T.Y.B.Sc. COMPUTER SCIENCE

SEMESTER - VI

Lab Manual

Subject Code: USCSP601

Subject Name: Data Science

(NEW SYLLABUS W.E.F. 2023-2024)

Course Writer:

Prof. Hasan Phudinawala

INDEX

Sr No.	Date	Practical	Signature
1	23/01/25	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.	
2	30/01/25	Feature Scaling and Dummification • Apply feature-scaling techniques like standardization and normalization to numerical features. • Perform feature dummification to convert categorical variables into numerical representations.	
3	30/01/25	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.	
4	06/02/25	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.	

5	06/02/25	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 analyze the cluster characteristics.	
6	13/02/25	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.	
7	20/02/25	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.	
8	20/02/25	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.	

Practical no : 01 : Data Frames and Basic Data Pre-processing

Aim: 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.

In [1]:
```
import pandas as pd
import numpy as np

# Reading a CSV file into a DataFrame
df=pd.read_csv(r"C:\Users\DELL\Desktop\MSC Data science\Data sets\Iris.csv")

print(df.head())  # Display the first 5 rows of the DataFrame
```

```
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa
```

In [2]:
```
# Step 2: Basic Data Exploration
df.head()
```

Out[2]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

In [3]:
```
df.tail()
```

Out[3]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

In [4]:

```
df.describe()
```

Out[4]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             150 non-null    int64  
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object 
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

In [6]:

```
df.shape
```

Out[6]: (150, 6)

In [7]:

```
df.dtypes
```

Out[7]:
```
Id                 int64
SepalLengthCm    float64
SepalWidthCm     float64
PetalLengthCm    float64
PetalWidthCm     float64
Species           object
dtype: object
```

In [8]:
```
df.columns
```

Out[8]:
```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

In [9]:
```
# Step 3: Checking for Missing Values
# Checking for missing values in each column of the CSV DataFrame
missing_values = df.isnull().sum()
print("\nMissing Values in CSV Data:")
print(missing_values)
```

```
Missing Values in CSV Data:
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

In [10]:
```
df = df.drop(columns=['Species'])
```

In [11]:
```
# Step 4: Handling Missing Values
# We will fill missing values in columns with the mean of the column
# (You could also drop missing rows or use other strategies depending on your need
fill= df.fillna(df.mean())
print("\nFilled Missing Values with Mean (CSV Data):")
print(df.head())
```

```
Filled Missing Values with Mean (CSV Data):
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
0   1            5.1           3.5            1.4           0.2
1   2            4.9           3.0            1.4           0.2
2   3            4.7           3.2            1.3           0.2
3   4            4.6           3.1            1.5           0.2
4   5            5.0           3.6            1.4           0.2
```

In [12]:
```
# Alternatively, you can drop rows with missing values:
# df_csv_dropped = df_csv.dropna()
# print("\nDropped Rows with Missing Values (CSV Data):")
# print(df_csv_dropped.head())
```

In [13]:
```
# Step 5: Handling Outliers
# Here we will calculate Z-scores and remove rows where Z-score is greater than 3
z_scores = np.abs((fill - fill.mean()) / fill.std())
z_scores
```

Out[13]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	1.714797	0.897674	1.028611	1.336794	1.308593
1	1.691780	1.139200	0.124540	1.336794	1.308593
2	1.668762	1.380727	0.336720	1.393470	1.308593
3	1.645745	1.501490	0.106090	1.280118	1.308593
4	1.622728	1.018437	1.259242	1.336794	1.308593
...	...	...	...	...	...
145	1.622728	1.034539	0.124540	0.816888	1.443121
146	1.645745	0.551486	1.277692	0.703536	0.918985
147	1.668762	0.793012	0.124540	0.816888	1.050019
148	1.691780	0.430722	0.797981	0.930239	1.443121
149	1.714797	0.068433	0.124540	0.760212	0.787951

150 rows × 5 columns

In [14]:

```
# Remove rows where any Z-score is greater than 3 (outliers)
do = fill[(z_scores < 3).all(axis=1)]
print("\nData After Removing Outliers (CSV Data):")
print(do.head())
```

```
Data After Removing Outliers (CSV Data):
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
0   1            5.1           3.5            1.4           0.2
1   2            4.9           3.0            1.4           0.2
2   3            4.7           3.2            1.3           0.2
3   4            4.6           3.1            1.5           0.2
4   5            5.0           3.6            1.4           0.2
```

In [15]:

```
# Step 6: Filtering Data (Example: Select rows where a column value is greater tha
threshold_value = 3  # Example threshold value
filter = do[do['SepalLengthCm'] > threshold_value]
print(f"\nFiltered Data (Rows with column_name > {threshold_value}):")
print(filter)
```

```
Filtered Data (Rows with column_name > 3):
      Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
0      1            5.1           3.5            1.4           0.2
1      2            4.9           3.0            1.4           0.2
2      3            4.7           3.2            1.3           0.2
3      4            4.6           3.1            1.5           0.2
4      5            5.0           3.6            1.4           0.2
..   ...            ...           ...            ...           ...
145  146            6.7           3.0            5.2           2.3
146  147            6.3           2.5            5.0           1.9
147  148            6.5           3.0            5.2           2.0
148  149            6.2           3.4            5.4           2.3
149  150            5.9           3.0            5.1           1.8

[149 rows x 5 columns]
```

In [16]:
```
# Step 7: Sorting Data (Sorting by a column in descending order)
df_sorted = filter.sort_values(by='SepalWidthCm', ascending=False)
print(df_sorted.head())
```

```
    Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
33  34            5.5           4.2            1.4           0.2
32  33            5.2           4.1            1.5           0.1
14  15            5.8           4.0            1.2           0.2
16  17            5.4           3.9            1.3           0.4
5    6            5.4           3.9            1.7           0.4
```

In [17]:
```
# Step 8: Grouping Data (Example: Group by a column and calculate the mean of anot
df_grouped = df_sorted.groupby('SepalLengthCm').agg({
    'PetalLengthCm': 'mean',  # Calculate the mean of 'another_column' for each gr
    'PetalWidthCm': 'sum'   # Calculate the sum of 'yet_another_column' for each gr
}).reset_index()  # Reset index to avoid multi-index
print("\nGrouped Data (Mean and Sum for Each Group):")
print(df_grouped.head())
```

```
Grouped Data (Mean and Sum for Each Group):
   SepalLengthCm  PetalLengthCm  PetalWidthCm
0            4.3       1.100000           0.1
1            4.4       1.333333           0.6
2            4.5       1.300000           0.3
3            4.6       1.325000           0.9
4            4.7       1.450000           0.4
```

Practical No 02: Feature Scaling and Dummification

Part I: Apply feature-scaling techniques like standardization and normalization to numerical features

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler, StandardScaler
```

In [2]:
```
df = pd.read_csv(r"C:\Users\DELL\Desktop\wine.csv")
df
```

Out[2]:

	Wine	Alcohol	Malic.acid	Ash	Acl	Mg	Phenols	Flavanoids	Nonflavanoid.pheno
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.2
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.2
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.3
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.2
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.3
...	...	...	...	...	...	...	...	...	
173	3	13.71	5.65	2.45	20.5	95	1.68	0.61	0.5
174	3	13.40	3.91	2.48	23.0	102	1.80	0.75	0.4
175	3	13.27	4.28	2.26	20.0	120	1.59	0.69	0.4
176	3	13.17	2.59	2.37	20.0	120	1.65	0.68	0.5
177	3	14.13	4.10	2.74	24.5	96	2.05	0.76	0.5

178 rows × 14 columns

In [3]:
```
df1 = pd.read_csv(r"C:\Users\DELL\Desktop\wine.csv", usecols=[0, 1, 2], skiprows=1
df1.columns = ['classlabel', 'Alcohol', 'Malic Acid']
print("Original DataFrame:")
df1
```

```
Original DataFrame:
```

Out[3]:

	classlabel	Alcohol	Malic Acid
0	1	13.20	1.78
1	1	13.16	2.36
2	1	14.37	1.95
3	1	13.24	2.59
4	1	14.20	1.76
...	...	...	...
172	3	13.71	5.65
173	3	13.40	3.91
174	3	13.27	4.28
175	3	13.17	2.59
176	3	14.13	4.10

177 rows × 3 columns

MinMax Scaler

There is another way of data scaling, where the minimum of feature is made equal to zero and the maximum of feature equal to one. MinMax Scaler shrinks the data within the given range, usually of 0 to 1. It transforms data by scaling features to a given range. It scales the values to a specific value range without changing the shape of the original distribution.

In [ ]:
```
scaling=MinMaxScaler()
scaled_value=scaling.fit_transform(df1[['Alcohol','Malic Acid']])
df1[['Alcohol','Malic Acid']]=scaled_value
print("\n Dataframe after MinMax Scaling")
df1
```

StandardScaler

StandardScaler is a preprocessing technique in scikit-learn used for standardizing features by removing the mean and scaling to unit variance. StandardScaler, a popular preprocessing technique provided by scikit-learn, offers a simple yet effective method for standardizing feature values. StandardScaler operates on the principle of normalization, where it transforms the distribution of each feature to have a mean of zero and a standard deviation of one. This process ensures that all features are on the same scale, preventing any single feature from dominating the learning process due to its larger magnitude.

In [4]:
```
scaling=StandardScaler()
scaled_standardvalue=scaling.fit_transform(df1[['Alcohol','Malic Acid']])
df1[['Alcohol','Malic Acid']]=scaled_standardvalue
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
print("\n Dataframe after Standard Scaling")
df1
```

```
Dataframe after Standard Scaling
```

Out[4]:

	classlabel	Alcohol	Malic Acid
0	1	0.255824	-0.501624
1	1	0.206229	0.018020
2	1	1.706501	-0.349315
3	1	0.305420	0.224086
4	1	1.495719	-0.519543
...	...	...	...
172	3	0.888171	2.965658
173	3	0.503803	1.406725
174	3	0.342617	1.738222
175	3	0.218628	0.224086
176	3	1.408926	1.576953

177 rows × 3 columns

Part II : Perform feature Dummification to convert categorical variables into numerical representations.

In [5]:

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
```

In [6]:

```
iris=pd.read_csv(r"C:\Users\DELL\Desktop\MSC Data science\Data sets\Iris.csv")
iris
```

Out[6]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

In [7]:

```
le=LabelEncoder()
iris['code']=le.fit_transform(iris.Species)
iris
```

Out[7]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species	code
0	1	5.1	3.5	1.4	0.2	Iris-setosa	0
1	2	4.9	3.0	1.4	0.2	Iris-setosa	0
2	3	4.7	3.2	1.3	0.2	Iris-setosa	0
3	4	4.6	3.1	1.5	0.2	Iris-setosa	0
4	5	5.0	3.6	1.4	0.2	Iris-setosa	0
...	...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica	2
146	147	6.3	2.5	5.0	1.9	Iris-virginica	2
147	148	6.5	3.0	5.2	2.0	Iris-virginica	2
148	149	6.2	3.4	5.4	2.3	Iris-virginica	2
149	150	5.9	3.0	5.1	1.8	Iris-virginica	2

150 rows × 7 columns

Pratical No: 03 - Regression and Its Types

Aim : To Implement simple linear regression using a dataset.Explore and interpret the regression model coefficients and goodness-of-fit measures.

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,r2_score
```

In [2]:
```
df = pd.read_csv(r"C:\Users\DELL\Downloads\fetch_california_housing.csv")
df.head()
```

Out[2]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitud
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.2
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.2
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.2
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.2
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.2

In [3]:
```
df.tail()
```

Out[3]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Lon
20635	1.5603	25.0	5.045455	1.133333	845.0	2.560606	39.48	-
20636	2.5568	18.0	6.114035	1.315789	356.0	3.122807	39.49	-
20637	1.7000	17.0	5.205543	1.120092	1007.0	2.325635	39.43	-
20638	1.8672	18.0	5.329513	1.171920	741.0	2.123209	39.43	-
20639	2.3886	16.0	5.254717	1.162264	1387.0	2.616981	39.37	-

In [4]:
```
df.shape
```

Out[4]: (20640, 9)

In [5]:
```
df.size
```

Out[5]: 185760

In [6]:
```
df.describe()
```

Out[6]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccu
count	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.00000
mean	3.870671	28.639486	5.429000	1.096675	1425.476744	3.07065
std	1.899822	12.585558	2.474173	0.473911	1132.462122	10.38605
min	0.499900	1.000000	0.846154	0.333333	3.000000	0.69230
25%	2.563400	18.000000	4.440716	1.006079	787.000000	2.42974
50%	3.534800	29.000000	5.229129	1.048780	1166.000000	2.81811
75%	4.743250	37.000000	6.052381	1.099526	1725.000000	3.28226
max	15.000100	52.000000	141.909091	34.066667	35682.000000	1243.33333

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   MedInc       20640 non-null  float64
 1   HouseAge     20640 non-null  float64
 2   AveRooms     20640 non-null  float64
 3   AveBedrms    20640 non-null  float64
 4   Population   20640 non-null  float64
 5   AveOccup     20640 non-null  float64
 6   Latitude     20640 non-null  float64
 7   Longitude    20640 non-null  float64
 8   MedHouseVal  20640 non-null  float64
dtypes: float64(9)
memory usage: 1.4 MB
```

In [8]:
```
df.dtypes
```

Out[8]:
```
MedInc         float64
HouseAge       float64
AveRooms       float64
AveBedrms      float64
Population     float64
AveOccup       float64
Latitude       float64
Longitude      float64
MedHouseVal    float64
dtype: object
```

In [9]:
```
#import ssl
#ssl._create_default_https_context = ssl._create_unverified_context

housing = fetch_california_housing()
```

```python
# Convert to DataFrame
housing_df = pd.DataFrame(housing.data, columns=housing.feature_names)
housing_df.head()  # Print first few rows
```

Out[9]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitud
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.2
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.2
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.2
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.2
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.2

In [10]:
```python
housing_df['PRICE']=housing.target
X=housing_df[['AveRooms']]
y=housing_df[['PRICE']]
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)
```

In [11]:
```python
model=LinearRegression()
model.fit(X_train,y_train)
```

Out[11]:

```
▾ LinearRegression
LinearRegression()
```

In [12]:
```python
mse=mean_squared_error(y_test,model.predict(X_test))
r2=r2_score(y_test,model.predict(X_test))
```

In [13]:
```python
print("Mean Squared Error: ", mse)
print("R-squared: ",r2)
print("Intercept: ",model.intercept_)
print("Co-efficient: ",model.coef_)
```

```
Mean Squared Error:  1.2923314440807299
R-squared:  0.013795337532284901
Intercept:  [1.65476227]
Co-efficient:  [[0.07675559]]
```

Part II: Extend the analysis to multiple linear regression and assess the impact of additional predictors.

In [14]:
```python
X = housing_df.drop('PRICE',axis=1)
y = housing_df['PRICE']
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=42
```

In [15]:
```python
model = LinearRegression()
model.fit(X_train,y_train)
y_pred = model.predict(X_test)
```

In [16]:
```
mse = mean_squared_error(y_test,y_pred)
r2 = r2_score(y_test,y_pred)
```

In [17]:
```
print("Mean Squared Error:",mse)
print("R-squared:",r2)
print("Intercept:",model.intercept_)
print("Coefficient:",model.coef_)
```

```
Mean Squared Error: 0.555891598695244
R-squared: 0.5757877060324511
Intercept: -37.023277706064064
Coefficient: [ 4.48674910e-01  9.72425752e-03 -1.23323343e-01  7.83144907e-01
 -2.02962058e-06 -3.52631849e-03 -4.19792487e-01 -4.33708065e-01]
```

Practical no:04

Aim: Logistic Regression and Decision Tree

Part I: Build a logistic regression model to predict a binary outcome. Evaluate the model's performance using classification metrics (e.g., accuracy, precision, recall).

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, classif
```

In [2]:
```
df=pd.read_csv(r"C:\Users\DELL\Desktop\MSC Data science\Data sets\Iris.csv")
df
```

Out[2]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

In [3]:
```
# Keep only two classes
df1 = df[df['Species'] != 2]
df1
```

Out[3]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

In [4]:

```
# Keep only two classes (filter out class 2)
df = df[df['Species'] != 2]

# Define features and target
X = df.drop('Species', axis=1)
y = df['Species']
```

In [5]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_st

logistic_model = LogisticRegression()
logistic_model.fit(X_train, y_train)
```

```
C:\Users\DELL\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\lin
ear_model\_logistic.py:460: ConvergenceWarning: lbfgs failed to converge (status=
1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[5]:

▾ LogisticRegression

LogisticRegression()

In [6]:

```
# Predictions
y_pred_logistic = logistic_model.predict(X_test)

# Evaluation
print("Accuracy:", accuracy_score(y_test, y_pred_logistic))
print("\nClassification Report")
print(classification_report(y_test, y_pred_logistic))
```

```
Accuracy: 1.0

Classification Report
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00         9
 Iris-virginica       1.00      1.00      1.00        11

       accuracy                           1.00        30
      macro avg       1.00      1.00      1.00        30
   weighted avg       1.00      1.00      1.00        30
```

Part II: Construct a decision tree model and interpret the decision rules for classification.

In [7]:

```
from sklearn.tree import DecisionTreeClassifier
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_st
```

In [8]:

```
model = DecisionTreeClassifier()

model.fit(X_train, y_train)
y_pred_tree = model.predict(X_test)
y_pred_tree
```

Out[8]:

```
array(['Iris-versicolor', 'Iris-setosa', 'Iris-virginica',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa',
       'Iris-versicolor', 'Iris-virginica', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-virginica', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-versicolor', 'Iris-virginica',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-virginica',
       'Iris-setosa', 'Iris-virginica', 'Iris-setosa', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-setosa', 'Iris-setosa'], dtype=object)
```

In [9]:

```
# Print Decision Tree Metrics
print("\nDecision Tree Metrics")
print("Accuracy: ", accuracy_score(y_test, y_pred_tree))
print("\nClassification Report")
print(classification_report(y_test, y_pred_tree))
```

```
Decision Tree Metrics
Accuracy:  1.0

Classification Report
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00         9
 Iris-virginica       1.00      1.00      1.00        11

       accuracy                           1.00        30
      macro avg       1.00      1.00      1.00        30
   weighted avg       1.00      1.00      1.00        30
```

In [ ]:

Practical No: 05 - K-Means Clustering

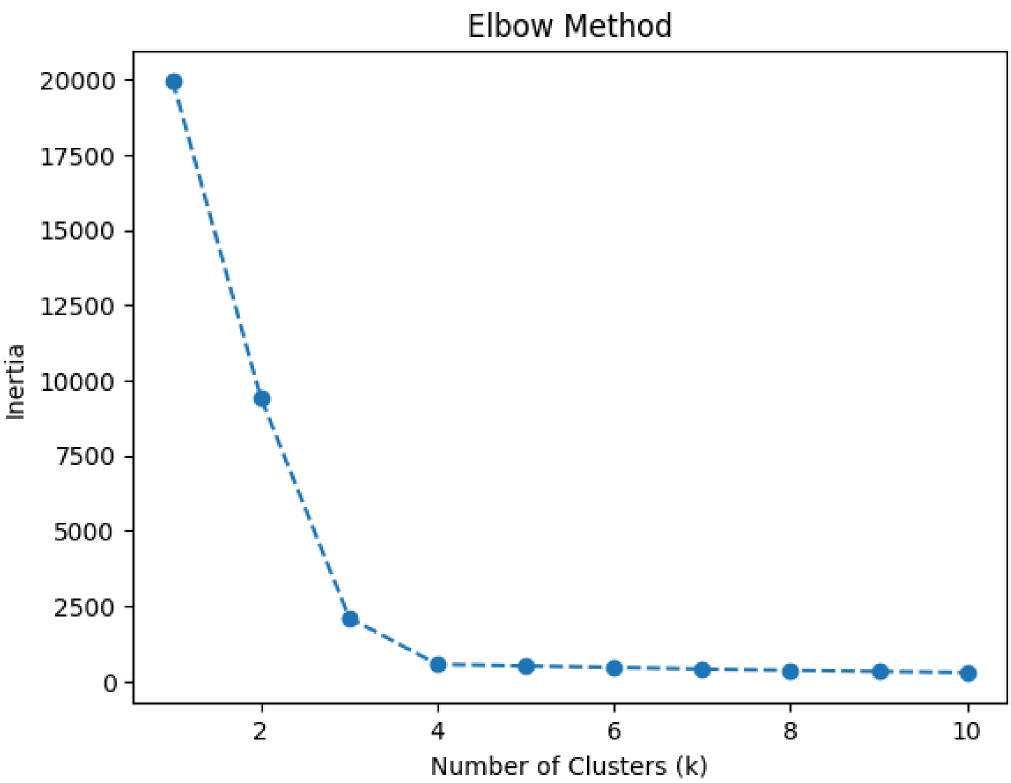
Aim:Apply the K-Means algorithm to group similar data points into clusters. Determine the optimal number of clusters using elbow method or silhouette analysis. Visualize the clustering results and analyze the cluster characteristics.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
```

In [2]:
```
# Generate synthetic data
X, _ = make_blobs(n_samples=300, centers=4, random_state=42)

# Step 1: Elbow Method to find the optimal number of clusters
inertia = []
K_range = range(1, 11)
```

In [3]:
```
for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X)
    inertia.append(kmeans.inertia_)
```

In [4]:
```
# Plot Elbow Curve
plt.plot(K_range, inertia, marker='o', linestyle='--')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Method')
plt.show()
```

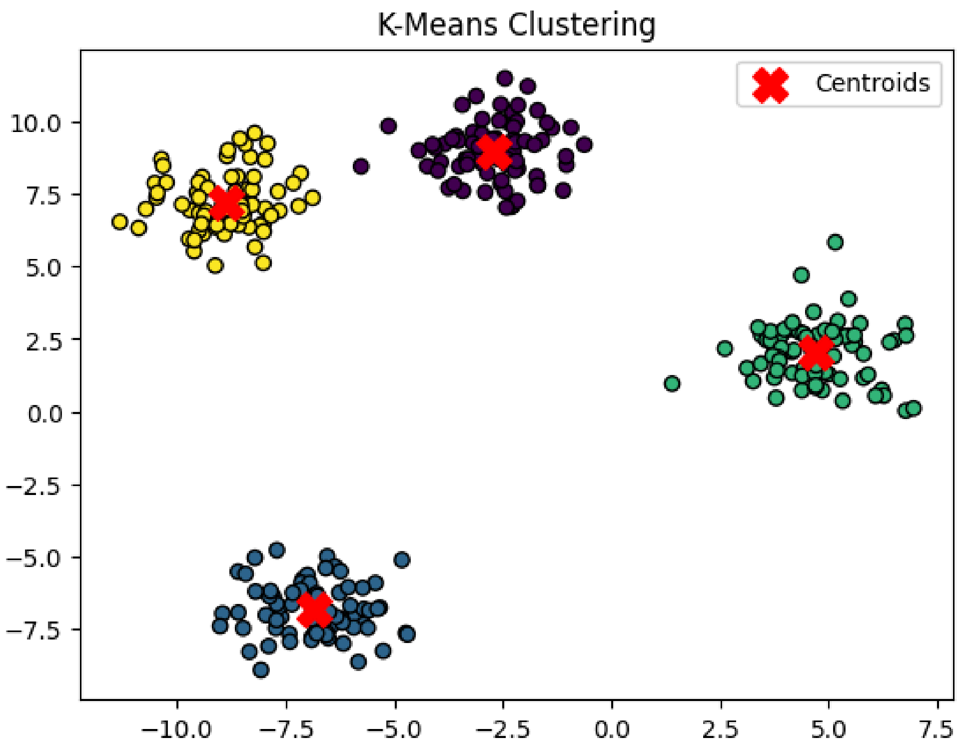


In [5]:

```
# Step 2: Apply K-Means with the chosen k (let's pick k=4)
kmeans = KMeans(n_clusters=4, random_state=42, n_init=10)
y_kmeans = kmeans.fit_predict(X)
```

In [6]:

```
# Step 3: Visualize Clustering Results
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, cmap='viridis', edgecolors='k')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            s=200, c='red', marker='X', label='Centroids')
plt.title('K-Means Clustering')
plt.legend()
plt.show()
```



Practical No: 06 - Principal Component Analysis (PCA)

Aim: erform 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.

In [1]:

```
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
```

In [2]:

```
# Load dataset (Iris dataset)
data = load_iris()
X = data.data   # Features
y = data.target  # Labels
```

In [3]:

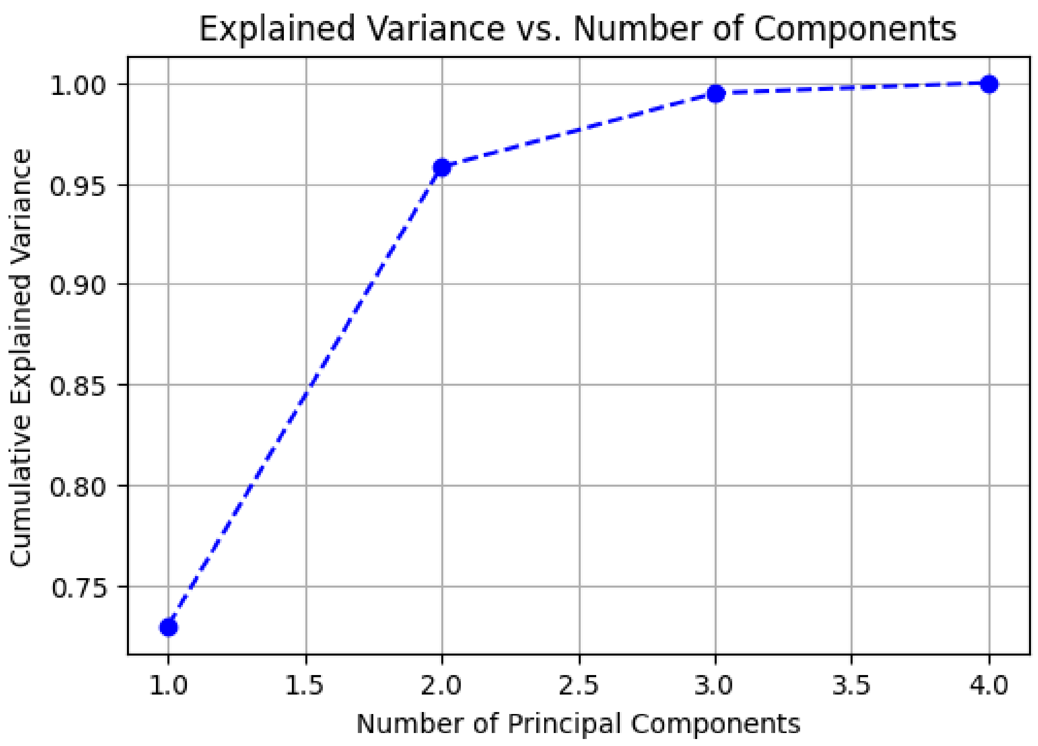
```
# Standardize the data (important for PCA)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Perform PCA
pca = PCA()
X_pca = pca.fit_transform(X_scaled)
```

In [4]:

```
# Evaluate explained variance
explained_variance = pca.explained_variance_ratio_
cumulative_variance = np.cumsum(explained_variance)
```

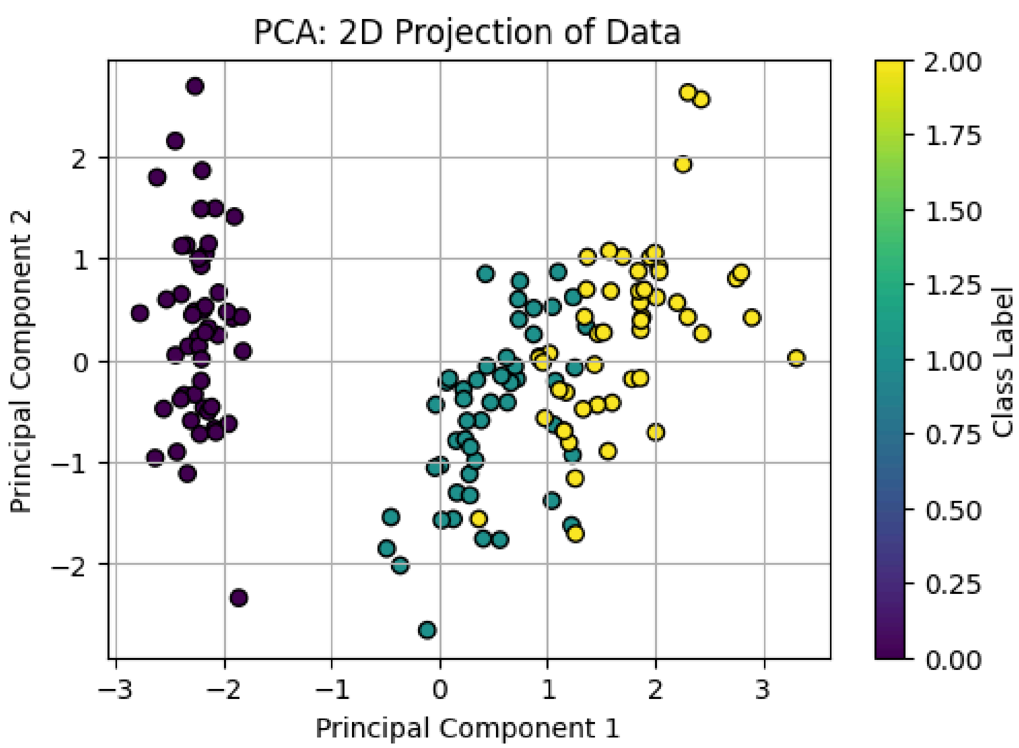
In [5]:

```
# Plot explained variance
plt.figure(figsize=(6, 4))
plt.plot(range(1, len(explained_variance) + 1), cumulative_variance, marker='o', line
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Explained Variance vs. Number of Components')
plt.grid(True)
plt.show()
```



In [6]:

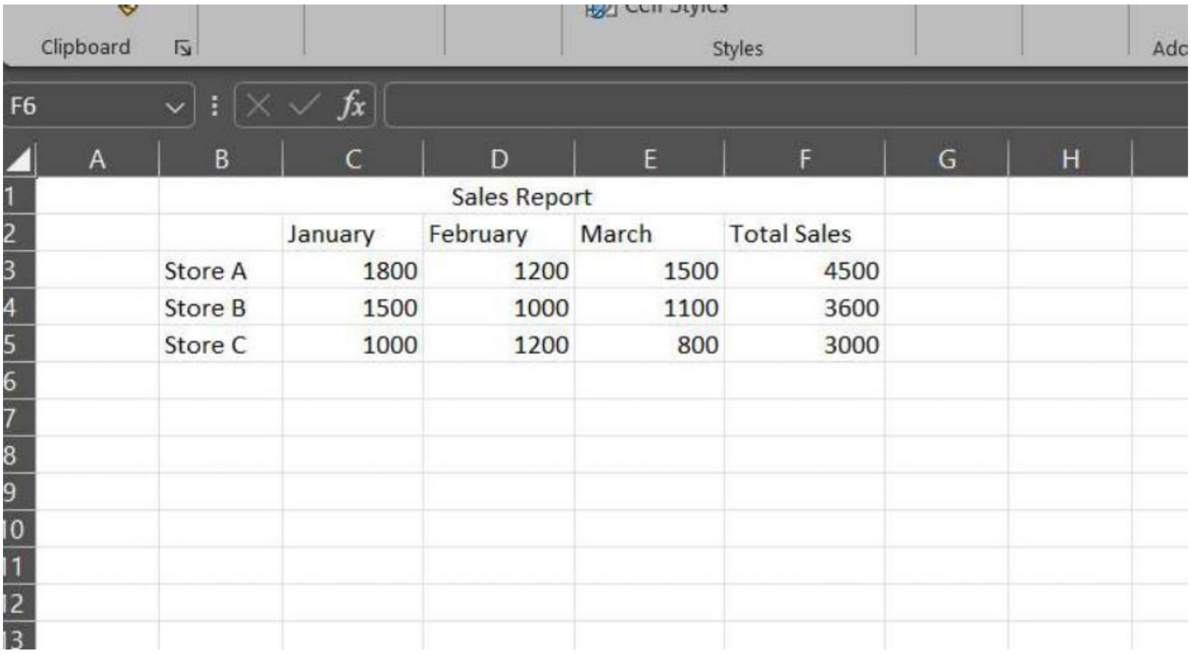
```python
# Choose first two principal components for visualization
pca_2d = PCA(n_components=2)
X_pca_2d = pca_2d.fit_transform(X_scaled)

# Scatter plot of the first two principal components
plt.figure(figsize=(6, 4))
plt.scatter(X_pca_2d[:, 0], X_pca_2d[:, 1], c=y, cmap='viridis', edgecolor='k')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA: 2D Projection of Data')
plt.colorbar(label='Class Label')
plt.grid(True)
plt.show()
```



PRACTICAL 7

Introduction to Excel

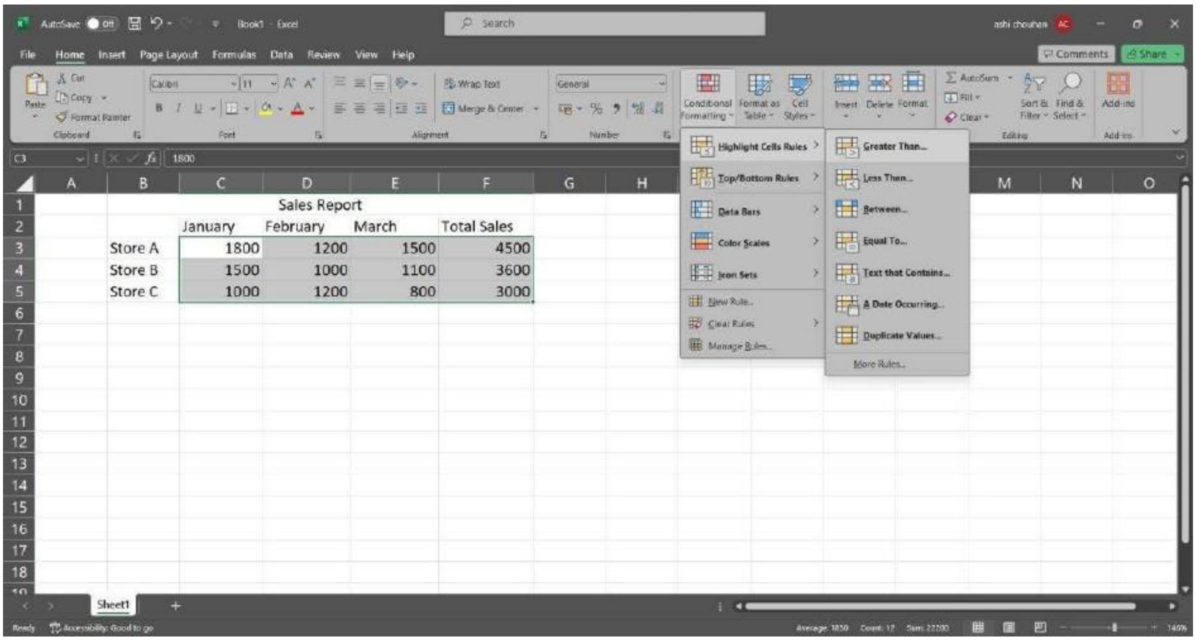
A. Perform conditional formatting on a dataset using various criteria.



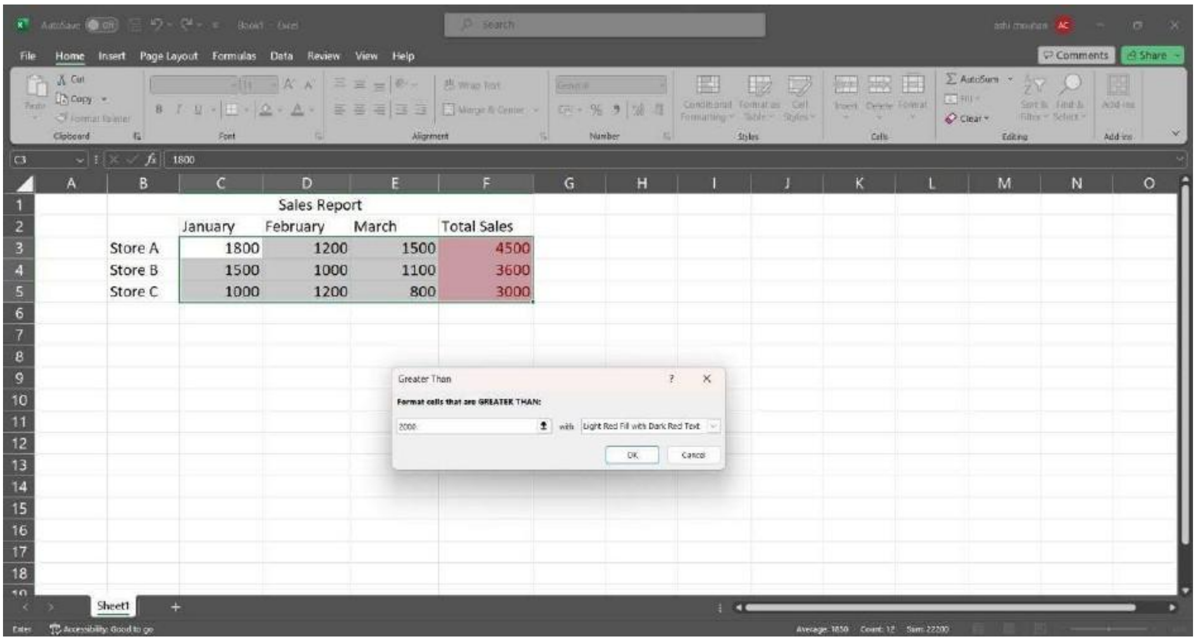
	Sales Report			
	January	February	March	Total Sales
Store A	1800	1200	1500	4500
Store B	1500	1000	1100	3600
Store C	1000	1200	800	3000

Steps

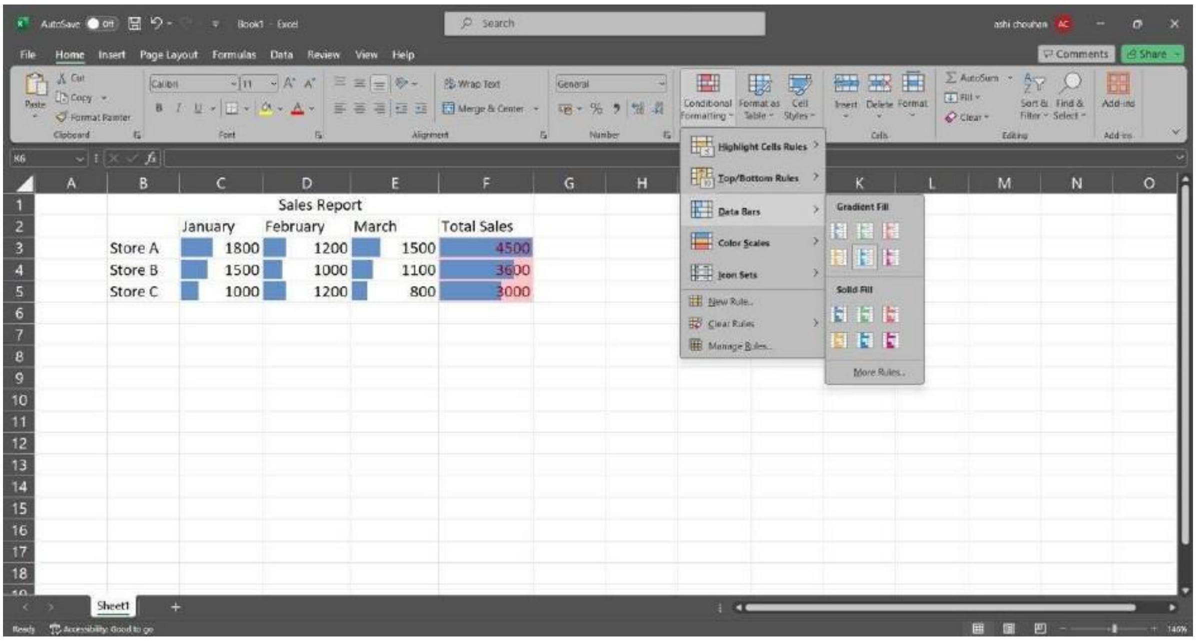
Step l: Go to conditional formatting > Greater Than



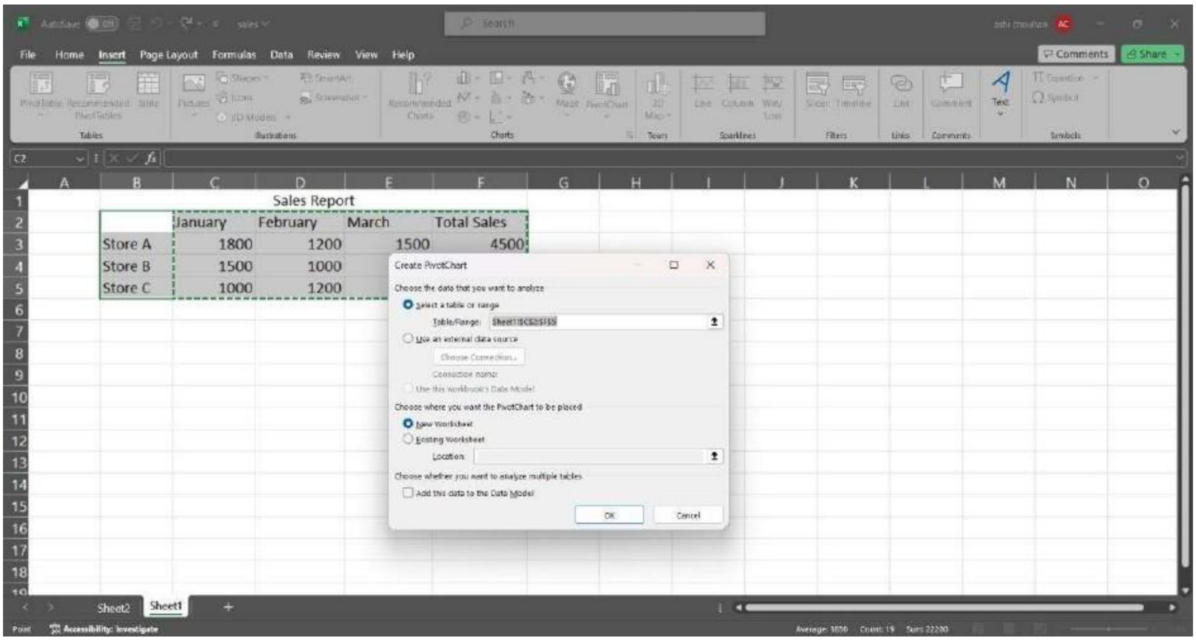
Step 2: Enter the greater than filter value for example 2000.



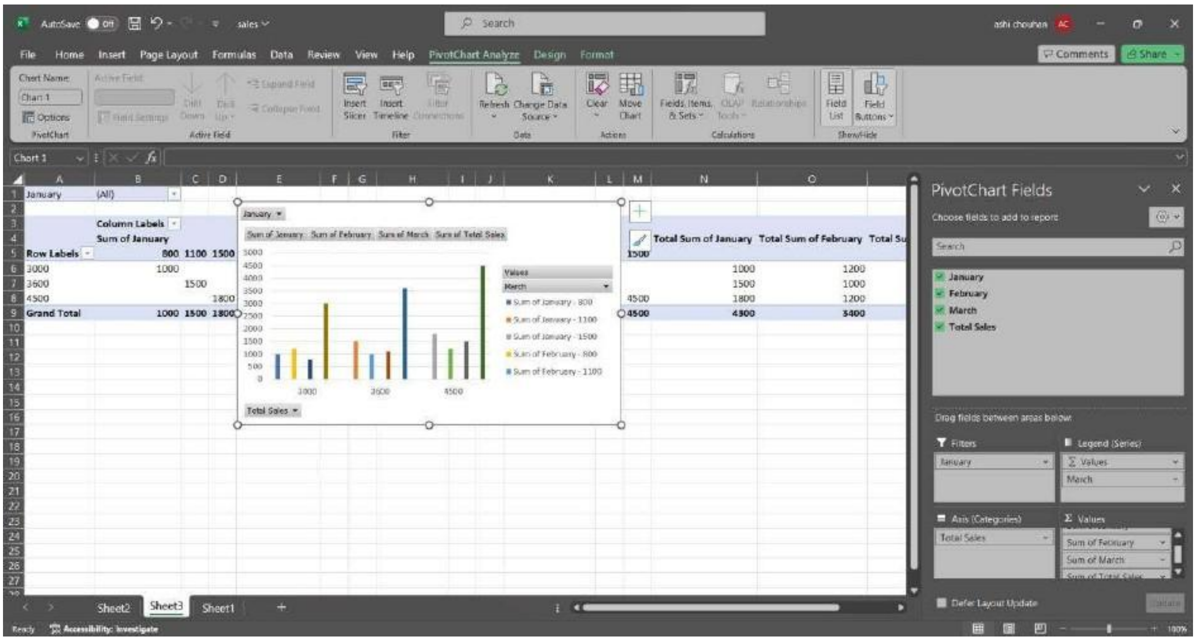
Step 3: Go to Data Bars > Solid Fill in conditional formatting.



B. Create a pivot table to analyse and summarize data. Steps
   Step I : 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.

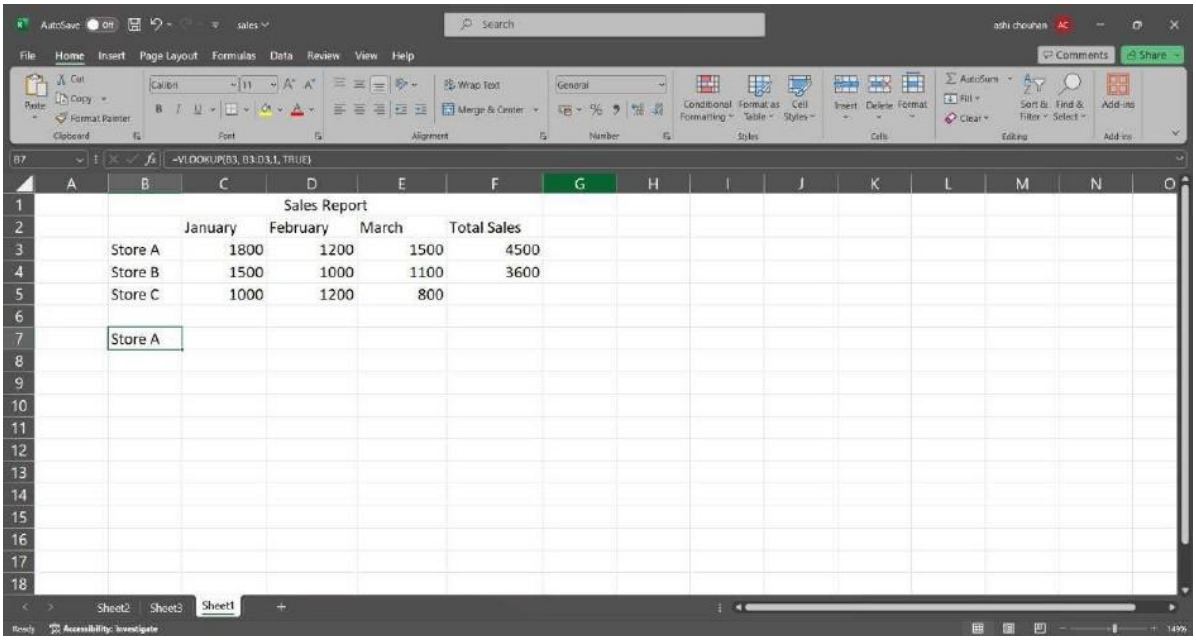


C. Use VLOOKUP function to retrieve information from a different worksheet or table.

Steps:

Step l : click on an empty cell and type the following command.

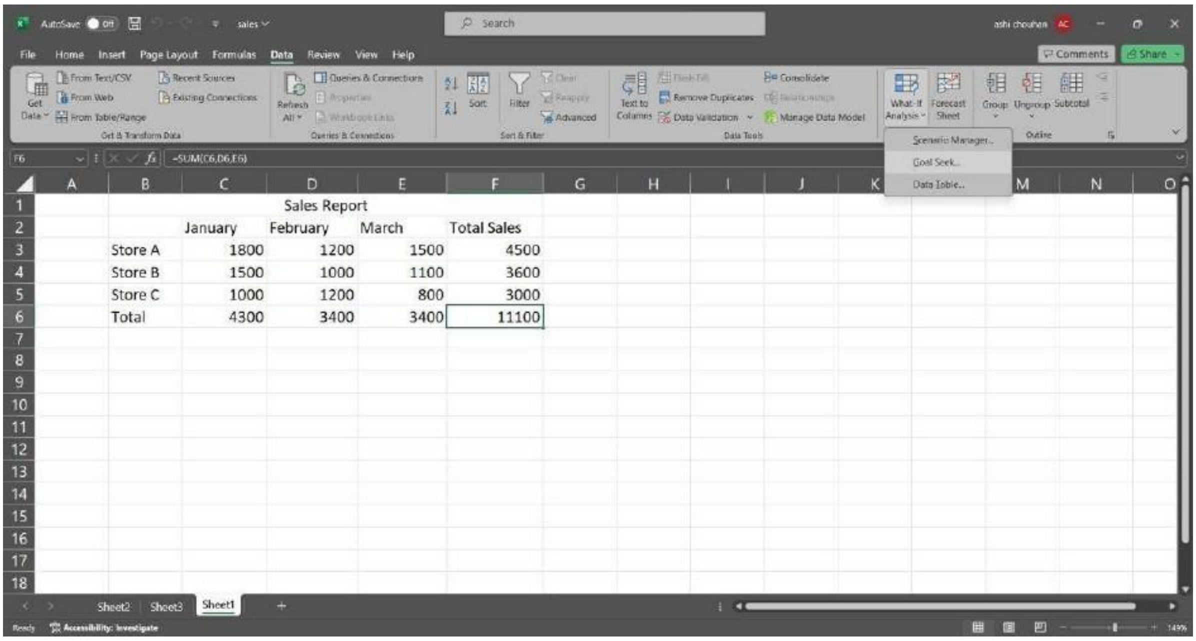
=VLOOKUP(B3, B3:D3,1,TRUE)



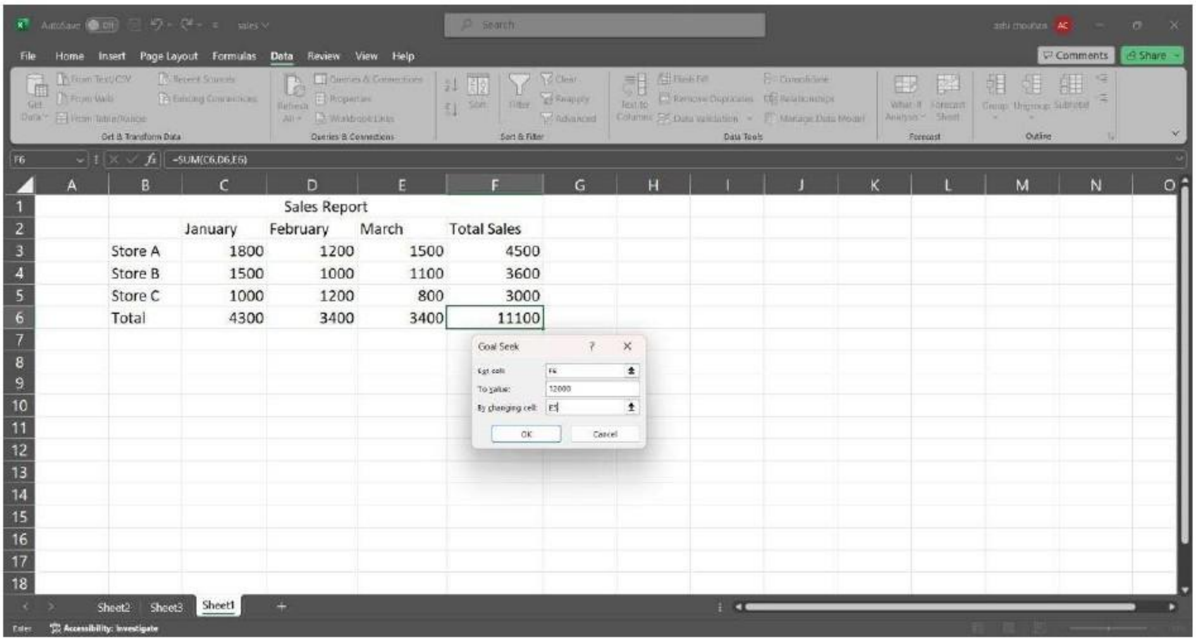
D. Perform what-if analysis using Goal Seek to determine input values for desired output.

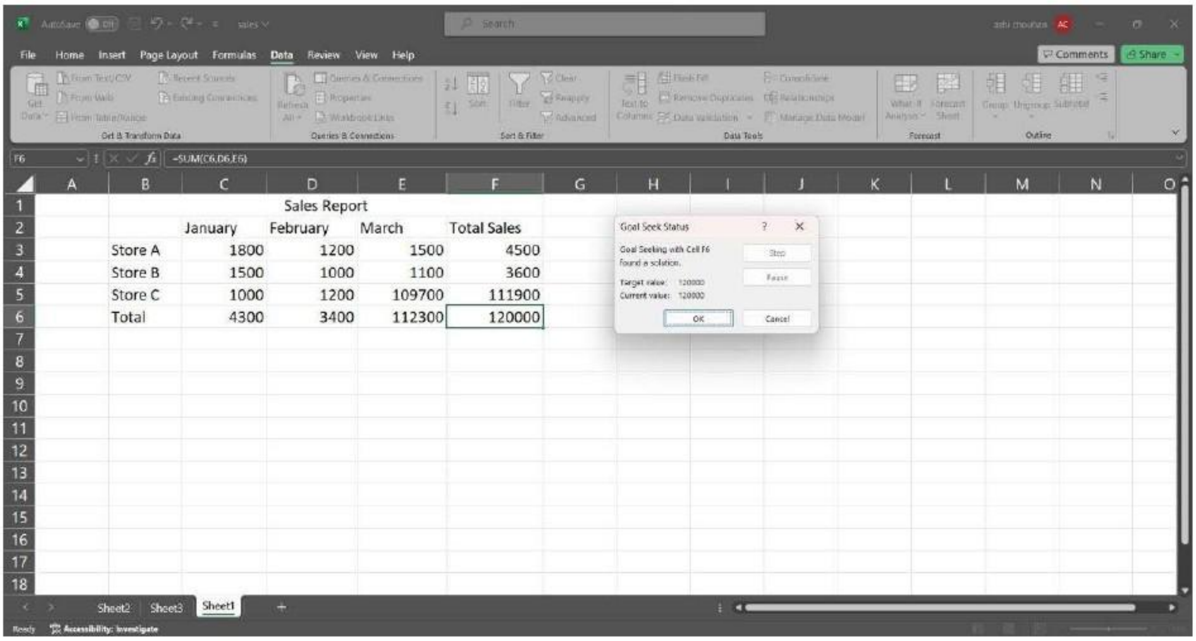
Steps-

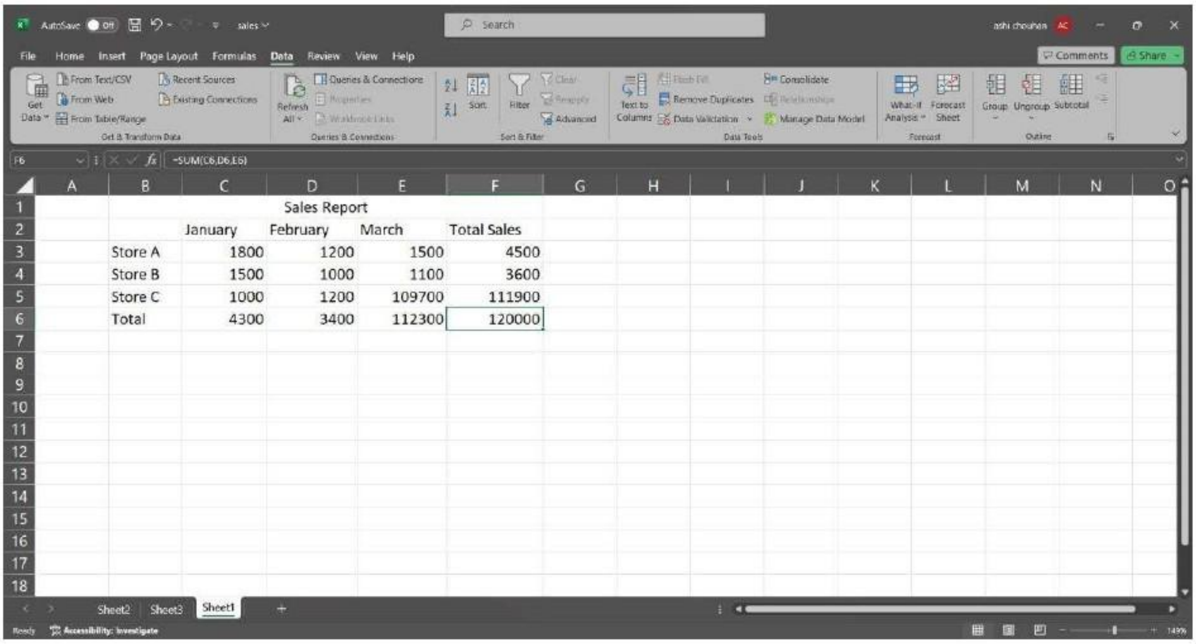
Step 1: In the Data tab go to the what if analysis>Goal seek.



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


File
Home
Insert
Page Layout
Formulas
Data
Review
View
Help
Comments
Share
=SUM(C6,D6,E6)
Sales Report
January
February
March
Total Sales
Store A 1800 1200 1500 4500
Store B 1500 1000 1100 3600
Store C 1000 1200 800 3000
Total 4300 3400 3400 11100
Goal Seek
Set cell: F6
To value: 12000
By changing cell: E5
OK
Cancel
Sheet2
Sheet3
Sheet1


File
Home
Insert
Page Layout
Formulas
Data
Review
View
Help
Comments
Share
=SUM(C6,D6,E6)
Sales Report
January
February
March
Total Sales
Store A 1800 1200 1500 4500
Store B 1500 1000 1100 3600
Store C 1000 1200 109700 111900
Total 4300 3400 112300 120000
Goal Seek Status
Goal Seeking with Cell F6
found a solution.
Target value: 120000
Current value: 120000
OK
Cancel
Sheet2
Sheet3
Sheet1



			Sales Report		
		January	February	March	Total Sales
	Store A	1800	1200	1500	4500
	Store B	1500	1000	1100	3600
	Store C	1000	1200	109700	111900
	Total	4300	3400	112300	120000

Aim:Create Pivot Table in Excel For following Analysis and Visualize the data using Pivot Chart

Steps:

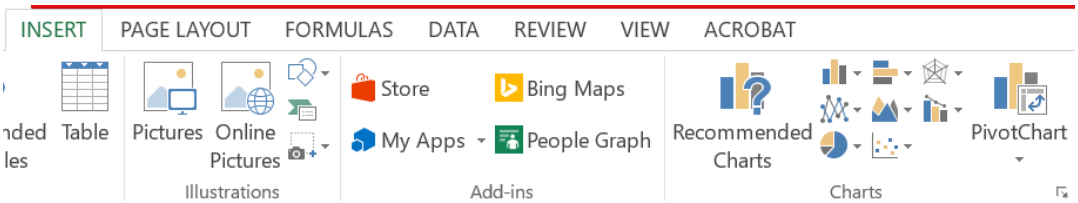
1.)First Create the Excel Data

	A	B	C	D	E
1	Date	Color	Region	Units	Sales
2	03-Jan-20	Red	West	1	110000
3	14-Jan-20	blue	South	8	96000
4	21-Jan-20	green	west	2	26000
5	30-Jan-20	blue	north	7	84000
6	07-Feb-20	green	north	8	25000
7	13-Feb-20	red	south	2	60000
8	22-Feb-20	blue	east	5	35000
9	01-Mar-20	green	west	2	87000
10	13-Mar-20	blue	east	8	69000
11	23-Mar-20	blue	north	7	54000
12					

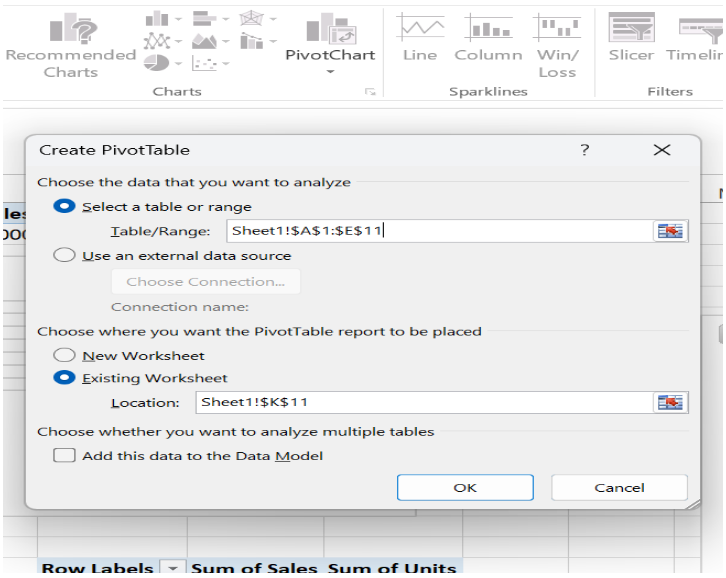
2.)Select the entire dataset

A	B	C	D	E
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

3.)Select the pivot chart and pivot table option from insert tab



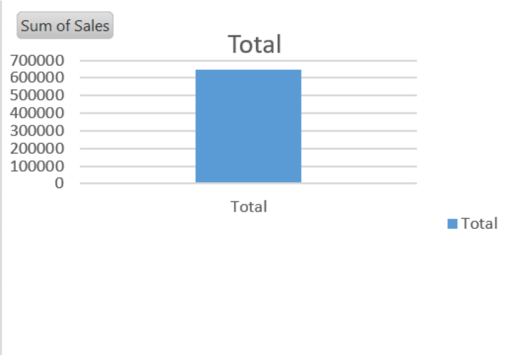
4.)Select the table range and cell where you want the output



5.)Perform the following Questions

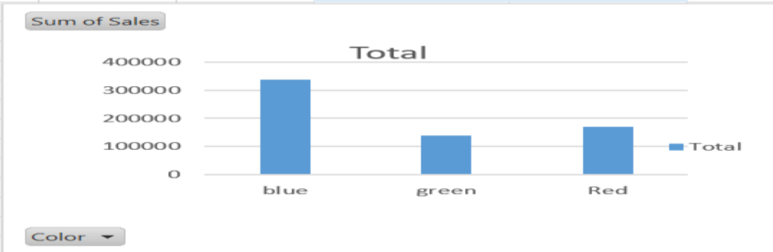
i)Find out the total sales

Sum of Sales
646000



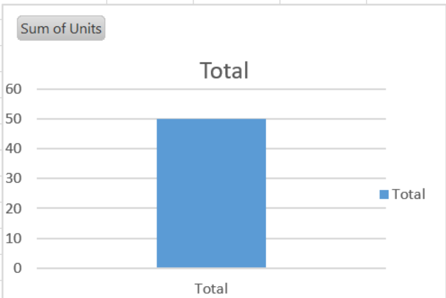
ii)Find out the sum of sales Color wise

Row Labels	Sum of Sales
blue	338000
green	138000
Red	170000
Grand Total	646000



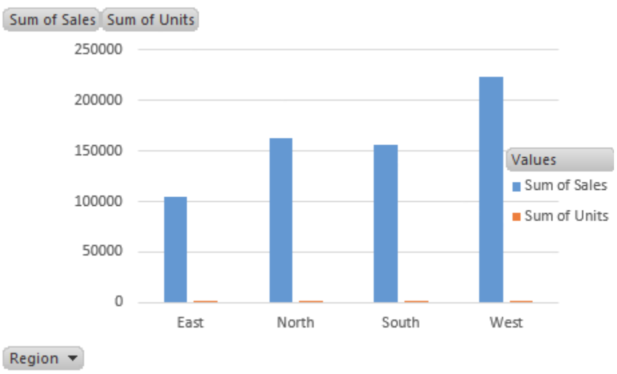
iii)Find out the sum of units

Sum of Units
50



iv)Find out Region wise total sales and total units

Row Labels	Sum of Sales	Sum of Units
east	104000	13
north	163000	22
South	156000	10
West	223000	5
Grand Total	646000	50



Aim:Apply Vlookup functions to retrieve information for the following Queries

B	C	D	E	F
Part Number	Part Name	Part Price	Status	Supplier ID
A001	water	6800	IN	SP301
A002	altenator	3800	IN	SP302
A003	air filter	4500	IN	SP303
A004	wheel bearing	3582	IN Stock	SP304
A005	muffler	1600	IN	SP305
A006	oil pan	1005	Out of stock	SP306
A007	brake pads	6500	IN	SP307
A008	brake rotors	8549	Out of stock	SP308
A009	headlight	6500	IN	SP309
A010	brake	1500	IN	SP310
A011	Strut	4500	IN	SP311
A012	Deive	1580	IN	SP312
A013	CV Book Kit	2650	IN Stock	SP313
A014	Oil Pump	4660	IN	SP314
A015	oil filter	4350	IN	SP315
A016	Fuel filter	1280	IN	SP316
A017	Tie Road End	1800	IN	SP317
A018	Ball joint	2500	IN	SP318
A019	Steering Rack	2700	Out of stock	SP319
A020	Piston	4500	Out of stock	SP320

Q.1)Find the part name for part number"A002"

1.) =VLOOKUP(B3,B2:E21,2,FALSE)

Output: altenator

Q.2)Find the Supplier ID for part name "Ball Joint"

2.) =VLOOKUP("Ball joint",C2:F21,4,FALSE)

Output: SP318

Q.3)Find the Part Price for part name "Muffer"

3.) =VLOOKUP("muffler",C2:E21,2,FALSE)

Output: 1600

Q.4)Find the Status of part number "A008"

4.) =VLOOKUP(B9,B2:E21,4,FALSE)

Output: Out of stock

PRACTICAL 8

Hypothesis Testing

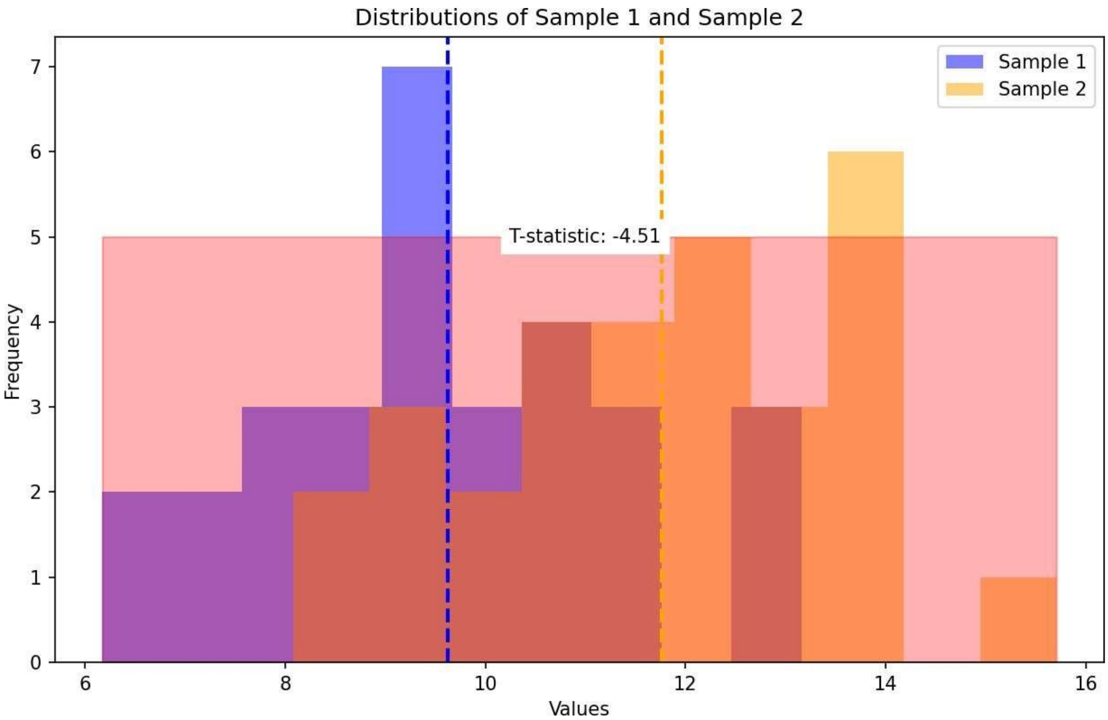
Conduct a hypothesis test using appropriate statistical tests (e.g., t-test, chi-square test)

```
# t-test
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt



# Generate two samples for demonstration purposes
np.random.seed(42)
sample1 = np.random.normal(loc=10, scale=2, size=30)
sample2 = np.random.normal(loc=12, scale=2, size=30)

# Perform a two-sample t-test
t_statistic, p_value = stats.ttest_ind(sample1, sample2)

# Set the significance level
alpha = 0.05

print("Results of Two-Sample t-test:")
print(f'T-statistic: {t_statistic}')
print(f'P-value: {p_value}')
print(f"Degrees of Freedom: {len(sample1) + len(sample2) - 2}")

# Plot the distributions
plt.figure(figsize=(10, 6))
plt.hist(sample1, alpha=0.5, label='Sample 1', color='blue')
```

```
plt.hist(sample2, alpha=0.5, label='Sample 2', color='orange')
plt.axvline(np.mean(sample1), color='blue', linestyle='dashed', linewidth=2)
plt.axvline(np.mean(sample2), color='orange', linestyle='dashed', linewidth=2)
plt.title('Distributions of Sample 1 and Sample 2')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.legend()

# Highlight the critical region if null hypothesis is rejected
if p_value < alpha:
    critical_region = np.linspace(min(sample1.min(), sample2.min()), max(sample1.max(), sample2.max()), 1000)
    plt.fill_between(critical_region, 0, 5, color='red', alpha=0.3, label='Critical Region')
    plt.text(11, 5, f'T-statistic: {t_statistic:.2f}', ha='center', va='center', color='black', backgroundcolor='white')

# Show the plot
plt.show()

# Draw Conclusions
if p_value < alpha:
    if np.mean(sample1) > np.mean(sample2):
        print("Conclusion: There is significant evidence to reject the null hypothesis.")
        print("Interpretation: The mean of Sample 1 is significantly higher than that of Sample 2.")
    else:
        print("Conclusion: There is significant evidence to reject the null hypothesis.")
        print("Interpretation: The mean of Sample 2 is significantly higher than that of Sample 1.")
else:
    print("Conclusion: Fail to reject the null hypothesis.")
    print("Interpretation: There is not enough evidence to claim a significant difference between the means.")
```

Output:

```
================================ RESTART: E:/all no
Results of Two-Sample t-test:
T-statistic: -4.512913234547555
P-value: 3.176506547470154e-05
Degrees of Freedom: 58
```



#chi-test

import pandas as pd

import numpy as np

import matplotlib as plt

import seaborn as sb

import warnings

```
from scipy import stats
warnings.filterwarnings('ignore')
df=sb.load_dataset('mpg')
print(df)
print(df['horsepower'].describe())
print(df['model_year'].describe())
bins=[0,75,150,240]
df['horsepower_new']=pd.cut(df['horsepower'],bins=bins,labels=['l','m','h'])
c=df['horsepower_new']
print(c)
ybins=[69,72,74,84]
label=['t1','t2','t3']
df['modelyear_new']=pd.cut(df['model_year'],bins=ybins,labels=label)
newyear=df['modelyear_new']
print(newyear)
df_chi=pd.crosstab(df['horsepower_new'],df['modelyear_new'])
print(df_chi)
print(stats.chi2_contingency(df_chi)
```

Output:

```
======================= RESTART: E:/all notes/DS/prac_4.1.py ==================
      mpg  cylinders  ...  origin                       name
0    18.0          8  ...     usa  chevrolet chevelle malibu
1    15.0          8  ...     usa          buick skylark 320
2    18.0          8  ...     usa         plymouth satellite
3    16.0          8  ...     usa              amc rebel sst
4    17.0          8  ...     usa                ford torino
..    ...        ...  ...     ...                        ...
393  27.0          4  ...     usa            ford mustang gl
394  44.0          4  ...  europe                  vw pickup
395  32.0          4  ...     usa              dodge rampage
396  28.0          4  ...     usa                ford ranger
397  31.0          4  ...     usa                 chevy s-10

[398 rows x 9 columns]
count    392.000000
mean     104.469388
std       38.491160
min       46.000000
25%       75.000000
50%       93.500000
75%      126.000000
max      230.000000
```

```
Name: horsepower, dtype: float64
count    398.000000
mean      76.010050
std        3.697627
min       70.000000
25%       73.000000
50%       76.000000
75%       79.000000
max       82.000000
Name: model_year, dtype: float64
0      m
1      h
2      m
3      m
4      m
      ..
393    m
394    l
395    m
396    m
397    m
```

```
Name: horsepower_new, Length: 398, dtype: category
Categories (3, object): ['l' < 'm' < 'h']
0      t1
1      t1
2      t1
3      t1
4      t1
       ..
393    t3
394    t3
395    t3
396    t3
397    t3
Name: modelyear_new, Length: 398, dtype: category
Categories (3, object): ['t1' < 't2' < 't3']
modelyear_new    t1  t2   t3
horsepower_new
l                 9  14   76
m                49  41  158
h                26  11    8
(54.95485392447537, 3.320518009555984e-11, 4, array([[ 21.21428571,  16.66836735,  61.11734694]
,
       [ 53.14285714,  41.75510204, 153.10204082],
       [  9.64285714,   7.57653061,  27.78061224]]))
```

Conclusion: There is sufficient evidence to reject the null hypothesis, indicating that there is a significant association between 'horsepower_new' and 'modelyear_new' categories.