INDEX

Practical No. 1

Aim: To Import dataset in colab notebook

Output:

Description:

Google Colabatory

Google Colaboratory, known as Colab, is a free Jupyter Notebook environment with many pre-installed libraries like Tensorflow, Pytorch, Keras, OpenCV, and many more. It is one of the cloud services that support GPU and TPU for free. Importing a dataset and training models on the data in the Colab facilitate coding experience.

Why Should We Use Google Colab?

There are several reasons to opt to use Google Colab instead of a plain Jupyter Notebook instance:

- Pre-Installed Libraries
- Saved on the Cloud
- Collaboration
- Free GPU and TPU Use

Import Dataset:

There are two methods by which we can import datset in colab

- Importing data from Google Drive
- Importing data in the local system

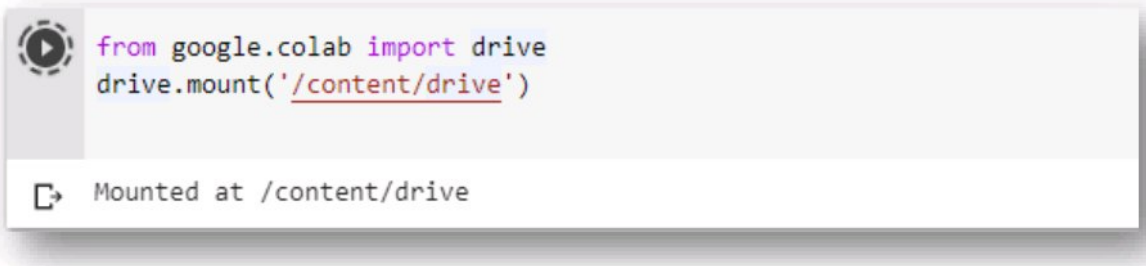
Method 1: Importing data from Google Drive

Mounting Google Drive

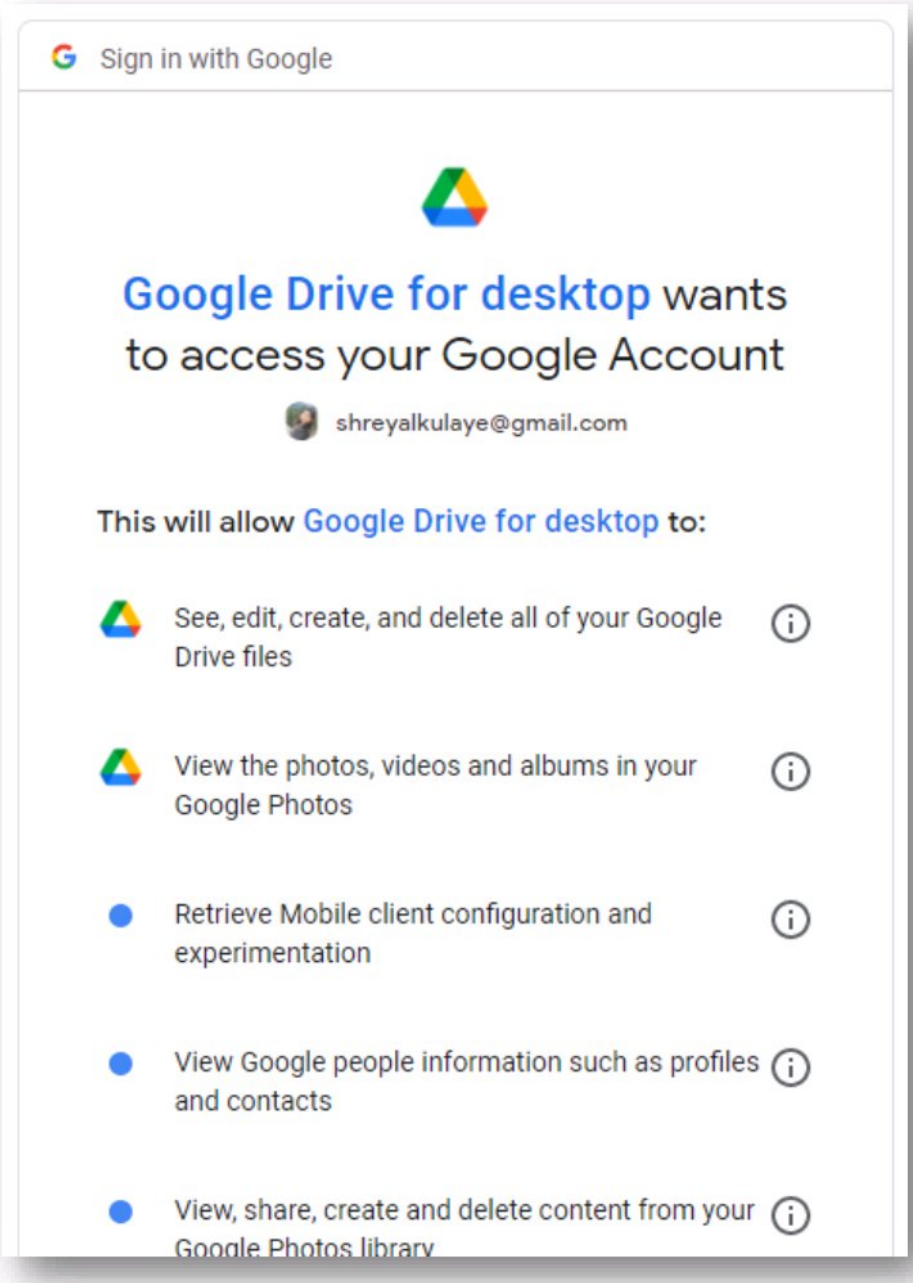
We can access files in drive using mounting Google Drive. Mounting Drive into the Colab meaning that setting up the google drive account as a virtual drive so that we can access the resources of the drive just like a local hard drive.

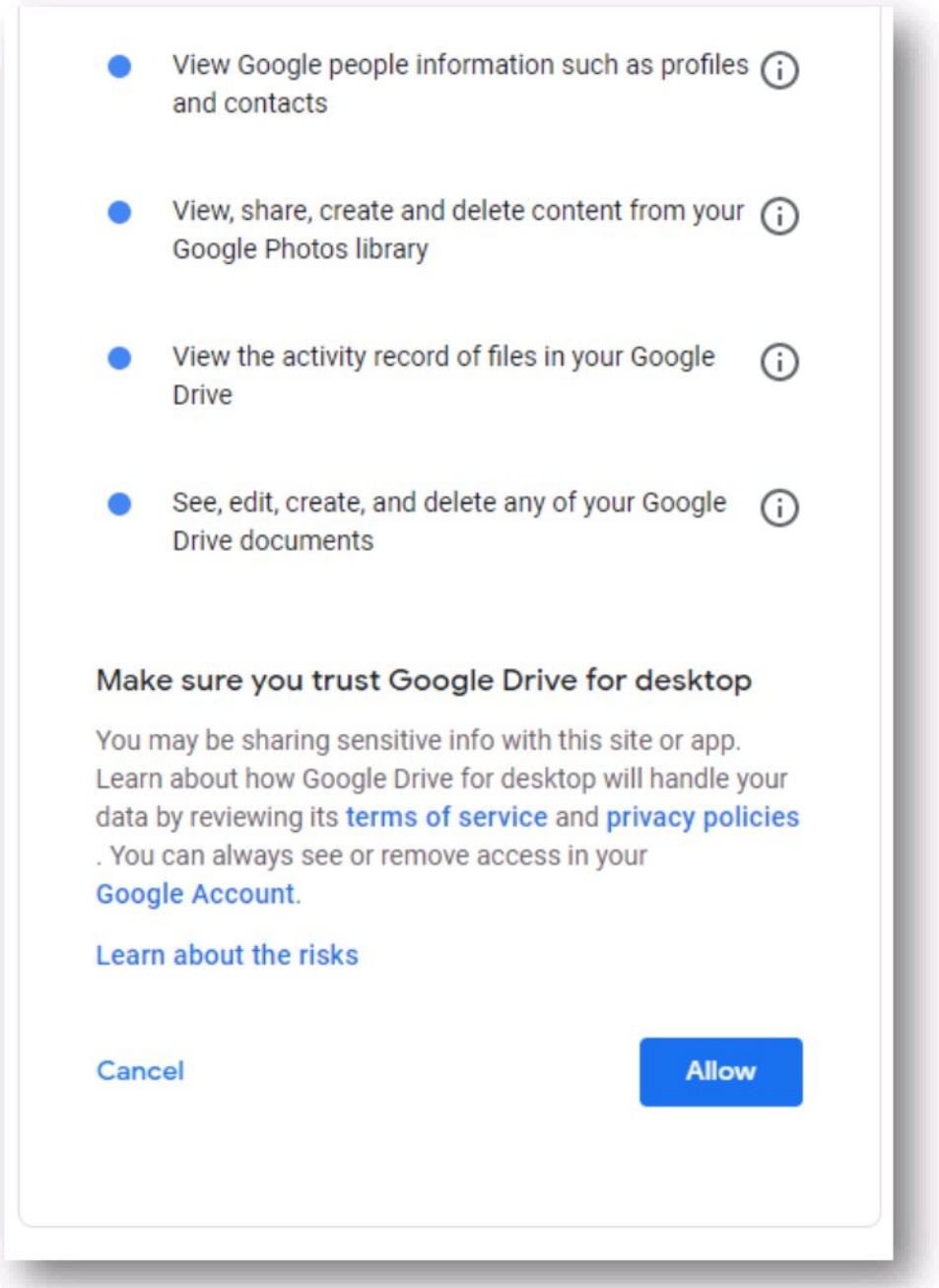
Step 1: To connect Google Drive (GDrive) with Colab, execute the following two lines of code in Colab:

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

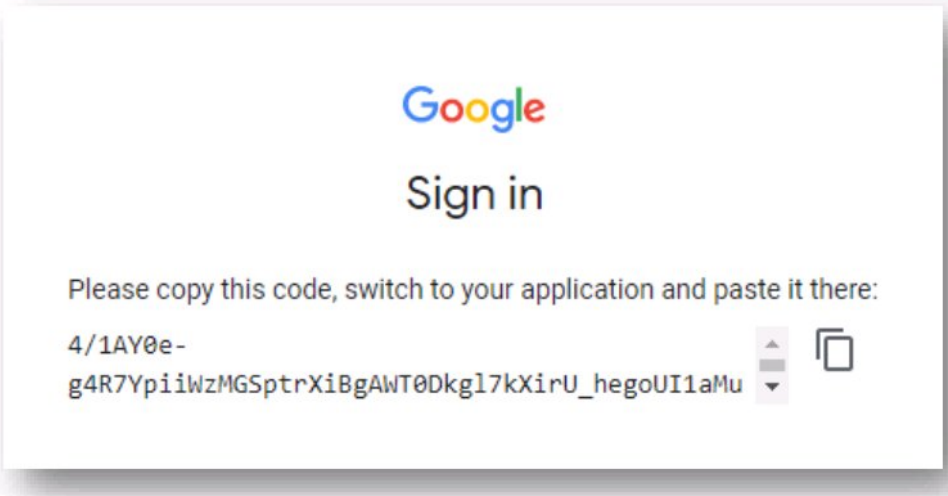


Running the shell will return a URL link and ask for an authorization code:





Step 2: Follow to the mentioned link, sign in Google account, and copy the authorization code:



Step 3: Paste the authorization code in the shell and finally, Google Drive will be mounted at */content/drive*. Note that, files in the drive are under the folder */content/drive/My Drive/*. Now, we can import files in GDrive using a library like Pandas.

Step 4: For instance, we have cancer dataset:

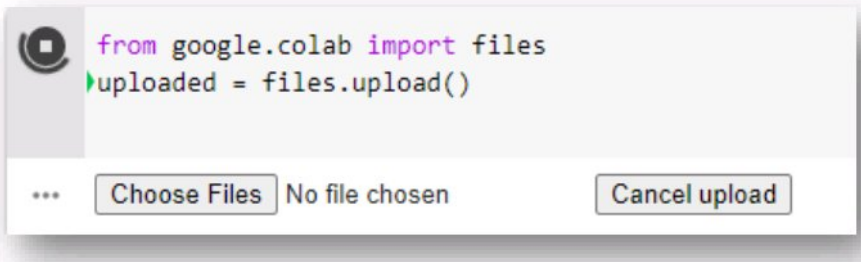
```
[ ]  print(os.listdir("/content/drive/My Drive/cancer/cancer/cancer/train"))
```

```
[ ]  TRAIN_DIR = "/content/drive/My Drive/cancer/cancer/cancer/train/"
     TEST_DIR =  "/content/drive/My Drive/cancer/cancer/cancer/test/"
```

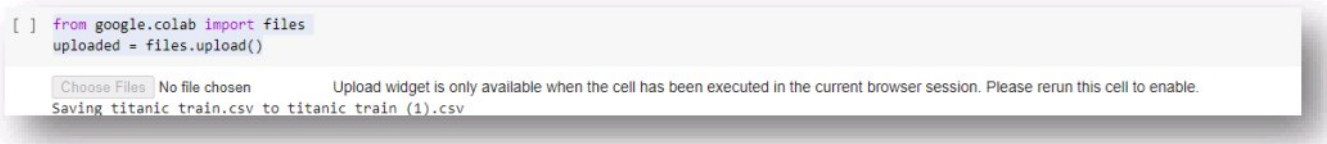
Method 2: Importing data in the local system

Step 1: Run the following two lines of code to import data from the local system.

```
from google.colab import files
uploaded = files.upload()
```



Step 2: Browsing directories in the local system, we can upload data into Colab:



Finally, we can read the data using a library like Pandas:

```
import io
data = pd.read_csv(io.BytesIO(uploaded['titanic_train.csv']))
```

Conclusion: Data is imported successfully in the colab notebook.

Practical No. 2

Aim: To summerize the dataset in colab notebook

Output:

Steps:

Step 1: Import necessary libraries

```
[ ]  import pandas as pd
```

Step 2: Now import data in the notebook

```
# Import data from GitHub (or from your local computer)
df = pd.read_csv("https://raw.githubusercontent.com/kirenz/datasets/master/wage.csv")
```

Step 3: We can calculate summary of a function using describe function in colab notebook

```
# summary statistics for all numerical columns
round(df.describe(),2)
```

	Unnamed: 0	year	age	logwage	wage
count	3000.00	3000.00	3000.00	3000.00	3000.00
mean	218883.37	2005.79	42.41	4.65	111.70
std	145654.07	2.03	11.54	0.35	41.73
min	7373.00	2003.00	18.00	3.00	20.09
25%	85622.25	2004.00	33.75	4.45	85.38
50%	228799.50	2006.00	42.00	4.65	104.92
75%	374759.50	2008.00	51.00	4.86	128.68
max	453870.00	2009.00	80.00	5.76	318.34

```
# summary statistics by groups
df['age'].groupby(df['education']).describe()
```

education	count	mean	std	min	25%	50%	75%	max
1. < HS Grad	268.0	41.794776	12.611111	18.0	33.0	41.5	50.25	75.0
2. HS Grad	971.0	42.217302	12.023480	18.0	33.0	42.0	50.00	80.0
3. Some College	650.0	40.887692	11.523327	18.0	32.0	40.0	49.00	80.0
4. College Grad	685.0	42.773723	10.902406	22.0	34.0	43.0	51.00	76.0
5. Advanced Degree	426.0	45.007042	10.263468	25.0	38.0	44.0	53.00	76.0

Conclusion: Imported dataset is summarized successfully.

Practical No. 3

Aim: To perform Feature Scaling on the dataset

Output:

Description:

Feature Scaling is a technique to standardize the independent features present in the data in a fixed range. It is performed during the data pre-processing to handle highly varying magnitudes or values or units. If feature scaling is not done, then a machine learning algorithm tends to weigh greater values, higher and consider smaller values as the lower values, regardless of the unit of the values.

Example: If an algorithm is not using feature scaling method then it can consider the value 3000 meter to be greater than 5 km but that's actually not true and in this case, the algorithm will give wrong predictions. So, we use Feature Scaling to bring all values to same magnitudes and thus, tackle this issue.

Techniques to perform Feature Scaling

Consider the two most important ones:

- **Min-Max Normalization:** This technique re-scales a feature or observation value with distribution value between 0 and 1.

$$X_{new} = \frac{X_i - \min(X)}{\max(x) - \min(X)}$$

- **Standardization:** It is a very effective technique which re-scales a feature value so that it has distribution with 0 mean value and variance equals to 1.

$$X_{new} = \frac{X_i - X_{mean}}{\text{Standard Deviation}}$$

Steps:

Step 1: Import necessary libraries

```
""" PART 1
    Importing Libraries """

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

# Sklearn library
from sklearn import preprocessing
```

Step 2: Import dataset

```
[ ] """ PART 2
        Importing Data """
    from google.colab import files
    uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving Data_for_Missing_Values.csv to Data_for_Missing_Values (1).csv
```

Step 3: Read dataset using colab

```
[ ] import io
    data = pd.read_csv(io.BytesIO(uploaded['Data_for_Missing_Values.csv']))
```

Step 4: Calculating size iof the data

```
[ ]  data.size

     40
```

Step 5: Calculating size of the data

```
data.isnull()
```

	Country	Age	Salary	Purchased
0	False	False	False	False
1	False	False	False	False
2	False	False	False	False
3	False	False	False	False
4	False	False	False	False
5	False	False	False	False
6	False	False	False	False
7	False	False	False	False
8	False	False	False	False
9	False	False	False	False

```
[ ]  data.isna().sum()

     Country      0
     Age          0
     Salary       0
     Purchased    0
     dtype: int64
```

There is no null data in the dataset.

Step 6: Exploring the dataset

```
[ ]  data.head(11)
```

	Country	Age	Salary	Purchased
0	France	44	72000	No
1	Spain	27	48000	Yes
2	Germany	30	54000	No
3	Spain	38	61000	No
4	Germany	40	1000	Yes
5	France	35	58000	Yes
6	Spain	78	52000	No
7	France	48	79000	Yes
8	Germany	50	83000	No
9	France	37	67000	Yes

Step 7: Slicing the data which is to be scaled from the dataset

```
# here Features - Age and Salary columns
# are taken using slicing
# to handle values with varying magnitude
x = data.iloc[:, 1:3].values
print ("\nOriginal data values : \n",  x)
```

```
Original data values :
 [[   44 72000]
 [   27 48000]
 [   30 54000]
 [   38 61000]
 [   40  1000]
 [   35 58000]
 [   78 52000]
 [   48 79000]
 [   50 83000]
 [   37 67000]]
```

Step 8: Importing the preprocessing libraries fdrom sklearn

```
from sklearn import preprocessing
```

Step 9: Performing **Min-Max Scaler** on the data

```
[ ]  """ MIN MAX SCALER """

     min_max_scaler = preprocessing.MinMaxScaler(feature_range =(0, 1))
```

```
     # Scaled feature
     x_after_min_max_scaler = min_max_scaler.fit_transform(x)

     print ("\nAfter min max Scaling : \n", x_after_min_max_scaler)
```

```
After min max Scaling :
 [[0.33333333 0.86585366]
 [0.         0.57317073]
 [0.05882353 0.64634146]
 [0.21568627 0.73170732]
 [0.25490196 0.        ]
 [0.15686275 0.69512195]
 [1.         0.62195122]
 [0.41176471 0.95121951]
 [0.45098039 1.        ]
 [0.19607843 0.80487805]]
```

Step 10: Performing **Standardization** on the data

```
[ ]  """ Standardisation """

     Standardisation = preprocessing.StandardScaler()
```

```
# Scaled feature
x_after_Standardisation = Standardisation.fit_transform(x)

print ("\nAfter Standardisation : \n", x_after_Standardisation)
```

```
After Standardisation :
 [[ 0.09536935  0.66527061]
 [-1.15176827 -0.43586695]
 [-0.93168516 -0.16058256]
 [-0.34479687  0.16058256]
 [-0.1980748  -2.59226136]
 [-0.56487998  0.02294037]
 [ 2.58964459 -0.25234403]
 [ 0.38881349  0.98643574]
 [ 0.53553557  1.16995867]
 [-0.41815791  0.43586695]]
```

Conclusion: The data is scaled successfully.

Practical No. 4

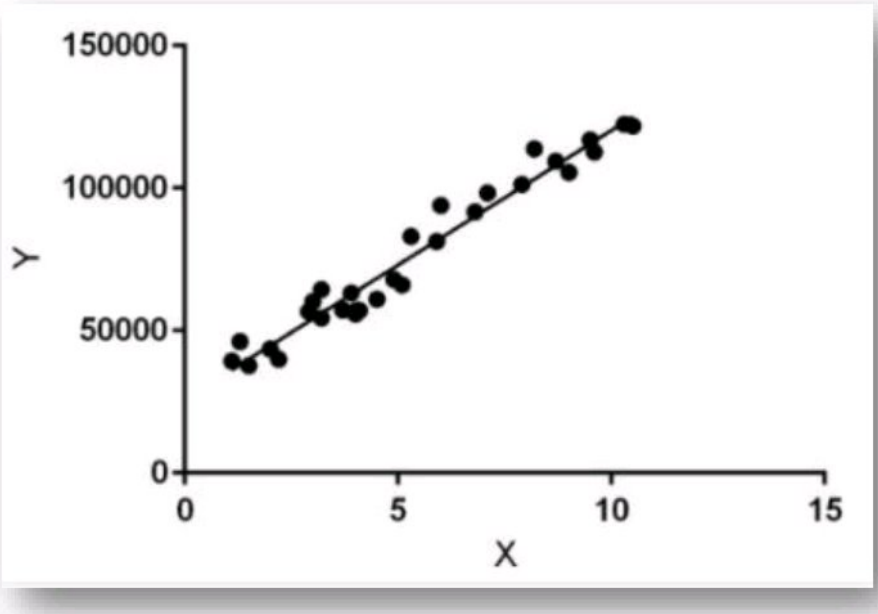
Aim: To perform linear regression using google colab

Output:

Description:

LINEAR REGRESSION:

- **Linear Regression** is a machine learning algorithm based on **supervised learning**.
- It performs a **regression task**. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting.
- Different regression models differ based on – the kind of relationship between dependent and independent variables, they are considering and the number of independent variables being used.



Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(output).

Hence, the name is Linear Regression. In the figure above, X (input) is the work experience and Y (output) is the salary of a person. The regression line is the best fit line for our model.

Steps:

Step 1: Import necessary libraries

```
# Python code to illustrate
# regression using data set
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
import pandas as pd
```

Step 2: Import dataset

```
[ ] from google.colab import files
    uploaded = files.upload()

    Choose Files  No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
    Saving Housing2.xlsx to Housing2.xlsx
```

Step 3: Reading the dataset

```
[ ] import io
    df = pd.read_excel(io.BytesIO(uploaded['Housing2.xlsx']))
```

Step 5: Printing the dataset

```
df
```

	price	tsft
0	100000	450
1	50000	300
2	70000	350
3	800000	800
4	630000	600
5	900000	1000
6	550000	550
7	700000	700
8	880000	950

Step 6: Initializing X and Y variables

```
[ ] Y = df['price']
    X = df['tsft']
```

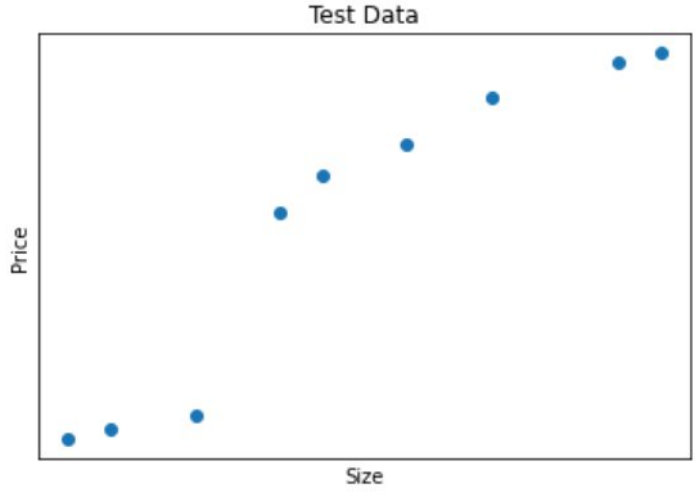
Step 7: Reshaping the values with respec to X and Y variables

```
[ ]  X=X.values.reshape(len(X),1)
     Y=Y.values.reshape(len(Y),1)
```

Step 8: Plotting the graph

```
[ ]  # Plot outputs
     plt.scatter(X, Y)
     plt.title('Test Data')
     plt.xlabel('Size')
     plt.ylabel('Price')
     plt.xticks(())
     plt.yticks(())
```

```
([], <a list of 0 Text major ticklabel objects>)
```



Step 9: Creating the object for the regression

```
# Create linear regression object
regr = linear_model.LinearRegression()
```

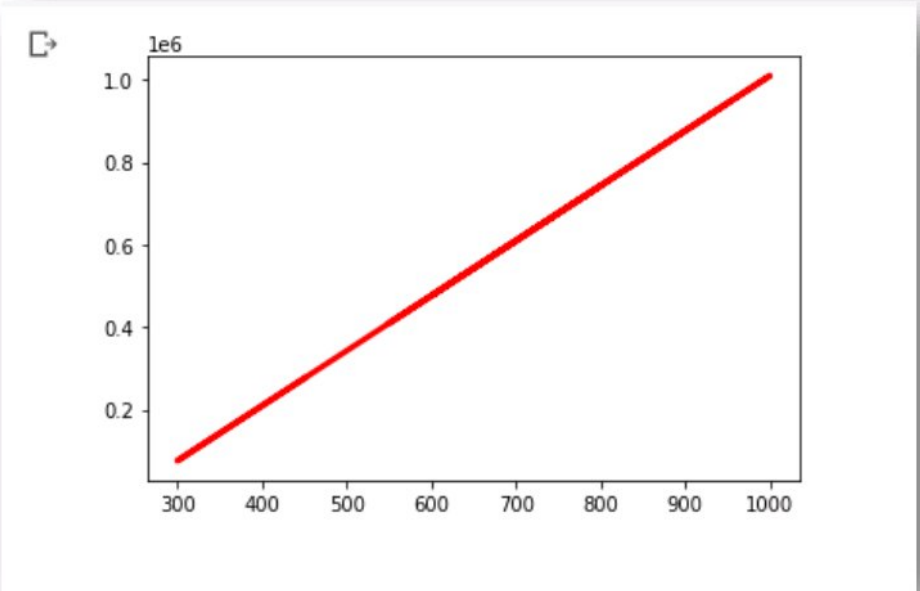
Step 10: Training the dataset

```
# Train the model using the training sets
regr.fit(X, Y)
```

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

Step 11: Plotting the regression line

```
# Plot outputs
plt.plot(X, regr.predict(X), color='red',linewidth=3)
plt.show()
```



Conclusion: Linear Regression model is implemented successfully.

Practical No. 5

Aim: To perform Multiple Linear Regression using google colab

Output:

Description:

MULTIPLE LINEAR REGRESSION

Multiple Linear Regression attempts to model the relationship between two or more features and a response by fitting a linear equation to observed data. The steps to perform multiple linear Regression are almost similar to that of simple linear Regression. The Difference Lies in the evaluation. We can use it to find out which factor has the highest impact on the predicted output and now different variable relate to each other.

Assumption of Regression Model :

- **Linearity:** The relationship between dependent and independent variables should be linear.
- **Homoscedasticity:** Constant variance of the errors should be maintained.
- **Multivariate normality:** Multiple Regression assumes that the residuals are normally distributed.
- **Lack of Multicollinearity:** It is assumed that there is little or no multicollinearity in the data.

Steps:

Step 1: Importing necessary libraries

```
import pandas as pd
import numpy as np
```

Step 2: Importing dataset

```
[3] from google.colab import files
    uploaded = files.upload()
```

Choose Files Folds5x2_pp.xlsx

- **Folds5x2_pp.xlsx**(application/vnd.openxmlformats-officedocument.spreadsheetml.sheet) - 2001356 bytes, last modified: 4/3/2021 - 100% done

```
Saving Folds5x2_pp.xlsx to Folds5x2_pp.xlsx
```

Step 3: Reading dataset

```
[4] import io
    df = pd.read_excel(io.BytesIO(uploaded['Folds5x2_pp.xlsx']))
```

Step 4: Printing data

```
df
```

	AT	V	AP	RH	PE
0	14.96	41.76	1024.07	73.17	463.26
1	25.18	62.96	1020.04	59.08	444.37
2	5.11	39.40	1012.16	92.14	488.56
3	20.86	57.32	1010.24	76.64	446.48
4	10.82	37.50	1009.23	96.62	473.90
...	...	...	...	...	...
9563	16.65	49.69	1014.01	91.00	460.03
9564	13.19	39.18	1023.67	66.78	469.62
9565	31.32	74.33	1012.92	36.48	429.57
9566	24.48	69.45	1013.86	62.39	435.74
9567	21.60	62.52	1017.23	67.87	453.28

9568 rows × 5 columns

Step 5: Checking for Null data

```
[7]  df.isnull()
```

	AT	V	AP	RH	PE
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	False
...	...	...	...	...	...
9563	False	False	False	False	False
9564	False	False	False	False	False
9565	False	False	False	False	False
9566	False	False	False	False	False
9567	False	False	False	False	False

9568 rows × 5 columns

No null data found

Step 6: Calculating summary statistics of data

```
df.describe()
```

	AT	V	AP	RH	PE
count	9568.000000	9568.000000	9568.000000	9568.000000	9568.000000
mean	19.651231	54.305804	1013.259078	73.308978	454.365009
std	7.452473	12.707893	5.938784	14.600269	17.066995
min	1.810000	25.360000	992.890000	25.560000	420.260000
25%	13.510000	41.740000	1009.100000	63.327500	439.750000
50%	20.345000	52.080000	1012.940000	74.975000	451.550000
75%	25.720000	66.540000	1017.260000	84.830000	468.430000
max	37.110000	81.560000	1033.300000	100.160000	495.760000

Step 7: Initializing X and Y variables and checking the values are initialized or not by printing them

```
[9] X = df.drop(['PE'], axis=1).values
    Y = df['PE'].values
```

Values are initialized

```
print(X)

[[  14.96   41.76 1024.07   73.17]
 [  25.18   62.96 1020.04   59.08]
 [   5.11   39.4  1012.16   92.14]
 ...
 [  31.32   74.33 1012.92   36.48]
 [  24.48   69.45 1013.86   62.39]
 [  21.6    62.52 1017.23   67.87]]

[11] print(Y)

[463.26 444.37 488.56 ... 429.57 435.74 453.28]
```

Values for X and Y variable are initialized successfully

Step 8: Splitting the dataset for training and testing using sklearn library

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=0)
```

Step 9: Importing Linear Regression library from sklearn

```
[14] from sklearn.linear_model import LinearRegression

[15] ml= LinearRegression()
```

Step 10: Testing the predictions

```
[17] Y_pred = ml.predict(X_test)
     print(Y_pred)

     [431.40245096 458.61474119 462.81967423 ... 432.47380825 436.16417243
      439.00714594]

[19] ml.predict([[14.96, 41.76, 1024.07, 73.17]])

     array([467.34820092])
```

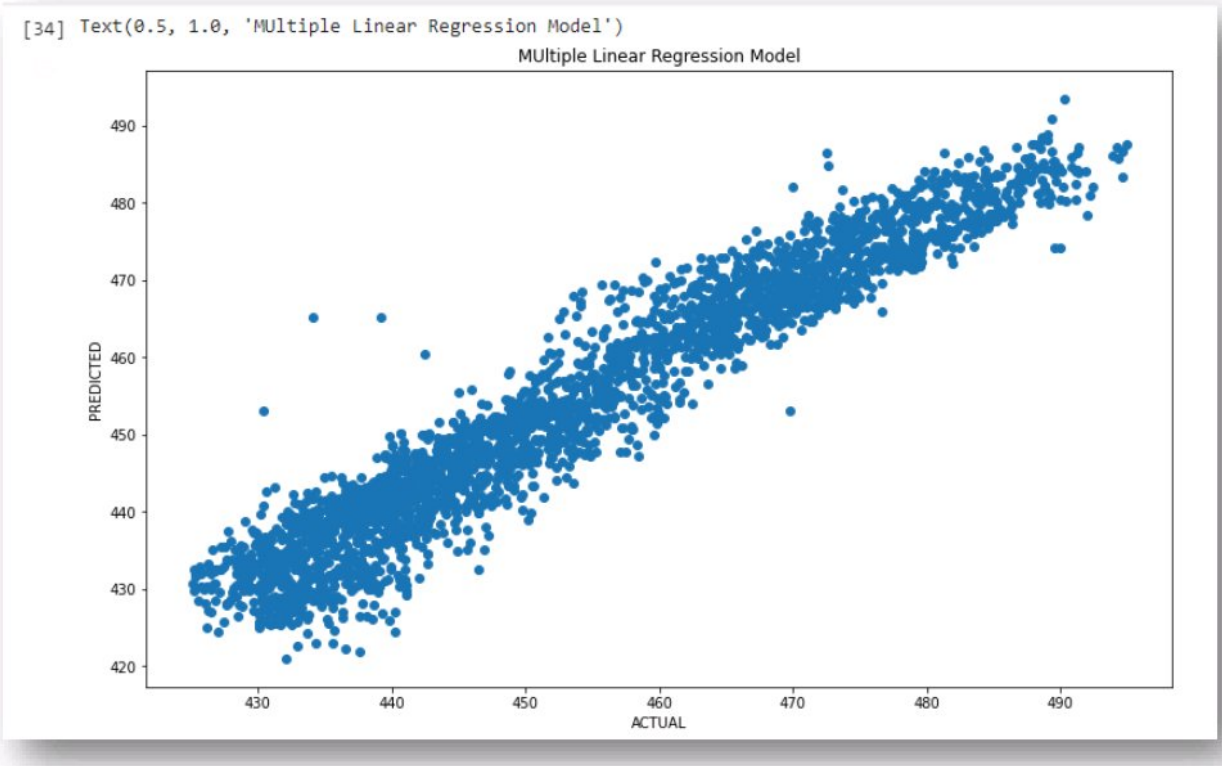
Step 11: Calculating r2 Score for evaluating model accuracy

```
[20] from sklearn.metrics import r2_score
     r2_score(Y_test, Y_pred)

     0.9304112159477683
```

The r2 score is **93%** which is good.

Step 12: Plotting the result

```
[34] import matplotlib.pyplot as plt
     plt.figure(figsize=(13,8))
     plt.scatter(Y_test, Y_pred)
     plt.xlabel('ACTUAL')
     plt.ylabel('PREDICTED')
     plt.title('MUltiple Linear Regression Model')

     Text(0.5, 1.0, 'MUltiple Linear Regression Model')
```



Conclusion: Multiple Linear Regression model is implemented successfully.

Practical No. 6

Aim: To perform Multiple Logistic Regression using google colab

Output:

Description:

LOGISTIC REGRESSION

Logistic regression is basically a supervised classification algorithm. In a classification problem, the target variable(or output), y, can take only discrete values for given set of features(or inputs), X.

Contrary to popular belief, logistic regression is a regression model. The model builds a regression model to predict the probability that a given data entry belongs to the category numbered as “1”. Just like Linear regression assumes that the data follows a linear function, Logistic regression models the data using the sigmoid function.

Steps:

For building logistic Regression Model, We have utilized titanic dataset and built the logistic regression model

Step 1: Importing necessary libraries

Titanic Dataset (Titanic Disaster Survival Using Logistic Regression)

Importing Necessary Libraries

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

Step 2: Loading the dataset

```
[ ]  from google.colab import files
     uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving titanic_train.csv to titanic_train (1).csv
```

Step 3: Reading the dataset

```
[ ]  import io
     data = pd.read_csv(io.BytesIO(uploaded['titanic_train.csv']))
```

Step 4: Exploring the dataset

```
[ ]  len(data)

     891
```

Viewing some record from data

```
data.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Getting Index Of the Data

```
[ ]  data.index

     RangeIndex(start=0, stop=891, step=1)
```

```
data.columns

Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

```
#Getting Datatype
data.dtypes
```

```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

```
[ ] data.describe()
```

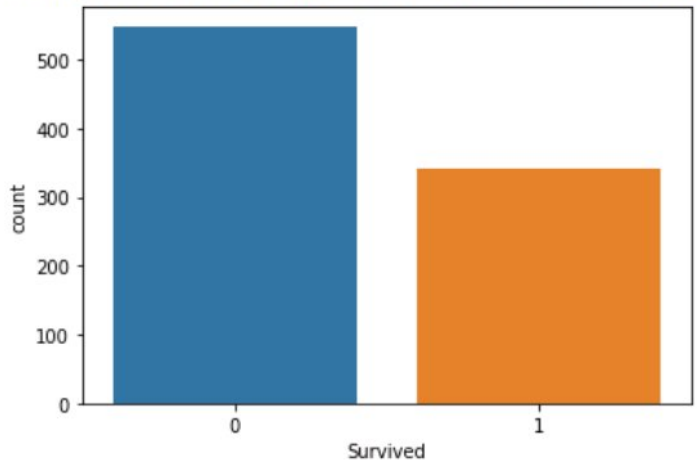
	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Step 5: Data Visualization

For Visual analysis of data we are going to check how many survived and died...

```
[ ] sns.countplot(x='Survived',data=data)
```

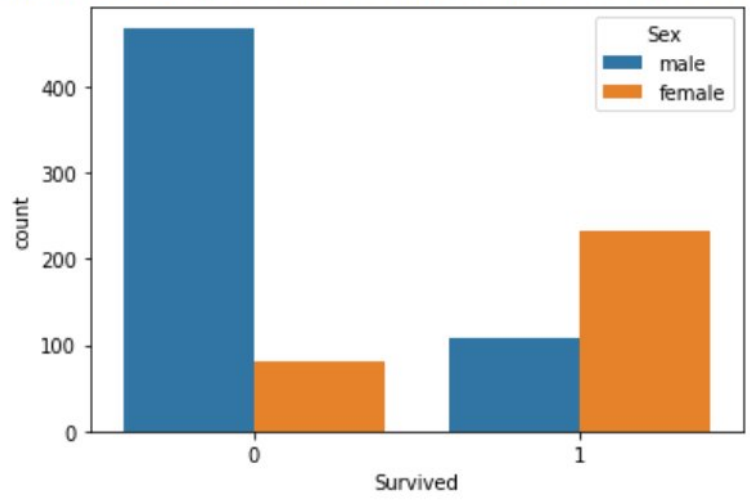
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb560ea5710>
```



Let's go deep inside and check female vs male survival

```
[ ]  sns.countplot(x='Survived',data=data,hue='Sex')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb560e2c7d0>
```



Step 5: Data Cleaning

```
[ ] data.isna()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	False	False	False	False	False	False	False	False	False	False	True	False
1	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	True	False
3	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	True	False
...	...	...	...	...	...	...	...	...	...	...	...	...
886	False	False	False	False	False	False	False	False	False	False	True	False
887	False	False	False	False	False	False	False	False	False	False	False	False
888	False	False	False	False	False	True	False	False	False	False	True	False
889	False	False	False	False	False	False	False	False	False	False	False	False
890	False	False	False	False	False	False	False	False	False	False	True	False

891 rows x 12 columns

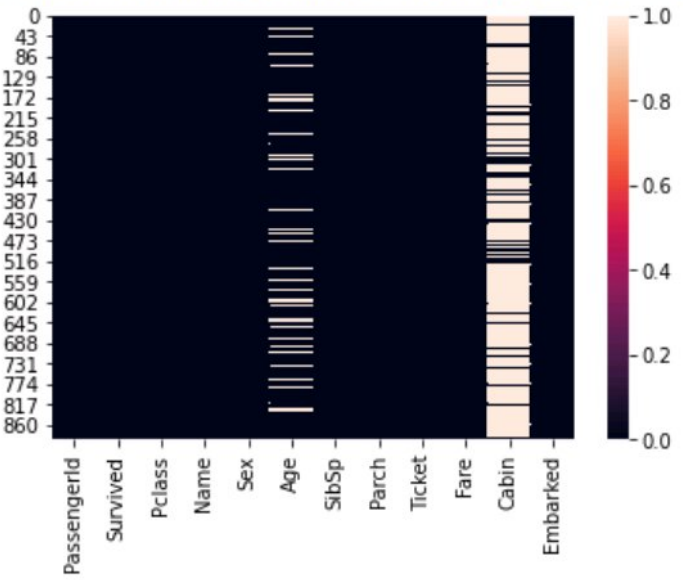
```
[ ] #Getting the null values in number
    data.isna().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

Visualizing null values

```
#Visualizing null values
sns.heatmap(data.isna())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb560dfb650>
```




```
#For making further decision whether to delete the age column or to drop I am calculating percentage then i'll decide
data['Age'].isna().sum()/len(data['Age'])
```

```
0.19865319865319866
```

```
(data['Age'].isna().sum()/len(data['Age']))*100
```

```
19.865319865319865
```

It is just equal to 20 percent so it can be imputed

```
#Now let's try for cabin column
data['Cabin'].isna().sum()/len(data['Cabin'])
```

```
0.7710437710437711
```

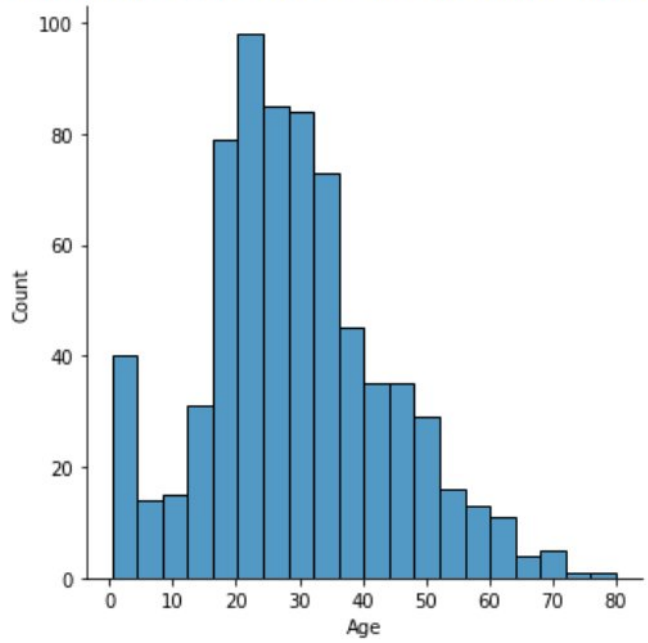
```
(data['Cabin'].isna().sum()/len(data['Cabin']))*100
```

```
77.10437710437711
```

77% is too much so we will discard the whole column

```
[ ]  sns.displot(x='Age', data=data)
```

```
<seaborn.axisgrid.FacetGrid at 0x7fb56299ecd0>
```



```
[ ] data['Age'].mean()
```

```
29.69911764705882
```

```
[ ] #Let's fill the age column
    data['Age'].fillna(data['Age'].mean(),inplace=True)
```

```
[ ] #Now just verifying
    data['Age'].isna()
```

```
0      False
1      False
2      False
3      False
4      False
       ...
886    False
887    False
888    False
889    False
890    False
Name: Age, Length: 891, dtype: bool
```

```
[ ] data['Age'].isna().sum()
```

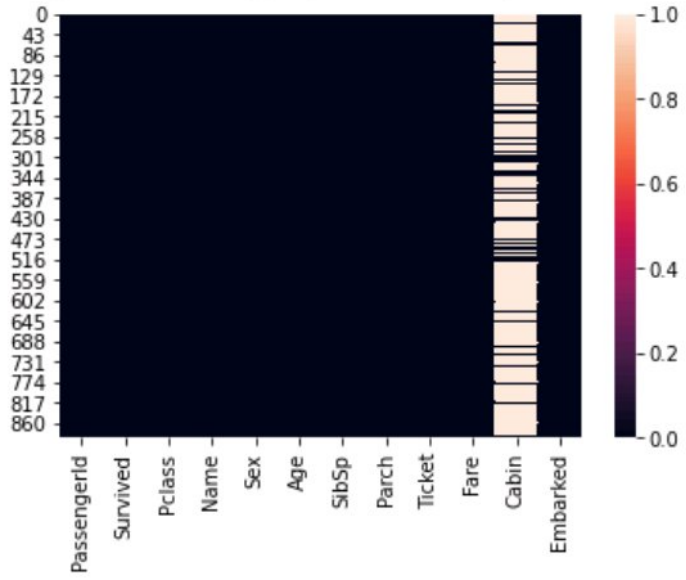
```
0
```

```
[ ] data['Age'].isna().sum()
```

```
0
```

```
[ ] #Visualizing null values
    sns.heatmap(data.isna())
```

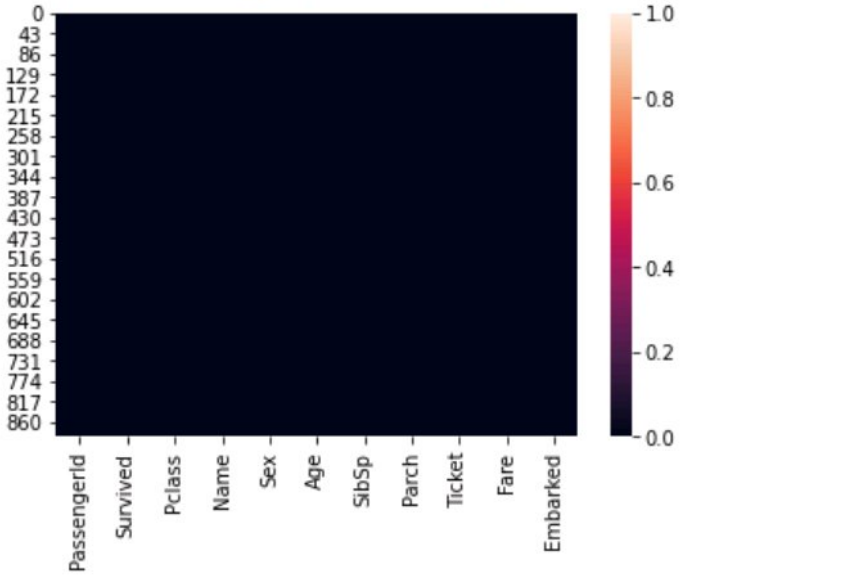
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb560c2f9d0>
```



```
[ ]  #Dropping Cabin Column
     data.drop('Cabin',axis=1,inplace=True)
```

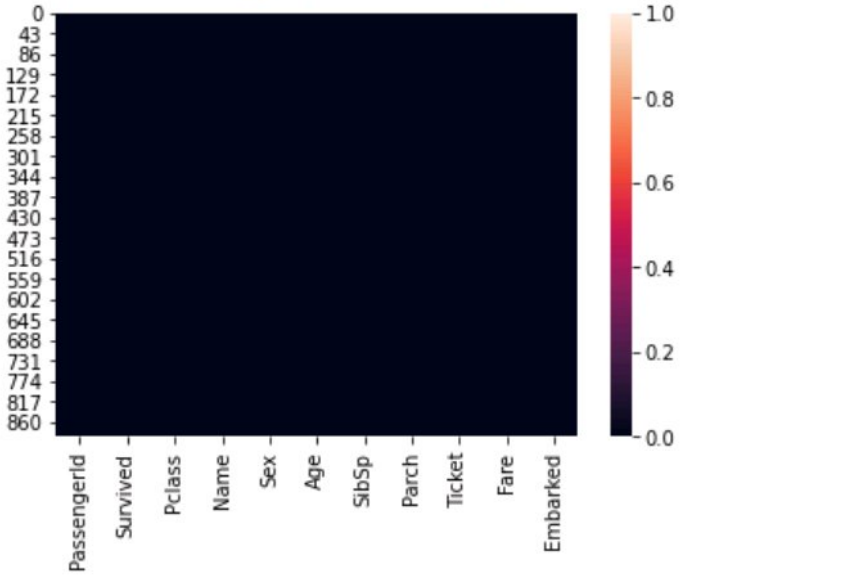
```
[ ]  #Visualizing null values
     sns.heatmap(data.isna())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb560b3b610>
```



```
[ ]  #Dropping Cabin Column
     data.drop('Cabin',axis=1,inplace=True)

[ ]  #Visualizing null values
     sns.heatmap(data.isna())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb560b3b610>
```



Data is cleaned now

Explored data again for conformation

```
[ ] data.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	S

Model Prep(Training)

```
[ ] data.info
```

```
<bound method DataFrame.info of      PassengerId  Survived  Pclass  ...            Ticket     Fare  Embarked
0              1         0       3  ...         A/5 21171   7.2500         S
1              2         1       1  ...          PC 17599  71.2833         C
2              3         1       3  ...  STON/O2. 3101282   7.9250         S
3              4         1       1  ...            113803  53.1000         S
4              5         0       3  ...            373450   8.0500         S
..           ...       ...     ...  ...               ...      ...       ...
886          887         0       2  ...            211536  13.0000         S
887          888         1       1  ...            112053  30.0000         S
888          889         0       3  ...        W./C. 6607  23.4500         S
889          890         1       1  ...            111369  30.0000         C
```

```
data.dtypes
```

```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Embarked        object
dtype: object
```

Step 6: Converting categorical data to numerical data

name, sex, age and fare are not numerical...

```
[ ]  pd.get_dummies(data['Sex'])
```

	female	male
0	0	1
1	1	0
2	1	0
3	1	0
4	0	1
...	...	...
886	0	1
887	1	0
888	1	0
889	0	1
890	0	1

891 rows × 2 columns

```
[ ]  pd.get_dummies(data['Sex'], drop_first=True)
```

	male
0	1
1	0
2	0
3	0
4	1
...	...
886	1
887	0
888	0
889	1
890	1

891 rows × 1 columns

```
[ ]  gender= pd.get_dummies(data['Sex'], drop_first=True)
```

```
[ ]  data['Gender']=gender
```

```
[ ]  data.columns
```

```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Embarked', 'Gender'],
      dtype='object')
```

```
[ ]  data.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	Gender
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	S	1
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C	0
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	S	0
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	S	0
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	S	1

Step 6: Dropping the columns which are not required for prediction

```
[ ]  #Dropping columns not required for prediction
     data.drop(['Name','Sex','Ticket','Embarked'],axis=1,inplace=True)
```

```
[ ]  data.head()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	Gender
0	1	0	3	22.0	1	0	7.2500	1
1	2	1	1	38.0	1	0	71.2833	0
2	3	1	3	26.0	0	0	7.9250	0
3	4	1	1	35.0	1	0	53.1000	0
4	5	0	3	35.0	0	0	8.0500	1

Step 7: Initializing variables of x and y

```
#Making variables for dependent and independent columns
x=data[['PassengerId','Pclass','Age','SibSp','Parch','Fare','Gender']]
y=data['Survived']
```

Step 8: Printing values of x and y

```
x
```

	PassengerId	Pclass	Age	SibSp	Parch	Fare	Gender
0	1	3	22.000000	1	0	7.2500	1
1	2	1	38.000000	1	0	71.2833	0
2	3	3	26.000000	0	0	7.9250	0
3	4	1	35.000000	1	0	53.1000	0
4	5	3	35.000000	0	0	8.0500	1
...	...	...	...	...	...	...	...
886	887	2	27.000000	0	0	13.0000	1
887	888	1	19.000000	0	0	30.0000	0
888	889	3	29.699118	1	2	23.4500	0
889	890	1	26.000000	0	0	30.0000	1
890	891	3	32.000000	0	0	7.7500	1

891 rows × 7 columns

```
[ ] y

    0      0
    1      1
    2      1
    3      1
    4      0
          ..
    886    0
    887    1
    888    0
    889    1
    890    0
    Name: Survived, Length: 891, dtype: int64
```

Step 9: Finally building logistic regression model with sklearn

Model Building

+ Code + Text

```
[ ] from sklearn.model_selection import train_test_split

[ ] x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_state=42)

[ ] from sklearn.linear_model import LogisticRegression

[ ] lr=LogisticRegression()
```

Step 10: Training the model

```
lr.fit(x_train,y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

Step 11: Testing the model and calculating confusion matrix

```
[ ] predict = lr.predict(x_test)
```

Testing

```
[ ] from sklearn.metrics import confusion_matrix
```

```
[ ] confusion_matrix(y_test,predict)
```

```
array([[151,  24],
       [ 38,  82]])
```

```
[ ] pd.DataFrame(confusion_matrix(y_test,predict), columns=['Predictred NO','Predicted YES'], index=['Actual NO','Actual YES'])
```

	Predictred NO	Predicted YES
Actual NO	151	24
Actual YES	38	82

```
from sklearn.metrics import classification_report
```

```
[ ] print(classification_report(y_test,predict))
```

```
                precision    recall  f1-score   support

             0       0.80      0.86      0.83       175
             1       0.77      0.68      0.73       120

      accuracy                           0.79       295
     macro avg       0.79      0.77      0.78       295
  weighted avg       0.79      0.79      0.79       295
```

Precision: Correctly predicted positive observations to the total predicted positive observations

Recall: Correctly predicted positive observations to the total actual positive observations

Score is weighted average of precision and recall

Conclusion: Logistic regression model is implemented successfully.

Practical No. 7

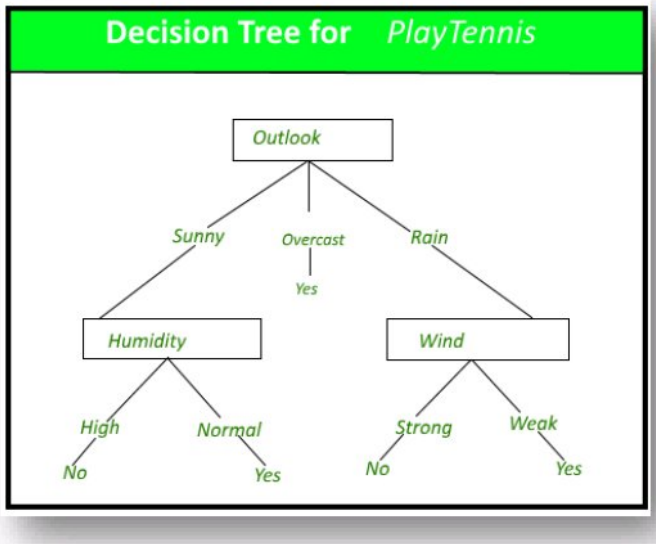
Aim: To build Decision Tree Model using google colab

Output:

Description:

DECISION TREE

Decision tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.



Steps:

Step 1: Importing necessary libraries

```
#importing dependencies
from sklearn import tree
import pandas as pd
import pydotplus
from IPython.display import Image
```

Step 2: Impoting dataset and reading it

```
#importing dataset
from google.colab import files
uploaded = files.upload()
```

```
Choose Files No file chosen  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving temp_decision.xlsx to temp_decision (1).xlsx
```

```
[ ] import io
data = pd.read_excel(io.BytesIO(uploaded['temp_decision.xlsx']))
```

Step 3: Printing data

```
data
```

	Outlook	Temperature	Humidity	Windy	Play
0	sunny	hot	high	f	no
1	sunny	hot	high	t	no
2	overcast	hot	high	f	yes
3	rainy	mild	high	t	yes
4	rainy	cool	normal	f	yes
5	rainy	cool	normal	t	no
6	overcast	cool	normal	f	yes
7	sunny	mild	high	t	no
8	sunny	cool	normal	f	yes
9	rainy	mild	normal	t	yes
10	sunny	mild	normal	f	yes
11	overcast	mild	high	t	yes
12	overcast	hot	normal	f	yes
13	rainy	mild	high	t	no

Step 4: Converting categorical data to numerical data and printing it

```
[ ] one_hot_data = pd.get_dummies(data[['Outlook','Temperature','Humidity','Windy']])
```

```
[ ] print(one_hot_data)
```

```
    Outlook_overcast  Outlook_rainy  ...  Windy_f  Windy_t
0                  0              0  ...        1        0
1                  0              0  ...        0        1
2                  1              0  ...        1        0
3                  0              1  ...        0        1
4                  0              1  ...        1        0
5                  0              1  ...        0        1
6                  1              0  ...        1        0
7                  0              0  ...        0        1
8                  0              0  ...        1        0
9                  0              1  ...        0        1
10                 0              0  ...        1        0
11                 1              0  ...        0        1
12                 1              0  ...        1        0
13                 0              1  ...        0        1
```

Step 5: Creating decision tree classifier and training it

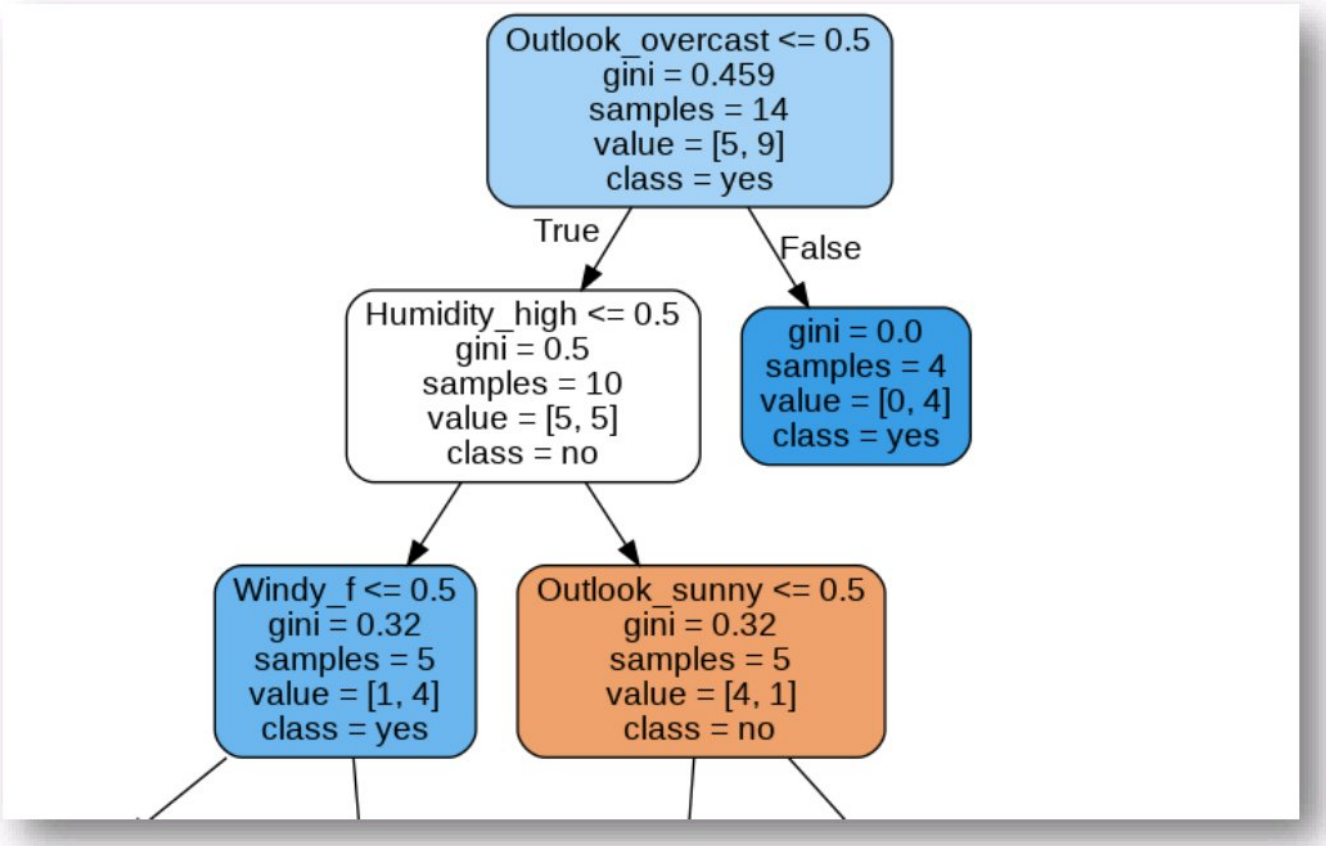
```
[ ] #creating decision tree classifier
    clf = tree.DecisionTreeClassifier()

    #training the decision tree
    clf_train = clf.fit(one_hot_data, data['Play'])
```

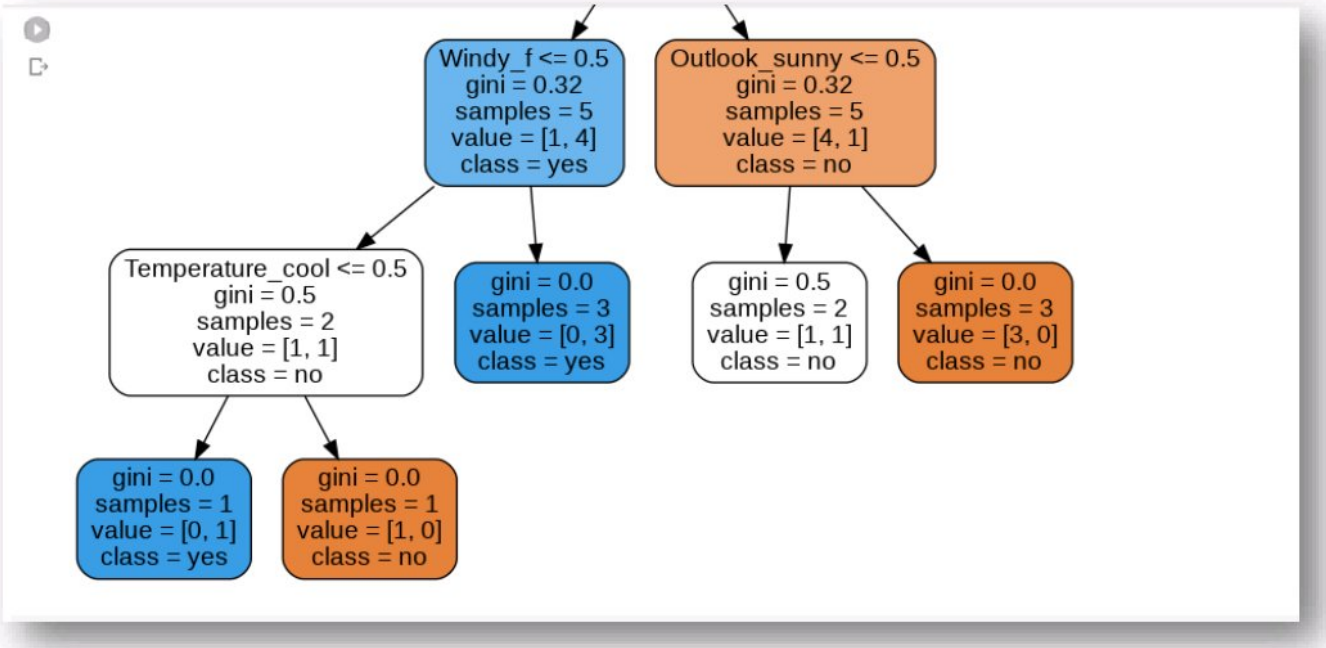
```
dot_data = tree.export_graphviz(clf_train, out_file = None, feature_names = list(one_hot_data.columns.values))
print(dot_data)
```

```
digraph Tree {
node [shape=box] ;
0 [label="Outlook_overcast <= 0.5\ngini = 0.459\nsamples = 14\nvalue = [5, 9]"] ;
1 [label="Humidity_high <= 0.5\ngini = 0.5\nsamples = 10\nvalue = [5, 5]"] ;
0 -> 1 [labeldistance=2.5, labelangle=45, headlabel="True"] ;
2 [label="Windy_f <= 0.5\ngini = 0.32\nsamples = 5\nvalue = [1, 4]"] ;
1 -> 2 ;
3 [label="Temperature_cool <= 0.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]"] ;
2 -> 3 ;
4 [label="gini = 0.0\nsamples = 1\nvalue = [0, 1]"] ;
3 -> 4 ;
5 [label="gini = 0.0\nsamples = 1\nvalue = [1, 0]"] ;
3 -> 5 ;
6 [label="gini = 0.0\nsamples = 3\nvalue = [0, 3]"] ;
2 -> 6 ;
7 [label="Outlook_sunny <= 0.5\ngini = 0.32\nsamples = 5\nvalue = [4, 1]"] ;
1 -> 7 ;
8 [label="gini = 0.5\nsamples = 2\nvalue = [1, 1]"] ;
7 -> 8 ;
9 [label="gini = 0.0\nsamples = 3\nvalue = [3, 0]"] ;
7 -> 9 ;
10 [label="gini = 0.0\nsamples = 4\nvalue = [0, 4]"] ;
0 -> 10 [labeldistance=2.5, labelangle=-45, headlabel="False"] ;
}
```

Step 6: Printing decision tree of the model

```
dot_data = tree.export_graphviz(clf_train, out_file = None, feature_names = list(one_hot_data.columns.values),rounded = True, filled = True,
graph = pydotplus.graph_from_dot_data(dot_data)

#Showing graph
Image(graph.create_png())
```





Step 7: Testing prediction model

```
[ ]  #testing prediction model
     #outlook = sunny, temperature = hot, humidity = normal, windy = false
     prediction = clf_train.predict([[0,0,1,0,1,0,0,1,1,0]])
     print(prediction)

     ['yes']
```

Conclusion: Decision Tree Model is implemented successfully.

Practical No. 8

Aim: To build Suport Vector Machine model (SVM) using google colab

Output:

Description:

SUPPORT VECTOR MACHINE(SVM)

Support Vector Machine (SVM) is a relatively simple **Supervised Machine Learning Algorithm** used for classification and/or regression. It is more preferred for classification but is sometimes very useful for regression as well. Basically, SVM finds a hyper-plane that creates a boundary between the types of data. In 2-dimensional space, this hyper-plane is nothing but a line.
In SVM, we plot each data item in the dataset in an N-dimensional space, where N is the number of features/attributes in the data. Next, find the optimal hyperplane to separate the data. So by this, you must have understood that inherently, SVM can only perform binary classification (i.e., choose between two classes). However, there are various techniques to use for multi-class problems.

Support Vector Machine for Multi-CLass Problems

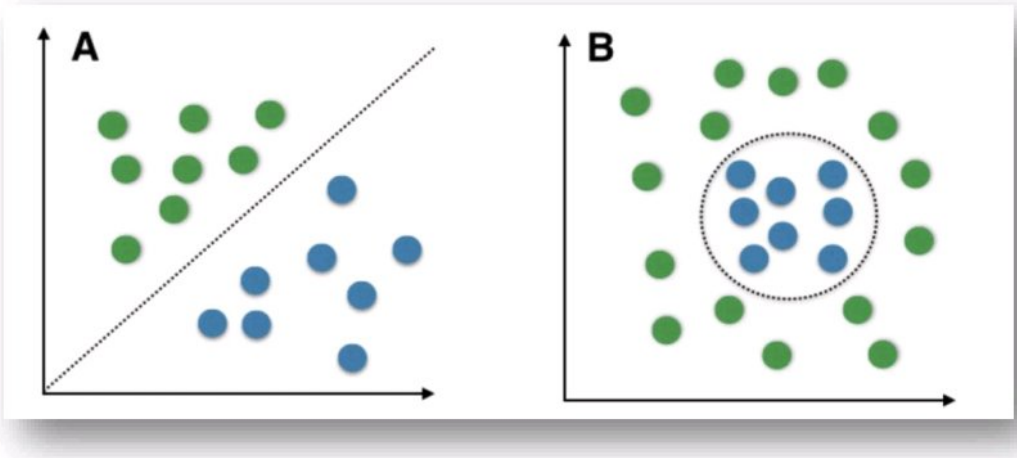
To perform SVM on multi-class problems, we can create a binary classifier for each class of the data. The two results of each classifier will be :

- The data point belongs to that class OR
- The data point does not belong to that class.

For example, in a class of fruits, to perform multi-class classification, we can create a binary classifier for each fruit. For say, the 'mango' class, there will be a binary classifier to predict if it IS a mango OR it is NOT a mango. The classifier with the highest score is chosen as the output of the SVM.

SVM for complex (Non Linearly Separable)

SVM works very well without any modifications for linearly separable data. **Linearly Separable Data** is any data that can be plotted in a graph and can be separated into classes using a straight line.



Linearly Separable & Non-Linearly Separable Data

Steps:

Step 1: Importing necessary libraries

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

Step 2: Importing dataset and reading the data

```
from google.colab import files
uploaded = files.upload()
```

```
Choose Files  No file chosen    Upload widget is only available when
Saving cell_samples.csv to cell_samples.csv
```

```
[ ] import io
data = pd.read_csv(io.BytesIO(uploaded['cell_samples.csv']))
```

Step 3: Exploring the dataset

```
[ ] len(data)
```

```
699
```

```
[ ] data.head()
```

	ID	Clump	UnifSize	UnifShape	MargAdh	SingEpiSize	BareNuc	BlandChrom	NormNucl	Mit	Class
0	1000025	5	1	1	1	2	1	3	1	1	2
1	1002945	5	4	4	5	7	10	3	2	1	2
2	1015425	3	1	1	1	2	2	3	1	1	2
3	1016277	6	8	8	1	3	4	3	7	1	2
4	1017023	4	1	1	3	2	1	3	1	1	2

```
[ ] data.tail()
```

	ID	Clump	UnifSize	UnifShape	MargAdh	SingEpiSize	BareNuc	BlandChrom	NormNucl	Mit	Class
694	776715	3	1	1	1	3	2	1	1	1	2
695	841769	2	1	1	1	2	1	1	1	1	2
696	888820	5	10	10	3	7	3	8	10	2	4
697	897471	4	8	6	4	3	4	10	6	1	4

```
[ ] data.index
```

```
RangeIndex(start=0, stop=699, step=1)
```

```
[ ] data.columns
```

```
Index(['ID', 'Clump', 'UnifSize', 'UnifShape', 'MargAdh', 'SingEpiSize',
       'BareNuc', 'BlandChrom', 'NormNucl', 'Mit', 'Class'],
      dtype='object')
```

```
[ ] #Getting Datatype
    data.dtypes
```

```
ID              int64
Clump           int64
UnifSize        int64
UnifShape       int64
MargAdh         int64
SingEpiSize     int64
BareNuc        object
BlandChrom      int64
NormNucl        int64
Mit             int64
Class           int64
dtype: object
```

```
[ ] data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 699 entries, 0 to 698
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   ID           699 non-null    int64
 1   Clump        699 non-null    int64
 2   UnifSize     699 non-null    int64
 3   UnifShape    699 non-null    int64
 4   MargAdh      699 non-null    int64
 5   SingEpiSize  699 non-null    int64
 6   BareNuc      699 non-null    object
 7   BlandChrom   699 non-null    int64
 8   NormNucl     699 non-null    int64
 9   Mit          699 non-null    int64
 10  Class        699 non-null    int64
dtypes: int64(10), object(1)
memory usage: 60.2+ KB
```

```
data.describe()
```

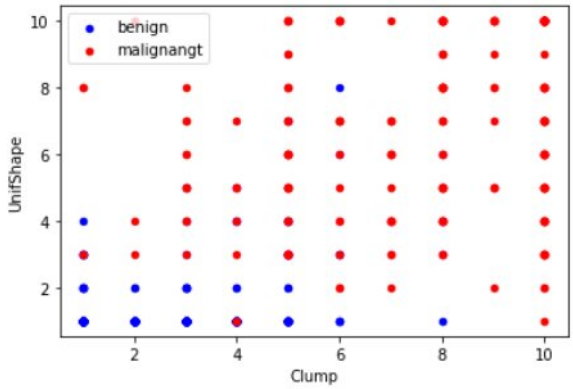
	ID	Clump	UnifSize	UnifShape	MargAdh	SingEpiSize	BlandChrom	NormNucl	Mit	Class
count	6.990000e+02	699.000000	699.000000	699.000000	699.000000	699.000000	699.000000	699.000000	699.000000	699.000000
mean	1.071704e+06	4.417740	3.134478	3.207439	2.806867	3.216023	3.437768	2.866953	1.589413	2.689557
std	6.170957e+05	2.815741	3.051459	2.971913	2.855379	2.214300	2.438364	3.053634	1.715078	0.951273
min	6.163400e+04	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	2.000000
25%	8.706885e+05	2.000000	1.000000	1.000000	1.000000	2.000000	2.000000	1.000000	1.000000	2.000000
50%	1.171710e+06	4.000000	1.000000	1.000000	1.000000	2.000000	3.000000	1.000000	1.000000	2.000000
75%	1.238298e+06	6.000000	5.000000	5.000000	4.000000	4.000000	5.000000	4.000000	1.000000	4.000000
max	1.345435e+07	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	4.000000

```
[ ] data['Class'].value_counts()

2    458
4    241
Name: Class, dtype: int64
```

```
[ ] benign_data=data[data['Class']==2] [0:200]
    malignant_data=data[data['Class']==4] [0:200]
```

```
[ ] axes=benign_data.plot(kind='scatter', x='Clump', y='UnifShape', color='blue', label='benign')
    malignant_data.plot(kind='scatter', x='Clump', y='UnifShape', color='red', label='malignangt', ax=axes)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff5e46fd210>
```



```
[ ] #Identifying unwanted rows
    data.dtypes
```

```
ID              int64
Clump           int64
UnifSize        int64
UnifShape       int64
MargAdh         int64
SingEpiSize     int64
BareNuc        object
BlandChrom      int64
NormNucl        int64
Mit             int64
Class           int64
dtype: object
```

```
[ ] data=data[pd.to_numeric(data['BareNuc'], errors='coerce').notnull()]
    data['BareNuc']=data['BareNuc'].astype('int')
    data.dtypes
```

```
ID              int64
Clump           int64
UnifSize        int64
UnifShape       int64
MargAdh         int64
SingEpiSize     int64
BareNuc         int64
BlandChrom      int64
NormNucl        int64
Mit             int64
```

Step 4: Data Cleaning

```
[ ] #Removing unwanted col.
    feature_data= data[['Clump','UnifSize','UnifShape', 'MargAdh', 'SingEpiSize', 'BareNuc', 'BlandChrom', 'NormNucl', 'Mit']]
```

Step 5: Initializing the X and Y variables

```
#independent var.
X = np.asarray(feature_data)

#dependent var.
y= np.asarray(data['Class'])
y[0:5]
```

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

Step 6: Training the dataset

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4)
```

```
[ ] X_train.shape
```

```
(546, 9)
```

```
[ ] y_train.shape
```

```
(546,)
```

```
[ ] X_test.shape
```

```
(137, 9)
```

```
[ ] y_test.shape
```

```
(137,)
```

Step 7: Testing the data and checking the results

```
[ ]  from sklearn import svm

     clr=svm.SVC(kernel='linear', gamma='auto', C=2)
```

```
[ ]  clr.fit(X_train, y_train)

     y_predict= clr.predict(X_test)
```

```
[ ]  #result
     from sklearn.metrics import classification_report
```

```
[ ]  print(classification_report(y_test,y_predict))
```

```
                precision    recall  f1-score   support

             2       1.00      0.94      0.97        90
             4       0.90      1.00      0.95        47

      accuracy                           0.96       137
     macro avg       0.95      0.97      0.96       137
  weighted avg       0.97      0.96      0.96       137
```

Conclusion: The SVM Model is implemented successfully.

Practical No. 9

Aim: To perform clustering using KMeans clustering

Output:

Description:

KMEANS CLUSTERING

***k*-means clustering** is a method of vector quantization, originally from signal processing, that aims to partition *n* observations into *k* clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells. *k*-means clustering minimizes within-cluster variances (squared Euclidean distances), but not regular Euclidean distances, which would be the more difficult Weber problem: the mean optimizes squared errors, whereas only the geometric median minimizes Euclidean distances. For instance, better Euclidean solutions can be found using k-medians and k-medoids.

Steps:

Step 1: Importing necessary libraries

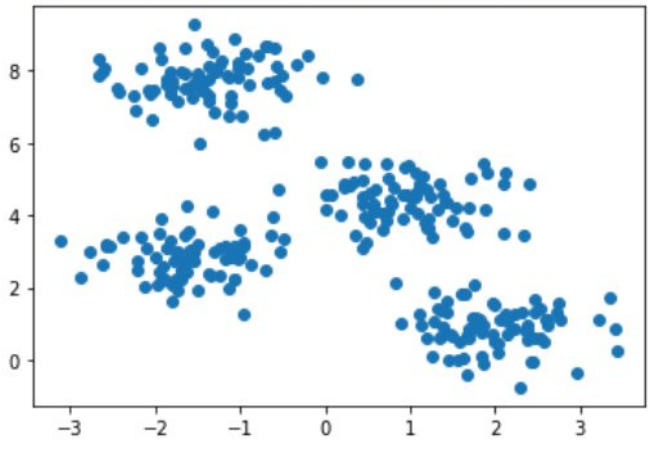
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.datasets.samples_generator import make_blobs
from sklearn.cluster import KMeans
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:144: FutureWarning: The sklearn.da
  warnings.warn(message, FutureWarning)
```

Step 2: Printed data

```
X, y = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)
plt.scatter(X[:,0], X[:,1])
```

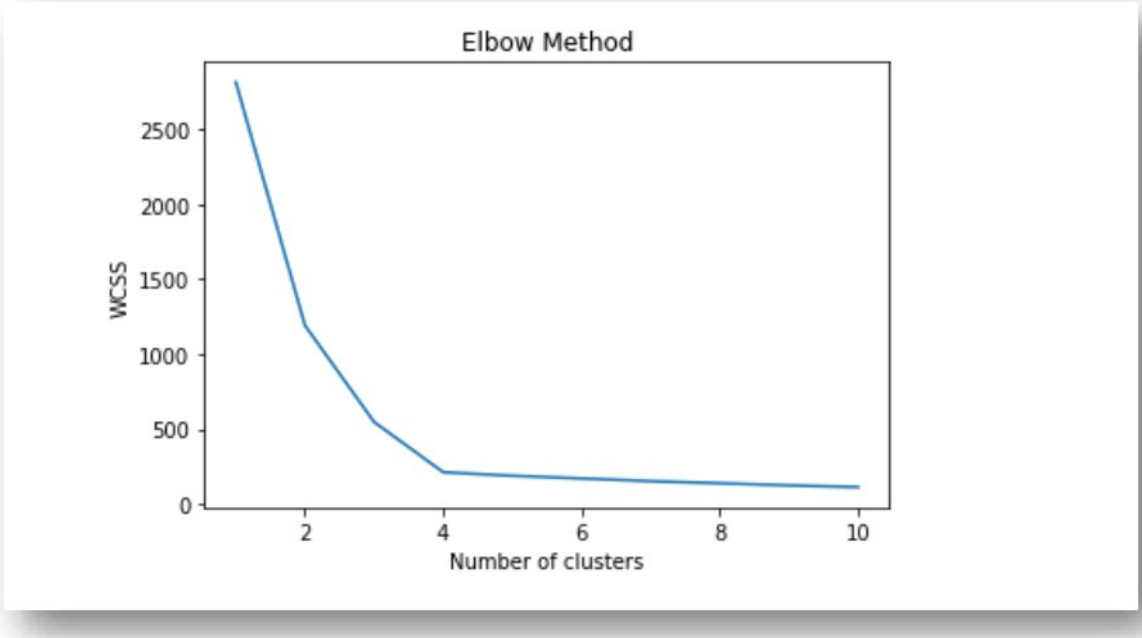
```
<matplotlib.collections.PathCollection at 0x7f3298a98940>
```



Step 3: Using Elbow method

Even though we already know the optimal number of clusters, I figured we could still benefit from determining it using the elbow method. To get the values used in the graph, we train multiple models using a different number of clusters and storing the value of the intertia_ property (WCSS) every time.

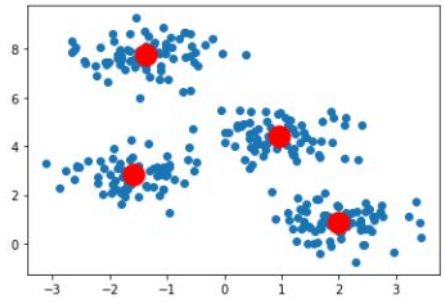
```
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=0)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



Step 4: Finally clustering the data

Next, we'll categorize the data using the optimum number of clusters (4) we determined in the last step. k-means++ ensures that you get don't fall into the random initialization trap.

```
[ ]  kmeans = KMeans(n_clusters=4, init='k-means++', max_iter=300, n_init=10, random_state=0)
     pred_y = kmeans.fit_predict(X)
     plt.scatter(X[:,0], X[:,1])
     plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='red')
     plt.show()
```



Conclusion: KMeans clustering is performed successfully .

Practical No. 10

Aim: To perform reinforcement learning

Output:

Description:

REINFORCEMENT LEARNING

Reinforcement learning (**RL**) is an area of machine learning concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward. Reinforcement learning is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning.

Reinforcement learning differs from supervised learning in not needing labeled input/output pairs be presented, and in not needing sub-optimal actions to be explicitly corrected. Instead the focus is on finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge).

Q* Learning with FrozenLake

The goal of this game is **to go from the starting state (S) to the goal state (G)** by walking only on frozen tiles (F) and avoid holes (H). However, the ice is slippery, **so you won't always move in the direction you intend (stochastic environment)**

Steps:

Step 1: Installing dependencies

```
!pip install numpy
!pip install gym

Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (1.14.6)
Collecting gym
  Downloading https://files.pythonhosted.org/packages/d4/22/4ff09745ade385ffe707fb5f053548f0f6a6e7d5e98a2b9d6c07f5b931a7/gym-0.10.9.tar.gz (1.5MB)
    100% |████████████████████████████████| 1.5MB 6.3MB/s
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packages (from gym) (1.1.0)
Requirement already satisfied: numpy>=1.10.4 in /usr/local/lib/python3.6/dist-packages (from gym) (1.14.6)
Requirement already satisfied: requests>=2.0 in /usr/local/lib/python3.6/dist-packages (from gym) (2.18.4)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from gym) (1.11.0)
Collecting pyglet>=1.2.0 (from gym)
  Downloading https://files.pythonhosted.org/packages/1c/fc/dad5eaaab68f0c21e2f906a94ddb98175662cc5a654eee404d59554ce0fa/pyglet-1.3.2-py2.py3-none-any
    100% |████████████████████████████████| 1.0MB 10.5MB/s
Requirement already satisfied: urllib3<1.23,>=1.21.1 in /usr/local/lib/python3.6/dist-packages (from requests>=2.0->gym) (1.22)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/python3.6/dist-packages (from requests>=2.0->gym) (3.0.4)
Requirement already satisfied: idna<2.7,>=2.5 in /usr/local/lib/python3.6/dist-packages (from requests>=2.0->gym) (2.6)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packages (from requests>=2.0->gym) (2018.11.29)
Requirement already satisfied: future in /usr/local/lib/python3.6/dist-packages (from pyglet>=1.2.0->gym) (0.16.0)
Building wheels for collected packages: gym
  Running setup.py bdist_wheel for gym ... done
  Stored in directory: /root/.cache/pip/wheels/6c/3a/0e/b86dee98876bb56cdb482cc1f72201035e46d1baf69d10d028
Successfully built gym
Installing collected packages: pyglet, gym
Successfully installed gym-0.10.9 pyglet-1.3.2
```

Step 2: Importing necessary libraries

```
[ ]  import numpy as np
     import gym
     import random
```

Step 3: Creating the environment

```
env = gym.make("FrozenLake-v0")

/usr/local/lib/python3.6/dist-packages/gym/envs/registration.py:14:
  result = entry_point.load(False)
```

Step 4: Initiating Q table

```
[ ]  action_size = env.action_space.n
     state_size = env.observation_space.n

     # Create our Q table with state_size rows and action_size columns (64x4)
     qtable = np.zeros((state_size, action_size))
     print(qtable)

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

Step 5: Creating hyper-parameters

```
total_episodes = 20000        # Total episodes
learning_rate = 0.7           # Learning rate
max_steps = 99                # Max steps per episode
gamma = 0.95                  # Discounting rate

# Exploration parameters
epsilon = 1.0                 # Exploration rate
max_epsilon = 1.0             # Exploration probability at start
min_epsilon = 0.01            # Minimum exploration probability
decay_rate = 0.005            # Exponential decay rate for exploration prob
```

Step 6: Q-Learning algorithm

```
# List of rewards
rewards = []

# 2 For life or until learning is stopped
for episode in range(total_episodes):
    # Reset the environment
    state = env.reset()
    step = 0
    done = False
    total_rewards = 0

    for step in range(max_steps):
        # 3. Choose an action a in the current world state (s)
        ## First we randomize a number
        exp_exp_tradeoff = random.uniform(0, 1)

        ## If this number > greater than epsilon --> exploitation (taking the biggest Q value for this state)
        if exp_exp_tradeoff > epsilon:
            action = np.argmax(qtable[state,:])
            #print(exp_exp_tradeoff, "action", action)

        # Else doing a random choice --> exploration
        else:
            action = env.action_space.sample()
            #print("action random", action)
```

```
[ ]
        ## If this number > greater than epsilon --> exploitation (taking the biggest Q value for this state)
        if exp_exp_tradeoff > epsilon:
            action = np.argmax(qtable[state,:])
            #print(exp_exp_tradeoff, "action", action)

        # Else doing a random choice --> exploration
        else:
            action = env.action_space.sample()
            #print("action random", action)


        # Take the action (a) and observe the outcome state(s') and reward (r)
        new_state, reward, done, info = env.step(action)

        # Update Q(s,a):= Q(s,a) + lr [R(s,a) + gamma * max Q(s',a') - Q(s,a)]
        # qtable[new_state,:] : all the actions we can take from new state
        qtable[state, action] = qtable[state, action] + learning_rate * (reward + gamma * np.max(qtable[new_state, :]) - qtable[state, action])

        total_rewards += reward

        # Our new state is state
        state = new_state

        # If done (if we're dead) : finish episode
        if done == True:
            break
```

```
[ ]
        # Reduce epsilon (because we need less and less exploration)
        epsilon = min_epsilon + (max_epsilon - min_epsilon)*np.exp(-decay_rate*episode)
        rewards.append(total_rewards)


    print ("Score over time: " +  str(sum(rewards)/total_episodes))
    print(qtable)
```

```
Score over time: 0.50795
[[3.31076341e-01 9.62218123e-02 7.25300527e-02 8.93317624e-02]
 [3.09107962e-02 2.11238465e-02 1.53940941e-03 8.15916450e-02]
 [2.30580059e-02 1.10205893e-02 3.05104836e-02 5.81989088e-02]
 [1.89392072e-03 1.42514492e-03 3.61505851e-03 3.33071549e-02]
 [4.10216080e-01 7.15773401e-02 1.85316436e-02 8.37170734e-02]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [3.27832769e-04 4.25377235e-04 1.36010420e-01 5.37383451e-04]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [5.36724237e-02 1.49053850e-01 1.32325076e-01 5.76347785e-01]
 [1.16731702e-02 7.47882851e-01 5.26449538e-02 5.56077547e-02]
 [6.90443526e-01 7.89701771e-02 1.79099164e-02 7.15556071e-03]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [1.35647619e-01 3.20232106e-01 8.67775997e-01 5.98079233e-04]
 [4.08023861e-01 9.53201064e-01 2.40692710e-01 3.94354541e-01]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]]
```

Step 8: Using Q table to play now

```
env.reset()

for episode in range(5):
    state = env.reset()
    step = 0
    done = False
    print("****************************************************")
    print("EPISODE ", episode)

    for step in range(max_steps):

        # Take the action (index) that have the maximum expected future reward given that state
        action = np.argmax(qtable[state,:])

        new_state, reward, done, info = env.step(action)

        if done:
            # Here, we decide to only print the last state (to see if our agent is on the goal or fall into an hole)
            env.render()
            if new_state == 15:
                print("We reached our Goal 🏆")
            else:
                print("We fell into a hole 💀")

            # We print the number of step it took.
            print("Number of steps", step)

            break
```

```
[ ]         if done:
                # Here, we decide to only print the last state (to see if our agent is on the goal or fall into an hole)
                env.render()
                if new_state == 15:
                    print("We reached our Goal 🏆")
                else:
                    print("We fell into a hole 👻")

                # We print the number of step it took.
                print("Number of steps", step)

                break
            state = new_state
    env.close()
```

```
**************************************************
EPISODE  0
  (Down)
SFFF
FHFH
FFFH
HFF
We reached our Goal 🏆
Number of steps 10
**************************************************
EPISODE  1
**************************************************
EPISODE  2
  (Right)
SFFF
```

```
**************************************************
EPISODE  3
  (Down)
SFFF
FHFH
FFFH
HFF
We reached our Goal 🏆
Number of steps 75
**************************************************
```

Conclusion : Reinforcement Learning model is implemented successfully.