

PRACTICAL 1

Practical 1 - Perform the analysis for the following

1A. Import the data warehouse data in Microsoft Excel and create the Pivot table and Pivot Chart.

Step 1: Import Data from a Data Warehouse into Excel

1. Open Excel

- Launch Microsoft Excel.

2. Go to Data Tab

- Click on "Get Data" > "From Other Sources" > "From SQL Server Database" (or any other relevant source).

3. Enter Server Details

- In the "SQL Server database" window:
 - Enter the **Server Name**.
 - Enter the **Database Name** (optional).
 - Click **OK**.

4. Select the Data to Import

- Choose the tables or views that you need from the data warehouse.
- Click **Load** to import the data into Excel.

Step 2: Create a Pivot Table

1. Select the Imported Data

- Click anywhere inside the imported data.

2. Go to Insert Tab

- Click on "**PivotTable**".

3. Choose Pivot Table Options

- In the "**Create PivotTable**" window:
 - Ensure the correct table/range is selected.
 - Choose where to place the PivotTable (New or Existing Worksheet).
 - Click **OK**.

4. Design the Pivot Table

- Drag and drop fields into the:
 - **Rows** area (e.g., Categories, Regions).
 - **Columns** area (e.g., Time Periods).
 - **Values** area (e.g., Sales, Revenue).
 - **Filters** area (optional).

Step 3: Create a Pivot Chart

1. Click on the Pivot Table

- Go to **Insert Tab > PivotChart**.

2. Select Chart Type

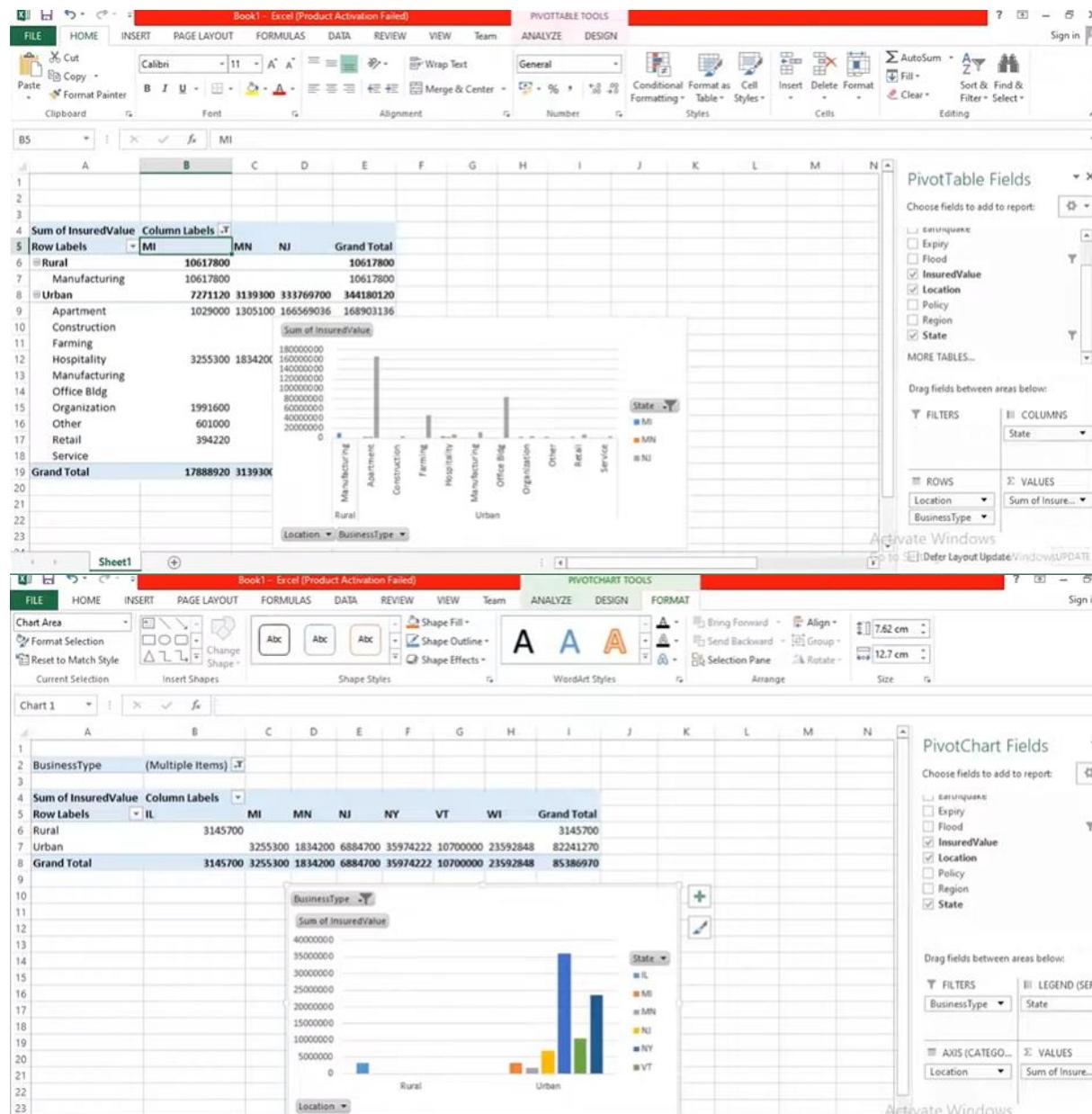
- Choose a chart type (e.g., Column, Line, Pie, Bar).
- Click **OK**.

3. Customize the Chart

- Add labels, titles, and format the chart as needed.
-

Step 4: Refresh Data (If Needed)

- If the data in the warehouse updates, right-click on the Pivot Table and select "Refresh" to get the latest data.



1B. Import the cube in Microsoft Excel and create the Pivot table and Pivot Chart to perform data analysis.

Step 1: Connect to an OLAP Cube in Excel

1. Open Microsoft Excel

- Launch Excel on your computer.

2. Go to the Data Tab

- Click on "**Get Data**" (Power Query) > "**From Database**" > "**From Analysis Services**" (Microsoft's OLAP server).

3. Enter Connection Details

- In the "**Data Connection Wizard**":
 - Enter the **Server Name** where the OLAP cube is hosted.
 - Click **Next**.

4. Select the OLAP Cube

- Choose the appropriate **database** and **cube** from the list.
- Click **Next** and then **Finish**.

5. Import Data into a PivotTable

- Choose "**PivotTable Report**" when prompted.
- Click **OK** to place the PivotTable in a new worksheet.

Step 2: Create a PivotTable for Analysis

1. Define Data Fields

- In the **PivotTable Fields Pane**, drag and drop fields into the respective areas:

- **Rows** (e.g., Product Category, Region).
- **Columns** (e.g., Year, Quarter).
- **Values** (e.g., Sales Amount, Profit).
- **Filters** (Optional, e.g., Country, Time Period).

2. Summarize & Analyze Data

- Apply filters, sort, and group data as needed.
 - Use calculated fields to derive additional insights.
-

Step 3: Create a PivotChart for Visualization

1. Click on the PivotTable

- Go to the **Insert Tab** > Click **PivotChart**.

2. Select Chart Type

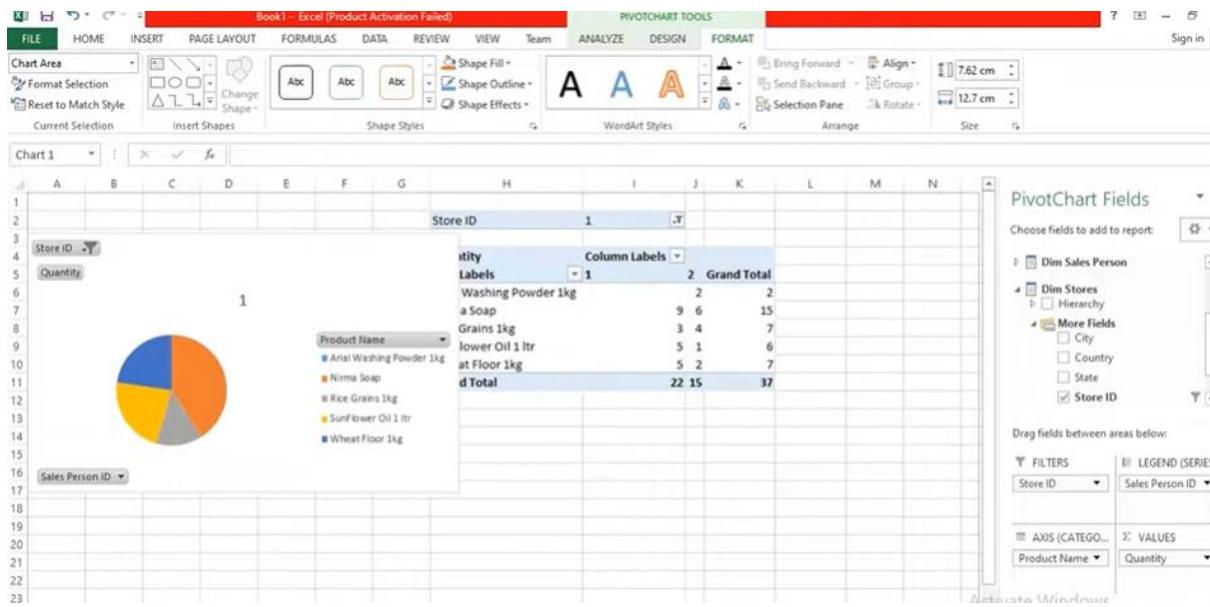
- Choose a suitable chart (e.g., Column, Line, Pie, Bar).
- Click **OK**.

3. Customize the Chart

- Add titles, labels, and format colors.
 - Apply slicers for interactive filtering.
-

Step 4: Refresh Data for Real-Time Analysis

- If the OLAP cube updates, right-click on the PivotTable and select "**Refresh**" to pull the latest data.



PRACTICAL 2

Practical 2 - Apply the what – if Analysis for data visualization.
Design and generate necessary reports based on the data warehouse data. Use Excel.

Step 1: Import Data Warehouse Data into Excel

- 1. Open Excel**
- 2. Go to the Data Tab**
 - Click "**Get Data**" > "**From Other Sources**" > "**From SQL Server Database**" (or any relevant source).
- 3. Enter Connection Details**
 - Provide **Server Name** and **Database Name**, then click **OK**.
- 4. Load Data**
 - Select required tables/views and click **Load**.

Step 2: Create PivotTables and PivotCharts

- 1. Insert a PivotTable**
 - Click anywhere inside the data.
 - Go to **Insert Tab** > Click **PivotTable**.
 - Choose a worksheet and click **OK**.
 - Drag and drop fields into Rows, Columns, Values, and Filters.
- 2. Create a PivotChart**
 - Select the PivotTable.
 - Go to **Insert Tab** > Click **PivotChart**.
 - Choose an appropriate chart type (Bar, Line, Pie).

- Format the chart for better visualization.

Step 3: Apply What-If Analysis

1. Scenario Manager (Best, Worst, and Expected Case Analysis)

- Go to **Data Tab** > Click **What-If Analysis** > Select **Scenario Manager**.
- Click **Add** and define different scenarios (e.g., Sales Increase, Revenue Drop).
- Enter different values for key inputs like **Sales Growth, Costs, Profit Margins**.
- Click **OK** and **Show** to compare scenarios.

2. Goal Seek (Find the Required Input for a Target Value)

- Go to **Data Tab** > Click **What-If Analysis** > Select **Goal Seek**.
- Set a target value for **Revenue or Profit** and change **Sales Growth or Price** to achieve it.
- Click **OK** to get the results.

3. Data Tables (Analyze Multiple Inputs)

- Select a table range with different **Price, Sales, and Profit**.
- Go to **Data Tab** > Click **What-If Analysis** > Select **Data Table**.
- Define **Row Input Cell** and **Column Input Cell** for changing values.
- Click **OK** to see the impact.

Step 4: Generate Reports Based on Analysis

- **Summary Report**
 - From Scenario Manager, click **Summary** to generate a comparison report.

- **Charts for Visualization**
 - Use **PivotCharts** and **Conditional Formatting** to highlight insights.
- **Dashboard Creation**
 - Combine **PivotTables**, **Charts**, and **Slicers** for an interactive dashboard.

PRACTICAL 3

Practical 3 - Perform the data classification using classification algorithm using R/Python

Code:

```
pip install pandas numpy scikit-learn matplotlib seaborn
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
from sklearn.datasets import load_iris

# Load the dataset
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target # Adding target labels

# Display first five rows
print(df.head())
# Splitting data into features (X) and target (y)
```

```
X = df.drop('target', axis=1) # Features
y = df['target'] # Target labels

# Splitting 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)

# Standardizing the data (important for some classifiers)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

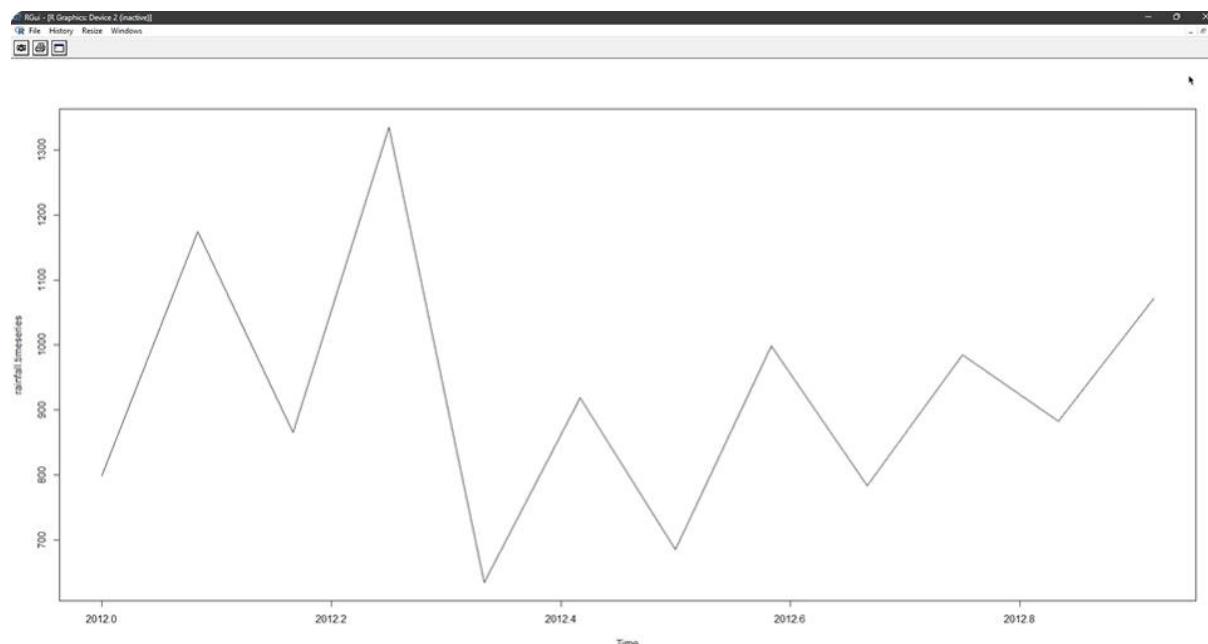
# Initialize the model
model = RandomForestClassifier(n_estimators=100,
random_state=42)

# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Classification report
```

```
print("Classification Report:\n", classification_report(y_test,  
y_pred))  
  
# Confusion matrix  
conf_matrix = confusion_matrix(y_test, y_pred)  
sns.heatmap(conf_matrix, annot=True, cmap="Blues", fmt="d")  
plt.xlabel("Predicted")  
plt.ylabel("Actual")  
plt.title("Confusion Matrix")  
plt.show()
```



PRACTICAL 4

Practical 4 - Perform the data clustering using clustering algorithm using R/Python.

Code:

```
pip install pandas numpy scikit-learn matplotlib seaborn
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler
# Generate sample data with 3 clusters
X, y = make_blobs(n_samples=300, centers=3, random_state=42,
cluster_std=1.0)

# Convert to DataFrame
df = pd.DataFrame(X, columns=['Feature1', 'Feature2'])

# Display first five rows
print(df.head())
# Standardize the data (important for distance-based clustering)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
# Define the number of clusters
k = 3

# Train K-Means model
kmeans = KMeans(n_clusters=k, random_state=42)
df['Cluster'] = kmeans.fit_predict(X_scaled)

# Cluster centers
centers = kmeans.cluster_centers_
plt.figure(figsize=(8, 6))

# Scatter plot of clusters
sns.scatterplot(x=df['Feature1'], y=df['Feature2'],
hue=df['Cluster'], palette='viridis', s=50)

# Plot cluster centers
plt.scatter(centers[:, 0], centers[:, 1], c='red', marker='X', s=200,
label='Centroids')

plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('K-Means Clustering Visualization')
plt.legend()
plt.show()
inertia = []
```

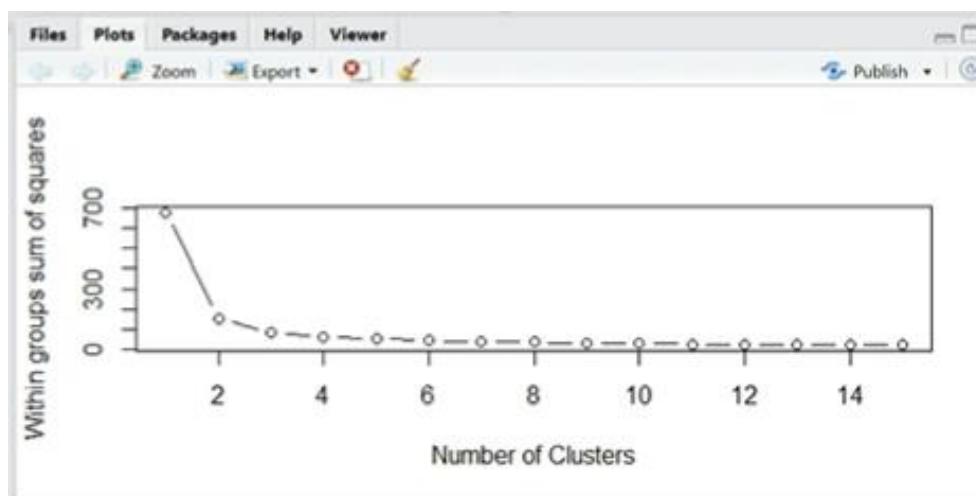
```
K_range = range(1, 10)
```

```
for k in K_range:
```

```
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)
```

```
# Plot the elbow curve
```

```
plt.figure(figsize=(8, 6))
plt.plot(K_range, inertia, marker='o', linestyle='--')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal k')
plt.show()
```



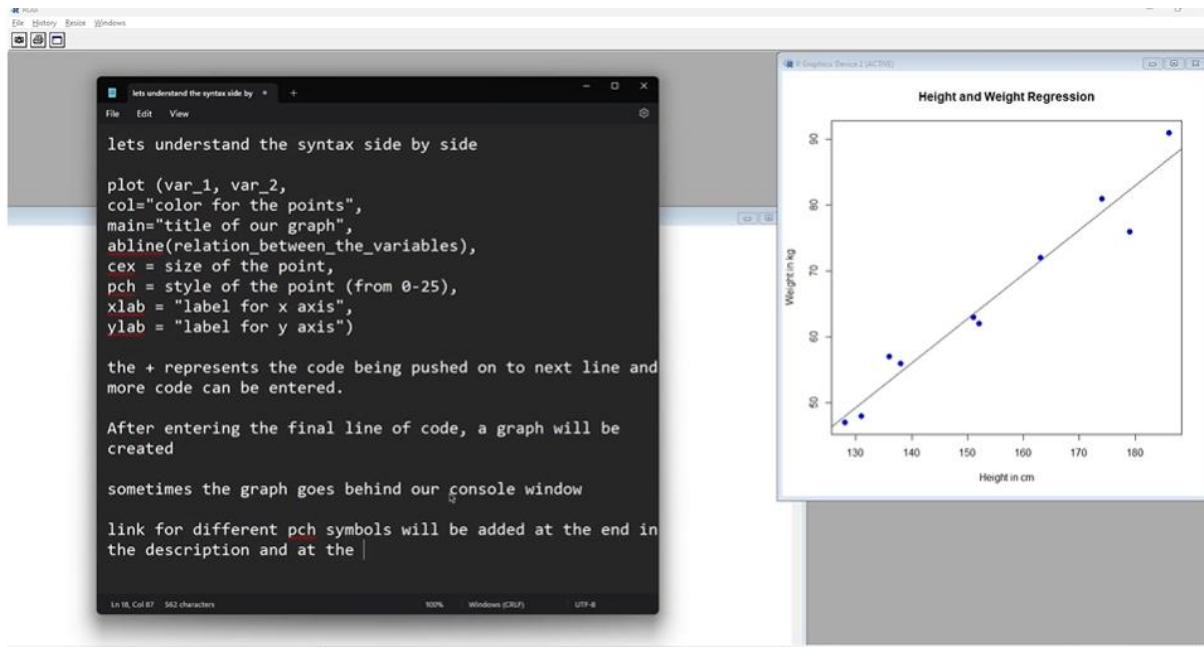
PRACTICAL 5

Practical 5 - Perform the Linear regression on the given data warehouse data using R/Python.

Code:

```
plot (var_1, var_2,
      col="color for the points",
      main="title of our graph",
      abline(relation_between_the_variables),
      cex = size of the point,
      pch = style of the point (from 0-25),
      xlab = "label for x axis",
      ylab = "label for y axis")
#x - represents height (in cms)
#y - represents weight (in kg)
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
#perform a linear regression where we specify the dependent and
the independent variable in the following manner:
#Syntax: lm(dependent_var ~ independent_var)
#in our case y is dependent and x is independent
relation <- lm(y ~ x)
#predicting the weight i.e. (y) from a given value of the height i.e.
(x) = 170 ; create a new data frame of the value
a <- data.frame(x=170)
```

```
#to find the result of our prediction we use the predict function  
with the relation and the dataframe  
  
#Syntax: predict(relation,data.frame)  
result <- predict(relation,a)  
print(result)  
  
#plotting the data on a graph  
  
#Syntax: plot(var_1,var_2,col = "point_color", main="title" ,  
abline("relation_between_lines"),cex = point_size , pch =  
shape_of_point , xlab = "label for x axis" , ylab = "label for y  
axis")  
  
plot(x, y,  
col = "blue",  
main = "Height and Weight Regression",  
abline(lm(y ~ x)),  
cex = 1.3,  
pch = 16,  
xlab = "Height in cm",  
ylab = "Weight in kg")  
  
#pch symbols image-link (in desc)  
https://r-charts.com/en/tags/base-r/pch-symbols\_files/figure-html/pch-symbols.png
```



PRACTICAL 6

Practical 6 - Perform the logistic regression on the given data warehouse data using R/Python.

Code:

```
pip install pandas numpy scikit-learn matplotlib seaborn
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
# Simulated dataset
data = {
    "Age": [25, 45, 35, 50, 23, 40, 30, 60, 27, 55],
    "Income": [30000, 80000, 50000, 90000, 25000, 70000, 45000,
100000, 32000, 85000],
    "Purchased": [0, 1, 0, 1, 0, 1, 0, 1, 0, 1] # Target variable (1 =
Purchased, 0 = Not Purchased)
}
# Convert to DataFrame
df = pd.DataFrame(data)
```

```
# Display first five rows
print(df.head())

# Define independent (X) and dependent (y) variables
X = df[['Age', 'Income']] # Features
y = df['Purchased']      # Target variable

# Split data into training (80%) and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Standardize the data (important for Logistic Regression)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize and train the model
model = LogisticRegression()
model.fit(X_train, y_train)

# Get predictions
y_pred = model.predict(X_test)

# Model performance metrics
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Classification report
print("Classification Report:\n", classification_report(y_test,
y_pred))
```

```
# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot=True, cmap="Blues", fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()

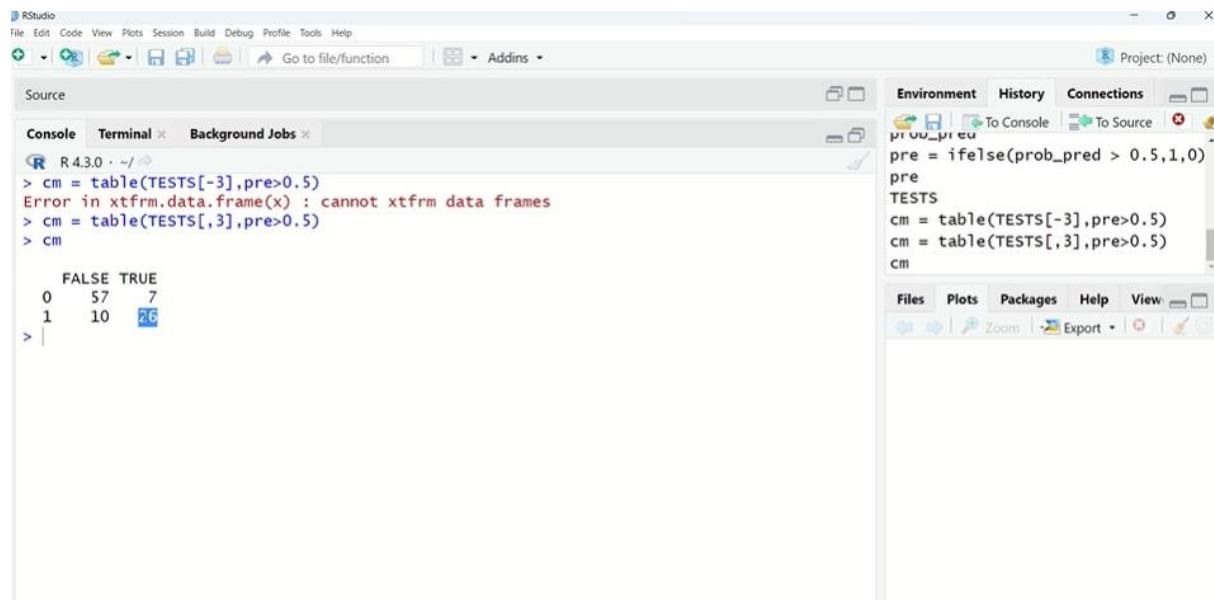
from matplotlib.colors import ListedColormap

# Generate mesh grid
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start=X_set[:, 0].min() - 1,
stop=X_set[:, 0].max() + 1, step=0.1),
np.arange(start=X_set[:, 1].min() - 1, stop=X_set[:, 1].max() + 1, step=1000))

# Plot decision boundary
plt.contourf(X1, X2, model.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
alpha=0.3, cmap=ListedColormap(('red', 'green')))

# Scatter plot of training data
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
c=ListedColormap(('red', 'green'))(i), label=j)
```

```
plt.xlabel('Age')
plt.ylabel('Income')
plt.title('Logistic Regression Decision Boundary')
plt.legend()
plt.show()
```



PRACTICAL 7

Practical 7 - Write a Python program to read data from a CSV file, perform simple data analysis, and generate basic insights.
(Use Pandas is a Python library).

```
pip install pandas numpy matplotlib seaborn
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Read Data from CSV File
file_path = "data.csv" # Replace with your CSV file path
df = pd.read_csv(file_path)

# Display Basic Information
print("\n ◆ First 5 Rows of Dataset:")
print(df.head())

print("\n ◆ Summary Statistics:")
print(df.describe())

print("\n ◆ Data Types and Missing Values:")
print(df.info())
```

```
# Handling Missing Values
missing_values = df.isnull().sum()
print("\n ◆ Missing Values Count:")
print(missing_values[missing_values > 0])

# Fill missing values with mean (if numerical)
df.fillna(df.mean(), inplace=True)

# Perform Basic Analysis
print("\n ◆ Column-Wise Unique Values Count:")
print(df.nunique())

# Generate Basic Insights
print("\n ◆ Correlation Matrix:")
print(df.corr())

# Data Visualization
plt.figure(figsize=(8, 6))
sns.heatmap(df.corr(), annot=True, cmap="coolwarm",
            linewidths=0.5)
plt.title("Correlation Heatmap")
plt.show()

# Histogram for Numeric Columns
df.hist(figsize=(10, 8), bins=20, color='skyblue', edgecolor='black')
```

```
plt.suptitle("Histogram of Numeric Variables")
plt.show()

# Save Cleaned Data to a New CSV
df.to_csv("cleaned_data.csv", index=False)

print("\n✓ Data Cleaning Completed. Saved as
'cleaned_data.csv'")
```

PRACTICAL 8

Practical 8 - Perform data visualization

8A. Perform data visualization using Python on any sales data.

Code:

```
pip install pandas numpy matplotlib seaborn
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
# Sample Sales Data
data = {
    "Date": pd.date_range(start="2023-01-01", periods=12,
freq='M'),
    "Sales": [5000, 7000, 8000, 6500, 7200, 9000, 11000, 10500,
9500, 9800, 12000, 13000],
    "Profit": [800, 1200, 1500, 1000, 1300, 1700, 2200, 2100, 1900,
2000, 2500, 2700],
    "Category": ["Electronics", "Clothing", "Electronics",
"Furniture", "Clothing", "Electronics",
        "Furniture", "Clothing", "Electronics", "Furniture",
"Clothing", "Electronics"]
}
# Convert to DataFrame
df = pd.DataFrame(data)
```

```
# Display first five rows
```

```
print(df.head())
```

1. Line Chart - Monthly Sales Trend

```
plt.figure(figsize=(10, 5))
```

```
plt.plot(df["Date"], df["Sales"], marker='o', linestyle='-',  
color='blue', label="Sales")
```

```
plt.xlabel("Month")
```

```
plt.ylabel("Sales ($)")
```

```
plt.title("Monthly Sales Trend")
```

```
plt.legend()
```

```
plt.grid(True)
```

```
plt.xticks(rotation=45)
```

```
plt.show()
```

2. Bar Chart - Sales by Category

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

```
sns.barplot(x=df["Category"], y=df["Sales"], palette="viridis")
```

```
plt.xlabel("Product Category")
```

```
plt.ylabel("Sales ($)")
```

```
plt.title("Sales by Product Category")
```

```
plt.show()
```

3.Scatter Plot - Sales vs. Profit

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

```
sns.scatterplot(x=df["Sales"], y=df["Profit"], hue=df["Category"],  
palette="deep", s=100)  
  
plt.xlabel("Sales ($)")  
plt.ylabel("Profit ($)")  
plt.title("Sales vs. Profit")  
plt.show()
```

4. Pie Chart - Sales Contribution by Category

```
plt.figure(figsize=(7, 7))  
  
df.groupby("Category")["Sales"].sum().plot.pie(autopct='%.1f%%',  
colors=["skyblue", "lightcoral", "gold"], startangle=90)  
  
plt.title("Sales Contribution by Category")  
plt.ylabel("")  
plt.show()
```

8B. Perform data visualization using PowerBI on any sales data.

Step 1: Prepare the Sales Data

Use an **Excel or CSV** file with the following sales data structure:

Date	Sales	Profit	Category	Region	Customer_Type
01-01-2023	5000	800	Electronics	East	Retail
01-02-2023	7000	1200	Clothing	West	Wholesale
01-03-2023	8000	1500	Electronics	North	Retail
01-04-2023	6500	1000	Furniture	South	Wholesale
...

Save this as SalesData.xlsx or SalesData.csv

Step 2: Load Data into Power BI

- 1. Open Power BI Desktop.**
- 2. Click on "Get Data" → "Excel" or "CSV"**
- 3. Select your SalesData.xlsx or SalesData.csv file and click Load**
- 4. The data will appear in the Data Model**

Step 3: Create Visualizations

1. Line Chart - Monthly Sales Trend

- Go to Visualizations Pane**
- Select Line Chart**
- Drag "Date" to X-axis**

- Drag "**Sales**" to **Y-axis**
- Format the chart (title, labels, colors)

 **Insight:** Shows how sales change over time

2. Bar Chart - Sales by Category

- Select **Clustered Bar Chart**
- Drag "**Category**" to **X-axis**
- Drag "**Sales**" to **Y-axis**
- Sort by descending order

 **Insight:** Identifies the best-selling categories

3. Scatter Chart - Sales vs. Profit

- Select **Scatter Chart**
- Drag "**Sales**" to X-axis and "**Profit**" to Y-axis
- Drag "**Category**" to the Legend field

 **Insight:** Shows the relationship between **Sales and Profit**

4. Pie Chart - Sales by Region

- Select **Pie Chart**
- Drag "**Region**" to **Legend**
- Drag "**Sales**" to **Values**

 **Insight:** Displays **regional sales contribution**

5.KPI Card - Total Sales

- Select **Card Visual**
- Drag "**Sales**" to **Values**
- Format to show currency (e.g., \$)

 **Insight:** Highlights the total sales revenue

Step 4: Create Interactive Dashboards

Use **Slicers** for filtering by **Region**, **Category**, or **Customer Type**
Add **Tooltips** to show detailed insights
Apply **Conditional Formatting** to highlight trends

Step 5: Publish and Share

1. Click **File** → **Publish** → **Power BI Service**
2. Share the dashboard link with your team