Machine Learning Practical’s

Q.1: Write a python program to Prepare Scatter Plot (Use Forge Dataset / Iris Dataset)

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

data = pd.read\_csv("Iris.csv")

print (data.head(10))

import pandas as pd

import matplotlib.pyplot as plt

iris = pd.read\_csv("Iris.csv")

plt.plot(iris.Id, iris["SepalLengthCm"], "r--")

plt.show

iris.plot(kind ="scatter",

x ='SepalLengthCm',

y ='PetalLengthCm')

plt.grid()

Output :-

Id SepalLengthCm SepalWidthCm PetalLengthCm PetalWidthCm Species

0 1 5.1 3.5 1.4 0.2 Iris-setosa

1 2 4.9 3.0 1.4 0.2 Iris-setosa

2 3 4.7 3.2 1.3 0.2 Iris-setosa

3 4 4.6 3.1 1.5 0.2 Iris-setosa

4 5 5.0 3.6 1.4 0.2 Iris-setosa

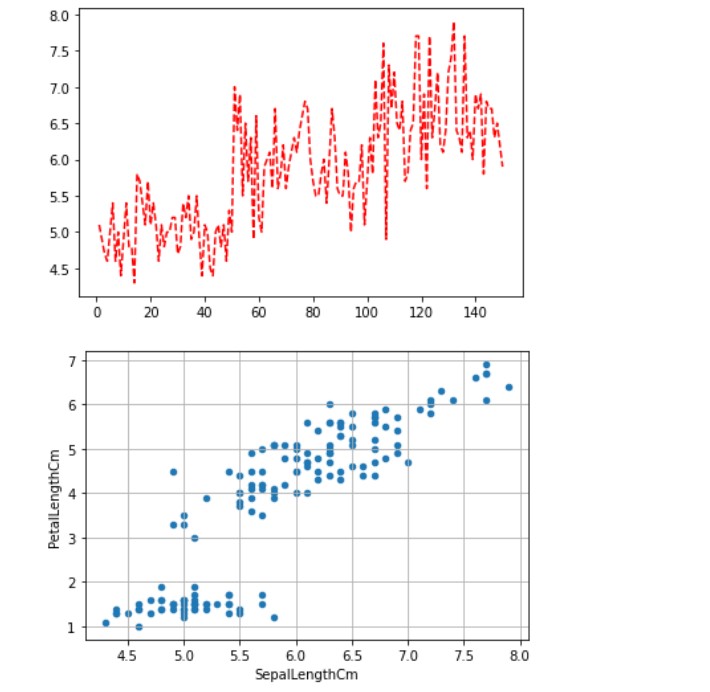
5 6 5.4 3.9 1.7 0.4 Iris-setosa

6 7 4.6 3.4 1.4 0.3 Iris-setosa

7 8 5.0 3.4 1.5 0.2 Iris-setosa

8 9 4.4 2.9 1.4 0.2 Iris-setosa

9 10 4.9 3.1 1.5 0.1 Iris-setosa



Q.2: Write a python program to find all null values in a given data set and remove them.

# importing pandas package

import pandas as pd

# making data frame from csv file

data = pd.read\_csv("employees.csv")

# creating bool series True for NaN values

bool\_series = pd.isnull(data["Gender"])

# filtering data

# displaying data only with Gender = NaN

data[bool\_series]

Output :-

| **First Name** | **Gender** | **Start Date** | **Last Login Time** | **Salary** | **Bonus %** | **Senior Management** | **Team** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **20** | Lois | NaN | 4/22/1995 | 7:18 PM | 64714 | 4.934 | True | Legal |
| **22** | Joshua | NaN | 3/8/2012 | 1:58 AM | 90816 | 18.816 | True | Client Services |
| **27** | Scott | NaN | 7/11/1991 | 6:58 PM | 122367 | 5.218 | False | Legal |
| **31** | Joyce | NaN | 2/20/2005 | 2:40 PM | 88657 | 12.752 | False | Product |
| **41** | Christine | NaN | 6/28/2015 | 1:08 AM | 66582 | 11.308 | True | Business Development |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **961** | Antonio | NaN | 6/18/1989 | 9:37 PM | 103050 | 3.050 | False | Legal |
| **972** | Victor | NaN | 7/28/2006 | 2:49 PM | 76381 | 11.159 | True | Sales |
| **985** | Stephen | NaN | 7/10/1983 | 8:10 PM | 85668 | 1.909 | False | Legal |
| **989** | Justin | NaN | 2/10/1991 | 4:58 PM | 38344 | 3.794 | False | Legal |
| **995** | Henry | NaN | 11/23/2014 | 6:09 AM | 132483 | 16.655 | False | Distribution |

145 rows × 8 columns

# importing pandas module

import pandas as pd

# making data frame from csv file

data = pd.read\_csv("employees.csv")

# making new data frame with dropped NA values

new\_data = data.dropna(axis = 0, how ='any')

new\_data

Output :-

| **First Name** | **Gender** | **Start Date** | **Last Login Time** | **Salary** | **Bonus %** | **Senior Management** | **Team** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | Douglas | Male | 8/6/1993 | 12:42 PM | 97308 | 6.945 | True | Marketing |
| **2** | Maria | Female | 4/23/1993 | 11:17 AM | 130590 | 11.858 | False | Finance |
| **3** | Jerry | Male | 3/4/2005 | 1:00 PM | 138705 | 9.340 | True | Finance |
| **4** | Larry | Male | 1/24/1998 | 4:47 PM | 101004 | 1.389 | True | Client Services |
| **5** | Dennis | Male | 4/18/1987 | 1:35 AM | 115163 | 10.125 | False | Legal |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **994** | George | Male | 6/21/2013 | 5:47 PM | 98874 | 4.479 | True | Marketing |
| **996** | Phillip | Male | 1/31/1984 | 6:30 AM | 42392 | 19.675 | False | Finance |
| **997** | Russell | Male | 5/20/2013 | 12:39 PM | 96914 | 1.421 | False | Product |
| **998** | Larry | Male | 4/20/2013 | 4:45 PM | 60500 | 11.985 | False | Business Development |
| **999** | Albert | Male | 5/15/2012 | 6:24 PM | 129949 | 10.169 | True | Sales |

764 rows × 8 columns

Q.3: Write a python program the Categorical values in numeric format for a given dataset.

import pandas as pd

import numpy as np

# Define the headers since the data does not have any

headers = ["symboling", "normalized\_losses", "make", "fuel\_type", "aspiration",

"num\_doors", "body\_style", "drive\_wheels", "engine\_location",

"wheel\_base", "length", "width", "height", "curb\_weight",

"engine\_type", "num\_cylinders", "engine\_size", "fuel\_system",

"bore", "stroke", "compression\_ratio", "horsepower", "peak\_rpm",

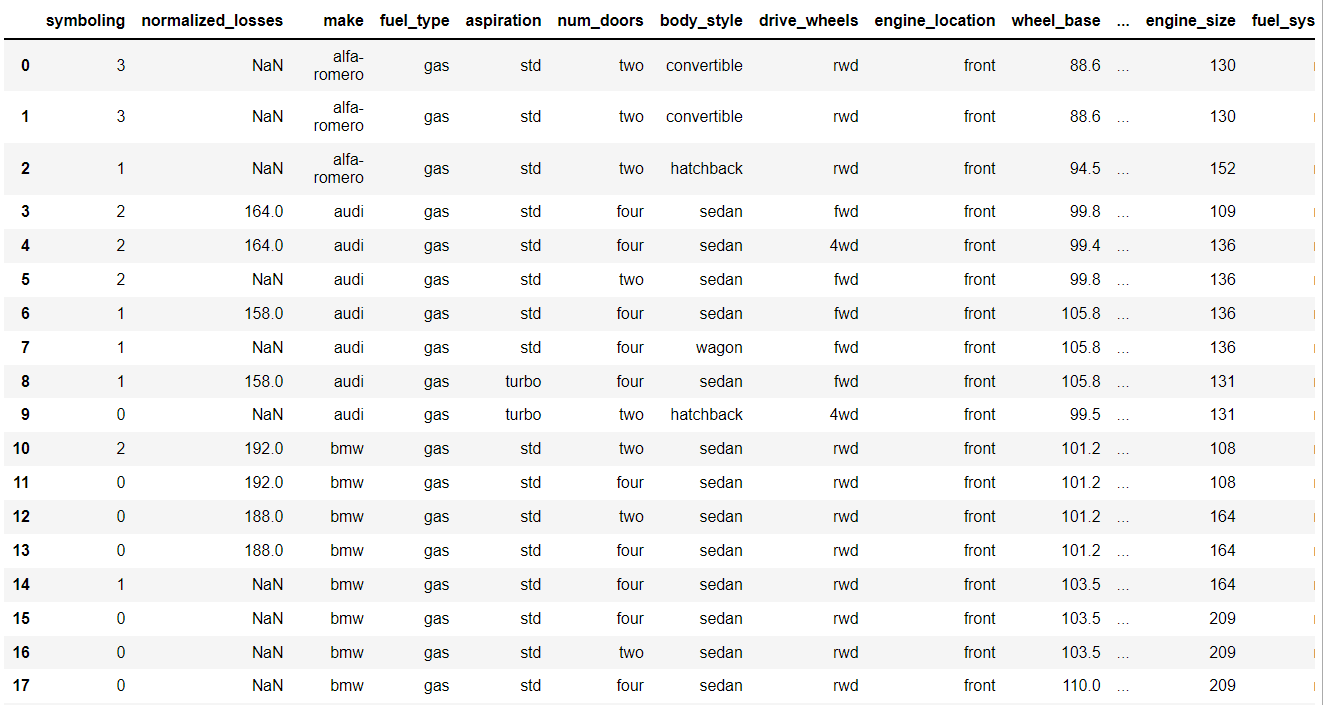
"city\_mpg", "highway\_mpg", "price"]

# Read in the CSV file and convert "?" to NaN

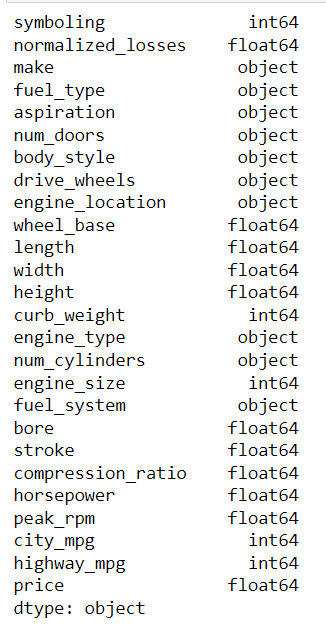
df = pd.read\_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/autos/imports-85.data",

header=None, names=headers, na\_values="?" )

df.head(30)

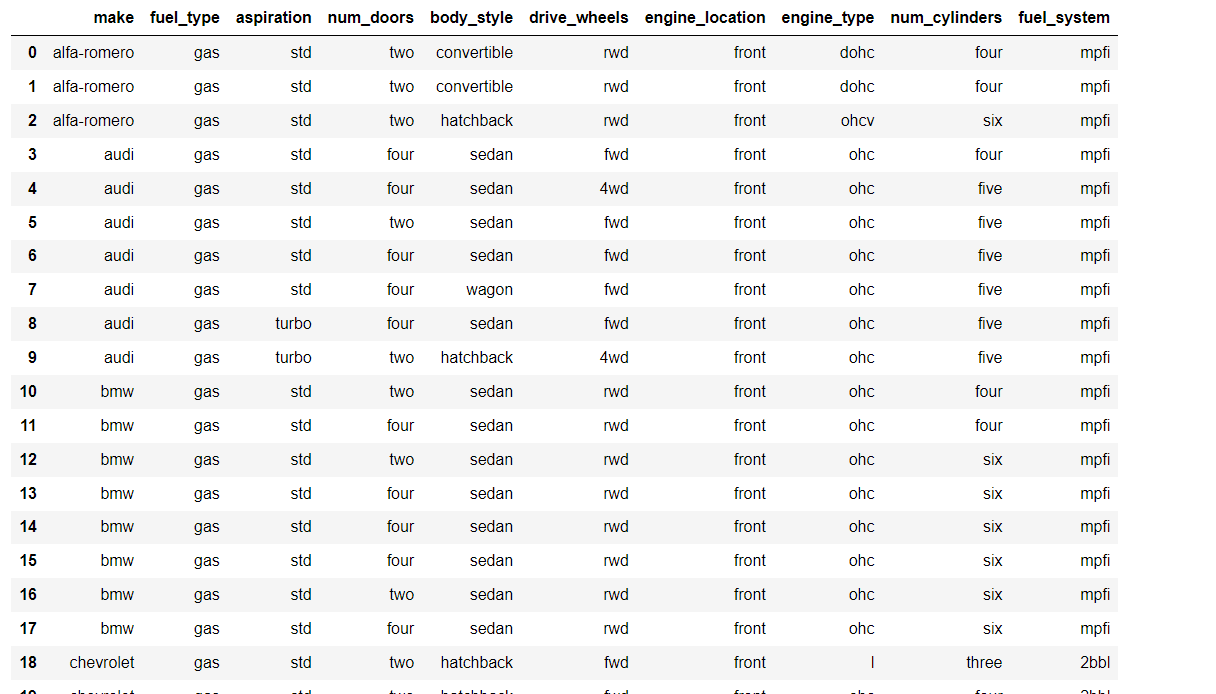


df.dtypes

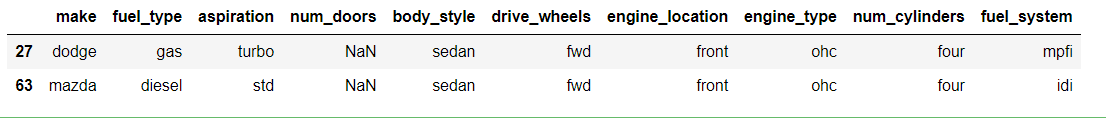


obj\_df = df.select\_dtypes(include=['object']).copy()

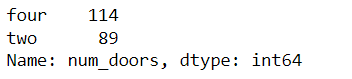
obj\_df.head(30)



obj\_df[obj\_df.isnull().any(axis=1)]



obj\_df["num\_doors"].value\_counts()

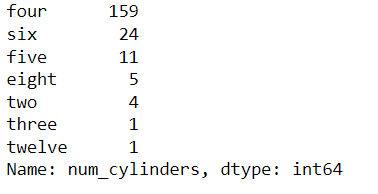


obj\_df = obj\_df.fillna({"num\_doors": "four"})

obj\_df



obj\_df["num\_cylinders"].value\_counts()



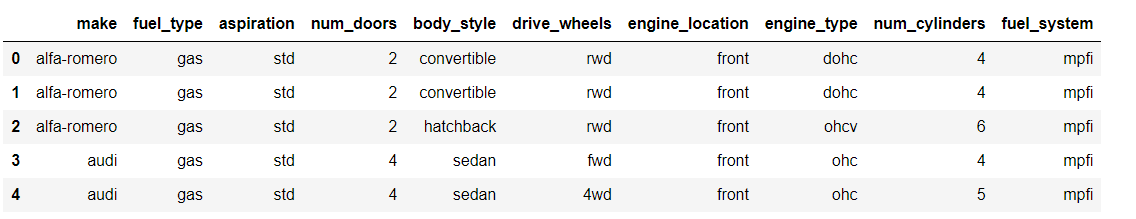
cleanup\_nums = {"num\_doors": {"four": 4, "two": 2},

"num\_cylinders": {"four": 4, "six": 6, "five": 5, "eight": 8,

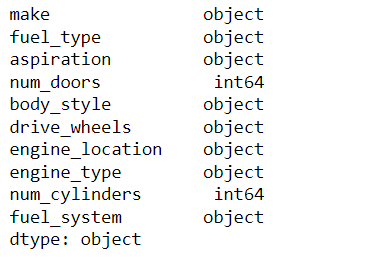
"two": 2, "twelve": 12, "three":3 }}

obj\_df = obj\_df.replace(cleanup\_nums)

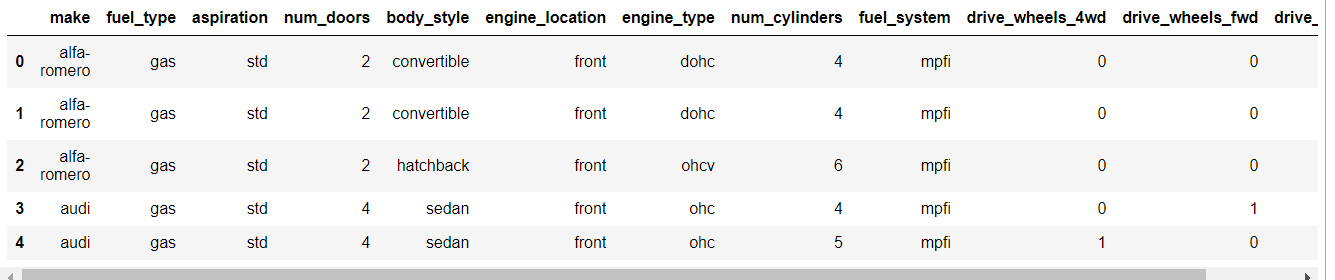
obj\_df.head()



obj\_df.dtypes



pd.get\_dummies(obj\_df, columns=["drive\_wheels"]).head()

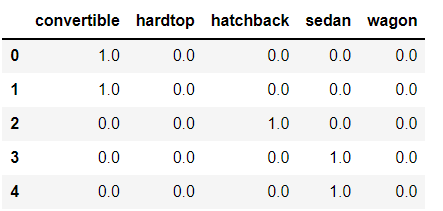


from sklearn.preprocessing import OneHotEncoder

oe\_style = OneHotEncoder()

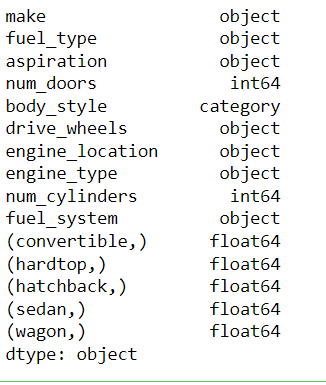
oe\_results = oe\_style.fit\_transform(obj\_df[["body\_style"]])

pd.DataFrame(oe\_results.toarray(), columns=oe\_style.categories\_).head()



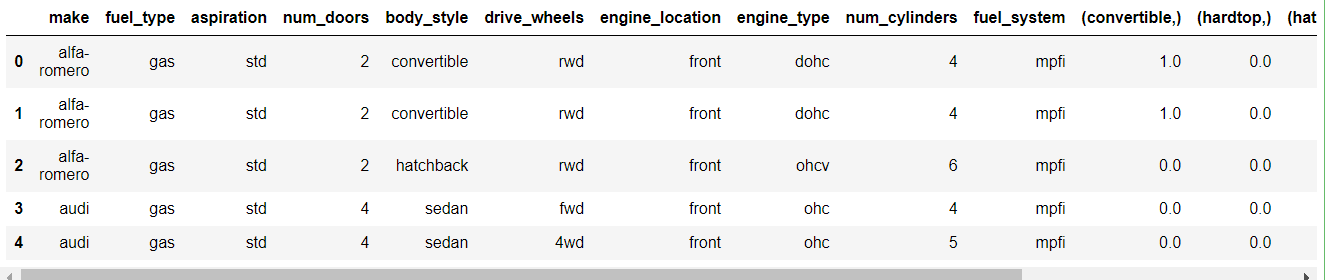
obj\_df["body\_style"] = obj\_df["body\_style"].astype('category')

obj\_df.dtypes

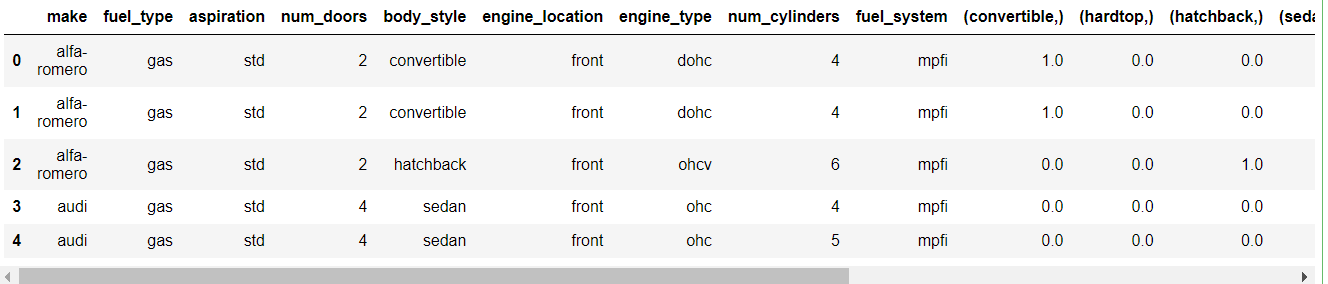


obj\_df["body\_style\_cat"] = obj\_df["body\_style"].cat.codes

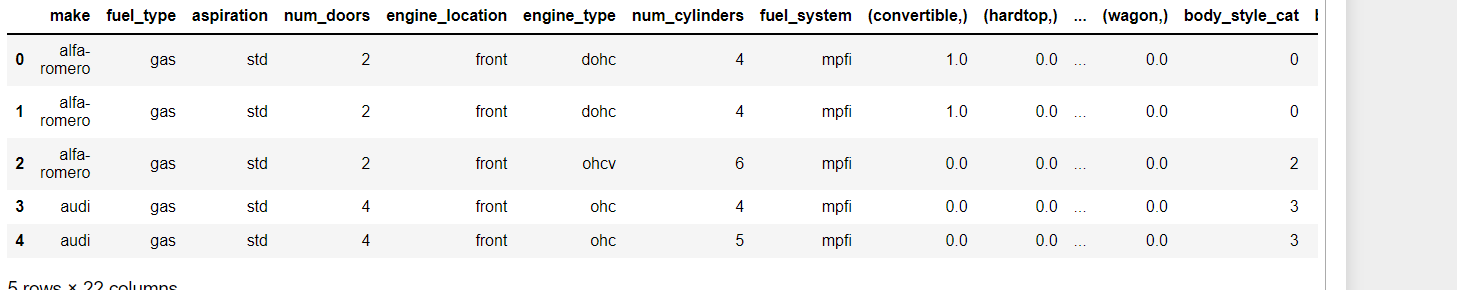
obj\_df.head()



pd.get\_dummies(obj\_df, columns=["drive\_wheels"]).head()



pd.get\_dummies(obj\_df, columns=["body\_style", "drive\_wheels"], prefix=["body", "drive"]).head()



Q.4: Write a python program to implement simple Linear Regression for predicting house price.

import numpy as np

import matplotlib.pyplot as plt

def estimate\_coef(x, y):

# number of observations/points

n = np.size(x)

# mean of x and y vector

m\_x = np.mean(x)

m\_y = np.mean(y)

# calculating cross-deviation and deviation about x

SS\_xy = np.sum(y\*x) - n\*m\_y\*m\_x

SS\_xx = np.sum(x\*x) - n\*m\_x\*m\_x

# calculating regression coefficients

b\_1 = SS\_xy / SS\_xx

b\_0 = m\_y - b\_1\*m\_x

return (b\_0, b\_1)

def plot\_regression\_line(x, y, b):

# plotting the actual points as scatter plot

plt.scatter(x, y, color = "m",

marker = "o", s = 30)

# predicted response vector

y\_pred = b[0] + b[1]\*x

# plotting the regression line

plt.plot(x, y\_pred, color = "g")

# putting labels

plt.xlabel('x')

plt.ylabel('y')

# function to show plot

plt.show()

def main():

# observations / data

x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

# estimating coefficients

b = estimate\_coef(x, y)

print("Estimated coefficients:\nb\_0 = {} \

\nb\_1 = {}".format(b[0], b[1]))

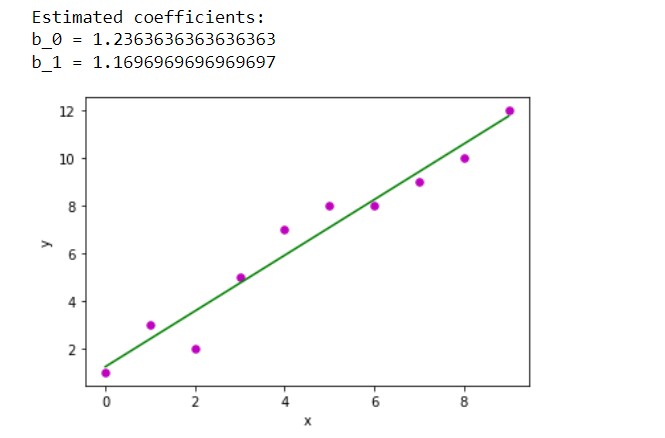
# plotting regression line

plot\_regression\_line(x, y, b)

if \_\_name\_\_ == "\_\_main\_\_":

main()

Output :-



Q.5: Write a python program to implement multiple Linear Regression for a given dataset.

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read\_csv('50\_Startups.csv')

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

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

from sklearn.compose import ColumnTransformer

from sklearn.preprocessing import OneHotEncoder

ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [3])],

remainder='passthrough')

X = np.array(ct.fit\_transform(X))

print(X)

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2)

from sklearn.linear\_model import LinearRegression

regressor = LinearRegression()

regressor.fit(X\_train, y\_train)

y\_pred = regressor.predict(X\_test)

df = pd.DataFrame({'Real Values':y\_test, 'Predicted Values':y\_pred})

df

Output :-

[[0.0 0.0 1.0 165349.2 136897.8 471784.1]

[1.0 0.0 0.0 162597.7 151377.59 443898.53]

[0.0 1.0 0.0 153441.51 101145.55 407934.54]

[0.0 0.0 1.0 144372.41 118671.85 383199.62]

[0.0 1.0 0.0 142107.34 91391.77 366168.42]

[0.0 0.0 1.0 131876.9 99814.71 362861.36]

[1.0 0.0 0.0 134615.46 147198.87 127716.82]

[0.0 1.0 0.0 130298.13 145530.06 323876.68]

[0.0 0.0 1.0 120542.52 148718.95 311613.29]

[1.0 0.0 0.0 123334.88 108679.17 304981.62]

[0.0 1.0 0.0 101913.08 110594.11 229160.95]

[1.0 0.0 0.0 100671.96 91790.61 249744.55]

[0.0 1.0 0.0 93863.75 127320.38 249839.44]

[1.0 0.0 0.0 91992.39 135495.07 252664.93]

[0.0 1.0 0.0 119943.24 156547.42 256512.92]

[0.0 0.0 1.0 114523.61 122616.84 261776.23]

[1.0 0.0 0.0 78013.11 121597.55 264346.06]

[0.0 0.0 1.0 94657.16 145077.58 282574.31]

[0.0 1.0 0.0 91749.16 114175.79 294919.57]

[0.0 0.0 1.0 86419.7 153514.11 0.0]

[1.0 0.0 0.0 76253.86 113867.3 298664.47]

[0.0 0.0 1.0 78389.47 153773.43 299737.29]

[0.0 1.0 0.0 73994.56 122782.75 303319.26]

[0.0 1.0 0.0 67532.53 105751.03 304768.73]

[0.0 0.0 1.0 77044.01 99281.34 140574.81]

[1.0 0.0 0.0 64664.71 139553.16 137962.62]

[0.0 1.0 0.0 75328.87 144135.98 134050.07]

[0.0 0.0 1.0 72107.6 127864.55 353183.81]

[0.0 1.0 0.0 66051.52 182645.56 118148.2]

[0.0 0.0 1.0 65605.48 153032.06 107138.38]

[0.0 1.0 0.0 61994.48 115641.28 91131.24]

[0.0 0.0 1.0 61136.38 152701.92 88218.23]

[1.0 0.0 0.0 63408.86 129219.61 46085.25]

[0.0 1.0 0.0 55493.95 103057.49 214634.81]

[1.0 0.0 0.0 46426.07 157693.92 210797.67]

[0.0 0.0 1.0 46014.02 85047.44 205517.64]

[0.0 1.0 0.0 28663.76 127056.21 201126.82]

[1.0 0.0 0.0 44069.95 51283.14 197029.42]

[0.0 0.0 1.0 20229.59 65947.93 185265.1]

[1.0 0.0 0.0 38558.51 82982.09 174999.3]

[1.0 0.0 0.0 28754.33 118546.05 172795.67]

[0.0 1.0 0.0 27892.92 84710.77 164470.71]

[1.0 0.0 0.0 23640.93 96189.63 148001.11]

[0.0 0.0 1.0 15505.73 127382.3 35534.17]

[1.0 0.0 0.0 22177.74 154806.14 28334.72]

[0.0 0.0 1.0 1000.23 124153.04 1903.93]

[0.0 1.0 0.0 1315.46 115816.21 297114.46]

[1.0 0.0 0.0 0.0 135426.92 0.0]

[0.0 0.0 1.0 542.05 51743.15 0.0]

[1.0 0.0 0.0 0.0 116983.8 45173.06]]

Out[1]:

|  | **Real Values** | **Predicted Values** |
| --- | --- | --- |
| **0** | 108552.04 | 113563.743391 |
| **1** | 78239.91 | 77687.491336 |
| **2** | 125370.37 | 130630.929762 |
| **3** | 166187.94 | 169855.491369 |
| **4** | 97483.56 | 98953.798984 |
| **5** | 146121.95 | 134426.858141 |
| **6** | 14681.40 | 51725.312640 |
| **7** | 152211.77 | 151561.905117 |
| **8** | 149759.96 | 155221.707849 |
| **9** | 144259.40 | 136251.235516 |

Q.6: Write a python program to implement Polynomial Linear Regression for given dataset.

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read\_csv('Position\_Salaries.csv')

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

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

dataset.head(5)

from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear\_model import LinearRegression

poly\_reg = PolynomialFeatures(degree = 4)

X\_poly = poly\_reg.fit\_transform(X)

lin\_reg = LinearRegression()

lin\_reg.fit(X\_poly, y)

y\_pred = lin\_reg.predict(X\_poly)

df = pd.DataFrame({'Real Values':y, 'Predicted Values':y\_pred})

df

X\_grid = np.arange(min(X), max(X), 0.1)

X\_grid = X\_grid.reshape((len(X\_grid), 1))

plt.scatter(X, y, color = 'red')

plt.scatter(X, y\_pred, color = 'green')

plt.plot(X\_grid,

lin\_reg.predict(poly\_reg.fit\_transform(X\_grid)), color =

'black')

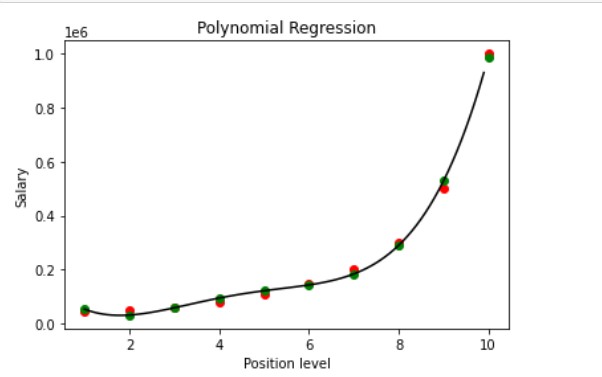
plt.title('Polynomial Regression')

plt.xlabel('Position level')

plt.ylabel('Salary')

plt.show()

Output :-



Q.7: Write a python program to Implement Naïve Bayes.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

import seaborn as sns

data=pd.read\_csv('pima-indians-diabetes.csv')

data.shape

data.isnull().sum()

data.isnull().values.any()

data.dtypes

#visualisation

data.hist()

columns=list(data)[0:-1]

data[columns].hist()

#identifty the correlation

data.corr()

sns.heatmap(data.corr(),annot=True)

sns.pairplot(data)

#calculate diabetes ratio of true or false target varible

n\_true=len(data.loc[data['class']==True])

n\_false=len(data.loc[data['class']==False])

print("No.of true cases:{0} {1}%".format(n\_true,(n\_true/(n\_true+n\_false))\*100))

print("No.of false cases:{0} {1}%".format(n\_false,(n\_false/(n\_true+n\_false))\*100))

#split the data

from sklearn.model\_selection import train\_test\_split

x=data.drop('class',axis=1)

y=data['class']

X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(x,y,test\_size=0.30,random\_state=1)

from sklearn.impute import SimpleImputer

rep\_0=SimpleImputer(missing\_values=0,strategy='mean')

cols=X\_train.columns

X\_train=pd.DataFrame(rep\_0.fit\_transform(X\_train))

X\_test=pd.DataFrame(rep\_0.fit\_transform(X\_test))

X\_train.columns=cols

X\_test.columns=cols

X\_train.head()

from sklearn.naive\_bayes import GaussianNB

diab\_mode=GaussianNB()

diab\_mode.fit(X\_train,Y\_train)

diab\_train\_predict=diab\_mode.predict(X\_train)

from sklearn import metrics

print("Model Accuracy:{0}".format(metrics.accuracy\_score(Y\_train,diab\_train\_predict)))

diab\_train\_predict=diab\_mode.predict(X\_test)

from sklearn import metrics

print("Model Accuracy:{0}".format(metrics.accuracy\_score(Y\_test,diab\_train\_predict)))

cm1=metrics.confusion\_matrix(Y\_test,diab\_train\_predict,labels=[1,0])

df\_cm1=pd.DataFrame(cm1,index=[i for i in['1','0']],columns=[i for i in['predict 1','predict o']] )

df\_cm1

Output :-

No.of true cases:268 34.89583333333333%

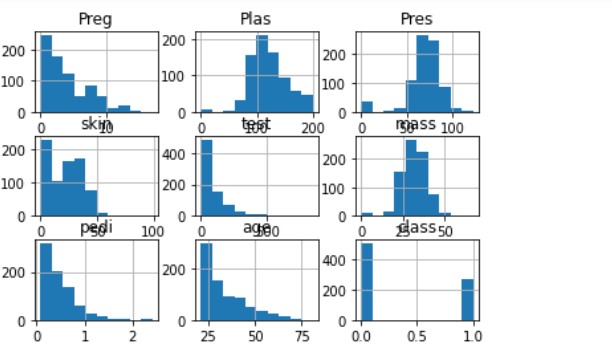
No.of false cases:500 65.10416666666666%

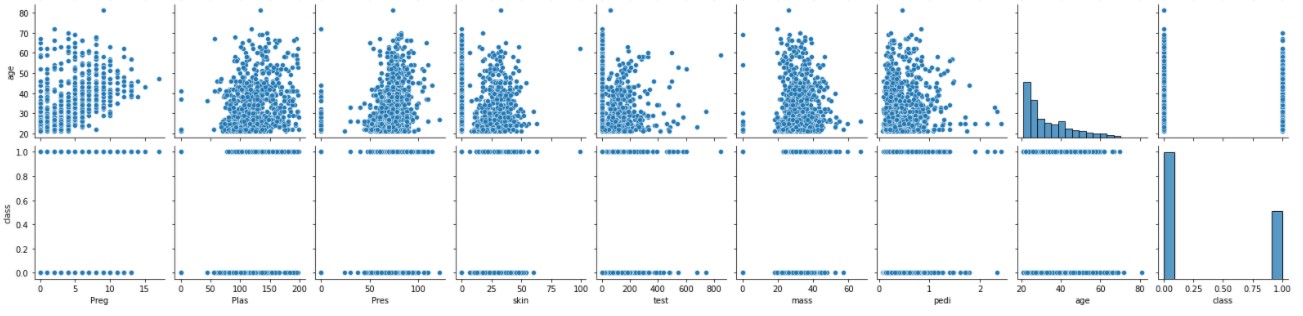
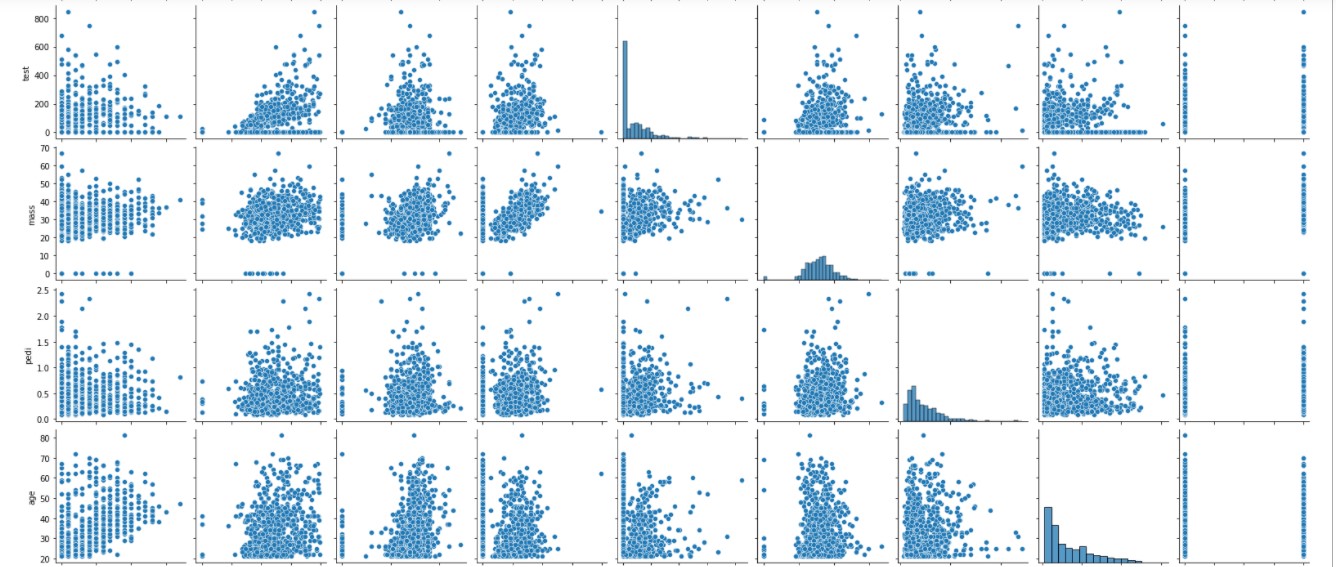
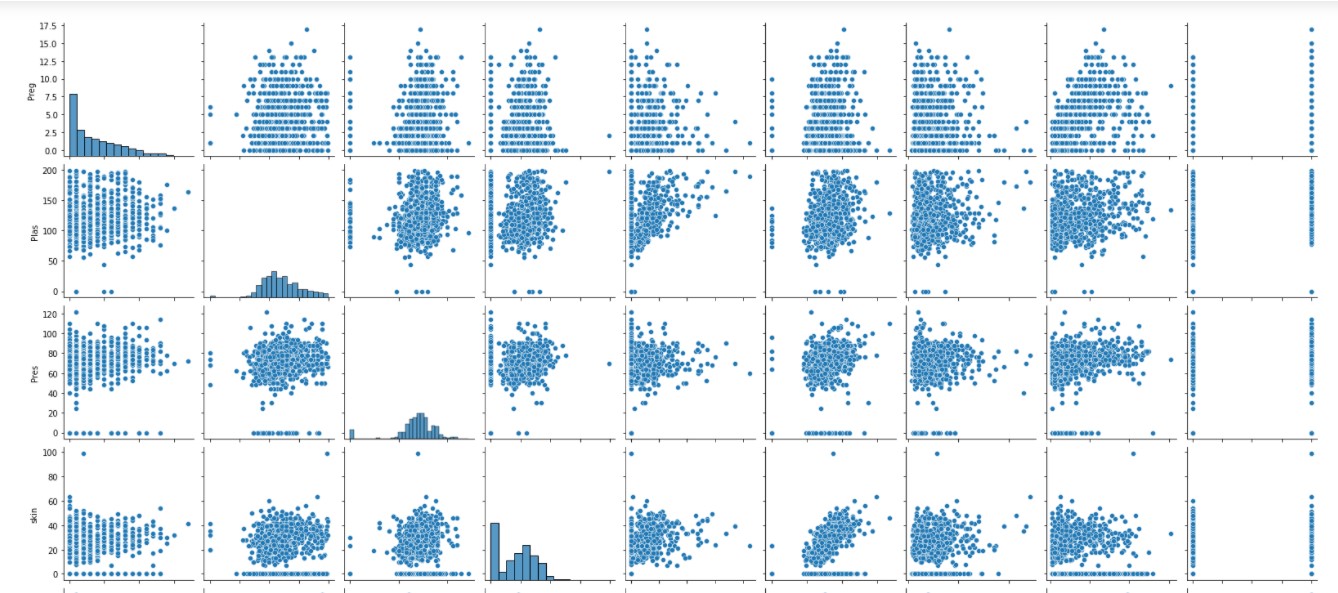
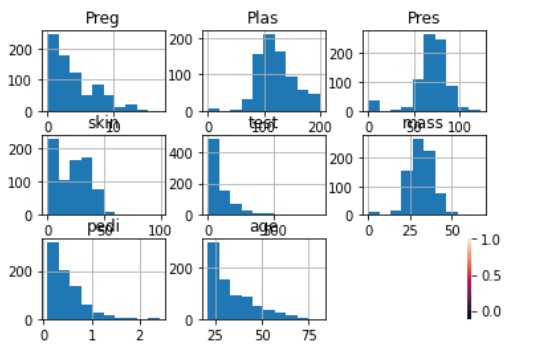
Model Accuracy:0.7392923649906891

Model Accuracy:0.7705627705627706

Out[2]:

|  | **predict 1** | **predict o** |
| --- | --- | --- |
| **1** | 55 | 30 |
| **0** | 23 | 123 |





Q.8: Write a python program to Implement Decision Tree whether or not to play tennis.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import train\_test\_split

from scipy.stats import zscore

import seaborn as sns

data=pd.read\_csv('PlayTennis.csv')

data.isnull().sum()

Output :-

outlook 0

temp 0

humidity 0

windy 0

play 0

dtype: int64

data.dtypes

Output :-

outlook object

temp object

humidity object

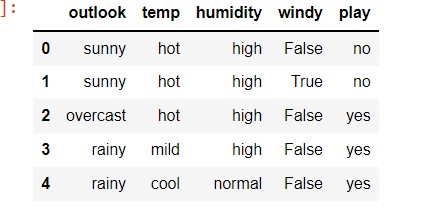
windy bool

play object

dtype: object

data.head()

Output:-



data.outlook.value\_counts()

Output :-

sunny 5

rainy 5

overcast 4

Name: outlook, dtype: int64

from sklearn.preprocessing import LabelEncoder

l=LabelEncoder()

for i in data.columns:

if data[i].dtypes=='object' or data[i].dtypes=='bool':

data[i]=pd.Categorical(data[i])

for i in data.columns:

data[i]=l.fit\_transform(data[i])

data.dtypes

Output :-

outlook int32

temp int32

humidity int32

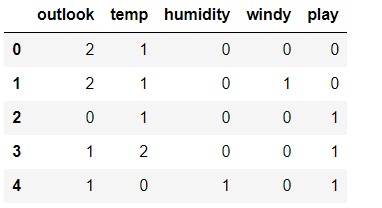
windy int32

play int32

dtype: object

data.head()

Output :-



x=data.drop(['play'],axis=1)

y=data['play']

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.30,random\_state=1)

dtree=DecisionTreeClassifier(criterion='gini',random\_state=1)

dtree.fit(x\_train,y\_train)

Output :-

DecisionTreeClassifier(random\_state=1)

In [16]:

print(dtree.score(x\_train,y\_train))#data is over fitted so we use max\_depth =5 means prunning technique

print(dtree.score(x\_test,y\_test))

Output:-

1.0

0.4

dtree1=DecisionTreeClassifier(criterion='gini',max\_depth=5,random\_state=1)

dtree1.fit(x\_train,y\_train)

Output :-

DecisionTreeClassifier(max\_depth=5, random\_state=1)

In [19]:

print(dtree1.score(x\_train,y\_train))#data is over fitted so we use max\_depth =5 means prunning technique

print(dtree1.score(x\_test,y\_test))

1.0

0.4

#but data remain overfitted

y\_predict=dtree.predict(x\_test)

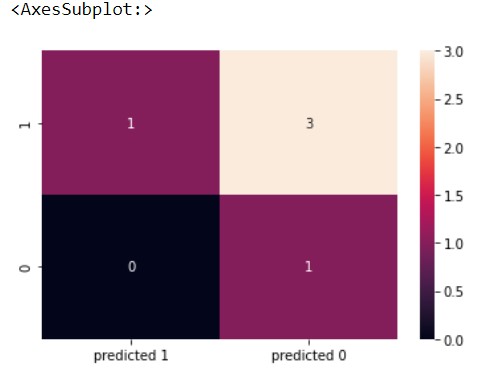
from sklearn import metrics

cm=metrics.confusion\_matrix(y\_test,y\_predict,labels=[1,0])

df\_cm=pd.DataFrame(cm,index=[i for i in['1','0']],columns=[i for i in['predicted 1','predicted 0']] )

df\_cm

sns.heatmap(df\_cm,annot=True)



from sklearn.metrics import classification\_report

m=classification\_report(y\_test,y\_predict)

print(m)

Output :-

precision recall f1-score support

0 0.25 1.00 0.40 1

1 1.00 0.25 0.40 4

accuracy 0.40 5

macro avg 0.62 0.62 0.40 5

weighted avg 0.85 0.40 0.40 5

Q.9: Write a python program to implement linear SVM.

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

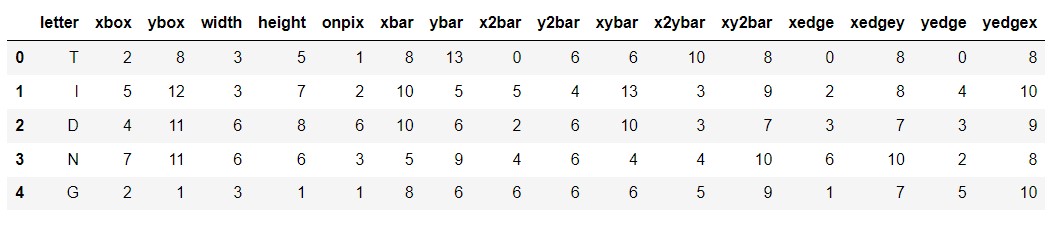
from sklearn.model\_selection import train\_test\_split

from sklearn import svm

data=pd.read\_csv('letterdata.csv')

data.head()

Output :-



def getaccuracy(testset,prediction):

correct=0

for x in range(len(testset)):

if(testset[x]==prediction[x]):

correct=correct+1

return (correct/float(len(testset)))\*100.0

data.isnull().values.any()

Output :-

False

#split data

X,Y=np.array(data)[:,1:16],np.array(data.letter)[:]

X\_train=X[:16000,:]

X\_test=X[16001:,:]

Y\_train=Y[:16000]

Y\_test=Y[16001:]

#build the model

clf=svm.SVC(gamma=0.025,C=3)

clf.fit(X\_train,Y\_train)

Output :-

SVC(C=3, gamma=0.025)

Y\_predict=clf.predict(X\_test)

getaccuracy(Y\_test,Y\_predict)

Output :-

96.07401850462615

y\_g=(np.column\_stack([Y\_test,Y\_predict]))

print(y\_g)

[['N' 'N']

['V' 'V']

['I' 'I']

...

['T' 'T']

['S' 'S']

['A' 'A']]

import numpy as np

import matplotlib.pyplot as plt

from sklearn import svm, datasets

# import some data to play with

iris = datasets.load\_iris()

X = iris.data[:, :2] # we only take the first two features. We could

# avoid this ugly slicing by using a two-dim dataset

y = iris.target

# we create an instance of SVM and fit out data. We do not scale our

# data since we want to plot the support vectors

C = 1.0 # SVM regularization parameter

svc = svm.SVC(kernel='linear', C=1,gamma=10).fit(X, y)

# create a mesh to plot in

x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1

y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1

h = (x\_max / x\_min)/100

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h),

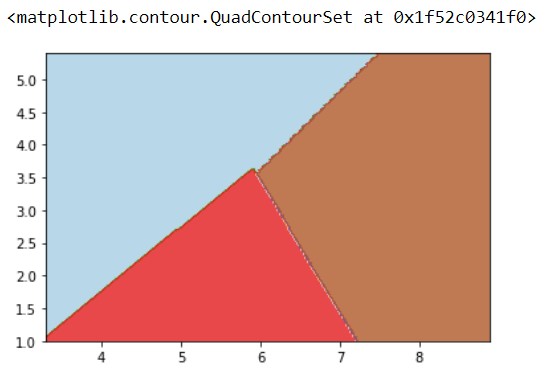
np.arange(y\_min, y\_max, h))

plt.subplot(1, 1, 1)

Z = svc.predict(np.c\_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)



plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired)

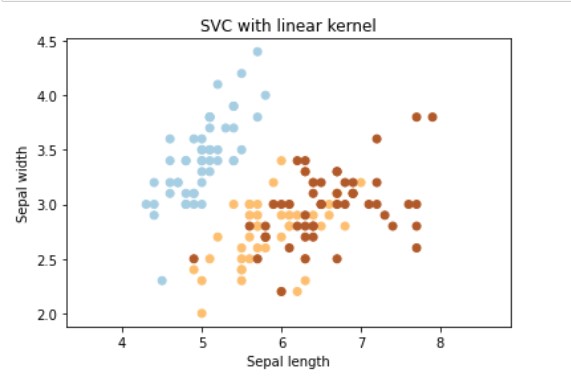
plt.xlabel('Sepal length')

plt.ylabel('Sepal width')

plt.xlim(xx.min(), xx.max())

plt.title('SVC with linear kernel')

plt.show()



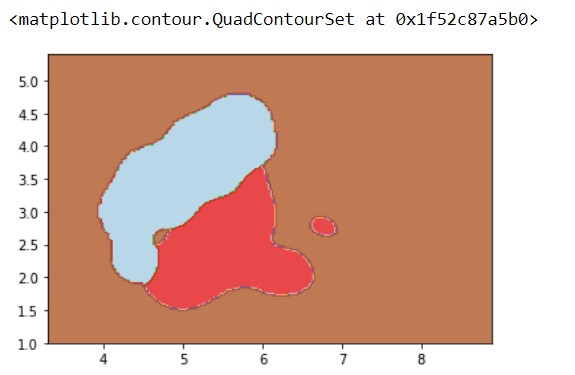
svc = svm.SVC(kernel='rbf', C=1,gamma=10).fit(X, y)

plt.subplot(1, 1, 1)

Z = svc.predict(np.c\_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)



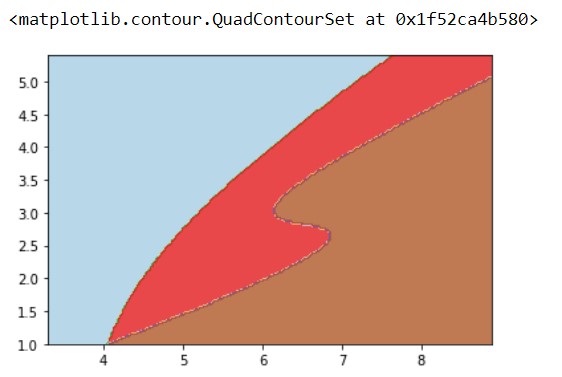
svc = svm.SVC(kernel='poly', C=1,gamma=100).fit(X, y)

plt.subplot(1, 1, 1)

Z = svc.predict(np.c\_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)



Q.10: Write a python program to find Decision boundary by using a neural network with 10 hidden units on two moons dataset

import numpy as np

from sklearn.datasets import make\_moons

from sklearn.model\_selection import train\_test\_split

import matplotlib.pyplot as plt

np.random.seed(0)

X, Y = make\_moons(500, noise=0.1)

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(

X, Y, test\_size=0.25, random\_state=73)

plt.figure(figsize=(12,8))

plt.scatter(X\_train[:,0], X\_train[:,1], c=Y\_train,

cmap=plt.cm.cividis, s=50)

plt.xlabel('X1')

plt.ylabel('X2')

plt.title('Random Training Data')

plt.show()

n\_inputs = X\_train.shape[0]

n\_inputs\_dim = X\_train.shape[1]

n\_h = 4 # Number of hidden nodes

n\_out = 1 # Number of output nodes = for binary classifier

c = np.sqrt(3 / (0.5 + n\_inputs\_dim + n\_out))

# Initialize weights and bias

W1 = np.random.uniform(low=-c, high=c,

size=(n\_inputs\_dim, n\_h))

b1 = np.zeros((1, n\_h))

W2 = np.random.uniform(low=-c, high=c,

size=(n\_h, n\_out))

b2 = np.zeros((1, n\_out))

def Elu(x, a=2):

"""

Compute the ELU output of x

"""

return np.where(x<=0, a \* (np.exp(x) - 1), x)

def sigmoid(x):

"""

Compute sigmoid of array x

"""

return 1 / (1 + np.exp(-x))

Z1 = np.dot(X\_train, W1) + b1

A1 = Elu(Z1)

Z2 = np.dot(A1, W2) + b2

A2 = sigmoid(Z2)

Y\_train = Y\_train.reshape(-1,1)

log\_probs = (np.multiply(np.log(A2), Y\_train) +

np.multiply(np.log(1 - A2), (1 - Y\_train)))

loss = -1 / n\_inputs \* np.sum(log\_probs)

loss

def dElu(x, a=2):

return np.where(x<=0,a \* np.exp(x), 0)

m = 1 / n\_inputs

dZ2 = A2 - Y\_train

dW2 = m \* np.dot(A1.T, dZ2)

db2 = m \* np.sum(dZ2, axis=0, keepdims=True)

dZ1 = m \* np.dot(dZ2, W2.T) \* dElu(Z1)

dW1 = m \* np.dot(X\_train.T, dZ1)

db1 = m \* np.sum(dZ1, axis=0, keepdims=True)

learning\_rate = 0.01

W2 -= learning\_rate \* dW2

b2 -= learning\_rate \* db2

W1 -= learning\_rate \* dW1

b1 -= learning\_rate \* db1

class network():

def \_\_init\_\_(self, X, Y):

self.X = X

self.Y = Y

self.n\_input\_dim = X.shape[1]

self.n\_output\_dim = 1

self.n\_inputs = X.shape[0]

def initialize(self, n\_hidden, seed=1):

self.n\_hidden = n\_hidden

np.random.seed(seed)

c = np.sqrt(3 / (0.5 + self.n\_input\_dim + self.n\_output\_dim))

W1 = np.random.uniform(low=-c, high=c,

size=(self.n\_input\_dim, self.n\_hidden))

b1 = np.zeros((1, self.n\_hidden))

W2 = np.random.uniform(low=-c, high=c,

size=(self.n\_hidden, self.n\_output\_dim))

b2 = np.zeros((1, self.n\_output\_dim))

self.params = {'W1': W1, 'b1': b1, 'W2': W2, 'b2': b2}

self.cache = {}

def Elu(self, x, a=2):

return np.where(x<=0, (a \* (np.exp(x) - 1)), x)

def dElu(self, x, a=2):

return np.where(x<=0, a \* np.exp(x), 1)

def sigmoid(self, x):

return 1. / (1 + np.exp(-x))

def forward\_prop(self, X=None, cache=True):

W1 = self.params['W1']

b1 = self.params['b1']

W2 = self.params['W2']

b2 = self.params['b2']

if X is None:

X = self.X.copy()

Z1 = X.dot(W1) + b1

A1 = self.Elu(Z1)

Z2 = np.dot(A1, W2) + b2

A2 = self.sigmoid(Z2)

probs = A2

if cache:

self.cache = {'Z1': Z1, 'A1': A1, 'Z2': Z2,

'A2': A2, 'probs': probs}

else:

return probs

def back\_prop(self):

# Import parameters and cached values

A1 = self.cache['A1']

A2 = self.cache['A2']

W1 = self.params['W1']

b1 = self.params['b1']

W2 = self.params['W2']

b2 = self.params['b2']

# Calculate derivatives

m = 1 / self.n\_inputs

dZ2 = A2 - self.Y.reshape(-1,1)

dW2 = m \* A1.T.dot(dZ2)

db2 = m\* np.sum(dZ2, axis=0, keepdims=True)

dZ1 = m \* dZ2.dot(W2.T) \* self.dElu(A1)

dW1 = m \* np.dot(self.X.T, dZ1)

db1 = m \* np.sum(dZ1, axis=0)

# Apply gradient descent updates

W1 -= self.learning\_rate \* dW1

b1 -= self.learning\_rate \* db1

W2 -= self.learning\_rate \* dW2

b2 -= self.learning\_rate \* db2

# Store updated network parameters

self.params = {'W1': W1, 'b1': b1,

'W2': W2, 'b2': b2}

def train(self, learning\_rate=1e-2,

n\_iters=10000, log\_loss=False):

self.learning\_rate = learning\_rate

loss = []

# Train the network

for i in range(n\_iters):

self.forward\_prop()

self.back\_prop()

# Calculate the loss value to track progress

if log\_loss:

loss.append(self.calculate\_loss())

if log\_loss:

return loss

def predict(self, X):

probs = self.forward\_prop(X, cache=False)

return np.where(probs<0.5,0,1)

def calculate\_loss(self):

probs = self.cache['probs']

W1 = self.params['W1']

W2 = self.params['W2']

Y = self.Y.reshape(-1,1)

loss = (np.multiply(np.log(probs), Y) +

np.multiply(np.log(1 - probs), (1 - Y)))

return -1 / self.n\_inputs \* np.sum(loss)

def train\_accuracy(self):

probs = self.cache['probs']

clf = np.where(probs<0.5, 0, 1)

return np.sum(self.Y.reshape(-1,1)==clf) / self.n\_inputs

# Call this function to view the decision boundary

def plot\_decision\_boundary(self):

# Determine grid range in x and y directions

x\_min, x\_max = self.X[:, 0].min()-0.1, self.X[:, 0].max()+0.1

y\_min, y\_max = self.X[:, 1].min()-0.1, self.X[:, 1].max()+0.1

# Set grid spacing parameter

spacing = min(x\_max - x\_min, y\_max - y\_min) / 100

# Create grid

XX, YY = np.meshgrid(np.arange(x\_min, x\_max, spacing),

np.arange(y\_min, y\_max, spacing))

# Concatenate data to match input

data = np.hstack((XX.ravel().reshape(-1,1),

YY.ravel().reshape(-1,1)))

# Pass data to predict method

clf = self.predict(data)

Z = clf.reshape(XX.shape)

plt.figure(figsize=(10,8))

plt.contourf(XX, YY, Z, cmap=plt.cm.Spectral, alpha=0.8)

plt.scatter(self.X[:,0], self.X[:,1], c=self.Y,

cmap=plt.cm.cividis, s=50)

plt.show()

# In[14]:

net = network(X\_train, Y\_train)

net.initialize(n\_hidden=3)

training\_loss = net.train(learning\_rate=0.1, log\_loss=True)

plt.figure(figsize=(12,8))

plt.plot(training\_loss)

plt.title("Network Training Loss")

plt.xlabel("Number of Training Iterations")

plt.ylabel("Cross-Entropy Loss")

plt.show()

print("Training Accuracy: %.2f" %net.train\_accuracy())

pred = net.predict(X\_test)

np.sum(Y\_test.reshape(-1,1)==pred) / len(Y\_test)

net.plot\_decision\_boundary()

n\_hidden = [10]

for n in n\_hidden:

net = network(X, Y)

net.initialize(n\_hidden=n)

net.train(learning\_rate=0.1, log\_loss=False)

print("%d Hidden Nodes" %n)

net.plot\_decision\_boundary()

Q.11: Write a python program to transform data with Principal Component Analysis (PCA)

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

get\_ipython().run\_line\_magic('matplotlib', 'inline')

import seaborn as sns

from sklearn.linear\_model import LinearRegression

from sklearn.decomposition import PCA

# In[9]:

data=pd.read\_csv('auto-mpg (1).csv')

# In[10]:

data.head()

# In[11]:

#drop non numeric datas

data=data.drop(['car name','origin'],axis=1)

# In[12]:

data.head()

# In[13]:

#dealing with missing values

# In[15]:

hpisdigit=pd.DataFrame(data.horsepower.str.isdigit())

# In[16]:

data[hpisdigit['horsepower']==False]

# In[17]:

data=data.replace('?',np.nan)

# In[18]:

data[hpisdigit['horsepower']==False]

# In[19]:

data.median()

# In[22]:

medianfiller=lambda x:x.fillna(x.median())

# In[23]:

data=data.apply(medianfiller,axis=0)

# In[24]:

data['horsepower']=data['horsepower'].astype('float64')

# In[25]:

x=data.drop(['mpg'],axis=1)

y=data[['mpg']]

# In[26]:

sns.pairplot(x)

# In[27]:

from scipy.stats import zscore

Xscaled=x.apply(zscore)

# In[28]:

Xscaled.head()

# In[29]:

Xscaled.head()

# In[30]:

print(covmatrix)

# In[63]:

pca=PCA(n\_components=5)

pca.fit(Xscaled)

# In[64]:

#eigen values

# In[65]:

print(pca.explained\_variance\_)

# In[66]:

#eigen vecltor

# In[67]:

print(pca.components\_)

# In[68]:

print(pca.explained\_variance\_ratio\_)

# In[69]:

xpca=pca.transform(Xscaled)

# In[70]:

regression\_model=LinearRegression()

# In[71]:

regression\_model.fit(Xscaled,y)

# In[72]:

regression\_model.score(Xscaled,y)

# In[73]:

regression\_model\_pca=LinearRegression()

regression\_model\_pca.fit(xpca,y)

# In[74]:

regression\_model\_pca.score(xpca,y)

# In[75]:

#take componets =3 to reducesa diamention

pca=PCA(n\_components=3)

pca.fit(Xscaled)

# In[76]:

#eigen values

# In[77]:

print(pca.explained\_variance\_)

# In[78]:

#eigen vecltor

# In[79]:

print(pca.components\_)

# In[80]:

print(pca.explained\_variance\_ratio\_)

# In[81]:

xpca=pca.transform(Xscaled)

# In[82]:

regression\_model=LinearRegression()

# In[83]:

regression\_model.fit(Xscaled,y)

# In[84]:

regression\_model.score(Xscaled,y)

# In[85]:

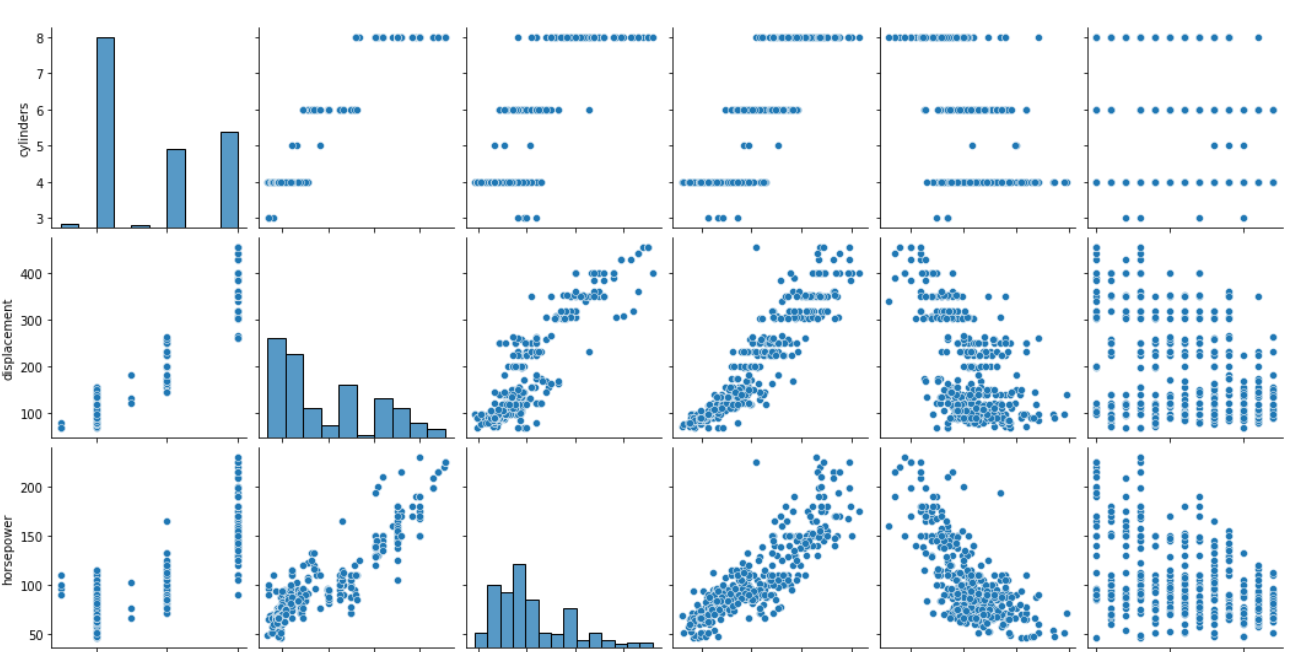
regression\_model\_pca=LinearRegression()

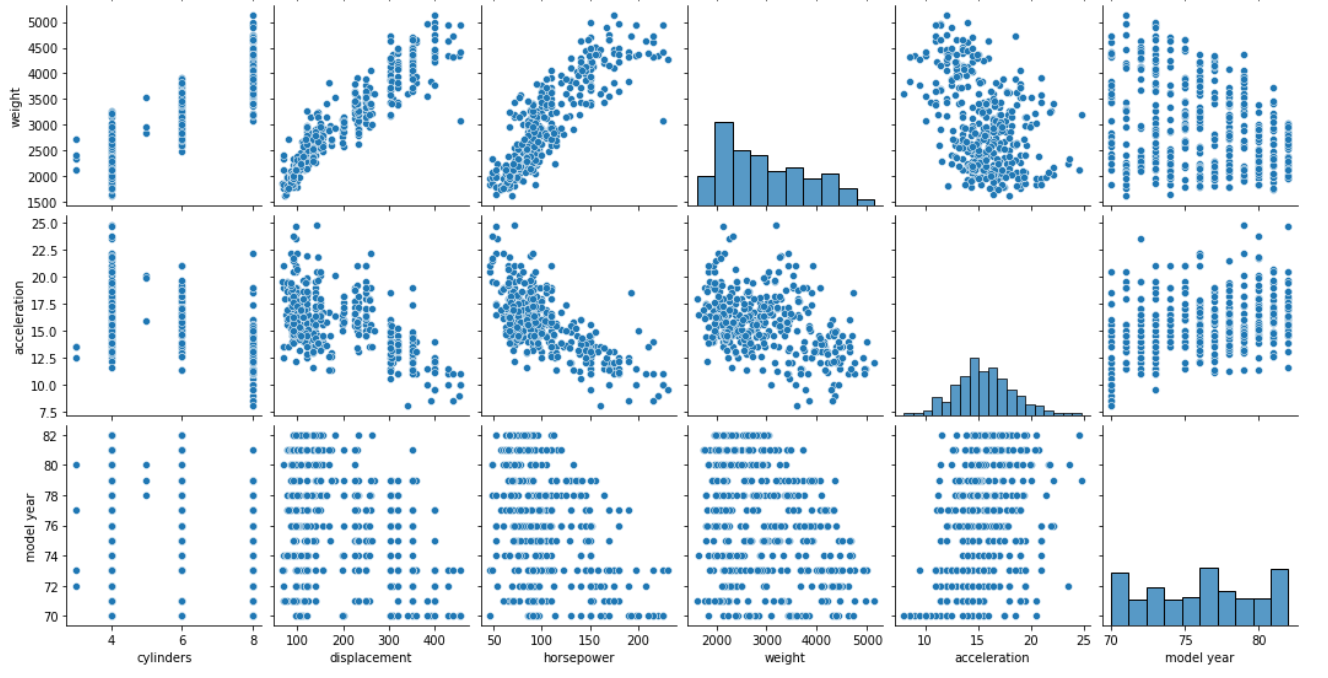
regression\_model\_pca.fit(xpca,y)

# In[86]:

regression\_model\_pca.score(xpca,y)

# In[ ]:





Q.12: Write a python program to implement k-nearest Neighbors ML algorithm to build

prediction model (Use Forge Dataset)

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

get\_ipython().run\_line\_magic('matplotlib', 'inline')

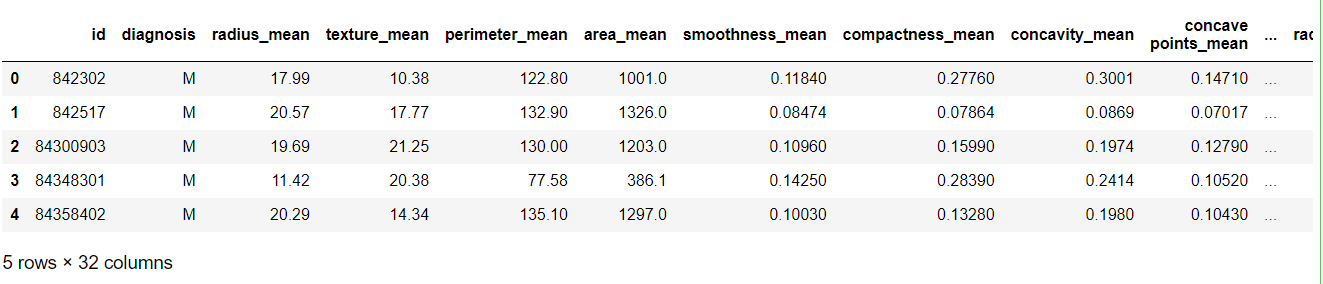
from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from scipy.stats import zscore

data=pd.read\_csv('wisc\_bc\_data.csv')

data.head()



data.shape

#569 r and 32 col

(569, 32)

data.dtypes

id int64

diagnosis object

radius\_mean float64

texture\_mean float64

perimeter\_mean float64

area\_mean float64

smoothness\_mean float64

compactness\_mean float64

concavity\_mean float64

concave points\_mean float64

symmetry\_mean float64

fractal\_dimension\_mean float64

radius\_se float64

texture\_se float64

perimeter\_se float64

area\_se float64

smoothness\_se float64

compactness\_se float64

concavity\_se float64

concave points\_se float64

symmetry\_se float64

fractal\_dimension\_se float64

radius\_worst float64

texture\_worst float64

perimeter\_worst float64

area\_worst float64

smoothness\_worst float64

compactness\_worst float64

concavity\_worst float64

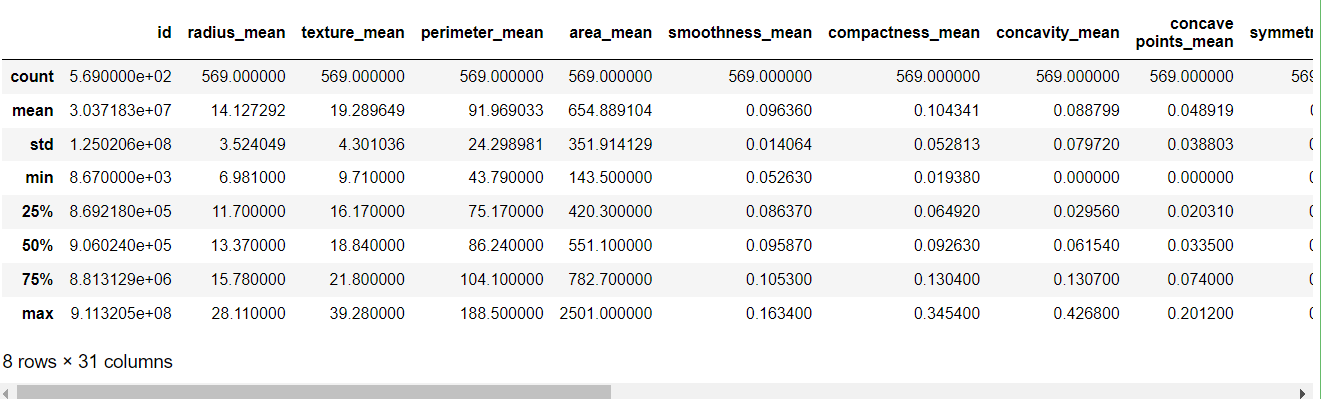
concave points\_worst float64

symmetry\_worst float64

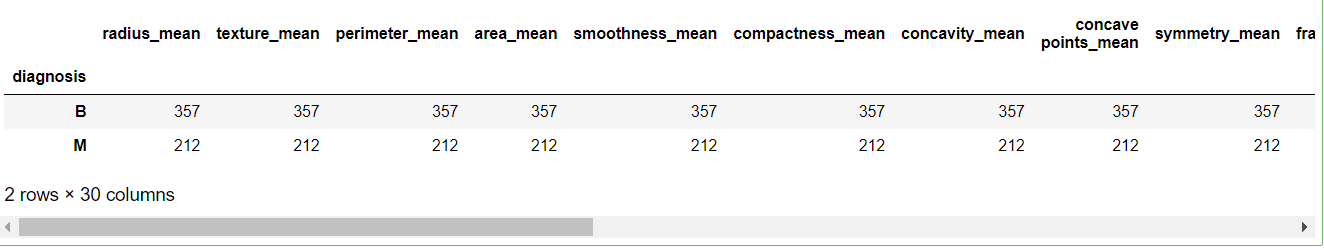
fractal\_dimension\_w

data['diagnosis']=data.diagnosis.astype('category')

data.describe()



data.groupby(['diagnosis']).count()



data=data.drop(labels='id',axis=1)

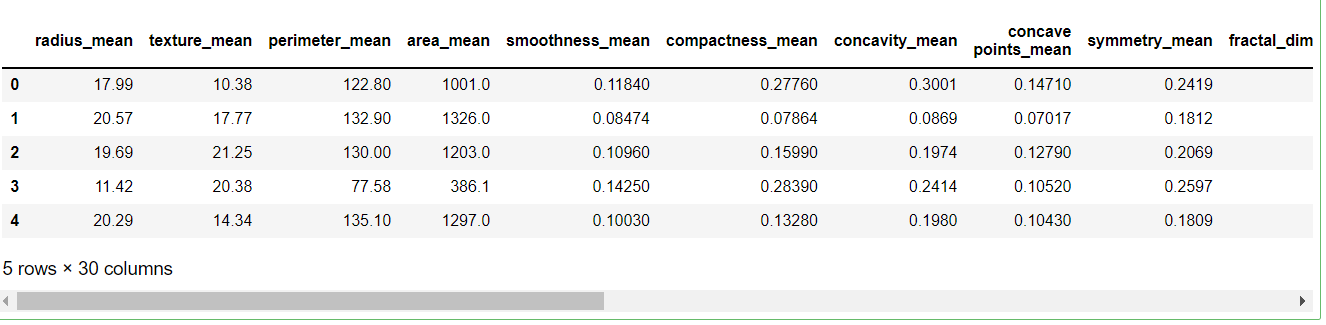
data.shape

data.head()

x=data.drop(labels='diagnosis',axis=1)

y=data['diagnosis']

x.head()



Xsacled=x.apply(zscore)

X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(Xsacled,y,test\_size=0.30,random\_state=1)

NNH= KNeighborsClassifier(n\_neighbors=5,weights='distance')

NNH.fit(X\_train,Y\_train)

Output :-

KNeighborsClassifier(weights='distance')

In [20]:

predicted\_labels=NNH.predict(X\_test)

NNH.score(X\_test,Y\_test)

Output :-

0.9532163742690059

In [21]:

from sklearn import metrics

print("confusion matrix")

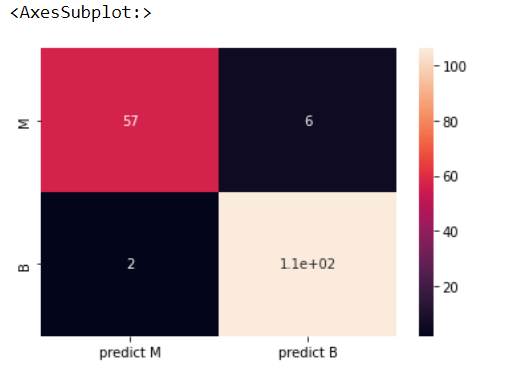
cm=metrics.confusion\_matrix(Y\_test,predicted\_labels,labels=['M','B'])

df\_cm=pd.DataFrame(cm,index=[i for i in['M','B']],columns=[i for i in['predict M','predict B']] )

confusion matrix

import seaborn as sns

sns.heatmap(df\_cm,annot=True)



scores=[]

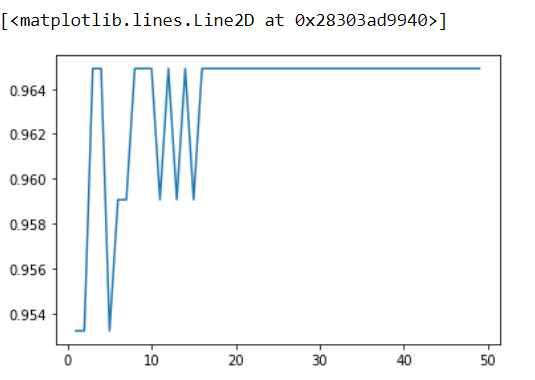
for k in range(1,50):

NNH=KNeighborsClassifier(n\_neighbors=k,weights='distance')

NNH.fit(X\_train,Y\_train)

scores.append(NNH.score(X\_test,Y\_test))

plt.plot(range(1,50),scores)



Q.13: Write a python program to implement k-means algorithm on a synthetic dataset.

import numpy as np

import pandas as pd

from sklearn.cluster import KMeans

from sklearn.model\_selection import train\_test\_split

import seaborn as sns

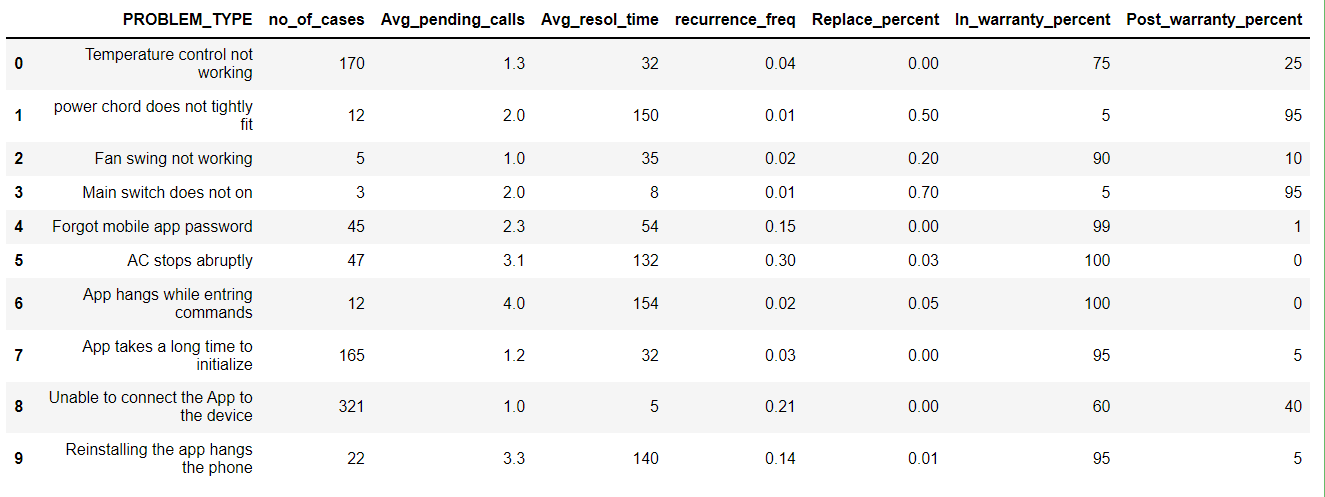
import matplotlib.pyplot as plt

%matplotlib inline

from scipy.stats import zscore

data=pd.read\_csv('technical\_support\_data-2.csv')

data.head(10)



data.dtypes

PROBLEM\_TYPE object

no\_of\_cases int64

Avg\_pending\_calls float64

Avg\_resol\_time int64

recurrence\_freq float64

Replace\_percent float64

In\_warranty\_percent int64

Post\_warranty\_percent int64

dtype: object

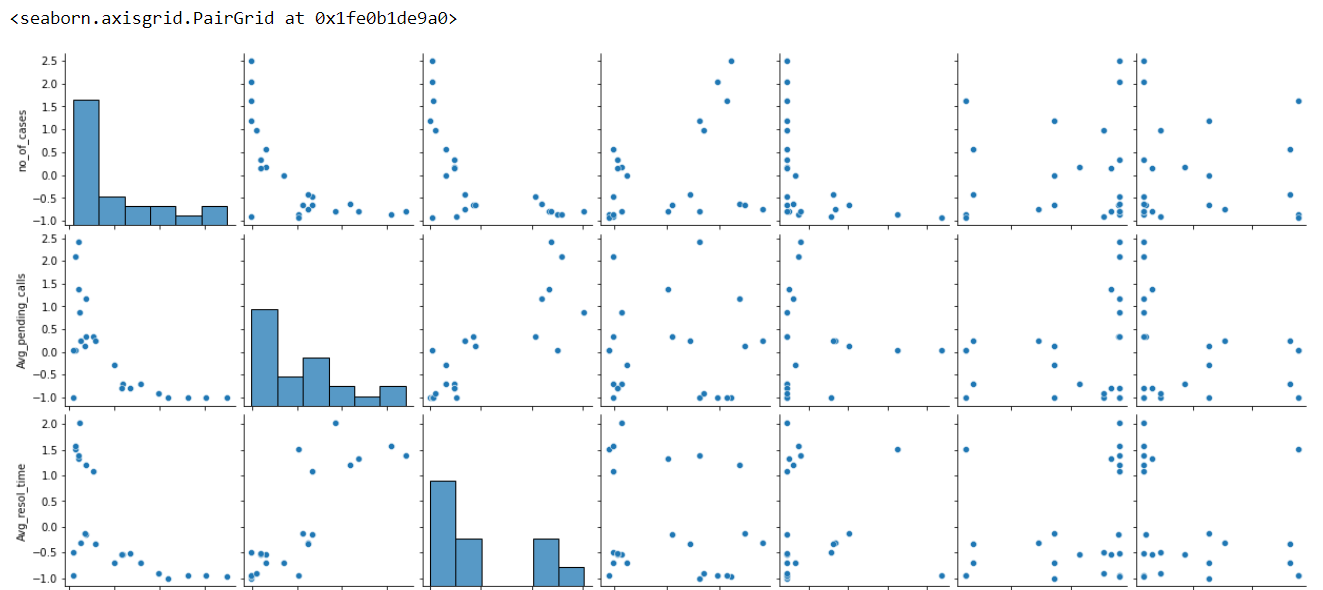
data.shape

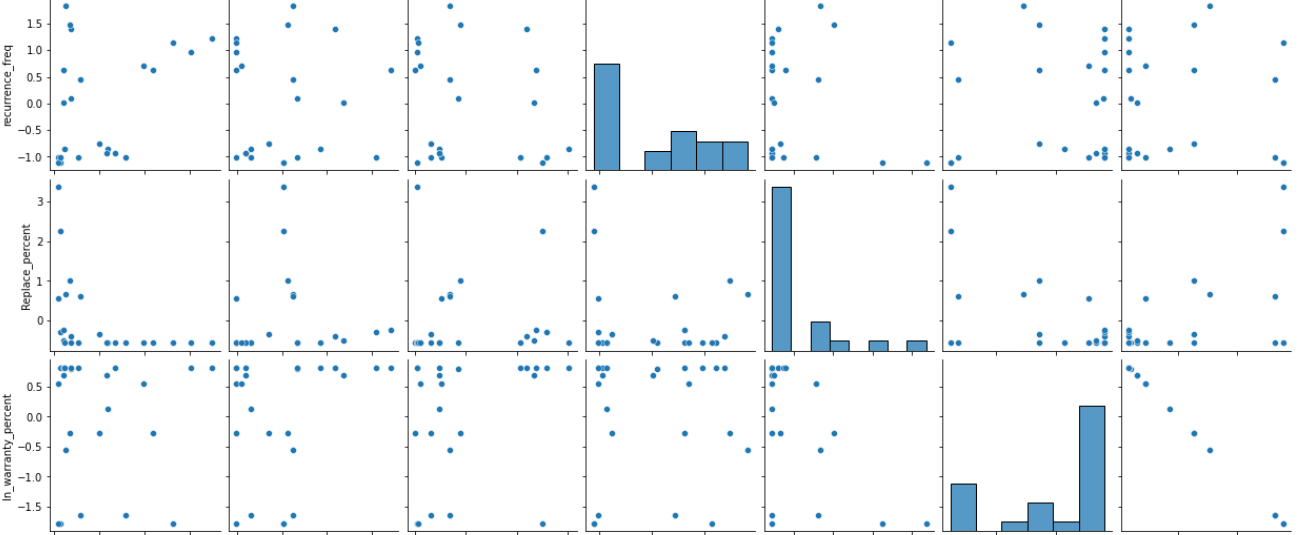
(23, 8)

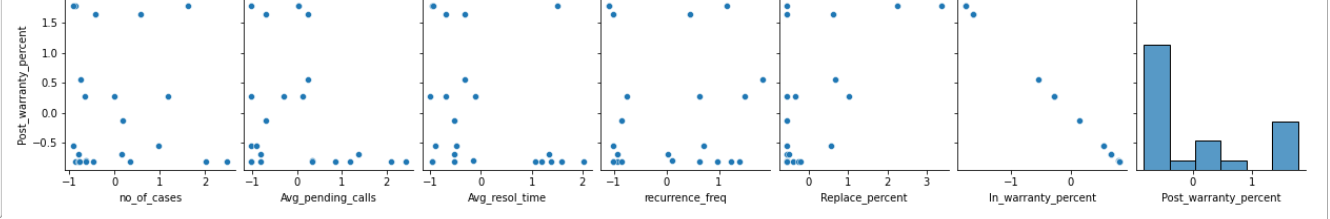
techsupport=data.iloc[:,1:]

techsupportscaled=techsupport.apply(zscore)

sns.pairplot(techsupportscaled)







#group data to similar cluster

from scipy.spatial.distance import cdist

clusters=range(1,10)

meanDistortions=[]

for k in clusters:

model=KMeans(n\_clusters=k)

model.fit(techsupportscaled)

prediction=model.predict(techsupportscaled)

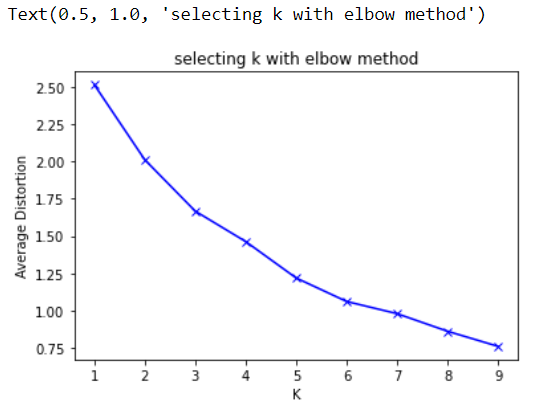
meanDistortions.append(sum(np.min(cdist(techsupportscaled,model.cluster\_centers\_,'euclidean'),axis=1))/techsupportscaled.shape[0])

plt.plot(clusters,meanDistortions,'bx-')

plt.xlabel('K')

plt.ylabel('Average Distortion')

plt.title('selecting k with elbow method')



#lets k=3 to build model

final\_model=KMeans(3)

final\_model.fit(techsupportscaled)

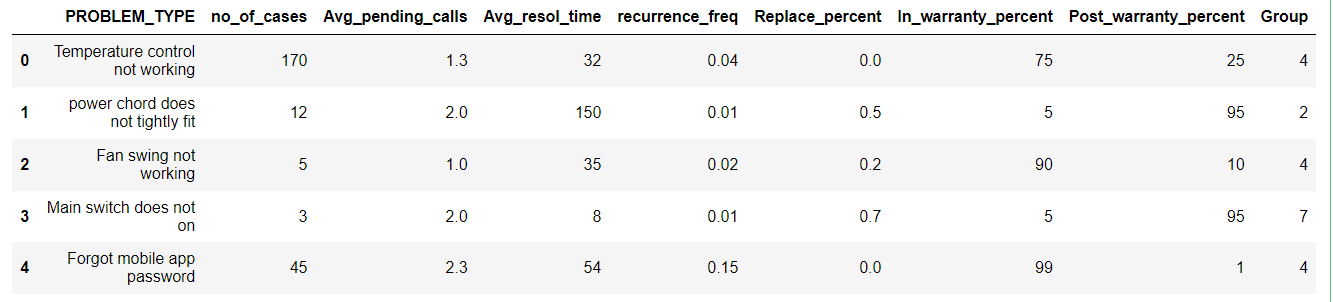
predication=final\_model.predict(techsupportscaled)

data['Group']=prediction

techsupportscaled['Group']=prediction

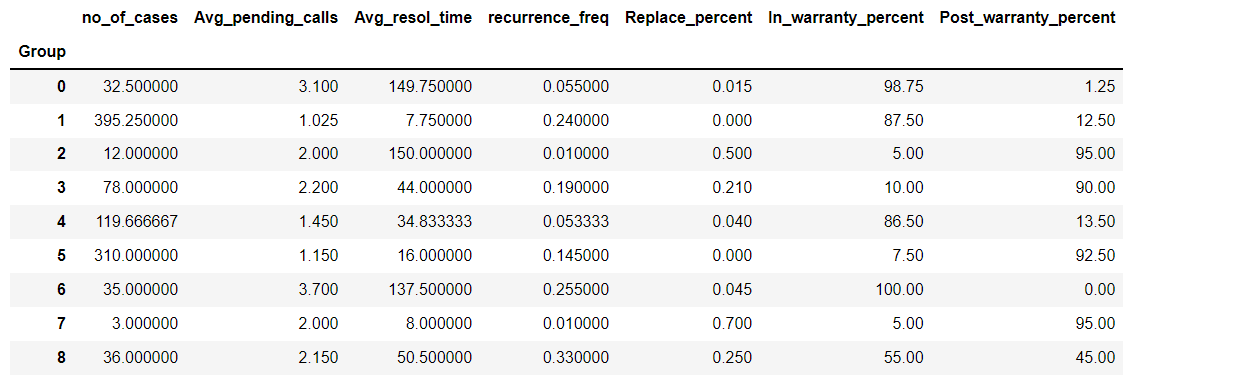
print("Groups assigned")

data.head()



techsuppclust=data.groupby(['Group'])

techsuppclust.mean()



#for k=5

final\_model=KMeans(5)

final\_model.fit(techsupportscaled)

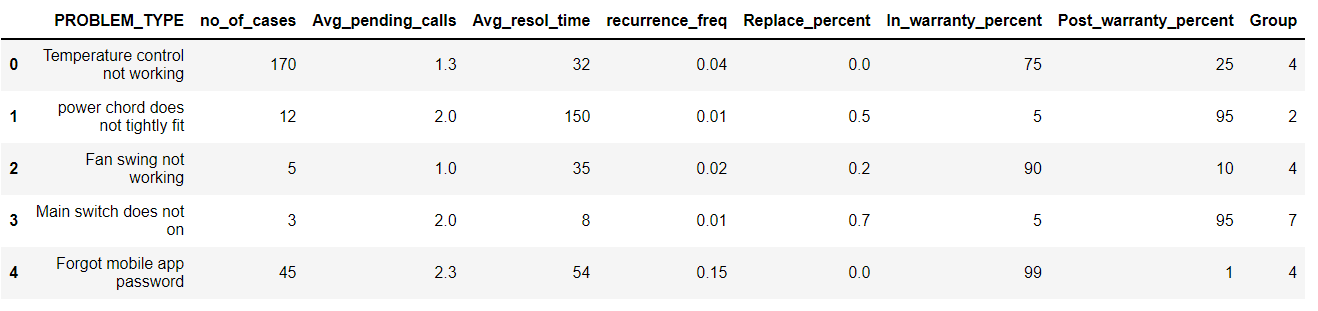
predication=final\_model.predict(techsupportscaled)

data['Group']=prediction

techsupportscaled['Group']=prediction

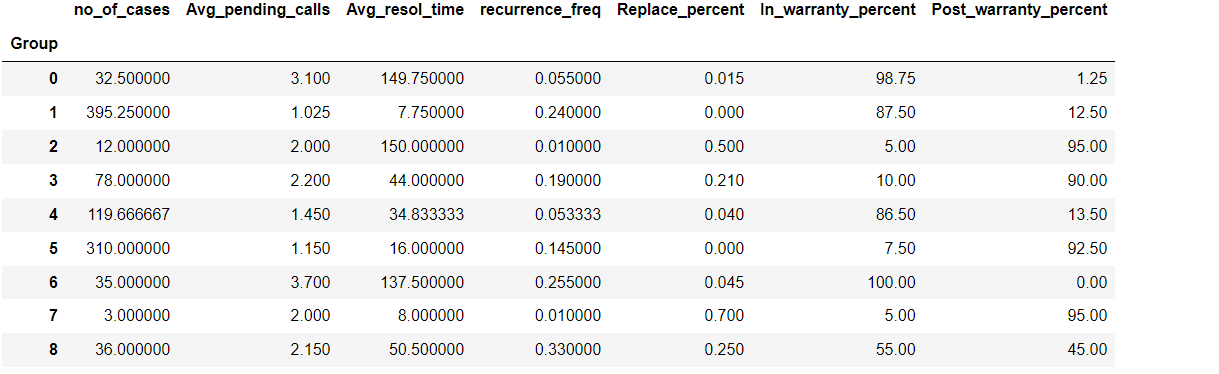
print("Groups assigned")

data.head()



techsuppclust=data.groupby(['Group'])

techsuppclust.mean()



Q.14: Write a python program to implement Agglomerative clustering on a synthetic dataset.

import pandas as pd

import numpy as np

import seaborn as sns

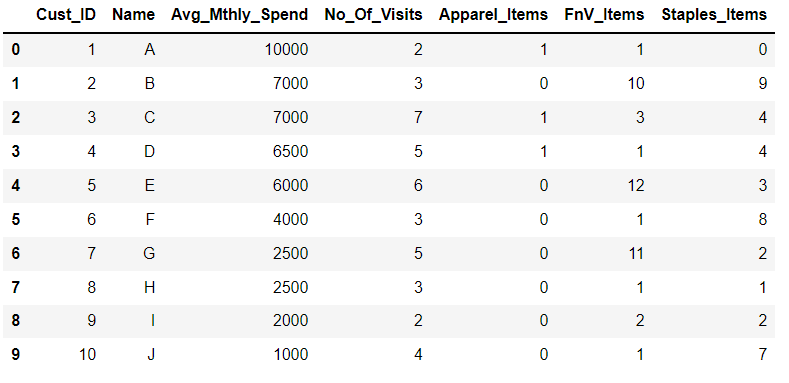
import matplotlib.pyplot as plt

%matplotlib inline

from scipy.stats import zscore

data=pd.read\_csv('Cust\_Spend\_Data.csv')

data.head(10)



data.isnull().sum()

Cust\_ID 0

Name 0

Avg\_Mthly\_Spend 0

No\_Of\_Visits 0

Apparel\_Items 0

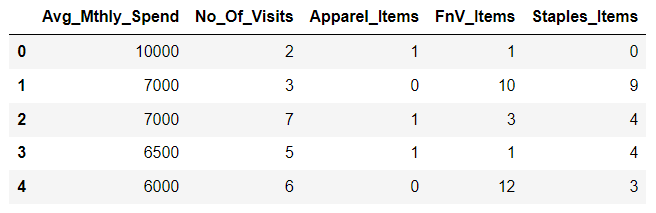
FnV\_Items 0

Staples\_Items 0

dtype: int64

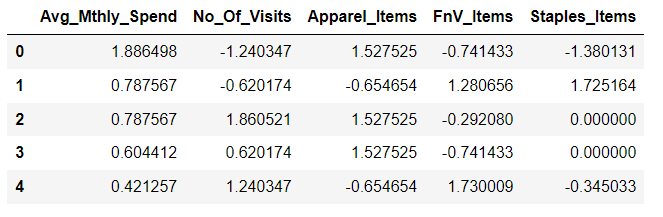
dataattr=data.iloc[:,2:]

dataattr.head()

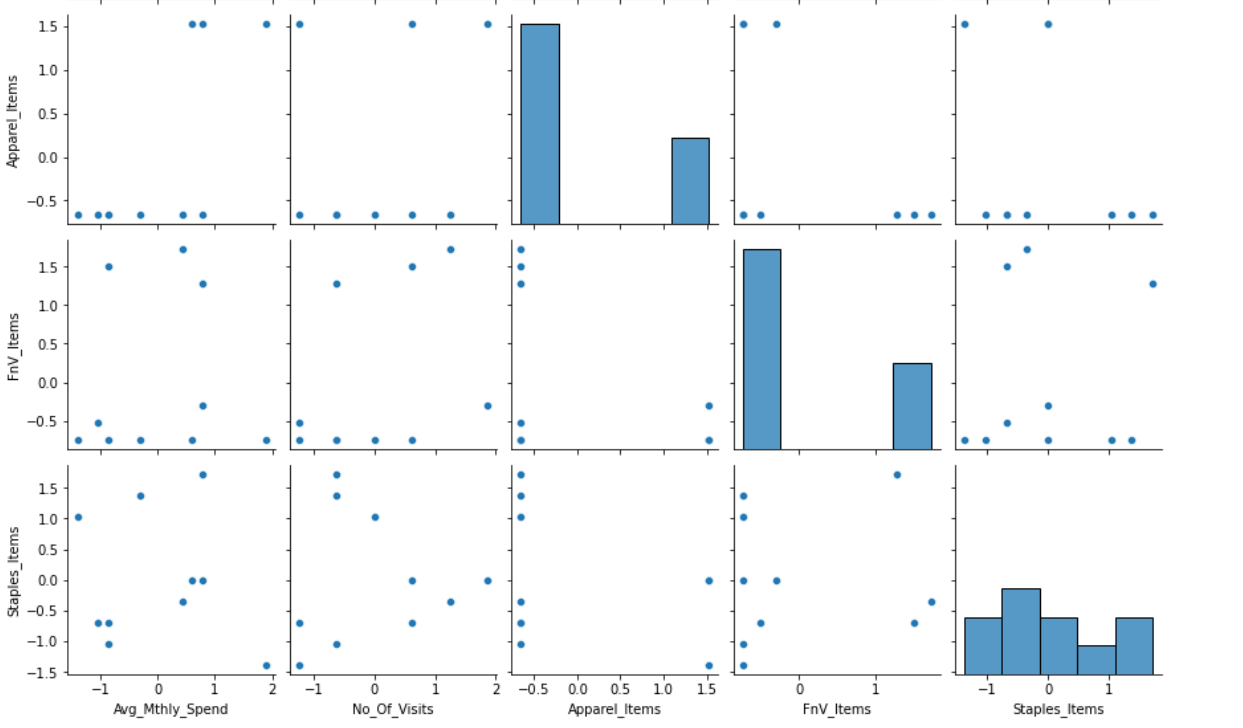
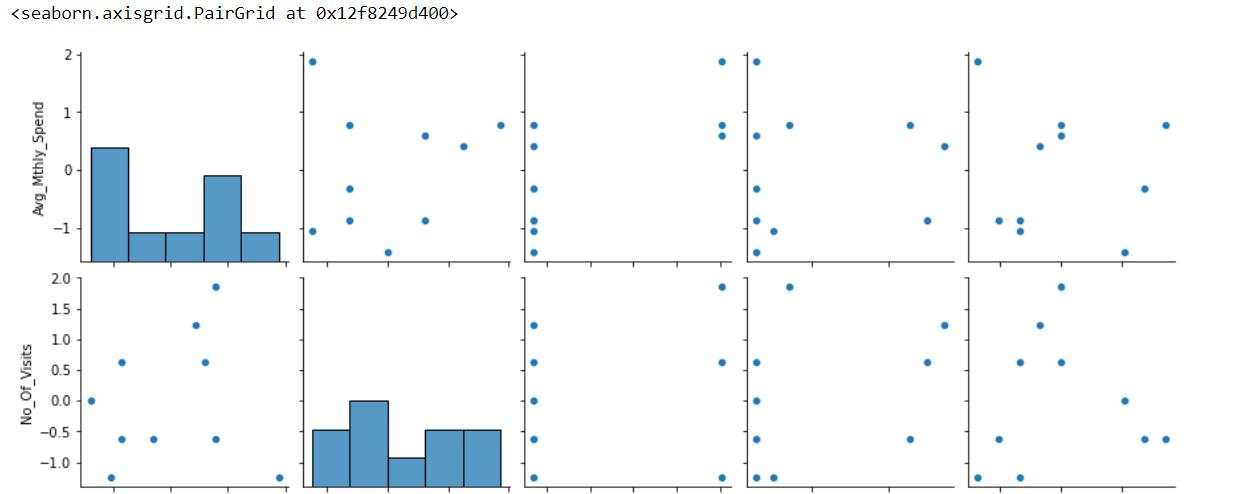


datascaled=dataattr.apply(zscore)

datascaled.head()



sns.pairplot(datascaled)

from sklearn.cluster import AgglomerativeClustering

model=AgglomerativeClustering(n\_clusters=3,affinity='euclidean',linkage='average')

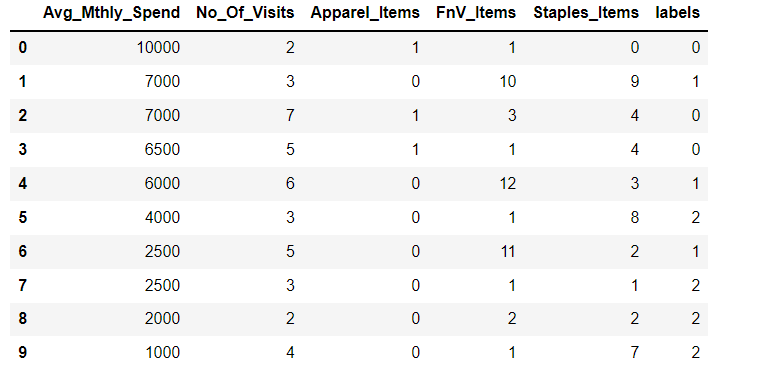
model.fit(datascaled)

AgglomerativeClustering(linkage='average', n\_clusters=3)

In [18]:

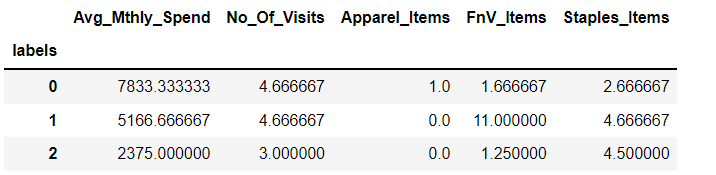
dataattr['labels']=model.labels\_

dataattr.head(15)



dataclust=dataattr.groupby(['labels'])

dataclust.mean()



from scipy.cluster.hierarchy import dendrogram,linkage

from scipy.spatial.distance import pdist

plt.figure(figsize=(10,15))

plt.title("Agglomerative clustering Dendrogram")

plt.xlabel('sample index')

plt.ylabel('Distance')

z=linkage(datascaled,metric='euclidean',method='average')

dendrogram(z,leaf\_rotation=90.,color\_threshold=40,leaf\_font\_size=8.)

plt.tight\_layout()

