Neeraj Rijhwani **EXPERIMENT NO: 1** D15C-64

Aim: Introduction to Data science and Data preparation using Pandas steps.

Theory:

Data preparation is a crucial step in data science, involving cleaning and transforming raw data into an analyzable format. Using Pandas, we can perform operations such as handling missing values, encoding categorical data, and scaling numerical features. Proper preprocessing ensures the dataset is reliable for analysis and modeling by addressing inconsistencies, missing data, and outliers.

Problem Statement:

The Vehicle Safety Recall dataset, provided by NHTSA, contains 15 columns detailing various aspects of recall events, such as manufacturers, affected components, and corrective actions. This analysis focuses on:

- **Manufacturer Trends**: Identifying manufacturers prone to frequent recalls or specific defects.
- **Impact Analysis**: Understanding recall types affecting the largest populations and assessing average completion rates.
- **Temporal Patterns**: Detecting trends in recalls over time and seasonal spikes.
- **Safety Implications**: Investigating critical safety advisories like "Do Not Drive" or "Park Outside" and their resolution rates.

By cleaning the dataset and applying data preprocessing steps, the goal is to enhance its quality and draw actionable insights for stakeholders.

Dataset Overview:

The dataset provides detailed information about vehicle safety recalls managed by the National Highway Traffic Safety Administration (NHTSA). It contains 15 columns, each capturing specific aspects of recall events. Below is a breakdown of the columns and their relevance:

1. **Report Received Date:** Date the recall was officially reported.
2. **NHTSA ID:** A unique identifier for each recall event.
3. **Recall Link:** A hyperlink to the recall details on the NHTSA website.
4. **Manufacturer:** Name of the vehicle or product manufacturer responsible for the recall.
5. **Subject:** Brief description of the recall issue.

6. **Component:** The affected part of the vehicle/product (e.g., "POWER TRAIN").
7. **Mfr Campaign Number:** Manufacturer's internal reference for the recall.
8. **Recall Type:** Type of product involved (e.g., vehicle, tire, or car seat).
9. **Potentially Affected:** Number of units potentially impacted by the recall.
10. **Recall Description:** Detailed explanation of the defect or issue.
11. **Consequence Summary:** Description of the risks or consequences associated with the defect.
12. **Corrective Action:** Steps taken to address the defect.
13. **Park Outside Advisory:** Indicates whether there's an advisory to park outside for safety.
14. **Do Not Drive Advisory:** Indicates whether there's an advisory not to drive the affected vehicle.
15. **Completion Rate %:** Percentage of affected vehicles repaired or addressed.

Steps:

1. Loading The Dataset

```
[1]  import pandas as pd
```

```
[2]  df = pd.read_csv('recalls.csv')
```

2. Description of the dataset

a. Information about dataset

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28671 entries, 0 to 28670
Data columns (total 15 columns):
 #   Column                                  Non-Null Count  Dtype
---  ------                                  --------------  -----
 0   Report Received Date                    28671 non-null  object
 1   NHTSA ID                                28671 non-null  object
 2   Recall Link                             28671 non-null  object
 3   Manufacturer                            28671 non-null  object
 4   Subject                                 28671 non-null  object
 5   Component                               28671 non-null  object
 6   Mfr Campaign Number                     28624 non-null  object
 7   Recall Type                             28671 non-null  object
 8   Potentially Affected                    28630 non-null  float64
 9   Recall Description                      26270 non-null  object
 10  Consequence Summary                     23783 non-null  object
 11  Corrective Action                       26283 non-null  object
 12  Park Outside Advisory                   28671 non-null  object
 13  Do Not Drive Advisory                   28671 non-null  object
 14  Completion Rate % (Blank - Not Reported)  10007 non-null  float64
dtypes: float64(2), object(13)
memory usage: 3.3+ MB
```

b. Description of Dataset

```
# Get the dataset's shape and basic statistics
print(f"Dataset Shape: {df.shape}")
print(df.describe(include='all'))
```

```
Dataset Shape: (28671, 15)
       Report Received Date    NHTSA ID  \
count                 28671       28671
unique                10023       28671
top                10/17/2013  25E002000
freq                     42           1
mean                    NaN         NaN
std                     NaN         NaN
min                     NaN         NaN
25%                     NaN         NaN
50%                     NaN         NaN
75%                     NaN         NaN
max                     NaN         NaN
```

```
                                        Recall Link  \
count                                         28671
unique                                        28671
top     Go to Recall (https://www.nhtsa.gov/recalls?nh...
freq                                              1
mean                                            NaN
std                                             NaN
min                                             NaN
25%                                             NaN
50%                                             NaN
75%                                             NaN
max                                             NaN
```

```
       Mfr Campaign Number Recall Type  Potentially Affected  \
count                28624       28671          2.863000e+04
unique               11341           4                   NaN
top      NR (Not Reported)     Vehicle                   NaN
freq                 16602       24940                   NaN
mean                   NaN         NaN          4.572011e+04
std                    NaN         NaN          3.730381e+05
min                    NaN         NaN          0.000000e+00
25%                    NaN         NaN          9.900000e+01
50%                    NaN         NaN          6.860000e+02
75%                    NaN         NaN          6.385500e+03
max                    NaN         NaN          3.200000e+07
```

```
                                       Recall Description  \
count                                               26270
unique                                              25523
top     ON CERTAIN TRAILERS EQUIPPED WITH SEALCO SPRIN...
freq                                                   28
mean                                                  NaN
std                                                   NaN
min                                                   NaN
25%                                                   NaN
50%                                                   NaN
75%                                                   NaN
max                                                   NaN
```

```
                                     Consequence Summary  \
count                                              23783
unique                                             17015
top     RELEASE OF COOLANT UNDER CERTAIN CONDITIONS CO...
freq                                                 128
mean                                                 NaN
std                                                  NaN
min                                                  NaN
25%                                                  NaN
50%                                                  NaN
75%                                                  NaN
max                                                  NaN
```

```
                                      Corrective Action  \
count                                             26283
unique                                            25579
top     DEALERS WILL EQUIP AIR SYSTEMS WITH A PRESSURE...
freq                                                 18
mean                                                NaN
std                                                 NaN
min                                                 NaN
25%                                                 NaN
50%                                                 NaN
75%                                                 NaN
max                                                 NaN
```

```
       Park Outside Advisory  Do Not Drive Advisory  \
count                  28671                  28671
unique                     2                      2
top                       No                     No
freq                   28601                  28510
mean                     NaN                    NaN
std                      NaN                    NaN
min                      NaN                    NaN
25%                      NaN                    NaN
50%                      NaN                    NaN
75%                      NaN                    NaN
max                      NaN                    NaN
```

```
       Completion Rate % (Blank - Not Reported)
count                              10007.000000
unique                                      NaN
top                                         NaN
freq                                        NaN
mean                                  67.874214
std                                   29.937993
min                                    0.000000
25%                                   48.350000
50%                                   76.390000
75%                                   93.765000
max                                  100.000000
```

3. Drop columns that aren't useful.

Columns that might not be necessary for analysis include Recall Link, Mfr Campaign Number, Park Outside Advisory,Completion rate(%).These columns do not provide much insight in the context of data analysis for recall trends or consequences. Therefore, you can drop them to simplify the dataset.

```
# Remove leading/trailing spaces from column names
df.columns = df.columns.str.strip()

# List of columns to drop
cols = ["Recall Link", "Mfr Campaign Number","Park Outside Advisory", "Do Not Drive Advisory", "Completion Rate % (Blank - Not Reported)"]
```

```
# Drop the columns that are present in the DataFrame
df = df.drop(cols, axis=1)

# Display the updated DataFrame
print(df.head())
```

```
  Report Received Date   NHTSA ID               Manufacturer  \
0           01/14/2025  25E002000             GKN Automotive
1           01/13/2025  25E001000  N&B Mobility Solutions LLC
2           01/13/2025  25V005000          Forest River, Inc.
3           01/13/2025  25V006000           Kia America, Inc.
4           01/13/2025  25V007000  Winnebago Industries, Inc.

                                      Subject          Component  \
0                            Driveshaft Can Break        POWER TRAIN
1    Charger Adapter May Cause Arcing or Shock Risk  ELECTRICAL SYSTEM
2  Cooktop Burner Tube May Crack and Cause Gas Leak          EQUIPMENT
3      Loss of Headlights and Taillights/FMVSS 108  ELECTRICAL SYSTEM
4                    Spare Tire Carrier May Detach          EQUIPMENT

  Recall Type  Potentially Affected  \
0   Equipment                 18.0
1   Equipment                130.0
2     Vehicle                396.0
3     Vehicle              74469.0
4     Vehicle                107.0
```

```
                                    Recall Description  \
0  GKN Automotive (GKN) is recalling certain repl...
1  N&B Mobility Solutions LLC (Nivion) is recalli...
2  Forest River, Inc. (Forest River) is recalling...
3  Kia America, Inc. (Kia) is recalling certain 2...
4  Winnebago Industries, Inc. (Winnebago) is reca...

                                 Consequence Summary  \
0  A cracked or broken driveshaft can cause a los...
1  Inadequate clearance between DC busbars may ca...
2  A gas leak in the presence of an ignition sour...
3  A loss of headlights and taillights can reduce...
4  A detached spare tire carrier can become a roa...

                                  Corrective Action
0  GKN will reimburse the cost of a replacement d...
1  Nivion will replace the defective adapters, fr...
2  Owners are advised not to use the cooktop unti...
3  Dealers will update the BDC software, free of ...
4  Dealers will inspect, replace, and correctly t...
```

Thus the columns now present in dataset are:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28671 entries, 0 to 28670
Data columns (total 10 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Report Received Date  28671 non-null  object
 1   NHTSA ID              28671 non-null  object
 2   Manufacturer          28671 non-null  object
 3   Subject               28671 non-null  object
 4   Component             28671 non-null  object
 5   Recall Type           28671 non-null  object
 6   Potentially Affected  28630 non-null  float64
 7   Recall Description    26270 non-null  object
 8   Consequence Summary   23783 non-null  object
 9   Corrective Action     26283 non-null  object
dtypes: float64(1), object(9)
memory usage: 2.2+ MB
```

4. **Take care of missing data.**
    a. **Drop rows with maximum missing values.**

Here we drop the rows which have more than 50% missing values.these can be done by dropna() function with threshold parameter=0.5.

```
print(f"Dataset Shape before Dropping Rows: {df.shape}")
# Drop rows with the highest number of missing values
threshold = len(df.columns) * 0.5  # Drop rows where over 50% of columns are missing
df = df.dropna(thresh=threshold)

print(f"Dataset Shape After Dropping Rows: {df.shape}")
```

```
Dataset Shape before Dropping Rows: (28671, 10)
Dataset Shape After Dropping Rows: (28671, 10)
```

```
print(df.isnull().sum())
```

```
Report Received Date       0
NHTSA ID                   0
Manufacturer               0
Subject                    0
Component                  0
Recall Type                0
Potentially Affected      41
Recall Description      2401
Consequence Summary     4888
Corrective Action       2388
dtype: int64
```

b. Handle Missing Data

Here the above information says Potential Affected ,Recall Description ,Consequence Summary and corrective action contain some null values thus we need to handle missing data.For these columns, either fill in with a placeholder (e.g., "Unknown") or drop the rows if the missing data is significant.

```
[12] # Fill missing numerical values with the median
df['Potentially Affected'] = df['Potentially Affected'].fillna(df['Potentially Affected'].median())
# Fill missing categorical values with a placeholder
df['Recall Description'] = df['Recall Description'].fillna('Not Known')
df['Consequence Summary'] = df['Consequence Summary'].fillna('Unknown')
df['Corrective Action'] = df['Corrective Action'].fillna('Unknown')

print(df.isnull().sum())  # Verify no missing values remain
```

```
Report Received Date    0
NHTSA ID                0
Manufacturer            0
Subject                 0
Component               0
Recall Type             0
Potentially Affected    0
Recall Description      0
Consequence Summary     0
Corrective Action       0
dtype: int64
```

5. Create dummy variables

For columns containing categorical data (e.g., Recall Type), we can create dummy variables. This is helpful for machine learning models.

```
# Convert categorical columns into dummy variables
df = pd.get_dummies(df, columns=['Recall Type'], drop_first=True)

print(df.head())
```

```
  Report Received Date   NHTSA ID               Manufacturer  \
0          01/14/2025  25E002000              GKN Automotive
1          01/13/2025  25E001000  N&B Mobility Solutions LLC
2          01/13/2025  25V005000         Forest River, Inc.
3          01/13/2025  25V006000          Kia America, Inc.
4          01/13/2025  25V007000  Winnebago Industries, Inc.

                                    Subject          Component  \
0                          Driveshaft Can Break        POWER TRAIN
1   Charger Adapter May Cause Arcing or Shock Risk  ELECTRICAL SYSTEM
2  Cooktop Burner Tube May Crack and Cause Gas Leak          EQUIPMENT
3      Loss of Headlights and Taillights/FMVSS 108  ELECTRICAL SYSTEM
4                    Spare Tire Carrier May Detach          EQUIPMENT

   Potentially Affected                                Recall Description  \
0                  18.0  GKN Automotive (GKN) is recalling certain repl...
1                 130.0  N&B Mobility Solutions LLC (Nivion) is recalli...
2                 396.0  Forest River, Inc. (Forest River) is recalling...
3               74469.0  Kia America, Inc. (Kia) is recalling certain 2...
4                 107.0  Winnebago Industries, Inc. (Winnebago) is reca...
```

```
                                Consequence Summary  \
0  A cracked or broken driveshaft can cause a los...
1  Inadequate clearance between DC busbars may ca...
2  A gas leak in the presence of an ignition sour...
3  A loss of headlights and taillights can reduce...
4  A detached spare tire carrier can become a roa...

                                    Corrective Action  Recall Type_Equipment  \
0  GKN will reimburse the cost of a replacement d...                   True
1  Nivion will replace the defective adapters, fr...                   True
2  Owners are advised not to use the cooktop unti...                  False
3  Dealers will update the BDC software, free of ...                  False
4  Dealers will inspect, replace, and correctly t...                  False

   Recall Type_Tire  Recall Type_Vehicle
0             False                False
1             False                False
2             False                 True
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28671 entries, 0 to 28670
Data columns (total 12 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Report Received Date    28671 non-null  object
 1   NHTSA ID                28671 non-null  object
 2   Manufacturer            28671 non-null  object
 3   Subject                 28671 non-null  object
 4   Component               28671 non-null  object
 5   Potentially Affected    28671 non-null  float64
 6   Recall Description      28671 non-null  object
 7   Consequence Summary     28671 non-null  object
 8   Corrective Action       28671 non-null  object
 9   Recall Type_Equipment   28671 non-null  bool
 10  Recall Type_Tire        28671 non-null  bool
 11  Recall Type_Vehicle     28671 non-null  bool
dtypes: bool(3), float64(1), object(8)
memory usage: 2.1+ MB
```

6. Find out outliers (manually):

Outliers can be detected by looking at numerical columns like Potentially Affected. One method for identifying outliers is by visualizing the data using box plots or using statistical methods like the Z-score.

First Quartile (Q1):=QUARTILE(H2:H28719, 1)
Q1 = 99
Third Quartile (Q3):=QUARTILE(H2:H28719, 3)
Q3 = 6386
Interquartile Range (IQR):=Q3 - Q1

IQR = 6386 - 99 = 6287

Outlier Boundaries:

Lower Bound:=Q1 - 1.5 * IQR

Lower Bound = 99 - (1.5 * 6287) = -9331.5

Upper Bound: =Q3 + 1.5 * IQR

Upper Bound = 6386 + (1.5 * 6287) = 15816.5

Identifying Outliers: Any value less than -9331.5 or greater than 15816.5 is considered an outlier.

01/31/2025	25V048000	Ford Motor Company	BACK OVER PREVENTIO	25S05	Vehicle	72624
01/30/2025	25V043000	Jayco, Inc.	EQUIPMENT	9901617	Vehicle	412
01/30/2025	25V045000	Autocar, LLC	ELECTRICAL SYSTEM	ACTT-2501	Vehicle	130
01/28/2025	25V037000	Mitsubishi Fuso Truck of Am	ELECTRICAL SYSTEM	C10129	Vehicle	233
01/24/2025	25V034000	Forest River, Inc.	EQUIPMENT	203-1889	Vehicle	64
01/23/2025	25V029000	Winnebago Towable	EQUIPMENT	CAM0000041	Vehicle	144
01/23/2025	25V031000	Honda (American Honda Mo	ELECTRICAL SYSTEM	EL1, AL0	Vehicle	294612
01/23/2025	25V033000	Subaru of America, Inc.	WHEELS	WRB-25	Vehicle	20366
01/23/2025	25V030000	Mack Trucks, Inc.	SERVICE BRAKES, AIR	SC0474	Vehicle	142
01/23/2025	25V032000	Honda (American Honda Mo	BACK OVER PREVENTIO	RKZ	Vehicle	9221
01/22/2025	25V027000	Forest River Bus, LLC	STRUCTURE	05-1890	Vehicle	37
01/22/2025	25V028000	Toyota Motor Engineering &	FUEL SYSTEM, GASOLIN	25TA01 / 25LA01	Vehicle	858
01/21/2025	25V026000	Mack Trucks, Inc.	SERVICE BRAKES, AIR	SC0473	Vehicle	21
01/21/2025	25E006000	Oshkosh Corporation	VEHICLE SPEED CONTR	NR (Not Reported)	Equipment	500
01/17/2025	25V021000	Forest River, Inc.	EQUIPMENT	503-1887	Vehicle	18
01/17/2025	25E004000	Cummins, Inc.	FUEL SYSTEM, DIESEL	C7111	Equipment	715
01/17/2025	25V024000	Kia America, Inc.	ELECTRICAL SYSTEM	SC332	Vehicle	80255
01/17/2025	25T001000	Pirelli Tire, LLC	TIRES	NR (Not Reported)	Tire	2023
01/17/2025	25V023000	Mercedes-Benz USA, LLC	TIRES	NR (Not Reported)	Vehicle	165
01/17/2025	25V020000	Ford Motor Company	POWER TRAIN	25S03	Vehicle	259
01/17/2025	25V019000	Ford Motor Company	ELECTRICAL SYSTEM	25S02	Vehicle	272817
01/17/2025	25V025000	Ford Motor Company	SUSPENSION	25S01	Vehicle	149449

7. standardization and normalization of column

Standardization and normalization are crucial when dealing with numerical data that varies in scale, especially for machine learning algorithms.

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
# Standardization: Transform data to have a mean of 0 and a standard deviation of 1
standard_scaler = StandardScaler()
df['Potentially Affected (Standardized)'] = standard_scaler.fit_transform(df[['Potentially Affected']])

# Normalization: Scale data between 0 and 1
min_max_scaler = MinMaxScaler()
df['Potentially Affected (Normalized)'] = min_max_scaler.fit_transform(df[['Potentially Affected']])

# Display the updated DataFrame
print(df[['Potentially Affected', 'Potentially Affected (Standardized)', 'Potentially Affected (Normalized)']].head())
```

```
   Potentially Affected  Potentially Affected (Standardized)  \
0                  18.0                            -0.122429
1                 130.0                            -0.122129
2                 396.0                            -0.121415
3               74469.0                             0.077295
4                 107.0                            -0.122190

   Potentially Affected (Normalized)
0                       5.625000e-07
1                       4.062500e-06
2                       1.237500e-05
3                       2.327156e-03
4                       3.343750e-06
```

Conclusion:

This experiment demonstrated effective data cleaning and preparation techniques. Issues such as missing values, irrelevant data, and outliers were addressed, and the dataset was scaled for uniformity. These steps are essential for ensuring high-quality data and reliable model outcomes.

Neeraj Rijhwani **EXPERIMENT NO: 2** D15C-64

Aim: Data Visualization/ Exploratory data Analysis using Matplotlib and Seaborn.

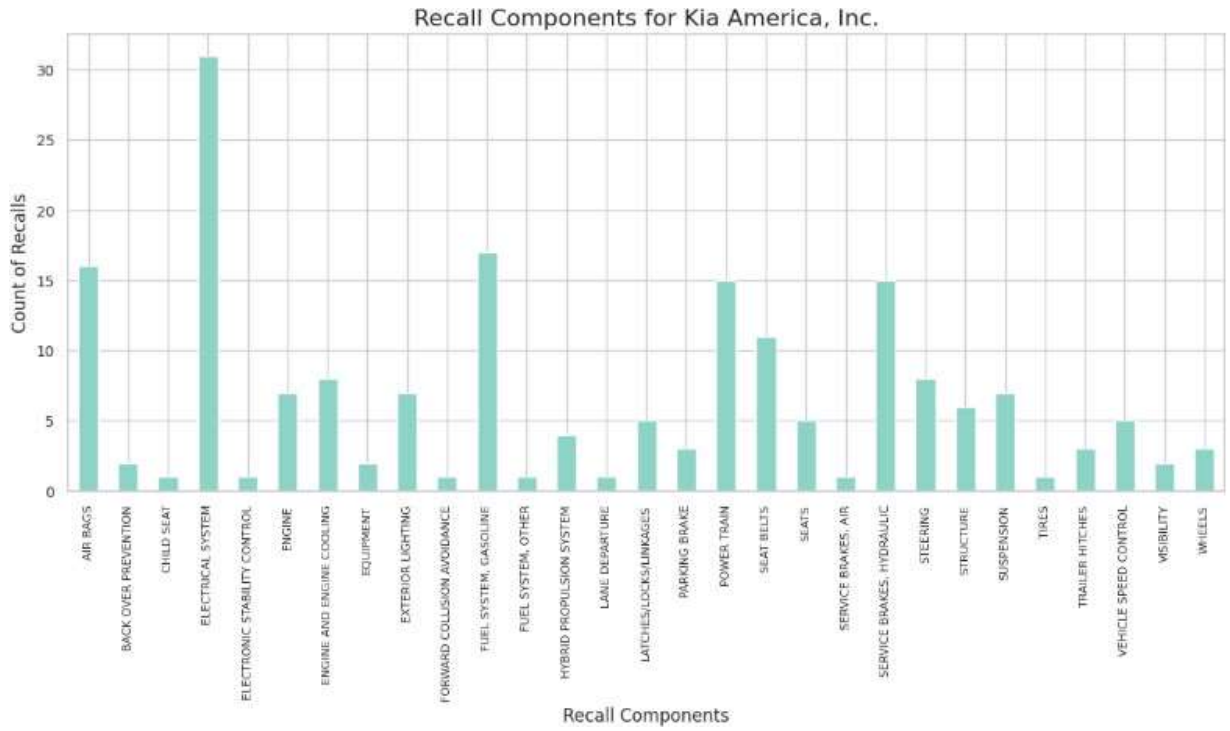
Theory:

1. Bar graph & contingency table using any two features :

1. Bar Graph:

This graph visualizes the count of recalls for different components associated with Kia America, Inc. Each bar represents a specific component and the number of recalls it has. It helps identify which components have the most recalls, aiding in targeting problem areas.

```
import pandas as pd
import matplotlib.pyplot as plt

kia_df = df[df['Manufacturer'] == 'Kia America, Inc.']
crosstb_kia = pd.crosstab(kia_df['Component'], kia_df['Manufacturer'])
plt.figure(figsize=(12, 6))
crosstb_kia.plot.bar(figsize=(12, 6), colormap='Set3', rot=90)
plt.legend().set_visible(False)
plt.title('Recall Components for Kia America, Inc.', fontsize=16)
plt.xlabel('Recall Components', fontsize=12)
plt.ylabel('Count of Recalls', fontsize=12)
plt.xticks(fontsize=8, rotation=90)
plt.yticks(fontsize=10)
plt.tight_layout()
plt.subplots_adjust(bottom=0.2)
plt.show()
```



2. Contingency table: Displays the frequency of recalls for components against manufacturers. For Kia America, Inc., it shows the relationship between components and recall counts. Thus it helps in understanding how components vary by recall count across manufacturers.

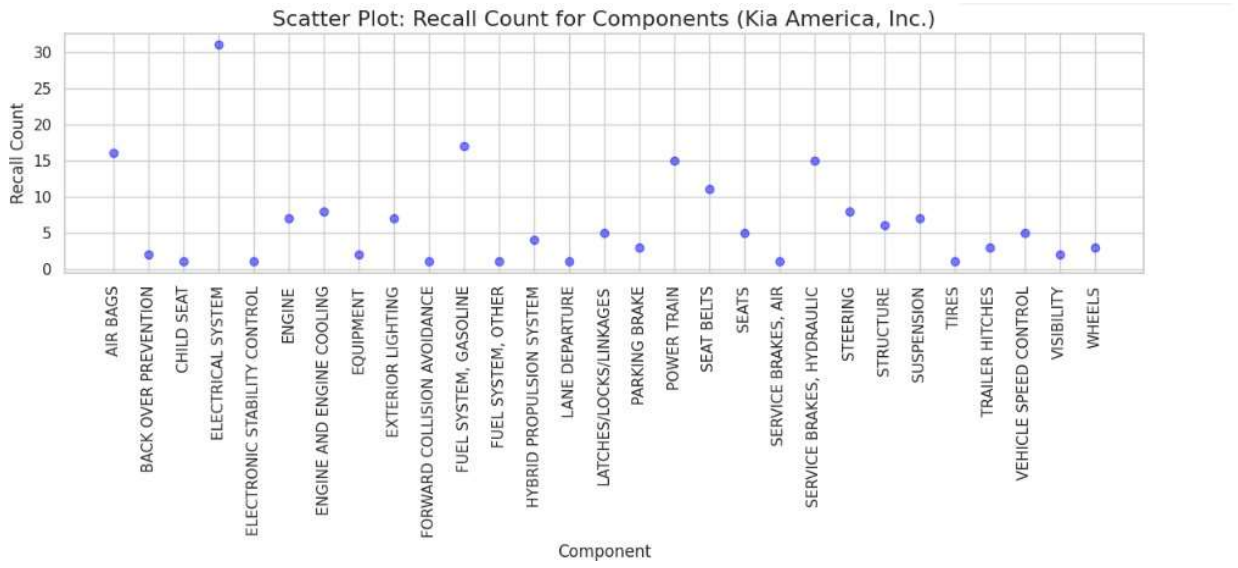
```
import pandas as pd
import matplotlib.pyplot as plt

# Assuming 'df' is your DataFrame

# Create a contingency table (crosstab) for Component vs Manufacturer
crosstb = pd.crosstab(df['Component'], df['Manufacturer'])

# Display the contingency table
print("Contingency Table:")
print(crosstb)
```

```
Contingency Table:
Manufacturer                                  1888653 Ontario Inc  \
Component
AIR BAGS                                                         0
BACK OVER PREVENTION                                             0
CHILD SEAT                                                       0
COMMUNICATION                                                    0
ELECTRICAL SYSTEM                                                0
ELECTRONIC STABILITY CONTROL                                     0
ELECTRONIC STABILITY CONTROL (ESC)                               0
ENGINE                                                           0
ENGINE AND ENGINE COOLING                                        0
EQUIPMENT                                                        1
EQUIPMENT ADAPTIVE/MOBILITY                                      0
EXTERIOR LIGHTING                                                0
FORWARD COLLISION AVOIDANCE                                      0
FUEL SYSTEM, DIESEL                                              0
FUEL SYSTEM, GASOLINE                                            0
FUEL SYSTEM, OTHER                                               0
HYBRID PROPULSION SYSTEM                                         0
INTERIOR LIGHTING                                                0
LANE DEPARTURE                                                   0
```

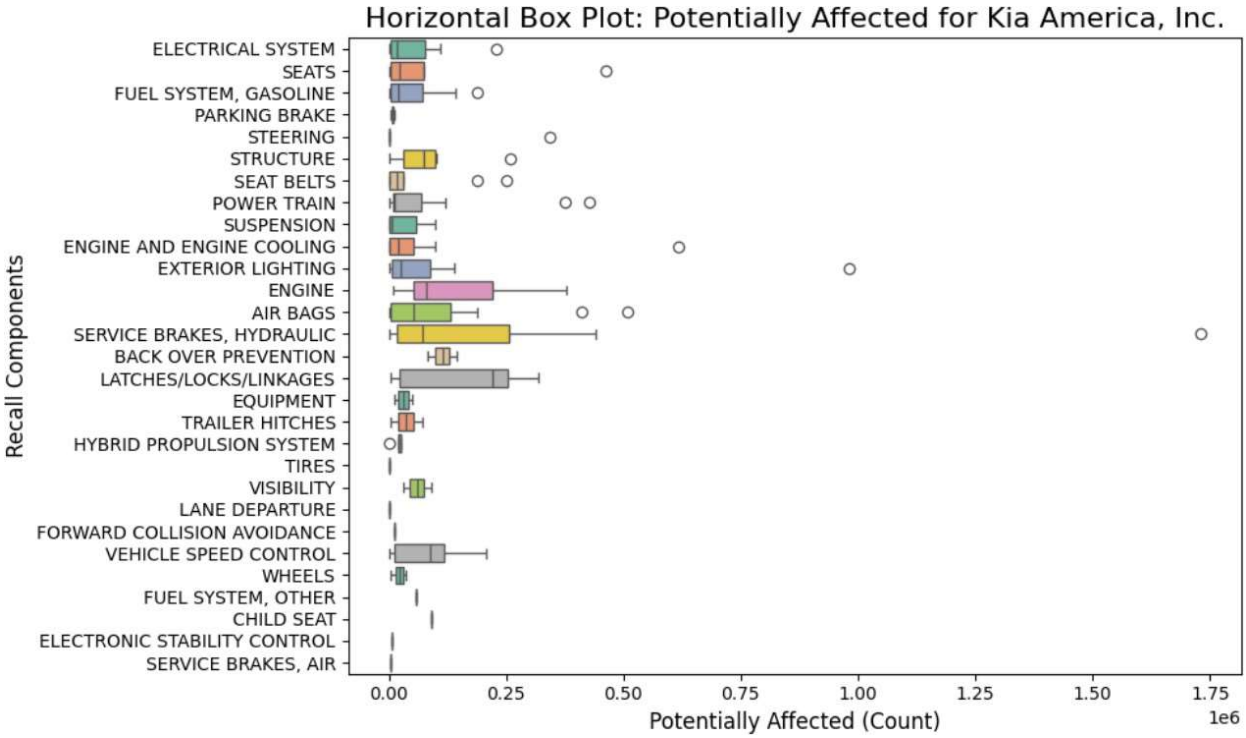
2. Plot Scatter plot, box plot, Heatmap using seaborn.

1. Scatter plot : This plot shows the recall count for various components of Kia America, Inc. Each point represents a component and its associated recall count.
Thus it highlights the distribution and patterns of recall counts, identifying outliers or components with extreme values.

```
import pandas as pd
import matplotlib.pyplot as plt
kia_df = df[df['Manufacturer'] == 'Kia America, Inc.']
crosstb_kia = pd.crosstab(kia_df['Component'], kia_df['Manufacturer'])
scatter_data = crosstb_kia.reset_index()
scatter_data = scatter_data.melt(id_vars=['Component'], value_vars=crosstb_kia.columns, var_name='Manufacturer', value_name='Recall Count')
plt.figure(figsize=(12, 6))
plt.scatter(scatter_data['Component'], scatter_data['Recall Count'], c='blue', alpha=0.5)
plt.title('Scatter Plot: Recall Count for Components (Kia America, Inc.)', fontsize=16)
plt.xlabel('Component', fontsize=12)
plt.ylabel('Recall Count', fontsize=12)
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



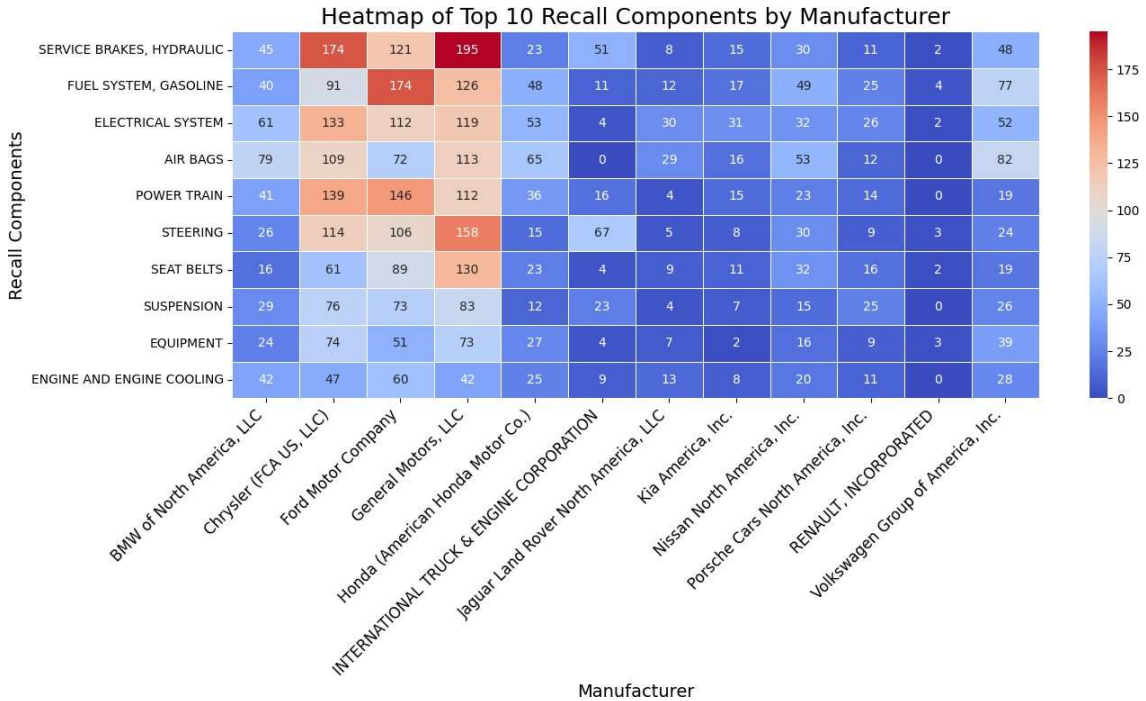
2. Box plot: The box plot shows the distribution of the number of potentially affected vehicles for each recalled component. The horizontal layout makes it easier to compare components.It identifies the spread, central tendency, and outliers in the data, with whiskers representing the data range and dots as potential outliers.




```
#box plot
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

kia_df = df[df['Manufacturer'] == 'Kia America, Inc.']
plt.figure(figsize=(10, 6))
sns.boxplot(x='Potentially Affected', y='Component', data=kia_df, palette='Set2')
plt.title('Horizontal Box Plot: Potentially Affected for Kia America, Inc.', fontsize=16)
plt.xlabel('Potentially Affected (Count)', fontsize=12)
plt.ylabel('Recall Components', fontsize=12)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)

plt.tight_layout()
plt.show()
```

3. Heat map: The heatmap visualizes the contingency table data, where the intensity of color represents the recall count for each component.Thus helps in identifying which components have a higher recall count at a glance, making it easy to spot patterns or

```
# Load data (ensure this is defined)
df = pd.read_csv('Recalls_Data.csv')  # Update with the correct file path

# Filter data for selected manufacturers
selected_manufacturers = [
    'Kia America, Inc.', 'Ford Motor Company', "Nissan North America, Inc.",'Mercedes-Benz
]

filtered_df = df[df['Manufacturer'].isin(selected_manufacturers)]

# Create crosstab for heatmap
heatmap_data = pd.crosstab(filtered_df['Component'], filtered_df['Manufacturer'])

# Fill NaN values with 0
heatmap_data = heatmap_data.fillna(0)

# Sort by total recalls and select only the top 10 components
top_10_components = heatmap_data.sum(axis=1).nlargest(10).index
heatmap_data = heatmap_data.loc[top_10_components]

# Plot heatmap
plt.figure(figsize=(14, 8))
sns.heatmap(heatmap_data, annot=True, cmap='coolwarm', fmt='d', linewidths=0.5)

# Labels and title
plt.title('Heatmap of Top 10 Recall Components by Manufacturer', fontsize=18)
plt.xlabel('Manufacturer', fontsize=14)
plt.ylabel('Recall Components', fontsize=14)
plt.xticks(fontsize=12, rotation=45, ha="right")
plt.yticks(fontsize=10)

plt.tight_layout()
plt.show()
```



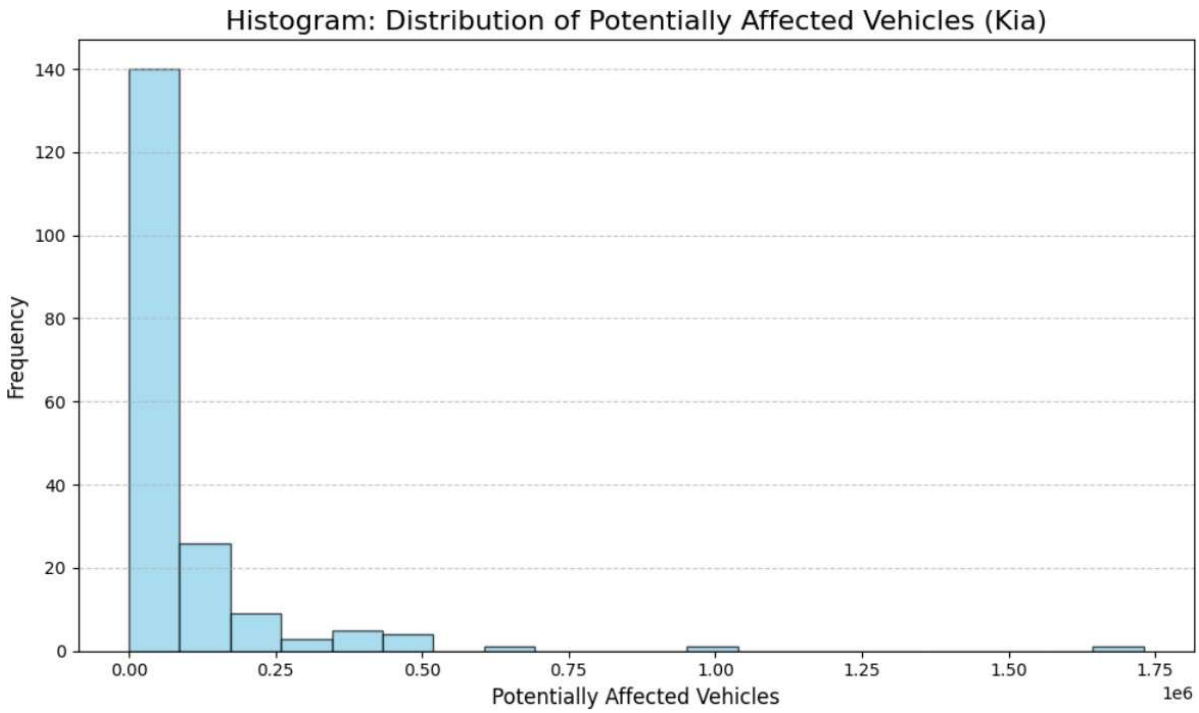
correlations.

3. Create histogram and normalized Histogram.

The histogram shows the frequency distribution of the "Potentially Affected" vehicles.

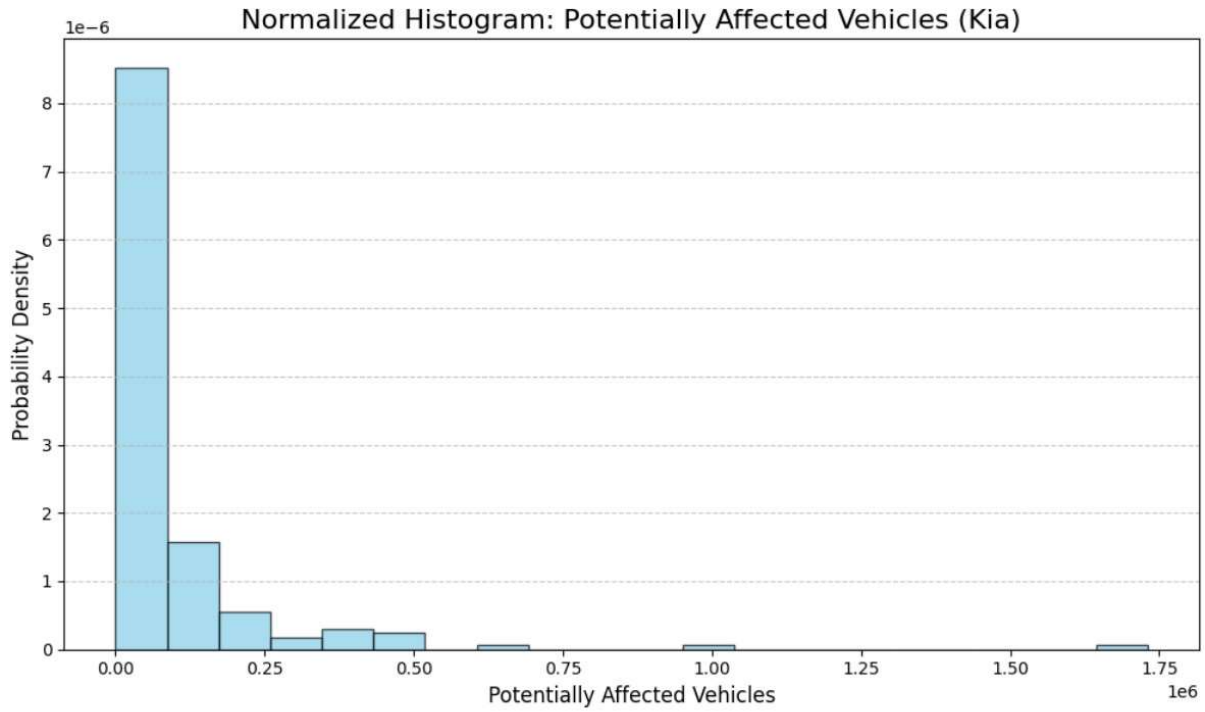
Regular Histogram: It displays counts in bins.

```
import pandas as pd, matplotlib.pyplot as plt
kia_df = df[df['Manufacturer'] == 'Kia America, Inc.']
plt.figure(figsize=(10,6))
plt.hist(kia_df['Potentially Affected'], bins=20, color='skyblue', edgecolor='black', alpha=0.7)
plt.title('Histogram: Distribution of Potentially Affected Vehicles (Kia)', fontsize=16)
plt.xlabel('Potentially Affected Vehicles', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



Normalized Histogram: It displays probability density.

```
import pandas as pd, matplotlib.pyplot as plt
kia_df = df[df['Manufacturer'] == 'Kia America, Inc.']
plt.figure(figsize=(10,6))
plt.hist(kia_df['Potentially Affected'], bins=20, color='skyblue', edgecolor='black', alpha=0.7, density=True)
plt.title('Normalized Histogram: Potentially Affected Vehicles (Kia)', fontsize=16)
plt.xlabel('Potentially Affected Vehicles', fontsize=12)
plt.ylabel('Probability Density', fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



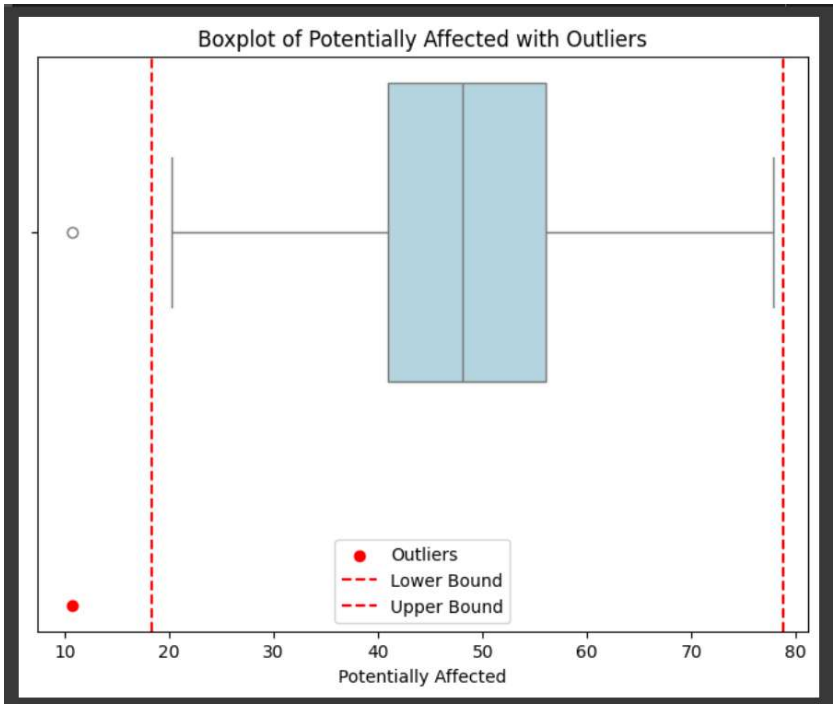
This highlights the data's skewness, central tendency, and spread. The normalized version shows the proportion of each bin relative to the total.

4. Handle outlier using box plot and Inter quartile range.

Outliers are values that deviate significantly from the rest of the data. Using the IQR method, data points beyond Q1−1.5×IQR or Q3+1.5×IQR are considered outliers.

Process: Outliers can be removed or replaced to reduce skewness and improve data accuracy.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
np.random.seed(42)
df = pd.DataFrame({'Potentially Affected': np.random.normal(50, 15, 100)})
col = 'Potentially Affected'
Q1 = df[col].quantile(0.25)
Q3 = df[col].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)]
plt.figure(figsize=(8, 6))
sns.boxplot(x=df[col], color='lightblue')
plt.scatter(outliers[col], [1] * len(outliers), color='red', label='Outliers', zorder=2)
plt.axvline(lower_bound, color='red', linestyle='dashed', label='Lower Bound')
plt.axvline(upper_bound, color='red', linestyle='dashed', label='Upper Bound')
plt.title(f'Boxplot of {col} with Outliers')
plt.legend()
plt.show()
```



Conclusion: Bar Graph and Scatter Plot are suitable for identifying trends and comparisons.Heatmap is excellent for understanding correlations between variables. Box Plot provides insights into distributions and outliers.Normalized Histogram helps understand probabilities and densities.Outlier Handling improves the accuracy of the results.

Aim: Perform Data Modeling on the dataset

Output :

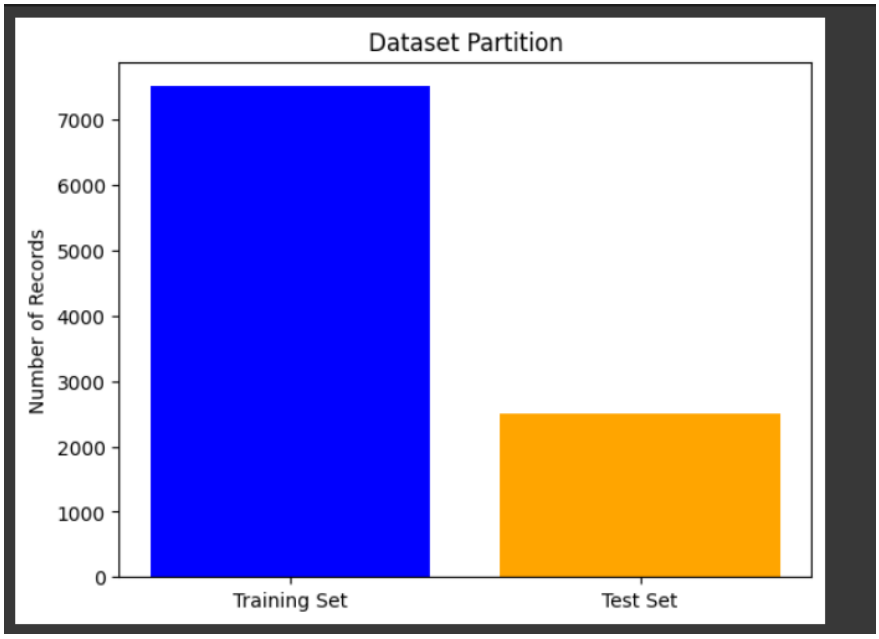
1. Partition of the dataset that is dividing into training and testing of data in **75-25** where 75% of the records are included in the training data and rest 25% are included in the test data.The total records

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from scipy import stats
```

```
proportions = [len(train_data), len(test_data)]
labels = ['Training Set', 'Test Set']

plt.bar(labels, proportions, color=['blue', 'orange'])
plt.title('Dataset Partition')
plt.ylabel('Number of Records')
plt.show()
```

2. Visualization using a bar graph to confirm the proportions of the data split into training and test sets and checking the records that are split up

```
Total records: 10014
Training set records: 7510
Test set records: 2504
```



3. Using a **two-sample Z-test**, we evaluated whether the data split introduced any bias in the popularity distribution

```
[11] train_mean = np.mean(train_data['popularity'])
     test_mean = np.mean(test_data['popularity'])
     train_std = np.std(train_data['popularity'], ddof=1)
     test_std = np.std(test_data['popularity'], ddof=1)

     z_score = (train_mean - test_mean) / np.sqrt((train_std**2/len(train_data)) + (test_std**2/len(test_data)))
     p_value = stats.norm.sf(abs(z_score)) * 2

     print(f"Z-Score: {z_score}")
     print(f"P-Value: {p_value}")
```

4. Now this test is performed for the comparison of **Popularit**y column in training and testing dataset. The Z-statistic was calculated based on the means, standard deviations, and sizes of both samples.

```
Z-Score: -1.59015224967257
P-Value: 0.11180049083548155
```

This validates that the partitioning process preserves the original distribution, ensuring that both sets are representative of the overall data.

Exp 4

Aim: Implementation of Statistical Hypothesis Test using Scipy and Sci-kit learn.

Theory and Output:

1. Loading dataset:

Data loading is the first step in data analysis. The dataset is stored in a CSV file and read using `pandas.read_csv()`.

The first few rows are displayed to understand the dataset structure

```
[1] import pandas as pd
    import scipy.stats as stats
```

```
df = pd.read_csv('/content/Employee.csv')
```

```
df.head()
```

	Education	JoiningYear	City	PaymentTier	Age	Gender	EverBenched	ExperienceInCurrentDomain	LeaveOrNot
0	Bachelors	2017	Bangalore	3	34	Male	No	0	0
1	Bachelors	2013	Pune	1	28	Female	No	3	1
2	Bachelors	2014	New Delhi	3	38	Female	No	2	0
3	Masters	2016	Bangalore	3	27	Male	No	5	1
4	Masters	2017	Pune	3	24	Male	Yes	2	1

Next steps: Generate code with df | View recommended plots | New interactive sheet

2. Pearson's Correlation Coefficient:

Pearson's Correlation Coefficient (denoted as **r**) measures the **linear** relationship between two continuous variables.
Values range from **-1 to +1**:

- **+1**: Perfect positive correlation
- **0**: No correlation
- **-1**: Perfect negative correlation

The formula for Pearson's Correlation Coefficient is:

$$r = \frac{\sum(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum(X_i - \bar{X})^2 \sum(Y_i - \bar{Y})^2}}$$

```
pearson_corr, pearson_p = stats.pearsonr(df['Age'], df['ExperienceInCurrentDomain'])

print(f"Pearson's Correlation Coefficient: {pearson_corr}")
print(f"P-value: {pearson_p}")
```

```
Pearson's Correlation Coefficient: -0.13464285083693067
P-value: 2.8637816441811323e-20
```

3. Spearman's Rank Correlation

- Spearman's Rank Correlation (denoted as ρ, rho) measures the monotonic relationship between two variables.
- It does not require normally distributed data.
- If ranks of two variables are related, it indicates correlation.
- The formula is:

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}$$

```
[13] spearman_corr, spearman_p = stats.spearmanr(df['Age'], df['ExperienceInCurrentDomain'])

     print(f"Spearman's Rank Correlation Coefficient: {spearman_corr}")
     print(f"P-value: {spearman_p}")
```

```
Spearman's Rank Correlation Coefficient: -0.14172932292026683
P-value: 2.6218815420869774e-22
```

4. Kendall's Rank Correlation

Theory:

- Kendall's Tau (τ) measures the **ordinal association** between two variables.
- It counts **concordant** and **discordant** pairs:
  - **Concordant pairs**: If one variable increases, the other also increases.
  - **Discordant pairs**: One increases while the other decreases.
- The formula is:

$$\tau = \frac{(C - D)}{\frac{1}{2}n(n-1)}$$

```
[14] kendall_corr, kendall_p = stats.kendalltau(df['Age'], df['ExperienceInCurrentDomain'])

     print(f"Kendall's Rank Correlation Coefficient: {kendall_corr}")
     print(f"P-value: {kendall_p}")
```

```
Kendall's Rank Correlation Coefficient: -0.05223701755751474
P-value: 2.2249017210277004e-06
```

5. Chi-Squared Test

- The **Chi-Squared Test** is used for **categorical data** to check if two variables are independent.
- It compares **observed** and **expected** frequencies.
- The formula is:

$$\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

```
df['Experience_Category'] = pd.cut(df['ExperienceInCurrentDomain'], bins=[0, 5, 10, 20, 30], labels=['0-5', '6-10', '11-20', '21-30'])
df['Performance_Category'] = pd.cut(df['Age'], bins=[20, 30, 40, 50, 60], labels=['20-30', '30-40', '40-50', '50-60'])

contingency_table = pd.crosstab(df['Experience_Category'], df['Performance_Category'])

chi2_stat, p_val, dof, expected = stats.chi2_contingency(contingency_table)

print(f"Chi-Squared Statistic: {chi2_stat}")
print(f"P-value: {p_val}")
print(f"Degrees of Freedom: {dof}")
print("Expected Frequencies Table:")
print(expected)
```

```
Chi-Squared Statistic: 43.97421499426579
P-value: 2.8256641457475885e-10
Degrees of Freedom: 2
Expected Frequencies Table:
[[3.08375430e+03 1.12254235e+03 7.47033504e+01]
 [1.22456957e+01 4.45765472e+00 2.96649604e-01]]
```

Conclusion

1. **Pearson's Correlation**: Measures **linear relationship** between numerical variables. If **p < 0.05**, the correlation is significant.
2. **Spearman's Correlation**: Checks for **monotonic relationship**. If **p < 0.05**, variables move together in a ranked order.
3. **Kendall's Correlation**: Identifies **ordinal association**. A small **p-value** means a strong relationship.
4. **Chi-Square Test**: Determines **independence of categorical variables**. If **p < 0.05**, variables are dependent; otherwise, they are independent.

Final Summary:

- If **p < 0.05**, the test indicates a significant relationship.
- If **p > 0.05**, no strong relationship exists.

These tests help understand **associations** in the dataset for data-driven decisions.

Experiment : 5

Aim: To implement the regression algorithm using Python.

Regression Algorithm:

Regression algorithms predict the output values based on input features from the data fed in the system. The go-to methodology is the algorithm builds a model on the features of training data and uses the model to predict the value for new data.

Regression has a wide range of real-life applications. It is essential for any machine learning problem that involves continuous numbers – this includes, but is not limited to, a host of examples, including:

- Financial forecasting (like house price estimates, or stock prices)
- Sales and promotions forecasting
- Testing automobiles
- Weather analysis and prediction
- Time series forecasting

Some of the most used regression algorithms are given below

1. **Simple Linear Regression model:**
    a. Simple linear regression is a statistical method that enables users to summarize and study relationships between two continuous (quantitative) variables.
    b. Linear regression is a linear model wherein a model that assumes a linear relationship between the input variables (x) and the single output variable (y).
    c. Here the “y” can be calculated from a linear combination of the input variables (x).
    d. When there is a single input variable (x), the method is called a simple linear regression. When there are multiple input variables, the procedure is referred to as multiple linear regression.

    Application: some of the most popular applications of Linear regression algorithms are in financial portfolio prediction, salary forecasting, real estate predictions and in traffic arriving at ETAs.

2. **Lasso Regression:**
    a. LASSO stands for Least Absolute Selection Shrinkage Operator wherein shrinkage is defined as a constraint on parameters.

b. The goal of lasso regression is to obtain the subset of predictors that minimize prediction error for a quantitative response variable. The algorithm operates by imposing a constraint on the model parameters that causes regression coefficients for some variables to shrink toward a zero.
c. Variables with a regression coefficient equal to zero after the shrinkage process are excluded from the model. Variables with non-zero regression coefficient variables are most strongly associated with the response variable.
d. Explanatory variables can be either quantitative, categorical or both. This lasso regression analysis is basically a shrinkage and variable selection method and it helps analysts to determine which of the predictors are most important.

Application: Lasso regression algorithms have been widely used in financial networks and economics. In finance, its application is seen in forecasting probabilities of default and Lasso-based forecasting models are used in assessing enterprise wide risk framework. Lasso-type regressions are also used to perform stress test platforms to analyze multiple stress scenarios.

3. **Logistic regression:**
   a. One of the most commonly used regression techniques in the industry which are extensively applied across fraud detection, credit card scoring and clinical trials, wherever the response is binary has a major advantage.
   b. One of the major upsides of this popular algorithm is that one can include more than one dependent variable which can be continuous or dichotomous. The other major advantage of this supervised machine learning algorithm is that it provides a quantified value to measure the strength of association according to the rest of variables.
   c. Despite its popularity, researchers have drawn out its limitations, citing a lack of robust technique and also a great model dependency.

Application: Today enterprises deploy Logistic Regression to predict house values in real estate business, customer lifetime value in the insurance sector and are leveraged to produce a continuous outcome such as whether a customer can buy/will buy scenario.

4. **Multivariate Regression algorithm:**
   a. This technique is used when there is more than one predictor variable in a multivariate regression model and the model is called a multivariate multiple regression.
   b. Termed as one of the simplest supervised machine learning algorithms by researchers, this regression algorithm is used to predict the response variable for a set of explanatory variables.

c. This regression technique can be implemented efficiently with the help of matrix operations and in Python, it can be implemented via the "numpy" library which contains definitions and operations for matrix objects.

Application: Industry application of Multivariate Regression algorithm is seen heavily in the retail sector where customers make a choice on a number of variables such as brand, price and product. The multivariate analysis helps decision makers to find the best combination of factors to increase footfalls in the store.

5. **Multiple Regression Algorithm:**
   a. This regression algorithm has several applications across the industry for product pricing, real estate pricing, marketing departments to find out the impact of campaigns.
   b. Unlike linear regression technique, multiple regression, is a broader class of regressions that encompasses linear and nonlinear regressions with multiple explanatory variables.

Application: Some of the business applications of multiple regression algorithms in the industry are in social science research, behavioral analysis and even in the insurance industry to determine claim worthiness.

Python Library Function Used:

Python library used for linear regression is Scikit-learn
sklearn.linear_model.LinearRegression

LinearRegression fits a linear model with coefficients w = (w1, …, wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

Methods for model.LinearRegression are given below

1. **fit(X, y[, sample_weight]):** Fit linear model.
2. **get_params([deep]):** Get parameters for this estimator.
3. **predict(X):** Predict using the linear model.
4. **score(X, y[, sample_weight]):** Return the coefficient of determination of the prediction.
5. **set_params(**params):** Set the parameters of this estimator.

Data Modeling and Analysis

Dataset: Boston Housing Dataset

Preprocessing:

Handling missing values -

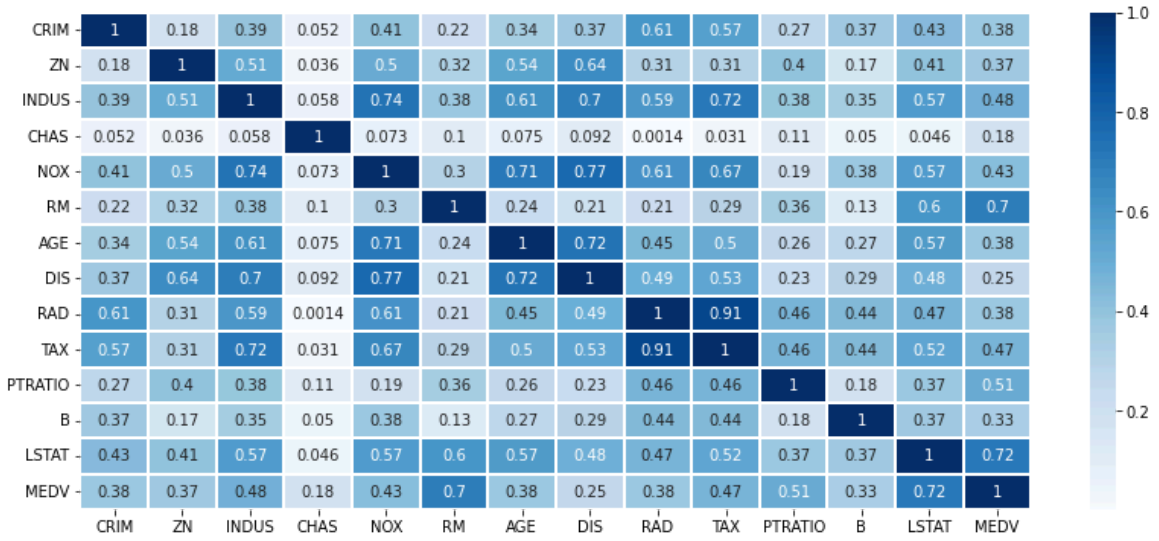
```
df.isnull().sum()
```

```
CRIM       20
ZN         20
INDUS      20
CHAS       20
NOX         0
RM          0
AGE        20
DIS         0
RAD         0
TAX         0
PTRATIO     0
B           0
LSTAT      20
MEDV        0
dtype: int64
```

```
df["CRIM"].fillna(df["CRIM"].mean(),inplace=True)
df["ZN"].fillna(df["ZN"].mean(),inplace=True)
df["INDUS"].fillna(df["INDUS"].mean(),inplace=True)
df["CHAS"].fillna(df["CHAS"].mean(),inplace=True)
df["AGE"].fillna(df["AGE"].mean(),inplace=True)
df["LSTAT"].fillna(df["LSTAT"].mean(),inplace=True)
```

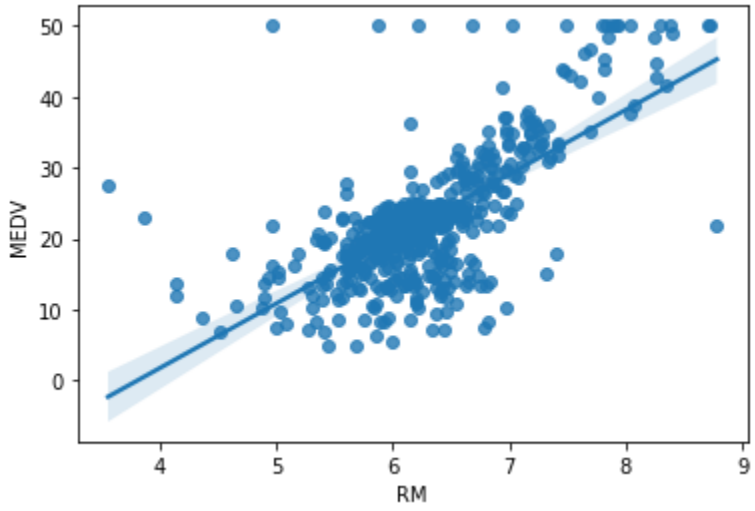
Correlation Analysis:

```
plt.figure(figsize=(14,6))
corr=abs(df.corr())
sns.heatmap(corr,annot=True,linewidth=1,cmap="Blues")
plt.show()
```

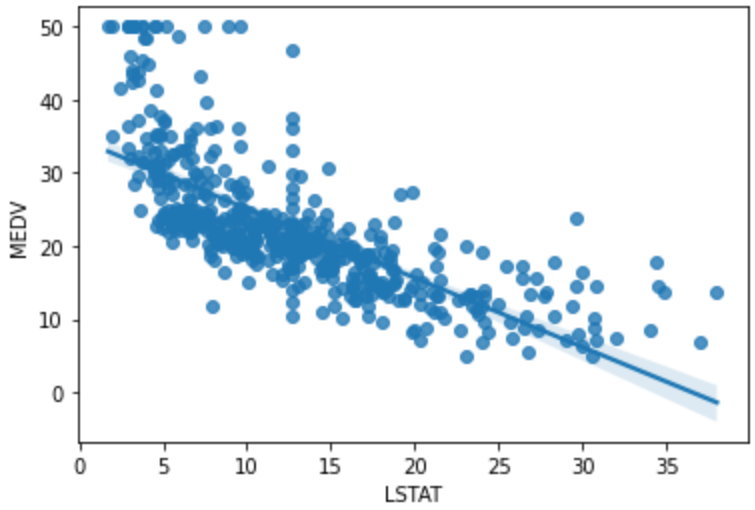


Data Visualization for Regression:

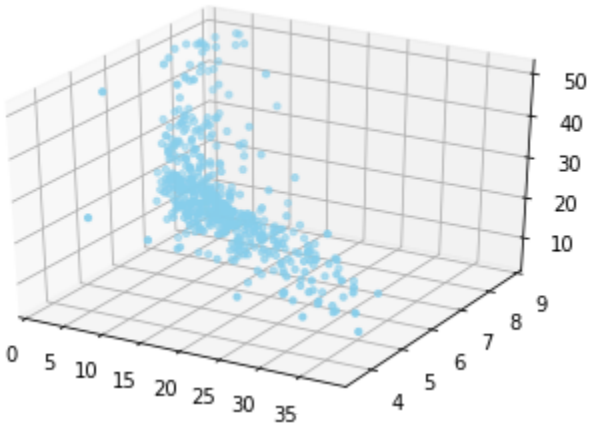
```
import seaborn as sns
import matplotlib.pyplot as plt
sns.regplot(x="RM", y="MEDV", data=df)
```




```
sns.regplot(x="LSTAT", y="MEDV", data=df)
```




```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(df['LSTAT'], df['RM'], df['MEDV'], c='skyblue', s=10)
plt.show()
```



Train Test Split:

```
X = df.drop(columns=['MEDV'])
y = df['MEDV']
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3,random_state=1)
```

Code and Observation:

Using Python Library:

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_train[["RM","LSTAT"]], y_train)
y_predicted = lr.predict(X_test[["RM","LSTAT"]])

from sklearn.metrics import mean_squared_error, r2_score
print("R square value : ",r2_score(y_test, y_predicted))
print("MSE value : ",mean_squared_error(y_test, y_predicted))
```

```
R square value :  0.6804441130852339
MSE value :  29.288730915533865
```

Using User Defined Function:

```
from itertools import zip_longest
from sklearn.metrics import mean_squared_error, r2_score
class CustomLinearRegression():

  def __init__(self, X1, X2, Y, N):
    self.X1 = X1
    self.X2 = X2
    self.Y = Y
    self.N = N
    self.a = 0
    self.b1 = 0
    self.b2 = 0

  def fit(self):
    sum_X1 = np.sum(self.X1)
    sum_X2 = np.sum(self.X2)
    sum_Y = np.sum(self.Y)
    X1_Mean = np.mean(self.X1)
    X2_Mean = np.mean(self.X2)
    Y_Mean = np.mean(self.Y)

    sum_X1_sq = np.sum(self.X1 ** 2)
    sum_X2_sq = np.sum(self.X2 ** 2)

    sum_X1_Y = np.sum(self.X1 * self.Y)
    sum_X2_Y = np.sum(self.X2 * self.Y)
    sum_X1_X2 = np.sum(self.X1 * self.X2)

    x12 = sum_X1_sq - ((sum_X1 ** 2)/self.N)
    x22 = sum_X2_sq - ((sum_X2 ** 2)/self.N)
```

```
    x1y = sum_X1_Y - ((sum_X1 * sum_Y)/self.N)
    x2y = sum_X2_Y - ((sum_X2 * sum_Y)/self.N)
    x1x2 = sum_X1_X2 - ((sum_X1 * sum_X2)/self.N)


    self.b1 =  ((x22 * x1y) - (x1x2*x2y)) / ((x12*x22) - (x1x2 ** 2))
    self.b2 =  ((x12 * x2y) - (x1x2*x1y)) / ((x12*x22) - (x1x2 ** 2))
    self.a = Y_Mean - self.b1* X1_Mean - self.b2* X2_Mean

  def print_equation(self):
    if self.a !=0 and self.b1 != 0 and self.b2 != 0:
      print(f"y = {self.a} + {self.b1}x1 + {self.b2}x2")
    else:
      print("You need to use fit method first.")

  def predict(self, x1_test, x2_test):

    y_pred = [self.a + self.b1*x1 + self.b2*x2 for x1,x2 in
zip_longest(x1_test, x2_test)]
    return y_pred

  def show_metrics(self, y_test, y_pred):
    print("R square value : ",r2_score(y_test, y_pred))
    print("MSE value : ",mean_squared_error(y_test, y_pred))

X1 = X_train["RM"]
X2 = X_train["LSTAT"]
Y = y_train
lin_reg = CustomLinearRegression(X1, X2, Y, N=len(Y))

lin_reg.fit()
lin_reg.print_equation()
```

```
y = 2.920993587665519 + 4.46903585639577x1 + -0.6546089737745551x2
```

```
x1_test = X_test["RM"]
x2_test = X_test["LSTAT"]
final_y_predicted = lin_reg.predict(x1_test, x2_test)
lin_reg.show_metrics(y_test, final_y_predicted)
```

```
R square value :  0.6804441130852366
MSE value :  29.28873091553362
```

Comparing Metrics:

	R Squared	MSE Value
Python Libraries	0.6804441130852339	29.288730915533865
User Defined Function	0.6804441130852366	29.28873091553362

Thus, the Linear Regression using sklearn Libraries gives us almost identical results as compared to our own defined function.

Conclusion:

Thus we have learnt about regression and its various types. We also used the scikit Python library for our own regression model, and compared it with a function of our own.

Experiment : 6

Aim:

To implement the classification algorithm using Python.

Classification Algorithm:

Classification algorithms are used to categorize data into a class or category. It can be performed on both structured or unstructured data. Classification can be of three types: binary classification, multiclass classification, multilabel classification.

Some of the classification algorithms are given below

1. Naive Bayes

Naive Bayes is based on Bayes's theorem which gives an assumption of independence among predictors. This classifier assumes that the presence of a particular feature in a class is not related to the presence of any other feature/variable.
Naive Bayes Classifiers are of three types: Multinomial Naive Bayes, Bernoulli Naive Bayes, Gaussian Naive Bayes.

Pros:

- This algorithm works very fast.
- It can also be used to solve multi-class prediction problems as it's quite useful with them.
- This classifier performs better than other models with less training data if the assumption of independence of features holds.

Cons:

- It assumes that all the features are independent. While it might sound great in theory, but in real life, anyone can hardly find a set of independent features.

Example:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=142)
Naive_Bayes = GaussianNB()
Naive_Bayes.fit(X_train, y_train)
prediction_results = Naive_Bayes.predict(X_test)
```

```
print(prediction_results)
```

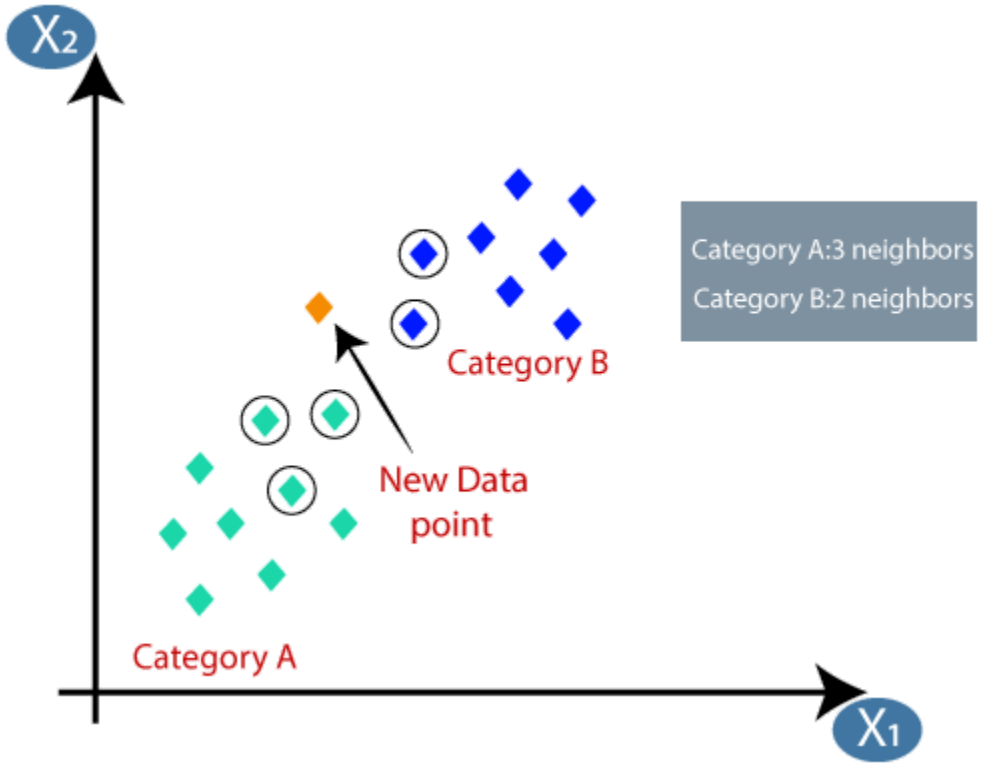
Output:

```
array([0, 1, 1, 2, 1, 1, 0, 0, 2, 1, 1, 1, 2, 0, 1, 0, 2, 1, 1, 2, 2,
1,0, 1, 2, 1, 2, 2, 0, 1, 2, 1, 2, 1, 2, 2, 1, 2])
```

These are the classes predicted for X_test data by our naive Bayes model.

2. K-Nearest Neighbor Algorithm

KNN works on the very same principle. It classifies the new data points depending upon the class of the majority of data points amongst the K neighbor, where K is the number of neighbors to be considered. KNN captures the idea of similarity (sometimes called distance, proximity, or closeness) with some basic mathematical distance formulas like euclidean distance, Manhattan distance, etc.



Choosing the right value for K

To select the K that's right for the data you want to train, run the KNN algorithm several times with different values of K and choose that value of K which reduces the number of errors on unseen data.

Pros:

- KNN is simple and easiest to implement.
- There's no need to build a model, tuning several parameters, or make additional assumptions like some of the other classification algorithms.
- It can be used for classification, regression, and search. So, it is flexible.

Cons:

- The algorithm gets significantly slower as the number of examples and/or predictors/independent variables increase.

Example:

```
from sklearn.neighbors import KNeighborsClassifier
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=142)
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
prediction_results = knn.predict(X_test[:5,:)
print(prediction_results)
```

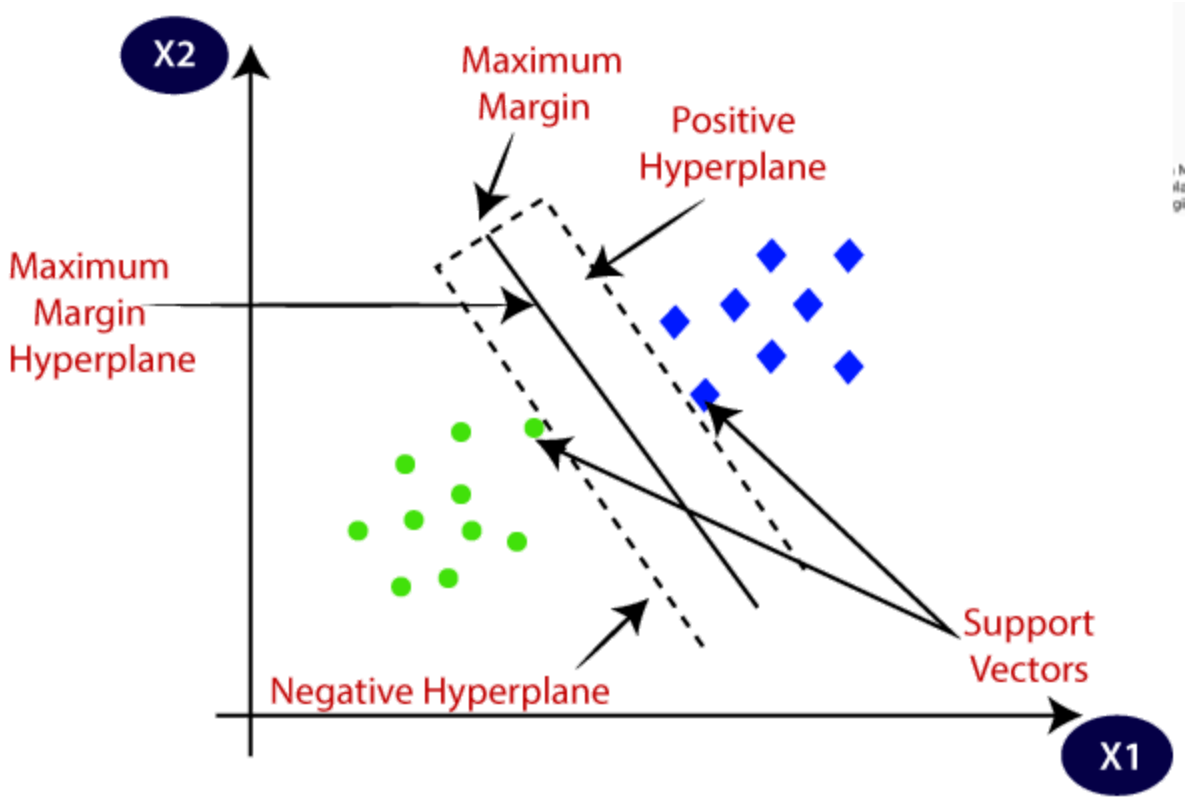
Output:

```
array([0, 1, 1, 2, 1])
```

We predicted our results for 5 sample rows. Hence we have 5 results in the array.

3. SVM

SVM stands for Support Vector Machine. This is a supervised machine learning algorithm that is very often used for both classification and regression challenges. However, it is mostly used in classification problems. The basic concept of the Support Vector Machine and how it works can be best understood by this simple example. So, just imagine you have two tags: green and blue, and our data has two features: x and y. We want a classifier that, given a pair of (x,y) coordinates, outputs if it's either green or blue. Plot labeled training data on a plane and then try to find a plane (hyperplane of dimensions increases) that segregates data points of both colors very clearly.



But this is the case with data that is linear. But what if data is non-linear, then it uses kernel trick. So, to handle this we increase dimension, this brings data in space and now data becomes linearly separable in two groups.

Pros:

- SVM works relatively well when there is a clear margin of separation between classes.
- SVM is more effective in high-dimensional spaces.

Cons:

- SVM is not suitable for large data sets.
- SVM does not perform very well when the data set has more noise i.e. when target classes are overlapping. So, it needs to be handled.

Example:

```
from sklearn import svm
svm_clf = svm.SVC()
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=142)
svm_clf.fit(X_train, y_train)
```

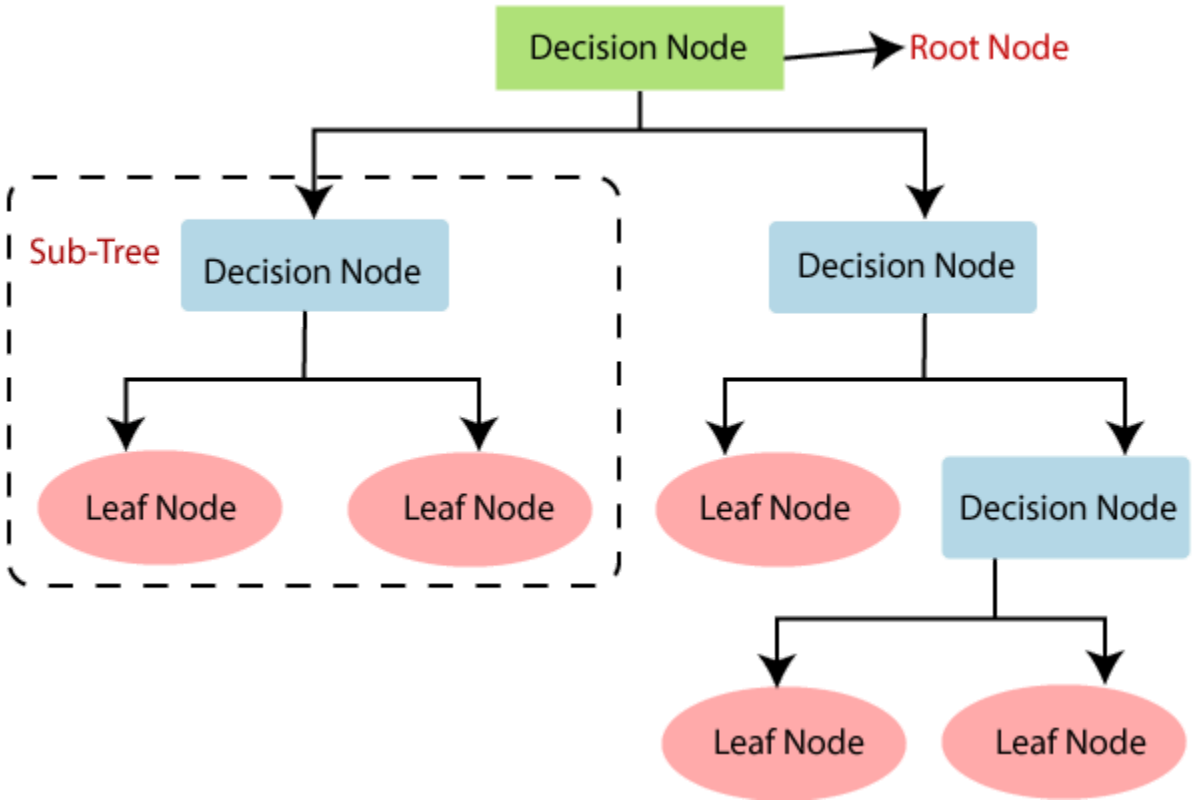
```
prediction_results = svm_clf.predict(X_test[:7,:])
print(prediction_results)
```

Output:

```
array([0, 1, 1, 2, 1, 1, 0])
```

4. Decision Tree

The decision tree is one of the most popular machine learning algorithms used. They are used for both classification and regression problems. Decision trees mimic human-level thinking so it's so simple to understand the data and make some good intuitions and interpretations. They actually make you see the logic for the data to interpret. Decision trees are not like black-box algorithms like SVM, Neural Networks, etc.



For example, if we are classifying a person as fit or unfit then the decision tree looks somewhat like this above in the image.

So, in short, a decision tree is a tree where each node represents a

feature/attribute, each branch represents a decision, a rule, and each leaf represents an outcome. This outcome may be categorical or continuous. Categorical in case of classification and continuous in case of regression applications.

Pros:

- When compared to other algorithms, decision trees require less effort for data preparation while pre-processing.
- They do not require normalization of data and scaling as well.
- Model made on the decision tree is very intuitive and easy to explain to technical teams as well as to stakeholders also.

Cons:

- If even a small change is done in the data, that can lead to a large change in the structure of the decision tree causing instability.
- Sometimes calculation can go far more complex compared to other algorithms.
- Decision trees often take higher time to train the model.

Example:

```
from sklearn import tree
dtc = tree.DecisionTreeClassifier()
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=142)
dtc.fit(X_train, y_train)
prediction_results = dtc.predict(X_test[:7,:])
print(prediction_results)
```

Output:

```
array([0, 1, 1, 2, 1, 1, 0])
```

Python Library Function Used:

Python library used for classification is scikit-learn
sklearn.tree.DecisionTreeClassifier

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation.

Some of the methods of DecisionTreeClassifier are given below

1. **fit(X, y[, sample_weight, check_input, ...]):**
   Build a decision tree classifier from the training set (X, y).

2. **predict(X[, check_input]):**
   Predict class or regression value for X.

3. **score(X, y[, sample_weight]):**
   Return the mean accuracy on the given test data and labels.

4. **set_params(**params):**
   Set the parameters of this estimator.

Data Modeling and Analysis

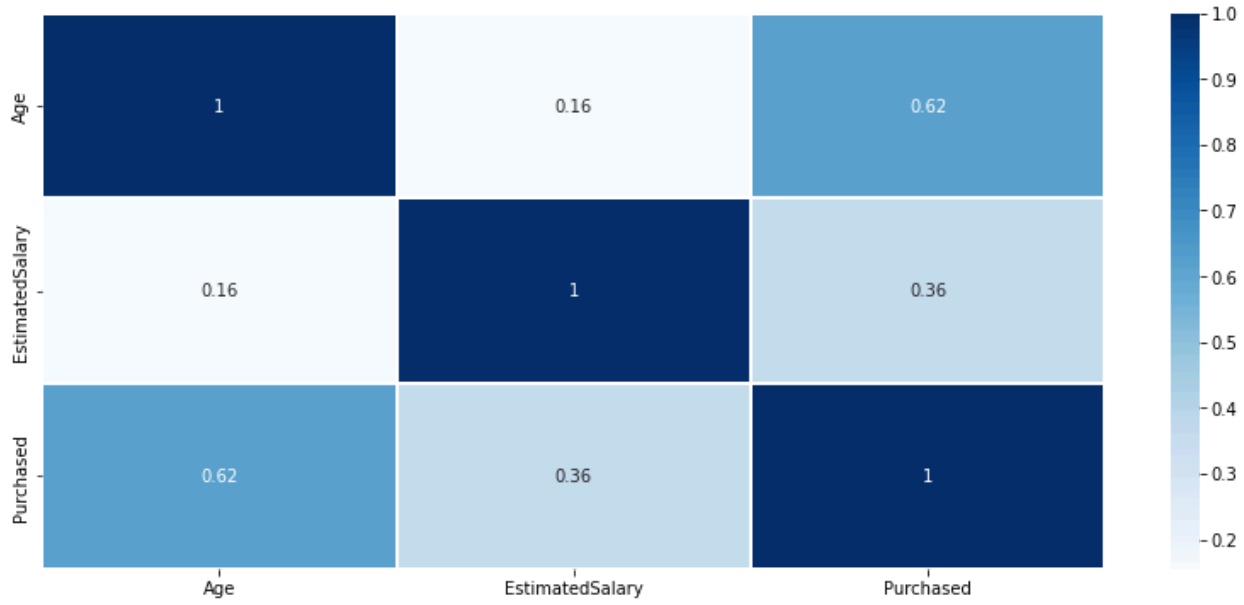
Dataset: Social Network Ads

Preprocessing:
df.isnull().sum()

```
Age              0
EstimatedSalary  0
Purchased        0
dtype: int64
```

Correlation Analysis:

```
plt.figure(figsize=(14,6))
corr=abs(df.corr())
sns.heatmap(corr,annot=True,linewidth=1,cmap="Blues")
plt.show()
```



Train Test Split:

```
X = df.drop(columns=['Purchased'])
y = df['Purchased']
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,random_state=1)
```

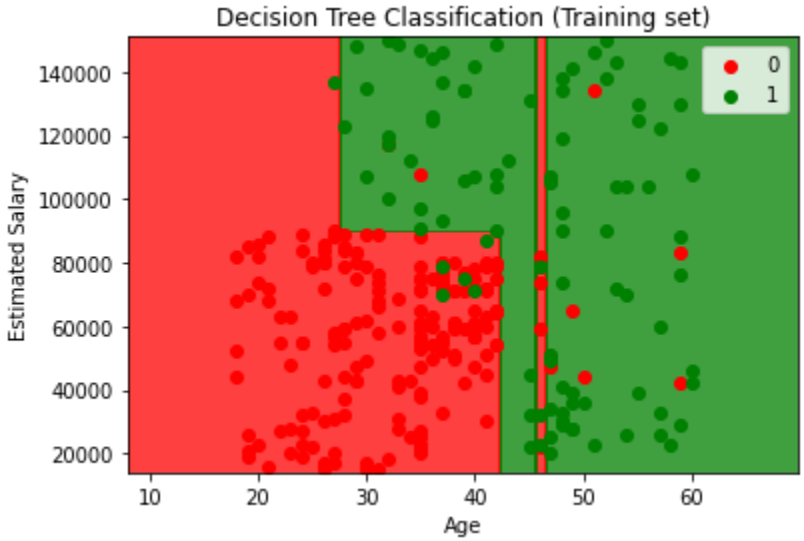
Feature Scaling:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Visualizing Classification in Training dataset

```
from matplotlib.colors import ListedColormap

X_set, y_set = sc.inverse_transform(X_train), y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop =
X_set[:, 0].max() + 10, step = 0.25),
                     np.arange(start = X_set[:, 1].min() - 1000, stop =
X_set[:, 1].max() + 1000, step = 0.25))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(),
X2.ravel()]).T)).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
```

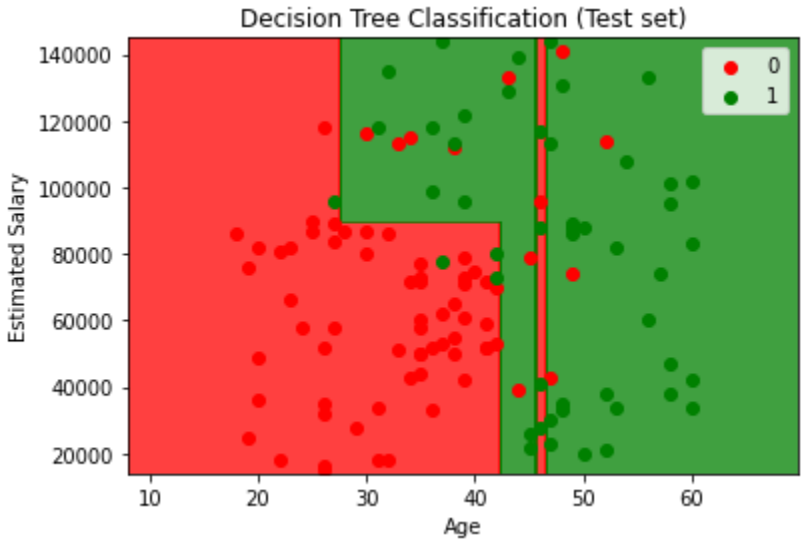
```
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c =
ListedColormap(('red', 'green'))(i), label = j)
plt.title('Decision Tree Classification (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



Visualizing Classification in Testing dataset

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop =
X_set[:, 0].max() + 10, step = 0.25),
                     np.arange(start = X_set[:, 1].min() - 1000, stop =
X_set[:, 1].max() + 1000, step = 0.25))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(),
X2.ravel()]).T)).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c =
ListedColormap(('red', 'green'))(i), label = j)
plt.title('Decision Tree Classification (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
```

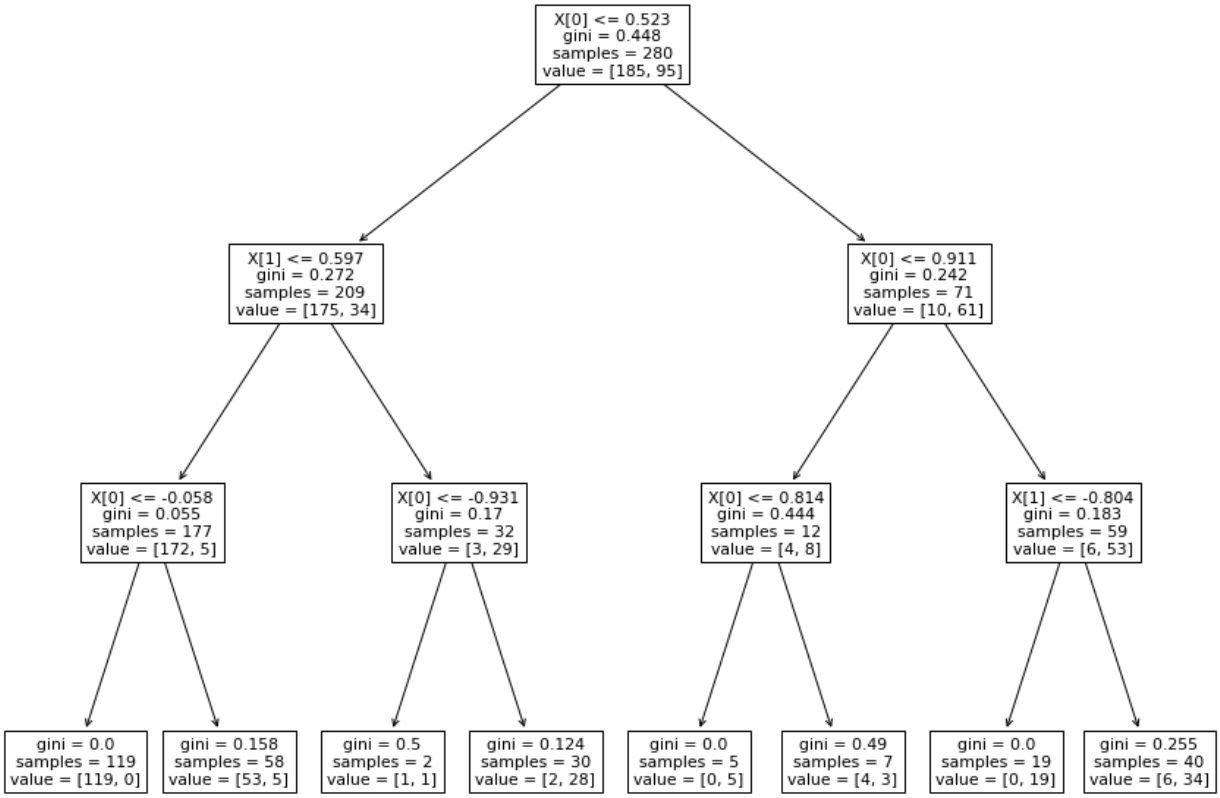
```
plt.show()
```



Code and Observation:

Classification using inbuilt library function

```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'gini', random_state =
0,max_depth=3)
classifier.fit(X_train, y_train)

from sklearn import tree
import matplotlib.pyplot as plt
plt.figure(figsize=(15,12))
tree.plot_tree(classifier)
```




```
y_pred = classifier.predict(X_test)
from sklearn.metrics import confusion_matrix, accuracy_score,
classification_report
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[61 11]
 [ 8 40]]
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.85      0.87        72
           1       0.78      0.83      0.81        48

    accuracy                           0.84       120
   macro avg       0.83      0.84      0.84       120
weighted avg       0.84      0.84      0.84       120
```

```
accuracy_score(y_test, y_pred)
```

```
0.8416666666666667
```

Classification using user-defined function

```
from collections import Counter

class Node:
    """
    Class for creating the nodes for a decision tree
    """
    def __init__(
        self,
        Y: list,
        X: pd.DataFrame,
        min_samples_split=None,
        max_depth=None,
        depth=None,
        node_type=None,
        rule=None
    ):
        # Saving the data to the node
        self.Y = Y
        self.X = X

        # Saving the hyper parameters
        self.min_samples_split = min_samples_split if min_samples_split else
20
        self.max_depth = max_depth if max_depth else 5

        # Default current depth of node
        self.depth = depth if depth else 0

        # Extracting all the features
        self.features = list(self.X.columns)

        # Type of node
        self.node_type = node_type if node_type else 'root'

        # Rule for spliting
        self.rule = rule if rule else ""

        # Calculating the counts of Y in the node
        self.counts = Counter(Y)
```

```
        # Getting the GINI impurity based on the Y distribution
        self.gini_impurity = self.get_GINI()

        # Sorting the counts and saving the final prediction of the node
        counts_sorted = list(sorted(self.counts.items(), key=lambda item:
item[1]))

        # Getting the last item
        yhat = None
        if len(counts_sorted) > 0:
            yhat = counts_sorted[-1][0]

        # Saving to object attribute. This node will predict the class with
the most frequent class
        self.yhat = yhat

        # Saving the number of observations in the node
        self.n = len(Y)

        # Initiating the left and right nodes as empty nodes
        self.left = None
        self.right = None

        # Default values for splits
        self.best_feature = None
        self.best_value = None

    @staticmethod
    def GINI_impurity(y1_count: int, y2_count: int) -> float:
        """
        Given the observations of a binary class calculate the GINI impurity
        """
        # Ensuring the correct types
        if y1_count is None:
            y1_count = 0

        if y2_count is None:
            y2_count = 0

        # Getting the total observations
        n = y1_count + y2_count
```

```
        # If n is 0 then we return the lowest possible gini impurity
        if n == 0:
            return 0.0

        # Getting the probability to see each of the classes
        p1 = y1_count / n
        p2 = y2_count / n

        # Calculating GINI
        gini = 1 - (p1 ** 2 + p2 ** 2)

        # Returning the gini impurity
        return gini

    @staticmethod
    def ma(x: np.array, window: int) -> np.array:
        """
        Calculates the moving average of the given list.
        """
        return np.convolve(x, np.ones(window), 'valid') / window

    def get_GINI(self):
        """
        Function to calculate the GINI impurity of a node
        """
        # Getting the 0 and 1 counts
        y1_count, y2_count = self.counts.get(0, 0), self.counts.get(1, 0)

        # Getting the GINI impurity
        return self.GINI_impurity(y1_count, y2_count)

    def best_split(self) -> tuple:
        """
        Given the X features and Y targets calculates the best split
        for a decision tree
        """
        # Creating a dataset for spliting
        df = self.X.copy()
        df['Y'] = self.Y

        # Getting the GINI impurity for the base input
        GINI_base = self.get_GINI()
```

```
        # Finding which split yields the best GINI gain
        max_gain = 0

        # Default best feature and split
        best_feature = None
        best_value = None

        for feature in self.features:
            # Droping missing values
            Xdf = df.dropna().sort_values(feature)

            # Sorting the values and getting the rolling average
            xmeans = self.ma(Xdf[feature].unique(), 2)

            for value in xmeans:
                # Spliting the dataset
                left_counts = Counter(Xdf[Xdf[feature]<value]['Y'])
                right_counts = Counter(Xdf[Xdf[feature]>=value]['Y'])

                # Getting the Y distribution from the dicts
                y0_left, y1_left, y0_right, y1_right = left_counts.get(0, 0),
left_counts.get(1, 0), right_counts.get(0, 0), right_counts.get(1, 0)

                # Getting the left and right gini impurities
                gini_left = self.GINI_impurity(y0_left, y1_left)
                gini_right = self.GINI_impurity(y0_right, y1_right)

                # Getting the obs count from the left and the right data
splits
                n_left = y0_left + y1_left
                n_right = y0_right + y1_right

                # Calculating the weights for each of the nodes
                w_left = n_left / (n_left + n_right)
                w_right = n_right / (n_left + n_right)

                # Calculating the weighted GINI impurity
                wGINI = w_left * gini_left + w_right * gini_right

                # Calculating the GINI gain
                GINIgain = GINI_base - wGINI

                # Checking if this is the best split so far
```

```
                if GINIgain > max_gain:
                    best_feature = feature
                    best_value = value

                    # Setting the best gain to the current one
                    max_gain = GINIgain

        return (best_feature, best_value)

    def grow_tree(self):
        """
        Recursive method to create the decision tree
        """
        # Making a df from the data
        df = self.X.copy()
        df['Y'] = self.Y

        # If there is GINI to be gained, we split further
        if (self.depth < self.max_depth) and (self.n >=
self.min_samples_split):

            # Getting the best split
            best_feature, best_value = self.best_split()

            if best_feature is not None:
                # Saving the best split to the current node
                self.best_feature = best_feature
                self.best_value = best_value

                # Getting the left and right nodes
                left_df, right_df = df[df[best_feature]<=best_value].copy(),
df[df[best_feature]>best_value].copy()

                # Creating the left and right nodes
                left = Node(
                    left_df['Y'].values.tolist(),
                    left_df[self.features],
                    depth=self.depth + 1,
                    max_depth=self.max_depth,
                    min_samples_split=self.min_samples_split,
                    node_type='left_node',
                    rule=f"{best_feature} <= {round(best_value, 3)}"
                    )
```

```
                self.left = left
                self.left.grow_tree()

                right = Node(
                    right_df['Y'].values.tolist(),
                    right_df[self.features],
                    depth=self.depth + 1,
                    max_depth=self.max_depth,
                    min_samples_split=self.min_samples_split,
                    node_type='right_node',
                    rule=f"{best_feature} > {round(best_value, 3)}"
                    )

                self.right = right
                self.right.grow_tree()

    def print_info(self, width=4):
        """
        Method to print the infromation about the tree
        """
        # Defining the number of spaces
        const = int(self.depth * width ** 1.5)
        spaces = "-" * const

        if self.node_type == 'root':
            print("Root")
        else:
            print(f"|{spaces} Split rule: {self.rule}")
        print(f"{' ' * const}   | GINI impurity of the node:
{round(self.gini_impurity, 2)}")
        print(f"{' ' * const}   | Class distribution in the node:
{dict(self.counts)}")
        print(f"{' ' * const}   | Predicted class: {self.yhat}")

    def print_tree(self):
        """
        Prints the whole tree from the current node to the bottom
        """
        self.print_info()

        if self.left is not None:
            self.left.print_tree()
```

```
        if self.right is not None:
            self.right.print_tree()

    def predict(self, X:pd.DataFrame):
        """
        Batch prediction method
        """
        predictions = []

        for _, x in X.iterrows():
            values = {}
            for feature in self.features:
                values.update({feature: x[feature]})

            predictions.append(self.predict_obs(values))

        return predictions

    def predict_obs(self, values: dict) -> int:
        """
        Method to predict the class given a set of features
        """
        cur_node = self
        while cur_node.depth < cur_node.max_depth:
            # Traversing the nodes all the way to the bottom
            best_feature = cur_node.best_feature
            best_value = cur_node.best_value

            if cur_node.n < cur_node.min_samples_split:
                break

            if (values.get(best_feature) < best_value):
                if self.left is not None:
                    cur_node = cur_node.left
            else:
                if self.right is not None:
                    cur_node = cur_node.right

        return cur_node.yhat

X = df.iloc[:, :-1].values
Y = df.iloc[:, -1].values.reshape(-1,1)
```

```
from sklearn.model_selection import train_test_split
train, test = train_test_split(df, test_size=0.2)

# Constructing the X and Y matrices
X = train[['EstimatedSalary', 'Age']]
Y = train['Purchased'].values.tolist()

# Initiating the Node
root = Node(Y, X, max_depth=3, min_samples_split=100)

# Getting the best split
root.grow_tree()

# Printing the tree information
root.print_tree()
```

```
Root
   | GINI impurity of the node: 0.45
   | Class distribution in the node: {0: 212, 1: 108}
   | Predicted class: 0
|-------- Split rule: Age <= 44.5
           | GINI impurity of the node: 0.27
           | Class distribution in the node: {0: 201, 1: 38}
           | Predicted class: 0
|---------------- Split rule: EstimatedSalary <= 90500.0
                   | GINI impurity of the node: 0.06
                   | Class distribution in the node: {0: 194, 1: 6}
                   | Predicted class: 0
|------------------------ Split rule: Age <= 41.5
                           | GINI impurity of the node: 0.03
                           | Class distribution in the node: {0: 186, 1: 3}
                           | Predicted class: 0
|------------------------ Split rule: Age > 41.5
                           | GINI impurity of the node: 0.4
                           | Class distribution in the node: {0: 8, 1: 3}
                           | Predicted class: 0
|---------------- Split rule: EstimatedSalary > 90500.0
                   | GINI impurity of the node: 0.29
                   | Class distribution in the node: {1: 32, 0: 7}
                   | Predicted class: 1
|-------- Split rule: Age > 44.5
           | GINI impurity of the node: 0.23
           | Class distribution in the node: {1: 70, 0: 11}
           | Predicted class: 1
```

```
# Predicting
pred = root.predict(test)
```

```
actual = list(test['Purchased'])
accuracy = sum(1 for x,y in zip(actual,pred) if x == y) / len(actual)
print("Accuracy:",accuracy)
```

```
Accuracy: 0.8875
```

Comparing Metrics:

	Accuracy
Using Python Libraries	84.167
Using User-Defined Function	88.750

Thus, classification using our defined class function gave better results, as compared to the classification using Python Libraries.

Conclusion:

Thus, we have learnt about classification and learnt how to implement it using python libraries and our own function.

Experiment : 7

Aim:

To implement the Clustering algorithm using Python.

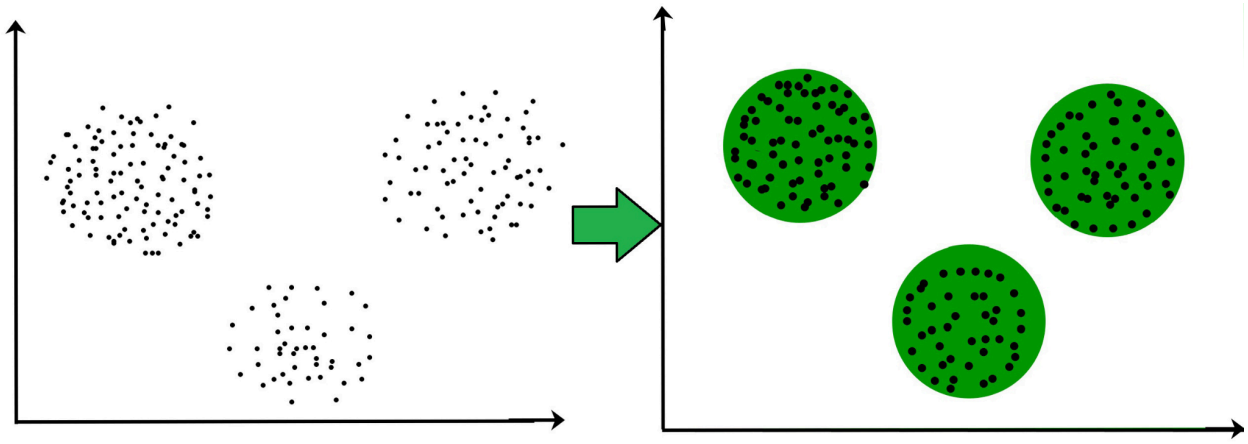
Clustering Algorithm:

Clustering

It is basically a type of unsupervised learning method. An unsupervised learning method is a method in which we draw references from datasets consisting of input data without labelled responses. Generally, it is used as a process to find meaningful structure, explanatory underlying processes, generative features, and groupings inherent in a set of examples.

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group than those in other groups. In simple words, the aim is to segregate groups with similar traits and assign them into clusters.

For ex– The data points in the graph below clustered together can be classified into one single group. We can distinguish the clusters, and we can identify that there are 3 clusters in the below picture.



Why Clustering?

Clustering is very much important as it determines the intrinsic grouping among the unlabelled data present. There are no criteria for good clustering. It depends on the user, what are the criteria they may use which satisfy their need. For instance, we could be interested in finding representatives for homogeneous groups (data reduction), in finding “natural clusters” and describe their unknown properties (“natural” data types), in finding useful and suitable groupings (“useful” data classes) or in finding unusual data objects (outlier detection). This algorithm must

make some assumptions that constitute the similarity of points and each assumption make different and equally valid clusters.

Applications of Clustering in different fields

1. Marketing: It can be used to characterize & discover customer segments for marketing purposes.
2. Biology: It can be used for classification among different species of plants and animals.
3. Libraries: It is used in clustering different books on the basis of topics and information.
4. Insurance: It is used to acknowledge the customers, their policies and identifying the frauds.
5. City Planning: It is used to make groups of houses and to study their values based on their geographical locations and other factors present.
6. Earthquake studies: By learning the earthquake-affected areas we can determine the dangerous zones.

Types of Clustering

Broadly speaking, clustering can be divided into two subgroups :

1. Hard Clustering: In hard clustering, each data point either belongs to a cluster completely or not. For example, in the above example each customer is put into one group out of the 10 groups.
2. Soft Clustering: In soft clustering, instead of putting each data point into a separate cluster, a probability or likelihood of that data point to be in those clusters is assigned. For example, from the above scenario each customer is assigned a probability to be in either of 10 clusters of the retail store.

Clustering Algorithms

When choosing a clustering algorithm, you should consider whether the algorithm scales to your dataset. Datasets in machine learning can have millions of examples, but not all clustering algorithms scale efficiently. Many clustering algorithms work by computing the similarity between all pairs of examples. This means their runtime increases as the square of the number of examples n, denoted as $O(n^2)$ in complexity notation. $O(n^2)$ algorithms are not practical when the number of examples are in millions.

1. Density-Based Methods:
   These methods consider the clusters as the dense region having some similarities and differences from the lower dense region of the space. These methods have good accuracy and the ability to merge two clusters. Example DBSCAN (Density-Based Spatial Clustering of Applications with Noise), OPTICS (Ordering Points to Identify Clustering Structure), etc.

2. Hierarchical Based Methods:
   The clusters formed in this method form a tree-type structure based on the hierarchy. New clusters are formed using the previously formed one. It is divided into two category
   1. Agglomerative (bottom-up approach)
   2. Divisive (top-down approach)

   examples CURE (Clustering Using Representatives), BIRCH (Balanced Iterative Reducing Clustering and using Hierarchies), etc.

3. Partitioning Methods:
   These methods partition the objects into k clusters and each partition forms one cluster. This method is used to optimize an objective criterion similarity function such as when the distance is a major parameter example K-means, CLARANS (Clustering Large Applications based upon Randomized Search), etc.

4. Grid-based Methods:
   In this method, the data space is formulated into a finite number of cells that form a grid-like structure. All the clustering operations done on these grids are fast and independent of the number of data objects, for example STING (Statistical Information Grid), wave cluster, CLIQUE (CLustering In Quest), etc.

K-Means Clustering Algorithm

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. It is an iterative algorithm that divides the unlabeled dataset into K different clusters in such a way that each dataset belongs to only one group that has similar properties. Here K defines the number of predefined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.

It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.

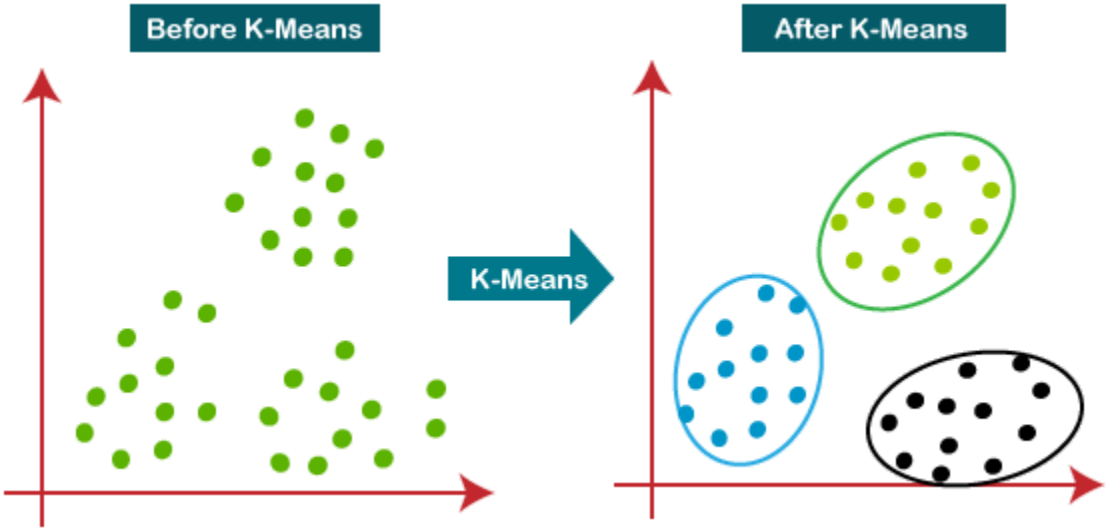
It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters. The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering algorithm mainly performs two tasks:

1. Determines the best value for K center points or centroids by an iterative process.

2. Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

Hence each cluster has data points with some commonalities, and it is away from other clusters.



How does the K-Means Algorithm Work?

The working of the K-Means algorithm is explained in the below steps:

Step-1: Select the number K to decide the number of clusters.
Step-2: Select random K points or centroids. (It can be different from the input dataset).
Step-3: Assign each data point to its closest centroid, which will form the predefined K clusters.
Step-4: Calculate the variance and place a new centroid in each cluster.
Step-5: Repeat the third step, which means assigning each data point to the new closest centroid of each cluster.
Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.
Step-7: The model is ready.

Python Library Function Used:

Python library used for classification is scikit-learn
sklearn.cluster.KMeans

The KMeans algorithm clusters data by trying to separate samples in n groups of equal variance, minimizing a criterion known as the inertia or within-cluster sum-of-squares (see below). This algorithm requires the number of clusters to be specified. It scales well to a large number of samples and has been used across a large range of application areas in many different fields.

The k-means algorithm divides a set of samples into disjoint clusters, each described by the mean

of the samples in the cluster. The means are commonly called the cluster “centroids”; note that they are not, in general, points from, although they live in the same space.

Parameters:

1. **n_clusters: int, default=8**
   The number of clusters to form as well as the number of centroids to generate.

2. **n_init: int, default=10**
   Number of times the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n_init consecutive runs in terms of inertia.

3. **max_iter: int, default=300**
   Maximum number of iterations of the k-means algorithm for a single run.

4. **tol: float, default=1e-4**
   Relative tolerance with regards to Frobenius norm of the difference in the cluster centres of two consecutive iterations to declare convergence.

5. **verbose: int, default=0**
   Verbosity mode.

Attributes:

1. **cluster_centers_: ndarray of shape (n_clusters, n_features)**

   Coordinates of cluster centres. If the algorithm stops before fully converging (see tol and max_iter), these will not be consistent with labels_.

2. **labels_: ndarray of shape (n_samples)**

   Labels of each point

Data Modeling and Analysis

Dataset: College data (Private and Public universities acceptance dataset)
Preprocessing:

```
# There are no null values in the dataset
df.isna().sum()
```

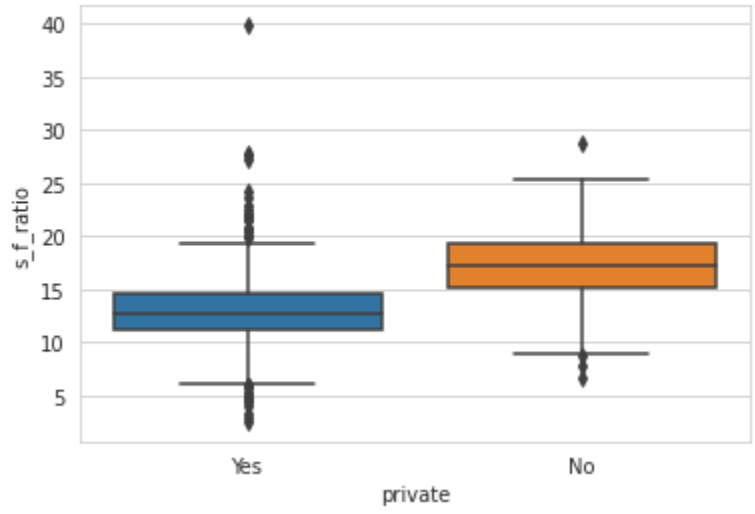
```
private        0
apps           0
accept         0
enroll         0
top10perc      0
top25perc      0
f_undergrad    0
p_undergrad    0
outstate       0
room_board     0
books          0
personal       0
phd            0
terminal       0
s_f_ratio      0
perc_alumni    0
expend         0
grad_rate      0
dtype: int64
```

EDA:

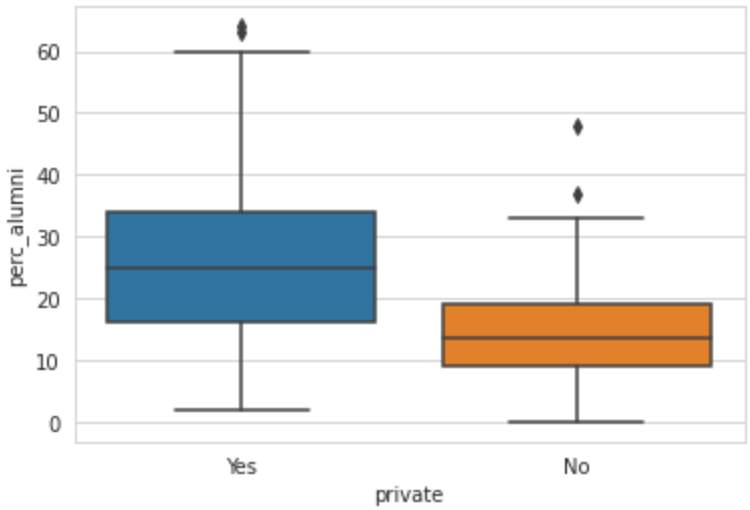
```
# Grad Rate vs Room Board where the points are colored by private
Column
sns.set_style('whitegrid')
sns.lmplot('room_board','grad_rate',data=df, hue='private',
           palette='coolwarm',height=6,aspect=1,fit_reg=True)
```

```
# f_undergrad vs Outstate
sns.set_style('whitegrid')
sns.lmplot('outstate','f_undergrad',data=df, hue='private',
           palette='coolwarm',height=6,aspect=1,fit_reg=False)
```

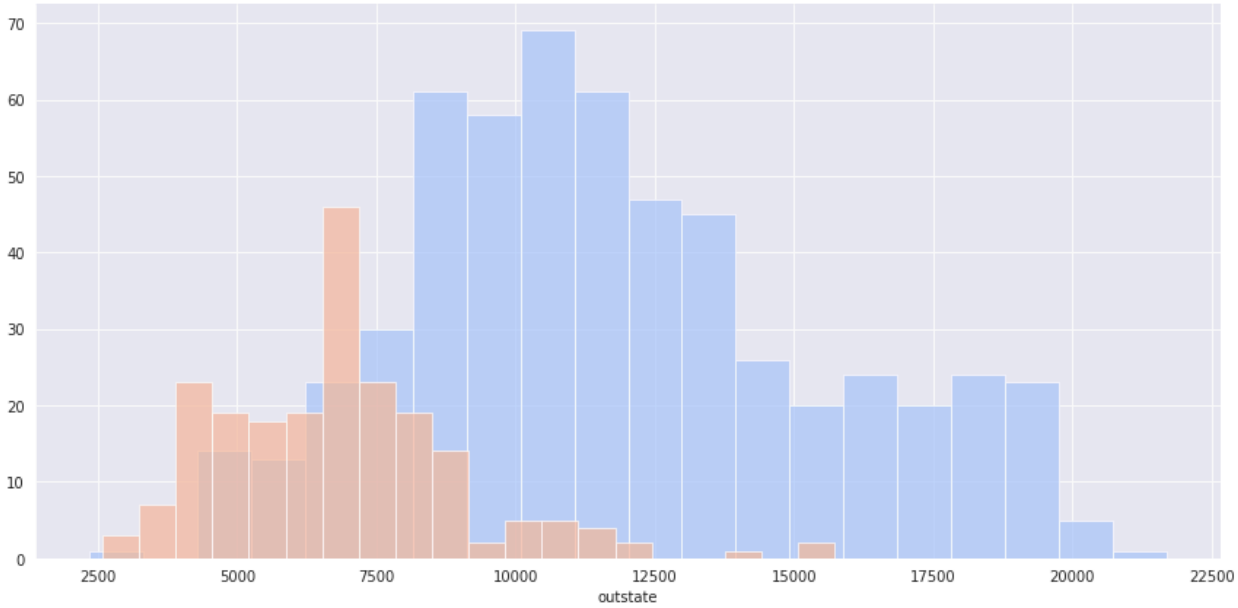

```
# Boxplot of student-faculty ratio based on type of college
sns.boxplot(x='private',y='s_f_ratio',data=df)
```




```
#Boxplot of Percentage of Alumni who donate based on type of college
sns.boxplot(x='private',y='perc_alumni',data=df)
```




```
#Histogram for Out of Station Students' Tuition based on Private
Column
sns.set_style('darkgrid')
g =
sns.FacetGrid(df,hue="private",palette='coolwarm',height=6,aspect=2)
g = g.map(plt.hist,'outstate',bins=20,alpha=0.7)
```



Code and Observation:

Clustering Using Python Libraries:

```
kmeans = KMeans(n_clusters=2,verbose=0,tol=1e-3,max_iter=300,n_init=20)
kmeans.fit(df[['room_board', 'expend']])
```

```
clus_cent=kmeans.cluster_centers_
clus_cent
```

```
array([[ 4187.68175,  8233.97372],
       [ 5622.13043, 20279.1413 ]])
```

```
# Stats for Private Colleges
df[df["private"] == 'Yes'].describe()
```

	apps	accept	enroll	top10perc	top25perc	f_undergrad	p_undergrad	outstate	room_board	books	personal	phd	terminal	s_f_ratio	perc_alumni
count	565.00000	565.00000	565.00000	565.00000	565.00000	565.00000	565.00000	565.00000	565.00000	565.00000	565.00000	565.00000	565.00000	565.00000	565.00000
mean	1977.92920	1305.70265	456.94513	29.33097	56.95752	1872.16814	433.96637	11801.69381	4586.14336	547.50619	1214.44071	71.09381	78.53451	12.94549	25.89027
std	2443.34132	1369.54948	457.52914	17.85139	19.58836	2110.66177	722.37049	3707.47082	1089.69756	174.93230	632.87965	17.35089	15.45025	3.51857	12.40075
min	81.00000	72.00000	35.00000	1.00000	9.00000	139.00000	1.00000	2340.00000	2370.00000	250.00000	250.00000	8.00000	24.00000	2.50000	2.00000
25%	619.00000	501.00000	206.00000	17.00000	42.00000	840.00000	63.00000	9100.00000	3736.00000	450.00000	800.00000	60.00000	68.00000	11.10000	16.00000
50%	1133.00000	859.00000	328.00000	25.00000	55.00000	1274.00000	207.00000	11200.00000	4400.00000	500.00000	1100.00000	73.00000	81.00000	12.70000	25.00000
75%	2186.00000	1580.00000	520.00000	36.00000	70.00000	2018.00000	541.00000	13970.00000	5400.00000	600.00000	1500.00000	85.00000	92.00000	14.50000	34.00000
max	20192.00000	13007.00000	4615.00000	96.00000	100.00000	27378.00000	10221.00000	21700.00000	8124.00000	2340.00000	6800.00000	100.00000	100.00000	39.80000	64.00000

```
# Stats for Non-Private Colleges
df[df["private"] == 'No'].describe()
```

	apps	accept	enroll	top10perc	top25perc	f_undergrad	p_undergrad	outstate	room_board	books	personal	phd	terminal	s_f_ratio	perc_alumni
count	212.00000	212.00000	212.00000	212.00000	212.00000	212.00000	212.00000	212.00000	212.00000	212.00000	212.00000	212.00000	212.00000	212.00000	212.00000
mean	5729.91981	3919.28774	1640.87264	22.83491	52.70283	8571.00472	1978.18868	6813.41038	3748.24057	554.37736	1676.98113	76.83491	82.81604	17.13915	14.35849
std	5370.67533	3477.26628	1261.59201	16.18044	20.09106	6467.69609	2321.03470	2145.24839	858.13993	135.72993	677.51568	12.31753	12.06967	3.41805	7.51893
min	233.00000	233.00000	153.00000	1.00000	12.00000	633.00000	9.00000	2580.00000	1780.00000	96.00000	400.00000	33.00000	33.00000	6.70000	0.00000
25%	2190.75000	1563.25000	701.75000	12.00000	37.00000	3601.00000	600.00000	5366.00000	3121.50000	500.00000	1200.00000	71.00000	76.00000	15.10000	9.00000
50%	4307.00000	2929.50000	1337.50000	19.00000	51.00000	6785.50000	1375.00000	6609.00000	3708.00000	550.00000	1649.00000	78.50000	86.00000	17.25000	13.50000
75%	7722.50000	5264.00000	2243.75000	27.50000	65.00000	12507.00000	2495.25000	7844.00000	4362.00000	612.00000	2051.25000	86.00000	92.00000	19.32500	19.00000
max	48094.00000	26330.00000	6392.00000	95.00000	100.00000	31643.00000	21836.00000	15732.00000	6540.00000	1125.00000	4288.00000	103.00000	100.00000	28.80000	48.00000

```
converter = lambda cluster : 1 if cluster == 'Yes' else 0
df1 = df
df1['Cluster'] = df['private'].apply(converter)

kmeans = KMeans(n_clusters=2,verbose=0,tol=1e-3,max_iter=50,n_init=10)
kmeans.fit(df[['room_board', 'expend']])
clus_cent=kmeans.cluster_centers_
df_desc=pd.DataFrame(df[['room_board', 'expend']].describe())
feat = list(df_desc.columns)
kmclus = pd.DataFrame(clus_cent,columns=feat)
a=np.array(kmclus.diff().iloc[1])
centroid_diff = pd.DataFrame(abs(a),columns=['K-means cluster
centroid-distance'],index=df_desc.columns)
centroid_diff['Mean of corresponding entity
(private)']=np.array(df[['room_board', 'expend']][df['private']==
'Yes'].mean())
```

```
centroid_diff['Mean of corresponding entity
(public)']=np.array(df[['room_board', 'expend']][df['private']==
'No'].mean())
centroid_diff
```

	K-means cluster centroid-distance	Mean of corresponding entity (private)	Mean of corresponding entity (public)
room_board	1346.04023	4586.14336	3748.24057
expend	11244.09413	10486.35398	7458.31604

Clustering Using User-defined function

```
# Considering Two Columns, Expenditure, Room_Board costs
df2 = df[["room_board", "expend"]]
df2['Cluster'] = df["private"].apply(converter)
df2.head()

from math import sqrt
def distance(p1, p2):
    x1 = p1[0]
    y1 = p1[1]
    x2 = p2[0]
    y2 = p2[1]
    return sqrt((x2-x1) ** 2 + (y2-y1) ** 2)
```

	room_board	expend	Cluster
0	3300	7041	1
1	6450	10527	1
2	3750	8735	1
3	5450	19016	1
4	4120	10922	1

```
cluster_choice = lambda d0, d1: 0 if d0 < d1 else 1
def calculate_centroids(point):
  arr1 = point['x']
  arr2 = point['y']
  return (np.mean(arr1), np.mean(arr2))

# Two Clusters, Private and Public
centroids = [(df2['room_board'][0], df2['expend'][0]), (df2['room_board'][1],
df2['expend'][1])]
clusters = {
      0: {
          'x': [],
```

```
          'y': []
      },
      1: {
          'x': [],
          'y': []
      }
    }

for k in range(20):

    labels = []

    for row in df2.iterrows():

        r = dict(row[1])

        point = (r['room_board'], r['expend'])
        if len(centroids) == 0:
          centroids = [(df2['room_board'][0], df2['expend'][0]),
(df2['room_board'][1], df2['expend'][1])]
        d1 = distance(point, centroids[0])
        d2 = distance(point, centroids[1])


        cluster = cluster_choice(d1, d2)
        clusters[cluster]['x'].append(r['room_board'])
        clusters[cluster]['y'].append(r['room_board'])

        labels.append(cluster)

    centroids = [calculate_centroids(clusters[0]),
calculate_centroids(clusters[1])]
    clusters = {
      0: {
          'x': [],
          'y': []
      },
      1: {
          'x': [],
          'y': []
      }
    }
    centroids = []
```

Comparing Metrics of both the implementations:

Using python libraries:

```
from sklearn.metrics import davies_bouldin_score
db_index = davies_bouldin_score(df1.drop(['private'], axis=1),
kmeans.labels_)
db_index
```

```
1.1333615669636636
```

Using custom function:

```
db_index_custom = davies_bouldin_score(df2, kmeans.labels_)
db_index_custom
```

```
1.9457780421226276
```

Thus, the implementation of clustering using python libraries gives better results as the Davies Bouldin Score of the same is lower than that of the custom function.

Conclusion:

Thus, we have learnt about clustering, various ways to implement clustering in a dataset and compared clustering using our function with the python libraries

Name: Neeraj Rijhwani Roll No: 64 Class:D15C

Experiment No 8

Aim: To implement a recommendation system on your dataset using the following machine learning techniques.

Theory:

Types of Recommendation Systems:

Recommendation systems are primarily classified into three types—content-based filtering, collaborative filtering, and hybrid methods. Content-based filtering recommends items by analyzing the features of items a user has previously liked and suggesting similar items. For example, if a user enjoys romantic comedies, the system will recommend movies with similar genres, actors, or directors. In contrast, collaborative filtering uses the preferences of many users to make recommendations. In user-based collaborative filtering, users with similar tastes are identified, and their liked items are suggested to each other. Item-based collaborative filtering recommends items that are frequently liked together by multiple users. Lastly, hybrid recommendation systems combine content-based and collaborative methods to leverage the strengths of both and reduce common issues like the cold start problem, where limited data about users or items affects the quality of recommendations.

Evaluation Measures for Recommendation Systems:

To assess the effectiveness of a recommendation system, various evaluation metrics are used. One of the most common is **Root Mean Square Error (RMSE)**, which measures the difference between the predicted ratings and the actual user ratings; a lower RMSE indicates more accurate predictions. **Precision@K** measures how many of the top-K recommended items are relevant, focusing on the accuracy of the system's most confident recommendations. **Recall@K**, on the other hand, measures how many of all relevant items are included in the top-K results, reflecting how comprehensive the recommendations are. The **F1-score** is a harmonic mean of precision and recall, providing a balanced evaluation when both accuracy and completeness are important. These metrics help in comparing different recommendation models and in fine-tuning them for better performance.

Steps :

1) **Import Required Libraries**

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
```

```
from sklearn.preprocessing import StandardScaler
from surprise import SVD, Dataset, Reader
from surprise.model_selection import train_test_split
from surprise import accuracy
```

This code snippet imports all the necessary libraries required to implement a recommendation system using both clustering and collaborative filtering techniques. It includes pandas and numpy for efficient data handling and numerical operations, while matplotlib.pyplot is used for visualizing the results. The KMeans and StandardScaler modules from sklearn are utilized for performing user/item clustering and standardizing the data, respectively. The surprise library is specifically designed for building recommendation systems, and here it is used to implement the SVD (Singular Value Decomposition) algorithm, along with tools like Dataset, Reader, train_test_split, and accuracy to load the dataset, split it for training and testing, and evaluate the model's performance. Overall, this import block lays the foundation for building and analyzing a recommendation system.

2) Load the Dataset

Code:

```
anime = pd.read_csv('/content/drive/MyDrive/anime.csv')
ratings = pd.read_csv('/content/drive/MyDrive/rating.csv')
```

This code loads two CSV files from your Google Drive into pandas DataFrames:

- anime.csv → Contains information about anime shows (like title, genre, type, rating, etc.).
- rating.csv → Contains user ratings for those anime (user ID, anime ID, and the score given).

These two datasets will be used together to build the recommendation system — one for the content metadata and the other for user interaction data.

3) Data Cleaning

Code:

```
anime.dropna(inplace=True)
anime = anime[anime['genre'] != 'Unknown']
ratings = ratings[ratings['rating'] != -1]
```

This snippet performs essential data cleaning to ensure that only relevant and meaningful data is used for building the recommendation system. First, it removes any

rows from the anime dataset that contain missing values using dropna(), which helps avoid issues during model training. Next, it filters out entries where the anime genre is marked as 'Unknown', as such values do not provide useful information for recommendations. Lastly, it cleans the ratings dataset by removing all instances where the rating is -1, which typically indicates that the user has not rated the anime and hence does not contribute to learning user preferences. These steps collectively help in refining the dataset for better model performance.

4) Merging Anime Metadata with Ratings

Code:

```
merged_df = ratings.merge(anime, on='anime_id')
```

This step performs an inner join on the ratings and anime DataFrames using the anime_id as the key. The result is a new DataFrame named merged_df that contains both user rating information and corresponding anime metadata like titles and genres. Merging these two datasets is essential for building a recommendation system because it allows the model to relate users' ratings with the specific attributes of each anime, enabling more personalized and accurate recommendations.

5) Clustering Anime Based on Popularity and Rating

Code:

```
anime_cluster = anime.copy()
anime_cluster = anime_cluster[anime_cluster['members'] > 0]
anime_cluster['rating'] = pd.to_numeric(anime_cluster['rating'], errors='coerce')
anime_cluster.dropna(subset=['rating'], inplace=True)

features = anime_cluster[['rating', 'members']]
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)

kmeans = KMeans(n_clusters=5, random_state=42, n_init='auto')
anime_cluster['cluster'] = kmeans.fit_predict(scaled_features)
```

In this step, a copy of the original anime dataset is created to prepare it for clustering. First, only anime with more than 0 members (i.e., people who have engaged with or rated it) are kept. The rating column is converted to numeric, with any non-convertible values handled gracefully. Then, missing ratings are dropped to ensure clean data for clustering.

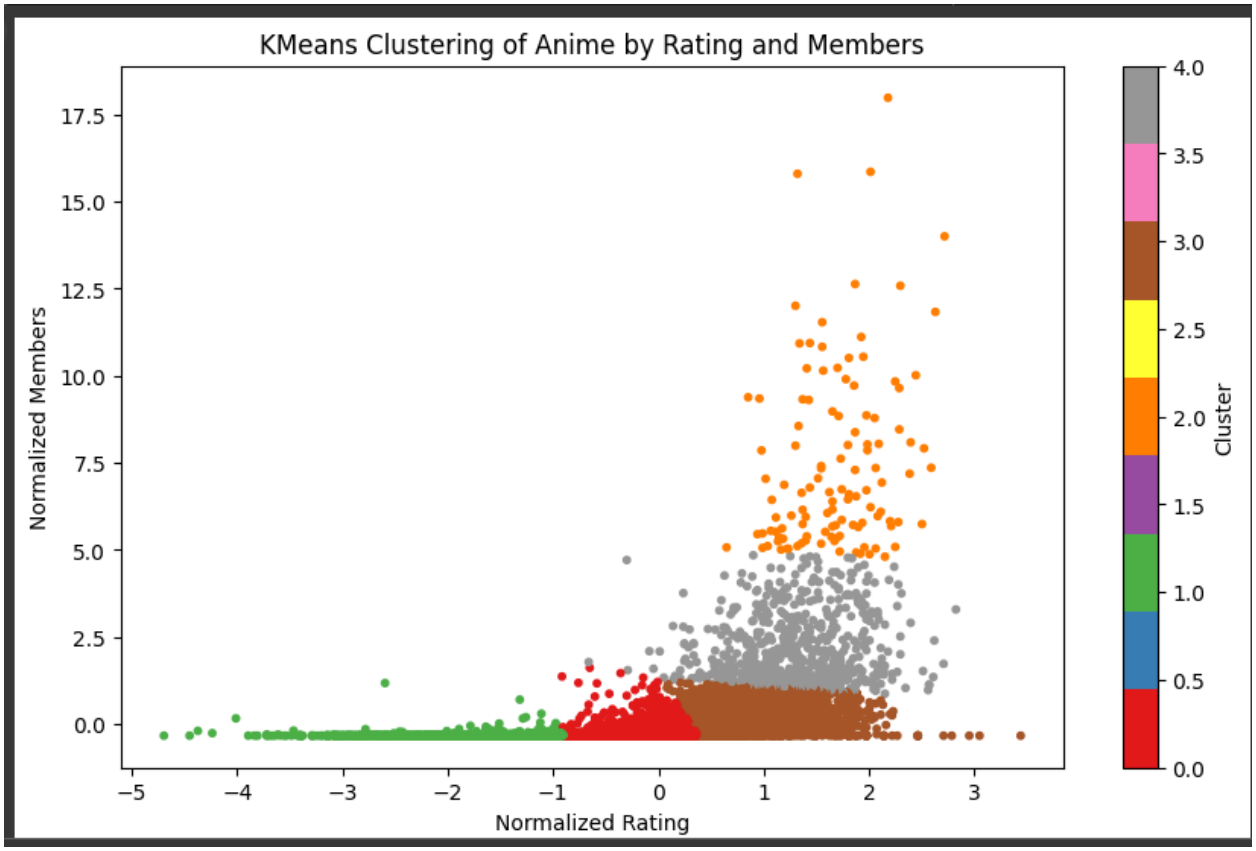
The clustering is based on two features: average user rating and the number of members. These features are standardized using StandardScaler to bring them to the same scale, which is essential for accurate clustering.
Finally, the KMeans algorithm is applied to group the anime into 5 clusters. Each anime is assigned a cluster label based on how similar it is in terms of rating and popularity, allowing further analysis or visualization of similar types of anime.

6) Visualizing Clusters of Anime

Code:

```
plt.figure(figsize=(10,6))
plt.scatter(scaled_features[:, 0], scaled_features[:, 1], c=anime_cluster['cluster'], cmap='Set1', s=10)
plt.title('KMeans Clustering of Anime by Rating and Members')
plt.xlabel('Normalized Rating')
plt.ylabel('Normalized Members')
plt.colorbar(label='Cluster')
plt.show()
```

Output:



This step generates a scatter plot to visually interpret the clusters formed by the KMeans algorithm. The x-axis represents the normalized ratings and the y-axis represents the normalized number of members for each anime. Each point on the graph corresponds to an anime title, and the color of the point represents the cluster it belongs to.
The cmap='Set1' provides distinct colors for different clusters, making it easier to differentiate between them. The colorbar on the side indicates which color corresponds to which cluster. This visualization helps in understanding how anime titles group together based on similarities in popularity (members) and user ratings, providing insight into consumer behavior and content trends.

7) Building and Training the SVD-Based Recommendation Model

Code:

```
reader = Reader(rating_scale=(1, 10))
data = Dataset.load_from_df(ratings[['user_id', 'anime_id', 'rating']], reader)
trainset, testset = train_test_split(data, test_size=0.2, random_state=42)

model = SVD()
model.fit(trainset)
predictions = model.test(testset)
```

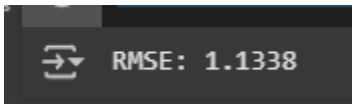
In this step, a recommendation system is built using the Singular Value Decomposition (SVD) algorithm from the Surprise library. The Reader object defines the scale of the ratings (1 to 10), and the Dataset.load_from_df() function converts the cleaned rating data into a format that Surprise can work with. The dataset is then split into a training set and a test set using an 80-20 ratio. The SVD model is trained on the training data to learn latent features of users and anime that explain rating behavior. After training, the model is used to predict ratings on the test set. This step effectively sets up the foundation for personalized anime recommendations based on collaborative filtering.

8) Model Evaluation using RMSE

Code:

```
rmse = accuracy.rmse(predictions)
```

Output:



In this step, we assess the performance of the recommendation model using the Root Mean Squared Error (RMSE). The RMSE value of 1.1338 indicates the average

deviation between the actual ratings and the predicted ratings made by the SVD model. Since the ratings range from 1 to 10, an RMSE around 1.1 is considered acceptable, suggesting that the model is reasonably accurate in its predictions and can be used to provide reliable anime recommendations to users.

9) Function to Generate Top-N Anime Recommendations for a User

Code:

```
def get_top_n_recommendations(user_id, anime_df, ratings_df, model, n=10):
    all_anime_ids = anime_df['anime_id'].unique()
    rated_anime_ids = ratings_df[ratings_df['user_id'] == user_id]['anime_id'].unique()
    unseen_anime_ids = [aid for aid in all_anime_ids if aid not in rated_anime_ids]
    predictions = [model.predict(user_id, aid) for aid in unseen_anime_ids]
    top_predictions = sorted(predictions, key=lambda x: x.est, reverse=True)[:n]
    top_anime_ids = [pred.iid for pred in top_predictions]
        recommendations = anime_df[anime_df['anime_id'].isin(top_anime_ids)][['name',
'genre', 'type', 'rating']]
    return recommendations
```

This function is designed to provide personalized anime recommendations for a specific user based on the trained recommendation model. It first retrieves all available anime IDs and filters out the ones the user has already rated. It then uses the model to predict ratings for the unseen anime and selects the top N (default 10) highest-rated predictions. The function finally returns detailed information (name, genre, type, and rating) about these top recommendations. This allows the system to generate relevant suggestions tailored to the user's interests.

10) Displaying Top 10 Anime Recommendations for a Specific User

Code:

```
user_id_sample =20
print(f"\nTop 10 Anime Recommendations for User ID {user_id_sample}:\n")
print(get_top_n_recommendations(user_id_sample, anime, ratings, model))
```

Name: Neeraj Rijhwani Roll No: 64 Class:D15C

Output:

```
Top 10 Anime Recommendations for User ID 20:

                                                name  \
10                                Clannad: After Story
11                                      Koe no Katachi
118                                   No Game No Life
248                               Kaichou wa Maid-sama!
264                                  Junjou Romantica 2
288                                          Fairy Tail
333         Final Fantasy VII: Advent Children Complete
336  Interstella5555: The 5tory of The 5ecret 5tar ...
368                                   Sekaiichi Hatsukoi
409                                       There She Is!!

                                                genre   type  rating
10   Drama, Fantasy, Romance, Slice of Life, Supern...     TV    9.06
11                             Drama, School, Shounen  Movie    9.05
118  Adventure, Comedy, Ecchi, Fantasy, Game, Super...     TV    8.47
248                  Comedy, Romance, School, Shoujo     TV    8.26
264                Comedy, Drama, Romance, Shounen Ai     TV    8.24
288  Action, Adventure, Comedy, Fantasy, Magic, Sho...     TV    8.22
333                     Action, Fantasy, Super Power    OVA    8.17
336                  Adventure, Drama, Music, Sci-Fi  Music    8.17
368                Comedy, Drama, Romance, Shounen Ai     TV    8.15
409                                   Comedy, Romance    ONA    8.11
```

In this step, the system fetches and prints the top 10 anime recommendations for a user with user_id 20. By calling the get_top_n_recommendations() function and passing in the required data and model, it displays a curated list of anime titles that the user has not yet rated but is likely to enjoy based on the trained SVD model. This output demonstrates how the recommendation system can be personalized for individual users and showcases the system's practical usage in making intelligent suggestions.

Conclusion:

In this experiment, we built a hybrid recommendation system using clustering and collaborative filtering. K-Means helped group similar anime based on features like rating and popularity, while SVD was used to predict user ratings for unseen anime. The model achieved good accuracy and provided personalized Top-N recommendations. This demonstrates the effectiveness of combining content-based and collaborative methods for building smart recommendation systems.

Experiment 09

Aim:

To perform Exploratory data analysis using Apache Spark and Pandas

Introduction to Big Data and Spark:

Big Data

Big Data is a collection of data that is huge in volume, yet growing exponentially with time. It is data with such large size and complexity that none of traditional data management tools can store it or process it efficiently. Big data is also data but with huge size. The definition of big data is data that contains greater variety, arriving in increasing volumes and with more velocity. This is also known as the three Vs.

The three Vs of big data:

1. Volume
   The amount of data matters. With big data, you'll have to process high volumes of low-density, unstructured data. This can be data of unknown value, such as Twitter data feeds, clickstreams on a web page or a mobile app, or sensor-enabled equipment. For some organizations, this might be tens of terabytes of data. For others, it may be hundreds of petabytes.
2. Velocity
   Velocity is the fast rate at which data is received and (perhaps) acted on. Normally, the highest velocity of data streams directly into memory versus being written to disk. Some internet-enabled smart products operate in real time or near real time and will require real-time evaluation and action.
3. Variety
   Variety refers to the many types of data that are available. Traditional data types were structured and fit neatly in a relational database. With the rise of big data, data comes in new unstructured data types. Unstructured and semistructured data types, such as text, audio, and video, require additional preprocessing to derive meaning and support metadata.

Big Data Mining

Big data mining refers to the collective data mining or extraction techniques that are performed on large sets /volume of data or the big data. Big data mining is primarily done to extract and retrieve desired information or patterns from humongous quantities of data.

This is usually performed on a large quantity of unstructured data that is stored over time by an organization. Typically, big data mining works on data searching, refinement , extraction and comparison algorithms. Big data mining also requires support from underlying computing

devices, specifically their processors and memory, for performing operations / queries on large amounts of data.
Big data mining techniques and processes are also used within big data analytics and business intelligence to deliver summarized targeted and relevant information, patterns and/or relationships between data, systems, processes and more.

Big Data Tools

Big Data requires a set of tools and techniques for analysis to gain insights from it. Big Data is an essential part of almost every organization these days and to get significant results through Big Data Analytics a set of tools is needed at each phase of data processing and analysis.
There are a few factors to be considered while opting for the set of tools i.e., the size of the datasets, pricing of the tool, kind of analysis to be done, and many more.
There are a number of big data tools available in the market such as Hadoop which helps in storing and processing large data, Spark helps in-memory calculation, Storm helps in faster processing of unbounded data, Apache Cassandra provides high availability and scalability of a database, MongoDB provides cross-platform capabilities, so there are different functions of every Big Data tool.

Here is the list of top 10 big data tools –

1. Apache Hadoop
2. Apache Spark
3. Flink
4. Apache Storm
5. Apache Cassandra
6. MongoDB
7. Kafka
8. Tableau
9. RapidMiner
10. R Programming

Spark

Apache Spark is an open-source, distributed processing system used for big data workloads. It utilizes in-memory caching and optimized query execution for fast queries against data of any size. Simply put, Spark is a fast and general engine for large-scale data processing.

The fast part means that it’s faster than previous approaches to work with Big Data like classical MapReduce. The secret for being faster is that Spark runs on memory (RAM), and that makes the processing much faster than on disk drives.

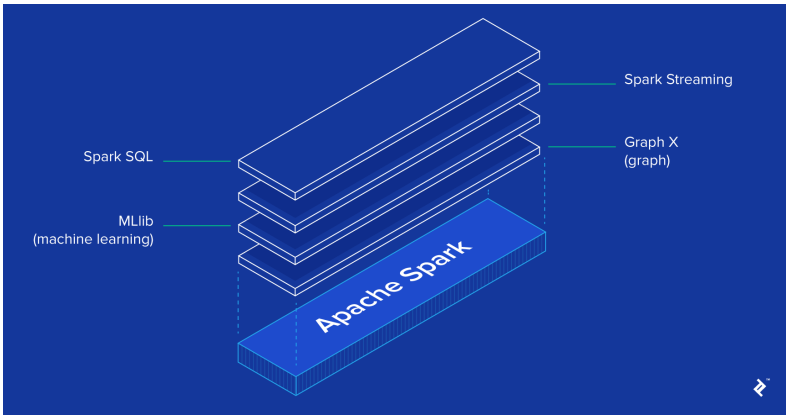
The general part means that it can be used for multiple things like running distributed SQL, creating data pipelines, ingesting data into a database, running Machine Learning algorithms, working with graphs or data streams, and much more.

Features of Spark

1. Fast processing – The most important feature of Apache Spark that has made the big data world choose this technology over others is its speed. Big data is characterized by volume, variety, velocity, and veracity which needs to be processed at a higher speed. Spark contains Resilient Distributed Dataset (RDD) which saves time in reading and writing operations, allowing it to run almost ten to one hundred times faster than Hadoop.
2. Flexibility – Apache Spark supports multiple languages and allows the developers to write applications in Java, Scala, R, or Python.
3. In-memory computing – Spark stores the data in the RAM of servers which allows quick access and in turn accelerates the speed of analytics.
4. Real-time processing – Spark is able to process real-time streaming data. Unlike MapReduce which processes only stored data, Spark is able to process real-time data and is, therefore, able to produce instant outcomes.
5. Better analytics – In contrast to MapReduce that includes Map and Reduce functions, Spark includes much more than that. Apache Spark consists of a rich set of SQL queries, machine learning algorithms, complex analytics, etc. With all these functionalities, analytics can be performed in a better fashion with the help of Spark.

The Spark framework includes:

1. Spark Core as the foundation for the platform
2. Spark SQL for interactive queries
3. Spark Streaming for real-time analytics
4. Spark MLlib for machine learning
5. Spark GraphX for graph processing



Spark Function and Libraries Used:

(Explain the library function used for EDA.)

pyspark.ml module

MLlib is Spark's machine learning (ML) library. Its goal is to make practical machine learning scalable and easy. At a high level, it provides tools such as:

1. ML Algorithms: common learning algorithms such as classification, regression, clustering, and collaborative filtering
2. Featurization: feature extraction, transformation, dimensionality reduction, and selection
3. Pipelines: tools for constructing, evaluating, and tuning ML Pipelines
4. Persistence: saving and load algorithms, models, and Pipelines
5. Utilities: linear algebra, statistics, data handling, etc.

pyspark.sql module

PySpark SQL is a module in Spark which integrates relational processing with Spark's functional programming API. We can extract the data by using an SQL query language. We can use the queries same as the SQL language.

If you have a basic understanding of RDBMS, PySpark SQL will be easy to use, where you can extend the limitation of traditional relational data processing. Spark also supports the Hive Query Language, but there are limitations of the Hive database.

Important classes of Spark SQL and DataFrames:

- pyspark.sql.SparkSession Main entry point for DataFrame and SQL functionality.
- pyspark.sql.DataFrame A distributed collection of data grouped into named columns.
- pyspark.sql.Column A column expression in a DataFrame.
- pyspark.sql.Row A row of data in a DataFrame.
- pyspark.sql.GroupedData Aggregation methods, returned by DataFrame.groupBy().
- pyspark.sql.DataFrameNaFunctions Methods for handling missing data (null values).
- pyspark.sql.DataFrameStatFunctions Methods for statistics functionality.
- pyspark.sql.functions List of built-in functions available for DataFrame.
- pyspark.sql.types List of data types available.
- pyspark.sql.Window For working with window functions.

class pyspark.sql.DataFrame

It is a distributed collection of data grouped into named columns. A DataFrame is similar to the relational table in Spark SQL, can be created using various functions in SQLContext.

```
sqlContext.read.csv("...")
```

After creation of dataframe, we can manipulate it using the several domain-specific-languages (DSL) which are predefined functions of DataFrame. Let’s get started with the functions:

select(): The select function helps us to display a subset of selected columns from the entire dataframe we just need to pass the desired column names. Let’s print any three columns of the dataframe using select().

withColumn(): The withColumn function is used to manipulate a column or to create a new column with the existing column. It is a transformation function, we can also change the datatype of any existing column.

groupBy(): The groupBy function is used to collect the data into groups on DataFrame and allows us to perform aggregate functions on the grouped data. This is a very common data analysis operation similar to the groupBy clause in SQL.

orderBy(): The orderBy function is used to sort the entire dataframe based on the particular column of the dataframe. It sorts the rows of the dataframe according to column values. By default, it sorts in ascending order.

split(): The split() is used to split a string column of the dataframe into multiple columns. This function is applied to the dataframe with the help of withColumn() and select().

lit(): The lit function is used to add a new column to the dataframe that contains literals or some constant value.

when(): The when the function is used to display the output based on the particular condition. It evaluates the condition provided and then returns the values accordingly. It is a SQL function that supports PySpark to check multiple conditions in a sequence and return the value. This function similarly works as if-then-else and switch statements.

filter(): The filter function is used to filter data in rows based on the particular column values.

isNull()/isNotNull(): These two functions are used to find out if there is any null value present in the DataFrame. It is the most essential function for data processing. It is the major tool used for data cleaning.

Conclusion:

Thus, we have learnt what big data is, how Apache Spark is a great big data tool, and also learnt how to use pyspark libraries to preprocess a dataset, and perform EDA on the same in python.

Experiment : 10

Aim:
To perform Batch and Streamed Data Analysis using Apache Spark.

Introduction:
Apache Spark is a powerful open-source data processing engine built for speed, ease of use, and sophisticated analytics. It supports both **batch processing** and **stream processing**, making it an ideal choice for handling large-scale data workloads.

- **Batch processing** deals with processing large volumes of data that are collected over a period. It's used when data doesn't need to be processed in real time.

- **Stream processing** involves real-time data processing, allowing organizations to analyze and react to data as it arrives.

	Batch processing	Stream processing
Data scope	Queries or processing over all or most of the data in the dataset.	Queries or processing over data within a rolling time window, or on just the most recent data record.
Data size	Large batches of data.	Individual records or micro batches consisting of a few records.
Performance	Latencies in minutes to hours.	Requires latency in the order of seconds or milliseconds.
Analysis	Complex analytics.	Simple response functions, aggregates, and rolling metrics.

Batch Processing Using Apache Spark

In batch processing, data is collected, entered, and processed in groups or batches. Apache Spark processes this data by loading it into memory, applying transformations, and then performing actions to generate outputs. It is commonly used for:

- Word counts from large text datasets
- Aggregation operations (e.g., sum, average)
- Log analysis
- ETL (Extract, Transform, Load) processes

Streamed Data Analysis Using Apache Spark

Big data streaming is the process of analyzing data in real time as it flows into the system. Apache Spark's **Structured Streaming** API enables scalable and fault-tolerant stream processing of live data.

Key Benefits of Streaming Data:

- **Real-time insights**: Enables instant decision-making based on current data.
- **Applicability**: Suitable for industries like finance, media, energy, e-commerce, and gaming.
- **Scalability**: Can handle high-throughput and low-latency data streams.
- **Advanced analytics**: Can incorporate machine learning models, time-windowed aggregations, and trend analysis.

Real-world Examples:

- **Financial sector**: Monitoring stock market trends and managing portfolios in real time.
- **Real estate apps**: Delivering property suggestions based on a user's location.
- **Solar energy companies**: Monitoring and maintaining power panels live to avoid performance penalties.
- **Media companies**: Analyzing user clickstreams to optimize content delivery.
- **Online gaming**: Reacting to player actions instantly to enhance user experience.

Challenges:

- Managing data security and privacy
- Handling high-frequency data without system lag

- Designing efficient pipelines for real-time operations

Approach:

Approach to count the words using Spark:

1. Let's create an RDD by using the following command
   data = sc.textFile("file_name.txt")

2. Here, pass any filename that contains the data. Now, we can read the generated result by using the following command.
   data.collect

3. Here, we split the existing data in the form of individual words by using the following command.
   splitdata= book.flatMap(lambda x: x.split()).countByValue()

4. Now, we can read the generated result by using the following command.
   splitdata.collect

5. Now, perform the map operation.
   for i, (word, count) in enumerate(word_counts.items()):
   if i == 100: break
   print(word, count)

   Here, we are assigning a value 1 to each word. Now, we can read the generated result by running the for loop.

6. Now, perform the reduce operation if needed.
   reducedata = mapdata.reduceByKey(lambda a,b : a+b)

   Here, we are summarizing the generated data.

Conclusion:

Thus, we have learnt what batch and stream processing is and also learnt how to implement it using Apache Spark.

Name: Neeraj Pijhwani
Class: DISC
Roll no: 64

04/05

Assignment - 1 (AIDS)

1. What is AI ? Consider the COVID-19 Pandemic situation, how AI helped to survive and renovated our way of life with different applications?

→ Artifical Intelligence (AI) refers to computer systems that can simulate human intelligence through learning, reasoning and Problem-Solving.

AI applications during Covid-19

(i) Disease Prediction and Diagnosis: AI helped Predict outbreaks and diagnose COVID-19 using x-ray and CT scan analysis.

(ii) Chatbots & virtual Assistants: AI-Powered bots provide healthcare guidance and answered COVID-related questions.

(iii) Drug Discovery: AI analyzed potential drugs and vaccines, speeding up the research Process.

(iv) Facial Recognition & Contactless Tech: AI helped in mask-detection and automated temperature screening.

(x) Remote work & education: AI tools like zoom and google meet improved online collaboration

2. What are AI agents terminology, explain with examples

→ An AI agent is an entity that perceives its environment through sensors and acts upon it using actuators.

Types of AI agents:

- Simple Reflex agents: React to current condition e.g thermostat.

- Model-Based Agents: Use internal models to make decisions (e.g self-driving cars).

- Goal-Based Agents: Take actions to achieve a goal. e.g Chess-playing AI

- Utility-Based Agents: optimize performance based on preferences. (e.g recommendation systems)

- Learning Agents - Improve behavior using past experiences (e.g spam filters)

3. How AI technique is used to solve 8 puzzle problem?

→ The 8-Puzzle problem consider of a 3x3 grid with numbered tiles and one empty space. The goal is to arrange the tiles in order.

AI techniques Used:

(i) Breadth-first Search (BFS): Explores all possible moves level-wise.

(ii) Depth-first search (DFS): Explores one path deeply before backtracking.

(iii) A* algorithm: Uses heuristics like Manhattan distance for optimal solution.

(iv) Greedy Search: focuses on moves that seems closes to the solution

4. What is PEAS descriptor? Give PEAS descriptor for following:

Taxi Driver
Medical diagnosis system
A music composer
An aircraft autolander
An essay evaluator
A robotic sentry gun for the Rock lab

System	Performance Marketing	Environment	Actuators	Sensors
Driver	Safety, time, efficiency	Roads, traffic	Steering, brakes	Camera, Sym
Medical Diagnosis	Accuracy, treatment success	Patient health data	Alerts, treatment plans	lab
Music composer	Quality of music	Music database	Sound output	User in melody
Aircraft Autolander	Smooth landing	Weather, runway	Brakes, flaps	GPS, altim
→ Essay Evaluator	Grammar, clarity	Student essays	Feedback	Text a
→ Robotic Sentry Gun	Target accuracy	Security area	firing system	Motion, sensors, camera

5. Categorize a shopping bot for an offline bookstore according to each of the 6 dimensions

Q:

→

Dimensions	Classification
Observability	Partially observable
Determinism	Stochastic
Episodic / Sequential	Sequential
Static / Dynamic	Dynamic
Discrete / Continuous	Discrete
Single / Multi-agent	Multi-agent

6. Differentiate b/w model based and utility based agent.

→

Model Based	Utility Based
(i) Uses an internal model of the world	Chooses actions based on utility function.
(ii) Relies on past states and knowledge	Scales the best action based on utility.
(iii) Maintains an internal model of environment	Evaluates different possible outcomes and ranks them
Eg: Self-driving cars, robotics navigation	Movies recommendation systems, stock market trading AI.

7. Explain the architecture of a knowledge based agent and learning agent.

→ Knowledge Based Agent (KBA)

It is an AI system that stores knowledge, processes it and make decisions based on that knowledge.

Architecture.

- A KBA consists of following components:

(i) Knowledge Base (KB): Stores fact and

(ii) Inference Engine: Uses logical reasoning to derive new facts.

(iii) Perception (Sensors): Gathers information from environment.

(iv) Action Execution (Actuators): Takes action b on reasoning.

Learning Agent:

A learning agent improves its performance ove time by learning from past experiences.

Architecture:

(i) Learning Element: Learns from experience (e.g neural networks)

(ii) Performance Element: Make decisions and takes actions.

(iii) Critic: Evaluates the agent's Performance.

(iv) Problem Generator: Suggests exploratory actions for learning.

8. Ans Same as Q1.

9. Convert the following Predicates:

a. Anita travels by car if available otherwise travels by bus.

→ Predicate: Travels (Anita, Car) ← Available (Car)

Predicate: Travels (Anita, Bus) ← ¬ Available (Car)

b. Bus goes via Andheri and Goregaon.

→ Predicate: Goes (Bus, Andheri) ∧ Goes (Bus, Goregaon)

c. Car has puncture, so it is not available

→ Predicate: ¬ Available (Car) ← Puncture (Car)

Given: Puncture (Car) = True, So Available (Car) = false.

Step-by-step forward Reasoning:

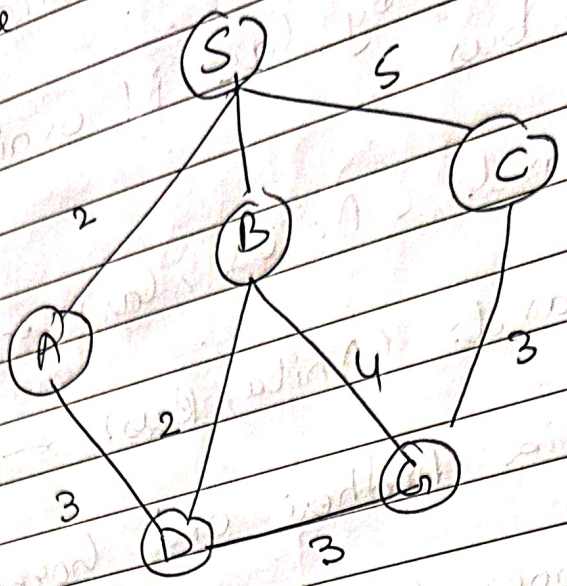
Step1: Since Puncture (Car) = True, we get Available (Car) = false.

Step2: Since Available (Car) = false, Anita must travel by Bus (Travels (Anita, Bus)).

Step3: The Bus goes via Andheri & Goregaon (Goes (Bus, Goregaon))

Step4: Since Anita is on the bus goes via Goregaon, Anita will travel via Goregaon.

10. Find the route



→ BFS algorithm execution.

(i) Starts at S → add S to the queue [S].

(ii) Expand S → Add its neighbors [A, B]

(iii) Expand A → Add its neighbors (excluding S).
[B, C, D]

(iv) Expand B → Add its neighbors (excluding S)
[C, D, E]

(v) Expand C → No new nodes. Queue remains
[D, E]

(vi) Expand D → found G!

Shortest Path by BFS.

S → A → D → G

11. Explain depth limited search ? Explain Iterative Deepening Search with example.

→ Depth-limited Search (DLS)

It is a variation of Depth-first Search (DFS) where a depth limit is set to avoid infinite recursion in case of infinite search spaces

How it works?
- Explores nodes depth-wise like DFS but stops at a specific depth.
- If the goal is not found, it does not continue searching deeper.

Advantages:
- Avoids infinte loops in large graphs.
- Uses less memory than BFS.

Disadvantages:
- If goal node is beyond the depth limit, it won't be found.
- Choosing the right depth limit is tricky.

Ex: If the depth limit is set to 2, DLS will only explore nodes within 2 levels of depth, ignoring deeper paths

Iterative Deepening

It combines BFS and DLS gradually the depth limit set by step until the goal is found.

How it works?
- Performs DLS with depth = 0
- Increases depth limit to 1, then 2, then 3 and so on.
- Stop when goal is found.

Advantages:
- finds the shortest path like BFS
- Uses less memory than BFS

Disadvantages:
- Some nodes get revisited multiple times making IDS slightly slower.

Eg: If the goal is at depth 2, IDS will explore:

Depth 0: Start Node
Depth 1: level 1 nodes
Depth 2: level 2 nodes
Depth 3: level 3 nodes
Depth 4: Goal found

12. Explain how climbing and its drawbacks

in detail with example. Also state limitations of Steepest - ascent hill climbing.

→ Hill Climbing is an AI algorithm that continuously moves towards the best immediate solution to maximize / minimize an objective function.

How it works?

(i) Start with an initial state
(ii) Evaluate all possible next states.
(iii) Move to the best neighbouring state (higher value).
(iv) Repeat until no better moves exist.

Eg: Imagine climbing a mountain in fog where you can only see a few steps ahead. You always move uphill but might get stuck at a local peak.

Drawbacks of Hill Climbing

(i) Local Maxima: The algorithm might stop at a peak that is not the best solution.

(ii) Plateau Problem: The algorithm get stuck in flat areas where no uphill moves exist.

(iii) Ridges: It cannot make lateral moves to find better paths.

Example: In a TSP (Travel
Climbing might find a sub
optimal if the algorithm get stuc
maximum minimum.

Limitations of Steepest-Ascent Hill Climbing

(i) It chooses the best possible move at each step but does not consider future paths.

(ii) Risk: Can miss better solutions if they require a temporary bad move to reach a higher peak.

(iii) Solution: Use Simulated Annealing or Random Restart Hill Climbing

13. Explain simulated annealing and write its algorithm.

→ Simulated Annealing (SA) is an AI optimization technique that allows occasional bad moves to escape local maxima and find a better global solution.

How it works?

(i) Start with an initial solution
(ii) Evaluate neighboring solutions

...ution is better, accept it.

v) If the new solution is worse, accept it with some probability (controlled by a decreasing "temperature" factor)

(v) Gradually reduce temperature, lowering the chances of accepting bad moves.

(vi) Stop when temperature reaches zero or no further improvements can be made.

14. Explain A* algorithm with an example.

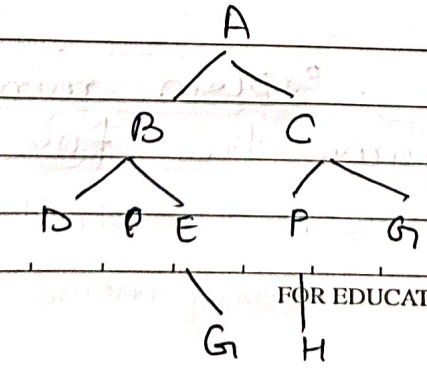
→ A* algorithm is a graph search algorithm that finds the shortest path efficiently using:

g(n): Cost to reach the current node.

h(n): Estimated cost from the current node to the goal (heuristic)

f(n) = g(n) + h(n) (total estimated cost)

Example: find shortest path from A to H



Assume costs:

g(n) : Distance from start

h(n) : Estimated distance to the goal

$f(n) = g(n) + h(n)$

Node	g(n)	h(n)	f(n)
A	0	6	6
B	2	4	6
C	3	3	6
D	4	5	9
E	5	3	8
F	6	2	8
G	7	0	7

Algorithm execution:

(i) Start at A ($f(A) = 6$)

(ii) Expand B & C ($f(B) = 6$, $f(C) = 8$)

(iii) Expand C (lower heuristic)

(iv) Expand F → then G (goal reached)

final Path : A → C → F → G

15. Explain Min Max. Explain minimax algorithm and draw game tree for Tic TAC Toe Game.

→ Minimax Algorithm is used in two-player turn-based games (like Tic Tac Toe, Chess). It helps a player choose the best move by minimizing the opponent's best possible advantage.

Key concepts:

(i) MAX Player: Tries to maximize the score (e.g AI in Tic Tac Toe)

(ii) MIN Player: Tries to minimize the score (e.g Human opponent)

(iii) Game Tree: A tree showing all possible moves and their outcomes.

(iv) Leaf Nodes: Represent the game's final outcome (Win = +1, Lose = -1, Draw = 0).

Tic Tac Toe Game Tree Example.

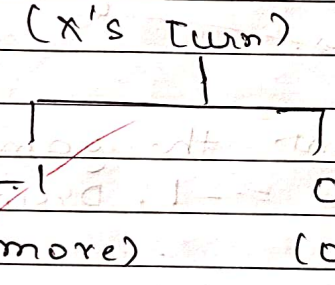
Consider a Tic Tac Toe board where it's MAX's turn (X) and the board looks like this

X	O	X
O	X	O
-	-	-

Possible moves for X:

1. Move in (2,1)
2. Move in (2,2)

Minmax Tree Representation



Here, max will choose 0 because -1 is worse. final move: X moves to (2,2) to get the best possible outcome.

16. Explain Alpha beta pruning algorithms for adversial search with example.

→ It is an optimization technique for

eliminates branches that don't
to be evaluated, reducing computa
time.

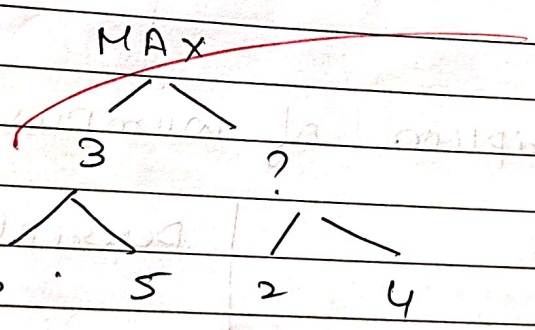
- Why Use Alpha - Bet Pruning?

(i) Speeds up the Minmax Algorithm

(ii) Ignores unnecessary calculations, leading to faster AI decision - making

Alpha - Beta Pruning Example

Consider a game tree where the MAX player wants to maximize the value, and the MIN player wants to minimize!



→ without pruning! The AI checks all nodes

→ With pruning: If a node's value is already worse than a known value, we ignore it.

17. Explain WUMPUS world environment giving its PEAS description. Explain how percept sequence is generated?

→ The Wumpus world is a grid-based environment used to test AI logical reasoning. It consists of:

Agent → AI that explores the world.

Wumpus (Monster) → Kills the agent if encountered.

Pits → Traps that kills the agent.

Gold → The agent must collect gold and exit safely

PEAs description of wumpus would

	Component	Description
(i)	Performance Marking	+1000 for gold, -1000 for death, -1 per move.
(ii)	Environment	4x4 grid with wumpus, pits and gold.
(iii)	Actuators	Move (left, right, up, down) Grab, Shoot.
(iv)	Sensors	Smell (near wumpus), Breeze (near pit), Glitter (near gold)

...owing crypto arithmetic puzzle:

SEND + MORE = MONEY

→

```
  S E N D
+ M O R E
---------
M O N E Y
```

(i) Assign unique digits

S = 9, E = 5, N = 6, D = 7

M = 1, O = 0, R = 8, Y = 2

(ii) Q & Verify the sum:

```
  9 5 6 7
+ 1 0 8 5
---------
1 0 6 5 2
```

19. Consider the following axioms:

→ Given axioms

(i) All people who are graduating are happy

(ii) All happy people are smiling

(iii) Someone is graduating

(i) Step 1: Convert to First Order Predicate Logic (FOL)

Graduating (x) → Happy (x) → (A1) → People who graduate are happy

Happy (x) → Smiling (x) → (A2) → Happy People are smiling

∃x Graduating (x) → (A3) → Someone is graduating

(ii) Step 2: Convert to Clause Form

- Convert implications (→) to clauses (¬P ∨ Q)

Graduating (x) → Happy (x) → ¬ Graduating (x) ∨ Happy (x)

Happy (x) → Smiling (x) → ¬ Happy (x) ∨ Smiling (x)

- Convert Existential Quantifier (∃) to skolem Constant (e.g., let a be a person who is graduating)

∃x Graduating (x) → Graduating (a)

Now the clause form is:

Graduating (x) ∨ Happy (x)

¬ Happy (x) ∨ Smiling (x)

Graduating (a) (From ∃x Graduating (x), we replace x with constant a)

(iii) Step 3: Prove "someone is smiling" Using Resolution

We need to prove Smiling (a) by applying resolution step-by-step.

- Unify Graduating (a) with ¬ Graduating (x) ∨ Happy (x):

Graduating (a)
¬ Graduating (a) ∨ Happy (a)

Happy (a)

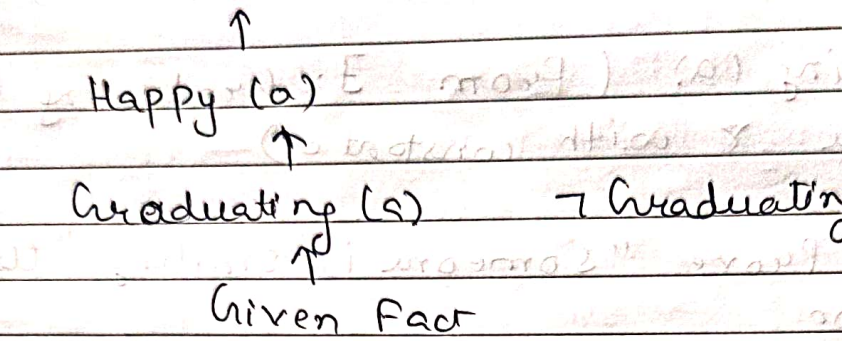
- Unify Happy (a) with ¬ Happy (x) ∨ Smiling (x):

Happy (a)
¬ Happy (a) ∨ Smiling (a)

Smiling (a)

FOR EDUCATIONAL USE

(iii) Resolution Tree



20. Explain Modus Ponen with suitable example.

→ Modus Ponens (MP) is a logical inference rule that states:

(i) If P → Q (i.e, "If P is true, then Q is true").

(ii) P is true

(iii) Therefore, Q must also be true

Example of Modus Ponens

(i) If it rains, the ground is wet → Raining (x)
→ Wet Ground (x)

(ii) It is raining → Raining (Today)

(iii) Therefore, the ground is wet → Wet Ground (Today)

P Q (Premise 1)
(Premise 2)
∴ ——————
R (conclusion)

Real - world Example:

(i) Premise 1: If a student studies hard, they will pass the exam (Study (x) → Pass (x))

(ii) Premise 2: The student studies hard (Study (Neeraj))

(iii) conclusion: The student will pass the exam (Pass (Neeraj))

21. Explain Forward chaining and backward chaining algorithm with the help of example

→ Forward Chaining

- It is a data-driven approach that starts from known facts and derives conclusion step by step.

Steps:

(i) Start with initial facts

(ii) Apply Inference rules to generate new facts

(iii) Continue until the goal is reached.

Example: Use cases

- Expert systems, rule-based AI (e.g., medical diagnosis)

Backward chaining

It is a goal-driven approach that starts from the goal and works backward to determine if fact support it.

Steps:

(i) Start with goal (G)
(ii) Check if G is directly known in knowledge Base (KB)
(iii) If not, check if G can be derived from other rules
(iv) Repeat until known facts are reached.

Example: Use Cases:

- AI planning, medical diagnosis, legal reasoning.

Name : Neeraj Rijhwani Roll No.64 Class: D15C

`

AIDS-I Assignment No: 2

Q.1: Use the following data set for question 1

82, 66, 70, 59, 90, 78, 76, 95, 99, 84, 88, 76, 82, 81, 91, 64, 79, 76, 85, 90

1. Find the Mean (10pts)

Mean:

To find the mean, we sum up all the values in the data set and divide by the total number of values. Let's calculate it for the given data set:

82 + 66 + 70 + 59 + 90 + 78 + 76 + 95 + 99 + 84 + 88 + 76 + 82 + 81 + 91 + 64 + 79 + 76 + 85 + 90 = 1555

There are 20 values in the data set, so the mean is:

Mean = 1555 / 20 = 77.75

Therefore, the mean of the data set is 77.75.

2. Find the Median (10pts)

Median:

To find the median, we arrange the values in ascending order and find the middle value. If there is an even number of values, we take the average of the two middle values.

Arranging the values in ascending order:

59, 64, 66, 70, 76, 76, 76, 78, 79, 81, 82, 82, 84, 85, 88, 90, 90, 91, 95, 99

Since there are 20 values, the middle two values are the 10th and 11th values, which are 81 and 82. The median is the average of these two values:

Median = (81 + 82) / 2 = 81.5

Therefore, the median of the data set is 81.5.

3. Find the Mode (10pts)

Mode:

The mode is the value that appears most frequently in the data set. In this case, there are two values that appear twice, which are 76 and 82. Therefore, the mode of the data set is 76 and 82.

4. Find the Interquartile range (20pts)

Interquartile Range:

To find the interquartile range, we first need to find the first quartile (Q1) and the third quartile (Q3). The interquartile range is the difference between Q3 and Q1.

Q1 is the median of the lower half of the data set, and Q3 is the median of the upper half.

Arranging the values in ascending order again:

59, 64, 66, 70, 76, 76, 76, 78, 79, 81, 82, 82, 84, 85, 88, 90, 90, 91, 95, 99

There are 20 values, so Q1 is the median of the first 10 values, which is the average of the 5th and 6th values:

Q1 = (76 + 76) / 2 = 76

Q3 is the median of the last 10 values, which is the average of the 15th and 16th values:

Q3 = (90 + 90) / 2 = 90

The interquartile range is the difference between Q3 and Q1:

Interquartile Range = Q3 - Q1 = 90 - 76 = 14

Therefore, the interquartile range of the data set is 14.

Q.2 1) Machine Learning for Kids 2) Teachable Machine

1. For each tool listed above
    - identify the target audience
    - discuss the use of this tool by the target audience
    - identify the tool's benefits and drawbacks
2. From the two choices listed below, how would you describe each tool listed above? Why did you choose the answer?
    - Predictive analytic
    - Descriptive analytic
3. From the three choices listed below, how would you describe each tool listed above? Why did you choose the answer?

- Supervised learning
- Unsupervised learning
- Reinforcement learning

1. Machine Learning for Kids

🔹 Target Audience:

- School students (ages 8–16), educators, and beginners in machine learning.

🔹 Use by Audience:

- Teachers use it in classrooms to introduce AI concepts.

- Students build simple ML models using visual blocks (like Scratch or Python).
- Helps them classify images, text, numbers, or recognize speech.

🔹 Benefits:

- Super beginner-friendly.
- No prior coding knowledge needed.
- Encourages learning through hands-on projects.

🔹 Drawbacks:

- Limited in complexity — not suitable for advanced ML tasks.
- Restricted dataset size.
- Not as flexible for real-world model deployment.

🔹 Analytic Type:
-Predictive Analytic
Why? Students train models to make predictions — like classifying animals, recognizing text patterns, etc.

🔹 Learning Type:
- Supervised Learning
Why? Learners give labeled training data (e.g., "This is a cat") and the model learns to predict similar outcomes.

2. Teachable Machine (Google)

🔹 Target Audience:

- General users, artists, students, hobbyists, and beginners in AI.

🔹 Use by Audience:

- Users train models with webcam data, audio, or images.
- Often used in interactive projects — e.g., gesture recognition, sounds, or object detection.

🔹 Benefits:

- Extremely fast and visual.
- No code required.
- Easily exportable to TensorFlow or Arduino.

🔹 Drawbacks:

- Basic functionality — not suited for large or complex datasets.
- Limited fine-tuning options.

- Performance might not be as strong as pro-level tools.

🔹 Analytic Type:
- Predictive Analytic
Why? It learns from provided examples and predicts real-time inputs (e.g., poses, sounds).

🔹 Learning Type:
- Supervised Learning

Q.3 Data Visualization: Read the following two short articles:

- Read the article Kakande, Arthur. February 12. "What's in a chart? A Step-by-Step guide to Identifying Misinformation in Data Visualization." Medium
- Read the short web page Foley, Katherine Ellen. June 25, 2020. "How bad Covid-19 data visualizations mislead the public." Quartz
- Research a current event which highlights the results of misinformation based on data visualization. Explain how the data visualization method failed in presenting accurate information. Use newspaper articles, magazines, online news websites or any other legitimate and valid source to cite this example. Cite the news source that you found.

A common failing in data visualization is misleading audiences by altering the y-axis to exaggerate or diminish trends. For example, if the y-axis starts at a value higher than zero, a graph might falsely appear to show a drastic change when the difference is relatively small. This was seen during the COVID-19 pandemic where visualizations sometimes failed to accurately reflect the true scale of infection or mortality.

Example: Misleading COVID-19 Data Visualizations

During the COVID-19 pandemic, visualizations often appeared in news articles and social media, aiming to inform the public about the spread and severity of the virus. However, some of these visualizations used techniques that could easily mislead viewers.

How Misleading Techniques Were Used:

Manipulating the y-axis:
As explained in the articles, a common tactic was to start the y-axis (vertical axis) at a value above zero, making small changes in the data look larger. For example, if the y-axis started at 10 instead of 0, a rise from 10 to 15 would look much larger than a rise from 0 to 5.

Using inappropriate chart types:
Some visualizations used chart types that were not suitable for the data they were representing, further complicating the visualization and making it harder for the audience to understand the data.

Overly complex visuals:
Some visualizations were designed to be visually appealing but lacked clarity, making it difficult for people to understand the message they were trying to convey.

Impact of Misleading Visualizations:

Misleading data visualizations can have significant real-world consequences, particularly during crises like the COVID-19 pandemic. They can lead to:

Misinformation:
People may interpret data incorrectly, leading to misunderstandings about the actual situation.

Fear and anxiety:
If a visualization exaggerates the severity of a problem, it can create unnecessary fear and anxiety among the public.

Poor decision-making:
If people are misled by inaccurate visualizations, they may make poor decisions based on that misinformation.

Conclusion:

Data visualization is a powerful tool for communicating information, but it can also be easily misused to mislead audiences. By understanding common techniques for manipulating data visualizations, viewers can be more critical of the information they consume and make more informed decisions.

Q. 4 Train Classification Model and visualize the prediction performance of trained model required information

- Data File: Classification data.csv
- Class Label: Last Column
- Use any Machine Learning model (SVM, Naïve Base Classifier)

Requirements to satisfy

- Programming Language: Python
- Class imbalance should be resolved
- Data Pre-processing must be used
- Hyper parameter tuning must be used
- Train, Validation and Test Split should be 70/20/10
- Train and Test split must be randomly done
- Classification Accuracy should be maximized
- Use any Python library to present the accuracy measures of trained model

Pima Indians Diabetes Database

Name : Neeraj Rijhwani Roll No.64 Class: D15C

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.svm import SVC
from imblearn.over_sampling import SMOTE
import seaborn as sns
import matplotlib.pyplot as plt
```

```
[2] # Load dataset
data = pd.read_csv("diabetes.csv")
```

```
[3] # Split features and label
X = data.iloc[:, :-1]
y = data.iloc[:, -1]
```

```
[4] # Handle missing values (if any)
X.replace(0, np.nan, inplace=True)
X.fillna(X.mean(), inplace=True)
```

```
[5] # Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
# Handle class imbalance with SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_scaled, y)
```

(parameter) random_state: Any | None

```
[7] # Split into train (70%), validation (20%), test (10%)
X_train, X_temp, y_train, y_temp = train_test_split(X_resampled, y_resampled, test_size=0.3, random_state=42, stratify=y_resampled)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=1/3, random_state=42, stratify=y_temp)
```

```
[8] # Hyperparameter tuning for SVM
param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto']
}

grid = GridSearchCV(SVC(), param_grid, refit=True, verbose=0, cv=5)
grid.fit(X_train, y_train)
```

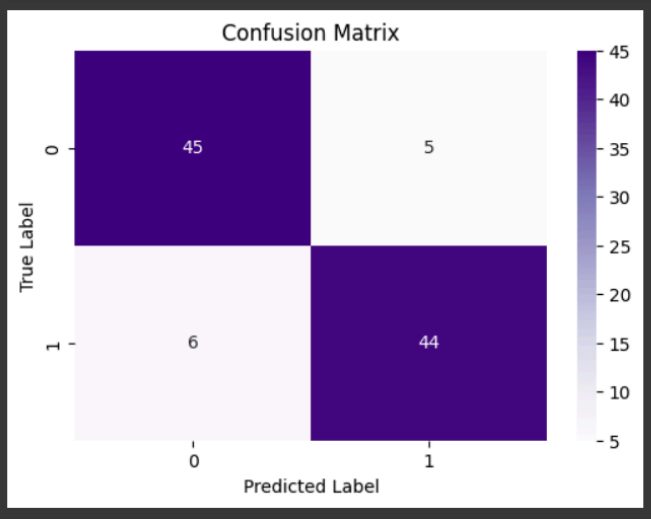
GridSearchCV
best_estimator_: SVC
SVC

```
[9]  # Best model
     model = grid.best_estimator_
```

```
[11] # Predict and evaluate on validation and test sets
     val_pred = model.predict(X_val)
     test_pred = model.predict(X_test)

     print("Validation Accuracy:", accuracy_score(y_val, val_pred))
     print("Test Accuracy:", accuracy_score(y_test, test_pred))
     print("\nClassification Report:\n", classification_report(y_test, test_pred))
```

```
Validation Accuracy: 0.81
Test Accuracy: 0.89

Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.90      0.89        50
           1       0.90      0.88      0.89        50

    accuracy                           0.89       100
   macro avg       0.89      0.89      0.89       100
weighted avg       0.89      0.89      0.89       100
```

```
# Confusion Matrix
cm = confusion_matrix(y_test, test_pred)
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Purples')
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



Q.5 Train Regression Model and visualize the prediction performance of trained model

- Data File: Regression data.csv
- Independent Variable: 1st Column
- Dependent variables: Column 2 to 5

Use any Regression model to predict the values of all Dependent variables using values of 1st column. Requirements to satisfy:

- Programming Language: Python
- OOP approach must be followed

Name : Neeraj Rijhwani Roll No.64 Class: D15C

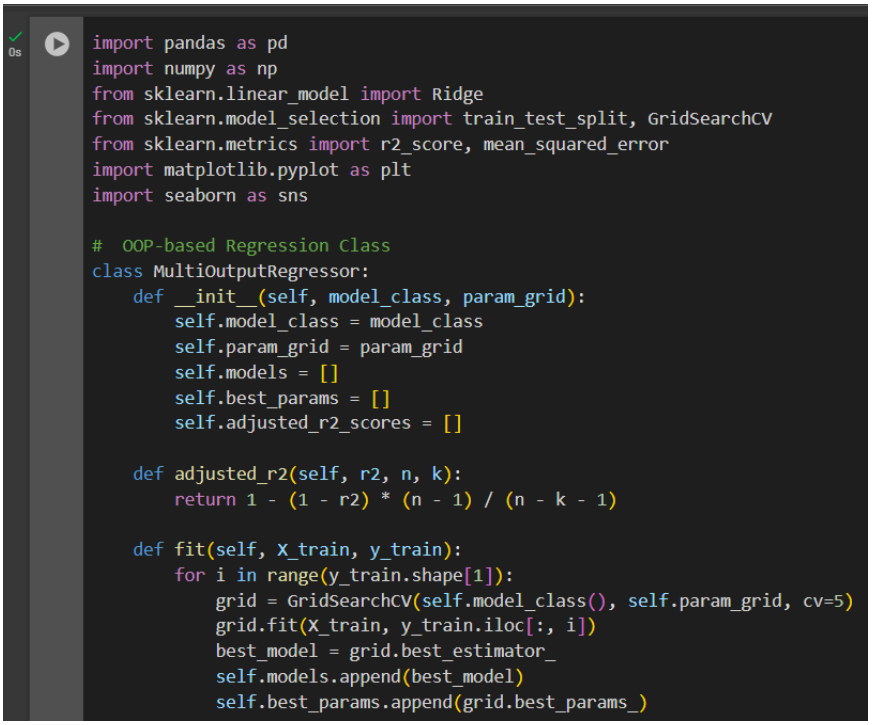
- Hyper parameter tuning must be used
- Train and Test Split should be 70/30
- Train and Test split must be randomly done
- Adjusted R2 score should more than 0.99
- Use any Python library to present the accuracy measures of trained model

https://github.com/Sutanoy/Public-Regression-Datasets

https://raw.githubusercontent.com/selva86/datasets/master/BostonHousing.csv

- URL: https://archive.ics.uci.edu/ml/machine-learning-databases/00477/Real%20estate%20valuation%20data%20set.xlsx

(Refer any one)



```python
import pandas as pd
import numpy as np
from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import r2_score, mean_squared_error
import matplotlib.pyplot as plt
import seaborn as sns

#  OOP-based Regression Class
class MultiOutputRegressor:
    def __init__(self, model_class, param_grid):
        self.model_class = model_class
        self.param_grid = param_grid
        self.models = []
        self.best_params = []
        self.adjusted_r2_scores = []

    def adjusted_r2(self, r2, n, k):
        return 1 - (1 - r2) * (n - 1) / (n - k - 1)

    def fit(self, X_train, y_train):
        for i in range(y_train.shape[1]):
            grid = GridSearchCV(self.model_class(), self.param_grid, cv=5)
            grid.fit(X_train, y_train.iloc[:, i])
            best_model = grid.best_estimator_
            self.models.append(best_model)
            self.best_params.append(grid.best_params_)
```

```
    def predict(self, X):
        predictions = []
        for model in self.models:
            predictions.append(model.predict(X))
        return np.array(predictions).T

    def evaluate(self, X_test, y_test):
        preds = self.predict(X_test)
        for i in range(y_test.shape[1]):
            r2 = r2_score(y_test.iloc[:, i], preds[:, i])
            adj_r2 = self.adjusted_r2(r2, len(X_test), 1)
            self.adjusted_r2_scores.append(adj_r2)
            print(f"Target {i+1} → R²: {r2:.5f} | Adjusted R²: {adj_r2:.5f}")

        return preds

# 📄 Load Dataset
data = pd.read_csv("Bank_Marketing.csv")

#  Define Independent and Dependent Variables
X = data.iloc[:, [0]]  # 1st column as independent
y = data.iloc[:, 1:5]  # Columns 2 to 5 as dependent

#  Train/Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# ⚙️ Define Model and Hyperparameter Grid
param_grid = {
    'alpha': [0.01, 0.1, 1, 10]
}
```

```
    'alpha': [0.01, 0.1, 1, 10]
}

#  Initialize and Train
regressor = MultiOutputRegressor(Ridge, param_grid)
regressor.fit(X_train, y_train)

#  Evaluate Model
predictions = regressor.evaluate(X_test, y_test)

#  Check if all Adjusted R² > 0.99
if all(r > 0.99 for r in regressor.adjusted_r2_scores):
    print(" All adjusted R² scores are above 0.99.")
else:
    print(" Some adjusted R² scores are below 0.99. Consider model/feature tuning.")

# Visualization of Predictions vs Actual
for i in range(y_test.shape[1]):
    plt.figure(figsize=(5, 4))
    plt.scatter(y_test.iloc[:, i], predictions[:, i], alpha=0.7, color='green')
    plt.xlabel("Actual")
    plt.ylabel("Predicted")
    plt.title(f"Target {i+1}: Actual vs Predicted")
    plt.grid(True)
    plt.plot([y_test.iloc[:, i].min(), y_test.iloc[:, i].max()],
             [y_test.iloc[:, i].min(), y_test.iloc[:, i].max()],
             'r--')
    plt.tight_layout()
    plt.show()
```

```
All adjusted R² scores are above 0.99.
```

Q.6 What are the key features of the wine quality data set? Discuss the importance of each feature in predicting the quality of wine? How did you handle missing data in the wine quality data set during the feature engineering process? Discuss the advantages and disadvantages of different imputation techniques. (Refer dataset from Kaggle).

https://www.kaggle.com/datasets/aradhanahirapara/product-retail-price-survey-2017-2025

Key Features of the Dataset

The dataset encompasses monthly retail price data for various consumer products across Canadian provinces from 2017 to 2025. The primary features include:

- Product Name:Identifies the specific product
- Category:Classifies the product into groups like food, beverages, etc
- Province:Indicates the Canadian province where the price was recorded
- Date:Specifies the month and year of the price recording
- Retail Price:The recorded price of the product
- Taxation:Details any applicable taxes on the product
- Essential Classification:Denotes whether the product is considered essential

Importance of Each Feature in Predicting Product Prices

- Product Name & Category Fundamental for identifying pricing patterns specific to products or category.
- Province Captures regional pricing variations due to factors like transportation costs and local demand.
- Date Essential for analyzing temporal trends, seasonality, and inflation effects on price.
- Taxation Influences the final retail price; variations can significantly impact pricin.
- Essential Classification Essential goods might have price controls or subsidies, affecting their pricing dynamic.

Handling Missing Data During Feature Engineering

While the dataset is comprehensive, missing data can occur. Here's how to address it:

1. Identify Missing Values

```
[15] import pandas as pd

     # Load the dataset
     df = pd.read_csv('Retail_Prices_of _Products.csv')

     # Check for missing values
     missing_values = df.isnull().sum()
     print(missing_values)
```

```
Year                0
Month               0
GEO                 0
Product Category    0
Products            0
VALUE               0
Taxable             0
Total tax rate      0
Value after tax     0
Essential           0
COORDINATE          0
UOM                 0
dtype: int64
```

2. Imputation Techniques

a. Mean/Median Imputation

- Usage: For numerical features like 'Retail Price'.

Implementation:

```
df['VALUE'] = df['VALUE'].fillna(df['VALUE'].median())
```

```
[20] print(df.columns.tolist())
```

```
['Year', 'Month', 'GEO', 'Product Category', 'Products', 'VALUE', 'Taxable', 'Total tax rate', 'Value after tax', 'Essential', 'COORDINATE', 'UOM']
```

- Pros: Simple and quick.
- Cons: Doesn't account for data variability; can skew distributions.

b. Mode Imputation

- Usage: For categorical features like 'Category' or 'Province'.

Implementation:

```
[21] # Mode imputation for 'Product Category'
     df['Product Category'] = df['Product Category'].fillna(df['Product Category'].mode()[0])
```

- Pros: Maintains the most frequent category.
- Cons: May not represent the missing data accurately if the mode is overly dominant.

c. K-Nearest Neighbors (KNN) Imputation

- Usage: Considers similarities between data points.

Implementation:

```
[22]  from sklearn.impute import KNNImputer

      imputer = KNNImputer(n_neighbors=5)
      df_imputed = pd.DataFrame(imputer.fit_transform(df.select_dtypes(include=[np.number])), columns=df.select_dtypes(include=[np.number]).columns)
```

- Pros: Accounts for data patterns.
- Cons: Computationally intensive; requires careful selection of ".

d. Dropping Missing Values

- Usage: When the proportion of missing data is minimal.

Implementation:

```
df.dropna(inplace=True)
```

- Pros: Ensures data integrity.
- Cons: Potential loss of valuable information.