Q1 .Write a Python program to prepare Scatter Plot for 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))
x=data["sepal_length"]
y=data["petal_length"]
plt.scatter(x,y)
plt.show()
```

Q2. Write a python program to find all null values in a given dataset and remove them.

```
import pandas

# reading the CSV file
csvFile = pandas.read_csv('employees.csv')
# displaying the contents of the CSV file
print(csvFile)
count=csvFile.isnull()
#displaying NULL content
print(count)
newdf = csvFile.dropna()
print(newdf)
```

Q3 . Write a python program to make Categorical values in numeric format for a given dataset

```
import pandas as pd

cars = pd.read_csv('data.csv')
print(cars.to_string())

ohe_cars = pd.get_dummies(cars[['Car']])
print(ohe_cars.to_string())
```

Q4. Write a python program to Implement Simple Linear Regression for predicting Salary .

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import  LinearRegression
from sklearn import metrics
from sklearn.preprocessing import StandardScaler

df = pd.read_csv('Salary_data.csv')
print(df.to_string())
des=df.describe()
print(des)
x=df['YearsExp']
y=df['salary']
#plt.scatter(x,y)
#plt.show()
x=df['YearsExp'].values.reshape(-1,1)
y=df['salary'].values.reshape(-1,1)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
sc=StandardScaler()
sc.fit(x_train)
x_train=sc.transform(x_train)
x_test=sc.transform(x_test)
y_train=sc.transform(y_train)
y_test=sc.transform(y_test)
LR=LinearRegression()
LR.fit(x_train,y_train)
print("Intercept",LR.intercept_)
print("Coefficient",LR.coef_)
y_pred=LR.predict(x_test)
plt.scatter(x_train,y_train)
plt.plot(x_test,y_pred,color='red')
plt.title("Simple Regression")
plt.xlabel("YearExperience")
```

```
plt.ylabel("Salary")
plt.show()
data=pd.DataFrame({'Actual':y_test.flatten(),'predicted':y_pred.flatten()})
print(data)
y_pred2=LR.predict([[3]])
print(y_pred2)
```

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

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import  LinearRegression
from sklearn import metrics
from sklearn.preprocessing import StandardScaler

df = pd.read_csv("data.csv")
data=df.head()
print(data)

x= df[['Weight','Volume']]
y= df['CO2']

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
MLR=LinearRegression()
MLR.fit(x_train,y_train)
print("Intercept",MLR.intercept_)
print("Coefficient",MLR.coef_)
#predict the CO2 emission of a car where the weight is 2300g, and the volume is 1300ccm:
predictedCO2 = MLR.predict([[1500, 1140]])
print(predictedCO2)
```

Q6. Write a python program to implement Polynomial Linear 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('data.csv')
datas
# 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
lin = LinearRegression()
lin.fit(X, y)
# Fitting Polynomial Regression to the dataset
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree = 4)
X_poly = poly.fit_transform(X)
poly.fit(X_poly, y)
lin2 = LinearRegression()
lin2.fit(X_poly, y)
# Visualising the Linear Regression results
plt.scatter(X, y, color = 'blue')
plt.plot(X, lin.predict(X), color = 'red')
plt.title('Linear Regression')
plt.xlabel('Temperature')
plt.ylabel('Pressure')
plt.show()
# Visualising the Polynomial Regression results
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()
```

Q7. Write a python program to implement Naive Bayes.

```
# Importing library
import math
import random
import csv
 # the categorical class names are changed to numberic data
# eg: yes and no encoded to 1 and 0
def encode_class(mydata):
    classes = []
    for i in range(len(mydata)):
        if mydata[i][-1] not in classes:
            classes.append(mydata[i][-1])
    for i in range(len(classes)):
        for j in range(len(mydata)):
            if mydata[j][-1] == classes[i]:
                mydata[j][-1] = i
    return mydata
# Splitting the data
def splitting(mydata, ratio):
    train_num = int(len(mydata) * ratio)
    train = []
    # initially testset will have all the dataset
    test = list(mydata)
    while len(train) < train_num:
        # index generated randomly from range 0
        # to length of testset
        index = random.randrange(len(test))
        # from testset, pop data rows and put it in train
        train.append(test.pop(index))
    return train, test
# Group the data rows under each class yes or
# no in dictionary eg: dict[yes] and dict[no]
def groupUnderClass(mydata):
     dict = {}
```

```
    for i in range(len(mydata)):
        if (mydata[i][-1] not in dict):
            dict[mydata[i][-1]] = []
        dict[mydata[i][-1]].append(mydata[i])
    return dict
# Calculating Mean
def mean(numbers):
    return sum(numbers) / float(len(numbers))
# Calculating Standard Deviation
def std_dev(numbers):
    avg = mean(numbers)
    variance = sum([pow(x - avg, 2) for x in numbers]) / float(len(numbers) - 1)
    return math.sqrt(variance)
def MeanAndStdDev(mydata):
    info = [(mean(attribute), std_dev(attribute)) for attribute in zip(*mydata)]
    # eg: list = [ [a, b, c], [m, n, o], [x, y, z]]
    # here mean of 1st attribute =(a + m+x), mean of 2nd attribute = (b + n+y)/3
    # delete summaries of last class
    del info[-1]
    return info
# find Mean and Standard Deviation under each class
def MeanAndStdDevForClass(mydata):
    info = {}
    dict = groupUnderClass(mydata)
    for classValue, instances in dict.items():
        info[classValue] = MeanAndStdDev(instances)
    return info
# Calculate Gaussian Probability Density Function
def calculateGaussianProbability(x, mean, stdev):
    expo = math.exp(-(math.pow(x - mean, 2) / (2 * math.pow(stdev, 2))))
    return (1 / (math.sqrt(2 * math.pi) * stdev)) * exp
# Calculate Class Probabilities
def calculateClassProbabilities(info, test):
    probabilities = {}
    for classValue, classSummaries in info.items():
```

```
        probabilities[classValue] = 1
        for i in range(len(classSummaries)):
            mean, std_dev = classSummaries[i]
            x = test[i]
            probabilities[classValue] *= calculateGaussianProbability(x, mean, std_dev)
    return probabilities

 # Make prediction - highest probability is the prediction
def predict(info, test):
    probabilities = calculateClassProbabilities(info, test)
    bestLabel, bestProb = None, -1
    for classValue, probability in probabilities.items():
        if bestLabel is None or probability > bestProb:
            bestProb = probability
            bestLabel = classValue
    return bestLabel
# returns predictions for a set of examples
def getPredictions(info, test):
    predictions = []
    for i in range(len(test)):
        result = predict(info, test[i])
        predictions.append(result)
    return predictions

# Accuracy score
def accuracy_rate(test, predictions):
    correct = 0
    for i in range(len(test)):
        if test[i][-1] == predictions[i]:
            correct += 1
    return (correct / float(len(test))) * 100.0



# driver code
```

```
# add the data path in your system

filename = r'E:\user\MACHINE LEARNING\machine learning algos\Naive bayes\filedata.csv'



# load the file and store it in mydata list

mydata = csv.reader(open(filename, "rt"))

mydata = list(mydata)

mydata = encode_class(mydata)

for i in range(len(mydata)):

   mydata[i] = [float(x) for x in mydata[i]]

# split ratio = 0.7

# 70% of data is training data and 30% is test data used for testing

ratio = 0.7

train_data, test_data = splitting(mydata, ratio)

print('Total number of examples are: ', len(mydata))

print('Out of these, training examples are: ', len(train_data))

print("Test examples are: ", len(test_data))

# prepare model

info = MeanAndStdDevForClass(train_data)

# test model

predictions = getPredictions(info, test_data)

accuracy = accuracy_rate(test_data, predictions)

print("Accuracy of your model is: ", accuracy)
```

Q8. Write a python program to implement Decision Tree whether or not to play Tennis

```
#numpy and pandas initialization
import numpy as np
import pandas as pd

#Loading the PlayTennis data
PlayTennis = pd.read_csv("PlayTennis.csv")

PlayTennis
from sklearn.preprocessing import LabelEncoder

Le = LabelEncoder()

PlayTennis['outlook'] = Le.fit_transform(PlayTennis['outlook'])

PlayTennis['temp'] = Le.fit_transform(PlayTennis['temp'])

PlayTennis['humidity'] = Le.fit_transform(PlayTennis['humidity'])

PlayTennis['windy'] = Le.fit_transform(PlayTennis['windy'])

PlayTennis['play'] = Le.fit_transform(PlayTennis['play'])

PlayTennis
```

```
#split the training data and its coresponding prediction values.
#y - holds all the decisions.
#X - holds the training data.
y = PlayTennis['play']
X = PlayTennis.drop(['play'],axis=1)
# Fitting the model
from sklearn import tree
clf = tree.DecisionTreeClassifier(criterion = 'entropy')
clf = clf.fit(X, y)
# We can visualize the tree using tree.plot_tree
tree.plot_tree(clf)
import graphviz
dot_data = tree.export_graphviz(clf, out_file=None)
graph = graphviz.Source(dot_data)
graph
```

Q9. 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()
```