PUNE INSTITUTE OF COMPUTER TECHNOLOGY

Subject: Machine Learning (LP-1 LAB)

Name: Aditya Kangune **Roll No. : 33323**

Batch: K11 **Academic Year: 2021-22**

Assignment 1
Data Preparation

Problem Statement:

Perform the following operations on the given dataset:

1. Find Shape of Data
2. Find Missing Values
3. Find the data type of each column
4. Finding out Zero's
5. Find Mean age of patients
6. Now extract only Age, Sex, chest pain, RestBP, Chol. Randomly divide dataset in training (75%) and testing (25%).
7. Through the diagnosis test, I predicted 100 reports as COVID positive, but only 45 of those were actually positive. A total of 50 people in my sample were actually COVID positive. I have a total of 500 samples.
8. Create a confusion matrix based on the above data and find
    a. Accuracy
    b. Precision
    c. Recall
    d. F-1 score

Objective:

This assignment will help the students to realize what is the need of data preparation.

Theory:

Data Preparation:

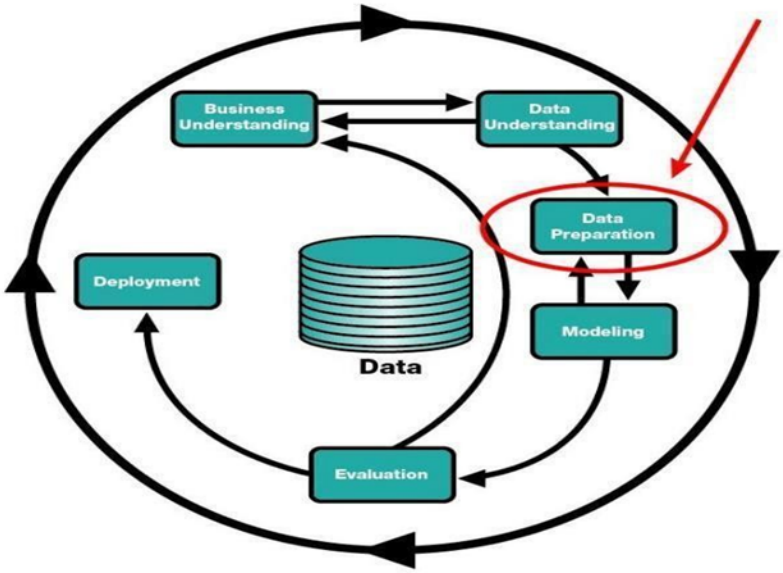
Data preparation (also referred to as “data preprocessing”) is the process of transforming raw data so that data scientists and analysts can run it through machine learning algorithms to uncover insights or make predictions.

Why is Data Preparation Important?

1. Most machine learning algorithms require data to be formatted in a very specific way, so datasets generally require some amount of preparation before they can yield useful insights.
2. Some datasets have values that are missing, invalid, or otherwise difficult for an algorithm to process.
3. If data is missing, the algorithm can’t use it.
4. If data is invalid, the algorithm produces less accurate or even misleading outcomes.
5. Some datasets are relatively clean but need to be shaped (e.g., aggregated or pivoted) and many datasets are just lacking useful business context (e.g., poorly defined ID values), hence the need for feature enrichment.
6. Good data preparation produces clean and well-curated data which leads to more practical, accurate model outcomes.

1. It is the most required process before feeding the data into the machine learning model.
2. The reason behind that the data set needs to be different and specific according to the model so that we have to find out the required features of that data.
3. The data preparation process offers a method via which we can prepare the data for defining the project and also for the project evaluation of ML algorithms.
4. Different many predicting machine learning models are there with a different process but some of the processes are common that are

performed in every model, and also it allows us to find out the actual business problem and their solutions.



Data Preparation [3]

Some of the data preparation processes are:

1) Determine the problems
2) Data cleaning
3) Feature selection
4) Data transformation
5) Feature engineering
6) Dimensionality reduction

1. Determine the problems:

1. This step tells us about the learning method of the project to find out the results for future prediction or forecasting.
2. For example, which ML model is suitable for the data set regression or classification or clustering algorithms.
3. This includes data collection that is useful for predicting the result and also involving communication to project stakeholders and domain expertise. We use classification and regression models for categorical and numerical data respectively.
4. It includes determining the relevant attributes with the stied data in form of .csv, .html, .json, .doc, and many, also for unstructured data in a form for audio, video, text, images, etc for scanning and detect the patterns of data with searching and identifying the data that have taken from external repositories.

2. Data cleaning:

1. After collecting the data, it is very necessary to clean that data and make it proper for the ML model.
2. It includes solving problems like outliers, inconsistency, missing values, incorrect, skewed, and trends.
3. Cleaning the data is very important as the model learns from that data only, so if we feed inconsistent, appropriate data to the model it will return garbage only, so it is required to make sure that the data does not contain any unseen problem.
4. For example, if we have a data set of sales, it might be possible that it contains some features like height, age, that cannot help in the model building so we can remove it.
5. We generally remove the null values columns, fill the missing values, make the data set consistent, and remove the outliers and skewed data in data cleaning.

3. Feature selection:

1. Sometimes we face the problem of identifying the related features from the set of data and deleting the irrelevant and less important data without touching the target variables to get the better accuracy of the model.
2. Features selection plays a wide role in building a machine learning model that impacts the performance and accuracy of the model.
3. It is that process that contributes mostly to the predictions or output that we need by selecting the features automatically or manually.
4. If we have irrelevant data that would cause the model with overfitting and underfitting.

The benefits of feature selection:

1. Reduce the overfitting/underfitting
2. Improves the accuracy
3. Reduced training/testing time
4. Improves performance

4. Data transformation:

1. Data transformation is the process that converts the data from one form to another.
2. It is required for data integration and data management. In data transformation, we can change the types of data, clear the data removing the null values or duplicate values, and get enriched data that depends on the requirements of the model.
3. It allows us to perform data mapping that determines how individual features are mapped, modified, filtered, aggregated, and joined.
4. Data transformation is needed for both structured and unstructured data but it is time-consuming, costly, slow.

5. Feature engineering:

1. Every ML algorithms use some input data for giving the required output and this input required some features which are in a structured form.
2. To get the proper result the algorithms required features with some specific characteristics which we find out with feature engineering.
3. We need to perform different feature engineering on different datasets and we can observe their effect on model performance.

   Here I am listing out the techniques of feature engineering.

   1. Imputation
   2. Handling outliers
   3. Binning
   4. Log transform
   5. one-hot encoding
   6. Grouping operations
   7. Feature split
   8. Scaling

6. Dimensionality reduction:

1. When we use the dataset for building an ML model, we need to work with 1000s of features that cause the curse of dimensionality, or we can say that it refers to the process to convert a set of data.
2. For the ML model, we have to access a large amount of data and that large amount of data can lead us to a situation where we can take possible data that can be available to feed it into a forecasting model to predict and give the result of the target variable.
3. It reduced the time that is required for training and testing our machine learning model and also helps to eliminate over-fitting.
4. It is kind of zipping the data for the model.

Conclusion:

Data preparation is recognized for helping businesses and analytics to get ready and prepare the data for operations.

Implementation:

33323 - Aditya Kangune Assignment 1 LP Lab

Double-click (or enter) to edit

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

```
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.model_selection import train_test_split
# from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_re
import random
```

Reading the CSV File

```
df = pd.read_csv("/content/drive/MyDrive/Datasets/heart_disease

df
```

	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak	Slope
1	63	1	typical	145	233	1	2	150	0	2.3	3

Shape of dataset

4	37	1	nonanginal	130	250	0	0	187	0	3.5	3

```
df.shape
```

```
(303, 14)
```

299	45	1	typical	110	264	0	0	132	0	1.2	2

Count of columns

302	57	0	nontypical	130	236	0	2	174	0	0.0	2

```
df.count()
```

```
Age          303
Sex          303
ChestPain    303
RestBP       303
Chol         303
Fbs          303
RestECG      303
MaxHR        303
ExAng        303
Oldpeak      303
Slope        303
Ca           299
Thal         301
AHD          303
dtype: int64
```

Describing the dataset

```
df.describe()
```

	Age	Sex	RestBP	Chol	Fbs	RestECG	MaxH
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.0000

Displaying info about the dataset

min	29.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.0000

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 303 entries, 1 to 303
Data columns (total 14 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Age        303 non-null    int64  
 1   Sex        303 non-null    int64  
 2   ChestPain  303 non-null    object 
 3   RestBP     303 non-null    int64  
 4   Chol       303 non-null    int64  
 5   Fbs        303 non-null    int64  
 6   RestECG    303 non-null    int64  
 7   MaxHR      303 non-null    int64  
 8   ExAng      303 non-null    int64  
 9   Oldpeak    303 non-null    float64
 10  Slope      303 non-null    int64  
 11  Ca         299 non-null    float64
 12  Thal       301 non-null    object 
 13  AHD        303 non-null    object 
dtypes: float64(2), int64(9), object(3)
memory usage: 35.5+ KB
```

Displaying first 5 records

```
df.head()
```

	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak	Slope	
1	63	1	typical	145	233	1	2	150	0	2.3	3	(
2	67	1	asymptomatic	160	286	0	2	108	1	1.5	2	3
3	67	1	asymptomatic	120	229	0	2	129	1	2.6	2	2
4	37	1	nonanginal	130	250	0	0	187	0	3.5	3	(
5	41	0	nontypical	130	204	0	2	172	0	1.4	1	(

Displaying last 5 records

```
df.tail()
```

	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak	Slope
299	45	1	typical	110	264	0	0	132	0	1.2	2
300	68	1	asymptomatic	144	193	1	0	141	0	3.4	2
301	57	1	asymptomatic	130	131	0	0	115	1	1.2	2
302	57	0	nontypical	130	236	0	2	174	0	0.0	2
303	38	1	nonanginal	138	175	0	0	173	0	0.0	1

Handling Categorical Data in Python

Categorical features are typically stored as text values which represent various traits of the observations. For example, gender is described as Male (M) or Female (F), product type could be described as electronics, apparels, food etc.

The boxplot is a simple way of representing statistical data on a plot in which a rectangle is drawn to represent the second and third quartiles, usually with a vertical line inside to indicate the median value.

```
for col in df.columns:
  if len(df[col].unique()) < 10:
    print(col, df[col].unique())
    print("----------------------------------------")
    print(col, df[col].value_counts(), sep="\n")
    print("----------------------------------------")
```

```
Sex [1 0]
----------------------------------
Sex
1    206
0     97
Name: Sex, dtype: int64
----------------------------------
ChestPain ['typical' 'asymptomatic' 'nonanginal' 'nontypical']
----------------------------------
ChestPain
asymptomatic    144
nonanginal       86
nontypical       50
typical          23
Name: ChestPain, dtype: int64
----------------------------------
Fbs [1 0]
----------------------------------
Fbs
```

```
0    258
1     45
Name: Fbs, dtype: int64
-------------------------------------
RestECG [2 0 1]
-------------------------------------
RestECG
0    151
2    148
1      4
Name: RestECG, dtype: int64
-------------------------------------
ExAng [0 1]
-------------------------------------
ExAng
0    204
1     99
Name: ExAng, dtype: int64
-------------------------------------
Slope [3 2 1]
-------------------------------------
Slope
1    142
2    140
3     21
Name: Slope, dtype: int64
-------------------------------------
Ca [ 0.  3.  2.  1. nan]
-------------------------------------
Ca
0.0    176
1.0     65
2.0     38
3.0     20
Name: Ca, dtype: int64
-------------------------------------
Thal ['fixed' 'normal' 'reversable' nan]
-------------------------------------
Thal
```

Displaying null values

```
df.isnull()
```

	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak	Slo
1	False	False	False	False	False	False	False	False	False	False	Fa
2	False	False	False	False	False	False	False	False	False	False	Fa
3	False	False	False	False	False	False	False	False	False	False	Fa
4	False	False	False	False	False	False	False	False	False	False	Fa
5	False	False	False	False	False	False	False	False	False	False	Fa

Getting count of null values per column

```
df.isnull().sum()
```

```
Age          0
Sex          0
ChestPain    0
RestBP       0
Chol         0
Fbs          0
RestECG      0
MaxHR        0
ExAng        0
Oldpeak      0
Slope        0
Ca           4
Thal         2
AHD          0
dtype: int64
```

Getting total count of null values

```
df.isnull().sum().sum()
```

```
6
```

Filling null values

```
df["Ca"].fillna( method ='ffill', inplace = True)
```

```
df["Thal"].fillna( method ='ffill', inplace = True)
```

```
df.isnull().sum()
```

```
Age          0
Sex          0
ChestPain    0
RestBP       0
Chol         0
Fbs          0
RestECG      0
MaxHR        0
ExAng        0
Oldpeak      0
Slope        0
Ca           0
Thal         0
AHD          0
dtype: int64
```

Sorting according to age

```
df.sort_values("Age")
```

	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak	Slope
133	29	1	nontypical	130	204	0	2	202	0	0.0	1
102	34	1	typical	118	182	0	2	174	0	0.0	1
226	34	0	nontypical	118	210	0	0	192	0	0.7	1
284	35	1	nontypical	122	192	0	0	174	0	0.0	1
118	35	0	asymptomatic	138	183	0	0	182	0	1.4	1
...	...	...	...	...	...	...	...	...	...	...	...
43	71	0	nontypical	160	302	0	0	162	0	0.4	1
104	71	0	nonanginal	110	265	1	2	130	0	0.0	1
234	74	0	nontypical	120	269	0	2	121	1	0.2	1
258	76	0	nonanginal	140	197	0	1	116	0	1.1	2
162	77	1	asymptomatic	125	304	0	2	162	1	0.0	1

303 rows × 14 columns

Sorting in reverse order according to Age

```
df.sort_values("Age", ascending=False, kind="mergesort")
```

	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak	Slope
162	77	1	asymptomatic	125	304	0	2	162	1	0.0	1
258	76	0	nonanginal	140	197	0	1	116	0	1.1	2
234	74	0	nontypical	120	269	0	2	121	1	0.2	1
43	71	0	nontypical	160	302	0	0	162	0	0.4	1
104	71	0	nonanginal	110	265	1	2	130	0	0.0	1
...	...	...	...	...	...	...	...	...	...	...	...
169	35	1	asymptomatic	126	282	0	2	156	1	0.0	1
284	35	1	nontypical	122	192	0	0	174	0	0.0	1
102	34	1	typical	118	182	0	2	174	0	0.0	1
226	34	0	nontypical	118	210	0	0	192	0	0.7	1
133	29	1	nontypical	130	204	0	2	202	0	0.0	1

Sorting by Multiple Columns in Ascending Order

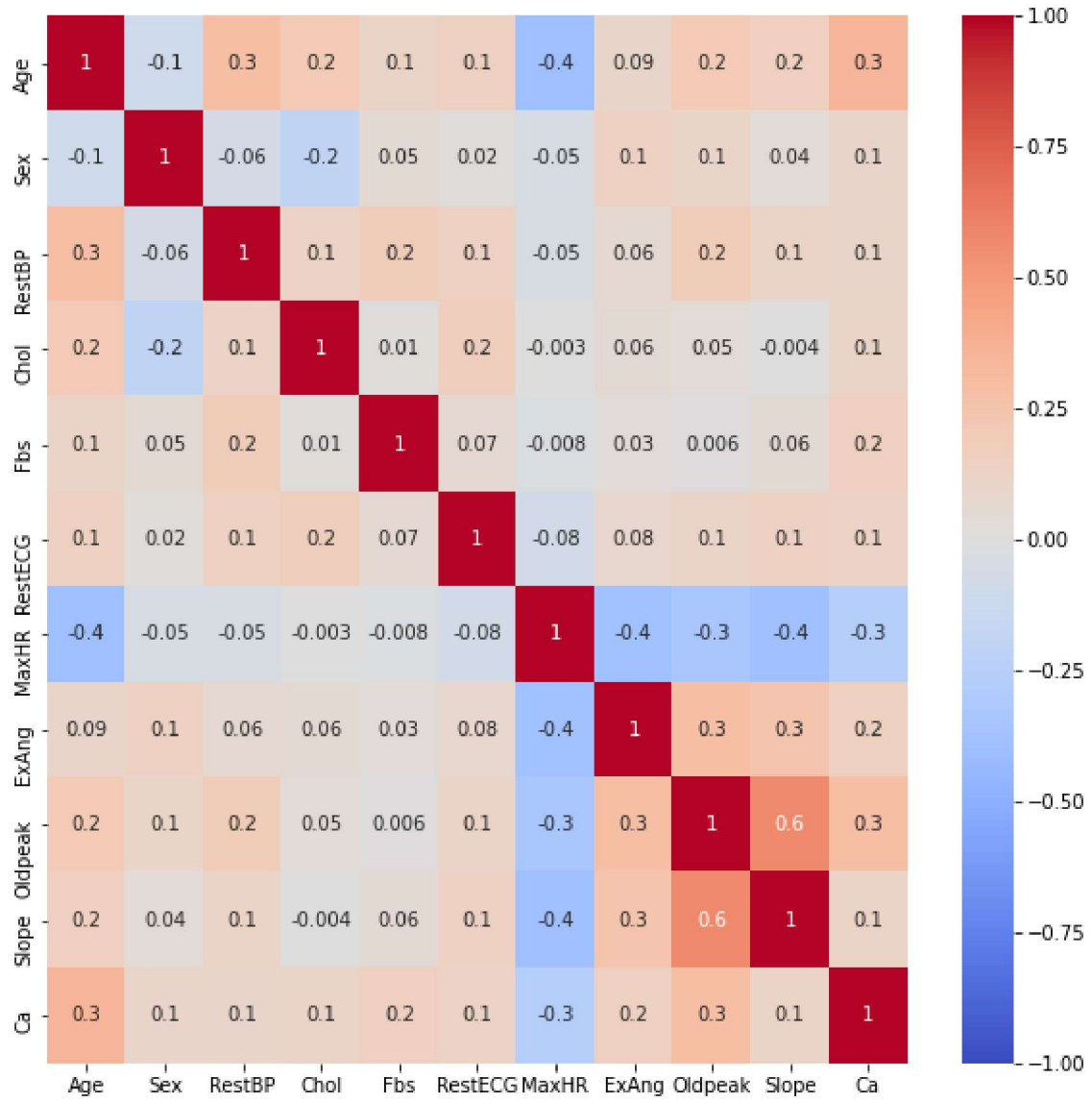
```
df.sort_values(by=["Age", "RestBP"])
```

	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak	Slope
133	29	1	nontypical	130	204	0	2	202	0	0.0	1
102	34	1	typical	118	182	0	2	174	0	0.0	1
226	34	0	nontypical	118	210	0	0	192	0	0.7	1
139	35	1	asymptomatic	120	198	0	0	130	1	1.6	2
284	35	1	nontypical	122	192	0	0	174	0	0.0	1
...	...	...	...	...	...	...	...	...	...	...	...
274	71	0	asymptomatic	112	149	0	0	125	0	1.6	2
43	71	0	nontypical	160	302	0	0	162	0	0.4	1
234	74	0	nontypical	120	269	0	2	121	1	0.2	1
258	76	0	nonanginal	140	197	0	1	116	0	1.1	2
162	77	1	asymptomatic	125	304	0	2	162	1	0.0	1

303 rows × 14 columns

Heatmap

```
plt.figure(figsize=(10, 10))
sns.heatmap(df.corr(), annot=True, center=0, vmin=-1, vmax=1, f
plt.show()
```




```
# sns.heatmap(df["ChestPain"])
# plt.show()

# a = df.iloc[:, [1, 3]]
# print(a)

X = df.iloc[:, :-1].values
# print(X)

y = df.iloc[:, -1].values
print(y)
```

```
['No' 'Yes' 'Yes' 'No' 'No' 'No' 'Yes' 'No' 'Yes' 'Yes' 'No' 'No' 'Yes'
```

```
'No' 'No' 'No' 'Yes' 'No' 'No' 'No' 'No' 'No' 'Yes' 'Yes' 'Yes' 'No' 'No'
'No' 'No' 'Yes' 'No' 'Yes' 'Yes' 'No' 'No' 'No' 'Yes' 'Yes' 'Yes' 'No'
'Yes' 'No' 'No' 'No' 'Yes' 'Yes' 'No' 'Yes' 'No' 'No' 'No' 'No' 'Yes'
'No' 'Yes' 'Yes' 'Yes' 'Yes' 'No' 'No' 'Yes' 'No' 'Yes' 'No' 'Yes' 'Yes'
'Yes' 'No' 'Yes' 'Yes' 'No' 'Yes' 'Yes' 'Yes' 'Yes' 'No' 'Yes' 'No' 'No'
'Yes' 'No' 'No' 'No' 'Yes' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'Yes' 'No'
'No' 'No' 'Yes' 'Yes' 'Yes' 'No' 'No' 'No' 'No' 'No' 'No' 'Yes' 'No'
'Yes' 'Yes' 'Yes' 'Yes' 'Yes' 'Yes' 'No' 'Yes' 'Yes' 'No' 'No' 'No' 'Yes'
'Yes' 'Yes' 'Yes' 'No' 'Yes' 'Yes' 'No' 'Yes' 'Yes' 'No' 'No' 'No' 'No'
'No' 'No' 'No' 'No' 'Yes' 'Yes' 'Yes' 'No' 'No' 'Yes' 'No' 'Yes' 'No'
'Yes' 'Yes' 'No' 'No' 'No' 'No' 'No' 'No' 'Yes' 'Yes' 'Yes' 'Yes' 'Yes'
'Yes' 'No' 'No' 'Yes' 'No' 'No' 'No' 'No' 'No' 'No' 'Yes' 'No' 'Yes' 'No'
'Yes' 'No' 'Yes' 'Yes' 'No' 'Yes' 'No' 'No' 'Yes' 'Yes' 'No' 'No' 'Yes'
'No' 'No' 'Yes' 'Yes' 'Yes' 'No' 'Yes' 'Yes' 'Yes' 'No' 'Yes' 'No' 'No'
'No' 'Yes' 'No' 'No' 'No' 'No' 'No' 'Yes' 'Yes' 'Yes' 'No' 'Yes' 'No'
'Yes' 'No' 'Yes' 'Yes' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'Yes'
'Yes' 'No' 'No' 'No' 'Yes' 'Yes' 'No' 'Yes' 'Yes' 'No' 'No' 'Yes' 'Yes'
'Yes' 'No' 'No' 'No' 'No' 'No' 'Yes' 'No' 'Yes' 'Yes' 'Yes' 'Yes' 'No'
'No' 'Yes' 'No' 'No' 'No' 'No' 'No' 'No' 'No' 'Yes' 'No' 'Yes' 'No' 'No'
'Yes' 'Yes' 'Yes' 'Yes' 'Yes' 'No' 'Yes' 'No' 'Yes' 'No' 'Yes' 'No' 'No'
'No' 'Yes' 'No' 'Yes' 'No' 'Yes' 'No' 'Yes' 'Yes' 'Yes' 'No' 'No' 'No'
'Yes' 'No' 'Yes' 'Yes' 'Yes' 'No' 'Yes' 'Yes' 'Yes' 'Yes' 'Yes' 'Yes'
'No']
```

```
from sklearn.model_selection import train_test_split
# X: Independant variable, y : Dependant variable.
# Parameters for train_test_split:
            # X - Matrix features
            # y - Dependant variable vector
            # Test size - 20% recommended
            # random_state - So that we get the same randomized
X_train, X_test, y_train, y_test = train_test_split(X, y, test_
```

```
X_train
```

```
array([[58, 1, 'asymptomatic', ..., 2, 3.0, 'reversable'],
       [54, 1, 'asymptomatic', ..., 2, 1.0, 'reversable'],
       [56, 1, 'asymptomatic', ..., 2, 1.0, 'normal'],
       ...,
       [62, 1, 'asymptomatic', ..., 2, 2.0, 'reversable'],
       [54, 1, 'asymptomatic', ..., 2, 2.0, 'normal'],
       [57, 1, 'asymptomatic', ..., 2, 1.0, 'fixed']], dtype=object)
```

```
X_test
```

```
array([[43, 1, 'asymptomatic', 110, 211, 0, 0, 161, 0, 0.0, 1, 0.0,
        'reversable'],
       [68, 1, 'nonanginal', 118, 277, 0, 0, 151, 0, 1.0, 1, 1.0,
        'reversable'],
       [59, 1, 'asymptomatic', 138, 271, 0, 2, 182, 0, 0.0, 1, 0.0,
        'normal'],
       [64, 1, 'asymptomatic', 145, 212, 0, 2, 132, 0, 2.0, 2, 2.0,
        'fixed'],
```

```
       [60, 0, 'asymptomatic', 158, 305, 0, 2, 161, 0, 0.0, 1, 0.0,
        'normal'],
       [41, 1, 'nontypical', 120, 157, 0, 0, 182, 0, 0.0, 1, 0.0,
        'normal'],
       [42, 1, 'nonanginal', 130, 180, 0, 0, 150, 0, 0.0, 1, 0.0,
        'normal'],
       [65, 1, 'asymptomatic', 135, 254, 0, 2, 127, 0, 2.8, 2, 1.0,
        'reversable'],
       [62, 0, 'asymptomatic', 138, 294, 1, 0, 106, 0, 1.9, 2, 3.0,
        'normal'],
       [64, 1, 'asymptomatic', 120, 246, 0, 2, 96, 1, 2.2, 3, 1.0,
        'normal'],
       [65, 1, 'asymptomatic', 120, 177, 0, 0, 140, 0, 0.4, 1, 0.0,
        'reversable'],
       [62, 1, 'nontypical', 128, 208, 1, 2, 140, 0, 0.0, 1, 0.0,
        'normal'],
       [57, 1, 'nontypical', 154, 232, 0, 2, 164, 0, 0.0, 1, 1.0,
        'normal'],
       [66, 1, 'asymptomatic', 112, 212, 0, 2, 132, 1, 0.1, 1, 1.0,
        'normal'],
       [55, 1, 'nontypical', 130, 262, 0, 0, 155, 0, 0.0, 1, 0.0,
        'normal'],
       [38, 1, 'nonanginal', 138, 175, 0, 0, 173, 0, 0.0, 1, 1.0,
        'normal'],
       [54, 1, 'nonanginal', 125, 273, 0, 2, 152, 0, 0.5, 3, 1.0,
        'normal'],
       [41, 0, 'nonanginal', 112, 268, 0, 2, 172, 1, 0.0, 1, 0.0,
        'normal'],
       [48, 0, 'nonanginal', 130, 275, 0, 0, 139, 0, 0.2, 1, 0.0,
        'normal'],
       [54, 1, 'asymptomatic', 110, 206, 0, 2, 108, 1, 0.0, 2, 1.0,
        'normal'],
       [56, 0, 'nontypical', 140, 294, 0, 2, 153, 0, 1.3, 2, 0.0,
        'normal'],
       [57, 1, 'asymptomatic', 130, 131, 0, 0, 115, 1, 1.2, 2, 1.0,
        'reversable'],
       [65, 0, 'nonanginal', 155, 269, 0, 0, 148, 0, 0.8, 1, 0.0,
        'normal'],
       [57, 1, 'asymptomatic', 165, 289, 1, 2, 124, 0, 1.0, 2, 3.0,
        'reversable'],
       [51, 1, 'nonanginal', 100, 222, 0, 0, 143, 1, 1.2, 2, 0.0,
        'normal'],
       [48, 1, 'nonanginal', 124, 255, 1, 0, 175, 0, 0.0, 1, 2.0,
        'normal'],
       [51, 1, 'asymptomatic', 140, 298, 0, 0, 122, 1, 4.2, 2, 3.0,
        'reversable'],
       [45, 1, 'asymptomatic', 104, 208, 0, 2, 148, 1, 3.0, 2, 0.0,
        'normal'],
       [66, 0, 'typical', 150, 226, 0, 0, 114, 0, 2.6, 3, 0.0, 'normal'],
       [55, 1, 'asymptomatic', 140, 217, 0, 0, 111, 1, 5.6, 3, 0.0,
```

y_train

```
array(['Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No',
       'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'No',
       'No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'No', 'No', 'Yes',
       'No', 'No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'No', 'Yes',
       'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'No', 'No', 'Yes', 'Yes',
       'Yes', 'No', 'No', 'No', 'No', 'Yes', 'Yes', 'No', 'No', 'No',
```

```
       'Yes', 'Yes', 'Yes', 'No', 'No', 'No', 'No', 'Yes', 'No', 'No',
       'No', 'No', 'No', 'No', 'Yes', 'No', 'No', 'Yes', 'No', 'Yes',
       'No', 'Yes', 'Yes', 'No', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No',
       'No', 'Yes', 'No', 'No', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'No',
       'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes',
       'Yes', 'No', 'Yes', 'Yes', 'No', 'No', 'No', 'No', 'No', 'No',
       'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'No', 'No', 'No', 'No',
       'Yes', 'No', 'No', 'Yes', 'No', 'No', 'No', 'Yes', 'Yes', 'Yes',
       'Yes', 'No', 'No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'No',
       'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes',
       'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No', 'No', 'No',
       'No', 'No', 'Yes', 'No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No',
       'No', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'No',
       'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No',
       'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'No', 'No', 'Yes',
       'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'No',
       'Yes', 'No', 'No', 'No', 'Yes', 'No', 'Yes', 'No', 'No', 'No',
       'No', 'No', 'Yes', 'Yes', 'No', 'No', 'No', 'No', 'No', 'Yes',
       'Yes', 'Yes'], dtype=object)
```

```
y_test
```

```
array(['No', 'No', 'No', 'Yes', 'Yes', 'No', 'No', 'Yes', 'Yes', 'Yes',
       'No', 'No', 'Yes', 'Yes', 'No', 'No', 'No', 'No', 'No', 'Yes',
       'No', 'Yes', 'No', 'Yes', 'No', 'No', 'Yes', 'No', 'No', 'Yes',
       'Yes', 'Yes', 'Yes', 'No', 'No', 'Yes', 'No', 'No', 'Yes', 'Yes',
       'No', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', 'No', 'No',
       'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'No', 'No', 'Yes', 'No',
       'Yes'], dtype=object)
```

Confusion Matrix

```
ones = np.ones(50, dtype=int)
print(ones)
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

```
zeros = np.zeros(50, dtype=int)
print(zeros)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

```
y_actual = np.concatenate((ones, zeros))
print(y_actual)
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

```
y_actual
```

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

```
# onesPredict = np.ones(45, dtype=int)
# print(ones)
```

```
# zerosPredict = np.zeros(55, dtype=int)
# print(zeros)
```

```
# y_pred = np.concatenate((ones, zeros))
# print(y_pred)
```

```
y_actual = np.array([0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1,
       0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0,
       0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1,
       0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0])
```

```
y_pred = np.array([0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1,
       0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0,
       0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1,
       0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1])
```

```
# random.shuffle(y_actual)
```

```
# random.shuffle(y_pred)
```

```
print(y_actual)
```

```
[0 1 1 1 1 0 0 0 0 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 0 0 1 1 1 0 0 0 1 0 0
 1 1 0 0 0 0 0 0 1 1 1 0 1 0 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0 0 0 0 0 1 1 1 1
 1 1 1 0 0 1 0 0 1 0 0 1 0 0 1 1 1 1 1 1 0 1 0 1 1 1 0]
```

```
print(y_pred)
```

```
[0 1 1 1 1 0 0 0 0 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 0 0 1 1 1 0 0 0 1 0 0
 1 1 0 0 0 0 0 0 1 1 1 0 1 0 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0 0 0 0 0 1 1 1 1
 1 1 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1]
```

```
classification_report(y_actual, y_pred)
```

```
'              precision    recall  f1-score   support\n\n           0       0.85
47\n           1       0.98      0.85      0.91        53\n\n    accuracy
91       100\n   macro avg       0.92      0.91      0.91       100\nweighted avg
```

```
con_mat = confusion_matrix(y_actual, y_pred)
print(con_mat)
```

```
[[46  1]
 [ 8 45]]
```

```
tn, fp, fn, tp = con_mat[0][0], con_mat[0][1], con_mat[1][0], c
print("True Negatives: ",tn)
print("False Positives: ",fp)
print("False Negatives: ",fn)
print("True Positives: ",tp)
```

```
True Negatives:  46
False Positives:  1
False Negatives:  8
True Positives:  45
```

Accuracy

```
Accuracy = (tn+tp)*100/(tp+tn+fp+fn)
print("Accuracy: {:0.2f}%".format(Accuracy))
```

```
Accuracy: 91.00%
```

Precision

```
Precision = tp/(tp+fp)
print("Precision: {:0.2f}".format(Precision))
```

```
Precision: 0.98
```

Recall

```
Recall = tp/(tp+fn)
print("Recall: {:0.2f}".format(Recall))
```

```
Recall: 0.85
```

F1 Score

```
f1 = (2*Precision*Recall)/(Precision + Recall)
print("F1 Score: {:0.2f}".format(f1))
```

```
F1 Score: 0.91
```

F-beta score calculation

```
def fbeta(precision, recall, beta):
    return ((1+pow(beta,2))*precision*recall)/(pow(beta,2)*prec

f2 = fbeta(Precision, Recall, 2)
f0_5 = fbeta(Precision, Recall, 0.5)

print("F2 {:0.2f}".format(f2))
print("\nF0.5 {:0.2f}".format(f0_5))
```

```
F2 0.87

F0.5 0.95
```

Specificity

```
Specificity = tn/(tn+fp)
print("Specificity: {:0.2f}".format(Specificity))
```

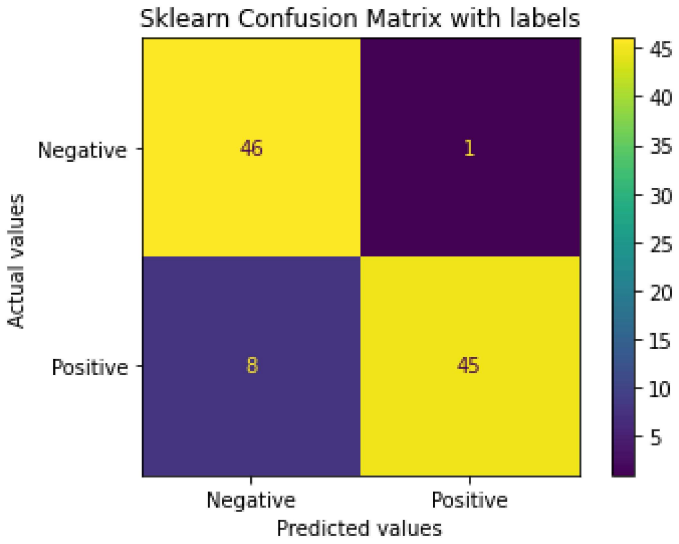
```
Specificity: 0.98
```

```
cmd_obj = ConfusionMatrixDisplay(con_mat, display_labels=["Nega

cmd_obj.plot()
```

```
cmd_obj.ax_.set(
                title='Sklearn Confusion Matrix with labels',
                xlabel='Predicted values',
                ylabel='Actual values')

plt.show()
```



Accuracy

```
91.0
```

PUNE INSTITUTE OF COMPUTER TECHNOLOGY

Subject: Machine Learning (LP-1 LAB)

Name: Aditya Kangune **Roll No. : 33323**

Batch: K11 **Academic Year: 2021-22**

Assignment 2
Regression technique

Problem Statement:

This data consists of temperatures of INDIA averaging the temperatures of all places month-wise.

Temperatures values are recorded in CELSIUS

A. Apply Linear Regression using suitable library function and predict the Month-wise temperature.
B. Assess the performance of regression models using MSE, MAE, and R-Square metrics
C. Visualize a simple regression model.

Objective:

This assignment will help the students to realize how Linear Regression can be used and predictions using the same can be performed.

Theory:

Definition of Linear Regression:

1. In layman's terms, we can define linear regression as it is used for learning the linear relationship between the target and one or more

forecasters, and it is probably one of the most popular and well-inferential algorithms in statistics.

2. Linear regression endeavors to demonstrate the connection between two variables by fitting a linear equation to observed information.
3. One variable is viewed as an explanatory variable, and the other is viewed as a dependent variable.

Types of Linear Regression:

Normally, linear regression is divided into two types: Multiple linear regression and Simple linear regression.

1. Multiple Linear Regression:

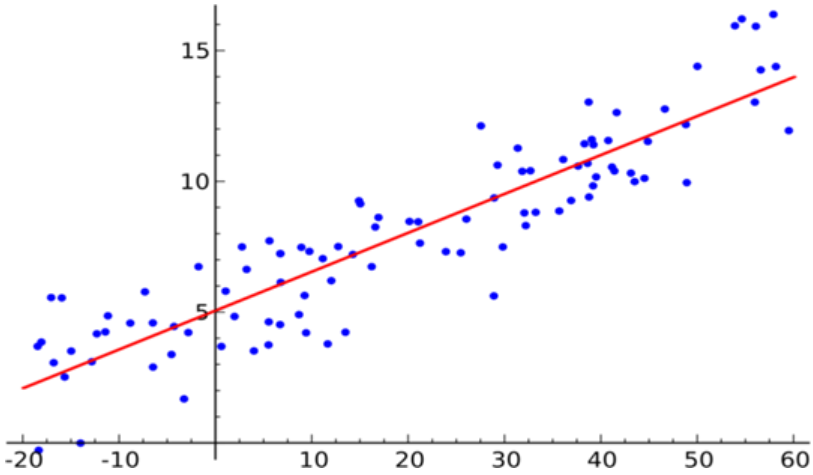
A. In this type of linear regression, we always attempt to discover the relationship between two or more independent variables or inputs and the corresponding dependent variable or output and the independent variables can be either continuous or categorical.

B. This linear regression analysis is very helpful in several ways like it helps in foreseeing trends, future values, and moreover predicting the impacts of changes.

2. Simple Linear Regression:

In simple linear regression, we aim to reveal the relationship between a single independent variable or you can say input, and a corresponding dependent variable or output. We can discuss this in a simple line as

$$y = \beta 0 + \beta 1 x + \varepsilon$$



Simple Linear Regression graph [3]

Here, Y speaks to the output or dependent variable, β0 and β1 are two obscure constants that speak to the intercept and coefficient that is slope separately, and the error term is ε Epsilon.
We can also discuss this in the form of a graph and here is a sample simple linear regression model graph.

What Actually is Simple Linear Regression?

It can be described as a method of statistical analysis that can be used to study the relationship between two quantitative variables.

Primarily, there are two things that can be found out by using the method of simple linear regression:

1. **Strength of the relationship between the given duo of variables:**

For example, the relationship between global warming and the melting of glaciers

2. How much the value of the dependent variable is at a given value of the independent variable:
For example, the amount of melting of a glacier at a certain level of global warming or temperature

1. Regression models are used for the lab
2. orated explanation of the relationship between two given variables. There are certain types of regression models like logistic regression models, nonlinear regression models, and linear regression models.
3. The linear regression model fits a straight line into the summarized data to establish the relationship between two variables.

Assumptions of Linear Regression:

To conduct a simple linear regression, one has to make certain assumptions about the data. This is because it is a parametric test.

The assumptions used while performing a simple linear regression are as follows:

- **Homogeneity of variance (homoscedasticity):** One of the main predictions in a simple linear regression method is that the size of the error stays constant.

This simply means that in the value of the independent variable, the error size never changes significantly.

- **Independence of observations:** All the relationships between the observations are transparent, which means that nothing is hidden, and only valid sampling methods are used during the collection of data.

- **Normality:** There is a normal rate of flow in the data.

These three are the assumptions of regression methods.

However, there is one additional assumption that has to be taken into consideration while specifically conducting a linear regression.

- **The line is always a straight line:** There is no curve or grouping factor during the conduction of linear regression. There is a linear relationship between the variables (dependent variable and independent variable). If the data fails the assumptions of homoscedasticity or normality, a nonparametric test might be used. (For example, the Spearman rank test)

Example of data that fails to meet the assumptions:

One may think that cured meat consumption and the incidence of colorectal cancer in the U.S have a linear relationship.

But later on, it comes to the knowledge that there is a very high range difference between the collection of data of both the variables.

Since the homoscedasticity assumption is being violated here, there can be no linear regression test.

However, a Spearman rank test can be performed to know about the relationship between the given variables.

Applications of Simple Linear Regression:

1. **Marks scored by students based on the number of hours studied (ideally):**

Here marks scored in exams are dependent and the number of hours studied is independent.

2. **Predicting crop yields based on the amount of rainfall:**

Yield is a dependent variable while the measure of precipitation is an independent variable.

3. **Predicting the Salary of a person based on years of experience:**

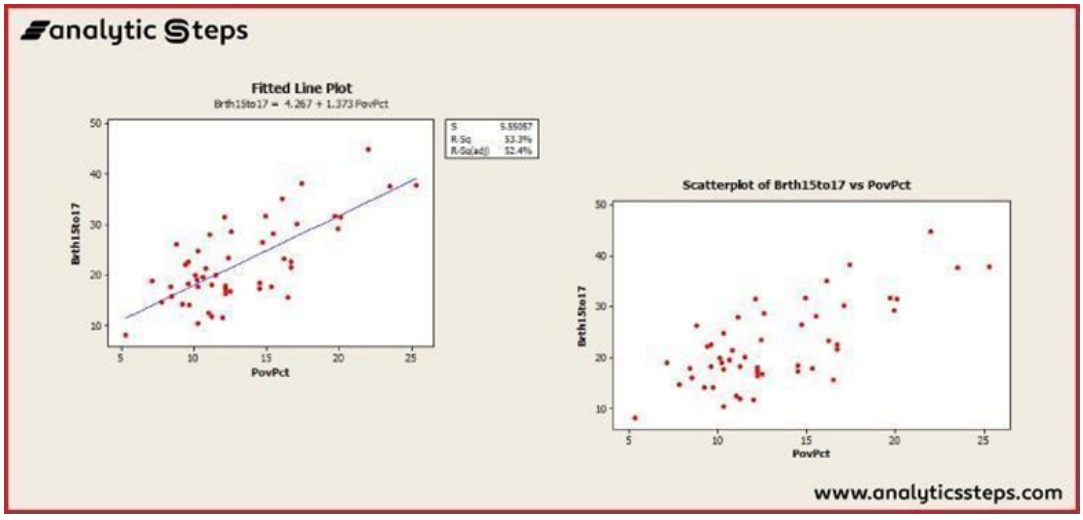
Therefore, Experience becomes independent while Salary turns into the dependent variable.

Limitations of Simple Linear Regression:

1. Indeed, even the best information doesn't recount a total story. Regression investigation is ordinarily utilized in examination to set up that a relationship exists between variables.
2. However, correlation isn't equivalent to causation: a connection between two variables doesn't mean one causes the other to occur.
3. Indeed, even a line in a simple linear regression that fits the information focuses well may not ensure circumstances and logical results relationship.
4. Utilizing a linear regression model will permit you to find whether a connection between variables exists by any means.
5. To see precisely what that relationship is and whether one variable causes another, you will require extra examination and statistical analysis.

Examples of Simple Linear Regression:

1. Now, let’s move towards understanding simple linear regression with the help of an example.
2. We will take an example of teen birth rate and poverty level data.
3. This dataset of size n = 51 is for the 50 states and the District of Columbia in the United States. The variables are y = year 2002 birth rate for every 1000 females 15 to 17 years of age and x = destitution rate, which is the percent of the state's populace living in families with wages underneath the governmentally characterized neediness level. (Information source: Mind On Statistics, 3rd version, Utts and Heckard).
4. Below is the graph in which you can see the (birth rate on the vertical) is indicating a normally linear relationship, on average, with a positive slope.

5. 

Example graph of simple linear regression [3]

6. As the poverty level builds, the birth rate for 15 to 17-year-old females will in general increment too.

Here is another graph which is showing a regression line superimposed on the data:

1. The condition of the fitted regression line is given close to the highest point of the plot. The condition should express that it is for the "average" birth rate (or "anticipated" birth rate would be alright as well) as a regression condition portrays the normal estimation of y as a component of at least one x-variables. In statistical documentation, the condition could be composed y^=4.267+1.373x.
2. The interpretation of the slope (value = 1.373) is that the 15 to 17-year-old birth rate increases 1.373 units, on average, for each one unit (one percent) increase in the poverty rate.
3. The translation of the intercept (value=4.267) is that if there were states with a population rate = 0, the anticipated normal for the 15 to 17-year-old birth rate would be 4.267 for those states.
4. Since there are no states with a poverty rate = 0 this understanding of the catch isn't basically significant for this model.
5. In the chart with a repression line present, we additionally observe the data that s = 5.55057 and r2 = 53.3%.

6. The estimation of s discloses to us generally the standard deviation of the contrasts between the y-estimations of individual perceptions and expectations of y dependent on the regression line.
7. The estimation of r2 can be deciphered to imply that destitution rates
8. "clarify" 53.3% of the noticed variety in the 15 to 17-year-old normal birth paces of the states.
9. The R2 (adj) value (52.4%) is a change in accordance with R2 dependent on the number of x- variables in the model (just one here) and the example size. With just a single x-variable, the charged R2 isn't significant.

Conclusion:

Simple linear regression is a regression model that figures out the relationship between one independent variable and one dependent variable using a straight line.

Implementation:

Simple Linear Regression

Importing the libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import math
```

Importing the dataset

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

```
dataset = pd.read_csv("/content/drive/MyDrive/Datasets/weight-h
print(dataset)
```

```
      Gender     Height      Weight
0       Male  73.847017  241.893563
1       Male  68.781904  162.310473
2       Male  74.110105  212.740856
3       Male  71.730978  220.042470
4       Male  69.881796  206.349801
...      ...        ...         ...
9995  Female  66.172652  136.777454
9996  Female  67.067155  170.867906
9997  Female  63.867992  128.475319
9998  Female  69.034243  163.852461
9999  Female  61.944246  113.649103

[10000 rows x 3 columns]
```

```
X = dataset.iloc[:,1:2]
y = dataset.iloc[:,-1]
```

```
print(X)
```

```
         Height
0     73.847017
```

```
1     68.781904
2     74.110105
3     71.730978
4     69.881796
...         ...
9995  66.172652
9996  67.067155
9997  63.867992
9998  69.034243
9999  61.944246

[10000 rows x 1 columns]
```

```
print(y)
```

```
0       241.893563
1       162.310473
2       212.740856
3       220.042470
4       206.349801
           ...
9995    136.777454
9996    170.867906
9997    128.475319
9998    163.852461
9999    113.649103
Name: Weight, Length: 10000, dtype: float64
```

Splitting the dataset into the Training set and Test set

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_
```

Training the Simple Linear Regression model on the Training set

```
from sklearn.linear_model import LinearRegression

regressor = LinearRegression()
# Upto here was the builing part, now we have to train the mode


# To connect our model we use the fit method

# X_train contains the features of the dataset
# y_train contins the dependant varaibles
regressor.fit(X_train, y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

Predicting the Test set results

```
regressor.predict(X_test)
# This returns a vector containg the predicted data
```

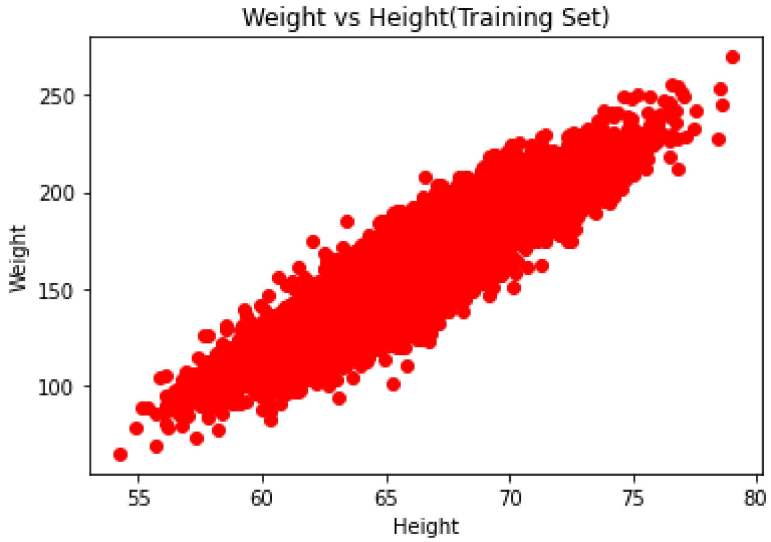
```
array([148.7894772 , 168.43520123, 224.31884497, ..., 159.17576427,
       155.86404539, 144.83449257])
```

```
y_pred = regressor.predict(X_test)
```

Visualising the Training set results

```
plt.title("Weight vs Height(Training Set)")
plt.xlabel("Height")
plt.ylabel("Weight")
# scatter method allows us to put points/coordinates
plt.scatter(X_train, y_train, color="red") # Real vlaues
```

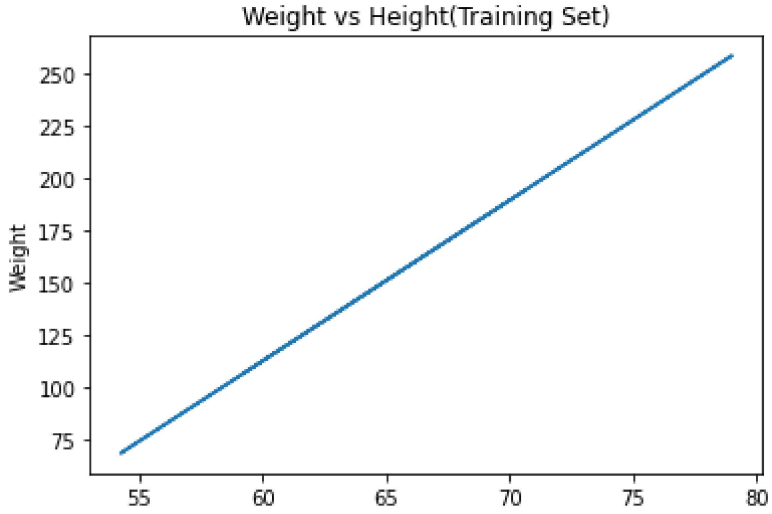
```
<matplotlib.collections.PathCollection at 0x7fc9687762d0>
```




```
# plot method is used to plot the curve of a function(y=b0+b1*x
plt.title("Weight vs Height(Training Set)")
plt.xlabel("Height")
plt.ylabel("Weight")
```

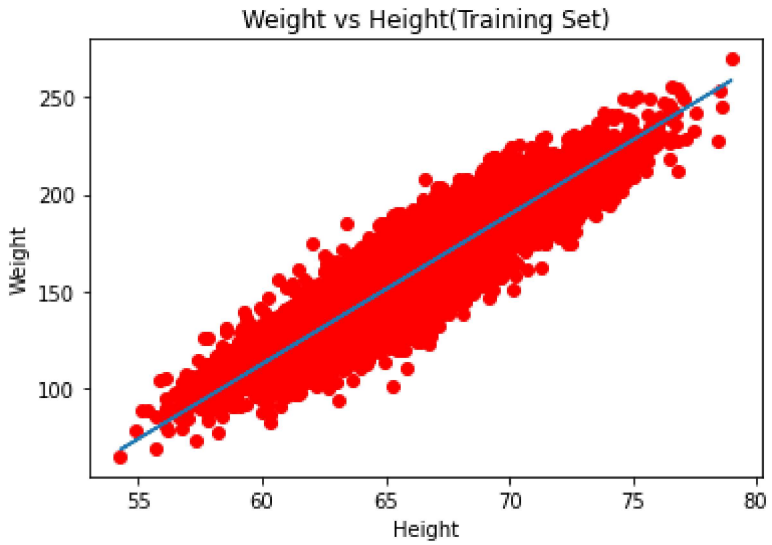
```
plt.plot(X_train, regressor.predict(X_train)) # Best fit line o
```

```
[<matplotlib.lines.Line2D at 0x7fc96aa54490>]
```




```
plt.title("Weight vs Height(Training Set)")
plt.xlabel("Height")
plt.ylabel("Weight")
plt.scatter(X_train, y_train, color="red")
plt.plot(X_train, regressor.predict(X_train))
# show is used to display the graphic in the output
plt.show()

# Red dots are real values of salary (x_train, y_train)
# Blue line is the best fit line on training X_train and predic
```



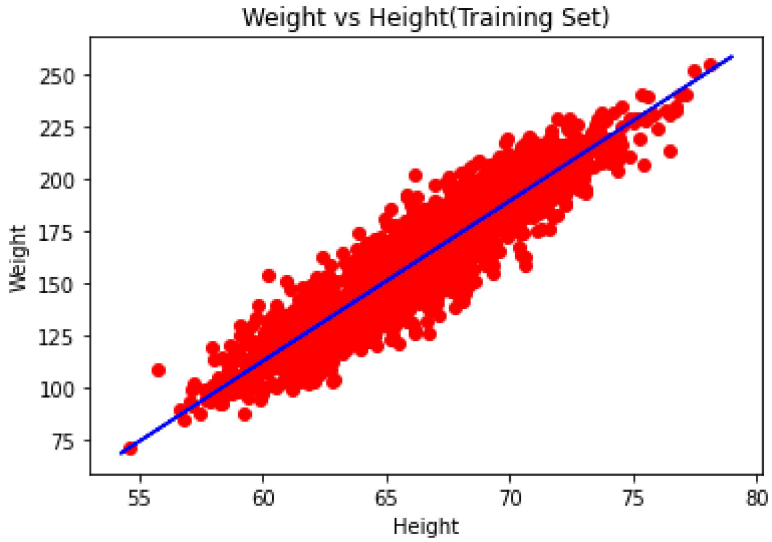
Visualising the Test set results

```
plt.title("Weight vs Height(Training Set)")
```

```
plt.xlabel("Height")
plt.ylabel("Weight")
plt.scatter(X_test, y_test, color="red")

# Predicted salries of test set will on the same rgression line
plt.plot(X_train, regressor.predict(X_train), color="blue")
plt.show()

# Red are new observations(test set)
# Blue is our best fit line after traning on the available trai
```



Making a single prediction (for example the salary of an employee with 12 years of experience)

```
input_value = int(input("Enter Height: "))
print(regressor.predict([[input_value]]))
```

```
Enter Height: 185
[1076.04180898]
```

Therefore, our model predicts that the salary of an employee with 12 years of experience is $ 138967,5.

Important note: Notice that the value of the feature (12 years) was input in a double pair of square brackets. That's because the "predict" method always expects a 2D array as the format of its inputs. And putting 12 into a double pair of square brackets makes the input exactly a 2D array. Simply put:

$12 \rightarrow \text{scalar}$

$[12] \rightarrow 1\text{D array}$

$[[12]] \rightarrow 2\text{D array}$

Getting the final linear regression equation with the values of the coefficients

```
print(regressor.coef_)
print(regressor.intercept_)
```

```
[7.70936331]
-350.1904028560757
```

Therefore, the equation of our simple linear regression model is:

$$\text{Weight} = 7.70936331 \times \text{Height} + -350.1904028560757.$$

Important Note: To get these coefficients we called the "coef_" and "intercept_" attributes from our regressor object. Attributes in Python are different than methods and usually return a simple value or an array of values.

```
from sklearn.metrics import mean_squared_error

mse = mean_squared_error(y_test, y_pred)
print("MSE: ", mse)
```

```
MSE:  146.53677213957428
```

```
rmse = math.sqrt(mse)
print(rmse)
```

```
12.105237384684957
```

Multiple Linear Regression

Importing the libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

Importing the dataset

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

```
dataset = pd.read_csv('/content/drive/MyDrive/Datasets/temperat
print(dataset)
```

```
     YEAR    JAN    FEB    MAR  ...  JAN-FEB  MAR-MAY  JUN-SEP  OCT-DEC
0    1901  22.40  24.14  29.07  ...    23.27    31.46    31.27    27.25
1    1902  24.93  26.58  29.77  ...    25.75    31.76    31.09    26.49
2    1903  23.44  25.03  27.83  ...    24.24    30.71    30.92    26.26
3    1904  22.50  24.73  28.21  ...    23.62    30.95    30.66    26.40
4    1905  22.00  22.83  26.68  ...    22.25    30.00    31.33    26.57
..    ...    ...    ...    ...  ...      ...      ...      ...      ...
112  2013  24.56  26.59  30.62  ...    25.58    32.58    31.33    27.83
113  2014  23.83  25.97  28.95  ...    24.90    31.82    32.00    27.81
114  2015  24.58  26.89  29.07  ...    25.74    31.68    31.87    28.27
115  2016  26.94  29.72  32.62  ...    28.33    34.57    32.28    30.03
116  2017  26.45  29.46  31.60  ...    27.95    34.13    32.41    29.69

[117 rows x 18 columns]
```

```
dataset.head()
```

	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DE
0	1901	22.40	24.14	29.07	31.91	33.41	33.18	31.21	30.39	30.47	29.97	27.31	24.4

```
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

3	1904	22.50	24.73	28.21	32.02	32.64	32.07	30.36	30.09	30.04	29.20	26.36	23.6

```
dataset.dtypes
```

```
YEAR         int64
JAN        float64
FEB        float64
MAR        float64
APR        float64
MAY        float64
JUN        float64
JUL        float64
AUG        float64
SEP        float64
OCT        float64
NOV        float64
DEC        float64
ANNUAL     float64
JAN-FEB    float64
MAR-MAY    float64
JUN-SEP    float64
OCT-DEC    float64
dtype: object
```

```
dataset.columns
```

```
Index(['YEAR', 'JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL', 'AUG', 'SEP',
       'OCT', 'NOV', 'DEC', 'ANNUAL', 'JAN-FEB', 'MAR-MAY', 'JUN-SEP',
       'OCT-DEC'],
      dtype='object')
```

```
dataset.describe()
```

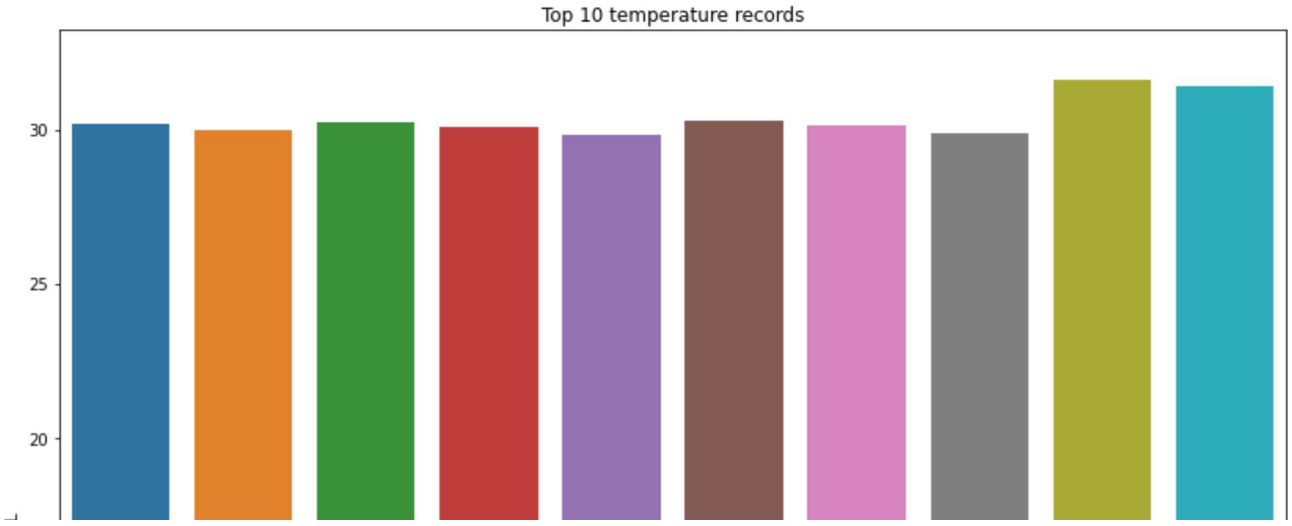
YEAR	JAN	FEB	MAR	APR	MAY	J

```
dataset.isnull().sum()
```

```
YEAR       0
JAN        0
FEB        0
MAR        0
APR        0
MAY        0
JUN        0
JUL        0
AUG        0
SEP        0
OCT        0
NOV        0
DEC        0
ANNUAL     0
JAN-FEB    0
MAR-MAY    0
JUN-SEP    0
OCT-DEC    0
dtype: int64
```

```
top_10_data = dataset.nlargest(10, "ANNUAL")
plt.figure(figsize=(14,12))
plt.title("Top 10 temperature records")
sns.barplot(x=top_10_data.YEAR, y=top_10_data.ANNUAL)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4b82d90d50>
```




```
X = dataset.iloc[:,1:13].values
y = dataset.iloc[:,13].values
```

```
print(X)
```

```
[[22.4  24.14 29.07 ... 29.97 27.31 24.49]
 [24.93 26.58 29.77 ... 29.12 26.31 24.04]
 [23.44 25.03 27.83 ... 29.04 26.08 23.65]
 ...
 [24.58 26.89 29.07 ... 31.04 28.1  25.67]
 [26.94 29.72 32.62 ... 31.98 30.11 28.01]
 [26.45 29.46 31.6  ... 32.29 29.6  27.18]]
```


```
print(y)
```

```
[28.96 29.22 28.47 28.49 28.3  28.73 28.65 28.83 28.38 28.53 28.62 28.95
 28.67 28.66 28.94 28.82 28.11 28.66 28.66 28.76 28.86 28.8  28.74 28.8
 28.67 28.7  28.59 28.98 28.76 28.65 29.15 29.09 28.49 29.03 28.76 28.71
 28.7  28.7  28.85 28.88 29.46 28.98 28.8  28.89 28.97 29.37 28.84 28.73
 28.89 28.47 29.09 29.16 29.43 28.92 28.76 28.63 28.64 29.34 29.02 29.31
 28.72 28.89 29.04 29.09 29.16 29.41 29.14 29.07 29.61 29.47 29.15 29.31
 29.44 29.26 28.89 29.27 29.41 29.23 29.63 29.58 29.32 29.12 29.11 29.28
 29.61 29.33 29.72 29.55 29.18 29.14 29.32 29.23 29.55 29.46 30.18 29.58
 29.05 29.7  29.81 29.75 29.99 30.23 29.75 29.79 29.6  30.06 29.84 29.64
 30.3  30.13 29.82 29.81 29.81 29.72 29.9  31.63 31.42]
```

Splitting the dataset into the Training set and Test set

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_
```

Training the Linear Regression model on the Training set

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

Predicting the Test set results

```
y_pred = regressor.predict(X_test)
# np.set_printoptions(precision=2)
# print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.r

# print(y_test.reshape())

y_test = np.array(y_test)
y_pred = np.array(y_pred)

print(y_test)
```

```
[28.62 29.31 29.58 29.23 28.83 29.72 28.59 29.81 28.74 30.18 30.23 28.47
 29.09 28.67 31.42 29.04 29.46 28.89 28.89 29.26 28.11 30.3  28.66 28.89]
```

```
print(y_pred)
```

```
[28.61298268 29.35069218 29.67648711 29.21980812 28.80814616 29.74217943
 28.616065   29.79996582 28.71994749 30.18927669 30.26622397 28.62326581
 29.11904085 28.70726098 31.4525398  29.04149196 29.43159102 28.88563393
 28.84233802 29.23703529 28.11811715 30.34928586 28.63771382 28.86496196]
```

```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, y_pred)
print(mse)
```

```
0.002028984063429594
```

```
import math
rmse = math.sqrt(mse)
print(rmse)
```

```
0.045044245619497214
```

Name: ***Aditya Kangune***

Roll number: ***33323***

Batch: ***K11***

Multiple Linear Regression

Importing the libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

Importing the dataset

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

```
dataset = pd.read_csv('/content/drive/MyDrive/Datasets/temperat
print(dataset)
```

```
     YEAR    JAN    FEB    MAR  ...  JAN-FEB  MAR-MAY  JUN-SEP  OCT-DEC
0    1901  22.40  24.14  29.07  ...    23.27    31.46    31.27    27.25
1    1902  24.93  26.58  29.77  ...    25.75    31.76    31.09    26.49
2    1903  23.44  25.03  27.83  ...    24.24    30.71    30.92    26.26
3    1904  22.50  24.73  28.21  ...    23.62    30.95    30.66    26.40
4    1905  22.00  22.83  26.68  ...    22.25    30.00    31.33    26.57
..    ...    ...    ...    ...  ...      ...      ...      ...      ...
112  2013  24.56  26.59  30.62  ...    25.58    32.58    31.33    27.83
113  2014  23.83  25.97  28.95  ...    24.90    31.82    32.00    27.81
114  2015  24.58  26.89  29.07  ...    25.74    31.68    31.87    28.27
115  2016  26.94  29.72  32.62  ...    28.33    34.57    32.28    30.03
116  2017  26.45  29.46  31.60  ...    27.95    34.13    32.41    29.69

[117 rows x 18 columns]
```

```
dataset.head()
```

	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DE
0	1901	22.40	24.14	29.07	31.91	33.41	33.18	31.21	30.39	30.47	29.97	27.31	24.4
1	1902	24.93	26.58	29.77	31.78	33.73	32.91	30.92	30.73	29.80	29.12	26.31	24.(
2	1903	23.44	25.03	27.83	31.39	32.91	33.00	31.34	29.98	29.85	29.04	26.08	23.(
3	1904	22.50	24.73	28.21	32.02	32.64	32.07	30.36	30.09	30.04	29.20	26.36	23.(
4	1905	22.00	22.83	26.68	30.01	33.32	33.25	31.44	30.68	30.12	30.67	27.52	23.8

```
# X = dataset.iloc[:, :-1].values
# y = dataset.iloc[:, -1].values
```

```
dataset.dtypes
```

```
YEAR         int64
JAN        float64
FEB        float64
MAR        float64
APR        float64
MAY        float64
JUN        float64
JUL        float64
AUG        float64
SEP        float64
OCT        float64
NOV        float64
DEC        float64
ANNUAL     float64
JAN-FEB    float64
MAR-MAY    float64
JUN-SEP    float64
OCT-DEC    float64
dtype: object
```

```
dataset.columns
```

```
Index(['YEAR', 'JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL', 'AUG', 'SEP',
       'OCT', 'NOV', 'DEC', 'ANNUAL', 'JAN-FEB', 'MAR-MAY', 'JUN-SEP',
       'OCT-DEC'],
      dtype='object')
```

```
dataset.describe()
```

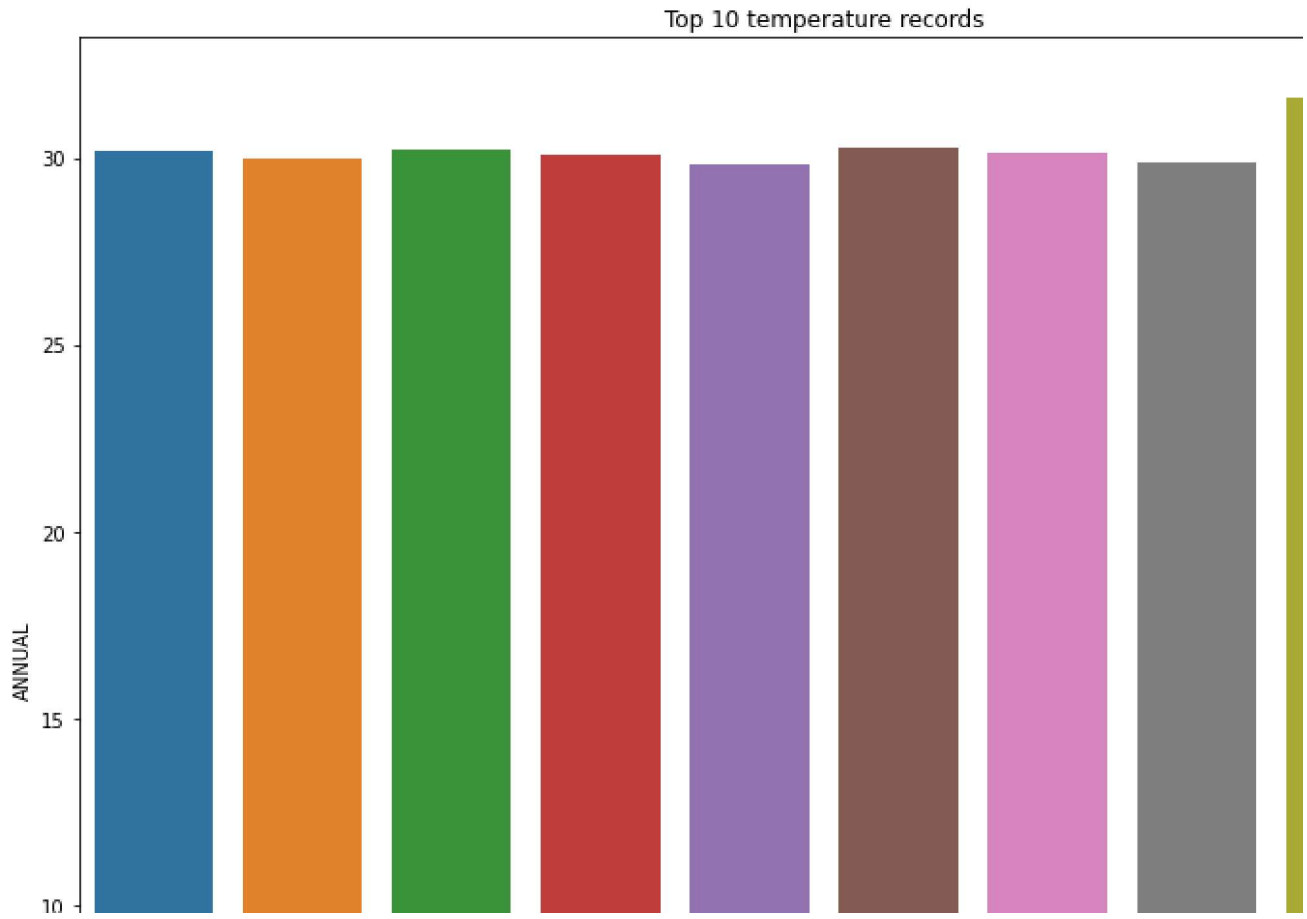
	YEAR	JAN	FEB	MAR	APR	MAY	J
count	117.000000	117.000000	117.000000	117.000000	117.000000	117.000000	117.0000
mean	1959.000000	23.687436	25.597863	29.085983	31.975812	33.565299	32.7742
std	33.919021	0.834588	1.150757	1.068451	0.889478	0.724905	0.6331
min	1901.000000	22.000000	22.830000	26.680000	30.010000	31.930000	31.1000
25%	1930.000000	23.100000	24.780000	28.370000	31.460000	33.110000	32.3400
50%	1959.000000	23.680000	25.480000	29.040000	31.950000	33.510000	32.7300
75%	1988.000000	24.180000	26.310000	29.610000	32.420000	34.030000	33.1800

```
dataset.isnull().sum()
```

```
YEAR        0
JAN         0
FEB         0
MAR         0
APR         0
MAY         0
JUN         0
JUL         0
AUG         0
SEP         0
OCT         0
NOV         0
DEC         0
ANNUAL      0
JAN-FEB     0
MAR-MAY     0
JUN-SEP     0
OCT-DEC     0
dtype: int64
```

```
top_10_data = dataset.nlargest(10, "ANNUAL")
plt.figure(figsize=(14,12))
plt.title("Top 10 temperature records")
sns.barplot(x=top_10_data.YEAR, y=top_10_data.ANNUAL)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2bf41ec410>
```



Taking in the required month for prediction

```
month_input = input("Enter month for prediction (Eg:'JAN'):")
```

```
Enter month for prediction (Eg:'JAN'):APR
```

```
X = dataset[month_input].values
y = dataset["ANNUAL"].values

# X = np.array(X, ndmin=2)
# y = np.array(y, ndmin=2)

print(X)
```

```
[31.91 31.78 31.39 32.02 30.01 31.93 31.79 32.42 30.79 31.42 31.27 31.29
 32.02 30.96 31.36 31.99 30.61 30.68 31.55 30.89 32.47 31.85 32.37 32.07
 32.53 30.42 31.5  31.7  31.59 30.98 32.65 32.08 30.53 31.95 30.24 31.52
 30.96 32.33 31.03 31.38 32.8  31.98 30.75 30.93 30.89 32.19 32.28 31.93
 31.85 31.16 30.3  32.22 32.1  32.47 31.12 32.04 30.91 32.51 32.33 31.9
 31.8  31.76 30.99 32.12 30.6  31.95 31.7  31.59 32.4  32.98 32.19 31.46
 33.3  32.68 32.36 31.71 31.7  32.23 33.11 33.36 32.26 31.65 31.1  32.19
 32.72 32.11 32.18 32.15 31.76 31.59 31.51 31.89 32.33 31.57 32.83 32.4
```

```
       31.   32.6  33.77 33.17 32.54 33.51 32.91 32.97 32.37 32.59 33.57 32.13
       33.09 34.07 31.7  32.46 32.66 32.74 31.87 35.38 34.95]
```

```
print(y)
```

```
[28.96 29.22 28.47 28.49 28.3  28.73 28.65 28.83 28.38 28.53 28.62 28.95
 28.67 28.66 28.94 28.82 28.11 28.66 28.66 28.76 28.86 28.8  28.74 28.8
 28.67 28.7  28.59 28.98 28.76 28.65 29.15 29.09 28.49 29.03 28.76 28.71
 28.7  28.7  28.85 28.88 29.46 28.98 28.8  28.89 28.97 29.37 28.84 28.73
 28.89 28.47 29.09 29.16 29.43 28.92 28.76 28.63 28.64 29.34 29.02 29.31
 28.72 28.89 29.04 29.09 29.16 29.41 29.14 29.07 29.61 29.47 29.15 29.31
 29.44 29.26 28.89 29.27 29.41 29.23 29.63 29.58 29.32 29.12 29.11 29.28
 29.61 29.33 29.72 29.55 29.18 29.14 29.32 29.23 29.55 29.46 30.18 29.58
 29.05 29.7  29.81 29.75 29.99 30.23 29.75 29.79 29.6  30.06 29.84 29.64
 30.3  30.13 29.82 29.81 29.81 29.72 29.9  31.63 31.42]
```

Splitting the dataset into the Training set and Test set

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_
```

Training the Linear Regression model on the Training set

```
X_train= X_train.reshape(-1, 1)
y_train= y_train.reshape(-1, 1)
X_test = X_test.reshape(-1, 1)

from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

Predicting the Test set results

```
y_pred = regressor.predict(X_test)
# np.set_printoptions(precision=2)
# print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.r

# print(y_test.reshape())

print(y_test)
```

```
[28.62 29.31 29.58 29.23 28.83 29.72 28.59 29.81 28.74 30.18 30.23 28.47
 29.09 28.67 31.42 29.04 29.46 28.89 28.89 29.26 28.11 30.3  28.66 28.89]
```

```
print(y_pred)
```

```
[[28.85675571]
 [29.13776779]
 [29.36079324]
 [29.13330728]
 [29.36971426]
 [29.26266204]
 [28.95934742]
 [29.47676648]
 [29.34741171]
 [29.55259513]
 [29.85590975]
 [28.91028182]
 [28.42408633]
 [29.41877986]
 [30.49822306]
 [28.73186146]
 [28.99057099]
 [29.3429512 ]
 [29.07532066]
 [29.48568749]
 [28.56236211]
 [29.66856837]
 [28.71847993]
 [28.70509841]]
```

```
y_test = np.array(y_test)
y_pred = np.array(y_pred)
```

Mean Squared Error

```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, y_pred)
print(mse)
```

```
0.21199560831993922
```

Root Mean Squared Error

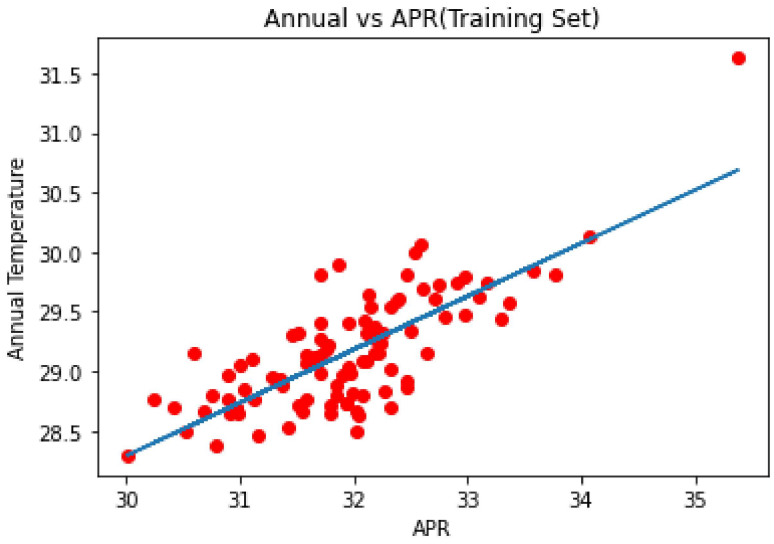
```
import math
rmse = math.sqrt(mse)
print(rmse)
```

```
0.4604298082443612
```

Visualising the Training set results

```
plt.title("Annual vs {month}(Training Set)".format(month=month_
plt.ylabel("Annual Temperature")
plt.xlabel("{month}".format(month=month_input))
plt.scatter(X_train, y_train, color="red")
plt.plot(X_train, regressor.predict(X_train))
# show is used to display the graphic in the output
plt.show()

# Red dots are real values of salary (x_train, y_train)
# Blue line is the best fit line on training X_train and predic
```

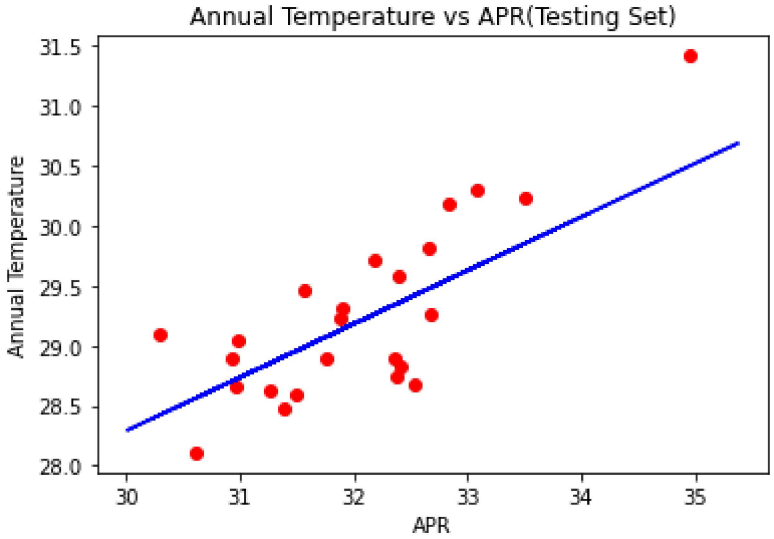


Visualising the Test set results

```
plt.title("Annual Temperature vs {month}(Testing Set)".format(m
plt.ylabel("Annual Temperature")
plt.xlabel("{month}".format(month=month_input))
plt.scatter(X_test, y_test, color="red")

# Predicted salries of test set will on the same rgression line
plt.plot(X_train, regressor.predict(X_train), color="blue")
plt.show()
```

```
# Red are new observations(test set)
# Blue is our best fit line after traning on the available trai
```




```
print(regressor.coef_)
print(regressor.intercept_)
```

```
[[0.44605091]]
[14.90874382]
```

Therefore, the equation of our simple linear regression model is:

$$\text{Annual\_temperature} = 0.44605091 \times \text{month} + 14.90874382.$$

Important Note: To get these coefficients we called the "coef_" and "intercept_" attributes from our regressor object. Attributes in Python are different than methods and usually return a simple value or an array of values.

PUNE INSTITUTE OF COMPUTER TECHNOLOGY

Subject: Machine Learning (LP-1 LAB)

Name: Aditya Kangune **Roll No. : 33323**

Batch: K11 **Academic Year: 2021-22**

Assignment 3
Classification using Machine Learning

Problem Statement:

Perform the following operations on the given dataset:

A. Apply Data pre-processing (Label Encoding, Data Transformation…) techniques if necessary.
B. Perform data-preparation (Train-Test Split)
C. Apply Decision tree classification Algorithm
D. Evaluate Model.

Objective:

This assignment will help the students to realize how the decision tree classifier can be used and predictions using the same can be performed.

Theory:

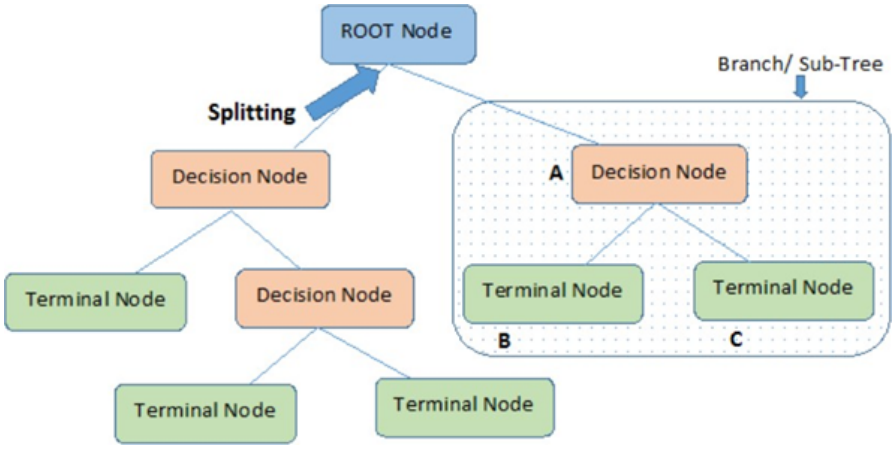
Classification:

1. Classification is the process of categorizing a given set of data into classes.
2. It can be performed on both structured or unstructured data.
3. The process starts with predicting the class of given data points.
4. The classes are often referred to as target, label or categories.

What is a Decision Tree?

It uses a flowchart like a tree structure to show the predictions that result from a series of feature-based splits.

It starts with a root node and ends with a decision made by leaves.



Root Nodes - It is the node present at the beginning of a decision tree. from this node, the population starts dividing according to various features.

Decision Nodes – the nodes we get after splitting the root nodes are called Decision Node

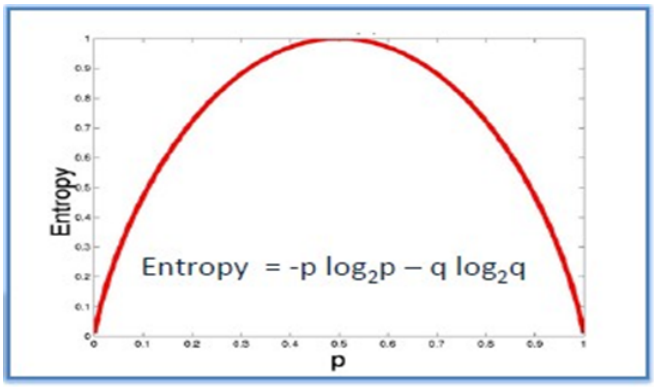
Leaf Nodes – the nodes where further splitting is not possible are called leaf nodes or terminal nodes

Sub-tree – just like a small portion of a graph is called sub-graph similarly a sub-section of this decision tree is called sub-tree.

Pruning – It is cutting down some nodes to stop overfitting.

Entropy:

Entropy is used to calculate the homogeneity of a sample. If the sample is completely homogeneous the entropy is zero and if the sample is equally divided it has an entropy of one.



Entropy = $-0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1$

a) Entropy using the frequency table of one attribute:

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

Play Golf	
Yes	No
9	5

Entropy(PlayGolf) = Entropy (5,9)
= Entropy (0.36, 0.64)
= - (0.36 $\log_2$ 0.36) - (0.64 $\log_2$ 0.64)
= 0.94

b) Entropy using the frequency table of two attributes:

$$E(T, X) = \sum_{c \in X} P(c) E(c)$$

		Play Golf		
		Yes	No	
Outlook	Sunny	3	2	5
	Overcast	4	0	4
	Rainy	2	3	5
				14

E(PlayGolf, Outlook) = **P**(Sunny)***E**(3,2) + **P**(Overcast)***E**(4,0) + **P**(Rainy)***E**(2,3)
= (5/14)*0.971 + (4/14)*0.0 + (5/14)*0.971
= 0.693

Information Gain

The information gain is based on the decrease in entropy after a dataset is split on an attribute.

Constructing a decision tree is all about finding attributes that return the highest information gain (i.e., the most homogeneous branches).

Step 1: Calculate the entropy of the target.

Entropy(PlayGolf) = Entropy (5,9)

= Entropy (0.36, 0.64)

= - (0.36 $\log_2$ 0.36) - (0.64 $\log_2$ 0.64)

= 0.94

Step 2: The dataset is then split into different attributes. The entropy for each branch is calculated.

Then it is added proportionally, to get total entropy for the split. The resulting entropy is subtracted from the entropy before the split.

The result is the Information Gain or decrease in entropy.

		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3
Gain = 0.247			

		Play Golf	
		Yes	No
Temp.	Hot	2	2
	Mild	4	2
	Cool	3	1
Gain = 0.029			

		Play Golf	
		Yes	No
Humidity	High	3	4
	Normal	6	1
Gain = 0.152			

		Play Golf	
		Yes	No
Windy	False	6	2
	True	3	3
Gain = 0.048			

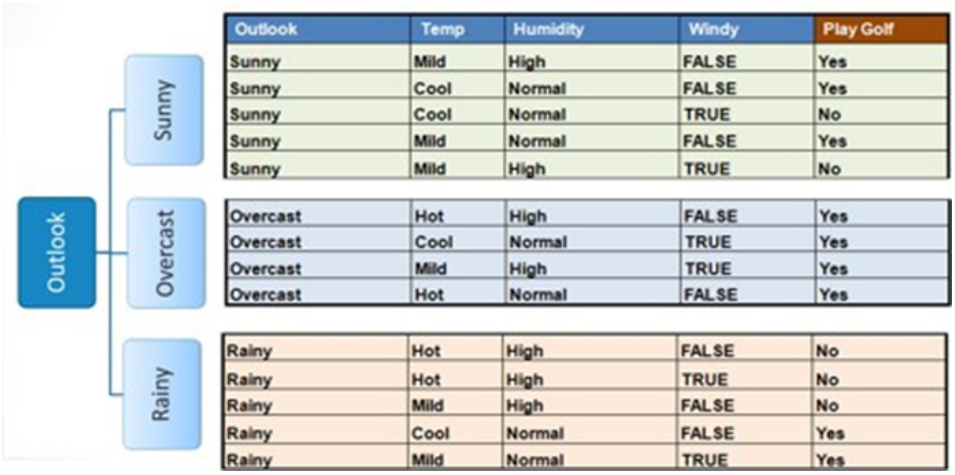
$$Gain(T, X) = Entropy(T) - Entropy(T, X)$$

G(PlayGolf, Outlook) = **E**(PlayGolf) – **E**(PlayGolf, Outlook)

= 0.940 – 0.693 = 0.247

Step 3: Choose the attribute with the largest information gain as the decision node, divide the dataset by its branches and repeat the same process on every branch.

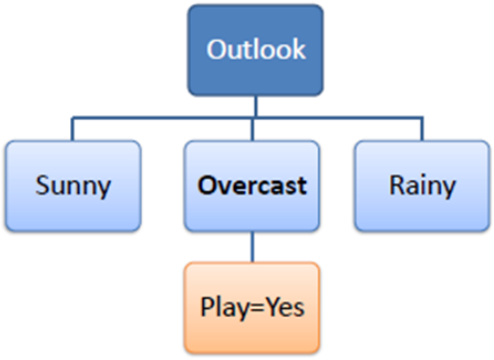
		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3
Gain = 0.247			



Outlook	Temp	Humidity	Windy	Play Golf
Sunny	Mild	High	FALSE	Yes
Sunny	Cool	Normal	FALSE	Yes
Sunny	Cool	Normal	TRUE	No
Sunny	Mild	Normal	FALSE	Yes
Sunny	Mild	High	TRUE	No
Overcast	Hot	High	FALSE	Yes
Overcast	Cool	Normal	TRUE	Yes
Overcast	Mild	High	TRUE	Yes
Overcast	Hot	Normal	FALSE	Yes
Rainy	Hot	High	FALSE	No
Rainy	Hot	High	TRUE	No
Rainy	Mild	High	FALSE	No
Rainy	Cool	Normal	FALSE	Yes
Rainy	Mild	Normal	TRUE	Yes

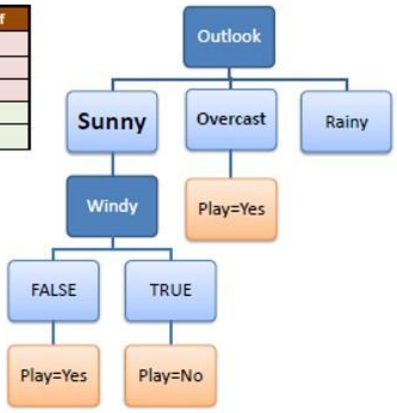
Step 4a: A branch with the entropy of 0 is a leaf node.

Temp	Humidity	Windy	Play Golf
Hot	High	FALSE	Yes
Cool	Normal	TRUE	Yes
Mild	High	TRUE	Yes
Hot	Normal	FALSE	Yes



Step 4b: A branch with an entropy of more than 0 needs further splitting.

Temp	Humidity	Windy	Play Golf
Mild	High	FALSE	Yes
Cool	Normal	FALSE	Yes
Mild	Normal	FALSE	Yes
Cool	Normal	TRUE	No
Mild	High	TRUE	No



Step 5: The ID3 algorithm is run recursively on the non-leaf branches until all data is classified.

Decision Tree to Decision Rules:

A decision tree can easily be transformed into a set of rules by mapping from the root node to the leaf nodes one by one.

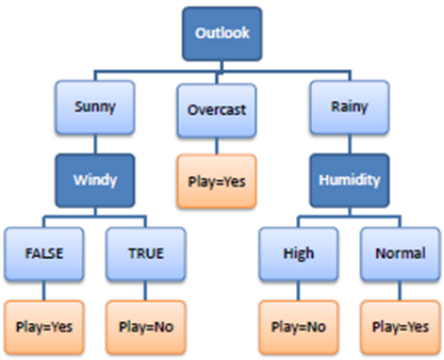
R_1: IF (Outlook=Sunny) AND (Windy=FALSE) THEN Play=Yes

R_2: IF (Outlook=Sunny) AND (Windy=TRUE) THEN Play=No

R_3: IF (Outlook=Overcast) THEN Play=Yes

R_4: IF (Outlook=Rainy) AND (Humidity=High) THEN Play=No

R_5: IF (Outlook=Rain) AND (Humidity=Normal) THEN Play=Yes



Pruning:

It is another method that can help us avoid overfitting. It helps in improving the performance of the tree by cutting the nodes or sub-nodes which are not significant. It removes the branches which have very low importance.

There are mainly 2 ways for pruning:

(i) **Pre-pruning** – we can stop growing the tree earlier, which means we can prune/remove/cut a node if it has low importance **while growing** the tree.

(ii)**Post-pruning** – once our **tree is built to its depth**, we can start pruning the nodes based on their significance.

Application:

Helpful in solving classification problems.

Conclusion:

Implemented classification algorithm on the given dataset and understood decision tree classifiers and how they are used for prediction.

Implementation:

```
#Data handling libraries
import numpy as np
import pandas as pd

#Data Visualization libraries
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px


#Data Preprocessing libraries
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split


df = pd.read_csv("/content/drive/MyDrive/Datasets/Admission_Pre
print(df)
```

```
     Serial No.  GRE Score  TOEFL Score  ...  CGPA  Research  Chance of Admit
0             1        337          118  ...  9.65         1              0.92
1             2        324          107  ...  8.87         1              0.76
2             3        316          104  ...  8.00         1              0.72
3             4        322          110  ...  8.67         1              0.80
4             5        314          103  ...  8.21         0              0.65
..          ...        ...          ...  ...   ...       ...               ...
395         396        324          110  ...  9.04         1              0.82
396         397        325          107  ...  9.11         1              0.84
397         398        330          116  ...  9.45         1              0.91
398         399        312          103  ...  8.78         0              0.67
399         400        333          117  ...  9.66         1              0.95

[400 rows x 9 columns]
```

```
df.describe()
```

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CG
count	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.0000

```
df.columns
```

```
Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
       'LOR ', 'CGPA', 'Research', 'Chance of Admit '],
      dtype='object')
```

25%	100.750000	308.000000	103.000000	2.000000	2.500000	3.000000	8.1700

```
df.head()
```

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	C
0	1	337	118	4	4.5	4.5	9.65	1	
1	2	324	107	4	4.0	4.5	8.87	1	
2	3	316	104	3	3.0	3.5	8.00	1	
3	4	322	110	3	3.5	2.5	8.67	1	
4	5	314	103	2	2.0	3.0	8.21	0	

```
#Drop the serial no.
df = df.drop(df.columns[0], axis=1)
```

Exploratory Data Analysis

Heatmap

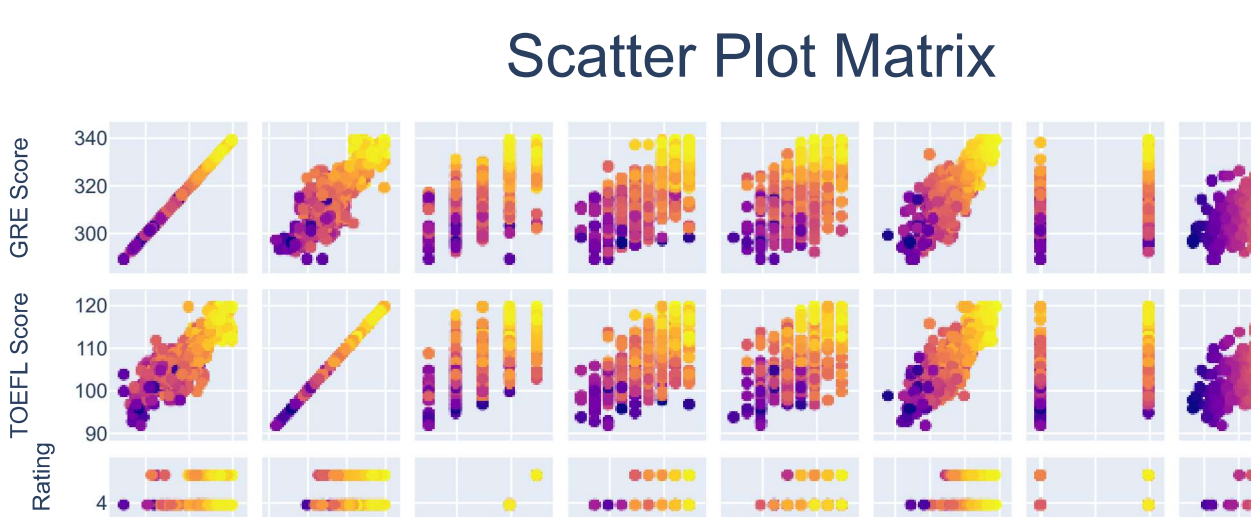
```
heatmp = sns.heatmap(np.corrcoef(df.values.T), annot=True, cbar
plt.show()
```

Scatter Plot Matrix

```
fig = px.scatter_matrix(df,
                        height=800, width=800,
                       color='Chance of Admit ',
                       )

fig.update_layout(font_family='Helvetica', font_size=10,
                  title=dict(text='Scatter Plot Matrix', x=0.5,
                  )

fig.show()
```




```
# s1 = df[df["Chance of Admit "]>0.80]
# s1.count()
```

```
df.loc[df["Chance of Admit "] >= 0.8, "Chance of Admit "] = 1
df.loc[df["Chance of Admit "] < 0.8, "Chance of Admit "] = 0
```

```
df.head()
```

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Adr
0	337	118	4	4.5	4.5	9.65	1	
1	324	107	4	4.0	4.5	8.87	1	
2	316	104	3	3.0	3.5	8.00	1	
3	322	110	3	3.5	2.5	8.67	1	
4	314	103	2	2.0	3.0	8.21	0	

```
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
```

```
X
```

```
array([[337.  , 118.  ,   4.  , ...,   4.5 ,   9.65,   1.  ],
       [324.  , 107.  ,   4.  , ...,   4.5 ,   8.87,   1.  ],
       [316.  , 104.  ,   3.  , ...,   3.5 ,   8.  ,   1.  ],
       ...,
       [330.  , 116.  ,   4.  , ...,   4.5 ,   9.45,   1.  ],
       [312.  , 103.  ,   3.  , ...,   4.  ,   8.78,   0.  ],
       [333.  , 117.  ,   4.  , ...,   4.  ,   9.66,   1.  ]])
```

```
y
```

```
array([1., 0., 0., 1., 0., 1., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
```

```
0., 0., 0., 0., 0., 1., 1., 1., 1., 0., 0., 0., 0., 0., 0., 1., 1.,
1., 1., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 0., 0.,
0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0., 0., 1., 1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1.,
0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 0., 0., 0.,
0., 0., 0., 0., 1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0., 1., 1., 0., 0., 0., 0., 1., 0., 1., 1., 1., 0., 0., 0., 1., 1.,
0., 0., 1., 0., 1., 1., 1., 1., 1., 1., 0., 1., 1., 0., 1., 1., 1.,
0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
0., 1., 1., 1., 1., 1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 1., 1.,
1., 1., 1., 1., 1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1.,
0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 0., 0.,
0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 1., 1., 1., 1.,
0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 1., 0., 0., 0., 0., 1., 1.,
0., 0., 0., 0., 1., 1., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
0., 0., 0., 0., 1., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1.,
0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 0., 0., 0., 0., 0., 0., 0.,
0., 1., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
0., 0., 1., 0., 0., 1., 0., 1., 0., 0., 0., 0., 1., 0., 1., 1., 1.,
0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0., 0., 1., 1., 1., 1., 0., 0., 1., 0., 0., 0., 0., 0., 1., 1., 0.,
0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 1., 1., 0., 0., 0., 0., 0.,
0., 1., 0., 1., 1., 1., 1., 0., 1.])
```

Splitting the dataset into the Training set and Test set

[ ] ↳ 5 cells hidden

Feature Scaling

[ ] ↳ 3 cells hidden

Training the Decision Tree Classification model on the Training set

[ ] ↳ 1 cell hidden

Predicting

```
y_pred = classifier.predict(X_test)
print(y_pred)
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 1. 0. 0.
 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 1. 0. 1. 0. 1. 0. 0. 0. 1. 1. 0. 0. 0.
 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 1. 1. 0. 1. 1. 1. 0. 1.
 1. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0.
 0. 0. 0. 0.]
```

Confusion Matrix

```
from sklearn.metrics import confusion_matrix
matrix = confusion_matrix(y_test, y_pred, labels=[0, 1])
print(matrix)
```

```
[[64  7]
 [ 4 25]]
```

Classification Report

```
from sklearn.metrics import classification_report
cr = classification_report(y_test, y_pred)
print(cr)
```

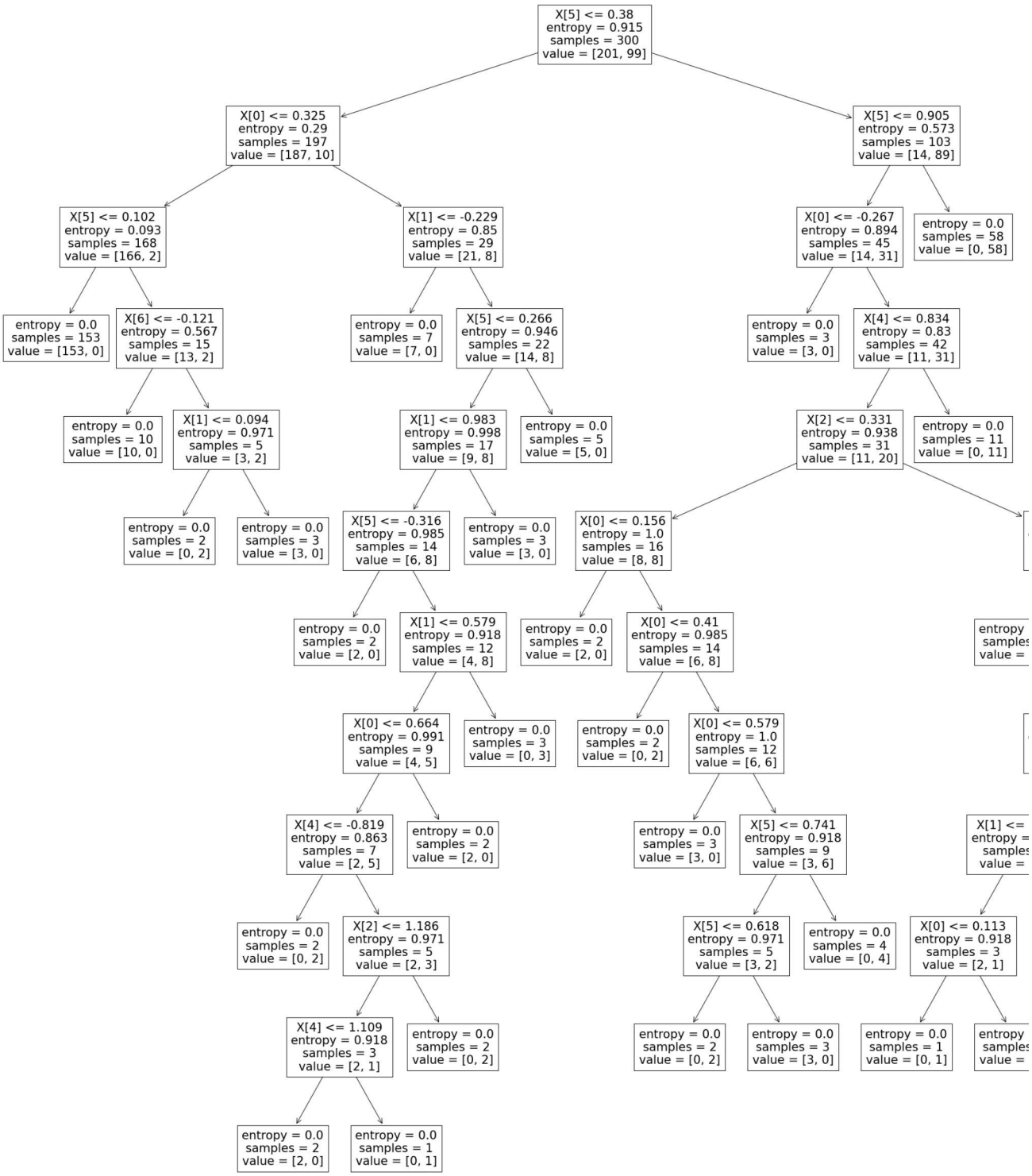
```
              precision    recall  f1-score   support

         0.0       0.94      0.90      0.92        71
         1.0       0.78      0.86      0.82        29

    accuracy                           0.89       100
   macro avg       0.86      0.88      0.87       100
weighted avg       0.89      0.89      0.89       100
```

Decision Tree

```
from sklearn.tree import plot_tree
fig = plt.figure(figsize=(40, 40))
plot_tree(classifier)
plt.show()
```


X[5] <= 0.38
entropy = 0.915
samples = 300
value = [201, 99]
X[0] <= 0.325
entropy = 0.29
samples = 197
value = [187, 10]
X[5] <= 0.905
entropy = 0.573
samples = 103
value = [14, 89]
X[5] <= 0.102
entropy = 0.093
samples = 168
value = [166, 2]
X[1] <= -0.229
entropy = 0.85
samples = 29
value = [21, 8]
X[0] <= -0.267
entropy = 0.894
samples = 45
value = [14, 31]
entropy = 0.0
samples = 58
value = [0, 58]
entropy = 0.0
samples = 153
value = [153, 0]
X[6] <= -0.121
entropy = 0.567
samples = 15
value = [13, 2]
entropy = 0.0
samples = 7
value = [7, 0]
X[5] <= 0.266
entropy = 0.946
samples = 22
value = [14, 8]
entropy = 0.0
samples = 3
value = [3, 0]
X[4] <= 0.834
entropy = 0.83
samples = 42
value = [11, 31]
entropy = 0.0
samples = 10
value = [10, 0]
X[1] <= 0.094
entropy = 0.971
samples = 5
value = [3, 2]
X[1] <= 0.983
entropy = 0.998
samples = 17
value = [9, 8]
entropy = 0.0
samples = 5
value = [5, 0]
X[2] <= 0.331
entropy = 0.938
samples = 31
value = [11, 20]
entropy = 0.0
samples = 11
value = [0, 11]
entropy = 0.0
samples = 2
value = [0, 2]
entropy = 0.0
samples = 3
value = [3, 0]
X[5] <= -0.316
entropy = 0.985
samples = 14
value = [6, 8]
entropy = 0.0
samples = 3
value = [3, 0]
X[0] <= 0.156
entropy = 1.0
samples = 16
value = [8, 8]
entropy = 0.0
samples = 2
value = [2, 0]
X[1] <= 0.579
entropy = 0.918
samples = 12
value = [4, 8]
entropy = 0.0
samples = 2
value = [2, 0]
X[0] <= 0.41
entropy = 0.985
samples = 14
value = [6, 8]
X[0] <= 0.664
entropy = 0.991
samples = 9
value = [4, 5]
entropy = 0.0
samples = 3
value = [0, 3]
entropy = 0.0
samples = 2
value = [0, 2]
X[0] <= 0.579
entropy = 1.0
samples = 12
value = [6, 6]
X[4] <= -0.819
entropy = 0.863
samples = 7
value = [2, 5]
entropy = 0.0
samples = 2
value = [2, 0]
entropy = 0.0
samples = 3
value = [3, 0]
X[5] <= 0.741
entropy = 0.918
samples = 9
value = [3, 6]
entropy = 0.0
samples = 2
value = [0, 2]
X[2] <= 1.186
entropy = 0.971
samples = 5
value = [2, 3]
X[5] <= 0.618
entropy = 0.971
samples = 5
value = [3, 2]
entropy = 0.0
samples = 4
value = [0, 4]
X[0] <= 0.113
entropy = 0.918
samples = 3
value = [2, 1]
X[4] <= 1.109
entropy = 0.918
samples = 3
value = [2, 1]
entropy = 0.0
samples = 2
value = [0, 2]
entropy = 0.0
samples = 2
value = [0, 2]
entropy = 0.0
samples = 3
value = [3, 0]
entropy = 0.0
samples = 1
value = [0, 1]
entropy = 0.0
samples = 2
value = [0, 2]
entropy = 0.0
samples = 2
value = [2, 0]
entropy = 0.0
samples = 1
value = [0, 1]

Results

```
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.res
```

```
[0. 0.]
[1. 0.]
[1. 1.]

[0. 0.]
[0. 0.]
[0. 0.]
[0. 0.]
[0. 0.]
[0. 0.]
[1. 1.]
[0. 0.]
[1. 1.]
[0. 0.]
[0. 0.]
[0. 0.]
[0. 0.]
[0. 0.]
[0. 0.]
[1. 1.]
[1. 0.]
[0. 0.]
[0. 0.]
[0. 0.]
[1. 1.]
[1. 1.]
[0. 1.]
[1. 1.]
[1. 1.]
[1. 0.]
[0. 0.]
[1. 1.]
[1. 0.]
[0. 0.]
[1. 1.]
[0. 0.]
[0. 0.]
[0. 0.]
[1. 1.]
[0. 0.]
[0. 0.]
[0. 0.]
[0. 0.]
[1. 1.]
[0. 0.]
[1. 1.]
[0. 0.]
[0. 0.]
```

```
[0. 1.]
[1. 1.]
[0. 0.]
[0. 0.]
[1. 1.]
[0. 0.]
[0. 0.]
[0. 1.]
[0. 0.]
[0. 1.]
[0. 0.]
[0. 0.]
[0. 0.]]
```

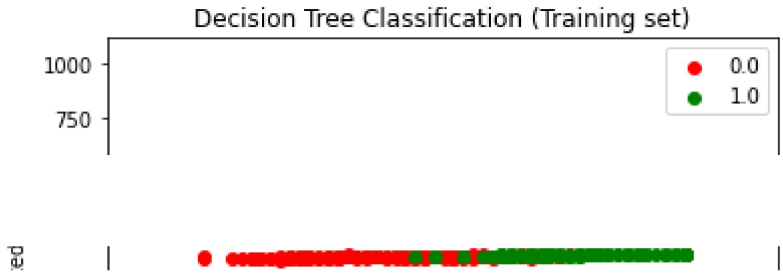
Checking Errors

[ ] ↳ 1 cell hidden

Visualizing

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_train), y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10,
                     np.arange(start = X_set[:, 1].min() - 1000
# plt.contourf(X1, X2, classifier.predict(sc.transform(np.array
             #  alpha = 0.75, cmap = ListedColormap(('red', 'gre
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c =
plt.title('Decision Tree Classification (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

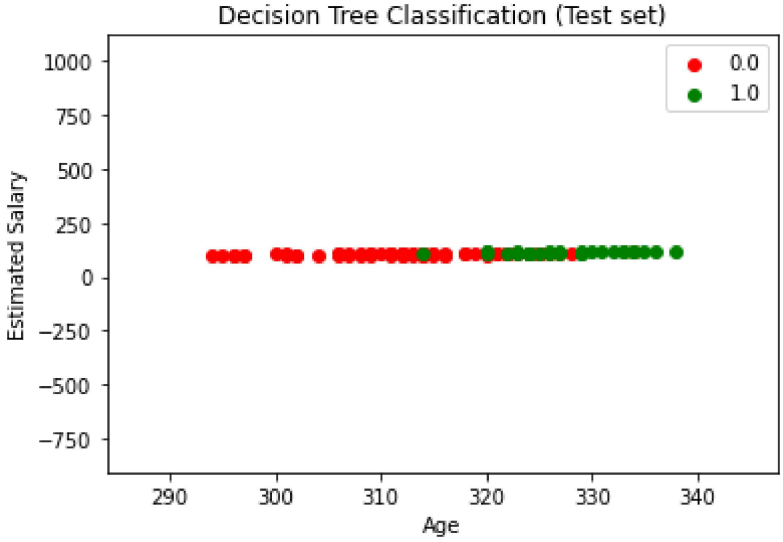
```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoide
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoide
```



Visualising the Test set results

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10,
                     np.arange(start = X_set[:, 1].min() - 1000
# plt.contourf(X1, X2, classifier.predict(sc.transform(np.array
#              alpha = 0.75, cmap = ListedColormap(('red', 'gre
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c =
plt.title('Decision Tree Classification (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoide
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoide
```



PUNE INSTITUTE OF COMPUTER TECHNOLOGY

Subject: Machine Learning (LP-1 LAB)

Name: Aditya Kangune **Roll No. : 33323**

Batch: K11 **Academic Year: 2021-22**

Assignment 4
K Means Clustering

Problem Statement:

Perform the following operations on the given dataset:

A. Apply Data pre-processing (Label Encoding, Data Transformation....) techniques if necessary.
B. Perform data-preparation (Train-Test Split)
C. Apply Machine Learning Algorithm
D. Evaluate Model.
E. Apply Cross-Validation and Evaluate Model

Objective:

This assignment will help the students to realize how to do Clustering using the K Means Clustering algorithm and Hierarchical Clustering Algorithm.

Theory:

KMeans Clustering:

1. K-means clustering is a type of unsupervised learning, which is used when you have unlabeled data (i.e., data without defined categories or groups).
2. The goal of this algorithm is to find groups in the data, with the number of groups represented by the variable K.
3. The algorithm works iteratively to assign each data point to one of the K groups based on the features that are provided. Data points are clustered based on feature similarity.

The **results** of the K-means clustering algorithm are:

1. The centroids of the K clusters, which can be used to label new data.

2. Labels for the training data (each data point is assigned to a single cluster) rather than defining groups before looking at the data, clustering allows you to find and analyze the groups that have formed organically.

The "**Choosing K**" section below describes how the number of groups can be determined. Each centroid of a cluster is a collection of feature values that define the resulting groups.

Examining the centroid feature weights can be used to qualitatively interpret what kind of group each cluster represents.

This introduction to the K-means clustering algorithm covers:

- Common business cases where K-means is used
- The steps involved in running the

  algorithm.

Some examples of use cases are:

- **Behavioral segmentation:**
    - o Segment by purchase history
    - o Segment by activities on application, website, or platform.
    - o Define personas based on interests
    - o Create profiles based on activity monitoring
- **Inventory categorization:**
    - o Group inventory by sales activity
    - o Group inventory by manufacturing metrics
- **Sorting sensor measurements:**
    - o Detect activity types in motion sensors o Group images
    - o Separate audio
    - o Identify groups in health monitoring
- **Detecting bots or anomalies:**
    - o Separate valid activity groups from bots
    - o Group valid activity to clean up outlier detection In addition, monitoring if a tracked data point switches between groups over time can be used to detect meaningful changes in the data.

Algorithm:

The K-means clustering algorithm uses iterative refinement to produce a final result. The algorithm inputs are the number of clusters K and the data set. The data set is a collection of features for each data point.

The algorithms start with initial estimates for the K centroids, which can either be randomly generated or randomly selected from the data set.

The algorithm then iterates between two steps:

1. Data assignment step:

$$\underset{c_i \in C}{\operatorname{argmin}}\ dist(c_i, x)^2$$

1. Each centroid defines one of the clusters.
2. In this step, each data point is assigned to its nearest centroid, based on the squared Euclidean distance.
3. More formally, if ci is the collection of centroids in set C, then each data point x is assigned to a cluster based on where dist(·) is the standard (L2) Euclidean distance
4. . Let the set of data point assignments for each i th cluster centroid be Si.
5.

2. Centroid update step:

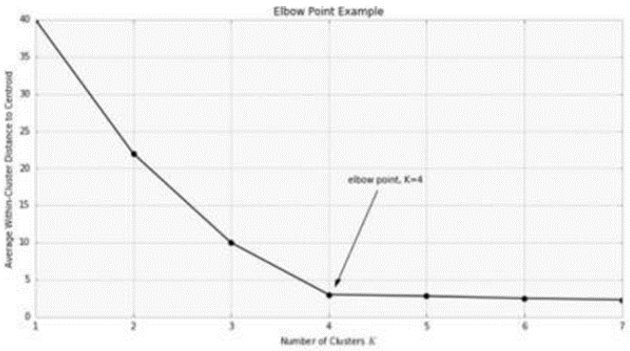
In this step, the centroids are recomputed. This is done by taking the mean of all data points assigned to that centroid's cluster.

$$c_i = \frac{1}{|S_i|}\sum_{x_i \in S_i} x_i$$

1. The algorithm iterates between steps one and two until a stopping criteria is met (i.e., no data points change clusters, the sum of the distances is minimized, or some maximum number of iterations is reached).
2. This algorithm is guaranteed to converge to a result.
3. The result may be a local optimum (i.e. not necessarily the best possible outcome), meaning that assessing more than one run of the algorithm with randomized starting centroids may give a better outcome.

Choosing K:

1. The algorithm described above finds the clusters and data set labels for a particular pre-chosen K.
2. To find the number of clusters in the data, the user needs to run the K-means clustering algorithm for a range of K values and compare the results.
3. In general, there is no method for determining the exact value of K, but an accurate estimate can be obtained using the following techniques.
4. One of the metrics that is commonly used to compare results across different values of K is the mean distance between data points and their cluster centroid.
5. Since increasing the number of clusters will always reduce the distance to data points, increasing K will always decrease this metric, to the extreme of reaching zero when K is the same as the number of data points.
6. Thus, this metric cannot be used as the sole target. Instead, mean distance to the centroid as a function of K is plotted and the "elbow point," where the rate of decrease sharply shifts, can be used to roughly determine K.
7. A number of other techniques exist for validating K, including cross-validation, information criteria, the information-theoretic jump method, the silhouette method, and the G-means algorithm.
8. In addition, monitoring the distribution of data points across groups provides insight into how the algorithm is splitting the data for each K.



Hierarchical Clustering:

Hierarchical clustering analysis is a method of cluster analysis that seeks to build a hierarchy of clusters i.e., tree-type structure based on the hierarchy.

Basically, there are two types of hierarchical cluster analysis strategies –

1. Agglomerative Clustering:

1. Also known as the bottom-up approach or hierarchical agglomerative clustering (HAC).
2. A structure that is more informative than the unstructured set of clusters returned by flat clustering.
3. This clustering algorithm does not require us to prespecify the number of clusters.
4. Bottom-up algorithms treat each data as a singleton cluster at the outset and then successively agglomerates pairs of clusters until all clusters have been merged into a single cluster that contains all data.

Algorithm :

given a dataset (d_1, d_2, d_3, ...d_N) of size N

compute the distance matrix

for i=1 to N:

  # as the distance matrix is symmetric about

  # the primary diagonal so we compute only lower

  # part of the primary diagonal

  for j=1 to i:

    dis_mat[i][j] = distance[d_i, d_j]

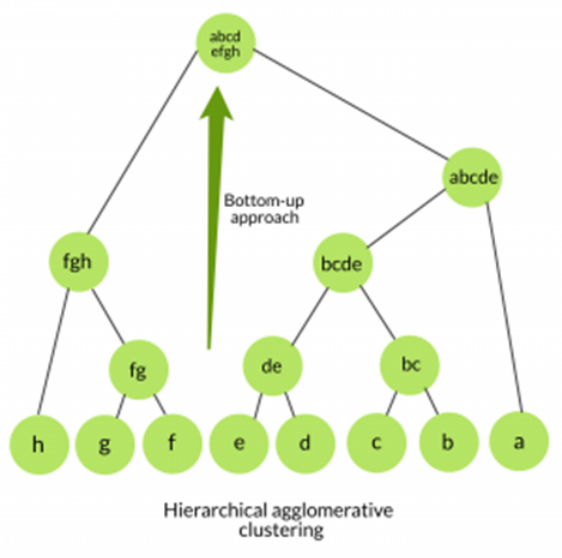
each data point is a singleton cluster

repeat

merge the two clusters having a minimum distance

update the distance matrix

until only a single cluster remains

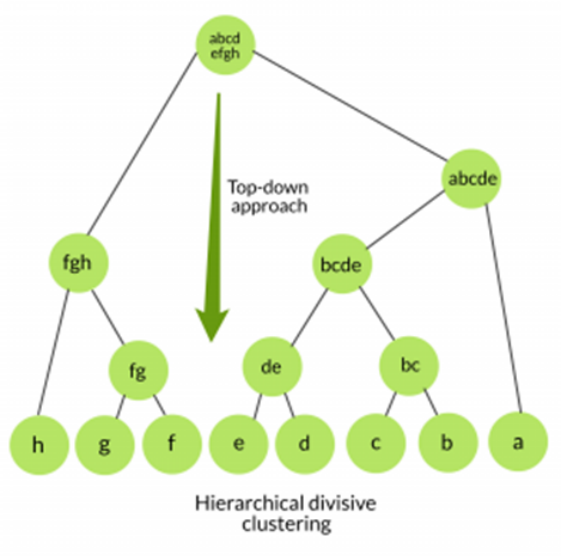


2. Divisive clustering:

1. Also known as a top-down approach.
2. This algorithm also does not require to prespecify the number of clusters.
3. Top-down clustering requires a method for splitting a cluster that contains the whole data and proceeds by splitting clusters recursively until individual data have been split into singleton clusters.

Algorithm:

given a dataset (d_1, d_2, d_3, ...d_N) of size N
at the top, we have all data in one cluster
the cluster is split using a flat clustering method eg. K-Means etc
repeat
choose the best cluster among all the clusters to split
split that cluster by the flat clustering algorithm
until each data is in its own singleton cluster



Conclusion:

Implemented K means and hierarchical clustering algorithm on the given dataset.

Implementation:

Assignment on Clustering Techniques

LP Lab Assignment 4

33323 - Aditya Kangune

Problem Statment

This dataset gives the data of Income and money spent by the customers visiting a Shopping Mall. The data set contains Customer ID, Gender, Age, Annual Income, Spending Score. Therefore, as a mall owner you need to find the group of people who are the profitable customers for the mall owner. Apply at least two clustering algorithms (based on Spending Score) to find the group of customers. A. Apply Data pre-processing (Label Encoding , Data Transformation….) techniques if necessary. B. Perform data-preparation(Train-Test Split) C. Apply Machine Learning Algorithm D. Evaluate Model. E. Apply Cross-Validation and Evaluate Model

K-Means Clustering

Importing the libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

Importing the dataset

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

```
df = pd.read_csv('/content/drive/MyDrive/Datasets/Mall_Customer
```

```
print(df)
```

```
     CustomerID   Genre  Age  Annual Income (k$)  Spending Score (1-100)
0             1    Male   19                  15                      39
1             2    Male   21                  15                      81
2             3  Female   20                  16                       6
3             4  Female   23                  16                      77
4             5  Female   31                  17                      40
..          ...     ...  ...                 ...                     ...
195         196  Female   35                 120                      79
196         197  Female   45                 126                      28
197         198    Male   32                 126                      74
198         199    Male   32                 137                      18
199         200    Male   30                 137                      83

[200 rows x 5 columns]
```

```
df.head()
```

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

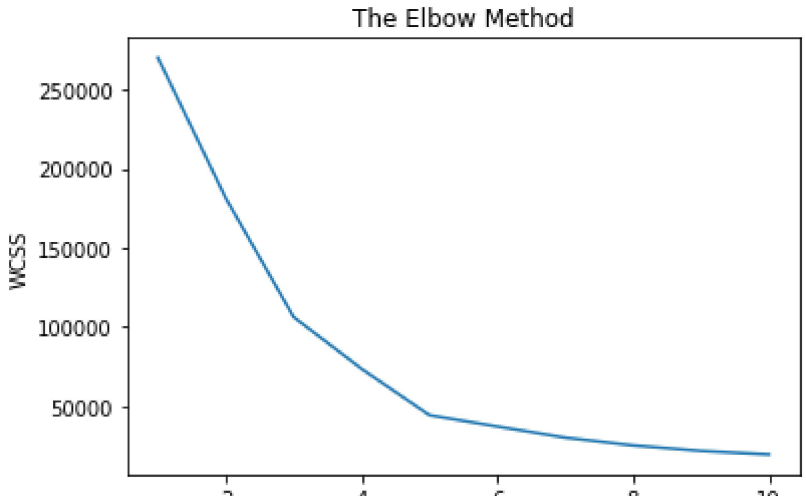
```
X = df.iloc[:, [3, 4]].values
print(X)
```

```
[[ 15  39]
 [ 15  81]
 [ 16   6]
 [ 16  77]
 [ 17  40]
 [ 17  76]
 [ 18   6]
 [ 18  94]
 [ 19   3]
 [ 19  72]
 [ 19  14]
 [ 19  99]
 [ 20  15]
 [ 20  77]
 [ 20  13]
 [ 20  79]
 [ 21  35]
 [ 21  66]
 [ 23  29]
 [ 23  98]
 [ 24  35]
 [ 24  73]
```

```
[ 25   5]
[ 25  73]
[ 28  14]
[ 28  82]
[ 28  32]
[ 28  61]
[ 29  31]
[ 29  87]
[ 30   4]
[ 30  73]
[ 33   4]
[ 33  92]
[ 33  14]
[ 33  81]
[ 34  17]
[ 34  73]
[ 37  26]
[ 37  75]
[ 38  35]
[ 38  92]
[ 39  36]
[ 39  61]
[ 39  28]
[ 39  65]
[ 40  55]
[ 40  47]
[ 40  42]
[ 40  42]
[ 42  52]
[ 42  60]
[ 43  54]
[ 43  60]
[ 43  45]
[ 43  41]
[ 44  50]
```

Using the elbow method to find the optimal number of clusters

```
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

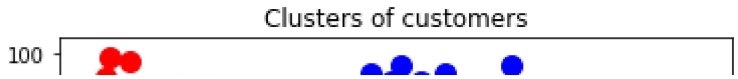


Training the K-Means model on the dataset

```
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_stat
y_kmeans = kmeans.fit_predict(X)
```

Visualising the clusters

```
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100,
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100,
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100,
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100,
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100,
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_cente
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```



Hierarchical Clustering

Hierarchical clustering starts by treating each observation as a separate cluster. Then, it repeatedly executes the following two steps:

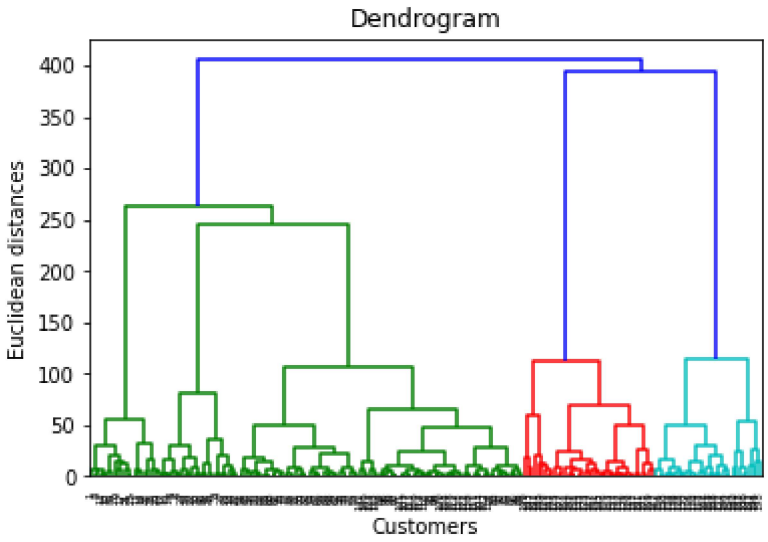
(1) identify the two clusters that are closest together.

(2) merge the two most similar clusters. This iterative process continues until all the clusters are merged together.

Using the dendrogram to find the optimal number of clusters

A Dendrogram is a tree-like diagram that records the sequences of merges or splits.

```
import scipy.cluster.hierarchy as sch
dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))
plt.title('Dendrogram')
plt.xlabel('Customers')
plt.ylabel('Euclidean distances')
plt.show()
```



Training the Hierarchical Clustering model on the dataset

Agglomerative Hierarchical clustering Technique: In this technique, initially each data point is considered as an individual cluster. At each iteration, the similar clusters merge with other clusters until one cluster or K clusters are formed.

Ward's Linkage: The linkage function specifying the distance between two clusters is computed as the increase in the "error sum of squares" (ESS) after fusing two clusters into a single cluster.

```
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters = 5, affinity = 'euclid
y_hc = hc.fit_predict(X)
```

Visualising the clusters

```
plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], s = 100, c = 'red
plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], s = 100, c = 'blu
plt.scatter(X[y_hc == 2, 0], X[y_hc == 2, 1], s = 100, c = 'gre
plt.scatter(X[y_hc == 3, 0], X[y_hc == 3, 1], s = 100, c = 'cya
plt.scatter(X[y_hc == 4, 0], X[y_hc == 4, 1], s = 100, c = 'mag
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```