ALL MODELS

LINEAR REGRESSION - R2 SCORE

LINEAR REGRESSION WITH GRADIENT DESCENT

K-MEANS CLUSTER - ELBOW METHOD

MID-LAB - DECISION TREE - CATEGORICAL- roc, auc

MID-LAB - FEATURE SELECTION - LINEAR REGRESSION - R2 SCORE

HIERARCHIAL - SINGLE LINK - COMPLETE LINK - DENDROGRAMS FOR BOTH

K-NN IRIS - CONFUSION MATRIX

LOGISTIC REG - DIABETES - STD SCALER

NAIVE BAYES - CONFUSION MATRIX

SVM - PLOT IN 3D OR WHICHEVER NECESSARY

LINEAR

LINEAR WITH GRAD DESCENT

MULTIPLE LINEAR

K-NN

DECISION TREE

LOGISTIC REGRESSION

NAIVE BAYES

SVM

SVM WITH GRADIENT DECISION

PCA DIMENSIONALITY REDUCTION

CLUSTERING

- K MEANS
- HIERARCHIAL

iris-classification-knn

April 30, 2024

this problem is a multi class classification task of predicting the species of the plant based on 4 features (sepal len,wid and petal len,wid)

LOADING DATASET

[6]:
```
import pandas as pd

#loading the dataset as a pandas dataframe
df = pd.read_csv("iris.csv")

#prints first 5 cols of the dataset
df.head()
```

[6]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

[7]:
```
#shows no of rows and cols in the dataset
df.shape
```

[7]:
```
(150, 5)
```

[8]:
```
#summary statistics of the dataset(only the numerical columns)
df.describe()
```

[8]:

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

To know how many classes are present in the target variable and also what is the induvidual no of examples for each class, we can use group by method

[9]:
```
df.groupby('species').size()
```

[9]:
```
species
setosa        50
versicolor    50
virginica     50
dtype: int64
```

DATA PREPROCESSING:

[10]:
```
#checking if dataset has null values
df.isnull().sum()
```

[10]:
```
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64
```

as there is no null values in the dataset, we can go for further preprocessing steps

[15]:
```
#splitting dataset into features(x) and target col(y)

X = df.drop(['species'],axis=1).values
y = df['species'].values
```

as y has 3 classes, we need to encode it using label encoder

[20]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)
y
```

[20]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 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, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

next the dataset is split into train and test sets

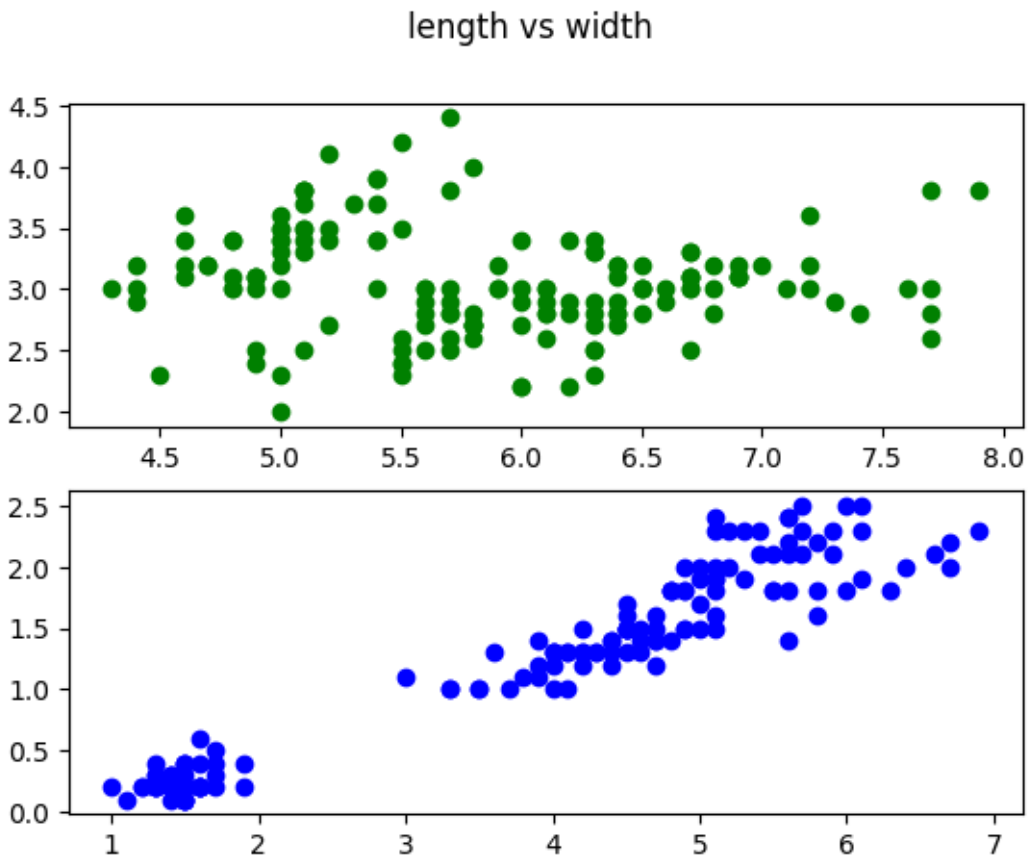
[37]:
```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,␣
 ↪random_state = 0)
```

DATA VISUALISATION

[38]:
```
#subplots for sepal len vs width and petal len vs width
import matplotlib.pyplot as plt
fig, axs = plt.subplots(2)
fig.suptitle('length vs width')
axs[0].plot(df.sepal_length, df.
 ↪sepal_width,ls='',marker='o',color='g',label='Sepal')
axs[1].plot(df.petal_length, df.
 ↪petal_width,ls='',marker='o',color='b',label='Petal')
```

[38]:
```
[<matplotlib.lines.Line2D at 0x7fa9f522e4c0>]
```



MODEL TRAINING

[39]:
```
#Fitting clasifier to the Training set
```

```
from sklearn.neighbors import KNeighborsClassifier

from sklearn.model_selection import cross_val_score

#Instantiate model (k = 3)
classifier = KNeighborsClassifier(n_neighbors=3)

#Fitting the model
classifier.fit(X_train, y_train)

#Predicting the Test set results
y_pred = classifier.predict(X_test)
```

MODEL EVALUATION

[40]:
```
#confusion matrix

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

[40]:
```
array([[16,  0,  0],
       [ 0, 17,  1],
       [ 0,  0, 11]])
```

[42]:
```
#classification report, it shows the precision, recall and F1 score of all the␣
 ↪3 classes separately and also the overall accuracy
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       1.00      0.94      0.97        18
           2       0.92      1.00      0.96        11

    accuracy                           0.98        45
   macro avg       0.97      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45
```

MODEL INTERPRETATION

=> In class 0, all the exaples in the test set belonging to this class has been correctly predicted

=> In class 1, only 94% of the samples actually belonging to this class have been perdicted correctly (recall=0.94)

=> In class 2, only 92% of the samples predicted as this class actually belong to class 2 (precision=0.92) y

=> In 98% of the cases, the model has predicted the class correctl

[ ]:

salary-lr

April 30, 2024

this problem is a simple regression problem of predicting the salary of an induvidual with using one feature, experience years

LOADING DATASET

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

dataset = pd.read_csv('Salary_Data.csv')
dataset.head()
```

[1]:

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

DATA PREPROCESSING

[2]:
```
#splitting feature and target cols
X = dataset.iloc[:,:-1].values
y = dataset.iloc[:,1].values
```

for using linear regression, the feature variable and target variable should have a corelation > 0.6, I have used spearman's corelation coefficient in this case

[4]:
```
from scipy.stats import spearmanr
corr, _ = spearmanr(X, y)
print('correlation: %.3f' % corr)
```

```
Spearmans correlation: 0.957
```

[5]:
```
#splitting train and test sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.
↪75,random_state=0)
```

MODEL TRAINING

[14]:
```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train,y_train)
```

[14]:
```
LinearRegression()
```

VISUALISATION OF THE REGRESSION LINE

[11]:
```
y_pred = regressor.predict(X_test)
y_pred
```

[11]:
```
array([ 39390.67454324, 119914.17685749,  63181.7093179 ,  61351.62971985,
       112593.85846529, 105273.54007308, 113508.89826431,  62266.66951888,
        74162.18690621,  97953.22168088,  52201.2317296 ,  72332.10730816,
        54946.35112667,  66841.86851401, 100698.34107795,  87887.7838916 ,
        37560.59494519, 121744.25645554,  53116.27152862,  45795.95313642,
        79652.42570037,  80567.46549939,  59521.5501218 ])
```

[12]:
```
import matplotlib.pyplot as plt
plt.scatter(X_train, y_train, color='red')
plt.plot(X_train, regressor.predict(X_train), color='blue')
plt.title("Salary vs Experience (Training set)")
plt.xlabel("Years of experience")
plt.ylabel("Salaries")
plt.show()
```

MODEL EVALUATION using R-square

[13]:
```
from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred)
print('r2 score: ', r2)
```

```
r2 score:  0.9448084643969682
```

[16]:
```
regressor.coef_
```

[16]:
```
array([9150.39799026])
```

[17]:
```
regressor.intercept_
```

[17]:
```
25665.077557856283
```

MODEL INTERPRETATION

=> slope of the equation is 9150.394.. which means that increase in 1 year of experience increases the salary by 9150... times

=> the intercept can be interpreted as the average salsry of a fresher/a person with 0 experience

[ ]:

midlab-linear-reg

April 30, 2024

21MIS1142 Abinaya R

[13]:
```
import numpy as np
import pandas as pd

df = pd.read_csv('auto_prize data.csv')
df.head()
```

[13]:

	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	bore	stroke	compression-ratio	horsepower	peak-rpm	city-mpg	highway-mpg	target
0	5	164	99.800003	176.600006	66.199997	54.299999	2337	109	3.19	3.4	10.0	102	5500	24	30	13950
1	5	164	99.400002	176.600006	66.400002	54.299999	2824	136	3.19	3.4	8.0	115	5500	18	22	17450
2	4	158	105.800003	192.699997	71.400002	55.700001	2844	136	3.19	3.4	8.5	110	5500	19	25	17710
3	4	158	105.800003	192.699997	71.400002	55.900002	3086	131	3.13	3.4	8.3	140	5500	17	20	23875
4	5	192	101.199997	176.800003	64.800003	54.299999	2395	108	3.50	2.8	8.8	101	5800	23	29	16430

[14]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 159 entries, 0 to 158
Data columns (total 16 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   symboling          159 non-null    int64
 1   normalized-losses  159 non-null    int64
```

```
 2   wheel-base          159 non-null    float64
 3   length              159 non-null    float64
 4   width               159 non-null    float64
 5   height              159 non-null    float64
 6   curb-weight         159 non-null    int64
 7   engine-size         159 non-null    int64
 8   bore                159 non-null    float64
 9   stroke              159 non-null    float64
 10  compression-ratio   159 non-null    float64
 11  horsepower          159 non-null    int64
 12  peak-rpm            159 non-null    int64
 13  city-mpg            159 non-null    int64
 14  highway-mpg         159 non-null    int64
 15  target              159 non-null    int64
dtypes: float64(7), int64(9)
memory usage: 20.0 KB
```

All features are numeric

We split the features and target variable

[15]:
```
X = df.iloc[:,:-1].values
y = df.iloc[:,-1].values
```

[16]:
```
df.corr()
```

[16]:

	symboling	normalized-losses	wheel-base	length	\
symboling	1.000000	0.518344	-0.520591	-0.336257	
normalized-losses	0.518344	1.000000	-0.060086	0.035541	
wheel-base	-0.520591	-0.060086	1.000000	0.871535	
length	-0.336257	0.035541	0.871535	1.000000	
width	-0.219186	0.109726	0.814991	0.838338	
height	-0.475185	-0.413702	0.555767	0.499251	
curb-weight	-0.251880	0.125858	0.810182	0.871291	
engine-size	-0.109453	0.207820	0.649206	0.725953	
bore	-0.256469	-0.031558	0.578159	0.646318	
stroke	-0.021285	0.063330	0.167449	0.121073	
compression-ratio	-0.138316	-0.127259	0.291431	0.184814	
horsepower	-0.003949	0.290511	0.516948	0.672063	
peak-rpm	0.199106	0.237697	-0.289234	-0.234074	
city-mpg	0.089550	-0.235523	-0.580657	-0.724544	
highway-mpg	0.149830	-0.188564	-0.611750	-0.724599	
target	-0.162794	0.202761	0.734419	0.760952	

	width	height	curb-weight	engine-size	bore	\
symboling	-0.219186	-0.475185	-0.251880	-0.109453	-0.256469	
normalized-losses	0.109726	-0.413702	0.125858	0.207820	-0.031558	
wheel-base	0.814991	0.555767	0.810182	0.649206	0.578159	

```
length              0.838338  0.499251     0.871291     0.725953  0.646318
width               1.000000  0.292706     0.870595     0.779253  0.572554
height              0.292706  1.000000     0.367052     0.111083  0.254836
curb-weight         0.870595  0.367052     1.000000     0.888626  0.645792
engine-size         0.779253  0.111083     0.888626     1.000000  0.595737
bore                0.572554  0.254836     0.645792     0.595737  1.000000
stroke              0.196619 -0.091313     0.173844     0.299683 -0.102581
compression-ratio   0.258752  0.233308     0.224724     0.141097  0.015119
horsepower          0.681872  0.034317     0.790095     0.812073  0.560239
peak-rpm           -0.232216 -0.245864    -0.259988    -0.284686 -0.312269
city-mpg           -0.666684 -0.199738    -0.762155    -0.699139 -0.590440
highway-mpg        -0.693338 -0.226136    -0.789338    -0.714095 -0.590850
target              0.843371  0.244836     0.893639     0.841496  0.533890

                     stroke  compression-ratio  horsepower  peak-rpm  \
symboling         -0.021285          -0.138316   -0.003949  0.199106
normalized-losses  0.063330          -0.127259    0.290511  0.237697
wheel-base         0.167449           0.291431    0.516948 -0.289234
length             0.121073           0.184814    0.672063 -0.234074
width              0.196619           0.258752    0.681872 -0.232216
height            -0.091313           0.233308    0.034317 -0.245864
curb-weight        0.173844           0.224724    0.790095 -0.259988
engine-size        0.299683           0.141097    0.812073 -0.284686
bore              -0.102581           0.015119    0.560239 -0.312269
stroke             1.000000           0.243587    0.148804 -0.011312
compression-ratio  0.243587           1.000000   -0.162305 -0.416769
horsepower         0.148804          -0.162305    1.000000  0.074057
peak-rpm          -0.011312          -0.416769    0.074057  1.000000
city-mpg          -0.020055           0.278332   -0.837214 -0.052929
highway-mpg       -0.012934           0.221483   -0.827941 -0.032777
target             0.160664           0.209361    0.759874 -0.171916

                   city-mpg  highway-mpg    target
symboling          0.089550     0.149830 -0.162794
normalized-losses -0.235523    -0.188564  0.202761
wheel-base        -0.580657    -0.611750  0.734419
length            -0.724544    -0.724599  0.760952
width             -0.666684    -0.693338  0.843371
height            -0.199738    -0.226136  0.244836
curb-weight       -0.762155    -0.789338  0.893639
engine-size       -0.699139    -0.714095  0.841496
bore              -0.590440    -0.590850  0.533890
stroke            -0.020055    -0.012934  0.160664
compression-ratio  0.278332     0.221483  0.209361
horsepower        -0.837214    -0.827941  0.759874
peak-rpm          -0.052929    -0.032777 -0.171916
city-mpg           1.000000     0.971999 -0.692273
```

```
highway-mpg          0.971999      1.000000 -0.720090
target              -0.692273     -0.720090  1.000000
```

We need to drop the features which are not linearly related to the target, to apply linear regression

[17]:
```
df = df.drop(["symboling", "normalized-losses","compression-ratio","peak-rpm"],␣
 ↪axis=1)
```

Scaling the features and splliting the dataset

[18]:
```
from sklearn.preprocessing import StandardScaler
# define min max scaler
scaler = StandardScaler()
# transform data
X = scaler.fit_transform(X)

#splitting train and test sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.
 ↪25,random_state=1)
```

MODEL TRAINING: linear regression model is used because the target variable is a continous value and the features are highly correlated with the target (linear relation)

[19]:
```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train,y_train)
```

[19]:
```
LinearRegression()
```

[20]:
```
y_pred = model.predict(X_test)
```

MODEL EVALUATION

[21]:
```
from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred)
print('r2 score: ', r2)
```

```
r2 score:  0.8132203257968371
```

[23]:
```
model.coef_
```

[23]:
```
array([ 193.58894557,   82.74311106, 1114.78833641, -641.20751454,
       1250.21137855,  158.14454003, 1548.34682768, 2227.11704827,
       -972.28136924, -884.57246491,  385.95660533,  849.97430991,
        230.90997148,  307.51759301, -515.63456682])
```

[24]:
```
model.intercept_
```

[24]: 11396.795252466629

[ ]:

swe4012-ml-class-1-ex

April 30, 2024

[17]:
```
import warnings
warnings.filterwarnings('ignore')

import pandas as pd
# loading file
filepath ='Iris_Data.csv'
```

[2]:
```
#import the data
data=pd.read_csv(filepath)
```

[3]:
```
print(data.shape)    # Number of roows and columns
```

```
(150, 5)
```

[4]:
```
data.head()
```

[4]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

[5]:
```
print(data.iloc[:5,:4])
```

```
   sepal_length  sepal_width  petal_length  petal_width
0           5.1          3.5           1.4          0.2
1           4.9          3.0           1.4          0.2
2           4.7          3.2           1.3          0.2
3           4.6          3.1           1.5          0.2
4           5.0          3.6           1.4          0.2
```

[18]:
```
data.mean()
```

[18]:
```
sepal_length    5.843333
sepal_width     3.054000
petal_length    3.758667
petal_width     1.198667
```

```
dtype: float64
```

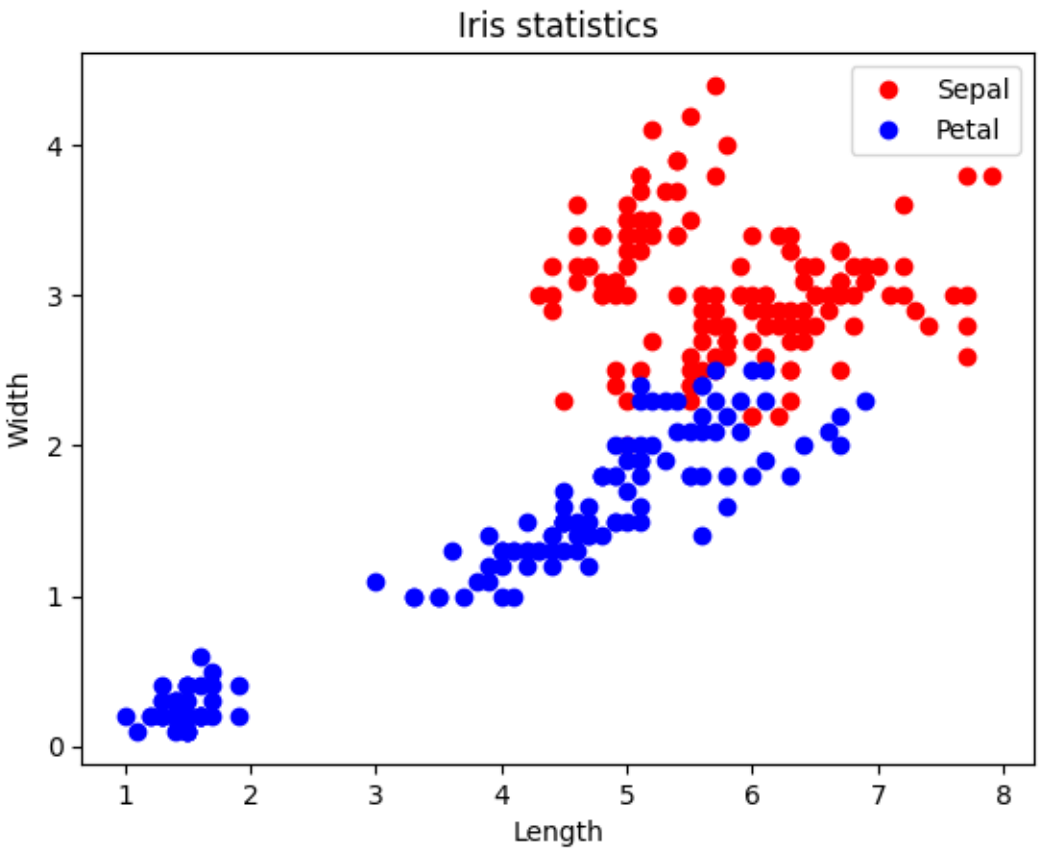
[7]:
```
data.describe()
```

[7]:
```
       sepal_length  sepal_width  petal_length  petal_width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.054000      3.758667     1.198667
std        0.828066     0.433594      1.764420     0.763161
min        4.300000     2.000000      1.000000     0.100000
25%        5.100000     2.800000      1.600000     0.300000
50%        5.800000     3.000000      4.350000     1.300000
75%        6.400000     3.300000      5.100000     1.800000
max        7.900000     4.400000      6.900000     2.500000
```

[24]:
```
import matplotlib.pyplot as plt    # from matplotlib import pyplot as plt
%matplotlib inline
```

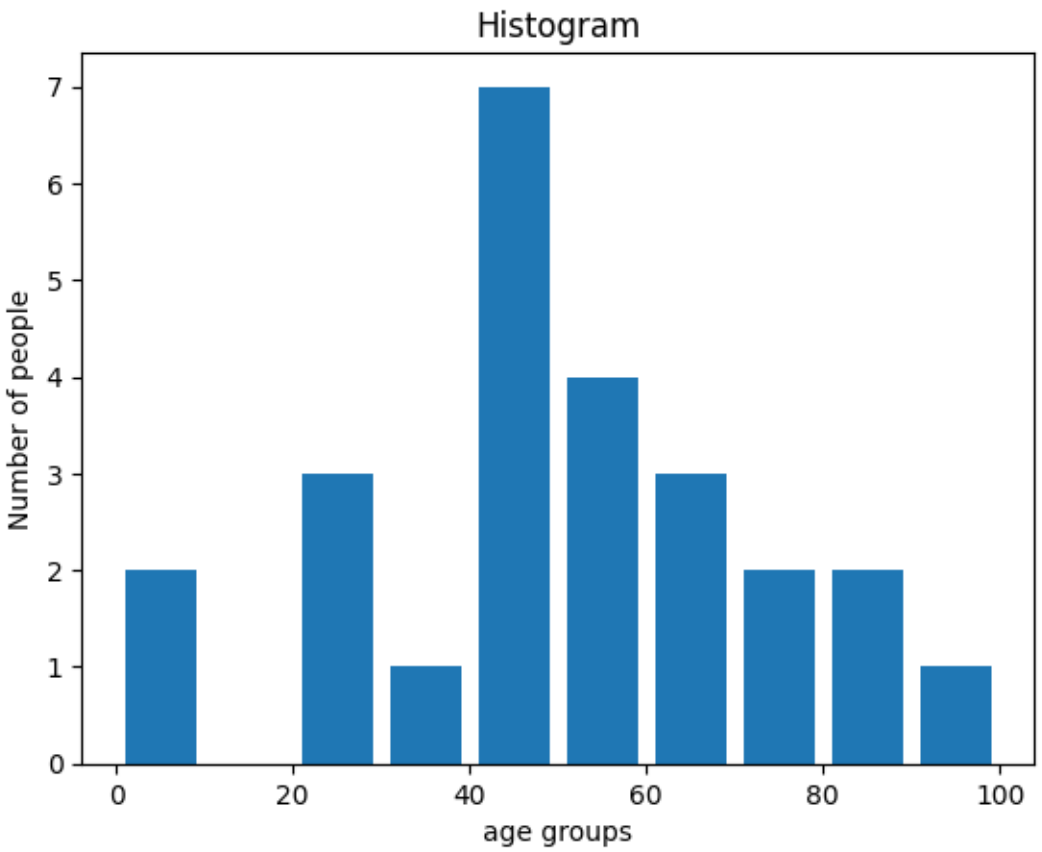
[26]:
```
plt.plot(data.sepal_length, data.
  ↪sepal_width,ls='',marker='o',color='r',label='Sepal')
plt.plot(data.petal_length, data.
  ↪petal_width,ls='',marker='o',color='b',label='Petal')


plt.title("Iris statistics")
plt.xlabel("Length")
plt.ylabel("Width")

plt.legend()

#plt.legend(loc='upper left')
```

[26]:
```
<matplotlib.legend.Legend at 0x7fbaca533c70>
```



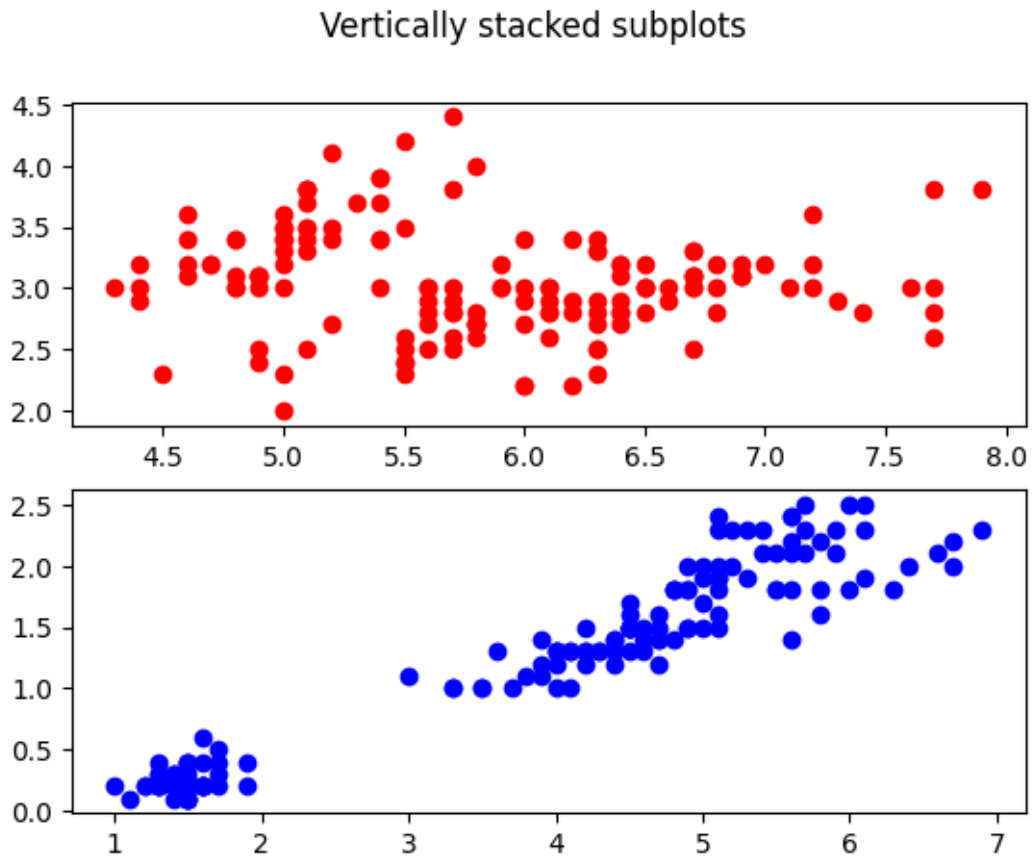
[10]:
```
import matplotlib.pyplot as plt
population_age =␣
 ↪[22,55,62,45,21,22,34,42,42,4,2,102,95,85,55,110,120,70,65,55,111,115,80,75,65,54,44,43,42,
bins = [0,10,20,30,40,50,60,70,80,90,100]
plt.hist(population_age, bins, histtype='bar', rwidth=0.8)   # histype='step'  ␣
 ↪or histtype='stepfilled'
plt.xlabel('age groups')
plt.ylabel('Number of people')
plt.title('Histogram')
#plt.show()
```

[10]: Text(0.5, 1.0, 'Histogram')



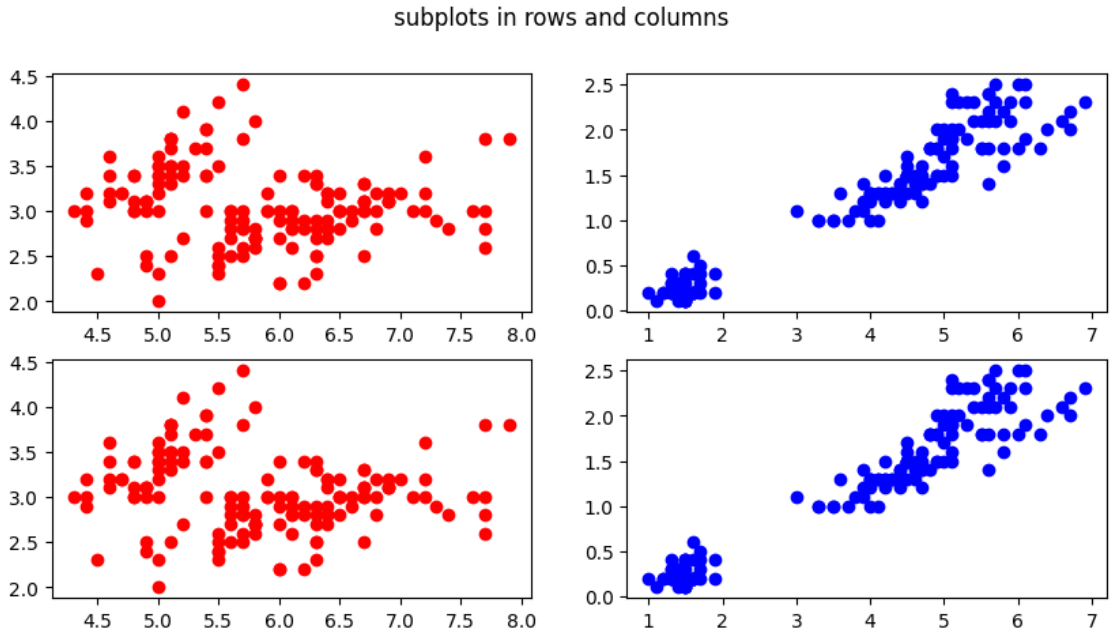
[11]:
```
fig, axs = plt.subplots(2)
fig.suptitle('Vertically stacked subplots')


axs[0].plot(data.sepal_length, data.
 ↪sepal_width,ls='',marker='o',color='r',label='Sepal')
axs[1].plot(data.petal_length, data.
 ↪petal_width,ls='',marker='o',color='b',label='Petal')
```

[11]:
```
[<matplotlib.lines.Line2D at 0x7fbacabd5fd0>]
```



[12]:
```
fig, axs = plt.subplots(2,2,figsize=(10,5))     # horizontal and verical space
fig.suptitle('subplots in rows and columns')


axs[0,0].plot(data.sepal_length, data.
 ↪sepal_width,ls='',marker='o',color='r',label='Sepal')
axs[0,1].plot(data.petal_length, data.
 ↪petal_width,ls='',marker='o',color='b',label='Petal')

axs[1,0].plot(data.sepal_length, data.
 ↪sepal_width,ls='',marker='o',color='r',label='Sepal')
axs[1,1].plot(data.petal_length, data.
 ↪petal_width,ls='',marker='o',color='b',label='Petal')
```

[12]:
```
[<matplotlib.lines.Line2D at 0x7fbacaa506d0>]
```



[13]:
```
X=[1,2,3,4,5]
fact=1
for i in range(5):
    fact*=X[i]
print(fact)
```

120

[14]:
```
X=[1,2,3,4,5,6]
fact=1
for i in X:
    fact*=i
print(fact)
```

720

[16]:
```
x = [i for i in range(1,10,2)]     #list comprehension
print(x)
```

[1, 3, 5, 7, 9]

[ ]:
```

```

naive-bayes

April 30, 2024

Loading the Dataset

[1]:
```
import numpy as np
import pandas as pd
col_names = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety',␣
 ↪'class']
df = pd.read_csv('car.data', header=None, names=col_names)
df.head()
```

[1]:

	buying	maint	doors	persons	lug_boot	safety	class
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc

[2]:
```
# Get a summary of the Dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   buying    1728 non-null   object
 1   maint     1728 non-null   object
 2   doors     1728 non-null   object
 3   persons   1728 non-null   object
 4   lug_boot  1728 non-null   object
 5   safety    1728 non-null   object
 6   class     1728 non-null   object
dtypes: object(7)
memory usage: 94.6+ KB
```

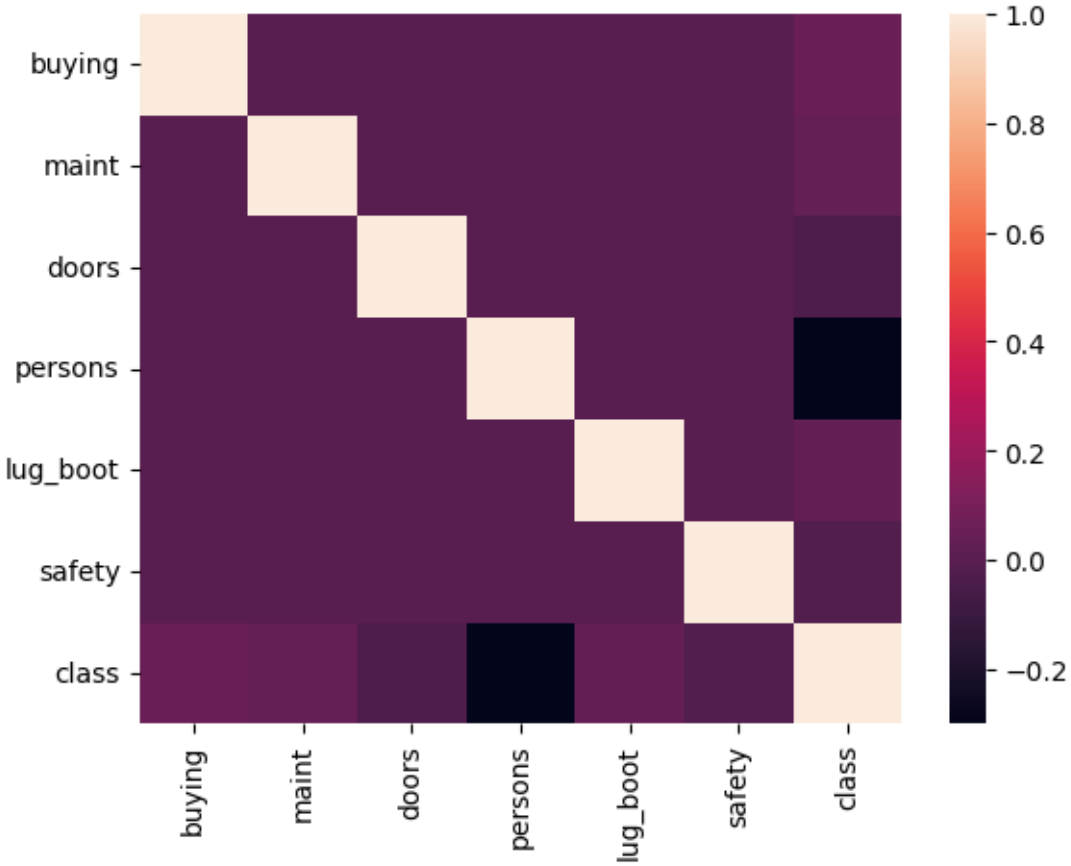
DATA PREPROCESSING: no null values in dataset, featured are encoded

[4]:
```
from sklearn.preprocessing import LabelEncoder
df_encoded = df.apply(LabelEncoder().fit_transform)
df_encoded.head()
```

[4]:

	buying	maint	doors	persons	lug_boot	safety	class
0	3	3	0	0	2	1	2
1	3	3	0	0	2	2	2
2	3	3	0	0	2	0	2
3	3	3	0	0	1	1	2
4	3	3	0	0	1	2	2

[5]:
```
import matplotlib.pyplot as mp
import seaborn as sb

# plotting correlation heatmap
dataplot=sb.heatmap(df_encoded.corr())

# displaying heatmap
mp.show()
```



From the above heatmap, we can see that no 2 independent variables are correlated to each other => Naive bayes can be applied

[29]:
```
#Splitting dataset into train and test sets

X = df_encoded.drop(['class'],axis=1).values
y = df_encoded['class'].values

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)
```

MODEL TRAINING

[30]:
```
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(X_train, y_train)
```

[30]:
```
GaussianNB()
```

[31]:
```
#Predicting the Test set results
y_pred = model.predict(X_test)
```

MODEL EVALUATION using confusion matrix and classification report

[32]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

[32]:
```
array([[ 15,   1,  37,  61],
       [  6,   0,   7,   8],
       [  4,   0, 302,  62],
       [  0,   0,   0,  16]])
```

[33]:
```
import warnings
warnings.filterwarnings('always')
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.60      0.13      0.22       114
           1       0.00      0.00      0.00        21
           2       0.87      0.82      0.85       368
           3       0.11      1.00      0.20        16

    accuracy                           0.64       519
   macro avg       0.40      0.49      0.31       519
weighted avg       0.75      0.64      0.65       519
```

[ ]:

decision-tree

April 30, 2024

Loading the dataset

[1]:
```
import numpy as np
import pandas as pd
data = pd.read_csv("mushroom_csv.csv")
```

[2]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8124 entries, 0 to 8123
Data columns (total 23 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   cap-shape                 8124 non-null   object
 1   cap-surface               8124 non-null   object
 2   cap-color                 8124 non-null   object
 3   bruises%3F                8124 non-null   object
 4   odor                      8124 non-null   object
 5   gill-attachment           8124 non-null   object
 6   gill-spacing              8124 non-null   object
 7   gill-size                 8124 non-null   object
 8   gill-color                8124 non-null   object
 9   stalk-shape               8124 non-null   object
 10  stalk-root                5644 non-null   object
 11  stalk-surface-above-ring  8124 non-null   object
 12  stalk-surface-below-ring  8124 non-null   object
 13  stalk-color-above-ring    8124 non-null   object
 14  stalk-color-below-ring    8124 non-null   object
 15  veil-type                 8124 non-null   object
 16  veil-color                8124 non-null   object
 17  ring-number               8124 non-null   object
 18  ring-type                 8124 non-null   object
 19  spore-print-color         8124 non-null   object
 20  population                8124 non-null   object
 21  habitat                   8124 non-null   object
 22  class                     8124 non-null   object
dtypes: object(23)
memory usage: 1.4+ MB
```

All the features are catagorical and the target variable has 2 value: poisonous or edible

DATA PREPROCESSING

1. Separating features and target cols

[3]:
```
X=data.drop('class',axis=1)
y=data['class']
X.head()
```

[3]:
```
  cap-shape cap-surface cap-color bruises%3F odor gill-attachment  \
0         x           s         n          t    p               f
1         x           s         y          t    a               f
2         b           s         w          t    l               f
3         x           y         w          t    p               f
4         x           s         g          f    n               f

  gill-spacing gill-size gill-color stalk-shape  … stalk-surface-below-ring  \
0            c         n          k           e  …                        s
1            c         b          k           e  …                        s
2            c         b          n           e  …                        s
3            c         n          n           e  …                        s
4            w         b          k           t  …                        s

  stalk-color-above-ring stalk-color-below-ring veil-type veil-color  \
0                      w                      w         p          w
1                      w                      w         p          w
2                      w                      w         p          w
3                      w                      w         p          w
4                      w                      w         p          w

  ring-number ring-type spore-print-color population habitat
0           o         p                 k          s       u
1           o         p                 n          n       g
2           o         p                 n          n       m
3           o         p                 k          s       u
4           o         e                 n          a       g

[5 rows x 22 columns]
```

Handling null values

[4]:
```
data = data.replace('?', np.NaN)
data.isnull().sum()
```

[4]:
```
cap-shape                 0
cap-surface               0
cap-color                 0
bruises%3F                0
```

```
odor                          0
gill-attachment               0
gill-spacing                  0
gill-size                     0
gill-color                    0
stalk-shape                   0
stalk-root                 2480
stalk-surface-above-ring      0
stalk-surface-below-ring      0
stalk-color-above-ring        0
stalk-color-below-ring        0
veil-type                     0
veil-color                    0
ring-number                   0
ring-type                     0
spore-print-color             0
population                    0
habitat                       0
class                         0
dtype: int64
```

[5]:
```
data["stalk-root"].fillna(data["stalk-root"].mode()[0],inplace = True)
```

```
C:\Users\Rajeev Sekar\AppData\Local\Temp\ipykernel_15056\2352973915.py:1:
FutureWarning: A value is trying to be set on a copy of a DataFrame or Series
through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  data["stalk-root"].fillna(data["stalk-root"].mode()[0],inplace = True)
```

[6]:
```
data.isnull().sum()
```

[6]:
```
cap-shape                 0
cap-surface               0
cap-color                 0
bruises%3F                0
odor                      0
gill-attachment           0
gill-spacing              0
gill-size                 0
```

```
gill-color                  0
stalk-shape                 0
stalk-root                  0
stalk-surface-above-ring    0
stalk-surface-below-ring    0
stalk-color-above-ring      0
stalk-color-below-ring      0
veil-type                   0
veil-color                  0
ring-number                 0
ring-type                   0
spore-print-color           0
population                  0
habitat                     0
class                       0
dtype: int64
```

Label encoding catagorical data

[7]:
```
from sklearn.preprocessing import LabelEncoder
Encoder_X = LabelEncoder()
for col in X.columns:
    X[col] = Encoder_X.fit_transform(X[col])
Encoder_y=LabelEncoder()
y = Encoder_y.fit_transform(y)
```

Splitting datset into test and train sets

[8]:
```
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=1)
```

MODEL TRAINING: Decision tree: id3 algorithm (entropy) with tree depth of 6

[9]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree


clf = DecisionTreeClassifier(criterion='entropy', max_depth=6, random_state=0)
clf.fit(X_train, y_train)
```

[9]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=6, random_state=0)
```

Visualising the tree

[10]:
```
plt.figure(figsize=(12,8))
tree.plot_tree(clf.fit(X_train, y_train))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[10], line 1
----> 1 plt.figure(figsize=(12,8))
      2 tree.plot_tree(clf.fit(X_train, y_train))

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

MODEL EVALUATION:

checking training and test set score to ensure there is no overfitting, because DT tend to overfit sometimes

[ ]:
```
print('Training set score: {:.4f}'.format(clf.score(X_train, y_train)))
print('Test set score: {:.4f}'.format(clf.score(X_test, y_test)))
```

=> model has not overfit

Classification report:

[ ]:
```
from sklearn.metrics import classification_report
y_pred=clf.predict(X_test)
print(classification_report(y_test, y_pred))
```

MODEL INTERPRETATION:

=> only 96% of the samples predicted as poisonous actually are poisonous (precision of class 0 = 0.96)

=> only 96% of the samples actually belonging in edible class have been predicted correctly (recall of class 1 = 0.96)

=> In 98% of the cases, the model has predicted if the mushroom is edible or poisonous correctly

[11]:
```
import scikitplot as skplt
import matplotlib.pyplot as plt

skplt.metrics.plot_roc_curve(y_test, y_pred)
plt.show()
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
Cell In[11], line 1
----> 1 import scikitplot as skplt
      2 import matplotlib.pyplot as plt
      4 skplt.metrics.plot_roc_curve(y_test, y_pred)

File D:\Python\Lib\site-packages\scikitplot\__init__.py:2
      1 from __future__ import absolute_import, division, print_function,␣
 ↪unicode_literals
```

```
----> 2 from . import metrics, cluster, decomposition, estimators
      3 __version__ = '0.3.7'
      6 from scikitplot.classifiers import classifier_factory

File D:\Python\Lib\site-packages\scikitplot\metrics.py:27
     24 from sklearn.calibration import calibration_curve
     25 from sklearn.utils import deprecated
---> 27 from scipy import interp
     29 from scikitplot.helpers import binary_ks_curve, validate_labels
     30 from scikitplot.helpers import cumulative_gain_curve

ImportError: cannot import name 'interp' from 'scipy' (D:
 ↪\Python\Lib\site-packages\scipy\__init__.py)
```

[ ]:

lab-preprocess-and-decision-tree

April 30, 2024

IMPORTING NECESSARY LIBS AND LOADING THE DATA

[217]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df=pd.read_csv("monica.csv")
```

[218]:
```
df.head(10)
```

[218]:

	Serial No.	outcome	sex	age	yronset	premi	smstat	diabetes	highbp	hichol	angina	stroke	hosp
0	1	live	f	63	85	n	x	n	y	y	n	n	y
1	2	live	m	59	85	y	x	n	y	n	n	n	y
2	3	live	m	68	85	n	n	n	y	n	n	n	y
3	4	live	m	46	85	n	c	n	n	n	n	n	y
4	5	dead	m	48	85	n	n	y	n	n	y	n	y
5	6	live	f	55	85	n	c	n	y	y	n	n	y
6	7	live	m	56	85	n	x	n	y	n	n	n	y
7	8	live	f	68	85	y	nk	nk	y	nk	y	n	y
8	9	live	m	69	85	n	n	n	y	y	n	n	y
9	10	live	f	64	85	n	x	n	y	n	y	n	y

here outcome is the target variable and other cols are features. Serial No is an irrelavant feature so it can be dropped

[219]:
```
# Drop serial no
df = df.drop(["Serial No."], axis=1)
```

[220]:
```
df.describe()
```

[220]:

	age	yronset
count	6367.000000	6367.000000
mean	59.419978	88.749018
std	7.853923	2.558180
min	35.000000	85.000000
25%	55.000000	87.000000
50%	61.000000	89.000000
75%	66.000000	91.000000
max	69.000000	93.000000

[221]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6367 entries, 0 to 6366
Data columns (total 12 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   outcome   6367 non-null   object
 1   sex       6367 non-null   object
 2   age       6367 non-null   int64
 3   yronset   6367 non-null   int64
 4   premi     6367 non-null   object
 5   smstat    6367 non-null   object
 6   diabetes  6367 non-null   object
 7   highbp    6367 non-null   object
 8   hichol    6367 non-null   object
 9   angina    6367 non-null   object
 10  stroke    6367 non-null   object
 11  hosp      6367 non-null   object
dtypes: int64(2), object(10)
memory usage: 597.0+ KB
```

by using the info and describe methods, we can see that age and yronset features are numerical features and the other features are catagorical

DATA PREPROCESSING:

[222]:
```
#checking null values
df.isnull().sum()
```

[222]:
```
outcome     0
sex         0
age         0
```

```
yronset     0
premi       0
smstat      0
diabetes    0
highbp      0
hichol      0
angina      0
stroke      0
hosp        0
dtype: int64
```

[223]:
```
df.groupby('outcome').size()
```

[223]:
```
outcome
dead    2842
live    3525
dtype: int64
```

=> It is a balanced dataset

Next we analyze the catagorical variables (How many different possible values they have)

[224]:
```
#catagorical variables
categorical = [var for var in df.columns if df[var].dtype == 'O']
categorical
```

[224]:
```
['outcome',
 'sex',
 'premi',
 'smstat',
 'diabetes',
 'highbp',
 'hichol',
 'angina',
 'stroke',
 'hosp']
```

[225]:
```
for col in categorical:
    print(df.groupby(col).size())
    print("\n")
```

```
outcome
dead    2842
live    3525
dtype: int64


sex
```

```
f    1762
m    4605
dtype: int64


premi
n     4122
nk     734
y     1511
dtype: int64


smstat
c     2051
n     1460
nk     918
x     1938
dtype: int64


diabetes
n     4664
nk     885
y      818
dtype: int64


highbp
n     2542
nk     948
y     2877
dtype: int64


hichol
n     3294
nk    1233
y     1840
dtype: int64


angina
n     3473
nk     975
y     1919
dtype: int64
```

```
stroke
n     4881
nk     926
y      560
dtype: int64
```

```
hosp
n    1925
y    4442
dtype: int64
```

Here the value "nk" in teh features (stroke,angina,hicol,highbp,smstat,premi) suggests that the state of those features for that particular patient is "not known", so we can consider those as null values so we remove the rows with null values

[226]:
```
#replace all nk with numpy NaN value
df = df.replace('nk', np.NaN)
```

[227]:
```
df.isnull().sum()
```

[227]:
```
outcome        0
sex            0
age            0
yronset        0
premi        734
smstat       918
diabetes     885
highbp       948
hichol      1233
angina       975
stroke       926
hosp           0
dtype: int64
```

[228]:
```
df=df.dropna()
df.shape
```

[228]:
```
(4889, 12)
```

[229]:
```
df.isnull().sum()
```

[229]:
```
outcome     0
sex         0
age         0
yronset     0
```

```
premi       0
smstat      0
diabetes    0
highbp      0
hichol      0
angina      0
stroke      0
hosp        0
dtype: int64
```

All the null or 'nk' (not known) values have been dropped. It has not been replaced with the mode value because, in a critical medicine related problem like this, we cant assume the values of important features like cholesterol etc, so it has been removed instead of replacing

[230]:
```
for col in categorical:
    print(df.groupby(col).size())
    print("\n")
```

```
outcome
dead    1700
live    3189
dtype: int64


sex
f    1298
m    3591
dtype: int64


premi
n    3596
y    1293
dtype: int64


smstat
c    1822
n    1323
x    1744
dtype: int64


diabetes
n    4208
y     681
dtype: int64
```

```
highbp
n    2336
y    2553
dtype: int64


hichol
n    3150
y    1739
dtype: int64


angina
n    3223
y    1666
dtype: int64


stroke
n    4426
y     463
dtype: int64


hosp
n    1117
y    3772
dtype: int64
```

Now we can see that there are only relevant values for all the features

Encoding the catagorical columns using label encoding

[231]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df[categorical] = df[categorical].apply(le.fit_transform)
```

[232]:
```
df.head(10)
```

[232]:

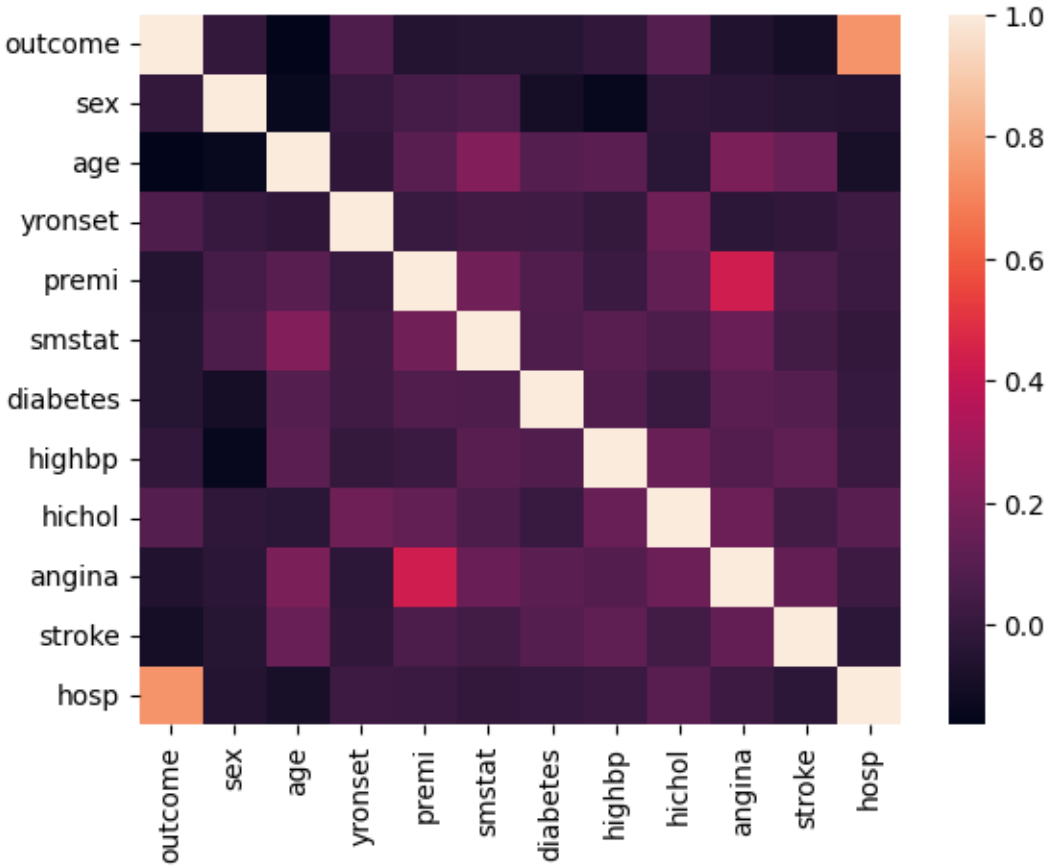
	outcome	sex	age	yronset	premi	smstat	diabetes	highbp	hichol	\
0	1	0	63	85	0	2	0	1	1	
1	1	1	59	85	1	2	0	1	0	
2	1	1	68	85	0	1	0	1	0	
3	1	1	46	85	0	0	0	0	0	
4	0	1	48	85	0	1	1	0	0	

5	1	0	55	85	0	0	0	1	1
6	1	1	56	85	0	2	0	1	0
8	1	1	69	85	0	1	0	1	1
9	1	0	64	85	0	2	0	1	0
11	1	1	55	85	0	1	1	0	0

	angina	stroke	hosp
0	0	0	1
1	0	0	1
2	0	0	1
3	0	0	1
4	1	0	1
5	0	0	1
6	0	0	1
8	0	0	1
9	1	0	1
11	0	0	1

=> outcome 1 is live, 0 is dead

[233]:
```
import seaborn as sn
hm = sn.heatmap(data = df.corr())
```



We can see that there is no multicollinearity between the features. But also there is no correlation between the features and taget varibale hence, we cannot apply Logistic Regression in this case

[234]:
```
#Separating features and target variable
X=df.drop('outcome',axis=1)
y=df['outcome']
```

[235]:
```
#splitting training and testing sets
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=24)
```

MODEL SELECTION: the data consists of mostly catagorical variables (Except age and yronset), Decision Tree(ID-3) works well for catagorical variables, hence we can use that algorithm to build the model. A decision tree of depth = 4 and criterion = entropy (id-3) is built

[236]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree


clf = DecisionTreeClassifier(criterion='entropy', max_depth=4)
clf.fit(X_train, y_train)
```

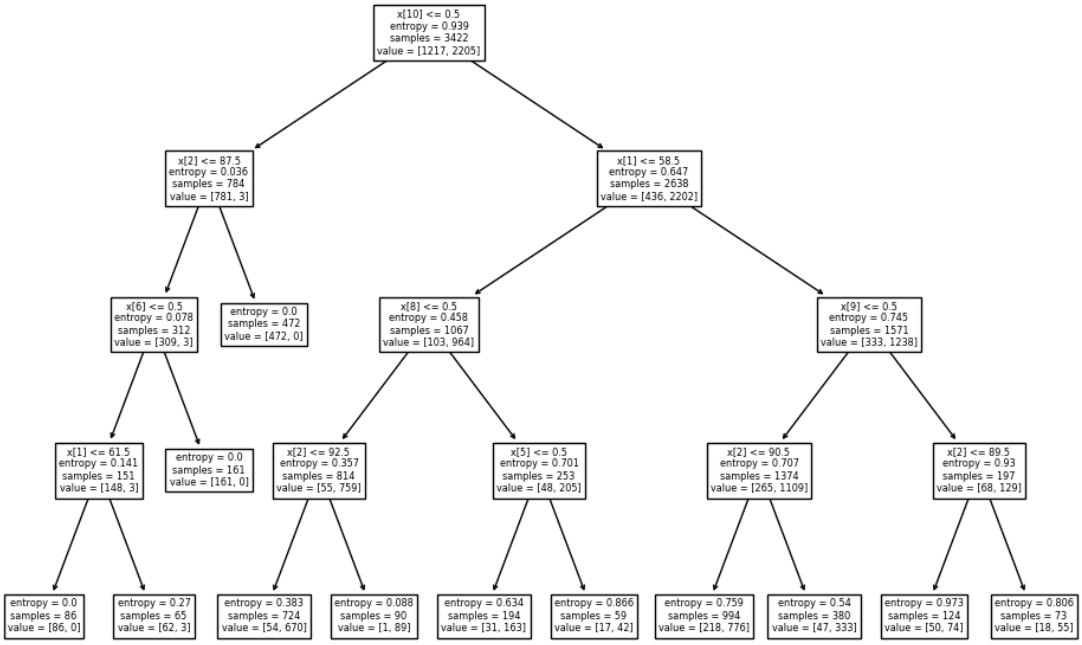
[236]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

Visualising the tree

[237]:
```
plt.figure(figsize=(12,8))
print(tree.plot_tree(clf.fit(X_train, y_train)))
```

```
[Text(0.4, 0.9, 'x[10] <= 0.5\nentropy = 0.939\nsamples = 3422\nvalue = [1217,
2205]'), Text(0.2, 0.7, 'x[2] <= 87.5\nentropy = 0.036\nsamples = 784\nvalue =
[781, 3]'), Text(0.15, 0.5, 'x[6] <= 0.5\nentropy = 0.078\nsamples = 312\nvalue
= [309, 3]'), Text(0.1, 0.3, 'x[1] <= 61.5\nentropy = 0.141\nsamples =
151\nvalue = [148, 3]'), Text(0.05, 0.1, 'entropy = 0.0\nsamples = 86\nvalue =
[86, 0]'), Text(0.15, 0.1, 'entropy = 0.27\nsamples = 65\nvalue = [62, 3]'),
Text(0.2, 0.3, 'entropy = 0.0\nsamples = 161\nvalue = [161, 0]'), Text(0.25,
0.5, 'entropy = 0.0\nsamples = 472\nvalue = [472, 0]'), Text(0.6, 0.7, 'x[1] <=
58.5\nentropy = 0.647\nsamples = 2638\nvalue = [436, 2202]'), Text(0.4, 0.5,
'x[8] <= 0.5\nentropy = 0.458\nsamples = 1067\nvalue = [103, 964]'), Text(0.3,
0.3, 'x[2] <= 92.5\nentropy = 0.357\nsamples = 814\nvalue = [55, 759]'),
Text(0.25, 0.1, 'entropy = 0.383\nsamples = 724\nvalue = [54, 670]'), Text(0.35,
0.1, 'entropy = 0.088\nsamples = 90\nvalue = [1, 89]'), Text(0.5, 0.3, 'x[5] <=
0.5\nentropy = 0.701\nsamples = 253\nvalue = [48, 205]'), Text(0.45, 0.1,
'entropy = 0.634\nsamples = 194\nvalue = [31, 163]'), Text(0.55, 0.1, 'entropy =
```

```
0.866\nsamples = 59\nvalue = [17, 42]'), Text(0.8, 0.5, 'x[9] <= 0.5\nentropy =
0.745\nsamples = 1571\nvalue = [333, 1238]'), Text(0.7, 0.3, 'x[2] <=
90.5\nentropy = 0.707\nsamples = 1374\nvalue = [265, 1109]'), Text(0.65, 0.1,
'entropy = 0.759\nsamples = 994\nvalue = [218, 776]'), Text(0.75, 0.1, 'entropy
= 0.54\nsamples = 380\nvalue = [47, 333]'), Text(0.9, 0.3, 'x[2] <=
89.5\nentropy = 0.93\nsamples = 197\nvalue = [68, 129]'), Text(0.85, 0.1,
'entropy = 0.973\nsamples = 124\nvalue = [50, 74]'), Text(0.95, 0.1, 'entropy =
0.806\nsamples = 73\nvalue = [18, 55]')]
```



MODEL EVALUATION

Sometimes decision tree models may overfit, so we check the testing and training accuracy to check overfitting

[238]:
```
print('Training set score: {:.4f}'.format(clf.score(X_train, y_train)))
print('Test set score: {:.4f}'.format(clf.score(X_test, y_test)))
```

```
Training set score: 0.8717
Test set score: 0.8978
```

=> NO overfitting

[239]:
```
from sklearn.metrics import classification_report
y_pred=clf.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.69      0.82       483
           1       0.87      1.00      0.93       984

    accuracy                           0.90      1467
   macro avg       0.93      0.84      0.87      1467
weighted avg       0.91      0.90      0.89      1467
```

INTERPRETATION:

=> 100% of the people predicted as dead actually are dead (precision of class 0 = 1.00)

=> But only 69% of the people who are actually dead are predicted correctly

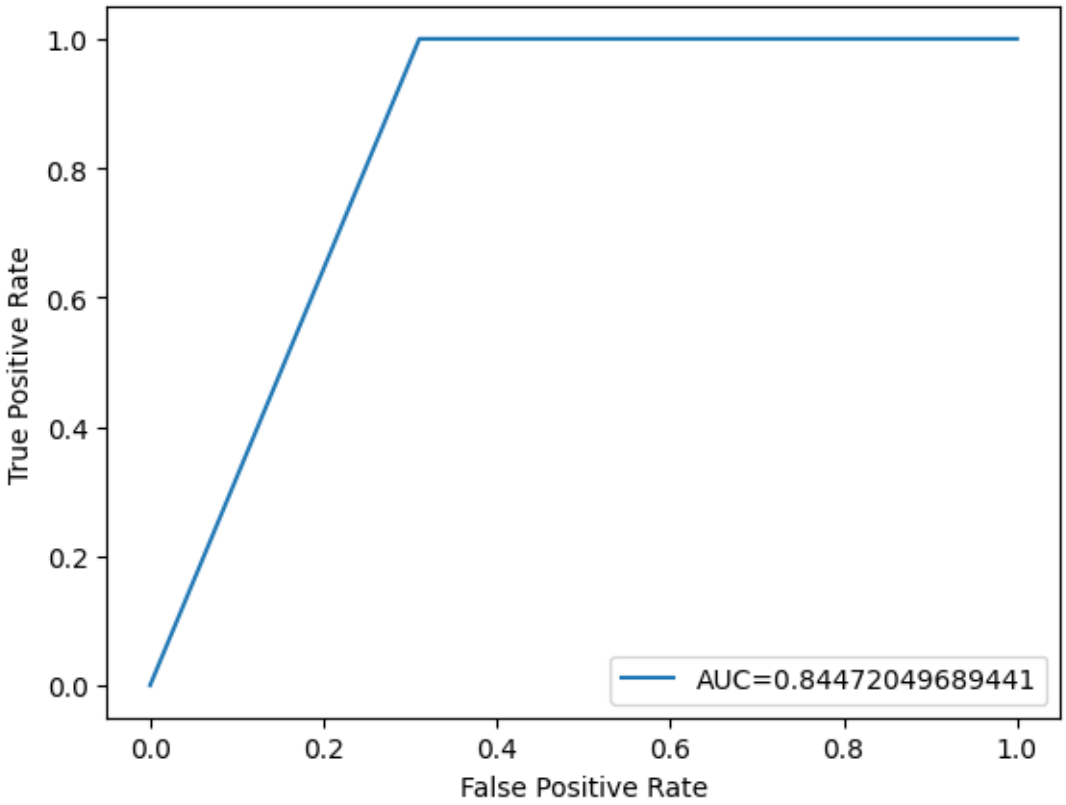
=> This is a good model because, predicting dead wrongly for an alive person is the least ideal case and that is avoided completely

=> But we can not depend only on this model if it gives the outcome as alive for a person (Then need to go for other tests)

=> Overall the model predicts 90% of the cases correctly

Plotting the Reciver operating curve and finding the area under it

[240]:
```
from sklearn import metrics
import matplotlib.pyplot as plt

fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred)
auc = metrics.roc_auc_score(y_test, y_pred)

plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```



=> The model is able to distinguish between alive or dead correctly, in 84.5% of the cases

[ ]:

svm

April 30, 2024

Loading the dataset

[17]:
```
import pandas as pd
cc_apps = pd.read_csv("cc_approvals.data", header = None)
cc_apps.head()
```

[17]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	b	30.83	0.000	u	g	w	v	1.25	t	t	1	f	g	00202	0	+
1	a	58.67	4.460	u	g	q	h	3.04	t	t	6	f	g	00043	560	+
2	a	24.50	0.500	u	g	q	h	1.50	t	f	0	f	g	00280	824	+
3	b	27.83	1.540	u	g	w	v	3.75	t	t	5	t	g	00100	3	+
4	b	20.17	5.625	u	g	w	v	1.71	t	f	0	f	s	00120	0	+

The col names are defined and the summary stats and info of the dataset is displayed

[18]:
```
cc_apps.columns=['Gender', 'Age', 'Debt', 'Married', 'BankCustomer',␣
 ↪'EducationLevel',
           'Ethnicity', 'YearsEmployed', 'PriorDefault', 'Employed',␣
 ↪'CreditScore', 'DriversLicense',
           'Citizen', 'ZipCode', 'Income','ApprovalStatus']

# Print summary statistics
cc_apps_description = cc_apps.describe()
print(cc_apps_description)
print('\n')

# Print DataFrame information
cc_apps_info = cc_apps.info()
print(cc_apps_info)
print('\n')
```

	Debt	YearsEmployed	CreditScore	Income
count	690.000000	690.000000	690.00000	690.000000
mean	4.758725	2.223406	2.40000	1017.385507
std	4.978163	3.346513	4.86294	5210.102598
min	0.000000	0.000000	0.00000	0.000000
25%	1.000000	0.165000	0.00000	0.000000
50%	2.750000	1.000000	0.00000	5.000000
75%	7.207500	2.625000	3.00000	395.500000

```
max     28.000000      28.500000     67.00000  100000.000000
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 690 entries, 0 to 689
Data columns (total 16 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Gender          690 non-null    object
 1   Age             690 non-null    object
 2   Debt            690 non-null    float64
 3   Married         690 non-null    object
 4   BankCustomer    690 non-null    object
 5   EducationLevel  690 non-null    object
 6   Ethnicity       690 non-null    object
 7   YearsEmployed   690 non-null    float64
 8   PriorDefault    690 non-null    object
 9   Employed        690 non-null    object
 10  CreditScore     690 non-null    int64
 11  DriversLicense  690 non-null    object
 12  Citizen         690 non-null    object
 13  ZipCode         690 non-null    object
 14  Income          690 non-null    int64
 15  ApprovalStatus  690 non-null    object
dtypes: float64(2), int64(2), object(12)
memory usage: 86.4+ KB
None
```

DATA PREPROCESSING

Handling null values and 2 cols are dropped as they are irrelavant for this task

[19]:
```
import numpy as np

# Drop the features DriversLicense and ZipCode
cc_apps = cc_apps.drop(["DriversLicense", "ZipCode"], axis=1)

#replace all ? with numpy NaN value
cc_apps = cc_apps.replace('?', np.NaN)

#change the datatype of age col to float
cc_apps['Age'] = cc_apps['Age'].astype(float)

#print no of null values
cc_apps.isnull().sum()
```

[19]:
```
Gender             12
Age                12
Debt                0
Married             6
BankCustomer        6
EducationLevel      9
Ethnicity           9
YearsEmployed       0
PriorDefault        0
Employed            0
CreditScore         0
Citizen             0
Income              0
ApprovalStatus      0
dtype: int64
```

[20]:
```
#catagorical variables
categorical = [var for var in cc_apps.columns if cc_apps[var].dtype == 'O']

#numeric variables
numerical = [var for var in cc_apps.columns if cc_apps[var].dtype != 'O']

#replace null values with mode value
for col in categorical:
    cc_apps[col].fillna(cc_apps[col].mode()[0],inplace = True)

#handle missing values of numeric features by replacing with mean
cc_apps[numerical] = cc_apps[numerical].fillna(cc_apps[numerical].mean())

# Count the number of NaNs in the datasets and print the counts to verify
print(cc_apps.isnull().sum())

print(cc_apps["ApprovalStatus"])
```

```
Gender             0
Age                0
Debt               0
Married            0
BankCustomer       0
EducationLevel     0
Ethnicity          0
YearsEmployed      0
PriorDefault       0
Employed           0
CreditScore        0
Citizen            0
Income             0
```

```
ApprovalStatus    0
dtype: int64
0      +
1      +
2      +
3      +
4      +
      ..
685    -
686    -
687    -
688    -
689    -
Name: ApprovalStatus, Length: 690, dtype: object
```

Separate feature and target values and spliting into train and test sets

[25]:
```
# split data into its X and y components
X = cc_apps.drop('ApprovalStatus',axis = 1)
y = cc_apps['ApprovalStatus']
X=pd.get_dummies(X)
```

[26]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,␣
 ↪random_state=0)
```

MODEL TRAINING: Linear SVM

[27]:
```
from sklearn import svm
clf = svm.SVC(kernel='linear')
clf.fit(X_train, y_train)
```

[27]:
```
SVC(kernel='linear')
```

MODEL EVALUATION

[28]:
```
from sklearn.metrics import confusion_matrix
y_pred = clf.predict(X_test)

confusion_matrix(y_test,y_pred)
```

[28]:
```
array([[ 74,  16],
       [ 17, 100]], dtype=int64)
```

[29]:
```
from sklearn.metrics import classification_report
y_pred=clf.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support
```

```
           +       0.81      0.82      0.82        90
           -       0.86      0.85      0.86       117

    accuracy                           0.84       207
   macro avg       0.84      0.84      0.84       207
weighted avg       0.84      0.84      0.84       207
```

84% of the cases are accurately predicted, onyl 81% of cases predicted as approved are actually approved and only 82% of the cases of the cases which are actually approved are predicted correctly

This shows the points considered as the support vectors

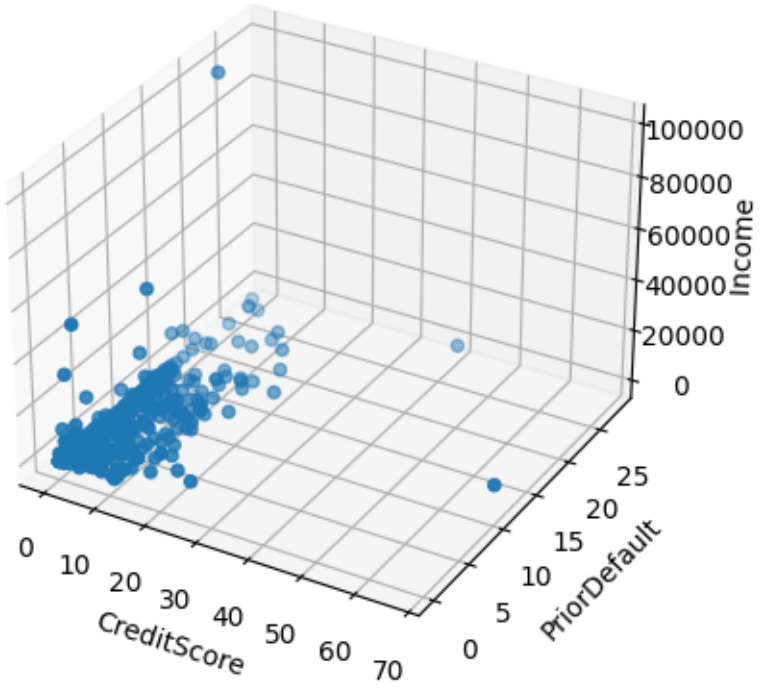
[30]:
```
print(clf.support_vectors_)
```

```
[[36.33   2.125  0.085 …  1.     0.     0.   ]
 [20.25   9.96   0.    …  1.     0.     0.   ]
 [21.33  10.5    3.    …  1.     0.     0.   ]
 …
 [35.25   3.165  3.75  …  1.     0.     0.   ]
 [25.67   2.21   4.    …  1.     0.     0.   ]
 [32.33   7.5    1.585 …  0.     0.     1.   ]]
```

[31]:
```
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import pyplot as plt
```

[35]:
```
ax = plt.axes(projection='3d')
ax.scatter(X['CreditScore'],X['Debt'], X['Income'])
ax.set_xlabel('CreditScore')
ax.set_ylabel('PriorDefault')
ax.set_zlabel('Income ')
```

[35]:
```
Text(0.5, 0, 'Income ')
```


100000
80000
60000
40000
20000
0
Income
0
10
20
30
40
50
60
70
CreditScore
0
5
10
15
20
25
PriorDefault

[ ]:

logreg

April 30, 2024

[15]:
```
#import pandas
import pandas as pd
# load dataset
pima = pd.read_csv("diabetes.csv")
pima.head()
```

[15]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

DATA PREPROCESSING

[16]:
```
#split dataset in features and target variable
feature_cols = ['Pregnancies', 'Glucose', 'BMI',␣
 ↪'Age','Insulin','BloodPressure','DiabetesPedigreeFunction']
X = pima[feature_cols] # Features
y = pima.Outcome # Target variable
```

[17]:
```
matrix = X.corr()
print(matrix)
```

	Pregnancies	Glucose	BMI	Age	Insulin	BloodPressure
Pregnancies	1.000000	0.129459	0.017683	0.544341	-0.073535	0.141282
Glucose	0.129459	1.000000	0.221071	0.263514	0.331357	0.152590
BMI	0.017683	0.221071	1.000000	0.036242	0.197859	0.281805
Age	0.544341	0.263514	0.036242	1.000000	-0.042163	0.239528
Insulin	-0.073535	0.331357	0.197859	-0.042163	1.000000	0.088933
BloodPressure	0.141282	0.152590	0.281805	0.239528	0.088933	1.000000

```
DiabetesPedigreeFunction    -0.033523  0.137337  0.140647  0.033561  0.185071
```

	BloodPressure	DiabetesPedigreeFunction
Pregnancies	0.141282	-0.033523
Glucose	0.152590	0.137337
BMI	0.281805	0.140647
Age	0.239528	0.033561
Insulin	0.088933	0.185071
BloodPressure	1.000000	0.041265
DiabetesPedigreeFunction	0.041265	1.000000

No features are correlated with each other => We can apply logistic regression algorithm

Scaling the features:

[42]:
```
from sklearn.preprocessing import StandardScaler

std_scaler = StandardScaler()

X = std_scaler.fit_transform(X)
```

Splitting train and test sets

[43]:
```
# split X and y into training and testing sets
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
 ↪random_state=16)
```

MODEL TRAINING

[44]:
```
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()

logreg.fit(X_train, y_train)

y_pred = logreg.predict(X_test)
```

MODEL EVALUATION

[45]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.82      0.92      0.87       125
           1       0.81      0.63      0.71        67

    accuracy                           0.82       192
```

```
   macro avg       0.81      0.77      0.79       192
weighted avg       0.82      0.82      0.81       192
```

Interpretation:

=> only 82% of patients predicted as having diabetes actually have diabetes

=> 92% of the patients actually having diabetes are predicted correctly

=> Overall, the model predicts 82% of the cases accurately

Displaying the model parameters:

1. The slope values of all features

[46]:
```
logreg.coef_
```

[46]:
```
array([[ 0.30213836,  1.02049564,  0.70136964,  0.15244513, -0.15129797,
        -0.25676181,  0.29104294]])
```

2. The intercept value

[47]:
```
logreg.intercept_
```

[47]:
```
array([-0.83662093])
```

[ ]:
```

```

crossvalidation

April 30, 2024

[1]:
```
# Cross Validation methods
    #  Hold-out
    #  K-folds
    #  Leave-one-out
# links followed
#https://www.cs.cmu.edu/~schneide/tut5/node42.html#:~:
 ↪text=Leave%2Done%2Dout%20cross%20validation,is%20made%20for%20that%20point.
#https://neptune.ai/blog/cross-validation-in-machine-learning-how-to-do-it-right
```

K-Fold

k-Fold Cross-Validation: The dataset is divided into k folds (in this case, 5 folds). Each fold serves as a test set exactly once, and the rest of the data is used for training. The KFold function from scikit-learn is used to create the fold indices.

[15]:
```
#imports
import pandas as pd
from sklearn.model_selection import cross_val_score, KFold
from sklearn.svm import SVC
from sklearn.datasets import load_iris
# Dataset Loading
file_path = '/home/ex5/Desktop/AdmissionChangedDataset.csv'
df = pd.read_csv(file_path)
df.head()
```

[15]:

	SerialNo	GREScore	TOEFLScore	UniversityRating	SOP	LOR	CGPA	\
0	1	337	118	4	4.5	4.5	9.65	
1	2	324	107	4	4.0	4.5	8.87	
2	3	316	104	3	3.0	3.5	8.00	
3	4	322	110	3	3.5	2.5	8.67	
4	5	314	103	2	2.0	3.0	8.21	

	Research	ChanceofAdmit	PassOrFail
0	1	0.92	0
1	1	0.76	0
2	1	0.72	0
3	1	0.80	0
4	0	0.65	1

[30]:
```
# Initialize a Support Vector Machine (SVM) classifier with a linear kernel
svm_classifier = SVC(kernel='linear')

# Split the dataset into features (X) and target (y)
X = df.drop(columns=["PassOrFail"])
y = df["PassOrFail"]

# Set the number of folds for k-fold cross-validation
num_folds = 5

# Initialize a KFold object for splitting the dataset into k folds
kf = KFold(n_splits=num_folds, shuffle=True, random_state=42)

# Perform k-fold cross-validation using the SVM classifier
cross_val_results = cross_val_score(svm_classifier, X, y, cv=kf)

# Print the cross-validation results
print(f'Cross-Validation Results (Accuracy): {cross_val_results}')
print(f'Mean Accuracy: {cross_val_results.mean()}')
```

```
Cross-Validation Results (Accuracy): [0.91 0.86 0.87 0.87 0.9 ]
Mean Accuracy: 0.882
```

Summary The cross-validation results indicate an average accuracy of 88.2% across five folds. This suggests that the model performs well in predicting whether a student passes or fails admission based on given features.

Interpretation: The model demonstrates strong predictive performance, with an average accuracy of 88.2%. This implies that it can effectively differentiate between students likely to pass or fail admission based on their GRE scores, TOEFL scores, university ratings, statements of purpose, letters of recommendation, CGPA, and research experience.

Justification: The high mean accuracy of 88.2% suggests that the model generalizes well to unseen data, indicating robustness and reliability. This is crucial for admission prediction systems, ensuring fair evaluation of candidates based on diverse academic and personal factors. Such accuracy reinforces confidence in using the model for admission decision-making.

==========

Leave-One-Out Cross-Validation (LOOCV)

the Leave-One-Out Cross-Validation (LOOCV) method is implicitly utilized through the cross_val_score function when the number of folds (cv) is set to the length of the dataset (len(X)). LOOCV essentially creates as many folds as there are data points in the dataset, where each data point serves as a separate test set, and the remaining data points are used for training.

Leave-One-Out Cross-Validation (LOOCV) is applied by setting the cv parameter of the cross_val_score function to None. When cv is set to None, scikit-learn automatically performs LOOCV.

[28]:
```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression

# Separate features (X) and target (y)
X1 = df.drop(columns=["PassOrFail"])
X1 = df.drop(columns=["SerialNo"])
y1 = df["PassOrFail"]

# Initialize Logistic Regression model
model1 = LogisticRegression()

# Perform Leave-One-Out Cross-Validation
cross_val_results = cross_val_score(model1, X1, y1, cv=None)

# Print cross-validation results
print(f'Cross-Validation Results (Accuracy): {cross_val_results}')
print(f'Mean Accuracy: {cross_val_results.mean()}')
```

```
Cross-Validation Results (Accuracy): [1. 1. 1. 1. 1.]
Mean Accuracy: 1.0
```

```
/home/ex5/.local/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/home/ex5/.local/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/home/ex5/.local/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/home/ex5/.local/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/home/ex5/.local/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

Interpretation: The cross-validation results indicate perfect accuracy of 100% across all folds. This suggests that the model predicts whether a student passes or fails admission with absolute certainty based on the given features.

Justification: The perfect mean accuracy of 100% demonstrates the model's i.e. Logistic Regression exceptional performance in accurately classifying students as pass or fail based on their GRE scores, TOEFL scores, university ratings, statements of purpose, letters of recommendation, CGPA, and research experience. This flawless accuracy implies that the model effectively captures the underlying patterns in the data and makes highly accurate predictions without errors. Such remarkable performance underscores the robustness and reliability of the model, making it a valuable tool for admission decision-making.

==========

Hold-Out

In this code for the holdout method, cross-validation is not explicitly applied. The holdout method is a simple technique where the dataset is split into two subsets: a training set and a testing set. The model is trained on the training set and evaluated on the testing set.

[29]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X1, y1, test_size=0.2,␣
 ↪random_state=42)
# Initialize Logistic Regression model
model = LogisticRegression()
# Train the model on the training set
model.fit(X_train, y_train)
# Predict the target variable on the testing set
y_pred = model.predict(X_test)
# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
# Print the accuracy
print(f'Holdout Method Results (Accuracy): {accuracy}')
```

```
Holdout Method Results (Accuracy): 1.0
```

```
/home/ex5/.local/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[ ]:

k-means-cluster-elbow

April 30, 2024

IMPORT PACKAGES

[112]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

LOAD DATASET

[113]:
```
df = pd.read_csv("Live.csv")
df.head()
```

[113]:

	status_id	status_type	status_published	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wows	num_hahas	num_sads	num_angrys	Column1	Column2	Column3	Column4
0	1	video	4/22/2018 6:00	529	512	262	432	92	3	1	1	0	NaN	NaN	NaN	NaN
1	2	photo	4/21/2018 22:45	150	0	0	150	0	0	0	0	0	NaN	NaN	NaN	NaN
2	3	video	4/21/2018 6:17	227	236	57	204	21	1	1	0	0	NaN	NaN	NaN	NaN
3	4	photo	4/21/2018 2:29	111	0	0	111	0	0	0	0	0	NaN	NaN	NaN	NaN
4	5	photo	4/18/2018 3:22	213	0	0	204	9	0	0	0	0	NaN	NaN	NaN	NaN

DATA PREPROCESSING

[114]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7050 entries, 0 to 7049
```

```
Data columns (total 16 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   status_id         7050 non-null   int64
 1   status_type       7050 non-null   object
 2   status_published  7050 non-null   object
 3   num_reactions     7050 non-null   int64
 4   num_comments      7050 non-null   int64
 5   num_shares        7050 non-null   int64
 6   num_likes         7050 non-null   int64
 7   num_loves         7050 non-null   int64
 8   num_wows          7050 non-null   int64
 9   num_hahas         7050 non-null   int64
 10  num_sads          7050 non-null   int64
 11  num_angrys        7050 non-null   int64
 12  Column1           0 non-null      float64
 13  Column2           0 non-null      float64
 14  Column3           0 non-null      float64
 15  Column4           0 non-null      float64
dtypes: float64(4), int64(10), object(2)
memory usage: 881.4+ KB
```

Cols 12 to 15 are of no meaning so we drop it

[115]:
```
df.drop(['Column1', 'Column2', 'Column3', 'Column4'], axis=1, inplace=True)
```

These 2 cols have unique values for most of the data so they will carry no meaning so they are dropped

[116]:
```
df.drop(['status_id', 'status_published'], axis=1, inplace=True)
```

[117]:
```
df.head()
```

[117]:

	status_type	num_reactions	num_comments	num_shares	num_likes	num_loves	\
0	video	529	512	262	432	92	
1	photo	150	0	0	150	0	
2	video	227	236	57	204	21	
3	photo	111	0	0	111	0	
4	photo	213	0	0	204	9	

	num_wows	num_hahas	num_sads	num_angrys
0	3	1	1	0
1	0	0	0	0
2	1	1	0	0
3	0	0	0	0
4	0	0	0	0

[118]:
```
df.describe()
```

[118]:

	num_reactions	num_comments	num_shares	num_likes	num_loves	\
count	7050.000000	7050.000000	7050.000000	7050.000000	7050.000000	
mean	230.117163	224.356028	40.022553	215.043121	12.728652	
std	462.625309	889.636820	131.599965	449.472357	39.972930	
min	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	17.000000	0.000000	0.000000	17.000000	0.000000	
50%	59.500000	4.000000	0.000000	58.000000	0.000000	
75%	219.000000	23.000000	4.000000	184.750000	3.000000	
max	4710.000000	20990.000000	3424.000000	4710.000000	657.000000	

	num_wows	num_hahas	num_sads	num_angrys
count	7050.000000	7050.000000	7050.000000	7050.000000
mean	1.289362	0.696454	0.243688	0.113191
std	8.719650	3.957183	1.597156	0.726812
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000
max	278.000000	157.000000	51.000000	31.000000

We are encoding the catagorical data (Only the status type variale) and then scaling the whole dataset using MinMaxScaling method

[119]:
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

df['status_type'] = le.fit_transform(df['status_type'])

ms = MinMaxScaler()
df = ms.fit_transform(df)

df
```

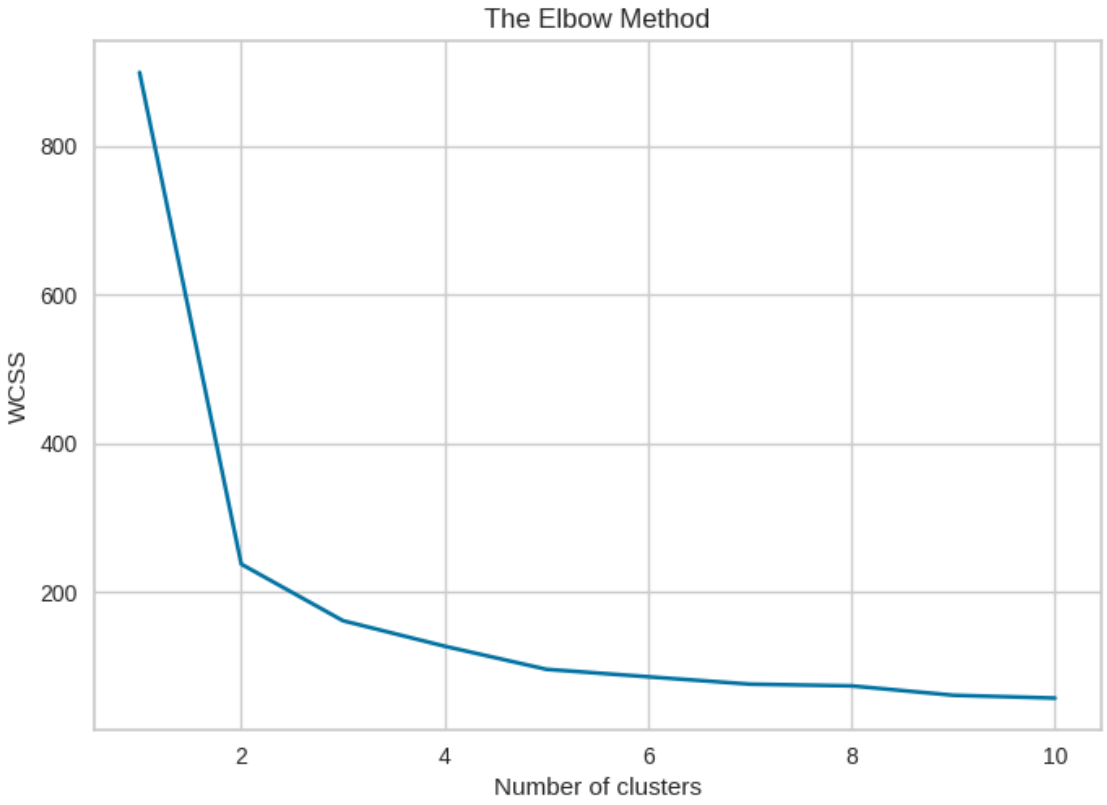
[119]:
```
array([[1.00000000e+00, 1.12314225e-01, 2.43925679e-02, …,
        6.36942675e-03, 1.96078431e-02, 0.00000000e+00],
       [3.33333333e-01, 3.18471338e-02, 0.00000000e+00, …,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [1.00000000e+00, 4.81953291e-02, 1.12434493e-02, …,
        6.36942675e-03, 0.00000000e+00, 0.00000000e+00],
       …,
       [3.33333333e-01, 4.24628450e-04, 0.00000000e+00, …,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [3.33333333e-01, 7.45222930e-02, 5.71700810e-04, …,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [3.33333333e-01, 3.60934183e-03, 0.00000000e+00, …,
```

```
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00]])
```

FINDING THE OPTIMAL K

Elbow method is used to plot the WCSS (Within cluster sum of square) vs the no of clusters which ranges from 1 t 10

[120]:
```
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'random', max_iter = 1000, n_init =␣
 ↪10, random_state = 0)
    kmeans.fit(df)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



Here we can see that the drop in intertia or WCSS value is very flat after k is 3 so we can go for that value

[121]:
```
kmeans = KMeans(n_clusters=3, random_state=0)

kmeans.fit(df)
```

```
/home/ex5/.local/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

[121]:
```
KMeans(n_clusters=3, random_state=0)
```

Prnting the 3 cluster centroids

[122]:
```
kmeans.cluster_centers_
```

[122]:
```
array([[9.63495146e-01, 4.95849772e-02, 2.78226802e-02, 3.05676663e-02,
        4.17542514e-02, 4.92500480e-02, 8.18188168e-03, 1.01094552e-02,
        8.39139539e-03, 7.52896962e-03],
       [3.28742853e-01, 1.99588196e-02, 6.50282622e-04, 5.37894046e-04,
        1.94880247e-02, 1.93982105e-03, 2.03104006e-03, 1.16647149e-03,
        2.84240297e-03, 1.51976868e-03],
       [4.91071429e-01, 3.99261955e-01, 3.03716055e-03, 4.36954133e-03,
        3.97921722e-01, 5.17322606e-03, 9.59232614e-03, 1.23218077e-03,
        9.33706816e-04, 1.92012289e-04]])
```

Viewing the inertia or WCSS when k=3

[123]:
```
kmeans.inertia_
```

[123]:
```
161.60463573139072
```

hierarchial-single-n-complete

April 30, 2024

[16]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.cluster.hierarchy as shc
from scipy.spatial.distance import squareform, pdist
```

Creating a random dataset with 2 features, a real time dataset is not used because, in hierarchial clustering we need to create n clusters for n points in the start (divisive) or in the end (agglomerative), so the plotting will be very difficult for large no of points. so for learning purpose ive used a random data with 10 points and 2 features

[17]:
```
a = np.random.random_sample(size = 10)
b = np.random.random_sample(size = 10)
```

[18]:
```
point = ['P1','P2','P3','P4','P5','P6','P7','P8','P9','P10']
data = pd.DataFrame({'Point':point, 'a':np.round(a,2), 'b':np.round(b,2)})
data = data.set_index('Point')
data
```

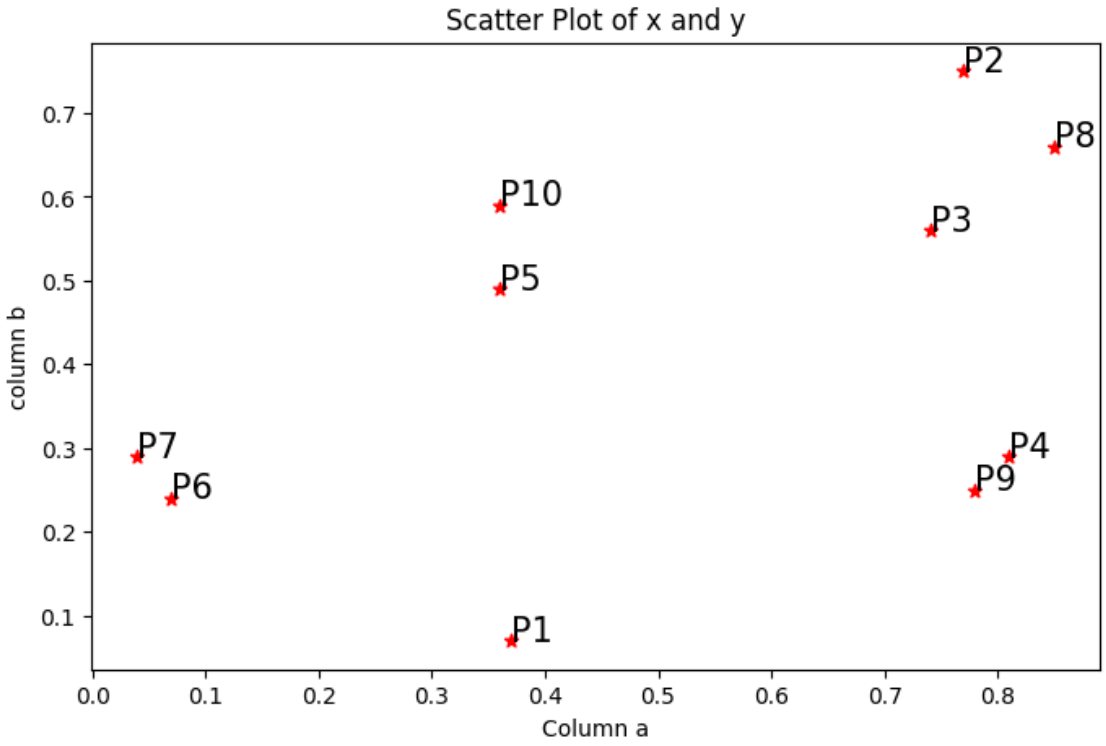
[18]:

Point	a	b
P1	0.37	0.07
P2	0.77	0.75
P3	0.74	0.56
P4	0.81	0.29
P5	0.36	0.49
P6	0.07	0.24
P7	0.04	0.29
P8	0.85	0.66
P9	0.78	0.25
P10	0.36	0.59

Plotting the points

[19]:
```
plt.figure(figsize=(8,5))
plt.scatter(data['a'], data['b'], c='r', marker='*')
plt.xlabel('Column a')
plt.ylabel('column b')
```

```
plt.title('Scatter Plot of x and y')
for j in data.itertuples():
    plt.annotate(j.Index, (j.a, j.b), fontsize=15)
```



Calculating the distance matrix

[20]:
```
dist = pd.DataFrame(squareform(pdist(data[['a', 'b']]), 'euclidean'),␣
 ↪columns=data.index.values, index=data.index.values)
dist
```

[20]:

	P1	P2	P3	P4	P5	P6	P7	\
P1	0.000000	0.788923	0.614003	0.491935	0.420119	0.344819	0.396611	
P2	0.788923	0.000000	0.192354	0.461736	0.485489	0.866083	0.862844	
P3	0.614003	0.192354	0.000000	0.278927	0.386394	0.742496	0.750267	
P4	0.491935	0.461736	0.278927	0.000000	0.492443	0.741687	0.770000	
P5	0.420119	0.485489	0.386394	0.492443	0.000000	0.382884	0.377359	
P6	0.344819	0.866083	0.742496	0.741687	0.382884	0.000000	0.058310	
P7	0.396611	0.862844	0.750267	0.770000	0.377359	0.058310	0.000000	
P8	0.760592	0.120416	0.148661	0.372156	0.518652	0.885889	0.890505	
P9	0.447772	0.500100	0.312570	0.050000	0.483735	0.710070	0.741080	
P10	0.520096	0.440114	0.381182	0.540833	0.100000	0.454533	0.438634	

	P8	P9	P10

```
P1   0.760592  0.447772  0.520096
P2   0.120416  0.500100  0.440114
P3   0.148661  0.312570  0.381182
P4   0.372156  0.050000  0.540833
P5   0.518652  0.483735  0.100000
P6   0.885889  0.710070  0.454533
P7   0.890505  0.741080  0.438634
P8   0.000000  0.415933  0.494975
P9   0.415933  0.000000  0.540370
P10  0.494975  0.540370  0.000000
```

perform single link clustering method, the shortest distance between 2 clusters is considered when one or both the clusters have 2 or more points

[24]:
```
def single_linkage(dist_matrix):
    n = len(dist_matrix)
    while n > 1:
        min_val = float('inf')
        min_index = None
        for i in range(n):
            for j in range(i+1, n):
                if dist_matrix.iloc[i, j] < min_val and dist_matrix.index[i] !=␣
 ↪dist_matrix.columns[j]:
                    min_val = dist_matrix.iloc[i, j]
                    min_index = (i, j)

        if min_val == float('inf'):
            break

        i, j = min_index
        cluster1, cluster2 = dist_matrix.index[i], dist_matrix.columns[j]

        print(f'Merging clusters {cluster1} and {cluster2} with distance␣
 ↪{min_val}')
        new_cluster = f'({cluster1},{cluster2})'
        dist_matrix[new_cluster] = dist_matrix[[cluster1, cluster2]].min(axis=1)
        dist_matrix.loc[new_cluster] = dist_matrix.loc[[cluster1, cluster2]].
 ↪min(axis=0)
        dist_matrix = dist_matrix.drop([cluster1, cluster2], axis=0)
        dist_matrix = dist_matrix.drop([cluster1, cluster2], axis=1)

        n -= 1

        print(dist_matrix)
        print("")

# Perform single linkage clustering
```

```
print("Single Linkage Clustering:")
single_linkage(dist.copy())
```

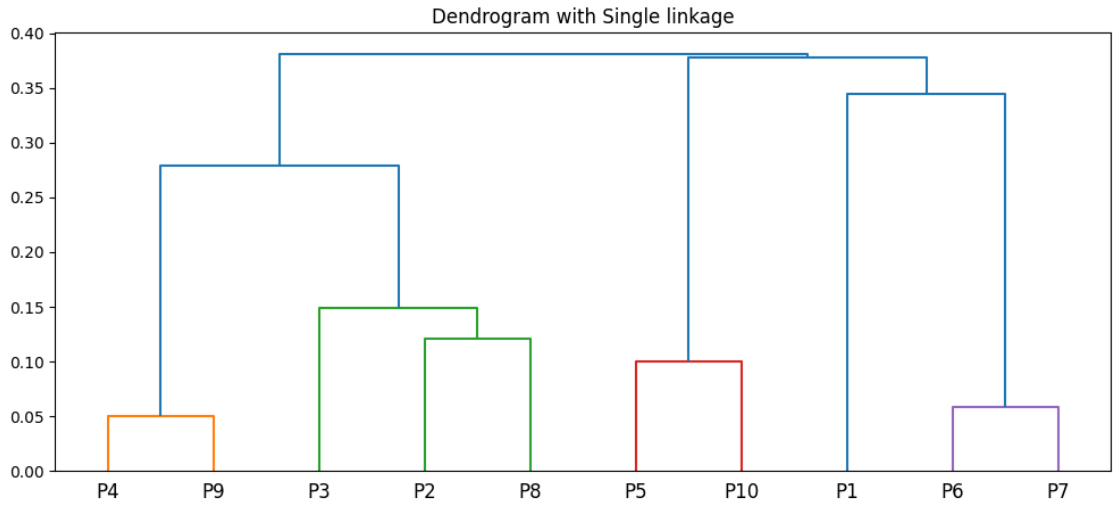
```
Single Linkage Clustering:
Merging clusters P4 and P9 with distance 0.05
              P1        P2        P3        P5        P6        P7        P8  \
P1      0.000000  0.788923  0.614003  0.420119  0.344819  0.396611  0.760592
P2      0.788923  0.000000  0.192354  0.485489  0.866083  0.862844  0.120416
P3      0.614003  0.192354  0.000000  0.386394  0.742496  0.750267  0.148661
P5      0.420119  0.485489  0.386394  0.000000  0.382884  0.377359  0.518652
P6      0.344819  0.866083  0.742496  0.382884  0.000000  0.058310  0.885889
P7      0.396611  0.862844  0.750267  0.377359  0.058310  0.000000  0.890505
P8      0.760592  0.120416  0.148661  0.518652  0.885889  0.890505  0.000000
P10     0.520096  0.440114  0.381182  0.100000  0.454533  0.438634  0.494975
(P4,P9) 0.447772  0.461736  0.278927  0.483735  0.710070  0.741080  0.372156

             P10   (P4,P9)
P1      0.520096  0.447772
P2      0.440114  0.461736
P3      0.381182  0.278927
P5      0.100000  0.483735
P6      0.454533  0.710070
P7      0.438634  0.741080
P8      0.494975  0.372156
P10     0.000000  0.540370
(P4,P9) 0.540370  0.000000

Merging clusters P6 and P7 with distance 0.058309518948452994
              P1        P2        P3        P5        P8       P10   (P4,P9)  \
P1      0.000000  0.788923  0.614003  0.420119  0.760592  0.520096  0.447772
P2      0.788923  0.000000  0.192354  0.485489  0.120416  0.440114  0.461736
P3      0.614003  0.192354  0.000000  0.386394  0.148661  0.381182  0.278927
P5      0.420119  0.485489  0.386394  0.000000  0.518652  0.100000  0.483735
P8      0.760592  0.120416  0.148661  0.518652  0.000000  0.494975  0.372156
P10     0.520096  0.440114  0.381182  0.100000  0.494975  0.000000  0.540370
(P4,P9) 0.447772  0.461736  0.278927  0.483735  0.372156  0.540370  0.000000
(P6,P7) 0.344819  0.862844  0.742496  0.377359  0.885889  0.438634  0.710070

          (P6,P7)
P1      0.344819
P2      0.862844
P3      0.742496
P5      0.377359
P8      0.885889
P10     0.438634
(P4,P9) 0.710070
(P6,P7) 0.000000
```

```
Merging clusters P5 and P10 with distance 0.09999999999999998
                P1        P2        P3        P8   (P4,P9)   (P6,P7)  (P5,P10)
P1        0.000000  0.788923  0.614003  0.760592  0.447772  0.344819  0.420119
P2        0.788923  0.000000  0.192354  0.120416  0.461736  0.862844  0.440114
P3        0.614003  0.192354  0.000000  0.148661  0.278927  0.742496  0.381182
P8        0.760592  0.120416  0.148661  0.000000  0.372156  0.885889  0.494975
(P4,P9)   0.447772  0.461736  0.278927  0.372156  0.000000  0.710070  0.483735
(P6,P7)   0.344819  0.862844  0.742496  0.885889  0.710070  0.000000  0.377359
(P5,P10)  0.420119  0.440114  0.381182  0.494975  0.483735  0.377359  0.000000

Merging clusters P2 and P8 with distance 0.1204159457879229
                P1        P3   (P4,P9)   (P6,P7)  (P5,P10)   (P2,P8)
P1        0.000000  0.614003  0.447772  0.344819  0.420119  0.760592
P3        0.614003  0.000000  0.278927  0.742496  0.381182  0.148661
(P4,P9)   0.447772  0.278927  0.000000  0.710070  0.483735  0.372156
(P6,P7)   0.344819  0.742496  0.710070  0.000000  0.377359  0.862844
(P5,P10)  0.420119  0.381182  0.483735  0.377359  0.000000  0.440114
(P2,P8)   0.760592  0.148661  0.372156  0.862844  0.440114  0.000000

Merging clusters P3 and (P2,P8) with distance 0.14866068747318503
                    P1   (P4,P9)   (P6,P7)  (P5,P10)  (P3,(P2,P8))
P1            0.000000  0.447772  0.344819  0.420119      0.614003
(P4,P9)       0.447772  0.000000  0.710070  0.483735      0.278927
(P6,P7)       0.344819  0.710070  0.000000  0.377359      0.742496
(P5,P10)      0.420119  0.483735  0.377359  0.000000      0.381182
(P3,(P2,P8))  0.614003  0.278927  0.742496  0.381182      0.000000

Merging clusters (P4,P9) and (P3,(P2,P8)) with distance 0.2789265136196271
                             P1   (P6,P7)  (P5,P10)  ((P4,P9),(P3,(P2,P8)))
P1                     0.000000  0.344819  0.420119                0.447772
(P6,P7)                0.344819  0.000000  0.377359                0.710070
(P5,P10)               0.420119  0.377359  0.000000                0.381182
((P4,P9),(P3,(P2,P8)))  0.447772  0.710070  0.381182                0.000000

Merging clusters P1 and (P6,P7) with distance 0.34481879299133333
                       (P5,P10)  ((P4,P9),(P3,(P2,P8)))  (P1,(P6,P7))
(P5,P10)               0.000000               0.381182      0.377359
((P4,P9),(P3,(P2,P8)))  0.381182               0.000000      0.447772
(P1,(P6,P7))           0.377359               0.447772      0.000000

Merging clusters (P5,P10) and (P1,(P6,P7)) with distance 0.37735924528226417
                        ((P4,P9),(P3,(P2,P8)))  ((P5,P10),(P1,(P6,P7)))
((P4,P9),(P3,(P2,P8)))                0.000000                0.381182
((P5,P10),(P1,(P6,P7)))               0.381182                0.000000

Merging clusters ((P4,P9),(P3,(P2,P8))) and ((P5,P10),(P1,(P6,P7))) with
distance 0.38118237105091834
```

```
(((P4,P9),(P3,(P2,P8))),((P5,P10),(P1,(P6,P7))))
(((P4,P9),(P3,(P2,P8))),((P5,P10),(P1,(P6,P7))))
0.0
```

Visualising the dendogram for single link clustering

[28]:
```
plt.figure(figsize=(12,5))
plt.title("Dendrogram with Single linkage")
dend = shc.dendrogram(shc.linkage(data[['a', 'b']], method='single'),␣
↪labels=data.index)
```



[ ]:
```
perform complete link clustering method, the longest distance between 2␣
↪clusters is considered when one or both the clusters have 2 or more points
```

[27]:
```
def complete_linkage(dist_matrix):
    n = len(dist_matrix)
    while n > 1:
        min_val = float('inf')
        min_index = None
        for i in range(n):
            for j in range(i+1, n):
                if dist_matrix.iloc[i, j] < min_val and dist_matrix.index[i] !=␣
↪dist_matrix.columns[j]:
                    min_val = dist_matrix.iloc[i, j]
                    min_index = (i, j)

        if min_val == float('inf'):
            break
```

```
        i, j = min_index
        cluster1, cluster2 = dist_matrix.index[i], dist_matrix.columns[j]

        print(f'Merging clusters {cluster1} and {cluster2} with distance␣
 ↪{min_val}')
        new_cluster = f'({cluster1},{cluster2})'
        dist_matrix[new_cluster] = dist_matrix[[cluster1, cluster2]].max(axis=1)
        dist_matrix.loc[new_cluster] = dist_matrix.loc[[cluster1, cluster2]].
 ↪max(axis=0)
        dist_matrix = dist_matrix.drop([cluster1, cluster2], axis=0)
        dist_matrix = dist_matrix.drop([cluster1, cluster2], axis=1)

        n -= 1

        print(dist_matrix)
        print("")

# Perform complete linkage clustering
print("Complete Linkage Clustering:")
complete_linkage(dist.copy())
```

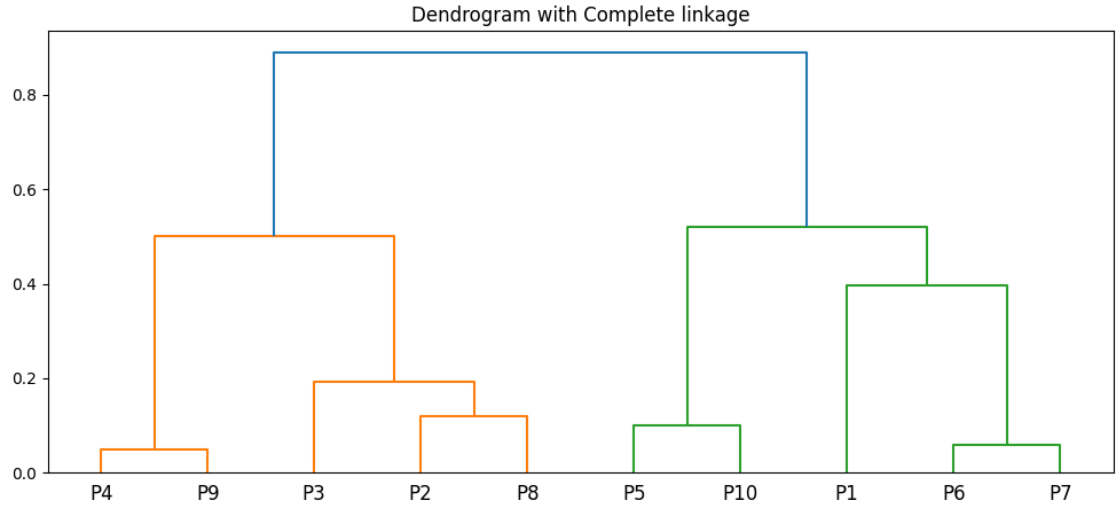
```
Complete Linkage Clustering:
Merging clusters P4 and P9 with distance 0.05
               P1        P2        P3        P5        P6        P7        P8  \
P1       0.000000  0.788923  0.614003  0.420119  0.344819  0.396611  0.760592
P2       0.788923  0.000000  0.192354  0.485489  0.866083  0.862844  0.120416
P3       0.614003  0.192354  0.000000  0.386394  0.742496  0.750267  0.148661
P5       0.420119  0.485489  0.386394  0.000000  0.382884  0.377359  0.518652
P6       0.344819  0.866083  0.742496  0.382884  0.000000  0.058310  0.885889
P7       0.396611  0.862844  0.750267  0.377359  0.058310  0.000000  0.890505
P8       0.760592  0.120416  0.148661  0.518652  0.885889  0.890505  0.000000
P10      0.520096  0.440114  0.381182  0.100000  0.454533  0.438634  0.494975
(P4,P9)  0.491935  0.500100  0.312570  0.492443  0.741687  0.770000  0.415933

              P10   (P4,P9)
P1       0.520096  0.491935
P2       0.440114  0.500100
P3       0.381182  0.312570
P5       0.100000  0.492443
P6       0.454533  0.741687
P7       0.438634  0.770000
P8       0.494975  0.415933
P10      0.000000  0.540833
(P4,P9)  0.540833  0.050000

Merging clusters P6 and P7 with distance 0.058309518948452994
               P1        P2        P3        P5        P8       P10   (P4,P9)  \
```

```
P1       0.000000  0.788923  0.614003  0.420119  0.760592  0.520096  0.491935
P2       0.788923  0.000000  0.192354  0.485489  0.120416  0.440114  0.500100
P3       0.614003  0.192354  0.000000  0.386394  0.148661  0.381182  0.312570
P5       0.420119  0.485489  0.386394  0.000000  0.518652  0.100000  0.492443
P8       0.760592  0.120416  0.148661  0.518652  0.000000  0.494975  0.415933
P10      0.520096  0.440114  0.381182  0.100000  0.494975  0.000000  0.540833
(P4,P9)  0.491935  0.500100  0.312570  0.492443  0.415933  0.540833  0.050000
(P6,P7)  0.396611  0.866083  0.750267  0.382884  0.890505  0.454533  0.770000

          (P6,P7)
P1       0.396611
P2       0.866083
P3       0.750267
P5       0.382884
P8       0.890505
P10      0.454533
(P4,P9)  0.770000
(P6,P7)  0.058310

Merging clusters P5 and P10 with distance 0.09999999999999998
                P1        P2        P3        P8   (P4,P9)   (P6,P7)  (P5,P10)
P1        0.000000  0.788923  0.614003  0.760592  0.491935  0.396611  0.520096
P2        0.788923  0.000000  0.192354  0.120416  0.500100  0.866083  0.485489
P3        0.614003  0.192354  0.000000  0.148661  0.312570  0.750267  0.386394
P8        0.760592  0.120416  0.148661  0.000000  0.415933  0.890505  0.518652
(P4,P9)   0.491935  0.500100  0.312570  0.415933  0.050000  0.770000  0.540833
(P6,P7)   0.396611  0.866083  0.750267  0.890505  0.770000  0.058310  0.454533
(P5,P10)  0.520096  0.485489  0.386394  0.518652  0.540833  0.454533  0.100000

Merging clusters P2 and P8 with distance 0.1204159457879229
                P1        P3   (P4,P9)   (P6,P7)  (P5,P10)   (P2,P8)
P1        0.000000  0.614003  0.491935  0.396611  0.520096  0.788923
P3        0.614003  0.000000  0.312570  0.750267  0.386394  0.192354
(P4,P9)   0.491935  0.312570  0.050000  0.770000  0.540833  0.500100
(P6,P7)   0.396611  0.750267  0.770000  0.058310  0.454533  0.890505
(P5,P10)  0.520096  0.386394  0.540833  0.454533  0.100000  0.518652
(P2,P8)   0.788923  0.192354  0.500100  0.890505  0.518652  0.120416

Merging clusters P3 and (P2,P8) with distance 0.1923538406167134
                    P1   (P4,P9)   (P6,P7)  (P5,P10)  (P3,(P2,P8))
P1            0.000000  0.491935  0.396611  0.520096      0.788923
(P4,P9)       0.491935  0.050000  0.770000  0.540833      0.500100
(P6,P7)       0.396611  0.770000  0.058310  0.454533      0.890505
(P5,P10)      0.520096  0.540833  0.454533  0.100000      0.518652
(P3,(P2,P8))  0.788923  0.500100  0.890505  0.518652      0.192354

Merging clusters P1 and (P6,P7) with distance 0.3966106403010388
               (P4,P9)  (P5,P10)  (P3,(P2,P8))  (P1,(P6,P7))
```

```
(P4,P9)        0.050000  0.540833      0.500100      0.770000
(P5,P10)       0.540833  0.100000      0.518652      0.520096
(P3,(P2,P8))   0.500100  0.518652      0.192354      0.890505
(P1,(P6,P7))   0.770000  0.520096      0.890505      0.396611

Merging clusters (P4,P9) and (P3,(P2,P8)) with distance 0.5000999900019995
                         (P5,P10)  (P1,(P6,P7))  ((P4,P9),(P3,(P2,P8)))
(P5,P10)                 0.100000      0.520096                0.540833
(P1,(P6,P7))             0.520096      0.396611                0.890505
((P4,P9),(P3,(P2,P8)))   0.540833      0.890505                0.500100

Merging clusters (P5,P10) and (P1,(P6,P7)) with distance 0.5200961449578337
                           ((P4,P9),(P3,(P2,P8)))  ((P5,P10),(P1,(P6,P7)))
((P4,P9),(P3,(P2,P8)))                   0.500100                 0.890505
((P5,P10),(P1,(P6,P7)))                  0.890505                 0.520096

Merging clusters ((P4,P9),(P3,(P2,P8))) and ((P5,P10),(P1,(P6,P7))) with
distance 0.8905054744357274
(((P4,P9),(P3,(P2,P8))),((P5,P10),(P1,(P6,P7))))
(((P4,P9),(P3,(P2,P8))),((P5,P10),(P1,(P6,P7))))
0.890505
```

Dendogram for complete link clustering

[30]:
```
plt.figure(figsize=(12,5))
plt.title("Dendrogram with Complete linkage")
dend = shc.dendrogram(shc.linkage(data[['a', 'b']], method='complete'),␣
  ↪labels=data.index)
```



[ ]:

In [108…

```
# PCA is a way of converting the higher dimesnion data into a
# lower dimension data by ensuring that it provides similar
# or maximum information
```

In [109…

```
# steps involved in PCA:

# 1) Standardize the datapoints: brings all the data to a same scale.

# 2) Compute the covariance matrix:
#     i) Variance: it is the spread of the data from the mean or the average point.
#     ii) Covariance matrix: it calculates the measure of how two variables vary each other.

# 3) Compute the eigen values and the eigen vectors
#     i) Eigen value ------ magnitude
#     ii) Eigen vectgor -------- directional_link

# 4) sort the eigen values -------- if the eigen values is high, information spread is high

# 5) select the no of pc's ------- scree plot

# 6) interpret the new data
```

In [3]:

```
# Importing libararies

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

In [4]:

```
# Loading the dataset

from sklearn import datasets
from sklearn.datasets import load_digits
digits = load_digits()
```

In [5]:

```
#
digits
```

Out[5]:

```
{'data': array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ..., 10.,  0.,  0.],
       [ 0.,  0.,  0., ..., 16.,  9.,  0.],
       ...,
       [ 0.,  0.,  1., ...,  6.,  0.,  0.],
       [ 0.,  0.,  2., ..., 12.,  0.,  0.],
       [ 0.,  0., 10., ..., 12.,  1.,  0.]]),
 'target': array([0, 1, 2, ..., 8, 9, 8]),
 'frame': None,
 'feature_names': ['pixel_0_0',
  'pixel_0_1',
  'pixel_0_2',
  'pixel_0_3',
  'pixel_0_4',
  'pixel_0_5',
  'pixel_0_6',
  'pixel_0_7',
  'pixel_1_0',
  'pixel_1_1',
  'pixel_1_2',
  'pixel_1_3',
  'pixel_1_4',
  'pixel_1_5',
  'pixel_1_6',
  'pixel_1_7',
  'pixel_2_0',
  'pixel_2_1',
  'pixel_2_2',
  'pixel_2_3',
  'pixel_2_4',
  'pixel_2_5',
  'pixel_2_6',
  'pixel_2_7',
  'pixel_3_0',
  'pixel_3_1',
  'pixel_3_2',
  'pixel_3_3',
  'pixel_3_4',
  'pixel_3_5',
  'pixel_3_6',
  'pixel_3_7',
  'pixel_4_0',
  'pixel_4_1',
  'pixel_4_2',
```

```
 'pixel_4_3',
 'pixel_4_4',
 'pixel_4_5',
 'pixel_4_6',
 'pixel_4_7',
 'pixel_5_0',
 'pixel_5_1',
 'pixel_5_2',
 'pixel_5_3',
 'pixel_5_4',
 'pixel_5_5',
 'pixel_5_6',
 'pixel_5_7',
 'pixel_6_0',
 'pixel_6_1',
 'pixel_6_2',
 'pixel_6_3',
 'pixel_6_4',
 'pixel_6_5',
 'pixel_6_6',
 'pixel_6_7',
 'pixel_7_0',
 'pixel_7_1',
 'pixel_7_2',
 'pixel_7_3',
 'pixel_7_4',
 'pixel_7_5',
 'pixel_7_6',
 'pixel_7_7'],
'target_names': array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
'images': array([[[ 0.,  0.,  5., ...,  1.,  0.,  0.],
        [ 0.,  0., 13., ..., 15.,  5.,  0.],
        [ 0.,  3., 15., ..., 11.,  8.,  0.],
        ...,
        [ 0.,  4., 11., ..., 12.,  7.,  0.],
        [ 0.,  2., 14., ..., 12.,  0.,  0.],
        [ 0.,  0.,  6., ...,  0.,  0.,  0.]],

       [[ 0.,  0.,  0., ...,  5.,  0.,  0.],
        [ 0.,  0.,  0., ...,  9.,  0.,  0.],
        [ 0.,  0.,  3., ...,  6.,  0.,  0.],
        ...,
        [ 0.,  0.,  1., ...,  6.,  0.,  0.],
        [ 0.,  0.,  1., ...,  6.,  0.,  0.],
```

```
        [ 0.,  0.,  0., ..., 10.,  0.,  0.]],

       [[ 0.,  0.,  0., ..., 12.,  0.,  0.],
        [ 0.,  0.,  3., ..., 14.,  0.,  0.],
        [ 0.,  0.,  8., ..., 16.,  0.,  0.],
        ...,
        [ 0.,  9., 16., ...,  0.,  0.,  0.],
        [ 0.,  3., 13., ..., 11.,  5.,  0.],
        [ 0.,  0.,  0., ..., 16.,  9.,  0.]],

       ...,

       [[ 0.,  0.,  1., ...,  1.,  0.,  0.],
        [ 0.,  0., 13., ...,  2.,  1.,  0.],
        [ 0.,  0., 16., ..., 16.,  5.,  0.],
        ...,
        [ 0.,  0., 16., ..., 15.,  0.,  0.],
        [ 0.,  0., 15., ..., 16.,  0.,  0.],
        [ 0.,  0.,  2., ...,  6.,  0.,  0.]],

       [[ 0.,  0.,  2., ...,  0.,  0.,  0.],
        [ 0.,  0., 14., ..., 15.,  1.,  0.],
        [ 0.,  4., 16., ..., 16.,  7.,  0.],
        ...,
        [ 0.,  0.,  0., ..., 16.,  2.,  0.],
        [ 0.,  0.,  4., ..., 16.,  2.,  0.],
        [ 0.,  0.,  5., ..., 12.,  0.,  0.]],

       [[ 0.,  0., 10., ...,  1.,  0.,  0.],
        [ 0.,  2., 16., ...,  1.,  0.,  0.],
        [ 0.,  0., 15., ..., 15.,  0.,  0.],
        ...,
        [ 0.,  4., 16., ..., 16.,  6.,  0.],
        [ 0.,  8., 16., ..., 16.,  8.,  0.],
        [ 0.,  1.,  8., ..., 12.,  1.,  0.]]]),
 'DESCR': ".. _digits_dataset:\n\nOptical recognition of handwritten digits dataset\n--------------------------------------------
-------\n\n**Data Set Characteristics:**\n\n    :Number of Instances: 1797\n    :Number of Attributes: 64\n    :Attribute Inform
ation: 8x8 image of integer pixels in the range 0..16.\n    :Missing Attribute Values: None\n    :Creator: E. Alpaydin (alpaydin
'@' boun.edu.tr)\n    :Date: July; 1998\n\nThis is a copy of the test set of the UCI ML hand-written digits datasets\nhttps://ar
chive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits\n\nThe data set contains images of hand-written digits:
10 classes where\neach class refers to a digit.\n\nPreprocessing programs made available by NIST were used to extract\nnormalize
d bitmaps of handwritten digits from a preprinted form. From a\ntotal of 43 people, 30 contributed to the training set and diffe
rent 13\nto the test set. 32x32 bitmaps are divided into nonoverlapping blocks of\n4x4 and the number of on pixels are counted i
n each block. This generates\nan input matrix of 8x8 where each element is an integer in the range\n0..16. This reduces dimensio
```

```
nality and gives invariance to small\ndistortions.\n\nFor info on NIST preprocessing routines, see M. D. Garris, J. L. Blue,
G.\nT. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C.\nL. Wilson, NIST Form-Based Handprint Recognition Sy
stem, NISTIR 5469,\n1994.\n\n.. topic:: References\n\n  - C. Kaynak (1995) Methods of Combining Multiple Classifiers and Their\n
Applications to Handwritten Digit Recognition, MSc Thesis, Institute of\n    Graduate Studies in Science and Engineering, Bogazi
ci University.\n  - E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.\n  - Ken Tang and Ponnuthurai N. Suganthan
and Xi Yao and A. Kai Qin.\n    Linear dimensionalityreduction using relevance weighted LDA. School of\n    Electrical and Elect
ronic Engineering Nanyang Technological University.\n    2005.\n  - Claudio Gentile. A New Approximate Maximal Margin Classifica
tion\n    Algorithm. NIPS. 2000.\n"}
```

In [6]:
```
# The images of the handwritten digits are contained in a digits.images array.
# Each element of this array is an image that is represented by an 8x8 matrix of
# numerical values that correspond to a grayscale from white, with a value of 0,
# to black, with the value 15.
digits.data
```

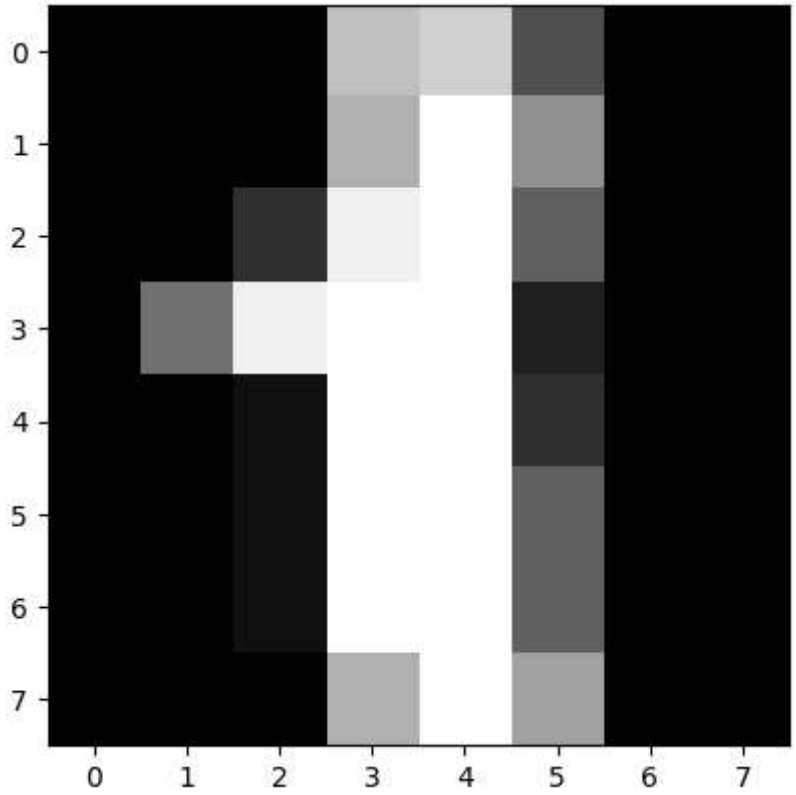
Out[6]:
```
array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ..., 10.,  0.,  0.],
       [ 0.,  0.,  0., ..., 16.,  9.,  0.],
       ...,
       [ 0.,  0.,  1., ...,  6.,  0.,  0.],
       [ 0.,  0.,  2., ..., 12.,  0.,  0.],
       [ 0.,  0., 10., ..., 12.,  1.,  0.]])
```

In [7]:
```
digits.target
```

Out[7]:
```
array([0, 1, 2, ..., 8, 9, 8])
```

In [8]:
```
# to display the image in 2d in grayscale , here we are displaying 1
plt.imshow(digits.images[1],cmap=plt.cm.gray)
```

Out[8]:
```
<matplotlib.image.AxesImage at 0x278253867f0>
```



In [9]:
```
# spliting the data where x contains images
x = pd.DataFrame(digits.data)
x
```

Out[9]:

	0	1	2	3	4	5	6	7	8	9	...	54	55	56	57	58	59	60	61	62	63
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	6.0	13.0	10.0	0.0	0.0	0.0
1	0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	11.0	16.0	10.0	0.0	0.0
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.0	0.0	...	5.0	0.0	0.0	0.0	0.0	3.0	11.0	16.0	9.0	0.0
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0	0.0	8.0	...	9.0	0.0	0.0	0.0	7.0	13.0	13.0	9.0	0.0	0.0
4	0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	2.0	16.0	4.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1792	0.0	0.0	4.0	10.0	13.0	6.0	0.0	0.0	0.0	1.0	...	4.0	0.0	0.0	0.0	2.0	14.0	15.0	9.0	0.0	0.0
1793	0.0	0.0	6.0	16.0	13.0	11.0	1.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	6.0	16.0	14.0	6.0	0.0	0.0
1794	0.0	0.0	1.0	11.0	15.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	2.0	9.0	13.0	6.0	0.0	0.0
1795	0.0	0.0	2.0	10.0	7.0	0.0	0.0	0.0	0.0	0.0	...	2.0	0.0	0.0	0.0	5.0	12.0	16.0	12.0	0.0	0.0
1796	0.0	0.0	10.0	14.0	8.0	1.0	0.0	0.0	0.0	2.0	...	8.0	0.0	0.0	1.0	8.0	12.0	14.0	12.0	1.0	0.0

1797 rows × 64 columns

In [10]:
```
digits.images[0]
```

Out[10]:
```
array([[ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.],
       [ 0.,  0., 13., 15., 10., 15.,  5.,  0.],
       [ 0.,  3., 15.,  2.,  0., 11.,  8.,  0.],
       [ 0.,  4., 12.,  0.,  0.,  8.,  8.,  0.],
       [ 0.,  5.,  8.,  0.,  0.,  9.,  8.,  0.],
       [ 0.,  4., 11.,  0.,  1., 12.,  7.,  0.],
       [ 0.,  2., 14.,  5., 10., 12.,  0.,  0.],
       [ 0.,  0.,  6., 13., 10.,  0.,  0.,  0.]])
```

In [11]:
```
# spliting the data where y contains corresponding value
y = pd.DataFrame(digits.target)
y
```

Out[11]:

	0
0	0
1	1
2	2
3	3
4	4
...	...
1792	9
1793	0
1794	8
1795	9
1796	8

1797 rows × 1 columns

In [12]:
```
from sklearn.preprocessing import StandardScaler
# standardization is the scaling technique where we make the mean as 0 and standard deviation as 1
# in pyhton we have the inbuilt function present for it
```

In [13]:
```
x_std = StandardScaler().fit_transform(x)
x_std
```

Out[13]:
```
array([[ 0.        , -0.33501649, -0.04308102, ..., -1.14664746,
        -0.5056698 , -0.19600752],
       [ 0.        , -0.33501649, -1.09493684, ...,  0.54856067,
        -0.5056698 , -0.19600752],
       [ 0.        , -0.33501649, -1.09493684, ...,  1.56568555,
         1.6951369 , -0.19600752],
       ...,
       [ 0.        , -0.33501649, -0.88456568, ..., -0.12952258,
        -0.5056698 , -0.19600752],
       [ 0.        , -0.33501649, -0.67419451, ...,  0.8876023 ,
        -0.5056698 , -0.19600752],
       [ 0.        , -0.33501649,  1.00877481, ...,  0.8876023 ,
        -0.26113572, -0.19600752]])
```

In [14]:
```
x_std.shape
```

Out[14]:
```
(1797, 64)
```

In [15]:
```
x1 = x_std.T
x1
```

Out[15]:
```
array([[ 0.        ,  0.        ,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [-0.33501649, -0.33501649, -0.33501649, ..., -0.33501649,
        -0.33501649, -0.33501649],
       [-0.04308102, -1.09493684, -1.09493684, ..., -0.88456568,
        -0.67419451,  1.00877481],
       ...,
       [-1.14664746,  0.54856067,  1.56568555, ..., -0.12952258,
         0.8876023 ,  0.8876023 ],
       [-0.5056698 , -0.5056698 ,  1.6951369 , ..., -0.5056698 ,
        -0.5056698 , -0.26113572],
       [-0.19600752, -0.19600752, -0.19600752, ..., -0.19600752,
        -0.19600752, -0.19600752]])
```

In [16]:
```
x1.shape #-------------->64*64
```

Out[16]:
```
(64, 1797)
```

In [17]:
```
#covariance matrix

cov_mat = np.cov(x1)
```

In [18]:
```
print(cov_mat)
```

```
[[ 0.          0.          0.         ...  0.          0.
   0.        ]
 [ 0.          1.00055679  0.55692803 ... -0.02988686  0.02656195
  -0.04391324]
 [ 0.          0.55692803  1.00055679 ... -0.04120565  0.07263924
   0.08256908]
 ...
 [ 0.         -0.02988686 -0.04120565 ...  1.00055679  0.64868875
   0.26213704]
 [ 0.          0.02656195  0.07263924 ...  0.64868875  1.00055679
   0.62077355]
 [ 0.         -0.04391324  0.08256908 ...  0.26213704  0.62077355
   1.00055679]]
```

In [19]:
```
cov_mat.shape
```

Out[19]:
```
(64, 64)
```

In [20]:
```
# eigen values and eigen vectors

eigen_val, eigen_vec = np.linalg.eig(cov_mat)
```

In [21]:
```
eigen_val
```

Out[21]:
```
array([7.34477606, 5.83549054, 5.15396118, 3.96623597, 2.9663452 ,
       2.57204442, 2.40600941, 2.06867355, 1.82993314, 1.78951739,
       1.69784616, 1.57287889, 1.38870781, 1.35933609, 1.32152536,
       1.16829176, 1.08368678, 0.99977862, 0.97438293, 0.90891242,
       0.82271926, 0.77631014, 0.71155675, 0.64552365, 0.59527399,
       0.5765018 , 0.52673155, 0.5106363 , 0.48686381, 0.45560107,
       0.44285155, 0.42230086, 0.3991063 , 0.39110111, 0.36094517,
       0.34860306, 0.3195963 , 0.29406627, 0.05037444, 0.27692285,
       0.06328961, 0.258273  , 0.24783029, 0.2423566 , 0.07635394,
       0.08246812, 0.09018543, 0.09840876, 0.10250434, 0.11188655,
       0.11932898, 0.12426371, 0.13321081, 0.14311427, 0.217582  ,
       0.15818474, 0.16875236, 0.20799593, 0.17612894, 0.2000909 ,
       0.18983516, 0.        , 0.        , 0.        ])
```

In [22]:
```
eigen_vec
```

Out[22]:
```
array([[ 0.        ,  0.        ,  0.        , ...,  1.        ,
         0.        ,  0.        ],
       [ 0.18223392, -0.04702701,  0.02358821, ...,  0.        ,
         0.        ,  0.        ],
       [ 0.285868  , -0.0595648 , -0.05679875, ...,  0.        ,
         0.        ,  0.        ],
       ...,
       [ 0.103198  ,  0.24261778, -0.02227952, ...,  0.        ,
         0.        ,  0.        ],
       [ 0.1198106 ,  0.16508926,  0.10036559, ...,  0.        ,
         0.        ,  0.        ],
       [ 0.07149362,  0.07132924,  0.09244589, ...,  0.        ,
         0.        ,  0.        ]])
```

In [23]:
```
len(eigen_val)
```

Out[23]:
```
64
```

In [24]:
```
len(eigen_vec)
```

Out[24]:
```
64
```

In [25]:
```

```

In [26]:
```
total = sum(eigen_val)
print(total)
```

```
61.033964365256246
```

In [27]:
```
7.34477606/61.03396436525629*100
```

Out[27]:
```
12.033916093087718
```

In [28]:
```
var_exp = [(i/total)*100 for i in eigen_val]
```

In [29]:
```
var_exp
```

Out[29]:
```
[12.033916097734894,
 9.561054403097884,
 8.444414892624563,
 6.498407907524179,
 4.8601548759664075,
 4.2141198692719515,
 3.9420828035673807,
 3.389380924638329,
 2.998221011625233,
 2.9320025512522165,
 2.7818054635503273,
 2.5770550925819933,
 2.2753033157642473,
 2.22717973951434,
 2.165229431849237,
 1.914166606442132,
 1.7755470851681974,
 1.6380692742844194,
 1.5964601688623399,
 1.489191187087823,
 1.3479695658179367,
 1.2719313702347568,
 1.165837350591948,
 1.0576465985363221,
 0.9753159471981127,
 0.9445589897319985,
 0.8630138269707233,
 0.8366428536685135,
 0.7976932484112402,
 0.7464713709260624,
 0.7255821513702759,
 0.6919112454811802,
 0.6539085355726166,
 0.6407925738459848,
 0.5913841117223416,
 0.5711624052235234,
 0.523636803416636,
 0.48180758644514166,
 0.08253509448180277,
 0.4537192598584488,
 0.103695730155717,
 0.4231627532327801,
 0.40605306997903834,
 0.39708480827582837,
```

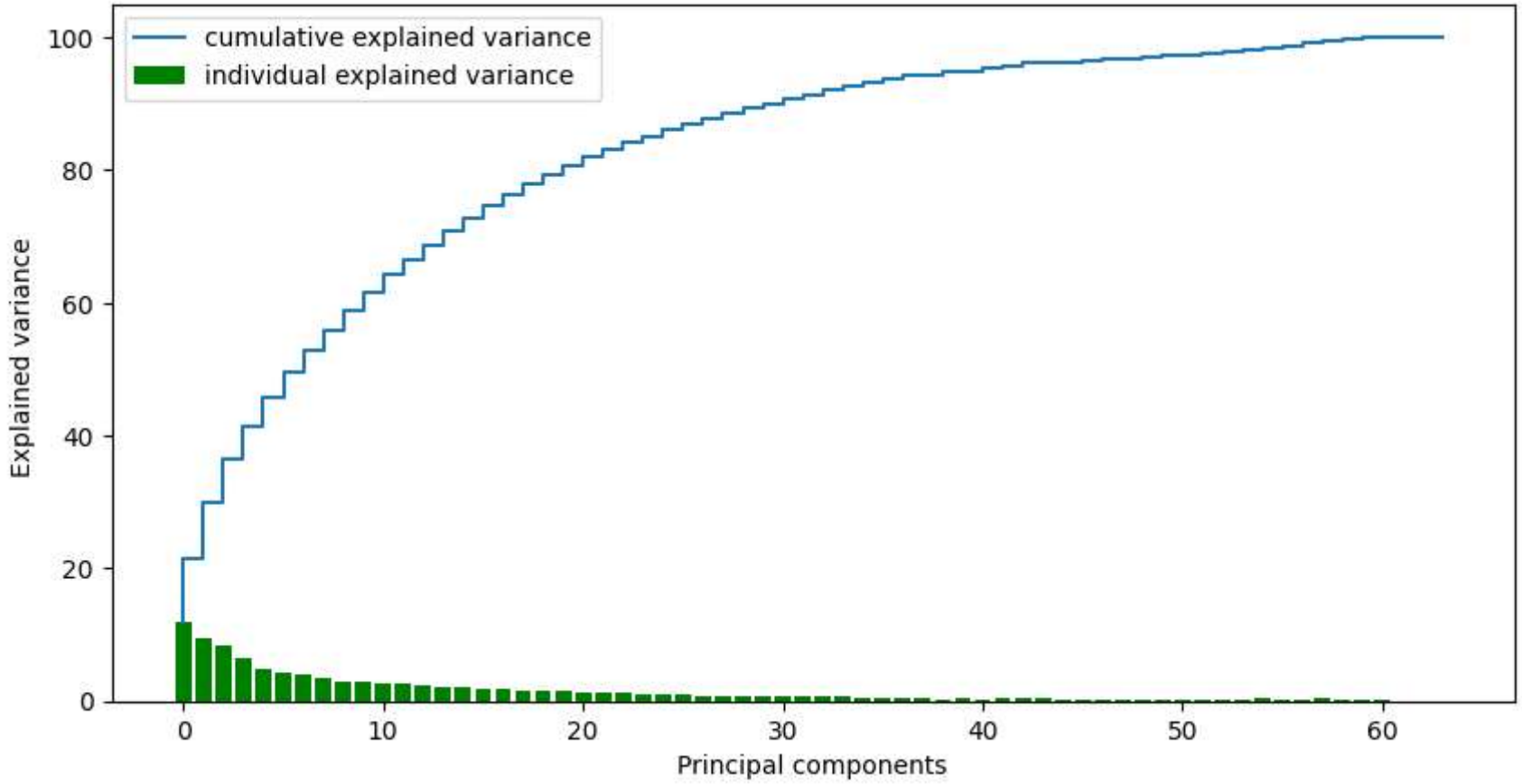
```
0.1251007424973014,
0.13511841133708574,
0.14776269410608758,
0.1612360622567286,
0.16794638749558413,
0.18331849919718302,
0.1955124260198182,
0.20359763452537666,
0.2182568577120085,
0.2344830055356357,
0.35649330314261785,
0.25917494088146376,
0.2764892635235473,
0.3407871814702996,
0.28857529410893457,
0.3278353352879547,
0.31103200734535735,
0.0,
0.0,
0.0]
```

In [30]:
```
#cumulative explained variance

cum_var_exp = np.cumsum(var_exp)
```

In [31]:
```
cum_var_exp
```

Out[31]:
```
array([ 12.0339161 ,  21.5949705 ,  30.03938539,  36.5377933 ,
        41.39794818,  45.61206805,  49.55415085,  52.94353177,
        55.94175279,  58.87375534,  61.6555608 ,  64.23261589,
        66.50791921,  68.73509895,  70.90032838,  72.81449499,
        74.59004207,  76.22811135,  77.82457152,  79.3137627 ,
        80.66173227,  81.93366364,  83.09950099,  84.15714759,
        85.13246353,  86.07702252,  86.94003635,  87.77667921,
        88.57437245,  89.32084382,  90.04642598,  90.73833722,
        91.39224576,  92.03303833,  92.62442244,  93.19558485,
        93.71922165,  94.20102924,  94.28356433,  94.73728359,
        94.84097932,  95.26414208,  95.67019515,  96.06727995,
        96.1923807 ,  96.32749911,  96.4752618 ,  96.63649786,
        96.80444425,  96.98776275,  97.18327518,  97.38687281,
        97.60512967,  97.83961267,  98.19610598,  98.45528092,
        98.73177018,  99.07255736,  99.36113266,  99.68896799,
       100.        , 100.        , 100.        , 100.        ])
```

In [32]:
```
# whats the use of screeplot
# scree plot is a line plot of the eigenvalues of factors or principal components in an analysis
# scree plot
plt.figure(figsize=(10, 5))
plt.bar(range(len(var_exp)), var_exp, label='individual explained variance', color = 'g')
plt.step(range(len(cum_var_exp)), cum_var_exp,label='cumulative explained variance')
plt.ylabel('Explained variance ')
plt.xlabel('Principal components')
plt.legend()
plt.show()
```



In [33]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x_std, y, train_size = 0.8)
```

In [34]:
```
(1797/100)*80
```

Out[34]:
```
1437.6
```

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

Out[35]:
```
(1437, 64)
```

In [36]:
```
(1797/100)*20
```

Out[36]:
```
359.4
```

In [37]:
```
x_test.shape
```

Out[37]:
```
(360, 64)
```

In [38]:
```
from sklearn.decomposition import PCA
```

In [39]:
```
# we can select the components from the scree plot or can directly pass the percentage of the information
# that we want to retain in the pca function

# pca = PCA(0.95)
pca = PCA(n_components= 20)
pca_x_train = pca.fit_transform(x_train)
pca_x_test = pca.transform(x_test)
```

In [40]:
```
pca_x_train.shape
```

Out[40]:
```
(1437, 20)
```

In [41]:
```
pca_x_test.shape
```

Out[41]:
```
(360, 20)
```

In [42]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [43]:
```
dt = DecisionTreeClassifier()
```

In [44]:
```
dt.fit(pca_x_train,y_train)
```

Out[44]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [45]:
```
predict = dt.predict(pca_x_test)
```

In [46]:
```
predict
```

Out[46]:
```
array([4, 0, 2, 2, 0, 7, 5, 6, 7, 0, 7, 1, 7, 3, 3, 1, 7, 9, 2, 6, 7, 1,
       2, 8, 5, 5, 2, 8, 6, 7, 3, 7, 6, 6, 4, 0, 5, 0, 5, 7, 3, 9, 2, 9,
       7, 2, 4, 6, 0, 8, 9, 9, 2, 0, 6, 2, 5, 9, 0, 2, 0, 5, 4, 0, 1, 4,
       1, 8, 5, 2, 3, 7, 5, 4, 6, 7, 3, 2, 6, 9, 9, 0, 3, 9, 2, 3, 1, 8,
       6, 0, 3, 0, 5, 7, 9, 0, 6, 6, 1, 6, 5, 0, 2, 6, 0, 4, 4, 2, 0, 1,
       7, 5, 0, 3, 9, 7, 6, 8, 9, 1, 7, 8, 6, 4, 2, 5, 0, 1, 1, 6, 6, 6,
       3, 3, 0, 5, 6, 1, 9, 7, 8, 1, 3, 8, 3, 3, 1, 2, 3, 0, 7, 1, 6, 6,
       2, 0, 5, 5, 1, 3, 7, 0, 9, 8, 9, 4, 8, 6, 2, 6, 5, 0, 9, 9, 7, 4,
       5, 0, 8, 8, 7, 0, 5, 2, 3, 0, 7, 3, 9, 9, 7, 1, 1, 3, 2, 1, 3, 7,
       5, 4, 9, 1, 2, 1, 4, 3, 8, 0, 6, 4, 0, 5, 9, 9, 5, 5, 8, 2, 0, 9,
       9, 9, 3, 6, 8, 5, 1, 3, 8, 3, 7, 2, 1, 9, 1, 0, 3, 8, 1, 9, 7, 0,
       2, 1, 3, 7, 4, 4, 9, 0, 2, 9, 7, 1, 8, 2, 1, 1, 2, 6, 2, 1, 2, 1,
       4, 5, 5, 3, 6, 8, 9, 8, 1, 9, 0, 9, 8, 0, 2, 4, 9, 9, 9, 4, 9, 1,
       0, 6, 1, 1, 4, 8, 4, 9, 3, 0, 6, 6, 2, 0, 4, 2, 9, 4, 4, 9, 1, 8,
       4, 5, 3, 4, 6, 6, 2, 6, 5, 4, 7, 9, 2, 2, 8, 5, 8, 0, 9, 9, 0, 0,
       4, 7, 5, 9, 3, 2, 3, 7, 8, 6, 9, 2, 9, 5, 0, 1, 5, 3, 7, 5, 1, 4,
       8, 3, 3, 1, 8, 8, 6, 3])
```

In [47]:
```
y_test
```

Out[47]:

	0
1512	4
1563	0
1232	2
1437	2
179	0
...	...
1130	3
630	8
1763	8
197	6
1170	3

360 rows × 1 columns

In [48]:
```
from sklearn.metrics import *
```

In [49]:
```
cm = confusion_matrix(y_test,predict)
```

In [50]:
```
cm.shape
```

Out[50]:
```
(10, 10)
```

In [51]:
```
cm
# the diagonals in the matrix is returning the correctly classified data
# and the rest other is missclassified data
```

Out[51]:
```
array([[41,  0,  0,  0,  0,  0,  0,  0,  0,  4],
       [ 0, 31,  1,  1,  0,  0,  2,  0,  0,  0],
       [ 0,  1, 33,  0,  0,  0,  0,  0,  1,  2],
       [ 0,  1,  2, 33,  0,  0,  1,  1,  4,  0],
       [ 0,  0,  0,  0, 28,  0,  0,  1,  0,  0],
       [ 0,  0,  0,  1,  0, 29,  1,  0,  0,  4],
       [ 1,  2,  1,  0,  0,  0, 31,  1,  0,  0],
       [ 0,  1,  0,  0,  0,  0,  1, 29,  0,  0],
       [ 0,  1,  0,  0,  0,  1,  0,  0, 23,  3],
       [ 0,  2,  1,  1,  1,  3,  0,  0,  2, 32]], dtype=int64)
```

In [52]:
```
accuracy_score(y_test,predict)
```

Out[52]:
```
0.8611111111111112
```

In [ ]:

Unsupervised learning

1. **K-Means Clustering**:
    - Use when:
        - You want to partition data into a predefined number of clusters.
        - You have a large dataset and need a scalable clustering algorithm.
        - You have numeric data and clusters are likely to be globular (spherical) in shape.
2. **Hierarchical Clustering**:
    - Use when:
        - You want to explore the hierarchical structure of your data.
        - You don't know the number of clusters in advance and want a dendrogram to visualize different cluster configurations.
        - You have relatively small to medium-sized datasets.
3. **Principal Component Analysis (PCA)**:
    - Use when:
        - You have high-dimensional data and want to reduce its dimensionality while preserving most of its variance.
        - You want to visualize high-dimensional data in a lower-dimensional space.
        - You want to remove noise or redundant features from your dataset.

Supervised learning

Certainly! Here's a list of common supervised machine learning models along with guidance on when and where to use them for different types of datasets:

1. Linear Regression:
- **When to Use**:
  - Predicting a continuous numerical outcome based on one or more input features.
  - Modeling linear relationships between variables.
- **Where to Use**:
  - Predicting house prices based on features like square footage, number of bedrooms, etc.
  - Predicting sales revenue based on advertising spend across different channels.

2. Logistic Regression:

- **When to Use**:
  - Binary classification problems where the target variable has two classes.
  - Probabilistic interpretation of class membership.
- **Where to Use**:
  - Predicting whether an email is spam or not spam based on its features.
  - Predicting whether a customer will churn (leave) a subscription service based on their behavior.

3. Decision Trees:
- **When to Use**:
  - Modeling non-linear relationships between features and target.
  - Easy to interpret and visualize.
- **Where to Use**:
  - Customer segmentation based on demographic and behavioral characteristics.
  - Diagnosis of medical conditions based on symptoms and patient data.

4. Random Forest:
- **When to Use**:
  - Handling large datasets with many features.
  - Reducing overfitting compared to single decision trees.
- **Where to Use**:
  - Predicting customer churn in a telecom company using various customer attributes.
  - Predicting the likelihood of loan default based on borrower characteristics.

5. Support Vector Machines (SVM):
- **When to Use**:
  - Binary classification problems where the data is separable or nearly separable.
  - Effective in high-dimensional spaces.
- **Where to Use**:
  - Text classification tasks such as sentiment analysis or spam detection.
  - Image classification tasks such as identifying objects in images.

6. k-Nearest Neighbors (kNN):
- **When to Use**:
  - Instances are close to each other in feature space are likely to belong to the same class.
  - Non-parametric and lazy learning algorithm.
- **Where to Use**:
  - Recommender systems for recommending similar products or movies to users.
  - Predicting the classification of a new data point based on the classes of its nearest neighbors.

7. Gradient Boosting Models (e.g., XGBoost, LightGBM):
- **When to Use**:
  - High predictive accuracy with ensemble learning.
  - Handles complex relationships and interactions well.
- **Where to Use**:
  - Click-through rate prediction in online advertising.

  - Predicting customer lifetime value based on historical transaction data.

8. Neural Networks (e.g., Multi-Layer Perceptrons):
- **When to Use**:
  - Handling complex, non-linear relationships in high-dimensional data.
  - Requires large amounts of data and computational resources.
- **Where to Use**:
  - Image recognition tasks such as identifying objects in images.
  - Natural language processing tasks such as sentiment analysis or machine translation.

Choose the appropriate supervised learning model based on factors such as the nature of the data, the problem at hand, the size of the dataset, and the desired interpretability of the model. Experimentation and evaluation with multiple models may be necessary to find the best fit for your specific task.

dngny9ii8

April 30, 2024

[3]:
```
#import necessary libraries
import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
```

[6]:
```
#Load the Dataset
df=pd.read_csv("C:/Users/Fadilah Thasnim/Desktop/Academics/6 - Semester/3. ML/
 ↪Lab/health care diabetes.csv")
df
```

[6]:

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

	DiabetesPedigreeFunction	Age	Outcome
0	0.627	50	1
1	0.351	31	0
2	0.672	32	1
3	0.167	21	0
4	2.288	33	1
..	…	…	…
763	0.171	63	0
764	0.340	27	0
765	0.245	30	0
766	0.349	47	1
767	0.315	23	0

```
[768 rows x 9 columns]
```

[7]:
```
px.histogram(data_frame=df, x='Outcome',␣
↪color='Outcome',color_discrete_sequence=['#05445E','#75E6DA'])
```

[9]:
```
px.
↪histogram(data_frame=df,x='Age',color='Age',color_discrete_sequence=['#05445E','#75E6DA'])
```

[10]:
```
px.
↪scatter(data_frame=df,x='BMI',color='BMI',color_discrete_sequence=['#05445E','#75E6DA'])
```

[11]:
```
# Data Preprocessing
df.describe().T
```

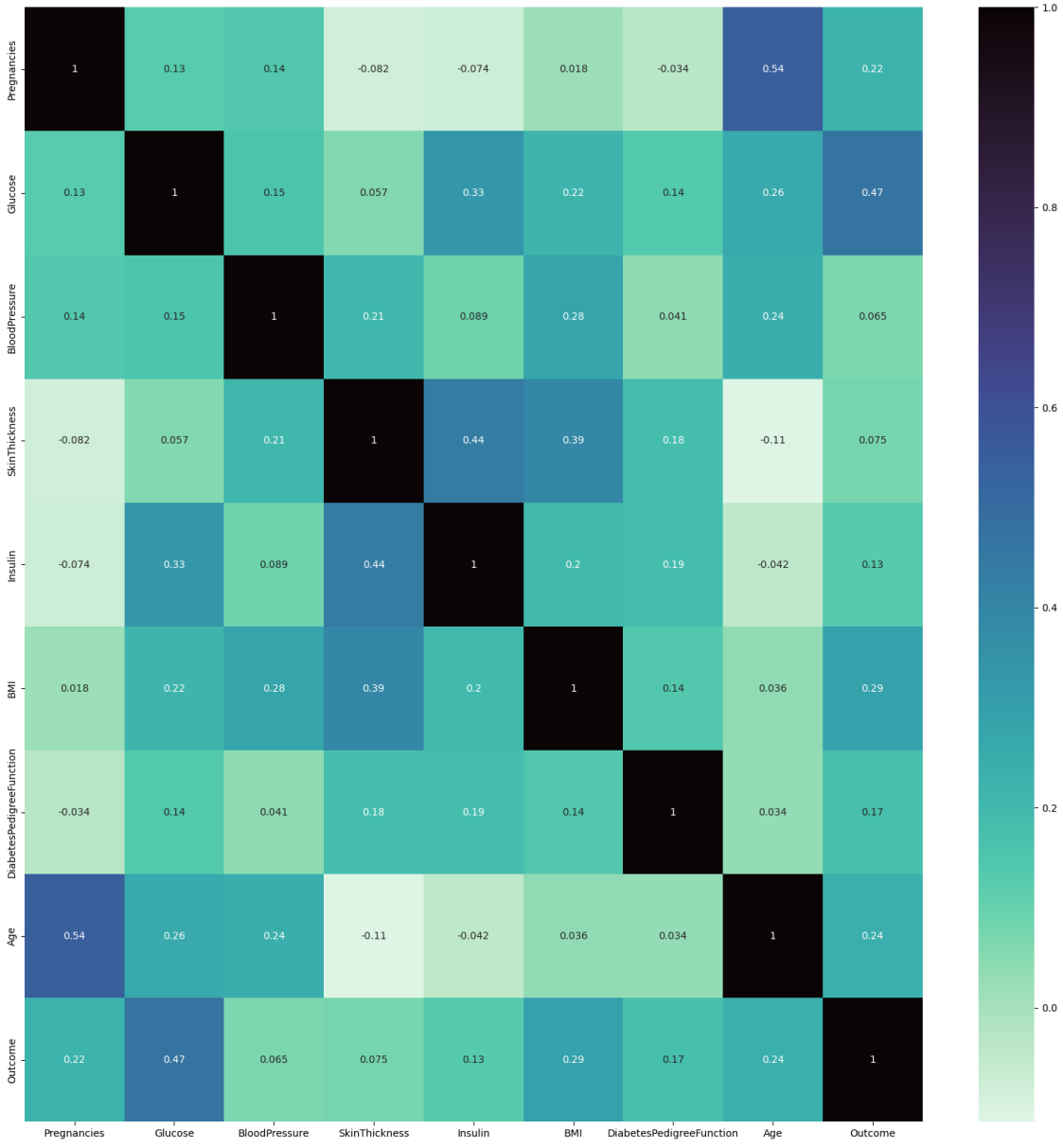
[11]:

	count	mean	std	min	25%	\
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	
Glucose	768.0	120.894531	31.972618	0.000	99.00000	
BloodPressure	768.0	69.105469	19.355807	0.000	62.00000	
SkinThickness	768.0	20.536458	15.952218	0.000	0.00000	
Insulin	768.0	79.799479	115.244002	0.000	0.00000	
BMI	768.0	31.992578	7.884160	0.000	27.30000	
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375	
Age	768.0	33.240885	11.760232	21.000	24.00000	
Outcome	768.0	0.348958	0.476951	0.000	0.00000	

	50%	75%	max
Pregnancies	3.0000	6.00000	17.00
Glucose	117.0000	140.25000	199.00
BloodPressure	72.0000	80.00000	122.00
SkinThickness	23.0000	32.00000	99.00
Insulin	30.5000	127.25000	846.00
BMI	32.0000	36.60000	67.10
DiabetesPedigreeFunction	0.3725	0.62625	2.42
Age	29.0000	41.00000	81.00
Outcome	0.0000	1.00000	1.00

[12]:
```
corr = df.corr()
```

[13]:
```
plt.figure(figsize=(20,20))
sns.heatmap(corr, cmap='mako_r',annot=True)
plt.show()
```



[14]:
```
# Get the absolute value of the correlation
cor_target = abs(corr["Outcome"])
# Select highly correlated features (thresold = 0.2)
relevant_features = cor_target[cor_target>0.2]
# Collect the names of the features
names = [index for index, value in relevant_features.iteritems()]
# Drop the target variable from the results
names.remove('Outcome')
# Display the results
print(names)
```

```
['Pregnancies', 'Glucose', 'BMI', 'Age']

C:\Users\Fadilah Thasnim\AppData\Local\Temp\ipykernel_25900\2249588575.py:6:
FutureWarning:

iteritems is deprecated and will be removed in a future version. Use .items
instead.
```

[16]:
```python
# Assign data and labels
X = df[names].values
y = df['Outcome']
```

[18]:
```python
def scale(X):
     # Calculate the mean and standard deviation of each feature
    mean = np.mean(X, axis=0)
    std = np.std(X, axis=0)

    # Standardize the data
    X = (X - mean) / std
    return X
X = scale(X)
```

[19]:
```python
# Split into train and Testing
def train_test_split(X, y, random_state=41, test_size=0.2):
     # Get number of samples
    n_samples = X.shape[0]

    # Set the seed for the random number generator
    np.random.seed(random_state)

    # Shuffle the indices
    shuffled_indices = np.random.permutation(np.arange(n_samples))

    # Determine the size of the test set
    test_size = int(n_samples * test_size)

    # Split the indices into test and train
    test_indices = shuffled_indices[:test_size]
    train_indices = shuffled_indices[test_size:]

    # Split the features and target arrays into test and train
    X_train, X_test = X[train_indices], X[test_indices]
    y_train, y_test = y[train_indices], y[test_indices]

    return X_train, X_test, y_train, y_test
```

[20]:
```
#split the  data into traing and validating
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
↪random_state=42)
```

implements a basic version of the SVM algorithm using gradient descent for optimization. Here's a summary of the functionalities within the class:

Initialization: The **init** method sets up the parameters of the SVM model, including the number of iterations for gradient descent (iterations), the learning rate (lr), and the regularization parameter (lambdaa). Parameter Initialization: The initialize_parameters method initializes the weight vector w and the bias b to zeros. Gradient Descent: The gradient_descent method implements the gradient descent algorithm to update the model parameters (w and b) based on the gradient of the loss function with respect to the parameters. Parameter Update: The update_parameters method updates the model parameters (w and b) based on the computed gradients. Fitting the Model: The fit method fits the SVM model to the training data by iteratively applying gradient descent to optimize the parameters. Prediction: The predict method predicts the class labels of input data based on the trained model. It calculates the dot product of the input features with the weight vector w, subtracts the bias b, and assigns class labels based on whether the output is greater than zero.

[29]:
```
class SVM:
    def __init__(self, iterations=1000, lr=0.01, lambdaa=0.01):
        self.lambdaa = lambdaa
        self.iterations = iterations
        self.lr = lr
        self.w = None
        self.b = None

    def initialize_parameters(self, X):
        m, n = X.shape
        self.w = np.zeros(n)
        self.b = 0

    def gradient_descent(self, X, y):
        y_ = np.where(y <= 0, -1, 1)
        for i, x in enumerate(X):
            if y_[i] * (np.dot(x, self.w) - self.b) >= 1:
                dw = 2 * self.lambdaa * self.w
                db = 0
            else:
                dw = 2 * self.lambdaa * self.w - np.dot(x, y_[i])
                db = y_[i]
            self.update_parameters(dw, db)

    def update_parameters(self, dw, db):
        self.w = self.w - self.lr * dw
        self.b = self.b - self.lr * db
```

```
    def fit(self, X, y):
        self.initialize_parameters(X)
        for i in range(self.iterations):
            self.gradient_descent(X, y)

    def predict(self, X):
        # get the outputs
        output = np.dot(X, self.w) - self.b
        # get the signs of the labels depending on if it's greater/less than zero
        label_signs = np.sign(output)
        # set predictions to 0 if they are less than or equal to -1 else set them to 1
        predictions = np.where(label_signs <= -1, 0, 1)
        return predictions
```

[30]:
```
def accuracy(y_true, y_pred):
    """
    Computes the accuracy of a classification model.

    Parameters:
    ----------
        y_true (numpy array): A numpy array of true labels for each data point.
        y_pred (numpy array): A numpy array of predicted labels for each data point.

    Returns:
    ----------
        float: The accuracy of the model
    """
    total_samples = len(y_true)
    correct_predictions = np.sum(y_true == y_pred)
    return (correct_predictions / total_samples)
```

[31]:
```
#performs the following tasks
   #Initializes an SVM model.
   #Fits the model to the training data.
   #Predicts labels for the test data.
   #Evaluates the accuracy of the predictions against the true labels.
model = SVM()
model.fit(X_train,y_train)
predictions = model.predict(X_test)

accuracy(y_test, predictions)
```

[31]: 0.7581699346405228

[32]:
```
# in sklearn implementation

from sklearn.svm import SVC
skmodel = SVC()
skmodel.fit(X_train, y_train)
sk_predictions = skmodel.predict(X_test)

accuracy(y_test, sk_predictions)
```

[32]: 0.738562091503268

[ ]: