**1. Implement a Linear Regression model to forecast salary packages using student CGPA data, and evaluate the applied model's performance.**

**SOURCE CODE:-**

**from sklearn.linear\_model import LinearRegression**

**from sklearn.metrics import mean\_squared\_error, r2\_score**

**import matplotlib.pyplot as plt**

**n = int(input("Enter number of students: "))**

**cgpa = []**

**salary = []**

**for i in range(n):**

**c = float(input(f"Enter CGPA of student {i+1}: "))**

**s = float(input(f"Enter Salary of student {i+1} (in LPA): "))**

**cgpa.append([c])**

**salary.append(s)**

**model = LinearRegression()**

**model.fit(cgpa, salary)**

**test\_cgpa = float(input("Enter CGPA to predict salary: "))**

**pred = model.predict([[test\_cgpa]])**

**print("Predicted Salary:", round(pred[0], 2), "LPA")**

**pred\_all = model.predict(cgpa)**

**print("MSE:", mean\_squared\_error(salary, pred\_all))**

**print("R² Score:", round(r2\_score(salary, pred\_all), 3))**

**x\_vals = [c[0] for c in cgpa]**

**plt.scatter(x\_vals, salary, color='blue', label='Actual')**

**plt.plot(x\_vals, pred\_all, color='red', label='Predicted Line')**

**plt.scatter([test\_cgpa], pred, color='green', label='User Prediction')**

**plt.xlabel("CGPA")**

**plt.ylabel("Salary (LPA)")**

**plt.legend()**

**plt.title("CGPA vs Salary Prediction")**

**plt.show()**

**SAMPLE INPUT/OUTPUT :-**

**Enter number of students: 3**

**Enter CGPA of student 1: 7**

**Enter Salary of student 1 (in LPA): 4**

**Enter CGPA of student 2: 8**

**Enter Salary of student 2 (in LPA): 5**

**Enter CGPA of student 3: 9**

**Enter Salary of student 3 (in LPA): 6**

**Enter CGPA to predict salary: 8**

**Predicted Salary: 5.0 LPA**

**MSE: 0.0**

**R² Score: 1.0**

**2. Implement Hypothesis Testing and Validation in Predicting Student Exam Scores using a Multivariate Regression model.**

**SOURCE CODE :-**

**import pandas as pd**

**import statsmodels.api as sm**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

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

**from sklearn.linear\_model import LinearRegression**

**from sklearn.metrics import mean\_squared\_error, r2\_score**

**url\_mat = '**[**https://raw.githubusercontent.com/arunk13/MSDA-Assignments/master/IS607Fall2015/Assignment3/student-mat.csv**](https://raw.githubusercontent.com/arunk13/MSDA-Assignments/master/IS607Fall2015/Assignment3/student-mat.csv)**'**

**df\_mat = pd.read\_csv(url\_mat, sep=';')**

**selected\_columns = ['G1', 'G2', 'studytime', 'failures', 'absences', 'G3']**

**df\_selected = df\_mat[selected\_columns]**

**correlations = df\_selected.corr()['G3'].drop('G3')**

**plt.figure(figsize=(8, 5))**

**correlations.sort\_values().plot(kind='barh', color='skyblue')**

**plt.title("Feature Correlation with Final Grade (G3)")**

**plt.xlabel("Correlation Coefficient")**

**plt.grid(True)**

**plt.tight\_layout()**

**plt.show()**

**X = df\_selected.drop(columns='G3')**

**y = df\_selected['G3']**

**X = pd.get\_dummies(X, drop\_first=True)**

**X\_sm = sm.add\_constant(X).astype(float)**

**y = y.astype(float)**

**ols = sm.OLS(y, X\_sm).fit()**

**print(ols.summary())**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)**

**lr = LinearRegression().fit(X\_train, y\_train)**

**y\_pred = lr.predict(X\_test)**

**print(f"\nValidation Results:")**

**print(f"MSE: {mean\_squared\_error(y\_test, y\_pred):.3f}")**

**print(f"R²: {r2\_score(y\_test, y\_pred):.3f}")**

**print("\nPredicted G3 Scores for Test Data:")**

**print(y\_pred)**

**comparison = pd.DataFrame({'Actual G3': y\_test.values, 'Predicted G3': y\_pred})**

**print(comparison.head())**

**3. Develop a spam classifier using Naive Bayes model to classify emails as spam or not spam from the spam\_or\_not\_spam dataset and demonstrate the model’s accuracy.**

**SOURCE CODE:-**

**import pandas as pd**

**import requests**

**import io**

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

**from sklearn.naive\_bayes import MultinomialNB**

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

**import matplotlib.pyplot as plt**

**import seaborn as sns**

**url = "https://archive.ics.uci.edu/ml/machine-learning-databases/spambase/spambase.data"**

**try:**

**response = requests.get(url)**

**response.raise\_for\_status()**

**data = io.StringIO(response.text)**

**df = pd.read\_csv(data, header=None)**

**except requests.exceptions.RequestException as e:**

**print("Error fetching data:", e)**

**exit()**

**df.columns = list(range(57)) + ['spam']**

**X = df.drop('spam', axis=1)**

**y = df['spam']**

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

**X, y, test\_size=0.2, random\_state=42, stratify=y**

**)**

**model = MultinomialNB()**

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

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

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

**print("\nClassification Report:")**

**print(classification\_report(y\_test, y\_pred, target\_names=["non-spam", "spam"]))**

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

**plt.figure(figsize=(6, 5))**

**sns.heatmap(cm, annot=True, fmt="d", cmap="Greens",**

**xticklabels=["non-spam", "spam"],**

**yticklabels=["non-spam", "spam"])**

**plt.xlabel("Predicted")**

**plt.ylabel("Actual")**

**plt.title("Naive Bayes Spam Classifier - Confusion Matrix")**

**plt.tight\_layout()**

**plt.show()**

**4. Implement the K-Nearest Neighbors (KNN) algorithm to classify handwritten digits in a MNIST dataset, and evaluate the model's performance in accuracy.**

**SOURCE CODE 1:-**

**from sklearn.datasets import load\_digits**

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

**from sklearn.neighbors import KNeighborsClassifier**

**from sklearn.metrics import accuracy\_score**

**import matplotlib.pyplot as plt**

**digits = load\_digits()**

**plt.gray()**

**plt.matshow(digits.images[1])**

**plt.title(f"Label: {digits.target[1]}")**

**plt.show()**

**X = digits.data**

**y = digits.target**

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

**knn = KNeighborsClassifier(n\_neighbors=3)**

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

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

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

**print("Model Accuracy:", accuracy)**

**SOURCE CODE 2:-**

**import numpy as np**

**import matplotlib.pyplot as plt**

**from sklearn.datasets import fetch\_openml**

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

**from sklearn.neighbors import KNeighborsClassifier**

**from sklearn.metrics import accuracy\_score**

**mnist = fetch\_openml('mnist\_784', version=1, cache=True, as\_frame=False, parser='auto')**

**X, y = mnist["data"], mnist["target"]**

**y = y.astype(np.uint8)**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)**

**k\_values = [1, 3, 5, 7, 9, 11]**

**accuracies = []**

**for k in k\_values:**

**knn\_model = KNeighborsClassifier(n\_neighbors=k, n\_jobs=-1)**

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

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

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

**accuracies.append(accuracy)**

**print(f"Accuracy for k={k}: {accuracy:.4f}")**

**plt.figure(figsize=(10, 6))**

**plt.plot(k\_values, accuracies, marker='o', linestyle='-')**

**plt.title('KNN Accuracy vs. Number of Neighbors (k)')**

**plt.xlabel('Number of Neighbors (k)')**

**plt.ylabel('Accuracy')**

**plt.xticks(k\_values)**

**plt.grid(True)**

**plt.show()**

**sample\_index = np.random.randint(0, len(X\_test))**

**sample\_image = X\_test[sample\_index].reshape(28, 28)**

**true\_label = y\_test[sample\_index]**

**best\_k\_index = np.argmax(accuracies)**

**best\_k = k\_values[best\_k\_index]**

**best\_accuracy = accuracies[best\_k\_index]**

**knn\_final = KNeighborsClassifier(n\_neighbors=best\_k, n\_jobs=-1)**

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

**predicted\_label = knn\_final.predict([X\_test[sample\_index]])[0]**

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

**plt.imshow(sample\_image, cmap='gray')**

**plt.title(f"True Label: {true\_label}\nPredicted Label: {predicted\_label}")**

**plt.axis('off')**

**plt.show()**

**plt.figure(figsize=(6, 5))**

**plt.bar(['Final Accuracy'], [best\_accuracy], color='skyblue')**

**plt.ylim(0.9, 1.0)**

**plt.title(f'Final Model Accuracy (k={best\_k})')**

**plt.ylabel('Accuracy')**

**plt.grid(axis='y', linestyle='--', alpha=0.7)**

**plt.show()**

**5. Implement and evaluate the performance of a Support Vector Machine (SVM) model to classify brain tumors from MRI images, and assess its accuracy.**

[**22AI503 - ML Exp-5 brain-tumor-detection.zip**](https://drive.google.com/open?id=1RC5G8KqJ23w7lqmpI7geGlhLR_eulnhe&usp=drive_copy)

**SOURCE CODE:-**

**import os**

**import cv2**

**import numpy as np**

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

**from sklearn.svm import SVC**

**from sklearn.metrics import accuracy\_score**

**import matplotlib.pyplot as plt**

**def load\_images\_from\_folder(folder, label):**

**images = []**

**labels = []**

**for filename in os.listdir(folder):**

**img\_path = os.path.join(folder, filename)**

**img = cv2.imread(img\_path, cv2.IMREAD\_GRAYSCALE)**

**if img is not None:**

**img = cv2.resize(img, (100, 100))**

**images.append(img.flatten())**

**labels.append(label)**

**return images, labels**

**tumor\_imgs, tumor\_labels = load\_images\_from\_folder(r" insert folder’s path here for brain tumor exists ", 1)**

**no\_tumor\_imgs, no\_tumor\_labels = load\_images\_from\_folder(r" insert folder’s path here for brain tumor not exists ", 0)**

**X = np.array(tumor\_imgs + no\_tumor\_imgs)**

**y = np.array(tumor\_labels + no\_tumor\_labels)**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42, stratify=y)**

**svm\_model = SVC(kernel='linear')**

**svm\_model.fit(X\_train, y\_train)**

**y\_pred = svm\_model.predict(X\_test)**

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

**print("Model Accuracy:", accuracy)**

**sample\_index = 1**

**plt.imshow(X\_test[sample\_index].reshape(100, 100), cmap='gray')**

**plt.title(f"Predicted: {y\_pred[sample\_index]}, Actual: {y\_test[sample\_index]}")**

**plt.axis('off')**

**plt.show()**

**6. Implement the K-Means clustering algorithm to identify the optimal number of clusters for segmenting social media users into similar communities, and evaluate its metrics.**

**SOURCE CODE:-**

**import numpy as np**

**import pandas as pd**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

**from sklearn.cluster import KMeans**

**from sklearn.preprocessing import StandardScaler**

**from sklearn.metrics import silhouette\_score**

**data = {**

**'Age': np.random.randint(18, 55, 100),**

**'Posts\_Per\_Week': np.random.randint(1, 50, 100),**

**'Likes\_Per\_Week': np.random.randint(20, 500, 100),**

**'Shares\_Per\_Week': np.random.randint(5, 100, 100)**

**}**

**df = pd.DataFrame(data)**

**scaler = StandardScaler()**

**scaled\_data = scaler.fit\_transform(df)**

**wcss = []**

**for k in range(1, 11):**

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

**kmeans.fit(scaled\_data)**

**wcss.append(kmeans.inertia\_)**

**plt.plot(range(1, 11), wcss, marker='o')**

**plt.title('Elbow Method for Optimal k')**

**plt.xlabel('Number of clusters (k)')**

**plt.ylabel('WCSS')**

**plt.grid(True)**

**plt.show()**

**optimal\_k = 3**

**kmeans = KMeans(n\_clusters=optimal\_k, random\_state=42)**

**clusters = kmeans.fit\_predict(scaled\_data)**

**df['Cluster'] = clusters**

**score = silhouette\_score(scaled\_data, clusters)**

**print("Silhouette Score:", score)**

**sns.scatterplot(data=df, x='Age', y='Posts\_Per\_Week',**

**hue='Cluster', palette='Set2')**

**plt.title('User Clusters Based on Age vs Posts')**

**plt.show()**