Practical Machine Learning Lab

Lab Manual with Student Lab Record

Developed By

Dr. K. Rajkumar

P. Bhuvaneswari

Roll No	205229133
Name	Viviyan Richards. W
Class	**I MSc Data Science**

Department of Data Science
Bishop Heber College (Autonomous)
Tiruchirappalli 620017, INDIA

March 2021

Department of Data Science
Bishop Heber College (Autonomous)
Tiruchirappalli 620017

BONAFIDE CERTIFICATE

Name: VIVIAN RICHARDS . W

Reg. No: 205229133 **Class:** I. M.Sc (

Course Title DATA SCIENCE

Certified that this is the bonafide record of work done by me during **Odd / Even** Semester of **2020 – 2021** and submitted for the Practical Examination on _____

Staff In-Charge **Head of the Department**

Examiners

1. _____

2. _____

Grade Sheet

Roll No	205829133	Name	VIJIYAN RICHARDS.W
Year	I	Semester	2
		Instructor Name	DR. JANANI SELVARAJ

Lab	Date	Activity	Grade
1	08.03.2021	WarmUp: Familiarity with Data and Visualization	
2	15.03.2021	Pizza Liking Prediction using kNN	
3	21.03.2021	Fuel Amount Prediction using Linear Regression	
4	25.03.2021	House Price Prediction using LR with Regularization	
5	09.04.2021	Diabetes Classification using Logistic Regression	
6	25.04.2021	Predictive Analytics for Hospitals	
7	28.04.2021	Loan Approval Classification using SVM	
8	04.05.2021	Animal Classification using Decision Trees	
9	11.05.2021	Employee Hopping Prediction using Random Forests	
10	27.05.2021	Patients Physical Activities Prediction using Boosting	
11	01.06.2021	Shopping Mall Customer Segmentation using Clustering	

Practical Machine Learning Lab-1

For this PML lab1 I have create a own sample data as ('ml_lab1data.csv') which contain month and no.of.accidents.

Roll no: 205229133

In [18]:
```
import pandas as pd
data=pd.read_csv('ml_lab1data.csv')
```

In [23]:
```
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12 entries, 0 to 11
Data columns (total 2 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Month            12 non-null     object
 1   No.of.Accidents  12 non-null     int64 
dtypes: int64(1), object(1)
memory usage: 320.0+ bytes
None
```

In [24]:
```
print(data.shape)
```

```
(12, 2)
```

In [25]:
```
print(data.size)
```

```
24
```

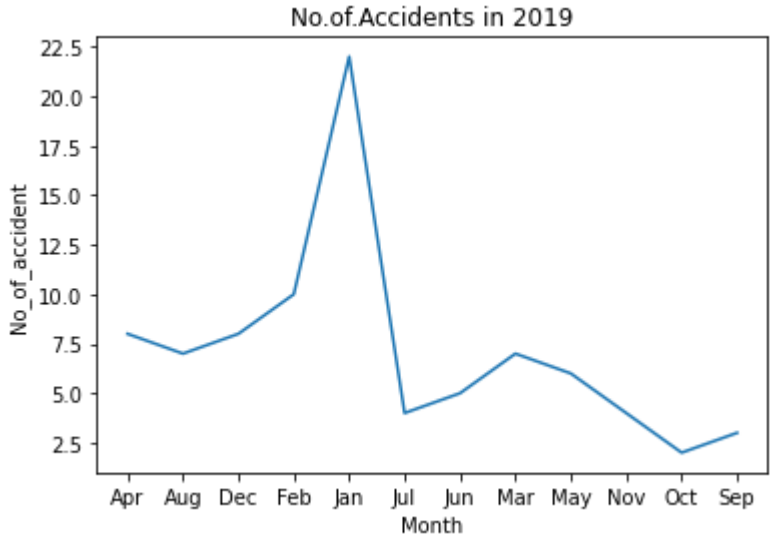
In [26]:
```
print(data.ndim)
```

```
2
```

1.Table Visualization

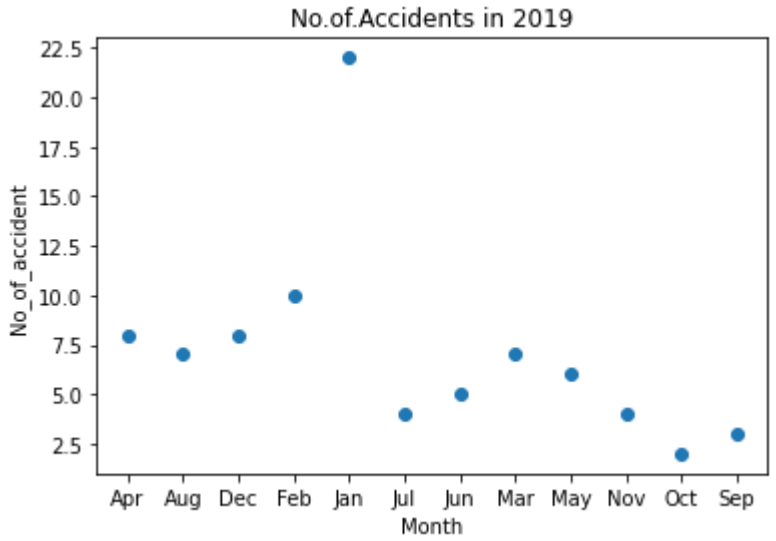
Graph Plot

In [4]:
```
import pandas as pd
from matplotlib import pyplot as plt

df = pd.read_csv('ml_lab1data.csv')

groupedby_acc = df.groupby('Month')[['No.of.Accidents']].sum()

fig, ax = plt.subplots()

ax.plot(groupedby_acc.index, groupedby_acc['No.of.Accidents'])

labels = ax.get_xticklabels()
plt.title('No.of.Accidents in 2019')
plt.xlabel('Month')
plt.ylabel('No_of_accident')

plt.show()
```



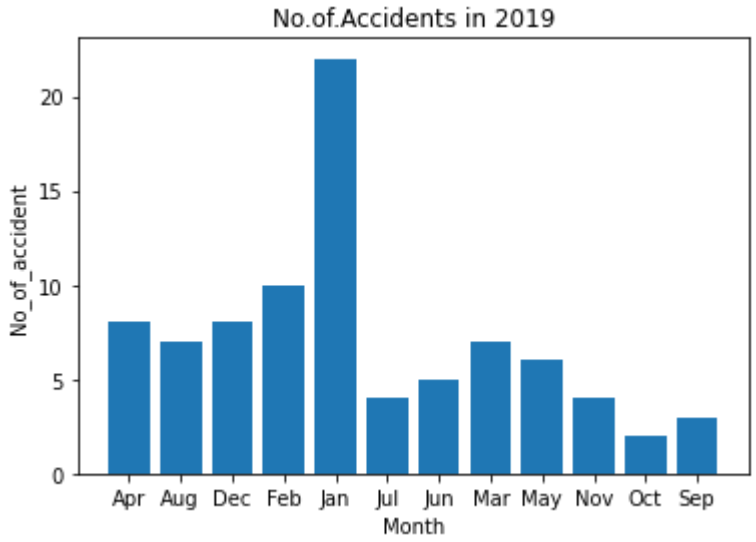
Scatter Plot

In [5]:
```
import pandas as pd
from matplotlib import pyplot as plt

df = pd.read_csv('ml_lab1data.csv')

groupedby_acc = df.groupby('Month')[['No.of.Accidents']].sum()

fig, ax = plt.subplots()

ax.scatter(groupedby_acc.index, groupedby_acc['No.of.Accidents'])

labels = ax.get_xticklabels()
plt.title('No.of.Accidents in 2019')
plt.xlabel('Month')
plt.ylabel('No_of_accident')

plt.show()
```



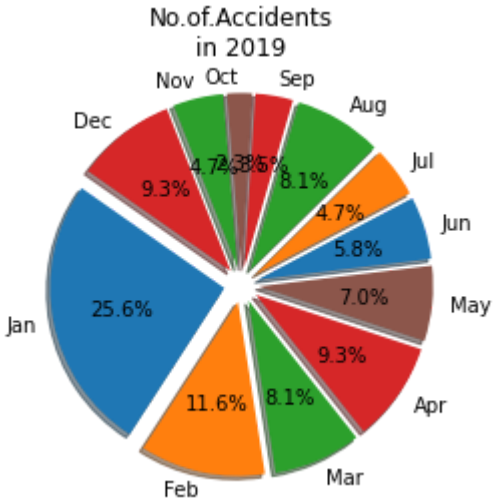
Bar Chart

In [29]:
```
import pandas as pd
from matplotlib import pyplot as plt

df = pd.read_csv('ml_lab1data.csv')

groupedby_acc = df.groupby('Month')[['No.of.Accidents']].sum()

fig, ax = plt.subplots()

ax.bar(groupedby_acc.index, groupedby_acc['No.of.Accidents'])

labels = ax.get_xticklabels()
plt.title('No.of.Accidents in 2019')
plt.xlabel('Month')
plt.ylabel('No_of_accident')

plt.show()
```



Pie Chart

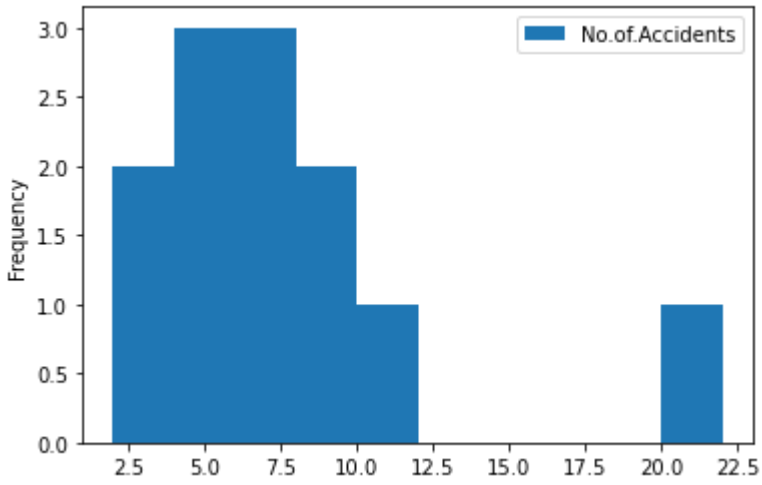
In [117]:
```
import matplotlib.pyplot as plt
import pandas as pd
df =  pd.read_csv('ml_lab1data.csv')
month_data = df["Month"]
acc_data = df["No.of.Accidents"]
colors = ["#1f77b4", "#ff7f0e", "#2ca02c", "#d62728", "#8c564b","#1f77b4", "#ff7f
explode = (0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1)
plt.pie(acc_data, labels=month_data, explode=explode, colors=colors,
autopct='%1.1f%%', shadow=True, startangle=145)
plt.title("No.of.Accidents\n"+"in 2019")
plt.show()
```



Histogram

In [136]:
```
import pandas as pd
from matplotlib import pyplot as plt
df = pd.read_csv('ml_lab1data.csv')
df.plot.hist()
```

Out[136]: <AxesSubplot:ylabel='Frequency'>



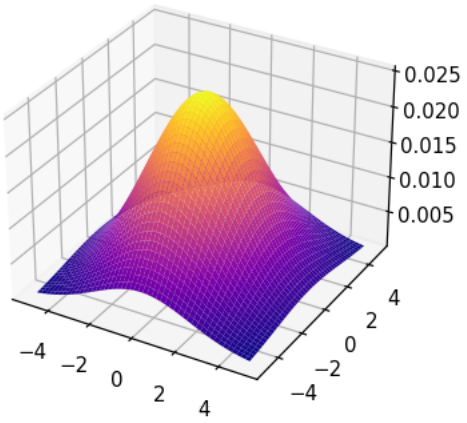
2.Image Visualization

In [6]:
```
%matplotlib inline
import imageio
import matplotlib.pyplot as plt
import matplotlib.cbook

pic=imageio.imread('bhc.png')
plt.figure(figsize=(6,6))
plt.imshow(pic)
plt.axis('off')
```

Out[6]: (-0.5, 699.5, 499.5, -0.5)

In [7]:
```
%matplotlib notebook
import matplotlib.pyplot as plt

from mpl_toolkits.mplot3d import Axes3D

import numpy as np

from scipy.stats import multivariate_normal

X = np.linspace(-5,5,50)

Y = np.linspace(-5,5,50)

X, Y = np.meshgrid(X,Y)

X_mean = 0; Y_mean = 0

X_var = 5; Y_var = 8

pos = np.empty(X.shape+(2,))

pos[:,:,0]=X

pos[:,:,1]=Y

rv = multivariate_normal([X_mean, Y_mean],[[X_var, 0], [0, Y_var]])

fig = plt.figure()

ax = fig.add_subplot(111, projection='3d')

ax.plot_surface(X, Y, rv.pdf(pos), cmap="plasma")

plt.show()
```

```
<IPython.core.display.Javascript object>
```



Video visualization

In [1]:
```
from IPython.display import HTML

HTML("""
    <video width="320" height="240" controls>
        <source src="Jeevamshamayi.mp4" type="video/mp4">
    </video>
""")
```

Out[1]:

Audio visualization

In [54]:
```
!pip install librosa
```

```
Collecting librosa
  Using cached librosa-0.8.0.tar.gz (183 kB)
Collecting audioread>=2.0.0
  Using cached audioread-2.1.9.tar.gz (377 kB)
Requirement already satisfied: numpy>=1.15.0 in c:\users\mahesh\anaconda3\lib
\site-packages (from librosa) (1.19.2)
Requirement already satisfied: scipy>=1.0.0 in c:\users\mahesh\anaconda3\lib
\site-packages (from librosa) (1.6.0)
Requirement already satisfied: scikit-learn!=0.19.0,>=0.14.0 in c:\users\mahe
sh\anaconda3\lib\site-packages (from librosa) (0.23.2)
Requirement already satisfied: joblib>=0.14 in c:\users\mahesh\anaconda3\lib
\site-packages (from librosa) (1.0.0)
Requirement already satisfied: decorator>=3.0.0 in c:\users\mahesh\anaconda3
\lib\site-packages (from librosa) (4.4.2)
Collecting resampy>=0.2.2
  Using cached resampy-0.2.2.tar.gz (323 kB)
Requirement already satisfied: numba>=0.43.0 in c:\users\mahesh\anaconda3\lib
\site-packages (from librosa) (0.51.2)
Collecting soundfile>=0.9.0
```

In [3]:
```
# Load imports
import IPython.display as ipd
import librosa
import librosa.display
import matplotlib.pyplot as plt
```

Audio Player

In [4]:
```
ipd.Audio('Jeevamshamayii.m4a')
```

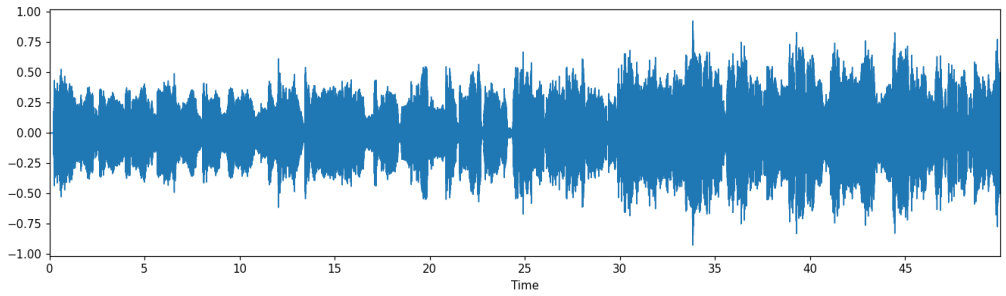
Out[4]:

0:00 / 0:00

spectrogram

In [64]:
```
filename1 = 'Jeevamshamayii.m4a'
plt.figure(figsize=(15,4))
data1,sample_rate1 = librosa.load(filename1, sr=22050, mono=True, offset=0.0, dur
librosa.display.waveplot(data1,sr=sample_rate1, max_points=50000.0, x_axis='time'
```

```
<IPython.core.display.Javascript object>
```



Out[64]:
```
<matplotlib.collections.PolyCollection at 0x1f2e3c7c160>
```

Text visualization

In [11]:
```
!pip install wordcloud
```

```
Collecting wordcloud
  Downloading wordcloud-1.8.1-cp38-cp38-win_amd64.whl (155 kB)
Requirement already satisfied: numpy>=1.6.1 in c:\users\mahesh\anaconda3\lib
\site-packages (from wordcloud) (1.19.2)
Requirement already satisfied: pillow in c:\users\mahesh\anaconda3\lib\site-p
ackages (from wordcloud) (8.1.0)
Requirement already satisfied: matplotlib in c:\users\mahesh\anaconda3\lib\si
te-packages (from wordcloud) (3.3.2)
Requirement already satisfied: certifi>=2020.06.20 in c:\users\mahesh\anacond
a3\lib\site-packages (from matplotlib->wordcloud) (2020.12.5)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in
c:\users\mahesh\anaconda3\lib\site-packages (from matplotlib->wordcloud) (2.
4.7)
Requirement already satisfied: cycler>=0.10 in c:\users\mahesh\anaconda3\lib
\site-packages (from matplotlib->wordcloud) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\mahesh\anaconda3
\lib\site-packages (from matplotlib->wordcloud) (1.3.1)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\mahesh\anacon
da3\lib\site-packages (from matplotlib->wordcloud) (2.8.1)
```

Reference *https://www.geeksforgeeks.org/generating-word-cloud-python/#:~:text=Generating%20Word%20Cloud%20in%20Python.%20Last%20Updated%3A%04-2020.,points%20can%20be%20highlighted%20using%20a%20word%20cloud (https://www.geeksforgeeks.org/generating-word-cloud-python/#:~:text=Generating%20Word%20Cloud%20in%20Python.%20Last%20Updated%3A%04-2020.,points%20can%20be%20highlighted%20using%20a%20word%20cloud).*

Reference data set *https://archive.ics.uci.edu/ml/machine-learning-databases/00380/YouTube-Spam-Collection-v1.zip (https://archive.ics.uci.edu/ml/machine-learning-databases/00380/YouTube-Spam-Collection-v1.zip)*

In [14]:
```
# Python program to generate WordCloud

# importing all necessery modules
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
import pandas as pd

# Reads 'Youtube04-Eminem.csv' file
df = pd.read_csv(r"Youtube04-Eminem.csv", encoding ="latin-1")

comment_words = ''
stopwords = set(STOPWORDS)

# iterate through the csv file
for val in df.CONTENT:
    val = str(val)
    tokens = val.split()
# Converts each token into lowercase
for i in range(len(tokens)):
    tokens[i] = tokens[i].lower()
    comment_words += " ".join(tokens)+" "

wordcloud = WordCloud(width = 800, height = 800,
background_color ='white',
stopwords = stopwords,
min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

```
<IPython.core.display.Javascript object>
```

In [16]:
```
!pip install plotly
```

```
Collecting plotly
  Downloading plotly-4.14.3-py2.py3-none-any.whl (13.2 MB)
Collecting retrying>=1.3.3
  Downloading retrying-1.3.3.tar.gz (10 kB)
Requirement already satisfied: six in c:\users\mahesh\anaconda3\lib\site-packag
es (from plotly) (1.15.0)
Building wheels for collected packages: retrying
  Building wheel for retrying (setup.py): started
  Building wheel for retrying (setup.py): finished with status 'done'
  Created wheel for retrying: filename=retrying-1.3.3-py3-none-any.whl size=114
29 sha256=cab58ae52f4edf04f59a45252096be9e097e53fabee195befe362fb063d8bd76
  Stored in directory: c:\users\mahesh\appdata\local\pip\cache\wheels\c4\a7\48
\0a434133f6d56e878ca511c0e6c38326907c0792f67b476e56
Successfully built retrying
Installing collected packages: retrying, plotly
Successfully installed plotly-4.14.3 retrying-1.3.3
```

Reference ***https://plotly.com/python/bubble-charts/ (https://plotly.com/python/bubble-charts/)***

In [27]:
```
import plotly.graph_objects as go

fig = go.Figure(data=[go.Scatter(
    x=[1, 2, 3, 4], y=[10, 11, 12, 13],
    text=['A<br>size: 10', 'B<br>size: 20', 'C<br>size: 30', 'D<br>size: 40'],
    mode='markers',
    marker=dict(
        color=['rgb(93, 164, 214)', 'rgb(255, 144, 14)',  'rgb(44, 160, 101)', 'r
        size=[10, 20, 30, 40],
    )
)])

fig.show()
```

In [ ]:
```

```

```
import pandas as pd
data = pd.read_csv('data.csv')
print(data.info())
print(data.shape)
(11,2)
print(data.size)
22
print(data.ndim)
2
```

Graph plot:

```
import matplotlib.pyplot as plt.
plt.plot(data['Year'],data['No. of bag sold'])
plt.show()
```

Bar plot:

```
import matplotlib.pyplot as plt.
plt.bar(data['year']:data['No. of bag sold']
plt.show()
```

Scatter plot:

```
import matplotlib.pyplot as plt.
plt.Scatter(data['year'], data['No. of bag sold'])
plt.show()
```

Lab1: Warm Up – Familiarity with Data types and Visualization

Objectives

In this lab, you will get familiarity with downloading, reading, printing properties and visualizing datasets. Also, you will work on various Notebooks such as Google Colab and Azure notebooks.

Learning Outcomes

After completing this lab, you will be able to

- Understand various data formats
- Understand various file formats
- Visualize various kinds of data such as text, images, video and audios
- Get familiarity with Google Colab and Azure notebooks

Step 1: Download the dataset files that belong to the following data formats from internet. The files may belong to any dataset available online.

Step 2: Read these files inside the python code. Some of the file formats cannot be read using default python packages. In this case, explore the python packages suitable for reading the files.

Step 3: Print the properties of the data files such as size, shape, dimensions, etc.

Step 4: Visualize each of these data files using graphs, diagrams, etc.

- Table data visualization: line graph, bar graph, histogram chart, pie chart, scatter plot
- Image visualization: image plot, 3d plot
- Video visualization: video player
- Audio visualization: audio player, spectrogram
- Text visualization: Word cloud, bubble cloud (some more in http://vallandingham.me/textvis-talk/)

1. Tabular, Spreadsheet and Interchange Data Formats

- "Table" — generic tabular data (.dat), "CSV" — comma-separated values (.csv), "TSV" — tab-separated values (.tsv), "ARFF" - Attribute-Relation File Format (.arff) – Read and visualize the data
- "XLS" — Excel spreadsheet (.xls), "XLSX" — Excel 2007 format (.xlsx), "ODS" — OpenDocument spreadsheet (.ods), "SXC" — OpenOffice 1.0 spreadsheet file (.sxc), "DIF" — VisiCalc data interchange format (.dif) – Read and visualize the data
- "JSON" — JavaScript Object Notation (.json), "UBJSON" — Universal Binary JSON (.ubj), "HTML" – Hypertext Markup Language (.html), "XML" - eXtensible Markup Language (.xml) - Read and Parse the data

2. Data File Formats

- PKL – Pickle format, HDF5, Zip, SQL, MAT, NPY, NPZ – Read and display the data

3. Image Data Formats

- JPG, PNG, BMP, TIFF – Read and display the image
- 3D medical Images: DICOM, MHA – Read and display the image

4. Video Data Formats

- MP4, AVI, MPEG – Read and play the video

5. Audio Data Formats

- MP3, MIDI, WAV – Read and play the audio

histogram:-

```
import matplotlib.pyplot as plt
plt.hist(data['No. of bag sold'])
plt.show()
```

Pie chart:-

```
import matplotlib.pyplot as plt
import pandas as pd
df = pd.read_csv('data.csv')
month_data = df["Year"]
acc_data = df["No. of bag sold"]
colors = ["#1f77b4", "#ff7f0e", "#2ca02c",
          "#d62728", "#8c564b", "#1f77b4", "#ff7f0e"]
explode = (0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1)
plt.pie(acc_data, labels=month_data, explode=
    explode, colours=colors, autopct='%1.1f%%',
    shadow = True, startangle = 145)
plt.title("No. of bag sold \n" + "in 2019")
plt.show()
```

6. Text Data Formats

- TXT, PDF, DOC – Read and parse the data

Familiarity: Get familiarity with **Google Colab Notebook** and Microsoft **Azure Notebooks.**

Image Visualization:

```
% matplotlib inline
import Image IO
import matplotlib.pyplot as plt.
import matplotlib.cbook
(-0.5, 699.5, 499.5, -0.5)
%. matplotlib notebook
import matplotlib.pyplot as plt.
from mpl_toolkits.mplot3d import Axes3D
import numpy as np.
from .scipy.stats import multivariate_normal
    X = np.linspace(-5, 5, 150)
    Y = np.linspace(-5, 5, 150)
X, Y = np.meshgrid(X, Y)
```

```
X_mean = 0 ; Y_mean = 0

X_var = 5 ; Y_var = 8

pos = np.empty( X.shape + (2,))

pos[:, :, 0] = X

pos[:, :, 1] = Y

rv = multivariate_normal([X_mean, Y_mean],

        [[X_var, 0], [0, X_var]])

fig = plt.figure()

ax = fig.add_subplot(111, projection = "3d")

ax.plot_surface(X, Y, rv.pdf(pos), cmap = "plasma")

plt.show()

from IPython.display import HTML

HTML("""<video width = "320", height = "240"

controls><source src = "JeevanShamaji.mp4" type =

"video/mp4">

</video>

""")
```

NOTES

```
import IPython.display as ipd.
import librosa
import librosa.display
import matplotlib.pyplot as plt.
ipd.Audio("Jeevamshamayi.mp4")
filename1 = ("Jeevamshamayi.mp4")
plt.figure(figsize('15,4))
data1, sample_rate1 = librosa.load(filename1,
    sr = 22050, mono=true, offset=0.0, duration
librosa.display.waveplot(data1, sr = sample_rate1,
    max_points = 50000.0, x-axis = "time"
```

REPORT

Lab1.Warmup: Familiarity with Data Visualization

In this lab we have learned basic visualization techniques and also working with various data formats, file formats.

Various file formats particularly include image visualization and audio, video visualization.

For me the challenging part is working with Attribute-Relation File Format(.arff) file I got very few answers also I was not able to visualize the data which is in arff file format.

Apart from this arff file format the major struggle is working with 3D medical image format such as DICOM, MHA these file formats are completely a strange to me.

Name:Viviyan Richards W

Roll no:205229133

step 2

In [9]:
```
import pandas as pd
```

In [10]:
```
df = pd.read_csv("pizza.csv")
```

In [11]:
```
df.head()
```

Out[11]:

	age	weight	likepizza
0	50	65	0
1	20	55	1
2	15	40	1
3	70	65	0
4	30	70	1

In [12]:
```
df.shape
```

Out[12]:
```
(6, 3)
```

In [13]:
```
for col in df.columns:
    print(col)
```

```
age
weight
likepizza
```

In [14]:
```
df.columns
```

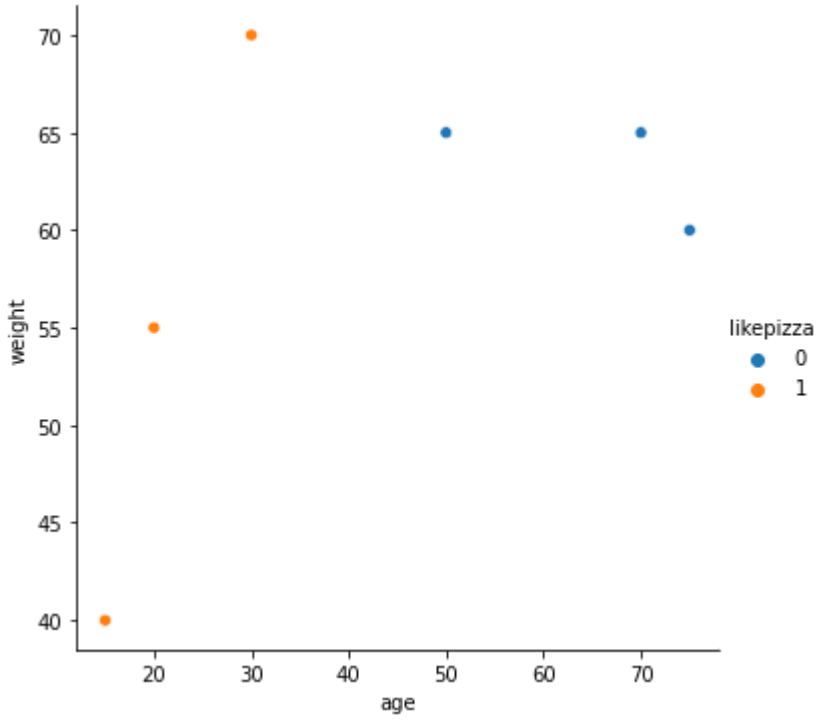
Out[14]:
```
Index(['age', 'weight', 'likepizza'], dtype='object')
```

step:3

In [15]:
```
import seaborn as sns
```

In [16]:
```
sns.relplot(x="age", y="weight", data=df, hue='likepizza')
```

Out[16]: <seaborn.axisgrid.FacetGrid at 0x2b5101f7040>



step:4

In [74]:
```
y=df.likepizza
deat = ['age','weight']
x=df[deat]
```

In [75]:
```
x
```

Out[75]:

	age	weight
0	50	65
1	20	55
2	15	40
3	70	65
4	30	70
5	75	60

In [76]:
```
y
```

Out[76]:
```
0    0
1    1
2    1
3    0
4    1
5    0
Name: likepizza, dtype: int64
```

In [77]:
```
x.dtypes
```

Out[77]:
```
age       int64
weight    int64
dtype: object
```

In [78]:
```
y.dtypes
```

Out[78]:
```
dtype('int64')
```

step 6

In [80]:
```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=2)
model.fit(x,y)
```

Out[80]:
```
KNeighborsClassifier(n_neighbors=2)
```

step 7

In [81]:
```
data = [[20,50]]
```

In [83]:
```
model.predict(data)
```

Out[83]:
```
array([1], dtype=int64)
```

In [84]:
```
data = [[60,60]]
```

In [85]:
```
model.predict(data)
```

Out[85]:
```
array([0], dtype=int64)
```

step 8

In [86]:
```
model2 =KNeighborsClassifier(n_neighbors=3)
model.fit(x,y)
data = [[32,35]]
```

In [29]:
```
model.predict(data)
```

Out[29]:
```
array([1], dtype=int64)
```

In [30]:
```
print(model.predict(data))
```

```
[1]
```

In [31]:
```
print(model.predict(x))
```

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

step 9

In [32]:
```
y_pred = model.predict(x)
def accuracy(actual, pred):
    return sum(actual==pred) / float(actual.shape[0])
print('Accuracy:',accuracy(y,y_pred))
```

```
Accuracy: 1.0
```

step 10

In [34]:
```
from sklearn.metrics import accuracy_score
```

In [35]:
```
print(accuracy_score(y,y_pred))
```

```
1.0
```

step 12

In [54]:
```
p1=pd.read_csv("pizza_test.csv")
```

In [55]:
```
p1.head()
```

Out[55]:

	age	weight	likepizza
0	48	68	1
1	35	45	1
2	15	40	0
3	55	65	0

In [56]:
```
p1.shape
```

Out[56]:
```
(4, 3)
```

In [57]:
```
p1.columns
```

Out[57]:
```
Index(['age', 'weight', 'likepizza'], dtype='object')
```

In [58]:
```
p1.info
```

Out[58]:
```
<bound method DataFrame.info of    age  weight  likepizza
0   48      68          1
1   35      45          1
2   15      40          0
3   55      65          0>
```

In [59]:
```
age_weight=['age','weight']
X=p1[age_weight]
X
```

Out[59]:

	age	weight
0	48	68
1	35	45
2	15	40
3	55	65

In [60]:
```
y=p1.likepizza
y
```

Out[60]:
```
0    1
1    1
2    0
3    0
Name: likepizza, dtype: int64
```

In [61]:
```
knn=KNeighborsClassifier(n_neighbors=2)
knn.fit(X,y)
```

Out[61]:
```
KNeighborsClassifier(n_neighbors=2)
```

In [62]:
```
knn.predict(X)
```

Out[62]:
```
array([0, 0, 0, 0], dtype=int64)
```

In [63]:
```
def accuracy(actual,pred):
    return sum(actual == pred) / float(actual.shape[0])
    return accuracy
```

In [64]:
```
y_pred=knn.predict(X)
```

In [65]:
```
accuracy(y,y_pred)
print("accuracy score: ", accuracy(y,y_pred) )
```

```
accuracy score:  0.5
```

step 13

In [66]:
```
scores = []
for k in range(1,4):
    best = KNeighborsClassifier(n_neighbors=k)
    best.fit(X, y)
    y_predt = best.predict(X)
    acc=accuracy(y,y_predt)
    scores.append((k,acc))
```

In [67]:
```
scores
```

Out[67]:
```
[(1, 1.0), (2, 0.5), (3, 0.5)]
```

step 14

In [68]:
```
from sklearn.metrics import accuracy_score
```

In [69]:
```
accuracy(y,y_pred)
print("accuracy_score: ", accuracy_score(y,y_pred) )
```

```
accuracy_score:  0.5
```

```
from sklearn.neighbors import KNeighborsClassifier.
model = kneighborsclassifier(n_neighbors = 2)
model.fit(x, y)
(model.predict(x))
data = [25, 150]
model.predict([data])
data = [60, 60]
model.predict([data])
models3 = KNeighborsclassifiers(n_neighbors = 3)
models3.fit(x,y)
models3 = kNeighborsclassifier(n_neighbors = 3)
models3.fit(x,y)
(model3.predict(x))
data 2 = [25, 150]
model.3.predict([data 2])
data 3 = [60, 60]
model 3.predict([[data 3]])
y_pred = model 3.predict(x)
y_pred
def .accuracy.(actual, pred):
  return sum(actual == pred)/float(actual.shap[0])
print("Accuracy:", accuracy(y, y_pred))
pizz = pd.read_csv("pizza_test.csv")
pizz.head()
pizz.tail()
pizz.shape()
df = pd.read_csv("pizza_test.csv")
df
df.info()
```

- Will a person who is 25 years with weight 50 kgs like Pizza or not? – The answer should be 1 (ie., YES)
- Will a person who is 60 years with weight 60 kgs like Pizza or not? – The answer should be 0 (ie., NO)

Step.8 [Change n_neighbors = 3]. Now, create new model, perform fit and predict steps. Check results for the above 2 queries. Are they same?

Step9. [Predict on entire dataset]. Now, perform prediction on entire X matrix and store result as y_pred.

Step10. [Accuracy function]. Create a ***function accuracy()*** and returns accuracy. The accuracy function can be defined as follows:

```
def accuracy(actual, pred):
    return sum(actual == pred) / float(actual.shape[0])
```

Step11. [Find accuracy]. Call accuracy() with **y and y_pred** as parameters and print accuracy score. Are you getting score as **1.0** ?

KNN predicted all samples in X correctly. Because, you created KNN model with those values of X and y. But, it may not be the same if you use different samples for testing.

Step12. [Prediction on Test Set]

Using Pandas, import **"pizza_test.csv"** file and print properties such as head(), shape, columns and info.

Using KNN model with n_neighbors=2, that you created previously, perform prediction on X values from pizza_test dataframe. Call accuracy function and print accuracy score. Are you getting a score of **0.5**? That is, our model has predicted 2 samples correctly and two wrongly, out of 4 samples in the test set.

Step13. [Find best value for k]. If you want to improve the accuracy of your model, then you should use the best value k for the nearest neighbors.

Now, let us rerun our KNN for various values of k (k = 1 to 3) and find accuracy scores. Then, we will be able to select the best k for which accuracy score is the highest.

```
scores = list()

for k in range(1, 4):
   create knn model
   perform fit on X and y
   predict test set X
   find accuracy on y values of test set and predicted values
   store k and accuracy as a tuple in scores list
```

When you print scores list, you will get the following output.

```
[(1, 0.5), (2, 0.5), (3, 0.5)]
```

Step14. [accuracy_score function]. Call accuracy_score() function with y_test and y_pred values. You can import as *"from sklearn.metrics import accuracy_score"*.

CONCLUSION

Our accuracy values remain same for all values of k. Because, our dataset is so small. When working on larger datasets, your will find accuracy improving on various values of k.

```
res = df.columns
print(res)

sns.relplot(x="age", y="weight", hue="like pizza",
            data=pizz)
sns.relplot(x='age', y='weight', data=pizz,
            kind="scatter")

rk = pizz.dropna(axis=0)
rk.
y = pizz.likepizza
doubt = ['age', 'weight']
X = pizz[doubt]
X
Y.
X.dtypes
X.dtypes.
from sklearn.neighbors import KNeighborsClassifier.
piz = KNeighborsClassifier(n_neighbors=2)
piz.fit(X,Y)
(piz1.predict(X))
def accuracy(actual, pred):
return sum(actual==pred)/float(actual.
                               shape[0])
```

NOTES

```
y_pred = piz.predict(x)
accuracy(y, y_pred)

scores = []
for k in range(1,4):
    best = kNeighborsClassifier(n_neighbors=k)
    best.fit(x, y)
    best.predict(y)
    y_predict = best.predict(x)
    acc = accuracy(y, y_predict)
    scores.append((k, acc))
scores
from sklearn.metrices import accuracy_score
accuracy_score(y, y_pred)
```

```
import pandas as pd.

import csv.

piz = pd.read_csv("pizza.csv")

piz = head()

piz.tail()

piz.shape

df=pd.read_csv("pizza.csv")

df

df.info()

res = df.columns

print(res)

import pandas as pd

import matplotlib.pyplot as plt.

import seaborn as sns.

import numpy as np.

sns.relplot(x="age", y="weight", hue="like pizza",
            data=piz)

y = piz.like pizza

dat = ['age', 'weight']

X = piz[dat]

X

X

X.dtypes

y.dtypes

pip install sklearn.
```

Lab2. Pizza Liking Prediction using kNN

Objectives

In this lab, you will build a k-Nearest-Neighbour model to predict whether a person will like pizza or not based on his age and weight.

Learning Outcomes

After completing this lab, you will

- Identify features (aka variables – dependent and independent) and create dataset
- Import dataset and understand properties such as size, data types and others
- Use sklearn and instantiate kNN model
- Perform fit and predict operations
- Compute accuracy
- Select the best value for K

Business Use Case

You are a Data Scientist. A local Pizza Restaurant in your city has approached you with the details of its customers such as age and weight and whether they liked their Pizza or not. The restaurant wanted you to help them to develop a system that will predict whether a customer will like their Pizzas given their age and weight. The restaurant has given you the following dataset for assessing your predictive analytics skills.

Step1. [Prepare your dataset]

Using MS Excel, create the following CSV file and save it as, **"pizza.csv"**

age	weight	likePizza
50	65	0
20	55	1
15	40	1
70	65	0
30	70	1
75	60	0

Create another CSV, as below, but with age and weight columns only and save it as, **"pizza_test.csv".**

age	weight	likePizza
48	68	1
35	45	1
15	40	0
55	65	0

Step2. [Import dataset]. Using Pandas, import **"pizza.csv"** file and print properties such as head(), shape, columns and info.

Step3. [Visualize Relationships]. Plot relplot between "age" and "weight", with hue as "likePizza"

Step4. [Prepare X matrix and y vector]. Extract "age" and "weight" columns and store into new dataframe **X**. Similarly, extract "likePizza" column and store into **y**.

Step5. [Examine X and y]. Print X, y, type of X and type of y.

Step6. [Model building]. Create **KNeighborsClassifier(n_neighbors=2)** from **sklearn** and perform **fit** on X and y.

Step7. [Model testing]. Using your KNN model, predict if a person will like Pizza or not.

REPORT

Lab2.Pizza Liking Prediction using kNN

In this lab we have learned how and where to use the k-Nearest-Neighbor classifier.

The major thing is to understand the given data and to identify dependent and independent variables then creating a dataset.

sci-kit learn a popular library which is fully focused on Machine learning algorithms. By the help of this we perform huge math operations just by calling built in functions and libraries in it.

To import knn classifier we use from sklearn.neighbours import kNeighboursClassifier after doing this we create a model to fit and perform prediction.

In [133]:
```
import pandas as pd
import csv
```

In [134]:
```
fuel = pd.read_csv('fuel_data.csv')
fuel
```

Out[134]:

	drivenKM	fuelAmount
0	390.00	3600.0
1	403.00	3705.0
2	396.50	3471.0
3	383.50	3250.5
4	321.10	3263.7
5	391.30	3445.2
6	386.10	3679.0
7	371.80	3744.5
8	404.30	3809.0
9	392.20	3905.0
10	386.43	3874.0
11	395.20	3910.0
12	381.00	4020.7
13	372.00	3622.0
14	397.00	3450.5
15	407.00	4179.0
16	372.40	3454.2
17	375.60	3883.8
18	399.00	4235.9

In [135]:
```
fuel.head()
```

Out[135]:

	drivenKM	fuelAmount
0	390.0	3600.0
1	403.0	3705.0
2	396.5	3471.0
3	383.5	3250.5
4	321.1	3263.7

In [136]:
```
fuel.shape
```

Out[136]: (19, 2)

In [137]:
```
fuel.columns
```

Out[137]:
```
Index(['drivenKM', 'fuelAmount'], dtype='object')
```

In [138]:
```
fuel.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19 entries, 0 to 18
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   drivenKM    19 non-null     float64
 1   fuelAmount  19 non-null     float64
dtypes: float64(2)
memory usage: 432.0 bytes
```

In [139]:
```
fuel.isnull()
```

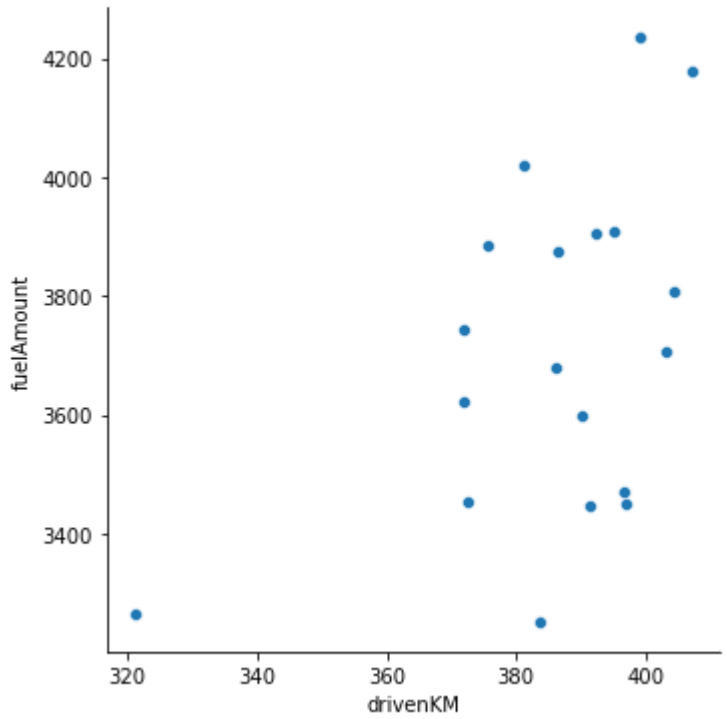
Out[139]:

	drivenKM	fuelAmount
0	False	False
1	False	False
2	False	False
3	False	False
4	False	False
5	False	False
6	False	False
7	False	False
8	False	False
9	False	False
10	False	False
11	False	False
12	False	False
13	False	False
14	False	False
15	False	False
16	False	False
17	False	False
18	False	False

In [140]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

In [141]:
```
sns.relplot(x="drivenKM", y="fuelAmount",data=fuel)
```

Out[141]: <seaborn.axisgrid.FacetGrid at 0x23b5cb31550>



In [142]:
```
data1 = ['drivenKM']
X=fuel[data1]
```

In [143]:
```
data2 = ['fuelAmount']
Y=fuel[data2]
```

In [144]:
```
print(X)
X.dtypes
```

```
    drivenKM
0     390.00
1     403.00
2     396.50
3     383.50
4     321.10
5     391.30
6     386.10
7     371.80
8     404.30
9     392.20
10    386.43
11    395.20
12    381.00
13    372.00
14    397.00
15    407.00
16    372.40
17    375.60
18    399.00
```

Out[144]:
```
drivenKM    float64
dtype: object
```

In [145]:
```
print(Y)
Y.dtypes
y_True=y
```

```
    fuelAmount
0       3600.0
1       3705.0
2       3471.0
3       3250.5
4       3263.7
5       3445.2
6       3679.0
7       3744.5
8       3809.0
9       3905.0
10      3874.0
11      3910.0
12      4020.7
13      3622.0
14      3450.5
15      4179.0
16      3454.2
17      3883.8
18      4235.9
```

In [146]:
```
from sklearn.model_selection import train_test_split
```

In [147]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,train_size=0.8,test_size=
```

In [148]:
```
X_train, X_test, Y_train, Y_test
```

Out[148]:
```
(    drivenKM
 9     392.20
 8     404.30
 5     391.30
 1     403.00
 14    397.00
 4     321.10
 16    372.40
 6     386.10
 7     371.80
 15    407.00
 10    386.43
 13    372.00
 17    375.60
 18    399.00
 3     383.50,
     drivenKM
 11     395.2
 12     381.0
 0      390.0
 2      396.5,
     fuelAmount
 9       3905.0
 8       3809.0
 5       3445.2
 1       3705.0
 14      3450.5
 4       3263.7
 16      3454.2
 6       3679.0
 7       3744.5
 15      4179.0
 10      3874.0
 13      3622.0
 17      3883.8
 18      4235.9
 3       3250.5,
     fuelAmount
 11      3910.0
 12      4020.7
 0       3600.0
 2       3471.0)
```

In [149]:
```
X_train.shape
```

Out[149]:
```
(15, 1)
```

In [150]:
```
X_test.shape
```

Out[150]:
```
(4, 1)
```

In [151]:
```
Y_train.shape
```

Out[151]:
```
(15, 1)
```

In [152]:
```
Y_test.shape
```

Out[152]:
```
(4, 1)
```

In [153]:
```
from sklearn.linear_model import LinearRegression
```

In [154]:
```
lin_reg = LinearRegression()
lin_reg.fit(X_train,Y_train)
```

Out[154]:
```
LinearRegression()
```

In [155]:
```
x=[[800]]
lin_reg.predict(x)
```

Out[155]:
```
array([[6857.09812249]])
```

In [156]:
```
y_pred=lin_reg.predict(X)
y_pred
```

Out[156]:
```
array([[3744.25861846],
       [3842.95840761],
       [3793.60851303],
       [3694.90872388],
       [3221.14973595],
       [3754.12859737],
       [3714.64868171],
       [3606.07891365],
       [3852.82838653],
       [3760.9616597 ],
       [3717.1541379 ],
       [3783.73853412],
       [3675.9279952 ],
       [3607.59737194],
       [3797.40465877],
       [3873.3275735 ],
       [3610.63428853],
       [3634.92962124],
       [3812.58924172]])
```

In [157]:
```
from sklearn.metrics import mean_squared_error, r2_score


mse = mean_squared_error(y_True, y_pred)
mse
```

Out[157]:
```
58987.44292405615
```

In [158]:
```
r2_score(y_True, y_pred)
```

Out[158]:
```
0.21643923225776895
```

In [159]:
```
lin_reg.coef_
```

Out[159]:
```
array([[7.59229147]])
```

In [160]:
```
lin_reg.intercept_
```

Out[160]:
```
array([783.2649439])
```

In [161]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
s_X_train=scaler.fit_transform(X_train)
s_X_train
```

Out[161]:
```
array([[ 0.39182783],
       [ 0.98313698],
       [ 0.34784616],
       [ 0.91960789],
       [ 0.62639675],
       [-3.08272426],
       [-0.57576896],
       [ 0.09372983],
       [-0.60509007],
       [ 1.11508199],
       [ 0.10985644],
       [-0.59531637],
       [-0.41938968],
       [ 0.7241338 ],
       [-0.03332833]])
```

In [162]:
```
s_X_test=scaler.transform(X_test)
s_X_test
```

Out[162]:
```
array([[ 0.5384334 ],
       [-0.15549964],
       [ 0.28431708],
       [ 0.60196248]])
```

In [163]:
```
s_lin_reg = LinearRegression()
s_lin_reg.fit( s_X_train,y_train)
```

Out[163]:
```
LinearRegression()
```

In [164]:
```
s_y_pred=s_lin_reg.predict(s_X_test)
s_y_pred
```

Out[164]:
```
array([[3745.68215321],
       [3676.96914641],
       [3720.51964368],
       [3751.97278059]])
```

In [165]:
```
mean_squared_error(y_test, s_y_pred)
```

Out[165]:
```
67297.78117999973
```

In [166]:
```
r2_score(y_test, s_y_pred)
```

Out[166]:
```
0.0345055936007026
```

In [167]:
```
s_lin_reg.coef_
```

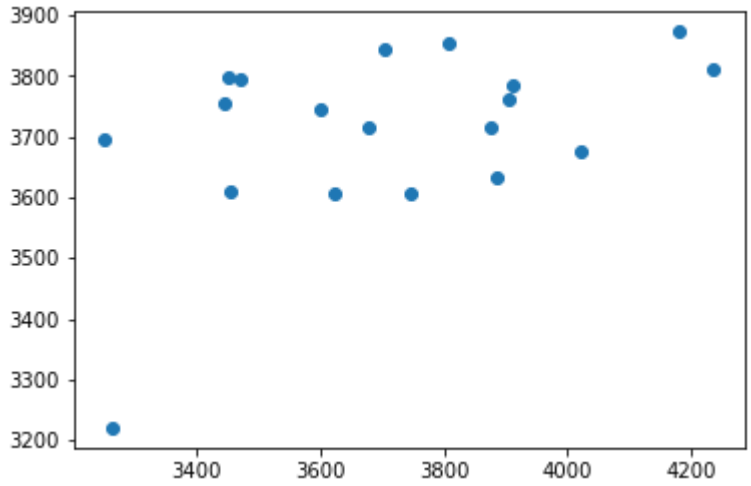
Out[167]:
```
array([[99.01964899]])
```

In [168]:
```
s_lin_reg.intercept_
```

Out[168]:
```
array([3692.36666667])
```

In [169]:
```
import matplotlib.pyplot as plt
import numpy as np

plt.scatter(y_True,y_pred)
```

Out[169]:
```
<matplotlib.collections.PathCollection at 0x23b5cf242b0>
```



In [170]:
```
from sklearn.preprocessing import MinMaxScaler
scalers = MinMaxScaler()
m_X_train=scalers.fit_transform(X_train)
m_X_train
```

Out[170]:
```
array([[0.82770664],
       [0.9685681 ],
       [0.81722934],
       [0.95343423],
       [0.88358556],
       [0.        ],
       [0.59720605],
       [0.75669383],
       [0.59022119],
       [1.        ],
       [0.76053551],
       [0.59254948],
       [0.63445867],
       [0.90686845],
       [0.72642608]])
```

In [171]:
```
m_X_test=scalers.transform(X_test)
m_X_test
```

Out[171]:
```
array([[0.86263097],
       [0.69732247],
       [0.80209546],
       [0.87776484]])
```

In [172]:
```
m_lin_reg = LinearRegression()
m_lin_reg.fit( m_X_train,y_train)
```

Out[172]:
```
LinearRegression()
```

In [173]:
```
m_y_pred=m_lin_reg.predict(m_X_test)
m_y_pred
```

Out[173]:
```
array([[3745.68215321],
       [3676.96914641],
       [3720.51964368],
       [3751.97278059]])
```

In [174]:
```
mean_squared_error(y_test, m_y_pred)
```

Out[174]:
```
67297.7811799998
```

In [175]:
```
r2_score(y_test, m_y_pred)
```

Out[175]:
```
0.03450559360070149
```

In [176]:
```
import numpy as np
from sklearn.neighbors import KNeighborsRegressor
neigh = KNeighborsRegressor(n_neighbors=5)
neigh.fit(X, y)
```

Out[176]:
```
KNeighborsRegressor()
```

In [177]:
```
n_y_pred=neigh.predict(X)
n_y_pred
```

Out[177]:
```
array([[3700.64],
       [3875.88],
       [3794.48],
       [3684.84],
       [3593.64],
       [3746.84],
       [3684.84],
       [3745.04],
       [3875.88],
       [3666.24],
       [3569.74],
       [3794.48],
       [3741.6 ],
       [3745.04],
       [3794.48],
       [3875.88],
       [3745.04],
       [3745.04],
       [3754.48]])
```

In [178]:
```
knn_mse=mean_squared_error(y_True, n_y_pred)
knn_mse
```

Out[178]:
```
70460.30507368421
```

In [179]:
```
r2_score(y_True,n_y_pred)
```

Out[179]:
```
0.06403925984775638
```

In [180]:
```
import numpy as np
from sklearn.linear_model import SGDRegressor
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
# Always scale the input. The most convenient way is to use a pipeline.
reg = make_pipeline(StandardScaler(), SGDRegressor(max_iter=1000, tol=1e-3))
reg.fit(X, y)
```

```
C:\Users\VISSWES\anaconda3\lib\site-packages\sklearn\utils\validation.py:72: Da
taConversionWarning: A column-vector y was passed when a 1d array was expected.
Please change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

Out[180]:
```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('sgdregressor', SGDRegressor())])
```

In [181]:
```
r_y_pred=reg.predict(X)
r_y_pred
```

Out[181]:
```
array([3740.41973845, 3830.0665893 , 3785.24316387, 3695.59631303,
       3265.29142895, 3749.38442353, 3713.52568319, 3614.91414726,
       3839.03127438, 3755.59074398, 3715.80133402, 3776.27847879,
       3678.35653402, 3616.29332958, 3788.69111968, 3857.65023571,
       3619.05169422, 3641.11861136, 3802.48294288])
```

In [182]:
```
sgd_mse=mean_squared_error(y_True,r_y_pred)
sgd_mse
```

Out[182]:
```
58824.133120863604
```

In [183]:
```
r2_score(y_True,r_y_pred)
```

Out[183]:
```
0.218608560989888
```

In [184]:
```
print('LR:',mse)
print('KNNR:',knn_mse)
print('SGDR:',sgd_mse)
```

```
LR: 58987.44292405615
KNNR: 70460.30507368421
SGDR: 58824.133120863604
```

In [ ]:

In [ ]:

lab:-3

Step:-1 import pandas as pd.

import csv.

fuel = pd.read_csv (fuel_data = csv'')

fuel

Step:-2

fuel.tail()

fuel.shape

df=pd.read_csv("fuel_data.csv'')

df

df.info()

df = df.columns

df.

Step:-3 df.isnull()

Step:-4 import pandas as pd.

import matplotlib.pyplot as plt.

import seaborn as sns.

import numpy as np.

Lab3. Fuel Amount Prediction using Linear Regression

Objectives

In this lab, you will build a Linear Regression model to predict the amount to be spent on fuel (ie., diesel) for your car for a road trip.

Learning Outcomes

After completing this lab, you will

- Acquire data and prepare dataset
- Split dataset for training and testing
- Build LR model, perform training, testing and compute MSE and R2 error values
- Apply normalization methods for feature scaling
- Compare performance with KNN Regressor and SGD Regressor

Business Use Case

Say you're planning a road trip to **Goa** with two of your best friends. You start off in **Trichy** and you know it's going to be a ~20h drive. While your friends are in charge of the party operations you're in charge of the all the logistics involved. You have to plan every detail: the schedule, when to stop and where, make sure you get there on time.

So, what's the first thing you do?. You get yourself a blank sheet of paper and start planning!. First item on your checklist? Budget! It's a 20h — approximately 800 kilometers — fun ride, so a total of 20h on the road. The follow-up question: **How much money should I allocate for diesel?**

This is a very important question. You don't want to stop in the middle of the highway and possibly walk a few miles just because you ran out of diesel ! How much money should you allocate for diesel ?.

You approach this problem with a science-oriented mindset, thinking that there must be a way to estimate the amount of money needed, based on the distance you're traveling.

First, you look at some data. You've been laboriously tracking your car's efficiency for the last year — because who doesn't! — so somewhere in your computer, you have stored in an Excel sheet (***fuel_data.csv***) , as shown below.

drivenKM	fuelAmount
390	3600
403	3705
396.5	3471
383.5	3250.5
321.1	3263.7
391.3	3445.2
386.1	3679
371.8	3744.5
404.3	3809
392.2	3905
386.43	3874
395.2	3910
381	4020.7
372	3622
397	3450.5
407	4179
372.4	3454.2

sns.relplot(x="drivenKM", y="fuel Amount",
data=fuel)

Step:5

data1 = ["drivenKM"]

X = fuel[data1]

data2 = ["fuel Amount"]

y = fuel Amount.

Step:6

print(X)

X.dtypes

print(y)

y.dtypes.

Step:7

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split
(X, y, train_size = 0.8, test_size = 0.2)

X_train, X_test, X_train, y_test.

X_train.shape

X_test.shape

y_train.shape

y_test.shape

375.6	3883.8
399	4235.9

Step1. [Prepare your dataset]. Create ***fuel_data.csv*** file as shown above.

Step2. [Import dataset]. Using Pandas, import ***"fuel_data.csv"*** file and print properties such as head(), shape, columns, type and info.

Step3. [Preprocessing]. Check for missing values (Use isnull() method)

Step4. [Visualize Relationships]. Plot ***relplot*** between "**drivenKM**" and "**fuelAmount**".

Step5. [Prepare X matrix and y vector]. Extract "**drivenKM**" column and store into new dataframe **X**. Similarly, extract "**fuelAmount**" and store into **y**.

Step6. [Examine X and y]. Print X, y, type of X and type of y.

Step7. [Split dataset]. Split dataset into 4 parts using ***train_test_split()*** method, such as *X_train, X_test, y_train* and *y_test*. Use **20%** for test size. Later you can play around with this test size. Print the shape of all 4 parts.

Part-I. Linear Regression Baseline Model

Step8. [Build Model]. Create Linear Regression model and train with ***fit()*** using X_train and y_train values.

Step9. [Predict price for 800 KM]. If I need to travel 800 KM, how much do I need to spend on Diesel?. Are you getting this ouput, **array([6905.64571567]). ?**

Step10. [Predict on entire dataset]. Now, perform prediction using entire **X_test** and store result as **y_pred**.

Step11. [Print Mean Squared Error and R2 Error]. Are you getting output **"MSE: 46181.0"**. Also, print values of model parameters: ***coef_*** and ***intercept_*** values.

Part-II. Linear Regression with Scaling using StandardScaler

Step12. [Normalize X_train and X_test values]. Use **StandardScaler**, scale X_train using **fit_transform()** method and X_test using **transform()** method.

Step13. [Build LR model]. Create a new LR model, fit on *scaled X_train* and predict on *scaled X_test*.

Step14. [Print Mean Squared Error and R2 Error]. What is the output?. MSE reduced or not?. Why?.

Step15. [Plot scatter plot]. Display Scatter Plot between actual y (aka ground truth) vs predicted y values. That is, between ***y_test*** and ***y_pred***.

Part-III. Linear Regression with Scaling using MinMaxScaler and Comparison with KNeighborsRegressor and SGDRegressor

Step16. [Repeat with MinmaxScaler]. Repeat scaling using **MinMaxScaler**, LR model creation, fit, predict and error computation steps.

Step17. [Compare KNN Regressor]. Repeat the above steps for **KNeighborsRegressor** model and compare MSE of LR with KNN Regressor.

Step 8:

```
from sklearn.linear_model.import linear
                                Register Regression
model = linear.Regression()
model.fit(x_train, dy_train)
```

Step 9:

```
n = [['800']]
m = model.predict(n)
m.
```

Step 10:

```
y_pred = model.predict(x_test)
y_pred.
```

Step 11:

```
from.sklearn.metrices.import mean_
                        Squared_Error.
from sklearn.metrices.import r2_score
mse_ln = mean_squared_error(y_test, y_pred)
mse_ln
r2 - score(y_test, y_pred)
```

Step18. [Compare SGD Regressor]. Repeat the above steps for **SGDRegressor** model and compare MSE of LR with SGD Regressor.

Step19. [Select best model]. Tabulate MSE values of LR, KNNR and SGDR and select the model with the lowest MSE.

```
model.coef_
model.intercept_
```

Step:12

```
from sklearn.preprocessing import standard scaler.
scaler = StandardScaler()
SS3 = scaler.fit_transform(x_train)
Print(SS3)
SS5 = scaler.transform(x_test)
print(SS5)
```

Step:13:

```
Model 1 = linear Regression()
model.1.fit(SS3, y_train)
S1_S_pred = model 1.predict(SS5)
S1_y_pred
```

Step:14:

```
mean_squared_error(y_test, S1_y_pred)
r2_score(y_test, S1_y_pred)
```

Step:15:

```
plt.scatter(y_test, y_pred)
```

Step: 16

```
from sklearn.preprocessing import minmax_scalar

mm_scalar = minmaxscalar()

mm_ss = mm_scalar.fit_transform(x_train)

mm_ss

mm_sst = mm_scalar.transform(x_test)

mm_sst

model2 = linear Regression()

model2.fit(mm_ss, y_train)

mms_y_pred = model2.predict(mm_sst)

mms_y_pred

mean_squared_error(y_test, mms_y_pred)

r2_score(y_test, mms_y_pred)
```

Step 17:

```
from sklearn.neighbours import KNeighborsRegressor

m_neig = kNeighbours Regressor(n_neighbours = 5)

m_neig.fit(x, y)

m1_y_pred = m_neig.predict(x)

m1_y_pred

mse = mean_squared_error(y, m1_y_pred)

mse

rg_score(y, m1_y_pred
```

NOTES

Step 15

```
from sklearn.linear_model import SGDRegressor
from sklearn.pipeline import make_pipeline
r = make_pipeline(standardScaler(), SGDRegressor(
        (max_iter=1000, tol=1e-3))
r.fit(x,y)
re_y_pred = r.predict(x)
re_y_pred
mse3 = mean_squared_error(y, re_y_pred)
mse3
r2_score(y, re_y_pred)
```

Step 16

```
print("SR Model", mse_1[1])
print("KNR model", mse)
print("SGDR model", mse3)
```

REPORT

Lab3.Fuel Amount Prediction using Linear Regression

In this lab we have build a Linear Regression model to predict the amount to be spent on fuel.

As usual first we need to understand the data and check their properties by using shape, data type, column name, null values.

Then after understanding the data will split the data to train the machine learning model and to perform prediction on test data sklearn's library called train test split helps to split the dataset with necessary parameters testsize=0.3 or trainsize=0.6 similarly.

Also understood how to apply normalization methods for feature scaling then finally we are comparing the performance with KNN regressor and SGD regressor. Each type of regressor, classifier performs different methods to make prediction, as a data scientist or data engineer we need to choose the right algorithm and right classifiers which gives the best accuracy based on the problem we have.

Lab #4: House Price Prediction using LR with Regularization

Step1. [Import dataset]. Using Pandas, import "Ames_House_Sales_Cropped.csv" file and print properties such as head, shape, columns, dtype, info and value_counts

In [1]:

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

In [30]:

```
df = pd.read_csv('Ames_House_Sales_Cropped.csv')
```

In [31]:

```
df
```

Out[31]:

	BldgType	CentralAir	1stFlrSF	2ndFlrSF	3SsnPorch	BedroomAbvGr	BsmtFinSF1	Bsmt
0	1Fam	Y	856.0	854.0	0.0	3	706.0	
1	1Fam	Y	1262.0	0.0	0.0	3	978.0	
2	1Fam	Y	920.0	866.0	0.0	3	486.0	
3	1Fam	Y	961.0	756.0	0.0	3	216.0	
4	1Fam	Y	1145.0	1053.0	0.0	4	655.0	
...	...	...	...	...	...	...	...	
1374	1Fam	Y	953.0	694.0	0.0	3	0.0	
1375	1Fam	Y	2073.0	0.0	0.0	3	790.0	
1376	1Fam	Y	1188.0	1152.0	0.0	4	275.0	
1377	1Fam	Y	1078.0	0.0	0.0	2	49.0	
1378	1Fam	Y	1256.0	0.0	0.0	3	830.0	

1379 rows × 39 columns

In [5]:

```
df.head()
```

Out[5]:

	BldgType	CentralAir	1stFlrSF	2ndFlrSF	3SsnPorch	BedroomAbvGr	BsmtFinSF1	BsmtFin
0	1Fam	Y	856.0	854.0	0.0	3	706.0	
1	1Fam	Y	1262.0	0.0	0.0	3	978.0	
2	1Fam	Y	920.0	866.0	0.0	3	486.0	
3	1Fam	Y	961.0	756.0	0.0	3	216.0	
4	1Fam	Y	1145.0	1053.0	0.0	4	655.0	

5 rows × 39 columns

In [6]:

```
df.shape
```

Out[6]:

```
(1379, 39)
```

In [7]:

```
df.columns
```

Out[7]:

```
Index(['BldgType', 'CentralAir', '1stFlrSF', '2ndFlrSF', '3SsnPorch',
       'BedroomAbvGr', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtFullBath',
       'BsmtHalfBath', 'BsmtUnfSF', 'EnclosedPorch', 'Fireplaces', 'FullBat
h',
       'GarageArea', 'GarageCars', 'GarageYrBlt', 'GrLivArea', 'HalfBath',
       'KitchenAbvGr', 'LotArea', 'LotFrontage', 'LowQualFinSF', 'MSSubClas
s',
       'MasVnrArea', 'MiscVal', 'MoSold', 'OpenPorchSF', 'OverallCond',
       'OverallQual', 'PoolArea', 'ScreenPorch', 'TotRmsAbvGrd', 'TotalBsmtS
F',
       'WoodDeckSF', 'YearBuilt', 'YearRemodAdd', 'YrSold', 'SalePrice'],
      dtype='object')
```

In [10]:

```
df.dtypes
```

Out[10]:

```
BldgType          object
CentralAir        object
1stFlrSF         float64
2ndFlrSF         float64
3SsnPorch        float64
BedroomAbvGr       int64
BsmtFinSF1       float64
BsmtFinSF2       float64
BsmtFullBath       int64
BsmtHalfBath       int64
BsmtUnfSF        float64
EnclosedPorch    float64
Fireplaces         int64
FullBath           int64
GarageArea       float64
GarageCars         int64
GarageYrBlt      float64
GrLivArea        float64
HalfBath           int64
KitchenAbvGr       int64
LotArea          float64
LotFrontage      float64
LowQualFinSF     float64
MSSubClass         int64
MasVnrArea       float64
MiscVal          float64
MoSold             int64
OpenPorchSF      float64
OverallCond        int64
OverallQual        int64
PoolArea         float64
ScreenPorch      float64
TotRmsAbvGrd       int64
TotalBsmtSF      float64
WoodDeckSF       float64
YearBuilt          int64
YearRemodAdd       int64
YrSold             int64
SalePrice        float64
dtype: object
```

In [12]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1379 entries, 0 to 1378
Data columns (total 39 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   BldgType       1379 non-null   object 
 1   CentralAir     1379 non-null   object 
 2   1stFlrSF       1379 non-null   float64
 3   2ndFlrSF       1379 non-null   float64
 4   3SsnPorch      1379 non-null   float64
 5   BedroomAbvGr   1379 non-null   int64  
 6   BsmtFinSF1     1379 non-null   float64
 7   BsmtFinSF2     1379 non-null   float64
 8   BsmtFullBath   1379 non-null   int64  
 9   BsmtHalfBath   1379 non-null   int64  
 10  BsmtUnfSF      1379 non-null   float64
 11  EnclosedPorch  1379 non-null   float64
 12  Fireplaces     1379 non-null   int64  
 13  FullBath       1379 non-null   int64  
 14  GarageArea     1379 non-null   float64
 15  GarageCars     1379 non-null   int64  
 16  GarageYrBlt    1379 non-null   float64
 17  GrLivArea      1379 non-null   float64
 18  HalfBath       1379 non-null   int64  
 19  KitchenAbvGr   1379 non-null   int64  
 20  LotArea        1379 non-null   float64
 21  LotFrontage    1379 non-null   float64
 22  LowQualFinSF   1379 non-null   float64
 23  MSSubClass     1379 non-null   int64  
 24  MasVnrArea     1379 non-null   float64
 25  MiscVal        1379 non-null   float64
 26  MoSold         1379 non-null   int64  
 27  OpenPorchSF    1379 non-null   float64
 28  OverallCond    1379 non-null   int64  
 29  OverallQual    1379 non-null   int64  
 30  PoolArea       1379 non-null   float64
 31  ScreenPorch    1379 non-null   float64
 32  TotRmsAbvGrd   1379 non-null   int64  
 33  TotalBsmtSF    1379 non-null   float64
 34  WoodDeckSF     1379 non-null   float64
 35  YearBuilt      1379 non-null   int64  
 36  YearRemodAdd   1379 non-null   int64  
 37  YrSold         1379 non-null   int64  
 38  SalePrice      1379 non-null   float64
dtypes: float64(21), int64(16), object(2)
memory usage: 420.3+ KB
```

In [13]:

```
df.value_counts
```

Out[13]:

```
<bound method DataFrame.value_counts of      BldgType CentralAir  1stFlrSF
2ndFlrSF  3SsnPorch  BedroomAbvGr  \
0        1Fam          Y     856.0     854.0        0.0            3
1        1Fam          Y    1262.0       0.0        0.0            3
2        1Fam          Y     920.0     866.0        0.0            3
3        1Fam          Y     961.0     756.0        0.0            3
4        1Fam          Y    1145.0    1053.0        0.0            4
...       ...        ...       ...       ...        ...          ...
1374     1Fam          Y     953.0     694.0        0.0            3
1375     1Fam          Y    2073.0       0.0        0.0            3
1376     1Fam          Y    1188.0    1152.0        0.0            4
1377     1Fam          Y    1078.0       0.0        0.0            2
1378     1Fam          Y    1256.0       0.0        0.0            3

      BsmtFinSF1  BsmtFinSF2  BsmtFullBath  BsmtHalfBath  ...  OverallQual
\
0          706.0         0.0             1             0  ...            7
1          978.0         0.0             0             1  ...            6
2          486.0         0.0             1             0  ...            7
3          216.0         0.0             1             0  ...            7
4          655.0         0.0             1             0  ...            8
...          ...         ...           ...           ...  ...          ...
1374         0.0         0.0             0             0  ...            6
1375       790.0       163.0             1             0  ...            6
1376       275.0         0.0             0             0  ...            7
1377        49.0      1029.0             1             0  ...            5
1378       830.0       290.0             1             0  ...            5

      PoolArea  ScreenPorch  TotRmsAbvGrd  TotalBsmtSF  WoodDeckSF  YearBuil
t  \
0          0.0          0.0             8        856.0         0.0       200
3
1          0.0          0.0             6       1262.0       298.0       197
6
2          0.0          0.0             6        920.0         0.0       200
1
3          0.0          0.0             7        756.0         0.0       191
5
4          0.0          0.0             9       1145.0       192.0       200
0
...        ...          ...           ...          ...         ...
...
1374       0.0          0.0             7        953.0         0.0       199
9
1375       0.0          0.0             7       1542.0       349.0       197
8
1376       0.0          0.0             9       1152.0         0.0       194
1
1377       0.0          0.0             5       1078.0       366.0       195
0
1378       0.0          0.0             6       1256.0       736.0       196
5

      YearRemodAdd  YrSold  SalePrice
0             2003    2008   208500.0
```

```
1             1976    2007   181500.0
2             2002    2008   223500.0
3             1970    2006   140000.0
4             2000    2008   250000.0
...            ...     ...        ...
1374          2000    2007   175000.0
1375          1988    2010   210000.0
1376          2006    2010   266500.0
1377          1996    2010   142125.0
1378          1965    2008   147500.0

[1379 rows x 39 columns]>
```

Step2. [Predict Sale Price without Categorical features]

1.Drop both categorical features – BldgType and CentralAir (USE drop() and pop() methods)

2.Prepare X matrix (36 feature columns) and y vector (ie., SalePrice column)

3.Split dataset for training and testing as X_train, X_test, y_train, y_test (use 25% test size).

4.Create LinearRegression model, fit on training set and predict on test set

5.Compute Mean Squared Error (MSE) on actual values and predicted values (you will get output as 1474827326.0).

In [14]:

```
df.pop('CentralAir')
```

Out[14]:

```
0       Y
1       Y
2       Y
3       Y
4       Y
       ..
1374    Y
1375    Y
1376    Y
1377    Y
1378    Y
Name: CentralAir, Length: 1379, dtype: object
```

In [16]:

```
ndf=df.drop(['BldgType'],axis=1)
ndf
```

Out[16]:

	1stFlrSF	2ndFlrSF	3SsnPorch	BedroomAbvGr	BsmtFinSF1	BsmtFinSF2	BsmtFullBath
0	856.0	854.0	0.0	3	706.0	0.0	1
1	1262.0	0.0	0.0	3	978.0	0.0	0
2	920.0	866.0	0.0	3	486.0	0.0	1
3	961.0	756.0	0.0	3	216.0	0.0	1
4	1145.0	1053.0	0.0	4	655.0	0.0	1
...	...	...	...	...	...	...	...
1374	953.0	694.0	0.0	3	0.0	0.0	0
1375	2073.0	0.0	0.0	3	790.0	163.0	1
1376	1188.0	1152.0	0.0	4	275.0	0.0	0
1377	1078.0	0.0	0.0	2	49.0	1029.0	1
1378	1256.0	0.0	0.0	3	830.0	290.0	1

1379 rows × 37 columns

In [18]:

```
y=ndf['SalePrice']
y
```

Out[18]:

```
0       208500.0
1       181500.0
2       223500.0
3       140000.0
4       250000.0
          ...
1374    175000.0
1375    210000.0
1376    266500.0
1377    142125.0
1378    147500.0
Name: SalePrice, Length: 1379, dtype: float64
```

In [19]:

```
col=['1stFlrSF', '2ndFlrSF', '3SsnPorch','BedroomAbvGr','BsmtFinSF1','BsmtFinSF2','BsmtFull
X=ndf[col]
X
```

Out[19]:

	1stFlrSF	2ndFlrSF	3SsnPorch	BedroomAbvGr	BsmtFinSF1	BsmtFinSF2	BsmtFullBath
0	856.0	854.0	0.0	3	706.0	0.0	1
1	1262.0	0.0	0.0	3	978.0	0.0	0
2	920.0	866.0	0.0	3	486.0	0.0	1
3	961.0	756.0	0.0	3	216.0	0.0	1
4	1145.0	1053.0	0.0	4	655.0	0.0	1
...	...	...	...	...	...	...	...
1374	953.0	694.0	0.0	3	0.0	0.0	0
1375	2073.0	0.0	0.0	3	790.0	163.0	1
1376	1188.0	1152.0	0.0	4	275.0	0.0	0
1377	1078.0	0.0	0.0	2	49.0	1029.0	1
1378	1256.0	0.0	0.0	3	830.0	290.0	1

1379 rows × 36 columns

In [20]:

```
from sklearn.model_selection import train_test_split
```

In [21]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.75,test_size=0.25)
```

In [22]:

```
from sklearn.linear_model import LinearRegression
```

In [23]:

```
model = LinearRegression()
model.fit(X_train,y_train)
```

Out[23]:

```
LinearRegression()
```

In [24]:

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

Out[24]:

```
array([278935.46760284, 184809.97301153, 182520.43414718,  76483.10947771,
       150887.34597605, 131258.56301645, 148585.24002343, 219097.29021401,
       111773.72913835, 154881.76229039, 256457.65115608, 318666.12035276,
       159561.7926296 , 169107.86682563, 284670.76042656, 181162.50230694,
       227278.48539688, 114165.89801476, 294931.91089851, 293854.41405179,
       111435.02196622, 382585.58559473, 165973.92154629, 169983.25590793,
       233764.73014481, 256793.62698861, 141571.18375276, 236487.34960355,
       110407.21426591, 256140.69565805, 116395.91328469, 246533.22979261,
       297638.63909922,  67305.0554531 ,  99905.83719632, 127802.47702706,
       164970.2581652 , 107724.12361544, 295939.4760221 , 191015.64411747,
       118704.79917791, 123437.69393744, 174063.85709086, 289498.87116956,
       224265.54740821, 324401.30417851, 255494.61362604, 239541.04302594,
        55786.42143874, 147705.02545961, 108706.2471813 , 179216.94913311,
       176248.36836102,  94056.17368155, 154939.16090248, 155810.5399734 ,
       121532.36318911, 183547.46384496, 189454.96089569,  89208.20961407,
       352481.06371935, 179664.02452966, 123126.97142585, 213608.94529321,
       175539.14858387, 143633.83226572, 253326.60653631, 117216.66854023,
       203411.64478104, 194715.49377294, 229472.71664896, 115651.09160892,
       178241.14827662, 236505.51781256, 219307.57126792, 204502.34677502,
       280994.60010451,  80073.90667113, 181434.71323535, 345032.82419352,
       236500.06566161, 212514.13302149, 192407.95486381, 142416.55271327,
       259468.7204445 , 120750.14329522, 226366.87209352,  96780.09952695,
       127163.17111758, 273574.37672692, 108517.67103149, 256173.98073332,
       228790.59367182,  88910.50049467, 107165.22214306, 217146.62770519,
        58964.35975604, 105063.33080949, 320544.61249786,  86138.74731506,
       116839.9388387 , 105712.01945374,  88230.90550046, 107652.68223998,
       287090.16939842, 229465.77535541, 158494.22780582, 204864.27385771,
       239609.37680689, 110442.75382776, 122941.92801304, 196560.69862189,
       281001.20763735, 220768.98319365, 175182.06264764, 177842.14883526,
       144528.98007098,  89249.33171787, 244389.89418398, 216270.93679401,
       248220.02700825, 138666.51927131, 154560.04755823, 221112.55156718,
       188362.59021471, 307203.62609492, 189581.32178303, 183577.46991967,
       225368.77356014, 174519.66453824, 101554.978302  , 293361.03611142,
       154012.24355422, 215330.16029933,  61812.07114214, 179234.77485403,
       225272.72744757, 177373.59763724, 151409.50180686, 111112.42636732,
       199536.361842  , 156681.59243024, 251611.1346998 , 134180.23992892,
       358140.1693967 , 154567.52731523, 204091.40490632, 121956.1236706 ,
       125826.70580346,  84907.0843445 , 196986.38415147,  85904.43755063,
       150656.93797411, 118650.29179834, 195818.871078  , 175546.47631959,
       170931.96314849, 208790.97470297,  67379.38866588, 187151.29277591,
       186719.96869974, 134660.74721073, 105614.18483508,  79513.26298969,
       124566.70915102, 122550.68752771, 206222.27957192, 207601.22965933,
       146159.01052126, 152363.80695361, 145495.0606997 , 173907.24721372,
       152747.5053473 ,  49317.18217061, 213494.73086963,  44082.42523023,
       225190.28027351, 258281.36606756, 311000.95634462, 106095.4649713 ,
       271811.12874892, 159944.11359786, 226011.83194734, 174465.73073269,
       123728.0160802 , 181138.90667533, 111038.64757558,  89889.85094022,
       123564.48045469, 250749.37890992, 226613.92673256, 210434.24370752,
       207660.39092568, 292946.37527581, 334214.98921072, 133817.05041267,
       111125.04462064, 139295.3398275 , 102650.43024674, 194340.182154  ,
       122023.34009653, 276061.24429208, 269946.5997433 , 196227.72886623,
       264731.11726338,  89901.09609902, 304188.42231002, 375521.98855508,
       111007.44643434, 350583.50095325, 342750.39394037, 110615.57193354,
       145956.36088972, 172249.53317499, 232427.35295181, 152857.99881604,
```

```
       119692.98800232, 116730.19848642, 204857.36322241, 244716.63533763,
       121656.14394599,  80838.88733147, 196484.62439457, 143926.78270525,
       106731.16943813, 194536.2272504 , 158474.12778809, 222614.21867115,
       101549.90251678, 147893.15229663, 320820.25505894,  54731.41824227,
       174055.67716066, 107403.96861738,  20317.15304097, 197697.74888904,
        87979.6495762 , 245044.00505482, 137239.8607613 , 178076.30627442,
       202934.38670694, 208592.35841259, 182895.66157271, 173665.02493717,
       187506.78653531, 191522.33185992, 195263.35443505, 262094.61033262,
       161424.46981284, 156789.32222922, 103129.22065772, 219923.35629208,
       159162.70875961, 259900.44388485, 192866.20003135, 312664.28514545,
       188000.48911174, 183052.42071648, 118131.16319897, 105606.97184016,
       174562.14609183,  98671.51317334, 110028.85776064, 247856.70997153,
        96614.73642864, 204370.74006904, 199319.43354319, 276169.70745246,
       137136.46481863, 189550.95222814, 127295.63955043, 258359.51237967,
       127024.88223935, 155902.77325132, 144286.75347553, 332376.7892343 ,
       243087.52260861, 210040.44140568, 114455.28504463, 202191.37155523,
       339030.87673504,  90609.78663929, 305497.84068171, 184070.6180781 ,
       287059.69348526,  79610.54167769, 294912.04178084,  66993.51568241,
       200053.34479378, 122697.83324424, 178671.12407925, 124541.03500813,
       256665.37732096, 233212.15027303,  94485.1448137 , 177833.98011138,
       138853.91780848, 183255.58095821, 154441.37726845, 152409.41675422,
       191323.66654094, 251395.00733515, 308279.40016699, 141436.09277925,
       270164.76638018, 221497.30954601, 147840.06207748, 295243.09480318,
       224575.0239214 , 367455.91468599, 230812.55543866, 105480.92918135,
       120875.02893366, 153886.45181622, 123686.71604039, 344316.10144485,
       223039.5741509 , 117427.00403197, 388531.26435691, 255607.40556548,
       244320.6800141 , 273483.63005768, 222328.99257133, 308105.45493609,
       283690.0908143 , 226343.71533342, 136431.10961914, 150610.65617637,
        44036.2995959 , 153481.99615241, 169535.67163367, 187240.21625236,
       291409.819297  ,  87683.66518659, 232799.80069605, 160632.59383701,
       138829.37808239, 166849.87559837,  95769.8914938 , 219251.88582997,
       136377.60902172, 147907.69764383, 294279.56152334, 194101.09245019,
       127886.62776903])
```

In [25]:

```
from sklearn.metrics import mean_squared_error
```

In [26]:

```
mean_squared_error(y_test,y_pred)
```

Out[26]:

```
930012256.3673348
```

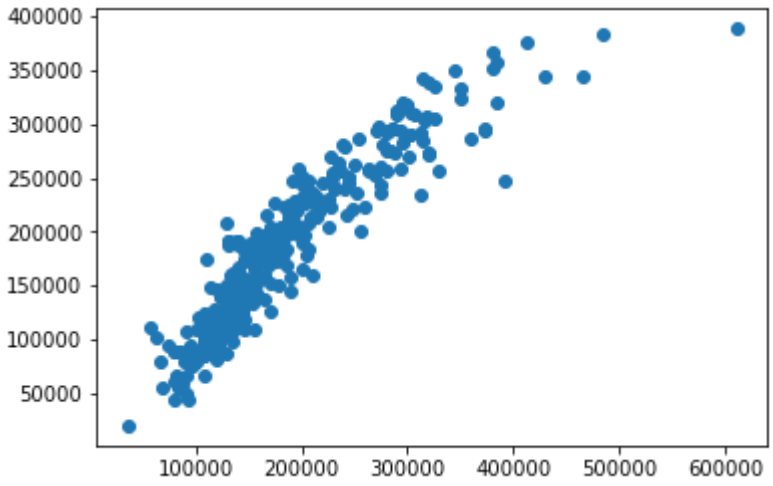
Step3. [Create Scatter Plot]. Plot Scatterplot between y_test and y_pred.

In [27]:

```
plt.scatter(y_test,y_pred)
```

Out[27]:

```
<matplotlib.collections.PathCollection at 0x3e09ca1130>
```



Step4. [Encode Categorical columns]. Using get_dummies() method, perform one hot encoding on the two categorical columns, BldgType and CentralAir.Now, you will get 5 columns for BldgType variable and 2 columns for CentralAir column. So, now you have 43 independent variables and 1 dependent variable

In [32]:

```
encode_df=pd.get_dummies(df, columns=["CentralAir","BldgType"])
encode_df.head()
```

Out[32]:

	1stFlrSF	2ndFlrSF	3SsnPorch	BedroomAbvGr	BsmtFinSF1	BsmtFinSF2	BsmtFullBath	Bs
0	856.0	854.0	0.0	3	706.0	0.0	1	
1	1262.0	0.0	0.0	3	978.0	0.0	0	
2	920.0	866.0	0.0	3	486.0	0.0	1	
3	961.0	756.0	0.0	3	216.0	0.0	1	
4	1145.0	1053.0	0.0	4	655.0	0.0	1	

5 rows × 44 columns

In [34]:

```
encode_df.shape
```

Out[34]:

```
(1379, 44)
```

In [35]:

```
encode_df.columns
```

Out[35]:

```
Index(['1stFlrSF', '2ndFlrSF', '3SsnPorch', 'BedroomAbvGr', 'BsmtFinSF1',
       'BsmtFinSF2', 'BsmtFullBath', 'BsmtHalfBath', 'BsmtUnfSF',
       'EnclosedPorch', 'Fireplaces', 'FullBath', 'GarageArea', 'GarageCar
s',
       'GarageYrBlt', 'GrLivArea', 'HalfBath', 'KitchenAbvGr', 'LotArea',
       'LotFrontage', 'LowQualFinSF', 'MSSubClass', 'MasVnrArea', 'MiscVal',
       'MoSold', 'OpenPorchSF', 'OverallCond', 'OverallQual', 'PoolArea',
       'ScreenPorch', 'TotRmsAbvGrd', 'TotalBsmtSF', 'WoodDeckSF', 'YearBuil
t',
       'YearRemodAdd', 'YrSold', 'SalePrice', 'CentralAir_N', 'CentralAir_
Y',
       'BldgType_1Fam', 'BldgType_2fmCon', 'BldgType_Duplex', 'BldgType_Twnh
s',
       'BldgType_TwnhsE'],
      dtype='object')
```

Step5. [Predict Sale Price with Categorical features]

In [37]:

```
en_y=encode_df['SalePrice']
en_y
```

Out[37]:

```
0       208500.0
1       181500.0
2       223500.0
3       140000.0
4       250000.0
          ...
1374    175000.0
1375    210000.0
1376    266500.0
1377    142125.0
1378    147500.0
Name: SalePrice, Length: 1379, dtype: float64
```

In [38]:

```
ecol=['1stFlrSF', '2ndFlrSF', '3SsnPorch', 'BedroomAbvGr', 'BsmtFinSF1',
       'BsmtFinSF2', 'BsmtFullBath', 'BsmtHalfBath', 'BsmtUnfSF',
       'EnclosedPorch', 'Fireplaces', 'FullBath', 'GarageArea', 'GarageCars',
       'GarageYrBlt', 'GrLivArea', 'HalfBath', 'KitchenAbvGr', 'LotArea',
       'LotFrontage', 'LowQualFinSF', 'MSSubClass', 'MasVnrArea', 'MiscVal',
       'MoSold', 'OpenPorchSF', 'OverallCond', 'OverallQual', 'PoolArea',
       'ScreenPorch', 'TotRmsAbvGrd', 'TotalBsmtSF', 'WoodDeckSF', 'YearBuilt',
       'YearRemodAdd', 'YrSold', 'CentralAir_N', 'CentralAir_Y',
       'BldgType_1Fam', 'BldgType_2fmCon', 'BldgType_Duplex', 'BldgType_Twnhs',
       'BldgType_TwnhsE']
en_X=encode_df[ecol]
en_X
```

Out[38]:

	1stFlrSF	2ndFlrSF	3SsnPorch	BedroomAbvGr	BsmtFinSF1	BsmtFinSF2	BsmtFullBath
0	856.0	854.0	0.0	3	706.0	0.0	1
1	1262.0	0.0	0.0	3	978.0	0.0	0
2	920.0	866.0	0.0	3	486.0	0.0	1
3	961.0	756.0	0.0	3	216.0	0.0	1
4	1145.0	1053.0	0.0	4	655.0	0.0	1
...	...	...	...	...	...	...	...
1374	953.0	694.0	0.0	3	0.0	0.0	0
1375	2073.0	0.0	0.0	3	790.0	163.0	1
1376	1188.0	1152.0	0.0	4	275.0	0.0	0
1377	1078.0	0.0	0.0	2	49.0	1029.0	1
1378	1256.0	0.0	0.0	3	830.0	290.0	1

1379 rows × 43 columns

In [42]:

```
en_X_train, en_X_test, en_y_train, en_y_test = train_test_split(en_X,en_y, train_size=0.75,
```

In [44]:

```
model1 = LinearRegression()
model1.fit(en_X_train,en_y_train)
```

Out[44]:

```
LinearRegression()
```

In [46]:

```
en_y_pred=model1.predict(en_X_test)
en_y_pred
```

Out[46]:

```
array([164570.2289465 ,  85752.90331142, 198303.44446022, 122475.14440184,
       175345.78182408, 242830.9535424 , 327131.8650892 , 208850.75498362,
       182132.06296091,  63457.73994568, 154016.77303419, 271424.75748695,
       196168.36447424, 231896.15634764,  99949.01255692, 199586.65283043,
       294455.29422805, 237034.98692393, 156399.80969278, 143322.19881947,
        87317.2057708 , 459342.96310705, 180678.56647699,  80595.77914091,
       150473.52639588, 163119.60771014,  87932.58954817, 298702.16262193,
       139136.14996624, 122356.71597519, 253724.88904318, 124943.48205879,
       190378.91666766, 144311.69637699, 138626.02445074, 108390.1027098 ,
       233825.57182049, 321069.91502648, 152596.17234961, 126562.81992625,
       178895.51749365, 174107.6608674 , 192718.60711014,  84271.1131178 ,
       190926.80720668, 351883.95006538, 237798.33729765, 194988.74848599,
        67864.93144598, 148773.78149566, 141287.95243988, 123734.33873651,
       217729.45043704, 264523.39376933, 284496.30996522, 128633.42861504,
       317463.52519012, 229500.59769751, 216913.34511063, 307023.56295031,
       358046.85973153,  88894.0337679 , 213448.6569423 , 151371.96447342,
       238996.23694776, 206439.35207547, 204300.08235509, 220515.23746768,
       179526.34220759,  97765.62725349, 133030.81385702, 121228.16797878,
       214547.60810556, 163777.79551537, 164400.69495282, 120648.22075953,
       200289.83660161, 530827.98834513, 181721.08012888, 139279.91703074,
       211403.97993066,  88370.23134691,  56223.93985288, 101568.38551049,
        42264.51151882, 269839.9568285 , 219235.93693798, 215559.95896189,
       126180.17980655,  94017.76423989, 186571.38998992, 239285.74298627,
       141136.12202831, 100748.65571204, 234996.84810377, 173317.89471206,
       181170.946485  , 249499.12608691,  59117.53960148, 229825.43138893,
       119229.96389029, 205049.89031161, 289237.16371655, 376713.79583967,
       235314.99706438, 181872.51623   , 131801.04010263,  81714.62196234,
       210450.63983055, 196657.14311654, 116418.2029669 , 157203.17235198,
       141407.83746433, 278946.26986854, 120653.05708992, 147431.71254596,
       120051.95320744, 357597.60225624, 147189.71484067, 143672.2215596 ,
        61735.64050934, 102990.92664158, 343747.05725358, 309335.75077198,
        95077.1881864 , 193859.12199213, 115282.82098695, 164363.93263394,
       299418.50596946, 160799.72506617, 237317.14924853, 150426.28776088,
       141741.9803877 ,  80971.65674589, 116287.05395   , 199189.16365547,
       221398.20911546, 164624.5967068 , 185917.00418407, 232085.78361981,
       161510.37140786, 104607.19634645, 200311.61620566, 193889.93887375,
       360335.63024868, 246880.79734329, 205218.08855951, 149324.66818697,
       219012.52290642, 167950.75652282, 145145.31856183, 213907.37040049,
       166208.00694456, 177565.36744714, 213638.54963519, 128934.89419079,
       127262.06873791, 125732.53616288, 208350.8422771 , 187367.98102733,
       151979.80533747,  98238.80563602, 216376.11908759, 172434.74819286,
       219380.06611483, 169934.88709872, 119439.25910028,  94943.44404374,
       194718.96343604, 119048.86428917, 223908.48489293, 371007.58160088,
       131761.77339371, 202862.19185752, 167584.64261554, 164880.4508151 ,
       135912.95156555, 204349.79412468, 191707.2384559 , 166312.47307931,
       248073.37082883, 158225.96622289, 208273.00886696, 206677.62501901,
       145066.05374636, 188972.2930174 , 112066.39047524,  60843.20451136,
       233862.34059841, 227048.34860413, 236280.16982428, 202861.74018479,
       256347.47173762, 163072.37056444, 208215.47237552, 158121.8268503 ,
       138950.11023881, 131493.60318191, 135309.51482775, 153401.6418395 ,
        87988.96611115, 164203.71884155,  77495.00725865, 190624.2880739 ,
       169614.69937832, 214544.41436752, 158770.76775524, 172081.73067651,
        72404.36280403, 134968.67610532, 121233.70422819, 106284.16747329,
```

```
178255.45274396, 111941.93036186, 214003.21483718, 118547.46238145,
212332.22284787, 167744.85468928, 106327.02745505, 260835.23902291,
207737.0688117 , 240112.93783816, 162717.39329454,  74013.00523018,
214061.17644297, 201153.36902766, 205434.63928939, 204243.08249187,
138793.30501713, 140952.80163748, 191512.24334499, 152215.21020833,
324085.39591175, 304774.23081278, 190512.0628138 , 148811.52086832,
183278.8260974 , 190898.89420528,  91427.2383567 , 240968.26030391,
185624.17527773, 109241.47600847, 120318.3714745 , 180293.28625808,
185887.05318141, 147999.96663853, 203640.95270351, 207262.65897148,
146553.60081434, 225114.56488878, 195969.54558455, 135946.68422957,
 98360.86043366, 145093.48293363, 187527.90880649, 112485.22475333,
162337.18285397, 137396.86809542, 107782.38128734, 200885.0785942 ,
186191.74170784, 246205.62704462, 236007.79504838, 109664.72894365,
 66069.691295  , 271856.0319398 , 252559.63167735, 131192.23890725,
203706.62006734, 112676.08277595, 134454.5979042 , 122227.56150572,
225246.88731473, 207800.75783298, 276095.60530002, 134212.74190231,
158840.30722338, 161414.7159675 , 247093.81102185, 125407.70093713,
140731.92957038, 285092.65610423, 122166.41527203, 265957.16664687,
141846.17715973, 215257.29075487, 133644.67328412, 150444.80270245,
138159.96247511, 370790.96568457, 190143.1932688 , 138562.74477412,
157486.7882223 , 206893.81926977, 193463.29137986, 162321.36289893,
247387.66849631, 176097.83285155, 319862.93484745, 108892.79206792,
157281.06435892, 114875.28570174, 150046.14802086, 224016.84426732,
182743.95014861, 184259.86747379, 216961.78153856, 217129.89206972,
150042.28914975, 183894.29966913, 241653.96996239, 185373.50085041,
142737.94707988, 213872.65242864, 227772.66374861, 371458.8533591 ,
159852.55997952, 105446.77692945,  74753.49864747, 218084.73229221,
115766.14039221, 124870.49571699, 158277.63592713,  42046.97387378,
298295.97719517, 113869.82743718, 311823.56766763, 150483.58747027,
113422.79568058, 110758.54387485, 281329.77955246, 168222.64224303,
186504.6443444 , 118499.73767561, 118439.53660929, 159798.94608422,
285467.65714161, 184380.8672137 , 429972.49494309, 157974.11370633,
198929.95549095, 229184.4784674 , 339430.23265359, 188218.7187889 ,
155268.47972511])
```

In [47]:

```
mean_squared_error(en_y_test,en_y_pred)
```

Out[47]:

```
1603931409.178651
```

Step6.[Normalize using StandardScaler and Predict Sale Price]

In [48]:

```
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
```

In [49]:

```
s= scale.fit_transform(en_X_train)
s
```

Out[49]:

```
array([[-0.75507716,  1.18098615, -0.10770338, ..., -0.17580466,
        -0.16683226, -0.28761522],
       [ 0.84436732, -0.81547136, -0.10770338, ..., -0.17580466,
        -0.16683226,  3.4768675 ],
       [ 4.34791239, -0.81547136, -0.10770338, ..., -0.17580466,
        -0.16683226, -0.28761522],
       ...,
       [-0.77030997, -0.81547136, -0.10770338, ..., -0.17580466,
        -0.16683226, -0.28761522],
       [ 2.26101815, -0.81547136, -0.10770338, ..., -0.17580466,
        -0.16683226, -0.28761522],
       [ 0.72758249, -0.81547136, -0.10770338, ..., -0.17580466,
        -0.16683226, -0.28761522]])
```

In [50]:

```
s1 = scale.transform (en_X_test)
s1
```

Out[50]:

```
array([[ 1.37751549, -0.81547136, -0.10770338, ...,  5.68813139,
        -0.16683226, -0.28761522],
       [ 0.24267154,  0.23565892, -0.10770338, ..., -0.17580466,
        -0.16683226, -0.28761522],
       [-0.98356923,  1.0544825 , -0.10770338, ..., -0.17580466,
        -0.16683226, -0.28761522],
       ...,
       [ 1.68724918, -0.81547136, -0.10770338, ..., -0.17580466,
        -0.16683226, -0.28761522],
       [-0.83124118,  1.15798549, -0.10770338, ..., -0.17580466,
        -0.16683226, -0.28761522],
       [ 1.49937792, -0.81547136, -0.10770338, ...,  5.68813139,
        -0.16683226, -0.28761522]])
```

In [51]:

```
model2 = LinearRegression()
model2.fit(s,en_y_train)
```

Out[51]:

```
LinearRegression()
```

In [52]:

```
se_y_pred = model2.predict(s1)
se_y_pred
```

Out[52]:

```
array([164570.22894655,  85752.90331148, 198303.4444603 , 122475.14440177,
       175345.78182411, 242830.95354235, 327131.86508911, 208850.75498374,
       182132.06296088,  63457.73994563, 154016.77303418, 271424.75748691,
       196168.36447425, 231896.15634765,  99949.01255693, 199586.65283044,
       294455.29422812, 237034.98692395, 156399.80969275, 143322.19881946,
        87317.20577081, 459342.96310711, 180678.56647711,  80595.7791409 ,
       150473.5263959 , 163119.60771012,  87932.58954817, 298702.16262202,
       139136.14996625, 122356.71597514, 253724.88904314, 124943.48205877,
       190378.91666769, 144311.69637702, 138626.0244507 , 108390.10270977,
       233825.57182049, 321069.91502646, 152596.17234961, 126562.81992627,
       178895.51749359, 174107.66086738, 192718.60711018,  84271.11311782,
       190926.80720662, 351883.95006535, 237798.33729765, 194988.748486  ,
        67864.93144594, 148773.78149566, 141287.95243988, 123734.33873654,
       217729.45043705, 264523.39376926, 284496.30996528, 128633.4286151 ,
       317463.52519014, 229500.59769751, 216913.34511065, 307023.5629503 ,
       358046.85973159,  88894.03376787, 213448.65694224, 151371.96447341,
       238996.23694776, 206439.35207551, 204300.0823551 , 220515.2374677 ,
       179526.34220757,  97765.6272535 , 133030.81385704, 121228.16797873,
       214547.60810557, 163777.7955153 , 164400.69495282, 120648.22075952,
       200289.83660161, 530827.98834524, 181721.08012888, 139279.91703071,
       211403.97993059,  88370.23134689,  56223.93985289, 101568.38551036,
        42264.51151872, 269839.95682852, 219235.93693798, 215559.95896183,
       126180.17980651,  94017.76423984, 186571.38998992, 239285.74298627,
       141136.12202832, 100748.65571206, 234996.84810376, 173317.89471207,
       181170.94648492, 249499.12608694,  59117.53960138, 229825.43138894,
       119229.96389044, 205049.8903116 , 289237.16371656, 376713.79583962,
       235314.9970644 , 181872.51622998, 131801.04010258,  81714.6219623 ,
       210450.63983049, 196657.14311654, 116418.20296697, 157203.17235194,
       141407.83746423, 278946.26986855, 120653.05708991, 147431.71254586,
       120051.95320748, 357597.60225625, 147189.71484064, 143672.22155962,
        61735.64050933, 102990.9266416 , 343747.05725362, 309335.75077205,
        95077.18818639, 193859.12199209, 115282.82098686, 164363.93263394,
       299418.5059694 , 160799.72506624, 237317.14924855, 150426.28776082,
       141741.98038769,  80971.65674594, 116287.05394993, 199189.16365541,
       221398.20911547, 164624.59670678, 185917.00418407, 232085.78361985,
       161510.37140789, 104607.1963465 , 200311.61620573, 193889.93887367,
       360335.63024878, 246880.79734332, 205218.08855951, 149324.66818695,
       219012.52290643, 167950.75652274, 145145.3185618 , 213907.37040049,
       166208.00694456, 177565.36744706, 213638.54963519, 128934.89419081,
       127262.06873794, 125732.53616292, 208350.84227703, 187367.98102726,
       151979.8053375 ,  98238.80563596, 216376.11908756, 172434.74819292,
       219380.06611489, 169934.88709875, 119439.25910027,  94943.44404374,
       194718.96343606, 119048.86428917, 223908.48489295, 371007.58160084,
       131761.7733937 , 202862.19185749, 167584.64261553, 164880.4508151 ,
       135912.95156556, 204349.79412469, 191707.23845592, 166312.47307932,
       248073.37082873, 158225.96622294, 208273.00886694, 206677.62501894,
       145066.05374631, 188972.29301732, 112066.39047525,  60843.20451138,
       233862.34059848, 227048.34860413, 236280.16982431, 202861.74018481,
       256347.4717376 , 163072.37056451, 208215.47237559, 158121.82685026,
       138950.11023881, 131493.60318192, 135309.51482778, 153401.6418395 ,
        87988.96611119, 164203.71884157,  77495.00725866, 190624.28807396,
       169614.69937832, 214544.41436747, 158770.76775523, 172081.73067662,
        72404.36280403, 134968.67610533, 121233.70422815, 106284.16747337,
       178255.45274396, 111941.930362  , 214003.2148373 , 118547.46238146,
```

```
212332.22284787, 167744.85468928, 106327.02745505, 260835.23902289,
207737.06881171, 240112.93783827, 162717.39329451,  74013.00523018,
214061.17644298, 201153.36902759, 205434.63928942, 204243.08249182,
138793.3050171 , 140952.80163746, 191512.24334502, 152215.21020836,
324085.39591178, 304774.23081285, 190512.06281379, 148811.52086828,
183278.82609735, 190898.89420526,  91427.23835665, 240968.26030388,
185624.17527773, 109241.47600849, 120318.37147449, 180293.28625808,
185887.05318141, 147999.96663851, 203640.95270348, 207262.6589715 ,
146553.60081428, 225114.56488868, 195969.54558458, 135946.68422952,
 98360.86043365, 145093.48293363, 187527.90880647, 112485.22475333,
162337.18285387, 137396.86809537, 107782.38128735, 200885.0785942 ,
186191.74170786, 246205.62704466, 236007.79504849, 109664.72894361,
 66069.691295  , 271856.03193978, 252559.63167736, 131192.23890726,
203706.62006731, 112676.08277586, 134454.59790426, 122227.56150567,
225246.88731479, 207800.75783312, 276095.60529998, 134212.74190233,
158840.30722344, 161414.71596746, 247093.81102187, 125407.70093715,
140731.92957045, 285092.65610426, 122166.41527204, 265957.1666469 ,
141846.17715971, 215257.29075479, 133644.67328416, 150444.80270244,
138159.96247507, 370790.96568459, 190143.19326877, 138562.74477407,
157486.7882223 , 206893.81926978, 193463.29137984, 162321.36289896,
247387.66849623, 176097.83285157, 319862.93484749, 108892.79206788,
157281.06435888, 114875.28570176, 150046.14802088, 224016.84426723,
182743.95014858, 184259.86747375, 216961.78153849, 217129.89206965,
150042.28914979, 183894.29966913, 241653.96996241, 185373.50085041,
142737.94707987, 213872.65242873, 227772.66374871, 371458.85335927,
159852.55997952, 105446.77692943,  74753.49864743, 218084.73229234,
115766.14039216, 124870.49571701, 158277.6359271 ,  42046.97387374,
298295.97719524, 113869.82743721, 311823.5676677 , 150483.58747027,
113422.7956805 , 110758.54387486, 281329.77955245, 168222.64224302,
186504.64434444, 118499.73767559, 118439.5366092 , 159798.94608423,
285467.65714171, 184380.86721368, 429972.49494319, 157974.11370633,
198929.95549095, 229184.47846748, 339430.23265358, 188218.71878896,
155268.47972515])
```

In [53]:

```
mean_squared_error(en_y_test,se_y_pred)
```

Out[53]:

```
1603931409.178804
```

Step7. [Normalize using MinMaxScaler and Predict Sale Price]

In [55]:

```
from sklearn.preprocessing import MinMaxScaler
m_scaler = MinMaxScaler()
```

In [57]:

```
ms = m_scaler.fit_transform(en_X_train)
ms1 = m_scaler.transform(en_X_test)
```

In [58]:

```
model3 =LinearRegression()
model3.fit(ms,en_y_train)
```

Out[58]:

```
LinearRegression()
```

In [60]:

```
me_y_pred = model3.predict(ms1)
me_y_pred
```

Out[60]:

```
array([164570.22894655,  85752.90331148, 198303.4444603 , 122475.14440177,
       175345.78182411, 242830.95354235, 327131.86508911, 208850.75498374,
       182132.06296088,  63457.73994563, 154016.77303418, 271424.75748691,
       196168.36447425, 231896.15634765,  99949.01255693, 199586.65283044,
       294455.29422812, 237034.98692395, 156399.80969275, 143322.19881946,
        87317.20577081, 459342.96310711, 180678.56647711,  80595.7791409 ,
       150473.5263959 , 163119.60771012,  87932.58954817, 298702.16262202,
       139136.14996625, 122356.71597514, 253724.88904314, 124943.48205877,
       190378.91666769, 144311.69637702, 138626.0244507 , 108390.10270977,
       233825.57182049, 321069.91502646, 152596.17234961, 126562.81992627,
       178895.51749359, 174107.66086738, 192718.60711018,  84271.11311782,
       190926.80720662, 351883.95006535, 237798.33729765, 194988.748486  ,
        67864.93144594, 148773.78149566, 141287.95243988, 123734.33873654,
       217729.45043706, 264523.39376926, 284496.30996528, 128633.4286151 ,
       317463.52519014, 229500.59769751, 216913.34511065, 307023.5629503 ,
       358046.85973159,  88894.03376787, 213448.65694224, 151371.96447341,
       238996.23694776, 206439.35207551, 204300.0823551 , 220515.2374677 ,
       179526.34220757,  97765.6272535 , 133030.81385704, 121228.16797873,
       214547.60810557, 163777.7955153 , 164400.69495282, 120648.22075952,
       200289.83660161, 530827.98834524, 181721.08012888, 139279.91703071,
       211403.97993059,  88370.23134689,  56223.93985289, 101568.38551036,
        42264.51151872, 269839.95682852, 219235.93693798, 215559.95896183,
       126180.17980651,  94017.76423985, 186571.38998992, 239285.74298627,
       141136.12202832, 100748.65571206, 234996.84810376, 173317.89471207,
       181170.94648492, 249499.12608694,  59117.53960138, 229825.43138894,
       119229.96389044, 205049.8903116 , 289237.16371656, 376713.79583962,
       235314.9970644 , 181872.51622998, 131801.04010258,  81714.6219623 ,
       210450.63983049, 196657.14311654, 116418.20296697, 157203.17235194,
       141407.83746423, 278946.26986855, 120653.05708991, 147431.71254586,
       120051.95320748, 357597.60225625, 147189.71484064, 143672.22155962,
        61735.64050933, 102990.9266416 , 343747.05725362, 309335.75077205,
        95077.18818639, 193859.12199209, 115282.82098686, 164363.93263394,
       299418.5059694 , 160799.72506624, 237317.14924855, 150426.28776082,
       141741.98038769,  80971.65674594, 116287.05394993, 199189.16365541,
       221398.20911546, 164624.59670678, 185917.00418407, 232085.78361985,
       161510.37140789, 104607.1963465 , 200311.61620573, 193889.93887367,
       360335.63024878, 246880.79734332, 205218.08855951, 149324.66818695,
       219012.52290643, 167950.75652274, 145145.3185618 , 213907.37040049,
       166208.00694456, 177565.36744706, 213638.54963519, 128934.89419081,
       127262.06873794, 125732.53616292, 208350.84227703, 187367.98102726,
       151979.8053375 ,  98238.80563596, 216376.11908756, 172434.74819292,
       219380.06611489, 169934.88709875, 119439.25910027,  94943.44404374,
       194718.96343606, 119048.86428917, 223908.48489295, 371007.58160084,
       131761.7733937 , 202862.19185749, 167584.64261553, 164880.4508151 ,
       135912.95156556, 204349.79412469, 191707.23845592, 166312.47307932,
       248073.37082873, 158225.96622294, 208273.00886694, 206677.62501894,
       145066.05374631, 188972.29301731, 112066.39047525,  60843.20451138,
       233862.34059848, 227048.34860413, 236280.16982431, 202861.74018481,
       256347.4717376 , 163072.37056451, 208215.47237559, 158121.82685026,
       138950.11023881, 131493.60318193, 135309.51482778, 153401.6418395 ,
        87988.96611119, 164203.71884157,  77495.00725866, 190624.28807396,
       169614.69937832, 214544.41436747, 158770.76775523, 172081.73067662,
        72404.36280403, 134968.67610533, 121233.70422815, 106284.16747337,
       178255.45274396, 111941.930362  , 214003.2148373 , 118547.46238146,
```

```
       212332.22284787, 167744.85468928, 106327.02745505, 260835.23902289,
       207737.06881171, 240112.93783827, 162717.39329451,  74013.00523018,
       214061.17644298, 201153.36902759, 205434.63928942, 204243.08249182,
       138793.3050171 , 140952.80163746, 191512.24334502, 152215.21020836,
       324085.39591178, 304774.23081285, 190512.06281379, 148811.52086828,
       183278.82609735, 190898.89420526,  91427.23835665, 240968.26030387,
       185624.17527773, 109241.47600849, 120318.37147449, 180293.28625808,
       185887.05318141, 147999.96663851, 203640.95270348, 207262.6589715 ,
       146553.60081428, 225114.56488868, 195969.54558458, 135946.68422952,
        98360.86043365, 145093.48293363, 187527.90880647, 112485.22475333,
       162337.18285387, 137396.86809537, 107782.38128735, 200885.0785942 ,
       186191.74170786, 246205.62704466, 236007.79504849, 109664.72894361,
        66069.691295  , 271856.03193978, 252559.63167735, 131192.23890726,
       203706.62006731, 112676.08277586, 134454.59790426, 122227.56150567,
       225246.88731479, 207800.75783312, 276095.60529998, 134212.74190233,
       158840.30722344, 161414.71596746, 247093.81102187, 125407.70093715,
       140731.92957045, 285092.65610426, 122166.41527204, 265957.1666469 ,
       141846.17715971, 215257.29075479, 133644.67328416, 150444.80270244,
       138159.96247507, 370790.96568459, 190143.19326877, 138562.74477407,
       157486.7882223 , 206893.81926978, 193463.29137984, 162321.36289896,
       247387.66849623, 176097.83285157, 319862.93484749, 108892.79206788,
       157281.06435888, 114875.28570176, 150046.14802088, 224016.84426723,
       182743.95014858, 184259.86747375, 216961.78153849, 217129.89206965,
       150042.28914979, 183894.29966913, 241653.96996241, 185373.50085041,
       142737.94707987, 213872.65242873, 227772.66374871, 371458.85335927,
       159852.55997952, 105446.77692943,  74753.49864743, 218084.73229234,
       115766.14039216, 124870.49571701, 158277.6359271 ,  42046.97387374,
       298295.97719524, 113869.82743721, 311823.5676677 , 150483.58747027,
       113422.7956805 , 110758.54387486, 281329.77955245, 168222.64224302,
       186504.64434444, 118499.73767559, 118439.5366092 , 159798.94608423,
       285467.65714171, 184380.86721368, 429972.49494319, 157974.11370633,
       198929.95549095, 229184.47846748, 339430.23265358, 188218.71878896,
       155268.47972515])
```

In [61]:

```
mean_squared_error(en_y_test,me_y_pred)
```

Out[61]:

```
1603931409.1788046
```

Step8. [Predict using SGD Regressor]¶

In [62]:

```
from sklearn.linear_model import SGDRegressor
from sklearn.pipeline import make_pipeline
```

In [63]:

```
SGD = make_pipeline(StandardScaler(), SGDRegressor(max_iter=1000, tol=1e-3))
SGD.fit(X, y)
```

Out[63]:

```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('sgdregressor', SGDRegressor())])
```

In [64]:

```
r_y_pred = SGD.predict(X)
r_y_pred
```

Out[64]:

```
array([234288.73879777, 194490.33616557, 228853.39973738, ...,
       227979.03201414, 122838.6043447 , 145512.54381131])
```

In [65]:

```
mean_squared_error(y,r_y_pred)
```

Out[65]:

```
1236182786.527191
```

Step8. [Predict using Ridge Regression]

In [66]:

```
from sklearn.linear_model import Ridge
```

In [69]:

```
RidgeCV = Ridge(alpha=1.0)
RidgeCV.fit(s,en_y_train)
ridge_y_pred = RidgeCV.predict(s)
ridge_y_pred
```

Out[69]:

```
array([201846.8910729 , 254209.54024493, 338463.77172917, ...,
        88246.41853185, 352706.2962025 , 210331.1763083 ])
```

In [70]:

```
mean_squared_error(en_y_train,ridge_y_pred)
```

Out[70]:

```
1065862697.5487247
```

Step8. [Predict using Lasso Regression]

In [71]:

```
from sklearn.linear_model import Lasso
```

In [72]:

```
LassoCV = Lasso(alpha=1.0)
LassoCV.fit(s,en_y_train)
```

```
C:\Users\Admin\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_
descent.py:529: ConvergenceWarning: Objective did not converge. You might wa
nt to increase the number of iterations. Duality gap: 188735847371.2207, tol
erance: 631484515.8030617
  model = cd_fast.enet_coordinate_descent(
```

Out[72]:

```
Lasso()
```

In [74]:

```
lasso_y_pred = LassoCV.predict(s)
lasso_y_pred
```

Out[74]:

```
array([201847.50954454, 254242.82719469, 338702.94581108, ...,
        88327.65667808, 352770.5563025 , 210188.32752228])
```

In [75]:

```
mean_squared_error(en_y_train,lasso_y_pred)
```

Out[75]:

```
1065858586.2635835
```

Step9.[RMSE]. Print Root Mean Squared Error values (use numpy.sqrt() method) as below and compare error values.¶

In [76]:

```
from math import sqrt
```

In [77]:

```
print("RMSE without one hot encoding:  ",sqrt(mean_squared_error(y_test,y_pred)))
print("RMSE with one hot encoding:  ",sqrt(mean_squared_error(en_y_test,en_y_pred)))
print("RMSE with one and Standard Scaling:  ",sqrt(mean_squared_error(en_y_test,se_y_pred))
print("RMSE with one and MinMax Scaling:  ",sqrt(mean_squared_error(en_y_test,me_y_pred)))
print("RMSE of SGDRegressor with one and Standard Scaler:  ",sqrt(mean_squared_error(y,r_y_
print("RMSE of RigdCV with one and Standard Scaler:  ",sqrt(mean_squared_error(en_y_train,r
print("RMSE of LassoCV with one and Standard Scaler:  ",sqrt(mean_squared_error(en_y_train,
```

```
RMSE without one hot encoding:   30496.102314350515
RMSE with one hot encoding:   40049.11246430626
RMSE with one and Standard Scaling:   40049.11246430817
RMSE with one and MinMax Scaling:   40049.11246430818
RMSE of SGDRegressor with one and Standard Scaler:   35159.39115694683
RMSE of RigdCV with one and Standard Scaler:   32647.552703820307
RMSE of LassoCV with one and Standard Scaler:   32647.489739083823
```

step: 1

```
import pandas as pd
import csv.
house = pd.read_csv("Ames_House_sales_Cropped.csv")
house
house.head()
house.shape.
df = pd.read_csv("Ames_House_sales_Cropped.csv")
df
d = df.columns
d
df.Pnl(00)
df.dtypes.value_counts
```

step: 3

```
.house.drop(['Bldg Type'],axis=1)
df.pop('Central Air')
```

Lab4. House Price Prediction using LR with Regularization

Objectives

In this lab, you will learn how to build Regression Models to predict the Sales Price of houses using Ames, Iowa dataset.

Learning Outcomes

After completing this lab, you will be able to

- Understand data and build baseline LR model
- Apply One Hot Encoding to categorical features and rebuild LR model
- Apply scaling using StandardScaler and MinMaxScaler and rebuild LR model
- Compare the performance of LR model with SGD Regressor, RidgeCV and LassoCV
- Compute RMSE for all models.

Business Use Case

Ames house sales price dataset contains past sales price details of 1379 homes from Ames, USA. This dataset has 79 feature columns (independent variables) and the dependent variable, sale price ("**SalePrice**"). There are three different types: integers (int64), floats (float64), and strings (object, categoricals).

In this lab, you will use a cropped dataset where we have removed all object columns, except two object columns – building type ("**BldgType**") and centralized air conditioning ("**CentralAir**"). Therefore, it contains only 38 independent variables.

With the help of the regression model you will build on this reduced dataset, we can predict the sale price of a house given the values for all independent variables.

Step1. [Import dataset]. Using Pandas, import "**Ames_House_Sales_Cropped.csv**" file and print properties such as head, shape, columns, dtype, info and value_counts.

Step2. [Predict Sale Price without Categorical features].

- Drop both categorical features – BldgType and CentralAir (USE drop() and pop() methods)
- Prepare X matrix (36 feature columns) and y vector (ie., SalePrice column)
- Split dataset for training and testing as X_train, X_test, y_train, y_test (use **25%** test size).
- Create LinearRegression model, fit on training set and predict on test set
- Compute Mean Squared Error (MSE) on actual values and predicted values (you will get output as 1474827326.0).

Step3. [Create Scatter Plot]. Plot Scatterplot between **y_test** and **y_pred**.

Step4. [Encode Categorical columns]. Using **get_dummies()** method, perform one hot encoding on the two categorical columns, *BldgType* and *CentralAir*. Now, you will get 5 columns for BldgType variable and 2 columns for CentralAir column. So, now you have 43 independent variables and 1 dependent variable.

Step5. [Predict Sale Price with Categorical features]

- Prepare X matrix (43 feature columns) and y vector (ie., SalePrice column)
- Split dataset for training and testing
- Create LinearRegression model, fit on training set and predict on test set
- Compute Mean Squared Error on actual values and predicted values (you will get output as 1461036570.0).

Step6. [Normalize using StandardScaler and Predict Sale Price]

- Using StandardScaler, perform **fit_transform()** on X_train and **transform()** on X_test matrix that you already splitted.

```
new_df = df.drop(['Bldg Type'], axis=1)
new_df.
y = new_df['Sales Price']
y
inp = ['1st FLrSF', '2ndFlrSF', '3SSn porch',
       'Bed roomAbvGr', 'BsmtFinSF1']
x = new_df[inp]
x
from sklearn.model selection import train_test_split.
x_train, x_test, y_train, y_test = train_test_split(x, y, train_size)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
from sklearn.linear_model import linear Regression
```

- Create a new Linear Regression model, fit on *scaled X_train* and *y_train* and predict on *scaled X_test*.
- Compute Mean Squared Error (MSE) on actual values and predicted values (you will get output as 1461036570.0).

Step7. [Normalize using MinMaxScaler and Predict Sale Price]

- Repeat Step6 using MinMaxScaler
- Mean Squared Error will be: 1461036570.0

Step8. [Predict using SGD Regressor]

- Use scaled X_train and X_test using StandardScaler that you computed before
- Create SGDRegressor, fit and predict
- Compute MSE on y_test and y_pred this time. You will get output as 1592430104.0.

Step8. [Predict using Ridge Regression]

- Use scaled X_train and X_test using StandardScaler that you computed before
- Create RidgeCV, fit and predict
- Compute MSE on y_test and y_pred this time. You will get output as 1442196000.3367693.

Step8. [Predict using Lasso Regression]

- Use scaled X_train and X_test using StandardScaler that you computed before
- Create LassoCV, fit and predict
- Compute MSE on y_test and y_pred this time. You will get output as 1409368613.5329669.

Step9.[RMSE]. Print Root Mean Squared Error values (use numpy.sqrt() method) as below and compare error values.

RMSE without one hot encoding: 38403.0
RMSE with One hot encoding: 38224.0
RMSE with OHE and Standard Scaling: 38224.0
RMSE with OHE and MinMax Scaling: 38224.0
RMSE of SGDRegressor with OHE and Standard Scaler: 38528.0
RMSE of RidgeCV with OHE and Standard Scaler: 37976.0
RMSE of LassoCV with OHE and Standard Scaler: 37542.0

```
model = LinearRegression()
model.fit(x_train, y_train)
    y_pred = model.predict(x_test)
    y_pred

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

Step 3:

```
plt.scatter(y_test y_pred)
```

Step 4:

```
en_df = pd.get_dummies(df, columns=["Central Air", "Bldg Type"])
    en_df.head()
    en.df.shape
    en_df.columns
```

Step 5

```
e_y = en_df["Sales price"]
    e_y
e_inp = ["1st FlrSF", "2nd FlrSF", "3SSnporch",
    "Bedroom AbvGr", "BsmtFins", "Enclosedporch",
    "Fireplaces", "Full Bath", "GrLivArea",
    "LotArea", "LotFrontage", "OverallQual", "FinSF",
    "HSSUbclass", "OverallCond", "PoolArea", "Screen
    Porch", "TotRmsAbvGrd", "Central Air_N"
```

NOTE

'CentralAir_Y', 'BldgType_1Fam', 'BldgType_2Fam']

```
e.X = en_df[e_inp]

e_X_train, ex_test, ey_train, ey_test = train_
                    test_split(e_X, e_y)

model1 = LinearRegression()
model1.fit(ex_train, ey_train)
ey_pred = model1.predict(ex_test)
ey_pred
```

Step6:

```
from sklearn.preprocessing import standard
                                  scaler.
scales = standard scaler()
ss = scale.fit.transform(ex_train)
ss
ss1 = scale.transform(ex_test)
ss1
model2 = Linear.Regression()
model2.fit(ss, ey_train)
se_X_pred = model12.predict(ss1)
se_X_pred
mean_squared_error(ey_test, se_y_pred)
```

Step 7:

```
from sklearn.preprocessing import MinMaxScaler
m_scaler = MinMaxScaler()
m_ss = m_scaler.fit_transform(ex_train)
m_ss
m_ss1 = m_scaler.transform(ex_test)
m_ss1
model3 = LinearRegression()
model3.fit(m_ss, ey_train)
me_y_pred = model3.predict(m_ss1)
me_y_pred
mean_squared_error(ey_test, me_y_pred)
```

Step 8:

```
from sklearn.linear_model import SGDRegressor
from sklearn.pipeline import make_pipeline
SGD = make_pipeline(StandardScaler(), SGDRegressor
    (max_iter=1000, t
SGD.fit(X, X)
r_y_pred = SGD.predict(X)
r_y_pred
mean_squared_error(ey_train, ridge_y_pred)
```

Step 9:

```
from math import sqrt
print("RMSE without One hot encoding", sqrt(mean_squared
    _error))
```

NOTE

```
print("RMSE with one and standard Scaling: ", sqrt(s-me))
print("RMSE with one and MinMax Scaling: ", sqrt(mse-mm))
print("RMSE of SGDRegressor with one and standard scaler:", sqrt(mse-sag))
print("RMSE of Ridge cv with one and standard scaler:", sqrt(mse-rid))
print("RMSE of Lassocv with one and standard scaler:", sqrt(mse-las))
```

REPORT

Lab4.House Price Prediction using LR with Regularization

In this lab we have learned how to build regression models to predict the sales price of houses using given dataset.

By understanding the given data if there are any missing values then we need to clean (pre-processing)

On this scenario we have categorical data here our job is to convert categorical data to numerical data think that if you have very huge dataset with the help of get_dummies () library this will convert all the categorical data to numerical data then we rebuild linear regression model.

As like the previous regression problem here we apply standard scaler and minmax scaler to normalize then we are comparing the performance of our LR model with various regressor like SGD regressor, RidgeCV and LassoCV.

Then finally will compute Root Mean Squared Error values by using numpy.sqrt () method then compare with all models what we have performed.

Name:Viviyan Richards W

#Roll no:205229133

Lab5. Diabetes Classification using Logistic Regression

Step1. [Understand Data]. Using Pandas, import "diabetes.csv" file and print properties such as head, shape, columns, dtype, info and value_counts

In [2]:
```
import pandas as pd
data = pd.read_csv('diabetes.csv')
data.head()
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	A
0	6	148	72	35	0	33.6	0.627	
1	1	85	66	29	0	26.6	0.351	
2	8	183	64	0	0	23.3	0.672	
3	1	89	66	23	94	28.1	0.167	
4	0	137	40	35	168	43.1	2.288	

In [3]:
```
data.shape
```

Out[3]:
```
(768, 9)
```

In [4]:
```
data.columns
```

Out[4]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [5]:
```
data.dtypes
```

Out[5]:
```
Pregnancies                 int64
Glucose                     int64
BloodPressure               int64
SkinThickness               int64
Insulin                     int64
BMI                       float64
DiabetesPedigreeFunction  float64
Age                         int64
Outcome                     int64
dtype: object
```

In [6]:
```
type(data)
```

Out[6]:
```
pandas.core.frame.DataFrame
```

In [7]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [8]:
```
data.value_counts()
```

Out[8]:
```
Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin  BMI   DiabetesPedi
greeFunction  Age  Outcome
0            57       60             0              0        21.7  0.735
67   0          1
             67       76             0              0        45.3  0.194
46   0          1
5            103      108            37             0        39.2  0.305
65   0          1
             104      74             0              0        28.8  0.153
48   0          1
             105      72             29             325      36.9  0.159
28   0          1

..
2            84       50             23             76       30.4  0.968
21   0          1
             85       65             0              0        39.6  0.930
27   0          1
             87       0              23             0        28.9  0.773
25   0          1
                      58             16             52       32.7  0.166
25   0          1
17           163      72             41             114      40.9  0.817
47   1          1
Length: 768, dtype: int64
```

Step2. [Build Logistic Regression Model]

Prepare X matrix (8 feature columns) and y vector (ie., Outcome column)

In [9]:
```
X=data.drop("Outcome", axis=1)
y=data[["Outcome"]]
```

In [10]:
```
X
```

Out[10]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
0	6	148	72	35	0	33.6	0.627
1	1	85	66	29	0	26.6	0.351
2	8	183	64	0	0	23.3	0.672
3	1	89	66	23	94	28.1	0.167
4	0	137	40	35	168	43.1	2.288
...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171
764	2	122	70	27	0	36.8	0.340
765	5	121	72	23	112	26.2	0.245
766	1	126	60	0	0	30.1	0.349
767	1	93	70	31	0	30.4	0.315

768 rows × 8 columns

In [11]:
```
y
```

Out[11]:

	Outcome
0	1
1	0
2	1
3	0
4	1
...	...
763	0
764	0
765	0
766	1
767	0

768 rows × 1 columns

Split dataset with stratified shuffle split for training and testing as X_train, X_test, y_train, y_test (use 25% test size).

In [12]:
```
from sklearn.model_selection import StratifiedShuffleSplit
```

In [13]:
```
StratifiedShuffleSplit()
shuffle = StratifiedShuffleSplit(n_splits=4, test_size=0.25,random_state=0)
```

In [14]:
```
shuffle.get_n_splits(X,y)
```

Out[14]:
```
4
```

Create LogisticRegression model, fit on training set and predict on test set

In [15]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [16]:
```
for train, test in shuffle.split(X, y):
    X_train, X_test = X.iloc[train], X.iloc[test]
    y_train, y_test = y.iloc[train], y.iloc[test]
```

In [17]:
```
from sklearn.linear_model import LogisticRegression
```

In [18]:
```
lmodel = LogisticRegression()
```

In [19]:
```
lmodel.fit(X_train,y_train)
```

Out[19]:
```
LogisticRegression()
```

In [20]:
```
y_predict = lmodel.predict(X_test)
y_predict
```

Out[20]:
```
array([0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
       1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0], dtype=int64)
```

In [21]:
```
lmodel.score(X_train,y_train)
```

Out[21]:
```
0.7899305555555556
```

Predict on a new sample

In [22]:
```
new_person = [[6, 200, 90, 10, 25, 23.3, 0.672, 42]]
```

In [23]:
```
print(lmodel.predict(new_person))
```

```
[1]
```

Step3. [Compute Classification Metrics]

Precision score

In [24]:
```
from sklearn.metrics import precision_score
print(precision_score(y_test, y_predict))
```

```
0.6727272727272727
```

Recall score

In [25]:
```
from sklearn.metrics import recall_score
print(recall_score(y_test, y_predict))
```

```
0.5522388059701493
```

Accuracy score

Which is 75% Accurate

In [26]:
```
from sklearn.metrics import accuracy_score
lor_ascore=accuracy_score(y_test, y_predict)
```

In [27]:
```
lor_ascore
```

Out[27]:
```
0.75
```

AUC score

AUC is the percentage of the ROC plot that is underneath the curve:

In [28]:
```
from sklearn.metrics import roc_auc_score
```

In [29]:
```
print(roc_auc_score(y_test, y_predict))
```

```
0.7041194029850747
```

Step4. [Understand Correlation]

Create confusion matrix between y_test and y_pred and plot confusion matrix values in a Heatmap. Explain the meaning of the 4 numbers you get.

In [30]:
```
from sklearn.metrics import confusion_matrix
cnf_matrix=confusion_matrix(y_test, y_predict)
cnf_matrix
```

Out[30]:
```
array([[107,  18],
       [ 30,  37]], dtype=int64)
```

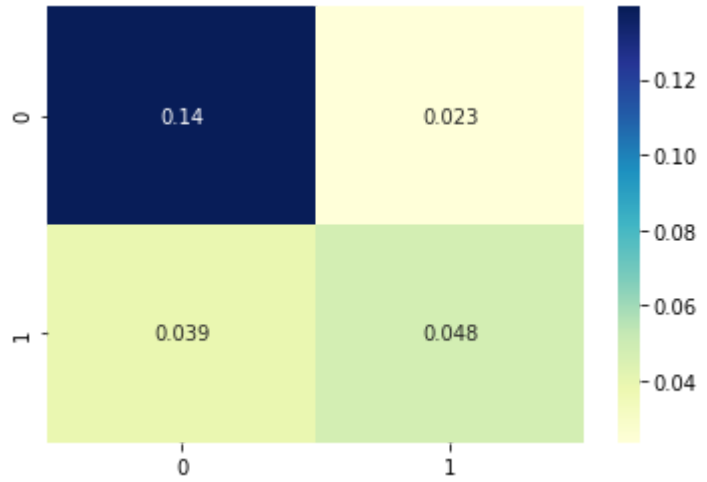
In [31]:
```
cf_ac_score = accuracy_score(y_test, y_predict)
```

In [32]:
```
cf_ac_score
```

Out[32]:
```
0.75
```

In [33]:
```
import seaborn as sns
sns.heatmap(confusion_matrix(y_test,y_predict) / len(y), cmap='YlGnBu', annot=Tru
```

Out[33]:
```
<AxesSubplot:>
```



X-axis is predicted value

Y-axis is real value

Step5. [Normalization using MinmaxScaler and rebuild LoR]

Now, normalize your X_train and X_test values using MinmaxScaler

In [34]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_trained = scaler.fit_transform(X_train)
X_tested = scaler.transform(X_test)
```

In [35]:
```
X_trained.shape
```

Out[35]:
```
(576, 8)
```

In [36]:
```
X_tested.shape
```

Out[36]:
```
(192, 8)
```

Create a new LogisticRegression model, fit on normalized training set and predict on the normalized test set

In [37]:
```
from sklearn.linear_model import LogisticRegression
lmodel1 = LogisticRegression()
```

In [38]:
```
lmodel1.fit(X_trained, y_train)
```

Out[38]:
```
LogisticRegression()
```

In [39]:
```
yn_predict = lmodel1.predict(X_tested)
yn_predict
```

Out[39]:
```
array([0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
       1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0], dtype=int64)
```

In [40]:
```
lmodel1.score(X_trained, y_train)
```

Out[40]:
```
0.7899305555555556
```

Precision score

In [41]:
```
from sklearn.metrics import precision_score
print(precision_score(y_test, yn_predict))
```

```
0.673469387755102
```

Recall score

In [42]:
```
from sklearn.metrics import recall_score
print(recall_score(y_test, yn_predict))
```

```
0.4925373134328358
```

Accuracy score

Which is 72% Accurate

In [43]:
```
from sklearn.metrics import accuracy_score
minmax_ascore=accuracy_score(y_test, yn_predict)
```

In [44]:
```
minmax_ascore
```

Out[44]:
```
0.7395833333333334
```

AUC score

AUC is the percentage of the ROC plot that is underneath the curve

In [45]:
```
from sklearn.metrics import roc_auc_score
```

In [46]:
```
lgr_auc=roc_auc_score(y_test, yn_predict)
lgr_auc1=('LoR minmax, AUC=',lgr_auc)
lgr_auc1
```

Out[46]:
```
('LoR minmax, AUC=', 0.6822686567164179)
```

Step6. [Normalization using StandardScaler and rebuild LoR]

In [47]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
Xs_trained = scaler.fit_transform(X_train)
Xs_tested = scaler.transform(X_test)
```

In [48]:
```
Xs_trained.shape
```

Out[48]:
```
(576, 8)
```

In [49]:
```
Xs_tested.shape
```

Out[49]:
```
(192, 8)
```

In [50]:
```
from sklearn.linear_model import LogisticRegression
lmodel2 = LogisticRegression()
```

In [51]:
```
lmodel2.fit(Xs_trained, y_train)
```

Out[51]:
```
LogisticRegression()
```

In [52]:
```
ys_predict = lmodel2.predict(Xs_tested)
ys_predict
```

Out[52]:
```
array([0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
       1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0], dtype=int64)
```

In [53]:
```
lmodel2.score(Xs_trained, y_train)
```

Out[53]:
```
0.7795138888888888
```

Precision score

In [54]:
```
from sklearn.metrics import precision_score
print(precision_score(y_test, ys_predict))
```

```
0.6851851851851852
```

Recall score

In [55]:
```
from sklearn.metrics import recall_score
print(recall_score(y_test, ys_predict))
```

```
0.5522388059701493
```

Accuracy score

Which is 75% Accurate

In [56]:
```
from sklearn.metrics import accuracy_score
ss_ascore=accuracy_score(y_test, ys_predict)
```

In [57]:
```
ss_ascore
```

Out[57]:
```
0.7552083333333334
```

AUC score

AUC is the percentage of the ROC plot that is underneath the curve

In [58]:
```
from sklearn.metrics import roc_auc_score
```

In [59]:
```
ss_auc=roc_auc_score(y_test, ys_predict)
ss_auc1=('AUC=',ss_auc)
ss_auc1
```

Out[59]:
```
('AUC=', 0.7081194029850746)
```

Among the 3 models, which model gives better classification scores?

In [60]:
```
print('StandardScaler:',ss_ascore)
print('MinmaxScaler:',minmax_ascore)
print('Logistic Regression Model:',lor_ascore)
```

```
StandardScaler: 0.7552083333333334
MinmaxScaler: 0.7395833333333334
Logistic Regression Model: 0.75
```

Step7. [Plot ROC curve]

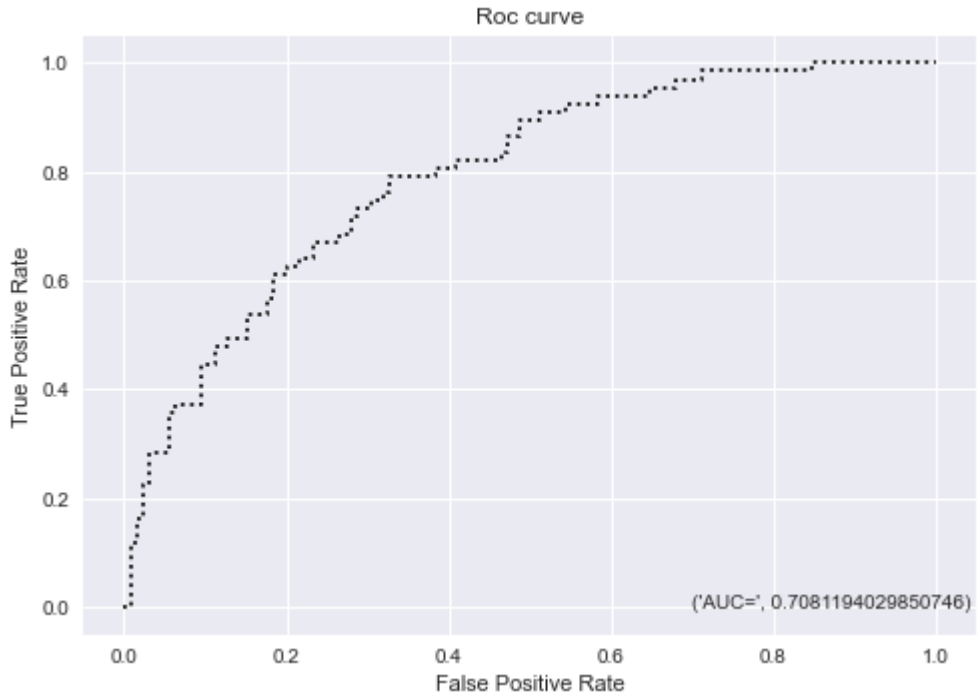
Plot ROC curve as shown below. You can use the MinmaxScaler scaled values of X_test for computing predict_proba() score.

In [61]:
```
from sklearn.metrics import roc_curve
```

In [62]:
```
predict_pb1 = lmodel1.predict_proba(X_tested)
fpr1, tpr1, threshold1 = roc_curve(y_test,predict_pb1[:,1], pos_label=1)
```

In [63]:
```
import matplotlib.pyplot as plt
plt.style.use('seaborn')
plt.annotate(xy=[0.7,0], s=ss_auc1)
plt.plot(fpr1, tpr1, linestyle=':', color='black',label='Logistic Regression')
plt.title('Roc curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
```

Out[63]:
```
Text(0, 0.5, 'True Positive Rate')
```



Step8. [Comparison with KNN classifier].

Create a KNN classifier with default values, fit on the scaled X using MinmaxScaler, predict and print classification metric scores.

In [64]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [65]:
```
lmodel3 = KNeighborsClassifier(n_neighbors=3)
```

In [66]:
```
lmodel3.fit(X_trained,y_train)
```

Out[66]:
```
KNeighborsClassifier(n_neighbors=3)
```

In [67]:
```
knn_y_predict = lmodel3.predict(X_tested)
knn_y_predict
```

Out[67]:
```
array([1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
       1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0,
       0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
       0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1], dtype=int64)
```

Precision score

In [68]:
```
from sklearn.metrics import precision_score
print(precision_score(y_test, knn_y_predict))
```

```
0.6271186440677966
```

Recall score

In [69]:
```
from sklearn.metrics import recall_score
print(recall_score(y_test, knn_y_predict))
```

```
0.5522388059701493
```

Accuracy score

In [70]:
```
from sklearn.metrics import accuracy_score
kn_ascore=accuracy_score(y_test, knn_y_predict)
kn_ascore
```

Out[70]:
```
0.7291666666666666
```

AUC score

In [71]:
```
from sklearn.metrics import roc_auc_score
kn_auc=roc_auc_score(y_test, knn_y_predict)
kn_auc1=('KNN minmax, AUC=',kn_auc)
kn_auc1
```

Out[71]:
```
('KNN minmax, AUC=', 0.6881194029850747)
```

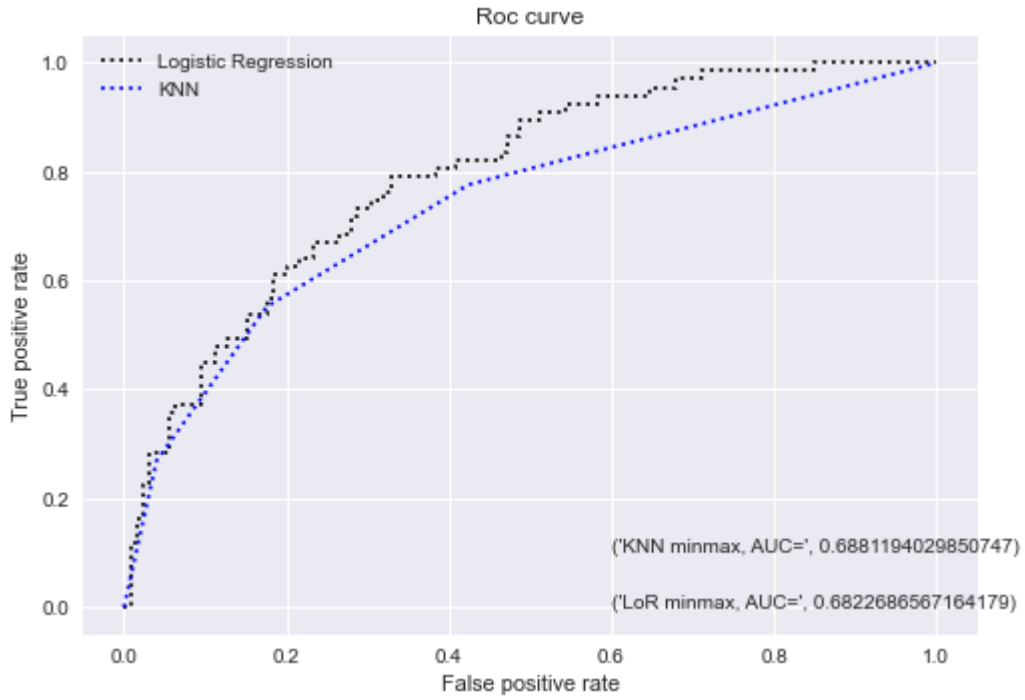
Step9. [Update ROC curve]

Update your ROC curve, this time, with one more curve of KNN classifier,

In [72]:
```
predict_pb2 = lmodel3.predict_proba(X_tested)
fpr1,tpr1,threshold1 = roc_curve(y_test, predict_pb1[:,1],pos_label=1)
fpr2,tpr2,threshold2 = roc_curve(y_test, predict_pb2[:,1],pos_label=1)
```

In [73]:
```
plt.plot(fpr1,tpr1,linestyle=':',color='black',label='Logistic Regression')
plt.plot(fpr2,tpr2,linestyle=':',color ='blue',label='KNN')
plt.annotate(xy=[0.6,0.1], s=kn_auc1)
plt.annotate(xy=[0.6,0], s=lgr_auc1)
plt.legend(loc='best')
plt.title('Roc curve')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
```

Out[73]:
```
Text(0, 0.5, 'True positive rate')
```



Step10. [Regularization]

In order to reduce overfitting of your data, you will use LogisticRegressionCV model with L1 and L2 regularization parameters. Create both models using the following statements

In [74]:
```
from sklearn.linear_model import LogisticRegressionCV
```

In [75]:
```
lmodel4 = LogisticRegressionCV(Cs=10, cv=4, penalty='l1', solver='liblinear')
lmodel5 = LogisticRegressionCV(Cs=10, cv=4, penalty='l2')
```

Perform fit using MinmaxScaler scaled values and predict

In [76]:
```
print(lmodel4.fit(X_trained,y_train))
print(lmodel5.fit(X_trained,y_train))
```

```
LogisticRegressionCV(cv=4, penalty='l1', solver='liblinear')
LogisticRegressionCV(cv=4)
```

In [77]:
```
lr_y_predict1 = lmodel4.predict(X_tested)
lr_y_predict2 = lmodel5.predict(X_tested)
print('Logistic RegressionCV L1:\n',lr_y_predict1)
print('--------------------------------------------------------------------------------')
print('Logistic RegressionCV L2:\n',lr_y_predict2)
```

```
Logistic RegressionCV L1:
 [0 0 1 1 0 1 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0
 0 0 0 0 1 0 1 1 0 1 0 1 0 0 0 1 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 1 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0
 0 0 1 1 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 1 0 1 1 1 1 0 0 0 0 0 0 1 0 0 0
 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 1 0 0 1 1 0 0
 0 0 0 1 0 0 0]
--------------------------------------------------------------------------------
Logistic RegressionCV L2:
 [0 0 1 1 0 1 1 0 0 1 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0
 0 0 0 0 1 0 1 1 0 1 0 1 0 0 0 1 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1
 1 0 0 0 0 1 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0
 0 0 1 1 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 1 0 1 1 1 1 0 0 0 1 0 0 1 0 0 0
 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 1 0 0 0 0 0 1 0 0 1 1 0 0
 0 0 0 1 0 0 0]
```

In [78]:
```
from sklearn.metrics import roc_auc_score
lrp_auc=roc_auc_score(y_test, lr_y_predict1)
lrp1_auc=('LoR L1 minmax, AUC=',lrp_auc)
lrp1_auc
```

Out[78]: ('LoR L1 minmax, AUC=', 0.6748059701492537)

In [79]:
```
from sklearn.metrics import roc_auc_score
lrp_auc2=roc_auc_score(y_test, lr_y_predict2)
lrp2_auc=('LoR L2 minmax, AUC=',lrp_auc2)
lrp2_auc
```

Out[79]: ('LoR L2 minmax, AUC=', 0.6931940298507463)

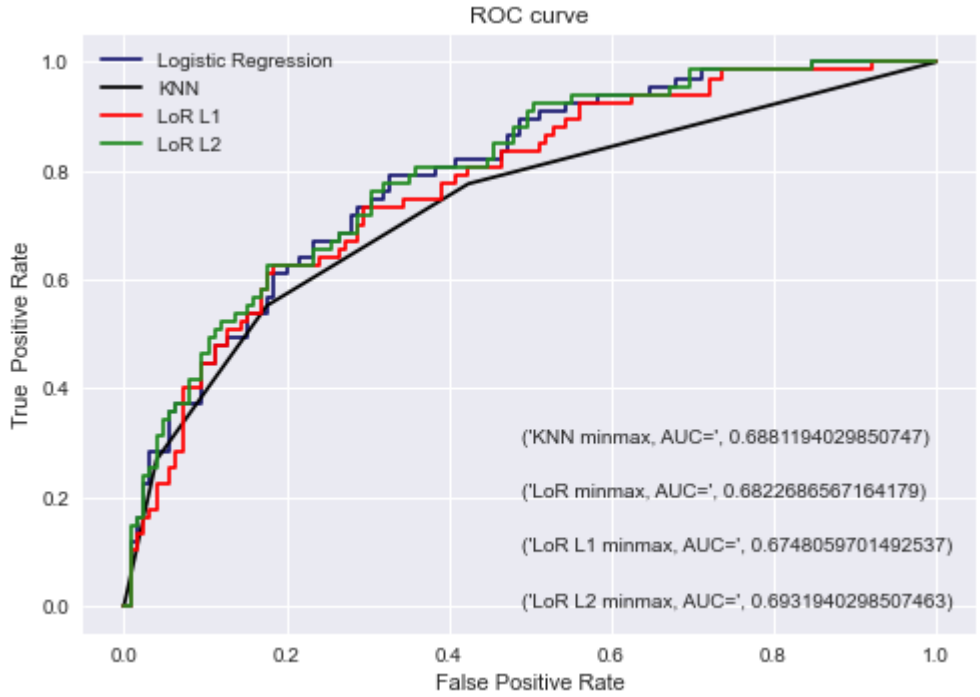
Step11. [Update ROC curve]

Update your ROC curve, this time, with two more curves

In [80]:
```
predict_pb3 = lmodel4.predict_proba(X_tested)
predict_pb4 = lmodel5.predict_proba(X_tested)
fpr1,tpr1,thresh1 = roc_curve(y_test, predict_pb1[:,1], pos_label=1)
fpr2,tpr2,thresh2 = roc_curve(y_test, predict_pb2[:,1], pos_label=1)
fpr3,tpr3,thresh3 = roc_curve(y_test, predict_pb3[:,1], pos_label=1)
fpr4,tpr4,thresh4 = roc_curve(y_test, predict_pb4[:,1], pos_label=1)
```

In [90]:
```
plt.plot(fpr1,tpr1,linestyle='-',color='midnightblue', label='Logistic Regressior
plt.plot(fpr2,tpr2,linestyle='-',color='black', label='KNN')
plt.plot(fpr3,tpr3,linestyle='-',color='red', label='LoR L1')
plt.plot(fpr4,tpr4,linestyle='-',color='forestgreen', label='LoR L2')
plt.annotate(xy=[0.49,0.3], s=kn_auc1)
plt.annotate(xy=[0.49,0.2], s=lgr_auc1)
plt.annotate(xy=[0.49,0.1], s=lrp1_auc)
plt.annotate(xy=[0.49,0], s=lrp2_auc)
plt.legend(loc='best')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True  Positive Rate')
```

Out[90]:
```
Text(0, 0.5, 'True  Positive Rate')
```



Logistic Regression L2 Minmax gives 69% of AUC score so this shows it is the best score

lab 5:-

step-1: import pandas as pd

import csv

```
dPab = pd.read_csv("diabetes.csv")
dPab.
dPab.head()
dPab.shape
df = pd.read_csv("diabetes.csv")
df.
y = df.columns
b df.info()
df.dtypes
df.dtypes.value_counts()
```

step2:-

```
X = df.drop("Outcome", axis=1)
Y = df[["Outcome"]]
X
Y
from sklearn.model_selection import.
    stratified_shuffle_split.
```

Lab5. Diabetes Classification using Logistic Regression

Objectives

In this lab, you will classify using Logistic Regression model, whether a person will become diabetic or not using PIMA diabetes dataset available in UCI. You will also predict a new person who is not in the dataset will become diabetic of not based on his details.

Learning Outcomes

After completing this lab, you will be able to

- Understand data and build baseline LogisticRegression model
- Create Heatmap with confusion matrix values
- Apply scaling using StandardScaler and MinMaxScaler and rebuild LoR model
- Print classification metrics scores and plot ROC curve
- Compare the performance of LoR model with LogisticRegressionCV with L1 and L2 regularization

Business Use Case

You are a data scientist. A leading hospital in your city has approached you with the medical details of their patients related to their diabetes information. The hospital has given you a file (diabetes.csv) that contains details of 768 patients and each patient is described with these 9 features. Here, ***Outcome*** is the ***dependent variable*** and ***all other 8 variables are independent variables.***

- Pregnancies: Number of times pregnant
- Glucose: Plasma glucose concentration over 2 hours in an oral glucose tolerance test
- BloodPressure: Diastolic blood pressure (mm Hg)
- SkinThickness: Triceps skin fold thickness (mm)
- Insulin: 2-Hour serum insulin (mu U/ml)
- BMI: Body mass index (weight in kg/(height in m)2)
- DiabetesPedigreeFunction: Diabetes pedigree function (a function which scores likelihood of diabetes based on family history)
- Age: Age (years)
- Outcome: Class variable (0 if non-diabetic, 1 if diabetic)

The hospital wants you to build a model so that the model will assess when a new patient visits them he will become diabetic or not.

Step1. [Understand Data]. Using Pandas, import **"diabetes.csv"** file and print properties such as head, shape, columns, dtype, info and value_counts.

Step2. [Build Logistic Regression Model]

- Prepare X matrix (8 feature columns) and y vector (ie., Outcome column)
- Split dataset with stratified shuffle split for training and testing as X_train, X_test, y_train, y_test (use **25%** test size).
- Create LogisticRegression model, fit on training set and predict on test set

Step3. [Predict on a new sample]

- Will this person become diabetic?. His details are given below.
- new_person = [[6, 200, 90, 10, 25, 23.3, 0.672, 42]]

Step3. [Compute Classification Metrics]

- Compute and print Accuracy, Precision, Recall and AUC scores

Step4. [Understand Correlation]

- Create confusion matrix between y_test and y_pred and plot confusion matrix values in a Heatmap. Explain the meaning of the 4 numbers you get.

stratified shuffle split()

```
shuf = StratifiedShuffleSplit(n_splits=4, test_size=0.25,
                              random_state=0)
shuf.get_n_splits(X,Y)
import warnings
warnings.filterwarnings("ignore")
for train, test in shuf.split(X,Y):
    X_train, X_test = X.iloc[train], X.iloc[test]
    y_train, y_test = y.iloc[train], y.iloc[test]
from sklearn.linear_model import LogisticRegression
    logmodel = LogisticRegression()
    logmodel.fit(X_train, Y_train)
      y_predic = logmodel.predict(X_test)
      y_predic
    logmodel.score(X_train, Y_train)
```

step 3:

```
new_person = [[6, 200, 90, 10, 25, 33.3, 0.672, 400]]
print(logmodel.predict(new_person))
```

Precision:

```
from sklearn.metrics import precision_score
print(precision_score(y_test, y_predic))
```

Recall:

```
from sklearn.metrics import recall_score
print(recall_score(y_test, y_predic))
```

Step5. [Normalization using MinmaxScaler and rebuild LoR]

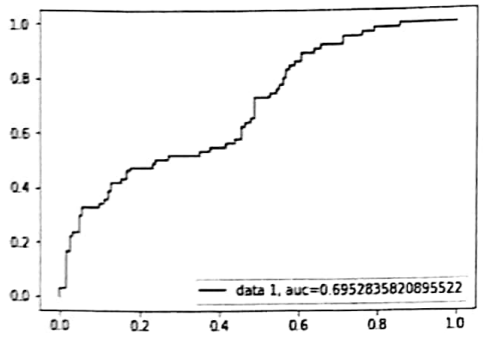
- Now, normalize your X_train and X_test values using MinmaxScaler
- Create a new LogisticRegression model, fit on normalized training set and predict on the normalized test set
- Compute and print Accuracy, Precision, Recall and AUC scores

Step6. [Normalization using StandardScaler and rebuild LoR]

- Repeat Step5 with StandardScaler
- Among the 3 models, which model gives better classification scores?

Step7. [Plot ROC curve]

- Plot ROC curve as shown below. You can use the MinmaxScaler scaled values of X_test for computing predict_proba() score.

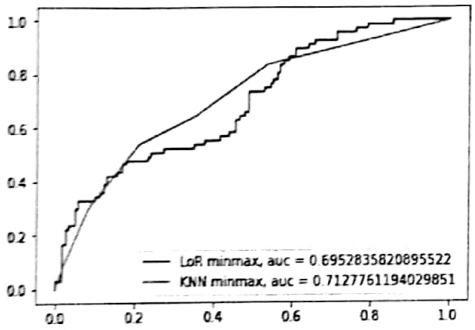


Step8. [Comparison with KNN classifier].

Create a KNN classifier with default values, fit on the scaled X using MinmaxScaler, predict and print classification metric scores.

Step9. [Update ROC curve]

- Update your ROC curve, this time, with one more curve of KNN classifier, as shown below.



Step10. [Regularization]

- In order to reduce overfitting of your data, you will use *LogisticRegressionCV* model with L1 and L2 regularization parameters. Create both models using the following statements
  - `model1 = LogisticRegressionCV(Cs=10, cv=4, penalty='l1', solver='liblinear')`
  - `model2 = LogisticRegressionCV(Cs=10, cv=4, penalty='l2')`
- Perform fit using MinmaxScaler scaled values and predict

Step11. [Update ROC curve]

- Update your ROC curve, this time, with two more curves, as shown below

Accuracy:

```
from sklearn.metrices import accuracy_score
log_acscore = accuracy_score(y_test, y_predic)
log_acscore
```

AUC Scores

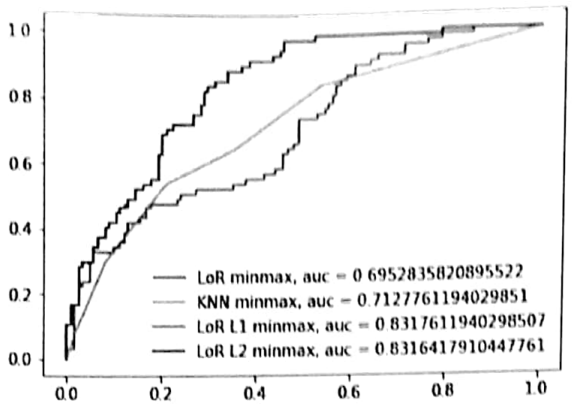
```
from sklearn.metrices import roc_auc_score
print(roc_auc_score(y_test, y_predic))
```

Step: 4

```
from sklearn.metrices import confusion_matrix
Confu_matrix = confusion_matrix(y_test, y_predic)
Confu_matrix
Confu_accu_score = accuracy_score(y_test, y_predic)
Confu_accu_score
import seaborn as sns
sns.heatmap(confusion_matrix(y_test, y_periodic)
/len(y), cmap="YlGnBu" annot=True)
```

Step: 5

```
from sklearn.preprocessing import minmax scalar
Scales = MinmaxScaler()
X_trained_min = Scaler.fit_transform(X_train)
X_test_min = Scaler.transform(x_test)
X_trained_min.shape
X_tested_min.shape
```



Out of these 4 models, which model performs the best?. How?. Why?.

```
from sklearn.linear_model import LogisticRegression.
logmodel1 = LogisticRegression()
logmodel1.fit(x_trained_min, y_train
y_predict = logmodel1.predict(x_tested_min)
y_predict.
logmodel1.score(x_trained.min, y_train)
```

Precision:

```
from Sklearn.metrices import precision_score
print(precision_score(y_test, y_predict)
```

Recall

```
from sklearn.metrices.import recall_score
print(recall_score(y_test, y_predict)
```

Accuracy.

```
from sklearn.metrices.import accuracy_score
min_accuscore = accuracy_score(y_test, y_predict)
min_arcScore
```

Auc Score:

```
from sklearn.metrics import roc_auc_score
log_auc_sc = roc_auc_score(y_test, y_predict)
log_auc1 = (LOR.minman, Auc =, log_acscore)
log_auc1.
```

step:6

```
from sklearn.preprocessing import standardscalex
scalex = standardscalex()
X_trained_stand = scalex.fit_transform(X_train)
X_tested_stand = scalex.transform(x_test)
X_trained_stand.shape
X_tested_stand.shape
from sklearn.linear_model import logistic Regression
logmodel2 = logistic Regression()
log model 2.fit(Xtrained_stand, y_train)
y_predict_stand = logmodel2.predict(X_tested_stand)
y_predict = stand
logmodel2.score(x_trained.stand, y_train)
```

precision score

```
from sklearn.metrics import precision_score.
print(precision_score(y_test, y_predict_stand))
```

NOTES

Recall score

```
from sklearn.metrices import recall_score
print (recall_score (y_test, y_predict_stand))
```

Accuracy:

```
from sklearn.metrices import accuracy_score
stand_accscore = accuracy_score(y_test, y_predict_stand)
stand_accuscore.
```

Auc score

```
from sklearn.metrices import roc_auc_score
stand_aucscore = roc_auc_score (y_test, y_predict_stand)
stand_aucs = (AUC= stand_aucscore)
stand aucs.
print ('logistic Regression model); log_accscore)
print ('Minmax Scales:', min_accuscore
print ('standard scaler:', stand_accscore)
```

Step 4:

```
from sklearn.metrices import roc_curve
pred_prb3 = log_model.predict_proba
(x_tested_min)
```

```
fpr3, tpr3, threshold = roc_curve
    (y_test, pred_prb3[:,1], pos_label=1)
import matplotlib.pyplot as plt
    plt.style.use('seaborn')
    plt.annotate(xy=[0.7,0], s=str(auc3))
plt.plot(fpr1, tpr1, linestyle=':', color='black',
                label='Logistic Regression')
plt.title('Roc curve')
plt.xlabel('False positive Rate')
plt.ylabel('True positive Rate')
```

step: 8:

```
from sklearn.neighbors import KNeighboursclassifier.
log_model3 = KNeighboursClassifier(n_neighbors=3)
log_model3.fit(x_trained_min, y_train)
    knn_y_pred = log_model3.predict(x_testedmin)
    knn_y_pred
```

REPORT

Lab5.Diabetes Classification using Logistic Regression

In this lab we have learned how to classify using logistic regression model whether a person will become diabetic or not using given diabetes dataset. We need to predict a new person who is not in the dataset will become diabetic of not based on his details.

On this scenario we split dataset by 25% of test size which is testsize=0.25 then we need to create a logistic regression model.

Will train the model with 25% of data then will predict using test data. After this will check by creating a new data with the name of new_person and make prediction on this data.

Next will print accuracy score, precision, recall and auc scores to understand the model then creating a heatmap with confusion matrix values.

Then apply scaling using standard scaler and minmax scaler and build a new logistic regression model. For better understanding we print the classification metrics scores and plot roc curve. Then finally compare performance of LoR model with LoRCV with L1 and L2 regularization which has different parameters.

Roll No: 205229133

Lab6. Predictive Analytics for Hospitals

Step1. [Import dataset]

In [1]:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:

```
hos = pd.read_csv("diabetes.csv")
```

In [3]:

```
hos.head()
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
0	6	148	72	35	0	33.6	0.62
1	1	85	66	29	0	26.6	0.35
2	8	183	64	0	0	23.3	0.67
3	1	89	66	23	94	28.1	0.16
4	0	137	40	35	168	43.1	2.28

In [4]:

```
hos.shape
```

Out[4]:

```
(768, 9)
```

In [5]:

```
hos.columns
```

Out[5]:

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insuli
n',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [6]:

```
hos.dtypes
```

Out[6]:

```
Pregnancies                 int64
Glucose                     int64
BloodPressure               int64
SkinThickness               int64
Insulin                     int64
BMI                       float64
DiabetesPedigreeFunction  float64
Age                         int64
Outcome                     int64
dtype: object
```

In [7]:

```
hos.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [8]:

```
hos.value_counts()
```

Out[8]:

```
Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin  BMI   DiabetesP
edigreeFunction  Age  Outcome
0            57       60             0              0        21.7  0.735
67   0          1
             67       76             0              0        45.3  0.194
46   0          1
5            103      108            37             0        39.2  0.305
65   0          1
             104      74             0              0        28.8  0.153
48   0          1
             105      72             29             325      36.9  0.159
28   0          1

..
2            84       50             23             76       30.4  0.968
21   0          1
             85       65             0              0        39.6  0.930
27   0          1
             87       0              23             0        28.9  0.773
25   0          1
                      58             16             52       32.7  0.166
25   0          1
17           163      72             41             114      40.9  0.817
47   1          1
Length: 768, dtype: int64
```

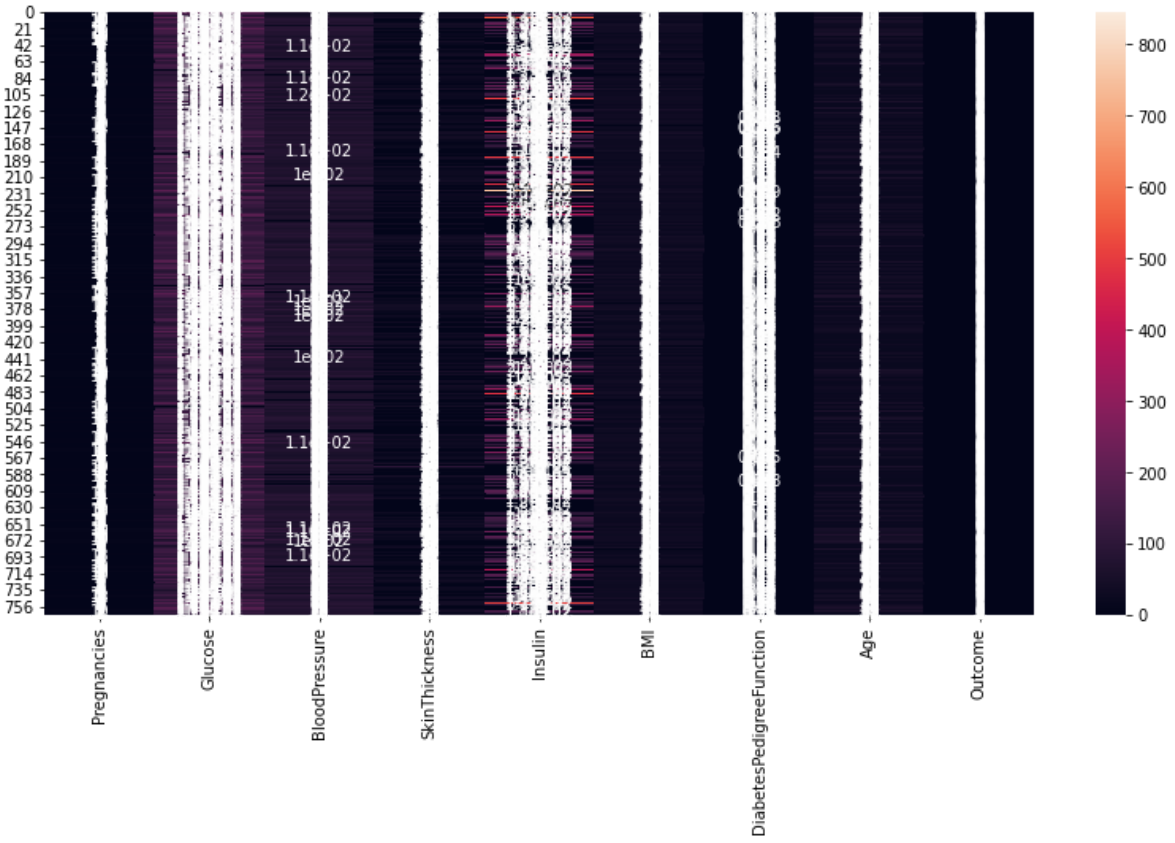
Step2. [Identify relationships between feature]

In [9]:

```
plt.figure(figsize=(14,7))
sns.heatmap(data=hos,annot=True)
```

Out[9]:

```
<AxesSubplot:>
```



Step3. [Prediction using one feature]

In [10]:

```
X=hos[['Age']]
```

In [11]:

```
y=hos.Outcome
```

In [12]:

```
X
```

Out[12]:

	Age
0	50
1	31
2	32
3	21
4	33
...	...
763	63
764	27
765	30
766	47
767	23

768 rows × 1 columns

In [13]:

```
y
```

Out[13]:

```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

In [14]:

```
from sklearn.linear_model import LogisticRegression
```

In [15]:

```
model1 = LogisticRegression()
```

In [16]:

```
model1.fit(X,y)
```

Out[16]:

```
LogisticRegression()
```

In [17]:

```
model1.coef_
```

Out[17]:

```
array([[0.04202466]])
```

In [18]:

```
model1.intercept_
```

Out[18]:

```
array([-2.04744865])
```

In [19]:

```
model1.predict([[60]])
```

Out[19]:

```
array([1], dtype=int64)
```

In [20]:

```
lrf = model1.coef_ * 60 + model1.intercept_
from scipy.special import expit
if expit(lrf) > 0.5:
    print(expit(lrf))
    print('YES, he will become diabetic')
else:
    print("NO, he will not be diabetic")
```

```
[[0.61633741]]
YES, he will become diabetic
```

Step4. [Prediction using many features]

In [21]:

```
X_=hos[['Age','BMI','Glucose']]
```

In [22]:

```
model2 = LogisticRegression()
```

In [23]:

```
model2.fit(X_,y)
```

Out[23]:

```
LogisticRegression()
```

In [24]:

```
model2.predict([[40,30,150]])
```

Out[24]:

```
array([1], dtype=int64)
```

In [25]:

```
model2.coef_
```

Out[25]:

```
array([[0.03015421, 0.08157404, 0.03251154]])
```

In [26]:

```
model2.intercept_
```

Out[26]:

```
array([-8.39311252])
```

In [27]:

```
lrf1 = model2.coef_[0][0] * 40 + model2.coef_[0][1]*30 + model2.coef_[0][2]*150 + model2.in
from scipy.special import expit
if expit(lrf1) > 0.5:
    print(expit(lrf1))
    print('YES, he will become diabetic')
else:
    print("NO, he will not be diabetic")
```

```
[0.53419838]
YES, he will become diabetic
```

In [28]:

```
model2.predict_proba([[150,30,40]])
```

Out[28]:

```
array([[0.53053646, 0.46946354]])
```

Step5. [Build LoR model with all features]

In [29]:

```
aX = hos.drop('Outcome',axis=1)
```

In [30]:

```
model3 = LogisticRegression()
```

In [31]:

```
from sklearn.model_selection import train_test_split
```

In [111]:

```
X_train,X_test,y_train,y_test = train_test_split(aX,y,train_size=0.8,test_size=0.2)
```

In [112]:

```
model3.fit(X_train,y_train)
```

```
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\linear_
model\_logistic.py:763: 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 (https://scik
it-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-regre
ssion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression)
  n_iter_i = _check_optimize_result(
```

Out[112]:

```
LogisticRegression()
```

In [113]:

```
y_test
```

Out[113]:

```
223    0
80     0
397    1
38     1
175    1
      ..
30     0
53     1
400    1
142    0
344    0
Name: Outcome, Length: 154, dtype: int64
```

In [114]:

```
y_pred = model3.predict(X_test)
y_pred
```

Out[114]:

```
array([1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0,
       1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0,
       0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0,
       0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
      dtype=int64)
```

In [115]:

```
from sklearn.metrics import roc_auc_score
```

In [116]:

```
print("LoR AUC  ",roc_auc_score(y_test,y_pred))
```

```
LoR AUC   0.7015306122448979
```

Step6. [Forward Selection Procedure]

In [37]:

```
type(hos.columns)
```

Out[37]:

```
pandas.core.indexes.base.Index
```

In [88]:

```
def auc(var,tar,df):
    fX = df[var]
    fy = df[tar]
    logreg = LogisticRegression()
    logreg.fit(fX,fy)
    pred=logreg.predict_proba(fX)[:,1]
    auc_val = roc_auc_score(y,pred)
    return auc_val
```

In [89]:

```
auc(["BMI","Glucose"],["Outcome"],hos)
```

```
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils\v
alidation.py:63: DataConversionWarning: A column-vector y was passed when a
1d array was expected. Please change the shape of y to (n_samples, ), for ex
ample using ravel().
  return f(*args, **kwargs)
```

Out[89]:

```
0.8109328358208956
```

In [90]:

```
auc(['Pregnancies', 'BloodPressure', 'SkinThickness'],["Outcome"],hos)
```

```
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils\v
alidation.py:63: DataConversionWarning: A column-vector y was passed when a
1d array was expected. Please change the shape of y to (n_samples, ), for ex
ample using ravel().
  return f(*args, **kwargs)
```

Out[90]:

```
0.6444962686567164
```

In [91]:

```
def next_best(current,cand,tar,df):
    best_auc = -1
    best_var = None
    for i in cand:
        auc_v = auc(current+[i],tar,df)
        if auc_v>=best_auc:
            best_auc = auc_v
            best_var = i
    return best_var
```

In [92]:

```
tar = ["Outcome"]
current = ['Insulin','BMI', 'DiabetesPedigreeFunction', 'Age']
cand = ['Pregnancies', 'BloodPressure', 'SkinThickness']
next_var = next_best(current,cand,tar,hos)
print(next_var)
```

```
SkinThickness

C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils\v
alidation.py:63: DataConversionWarning: A column-vector y was passed when a
1d array was expected. Please change the shape of y to (n_samples, ), for ex
ample using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils\v
alidation.py:63: DataConversionWarning: A column-vector y was passed when a
1d array was expected. Please change the shape of y to (n_samples, ), for ex
ample using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils\v
alidation.py:63: DataConversionWarning: A column-vector y was passed when a
1d array was expected. Please change the shape of y to (n_samples, ), for ex
ample using ravel().
  return f(*args, **kwargs)
```

In [93]:

```
tar = ["Outcome"]
current = []
cand = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin','BMI', 'Diabe
max_num = 5
num_it = min(max_num,len(cand))
for i in range(0,num_it):
    next_var = next_best(current,cand,tar,hos)
    current = current + [next_var]
    cand.remove(next_var)
    print("Variable added in step " + str(i+1)  + " is " + next_var + ".")
print(current)
```

```
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils
\validation.py:63: DataConversionWarning: A column-vector y was passed whe
n a 1d array was expected. Please change the shape of y to (n_samples, ),
for example using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils
\validation.py:63: DataConversionWarning: A column-vector y was passed whe
n a 1d array was expected. Please change the shape of y to (n_samples, ),
for example using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils
\validation.py:63: DataConversionWarning: A column-vector y was passed whe
n a 1d array was expected. Please change the shape of y to (n_samples, ),
for example using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils
\validation.py:63: DataConversionWarning: A column-vector y was passed whe
n a 1d array was expected. Please change the shape of y to (n_samples, ),
for example using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils
\validation.py:63: DataConversionWarning: A column-vector y was passed whe
n a 1d array was expected. Please change the shape of y to (n_samples, ),
for example using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils
\validation.py:63: DataConversionWarning: A column-vector y was passed whe
n a 1d array was expected. Please change the shape of y to (n_samples, ),
for example using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils
\validation.py:63: DataConversionWarning: A column-vector y was passed whe
n a 1d array was expected. Please change the shape of y to (n_samples, ),
for example using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils
\validation.py:63: DataConversionWarning: A column-vector y was passed whe
n a 1d array was expected. Please change the shape of y to (n_samples, ),
for example using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils
\validation.py:63: DataConversionWarning: A column-vector y was passed whe
n a 1d array was expected. Please change the shape of y to (n_samples, ),
for example using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils
```

```
\validation.py:63: DataConversionWarning: A column-vector y was passed whe
n a 1d array was expected. Please change the shape of y to (n_samples, ),
for example using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils
\validation.py:63: DataConversionWarning: A column-vector y was passed whe
n a 1d array was expected. Please change the shape of y to (n_samples, ),
for example using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils
\validation.py:63: DataConversionWarning: A column-vector y was passed whe
n a 1d array was expected. Please change the shape of y to (n_samples, ),
for example using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils
\validation.py:63: DataConversionWarning: A column-vector y was passed whe
n a 1d array was expected. Please change the shape of y to (n_samples, ),
for example using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils
\validation.py:63: DataConversionWarning: A column-vector y was passed whe
n a 1d array was expected. Please change the shape of y to (n_samples, ),
for example using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils
\validation.py:63: DataConversionWarning: A column-vector y was passed whe
n a 1d array was expected. Please change the shape of y to (n_samples, ),
for example using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils
\validation.py:63: DataConversionWarning: A column-vector y was passed whe
n a 1d array was expected. Please change the shape of y to (n_samples, ),
for example using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils
\validation.py:63: DataConversionWarning: A column-vector y was passed whe
n a 1d array was expected. Please change the shape of y to (n_samples, ),
for example using ravel().
  return f(*args, **kwargs)
```

```
Variable added in step 1 is Glucose.
Variable added in step 2 is BMI.
Variable added in step 3 is Pregnancies.
```

```
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils
\validation.py:63: DataConversionWarning: A column-vector y was passed whe
n a 1d array was expected. Please change the shape of y to (n_samples, ),
for example using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils
\validation.py:63: DataConversionWarning: A column-vector y was passed whe
n a 1d array was expected. Please change the shape of y to (n_samples, ),
for example using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils
\validation.py:63: DataConversionWarning: A column-vector y was passed whe
n a 1d array was expected. Please change the shape of y to (n_samples, ),
for example using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils
\validation.py:63: DataConversionWarning: A column-vector y was passed whe
```

```
n a 1d array was expected. Please change the shape of y to (n_samples, ),
for example using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils
\validation.py:63: DataConversionWarning: A column-vector y was passed whe
n a 1d array was expected. Please change the shape of y to (n_samples, ),
for example using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils
\validation.py:63: DataConversionWarning: A column-vector y was passed whe
n a 1d array was expected. Please change the shape of y to (n_samples, ),
for example using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils
\validation.py:63: DataConversionWarning: A column-vector y was passed whe
n a 1d array was expected. Please change the shape of y to (n_samples, ),
for example using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils
\validation.py:63: DataConversionWarning: A column-vector y was passed whe
n a 1d array was expected. Please change the shape of y to (n_samples, ),
for example using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils
\validation.py:63: DataConversionWarning: A column-vector y was passed whe
n a 1d array was expected. Please change the shape of y to (n_samples, ),
for example using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils
\validation.py:63: DataConversionWarning: A column-vector y was passed whe
n a 1d array was expected. Please change the shape of y to (n_samples, ),
for example using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils
\validation.py:63: DataConversionWarning: A column-vector y was passed whe
n a 1d array was expected. Please change the shape of y to (n_samples, ),
for example using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils
\validation.py:63: DataConversionWarning: A column-vector y was passed whe
n a 1d array was expected. Please change the shape of y to (n_samples, ),
for example using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils
\validation.py:63: DataConversionWarning: A column-vector y was passed whe
n a 1d array was expected. Please change the shape of y to (n_samples, ),
for example using ravel().
  return f(*args, **kwargs)
Variable added in step 4 is DiabetesPedigreeFunction.
Variable added in step 5 is BloodPressure.
['Glucose', 'BMI', 'Pregnancies', 'DiabetesPedigreeFunction', 'BloodPressur
e']
```

Step7. [Plot Line graph of AUC values and select cut-off]

In [144]:

```
X_train,X_test,y_train,y_test = train_test_split(aX,y,test_size = 0.5,stratify =y)
```

In [156]:

```
pred2 = model3.predict_proba(X_test)
```

In [145]:

```
train = pd.concat([X_train,y_train], axis=1)
test = pd.concat([X_test,y_test], axis=1)
```

In [146]:

```
def auc_train_test(variables,target,train,test):
    X_train = train[variables]
    X_test = test[variables]
    Y_train = train[target]
    Y_test = test[target]
    logreg = LogisticRegression()

    # Fit the model on train data
    logreg.fit(X_train, Y_train)

    # Calculate the predictions both on train and test data
    predictions_train = logreg.predict_proba(X_train)[:,1]
    predictions_test = logreg.predict_proba(X_test)[:,1]

    # Calculate the AUC both on train and test data
    auc_train = roc_auc_score(Y_train, predictions_train)
    auc_test = roc_auc_score(Y_test,predictions_test)
    return(auc_train, auc_test)
```

In [147]:

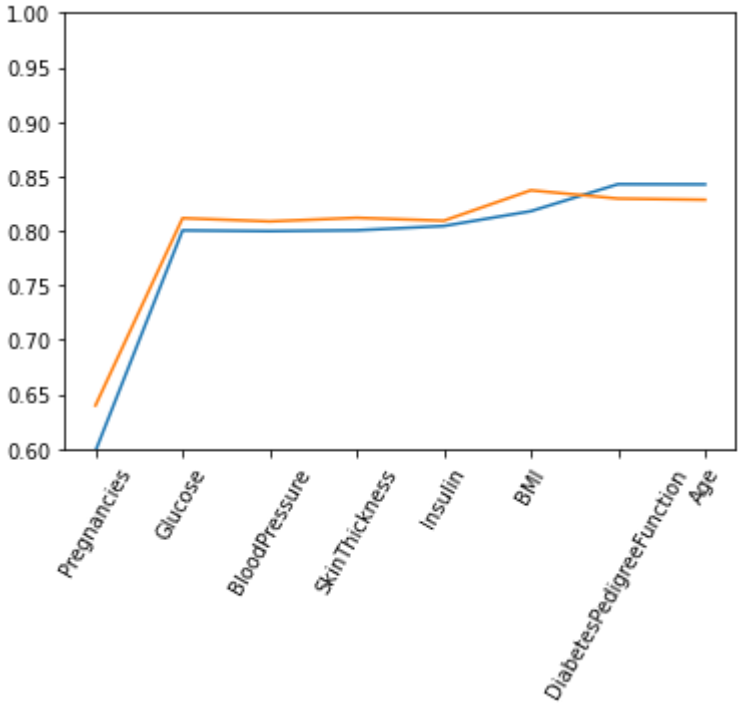
```
auc_values_train = []
auc_values_test = []
variables_evaluate = []

# Iterate over the variables in variables
for v in aX.columns:

    # Add the variable
    variables_evaluate.append(v)

    # Calculate the train and test AUC of this set of variables
    auc_train, auc_test = auc_train_test(variables_evaluate,["Outcome"],train,test)

    # Append the values to the lists
    auc_values_train.append(auc_train)
    auc_values_test.append(auc_test)
```

```
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils\v
alidation.py:63: DataConversionWarning: A column-vector y was passed when a
1d array was expected. Please change the shape of y to (n_samples, ), for ex
ample using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils\v
alidation.py:63: DataConversionWarning: A column-vector y was passed when a
1d array was expected. Please change the shape of y to (n_samples, ), for ex
ample using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils\v
alidation.py:63: DataConversionWarning: A column-vector y was passed when a
1d array was expected. Please change the shape of y to (n_samples, ), for ex
ample using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils\v
alidation.py:63: DataConversionWarning: A column-vector y was passed when a
1d array was expected. Please change the shape of y to (n_samples, ), for ex
ample using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils\v
alidation.py:63: DataConversionWarning: A column-vector y was passed when a
1d array was expected. Please change the shape of y to (n_samples, ), for ex
ample using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils\v
alidation.py:63: DataConversionWarning: A column-vector y was passed when a
1d array was expected. Please change the shape of y to (n_samples, ), for ex
ample using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils\v
alidation.py:63: DataConversionWarning: A column-vector y was passed when a
1d array was expected. Please change the shape of y to (n_samples, ), for ex
ample using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\linear_
model\_logistic.py:763: 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 (https://scik
it-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-regre
ssion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression)
  n_iter_i = _check_optimize_result(
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\utils\v
alidation.py:63: DataConversionWarning: A column-vector y was passed when a
 1d array was expected. Please change the shape of y to (n_samples, ), for e
xample using ravel().
  return f(*args, **kwargs)
C:\Users\Arzoo Sah\anaconda3\envs\notebook\lib\site-packages\sklearn\linear_
model\_logistic.py:763: 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 (https://scik
it-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-regre
ssion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression)
  n_iter_i = _check_optimize_result(
```

In [148]:

```
# Make plot of the AUC values
import matplotlib.pyplot as plt
import numpy as np

x = np.array(range(0,len(auc_values_train)))
my_train = np.array(auc_values_train)
my_test = np.array(auc_values_test)
plt.xticks(x,aX.columns,rotation=60)
plt.plot(x,my_train)
plt.plot(x,my_test)
plt.ylim((0.6,1.0))
plt.show()
```



Step8. [Draw Cumulative Gain Chart and Lift Chart]

In [66]:

```
!pip install scikit-plot
```

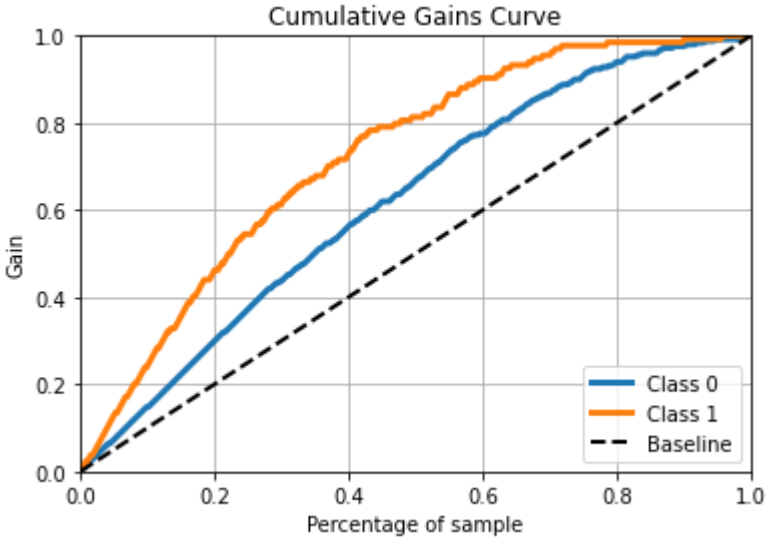
...

In [149]:

```
import scikitplot as skplt
```

In [158]:

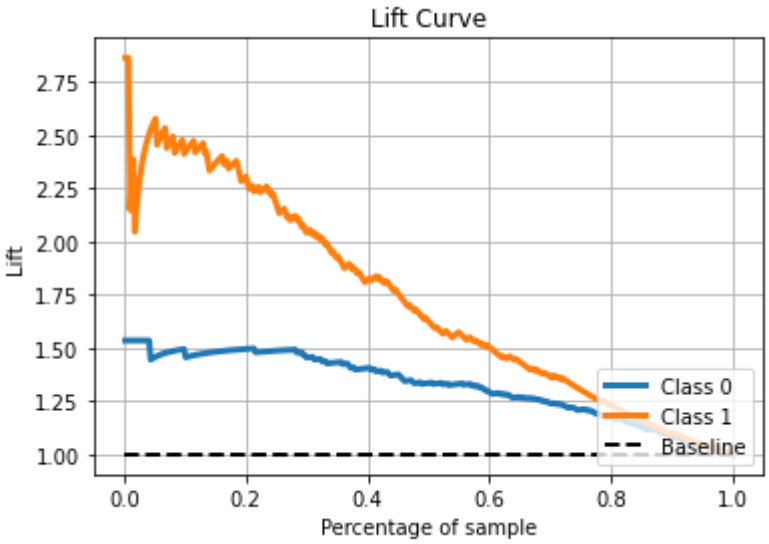
```
skplt.metrics.plot_cumulative_gain(y_test,pred2)
plt.show()
```



In [159]:

```
plt.figure(figsize=(7,7))
skplt.metrics.plot_lift_curve(y_test,pred2)
plt.show()
```

```
<Figure size 504x504 with 0 Axes>
```



In [ ]:

lab:6

```
import pandas as pd
data= pd.read_csv ("diabetes.csv")
data.head()
  data.shape
  data.columns
  data.info
  data.value_counts()
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
sns.heatmap(data.head(20),(cmap='Reds',annot=True
                                    linewidth=.5)
    X=data[['Age']]
    Y= data['outcome']
from sklearn.linear_model import logistic.Regression
    lrm1 = logisticRegression()
    lrm2 .fit(X,Y)
    lrm1.coeff
  lrm1.intercept
  years-old = [[60]]
  lrm1.predict(year-old)
lrb = lrm1.coef_ * 60 + lrm1.intercept_
from scipy.special import expit
```

Lab6. Predictive Analytics for Hospitals

Objectives

In this lab, you will continue to work on Diabetes data from Lab4 and apply Predictive Analytics principles to recommend the course of action to be taken by the hospital.

Learning Outcomes

After completing this lab, you will be able to

- Perform prediction on single and multiple independent variables for the target
- Apply Forward Selection procedure to find the best features based on AUC scores
- Draw a line plot of AUC scores for variables and select them using cut-off
- Plot Gain curves and Life curves and interpret results

Business Use Case

As a Data Scientist, you are requested by the hospital to build a predictive model for them. The hospital will apply your predictive model to perform prediction on new patients whether they will become diabetic or not. Further, based on the prediction, the hospital will recommend action plan such as developing a diet plan, physical activities and others.

Step1. [Import dataset]

- Using Pandas, import **"diabetes.csv"** file and print properties such as head, shape, columns, dtype, info and value_counts.

Step2. [Identify relationships between feature]

- Create a Heatmap for the dataset and understand the data

Step3. [Prediction using one feature]

Will older people become diabetic?

- Create LogisticRegression model, train with "Age" as **X** and "Outcome" feature as **y**.
- Print model parameter values: coef_ and intercept_
- **Query:** A person is 60 years old. Will he be diabetic?
- Use model parameters and find function value. Your code will be as below.

```
lrf = logreg.coef_ * 60 + logreg.intercept_
from scipy.special import expit
expit(lrf)
```

If your output > 0.5, YES he will become diabetic. Otherwise, NO, he will not be diabetic.

Step4. [Prediction using many features]

Will Glucose, BMI and Age values make someone diabetic?

- Select the three features 'Glucose', 'BMI' and 'Age' from your dataset, call it as **X**
- Create a new LogisticRegression model, train with X and 'Outcome' as **y**.
- Query: For a person, glucose=150, bmi=30, age=40. Will he be diabetic?
- Find the value of expit() as before. Output will be: 0.5208271643241003

Note: You can also verify expit() output value as below

```
logreg.predict_proba([[150, 30, 40]])
```

Step5. [Build LoR model with all features]

- Create LoR model, train it with X_train and y_train values
- Now, compute and print its AUC value
- Can we get this AUC value with limited set of good features?. Yes, we are going to find with 'Forward Selection Procedure'.

```
if expit(lrf) > 0.5:
    print("YES, he will become diabetic")
else:
    print("No, he will not be diabetic")

X1 = data[["Age", "BMI", "Glucose"]]
lm2 = logistic.Regression()
lr m2.fit(X1, y)
l2 M2.predict([[150, 30, 40]])
l2M2.predict_proba[[[150, 30, 40]]]

import warnings
warnings.filterwarnings("ignore")
X3 = data.drop("Outcome", axis=1)
lrm3 = logistic Regression()

from sklearn.model_selection train_test_size_as20
X_train, X_test, Y_train, Y_test = train_test_split
                    (X3, Y, train_size=0.8, test_size=0.2
lrm3.fit(x_train, Y_train)
y_pred = lrm3.predict(x_test)
from sklearn.metrics import roc_auc_score.
print("LOR Ave", roc_auc_score(y_test, y_pred)
type(data.columns)
def get_auc(var, tar, df):
```

Step6. [Forward Selection Procedure]

Now, you have to find and select a good set of features with the best AUC score. The algorithm is

Forward stepwise variable selection procedure

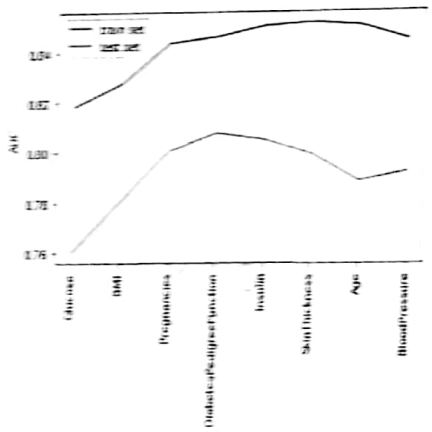
1. Empty set
2. Find best variable v
3. Find best variable v in combination with v
4. Find best variable v in combination with v , v
5. ...
6. Until all variables are added or until predefined number of variables is added)

Implementation Steps of the forward stepwise procedure

1. Define a function get_auc() that calculates AUC given a certain set of variables
2. Define a function best_next() that returns next best variable in combination with current variables
3. Loop until desired number of variables

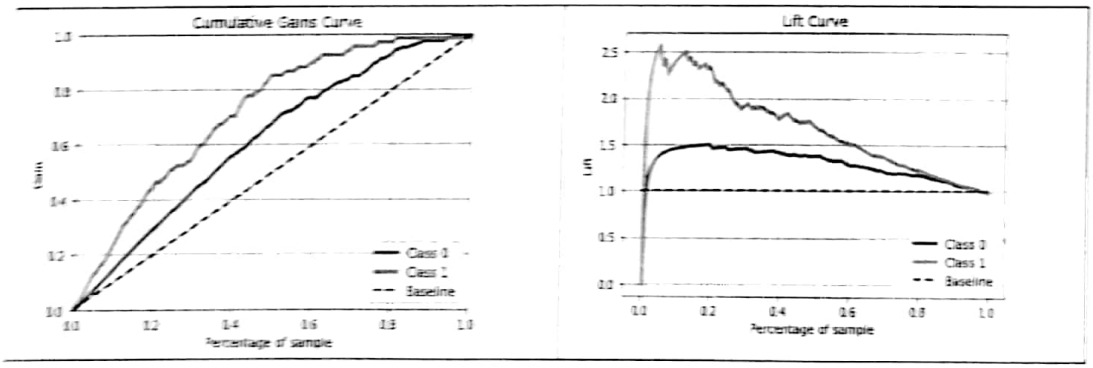
Step7. [Plot Line graph of AUC values and select cut-off]

- Split your dataset equally for training and testing
- Plot AUC values for each variable. The graph will appear as below.



The cut-off line can be drawn at the 4th variable. It gives the best AUC score around 0.81. Therefore, it will be sufficient to use only the first 4 features for training and testing. Otherwise, AUC is going down gradually.

Step8. [Draw Cumulative Gain Chart and Lift Chart]



Interpret these curves.

```
    dx = df[[var]]
    dy = df[tar]
  logreg = logistic Regression()
from sklearn.metrics import roc_auc_score
print ["Log Auc", roc_auc_score(y_test, y_pred)]
      type(data.columns)
  def get_auc(var, tar, df):
      dx = df[var]
      dy = df[tar]
    logreg = Logistic Regression[]
    log.reg.fit(dx, dy)
  pred = log.reg.predict_proba(dx)[:, 1]
  auc_val = roc_auc_score(dy, pred())
  return auc_val)
get_auc(["BHI", "Glucose"], ["Outcome"], data)
get.auc(['pregancies', 'Blood Pressure', 'Skin thickness'],
                        ["Outcome"], data)
  def next_best(current_card, tar, df):
      best_auc = -1
      best_var = None
      for in in card:
          auc_v = get_auc(current + [i], tar, df)
            if auc_v > best_auc:
            best_var = None
```

NOTES

```
        best_var = i
        return best_var
    tar = ["Outcome"]
Current = ['Insulin', 'BMI', 'Diabetes pedigreefunction',
                                                'Age']
Card = ['Pregnancies', 'Blood pressure', 'Skin Thickness']
next_var = next_best current, Card, tar, data)
print (next_var)
tar = ["Outcome"]
Current = [ ]
Card = ["Pregnancies', 'Glucose', 'Blood pressure', 'Skin',
        'Thickness', 'Insulin', 'BMI', 'Diabetes Pedigree
        function', 'Age']
max_num = 5
num_it = min (max_num, len Card))
for i in range (0, num_it):
    next_var = next_best (Current, Card, tar, data)
    Current = Current + [next_var]
    Card.remove (next_var)
    print ('variable added in step' + str(i+1) + "is" +
                                        next_var + ".")
print (current)
X_train, X_test, Y_train, Y_test = train_test_split
                        (X, Y, test_size = 0.5, stratify=Y)
```

```
pred 2 = lrm 3. predict_proba(X_test)

train = pd.concat([X_train, Y_train], axis=1)
test = pd.concat([X_test, Y_test], axis=1)
def auc_train_test(variables, target, train, test):
        X_train = train[variables]
        X_test = train[variables]
        Y_train = train[target]
        Y_test = test[target]
        logreg = LogisticRegression()
log.reg.fit(X_train, Y_train).
predictions_train = logreg.predict_proba(X_train)
                                          [:, 1]

predictions test = logreg_predict_proba(X_test)[:,1]

auc_train = roc_auc_score(Y_train, predictions_train)
auctest = roc_auc_score(Y_test, prediction_test)
return (auc_train, auc_test)
  import matplotlib.pyplot as plt
  import numpy as np
  X = np.array(range(0, len(auc_values_train)))
my_train = np.array(auc_values_train)
my_test = np.array(auc_values_test)
  plt.xticks(X, X_3.columns, rotation = 90)
        plt.plot(X, my_train)
        plt.plot(X, my_test)
        plt.ylim(0.6, 1.0)  plt.show()
```

NOTES

```
Import scikitplot as skplt.
skplt.metrics.plot_cumulative_gain, (y_test, pred_a)
plt.show()
plt.figure(figsize=(7,7))
Skplt.metrics.plot_lift_curve(y_test, pred_a)
plt.show()
```

REPORT

Lab6.Predictive Analytics for Hospitals

In this lab we have to perform same logic which we have done in previous lab diabetic prediction. But in this case a small change in target variables.

Build a LoR model and perform prediction on single and multiple independent variables. Then here we perform forward selection procedure to find and select a good set of features with the best AUC score.

Then we are creating a function to calculate AUC for certain set of variables. First will create get_auc () function that should calculate the auc score for given variables in that function.

Similarly, we create best_next () function that should return best variable in combination with current variables. Then plotting AUC scores for variables also for gain curves and life curves.

Roll No: 205229133

Lab7. Loan Approval Classification using SVM

Objectives ¶

In this lab, you will build a classification model to classify the loan applicants into eligible applicants or not eligible applicants using Support Vector Machine.

Learning Outcomes

After completing this lab, you will be able to

- Apply data cleaning methods
- Perform EDA and understand who got their loans approved
- Do feature engineering with One Hot Encoding
- Create LinearSVC model, train and predict on the data
- Print accuracy, confusion matrix and classification report
- Compare LinearSVC model with SVC and SGDClassifier models

Business Use Case

Heber Housing Finance deals in all home loans. They have presence across all urban, semi urban and rural areas. Customer first apply for home loan after that company validates the customer eligibility for loan. However doing this manually takes a lot of time. Hence, it wants you to automate the loan approval process (real time) based on customer information. So you should identify all features and build a model that enable the company to approve the load application or not.

The dataset contains the details of 614 loan applicants, where each applicant is described with 12 features. Loan_Status is the target feature (ie., dependent variable) and all others are independent variables.

Step1. [Understand Data]. Using Pandas, import “train_loan.csv” file and print properties such as head, shape, columns, dtype, info and value_counts

In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt

train_data = pd.read_csv('train_loan.csv')
train_data.head()
```

Out[1]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapp
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	
4	LP001008	Male	No	0	Graduate	No	6000	

In [2]:

```
train_data.shape
```

Out[2]:

```
(614, 13)
```

In [3]:

```
train_data.columns
```

Out[3]:

```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmoun
t',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Statu
s'],
      dtype='object')
```

In [4]:

```
train_data.dtypes
```

Out[4]:

```
Loan_ID               object
Gender                object
Married               object
Dependents            object
Education             object
Self_Employed         object
ApplicantIncome        int64
CoapplicantIncome    float64
LoanAmount           float64
Loan_Amount_Term     float64
Credit_History       float64
Property_Area         object
Loan_Status           object
dtype: object
```

In [5]:

```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Loan_ID            614 non-null    object 
 1   Gender             601 non-null    object 
 2   Married            611 non-null    object 
 3   Dependents         599 non-null    object 
 4   Education          614 non-null    object 
 5   Self_Employed      582 non-null    object 
 6   ApplicantIncome    614 non-null    int64  
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object 
 12  Loan_Status        614 non-null    object 
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

In [6]:

```
train_data.value_counts
```

Out[6]:

```
<bound method DataFrame.value_counts of       Loan_ID  Gender Married Depend
ents     Education Self_Employed  \
0    LP001002    Male      No          0      Graduate            No
1    LP001003    Male     Yes          1      Graduate            No
2    LP001005    Male     Yes          0      Graduate           Yes
3    LP001006    Male     Yes          0  Not Graduate            No
4    LP001008    Male      No          0      Graduate            No
..        ...     ...     ...        ...           ...           ...
609  LP002978  Female      No          0      Graduate            No
610  LP002979    Male     Yes         3+      Graduate            No
611  LP002983    Male     Yes          1      Graduate            No
612  LP002984    Male     Yes          2      Graduate            No
613  LP002990  Female      No          0      Graduate           Yes

     ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0               5849                0.0         NaN             360.0
1               4583             1508.0       128.0             360.0
2               3000                0.0        66.0             360.0
3               2583             2358.0       120.0             360.0
4               6000                0.0       141.0             360.0
..               ...                ...         ...               ...
609             2900                0.0        71.0             360.0
610             4106                0.0        40.0             180.0
611             8072              240.0       253.0             360.0
612             7583                0.0       187.0             360.0
613             4583                0.0       133.0             360.0

     Credit_History Property_Area Loan_Status
0               1.0         Urban           Y
1               1.0         Rural           N
2               1.0         Urban           Y
3               1.0         Urban           Y
4               1.0         Urban           Y
..              ...           ...         ...
609             1.0         Rural           Y
610             1.0         Rural           Y
611             1.0         Urban           Y
612             1.0         Urban           Y
613             0.0     Semiurban           N

[614 rows x 13 columns]>
```

Step2. [Data Cleaning]

- Replace numbers as string by integer in “Dependents” column
- Fill missing data in categorical columns (Gender, Married, Dependents, Education, Self_Employed, Credit_History) by its mode value
- Handle missing values in numberical columns
- Drop Loan_ID column

In [7]:

```
train_data.Dependents.value_counts()
```

Out[7]:

```
0     345
1     102
2     101
3+     51
Name: Dependents, dtype: int64
```

In [8]:

```
#Replace numbers as string by integer in 'Dependents' column
def string(x):
    if x == '0':
        return 'bad'
    elif x == '1':
        return 'average'
    elif x == '2':
        return 'good'
    else:
        return 'excellent'
```

In [9]:

```
train_data['Dependents'] = train_data['Dependents'].apply(string)
```

In [10]:

```
train_data.isna().sum()
```

Out[10]:

```
Loan_ID               0
Gender               13
Married               3
Dependents            0
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

In [11]:

```
#categorical
train_data['Gender'].fillna(train_data['Gender'].mode()[0], inplace=True)
train_data['Married'].fillna(train_data['Married'].mode()[0], inplace=True)
train_data['Dependents'].fillna(train_data['Dependents'].mode()[0], inplace=True)
train_data['Loan_Amount_Term'].fillna(train_data['Loan_Amount_Term'].mode()[0], inplace=Tru
train_data['Credit_History'].fillna(train_data['Credit_History'].mode()[0], inplace=True)
train_data['Self_Employed'].fillna(train_data['Self_Employed'].mode()[0], inplace=True)
#numerical
train_data['LoanAmount'].fillna(train_data['LoanAmount'].mean(), inplace=True)
```

In [12]:

```
train_data=train_data.drop(['Loan_ID'],axis=1)
```

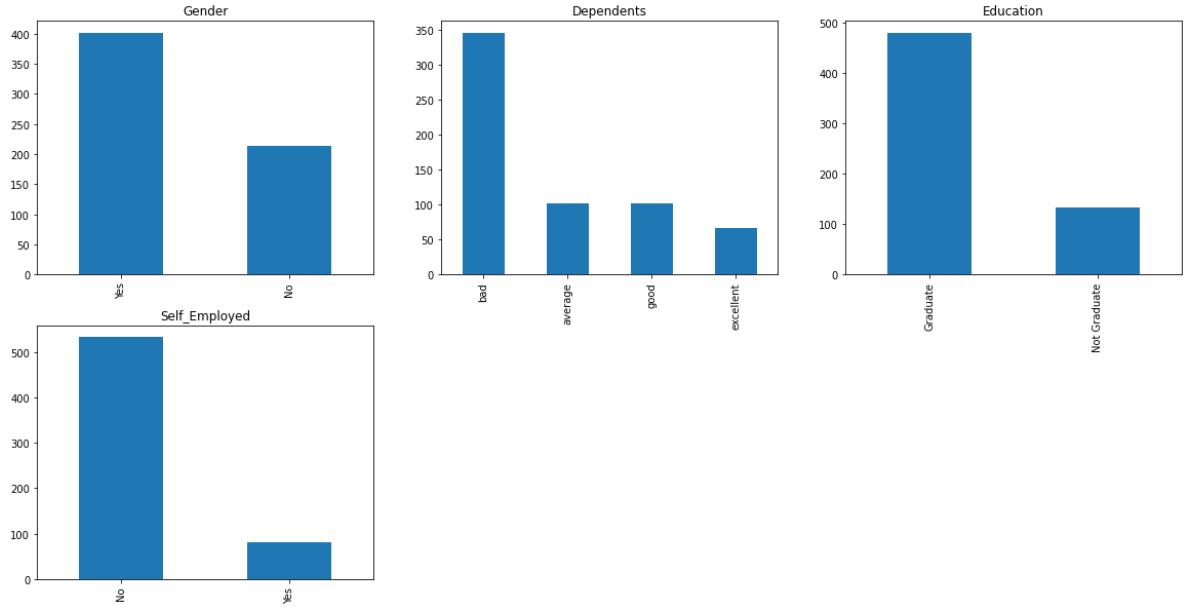
Step3. [OPTIONAL: Exploratory Data Analysis - Who got their loan approved]

Draw count plot for

- Married?
- Dependants?
- Graduates?
- Self-employed?

In [13]:

```
plt.subplot(231)
train_data['Married'].value_counts().plot(kind='bar',title='Gender',figsize = (20,10))

plt.subplot(232)
train_data['Dependents'].value_counts().plot(kind='bar',title='Dependents')

plt.subplot(233)
train_data['Education'].value_counts().plot(kind='bar',title='Education')

plt.subplot(234)
train_data['Self_Employed'].value_counts().plot(kind='bar',title='Self_Employed')

plt.show()
```



Step4. [Extract X and y] from the dataframe

In [14]:

```
X = train_data.drop(['Loan_Status'],axis=1)
y = train_data.Loan_Status
```

Step5. [One Hot Encoding]

Perform OHE on categorical columns, use this method: X = pd.get_dummies(X)

In [15]:

```
import warnings
warnings.filterwarnings('ignore')
```

In [16]:

```
X = pd.get_dummies(X)
```

Step6. [Model Building]

- Split X and y for training and testing
- Using StandardScaler, fit_transform on X_train and transform on X_test values
- create LinearSVC model, train and test
- print accuracy value
- Print confusion matrix between y_test and y_pred
- Print classification_report

In [17]:

```
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)
```

In [18]:

```
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
```

In [19]:

```
ss = scale.fit_transform(X_train)
ss1 = scale.transform(X_test)
```

In [20]:

```
from sklearn.svm import LinearSVC
model = LinearSVC()
model.fit(ss,y_train)
```

Out[20]:

```
LinearSVC()
```

In [21]:

```
Lsvc_y_pred = model.predict(ss1)
Lsvc_y_pred
```

Out[21]:

```
array(['Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y',
       'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y',
       'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y',
       'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N',
       'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'N', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N',
       'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y'], dtype=object)
```

In [22]:

```
from sklearn.metrics import accuracy_score,confusion_matrix, classification_report
accuracy_score(y_test,Lsvc_y_pred)
```

Out[22]:

```
0.8324324324324325
```

In [23]:

```
confusion_matrix(y_test,Lsvc_y_pred)
```

Out[23]:

```
array([[ 22,  29],
       [  2, 132]], dtype=int64)
```

In [24]:

```
print(classification_report(y_test,Lsvc_y_pred))
```

```
              precision    recall  f1-score   support

           N       0.92      0.43      0.59        51
           Y       0.82      0.99      0.89       134

    accuracy                           0.83       185
   macro avg       0.87      0.71      0.74       185
weighted avg       0.85      0.83      0.81       185
```

Step7. [Performance Comparisons]

1. Compare the performance of LinearSVC against LogisticRegression

In [25]:

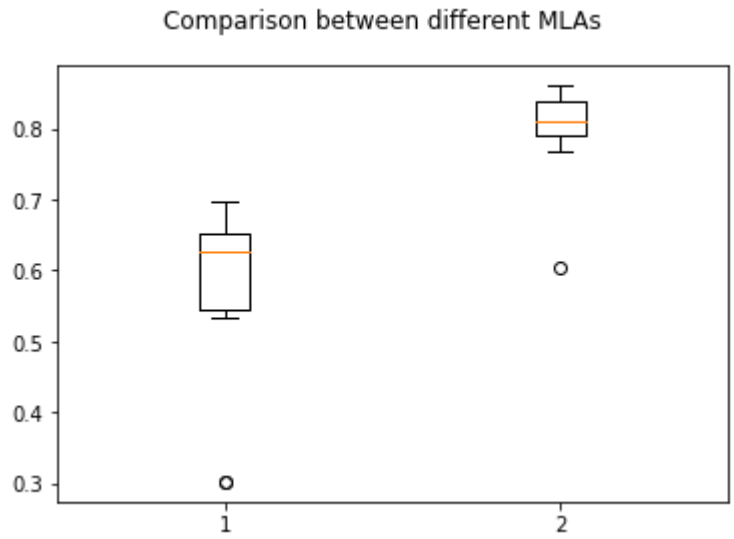
```
from sklearn.linear_model import LogisticRegression

lgr= LogisticRegression()
lgr.fit(ss,y_train)
lr_y_pred = lgr.predict(ss1)

from sklearn.svm import LinearSVC

l_svc = LinearSVC()
l_svc.fit(ss,y_train)
lsvc_y_pred = l_svc.predict(ss1)

print("LogisticRegression:",accuracy_score(y_test,lr_y_pred))
print("LinearSVC         :",accuracy_score(y_test,lsvc_y_pred))
```

```
LogisticRegression: 0.8324324324324325
LinearSVC         : 0.8324324324324325
```

In [26]:

```
from sklearn import svm,model_selection
models = []
models.append(('SVC', LinearSVC()))
models.append(('LR', LogisticRegression()))

# evaluate each model in turn
results = []
names=[]
scoring = 'accuracy'
for name,model in models:
    kfold = model_selection.KFold(n_splits=10)
    cv_results = model_selection.cross_val_score(model, X_train, y_train, cv=kfold, scoring
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
# boxplot algorithm comparison
fig = plt.figure()
fig.suptitle('Comparison between different MLAs')
ax = fig.add_subplot(111)
plt.boxplot(results)
plt.show()
```

```
SVC: 0.564120 (0.138692)
LR: 0.797231 (0.070487)
```

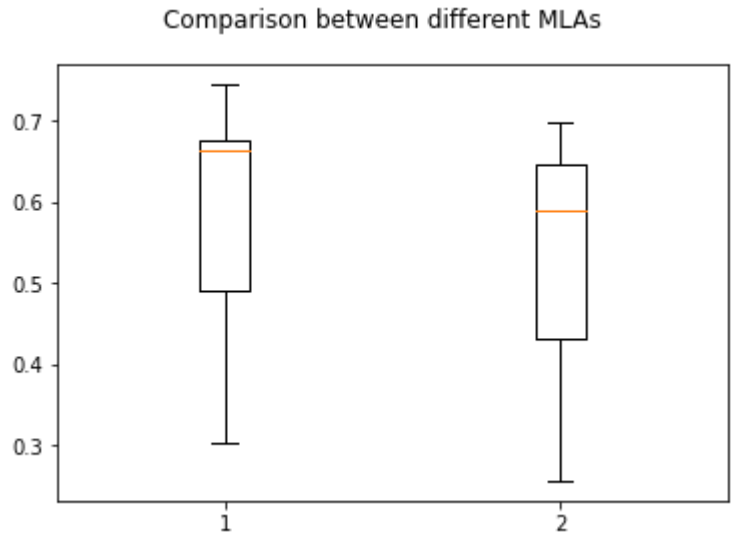


2. Compare the performance of LinearSVC against SGDClassifier

In [27]:

```
from sklearn.linear_model import SGDClassifier

modelss = []
modelss.append(('SVC', LinearSVC()))
modelss.append(('SGD', SGDClassifier()))

# evaluate each model in turn
results = []
names=[]
scoring = 'accuracy'
for name,model in modelss:
    kfold = model_selection.KFold(n_splits=10)
    cv_results = model_selection.cross_val_score(model, X_train, y_train, cv=kfold, scoring
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
# boxplot algorithm comparison
fig = plt.figure()
fig.suptitle('Comparison between different MLAs')
ax = fig.add_subplot(111)
plt.boxplot(results)
plt.show()
```

```
SVC: 0.577796 (0.156098)
SGD: 0.529236 (0.153889)
```



In [28]:

```
from sklearn.linear_model import SGDClassifier

sgd = SGDClassifier()
sgd.fit(ss,y_train)
sgdc_y_pred = sgd.predict(ss1)

from sklearn.svm import LinearSVC

l_svc = LinearSVC()
l_svc.fit(ss,y_train)
lsvc_y_pred = l_svc.predict(ss1)

print("SGDClassifier:", accuracy_score(y_test,sgdc_y_pred))
print("LinearSVC    :",accuracy_score(y_test,lsvc_y_pred))
```

```
SGDClassifier: 0.6324324324324324
LinearSVC    : 0.8324324324324325
```

3. Compare LinearSVC against SVC with various kernels such as 'linear', 'poly', 'rbf' and 'sigmoid'

In [29]:

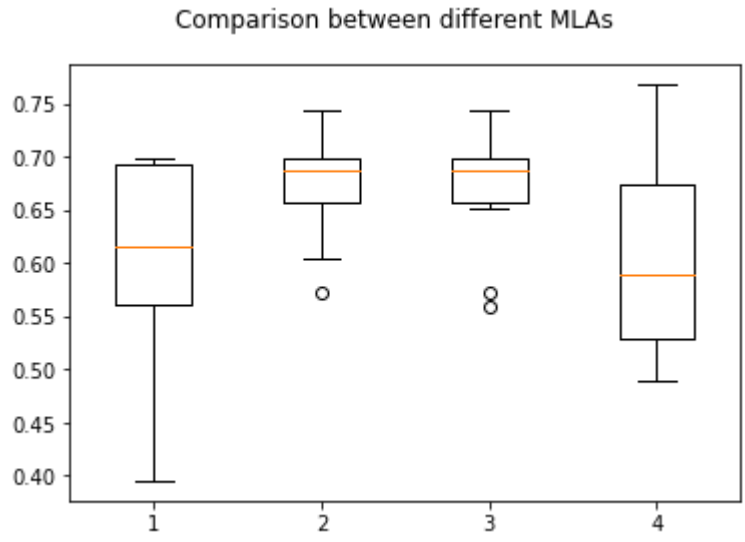
```
from sklearn.svm import SVC

l_svc = LinearSVC()
l_svc.fit(ss,y_train)
lsvc_y_pred = l_svc.predict(ss1)


poly_svc = svm.SVC(kernel='poly', C = 1.0)
poly_svc.fit(ss,y_train)
psvc_y_pred=poly_svc.predict(ss1)


rbf_svc = svm.SVC(kernel='rbf', C = 1.0)
rbf_svc.fit(ss,y_train)
rbfsvc_y_pred=rbf_svc.predict(ss1)


sigmoid_svc = svm.SVC(kernel='sigmoid', C = 1.0)
sigmoid_svc.fit(ss,y_train)
sigsvc_y_pred=sigmoid_svc.predict(ss1)


print("LinearSVC   :",accuracy_score(y_test,lsvc_y_pred))
print("poly SVC    :",accuracy_score(y_test,psvc_y_pred))
print("rbf SVC     :",accuracy_score(y_test,rbfsvc_y_pred))
print("Sigmoid SVC :",accuracy_score(y_test,sigsvc_y_pred))
```

```
LinearSVC   : 0.8324324324324325
poly SVC    : 0.8162162162162162
rbf SVC     : 0.8324324324324325
Sigmoid SVC : 0.8054054054054054
```

In [30]:

```
modelsss = []
modelsss.append(('SVC', LinearSVC()))
modelsss.append(('SVC POLY', svm.SVC(kernel='poly', C = 1.0)))
modelsss.append(('SVC rbf', svm.SVC(kernel='rbf', C = 1.0)))
modelsss.append(('SVC POLY', svm.SVC(kernel='sigmoid', C = 1.0)))

# evaluate each model in turn
results = []
names=[]
scoring = 'accuracy'
for name,model in modelsss:
    kfold = model_selection.KFold(n_splits=10)
    cv_results = model_selection.cross_val_score(model, X_train, y_train, cv=kfold, scoring
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
# boxplot algorithm comparison
fig = plt.figure()
fig.suptitle('Comparison between different MLAs')
ax = fig.add_subplot(111)
plt.boxplot(results)
plt.show()
```

```
SVC: 0.605980 (0.091265)
SVC POLY: 0.671096 (0.047891)
SVC rbf: 0.666445 (0.055735)
SVC POLY: 0.603710 (0.092483)
```



4. Interpret the results

In [31]:

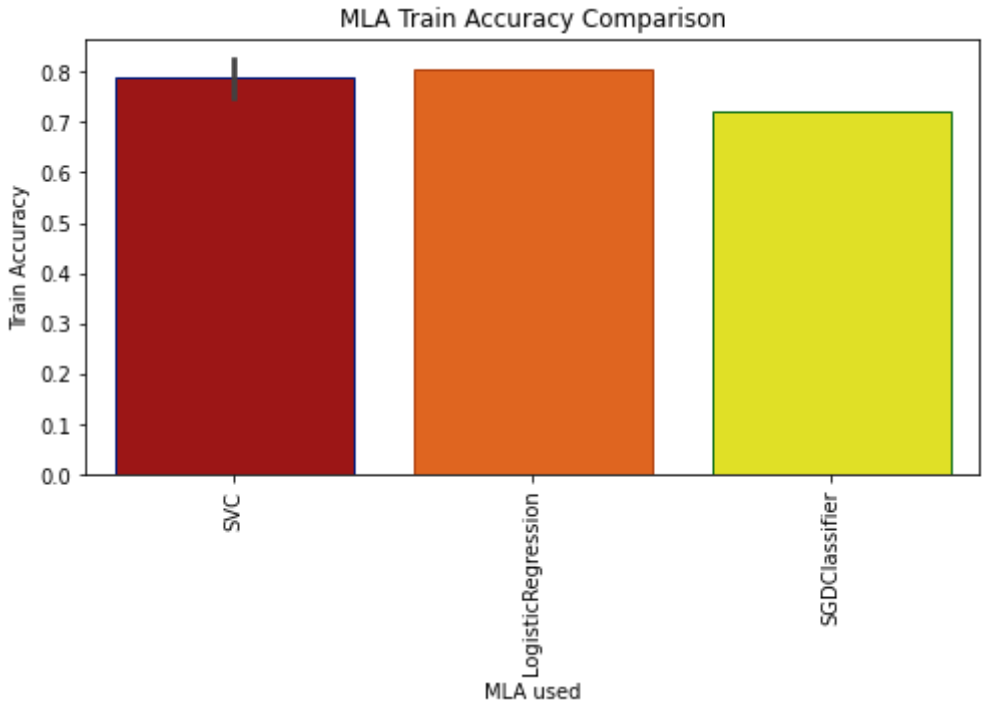
```
import numpy as np
from sklearn.metrics import roc_curve
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import auc

MLA  = [model,lgr,sgd,poly_svc,rbf_svc,sigmoid_svc]
MLA_columns = []
MLA_compare = pd.DataFrame(columns = MLA_columns)

row_index = 0
for alg in MLA:

    predicted = alg.fit(ss, y_train).predict(ss1)
    predicted=np.where(predicted=='Y',1,0)
    y_testb=np.where(y_test=='Y',1,0)
    fp, tp, th = roc_curve(y_testb, predicted)
    MLA_name = alg.__class__.__name__
    MLA_compare.loc[row_index,'MLA used'] = MLA_name
    MLA_compare.loc[row_index, 'Train Accuracy'] = round(alg.score(ss,y_train), 4)
    MLA_compare.loc[row_index, 'Test Accuracy'] = round(alg.score(ss1,y_test), 4)
    MLA_compare.loc[row_index, 'Precission'] = precision_score(y_testb, predicted)
    MLA_compare.loc[row_index, 'Recall'] = recall_score(y_testb, predicted)
    MLA_compare.loc[row_index, 'AUC'] = auc(fp, tp)
    row_index+=1

MLA_compare
```

Out[31]:

	MLA used	Train Accuracy	Test Accuracy	Precission	Recall	AUC
0	SVC	0.7506	0.8054	0.810127	0.955224	0.683494
1	LogisticRegression	0.8042	0.8324	0.819876	0.985075	0.708224
2	SGDClassifier	0.7203	0.7514	0.818841	0.843284	0.676544
3	SVC	0.8368	0.8162	0.820513	0.955224	0.703102
4	SVC	0.8135	0.8324	0.819876	0.985075	0.708224
5	SVC	0.7506	0.8054	0.810127	0.955224	0.683494

In [32]:

```
import seaborn as sns
# Creating plot to show the train accuracy
plt.subplots(figsize=(8,4))
sns.barplot(x="MLA used", y="Train Accuracy",data=MLA_compare,palette='hot',edgecolor=sns.c
plt.xticks(rotation=90)
plt.title('MLA Train Accuracy Comparison')
plt.show()
```



Step :1

```
import pandas as pd
import matplotlib.pyplot as plt.
train_data= pd.read_csv('train_loan.csv')
train_data.head()
train_data.shape
train_data.columns
train_data.dtypes
train_data.info()
train_data.value_counts
train_data.Dependants.value_counts()
#Replace numbers as string by integer in 'Dependants' columns
def string(x):
    if x == '0':
        return 'bad'
    elif x=='1':
        return 'average'
    elif x == '2':
        return 'good'
    else:
        return 'excellent'
train_data['Dependents'] = train_data['Dependents'].apply(string)
train_data.isna().sum()
```

Lab7. Loan Approval Classification using SVM

Objectives

In this lab, you will build a classification model to classify the loan applicants into eligible applicants or not eligible applicants using Support Vector Machine.

Learning Outcomes

After completing this lab, you will be able to

- Apply data cleaning methods
- Perform EDA and understand who got their loans approved
- Do feature engineering with One Hot Encoding
- Create LinearSVC model, train and predict on the data
- Print accuracy, confusion matrix and classification report
- Compare LinearSVC model with SVC and SGDClassifier models

Business Use Case

Heber Housing Finance deals in all home loans. They have presence across all urban, semi urban and rural areas. Customer first apply for home loan after that company validates the customer eligibility for loan. However doing this manually takes a lot of time. Hence, it wants you to automate the loan approval process (real time) based on customer information. So you should identify all features and build a model that enable the company to approve the load application or not.

The dataset contains the details of 614 loan applicants, where each applicant is described with 12 features. Loan_Status is the target feature (ie., dependent variable) and all others are independent variables.

Step1. [Understand Data]. Using Pandas, import **"train_loan.csv"** file and print properties such as head, shape, columns, dtype, info and value_counts.

Step2. [Data Cleaning]

- Replace numbers as string by integer in "Dependents" column
- Fill missing data in categorical columns (Gender, Married, Dependents, Education, Self_Employed, Credit_History) by its mode value
- Handle missing values in numberical columns
- Drop Loan_ID column

Step3. [OPTIONAL: Exploratory Data Analysis - Who got their loan approved]

Draw count plot for

- Married?
- Dependants?
- Graduates?
- Self-employed?

Step4. [Extract X and y] from the dataframe

Step5. [One Hot Encoding]

Perform OHE on categorical columns, use this method: X = pd.get_dummies(X)

Step6. [Model Building]

- Split X and y for training and testing
- Using StandardScaler, fit_transform on X_train and transform on X_test values
- create LinearSVC model, train and test
- print accuracy value
- Print confusion matrix between y_test and y_pred
- Print classification_report

Categorical

```
train_data['Gender'].fillna(train_data['Gender'].mode()[0],
                            inplace=True)
train_data['Married'].fillna(train_data['Married'].mode()[0],
                             inplace=True)
train_data['Dependents'].fillna(train_data['Dependents'].
                                mode()[0], inplace=True)
train_data['Loan_Amount_Term'].fillna(train_data['Loan_Amount_Term'].
                                      mode()[0], inplace=True
train_data['Credit_History'].fillna(train_data['Credit_History'].
                                    mode()[0], inplace=True)
train_data['Self_Employed'].fillna(train_data['Self_Employed'].
                                   mode()[0], inplace=True)
```

numerical

```
train_data['Loan Amount'].fillna(train_data['Loan Amount'].
train_data = train_data.drop(['Loan_ID'], axis=1) mean(), inplace=True)
plt.subplot(231)
train_data['Married'].value_counts().plot(kind='bar',
                                          title='Gender',
plt.subplot(232)                          figsize=(20,10))
train_data['Dependents'].value_counts().plot(kind='bar',
                                             title='Dependents')
plt.subplot(233)
train_data['Education'].value_counts().plot(kind='bar',
                                            title='Education')
plt.subplot(234)
train_data['Self_Employed'].value_counts().
                                    plot(kind='bar',
plt.show()
```

Step7. [Performance Comparisons]

- Compare the performance of LinearSVC against LogisticRegression
- Compare the performance of LinearSVC against SGDClassifier
- Compare LinearSVC against SVC with various kernels such as 'linear', 'poly', 'rbf' and 'sigmoid'
- Interpret the results.

```
X = train_data.drop(['Loan_status'],axis=1)
y = train_data.Loan_status

import warnings
warnings.filterwarnings('ignore')
X = pd.get_dummies(X)
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)

from sklearn.preprocessing import StandardScaler
Scale = StandardScaler()
ss = Scale.fit_transform(X_train)
ss1 = Scale.transform(X_test)
from sklearn.svm import LinearSVC
model = LinearSVC()
model.fit(ss, y_train)
LSVC_y_pred = model.predict(ss1)
LSVC_y_pred
from sklearn.metrics import accuracy_score, confusion_matrix,
                            classification_report
accuracy_score(y_test, LSVC_y_pred)
```

final prog

```
rbf_svc = svm.SVC(kernel='rbf', C=1.0)
rbf_svc.fit(ss, y_train)
rbfsvc_y_pred = rbf_svc.predict(ss1)
sigmoid_svc = svm.SVC(kernel='sigmoid', C=1.0)
sigmoid_svc.fit(ss, y_train)
sigsvc_y_pred = sigmoid_svc.predict(ss1)
print("Linear SVC :", accuracy_score(y_test, lsvc_ypred))
print("poly SVC :", accuracy_score(y_test, psvc_y_pred))
print("rbf svc:", accuracy_score(y_test, rbfsvc_y_pred))
print("Sigmoid svc:", accuracy_score(y_test, sigsvc_y_pred))

Models = []
models.append(('svc', LinearSVC()))
modelsss.append(('svc_poly', svm.SVC(kernel='poly',
                                     C=1.0)))
modelsss.append(('svc_rbf', svm.SVC(kernel='rbf',
                                    C=1.0)))
modelsss.append(('svc_poly', svm.SVC(kernel='sigmoid',
                                     C=1.0)))

results = []
names = []
scoring = 'accuracy'
for name, model in modelsss:
    kfold = model_selection.KFold(n_split=10)
    cv_results = model_selection.cross_val_score(model,
    results.append(cv_results)
    names.append(cv_results)
    names.append(name)
    print(msg)
```

NOTES

```
Confusion_matrix(y_test, Lsvc_y_pred)

print('classification_report(y_test, Lsvc_y_pred))

from sklearn.linear_model import LogisticRegression

    lgr = LogisticRegression()
    lgr.fit(ss, y_train)
    lr_y_pred = lgr.predict(ss1)
  from .sklearn.svm import linear SVC.
      l_svc = linear SVC()
      l_svc.fit(ss, Y_train)
      lsvc_y_pred = l_svc.predict(ss1)
    print("Logistic Regression :", accuracy_score(y_test, lr_y_pred))
    print("Linear SVC :", accuracy_score(Y_test, lsvc_y_pred))
  from sklearn import svm, model_selection
    model = []
    model.append(('svc', LinearSVC()))
    model.append(('LR', LogisticRegression()))
      results = []
        names = []
        scoring = 'accuracy'
    for name, model in model:
        kfold = model_selection.kfold(n_splits=10)
```

```
cv_results = model_selection.cross_val_score(model, X_train, Y_train,
                                             scoring)
results.append(cv_results)
names.append(name)
msg = "%s: %f (%f)" % (name, cv_results.mean(),
                       cv_results.std())
print(msg)
fig = plt.figure()
fig.suptitle('Comparison between different MLAs')
ax = fig.add_subplot(111)
plt.boxplot(results)
plt.show()
from sklearn.linear_model import SGDClassifier
models = []
models.append(('SVC', LinearSVC()))
models.append(('SGD', SGDClassifier()))
results = []
names = []
scoring = 'accuracy'
for name, model in models:
    kfold = model_selection.KFold(n_splits=10)
    cv_results = model_selection.cross_val_score(model, X_train,
                                                 Y_train, cv=kfold, scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(),
                           cv_results.std())
    print(msg)
```

NOTES

```
fig = plt.figure()
fig.suptitle('Comparison between different MLAs')
ax = fig.add_subplot(111)
plt.boxplot(results)
plt.show()
from sklearn.linear_model import SGDClassifier
sgd = SGDclassifier()
sgd.fit(ss, y_train)
sgdc_y_pred = sgd.predict(ss1)
from sklearn.svm import linearSVC
l_svc = LinearSVC()
l_svc.fit(ss, y_train)
l_svc_y_pred = l_svc.predict(ss1)
print("SGDclassifiers", accuracy_score(y_test, sgdc_y_pred))
print("LinearSVC:", accuracy_score(y_test, lsvc_y_pred))
```

3)

```
from sklearn.svm import SVC
l_svc = LinearSVC()
l_svc.fit(ss, y_train)
l_svc_y_pred = l_svc.predict(ss1)
poly_svc = svm.SVC(kernel='poly', C=1.0)
poly_svc.fit(ss, y_train)
psvc_y_pred = poly_svc.predict(ss1)
```

REPORT

Lab7.Loan Approval Classification using SVM

In this lab we have learned to build a classification model to classify the loan applicants into eligible applicants or not eligible applicants by using Support Vector Machine.

Our first step is to check the properties of given data and clean if any missing values occur then for better understanding who got their loan approved, we just visualize using count plot for those who married, graduates, self-employed likewise.

Perform one hot encoding to categorical columns just simply by using pd.get_dummies () then build a LinearSVC model print accuracy value and classification_report then finally compare the performance of LinearSVC against LogisticRegression similarly for LinearSVC against SGDClassifier.

REPORT

Lab8.Animal Classification using Decision Trees

In this lab we have learned on how to build ID3 and CART Decision Tree to classify whether an animal is Mammal or Reptile.

On this case we have to create a dataset and save it as csv file then import using pandas pd.read_csv and check the properties of data.

Building ID3 model by creating Decision Tree using entropy criterion then perform training and testing also print accuracy and classification report then visualize your Decision Tree model using graphviz.

Then create a test data to check whether our predictions are correct? For CART Decision Tree will use criterion='gini' then for this model we train with full data without splitting it. Predict that for test file also visualize your CART Decision Tree model using graphviz.

This same thing we are performing on full animal dataset at https://archive.ics.uci.edu/ml/datasets/Zoo

Step 1

```
import pandas as pd
animal_data = pd.read_csv('animal.csv')
```

Step 2

```
animal_data
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
animal_data["Label"] = label_encoder.fit_transform(animal_data["species"])
animal_data
Categories = list(label_encoder.inverse_transform([0,1]))
Categories
X = animal_data.drop(['Label', 'species'], axis=)
Y = animal_data.Label
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
                                                    random_state=0)
clf = DecisionTreeClassifier(criterion='entropy', max_depth=4,
                             random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
y_pred
print("Accuracy of train:", clf.score(X_train, y_train))
print("Accuracy of test:", clf.score(X_test, y_test))
print('\n')
print(classification_report(y_test, y_pred))
```

Lab8. Animal Classification using Decision Trees

Objectives
In this lab, you will build ID3 and CART Decision Tree to classify whether an animal is a Mammal or Reptile.

Learning Outcomes
After completing this lab, you will be able to

- Create and import training dataset and test dataset
- Create ID3 Decision Tree using Entropy metric
- Create CART Decision Tree using Gini metric
- Visualize graph using graphviz

Step1. [Create Dataset]
Create the following dataset using Excel and save it as CSV file.

Toothed	hair	breathes	legs	species
TRUE	TRUE	TRUE	TRUE	Mammal
TRUE	TRUE	TRUE	TRUE	Mammal
TRUE	FALSE	TRUE	FALSE	Reptile
FALSE	TRUE	TRUE	TRUE	Mammal
TRUE	TRUE	TRUE	TRUE	Mammal
TRUE	TRUE	TRUE	TRUE	Mammal
TRUE	FALSE	FALSE	FALSE	Reptile
TRUE	FALSE	TRUE	FALSE	Reptile
TRUE	TRUE	TRUE	TRUE	Mammal
FALSE	FALSE	TRUE	TRUE	Reptile

Step2. [Model building using ID3]

- Import your data set
- Create DT model using 'entropy' criterion
- Perform training and testing
- Print accuracy and classification report.
- Interpret your results
- Visualize your DT model using graphviz

Install necessary package for visualization. The following sample code will help you to understand visualization.

```
with open("tree1.dot", 'w') as f:
     f = tree.export_graphviz(dtc_entropy1,
                              out_file=f,
                              max_depth = 4,
                              impurity = False,
                              feature_names = X.columns.values,
                              class_names = ['Reptile', 'Mammal'],
                              filled= True )
```

Note: Once dot file is created, then you can visualize online at http://www.webgraphviz.com/. Open and copy dot file contents and paste it here. You can view your graph.

Step3. [Create a Test Set]
Create a testing csv file with 3 samples as below. (columns 2, 3, 4, 5 only)

```
from sklearn import tree
from sklearn.tree import export_graphviz
with open("tree.dot", 'w') as f:
    f = tree.export_graphviz(clf, out_file=f, max_depth=4,
                             impurity=False, feature_names=)
```

Another Way to Visualize:

```
from sklearn import tree
%matplotlib inline
tree.plot_tree(clf)
```

Step:3

```
import csv
fields = ['Name', 'Teethed', 'Hair', 'Breathes', 'Legs', 'Species']
data rows of csv file
rows = [ ['Turtle', 'FALSE', 'FALSE', 'TRUE', 'FALSE', 'Reptile',
          'Blue Whales', 'FALSE', 'TRUE', 'TRUE', 'TRUE', 'Mammal'
          'Crocodile', 'TRUE', 'FALSE', 'TRUE', 'TRUE', 'Reptile'] ]
filename = "testing.csv"
with open(filename, 'w') as file:
    writer = csv.writer(file)
    writer = writerow(fields)
    writer.writerows(rows)
test_set = pd.read_csv('testing.csv')
test_set
test_set["label"] = label_encoder.fit_transform
                    (test_set["species"])
test_set.
```

Turtile	FALSE	FALSE	TRUE	FALSE	Reptile
Blue whales	FALSE	TRUE	TRUE	TRUE	Mammal
Crocodile	TRUE	FALSE	TRUE	TRUE	Reptile

Step4. [Perform prediction]
Use your ID3 DT model that you created before and predict labels for this test set.
Check your predictions. Correct?

Step5. [Build CART Decision Tree Model]

- Now, you are going to build a new CART decision tree using criterion='gini'.
- Train you model with full training data (No, train test split, this time)
- Predict samples for the test file
- Visualize your CART DT using graphviz

Step6. [Buid DT with Zoo dataset]

- Download full animal dataset at https://archive.ics.uci.edu/ml/datasets/Zoo
- Import, build model using ID3 and CART, train and test accuracy. Print classification report. Visualize your trees.

Step:4

```
stp4 = test_set.drop(['Name', 'species', 'Label'], axis=1)
y_pred = clf.predict(stp4)
y_pred
accuracy_score(test_set.Label, y_pred)
```

Step:5

```
clf_1 = DecisionTreeClassifier(criterion = 'gini', max_depth=4,
                               random_state=42)
clf_1.fit(x,y)

DecisionTreeClassifier(max_depth=4, random_state=42)

clf_1.predict(stp4)

with open("tree2.txt", 'w') as f:
    f = tree.export_graphviz(clf_1, out_file=f, max_depth=4,
                             impurity=False, feature_names
```

Another way to visualize:

```
from sklearn import tree
tree.plot_tree(clf_1)
```

step: 6

```
animal_2 = pd.read_csv("zoo.csv")
animal_2.head()
animal_2.shape
animal_2.info()
X1 = animal_2.drop(['name','type'], axis=1)
y1 = animal_2.type
X_train, X_test, y_train, y_test = train_test_split(X1, y1, test_size=
clf_2 = DecisionTreeClassifier(criterion='entropy', max_depth=3,
                               random_state=42)
clf_2.fit(x_train, y_train)
clf_2.predict(x_test)
clf_3 = DecisionTreeClassifier(criterion='gini', max_depth=4,
                               random_state=42)
clf_3.fit(X_train, y_train)
y_pred = clf_3.predict(X_test)
y_pred
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
SS = scale.fit_transform(X_train)
SS1 = scale.transform(X_test)
print("model accuracy:", accuracy_score(y_test, y_pred))
print("train accuracy", clf_3.score(SS, y_train))
print("test accuracy:", clf_3.score(SS1, y_test))
animal_2.type.value_counts(dropna=False)
```

NOTES

```
with open("trees.txt", 'w') as f:
f = tree.export_graphviz(clf, out_file=f, max_depth=16,
                         impurity=False.

print(classification_report(y_test, y_pred))

from sklearn import tree

tree.plot_tree(clf, 3)
```

Roll No: 205229133 ¶

Lab8. Animal Classification using Decision Trees

Objectives

In this lab, you will build ID3 and CART Decision Tree to classify whether an animal is a Mammal or Reptile.

Learning Outcomes

After completing this lab, you will be able to

- Create and import training dataset and test dataset
- Create ID3 Decision Tree using Entropy metric
- Create CART Decision Tree using Gini metric
- Visualize graph using graphviz

Import necessary Library

In [1]:

```
import csv
import pandas as pd
from sklearn import tree
from sklearn.tree import export_graphviz
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,classification_report
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report,auc,roc_c
import warnings
warnings.filterwarnings('ignore')
```

Step1. [Create Dataset]

Create the following dataset using Excel and save it as CSV file.

In [2]:

```
import pandas as pd
animal_data = pd.read_csv('animal.csv')
```

Step2. [Model building using ID3]

- Import your data set
- Create DT model using ‘entropy’ criterion
- Perform training and testing
- Print accuracy and classification report.
- Interpret your results

- Visualize your DT model using graphviz

In [3]:

```
animal_data
```

Out[3]:

	Toothed	Hair	Breathes	Legs	Species
0	True	True	True	True	Mammal
1	True	True	True	True	Mammal
2	True	False	True	False	Repite
3	False	True	True	True	Mammal
4	True	True	True	True	Mammal
5	True	True	True	True	Mammal
6	True	False	False	False	Repite
7	True	False	True	False	Repite
8	True	True	True	True	Mammal
9	False	False	True	True	Repite

In [4]:

```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
animal_data["Label"] = label_encoder.fit_transform(animal_data["Species"])
animal_data
```

Out[4]:

	Toothed	Hair	Breathes	Legs	Species	Label
0	True	True	True	True	Mammal	0
1	True	True	True	True	Mammal	0
2	True	False	True	False	Repite	1
3	False	True	True	True	Mammal	0
4	True	True	True	True	Mammal	0
5	True	True	True	True	Mammal	0
6	True	False	False	False	Repite	1
7	True	False	True	False	Repite	1
8	True	True	True	True	Mammal	0
9	False	False	True	True	Repite	1

In [5]:

```
categories = list(label_encoder.inverse_transform([0,1]))
categories
```

Out[5]:

```
['Mammal', 'Repite']
```

In [6]:

```
X = animal_data.drop(['Label','Species'],axis=1)
```

In [7]:

```
y = animal_data.Label
```

In [8]:

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score,classification_report

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.33,random_state=0)
clf = DecisionTreeClassifier(criterion='entropy',max_depth=4, random_state=42)
clf.fit(X_train,y_train)
```

Out[8]:

```
DecisionTreeClassifier(criterion='entropy', max_depth=4, random_state=42)
```

In [9]:

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

Out[9]:

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

In [10]:

```
print("Accuracy of train :",clf.score(X_train,y_train))
print("Accuracy of test  :",clf.score(X_test,y_test))
print('\n')
print(classification_report(y_test,y_pred))
```

```
Accuracy of train : 1.0
Accuracy of test  : 0.75


              precision    recall  f1-score   support

           0       0.67      1.00      0.80         2
           1       1.00      0.50      0.67         2

    accuracy                           0.75         4
   macro avg       0.83      0.75      0.73         4
weighted avg       0.83      0.75      0.73         4
```

In [11]:

```
from sklearn import tree
from sklearn.tree import export_graphviz

with open("tree1.dot", 'w') as f:
    f = tree.export_graphviz(clf,out_file=f,max_depth = 4,impurity = False,feature_names =
```

Now open tree1.txt file which will be created in your working directory then Copy and paste the code to http://webgraphviz.com/ (http://webgraphviz.com/)

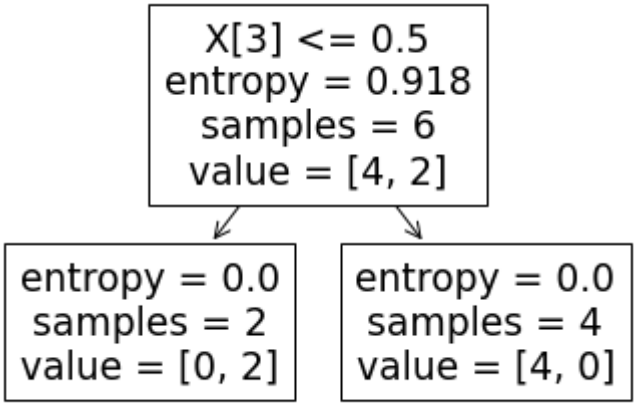
Another Way to visualize

In [12]:

```
from sklearn import tree
%matplotlib inline

tree.plot_tree(clf)
```

Out[12]:

```
[Text(167.4, 163.07999999999998, 'X[3] <= 0.5\nentropy = 0.918\nsamples = 6
\nvalue = [4, 2]'),
 Text(83.7, 54.360000000000014, 'entropy = 0.0\nsamples = 2\nvalue = [0,
2]'),
 Text(251.10000000000002, 54.360000000000014, 'entropy = 0.0\nsamples = 4\nv
alue = [4, 0]')]
```



Step3. [Create a Test Set]

In [13]:

```
import csv
fields = ['Name','Toothed', 'Hair', 'Breathes', 'Legs', 'Species']

# data rows of csv file
rows = [ ['Turtile','FALSE','FALSE','TRUE','FALSE','Reptile'],
         ['Blue Whales','FALSE','TRUE','TRUE','TRUE','Mammal'],
         ['Crocodile','TRUE','FALSE','TRUE','TRUE','Reptile'] ]

# name of csv file
filename = "testing.csv"


with open(filename, 'w') as file:
    # creating a csv dict writer object
    writer = csv.writer(file)

    # writing headers (field names)
    writer.writerow(fields)

    # writing data rows
    writer.writerows(rows)
```

In [14]:

```
test_set = pd.read_csv('testing.csv')
test_set
```

Out[14]:

	Name	Toothed	Hair	Breathes	Legs	Species
0	Turtile	False	False	True	False	Reptile
1	Blue Whales	False	True	True	True	Mammal
2	Crocodile	True	False	True	True	Reptile

In [15]:

```
test_set["Label"] = label_encoder.fit_transform(test_set["Species"])
test_set
```

Out[15]:

	Name	Toothed	Hair	Breathes	Legs	Species	Label
0	Turtile	False	False	True	False	Reptile	1
1	Blue Whales	False	True	True	True	Mammal	0
2	Crocodile	True	False	True	True	Reptile	1

Step4. [Perform prediction]

- Use your ID3 DT model that you created before and predict labels for this test set. Check your predictions. Correct?

In [16]:

```
stp4 = test_set.drop(['Name','Species','Label'],axis=1)
y_prd = clf.predict(stp4)
y_prd
```

Out[16]:

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

In [17]:

```
accuracy_score(test_set.Label,y_prd)
```

Out[17]:

```
0.6666666666666666
```

Step5. [Build CART Decision Tree Model]

- Now, you are going to build a new CART decision tree using criterion='gini'.
- Train you model with full training data (No, train test split, this time)
- Predict samples for the test file
- Visualize your CART DT using graphviz

In [18]:

```
clf_1 = DecisionTreeClassifier(criterion='gini',max_depth=4, random_state=42)
clf_1.fit(X,y)
```

Out[18]:

```
DecisionTreeClassifier(max_depth=4, random_state=42)
```

In [19]:

```
clf_1.predict(stp4)
```

Out[19]:

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

In [20]:

```
with open("tree2.txt", 'w') as f:
    f = tree.export_graphviz(clf_1,out_file=f,max_depth = 4,impurity = False,feature_names
```

Now open tree2.txt file which will be created in your working directory then Copy and paste the code to [http://webgraphviz.com/ (http://webgraphviz.com/)](http://webgraphviz.com/)

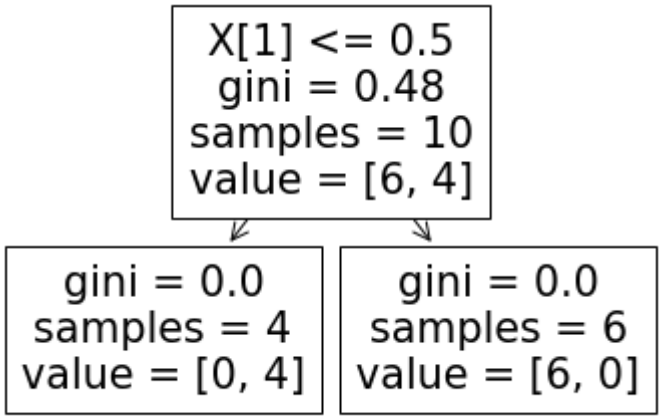
Another Way to visualize

In [21]:

```
from sklearn import tree
tree.plot_tree(clf_1)
```

Out[21]:

```
[Text(167.4, 163.07999999999998, 'X[1] <= 0.5\ngini = 0.48\nsamples = 10\nva
lue = [6, 4]'),
 Text(83.7, 54.360000000000014, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
 Text(251.10000000000002, 54.360000000000014, 'gini = 0.0\nsamples = 6\nvalu
e = [6, 0]')]
```



Step6. [Buid DT with Zoo dataset]

- Download full animal dataset at https://archive.ics.uci.edu/ml/datasets/Zoo (https://archive.ics.uci.edu/ml/datasets/Zoo)
- Import, build model using ID3 and CART, train and test accuracy. Print classification report. Visualize your trees.

In [22]:

```
animal_2 = pd.read_csv("zoo.csv")
```

In [23]:

```
animal_2.head()
```

Out[23]:

	name	hair	feathers	eggs	milk	airborne	aquatic	predator	toothed	backbone	breath
0	aardvark	1	0	0	1	0	0	1	1	1	
1	antelope	1	0	0	1	0	0	0	1	1	
2	bass	0	0	1	0	0	1	1	1	1	
3	bear	1	0	0	1	0	0	1	1	1	
4	boar	1	0	0	1	0	0	1	1	1	

In [24]:

```
animal_2.shape
```

Out[24]:

```
(101, 18)
```

In [25]:

```
animal_2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 101 entries, 0 to 100
Data columns (total 18 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   name      101 non-null    object
 1   hair      101 non-null    int64 
 2   feathers  101 non-null    int64 
 3   eggs      101 non-null    int64 
 4   milk      101 non-null    int64 
 5   airborne  101 non-null    int64 
 6   aquatic   101 non-null    int64 
 7   predator  101 non-null    int64 
 8   toothed   101 non-null    int64 
 9   backbone  101 non-null    int64 
 10  breathes  101 non-null    int64 
 11  venomous  101 non-null    int64 
 12  fins      101 non-null    int64 
 13  legs      101 non-null    int64 
 14  tail      101 non-null    int64 
 15  domestic  101 non-null    int64 
 16  catsize   101 non-null    int64 
 17  type      101 non-null    int64 
dtypes: int64(17), object(1)
memory usage: 14.3+ KB
```

In [26]:

```
X1 = animal_2.drop(['name','type'],axis=1)
y1 = animal_2.type
```

In [27]:

```
X__train,X__test,y__train,y__test = train_test_split(X1,y1,test_size=0.33,random_state=0)
```

In [28]:

```
clf_2 = DecisionTreeClassifier(criterion='entropy',max_depth=3, random_state=52)
clf_2.fit(X__train,y__train)
```

Out[28]:

```
DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=52)
```

In [29]:

```
clf_2.predict(X__test)
```

Out[29]:

```
array([4, 4, 4, 1, 1, 1, 2, 4, 1, 1, 7, 1, 2, 7, 4, 6, 1, 7, 2, 4, 2, 4,
       1, 2, 1, 1, 1, 2, 4, 4, 4, 4, 4, 1], dtype=int64)
```

In [30]:

```
clf_3 = DecisionTreeClassifier(criterion='gini',max_depth=4, random_state=42)
clf_3.fit(X__train,y__train)
```

Out[30]:

```
DecisionTreeClassifier(max_depth=4, random_state=42)
```

In [31]:

```
y_pred=clf_3.predict(X__test)
y_pred
```

Out[31]:

```
array([7, 4, 4, 1, 1, 1, 2, 4, 1, 1, 7, 1, 2, 7, 4, 6, 1, 7, 2, 4, 2, 7,
       1, 2, 1, 1, 1, 2, 4, 7, 4, 7, 7, 1], dtype=int64)
```

In [32]:

```
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
ss = scale.fit_transform(X__train)
ss1 = scale.transform(X__test)

print("model accuracy :",accuracy_score(y__test,y_pred))
print("Train accuracy :",clf_3.score(ss,y__train))
print("Test accuracy  :",clf_3.score(ss1,y__test))
```

```
model accuracy : 0.8235294117647058
Train accuracy : 0.9253731343283582
Test accuracy  : 0.8235294117647058
```

In [33]:

```
animal_2.type.value_counts(dropna=False)
```

Out[33]:

```
1    41
2    20
4    13
7    10
6     8
3     5
5     4
Name: type, dtype: int64
```

In [34]:

```
with open("tree3.txt", 'w') as f:
    f = tree.export_graphviz(clf_3,out_file=f,max_depth = 16,impurity = False,feature_names
```

In [35]:

```
print(classification_report(y__test,y_pred))
```

```
              precision    recall  f1-score   support

           1       1.00      1.00      1.00        12
           2       1.00      1.00      1.00         6
           3       0.00      0.00      0.00         4
           4       1.00      1.00      1.00         7
           5       0.00      0.00      0.00         1
           6       1.00      0.50      0.67         2
           7       0.25      1.00      0.40         2

    accuracy                           0.82        34
   macro avg       0.61      0.64      0.58        34
weighted avg       0.81      0.82      0.80        34
```

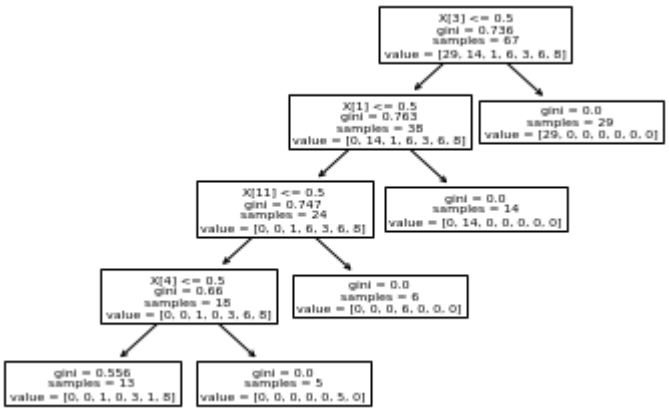
Now open tree3.txt file which will be created in your working directory then Copy and paste the code to [http://webgraphviz.com/ (http://webgraphviz.com/)](http://webgraphviz.com/)

In [36]:

```
from sklearn import tree
tree.plot_tree(clf_3)
```

Out[36]:

```
[Text(239.14285714285714, 195.696, 'X[3] <= 0.5\ngini = 0.736\nsamples = 67
\nvalue = [29, 14, 1, 6, 3, 6, 8]'),
 Text(191.31428571428572, 152.208, 'X[1] <= 0.5\ngini = 0.763\nsamples = 38
\nvalue = [0, 14, 1, 6, 3, 6, 8]'),
 Text(143.4857142857143, 108.72, 'X[11] <= 0.5\ngini = 0.747\nsamples = 24\n
value = [0, 0, 1, 6, 3, 6, 8]'),
 Text(95.65714285714286, 65.232, 'X[4] <= 0.5\ngini = 0.66\nsamples = 18\nva
lue = [0, 0, 1, 0, 3, 6, 8]'),
 Text(47.82857142857143, 21.744, 'gini = 0.556\nsamples = 13\nvalue = [0, 0,
1, 0, 3, 1, 8]'),
 Text(143.4857142857143, 21.744, 'gini = 0.0\nsamples = 5\nvalue = [0, 0, 0,
0, 0, 5, 0]'),
 Text(191.31428571428572, 65.232, 'gini = 0.0\nsamples = 6\nvalue = [0, 0,
0, 6, 0, 0, 0]'),
 Text(239.14285714285714, 108.72, 'gini = 0.0\nsamples = 14\nvalue = [0, 14,
0, 0, 0, 0, 0]'),
 Text(286.9714285714286, 152.208, 'gini = 0.0\nsamples = 29\nvalue = [29, 0,
0, 0, 0, 0, 0]')]
```



lab 9

In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import tree
from sklearn.metrics import precision_score, recall_score,accuracy_score,roc_auc_
from sklearn.tree import export_graphviz
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')
```

Step1. [Understand Data].

In [2]:
```
df = pd.read_csv("Employee_Hopping.csv")
```

In [3]:
```
df.head()
```

Out[3]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	Educati
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life S
1	49	No	Travel_Frequently	279	Research & Development	8	1	Life S
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	
3	33	No	Travel_Frequently	1392	Research & Development	3	4	Life S
4	27	No	Travel_Rarely	591	Research & Development	2	1	

5 rows × 35 columns

In [4]:
```
df.shape
```

Out[4]:
```
(1470, 35)
```

In [5]:
```
df.columns
```

Out[5]:
```
Index(['Age', 'Attrition', 'BusinessTravel', 'DailyRate', 'Department',
       'DistanceFromHome', 'Education', 'EducationField', 'EmployeeCount',
       'EmployeeNumber', 'EnvironmentSatisfaction', 'Gender', 'HourlyRate',
       'JobInvolvement', 'JobLevel', 'JobRole', 'JobSatisfaction',
       'MaritalStatus', 'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked',
       'Over18', 'OverTime', 'PercentSalaryHike', 'PerformanceRating',
       'RelationshipSatisfaction', 'StandardHours', 'StockOptionLevel',
       'TotalWorkingYears', 'TrainingTimesLastYear', 'WorkLifeBalance',
       'YearsAtCompany', 'YearsInCurrentRole', 'YearsSinceLastPromotion',
       'YearsWithCurrManager'],
      dtype='object')
```

In [6]:
```
df.dtypes
```

Out[6]:
```
Age                         int64
Attrition                  object
BusinessTravel             object
DailyRate                   int64
Department                 object
DistanceFromHome            int64
Education                   int64
EducationField             object
EmployeeCount               int64
EmployeeNumber              int64
EnvironmentSatisfaction     int64
Gender                     object
HourlyRate                  int64
JobInvolvement              int64
JobLevel                    int64
JobRole                    object
JobSatisfaction             int64
MaritalStatus              object
MonthlyIncome               int64
MonthlyRate                 int64
NumCompaniesWorked          int64
Over18                     object
OverTime                   object
PercentSalaryHike           int64
PerformanceRating           int64
RelationshipSatisfaction    int64
StandardHours               int64
StockOptionLevel            int64
TotalWorkingYears           int64
TrainingTimesLastYear       int64
WorkLifeBalance             int64
YearsAtCompany              int64
YearsInCurrentRole          int64
YearsSinceLastPromotion     int64
YearsWithCurrManager        int64
dtype: object
```

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 35 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   Age                       1470 non-null   int64 
 1   Attrition                 1470 non-null   object
 2   BusinessTravel            1470 non-null   object
 3   DailyRate                 1470 non-null   int64 
 4   Department                1470 non-null   object
 5   DistanceFromHome          1470 non-null   int64 
 6   Education                 1470 non-null   int64 
 7   EducationField            1470 non-null   object
 8   EmployeeCount             1470 non-null   int64 
 9   EmployeeNumber            1470 non-null   int64 
 10  EnvironmentSatisfaction   1470 non-null   int64 
 11  Gender                    1470 non-null   object
 12  HourlyRate                1470 non-null   int64 
 13  JobInvolvement            1470 non-null   int64 
 14  JobLevel                  1470 non-null   int64 
 15  JobRole                   1470 non-null   object
 16  JobSatisfaction           1470 non-null   int64 
 17  MaritalStatus             1470 non-null   object
 18  MonthlyIncome             1470 non-null   int64 
 19  MonthlyRate               1470 non-null   int64 
 20  NumCompaniesWorked        1470 non-null   int64 
 21  Over18                    1470 non-null   object
 22  OverTime                  1470 non-null   object
 23  PercentSalaryHike         1470 non-null   int64 
 24  PerformanceRating         1470 non-null   int64 
 25  RelationshipSatisfaction  1470 non-null   int64 
 26  StandardHours             1470 non-null   int64 
 27  StockOptionLevel          1470 non-null   int64 
 28  TotalWorkingYears         1470 non-null   int64 
 29  TrainingTimesLastYear     1470 non-null   int64 
 30  WorkLifeBalance           1470 non-null   int64 
 31  YearsAtCompany            1470 non-null   int64 
 32  YearsInCurrentRole        1470 non-null   int64 
 33  YearsSinceLastPromotion   1470 non-null   int64 
 34  YearsWithCurrManager      1470 non-null   int64 
dtypes: int64(26), object(9)
memory usage: 402.1+ KB
```

In [8]:
```
df.value_counts
```

Out[8]:
```
<bound method DataFrame.value_counts of       Age Attrition     BusinessTrave
l  DailyRate              Department  \
0      41       Yes      Travel_Rarely       1102                   Sales
1      49        No  Travel_Frequently        279  Research & Development
2      37       Yes      Travel_Rarely       1373  Research & Development
3      33        No  Travel_Frequently       1392  Research & Development
4      27        No      Travel_Rarely        591  Research & Development
...   ...       ...                 ...        ...                     ...
1465   36        No  Travel_Frequently        884  Research & Development
1466   39        No      Travel_Rarely        613  Research & Development
1467   27        No      Travel_Rarely        155  Research & Development
1468   49        No  Travel_Frequently       1023                   Sales
1469   34        No      Travel_Rarely        628  Research & Development

      DistanceFromHome  Education EducationField  EmployeeCount  \
0                    1          2  Life Sciences              1
1                    8          1  Life Sciences              1
2                    2          2          Other              1
3                    3          4  Life Sciences              1
```

In [9]:
```
df.isnull().sum()
```

Out[9]:
```
Age                         0
Attrition                   0
BusinessTravel              0
DailyRate                   0
Department                  0
DistanceFromHome            0
Education                   0
EducationField              0
EmployeeCount               0
EmployeeNumber              0
EnvironmentSatisfaction     0
Gender                      0
HourlyRate                  0
JobInvolvement              0
JobLevel                    0
JobRole                     0
JobSatisfaction             0
MaritalStatus               0
MonthlyIncome               0
MonthlyRate                 0
NumCompaniesWorked          0
Over18                      0
OverTime                    0
PercentSalaryHike           0
PerformanceRating           0
RelationshipSatisfaction    0
StandardHours               0
StockOptionLevel            0
TotalWorkingYears           0
TrainingTimesLastYear       0
WorkLifeBalance             0
YearsAtCompany              0
YearsInCurrentRole          0
YearsSinceLastPromotion     0
YearsWithCurrManager        0
dtype: int64
```

Step2. [Extract X and y].

In [10]:
```
X = df.drop(['Attrition'],axis=1)
y = df.Attrition
```

In [11]:
```
y = y.apply(lambda x:1 if x == 'Yes' else 0)
```

In [12]:
```
df.select_dtypes(include=['object']).dtypes
```

Out[12]:
```
Attrition          object
BusinessTravel     object
Department         object
EducationField     object
Gender             object
JobRole            object
MaritalStatus      object
Over18             object
OverTime           object
dtype: object
```

Step3. [Feature Engineering]

In [13]:
```
df=pd.get_dummies(df,columns=["BusinessTravel","Department",'EducationField',"Ger
df.head()
```

Out[13]:

	Age	Attrition	DailyRate	DistanceFromHome	Education	EmployeeCount	EmployeeNumber	Env
0	41	Yes	1102	1	2	1	1	
1	49	No	279	8	1	1	2	
2	37	Yes	1373	2	2	1	4	
3	33	No	1392	3	4	1	5	
4	27	No	591	2	1	1	7	

5 rows × 56 columns

Step4. Now, check shape of X and y.

In [14]:
```
X = df.drop(['Attrition'],axis=1)
X.shape
```

Out[14]:
```
(1470, 55)
```

In [15]:
```
y.shape
```

Out[15]:
```
(1470,)
```

Step5. [Model Development]

In [16]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=0
```

In [17]:
```python
seed = 0
rfc = RandomForestClassifier(n_estimators=1000, max_features=0.3, max_depth=4, mi
```

In [18]:
```python
rfc.fit(X_train,y_train)
```

Out[18]:
```
RandomForestClassifier(max_depth=4, max_features=0.3, min_samples_leaf=2,
                       n_estimators=1000, n_jobs=-1, random_state=0,
                       warm_start=True)
```

In [19]:
```python
y_pred=rfc.predict(X_test)
y_pred
```

Out[19]:
```
array([0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 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, 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, 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,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 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, 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, 1, 0, 0, 0, 0, 0, 0, 0, 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, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 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, 0, 0, 0, 0,
       0], dtype=int64)
```

Step6. [Testing]

In [20]:
```python
accuracy_score(y_test,y_pred)
```

Out[20]:
```
0.8639455782312925
```

In [21]:
```python
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.86      0.99      0.92       371
           1       0.86      0.17      0.29        70

    accuracy                           0.86       441
   macro avg       0.86      0.58      0.61       441
weighted avg       0.86      0.86      0.82       441
```

Step7. [Feature importance value]

In [22]:
```
print(rfc.feature_importances_)
```

```
[6.98321450e-02 3.65227658e-02 2.00159132e-02 5.21457627e-03
 0.00000000e+00 1.93136701e-02 3.09477993e-02 2.51635390e-02
 1.66715720e-02 5.04884918e-02 1.37024559e-02 1.15212766e-01
 1.89715997e-02 1.83688986e-02 1.16183893e-02 6.52783762e-04
 9.05995065e-03 0.00000000e+00 2.83057741e-02 6.73115246e-02
 5.96985891e-03 1.88628396e-02 5.02462199e-02 1.64563675e-02
 8.78154018e-03 4.70126629e-02 3.99898213e-03 1.62801031e-02
 2.47672036e-03 5.60666617e-04 3.92298011e-03 4.67197125e-03
 2.85590037e-03 1.92092323e-03 3.48874178e-03 2.75324633e-03
 3.67258786e-04 4.33772973e-03 1.65455825e-03 1.66961888e-03
 3.66392947e-04 7.05642476e-04 4.35437163e-03 3.37746500e-04
 1.17707045e-03 6.70465871e-05 4.98737913e-03 6.52221544e-03
 1.25986442e-02 2.39045147e-03 3.31770313e-03 2.24531725e-02
 0.00000000e+00 9.21168370e-02 9.29418217e-02]
```

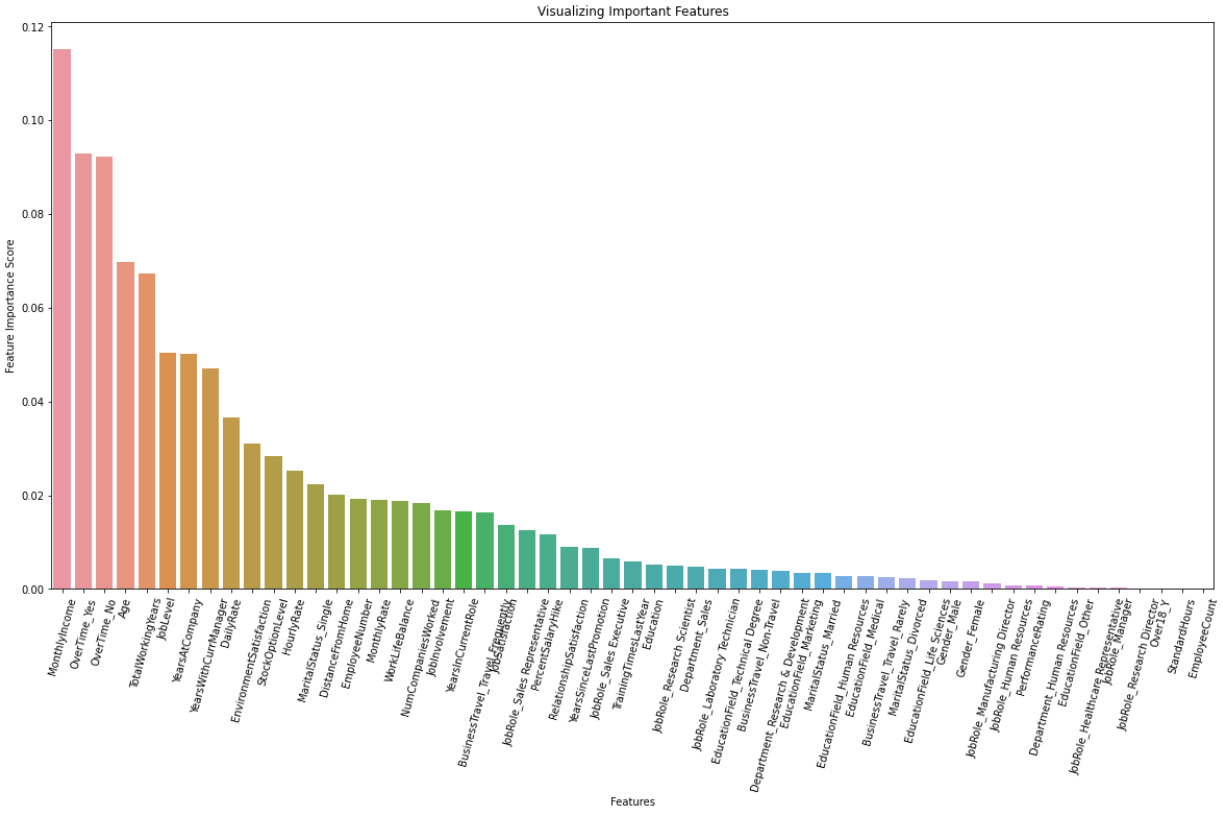
In [23]:
```
feature_imp = pd.DataFrame(rfc.feature_importances_,index=X_train.columns,columns
feature_imp
```

Out[23]:

	Important score
MonthlyIncome	0.115213
OverTime_Yes	0.092942
OverTime_No	0.092117
Age	0.069832
TotalWorkingYears	0.067312
JobLevel	0.050488
YearsAtCompany	0.050246
YearsWithCurrManager	0.047013
DailyRate	0.036523
EnvironmentSatisfaction	0.030948
StockOptionLevel	0.028306
HourlyRate	0.025164
MaritalStatus_Single	0.022453
DistanceFromHome	0.020016
EmployeeNumber	0.019314
MonthlyRate	0.018972
WorkLifeBalance	0.018863
NumCompaniesWorked	0.018369
JobInvolvement	0.016672
YearsInCurrentRole	0.016456
BusinessTravel_Travel_Frequently	0.016280
JobSatisfaction	0.013702
JobRole_Sales Representative	0.012599
PercentSalaryHike	0.011618
RelationshipSatisfaction	0.009060
YearsSinceLastPromotion	0.008782
JobRole_Sales Executive	0.006522
TrainingTimesLastYear	0.005970
Education	0.005215
JobRole_Research Scientist	0.004987
Department_Sales	0.004672
JobRole_Laboratory Technician	0.004354
EducationField_Technical Degree	0.004338

	Important score
BusinessTravel_Non-Travel	0.003999
Department_Research & Development	0.003923
EducationField_Marketing	0.003489
MaritalStatus_Married	0.003318
EducationField_Human Resources	0.002856
EducationField_Medical	0.002753
BusinessTravel_Travel_Rarely	0.002477
MaritalStatus_Divorced	0.002390
EducationField_Life Sciences	0.001921
Gender_Male	0.001670
Gender_Female	0.001655
JobRole_Manufacturing Director	0.001177
JobRole_Human Resources	0.000706
PerformanceRating	0.000653
Department_Human Resources	0.000561
EducationField_Other	0.000367
JobRole_Healthcare Representative	0.000366
JobRole_Manager	0.000338
JobRole_Research Director	0.000067
Over18_Y	0.000000
StandardHours	0.000000
EmployeeCount	0.000000

In [24]:
```
plt.figure(figsize=(20,10))
sns.barplot(x=feature_imp.index, y=feature_imp['Important score'])
# Add labels to your graph

plt.ylabel('Feature Importance Score')
plt.xlabel('Features')
plt.title("Visualizing Important Features")
plt.xticks(rotation=75)
plt.show()
```



Step8. [Visualize your RF Decision Tree using graphviz]

http://www.webgraphviz.com/ (http://www.webgraphviz.com/).

In [25]:
```
estimator = rfc.estimators_[5]
```

In [36]:
```
export_graphviz(estimator, out_file='rtree1.txt',feature_names = X_train.columns.
```

Step9. [RF with a range of trees]

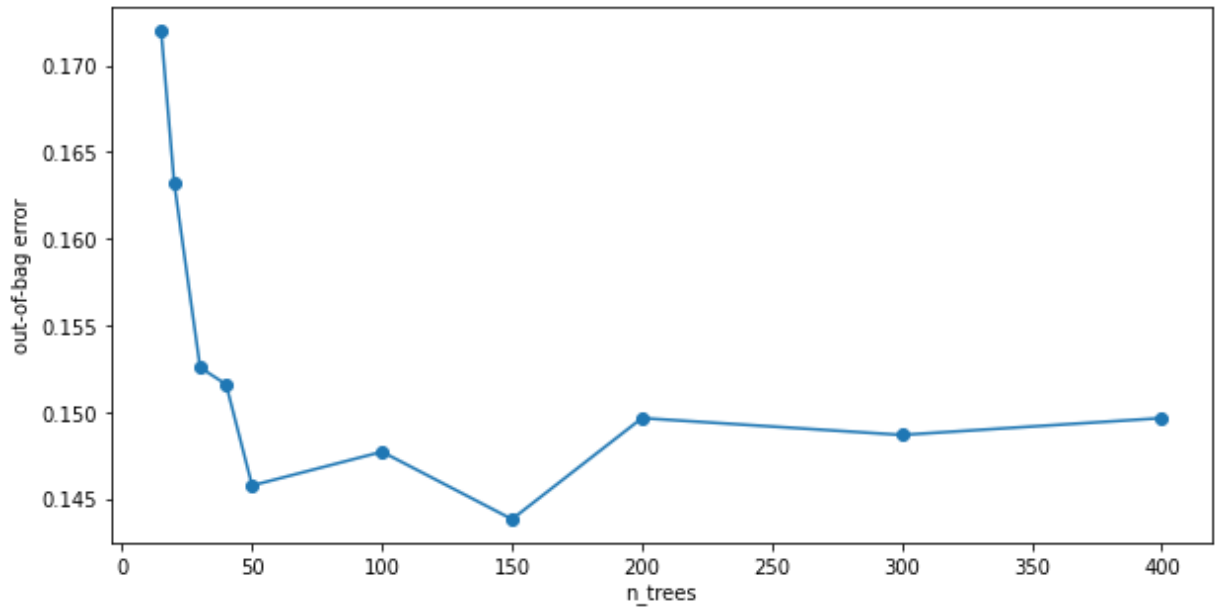
In [27]:
```
rf2 = RandomForestClassifier(oob_score=True,random_state=42,warm_start=True,n_job
oob_list = list()
# Iterate through all of the possibilities for number of trees

for n_trees in [15, 20, 30, 40, 50, 100, 150, 200, 300, 400]:
    rf2.set_params(n_estimators=n_trees)
    rf2.fit(X_train, y_train)

    # Get the oob error

    oob_error = 1 - rf2.oob_score_
    oob_list.append(pd.Series({'n_trees': n_trees, 'oob': oob_error}))

rf_oob_df = pd.concat(oob_list, axis=1).T.set_index('n_trees')
rf_oob_df
```

Out[27]:

n_trees	oob
15.0	0.172012
20.0	0.163265
30.0	0.152575
40.0	0.151603
50.0	0.145773
100.0	0.147716
150.0	0.143829
200.0	0.149660
300.0	0.148688
400.0	0.149660

In [28]:
```
ax = rf_oob_df.plot(legend=False, marker='o', figsize=(10,5))
ax.set(ylabel='out-of-bag error')
```

Out[28]:
```
[Text(0, 0.5, 'out-of-bag error')]
```



Step11. [Compare with DecisionTreeClassifier]

In [29]:
```
clf_1 = DecisionTreeClassifier(criterion='gini',max_depth=4, random_state=42)
```

In [30]:
```
clf_1.fit(X_train,y_train)
```

Out[30]:
```
DecisionTreeClassifier(max_depth=4, random_state=42)
```

In [31]:
```
y_pred1=clf_1.predict(X_test)
y_pred1
```

Out[31]:
```
array([0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 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, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 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, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 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, 1, 0, 0, 1, 1, 0, 0,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 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, 1, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0], dtype=int64)
```

In [32]:
```
with open("rtree2.txt", 'w') as f:
    f = tree.export_graphviz(clf_1,out_file=f,max_depth = 4,impurity = False,feat
```

http://www.webgraphviz.com/ (http://www.webgraphviz.com/).

In [33]:
```
accuracy_score(y_test,y_pred1)
```

Out[33]:
```
0.8480725623582767
```

In [34]:
```
print(classification_report(y_test,y_pred1))
```

```
              precision    recall  f1-score   support

           0       0.89      0.94      0.91       371
           1       0.53      0.39      0.45        70

    accuracy                           0.85       441
   macro avg       0.71      0.66      0.68       441
weighted avg       0.83      0.85      0.84       441
```

In [35]:
```
print("RF model:      ",accuracy_score(y_test,y_pred))
print("RF Precision:  ",precision_score(y_test,y_pred))
print("RF Recall:     ",recall_score(y_test,y_pred))
print("RF F1 score:   ",f1_score(y_test,y_pred))
print("\n")
print("DT model:      ",accuracy_score(y_test,y_pred1))
print("DT Precision:  ",precision_score(y_test,y_pred1))
print("DT Recall:     ",recall_score(y_test,y_pred1))
print("DT F1 score:   ",f1_score(y_test,y_pred1))
```

```
RF model:       0.8639455782312925
RF Precision:   0.8571428571428571
RF Recall:      0.17142857142857143
RF F1 score:    0.2857142857142857


DT model:       0.8480725623582767
DT Precision:   0.5294117647058824
DT Recall:      0.38571428571428573
DT F1 score:    0.4462809917355372
```

```
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt
from sklearn import tree
from sklearn.metrics import precision_score, recall_score,
    recall_score, accuracy_score, roc_auc_score, classification_
from sklearn.tree.export_graphviz report, reportf, f_score
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')
```

Step 1

```
df = pd.read_csv("Employee_Hopping.csv")
df.head()
df.shape
df.columns
df.dtypes
df.value_counts()
```

Step 2

```
x = df.drop(['Attrition'], axis=1)
y = df.Attrition
y = y.apply(lambda x: 1 if x == 'Yes' else 0)
df.select_dtypes(include=['object']).dtypes
```

Lab9. Employee Hopping Prediction using Random Forests

Objectives

In this lab, you will build a classification model to predict whether employee will continue to work or quit his job in the company using Random Forest classifier.

Learning Outcomes

After completing this lab, you will be able to

- Perform One Hot Encoding on categorical columns
- Create RandomForestClassifier, perform training and testing
- Print accuracy score and classification report
- Print feature importance values
- Select the best number of trees based on out-of-bag error values
- Compare against Decision Tree model
- Visualize trees using graphviz

Business Use Case

You are a data scientist. Heber Software Solutions is a leading IT industry in your city. They have collected the details of employees and if they work or left the company. They ask you to build a prediction model so that they can use your model to check if employees will continue to work or leave. Based on this analysis, they will understand what make them to quit and accordingly they will design employee welfare management schemes.

Step1. [Understand Data].

- Using Pandas, import **"Employee_Hopping.csv "** file and print properties such as head, shape, columns, dtype, info and value_counts.

Step2. [Extract X and y]

- Create X and y columns from the dataframe

Step3. [Feature Engineering]

- There are 8 categorical columns (where dtype="object"). Perform one hot encoding and create new columns

Step4. Now, check shape of X and y.

Step5. [Model Development]

- Split X and y for training and testing
- Create RandomForestClassifier model, fit (no need to scale) and predict

Step6. [Testing]

- Print accuracy score between y_test and y_pred
- Print classification report between y_test and y_pred and observe the results

Step7. [Feature importance value]

- You can look at feature importance values using the property, rf.feature_importances_
- Print feature name and its rf.feature_importances_ values and understand important features
- Show a Bar plot between feature column names and feature_importances_ score.

Step8. [Visualize your RF Decision Tree using graphviz]

- Figure below show a segment of the RF tree which is visualized at http://www.webgraphviz.com/. You should copy and paste .dot file in this page.

Step: 3 [Feature Engineering]

```
df = pd.get_dummies(df, columns=["BusinessTravel", "Department",
                                 "EducationField", "Gender", "JobRole",
                                 "MaritalStatus", "over18", "OverTime"])
df.head()
```

Step: 4:

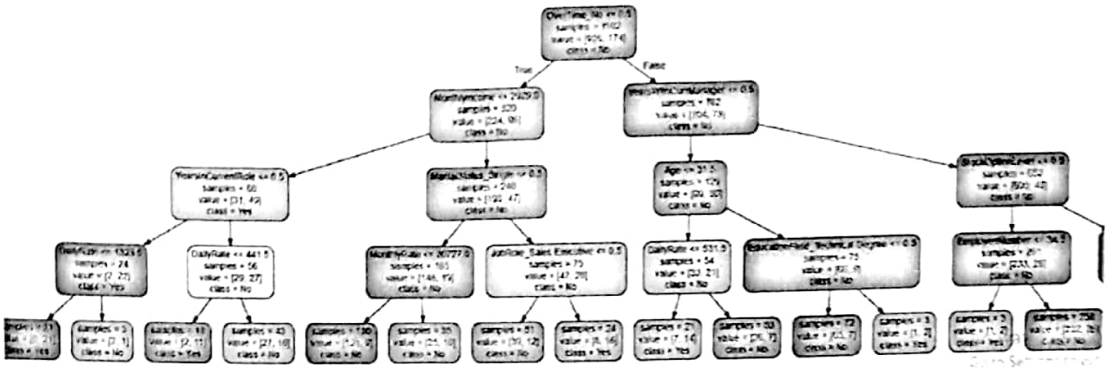
```
x = df.drop("Attrition", axis=1)
x.shape
y.shape
```

Step: 5

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,
                                                    random_state=0)
seed=0
rfc = RandomForestClassifier(n_estimators=1000, max_features=0.3,
                             max_depth=4, min_samples_leaf=2,
                             n_jobs=-1, random_state=seed, warm_start=True,
                             verbose=0)
rfc.fit(x_train, y_train)
y_pred = rfc.predict(x_test)
y_pred
```

Step: 6

```
accuracy_score(y_test, y_pred)
print(classification_report(y_test, y_pred))
```



Step9. [RF with a range of trees]

Fit random forest models with a range of tree numbers [15, 20, 30, 40, 50, 100, 150, 200, 300, 400] and print Out-Of-Bag error for each of these model. Use model.oob_score_ to get score and subtract this score from 1 to get the oob-error. That is, oob-error = 1 - model.oob_score_.

Hint: since the only thing changing is the number of trees, the `warm_start` flag can be used so that the model just adds more trees to the existing model each time. Use the `set_params` method to update the number of trees. The following code many help to understand this part.

```
rf2 = RandomForestClassifier(oob_score=True,
                             random_state=42,
                             warm_start=True,
                             n_jobs=-1)

oob_list = list()

# Iterate through all of the possibilities for number of trees
for n_trees in [15, 20, 30, 40, 50, 100, 150, 200, 300, 400]:

    # Use this to set the number of trees
    rf2.set_params(n_estimators=n_trees)

    # Fit the model
    rf2.fit(X_train, y_train)

    # Get the oob error
    oob_error = 1 - rf2.oob_score_

    # Store it
    oob_list.append(pd.Series({'n_trees': n_trees, 'oob': oob_error}))

rf_oob_df = pd.concat(oob_list, axis=1).T.set_index('n_trees')

rf_oob_df
```

Step10. [Plot oob-error for each tree]

The following lines will help you

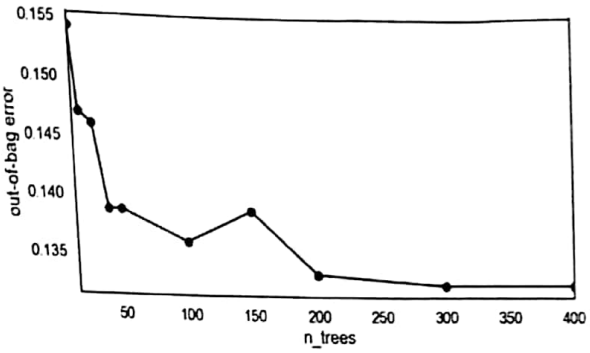
```
ax = rf_oob_df.plot(legend=False, marker='o', figsize=(10,5))
ax.set(ylabel='out-of-bag error')
```

Step 7:

```
print(rfc.feature_importances_)
feature_imp = pd.DataFrame(rfc.feature_importances_, index=X_train.
                           columns, columns=['Important score'])
feature_imp
plt.figure(figsize=(20,10))
sns.barplot(x=feature_imp.index, y=feature_imp['Important score'])
plt.ylabel("Feature Importance Score")
plt.xlabel('Features')
plt.title("Visualizing Important features")
plt.yticks(rotation=45)
plt.show()
```

Step 8:

```
estimator = rfc.estimators_[5]
export_graphviz(estimator, out_file='tree1.txt',
                feature_names = X_train.columns.values,
                class_names=['Yes', 'No'], rounded = true,
                proportion = false, precision=2, filled = true
```



Step11. [Compare with DecisionTreeClassifier]

- Create DecisionTreeClassifier, fit and predict on test set
- Visualize the tree using graphviz
- Print accuracy score
- Print classification report
- What is the result of the comparision between RF and DT models? Which gives best accuracy?.
- What is your comment on precision, recall, f1 score values?

Step:9

```
rf2 = RandomForestClassifier(oob_score=True, random_state=42,
                             warm_start=True, n_jobs=-1)
oob_list = list()
for n_trees in [15,20,30,40,50,100,150,200,300,400]:
    rf2.set_params(n_estimators=n_trees)
    rf2.fit(x_train, y_train)
    oob_error = 1 - rf2.oob_score_
    oob_list.append(pd.Series({'n_trees': n_trees, 'oob': oob_error}))
rf_oob_df = pd.concat(oob_list, axis=1).T.set_index('n_trees')
rf_oob_df
ax = rf_oob_df.plot(legend=False, marker='o', figsize=(10,5))
ax.set(ylabel='out-of-bag error')
```

Step : 11

```
clf_1 = DecisionTreeClassifier(criterion='gini', max_depth=4,
                               random_state=42)
clf_1.fit(X_train, y_train)

y_pred1 = clf_1.predict(x_test)
y_pred1

with open("tree2.txt", 'w') as f:
    f = tree.export_graphviz(clf_1, out_file=f, max_depth=4,
                             impurity=False, feature_names=
                             X_train.columns.values,
                             class_names=['yes', 'no'], filled=True)

accuracy = score(y_test, y_pred1)
print(classification_report(y_test, y_pred1))

print("RF Model): ", accuracy_score(y_test, y_pred))
print("RF Precision: ", precision_score(y_test, y_pred))
print("RF Recall: ", recall_score(y_test, y_pred))
print("RF F1 score: ", f1_score(y_test, y_pred))
print("\n")
print("DT model: ", accuracy_score(y_test, y_pred1))
print("DT Precision: ", precision_score(y_test, y_pred1))
print("DT Recall: ", recall_score(y_test, y_pred1))
print("DT F1 score: ", f1_score(y_test, y_pred1))
```

REPORT

Lab9.Employee Hopping Prediction using Random Forests

In this lab we have learned how to build a classification model to predict whether employee will continue to work or quit his job in the company using Random Forest Classifier.

Import data then perform pre-processing and data cleaning then perform one hot encoding on categorical columns then create RandomForestClassifier and perform training, testing, accuracy score and classification report.

Understanding feature_importances by using the property, rf.feature_importances_ then compare random forest model with decision tree model also visualize using graphviz

Lab10. Patients Physical Activities Prediction using Boosting

In [1]:
```
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import accuracy_score,classification_report
from sklearn.ensemble import GradientBoostingClassifier,AdaBoostClassifier,Randor
from sklearn.linear_model import LogisticRegressionCV
from sklearn.model_selection import train_test_split,GridSearchCV,cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
```

Step 1 [Understand Data]

In [2]:
```
original = pd.read_csv("Human_Activity_Data.csv")
```

In [3]:
```
original.head()
```

Out[3]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y	tE
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.983185	-
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.974914	-
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.963668	-
3	0.279174	-0.026201	-0.123283	-0.996091	-0.983403	-0.990675	-0.997099	-0.982750	-
4	0.276629	-0.016570	-0.115362	-0.998139	-0.980817	-0.990482	-0.998321	-0.979672	-

5 rows × 562 columns

In [4]:
```
original.columns
```

Out[4]:
```
Index(['tBodyAcc-mean()-X', 'tBodyAcc-mean()-Y', 'tBodyAcc-mean()-Z',
       'tBodyAcc-std()-X', 'tBodyAcc-std()-Y', 'tBodyAcc-std()-Z',
       'tBodyAcc-mad()-X', 'tBodyAcc-mad()-Y', 'tBodyAcc-mad()-Z',
       'tBodyAcc-max()-X',
       ...
       'fBodyBodyGyroJerkMag-skewness()', 'fBodyBodyGyroJerkMag-kurtosis()',
       'angle(tBodyAccMean,gravity)', 'angle(tBodyAccJerkMean),gravityMean)',
       'angle(tBodyGyroMean,gravityMean)',
       'angle(tBodyGyroJerkMean,gravityMean)', 'angle(X,gravityMean)',
       'angle(Y,gravityMean)', 'angle(Z,gravityMean)', 'Activity'],
      dtype='object', length=562)
```

In [5]:
```
original.shape
```

Out[5]:
```
(10299, 562)
```

In [6]:
```
original.dtypes
```

Out[6]:
```
tBodyAcc-mean()-X                       float64
tBodyAcc-mean()-Y                       float64
tBodyAcc-mean()-Z                       float64
tBodyAcc-std()-X                        float64
tBodyAcc-std()-Y                        float64
                                          ...
angle(tBodyGyroJerkMean,gravityMean)    float64
angle(X,gravityMean)                    float64
angle(Y,gravityMean)                    float64
angle(Z,gravityMean)                    float64
Activity                                 object
Length: 562, dtype: object
```

In [7]:
```
original.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10299 entries, 0 to 10298
Columns: 562 entries, tBodyAcc-mean()-X to Activity
dtypes: float64(561), object(1)
memory usage: 44.2+ MB
```

In [8]:
```
original.value_counts()
```

Out[8]:
```
tBodyAcc-mean()-X  tBodyAcc-mean()-Y  tBodyAcc-mean()-Z  tBodyAcc-std()-X  tB
odyAcc-std()-Y  tBodyAcc-std()-Z  tBodyAcc-mad()-X  tBodyAcc-mad()-Y  tBodyAc
c-mad()-Z  tBodyAcc-max()-X  tBodyAcc-max()-Y  tBodyAcc-max()-Z  tBodyAcc-min
()-X  tBodyAcc-min()-Y  tBodyAcc-min()-Z  tBodyAcc-sma()  tBodyAcc-energy()-X
tBodyAcc-energy()-Y  tBodyAcc-energy()-Z  tBodyAcc-iqr()-X  tBodyAcc-iqr()-Y
tBodyAcc-iqr()-Z  tBodyAcc-entropy()-X  tBodyAcc-entropy()-Y  tBodyAcc-entrop
y()-Z  tBodyAcc-arCoeff()-X,1  tBodyAcc-arCoeff()-X,2  tBodyAcc-arCoeff()-X,3
tBodyAcc-arCoeff()-X,4  tBodyAcc-arCoeff()-Y,1  tBodyAcc-arCoeff()-Y,2  tBody
Acc-arCoeff()-Y,3  tBodyAcc-arCoeff()-Y,4  tBodyAcc-arCoeff()-Z,1  tBodyAcc-a
rCoeff()-Z,2  tBodyAcc-arCoeff()-Z,3  tBodyAcc-arCoeff()-Z,4  tBodyAcc-correl
ation()-X,Y  tBodyAcc-correlation()-X,Z  tBodyAcc-correlation()-Y,Z  tGravity
Acc-mean()-X  tGravityAcc-mean()-Y  tGravityAcc-mean()-Z  tGravityAcc-std()-X
tGravityAcc-std()-Y  tGravityAcc-std()-Z  tGravityAcc-mad()-X  tGravityAcc-ma
d()-Y  tGravityAcc-mad()-Z  tGravityAcc-max()-X  tGravityAcc-max()-Y  tGravit
yAcc-max()-Z  tGravityAcc-min()-X  tGravityAcc-min()-Y  tGravityAcc-min()-Z
tGravityAcc-sma()  tGravityAcc-energy()-X  tGravityAcc-energy()-Y  tGravityAc
c-energy()-Z  tGravityAcc-iqr()-X  tGravityAcc-iqr()-Y  tGravityAcc-iqr()-Z
tGravityAcc-entropy()-X  tGravityAcc-entropy()-Y  tGravityAcc-entropy()-Z  tG
ravityAcc-arCoeff()-X,1  tGravityAcc-arCoeff()-X,2  tGravityAcc-arCoeff()-X,3
```

In [9]:
```
label_encoder = LabelEncoder()
original["label_Activity"] = label_encoder.fit_transform(original["Activity"])
```

Step2. [Build a small dataset]

In [10]:
```
original.Activity.value_counts()
```

Out[10]:
```
LAYING                1944
STANDING              1906
SITTING               1777
WALKING               1722
WALKING_UPSTAIRS      1544
WALKING_DOWNSTAIRS    1406
Name: Activity, dtype: int64
```

In [11]:
```
original.label_Activity.value_counts()
```

Out[11]:
```
0    1944
2    1906
1    1777
3    1722
5    1544
4    1406
Name: label_Activity, dtype: int64
```

Take first 3000 samples for each 6 activities and build classifier

In [12]:
```
tem1 = original[original['Activity']=='LAYING'][:500]
tem2 = original[original['Activity']=='SITTING'][:500]
tem3 = original[original['Activity']=='WALKING'][:500]
tem4 = original[original['Activity']=='STANDING'][:500]
tem5 = original[original['Activity']=='WALKING_UPSTAIRS'][:500]
tem6 = original[original['Activity']=='WALKING_DOWNSTAIRS'][:500]
```

In [13]:
```
new_df = pd.concat([tem1,tem2,tem3,tem4,tem5,tem6])
```

In [14]:
```
new_df.to_csv("human_activity_clipped3000.csv")
```

In [15]:
```
new_df = pd.read_csv("human_activity_clipped3000.csv")
```

In [16]:
```
new_df.head()
```

Out[16]:

	Unnamed: 0	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tB
0	51	0.403474	-0.015074	-0.118167	-0.914811	-0.895231	-0.891748	-0.917696	-(
1	52	0.278373	-0.020561	-0.096825	-0.984883	-0.991118	-0.982112	-0.987985	-(
2	53	0.276555	-0.017869	-0.107621	-0.994195	-0.996372	-0.995615	-0.994901	-(
3	54	0.279575	-0.017276	-0.109481	-0.996135	-0.995812	-0.998689	-0.996393	-(
4	55	0.276527	-0.016819	-0.107983	-0.996775	-0.997256	-0.995422	-0.997167	-(

5 rows × 564 columns

In [17]:
```
new_df.shape
```

Out[17]:
```
(3000, 564)
```

In [18]:
```
new_df.columns
```

Out[18]:
```
Index(['Unnamed: 0', 'tBodyAcc-mean()-X', 'tBodyAcc-mean()-Y',
       'tBodyAcc-mean()-Z', 'tBodyAcc-std()-X', 'tBodyAcc-std()-Y',
       'tBodyAcc-std()-Z', 'tBodyAcc-mad()-X', 'tBodyAcc-mad()-Y',
       'tBodyAcc-mad()-Z',
       ...
       'fBodyBodyGyroJerkMag-kurtosis()', 'angle(tBodyAccMean,gravity)',
       'angle(tBodyAccJerkMean),gravityMean)',
       'angle(tBodyGyroMean,gravityMean)',
       'angle(tBodyGyroJerkMean,gravityMean)', 'angle(X,gravityMean)',
       'angle(Y,gravityMean)', 'angle(Z,gravityMean)', 'Activity',
       'label_Activity'],
      dtype='object', length=564)
```

In [19]:
```
new_df.dtypes
```

Out[19]:
```
Unnamed: 0                int64
tBodyAcc-mean()-X       float64
tBodyAcc-mean()-Y       float64
tBodyAcc-mean()-Z       float64
tBodyAcc-std()-X        float64
                          ...
angle(X,gravityMean)    float64
angle(Y,gravityMean)    float64
angle(Z,gravityMean)    float64
Activity                 object
label_Activity            int64
Length: 564, dtype: object
```

In [20]:
```
new_df.value_counts()
```

Out[20]:
```
Unnamed: 0  tBodyAcc-mean()-X  tBodyAcc-mean()-Y  tBodyAcc-mean()-Z  tBodyAcc
-std()-X  tBodyAcc-std()-Y  tBodyAcc-std()-Z  tBodyAcc-mad()-X  tBodyAcc-mad
()-Y  tBodyAcc-mad()-Z  tBodyAcc-max()-X  tBodyAcc-max()-Y  tBodyAcc-max()-Z
tBodyAcc-min()-X  tBodyAcc-min()-Y  tBodyAcc-min()-Z  tBodyAcc-sma()  tBodyAc
c-energy()-X  tBodyAcc-energy()-Y  tBodyAcc-energy()-Z  tBodyAcc-iqr()-X  tBo
dyAcc-iqr()-Y  tBodyAcc-iqr()-Z  tBodyAcc-entropy()-X  tBodyAcc-entropy()-Y
tBodyAcc-entropy()-Z  tBodyAcc-arCoeff()-X,1  tBodyAcc-arCoeff()-X,2  tBodyAc
c-arCoeff()-X,3  tBodyAcc-arCoeff()-X,4  tBodyAcc-arCoeff()-Y,1  tBodyAcc-arC
oeff()-Y,2  tBodyAcc-arCoeff()-Y,3  tBodyAcc-arCoeff()-Y,4  tBodyAcc-arCoeff
()-Z,1  tBodyAcc-arCoeff()-Z,2  tBodyAcc-arCoeff()-Z,3  tBodyAcc-arCoeff()-Z,
4  tBodyAcc-correlation()-X,Y  tBodyAcc-correlation()-X,Z  tBodyAcc-correlati
on()-Y,Z  tGravityAcc-mean()-X  tGravityAcc-mean()-Y  tGravityAcc-mean()-Z  t
GravityAcc-std()-X  tGravityAcc-std()-Y  tGravityAcc-std()-Z  tGravityAcc-mad
()-X  tGravityAcc-mad()-Y  tGravityAcc-mad()-Z  tGravityAcc-max()-X  tGravity
Acc-max()-Y  tGravityAcc-max()-Z  tGravityAcc-min()-X  tGravityAcc-min()-Y  t
GravityAcc-min()-Z  tGravityAcc-sma()  tGravityAcc-energy()-X  tGravityAcc-en
ergy()-Y  tGravityAcc-energy()-Z  tGravityAcc-iqr()-X  tGravityAcc-iqr()-Y  t
GravityAcc-iqr()-Z  tGravityAcc-entropy()-X  tGravityAcc-entropy()-Y  tGravit
yAcc-entropy()-Z  tGravityAcc-arCoeff()-X,1  tGravityAcc-arCoeff()-X,2  tGrav
```

In [21]:
```
X_=new_df.drop(['Activity','label_Activity'],axis=1)
y_=new_df.label_Activity
X__train,X__test,y__train,y__test = train_test_split(X_,y_,test_size=0.2,random_s
```

Build GradientBoostingClassifier for 3000 samples

In [22]:
```
model_ = GradientBoostingClassifier(subsample=0.5,n_estimators=100,learning_rate=
model_.fit(X__train,y__train)
y__pred=model_.predict(X__test)
```

In [23]:
```
print(accuracy_score(y__test,y__pred))
print(classification_report(y__test,y__pred))
```

```
0.705
              precision    recall  f1-score   support

           0       0.36      0.11      0.16        94
           1       0.57      0.70      0.63        97
           2       0.54      0.74      0.63       101
           3       0.82      0.92      0.87       105
           4       0.87      0.87      0.87       103
           5       0.88      0.83      0.86       100

    accuracy                           0.70       600
   macro avg       0.68      0.70      0.67       600
weighted avg       0.68      0.70      0.68       600
```

Build AdaBoostClassifier for 3000 samples

In [24]:
```
base = DecisionTreeClassifier(max_features=4)
base2 = AdaBoostClassifier(base_estimator=base,random_state=0)
param_grid = {'n_estimators': [100, 150, 200], 'learning_rate': [0.01, 0.001]}
```

In [25]:
```
model2_ = GridSearchCV(base2,param_grid,cv=10,n_jobs=-1)
model2_.fit(X__train,y__train)
y__pred2=model2_.predict(X__test)
```

In [26]:
```
print(accuracy_score(y__test,y__pred2))
print(classification_report(y__test,y__pred2))
```

```
0.7716666666666666
              precision    recall  f1-score   support

           0       0.76      0.69      0.72        94
           1       0.62      0.65      0.63        97
           2       0.69      0.71      0.70       101
           3       0.82      0.92      0.87       105
           4       0.90      0.81      0.85       103
           5       0.86      0.83      0.85       100

    accuracy                           0.77       600
   macro avg       0.77      0.77      0.77       600
weighted avg       0.78      0.77      0.77       600
```

In [27]:
```
model2_.best_estimator_
```

Out[27]:
```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_features=4),
                   learning_rate=0.01, n_estimators=100, random_state=0)
```

Build LogisticRegressionCV classifier for 3000 samples

In [28]:
```
model3_ = LogisticRegressionCV(cv=4,Cs=5,penalty='l2')
model3_.fit(X__train,y__train)
y__pred3=model3_.predict(X__test)
```

In [29]:
```
print(accuracy_score(y__test,y__pred3))
print(classification_report(y__test,y__pred3))
```

```
0.9766666666666667
              precision    recall  f1-score   support

           0       0.99      1.00      0.99        94
           1       0.96      0.92      0.94        97
           2       0.93      0.96      0.95       101
           3       1.00      0.99      1.00       105
           4       0.98      1.00      0.99       103
           5       1.00      0.99      0.99       100

    accuracy                           0.98       600
   macro avg       0.98      0.98      0.98       600
weighted avg       0.98      0.98      0.98       600
```

Find Best no. of trees and Best Learning Rate using Grid Search and Cross Validation for 3000 samples

In [30]:
```
Param_grid={'n_estimators':[50,100,200, 400],'learning_rate':[0.1,0.01]}
```

In [31]:
```
all_scores = cross_val_score(estimator=model_,X=X__train,y=y__train,cv=5)
print(all_scores)
```

```
[0.10416667 0.55416667 0.72708333 0.80416667 0.95      ]
```

In [32]:
```
model4_ = GridSearchCV(estimator=model_,param_grid=Param_grid,cv=5,n_jobs=-1)
model4_.fit(X__train,y__train)
y__pred4=model4_.predict(X__test)
```

In [33]:
```
print(accuracy_score(y__test,y__pred4))
print(classification_report(y__test,y__pred4))
```

```
0.985
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        94
           1       0.97      0.95      0.96        97
           2       0.95      0.97      0.96       101
           3       1.00      0.99      1.00       105
           4       0.99      1.00      1.00       103
           5       1.00      1.00      1.00       100

    accuracy                           0.98       600
   macro avg       0.99      0.98      0.98       600
weighted avg       0.99      0.98      0.98       600
```

In [34]:
```
model4_.best_estimator_
```

Out[34]:
```
GradientBoostingClassifier(max_depth=1, max_features=4, n_estimators=400,
                           random_state=0, subsample=0.5)
```

Build VotingClassifier for 3000 samples

In [35]:
```
model5_=VotingClassifier(estimators=[('lr',model3_),('gbc',base2)],voting='soft')
model5_.fit(X__train,y__train)
y__pred5=model5_.predict(X__test)
```

In [36]:
```
print(accuracy_score(y__test,y__pred5))
print(classification_report(y__test,y__pred5))
```

```
0.7716666666666666
              precision    recall  f1-score   support

           0       0.76      0.69      0.72        94
           1       0.62      0.65      0.63        97
           2       0.69      0.71      0.70       101
           3       0.82      0.92      0.87       105
           4       0.90      0.81      0.85       103
           5       0.86      0.83      0.85       100

    accuracy                           0.77       600
   macro avg       0.77      0.77      0.77       600
weighted avg       0.78      0.77      0.77       600
```

From this 3000 smaples you should take 1500 samples for each activites

In [37]:
```
temp1 = new_df[new_df['Activity']=='LAYING'][:500]
temp2 = new_df[new_df['Activity']=='SITTING'][:500]
temp3 = new_df[new_df['Activity']=='WALKING'][:500]
```

In [38]:
```
df = pd.concat([temp1,temp2,temp3])
```

In [39]:
```
df.to_csv("human_activity_clipped1500.csv")
```

In [40]:
```
df = pd.read_csv("human_activity_clipped1500.csv")
```

In [41]:
```
df.head()
```

Out[41]:

	Unnamed: 0	Unnamed: 0.1	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tB
0	0	51	0.403474	-0.015074	-0.118167	-0.914811	-0.895231	-0.891748	-(
1	1	52	0.278373	-0.020561	-0.096825	-0.984883	-0.991118	-0.982112	-(
2	2	53	0.276555	-0.017869	-0.107621	-0.994195	-0.996372	-0.995615	-(
3	3	54	0.279575	-0.017276	-0.109481	-0.996135	-0.995812	-0.998689	-(
4	4	55	0.276527	-0.016819	-0.107983	-0.996775	-0.997256	-0.995422	-(

5 rows × 565 columns

In [42]:
```
df.shape
```

Out[42]:
```
(1500, 565)
```

In [43]:
```
df.columns
```

Out[43]:
```
Index(['Unnamed: 0', 'Unnamed: 0.1', 'tBodyAcc-mean()-X', 'tBodyAcc-mean()-Y',
       'tBodyAcc-mean()-Z', 'tBodyAcc-std()-X', 'tBodyAcc-std()-Y',
       'tBodyAcc-std()-Z', 'tBodyAcc-mad()-X', 'tBodyAcc-mad()-Y',
       ...
       'fBodyBodyGyroJerkMag-kurtosis()', 'angle(tBodyAccMean,gravity)',
       'angle(tBodyAccJerkMean),gravityMean)',
       'angle(tBodyGyroMean,gravityMean)',
       'angle(tBodyGyroJerkMean,gravityMean)', 'angle(X,gravityMean)',
       'angle(Y,gravityMean)', 'angle(Z,gravityMean)', 'Activity',
       'label_Activity'],
      dtype='object', length=565)
```

In [44]:
```
df.dtypes
```

Out[44]:
```
Unnamed: 0                int64
Unnamed: 0.1              int64
tBodyAcc-mean()-X       float64
tBodyAcc-mean()-Y       float64
tBodyAcc-mean()-Z       float64
                         ...
angle(X,gravityMean)    float64
angle(Y,gravityMean)    float64
angle(Z,gravityMean)    float64
Activity                 object
label_Activity            int64
Length: 565, dtype: object
```

In [45]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1500 entries, 0 to 1499
Columns: 565 entries, Unnamed: 0 to label_Activity
dtypes: float64(561), int64(3), object(1)
memory usage: 6.5+ MB
```

In [46]:
```
df.value_counts()
```

Out[46]:
```
Unnamed: 0  Unnamed: 0.1  tBodyAcc-mean()-X  tBodyAcc-mean()-Y  tBodyAcc-mean
()-Z  tBodyAcc-std()-X  tBodyAcc-std()-Y  tBodyAcc-std()-Z  tBodyAcc-mad()-X
tBodyAcc-mad()-Y  tBodyAcc-mad()-Z  tBodyAcc-max()-X  tBodyAcc-max()-Y  tBody
Acc-max()-Z  tBodyAcc-min()-X  tBodyAcc-min()-Y  tBodyAcc-min()-Z  tBodyAcc-s
ma()  tBodyAcc-energy()-X  tBodyAcc-energy()-Y  tBodyAcc-energy()-Z  tBodyAcc
-iqr()-X  tBodyAcc-iqr()-Y  tBodyAcc-iqr()-Z  tBodyAcc-entropy()-X  tBodyAcc-
entropy()-Y  tBodyAcc-entropy()-Z  tBodyAcc-arCoeff()-X,1  tBodyAcc-arCoeff()
-X,2  tBodyAcc-arCoeff()-X,3  tBodyAcc-arCoeff()-X,4  tBodyAcc-arCoeff()-Y,1
tBodyAcc-arCoeff()-Y,2  tBodyAcc-arCoeff()-Y,3  tBodyAcc-arCoeff()-Y,4  tBody
Acc-arCoeff()-Z,1  tBodyAcc-arCoeff()-Z,2  tBodyAcc-arCoeff()-Z,3  tBodyAcc-a
rCoeff()-Z,4  tBodyAcc-correlation()-X,Y  tBodyAcc-correlation()-X,Z  tBodyAc
c-correlation()-Y,Z  tGravityAcc-mean()-X  tGravityAcc-mean()-Y  tGravityAcc-
mean()-Z  tGravityAcc-std()-X  tGravityAcc-std()-Y  tGravityAcc-std()-Z  tGra
vityAcc-mad()-X  tGravityAcc-mad()-Y  tGravityAcc-mad()-Z  tGravityAcc-max()-
X  tGravityAcc-max()-Y  tGravityAcc-max()-Z  tGravityAcc-min()-X  tGravityAcc
-min()-Y  tGravityAcc-min()-Z  tGravityAcc-sma()  tGravityAcc-energy()-X  tGr
avityAcc-energy()-Y  tGravityAcc-energy()-Z  tGravityAcc-iqr()-X  tGravityAcc
-iqr()-Y  tGravityAcc-iqr()-Z  tGravityAcc-entropy()-X  tGravityAcc-entropy()
-Y  tGravityAcc-entropy()-Z  tGravityAcc-arCoeff()-X,1  tGravityAcc-arCoeff()
```

Step3. [Build GradientBoostingClassifier]

In [47]:
```
X=df.drop(['Activity','label_Activity'],axis=1)
y=df.label_Activity
```

In [48]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=2
```

In [49]:
```
model = GradientBoostingClassifier(subsample=0.5,n_estimators=100,learning_rate=1
```

In [50]:
```
model.fit(X_train,y_train)
```

Out[50]:
```
GradientBoostingClassifier(learning_rate=1.0, max_depth=1, max_features=4,
                           random_state=42, subsample=0.5)
```

In [51]:
```
y_pred=model.predict(X_test)
y_pred
```

Out[51]:
```
array([0, 1, 1, 1, 3, 1, 0, 0, 3, 3, 1, 3, 3, 3, 3, 1, 3, 3, 3, 0, 3, 1,
       0, 1, 0, 0, 0, 1, 0, 1, 3, 0, 3, 1, 1, 3, 0, 0, 3, 1, 0, 1, 0, 0,
       3, 0, 3, 3, 3, 1, 0, 0, 0, 1, 1, 0, 3, 3, 0, 3, 1, 1, 0, 1, 0, 3,
       3, 0, 3, 0, 0, 3, 0, 0, 1, 3, 3, 0, 3, 0, 3, 3, 1, 3, 1, 1, 1, 3,
       0, 3, 1, 3, 1, 1, 1, 3, 1, 0, 3, 1, 3, 0, 0, 3, 1, 3, 1, 0, 1, 0,
       1, 0, 0, 0, 1, 1, 1, 1, 3, 0, 0, 0, 0, 0, 1, 3, 3, 1, 1, 1, 1, 3,
       0, 1, 0, 3, 3, 3, 3, 3, 3, 1, 0, 3, 1, 1, 3, 3, 3, 0, 1, 1, 0, 0,
       0, 3, 0, 1, 3, 3, 1, 1, 1, 1, 0, 1, 0, 0, 0, 3, 0, 1, 1, 3, 0, 0,
       0, 0, 1, 3, 1, 0, 1, 1, 3, 1, 1, 0, 3, 0, 1, 3, 0, 3, 0, 1, 3, 0,
       1, 1, 3, 1, 0, 0, 0, 3, 3, 1, 1, 3, 3, 3, 3, 1, 0, 1, 0, 0, 3, 0,
       3, 3, 0, 0, 3, 1, 3, 0, 3, 1, 1, 3, 0, 1, 1, 1, 0, 0, 3, 1, 0, 3,
       3, 0, 3, 1, 0, 1, 1, 3, 1, 3, 3, 0, 0, 1, 3, 0, 0, 3, 0, 1, 0, 1,
       3, 1, 1, 0, 0, 3, 0, 1, 0, 0, 0, 1, 3, 0, 0, 1, 3, 3, 1, 3, 1, 3,
       3, 3, 1, 3, 3, 1, 1, 1, 0, 0, 1, 1, 0, 3], dtype=int64)
```

In [52]:
```
accuracy_score(y_test,y_pred)
```

Out[52]:
```
1.0
```

In [53]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       100
           1       1.00      1.00      1.00       100
           3       1.00      1.00      1.00       100

    accuracy                           1.00       300
   macro avg       1.00      1.00      1.00       300
weighted avg       1.00      1.00      1.00       300
```

Step4. [Find Best no. of trees and Best Learning Rate using Grid Search and Cross Validation]

In [54]:
```
Param_grid={'n_estimators':[50, 100, 200, 400],'learning_rate':[0.1, 0.01]}
```

In [55]:
```
all_scores = cross_val_score(estimator=model,X=X_train,y=y_train,cv=5)
print(all_scores)
```

```
[0.99166667 1.         1.         1.         1.        ]
```

In [56]:
```
model2 = GridSearchCV(estimator=model,param_grid=Param_grid,cv=5,n_jobs=-1)
```

In [57]:
```
model2.fit(X_train,y_train)
```

Out[57]:
```
GridSearchCV(cv=5,
             estimator=GradientBoostingClassifier(learning_rate=1.0,
                                                  max_depth=1, max_features=4,
                                                  random_state=42,
                                                  subsample=0.5),
             n_jobs=-1,
             param_grid={'learning_rate': [0.1, 0.01],
                         'n_estimators': [50, 100, 200, 400]})
```

In [58]:
```
y_pred2=model2.predict(X_test)
y_pred2
```

Out[58]:
```
array([0, 1, 1, 1, 3, 1, 0, 0, 3, 3, 1, 3, 3, 3, 3, 1, 3, 3, 3, 0, 3, 1,
       0, 1, 0, 0, 0, 1, 0, 1, 3, 0, 3, 1, 1, 3, 0, 0, 3, 1, 0, 1, 0, 0,
       3, 0, 3, 3, 3, 1, 0, 0, 0, 1, 1, 0, 3, 3, 0, 3, 1, 1, 0, 1, 0, 3,
       3, 0, 3, 0, 0, 3, 0, 0, 1, 3, 3, 0, 3, 0, 3, 3, 1, 3, 1, 1, 1, 3,
       0, 3, 1, 3, 1, 1, 1, 3, 1, 0, 3, 1, 3, 0, 0, 3, 1, 3, 1, 0, 1, 0,
       1, 0, 0, 0, 1, 1, 1, 1, 3, 0, 0, 0, 0, 0, 1, 3, 3, 1, 1, 1, 1, 3,
       0, 1, 0, 3, 3, 3, 3, 3, 3, 1, 0, 3, 1, 1, 3, 3, 3, 0, 1, 1, 0, 0,
       0, 3, 0, 1, 3, 3, 1, 1, 1, 1, 0, 1, 0, 0, 0, 3, 0, 1, 1, 3, 0, 0,
       0, 0, 1, 3, 1, 0, 1, 1, 3, 1, 1, 0, 3, 0, 1, 3, 0, 3, 0, 1, 3, 0,
       1, 1, 3, 1, 0, 0, 0, 3, 3, 1, 1, 3, 3, 3, 3, 1, 0, 1, 0, 0, 3, 0,
       3, 3, 0, 0, 3, 1, 3, 0, 3, 1, 1, 3, 0, 1, 1, 1, 0, 0, 3, 1, 0, 3,
       3, 0, 3, 1, 0, 1, 1, 3, 1, 3, 3, 0, 0, 1, 3, 0, 0, 3, 0, 1, 0, 1,
       3, 1, 1, 0, 0, 3, 0, 1, 0, 0, 0, 1, 3, 0, 0, 1, 3, 3, 1, 3, 1, 3,
       3, 3, 1, 3, 3, 1, 1, 1, 0, 0, 1, 1, 0, 3], dtype=int64)
```

In [59]:
```
accuracy_score(y_test,y_pred2)
```

Out[59]:
```
1.0
```

In [60]:
```
print(classification_report(y_test,y_pred2))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       100
           1       1.00      1.00      1.00       100
           3       1.00      1.00      1.00       100

    accuracy                           1.00       300
   macro avg       1.00      1.00      1.00       300
weighted avg       1.00      1.00      1.00       300
```

In [61]:
```
model2.best_estimator_
```

Out[61]:
```
GradientBoostingClassifier(max_depth=1, max_features=4, n_estimators=200,
                           random_state=42, subsample=0.5)
```

Step5. [Build AdaBoostClassifier]

In [62]:
```
base = DecisionTreeClassifier(max_features=4)
base2 = AdaBoostClassifier(base_estimator=base,random_state=0)
param_grid = {'n_estimators': [100, 150, 200], 'learning_rate': [0.01, 0.001]}
```

In [63]:
```
model3 = GridSearchCV(base2,param_grid,cv=10,n_jobs=-1)
```

In [64]:
```
model3.fit(X_train,y_train)
```

Out[64]:
```
GridSearchCV(cv=10,
             estimator=AdaBoostClassifier(base_estimator=DecisionTreeClassifier
(max_features=4),
                                          random_state=0),
             n_jobs=-1,
             param_grid={'learning_rate': [0.01, 0.001],
                         'n_estimators': [100, 150, 200]})
```

In [65]:
```
y_pred3=model3.predict(X_test)
y_pred3
```

Out[65]:
```
array([0, 1, 1, 1, 3, 0, 0, 0, 3, 3, 1, 3, 3, 3, 3, 1, 3, 3, 3, 0, 3, 1,
       0, 1, 0, 0, 0, 0, 0, 1, 3, 0, 3, 1, 1, 3, 0, 0, 3, 1, 1, 1, 0, 0,
       3, 0, 3, 3, 3, 1, 1, 0, 0, 1, 1, 0, 3, 3, 1, 3, 1, 1, 0, 1, 0, 3,
       3, 0, 3, 0, 0, 3, 0, 0, 1, 3, 3, 0, 3, 0, 3, 3, 1, 3, 1, 1, 1, 3,
       0, 3, 1, 3, 1, 1, 1, 3, 1, 1, 3, 1, 3, 0, 0, 3, 0, 3, 1, 0, 1, 0,
       1, 1, 0, 0, 0, 1, 1, 1, 3, 0, 0, 0, 0, 0, 1, 3, 3, 1, 1, 1, 1, 3,
       1, 1, 0, 3, 3, 3, 3, 3, 3, 0, 0, 3, 1, 1, 3, 3, 3, 0, 1, 1, 0, 0,
       0, 3, 0, 1, 3, 3, 1, 1, 0, 1, 0, 1, 1, 0, 0, 3, 0, 0, 1, 3, 0, 0,
       1, 1, 1, 3, 0, 0, 1, 1, 3, 1, 1, 0, 3, 1, 1, 3, 0, 3, 1, 1, 3, 0,
       1, 1, 3, 1, 1, 0, 0, 3, 3, 1, 1, 3, 3, 3, 3, 1, 0, 0, 0, 0, 3, 0,
       3, 3, 0, 0, 3, 1, 3, 0, 3, 0, 1, 3, 0, 0, 1, 1, 0, 1, 3, 1, 0, 3,
       3, 0, 3, 1, 0, 1, 1, 3, 1, 3, 3, 0, 0, 1, 3, 0, 0, 3, 0, 1, 0, 1,
       3, 1, 0, 0, 0, 3, 0, 1, 0, 0, 1, 1, 3, 0, 0, 1, 3, 3, 1, 3, 1, 3,
       3, 3, 1, 3, 3, 1, 1, 1, 0, 0, 1, 1, 0, 3], dtype=int64)
```

In [66]:
```
accuracy_score(y_test,y_pred3)
```

Out[66]:
```
0.9133333333333333
```

In [67]:
```
print(classification_report(y_test,y_pred3))
```

```
              precision    recall  f1-score   support

           0       0.88      0.86      0.87       100
           1       0.86      0.88      0.87       100
           3       1.00      1.00      1.00       100

    accuracy                           0.91       300
   macro avg       0.91      0.91      0.91       300
weighted avg       0.91      0.91      0.91       300
```

In [68]:
```
model3.best_estimator_
```

Out[68]:
```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_features=4),
                   learning_rate=0.01, n_estimators=100, random_state=0)
```

Step6. [Build LogisticRegressionCV classifier]

In [69]:
```
model4 = LogisticRegressionCV(cv=4,Cs=5,penalty='l2')
```

In [70]:
```
model4.fit(X_train,y_train)
```

Out[70]:
```
LogisticRegressionCV(Cs=5, cv=4)
```

In [71]:
```
y_pred4=model4.predict(X_test)
y_pred4
```

Out[71]:
```
array([0, 1, 1, 1, 3, 1, 0, 0, 3, 3, 1, 3, 3, 3, 3, 1, 3, 3, 3, 0, 3, 1,
       0, 1, 0, 0, 0, 1, 0, 1, 3, 0, 3, 1, 1, 3, 0, 0, 3, 1, 0, 1, 0, 0,
       3, 0, 3, 3, 3, 1, 0, 0, 0, 1, 1, 0, 3, 3, 0, 3, 1, 1, 0, 1, 0, 3,
       3, 0, 3, 0, 0, 3, 0, 0, 1, 3, 3, 0, 3, 0, 3, 3, 1, 3, 1, 1, 1, 3,
       0, 3, 1, 3, 1, 1, 1, 3, 1, 0, 3, 1, 3, 0, 0, 3, 1, 3, 1, 0, 1, 0,
       1, 0, 0, 0, 1, 1, 1, 1, 3, 0, 0, 0, 0, 0, 1, 3, 3, 1, 1, 1, 1, 3,
       0, 1, 0, 3, 3, 3, 3, 3, 3, 1, 0, 3, 1, 1, 3, 3, 3, 0, 1, 1, 0, 0,
       0, 3, 0, 1, 3, 3, 1, 1, 1, 1, 0, 1, 0, 0, 0, 3, 0, 1, 1, 3, 0, 0,
       0, 0, 1, 3, 1, 0, 1, 1, 3, 1, 1, 0, 3, 0, 1, 3, 0, 3, 0, 1, 3, 0,
       1, 1, 3, 1, 0, 0, 0, 3, 3, 1, 1, 3, 3, 3, 3, 1, 0, 1, 0, 0, 3, 0,
       3, 3, 0, 0, 3, 1, 3, 0, 3, 1, 1, 3, 0, 1, 1, 1, 0, 0, 3, 1, 0, 3,
       3, 0, 3, 1, 0, 1, 1, 3, 1, 3, 3, 0, 0, 1, 3, 0, 0, 3, 0, 1, 0, 1,
       3, 1, 1, 0, 0, 3, 0, 1, 0, 0, 0, 1, 3, 0, 0, 1, 3, 3, 1, 3, 1, 3,
       3, 3, 1, 3, 3, 1, 1, 1, 0, 0, 1, 1, 0, 3], dtype=int64)
```

In [72]:
```
accuracy_score(y_test,y_pred4)
```

Out[72]:
```
1.0
```

In [73]:
```
print(classification_report(y_test,y_pred4))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       100
           1       1.00      1.00      1.00       100
           3       1.00      1.00      1.00       100

    accuracy                           1.00       300
   macro avg       1.00      1.00      1.00       300
weighted avg       1.00      1.00      1.00       300
```

Step 7 [Build VotingClassifier]

In [74]:
```
model5=VotingClassifier(estimators=[('lr',model4),('gbc',base2)],voting='soft')
```

In [75]:
```
model5.fit(X_train,y_train)
```

Out[75]:
```
VotingClassifier(estimators=[('lr', LogisticRegressionCV(Cs=5, cv=4)),
                             ('gbc',
                              AdaBoostClassifier(base_estimator=DecisionTreeCla
ssifier(max_features=4),
                                                 random_state=0))],
                 voting='soft')
```

In [76]:
```
y_pred5=model5.predict(X_test)
y_pred5
```

Out[76]:
```
array([0, 1, 1, 1, 3, 0, 0, 0, 3, 3, 1, 3, 3, 3, 3, 1, 3, 3, 3, 0, 3, 1,
       0, 1, 0, 0, 0, 0, 0, 1, 3, 0, 3, 1, 1, 3, 0, 0, 3, 1, 1, 1, 0, 0,
       3, 0, 3, 3, 3, 1, 1, 0, 0, 1, 1, 0, 3, 3, 0, 3, 1, 1, 0, 1, 0, 3,
       3, 0, 3, 0, 0, 3, 0, 0, 1, 3, 3, 0, 3, 0, 3, 3, 1, 3, 1, 1, 1, 3,
       0, 3, 1, 3, 1, 1, 1, 3, 1, 0, 3, 1, 3, 0, 0, 3, 0, 3, 1, 0, 1, 0,
       1, 1, 0, 0, 0, 1, 1, 1, 3, 0, 0, 0, 0, 0, 1, 3, 3, 1, 1, 1, 1, 3,
       1, 1, 0, 3, 3, 3, 3, 3, 3, 0, 0, 3, 1, 1, 3, 3, 3, 0, 1, 1, 0, 0,
       0, 3, 0, 1, 3, 3, 1, 1, 0, 1, 0, 1, 0, 0, 0, 3, 0, 0, 1, 3, 0, 0,
       1, 1, 1, 3, 0, 0, 1, 1, 3, 1, 1, 0, 3, 1, 1, 3, 0, 3, 0, 1, 3, 0,
       1, 1, 3, 1, 0, 0, 0, 3, 3, 1, 1, 3, 3, 3, 3, 1, 0, 0, 0, 0, 3, 0,
       3, 3, 0, 0, 3, 1, 3, 0, 3, 0, 1, 3, 0, 0, 1, 1, 0, 1, 3, 1, 0, 3,
       3, 0, 3, 1, 0, 1, 1, 3, 1, 3, 3, 0, 0, 1, 3, 0, 0, 3, 0, 1, 0, 1,
       3, 1, 0, 0, 0, 3, 0, 1, 0, 0, 1, 1, 3, 0, 0, 1, 3, 3, 1, 3, 1, 3,
       3, 3, 1, 3, 3, 1, 1, 1, 0, 0, 1, 1, 0, 3], dtype=int64)
```

In [77]:
```
accuracy_score(y_test,y_pred5)
```

Out[77]:
```
0.93
```

In [78]:
```
print(classification_report(y_test,y_pred5))
```

```
              precision    recall  f1-score   support

           0       0.88      0.91      0.90       100
           1       0.91      0.88      0.89       100
           3       1.00      1.00      1.00       100

    accuracy                           0.93       300
   macro avg       0.93      0.93      0.93       300
weighted avg       0.93      0.93      0.93       300
```

Step8. [Interpret your results]

GradientBoostingClassifier(n_estimators=50)

GradientBoostingClassifier(n_estimators=50,learning_rate=1.0,max_depth=1,random_state=3

In [79]:
```
model6 = GradientBoostingClassifier(n_estimators=50,learning_rate=1.0,max_depth=1
```

In [80]:
```
model6.fit(X_train,y_train)
```

Out[80]:
```
GradientBoostingClassifier(learning_rate=1.0, max_depth=1, n_estimators=50,
                           random_state=32)
```

In [81]:
```
y_pred6=model6.predict(X_test)
y_pred6
```

Out[81]:
```
array([0, 1, 1, 1, 3, 1, 0, 0, 3, 3, 1, 3, 3, 3, 3, 1, 3, 3, 3, 0, 3, 1,
       0, 1, 0, 0, 0, 1, 0, 1, 3, 0, 3, 1, 1, 3, 0, 0, 3, 1, 0, 1, 0, 0,
       3, 0, 3, 3, 3, 1, 0, 0, 0, 1, 1, 0, 3, 3, 0, 3, 1, 1, 0, 1, 0, 3,
       3, 0, 3, 0, 0, 3, 0, 0, 1, 3, 3, 0, 3, 0, 3, 3, 1, 3, 1, 1, 1, 3,
       0, 3, 1, 3, 1, 1, 1, 3, 1, 0, 3, 1, 3, 0, 0, 3, 1, 3, 1, 0, 1, 0,
       1, 0, 0, 0, 1, 1, 1, 1, 3, 0, 0, 0, 0, 0, 1, 3, 3, 1, 1, 1, 1, 3,
       0, 1, 0, 3, 3, 3, 3, 3, 3, 1, 0, 3, 1, 1, 3, 3, 3, 0, 1, 1, 0, 0,
       0, 3, 0, 1, 3, 3, 1, 1, 1, 1, 0, 1, 0, 0, 0, 3, 0, 1, 1, 3, 0, 0,
       0, 0, 1, 3, 1, 0, 1, 1, 3, 1, 1, 0, 3, 0, 1, 3, 0, 3, 0, 1, 3, 0,
       1, 1, 3, 1, 0, 0, 0, 3, 3, 1, 1, 3, 3, 3, 3, 1, 0, 1, 0, 0, 3, 0,
       3, 3, 0, 0, 3, 1, 3, 0, 3, 1, 1, 3, 0, 1, 1, 1, 0, 0, 3, 1, 0, 3,
       3, 0, 3, 1, 0, 1, 1, 3, 1, 3, 3, 0, 0, 1, 3, 0, 0, 3, 0, 1, 0, 1,
       3, 1, 1, 0, 0, 3, 0, 1, 0, 0, 0, 1, 3, 0, 0, 1, 3, 3, 1, 3, 1, 3,
       3, 3, 1, 3, 3, 1, 1, 1, 0, 0, 1, 1, 0, 3], dtype=int64)
```

In [82]:
```
accuracy_score(y_test,y_pred6)
```

Out[82]:
```
1.0
```

In [83]:
```
print(classification_report(y_test,y_pred6))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       100
           1       1.00      1.00      1.00       100
           3       1.00      1.00      1.00       100

    accuracy                           1.00       300
   macro avg       1.00      1.00      1.00       300
weighted avg       1.00      1.00      1.00       300
```

AdaBoostClassifier

AdaBoostClassifier(base_estimator=DecisionTreeClassifier(), learning_rate=0.01, n_estimators=75, random_state=0)

In [84]:
```
base3 = AdaBoostClassifier(base_estimator=base,learning_rate=0.01,n_estimators=75
```

In [85]:
```
model7 = GridSearchCV(base3,param_grid,cv=5,n_jobs=-1)
```

In [86]:
```
model7.fit(X_train,y_train)
```

Out[86]:
```
GridSearchCV(cv=5,
             estimator=AdaBoostClassifier(base_estimator=DecisionTreeClassifier
(max_features=4),
                                          learning_rate=0.01, n_estimators=75,
                                          random_state=0),
             n_jobs=-1,
             param_grid={'learning_rate': [0.01, 0.001],
                         'n_estimators': [100, 150, 200]})
```

In [87]:
```
y_pred7=model7.predict(X_test)
y_pred7
```

Out[87]:
```
array([0, 1, 1, 1, 3, 0, 0, 0, 3, 3, 1, 3, 3, 3, 3, 1, 3, 3, 3, 0, 3, 1,
       0, 1, 0, 0, 0, 0, 0, 1, 3, 0, 3, 1, 1, 3, 0, 0, 3, 1, 1, 1, 0, 0,
       3, 0, 3, 3, 3, 1, 1, 0, 0, 1, 1, 0, 3, 3, 1, 3, 1, 1, 0, 1, 0, 3,
       3, 0, 3, 0, 0, 3, 0, 0, 1, 3, 3, 0, 3, 0, 3, 3, 1, 3, 1, 1, 1, 3,
       0, 3, 1, 3, 1, 1, 1, 3, 1, 1, 3, 1, 3, 0, 0, 3, 0, 3, 1, 0, 1, 0,
       1, 1, 0, 0, 0, 1, 1, 1, 3, 0, 0, 0, 0, 0, 1, 3, 3, 1, 1, 1, 1, 3,
       1, 1, 0, 3, 3, 3, 3, 3, 3, 0, 0, 3, 1, 1, 3, 3, 3, 0, 1, 1, 0, 0,
       0, 3, 0, 1, 3, 3, 1, 1, 0, 1, 0, 1, 1, 0, 0, 3, 0, 0, 1, 3, 0, 0,
       1, 1, 1, 3, 0, 0, 1, 1, 3, 1, 1, 0, 3, 1, 1, 3, 0, 3, 1, 1, 3, 0,
       1, 1, 3, 1, 1, 0, 0, 3, 3, 1, 1, 3, 3, 3, 3, 1, 0, 0, 0, 0, 3, 0,
       3, 3, 0, 0, 3, 1, 3, 0, 3, 0, 1, 3, 0, 0, 1, 1, 0, 1, 3, 1, 0, 3,
       3, 0, 3, 1, 0, 1, 1, 3, 1, 3, 3, 0, 0, 1, 3, 0, 0, 3, 0, 1, 0, 1,
       3, 1, 0, 0, 0, 3, 0, 1, 0, 0, 1, 1, 3, 0, 0, 1, 3, 3, 1, 3, 1, 3,
       3, 3, 1, 3, 3, 1, 1, 1, 0, 0, 1, 1, 0, 3], dtype=int64)
```

In [88]:
```
accuracy_score(y_test,y_pred7)
```

Out[88]:
```
0.9133333333333333
```

In [89]:
```
print(classification_report(y_test,y_pred7))
```

```
              precision    recall  f1-score   support

           0       0.88      0.86      0.87       100
           1       0.86      0.88      0.87       100
           3       1.00      1.00      1.00       100

    accuracy                           0.91       300
   macro avg       0.91      0.91      0.91       300
weighted avg       0.91      0.91      0.91       300
```

```
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import accuracy_score, classification_report
from sklearn.ensemble import GradientBoostingClassifier, AdaBoost classifier,
from sklearn.linear_model import LogisticRegressionCV.
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier CV.
from sklearn.preprocessing import LabelEncoder.
```

Step:1

```
Original = pd.read_csv("Human_Activity_Data.csv")
Original.head()
Original.columns
Original.shape
Original.dtypes.
Original.info()
Original.value_counts()
label_encoder = LabelEncoder()
Original["Label_Activity"] = label_encoder.fit_transform
                              (original["Activity"])
```

Step:2

```
original.Activity.value_counts()
Original.label_Activity.value_counts()
```

Lab10. Patients Physical Activities Prediction using Boosting

Objectives

In this lab, you will recognize physical activities such as 'laying', 'sitting' or 'walking' using Gradient Boosting, AdaBoost and VotingClassifiers.

Learning Outcomes

After completing this lab, you will be able to

- Create a small dataset with selected rows based on fewer target labels
- Build GradientBoostingClassifier, fit and predict on test data
- Print accuracy and classification report
- Find the best no. of decision trees and learning rate using GridSearch and Cross Validation
- Build AdaBoost classifier model with GridSearchCV, fit and predict
- Select best parameter values for n_estimators and learning_rate
- Build LogisticRegressionCV model, fit, predict and print scores
- Build VotingClassifier using other models, fit, predict and print scores
- Interpret results and parameter values
- Change parameter values and play around with models

Business Use Case

As you a data scientist, one popular hospital in your city has asked you to design a model based on the mobile phone data of their patients. Your model will help the hospital to understand the physical activity level of their patients. The physical activities are classified into 3 levels - Laying, Sitting and Walking. Based on the prediction, Doctors will recommend health diets, exercise and physical activities to the new patients.

Dataset

You will use a dataset named, ***Human_Activity_Data.csv***.

Step1. [Understand Data]

- Using Pandas, import "***Human_Activity_Data.csv*** " file and print properties such as head, shape, columns, dtype, info and value_counts.

Step2. [Build a small dataset]

- As it is a big dataset, execution time will be long for training and testing. So build a small dataset with only 3 classes, laying, sitting and walking, where each class with 500 samples. So, shape of this new dataframe will be (1500, 563). That is 1500 rows and 563 features.
- Store this new dataframe as a CSV file.

Step3. [Build GradientBoostingClassifier]

- Import your new reduced CSV file
- Print basic properties of the new CSV file
- Split it into training set and test set (30% samples for testing)
- Create GradientBoostingClassifier, fit and predict
- Print accuracy and classification report

Step4. [Find Best no. of trees and Best Learning Rate using Grid Search and Cross Validation]

- Create GridSearchCV model with GradientBoostingClassifier
- Parameters: *param_grid = {'n_estimators': [50, 100, 200, 400], 'learning_rate': [0.1, 0.01]}*
- Perform fit and predict
- Print accuracy, classification report
- Print best parameters such as best no. of trees and learning rate. Use the attribute best_estimator_

Take first 3000 samples:

```
tem1 = original [original ['Activity'] == 'LAYING'] [:500]
tem2 = original [original [original ['Activity'] == 'SITTING'] [:500]
tem3 = original [original ['Activity'] == 'WALKING'] [:500]
tem4 = original [original ['Activity'] == 'STANDING'] [:500]
tem5 = original [original ['Activity'] == 'WALKING_UPSTAIRS'
tem6 = original [original ['Activity'] == 'WALKING_DOWNSTAIRS' [:500] [:500]
new_df = pd.concat ([tem1, tem2, tem3, tem4, tem5 tem6])
new_df.to_csv ("human_activity_clipped 3000.csv")
new_df.head()
new_df.shape
new_df.columns
new_df.dtypes
new_df.value_counts()
X = new_df.drop ['Activity', 'label_Activity'], axis=1)
y = new_df.label_Activity
X_train, X_test, y_train, y_test = train_test_split
                                   (X, y, test_size=0.2,
```

Build Gradient Boosting:

```
model = GradientBoostingClassifier (subsample=0.5, n_estimators=100,
model.fit (X_train, y_train)
y_pred = model.predict(X_test)
print (accuracy_score (y_test, y_pred))
print (classification_report (y_test, y_pred))
```

Step5. [Build AdaBoostClassifier]

- Create AdaBoostClassifier with DecisionTreeClassifier
- Create GridSearchCV with AdaBoostClassifier model that you created as before
- Parameters: *param_grid = {'n_estimators': [100, 150, 200], 'learning_rate': [0.01, 0.001]}*
- Perform fit, predict
- Print accuracy, classification report
- Print best parameters such as best no. of trees and learning rate. Use the attribute best_estimator_

Step6. [Build LogisticRegressionCV classifier]

- Create a LogisticRegressionCV model with the parameters Cs=5, cv=4, penalty='l2'.
- Perform fit and predict
- Print classification report

Step7. [Build VotingClassifier]

- Build VotingClassifier model with GradientBoostingClassifier and LogisticRegressionCV that you created in the previous steps
- Perform fit and predict operations
- Print classification report

Step8. [Interpret your results]

- Analyze your results
- Change parameters and play with your code

```
base = DecisionTree.classifier(max_features=4)
base2 = AdaBoostClassifier(base_estimator=base, random_state=0)
param_grid = {'n_estimators': [100,150,200], 'learning_rate': [0.01, 0.001]}
model2 = GridSearchCV(base2, param_grid, cv=10, n_jobs=-1)
model2.fit(x_train, y_train)
y_pred2 = model2.predict(x_test)
Print(accuracy_score(y_test, y_pred2))
print(classification_report(y_test, y_pred2))
model2_best_estimator_
Build Logistic Regression
model3 = LogisticRegressionCV(cv=4, Cs=5, penalty='l2')
model3.fit(x_train, y_train)
y_pred3 = model3.predict(x_test)
```

```
Print (accuracy_score(y_test, y_pred3))
print (classification_report (y_test, y_pred3))

Find Best No of trees:

param_grid = {'n_estimators': [50, 100, 200, 400],
              'learning_rate' = [0.1, 0.01]}

all_scores = cross_val_score(estimator= model_, X= X_train,
                             y = y_train, cv=5)
print(all_scores)

model_4 = GridSearchCV(estimator = model4, param_grid = param
                       -grid, cv=5, n= jobs=-1),

model4 fit (X_train, y_train)

y_pred4 = model4. predict (x_test)

Point (accuracy_score (y_test, y_pred4))

print (Classification_report (y_test, y_pred4))

model4_best = estimator_

Build Voting classifier for 3000 samples:

model5 = VotingClassifier (estimators = [('lr', model3_),
                                         ('gbc', base 2)], voting = 'soft'
model5 fit (x_train, y_train)

y_pred5 = model5_predict (x_test)

Print (accuracy_score (y_test, y_pred5))

print (classification_report (y_test, y_pred5))

From this 3000 samples you should:

temp1 = new_df [new_df ['Activity'] == 'LAYING'] [:500]
temp2 = new df [new_df ['Activity'] == 'SITTING'] [:500]
```

NOTE

temp3 = raw_df[raw_df['Activity'] == 'WALKING'][:500]

df = pd.concat([temp1, temp2, temp3])

df.to_csv("human_activity_clipped1500.csv")

df = pd.read_csv("human_activity_clipped1500.csv")

df.head()

df.shape

df.columns

df.dtypes

df.info()

df.value_counts()

Step:3 [Build GradientBoostingClassifier]

```
X = df.drop(['Activity', 'label_Activity'], axis=1)
y = df.label_Activity
X_train, X_test, y_train, y_test = train_test_split
                                   (X, y, test_size = 0.2)
model = GradientBoostingClassifier(subsample = 0.5, n_estimators=100,
                                   learning_rate =1.0)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
y_pred
```

accuracy_score(y_test, y_pred)

print(classification_report(y_test, y_pred))

step:4

param_grid = {'n_estimators': [50, 100, 200, 400], 'learning_rate': [0.1, 0.01]}

all_scores = cross_val_score(estimator=model, X=x_train, y=y_train, cv=5)

print(all_scores)

model2 = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, n_jobs=-1)

print(all_scores)

model2 = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, n_job=-1)

model2.fit(x_train, y_train)

y_pred2 = model2.predict(x_test)

y_pred2

accuracy_score(y_test, y_pred2)

model2.best_estimator_

step:5

base = DecisionTreeClassifier(max_features=4)

base2 = AdaBoostClassifier(base_estimator=base, random_state=0)

param_grid = {'n_estimators': [100, 150, 200], 'learning_rate': [0.001, 0.01]}

model3 = GridSearchCV(base2, param_grid, cv=10, n_jobs=-1)

model3.fit(x_train, y_train)

y_pred3 = model3.predict(x_test)

y_pred3

accuracy_score(y_test, y_pred3)

print(classification_report(y_test, y_pred3))

NOTE

model 3. best_estimator_

Step :6

```
model4 = LogisticRegressionCV (CV=4, CS=5, penalty='l2')
model4.fit (X_train, y_train)
y_pred4 = model4.predict (X_test)
y_pred4
accuracy_score (y_test, y_pred4)
print (classification_report (y_test, y_pred4))
```

Step: 7

```
model5 = VotingClassifier (estimators=[('lr', model4),
                                       ('gbc', base2)], voting='soft')
model5.fit (X_train, y_train)
y_pred5 = model5.predict (X_test)
y_pred5
accuracy: score (y_test, y_pred5)
print (classification_report (y_test, y_pred5))
```

Step:8

GradientBoostingClassifier

```
model6 = GradientBoostingClassifier (n_estimators=50,
                                     learning_rate=1.0,
model6.fit (X_train, y_train)
y_pred6 = model6.predict (X_test)
y_pred6
```

```
accuracy_score(y_test, y_pred6)

print(classification_report(y_test, y_pred6))
```

Ada Boost Classifier:-

```
base3 = AdaBoostClassifier(base_estimator=base,
model7 = GridSearchCV(base3, param_grid, cv=5, n_jobs=-1)
model7.fit(x_train, y_train)
y_pred7 = model7.predict(x_test)
y_pred7.
accuracy_score(y_test, y_pred7)
print(classification_report(y_test, y_pred7))
```

REPORT

Lab10.Patients Physical Activities Prediction using Boosting

In this lab we have learned to recognize patient's physical activities such as laying, sitting or walking using Gradient Boosting, AdaBoost and VotingClassifiers.

After checking the properties of dataset, we have seen that we have huge set of data so will split data by dependent values such as laying sitting or walking.

So after creating a small dataset with the shape of (1500,563) rows and columns and creating a gradient boosting classifier with 30% of testing size the perform fit and predict also print accuracy and classification report then find best no. of decision trees and learning rate using GridSearch and Cross Validation.

Then build AdaBoost classifier model with GridSearchCV perform fit and predict then print accuracy score then do the same for LogisticRegressionCV model and VotingClassifier then finally interpret results and parameter values.

Make changes with the parameter to get better prediction.

lab-11

In [1]:
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.decomposition import PCA
import seaborn as sns
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import warnings
warnings.filterwarnings('ignore')
import scipy.cluster.hierarchy as shc
from sklearn.cluster import MeanShift,AgglomerativeClustering
import statistics
```

Step1. [Understand Data]

- Using Pandas, import “Mall_Customers.csv” file and print properties such as head, shape, columns, dtype, info and value_counts.
- For example: customers_data = pd.read_csv("Mall_Customers.csv")

In [2]:
```
import pandas as pd
data = pd.read_csv("Mall_Customers.csv")
data.head()
```

Out[2]:

	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

In [3]:
```
data.shape
```

Out[3]:
```
(200, 5)
```

In [4]:
```
data.columns
```

Out[4]:
```
Index(['CustomerID', 'Genre', 'Age', 'Annual Income (k$)',
       'Spending Score (1-100)'],
      dtype='object')
```

In [5]:
```
data.dtypes
```

Out[5]:
```
CustomerID                int64
Genre                    object
Age                       int64
Annual Income (k$)        int64
Spending Score (1-100)    int64
dtype: object
```

In [6]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              200 non-null    int64
 1   Genre                   200 non-null    object
 2   Age                     200 non-null    int64
 3   Annual Income (k$)      200 non-null    int64
 4   Spending Score (1-100)  200 non-null    int64
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```

In [7]:
```
data.value_counts()
```

Out[7]:
```
CustomerID  Genre   Age  Annual Income (k$)  Spending Score (1-100)
1           Male    19   15                  39                        1
138         Male    32   73                  73                        1
128         Male    40   71                  95                        1
129         Male    59   71                  11                        1
130         Male    38   71                  75                        1
                                                                      ..
70          Female  32   48                  47                        1
71          Male    70   49                  55                        1
72          Female  47   49                  42                        1
73          Female  60   50                  49                        1
200         Male    30   137                 83                        1
Length: 200, dtype: int64
```

Step2. [Label encode gender]

- Genre (ie., gender) is a string, so label encode into binary

In [8]:
```
data.drop(['CustomerID'],axis=1,inplace=True)
```

In [9]:
```
label_encoder = LabelEncoder()
data["Genre"] = label_encoder.fit_transform(data["Genre"])
data
```

Out[9]:

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

200 rows × 4 columns

In [10]:
```
data.Genre.value_counts()
```

Out[10]:
```
0    112
1     88
Name: Genre, dtype: int64
```

In [11]:
```
df = data.copy()
```

Step3. [Check for variance]

- Use describe() on your data frame and check for variance. If variance is high for float columns, you need to normalize. Otherwise, ignore

In [12]:
```
data.describe(include='all')
```

Out[12]:

	Genre	Age	Annual Income (k$)	Spending Score (1-100)
count	200.000000	200.000000	200.000000	200.000000
mean	0.440000	38.850000	60.560000	50.200000
std	0.497633	13.969007	26.264721	25.823522
min	0.000000	18.000000	15.000000	1.000000
25%	0.000000	28.750000	41.500000	34.750000
50%	0.000000	36.000000	61.500000	50.000000
75%	1.000000	49.000000	78.000000	73.000000
max	1.000000	70.000000	137.000000	99.000000

In [13]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Genre                   200 non-null    int32
 1   Age                     200 non-null    int64
 2   Annual Income (k$)      200 non-null    int64
 3   Spending Score (1-100)  200 non-null    int64
dtypes: int32(1), int64(3)
memory usage: 5.6 KB
```

In [14]:
```
data.var()
```

Out[14]:
```
Genre                     0.247638
Age                     195.133166
Annual Income (k$)      689.835578
Spending Score (1-100)  666.854271
dtype: float64
```

In [15]:
```
data.corr()
```

Out[15]:

	Genre	Age	Annual Income (k$)	Spending Score (1-100)
Genre	1.000000	0.060867	0.056410	-0.058109
Age	0.060867	1.000000	-0.012398	-0.327227
Annual Income (k$)	0.056410	-0.012398	1.000000	0.009903
Spending Score (1-100)	-0.058109	-0.327227	0.009903	1.000000

Step4. [Check skewness]

- Check if float columns are skewed. Use skew() on your data frame. If skew value is greater than 0.75, then you can perform log transformation on those skew columns.

In [16]:
```
data.skew()
```

Out[16]:
```
Genre                     0.243578
Age                       0.485569
Annual Income (k$)        0.321843
Spending Score (1-100)   -0.047220
dtype: float64
```

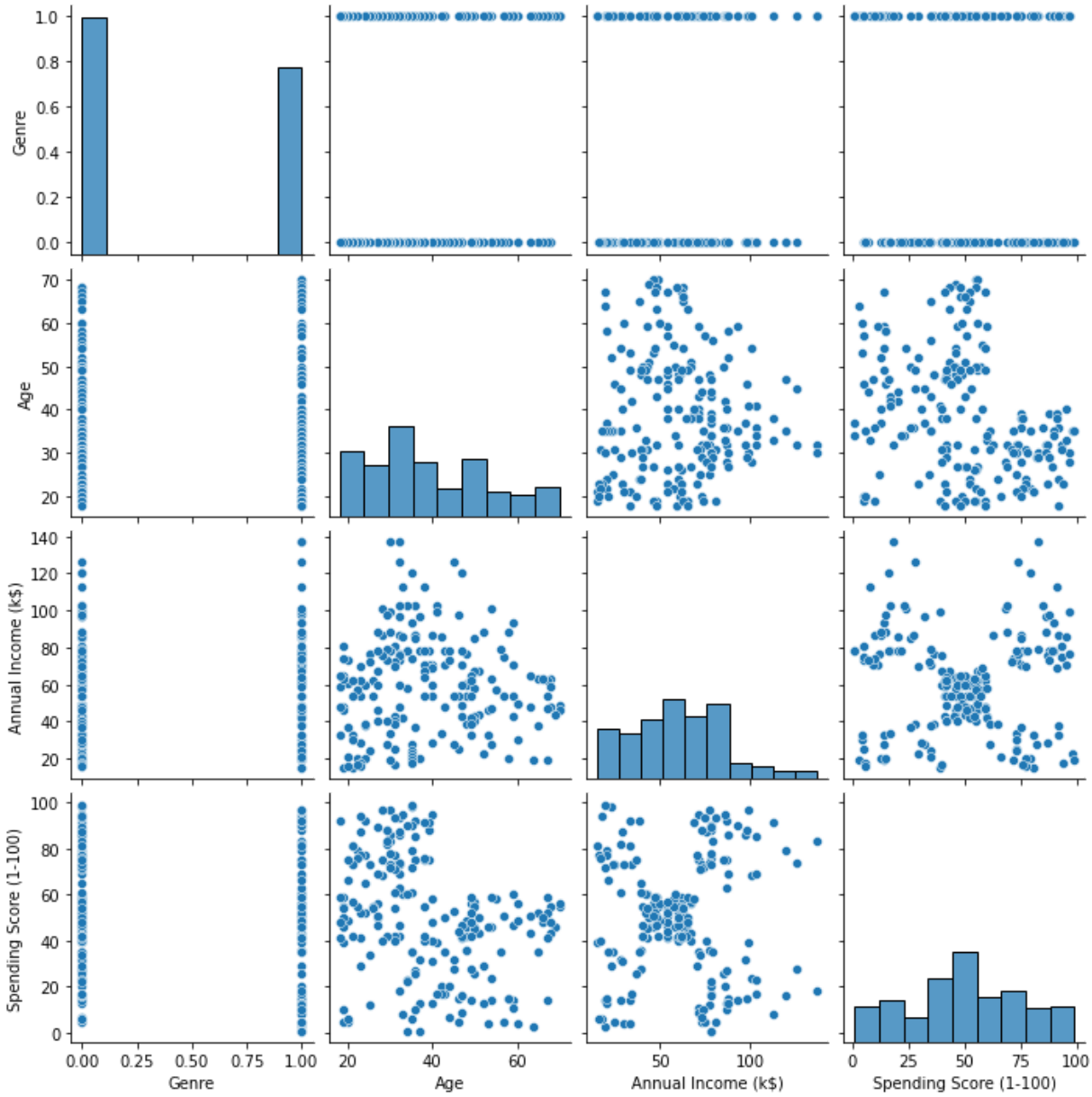
In [17]:
```
data.sort_values(by=["Genre","Age","Annual Income (k$)","Spending Score (1-100)"]
```

Step5. [Pair plot]

- Draw pair plot and observe correlations

In [18]:
```
sns.pairplot(data)
```

Out[18]:
```
<seaborn.axisgrid.PairGrid at 0x1f9ac65cd30>
```



Step6. [Build KMeans]

- Create and fit KMeans (n_clusters variable can be set with any value)
- Print label_ and cluster_centers_ values

In [19]:
```
model = KMeans(n_clusters=5)
```

In [20]:
```
model.fit(data)
```

Out[20]:
```
KMeans(n_clusters=5)
```

In [21]:
```
model.labels_
```

Out[21]:
```
array([0, 0, 0, 0, 0, 0, 3, 0, 0, 3, 0, 3, 1, 1, 0, 0, 1, 0, 0, 0, 3, 3,
       1, 0, 1, 0, 0, 0, 3, 1, 0, 1, 1, 0, 1, 1, 2, 0, 2, 2, 2, 2, 0, 1,
       3, 2, 1, 2, 4, 3, 2, 1, 1, 0, 1, 2, 2, 2, 2, 4, 2, 2, 4, 2, 2, 2,
       2, 0, 0, 0, 1, 4, 0, 4, 4, 4, 1, 0, 4, 1, 1, 0, 0, 0, 3, 0, 0, 4,
       0, 0, 0, 0, 3, 0, 0, 0, 3, 1, 1, 0, 1, 0, 3, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 3, 3, 1, 1, 0, 0, 0, 3, 1, 0, 3, 1, 1, 0, 3, 1, 1, 0,
       0, 3, 2, 2, 0, 0, 1, 2, 1, 2, 3, 2, 2, 4, 3, 4, 3, 1, 1, 0, 2, 2,
       2, 2, 2, 0, 0, 2, 2, 2, 0, 0, 0, 3, 2, 2, 2, 4, 4, 2, 2, 2, 0, 0,
       2, 0, 0, 0, 1, 4, 4, 1, 0, 0, 4, 4, 4, 0, 4, 0, 0, 4, 4, 4, 3, 0,
       0, 0])
```

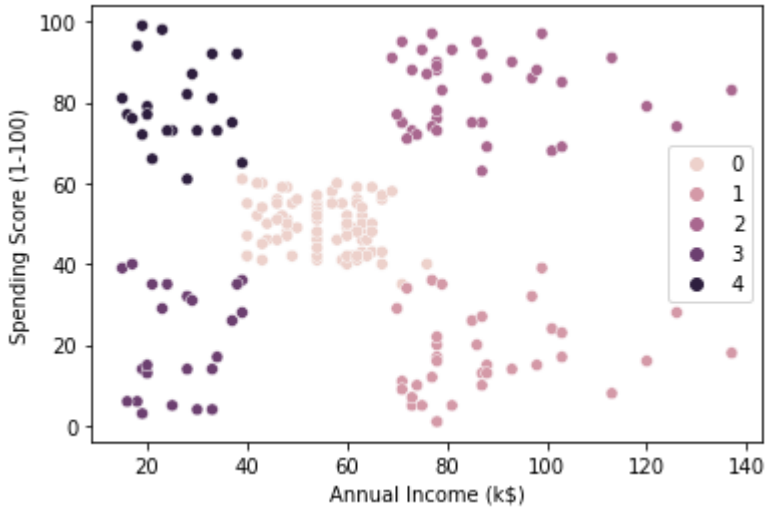
In [22]:
```
model.cluster_centers_
```

Out[22]:
```
array([[ 0.4125    , 42.9375    , 55.0875    , 49.7125    ],
       [ 0.52777778, 40.66666667, 87.75      , 17.58333333],
       [ 0.46153846, 32.69230769, 86.53846154, 82.12820513],
       [ 0.39130435, 45.2173913 , 26.30434783, 20.91304348],
       [ 0.40909091, 25.27272727, 25.72727273, 79.36363636]])
```

Step7. [Scatter plot]

- Draw scatter plot between any two features with hue as “labels_”.

In [23]:
```
sns.scatterplot(data['Annual Income (k$)'],data['Spending Score (1-100)'],hue=moc
plt.show()
```



Step8. [Cluster Analysis].

Now, predict cluster labels for the same data. For example,

```
kmeans2 = KMeans(n_clusters = 5, init='k-means++')
kmeans2.fit(customers_data)
pred = kmeans2.predict(customers_data)
```

Now, add a new column for pred in a new dataframe, such as

```
frame = pd.DataFrame(customers_data)
frame['cluster'] = pred
```

This will create a new column to frame. That means, you have added a cluster prediction column, whose values say the cluster number to which the row belongs to. That is, that customer belongs to that cluster number.

Now, group customers based on cluster number. Remember, here we have 5 clusters from 0 to 4.

For each cluster group, print the following details.

```
Average age: 45.21739130434783
Average annual income: 26.304347826086957
Deviation of the mean for annual income: 7.893811054517766
No of customers ie shape: (23, 5)
From those customers we have 9 male and 14 female
```

In [24]:
```
data.head()
```

Out[24]:

	Genre	Age	Annual Income (k$)	Spending Score (1-100)
70	1	70	49	55
60	1	70	46	56
57	1	69	44	46
108	1	68	63	43
102	1	67	62	59

In [25]:
```
x = data.iloc[:,[2,3]].values
```

In [26]:
```
kmeans2 = KMeans(n_clusters = 5, init='k-means++')
kmeans2.fit(x)
pred = kmeans2.predict(x)
```

In [27]:
```
frame = pd.DataFrame(x)
frame['cluster'] = pred
data['cluster'] = pred
```

In [28]:
```
frame.cluster.value_counts()
```

Out[28]:
```
2    81
0    39
3    35
1    23
4    22
Name: cluster, dtype: int64
```

In [29]:
```
frame
```

Out[29]:

	0	1	cluster
0	49	55	2
1	46	56	2
2	44	46	2
3	63	43	2
4	62	59	2
...	...	...	...
195	37	75	4
196	16	6	1
197	65	50	2
198	63	54	2
199	65	48	2

200 rows × 3 columns

In [30]:
```
d_frameC0 = data[data['cluster'] == 0]
d_frameC1 = data[data['cluster'] == 1]
d_frameC2 = data[data['cluster'] == 2]
d_frameC3 = data[data['cluster'] == 3]
d_frameC4 = data[data['cluster'] == 4]
```

In [31]:
```
print("Average age for cluster 0 :",d_frameC0['Age'].mean())
print("Average Annual Income for cluster 0 :",d_frameC0['Annual Income (k$)'].mea
print("Deviation of the mean for annual income :",statistics.stdev(d_frameC0['Anr
print("No.of customers i.e shape of cluster 0 :",d_frameC0.shape)
print("From those customers we have",d_frameC0.Genre.value_counts()[1],"male and'
```

```
Average age for cluster 0 : 32.69230769230769
Average Annual Income for cluster 0 : 86.53846153846153
Deviation of the mean for annual income : 16.312484972924967
No.of customers i.e shape of cluster 0 : (39, 5)
From those customers we have 18 male and 21 female
```

In [32]:
```
print("Average age for cluster 1 :",d_frameC1.Age.mean())
print("Average Annual Income for cluster 1 :",d_frameC1['Annual Income (k$)'].mea
print("Deviation of the mean for annual income :",statistics.stdev(d_frameC1['Anr
print("No.of customers i.e shape of cluster 1 :",d_frameC1.shape)
print("From those customers we have",d_frameC1.Genre.value_counts()[1],"male and'
```

```
Average age for cluster 1 : 45.21739130434783
Average Annual Income for cluster 1 : 26.304347826086957
Deviation of the mean for annual income : 7.893811054517766
No.of customers i.e shape of cluster 1 : (23, 5)
From those customers we have 9 male and 14 female
```

In [33]:
```
print("Average age for cluster 2 :",d_frameC2.Age.mean())
print("Average Annual Income for cluster 2 :",d_frameC2['Annual Income (k$)'].mea
print("Deviation of the mean for annual income :",statistics.stdev(d_frameC2['Ann
print("No.of customers i.e shape of cluster 2 :",d_frameC2.shape)
print("From those customers we have",d_frameC2.Genre.value_counts()[1],"male and'
```

```
Average age for cluster 2 : 42.71604938271605
Average Annual Income for cluster 2 : 55.2962962962963
Deviation of the mean for annual income : 8.988109429190942
No.of customers i.e shape of cluster 2 : (81, 5)
From those customers we have 33 male and 48 female
```

In [34]:
```
print("Average age for cluster 3 :",d_frameC3.Age.mean())
print("Average Annual Income for cluster 3 :",d_frameC3['Annual Income (k$)'].mea
print("Deviation of the mean for annual income :",statistics.stdev(d_frameC3['Ann
print("No.of customers i.e shape of cluster 3 :",d_frameC3.shape)
print("From those customers we have",d_frameC3.Genre.value_counts()[1],"male and'
```

```
Average age for cluster 3 : 41.114285714285714
Average Annual Income for cluster 3 : 88.2
Deviation of the mean for annual income : 16.399067405334545
No.of customers i.e shape of cluster 3 : (35, 5)
From those customers we have 19 male and 16 female
```

In [35]:
```
print("Average age for cluster 4 :",d_frameC4.Age.mean())
print("Average Annual Income for cluster 4 :",d_frameC4['Annual Income (k$)'].mea
print("Deviation of the mean for annual income :",statistics.stdev(d_frameC4['Ann
print("No.of customers i.e shape of cluster 4 :",d_frameC4.shape)
print("From those customers we have",d_frameC4.Genre.value_counts()[1],"male and'
```

```
Average age for cluster 4 : 25.272727272727273
Average Annual Income for cluster 4 : 25.727272727272727
Deviation of the mean for annual income : 7.566730552584204
No.of customers i.e shape of cluster 4 : (22, 5)
From those customers we have 9 male and 13 female
```

Step9. [Find the best number of clusters]

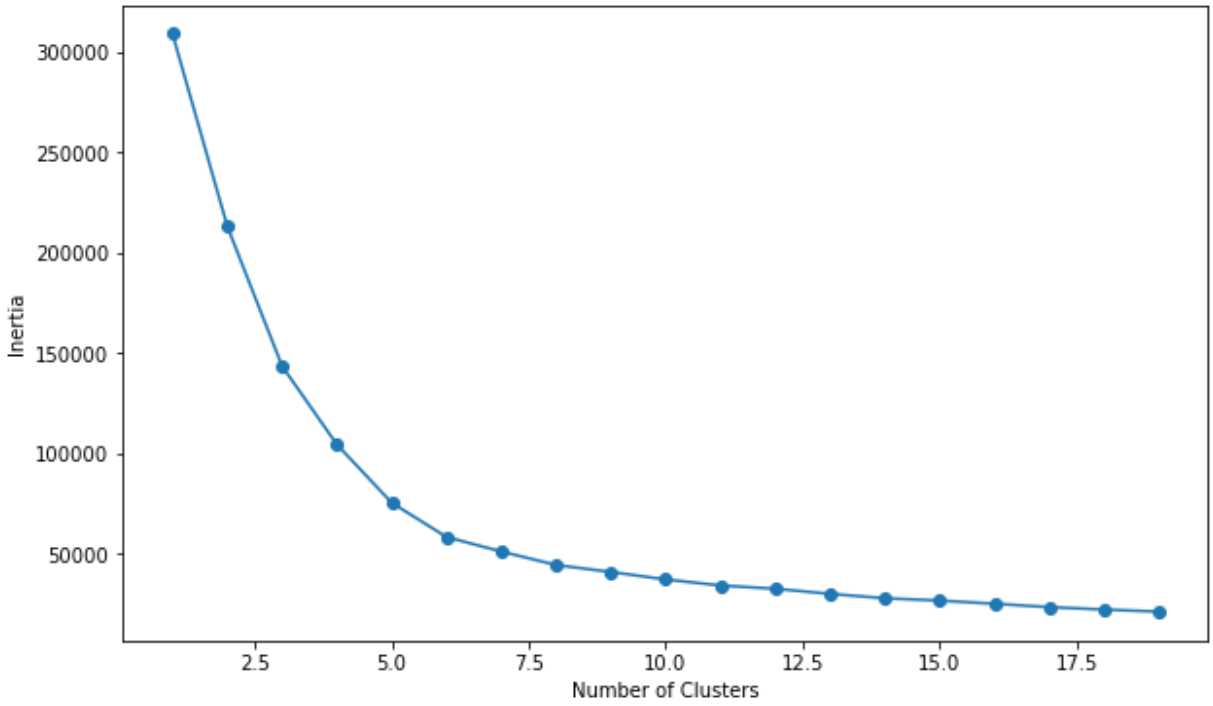
Compute inertia value as shown below

```
SSE = []
for clust in range(1,20):
   km = KMeans(n_clusters=clust, init="k-means++")
   km = km.fit(customers_data)
   SSE.append(km.inertia_)
```

Plot a line chart between cluster number and its inertia value. You will get graph as below. Can you identify the best number of clusters from this graph?

In [36]:
```
SSE = []
for cluster in range(1,20):
    km = KMeans(n_clusters=cluster, init="k-means++")
    km.fit(data)
    SSE.append(km.inertia_)
```

In [37]:
```
plt.figure(figsize=(10,6))
plt.plot(np.arange(1,20),SSE,"o-")
plt.xlabel("Number of Clusters")
plt.ylabel("Inertia")
plt.show()
```



Step10. [Reduce Dimensions using PCA]

- Reduce 4 dimensions into 2 dimensions using PCA
- Create KMeans model, fit on the reduced dataset
- Print cluster_centers_ and labels_

In [38]:
```
y = data
```

In [39]:
```
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(y)
PCA_components = pd.DataFrame(principalComponents)
```

In [40]:
```
PCA_components
```

Out[40]:

	0	1
0	-8.937770	-7.994355
1	-9.925562	-10.990670
2	-18.762148	-6.975124
3	-9.703074	9.949852
4	2.444558	-0.205293
...	...	...
195	9.052637	-35.780641
196	-57.422825	-12.963479
197	6.206044	1.085767
198	8.162067	-2.831962
199	4.825819	2.106376

200 rows × 2 columns

In [41]:
```
model1 = KMeans(n_clusters=5)
```

In [42]:
```
model1.fit(PCA_components)
```

Out[42]:
```
KMeans(n_clusters=5)
```

In [43]:
```
model1.cluster_centers_
```

Out[43]:
```
array([[ -4.4197535 ,  -3.08886972],
       [ 41.58214146,   1.76309289],
       [-10.12791659,  42.35498015],
       [  4.82223183, -46.69462455],
       [-44.3923333 ,  -9.92438654]])
```

In [44]:
```
model1.labels_
```

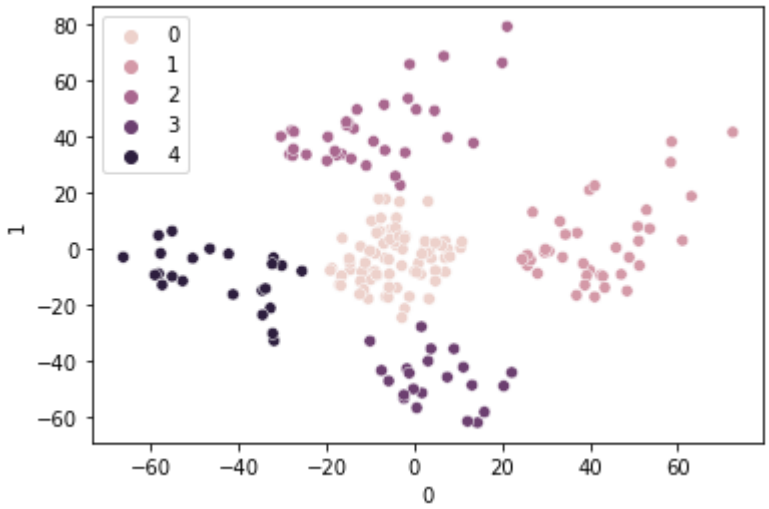
Out[44]:
```
array([0, 0, 0, 0, 0, 0, 4, 0, 0, 4, 0, 4, 2, 2, 0, 0, 2, 0, 0, 0, 4, 4,
       2, 0, 2, 0, 0, 0, 4, 2, 0, 2, 2, 0, 2, 2, 1, 0, 1, 1, 1, 1, 0, 2,
       4, 1, 2, 1, 3, 4, 1, 2, 2, 0, 2, 1, 1, 1, 1, 3, 1, 1, 3, 1, 1, 1,
       1, 0, 0, 0, 2, 3, 0, 3, 3, 3, 2, 0, 3, 2, 2, 0, 0, 0, 4, 0, 0, 3,
       0, 0, 0, 0, 4, 0, 0, 0, 4, 2, 2, 0, 2, 0, 4, 2, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 4, 4, 2, 2, 0, 0, 0, 4, 2, 0, 4, 2, 2, 0, 4, 2, 2, 0,
       0, 4, 1, 1, 0, 0, 2, 1, 2, 1, 4, 1, 1, 3, 4, 3, 4, 2, 2, 0, 1, 1,
       1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 4, 1, 1, 1, 3, 3, 1, 1, 1, 0, 0,
       1, 0, 0, 0, 0, 3, 3, 0, 0, 0, 3, 3, 3, 0, 3, 0, 0, 3, 3, 3, 4, 0,
       0, 0])
```

Step11. [Scatter plot]

- Draw a scatter plot between the 2 reduced dimensions, with hue as label_
- Your scatter plot may look like below

In [45]:
```
sns.scatterplot(PCA_components[0],PCA_components[1],hue=model1.labels_)
```

Out[45]:
```
<AxesSubplot:xlabel='0', ylabel='1'>
```



Step12. [MeanShift clustering]

- Create MeanShift clustering model and fit on the reduced data of PCA and visualize clusters on the reduced data, as shown below.

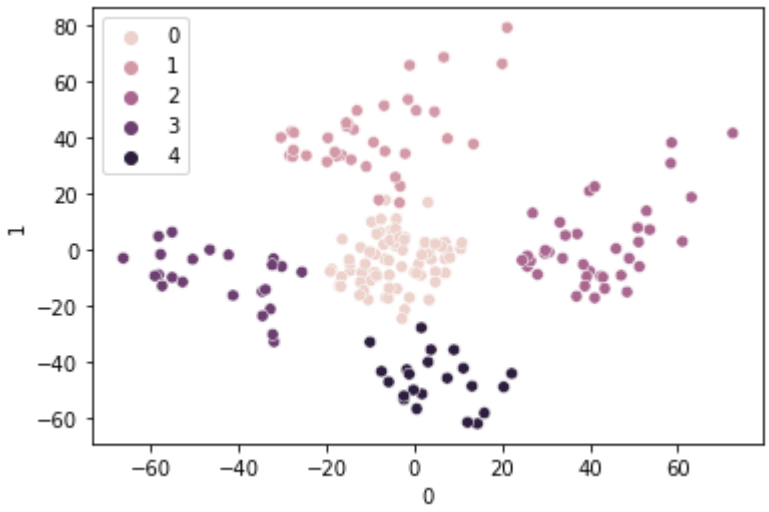
In [46]:
```
model2 = MeanShift(bandwidth=25)
model2.fit(PCA_components)
```

Out[46]:
```
MeanShift(bandwidth=25)
```

In [47]:
```
cluster_centers = model2.cluster_centers_
```

In [48]:
```
sns.scatterplot(PCA_components[0],PCA_components[1],hue=model.labels_)
```

Out[48]:
```
<AxesSubplot:xlabel='0', ylabel='1'>
```



Step13. [Predict hierarchical clusters using AgglomerativeClustering]

- Create 5 clusters, using AgglomerativeClustering class. Your dendrogram will look like as below.

In [49]:
```
model3 = AgglomerativeClustering(n_clusters=5,linkage='ward',compute_full_tree=Tr
model3.fit(df)
```

Out[49]:
```
AgglomerativeClustering(compute_full_tree=True, n_clusters=5)
```

In [50]:
```
model3.labels_
```

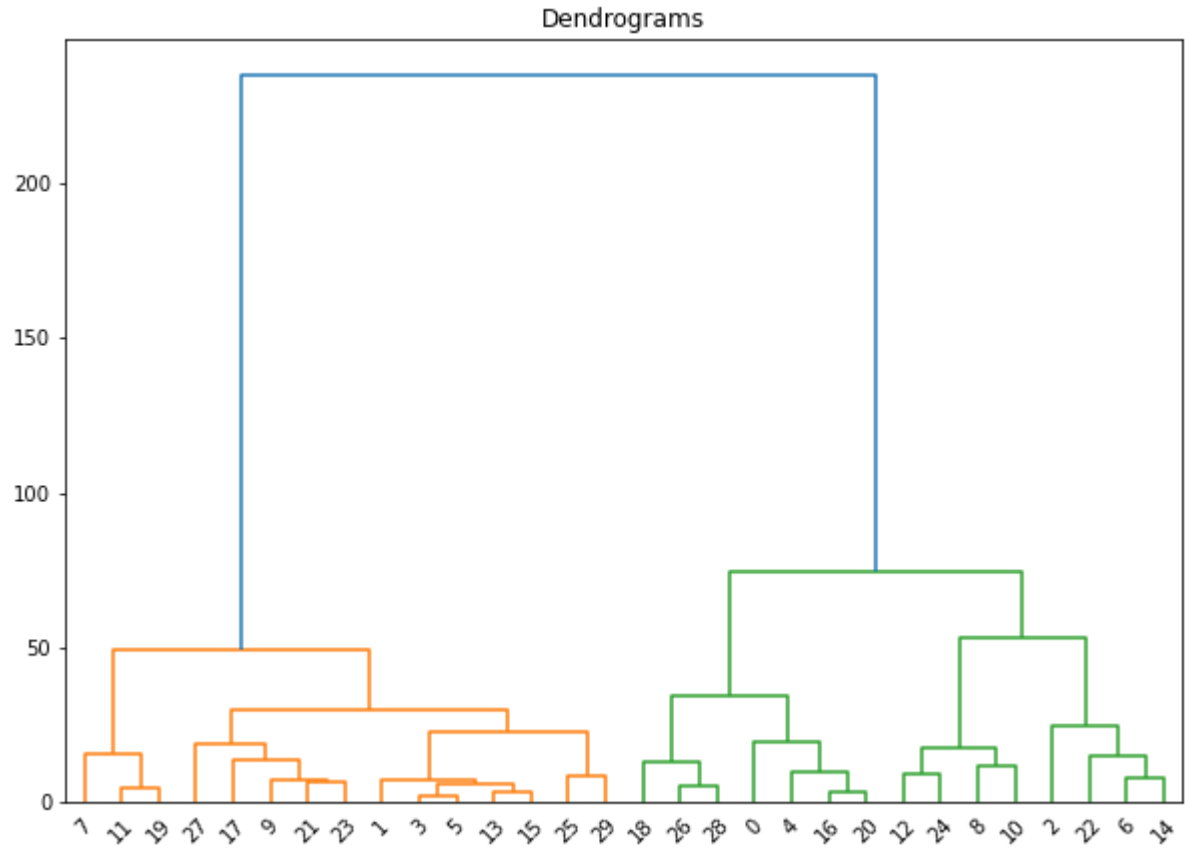
Out[50]:
```
array([4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3,
       4, 3, 4, 3, 4, 0, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 0,
       4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2, 1, 2, 1, 2, 1, 2,
       0, 2, 1, 2, 1, 2, 1, 2, 1, 2, 0, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2,
       1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2,
       1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2,
       1, 2], dtype=int64)
```

In [51]:
```
frame = df.copy()
frame['cluster'] = model3.labels_
```

In [52]:
```
frame.value_counts()
```

Out[52]:
```
Genre  Age  Annual Income (k$)  Spending Score (1-100)  cluster
0      18   65                  48                      0          1
1      29   28                  82                      3          1
       24   60                  52                      0          1
       25   24                  73                      3          1
            77                  12                      1          1
                                                                   ..
0      41   99                  39                      1          1
            103                 17                      1          1
       42   34                  17                      4          1
       43   48                  50                      0          1
1      70   49                  55                      0          1
Length: 200, dtype: int64
```

In [55]:
```
plt.figure(figsize=(10,7))
plt.title("Dendrograms")
dend = shc.dendrogram(shc.linkage(frame[:30], method='ward'))
```



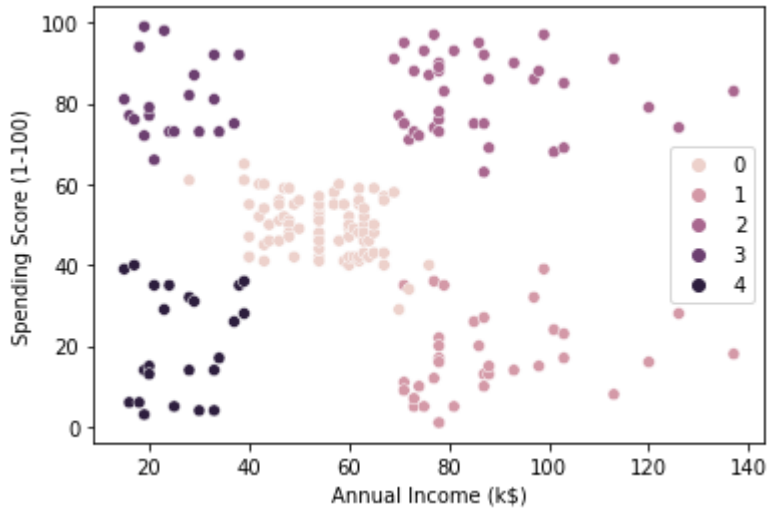
Step14. [Visualize scatter plot with hue as agglomerativeclustering

labels_]

- Visualize agglomerative clusters using the predicted label. Select any two features for X and Y with hue as labels_. Your scatter plot will look like below

In [54]:
```
sns.scatterplot(frame['Annual Income (k$)'],frame['Spending Score (1-100)'],hue=m
```

Out[54]:
```
<AxesSubplot:xlabel='Annual Income (k$)', ylabel='Spending Score (1-100)'>
```



import pandas as pd

from sklearn.preprocessing import LabelEncoder.

from sklearn.decomposition import pca

import seaborn as sns.

from sklearn.preprocessing import standardscalar.

import matplotlib.pyplot as plt.

import numpy as np

from sklearn.cluster.import K-Means.

from sklearn.decomposing import PCA

import warnings

warnings.filterwarning('ignore')

import scipy.cluster.hierarchy as shc.

from sklearn.cluster import Meanshift. Agglomerativeclustering

import statistics.

import numpy as np.

=) X = np.array([[1,2],[1,4],[1,4],[7,0],[10,2],[10,4],[10,4],[10,0]])

=) X = X[:,0]

Y = X[:,1]

sns.scatterplot(X,Y)

Step: 1

=) import pandas as pd.

data = pd.read_csv("Mall_customers.csv")

data.head()

=) km = KMeans(n_clusters=3, random_state=0)

km.fit(X)

=) km.cluster_centers

X = X[:,0]

Y = X[:,0]

sns.scatterplot(X,Y, hue = km.labels)

=) km.predict([[2,3]])

=) data.shape

=) data.columns

=) data.dtypes.

=) data.info()

=) data.value_counts()

Lab11. Shopping Mall Customer Segmentation using Clustering

Objectives

In this lab, you will detect clusters from customer data and perform analytics to understand customers who visit malls, using KMeans and Agglomerative Clustering methods.

Learning Objectives

After completing this lab, you will be able to

- Perform Skew analysis and interpretation
- Check if data normalization is required
- Build KMeans model with the required no. of clusters
- Visualize clusters based on selected features
- Select the best value for K using inertia error values and Elbow method
- Perform Cluster Analysis to understand cluster statistics
- Reduce dimensions of data using PCA and build KMeans model
- Build MeanShift clustering model on dimensions reduced data
- Create hierarchical clusters using Agglomerative Clustering
- Visualize hierarchical clusters using Dendrogram

Business Use Case

You are given the data of customers who have visited your mall. The details of customers such as age, annual income, spending score and gender are collected for each customer. They have asked you to analyse this data and give insights. This will help them to design promotion strategies, marketing models and others so as to reach out specific group of customers. Also, those group of customers will be targeted via surveys to collect further details. Based on that feedback we can decide whether the new strategy is good for that customer segment or not, even before the strategy is released.

Step1. [Understand Data]

- Using Pandas, import ***"Mall_Customers.csv"*** file and print properties such as head, shape, columns, dtype, info and value_counts.
- For example: **customers_data = pd.read_csv("Mall_Customers.csv")**

Step2. [Label encode gender]

- Genre (ie., gender) is a string, so label encode into binary

Step3. [Check for variance]

- Use describe() on your data frame and check for variance. If variance is high for float columns, you need to normalize. Otherwise, ignore

Step4. [Check skewness]

- Check if float columns are skewed. Use skew() on your data frame. If skew value is greater than 0.75, then you can perform log transformation on those skew columns.

Step5. [Pair plot]

- Draw pair plot and observe correlations.

Step6. [Build KMeans]

- Create and fit KMeans (n_clusters variable can be set with any value)
- Print label_ and cluster_centers_ values

Step7. [Scatter plot]

- Draw scatter plot between any two features with hue as "labels_". This figure shows 5 clusters.

Step 2:

=> data.drop(['Customers ID'], axis=1, Inplace = True)

=> label_encodn = LabelEncodn()

data['Genre'] = label_encoder.fit_transform(data['Genre'])

data

=> data.course.value_counts()

=> df = data.copy()

Step: 3

=> data.describe('include = 'all')

=> data.info()

=> data.var()

=> data.Cor()

Step: 4

=> data.show()

=> data.sort_values(by= ["Genre", "Age", "Annual Income (k$)",

encoding= False, Inplace = True)

Step: 5

=> sns_pairplot(data)

Step: 6

=> model. = kMeans(n_clusters = 5)

=> model.fit(data)

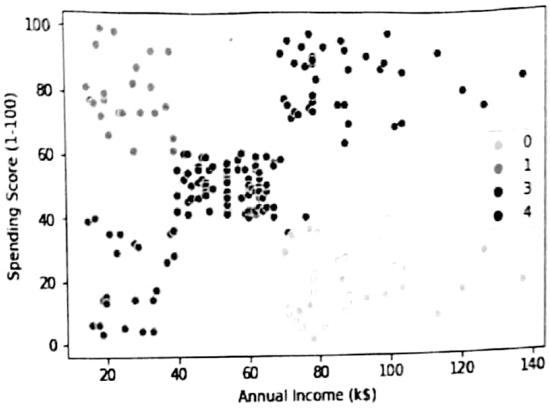
=> model.labels_

=> model.clusters_centers_

=> sns.scatterplot(data['Age'], data["Annual Income(k$)"],

plt.show()

hue = model.labels_)



Step8. [Cluster Analysis].

Now, predict cluster labels for the same data. For example,

```
kmeans2 = KMeans(n_clusters = 5, init='k-means++')
kmeans2.fit(customers_data)
pred = kmeans2.predict(customers_data)
```

Now, add a new column for pred in a new dataframe, such as

```
frame = pd.DataFrame(customers_data)
frame['cluster'] = pred
```

This will create a new column to frame. That means, you have added a cluster prediction column, whose values say the cluster number to which the row belongs to. That is, that customer belongs to that cluster number.

Now, group customers based on cluster number. Remember, here we have 5 clusters from 0 to 4.

For each cluster group, print the following details.

```
Average age:  45.21739130434783
Average annual income:  26.304347826086957
Deviation of the mean for annual income:  7.893811054517766
No of customers ie shape:  (23, 5)
From those customers we have 9 male and 14 female
```

Step9. [Find the best number of clusters]

Compute inertia value as shown below

```
SSE = []
for clust in range(1,20):
    km = KMeans(n_clusters=clust, init="k-means++")
    km = km.fit(customers_data)
    SSE.append(km.inertia_)
```

Plot a line chart between cluster number and its inertia value. You will get graph as below. Can you identify the best number of clusters from this graph?

Step: 7

=> Sns.Scatterplot (data['Annual Income (k$)'], data['spending Score(1-100)'], hue = model.labels').

plt.show()

Step: 8

=> data.head()

=> x = data.iloc[:, [2,3]].values.

=> kmeans2 = kMeans(n_clusters=5, init='k-means++')

kMeans2.fit(x)

pred = kmeans2.predict(x)

=> frame = pd.DataFrame(x).

frame['cluster'] = pred.

data['Clusters'] = pred

=> frame.cluster.values_counts().

=> frame.

=> d_frame c0 = data[data['Cluster'] == 0]

d.frame C1 = data[data['cluster'] == 1]

d.frame C2 = data[data['Cluster'] == 2]

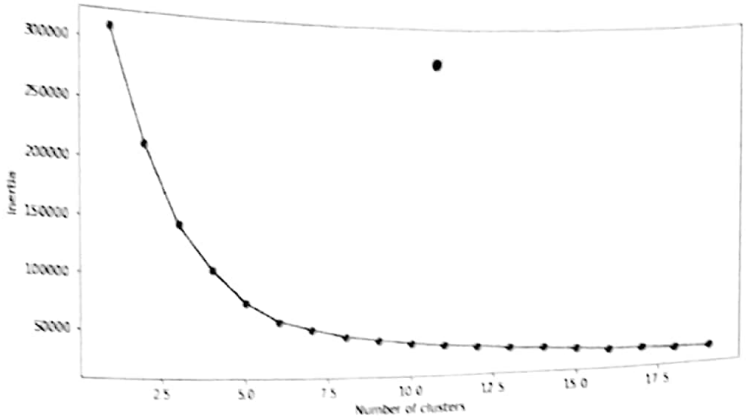
d.frame C3 = data[data['Cluster'] == 3]

d.frame C4 = data[data['cluster'] == 4]

=> Print("Average age for cluster 0:", d_frame['Age'].mean())

Print("Average Annual Income for cluster 0:", d_frame['Annual Income (k$)'].mean())

print("Deviation of the mean for annual Income:", statistics.stdev(d_frameC0['Annual Income (k$)']))

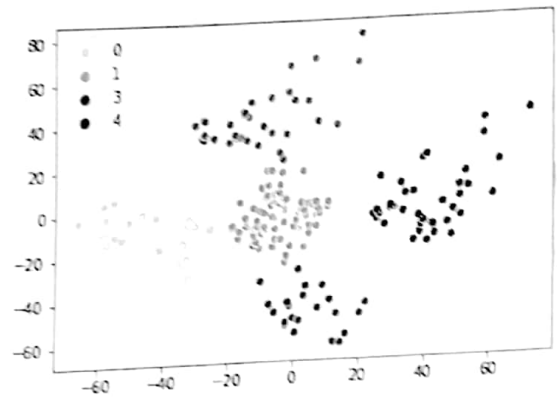


Step10. [Reduce Dimensions using PCA]

- Reduce 4 dimensions into 2 dimensions using PCA
- Create KMeans model, fit on the reduced dataset
- Print cluster_centers_ and labels_

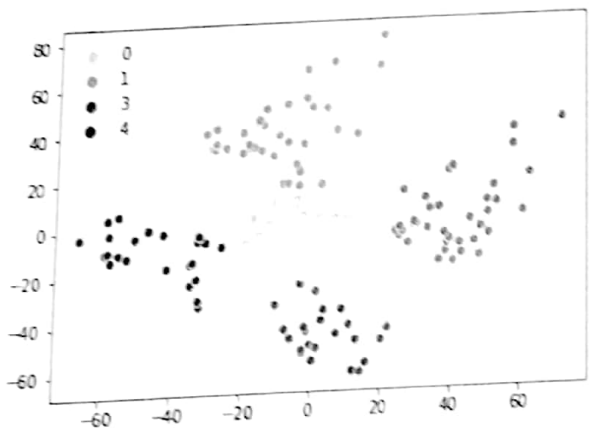
Step11. [Scatter plot]

- Draw a scatter plot between the 2 reduced dimensions, with hue as label_
- Your scatter plot may look like below



Step12. [MeanShift clustering]

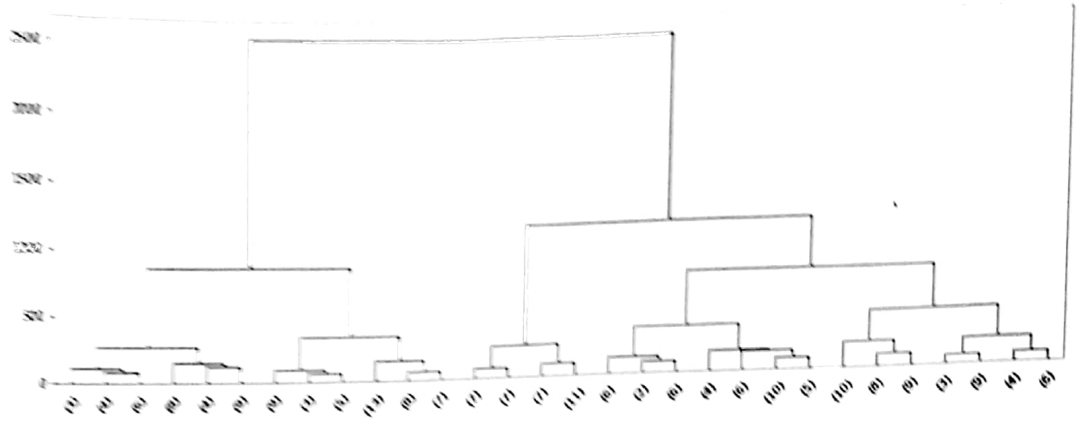
- Create MeanShift clustering model and fit on the reduced data of PCA and visualize clusters on the reduced data, as shown below.



```
print("No. of Customers & Shape of cluster 0:", d.frame[0.shape)
print("From those customers we have", d.frame[0].Curse_value_counts
                                        ($[0], "female")
=> print("Average age for cluster 1:", d_frame[1, Age.mean()).
   print("Deviation of the Mean for annual Income:", statistics.stdev.
                    {d.frame.(1('Annual Income(k$)']]).
   print("No. of customers & shape of cluster 1:", d_func 1. shape)
   print("From those customers we have", d.frame(1.Genre, value_counts
                                                    ()[1]
   "male end", d.frame.(1, Genre.value_counts()[:,], "female")
=> print("Average age for cluster 2:", d.frame (2. Age.mean())
   print("Average Annual Income for cluster 2:", d.frame (2("Annual Income
                                                    (k$)']mean()]
   print("Deviation of the Mean for annual income:", statistics.stdev.
                    cd.frame2["Annual Income(k$)']))
=> print("Average age for cluster 3:", d.fram (3. Age. mean())
   print("Average Annual Income for cluster 3:", d_from()['Annual Income
                                        (k$)']].mean()]
   print("Deviation of the mean for annual Income:", statistics.stdev.
                        cd.from(3['Annual Income(k$)']))
   Print("No. of Customers is shape of cluster 3:", d.frame().shape)
   Print("From those customers we have", d.frame().Genre.
        value_counts(3[2], "male end", d.frame.Genre.Vdir.
                                        counts()[0] female")
```

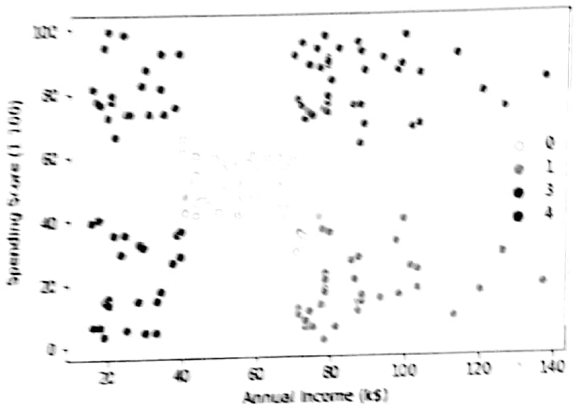
Step 13. [Predict hierarchical clusters using AgglomerativeClustering]

- Create 5 clusters using AgglomerativeClustering class. Your dendrogram will look like as below.



Step 14. [Visualize scatter plot with hue as agglomerativeclustering labels_]

- Visualize agglomerative clusters using the predicted label. Select any two features for X and Y with hue as labels_. Your scatter plot will look like below



5) print ("Average age for Cluster 4:", d.frame[4].Age.mean()).

print ("Average Annual Income for cluster 4:" d.frame[4]['Annual Income (k$)'].mean()).

Print (" Decision of the Mean for annual Income:" statistics.mode(d.frame[4]['Annual Income (k$)']))

print(" No. of customers in Shape of cluster 4:", d.frame[4].shape).

print("From these customers we have", d.frame[4].Curve.value_counts()[1] "male and", d.frame[4].Curve.value_counts().counts()[0], "female")

Step 9:

```
=> SSE = [ ]
   for cluster in range(1, 20):
      km = KMeans(n_clusters = cluster, init = "k-means++")
      km.fit(data)
      SSE.append(km.inertia)
=> plt.figure(figsize=(10,6))
=> plt.plot(np.arange(1,20), SSE, "o-")
=> plt.xlabel("Number of clusters")
=> plt.ylabel("Inertia")
   plt.show()
```

Step 10:

```
=> Y=data
=> pca = PCA(n_components=2)
   principalComponents = pca.fit_transform(Y)
   PCA_components = pd.DataFrame(principalComponents)
=> PCA_components
=> model1 = KMeans(n_clusters=5)
=> model1.fit(PCA_components)
=> model1.cluster_centers_
=> model1.labels_
```

Step 11:

```
=> sns.scatterplot(PCA_components[0], PCA_components[1],
                   hue = model1.labels)
```

Step 12:

```
=> model2 = MeanShift(bandwidth = 25)
   model2.fit(PCA_components)
```

NOTE

=) cluster_centers = model3.cluster_centers.

=) sns.scatterplot (pca_components[0], pca_components[1], hue=model3.labels).

Step: 13:

=) model3 = AgglomerativeClustering(n_clusters = 5, linkage = 'ward', compute_full_tree = True)

model3.fit(df)

=) model3.labels =

=) frame = df.copy()

frame['Cluster'] = model3.labels_

=) frame.value_counts()

=) plt.figure(figsize=(10,5))

plt.title('Dendrogram').

dend = shc.dendrogram(shc.linkage(frame[:30], method = 'ward'))

Step: 14

=) sns.Scatterplot(frame['Annual Income (k$)'], frame['Spending Score (1-100)'], hue = model3.labels_)

REPORT

Lab11.Shopping Mall Customer Segmentation using Clustering

In this lab we have learned to detect clusters from customer data and perform analytics to understand customers who visit malls, using KMeans and Agglomerative Clustering methods.

This kind of problem comes under unsupervised machine learning where we won't train our model will just make understanding just by visualizing with various plot.

Our first step is to check the data and perform skew analysis and perform normalization if it is required on our case we don't need to normalize the data next we need to build KMeans model with number of clusters.

Select the best value for K using inertia error values and Elbow method then perform cluster analysis to understand cluster statistics.

Now reduce dimensions of data using PCA components and build KMeans model, MeanShift clustering model on dimension reduced data. Now Create Agglomerative Clustering then visualize hierarchical clusters using Dendrogram.