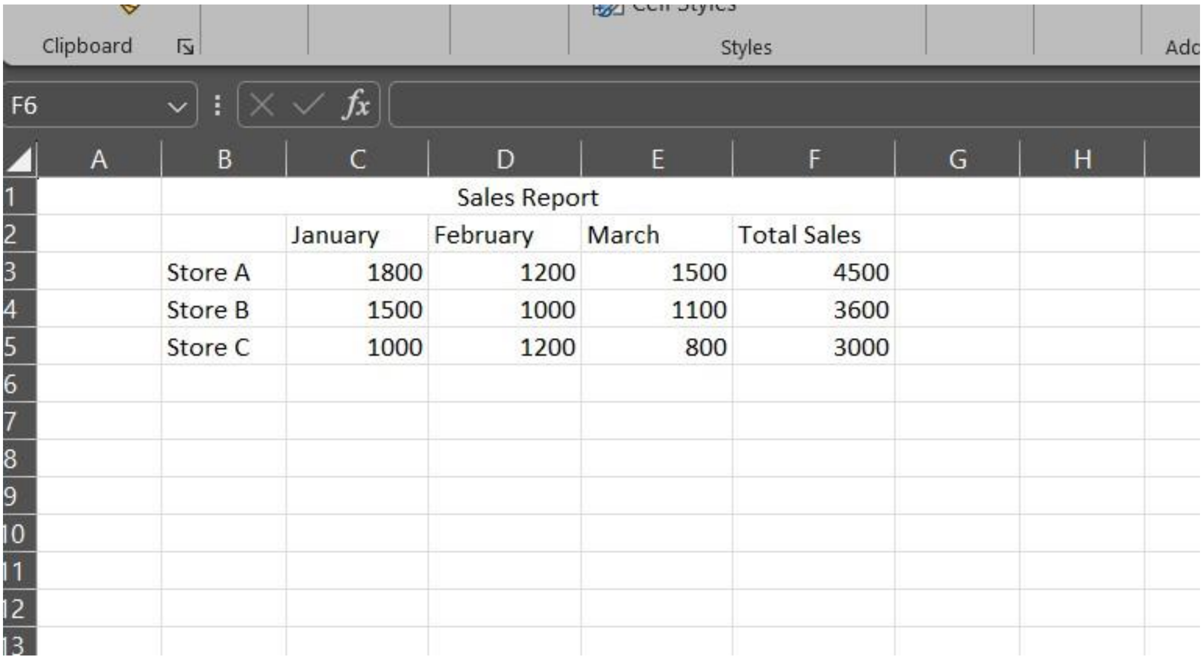
PRACTICAL 1

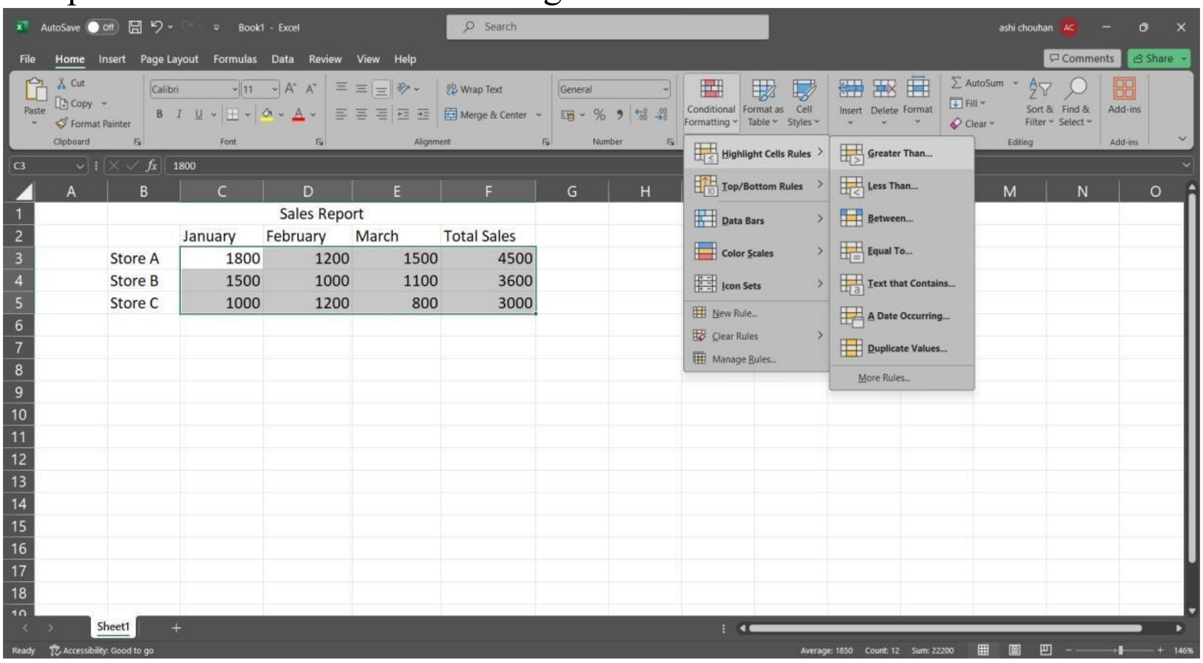
Introduction to Excel

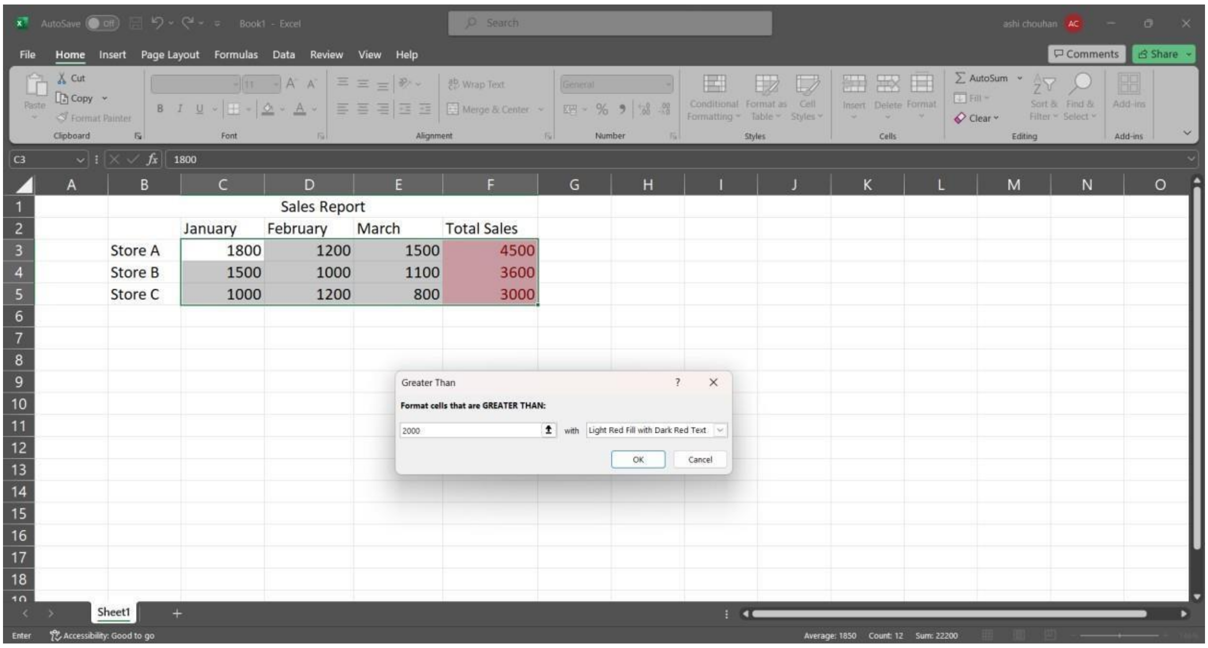
A. Perform conditional formatting on a dataset using various criteria.



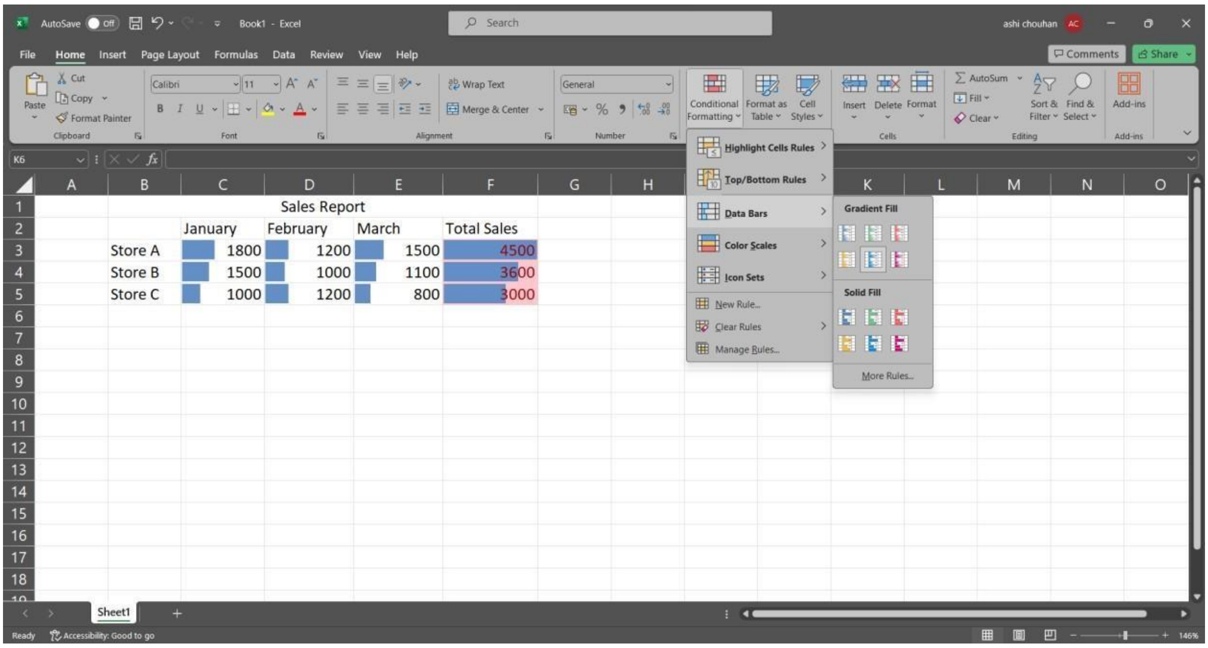
Steps

Step 1: Go to conditional formatting > Greater Than





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

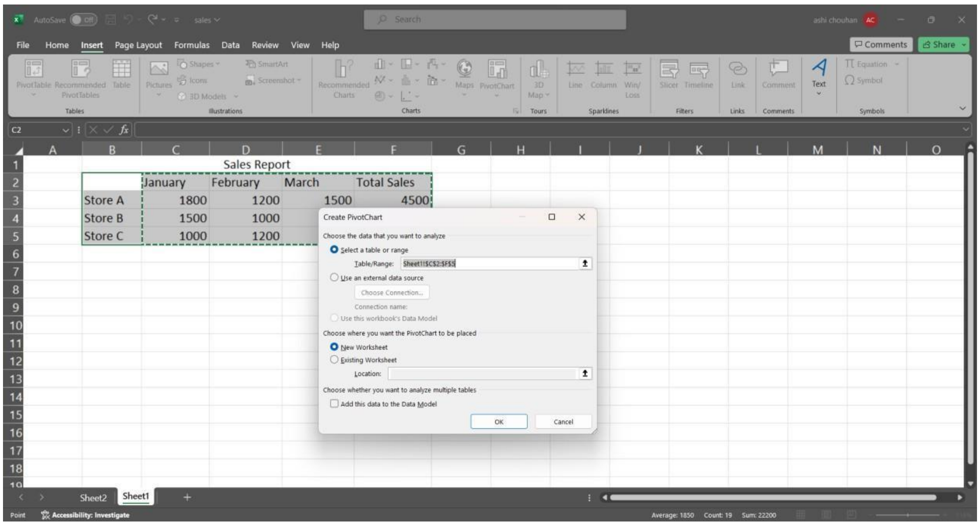


A. Create a pivot table to analyse and summarize data.

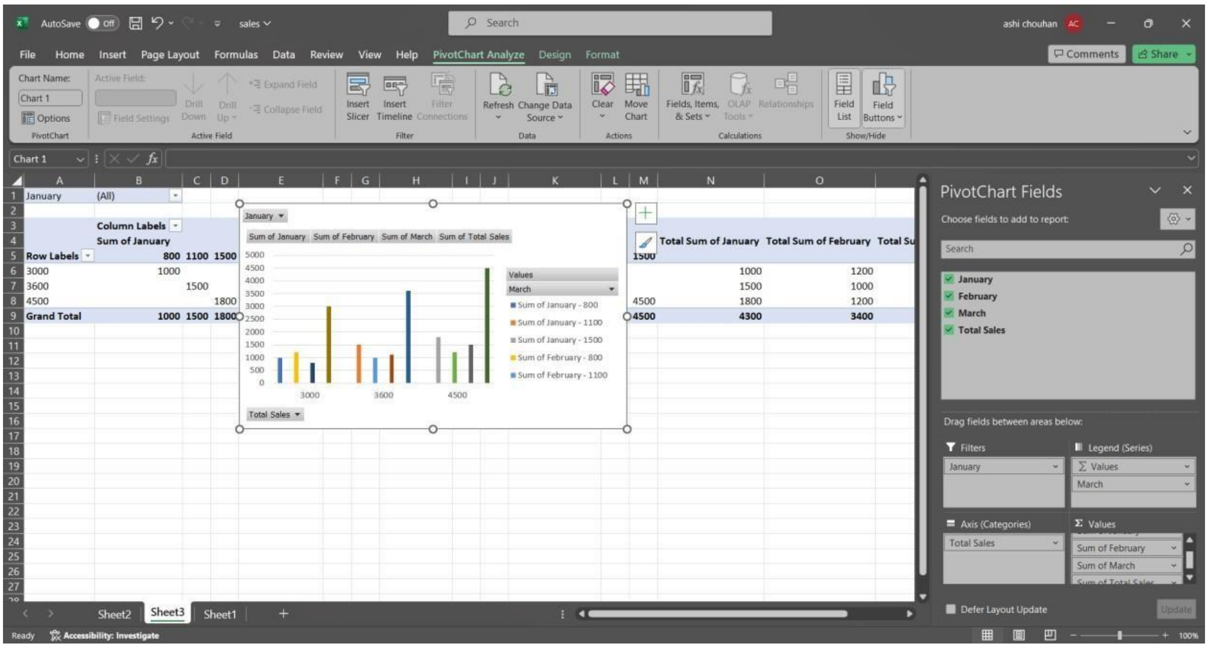
Steps

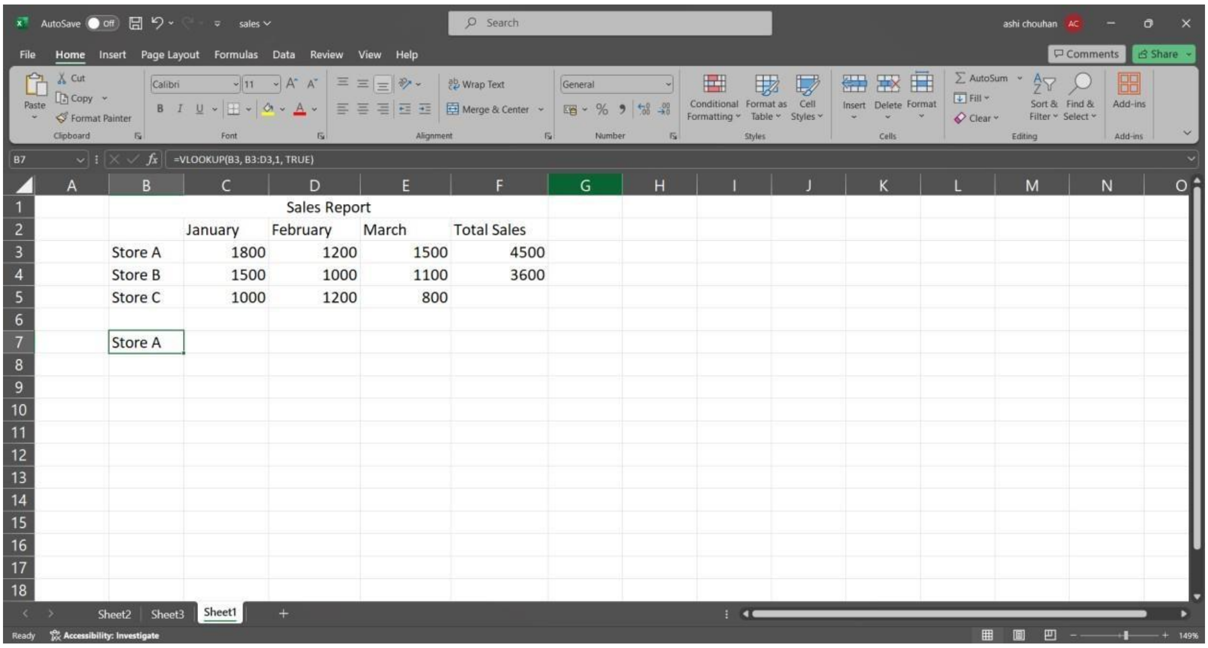
Step 1: select the entire table and go to Insert tab PivotChart > Pivotchart .

Step 2: Select “New worksheet” in the create pivot chart window.



Step 3: Select and drag attributes in the below boxes.

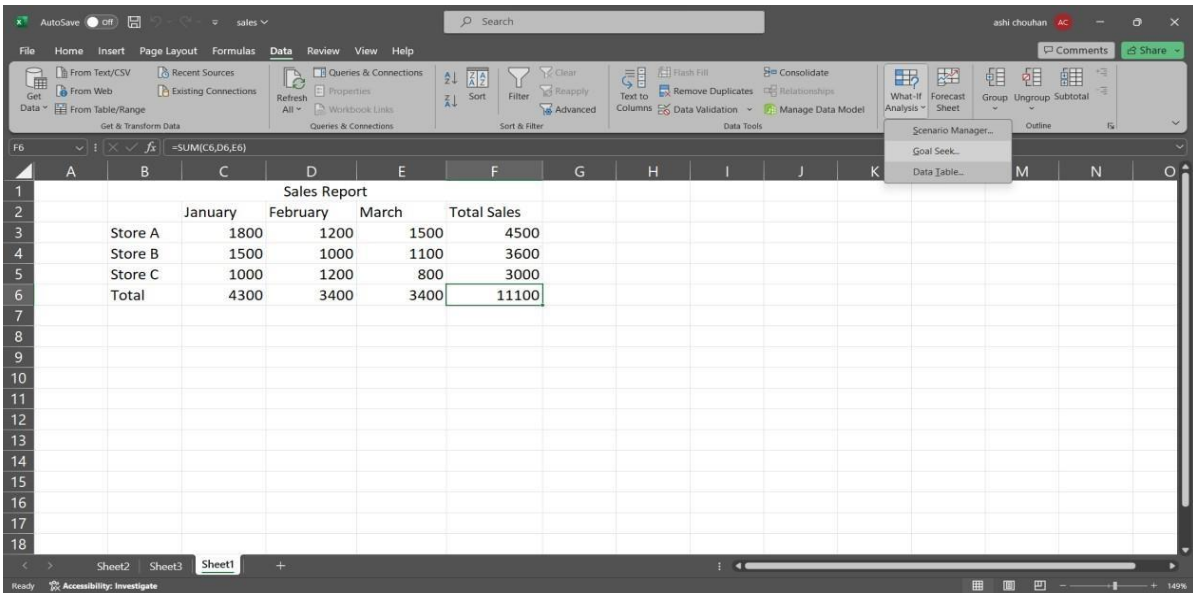




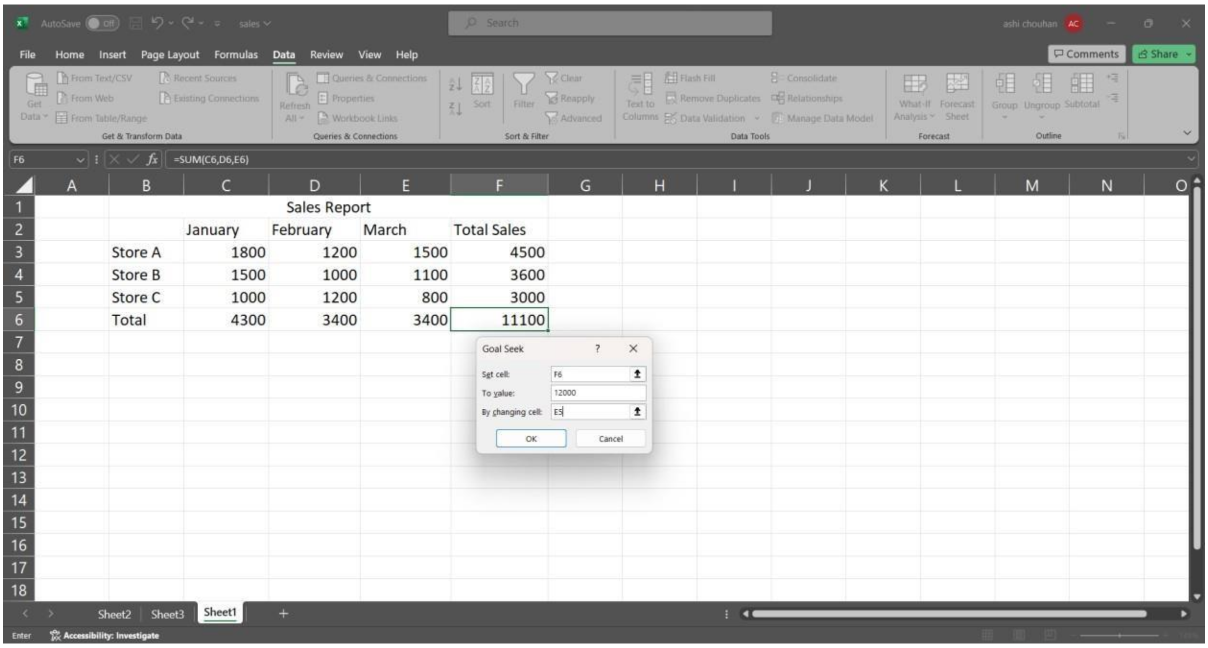
B. Perform what-if analysis using Goal Seek to determine input values for desired output.

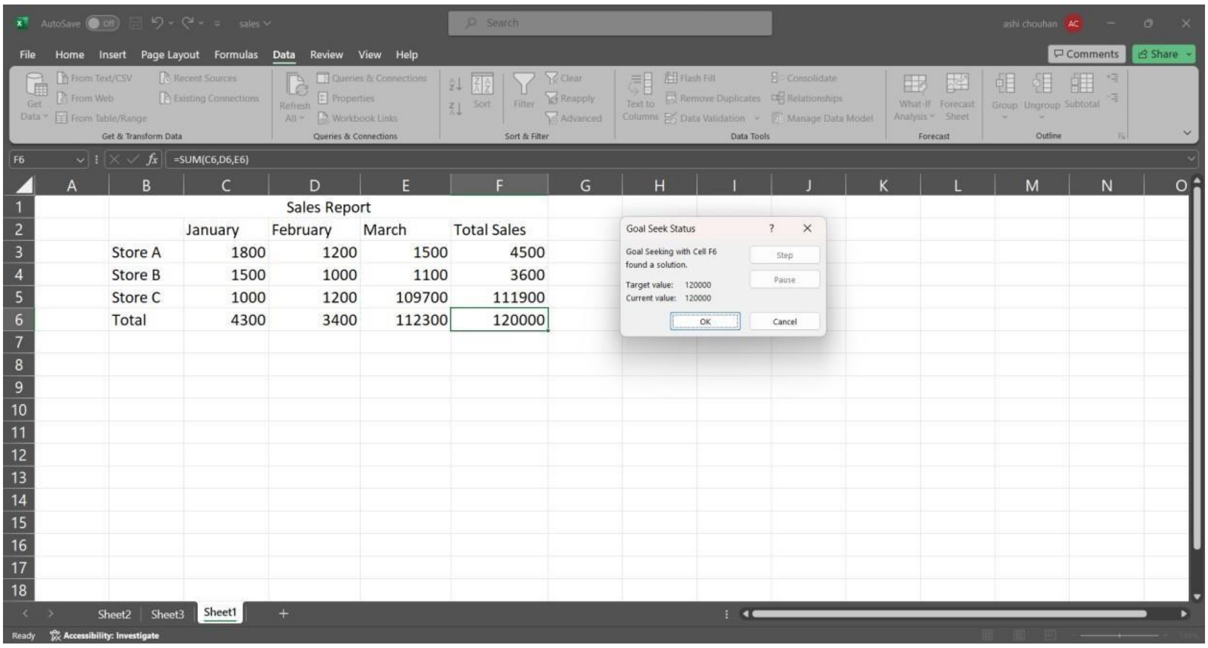
Steps-

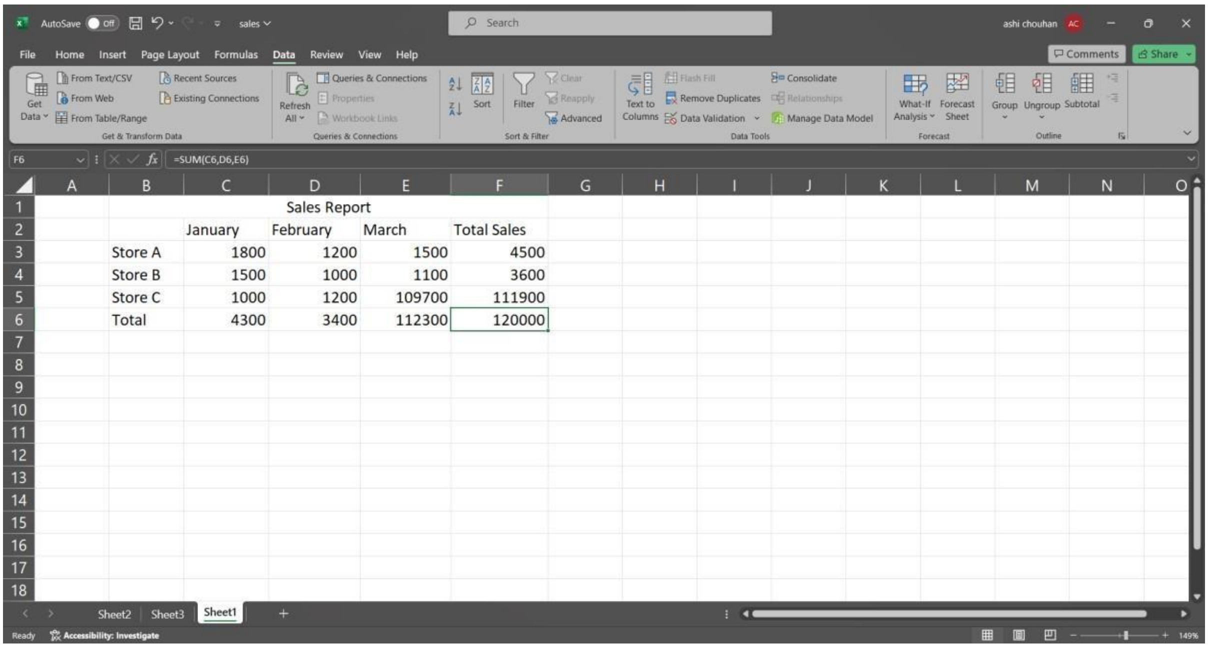
Step 1: In the Data tab go to the what if analysis>Goal seek



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






AutoSave
sales
Search
ashi chouhan
File Home Insert Page Layout Formulas Data Review View Help
Comments
Share
F6
=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
Sheet2 Sheet3 Sheet1
Ready
Accessibility: Investigate

PRACTICAL 2

Aim: Data Frames and Basic Data Pre-processing

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

Code:

```
#Read data from CSV and JSON files into a data frame.
# Read data from a csv file
import pandas as pd
df = pd.read_csv('Student_Marks.csv')
print("Our dataset",df)
# Reading data from a JSON file
import pandas as pd
df = pd.read_json('dataset.json')
print("Our dataset",df)
```

Output:

```
= RESTART: C:\Users\Neeraj\Desktop\Tycs\sem 6\Data Science\practical\practical no 2 a.py
Our dataset   Unnamed: 0 Gender         DOB  Maths  ...  Biology  Economics  History  Civics
0        John      M  05-04-1988     55  ...       21         52         89       65
1      Suresh      M  04-05-1987     75  ...       90         61         58        2
2      Ramesh      M  25-05-1989     25  ...       95         87         56       74
3     Jessica      F  12-08-1990     78  ...       54         89         75       45
4    Jennifer      F  02-09-1989     58  ...       96         77         83       53
5        Annu      F  05-04-1988     45  ...       55         89         87       52
6       pooja      F  04-05-1987     55  ...       75         58         64       61
7      Ritesh      M  25-05-1989     54  ...       25         56         76       87
8       Farha      F  12-08-1990     55  ...       78         75         63       89
9      Mukesh      M  02-09-1989     96  ...       58         83         46       77

[10 rows x 11 columns]
Our dataset     fruit    size   color
0   Apple   Large     Red
1  Banana  Medium  Yellow
2  Orange   Small  Orange
```

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

Code:

```
#Perform basic data pre-processing tasks such as handling missing values and outliers.
# Replacing NA values using fillna()
import pandas as pd
df =pd.read_csv('titanic.csv')
print(df)
df.head(10)
print("Dataset after filling NA values with 0:")
df2=df.fillna(value=0)
print(df2)
```

Output:

```
= RESTART: C:\Users\Neeraj\Desktop\Tycs\sem 6\Data Science\practical\practical no 2 b.py
     PassengerId  Survived  Pclass  ...     Fare Cabin  Embarked
0              1         0       3  ...   7.2500   NaN         S
1              2         1       1  ...  71.2833   C85         C
2              3         1       3  ...   7.9250   NaN         S
3              4         1       1  ...  53.1000  C123         S
4              5         0       3  ...   8.0500   NaN         S
..           ...       ...     ...  ...      ...   ...       ...
886          887         0       2  ...  13.0000   NaN         S
887          888         1       1  ...  30.0000   B42         S
888          889         0       3  ...  23.4500   NaN         S
889          890         1       1  ...  30.0000  C148         C
890          891         0       3  ...   7.7500   NaN         Q

[891 rows x 12 columns]
Dataset after filling NA values with 0:
     PassengerId  Survived  Pclass  ...     Fare Cabin  Embarked
0              1         0       3  ...   7.2500     0         S
1              2         1       1  ...  71.2833   C85         C
2              3         1       3  ...   7.9250     0         S
3              4         1       1  ...  53.1000  C123         S
4              5         0       3  ...   8.0500     0         S
..           ...       ...     ...  ...      ...   ...       ...
886          887         0       2  ...  13.0000     0         S
887          888         1       1  ...  30.0000   B42         S
888          889         0       3  ...  23.4500     0         S
889          890         1       1  ...  30.0000  C148         C
890          891         0       3  ...   7.7500     0         Q

[891 rows x 12 columns]
```

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

Code:

```
import pandas as pd
# Load dataset
iris = pd.read_csv('Iris.csv')
# Check column names
print("Columns:", iris.columns)
# Filter 'setosa' species (correct column name is 'target')
setosa = iris[iris['target'] == 'Iris-setosa']
print(setosa.head())
# Sorting dataset
sorted_iris = iris.sort_values(by='sepal length (cm)', ascending=False)
print(sorted_iris.head())
# Group by 'target'
grouped_species = iris.groupby('target').mean(numeric_only=True)
print(grouped_species)
```

Output:

```
= RESTART: C:\Users\Neeraj\Desktop\Tycs\sem 6\Data Science\practical\practical no 2 c.py
Columns: Index(['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
       'petal width (cm)', 'target'],
      dtype='object')
   sepal length (cm)  sepal width (cm)  ...  petal width (cm)       target
0                5.1               3.5  ...               0.2  Iris-setosa
1                4.9               3.0  ...               0.2  Iris-setosa
2                4.7               3.2  ...               0.2  Iris-setosa
3                4.6               3.1  ...               0.2  Iris-setosa
4                5.0               3.6  ...               0.2  Iris-setosa

[5 rows x 5 columns]
     sepal length (cm)  sepal width (cm)  ...  petal width (cm)          target
131                7.9               3.8  ...               2.0  Iris-virginica
122                7.7               2.8  ...               2.0  Iris-virginica
118                7.7               2.6  ...               2.3  Iris-virginica
117                7.7               3.8  ...               2.2  Iris-virginica
135                7.7               3.0  ...               2.3  Iris-virginica

[5 rows x 5 columns]
                 sepal length (cm)  ...  petal width (cm)
target                              ...
Iris-setosa                  5.006  ...             0.244
Iris-versicolor              5.936  ...             1.326
Iris-virginica               6.588  ...             2.026

[3 rows x 4 columns]
```

PRACTICAL 3

Aim: Feature Scaling and Dummification

A. Apply feature-scaling techniques like standardization and normalization to numerical features.

Code:

```
# Apply feature-scaling techniques like standardization and normalization to
numericalfeatures.
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler, StandardScaler
# Load the dataset
df = pd.read_csv('wine.csv', header=None, usecols=[0, 1, 2], skiprows=1)
df.columns = ['classlabel', 'Alcohol', 'Malic Acid']
# Display the original dataframe
print("Original DataFrame:")
print(df.head())  # Print first few rows for better readability
# Apply Min-Max Scaling
minmax_scaler = MinMaxScaler()
df[['Alcohol', 'Malic Acid']] = minmax_scaler.fit_transform(df[['Alcohol', 'Malic Acid']])
print("\nDataFrame after Min-Max Scaling:")
print(df.head())
# Apply Standard Scaling (Z-score normalization)
standard_scaler = StandardScaler()
df[['Alcohol', 'Malic Acid']] = standard_scaler.fit_transform(df[['Alcohol', 'Malic Acid']])
print("\nDataFrame after Standard Scaling:")
print(df.head())
```

Output:

```
= RESTART: C:\Users\Neeraj\Desktop\Tycs\sem 6\Data Science\practical\practical no 3 a.py
Original DataFrame:
   classlabel  Alcohol  Malic Acid
0           1    14.23        1.71
1           1    13.20        1.78
2           1    13.16        2.36
3           1    14.37        1.95
4           1    13.24        2.59

DataFrame after Min-Max Scaling:
   classlabel   Alcohol  Malic Acid
0           1  0.884298    0.000000
1           1  0.033058    0.079545
2           1  0.000000    0.738636
3           1  1.000000    0.272727
4           1  0.066116    1.000000

DataFrame after Standard Scaling:
   classlabel   Alcohol  Malic Acid
0           1  1.089979   -1.078368
1           1 -0.812866   -0.873243
2           1 -0.886763    0.826358
3           1  1.348618   -0.375084
4           1 -0.738969    1.500338
```

B. **Perform feature dummification to convert categorical variables into numerical representations**.

Code:

```
#Perform feature dummification to convert categorical variables into numerical
representations.
import pandas as pd
from sklearn.preprocessing import LabelEncoder
# Load dataset
iris = pd.read_csv("Iris.csv")
# Check column names
print("Columns in dataset:", iris.columns)
# Encode the 'target' column instead of 'Species'
le = LabelEncoder()
iris['code'] = le.fit_transform(iris['target'])
# Print updated DataFrame
print(iris.head())
```

Output:

```
= RESTART: C:\Users\Neeraj\Desktop\Tycs\sem 6\Data Science\practical\practical no 3 b.py
Columns in dataset: Index(['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
       'petal width (cm)', 'target'],
      dtype='object')
   sepal length (cm)  sepal width (cm)  ...       target  code
0                5.1               3.5  ...  Iris-setosa     0
1                4.9               3.0  ...  Iris-setosa     0
2                4.7               3.2  ...  Iris-setosa     0
3                4.6               3.1  ...  Iris-setosa     0
4                5.0               3.6  ...  Iris-setosa     0

[5 rows x 6 columns]
```

PRACTICAL 4

Aim: Hypothesis Testing

A. **Formulate null and alternative hypotheses for a given problem.**
B. **Conduct a hypothesis test using appropriate statistical tests (e.g., t-test, chisquare test).**
C. **Interpret the results and draw conclusions based on the test outcomes.**

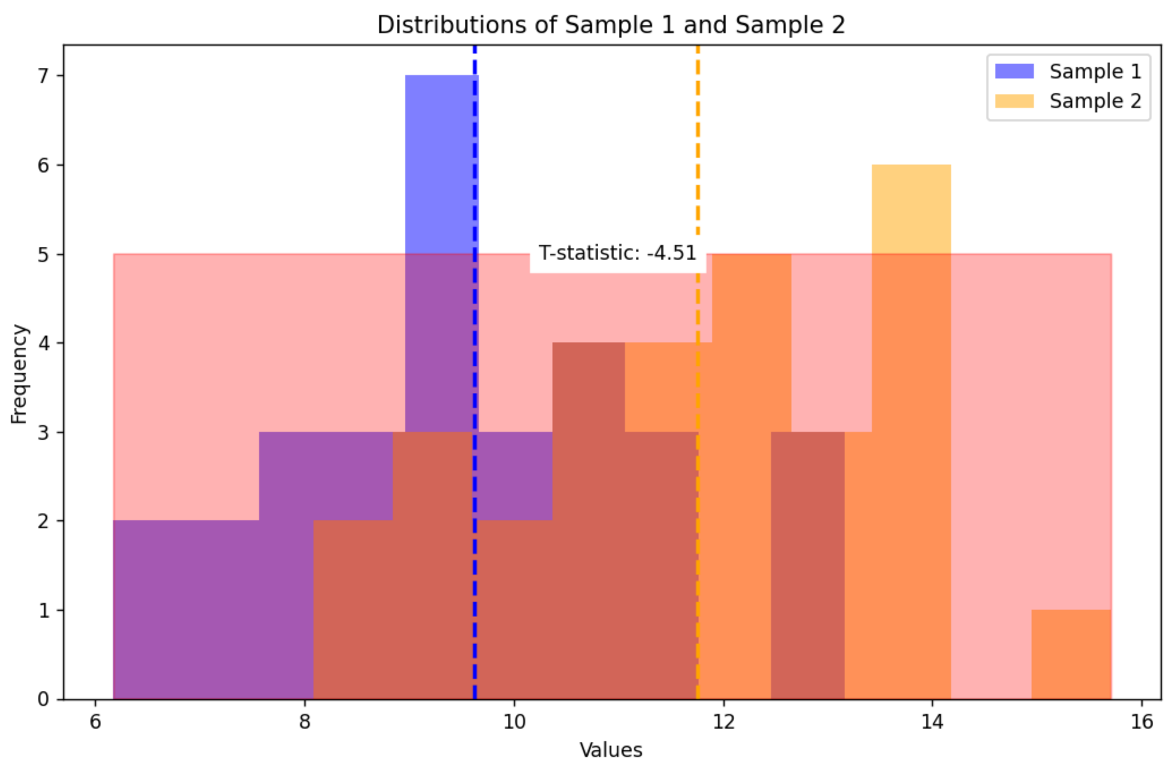
Code:

```
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 the results of the two-sample t-test
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 of the two samples
plt.figure(figsize=(10, 6))
plt.hist(sample1, alpha=0.5, label='Sample 1', color='blue', bins=10)
plt.hist(sample2, alpha=0.5, label='Sample 2', color='orange', bins=10)
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 the 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: C:\Users\Neeraj\Desktop\Tycs\sem 6\Data Science\practical\practical no 4.py
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.pyplot as plt
import seaborn as sb
import warnings
from scipy import stats
# Ignore warnings
warnings.filterwarnings('ignore')
# Load the dataset
df = sb.load_dataset('mpg')
# Display the dataframe and describe columns
print(df)
print(df['horsepower'].describe())
print(df['model_year'].describe())
# Define bins and categorize 'horsepower'
bins = [0, 75, 150, 240]
df['horsepower_new'] = pd.cut(df['horsepower'], bins=bins, labels=['l', 'm', 'h'])
# Display the new 'horsepower' categories
c = df['horsepower_new']
print(c)
# Define bins for 'model_year' and categorize
ybins = [69, 72, 74, 84]
labels = ['t1', 't2', 't3']
df['modelyear_new'] = pd.cut(df['model_year'], bins=ybins, labels=labels)
# Display the new 'model_year' categories
newyear = df['modelyear_new']
print(newyear)
# Perform chi-squared test of independence
df_chi = pd.crosstab(df['horsepower_new'], df['modelyear_new'])
print(df_chi)
# Perform chi-squared test
chi2_stat, p_value, dof, expected = stats.chi2_contingency(df_chi)
print(f"Chi2 Statistic: {chi2_stat}")
print(f"P-value: {p_value}")
print(f"Degrees of Freedom: {dof}")
print(f"Expected Frequencies: \n{expected}")
```

```
= RESTART: C:\Users\Neeraj\Desktop\Tycs\sem 6\Data Science\horsepower.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
Chi2 Statistic: 54.95485392447537
P-value: 3.320518009555984e-11
Degrees of Freedom: 4
Expected Frequencies:
[[ 21.21428571  16.66836735  61.11734694]
 [ 53.14285714  41.75510204 153.10204082]
 [  9.64285714   7.57653061  27.78061224]]
```

PRACTICAL 5

Aim: ANOVA (Analysis of Variance)

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

Code:

```
import pandas as pd
import scipy.stats as stats
from statsmodels.stats.multicomp import pairwise_tukeyhsd
# Data for each group
group1 = [23, 25, 29, 34, 30]
group2 = [19, 20, 22, 24, 25]
group3 = [15, 18, 20, 21, 17]
group4 = [28, 24, 26, 30, 29]
# Combine data into a DataFrame
data = pd.DataFrame({
    'value': group1 + group2 + group3 + group4,
    'group': ['Group1'] * len(group1) + ['Group2'] * len(group2) + ['Group3'] * len(group3) + ['Group4']
* len(group4)
})
# Perform one-way ANOVA
f_statistics, p_value = stats.f_oneway(group1, group2, group3, group4)
print("One-way ANOVA:")
print("F-statistic:", f_statistics)
print("p-value:", p_value)
# Perform Tukey-Kramer post-hoc test if ANOVA is significant
if p_value < 0.05:
    tukey_results = pairwise_tukeyhsd(data['value'], data['group'])
    print("\nTukey-Kramer post-hoc test:")
    print(tukey_results)
else:
    print("\nNo significant difference found in ANOVA, so Tukey-Kramer test is not performed.")
```

Output:

```
= RESTART: C:\Users\Neeraj\Desktop\Tycs\sem 6\Data Science\practical\practical no 5.py
One-way ANOVA:
F-statistic: 12.139872842870115
p-value: 0.00021465200901629603

Tukey-Kramer post-hoc test:
 Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
group1 group2 meandiff p-adj   lower    upper  reject
-----------------------------------------------------
Group1 Group2     -6.2  0.024 -11.6809 -0.7191   True
Group1 Group3    -10.0 0.0004 -15.4809 -4.5191   True
Group1 Group4     -0.8 0.9747  -6.2809  4.6809  False
Group2 Group3     -3.8 0.2348  -9.2809  1.6809  False
Group2 Group4      5.4 0.0542  -0.0809 10.8809  False
Group3 Group4      9.2  0.001   3.7191 14.6809   True
-----------------------------------------------------
```

PRACTICAL 6

Aim: Regression and Its Types

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

Code:

```
import numpy as np
import pandas as pd
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
# Load California housing dataset
print("Loading the California housing dataset...")
housing = fetch_california_housing()
housing_df = pd.DataFrame(housing.data, columns=housing.feature_names)
housing_df['PRICE'] = housing.target
# Check first few rows of the dataframe to make sure data is loaded
print(housing_df.head())
# Define features (X) and target (y)
X = housing_df[['AveRooms']]
y = housing_df['PRICE']
# Split the data into training and testing sets
print("Splitting the data into training and testing sets...")
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Train the model
print("Training the model...")
model = LinearRegression()
model.fit(X_train, y_train)
print("Model training complete.")
# Make predictions and evaluate the model
print("Making predictions and evaluating the model...")
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
# Output the results
print("Mean Squared Error:", mse)
print("R-squared:", r2)
print("Intercept:", model.intercept_)
print("Coefficient:", model.coef_)
```

Output:

```
= RESTART: C:\Users\Neeraj\Desktop\Tycs\sem 6\Data Science\practical\practical no 6.py
Loading the California housing dataset...
   MedInc  HouseAge  AveRooms  AveBedrms  ...  AveOccup  Latitude  Longitude  PRICE
0  8.3252      41.0  6.984127   1.023810  ...  2.555556     37.88    -122.23  4.526
1  8.3014      21.0  6.238137   0.971880  ...  2.109842     37.86    -122.22  3.585
2  7.2574      52.0  8.288136   1.073446  ...  2.802260     37.85    -122.24  3.521
3  5.6431      52.0  5.817352   1.073059  ...  2.547945     37.85    -122.25  3.413
4  3.8462      52.0  6.281853   1.081081  ...  2.181467     37.85    -122.25  3.422

[5 rows x 9 columns]
Splitting the data into training and testing sets...
Training the model...
Model training complete.
Making predictions and evaluating the model...
Mean Squared Error: 1.2923314440807299
R-squared: 0.013795337532284901
Intercept: 1.654762268596842
Coefficient: [0.07675559]
```

- **Extend the analysis to multiple linear regression and assess the impact of additional predictors.**

<u>Code:</u>

```
import numpy as np
import pandas as pd
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
# Load California housing dataset
housing = fetch_california_housing()
housing_df = pd.DataFrame(housing.data, columns=housing.feature_names)
housing_df['PRICE'] = housing.target
# Define features (X) and target (y)
X = housing_df.drop('PRICE', axis=1)  # Drop 'PRICE' column from the features
y = housing_df['PRICE']  # 'PRICE' is the target variable
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)
# Make predictions
y_pred = model.predict(X_test)
# Calculate evaluation metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
# Output the results
print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

print("Intercept:", model.intercept_)
print("Coefficients:", model.coef_)

Output:

```
= RESTART: C:/Users/Neeraj/Desktop/Tycs/sem 6/Data Science/practical/6 b.py
Mean Squared Error: 0.555891598695244
R-squared: 0.5757877060324511
Intercept: -37.023277706064064
Coefficients: [ 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 7

Aim: 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 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.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, classification_report
# Load the Iris dataset and create a binary classification problem
iris = load_iris()
iris_df = pd.DataFrame(data=np.c_[iris['data'], iris['target']],
                    columns=iris['feature_names'] + ['target'])
# Filter the dataset to create a binary classification problem (target != 2)
binary_df = iris_df[iris_df['target'] != 2]
# Define features (X) and target (y)
X = binary_df.drop('target', axis=1)
y = binary_df['target']
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Train a logistic regression model and evaluate its performance
logistic_model = LogisticRegression()
logistic_model.fit(X_train, y_train)
y_pred_logistic = logistic_model.predict(X_test)
print("Logistic Regression Metrics")
print("Accuracy: ", accuracy_score(y_test, y_pred_logistic))
print("Precision:", precision_score(y_test, y_pred_logistic))
print("Recall: ", recall_score(y_test, y_pred_logistic))
print("\nClassification Report:")
print(classification_report(y_test, y_pred_logistic))
# Train a decision tree model and evaluate its performance
decision_tree_model = DecisionTreeClassifier()
decision_tree_model.fit(X_train, y_train)
y_pred_tree = decision_tree_model.predict(X_test)
print("\nDecision Tree Metrics")
print("Accuracy: ", accuracy_score(y_test, y_pred_tree))
print("Precision:", precision_score(y_test, y_pred_tree))
print("Recall: ", recall_score(y_test, y_pred_tree))
```

```python
print("\nClassification Report:")
print(classification_report(y_test, y_pred_tree))
```

Output:

```
= RESTART: C:/Users/Neeraj/Desktop/Tycs/sem 6/Data Science/practical/practical no 7.py
Logistic Regression Metrics
Accuracy:  1.0
Precision: 1.0
Recall:  1.0

Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        12
         1.0       1.00      1.00      1.00         8

    accuracy                           1.00        20
   macro avg       1.00      1.00      1.00        20
weighted avg       1.00      1.00      1.00        20


Decision Tree Metrics
Accuracy:  1.0
Precision: 1.0
Recall:  1.0

Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        12
         1.0       1.00      1.00      1.00         8

    accuracy                           1.00        20
   macro avg       1.00      1.00      1.00        20
weighted avg       1.00      1.00      1.00        20
```

PRACTICAL 8

Aim: K-Means Clustering

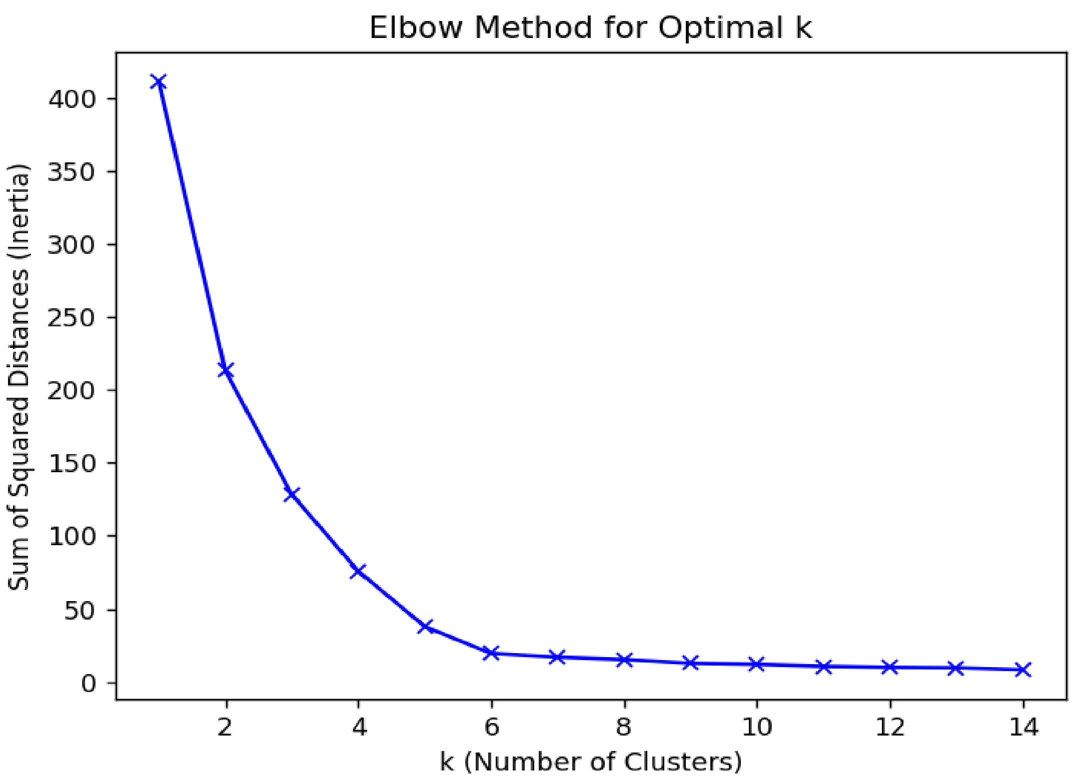
- Apply the K-Means algorithm to group similar data points into clusters.
- Determine the optimal number of clusters using elbow method or silhouette analysis.
- Visualize the clustering results and analyze the cluster characteristics.

Code:

```
import pandas as pd

from sklearn.preprocessing import MinMaxScaler

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

# Load the dataset

data = pd.read_csv("Wholesale.csv")

# Display the first few rows of the data

print(data.head())

# Define categorical and continuous features

categorical_features = ['Channel', 'Region']

continuous_features = ['Fresh', 'Milk', 'Grocery', 'Frozen', 'Detergents_Paper', 'Delicassen']

# Display summary statistics of the continuous features

print(data[continuous_features].describe())

# One-hot encoding for categorical features

for col in categorical_features:

    dummies = pd.get_dummies(data[col], prefix=col)

    data = pd.concat([data, dummies], axis=1)

    data.drop(col, axis=1, inplace=True)

# Display the data after encoding

print(data.head())

# Apply Min-Max Scaling to the data

mms = MinMaxScaler()

data_transformed = mms.fit_transform(data)

# Elbow method to determine the optimal number of clusters
```

```
sum_of_squared_distances = []
K = range(1, 15)
for k in K:
    km = KMeans(n_clusters=k)
    km.fit(data_transformed)
    sum_of_squared_distances.append(km.inertia_)
# Plotting the elbow graph
plt.plot(K, sum_of_squared_distances, 'bx-')
plt.xlabel('k (Number of Clusters)')
plt.ylabel('Sum of Squared Distances (Inertia)')
plt.title('Elbow Method for Optimal k')
plt.show()
```

Output:



PRACTICAL 9

Aim: Principal Component Analysis (PCA)

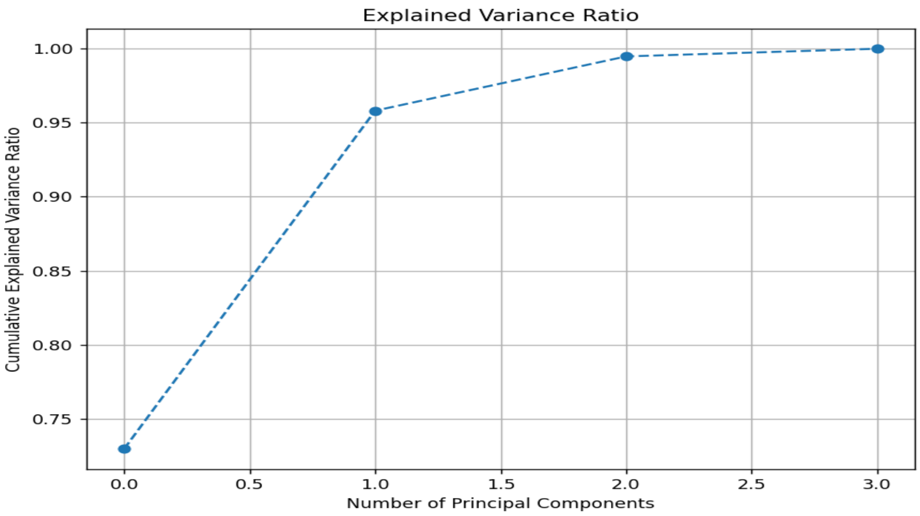
- Perform PCA on a dataset to reduce dimensionality.
- Evaluate the explained variance and select the appropriate number of principal components.
- Visualize the data in the reduced-dimensional space.

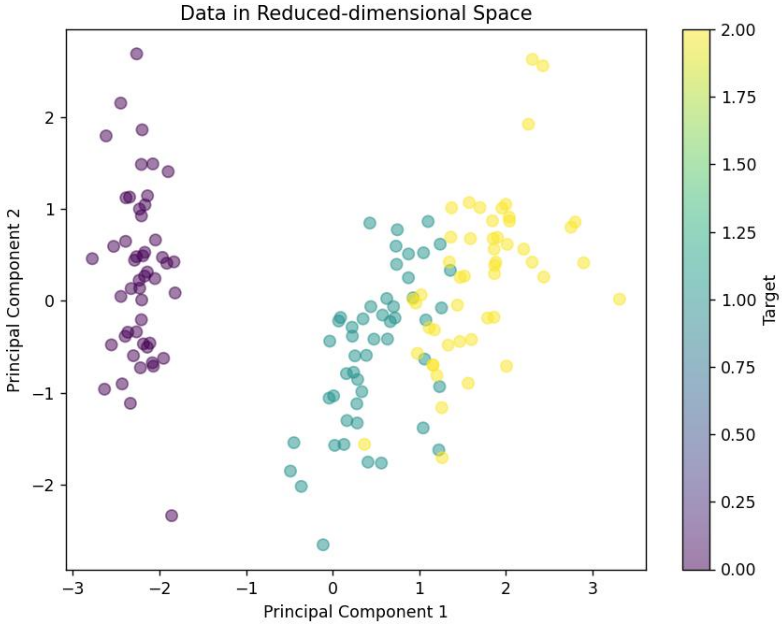
Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
# Load the Iris dataset
iris = load_iris()
iris_df = pd.DataFrame(data=np.c_[iris['data'], iris['target']],
                    columns=iris['feature_names'] + ['target'])
# Features and target variables
X = iris_df.drop('target', axis=1)
y = iris_df['target']
# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Apply PCA
pca = PCA()
X_pca = pca.fit_transform(X_scaled)
# Explained variance ratio
explained_variance_ratio = pca.explained_variance_ratio_
# Plot the cumulative explained variance ratio
plt.figure(figsize=(8, 6))
plt.plot(np.cumsum(explained_variance_ratio), marker='o', linestyle='--')
plt.title('Explained Variance Ratio')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance Ratio')
plt.grid(True)
plt.show()
# Find the number of components that explain 95% of the variance
cumulative_variance_ratio = np.cumsum(explained_variance_ratio)
n_components = np.argmax(cumulative_variance_ratio >= 0.95) + 1
print(f"Number of principal components to explain 95% variance: {n_components}")
# Reduce the data to the chosen number of components
pca = PCA(n_components=n_components)
X_reduced = pca.fit_transform(X_scaled)
# Plot the reduced data in 2D space
```

```
plt.figure(figsize=(8, 6))
plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=y, cmap='viridis', s=50, alpha=0.5)
plt.title('Data in Reduced-dimensional Space')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.colorbar(label='Target')
plt.show()
```

Output:






```
= RESTART: C:/Users/Neeraj/Desktop/Tycs/sem 6/Data Science/practical/practical no 9.py
Number of principal components to explain 95% variance: 2
```

PRACTICAL 10

Aim: Data Visualization and Storytelling

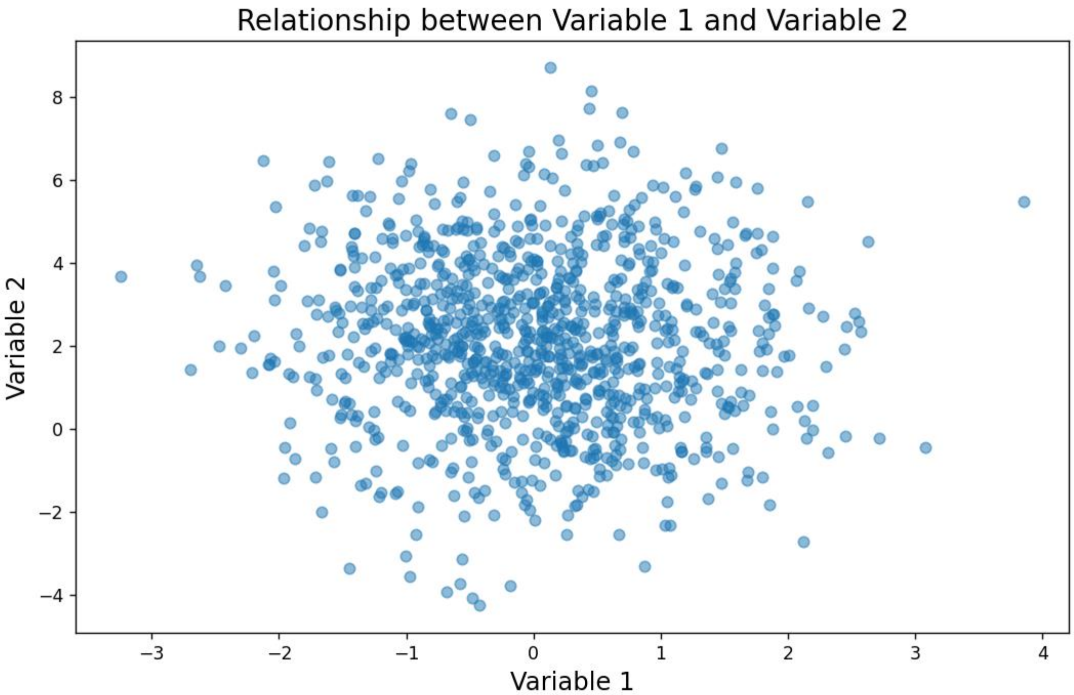
- Create meaningful visualizations using data visualization tools
- Combine multiple visualizations to tell a compelling data story.
- Present the findings and insights in a clear and concise manner.

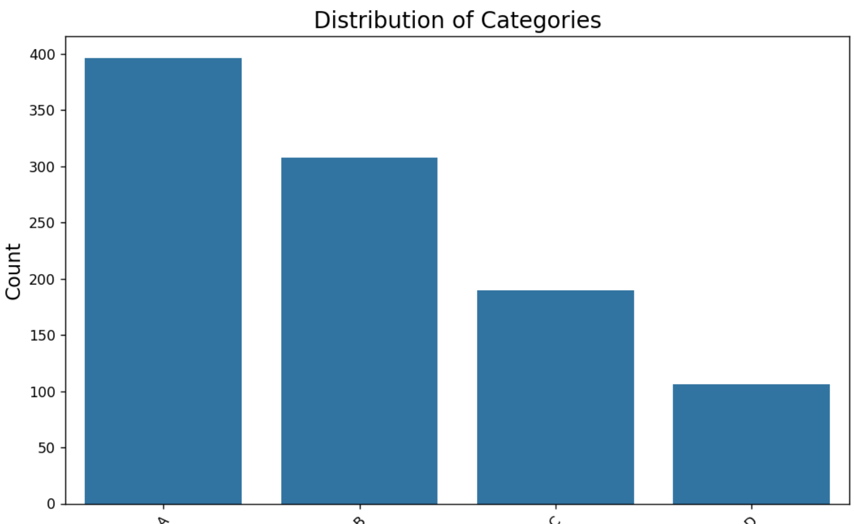
Code:

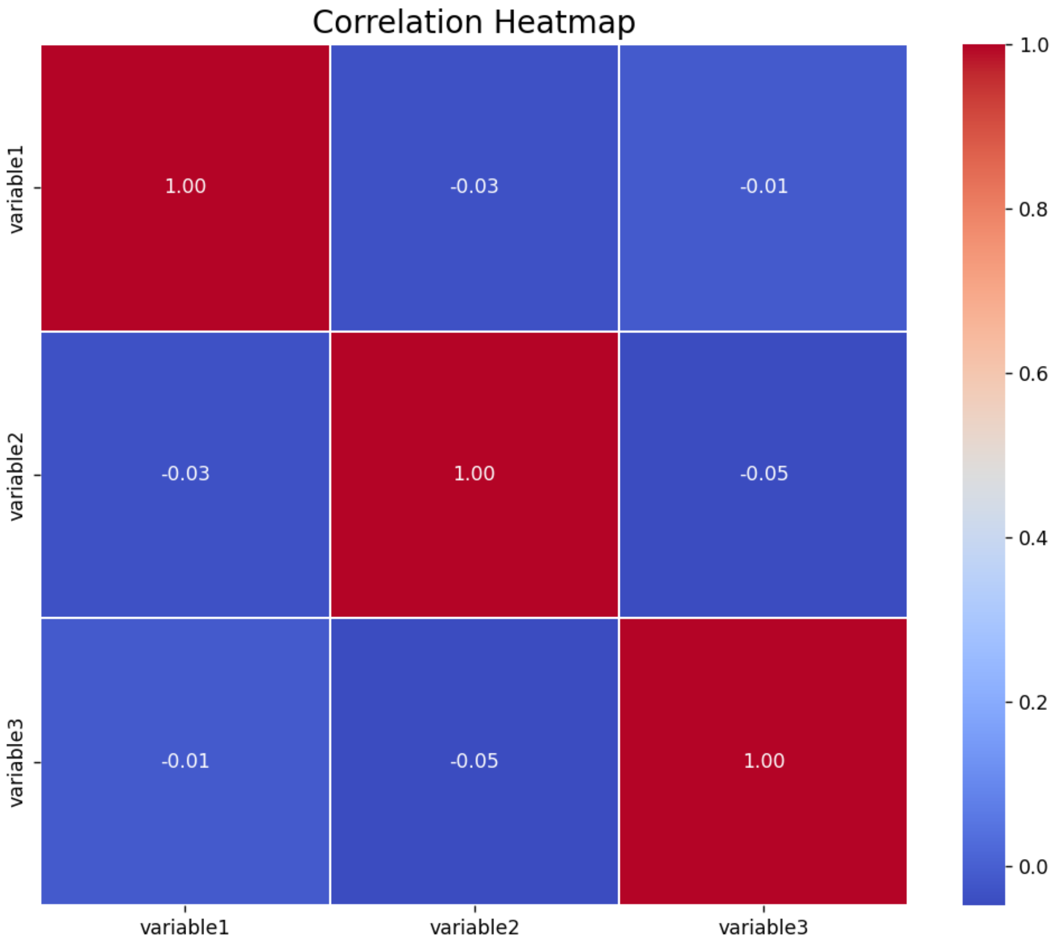
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
# Generate random data
np.random.seed(42)  # Set a seed for reproducibility
# Create a DataFrame with random data
data = pd.DataFrame({
   'variable1': np.random.normal(0, 1, 1000),
   'variable2': np.random.normal(2, 2, 1000) + 0.5 * np.random.normal(0, 1, 1000),
   'variable3': np.random.normal(-1, 1.5, 1000),
   'category': pd.Series(np.random.choice(['A', 'B', 'C', 'D'], size=1000, p=[0.4, 0.3, 0.2, 0.1]),
dtype='category')
})
# Create a scatter plot to visualize the relationship between two variables
plt.figure(figsize=(10, 6))
plt.scatter(data['variable1'], data['variable2'], alpha=0.5)
plt.title('Relationship between Variable 1 and Variable 2', fontsize=16)
plt.xlabel('Variable 1', fontsize=14)
plt.ylabel('Variable 2', fontsize=14)
plt.show()
# Create a bar chart to visualize the distribution of a categorical variable
plt.figure(figsize=(10, 6))
sns.countplot(x='category', data=data)
plt.title('Distribution of Categories', fontsize=16)
plt.xlabel('Category', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.xticks(rotation=45)
plt.show()
# Create a heatmap to visualize the correlation between numerical variables
plt.figure(figsize=(10, 8))
numerical_cols = ['variable1', 'variable2', 'variable3']
```

```
sns.heatmap(data[numerical_cols].corr(), annot=True, cmap='coolwarm', fmt='.2f', linewidths=1)
plt.title('Correlation Heatmap', fontsize=16)
plt.show()
# Data Storytelling
print("Title: Exploring the Relationship between Variable 1 and Variable 2\n")
print("The scatter plot (Figure 1) shows the relationship between Variable 1 and Variable 2.")
print("\nFigure 1: Scatter Plot of Variable 1 and Variable 2")
print("\nTo better understand the distribution of the categorical variable 'category', we created a bar chart.")
print("\nFigure 2: Distribution of Categories")
print("\nAdditionally, we explored the correlation between numerical variables using a heatmap.")
print("\nFigure 3: Correlation Heatmap")
print("\nIn summary, the visualizations and analysis provide insights into the relationships between the variables, such as the correlation between the numerical variables and the distribution of categories.")
```

Output:








```
= RESTART: C:/Users/Neeraj/Desktop/Tycs/sem 6/Data Science/practical/practical no 10.py
Title: Exploring the Relationship between Variable 1 and Variable 2

The scatter plot (Figure 1) shows the relationship between Variable 1 and Variable 2.

Figure 1: Scatter Plot of Variable 1 and Variable 2

To better understand the distribution of the categorical variable 'category', we created a bar chart.

Figure 2: Distribution of Categories

Additionally, we explored the correlation between numerical variables using a heatmap.

Figure 3: Correlation Heatmap

In summary, the visualizations and analysis provide insights into the relationships between the variables, such as the correlation between the numerical variables and the distribution
of categories.
```