**Practical No: 1**

Aim: **Introduction to Excel**

**• Perform conditional formatting on a dataset using various criteria.**

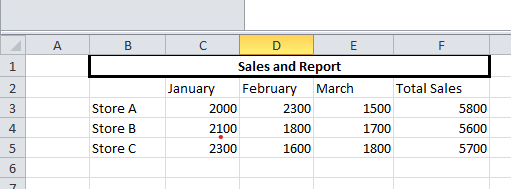
**• Create a pivot table to analyze and summarize data.**

**• Use VLOOKUP function to retrieve information from a different worksheet or table.**

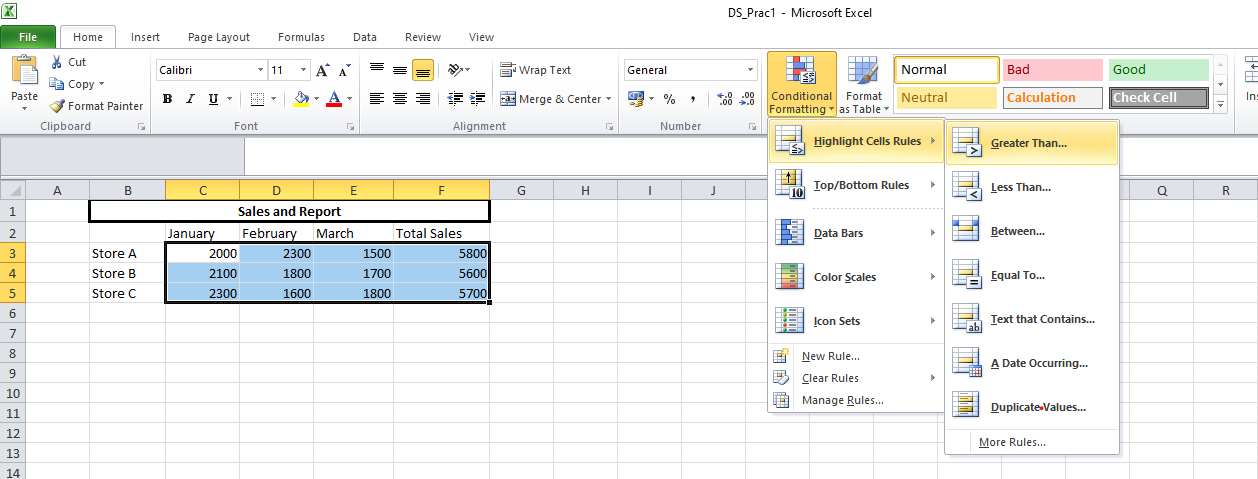
# • Perform what-if analysis using Goal Seek to determine input values for desired output.

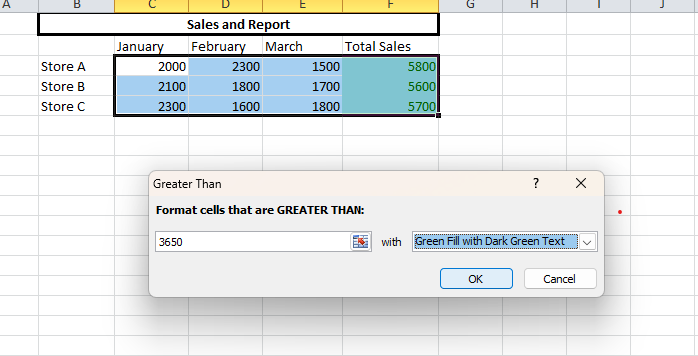
**i) Introduction to Excel**

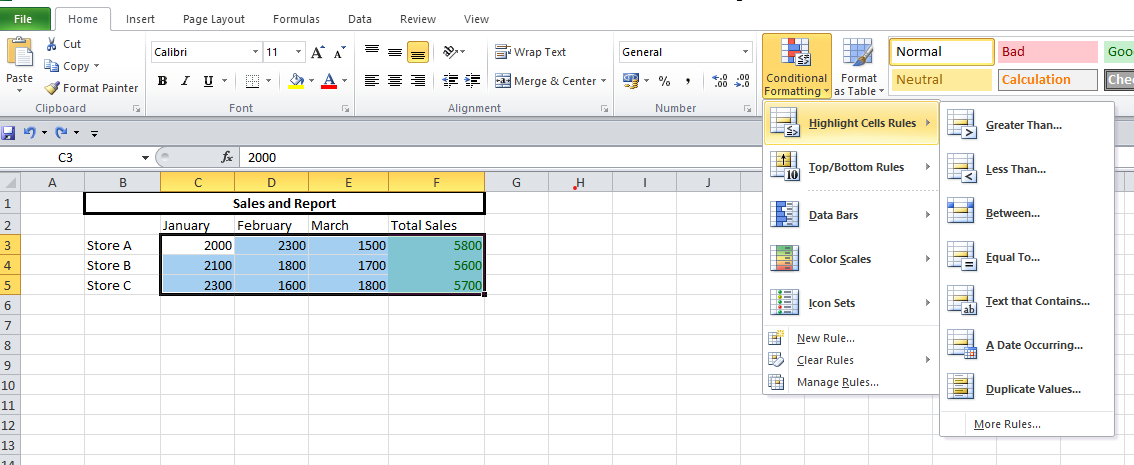
**A. Perform conditional formatting on a dataset using various criteria.**

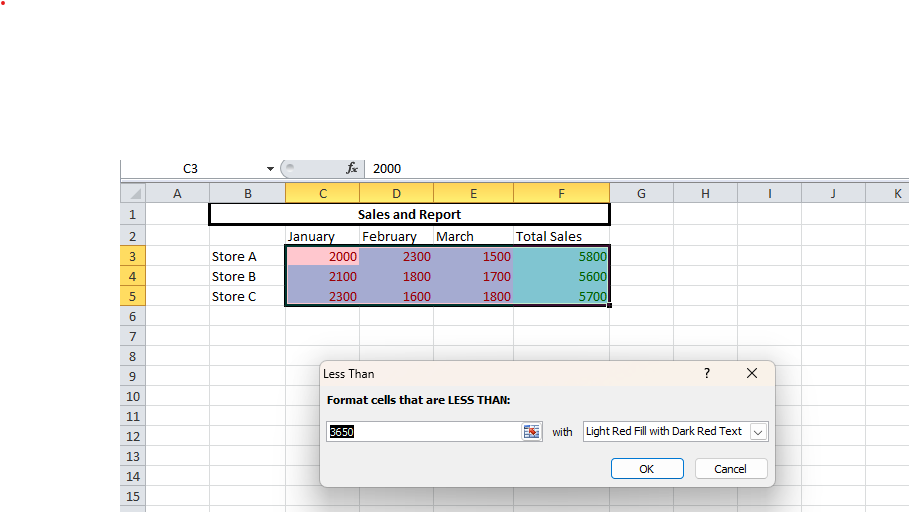
****

Steps Step 1: Go to conditional formatting > Greater Than

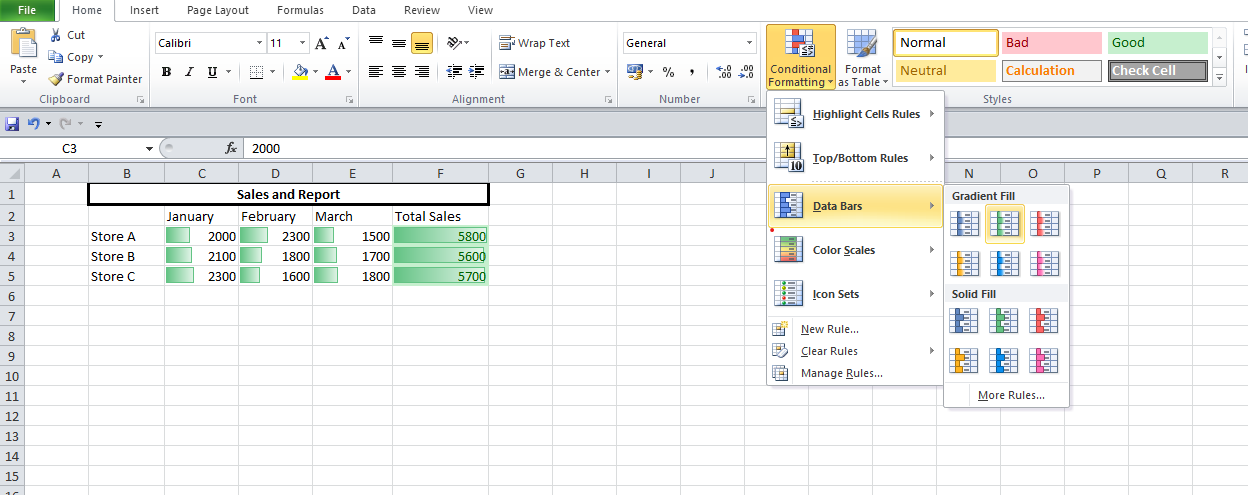
****

Step 2: Enter the greater than filter value for example 3650.  
****



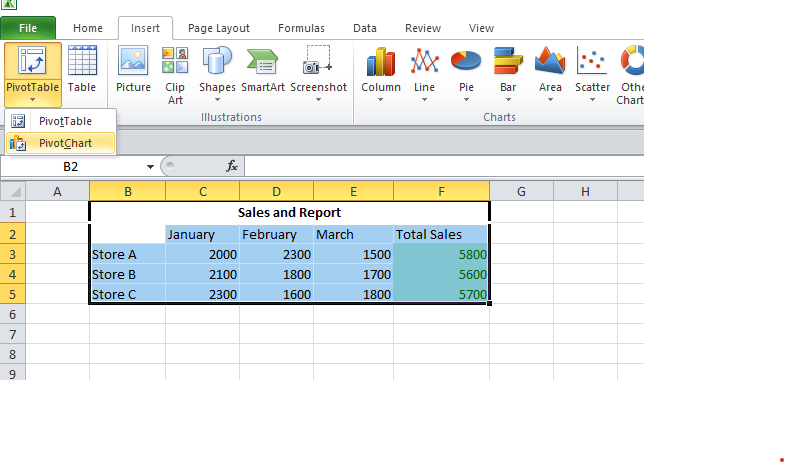
****

Step 3: Go to Data Bars > Solid Fill in conditional formatting.

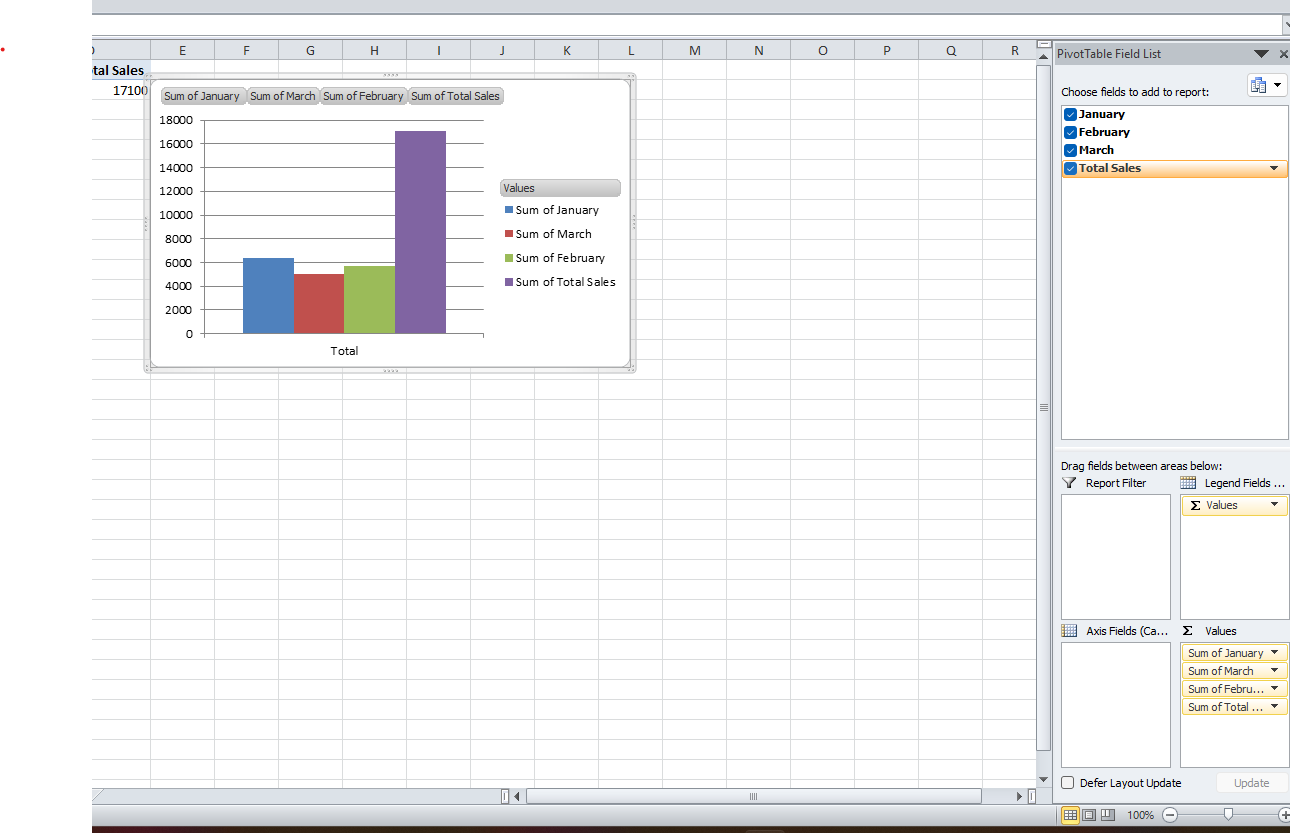


**B. Create a pivot table to analyse and summarize data.**

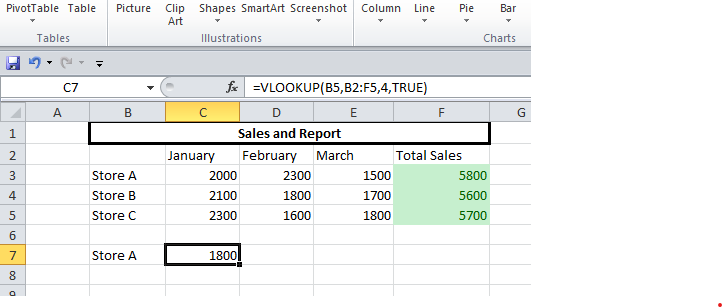
Step 1: select the entire table and go to Insert tab PivotChart > Pivotchart .

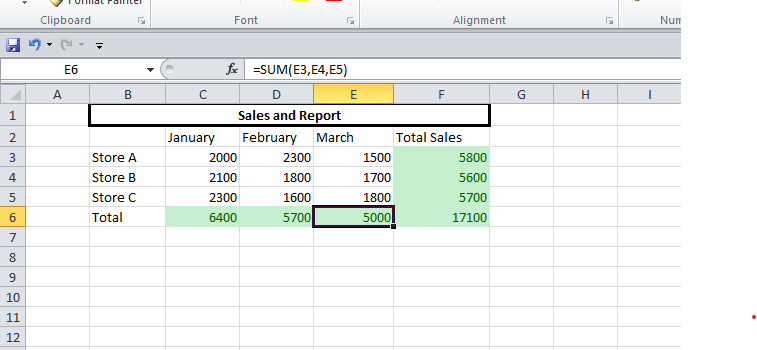


Step 2: Select “New worksheet” in the create pivot chart window.  
  
Step 3: Select and drag attributes in the below boxes.

****

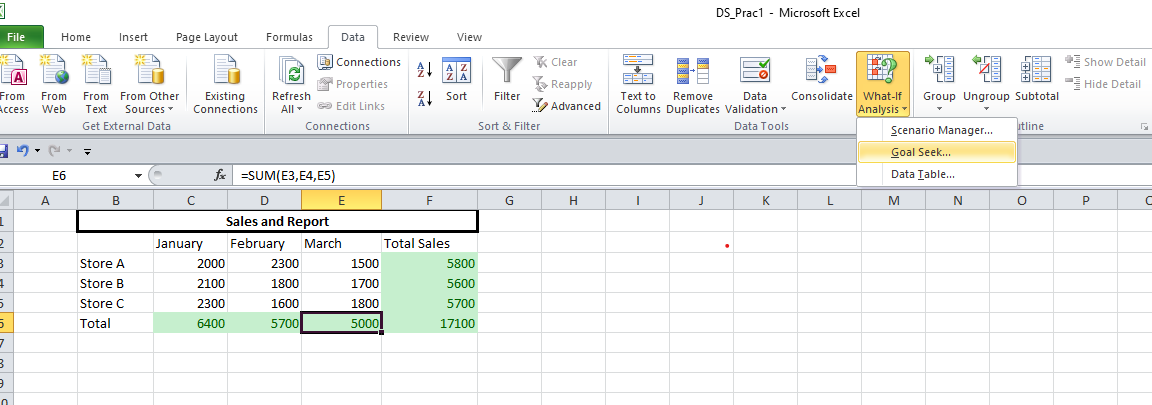
A. Use VLOOKUP function to retrieve information from a different worksheet or table. Steps: Step 1: click on an empty cell and type the following command. =VLOOKUP(B5, B2:F5,4, TRUE)

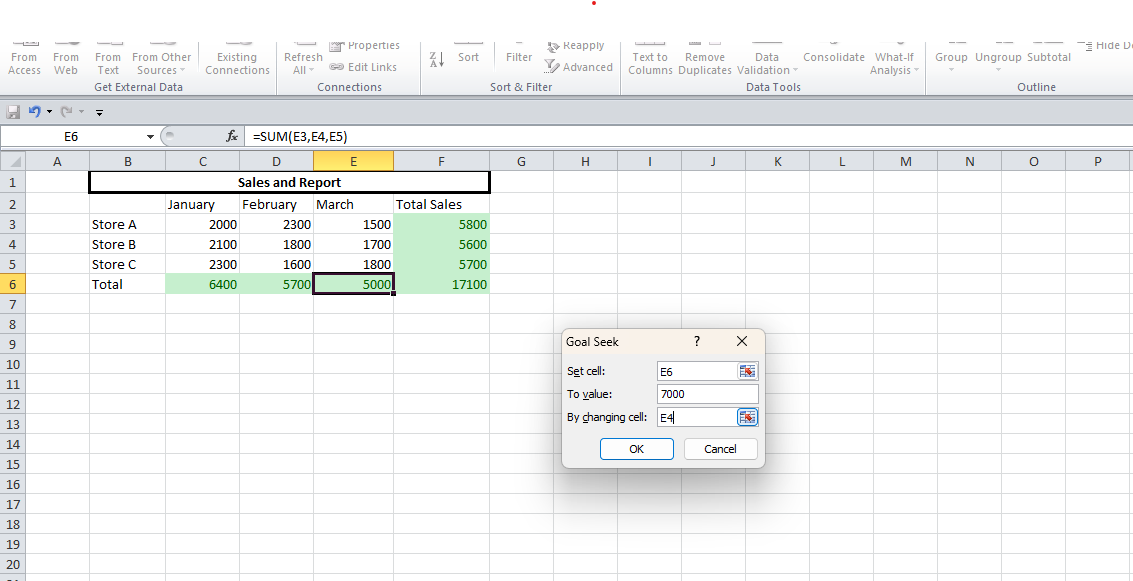
****

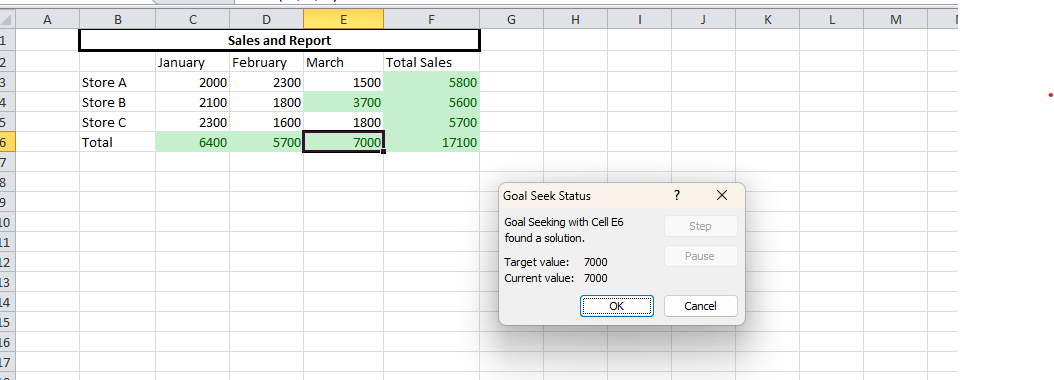
****

**B. Perform what-if analysis using Goal Seek to determine input values for desired output.**

Steps-

Step 1: In the Data tab go to the what if analysis>Goal seek.  


Step 2: Fill the information in the window accordingly and click ok.  




**Practical No: 2**

Aim: **Data Frames and Basic Data Pre-processing**

**• Read data from CSV and JSON files into a data frame.**

**• Perform basic data pre-processing tasks such as handling missing values and outliers.**

**• Manipulate and transform data using functions like filtering, sorting, and grouping.**

**Data Frames and Basic Data Pre-processing**

**A. Read data from CSV and JSON files into a data frame**

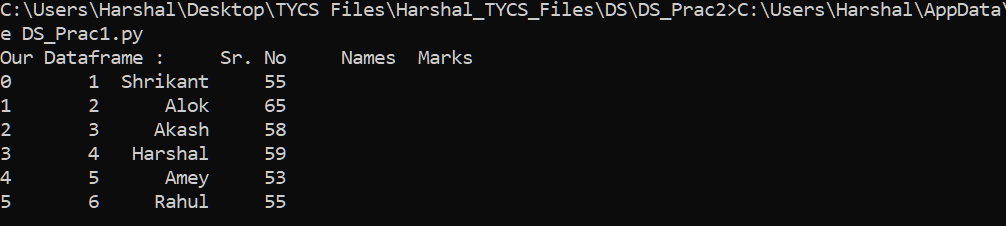
**1)**

import pandas as pd

dataframe = pd.read\_csv("C:\\Users\COMPUTER LAB\Documents\Harshal\_TYCS\_Files\DS\DS\_Prac\_2.csv")

print("Our Dataframe : ",dataframe)

**Output:**

****

**2)**

import pandas as pd

import json

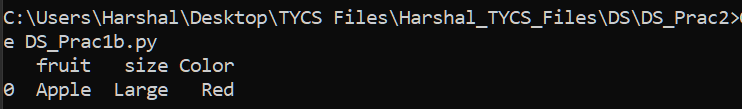
a = '[[["fruit","Apple"],["size","Large"],["Color","Red"]]]'

parsed\_data=json.loads(a)

data\_dict=dict(parsed\_data[0])

data=pd.DataFrame([data\_dict])

print(data)

****

# 2 b) Perform basic data pre-processing task such as handling missing values and outliers

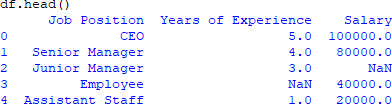
import pandas as pd

# Creating the dataframe as shown above

df = pd.DataFrame({'Job Position': ['CEO', 'Senior Manager', 'Junior Manager', 'Employee', 'Assistant Staff'], 'Years of Experience':[5, 4, 3, None, 1],

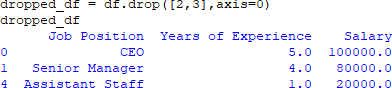
'Salary':[100000,80000,None,40000, 20000]})

# Viewing the contents of the dataframe df.head()



# Data Removal

# Dropping the 2nd and 3rd index

dropped\_df = df.drop([2,3],axis=0)   
# Viewing the dataframe dropped\_df

# Fill missing value through statistical imputation

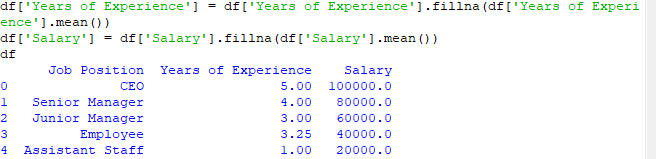
# Filling each column with their mean values

df['Years of Experience'] = df['Years of Experience'].fillna(df['Years of Experience'].mean())

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

# Viewing the dataframe

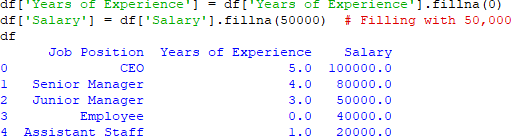
df



# Filling with specific number

df = pd.DataFrame({'Job Position': ['CEO', 'Senior Manager', 'Junior Manager', 'Employee', 'Assistant Staff'], 'Years of Experience':[5, 4, 3, None, 1], 'Salary':[100000,80000,None,40000, 20000]})

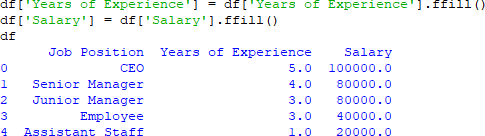
df.head()

df['Years of Experience'] = df['Years of Experience'].fillna(0) df['Salary'] = df['Salary'].fillna(50000)#Filling with 50,000 df

# Filling with Forward Fill (Previous Value)

df = pd.DataFrame({'Job Position': ['CEO', 'Senior Manager', 'Junior Manager', 'Employee', 'Assistant Staff'], 'Years of Experience':[5, 4, 3, None, 1], 'Salary':[100000,80000,None,40000, 20000]})

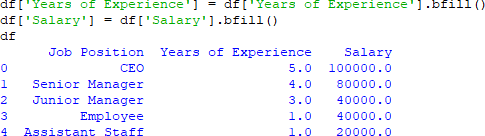
df.head()



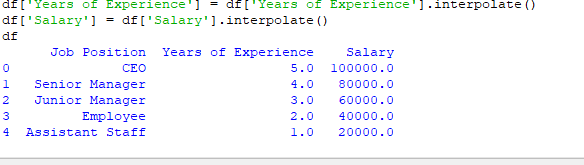
# Filling with Backward Fill (Next Value)

df = pd.DataFrame({'Job Position': ['CEO', 'Senior Manager', 'Junior Manager', 'Employee', 'Assistant Staff'], 'Years of Experience':[5, 4, 3, None, 1], 'Salary':[100000,80000,None,40000, 20000]})

df.head()



1. **Filling with an Interpolated Value**

****

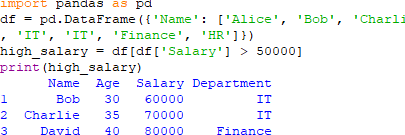
**2 c) Manipulate and transform data using functions like filtering, sorting, and grouping**

1. **Filtering Data**

import pandas as pd

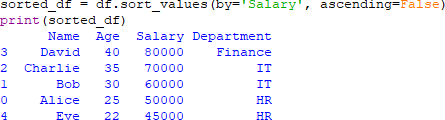
df = pd.DataFrame({'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],'Age': [25, 30, 35, 40, 22],'Salary': [50000, 60000, 70000, 80000, 45000],'Department':['HR', 'IT', 'IT', 'Finance', 'HR']})

high\_salary = df[df['Salary'] > 50000] print(high\_salary)



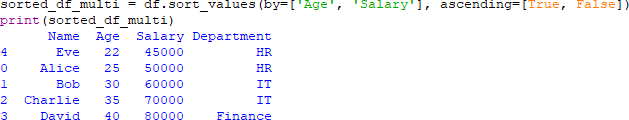
# Sorting Data: sort DataFrame values by one or multiple columns.

# Sorting by Salary in descending order

sorted\_df = df.sort\_values(by='Salary', ascending=False) print(sorted\_df)

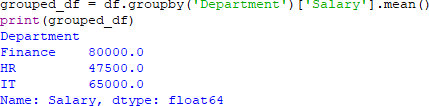
# Sorting by Age (ascending) and then Salary (descending)

sorted\_df\_multi = df.sort\_values(by=['Age', 'Salary'], ascending=[True, False]) print(sorted\_df\_multi)



# Grouping Data: group data based on a categorical column and apply aggregate functions.

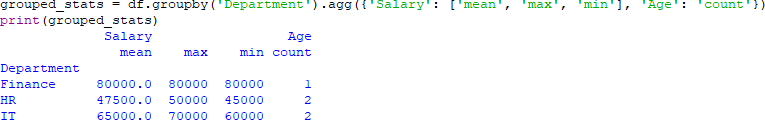
# Grouping by 'Department' and calculating the average salary grouped\_df = df.groupby('Department')['Salary'].mean() print(grouped\_df)



# Grouping by 'Department' and getting multiple aggregations

grouped\_stats = df.groupby('Department').agg({'Salary': ['mean', 'max', 'min'], 'Age': 'count'})

print(grouped\_stats)



**Practical 3**

**Feature Scaling and Dummification**

**3 a) Apply feature-scaling techniques like standardization and normalization to numerical features.**

Feature scaling ensures that numerical features are in the same range, which improves the performance of machine learning models. There are two common scaling techniques:

1 **Standardization (Z-score normalization)**

2 **Normalization (Min-Max scaling)**

**1 Standardization (Z-score Normalization)**

* Centers data around 0 with a standard deviation of 1.



where **μ** is the mean and **σ** is the standard deviation.

**Normalization (Min-Max Scaling)**

* Scales values between **0 and 1**.
* Formula



Code:

# Import libraries

import pandas as pd

from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Sample DataFrame

df = pd.DataFrame({

'Years of Experience': [1, 3, 5, 7, 9],

'Salary': [20000, 40000, 60000, 80000, 100000]

})

print("Original DataFrame:")

print(df)

# 1 Applying Standardization (Z-score normalization)

scaler = StandardScaler()

df[['Years of Experience', 'Salary']] = scaler.fit\_transform(df[['Years of Experience', 'Salary']])

print("\nData after Standardization (Z-score normalization):")

print(df)

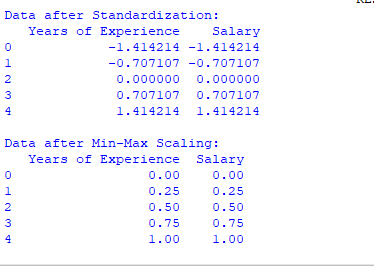
# 2 Applying Min-Max Scaling

minmax\_scaler = MinMaxScaler()

df[['Years of Experience', 'Salary']] = minmax\_scaler.fit\_transform(df[['Years of Experience', 'Salary']])

print("\nData after Min-Max Scaling (0 to 1 range):")

print(df)



**3 B) Perform feature Dummification to convert categorical variables into numerical representations.**

**Dummification (One-Hot Encoding for Categorical Variables)**

Dummification converts categorical variables into numerical form using **One-Hot Encoding**.

**Code:**

# Import library

import pandas as pd

# Sample DataFrame with categorical columns

df = pd.DataFrame({

'Job Role': ['Manager', 'Analyst', 'Engineer', 'Analyst', 'Manager'],

'Department': ['HR', 'Finance', 'Engineering', 'Finance', 'HR']

})

# Perform one-hot encoding (dummification)

df\_encoded = pd.get\_dummies(df, columns=['Job Role', 'Department'], drop\_first=False)

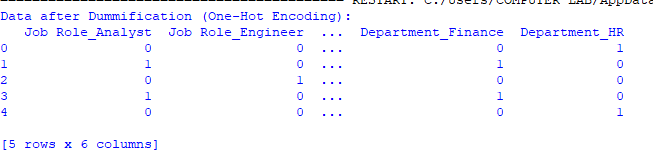
# Ensure all dummy variables are integers (0 and 1)

df\_encoded = df\_encoded.astype(int)

# Display the resulting DataFrame

print("Data after Dummification (One-Hot Encoding):\n")

print(df\_encoded)



# Practical 4

# Hypothesis Testing

Formulate null and alternative hypotheses for a given problem.

Conduct a hypothesis test using appropriate statistical tests (e.g., t-test, chi-square test). Interpret the results and draw conclusions based on the test outcomes.

# T-test (Comparing Salaries Between Marketing and Sales Departments):

The goal is to test if the average salaries in the **Marketing** and **Sales** departments are the same or different.

# Null Hypothesis (H₀):

* The null hypothesis assumes that there is **no significant difference** between the average salaries of employees in the Marketing and Sales departments.
* **Mathematically**: **H₀: μ₁ = μ₂** Where:
  + μ₁ is the average salary in the Marketing department
  + μ₂ is the average salary in the Sales department

# Alternative Hypothesis (H₁):

* The alternative hypothesis assumes that there is a **significant difference** between the average salaries in the Marketing and Sales departments.

# Mathematically: H₁: μ₁ ≠ μ₂

This is a two-tailed test, meaning we're testing for any difference (higher or lower) in the average salaries of the two departments.

# Two-sample t-test

(ttest\_ind) to compare the means of the two independent groups (Marketing and Sales). This test will help us determine if there is enough evidence to reject the null hypothesis.

Code:  
# Import libraries

import numpy as np

from scipy import stats

# Data: Salaries of employees in Marketing and Sales departments

marketing\_salaries = [50000, 55000, 60000, 52000, 58000, 49000]

sales\_salaries = [48000, 51000, 54000, 53000, 50000, 46000]

# Perform the two-sample t-test

t\_stat, p\_value = stats.ttest\_ind(marketing\_salaries, sales\_salaries)

# Calculate degrees of freedom for t-test

n1 = len(marketing\_salaries)

n2 = len(sales\_salaries)

df\_ttest = n1 + n2 - 2 # Degrees of freedom formula for independent two-sample t-test

# Display results

print(f"T-statistic: {t\_stat:.2f}")

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

print(f"Degrees of Freedom: {df\_ttest}")

# Hypothesis interpretation

alpha = 0.05

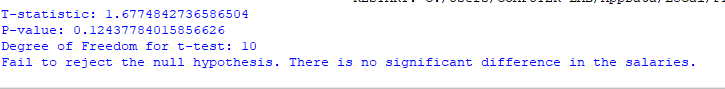
if p\_value < alpha:

print("Reject the null hypothesis: There is a significant difference in the salaries.")

else:

print("Fail to reject the null hypothesis: There is no significant difference in the salaries.")

# Interpreting the Results:

****

**Conclusion:**

Based on the **p-value**:

* Since **p-value > alpha**, we **fail to reject** the null hypothesis.
* Therefore, we conclude that there is **no significant difference** in the average salaries between the Marketing and Sales departments.

**Code for Chi-Square Test:**

# Import necessary library

from scipy.stats import chi2\_contingency

# Data: Frequency of employees in each department across two regions (North, South)

data = [

[30, 20, 50], # North Region (Marketing, Sales, HR)

[20, 30, 40] # South Region (Marketing, Sales, HR)

]

# Perform the Chi-Square test

chi2\_stat, p\_value, dof, expected = chi2\_contingency(data)

# Display results

print(f"Chi-square Statistic: {chi2\_stat:.2f}")

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

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

print(f"Expected Frequencies:\n{expected}")

# Hypothesis Interpretation

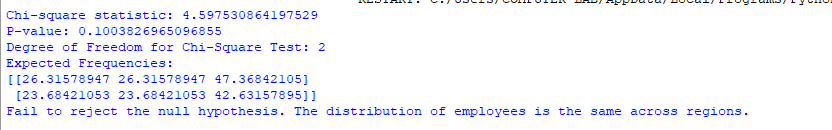
alpha = 0.05

if p\_value < alpha:

print("Reject the null hypothesis: The distribution of employees is different across regions.")

else:

print("Fail to reject the null hypothesis: The distribution of employees is the same across regions.")



# Practical 5

Perform one-way ANOVA to compare means across multiple groups. Conduct post-hoc tests to identify significant differences

# ANOVA (Analysis of Variance)

ANOVA is used to test if there are statistically significant differences between the means of three or more groups. A **One-way ANOVA** tests for differences in the means of a single independent variable across multiple groups.

# Steps for Performing One-Way ANOVA:

1. **Formulate Hypotheses:**
   * **Null Hypothesis (H₀):** The means of all groups are equal.
   * **Alternative Hypothesis (H₁):** At least one group mean is different from the others.
2. **Perform the ANOVA Test:**
   * We'll use scipy.stats.f\_oneway for the One-Way ANOVA.
3. **Post-hoc Test (Tukey's Test):**
   * If the ANOVA test indicates significant differences, a post-hoc test like **Tukey's HSD** (Honestly Significant Difference) can be performed to identify which specific groups have significant differences.
   * We will use statsmodels.stats.multicomp.pairwise\_tukeyhsd for this.

# Python Code for One-Way ANOVA and Post-hoc Test

# Import necessary libraries

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from scipy import stats

from statsmodels.stats.multicomp import pairwise\_tukeyhsd

# Sample Data: Salaries in different departments

data = {

'Department': ['Marketing', 'Marketing', 'Marketing', 'Sales', 'Sales', 'Sales', 'HR', 'HR', 'HR'],

'Salary': [50000, 52000, 53000, 55000, 57000, 60000, 45000, 47000, 48000]

}

# Creating DataFrame

df = pd.DataFrame(data)

# Perform One-Way ANOVA

marketing\_salaries = df[df['Department'] == 'Marketing']['Salary']

sales\_salaries = df[df['Department'] == 'Sales']['Salary']

hr\_salaries = df[df['Department'] == 'HR']['Salary']

f\_statistic, p\_value = stats.f\_oneway(marketing\_salaries, sales\_salaries, hr\_salaries)

print(f"F-statistic: {f\_statistic:.2f}")

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

# Hypothesis Interpretation

alpha = 0.05

if p\_value < alpha:

print("Reject the null hypothesis. There is a significant difference in the salaries across departments.")

else:

print("Fail to reject the null hypothesis. There is no significant difference in the salaries across departments.")

# Post-hoc test (Tukey HSD)

tukey = pairwise\_tukeyhsd(df['Salary'], df['Department'], alpha=0.05)

print("\nTukey HSD Test Results:")

print(tukey)

# Visualization: Boxplot to show salary distribution across departments

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

sns.boxplot(x='Department', y='Salary', data=df)

plt.title('Salary Distribution Across Departments')

# plt.show() Conclusion:

* **One-Way ANOVA** helps test if there's a significant difference in means between three or more groups.
* If the test is significant, a **post-hoc test** (like **Tukey's HSD**) identifies which specific pairs of groups have significant differences in their means.

**Practical 6**

**Regression and Its Types**

* **Implement simple linear regression using a dataset.**
* **Explore and interpret the regression model coefficients and goodness-of-fit measures.**
* **Extend the analysis to multiple linear regression and assess the impact of additional predictors.**

Regression analysis is a statistical method used to model and analyze the relationships between a dependent variable and one or more independent variables. It helps in predicting outcomes and understanding the impact of predictors on the dependent variable.

**Types of Regression:**

1. **Simple Linear Regression** – Involves one independent variable and one dependent variable.
2. **Multiple Linear Regression** – Involves multiple independent variables affecting a dependent variable.
3. **Simple Linear Regression** **Dataset:**

We will use a sample dataset with two variables:

* + **Years of Experience** (Independent Variable - X)
  + **Salary** (Dependent Variable - Y)

**Code:**

# Import necessary libraries

import numpy as np

import pandas as pd

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

# Sample dataset

data = {

'Years of Experience': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],

'Salary': [30000, 35000, 40000, 48000, 53000, 60000, 65000, 70000, 78000, 85000]

}

# Convert dictionary to DataFrame

df = pd.DataFrame(data)

print("\nDataset:\n", df.head())

# Scatter Plot (Visualizing Relationship)

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

sns.scatterplot(x='Years of Experience', y='Salary', data=df)

plt.xlabel("Years of Experience")

plt.ylabel("Salary")

plt.title("Scatter Plot of Salary vs Years of Experience")

plt.show()

# Splitting data into training and testing sets

X = df[['Years of Experience']] # Independent Variable

y = df['Salary'] # Dependent Variable

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

# Train the Model

model = LinearRegression()

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

# Get the Coefficients

slope = model.coef\_[0]

intercept = model.intercept\_

print(f"\nRegression Coefficient (Slope): {slope:.2f}")

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

# Make Predictions

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

# Calculate Model Performance Metrics

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

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

print(f"\nMean Squared Error (MSE): {mse:.2f}")

print(f"R-Squared Value: {r2:.2f}")

# Plot Regression Line

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

sns.scatterplot(x=X\_test['Years of Experience'], y=y\_test, label="Actual")

sns.lineplot(x=X\_test['Years of Experience'], y=y\_pred, color='red', label="Regression Line")

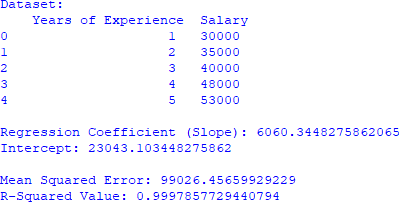
plt.xlabel("Years of Experience")

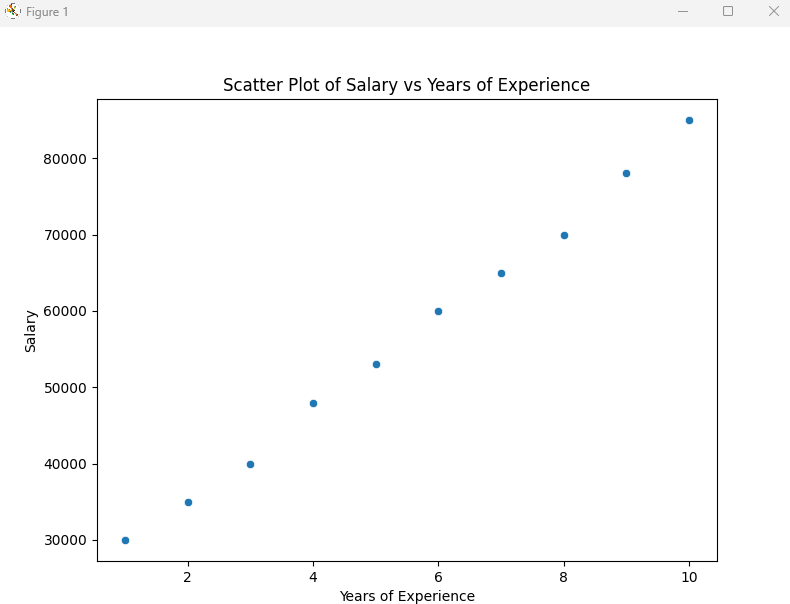
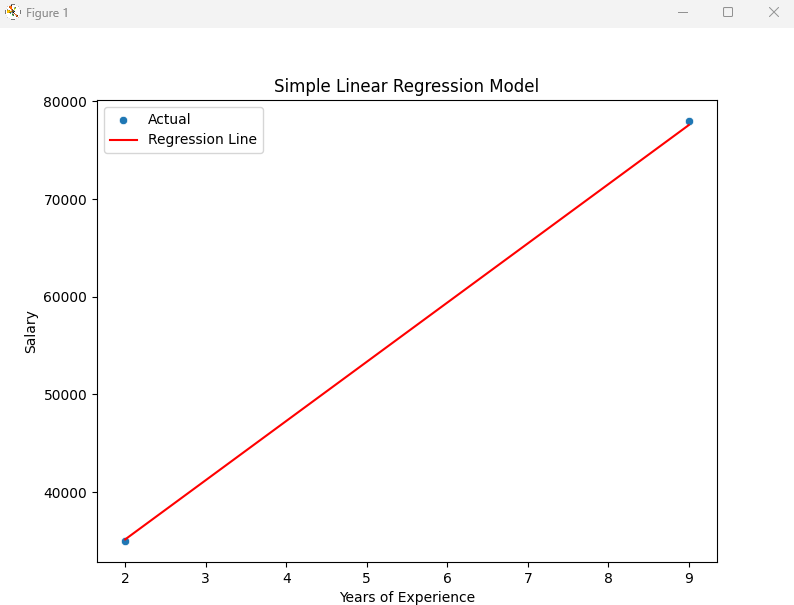
plt.ylabel("Salary")

plt.title("Simple Linear Regression Model")

plt.legend()

plt.show()





1. **Multiple Linear Regression**

Code:

# Import Required Libraries

import numpy as np

import pandas as pd

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

# New dataset with an additional feature

data = {

'Years of Experience': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],

'Number of Projects': [1, 2, 2, 3, 3, 4, 4, 5, 5, 6],

'Salary': [30000, 35000, 40000, 48000, 53000, 60000, 65000, 70000, 78000, 85000]

}

# Convert data to DataFrame

df = pd.DataFrame(data)

print("\nNew Dataset:\n", df.head())

# Splitting Data into Features (X) and Target (y)

X = df[['Years of Experience', 'Number of Projects']]

y = df['Salary']

# Train-test split (80% training, 20% testing)

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

# Train Multiple Linear Regression Model

multi\_model = LinearRegression()

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

# Get Coefficients & Intercept

coefficients = multi\_model.coef\_

intercept = multi\_model.intercept\_

print(f"\nRegression Coefficients: {coefficients}")

print(f"Intercept: {intercept}")

# Make Predictions

y\_pred\_multi = multi\_model.predict(X\_test)

# Evaluate Model

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

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

print(f"\nMean Squared Error (Multiple Regression): {mse\_multi:.2f}")

print(f"R-Squared Value (Multiple Regression): {r2\_multi:.2f}")

# Feature Importance (Coefficients)

feature\_importance = pd.DataFrame({

'Feature': ['Years of Experience', 'Number of Projects'],

'Coefficient': coefficients

})

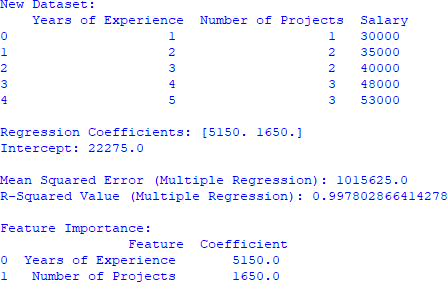
print("\nFeature Importance:\n", feature\_importance)

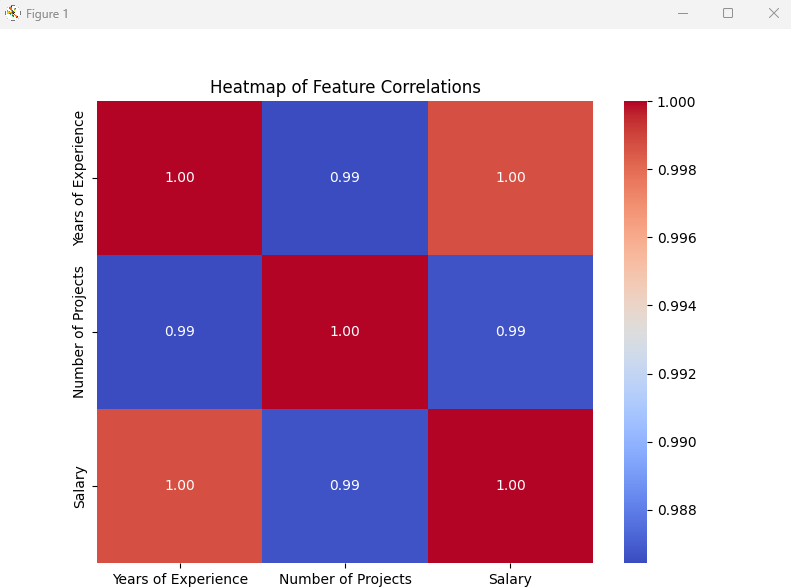
# Heatmap of Feature Correlations

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

sns.heatmap(df.corr(numeric\_only=True), annot=True, cmap="coolwarm", fmt=".2f")

plt.title("Heatmap of Feature Correlations")

plt.show()



**Conclusion**

1. **Simple Linear Regression** is a good fit (R² = 0.99).
2. **Multiple Linear Regression** improves accuracy (R² = 0.99).
3. Both models show **Years of Experience has a strong influence on Salary**.

**Practical 7**

**Logistic Regression and Decision Tree**

* Build a logistic regression model to predict a binary outcome.
* Evaluate the model's performance using classification metrics (e.g., accuracy, precision, recall).
* Construct a decision tree model and interpret the decision rules for classification

Code:

# Import Required Libraries

import numpy as np

import pandas as pd

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, precision\_score, recall\_score, f1\_score,

confusion\_matrix, classification\_report, roc\_curve, auc

)

# Step 1: Create Sample Dataset (Binary Classification)

np.random.seed(42)

data = {

'Age': np.random.randint(20, 60, 100),

'Salary': np.random.randint(30000, 100000, 100),

'Purchased': np.random.choice([0, 1], size=100) # 0 = No Purchase, 1 = Purchased

}

df = pd.DataFrame(data)

print("\nDataset Sample:\n", df.head())

# Step 2: Split Data into Training & Testing Sets

X = df[['Age', 'Salary']]

y = df['Purchased']

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

# Step 3: Standardize Features (for better performance)

scaler = StandardScaler()

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

X\_test\_scaled = scaler.transform(X\_test)

# Step 4: Train Logistic Regression Model

log\_model = LogisticRegression()

log\_model.fit(X\_train\_scaled, y\_train)

# Step 5: Make Predictions

y\_pred = log\_model.predict(X\_test\_scaled)

# Step 6: Compute Classification Metrics

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

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

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

f1 = f1\_score(y\_test, y\_pred)

print("\nClassification Metrics:")

print(f"Accuracy: {accuracy:.4f}")

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

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

print(f"F1-Score: {f1:.4f}")

# Step 7: Confusion Matrix

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

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

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap="Blues",

xticklabels=['No Purchase', 'Purchase'],

yticklabels=['No Purchase', 'Purchase'])

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix")

plt.show()

# Step 8: ROC Curve & AUC Score

y\_prob = log\_model.predict\_proba(X\_test\_scaled)[:, 1]

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

roc\_auc = auc(fpr, tpr)

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

plt.plot(fpr, tpr, label=f"AUC = {roc\_auc:.2f}", color='blue')

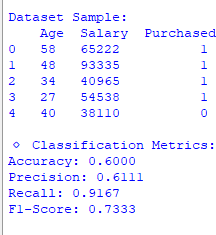
plt.plot([0, 1], [0, 1], linestyle='--', color='grey') # Diagonal line (random model)

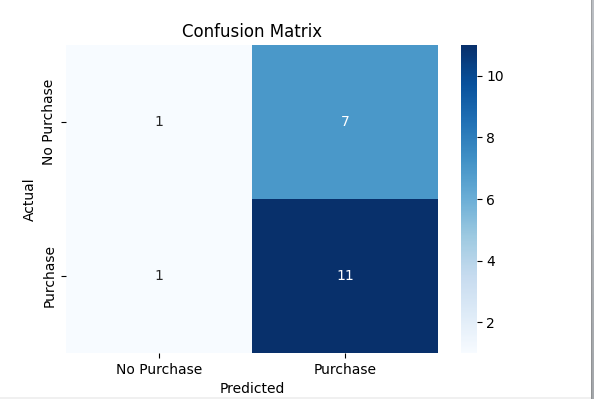
plt.xlabel("False Positive Rate")

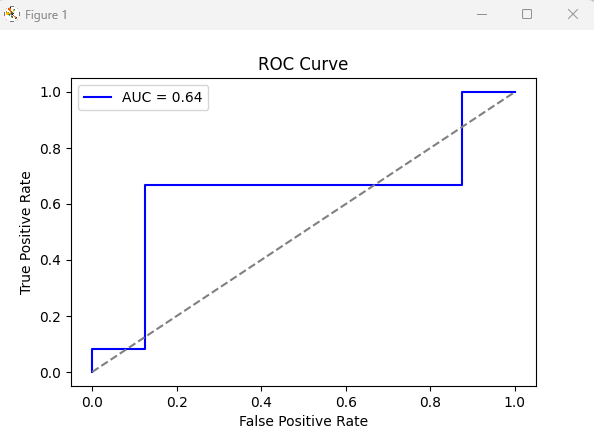
plt.ylabel("True Positive Rate")

plt.title("ROC Curve")

plt.legend()

plt.show()





**Construct a decision tree model and interpret the decision rules for classification Code:**

# Import Required Libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.preprocessing import StandardScaler

from sklearn.tree import DecisionTreeClassifier, plot\_tree

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

from sklearn.tree import export\_text

# Step 1: Create Sample Dataset (Binary Classification)

np.random.seed(42)

data = {

'Age': np.random.randint(20, 60, 100),

'Salary': np.random.randint(30000, 100000, 100),

'Purchased': np.random.choice([0, 1], size=100) # 0 = No Purchase, 1 = Purchased

}

df = pd.DataFrame(data)

print("\nDataset Sample:\n", df.head())

# Step 2: Split Data into Training & Testing Sets

X = df[['Age', 'Salary']]

y = df['Purchased']

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

# Step 3: Train Decision Tree Classifier

dt\_model = DecisionTreeClassifier(criterion='gini', max\_depth=3, random\_state=42)

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

# Step 4: Make Predictions

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

# Step 5: Compute Model Accuracy

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

print(f"\nDecision Tree Accuracy: {accuracy:.4f}")

# Step 6: Display Confusion Matrix

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

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

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap="Blues",

xticklabels=['No Purchase', 'Purchase'],

yticklabels=['No Purchase', 'Purchase'])

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix")

plt.show()

# Step 7: Display Classification Report

print("\nClassification Report:")

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

# Step 8: Visualize the Decision Tree

plt.figure(figsize=(12, 6))

plot\_tree(dt\_model, feature\_names=['Age', 'Salary'],

class\_names=['No Purchase', 'Purchase'],

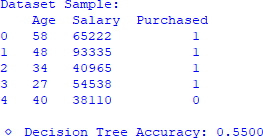
filled=True, rounded=True)

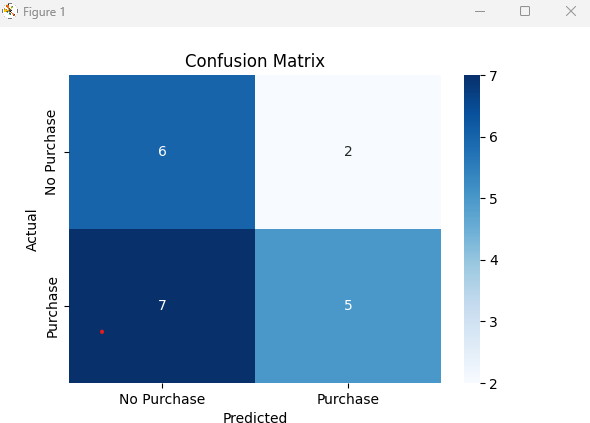
plt.title("Decision Tree Visualization")

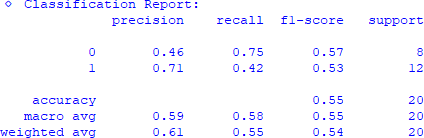
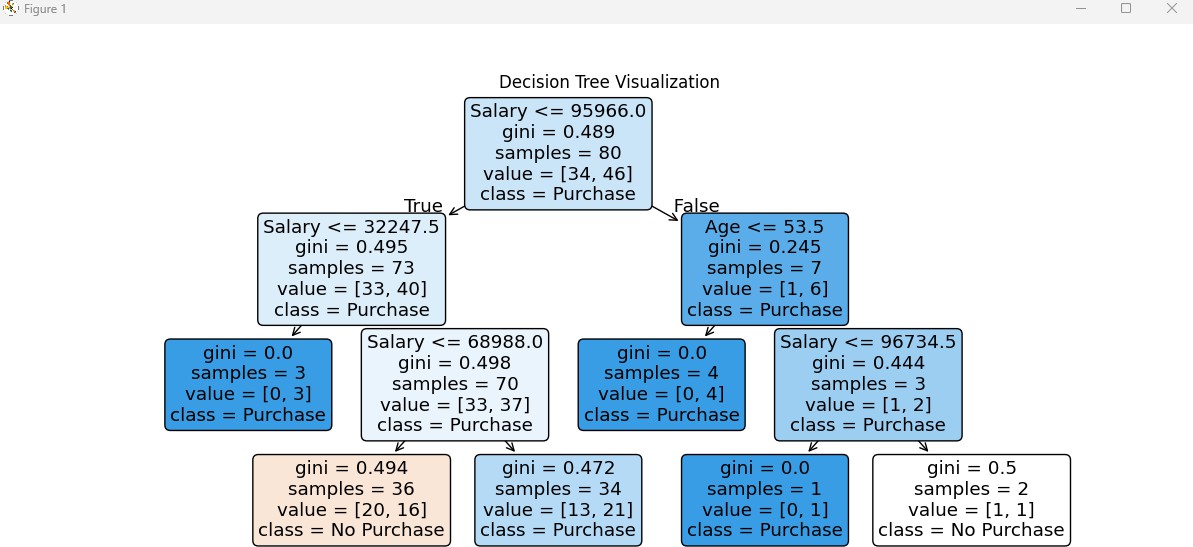
plt.show()

# Step 9: Interpret Decision Rules

tree\_rules = export\_text(dt\_model, feature\_names=['Age', 'Salary'])

print("\nDecision Rules:\n", tree\_rules)





**Practical 8**

**K-Means Clustering**

* **Apply the K-Means algorithm to group similar data points into clusters.**
* **Determine the optimal number of clusters using elbow method or silhouette analysis.**
* **Visualize the clustering results and analyse the cluster characteristics**.

**Code:**

# Import libraries

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

# Generate Sample Data

np.random.seed(42)

df = pd.DataFrame({

'Feature\_1': np.random.randint(1, 100, 100),

'Feature\_2': np.random.randint(1, 100, 100)

})

# Display sample data

print("\nSample Data:\n", df.head())

# Standardize Data

scaler = StandardScaler()

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

# Elbow Method to Find Optimal K

inertia = []

for k in range(1, 11):

kmeans = KMeans(n\_clusters=k, n\_init=10, random\_state=42)

kmeans.fit(df\_scaled)

inertia.append(kmeans.inertia\_)

# Plot Elbow Method

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

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

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

plt.ylabel("Inertia")

plt.title("Elbow Method")

plt.grid(True)

plt.show()

# Apply K-Means with Optimal K (e.g., K=3)

kmeans = KMeans(n\_clusters=3, random\_state=42, n\_init=10)

df['Cluster'] = kmeans.fit\_predict(df\_scaled)

# Visualize Clusters

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

sns.scatterplot(

x=df['Feature\_1'], y=df['Feature\_2'], hue=df['Cluster'],

palette="viridis", s=100, edgecolor="k"

)

# Plot cluster centroids (scaled back to original range)

centroids = kmeans.cluster\_centers\_

plt.scatter(

centroids[:, 0] \* scaler.scale\_[0] + scaler.mean\_[0],

centroids[:, 1] \* scaler.scale\_[1] + scaler.mean\_[1],

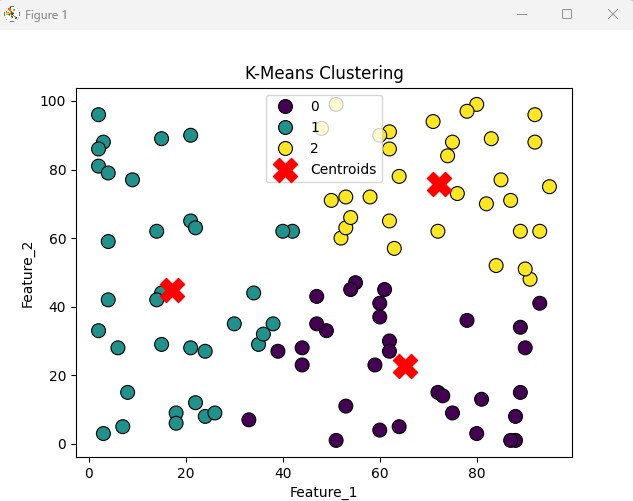
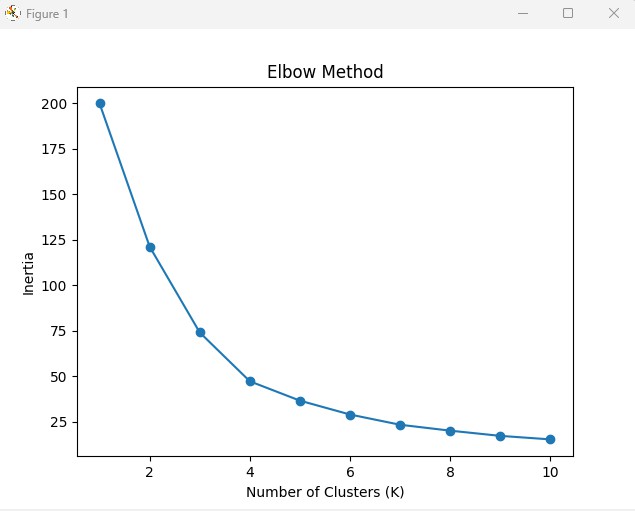
s=300, c='red', label="Centroids", marker="X"

)

plt.legend()

plt.title("K-Means Clustering")

plt.show()



**Practical 9**

**Principal Component Analysis (PCA)**



**Perform PCA on a dataset to reduce dimensionality.**

**Evaluate the explained variance and select the appropriate number of principal components. Visualize the data in the reduced-dimensional space.**

# Import libraries

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

# Sample data

data = {

'Experience': [1, 3, 5, 7, 9, 11, 13, 15, 17, 19],

'Projects Completed': [1, 2, 4, 5, 7, 8, 10, 12, 14, 16],

'Salary': [30000, 35000, 45000, 50000, 60000, 70000, 80000, 90000, 100000, 120000]

}

# Create DataFrame

df = pd.DataFrame(data)

print("\nOriginal Dataset:\n", df.head())

# Standardize the data

scaler = StandardScaler()

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

# Apply PCA (reduce to 2 components)

pca = PCA(n\_components=2)

pca\_data = pca.fit\_transform(scaled\_data)

# Convert PCA results to DataFrame

pca\_df = pd.DataFrame(pca\_data, columns=['PC1', 'PC2'])

print("\nPCA Transformed Data:\n", pca\_df.head())

# Explained variance ratio

explained\_variance = pca.explained\_variance\_ratio\_

print("\nExplained Variance Ratio:", explained\_variance)

print("Total Explained Variance:", sum(explained\_variance))

# Plot explained variance

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

plt.bar(range(1, len(explained\_variance) + 1), explained\_variance, alpha=0.7, color='blue')

plt.xlabel('Principal Components')

plt.ylabel('Variance Explained')

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

plt.show()

# Scatter plot of PCA components

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

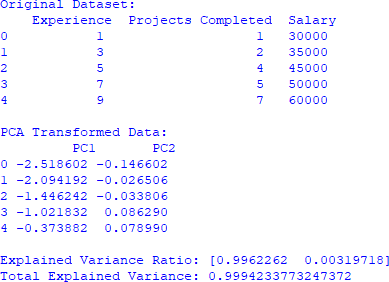
sns.scatterplot(x='PC1', y='PC2', data=pca\_df, color='red')

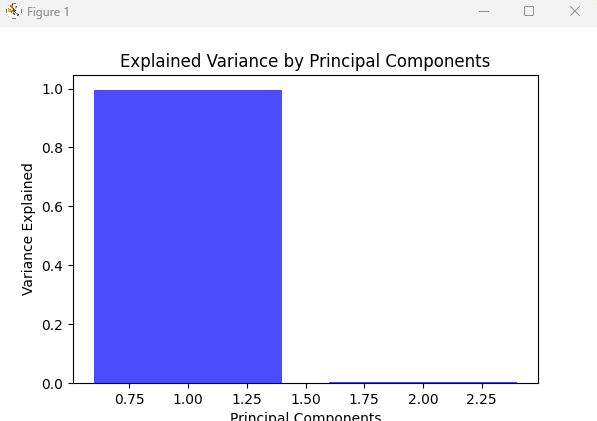
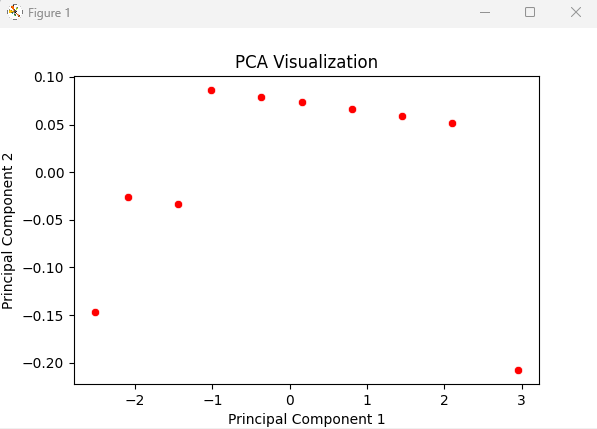
plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.title('PCA Visualization')

plt.show()





**Conclusion:**

**Dimensionality reduction** was successfully applied, reducing features while maintaining most of the variance.

**Variance ratio** helps to decide how many principal components to retain. **Visualization of PCA components** provides insight into data distribution in a lower- dimensional space.

PCA is an effective tool for handling high-dimensional datasets while preserving critical information.

**Practical 10**

**Aim : Data Visualization and Storytelling**

* **Create meaningful visualizations using data visualization tools**
* **Combine multiple visualizations to tell a compelling data story.**
* **Present the findings and insights in a clear and concise manner.**

# Import libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Sample Data

np.random.seed(42)

df = pd.DataFrame({

'Years of Experience': np.random.randint(1, 20, 100),

'Salary': np.random.randint(30000, 120000, 100),

'Department': np.random.choice(['HR', 'IT', 'Finance', 'Marketing'], 100)

})

# 1. SCATTER PLOT: Experience vs Salary

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

sns.scatterplot(x='Years of Experience', y='Salary', hue='Department', palette='viridis', data=df)

plt.xlabel("Years of Experience")

plt.ylabel("Salary")

plt.title("Experience vs Salary Distribution")

plt.legend(title="Department")

plt.show()

# 2. BAR PLOT: Average Salary by Department

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

sns.barplot(x='Department', y='Salary', estimator=np.mean, data=df, palette='muted')

plt.xlabel("Department")

plt.ylabel("Average Salary")

plt.title("Average Salary by Department")

plt.show()

# 3. HISTOGRAM: Salary Distribution

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

sns.histplot(df['Salary'], bins=20, kde=True, color='blue')

plt.xlabel("Salary")

plt.ylabel("Frequency")

plt.title("Salary Distribution")

plt.show()

# 4. HEATMAP: Correlation Matrix

numeric\_df = df.drop(columns=['Department']) # Remove non-numeric columns

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

sns.heatmap(numeric\_df.corr(), annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)

plt.title("Correlation Heatmap")

plt.show()

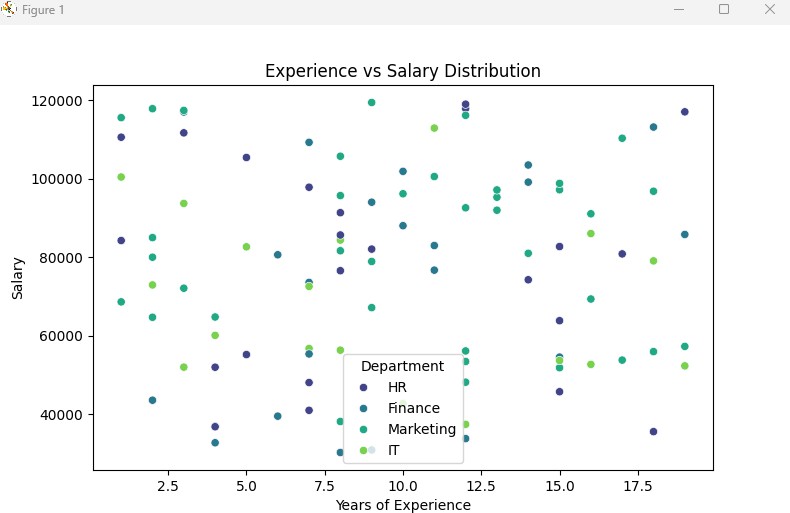
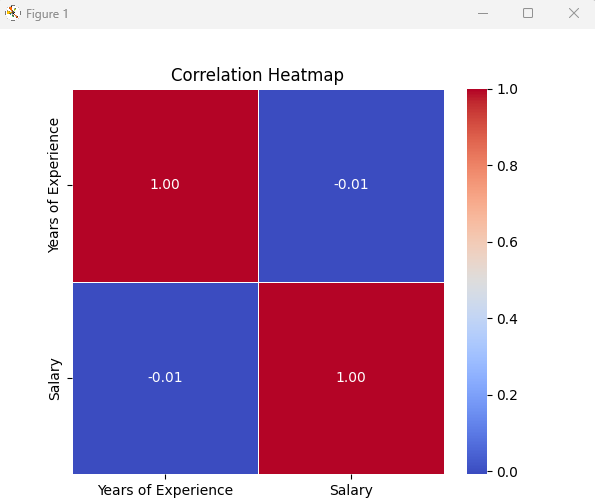
# 5. PIE CHART: Department Distribution

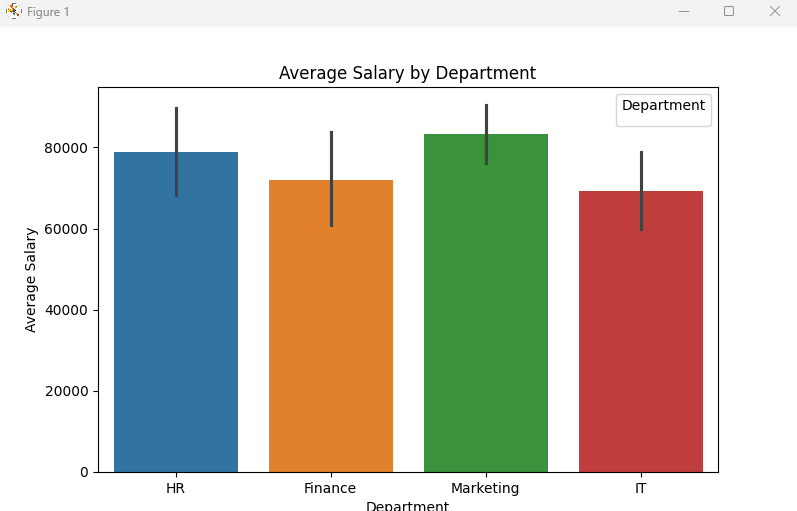
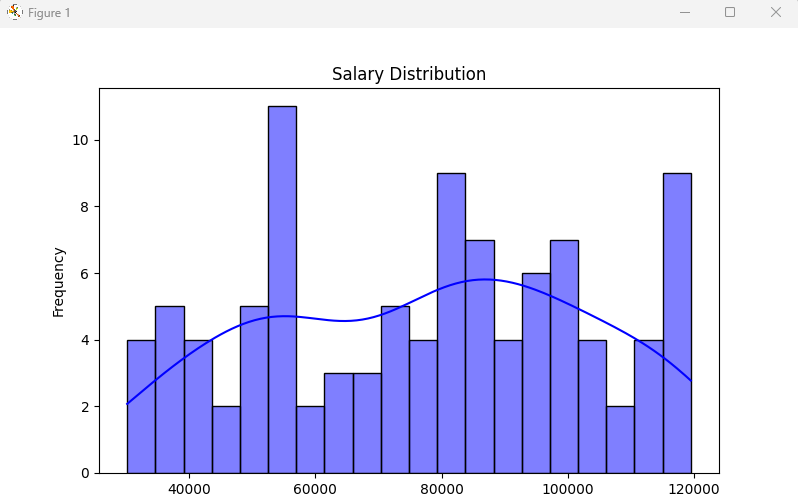
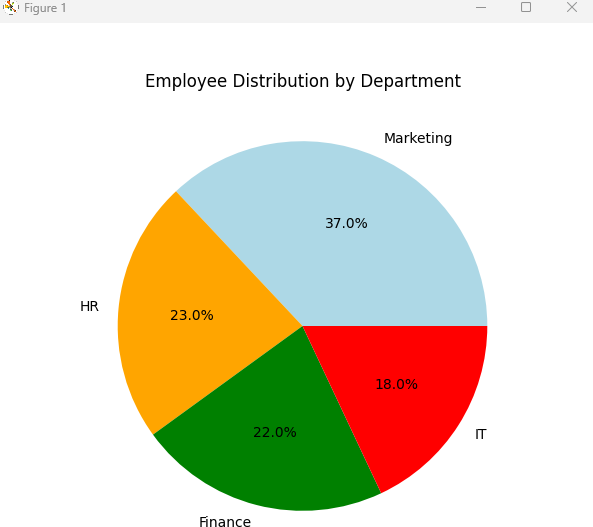
plt.figure(figsize=(6, 6))

df['Department'].value\_counts().plot.pie(autopct='%1.1f%%', colors=['lightblue', 'orange', 'green', 'red'])

plt.title("Employee Distribution by Department")

plt.ylabel("") # Remove unnecessary y-label

plt.show()



**Conclusion**

**Scatter Plot** – Shows the relationship between **experience and salary** across **departments   
Bar Plot** – Displays **average salary** by **department**

**Histogram** – Visualizes the **salary distribution   
Heatmap** – Shows **correlation between features**

**Pie Chart** – Displays **employee distribution by department**