data = [115.3, 195.5, 120.5, 110.2, 90.4, 105.6, 110.9, 116.3, 122.3, 125.4]

# Mean

n = len(data)

mean = sum(data) / n

# Median

sorted\_data = sorted(data)

mid = n // 2

median = (sorted\_data[mid] + sorted\_data[mid - 1]) / 2 if n % 2 == 0 else sorted\_data[mid]

# Mode

freq = {}

for num in data:

    freq[num] = freq.get(num, 0) + 1

max\_freq = max(freq.values())

mode = "No mode" if max\_freq == 1 else print(max\_freq)

# Variance and Standard Deviation

variance = sum((x - mean) \*\* 2 for x in data) / n

std\_dev = variance \*\* 0.5

# Min-Max Normalization

min\_val, max\_val = min(data), max(data)

min\_max\_norm = [(x - min\_val) / (max\_val - min\_val) for x in data]

# Standardization (Z-score)

standardized = [(x - mean) / std\_dev for x in data]

# Print results

print("Mean:", round(mean, 2))

print("Median:", median)

print("Mode:", mode)

print("Variance:", round(variance, 4))

print("Standard Deviation:", round(std\_dev, 4))

print("Min-Max Normalized:", [round(x, 3) for x in min\_max\_norm])

print("Standardized (Z-score):", [round(x, 3) for x in standardized])

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

from sklearn.datasets import load\_iris

iris = load\_iris()

x = iris.data

y = iris.target

x\_std = (x - np.mean(x, axis=0)) / np.std(x, axis=0)

cov\_matrix = np.cov(x\_std, rowvar=False)

print("Covariance Matrix:\n", cov\_matrix)

eigenvalues, eigenvectors = np.linalg.eig(cov\_matrix)

print("\nEigenvalues:\n", eigenvalues)

print("\nEigenvectors:\n", eigenvectors)

pca = PCA(n\_components=2)

x\_pca = pca.fit\_transform(x\_std)

plt.figure(figsize=(8, 6))

for target, color in zip(range(3), ('r', 'g', 'b')):

    plt.scatter(x\_pca[y == target, 0], x\_pca[y == target, 1], color=color, label=iris.target\_names[target])

plt.xlabel('First Principal Component')

plt.ylabel('Second Principal Component')

plt.title('PCA of Iris Dataset')

plt.legend()

plt.grid(True)

plt.show()

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

X = np.array([

    [5.9, 3.2], [4.6, 2.9], [6.2, 2.8], [4.7, 3.2], [5.5, 4.2],

    [5.0, 3.0], [4.9, 3.1], [6.7, 3.1], [5.1, 3.8], [6.0, 3.0]

])

init = np.array([[6.2, 3.2], [6.6, 3.7], [6.5, 3.0]])

k1 = KMeans(n\_clusters=3, init=init, n\_init=1, max\_iter=1).fit(X)

print("Red after 1 iter:", np.round(k1.cluster\_centers\_[0], 3))

k2 = KMeans(n\_clusters=3, init=k1.cluster\_centers\_, n\_init=1, max\_iter=1).fit(X)

print("Green after 2 iter:", np.round(k2.cluster\_centers\_[1], 3))

kf = KMeans(n\_clusters=3, init=init, n\_init=1).fit(X)

print("Blue on converge:", np.round(kf.cluster\_centers\_[2], 3))

print("Iterations to converge:", kf.n\_iter\_)

plt.figure(figsize=(8, 6))

plt.scatter(X[:, 0], X[:, 1], c='gray', s=100, label='Data Points')

plt.scatter(k1.cluster\_centers\_[:, 0], k1.cluster\_centers\_[:, 1],

            c='red', marker='X', s=150, label='After 1 Iteration')

plt.scatter(k2.cluster\_centers\_[:, 0], k2.cluster\_centers\_[:, 1],

            c='green', marker='^', s=150, label='After 2 Iterations')

plt.scatter(kf.cluster\_centers\_[:, 0], kf.cluster\_centers\_[:, 1],

            c='blue', marker='D', s=150, label='Final Centroids')

plt.title('KMeans Clustering - Iterative Centroid Updates')

plt.legend()

plt.grid(True)

plt.show()

import pandas as pd

import matplotlib.pyplot as plt

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

from sklearn.tree import DecisionTreeClassifier, plot\_tree

from sklearn.metrics import confusion\_matrix, accuracy\_score, recall\_score, precision\_score

zoo = pd.read\_csv('/content/DecisionTree\_zoo.csv')

x = zoo.drop(columns=['animal\_name', 'class\_type'])

y = zoo['class\_type']

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, random\_state=60)

model = DecisionTreeClassifier(criterion='entropy')

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

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

cm = confusion\_matrix(y\_test, y\_pred)

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

precision = precision\_score(y\_test, y\_pred, average='weighted')

recall = recall\_score(y\_test, y\_pred, average='macro')

print("Confusion Matrix:\n", cm)

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

print(f"Precision (Weighted): {precision:.4f}")

print(f"Recall (Macro): {recall:.4f}")

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

plot\_tree(model, feature\_names=x.columns, filled=True)

plt.show()

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

data = pd.read\_csv('/content/linear\_Salary\_Data (1).csv')

plt.scatter(data['YearsExperience'], data['Salary'], color='blue')

plt.title('Salary vs Experience')

plt.xlabel('Years of Experience')

plt.ylabel('Salary')

plt.grid(True)

plt.show()

print("Correlation Matrix:\n", data.corr())

x = data['YearsExperience'].values

y = data['Salary'].values

m = sum((x - np.mean(x)) \* (y - np.mean(y))) / sum((x - np.mean(x)) \*\* 2)

c = np.mean(y) - m \* np.mean(x)

print(f"\nSlope (m): {m:.2f}")

print(f"Intercept (c): {c:.2f}")

y\_pred = m \* x + c

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

plt.plot(x, y\_pred, color='red', label='Regression Line')

plt.title('Linear Regression Fit')

plt.xlabel('Years of Experience')

plt.ylabel('Salary')

plt.legend()

plt.grid(True)

plt.show()

sse = sum((y - y\_pred) \*\* 2)

ssr = sum((y\_pred - np.mean(y)) \*\* 2)

sst = sse + ssr

cost = sse / len(x)

r2 = ssr / sst

print(f"\nSSE (Sum of Squared Errors): {sse:.2f}")

print(f"SSR (Sum of Squares due to Regression): {ssr:.2f}")

print(f"SST (Total Sum of Squares): {sst:.2f}")

print(f"Cost (MSE): {cost:.2f}")

print(f"R² Score: {r2:.3f}")

import numpy as np

import pandas as pd

from sklearn.datasets import load\_iris

import matplotlib.pyplot as plt

iris= load\_iris()

x = iris.data

y = iris.target

x = (x-np.mean(x,axis=0))/np.std(x,axis=0)

x= np.c\_[np.ones(x.shape[0]),x]

y\_onehot = np.eye(3)[y]

def softmax(z):

  exp\_z = np.exp(z-np.max(z,axis=1,keepdims=True))

  return exp\_z/np.sum(exp\_z,axis=1,keepdims=True)

theta = np.zeros((x.shape[1],3))

lr=0.1

loss\_history = []

for i in range(1000):

    z = x @ theta

    h = softmax(z)

    loss = -np.mean(y\_onehot \* np.log(h))

    loss\_history.append(loss)

    grad = x.T @ (h - y\_onehot) / x.shape[0]

    theta -= lr \* grad

plt.figure()

plt.plot(loss\_history)

plt.title('Gradient Descent Convergence')

plt.xlabel('Epoch')

plt.ylabel('Loss (Cross-Entropy)')

plt.show()

z = x @ theta

h = softmax(z)

y\_pred = np.argmax(h, axis=1)

accuracy = np.mean(y\_pred == y)

print(f"Final Parameters:\n{theta}")

print(f"Training Accuracy: {accuracy:.2%}")

#importing libraries

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.datasets  import load\_iris

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

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

#load datasets

data=load\_iris()

X=data.data

y=data.target

target\_names=data.target\_names

#train and test data

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

#model

kernels = ['rbf','poly','sigmoid']

models={}

accuracies={}

for kernel in kernels:

  model=SVC(kernel=kernel, gamma="scale")

  model.fit(X\_train,y\_train)

  y\_pred=model.predict(X\_test)

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

  models[kernel]=model

  accuracies[kernel]=acc

  print(f"\n=======kernel: {kernel.upper()}========")

  print(f"Accuracy: {acc:.2f}")

  print("Classification Report:")

  print(classification\_report(y\_test,y\_pred,target\_names=target\_names))

  print("Confusion Matrix:")

  print(confusion\_matrix(y\_test,y\_pred))

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

sns.barplot(x=list(accuracies.keys()),y=list(accuracies.values()),palette="Set2")

plt.title("Model Accuracy Comparision")

plt.ylabel("Accuracy")

plt.ylim(0.1,1)

plt.show()

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.datasets import load\_breast\_cancer

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

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

#load dataset

data=load\_breast\_cancer()

X=data.data

y=data.target

#split data

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

#DecisionTreeClassifier

dt\_model=DecisionTreeClassifier()

dt\_model.fit(X\_train,y\_train)

y\_pred\_dt=dt\_model.predict(X\_test)

#RandomForestClassifier

rf\_model=RandomForestClassifier()

rf\_model.fit(X\_train,y\_train)

y\_pred\_rf=rf\_model.predict(X\_test)

#Accuracy Scores

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

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

print(f"Decision Tree Accuracy: {acc\_dt:.4f}")

print(f"Random Forest Accuracy: {acc\_rf:.4f}")

#Classification Report

print("\n ------Decision Tree Report------")

print(classification\_report(y\_test,y\_pred\_dt))

print("\n------random Forest Report------")

print(classification\_report(y\_test,y\_pred\_rf))

#Confusion Matrix

plt.figure(figsize=(12,5))

plt.subplot(1,2,1)

sns.heatmap(confusion\_matrix(y\_test,y\_pred\_dt),annot=True, cmap="Blues")

plt.title("Decision Tree Confusion Matrix")

plt.subplot(1,2,2)

sns.heatmap(confusion\_matrix(y\_test,y\_pred\_rf),annot=True, cmap="Greens")

plt.title("Random Forest Confusion Matrix")

plt.tight\_layout()

plt.show()

#Accuracy Score Plot

plt.figure(figsize=(6,4))

sns.barplot(x=["Decision Tree","Random forest"],y=[acc\_dt,acc\_rf],palette="Set2")

plt.title("Model Accuracy Comparision")

plt.ylabel("Accuracy")

plt.ylim(0.85,1.0)

plt.show()

#importing libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.model\_selection import train\_test\_split, validation\_curve

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import (accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_curve,auc)

#load datasets

data=pd.read\_csv("/content/covid(For Naive Bayes Program) (1).csv")

data=data.drop(columns=["no"])

X=data.drop(columns=["diagnosis"]).apply(lambda col: pd.factorize(col)[0])

y=pd.factorize(data["diagnosis"])[0]

#train test and split

X\_train, X\_test, y\_train, y\_test=train\_test\_split(X,y,test\_size=0.2,stratify=y,random\_state=40)

#model train

model=GaussianNB()

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

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

y\_proba=model.predict\_proba(X\_test)[:,1]

#print

print("Accuracy:", accuracy\_score(y\_test,y\_pred))

print("Precision Score:", precision\_score(y\_test,y\_pred))

print("Recall Score:", recall\_score(y\_test,y\_pred))

print("F1 Score:", f1\_score(y\_test,y\_pred))

#ROC curve

fpr, tpr,\_=roc\_curve(y\_test,y\_proba)

plt.figure(figsize=(6,4))

plt.plot(fpr,tpr,label="ROC curve (AUC {:.2f})".format(auc(fpr,tpr)))

plt.plot([0,1],[0,1],"--",color="gray")

plt.title("ROC Curve")

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

param\_range=np.logspace(-9,0,10)

train\_scores,test\_scores=validation\_curve(GaussianNB(),X,y,param\_name="var\_smoothing",param\_range=param\_range,cv=5, scoring="accuracy")

plt.figure(figsize=(6,4))

plt.semilogx(param\_range,train\_scores.mean(axis=1),label="train", marker='o')

plt.semilogx(param\_range,test\_scores.mean(axis=1),label="test", marker='s')

plt.title("Validation Curve")

plt.xlabel("var\_smoothing")

plt.ylabel("Accuracy")

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()