**Aim 1. Implement Decision Tree Model using Iris dataset using Python/R and interpret decision rules of classification.**

# Import necessary libraries

from sklearn.datasets import load\_iris

from sklearn.tree import DecisionTreeClassifier, export\_text, plot\_tree

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

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

import matplotlib.pyplot as plt

# Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

# Split the dataset into train and test sets

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

# Create and train the Decision Tree model

clf = DecisionTreeClassifier(criterion='entropy', random\_state=42)

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

# Predict on test data

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

# Print accuracy and classification report

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

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred, target\_names=iris.target\_names))

# Print decision rules

print("\nDecision Tree Rules:\n")

tree\_rules = export\_text(clf, feature\_names=iris.feature\_names)

print(tree\_rules)

# Plot the decision tree

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

plot\_tree(clf, filled=True, feature\_names=iris.feature\_names, class\_names=iris.target\_names)

plt.title("Decision Tree for Iris Dataset")

plt.show()

**Aim 2. Load Iris Dataset. Apply K-means Algorithm using Python/R to group similar data points into clusters. Determine optimal number of clusters using Elbow Method. Visualize clustering results and analyze cluster characteristics.**

# Import necessary libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.cluster import KMeans

from sklearn.decomposition import PCA

# Load Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

feature\_names = iris.feature\_names

# Create a DataFrame

df = pd.DataFrame(X, columns=feature\_names)

# Elbow Method to find the optimal number of clusters

wcss = [] # within-cluster sum of squares

for i in range(1, 11):

kmeans = KMeans(n\_clusters=i, init='k-means++', random\_state=42)

kmeans.fit(X)

wcss.append(kmeans.inertia\_)

# Plot the Elbow Graph

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

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

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

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

plt.ylabel('WCSS')

plt.grid(True)

plt.show()

# Apply KMeans with optimal number of clusters (e.g., 3 from Elbow graph)

kmeans = KMeans(n\_clusters=3, init='k-means++', random\_state=42)

y\_kmeans = kmeans.fit\_predict(X)

# Add cluster info to DataFrame

df['Cluster'] = y\_kmeans

# Visualize Clusters using PCA (2D projection)

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X)

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

for cluster in np.unique(y\_kmeans):

plt.scatter(X\_pca[y\_kmeans == cluster, 0], X\_pca[y\_kmeans == cluster, 1],

label=f'Cluster {cluster}')

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

s=200, c='black', label='Centroids', marker='X')

plt.title('K-Means Clustering on Iris (PCA-reduced)')

plt.xlabel('PCA Component 1')

plt.ylabel('PCA Component 2')

plt.legend()

plt.grid(True)

plt.show()

# Analyze Cluster Characteristics

print("\nCluster Centers (original feature space):")

centroids\_df = pd.DataFrame(kmeans.cluster\_centers\_, columns=feature\_names)

print(centroids\_df)

print("\nNumber of samples in each cluster:")

print(df['Cluster'].value\_counts())

**Aim 3. Load Iris Dataset. Apply K-means Algorithm using Python/R to group similar data points into clusters. Determine optimal number of clusters using Silhouette analysis. Visualize clustering results and analyze cluster characteristics.**

# Import necessary libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette\_score

from sklearn.decomposition import PCA

# Load the Iris dataset

iris = load\_iris()

X = iris.data

feature\_names = iris.feature\_names

# Create a DataFrame

df = pd.DataFrame(X, columns=feature\_names)

# Silhouette analysis to determine the optimal number of clusters

silhouette\_scores = []

K\_range = range(2, 11) # Silhouette score is not defined for k=1

for k in K\_range:

kmeans = KMeans(n\_clusters=k, init='k-means++', random\_state=42)

labels = kmeans.fit\_predict(X)

score = silhouette\_score(X, labels)

silhouette\_scores.append(score)

# Plot silhouette scores

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

plt.plot(K\_range, silhouette\_scores, marker='o', color='green')

plt.title("Silhouette Analysis For Optimal k")

plt.xlabel("Number of Clusters (k)")

plt.ylabel("Silhouette Score")

plt.grid(True)

plt.show()

# Choose the best k (maximum silhouette score)

optimal\_k = K\_range[np.argmax(silhouette\_scores)]

print(f"\nOptimal number of clusters based on silhouette score: {optimal\_k}")

# Apply KMeans with the optimal number of clusters

kmeans = KMeans(n\_clusters=optimal\_k, init='k-means++', random\_state=42)

cluster\_labels = kmeans.fit\_predict(X)

# Add cluster labels to the DataFrame

df['Cluster'] = cluster\_labels

# Visualize clusters using PCA (2D projection)

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X)

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

for cluster in np.unique(cluster\_labels):

plt.scatter(X\_pca[cluster\_labels == cluster, 0], X\_pca[cluster\_labels == cluster, 1],

label=f'Cluster {cluster}')

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

s=200, c='black', label='Centroids', marker='X')

plt.title("K-Means Clustering on Iris Dataset (PCA-reduced)")

plt.xlabel("PCA Component 1")

plt.ylabel("PCA Component 2")

plt.legend()

plt.grid(True)

plt.show()

# Analyze cluster characteristics

print("\nCluster Centers (in original feature space):")

centroids\_df = pd.DataFrame(kmeans.cluster\_centers\_, columns=feature\_names)

print(centroids\_df)

print("\nNumber of samples in each cluster:")

print(df['Cluster'].value\_counts())

**Aim 4. Consider a scenario where you have test scores from a sample of students and you want to compare the mean of these scores with hypothesized population mean.**

**Student Score = [72, 88, 64, 74, 67, 79, 85, 75, 89,77]**

**Apply One Sampled T-Test using Python/R for above problem. Assume hypothesized mean as 70. Formulate Null and Alternative Hypothesis for a given problem. Interpret the results and draw the conclusion.**

from scipy import stats

# Sample student scores

student\_scores = [72, 88, 64, 74, 67, 79, 85, 75, 89, 77]

# Hypothesized population mean

mu = 70

# Formulate Hypotheses

print("Null Hypothesis (H0): The mean score = 70")

print("Alternative Hypothesis (H1): The mean score ≠ 70\n")

# Perform one-sample t-test

t\_statistic, p\_value = stats.ttest\_1samp(student\_scores, mu)

# Print results

print(f"T-Statistic: {t\_statistic:.4f}")

print(f"P-Value: {p\_value:.4f}")

# Interpret the result at 5% significance level

alpha = 0.05

if p\_value < alpha:

print("\nConclusion: Reject the null hypothesis.")

print("There is a statistically significant difference between the sample mean and the hypothesized mean.")

else:

print("\nConclusion: Fail to reject the null hypothesis.")

print("There is no statistically significant difference between the sample mean and the hypothesized mean.")

**Aim 5. Apply Feature Scaling technique like standardization and normalization using Python/R to Boston Housing dataset.**

import pandas as pd

from sklearn.preprocessing import StandardScaler, MinMaxScaler

import numpy as np

# Step 1: Load the Boston Housing dataset

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

# Display first few rows to understand the dataset structure

print("Original Data:")

print(df.head())

# Step 2: Apply Standardization (Z-score normalization)

# Standardization transforms the data to have mean 0 and variance 1

# Initialize the StandardScaler

scaler = StandardScaler()

# Assuming the dataset has all numerical features except 'target' (label), standardize the numerical columns

numerical\_columns = df.select\_dtypes(include=[np.number]).columns

df\_standardized = df.copy()

df\_standardized[numerical\_columns] = scaler.fit\_transform(df[numerical\_columns])

print("\nStandardized Data (Mean = 0, Variance = 1):")

print(df\_standardized.head())

# Step 3: Apply Normalization (Min-Max scaling)

# Normalization transforms the data to a range of [0, 1]

# Initialize the MinMaxScaler

minmax\_scaler = MinMaxScaler()

# Normalize the numerical columns

df\_normalized = df.copy()

df\_normalized[numerical\_columns] = minmax\_scaler.fit\_transform(df[numerical\_columns])

print("\nNormalized Data (Range [0, 1]):")

print(df\_normalized.head())

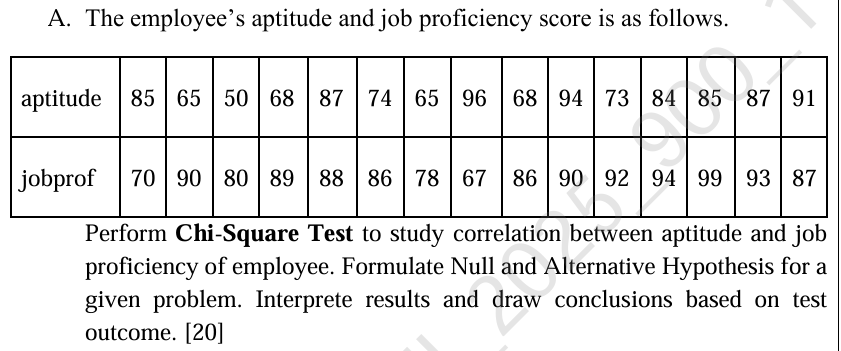
# Optional: Save the transformed datasets to new CSV files

df\_standardized.to\_csv('BostonHousing\_standardized.csv', index=False)

df\_normalized.to\_csv('BostonHousing\_normalized.csv', index=False)

print("\nFiles with standardized and normalized data have been saved.")

**Aim 6.**



import pandas as pd

from scipy.stats import chi2\_contingency

# Employee data from image

aptitude = [85, 65, 50, 68, 87, 74, 65, 96, 68, 94, 73, 84, 85, 87, 91]

jobprof = [70, 90, 80, 89, 88, 86, 78, 67, 86, 90, 92, 94, 99, 93, 87]

# Create a DataFrame

df = pd.DataFrame({'Aptitude': aptitude, 'JobProficiency': jobprof})

# Convert continuous data to categorical by binning (for chi-square)

df['Aptitude\_cat'] = pd.cut(df['Aptitude'], bins=3, labels=["Low", "Medium", "High"])

df['JobProf\_cat'] = pd.cut(df['JobProficiency'], bins=3, labels=["Low", "Medium", "High"])

# Create contingency table

contingency\_table = pd.crosstab(df['Aptitude\_cat'], df['JobProf\_cat'])

print("Contingency Table:")

print(contingency\_table)

# Apply Chi-Square Test

chi2, p, dof, expected = chi2\_contingency(contingency\_table)

print(f"\nChi-Square Statistic = {chi2}")

print(f"Degrees of Freedom = {dof}")

print(f"P-value = {p}")

# Interpretation

alpha = 0.05

if p < alpha:

print("\nConclusion: Reject Null Hypothesis — There is a significant relationship between aptitude and job proficiency.")

else:

print("\nConclusion: Fail to Reject Null Hypothesis — No significant relationship between aptitude and job proficiency.")

**Aim 7. Perform Logistic Regression on the Iris dataset using Python/R to predict binary outcome. Evaluate model’s performance using classification metrics.**

# Import required libraries

from sklearn.datasets import load\_iris

from sklearn.linear\_model import LogisticRegression

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

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

import pandas as pd

# Load Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

# Convert problem to binary classification (e.g., class 0 vs not class 0)

# Class 0 = Setosa, Classes 1 & 2 = Versicolor & Virginica

y\_binary = (y == 0).astype(int)

# Split data into training and testing sets

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

# Create and train Logistic Regression model

model = LogisticRegression()

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

# Make predictions

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

# Evaluate model performance

print("Confusion Matrix:")

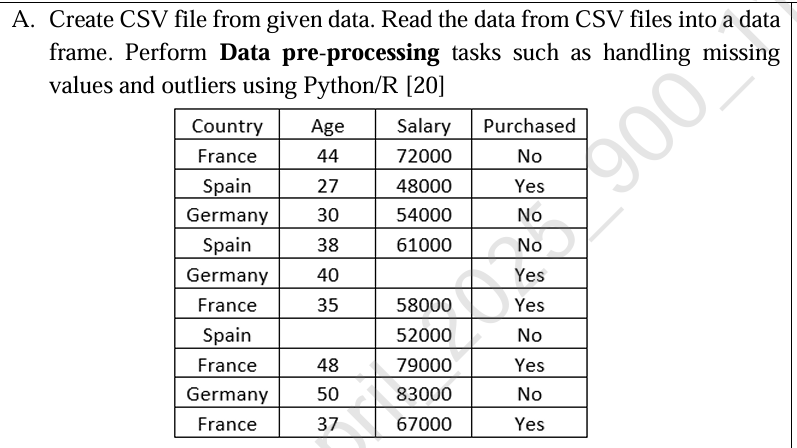
print(confusion\_matrix(y\_test, y\_pred))

print("\nClassification Report:")

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

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

**Aim 8.**



import pandas as pd

# Step 1: Create CSV from given data

data = {

'Country': ['France', 'Spain', 'Germany', 'Spain', 'Germany',

'France', 'Spain', 'France', 'Germany', 'France'],

'Age': [44, 27, 30, 38, 40, 35, 27, 48, 50, 37],

'Salary': [72000, 48000, 54000, 61000, 63777, 58000, 52000, 79000, 83000, 67000],

'Purchased': ['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes']

}

df = pd.DataFrame(data)

df.to\_csv('employee\_data.csv', index=False)

print("CSV file 'employee\_data.csv' created.\n")

# Step 2: Read CSV into DataFrame

df = pd.read\_csv('employee\_data.csv')

print("Original Data:")

print(df)

# Step 3: Introduce missing values for demonstration

df.loc[4, 'Age'] = None

df.loc[7, 'Salary'] = None

print("\nData with Missing Values:")

print(df)

# Step 4: Handle Missing Values

df['Age'].fillna(df['Age'].mean(), inplace=True)

df['Salary'].fillna(df['Salary'].median(), inplace=True)

print("\nAfter Handling Missing Values:")

print(df)

# Step 5: Detect and Remove Outliers (using IQR method on 'Salary')

Q1 = df['Salary'].quantile(0.25)

Q3 = df['Salary'].quantile(0.75)

IQR = Q3 - Q1

lower\_bound = Q1 - 1.5 \* IQR

upper\_bound = Q3 + 1.5 \* IQR

print(f"\nOutlier Detection Range for Salary: {lower\_bound} to {upper\_bound}")

outliers = df[(df['Salary'] < lower\_bound) | (df['Salary'] > upper\_bound)]

print("\nOutliers Detected in Salary:")

print(outliers)

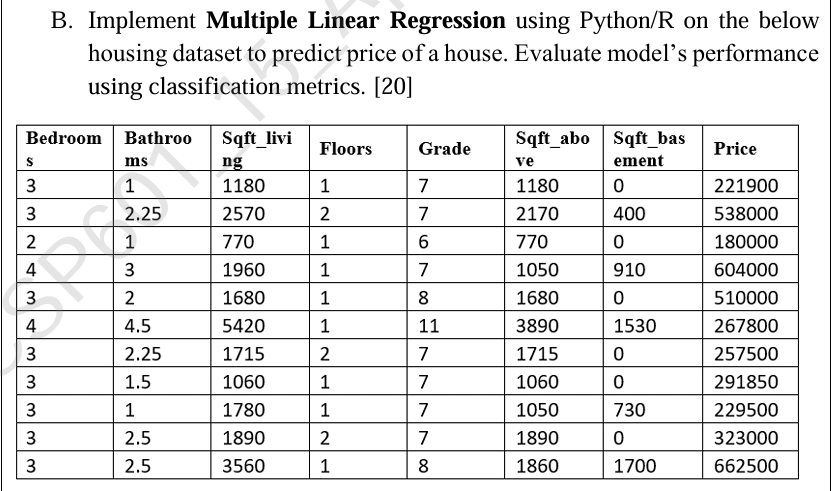
# Remove Outliers

df\_cleaned = df[(df['Salary'] >= lower\_bound) & (df['Salary'] <= upper\_bound)]

print("\nData After Removing Outliers:")

print(df\_cleaned)

**Aim 9.**



import pandas as pd

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

from sklearn.linear\_model import LinearRegression

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

# Step 1: Create the dataset

data = {

'Bedrooms': [3, 3, 2, 4, 3, 4, 3, 3, 1, 3],

'Bathrooms': [1, 2.25, 1, 1, 2, 4.5, 2.25, 1.5, 1, 2.5],

'Sqft\_living': [1180, 2570, 770, 1960, 1680, 5420, 1715, 1060, 1780, 3560],

'Floors': [1, 2, 1, 1, 1, 1, 2, 1, 1, 1],

'Grade': [7, 7, 6, 7, 8, 11, 7, 6, 7, 8],

'Sqft\_above': [1180, 2170, 770, 1050, 1680, 3890, 1715, 1060, 1050, 1860],

'Sqft\_basement': [0, 400, 0, 910, 0, 1530, 0, 0, 730, 1700],

'Price': [221900, 538000, 180000, 604000, 510000, 267800, 360000, 291850, 229500, 662500]

}

df = pd.DataFrame(data)

# Step 2: Define features and target

X = df.drop('Price', axis=1)

y = df['Price']

# Step 3: Split data into training and testing sets

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

# Step 4: Train the Multiple Linear Regression model

model = LinearRegression()

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

# Step 5: Make predictions

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

# Step 6: Evaluate the model

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

# Step 7: Print results

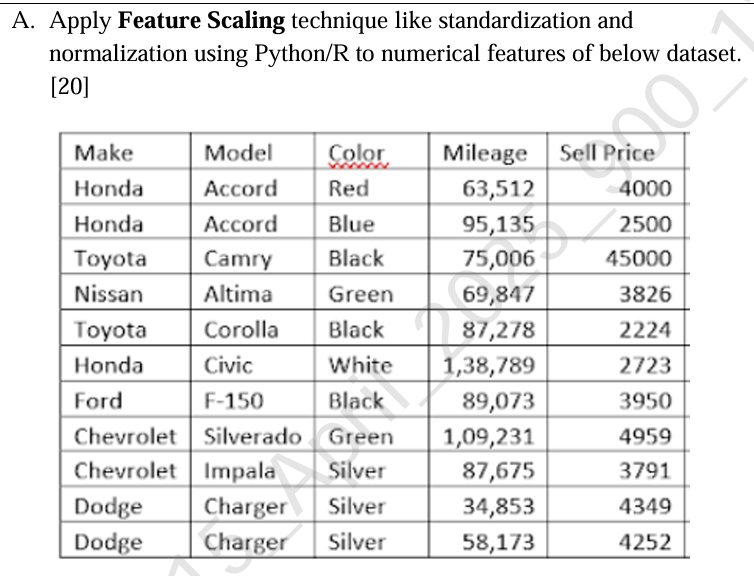
print("Actual Prices:\n", y\_test.values)

print("\nPredicted Prices:\n", y\_pred)

print("\nMean Squared Error:", round(mse, 2))

print("R-squared Score:", round(r2, 4))

**Aim 10.**

****

# Import necessary libraries

import pandas as pd

from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Define the dataset

data = {

'Make': ['Honda', 'Honda', 'Toyota', 'Nissan', 'Toyota', 'Honda', 'Ford', 'Chevrolet', 'Chevrolet', 'Dodge', 'Dodge'],

'Model': ['Accord', 'Accord', 'Camry', 'Altima', 'Corolla', 'Civic', 'F-150', 'Silverado', 'Impala', 'Charger', 'Charger'],

'Color': ['Red', 'Blue', 'Black', 'Green', 'Black', 'White', 'Black', 'Green', 'Silver', 'Silver', 'Silver'],

'Mileage': [63512, 95135, 75006, 69847, 87278, 138789, 89073, 109231, 87675, 34853, 58173],

'Sell Price': [4000, 2500, 45000, 3826, 2224, 2723, 3950, 4959, 3791, 4349, 4252]

}

# Convert to DataFrame

df = pd.DataFrame(data)

# Display the original dataset

print("Original Dataset:")

print(df)

# Select the numerical features (Mileage and Sell Price)

numerical\_features = ['Mileage', 'Sell Price']

# Standardization (Z-score normalization)

scaler\_standard = StandardScaler()

df\_standardized = df.copy()

df\_standardized[numerical\_features] = scaler\_standard.fit\_transform(df[numerical\_features])

# Normalization (Min-Max scaling)

scaler\_minmax = MinMaxScaler()

df\_normalized = df.copy()

df\_normalized[numerical\_features] = scaler\_minmax.fit\_transform(df[numerical\_features])

# Display the results

print("\nStandardized Dataset (Z-score normalization):")

print(df\_standardized)

print("\nNormalized Dataset (Min-Max scaling):")

print(df\_normalized)

**Aim 11. Implement Multiple Linear Regression on the “Pima Indian Diabetes dataset” using Python/R.**

import pandas as pd

import numpy as np

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

from sklearn.linear\_model import LinearRegression

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

# Step 1: Load the dataset

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

# Step 2: Display the first few rows

print("Dataset Preview:")

print(df.head())

# Step 3: Define Features (X) and Target (y)

X = df.drop('Outcome', axis=1) # Features (independent variables)

y = df['Outcome'] # Target (dependent variable)

# Step 4: Split the data into training and testing sets

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

# Step 5: Create and train the Linear Regression model

model = LinearRegression()

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

# Step 6: Predict on test data

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

# Step 7: Evaluate the model

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print("\nModel Coefficients:")

print(model.coef\_)

print("\nModel Intercept:")

print(model.intercept\_)

print("\nMean Squared Error:", mse)

print("R-squared Score:", r2)

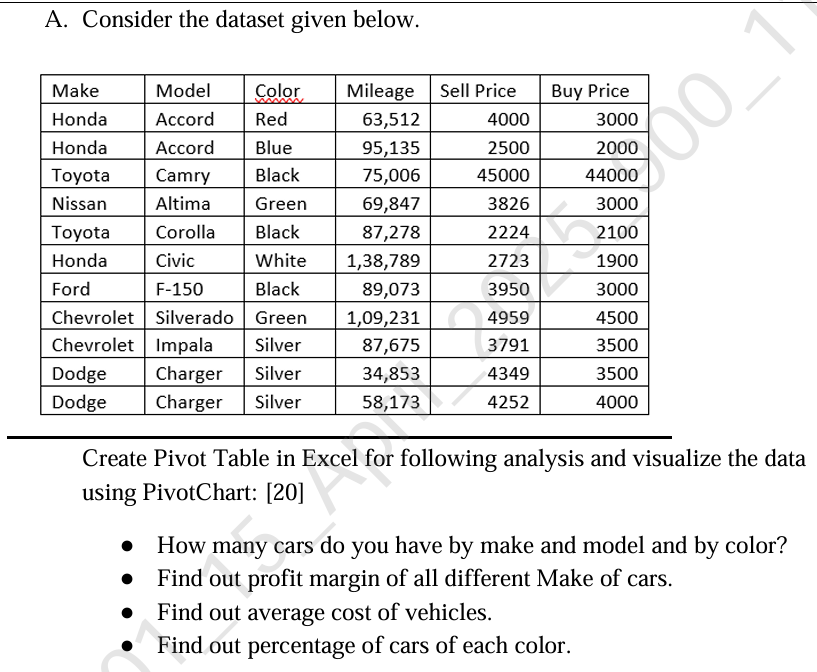
# Step 8: Optional - Show predicted vs actual values

comparison = pd.DataFrame({'Actual': y\_test, 'Predicted': np.round(y\_pred, 2)})

print("\nActual vs Predicted Outcomes:")

print(comparison.head())

**Aim 12.**

****

**Step-by-Step Guide:**

**1. Prepare the Data**

* **Open Excel** and input your dataset into the spreadsheet. Here's how your dataset should look in Excel:

| **Make** | **Model** | **Color** | **Mileage** | **Sell Price** | **Buy Price** |
| --- | --- | --- | --- | --- | --- |
| Honda | Accord | Red | 63,512 | 4000 | 3000 |
| Honda | Accord | Blue | 95,135 | 2500 | 2000 |
| Toyota | Camry | Black | 75,006 | 45000 | 44000 |
| Nissan | Altima | Green | 69,847 | 3826 | 3000 |
| Toyota | Corolla | Black | 87,278 | 2224 | 2100 |
| Honda | Civic | White | 138,789 | 2723 | 1900 |
| Ford | F-150 | Black | 89,073 | 3950 | 3000 |
| Chevrolet | Silverado | Green | 109,231 | 4959 | 4500 |
| Chevrolet | Impala | Silver | 87,675 | 3791 | 3500 |
| Dodge | Charger | Silver | 34,853 | 4349 | 3500 |
| Dodge | Charger | Silver | 58,173 | 4252 | 4000 |

**2. Create Pivot Table: Count the Number of Cars by Make, Model, and Color**

* **Select your dataset** (including headers).
* **Go to the "Insert" tab** in the ribbon.
* Click **PivotTable**. A dialog box will appear asking you to select the data range.
* Make sure the selected range is correct and choose to place the Pivot Table in a **New Worksheet**.
* Click **OK**.

Now, follow these steps to create the Pivot Table:

* In the **PivotTable Field List** on the right:
  + **Drag "Make" to Rows**.
  + **Drag "Model" under "Make"** (this will group models under each make).
  + **Drag "Color" to Columns** (this will break the data down by car color).
  + **Drag "Make" again to Values**. It should show "Count of Make," which counts how many cars there are for each make and model.

**Result:**

This will give you a Pivot Table showing the count of cars grouped by Make, Model, and Color.

**3. Calculate the Profit Margin for Each Make of Car**

* **Add a new column in your dataset** called "Profit Margin" where you calculate the profit margin for each row:
  + Formula for Profit Margin:  
    Profit Margin = (Sell Price - Buy Price) / Buy Price
* Enter this formula in the new column for each row. It should look like this:

| **Make** | **Model** | **Color** | **Mileage** | **Sell Price** | **Buy Price** | **Profit Margin** |
| --- | --- | --- | --- | --- | --- | --- |
| Honda | Accord | Red | 63,512 | 4000 | 3000 | 0.33 |
| Honda | Accord | Blue | 95,135 | 2500 | 2000 | 0.25 |
| Toyota | Camry | Black | 75,006 | 45000 | 44000 | 0.023 |
| Nissan | Altima | Green | 69,847 | 3826 | 3000 | 0.28 |
| Toyota | Corolla | Black | 87,278 | 2224 | 2100 | 0.059 |
| Honda | Civic | White | 138,789 | 2723 | 1900 | 0.43 |
| Ford | F-150 | Black | 89,073 | 3950 | 3000 | 0.32 |
| Chevrolet | Silverado | Green | 109,231 | 4959 | 4500 | 0.10 |
| Chevrolet | Impala | Silver | 87,675 | 3791 | 3500 | 0.083 |
| Dodge | Charger | Silver | 34,853 | 4349 | 3500 | 0.24 |
| Dodge | Charger | Silver | 58,173 | 4252 | 4000 | 0.063 |

* **Create a Pivot Table** for the profit margin:
  + **Insert a new PivotTable**.
  + **Drag "Make" to Rows**.
  + **Drag "Profit Margin" to Values**. It will automatically calculate the **Average** profit margin for each make.

**Result:**

This will give you the average **Profit Margin** for each make of car.

**4. Find the Average Cost of Vehicles**

* To calculate the average cost, you'll need to create a Pivot Table that shows the **average of the "Buy Price"** for each make:
  + **Insert a new PivotTable**.
  + **Drag "Make" to Rows**.
  + **Drag "Buy Price" to Values** and set the calculation to **Average** (right-click the value and select **Summarize Values By → Average**).

**Result:**

This will give you the average **Buy Price** (cost) of vehicles for each make.

**5. Calculate the Percentage of Cars of Each Color**

* **Create a Pivot Table** that shows the count of cars by **Color**.
  + **Insert a new Pivot Table**.
  + **Drag "Color" to Rows**.
  + **Drag "Make" to Values** and set it to **Count**.
* To calculate the percentage, you'll need to use the **Calculated Field** feature:
  + **Click on your Pivot Table** and go to the **PivotTable Analyze** tab.
  + Click **Fields, Items & Sets**, then **Calculated Field**.
  + In the formula area, type:
    - Formula: = (Count of Make) / Total Count of Cars
    - For "Total Count of Cars," you can either manually count the total cars or use a formula.
  + Click **OK**.

**Result:**

This will give you the percentage of cars for each color.

**6. Create PivotChart**

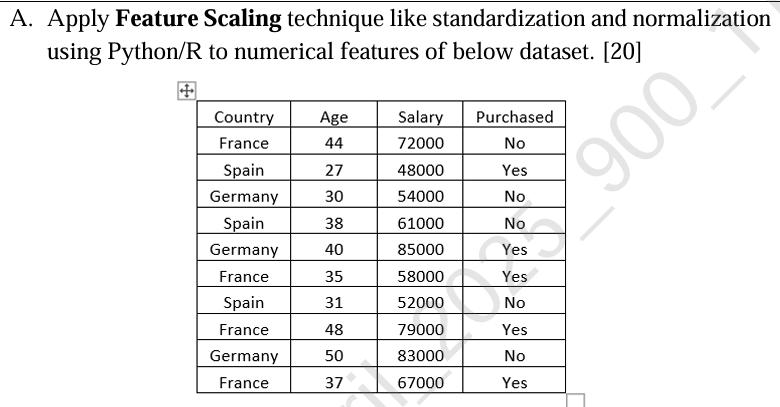
To visualize your data with PivotCharts:

* **Click on your Pivot Table**.
* Go to the **Insert** tab, and choose the **PivotChart** option.
* Select the type of chart that best suits your analysis (e.g., Bar Chart, Column Chart, Pie Chart for percentage, etc.).
* **Customize** the chart as needed to improve readability.

**Result:**

Your Pivot Chart will display the data visually, making it easier to compare the number of cars, profit margins, average cost, and car color percentages.

**Aim 13.**

****

# Import necessary libraries

import pandas as pd

from sklearn.preprocessing import StandardScaler, MinMaxScaler

from sklearn.impute import SimpleImputer

# Create the dataset

data = {

'Country': ['France', 'Spain', 'Germany', 'Spain', 'Germany', 'France', 'Spain', 'France', 'Germany', 'France'],

'Age': [44.0, 27.0, 30.0, 38.0, 40.0, 35.0, None, 48.0, 50.0, 37.0],

'Salary': [72000.0, 48000.0, 54000.0, 61000.0, 85000.0, 58000.0, 52000.0, 79000.0, 83000.0, 67000.0],

'Purchased': ['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes']

}

# Convert the data into a pandas DataFrame

df = pd.DataFrame(data)

# Display the original dataset

print("Original Dataset:")

print(df)

# Handle missing values (impute missing values with mean)

imputer = SimpleImputer(strategy='mean')

df['Age'] = imputer.fit\_transform(df[['Age']])

# Separate the numerical features (Age, Salary) for scaling

numerical\_features = ['Age', 'Salary']

# Standardization (Z-score normalization)

scaler\_standard = StandardScaler()

df\_standardized = df.copy()

df\_standardized[numerical\_features] = scaler\_standard.fit\_transform(df[numerical\_features])

# Normalization (Min-Max scaling)

scaler\_minmax = MinMaxScaler()

df\_normalized = df.copy()

df\_normalized[numerical\_features] = scaler\_minmax.fit\_transform(df[numerical\_features])

# Display the results

print("\nStandardized Dataset (Z-score normalization):")

print(df\_standardized)

print("\nNormalized Dataset (Min-Max scaling):")

print(df\_normalized)

**Aim 14. Load the Iris dataset. Perform Principal component Analysis (PCA) using Python/R on a dataset to reduce dimensionality. Select appropriate number of principle components. Visualize the data in the reduced dimensional space.**

# Import necessary libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from sklearn.datasets import load\_iris

from sklearn.preprocessing import StandardScaler

# Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

# Convert to DataFrame for better visualization

df = pd.DataFrame(X, columns=iris.feature\_names)

df['species'] = iris.target\_names[y]

# Standardize the data before applying PCA

scaler = StandardScaler()

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

# Apply PCA to reduce dimensionality (Choose 2 principal components)

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X\_scaled)

# Create a DataFrame to hold the PCA components and the target variable

pca\_df = pd.DataFrame(data=X\_pca, columns=['Principal Component 1', 'Principal Component 2'])

pca\_df['species'] = iris.target\_names[y]

# Visualize the data in the reduced 2D space

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

for species in iris.target\_names:

species\_data = pca\_df[pca\_df['species'] == species]

plt.scatter(species\_data['Principal Component 1'], species\_data['Principal Component 2'], label=species)

plt.title('PCA of Iris Dataset (2D)')

plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.legend()

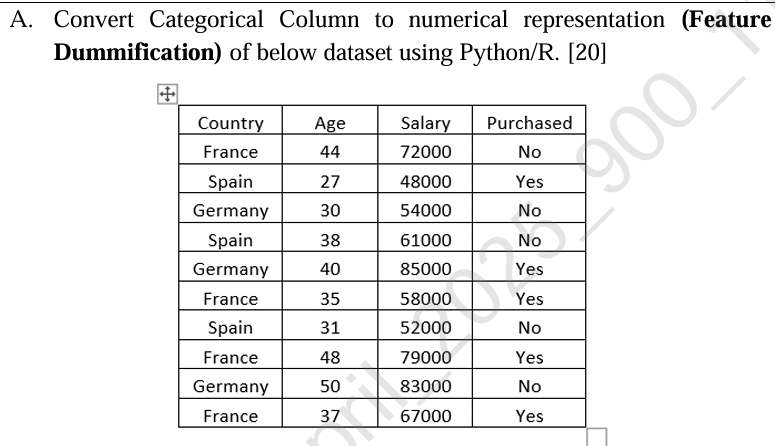
plt.grid(True)

plt.show()

# Print explained variance to determine how much variance is explained by the selected components

print(f"Explained variance ratio of the selected components: {pca.explained\_variance\_ratio\_}")

**Aim 15.**

****

# Import necessary libraries

import pandas as pd

# Define the dataset

data = {

'Country': ['France', 'Spain', 'Germany', 'Spain', 'Germany', 'France', 'Spain', 'France', 'Germany', 'France'],

'Age': [44.0, 27.0, 30.0, 38.0, 40.0, 35.0, None, 48.0, 50.0, 37.0],

'Salary': [72000.0, 48000.0, 54000.0, 61000.0, 85000.0, 58000.0, 52000.0, 79000.0, 83000.0, 67000.0],

'Purchased': ['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes']

}

# Convert to DataFrame

df = pd.DataFrame(data)

# Display the original dataset

print("Original Dataset:")

print(df)

# Apply Feature Dummification (One-Hot Encoding) to the 'Country' and 'Purchased' columns

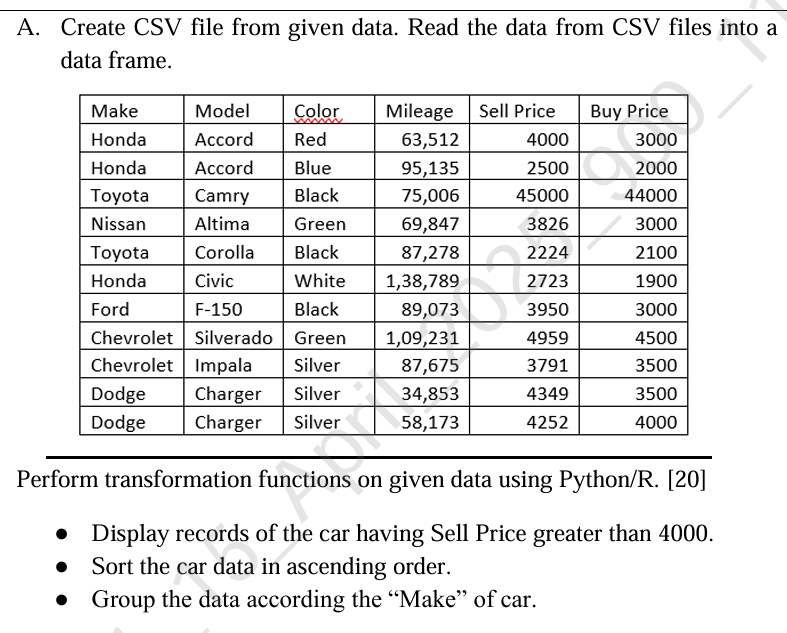
df\_dummified = pd.get\_dummies(df, columns=['Country', 'Purchased'], drop\_first=True)

# Display the modified dataset

print("\nDataset After Feature Dummification (One-Hot Encoding):")

print(df\_dummified)

**Aim 16.**

****

import pandas as pd

# Define the dataset

data = {

'Make': ['Honda', 'Honda', 'Toyota', 'Nissan', 'Toyota', 'Honda', 'Ford', 'Chevrolet', 'Chevrolet', 'Dodge', 'Dodge'],

'Model': ['Accord', 'Accord', 'Camry', 'Altima', 'Corolla', 'Civic', 'F-150', 'Silverado', 'Impala', 'Charger', 'Charger'],

'Color': ['Red', 'Blue', 'Black', 'Green', 'Black', 'White', 'Black', 'Green', 'Silver', 'Silver', 'Silver'],

'Mileage': ['63,512', '95,135', '75,006', '69,847', '87,278', '1,38,789', '89,073', '1,09,231', '87,675', '34,853', '58,173'],

'Sell Price': [4000, 2500, 45000, 3826, 2224, 2723, 3950, 4959, 3791, 4349, 4252],

'Buy Price': [3000, 2000, 44000, 3000, 2100, 1900, 3000, 4500, 3500, 3500, 4000]

}

# Create DataFrame from the data

df = pd.DataFrame(data)

# Convert 'Mileage' column to numeric by removing commas

df['Mileage'] = df['Mileage'].replace({',': ''}, regex=True).astype(int)

# Save the DataFrame to a CSV file

df.to\_csv('car\_data.csv', index=False)

# Read the data back from the CSV file into a DataFrame

df\_from\_csv = pd.read\_csv('car\_data.csv')

# Display the DataFrame

print("Data loaded from CSV:")

print(df\_from\_csv)

# 1. Display records of the car having Sell Price greater than 4000

print("\nCars with Sell Price greater than 4000:")

cars\_above\_4000 = df\_from\_csv[df\_from\_csv['Sell Price'] > 4000]

print(cars\_above\_4000)

# 2. Sort the cars in ascending order by 'Sell Price'

print("\nCars sorted by Sell Price (ascending):")

sorted\_cars = df\_from\_csv.sort\_values(by='Sell Price', ascending=True)

print(sorted\_cars)

# 3. Group the data by 'Make' of the car

print("\nGrouped data by Make:")

grouped\_by\_make = df\_from\_csv.groupby('Make').agg({

'Sell Price': 'mean',

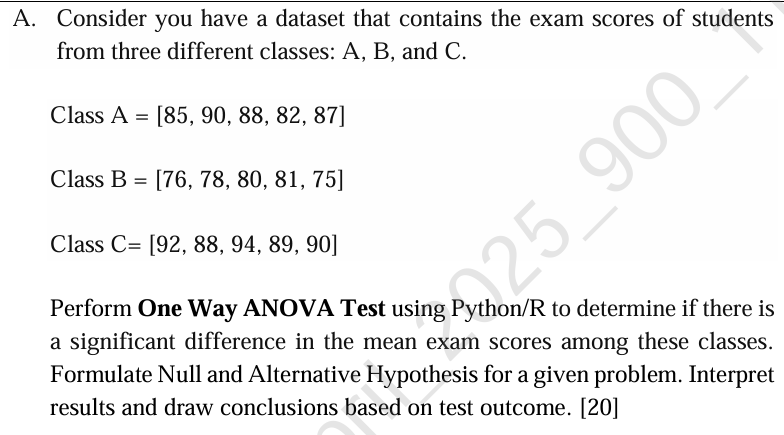
'Buy Price': 'mean',

'Mileage': 'mean'

})

print(grouped\_by\_make)

**Aim 17.**



import numpy as np

from scipy import stats

# Define the exam scores for the three classes

class\_A = [85, 90, 88, 82, 87]

class\_B = [76, 78, 80, 81, 75]

class\_C = [92, 88, 94, 89, 90]

# Perform One-Way ANOVA test

f\_statistic, p\_value = stats.f\_oneway(class\_A, class\_B, class\_C)

# Display the results

print("F-Statistic:", f\_statistic)

print("P-Value:", p\_value)

# Interpretation of results

alpha = 0.05 # Significance level

if p\_value < alpha:

print("\nReject the null hypothesis: There is a significant difference in the mean exam scores among the classes.")

else:

print("\nFail to reject the null hypothesis: There is no significant difference in the mean exam scores among the classes.")

**Aim 18. Load the Wine Quality dataset. Perform Principal component Analysis (PCA) using Python/R on a dataset to reduce dimensionality. Select appropriate number of principle components. Visualize the data in the reduced-dimensional space.**

import pandas as pd

import numpy as np

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt

# Load the Wine Quality dataset (assuming it's stored in 'winequality.csv')

# You can replace the path with your local file path if needed

df = pd.read\_csv('winequality-red.csv')

# Display the first few rows to understand the data

print(df.head())

# Separate the features and the target

X = df.drop('quality', axis=1) # Features (drop the target column)

y = df['quality'] # Target (Wine Quality)

# Standardize the data (important for PCA)

scaler = StandardScaler()

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

# Perform PCA

pca = PCA()

X\_pca = pca.fit\_transform(X\_scaled)

# Plot the explained variance ratio for each principal component

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

plt.plot(np.cumsum(pca.explained\_variance\_ratio\_))

plt.title('Explained Variance by Principal Components')

plt.xlabel('Number of Principal Components')

plt.ylabel('Cumulative Explained Variance')

plt.grid(True)

plt.show()

# Select the number of components that explain at least 95% variance

n\_components = np.argmax(np.cumsum(pca.explained\_variance\_ratio\_) >= 0.95) + 1

print(f"Number of principal components selected: {n\_components}")

# Perform PCA with the selected number of components

pca = PCA(n\_components=n\_components)

X\_pca\_reduced = pca.fit\_transform(X\_scaled)

# Visualize the data in the reduced dimensionality space (2D for simplicity)

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

plt.scatter(X\_pca\_reduced[:, 0], X\_pca\_reduced[:, 1], c=y, cmap='viridis', edgecolor='k', alpha=0.7)

plt.title(f'PCA of Wine Quality Dataset (First {n\_components} components)')

plt.xlabel('Principal Component 1')

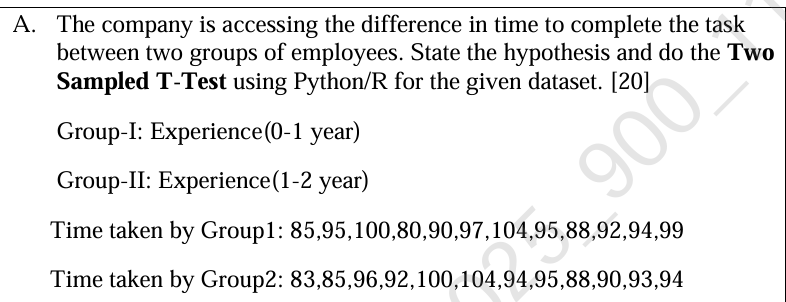
plt.ylabel('Principal Component 2')

plt.colorbar(label='Wine Quality')

plt.grid(True)

plt.show()

**Aim 19.**

****

import scipy.stats as stats

# Time taken by Group-I (Experience 0-1 year)

group1 = [85, 95, 100, 80, 90, 97, 104, 95, 88, 92, 94, 99]

# Time taken by Group-II (Experience 1-2 years)

group2 = [83, 85, 96, 92, 100, 104, 94, 95, 88, 90, 93, 94]

# Perform Two-Sample (Independent) T-Test

t\_stat, p\_value = stats.ttest\_ind(group1, group2)

# Print results

print(f"T-statistic: {t\_stat:.4f}")

print(f"P-value: {p\_value:.4f}")

# Interpret the result

alpha = 0.05

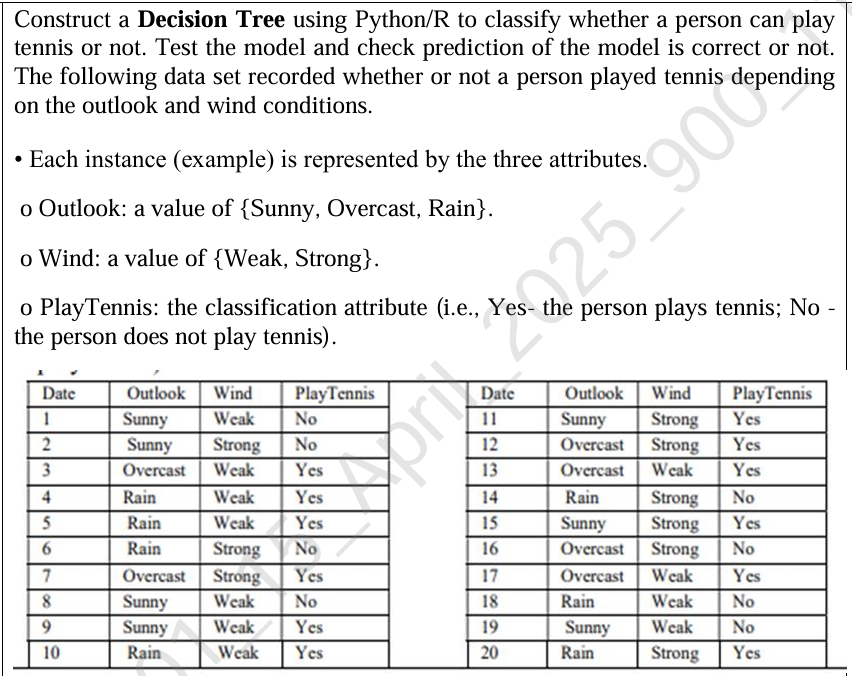
if p\_value < alpha:

print("Conclusion: Reject the null hypothesis. There is a significant difference in task completion time between the two groups.")

else:

print("Conclusion: Fail to reject the null hypothesis. There is no significant difference in task completion time between the two groups.")

**Aim 20.**

****

import pandas as pd

from sklearn.preprocessing import LabelEncoder

from sklearn.tree import DecisionTreeClassifier, plot\_tree

import matplotlib.pyplot as plt

# Create the dataset

data = {

'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Rain', 'Overcast', 'Sunny', 'Sunny', 'Rain',

'Sunny', 'Overcast', 'Overcast', 'Rain', 'Sunny', 'Overcast', 'Overcast', 'Rain', 'Sunny', 'Rain'],

'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Weak', 'Weak',

'Strong', 'Strong', 'Weak', 'Strong', 'Strong', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong'],

'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes',

'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'No', 'Yes']

}

# Convert to DataFrame

df = pd.DataFrame(data)

# Encode categorical features

le\_outlook = LabelEncoder()

le\_wind = LabelEncoder()

le\_play = LabelEncoder()

df['Outlook\_encoded'] = le\_outlook.fit\_transform(df['Outlook'])

df['Wind\_encoded'] = le\_wind.fit\_transform(df['Wind'])

df['PlayTennis\_encoded'] = le\_play.fit\_transform(df['PlayTennis'])

# Define features and target

X = df[['Outlook\_encoded', 'Wind\_encoded']]

y = df['PlayTennis\_encoded']

# Train Decision Tree Classifier

clf = DecisionTreeClassifier(criterion='entropy', random\_state=0)

clf.fit(X, y)

# Predict on the training data itself (for testing correctness)

y\_pred = clf.predict(X)

# Show actual vs predicted

df['Predicted'] = le\_play.inverse\_transform(y\_pred)

print("Actual vs Predicted:")

print(df[['Outlook', 'Wind', 'PlayTennis', 'Predicted']])

# Visualize the decision tree

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

plot\_tree(clf, feature\_names=['Outlook', 'Wind'],

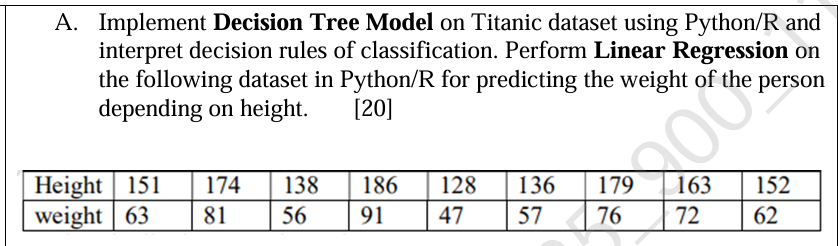
class\_names=le\_play.classes\_,

filled=True, rounded=True)

plt.title("Decision Tree for PlayTennis Classification")

plt.show()

**Aim 21.**

****

# ------------------ Part A: Decision Tree on Titanic Dataset ------------------

import pandas as pd

import seaborn as sns

from sklearn.tree import DecisionTreeClassifier, plot\_tree

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

from sklearn.preprocessing import LabelEncoder

import matplotlib.pyplot as plt

# Load Titanic dataset

titanic = sns.load\_dataset('titanic')

# Select features and drop missing values

df = titanic[['survived', 'pclass', 'sex', 'age', 'embarked']].dropna()

# Encode categorical data

le\_sex = LabelEncoder()

le\_embarked = LabelEncoder()

df['sex'] = le\_sex.fit\_transform(df['sex'])

df['embarked'] = le\_embarked.fit\_transform(df['embarked'])

# Define features and target

X = df[['pclass', 'sex', 'age', 'embarked']]

y = df['survived']

# Train-test split

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

# Train Decision Tree

clf = DecisionTreeClassifier(criterion='entropy', random\_state=42)

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

# Print accuracy

print(f"Decision Tree Model Accuracy on Titanic Test Set: {clf.score(X\_test, y\_test):.2f}")

# Plot decision tree

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

plot\_tree(clf, feature\_names=X.columns, class\_names=['Died', 'Survived'], filled=True, rounded=True)

plt.title("Decision Tree for Titanic Survival Prediction")

plt.show()

# ------------------ Part B: Linear Regression on Height vs Weight ------------------

import numpy as np

from sklearn.linear\_model import LinearRegression

# Dataset

height = np.array([151, 174, 138, 186, 128, 136, 179, 163, 152]).reshape(-1, 1)

weight = np.array([63, 81, 56, 91, 47, 57, 76, 72, 62])

# Train Linear Regression model

model = LinearRegression()

model.fit(height, weight)

# Get regression equation

slope = model.coef\_[0]

intercept = model.intercept\_

print(f"Linear Regression Equation: weight = {slope:.2f} \* height + {intercept:.2f}")

# Predict weights

predicted\_weight = model.predict(height)

# Plot regression line

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

plt.scatter(height, weight, color='blue', label='Actual Data')

plt.plot(height, predicted\_weight, color='red', linewidth=2, label='Regression Line')

plt.xlabel('Height (cm)')

plt.ylabel('Weight (kg)')

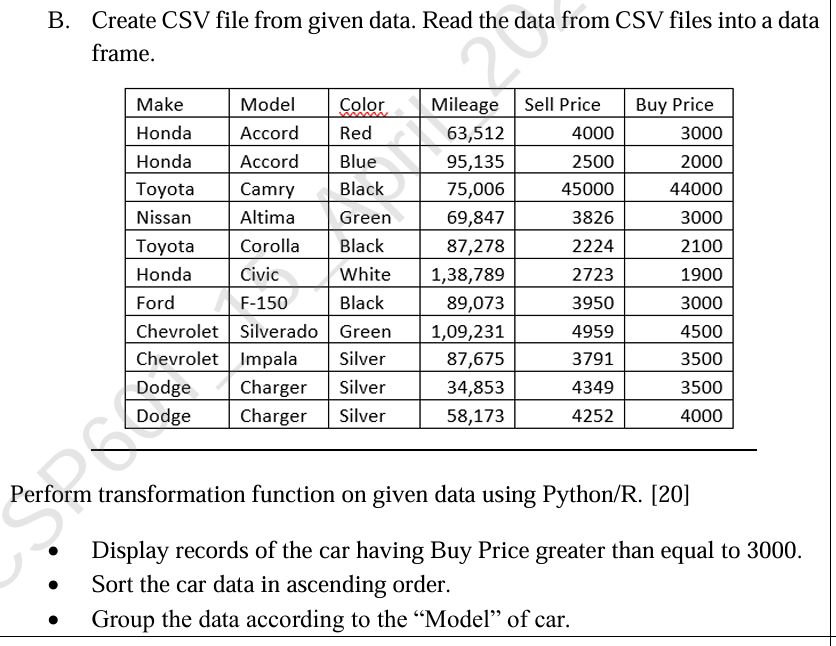
plt.title('Linear Regression: Height vs Weight')

plt.legend()

plt.grid(True)

plt.show()

**Aim 22.**

****

import pandas as pd

# Step 1: Create data

data = {

'Make': ['Honda', 'Honda', 'Toyota', 'Nissan', 'Toyota', 'Honda', 'Ford', 'Chevrolet', 'Chevrolet', 'Dodge', 'Dodge'],

'Model': ['Accord', 'Accord', 'Camry', 'Altima', 'Corolla', 'Civic', 'F-150', 'Silverado', 'Impala', 'Charger', 'Charger'],

'Color': ['Red', 'Blue', 'Black', 'Green', 'Black', 'White', 'Black', 'Green', 'Silver', 'Silver', 'Silver'],

'Mileage': ['63,512', '95,135', '75,006', '69,847', '87,278', '1,38,789', '89,073', '1,09,231', '87,675', '34,853', '58,173'],

'Sell Price': [4000, 2500, 45000, 3826, 2224, 2723, 3950, 4959, 3791, 4349, 4252],

'Buy Price': [3000, 2000, 44000, 3000, 2100, 1900, 3000, 4500, 3500, 3500, 4000]

}

# Step 2: Convert to DataFrame

df = pd.DataFrame(data)

# Step 3: Save to CSV

df.to\_csv('car\_data.csv', index=False)

# Step 4: Read from CSV

df = pd.read\_csv('car\_data.csv')

# Optional: Clean up the 'Mileage' column by removing commas and converting to int

df['Mileage'] = df['Mileage'].str.replace(',', '').astype(int)

# Step 5: Transformation operations

# 1. Display records where Buy Price >= 3000

print("\nCars with Buy Price >= 3000:")

print(df[df['Buy Price'] >= 3000])

# 2. Sort the car data in ascending order (by 'Make' then 'Model')

sorted\_df = df.sort\_values(by=['Make', 'Model'])

print("\nSorted Car Data (by Make, then Model):")

print(sorted\_df)

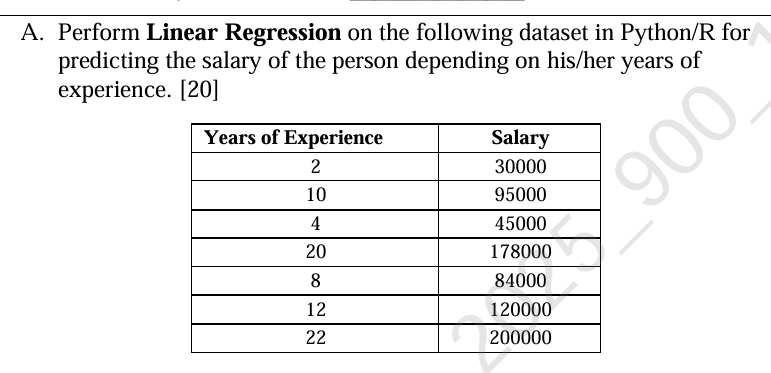
# 3. Group data by 'Model' and display grouped statistics

grouped\_df = df.groupby('Model').mean(numeric\_only=True)

print("\nGrouped Data (Average values by Model):")

print(grouped\_df)

**Aim 23.**



import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

# Step 1: Create the dataset

data = {

'YearsExperience': [2, 10, 4, 20, 8, 12, 22],

'Salary': [30000, 95000, 45000, 178000, 84000, 120000, 200000]

}

# Step 2: Convert to DataFrame

df = pd.DataFrame(data)

# Step 3: Split features and target

X = df[['YearsExperience']] # Features must be 2D

y = df['Salary']

# Step 4: Create and train the model

model = LinearRegression()

model.fit(X, y)

# Step 5: Predict salaries

y\_pred = model.predict(X)

# Step 6: Print regression coefficients

print(f"Intercept: {model.intercept\_:.2f}")

print(f"Coefficient: {model.coef\_[0]:.2f}")

# Step 7: Compare actual vs predicted

df['PredictedSalary'] = y\_pred

print("\nActual vs Predicted Salaries:")

print(df)

# Step 8: Plotting

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

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

plt.plot(X, y\_pred, color='red', linewidth=2, label='Regression Line')

plt.title('Salary vs Years of Experience')

plt.xlabel('Years of Experience')

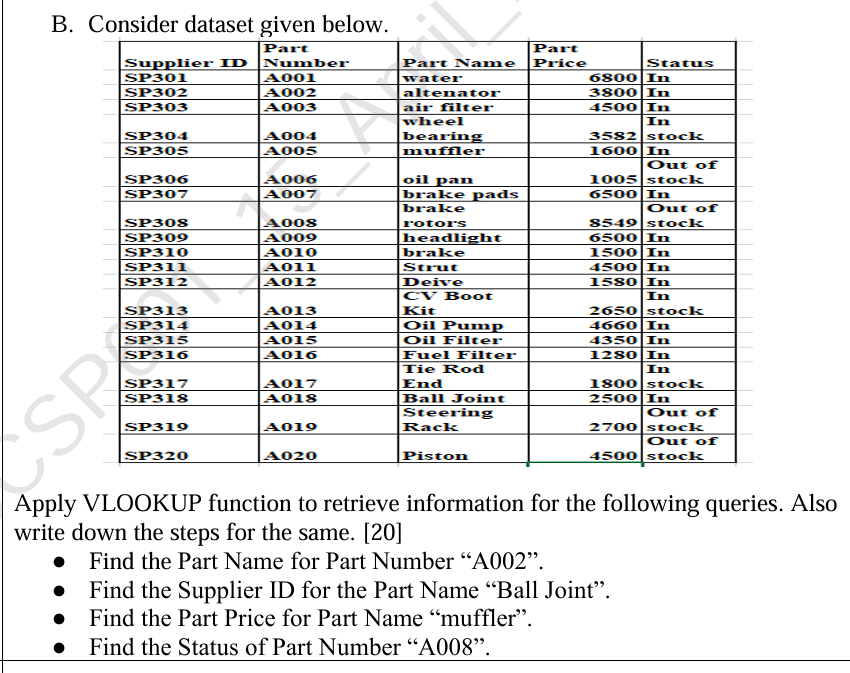
plt.ylabel('Salary')

plt.legend()

plt.grid(True)

plt.show()

**Aim 24.**

****

**Using Code:**

import pandas as pd

# Create the dataset as a DataFrame

data = {

'Supplier ID': ['SP301', 'SP301', 'SP302', 'SP304', 'SP305', 'SP306', 'SP307', 'SP308', 'SP309', 'SP310',

'SP311', 'SP312', 'SP313', 'SP314', 'SP315', 'SP316', 'SP317', 'SP318', 'SP319', 'SP320'],

'Part Number': ['A001', 'A002', 'A003', 'A004', 'A005', 'A006', 'A007', 'A008', 'A009', 'A010',

'A011', 'A012', 'A013', 'A014', 'A015', 'A016', 'A017', 'A018', 'A019', 'A020'],

'Part Name': ['water', 'alternator', 'air filter', 'wheel bearing', 'muffler', 'oil pan', 'brake pads', 'brake rotors',

'headlight', 'brake', 'Strut', 'Drive', 'CV Boot Kit', 'Oil Pump', 'Oil Filter', 'Fuel Filter', 'End',

'Ball Joint', 'Steering Rack', 'Piston'],

'Part Price': [6800, 3800, 4500, 3582, 1600, 1005, 6500, 8549, 6500, 1500, 4500, 1580, 2650, 4660, 4350, 1280, 1800,

2500, 2700, 4500],

'Status': ['In', 'In', 'In', 'Stock', 'In', 'Stock', 'In', 'Stock', 'In', 'In', 'In', 'In', 'Stock', 'In', 'In',

'In', 'Stock', 'In', 'Stock', 'Stock']

}

# Convert data into a DataFrame

df = pd.DataFrame(data)

# Find the Part Name for Part Number "A002"

part\_name\_a002 = df.loc[df['Part Number'] == 'A002', 'Part Name'].values[0]

print(f'Part Name for Part Number A002: {part\_name\_a002}')

# Find the Supplier ID for Part Name "Ball Joint"

supplier\_id\_ball\_joint = df.loc[df['Part Name'] == 'Ball Joint', 'Supplier ID'].values[0]

print(f'Supplier ID for Part Name "Ball Joint": {supplier\_id\_ball\_joint}')

# Find the Part Price for Part Name "muffler"

part\_price\_muffler = df.loc[df['Part Name'] == 'muffler', 'Part Price'].values[0]

print(f'Part Price for Part Name "muffler": {part\_price\_muffler}')

# Find the Status of Part Number "A008"

status\_a008 = df.loc[df['Part Number'] == 'A008', 'Status'].values[0]

print(f'Status of Part Number A008: {status\_a008}')

**Using Excel VLOOKUP Instructions:**

**1. Prepare Your Data**

Make sure your dataset is in a tabular format like this in Excel:

| **Supplier ID** | **Part Number** | **Part Name** | **Part Price** | **Status** |
| --- | --- | --- | --- | --- |
| SP301 | A001 | water | 6800 | In |
| SP301 | A002 | alternator | 3800 | In |
| SP302 | A003 | air filter | 4500 | In |
| SP304 | A004 | wheel bearing | 3582 | Stock |
| SP305 | A005 | muffler | 1600 | In |
| SP306 | A006 | oil pan | 1005 | Stock |
| SP307 | A007 | brake pads | 6500 | In |
| SP308 | A008 | brake rotors | 8549 | Stock |
| SP309 | A009 | headlight | 6500 | In |
| SP310 | A010 | brake | 1500 | In |
| SP311 | A011 | Strut | 4500 | In |
| SP312 | A012 | Drive | 1580 | In |
| SP313 | A013 | CV Boot Kit | 2650 | Stock |
| SP314 | A014 | Oil Pump | 4660 | In |
| SP315 | A015 | Oil Filter | 4350 | In |
| SP316 | A016 | Fuel Filter | 1280 | In |
| SP317 | A017 | End | 1800 | Stock |
| SP318 | A018 | Ball Joint | 2500 | In |
| SP319 | A019 | Steering Rack | 2700 | Stock |
| SP320 | A020 | Piston | 4500 | Stock |

**2. Use VLOOKUP to Find the Information**

**Query 1: Find the Part Name for Part Number "A002"**

1. **Step 1**: Click on the cell where you want the result to be displayed (e.g., G1).
2. **Step 2**: Type the following VLOOKUP formula:
3. =VLOOKUP("A002", B2:E21, 2, FALSE)
   * "A002" is the value you are searching for.
   * B2:E21 is the range where the data is located (make sure the Part Number column is in the first column of the range).
   * 2 is the column index number from which to retrieve the result (Part Name is in the 2nd column).
   * FALSE ensures that the search looks for an exact match.
4. **Step 3**: Press Enter, and Excel will return the value "alternator" (the Part Name for Part Number A002).

**Query 2: Find the Supplier ID for Part Name "Ball Joint"**

1. **Step 1**: Click on the cell where you want the result to be displayed (e.g., G2).
2. **Step 2**: Type the following VLOOKUP formula:
3. =VLOOKUP("Ball Joint", C2:E21, 1, FALSE)
   * "Ball Joint" is the value you are searching for.
   * C2:E21 is the range of data to search in (Part Name is the 1st column in the selected range).
   * 1 is the column index number from which to retrieve the result (Supplier ID is in the 1st column).
   * FALSE ensures that the search looks for an exact match.
4. **Step 3**: Press Enter, and Excel will return the value "SP318" (the Supplier ID for Part Name Ball Joint).

**Query 3: Find the Part Price for Part Name "muffler"**

1. **Step 1**: Click on the cell where you want the result to be displayed (e.g., G3).
2. **Step 2**: Type the following VLOOKUP formula:
3. =VLOOKUP("muffler", C2:E21, 3, FALSE)
   * "muffler" is the value you are searching for.
   * C2:E21 is the range of data to search in (Part Name is the 1st column in the selected range).
   * 3 is the column index number from which to retrieve the result (Part Price is in the 3rd column).
   * FALSE ensures that the search looks for an exact match.
4. **Step 3**: Press Enter, and Excel will return the value "1600" (the Part Price for Part Name muffler).

**Query 4: Find the Status of Part Number "A008"**

1. **Step 1**: Click on the cell where you want the result to be displayed (e.g., G4).
2. **Step 2**: Type the following VLOOKUP formula:
3. =VLOOKUP("A008", B2:E21, 4, FALSE)
   * "A008" is the value you are searching for.
   * B2:E21 is the range of data to search in (Part Number is the 1st column in the selected range).
   * 4 is the column index number from which to retrieve the result (Status is in the 4th column).
   * FALSE ensures that the search looks for an exact match.
4. **Step 3**: Press Enter, and Excel will return the value "Stock" (the Status of Part Number A008).

**Aim 25. Perform Linear Regression on the Iris dataset of R/Python for predicting petal.width on petal.length.**

# Import necessary libraries

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.metrics import mean\_squared\_error, r2\_score

# Load the Iris dataset (replace with the correct path to your Iris.csv)

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

# Check the first few rows to understand the dataset

print(df.head())

# Select the features for Linear Regression

X = df[['PetalLengthCm']] # Features (independent variable)

y = df['PetalWidthCm'] # Target (dependent variable)

# Split the data into training and testing sets (80% training, 20% testing)

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

# Initialize the Linear Regression model

model = LinearRegression()

# Train the model on the training data

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

# Make predictions on the test data

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

# Evaluate the model

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

# Print the results

print(f"Mean Squared Error: {mse}")

print(f"R-squared: {r2}")

# Visualize the regression line

plt.scatter(X\_test, y\_test, color='blue', label='Actual Data')

plt.plot(X\_test, y\_pred, color='red', label='Fitted Line')

plt.title('Linear Regression: Petal Width vs Petal Length')

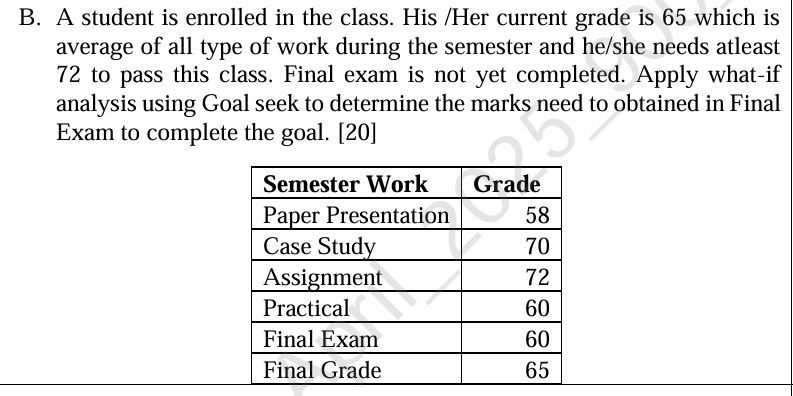
plt.xlabel('Petal Length')

plt.ylabel('Petal Width')

plt.legend()

plt.show()

**Aim 26.**

****

**Using code:**

# Given data for the student's semester work and grades

semester\_work = {

'Paper Presentation': 58,

'Case Study': 70,

'Assignment': 72,

'Practical': 60,

'Final Exam': 60

}

# The current final grade (average of all semester work excluding the final exam)

current\_grade = 65

# The target grade the student wants to achieve

target\_grade = 72

# Total work components (including the final exam)

total\_components = len(semester\_work)

# Calculate the current total without final exam

current\_total\_without\_final\_exam = sum(value for key, value in semester\_work.items() if key != 'Final Exam')

# Function to calculate the required final exam grade

def calculate\_required\_final\_exam(current\_grade, target\_grade, current\_total\_without\_final\_exam, total\_components):

# Calculate the total grade required (final grade \* total number of components)

total\_grade\_required = target\_grade \* total\_components

# The required final exam grade is what is left to achieve the target grade

required\_final\_exam\_grade = total\_grade\_required - current\_total\_without\_final\_exam

return required\_final\_exam\_grade

# Calculate the required final exam marks to achieve the target grade

required\_final\_exam = calculate\_required\_final\_exam(current\_grade, target\_grade, current\_total\_without\_final\_exam, total\_components)

# Display the result

print(f"To achieve the target grade of {target\_grade}, the student needs to score at least {required\_final\_exam:.2f} in the Final Exam.")

**Using Excel:**

To solve this problem in Excel using **Goal Seek** for What-If analysis, follow these steps:

**Step-by-Step Guide:**

1. **Open Excel** and enter the data in a similar structure as shown below:

| **Semester Work** | **Grade** |
| --- | --- |
| Paper Presentation | 58 |
| Case Study | 70 |
| Assignment | 72 |
| Practical | 60 |
| Final Exam | 60 |
| **Final Grade** | 65 |

* **Column A**: Semester Work (e.g., Paper Presentation, Case Study, etc.)
* **Column B**: Grades corresponding to each type of work.

1. **Calculate the Final Grade (Current Average)**:
   * In cell **B6**, you need to calculate the current average grade.
   * The **Final Grade** (cell B6) is the average of all the grades except the Final Exam grade (since that is yet to be determined).

Enter the following formula in **B6**:

=AVERAGE(B2:B5)

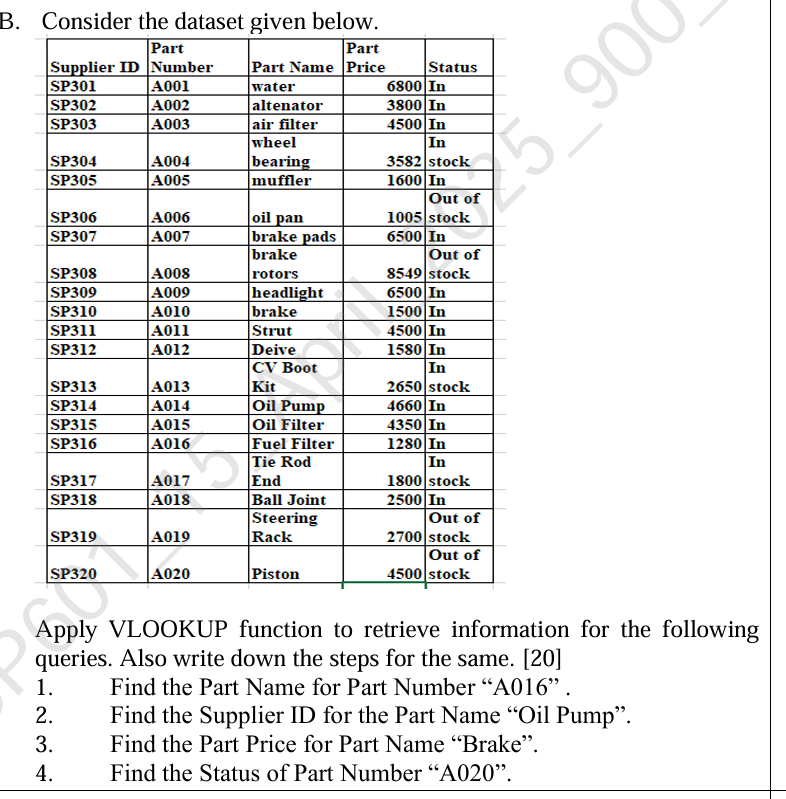
* + This formula calculates the average of the grades for Paper Presentation, Case Study, Assignment, and Practical.

1. **Set up Goal Seek to Find Required Final Exam Marks**:
   * Now, the objective is to find the marks required in the **Final Exam** (cell B5) to ensure the **Final Grade** (cell B6) reaches 72.
   * Select **Data** from the menu bar at the top.
   * Click on **What-If Analysis** in the Data tab.
   * From the dropdown, select **Goal Seek**.
2. **Configure Goal Seek**:
   * In the **Goal Seek** dialog box:
     + **Set cell**: B6 (because this cell represents the current final grade, which you want to change).
     + **To value**: 72 (because this is the target final grade you want to achieve).
     + **By changing cell**: B5 (because you want to change the Final Exam grade to reach the target final grade).

Click **OK** to run Goal Seek.

1. **Review the Result**:
   * After running Goal Seek, Excel will calculate and update the **Final Exam** grade (cell B5).
   * The required grade for the **Final Exam** will be displayed in **cell B5**.

**Aim 27.**

****

**Using code:**

import pandas as pd

# Create the dataset as a DataFrame

data = {

'Supplier ID': ['SP301', 'SP301', 'SP302', 'SP304', 'SP305', 'SP306', 'SP307', 'SP308', 'SP309', 'SP310',

'SP311', 'SP312', 'SP313', 'SP314', 'SP315', 'SP316', 'SP317', 'SP318', 'SP319', 'SP320'],

'Part Number': ['A001', 'A002', 'A003', 'A004', 'A005', 'A006', 'A007', 'A008', 'A009', 'A010',

'A011', 'A012', 'A013', 'A014', 'A015', 'A016', 'A017', 'A018', 'A019', 'A020'],

'Part Name': ['water', 'alternator', 'air filter', 'wheel bearing', 'muffler', 'oil pan', 'brake pads', 'brake rotors',

'headlight', 'brake', 'Strut', 'Drive', 'CV Boot Kit', 'Oil Pump', 'Oil Filter', 'Fuel Filter', 'Tie Rod',

'Ball Joint', 'Steering Rack', 'Piston'],

'Part Price': [6800, 3800, 4500, 3582, 1600, 1005, 6500, 8549, 6500, 1500, 4500, 1580, 2650, 4660, 4350, 1280, 1800,

2500, 2700, 4500],

'Status': ['In', 'In', 'In', 'Stock', 'Out of Stock', 'Stock', 'Out of Stock', 'Stock', 'In', 'In', 'In', 'In', 'Stock', 'In', 'In',

'In', 'Stock', 'In', 'Out of Stock', 'Stock']

}

# Convert data into a DataFrame

df = pd.DataFrame(data)

# 1. Find the Part Name for Part Number "A016"

part\_name\_a016 = df.loc[df['Part Number'] == 'A016', 'Part Name'].values[0]

print(f'Part Name for Part Number A016: {part\_name\_a016}')

# 2. Find the Supplier ID for the Part Name "Oil Pump"

supplier\_id\_oil\_pump = df.loc[df['Part Name'] == 'Oil Pump', 'Supplier ID'].values[0]

print(f'Supplier ID for Part Name "Oil Pump": {supplier\_id\_oil\_pump}')

# 3. Find the Part Price for Part Name "Brake"

part\_price\_brake = df.loc[df['Part Name'] == 'brake', 'Part Price'].values[0]

print(f'Part Price for Part Name "brake": {part\_price\_brake}')

# 4. Find the Status of Part Number "A020"

status\_a020 = df.loc[df['Part Number'] == 'A020', 'Status'].values[0]

print(f'Status of Part Number A020: {status\_a020}')

**Using Excel:**

Here is a step-by-step guide to performing the requested queries using the **VLOOKUP** function in Excel:

**Steps for Excel:**

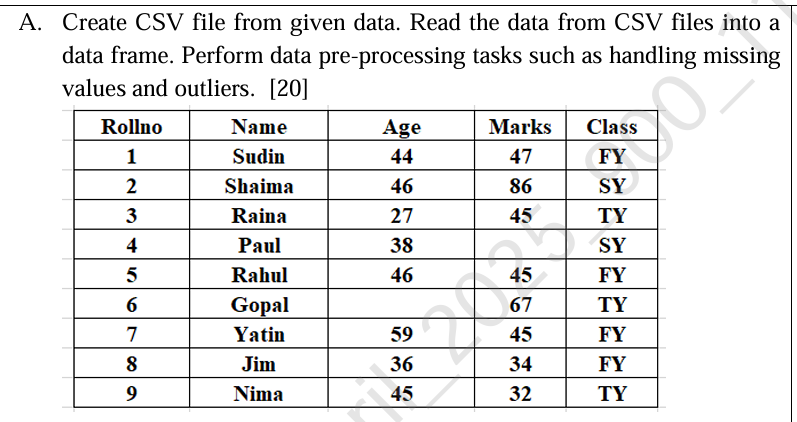
1. **Organize Your Data:**
   * First, enter the dataset in Excel. Assume that your data is placed in cells **A1:E21** with the following column headers:
     + **Column A**: Supplier ID
     + **Column B**: Part Number
     + **Column C**: Part Name
     + **Column D**: Part Price
     + **Column E**: Status

The data should look like this:

| **Supplier ID** | **Part Number** | **Part Name** | **Part Price** | **Status** |
| --- | --- | --- | --- | --- |
| SP301 | A001 | water | 6800 | In |
| SP301 | A002 | alternator | 3800 | In |
| ... | ... | ... | ... | ... |
| SP320 | A020 | Piston | 4500 | Stock |

1. **Performing VLOOKUP for Query 1: "Find the Part Name for Part Number 'A016'"**
   * Go to an empty cell where you want to display the result, for example, **G2**.
   * Use the following **VLOOKUP** formula:
   * =VLOOKUP("A016", B2:E21, 3, FALSE)
     + **Explanation**:
       - "A016": The lookup value (the Part Number you're searching for).
       - B2:E21: The range where you want to search (includes the **Part Number** column as the first column).
       - 3: The column index number from the lookup range where you want the result (3 refers to **Part Name** column).
       - FALSE: Ensures an exact match for the lookup value.
   * After pressing **Enter**, Excel will display the result: **"Fuel Filter"**.
2. **Performing VLOOKUP for Query 2: "Find the Supplier ID for Part Name 'Oil Pump'"**
   * Go to an empty cell where you want to display the result, for example, **G3**.
   * Use the following **VLOOKUP** formula:
   * =VLOOKUP("Oil Pump", C2:E21, 1, FALSE)
     + **Explanation**:
       - "Oil Pump": The lookup value (the Part Name you're searching for).
       - C2:E21: The range where you want to search (includes the **Part Name** column as the first column).
       - 1: The column index number from the lookup range where you want the result (1 refers to **Supplier ID** column).
       - FALSE: Ensures an exact match for the lookup value.
   * After pressing **Enter**, Excel will display the result: **"SP314"**.
3. **Performing VLOOKUP for Query 3: "Find the Part Price for Part Name 'Brake'"**
   * Go to an empty cell where you want to display the result, for example, **G4**.
   * Use the following **VLOOKUP** formula:
   * =VLOOKUP("brake", C2:E21, 2, FALSE)
     + **Explanation**:
       - "brake": The lookup value (the Part Name you're searching for).
       - C2:E21: The range where you want to search (includes the **Part Name** column as the first column).
       - 2: The column index number from the lookup range where you want the result (2 refers to **Part Price** column).
       - FALSE: Ensures an exact match for the lookup value.
   * After pressing **Enter**, Excel will display the result: **"1500"**.
4. **Performing VLOOKUP for Query 4: "Find the Status of Part Number 'A020'"**
   * Go to an empty cell where you want to display the result, for example, **G5**.
   * Use the following **VLOOKUP** formula:
   * =VLOOKUP("A020", B2:E21, 4, FALSE)
     + **Explanation**:
       - "A020": The lookup value (the Part Number you're searching for).
       - B2:E21: The range where you want to search (includes the **Part Number** column as the first column).
       - 4: The column index number from the lookup range where you want the result (4 refers to **Status** column).
       - FALSE: Ensures an exact match for the lookup value.
   * After pressing **Enter**, Excel will display the result: **"Stock"**.

**Aim 28.**

****

import pandas as pd

import numpy as np

# Step 1: Create CSV file from given data

data = {

'Rollno': [1, 2, 3, 4, 5, 6, 7, 8, 9],

'Name': ['Sudin', 'Shaima', 'Raina', 'Paul', 'Rahul', 'Gopal', 'Yatin', 'Jim', 'Nima'],

'Age': [44, 46, 27, 38, 46, np.nan, 59, 36, 45],

'Marks': [47, 86, 45, 22, 45, 67, 45, 34, 32],

'Class': ['FY', 'SY', 'TY', 'SY', 'FY', 'TY', 'FY', 'FY', 'TY']

}

# Creating the DataFrame

df = pd.DataFrame(data)

# Save the DataFrame to CSV

df.to\_csv('student\_data.csv', index=False)

# Step 2: Read the data from CSV file into a DataFrame

df = pd.read\_csv('student\_data.csv')

# Step 3: Handle Missing Values

# Fill missing values in the 'Age' column with the mean of the column

df['Age'] = df['Age'].fillna(df['Age'].mean())

# Step 4: Handle Outliers

# Calculate the IQR for 'Marks' column

Q1 = df['Marks'].quantile(0.25)

Q3 = df['Marks'].quantile(0.75)

IQR = Q3 - Q1

# Identify outliers using IQR

outlier\_lower\_limit = Q1 - 1.5 \* IQR

outlier\_upper\_limit = Q3 + 1.5 \* IQR

# Filter out outliers in 'Marks' column

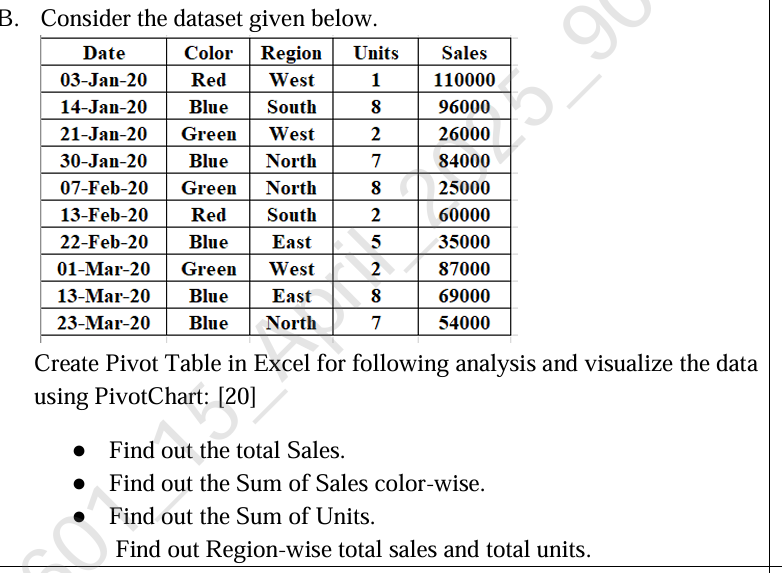
df\_filtered = df[(df['Marks'] >= outlier\_lower\_limit) & (df['Marks'] <= outlier\_upper\_limit)]

# Output the cleaned DataFrame

print("Cleaned Data (Missing values filled and outliers removed):")

print(df\_filtered)

**Aim 29.**

****

Here's a detailed step-by-step guide to creating a Pivot Table and visualizing the data in Excel to answer the queries mentioned:

**Step 1: Prepare Your Data**

1. **Input the Data**: Open Excel and input the dataset you provided. The data should be organized as follows:

| **Date** | **Color** | **Region** | **Units** | **Sales** |
| --- | --- | --- | --- | --- |
| 03-Jan-20 | Red | West | 1 | 110000 |
| 14-Jan-20 | Blue | South | 8 | 96000 |
| 21-Jan-20 | Green | West | 2 | 26000 |
| 30-Jan-20 | Blue | North | 7 | 84000 |
| 07-Feb-20 | Green | North | 8 | 25000 |
| 13-Feb-20 | Red | South | 2 | 60000 |
| 22-Feb-20 | Blue | East | 5 | 35000 |
| 01-Mar-20 | Green | West | 2 | 87000 |
| 13-Mar-20 | Blue | East | 8 | 69000 |
| 23-Mar-20 | Blue | North | 7 | 54000 |

**Step 2: Insert Pivot Table**

1. **Select the Data**: Highlight the entire range of your data, including the headers (A1:E11).
2. **Insert Pivot Table**:
   * Go to the **Insert** tab on the Excel ribbon.
   * Click on **PivotTable**.
   * In the dialog box that appears, Excel will automatically detect the range for your data. Make sure it includes all the rows and columns.
   * Choose to place the Pivot Table in a **New Worksheet** for a clean layout.
   * Click **OK**.

**Step 3: Set Up the Pivot Table Fields**

You should now see a new worksheet with an empty Pivot Table and a field list on the right.

1. **Find Total Sales**:
   * In the Pivot Table field list, drag **Sales** to the **Values** area. By default, Excel will sum the sales, which will give you the total sales.
   * Your Pivot Table will now show the total sales for the entire dataset.
2. **Sum of Sales Color-Wise**:
   * Drag **Color** to the **Rows** area.
   * Drag **Sales** to the **Values** area again (if not already there). It will automatically calculate the sum of Sales for each color.
   * Your Pivot Table will show total sales for each color (Red, Blue, Green).
3. **Sum of Units**:
   * Drag **Units** to the **Values** area.
   * It will automatically calculate the sum of units sold. You will now see the total units in addition to the sales.
4. **Region-Wise Total Sales and Total Units**:
   * Drag **Region** to the **Rows** area.
   * You already have **Sales** and **Units** in the **Values** area. These will show the total sales and total units for each region.
   * Your Pivot Table will now display total sales and units for each region (West, South, North, East).

**Step 4: Customize the Pivot Table (Optional)**

* You can right-click on any of the numbers in the **Values** area and choose **Value Field Settings** to change how the data is summarized (e.g., sum, average, count).
* You can also format the numbers by right-clicking them and choosing **Format Cells**.

**Step 5: Create PivotChart**

1. **Insert PivotChart**:
   * Click anywhere inside the Pivot Table.
   * Go to the **Insert** tab.
   * Click on **PivotChart** in the Charts group.
   * Choose a chart type (e.g., **Column Chart**, **Bar Chart**, **Pie Chart**, etc.) and click **OK**.
   * The chart will appear next to your Pivot Table, displaying a visual representation of the data.
2. **Customize the PivotChart**:
   * You can customize the chart by changing its title, colors, and other design elements using the **Chart Tools** in the ribbon.
   * You can also filter the chart data by selecting different regions or colors.

**Step 6: Final Output**

At this point, you will have:

1. A **Pivot Table** showing the total sales, color-wise sales, unit sums, and region-wise sales and units.
2. A **PivotChart** visualizing the data in your chosen chart format.

**Example Output:**

1. **Total Sales**: The grand total of all sales across the entire dataset.
2. **Sum of Sales (Color-Wise)**:
   * Red: Total Sales for Red.
   * Blue: Total Sales for Blue.
   * Green: Total Sales for Green.
3. **Sum of Units**: The total number of units sold across all records.
4. **Region-Wise Total Sales and Units**:
   * West: Total sales and total units sold in the West region.
   * South: Total sales and total units sold in the South region.
   * North: Total sales and total units sold in the North region.
   * East: Total sales and total units sold in the East region.