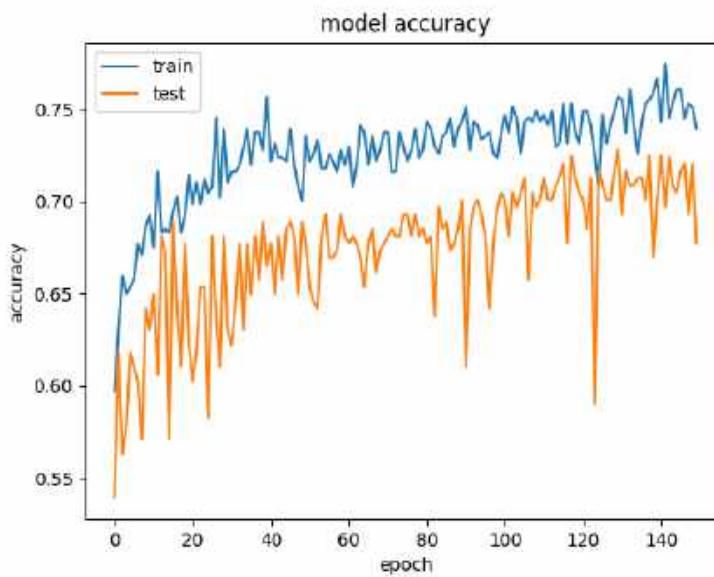
# summarize history for LOSS
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])

plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')

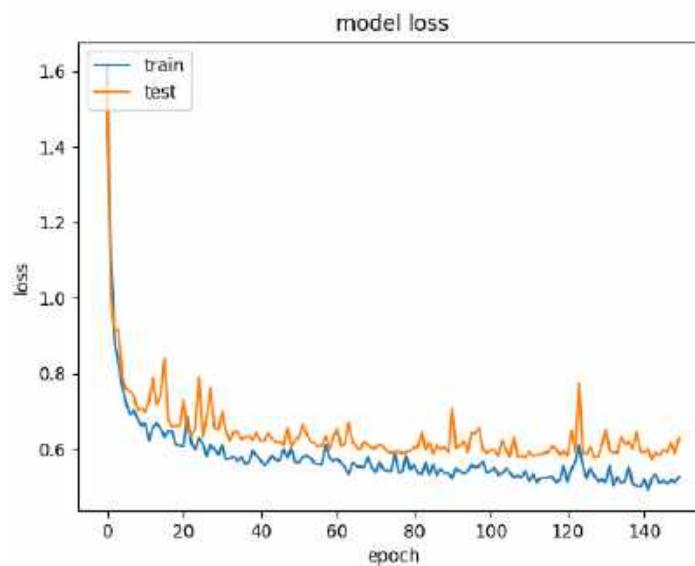
plt.legend(['train', 'test'], loc='upper left')
plt.show()

print("Done")
```

### Output



## Data Science – DL – Visualize model training history



# Data Science – NLP - Introduction

---

## 1. NLP - Introduction

### Contents

1. Natural Language .....	2
2. Now think about speech.....	2
3. Text data.....	2
4. Natural Language Processing .....	2
5. NLP popularity is raising everyday.....	2
6. IMP components in NLP .....	3
7. Semantics.....	3
8. Deep learning importance .....	3
9. NLTK.....	3

## Data Science – NLP - Introduction

---

### 1. NLP – Introduction

#### **1. Natural Language**

- ✓ We are all used communicate with natural language.
  - We may speak to each other on daily base for different purpose.
  - It is very easy to speak than to write.
  - We are surrounded by text.
  
- ✓ Think about how much text you see each day:
  - Signs/signals
  - Menus
  - Email
  - SMS
  - Web Pages and much more...
  - The list is endless.

#### **2. Now think about speech.**

- ✓ We may speak to each other on daily base for different purpose.
- ✓ It is very easy to speak than to write.

#### **3. Text data**

- ✓ We need understand more about text related data.
- ✓ Text data also having separate methodologies to analyse the data.

#### **4. Natural Language Processing**

- ✓ The short name for Natural Language Processing is NLP.
- ✓ NLP is a way of computers to **analyse**, **understand** and **derive** meaning from human languages such as English, Spanish, Hindi, etc.
- ✓ With NLP, machine learning algorithms can be applied to speech and text.

#### **5. NLP popularity is raising everyday**

- ✓ Yes, the popularity of NLP is rising every day.
- ✓ With NLP, a computer can understand the human language while speaking
- ✓ By using big data we can make communication in between human and machine.
- ✓ With NLP an intelligent system such as a robot can perform according to our instructions issued in a plain language such as English.

## Data Science – NLP - Introduction

---

### 6. IMP components in NLP

- ✓ There are mainly two components in NLP
  - Syntax
  - Semantic analysis

#### Syntax

- ✓ Syntax explains about the arrangement of the words in sentence and grammar as well
- ✓ NLP makes use of syntax to analyse to get the meaning from a language depending on grammatical rules.
- ✓ Some syntax techniques that are used for this include,
  - Parsing,
  - Word segmentation,
  - Sentence breaking,
  - Morphological segmentation and stemming.

### 7. Semantics

- ✓ Semantic is associated with the use and the meaning of various words.
- ✓ In NLP, algorithms are used to analyse and determine the meaning and structure of sentences.
- ✓ Some NLP techniques used for semantics include word understanding, named entity recognition and natural language generation.

### 8. Deep learning importance

- ✓ Most of the current approaches to NLP are using deep learning.
- ✓ Deep learning is a type of artificial intelligence that relies on the patterns in data to improve the understanding of a program.
- ✓ Deep learning models usually require huge amounts of labelled data to train and identify any correlations between the data elements.

### 9. NLTK

- ✓ NLTK means Natural Language Toolkit.
- ✓ It is a python module to work with nlp
- ✓ It is open source library
- ✓ To install this library,
  - pip install nltk

## Data Science – NLP – Text wrangling and Cleaning

---

### 2. NLP – Text wrangling and cleaning

#### Contents

1. Text wrangling and cleansing.....	2
2. Sentence Splitting.....	3
3. Tokenization .....	5
4. Word tokenization .....	6
5. Stemming.....	8
6. Lemmatization.....	10
7. Stop word removal.....	12

## Data Science – NLP – Text wrangling and Cleaning

---

### 2. NLP – Text wrangling and cleaning

#### 1. Text wrangling and cleansing

- ✓ The ultimate goal of doing text processing is to, machine can understand the text data.
- ✓ Text processing is very important as it helps us understand our data better and gain insights from it.
- ✓ Text processing normally involves the following steps,
  - Tokenization
  - Lemmatization
  - Stemming
  - Stop word removal

<b>Program Name</b>	Downloading punkt demo1.py
	<pre>import nltk nltk.download("punkt")</pre>
<b>Output</b>	[nltk_data] Downloading package punkt to [nltk_data]   C:\Users\admin\AppData\Roaming\nltk_data... [nltk_data] Package punkt is already up-to-date!

- ✓ We have then downloaded the punkt, which is a Tokenizer for a text into a list of sentences.
- ✓ We can now do text wrangling.

## Data Science – NLP – Text wrangling and Cleaning

---

### 2. Sentence Splitting

- ✓ Generally we used to understand sentence by reading words
- ✓ We will start reading sentence to understand paragraph.
- ✓ So, a paragraph we need to split it into a set of sentences.
- ✓ So, let's understand words, sentences and paragraph.
- ✓ This we can easily do by using nltk package

<b>Program Name</b>	Sentence splitting by using split() method demo2.py
	myString = "This is a paragraph. It should split at the end of sentence marker, such as a period. It can tell that the period in Mr.Daniel is not an end. Run it!, Hey How are you doing"
<b>Output</b>	result = myString.split(".") print(result)

<b>Program Name</b>	Sentence splitting demo3.py
	from nltk.tokenize import sent_tokenize
<b>Output</b>	myString = "This is a paragraph. It should split at the end of sentence marker, such as a period. It can tell that the period in Mr.Daniel is not an end. Run it!, Hey How are you doing"

```
tokenized_sentence = sent_tokenize(myString)
print(tokenized_sentence)
```

<b>Output</b>	['This is a paragraph.', 'It should split at the end of sentence marker, such as a period.', 'It can tell that the period in Mr.Daniel is not an end.', 'Run it!, Hey How are you doing']
---------------	---

## Data Science – NLP – Text wrangling and Cleaning

---

### Note

- ✓ We can understand from above output, the paragraph was split into the exact sentences.
- ✓ It was also able to tell the difference between a period that has been used to end a sentence from one used on the name Mr.Daniel also

## Data Science – NLP – Text wrangling and Cleaning

---

### 3. Tokenization

- ✓ Tokenization refers to the process by which a large text is broken down into various pieces.
- ✓ In NLP, a token is the minimal piece of text that a machine can be able to understand.
- ✓ A text may be tokenized into words or sentences.
  - In the case of sentence tokenization, every sentence will be identified as a token.
  - In the case of word tokenization, every word will be identified as a token.

## Data Science – NLP – Text wrangling and Cleaning

---

### 4. Word tokenization

- ✓ There are various ways through which tokenization can be done, but the most popular way of doing it is through word tokenization.
- ✓ What happens in word tokenization is that a large text is broken down into words and the words are used as the tokens.
  - word\_tokenizer
  - sent\_tokenizer
  - punkt\_tokenizer
  - Regexp\_tokenizer
- ✓ To tokenize a text, we can still apply split() method from string

<b>Program Name</b>	Word tokenization demo4.py
	<pre>myString = "These are sentences. Let us tokenize it! Run it!"  print(myString.split())</pre>
<b>Output</b>	<pre>['These', 'are', 'sentences.', 'Let', 'us', 'tokenize', 'it!', 'Run', 'it!']</pre>

<b>Program Name</b>	Word tokenization with word_tokenize function demo5.py
	<pre>from nltk.tokenize import word_tokenize  myString = "These are sentences. Let us tokenize it! Run it!"  print(word_tokenize(myString))</pre>
<b>Output</b>	<pre>['These', 'are', 'sentences', '!', 'Let', 'us', 'tokenize', 'it', '!', 'Run', 'it', '!']</pre>

## Data Science – NLP – Text wrangling and Cleaning

**Program Name** Word tokenization with regexp\_tokenize function  
demo6.py

```
from nltk.tokenize import regexp_tokenize

myString = "These are sentences. Let us tokenize it! Run it!"

print(regexp_tokenize(myString, pattern="\w+"))
```

**Output**

```
['These', 'are', 'sentences', 'Let', 'us', 'tokenize', 'it', 'Run', 'it']
```

**Program Name** Capture the digit from sentence  
demo7.py

```
from nltk.tokenize import regexp_tokenize

myString = "These are 3 sentences. Let us tokenize them! Run the code!"

print(regexp_tokenize(myString, pattern="\d+"))
```

**Output**

```
['3']
```

## Data Science – NLP – Text wrangling and Cleaning

---

### 5. Stemming

- ✓ This is another step in text wrangling.
- ✓ From the name, you can get the meaning, cutting down a token to its root stem.
- ✓ Stemming algorithm works by cutting the suffix from the word.
- ✓ In simple, it cuts either the beginning or end of the word.
- ✓ Consider the word “**cutting**”.
- ✓ This word can be broken down to its root, which is “**cut**”.

<b>Program Name</b>	Stemming demo8.py
<pre>from nltk.stem import PorterStemmer  porter = PorterStemmer() print(porter.stem("cutting"))</pre>	
<b>Output</b>	cut

<b>Program Name</b>	Stemming demo9.py
<pre>from nltk.stem import PorterStemmer  e_words= ["wait", "waiting", "waited", "waits"]  ps = PorterStemmer()  for w in e_words:     rootWord = ps.stem(w)     print(rootWord)</pre>	
<b>Output</b>	wait wait wait wait

## Data Science – NLP – Text wrangling and Cleaning

Program Name

Stemming  
demo10.py

```
import nltk
from nltk.stem.porter import PorterStemmer

porter_stemmer = PorterStemmer()
text = "studies studying cries cry"
tokenization = nltk.word_tokenize(text)

for w in tokenization:
    print("Stemming for {} is {}".format(w, porter_stemmer.stem(w)))
```

Output

```
Stemming for studies is studi
Stemming for studying is studi
Stemming for cries is cri
Stemming for cry is cri
```

- ✓ Stemming is good for its simplicity when it comes to dealing with NLP problems.
- ✓ However, when more complex stemming is needed, stemming is not the best option, but we have lemmatization.

## Data Science – NLP – Text wrangling and Cleaning

### 6. Lemmatization

- ✓ Lemmatization is a more advanced compared to stemming.
- ✓ It follows specific rules to get results.
- ✓ It understands the context, parts of the speech to understand the root of the word

**Program Name**

Lemmatization

demo11.py

```
from nltk.stem import WordNetLemmatizer  
  
wl = WordNetLemmatizer()  
  
print("rocks :", wl.lemmatize("rocks"))  
print("corpora :", wl.lemmatize("corpora"))  
print("better :", wl.lemmatize("better", pos = "a"))
```

**Output**

```
rocks : rock  
corpora : corpus  
better : good
```

## Data Science – NLP – Text wrangling and Cleaning

**Program Name**

Lemmatization

demo12.py

```
import nltk
from nltk.stem import WordNetLemmatizer

wl = WordNetLemmatizer()

text = "studies studying cries cry"

tokens = nltk.word_tokenize(text)

for word in tokens:
    print("Lemmatization for {} is {}".format(word, wl.lemmatize(word)))
```

**Output**

```
Lemmatization for studies is study
Lemmatization for studying is studying
Lemmatization for cries is cry
Lemmatization for cry is cry
```

## Data Science – NLP – Text wrangling and Cleaning

---

### 7. Stop word removal

- ✓ Some common words that are present in text but do not contribute in the meaning of a sentence.
- ✓ Such words are not at all important for the purpose of information retrieval or natural language processing.
- ✓ The most common stopwords are 'a', 'in' 'the' etc
- ✓ The good news is that nltk comes with a list of stop words and in different languages.

<b>Program Name</b>	list of the stopwords demo13.py
	<pre>import nltk from nltk.corpus import stopwords  print(stopwords.words('english'))</pre>
<b>Output</b>	<pre>['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]</pre>

## Data Science – NLP – Text wrangling and Cleaning

**Program Name** list of the languages  
demo13.py

```
from nltk.corpus import stopwords

langs = stopwords.fileids()
print(len(langs))
```

**Output**

29

**Program Name** Lemmatization  
demo14.py

```
import nltk
from nltk.corpus import stopwords

nltk.download('stopwords')

mylist = stopwords.words('english')

line = "This is really good, how are you doing"

postPa = [word for word in line.split()]

print(postPa)
```

**Output**

['This', 'is', 'really', 'good,', 'how', 'are', 'you', 'doing']

## Data Science – NLP – Text wrangling and Cleaning

<b>Program Name</b>	Lemmatization demo15.py
<pre>import nltk from nltk.corpus import stopwords  nltk.download('stopwords')  mylist = stopwords.words('english')  line = "This is really good, how are you doing"  postPa = [word      for word in line.split()      if word not in mylist]  print(postPa)</pre>	
<b>Output</b>	[This', 'really', 'good,']

# Data Science – NLP – Replacing and Correcting words

---

## 3. NLP – Replacing and Correcting words

### Contents

1. Text conversions.....	2
2. Removing numbers .....	3
3. Removing punctuations .....	4
4. Removing whitespaces.....	5
5. Part of Speech Tagging (POS).....	6
6. Information Extraction .....	9
7. Information extraction has many applications including .....	10
8. Collocations: Bigrams and Trigrams .....	11
9. Wordnet.....	13

## Data Science – NLP – Replacing and Correcting words

### 3. NLP – Replacing and correcting words

#### 1. Text conversions

- ✓ We can convert the text from lower to upper and upper to lower case

**Program Name** Converting lower case to upper case  
demo1.py

```
text ="hello good morning"  
print(text.upper())
```

**Output**

HELLO GOOD MORNING

**Program Name** Converting upper case to lower case  
demo2.py

```
text ="HELLO GOOD MORNING"  
print(text.lower())
```

**Output**

hello good morning

## Data Science – NLP – Replacing and Correcting words

### 2. Removing numbers

- ✓ By using regular expression we can remove numbers from the text

**Program**

Removing numbers from the text

**Name**

demo3.py

```
import re
```

```
myString = 'Box A has 4 red and 6 white balls, while Box B has 3 red and 5  
blue balls.'
```

```
output = re.sub(r'\d+', '', myString)  
print(output)
```

**Output**

```
Box A has red and white balls, while Box B has red and blue balls.
```

## Data Science – NLP – Replacing and Correcting words

### 3. Removing punctuations

- ✓ By using regular expression we can remove the punctuations from the text

**Program Name** Removing the punctuations from the text  
demo4.py

```
import re

text = "Hello $@## Good !@#!@# morning #*#@&@#"

print("Text is:", text)

res = re.sub(r'[^w\s]', " ", text )

print("After punctuations:", res)
```

**Output**

```
Text is: Hello $@## Good !@#!@# morning #*#@&@#
After punctuations: Hello Good morning
```

## Data Science – NLP – Replacing and Correcting words

### 4. Removing whitespaces

- ✓ We can remove the whitespaces in string by using strip() method.

**Program** Removing whitespaces from text

**Name** demo5.py

```
text = "      a sample string      "

print(text)
res = text.strip()
print(res)
```

**Output**

```
      a sample string
a sample string
```

## Data Science – NLP – Replacing and Correcting words

---

### 5. Part of Speech Tagging (POS)

- ✓ The goal of POS is to assign the various parts of a speech to every word of the provided text like nouns, adjectives, verbs, etc.
- ✓ This is normally done based on the definition and the context.
- ✓ Install textblob library,
  - pip install textblob

<b>Program Name</b>	Removing whitespaces from text demo6.py
	<pre>from textblob import TextBlob import nltk  nltk.download('averaged_perceptron_tagger')  myString = "Parts of speech: an article, to run, fascinating, quickly, and, of"  output = TextBlob(myString) print(output.tags)</pre>
<b>Output</b>	<pre>[('Parts', 'NNS'), ('of', 'IN'), ('speech', 'NN'), ('an', 'DT'), ('article', 'NN'), ('to', 'TO'), ('run', 'VB'), ('fascinating', 'VBG'), ('quickly', 'RB'), ('and', 'CC'), ('of', 'IN')]</pre>

## Data Science – NLP – Replacing and Correcting words

---

Some examples are as below:

Abbreviation	Meaning
CC	coordinating conjunction
CD	cardinal digit
DT	determiner
EX	existential there
FW	foreign word
IN	preposition/subordinating conjunction
JJ	adjective (large)
JJR	adjective, comparative (larger)
JJS	adjective, superlative (largest)
LS	list marker
MD	modal (could, will)
NN	noun, singular (cat, tree)
NNS	noun plural (desks)
NNP	proper noun, singular (sarah)
NNPS	proper noun, plural (indians or americans)
PDT	predeterminer (all, both, half)
POS	possessive ending (parent's)
PRP	personal pronoun (hers, herself, him,himself)
PRPS	possessive pronoun (her, his, mine, my, our )
RB	adverb (occasionally, swiftly)
RBR	adverb, comparative (greater)
RBS	adverb, superlative (biggest)
RP	particle (about)
TO	infinite marker (to)
UH	interjection (goodbye)

## Data Science – NLP – Replacing and Correcting words

VB	verb (ask)
VBG	verb gerund (judging)
VBD	verb past tense (pleaded)
VBN	verb past participle (reunified)
VBP	verb, present tense not 3rd person singular(wrap)
VBZ	verb, present tense with 3rd person singular (bases)
WDT	wh-determiner (that, what)
WP	wh- pronoun (who)
WRB	wh- adverb (how)

**Program Name** pos example  
demo7.py

```
from nltk.corpus import wordnet

syn = wordnet.synsets('hello')[0]
print("Syn tag : ", syn.pos())

syn = wordnet.synsets('doing')[0]
print("Syn tag : ", syn.pos())

syn = wordnet.synsets('beautiful')[0]
print("Syn tag : ", syn.pos())
```

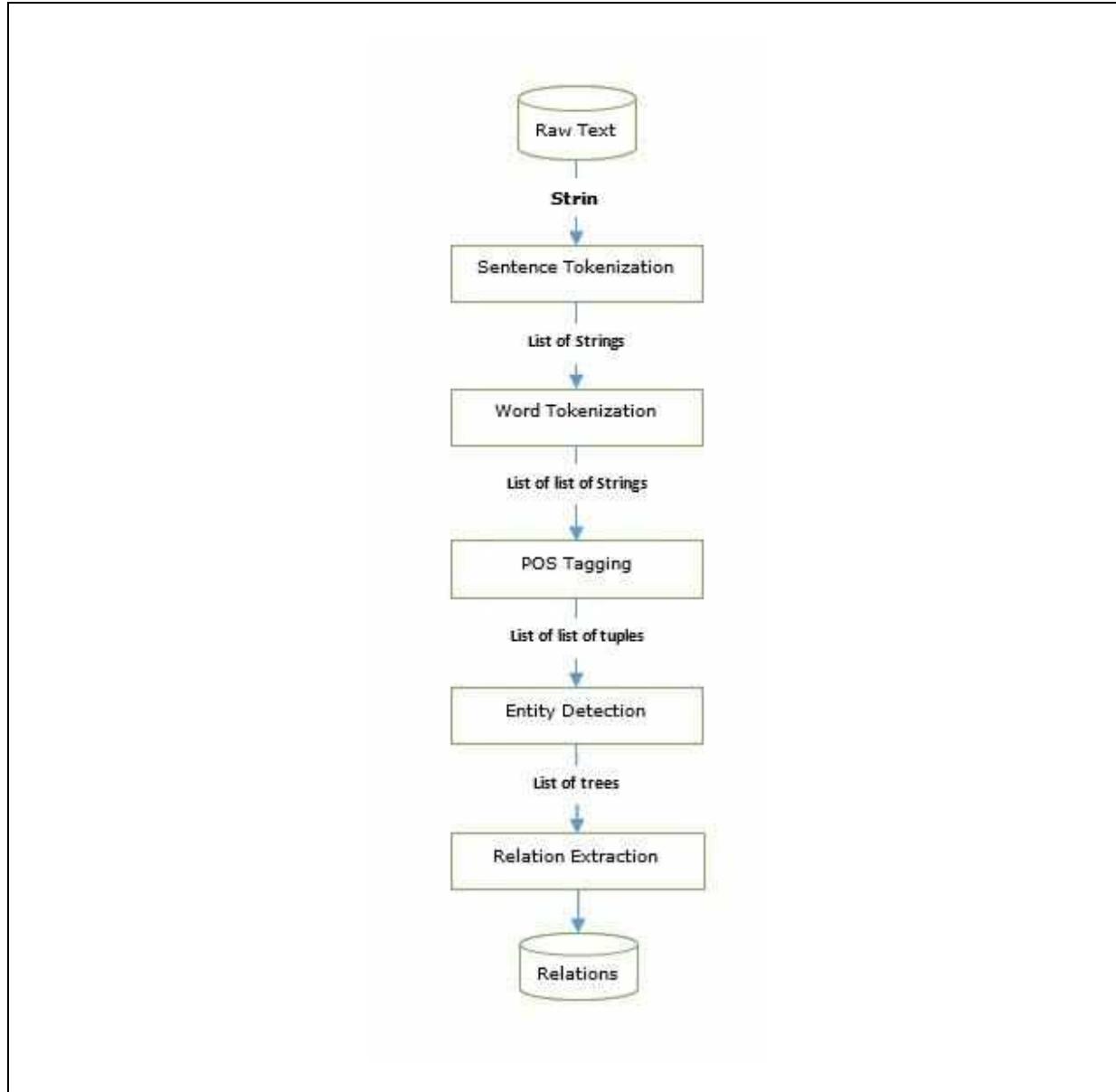
**Output**

Syn tag : n  
Syn tag : v  
Syn tag : a  
Syn tag : r

## Data Science – NLP – Replacing and Correcting words

### 6. Information Extraction

- ✓ We need to understand the tags and parsers to build information extraction engine.
- ✓ Let us see a basic information extraction pipeline



## Data Science – NLP – Replacing and Correcting words

---

### 7. Information extraction has many applications including

- ✓ Business intelligence
- ✓ Resume harvesting
- ✓ Media analysis
- ✓ Sentiment detection
- ✓ Patent search
- ✓ Email scanning

## Data Science – NLP – Replacing and Correcting words

---

### 8. Collocations: Bigrams and Trigrams

#### What is Collocations?

- ✓ Collocations are the pairs of words occurring together many times in paragraphs.
- ✓ It is calculated by the number of those pair occurring together to the overall word count of the paragraph.
- ✓ We can say that finding collocations requires calculating the frequencies of words and their appearance in the context of other words.

#### Bigrams and Trigrams

- ✓ Collocation can be categorized into two types,
  - Bigrams combination of two words
  - Trigrams combination of three words
- ✓ Bigrams and Trigrams provide more meaningful and useful features for the feature extraction stage.
- ✓ These are especially useful in text-based sentimental analysis.

<b>Program Name</b>	Bigram example demo8.py
<pre><code>import nltk  text = "Data Science is a totally new kind of learning experience." Tokens = nltk.word_tokenize(text) output = list(nltk.bigrams(Tokens))  print(output)</code></pre>	
<b>Output</b>	
<pre><code>[('Data', 'Science'), ('Science', 'is'), ('is', 'a'), ('a', 'totally'), ('totally', 'new'), ('new', 'kind'), ('kind', 'of'), ('of', 'learning'), ('learning', 'experience'), ('experience', '.')]</code></pre>	

## Data Science – NLP – Replacing and Correcting words

**Program Name** Trigram example  
demo9.py

```
import nltk

text = "Data Science is a totally new kind of learning experience."
Tokens = nltk.word_tokenize(text)
output = list(nltk.trigrams(Tokens))

print(output)
```

**Output**

```
[('Data', 'Science', 'is'), ('Science', 'is', 'a'), ('is', 'a', 'totally'), ('a', 'totally', 'new'), ('totally', 'new', 'kind'), ('new', 'kind', 'of'), ('kind', 'of', 'learning'), ('of', 'learning', 'experience'), ('learning', 'experience', '.')]
```

## Data Science – NLP – Replacing and Correcting words

### 9. Wordnet

- ✓ Wordnet is an NLTK lexical database for English.
- ✓ It can be used to find the meaning of words, synonym or antonym.

#### synset

- ✓ Synset is a special kind of a simple interface that is present in NLTK to look up words in Wordnet.
- ✓ Synset instances are the groupings of synonymous words that express the same concept.

#### Program

Name wordnet example

demo10.py

```
from nltk.corpus import wordnet  
  
syn = wordnet.synsets('hello')[0]  
  
print ("Synset name :", syn.name())  
print ("Synset meaning :", syn.definition())  
print ("Synset example :", syn.examples())
```

#### Output

```
Synset name : hello.n.01  
Synset meaning : an expression of greeting  
Synset example : ['every morning they exchanged polite hellos']
```

## Data Science – NLP – Replacing and Correcting words

---

<b>Program Name</b>	wordnet example demo11.py
	<pre>from nltk.corpus import wordnet  syn = wordnet.synsets('boy')[0]  print ("Synset name :", syn.name()) print ("Synset meaning :", syn.definition()) print ("Synset example :", syn.examples())</pre>
<b>Output</b>	Synset name : male_child.n.01     Synset meaning : a youthful male person     Synset example : ['the baby was a boy', 'she made the boy brush his teeth every night', 'most soldiers are only boys in uniform']

<b>Program Name</b>	wordnet example demo12.py
	<pre>from nltk.corpus import wordnet  syn = wordnet.synsets('good')[0]  print ("Synset name :", syn.name()) print ("Synset meaning :", syn.definition()) print ("Synset example :", syn.examples())</pre>
<b>Output</b>	Synset name : good.n.01     Synset meaning : benefit     Synset example: ['for your own good', "what's the good of worrying?"]

## Data Science – NLP – Components in NLP

---

### 4. NLP – Components in NLP

#### Contents

<b>1. Common nlp libraries .....</b>	<b>2</b>
<b>2. Components in NLP .....</b>	<b>2</b>
2.1. Lexical Analysis.....	3
2.2. Syntactic Analysis.....	3
2.3. Semantic Analysis.....	3
2.4. Discourse Analysis.....	3
2.5. Pragmatic Analysis .....	3
<b>3. Use case .....</b>	<b>4</b>
<b>4. Word Cloud.....</b>	<b>17</b>

## Data Science – NLP – Components in NLP

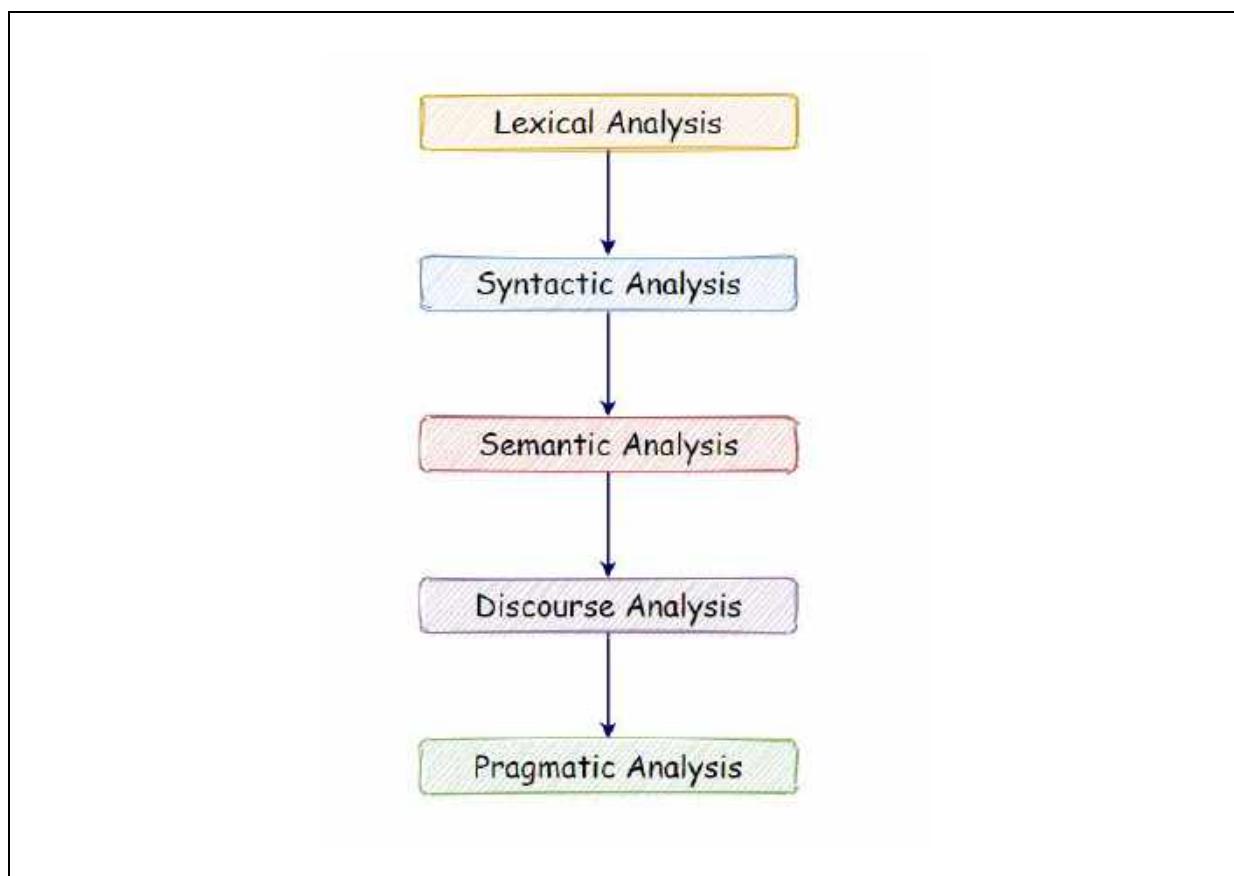
### 4. NLP – Components in NLP

#### 1. Common nlp libraries

- ✓ NLTK library
- ✓ Spacy library
- ✓ TextBlob library
- ✓ Gensim framework
- ✓ Pattern framework

#### 2. Components in NLP

- ✓ There are mainly five components in NLP
  - Lexical Analysis
  - Syntactic Analysis
  - Semantic Analysis
  - Discourse Analysis
  - Pragmatic analysis



## Data Science – NLP – Components in NLP

---

### 2.1. Lexical Analysis

- ✓ Lexical means relating to the words or vocabulary of a language.
- ✓ With lexical analysis, we divide a whole chunk of text into,
  - Paragraphs.
  - Sentences.
  - Words.
- ✓ It involves identifying and analysing words' structure.

### 2.2. Syntactic Analysis

- ✓ Syntactic means according to the syntax
- ✓ In syntactic analysis, it is the process of analysing the words in sentence.
- ✓ Analysing the grammar and arranging the words in a manner that shows the relationship among the words.
- ✓ Example
  - The sentence “The shop goes to the house” does not pass means invalid

### 2.3. Semantic Analysis

- ✓ Syntactic means relating to meaning in language.
- ✓ Semantic analysis draws the exact meaning for the **words**, and it analyses the text to get meaningful.
- ✓ Example
  - The sentences such as “hot ice-cream” do not pass or invalid

### 2.4. Discourse Analysis

- ✓ Discourse means written or spoken communication.
- ✓ It considers the meaning of the **sentence** before it ends.
- ✓ Example
  - The sentences such as “He works at Google” in this sentence “he” should be first word in sentence

### 2.5. Pragmatic Analysis

- ✓ Pragmatic means practical, especially when making decisions
- ✓ Pragmatic analysis deals with overall communication and interpretation of language.
- ✓ It deals with deriving meaningful use of language in various situations.

## Data Science – NLP – Components in NLP

### 3. Use case

- ✓ Process the below textual information

#### story\_input.txt

Once upon a time there was an old mother pig that had three little pigs and not enough food to feed them. So, when they were old enough, she sent them out into the world to seek their fortunes.

The first little pig was very lazy. He didn't want to work at all and he built his house out of straw. The second little pig worked a little bit harder but he was somewhat lazy too and he builds his house out of sticks. Then, they sang and danced and played together and rest of the day.

The third little pig worked hard all day and built his house with bricks. It was a sturdy house complete with a fine fireplace and chimney. It looked like it could withstand the strongest winds.

## Data Science – NLP – Components in NLP

**Program Name** Reading textual information from file  
demo1.py

```
data = open("story_input.txt")  
  
text = data.read()  
  
print(text)  
print()  
print(type(text))  
print("Length of the text is:", len(text))
```

**Output**

```
Once upon a time there was an old mother pig that had three little pigs and not enough food to feed them. So, when they were old enough, she sent them out into the world to seek their fortunes.  
  
The first little pig was very lazy. He didn't want to work at all and he built his house out of straw. The second little pig worked a little bit harder but he was somewhat lazy too and he builds his house out of sticks. Then, they sang and danced and played together and rest of the day.  
  
The third little pig worked hard all day and built his house with bricks. It was a sturdy house complete with a fine fireplace and chimney. It looked like it could withstand the strongest winds.  
<class 'str'>  
Length of the text is: 678
```

## Data Science – NLP – Components in NLP

**Program Name** Sentence tokenization  
demo2.py

```
import nltk
from nltk import sent_tokenize

data = open("story_input.txt")
text = data.read()

sentences = sent_tokenize(text)

print("Total sentences:", len(sentences))

for sent in sentences:
    print(sent)
```

**Output**

```
Total sentences: 9
Once upon a time there was an old mother pig that had three little pigs and not enough food to feed them.
So, when they were old enough, she sent them out into the world to seek their fortunes.
The first little pig was very lazy.
He didn't want to work at all and he built his house out of straw.
The second little pig worked a little bit harder but he was somewhat lazy too and he builds his house out of sticks.
Then, they sang and danced and played together and rest of the day.
The third little pig worked hard all day and built his house with bricks.
It was a sturdy house complete with a fine fireplace and chimney.
It looked like it could withstand the strongest winds.
```

## Data Science – NLP – Components in NLP

---

**Program Name** Word tokenization  
demo3.py

```
import nltk
from nltk import word_tokenize

data = open("story_input.txt")
text = data.read()

words = word_tokenize(text)
print(words)
```

**Output**

```
['Once', 'upon', 'a', 'time', 'there', 'was', 'an', 'old', 'mother', 'pig', 'that', 'had', 'three', 'little', 'pigs', 'and', 'not', 'enough', 'food', 'to', 'feed', 'them', 'So', ' ', 'when', 'they', 'were', 'old', 'enough', ' ', 'she', 'sent', 'them', 'out', 'into', 'the', 'world', 'to', 'seek', 'their', 'fortunes', '.', 'The', 'first', 'little', 'pig', 'was', 'very', 'lazy', '.', 'He', 'did', 'n\'t', 'want', 'to', 'work', 'at', 'all', 'and', 'he', 'built', 'his', 'house', 'out', 'of', 'straw', '.', 'The', 'second', 'little', 'pig', 'worked', 'a', 'little', 'bit', 'harder', 'but', 'he', 'was', 'somewhat', 'lazy', 'too', 'and', 'he', 'builds', 'his', 'house', 'out', 'of', 'sticks', '.', 'Then', ' ', 'they', 'sang', 'and', 'danced', 'and', 'played', 'together', 'and', 'rest', 'of', 'the', 'day', ' ', 'The', 'third', 'little', 'pig', 'worked', 'hard', 'all', 'day', 'and', 'built', 'his', 'house', 'with', 'bricks', '.', 'It', 'was', 'a', 'sturdy', 'house', 'complete', 'with', 'a', 'fine', 'fireplace', 'and', 'chimney', '.', 'It', 'looked', 'like', 'it', 'could', 'withstand', 'the', 'strongest', 'winds', '.']
```

## Data Science – NLP – Components in NLP

**Program Name** Finding word frequency  
demo4.py

```
import nltk
from nltk import word_tokenize
from nltk.probability import FreqDist

data = open("story_input.txt")
text = data.read()

words = word_tokenize(text)

fdist = FreqDist(words)
result = fdist.most_common(10)
print(result)
```

**Output**

```
[('.', 9),
('and', 8),
('little', 5),
('a', 4),
('was', 4),
('pig', 4),
('house', 4),
('to', 3),
(' ', 3),
('out', 3)]
```

### Note

- ✓ Notice that the most used words are punctuation marks and stopwords.
- ✓ We will have to remove such words to analyse the actual text.

## Data Science – NLP – Components in NLP

**Program Name**

Plotting the frequency graph of words with punctuations  
demo5.py

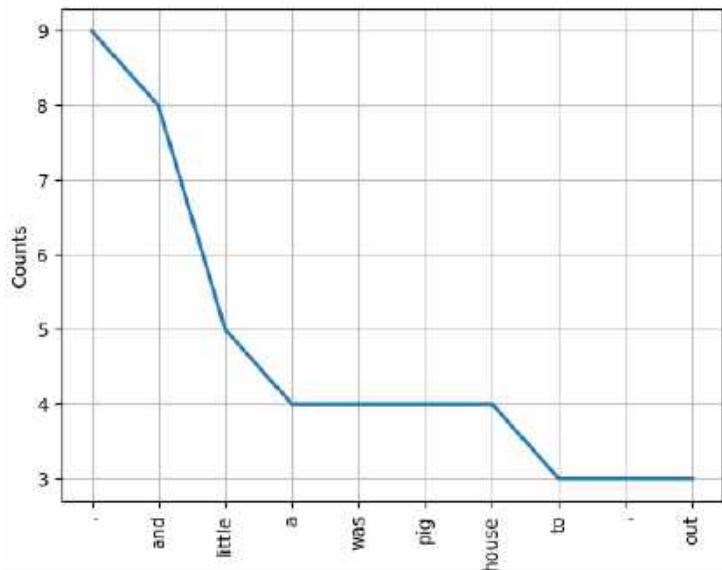
```
import nltk
from nltk import word_tokenize
from nltk.probability import FreqDist
import matplotlib.pyplot as plt

data = open("story_input.txt")
text = data.read()

words = word_tokenize(text)

fdist = FreqDist(words)
fdist.plot(10)
```

**Output**



- ✓ In the graph above, notice that a period “.” is used nine times in our text.
- ✓ Analytically speaking, punctuation marks are not that important for natural language processing.
- ✓ Therefore, in the next step, we will be removing such punctuation marks.

## Data Science – NLP – Components in NLP

---

**Program Name** Removing the punctuation marks  
demo6.py

```

import nltk
from nltk import word_tokenize

data = open("story_input.txt")
text = data.read()

words = word_tokenize(text)

words_no_punc = []

for w in words:
    if w.isalpha():
        words_no_punc.append(w.lower())

print(words_no_punc)
print("Length of words:", len(words_no_punc))

```

### Output

```

['once', 'upon', 'a', 'time', 'there', 'was', 'an', 'old', 'mother', 'pig', 'that', 'had', 'three', 'little', 'pigs', 'and', 'not', 'enough', 'food', 'to', 'feed', 'them', 'so', 'when', 'they', 'were', 'old', 'enough', 'she', 'sent', 'them', 'out', 'into', 'the', 'world', 'to', 'seek', 'their', 'fortunes', 'the', 'first', 'little', 'pig', 'was', 'very', 'lazy', 'he', 'did', 'want', 'to', 'work', 'at', 'all', 'and', 'he', 'built', 'his', 'house', 'out', 'of', 'straw', 'the', 'second', 'little', 'pig', 'worked', 'a', 'little', 'bit', 'harder', 'but', 'he', 'was', 'somewhat', 'lazy', 'too', 'and', 'he', 'builds', 'his', 'house', 'out', 'of', 'sticks', 'then', 'they', 'sang', 'and', 'danced', 'and', 'played', 'together', 'and', 'rest', 'of', 'the', 'day', 'the', 'third', 'little', 'pig', 'worked', 'hard', 'all', 'day', 'and', 'built', 'his', 'house', 'with', 'bricks', 'it', 'was', 'a', 'sturdy', 'house', 'complete', 'with', 'a', 'fine', 'fireplace', 'and', 'chimney', 'it', 'looked', 'like', 'it', 'could', 'withstand', 'the', 'strongest', 'winds']
Length of words: 132

```

## Data Science – NLP – Components in NLP

**Program Name**

Plotting the frequency graph of words without punctuations  
demo7.py

```
import nltk
from nltk import word_tokenize
from nltk.probability import FreqDist
import matplotlib.pyplot as plt

data = open("story_input.txt")
text = data.read()

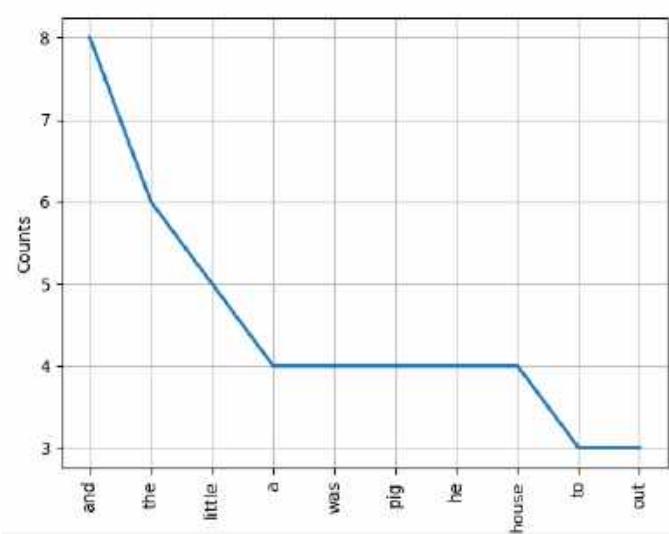
words = word_tokenize(text)

words_no_punc = []

for w in words:
    if w.isalpha():
        words_no_punc.append(w.lower())

fdist = FreqDist(words_no_punc)
fdist.plot(10)
```

**Output**



- ✓ Notice that we still have many words that are not very useful in the analysis of our text file sample, such as “and,” “but,” “so,” and others.
- ✓ Next, we need to remove coordinating conjunctions.

## Data Science – NLP – Components in NLP

**Program Name** List of the stop words  
demo8.py

```
from nltk.corpus import stopwords

stopwords = stopwords.words('english')
print(stopwords)
```

**Output**

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", "your", "yours", "yourself", "yourselves", 'he', 'him', 'his', 'himself', 'she', "she's", "her", 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

## Data Science – NLP – Components in NLP

---

**Program Name** Removing the stop words  
demo9.py

```

import nltk
from nltk import word_tokenize
from nltk.probability import FreqDist
import matplotlib.pyplot as plt
from nltk.corpus import stopwords

data = open("story_input.txt")
text = data.read()

stopwords = stopwords.words('english')
words = word_tokenize(text)

words_no_punc = []

for w in words:
    if w.isalpha():
        words_no_punc.append(w.lower())

clean_words = []

for w in words_no_punc:
    if w not in stopwords:
        clean_words.append(w)

print(clean_words)
print("Length of clean words:", len(clean_words))

```

### Output

```

['upon', 'time', 'old', 'mother', 'pig', 'three', 'little', 'pigs', 'enough', 'food', 'feed', 'old', 'e
nough', 'sent', 'world', 'seek', 'fortunes', 'first', 'little', 'pig', 'lazy', 'want', 'work', 'built',
'house', 'straw', 'second', 'little', 'pig', 'worked', 'little', 'bit', 'harder', 'somewhat', 'lazy',
'builds', 'house', 'sticks', 'sang', 'danced', 'played', 'together', 'rest', 'day', 'third', 'little',
'pig', 'worked', 'hard', 'day', 'built', 'house', 'bricks', 'sturdy', 'house', 'complete', 'fine', 'fir
eplace', 'chimney', 'looked', 'like', 'could', 'withstand', 'strongest', 'winds']
Length of clean words 65

```

## Data Science – NLP – Components in NLP

---

**Program Name** Word frequency distribution  
demo10.py

```

import nltk
from nltk import word_tokenize
from nltk.probability import FreqDist
import matplotlib.pyplot as plt
from nltk.corpus import stopwords

data = open("story_input.txt")
text = data.read()

stopwords = stopwords.words('english')
words = word_tokenize(text)

words_no_punc = []

for w in words:
    if w.isalpha():
        words_no_punc.append(w.lower())

clean_words = []

for w in words_no_punc:
    if w not in stopwords:
        clean_words.append(w)

fdist = FreqDist(clean_words)
result = fdist.most_common(10)
print(result)

```

**Output**

```

[('little', 5),
 ('pig', 4),
 ('house', 4),
 ('old', 2),
 ('enough', 2),
 ('lazy', 2),
 ('built', 2),
 ('worked', 2),
 ('day', 2),
 ('upon', 1)]

```

## Data Science – NLP – Components in NLP

**Program Name** Plotting the useful words  
demo11.py

```
import nltk
from nltk import word_tokenize
from nltk.probability import FreqDist
import matplotlib.pyplot as plt
from nltk.corpus import stopwords

data = open("story_input.txt")
text = data.read()

stopwords = stopwords.words('english')
words = word_tokenize(text)

words_no_punc = []

for w in words:
    if w.isalpha():
        words_no_punc.append(w.lower())

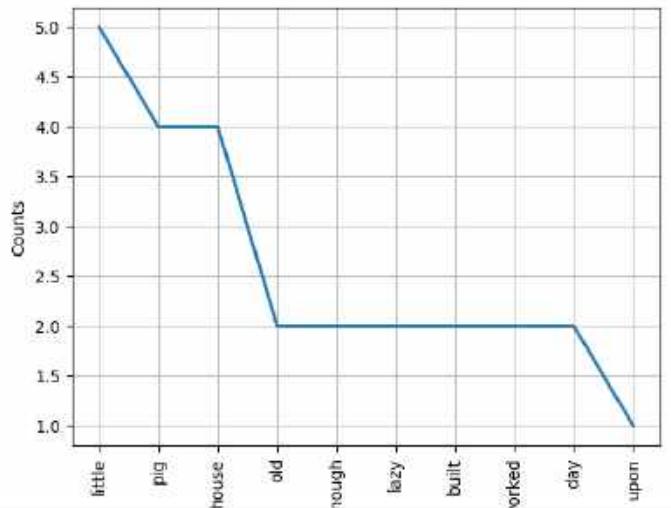
clean_words = []

for w in words_no_punc:
    if w not in stopwords:
        clean_words.append(w)

fdist = FreqDist(clean_words)
fdist.most_common(10)
fdist.plot(10)
```

## Data Science – NLP – Components in NLP

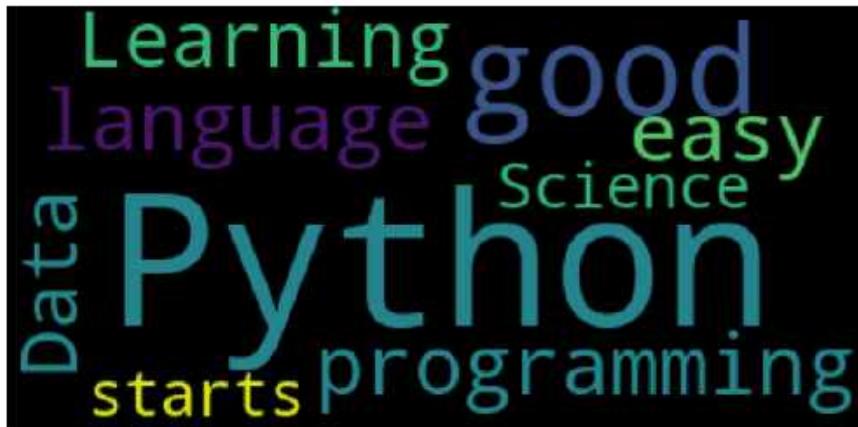
### Output



- ✓ As shown above, the final graph has many useful words that help us understand what our sample data is about, showing how essential it is to perform data cleaning on NLP.

#### 4. Word Cloud

- ✓ Word Cloud is a data visualization technique.
- ✓ In which words from a given text display on the main chart.
- ✓ In this technique, more frequent or essential words display in a larger and bolder font.
- ✓ Less frequent or essential words display in smaller or thinner fonts.
- ✓ It is a beneficial technique in NLP that gives us a glance at what text should be analysed.

<b>Program Name</b>	Wordcloud example demo12.py
<pre> import matplotlib.pyplot as plt from wordcloud import WordCloud  text = "Python is good programming language, Python is very easy, Learning Data Science starts from Python"  wordcloud = WordCloud().generate(text)  plt.figure(figsize = (12, 12)) plt.imshow(wordcloud)  plt.axis('off') plt.show() </pre>	
<b>Output</b>	

# Data Science – NLP – Bag of words, TF and IDF

---

## 5. NLP – Bag of words, TF and IDF

### Contents

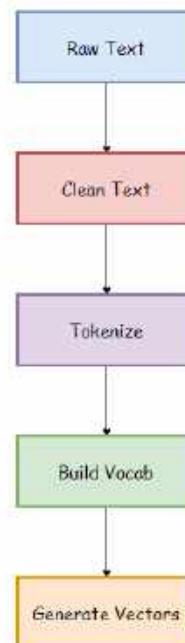
<b>1. NLP – Components in NLP Bag-of-Words .....</b>	<b>2</b>
1.1. Raw Text.....	3
1.2. Clean Text.....	3
1.3. Tokenize .....	3
1.4. Building Vocab .....	3
1.5. Generate Vocab .....	3
<b>2. Use case .....</b>	<b>4</b>
2.1. Creating a basic structure .....	4
2.2. Words with frequencies.....	5
2.3. Combining all the words .....	5
2.4. Final model.....	6
<b>3. Term Frequency-Inverse Document Frequency (TF-IDF) .....</b>	<b>8</b>

## Data Science – NLP – Bag of words, TF and IDF

### 5. NLP – Bag of words, TF and IDF

#### 1. NLP – Components in NLP Bag-of-Words

- ✓ It is a method of extracting essential features from raw text.
- ✓ So that we can use it for machine learning models.
- ✓ A bag of words model converts the raw text into words, and it also counts the frequency for the words in the text.



**1.1. Raw Text**

- ✓ This is the original text on which we want to perform analysis.

**1.2. Clean Text**

- ✓ Since our raw text contains some unnecessary data like punctuation marks and stopwords, so we need to clean up our text.

**1.3. Tokenize**

- ✓ Tokenization represents the sentence as a group of tokens or words.

**1.4. Building Vocab**

- ✓ It contains total words used in the text after removing unnecessary data.

**1.5. Generate Vocab**

- ✓ It contains the words along with their frequencies in the sentences.

## Data Science – NLP – Bag of words, TF and IDF

### 2. Use case

Let's take few sentences

- ✓ Jim and Pam travelled by bus.
- ✓ The train was late.
- ✓ The flight was full. Travelling by flight is expensive.

#### 2.1. Creating a basic structure

Sentence 1	Sentence2	Sentence 3
Jim	The	The
and	train	flight
Pam	was	was
travelled	late	full
by		Travelling
the		by
bus		flight
		is
		expensive

## Data Science – NLP – Bag of words, TF and IDF

### 2.2. Words with frequencies

Sentence1	Count	Sentence2	Count	Sentence3	Count
Jim	1	The	1	The	1
and	1	train	1	flight	2
Pam	1	was	1	was	1
travelled	1	late	1	full	1
by	1			Travelling	1
the	1			by	1
bus	1			is	1
				expensive	1

### 2.3. Combining all the words

Sentence	Frequency
and	1
bus	1
by	2
expensive	1
flight	2
full	1
is	1
jim	1
late	1
pam	1
the	3
train	1
travelled	1
travelling	1
was	1

## Data Science – NLP – Bag of words, TF and IDF

### 2.4. Final model

	and	bus	by	expensive	flight	full	is	jim	Late	pam	The	train	travelled	travelling	was
S-1	1	1	1	0	0	0	0	1	0	1	1	0	1	0	0
S-2	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1
S-3	0	0	1	1	2	1	1	0	0	0	1	0	0	1	1

Program Name

Bag of the words  
demo1.py

```
from sklearn.feature_extraction.text import CountVectorizer

sentences = ["Jim and Pam travelled by bus.",
"The train was late",
"The flight was full. Travelling by flight is expensive"]

cv = CountVectorizer()

B_O_W = cv.fit_transform(sentences).toarray()

print(B_O_W)
```

Output

```
[[1 1 1 0 0 0 0 1 0 1 0 0 1 0 0]
 [0 0 0 0 0 0 0 0 1 0 1 1 0 0 1]
 [0 0 1 1 2 1 1 0 0 0 1 0 0 1 1]]
```

### Applications

- ✓ This concept we can use in nlp applications
- ✓ Information retrieval from documents.
- ✓ Classifications of documents.

## Data Science – NLP – Bag of words, TF and IDF

---

### Limitations

- ✓ Semantic meaning:
  - It does not consider the semantic meaning of a word.
- ✓ Vector size:
  - For large documents, the vector size increase, which may result in higher computational time?
- ✓ Preprocessing:
  - In preprocessing, we need to perform data cleansing before using it.

### 3. Term Frequency-Inverse Document Frequency (TF-IDF)

- ✓ “Term Frequency – Inverse Document Frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection.”



Here's a sample of reviews about a particular horror movie:

- ✓ Review 1: This movie is very scary and long
- ✓ Review 2: This movie is not scary and is slow
- ✓ Review 3: This movie is spooky and good

## Data Science – NLP – Bag of words, TF and IDF

---

**TF: Term Frequency**

$$TF = \frac{\text{Frequency of the word in the sentence}}{\text{Total number of words in the sentence}}$$

Term	Review 1	Review 2	Review 3	TF (Review 1)	TF (Review 2)	TF (Review 3)
This	1	1	1	1/7	1/8	1/6
movie	1	1	1	1/7	1/8	1/6
is	1	2	1	1/7	1/4	1/6
very	1	0	0	1/7	0	0
scary	1	1	0	1/7	1/8	0
and	1	1	1	1/7	1/8	1/6
long	1	0	0	1/7	0	0
not	0	1	0	0	1/8	0
slow	0	1	0	0	1/8	0
spooky	0	0	1	0	0	1/6
good	0	0	1	0	0	1/6

## Data Science – NLP – Bag of words, TF and IDF

### Inverse Document Frequency (IDF)

- ✓ IDF is a measure of how important a term is in a sentence
- ✓ We need the IDF value because computing just the TF alone is not sufficient to understand the importance of words

$$idf_t = \log \frac{\text{number of documents}}{\text{number of documents with term 't'}}$$

### Example: Review 2

- ✓ IDF ('movie') =  $\log(3/3) = 0$
- ✓ IDF ('is') =  $\log(3/3) = 0$
- ✓ IDF ('not') =  $\log(3/1) = 0.48$
- ✓ IDF ('scary') =  $\log(3/2) = 0.18$
- ✓ IDF ('and') =  $\log(3/3) = 0$
- ✓ IDF ('slow') =  $\log(3/1) = 0.48$

### Calculating IDF

Term	Review 1	Review 2	Review 3	IDF
This	1	1	1	0.00
movie	1	1	1	0.00
is	1	2	1	0.00
very	1	0	0	0.48
scary	1	1	0	0.18
and	1	1	1	0.00
long	1	0	0	0.48
not	0	1	0	0.48
slow	0	1	0	0.48
spooky	0	0	1	0.48
good	0	0	1	0.48

- ✓ We can now compute the TF-IDF score for each word in the corpus.
- ✓ Words with a higher score are more important, and lower score are less important

## Data Science – NLP – Bag of words, TF and IDF

### Calculating final TF-IDF values:

✓  $\text{TF-IDF} = \text{TF} * \text{IDF}$

### Example

Review 2

$$\text{TF-IDF('this')} = \text{TF('this')} * \text{IDF('this')} = 1/8 * 0 = 0$$

Term	TF-IDF (Review 1)	TF-IDF (Review 2)	TF-IDF (Review 3)
This	0.000	0.000	0.000
movie	0.000	0.000	0.000
is	0.000	0.000	0.000
very	0.068	0.000	0.000
scary	0.025	0.022	0.000
and	0.000	0.000	0.000
long	0.068	0.000	0.000
not	0.000	0.060	0.000
slow	0.000	0.060	0.000
spooky	0.000	0.000	0.080
good	0.000	0.000	0.080

## Data Science – NLP – Twitter Sentiment Analysis - Textblob

---

### 6. NLP – Twitter Sentiment Analysis – Textblob

#### Contents

1. TextBlob.....	2
2. Installing textblob library.....	2
3. Simple TextBlob Sentiment Analysis Example .....	3
4. Using NLTK's Twitter Corpus .....	5

## Data Science – NLP – Twitter Sentiment Analysis - Textblob

---

### 6. NLP – Twitter Sentiment Analysis – Textblob

#### 1. TextBlob

- ✓ TextBlob provides an API that can perform different Natural Language Processing (NLP) tasks like,
  - Part-of-Speech Tagging
  - Noun Phrase Extraction
  - Sentiment Analysis
  - Classification (Naive Bayes, Decision Tree)
  - Language Translation and Detection
  - Spelling Correction etc.
- ✓ TextBlob is built upon Natural Language Toolkit (NLTK).
- ✓ Sentiment Analysis means analysing the sentiment of a given text or document and categorizing the text/document into a specific class or category (like positive and negative).
- ✓ Basically, the classification is done for two classes: positive and negative.
- ✓ However, we can add more classes like neutral, highly positive, highly negative, etc.

#### 2. Installing textblob library

- ✓ pip install -U textblob

## Data Science – NLP – Twitter Sentiment Analysis - Textblob

---

### 3. Simple TextBlob Sentiment Analysis Example

- ✓ We can apply textblob on any text to do Sentiment Analysis
- ✓ The sentiment property gives the sentiment scores to the given text.
- ✓ There are two scores given: Polarity and Subjectivity.
  - The **polarity** score is a float within the range [-1.0, 1.0] where negative value indicates negative text and positive value indicates that the given text is positive.
  - The **subjectivity** is a float within the range [0.0, 1.0] where 0.0 is very objective and 1.0 is very subjective.

<b>Program Name</b>	Sentiment analysis demo1.py
	<pre>from textblob import TextBlob  text = TextBlob("It was a wonderful movie. I liked it very much.")  print (text.sentiment) print ('polarity: {}'.format(text.sentiment.polarity)) print ('subjectivity: {}'.format(text.sentiment.subjectivity))</pre>
<b>Output</b>	<pre>Sentiment(polarity=0.62, subjectivity=0.6866666666666666) polarity: 0.62 subjectivity: 0.6866666666666666</pre>

## Data Science – NLP – Twitter Sentiment Analysis - Textblob

---

**Program Name**

Sentiment analysis

demo2.py

```
from textblob import TextBlob

text = TextBlob("I liked the acting of the lead actor but I didn't like the movie
overall.")

print (text.sentiment)
print ('polarity: {}'.format(text.sentiment.polarity))
print ('subjectivity: {}'.format(text.sentiment.subjectivity))
```

**Output**

```
Sentiment(polarity=0.1999999999999998, subjectivity=0.2666666666666666)
polarity: 0.1999999999999998
subjectivity: 0.2666666666666666
```

**Program Name**

Sentiment analysis

demo3.py

```
from textblob import TextBlob
```

```
text = TextBlob("I liked the acting of the lead actor and I liked the movie
overall.")
```

```
print (text.sentiment)
print ('polarity: {}'.format(text.sentiment.polarity))
print ('subjectivity: {}'.format(text.sentiment.subjectivity))
```

**Output**

```
Sentiment(polarity=0.3, subjectivity=0.4)
polarity: 0.3
subjectivity: 0.4
```

## Data Science – NLP – Twitter Sentiment Analysis - Textblob

---

### 4. Using NLTK's Twitter Corpus

- ✓ We use the twitter\_samples corpus to train the TextBlob's NaiveBayesClassifier.
- ✓ Using the twitter\_samples corpus, we create a train set and test set containing a certain amount of positive and negative tweets.
- ✓ And, then we test the accuracy of the trained classifier.

<b>Program Name</b>	Getting twitter sample datasets <b>demo4.py</b>  <pre>from nltk.corpus import twitter_samples import nltk nltk.download('twitter_samples')  print(twitter_samples.fileids())</pre>
<b>Output</b>	<pre>['negative_tweets.json', 'positive_tweets.json', 'tweets.20150430-223406.json']</pre>

<b>Program Name</b>	Getting twitter sample datasets and checking length <b>demo5.py</b>  <pre>from nltk.corpus import twitter_samples  pos_tweets = twitter_samples.strings('positive_tweets.json') print(len(pos_tweets))  neg_tweets = twitter_samples.strings('negative_tweets.json') print(len(neg_tweets))</pre>
<b>Output</b>	<pre>5000 5000</pre>

## Data Science – NLP – Twitter Sentiment Analysis - Textblob

**Program Name** Getting twitter sample datasets and checking length  
demo6.py

```
from nltk.corpus import twitter_samples

all_tweets = twitter_samples.strings('tweets.20150430-223406.json')
print (len(all_tweets))
```

**Output**

20000

**Program Name** Getting positive tweets  
demo7.py

```
from nltk.corpus import twitter_samples

pos_tweets = twitter_samples.strings('positive_tweets.json')

pos_tweets_set = []

for tweet in pos_tweets:
    pos_tweets_set.append((tweet, 'pos'))

for i in pos_tweets_set:
    print(i)
```

**Output**

```
('#FollowFriday @France_Interactive @PKuchly57 @Milipol_Paris for being top engaged members in my community this week :)', 'pos')
('@Lamb2ja Hey James! How odd :/ Please call our Contact Centre on 02392441234 and we will be able to assist you :) Many thanks!', 'pos')
('@DespiteOfficial we had a listen last night :) As You Bleed is an amazing track. When are you in Scotland?!?', 'pos')
('@97sides CONGRATS :)', 'pos')
('yeaaaah yippyyy!!! my accnt verified rqst has succeed got a blue tick mark on my fb profile :) in 15 days', 'pos')
('@ShaktisBanter @PallaviRuhail This one is irresistible :) \n#FlipkartFashionFriday http://t.co/Ebz0L2vENM', 'pos')
('We don't like to keep our lovely customers waiting for long! We hope you enjoy! Happy Friday! - LWWF :) https://t.co/smYriipxI', 'pos')
('@Impatientraider On second thought, there's just not enough time for a DD :) But new shorts entering system. Sheep must be buying.', 'pos')
```

## Data Science – NLP – Twitter Sentiment Analysis - Textblob

**Program Name** Getting negative tweets  
demo8.py

```
from nltk.corpus import twitter_samples

neg_tweets = twitter_samples.strings('negative_tweets.json')

neg_tweets_set = []

for tweet in neg_tweets:
    neg_tweets_set.append((tweet, 'neg'))

for i in neg_tweets_set:
    print(i)
```

**Output**

```
('hopeless for tmr :(', 'neg')
('Everything in the kids section of IKEA is so cute. Shame I'm nearly 19 in 2 months :(', 'neg')
('@Hegelbon That heart sliding into the waste basket. :(', 'neg')
('@"ketchBurning: I hate Japanese call him "bani" :( :("n\nMe too", 'neg')
('Dang starting next week I have "work" :(', 'neg')
('oh god, my babies' faces :( https://t.co/9fcwGvaki0", 'neg')
('@RileyMcDonough make me smile :(((' , 'neg')
('@eggstar @stuartthull work neighbour on motors. Asked why and he said hates the updates on search :( http://t.co/xvmTukwln', 'neg')
('why?:(@ahuodyy: sialan:( https://t.co/Hv1i0xcrL2", 'neg')
('Athabasca glacier was there in #1948 :-( #athabasca #glacier #jasper #jaspernationalpark #alberta #explorealberta #. http://t.co/dZdqmf7Cz', 'neg')
('I have a really good & idea but I'm never going to meet them :((( , 'neg')
('@Rampageinthebox mare ivan :(', 'neg')
```

## Data Science – NLP – Twitter Sentiment Analysis - Textblob

**Program Name** Splitting Training and testing datasets  
demo9.py

```
from nltk.corpus import twitter_samples

pos_tweets = twitter_samples.strings('positive_tweets.json')
neg_tweets = twitter_samples.strings('negative_tweets.json')

pos_tweets_set = []

for tweet in pos_tweets:
    pos_tweets_set.append((tweet, 'pos'))

neg_tweets_set = []

for tweet in neg_tweets:
    neg_tweets_set.append((tweet, 'neg'))

from random import shuffle

shuffle(pos_tweets_set)
shuffle(neg_tweets_set)

test_set = pos_tweets_set[:300] + neg_tweets_set[:300]
train_set = pos_tweets_set[300:600] + neg_tweets_set[300:600]

print("Training and testing datasets")
```

**Output**  
Training and testing datasets

## Data Science – NLP – Twitter Sentiment Analysis - Textblob

**Program Name** Creating model and training the model  
demo10.py

```
from nltk.corpus import twitter_samples

pos_tweets = twitter_samples.strings('positive_tweets.json')
neg_tweets = twitter_samples.strings('negative_tweets.json')

pos_tweets_set = []

for tweet in pos_tweets:
    pos_tweets_set.append((tweet, 'pos'))

neg_tweets_set = []

for tweet in neg_tweets:
    neg_tweets_set.append((tweet, 'neg'))

from random import shuffle

shuffle(pos_tweets_set)
shuffle(neg_tweets_set)

test_set = pos_tweets_set[:300] + neg_tweets_set[:300]
train_set = pos_tweets_set[300:600] + neg_tweets_set[300:600]

from textblob.classifiers import NaiveBayesClassifier
classifier = NaiveBayesClassifier(train_set)

print("Model got trained")
```

**Output**  
Model got trained

## Data Science – NLP – Twitter Sentiment Analysis - Textblob

Program  
Name

Testing model accuracy  
demo11.py

```
from nltk.corpus import twitter_samples

pos_tweets = twitter_samples.strings('positive_tweets.json')
neg_tweets = twitter_samples.strings('negative_tweets.json')

pos_tweets_set = []

for tweet in pos_tweets:
    pos_tweets_set.append((tweet, 'pos'))

neg_tweets_set = []

for tweet in neg_tweets:
    neg_tweets_set.append((tweet, 'neg'))

from random import shuffle

shuffle(pos_tweets_set)
shuffle(neg_tweets_set)

test_set = pos_tweets_set[:300] + neg_tweets_set[:300]
train_set = pos_tweets_set[300:600] + neg_tweets_set[300:600]

from textblob.classifiers import NaiveBayesClassifier
classifier = NaiveBayesClassifier(train_set)

accuracy = classifier.accuracy(test_set)
print(accuracy)
```

Output

0.7366666666666667

## Data Science – NLP – Twitter Sentiment Analysis - Textblob

**Program Name** Testing the model  
demo12.py

```
from nltk.corpus import twitter_samples

pos_tweets = twitter_samples.strings('positive_tweets.json')
neg_tweets = twitter_samples.strings('negative_tweets.json')

pos_tweets_set = []

for tweet in pos_tweets:
    pos_tweets_set.append((tweet, 'pos'))

neg_tweets_set = []

for tweet in neg_tweets:
    neg_tweets_set.append((tweet, 'neg'))

from random import shuffle

shuffle(pos_tweets_set)
shuffle(neg_tweets_set)

test_set = pos_tweets_set[:300] + neg_tweets_set[:300]
train_set = pos_tweets_set[300:600] + neg_tweets_set[300:600]

from textblob.classifiers import NaiveBayesClassifier
classifier = NaiveBayesClassifier(train_set)

text = "It was a wonderful movie. I liked it very much."
print(classifier.classify(text))
```

**Output**

0.735  
pos

## Data Science – NLP – Twitter Sentiment Analysis - Textblob

---

<b>Program Name</b>	Testing the model demo13.py
<pre> from nltk.corpus import twitter_samples  pos_tweets = twitter_samples.strings('positive_tweets.json') neg_tweets = twitter_samples.strings('negative_tweets.json')  pos_tweets_set = []  for tweet in pos_tweets:     pos_tweets_set.append((tweet, 'pos'))  neg_tweets_set = []  for tweet in neg_tweets:     neg_tweets_set.append((tweet, 'neg'))  from random import shuffle  shuffle(pos_tweets_set) shuffle(neg_tweets_set)  test_set = pos_tweets_set[:300] + neg_tweets_set[:300] train_set = pos_tweets_set[300:600] + neg_tweets_set[300:600]  from textblob.classifiers import NaiveBayesClassifier classifier = NaiveBayesClassifier(train_set)  text = "I don't like movies having happy ending." print (classifier.classify(text)) </pre>	
<b>Output</b>	0.7266666666666666 neg

## Data Science – NLP – Twitter Sentiment Analysis - Textblob

**Program Name**

Testing the model  
demo14.py

```
from nltk.corpus import twitter_samples
from textblob import TextBlob

pos_tweets = twitter_samples.strings('positive_tweets.json')
neg_tweets = twitter_samples.strings('negative_tweets.json')

pos_tweets_set = []

for tweet in pos_tweets:
    pos_tweets_set.append((tweet, 'pos'))

neg_tweets_set = []

for tweet in neg_tweets:
    neg_tweets_set.append((tweet, 'neg'))

from random import shuffle

shuffle(pos_tweets_set)
shuffle(neg_tweets_set)

test_set = pos_tweets_set[:300] + neg_tweets_set[:300]
train_set = pos_tweets_set[300:600] + neg_tweets_set[300:600]

from textblob.classifiers import NaiveBayesClassifier
classifier = NaiveBayesClassifier(train_set)

text = "It was a wonderful movie. I liked it very much."
print(classifier.classify(text))

blob = TextBlob(text, classifier=classifier)

print(blob)
print(blob.classify())
```

**Output**

```
0.695
pos
It was a wonderful movie. I liked it very much.
pos
```

# Data Science – NLP – Spacy library

---

## 7. NLP – Spacy library

### Contents

<b>1. Introduction</b> .....	2
<b>2. Install and Load Main Python Libraries for NLP</b> .....	3
2.1. Installation .....	3
<b>3. Text Pre-processing</b> .....	4
<b>4. What is Tokenization?</b> .....	5
<b>5. Stop words in spacy library</b> .....	10
<b>6. Stemming</b> .....	12
<b>7. Lemmatization</b> .....	13
<b>8. Lemmatization using spacy</b> .....	13
<b>9. Word Frequency Analysis</b> .....	14
<b>10. Counter predefined class</b> .....	15
<b>11. Part of Speech Tagging</b> .....	21
<b>12. Parts of Speech</b> .....	21
<b>13. Pos by using spacy</b> .....	22
<b>14. Named Entity Recognition</b> .....	27
<b>15. What is NER?</b> .....	27
<b>16. NER by using nltk and spacy</b> .....	27
<b>17. NER with spacy</b> .....	28
<b>18. The few common labels are:</b> .....	28

## Data Science – NLP – Spacy library

---

### 6. NLP – Spacy library

#### 1. Introduction

- ✓ There are mainly 3 types of data we have
  - Structured data
  - Semi structured data
  - Unstructured data
- ✓ Today more than 80% of the data available as Unstructured Data.
- ✓ Unstructured data is a data which cannot be represented in tabular format
  - Texts,
  - Videos,
  - Images
- ✓ We can use Natural Language processing to process and interpret the unstructured data
- ✓ NLP applications,
  - Sentiment analysis
  - Speech recognition
  - Text Classification
  - Machine Translation
  - Semantic Search
  - News/article Summarization
  - Answering Questions
- ✓ Live examples,
  - Google Assistant.
  - Amazon Echo.
  - ChatBot,
  - Language Translations,
  - Article summarization and so on.

## Data Science – NLP – Spacy library

---

### 2. Install and Load Main Python Libraries for NLP

- ✓ We do have different python libraries to work with NLP.
  - Nltk
  - Spacy
  - Genism
  - transformers

#### 2.1. Installation

- ✓ Open a command prompt and run below commands

- pip install nltk
  - pip install spacy
  - pip install gensim
  - pip install transformers

#### Note: Run below command

- ✓ python -m spacy download en\_core\_web\_sm

## Data Science – NLP – Spacy library

### 3. Text Pre-processing

- ✓ The raw text data also called as text corpus, it has a lot of noise.
- ✓ Raw text having punctuation, special symbols, stop words & etc
- ✓ From the raw data we need to process the text by using nlp techniques.
- ✓ A text can be converted into nlp object of spacy.

Program

spacy example

Name

demo1.py

```
import spacy
```

```
nlp = spacy.load('en_core_web_sm')
```

```
raw_text = 'NLP is a very powerful tool'
```

```
text_doc = nlp(raw_text)
```

```
print(text_doc)
```

```
print(type(text_doc))
```

Output

```
NLP is a very powerful tool
<class 'spacy.tokens.doc.Doc'>
```

## Data Science – NLP – Spacy library

### 4. What is Tokenization?

- ✓ The process of extracting tokens from a text file/document is referred as tokenization.

**Program Name** Tokenization using spacy  
demo2.py

```
import spacy

nlp = spacy.load('en_core_web_sm')

raw_text = 'NLP is a very powerful tool'

text_doc = nlp(raw_text)

print(text_doc)
print()
for word in text_doc:
    print(word.text)
```

**Output**

```
NLP is a very powerful tool

NLP
is
a
very
powerful
tool
```

## Data Science – NLP – Spacy library

**Program Name** Checking every token either it is alphabet or not by using spacy demo3.py

```
import spacy

nlp = spacy.load('en_core_web_sm')

mixed_text = 'My salary is $1000 dollars'
mixed_text_doc = nlp(mixed_text)

for word in mixed_text_doc:
    print(word.text.ljust(10), word.is_alpha)
```

**Output**

```
My           True
salary      True
is          True
$            False
1000        False
dollars     True
```

## Data Science – NLP – Spacy library

**Program Name** Checking every token alpha, stop word and punctuation  
demo4.py

```
import spacy

nlp = spacy.load('en_core_web_sm')

text='My salary is $1000 dollars'
text_doc=nlp(text)

print("Text","\t\t", "Alpha", "Stop", "Punct")

for word in text_doc:
    print(word.text.ljust(15), ':', word.is_alpha, word.is_stop,
          word.is_punct)
```

### Output

Text	Alpha	Stop	Punct
My	: True	True	False
salary	: True	False	False
is	: True	True	False
\$	: False	False	False
1000	: False	False	False
dollars	: True	False	False

## Data Science – NLP – Spacy library

---

**Program Name**

Checking tokens and count of tokens by using spacy  
demo5.py

```
import spacy

nlp = spacy.load('en_core_web_sm')

raw_text= 'Amazon Alexa, also known simply as Alexa, is a virtual assistant AI technology developed by Amazon, first used in the Amazon Echo smart speakers developed by Amazon Lab126. It is capable of voice interaction, music playback, making to-do lists, setting alarms, streaming podcasts, playing audio books, and providing weather, traffic, sports, and other real-time information, such as news. Alexa can also control several smart devices using itself as a home automation system. Users are able to extend the Alexa capabilities by installing "skills" additional functionality developed by third-party vendors, in other settings more commonly called apps such as weather programs and audio features. Most devices with Alexa allow users to activate the device using a wake-word (such as Alexa or Amazon); other devices (such as the Amazon mobile app on iOS or Android and Amazon Dash Wand) require the user to push a button to activate Alexa listening mode, although, some phones also allow a user to say a command, such as "Alexa" or "Alexa wake". Currently, interaction and communication with Alexa are available only in English, German, French, Italian, Spanish, Portuguese, Japanese, and Hindi. '

text_doc = nlp(raw_text)

token_count = 0

for word in text_doc:
    print(word.text)
    token_count = token_count + 1

print("Token count is:", token_count)
```

**Output**

```
Amazon
Alexa
,
also
known
simply
as
```

## Data Science – NLP – Spacy library

Alexa

,

is

a

.....

Token count is: 235

### Note

- ✓ So, we have a lot of tokens here. The stop words like 'it', 'was' 'that', 'to'..., so on.
- ✓ These words will not give us much information, especially for models.
- ✓ Punctuations also will not provide much info
  - While dealing with large text files, the stop words and punctuations will be repeated at high levels, misguiding us to think they are important.
- ✓ So, it is necessary to filter out the stop words.

## Data Science – NLP – Spacy library

### 5. Stop words in spacy library

- ✓ Spacy has an inbuilt list of stop words.

**Program Name** Checking stop words by using spacy demo6.py

```
import spacy

nlp = spacy.load('en_core_web_sm')

stopwords = spacy.lang.en.stop_words.STOP_WORDS

list_stopwords = list(stopwords)

print(list_stopwords)
```

#### Output

```
[ 're', 'through', "'ll", 'besides', 'in', 'per', 'along', 'rather', 'mostly', 'ourselves', 'around', 'further', 'always', 'same', 'among', 'that', 'we', 'yet', "n't", 'hereupon', 'own', 'have', 'must', 'the before', 'except', 'into', 'some', 'too', 'her', 'could', 'are', 'but', 'hereby', "'s", "'ve", 'least', 'indeed', 'over', 'using', 'do', 'whenever', 'nowhere', 'therein', 'call', 'wherein', 'your', 'meanwhile', 'due', "'s", 'go', 'part', 'no', 'themselves', 'onto', 'moreover', 'almost', 'without', 'am', 'side', 'a', 'make', 'anyway', 'just', 'then', 'while', 'been', 'every', 'under', 'will', 'seem', 'none', 'latterly', 'about', 'please', 'say', 'beside', 'see', 'forty', 'unless', 'ca', 'whereupon', 'everyone', 'become', 'had', 'why', 'though', 'eleven', 'take', 'be', 'after', 'via', 'than', 'these', 'where', 'a nyhow', 'each', 'first', 'made', 'many', 'most', 'has', 'name', 'nobody', 'amongst', 'one', 'thru', 'us ed', 'whence', 'very', 'everything', 'cannot', 'yourselves', 'somewhere', 'would', 'of', 'four', 'at', 'elsewhere', 'may', 'last', 'before', 'much', 'next', 'hence', 'throughout', 'for', 'twelve', 'you', 's till', 'either', 'm', 'when', 'top', 'were', 'wherever', 'with', 'alone', 'eight', 'becomes', 'front', 'an', 'again', 'fifty', 'whether', "'ll", 'latter', 'against', 'became', 'and', 'on', 'anyone', 'anything', 'sometime', 'thereupon', 'i', 'various', 'up', 'thence', "n't", 'here', 'less', 'seeming', "'s", 'itself', 'out', 'sometimes', 'thereby', 'third', 'to', 'quite', 'together', 've', 'yours', 'often', 'nor', 'move', 'twenty', 'my', 'also', 'all', 'keep', 'doing', 'however', 'hundred', 'back', 'since', 'o thers', 'something', 'those', 'well', 'because', 'whereby', 'hereafter', 'himself', 'me', 'full', 'hers elf', 'myself', 'whatever', 'seemed', 'even', 'within', 'behind', 'n't', 'it', 'formerly', 'perhaps', 'down', 'becoming', 'them', 'upon', 'they', 'afterwards', 'ten', 'whoever', 'us', "'ve", 'she', 'otherwi se', 'beyond', 'ever', 'which', 'once', 'was', 'so', 'whom', 'm', 'bottom', 'neither', 'any', 'anywhere', 'else', 'his', 'now', 'several', 're', 'few', 'd', 'above', 'there', 'by', 'off', 'is', 'serious', 'if', 'get', 'sixty', 'during', 'seems', 'everywhere', 'regarding', 'namely', 're', 'give', 'd', 'o ther', 'd', 'him', 'thereafter', 'towards', 'between', 'or', 'although', 'does', 'below', 'noone', 'be
```

## Data Science – NLP – Spacy library

Program Name

Removing **stop words** and **punctuations** from text using spacy  
demo7.py

```
import spacy

nlp = spacy.load('en_core_web_sm')

raw_text = "Hello good!!! morning how are you, today climate is very cool"

text_doc = nlp(raw_text)

final = [
    token
    for token in text_doc
    if not token.is_stop and not token.is_punct
]

print("Actual text:", raw_text)

for token in final:
    print(token)
```

Output

```
Actual text: Hello good!!! morning how are you, today climate is very cool
Hello
good
morning
today
climate
cool
```

## 6. Stemming

- ✓ General words are,
  - ‘calculator’,
  - ‘calculating’,
  - ‘Calculation’.
- ✓ We know that these words are not very distinct from each other.
- ✓ It can be achieved through stemming.
- ✓ Stemming means reducing a word to its ‘root form’.
- ✓ We can implement this by using,
  - PorterStemmer predefined class

<b>Program Name</b>	Stemming example demo8.py
<pre>from nltk.stem import PorterStemmer  ps = PorterStemmer()  words = ['dance ','dances', 'dancing ','danced']  for word in words:     print(word,'-----&gt;', ps.stem(word))</pre>	

### Output

```
dance -----> dance
dances -----> danc
dancing -----> dancing
danced -----> danc
```

### Note

- ✓ You can observe that some words were reduced to the base stems ‘danc’
- ✓ But, the word ‘danc’ is **not semantically correct**.
- ✓ Stemming may give us root words that are not present in the dictionary.
- ✓ How to overcome this?
  - That’s where Lemmatization comes to rescue

## 7. Lemmatization

- ✓ It is similar to stemming, except that the root word is correct and always meaningful.
- ✓ There are different ways to perform lemmatization.
- ✓ We will learn lemmatization by using nltk and spacy in below examples.

## 8. Lemmatization using spacy

- ✓ In spacy, the token object has an attribute `.lemma_` which allows you to access the lemmatized version of that token.

<b>Program Name</b>	Lemmatization using spacy demo9.py
	<pre>import spacy nlp = spacy.load('en_core_web_sm')  text = 'dance dances dancing danced' text_doc = nlp(text)  for token in text_doc:     print(token.text,'-----', token.lemma_)</pre>
<b>Output</b>	<pre>dance ----- dance dances ----- dance dancing ----- dance danced ----- dance</pre>

### Note

- ✓ Here, all words are reduced to ‘dance’ which is meaningful and just as required.
- ✓ It is highly preferred over stemming.

### 9. Word Frequency Analysis

- ✓ Once we removed stop words and applied lemmatization then the next important step is, we need analyse for further information about the text data.
- ✓ If any specific word repeated more times then we need to focus on that well.
- ✓ So, the words which occur more frequently those are very key concepts
- ✓ So, we need to store all tokens with their frequencies separately to analyse well

## Data Science – NLP – Spacy library

---

### 10. Counter predefined class

- ✓ We can use Counter to get the frequency of each token.
- ✓ If you provide a list to the Counter it returns a dictionary of all elements with their frequency as values.

<b>Program Name</b>	Counter example demo10.py
<pre> import collections from collections import Counter  import spacy nlp = spacy.load('en_core_web_sm')  data = 'It is my birthday today. I could not have a birthday party. I felt sad' data_doc = nlp(data)  list_of_tokens = [token.text for token in data_doc if not token.is_stop and not token.is_punct]  token_frequency = Counter(list_of_tokens) print(token_frequency) </pre>	
<b>Output</b>	
<pre>Counter({'birthday': 2, 'today': 1, 'party': 1, 'felt': 1, 'sad': 1})</pre>	

### Note

- ✓ From above output, the word ‘birthday’ is most common.
- ✓ So, it gives us an idea that the text is about a birthday.
- ✓ Let us try with another example with more larger data
- ✓ For example,
  - If we have a text data about a particular place, and you want to know the important factors.

<b>Program Name</b>	Counter example on large text demo11.py
	<pre>import collections from collections import Counter  import spacy nlp = spacy.load('en_core_web_sm')  longtext = 'Gangtok is a city, municipality, the capital and the largest town of the Indian state of Sikkim. It is also the headquarters of the East Sikkim district. Gangtok is in the eastern Himalayan range, at an elevation of 1,650. The towns population of 100000 are from different ethnicities such as Bhutia, Lepchas and Indian Gorkhas. Within the higher peaks of the Himalaya and with a year-round mild temperate climate, Gangtok is at the centre of Sikkims tourism industry. Gangtok rose to prominence as a popular Buddhist pilgrimage site after the construction of the Enchey Monastery in 1840. In 1894, the ruling Sikkimese Chogyal, Thutob Namgyal, transferred the capital to Gangtok. In the early 20th century, Gangtok became a major stopover on the trade route between Lhasa in Tibet and cities such as Kolkata (then Calcutta) in British India. After India won its independence from Britain in 1947, Sikkim chose to remain an independent monarchy, with Gangtok as its capital. In 1975, after the integration with the union of India, Gangtok was made Indias 22nd state capital. Like the rest of Sikkim, not much is known about the early history of Gangtok. The earliest records date from the construction of the hermitic Gangtok monastery in 1716.[7] Gangtok remained a small hamlet until the construction of the Enchey Monastery in 1840 made it a pilgrimage center. It became the capital of what was left of Sikkim after an English conquest in the mid-19th century in response to a hostage crisis. After the defeat of the Tibetans by the British, Gangtok became a major stopover in the trade between Tibet and British India at the end of the 19th century. Most of the roads and the telegraph in the area were built during this time. In 1894, Thutob Namgyal, the Sikkimese monarch under British rule, shifted the capital from Tumlong to Gangtok, increasing the citys importance. A new grand palace along with other state buildings was built in the new capital. Following India independence in 1947, Sikkim became a nation-state with Gangtok as its capital. Sikkim came under the suzerainty of India, with the condition that it would retain its independence, by the treaty signed between the Chogyal and the then Indian Prime Minister Jawaharlal Nehru.[9] This pact gave the Indians control of external affairs on behalf of Sikkimese. Trade between India and Tibet continued to flourish through the Nathula and Jelepla passes, offshoots of the ancient Silk Road near Gangtok. These border passes were sealed after the Sino-Indian War in 1962, which deprived Gangtok of its trading business.[10] The Nathula pass'</pre>

## Data Science – NLP – Spacy library

was finally opened for limited trade in 2006, fuelling hopes of economic boomIn 1975, after years of political uncertainty and struggle, including riots, the monarchy was abrogated and Sikkim became India twenty-second state, with Gangtok as its capital after a referendum. Gangtok has witnessed annual landslides, resulting in loss of life and damage to property. The largest disaster occurred in June 1997, when 38 were killed and hundreds of buildings were destroyed'

```
long_text = nlp(longtext)

list_of_tokens = [token.text for token in long_text if not token.is_stop and
not token.is_punct]

token_frequency = Counter(list_of_tokens)
print(token_frequency)
```

### Output

```
Counter({'Gangtok': 18, 'capital': 9, 'Sikkim': 8, 'India': 8, 'state': 5, 'Indian': 4,
'British': 4, 'construction': 3, 'Sikkimese': 3, 'century': 3, 'trade': 3, 'Tibet': 3,
'independence': 3, 'largest': 2, 'pilgrimage': 2, 'Enchey': 2, 'Monastery': 2,
'1840': 2, '1894': 2, 'Chogyal': 2, 'Thutob': 2, 'Namgyal': 2, 'early': 2, 'major': 2,
'stopover': 2, '1947': 2, 'monarchy': 2, '1975': 2, 'built': 2, 'new': 2, 'buildings': 2,
'Nathula': 2, 'passes': 2, 'city': 1, 'municipality': 1, 'town': 1, 'headquarters': 1,
'East': 1, 'district': 1, 'eastern': 1, 'Himalayan': 1, 'range': 1, 'elevation': 1,
'1,650': 1, 'towns': 1, 'population': 1, '100000': 1, 'different': 1, 'ethnicities': 1,
'Bhutia': 1, 'Lepchas': 1, 'Gorkhas': 1, 'higher': 1, 'peaks': 1, 'Himalaya': 1,
'year': 1, 'round': 1, 'mild': 1, 'temperate': 1, 'climate': 1, 'centre': 1, 'Sikkims': 1,
'tourism': 1, 'industry': 1, 'rose': 1, 'prominence': 1, 'popular': 1, 'Buddhist': 1,
'site': 1, 'ruling': 1, 'transferred': 1, '20th': 1, 'route': 1, 'Lhasa': 1, 'cities': 1,
'Kolkata': 1, 'Calcutta': 1, 'won': 1, 'Britain': 1, 'chose': 1, 'remain': 1,
'independent': 1, 'integration': 1, 'union': 1, 'Indias': 1, '22nd': 1, 'Like': 1,
'rest': 1, 'known': 1, 'history': 1, 'earliest': 1, 'records': 1, 'date': 1, 'hermitic': 1,
'monastery': 1, '1716.[7': 1, 'remained': 1, 'small': 1, 'hamlet': 1, 'center': 1,
'left': 1, 'English': 1, 'conquest': 1, 'mid-19th': 1, 'response': 1, 'hostage': 1,
'crisis': 1, 'defeat': 1, 'Tibetans': 1, 'end': 1, '19th': 1, 'roads': 1, 'telegraph': 1,
'area': 1, 'time': 1, 'monarch': 1, 'rule': 1, 'shifted': 1, 'Tumlong': 1, 'increasing': 1,
'citys': 1, 'importance': 1, 'grand': 1, 'palace': 1, 'Following': 1, 'nation': 1,
'came': 1, 'suzerainty': 1, 'condition': 1, 'retain': 1, 'treaty': 1, 'signed': 1,
'Prime': 1, 'Minister': 1, 'Jawaharlal': 1, 'Nehru.[9': 1, 'pact': 1, 'gave': 1,
'Indians': 1, 'control': 1, 'external': 1, 'affairs': 1, 'behalf': 1, 'Trade': 1,
'continued': 1, 'flourish': 1, 'Jelepla': 1, 'offshoots': 1, 'ancient': 1, 'Silk': 1,
'Road': 1, 'near': 1, 'border': 1, 'sealed': 1, 'Sino': 1, 'War': 1, '1962': 1,
'deprived': 1, 'trading': 1, 'business.[10': 1, 'pass': 1, 'finally': 1, 'opened': 1,
'limited': 1, '2006': 1, 'fuelling': 1, 'hopes': 1, 'economic': 1, 'boomIn': 1,
'years': 1, 'political': 1, 'uncertainty': 1, 'struggle': 1, 'including': 1, 'riots': 1,
```

## Data Science – NLP – Spacy library

```
'abrogated': 1, 'second': 1, 'referendum': 1, 'witnessed': 1, 'annual': 1,  
'landslides': 1, 'resulting': 1, 'loss': 1, 'life': 1, 'damage': 1, 'property': 1,  
'disaster': 1, 'occurred': 1, 'June': 1, '1997': 1, '38': 1, 'killed': 1, 'hundreds': 1,  
'destroyed': 1})
```

### Note

- ✓ As you can see, as the length or size of text data increases, it is difficult to analyse frequency of all tokens.
- ✓ So, you can print the n most common tokens using `most_common` function of Counter.
- ✓ see the below example on how to print the 6 most frequent words of the text

## Data Science – NLP – Spacy library

---

<b>Program Name</b>	Most common words from text demo12.py
	<pre> import collections from collections import Counter  import spacy nlp = spacy.load('en_core_web_sm')  longtext = 'Gangtok is a city, municipality, the capital and the largest town of the Indian state of Sikkim. It is also the headquarters of the East Sikkim district. Gangtok is in the eastern Himalayan range, at an elevation of 1,650. The towns population of 100000 are from different ethnicities such as Bhutia, Lepchas and Indian Gorkhas. Within the higher peaks of the Himalaya and with a year-round mild temperate climate, Gangtok is at the centre of Sikkims tourism industry. Gangtok rose to prominence as a popular Buddhist pilgrimage site after the construction of the Enchey Monastery in 1840. In 1894, the ruling Sikkimese Chogyal, Thutob Namgyal, transferred the capital to Gangtok. In the early 20th century, Gangtok became a major stopover on the trade route between Lhasa in Tibet and cities such as Kolkata (then Calcutta) in British India. After India won its independence from Britain in 1947, Sikkim chose to remain an independent monarchy, with Gangtok as its capital. In 1975, after the integration with the union of India, Gangtok was made Indias 22nd state capital. Like the rest of Sikkim, not much is known about the early history of Gangtok. The earliest records date from the construction of the hermitic Gangtok monastery in 1716.[7] Gangtok remained a small hamlet until the construction of the Enchey Monastery in 1840 made it a pilgrimage center. It became the capital of what was left of Sikkim after an English conquest in the mid-19th century in response to a hostage crisis. After the defeat of the Tibetans by the British, Gangtok became a major stopover in the trade between Tibet and British India at the end of the 19th century. Most of the roads and the telegraph in the area were built during this time. In 1894, Thutob Namgyal, the Sikkimese monarch under British rule, shifted the capital from Tumlong to Gangtok, increasing the citys importance. A new grand palace along with other state buildings was built in the new capital. Following India independence in 1947, Sikkim became a nation-state with Gangtok as its capital. Sikkim came under the suzerainty of India, with the condition that it would retain its independence, by the treaty signed between the Chogyal and the then Indian Prime Minister Jawaharlal Nehru.[9] This pact gave the Indians control of external affairs on behalf of Sikkimese. Trade between India and Tibet continued to flourish through the Nathula and Jelepla passes, offshoots of the ancient Silk Road near Gangtok. These border passes were sealed after the Sino-Indian War in 1962, which deprived Gangtok of its trading business.[10] The Nathula pass was finally opened for limited trade in 2006, fuelling hopes of economic </pre>

## Data Science – NLP – Spacy library

---

boomIn 1975, after years of political uncertainty and struggle, including riots, the monarchy was abrogated and Sikkim became India twenty-second state, with Gangtok as its capital after a referendum. Gangtok has witnessed annual landslides, resulting in loss of life and damage to property. The largest disaster occurred in June 1997, when 38 were killed and hundreds of buildings were destroyed'

```
long_text = nlp(longtext)

list_of_tokens = [
    token.text
    for token in long_text
    if not token.is_stop and not token.is_punct
]

token_frequency = Counter(list_of_tokens)

most_frequent_tokens = token_frequency.most_common(6)
print(most_frequent_tokens)
```

### Output

```
[('Gangtok', 18), ('capital', 9), ('Sikkim', 8), ('India', 8), ('state', 5), ('Indian', 4)]
```

### Note

- ✓ We can see that the keywords are gangtok , sikkim, Indian and so on.
- ✓ It gives us an idea of the text to start with.

**11. Part of Speech Tagging**

- ✓ Each word has its own role in a sentence.
- ✓ For example,
  - 'Daniel is dancing'.
  - Daniel is the person or 'Noun' and dancing is the action performed by him.
  - So it is a 'Verb'.
  - Likewise, each word can be classified.
  - This is referred as POS or Part of Speech Tagging.

**12. Parts of Speech**

- ✓ Noun
- ✓ Verb
- ✓ Adjective
- ✓ Pronoun
- ✓ Adverb
- ✓ Preposition
- ✓ Conjunction
- ✓ Interjection and so on.

## Data Science – NLP – Spacy library

---

### 13. Pos by using spacy

- ✓ POS can be implemented using both spaCy and nltk.
- ✓ First, let us see the method using spaCy
- ✓ In spaCy, the POS tags are present in the attribute of Token object.
- ✓ We can access the POS tag of particular token through the token.pos\_ attribute.

<b>Program Name</b>	Pos example demo13.py
	<pre> import spacy nlp = spacy.load('en_core_web_sm')  text = 'Daniel is singing loudly and his roommates are enjoying too' text_doc = nlp(text)  for word in text_doc:     print(word.text.ljust(10), '----', word.pos_) </pre>
<b>Output</b>	<pre> Nireekshan ----- PROPN is           ----- AUX singing      ----- VERB loudly       ----- ADV and          ----- CCONJ his          ----- DET roommates   ----- NOUN are          ----- AUX enjoying     ----- VERB too          ----- ADV </pre>

**covid.txt**

Even as the UK became the first country to grant emergency approval to Pfizer-BioNTech's Covid-19 vaccine, the US firm said it is "committed" to engaging with the Indian government to "explore opportunities" to roll it out here. The vaccine will be a challenge for India as it requires storage at -70 degrees, experts say.

Pfizer spokeswoman Roma Nair told TOI: "We are committed to advance our dialogue with the Indian government. We are working with governments across the world to understand the infrastructure requirements of each country and we have logistical plans in place. We are confident the rollout can be managed in India."

## Data Science – NLP – Spacy library

**Program Name** Preprocessing the text by using spacy  
demo14.py

```
import spacy
nlp = spacy.load('en_core_web_sm')

with open('covid.txt') as file:
    robot_text = file.read()

robot_doc = [
    word
    for word in robot_doc
    if not word.is_stop and not word.is_punct
]

print(robot_doc)
```

**Output**

[UK, country, grant, emergency, approval, Pfizer, BioNTech, Covid-19, vaccine, firm, said, committed, engaging, Indian, government, explore, opportunities, roll, vaccine, challenge, India, requires, storage, -70, degrees, experts, , Pfizer, spokeswoman, Roma, Nair, told, TOI, committed, advance, dialogue, Indian, government, working, governments, world, understand, infrastructure, requirements, country, logistical, plans, place, confident, rollout, managed, India]

## Data Science – NLP – Spacy library

---

<b>Program Name</b>	List of nouns and verbs demo15.py
---------------------	--------------------------------------

```

import spacy
nlp = spacy.load('en_core_web_sm')

with open('covid.txt') as file:
    robot_text = file.read()

robot_doc = nlp(robot_text)

robot_doc = [
    word
    for word in robot_doc
    if not word.is_stop and not word.is_punct
]

nouns = []
verbs = []

for word in robot_doc:
    if word.pos_ == 'NOUN':
        nouns.append(word)
    if word.pos_ == 'VERB':
        verbs.append(word)

print('List of Nouns in the text:', nouns)
print()
print('List of verbs in the text:', verbs)

```

**Output**

List of Nouns in the text:  
[country, emergency, approval, vaccine, firm, government, opportunities, vaccine, challenge, storage, degrees, experts, spokeswoman, dialogue, government, governments, world, infrastructure, requirements, country, plans, place, rollout]

List of verbs in the text:  
[grant, said, committed, engaging, explore, roll, requires, told, advance, working, understand, managed]

## Data Science – NLP – Spacy library

**Program Name** We can use displacy function for display.  
demo16.py

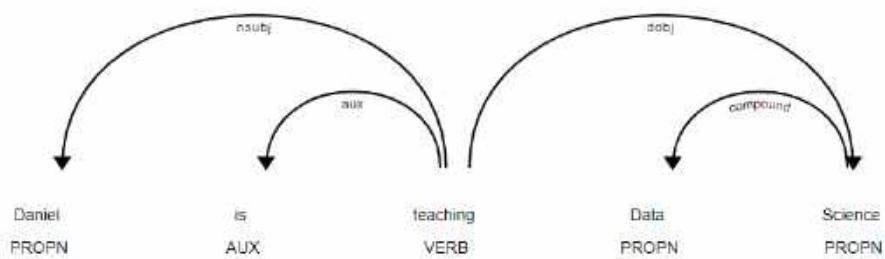
```
from spacy import displacy
import spacy
nlp = spacy.load('en_core_web_sm')

text = "Daniel is teaching Data Science"

doc = nlp(text)
displacy.render(doc)
```

**Note**

Execute above example in jupyter to see output image



### 14. Named Entity Recognition

#### Named Entity Reorganization

- ✓ Suppose you have a collection of news articles text data.
- ✓ From these we can capture,
  - companies/organization names
  - People names
- ✓ By using NER topic, we can do this

### 15. What is NER?

- ✓ NER is the technique of identifying named entities in the text corpus and assigning them pre-defined categories such as ' person names' , ' locations' , 'organizations', etc..
- ✓ It is a very useful method especially in the field of classification problems and search engine optimizations.

### 16. NER by using nltk and spacy

- ✓ NER can be implemented through both nltk and spacy'.

## Data Science – NLP – Spacy library

---

### 17. NER with spacy

- ✓ Spacy is highly flexible and advanced library.
- ✓ Named entity recognition through spacy is easy.

### 18. The few common labels are:

- ✓ 'ORG' : companies, organizations, etc.
- ✓ 'PERSON' : names of people
- ✓ 'GPE' : countries, states, etc.
- ✓ 'PRODUCT' : vehicles, food products and so on
- ✓ 'LANGUAGE' : Names of different languages
  
- ✓ There are many other labels too.

<b>Program Name</b>	Print all the named entities demo20.py
<pre>import spacy nlp = spacy.load('en_core_web_sm')  sentence=' The building is located at London. It is the headquarters of Google. Mark works there. He speaks English'  doc=nlp(sentence)  for entity in doc.ents:     print(entity.text,'--', entity.label_)</pre>	

#### Output

```
London -- GPE
Google -- ORG
Mark -- PERSON
English -- LANGUAGE
```

## Data Science – NLP – Spacy library

**Program Name**

Visualize all the named entities  
demo21.py

```
import spacy

nlp = spacy.load('en_core_web_sm')

sentence=' The building is located at London. It is the headquarters of
Google. Mark works there. He speaks English'

doc=nlp(sentence)

displacy.render(doc, style='ent', jupyter=True)
```

**Output**

The building is located at London GPE . It is the headquarters of Google ORG . Mark PERSON works there. He speaks English LANGUAGE

## Data Science – NLP – Spacy library

---

### Use case - 1

- ✓ Let us take the text data of collection of news headlines.
- ✓ Can we get list of all names that have occurred in the news?

<b>Program Name</b>	Display only person names from the text demo22.py
	<pre>from spacy import displacy import spacy  nlp = spacy.load('en_core_web_sm')  news_articles = 'Honoured to serve India, Narendra Modi, 68, wrote in a Twitter post that popped up on the micro blogging site as the ceremony started at 7 p.m. before an audience of 8,000 people, who included United Progressive Alliance chairperson Sonia Gandhi and her son and Congress president Rahul Gandhi, both seated on the front row. Shah, 54, whose inclusion in the cabinet had been much speculated upon, was administered the oaths after Narendra Modi and Rajnath Singh, indicating the order of seniority in the new government , which took office exactly a week after results from the 17th general elections were declared'  news_doc=nlp(news_articles)  list_of_people=[]  for token in news_doc:     if token.ent_type_=='PERSON':         list_of_people.append(token.text)  print(list_of_people)</pre>
<b>Output</b>	[Sonia', 'Gandhi', 'Rahul', 'Gandhi', 'Narendra', 'Modi', 'Rajnath', 'Singh']

**E-Commerce Purchase Intention Model****Contents**

<b>1. E-Commerce application goal .....</b>	3
<b>2. Purchase intention model.....</b>	4
<b>3. Google Ads.....</b>	4
<b>4. Work flow of ecommerce purchase intention models .....</b>	5
<b>5. Important features from the dataset .....</b>	6
<b>6. Install packages .....</b>	7
<b>7. Loading dataset using pandas.....</b>	8
<b>8. Number of rows and columns.....</b>	9
<b>9. Dataset information .....</b>	10
<b>10. Feature engineering .....</b>	11
10.1. replace() method .....	11
10.2. Weekend column .....	12
10.3. Revenue column .....	14
10.4. VisitorType column .....	16
10.5. VisitorType column .....	17
10.6. Month column .....	19
<b>11. Target variable is Revenue column.....</b>	23
<b>12. Pearson correlation.....</b>	24
<b>13. PageValues column .....</b>	24
<b>14. How to identify customers interest.....</b>	24
<b>15. Create the training and test data.....</b>	33
<b>16. Train and Test data.....</b>	35
<b>17. A Machine Learning pipeline .....</b>	37
<b>18. Create a model pipeline .....</b>	37
18.1. SelectKBest and SMOTE .....	37
18.2. model_pipeline(X, model) function explanation .....	38
18.3. Kind note over model pipeline.....	38
<b>19. Select best model .....</b>	40
19.1. Ameerpet Standard.....	40
19.2. Hi-Tech City Standard .....	40
19.3. Function explanation .....	41
19.4. Kind note over select best model .....	41

## Data Science – E-Commerce Purchase Intention Model

---

<b>20. let's select_model function.....</b>	44
<b>21. The best model is .....</b>	53
<b>22. Examine the performance of the best model .....</b>	53
<b>23. Best model score .....</b>	53
<b>24. Final results and words .....</b>	65
<b>25. Happy learning .....</b>	65

### E-Commerce Purchase Intention Model

#### 1. E-Commerce application goal

- ✓ Any eCommerce application main goal is,
  - Converting browsers into buyers.



## Data Science – E-Commerce Purchase Intention Model

---

### 2. Purchase intention model

- ✓ Ecommerce purchase intention models predict the probability of each customer making a purchase or not.
- ✓ Once we identify then we can target those customers.
- ✓ We will learn here how they work and build model
- ✓ Ecommerce purchase intention models analyse click-stream consumer behaviour data from web analytics platforms.
- ✓ After analysis it can predict whether a customer will make a purchase during their visit.
- ✓ These online shopping models are used to examine real time web analytics data and predict the probability of each customer making a purchase, so the retailer can serve a carefully targeted promotion to try and persuade those less likely to purchase.

### 3. Google Ads

- ✓ Generally while browsing we can see some Google ads
- ✓ Surprisingly person to person these ads are different.
- ✓ The point is, Google Company implemented Analytical techniques to target customer by using Google Ads.

### 4. Work flow of ecommerce purchase intention models

- ✓ The problem statement is, customer will BUY product or NOT
- ✓ So, this is classification.
- ✓ E-Commerce purchase intention models are classifiers.
- ✓ These models designed to analyse web analytics data and predict whether a customer will buy product or not during their visit.
- ✓ These things needs to be monitored,
  - Number of times URLs visited
  - Information about product
  - Recorded the number of each page type visited
  - Time spent on the pages.

## Data Science – E-Commerce Purchase Intention Model

---

### 5. Important features from the dataset

- ✓ Below features are very important from the dataset.
- ✓ From those features we can apply feature engineering and creating new features.
- ✓ Based on our requirement few of features we can convert them into numeric.
- ✓ We need to identify the correlation of the target variable.

Feature	Description	Type
Day	Measures the closeness of the visit date to a key trading event.	Numerical
Operating system	The operating system used during the visit.	Categorical
Browser	The web browser used during the visit.	Categorical
Region	The geographic region of the visitor.	Categorical
Visitor type	Whether the customer was a new visitor or returning visitor.	Categorical
Weekend	A Boolean value indicating whether the visit fell on a weekend.	Categorical
Month	The month of the user's visit.	Categorical
Revenue	The target variable indicating whether the visit generated revenue.	Categorical

## Data Science – E-Commerce Purchase Intention Model

---

### 6. Install packages

```
pip install lightgbm  
pip install imblearn
```

## Data Science – E-Commerce Purchase Intention Model

### 7. Loading dataset using pandas

Program Name: demo1.py  
File: online\_shoppers\_intention.csv

```
import pandas as pd

df = pd.read_csv("online_shoppers_intention.csv")

print(df.head())
```

#### Output

	Administrative	Administrative_Duration	Informational	...	VisitorType	Weekend	Revenue
0	0	0.0	0	...	Returning_Visitor	False	False
1	0	0.0	0	...	Returning_Visitor	False	False
2	0	0.0	0	...	Returning_Visitor	False	False
3	0	0.0	0	...	Returning_Visitor	False	False
4	0	0.0	0	...	Returning_Visitor	True	False

[5 rows x 18 columns]

## Data Science – E-Commerce Purchase Intention Model

### 8. Number of rows and columns

**Program Name** Checking number of rows and column

**Name** demo2.py

**File** online\_shoppers\_intention.csv

```
import pandas as pd

df = pd.read_csv("online_shoppers_intention.csv")

print(df.shape)
```

**Output**

(12330, 18)

## 9. Dataset information

**Program Name** Checking Dataset information  
**File** demo3.py  
 online\_shoppers\_intention.csv

```
import pandas as pd

df = pd.read_csv("online_shoppers_intention.csv")

print(df.info())
```

### Output

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12330 entries, 0 to 12329
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype  
 ---  --  
 0   Administrative    12330 non-null   int64  
 1   Administrative_Duration 12330 non-null   float64 
 2   Informational     12330 non-null   int64  
 3   Informational_Duration 12330 non-null   float64 
 4   ProductRelated    12330 non-null   int64  
 5   ProductRelated_Duration 12330 non-null   float64 
 6   BounceRates       12330 non-null   float64 
 7   ExitRates         12330 non-null   float64 
 8   PageValues        12330 non-null   float64 
 9   SpecialDay        12330 non-null   float64 
 10  Month            12330 non-null   object  
 11  OperatingSystems 12330 non-null   int64  
 12  Browser          12330 non-null   int64  
 13  Region           12330 non-null   int64  
 14  TrafficType      12330 non-null   int64  
 15  VisitorType      12330 non-null   object  
 16  Weekend          12330 non-null   bool   
 17  Revenue          12330 non-null   bool  
dtypes: bool(2), float64(7), int64(7), object(2)
memory usage: 1.5+ MB
```

### 10. Feature engineering

- ✓ The **Weekend** and **Revenue** columns are currently set to Boolean values, so we first need to convert into binary values.

#### 10.1. replace() method

- ✓ `replace()` is predefined method in Series class.
- ✓ We should access this method by using series object.
- ✓ This method replace existing values with desired values.

## Data Science – E-Commerce Purchase Intention Model

### 10.2. Weekend column

**Program** Checking unique values in Weekend column

**Name** demo4.py

**File** online\_shoppers\_intention.csv

```
import pandas as pd

df = pd.read_csv("online_shoppers_intention.csv")

print(df['Weekend'].unique())
```

**Output**

[False True]

## Data Science – E-Commerce Purchase Intention Model

**Program Name** Convert Weekend column to binary values  
**File** demo5.py  
online\_shoppers\_intention.csv

```
import pandas as pd

df = pd.read_csv("online_shoppers_intention.csv")

df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))

print(df.head())
```

### Output

```
   Administrative  Administrative_Duration  Informational ...  VisitorType  Weekend  Revenue
0:           0                   0.0          0.0 ...  Returning_Visitor     0    False
1:           0                   0.0          0.0 ...  Returning_Visitor     0    False
2:           0                   0.0          0.0 ...  Returning_Visitor     0    False
3:           0                   0.0          0.0 ...  Returning_Visitor     0    False
4:           0                   0.0          0.0 ...  Returning_Visitor     1    False
[5 rows x 18 columns]
```

## Data Science – E-Commerce Purchase Intention Model

### 10.3. Revenue column

**Program Name** Checking unique values in Revenue column

**Name** demo6.py

**File** online\_shoppers\_intention.csv

```
import pandas as pd

df = pd.read_csv("online_shoppers_intention.csv")

print(df['Revenue'].unique())
```

**Output**

[False True]

## Data Science – E-Commerce Purchase Intention Model

**Program Name** Convert Revenue column to binary values  
**File** demo7.py  
online\_shoppers\_intention.csv

```
import pandas as pd

df = pd.read_csv("online_shoppers_intention.csv")

df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))
df['Revenue'] = df['Revenue'].replace((True, False), (1, 0))

print(df.head())
```

### Output

```
   Administrative  Administrative_Duration  Informational  ...  VisitorType  Weekend  Revenue
0            0                  0.0           0 ...  Returning_Visitor     0      0
1            0                  0.0           0 ...  Returning_Visitor     0      0
2            0                  0.0           0 ...  Returning_Visitor     0      0
3            0                  0.0           0 ...  Returning_Visitor     0      0
4            0                  0.0           0 ...  Returning_Visitor     1      0

[5 rows x 18 columns]
```

#### 10.4. Let's understand VisitorType column

- ✓ VisitorType contains either Returning\_Visitor or New\_Visitor.
- ✓ For us one value is enough because other value is opposite to existing value.

Program

Checking unique values in VisitorType column

Name

demo8.py

File

online\_shoppers\_intention.csv

```
import pandas as pd

df = pd.read_csv("online_shoppers_intention.csv")

df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))
df['Revenue'] = df['Revenue'].replace((True, False), (1, 0))

print(df['VisitorType'].unique())
```

Output

```
['Returning_Visitor' 'New_Visitor' 'Other']
```

## Data Science – E-Commerce Purchase Intention Model

---

### 10.5. VisitorType column

- ✓ VisitorType contains either Returning\_Visitor or New\_Visitor.
- ✓ For us one value is enough because other value is opposite to existing value.
- ✓ Let's add Returning\_Visitor column to existing dataframe

**Program** Adding Returning\_Visitor column and Convert VisitorType column to binary values

**Name** demo9.py  
**File** online\_shoppers\_intention.csv

```
import pandas as pd
import numpy as np

df = pd.read_csv("online_shoppers_intention.csv")

df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))
df['Revenue'] = df['Revenue'].replace((True, False), (1, 0))

condition = df['VisitorType']=='Returning_Visitor'

df['Returning_Visitor'] = np.where(condition, 1, 0)

print(df.head())
```

### Output

	Administrative	Administrative_Duration	Informational	...	Weekend	Revenue	Returning_Visitor
0	0	0.0	0	...	0	0	1
1	0	0.0	0	...	0	0	1
2	0	0.0	0	...	0	0	1
3	0	0.0	0	...	0	0	1
4	0	0.0	0	...	1	0	1

[5 rows x 19 columns]

## Data Science – E-Commerce Purchase Intention Model

---

**Program Name** Drop VisitorType column  
**File** demo10.py  
 online\_shoppers\_intention.csv

```
import pandas as pd
import numpy as np

df = pd.read_csv("online_shoppers_intention.csv")

df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))
df['Revenue'] = df['Revenue'].replace((True, False), (1, 0))

condition = df['VisitorType']=='Returning_Visitor'

df['Returning_Visitor'] = np.where(condition, 1, 0)

df = df.drop(columns = ['VisitorType'])

print(df.head())
```

### Output

	Administrative	Administrative_Duration	Informational	...	Weekend	Revenue	Returning_Visitor
0	0	0.0	0	...	0	0	1
1	0	0.0	0	...	0	0	1
2	0	0.0	0	...	0	0	1
3	0	0.0	0	...	0	0	1
4	0	0.0	0	...	1	0	1

[5 rows x 19 columns]

### 10.6. Month column

- ✓ We do have Month column name in DataFrame.
- ✓ By default this column type recognised as object means string type
- ✓ Let's apply Ordinal Encoding over this column.

## Data Science – E-Commerce Purchase Intention Model

**Program Name** Checking all columns data type  
**File** demo11.py  
 online\_shoppers\_intention.csv

```
import pandas as pd
import numpy as np

df = pd.read_csv("online_shoppers_intention.csv")

df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))
df['Revenue'] = df['Revenue'].replace((True, False), (1, 0))

condition = df['VisitorType']=='Returning_Visitor'

df['Returning_Visitor'] = np.where(condition, 1, 0)

df = df.drop(columns = ['VisitorType'])

print(df.dtypes)
```

### Output

Administrative	int64
Administrative_Duration	float64
Informational	int64
Informational_Duration	float64
ProductRelated	int64
ProductRelated_Duration	float64
BounceRates	float64
ExitRates	float64
PageValues	float64
SpecialDay	float64
Month	object
OperatingSystems	int64
Browser	int64
Region	int64
TrafficType	int64
Weekend	int64
Revenue	int64
Returning_Visitor	int32
<b>dtype:</b>	<b>object</b>

## Data Science – E-Commerce Purchase Intention Model

**Program Name** Checking unique values in Month column

**File** demo12.py

online\_shoppers\_intention.csv

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OrdinalEncoder

df = pd.read_csv("online_shoppers_intention.csv")

df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))
df['Revenue'] = df['Revenue'].replace((True, False), (1, 0))

condition = df['VisitorType']=='Returning_Visitor'

df['Returning_Visitor'] = np.where(condition, 1, 0)

df = df.drop(columns = ['VisitorType'])

print(df['Month'].unique())
```

**Output**

```
['Feb' 'Mar' 'May' 'Oct' 'June' 'Jul' 'Aug' 'Nov' 'Sep' 'Dec']
```

## Data Science – E-Commerce Purchase Intention Model

**Program Name** Applying Ordinal Encoding on Month column

**File** demo13.py

online\_shoppers\_intention.csv

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OrdinalEncoder

df = pd.read_csv("online_shoppers_intention.csv")

df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))
df['Revenue'] = df['Revenue'].replace((True, False), (1, 0))

condition = df['VisitorType']=='Returning_Visitor'

df['Returning_Visitor'] = np.where(condition, 1, 0)

df = df.drop(columns = ['VisitorType'])

ordinal_encoder = OrdinalEncoder()

df['Month'] = ordinal_encoder.fit_transform(df[['Month']])

print(df['Month'].unique())
```

**Output**

[2. 5. 6. 8. 4. 3. 0. 7. 9. 1.]

## Data Science – E-Commerce Purchase Intention Model

### 11. Target variable is Revenue column

- ✓ Revenue column is the target variable
- ✓ Let's check the value\_counts() on this column.

**Program Name**

Let's understand the Revenue column

**File**

demo14.py

online\_shoppers\_intention.csv

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OrdinalEncoder

df = pd.read_csv("online_shoppers_intention.csv")

df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))
df['Revenue'] = df['Revenue'].replace((True, False), (1, 0))

condition = df['VisitorType']=='Returning_Visitor'

df['Returning_Visitor'] = np.where(condition, 1, 0)

df = df.drop(columns = ['VisitorType'])

ordinal_encoder = OrdinalEncoder()

df['Month'] = ordinal_encoder.fit_transform(df[['Month']])

print(df.Revenue.value_counts())
```

**Output**

```
Revenue
0      10422
1       1908
Name: count, dtype: int64
```

### 12. Pearson correlation

- ✓ We can check which of the features are correlated with the target variable by using Pearson correlation.

### 13. PageValues column

- ✓ The strongest predictor of conversion was the PageValues column.
- ✓ This column contained the Page Value metric.
- ✓ This is obviously higher for customers who have viewed product, basket, and checkout pages.
- ✓ So it makes total sense that it plays a significant role.

### 14. How to identify customers interest

- ✓ Customers who have viewed more product pages and spent longer looking at them were also much more likely to have purchased.
- ✓ Customers who shopped using specific browsers.
- ✓ Specific days shopping like weekday or weekend etc

## Data Science – E-Commerce Purchase Intention Model

---

**Program Name** Access required columns  
**File** demo15.py  
 online\_shoppers\_intention.csv

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OrdinalEncoder

df = pd.read_csv("online_shoppers_intention.csv")

df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))
df['Revenue'] = df['Revenue'].replace((True, False), (1, 0))

condition = df['VisitorType']=='Returning_Visitor'

df['Returning_Visitor'] = np.where(condition, 1, 0)

df = df.drop(columns = ['VisitorType'])

ordinal_encoder = OrdinalEncoder()

df['Month'] = ordinal_encoder.fit_transform(df[['Month']])

result = df.columns[1:]

print(result)
```

### Output

```
Index(['Administrative_Duration', 'Informational', 'Informational_Duration',
       'ProductRelated', 'ProductRelated_Duration', 'BounceRates', 'ExitRates',
       'PageValues', 'SpecialDay', 'Month', 'OperatingSystems', 'Browser',
       'Region', 'TrafficType', 'Weekend', 'Revenue', 'Returning_Visitor'],
      dtype='object')
```

## Data Science – E-Commerce Purchase Intention Model

---

**Program** Create a DataFrame with required columns

**Name** demo16.py

**File** online\_shoppers\_intention.csv

```

import pandas as pd
import numpy as np
from sklearn.preprocessing import OrdinalEncoder

df = pd.read_csv("online_shoppers_intention.csv")

df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))
df['Revenue'] = df['Revenue'].replace((True, False), (1, 0))

condition = df['VisitorType']=='Returning_Visitor'

df['Returning_Visitor'] = np.where(condition, 1, 0)

df = df.drop(columns = ['VisitorType'])

ordinal_encoder = OrdinalEncoder()

df['Month'] = ordinal_encoder.fit_transform(df[['Month']])

result = df[df.columns[1:]]

print(result)

```

**Output**

	Administrative_Duration	Informational	...	Revenue	Returning_Visitor
0	0.0	0	...	0	1
1	0.0	0	...	0	1
2	0.0	0	...	0	1
3	0.0	0	...	0	1
4	0.0	0	...	0	1
...	...	...	...	...	...
12325	145.0	0	...	0	1
12326	0.0	0	...	0	1
12327	0.0	0	...	0	1
12328	75.0	0	...	0	1
12329	0.0	0	...	0	0
[12330 rows x 17 columns]					

## Data Science – E-Commerce Purchase Intention Model

**Program Name** Correlation  
**File** demo17.py  
online\_shoppers\_intention.csv

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OrdinalEncoder

df = pd.read_csv("online_shoppers_intention.csv")

df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))
df['Revenue'] = df['Revenue'].replace((True, False), (1, 0))

condition = df['VisitorType']=='Returning_Visitor'

df['Returning_Visitor'] = np.where(condition, 1, 0)

df = df.drop(columns = ['VisitorType'])

ordinal_encoder = OrdinalEncoder()

df['Month'] = ordinal_encoder.fit_transform(df[['Month']])

result = df[df.columns[1:]].corr()

print(result)
```

**Output**

## Data Science – E-Commerce Purchase Intention Model

	Administrative_Duration	Informational	...	Revenue	Returning_Visitor
Administrative_Duration	1.000000	0.382710	...	0.093587	-0.022525
Informational	0.302710	1.000000	...	0.095208	0.057399
Informational_Duration	0.238031	0.618955	...	0.070345	0.045501
ProductRelated	0.289087	0.374164	...	0.158538	0.128738
ProductRelated_Duration	0.355422	0.387505	...	0.152373	0.120489
BounceRates	-0.144170	-0.116114	...	-0.150673	0.129908
ExitRates	-0.285798	-0.163666	...	-0.287871	0.171987
PageValues	0.067508	0.048632	...	0.492569	-0.115825
SpecialDay	-0.073304	-0.048219	...	0.082305	0.087123
Month	0.029061	0.019743	...	0.080150	0.036689
OperatingSystems	-0.007343	-0.009527	...	-0.014668	-0.038345
Browser	-0.015392	-0.038235	...	0.023984	-0.058836
Region	-0.005561	-0.029169	...	-0.011595	-0.050829
TrafficType	-0.014376	-0.034491	...	-0.005113	-0.026219
Weekend	0.014990	0.035785	...	0.029295	-0.039444
Revenue	0.093587	0.095208	...	1.000000	-0.103843
Returning_Visitor	-0.022525	0.057399	...	-0.103843	1.000000

[17 rows x 17 columns]

## Data Science – E-Commerce Purchase Intention Model

**Program Name** Correlation  
**File** demo18.py  
online\_shoppers\_intention.csv

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OrdinalEncoder

df = pd.read_csv("online_shoppers_intention.csv")

df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))
df['Revenue'] = df['Revenue'].replace((True, False), (1, 0))

condition = df['VisitorType']=='Returning_Visitor'

df['Returning_Visitor'] = np.where(condition, 1, 0)

df = df.drop(columns = ['VisitorType'])

ordinal_encoder = OrdinalEncoder()

df['Month'] = ordinal_encoder.fit_transform(df[['Month']])

result = df[df.columns[1:]].corr()['Revenue']
print(result)
```

**Output**

## Data Science – E-Commerce Purchase Intention Model

```
Administrative_Duration      0.093587
Informational                  0.095200
Informational_Duration        0.070345
ProductRelated                 0.158538
ProductRelated_Duration       0.152373
BounceRates                   -0.150673
ExitRates                      -0.207071
PageValues                     0.492569
SpecialDay                     -0.082305
Month                           0.080150
OperatingSystems                -0.014668
Browser                         0.023984
Region                          -0.011595
TrafficType                     -0.005113
Weekend                         0.029295
Revenue                         1.000000
Returning_Visitor              -0.103843
Name: Revenue, dtype: float64
```

## Data Science – E-Commerce Purchase Intention Model

**Program Name** Correlation  
**File** demo19.py  
online\_shoppers\_intention.csv

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OrdinalEncoder

df = pd.read_csv("online_shoppers_intention.csv")

df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))
df['Revenue'] = df['Revenue'].replace((True, False), (1, 0))

condition = df['VisitorType']=='Returning_Visitor'

df['Returning_Visitor'] = np.where(condition, 1, 0)

df = df.drop(columns = ['VisitorType'])

ordinal_encoder = OrdinalEncoder()

df['Month'] = ordinal_encoder.fit_transform(df[['Month']])

result = df[df.columns[1:]].corr()['Revenue']
result1 = result.sort_values(ascending=False)

print(result1)
```

**Output**

## Data Science – E-Commerce Purchase Intention Model

```
Revenue           1.000000
PageValues        0.492569
ProductRelated    0.158538
ProductRelated_Duration 0.152373
Informational     0.095200
Administrative_Duration 0.093587
Month             0.080150
Informational_Duration 0.070345
Weekend           0.029295
Browser            0.023984
TrafficType       -0.005113
Region             -0.011595
OperatingSystems   -0.014668
SpecialDay         -0.082305
Returning_Visitor  -0.103843
BounceRates        -0.150673
ExitRates          -0.207071
Name: Revenue, dtype: float64
```

### 15. Create the training and test data

- ✓ Now we need to prepare the feature set, we need to define X and y.
- ✓ The X feature set will include all the features we created above
- ✓ The y feature having target variable alone.

## Data Science – E-Commerce Purchase Intention Model

---

**Program Name** Create features and target  
**File** demo20.py  
 online\_shoppers\_intention.csv

```

import pandas as pd
import numpy as np
from sklearn.preprocessing import OrdinalEncoder

df = pd.read_csv("online_shoppers_intention.csv")

df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))
df['Revenue'] = df['Revenue'].replace((True, False), (1, 0))

condition = df['VisitorType']=='Returning_Visitor'

df['Returning_Visitor'] = np.where(condition, 1, 0)

df = df.drop(columns = ['VisitorType'])

ordinal_encoder = OrdinalEncoder()

df['Month'] = ordinal_encoder.fit_transform(df[['Month']])

result = df[df.columns[1:]].corr()['Revenue']
result1 = result.sort_values(ascending=False)

X = df.drop(['Revenue'], axis=1)
y = df['Revenue']

print("Features and target created")
  
```

### Output

Features and target created

## 16. Train and Test data

- ✓ We need to split(randomly) data into training(70%) and testing(30%) datasets.
- ✓ We can split data by using `train_test_split(X, y, test_size = 0.3, random_state)` function.
- ✓ The `random_state` value ensures we get reproducible results each time we run the code.

## Data Science – E-Commerce Purchase Intention Model

---

**Program Name** Creating train and test datasets  
**File** demo21.py  
 online\_shoppers\_intention.csv

```

import pandas as pd
import numpy as np
from sklearn.preprocessing import OrdinalEncoder

df = pd.read_csv("online_shoppers_intention.csv")

df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))
df['Revenue'] = df['Revenue'].replace((True, False), (1, 0))

condition = df['VisitorType']=='Returning_Visitor'

df['Returning_Visitor'] = np.where(condition, 1, 0)

df = df.drop(columns = ['VisitorType'])

ordinal_encoder = OrdinalEncoder()

df['Month'] = ordinal_encoder.fit_transform(df[['Month']])

result = df[df.columns[1:]].corr()['Revenue']
result1 = result.sort_values(ascending=False)

X = df.drop(['Revenue'], axis=1)
y = df['Revenue']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size = 0.3, random_state = 0)

print("Created train and test datasets")

```

### Output

Created train and test datasets

### 17. A Machine Learning pipeline

- ✓ This is the process of automating the workflow of a complete machine learning task.
- ✓ This is having group of steps like data to be transformed and correlated together in a model that can be analysed to get the output.
- ✓ A typical pipeline includes raw data input, features, outputs, model parameters, ML models, and Predictions.
- ✓ Pipeline also having sequential steps that perform everything like data extraction and pre-processing to model training and deployment in Machine learning in a modular approach.
- ✓ It means that in the pipeline, each step is designed as an independent module, and all these modules are tied together to get the final result.

### 18. Create a model pipeline

- ✓ Now we need to create a model pipeline for our project
- ✓ This Pipeline handles the encoding of data using the ColumnTransformer().
- ✓ By using this we can even avoid like manually generating some of the features we created above.
- ✓ This also imputes any missing values with realistic values.
- ✓ It scales the data before we pass it to the model.

#### 18.1. SelectKBest and SMOTE

- ✓ By using SelectKBest() we can select the optimal features. From this we are getting around 6 features as per shown in Pearson correlation.
- ✓ By using SMOTE(Synthetic Minority Oversampling Technique) we can handle class imbalance.

### 18.2. **model\_pipeline(X, model)** function explanation

- ✓ `model_pipeline(X, model)` is a user defined function for this project.
- ✓ This function returns pipeline to pre-process data and bundle with a model.
- ✓ There are two arguments for this function X means `X_train` data and model means model object Ex: `XGBClassifier` object.

### 18.3. Kind note over model pipeline

- ✓ Requesting kindly spend some time to understand this pipeline code.
- ✓ Don't worry I am happy to explain every piece of line.
  - **From Daniel**

## Data Science – E-Commerce Purchase Intention Model

**Program Name** A function to create model pipeline  
daniel\_model\_pipeline.py

```
def model_pipeline(X, model):

    n_c = X.select_dtypes(exclude=['object']).columns.tolist()
    c_c = X.select_dtypes(include=['object']).columns.tolist()

    numeric_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy='constant')),
        ('scaler', MinMaxScaler())
    ])

    categorical_pipeline = Pipeline([
        ('encoder', OneHotEncoder(handle_unknown='ignore'))
    ])

    preprocessor = ColumnTransformer([
        ('numeric', numeric_pipeline, n_c),
        ('categorical', categorical_pipeline, c_c)
    ], remainder='passthrough')

    final_steps = [
        ('preprocessor', preprocessor),
        ('smote', SMOTE(random_state=1)),
        ('feature_selection', SelectKBest(score_func = chi2, k = 6)),
        ('model', model)
    ]

    return IMBPipeline(steps = final_steps)
```

## 19. Select best model

### 19.1. Ameerpet Standard

- ✓ Generally we can select single model to proceeding in further steps.
- ✓ Many times we use to run our code repeatedly by taking different models as well.
- ✓ Instead of checking every time with different models, we can create a function to select the best model

### 19.2. Hi-Tech City Standard

- ✓ It would be **good** to create one function to select the best model.
- ✓ This function **automatically** test all models and finally returns the best model as per our requirement
- ✓ In this function, in very first step we are creating **dictionary** with XGBoost, Random Forest, Decision Tree, SVC, and a Multilayer Perceptron among others.
- ✓ We can **loop through** these models, run the data through the pipeline for each model.
- ✓ Use **cross-validation** to get good performance, reliability and validity of each model.
- ✓ Store the **model results** in pandas DataFrame and print the output.
- ✓ Finally selecting the **BEST MODEL** with highest ROC/AUC score

### 19.3. Function explanation

- ✓ `select_model(X, y, pipeline = None)` is user defined function created by us.
- ✓ This function takes **2** non default parameters and **1** default parameter
  - `X` (object) : Pandas dataframe containing `X_train` data.
  - `y` (object) : Pandas dataframe containing `y_train` data.
  - `pipeline` : Pipeline from `model_pipeline()`.

### 19.4. Kind note over select best model

- ✓ Requesting kindly spend some time to understand this select best model code.
- ✓ Don't worry I am happy to explain every piece of line.
  - **From Daniel**

## Data Science – E-Commerce Purchase Intention Model

**Program Name** A function to select model  
daniel\_select\_model.py

```
def select_model(X, y, pipeline=None):

    classifiers = {}

    c_d1 = {"DummyClassifier": DummyClassifier(strategy='most_frequent')}
    classifiers.update(c_d1)

    c_d2 = {"RandomForestClassifier": RandomForestClassifier()}
    classifiers.update(c_d2)

    c_d3 = {"DecisionTreeClassifier": DecisionTreeClassifier()}
    classifiers.update(c_d3)

    c_d4 = {"KNeighborsClassifier": KNeighborsClassifier()}
    classifiers.update(c_d4)

    c_d5 = {"SVC": SVC()}
    classifiers.update(c_d5)

    mlpc = {
        "MLPClassifier (paper)": MLPClassifier(hidden_layer_sizes=(27, 50),
                                                max_iter=300,
                                                activation='relu',
                                                solver='adam',
                                                random_state=1)
    }
```

## Data Science – E-Commerce Purchase Intention Model

```
c_d6 = mlpc
classifiers.update(c_d6)

cols = ['model', 'run_time', 'roc_auc']
df_models = pd.DataFrame(columns = cols)

for key in classifiers:
    start_time = time.time()
    pipeline = model_pipeline(X_train, classifiers[key])

    cv = cross_val_score(pipeline, X, y, cv=10,
                          scoring='roc_auc')

    row = {'model': key,
           'run_time': format(round((time.time() -
start_time)/60,2)),
           'roc_auc': cv.mean(),
           }

    df_models = df_models.append(row,
                                 ignore_index=True)

df_models = df_models.sort_values(by='roc_auc',
                                   ascending = False)

return df_models
```

## 20. let's select\_model function

- Once we call select\_model function then we will get best model from all models.

**Program Name** Access select\_model function  
**File** access\_select\_model.py  
 online\_shoppers\_intention.csv

```
#####
# Importing required librarie #
#####

print("Step 1: Required librarie imported successfully")

import time
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

from imblearn.over_sampling import SMOTE

from imblearn.pipeline import Pipeline as imbpipeline
from sklearn.pipeline import Pipeline

from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer

from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder
```

## Data Science – E-Commerce Purchase Intention Model

```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score

from sklearn.svm import SVC
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from sklearn.dummy import DummyClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import ExtraTreeClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.linear_model import RidgeClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.naive_bayes import BernoulliNB
from sklearn.neural_network import MLPClassifier
```

#####

## Data Science – E-Commerce Purchase Intention Model

```
# To ignore warning #
#####
# from warnings import simplefilter
# from sklearn.exceptions import ConvergenceWarning
# simplefilter("ignore", category=ConvergenceWarning)

#####
# Loading online_shoppers_intention.csv dataset #
#####

print("Step 2: Created DataFrame successfully")

df = pd.read_csv("online_shoppers_intention.csv")

#####
# Feature Engineering #
#####

print("Step 3: Feature Engineering Done successfully on Weekend, Revenue")

df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))
df['Revenue'] = df['Revenue'].replace((True, False), (1, 0))

#####
```

## Data Science – E-Commerce Purchase Intention Model

```
# Added Returning_Visitor column #
#####
#
print("Step 4: Added Returning_Visitor column successfully")

condition = df['VisitorType']=='Returning_Visitor'

df['Returning_Visitor'] = np.where(condition, 1, 0)

df = df.drop(columns=['VisitorType'])

#####
#
# Applying One Hot Encoding on Month column #
#####

print("Step 5: Applied one hot encoding successfully on Month
column")

ordinal_encoder = OrdinalEncoder()
df['Month'] = ordinal_encoder.fit_transform(df[['Month']])

#####
#
# Checking correlation on Revenue column #
#####

print("Step 6: Checking correlation done successfully")

result = df[df.columns[1:]].corr()['Revenue']

result1 = result.sort_values(ascending=False)
```

## Data Science – E-Commerce Purchase Intention Model

```
#####
# Preparing Features as X and target as y #
#####

print("Step 7: Preparing features as X and target as y done
successfully")

X = df.drop(['Revenue'], axis=1)
y = df['Revenue']

#####

# Preparing Train and Test Dataset#
#####

print("Step 8: Splitting data X_train, X_test, y_train & y_test done
successfully")

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state = 0)

#####
```

## Data Science – E-Commerce Purchase Intention Model

```
# Model Pipeline #
#####
print("Step 9: model_pipeline function created done successfully")

def model_pipeline(X, model):

    n_c = X.select_dtypes(exclude=['object']).columns.tolist()
    c_c = X.select_dtypes(include=['object']).columns.tolist()

    numeric_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy='constant')),
        ('scaler', MinMaxScaler())
    ])

    categorical_pipeline = Pipeline([
        ('encoder', OneHotEncoder(handle_unknown='ignore'))
    ])

    preprocessor = ColumnTransformer([
        ('numeric', numeric_pipeline, n_c),
        ('categorical', categorical_pipeline, c_c)
    ], remainder='passthrough')

    final_steps = [
        ('preprocessor', preprocessor),
        ('smote', SMOTE(random_state=1)),
        ('feature_selection', SelectKBest(score_func = chi2, k = 6)),
        ('model', model)
    ]

    return IMBPipeline(steps = final_steps)
```

## Data Science – E-Commerce Purchase Intention Model

```
#####
# Model Selection #
#####

print("Step 10: select_model function created done successfully")

def select_model(X, y, pipeline=None):

    classifiers = {}

    c_d1 = {"DummyClassifier": DummyClassifier(strategy='most_frequent')}
    classifiers.update(c_d1)

    c_d4 = {"RandomForestClassifier": RandomForestClassifier()}
    classifiers.update(c_d4)

    c_d5 = {"DecisionTreeClassifier": DecisionTreeClassifier()}
    classifiers.update(c_d5)

    c_d9 = {"KNeighborsClassifier": KNeighborsClassifier()}
    classifiers.update(c_d9)

    c_d10 = {"RidgeClassifier": RidgeClassifier()}
    classifiers.update(c_d10)

    c_d14 = {"SVC": SVC()}
    classifiers.update(c_d14)

    mlpc = {
        "MLPClassifier (paper)": MLPClassifier(hidden_layer_sizes=(27, 50),
                                                max_iter=300,
                                                activation='relu',
                                                solver='adam',
                                                random_state=1)
    }
```

## Data Science – E-Commerce Purchase Intention Model

```
c_d16 = mlpc
classifiers.update(c_d16)

cols = ['model', 'run_time', 'roc_auc']
df_models = pd.DataFrame(columns = cols)

for key in classifiers:
    start_time = time.time()
    print()
    print("Step 12: model_pipeline run successfully on",
          key)

    pipeline = model_pipeline(X_train, classifiers[key])

    cv = cross_val_score(pipeline, X, y, cv=10,
                         scoring='roc_auc')

    row = {'model': key,
           'run_time': format(round((time.time() -
                                      start_time)/60,2)),
           'roc_auc': cv.mean(),
           }

    df_models = pd.concat([df_models,
                           pd.DataFrame([row])], ignore_index=True)

df_models = df_models.sort_values(by='roc_auc',
                                   ascending = False)

return df_models
```

## Data Science – E-Commerce Purchase Intention Model

```
#####
# Access Model select_model function #
#####

print("Step 11: Accessing select_model function done
successfully")

models = select_model(X_train, y_train)

#####

# Let's see total model with score #
#####

print("Step 13: Accessing select_model function done
successfully")

print(models)
```

### Output

	model	run_time	roc_auc
14	MLPClassifier	1.79	0.903222
15	MLPClassifier (paper)	2.86	0.900244
2	LGBMClassifier	0.04	0.897217
1	XGBClassifier	0.15	0.891174
10	SGDClassifier	0.02	0.889067
7	AdaBoostClassifier	0.1	0.888264
13	SVC	0.83	0.885963
3	RandomForestClassifier	0.27	0.884997
11	BaggingClassifier	0.07	0.863183
12	BernoulliNB	0.01	0.857851
9	RidgeClassifier	0.01	0.855441
8	KNeighborsClassifier	0.02	0.840505
6	ExtraTreesClassifier	0.01	0.770749
5	ExtraTreeClassifier	0.01	0.754475
4	DecisionTreeClassifier	0.02	0.734040
0	DummyClassifier	0.01	0.500000

### 21. The best model is

- ✓ The top performer was MLPClassifier(), which generated a ROC/AUC score of 0.902.
- ✓ We'll select this model as our best one and examine the results in a bit more detail to see how well it works.

### 22. Examine the performance of the best model

- ✓ By re-running the model\_pipeline() function on our selected model as the MLPClassifier()
- ✓ We can generate predictions and assess their accuracy on the test data.

### 23. Best model score

- ✓ What this shows is that the MLPClassifier model generated a ROC/AUC score of 0.902 on the training data, which is really good, but a slightly lower ROC/AUC score of 0.836 on the test data, with an overall accuracy of 0.88.

## Data Science – E-Commerce Purchase Intention Model

**Program Name** Final code with best model and result  
**File** final\_code.py  
online\_shoppers\_intention.csv

```
#####
# Importing required librarie #
#####

print("Step 1: Required librarie imported successfully")

import time
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

from imblearn.over_sampling import SMOTE

from imblearn.pipeline import Pipeline as imbpipeline
from sklearn.pipeline import Pipeline

from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer

from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
```

## Data Science – E-Commerce Purchase Intention Model

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score

from sklearn.svm import SVC
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from sklearn.dummy import DummyClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import ExtraTreeClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.linear_model import RidgeClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.naive_bayes import BernoulliNB
from sklearn.neural_network import MLPClassifier
```

## Data Science – E-Commerce Purchase Intention Model

```
#####
# To ignore warning #
#####

from warnings import simplefilter
from sklearn.exceptions import ConvergenceWarning
simplefilter("ignore", category=ConvergenceWarning)

#####
# Loading online_shoppers_intention.csv dataset #
#####

print("Step 2: Created DataFrame successfully")

df = pd.read_csv("online_shoppers_intention.csv")

#####

# Feature Engineering #
#####

print("Step 3: Feature Engineering Done successfully on Weekend, Revenue")

df['Weekend'] = df['Weekend'].replace((True, False), (1, 0))
df['Revenue'] = df['Revenue'].replace((True, False), (1, 0))
```

## Data Science – E-Commerce Purchase Intention Model

```
#####
# Added Returning_Visitor column #
#####

print("Step 4: Added Returning_Visitor column successfully")

condition = df['VisitorType']=='Returning_Visitor'

df['Returning_Visitor'] = np.where(condition, 1, 0)

df = df.drop(columns=['VisitorType'])

#####

# Applying One Hot Encoding on Month column #
#####

print("Step 5: Applied one hot encoding successfully on Month column")

ordinal_encoder = OrdinalEncoder()
df['Month'] = ordinal_encoder.fit_transform(df[['Month']])

#####

# Checking correlation on Revenue column #
#####

print("Step 6: Checking correlation done successfully")

result = df[df.columns[1:]].corr()['Revenue']

result1 = result.sort_values(ascending=False)
```

## Data Science – E-Commerce Purchase Intention Model

```
#####
# Preparing Features as X and target as y #
#####

print("Step 7: Preparing features as X and target as y done
successfully")

X = df.drop(['Revenue'], axis=1)
y = df['Revenue']

#####

# Preparing Train and Test Dataset#
#####

print("Step 8: Splitting data X_train, X_test, y_train & y_test done
successfully")

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state = 0)
```

## Data Science – E-Commerce Purchase Intention Model

```
#####
# Model Pipeline #
#####

print("Step 9: model_pipeline function created done successfully")

def model_pipeline(X, model):

    n_c = X.select_dtypes(exclude=['object']).columns.tolist()
    c_c = X.select_dtypes(include=['object']).columns.tolist()

    numeric_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy='constant')),
        ('scaler', MinMaxScaler())
    ])

    categorical_pipeline = Pipeline([
        ('encoder', OneHotEncoder(handle_unknown='ignore'))
    ])

    preprocessor = ColumnTransformer([
        ('numeric', numeric_pipeline, n_c),
        ('categorical', categorical_pipeline, c_c)
    ], remainder='passthrough')

    final_steps = [
        ('preprocessor', preprocessor),
        ('smote', SMOTE(random_state=1)),
        ('feature_selection', SelectKBest(score_func = chi2, k =
6)),
        ('model', model)
    ]

    return IMBPipeline(steps = final_steps)
```

## Data Science – E-Commerce Purchase Intention Model

```
#####
# Model Selection #
#####

print("Step 10: select_model function created done successfully")

def select_model(X, y, pipeline=None):

    classifiers = {}

    c_d1 = {"DummyClassifier": DummyClassifier(strategy='most_frequent')}
    classifiers.update(c_d1)

    c_d4 = {"RandomForestClassifier": RandomForestClassifier()}
    classifiers.update(c_d4)

    c_d5 = {"DecisionTreeClassifier": DecisionTreeClassifier()}
    classifiers.update(c_d5)

    c_d9 = {"KNeighborsClassifier": KNeighborsClassifier()}
    classifiers.update(c_d9)

    c_d14 = {"SVC": SVC()}
    classifiers.update(c_d14)

    mlpc = {
        "MLPClassifier (paper)": MLPClassifier(hidden_layer_sizes=(27, 50),
                                                max_iter=300,
                                                activation='relu',
                                                solver='adam',
                                                random_state=1)
    }
    c_d16 = mlpc
    classifiers.update(c_d16)
```

## Data Science – E-Commerce Purchase Intention Model

```
cols = ['model', 'run_time', 'roc_auc']
df_models = pd.DataFrame(columns = cols)

for key in classifiers:
    start_time = time.time()
    print()
    print("Step 12: model_pipeline run successfully on",
          key)

    pipeline = model_pipeline(X_train, classifiers[key])

    cv = cross_val_score(pipeline, X, y, cv=10,
                         scoring='roc_auc')

    row = {'model': key,
           'run_time': format(round((time.time() -
                                      start_time)/60,2)),
           'roc_auc': cv.mean(),
           }

    df_models = pd.concat([df_models,
                           pd.DataFrame([row])], ignore_index=True)

    df_models = df_models.sort_values(by='roc_auc',
                                      ascending = False)

return df_models
```

## Data Science – E-Commerce Purchase Intention Model

```
#####
# Access Model select_model function #
#####

print("Step 11: Accessing select_model function done
successfully")

models = select_model(X_train, y_train)

#####

# Let's see total model with score #
#####

print("Step 13: Accessing select_model function done
successfully")

print(models)

#####

# Accessing best model and training #
#####

print("Step 14: Accessing select_model function done
successfully")

selected_model = MLPClassifier()
bundled_pipeline = model_pipeline(X_train, selected_model)
bundled_pipeline.fit(X_train, y_train)
```

## Data Science – E-Commerce Purchase Intention Model

```
#####
# Accessing best model and training #
#####

print("Step 15: Results predicted successfully")
y_pred = bundled_pipeline.predict(X_test)

print(y_pred)

#####
# ROC and AOC score #
#####

print("Step 16: ROC and AOC scores")

roc_auc = roc_auc_score(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
f1_score = f1_score(y_test, y_pred)

print('ROC/AUC:', roc_auc)
print('Accuracy:', accuracy)
print('F1 score:', f1_score)

#####
# Classification report #
#####

print("Step 17: classification report generated successfully")

classif_report = classification_report(y_test, y_pred)

print(classif_report)
```

## Data Science – E-Commerce Purchase Intention Model

```
#####
# This is done, BOSS it's a right time to celebrate :) #
#####
```

### Output

	model	run_time	roc_auc
14	MLPClassifier	1.79	0.903222
15	MLPClassifier (paper)	2.86	0.900244
2	LGBMClassifier	0.04	0.897217
1	XGBClassifier	0.15	0.891174
10	SGDClassifier	0.02	0.889067
7	AdaBoostClassifier	0.1	0.888264
13	SVC	0.83	0.885963
3	RandomForestClassifier	0.27	0.884997
11	BaggingClassifier	0.07	0.863183
12	BernoulliNB	0.01	0.857851
9	RidgeClassifier	0.01	0.855441
8	KNeighborsClassifier	0.02	0.840505
6	ExtraTreesClassifier	0.01	0.770749
5	ExtraTreeClassifier	0.01	0.754475
4	DecisionTreeClassifier	0.02	0.734040
0	DummyClassifier	0.01	0.500000

ROC/AUC: 0.8346658696876629  
 Accuracy: 0.8764530954311976  
 F1 score: 0.6774876499647141

	precision	recall	f1-score	support
0	0.95	0.90	0.92	3077
1	0.60	0.77	0.68	622
accuracy			0.88	3699
macro avg	0.78	0.83	0.80	3699
weighted avg	0.89	0.88	0.88	3699

### 24. Final results and words

- ✓ Examining the confusion matrix for the selected model, it shows that we correctly predicted customers (90%) wouldn't purchase during their session, and we correctly predicted that 77% of customers who would purchase during their sessions.
- ✓ Clearly, there's still more we could do, and the overall accuracy of 88%
- ✓ The results show that it's possible to predict purchase intention from consumer behaviour data with a good degree of accuracy.

### 25. Happy learning

- ✓ While learning this, if having any questions then I am happy to help/support you guys.
- ✓ From Daniel

### MySQL Installation

#### Contents

1. What is Mysql? .....	2
2. Logo.....	2
3. Steps to download and install .....	3

## MySQL Installation

### 1. What is MySql?

- ✓ MySQL is an open-source relational database management system.
- ✓ SQL is a query language to create, modify and extract data from the relational database, as well as control user access to the database.

### 2. Logo



### 3. Steps to download and install

- ✓ Click on below url

<https://dev.mysql.com/downloads/mysql/>

The screenshot shows a web browser displaying the MySQL Community Downloads page. The URL in the address bar is [dev.mysql.com/downloads/mysql/](https://dev.mysql.com/downloads/mysql/). The main heading is "MySQL Community Downloads". Below it, there's a breadcrumb navigation: "MySQL Community Server". A navigation bar at the top includes tabs for "General Availability (GA) Releases" (which is selected), "Archives", and "Help". The main content area is titled "MySQL Community Server 8.2.0 Innovation". It features two dropdown menus: "Select Version" set to "8.2.0 Innovation" and "Select Operating System" set to "Microsoft Windows". Below these are three download options:

Format	Version	File Size	Action
Windows (x86, 64-bit), MSI Installer (mysql-8.2.0-win64.msi)	8.2.0	130.4M	<a href="#">Download</a>
Windows (x86, 64-bit), ZIP Archive (mysql-8.2.0-win64.zip)	8.2.0	241.5M	<a href="#">Download</a>
Windows (x86, 64-bit), ZIP Archive Debug Binaries & Test Suite	8.2.0	683.5M	<a href="#">Download</a>

## Data Science – MySql

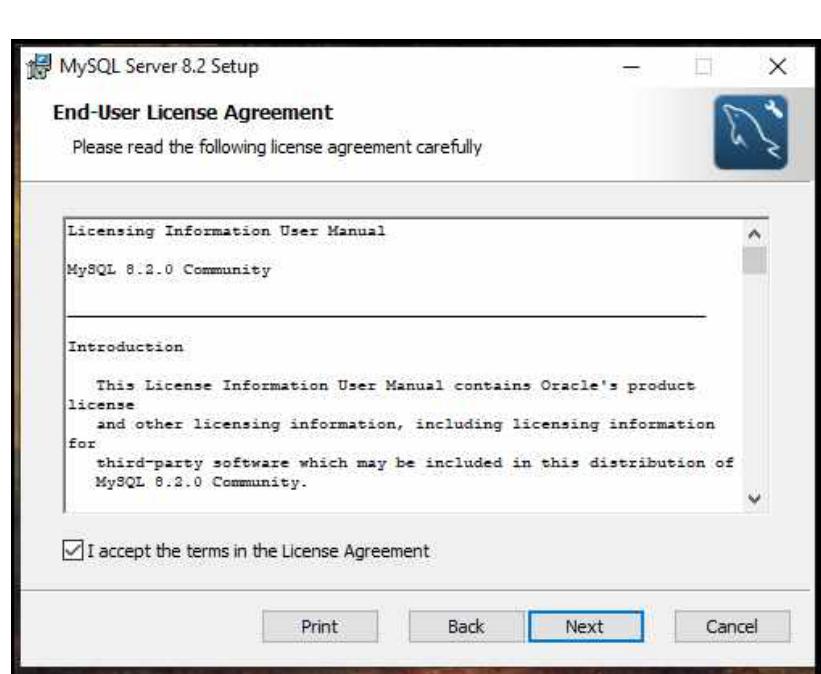
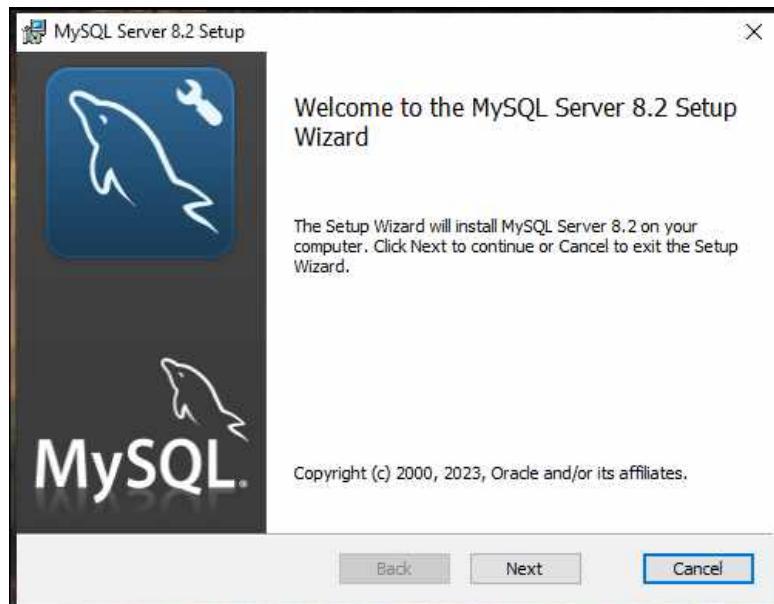
The screenshot shows a web browser window for 'MySQL :: Begin Your Download' at [dev.mysql.com/downloads/file/?id=523158](http://dev.mysql.com/downloads/file/?id=523158). The page features a header with a magnifying glass icon and the text 'MySQL Community Downloads'. Below this, there are links for 'Login Now' and 'Sign Up for a free account'. A section titled 'An Oracle Web Account provides you with the following advantages:' lists four items: 'Fast access to MySQL software downloads', 'Download technical White Papers and Presentations', 'Post messages in the MySQL Discussion Forums', and 'Report and track bugs in the MySQL bug system'. At the bottom, a note states 'MySQL.com is using Oracle SSO for authentication. If you already have an Oracle Web account, click the Login link. Otherwise, you can signup for a free account by clicking the Sign Up link and following the instructions.' A blue oval highlights the text 'No thanks, just start my download.'

✓ Its .exe file

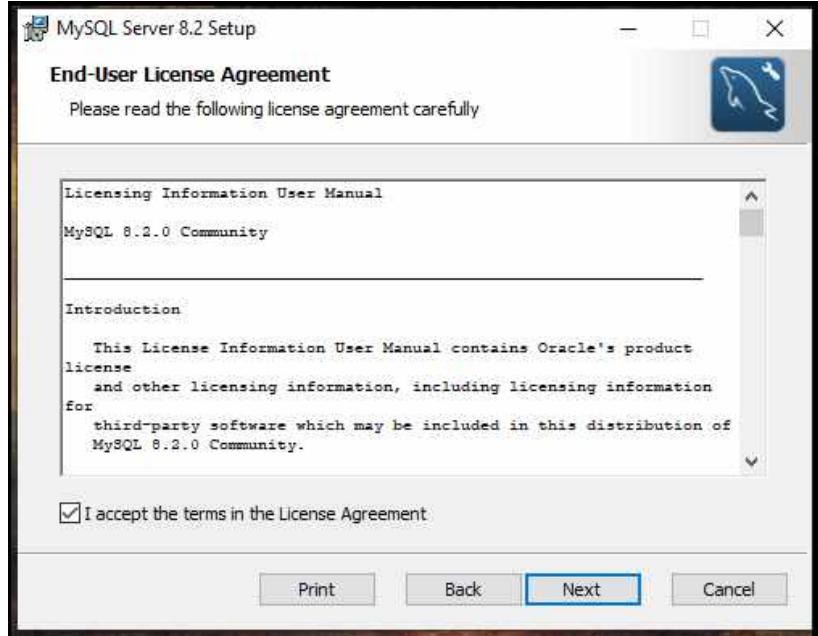
**mysql-8.2.0-winx64**

## Data Science – MySql

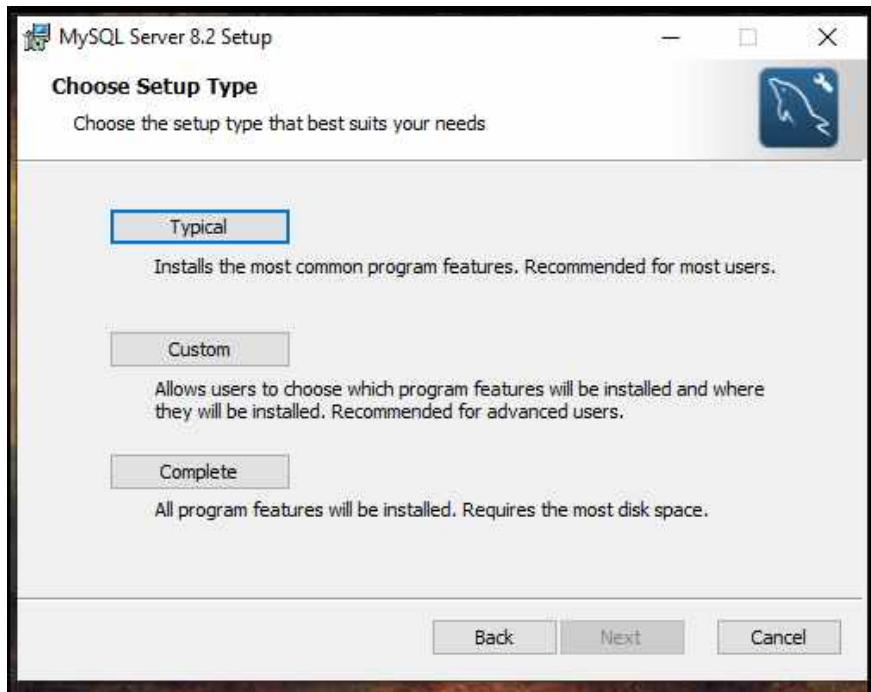
- ✓ Give double click on software.



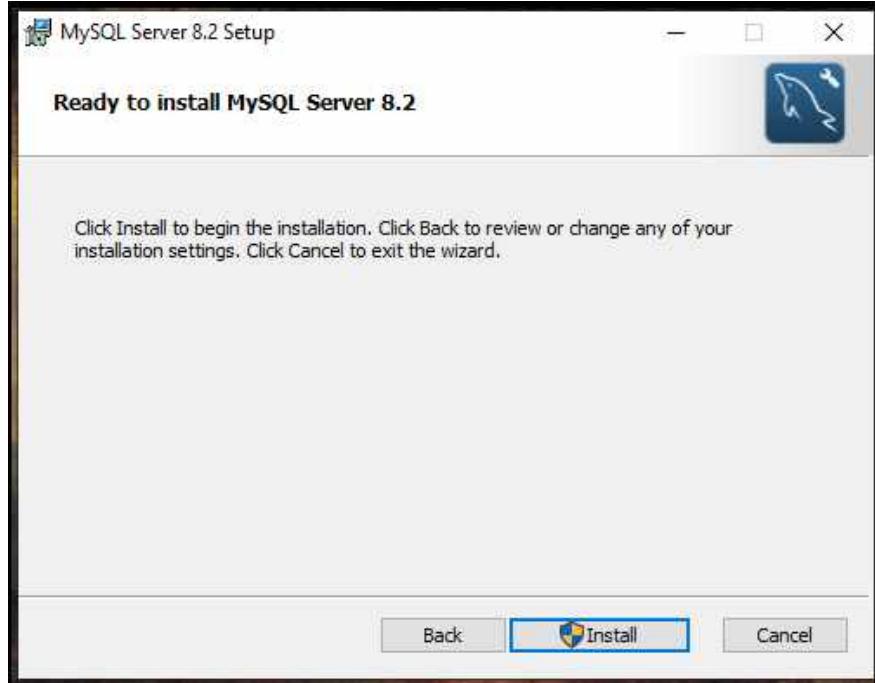
## Data Science – MySql



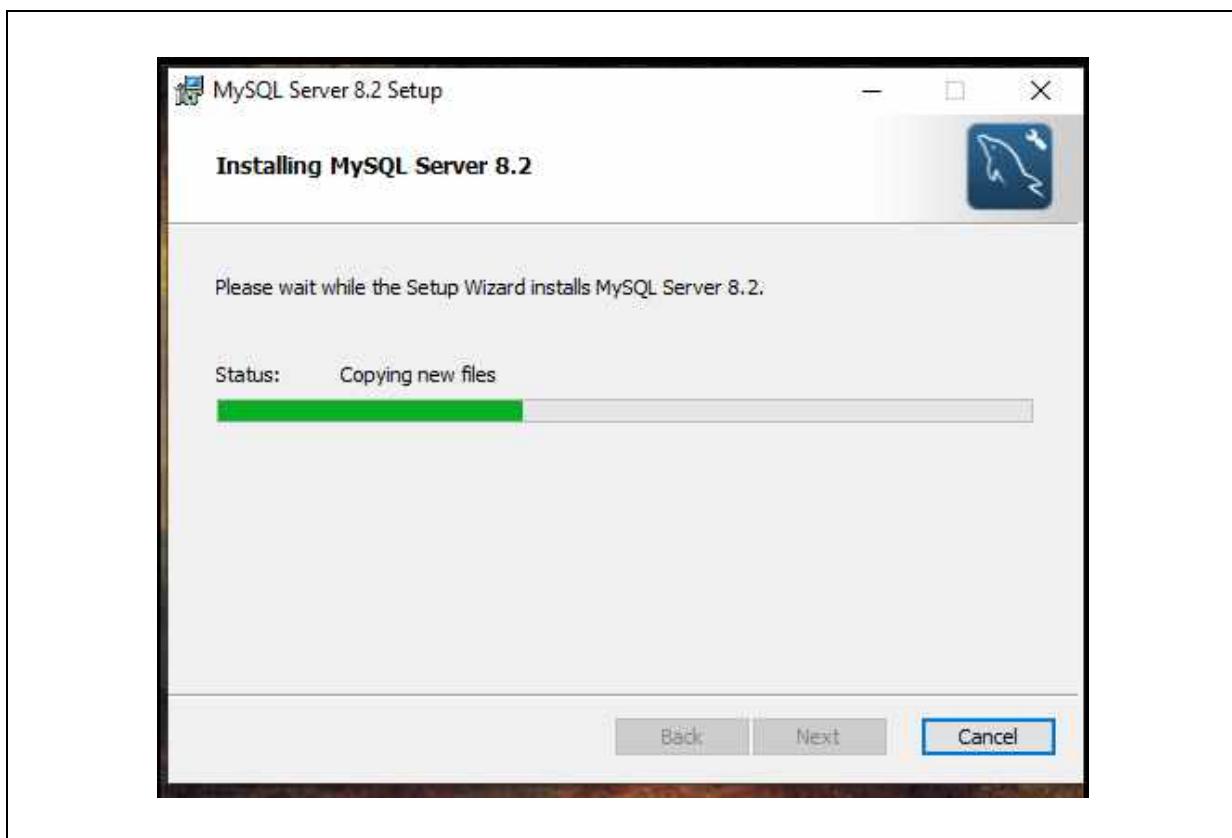
## Data Science – MySql



## Data Science – MySql

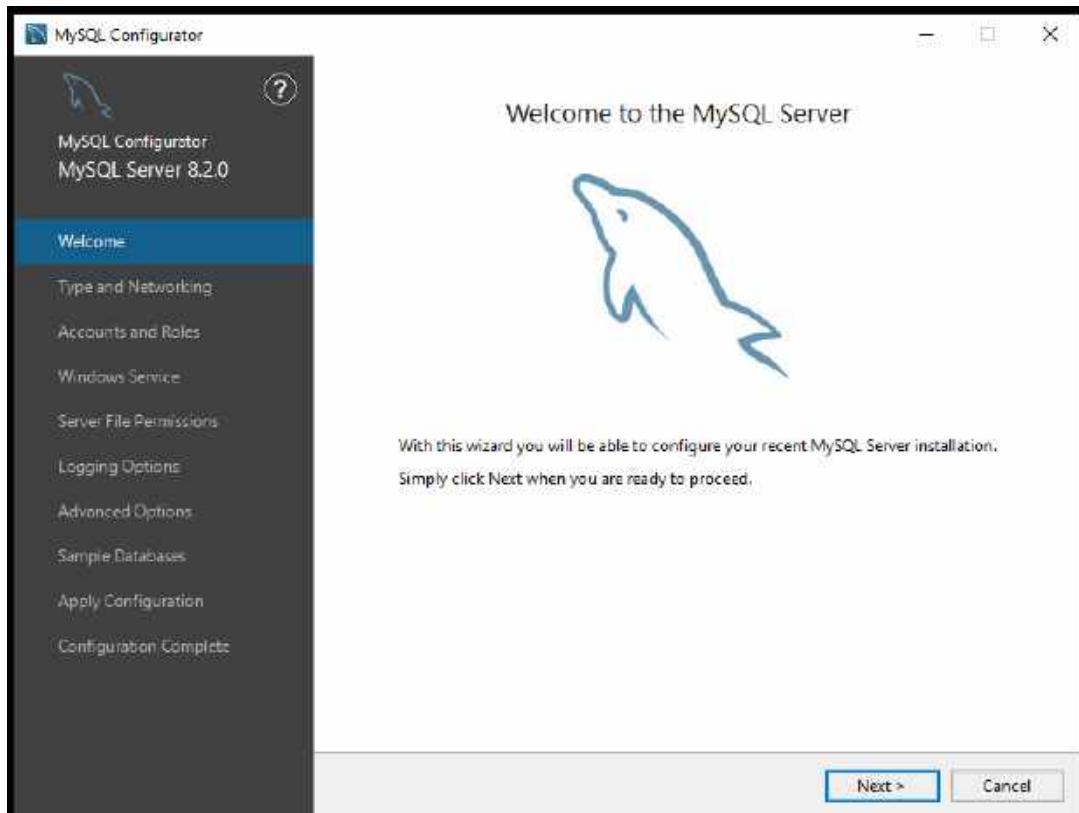


## Data Science – MySql

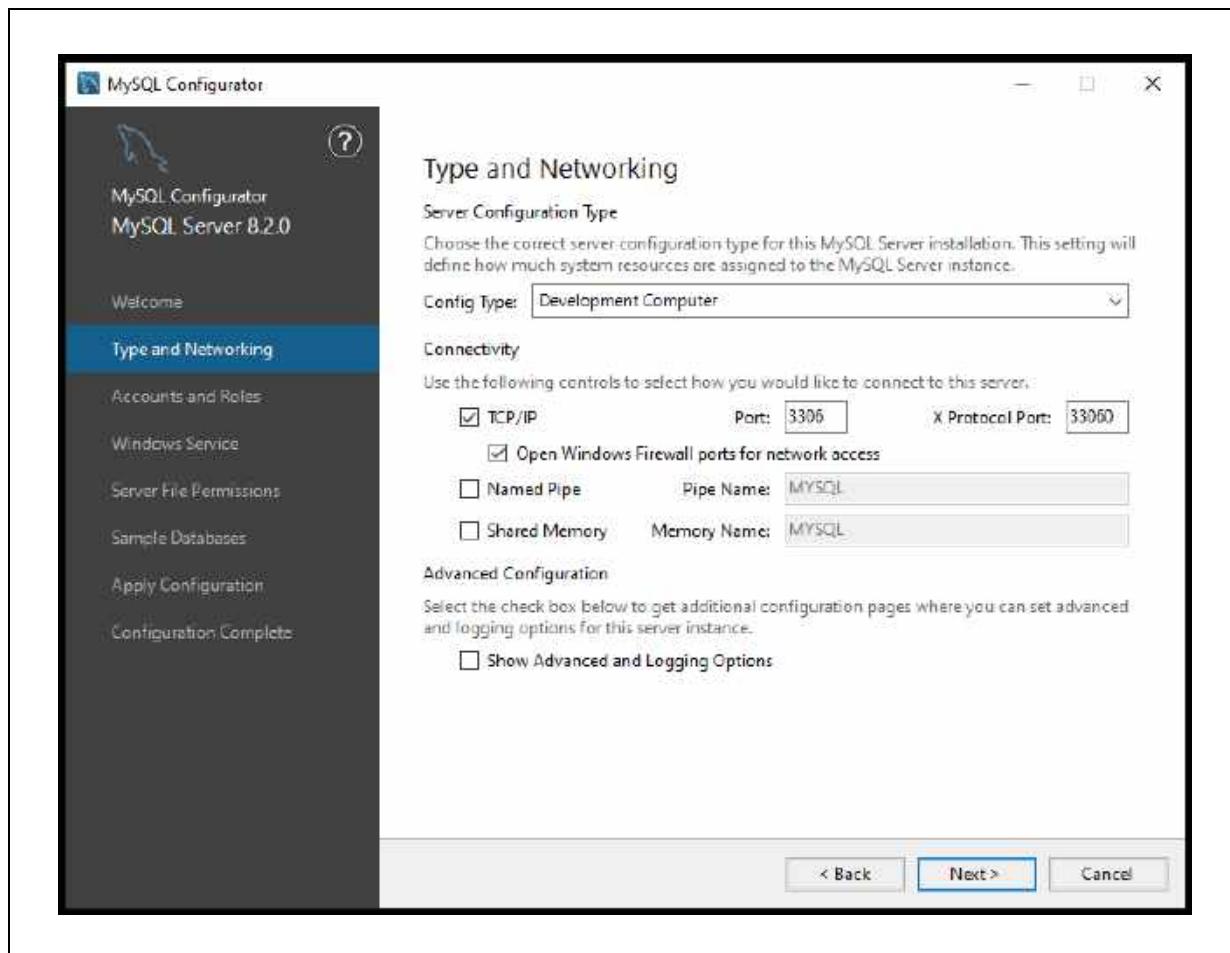


## Data Science – MySql

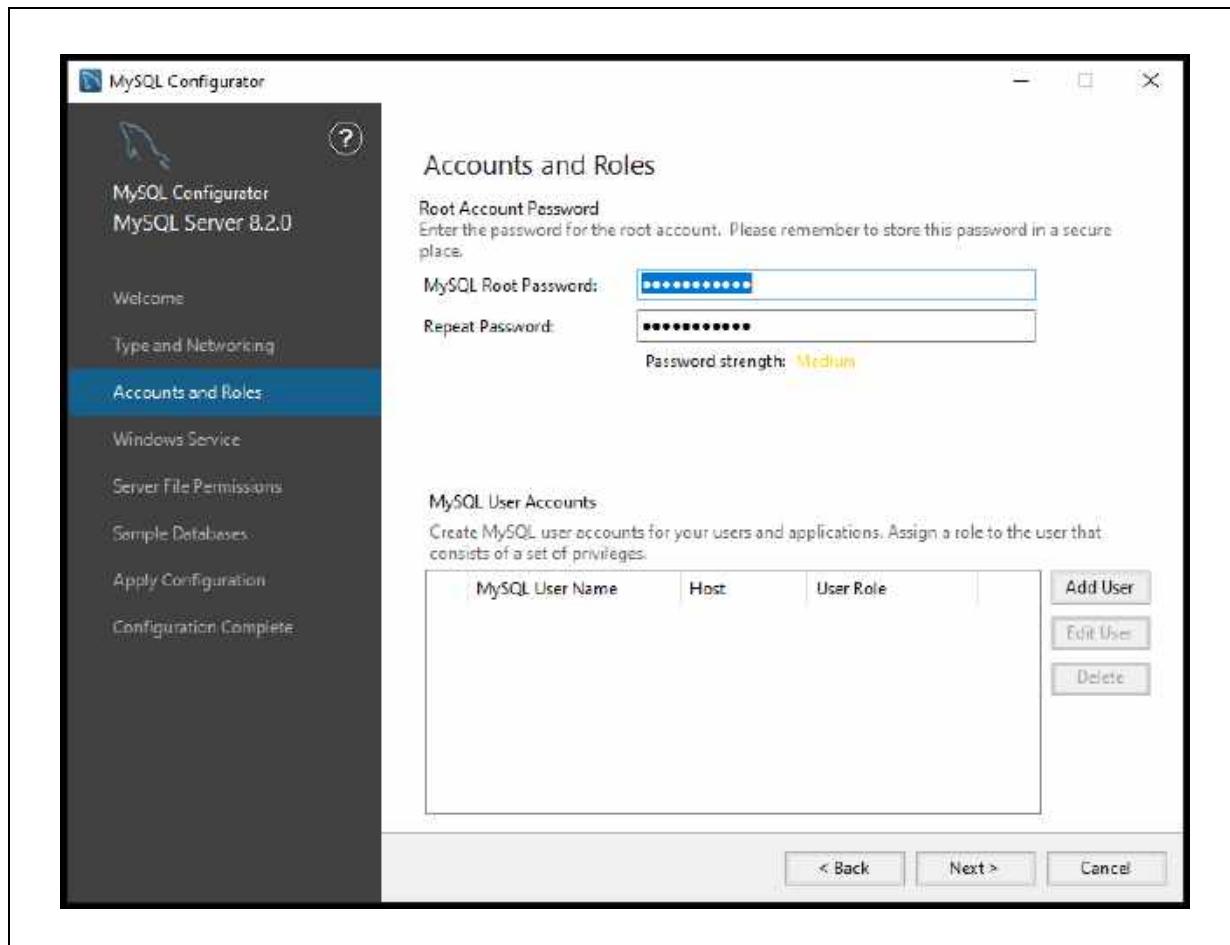
Click on finish



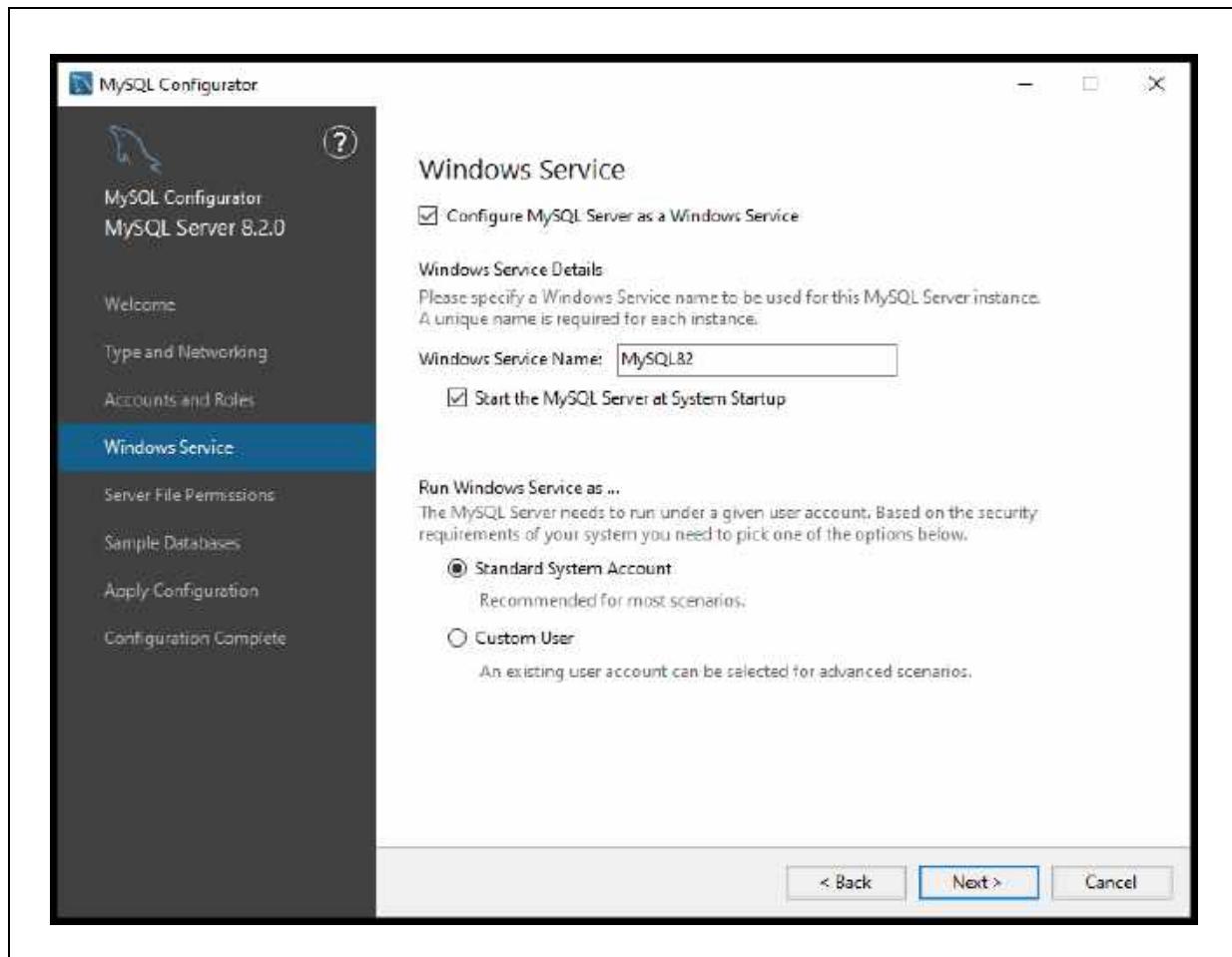
## Data Science – MySql



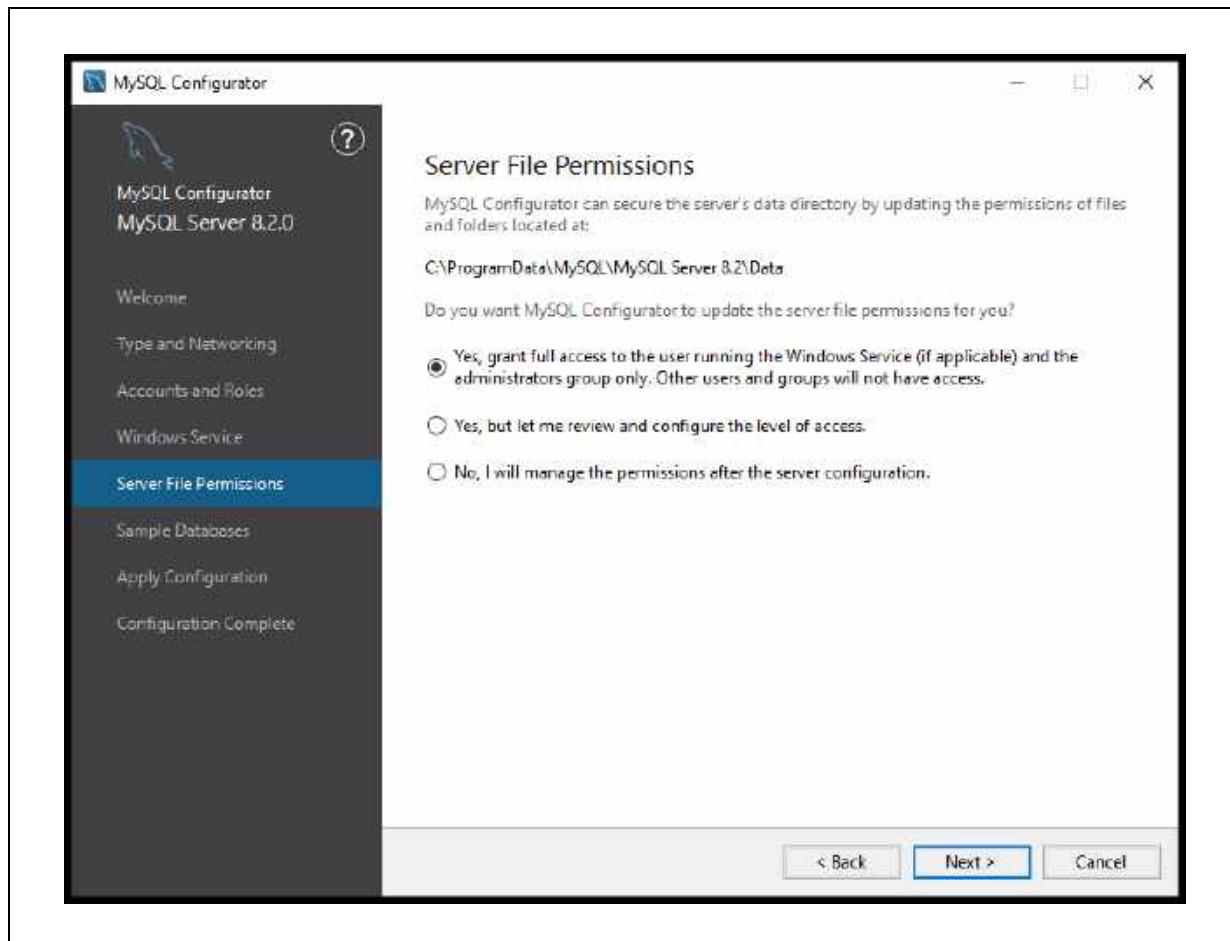
## Data Science – MySql



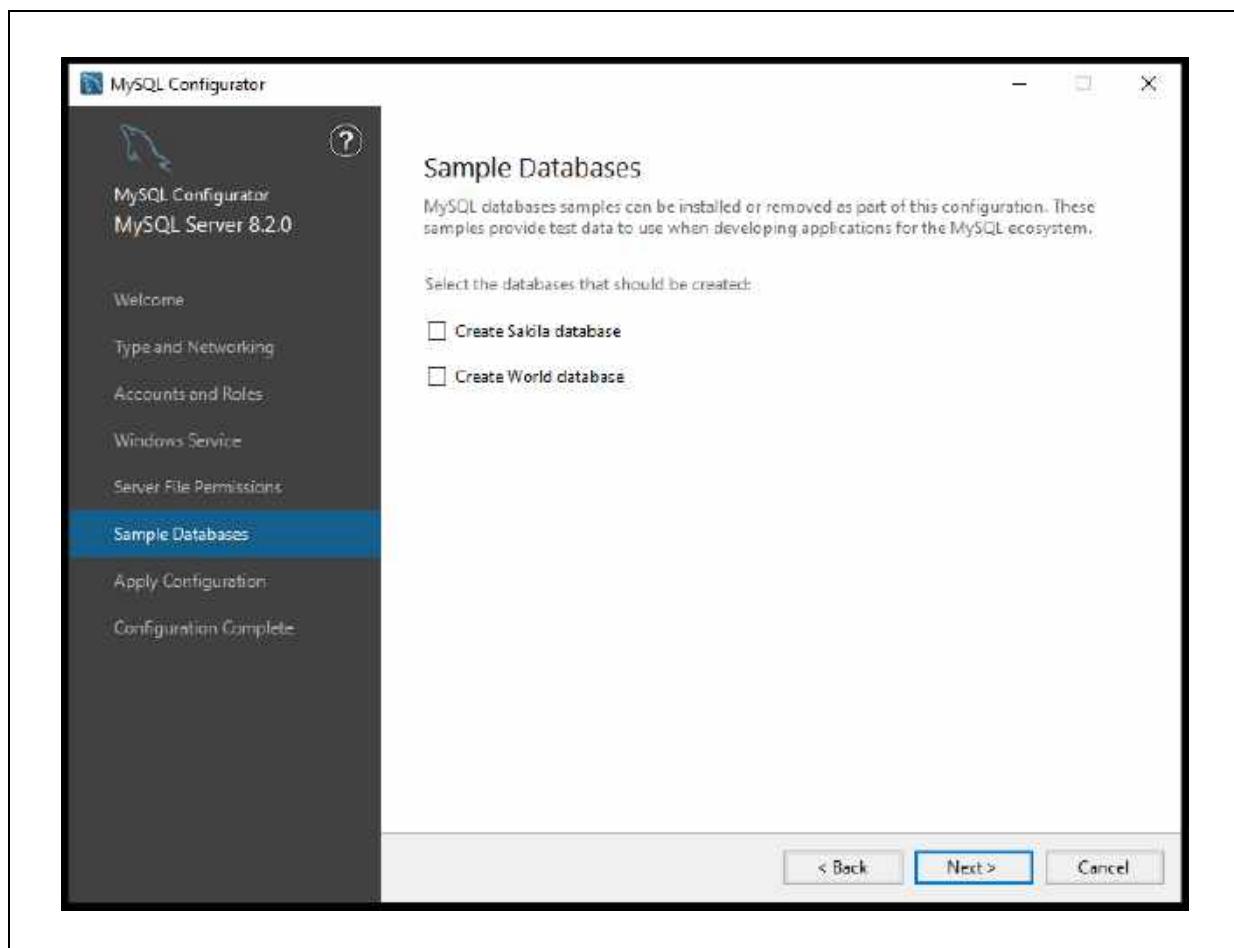
## Data Science – MySql



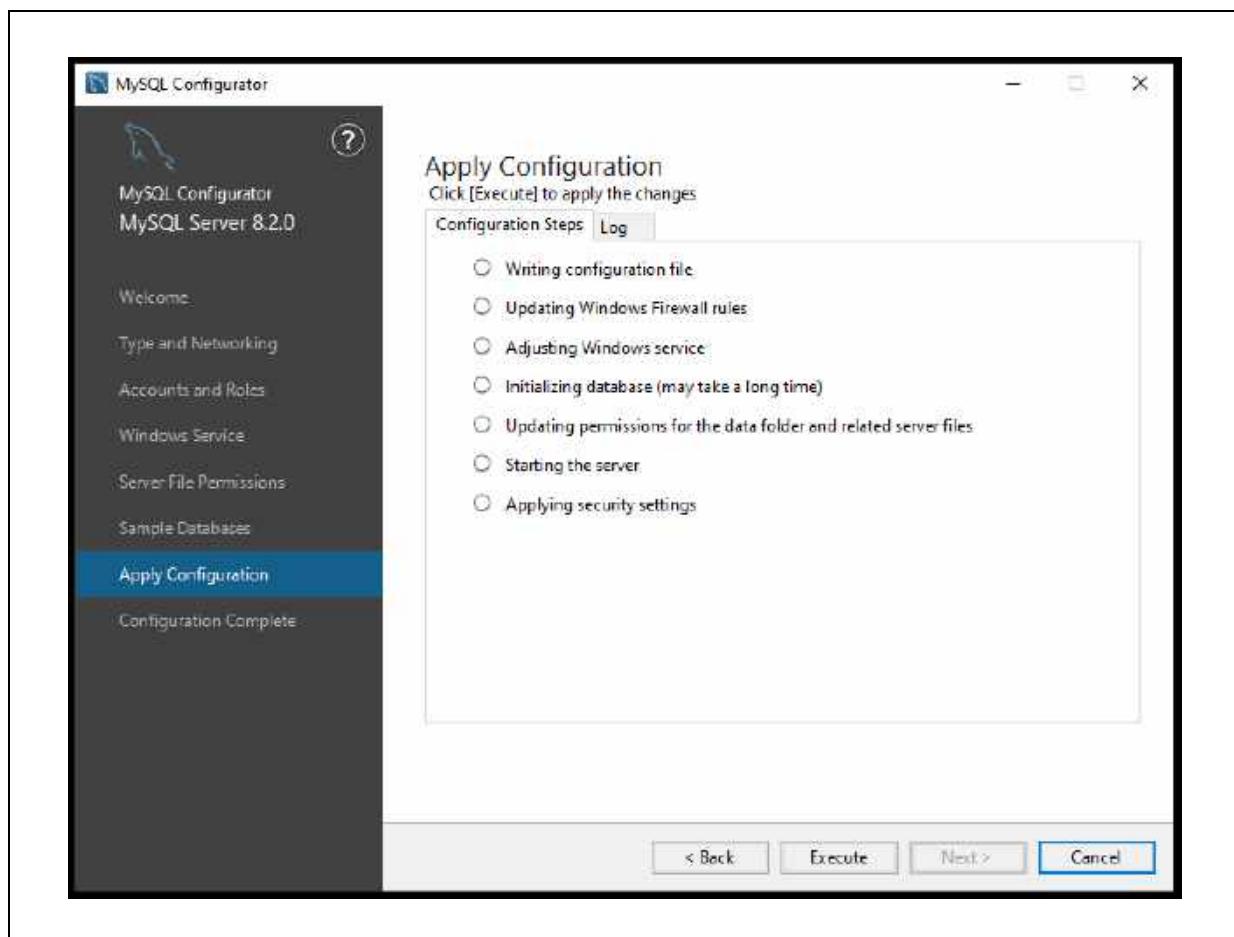
## Data Science – MySql



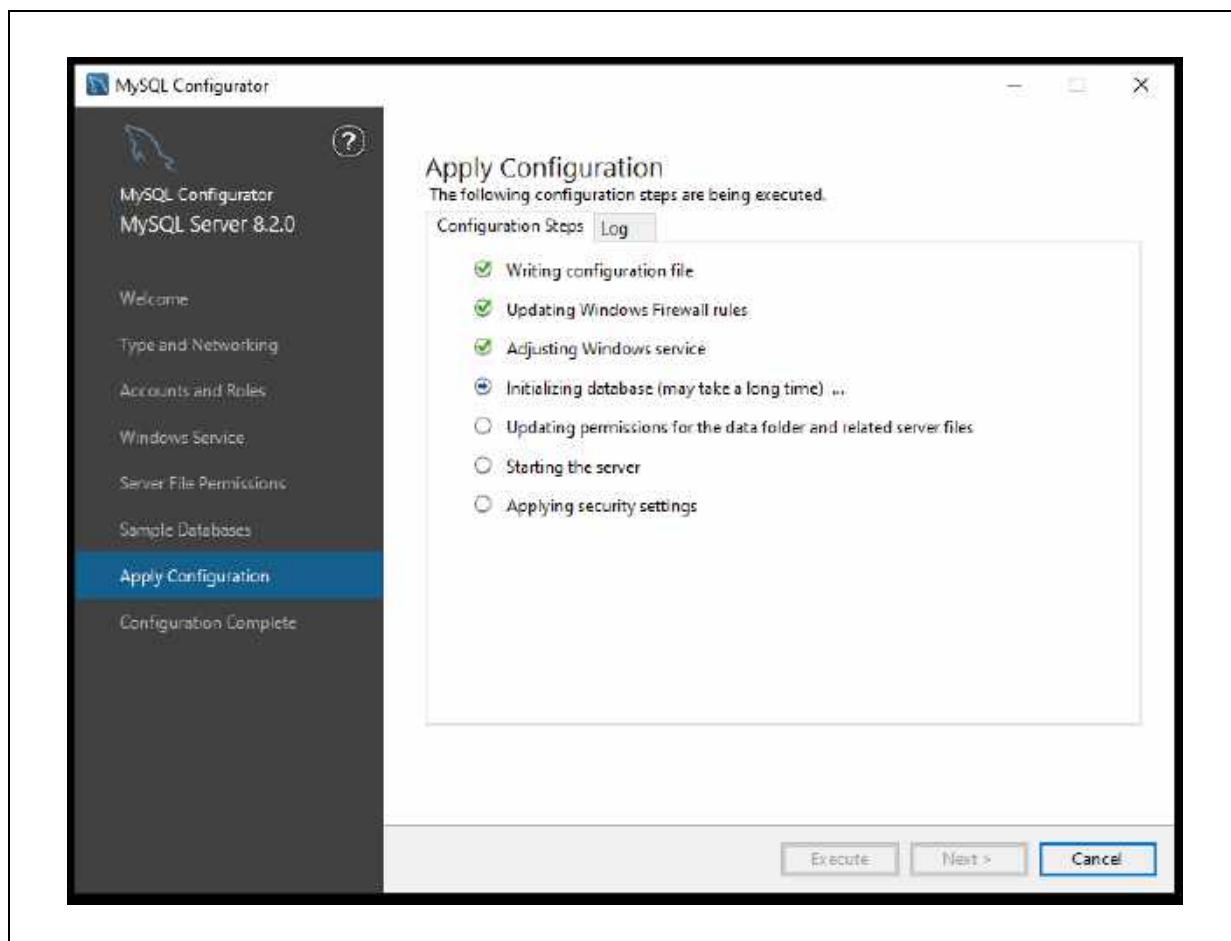
## Data Science – MySql



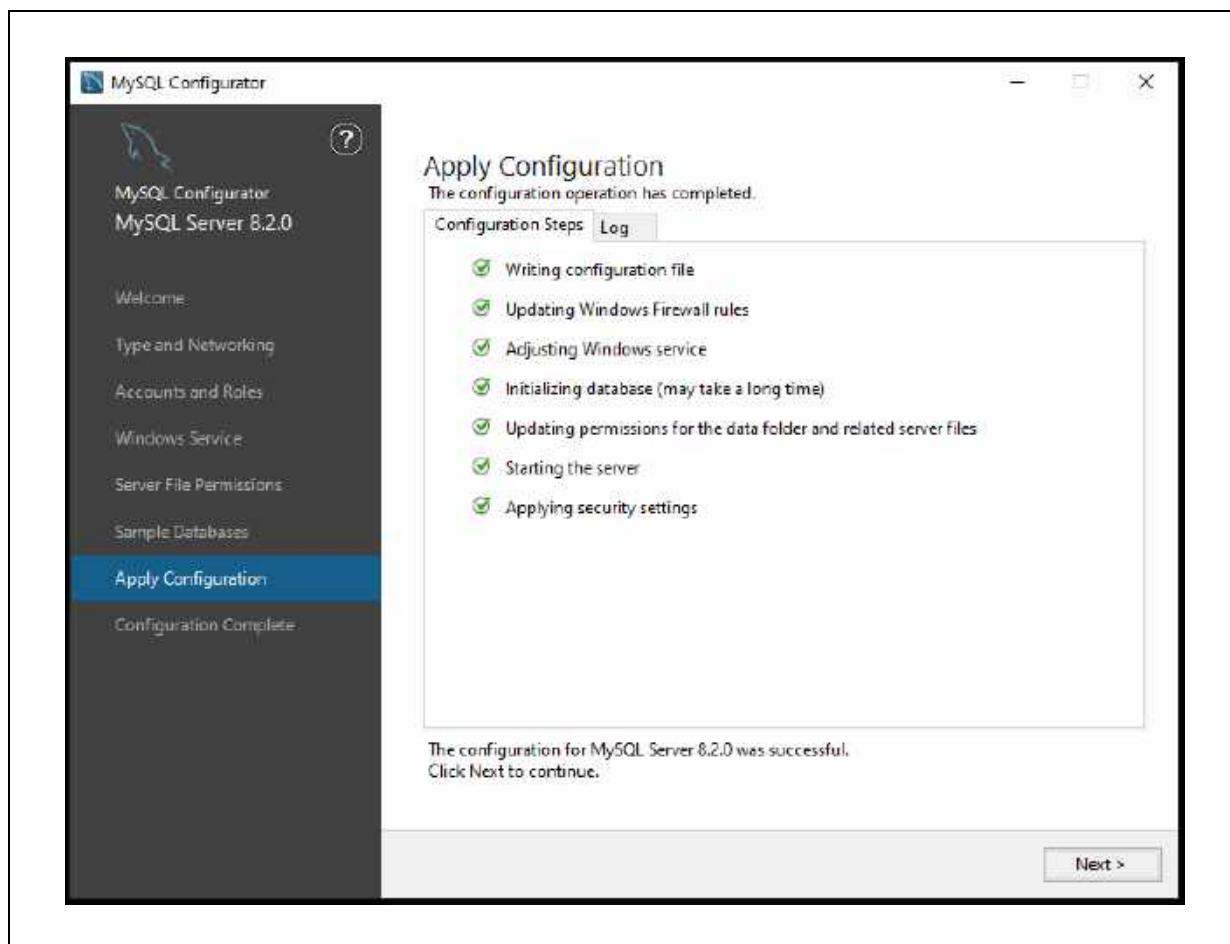
## Data Science – MySql



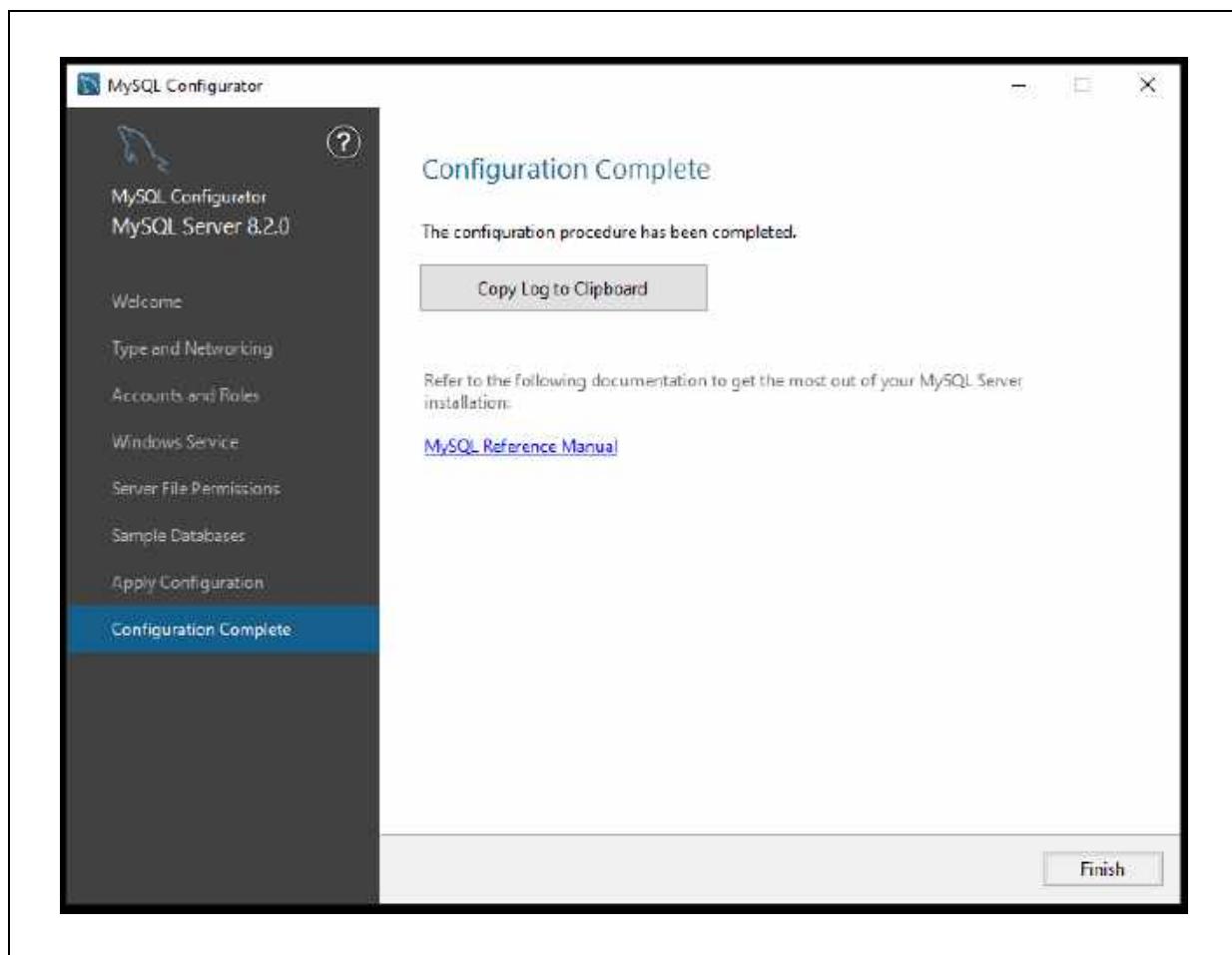
## Data Science – MySql



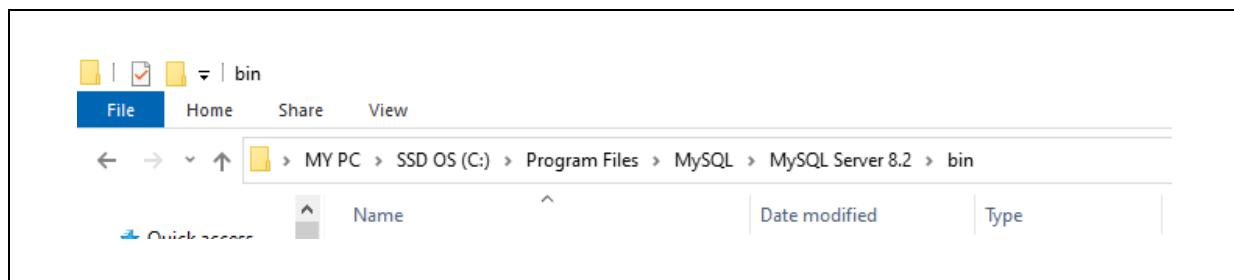
## Data Science – MySql



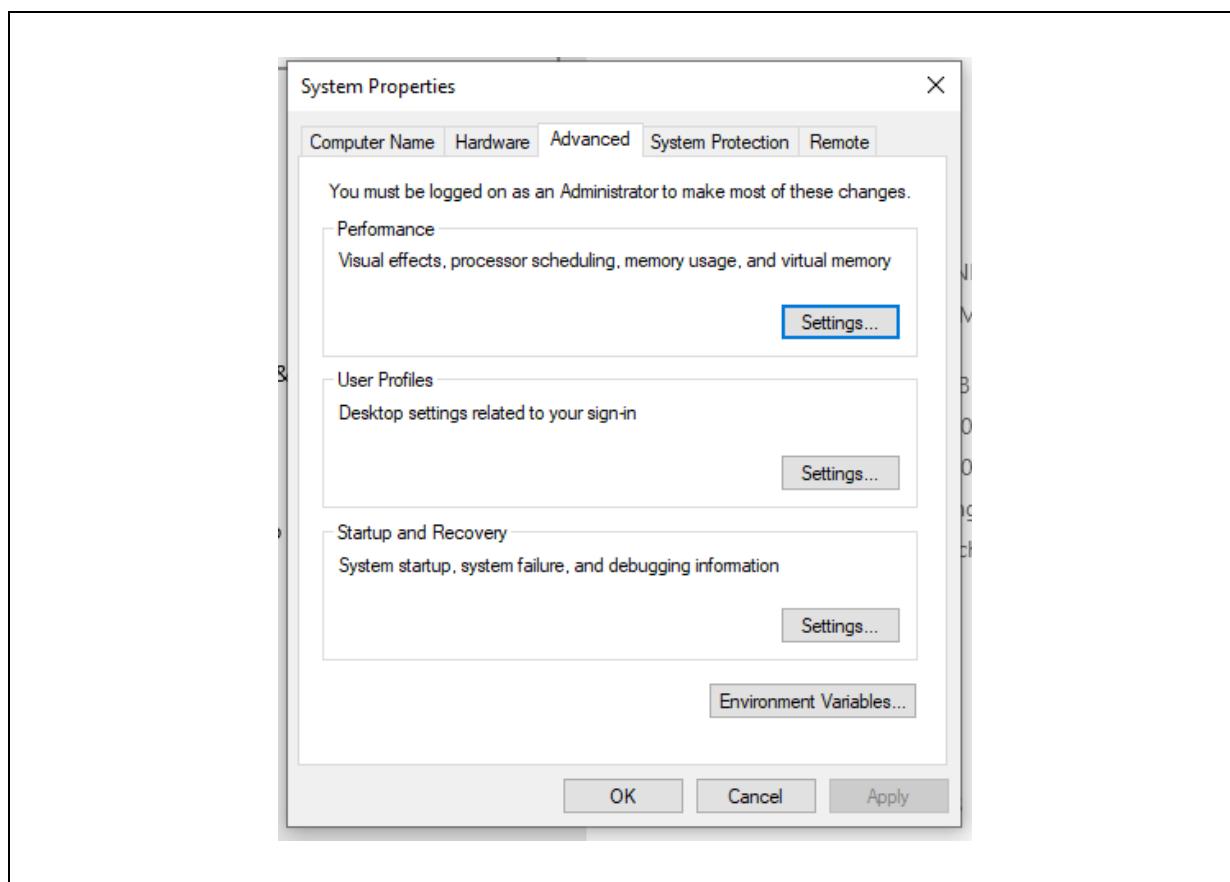
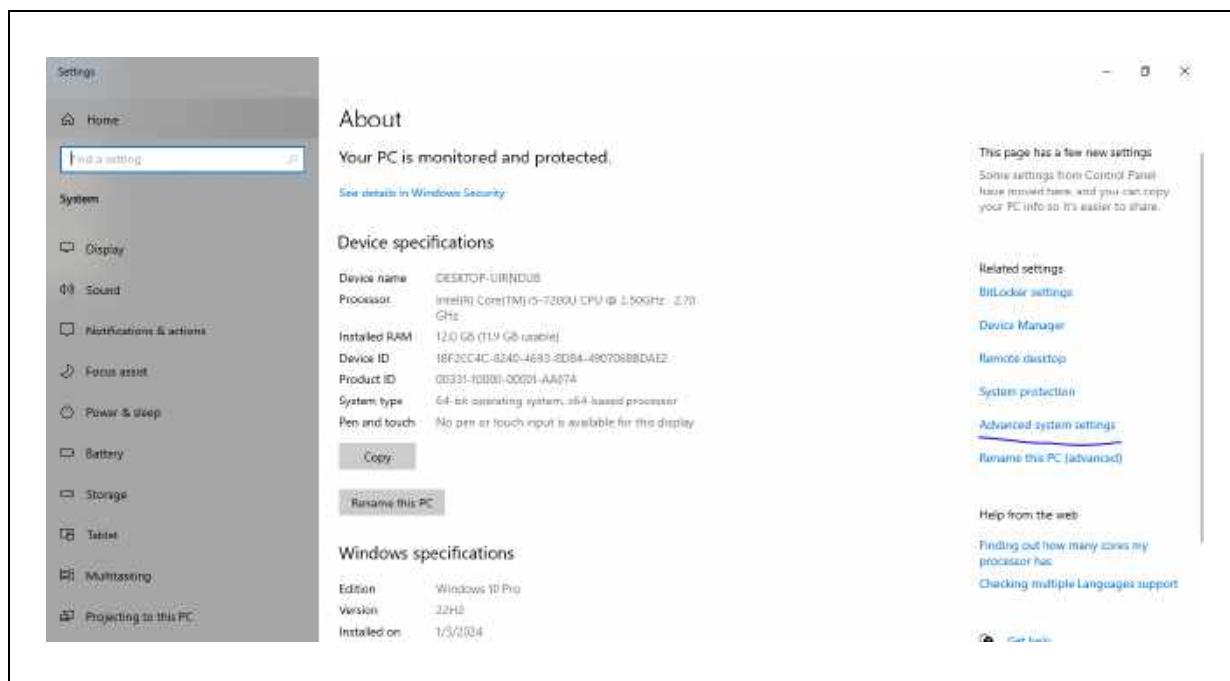
## Data Science – MySql

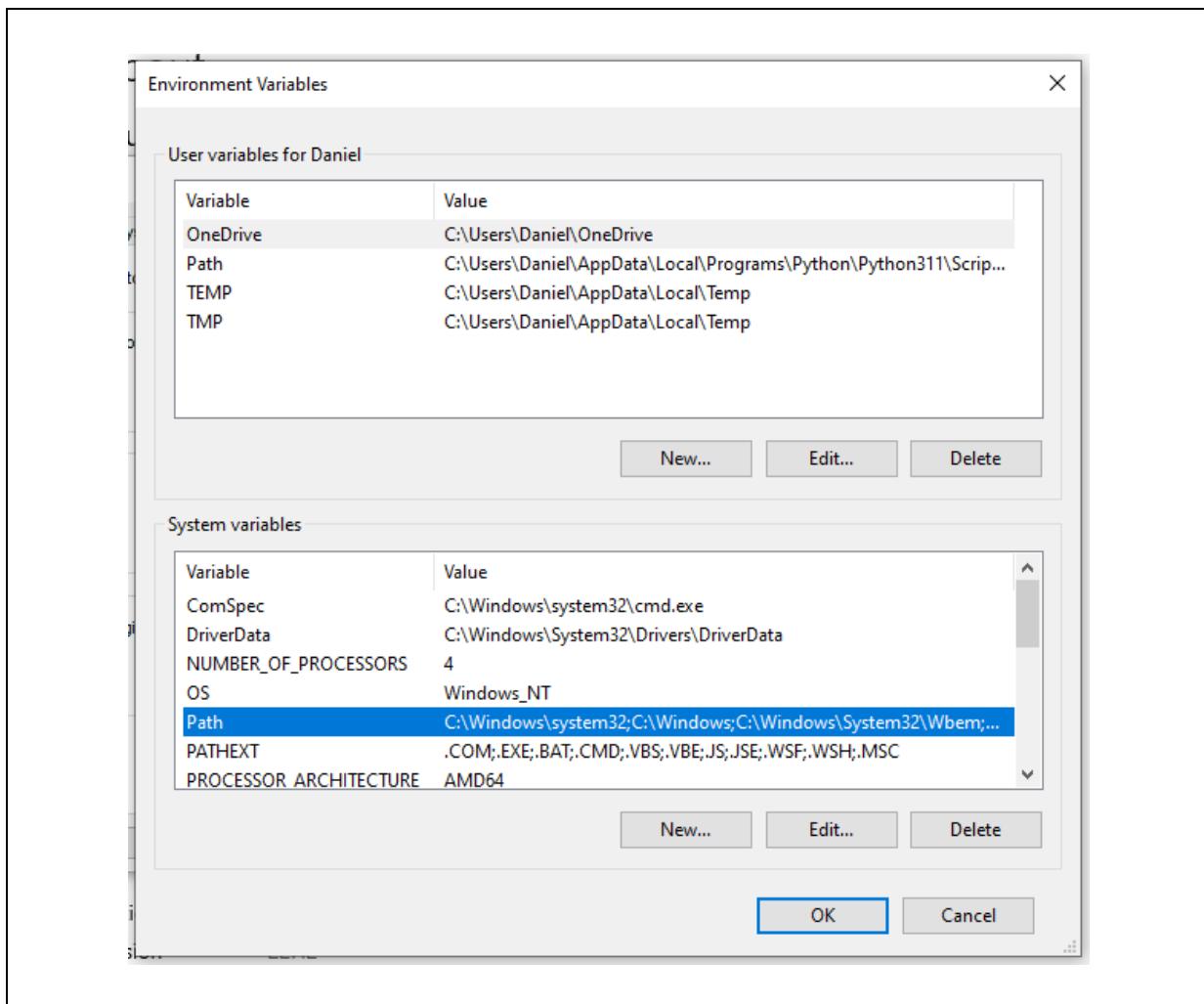


### Setting class path

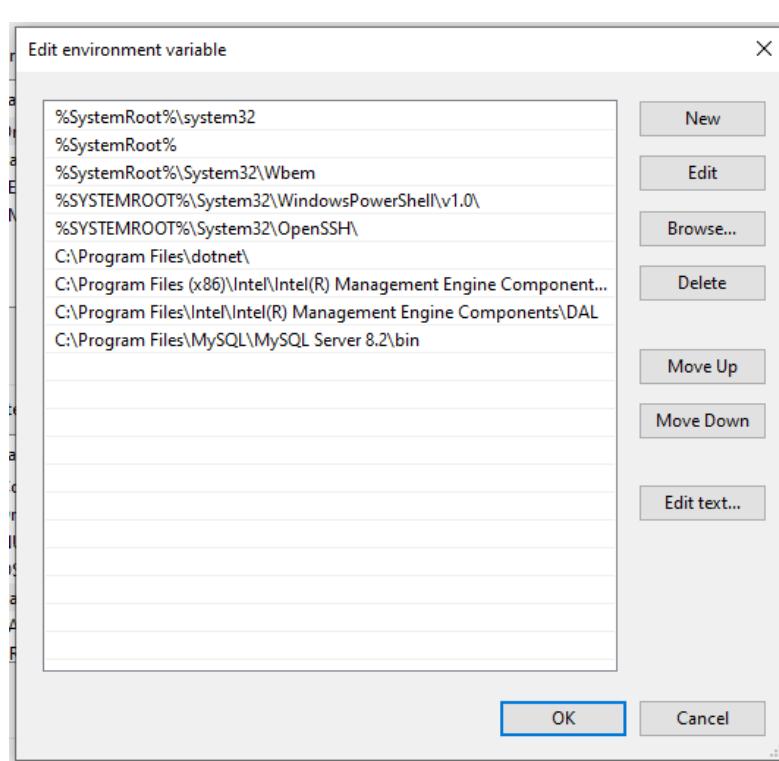
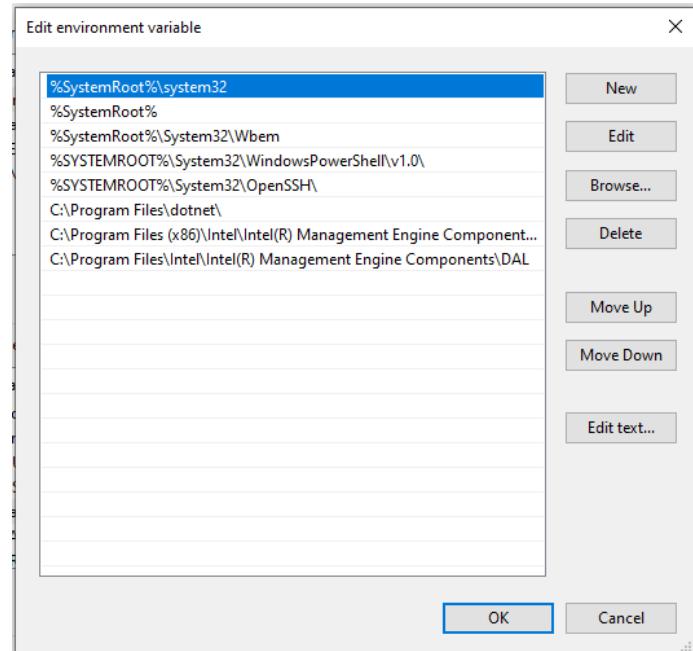


## Data Science – MySql





## Data Science – MySql



# Data Science – MySql - Introduction

---

## 1. MySql – Introduction

### Contents

1. MySQL.....	2
2. SQL.....	2
3. Database.....	2
4. DBMS .....	2
5. RDBMS .....	2
6. Query .....	3
7. SQL Usage .....	3
8. Important SQL Commands.....	4

### 1. MySql – Introduction

#### 1. MySql

- ✓ MySQL is a relational database management system
- ✓ This is an open-source
- ✓ We can use for both small and large applications
- ✓ It is very fast, reliable, scalable, and easy to use
- ✓ It was first released in 1995
- ✓ MySQL is developed, distributed, and supported by Oracle Corporation
- ✓ MySQL is software, but SQL is a database language.

#### 2. SQL

- ✓ SQL stands for Structured Query Language.
- ✓ SQL is a standard language for accessing and manipulating databases.

#### 3. Database

- ✓ A Database is an organized collection of structured data stored in a computer.
- ✓ A database is usually controlled by a Database Management System (DBMS).

#### 4. DBMS

- ✓ Data Base Management System is a software which is used to manage the database.

#### 5. RDBMS

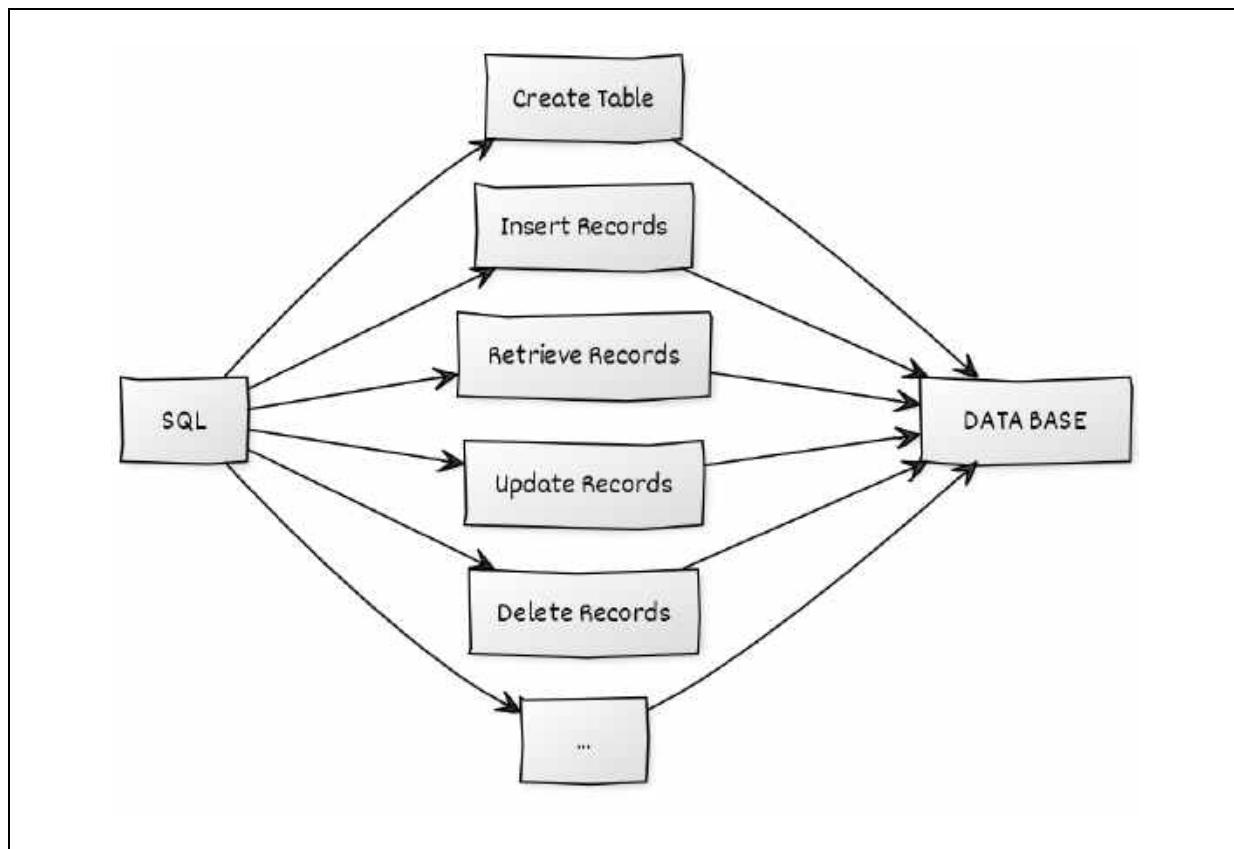
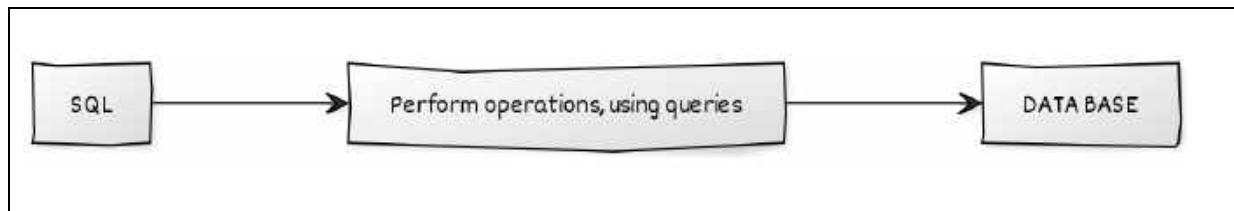
- ✓ Relational Data Base Management System is a software which is used to manage the database.
- ✓ RDBMS is the basis for SQL, and for all modern database systems such as Oracle, MySql, IBM DB2 & etc.
- ✓ In RDBMS the data used to be stored in the form of tables.
  - Table contains rows and columns.

## Data Science – MySql - Introduction

### 6. Query

- ✓ A query is a request for data from a database table(s)

### 7. SQL Usage



### 8. Important SQL Commands

- |                   |                                     |
|-------------------|-------------------------------------|
| ✓ CREATE DATABASE | - Creates a new database.           |
| ✓ CREATE TABLE    | - Creates a new table.              |
| ✓ INSERT INTO     | - Inserts new data into a database. |
| ✓ SELECT          | - Extracts data from a database.    |
| ✓ UPDATE          | - Updates data in a database.       |
| ✓ DROP TABLE      | - Deletes a table.                  |
| ✓ DELETE          | - Deletes data from a database.     |
| ✓ ALTER DATABASE  | - Modifies a database etc.          |

## 2. MySql – Database

### Contents

<b>1. Database</b> .....	2
<b>2. Table Example</b> .....	2
<b>3. Login to MYSQL</b> .....	4
<b>4. Exit from MYSQL</b> .....	6
<b>5. Show Databases</b> .....	7
<b>6. Show schemas</b> .....	8
<b>7. Semicolon</b> .....	8
<b>8. SQL is not case sensitive</b> .....	9
<b>9. Create Database</b> .....	10
<b>10. Can't create existing database</b> .....	11
<b>11. Creating two databases in one line</b> .....	12
<b>12. Drop Database</b> .....	13
<b>13. Use Database</b> .....	14

**2. MySql – Database****1. Database**

- ✓ A Database having one or more tables.
- ✓ Table having data in the form of rows and column.

**2. Table Example**

- ✓ Below is the **employees** table.

<b>Number</b>	<b>Name</b>	<b>Salary</b>
101	Ranjan	10000
102	Akshay	20000
103	Daniel	30000
104	Veeru	40000

**Columns and Rows are**

✓ Columns are

- First column name is : Number
- Second column name is : Name
- Third column name is : Salary

✓ Rows are

- First row data is : 101 Ranjan 10000
- Second row data is : 102 Akshay 20000
- Third row data is : 103 Daniel 10000
- Forth row data is : 104 Veeru 10000

## Data Science – MySql - Database

### 3. Login to MYSQL

- ✓ Open a command prompt and enter below command to login into mysql.

**Login**      Login with valid credentials  
**Command**    mysql –u root -p

```
C:\Users\Nireekshan>mysql -u root -p
Enter password: *****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 27
Server version: 8.0.33 MySQL Community Server - GPL

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

### Explanation

Its successfully connected to MYSQL

## Data Science – MySql - Database

**Login** If login with invalid credentials

**Command** mysql –u root -p

```
C:\Windows\system32\cmd.exe
C:\Users\Nireekshan>mysql -u root -p
Enter password: *****
ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: YES)
C:\Users\Nireekshan>
```

### Explanation

If we enter wrong password then we will get above response

**Login** If login with invalid command

**Command** mysql –u

**Command** mysql –u root

```
C:\Users\Nireekshan>mysql -u
mysql: [ERROR] mysql: option '-u' requires an argument.

C:\Users\Nireekshan>mysql -u root
ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: NO)
```

### Explanation

Kindly use right command to login mysql

### 4. Exit from MYSQL

- ✓ We can exit from mysql by using exit command.

**Login**      Login with valid credentials  
**Query**      To display existing databases.

**mysql> exit**

#### Output

```
C:\Users\Nireekshan>mysql -u root -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 41
Server version: 8.0.33 MySQL Community Server - GPL

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> exit
Bye
```

## 5. Show Databases

- ✓ Once successfully login into mysql then we can check total existing databases by using SQL statements.
- ✓ By default, few databases existing in mysql.

**Login**      Login with valid credentials  
**Query**      To display existing databases.

**mysql> show databases;**

### Output

```
+-----+  
| Database |  
+-----+  
| information_schema |  
| mysql |  
| performance_schema |  
| sys |  
+-----+  
4 rows in set (0.00 sec)
```

## 6. Show schemas

- ✓ We can still use schemas command alternatively to check the databases.

**Login**      Login with valid credentials  
**Query**      To display existing databases.

**mysql> show schemas;**

### Output

```
+-----+  
| Database |  
+-----+  
| information_schema |  
| mysql |  
| performance_schema |  
| sys |  
+-----+  
4 rows in set (0.00 sec)
```

## 7. Semicolon

- ✓ Semicolon is the standard way to separate each SQL statement in database systems.
- ✓ By using this we can execute more than one SQL statements.

## 8. SQL is not case sensitive

- ✓ SQL keywords are not case sensitive.
- ✓ **select** is the same as **SELECT**
- ✓ Both are valid like,
  - show databases; == SHOW DATABASES;

**Login**      Login with right credentials  
**Query**      To display existing databases.

**mysql> SHOW DATABASES;**

### Output

```
+----+  
| Database |  
+----+  
| information_schema |  
| mysql |  
| performance_schema |  
| sys |  
+----+  
4 rows in set (0.00 sec)
```

## Data Science – MySql - Database

### 9. Create Database

- ✓ Once successfully login into mysql then we can create a database by using SQL statements.

**Login**      Login with valid credentials  
**Purpose**      Creating danialdb1 Database.

```
mysql> create database danialdb1;  
Query OK, 1 row affected (0.01 sec)
```

#### Output

danialdb1 database is created successfully

**Login**      Login with valid credentials  
**Purpose**      Checking all Database.

```
mysql> show databases;
```

#### Output

```
+-----+  
| Database |  
+-----+  
| danialdb1 |  
| information_schema |  
| mysql |  
| performance_schema |  
| sys |  
+-----+  
5 rows in set (0.00 sec)
```

## 10. Can't create existing database

- ✓ We should not create existing database.
- ✓ If we are creating existing database then we will get an error like,
  - **ERROR 1007 (HY000):** Can't create database 'danieldb1'; database exists.

**Login**      Login with valid credentials  
**Purpose**      Creating existing Database.

```
mysql> create database danieldb1;  
ERROR 1007 (HY000): Can't create database 'danieldb1'; database exists
```

### Output

```
mysql> create database danieldb1;  
ERROR 1007 (HY000): Can't create database 'danieldb1'; database exists
```

## 11. Creating two databases in one line

- ✓ We can create two databases in one line.
- ✓ We can run two SQL queries by separating semi colon symbol.

**Login**      Login with valid credentials  
**Purpose**      Creating existing Database.

```
mysql> create database danieldb2;create database danieldb3;
```

### Output

```
mysql> create database danieldb2;create database danieldb3;
Query OK, 1 row affected (0.00 sec)

Query OK, 1 row affected (0.00 sec)
```

**Login**      Login with valid credentials  
**Purpose**      Creating existing Database.

```
mysql> show databases;
```

### Output

```
+-----+
| Database           |
+-----+
| danieldb1          |
| danieldb2          |
| danieldb3          |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+-----+
7 rows in set (0.00 sec)
```

## 12. Drop Database

- ✓ We can delete or drop existing database.

**Login**      Login with valid credentials  
**Purpose**      Creating Database.

```
mysql> drop database danieldb3;  
Query OK, 1 row affected (0.01 sec)
```

### Output

```
+-----+  
| Database |  
+-----+  
| danieldb1 |  
| danieldb2 |  
| information_schema |  
| mysql |  
| performance_schema |  
| sys |  
+-----+  
6 rows in set (0.00 sec)
```

### 13. Use Database

- ✓ Once successfully created database then we can use the database.

**Login**      Login with valid credentials  
**Purpose**      Use the created Database.

```
mysql> use danieldb1;  
Database changed
```

**Output**

```
mysql> use danieldb1;  
Database changed
```

## Data Science – MySql – Data types

---

### 3. MySql – Data types

#### Contents

<b>1. Data.....</b>	<b>2</b>
<b>2. Data type .....</b>	<b>2</b>
<b>3. String data types .....</b>	<b>3</b>
<b>4. Numeric data types .....</b>	<b>4</b>
<b>5. Date and Time data types .....</b>	<b>5</b>

**3. MySql – Data types****1. Data**

- ✓ Data is collection of facts.
- ✓ Facts can be values like numbers, strings, alphanumeric, symbols.
- ✓ So, every value having different data type

**2. Data type**

- ✓ We can create a table in database.
- ✓ Table having data in the form of rows and column.
- ✓ Each column in a table is required to have a **name** and a **data type**.
- ✓ Every column can hold the different type of data like, integer, float, character, strings, date and time etc.

### 3. String data types

- ✓ A group of characters is called as a String.
- ✓ To represent group of characters, String data type helps.

Data Type	Description
✓ CHAR	<ul style="list-style-type: none"> <li>✓ It can store group or characters in the form of letters, numbers, and special characters as well.</li> <li>✓ The column lenght is 0 to 255.</li> </ul>
✓ VARCHAR	<ul style="list-style-type: none"> <li>✓ It can store group or characters in the form of letters, numbers, and special characters as well.</li> <li>✓ The column lenght is 0 to 65535.</li> </ul>
<ul style="list-style-type: none"> <li>✓ We do have other string data types but not much important as part of our sessions. Thanks for understanding.</li> </ul>	

## Data Science – MySql – Data types

### 4. Numeric data types

- ✓ There are two types of numeric data types.

- Int/integer : Value without decimal
- Double : Value with decimal

Data Type	Description
✓ INT	<ul style="list-style-type: none"><li>✓ A number without decimal values.</li><li>✓ The column lenght is from - 2147483648 to + 2147483647</li></ul>
✓ FLOAT	<ul style="list-style-type: none"><li>✓ A number with decimal values.</li></ul>
<ul style="list-style-type: none"><li>✓ We do have other numeric data types but not much important as part of our sessions.</li></ul>	

## Data Science – MySql – Data types

### 5. Date and Time data types

- ✓ By using these data types we can store Date and time values in table columns.

Data Type	Description
✓ DATE	<ul style="list-style-type: none"><li>✓ A date. Format: YYYY-MM-DD.</li><li>✓ The supported range is from '1000-01-01' to '9999-12-31'</li></ul>
✓ DATETIME	<ul style="list-style-type: none"><li>✓ A date and time combination. Format: YYYY-MM-DD hh:mm:ss.</li><li>✓ The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'.</li></ul>

## Data Science – MySql – DDL and DML

---

### 4. MySql – DDL and DML

#### Contents

1. DDL.....	2
2. DML.....	2

## Data Science – MySql – DDL and DML

---

### 4. MySql – DDL and DML

#### 1. DDL

- ✓ The full form of DDL is **Data Definition Language**.
- ✓ By using this we can create the database structure or schema.
- ✓ DDL command are like,
  - CREATE
  - ALTER
  - DROP
  - TRUNCATE
  - COMMENT
  - RENAME, etc.
- ✓ DDL commands are auto-committed. So, the changes are saved in the database permanently.
- ✓ DDL statements affect the whole table.

#### 2. DML

- ✓ The full form of DDL is **Data Manipulation Language**.
- ✓ By using this we can change the data which is stored in the database.
- ✓ DML command are like,
  - INSERT
  - UPDATE
  - DELETE
  - MERGE & etc.
- ✓ DML commands used to populate and manipulate database
- ✓ It adds or updates the row of the table
- ✓ DML effects one or more rows.

# Data Science – MySql – Table

---

## 5. MySql – Table

### Contents

<b>1. Table</b> .....	2
<b>2. Table Example</b> .....	2
<b>3. Create a table</b> .....	3
<b>4. Show tables</b> .....	4
<b>5. Drop table</b> .....	5
<b>6. Rename the table</b> .....	7
<b>7. Insert data into table</b> .....	10
<b>8. Select the table</b> .....	12
8.1. Select few columns from the table .....	13
<b>9. Select Distinct</b> .....	15
<b>10. Alter table</b> .....	16
10.1. Add column in table .....	17
10.2. Drop column from table .....	18
10.3. Modify column name in table .....	19

## Data Science – MySql – Table

### 5. MySql – Table

#### 1. Table

- ✓ We can create table in mysql.
- ✓ The purpose of creating table to store the data.
- ✓ Table having group of rows and columns.
- ✓ Below is the **employees** table.

#### 2. Table Example

- ✓ Below is the **employees** table.

Number	Name	Salary
101	Ranjan	10000
102	Akshay	20000
103	Daniel	30000
104	Veeru	40000

## Data Science – MySql – Table

### 3. Create a table

- ✓ We can **create a table** by using create table command
- ✓ While creating table we need to specify the name of the table.
- ✓ The table name should be unique in a database.
- ✓ We can use **IF NOT EXISTS** command (optional) while creating table.
- ✓ This command checks if the table exists in the database or not, if table exists then MySQL ignore to create new table.

**Login**      Login with valid credentials  
**Query**      To create table in danialdb1 database.

```
mysql> create table Persons(
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

#### Output

```
mysql> CREATE TABLE Persons(
    ->     PersonID int,
    ->     LastName varchar(255),
    ->     FirstName varchar(255),
    ->     Address varchar(255),
    ->     City varchar(255)
    -> );
Query OK, 0 rows affected (0.02 sec)
```

#### Explanation

- ✓ The PersonID column is of type int.
- ✓ The LastName, FirstName, Address, and City columns are varchar type and the maximum length for these fields is 255 characters.
- ✓ Currently **Persons** table is **empty** table

## Data Science – MySql – Table

### 4. Show tables

- ✓ We can check created tables in database by using show tables command.

**Login**      Login with valid credentials  
**Query**      To display the tables in danialdb1 database.

**mysql> show tables;**

#### Output

```
+-----+  
| Tables_in_danialdb1 |  
+-----+  
| persons           |  
+-----+  
1 row in set (0.00 sec)
```

## Data Science – MySql – Table

### 5. Drop table

- ✓ We can drop/delete a table by using drop table command.
- ✓ Once we deleted the table then, that table is not available in database.
- ✓ Let's create a dummy table like Persons123 and will drop the same.

**Login**      Login with valid credentials  
**Query**      To create table in danialdb1 database.

```
mysql> create table Persons123(  
    PersonID int,  
    LastName varchar(255),  
    FirstName varchar(255),  
    Address varchar(255),  
    City varchar(255)  
);
```

#### Output

```
mysql> CREATE TABLE Persons123(  
    -> PersonID int,  
    -> LastName varchar(255),  
    -> FirstName varchar(255),  
    -> Address varchar(255),  
    -> City varchar(255)  
    -> );  
Query OK, 0 rows affected (0.02 sec)
```

## Data Science – MySql – Table

**Login**      Login with valid credentials  
**Query**      Display all tables in danialdb1 database.

```
mysql> show tables;
```

### Output

```
+-----+  
| Tables_in_danialdb1 |  
+-----+  
| persons           |  
| persons123        |  
+-----+  
2 rows in set (0.00 sec)
```

**Login**      Login with valid credentials  
**Query**      To drop Persons123 table from danialdb1 database.

```
mysql> drop table Persons123;  
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> show tables;
```

### Output

```
+-----+  
| Tables_in_danialdb1 |  
+-----+  
| persons           |  
+-----+  
1 row in set (0.00 sec)
```

## Data Science – MySql – Table

### 6. Rename the table

- ✓ We can rename the existing table by using rename or alter commands.
- ✓ Let's create a dummy table like Persons111 and will rename the same.

**Login**

Login with valid credentials

**Query**

To create Persons111 table in danialdb1 database.

```
mysql> create table Persons111(
    PersonID int,
    LastName varchar(255)
);
```

**Output**

```
mysql> CREATE TABLE Persons111(
    ->     PersonID int,
    ->     LastName varchar(255)
    -> );
Query OK, 0 rows affected (0.01 sec)
```

**Login**

Login with valid credentials

**Query**

Display all tables in danialdb1 database.

```
mysql> show tables;
```

**Output**

```
+-----+
| Tables_in_danialdb1 |
+-----+
| persons           |
| persons111        |
+-----+
2 rows in set (0.00 sec)
```

## Data Science – MySql – Table

**Login**

Login with valid credentials

**Query**

Rename table and display all tables in danialdb1 database.

```
mysql> rename table Persons111 to Persons222;  
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> show tables;
```

**Output**

```
+-----+  
| Tables_in_danialdb1 |  
+-----+  
| persons           |  
| persons222        |  
+-----+  
2 rows in set (0.00 sec)
```

## Data Science – MySql – Table

**Login  
Query**

Login with valid credentials  
Alter table and display all tables in danialdb1 database.

```
mysql> alter table Persons222 rename Persons333;  
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> show tables;
```

**Output**

```
+-----+  
| Tables_in_danialdb1 |  
+-----+  
| persons           |  
| persons333        |  
+-----+  
2 rows in set (0.00 sec)
```

### 7. Insert data into table

- ✓ Once we created the table then we can insert data into the table.
  - We can insert data into table by specifying column names and values.
  - Without specifying column names also we can insert the data into a table. We ensure that the **order of** the values and columns should be same.

## Data Science – MySql – Table

**Login**

Login with valid credentials

**Query**

To insert data into Persons table in danialdb1 database.

```
mysql> insert into Persons(PersonID, LastName, FirstName, Address, City) values(101, 'Danial', 'K', 'Near to Data Science Area', 'Hyderabad');
```

Query OK, 1 row affected (0.00 sec)

```
mysql> insert into Persons(PersonID, LastName, FirstName, Address, City) values(102, 'Nireekshan', 'D', 'Near to AI Theatre', 'Bangalore');
```

Query OK, 1 row affected (0.00 sec)

```
mysql> insert into Persons values(103, 'Ranjan', 'M', 'Near to Python Theatre', 'Hyderabad');
```

Query OK, 1 row affected (0.00 sec)

```
mysql> insert into Persons values(104, 'Prasad', 'K', 'Near to Python Road', 'Hyderabad');
```

Query OK, 1 row affected (0.00 sec)

**Output**

Successfully data inserted into Persons table.

## Data Science – MySql – Table

### 8. Select the table

- ✓ We can see the data in a table by using select command.
- ✓ While using select command, table name and column names should be match otherwise we will get an error.

**Login  
Query**

Login with valid credentials  
To select the data from Persons table in danialdb1 database.

```
mysql> select * from Persons;
```

**Output**

PersonID	LastName	FirstName	Address	City
101	Danial	K	Near to Data Science Area	Hyderabad
102	Nireekshan	D	Near to AI Theatre	Bangalore
103	Ranjan	M	Near to Python Theatre	Hyderabad
104	Prasad	K	Near to Python Road	Hyderabad

4 rows in set (0.00 sec)

**Login  
Query**

Login with valid credentials  
Ensure that table name should exists

```
mysql> select * from Persons444;
```

**Output**

**ERROR 1146 (42S02): Table 'danialdb1.persons444' doesn't exist**

## Data Science – MySql – Table

### 8.1. Select few columns from the table

- ✓ We can see the data in a table by using select command.

**Login Query** Login with valid credentials  
To select few columns from Persons table in danialdb1 database.

```
mysql> select PersonID, LastName from Persons;
```

#### Output

```
+-----+-----+
| PersonID | LastName |
+-----+-----+
|      101 | Danial   |
|      102 | Nireekshan |
|      103 | Ranjan    |
|      104 | Prasad    |
+-----+-----+
4 rows in set (0.00 sec)
```

## Data Science – MySql – Table

---

### Login

Login with valid credentials

### Query

To select few columns from Persons table in danialdb1 database.

```
mysql> select PersonID, LastName, City from Persons;
```

### Output

PersonID	LastName	City
101	Danial	Hyderabad
102	Nireekshan	Bangalore
103	Ranjan	Hyderabad
104	Prasad	Hyderabad

4 rows in set (0.00 sec)

### Login

Login with valid credentials

### Query

Ensure that column name should exists in table

```
mysql> select PersonID, LastName, City111 from Persons;
```

### Output

ERROR 1054 (42S22): Unknown column 'city111' in 'field list'

## Data Science – MySql – Table

### 9. Select Distinct

- ✓ By default select statement will returns all values including duplicates.
- ✓ If we wanted to display unique values then we can use select distinct statement.

Login

Login with valid credentials

Query

To select city column values from Persons table

```
mysql> select city from Persons;
```

Output

```
+-----+  
| city |  
+-----+  
| Hyderabad |  
| Bangalore |  
| Hyderabad |  
| Hyderabad |  
+-----+  
4 rows in set (0.00 sec)
```

Login

Login with valid credentials

Query

To select distinct values from city column in Persons table

```
mysql> select distinct city from persons;
```

Output

```
+-----+  
| city |  
+-----+  
| Hyderabad |  
| Bangalore |  
+-----+
```

**10. Alter table**

- ✓ Once we created the table then we can apply do modifications based on requirement on existing table.
- ✓ We can add, delete, or modify columns in an existing table.
- ✓ We can add and drop various constraints on an existing table.

## Data Science – MySql – Table

### 10.1. Add column in table

- ✓ We can add column to existing table.

**Login**      Login with valid credentials  
**Query**      Add email column to Persons table

```
mysql> alter table persons
-> add email varchar(255);
```

**Output**

```
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

**Login**      Login with valid credentials  
**Query**      Display the persons table

```
mysql> select * from persons;
```

**Output**

PersonID	LastName	FirstName	Address	City	email
101	Danial	K	Near to Data Science Area	Hyderabad	NULL
102	Nireekshan	D	Near to AI Theatre	Bangalore	NULL
103	Ranjan	M	Near to Python Theatre	Hyderabad	NULL
104	Prasad	K	Near to Python Road	Hyderabad	NULL

4 rows in set (0.00 sec)

## Data Science – MySql – Table

### 10.2. Drop column from table

- ✓ Based on requirement we can drop column from table.

**Login**      Login with valid credentials  
**Query**      Drop email column from Persons table

```
mysql> alter table persons
-> drop column email;
```

**Output**

```
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

**Login**      Login with valid credentials  
**Query**      Display persons table.

```
mysql> select * from persons;
```

**Output**

PersonID	LastName	FirstName	Address	City
101	Danial	K	Near to Data Science Area	Hyderabad
102	Nireekshan	D	Near to AI Theatre	Bangalore
103	Ranjan	M	Near to Python Theatre	Hyderabad
104	Prasad	K	Near to Python Road	Hyderabad

4 rows in set (0.00 sec)

## Data Science – MySql – Table

### 10.3. Modify column name in table

- ✓ We can modify column name in existing table.

**Login**      Login with valid credentials  
**Query**      Modify city column to location in persons table

```
mysql> alter table persons
-> rename column city to location;
```

**Output**

```
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

**Login**      Login with valid credentials  
**Query**      Display persons table.

```
mysql> select * from persons;
```

**Output**

PersonID	LastName	FirstName	Address	location
101	Danial	K	Near to Data Science Area	Hyderabad
102	Nireekshan	D	Near to AI Theatre	Bangalore
103	Ranjan	M	Near to Python Theatre	Hyderabad
104	Prasad	K	Near to Python Road	Hyderabad

4 rows in set (0.00 sec)

# Data Science – MySql - Where clause, Operators

---

## 6. MySql – Where clause, Operators

### Contents

<b>1. Where clause .....</b>	2
<b>2. Where clause with operators.....</b>	4
2.1. Where clause with greater than operator .....	5
2.2. Where clause with between and operators .....	6
2.3. Where clause with like operator.....	7
2.4. Where clause with in operator .....	8
<b>3. Where clause with AND, OR, NOT operators .....</b>	10
3.1. Where clause with and operator .....	10
3.2. Where clause with or operator.....	11
3.3. Where clause with not operator.....	12

## Data Science – MySql - Where clause, Operators

### 6. MySql – Where clause, Operators

#### 1. Where clause

- ✓ Based on condition we can filter and get matched rows from table.

Login  
Query

Login with valid credentials  
Get matching rows by using where clause.

```
mysql> select * from persons  
-> where LastName = 'Danial';
```

Output

```
+-----+-----+-----+-----+  
| PersonID | LastName | FirstName | Address | location |  
+-----+-----+-----+-----+  
|      101 | Danial   | K        | Near to Data Science Area | Hyderabad |  
+-----+-----+-----+-----+  
1 row in set (0.00 sec)
```

## Data Science – MySql - Where clause, Operators

**Login** Login with valid credentials  
**Query** where clause with equal operator

```
mysql> select * from persons
-> where location = 'hyderabad';
```

**Output**

PersonID	LastName	FirstName	Address	location
101	Danial	K	Near to Data Science Area	Hyderabad
103	Ranjan	M	Near to Python Theatre	Hyderabad
104	Prasad	K	Near to Python Road	Hyderabad

3 rows in set (0.00 sec)

## Data Science – MySql - Where clause, Operators

---

### 2. Where clause with operators

- ✓ Below operators we can use with where clause.

Operator	Description
=	Equal
>	Greater than
<	Less than
>=	Greater than or Equal
<=	Less than or Equal
<>	Not Equal, as !=
between	Between a certain range
in	To check multiple possible values in column

## Data Science – MySql - Where clause, Operators

### 2.1. Where clause with greater than operator

- ✓ We can apply where clause with greater than operator on specific column.
- ✓ This returns the matched records from the table.

**Login**      Login with valid credentials  
**Query**      where clause with greater than operator

```
mysql> select * from persons  
-> where PersonID > 101;
```

#### Output

PersonID	LastName	FirstName	Address	location
102	Nireekshan	D	Near to AI Theatre	Bangalore
103	Ranjan	M	Near to Python Theatre	Hyderabad
104	Prasad	K	Near to Python Road	Hyderabad

3 rows in set (0.00 sec)

## Data Science – MySql - Where clause, Operators

### 2.2. Where clause with between and operators

- ✓ We can apply where clause with between, and operators.
- ✓ This returns within range of values.
- ✓ Here start and end values are included.

**Login**      Login with valid credentials  
**Query**      where clause with between and operators

```
mysql> select * from persons
-> where PersonID between 101 and 104;
```

#### Output

PersonID	LastName	FirstName	Address	location
101	Danial	K	Near to Data Science Area	Hyderabad
102	Nireekshan	D	Near to AI Theatre	Bangalore
103	Ranjan	M	Near to Python Theatre	Hyderabad
104	Prasad	K	Near to Python Road	Hyderabad

4 rows in set (0.00 sec)

## Data Science – MySql - Where clause, Operators

### 2.3. Where clause with like operator

- ✓ We can apply where clause with like operators.
- ✓ By using this we can search specific pattern in a column
  - 'h%' it returns starts with h values
  - '%h' it returns ends with h values

**Login Query** Login with valid credentials  
where clause with like operator

```
mysql> select * from persons
-> where location like 'h%';
```

**Output**

PersonID	LastName	FirstName	Address	location
101	Danial	K	Near to Data Science Area	Hyderabad
103	Ranjan	M	Near to Python Theatre	Hyderabad
104	Prasad	K	Near to Python Road	Hyderabad

3 rows in set (0.00 sec)

## Data Science – MySql - Where clause, Operators

### 2.4. Where clause with in operator

- ✓ We can apply where clause with in operator.
- ✓ By using this we can get the matched records from the table.

**Login Query** Login with valid credentials  
where clause with in operator

```
mysql> select * from persons
-> where Lastname in('Nireekshan', 'Danial');
```

**Output**

PersonID	Lastname	FirstName	Address	location
101	Danial	K	Near to Data Science Area	Hyderabad
102	Nireekshan	D	Near to AI Theatre	Bangalore

2 rows in set (0.00 sec)

## Data Science – MySql - Where clause, Operators

**Login Query** Login with valid credentials  
where clause with not in operator

```
mysql> select * from persons
-> where Lastname not in('Nireekshan', 'Danial');
```

**Output**

PersonID	Lastname	FirstName	Address	location
103	Ranjan	M	Near to Python Theatre	Hyderabad
104	Prasad	K	Near to Python Road	Hyderabad

2 rows in set (0.00 sec)

### 3. Where clause with AND, OR, NOT operators

- ✓ We can use below operators along with where clause
  - And operator
  - Or operators
  - Not operator

#### 3.1. Where clause with and operator

- ✓ If both conditions are true then AND operator returns the matched records.

**Login Query** Login with valid credentials  
where clause with and operator

```
mysql> select * from Persons
-> where LastName = 'Danial' and location = 'Hyderabad';
```

#### Output

PersonID	LastName	FirstName	Address	location
101	Danial	K	Near to Data Science Area	Hyderabad

1 row in set (0.00 sec)

## Data Science – MySql - Where clause, Operators

### 3.2. Where clause with or operator

- ✓ If at least one condition is matching from multiple conditions then or operator returns the corresponding records

**Login Query** Login with valid credentials  
where clause with and operator

```
mysql> select * from Persons  
-> where LastName = 'Prasad' or location = 'UK';
```

#### Output

PersonID	LastName	FirstName	Address	location
104	Prasad	K	Near to Python Road	Hyderabad

1 row in set (0.00 sec)

## Data Science – MySql - Where clause, Operators

### 3.3. Where clause with not operator

- ✓ Not operator displays the records if the condition is not true.

**Login Query** Login with valid credentials  
where clause with and operator

```
mysql> select * from Persons
-> where not location = 'Bangalore';
```

**Output**

```
+-----+-----+-----+-----+
| PersonID | LastName | FirstName | Address           | location |
+-----+-----+-----+-----+
|     101 | Danial   | K         | Near to Data Science Area | Hyderabad |
|     103 | Ranjan    | M         | Near to Python Theatre  | Hyderabad |
|     104 | Prasad    | K         | Near to Python Road    | Hyderabad |
+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

# Data Science – MySql - Constraints

---

## 7. MYSQL – Constraints

### Contents

<b>1. Constraints.....</b>	<b>2</b>
<b>2. Constraints table .....</b>	<b>2</b>
<b>3. Not null constraint .....</b>	<b>3</b>
<b>4. Primary key constraint .....</b>	<b>6</b>
<b>5. Check constraint.....</b>	<b>8</b>

## 7. MySql – Constraints

### 1. Constraints

- ✓ SQL constraints are used to specify **rules** for data in a table.
- ✓ We can specify constraints during table creation of the table or after table created with alter table.
- ✓ By using this we can restricts the type of the data to the column.
- ✓ This ensures the accuracy and reliability of the data in the table.
- ✓ We can apply constraints on column and table level as well.

### 2. Constraints table

Name	Description
✓ Not null	✓ Ensures that a column cannot have a NULL value
✓ Primary key	✓ Uniquely identifies each row in a table
✓ Check	✓ Ensures that the values in a column satisfies a specific condition

## Data Science – MySql - Constraints

### 3. Not null constraint

- ✓ By default, a column values can store null values.
- ✓ If we apply not null constraint over column then columns will not allow null values.
- ✓ It means, columns should have a value instead of null.

Login

Query

Login with valid credentials

Creating a table

```
mysql> create table Persons10(  
      id int,  
      lastname varchar(255),  
      firstname varchar(255),  
      age int  
);
```

Output

Successfully table created

## Data Science – MySql - Constraints

**Login** Login with valid credentials  
**Query** Inserting data into table

```
mysql> insert into Persons10(id, lastname, firstname, age)
      values(NULL, 'daniel', "", 16);
```

**Output**

```
mysql> select * from persons10;
+----+-----+-----+----+
| id | lastname | firstname | age |
+----+-----+-----+----+
| NULL | daniel |          |   16 |
+----+-----+-----+----+
1 row in set (0.00 sec)
```

**Login** Login with valid credentials  
**Query** not null constraint

```
mysql> create table Persons11(
      id int not null,
      lastname varchar(255) not null,
      firstname varchar(255) not null,
      age int
    );
```

**Output**

Successfully table created

## Data Science – MySql - Constraints

---

**Login Query** Login with valid credentials  
not null constraint

```
mysql> insert into Persons11(id, lastname, firstname, age)
      values(NULL, 'daniel', "", 16);
```

**Output**

```
ERROR 1048 (23000): Column 'id' cannot be null
```

## Data Science – MySql - Constraints

### 4. Primary key constraint

- ✓ The primary key constraint uniquely identifies each record in a table.
- ✓ Primary keys must contain UNIQUE values, and cannot contain null values.

**Login**      Login with valid credentials  
**Query**      Primary key constraint

```
mysql> create table Persons12(
           id int not null,
           lastname varchar(255) not null,
           firstname varchar(255) not null,
           age int,
           primary key(id)
);
```

**Output**

Successfully table created

**Login**      Login with valid credentials  
**Query**      not null constraint

```
mysql> insert into Persons12(id, lastname, firstname, age)
           values(101, 'daniel', 'K', 16);
```

**Output**

Query OK, 1 row affected (0.01 sec)

## Data Science – MySql - Constraints

**Login** Login with valid credentials  
**Query** not null constraint

```
mysql> insert into Persons12(id, lastname, firstname, age)
      values(101, 'daniel', 'K', 16);
```

**Output**

```
ERROR 1062 (23000): Duplicate entry '101' for key
'persons12.PRIMARY'
```

## Data Science – MySql - Constraints

### 5. Check constraint

- ✓ The check constraint is used to limit the value range that can be placed in a column.
- ✓ If we define a check constraint on a column it will allow only certain values for this column.

Login

Query

Login with valid credentials

check constraint

```
mysql> create table Persons13(
    id int not null,
    lastname varchar(255) not null,
    firstname varchar(255) not null,
    age int,
    check (age >= 18)
);
```

Output

Successfully table created

## Data Science – MySql - Constraints

**Login**      Login with valid credentials  
**Query**      not null constraint

```
mysql> insert into Persons13(id, lastname, firstname, age)
      values(101, 'daniel', 'K', 16);
```

**Output**

```
ERROR 3819 (HY000): Check constraint 'persons13_chk_1' is
violated
```

## Data Science – MySql – Imp Functions

---

### 8. MySql – Imp functions

#### Contents

1. Functions .....	2
2. min(column name) function .....	3
3. max(column name) function.....	4
4. count(column name) function .....	5
5. avg(column name) function .....	7
6. sum(column name) function.....	9

### 8. MySql – Imp functions

#### 1. Functions

- ✓ In mysql there are predefined functions.
- ✓ These functions helps us to finish basic requirements.

## Data Science – MySql – Imp Functions

---

### 2. min(column name) function

- ✓ The min(column) is a predefined function in mysql.
- ✓ By using this function we can get minimum value from specific column

**Login**      Login with valid credentials  
**Query**     Creating a table

```
mysql> select * from persons;
```

#### Output

```
mysql> select * from persons;
+-----+-----+-----+-----+-----+
| PersonID | LastName | FirstName | Address | location |
+-----+-----+-----+-----+-----+
| 101 | Danial | K | Near to Data Science Area | Hyderabad |
| 102 | Nireekshan | D | Near to AI Theatre | Bengaluru |
| 103 | Ranjan | M | Near to Python Theatre | Hyderabad |
| 104 | Prasad | K | Near to Python Road | Hyderabad |
+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

**Login**      Login with valid credentials  
**Query**     Creating a table

```
mysql> select min(personid) from persons;
```

#### Output

```
+-----+
| min(personid) |
+-----+
| 101 |
+-----+
```

## Data Science – MySql – Imp Functions

---

### 3. max(column name) function

- ✓ The max(column) is a predefined function in mysql.
- ✓ By using this function we can get maximum value from specific column

**Login**      Login with valid credentials  
**Query**     Creating a table

```
mysql> select * from persons;
```

#### Output

```
mysql> select * from persons;
+-----+-----+-----+-----+-----+
| PersonID | LastName | FirstName | Address | location |
+-----+-----+-----+-----+-----+
| 101 | Danial | K | Near to Data Science Area | Hyderabad |
| 102 | Nireekshan | D | Near to AI Theatre | Bengaluru |
| 103 | Ranjan | M | Near to Python Theatre | Hyderabad |
| 104 | Prasad | K | Near to Python Road | Hyderabad |
+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

**Login**      Login with valid credentials  
**Query**     Creating a table

```
mysql> select max(personid) from persons;
```

#### Output

```
+-----+
| max(personid) |
+-----+
| 104 |
+-----+
1 row in set (0.00 sec)
```

## Data Science – MySql – Imp Functions

---

### 4. count(column name) function

- ✓ The count(column) is a predefined function in mysql.
- ✓ The count(column name) function returns the number of rows from the table.
- ✓ This function even returns the number of rows that matches a specified criterion.

**Login**      Login with valid credentials  
**Query**      Creating a table

```
mysql> select * from persons;
```

#### Output

```
mysql> select * from persons;
+-----+-----+-----+-----+-----+
| PersonID | LastName | FirstName | Address | location |
+-----+-----+-----+-----+-----+
| 101 | Danial | K | Near to Data Science Area | Hyderabad |
| 102 | Nireekshan | D | Near to AI Theatre | Bengaluru |
| 103 | Ranjan | M | Near to Python Theatre | Hyderabad |
| 104 | Prasad | K | Near to Python Road | Hyderabad |
+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

**Login**      Login with valid credentials  
**Query**      Creating a table

```
mysql> select count(location)
      from persons;
```

#### Output

```
+-----+
| count(location) |
+-----+
| 4 |
+-----+
```

## Data Science – MySql – Imp Functions

**Login** Login with valid credentials  
**Query** Creating a table

```
mysql> select count(location)
      from persons
     where location = 'hyderabad';
```

**Output**

```
+-----+
| count(location) |
+-----+
|            3 |
+-----+
1 row in set (0.00 sec)
```

## Data Science – MySql – Imp Functions

---

### 5. avg(column name) function

- ✓ The avg(column) is a predefined function in mysql.
- ✓ The avg(column name) function returns average value from the values

**Login  
Query**

Login with valid credentials  
Creating a table

```
mysql> create table products(
    prodname varchar(255),
    prodprice int
);
```

**Output**

Table created successfully

**Login  
Query**

Login with valid credentials  
Insert the data into table

```
mysql> insert into products(prodname, prodprice)
    values ('samsung', 12000);
```

Query OK, 1 row affected (0.01 sec)

```
mysql> insert into products(prodname, prodprice)
    values ('iphone 14', 75000);
```

Query OK, 1 row affected (0.01 sec)

```
mysql> insert into products(prodname, prodprice)
    values ('nokia', 10000);
```

Query OK, 1 row affected (0.01 sec)

## Data Science – MySql – Imp Functions

### Output

Data inserted successfully into the products table

### Login Query

Login with valid credentials

Select table

```
mysql> select * from products;
```

### Output

prodname	prodprice
samsung	12000
iphone 14	75000
nokia	10000

### Login Query

Login with valid credentials

Apply avg(column) function

```
mysql> select avg(prodprice)  
       from products;
```

### Output

avg(prodprice)
32333.3333

## Data Science – MySql – Imp Functions

### 6. sum(column name) function

- ✓ The sum(column) is a predefined function in mysql.
- ✓ The sum(column name) function returns sum of all values in column

#### Login Query

Login with valid credentials  
Select table

```
mysql> select * from products;
```

#### Output

prodname	prodprice
samsung	12000
iphone 14	75000
nokia	10000

#### Login Query

Login with valid credentials  
Apply sum(column) function

```
mysql> select sum(prodprice)  
       from products;
```

#### Output

sum(prodprice)
97000

# Data Science – MySql – Pandas - DataFrame

---

## 9. MySql – Pandas - DataFrame

### Contents

1. DataFrame .....	2
2. Install the required libraries .....	2

### 9. MySql – Pandas - DataFrame

#### 1. DataFrame

- ✓ Being a Data Scientist, we are very familiar with Pandas DataFrame
- ✓ We can create DataFrame from,
  - Csv file
  - Excel file
  - Json file
  - Mysql table
  - etc

#### 2. Install the required libraries

- ✓ pip install sqlalchemy
- ✓ pip install pymysql
- ✓ pip install cryptography

## Data Science – MySql – Pandas - DataFrame

---

**Program Name**

Accessing table from mysql  
demo1.py

```

print("Step 1: Importing libraries")
import pandas as pd
from urllib.parse import quote_plus
from sqlalchemy import create_engine

print("Step 2: Preparing database connection string")
db_connection_str =
    'mysql+pymysql://root:%s@localhost/danielsdb1' %
        quote_plus("D@1234niel#")

print("Step 3: Create database connection")
db_connection = create_engine(db_connection_str)

print("Step 4: Accessing read_sql function")
df = pd.read_sql('SELECT * FROM persons', con = db_connection)

print("Step 5: Accessing DataFrame")

print()
print(df)

```

**Output**

	PersonID	LastName	FirstName		Address	location
0	101	Dania	L	K	Near to Data Science Area	Hyderabad
1	101	Nireekshan		D	Near to AI theater	Bangalore
2	103	Ranjan		M	Near to Python theater	Hyderabad
3	104	Prasad		K	Near to Python Road	Hyderabad

# Generative AI

---

## Generative AI - LLM

### Table of Contents

<b>1. Generative AI</b> .....	3
1.1. Generate the Text .....	4
1.2. Generate the Images.....	5
1.3. Generate the Video.....	5
<b>2. Generative AI Diagram</b> .....	6
<b>3. Generative AI is generating: Text, Images, Video, Code &amp; etc</b> .....	6
3.1. Text .....	6
3.2. Images .....	6
3.3. Videos.....	6
6.4. Code .....	7
3.5. Music.....	7
<b>4. Artificial Intelligence</b> .....	7
<b>5. AI Model</b> .....	7
<b>6. Generative AI Models</b> .....	8
6.1. Generative Pre-trained Transformer (GPT) .....	8
6.2. Llama 2 .....	8
6.3. Claude .....	8
6.4. Gemini.....	9
6.5. PaLM2 .....	9
6.6. DALL-E .....	9
6.7. Stable Diffusion.....	10
6.8. Midjourney.....	10
6.9. CodeWhisperer .....	10
6.10. CodeLlama.....	10
6.11. Codex .....	11
<b>7. Use cases of Generative AI</b> .....	11
7.1. Content Generation .....	11
7.2. Personalized marketing.....	11
7.3. Customer service .....	11
7.4. Risk management.....	12
7.5. Compliance .....	12
7.6. Software Development.....	12
7.7. Data Augmentation.....	12

# Generative AI

---

<b>8. Below domains are using the Generative AI .....</b>	13
8.1. Financial .....	13
8.2. Healthcare.....	13
8.3. Manufacturing .....	13
8.4. Retail and Consumer Packaged Goods .....	13
8.4. Marketing and Sales.....	14
<b>9. OpenAI's Kick Introduction .....</b>	15
<b>10. Installation.....</b>	15
<b>11. OpenAI's Hello World Program.....</b>	15
<b>12. Understanding the Hello World Program.....</b>	16
<b>13. Hugging Face.....</b>	17
13.1. transformers library .....	17
<b>14. Large Language Model (LLM) .....</b>	20
<b>15. Large Language Model Architecture.....</b>	20
15.1. Input Embedding:.....	21
15.2. Positional Encoding:.....	21
15.3. Add & Norm:.....	21
15.4. Multi-Head Attention:.....	21
15.5. Feed Forward: .....	21
15.6. Encoder Block (Left Side): .....	21
15.7. Masked Multi-Head Attention:.....	22
15.8. Decoder Block (Right Side):.....	22
15.9. Linear + Softmax: .....	22
15.10. Output Probabilities:.....	22

## Generative AI

### Generative AI

#### 1. Generative AI

**Generative AI = Generative + AI**

- ✓ Generative AI is a type of Artificial Intelligence technology.
- ✓ It is capable to **generate** the **data** or new content.
- ✓ Data means,
  - Text.
  - Image.
  - Audio.
  - Video.
  - Code & etc.

**Generative AI = Generative + AI**

**Generative** → **Generate the content.**  
**AI** → **Using Artificial Intelligence**

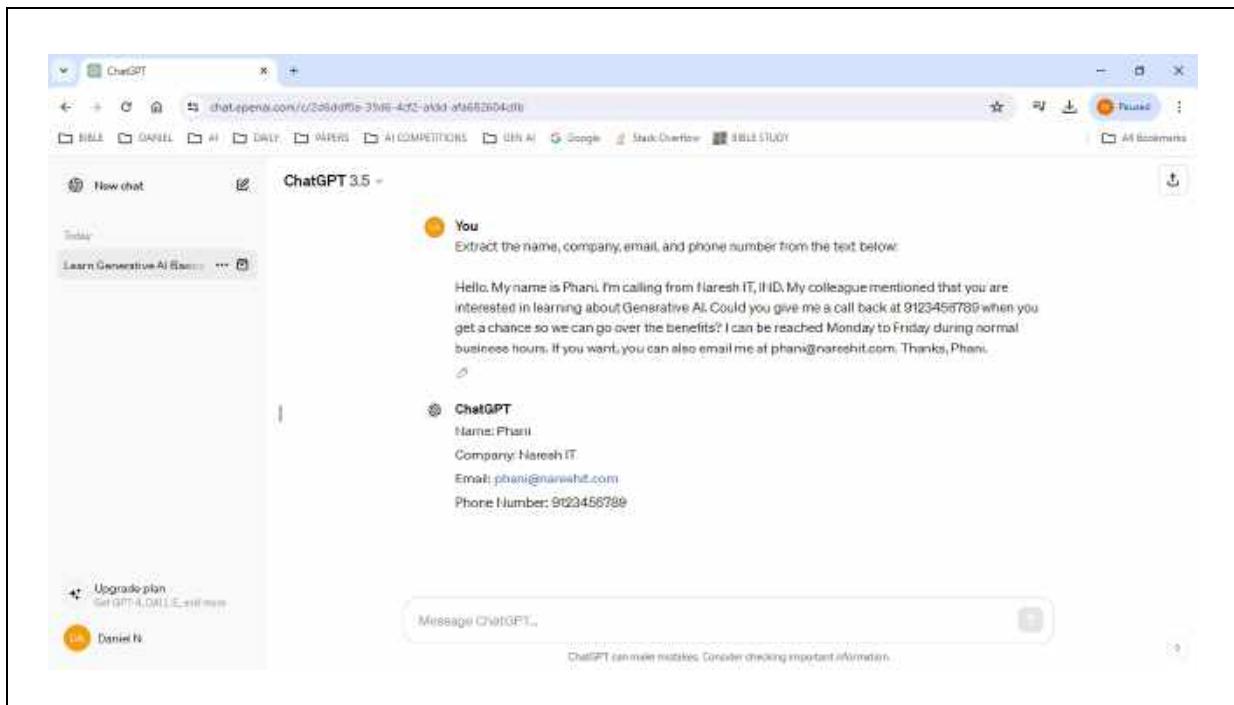
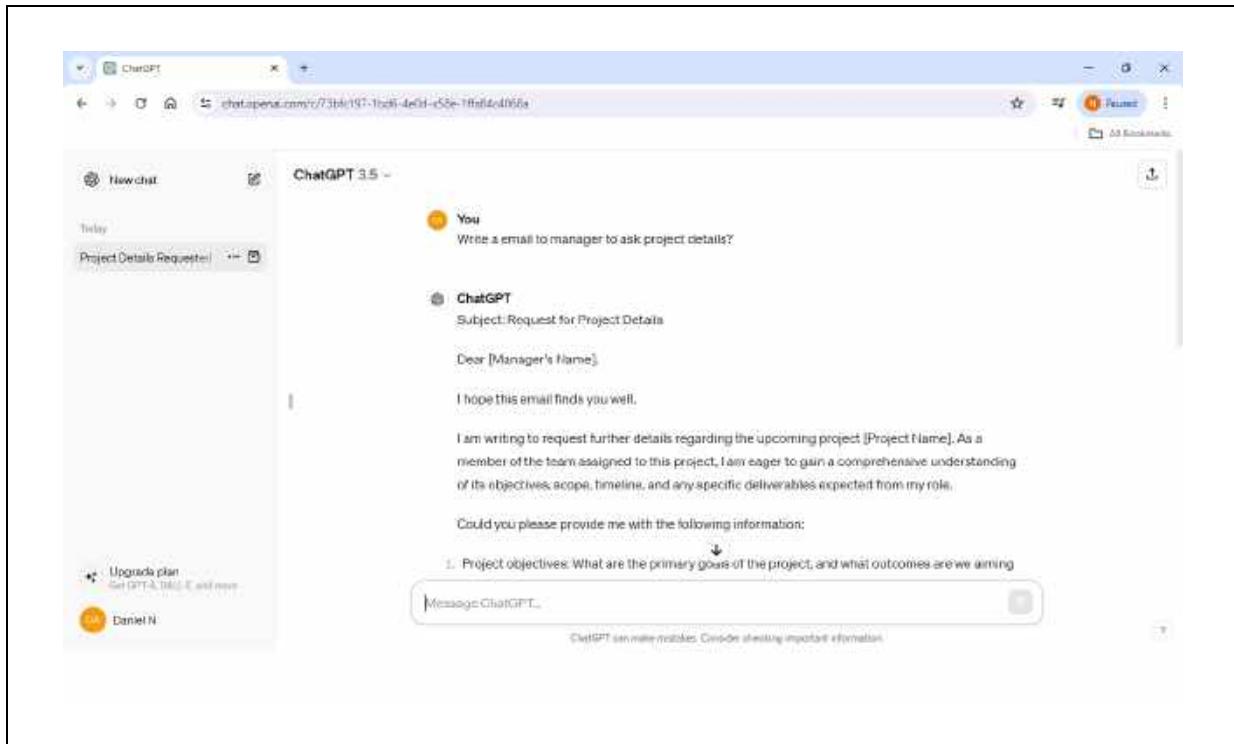
#### Kind note

- ✓ Generative AI **full form** is **Generative Artificial Intelligence**.
- ✓ Generative AI **short form** is **Gen AI**.

# Generative AI

## 1.1. Generate the Text

- ✓ By using Generative AI tool, we can generate the **Text**.
- ✓ One of the Gen AI tools is,
  - ChatGPT



## Generative AI

### 1.2. Generate the Images

- ✓ By using Generative AI tool, we can generate the **Images**.
- ✓ One of the Gen AI tools is,
  - dall-e-2



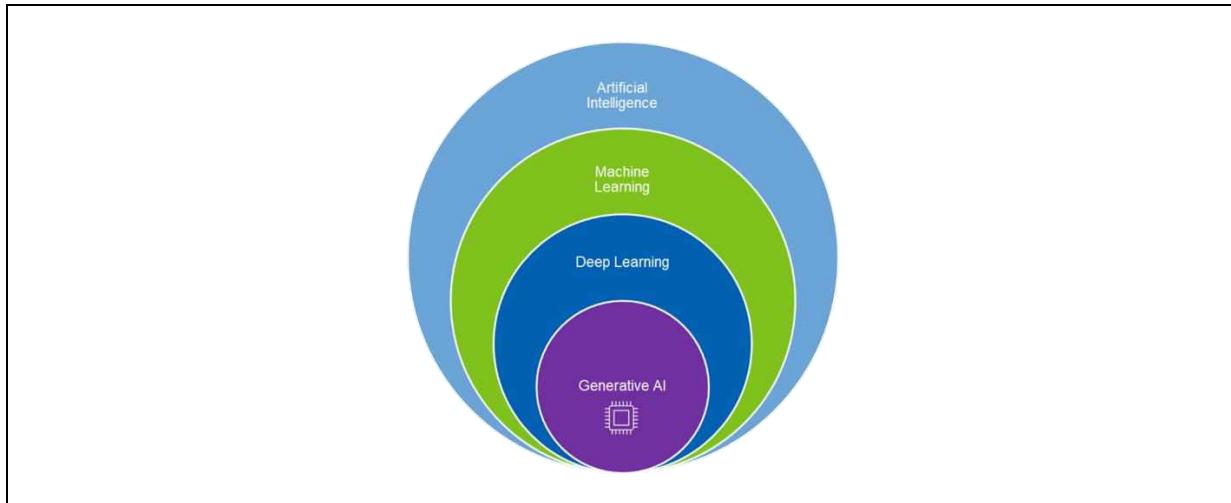
A photo of Michelangelo's sculpture of David wearing headphones dancing.

### 1.3. Generate the Video

- ✓ By using Generative AI tool, we can generate the **Images**.
- ✓ One of the Gen AI tools is,
  - [www.videogen.io](http://www.videogen.io)

## Generative AI

### 2. Generative AI Diagram



### 3. Generative AI is generating: Text, Images, Video, Code & etc

#### 3.1. Text

- ✓ Creating a realistic text like news articles, blog posts.
- ✓ Many platforms are using this like,
  - Generating content for websites and social media
  - Creating personalized marketing materials etc.

#### 3.2. Images

- ✓ Creating a realistic image of people, objects and scenes that do not exists in the real world.
  - Generating realistic product images for e-commerce.
  - Creating training data for other AI models & etc

#### 3.3. Videos

- ✓ Creating videos that do not exist in real world.
  - Creating special effects for movies and TV shows.
  - Creating personalized video content for marketing and advertising.

## Generative AI

---

### 6.4. Code

- ✓ Generative AI models generate the code for different programming languages.
- ✓ This can be helpful to programmers/developers.

### 3.5. Music

- ✓ Generative AI models are being used to create new music.
  - Creating music for movies and TV shows.
  - Generating personalized playlists & etc.

## 4. Artificial Intelligence

- ✓ Artificial Intelligence is the **ability** for a computer to think, learn and do tasks.
  - Problem solving.
  - Understanding Language.
  - Making decisions & etc.
- ✓ AI having **below** topics like,
  - Machine Learning.
  - Deep Learning.
  - Natural Language Processing.
  - Computer Vision.
- ✓ By using above topics we can do,
  - Prediction.
  - Classification.
  - Sentiment analysis & etc.
- ✓ AI can **enable** machines to mimic human.
- ✓ Artificial Intelligence short form is **AI**.

## 5. AI Model

- ✓ AI model is a **program**.
- ✓ This program **analyses** datasets to find **patterns** and make **predictions**.

## Generative AI

---

### 6. Generative AI Models

- ✓ In Generative AI, models generate the data.

#### 6.1. Generative Pre-trained Transformer (GPT)

- ✓ **GPT** is Large Language Model.
- ✓ This is developed by OpenAI.
- ✓ It's trained on a massive dataset of text and code.
- ✓ It is Capable of,
  - Generating text.
  - Translating languages.
  - Writing different creative content.
  - Answering your questions.
- ✓ GPT - 4 is the latest version at the time of this writing.

#### 6.2. Llama 2

- ✓ **Llama 2** is Large Language Model.
- ✓ This is developed by Meta.
- ✓ Llama 2 is second version of a natural large language model.

#### 6.3. Claude

- ✓ Claude is Large Language Model.
- ✓ This is developed by startup company called Anthropic.
- ✓ This is like a ChatGPT, it can,
  - Generate text
  - Write code
  - Summarize & etc.

## Generative AI

---

### 6.4. Gemini

- ✓ Gemini is Generative AI model.
- ✓ This is developed by Google company.
- ✓ It's a Google's new multi-modal model.
- ✓ This can understand,
  - Text.
  - Images.
  - Videos.
  - Audio.
- ✓ It will be available in different sizes (Ultra, Pro, and Nano), each with different capabilities.

### 6.5. PaLM2

- ✓ PaLM is Large Language Model.
- ✓ The full form of PaLM is Pathway Language Model.
- ✓ It is a multi-modal model
- ✓ This is developed by Google company.
- ✓ This can process,
  - Text.
  - Code.
  - Images.

### 6.6. DALL-E

- ✓ DALL-E is Visual AI model.
- ✓ This is developed by OpenAI.
- ✓ It can create,
  - Realistic images from text prompts.

### 6.7. Stable Diffusion

- ✓ Stable Diffusion is an image generation model.
- ✓ This is developed by OpenAI.
- ✓ It can,
  - Generate detailed images.
  - Text descriptions.
  - Inpainting and out painting.
  - Generate image-to-image translations.
- ✓ This model generates these are by prompt as input.

### 6.8. Midjourney

- ✓ Midjourney is an image generation model.
- ✓ This is developed by startup called Midjourney, Inc.
- ✓ This is like DALL-E and Stable Diffusion.

### 6.9. CodeWhisperer

- ✓ CodeWhisperer is a **code** generation model.
- ✓ This is developed by AWS.
- ✓ This can generate the,
  - Code in several programming languages. (Python, Java, JavaScript, TypeScript & etc)

### 6.10. CodeLlama

- ✓ CodeLlama is a large language model.
- ✓ This is built on Llama 2.
- ✓ This model specifically trained on code.
- ✓ It also comes in various sizes and supports multiple popular programming languages.

## 6.11. Codex

- ✓ Codex is a large language model.
- ✓ This is a **code** generation model.
- ✓ This model specifically trained on code.
- ✓ This can generate the,
  - Code in several programming languages. (Python, C#, Java, JavaScript, SQL, Go, PHP, and Shell).

## 7. Use cases of Generative AI

- ✓ In Generative AI, models generate the data.

### 7.1. Content Generation

- ✓ Generative AI helpful to generate the content,
  - Blogs,
  - Reports,
  - e-mails
  - Social media posts.
- ✓ This content helpful to business for marketing.

### 7.2. Personalized marketing

- ✓ Generative AI can create personalized marketing content,
  - e-mails
  - Landing pages.
  - Social media posts.
- ✓ This content helpful to businesses to reach their target audience more effectively and increase conversion rates.

### 7.3. Customer service

- ✓ Generative AI can be used to create chatbots that can answer customer questions and resolve issues.
- ✓ This is really great advantage like, free human customer service.

## Generative AI

---

### 7.4. Risk management

- ✓ Generative AI can identify and predict risks,
  - Fraud.
  - Cyberattacks
  - Supply chain disruptions.
- ✓ This will help businesses to protect their assets.

### 7.5. Compliance

- ✓ Generative AI can,
  - Generate compliant documents,
    - Contracts.
    - Reports.
    - Disclosures.
- ✓ This can help businesses to save time and money and reduce the risk of non-compliance.

### 7.6. Software Development

- ✓ Generative AI can,
  - Generate new code
  - Provide code snippets, or even write simple software.
  - Potentially saving time and reducing errors.
- ✓ In addition, it also helps document code, refactor, generate test cases, and optimize existing code.

### 7.7. Data Augmentation

- ✓ Generative AI can create synthetic data for Data Science projects if needed.

## Generative AI

---

### 8. Below domains are using the Generative AI

#### 8.1. Financial

- ✓ Generative AI can help with,
  - Decision-making.
  - Risk model assessment.
  - Development of new financial products and services.
- ✓ Customer operations to enhance services and resolutions for each client based on transactions and history.

#### 8.2. Healthcare

- ✓ Generative AI is used to develop,
  - New drugs and treatments.
  - Design medical devices.
  - Create personalized patient treatment plans.
  - Generate patient documentation on instructions, risks, and drug interactions.

#### 8.3. Manufacturing

- ✓ Generative AI is used to develop,
  - Design new products.
  - Optimize manufacturing processes.
  - Improve quality control.

#### 8.4. Retail and Consumer Packaged Goods

- ✓ Generative AI is used to,
  - Personalize shopping experiences.
  - Recommend products.
  - Manage inventory.
  - Accelerate consumer research.
  - Enhance the supply chain & etc.

### 8.4. Marketing and Sales

- ✓ Generative AI is helping enhance,
  - Understand real-time customer trends.
  - Personalized outreach embedded into virtual assistants.
  - Dynamic customer journeys & etc

## 9. OpenAI's Introduction

- ✓ OpenAI is an artificial intelligence research organization and technology company founded in December 2015.
- ✓ Its mission is to ensure that artificial general intelligence (AGI) benefits all of humanity.
- ✓ OpenAI develops AI models and technologies, including the famous language models like GPT (Generative Pre-trained Transformer).

## 10. Installation

- ✓ pip install openai

## Hello World example

### 11. OpenAI's Hello World Program

- ✓ Let's write and run the basic example by using OpenAI's API.

#### Kind note

- ✓ To run below program, we should set API key first.

**Program Name** OpenAI's API Hello World Program  
demo1.py

```
from openai import OpenAI

client = OpenAI()

response = client.completions.create(
    model = "gpt-3.5-turbo-instruct",
    prompt = "How are you."
)

print(response.choices[0].text)
```

#### Output

I am an AI and do not have the capability to feel emotions.

### 12. Understanding the Hello World Program

- ✓ **openai** is library, we are importing.
- ✓ **OpenAI** is predefined class, we are importing from openai library.
- ✓ **client** is an object to OpenAI class.
- ✓ **completion** is an object, accessing by using client.
- ✓ **create** is method, accessing by using client.completions.
- ✓ **create** method having two parameters,
  - **model**
  - **prompt**

### 13. Hugging Face

- ✓ Hugging Face is a leading company in the field of natural language processing (NLP) and machine learning.
- ✓ Founded in 2016, it has become well-known for its contributions to the AI community, particularly through its open-source libraries.

#### 13.1. transformers library

- ✓ pip install transformers
  - It Provides a wide range of pre-trained models for various NLP tasks, such as text generation, translation, summarization, and classification.
  - Supports models like BERT, GPT, and many others.

##### Program

Name Text Classification

demo1.py

```
from transformers import pipeline

# Load a pre-trained model for text classification
classifier = pipeline("text-classification")

# Classify a text
a = "I love using Hugging Face's transformers library!"
result = classifier(a)
print(result)
```

##### Output

```
[{'label': 'POSITIVE', 'score': 0.9978122711181641}]
```

## Generative AI

**Program Name** Text Generation  
demo2.py

```
from transformers import pipeline

# Load a pre-trained model for text generation
generator = pipeline("text-generation")

# Generate text based on a prompt
a = "Once upon a time, in a land far away,"
result = generator(a, max_length = 50)
print(result)
```

**Output**

```
[{'generated_text': 'Once upon a time, in a land far away, the land of the dead, and the land of a living creature, the human race had become. In them died the dead, and in them dwelt men, who in these times had never'}]
```

## Generative AI

**Program Name**

Question Answering  
demo3.py

```
from transformers import pipeline

# Load a pre-trained model for question answering
qa = pipeline("question-answering")

# Answer a question based on a context
context = "Hugging Face is an AI company based in New York.  
They are known for their work in natural language processing."
question = "Where is Hugging Face based?"

result = qa(question=question, context=context)
print(result)
```

**Output**

```
{'score': 0.9903117418289185, 'start': 39, 'end': 47, 'answer': 'New  
York'}
```

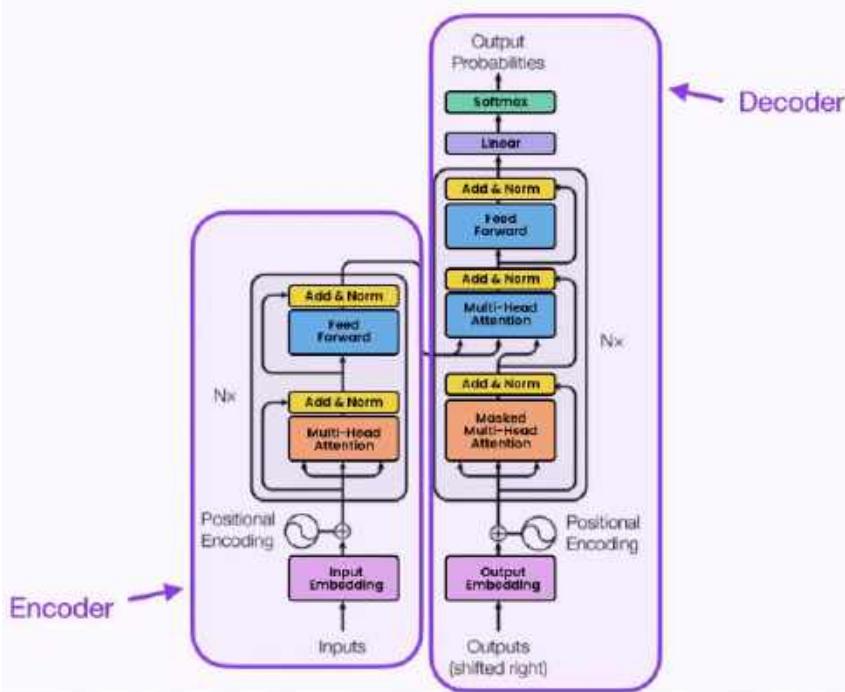
## 14. Large Language Model (LLM)

- ✓ LLM stands for "Large Language Model".
- ✓ It is a type of AI trained model on text data,
  - To understand human text
  - To generate human-like text
  - Handling tasks like answering questions
  - Summarizing text data
  - Translating the text data
  - Creating new content.

## 15. Large Language Model Architecture

- ✓ Large Language Models (LLMs) use the Transformer neural network architecture.
- ✓ It is introduced in the 2017 paper "Attention is All You Need" by Vaswani et al.
- ✓ Here we can find effectively processing large-scale data and capturing complex language patterns.

### The Transformer - Model Architecture



## 15.1. Input Embedding:

- ✓ **Purpose:** Each word in the input sequence is represented as a high-dimensional vector, capturing **semantic meaning**.

## 15.2. Positional Encoding:

- ✓ **Purpose:** Adds information about the position of each token. Positional encoding is added to the Transformer model to provide information **about the order of tokens**

## 15.3. Add & Norm:

- ✓ **Purpose:** Combines the output of a sub-layer with the input and normalizes it. The "Add" operation is a residual connection that helps the model avoid vanishing gradients, while "Norm" refers to Layer Normalization, which stabilizes and **speeds up training**.

## 15.4. Multi-Head Attention:

- ✓ **Purpose:** Multi-head attention uses several attention mechanisms (heads) in parallel. Each head processes the input differently, allowing the model to capture **various relationships** in the data.

## 15.5. Feed Forward:

- ✓ **Purpose:** After attention is applied, a feed-forward network processes the data, adding non-linearity and allowing the model to **capture more complex patterns**.

## 15.6. Encoder Block (Left Side):

- ✓ **Purpose:** Consists of several identical layers (denoted as Nx), each containing Multi-Head Attention, Add & Norm, and Feed Forward sub-layers. Processes the input sequence to **create a context-aware representation** of the input tokens.

### 15.7. Masked Multi-Head Attention:

- ✓ **Purpose:** In the decoder, this layer masks future positions to ensure that predictions for a token depend only on previous tokens, not on future ones.

### 15.8. Decoder Block (Right Side):

- ✓ **Purpose:** Similar to the encoder, it consists of several identical layers ( $N_x$ ), but with an additional masked attention mechanism. Generates the output sequence by attending to both the encoder's output and the previously generated tokens.

### 15.9. Linear + Softmax:

- ✓ **Purpose:** The linear layer reduces the dimensionality of the decoder's output, and the softmax function converts this into a probability distribution, predicting the next token in the sequence.

### 15.10. Output Probabilities:

- ✓ **Purpose:** The final prediction of the model. The output probabilities are used to determine the next token in the sequence, forming the basis for tasks like translation, summarization, or text generation.

# Data Science – Power BI Installation

---

## Power BI Installation

### Contents

1. Steps to follow download and install .....	2
---	---

# Data Science – Power BI Installation

## Power BI Installation

### 1. Steps to follow download and install

- ✓ Click on below url

<https://www.microsoft.com/en-us/download/details.aspx?id=58494>

The screenshot shows a Microsoft download details page. At the top, there is a message: "Important! Selecting a language below will dynamically change the complete page content to that language." Below this is a "Select language" dropdown set to "English" and a blue "Download" button. Underneath, there are two links: "Expand all" and "Collapse all". A section titled "Details" is expanded, showing the following information:

Version:	Date Published:
2.138.1004.0	11/19/2024
File Name:	File Size:
PBIDesktopSetup.exe	504.9 MB
PBIDesktopSetup_x64.exe	544.9 MB

## Data Science – Power BI Installation

- ✓ Select to download

Choose the download you want X

<input type="checkbox"/> File Name	Size
<input type="checkbox"/> PBIDesktopSetup.exe	504.9 MB
<input type="checkbox"/> PBIDesktopSetup_x64.exe	544.9 MB

Download Total size: 0 bytes

Choose the download you want X

<input type="checkbox"/> File Name	Size
<input checked="" type="checkbox"/> PBIDesktopSetup.exe	504.9 MB
<input type="checkbox"/> PBIDesktopSetup_x64.exe	544.9 MB

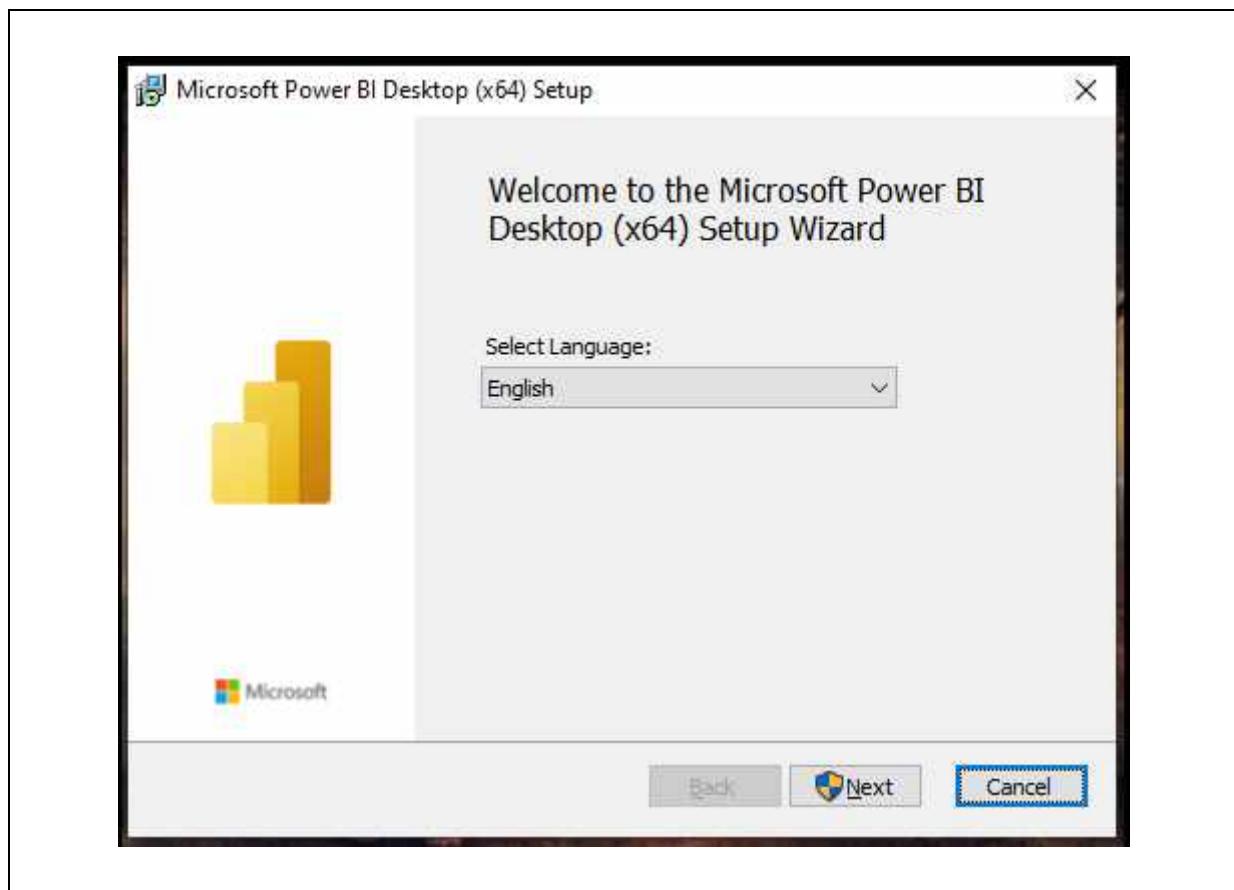
Download Total size: 504.9 MB

## Data Science – Power BI Installation

- ✓ Its .exe file

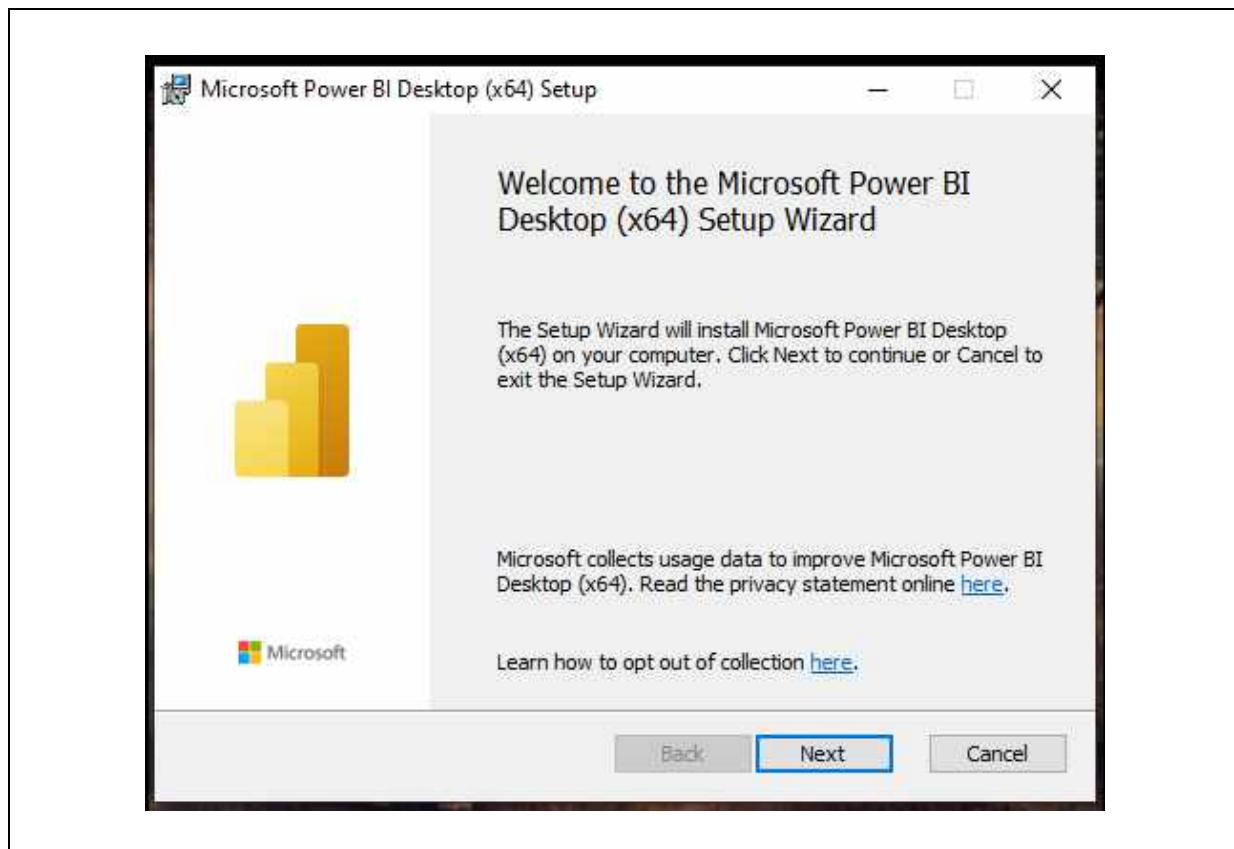


- ✓ Just give double click on the software



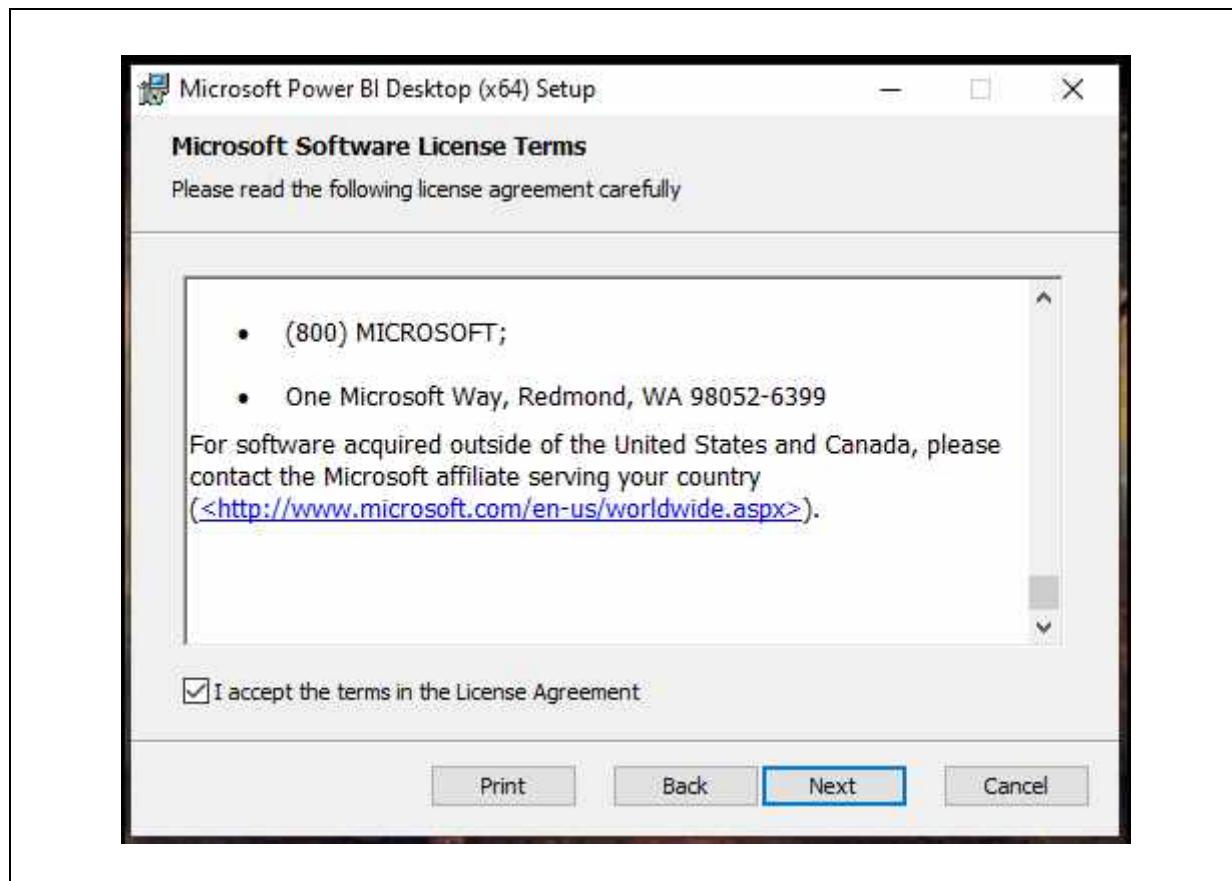
## Data Science – Power BI Installation

- ✓ Click on Next button



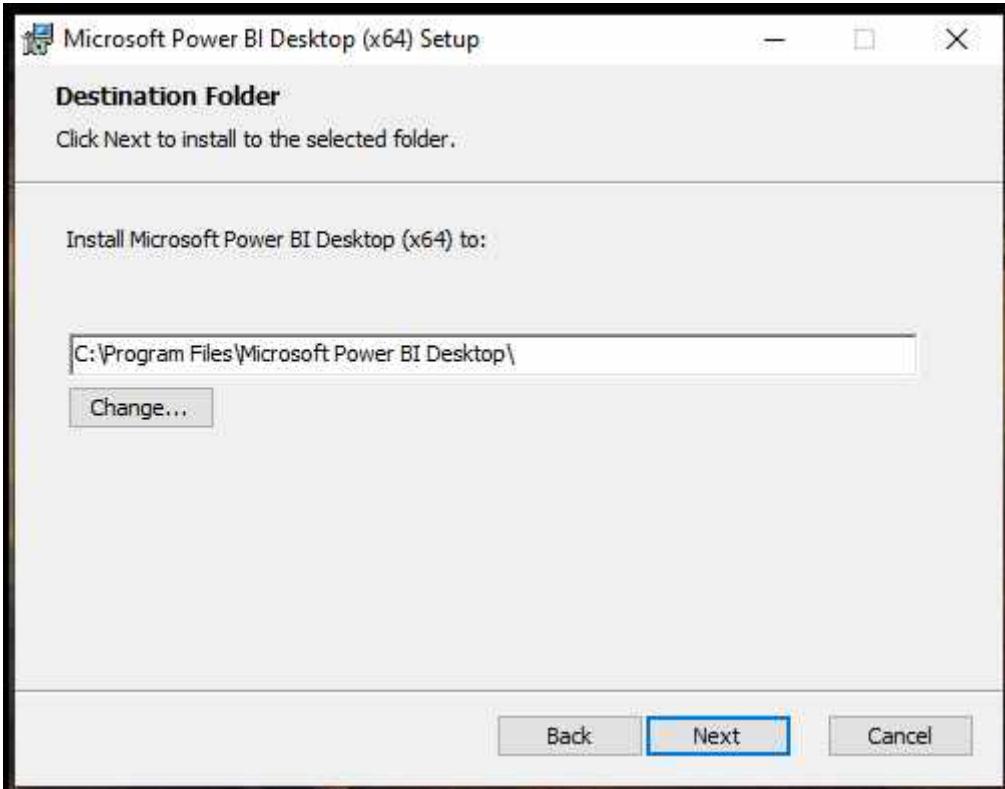
## Data Science – Power BI Installation

- ✓ Click on Next button



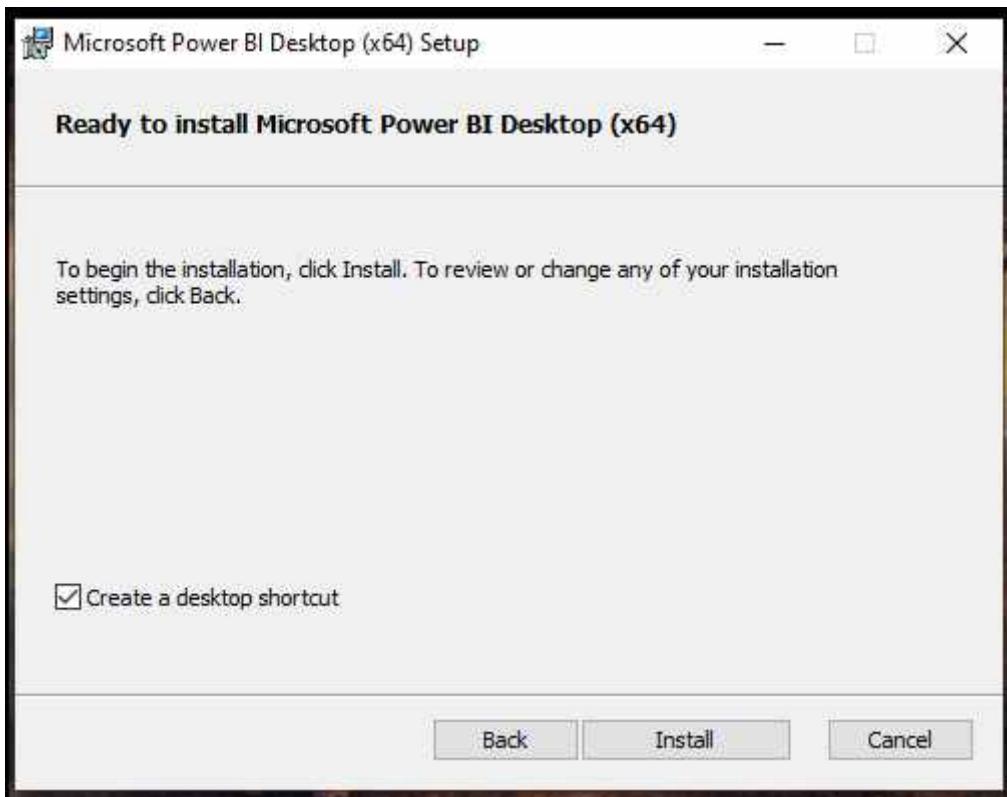
## Data Science – Power BI Installation

- ✓ Click on Next



## Data Science – Power BI Installation

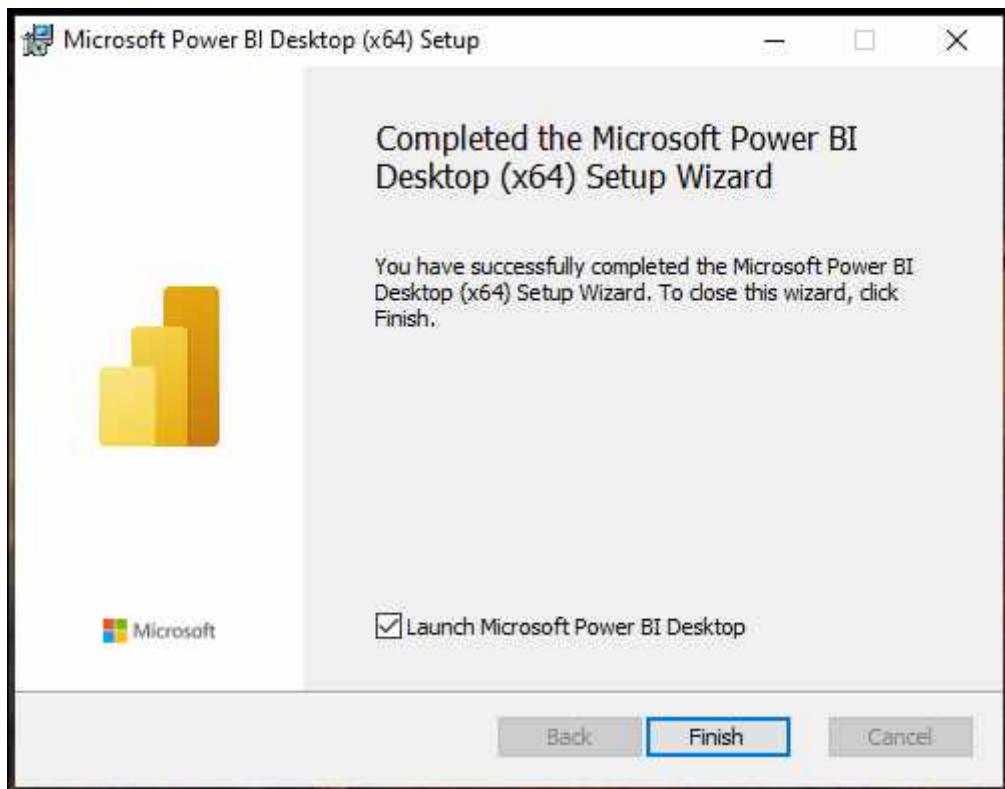
- ✓ Click on Install



## Data Science – Power BI Installation



## Data Science – Power BI Installation



# Data Science – Power BI - Introduction

---

## 1. Power BI - Introduction

### Contents

1. Power BI.....	2
2. Power BI: Key Features .....	3
3. Advantages .....	4
4. Power BI Core components.....	5
5. Power BI Desktop .....	6
5.1. Key Features of Power BI Desktop.....	6
6. Power Service .....	7
6.1. Key Features of Power BI Service.....	7
7. Comparison: Power BI Desktop vs. Power BI Service .....	8

## Data Science – Power BI - Introduction

### 1. Power BI - Introduction



#### 1. Power BI

- ✓ **Power BI** is a Data Visualization and Business Intelligence **tool** from Microsoft.
- ✓ By using this we can collect, analyse, and visualize data.
- ✓ We can create interactive reports and dashboards to see important business data, like sales numbers or website traffic, in one place.
- ✓ It's like turning numbers into pictures (charts, graphs, etc.) to help make better decisions.

#### Power BI Logo



## 2. Power BI: Key Features

### ✓ Power Query

- Its an ETL tool, extract, transform & load the data
- Cleans, combines, and reshapes data.

### ✓ Power Pivot

- A data modeling tool.
- Establishes relationships between tables

### ✓ Power View

- A visualization tool
- Creates interactive data visualizations and dashboards.
- Designs charts, maps, and other visuals to represent data insights.

### ✓ Power BI Service

- A service for sharing and collaboration.
- Publishes, shares, and consumes reports and dashboards.
- Allows access to reports and dashboards via the web.
- Provides real-time data monitoring and alerts.

## Data Science – Power BI - Introduction

---

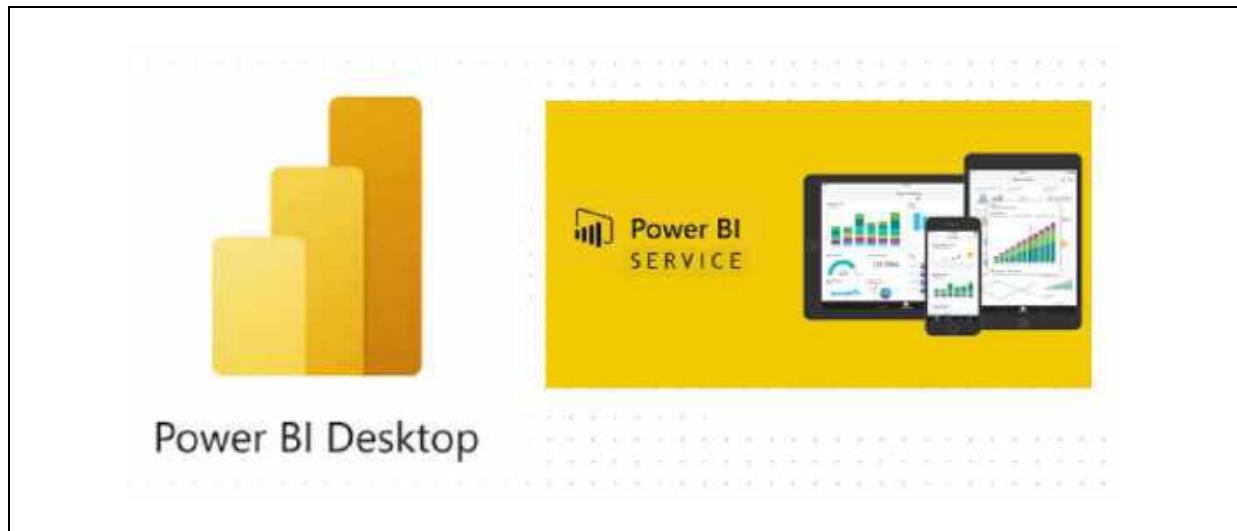
### 3. Advantages

- ✓ Easy to Use
  - It's simple to learn, even for people without a tech background.  
You can drag and drop data to create reports.
- ✓ Connects to Many Data Sources
  - Power BI can pull data from lots of places—Excel, databases, cloud services like Google Analytics, and more.
- ✓ Data Easy to Understand
  - You can create visual reports (like graphs and charts) that are easy to read and explore.
- ✓ Real-Time Updates
  - You can see up-to-the-minute data, which is helpful for tracking business performance right now.
- ✓ Sharing and Collaboration
  - It's easy to share your reports with others, and you can work together in real time.
- ✓ Secure
  - It has built-in security to keep your data safe.
- ✓ Smart Features
  - Power BI uses Artificial Intelligence to help you spot trends or get automatic insights from your data.

## Data Science – Power BI - Introduction

### 4. Power BI Core components

- ✓ Power BI Desktop
- ✓ Power BI Service



## Data Science – Power BI - Introduction

### 5. Power BI Desktop

- ✓ Power BI Desktop is a Windows-based application.
- ✓ Users can create, design, and develop interactive reports and data models.
- ✓ It is the primary tool for data preparation, modeling, and visualization.



#### 5.1. Key Features of Power BI Desktop

- ✓ Data Import and Transformation
  - Connects to various data sources (Excel, databases, APIs, etc.).
  - Cleans and transforms data using Power Query.
- ✓ Data Modeling
  - Establishes relationships between tables.
  - Enables advanced calculations using DAX (Data Analysis Expressions).
- ✓ Visualization Creation
  - Offers a wide range of charts, graphs, and custom visuals.
- ✓ No Internet Required
  - Fully functional offline for data analysis and report creation.

### 6. Power Service

- ✓ Power BI Service is a **cloud-based platform**.
- ✓ Users can publish, share, and collaborate on Power BI reports and dashboards.
- ✓ It enables real-time access to reports sharing across teams and organizations.

#### 6.1. Key Features of Power BI Service

- ✓ Report and Dashboard Hosting
  - Hosts reports created in Power BI Desktop.
- ✓ Collaboration and Sharing
  - Shares reports and dashboards with colleagues or stakeholders securely.
  - Supports comments, feedback, and collaborative analysis.
- ✓ Real-Time Data Monitoring
  - Connects to live data sources for real-time insights.
  - Sends alerts for data changes or thresholds via notifications.
- ✓ Access Anywhere
  - Accessible from any device with an internet connection, including mobile devices.
- ✓ Scheduled Data Refresh
  - Automatically updates data from connected sources at regular intervals.

## Data Science – Power BI - Introduction

---

### 7. Comparison: Power BI Desktop vs. Power BI Service

Feature	Power BI Desktop	Power BI Service
1. Purpose	✓ Create & design reports	✓ Share, collaborate & access reports
2. Platform	✓ Windows application	✓ Cloud based application
3. Visualization	✓ Full report creation capabilities	✓ Primarily dashboard creation
4. Collaboration	✓ No collaboration features	✓ Enables sharing and commenting
5. Internet Dependency	✓ Works offline	✓ Requires internet
6. Real-Time Data	✓ Not supported directly	✓ Supports real-time updates

**2. Power BI – Creating Charts****Contents**

<b>1. Column Chart.....</b>	2
1.1. Initial steps .....	2
1.2. Steps to create column chart.....	5
<b>2. Stacked Column Chart.....</b>	9
2.1. Steps to create stacked column chart.....	9
<b>3. Pie Chart .....</b>	12
3.1. Steps to create stacked column chart.....	12
<b>4. Donut Chart .....</b>	14
4.1. Steps to create stacked column chart.....	14
<b>5. Funnel Chart .....</b>	16
5.1. Steps to create Funnel chart .....	16
<b>6. Include and Exclude.....</b>	18
6.1. Steps to create stacked column chart.....	18
<b>7. View Data and Export in CSV from Power BI.....</b>	21

# Data Science – Power BI – Creating Charts

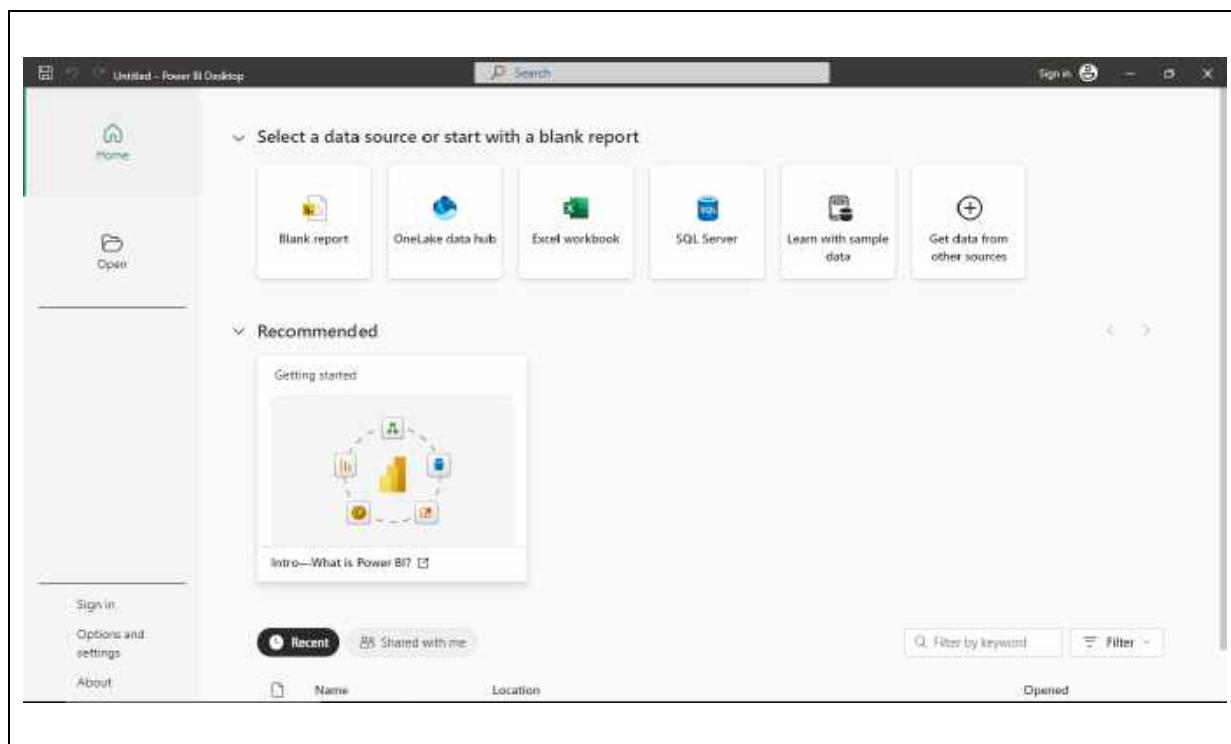
## 2. Power BI – Creating Charts

### 1. Column Chart

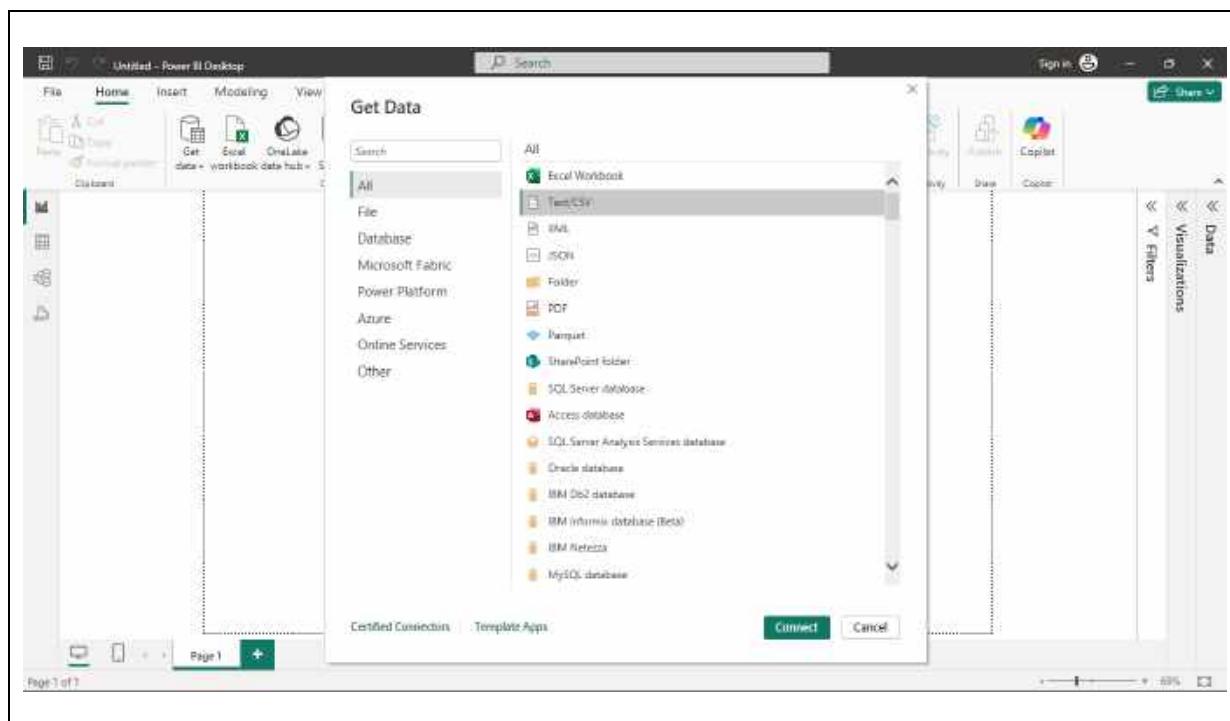
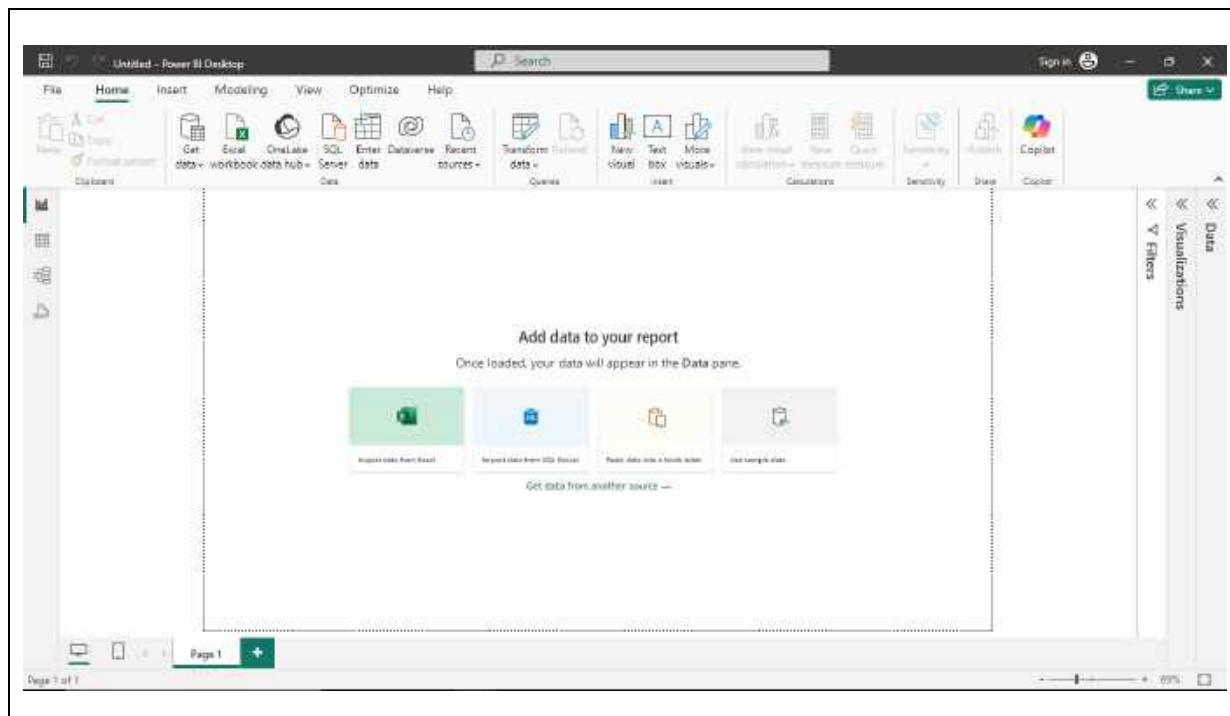
- ✓ A **Column Chart** in Power BI is a visual representation of data using vertical bars to show values for different categories.
- ✓ It is one of the most commonly used charts.
- ✓ By using this we can compare the quantities across categories or tracking changes over time.

#### 1.1. Initial steps

- ✓ Open Power BI Desktop.
- ✓ Click on blank report.
- ✓ Import your dataset by clicking **Home > Get Data** and selecting the file type (Excel, CSV, database, etc.).
- ✓ Ensure your data loaded.



## Data Science – Power BI – Creating Charts



## Data Science – Power BI – Creating Charts

The screenshot shows the 'Data Load Preview' window in Power BI. The file 'sales1.csv' is loaded, displaying a preview of the data. The columns are labeled: Order ID, Customer Name, Product, and Quantity. The data includes various purchases like '34in Ultrawide Monitor', 'Samsung m10', and 'iPhone 11'. A note at the bottom states: 'The data in the preview has been truncated due to size limits.' At the bottom right are buttons for 'Load', 'Transform Data', and 'Cancel'.

Order ID	Customer Name	Product	Quantity
166837	Veeru	34in Ultrawide Monitor	2
166838	Tarun	Samsung m10	3
166839	Kedar	20in Monitor	1
166840	Lavanya	iPhone 11	3
166841	Venu	Macbook Pro Laptop	2
166842	Venu	Bose SoundSport Headphones	2
166843	Harsha	27in 4K Gaming Monitor	2
166844	Harsha	Samsung m10	1
166845	Siddhu	34in Ultrawide Monitor	2
166846	Siddhu	iPhone 11	1
166847	Venu	Samsung m10	2
166848	Karteek	Macbook Pro Laptop	1
166849	Shahid	ThinkPad Laptop	1
166849	Siddhu	27in 4K Gaming Monitor	1
166850	Jaya Chandra	Bose SoundSport Headphones	1
166851	Jaya Chandra	Macbook Pro Laptop	1
166852	Madhurima	iPhone 7s	2
166853	Kedar	Samsung m10	1
166854	Tarun	iPhone 7	1
166855	Sumenth	Samsung m10	2

The data in the preview has been truncated due to size limits.

Extract Table Using Examples      Load      Transform Data      Cancel

The screenshot shows the 'Data' pane in the Power BI Data Source view. It displays the 'sales1' dataset with four columns: Customer Name, Order ID, Product, and Quantity. The 'Filters' pane on the left is visible, showing the current filters applied to the data.

Data

Search

sales1

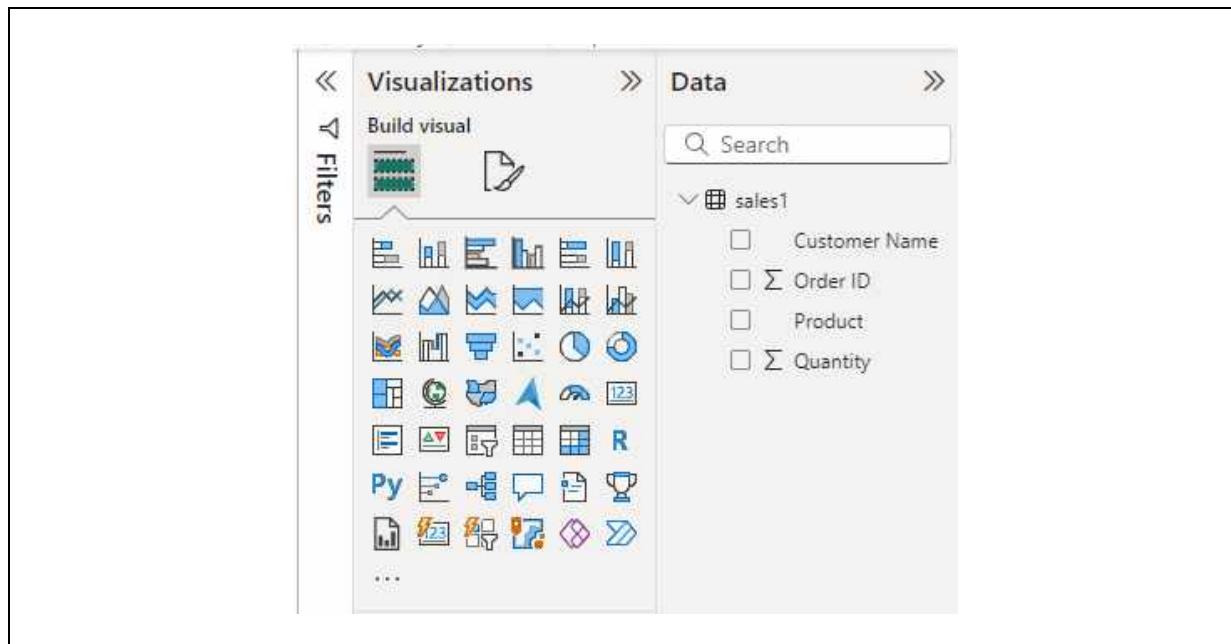
- Customer Name
- $\sum$  Order ID
- Product
- $\sum$  Quantity

Filters

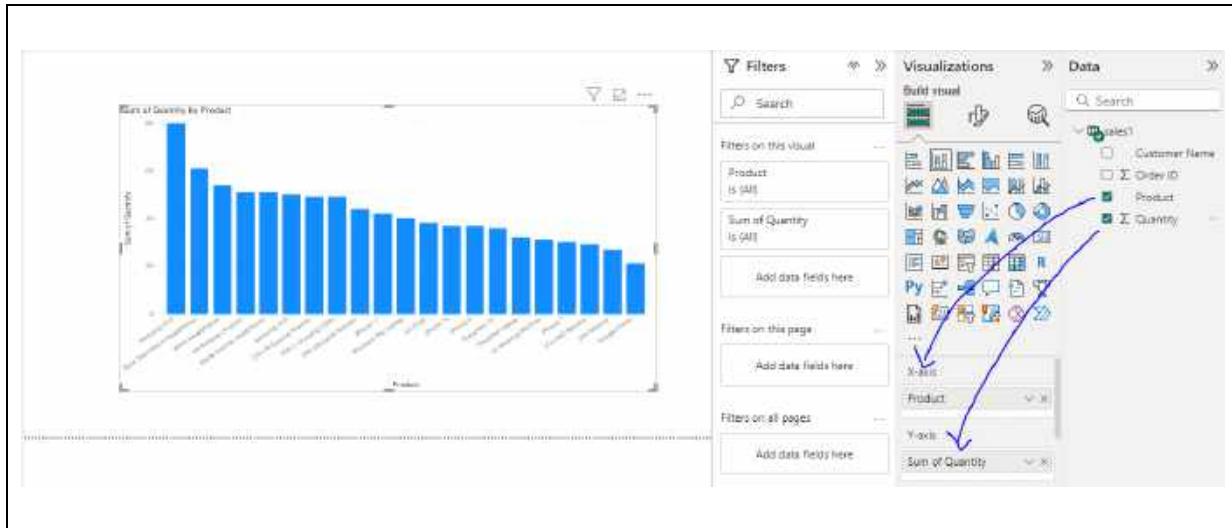
## Data Science – Power BI – Creating Charts

### 1.2. Steps to create column chart

- ✓ Open Power BI Desktop.
- ✓ Load the data: **sales1.csv**
- ✓ Select the Column Chart from the Visualizations pane.
- ✓ Drag fields to:
  - X Axis: product
  - Y Axis: quantity
  - Legend: The subcategory (if having)
- ✓ Customize using the Format Visual pane to adjust colors, labels, and axis titles.



## Data Science – Power BI – Creating Charts



## Data Science – Power BI – Creating Charts

---

### More options

- ✓ Format your visual
- ✓ **Visual**
  - X-axis
    - Values -> On or off
      - Font
      - Colors
    - Title -> On or off
      - Title Text
      - Style
      - Font
      - Color
    - Layout
      - Width
  - Y-axis
    - Range(optional)
    - Values -> On or off
      - Font
      - Colors
      - Display units
    - Title -> On or off
      - Title Text
      - Style
      - Font
      - Color
  - Legend -> On or off
    - Options
      - Top left
      - Top to centre
      - Lot of other options too
    - Text
      - Font
      - Color
    - Title -> On or off

## Data Science – Power BI – Creating Charts

---

- Title Text
  - Data labels -> On or off
  - Total labels -> On or off
- 
- ✓ General
- Title -> On or off

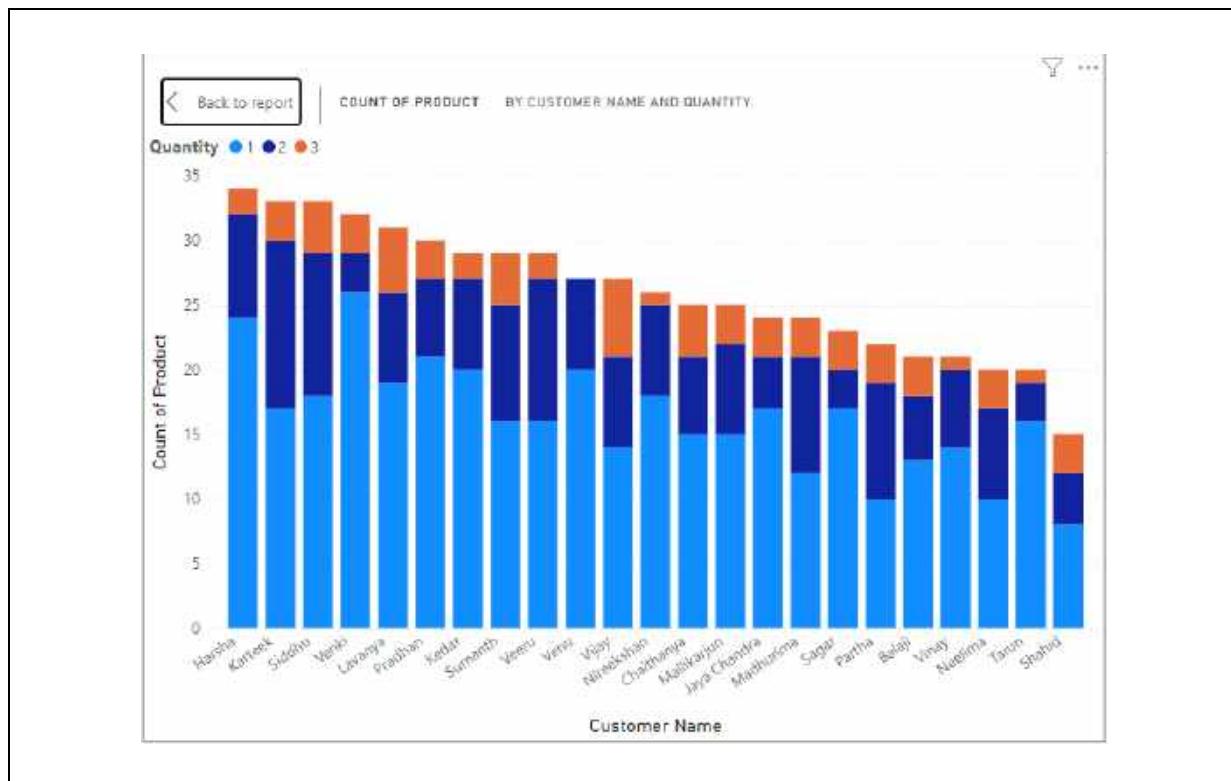
## Data Science – Power BI – Creating Charts

### 2. Stacked Column Chart

- ✓ A **Stacked Column Chart** in Power BI is a variation of the column chart where the values for each category are represented as **stacked segments** on a single vertical bar.
- ✓ Each segment of the bar represents a **subcategory's contribution** to the total value of that category.
- ✓ In the column chart, data values are displayed side-by-side, whereas they are stacked one over the other in the stacked chart.

#### 2.1. Steps to create stacked column chart

- ✓ Open Power BI Desktop.
- ✓ Load the data: **sales1.csv**
- ✓ Select the Stacked Column Chart from the Visualizations pane.
- ✓ Drag fields to:
  - X Axis: product
  - Y Axis: quantity
  - Legend: The subcategory (if having)
- ✓ Customize using the Format Visual pane to adjust colors, labels, and axis titles.



## Data Science – Power BI – Creating Charts

---

### More options

- ✓ Format your visual
- ✓ **Visual**
  - X-axis
    - Values -> On or off
      - Font
      - Colors
    - Title -> On or off
      - Title Text
      - Style
      - Font
      - Color
    - Layout
      - Width
  - Y-axis
    - Range(optional)
    - Values -> On or off
      - Font
      - Colors
      - Display units
    - Title -> On or off
      - Title Text
      - Style
      - Font
      - Color
  - Legend -> On or off
    - Options
      - Top left
      - Top to centre
      - Lot of other options too
    - Text
      - Font
      - Color

## Data Science – Power BI – Creating Charts

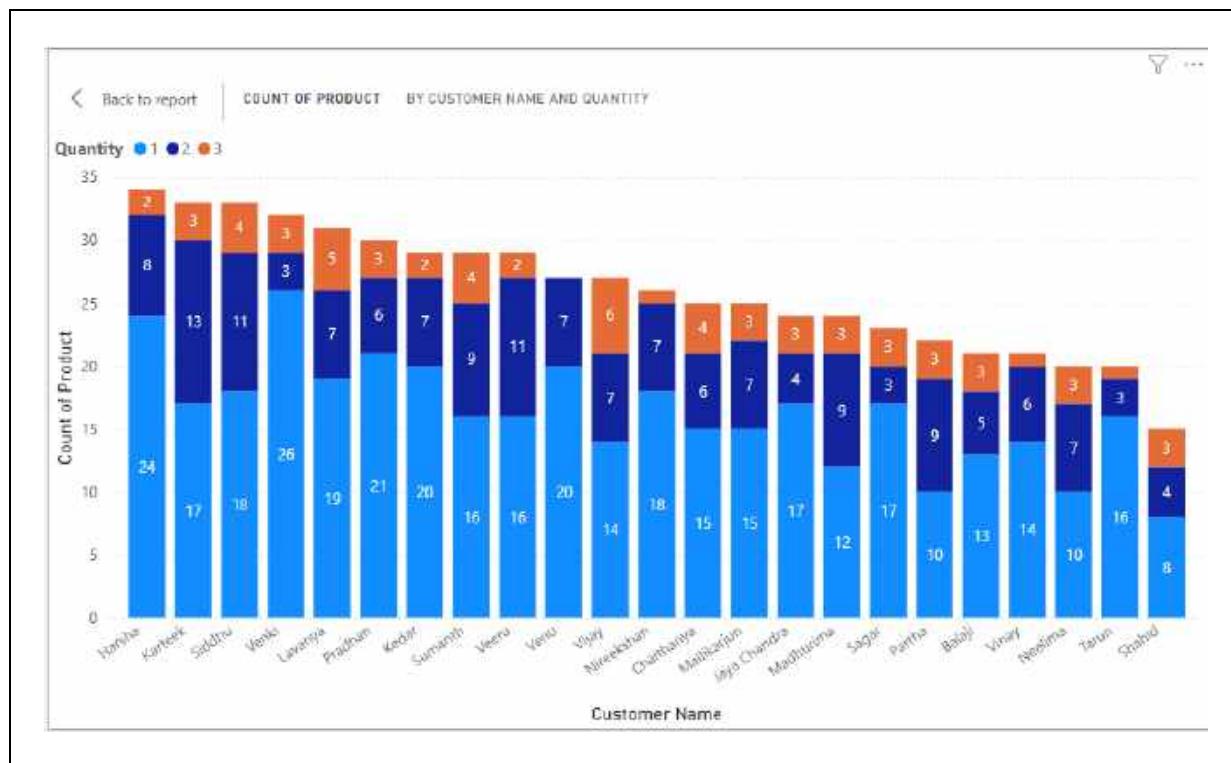
- Title -> On or off

- Title Text

- Data labels -> On or off
  - Total labels -> On or off

### ✓ General

- Title -> On or off



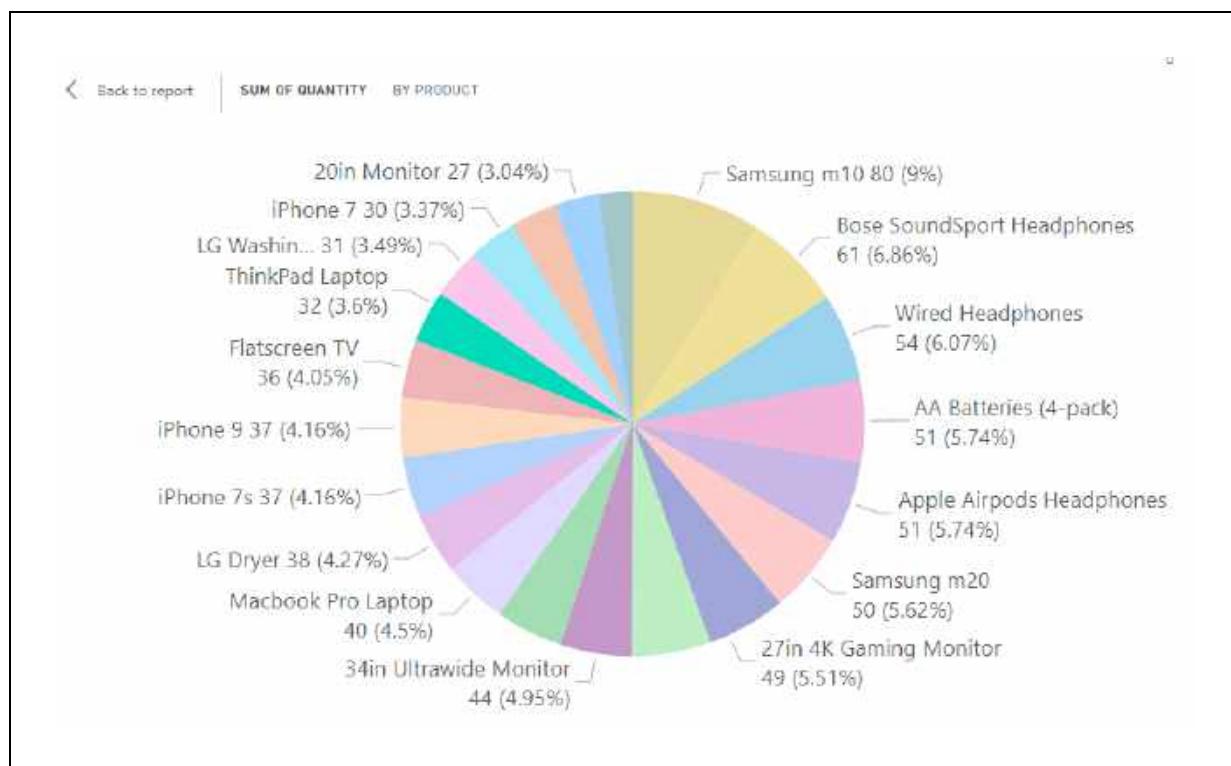
## Data Science – Power BI – Creating Charts

### 3. Pie Chart

- ✓ A pie chart in Power BI is a visual representation used to show the proportions or percentages of a whole data.
- ✓ It is shaped like a circle divided into slices.
- ✓ Each slice represents a category corresponds to its value compared to the total.
- ✓ For example, to see the total sales split by product.

#### 3.1. Steps to create stacked column chart

- ✓ Open Power BI Desktop.
- ✓ Load the data: **sales1.csv**
- ✓ Select the Pie chart from the Visualizations pane.
- ✓ Drag fields to:
  - Legend: product
  - Values: quantity
- ✓ Customize using the Format Visual pane to adjust colors, labels



## Data Science – Power BI – Creating Charts

---

### More options

- ✓ Format your visual
- ✓ Visual
  - Legend -> On or off
    - Options
      - Top left
      - Top to centre
      - Lot of other options too
    - Text
      - Font
      - Color
    - Title -> On or off
      - Title Text
  - Slices
    - Colors
- ✓ Detail labels -> On or off
  - Options
    - Position
      - Outside
      - Inside
      - Prefer outside
      - Prefer inside
    - Label contents
      - Category
      - Data values
      - Percent of values
      - Category, Data values
      - Category, Percent of values
      - Data values, Percent of values
      - All detail labels
  - Values
    - Font
    - Color
    - Background

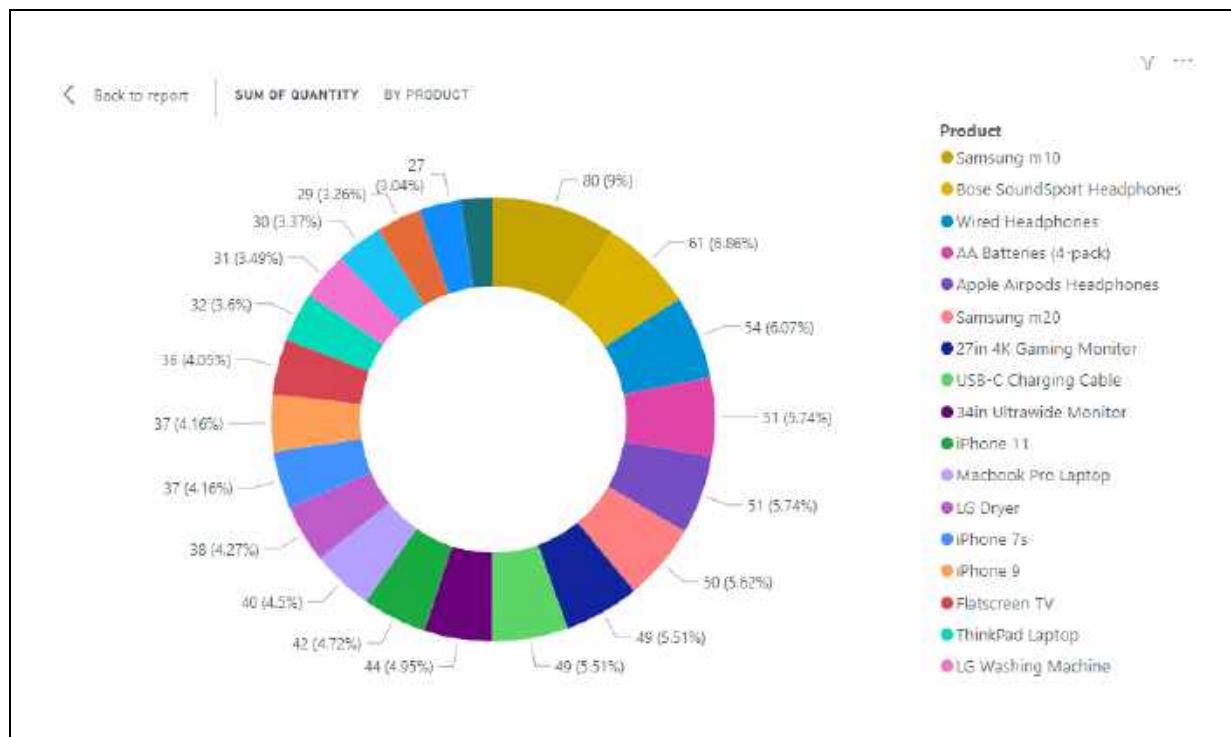
## Data Science – Power BI – Creating Charts

### 4. Donut Chart

- ✓ A donut chart is a circular chart, which could present values of a dataset in the form of slices of a donut.
- ✓ The donut chart is exactly the same as a pie chart.
- ✓ The only difference is pie chart has a circle, but a donut chart is in donut shape.

#### 4.1. Steps to create stacked column chart

- ✓ Open Power BI Desktop.
- ✓ Load the data: **sales1.csv**
- ✓ Select the Donut chart from the Visualizations pane.
- ✓ Drag fields to:
  - Legend: product
  - Values: quantity
- ✓ Customize using the Format Visual pane to adjust colors, labels



## Data Science – Power BI – Creating Charts

---

### More options

- ✓ Format your visual
- ✓ Visual
  - Legend -> On or off
    - Options
      - Top left
      - Top to centre
      - Lot of other options too
    - Text
      - Font
      - Color
    - Title -> On or off
      - Title Text
  - Slices
    - Colors
- ✓ Detail labels -> On or off
  - Options
    - Position
      - Outside
      - Inside
      - Prefer outside
      - Prefer inside
    - Label contents
      - Category
      - Data values
      - Percent of values
      - Category, Data values
      - Category, Percent of values
      - Data values, Percent of values
      - All detail labels
  - Values
    - Font
    - Color
    - Background

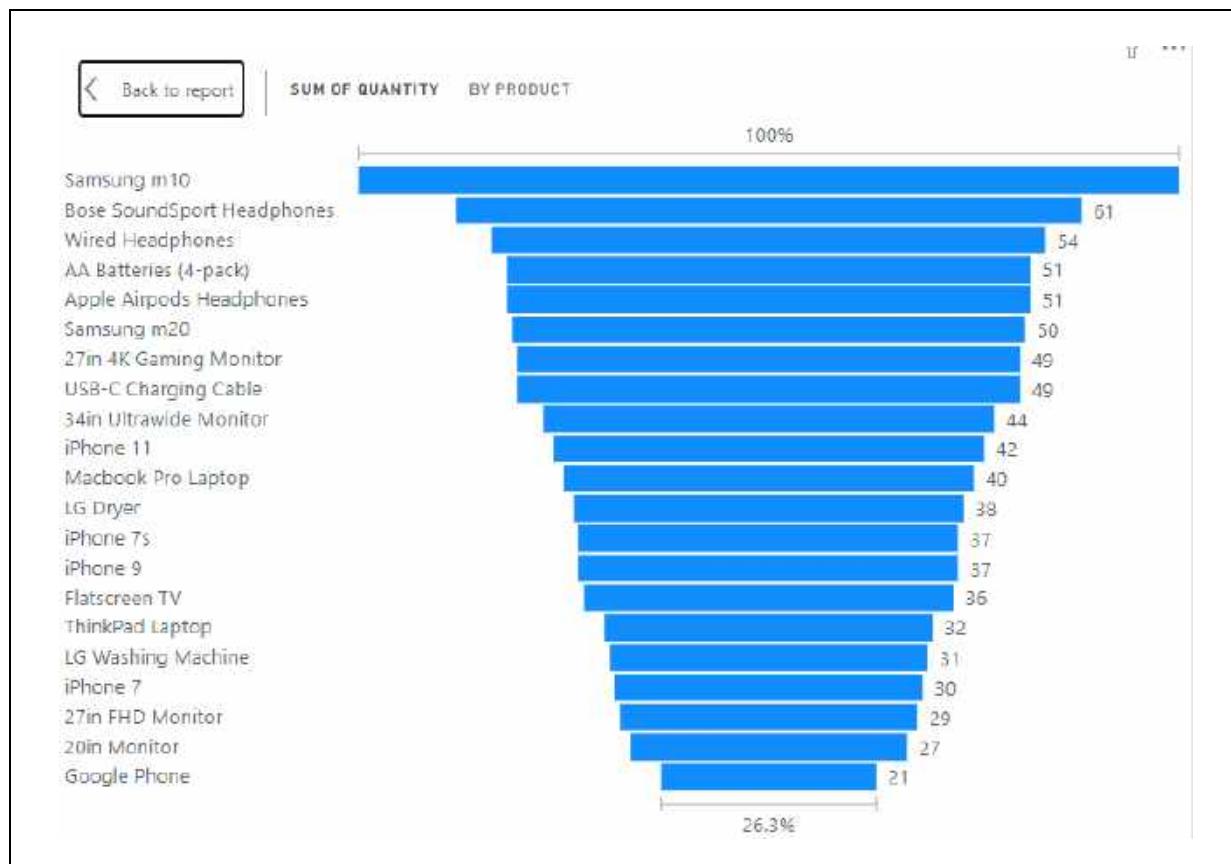
## Data Science – Power BI – Creating Charts

### 5. Funnel Chart

- ✓ A Funnel Chart in Power BI is a visual representation used to show data across stages in a process.
- ✓ It is particularly useful for visualizing workflows, sales pipelines, or any process that involves sequential stages.

#### 5.1. Steps to create Funnel chart

- ✓ Open Power BI Desktop.
- ✓ Load the data: **sales1.csv**
- ✓ Select the Funnel chart from the Visualizations pane.
- ✓ Drag fields to:
  - Category: product
  - Values: quantity
- ✓ Customize using the Format Visual pane to adjust colors, labels



## Data Science – Power BI – Creating Charts

---

### More options

- ✓ Colors
  - Default
  - Show all → On or off
- ✓ Data labels → On or off
  - Options
    - Position
    - Label contents
      - Data value
      - Percent of first
      - Percent of previous
      - Data value, percent of value
      - Data value, percent of previous
  - Values
    - Font      Size
    - Color
    - Display units
    - Value decimal places
    - Percentage decimal places
  - Background
    - Color
    - Transparency
- ✓ Category labels → On or off
  - Values
    - Font
    - Color
- ✓ Conversion labels → On or off
  - Values
    - Font
    - Color

## Data Science – Power BI – Creating Charts

### 6. Include and Exclude

- ✓ In Power BI, Include and Exclude are sensitive filtering options.
- ✓ By using this we can refine and customize data visualizations by dynamically adding or removing data points.

#### Include

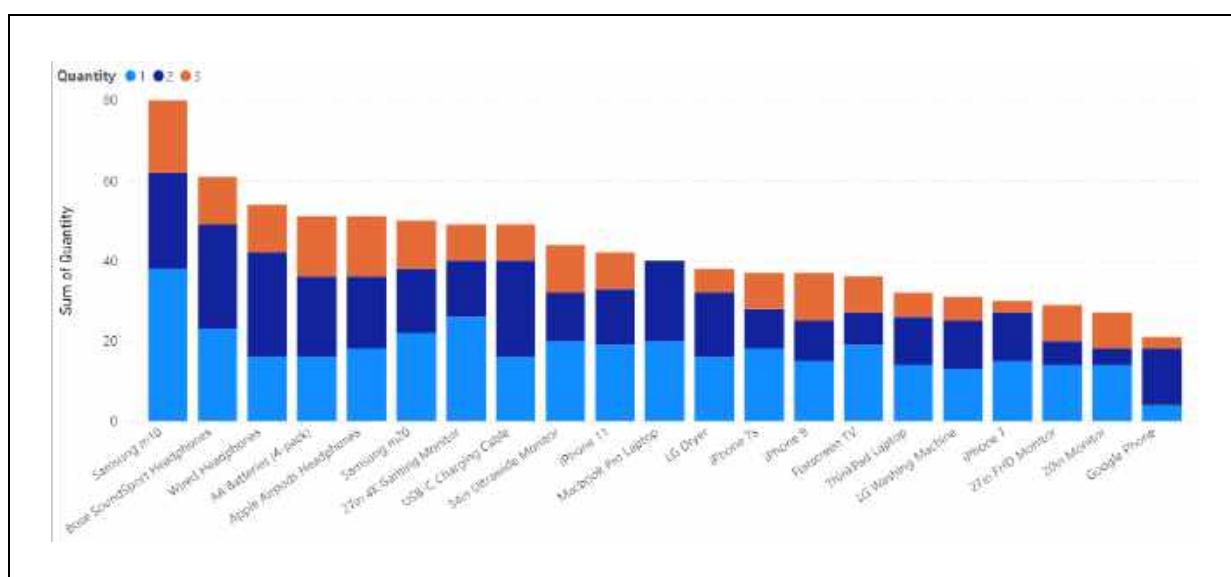
- ✓ Displays only the selected data points in a visual, helping focus on specific items.

#### Exclude

- ✓ Removes the selected data points from a visual, hiding irrelevant or unwanted items.

### 6.1. Steps to create stacked column chart

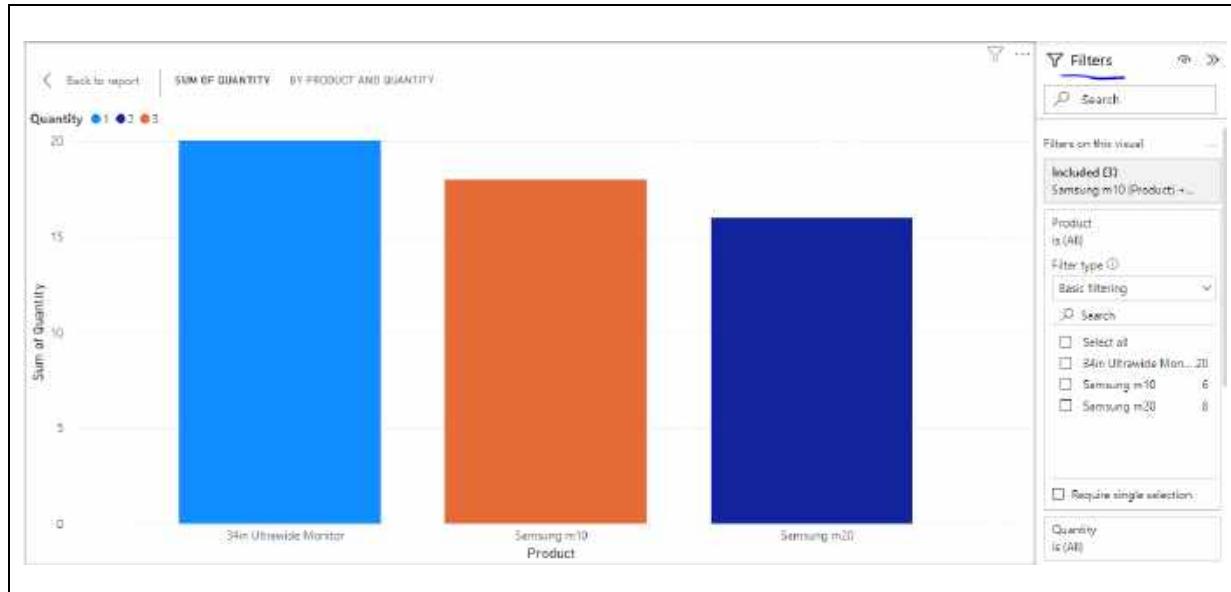
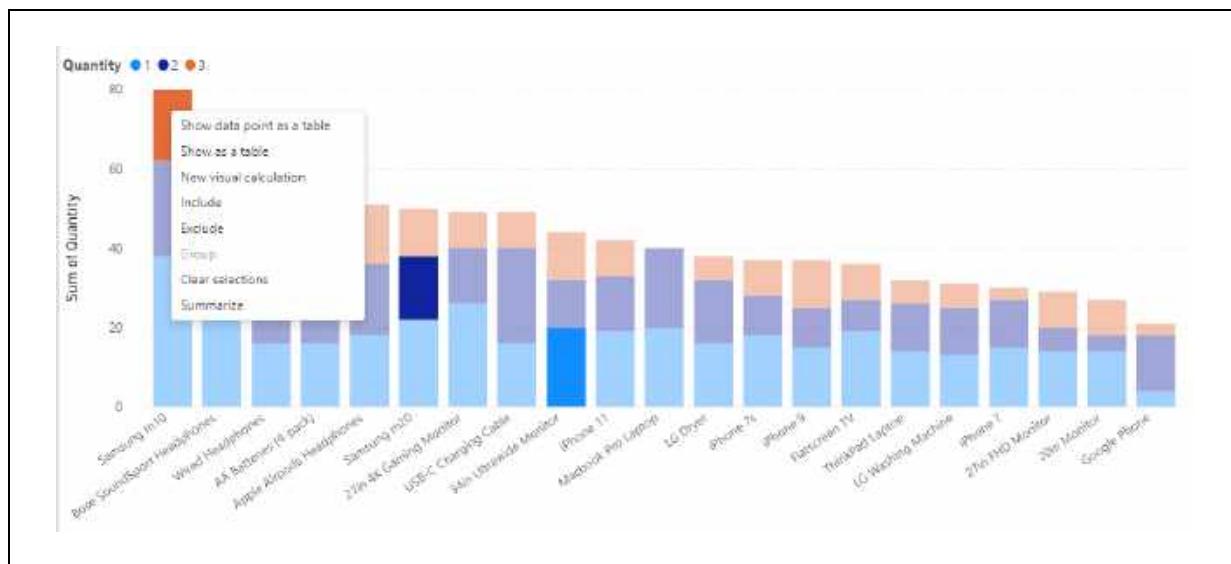
- ✓ Open Power BI Desktop.
- ✓ Load the data: **sales1.csv**
- ✓ Select the Stacked Column Chart from the Visualizations pane.
- ✓ Drag fields to:
  - X Axis: product
  - Y Axis: quantity
- ✓ Customize using the Format Visual pane to adjust colors, labels, and axis titles.



## Data Science – Power BI – Creating Charts

### Include

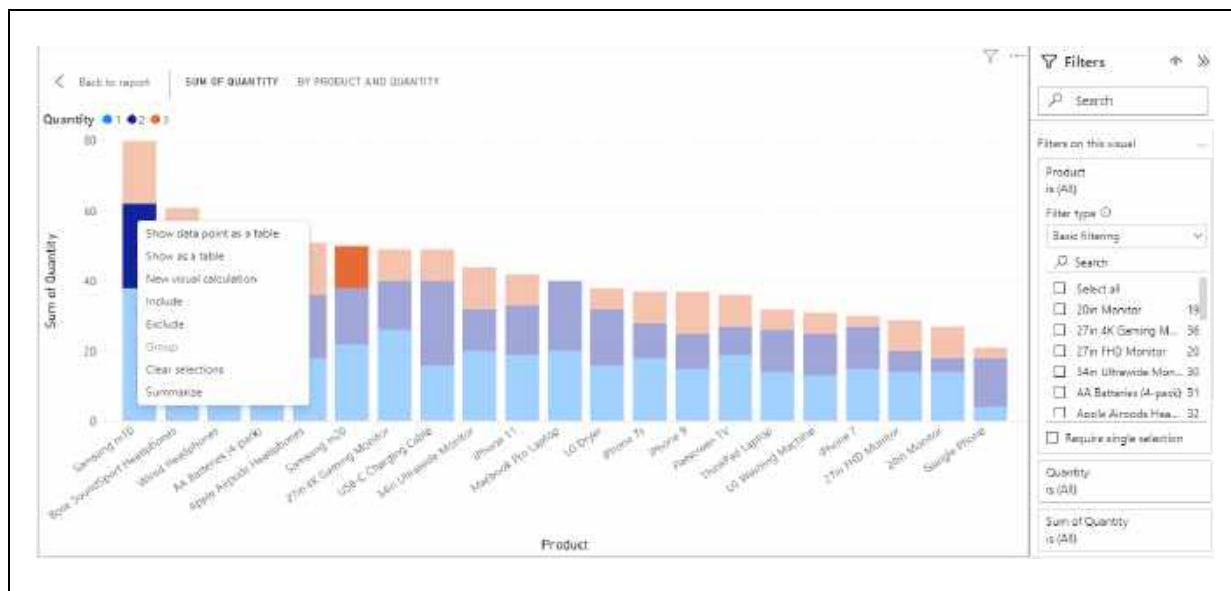
- ✓ Displays only the selected data points in a visual, helping focus on specific items.
- ✓ Select items (using control) to include and compare each other.
- ✓ Give right click and select include option.
- ✓ To delete the include option items, select Filter panel and click on x(close) button



## Data Science – Power BI – Creating Charts

### Exclude

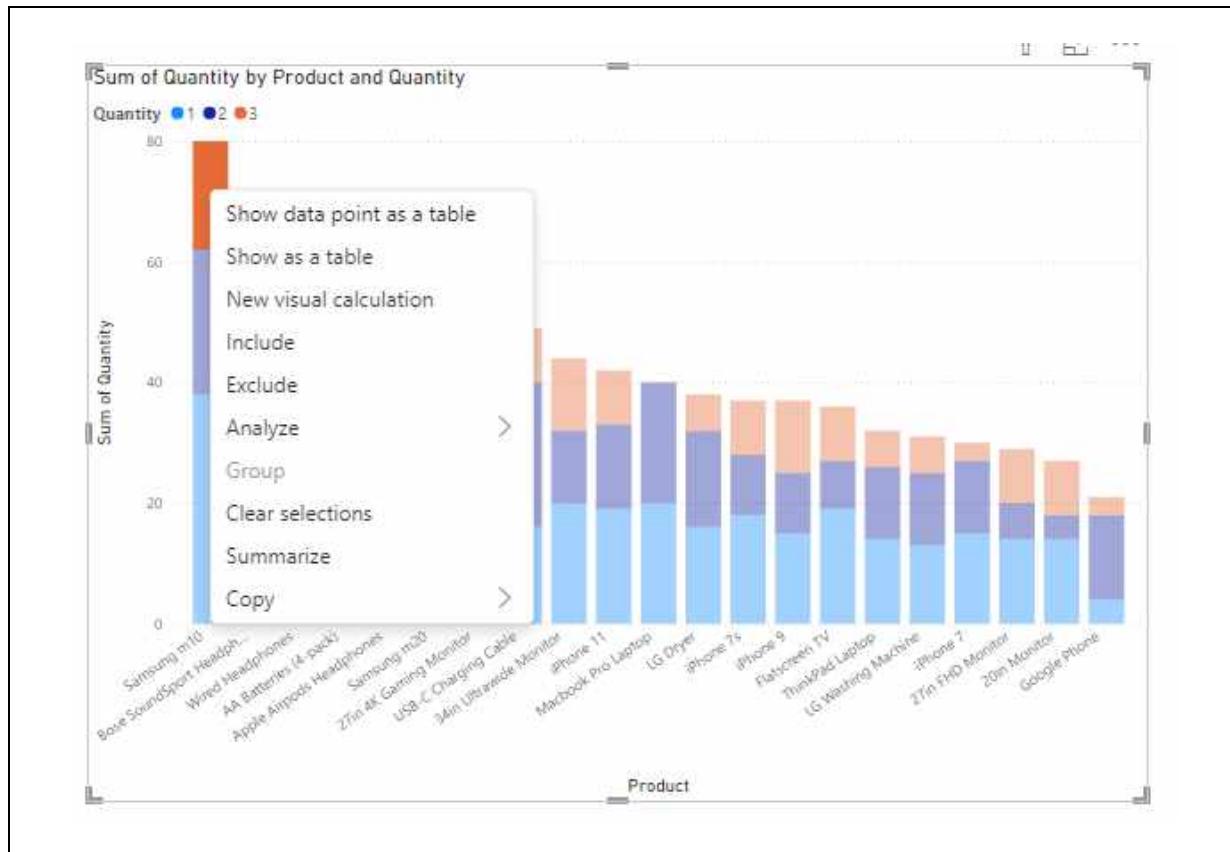
- ✓ Removes the selected data points from a visual, hiding irrelevant or unwanted items.
- ✓ Select items (using control) to exclude and compare others.
- ✓ Give right click and select exclude option.
- ✓ To delete the exclude option items, select Filter panel and click on x(close) button.



## Data Science – Power BI – Creating Charts

### 7. View Data and Export in CSV from Power BI

- ✓ We can View Data and Export in CSV from Power BI.
- ✓ After exporting we can do further analysis.



## 1. Computer Vision - ILSVRC DATASET, RESNET

---

### 1. Computer Vision – ILSVRC DATASET, RESNET

#### Contents

<b>1. Computer vision .....</b>	<b>2</b>
<b>2. Human &amp; computer vision systems.....</b>	<b>2</b>
<b>3. History of Computer Vision.....</b>	<b>3</b>
<b>4. How Computer Vision works? .....</b>	<b>4</b>
<b>5. Image means pixel to computer .....</b>	<b>5</b>
<b>6. Common computer vision tasks .....</b>	<b>6</b>
<b>7. Human Image recognition.....</b>	<b>7</b>
<b>8. Short Introduction: Convolutional Neural Network .....</b>	<b>8</b>
8.1. Convolutional Layer .....	9
8.2. Pooling layers.....	9
8.3. Fully Connected Layer.....	9
<b>9. Image recognition task by CNN.....</b>	<b>9</b>
<b>10. ImageNet .....</b>	<b>10</b>
10.1. The Birth of ImageNet.....	10
10.2. Dataset Information.....	10
10.3. Dataset Information.....	10
10.4. The Birth of the Competition: ILSVRC.....	11
<b>11. The problem with deep neural networks.....</b>	<b>11</b>
<b>12. If we increase more layers then.....</b>	<b>11</b>
<b>13. Vanishing / Exploding Gradients .....</b>	<b>12</b>
<b>14. How to solve Vanishing / Exploding Gradients problem? .....</b>	<b>12</b>
<b>15. History of ResNet .....</b>	<b>12</b>
15.1. ResNet Paper.....	12
15.2. Winner of ILSVRC 2015 .....	12
<b>16. ResNet Introduction .....</b>	<b>13</b>
<b>17. ResNet Example.....</b>	<b>14</b>

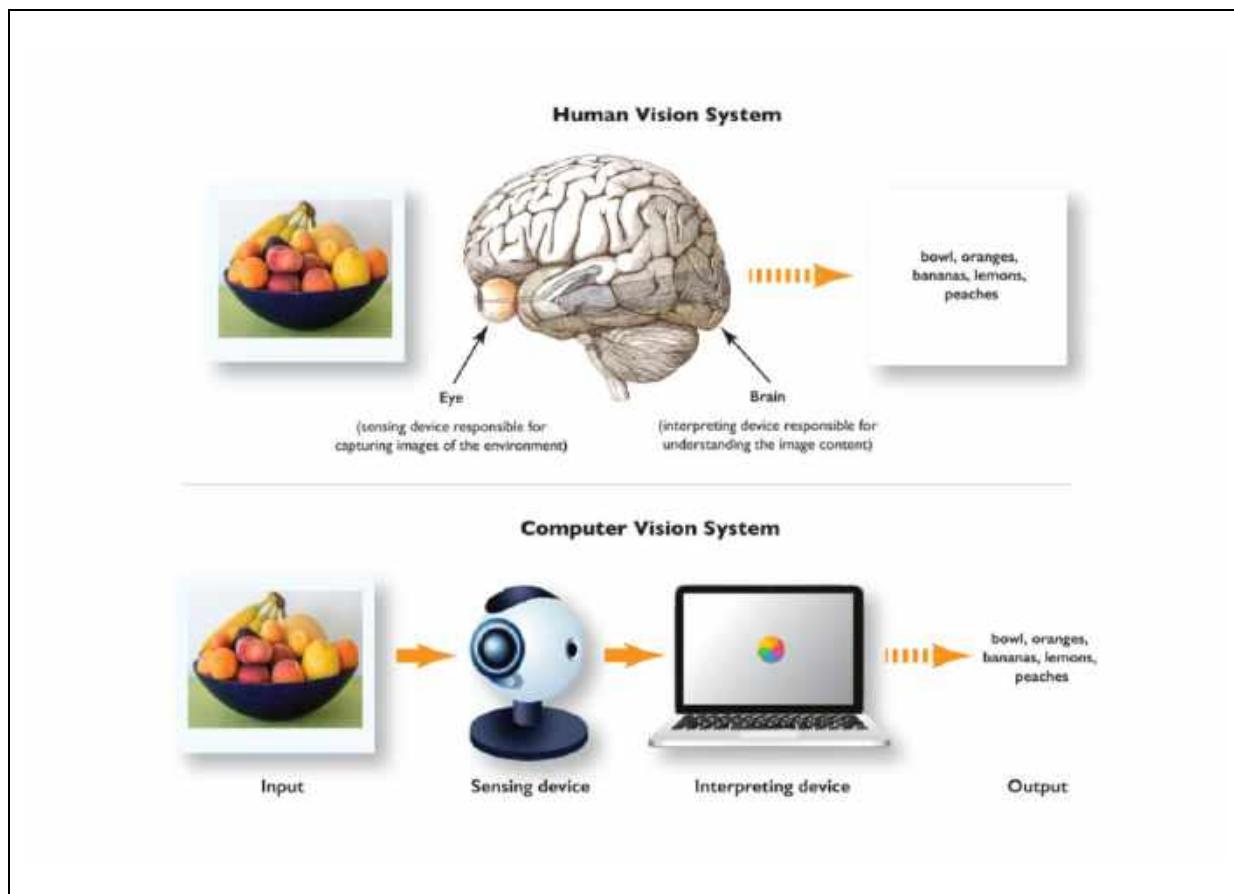
## 1. Computer Vision - ILSVRC DATASET, RESNET

### 1. Computer Vision – ILSVRC DATASET, RESNET

#### 1. Computer vision

- ✓ Computer Vision is a subfield of Deep Learning and Artificial Intelligence.
- ✓ By using this human can teach computers to see and interpret the world around them.

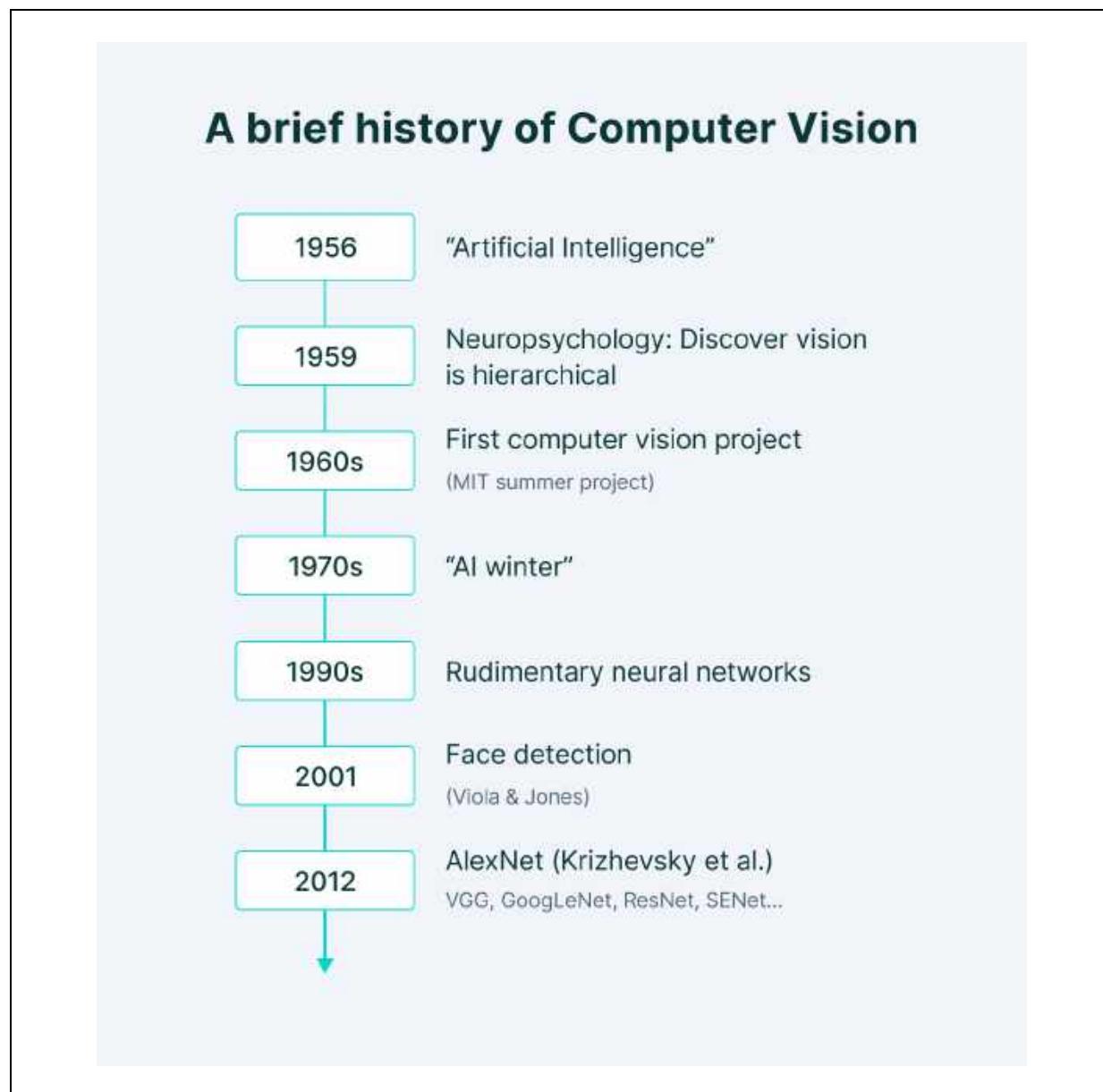
#### 2. Human & computer vision systems



## 1. Computer Vision - ILSVRC DATASET, RESNET

### 3. History of Computer Vision

- ✓ Machine vision, inspired by animal vision since 1959, evolved with advances in image technology.
- ✓ Key milestones: AI in the 1960s, OCR (Optical Character Recognition) in 1974, and complex topics by the 2000s.
  - Object identification
  - Facial recognition
  - Image Segmentation
  - Image Classification



## 1. Computer Vision - ILSVRC DATASET, RESNET

### 4. How Computer Vision works?

- ✓ There are mainly three steps,
  - Acquiring an image
  - Processing the image
  - Understanding the image

### How does Computer Vision work?

The diagram consists of three circular icons, each containing a teal-colored icon representing a step in the process. To the right of each icon is a title and a detailed description of the step.

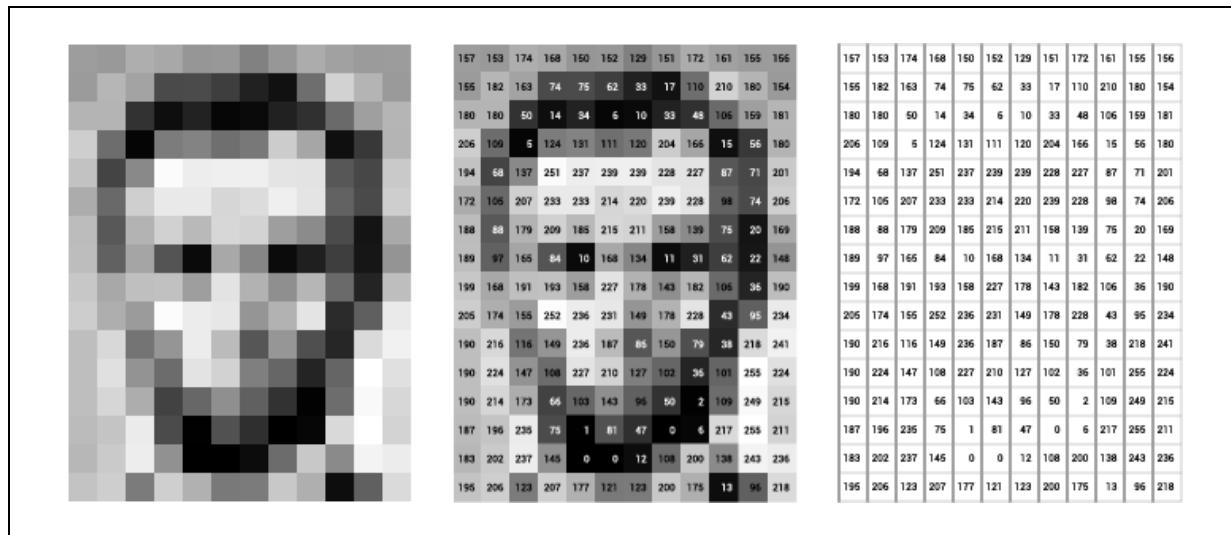
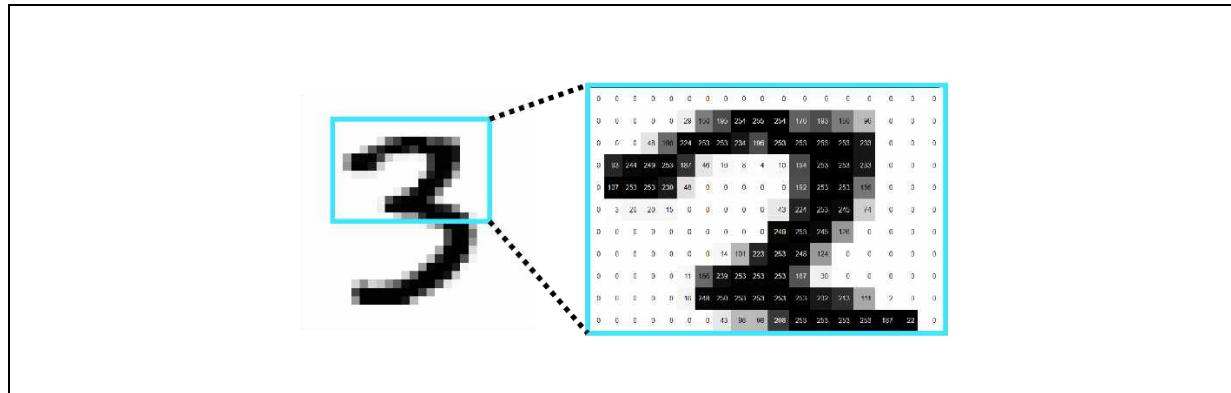
- Acquiring an image**  
Images, even large sets, can be acquired in real-time through video, photos or 3D technology for analysis.
- Processing the image**  
Deep learning models automate much of this process, but the models are often trained by first being fed a thousand of labeled or pre-identified images.
- Understanding the image**  
The final step is the interpretative step, where an object is identified or classified.

## 1. Computer Vision - ILSVRC DATASET, RESNET

---

### 5. Image means pixel to computer

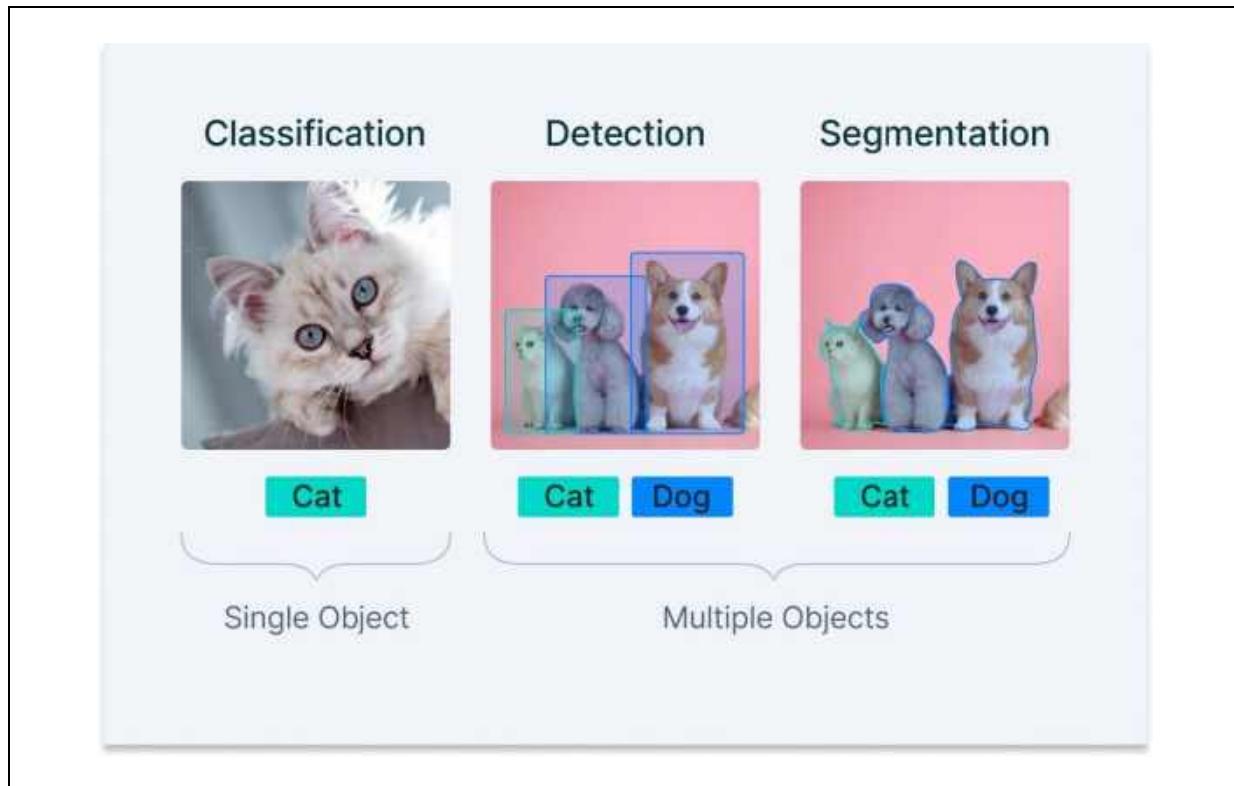
- ✓ An image is group of pixels.
- ✓ Computers process images as pixel arrays.
- ✓ Computer vision uses,
  - Linear algebra for simple tasks and convolutions with pooling for complex tasks.



## 1. Computer Vision - ILSVRC DATASET, RESNET

### 6. Common computer vision tasks

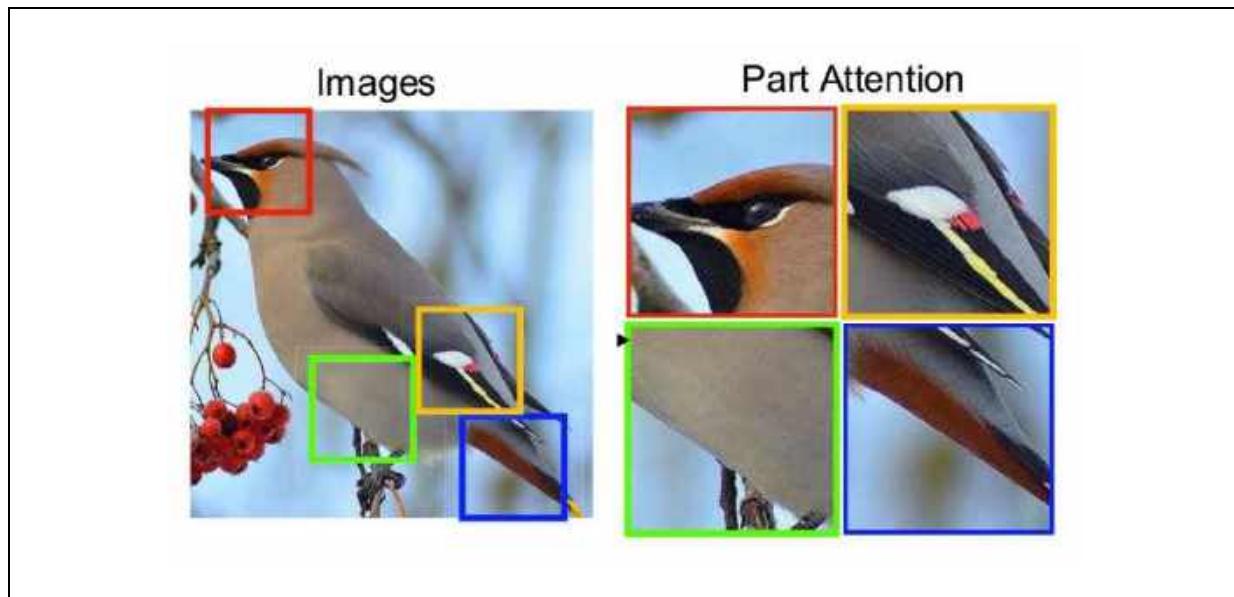
- ✓ Classification
- ✓ Detection
- ✓ Segmentation
- ✓ Face recognition
- ✓ Edge detection & etc



## 1. Computer Vision - ILSVRC DATASET, RESNET

### 7. Human Image recognition

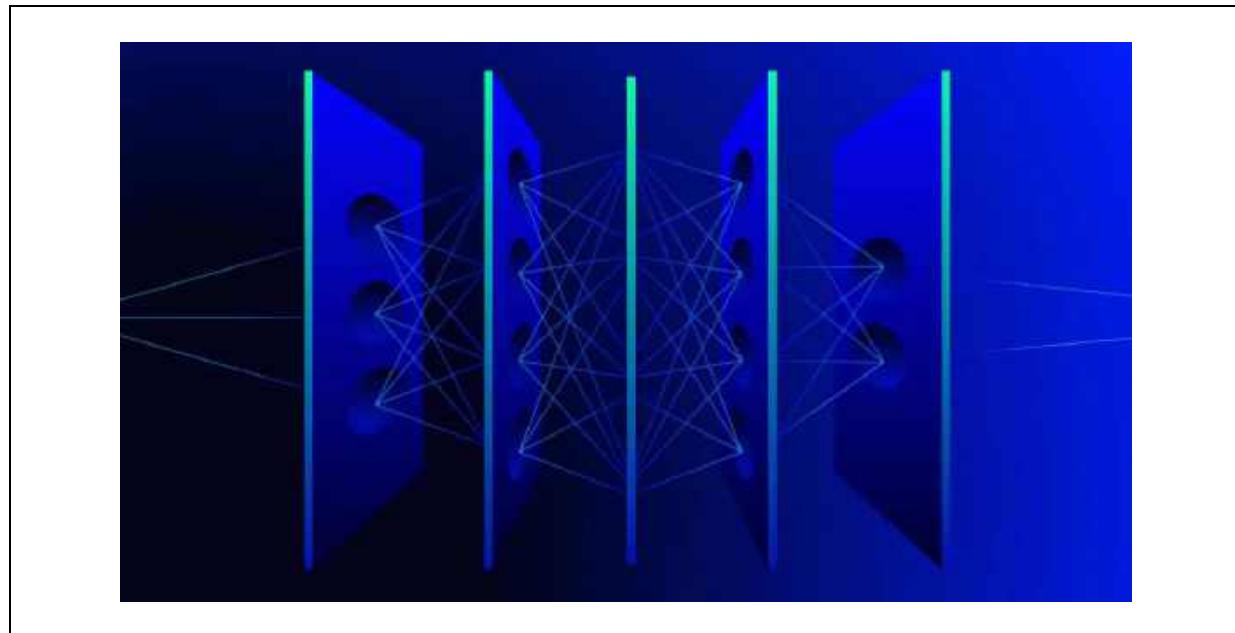
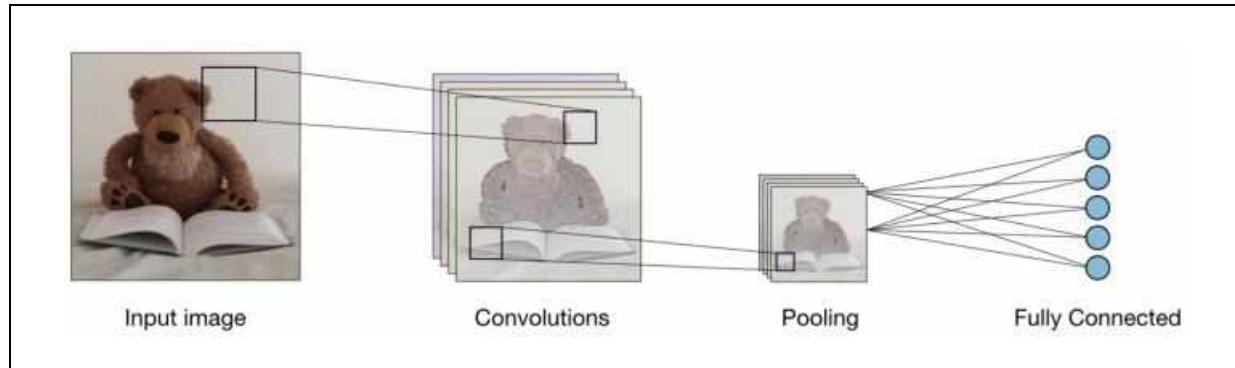
- ✓ Image recognition means, a computer or machine can see/observe the images.
- ✓ This is something like, **how our eye works.**
  - If we see an image then automatically it **split** into parts of image.
  - In next step our eyes can **analyse** sub-parts individually.
  - By assembling these parts, our eyes **process** and **interpret** the image.
- ✓ The same process was implemented in **CNN** or Convolutional Neural Network.



## 1. Computer Vision - ILSVRC DATASET, RESNET

### 8. Short Introduction: Convolutional Neural Network

- ✓ CNN is a type of deep learning model.
- ✓ This is commonly used for image recognition and classification tasks.
- ✓ The main parts in CNN,
  - Convolution Layer
  - Pooling Layer
  - Fully-Connected Layer



## 1. Computer Vision - ILSVRC DATASET, RESNET

---

### 8.1. Convolutional Layer

- ✓ In this layer, the input image is convolved (In maths: combine one function or series) with a set of learnable filters, also known as kernels.
- ✓ Every filter extracts different features from the input image by performing element-wise multiplication and summation operations.

### 8.2. Pooling layers

- ✓ Pooling layers reduce the size of the input by taking either the maximum or average value from small regions.
- ✓ This makes the network faster and helps prevent overfitting.

### 8.3. Fully Connected Layer

- ✓ Connects every neuron in the previous layer to every neuron in the next layer.
- ✓ This layer learns global features.

## 9. Image recognition task by CNN

- ✓ CNN having layers to recognize the image.
- ✓ Every layer can learn and delivery each task as,
  - The first layer can learn to detect basic edges and shapes.
  - The second layer can recognize these more detailed features.
  - The third layer can recognize parts of objects etc.
- ✓ This learning process helps the network to build a detailed understanding of the data.

## 1. Computer Vision - ILSVRC DATASET, RESNET

---

### 10. ImageNet

#### 10.1. The Birth of ImageNet

- ✓ Once upon a time, in the mid-2000s, a brilliant researcher named Fei-Fei Li.
  - <https://profiles.stanford.edu/fei-fei-li>
  - [https://en.wikipedia.org/wiki/Fei-Fei\\_Li](https://en.wikipedia.org/wiki/Fei-Fei_Li)
- ✓ She had an idea like, just as humans learn by seeing countless objects and scenes throughout their lives.
- ✓ Her idea to make computers see and understand the world like humans.
- ✓ So, here needed a vast amount of data.
- ✓ She and her team aimed to create ImageNet.
- ✓ Started gathering images and created dataset
- ✓ The ImageNet is a very large collection of a dataset.
  - <https://image-net.org/index.php>

#### 10.2. Dataset Information

- ✓ Gathering images, labelling them accurately was really a challenging task.
- ✓ Amazon Mechanical Turk thousands of team members manually verify and annotate the images.

#### 10.3. Dataset Information

- ✓ This dataset contains,
  - More than 14 million images
  - More than 22 thousand groups or classes.
  - More than 1 million images that have bounding box annotations

## 1. Computer Vision - ILSVRC DATASET, RESNET

---

### 10.4. The Birth of the Competition: ILSVRC

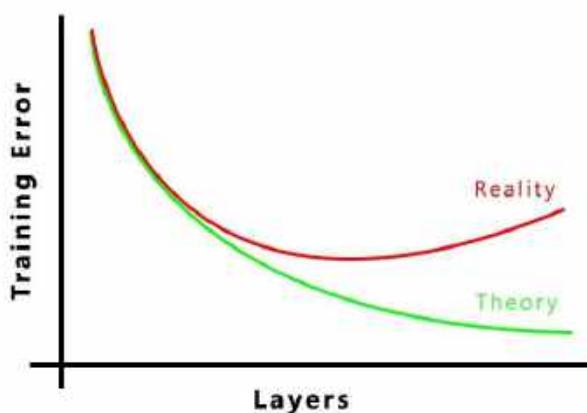
- ✓ ILSVRC the full form is,
  - ImageNet Large Scale Visual Recognition Challenge
- ✓ In 2010, Fei-Fei Li and her team introduced the ImageNet Large Scale Visual Recognition Challenge (ILSVRC).
- ✓ This annual competition invited researchers from around the world to develop algorithms that could classify and detect objects within the ImageNet dataset.
- ✓ It quickly became the benchmark for measuring progress in computer vision.

### 11. The problem with deep neural networks

- ✓ To get the better results, the architecture used to add more and more layers, but this can lead to vanishing gradient issues.

### 12. If we increase more layers then...

- ✓ If we increase more layers (deep networks) then we may get the problem of vanishing/exploding gradients.



## 1. Computer Vision - ILSVRC DATASET, RESNET

---

### 13. Vanishing / Exploding Gradients

- ✓ If we add more layers to the neural network then one of the problems is like, gradients can become too small (vanishing) or too large (exploding).
- ✓ Due to this Neural Network cannot learn effectively.

### 14. How to solve Vanishing / Exploding Gradients problem?

- ✓ We can solve this problem by using **ResNet**.

### 15. History of ResNet

- ✓ ResNet was implemented by Kaiming He and his team, in 2015.
- ✓ This research team achieved top results for object detection and object detection with localization tasks.

#### 15.1. ResNet Paper

- ✓ This research paper written for Deep Residual Learning for Image Recognition.
- ✓ Find the below link for research paper,
  - <https://arxiv.org/abs/1512.03385>

#### 15.2. Winner of ILSVRC 2015

- ✓ ResNet won the champion for ILSVRC 2015 in,
  - Image classification, detection, and localization

## 1. Computer Vision - ILSVRC DATASET, RESNET

---

### 16. ResNet Introduction

- ✓ ResNet stands for Residual Network.
- ✓ It's a type of deep learning model.
- ✓ The main goal of ResNet (Residual Network) is to address the vanishing gradient problem.
- ✓ ResNet proves that, it is easy to train very deep neural networks.

## 1. Computer Vision - ILSVRC DATASET, RESNET

---

### 17. ResNet Example

**Program Name**

ResNet Example

**Name**

demo1.py

```

import numpy as np

from keras.preprocessing import image
from keras.applications import ResNet50
from keras.applications.resnet50 import preprocess_input
from keras.applications.resnet50 import decode_predictions

model = ResNet50(weights = 'imagenet')

def load_and_preprocess_image(img_path):
    img = image.load_img(img_path, target_size = (224, 224))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis = 0)
    img_array = preprocess_input(img_array)

    return img_array

img_path = '1.jpeg'
img_array = load_and_preprocess_image(img_path)
predictions = model.predict(img_array)
decoded_predictions = decode_predictions(predictions, top = 1)[0]

for i, (imagenet_id, label, score) in
enumerate(decoded_predictions):
    print(f"{i + 1}: {label} ({score:.2f})")

```

**Output**

goldfish

## 2. Computer Vision - Features

---

### 2. Computer Vision – Features

#### Contents

1. Computer Vision .....	2
2. Computer Vision and Image Processing .....	2
3. Computer vision applications .....	3
4. Automatic Feature Extraction .....	3
5. Reuse the models .....	3
6. Superior Performance .....	3

## 2. Computer Vision - Features

---

### 2. Computer Vision – Features

#### 1. Computer Vision

- ✓ Computer vision is the automated extraction of information from images.
- ✓ The goal of computer vision is to understand the content of digital images.
- ✓ Computer vision is a field of study focused on the problem of helping computers to see.

#### 2. Computer Vision and Image Processing

- ✓ Computer vision is different from image processing.
- ✓ Image processing is the process of creating a new image from an existing image
- ✓ Computer vision system may require image processing.
- ✓ Examples of image processing
  - Increase image brightness or change color.
  - Cropping the images
  - Removing digital noise from an image means low light levels.

## 2. Computer Vision - Features

---

### 3. Computer vision applications

- ✓ Object Classification
  - Broad category of object in image
- ✓ Object Identification
  - Type of a given object in image
- ✓ Object Verification
  - Is the object existing in images?
- ✓ Object Detection
  - Where are the objects in images?
- ✓ Object Landmark Detection
  - Key points for the object in the image
- ✓ Object Segmentation
  - What pixels belong to the object in the image
- ✓ Object Recognition
  - What objects are in this photograph and where are they?

### 4. Automatic Feature Extraction

- ✓ Features can be automatically learned and extracted from raw image data.

### 5. Reuse the models

- ✓ We can reuse the existing models

### 6. Superior Performance

- ✓ Computer vision is very good to get the best results

### 3. Computer Vision – Load & work with images

---

#### 3. Computer Vision – Load & work with images

##### Contents

1. Loading the images in different ways.....	2
--	---

### 3. Computer Vision – Load & work with images

---

#### 3. Computer Vision – Load & work with images

##### 1. Loading the images in different ways

- ✓ Pillow is open-source python library.
- ✓ By using this we can load and manipulate images.

```
pip install Pillow
```

### 3. Computer Vision – Load & work with images

---

**Program Name** Loading image by using pillow  
demo1.py

```
from PIL import Image  
  
image = Image.open("opera_house.jpg")  
  
image.show()
```

**output**



### 3. Computer Vision – Load & work with images

**Program Name** Loading image by using matplotlib  
demo2.py

```
from matplotlib import image
from matplotlib import pyplot

data = image.imread("opera_house.jpg")

pyplot.imshow(data)
pyplot.show()
```

**output**



### 3. Computer Vision – Load & work with images

**Program Name** Converting normal images to grayscale  
demo3.py

```
from PIL import Image

image = Image.open("opera_house.jpg")

gs_image = image.convert(mode = "L")

image.show()
gs_image.show()
```

**output**



### 3. Computer Vision – Load & work with images

**Program Name** Converting normal images to grayscale, save image demo4.py

```
from PIL import Image

image = Image.open("opera_house.jpg")

gs_image = image.convert(mode = "L")

gs_image.save("opera_house_grayscale.jpg")

image2 = Image.open("opera_house_grayscale.jpg")

image2.show()
```

**output**



### 3. Computer Vision – Load & work with images

---

**Program Name** Resize the images  
demo5.py

```
from PIL import Image

image = Image.open("opera_house.jpg")

image.thumbnail((100,100))

image.show()
```

**output**



### 3. Computer Vision – Load & work with images

**Program Name** Flipping the image  
demo6.py

```
from PIL import Image
from matplotlib import pyplot

image = Image.open("opera_house.jpg")

hoz_flip = image.transpose(Image.FLIP_LEFT_RIGHT)
ver_flip = image.transpose(Image.FLIP_TOP_BOTTOM)

pyplot.subplot(311)
pyplot.imshow(image)

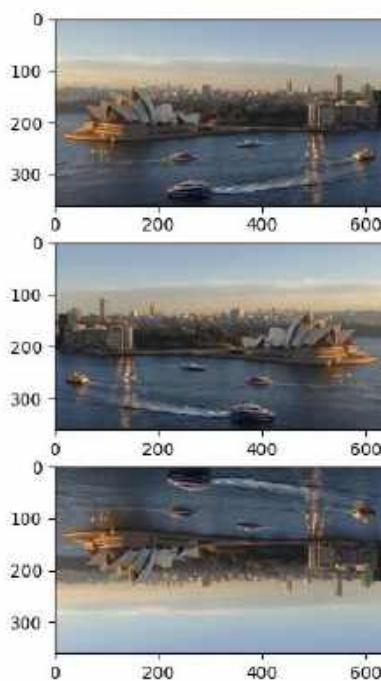
pyplot.subplot(312)
pyplot.imshow(hoz_flip)

pyplot.subplot(313)
pyplot.imshow(ver_flip)

pyplot.show()
```

**output**

### 3. Computer Vision – Load & work with images



### 3. Computer Vision – Load & work with images

**Program Name** Rotate the images  
demo7.py

```
from PIL import Image
from matplotlib import pyplot

image = Image.open("opera_house.jpg")

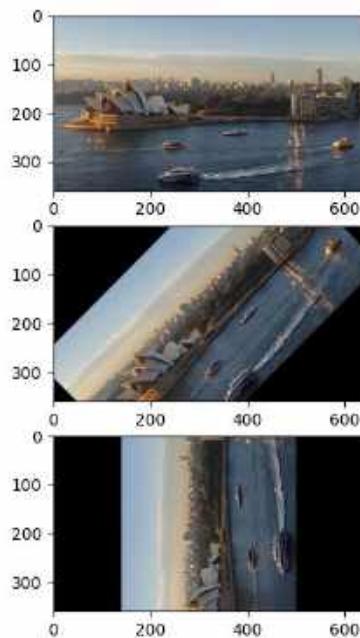
pyplot.subplot(311)
pyplot.imshow(image)

pyplot.subplot(312)
pyplot.imshow(image.rotate(45))

pyplot.subplot(313)
pyplot.imshow(image.rotate(90))

pyplot.show()
```

**output**



## 4. Computer Vision – Load images with keras

---

### 4. Computer Vision – Load images with keras

#### Contents

1. Loading the images in different ways.....	2
--	---

## 4. Computer Vision – Load images with keras

---

### 4. Computer Vision – Load images with keras

#### 1. Loading the images with keras

- ✓ We can load & save the images with keras package

## 4. Computer Vision – Load images with keras

**Program Name** Loading image by using keras  
demo1.py

```
from tensorflow.keras.utils import load_img  
  
img = load_img("opera_house.jpg")  
  
img.show()
```

**output**



## 4. Computer Vision – Load images with keras

**Program Name** Loading & saving the image by using keras  
demo2.py

```
from tensorflow.keras.utils import load_img
from tensorflow.keras.utils import save_img
from tensorflow.keras.utils import img_to_array

img = load_img("opera_house.jpg", color_mode= "grayscale")
img_array = img_to_array(img)

save_img("bondi_beach_grayscale.jpg", img_array)
img = load_img("bondi_beach_grayscale.jpg")

img.show()
```

**output**



## 5. Computer Vision – Image Data Augmentation in Keras

---

### 5. Computer Vision – Image Data Augmentation in Keras

#### Contents

1. Image Data Augmentation in Keras.....	2
2. Different type of Image Data Augmentation in Keras .....	2
3. ImageDataGenerator class.....	2
4. Random Brightness Augmentation.....	11
5. Random Zoom Augmentation.....	14

## 5. Computer Vision – Image Data Augmentation in Keras

---

### 5. Computer Vision – Image Data Augmentation

#### 1. Image Data Augmentation in Keras

- ✓ Image data augmentation is a technique, by using this we can create new images.
- ✓ It helps to increase the training dataset.
- ✓ If more data/images then model will gives good accuracy.

#### 2. Different type of Image Data Augmentation in Keras

- ✓ Image Augmentation With ImageDataGenerator
- ✓ Horizontal and Vertical Shift Augmentation
- ✓ Horizontal and Vertical Flip Augmentation
- ✓ Random Rotation Augmentation
- ✓ Random Brightness Augmentation
- ✓ Random Zoom Augmentation

#### 3. ImageDataGenerator class

- ✓ **ImageDataGenerator** is a predefined class
  - Here keyword arguments plays important role.
- ✓ **Keyword arguments.**
  - width\_shift\_range
  - height\_shift\_range
  - horizontal\_flip
  - rotation\_range

## 5. Computer Vision – Image Data Augmentation in Keras

Program Name Image data augmentation, **width\_shift\_range**  
demo1.py

```
from numpy import expand_dims
from tensorflow.keras.utils import load_img
from tensorflow.keras.utils import img_to_array
from keras.preprocessing.image import ImageDataGenerator
from matplotlib import pyplot

img = load_img("bird.jpg")
data = img_to_array(img)
samples = expand_dims(data, 0)

datagen = ImageDataGenerator(width_shift_range = [-80, 80])

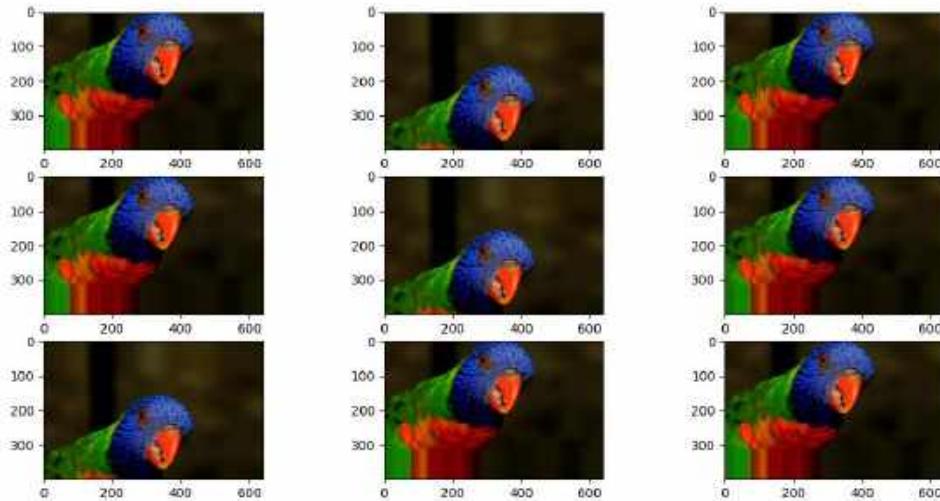
it = datagen.flow(samples, batch_size = 1)

for i in range(9):
    pyplot.subplot(330 + 1 + i)
    batch = it.next()
    image = batch[0].astype("uint8")
    pyplot.imshow(image)

pyplot.show()
```

## 5. Computer Vision – Image Data Augmentation in Keras

output



## 5. Computer Vision – Image Data Augmentation in Keras

Program Name      Image data augmentation, **height\_shift\_range**  
                  demo2.py

```
from numpy import expand_dims
from tensorflow.keras.utils import load_img
from tensorflow.keras.utils import img_to_array
from keras.preprocessing.image import ImageDataGenerator
from matplotlib import pyplot

img = load_img("bird.jpg")
data = img_to_array(img)
samples = expand_dims(data, 0)

datagen = ImageDataGenerator(height_shift_range = 0.8)

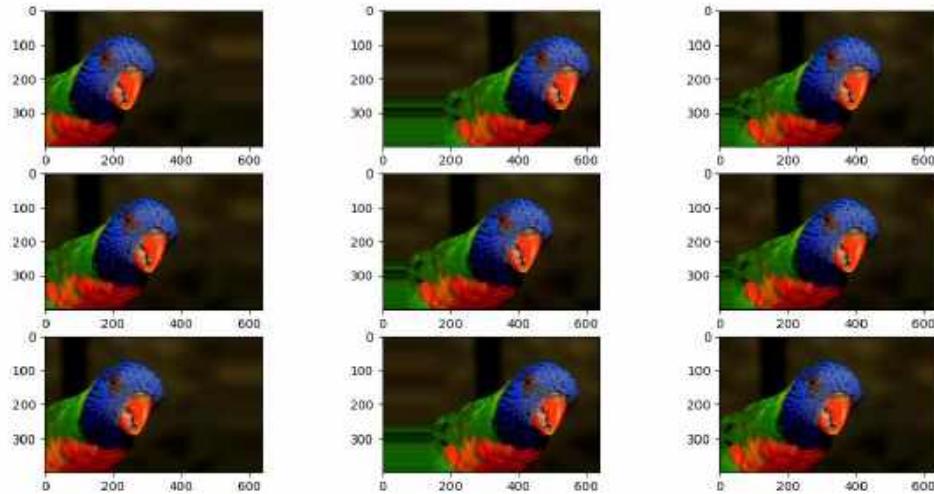
it = datagen.flow(samples, batch_size = 1)

for i in range(9):
    pyplot.subplot(330 + 1 + i)
    batch = it.next()
    image = batch[0].astype("uint8")
    pyplot.imshow(image)

pyplot.show()
```

## 5. Computer Vision – Image Data Augmentation in Keras

output



## 5. Computer Vision – Image Data Augmentation in Keras

Program Name      Image data augmentation, **horizontal\_flip**  
                  demo3.py

```
from numpy import expand_dims
from tensorflow.keras.utils import load_img
from tensorflow.keras.utils import img_to_array
from keras.preprocessing.image import ImageDataGenerator
from matplotlib import pyplot

img = load_img("bird.jpg")
data = img_to_array(img)
samples = expand_dims(data, 0)

datagen = ImageDataGenerator(horizontal_flip = True)

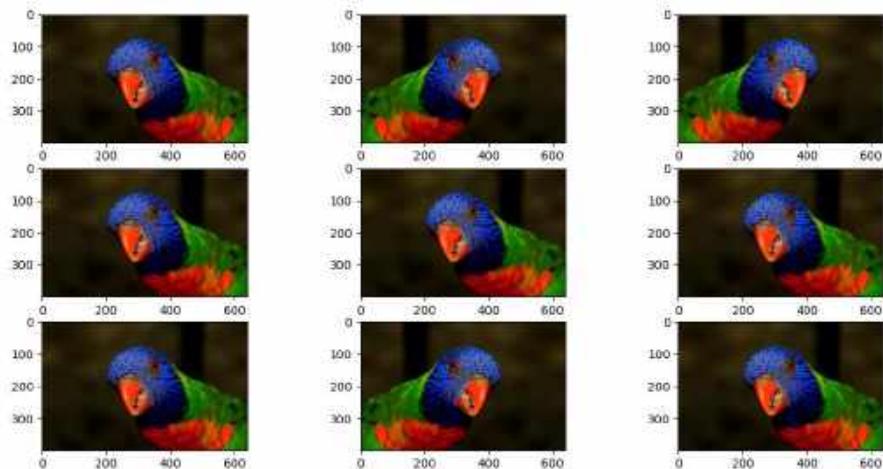
it = datagen.flow(samples, batch_size = 1)

for i in range(9):
    pyplot.subplot(330 + 1 + i)
    batch = it.next()
    image = batch[0].astype("uint8")
    pyplot.imshow(image)

pyplot.show()
```

## 5. Computer Vision – Image Data Augmentation in Keras

output



## 5. Computer Vision – Image Data Augmentation in Keras

Program Name      Image data augmentation, **rotation\_range**  
                      demo4.py

```
from numpy import expand_dims
from tensorflow.keras.utils import load_img
from tensorflow.keras.utils import img_to_array
from keras.preprocessing.image import ImageDataGenerator
from matplotlib import pyplot

img = load_img("bird.jpg")
data = img_to_array(img)
samples = expand_dims(data, 0)

datagen = ImageDataGenerator(rotation_range = 90)

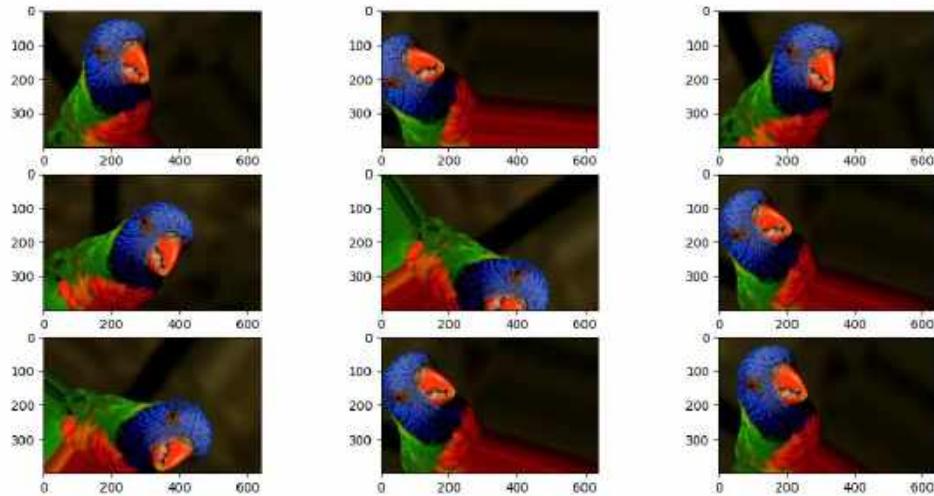
it = datagen.flow(samples, batch_size = 1)

for i in range(9):
    pyplot.subplot(330 + 1 + i)
    batch = it.next()
    image = batch[0].astype("uint8")
    pyplot.imshow(image)

pyplot.show()
```

## 5. Computer Vision – Image Data Augmentation in Keras

output



## 5. Computer Vision – Image Data Augmentation in Keras

---

### 4. Random Brightness Augmentation

- ✓ The brightness of the image can be augmented by either randomly darkening images, brightening images, or both.
- ✓ **ImageDataGenerator** is a predefined class
  - Here keyword arguments plays important role.
- ✓ **Keyword arguments.**
  - brightness\_range

## 5. Computer Vision – Image Data Augmentation in Keras

**Program Name** Image data augmentation, **brightness\_range\_range**  
demo5.py

```
from numpy import expand_dims
from tensorflow.keras.utils import load_img
from tensorflow.keras.utils import img_to_array
from keras.preprocessing.image import ImageDataGenerator
from matplotlib import pyplot

img = load_img("bird.jpg")
data = img_to_array(img)
samples = expand_dims(data, 0)

datagen = ImageDataGenerator(brightness_range = [0.2, 1.0])

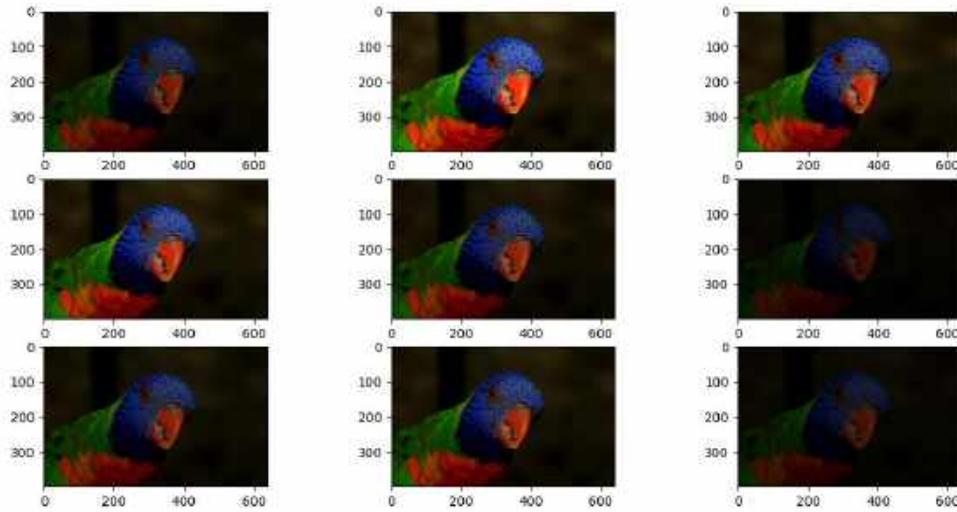
it = datagen.flow(samples, batch_size = 1)

for i in range(9):
    pyplot.subplot(330 + 1 + i)
    batch = it.next()
    image = batch[0].astype("uint8")
    pyplot.imshow(image)

pyplot.show()
```

## 5. Computer Vision – Image Data Augmentation in Keras

output



## 5. Computer Vision – Image Data Augmentation in Keras

---

### 5. Random Zoom Augmentation

- ✓ A zoom augmentation randomly zooms the image and either adds new pixel values around the image or interpolates pixel values respectively.
- ✓ **ImageDataGenerator** is a predefined class
  - Here keyword arguments plays important role.
- ✓ **Keyword arguments.**
  - `zoom_range`
    - For example, [0.7, 1.3] means 70% (zoom in) and 130% (zoom out).

## 5. Computer Vision – Image Data Augmentation in Keras

Program      Image data augmentation, **zoom\_range**  
Name        demo6.py

```
from numpy import expand_dims
from tensorflow.keras.utils import load_img
from tensorflow.keras.utils import img_to_array
from keras.preprocessing.image import ImageDataGenerator
from matplotlib import pyplot

img = load_img("bird.jpg")
data = img_to_array(img)
samples = expand_dims(data, 0)

datagen = ImageDataGenerator(zoom_range = [0.5,1.0])

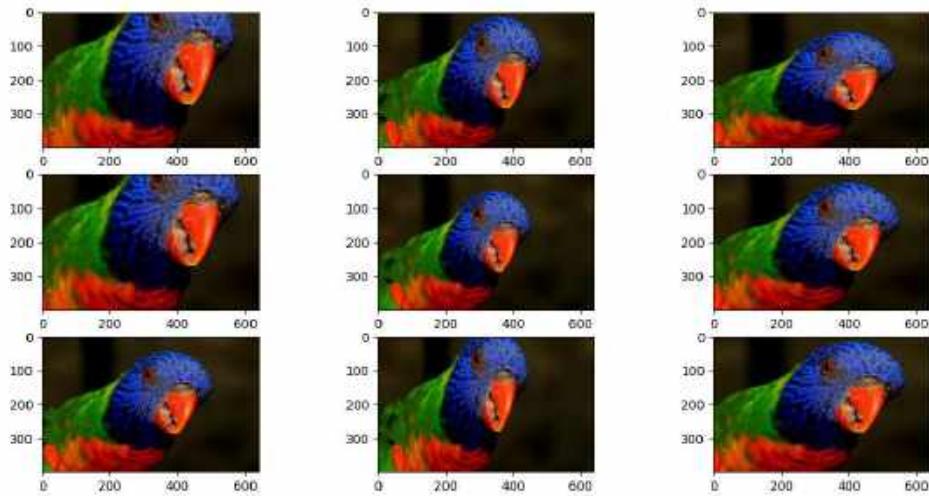
it = datagen.flow(samples, batch_size = 1)

for i in range(9):
    pyplot.subplot(330 + 1 + i)
    batch = it.next()
    image = batch[0].astype("uint8")
    pyplot.imshow(image)

pyplot.show()
```

## 5. Computer Vision – Image Data Augmentation in Keras

output



## 6. Computer Vision – Use Cases

---

### 6. Computer Vision – Use Cases

#### Contents

1. Edge & Contour Detection .....	2
2. Contours .....	4

## 6. Computer Vision – Use Cases

---

### 6. Computer Vision – Use Cases

#### 1. Edge & Contour Detection

- ✓ CV applications detect edges first and then collect other information.
- ✓ There are many edge detection algorithms, and the most popular is the Canny edge detector because it's pretty effective compared to others.
- ✓ It's also a complex edge-detection technique.
- ✓ Below are the steps for Canny edge detection:
  - Reduce noise and smoothen image
  - Calculate the gradient
  - Non-maximum suppression
  - Double the threshold
  - Linking and edge detecting

**Program Name**

Canny edge detection

**demo1.py**

```
import cv2
import matplotlib.pyplot as plt

img = cv2.imread('1.jpg')
edges = cv2.Canny(img, 100, 200, 3, L2gradient = True)

plt.figure()
plt.title('Spider')
plt.imsave('dancing-spider-canny.png', edges, cmap = 'gray',
format='png')
plt.imshow(edges, cmap='gray')
plt.show()
```

**Input**

## 6. Computer Vision – Use Cases



## 6. Computer Vision – Use Cases

### 2. Contours

- ✓ Contours are lines joining all the continuous objects or points (along the boundary), having the same color or intensity.
- ✓ For example, it detects the shape of a leaf based on its parameters or border.
- ✓ Contours are an important tool for shape and object detection.
- ✓ The contours of an object are the boundary lines that make up the shape of an object as it is.
- ✓ Contours are also called outline, edges, or structure, for a very good reason.

<b>Program Name</b>	Contours demo2.py
---------------------	----------------------

```
import cv2
import numpy as np

image = cv2.imread('2.jpg')
cv2.waitKey(0)

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

edged = cv2.Canny(gray, 30, 200)
cv2.waitKey(0)

contours, hierarchy = cv2.findContours(edged,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)

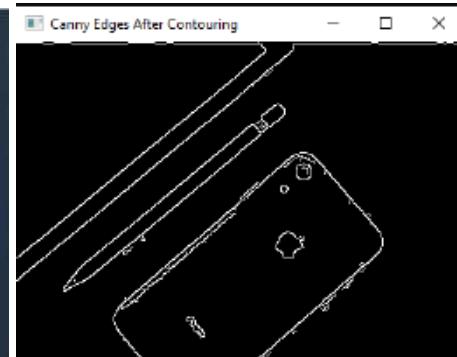
cv2.imshow('Canny Edges After Contouring', edged)
cv2.waitKey(0)

cv2.drawContours(image, contours, -1, (0, 255, 0), 3)

cv2.imshow('Contours', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

## 6. Computer Vision – Use Cases

Input



Material : Generative AI

Topic : Vector Database



Daniel

danielgenai77@gmail.com

## Gen AI – Vector Database

---

### Gen AI – Vector Database

<b>1. Data.....</b>	2
<b>2. Types of data .....</b>	2
2.1. Structured Data:.....	2
2.2. Semi-structured Data:.....	2
2.3. Unstructured Data: .....	2
<b>3. Why Vector Database?.....</b>	2
<b>4. Embedding.....</b>	3
4.1. Why Embeddings? .....	3
4.2. Example: Word Embeddings .....	3
4.3. Types of Data That Can Be Embedded.....	4
4.4. Real-World Uses.....	4
<b>5. Before embedding.....</b>	4
<b>6. Vector Database .....</b>	5
<b>7. Open-Source Models for Embeddings .....</b>	5
<b>8. Environment setup: sentence-transformers .....</b>	6
<b>9. Convert Text to Embeddings using transformers.....</b>	6
<b>10. Convert Text to Embeddings using transformers.....</b>	7
<b>11. Clustering of Similar Concepts in Word Embeddings.....</b>	9
<b>12. Semantic Search with Embeddings and Vector Databases.....</b>	9
<b>13. Why Encode Unstructured Data? .....</b>	10
<b>14. Examples.....</b>	10
14.1. Example #1: Vacation Photos .....	10
14.2. Example #2: News Search .....	13
<b>15. How to generate embeddings?.....</b>	14
<b>16. How Word Embeddings Were Created Before Transformers.....</b>	14
<b>17. Popular Pre-Transformer Embedding Models.....</b>	15
<b>18. BERT &amp; Sentence Embeddings .....</b>	15
<b>19. Vector Database Providers .....</b>	15

## Gen AI – Vector Database

---

### Gen AI – Vector Database

#### 1. Data

- ✓ Data is a collection of facts or information used for analysis and decision-making.

#### 2. Types of data

- ✓ There are mainly three types of data.
  - Structured data
  - Semi structured data
  - Unstructured data

##### 2.1. Structured Data:

- ✓ Organized in clear rows and columns, like spreadsheets or databases (e.g., names, dates).

##### 2.2. Semi-structured Data:

- ✓ Partially organized but doesn't fit rigid tables.
- ✓ It uses tags or markers to separate data elements, like JSON or XML files.

##### 2.3. Unstructured Data:

- ✓ No fixed format, including text, images, videos, and audio.

#### 3. Why Vector Database?

- ✓ Unstructured data includes information that doesn't fit into rows and columns, such as text, images, audio, and video.
- ✓ Because this data lacks a fixed format.
- ✓ Traditional databases struggle to store and search it efficiently.
- ✓ Vector databases solve this problem.

## Gen AI – Vector Database

---

### 4. Embedding

- ✓ Embeddings are numerical representations of data (like text, images, or audio) in a way that captures its meaning, context, or key features.

#### 4.1. Why Embeddings?

- ✓ Real-world data is messy — documents, images, videos, audio — and machines can't "understand" them directly.
- ✓ Embeddings translate this data into a numerical form that preserves its meaning, semantic relationships, or structure.

#### 4.2. Example: Word Embeddings

- ✓ Words like:
  - "king" → [0.25, 0.11, -0.78, ...]
  - "queen" → [0.21, 0.09, -0.80, ...]
  - "apple" → [-0.44, 0.87, 0.31, ...]
- ✓ You'll notice that:
  - king - man + woman ≈ queen → embeddings capture relationships!
  - "king" and "queen" are closer than "king" and "apple" in vector space.

## Gen AI – Vector Database

---

### 4.3. Types of Data That Can Be Embedded

Data Type	Embedding Example
✓ Text (word/sentence)	✓ Word2Vec, BERT, GPT embeddings
✓ Images	✓ CLIP, ResNet embeddings
✓ Audio	✓ Whisper, Wav2Vec embeddings
✓ Documents	✓ Sentence Transformers, Doc2Vec

### 4.4. Real-World Uses

- ✓ **Search engines:** Retrieve semantically similar documents.
- ✓ **Recommendation systems:** Recommend items based on similarity.
- ✓ **Chatbots/LLMs:** Understand user inputs more deeply.
- ✓ **Clustering/classification:** Group similar items together.

## 5. Before embedding

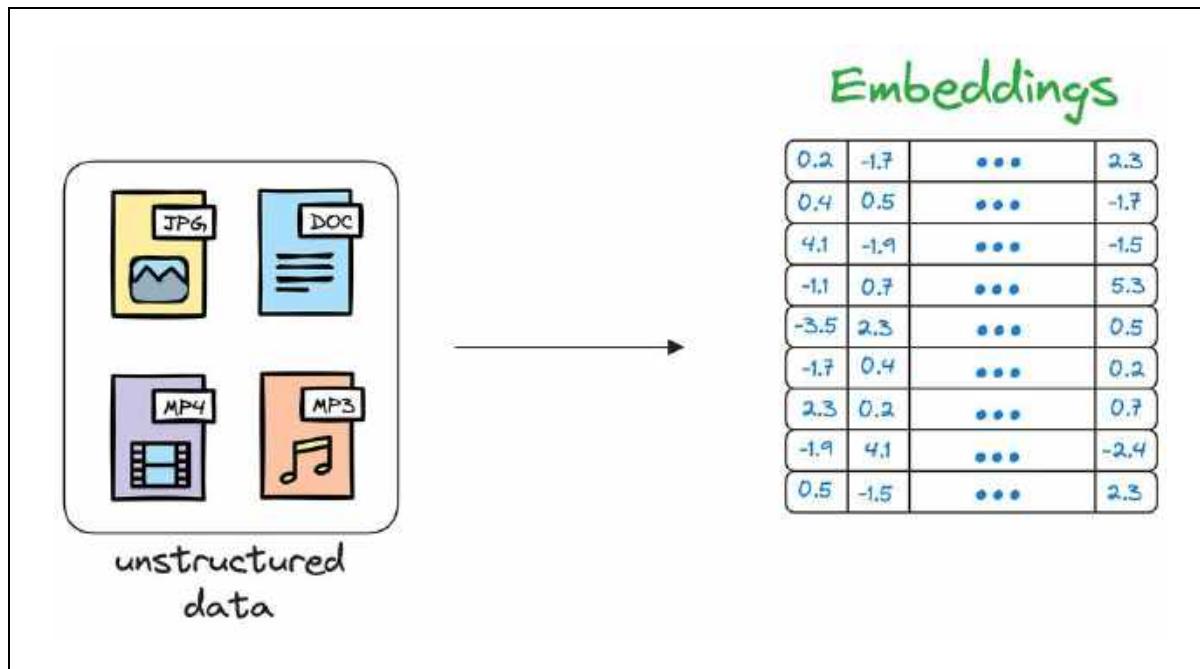
 Why Embeddings Replaced These

Feature	One-Hot / BoW / TF-IDF	Embeddings
Understand meaning?	✗	✓
Dense vector?	✗ Sparse	✓ Dense
Captures similarity?	✗	✓
Learns from context?	✗	✓ (especially with neural models)

## Gen AI – Vector Database

### 6. Vector Database

- ✓ Vector database stores unstructured data (text, images, audio, video, etc.) in the form of vector embeddings (numeric representations).
- ✓ Each data point (e.g., word, image, document) is converted into a numerical vector called an *embedding*.



### 7. Open-Source Models for Embeddings

- ✓ All these are available via Hugging Face.

Model Name	Description
all-MiniLM-L6-v2	suitable for semantic search
multi-qa-MiniLM-L6-cos-v1	Optimized for question answer tasks
paraphrase-MiniLM-L12-v2	Great for comparing similar sentences

## Gen AI – Vector Database

### 8. Environment setup: sentence-transformers

```
pip install sentence-transformers
```

### 9. Convert Text to Embeddings using transformers

<p><b>Program Name</b></p> <p>Convert Text to Embeddings using transformers demo1.py</p> <pre>from sentence_transformers import SentenceTransformer  # Load a pre-trained embedding model model = SentenceTransformer('all-MiniLM-L6-v2')  # Your input text text = "The Eiffel Tower is located in Paris and was built in 1889."  # Get the embedding embedding = model.encode(text)  # Print part of the embedding print("Embedding vector (truncated):", embedding[:10])</pre> <p><b>Output</b></p> <pre>Embedding vector (truncated): [ 3.9757401e-02  5.1625822e-02  1.9713903e-05  1.6055735e-03  1.1941780e-02 -1.3450466e-02 -8.6841680e-02  3.2891795e-02  5.2027605e-03 -3.1023417e-02]</pre>
---

## Gen AI – Vector Database

---

### 10. Convert Text to Embeddings using transformers

<b>Program Name</b>	Multiple Sentence Embeddings Example demo1.py
---------------------	--

```

from sentence_transformers import SentenceTransformer
import numpy as np

# Load a pre-trained embedding model
model = SentenceTransformer('all-MiniLM-L6-v2')

# List of sentences to embed
sentences = [
    "The Eiffel Tower is located in Paris and was built in 1889.",
    "Paris is home to the famous Eiffel Tower.",
    "The Statue of Liberty is in New York City.",
    "Bananas are yellow and rich in potassium.",
    "Artificial intelligence is transforming the world."
]

# Generate embeddings for each sentence
embeddings = model.encode(sentences)

# Print the first 10 values of each embedding
for i, (sentence, embedding) in enumerate(zip(sentences,
embeddings)):
    print("Sentence {i+1}: {sentence}")
    print("Embedding vector (truncated):", embedding[:10])

```

#### Output

Sentence 1: The Eiffel Tower is located in Paris and was built in 1889.  
Embedding vector (truncated): [ 3.97573858e-02 5.16258739e-02 1.97258523e-05 1.60556904e-03 1.19417915e-02 -1.34504791e-02 -8.68416578e-02 3.28918062e-02 5.20278560e-03 -3.10233738e-02]

## Gen AI – Vector Database

Sentence 2: Paris is home to the famous Eiffel Tower.

Embedding vector (truncated): [ 0.08414423 0.04866021  
0.01386443 0.00248201 0.04932949 0.00215907  
-0.0550714 0.01174218 -0.0026298 -0.02511607]

Sentence 3: The Statue of Liberty is in New York City.

Embedding vector (truncated): [ 0.08450228 0.09561136  
0.03509006 0.01049995 0.04243909 0.01134018  
-0.00709057 -0.0134034 0.00454255 -0.01797276]

Sentence 4: Bananas are yellow and rich in potassium.

Embedding vector (truncated): [-0.00945398 -0.03126404  
0.02787722 0.03488883 0.00328189 0.07226413  
0.0417771 -0.01045569 0.02283465 0.07436628]

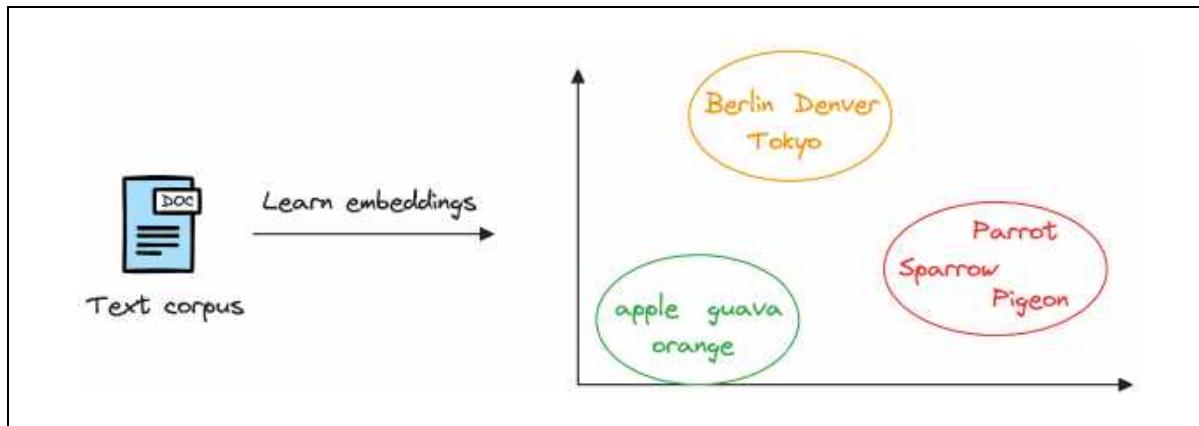
Sentence 5: Artificial intelligence is transforming the world.

Embedding vector (truncated): [ 0.03872412 -0.00110552  
0.08271616 -0.01628856 0.04654312 -0.0095303  
-0.02997492 0.00349416 0.01119627 0.00263023]

## Gen AI – Vector Database

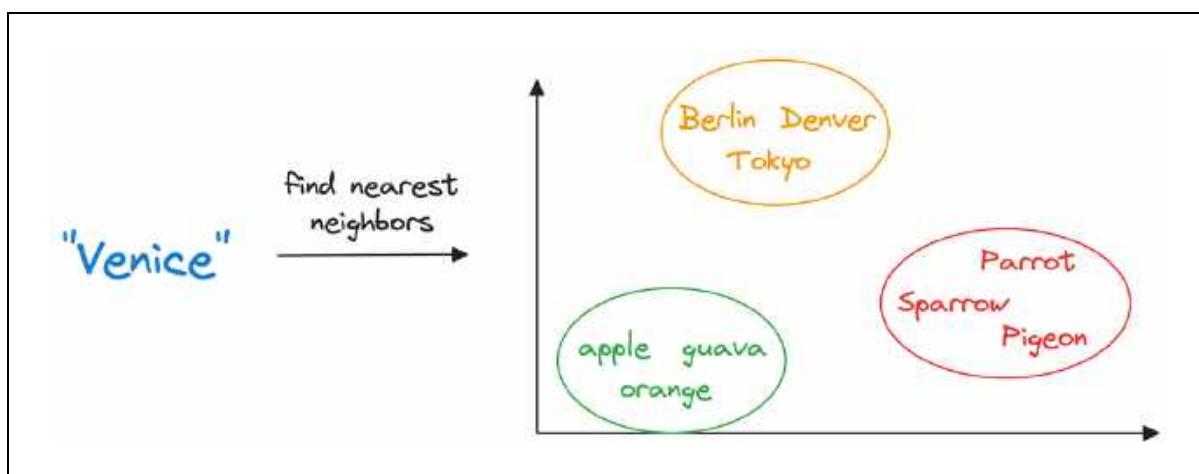
### 11. Clustering of Similar Concepts in Word Embeddings

- ✓ In word embeddings, similar concepts—like fruits or cities—tend to cluster together in the embedding space.



### 12. Semantic Search with Embeddings and Vector Databases

- ✓ This shows that, when properly trained, embeddings can capture the semantic meaning of the entities they represent.
- ✓ Once stored in a vector database, embeddings allow us to retrieve original objects similar to a given query on unstructured data.



## Gen AI – Vector Database

### 13. Why Encode Unstructured Data?

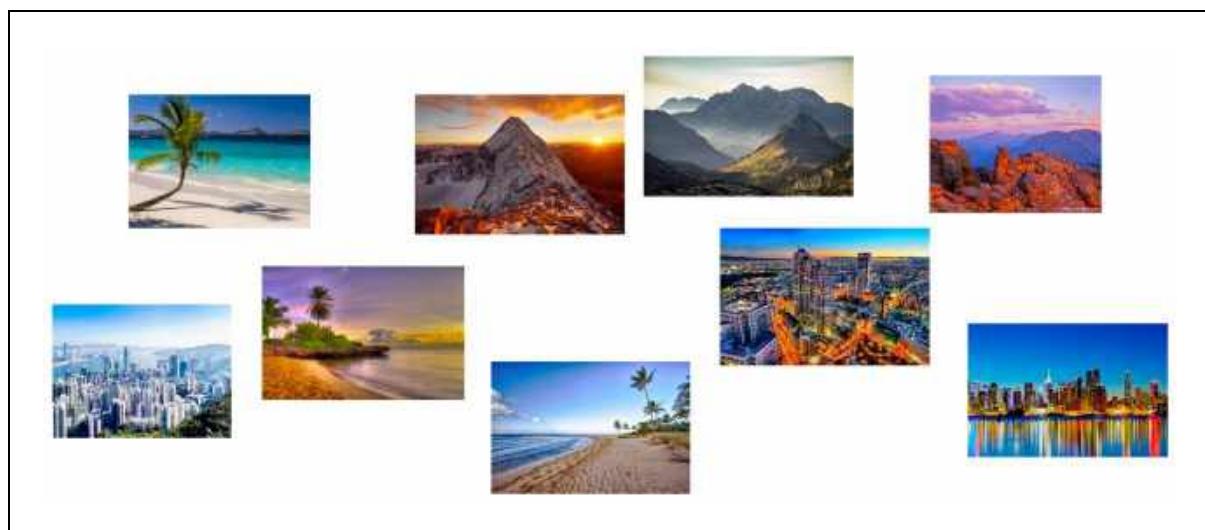
- ✓ Encoding unstructured data makes it easier to perform tasks like search, clustering, and classification.
- ✓ For example, when an e-commerce site recommends similar products, it's often powered by vector databases.

### 14. Examples

- ✓ **Image Search:** Find similar images by comparing vector representations.
- ✓ **Product Recommendations:** Suggest related items using product vectors.
- ✓ **Semantic Text Search:** Match user queries with relevant results based on meaning, not just keywords.

#### 14.1. Example #1: Vacation Photos

- ✓ Imagine a collection of vacation photos—beaches, mountains, cities, forests.
- ✓ By converting these images into vectors, we can easily find and group similar scenes, like all beach photos, without manual tagging.



## Gen AI – Vector Database

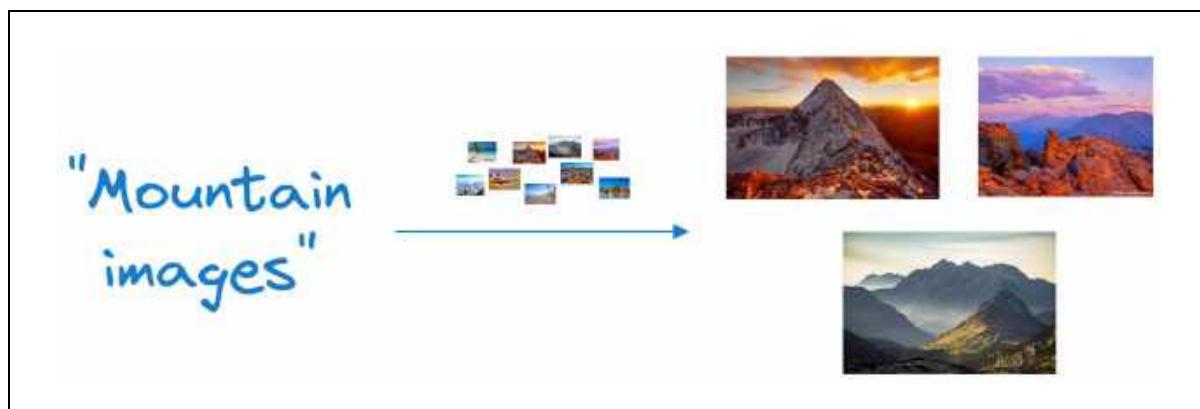
### The Need for Smarter Organization

- ✓ Now, we want to organize these photos to easily find similar ones.
- ✓ We can sort photos by date or location.
- ✓ But that doesn't help if we want all beach photos or city



### Smarter Search with Vectors

- ✓ We convert each photo into a vector based on features like color, shape, and texture.
- ✓ These vectors sit in a multi-dimensional space.
- ✓ To find similar photos—like mountains—we encode a text query into a vector and retrieve the closest image vectors.
- ✓ This makes searching by content fast and intuitive.



## Gen AI – Vector Database

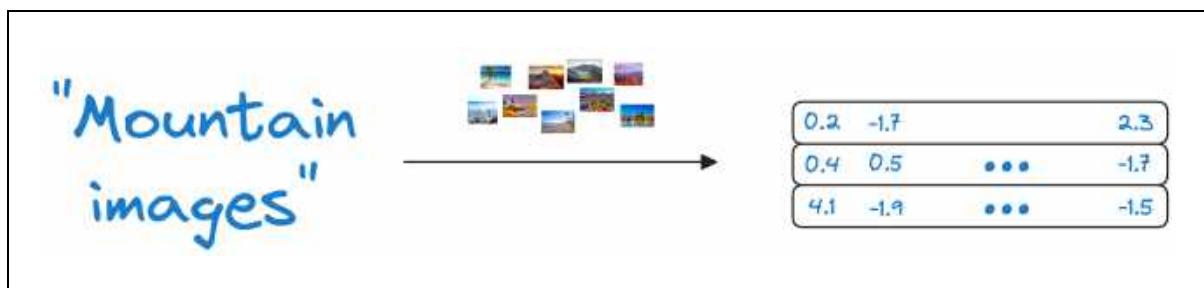
---

### Important Note

- ✓ A vector database is not just for storing embeddings—it also keeps the original data (like images or text) linked to those embeddings, enabling efficient retrieval and meaningful results.

### Why Store Raw Data Too?

- ✓ If the database only stores vectors, we can't show actual results.
- ✓ For image search, users need the images—not just vectors—so the database must store both to return meaningful results.



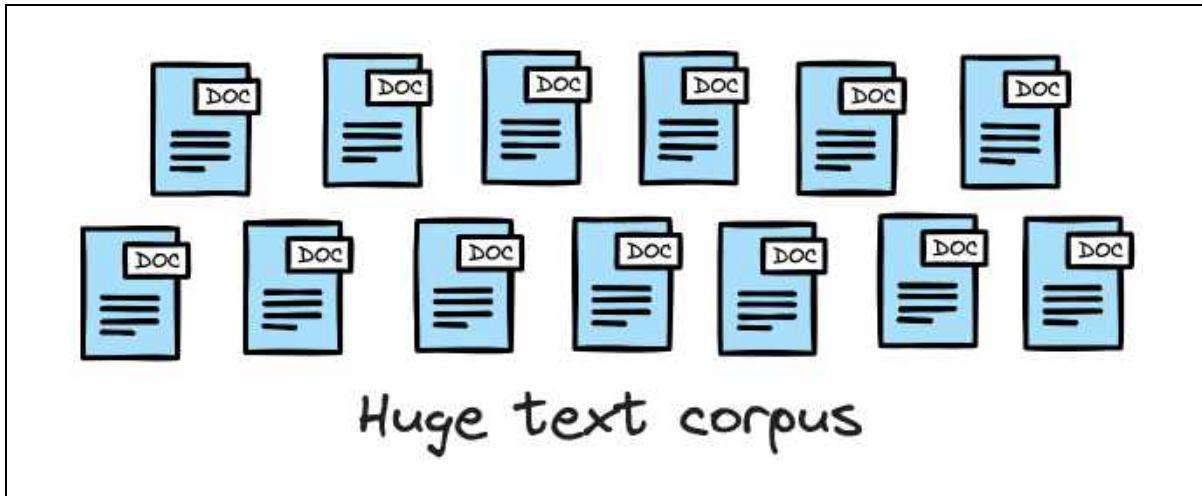
### Complete Retrieval

- ✓ By storing both embeddings and raw data, a vector database ensures that search results include not just similar vectors but also the actual images users want to see.

## Gen AI – Vector Database

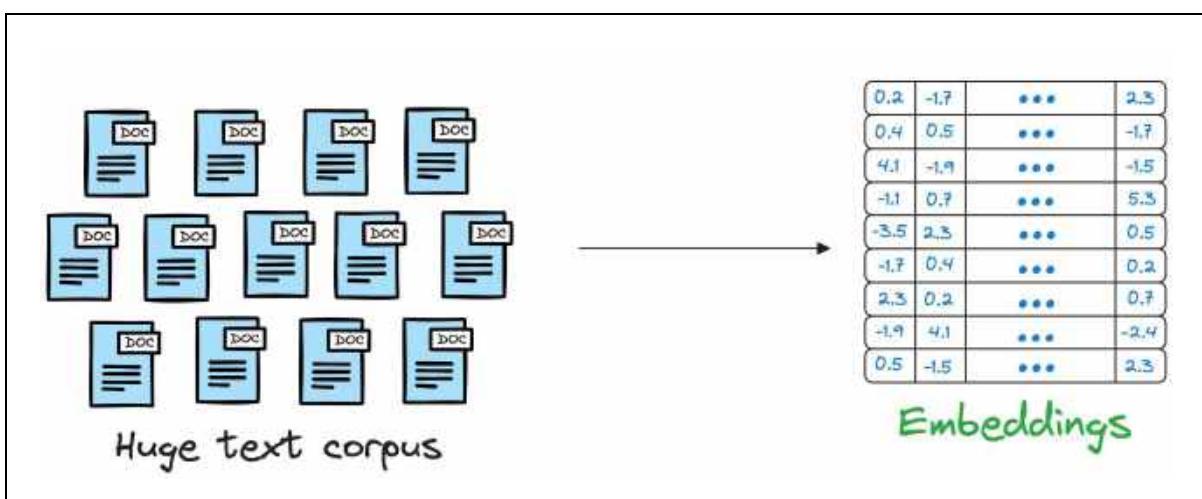
### 14.2. Example #2: News Search

- ✓ With thousands of news articles, vector search helps find relevant answers by matching meaning, not just keywords—making retrieval more accurate and insightful.



### Limitations of Keyword Search

- ✓ Traditional keyword search misses the nuance of language—different phrases can mean the same thing.
- ✓ By encoding text as vectors, we capture meaning, enabling smarter and more flexible search in a vector database.



## Gen AI – Vector Database

---

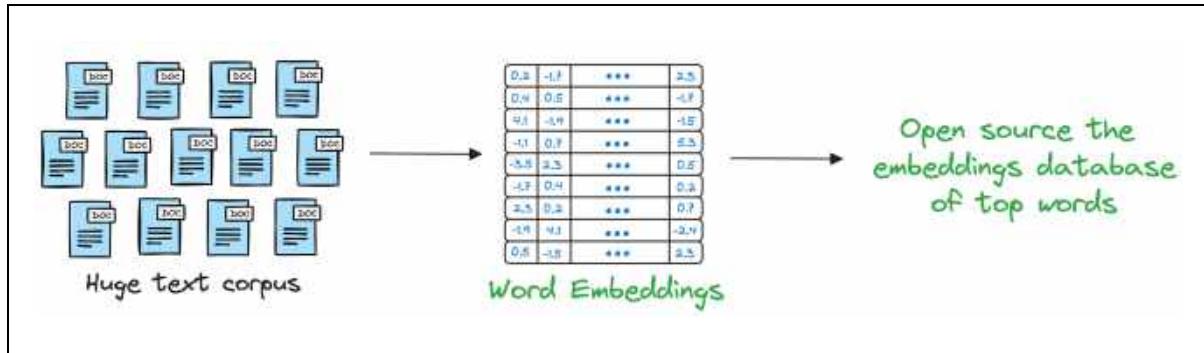
### Smarter Matching

- ✓ A vector database compares the query's vector with text vectors, allowing it to find relevant results—even when the wording is different.

### 15. How to generate embeddings?

- ✓ Embeddings change words or sentences into numbers.
- ✓ So, computers can understand their meaning.
- ✓ This helps with things like search, matching, or sorting text.

### 16. How Word Embeddings Were Created Before Transformers



- ✓ This image shows how word embeddings were built before Transformers:
  - **Huge text corpus:** A large collection of documents is used as training data.
  - **Word embeddings:** Words from the text are converted into numerical vectors using deep learning.
  - **Open-source database:** The resulting embeddings for common words are shared publicly for reuse.
- ✓ In short:
  - Train on lots of text → get word vectors → share for others to use.

## Gen AI – Vector Database

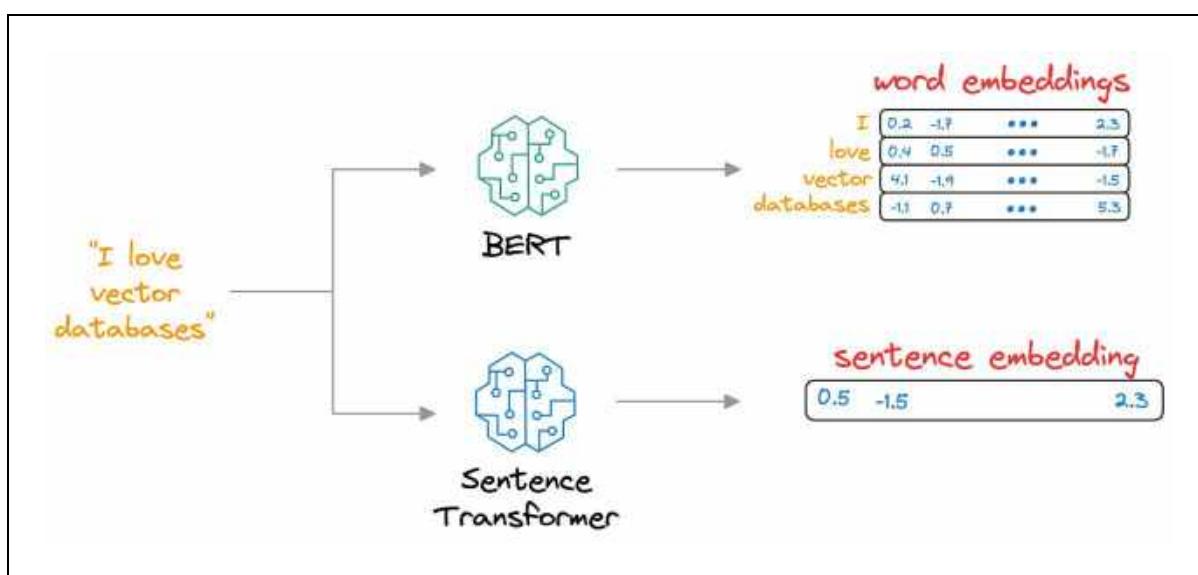
---

### 17. Popular Pre-Transformer Embedding Models

- ✓ Others used these shared embeddings in their projects.
- ✓ Models like **Word2Vec**, **GloVe**, and **FastText** (2013–2017) were widely used and showed strong results in capturing word relationships.

### 18. BERT & Sentence Embeddings

- ✓ **BERT**: A language model trained using two techniques:
  - **Masked Language Modeling (MLM)**: Predict a missing word in the sentence, given the surrounding words.
  - **Next Sentence Prediction (NSP)**.
- ✓ **SentenceTransformer**: Essentially, the SentenceTransformer model takes an entire sentence and generates an embedding for that sentence.



### 19. Vector Database Providers

- ✓ **Pinecone** : Managed, fast, and scalable vector search.
- ✓ **Weaviate** : Open-source, ML-powered, supports multiple data types.
- ✓ **Milvus** : Open-source, built for large-scale similarity search.
- ✓ **Qdrant** : Filter-rich, production-ready semantic search engine.

Material : Generative AI

Topic : **RAG (Retrieval Augmented Generation)**



Daniel  
[danielgenai77@gmail.com](mailto:danielgenai77@gmail.com)

## Gen AI – RAG (Retrieval Augmented Generation)

---

### Gen AI – RAG (Retrieval-Augmented Generation)

<b>1. RAG .....</b>	2
<b>2. Why RAG?.....</b>	2
<b>3. RAG Architecture: Step by step .....</b>	3
3.1. Encode Docs.....	3
3.2. Index.....	3
3.3. Encode Query.....	3
3.4. Similarity Search:.....	3
3.5. Retrieve:.....	3
3.6. Prompt LLM:.....	3
3.7. Generate Response:.....	4
<b>4. RAG: Retrieval Augmented Generation .....</b>	5
4.1. Retrieval .....	5
4.2. Augmented.....	5
4.3. Generation .....	5
<b>5. Workflow of a RAG System.....</b>	6
5.1. Addition knowledge base.....	7
5.2. Create Chunks .....	7
5.3. Generate embeddings.....	8
5.4. Store embeddings in a vector database .....	8
5.5. User input query .....	9
5.6. Embed the query.....	9
5.7. Retrieve similar chunks .....	10
5.8. Re-rank Chunks .....	10
5.9. Generate Final Response .....	11
<b>6. Tool Stack for Building a RAG System.....</b>	12
6.1. Llamaindex .....	12
6.2. Qdrant .....	12
6.3. Ollama .....	12
<b>7. RAG application using Llama 3.2 .....</b>	13

## Gen AI – RAG (Retrieval Augmented Generation)

---

### Gen AI – RAG (Retrieval-Augmented Generation)

#### 1. RAG

- ✓ **RAG** stands for Retrieval-Augmented Generation.
- ✓ RAG is an NLP approach that:
  - **Retrieves** relevant info from external sources.
  - **Augments** the model input with that info.
  - **Generates** more accurate and factual responses using a language model.
- ✓ It helps LLMs handle large or constantly changing knowledge bases more effectively.
- ✓ RAG in GenAI = **Search + Generate**.
  - It gives large language models "open-book access" to relevant information, making them smarter, safer, and more adaptable.

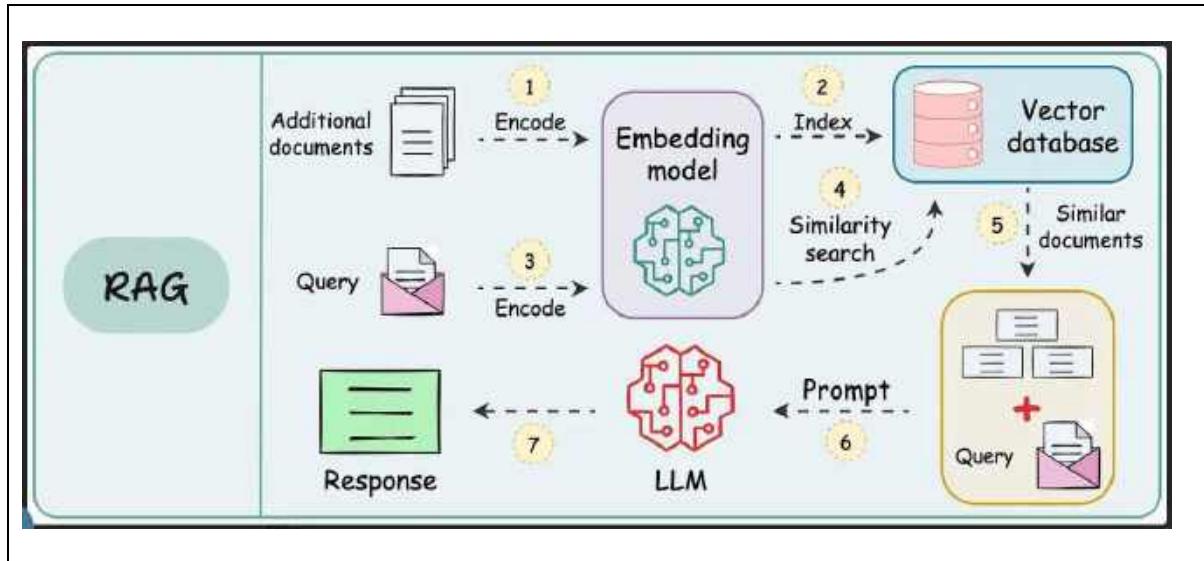
#### 2. Why RAG?

- ✓ LLMs often lack up-to-date or specific knowledge.
- ✓ RAG solves this by retrieving relevant information from external sources and adding it to the prompt.
- ✓ This allows the LLM to generate more accurate, grounded, and context-aware responses.
- ✓ Here RAG helps.

## Gen AI – RAG (Retrieval Augmented Generation)

---

### 3. RAG Architecture: Step by step



#### 3.1. Encode Docs

- ✓ Convert documents into embeddings using an embedding model.

#### 3.2. Index

- ✓ Store those embeddings in a vector database.

#### 3.3. Encode Query

- ✓ Convert the user query into an embedding.

#### 3.4. Similarity Search:

- ✓ Search the vector DB for documents similar to the query.

#### 3.5. Retrieve:

- ✓ Get the top matching documents.

#### 3.6. Prompt LLM:

- ✓ Combine the retrieved documents with the query and send to the language model.

## Gen AI – RAG (Retrieval Augmented Generation)

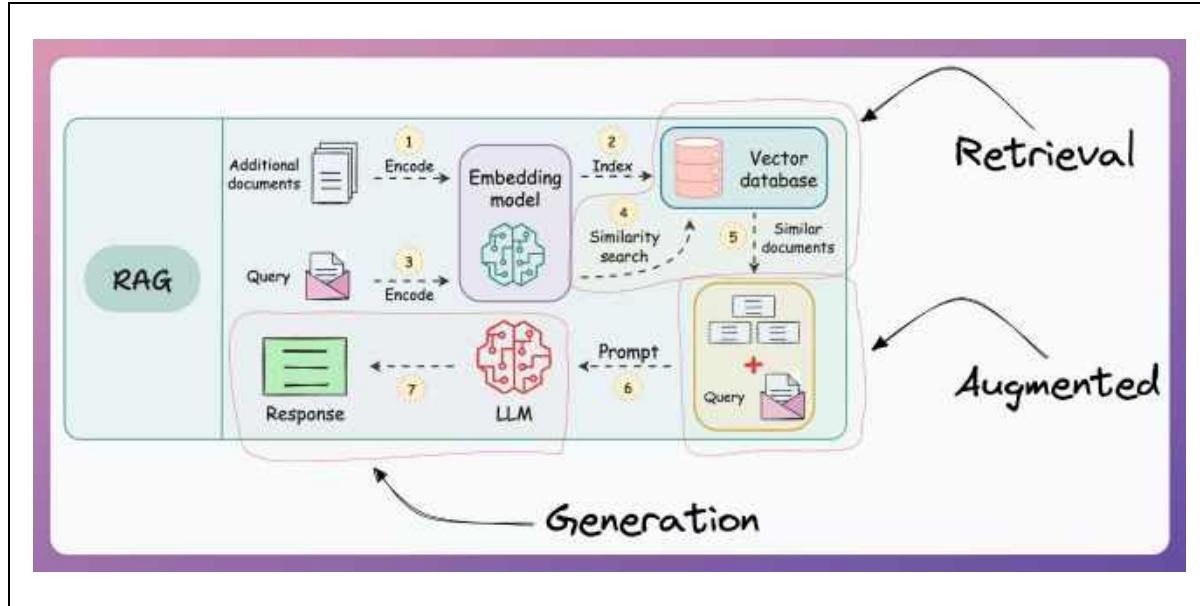
---

### 3.7. Generate Response:

- ✓ The LLM generates a final answer based on the augmented input.

## Gen AI – RAG (Retrieval Augmented Generation)

### 4. RAG: Retrieval Augmented Generation



#### 4.1. Retrieval

- ✓ Fetching relevant info from a source (e.g., database).

#### 4.2. Augmented

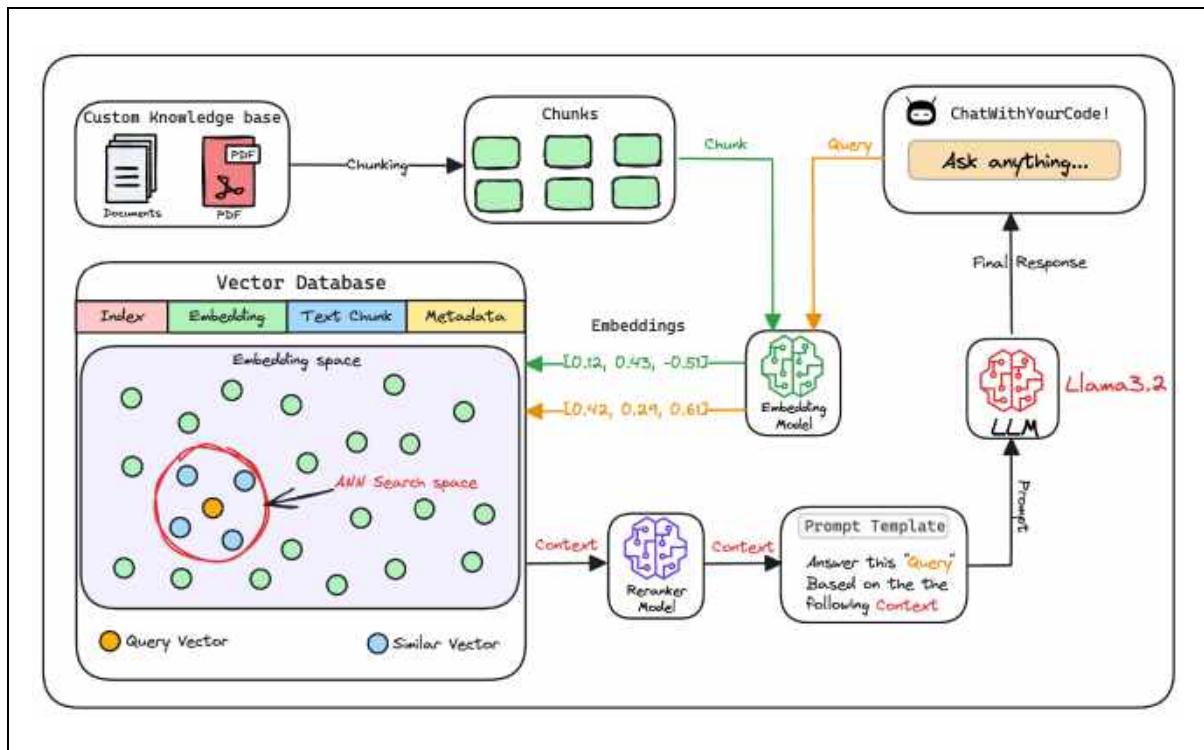
- ✓ Adding extra context to improve the process (e.g., text generation).

#### 4.3. Generation

- ✓ Creating or producing something, like generating text.

## Gen AI – RAG (Retrieval Augmented Generation)

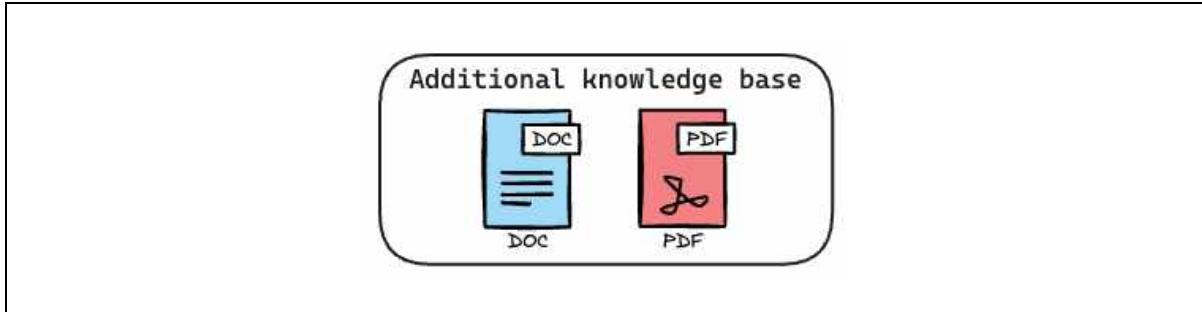
### 5. Workflow of a RAG System



## Gen AI – RAG (Retrieval Augmented Generation)

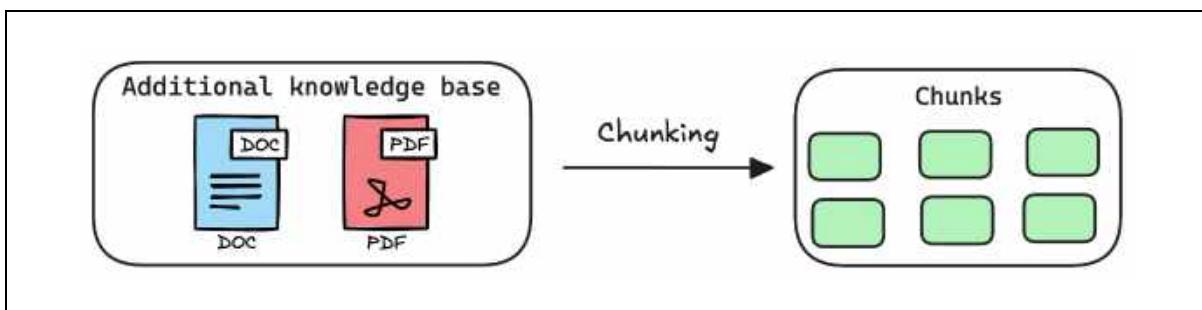
### 5.1. Addition knowledge base

- ✓ We start with some external knowledge that wasn't seen during training, and we want to enhance the LLM.



### 5.2. Create Chunks

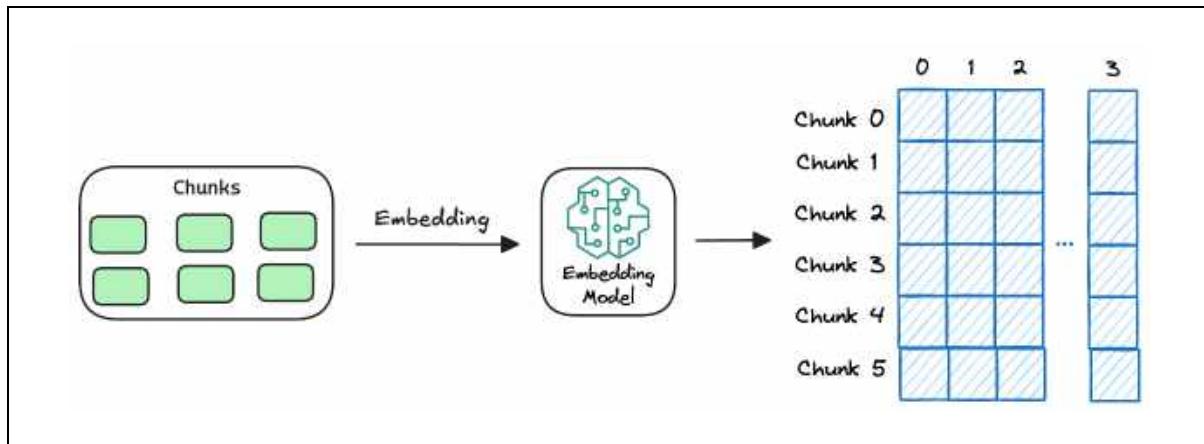
- ✓ Before storing knowledge, it's broken into smaller, meaningful chunks.
- ✓ This improves retrieval accuracy and ensures each piece of information is easily searchable during query time.



## Gen AI – RAG (Retrieval Augmented Generation)

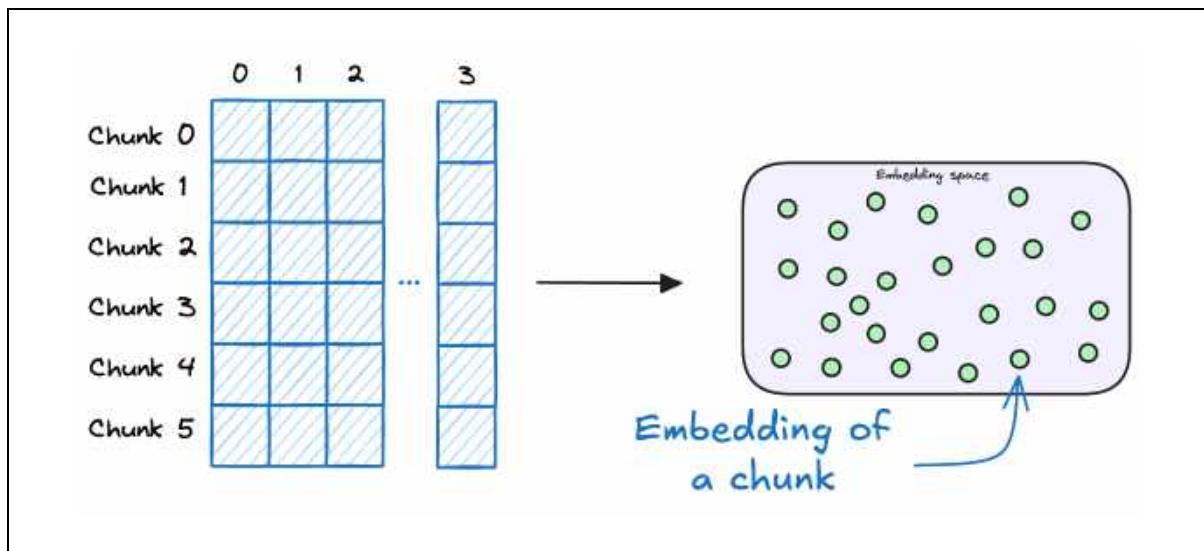
### 5.3. Generate embeddings

- ✓ After chunking, we embed the chunks using an embedding model.



### 5.4. Store embeddings in a vector database

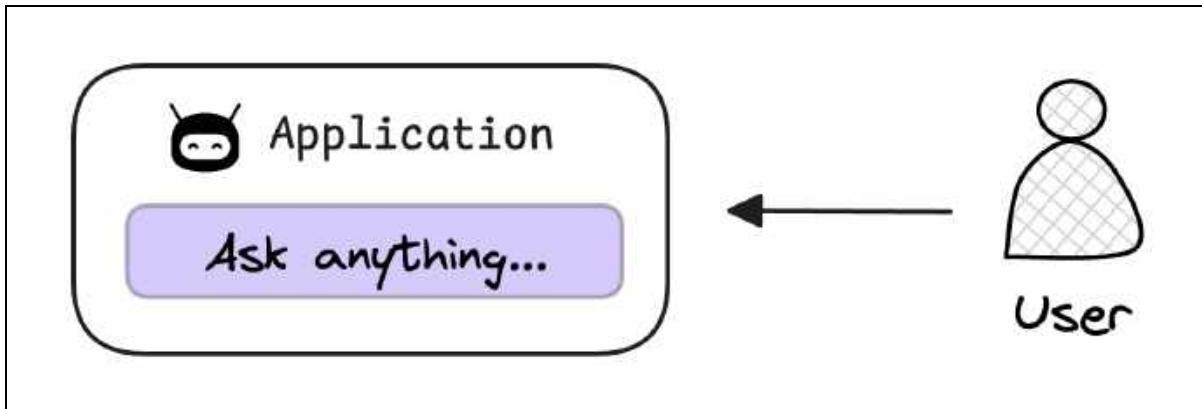
- ✓ These embeddings are then stored in the vector database



## Gen AI – RAG (Retrieval Augmented Generation)

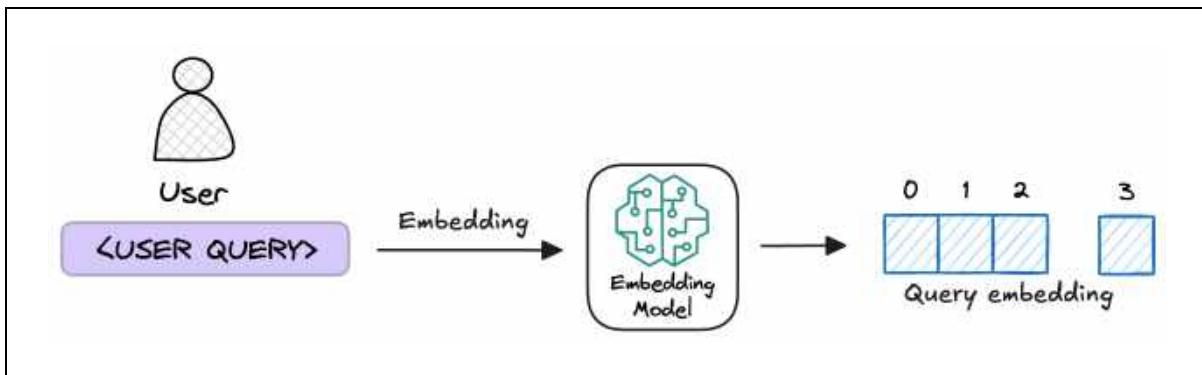
### 5.5. User input query

- ✓ Next, the user inputs a query, a string representing the information they're seeking.



### 5.6. Embed the query

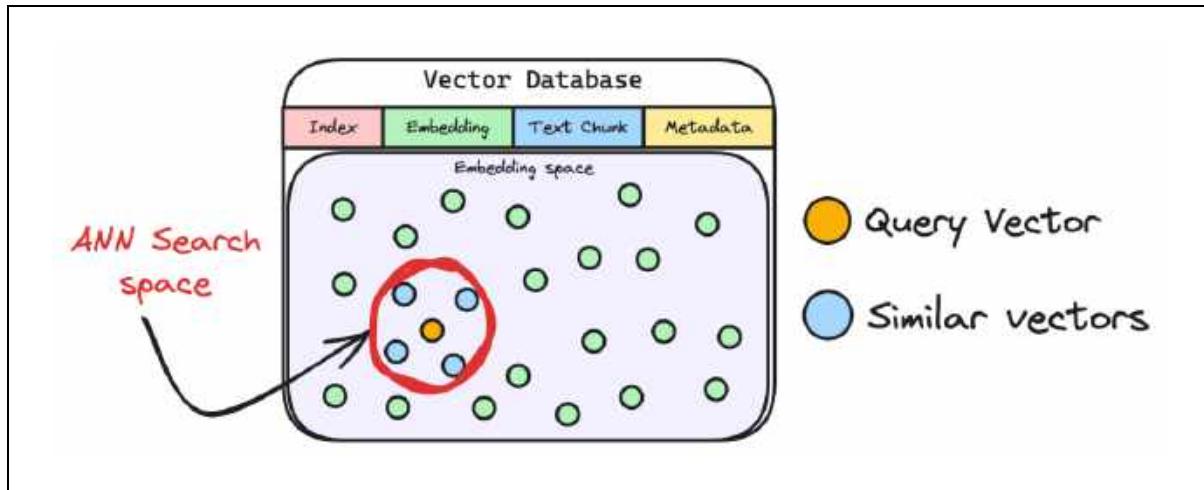
- ✓ This query is transformed into a vector using the same embedding model we used to embed the chunks earlier in Step 2.



## Gen AI – RAG (Retrieval Augmented Generation)

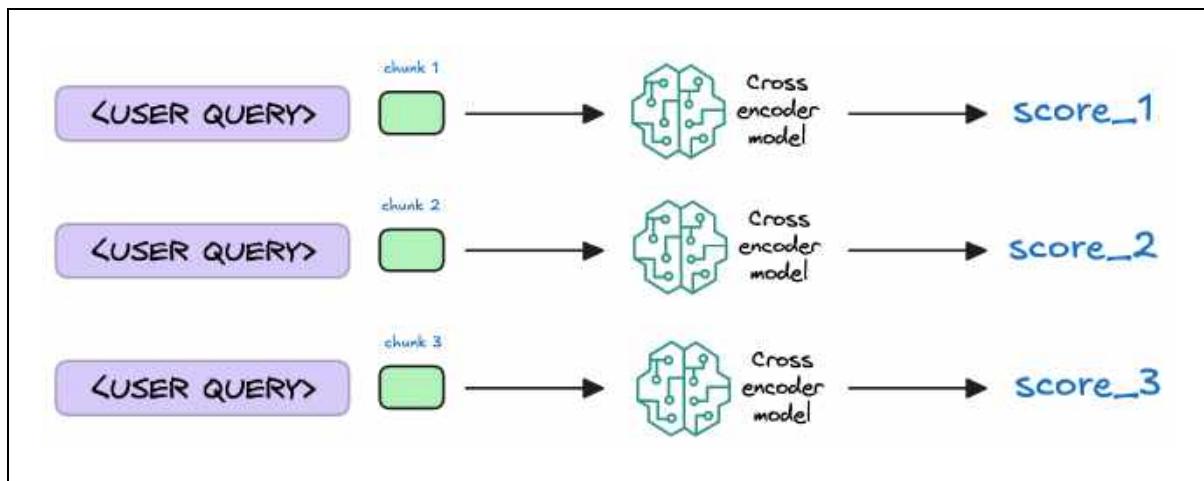
### 5.7. Retrieve similar chunks

- ✓ The vectorized query is then compared against our existing vectors in the database to find the most similar information.



### 5.8. Re-rank Chunks

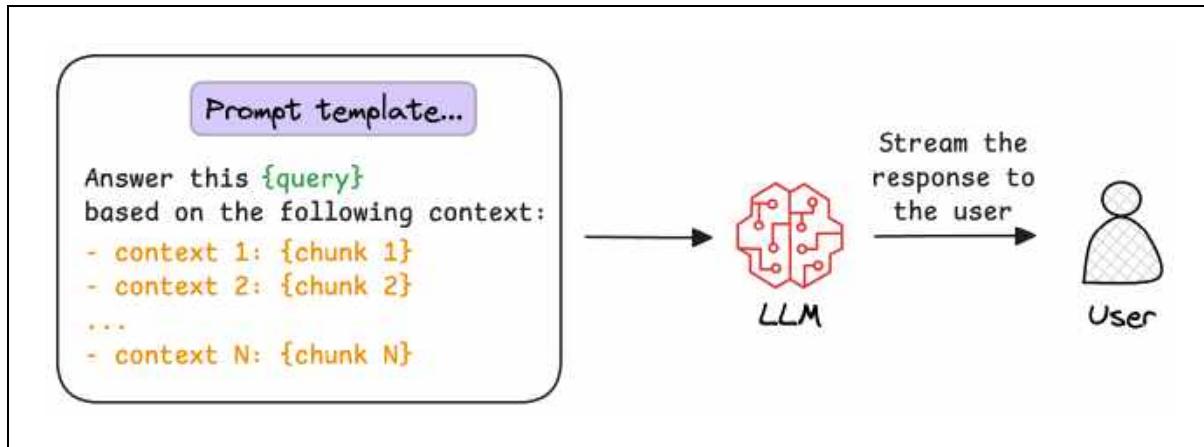
- ✓ After retrieval, a cross-encoder re-scores the chunks to prioritize the most relevant ones based on the query.



## Gen AI – RAG (Retrieval Augmented Generation)

### 5.9. Generate Final Response

- ✓ The top-ranked chunks and the user query are combined into a prompt and sent to the LLM, which generates a final, informed response.



## Gen AI – RAG (Retrieval Augmented Generation)

---

### 6. Tool Stack for Building a RAG System

#### 6.1. LlamaIndex

- ✓ Simplifies connecting LLMs with external data sources for indexing and querying.

#### 6.2. Qdrant

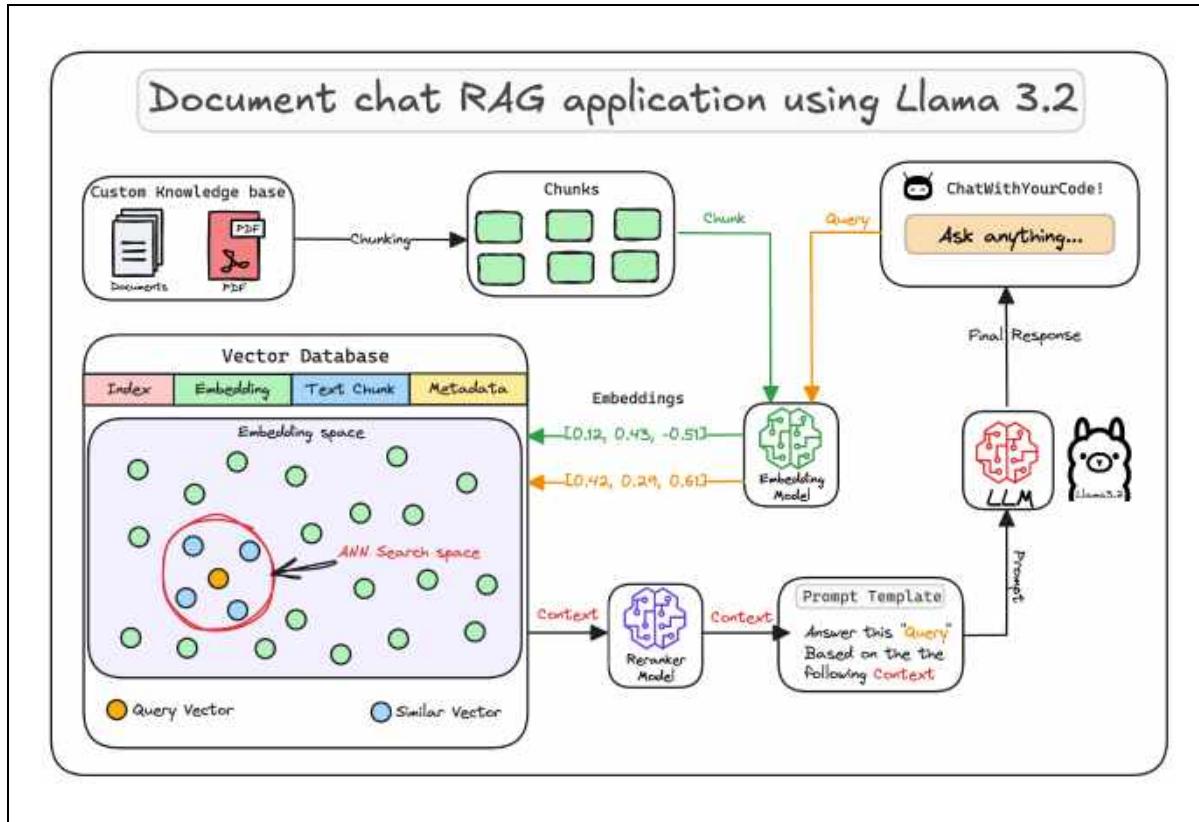
- ✓ Open-source vector database for fast, filtered similarity search at scale.

#### 6.3. Ollama

- ✓ Runs LLMs locally, ideal for privacy-focused applications

## Gen AI – RAG (Retrieval Augmented Generation)

### 7. RAG application using Llama 3.2



- ✓ **Custom Knowledge Base:** Input documents (e.g., PDFs, text files)
- ✓ **Chunking:** Break documents into smaller pieces (chunks)
- ✓ **Embedding:** Convert chunks into vector embeddings using an Embedding Model
- ✓ **Vector Database:** Store embeddings and related metadata. Perform ANN (Approximate Nearest Neighbor) search to find similar vectors to the user's query
- ✓ **Query Processing:** User submits a question. Convert query to a vector. Retrieve top matching chunks
- ✓ **Re-ranking (Optional):** Use a Re-ranker Model to refine and prioritize the most relevant chunks
- ✓ **Prompt Construction:** Fill a prompt template with the query and retrieved chunks
- ✓ **LLM Generation (Llama 3.2):** Final response is generated and shown to the user