Laboratory Manual

(Machine Learning for Real World Application Lab & PCC-CSD691)

Table of Contents

Experiment/Lab Exercise/Activity No.	Name of the Experiment/Activity/Exercise	Page Number(s)
1	To perform different Arithmetic Operations on numbers in Python	1-4
2	Write a program to compute summary statistics such as mean, median, mode, standard deviation and variance of the given different types of data.	5-9
3	Plot the different results obtained by algebraic and statistical operations in python.	10-21
4	Model building using Linear Regression	22-27
5	Model building using Logistic Regression	28-37
6	Model building using K-Means Clustering	38-44
7	Performing EDA with Univariate analysis	45-50
8	Performing EDA with Bivariate analysis.	51-60
9	Performing EDA with Multivariate analysis.	61-68
10	Evaluate linear regression model using R2 and Adjusted R2 method.	69-77
11	Evaluate logistic regression model using Confusion matrix indices.	78-87
12	Evaluate K-Means clustering model using Confusion matrix indices.	88-93
13	Case study using Linear Regression	94-132
14	Case study using Logistic Regression	133-145
15	Case study using Decision Tree	146-150

.

SUSMIT CHAKRABORTY
Assistant Professor, CSE department
Brainware University, Kolkata

Experiment No 1

1.1 Aim/Purpose of the Experiment

To familiarize the students with different algebraic operations in python environment.

1.2 Learning Outcomes

Knowledge of the mathematical operations with user defined datasets in python.

1.3 Prerequisites

Basic knowledge of programming. Basic algebra.

1.4 Materials/Equipment/Apparatus / Devices/Software required

Jupyter Notebook.

1.5 Introduction and Theory

Basic algebraic operations are fundamental mathematical operations that involve manipulating numbers and symbols according to certain rules. These operations are essential for solving equations, simplifying expressions, and understanding mathematical relationships.

1. Addition (+): Adds two numbers.
2. Subtraction (-): Subtracts the second number from the first.
3. Multiplication (*): Multiplies two numbers.
4. Division (/): Divides the first number by the second (results in a floating-point number).
5. Integer Division (//): Divides the first number by the second and returns an integer.
6. Modulo (%): Returns the remainder of the division of the first number by the second.
7. Exponentiation (**): Raises the first number to the power of the second.

```
# Addition
result_addition = num1 + num2
print("Addition:", result_addition)

# Subtraction
result_subtraction = num1 - num2
print("Subtraction:", result_subtraction)

# Multiplication
result_multiplication = num1 * num2
print("Multiplication:", result_multiplication)
```

```
# Division
result_division = num1 / num2
print("Division:", result_division)

# Integer Division
result_integer_division = num1 // num2
print("Integer Division:", result_integer_division)

# Modulo
result_modulo = num1 % num2
print("Modulo:", result_modulo)

# Exponentiation
result_exponentiation = num1 ** num2
print("Exponentiation:", result_exponentiation)


# Taking inputs from the user
num1 = int(input("Enter your name: "))
num2 = int(input("Enter your age: "))
```

1.6 Operating Procedure

- Open Jupyter note book
- Take a new python file
- Type the code
- Run it
- Take inputs from user
- Observe the results
- Verify the results manually
- Store the note book file

1.7 Precautions and/or Troubleshooting

Precautions:

- Save Your Work: Regularly save your Jupyter Notebook to avoid losing your work. You can save your notebook by clicking on the save icon or using the keyboard shortcut Ctrl + S (or Cmd + S on Mac).

- Restart Kernel: If you encounter unexpected behavior or errors, try restarting the kernel. This clears all the variables and imported modules, essentially resetting the notebook's state. You can restart the kernel by going to the "Kernel" menu and selecting "Restart."

- Clear Outputs: To reduce clutter and confusion, consider clearing the outputs of code cells that are no longer relevant. You can do this by selecting "Clear Outputs" from the "Edit" menu.

- Readability: Keep your code and comments clear and well-organized to make it easier to understand and maintain. Use markdown cells for explanations, headings, and documentation.

- Check Dependencies: If you're using external libraries or packages, ensure they are properly installed in your Jupyter environment. You can check the installed packages by running !pip list or !conda list in a code cell.

- Kernel Selection: Make sure you're using the correct kernel for your notebook. The kernel determines the programming language and environment in which your code runs. You can change the kernel by clicking on "Kernel" > "Change kernel" in the menu.

- Resource Usage: Be mindful of the resources your notebook is using, especially if you're working with large datasets or running intensive computations. Check system monitor tools to ensure you're not exhausting memory or CPU resources.

Troubleshooting:

- Syntax Errors: Check for syntax errors in your code. Python is sensitive to indentation and syntax, so ensure your code is properly formatted.

- Variable Scope: Be aware of variable scope issues, especially if you're reusing variable names or working with nested functions.

- Library Installation: If you encounter Module Not Found Error or similar errors, ensure that the required libraries are installed in your Jupyter environment. You can install libraries using !pip install <library> or !conda install <library> in a code cell.

- Kernel Crashes: If the kernel crashes frequently, consider reducing the complexity of your code or optimizing resource usage. Large datasets or intensive computations can sometimes overwhelm the kernel.

- Browser Issues: If you experience rendering or responsiveness issues in the notebook interface, try clearing your browser cache or using a different browser.

- Documentation: Consult the official Jupyter documentation and community forums for additional troubleshooting tips and solutions to common problems.

1.8 Observations

Observe the results obtained in each operation.

1.9 Calculations & Analysis

Calculations should be given for each operation.

1.10 Result & Interpretation

Result should be printed and pasted in laboratory copy found from Jupyter note book.

1.11 Follow-up Questions

- What is Python?

- What is the difference between Python Arrays and lists?
- What are the common built-in data types in Python?
- What are local variables and global variables in Python?
- What is type conversion in Python?
- Explain the requirement of indentation in python.
- What is the method to write comments in Python?
- Is Python fully object oriented?
- What is the difference between %,/,// ?

1.12 Extension and Follow-up Activities (if applicable)

NA

1.13 Assessments

1.14 Suggested reading

NA

Experiment No 2

2.1 Aim/Purpose of the Experiment

To familiarize the students with different statistical operations in python environment.

2.2 Learning Outcomes

Knowledge of the statistical operations with user defined datasets in python.

2.3 Prerequisites

Basic knowledge of programming. Basic algebra.

2.4 Materials/Equipment/Apparatus / Devices/Software required

Jupyter Notebook.

2.5 Introduction and Theory

Statistics is a branch of mathematics that deals with collecting, analyzing, interpreting, presenting, and organizing data. Basic statistics provide essential tools for summarizing and understanding data, making informed decisions, and drawing conclusions from observations or experiments.

- Mean: The arithmetic mean, often simply called the "mean," is the sum of all values in a dataset divided by the number of values. It represents the average value of the data.

- Median: The median is the middle value of a dataset when the values are sorted in ascending or descending order. If the dataset has an even number of values, the median is the average of the two middle values.

- Mode: The mode is the value that appears most frequently in a dataset. A dataset can have one mode (unimodal), two modes (bimodal), or more modes (multimodal).

- Variance: The variance measures how much the values in a dataset deviate from the mean. It is the average of the squared differences between each value and the mean.

- Standard Deviation: The standard deviation is the square root of the variance. It provides a measure of the average distance between each data point and the mean.

```
import statistics

def compute_summary_statistics(data):
    mean = statistics.mean(data)
    median = statistics.median(data)
    try:
        mode = statistics.mode(data)
    except statistics.StatisticsError:
        mode = "No unique mode found"
    std_dev = statistics.stdev(data)
    variance = statistics.variance(data)

    return mean, median, mode, std_dev, variance

# User Defined dataset
n = int(input("enter the counts of the numbers: "))
numbers = []
for i in range(n):
    a = int(input("enter the number: "))
```

```
    numbers.append(a)
print(numbers)

# Compute summary statistics
mean, median, mode, std_dev, variance = compute_summary_statistics(numbers)

# Print results
print("Mean:", mean)
print("Median:", median)
print("Mode:", mode)
print("Standard Deviation:", std_dev)
print("Variance:", variance)
```

2.6 Operating Procedure

- Open Jupyter note book
- Take a new python file
- Type the code
- Run it
- Take inputs from user
- Observe the results
- Verify the results manually
- Store the note book file

2.7 Precautions and/or Troubleshooting

Precautions:

- Save Your Work: Regularly save your Jupyter Notebook to avoid losing your work. You can save your notebook by clicking on the save icon or using the keyboard shortcut Ctrl + S (or Cmd + S on Mac).

- Restart Kernel: If you encounter unexpected behavior or errors, try restarting the kernel. This clears all the variables and imported modules, essentially resetting the notebook's state. You can restart the kernel by going to the "Kernel" menu and selecting "Restart."

- Clear Outputs: To reduce clutter and confusion, consider clearing the outputs of code cells that are no longer relevant. You can do this by selecting "Clear Outputs" from the "Edit" menu.

- Readability: Keep your code and comments clear and well-organized to make it easier to understand and maintain. Use markdown cells for explanations, headings, and documentation.

- Check Dependencies: If you're using external libraries or packages, ensure they are properly installed in your Jupyter environment. You can check the installed packages by running !pip list or !conda list in a code cell.

- Kernel Selection: Make sure you're using the correct kernel for your notebook. The kernel determines the programming language and environment in which your code runs. You can change the kernel by clicking on "Kernel" > "Change kernel" in the menu.

- Resource Usage: Be mindful of the resources your notebook is using, especially if you're working with large datasets or running intensive computations. Check system monitor tools to ensure you're not exhausting memory or CPU resources.

Troubleshooting:

- Syntax Errors: Check for syntax errors in your code. Python is sensitive to indentation and syntax, so ensure your code is properly formatted.
- Variable Scope: Be aware of variable scope issues, especially if you're reusing variable names or working with nested functions.
- Library Installation: If you encounter Module Not Found Error or similar errors, ensure that the required libraries are installed in your Jupyter environment. You can install libraries using !pip install <library> or !conda install <library> in a code cell.
- Kernel Crashes: If the kernel crashes frequently, consider reducing the complexity of your code or optimizing resource usage. Large datasets or intensive computations can sometimes overwhelm the kernel.
- Browser Issues: If you experience rendering or responsiveness issues in the notebook interface, try clearing your browser cache or using a different browser.
- Documentation: Consult the official Jupyter documentation and community forums for additional troubleshooting tips and solutions to common problems.

2.8 Observations

Observe the results obtained in each operation.

2.9 Calculations & Analysis

Calculations should be given for each operation.

2.10 Result & Interpretation

Result should be printed and pasted in laboratory copy found from Jupyter note book.

2.11 Follow-up Questions

- What is Python?
- What is the difference between Python Arrays and lists?
- What are the common built-in data types in Python?
- What are local variables and global variables in Python?
- What is type conversion in Python?
- Explain the requirement of indentation in python.
- What is the method to write comments in Python?
- Is Python fully object oriented?
- What is package?

- What is the try method in python?
- What is the except method in python?
- What is function in python?

2.12 Extension and Follow-up Activities (if applicable)

NA

2.13 Assessments

2.14 Suggested reading

NA

Experiment No 3

3.1 Aim/Purpose of the Experiment

To familiarize the students with different graphical representation methods in python environment.

3.2 Learning Outcomes

Knowledge of different graphical representation methods in python.

3.3 Prerequisites

Basic knowledge of programming. Basic algebra. Python Packages.

3.4 Materials/Equipment/Apparatus / Devices/Software required

Jupyter Notebook.

3.5 Introduction and Theory

Data visualization is an important skill to possess for anyone trying to extract and communicate insights from data. Great business narratives and presentations often stem from brilliant visualizations that convey the key ideas in a concise and aesthetic manner. In the field of machine learning, visualization plays a key role throughout the entire process of analysis - to obtain relationships, observe trends and portray the final results as well.
There are two basic libraries that deals with the data visualization.

- matplotlib.pyplot
- seaborn

The Data visualization methods are as follows:

- Bar chart
- Scatter plot
- Line graph
- Histogram
- Box plot

Example - 1

Bar Chart - Plot sales across each product category

- A bar chart uses bars to show comparisons between categories of data.
- A bar graph will always have two axis.
- One axis will generally have numerical values or measures,
- The other will describe the types of categories being compared or dimensions.

```
import numpy as np

import matplotlib.pyplot as plt


product_category = np.array(['Furniture', 'Technology', 'Office Supplies'])

sales = np.array ([4110451.90, 4744557.50, 3787492.52] )


plt.bar(product_category, sales)

plt.show()
```

Bar Chart - Plot sales across each product category

1. Adding labels to Axes
2. Reducing the bar width
3. Giving Title to the chart
4. Modifying the ticks to show information in (million dollars)

```
import numpy as np

import matplotlib.pyplot as plt


product_category = np.array(['Furniture', 'Technology', 'Office Supplies'])

sales = np.array ([4110451.90, 4744557.50, 3787492.52] )


# plotting bar chart and setting bar width to 0.5 and aligning it to center

plt.bar(product_category, sales, width= 0.5, align='center', edgecolor='Orange',color='cyan')


# Adding and formatting title

plt.title("Sales Across Product Categories\n", fontdict={'fontsize': 20, 'fontweight' : 5, 'color' : 'Green'})


# Labeling Axes

plt.xlabel("Product Category", fontdict={'fontsize': 12, 'fontweight' : 5, 'color' : 'Brown'})
```


```
plt.ylabel("Sales", fontdict={'fontsize': 12, 'fontweight' : 5, 'color' : 'Brown'})


# Modifying the ticks to show information in (million dollars)

ticks = np.arange(0, 6000000, 1000000)

labels = ["{}M".format(i//1000000) for i in ticks]

plt.yticks(ticks, labels)


plt.show()
```

Example - 2

Scatter Chart - Plot Sales versus Profits across various Countries and Product Categories

Scatter plots are used when you want to show the relationship between two facts or measures.

- Profit is measured across each product category

```
import numpy as np
import matplotlib.pyplot as plt

# Data
sales = np.array ([1013.14, 8298.48, 875.51, 22320.83, 9251.6, 4516.86, 585.16, 836154.03, 216748.48, 174.2,
27557.79, 563.25, 558.11, 37117.45, 357.36, 2206.96, 709.5, 35064.03, 7230.78, 235.33, 148.32, 3973.27, 11737.8,
7104.63, 83.67, 5569.83, 92.34, 107104.36, 1045.62, 9072.51, 42485.82, 5093.82, 14846.16, 943.92, 684.36,
15012.03, 38196.18, 2448.75, 28881.96, 13912.14, 4507.2, 4931.06, 12805.05, 67912.73, 4492.2, 1740.01, 458.04,
16904.32, 21744.53, 10417.26, 18665.33, 2808.42, 54195.57, 67332.5, 24390.95, 1790.43, 2234.19, 9917.5, 7408.14,
36051.99, 1352.22, 1907.7, 245722.14, 2154.66, 1078.21, 3391.65, 28262.73, 5177.04, 66.51, 2031.34, 1683.72,
1970.01, 6515.82, 1055.31, 1029.48, 5303.4, 1850.96, 1159.41, 39989.13, 1183.87, 96365.09, 8356.68, 7010.24,
23119.23, 46109.28, 146071.84, 242259.03, 9058.95, 1313.67, 31525.06, 2019.94, 703.04, 1868.79, 700.5, 55512.02,
243.5, 2113.18, 11781.81, 262189.49, 3487.29, 513.12, 312050.42, 5000.7, 121.02, 1302.78, 169.92, 124.29,
57366.05, 29445.93, 4614.3, 45009.98, 309.24, 3353.67, 41348.34, 2280.27, 61193.7, 1466.79, 12419.94, 445.12,
25188.65, 263514.92, 12351.23, 1152.3, 26298.81, 9900.78, 5355.57, 2325.66, 6282.81, 127707.92, 1283.1, 3560.15,
3723.84, 13715.01, 4887.9, 3396.89, 33348.42, 625.02, 1665.48, 32486.97, 340212.44, 20516.22, 8651.16, 13590.06,
2440.35, 6462.57, 1770.13, 7527.18, 1433.65, 423.3, 21601.72, 10035.72, 2378.49, 3062.38, 719469.32, 179366.79,
345.17, 30345.78, 300.71, 940.81, 36468.08, 1352.85, 1755.72, 2391.96, 19.98, 19792.8, 15633.88, 7.45, 521.67,
1118.24, 7231.68, 12399.32, 204.36, 23.64, 5916.48, 313.98, 108181.5, 9212.42, 27476.91, 1761.33, 289.5, 780.3,
15098.46, 813.27, 47.55, 8323.23, 22634.64, 1831.02, 28808.1, 10539.78, 588.99, 939.78, 7212.41, 15683.01,
41369.09, 5581.6, 403.36, 375.26, 12276.66, 15393.56, 76.65, 5884.38, 18005.49, 3094.71, 43642.78, 35554.83,
22977.11, 1026.33, 665.28, 9712.49, 6038.52, 30756.51, 3758.25, 4769.49, 2463.3, 160153.16, 967.11, 2311.74,
1414.83, 12764.91, 4191.24, 110.76, 637.34, 1195.12, 2271.63, 804.12, 196.17, 167.67, 131.77, 2842.05, 9969.12,
1784.35, 3098.49, 25005.54, 1300.1, 118697.39, 7920.54, 6471.78, 31707.57, 37636.47, 118777.77, 131170.76,
```

3980.88, 3339.39, 26563.9, 4038.73, 124.8, 196.65, 2797.77, 29832.76, 184.84, 79.08, 8047.83, 205313.25, 1726.98, 899.73, 224.06, 304763.54, 6101.31, 729.6, 896.07, 17.82, 26.22, 46429.78, 31167.27, 2455.94, 37714.3, 1506.93, 3812.78, 25223.34, 3795.96, 437.31, 41278.86, 2091.81, 6296.61, 468.82, 23629.64, 160435.53, 9725.46, 1317.03, 1225.26, 30034.08, 7893.45, 2036.07, 215.52, 3912.42, 82783.43, 253.14, 966.96, 3381.26, 164.07, 1984.23, 75.12, 25168.17, 3295.53, 991.12, 10772.1, 44.16, 1311.45, 35352.57, 245783.54, 20.49, 13471.06, 8171.16, 14075.67, 611.82, 3925.56, 981.84, 10209.84, 156.56, 243.06, 21287.52, 7300.51, 434.52, 6065.0, 741577.51, 132461.03, 224.75, 28953.6, 757.98, 528.15, 34922.41, 50.58, 2918.48, 1044.96, 22195.13, 3951.48, 6977.64, 219.12, 5908.38, 10987.46, 4852.26, 445.5, 71860.82, 14840.45, 24712.08, 1329.9, 1180.44, 85.02, 10341.63, 690.48, 1939.53, 20010.51, 914.31, 25223.82, 12804.66, 2124.24, 602.82, 2961.66, 15740.79, 74138.35, 7759.39, 447.0, 2094.84, 22358.95, 21734.53, 4223.73, 17679.53, 1019.85, 51848.72, 69133.3, 30146.9, 705.48, 14508.88, 7489.38, 20269.44, 246.12, 668.13, 768.93, 215677.35, 899.16, 2578.2, 4107.99, 20334.57, 366.84, 3249.27, 98.88, 3497.88, 3853.05, 786.75, 1573.68, 458.36, 1234.77, 1094.22, 2300.61, 970.14, 3068.25, 35792.85, 4277.82, 71080.28, 3016.86, 3157.49, 15888.0, 30000.36, 140037.89, 216056.25, 1214.22, 1493.94, 32036.69, 4979.66, 106.02, 46257.68, 1033.3, 937.32, 3442.62, 160633.45, 213.15, 338.88, 242117.13, 9602.34, 2280.99, 73759.08, 23526.12, 6272.74, 43416.3, 576.78, 1471.61, 20844.9, 3497.7, 56382.38, 902.58, 6235.26, 48.91, 32684.24, 276611.58, 13370.38, 10595.28, 4555.14, 10084.38, 267.72, 1012.95, 4630.5, 149433.51, 364.32, 349.2, 4647.56, 504.0, 10343.52, 5202.66, 2786.26, 34135.95, 2654.58, 24699.51, 339239.87, 136.26, 23524.51, 8731.68, 8425.86, 835.95, 11285.19])

profit = np.array([-1213.46, 1814.13, -1485.7, -2286.73, -2872.12, 946.8, 198.48, 145454.95, 49476.1, -245.56, 5980.77, -790.47, -895.72, -34572.08, 117.9, 561.96, 152.85, 1426.05, 1873.17, -251.03, 68.22, 635.11, 3722.4, -3168.63, 27.6, 952.11, 7.38, 20931.13, 186.36, -5395.38, 9738.45, 525.27, 3351.99, 120.78, 266.88, 3795.21, 8615.97, 609.54, 7710.57, 2930.43, 1047.96, -2733.32, 2873.73, -5957.89, -909.6, 163.41, -376.02, -6322.68, -10425.86, 2340.36, -28430.53, 756.12, 12633.33, 7382.54, -14327.69, 436.44, 683.85, -694.91, 1960.56, 10925.82, 334.08, 425.49, 53580.2, 1024.56, 110.93, 632.22, 8492.58, 1418.88, 19.26, -2567.57, 346.26, 601.86, 1318.68, 304.05, 428.37, 1416.24, -2878.18, 283.41, 12611.04, 261.95, -648.43, 1112.88, -2640.29, 6154.32, 11558.79, 15291.4, 56092.65, 1515.39, 342.03, -10865.66, -902.8, 351.52, 364.17, 87.72, 11565.66, 75.4, 289.33, 3129.63, 50795.72, 783.72, 215.46, 29196.89, 1147.26, 53.22, 286.56, 73.02, 42.24, 13914.85, 5754.54, 998.04, -1476.04, 86.58, -1636.35, 10511.91, 647.34, 13768.62, 338.67, 3095.67, 173.84, 5632.93, 64845.11, 3297.33, 338.61, 7246.62, 2255.52, 1326.36, 827.64, 1100.58, 9051.36, 412.23, 1063.91, 940.59, 3891.84, 1599.51, 1129.57, 8792.64, 6.24, 592.77, 8792.85, 47727.5, -4597.68, 2242.56, 3546.45, 321.87, 1536.72, -2463.29, 1906.08, -1916.99, 186.24, 3002.05, -3250.98, 554.7, 830.64, 122612.79, 33894.21, -559.03, 7528.05, -477.67, -1660.25, -33550.96, 481.68, 425.08, 450.3, 9.57, -3025.29, 2924.62, -11.84, 87.36, 26.51, 1727.19, -6131.18, 59.16, 3.06, 1693.47, 74.67, 24729.21, -4867.94, 6705.18, 410.79, 70.74, 101.7, 3264.3, 137.01, 6.18, 2100.21, 5295.24, 520.29, 7205.52, 2602.65, 116.67, 224.91, -5153.93, 3882.69, -6535.24, -1254.1, 84.56, -186.38, -3167.2, -7935.59, 37.02, 1908.06, -27087.84, 829.32, 8727.44, 2011.47, -11629.64, 234.96, 53.1, 1248.14, 1511.07, 7374.24, 1193.28, 1090.23, 553.86, 38483.86, 255.81, 528.54, 326.07, 3924.36, 1018.92, 36.48, 113.24, -1770.05, 527.64, 224.49, 79.53, 64.77, 38.08, 868.08, 2265.06, -2643.62, 833.73, 5100.03, 326.44, 18158.84, 1682.01, -3290.22, 8283.33, 7926.18, 1694.41, 30522.92, 1214.07, 900.6, -6860.8, -865.91, 26.16, 47.22, 863.52, 7061.26, 73.92, 33.12, 1801.23, 38815.44, 431.13, 216.81, 16.5, 53688.2, 1210.32, 236.94, 210.84, 3.18, 2.22, 10265.64, 7212.3, 343.56, 3898.28, 568.11, -1867.85, 5782.38, 697.29, -192.06, 10179.02, 616.32, 1090.47, 165.84, 6138.28, 39723.06, 2085.14, 90.0, 129.93, 7957.53, 2131.86, 562.44, 99.12, 1298.37, 7580.33, 113.73, 139.71, 456.0, 21.24, 292.68, 30.34, 5817.15, 1060.89, 252.9, 3060.61, 6.6, 219.09, 8735.82, 31481.09, 2.85, -3124.72, 2195.94, 3464.7, 141.12, 1125.69, -1752.03, 3281.52, -303.77, 114.18, -2412.63, -5099.61, 146.64, 660.22, 18329.28, 28529.84, -232.27, 7435.41, -1157.94, -746.73, -30324.2, 2.52, 1313.44, 213.72, -5708.95, 930.18, 1663.02, 31.59, 1787.88, -8219.56, 973.92, 4.32, 8729.78, -2529.52, 5361.06, 69.21, 519.3, 13.56, 2236.77, 213.96, 367.98, 5074.2, 206.61, 7620.36, 2093.19, 164.07, 230.01, -815.82, 4226.7, -3635.09, -3344.17, 167.26, 143.79, -8233.57, -4085.21, 919.35, -25232.35, 234.33, 12040.68, 7206.28, -15112.76, 206.04, -2662.49, 2346.81, 4461.36, 93.48, 82.11, 147.87, 10389.53, 395.58, 474.74, 1333.26, 3913.02, 117.36, 858.78, 6.9, -4628.49, 1170.6, 218.55, 539.58, -211.0, 438.87, 317.16, 310.8, -1578.09, 706.56, 6617.4, 803.84, 2475.26, 764.34, -1461.88, 3805.56, 7371.27, -1377.13, 42435.03, 472.47, 315.48, -11755.91, -2418.6, 6.36, 9317.76, 326.88, -287.31, 637.68, 17579.17, 70.83, 47.4, 26143.92, 1548.15, 612.78, 17842.76, 6735.39, 1206.5, -10035.74, 149.4, -777.85, 5566.29, 748.92, 14941.58, 348.93, 1944.06, -5.51, 7026.84, 46114.92, 2361.86, 2613.24, 1277.37, 2587.74, 103.08, 311.43, 1250.58, 13055.21, 18.21, 108.24, 709.44, 115.92, 1863.6, 1873.86, 817.32, 7577.64, 1019.19, 6813.03, 24698.84, 66.24, -10971.39, 2056.47, 2095.35, 246.33, 2797.89])

```
product_category = np.array(['Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology',
'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology',
'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology',
'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology',
'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology',
'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology',
'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology',
'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology',
'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology',
'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology',
'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology',
'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology',
'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology',
'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology',
'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology',
'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology',
'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology',
'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology', 'Technology',
'Technology', 'Technology', 'Technology', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies',
'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office
Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies',
'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office
Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies',
'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office
Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies',
'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office
Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies',
'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office
Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies',
'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office
Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies',
'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office
Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies',
'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office
Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies',
'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office
Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies',
'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office
Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies',
'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office
Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies', 'Office Supplies',
'Office Supplies', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture',
'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture',
'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture',
'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture',
'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture',
'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture',
'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture',
'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture',
'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture',
```

```
'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture',
'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture',
'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture',
'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture',
'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture',
'Furniture', 'Furniture', 'Furniture', 'Furniture', 'Furniture'])
country = np.array(['Zimbabwe', 'Zambia', 'Yemen', 'Vietnam', 'Venezuela', 'Uzbekistan', 'Uruguay', 'United States',
'United Kingdom', 'United Arab Emirates', 'Ukraine', 'Uganda', 'Turkmenistan', 'Turkey', 'Tunisia', 'Trinidad and
Tobago', 'Togo', 'Thailand', 'Tanzania', 'Tajikistan', 'Taiwan', 'Syria', 'Switzerland', 'Sweden', 'Swaziland', 'Sudan', 'Sri
Lanka', 'Spain', 'South Sudan', 'South Korea', 'South Africa', 'Somalia', 'Singapore', 'Sierra Leone', 'Serbia', 'Senegal',
'Saudi Arabia', 'Rwanda', 'Russia', 'Romania', 'Qatar', 'Portugal', 'Poland', 'Philippines', 'Peru', 'Paraguay', 'Papua New
Guinea', 'Panama', 'Pakistan', 'Norway', 'Nigeria', 'Niger', 'Nicaragua', 'New Zealand', 'Netherlands', 'Nepal', 'Namibia',
'Myanmar (Burma)', 'Mozambique', 'Morocco', 'Mongolia', 'Moldova', 'Mexico', 'Mauritania', 'Martinique', 'Mali',
'Malaysia', 'Madagascar', 'Luxembourg', 'Lithuania', 'Libya', 'Liberia', 'Lesotho', 'Lebanon', 'Kyrgyzstan', 'Kenya',
'Kazakhstan', 'Jordan', 'Japan', 'Jamaica', 'Italy', 'Israel', 'Ireland', 'Iraq', 'Iran', 'Indonesia', 'India', 'Hungary', 'Hong
Kong', 'Honduras', 'Haiti', 'Guyana', 'Guinea-Bissau', 'Guinea', 'Guatemala', 'Guadeloupe', 'Greece', 'Ghana', 'Germany',
'Georgia', 'Gabon', 'France', 'Finland', 'Ethiopia', 'Estonia', 'Eritrea', 'Equatorial Guinea', 'El Salvador', 'Egypt',
'Ecuador', 'Dominican Republic', 'Djibouti', 'Denmark', 'Democratic Republic of the Congo', 'Czech Republic', 'Cuba',
'Croatia', "Cote d'Ivoire", 'Costa Rica', 'Colombia', 'China', 'Chile', 'Central African Republic', 'Canada', 'Cameroon',
'Cambodia', 'Burkina Faso', 'Bulgaria', 'Brazil', 'Bosnia and Herzegovina', 'Bolivia', 'Benin', 'Belgium', 'Belarus',
'Barbados', 'Bangladesh', 'Bahrain', 'Azerbaijan', 'Austria', 'Australia', 'Argentina', 'Angola', 'Algeria', 'Albania',
'Afghanistan', 'Zimbabwe', 'Zambia', 'Yemen', 'Western Sahara', 'Vietnam', 'Venezuela', 'Uzbekistan', 'Uruguay',
'United States', 'United Kingdom', 'United Arab Emirates', 'Ukraine', 'Uganda', 'Turkmenistan', 'Turkey', 'Tunisia',
'Trinidad and Tobago', 'Togo', 'The Gambia', 'Thailand', 'Tanzania', 'Tajikistan', 'Taiwan', 'Syria', 'Switzerland',
'Sweden', 'Swaziland', 'Suriname', 'Sudan', 'Sri Lanka', 'Spain', 'South Korea', 'South Africa', 'Somalia', 'Slovenia',
'Slovakia', 'Singapore', 'Sierra Leone', 'Serbia', 'Senegal', 'Saudi Arabia', 'Rwanda', 'Russia', 'Romania', 'Republic of the
Congo', 'Qatar', 'Portugal', 'Poland', 'Philippines', 'Peru', 'Paraguay', 'Papua New Guinea', 'Panama', 'Pakistan', 'Oman',
'Norway', 'Nigeria', 'Niger', 'Nicaragua', 'New Zealand', 'Netherlands', 'Nepal', 'Namibia', 'Myanmar (Burma)',
'Mozambique', 'Morocco', 'Montenegro', 'Mongolia', 'Moldova', 'Mexico', 'Mauritania', 'Martinique', 'Mali', 'Malaysia',
'Madagascar', 'Macedonia', 'Luxembourg', 'Lithuania', 'Libya', 'Liberia', 'Lesotho', 'Lebanon', 'Laos', 'Kyrgyzstan',
'Kenya', 'Kazakhstan', 'Jordan', 'Japan', 'Jamaica', 'Italy', 'Israel', 'Ireland', 'Iraq', 'Iran', 'Indonesia', 'India', 'Hungary',
'Hong Kong', 'Honduras', 'Haiti', 'Guyana', 'Guinea-Bissau', 'Guinea', 'Guatemala', 'Guadeloupe', 'Greece', 'Ghana',
'Germany', 'Georgia', 'Gabon', 'French Guiana', 'France', 'Finland', 'Ethiopia', 'Estonia', 'Eritrea', 'Equatorial Guinea', 'El
Salvador', 'Egypt', 'Ecuador', 'Dominican Republic', 'Djibouti', 'Denmark', 'Democratic Republic of the Congo', 'Czech
Republic', 'Cyprus', 'Cuba', 'Croatia', "Cote d'Ivoire", 'Costa Rica', 'Colombia', 'China', 'Chile', 'Chad', 'Central African
Republic', 'Canada', 'Cameroon', 'Cambodia', 'Burkina Faso', 'Bulgaria', 'Brazil', 'Botswana', 'Bosnia and Herzegovina',
'Bolivia', 'Bhutan', 'Benin', 'Belize', 'Belgium', 'Belarus', 'Barbados', 'Bangladesh', 'Bahrain', 'Azerbaijan', 'Austria',
'Australia', 'Armenia', 'Argentina', 'Angola', 'Algeria', 'Albania', 'Afghanistan', 'Zimbabwe', 'Zambia', 'Yemen',
'Western Sahara', 'Vietnam', 'Venezuela', 'Uzbekistan', 'Uruguay', 'United States', 'United Kingdom', 'United Arab
Emirates', 'Ukraine', 'Uganda', 'Turkmenistan', 'Turkey', 'Tunisia', 'Trinidad and Tobago', 'Togo', 'Thailand', 'Tanzania',
'Taiwan', 'Syria', 'Switzerland', 'Sweden', 'Sudan', 'Sri Lanka', 'Spain', 'South Korea', 'South Africa', 'Somalia',
'Slovenia', 'Slovakia', 'Singapore', 'Sierra Leone', 'Senegal', 'Saudi Arabia', 'Rwanda', 'Russia', 'Romania', 'Republic of
the Congo', 'Qatar', 'Portugal', 'Poland', 'Philippines', 'Peru', 'Paraguay', 'Papua New Guinea', 'Panama', 'Pakistan',
'Norway', 'Nigeria', 'Niger', 'Nicaragua', 'New Zealand', 'Netherlands', 'Nepal', 'Myanmar (Burma)', 'Mozambique',
'Morocco', 'Montenegro', 'Mongolia', 'Moldova', 'Mexico', 'Mauritania', 'Martinique', 'Mali', 'Malaysia', 'Malawi',
'Madagascar', 'Macedonia', 'Lithuania', 'Libya', 'Liberia', 'Lebanon', 'Laos', 'Kyrgyzstan', 'Kuwait', 'Kenya',
'Kazakhstan', 'Jordan', 'Japan', 'Jamaica', 'Italy', 'Israel', 'Ireland', 'Iraq', 'Iran', 'Indonesia', 'India', 'Hungary', 'Hong
Kong', 'Honduras', 'Haiti', 'Guyana', 'Guatemala', 'Guadeloupe', 'Greece', 'Ghana', 'Germany', 'Georgia', 'Gabon',
'France', 'Finland', 'Estonia', 'El Salvador', 'Egypt', 'Ecuador', 'Dominican Republic', 'Djibouti', 'Denmark', 'Democratic
Republic of the Congo', 'Czech Republic', 'Cuba', 'Croatia', "Cote d'Ivoire", 'Costa Rica', 'Colombia', 'China', 'Chile',
'Canada', 'Cameroon', 'Cambodia', 'Burundi', 'Burkina Faso', 'Bulgaria', 'Brazil', 'Botswana', 'Bosnia and Herzegovina',
'Bolivia', 'Benin', 'Belgium', 'Belarus', 'Barbados', 'Bangladesh', 'Azerbaijan', 'Austria', 'Australia', 'Armenia',
```


```
'Argentina', 'Angola', 'Algeria', 'Albania', 'Afghanistan'])

# plotting scatter chart

plt.scatter(profit, sales, alpha= 0.7, s = 50 )

# Adding and formatting title
plt.title("Sales versus Profits across various Countries and Product Categories\n", fontdict={'fontsize': 20, 'fontweight'
: 5, 'color' : 'Green'})

# Labeling Axes
plt.xlabel("Profit", fontdict={'fontsize': 12, 'fontweight' : 5, 'color' : 'Brown'})
plt.ylabel("Sales", fontdict={'fontsize': 12, 'fontweight' : 5, 'color' : 'Brown'})

plt.show()
```

Scatter Chart - Plot Sales versus Profits across various Countries and Product Categories

- Represent product category using different colors
- Adding a Legend to Product Categories

```
plt.scatter(profit[product_category == "Technology"], sales[product_category == "Technology"],
        c= 'Green', alpha= 0.7, s = 150, label="Technology" )

plt.scatter(profit[product_category == "Office Supplies"], sales[product_category == "Office Supplies"],
        c= 'Yellow', alpha= 0.7, s = 100, label="Office Supplies" )

plt.scatter(profit[product_category == "Furniture"], sales[product_category == "Furniture"],
        c= 'Cyan', alpha= 0.7, s = 50, label="Furniture" )

# Adding and formatting title
plt.title("Sales versus Profits across various Countries and Product Categories\n", fontdict={'fontsize': 20, 'fontweight'
: 5, 'color' : 'Green'})

# Labeling Axes
plt.xlabel("Profit", fontdict={'fontsize': 12, 'fontweight' : 5, 'color' : 'Brown'})
plt.ylabel("Sales", fontdict={'fontsize': 12, 'fontweight' : 5, 'color' : 'Brown'})

plt.legend()

plt.show()
```

Example - 3

Line Chart - Plot Sales across 2015

A line chart or line plot or line graph or curve chart is a type of chart which displays information as a series of data points called 'markers' connected by straight line segments. Most commonly used with time data.

```
import numpy as np
import matplotlib.pyplot as plt

months = np.array(['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October',
'November', 'December'])
sales = np.array([241268.56, 184837.36, 263100.77, 242771.86, 288401.05, 401814.06, 258705.68, 456619.94,
481157.24, 422766.63, 555279.03, 503143.69])
plt.plot(months, sales)

# Adding and formatting title
plt.title("Sales across 2015\n", fontdict={'fontsize': 20, 'fontweight' : 5, 'color' : 'Green'})

# Labeling Axes
plt.xlabel("Months", fontdict={'fontsize': 12, 'fontweight' : 5, 'color' : 'Brown'})
plt.ylabel("Sales", fontdict={'fontsize': 12, 'fontweight' : 5, 'color' : 'Brown'} )

ticks = np.arange(0, 600000, 50000)
labels = ["{}K".format(i//1000) for i in ticks]
plt.yticks(ticks, labels)

plt.xticks(rotation=90)

plt.show()

plt.plot(months, sales)

# Adding and formatting title
plt.title("Sales across 2015\n", fontdict={'fontsize': 20, 'fontweight' : 5, 'color' : 'Green'})

# Labeling Axes
plt.xlabel("Months", fontdict={'fontsize': 12, 'fontweight' : 5, 'color' : 'Brown'})
plt.ylabel("Sales", fontdict={'fontsize': 12, 'fontweight' : 5, 'color' : 'Brown'} )

ticks = np.arange(0, 600000, 50000)
labels = ["{}K".format(i//1000) for i in ticks]
plt.yticks(ticks, labels)

plt.xticks(rotation=90)

for xy in zip(months, sales):
    plt.annotate(s = "{}K".format(xy[1]//1000), xy = xy,  textcoords='data')

plt.show()
```

Example - 4

Box and Whisker Chart - Sales across Countries and Product Categories

A Box and Whisker Plot (or Box Plot) is a convenient way of visually displaying the data distribution through their quartiles. The lines extending parallel from the boxes are known as the “whiskers”, which are used to indicate variability

outside the upper and lower quartiles. Outliers are sometimes plotted as individual dots that are in-line with whiskers. Box Plots can be drawn either vertically or horizontally.

```
mport numpy as np
import matplotlib.pyplot as plt

# Data
sales_technology = np.array ([1013.14, 8298.48, 875.51, 22320.83, 9251.6, 4516.86, 585.16, 174.2, 27557.79,
563.25, 558.11, 37117.45, 357.36, 2206.96, 709.5, 35064.03, 7230.78, 235.33, 148.32, 3973.27, 11737.8, 7104.63,
83.67, 5569.83, 92.34, 1045.62, 9072.51, 42485.82, 5093.82, 14846.16, 943.92, 684.36, 15012.03, 38196.18,
2448.75, 28881.96, 13912.14, 4507.2, 4931.06, 12805.05, 67912.73, 4492.2, 1740.01, 458.04, 16904.32, 21744.53,
10417.26, 18665.33, 2808.42, 54195.57, 67332.5, 24390.95, 1790.43, 2234.19, 9917.5, 7408.14, 36051.99, 1352.22,
1907.7, 2154.66, 1078.21, 3391.65, 28262.73, 5177.04, 66.51, 2031.34, 1683.72, 1970.01, 6515.82, 1055.31,
1029.48, 5303.4, 1850.96, 1159.41, 39989.13, 1183.87, 96365.09, 8356.68, 7010.24, 23119.23, 46109.28, 9058.95,
1313.67, 31525.06, 2019.94, 703.04, 1868.79, 700.5, 55512.02, 243.5, 2113.18, 11781.81, 3487.29, 513.12, 5000.7,
121.02, 1302.78, 169.92, 124.29, 57366.05, 29445.93, 4614.3, 45009.98, 309.24, 3353.67, 41348.34, 2280.27,
61193.7, 1466.79, 12419.94, 445.12, 25188.65, 12351.23, 1152.3, 26298.81, 9900.78, 5355.57, 2325.66, 6282.81,
1283.1, 3560.15, 3723.84, 13715.01, 4887.9, 3396.89, 33348.42, 625.02, 1665.48, 32486.97, 20516.22, 8651.16,
13590.06, 2440.35, 6462.57])
sales_office_supplies = np.array ([1770.13, 7527.18, 1433.65, 423.3, 21601.72, 10035.72, 2378.49, 3062.38, 345.17,
30345.78, 300.71, 940.81, 36468.08, 1352.85, 1755.72, 2391.96, 19.98, 19792.8, 15633.88, 7.45, 521.67, 1118.24,
7231.68, 12399.32, 204.36, 23.64, 5916.48, 313.98, 9212.42, 27476.91, 1761.33, 289.5, 780.3, 15098.46, 813.27,
47.55, 8323.23, 22634.64, 1831.02, 28808.1, 10539.78, 588.99, 939.78, 7212.41, 15683.01, 41369.09, 5581.6,
403.36, 375.26, 12276.66, 15393.56, 76.65, 5884.38, 18005.49, 3094.71, 43642.78, 35554.83, 22977.11, 1026.33,
665.28, 9712.49, 6038.52, 30756.51, 3758.25, 4769.49, 2463.3, 967.11, 2311.74, 1414.83, 12764.91, 4191.24,
110.76, 637.34, 1195.12, 2271.63, 804.12, 196.17, 167.67, 131.77, 2842.05, 9969.12, 1784.35, 3098.49, 25005.54,
1300.1, 7920.54, 6471.78, 31707.57, 37636.47, 3980.88, 3339.39, 26563.9, 4038.73, 124.8, 196.65, 2797.77,
29832.76, 184.84, 79.08, 8047.83, 1726.98, 899.73, 224.06, 6101.31, 729.6, 896.07, 17.82, 26.22, 46429.78,
31167.27, 2455.94, 37714.3, 1506.93, 3812.78, 25223.34, 3795.96, 437.31, 41278.86, 2091.81, 6296.61, 468.82,
23629.64, 9725.46, 1317.03, 1225.26, 30034.08, 7893.45, 2036.07, 215.52, 3912.42, 82783.43, 253.14, 966.96,
3381.26, 164.07, 1984.23, 75.12, 25168.17, 3295.53, 991.12, 10772.1, 44.16, 1311.45, 35352.57, 20.49, 13471.06,
8171.16, 14075.67, 611.82, 3925.56])
sales_furniture = np.array ([981.84, 10209.84, 156.56, 243.06, 21287.52, 7300.51, 434.52, 6065.0, 224.75, 28953.6,
757.98, 528.15, 34922.41, 50.58, 2918.48, 1044.96, 22195.13, 3951.48, 6977.64, 219.12, 5908.38, 10987.46,
4852.26, 445.5, 71860.82, 14840.45, 24712.08, 1329.9, 1180.44, 85.02, 10341.63, 690.48, 1939.53, 20010.51,
914.31, 25223.82, 12804.66, 2124.24, 602.82, 2961.66, 15740.79, 74138.35, 7759.39, 447.0, 2094.84, 22358.95,
21734.53, 4223.73, 17679.53, 1019.85, 51848.72, 69133.3, 30146.9, 705.48, 14508.88, 7489.38, 20269.44, 246.12,
668.13, 768.93, 899.16, 2578.2, 4107.99, 20334.57, 366.84, 3249.27, 98.88, 3497.88, 3853.05, 786.75, 1573.68,
458.36, 1234.77, 1094.22, 2300.61, 970.14, 3068.25, 35792.85, 4277.82, 71080.28, 3016.86, 3157.49, 15888.0,
30000.36, 1214.22, 1493.94, 32036.69, 4979.66, 106.02, 46257.68, 1033.3, 937.32, 3442.62, 213.15, 338.88,
9602.34, 2280.99, 73759.08, 23526.12, 6272.74, 43416.3, 576.78, 1471.61, 20844.9, 3497.7, 56382.38, 902.58,
6235.26, 48.91, 32684.24, 13370.38, 10595.28, 4555.14, 10084.38, 267.72, 1012.95, 4630.5, 364.32, 349.2, 4647.56,
504.0, 10343.52, 5202.66, 2786.26, 34135.95, 2654.58, 24699.51, 136.26, 23524.51, 8731.68, 8425.86, 835.95,
11285.19])

plt.boxplot([sales_technology, sales_office_supplies, sales_furniture])

# Adding and formatting title
plt.title("Sales across Countries and Product Categories\n", fontdict={'fontsize': 20, 'fontweight' : 5, 'color' : 'Green'})

# Labeling Axes
plt.xlabel("Product Category", fontdict={'fontsize': 12, 'fontweight' : 5, 'color' : 'Brown'})
```

```
plt.ylabel("Sales", fontdict={'fontsize': 12, 'fontweight' : 5, 'color' : 'Brown'})

plt.xticks((1,2,3),["Technology", "Office Supplies", "Furniture"])

plt.show()
```

Example - 5

Histogram - Plot a histogram for Sales distribution across Countries.

A histogram is a plot that lets you discover, and show, the underlying frequency distribution (shape) of a set of continuous data.

```
import numpy as np
import matplotlib.pyplot as plt

# Data
profit = np.array([-5428.79, 7001.73, -3706.46, 300.42, -1697.31, -11222.71, 1648.14, 1689.34, -1036.86, 20944.23, -2426.09, -3302.71, 602.1, 2300.48, 816.87, 9.57, -7308.2, 5727.97, -262.87, 1818.6, 693.21, 7237.47, -17519.37, 86.76, 3.06, 3619.5, 86.37, 54390.12, 186.36, -12792.83, 21804.69, 1005.27, 590.04, 115.26, 8853.06, 471.75, 273.06, 6263.4, 18985.41, 1336.44, 22536.45, 7626.27, 280.74, 1502.88, -8703.06, 10983.12, -16128.23, -5507.88, 415.23, -418.61, -17723.45, -22446.65, 37.02, 5167.77, 1819.77, 33401.44, 16600.28, 877.44, 736.95, -2109.26, 5818.44, 22761.42, 1286.76, 1506.42, 1127.22, 1675.95, 1114.21, 2291.55, 16329.96, 117.36, 3296.58, 43.38, 132.5, -8966.12, 2044.5, 1044.9, 1398.21, 908.4, -172.92, 1735.32, 317.16, 3992.1, -7099.9, 1823.7, 24328.47, 1392.23, 19985.68, 3559.23, -7392.38, 18243.21, 26856.24, 15608.68, 3201.93, 1558.11, -29482.37, -4187.31, 384.04, 411.39, 951.24, 27944.69, 476.2, 35.14, 5568.54, 1285.68, 479.67, 16.5, 3905.73, 290.16, 1110.18, 76.2, 44.46, 42023.24, 19702.23, 2548.1, -7613.5, 804.09, -4282.05, 21860.58, 2093.55, -192.06, 38889.22, 1303.92, 6130.2, 334.17, 18798.05, 7744.33, 90.0, 468.54, 17817.39, 5664.75, 4476.54, 103.08, 1238.19, 3649.53, 29686.9, 131.94, 660.18, 2229.35, 21.24, 1349.19, 30.34, 11572.59, 4534.26, 2199.79, 19430.89, 12.84, 1831.05, 24341.7, 69.09, -18693.8, 6494.97, 9106.5, 709.32, 5460.3])

plt.hist(profit, bins = 100,edgecolor='Orange',color='cyan')

plt.show()
```

3.6 Operating Procedure

- Open Jupyter note book
- Take a new python file
- Type the code
- Run it
- Take inputs from user
- Observe the results
- Verify the results manually
- Store the note book file

3.7 Precautions and/or Troubleshooting

Precautions:

- Save Your Work: Regularly save your Jupyter Notebook to avoid losing your work. You can save your notebook by clicking on the save icon or using the keyboard shortcut Ctrl + S (or Cmd + S on Mac).
- Restart Kernel: If you encounter unexpected behavior or errors, try restarting the kernel. This clears all the variables and imported modules, essentially resetting the notebook's state. You can restart the kernel by going to the "Kernel" menu and selecting "Restart."
- Clear Outputs: To reduce clutter and confusion, consider clearing the outputs of code cells that are no longer relevant. You can do this by selecting "Clear Outputs" from the "Edit" menu.
- Readability: Keep your code and comments clear and well-organized to make it easier to understand and maintain. Use markdown cells for explanations, headings, and documentation.
- Check Dependencies: If you're using external libraries or packages, ensure they are properly installed in your Jupyter environment. You can check the installed packages by running !pip list or !conda list in a code cell.
- Kernel Selection: Make sure you're using the correct kernel for your notebook. The kernel determines the programming language and environment in which your code runs. You can change the kernel by clicking on "Kernel" > "Change kernel" in the menu.
- Resource Usage: Be mindful of the resources your notebook is using, especially if you're working with large datasets or running intensive computations. Check system monitor tools to ensure you're not exhausting memory or CPU resources.

Troubleshooting:

- Syntax Errors: Check for syntax errors in your code. Python is sensitive to indentation and syntax, so ensure your code is properly formatted.
- Variable Scope: Be aware of variable scope issues, especially if you're reusing variable names or working with nested functions.
- Library Installation: If you encounter Module Not Found Error or similar errors, ensure that the required libraries are installed in your Jupyter environment. You can install libraries using !pip install <library> or !conda install <library> in a code cell.
- Kernel Crashes: If the kernel crashes frequently, consider reducing the complexity of your code or optimizing resource usage. Large datasets or intensive computations can sometimes overwhelm the kernel.
- Browser Issues: If you experience rendering or responsiveness issues in the notebook interface, try clearing your browser cache or using a different browser.
- Documentation: Consult the official Jupyter documentation and community forums for additional troubleshooting tips and solutions to common problems.

3.8 Observations
Observe the results obtained in each operation.

3.9 Calculations & Analysis
Calculations should be given for each operation.

3.10 Result & Interpretation

Result should be printed and pasted in laboratory copy found from Jupyter note book.

3.11 Follow-up Questions
- How can you create a histogram in Matplotlib?
- How can you add a legend to a plot in Matplotlib?
- How can you set the font size of a plot in Matplotlib?
- What is the difference between a scatter plot and a line plot in Matplotlib?
- How can you add text to a plot in Matplotlib?
- What is the difference between a bar plot and a histogram in Matplotlib?
- How can you set the color of a plot in Matplotlib?
- What is the purpose of the plt.grid() function in Matplotlib?
- How can you create a box plot in Matplotlib?
- How can you set the size of a plot in Matplotlib?
- How can you change the linestyle of a plot in Matplotlib?
- How can you change the marker style of a plot in Matplotlib?

3.12 Extension and Follow-up Activities (if applicable)
NA

3.13 Assessments

3.14 Suggested reading
NA

Experiment No 4

4.1 Aim/Purpose of the Experiment
Model building using Linear Regression.

4.2 Learning Outcomes
Knowledge of the Data cleaning, modelling using linear regression, and different libraries in python.

4.3 Prerequisites

Basic knowledge of programming. Coordinate geometry (Straight line), Matplotlib, seaborn, etc.

4.4 Materials/Equipment/Apparatus / Devices/Software required

Jupyter Notebook.

4.5 Introduction and Theory

Linear regression is a fundamental statistical method used for modeling the relationship between a dependent variable and one or more independent variables. It's often employed in predictive analysis and understanding the association between variables.

- Dependent variable (Y): This is the variable that you want to predict or explain. It's also known as the response variable.

- Independent variable(s) (X): These are the variables that are used to predict the dependent variable. They are also referred to as predictor variables or features.

- Linear relationship: Linear regression assumes that there is a linear relationship between the independent variables and the dependent variable. This means that the change in the dependent variable is directly proportional to a change in the independent variable(s), with a constant rate of change.

- Simple linear regression: When there is only one independent variable, it's called simple linear regression. The relationship between the independent and dependent variables is modeled using a straight line equation: $Y = \beta_0 + \beta_1 X + \varepsilon$, where β_0 is the intercept, β_1 is the slope coefficient, X is the independent variable, and ε represents the error term.

- Multiple linear regression: When there are multiple independent variables, it's called multiple linear regression. The relationship between the independent and dependent variables is modeled using the equation: $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_r X_r + \varepsilon$, where X_1, X_2, ..., X_r are the independent variables, β_0 is the intercept, β_1, β_2, ..., β_r are the coefficients, and ε represents the error term.

Simple Linear Regression

Step 1: Reading and Understanding the Data

Let's start with the following steps:

1. Importing data using the pandas library
2. Understanding the structure of the data

```
# Supress Warnings

import warnings
warnings.filterwarnings('ignore')

# Import the numpy and pandas package

import numpy as np
import pandas as pd

# Read the given CSV file, and view some sample records

advertising = pd.read_csv("advertising.csv")
advertising.head()
advertising.shape

advertising.info()

advertising.describe()
```

Step 2: Visualising the Data

Let's now visualise our data using seaborn. We'll first make a pairplot of all the variables present to visualise which variables are most correlated to Sales

```
import matplotlib.pyplot as plt
import seaborn as sns

sns.pairplot(advertising, x_vars=['TV', 'Newspaper', 'Radio'], y_vars='Sales',size=4, aspect=1, kind='scatter')
plt.show()

sns.heatmap(advertising.corr(), cmap="YlGnBu", annot = True)
plt.show()
```

Step 3: Performing Simple Linear Regression

Equation of linear regression

$y = c + m_1 x_1 + m_2 x_2 + \ldots + m_n x_n$

- y is the response
- c is the intercept
- m_1 is the coefficient for the first feature
- m_n is the coefficient for the nth feature

In our case:

$y = c + m_1 \times TV$

The m values are called the model **coefficients** or **model parameters**.

Generic Steps in model building using statsmodels
We first assign the feature variable, TV, in this case, to the variable X and the response variable, Sales, to the variable y.

```
X = advertising['TV']
y = advertising['Sales']
```

Train-Test Split
You now need to split our variable into training and testing sets. You'll perform this by importing train_test_split from the sklearn.model_selection library. It is usually a good practice to keep 70% of the data in your train dataset and the rest 30% in your test dataset

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.7, test_size = 0.3, random_state = 100)

# Let's now take a look at the train dataset

X_train.head()

y_train.head()
```

Building a Linear Model

You first need to import the `statsmodel.api` library using which you'll perform the linear regression.

```
import statsmodels.api as sm
```

By default, the statsmodels library fits a line on the dataset which passes through the origin. But in order to have an intercept, you need to manually use the add_constant attribute of statsmodels. And once you've added the constant to your X_train dataset, you can go ahead and fit a regression line using the OLS (Ordinary Least Squares) attribute of statsmodels as shown below

```
# Add a constant to get an intercept
X_train_sm = sm.add_constant(X_train)
```

```
# Fit the resgression line using 'OLS'
lr = sm.OLS(y_train, X_train_sm).fit()

# Print the parameters, i.e. the intercept and the slope of the regression line fitted
lr.params

# Performing a summary operation lists out all the different parameters of the regression line fitted
print(lr.summary())
```

4.6 Operating Procedure

- Open Jupyter note book
- Take a new python file
- Type the code
- Run it
- Take inputs from user
- Observe the results
- Verify the results manually
- Store the note book file

4.7 Precautions and/or Troubleshooting

Precautions:

- Save Your Work: Regularly save your Jupyter Notebook to avoid losing your work. You can save your notebook by clicking on the save icon or using the keyboard shortcut Ctrl + S (or Cmd + S on Mac).

- Restart Kernel: If you encounter unexpected behavior or errors, try restarting the kernel. This clears all the variables and imported modules, essentially resetting the notebook's state. You can restart the kernel by going to the "Kernel" menu and selecting "Restart."

- Clear Outputs: To reduce clutter and confusion, consider clearing the outputs of code cells that are no longer relevant. You can do this by selecting "Clear Outputs" from the "Edit" menu.

- Readability: Keep your code and comments clear and well-organized to make it easier to understand and maintain. Use markdown cells for explanations, headings, and documentation.

- Check Dependencies: If you're using external libraries or packages, ensure they are properly installed in your Jupyter environment. You can check the installed packages by running !pip list or !conda list in a code cell.

- Kernel Selection: Make sure you're using the correct kernel for your notebook. The kernel determines the programming language and environment in which your code runs. You can change the kernel by clicking on "Kernel" > "Change kernel" in the menu.

- Resource Usage: Be mindful of the resources your notebook is using, especially if you're working with large datasets or running intensive computations. Check system monitor tools to ensure you're not exhausting memory or CPU resources.

Troubleshooting:

- Syntax Errors: Check for syntax errors in your code. Python is sensitive to indentation and syntax, so ensure your code is properly formatted.
- Variable Scope: Be aware of variable scope issues, especially if you're reusing variable names or working with nested functions.
- Library Installation: If you encounter Module Not Found Error or similar errors, ensure that the required libraries are installed in your Jupyter environment. You can install libraries using !pip install <library> or !conda install <library> in a code cell.
- Kernel Crashes: If the kernel crashes frequently, consider reducing the complexity of your code or optimizing resource usage. Large datasets or intensive computations can sometimes overwhelm the kernel.
- Browser Issues: If you experience rendering or responsiveness issues in the notebook interface, try clearing your browser cache or using a different browser.
- Documentation: Consult the official Jupyter documentation and community forums for additional troubleshooting tips and solutions to common problems.

4.8 Observations

Observe the results obtained in each step.

4.9 Calculations & Analysis

Calculations should be given for model output.

4.10 Result & Interpretation

Result should be printed and pasted in laboratory copy found from Jupyter note book.

4.11 Follow-up Questions

- What is linear regression, and how does it work?
- What are the assumptions of a linear regression model?
- What are outliers? How do you detect and treat them? How do you deal with outliers in a linear regression model?
- What is the difference between simple and multiple linear regression?
- What is multicollinearity and how does it affect linear regression analysis?
- What is the difference between linear regression and non-linear regression?
- Can you explain the concept of overfitting in linear regression?
- What are the limitations of linear regression?
- What is the curse of dimensionality?

4.12 Extension and Follow-up Activities (if applicable)

NA

4.13 Assessments

4.14 Suggested reading

NA

Experiment No 5

5.1 Aim/Purpose of the Experiment

To familiarize the students with Model building using Logistic Regression.

5.2 Learning Outcomes

Knowledge of the Data cleaning, modelling using logistic regression, and different libraries in python.

5.3 Prerequisites

Basic knowledge of programming, python syntax, matplotlib, seaborn, different libraries.

5.4 Materials/Equipment/Apparatus / Devices/Software required

Jupyter Notebook.

5.5 Introduction and Theory

Logistic regression is a statistical method used for modeling the probability of a binary outcome based on one or more predictor variables. It's widely used for classification tasks where the dependent variable is categorical and has two possible outcomes.
Here's an overview of the key concepts and components of logistic regression:

- Binary outcome: Logistic regression is specifically designed for situations where the dependent variable (also known as the response variable or target variable) is binary, meaning it has only two possible outcomes. These outcomes are typically represented as 0 and 1, or as "success" and "failure", "yes" and "no", etc.
- Logistic function (sigmoid function): In logistic regression, the relationship between the predictor variables and the probability of the binary outcome is modeled using the logistic function, also known as the sigmoid function. The logistic function is an S-shaped curve that maps any real-valued number to a value between 0 and 1, representing probabilities.
- Probability prediction: Unlike linear regression, where the output is continuous, logistic regression predicts the probability that a given observation belongs to a particular category (e.g., the probability of a customer buying a product). The predicted probabilities are then used to make classifications.
- Logit transformation: The logistic function is expressed in terms of the log-odds, also known as the logit function. The logit of the probability of the event occurring (p) is defined as the logarithm of the odds ratio (p / (1 - p)). Mathematically, it can be represented as log(p / (1 - p)).
- Model parameters: Similar to linear regression, logistic regression estimates parameters (coefficients) that define the relationship between the predictor variables and the log-odds of the binary outcome. These parameters are estimated using maximum likelihood estimation or other optimization techniques.
- Interpretation of coefficients: The coefficients obtained from logistic regression represent the change in the log-odds of the outcome associated with a one-unit change in the corresponding predictor

variable, holding other variables constant.

- Decision boundary: In logistic regression, a decision boundary is used to classify observations into different categories based on their predicted probabilities. The decision boundary is typically set at 0.5, meaning that observations with predicted probabilities greater than 0.5 are classified into one category, while those with predicted probabilities less than or equal to 0.5 are classified into the other category.

Logistic Regression

Step 1: Importing and Merging Data

```
# Suppressing Warnings
import warnings
warnings.filterwarnings

# Importing Pandas and NumPy
import pandas as pd, numpy as np

# Importing all datasets
churn_data = pd.read_csv("churn_data.csv")
churn_data.head()

customer_data = pd.read_csv("customer_data.csv")
customer_data.head()

internet_data = pd.read_csv("internet_data.csv")
internet_data.head()

#Combining all data files into one consolidated dataframe
# Merging on 'customerID'
df_1 = pd.merge(churn_data, customer_data, how='inner', on='customerID')

# Final dataframe with all predictor variables
telecom = pd.merge(df_1, internet_data, how='inner', on='customerID')
```

Step 2: Inspecting the Dataframe

```
# Let's see the head of our master dataset
telecom.head()

# Let's check the dimensions of the dataframe
telecom.shape

# let's look at the statistical aspects of the dataframe
telecom.describe()

# Let's see the type of each column
telecom.info()
```

Step 3: Data Preparation

```
#Converting some binary variables (Yes/No) to 0/1

# List of variables to map

varlist = ['PhoneService', 'PaperlessBilling', 'Churn', 'Partner', 'Dependents']

# Defining the map function
def binary_map(x):
    return x.map({'Yes': 1, "No": 0})

# Applying the function to the housing list
telecom[varlist] = telecom[varlist].apply(binary_map)

telecom.head()

#For categorical variables with multiple levels, create dummy features (one-hot encoded)
```

```
# Creating a dummy variable for some of the categorical variables and dropping the first one.
dummy1 = pd.get_dummies(telecom[['Contract', 'PaymentMethod', 'gender', 'InternetService']], drop_first=True)

# Adding the results to the master dataframe
telecom = pd.concat([telecom, dummy1], axis=1)

telecom.head()

# Creating dummy variables for the remaining categorical variables and dropping the level with big names.

# Creating dummy variables for the variable 'MultipleLines'
ml = pd.get_dummies(telecom['MultipleLines'], prefix='MultipleLines')
# Dropping MultipleLines_No phone service column
ml1 = ml.drop(['MultipleLines_No phone service'], 1)
#Adding the results to the master dataframe
telecom = pd.concat([telecom,ml1], axis=1)

# Creating dummy variables for the variable 'OnlineSecurity'.
os = pd.get_dummies(telecom['OnlineSecurity'], prefix='OnlineSecurity')
os1 = os.drop(['OnlineSecurity_No internet service'], 1)
# Adding the results to the master dataframe
telecom = pd.concat([telecom,os1], axis=1)

# Creating dummy variables for the variable 'OnlineBackup'.
ob = pd.get_dummies(telecom['OnlineBackup'], prefix='OnlineBackup')
ob1 = ob.drop(['OnlineBackup_No internet service'], 1)
# Adding the results to the master dataframe
telecom = pd.concat([telecom,ob1], axis=1)

# Creating dummy variables for the variable 'DeviceProtection'.
dp = pd.get_dummies(telecom['DeviceProtection'], prefix='DeviceProtection')
dp1 = dp.drop(['DeviceProtection_No internet service'], 1)
```

```
# Adding the results to the master dataframe
telecom = pd.concat([telecom,dp1], axis=1)

# Creating dummy variables for the variable 'TechSupport'.
ts = pd.get_dummies(telecom['TechSupport'], prefix='TechSupport')
ts1 = ts.drop(['TechSupport_No internet service'], 1)
# Adding the results to the master dataframe
telecom = pd.concat([telecom,ts1], axis=1)

# Creating dummy variables for the variable 'StreamingTV'.
st =pd.get_dummies(telecom['StreamingTV'], prefix='StreamingTV')
st1 = st.drop(['StreamingTV_No internet service'], 1)
# Adding the results to the master dataframe
telecom = pd.concat([telecom,st1], axis=1)

# Creating dummy variables for the variable 'StreamingMovies'.
sm = pd.get_dummies(telecom['StreamingMovies'], prefix='StreamingMovies')
sm1 = sm.drop(['StreamingMovies_No internet service'], 1)
# Adding the results to the master dataframe
telecom = pd.concat([telecom,sm1], axis=1)

telecom.head()
```

Dropping the repeated variables

```
# We have created dummies for the below variables, so we can drop them
telecom = telecom.drop(['Contract','PaymentMethod','gender','MultipleLines','InternetService',
'OnlineSecurity', 'OnlineBackup', 'DeviceProtection',
      'TechSupport', 'StreamingTV', 'StreamingMovies'], 1)

#The varaible was imported as a string we need to convert it to float
telecom['TotalCharges'] = telecom['TotalCharges'].convert_objects(convert_numeric=True)
```

```
telecom.info()

#Checking for Outliers

# Checking for outliers in the continuous variables
num_telecom = telecom[['tenure','MonthlyCharges','SeniorCitizen','TotalCharges']]

#Checking for Missing Values and Inputing Them
# Adding up the missing values (column-wise)
telecom.isnull().sum()

# Checking the percentage of missing values
round(100*(telecom.isnull().sum()/len(telecom.index)), 2)

# Removing NaN TotalCharges rows
telecom = telecom[~np.isnan(telecom['TotalCharges'])]

# Checking percentage of missing values after removing the missing values
round(100*(telecom.isnull().sum()/len(telecom.index)), 2)
```

Step 4: Test-Train Split

```
from sklearn.model_selection import train_test_split

# Putting feature variable to X
X = telecom.drop(['Churn','customerID'], axis=1)

X.head()

# Putting response variable to y
y = telecom['Churn']

y.head()
```

```
# Splitting the data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, test_size=0.3, random_state=100)
```

Step 5: Feature Scaling

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train[['tenure','MonthlyCharges','TotalCharges']] =
scaler.fit_transform(X_train[['tenure','MonthlyCharges','TotalCharges']])

X_train.head()
```

Checking the Churn Rate

```
churn = (sum(telecom['Churn'])/len(telecom['Churn'].index))*100
churn
```

Step 6: Looking at Correlations

```
# Importing matplotlib and seaborn
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# Let's see the correlation matrix
plt.figure(figsize = (20,10))        # Size of the figure
sns.heatmap(telecom.corr(),annot = True)
plt.show()

#Dropping highly correlated dummy variables
```

```
X_test =
X_test.drop(['MultipleLines_No','OnlineSecurity_No','OnlineBackup_No','DeviceProtection_No','TechSupport_No','StreamingTV_No','StreamingMovies_No'], 1)

X_train =
X_train.drop(['MultipleLines_No','OnlineSecurity_No','OnlineBackup_No','DeviceProtection_No','TechSupport_No', 'StreamingTV_No','StreamingMovies_No'], 1)


#Checking the Correlation Matrix

#After dropping highly correlated variables now let's check the correlation matrix again.

plt.figure(figsize = (20,10))

sns.heatmap(X_train.corr(),annot = True)

plt.show()



Step 7: Model Building

import statsmodels.api as sm


# Logistic regression model

logm1 = sm.GLM(y_train,(sm.add_constant(X_train)), family = sm.families.Binomial())

logm1.fit().summary()
```

5.6 Operating Procedure

- Open Jupyter note book
- Take a new python file
- Type the code
- Run it
- Take inputs from user
- Observe the results
- Verify the results manually
- Store the note book file

5.7 Precautions and/or Troubleshooting

Precautions:

- Save Your Work: Regularly save your Jupyter Notebook to avoid losing your work. You can save your notebook by clicking on the save icon or using the keyboard shortcut Ctrl + S (or Cmd + S on Mac).

- Restart Kernel: If you encounter unexpected behavior or errors, try restarting the kernel. This

clears all the variables and imported modules, essentially resetting the notebook's state. You can restart the kernel by going to the "Kernel" menu and selecting "Restart."

- Clear Outputs: To reduce clutter and confusion, consider clearing the outputs of code cells that are no longer relevant. You can do this by selecting "Clear Outputs" from the "Edit" menu.

- Readability: Keep your code and comments clear and well-organized to make it easier to understand and maintain. Use markdown cells for explanations, headings, and documentation.

- Check Dependencies: If you're using external libraries or packages, ensure they are properly installed in your Jupyter environment. You can check the installed packages by running !pip list or !conda list in a code cell.

- Kernel Selection: Make sure you're using the correct kernel for your notebook. The kernel determines the programming language and environment in which your code runs. You can change the kernel by clicking on "Kernel" > "Change kernel" in the menu.

- Resource Usage: Be mindful of the resources your notebook is using, especially if you're working with large datasets or running intensive computations. Check system monitor tools to ensure you're not exhausting memory or CPU resources.

Troubleshooting:

- Syntax Errors: Check for syntax errors in your code. Python is sensitive to indentation and syntax, so ensure your code is properly formatted.

- Variable Scope: Be aware of variable scope issues, especially if you're reusing variable names or working with nested functions.

- Library Installation: If you encounter Module Not Found Error or similar errors, ensure that the required libraries are installed in your Jupyter environment. You can install libraries using !pip install <library> or !conda install <library> in a code cell.

- Kernel Crashes: If the kernel crashes frequently, consider reducing the complexity of your code or optimizing resource usage. Large datasets or intensive computations can sometimes overwhelm the kernel.

- Browser Issues: If you experience rendering or responsiveness issues in the notebook interface, try clearing your browser cache or using a different browser.

- Documentation: Consult the official Jupyter documentation and community forums for additional troubleshooting tips and solutions to common problems.

5.8 Observations

Observe the results obtained in each operation.

5.9 Calculations & Analysis

Calculations should be given for each operation.

5.10 Result & Interpretation

Result should be printed and pasted in laboratory copy found from Jupyter note book.

5.11 Follow-up Questions

- What do you mean by the Logistic Regression?
- What are the different types of Logistic Regression?
- What are the odds?
- What is the Impact of Outliers on Logistic Regression?
- What is the difference between the outputs of the Logistic model and the Logistic function?
- How do we handle categorical variables in Logistic Regression?
- What are the assumptions made in Logistic Regression?
- Why is Logistic Regression termed as Regression and not classification?
- What are the advantages of Logistic Regression?
- What are the disadvantages of Logistic Regression?

5.12 Extension and Follow-up Activities (if applicable)

NA

5.13 Assessments

5.14 Suggested reading

NA

Experiment No 6

6.1 Aim/Purpose of the Experiment

To familiarize the students with Model building using K-Means Clustering.

6.2 Learning Outcomes

Knowledge of the Data cleaning, Data preparation, modelling K-Means Clustering, and different libraries in python.

6.3 Prerequisites

Basic knowledge of programming, python syntax, matplotlib, seaborn, different libraries.

6.4 Materials/Equipment/Apparatus / Devices/Software required

Jupyter Notebook.

6.5 Introduction and Theory

Logistic regression is a statistical method used for modeling the probability of a binary outcome based on one or more predictor variables. It's widely used for classification tasks where the dependent variable is categorical and has two possible outcomes.
Here's an overview of the key concepts and components of logistic regression:

- Binary outcome: Logistic regression is specifically designed for situations where the dependent variable (also known as the response variable or target variable) is binary, meaning it has only two possible outcomes. These outcomes are typically represented as 0 and 1, or as "success" and "failure", "yes" and "no", etc.
- Logistic function (sigmoid function): In logistic regression, the relationship between the predictor variables and the probability of the binary outcome is modeled using the logistic function, also known as the sigmoid function. The logistic function is an S-shaped curve that maps any real-valued number to a value between 0 and 1, representing probabilities.
- Probability prediction: Unlike linear regression, where the output is continuous, logistic regression predicts the probability that a given observation belongs to a particular category (e.g., the probability of a customer buying a product). The predicted probabilities are then used to make classifications.
- Logit transformation: The logistic function is expressed in terms of the log-odds, also known as the logit function. The logit of the probability of the event occurring (p) is defined as the logarithm of the odds ratio (p / (1 - p)). Mathematically, it can be represented as log(p / (1 - p)).
- Model parameters: Similar to linear regression, logistic regression estimates parameters (coefficients) that define the relationship between the predictor variables and the log-odds of the binary outcome. These parameters are estimated using maximum likelihood estimation or other optimization techniques.
- Interpretation of coefficients: The coefficients obtained from logistic regression represent the change in the log-odds of the outcome associated with a one-unit change in the corresponding predictor variable, holding other variables constant.

- Decision boundary: In logistic regression, a decision boundary is used to classify observations into different categories based on their predicted probabilities. The decision boundary is typically set at 0.5, meaning that observations with predicted probabilities greater than 0.5 are classified into one category, while those with predicted probabilities less than or equal to 0.5 are classified into the other category.

K-Means Clustering

1. Read and visualise the data

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

import datetime as dt

import sklearn
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

from scipy.cluster.hierarchy import linkage
from scipy.cluster.hierarchy import dendrogram
from scipy.cluster.hierarchy import cut_tree

# read the dataset
retail_df = pd.read_csv("Online_Retail.csv", sep=",", encoding="ISO-8859-1", header=0)
retail_df.head()

# read the dataset
retail_df = pd.read_csv("Online_Retail.csv", sep=",", encoding="ISO-8859-1", header=0)
retail_df.head()
```

2. Clean the data

```
# missing values
round(100*(retail_df.isnull().sum())/len(retail_df), 2)

# drop all rows having missing values
retail_df = retail_df.dropna()
retail_df.shape

retail_df.head()

# new column: amount
retail_df['amount'] = retail_df['Quantity']*retail_df['UnitPrice']
retail_df.head()
```

3. Prepare the data for modelling
#R (Recency): Number of days since last purchase
#F (Frequency): Number of tracsactions
#M (Monetary): Total amount of transactions (revenue contributed)

```
# monetary
grouped_df = retail_df.groupby('CustomerID')['amount'].sum()
grouped_df = grouped_df.reset_index()
grouped_df.head()

# frequency
frequency = retail_df.groupby('CustomerID')['InvoiceNo'].count()
frequency = frequency.reset_index()
frequency.columns = ['CustomerID', 'frequency']
frequency.head()

# merge the two dfs
grouped_df = pd.merge(grouped_df, frequency, on='CustomerID', how='inner')
grouped_df.head()

retail_df.head()

# recency
# convert to datetime
retail_df['InvoiceDate'] = pd.to_datetime(retail_df['InvoiceDate'],
                                          format='%d-%m-%Y %H:%M')

retail_df.head()

# compute the max date
max_date = max(retail_df['InvoiceDate'])
max_date

# compute the diff
retail_df['diff'] = max_date - retail_df['InvoiceDate']
retail_df.head()

# recency
last_purchase = retail_df.groupby('CustomerID')['diff'].min()
last_purchase = last_purchase.reset_index()
last_purchase.head()

# merge
grouped_df = pd.merge(grouped_df, last_purchase, on='CustomerID', how='inner')
grouped_df.columns = ['CustomerID', 'amount', 'frequency', 'recency']
grouped_df.head()

# number of days only
```

```
grouped_df['recency'] = grouped_df['recency'].dt.days
grouped_df.head()

# 1. outlier treatment
plt.boxplot(grouped_df['recency'])

# two types of outliers:
# - statistical
# - domain specific

# removing (statistical) outliers
Q1 = grouped_df.amount.quantile(0.05)
Q3 = grouped_df.amount.quantile(0.95)
IQR = Q3 - Q1
grouped_df = grouped_df[(grouped_df.amount >= Q1 - 1.5*IQR) & (grouped_df.amount <= Q3 +
1.5*IQR)]

# outlier treatment for recency
Q1 = grouped_df.recency.quantile(0.05)
Q3 = grouped_df.recency.quantile(0.95)
IQR = Q3 - Q1
grouped_df = grouped_df[(grouped_df.recency >= Q1 - 1.5*IQR) & (grouped_df.recency <= Q3 +
1.5*IQR)]

# outlier treatment for frequency
Q1 = grouped_df.frequency.quantile(0.05)
Q3 = grouped_df.frequency.quantile(0.95)
IQR = Q3 - Q1
grouped_df = grouped_df[(grouped_df.frequency >= Q1 - 1.5*IQR) & (grouped_df.frequency <= Q3 +
1.5*IQR)]


# 2. rescaling
rfm_df = grouped_df[['amount', 'frequency', 'recency']]

# instantiate
scaler = StandardScaler()

# fit_transform
rfm_df_scaled = scaler.fit_transform(rfm_df)
rfm_df_scaled.shape

rfm_df_scaled = pd.DataFrame(rfm_df_scaled)
rfm_df_scaled.columns = ['amount', 'frequency', 'recency']
rfm_df_scaled.head()

4. Modelling
# k-means with some arbitrary  k
```

```
kmeans = KMeans(n_clusters=4, max_iter=50)
kmeans.fit(rfm_df_scaled)

kmeans.labels_

# assign the label
grouped_df['cluster_id'] = kmeans.labels_
grouped_df.head()

# plot
sns.boxplot(x='cluster_id', y='amount', data=grouped_df)
```

6.6 Operating Procedure

- Open Jupyter note book
- Take a new python file
- Type the code
- Run it
- Take inputs from user
- Observe the results
- Verify the results manually
- Store the note book file

6.7 Precautions and/or Troubleshooting

Precautions:

- Save Your Work: Regularly save your Jupyter Notebook to avoid losing your work. You can save your notebook by clicking on the save icon or using the keyboard shortcut Ctrl + S (or Cmd + S on Mac).

- Restart Kernel: If you encounter unexpected behavior or errors, try restarting the kernel. This clears all the variables and imported modules, essentially resetting the notebook's state. You can restart the kernel by going to the "Kernel" menu and selecting "Restart."

- Clear Outputs: To reduce clutter and confusion, consider clearing the outputs of code cells that are no longer relevant. You can do this by selecting "Clear Outputs" from the "Edit" menu.

- Readability: Keep your code and comments clear and well-organized to make it easier to understand and maintain. Use markdown cells for explanations, headings, and documentation.

- Check Dependencies: If you're using external libraries or packages, ensure they are properly installed in your Jupyter environment. You can check the installed packages by running !pip list or !conda list in a code cell.

- Kernel Selection: Make sure you're using the correct kernel for your notebook. The kernel determines the programming language and environment in which your code runs. You can change the kernel by clicking on "Kernel" > "Change kernel" in the menu.

- Resource Usage: Be mindful of the resources your notebook is using, especially if you're working with large datasets or running intensive computations. Check system monitor tools to ensure you're not exhausting memory or CPU resources.

Troubleshooting:

- Syntax Errors: Check for syntax errors in your code. Python is sensitive to indentation and syntax, so ensure your code is properly formatted.
- Variable Scope: Be aware of variable scope issues, especially if you're reusing variable names or working with nested functions.
- Library Installation: If you encounter Module Not Found Error or similar errors, ensure that the required libraries are installed in your Jupyter environment. You can install libraries using !pip install <library> or !conda install <library> in a code cell.
- Kernel Crashes: If the kernel crashes frequently, consider reducing the complexity of your code or optimizing resource usage. Large datasets or intensive computations can sometimes overwhelm the kernel.
- Browser Issues: If you experience rendering or responsiveness issues in the notebook interface, try clearing your browser cache or using a different browser.
- Documentation: Consult the official Jupyter documentation and community forums for additional troubleshooting tips and solutions to common problems.

6.8 Observations

Observe the results obtained in each operation.

6.9 Calculations & Analysis

Calculations should be given for each operation.

6.10 Result & Interpretation

Result should be printed and pasted in laboratory copy found from Jupyter note book.

6.11 Follow-up Questions

- What is K means Clustering Algorithm?
- Is Feature Scaling required for the K means Algorithm?
- Why do you prefer Euclidean distance over Manhattan distance in the K means Algorithm?
- Does centroid initialization affect K means Algorithm?
- What are the advantages and disadvantages of the K means Algorithm?
- How to decide the optimal number of K in the K means Algorithm?
- What are the possible stopping conditions in the K means Algorithm?
- What is the effect of the number of variables on the K means Algorithm?

6.12 Extension and Follow-up Activities (if applicable)

NA

6.13 Assessments

6.14 Suggested reading

NA

Experiment No 7

7.1 Aim/Purpose of the Experiment

To familiarize the students with data visualization using one feature variables.

7.2 Learning Outcomes

Knowledge of the Data cleaning, Data preparation and data visualization using univariate analysis in python.

7.3 Prerequisites

Basic knowledge of programming, python syntax, matplotlib, seaborn, different libraries.

7.4 Materials/Equipment/Apparatus / Devices/Software required

Jupyter Notebook.

7.5 Introduction and Theory

Univariate analysis is a statistical method used to describe and understand the distribution, central tendency, and variability of a single variable. In Python, you can perform univariate analysis using libraries such as NumPy, Pandas, and Matplotlib/Seaborn for data manipulation, analysis, and visualization. Here's a brief outline of the process:

- Data Preparation: Load your dataset into a Pandas DataFrame and clean/preprocess the data if necessary. Ensure that the variable of interest is properly formatted and ready for analysis.
- Descriptive Statistics: Compute descriptive statistics for the variable of interest. This includes measures such as mean, median, mode, standard deviation, variance, minimum, maximum, and quartiles. These statistics provide an initial understanding of the distribution and characteristics of the variable.
- Visualization: Create visualizations to explore the distribution of the variable. Common plots for univariate analysis include histograms, box plots, density plots, and bar plots. Matplotlib and Seaborn are popular libraries for creating these visualizations.
- Central Tendency: Analyze the central tendency of the variable by examining the mean, median, and mode. This helps to understand the typical or central value around which the data is distributed.
- Variability: Explore the variability of the variable using measures such as standard deviation and variance. Variability indicates how spread out the data points are from the central tendency and provides insights into the dispersion of the data.
- Skewness and Kurtosis: Examine skewness and kurtosis to understand the shape of the distribution. Skewness measures the asymmetry of the distribution, while kurtosis measures the tail heaviness or peakedness of the distribution.

- Outlier Detection: Identify outliers in the data, which are data points that significantly deviate from the rest of the observations. Outliers can be detected visually using box plots or statistically using methods such as z-scores or interquartile range (IQR).

- Interpretation: Interpret the results of the analysis in the context of your research or problem domain. Draw conclusions about the distribution, central tendency, and variability of the variable, and consider any implications for further analysis or decision-making.

Univariate Analysis

```
import pandas as pd
import numpy as np

df=pd.read_csv('iris.csv')

df

#Finding the data types of variables in the DataFrame
df.dtypes

#Importing libraries essential for data visualization
#MATPLOTLIB
import matplotlib.pyplot as plt
%matplotlib inline

#SEABORN
import seaborn as sns

#Plots for continuous variables' analysis
#ENUMERATIVE PLOTS
#UNIVARIATE SCATTER PLOT
plt.scatter(df.index,df['sepal.width'])
plt.show()

sns.scatterplot(x=df.index,y=df['sepal.width'],hue=df['variety'])

#LINE PLOT WITH MARKERS
#Setting title, figure size, labels and font size in matplotlib

plt.figure(figsize=(6,6))
plt.title('Line plot of petal length')
plt.xlabel('index',fontsize=20)
plt.ylabel('petal length',fontsize=20)
plt.plot(df.index,df['petal.length'],markevery=1,marker='d')
```


```
for name, group in df.groupby('variety'):
   plt.plot(group.index,  group['petal.length'], label=name,markevery=1,marker='d')
plt.legend()
plt.show()

#Setting title, figure size,labels and font size in seaborn
sns.set(rc={'figure.figsize':(7,7)})
sns.set(font_scale=1.5)
fig=sns.lineplot(x=df.index,y=df['petal.length'],markevery=1,marker='d',data=df,hue=df['variety'])
fig.set(xlabel='index')

#STRIP PLOT
sns.stripplot(y=df['sepal.width'])

# Strip-plot(category wise)
sns.stripplot(x=df['variety'],y=df['sepal.width'])

#SWARM PLOT
#Setting figure size
sns.set(rc={'figure.figsize':(5,5)})

#Swarm-plot
sns.swarmplot(x=df['sepal.width'])

#Swarm-plot category wise
sns.swarmplot(x=df['variety'],y=df['sepal.width'])

#SUMMARY PLOTS
#HISTOGRAM
plt.hist(df['petal.width'])
sns.distplot(df['petal.width'],kde=False,color='black',bins=10)

#DENSITY PLOT
plt.figure(figsize=(5,5))
df['petal.length'].plot(kind='density')

sns.set(rc={'figure.figsize':(5,5)})
sns.kdeplot(df['petal.length'],shade=True)

#RUG PLOT
fig, ax = plt.subplots()
sns.rugplot(df['sepal.length'])
ax.set_xlim(3,9)
plt.show()
```


```
from scipy import stats
import numpy as np
kdf=df['sepal.length'].to_numpy()
rdf=np.hstack(kdf)
density = stats.kde.gaussian_kde(rdf)
x = np.arange(3,9,0.1)
plt.plot(x, density(x))
plt.plot(rdf,[0.01]*len(rdf), '|')

sns.distplot(df['sepal.length'],rug=True,hist=False)

#BOX PLOT
plt.boxplot(df['sepal.width'])

#Removing the column with categorical variables
dfM=df.drop('variety',axis=1)
plt.figure(figsize=(9,9))

#Set Title
plt.title('Box plots of the 4 variables')
plt.boxplot(dfM.values,labels=['SepalLength','SepalWidth','PetalLength','PetalWidth'])

sns.boxplot(df['sepal.width'])

sns.set(rc={'figure.figsize':(9,9)})
sns.boxplot(x="variable", y="value", data=pd.melt(dfM))

#distplot()
sns.set(rc={'figure.figsize':(6,6)})
sns.distplot(df['petal.length'],color='black',rug=True)

#VIOLIN PLOT
plt.figure(figsize=(7,7))
plt.violinplot(dfM.values,showmedians=True)

sns.set(rc={'figure.figsize':(5,5)})
sns.violinplot(df['sepal.width'],orient='vertical')

sns.set(rc={'figure.figsize':(9,9)})
sns.violinplot(x=df['variety'], y=df['petal.width'],data=df)

#Plots for categorical variables' analysis
#BAR PLOT
df['variety'].value_counts().plot.bar()
```

```
sns.countplot(df['variety'])

#PIE CHART
plt.pie(df['variety'].value_counts(),labels=['SETOSA','VERSICOLOR','VIRGINICA'],shadow=True)

df1=df.sample(frac=0.35)

plt.figure(figsize=(5,5))
plt.pie(df1['variety'].value_counts(),startangle=90,autopct='%.3f',labels=['SETOSA','VERSICOLOR','VIRGINICA'],shadow=True)
```

7.6 Operating Procedure

- Open Jupyter note book
- Take a new python file
- Type the code
- Run it
- Take inputs from user
- Observe the results
- Verify the results manually
- Store the note book file

7.7 Precautions and/or Troubleshooting

Precautions:

- Save Your Work: Regularly save your Jupyter Notebook to avoid losing your work. You can save your notebook by clicking on the save icon or using the keyboard shortcut Ctrl + S (or Cmd + S on Mac).

- Restart Kernel: If you encounter unexpected behavior or errors, try restarting the kernel. This clears all the variables and imported modules, essentially resetting the notebook's state. You can restart the kernel by going to the "Kernel" menu and selecting "Restart."

- Clear Outputs: To reduce clutter and confusion, consider clearing the outputs of code cells that are no longer relevant. You can do this by selecting "Clear Outputs" from the "Edit" menu.

- Readability: Keep your code and comments clear and well-organized to make it easier to understand and maintain. Use markdown cells for explanations, headings, and documentation.

- Check Dependencies: If you're using external libraries or packages, ensure they are properly installed in your Jupyter environment. You can check the installed packages by running !pip list or !conda list in a code cell.

- Kernel Selection: Make sure you're using the correct kernel for your notebook. The kernel determines the programming language and environment in which your code runs. You can change the kernel by clicking on "Kernel" > "Change kernel" in the menu.

- Resource Usage: Be mindful of the resources your notebook is using, especially if you're

working with large datasets or running intensive computations. Check system monitor tools to ensure you're not exhausting memory or CPU resources.

Troubleshooting:

- Syntax Errors: Check for syntax errors in your code. Python is sensitive to indentation and syntax, so ensure your code is properly formatted.
- Variable Scope: Be aware of variable scope issues, especially if you're reusing variable names or working with nested functions.
- Library Installation: If you encounter Module Not Found Error or similar errors, ensure that the required libraries are installed in your Jupyter environment. You can install libraries using !pip install <library> or !conda install <library> in a code cell.
- Kernel Crashes: If the kernel crashes frequently, consider reducing the complexity of your code or optimizing resource usage. Large datasets or intensive computations can sometimes overwhelm the kernel.
- Browser Issues: If you experience rendering or responsiveness issues in the notebook interface, try clearing your browser cache or using a different browser.
- Documentation: Consult the official Jupyter documentation and community forums for additional troubleshooting tips and solutions to common problems.

7.8 Observations

Observe the results obtained in each operation.

7.9 Calculations & Analysis

Calculations should be given for each operation.

7.10 Result & Interpretation

Result should be printed and pasted in laboratory copy found from Jupyter note book.

7.11 Follow-up Questions

- You need to check the relationship between the two variables. Which graph would you use?
- You need to check if a variable has outliers. Which graph would you use?
- You need to perform a univariate analysis. Which graph will you use?
- What is a data cleaning step?
- What are the ways to handle missing data?
- What are some of the methods for univariate analysis?
- What problems can outliers cause?

7.12 Extension and Follow-up Activities (if applicable)

NA

7.13 Assessments

7.14 Suggested reading

Experiment No 8

8.1 Aim/Purpose of the Experiment

To familiarize the students with data visualization using two feature variables.

8.2 Learning Outcomes

Knowledge of the Data cleaning, Data preparation and data visualization using bivariate analysis in python.

8.3 Prerequisites

Basic knowledge of programming, python syntax, matplotlib, seaborn, different libraries.

8.4 Materials/Equipment/Apparatus / Devices/Software required

Jupyter Notebook.

8.5 Introduction and Theory

Bivariate analysis is a statistical method used to examine the relationship between two variables. In Python, you can perform bivariate analysis using libraries such as NumPy, Pandas, and Matplotlib/Seaborn for data manipulation, analysis, and visualization. Here's a brief outline of the process:

- Data Preparation: Load your dataset into a Pandas DataFrame and clean/preprocess the data if necessary. Ensure that the two variables of interest are numeric or can be appropriately converted into numeric format.

- Descriptive Analysis: Compute descriptive statistics for each variable separately using methods like mean, median, standard deviation, etc. This provides initial insights into the characteristics of the variables.

- Visualization: Create visualizations to explore the relationship between the two variables. Common plots for bivariate analysis include scatter plots, line plots, box plots, and correlation matrices. Seaborn is particularly useful for creating attractive statistical visualizations.

- Correlation Analysis: Calculate the correlation coefficient between the two variables to measure the strength and direction of the linear relationship. Pearson correlation coefficient is commonly used for this purpose.

Case Study:

Term deposits also called fixed deposits, are the cash investments made for a specific time period ranging from 1 month to 5 years for predetermined fixed interest rates. The fixed interest rates offered for term deposits are higher than the regular interest rates for savings accounts. The customers receive the total amount (investment plus the interest) at the end of the maturity period. Also, the money can only be withdrawn at the end of the maturity period. Withdrawing

money before that will result in an added penalty associated, and the customer will not receive any interest returns.
Your target is to do end to end EDA on this bank telemarketing campaign data set to infer knowledge that where bank has to put more effort to improve it's positive response rate.

Bivariate Analysis

```
#import the warnings.
import warnings
warnings.filterwarnings("ignore")
#import the useful libraries.
import pandas as pd, numpy as np
import matplotlib.pyplot as plt, seaborn as sns
%matplotlib inline
```

Session- 2, Data Cleaning
Segment- 2, Data Types
There are multiple types of data types available in the data set. some of them are numerical type and some of categorical type. You are required to get the idea about the data types after reading the data frame.
Following are the some of the types of variables:
• Numeric data type: banking dataset: salary, balance, duration and age.
• Categorical data type: banking dataset: education, job, marital, poutcome and month etc.
• Ordinal data type: banking dataset: Age group.
• Time and date type
• Coordinates type of data: latitude and longitude type.

```
#read the data set of "bank telemarketing campaign" in inp0.
inp0= pd.read_csv("bank_marketing_updated_v1.csv")
#Print the head of the data frame.
inp0.head()
```

Segment- 3, Fixing the Rows and Columns
Checklist for fixing rows:
• Delete summary rows: Total and Subtotal rows
• Delete incorrect rows: Header row and footer row
• Delete extra rows: Column number, indicators, Blank rows, Page No.
Checklist for fixing columns:
• Merge columns for creating unique identifiers, if needed, for example, merge the columns State and City into the column Full address.
• Split columns to get more data: Split the Address column to get State and City columns

to analyse each separately.
• Add column names: Add column names if missing.
• Rename columns consistently: Abbreviations, encoded columns.
• Delete columns: Delete unnecessary columns.
• Align misaligned columns: The data set may have shifted columns, which you need to align correctly

```
#read the file in inp0 without first two rows as it is of no use.
inp0=pd.read_csv("bank_marketing_updated_v1.csv", skiprows= 2)
#print the head of the data frame.
inp0.head()

#drop the customer id as it is of no use.
inp0.drop("customerid", axis=1, inplace=True)
inp0.head()


#Extract job in newly created 'job' column from "jobedu" column.
inp0['job']=inp0.jobedu.apply(lambda x: x.split(",")[0])
inp0.head()

#Extract education in newly created 'education' column from "jobedu"
column.
inp0['education']=inp0.jobedu.apply(lambda x: x.split(",")[1])
inp0.head()

#drop the "jobedu" column from the dataframe.
inp0.drop('jobedu',axis= 1, inplace= True)
inp0.head()

inp0[inp0.month.apply(lambda x: isinstance(x,float))== True]

inp0.isnull().sum()
```

Segment- 4, Impute/Remove missing values
Take aways from the lecture on missing values:
• Set values as missing values: Identify values that indicate missing data, for example, treat blank strings, "NA", "XX", "999", etc., as missing.
• Adding is good, exaggerating is bad: You should try to get information from reliable external sources as much as possible, but if you can't, then it is better to retain missing values rather than exaggerating the existing rows/columns.
• Delete rows and columns: Rows can be deleted if the number of missing values is insignificant, as this would not impact the overall analysis results. Columns can be removed if the missing values are quite significant in number.
• Fill partial missing values using business judgement: Such values include missing time

zone, century, etc. These values can be identified easily.
Types of missing values:
• MCAR: It stands for Missing completely at random (the reason behind the missing value is not dependent on any other feature).
• MAR: It stands for Missing at random (the reason behind the missing value may be associated with some other features).
• MNAR: It stands for Missing not at random (there is a specific reason behind the missing value).

```
#count the missing values in age column.
inp0.age.isnull().sum()

#pring the shape of dataframe inp0
inp0.shape

#calculate the percentage of missing values in age column.
float(100.0*20/45211)

#drop the records with age missing in inp0 and copy in inp1 dataframe.
inp1=inp0[-inp0.age.isnull()].copy()
inp1.shape

#count the missing values in month column in inp1.
inp1.month.isnull().sum()


#print the percentage of each month in the data frame inp1.
float(100.0*50/45191)

#find the mode of month in inp1
month_mode=inp1.month.mode()[0]
month_mode


# fill the missing values with mode value of month in inp1.
inp1.month.fillna(month_mode, inplace= True)
inp1.month.value_counts(normalize= True)


#let's see the null values in the month column.
inp1.month.isnull().sum()
0

#count the missing values in response column in inp1.
inp1.response.isnull().sum()
```

30

#calculate the percentage of missing values in response column.
float(100.0*30/45191)
0.06638489964816004

#drop the records with response missings in inp1.
inp1= inp1[~inp1.response.isnull()]
#calculate the missing values in each column of data frame: inp1.
inp1.isnull().sum()

#describe the pdays column of inp1.
inp1.pdays.describe()

-1 indicates the missing values. Missing value does not always be present as null. How to handle it:
Objective is:
• you should ignore the missing values in the calculations
• simply make it missing - replace -1 with NaN.
• all summary statistics- mean, median etc. we will ignore the missing values of pdays.

#describe the pdays column with considering the -1 values.
inp1.loc[inp1.pdays<0,"pdays"]=np.NaN
inp1.pdays.describe()

Session- 4, Bivariate and Multivariate Analysis
Segment-2, Numeric- numeric analysis
There are three ways to analyse the numeric- numeric data types simultaneously.
• Scatter plot: describes the pattern that how one variable is varying with other variable.
• Correlation matrix: to describe the linearity of two numeric variables.
• Pair plot: group of scatter plots of all numeric variables in the data frame.

#plot the scatter plot of balance and salary variable in inp1
plt.scatter(inp1.salary, inp1.balance)
plt.show()

#plot the scatter plot of balance and age variable in inp1
inp1.plot.scatter(x='age', y='balance')
plt.show()

#plot the pair plot of salary, balance and age in inp1 dataframe.
sns.pairplot(data=inp1, vars=["salary","balance", "age"])

```
plt.show()

#plot the correlation matrix of salary, balance and age in inp1
dataframe.
sns.heatmap( inp1[["salary","balance", "age"]].corr(), annot= True,
cmap= "Reds")
plt.show()
```

Segment- 4, Numerical categorical variable
Salary vs response

```
#groupby the response to find the mean of the salary with response no
& yes seperatly.
inp1.groupby("response")["salary"].mean()


#groupby the response to find the median of the salary with response
no & yes seperatly.
inp1.groupby("response")["salary"].median()

#plot the box plot of salary for yes & no responses.
sns.boxplot(data=inp1,x="response",  y="salary")
plt.show()

#plot the box plot of balance for yes & no responses.
sns.boxplot(data=inp1,x="response",  y="balance")
plt.show()

#groupby the response to find the mean of the balance with response no
& yes seperatly.
inp1.groupby("response")["balance"].mean()

#groupby the response to find the median of the balance with response
no & yes seperatly.
inp1.groupby("response")["balance"].median()

#function to find the 75th percentile.
def p75(x):
 return np.quantile(x, 0.75)
#calculate the mean, median and 75th percentile of balance with
response
inp1.groupby("response")["balance"].aggregate(["mean","median",p75])

#plot the bar graph of balance's mean an median with response.
inp1.groupby("response")
```

```
["balance"].aggregate(["mean","median"]).plot.bar()
plt.show()
```

Education vs salary

```
#groupby the education to find the mean of the salary education
category.
inp1.groupby("education")["salary"].mean()

#groupby the education to find the median of the salary for each
education category.
inp1.groupby("education")["salary"].median()
```

Job vs salary

```
#groupby the job to find the mean of the salary for each job category.
inp1.groupby('job')['salary'].mean()

inp1.groupby('job')['salary'].median()
```

Segment- 5, Categorical categorical variable

```
#create response_flag of numerical data type where response "yes"= 1,
"no"= 0
inp1["response_flag"]=np.where(inp1.response=="yes",  1, 0)
inp1.response.value_counts()

inp1.response.value_counts(normalize=  True)

inp1.response_flag.mean()
```

Education vs response rate

```
#calculate the mean of response_flag with different education
categories.
inp1.groupby("education")["response_flag"].mean()
```

Marital vs response rate

```
#calculate the mean of response_flag with different marital status
categories.
inp1.groupby(["marital"])["response_flag"].mean()

#plot the bar graph of marital status with average value of
response_flag
inp1.groupby(["marital"])["response_flag"].mean().plot.barh()
plt.show()
```

```
Loans vs response rate
#plot the bar graph of personal loan status with average value of
response_flag
inp1.groupby(["loan"])["response_flag"].mean().plot.bar()
plt.show()

Housing loans vs response rate
#plot the bar graph of housing loan status with average value of
response_flag
inp1.groupby(["housing"])["response_flag"].mean().plot.bar()
plt.show()

Age vs response
#plot the boxplot of age with response_flag
sns.boxplot(data=inp1, x="response",y="age")
plt.show()

#plot the bar graph of job categories with response_flag mean value.
inp1.groupby(['job'])['response_flag'].mean().plot.barh()
plt.show()
```

8.6 Operating Procedure

- Open Jupyter note book
- Take a new python file
- Type the code
- Run it
- Take inputs from user
- Observe the results
- Verify the results manually
- Store the note book file

8.7 Precautions and/or Troubleshooting

Precautions:

- Save Your Work: Regularly save your Jupyter Notebook to avoid losing your work. You can save your notebook by clicking on the save icon or using the keyboard shortcut Ctrl + S (or Cmd + S on Mac).

- Restart Kernel: If you encounter unexpected behavior or errors, try restarting the kernel. This clears all the variables and imported modules, essentially resetting the notebook's state. You can restart the kernel by going to the "Kernel" menu and selecting "Restart."

- Clear Outputs: To reduce clutter and confusion, consider clearing the outputs of code cells that are no longer relevant. You can do this by selecting "Clear Outputs" from the "Edit" menu.

- Readability: Keep your code and comments clear and well-organized to make it easier to understand and maintain. Use markdown cells for explanations, headings, and documentation.

- Check Dependencies: If you're using external libraries or packages, ensure they are properly installed in your Jupyter environment. You can check the installed packages by running !pip list or !conda list in a code cell.

- Kernel Selection: Make sure you're using the correct kernel for your notebook. The kernel determines the programming language and environment in which your code runs. You can change the kernel by clicking on "Kernel" > "Change kernel" in the menu.

- Resource Usage: Be mindful of the resources your notebook is using, especially if you're working with large datasets or running intensive computations. Check system monitor tools to ensure you're not exhausting memory or CPU resources.

Troubleshooting:

- Syntax Errors: Check for syntax errors in your code. Python is sensitive to indentation and syntax, so ensure your code is properly formatted.

- Variable Scope: Be aware of variable scope issues, especially if you're reusing variable names or working with nested functions.

- Library Installation: If you encounter Module Not Found Error or similar errors, ensure that the required libraries are installed in your Jupyter environment. You can install libraries using !pip install <library> or !conda install <library> in a code cell.

- Kernel Crashes: If the kernel crashes frequently, consider reducing the complexity of your code or optimizing resource usage. Large datasets or intensive computations can sometimes overwhelm the kernel.

- Browser Issues: If you experience rendering or responsiveness issues in the notebook interface, try clearing your browser cache or using a different browser.

- Documentation: Consult the official Jupyter documentation and community forums for additional troubleshooting tips and solutions to common problems.

8.8 Observations

Observe the results obtained in each operation.

8.9 Calculations & Analysis

Calculations should be given for each operation.

8.10 Result & Interpretation

Result should be printed and pasted in laboratory copy found from Jupyter note book.

8.11 Follow-up Questions

- You need to check the relationship between the two variables. Which graph would you use?
- You need to check if a variable has outliers. Which graph would you use?
- You need to perform a univariate analysis. Which graph will you use?
- What is a data cleaning step?
- What are the ways to handle missing data?
- What are some of the methods for univariate analysis?
- What problems can outliers cause?

8.12 Extension and Follow-up Activities (if applicable)

NA

8.13 Assessments

8.14 Suggested reading

NA

Experiment No 9

9.1 Aim/Purpose of the Experiment

To familiarize the students with data visualization using more than two feature variables.

9.2 Learning Outcomes

Knowledge of the Data cleaning, Data preparation and data visualization using multivariate analysis in python.

9.3 Prerequisites

Basic knowledge of programming, python syntax, matplotlib, seaborn, different libraries.

9.4 Materials/Equipment/Apparatus / Devices/Software required

Jupyter Notebook.

9.5 Introduction and Theory

Multivariate analysis is a statistical method used to understand the relationship between multiple variables simultaneously. In Python, you can perform multivariate analysis using libraries such as NumPy, Pandas, and Matplotlib/Seaborn for data manipulation, analysis, and visualization. Here's a brief outline of the process:

- Data Preparation: Load your dataset into a Pandas DataFrame and clean/preprocess the data if necessary. Ensure that the variables of interest are properly formatted and ready for analysis.
- Descriptive Statistics: Compute descriptive statistics for all variables in the dataset. This includes measures such as mean, median, mode, standard deviation, variance, minimum, maximum, and quartiles for each variable.
- Visualization: Create visualizations to explore the relationships between multiple variables. Common plots for multivariate analysis include scatter plots, pair plots, heatmap correlation matrices, and parallel coordinate plots. Matplotlib and Seaborn are popular libraries for creating these visualizations.
- Correlation Analysis: Calculate correlation coefficients between pairs of variables to measure the strength and direction of the linear relationship. This helps identify potential associations between variables and can guide further analysis.
- Dimensionality Reduction: If dealing with a high-dimensional dataset, consider dimensionality reduction techniques such as Principal Component Analysis (PCA) or t-distributed Stochastic Neighbor Embedding (t-SNE) to visualize and analyze the data in a lower-dimensional space while preserving its important features.

- Cluster Analysis: Explore patterns and groupings within the data using clustering techniques such as K-means clustering or hierarchical clustering. This helps identify naturally occurring clusters of similar observations based on the values of multiple variables.

- Regression Analysis: Perform regression analysis to model the relationship between one or more independent variables and a dependent variable. This can help predict the value of the dependent variable based on the values of the independent variables.

- Hypothesis Testing (Optional): If applicable, conduct hypothesis tests to determine if there are significant differences or associations between groups or variables. This could involve techniques such as ANOVA, Chi-square tests, or regression analysis with hypothesis tests on coefficients.

- Interpretation: Interpret the results of the analysis in the context of your research or problem domain. Draw conclusions about the relationships, patterns, and associations observed between multiple variables and consider any implications for further analysis or decision-making.

Case Study:

Term deposits also called fixed deposits, are the cash investments made for a specific time period ranging from 1 month to 5 years for predetermined fixed interest rates. The fixed interest rates offered for term deposits are higher than the regular interest rates for savings accounts. The customers receive the total amount (investment plus the interest) at the end of the maturity period. Also, the money can only be withdrawn at the end of the maturity period. Withdrawing money before that will result in an added penalty associated, and the customer will not receive any interest returns.

Your target is to do end to end EDA on this bank telemarketing campaign data set to infer knowledge that where bank has to put more effort to improve it's positive response rate.

Multivariate Analysis

```
#import the warnings.
import warnings
warnings.filterwarnings("ignore")
#import the useful libraries.
import pandas as pd, numpy as np
import matplotlib.pyplot as plt, seaborn as sns
%matplotlib inline
```

Session- 2, Data Cleaning

Segment- 2, Data Types

There are multiple types of data types available in the data set. some of them are numerical type and some of categorical type. You are required to get the idea about the data types after reading the data frame.

Following are the some of the types of variables:

- Numeric data type: banking dataset: salary, balance, duration and age.
- Categorical data type: banking dataset: education, job, marital, poutcome and month

etc.

- Ordinal data type: banking dataset: Age group.
- Time and date type
- Coordinates type of data: latitude and longitude type.

```
#read the data set of "bank telemarketing campaign" in inp0.
inp0= pd.read_csv("bank_marketing_updated_v1.csv")
#Print the head of the data frame.
inp0.head()
```

Segment- 3, Fixing the Rows and Columns

Checklist for fixing rows:

- Delete summary rows: Total and Subtotal rows
- Delete incorrect rows: Header row and footer row
- Delete extra rows: Column number, indicators, Blank rows, Page No.

Checklist for fixing columns:

- Merge columns for creating unique identifiers, if needed, for example, merge the columns State and City into the column Full address.
- Split columns to get more data: Split the Address column to get State and City columns to analyse each separately.
- Add column names: Add column names if missing.
- Rename columns consistently: Abbreviations, encoded columns.
- Delete columns: Delete unnecessary columns.
- Align misaligned columns: The data set may have shifted columns, which you need to align correctly

```
#read the file in inp0 without first two rows as it is of no use.
inp0=pd.read_csv("bank_marketing_updated_v1.csv", skiprows= 2)
#print the head of the data frame.
inp0.head()

#drop the customer id as it is of no use.
inp0.drop("customerid", axis=1, inplace=True)
inp0.head()


#Extract job in newly created 'job' column from "jobedu" column.
inp0['job']=inp0.jobedu.apply(lambda x: x.split(",")[0])
inp0.head()

#Extract education in newly created 'education' column from "jobedu"
column.
inp0['education']=inp0.jobedu.apply(lambda x: x.split(",")[1])
inp0.head()

#drop the "jobedu" column from the dataframe.
```


```
inp0.drop('jobedu',axis= 1, inplace= True)
inp0.head()

inp0[inp0.month.apply(lambda x: isinstance(x,float))== True]

inp0.isnull().sum()
```

Segment- 4, Impute/Remove missing values

Take aways from the lecture on missing values:

• Set values as missing values: Identify values that indicate missing data, for example, treat blank strings, "NA", "XX", "999", etc., as missing.

• Adding is good, exaggerating is bad: You should try to get information from reliable external sources as much as possible, but if you can't, then it is better to retain missing values rather than exaggerating the existing rows/columns.

• Delete rows and columns: Rows can be deleted if the number of missing values is insignificant, as this would not impact the overall analysis results. Columns can be removed if the missing values are quite significant in number.

• Fill partial missing values using business judgement: Such values include missing time zone, century, etc. These values can be identified easily.

Types of missing values:

• MCAR: It stands for Missing completely at random (the reason behind the missing value is not dependent on any other feature).

• MAR: It stands for Missing at random (the reason behind the missing value may be associated with some other features).

• MNAR: It stands for Missing not at random (there is a specific reason behind the missing value).

```
#count the missing values in age column.
inp0.age.isnull().sum()

#pring the shape of dataframe inp0
inp0.shape

#calculate the percentage of missing values in age column.
float(100.0*20/45211)

#drop the records with age missing in inp0 and copy in inp1 dataframe.
inp1=inp0[-inp0.age.isnull()].copy()
inp1.shape

#count the missing values in month column in inp1.
inp1.month.isnull().sum()


#print the percentage of each month in the data frame inp1.
float(100.0*50/45191)
```


```
#find the mode of month in inp1
month_mode=inp1.month.mode()[0]
month_mode


# fill the missing values with mode value of month in inp1.
inp1.month.fillna(month_mode, inplace= True)
inp1.month.value_counts(normalize= True)


#let's see the null values in the month column.
inp1.month.isnull().sum()
0

#count the missing values in response column in inp1.
inp1.response.isnull().sum()
30

#calculate the percentage of missing values in response column.
float(100.0*30/45191)
0.06638489964816004

#drop the records with response missings in inp1.
inp1= inp1[~inp1.response.isnull()]
#calculate the missing values in each column of data frame: inp1.
inp1.isnull().sum()


#describe the pdays column of inp1.
inp1.pdays.describe()
```

-1 indicates the missing values. Missing value does not always be present as null. How to handle it:
Objective is:

- you should ignore the missing values in the calculations
- simply make it missing - replace -1 with NaN.
- all summary statistics- mean, median etc. we will ignore the missing values of pdays.

```
#describe the pdays column with considering the -1 values.
inp1.loc[inp1.pdays<0,"pdays"]=np.NaN
inp1.pdays.describe()


Multivariate analysis
Education vs marital vs response
res=pd.pivot_table(data=inp1, index="education", columns="marital",
values="response_flag")
```

```
res


#create heat map of education vs marital vs response_flag
sns.heatmap(res, annot= True, cmap="RdYlGn")
plt.show()

sns.heatmap(res, annot= True, cmap="RdYlGn", center= 0.117)
plt.show()

Job vs marital vs response
res=pd.pivot_table(data=inp1, index="job", columns="marital",
values="response_flag")
res


#create the heat map of Job vs marital vs response_flag.
sns.heatmap(res, annot= True, cmap="RdYlGn", center= 0.117)
plt.show()

Education vs poutcome vs response
#create the heat map of education vs poutcome vs response_flag.
res=pd.pivot_table(data=inp1, index="education", columns="poutcome",
values="response_flag")

sns.heatmap(res, annot= True, cmap="RdYlGn", center= 0.117)
plt.show()

inp1[inp1.pdays>0].response_flag.mean()

res=pd.pivot_table(data=inp1, index="education", columns="poutcome",
values="response_flag")
sns.heatmap(res, annot= True, cmap="RdYlGn", center= 0.2308)
plt.show()
```

9.6 Operating Procedure

- Open Jupyter note book
- Take a new python file
- Type the code
- Run it
- Take inputs from user
- Observe the results
- Verify the results manually
- Store the note book file

9.7 Precautions and/or Troubleshooting

Precautions:

- Save Your Work: Regularly save your Jupyter Notebook to avoid losing your work. You can save your notebook by clicking on the save icon or using the keyboard shortcut Ctrl + S (or Cmd + S on Mac).
- Restart Kernel: If you encounter unexpected behavior or errors, try restarting the kernel. This clears all the variables and imported modules, essentially resetting the notebook's state. You can restart the kernel by going to the "Kernel" menu and selecting "Restart."
- Clear Outputs: To reduce clutter and confusion, consider clearing the outputs of code cells that are no longer relevant. You can do this by selecting "Clear Outputs" from the "Edit" menu.
- Readability: Keep your code and comments clear and well-organized to make it easier to understand and maintain. Use markdown cells for explanations, headings, and documentation.
- Check Dependencies: If you're using external libraries or packages, ensure they are properly installed in your Jupyter environment. You can check the installed packages by running !pip list or !conda list in a code cell.
- Kernel Selection: Make sure you're using the correct kernel for your notebook. The kernel determines the programming language and environment in which your code runs. You can change the kernel by clicking on "Kernel" > "Change kernel" in the menu.
- Resource Usage: Be mindful of the resources your notebook is using, especially if you're working with large datasets or running intensive computations. Check system monitor tools to ensure you're not exhausting memory or CPU resources.

Troubleshooting:

- Syntax Errors: Check for syntax errors in your code. Python is sensitive to indentation and syntax, so ensure your code is properly formatted.
- Variable Scope: Be aware of variable scope issues, especially if you're reusing variable names or working with nested functions.
- Library Installation: If you encounter Module Not Found Error or similar errors, ensure that the required libraries are installed in your Jupyter environment. You can install libraries using !pip install <library> or !conda install <library> in a code cell.
- Kernel Crashes: If the kernel crashes frequently, consider reducing the complexity of your code or optimizing resource usage. Large datasets or intensive computations can sometimes overwhelm the kernel.
- Browser Issues: If you experience rendering or responsiveness issues in the notebook interface, try clearing your browser cache or using a different browser.
- Documentation: Consult the official Jupyter documentation and community forums for additional troubleshooting tips and solutions to common problems.

9.8 Observations

Observe the results obtained in each operation.

9.9 Calculations & Analysis

Calculations should be given for each operation.

9.10 Result & Interpretation

Result should be printed and pasted in laboratory copy found from Jupyter note book.

9.11 Follow-up Questions

- You need to check the relationship between the two variables. Which graph would you use?
- You need to check if a variable has outliers. Which graph would you use?
- You need to perform a univariate analysis. Which graph will you use?
- What is a data cleaning step?
- What are the ways to handle missing data?
- What are some of the methods for univariate analysis?
- What problems can outliers cause?

9.12 Extension and Follow-up Activities (if applicable)

NA

9.13 Assessments

9.14 Suggested reading

NA

Experiment No 10

10.1 Aim/Purpose of the Experiment

To familiarize the students with model evaluation methods using R square, Adjusted R square and VIF for a linear regression.

10.2 Learning Outcomes

Knowledge of the model evaluation methods using R square, Adjusted R square and VIF for a linear regression.in python.

10.3 Prerequisites

Basic knowledge of programming, python syntax, matplotlib, seaborn, different libraries.

10.4 Materials/Equipment/Apparatus / Devices/Software required

Jupyter Notebook.

10.5 Introduction and Theory

1. **R^2 (R-squared)**:
   - R^2 is a measure of how well the independent variables explain the variability of the dependent variable in the model.
   - It ranges from 0 to 1, with higher values indicating a better fit.
   - However, R^2 tends to increase as you add more independent variables to the model, even if they don't improve the model's predictive power.
   - Therefore, it's essential to use adjusted R^2 when comparing models with different numbers of predictors.
   - R-squared = (TSS-RSS)/TSS
   - TSS = Total variation in target variable is the sum of squares of the difference between the actual values and their mean.
   - RSS = Sum of square of the Residual obtained for a point where Residual in the data is the difference between the actual value and the value predicted by the linear regression model.
2. **Adjusted R^2 (adjusted R-squared)**:
   - Adjusted R^2 takes into account the number of predictors in the model and penalizes overly complex models.
   - It adjusts R^2 downward if adding more predictors doesn't significantly improve the model's fit.

- Like R^2, it ranges from 0 to 1, with higher values indicating a better fit.
- Adjusted R^2 is preferred over R^2 for comparing models with different numbers of predictors.

$$Adjusted\ R^2 = \{1 - [\frac{(1 - R^2)(n-1)}{(n-k-1)}]\}$$

- Here,
-
- n represents the number of data points in our dataset
- k represents the number of independent variables, and
- R represents the R-squared values determined by the model.

3. **Variance Inflation Factor (VIF)**:
   - VIF measures the extent to which the variance of an estimated regression coefficient increases because of multicollinearity.
   - Multicollinearity occurs when independent variables in a regression model are highly correlated with each other.
   - VIF values greater than 10 are often considered indicative of multicollinearity, although the threshold may vary depending on the context.
   - High VIF values suggest that the regression coefficients may be unstable due to multicollinearity, which can lead to unreliable estimates and inflated standard errors.

$$VIF_i = \frac{1}{1 - R_i^2} = \frac{1}{\text{Tolerance}}$$

-

To evaluate a linear regression model using these metrics in Python, you can follow these steps:

1. Fit the linear regression model using a library such as scikit-learn or statsmodels.
2. Calculate R^2 and adjusted R^2 to assess the goodness of fit.
3. Calculate VIF for each predictor variable to check for multicollinearity.

Housing Case Study
Problem Statement:
Consider a real estate company that has a dataset containing the prices of properties in the Delhi region. It wishes to use the data to optimise the sale prices of the properties based on important factors such as area, bedrooms, parking, etc.
Essentially, the company wants —
• To identify the variables affecting house prices, e.g. area, number of rooms, bathrooms, etc.
• To create a linear model that quantitatively relates house prices with variables such as number of rooms, area, number of bathrooms, etc.
• To know the accuracy of the model, i.e. how well these variables can predict house prices.
So interpretation is important!

Step 1: Reading and Understanding the Data

Let us first import NumPy and Pandas and read the housing dataset

```
# Supress Warnings
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
housing = pd.read_csv("Housing.csv")
# Check the head of the dataset
housing.head()

housing.shape

housing.info()

housing.describe()
```

Step 2: Visualising the Data

Let's now spend some time doing what is arguably the most important step - understanding the data.

- If there is some obvious multicollinearity going on, this is the first place to catch it
- Here's where you'll also identify if some predictors directly have a strong association with the outcome variable

We'll visualise our data using matplotlib and seaborn.

```
import matplotlib.pyplot as plt
import seaborn as sns
```

Visualising Numeric Variables

Let's make a pairplot of all the numeric variables

```
sns.pairplot(housing)
plt.show()
```

Visualising Categorical Variables

As you might have noticed, there are a few categorical variables as well. Let's make a boxplot for some of these variables.

```
plt.figure(figsize=(20, 12))
plt.subplot(2,3,1)
sns.boxplot(x = 'mainroad', y = 'price', data = housing)
plt.subplot(2,3,2)
sns.boxplot(x = 'guestroom', y = 'price', data = housing)
plt.subplot(2,3,3)
sns.boxplot(x = 'basement', y = 'price', data = housing)
plt.subplot(2,3,4)
sns.boxplot(x = 'hotwaterheating', y = 'price', data = housing)
plt.subplot(2,3,5)
sns.boxplot(x = 'airconditioning', y = 'price', data = housing)
plt.subplot(2,3,6)
sns.boxplot(x = 'furnishingstatus', y = 'price', data = housing)
```

```
plt.show()
```

We can also visualise some of these categorical features parallelly by using the hue argument. Below is the plot for furnishingstatus with airconditioning as the hue.

```
plt.figure(figsize = (10, 5))
sns.boxplot(x = 'furnishingstatus', y = 'price', hue =
'airconditioning', data = housing)
plt.show()
```

Step 3: Data Preparation

• You can see that your dataset has many columns with values as 'Yes' or 'No'.
• But in order to fit a regression line, we would need numerical values and not string.

Hence, we need to convert them to 1s and 0s, where 1 is a 'Yes' and 0 is a 'No'.

```
# List of variables to map
varlist = ['mainroad', 'guestroom', 'basement', 'hotwaterheating',
'airconditioning', 'prefarea']
# Defining the map function
def binary_map(x):
 return x.map({'yes': 1, "no": 0})
# Applying the function to the housing list
housing[varlist] = housing[varlist].apply(binary_map)
# Check the housing dataframe now
housing.head()
```

Dummy Variables

The variable furnishingstatus has three levels. We need to convert these levels into integer as well.

For this, we will use something called dummy variables.

```
# Get the dummy variables for the feature 'furnishingstatus' and store
it in a new variable - 'status'
status = pd.get_dummies(housing['furnishingstatus'])
# Check what the dataset 'status' looks like
status.head()
```

Now, you don't need three columns. You can drop the furnished column, as the type of furnishing can be identified with just the last two columns where —

• 00 will correspond to furnished
• 01 will correspond to unfurnished
• 10 will correspond to semi-furnished

```
# Let's drop the first column from status df using 'drop_first = True'
status = pd.get_dummies(housing['furnishingstatus'], drop_first =
True)
# Add the results to the original housing dataframe
housing = pd.concat([housing, status], axis = 1)
# Now let's see the head of our dataframe.
housing.head()
```

```
# Drop 'furnishingstatus' as we have created the dummies for it
housing.drop(['furnishingstatus'], axis = 1, inplace = True)
housing.head()
```

Step 4: Splitting the Data into Training and Testing Sets
As you know, the first basic step for regression is performing a train-test split.

```
from sklearn.model_selection import train_test_split
# We specify this so that the train and test data set always have the
same rows, respectively
np.random.seed(0)
df_train, df_test = train_test_split(housing, train_size = 0.7,
test_size = 0.3, random_state = 100)
```

Rescaling the Features
As you saw in the demonstration for Simple Linear Regression, scaling doesn't impact your model. Here we can see that except for area, all the columns have small integer values. So it is extremely important to rescale the variables so that they have a comparable scale. If we don't have comparable scales, then some of the coefficients as obtained by fitting the regression model might be very large or very small as compared to the other coefficients. This might become very annoying at the time of model evaluation. So it is advised to use standardization or normalization so that the units of the coefficients obtained are all on the same scale. As you know, there are two common ways of rescaling:
1. Min-Max scaling
2. Standardisation (mean-0, sigma-1)
This time, we will use MinMax scaling.

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
# Apply scaler() to all the columns except the 'yes-no' and 'dummy'
variables
num_vars = ['area', 'bedrooms', 'bathrooms', 'stories',
'parking','price']
df_train[num_vars] = scaler.fit_transform(df_train[num_vars])
df_train.head()
```

```
# Let's check the correlation coefficients to see which variables are
highly correlated
plt.figure(figsize = (16, 10))
sns.heatmap(df_train.corr(), annot = True, cmap="YlGnBu")
plt.show()
```

As you might have noticed, area seems to the correlated to price the most. Let's see a pairplot for area vs price.


```
plt.figure(figsize=[6,6])
plt.scatter(df_train.area, df_train.price)
plt.show()
```

So, we pick area as the first variable and we'll try to fit a regression line to that.
Dividing into X and Y sets for the model building

```
y_train = df_train.pop('price')
X_train = df_train
```

Step 5: Building a linear model
Fit a regression line through the training data using statsmodels. Remember that in statsmodels, you need to explicitly fit a constant using sm.add_constant(X) because if we don't perform this step, statsmodels fits a regression line passing through the origin, by default.

```
import statsmodels.api as sm
# Add a constant
X_train_lm = sm.add_constant(X_train[['area']])
# Create a first fitted model
lr = sm.OLS(y_train, X_train_lm).fit()
# Check the parameters obtained
lr.params

# Let's visualise the data with a scatter plot and the fitted
regression line
plt.scatter(X_train_lm.iloc[:, 1], y_train)
plt.plot(X_train_lm.iloc[:, 1], 0.127 + 0.462*X_train_lm.iloc[:, 1],
'r')
plt.show()


# Print a summary of the linear regression model obtained
print(lr.summary())
```

Adding another variable
The R-squared value obtained is 0.283. Since we have so many variables, we can clearly do better than this. So let's go ahead and add the second most highly correlated variable, i.e. bathrooms.

```
# Assign all the feature variables to X
X_train_lm = X_train[['area', 'bathrooms']]
# Build a linear model
import statsmodels.api as sm
X_train_lm = sm.add_constant(X_train_lm)
lr = sm.OLS(y_train, X_train_lm).fit()
lr.params

# Check the summary
print(lr.summary())
```

We have clearly improved the model as the value of adjusted R-squared as its value has gone up to 0.477 from 0.281. Let's go ahead and add another variable, bedrooms.

```
# Assign all the feature variables to X
X_train_lm = X_train[['area', 'bathrooms','bedrooms']]
# Build a linear model
import statsmodels.api as sm
X_train_lm = sm.add_constant(X_train_lm)
lr = sm.OLS(y_train, X_train_lm).fit()
lr.params

# Print the summary of the model
print(lr.summary())
```

Adding all the variables to the model

```
# Check all the columns of the dataframe
housing.columns
Index(['price', 'area', 'bedrooms', 'bathrooms', 'stories',
'mainroad',
 'guestroom', 'basement', 'hotwaterheating', 'airconditioning',
 'parking', 'prefarea', 'semi-furnished', 'unfurnished'],
 dtype='object')
#Build a linear model
import statsmodels.api as sm
X_train_lm = sm.add_constant(X_train)
lr_1 = sm.OLS(y_train, X_train_lm).fit()
lr_1.params

print(lr_1.summary())
```

Looking at the p-values, it looks like some of the variables aren't really significant (in the presence of other variables).
Maybe we could drop some?
We could simply drop the variable with the highest, non-significant p value. A better way would be to supplement this with the VIF information.
Checking VIF
Variance Inflation Factor or VIF, gives a basic quantitative idea about how much the feature variables are correlated with each other. It is an extremely important parameter to test our linear model. The formula for calculating VIF is:

$$ VIF_i = \frac{1}{1 - {R_i}^2} $$

```
# Check for the VIF values of the feature variables.
from statsmodels.stats.outliers_influence import
variance_inflation_factor
# Create a dataframe that will contain the names of all the feature
variables and their respective VIFs
vif = pd.DataFrame()
vif['Features'] = X_train.columns
vif['VIF'] = [variance_inflation_factor(X_train.values, i) for i in
```

```
range(X_train.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

10.6 Operating Procedure

- Open Jupyter note book
- Take a new python file
- Type the code
- Run it
- Take inputs from user
- Observe the results
- Verify the results manually
- Store the note book file

10.7 Precautions and/or Troubleshooting

Precautions:

- Save Your Work: Regularly save your Jupyter Notebook to avoid losing your work. You can save your notebook by clicking on the save icon or using the keyboard shortcut Ctrl + S (or Cmd + S on Mac).

- Restart Kernel: If you encounter unexpected behavior or errors, try restarting the kernel. This clears all the variables and imported modules, essentially resetting the notebook's state. You can restart the kernel by going to the "Kernel" menu and selecting "Restart."

- Clear Outputs: To reduce clutter and confusion, consider clearing the outputs of code cells that are no longer relevant. You can do this by selecting "Clear Outputs" from the "Edit" menu.

- Readability: Keep your code and comments clear and well-organized to make it easier to understand and maintain. Use markdown cells for explanations, headings, and documentation.

- Check Dependencies: If you're using external libraries or packages, ensure they are properly installed in your Jupyter environment. You can check the installed packages by running !pip list or !conda list in a code cell.

- Kernel Selection: Make sure you're using the correct kernel for your notebook. The kernel determines the programming language and environment in which your code runs. You can change the kernel by clicking on "Kernel" > "Change kernel" in the menu.

- Resource Usage: Be mindful of the resources your notebook is using, especially if you're working with large datasets or running intensive computations. Check system monitor tools to ensure you're not exhausting memory or CPU resources.

Troubleshooting:

- Syntax Errors: Check for syntax errors in your code. Python is sensitive to indentation and

syntax, so ensure your code is properly formatted.

- Variable Scope: Be aware of variable scope issues, especially if you're reusing variable names or working with nested functions.

- Library Installation: If you encounter Module Not Found Error or similar errors, ensure that the required libraries are installed in your Jupyter environment. You can install libraries using !pip install <library> or !conda install <library> in a code cell.

- Kernel Crashes: If the kernel crashes frequently, consider reducing the complexity of your code or optimizing resource usage. Large datasets or intensive computations can sometimes overwhelm the kernel.

- Browser Issues: If you experience rendering or responsiveness issues in the notebook interface, try clearing your browser cache or using a different browser.

- Documentation: Consult the official Jupyter documentation and community forums for additional troubleshooting tips and solutions to common problems.

10.8 Observations

Observe the results obtained in each operation.

10.9 Calculations & Analysis

Calculations should be given for each operation.

10.10 Result & Interpretation

Result should be printed and pasted in laboratory copy found from Jupyter note book.

10.11 Follow-up Questions

- You need to check the relationship between the two variables. Which graph would you use?
- You need to check if a variable has outliers. Which graph would you use?
- You need to perform a univariate analysis. Which graph will you use?
- What is a data cleaning step?
- What are the ways to handle missing data?
- What are some of the methods for univariate analysis?
- What problems can outliers cause?
- What is R2 square?
- What is adjusted R2square?
- What is VIF?

10.12 Extension and Follow-up Activities (if applicable)

NA

10.13 Assessments

10.14 Suggested reading

NA

Experiment No 11

11.1 Aim/Purpose of the Experiment

To familiarize the students with model evaluation methods using Confusion Matrix for Logistic Regression.

11.2 Learning Outcomes

Knowledge of the model evaluation methods using Confusion Matrix for Logistic Regression.in python.

11.3 Prerequisites

Basic knowledge of programming, python syntax, matplotlib, seaborn, different libraries.

11.4 Materials/Equipment/Apparatus / Devices/Software required

Jupyter Notebook.

11.5 Introduction and Theory

Telecom Churn Case Study
With 21 predictor variables we need to predict whether a particular customer will switch to
another telecom provider or not. In telecom terminology, this is referred to as churning and not
churning, respectively.
Step 1: Importing and Merging Data

```
# Suppressing Warnings
import warnings
warnings.filterwarnings('ignore')
# Importing Pandas and NumPy
import pandas as pd, numpy as np
# Importing all datasets
churn_data = pd.read_csv("churn_data.csv")
churn_data.head()
```

Combining all data files into one consolidated dataframe

```
# Merging on 'customerID'
df_1 = pd.merge(churn_data, customer_data, how='inner',
on='customerID')
```

```
# Final dataframe with all predictor variables
telecom = pd.merge(df_1, internet_data, how='inner', on='customerID')
```

Step 2: Inspecting the Dataframe

```
# Let's see the head of our master dataset
telecom.head()
```

```
# Let's check the dimensions of the dataframe
telecom.shape
```

```
# let's look at the statistical aspects of the dataframe
telecom.describe()
```

```
# Let's see the type of each column
telecom.info()
```

Step 3: Data Preparation

Converting some binary variables (Yes/No) to 0/1

```
# List of variables to map
varlist = ['PhoneService', 'PaperlessBilling', 'Churn', 'Partner',
'Dependents']
# Defining the map function
def binary_map(x):
 return x.map({'Yes': 1, "No": 0})
# Applying the function to the housing list
telecom[varlist] = telecom[varlist].apply(binary_map)
telecom.head()
```

For categorical variables with multiple levels, create dummy features (one-hot encoded)

```
# Creating a dummy variable for some of the categorical variables and
dropping the first one.
dummy1 = pd.get_dummies(telecom[['Contract', 'PaymentMethod',
'gender', 'InternetService']], drop_first=True)
# Adding the results to the master dataframe
telecom = pd.concat([telecom, dummy1], axis=1)
telecom.head()
```

```
# Creating dummy variables for the remaining categorical variables and
dropping the level with big names.
# Creating dummy variables for the variable 'MultipleLines'
ml = pd.get_dummies(telecom['MultipleLines'], prefix='MultipleLines')
# Dropping MultipleLines_No phone service column
ml1 = ml.drop(['MultipleLines_No phone service'], 1)
#Adding the results to the master dataframe
telecom = pd.concat([telecom,ml1], axis=1)
# Creating dummy variables for the variable 'OnlineSecurity'.
os = pd.get_dummies(telecom['OnlineSecurity'],
prefix='OnlineSecurity')
os1 = os.drop(['OnlineSecurity_No internet service'], 1)
# Adding the results to the master dataframe
telecom = pd.concat([telecom,os1], axis=1)
# Creating dummy variables for the variable 'OnlineBackup'.
ob = pd.get_dummies(telecom['OnlineBackup'], prefix='OnlineBackup')
ob1 = ob.drop(['OnlineBackup_No internet service'], 1)
# Adding the results to the master dataframe
telecom = pd.concat([telecom,ob1], axis=1)
# Creating dummy variables for the variable 'DeviceProtection'.
dp = pd.get_dummies(telecom['DeviceProtection'],
prefix='DeviceProtection')
dp1 = dp.drop(['DeviceProtection_No internet service'], 1)
# Adding the results to the master dataframe
telecom = pd.concat([telecom,dp1], axis=1)
# Creating dummy variables for the variable 'TechSupport'.
ts = pd.get_dummies(telecom['TechSupport'], prefix='TechSupport')
ts1 = ts.drop(['TechSupport_No internet service'], 1)
# Adding the results to the master dataframe
telecom = pd.concat([telecom,ts1], axis=1)
# Creating dummy variables for the variable 'StreamingTV'.
st =pd.get_dummies(telecom['StreamingTV'], prefix='StreamingTV')
st1 = st.drop(['StreamingTV_No internet service'], 1)
# Adding the results to the master dataframe
telecom = pd.concat([telecom,st1], axis=1)
# Creating dummy variables for the variable 'StreamingMovies'.
sm = pd.get_dummies(telecom['StreamingMovies'],
```

```
prefix='StreamingMovies')
sm1 = sm.drop(['StreamingMovies_No internet service'], 1)
# Adding the results to the master dataframe
telecom = pd.concat([telecom,sm1], axis=1)
telecom.head()

Dropping the repeated variables
# We have created dummies for the below variables, so we can drop them
telecom =
telecom.drop(['Contract','PaymentMethod','gender','MultipleLines','Int
ernetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection',
 'TechSupport', 'StreamingTV', 'StreamingMovies'], 1)
#The varaible was imported as a string we need to convert it to float
telecom['TotalCharges'] =
telecom['TotalCharges'].convert_objects(convert_numeric=True)
telecom.info()

Checking for Missing Values and Inputing Them
# Adding up the missing values (column-wise)
telecom.isnull().sum()


# Checking the percentage of missing values
round(100*(telecom.isnull().sum()/len(telecom.index)), 2)

# Removing NaN TotalCharges rows
telecom = telecom[~np.isnan(telecom['TotalCharges'])]
# Checking percentage of missing values after removing the missing
values
round(100*(telecom.isnull().sum()/len(telecom.index)), 2)


Step 4: Test-Train Split
from sklearn.model_selection import train_test_split
# Putting feature variable to X
X = telecom.drop(['Churn','customerID'], axis=1)
X.head()
```

```
# Putting response variable to y
y = telecom['Churn']
y.head()

# Splitting the data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y,
train_size=0.7, test_size=0.3, random_state=100)

Step 5: Feature Scaling
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train[['tenure','MonthlyCharges','TotalCharges']] =
scaler.fit_transform(X_train[['tenure','MonthlyCharges','TotalCharges'
]])
X_train.head()

### Checking the Churn Rate
churn = (sum(telecom['Churn'])/len(telecom['Churn'].index))*100
churn


Step 6: Looking at Correlations
# Importing matplotlib and seaborn
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
# Let's see the correlation matrix
plt.figure(figsize = (20,10)) # Size of the figure
sns.heatmap(telecom.corr(),annot = True)
plt.show()


Dropping highly correlated dummy variables
X_test =
X_test.drop(['MultipleLines_No','OnlineSecurity_No','OnlineBackup_No',
'DeviceProtection_No','TechSupport_No',
 'StreamingTV_No','StreamingMovies_No'], 1)
X_train =
X_train.drop(['MultipleLines_No','OnlineSecurity_No','OnlineBackup_No'
```

```
,'DeviceProtection_No','TechSupport_No',
 'StreamingTV_No','StreamingMovies_No'], 1)
```

Checking the Correlation Matrix

After dropping highly correlated variables now let's check the correlation matrix again.

```
plt.figure(figsize = (20,10))
sns.heatmap(X_train.corr(),annot = True)
plt.show()
```

Step 7: Model Building

Let's start by splitting our data into a training set and a test set.

Running Your First Training Model

```
import statsmodels.api as sm
# Logistic regression model
logm1 = sm.GLM(y_train,(sm.add_constant(X_train)), family =
sm.families.Binomial())
logm1.fit().summary()
```

Step 8: Feature Selection Using RFE

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
from sklearn.feature_selection import RFE
rfe = RFE(logreg, 15) # running RFE with 13 variables as
output
rfe = rfe.fit(X_train, y_train)
rfe.support_

col = X_train.columns[rfe.support_]
X_train.columns[~rfe.support_]
```

Assessing the model with StatsModels

```
X_train_sm = sm.add_constant(X_train[col])
logm2 = sm.GLM(y_train,X_train_sm, family = sm.families.Binomial())
res = logm2.fit()
res.summary()

# Getting the predicted values on the train set
```

```
y_train_pred = res.predict(X_train_sm)
y_train_pred[:10]

y_train_pred = y_train_pred.values.reshape(-1)
y_train_pred[:10]

Creating a dataframe with the actual churn flag and the predicted probabilities
y_train_pred_final = pd.DataFrame({'Churn':y_train.values,
'Churn_Prob':y_train_pred})
y_train_pred_final['CustID'] = y_train.index
y_train_pred_final.head()


Creating new column 'predicted' with 1 if Churn_Prob > 0.5 else 0
y_train_pred_final['predicted'] =
y_train_pred_final.Churn_Prob.map(lambda x: 1 if x > 0.5 else 0)
# Let's see the head
y_train_pred_final.head()

from sklearn import metrics
# Confusion matrix
confusion = metrics.confusion_matrix(y_train_pred_final.Churn,
y_train_pred_final.predicted )
print(confusion)

# Let's check the overall accuracy.
print(metrics.accuracy_score(y_train_pred_final.Churn,
y_train_pred_final.predicted))

# Let's take a look at the confusion matrix again
confusion = metrics.confusion_matrix(y_train_pred_final.Churn,
y_train_pred_final.predicted )
confusion

TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
# Let's see the sensitivity of our logistic regression model
```

```
TP / float(TP+FN)

# Let us calculate specificity
TN / float(TN+FP)


# Calculate false postive rate - predicting churn when customer does
not have churned
print(FP/ float(TN+FP))

# positive predictive value
print (TP / float(TP+FP))


# Negative predictive value
print (TN / float(TN+ FN))
```

11.6 Operating Procedure

- Open Jupyter note book
- Take a new python file
- Type the code
- Run it
- Take inputs from user
- Observe the results
- Verify the results manually
- Store the note book file

11.7 Precautions and/or Troubleshooting

Precautions:

- Save Your Work: Regularly save your Jupyter Notebook to avoid losing your work. You can save your notebook by clicking on the save icon or using the keyboard shortcut Ctrl + S (or Cmd + S on Mac).

- Restart Kernel: If you encounter unexpected behavior or errors, try restarting the kernel. This clears all the variables and imported modules, essentially resetting the notebook's state. You can restart the kernel by going to the "Kernel" menu and selecting "Restart."

- Clear Outputs: To reduce clutter and confusion, consider clearing the outputs of code cells that are no longer relevant. You can do this by selecting "Clear Outputs" from the "Edit" menu.

- Readability: Keep your code and comments clear and well-organized to make it easier to

understand and maintain. Use markdown cells for explanations, headings, and documentation.

- Check Dependencies: If you're using external libraries or packages, ensure they are properly installed in your Jupyter environment. You can check the installed packages by running !pip list or !conda list in a code cell.

- Kernel Selection: Make sure you're using the correct kernel for your notebook. The kernel determines the programming language and environment in which your code runs. You can change the kernel by clicking on "Kernel" > "Change kernel" in the menu.

- Resource Usage: Be mindful of the resources your notebook is using, especially if you're working with large datasets or running intensive computations. Check system monitor tools to ensure you're not exhausting memory or CPU resources.

Troubleshooting:

- Syntax Errors: Check for syntax errors in your code. Python is sensitive to indentation and syntax, so ensure your code is properly formatted.

- Variable Scope: Be aware of variable scope issues, especially if you're reusing variable names or working with nested functions.

- Library Installation: If you encounter Module Not Found Error or similar errors, ensure that the required libraries are installed in your Jupyter environment. You can install libraries using !pip install <library> or !conda install <library> in a code cell.

- Kernel Crashes: If the kernel crashes frequently, consider reducing the complexity of your code or optimizing resource usage. Large datasets or intensive computations can sometimes overwhelm the kernel.

- Browser Issues: If you experience rendering or responsiveness issues in the notebook interface, try clearing your browser cache or using a different browser.

- Documentation: Consult the official Jupyter documentation and community forums for additional troubleshooting tips and solutions to common problems.

11.8 Observations

Observe the results obtained in each operation.

11.9 Calculations & Analysis

Calculations should be given for each operation.

11.10 Result & Interpretation

Result should be printed and pasted in laboratory copy found from Jupyter note book.

11.11 Follow-up Questions

- What is Confusion matrix?
- What is sensitivity?
- What is specificity?
- What is TPR?
- What is FPR?
- What is precession ?
- What is recall?

11.12 Extension and Follow-up Activities (if applicable)

NA

11.13 Assessments

11.14 Suggested reading

NA

Experiment No 12

12.1 Aim/Purpose of the Experiment

To familiarize the students with model evaluation methods using Confusion Matrix for K-Means Clustering algorithm.

12.2 Learning Outcomes

Knowledge of the model evaluation methods using Confusion Matrix for K-Means clustering.in python.

12.3 Prerequisites

Basic knowledge of programming, python syntax, matplotlib, seaborn, different libraries.

12.4 Materials/Equipment/Apparatus / Devices/Software required

Jupyter Notebook.

12.5 Introduction and Theory

```
Importing Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
Importing Data
df = pd.read_csv('College_Data.csv')
Check the head of the data

df.head(3)

df.info()


df.isnull().sum()

df.describe()
```

EDA

```
sns.set_style('darkgrid')
sns.lmplot('Room.Board','Grad.Rate',data=df,
hue='Private',palette='magma_r',size=6,aspect=1,fit_reg=False)
```

scatterplot of Grad.Rate versus Room.Board where the points are colored by the Private column.

```
sns.set_style('darkgrid')
sns.lmplot('Outstate','F.Undergrad',data=df,
hue='Private',palette='magma_r',size=6,aspect=1,fit_reg=False)
```

scatterplot of F.Undergrad versus Outstate where the points are colored by the Private column.

```
sns.set_style('dark')
h = sns.FacetGrid(df,hue="Private",palette='coolwarm',size=6,aspect=2)
h = h.map(plt.hist,'Outstate',bins=20,alpha=0.7)
```

A stacked histogram showing Out of State Tuition based on the Private column.

```
sns.set_style('dark')
g = sns.FacetGrid(df,hue="Private",palette='coolwarm',size=6,aspect=2)
g = g.map(plt.hist,'Grad.Rate',bins=20,alpha=0.7)
```

A histogram for the Grad.Rate column.

We can see there seems to be a private school with a graduation rate of higher than 100%.

we need to set that school's graduation rate to 100 so it makes sense. we get a warning not an (error) , so we use dataframe operations and check the histogram visualization to make sure it actually changed.

```
df.set_value(95, 'Grad.Rate', 100)
```

```
df[df['Grad.Rate'] > 100]
```

```
sns.set_style('darkgrid')
g = sns.FacetGrid(df,hue="Private",palette='coolwarm',size=6,aspect=2)
g = g.map(plt.hist,'Grad.Rate',bins=20,alpha=0.7)
```

```
df=df.drop(['NameofInstitution',],axis=1)
```

Import KMeans from SciKit Learn.

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=2)
```

Fit the model to all the data except for the Private label.

```
kmeans.fit(df.drop('Private',axis=1))
```

The cluster center vectors for unlabeled data

```
means=kmeans.cluster_centers_
print(means)
```

Evaluation

There is no perfect way to evaluate clustering if we don't have the labels, however, we do have the labels, so we take advantage of this to evaluate our clusters.

Create a new column for df called 'Cluster', which is a 1 for a Private school, and a 0 for a public school.

```
def converter(cluster):
    if cluster=='Yes':
        return 1
    else:
        return 0
df['Cluster'] = df['Private'].apply(converter)
```

```
df.head(3)
```

```
df.Private.value_counts()
```

Creating a confusion matrix and classification report to see how well the Kmeans clustering worked without being given any labels.

```
from sklearn.metrics import confusion_matrix,classification_report
print(confusion_matrix(df['Cluster'],kmeans.labels_))
print(classification_report(df['Cluster'],kmeans.labels_))
```

12.6 Operating Procedure

- Open Jupyter note book
- Take a new python file
- Type the code
- Run it
- Take inputs from user
- Observe the results
- Verify the results manually
- Store the note book file

12.7 Precautions and/or Troubleshooting

Precautions:

- Save Your Work: Regularly save your Jupyter Notebook to avoid losing your work. You can save your notebook by clicking on the save icon or using the keyboard shortcut Ctrl + S (or Cmd + S on Mac).

- Restart Kernel: If you encounter unexpected behavior or errors, try restarting the kernel. This clears all the variables and imported modules, essentially resetting the notebook's state. You can restart the kernel by going to the "Kernel" menu and selecting "Restart."

- Clear Outputs: To reduce clutter and confusion, consider clearing the outputs of code cells that are no longer relevant. You can do this by selecting "Clear Outputs" from the "Edit" menu.

- Readability: Keep your code and comments clear and well-organized to make it easier to understand and maintain. Use markdown cells for explanations, headings, and documentation.

- Check Dependencies: If you're using external libraries or packages, ensure they are properly installed in your Jupyter environment. You can check the installed packages by running !pip list or !conda list in a code cell.

- Kernel Selection: Make sure you're using the correct kernel for your notebook. The kernel determines the programming language and environment in which your code runs. You can change the kernel by clicking on "Kernel" > "Change kernel" in the menu.

- Resource Usage: Be mindful of the resources your notebook is using, especially if you're working with large datasets or running intensive computations. Check system monitor tools to ensure you're not exhausting memory or CPU resources.

Troubleshooting:

- Syntax Errors: Check for syntax errors in your code. Python is sensitive to indentation and syntax, so ensure your code is properly formatted.
- Variable Scope: Be aware of variable scope issues, especially if you're reusing variable names or working with nested functions.
- Library Installation: If you encounter Module Not Found Error or similar errors, ensure that the required libraries are installed in your Jupyter environment. You can install libraries using !pip install <library> or !conda install <library> in a code cell.
- Kernel Crashes: If the kernel crashes frequently, consider reducing the complexity of your code or optimizing resource usage. Large datasets or intensive computations can sometimes overwhelm the kernel.
- Browser Issues: If you experience rendering or responsiveness issues in the notebook interface, try clearing your browser cache or using a different browser.
- Documentation: Consult the official Jupyter documentation and community forums for additional troubleshooting tips and solutions to common problems.

12.8 Observations

Observe the results obtained in each operation.

12.9 Calculations & Analysis

Calculations should be given for each operation.

12.10 Result & Interpretation

Result should be printed and pasted in laboratory copy found from Jupyter note book.

12.11 Follow-up Questions

- What is K means Clustering Algorithm?
- Is Feature Scaling required for the K means Algorithm?
- Why do you prefer Euclidean distance over Manhattan distance in the K means Algorithm?
- Which metrics can you use to find the accuracy of the K means Algorithm?
- What is a centroid point in K means Clustering?
- Does centroid initialization affect K means Algorithm?
- What are the advantages and disadvantages of the K means Algorithm?

12.12 Extension and Follow-up Activities (if applicable)

NA

12.13 Assessments

12.14 Suggested reading

NA

Experiment No 13

13.1 Aim/Purpose of the Experiment

To Obtain a Model for a real-life dataset (Bike Sharing data in Covid Time) in python using Linear Regression.

13.2 Learning Outcomes

Knowledge of the model formation, evaluation using Linear Regression.

13.3 Prerequisites

Basic knowledge of programming, python syntax, matplotlib, seaborn, different libraries.

13.4 Materials/Equipment/Apparatus / Devices/Software required

Jupyter Notebook.

13.5 Introduction and Theory

Problem Definition

A US bike-sharing provider BoomBikes has recently suffered considerable dips in their revenues due to the ongoing Corona pandemic. The company is finding it very difficult to sustain in the current market scenario. So, it has decided to come up with a mindful business plan to be able to accelerate its revenue as soon as the ongoing lockdown comes to an end, and the economy restores to a healthy state.

In such an attempt, BoomBikes aspires to understand the demand for shared bikes among the people after this ongoing quarantine situation ends across the nation due to Covid-19. They have planned this to prepare themselves to cater to the people's needs once the situation gets better all around and stand out from other service providers and make huge profits.

They have contracted a consulting company to understand the factors on which the demand for these shared bikes depends. Specifically, they want to understand the factors affecting the demand for these shared bikes in the American market. The company wants to know:

Which variables are significant in predicting the demand for shared bikes. How well those variables describe the bike demands Based on various meteorological surveys and

people's styles, the service provider firm has gathered a large dataset on daily bike demands across the American market based on some factors.

Step 1: Reading and Understanding the Data

```
#IMPORT NECESSARY LIBRARIES
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

#TO ASSIGN THE CSV FILE INTO A VARIABLE
BB = pd.read_csv("day.csv")

#HAVE LOOK AT DATA
BB.head()

#To check the columns in dataset
print(BB.columns)

#Get statistical describtion of data using describe()
BB.describe()

#To check null values in dataset
count=BB.isnull().sum()
print(count)

#To checkdatatypes of columns
BB.info()

: # To check the first 10 data in the dataset
BB.head(10)

#check for datatypes again
BB.info()
```

Step 2: Preparation of the Data

Few VERS like ‘instant’,‘dteday’,‘casual’,‘registered’ has no use in the analysis,

```
therefore We drop
those
BB.drop(['instant'],axis=1,inplace=True)
BB.drop(['dteday'],axis=1,inplace=True)
BB.drop(['casual','registered'],axis=1,inplace=True)

season,yr,mnth,holiday,weekday,workingday,weathersit are the categorical variables, so
we need to
replace with the appropiate values

BB['season'].replace({1:"spring",2:"summer",3:"fall",4:"winter"},inplace=True)
BB['weathersit'].replace({1:"Clear_Few  Clouds",2:"Mist_cloudy",3:"Light  rain_Light
snow_Thunderstorm",4:'Heavy Rain_Ice Pallets_Thunderstorm_Mist'},inplace=True)
BB['weekday'].replace({0:"Sunday",1:"Monday",2:"Tuesday",3:"Wednesday",4:"Thursd
ay",5:"Friday",6:"Saturday"},inplace=True)


#To checkdatatypes of columns
BB.info()

# To check the first 10 data in the dataset
BB.head(10)

#check for datatypes again
BB.info()


#changing datatypes of numerical columns to appropriate types

BB[['temp','atemp','hum','windspeed','cnt']]=BB[['temp','atemp','hum','windspeed','cnt']].a
pply(pd.to_numeric)


# To check the datatypes again

BB.info()

Step 2.1: Performing the EDA
PAIRPLOTS TO UNDERSTAND NUMERICAL VARIABLES
```

```
# To check the corellation between different numerical variables
sns.pairplot(BB, vars=['temp','atemp','hum','windspeed',"cnt"])
plt.show()

# To check the correlation
plt.figure(figsize = (16, 10))
sns.heatmap(BB.corr(), annot = True, cmap="YlGnBu")
plt.show()
```

From above two visualizations we can say that temp and atemp have a relationship with 0.99 value, so need to drop it

```
#To drop temp and consider atemp

BB.drop(['temp'],axis=1,inplace=True)
BB.head()


#To visualize the boxplot for categorical Variables
plt.figure(figsize=(30, 15))
plt.subplot(3,3,1)
sns.boxplot(x = 'season', y = 'cnt', data = BB)
plt.subplot(3,3,2)
sns.boxplot(x = 'yr', y = 'cnt', data = BB)
plt.subplot(3,3,3)
sns.boxplot(x = 'mnth', y = 'cnt', data = BB)
plt.subplot(3,3,4)
sns.boxplot(x = 'workingday', y = 'cnt', data = BB)
plt.subplot(3,3,5)
sns.boxplot(x = 'weathersit', y = 'cnt', data = BB)
plt.subplot(3,3,6)
sns.boxplot(x = 'weekday', y = 'cnt', data = BB)
plt.subplot(3,3,7)
sns.boxplot(x = 'holiday', y = 'cnt', data = BB)
plt.show()

#Convert some variables to object type
```

```
BB['mnth']=BB['mnth'].astype(object)
BB['season']=BB['season'].astype(object)
BB['weathersit']=BB['weathersit'].astype(object)
BB['weekday']=BB['weekday'].astype(object)
BB.info()
```

```
#TO CREAT DUMMY VARIABLES FOR CATEGORICAL DATA

Season_condition=pd.get_dummies(BB['season'],drop_first=True)
Weather_condition=pd.get_dummies(BB['weathersit'],drop_first=True)
Day_of_week=pd.get_dummies(BB['weekday'],drop_first=True)
Month=pd.get_dummies(BB['mnth'],drop_first=True)
```

```
# TO CONCATINATE THE CATEGORICAL VARIABLES ALONG WITH DUMMY VARS WITH THE DATASET
BB=pd.concat([BB,Season_condition],axis=1)
BB=pd.concat([BB,Weather_condition],axis=1)
BB=pd.concat([BB,Day_of_week],axis=1)
BB=pd.concat([BB,Month],axis=1)
BB.info()
```

```
#To delete the orginal columns season.weathersit,weekday,mnth
BB.drop(['season'],axis=1,inplace=True)
BB.drop(['weathersit'],axis=1,inplace=True)

BB.drop(['weekday'],axis=1,inplace=True)

BB.drop(['mnth'],axis=1,inplace=True)

BB.head()
```

Step 3: Rescaling the Data

```
# To split the dataset into test and train sets
from sklearn.model_selection import train_test_split
```

```
# To split the dataset as 70% train and 30% test set
np.random.seed(0)
BB_train, BB_test = train_test_split(BB, train_size = 0.7, test_size = 0.3, random_state = 100)

# To check the train set head
BB_train.head()


# To check the test head
BB_test.head()

# To find the name of all train columns
BB_train.columns


#IMPORTING THE LIBRARIES FOR SCALING THE NUMERICAL VERS WITH MIN AND MAX VALUES
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()

# Scaling the Numerical vars with min and max values
num_vars=['atemp','hum','windspeed','cnt']
BB_train[num_vars] = scaler.fit_transform(BB_train[num_vars])

# To check the head of the train dataset
BB_train.head()

# To check the description of training dataset
BB_train.describe()

# CREATING X AND y vers for analysing the data
y_train = BB_train.pop('cnt')
X_train = BB_train


# To check the head of the X_train dataset
X_train.head()
```

```
# To check the head of the y_train dataset
y_train.head()
```

Step 4: Performing the Linear Regression

```
# Importing the important libraries for feature selection
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression

#TO SELECT THE FEATURES USING RFE METHOD
# STARTING RFE WITH 15 VERS

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

rfe = RFE(lm, n_features_to_select= 15)
rfe = rfe.fit(X_train, y_train)

list(zip(X_train.columns,rfe.support_,rfe.ranking_))

# The columns that supoort the RFE
col = X_train.columns[rfe.support_]
col


# The columns that do not supoort the RFE
X_train.columns[~rfe.support_]


# To assign a new variable that support the RFE
X_train_rfe = X_train[col]


# To import the statmodels library and add one constant with X_train_rfe

import statsmodels.api as sm
```

```
# To assign a new VER named as X_train_rfe1
X_train_rfe1 = sm.add_constant(X_train_rfe)

# To build the model using statmodels
lm = sm.OLS(y_train,X_train_rfe1).fit()

# To print the summery of Linear Regrassion
print(lm.summary())


# To use of VIF in the analysis
from statsmodels.stats.outliers_influence import variance_inflation_factor

vif = pd.DataFrame()
X = X_train_rfe
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif

# To check the head of X_train_rfe1
X_train_rfe1.head()

# 'hum' HAS A VERY HIGH VIF SO WE DROP IT
X_train_rfe=X_train_rfe.drop(['hum'],axis=1)

# Fit a linear model using Ordinary Least Squares
lm1 = sm.OLS(y_train,X_train_rfe1).fit()

# To print the summery of Linear Regrassion
print(lm1.summary())


# To use of VIF in the analysis again
from statsmodels.stats.outliers_influence import variance_inflation_factor

vif = pd.DataFrame()
```

```
X = X_train_rfe
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

```
#COLUMN temp HAS A VERY HIGH VIF SO WE DROP IT
X_train_rfe=X_train_rfe.drop(['atemp'],axis=1)
```

```
# To build the model using statmodels again
X_train_rfe2 = sm.add_constant(X_train_rfe)
lm2 = sm.OLS(y_train,X_train_rfe2).fit()
print(lm2.summary())
```

```
# To use of VIF in the analysis again
vif = pd.DataFrame()
X = X_train_rfe
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

```
#COLUMN WINTER HAS A VERY HIGH P VALUE SO WE DROP IT
X_train_rfe=X_train_rfe.drop(['winter'],axis=1)
```

```
# To build the model using statmodels again
X_train_rfe3 = sm.add_constant(X_train_rfe)
lm3 = sm.OLS(y_train,X_train_rfe3).fit()
print(lm3.summary())
```

```
# To use of VIF in the analysis again
vif = pd.DataFrame()
X = X_train_rfe
```

```
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

```
#COLUMN 4 HAS A VERY HIGH P VALUE SO WE DROP IT
X_train_rfe=X_train_rfe.drop([4],axis=1)
```

```
# To build the model using statmodels again
X_train_rfe4 = sm.add_constant(X_train_rfe)
lm4 = sm.OLS(y_train,X_train_rfe4).fit()
print(lm4.summary())
```

```
# To use of VIF in the analysis again
vif = pd.DataFrame()
X = X_train_rfe
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

```
#ADDING WORKING DAY TO CHECK IF MODEL IMPROVES
X_train_rfe['workingday']=X_train['workingday']
X_train_rfe.head()
```

```
# To build the model using statmodels again
X_train_rfe5 = sm.add_constant(X_train_rfe)
lm5 = sm.OLS(y_train,X_train_rfe5).fit()
print(lm5.summary())
```

```
# To use of VIF in the analysis again
vif = pd.DataFrame()
X = X_train_rfe
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
```

```
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

```
#ADDING summer TO CHECK IF MODEL IMPROVES
X_train_rfe['summer']=X_train['summer']
X_train_rfe.head()
```

```
# To build the model using statmodels again
X_train_rfe6 = sm.add_constant(X_train_rfe)
lm6 = sm.OLS(y_train,X_train_rfe6).fit()
print(lm6.summary())
```

```
# To use of VIF in the analysis again
vif = pd.DataFrame()
X = X_train_rfe
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

```
#ADDING monday TO CHECK IF MODEL IMPROVES
X_train_rfe['Monday']=X_train['Monday']
X_train_rfe.head()
```

```
# To build the model using statmodels again
X_train_rfe7 = sm.add_constant(X_train_rfe)
lm7 = sm.OLS(y_train,X_train_rfe7).fit()
print(lm7.summary())
```

```
# To use of VIF in the analysis again
vif = pd.DataFrame()
X = X_train_rfe
```

```
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

```
#COLUMN MONDAY HAS A VERY HIGH P VALUE SO WE DROP IT
X_train_rfe=X_train_rfe.drop(['Monday'],axis=1)
```

```
#ADDING Saturday TO CHECK IF MODEL IMPROVES
X_train_rfe['Saturday']=X_train['Saturday']
X_train_rfe.head()
```

```
# To build the model using statmodels again
X_train_rfe8 = sm.add_constant(X_train_rfe)
lm8 = sm.OLS(y_train,X_train_rfe8).fit()
print(lm8.summary())
```

```
# To use of VIF in the analysis again
vif = pd.DataFrame()
X = X_train_rfe
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

```
#COLUMN Saturday HAS A VERY HIGH P VALUE SO WE DROP IT
X_train_rfe=X_train_rfe.drop(['Saturday'],axis=1)
```

```
#ADDING Sunday TO CHECK IF MODEL IMPROVES
X_train_rfe['Sunday']=X_train['Sunday']
X_train_rfe.head()
```

```
# To build the model using statmodels again
```

```
X_train_rfe9 = sm.add_constant(X_train_rfe)
lm9 = sm.OLS(y_train,X_train_rfe9).fit()
print(lm9.summary())
```

```
#COLUMN SUNDAY HAS A VERY HIGH P VALUE SO WE DROP IT
X_train_rfe=X_train_rfe.drop(['Sunday'],axis=1)
```

```
#ADDING Thursday TO CHECK IF MODEL IMPROVES
X_train_rfe['Thursday']=X_train['Thursday']
X_train_rfe.head()
```

```
# To build the model using statmodels again
X_train_rfe10 = sm.add_constant(X_train_rfe)
lm10 = sm.OLS(y_train,X_train_rfe10).fit()
print(lm10.summary())
```

```
# To use of VIF in the analysis again
vif = pd.DataFrame()
X = X_train_rfe
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

```
#ADDING Tuesday TO CHECK IF MODEL IMPROVES
X_train_rfe['Tuesday']=X_train['Tuesday']
X_train_rfe.head()
```

```
# To build the model using statmodels again
X_train_rfe11 = sm.add_constant(X_train_rfe)
lm11 = sm.OLS(y_train,X_train_rfe11).fit()
print(lm11.summary())
```

```
# To use of VIF in the analysis again
```

```
vif = pd.DataFrame()
X = X_train_rfe
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

```
#COLUMN TUESDAY HAS A VERY HIGH P VALUE SO WE DROP IT
X_train_rfe=X_train_rfe.drop(['Tuesday'],axis=1)
```

```
#ADDING Wednesday TO CHECK IF MODEL IMPROVES
X_train_rfe['Wednesday']=X_train['Wednesday']
X_train_rfe.head()
```

```
# To build the model using statmodels again
X_train_rfe12 = sm.add_constant(X_train_rfe)
lm12 = sm.OLS(y_train,X_train_rfe12).fit()
print(lm12.summary())
```

```
# To use of VIF in the analysis again
vif = pd.DataFrame()
X = X_train_rfe
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

```
#COLUMN Wednesday HAS A VERY HIGH P VALUE SO WE DROP IT
X_train_rfe=X_train_rfe.drop(['Wednesday'],axis=1)
```

```
#ADDING February Month TO CHECK IF MODEL IMPROVES
```

```
X_train_rfe[2]=X_train[2]
X_train_rfe.head()
```

```
# To build the model using statmodels again
X_train_rfe13 = sm.add_constant(X_train_rfe)
lm13 = sm.OLS(y_train,X_train_rfe13).fit()
print(lm13.summary())
```

```
# To use of VIF in the analysis again
vif = pd.DataFrame()
X = X_train_rfe
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

```
#COLUMN February month HAS A VERY HIGH P VALUE SO WE DROP IT
X_train_rfe=X_train_rfe.drop([2],axis=1)
```

```
#ADDING Month July TO CHECK IF MODEL IMPROVES
X_train_rfe[7]=X_train[7]
X_train_rfe.head()
```

```
# To build the model using statmodels again
X_train_rfe14 = sm.add_constant(X_train_rfe)
lm14 = sm.OLS(y_train,X_train_rfe14).fit()
print(lm14.summary())
```

```
# To use of VIF in the analysis again
vif = pd.DataFrame()
X = X_train_rfe
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
```

```
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

```
#ADDING Month October TO CHECK IF MODEL IMPROVES
X_train_rfe[10]=X_train[10]
X_train_rfe.head()
```

```
# To build the model using statmodels again
X_train_rfe15 = sm.add_constant(X_train_rfe)
lm15 = sm.OLS(y_train,X_train_rfe15).fit()
print(lm15.summary())
```

```
# To use of VIF in the analysis again
vif = pd.DataFrame()
X = X_train_rfe
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

```
#ADDING Month November TO CHECK IF MODEL IMPROVES
X_train_rfe[11]=X_train[11]
X_train_rfe.head()
```

```
# To build the model using statmodels again
X_train_rfe16 = sm.add_constant(X_train_rfe)
lm16 = sm.OLS(y_train,X_train_rfe16).fit()
print(lm16.summary())
```

```
# To use of VIF in the analysis again
vif = pd.DataFrame()
X = X_train_rfe
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
```

```
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

```
#COLUMN TUESDAY HAS A VERY HIGH P VALUE SO WE DROP IT
X_train_rfe=X_train_rfe.drop([11],axis=1)
```

```
#ADDING Month November TO CHECK IF MODEL IMPROVES
X_train_rfe[12]=X_train[12]
X_train_rfe.head()
```

```
# To build the model using statmodels again
X_train_rfe17 = sm.add_constant(X_train_rfe)
lm17 = sm.OLS(y_train,X_train_rfe17).fit()
print(lm17.summary())
```

```
# To use of VIF in the analysis again
vif = pd.DataFrame()
X = X_train_rfe
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

```
#COLUMN TUESDAY HAS A VERY HIGH P VALUE SO WE DROP IT
X_train_rfe=X_train_rfe.drop([12],axis=1)
```

```
X_train_rfe.head()
```

```
# To build the model using statmodels again
X_train_rfe18 = sm.add_constant(X_train_rfe)
lm18 = sm.OLS(y_train,X_train_rfe18).fit()
print(lm18.summary())
```

```
#COLUMN summer HAS A VERY HIGH P VALUE SO WE DROP IT
X_train_rfe=X_train_rfe.drop(['summer'],axis=1)
```

```
# To build the model using statmodels again
X_train_rfe19 = sm.add_constant(X_train_rfe)
lm19 = sm.OLS(y_train,X_train_rfe19).fit()
print(lm19.summary())

# To use of VIF in the analysis again
vif = pd.DataFrame()
X = X_train_rfe
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

We have considered all columns and checked.Now we stop the model building and check on which model can we choose Out all the models model lm19 seems to give good result so we choose it.

```
#Predict the values and store as a new vers y_train_cnt
y_train_cnt = lm19.predict(X_train_rfe14)

# Step 5: Performing the Residual Analysis

#CALCULATING RESIDUALS

res=y_train - y_train_cnt

#Checking ASSUMPTION OF NORMALITY:
# Plot the histogram of the error terms
fig = plt.figure()
sns.distplot((res), bins = 20)
fig.suptitle('Error Terms', fontsize = 20)                # Plot heading
plt.xlabel('Errors', fontsize = 18)                       # X-label
plt.show()
```

```
# Checking the pattern in residuals
plt.scatter(y_train,res)
plt.show()
```

There is no visible pattern present in residuals

Step 6: Evaluating the model with Test data

```
#Scaling the test data

num_vars=['atemp','hum','windspeed','cnt']
BB_test[num_vars] = scaler.fit_transform(BB_test[num_vars])

#Creating x and y sets

y_test = BB_test.pop('cnt')
X_test = BB_test

X_train_new=X_train_rfe19.drop(['const'], axis=1)

# Now let's use our model to make predictions.
# Creating X_test_new dataframe by dropping variables from X_test

X_test_new = X_test[X_train_new.columns]

# Adding a constant variable
X_test_new = sm.add_constant(X_test_new)

#Making predictions on the chosen model
```

```
y_pred = lm19.predict(X_test_new)
```

```
#CHECKING PREDICTED V/s TEST DATA
```

```
fig = plt.figure()
plt.scatter(y_test,y_pred)
fig.suptitle('y_test vs y_pred', fontsize=20)              # Plot heading
plt.xlabel('y_test', fontsize=18)                          # X-label
plt.ylabel('y_pred', fontsize=16)                          # Y-label
plt.show()
```

We have a model that seems good enough to predict demand of bikes. The actual and predicted cnt i.e demand significantly overlapped, thus indicating that the model is able to explain the change in demand very well.

Step 7: Final Evaluation of the Model

```
# Import mean_squared_error and r2_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

```
#Calculate the r square for test
```

```
r_squared = r2_score(y_test, y_pred)
r_squared
```

Problem Definition

A US bike-sharing provider BoomBikes has recently suffered considerable dips in their revenues due to the ongoing Corona pandemic. The company is finding it very difficult to sustain in the current market scenario. So, it has decided to come up with a mindful business plan to be able to accelerate its revenue as soon as the ongoing lockdown comes to an end, and the economy restores to a healthy state.

In such an attempt, BoomBikes aspires to understand the demand for shared bikes among the people after this ongoing quarantine situation ends across the nation due to Covid-19. They have planned this to prepare themselves to cater to the people's needs once the situation gets better all around and stand out from other service providers and make huge profits.

They have contracted a consulting company to understand the factors on which the demand for these shared bikes depends. Specifically, they want to understand the factors affecting the demand for these shared bikes in the American market. The company wants to know:

Which variables are significant in predicting the demand for shared bikes. How well those variables describe the bike demands Based on various meteorological surveys and people's styles, the service provider firm has gathered a large dataset on daily bike demands across the American market based on some factors.

Step 1: Reading and Understanding the Data

```
#IMPORT NECESSARY LIBRARIES
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
#TO ASSIGN THE CSV FILE INTO A VARIABLE
BB = pd.read_csv("day.csv")
#HAVE LOOK AT DATA
BB.head()
#To check the columns in dataset
print(BB.columns)
```

```
#Get statistical describtion of data using describe()
BB.describe()
#To check null values in dataset

count=BB.isnull().sum()
print(count)
#To checkdatatypes of columns
BB.info()
# To check the first 10 data in the dataset
BB.head(10)
#check for datatypes again
BB.info()
Step 2: Preparation of the Data
Few VERS like 'instant','dteday','casual','registered' has no use in the analysis, therefore We drop those

BB.drop(['instant'],axis=1,inplace=True)
BB.drop(['dteday'],axis=1,inplace=True)
BB.drop(['casual','registered'],axis=1,inplace=True)
season,yr,mnth,holiday,weekday,workingday,weathersit are the categorical variables, so we need to replace with the appropiate values

BB['season'].replace({1:"spring",2:"summer",3:"fall",4:"winter"},inplace=True)
BB['weathersit'].replace({1:"Clear_Few Clouds",2:"Mist_cloudy",3:"Light rain_Light snow_Thunderstorm",4:'Heavy Rain_Ice Pallets_Thunderstorm_Mist'},inplace=True)
BB['weekday'].replace({0:"Sunday",1:"Monday",2:"Tuesday",3:"Wednesday",4:"Thursday",5:"Friday",6:"Saturday"},inplace=True)

# To check the head of the dataset again
BB.head(10)
# To check the datatypes again
BB.info()
#changing datatypes of numerical columns to appropriate types

BB[['temp','atemp','hum','windspeed','cnt']]=BB[['temp','atemp','hum','windspeed','cnt']].apply(pd.to_numeric)

# To check the datatypes again
```

```
BB.info()
```

Step 2.1: Performing the EDA

PAIRPLOTS TO UNDERSTAND NUMERICAL VARIABLES

```
# To check the corellation between different numerical variables
sns.pairplot(BB, vars=['temp','atemp','hum','windspeed',"cnt"])
plt.show()
# To check the correlation
plt.figure(figsize = (16, 10))
sns.heatmap(BB.corr(), annot = True, cmap="YlGnBu")
plt.show()
```

From above two visualizations we can say that temp and atemp have a relationship with 0.99 value, so need to drop it

```
#To drop temp and consider atemp

BB.drop(['temp'],axis=1,inplace=True)
BB.head()
#To visualize the boxplot for categorical Variables
plt.figure(figsize=(30, 15))
plt.subplot(3,3,1)
sns.boxplot(x = 'season', y = 'cnt', data = BB)
plt.subplot(3,3,2)
sns.boxplot(x = 'yr', y = 'cnt', data = BB)
plt.subplot(3,3,3)
sns.boxplot(x = 'mnth', y = 'cnt', data = BB)
plt.subplot(3,3,4)
sns.boxplot(x = 'workingday', y = 'cnt', data = BB)
plt.subplot(3,3,5)
sns.boxplot(x = 'weathersit', y = 'cnt', data = BB)
plt.subplot(3,3,6)
sns.boxplot(x = 'weekday', y = 'cnt', data = BB)
plt.subplot(3,3,7)
sns.boxplot(x = 'holiday', y = 'cnt', data = BB)
plt.show()
#Convert some variables to object type
BB['mnth']=BB['mnth'].astype(object)
BB['season']=BB['season'].astype(object)
```

```
BB['weathersit']=BB['weathersit'].astype(object)
BB['weekday']=BB['weekday'].astype(object)
BB.info()
#TO CREAT DUMMY VARIABLES FOR CATEGORICAL DATA

Season_condition=pd.get_dummies(BB['season'],drop_first=True)
Weather_condition=pd.get_dummies(BB['weathersit'],drop_first=True)
Day_of_week=pd.get_dummies(BB['weekday'],drop_first=True)
Month=pd.get_dummies(BB['mnth'],drop_first=True)
# TO CONCATINATE THE CATEGORICAL VARIABLES ALONG WITH DUMMY VARS WITH THE DATASET
BB=pd.concat([BB,Season_condition],axis=1)
BB=pd.concat([BB,Weather_condition],axis=1)
BB=pd.concat([BB,Day_of_week],axis=1)
BB=pd.concat([BB,Month],axis=1)
BB.info()
#To delete the orginal columns season.weathersit,weekday,mnth
BB.drop(['season'],axis=1,inplace=True)
BB.drop(['weathersit'],axis=1,inplace=True)

BB.drop(['weekday'],axis=1,inplace=True)

BB.drop(['mnth'],axis=1,inplace=True)

BB.head()
Step 3: Rescaling the Data
# To split the dataset into test and train sets
from sklearn.model_selection import train_test_split

# To split the dataset as 70% train and 30% test set
np.random.seed(0)
BB_train, BB_test = train_test_split(BB, train_size = 0.7, test_size = 0.3, random_state = 100)

# To check the train set head
BB_train.head()
# To check the test head
BB_test.head()
# To find the name of all train columns
```

```
BB_train.columns


#IMPORTING THE LIBRARIES FOR SCALING THE NUMERICAL VERS WITH MIN AND MAX VALUES
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
# Scaling the Numerical vars with min and max values
num_vars=['atemp','hum','windspeed','cnt']
BB_train[num_vars] = scaler.fit_transform(BB_train[num_vars])
# To check the head of the train dataset
BB_train.head()
# To check the description of training dataset
BB_train.describe()
# CREATING X AND y vers for analysing the data
y_train = BB_train.pop('cnt')
X_train = BB_train
# To check the head of the X_train dataset
X_train.head()
# To check the head of the y_train dataset
y_train.head()
Step 4: Performing the Linear Regression
# Importing the important libraries for feature selection
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
#TO SELECT THE FEATURES USING RFE METHOD
# STARTING RFE WITH 15 VERS

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

rfe = RFE(lm, n_features_to_select= 15)
rfe = rfe.fit(X_train, y_train)
list(zip(X_train.columns,rfe.support_,rfe.ranking_))
# The columns that supoort the RFE
col = X_train.columns[rfe.support_]
col
# The columns that do not supoort the RFE
```

```
X_train.columns[~rfe.support_]
# To assign a new variable that support the RFE
X_train_rfe = X_train[col]
# To import the statmodels library and add one constant with X_train_rfe

import statsmodels.api as sm

# To assign a new VER named as X_train_rfe1
X_train_rfe1 = sm.add_constant(X_train_rfe)
# To build the model using statmodels
lm = sm.OLS(y_train,X_train_rfe1).fit()
# To print the summery of Linear Regrassion
print(lm.summary())
# To use of VIF in the analysis
from statsmodels.stats.outliers_influence import variance_inflation_factor

vif = pd.DataFrame()
X = X_train_rfe
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
# To check the head of X_train_rfe1
X_train_rfe1.head()
# 'hum' HAS A VERY HIGH VIF SO WE DROP IT
X_train_rfe=X_train_rfe.drop(['hum'],axis=1)
# Fit a linear model using Ordinary Least Squares
lm1 = sm.OLS(y_train,X_train_rfe1).fit()
# To print the summery of Linear Regrassion
print(lm1.summary())
# To use of VIF in the analysis again
from statsmodels.stats.outliers_influence import variance_inflation_factor

vif = pd.DataFrame()
X = X_train_rfe
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
```

```
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
#COLUMN temp HAS A VERY HIGH VIF SO WE DROP IT
X_train_rfe=X_train_rfe.drop(['atemp'],axis=1)
# To build the model using statmodels again
X_train_rfe2 = sm.add_constant(X_train_rfe)
lm2 = sm.OLS(y_train,X_train_rfe2).fit()
print(lm2.summary())
# To use of VIF in the analysis again
vif = pd.DataFrame()
X = X_train_rfe
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
#COLUMN WINTER HAS A VERY HIGH P VALUE SO WE DROP IT
X_train_rfe=X_train_rfe.drop(['winter'],axis=1)
# To build the model using statmodels again
X_train_rfe3 = sm.add_constant(X_train_rfe)
lm3 = sm.OLS(y_train,X_train_rfe3).fit()
print(lm3.summary())
# To use of VIF in the analysis again
vif = pd.DataFrame()
X = X_train_rfe
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
#COLUMN 4 HAS A VERY HIGH P VALUE SO WE DROP IT
X_train_rfe=X_train_rfe.drop([4],axis=1)
# To build the model using statmodels again
X_train_rfe4 = sm.add_constant(X_train_rfe)
lm4 = sm.OLS(y_train,X_train_rfe4).fit()
print(lm4.summary())
# To use of VIF in the analysis again
```

```
vif = pd.DataFrame()
X = X_train_rfe
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
#ADDING WORKING DAY TO CHECK IF MODEL IMPROVES
X_train_rfe['workingday']=X_train['workingday']
X_train_rfe.head()
# To build the model using statmodels again
X_train_rfe5 = sm.add_constant(X_train_rfe)
lm5 = sm.OLS(y_train,X_train_rfe5).fit()
print(lm5.summary())
# To use of VIF in the analysis again
vif = pd.DataFrame()
X = X_train_rfe
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
#ADDING summer TO CHECK IF MODEL IMPROVES
X_train_rfe['summer']=X_train['summer']
X_train_rfe.head()

# To build the model using statmodels again
X_train_rfe6 = sm.add_constant(X_train_rfe)
lm6 = sm.OLS(y_train,X_train_rfe6).fit()
print(lm6.summary())
# To use of VIF in the analysis again
vif = pd.DataFrame()
X = X_train_rfe
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

```
#ADDING monday TO CHECK IF MODEL IMPROVES
X_train_rfe['Monday']=X_train['Monday']
X_train_rfe.head()

# To build the model using statmodels again
X_train_rfe7 = sm.add_constant(X_train_rfe)
lm7 = sm.OLS(y_train,X_train_rfe7).fit()
print(lm7.summary())

# To use of VIF in the analysis again
vif = pd.DataFrame()
X = X_train_rfe
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
#COLUMN MONDAY HAS A VERY HIGH P VALUE SO WE DROP IT
X_train_rfe=X_train_rfe.drop(['Monday'],axis=1)
#ADDING Saturday TO CHECK IF MODEL IMPROVES
X_train_rfe['Saturday']=X_train['Saturday']
X_train_rfe.head()

# To build the model using statmodels again
X_train_rfe8 = sm.add_constant(X_train_rfe)
lm8 = sm.OLS(y_train,X_train_rfe8).fit()
print(lm8.summary())


# To use of VIF in the analysis again
vif = pd.DataFrame()
X = X_train_rfe
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
#COLUMN Saturday HAS A VERY HIGH P VALUE SO WE DROP IT
```

```
X_train_rfe=X_train_rfe.drop(['Saturday'],axis=1)
#ADDING Sunday TO CHECK IF MODEL IMPROVES
X_train_rfe['Sunday']=X_train['Sunday']
X_train_rfe.head()
# To build the model using statmodels again
X_train_rfe9 = sm.add_constant(X_train_rfe)
lm9 = sm.OLS(y_train,X_train_rfe9).fit()
print(lm9.summary())

#COLUMN SUNDAY HAS A VERY HIGH P VALUE SO WE DROP IT
X_train_rfe=X_train_rfe.drop(['Sunday'],axis=1)
#ADDING Thursday TO CHECK IF MODEL IMPROVES
X_train_rfe['Thursday']=X_train['Thursday']
X_train_rfe.head()

# To build the model using statmodels again
X_train_rfe10 = sm.add_constant(X_train_rfe)
lm10 = sm.OLS(y_train,X_train_rfe10).fit()
print(lm10.summary())

# To use of VIF in the analysis again
vif = pd.DataFrame()
X = X_train_rfe
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
#ADDING Tuesday TO CHECK IF MODEL IMPROVES
X_train_rfe['Tuesday']=X_train['Tuesday']
X_train_rfe.head()

# To build the model using statmodels again
X_train_rfe11 = sm.add_constant(X_train_rfe)
lm11 = sm.OLS(y_train,X_train_rfe11).fit()
print(lm11.summary())

# To use of VIF in the analysis again
```

```
vif = pd.DataFrame()
X = X_train_rfe
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
#COLUMN TUESDAY HAS A VERY HIGH P VALUE SO WE DROP IT
X_train_rfe=X_train_rfe.drop(['Tuesday'],axis=1)
#ADDING Wednesday TO CHECK IF MODEL IMPROVES
X_train_rfe['Wednesday']=X_train['Wednesday']
X_train_rfe.head()

# To build the model using statmodels again
X_train_rfe12 = sm.add_constant(X_train_rfe)
lm12 = sm.OLS(y_train,X_train_rfe12).fit()
print(lm12.summary())

# To use of VIF in the analysis again
vif = pd.DataFrame()
X = X_train_rfe
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
#COLUMN Wednesday HAS A VERY HIGH P VALUE SO WE DROP IT
X_train_rfe=X_train_rfe.drop(['Wednesday'],axis=1)
#ADDING February Month TO CHECK IF MODEL IMPROVES
X_train_rfe[2]=X_train[2]
X_train_rfe.head()


# To build the model using statmodels again
X_train_rfe13 = sm.add_constant(X_train_rfe)
lm13 = sm.OLS(y_train,X_train_rfe13).fit()
print(lm13.summary())
```

```
# To use of VIF in the analysis again
vif = pd.DataFrame()
X = X_train_rfe
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
#COLUMN February month HAS A VERY HIGH P VALUE SO WE DROP IT
X_train_rfe=X_train_rfe.drop([2],axis=1)
#ADDING Month July TO CHECK IF MODEL IMPROVES
X_train_rfe[7]=X_train[7]
X_train_rfe.head()


# To build the model using statmodels again
X_train_rfe14 = sm.add_constant(X_train_rfe)
lm14 = sm.OLS(y_train,X_train_rfe14).fit()
print(lm14.summary())

# To use of VIF in the analysis again
vif = pd.DataFrame()
X = X_train_rfe
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
#ADDING Month October TO CHECK IF MODEL IMPROVES
X_train_rfe[10]=X_train[10]
X_train_rfe.head()

# To build the model using statmodels again
X_train_rfe15 = sm.add_constant(X_train_rfe)
lm15 = sm.OLS(y_train,X_train_rfe15).fit()
print(lm15.summary())


# To use of VIF in the analysis again
```

```
vif = pd.DataFrame()
X = X_train_rfe
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
#ADDING Month November TO CHECK IF MODEL IMPROVES
X_train_rfe[11]=X_train[11]
X_train_rfe.head()


# To build the model using statmodels again
X_train_rfe16 = sm.add_constant(X_train_rfe)
lm16 = sm.OLS(y_train,X_train_rfe16).fit()
print(lm16.summary())

# To use of VIF in the analysis again
vif = pd.DataFrame()
X = X_train_rfe
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
#COLUMN TUESDAY HAS A VERY HIGH P VALUE SO WE DROP IT
X_train_rfe=X_train_rfe.drop([11],axis=1)
#ADDING Month November TO CHECK IF MODEL IMPROVES
X_train_rfe[12]=X_train[12]
X_train_rfe.head()

# To build the model using statmodels again
X_train_rfe17 = sm.add_constant(X_train_rfe)
lm17 = sm.OLS(y_train,X_train_rfe17).fit()
print(lm17.summary())

# To use of VIF in the analysis again
vif = pd.DataFrame()
```

```
X = X_train_rfe
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
#COLUMN TUESDAY HAS A VERY HIGH P VALUE SO WE DROP IT
X_train_rfe=X_train_rfe.drop([12],axis=1)
X_train_rfe.head()
# To build the model using statmodels again
X_train_rfe18 = sm.add_constant(X_train_rfe)
lm18 = sm.OLS(y_train,X_train_rfe18).fit()
print(lm18.summary())

#COLUMN summer HAS A VERY HIGH P VALUE SO WE DROP IT
X_train_rfe=X_train_rfe.drop(['summer'],axis=1)
# To build the model using statmodels again
X_train_rfe19 = sm.add_constant(X_train_rfe)
lm19 = sm.OLS(y_train,X_train_rfe19).fit()
print(lm19.summary())
# To use of VIF in the analysis again
vif = pd.DataFrame()
X = X_train_rfe
vif['Features'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

We have considered all columns and checked.Now we stop the model building and check on which model can we choose Out all the models model lm19 seems to give good result so we choose it.

```
#Predict the values and store as a new vers y_train_cnt
y_train_cnt = lm19.predict(X_train_rfe14)


# Step 5: Performing the Residual Analysis
```

```
#CALCULATING RESIDUALS

res=y_train - y_train_cnt
#Checking ASSUMPTION OF NORMALITY:
# Plot the histogram of the error terms
fig = plt.figure()
sns.distplot((res), bins = 20)
fig.suptitle('Error Terms', fontsize = 20)                  # Plot heading
plt.xlabel('Errors', fontsize = 18)                         # X-label
plt.show()
# Checking the pattern in residuals
plt.scatter(y_train,res)
plt.show()
There is no visible pattern present in residuals

Step 6: Evaluating the model with Test data
#Scaling the test data

num_vars=['atemp','hum','windspeed','cnt']
BB_test[num_vars] = scaler.fit_transform(BB_test[num_vars])

#Creating x and y sets

y_test = BB_test.pop('cnt')
X_test = BB_test

X_train_new=X_train_rfe19.drop(['const'], axis=1)
# Now let's use our model to make predictions.
# Creating X_test_new dataframe by dropping variables from X_test

X_test_new = X_test[X_train_new.columns]

# Adding a constant variable
X_test_new = sm.add_constant(X_test_new)
#Making predictions on the chosen model

y_pred = lm19.predict(X_test_new)
#CHECKING PREDICTED V/s TEST DATA
```

```
fig = plt.figure()
plt.scatter(y_test,y_pred)
fig.suptitle('y_test vs y_pred', fontsize=20)              # Plot heading
plt.xlabel('y_test', fontsize=18)                          # X-label
plt.ylabel('y_pred', fontsize=16)                          # Y-label
plt.show()
```

We have a model that seems good enough to predict demand of bikes. The actual and predicted cnt i.e demand significantly overlapped, thus indicating that the model is able to explain the change in demand very well.

Step 7: Final Evaluation of the Model

```
# Import mean_squared_error and r2_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

#Returns the mean squared error; we'll take a square root
np.sqrt(mean_squared_error(y_test, y_pred))

#Calculate the r square for test

r_squared = r2_score(y_test, y_pred)
r_squared
```

When we have time series data (e.g. yearly data), then the regression is likely to suffer from autocorrelation because demand next year will certainly be dependent on demand this year. Hence, error terms in different observations will surely be correlated with each other

```
# To check the head of the dataset of X_train_new
X_train_new.head()


# To print the all variables of final model
print(X_train_rfe19.columns)


# To build the FINAL model using statmodels
print(lm19.summary())
```

We can see that the equation for best fitted line is:

cnt = 0.4677 + (0.2471 * yr) - (0.0919 * holiday) - (windspeed * 0.1469) - (spring * 0.1945) - (Light rain_Light snow_Thunderstorm * 0.2992) - (Mist_cloudy * 0.0908) + (Mar * 0.0697) + (May * 0.1168) + (Jun * 0.1435) + (Aug * 0.1435) + (Sep * 0.1806) - (workingday * 0.0287) + (Thursday * 0.0272) + (Jul * 0.1123) + (Oct * 0.1091)

Where 3 = Mar , 5 = May, 6 = Jun, 7 = Jul, 8 = Aug, 9 = Sep, 10 = Oct

We can see the demand for bikes depends mainly on below variables:

yr , holiday ,windspeed, spring, Mist_Cloudy, Light rain_Light snow_Thunderstorm, Mar , May, Jun, Jul, Aug, Sep, Oct, workingday, Thursday,

Demands increases in the month of Mar , May, Jun, Jul, Aug, Sep, Oct

Demand decreases if it is holiday , windspeed, spring, Light rain_Light snow_Thunderstorm, Mist_cloudy, workingday

13.6 Operating Procedure

- Open Jupyter note book
- Take a new python file
- Type the code
- Run it
- Take inputs from user
- Observe the results
- Verify the results manually
- Store the note book file

13.7 Precautions and/or Troubleshooting

Precautions:

- Save Your Work: Regularly save your Jupyter Notebook to avoid losing your work. You can save your notebook by clicking on the save icon or using the keyboard shortcut Ctrl + S (or Cmd + S on Mac).

- Restart Kernel: If you encounter unexpected behavior or errors, try restarting the kernel. This clears all the variables and imported modules, essentially resetting the notebook's state. You can restart the kernel by going to the "Kernel" menu and selecting "Restart."

- Clear Outputs: To reduce clutter and confusion, consider clearing the outputs of code cells that are no longer relevant. You can do this by selecting "Clear Outputs" from the "Edit" menu.

- Readability: Keep your code and comments clear and well-organized to make it easier to understand and maintain. Use markdown cells for explanations, headings, and documentation.

- Check Dependencies: If you're using external libraries or packages, ensure they are properly installed in your Jupyter environment. You can check the installed packages by running !pip list or !conda list in a code cell.

- Kernel Selection: Make sure you're using the correct kernel for your notebook. The kernel determines the programming language and environment in which your code runs. You can change the kernel by clicking on "Kernel" > "Change kernel" in the menu.

- Resource Usage: Be mindful of the resources your notebook is using, especially if you're working with large datasets or running intensive computations. Check system monitor tools to ensure you're not exhausting memory or CPU resources.

Troubleshooting:

- Syntax Errors: Check for syntax errors in your code. Python is sensitive to indentation and syntax, so ensure your code is properly formatted.

- Variable Scope: Be aware of variable scope issues, especially if you're reusing variable names or working with nested functions.

- Library Installation: If you encounter Module Not Found Error or similar errors, ensure that the required libraries are installed in your Jupyter environment. You can install libraries using !pip install <library> or !conda install <library> in a code cell.

- Kernel Crashes: If the kernel crashes frequently, consider reducing the complexity of your code or optimizing resource usage. Large datasets or intensive computations can sometimes overwhelm the kernel.

- Browser Issues: If you experience rendering or responsiveness issues in the notebook interface, try clearing your browser cache or using a different browser.

- Documentation: Consult the official Jupyter documentation and community forums for additional troubleshooting tips and solutions to common problems.

13.8 Observations

Observe the results obtained in each operation.

13.9 Calculations & Analysis

Calculations should be given for each operation.

13.10 Result & Interpretation

Result should be printed and pasted in laboratory copy found from Jupyter note book.

13.11 Follow-up Questions

- Can you explain what linear regression is and how it is used in data analysis?
- In the context of a linear regression case study, what is the significance of the slope and intercept terms?
- How do you evaluate the goodness of fit of a linear regression model?
- What are some assumptions underlying linear regression models? How would you verify if these assumptions hold true for your data?
- Can you discuss multicollinearity and its impact on linear regression models? How would you address multicollinearity if it's present in your data?
- Describe the process you would follow to build a linear regression model for a given dataset.
- How do you interpret the coefficients of a linear regression model?
- What is the purpose of residual analysis in linear regression? How do you interpret residuals?
- How would you handle outliers in the dataset when performing linear regression analysis?
- What are some techniques for feature selection in linear regression? Can you explain the pros and cons of each method?
- What is the difference between simple linear regression and multiple linear regression? When would you choose one over the other?
- Can you explain the concept of regularization in the context of linear regression? How does it help in model improvement?
- How would you deal with heteroscedasticity in a linear regression model?
- Can you discuss the difference between R-squared and adjusted R-squared? When would you prefer to use adjusted R-squared?
- How do you handle categorical variables in a linear regression model?
- Can you explain the concept of overfitting in the context of linear regression? How would you prevent overfitting in your model?
- What are some common pitfalls or challenges when interpreting the results of a linear regression model?
- How would you validate the performance of your linear regression model? What metrics would you use, and why?
- Can you discuss the assumptions of normality in linear regression residuals? How would you assess if these assumptions are met?
- In what scenarios might linear regression not be an appropriate modeling technique? What alternative approaches could be considered?

13.12 Extension and Follow-up Activities (if applicable)

NA

13.13 Assessments

13.14 Suggested reading

NA

Experiment No 14

14.1 Aim/Purpose of the Experiment

To Obtain a Model for a real-life dataset (Bike Sharing data in Covid Time) in python using Logistic Regression.

14.2 Learning Outcomes

Knowledge of the model formation, evaluation using Logistic Regression.

14.3 Prerequisites

Basic knowledge of programming, python syntax, matplotlib, seaborn, different libraries.

14.4 Materials/Equipment/Apparatus / Devices/Software required

Jupyter Notebook.

14.5 Introduction and Theory

Problem Statement

In the telecom industry, customers are able to choose from multiple service providers and actively switch from one operator to another. In this highly competitive market, the telecommunications industry experiences an average of 15-25% annual churn rate. Given the fact that it costs 5-10 times more to acquire a new customer than to retain an existing one, customer retention has now become even more important than customer acquisition.

For many incumbent operators, retaining high profitable customers is the number one business goal.

To reduce customer churn, telecom companies need to predict which customers are at high risk of churn.

In this project, you will analyse customer-level data of a leading telecom firm, build predictive models to identify customers at high risk of churn and identify the main indicators of churn.

Step 1: Reading and Understanding the Data

```
# Suppressing Warnings
import warnings
warnings.filterwarnings('ignore')
# Importing Different libraries
```

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


# Importing the datasets
cd = pd.read_csv("churn_data.csv")
cd2 = pd.read_csv("customer_data.csv")
id = pd.read_csv("internet_data.csv")


# Checking the head of the datasets
cd.head(10)


# Checking the head of the datasets
cd2.head(10)


# Checking the head of the datasets
id.head(10)


# Merging all datasets in one common dataset on 'customerID'
cd3 = pd.merge(cd, cd2, how='inner', on='customerID')


# Merging all datasets in one common dataset on 'customerID'
cd4 = pd.merge(cd3, id, how='inner', on='customerID')
```

Step 2:understanding the structure of the merged dataset

```
#checking the head of the dataset
cd4.head(10)


# checking the shape of the dataframe
cd4.shape
```

```
# Checking at the statistical info of the dataframe
cd4.describe()
```

```
# Checkinge the type of each column
cd4.info()
```

Step 3: Data Preparation

```
# Extracting the 'object' data types only
cd5=cd4.select_dtypes(include='object')
cd5.columns
```

```
# Checking the unique value counts for object data types
cd5.nunique()
```

```
# Checking the columns which have only 2 unique values
cd6 =
cd4[["PhoneService","PaperlessBilling","Churn","gender","Partner","Dependents"]]

cd6.head()
```

```
# Convert 2 unique (YES/NO) object columns to 1/0 values.
# Converting Yes to 1 and No to 0
cd4['PhoneService'] = cd4['PhoneService'].map({'Yes': 1, 'No': 0})
cd4['PaperlessBilling'] = cd4['PaperlessBilling'].map({'Yes': 1, 'No': 0})
cd4['Churn'] = cd4['Churn'].map({'Yes': 1, 'No': 0})
cd4['Partner'] = cd4['Partner'].map({'Yes': 1, 'No': 0})
cd4['Dependents'] = cd4['Dependents'].map({'Yes': 1, 'No': 0})
```

```
cd7 =
```

```
cd5[["Contract","MultipleLines","InternetService","OnlineSecurity","OnlineBackup","DeviceProtection","TechSupport","StreamingTV","StreamingMovies"]]
cd7.head()


# Checking the columns which have only 3 unique values
for column in cd7:
    print(cd5[column].value_counts())

# Creating a dummy variable for the variable 'Contract' and dropping the first one.
cont = pd.get_dummies(cd4['Contract'],prefix='Contract',drop_first=True)
#Adding the results to the master dataframe
cd4 = pd.concat([cd4,cont],axis=1)

# Creating a dummy variable for the variable 'PaymentMethod' and dropping the first one.
pm = pd.get_dummies(cd4['PaymentMethod'],prefix='PaymentMethod',drop_first=True)
#Adding the results to the master dataframe
cd4 = pd.concat([cd4,pm],axis=1)

# Creating a dummy variable for the variable 'gender' and dropping the first one.
gen = pd.get_dummies(cd4['gender'],prefix='gender',drop_first=True)
#Adding the results to the master dataframe
cd4 = pd.concat([cd4,gen],axis=1)

# Creating a dummy variable for the variable 'MultipleLines' and dropping the first one.
ml = pd.get_dummies(cd4['MultipleLines'],prefix='MultipleLines')
#  dropping MultipleLines_No phone service column
ml1 = ml.drop(['MultipleLines_No phone service'],1)
#Adding the results to the master dataframe
cd4 = pd.concat([cd4,ml1],axis=1)

# Creating a dummy variable for the variable 'InternetService' and dropping the first one.
iser = pd.get_dummies(cd4['InternetService'],prefix='InternetService',drop_first=True)
#Adding the results to the master dataframe
cd4 = pd.concat([cd4,iser],axis=1)

# Creating a dummy variable for the variable 'OnlineSecurity'.
os = pd.get_dummies(cd4['OnlineSecurity'],prefix='OnlineSecurity')
```

```
os1= os.drop(['OnlineSecurity_No internet service'],1)
#Adding the results to the master dataframe
cd4 = pd.concat([cd4,os1],axis=1)

# Creating a dummy variable for the variable 'OnlineBackup'.
ob =pd.get_dummies(cd4['OnlineBackup'],prefix='OnlineBackup')
ob1 =ob.drop(['OnlineBackup_No internet service'],1)
#Adding the results to the master dataframe
cd4 = pd.concat([cd4,ob1],axis=1)

# Creating a dummy variable for the variable 'DeviceProtection'.
dp =pd.get_dummies(cd4['DeviceProtection'],prefix='DeviceProtection')
dp1 = dp.drop(['DeviceProtection_No internet service'],1)
#Adding the results to the master dataframe
cd4 = pd.concat([cd4,dp1],axis=1)

# Creating a dummy variable for the variable 'TechSupport'.
ts =pd.get_dummies(cd4['TechSupport'],prefix='TechSupport')
ts1 = ts.drop(['TechSupport_No internet service'],1)
#Adding the results to the master dataframe
cd4 = pd.concat([cd4,ts1],axis=1)

# Creating a dummy variable for the variable 'StreamingTV'.
st =pd.get_dummies(cd4['StreamingTV'],prefix='StreamingTV')
st1 = st.drop(['StreamingTV_No internet service'],1)
#Adding the results to the master dataframe
cd4 = pd.concat([cd4,st1],axis=1)

# Creating a dummy variable for the variable 'StreamingMovies'.
sm =pd.get_dummies(cd4['StreamingMovies'],prefix='StreamingMovies')
sm1 = sm.drop(['StreamingMovies_No internet service'],1)
#Adding the results to the master dataframe
cd4 = pd.concat([cd4,sm1],axis=1)

# Checking the head of the processed dataset
cd4.head()


# Droping the original columns on which dummies are created
```

```
cd4 = cd4.drop(['Contract','PaymentMethod','gender','MultipleLines','InternetService',
'OnlineSecurity', 'OnlineBackup', 'DeviceProtection',
       'TechSupport', 'StreamingTV', 'StreamingMovies'], 1)


# Checking the info of dataset
cd4.info()


# Dropping CustomerID as it is useless for model prediction
cd4 = cd4.drop(['customerID'], axis=1)
cd4.head()


# converting the datatype of 'MonthlyCharges','TotalCharges' from object to float
cd4.MonthlyCharges = cd4.MonthlyCharges.astype(float, errors="ignore")
cd4['TotalCharges'] = pd.to_numeric(cd4['TotalCharges'], errors = 'coerce')


# Checking the head of MonthlyCharges
cd4.MonthlyCharges.head()


# Checking the percentage of missing values
round(100*(cd4.isnull().sum()/len(cd4.index)), 2)

# Checking for outliers in the continuous variables
cd8 = cd4[['tenure','MonthlyCharges','SeniorCitizen','TotalCharges']]
sns.boxplot(data = cd8)
plt.show()


# There is sum NAN values in TotalCharge columns, so it should be dropped
cd4[['TotalCharges']].value_counts()
# Removing NaN TotalCharges rows
cd4 = cd4[~np.isnan(cd4['TotalCharges'])]
```

Step4: Model Building Preprocessing

```
# Importing test_train_split
from sklearn.model_selection import train_test_split
```

```
# Putting feature variables to X
X = cd4.drop(['Churn'], axis=1)
```

```
X.head()
```

```
# Putting response variable to y
y = cd4['Churn']
```

```
y.head()
```

```
# Splitting the data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, test_size=0.3, random_state=100)
```

```
import statsmodels.api as sm
```

```
# Logistic regression model
logm1 = sm.GLM(y_train,(sm.add_constant(X_train)), family = sm.families.Binomial())
logm1.fit().summary()
```

```
# Let's see the correlation matrix
plt.figure(figsize = (20,10))        # Size of the figure
sns.heatmap(cd4.corr(),annot = True)
```

```
## Dropping highly correlated variables.
```

```
X_test2 = X_test.drop(['MultipleLines_No','OnlineSecurity_No','OnlineBackup_No','DeviceProtection_No','TechSupport_No','StreamingTV_No','StreamingMovies_No'],1)
```

```
X_train2 = X_train.drop(['MultipleLines_No','OnlineSecurity_No','OnlineBackup_No','DeviceProtection_No','TechSupport_No','StreamingTV_No','StreamingMovies_No'],1)


plt.figure(figsize = (20,10))
sns.heatmap(X_train2.corr(),annot = True)
```

Step5: Model Building Using Logistic Regression

```
# # Logistic regression model 2
logm2 = sm.GLM(y_train,(sm.add_constant(X_train2)), family = sm.families.Binomial())
logm2.fit().summary()


from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
from sklearn.feature_selection import RFE
rfe = RFE(logreg, n_features_to_select = 13)          # running RFE with 13 variables as output
rfe = rfe.fit(X,y)
print(rfe.support_)          # Printing the boolean results
print(rfe.ranking_)          # Printing the ranking

# Variables selected by RFE
col = ['PhoneService', 'PaperlessBilling', 'Contract_One year', 'Contract_Two year',
       'PaymentMethod_Electronic check','MultipleLines_No','InternetService_Fiber optic', 'InternetService_No',

'OnlineSecurity_Yes','TechSupport_Yes','StreamingMovies_No','tenure','TotalCharges']


# Running the model using the selected variables
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
```

```
logsk = LogisticRegression(C=1e9)

logsk.fit(X_train, y_train)


#Comparing the model with StatsModels
logm4 = sm.GLM(y_train,(sm.add_constant(X_train)), family = sm.families.Binomial())
modres = logm4.fit()
logm4.fit().summary()


# Checking the shape of test dataset
X_test[col].shape


### Making Predictions

# Predicted probabilities
y_pred = logsk.predict_proba(X_test)
# Converting y_pred to a dataframe which is an array
y_pred_df = pd.DataFrame(y_pred)
# Converting to column dataframe
y_pred_1 = y_pred_df.iloc[:,[1]]
# Checking the head
y_pred_1.head()


# Converting y_test to dataframe
y_test_df = pd.DataFrame(y_test)
y_test_df.head()



# Putting CustID to index
y_test_df['CustID'] = y_test_df.index
# Removing index for both dataframes to append them side by side
y_pred_1.reset_index(drop=True, inplace=True)
y_test_df.reset_index(drop=True, inplace=True)
# Appending y_test_df and y_pred_1
```

```
y_pred_final = pd.concat([y_test_df,y_pred_1],axis=1)
# Renaming the column
y_pred_final= y_pred_final.rename(columns={ 1 : 'Churn_Prob'})
# Rearranging the columns
y_pred_final = y_pred_final.reindex(['CustID','Churn','Churn_Prob'], axis=1)
# Checking the head of y_pred_final
y_pred_final.head()



# Creating new column 'predicted' with 1 if Churn_Prob>0.5 else 0
y_pred_final['predicted'] = y_pred_final.Churn_Prob.map( lambda x: 1 if x > 0.5 else 0)
# Checking the head
y_pred_final.head()



# Model Evaluation

from sklearn import metrics

# Confusion matrix
confusion = metrics.confusion_matrix( y_pred_final.Churn, y_pred_final.predicted )
confusion

# Predicted     Churn  not_churn  __all__
# Actual
# Churn          1359   169     1528
# not_churn       256   326      582
# __all__        1615   751     2110

#Checking the overall accuracy
metrics.accuracy_score(y_pred_final.Churn, y_pred_final.predicted)

# Define the ROC
def draw_roc( actual, probs ):
    fpr, tpr, thresholds = metrics.roc_curve( actual, probs,
                                              drop_intermediate = False )
    auc_score = metrics.roc_auc_score( actual, probs )
```

```
plt.figure(figsize=(6, 6))
plt.plot( fpr, tpr, label='ROC curve (area = %0.2f)' % auc_score )
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate or [1 - True Negative Rate]')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```

```
#draw_roc(y_pred_final.Churn, y_pred_final.predicted)
"{:2.2f}".format(metrics.roc_auc_score(y_pred_final.Churn, y_pred_final.Churn_Prob))
```

```
# Checking the shape of the dataset
X_train.shape
```

14.6 Operating Procedure

- Open Jupyter note book
- Take a new python file
- Type the code
- Run it
- Take inputs from user
- Observe the results
- Verify the results manually
- Store the note book file

14.7 Precautions and/or Troubleshooting

Precautions:

- Save Your Work: Regularly save your Jupyter Notebook to avoid losing your work. You can save your notebook by clicking on the save icon or using the keyboard shortcut Ctrl + S (or Cmd + S on Mac).

- Restart Kernel: If you encounter unexpected behavior or errors, try restarting the kernel. This clears all the variables and imported modules, essentially resetting the notebook's state. You can restart the kernel by going to the "Kernel" menu and selecting "Restart."

- Clear Outputs: To reduce clutter and confusion, consider clearing the outputs of code cells that

are no longer relevant. You can do this by selecting "Clear Outputs" from the "Edit" menu.

- Readability: Keep your code and comments clear and well-organized to make it easier to understand and maintain. Use markdown cells for explanations, headings, and documentation.

- Check Dependencies: If you're using external libraries or packages, ensure they are properly installed in your Jupyter environment. You can check the installed packages by running !pip list or !conda list in a code cell.

- Kernel Selection: Make sure you're using the correct kernel for your notebook. The kernel determines the programming language and environment in which your code runs. You can change the kernel by clicking on "Kernel" > "Change kernel" in the menu.

- Resource Usage: Be mindful of the resources your notebook is using, especially if you're working with large datasets or running intensive computations. Check system monitor tools to ensure you're not exhausting memory or CPU resources.

Troubleshooting:

- Syntax Errors: Check for syntax errors in your code. Python is sensitive to indentation and syntax, so ensure your code is properly formatted.

- Variable Scope: Be aware of variable scope issues, especially if you're reusing variable names or working with nested functions.

- Library Installation: If you encounter Module Not Found Error or similar errors, ensure that the required libraries are installed in your Jupyter environment. You can install libraries using !pip install <library> or !conda install <library> in a code cell.

- Kernel Crashes: If the kernel crashes frequently, consider reducing the complexity of your code or optimizing resource usage. Large datasets or intensive computations can sometimes overwhelm the kernel.

- Browser Issues: If you experience rendering or responsiveness issues in the notebook interface, try clearing your browser cache or using a different browser.

- Documentation: Consult the official Jupyter documentation and community forums for additional troubleshooting tips and solutions to common problems.

14.8 Observations

Observe the results obtained in each operation.

14.9 Calculations & Analysis

Calculations should be given for each operation.

14.10 Result & Interpretation

Result should be printed and pasted in laboratory copy found from Jupyter note book.

14.11 Follow-up Questions

- Can you explain what logistic regression is and how it is used in data analysis?
- In the context of a logistic regression case study, what is the difference between odds and probability?
- How do you interpret the coefficients of a logistic regression model?
- What are some common applications of logistic regression in real-world scenarios?
- Describe the process you would follow to build a logistic regression model for a given dataset.
- What are the key assumptions of logistic regression? How would you verify if these assumptions hold true for your data?
- Can you discuss the concept of the logit function in logistic regression?
- How do you evaluate the performance of a logistic regression model? What metrics would you use, and why?
- What is the purpose of the threshold in logistic regression? How do you choose an appropriate threshold for classification?
- Can you explain the difference between binary logistic regression and multinomial logistic regression?
- How would you handle imbalanced classes in a logistic regression problem?
- What techniques can be used for feature selection in logistic regression? Can you discuss the pros and cons of each method?
- How do you deal with multicollinearity in logistic regression?
- Can you discuss the concept of regularization in logistic regression? How does it help in model improvement?
- What are some common pitfalls or challenges when interpreting the results of a logistic regression model?
- How would you handle missing data in a logistic regression analysis?
- Can you explain the concept of odds ratio in logistic regression? How is it useful in interpreting the relationship between predictors and the outcome variable?
- What are some alternatives to logistic regression for binary classification tasks?
- In what scenarios might logistic regression not be an appropriate modeling technique? What alternative approaches could be considered?
- Can you discuss the difference between logistic regression and linear regression? When would you choose one over the other?

14.12 Extension and Follow-up Activities (if applicable)

NA

14.13 Assessments

14.14 Suggested reading

NA

Experiment No 15

15.1 Aim/Purpose of the Experiment

To Obtain a Model for a real-life dataset (Bike Sharing data in Covid Time) in python using Logistic Regression.

15.2 Learning Outcomes

Knowledge of the model formation, evaluation using Logistic Regression.

15.3 Prerequisites

Basic knowledge of programming, python syntax, matplotlib, seaborn, different libraries.

15.4 Materials/Equipment/Apparatus / Devices/Software required

Jupyter Notebook.

15.5 Introduction and Theory

```
# Importing the required libraries
import pandas as pd, numpy as np
import matplotlib.pyplot as plt, seaborn as sns
%matplotlib inline

# Reading the csv file and putting it into 'df' object.
df = pd.read_csv('heart_v2.csv')

df.columns
```

```
Index(['age', 'sex', 'BP', 'cholestrol', 'heart disease'], dtype='object')
```

```
df.head()


# Putting feature variable to X
X = df.drop('heart disease',axis=1)

# Putting response variable to y
y = df['heart disease']
```

```
from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, random_state=42)
X_train.shape, X_test.shape

Fitting the decision tree with default hyperparameters, apart from max_depth which is 3
so that we can plot and read the tree.

from sklearn.tree import DecisionTreeClassifier

!pip install six

# Importing required packages for visualization
from IPython.display import Image
from six import StringIO
from sklearn.tree import export_graphviz
import pydotplus, graphviz


 plotting tree with max_depth=3
dot_data = StringIO()

export_graphviz(dt, out_file=dot_data, filled=True, rounded=True,
                feature_names=X.columns,
                class_names=['No Disease', "Disease"])

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
#Image(graph.create_png(),width=800,height=900)
#graph.write_pdf("dt_heartdisease.pdf")

y_train_pred = dt.predict(X_train)
y_test_pred = dt.predict(X_test)

from sklearn.metrics import confusion_matrix, accuracy_score

print(accuracy_score(y_train, y_train_pred))
confusion_matrix(y_train, y_train_pred)
```

```
print(accuracy_score(y_test, y_test_pred))
confusion_matrix(y_test, y_test_pred)
```

15.6 Operating Procedure

- Open Jupyter note book
- Take a new python file
- Type the code
- Run it
- Take inputs from user
- Observe the results
- Verify the results manually
- Store the note book file

15.7 Precautions and/or Troubleshooting

Precautions:

- Save Your Work: Regularly save your Jupyter Notebook to avoid losing your work. You can save your notebook by clicking on the save icon or using the keyboard shortcut Ctrl + S (or Cmd + S on Mac).

- Restart Kernel: If you encounter unexpected behavior or errors, try restarting the kernel. This clears all the variables and imported modules, essentially resetting the notebook's state. You can restart the kernel by going to the "Kernel" menu and selecting "Restart."

- Clear Outputs: To reduce clutter and confusion, consider clearing the outputs of code cells that are no longer relevant. You can do this by selecting "Clear Outputs" from the "Edit" menu.

- Readability: Keep your code and comments clear and well-organized to make it easier to understand and maintain. Use markdown cells for explanations, headings, and documentation.

- Check Dependencies: If you're using external libraries or packages, ensure they are properly installed in your Jupyter environment. You can check the installed packages by running !pip list or !conda list in a code cell.

- Kernel Selection: Make sure you're using the correct kernel for your notebook. The kernel determines the programming language and environment in which your code runs. You can change the kernel by clicking on "Kernel" > "Change kernel" in the menu.

- Resource Usage: Be mindful of the resources your notebook is using, especially if you're working with large datasets or running intensive computations. Check system monitor tools to ensure you're not exhausting memory or CPU resources.

Troubleshooting:

- Syntax Errors: Check for syntax errors in your code. Python is sensitive to indentation and syntax, so ensure your code is properly formatted.

- Variable Scope: Be aware of variable scope issues, especially if you're reusing variable names or working with nested functions.

- Library Installation: If you encounter Module Not Found Error or similar errors, ensure that the required libraries are installed in your Jupyter environment. You can install libraries using !pip install <library> or !conda install <library> in a code cell.

- Kernel Crashes: If the kernel crashes frequently, consider reducing the complexity of your code or optimizing resource usage. Large datasets or intensive computations can sometimes overwhelm the kernel.

- Browser Issues: If you experience rendering or responsiveness issues in the notebook interface, try clearing your browser cache or using a different browser.

- Documentation: Consult the official Jupyter documentation and community forums for additional troubleshooting tips and solutions to common problems.

15.8 Observations
Observe the results obtained in each operation.

15.9 Calculations & Analysis
Calculations should be given for each operation.

15.10 Result & Interpretation

Result should be printed and pasted in laboratory copy found from Jupyter note book.

15.11 Follow-up Questions
- Can you explain what a decision tree is and how it is used in machine learning?
- Describe the process of building a decision tree for a given dataset.
- What are the advantages and disadvantages of decision trees compared to other machine learning algorithms?
- How do decision trees handle both categorical and numerical data?
- What criteria are commonly used to split nodes in a decision tree?
- Can you explain the concept of entropy and information gain in the context of decision trees?
- How do you handle missing values in a dataset when using decision trees?
- What is the difference between a classification tree and a regression tree?
- How do you prevent overfitting when building a decision tree model?
- Can you discuss pruning techniques in decision trees? When is pruning necessary, and how does it improve model performance?
- How do you interpret the results of a decision tree model?
- What are ensemble methods, and how are they related to decision trees?
- Can you discuss the difference between bagging and boosting in the context of decision trees?
- What are some common algorithms used for building decision trees?
- How would you handle imbalanced classes in a classification problem using decision trees?
- What are some strategies for feature selection in decision tree-based models?

- Can you explain the concept of Gini impurity and how it is used in decision tree algorithms?
- How do decision trees handle nonlinear relationships between features and the target variable?
- In what scenarios might decision trees not be an appropriate modeling technique? What alternative approaches could be considered?
- Can you discuss the importance of hyperparameter tuning in optimizing a decision tree model?

15.12 Extension and Follow-up Activities (if applicable)

NA

15.13 Assessments

15.14 Suggested reading

NA