**BCA 507(C) Lab on Machine Learning using Python**

1. Write a python program to find mean, mode, median.

**Ass: Mean:**

import numpy

speed = [99,86,87,88,111,86,103,87,94,78,77,85,86]

x = numpy.mean(speed)

print(x)

**Output: 89.76923076923077**

**Median:**

import numpy

speed = [99,86,87,88,111,86,103,87,94,78,77,85,86]

x = numpy.median(speed)

print(x)

**Output: 87.0**

**Mode:**

from scipy import stats

speed = [99,86,87,88,111,86,103,87,94,78,77,85,86]

x = stats.mode(speed)

print(x)

**Output: ModeResult(mode=array([86]), count=array([3]))**

**2. Write a python program to typical normal data distribution.**

**Ass:** from numpy import random

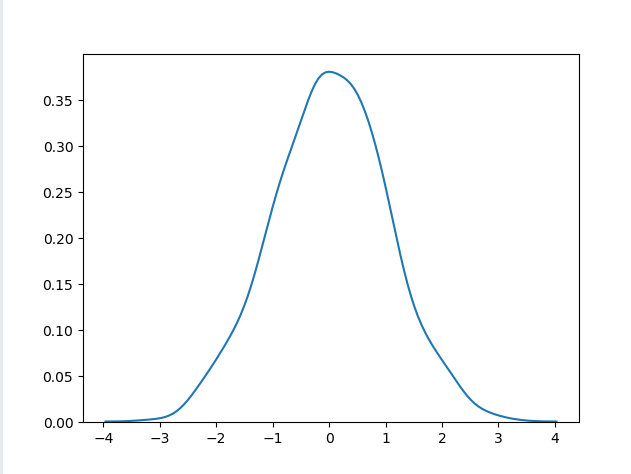
import matplotlib.pyplot as plt

import seaborn as sns

sns.distplot(random.normal(size=1000), hist=False)

plt.show()

**Output:**



**3. Write a python program to draw scatter plot of linear regression.**

**Ass:** import numpy as np

import matplotlib.pyplot as plt

x = np.array([1, 2, 3, 4, 5])

y = np.array([2, 3, 5, 7, 11])

plt.scatter(x, y, color='blue', label='Data points')

m, b = np.polyfit(x, y, 1)

plt.plot(x, m\*x + b, color='red', label='Regression line')

plt.xlabel('X-axis')

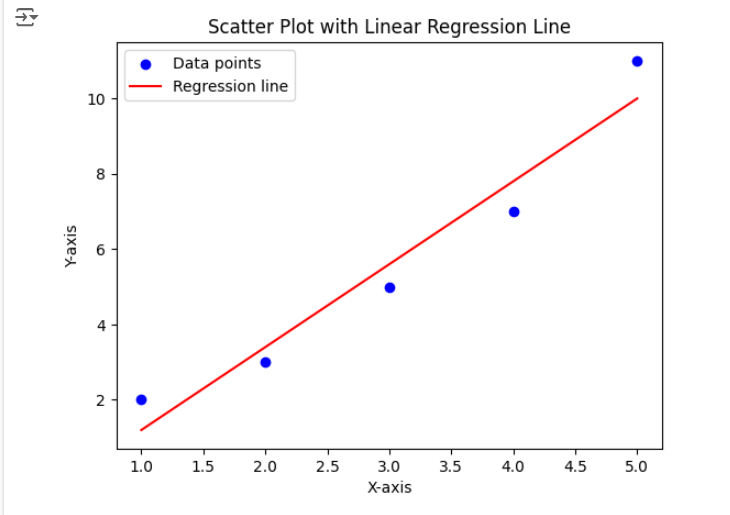
plt.ylabel('Y-axis')

plt.title('Scatter Plot with Linear Regression Line')

plt.legend()

plt.show()

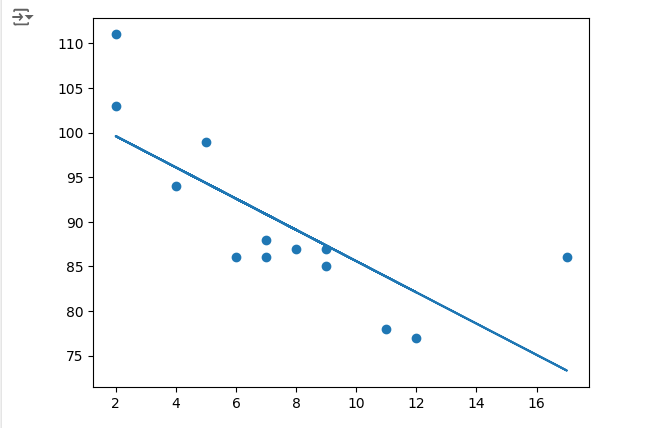
**Output:**



**4. Write a python program to draw the line of Linear Regression.**

**Ass:** import matplotlib.pyplot as plt  
from scipy import stats  
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]  
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]  
slope, intercept, r, p, std\_err = stats.linregress(x, y)  
def myfunc(x):  
  return slope \* x + intercept  
mymodel = list(map(myfunc, x))  
plt.scatter(x, y)  
plt.plot(x, mymodel)  
plt.show()

**Output:**



**5. Write a python program to predict the speed of a 5 years old car.**

**Ass:** import numpy as np

from scipy import stats

import matplotlib.pyplot as plt

# Sample data: ages of cars (in years) and their speeds (in km/h)

ages = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

speeds = np.array([120, 115, 110, 105, 100, 95, 90, 85, 80, 75])

slope, intercept, r\_value, p\_value, std\_err = stats.linregress(ages, speeds)

# Function to predict speed based on age

def predict\_speed(age):

return slope \* age + intercept

predicted\_speed = predict\_speed(5)

print(f"The predicted speed of a 5-year-old car is {predicted\_speed:.2f} km/h")

plt.scatter(ages, speeds, color='blue', label='Actual data')

plt.plot(ages, predict\_speed(ages), color='red', label='Regression line')

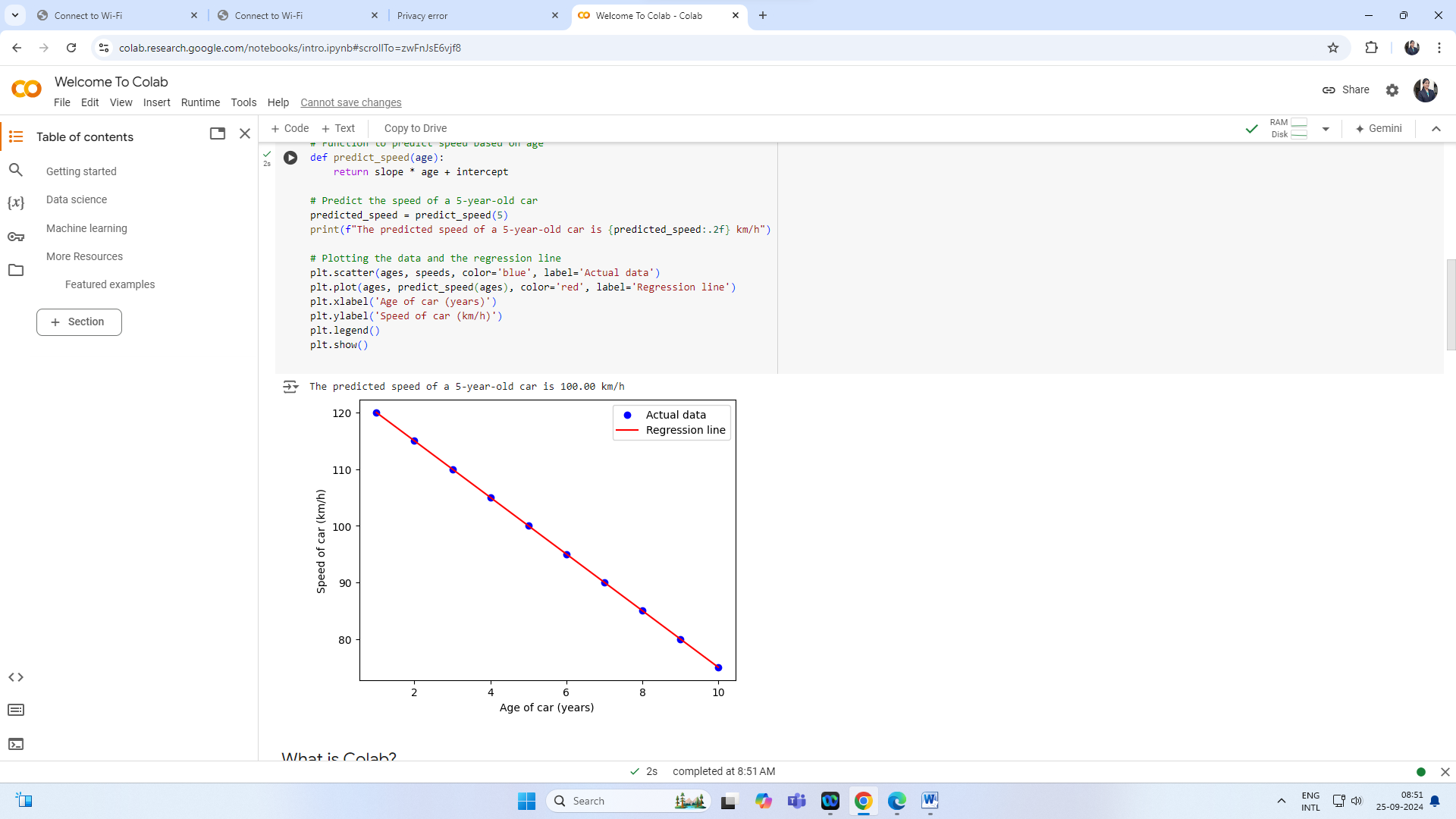
plt.xlabel('Age of car (years)')

plt.ylabel('Speed of car (km/h)')

plt.legend()

plt.show()

**Output:**

****

**6. Write a python program to print the coefficient values of the regression object.**

**Ass:** import numpy as np

from sklearn.linear\_model import LinearRegression

# Sample data

X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])

y = np.dot(X, np.array([1, 2])) + 3

# Create and fit the model

model = LinearRegression().fit(X, y)

# Print the coefficients

print("Coefficients:", model.coef\_)

print("Intercept:", model.intercept\_)

**Output:**



**7. Write a python program to 2d binary classification data generated by make\_circles()**

**have a spherical decision boundary.**

**Ass:** import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make\_circles

from sklearn.svm import SVC

from sklearn.preprocessing import StandardScaler

# Generate 2D binary classification data

X, y = make\_circles(n\_samples=300, factor=0.5, noise=0.1, random\_state=42)

# Standardize the data

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Fit the SVM model with RBF kernel

svm = SVC(kernel='rbf', C=1.0, gamma='auto')

svm.fit(X\_scaled, y)

# Plot the decision boundary

def plot\_decision\_boundary(model, X, y):

h = .02 # step size in the mesh

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

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

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

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

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

Z = Z.reshape(xx.shape)

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

plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', marker='o')

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

plt.ylim(yy.min(), yy.max())

plt.xlabel('Feature 1')

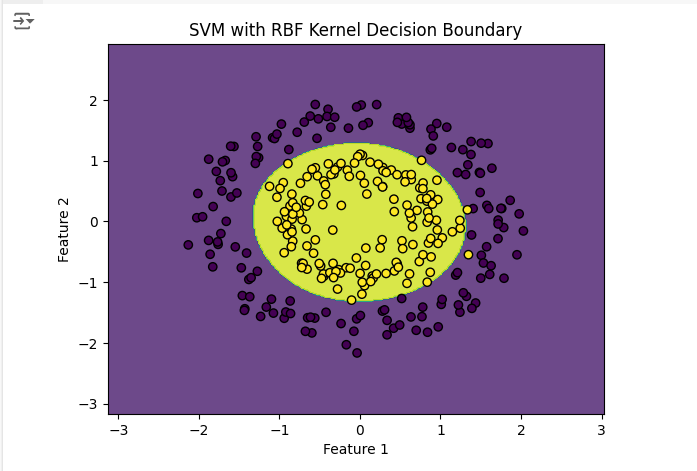
plt.ylabel('Feature 2')

plt.title('SVM with RBF Kernel Decision Boundary')

plt.show()

plot\_decision\_boundary(svm, X\_scaled, y)

Output:



**8. Write a python program to display the plot we can use the functions plot() and show()**

**from pyplot.**

**Ass:** import matplotlib.pyplot as plt

# Data for plotting

x = [1, 2, 3, 4]

y = [2, 4, 1, 3]

# Creating the plot

plt.plot(x, y)

# Adding labels and title

plt.xlabel('x - axis')

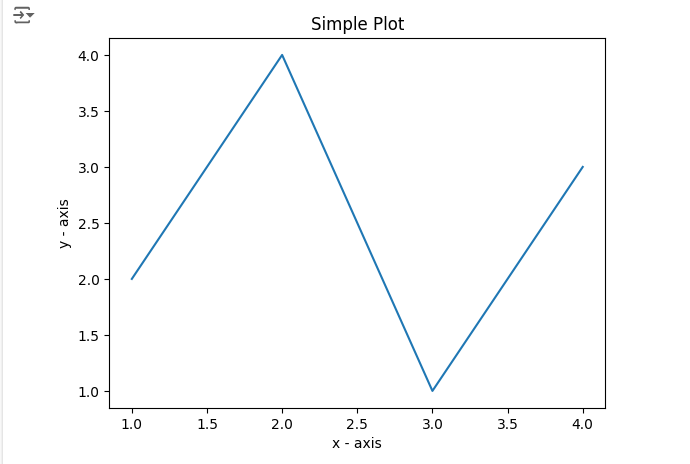
plt.ylabel('y - axis')

plt.title('Simple Plot')

# Displaying the plot

plt.show()

Output:



**9. Write a python program to data generated by the function make\_blobs() are blobs that**

**can be utilized for clustering.**

**Ass:** import matplotlib.pyplot as plt

from sklearn.datasets import make\_blobs

from sklearn.cluster import KMeans

# Generate synthetic data

X, y = make\_blobs(n\_samples=300, centers=4, n\_features=2, cluster\_std=1.0, random\_state=42)

# Apply K-Means clustering

kmeans = KMeans(n\_clusters=4, random\_state=42)

kmeans.fit(X)

y\_kmeans = kmeans.predict(X)

# Plot the clusters

plt.scatter(X[:, 0], X[:, 1], c=y\_kmeans, s=50, cmap='viridis')

centers = kmeans.cluster\_centers\_

plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200, alpha=0.75, marker='X')

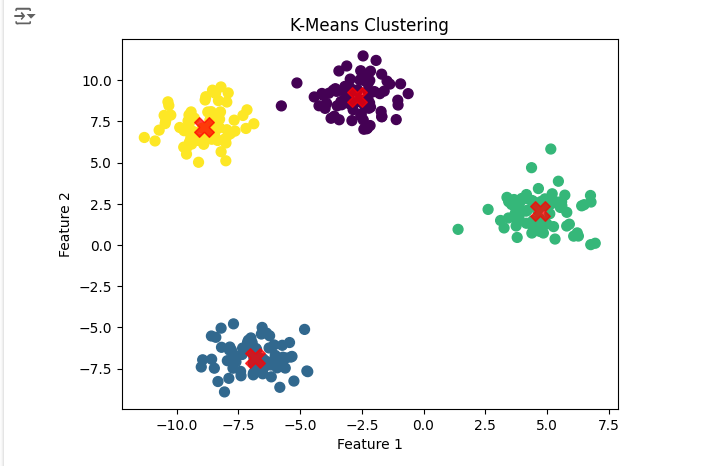
plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.title('K-Means Clustering')

plt.show()

**Output:**



**10. Write a python program to random multi-label classification data is created by the**

**function make make\_multilabel\_classification().**

Ass: from sklearn.datasets import make\_multilabel\_classification

import matplotlib.pyplot as plt

# Generate random multi-label classification data

X, y = make\_multilabel\_classification(n\_samples=100, n\_features=20, n\_classes=5, n\_labels=3, random\_state=42)

# Print the shape of the features and labels

print("Features shape:", X.shape)

print("Labels shape:", y.shape)

# Plot the first two features

plt.scatter(X[:, 0], X[:, 1], marker='o', c=y[:, 0], edgecolor='k')

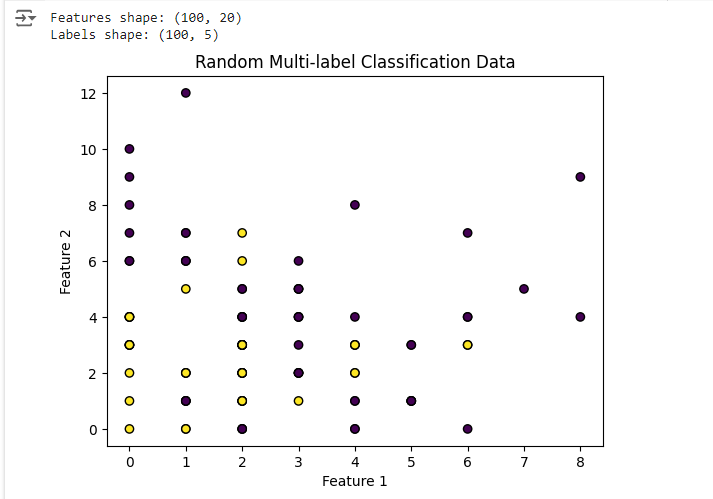
plt.title("Random Multi-label Classification Data")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.show()

**Output:**



**11. Write a python program to implement the KNN algorithm.**

**Ass:** import numpy as np

from sklearn.datasets import load\_iris

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

# Load the Iris dataset

iris = load\_iris()

X, y = iris.data, iris.target

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Create the KNN classifier

knn = KNeighborsClassifier(n\_neighbors=3)

# Fit the classifier to the training data

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

# Predict the labels for the test set

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

# Calculate the accuracy of the classifier

accuracy = accuracy\_score(y\_test, y\_pred)

print(f'Accuracy: {accuracy \* 100:.2f}%')

**Output:**



**12. Write a python program to creating a dataframe to implement one hot encoding from**

**CSV file.**

**Ass: (Using Data frame)**

import pandas as pd

# Load the CSV file into a DataFrame

df = pd.read\_csv('Candy\_Sales.csv')

# Identify categorical columns

categorical\_columns = df.select\_dtypes(include=['object']).columns

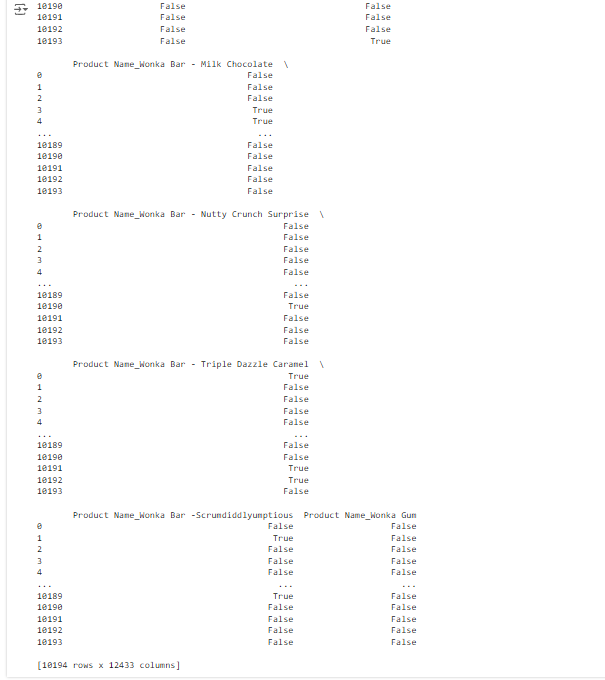
# Perform one-hot encoding

df\_encoded = pd.get\_dummies(df, columns=categorical\_columns)

# Print the encoded

print(df\_encoded)

Output:



**Ass: (Without Using Data frame)**

# Program for demonstration of one hot encoding

# import libraries

import numpy as np

import pandas as pd

# import the data required

data = pd.read\_csv('Candy\_Sales.csv')

print(data.head())

**Output:**

