

Practical 1

A. Introduction to Excel.

Open excel

open dataset.csv

The screenshot shows a Microsoft Excel spreadsheet titled "data.csv (1).xlsx". The data is organized into columns: "product", "category", "sales", and "cost". The "sales" column contains values 1000, 1500, 2000, 3456, and 7890. The "cost" column contains values 1234, 567, 345, 632, and 123. The "category" column includes entries like "Electronics", "clothing", and "home appliances". The "product" column has entries A, B, C, D, and E. The Excel ribbon at the top shows the Home tab selected. The status bar at the bottom indicates the file is saved to the PC and shows the date and time as 29-04-2024 06:20 PM.

Select all

Then go to conditional formatting

Then select Greater than

The screenshot shows the Microsoft Excel ribbon with the Conditional Formatting icon selected. A dropdown menu is open, showing various options under "Highlight Cells Rules". The "Greater Than..." option is highlighted. The main Excel window displays the same dataset as before. The status bar at the bottom shows the file is saved to the PC and the date and time as 29-04-2024 06:22 PM.

Enter the greater than filters value for example 2000

Go to data bars

Then select solid fill in conditional formatting

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

Select entire table and go to insert tab PivotChart

Then select New Worksheet in the create pivot chart window

The screenshot shows a Microsoft Excel spreadsheet titled 'data.csv (1).xlsx'. The 'Insert' tab is active. In the 'Charts' group, the 'PivotChart' icon is highlighted. Below the ribbon, a data range 'product' is selected, corresponding to the following table:

product	category	sales	cost
A	Electronics	1000	1234
B	clothing	1500	567
C	Electronics	2000	345
D	home appl	3456	632
E	clothing	7890	123

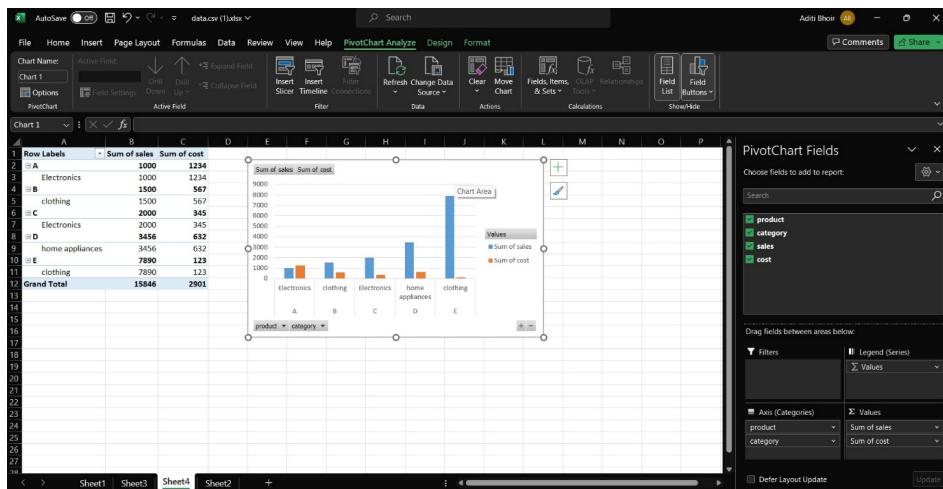
An 'Insert' dialog box is open, titled 'Create PivotChart'. It contains the following settings:

- Choose the data that you want to analyze:
 - Select a table or range: Sheet1!\$D\$8:\$G\$13
 - Use an external data source: Choose Connection...
 - Use this workbook's Data Model: Connection name: [empty]
- Choose where you want the PivotChart to be placed:
 - New Worksheet
 - Existing Worksheet: Location: [empty]
- Choose whether you want to analyze multiple tables:
 - Add this data to the Data Model

At the bottom right of the dialog are 'OK' and 'Cancel' buttons.

Then select and drag the attributes in the below boxes

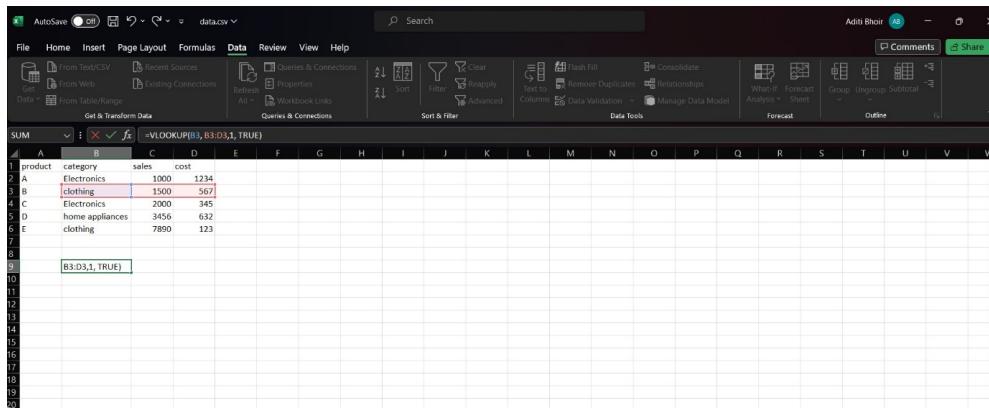
The screenshot shows the Microsoft Excel ribbon with the 'PivotChart Analyze' tab selected. The main area displays a blank PivotChart canvas with a placeholder message: 'To build a PivotChart, choose fields from the PivotChart Field List.' To the left of the canvas is a 'PivotTable Field List' icon. On the right side of the screen is the 'PivotChart Fields' pane, which lists the available fields: product, category, sales, and cost. Below this is the 'Fields, Items, & Sets' pane, which currently shows 'Axis (Categories)' and 'Values' under their respective sections. At the bottom of the screen, the ribbon tabs are visible, including 'File', 'Home', 'Insert', 'Page Layout', 'Formulas', 'Data', 'Review', 'View', 'PivotChart Analyze', 'Design', and 'Format'. The status bar at the bottom shows 'Sheet1 | Sheet3 | Sheet4 | Sheet2 | +', 'Defer Layout Update', and 'Update'.



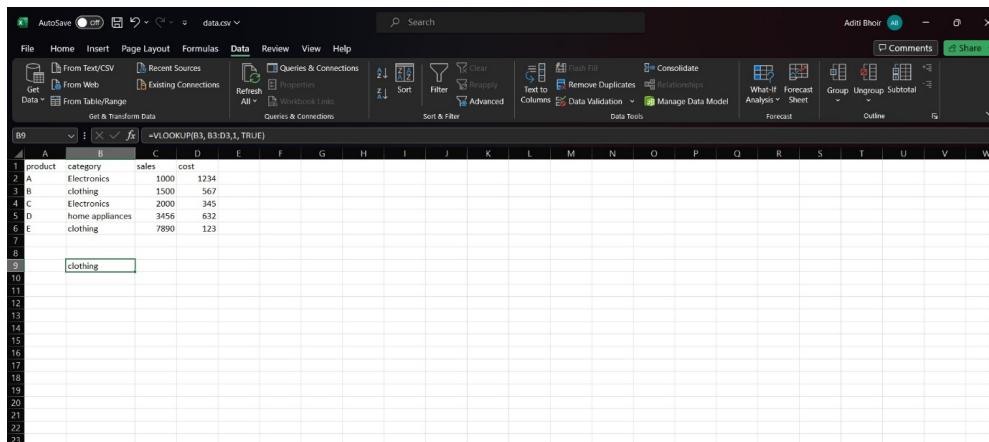
C. Use VLOOKUP function to retrieve information from a different worksheet or table

Step 1: click on an empty cell and type the following command in the command line

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



Click enter button



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

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

The screenshot shows a Microsoft Excel spreadsheet titled "data.csv". The Data tab is selected in the ribbon. In the "Data Tools" section of the ribbon, the "What-If Analysis" dropdown is open, showing options like "Scenario Manager...", "Goal Seek...", and "Data Table...". A tooltip for "Goal Seek..." provides the description: "Find the right input for the value you want."

product	category	sales	cost	
2	A	Electronics	1000	1234
3	B	clothing	1500	567
4	C	Electronics	2000	345
5	D	home appliances	3456	632
6	E	clothing	7890	123

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

The screenshot shows the same Excel spreadsheet with the "Goal Seek" dialog box open. The "Set cell:" field is set to E6, "To value:" is 12000, and "By changing cell:" is D5. The "OK" button is highlighted.

product	category	sales	cost	total
2	A	Electronics	1000	1234
3	B	clothing	1500	567
4	C	Electronics	2000	345
5	D	home appl	3456	2345
6	E	clothing	7890	123

The screenshot shows the spreadsheet with the "Goal Seek Status" dialog box open. It displays the message: "Goal Seeking with Cell E6 may not have found a solution." Below it, it shows the "Target value: 12000" and "Current value: 8013". The "OK" button is highlighted.

product	category	sales	cost	total
2	A	Electronics	1000	1234
3	B	clothing	1500	567
4	C	Electronics	2000	345
5	D	home appl	3456	2345
6	E	clothing	7890	123

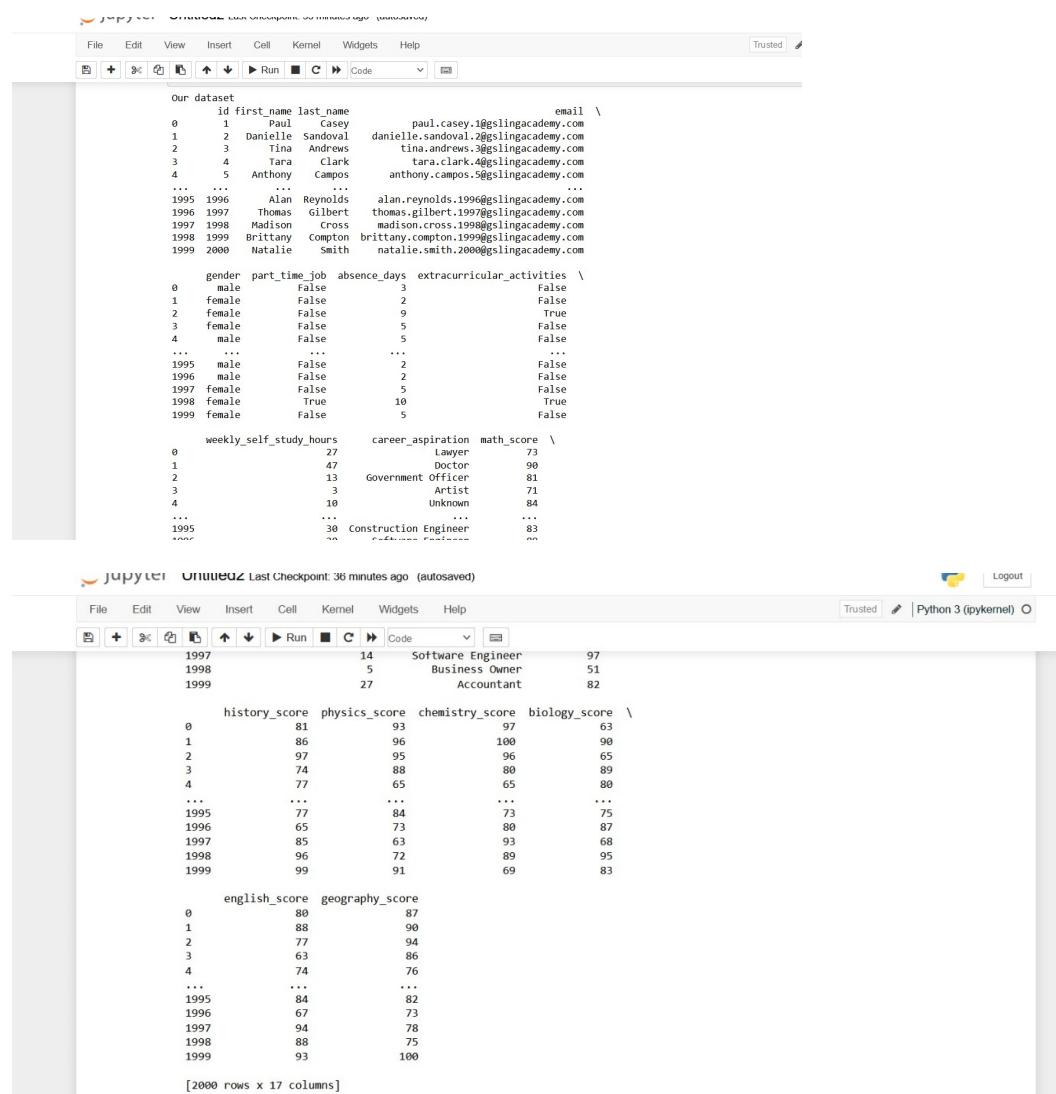
Practical 2

Aim : Data Frames and Basic Pre-Processing

A. Read data from CSV and JSON files into a Data Frame

1. Read data from a csv file

```
import pandas as pd
df = pd.read_csv('D:\data science\student-scores.csv')
print("Our dataset ")
print(df)
```



The screenshot shows a Jupyter Notebook interface with two code cells and their outputs.

Code Cell 1:

```
import pandas as pd
df = pd.read_csv('D:\data science\student-scores.csv')
print("Our dataset ")
print(df)
```

Output 1:

```
Our dataset
   id first_name last_name          email \
0     1    Paul      Casey paul.casey.1@gslingacademy.com
1     2  Danielle    Sandoval danielle.sandoval.2@gslingacademy.com
2     3     Tina    Andrews tina.andrews.3@gslingacademy.com
3     4     Tara     Clark  tara.clark.4@gslingacademy.com
4     5  Anthony    Campos anthony.campos.5@gslingacademy.com
...
1995 1996     Alan    Reynolds alan.reynolds.1996@gslingacademy.com
1996 1997     Thomas    Gilbert thomas.gilbert.1997@gslingacademy.com
1997 1998   Madison     Cross madison.cross.1998@gslingacademy.com
1998 1999  Brittany   Compton brittany.compton.1999@gslingacademy.com
1999 2000   Natalie     Smith natalie.smith.2000@gslingacademy.com

   gender part_time_job absence_days extracurricular_activities \
0    male           False            3                  False
1  female           False            2                  False
2  female           False            9                  True
3  female           False            5                  False
4    male           False            5                  False
...
1995    male           False            2                  False
1996    male           False            2                  False
1997  female           False            5                  False
1998  female            True            10                 True
1999  female           False            5                  False

   weekly_self_study_hours career_aspiration  math_score \
0                      27        Lawyer             73
1                      47        Doctor             98
2                      13  Government Officer            81
3                      3         Artist             71
4                      10       Unknown             84
...
1995                      30  Construction Engineer            83
1996                      28  Construction Engineer            83
```

Code Cell 2:

```
[2000 rows x 17 columns]
```

2. Reading data from a JSON file

```

import pandas as pd
data = pd.read_json(' D:\\\\data science\\\\ sales.json')
print(data)

```

```

>>> ===== RESTART: D:/Notes/sem-6/data science/p:
    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:

1. 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)

```

The screenshot shows two code cells in a Jupyter Notebook. The first cell contains the code to read the 'titanic.csv' file and print its head. The second cell contains the code to fill NA values with 0 and print the modified dataset. Both cells output the same data frame, which is identical to the original 'titanic.csv' dataset provided in the image above.

This screenshot shows two code cells in a Jupyter Notebook. The first cell contains the code to read the 'titanic.csv' file and print its head. The second cell contains the code to fill NA values with 0 and print the modified dataset. Both cells output the same data frame, which is identical to the original 'titanic.csv' dataset provided in the image above.

2. Dropping NA values using dropna()

```

import pandas as pd
df = pd.read_csv('titanic.csv')
print(df)
df.head(10)
print("Dataset after dropping NA values: ")
df.dropna(inplace = True)
print(df)

```

The screenshot shows two code cells in a Jupyter Notebook. The first cell contains the code provided above. The second cell displays the output:

```

[891 rows x 10 columns]
Dataset after dropping NA values:
   sex age sibsp parch fare embarked class who alone \
0 male 22.0 1 0 7.2500 S Third man False
1 female 38.0 1 0 71.2833 C First woman False
2 female 26.0 0 0 7.9250 S Third woman True
3 female 35.0 1 0 53.1000 S First woman False
4 male 35.0 0 0 8.0500 S Third man True
...
886 male 27.0 0 0 13.0000 S Second man True
887 female 30.0 0 0 36.0000 S First woman True
888 female Null 1 2 23.4500 S Third woman False
889 male 26.0 0 0 30.0000 C First man True
890 male 32.0 0 0 7.7500 Q Third man True

survived
0 0
1 1
2 1
3 1
4 0
...
886 0
887 1
888 0
889 1
890 0

[891 rows x 10 columns]
[712 rows x 10 columns]

```

The screenshot shows two code cells in a Jupyter Notebook. The first cell contains the code provided above. The second cell displays the output:

```

[891 rows x 10 columns]
Dataset after dropping NA values:
   sex age sibsp parch fare embarked class who alone \
0 male 22.0 1 0 7.2500 S Third man False
1 female 38.0 1 0 71.2833 C First woman False
2 female 26.0 0 0 7.9250 S Third woman True
3 female 35.0 1 0 53.1000 S First woman False
4 male 35.0 0 0 8.0500 S Third man True
...
885 female 39.0 0 5 29.1250 Q Third woman False
886 male 27.0 0 0 13.0000 S Second man True
887 female 19.0 0 0 30.0000 S First woman True
888 male 26.0 0 0 30.0000 C First man True
890 male 32.0 0 0 7.7500 Q Third man True

survived
0 0
1 1
2 1
3 1
4 0
...
885 0
886 0
887 1
888 1
890 0

[712 rows x 10 columns]

```

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

Code:

```

import pandas as pd
# Load iris dataset
iris = pd.read_csv('Iris.csv')

# Filtering data based on a condition
setosa = iris[iris['Species'] == 'setosa']
print("Setosa samples:")
print(setosa.head())

# Sorting data
sorted_iris = iris.sort_values(by='SepalLengthCm', ascending=False) print("\nSorted iris dataset:")
print(sorted_iris.head())

# Grouping data
grouped_species = iris.groupby('Species').mean()
print("\nMean measurements for each species:")
print(grouped_species)

```

Home Page - Select or create a X Untitled2 - Jupyter Notebook +

localhost:8888/notebooks/Untitled2.ipynb?kernel_name=python3

jupyter Untitled2 Last Checkpoint: 7 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel) Logout

Setosa samples:
Empty DataFrame
Columns: [Id, SepalLengthCm, SepalWidthCm, PetalLengthCm, PetalWidthCm, Species]
Index: []

Sorted iris dataset:
 Id SepalLengthCm SepalWidthCm PetalLengthCm PetalWidthCm \\\n131 132 7.9 3.8 6.4 2.0\n135 136 7.7 3.0 6.1 2.3\n122 123 7.7 2.8 6.7 2.0\n117 118 7.7 3.8 6.7 2.2\n118 119 7.7 2.6 6.9 2.3

Species
131 Iris-virginica
135 Iris-virginica
122 Iris-virginica
117 Iris-virginica
118 Iris-virginica

Mean measurements for each species:
 Id SepalLengthCm SepalWidthCm PetalLengthCm PetalWidthCm \\\nSpecies
Iris-setosa 25.5 5.006 3.418 1.464
Iris-versicolor 75.5 5.936 2.770 4.268
Iris-virginica 125.5 6.588 2.974 5.552

PetalWidthCm
Species
Iris-setosa 0.244
Iris-versicolor 1.326
Iris-virginica 2.026

Practical 3

Aim : Feature Scaling and Dummification

A. Apply feature-scaling techniques like standardisation and normalisation to numerical features.

Code:

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Read the dataset
df = pd.read_csv('D:\\data science\\wine.csv', header=None, usecols=[0, 1, 2], skiprows=1)
df.columns = ['classlabel', 'Alcohol', 'Malic Acid']

# Print the original DataFrame
print("Original DataFrame:")
print(df)

# Perform Min-Max scaling
scaling = MinMaxScaler()
scaled_value = scaling.fit_transform(df[['Alcohol','Malic Acid']])
df[['Alcohol','Malic Acid']] = scaled_value

# Print the DataFrame after Min-Max scaling
print("\nDataFrame after MinMax Scaling:")
print(df)

# Perform Standard scaling
scaling = StandardScaler()
scaled_standardvalue = scaling.fit_transform(df[['Alcohol','Malic Acid']])
df[['Alcohol','Malic Acid']] = scaled_standardvalue

# Print the DataFrame after Standard scaling
print("\nDataFrame after Standard Scaling:")
print(df)
```

The screenshot shows a Jupyter Notebook interface with the following content:

- 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
..
173	3	13.71	5.65
174	3	13.40	3.91
175	3	13.27	4.28
176	3	13.17	2.59
177	3	14.13	4.10

[178 rows x 3 columns]
- Dataframe after MinMax Scaling**

	classlabel	Alcohol	Malic Acid
0	1	0.842700	0.100000
1	1	0.705324	0.055324
2	1	0.560526	0.320158
3	1	0.878947	0.239139
4	1	0.581579	0.365613
..
173	3	0.705263	0.970356
174	3	0.664032	0.829392
175	3	0.580474	0.699685
176	3	0.562158	0.365613
177	3	0.815789	0.664032

[178 rows x 3 columns]
- Dataframe after Standard Scaling**

	classlabel	Alcohol	Malic Acid
0	1	1.518613	-0.562250

```
1 0.571053 0.205534
2 0.560526 0.320158
3 0.878947 0.239130
4 0.581579 0.365613
...
173 0.705263 0.979356
174 0.623684 0.626482
175 0.589474 0.699605
176 0.563158 0.365613
177 0.815789 0.664032
[178 rows x 3 columns]

Dataframe after Standard Scaling
   classlabel  Alcohol  Malic Acid
0          1  1.518613 -0.562250
1          1  0.246290 -0.499413
2          1  0.196879  0.021231
3          1  1.691550 -0.346811
4          1  0.295700  0.227694
...
173        3  0.876275  2.974543
174        3  0.493343  1.412669
175        3  0.332758  1.744744
176        3  0.209232  0.227694
177        3  1.395086  1.583165
[178 rows x 3 columns]
```

B. Perform Feature Dummification to convert categorical variable into numerical representation

Code:

```
import pandas as pd

# Read the Iris dataset
iris = pd.read_csv("D:\\data science\\Iris.csv")

# Print the original dataset
print("Original DataFrame:")
print(iris)

# Encode the 'Species' column using LabelEncoder
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
iris['code'] = le.fit_transform(iris.Species)

# Print the DataFrame after encoding
print("\nDataFrame after Label Encoding:")
print(iris)
```

Home Page - Select or create a Untitled2 - Jupyter Notebook +

localhost:8888/notebooks/Untitled2.ipynb?kernel_name=python3

jupyter Untitled2 Last Checkpoint: 23 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel) ○

[150 rows × 6 columns]

	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

Species

	Species
0	Iris-setosa
1	Iris-setosa
2	Iris-setosa
3	Iris-setosa
4	Iris-setosa
..	...
145	Iris-virginica
146	Iris-virginica
147	Iris-virginica
148	Iris-virginica
149	Iris-virginica

[150 rows × 6 columns]

	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

Home Page - Select or create a Untitled2 - Jupyter Notebook +

localhost:8888/notebooks/Untitled2.ipynb?kernel_name=python3

jupyter Untitled2 Last Checkpoint: 23 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel) ○

147 Iris-virginica
148 Iris-virginica
149 Iris-virginica

[150 rows × 6 columns]

	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

Species code

	Species	code
0	Iris-setosa	0
1	Iris-setosa	0
2	Iris-setosa	0
3	Iris-setosa	0
4	Iris-setosa	0
..
145	Iris-virginica	2
146	Iris-virginica	2
147	Iris-virginica	2
148	Iris-virginica	2
149	Iris-virginica	2

[150 rows × 7 columns]

Practical 4

Aim : 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}")

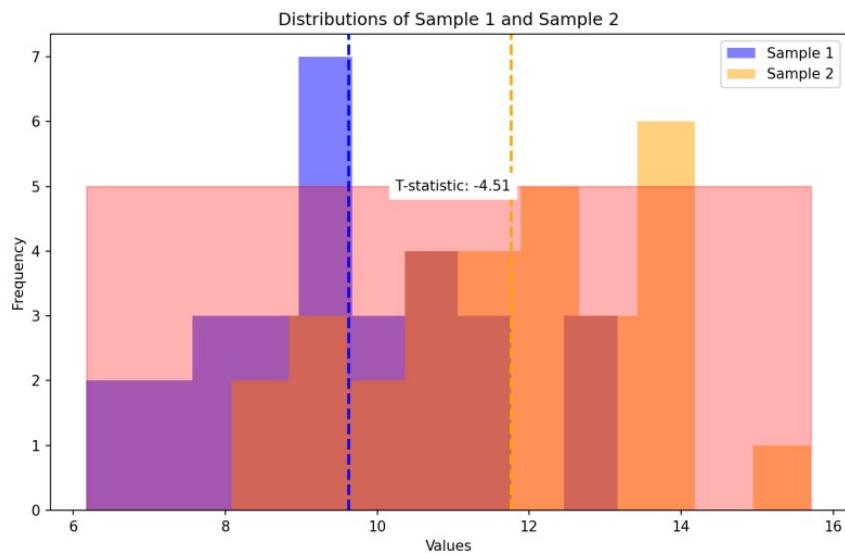
# Plot the distributions
plt.figure(figsize=(10, 6))
plt.hist(sample1, alpha=0.5, label='Sample 1', color='blue')
plt.hist(sample2, alpha=0.5, label='Sample 2', color='orange')
plt.axvline(np.mean(sample1), color='blue', linestyle='dashed', linewidth=2)
plt.axvline(np.mean(sample2), color='orange', linestyle='dashed', 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')

# 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 Sample2.")
    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 Sample1.")
else:
    print("Conclusion: Fail to reject the null hypothesis.")
    print("Interpretation: There is not enough evidence to claim a significant difference between the means.")

plt.show()
```

Results of Two-Sample t-test:
T-statistic: -4.512913234547555
P-value: 3.176506547470154e-05
Degrees of Freedom: 58



Chi-test. (Run in idle)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sb
import warnings
from scipy import stats
warnings.filterwarnings('ignore')

# Load the dataset
df = sb.load_dataset('mpg')

# Print the dataset
print(df)

# Print the summary statistics of 'horsepower' and 'model_year' columns
print(df['horsepower'].describe())
print(df['model_year'].describe())

# Create bins for 'horsepower' and 'model_year'
bins = [0, 75, 150, 240]
df['horsepower_new'] = pd.cut(df['horsepower'], bins=bins, labels=['l', 'm', 'h'])

ybins = [69, 72, 74, 84]
label = ['t1', 't2', 't3']
df['modelyear_new'] = pd.cut(df['model_year'], bins=ybins, labels=label)

# Print the new 'horsepower_new' and 'modelyear_new' columns
print(df['horsepower_new'])
print(df['modelyear_new'])

# Create a contingency table
df_chi = pd.crosstab(df['horsepower_new'], df['modelyear_new'])

# Perform the chi-square test of independence
chi2, p, dof, expected = stats.chi2_contingency(df_chi)
print(df_chi)
print("Chi-square statistic:", chi2)
print("P-value:", p)
print("Degrees of freedom:", dof)
print("Expected frequencies table:")
print(expected)
```

```

      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: 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
Chi2ContingencyResult(statistic=54.95485392447537, pvalue=3.320518009555984e-11, dof=4, expected_freq=array([[ 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

# Define the 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 all data into a single list
all_data = group1 + group2 + group3 + group4

# Create a list of group labels corresponding to each data point
group_labels = ['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-statistics:", f_statistics)
print("p-value:", p_value)

# Perform Tukey's HSD test
tukey_results = pairwise_tukeyhsd(all_data, group_labels)
print("\nTukey-Kramer post-hoc test:")
print(tukey_results)
```

Output :

```
one-way ANOVA:
F-statistics: 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.

##. Simple Linear Regression

```
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 the California housing dataset
housing = fetch_california_housing()
housing_df = pd.DataFrame(housing.data, columns=housing.feature_names)

# Add the target variable 'PRICE' to the DataFrame
housing_df['PRICE'] = housing.target

# Simple Linear Regression
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)

model = LinearRegression()
model.fit(X_train, y_train)

mse = mean_squared_error(y_test, model.predict(X_test))
r2 = r2_score(y_test, model.predict(X_test))
print("Simple Linear Regression:")
print("Mean Squared Error:", mse)
print("R-squared:", r2)
print("Intercept:", model.intercept_)
print("Coefficient:", model.coef_)
```

##. Multiple Linear Regression

```
# Multiple Linear Regression
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)

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("\nMultiple Linear Regression:")
print("Mean Squared Error:", mse)
print("R-squared:", r2)
print("Intercept:", model.intercept_)
print("Coefficients:")
for i, col in enumerate(X.columns):
    print(f"{col}: {model.coef_[i]}")
```

Output :

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

	Longitude
0	-122.23
1	-122.22
2	-122.24
3	-122.25
4	-122.25
...	...
20635	-121.09
20636	-121.21
20637	-121.22
20638	-121.32
20639	-121.24

[20640 rows x 8 columns]

```
[20640 rows x 8 columns]
Mean Squared Error: 1.2923314440807299
R-squared: 0.013795337532284901
Intercept: 1.654762268596842
Coefficient: [0.07675559]
Mean Squared Error: 0.5558915986952441
R-squared: 0.575787706032451
Intercept: -37.02327770606414
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 7

Aim : Logistic Regression and Decision Tree.

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'])
binary_df = iris_df[iris_df['target'] != 2]
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))
print("\nClassification Report")
print(classification_report(y_test, y_pred_tree))
```

Output :

jupyter Untitled2 (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipyk)

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.

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("/Users/krishnasingh/Desktop/csv files/wholesale.csv")

# Define categorical and continuous features
categorical_features = ['Channel', 'Region']
continuous_features = ['Fresh', 'Milk', 'Grocery', 'Frozen', 'Detergents_Paper', 'Delicassen']

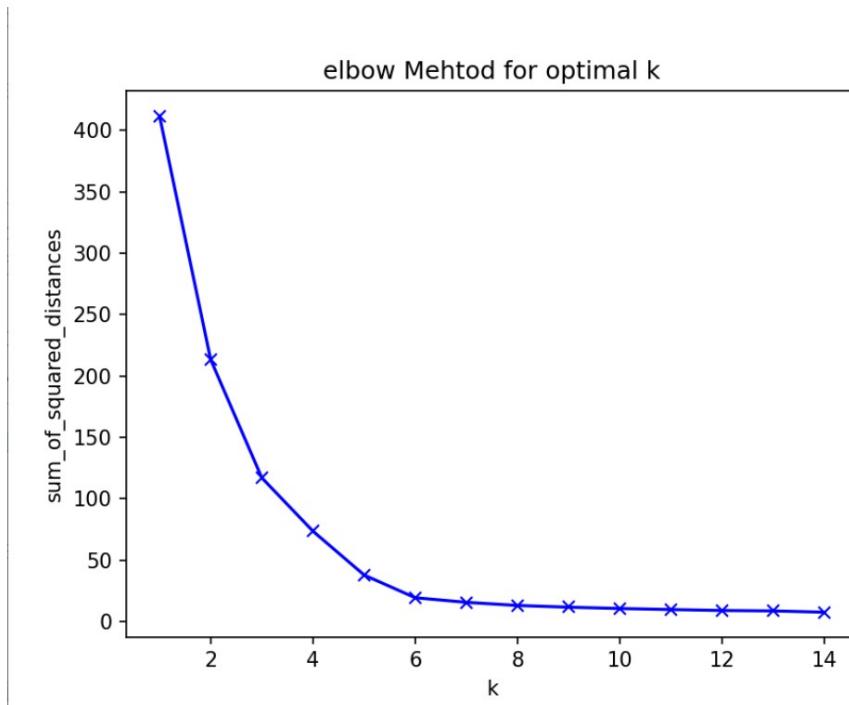
# Convert categorical features into dummy/indicator variables
for col in categorical_features:
    dummies = pd.get_dummies(data[col], prefix=col)
    data = pd.concat([data, dummies], axis=1)
    data.drop(col, axis=1, inplace=True)

# Scale the data
mms = MinMaxScaler()
data_transformed = mms.fit_transform(data)

# Find the optimal number of clusters using the elbow method
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_)

# Plot the elbow curve
plt.plot(K, sum_of_squared_distances, 'bx-')
plt.xlabel('k')
plt.ylabel('Sum of Squared Distances')
plt.title('Elbow Method for Optimal k')
plt.show()
```

Output :



Practical 9

Aim : Principal Component Analysis (PCA).

```
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'])
X = iris_df.drop('target', axis=1)
y = iris_df['target']

# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Perform PCA to transform the data into principal components
pca = PCA()
X_pca = pca.fit_transform(X_scaled)
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()

# Determine the number of principal components to explain 95% 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}")

# Perform PCA to transform the data into principal components
pca = PCA()
X_pca = pca.fit_transform(X_scaled)
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()

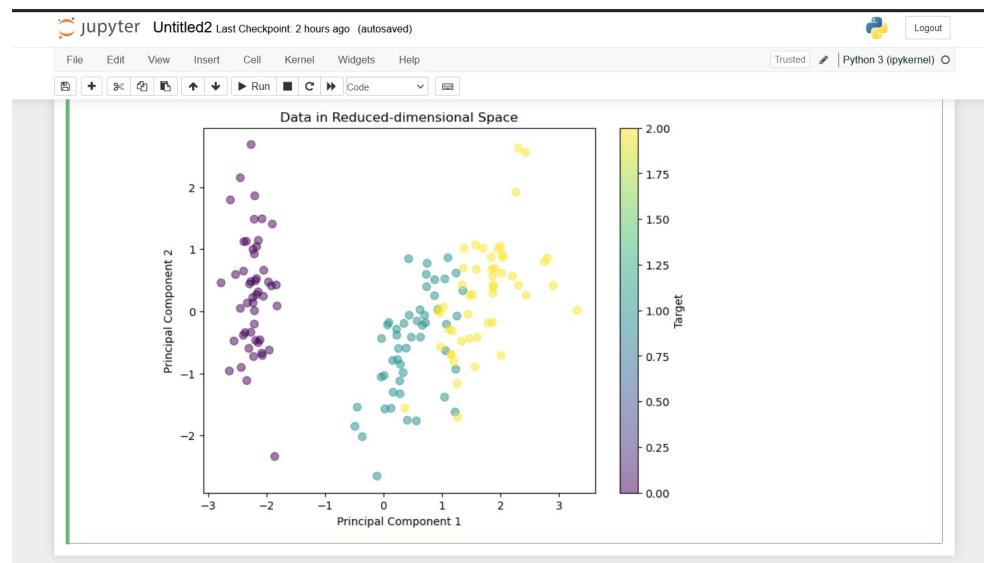
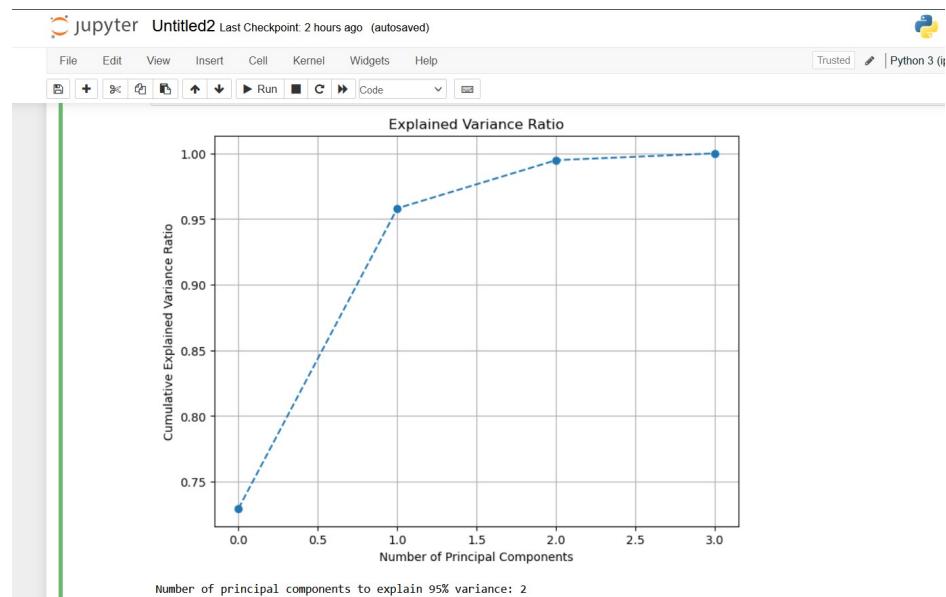
# Determine the number of principal components to explain 95% variance
cumulative_variance_ratio = np.cumsum(explained_variance_ratio)
n_components = np.argmax(cumulative_variance_ratio >= 0.95) + 1
print(f"Number of principal components to explain 95% variance: {n_components}")
```

Code :

```
# Perform PCA with the specified number of components
pca = PCA(n_components=n_components)
X_reduced = pca.fit_transform(X_scaled)

# Visualize the data in the reduced-dimensional 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 :



Practical 10

Aim : Data Visualisation and Storytelling.

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

```
# 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

```
# 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

```
# 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
```

```
# 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')
plt.title('Correlation Heatmap', fontsize=16)
plt.show()
```

Data Storytelling

```
print("Title: Exploring the Relationship between Variable 1 and Variable 2")
```

```
print("\nThe scatter plot (Figure 1) shows the relationship between Variable 1 and Variable 2. We can observe a positive correlation, indicating that as Variable 1 increases, Variable 2 tends to increase as well. However, there is a considerable amount of scatter, suggesting that other factors may influence this relationship.")
```

```
print("\nScatter Plot")
```

```
print("Figure 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 (Figure 2). The chart reveals that Category A has the highest frequency, followed by Category B, Category C, and Category D. This information could be useful for further analysis or decision-making processes.")
```

```
print("\nBar Chart")
```

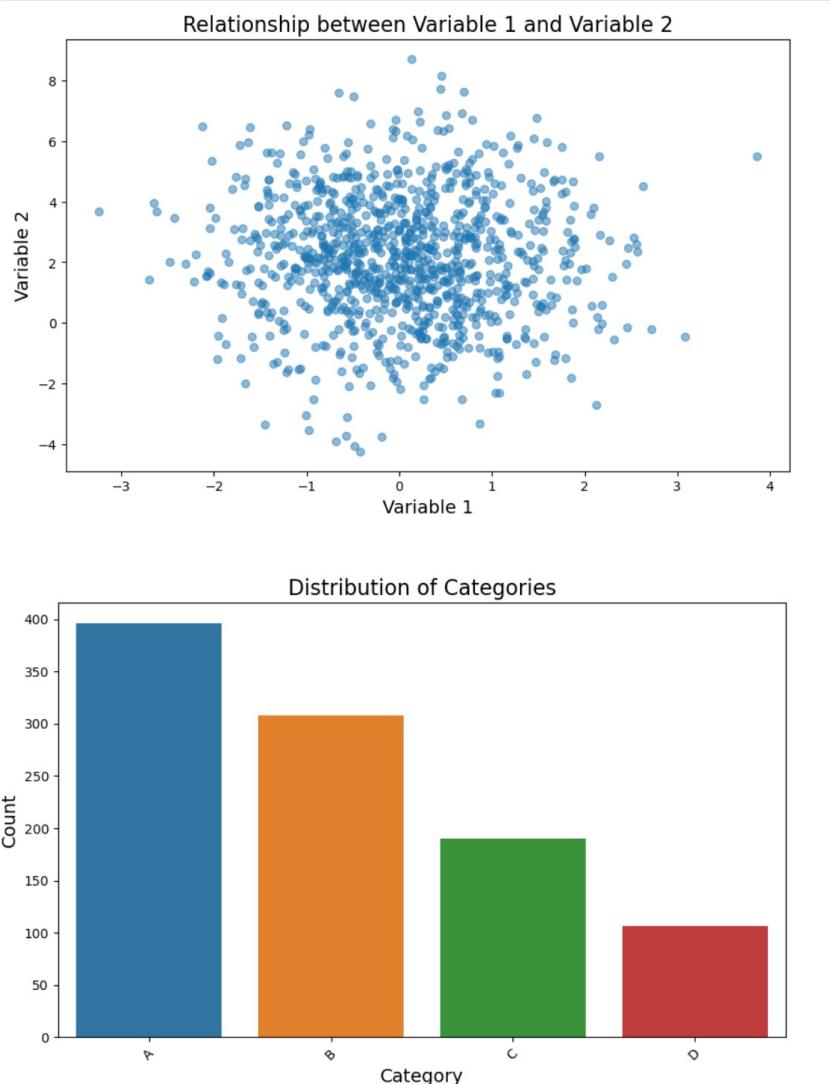
```
print("Figure 2: Distribution of Categories")
```

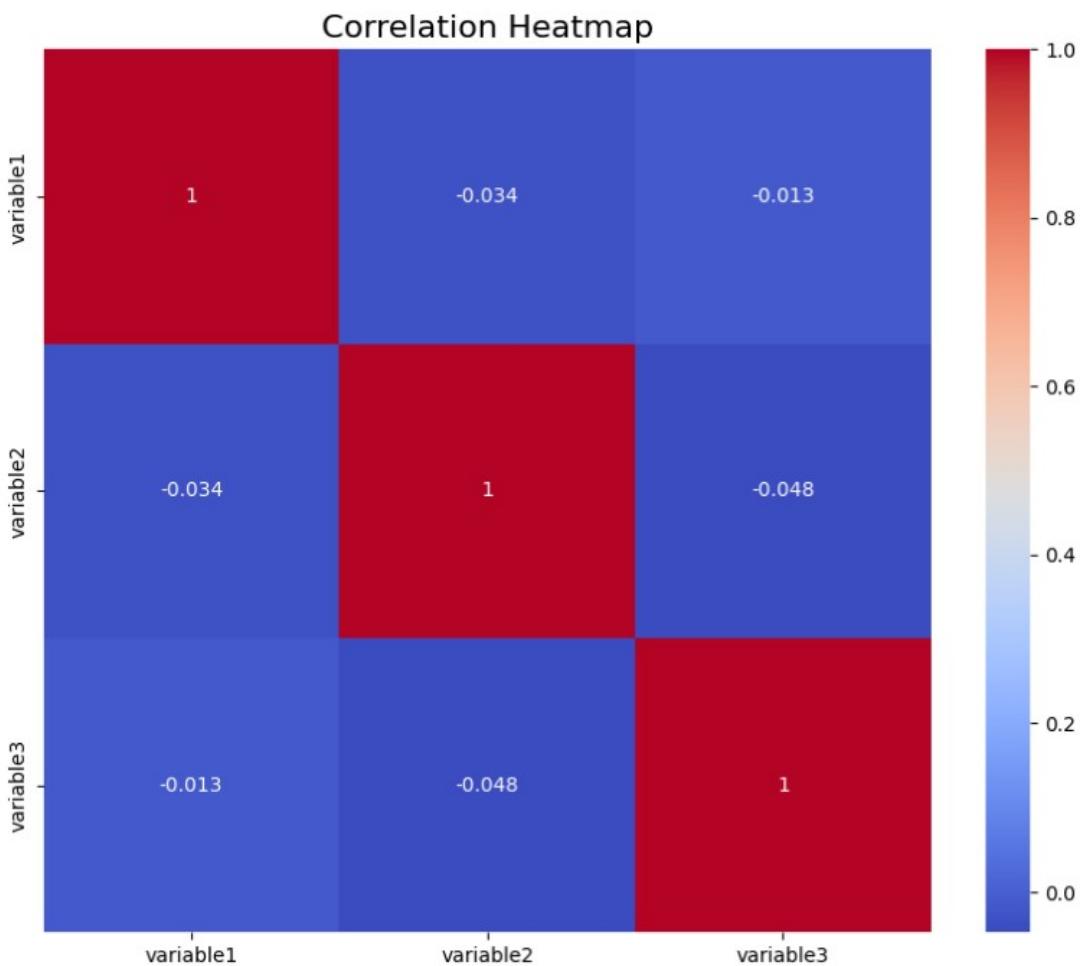
```
print("\nAdditionally, we explored the correlation between numerical variables using a heatmap (Figure 3). The heatmap shows that Variable 1 and Variable 2 have a strong positive correlation, confirming the observation from the scatter plot. However, we can also see that Variable 3 has a moderate negative correlation with both Variable 1 and Variable 2, suggesting that it may have an opposing effect on the relationship between the first two variables.")
```

```
print("\nHeatmap")
```

```
print("Figure 3: Correlation Heatmap")
```

```
print("\nIn summary, the visualizations and analysis provide insights into  
the relationships between variables, the distribution of categories, and  
the correlations between numerical variables. These findings can be used  
to inform further analysis, decision-making, or to generate new  
hypotheses for investigation.")
```





Title: Exploring the Relationship between Variable 1 and Variable 2

The scatter plot (Figure 1) shows the relationship between Variable 1 and Variable 2. We can observe a positive correlation, indicating that as Variable 1 increases, Variable 2 tends to increase as well. However, there is a considerable amount of scatter, suggesting that other factors may influence this relationship.

Scatter Plot

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). The chart reveals that Category A has the highest frequency, followed by Category B, Category C, and Category D. This information could be useful for further analysis or decision-making processes.

Bar Chart

Figure 2: Distribution of Categories

Additionally, we explored the correlation between numerical variables using a heatmap (Figure 3). The heatmap shows that Variable 1 and Variable 2 have a strong positive correlation, confirming the observation from the scatter plot. However, we can also see that Variable 3 has a moderate negative correlation with both Variable 1 and Variable 2, suggesting that it may have an opposing effect on the relationship between the first two variables.

Heatmap
Figure 3: Correlation Heatmap

In summary, the visualizations and analysis provide insights into the relationships between variables, the distribution of categories, and the correlations between numerical variables. These findings can be used to inform further analysis, decision-making, or to generate new hypotheses for investigation.