Q1. Write a python program to prepare scatter plot(use Forge/Iris Dataset)

# importing libraries

import matplotlib.pyplot as plt

import pandas as pd

# Loading data from csv file

dataset = pd.read\_csv('C:\\Users\Admin\\Desktop\\iris.csv')

# creating 3 different dataframes for variety values

setosa = dataset[dataset['variety']=='Setosa']

virginica = dataset[dataset['variety']=='Virginica']

versicolor = dataset[dataset['variety']=='Versicolor']

print(dataset.describe())

print (setosa.describe())

fig,ax = plt.subplots(1,2,figsize=(9,9))

setosa.plot(x="petal.length", y= "sepal.width", kind="scatter", ax=ax[0], label='setosa', color='r')

versicolor.plot(x="petal.length", y= "sepal.width", kind="scatter", ax=ax[0], label='versicolor', color='b')

virginica.plot(x="petal.length", y= "sepal.width", kind="scatter", ax=ax[0], label='virginica', color='g')

setosa.plot(x="petal.length", y= "sepal.width", kind="scatter", ax=ax[1], label='setosa', color='r')

versicolor.plot(x="petal.length", y= "sepal.width", kind="scatter", ax=ax[1], label='versicolor', color='b')

virginica.plot(x="petal.length", y= "sepal.width", kind="scatter", ax=ax[1], label='virginica', color='g')

ax[0].set(title='Sepal Comparision', ylabel= 'sepal-width')

ax[1].set(title='Petal Comparision', ylabel= 'sepal-width')

plt.show()

# output

sepal.length sepal.width petal.length petal.width

count 150.000000 150.000000 150.000000 150.000000

mean 5.843333 3.057333 3.758000 1.199333

std 0.828066 0.435866 1.765298 0.762238

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

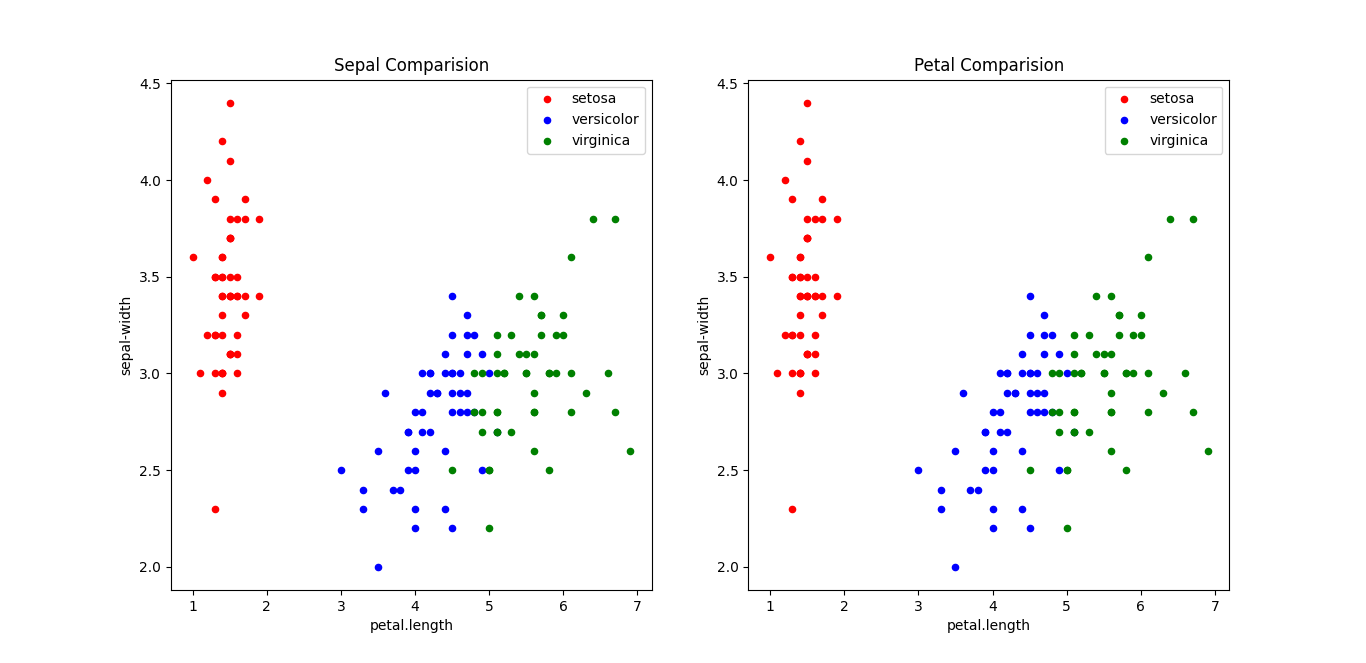
max 7.900000 4.400000 6.900000 2.500000

sepal.length sepal.width petal.length petal.width

count 50.00000 50.000000 50.000000 50.000000

mean 5.00600 3.428000 1.462000 0.246000

std 0.35249 0.379064 0.173664 0.105386

min 4.30000 2.300000 1.000000 0.100000

50% 5.00000 3.400000 1.500000 0.200000

75% 5.20000 3.675000 1.575000 0.300000

max 5.80000 4.400000 1.900000 0.600000

Q.2 Write a Python program to find null values in given dataset and remove them

# importing Libraries

import pandas as pd

# Reading dataset

dataset = pd.read\_csv('employees.xlsx')

print(dataset.describe())

print(dataset)

dataset.dropna(inplace=True)

print(dataset)

#output#

Salary Bonus %

count 1000.000000 1000.000000

mean 90662.181000 10.207555

std 32923.693342 5.528481

min 35013.000000 1.015000

25% 62613.000000 5.401750

50% 90428.000000 9.838500

75% 118740.250000 14.838000

max 149908.000000 19.944000

First Name Gender ... Senior Management Team

0 Douglas Male ... True Marketing

1 Thomas Male ... True NaN

2 Maria Female ... False Finance

3 Jerry Male ... True Finance

4 Larry Male ... True Client Services

.. ... ... ... ... ...

995 Henry NaN ... False Distribution

996 Phillip Male ... False Finance

997 Russell Male ... False Product

998 Larry Male ... False Business Development

999 Albert Male ... True Sales

[1000 rows x 8 columns]

First Name Gender ... Senior Management Team

0 Douglas Male ... True Marketing

2 Maria Female ... False Finance

3 Jerry Male ... True Finance

4 Larry Male ... True Client Services

5 Dennis Male ... False Legal

.. ... ... ... ... ...

994 George Male ... True Marketing

996 Phillip Male ... False Finan

Q.3 Write a python program to encode categorical values into numeric values for given dataset

#importing libraries

import pandas as pd

# Loading dataset from csv file

dataset=pd.read\_csv('C:\\Users\Admin\\Desktop\\iris.csv')

# Converting column to category using pandas

dataset["variety"]=dataset["variety"].astype('category')

print("\*\*\*\*\*\*\* Data types of fields in iris.csv \*\*\*\*\*\*")

print(dataset.dtypes)

print(dataset["variety"])

# Assigning encoded variable to new column using cat.codes

print("\*\*\*\*\* Adding new column \*\*\*\*\*\*")

dataset['variety\_num']=dataset['variety'].cat.codes

print(dataset.dtypes)

print()

print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

# To view full dataset

pd.set\_option('display.max\_rows',150)

pd.set\_option('display.max\_columns',7)

print(dataset)

#Output#

\*\*\*\*\*\*\* Data types of fields in iris.csv \*\*\*\*\*\*

sepal.length float64

sepal.width float64

petal.length float64

petal.width float64

variety category

dtype: object

0 Setosa

1 Setosa

2 Setosa

3 Setosa

4 Setosa

...

145 Virginica

146 Virginica

147 Virginica

148 Virginica

149 Virginica

Name: variety, Length: 150, dtype: category

Categories (3, object): ['Setosa', 'Versicolor', 'Virginica']

\*\*\*\*\* Adding new column \*\*\*\*\*\*

sepal.length float64

sepal.width float64

petal.length float64

petal.width float64

variety category

variety\_num int8

dtype: object

Q.4Write a python program to implement simple linear regression to predict house price

# Importing Libraries

import pandas as pd

import numpy as np

from sklearn import linear\_model

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

#Load the Boston Housing Data Set from sklearn.datasets and print it

#from sklearn.datasets import load\_boston

boston = pd.read\_csv('housing.csv')

print("\*\*\*\*\*\*\*\*\*\*\*\* Priting dataset \*\*\*\*\*\*\*\*\*\*\*\*")

print(boston)

#Transform the data set into a data frame

#NOTE: boston.data = the data we want,

# boston.feature\_names = the column names of the data

# boston.target = Our target variable or the price of the houses

#boston\_data=pd.read\_csv('housing.csv',usecols=['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',

#'RAD', 'TAX', 'PTRATIO'])

boston\_feature\_names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',

'RAD', 'TAX', 'PTRATIO']

df\_x = pd.DataFrame(boston, columns = boston\_feature\_names)

df\_y = pd.DataFrame(boston['MEDV'])

#Get some statistics from our data set, count, mean standard deviation etc.

df\_x.describe()

#Initialize the linear regression model

#Split the data into 67% training and 33% testing data

#NOTE: We have to split the dependent variables (x) and the target or independent variable (y)

reg = linear\_model.LinearRegression()

x\_train, x\_test, y\_train, y\_test = train\_test\_split(df\_x, df\_y, test\_size=0.33)

#Train our model with the training data

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

#Print the coefecients/weights for each feature/column of our model

print("\*\*\*\*\*\*\*\*\*\* Printing Coefficient weight for each column \*\*\*\*\*\*\*\*\*\*\*")

print(reg.coef\_)

p

#print our price predictions on our test data

y\_pred = reg.predict(x\_test)

print("\*\*\*\*\*\*\*\*\*\*\* Printing prediction based on test \*\*\*\*\*\*\*\*\*\*")

print(y\_pred)

#Print the the prediction for the third row of our test data actual price = 13.6

print("\*\*\*\*\*\* Printing prediction for third row of dataset \*\*\*\*\*\*")

print(y\_pred[2])

#print the actual price of houses from the testing data set

print("\*\*\*\*\*\*\*\*\* Actual Price \*\*\*\*\*\*\*")

print(y\_test)

# To check model performance/accuracy using,

# mean squared error which tells you how close a regression line is to a set of points.

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\* Checking Accuracy of model \*\*\*\*\*\*\*\*\*\*")

print(np.mean((y\_pred-y\_test)\*\*2))

#output#

\*\*\*\*\*\*\*\*\*\*\*\* Priting dataset \*\*\*\*\*\*\*\*\*\*\*\*

TOWN TRACT LON LAT MEDV ... AGE DIS RAD TAX PTRATIO

0 Nahant 2011 -70.9550 42.2550 24.0 ... 65.2 4.0900 1 296 15.3

1 Swampscott 2021 -70.9500 42.2875 21.6 ... 78.9 4.9671 2 242 17.8

2 Swampscott 2022 -70.9360 42.2830 34.7 ... 61.1 4.9671 2 242 17.8

3 Marblehead 2031 -70.9280 42.2930 33.4 ... 45.8 6.0622 3 222 18.7

4 Marblehead 2032 -70.9220 42.2980 36.2 ... 54.2 6.0622 3 222 18.7

.. ... ... ... ... ... ... ... ... ... ... ...

501 Winthrop 1801 -70.9860 42.2312 22.4 ... 69.1 2.4786 1 273 21.0

502 Winthrop 1802 -70.9910 42.2275 20.6 ... 76.7 2.2875 1 273 21.0

503 Winthrop 1803 -70.9948 42.2260 23.9 ... 91.0 2.1675 1 273 21.0

504 Winthrop 1804 -70.9875 42.2240 22.0 ... 89.3 2.3889 1 273 21.0

505 Winthrop 1805 -70.9825 42.2210 19.0 ... 80.8 2.5050 1 273 21.0

[506 rows x 16 columns]

\*\*\*\*\*\*\*\*\*\* Printing Coefficient weight for each column \*\*\*\*\*\*\*\*\*\*\*

[[-1.25919443e-01 3.04423059e-02 3.47335786e-02 1.88086783e+00

-1.74122110e+01 7.10085518e+00 -5.77124245e-02 -1.38677089e+00

9.58835794e-02 9.58835794e-02 -1.38640347e-02 -1.00589152e+00]]

\*\*\*\*\*\*\*\*\*\*\* Printing prediction based on test \*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\* Printing prediction for third row of dataset \*\*\*\*\*\*

[24.55701543]

\*\*\*\*\*\*\*\*\* Actual Price \*\*\*\*\*\*\*

MEDV

147 14.6

318 23.1

329 22.6

202 42.3

435 13.4

.. ...

139 17.8

448 14.1

162 50.0

28 18.4

36 20.0

[167 rows x 1 columns]

\*\*\*\*\*\*\*\*\*\*\*\*\*\* Checking Accuracy of model \*\*\*\*\*\*\*\*\*\*

MEDV 20.256286

dtype: float64

Q5. Write a python program to implement multiple linear regression for given dataset

import pandas as pd

from sklearn import linear\_model

import statsmodels.api as sm

Stock\_Market = {'Year': [2017,2017,2017,2017,2017,2017,2017,

2017,2017,2017,2017,2017,2016,2016,2016,2016,2016,2016,2016,2016,2016,2016,2016,2016],

'Month': [12, 11,10,9,8,7,6,5,4,3,2,1,12,

11,10,9,8,7,6,5,4,3,2,1],

'Interest\_Rate': [2.75,2.5,2.5,2.5,2.5,2.5,

2.5,2.25,2.25,2.25,2,2,2,

1.75,1.75,1.75,1.75,1.75,1.75,1.75,1.75,1.75,1.75,1.75],

'Unemployment\_Rate': [5.3,5.3,5.3,5.3,5.4,

5.6,5.5,5.5,5.5,5.6,5.7,5.9,6,5.9,5.8,6.1,6.2,6.1,6.1,6.1,5.9,6.2,6.2,6.1],

'Stock\_Index\_Price': [1464,1394,1357,1293,

1256,1254,1234,1195,1159,1167,1130,1075,1047,965,943,958,971,949,884,866,876,822,704,719]

}

df = pd.DataFrame(Stock\_Market,columns=['Year','Month','Interest\_Rate','Unemployment\_Rate','Stock\_Index\_Price'])

X = df[['Interest\_Rate','Unemployment\_Rate']]

# here we have 2 variables for multiple regression. If you just want to use one variable for simple linear regression, then use X = df['Interest\_Rate'] for example.Alternatively, you may add additional variables within the brackets

Y = df['Stock\_Index\_Price']

# with sklearn

regr = linear\_model.LinearRegression()

regr.fit(X, Y)

print('Intercept: \n', regr.intercept\_)

print('Coefficients: \n', regr.coef\_)

# prediction with sklearn

New\_Interest\_Rate = 2.75

New\_Unemployment\_Rate = 5.3

print ('Predicted Stock Index Price: \n', regr.predict([[New\_Interest\_Rate ,New\_Unemployment\_Rate]]))

# with statsmodels

model = sm.OLS(Y, X).fit()

predictions = model.predict(X)

print(predictions)

print\_model = model.summary()

print(print\_model)

#OutPut#

Intercept:

1798.4039776258544

Coefficients:

[ 345.54008701 -250.14657137]

Predicted Stock Index Price:

[1422.86238865]

0 1455.365946

1 1315.638754

2 1315.638754

3 1315.638754

4 1314.098505

5 1311.018009

6 1312.558257

7 1172.831065

8 1172.831065

9 1171.290817

10 1030.023376

11 1026.942880

12 1025.402631

13 887.215688

14 888.755936

15 884.135191

16 882.594943

17 884.135191

18 884.135191

19 884.135191

20 887.215688

21 882.594943

22 882.594943

23 884.135191

dtype: float64

OLS Regression Results

=======================================================================================

Dep. Variable: Stock\_Index\_Price R-squared (uncentered): 0.996

Model: OLS Adj. R-squared (uncentered): 0.995

Method: Least Squares F-statistic: 2508.

Date: Sat, 08 Oct 2022 Prob (F-statistic): 1.10e-26

Time: 09:15:03 Log-Likelihood: -136.70

No. Observations: 24 AIC: 277.4

Df Residuals: 22 BIC: 279.8

Df Model: 2

Covariance Type: nonrobust

=====================================================================================

coef std err t P>|t| [0.025 0.975]

-------------------------------------------------------------------------------------

Interest\_Rate 558.9088 34.043 16.418 0.000 488.308 629.510

Unemployment\_Rate -15.4025 12.366 -1.246 0.226 -41.047 10.242

==============================================================================

Omnibus: 4.661 Durbin-Watson: 0.513

Prob(Omnibus): 0.097 Jarque-Bera (JB): 3.139

Skew: -0.870 Prob(JB): 0.208

Kurtosis: 3.330 Cond. No. 14.4

==============================================================================

Notes:

[1] R² is computed without centering (uncentered) since the model does not contain a constant.

[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Q.6 Write a python program to implement polynomial regression for given dataset.

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

#Importing the dataset

datas=pd.read\_csv('data1.csv')

print(datas)

print(datas.head())

# Dividing the dataset into 2 components

X = datas.iloc[:, 1:2].values

y = datas.iloc[:, 2].values

# Fitting Linear Regression to the dataset

from sklearn.linear\_model import LinearRegression

line = LinearRegression()

line.fit(X, y)

# Visualising the Linear Regression results

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

plt.plot(X,line.predict(X), color = 'red')

plt.title('Linear Regression')

plt.xlabel('Temperature')

plt.ylabel('Pressure')

plt.show()

# Fitting Polynomial Regression to the dataset

from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree = 8)

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

lin2 = LinearRegression()

lin2.fit(X\_poly, y)

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

plt.plot(X, lin2.predict(poly.fit\_transform(X)), color = 'red')

plt.title('Polynomial Regression')

plt.xlabel('Temperature')

plt.ylabel('Pressure')

plt.show()

#Output#

srno Temperature Pressure

0 1 0 0.0002

1 2 20 0.0012

2 3 40 0.0060

3 4 60 0.0300

4 5 80 0.0900

5 6 100 0.2700

6 7 120 0.2900

7 8 140 0.3200

srno Temperature Pressure

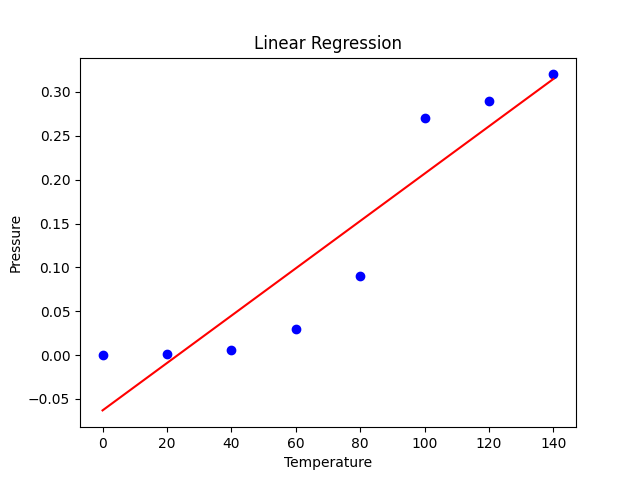
0 1 0 0.0002

1 2 20 0.0012

2 3 40 0.0060

3 4 60 0.0300

4 5 80 0.0900



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

# load the iris dataset

from sklearn.datasets import load\_iris

iris = load\_iris()

# store the feature matrix (X) and response vector (y)

X = iris.data

y = iris.target

# splitting 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=0.4, random\_state=1)

# training the model on training set

from sklearn.naive\_bayes import GaussianNB

gnb = GaussianNB()

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

# making predictions on the testing set

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

# comparing actual response values (y\_test) with predicted response values (y\_pred)

from sklearn import metrics

print("Gaussian Naive Bayes model accuracy(in %):", metrics.accuracy\_score(y\_test, y\_pred)\*100)

#Output#

Gaussian Naive Bayes model accuracy(in %): 95.0

Q.8 Write python program to implement decision tree whether or not to play tennis

# importing data

import numpy as np

import pandas as pd

df = pd.read\_csv('weather.csv')

# converting categorial variables into dummies/indicator variables

df\_getdummy = pd.get\_dummies(data=df, columns= ['Temperature', 'Outlook', 'Windy', 'Humidity'])

# Separating training set and test set

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

x= df\_getdummy.drop('Played?', axis=1)

y= df\_getdummy['Played?']

X\_train, X\_test, y\_train,y\_test = train\_test\_split(x,y,test\_size=0.30, random\_state=101)

#visualising the decision tree diagram

from sklearn.tree import DecisionTreeClassifier

dtree = DecisionTreeClassifier(max\_depth=3)

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

predictions = dtree.predict(X\_test)

print(predictions)

from sklearn.tree import plot\_tree

import matplotlib.pyplot as plt

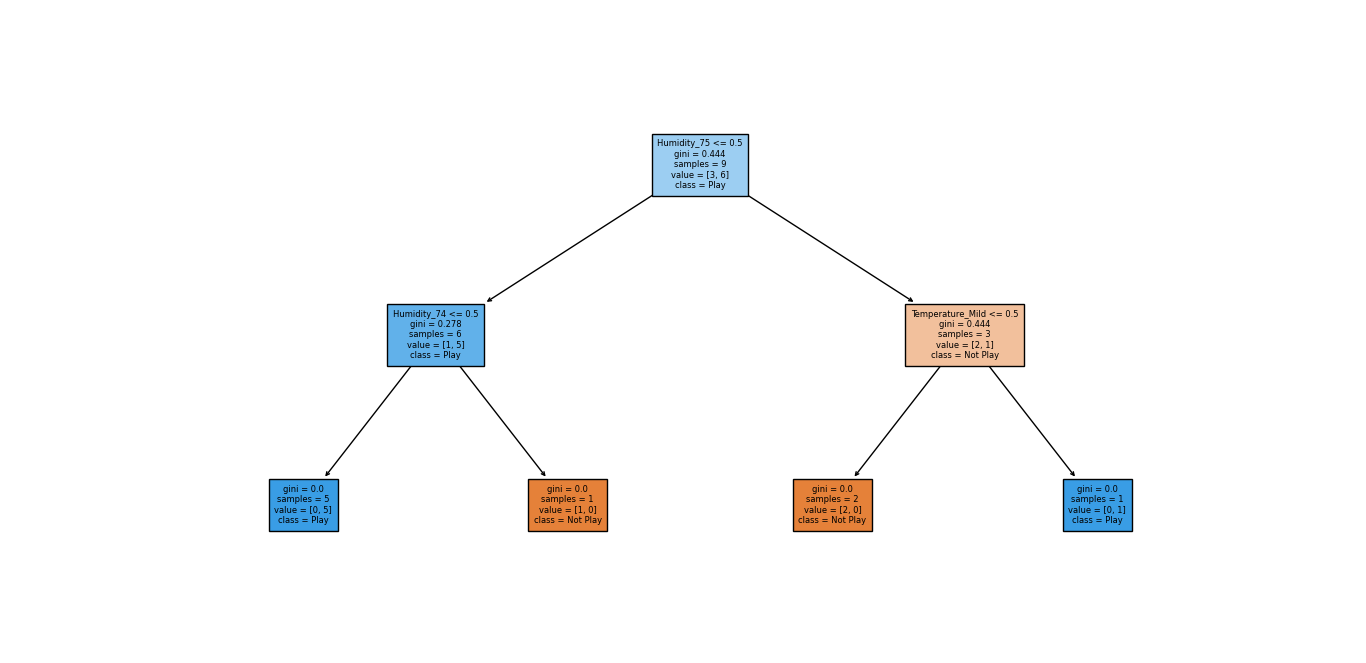
fig = plt.figure(figsize=(6,6))

plot\_tree(dtree,feature\_names=df\_getdummy.columns,fontsize=6,filled=True, class\_names=['Not Play','Play'])

plt.show()

OutPut

['Yes' 'Yes' 'Yes' 'Yes' 'Yes']



Q.9)Write a python program to implement linear SVM.#

# Import the Libraries

import numpy as np

import matplotlib.pyplot as plt

from sklearn import svm, datasets

# Import some Data from the iris Data Set

iris = datasets.load\_iris()

# Take only the first two features of Data.

# To avoid the slicing, Two-Dim Dataset can be used

X = iris.data[:, :2]

y = iris.target

# C is the SVM regularization parameter

C = 1.0

# Create an Instance of SVM and Fit out the data.

# Data is not scaled so as to be able to plot the support vectors

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

# create a mesh to plot

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))

# Plot the data for Proper Visual Representation

plt.subplot(1, 1, 1)

# Predict the result by giving Data to the model

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')

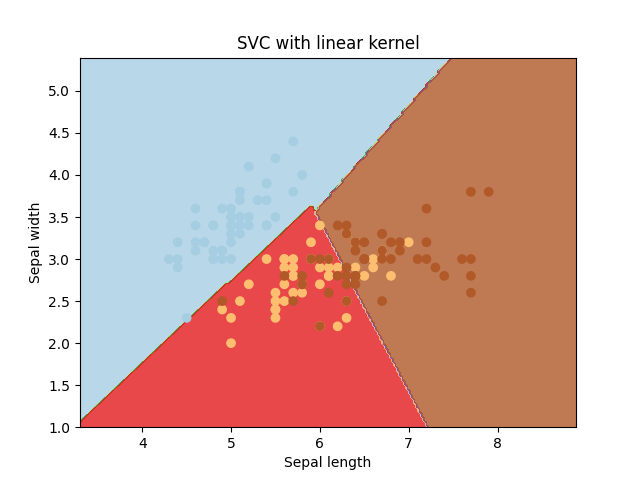
plt.xlim(xx.min(), xx.max())

plt.title('SVC with linear kernel')

# Output the Plot

plt.show()

#output#



Q10)

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

from sklearn.preprocessing import scale

from sklearn.linear\_model import LinearRegression,Ridge,RidgeCV,Lasso,LassoCV

from sklearn.model\_selection import KFold, cross\_val\_score, train\_test\_split

from sklearn.metrics import mean\_squared\_error

from sklearn.decomposition import PCA

df=pd.read\_csv('PlayTennis.csv',sep=",",index\_col=0)

df.head()

target="price"

x=df.drop(target,axis=1)

y=df[target]

X.head()

X.['Outlook'].unique()

Outlook\_mapping={'Sunny':1,'Overcast':2.,'Rain':3}

X.['Outlook']=X.['Outlook'].map(Outlook\_mapping)

X.['Temperature'].unique()

Temperature\_mapping={'Hot':1,'Mild':2.,'Cold':3,}

X.['Temperature']=X.['Temperature'].map(Temperature\_mapping)

X.['Humidity'].unique()

Humidity\_mapping={'High':1,'Normal':2}

X.['Humidity']=X.['Humidity'].map(Humidity\_mapping)

X.head()

X\_train,X\_test,ytrain,ytest=train\_test\_split(x,y,test\_size=0.1,random\_state=0.2)

X\_train.shape

X\_train.scaled,X\_test.scaled=scale(X\_train),scale(X\_test)

X\_train.scaled

cv=KFold(n\_splits=10,shuffle=True,random\_state=42)

lin\_reg=LinearRegression().fit(X\_train.scaled,y\_train)

lr\_scores=1\*cross\_Val\_score(lin\_reg,X\_train.scaled,ytrain,cv=cv,scoring="neg\_root\_mean\_squared\_error")

lr\_scores

lr\_score\_train=np.mean(lr\_scores)

lr\_score\_train

**y\_predicted=lin\_reg.predict(X\_train.scaled)**

**lr\_score\_test=mean\_squared\_error(ytest,y\_predicted,squared=false)**

**lr\_score\_test**

Q.11Write a python program to implement k-nearest Neighbours ML algorithm to build prediction model (Use Forge/Iris/housing Dataset).#

# Import necessary modules

from sklearn.neighbors import KNeighborsClassifier

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

from sklearn.datasets import load\_iris

import numpy as np

import matplotlib.pyplot as plt

irisData = load\_iris()

# Create feature and target arrays

X = irisData.data

y = irisData.target

# Split into training and test set

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

X, y, test\_size = 0.2, random\_state=42)

# Setup arrays to store train and test accuracies

neighbors = np.arange(1, 9)

train\_accuracy = np.empty(len(neighbors))

test\_accuracy = np.empty(len(neighbors))

# Loop over different values of k

for i, k in enumerate(neighbors):

# Setup a k-NN Classifier with k neighbors: knn

knn = KNeighborsClassifier(n\_neighbors=k)

# Fit the classifier to the training data

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

# Compute training and test data accuracy

train\_accuracy[i] = knn.score(X\_train, y\_train)

test\_accuracy[i] = knn.score(X\_test, y\_test)

# Generate plot

plt.plot(neighbors, test\_accuracy, label = 'Testing dataset Accuracy')

plt.plot(neighbors, train\_accuracy, label = 'Training dataset Accuracy')

plt.legend()

plt.xlabel('n\_neighbors')

plt.ylabel('Accuracy')

plt.show()

#Output#

