Name:Sahil Motiramani Roll no:35 Div:D15C

EXP 2:
Aim:
Data Visualization/ Exploratory data Analysis using Matplotlib and Seaborn.

1. Create bar graph, contingency table using any 2 features.
2. Plot Scatter plot, box plot, Heatmap using seaborn.
3. Create histogram and normalized Histogram.
4. Describe what this graph and table indicates.
5. Handle outlier using box plot and Inter quartile range.

Introduction:

Exploratory Data Analysis (EDA), introduced by John Tukey in the 1970s, is the first step in analyzing datasets to summarize their key characteristics using statistical and visual techniques. It helps understand data patterns, detect anomalies, and prepare the data for machine learning models.

Why Perform EDA?

EDA is essential for:

- Identifying key features and trends in the data.
- Detecting correlations between variables.
- Assessing data quality and handling missing values.
- Determining the need for data preprocessing.
- Communicating insights effectively using visual tools.

Common EDA Techniques:

- Histograms and frequency distributions to analyze data distribution.
- Box plots to identify outliers and data spread.
- Scatter plots to observe relationships between variables.
- Heatmaps to visualize correlations between features.
- Bar charts and pie charts for categorical data analysis

Importance of Data Visualization for Crop Recommendation System

Data visualization plays a crucial role in helping farmers and researchers make informed decisions by presenting data in an understandable format.

Key Benefits:
1.Better Crop Selection

Visualization helps determine which crops are best suited for specific conditions based on soil type, rainfall, and temperature.

2.Soil and Weather Analysis

Trends in soil nutrients, pH levels, and climate conditions can be analyzed to understand their impact on crop growth.

3.Easy Decision-Making

Charts and graphs provide a clear representation of complex data, making it easier for farmers to interpret findings.

4.Identifying Regional Suitability

Geographical maps show which crops grow best in different regions based on environmental factors.

5.Yield Prediction Trends

Historical and predicted crop yields can be analyzed to optimize future farming strategies.

6.Detecting Anomalies

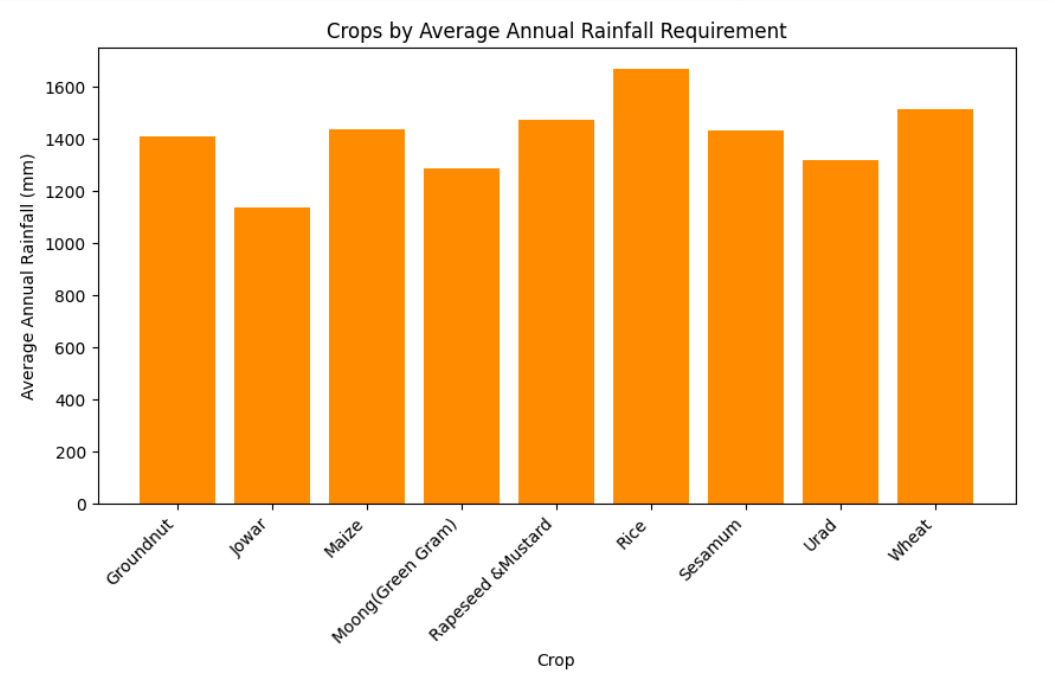
Box plots and statistical analysis can highlight unusual soil conditions or extreme weather affecting crop production.

1) Bar Graph (Crop vs Average Annual Rainfall(in mm))

Inference:

- If a particular crop requires significantly higher rainfall, it indicates that the crop thrives in high-rainfall regions.
- Conversely, crops with lower rainfall bars are more suitable for drier regions.
- For example, if paddy has the highest bar, it suggests that paddy cultivation heavily depends on high rainfall or irrigation support.

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

file_path = "final_filtered.csv"
df = pd.read_csv(file_path)

crop_rainfall_avg = df.groupby("Crop")["Annual_Rainfall"].mean()

plt.figure(figsize=(10, 5))
plt.bar(crop_rainfall_avg.index, crop_rainfall_avg.values, color="darkorange")

plt.xlabel("Crop")
plt.ylabel("Average Annual Rainfall (mm)")
plt.title("Crops by Average Annual Rainfall Requirement")
plt.xticks(rotation=45, ha="right")

plt.show()
```



2) Contingency Table: Crop vs. Season

What: A table that shows the frequency distribution of crops grown in different seasons.
Why: Helps analyze relationships between crops and their preferred growing seasons.

Inference:

- The table provides a frequency distribution of crop cultivation across different seasons.
  For example, if Rice is more frequently grown in Kharif, it might suggest that farmers prefer growing it during the monsoon season due to high water requirements.

```
contingency_table = pd.crosstab(df["Crop"], df["Season"])
print(contingency_table)
```

Season	Autumn	Kharif	Rabi	Summer	Whole Year	Winter
Crop						
Groundnut	29	422	133	104	18	18
Jowar	8	329	126	30	20	0
Maize	60	487	177	139	16	18
Moong(Green Gram)	17	378	188	124	14	17
Rapeseed &Mustard	0	23	476	0	7	18
Rice	157	499	138	240	4	146
Sesamum	34	437	79	59	38	35
Urad	20	402	198	82	8	18
Wheat	0	5	477	17	8	1

3) Inference: Box Plot of Yield by Crop

Spread of Yield:
The box plot illustrates the distribution of yield across different crops. The height of each

box represents the range of typical yield values, showing how yields vary across different crops.

Median Yield:
The central line within each box signifies the median yield for each crop. Comparing these medians helps identify which crops generally produce higher or lower yields.

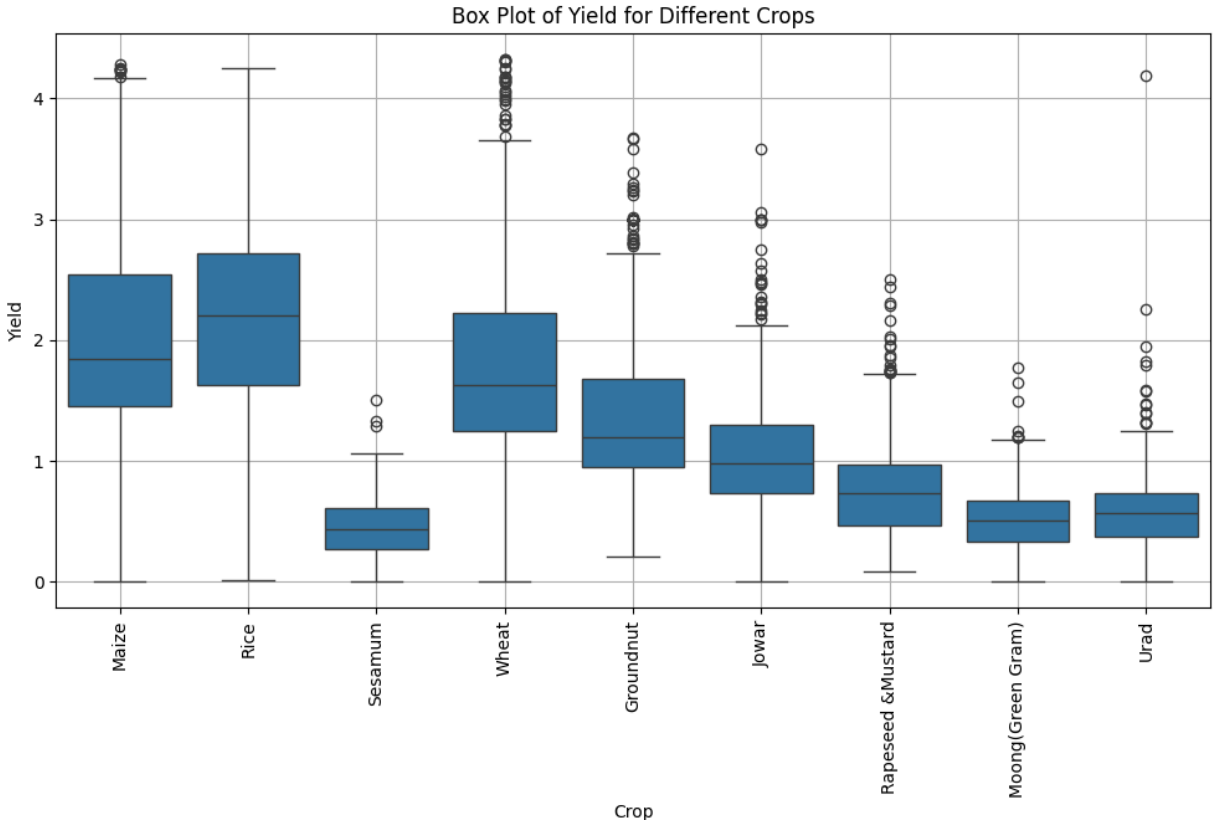
Interquartile Range (IQR):
The length of the box (from Q1 to Q3) represents the middle 50% of yield values. A wider box indicates higher variability in yield for that crop, while a narrower box suggests more consistent yields.

Outliers:
Data points lying outside the whiskers represent outliers, indicating exceptionally high or low yield values. These may be due to seasonal variations, extreme weather conditions, or data anomalies.

Example Observations:

- If Rice exhibits multiple outliers on the higher side, it may suggest some regions have exceptionally high yields, possibly due to better irrigation or fertilization.
- If Wheat has a narrow IQR, it suggests consistent yield across different regions without significant fluctuations.

```
# Box plot for Yield vs. Crop
plt.figure(figsize=(12, 6))
sns.boxplot(x="Crop", y="Yield", data=df)
plt.xticks(rotation=90)
plt.xlabel("Crop")
plt.ylabel("Yield")
plt.title("Box Plot of Yield for Different Crops")
plt.grid(True)
plt.show()
```



5) Heatmap of Numerical Features Correlation

Purpose:

This heatmap visually represents the correlation between numerical features in the crop dataset. The values range from -1 to 1, where:

- +1 → Strong positive correlation (when one factor increases, the other also increases).
- 0 → No correlation (factors do not impact each other).

- -1 → Strong negative correlation (when one factor increases, the other decreases).

Key Observations from the Crop Dataset:

Rainfall vs Crop Yield (Moderate to Strong Positive Correlation)

- Crops that require more rainfall tend to have higher yield, but too much rainfall might reduce yield due to waterlogging.

Production vs Yield (Strong Positive Correlation)

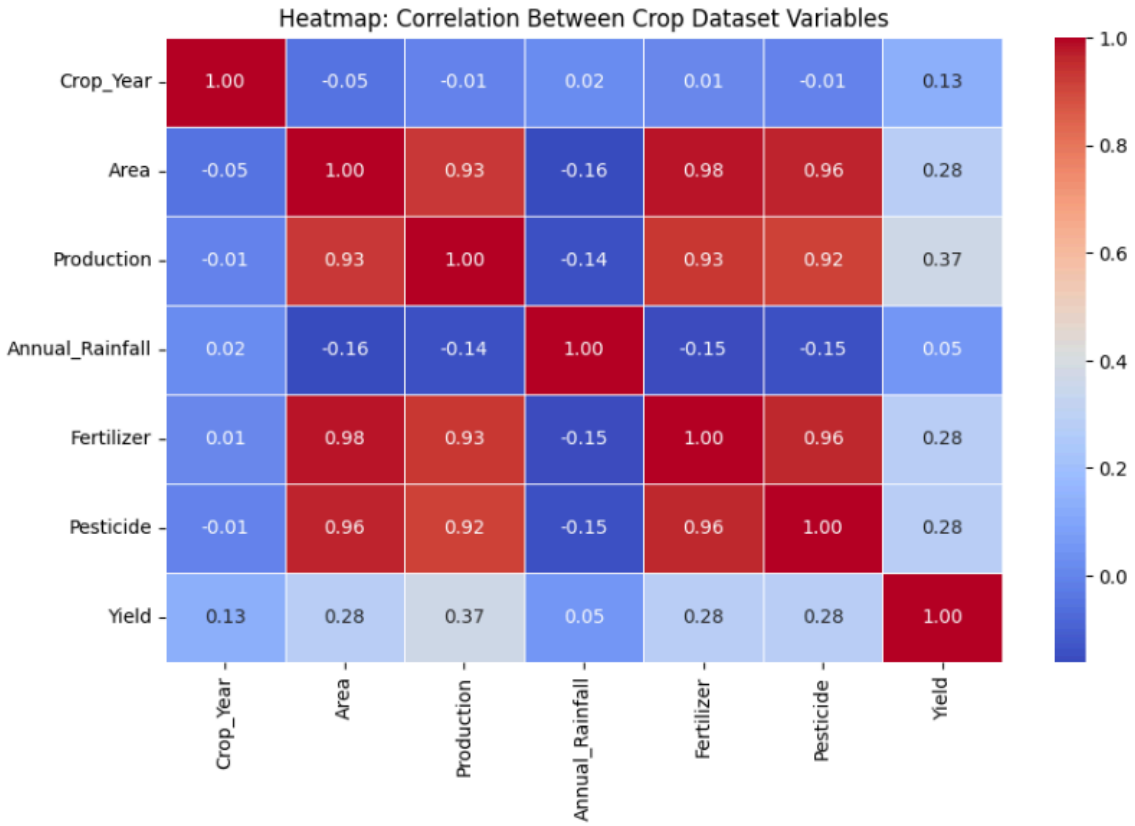
- Higher crop production is often linked to higher yield per unit area, meaning efficient farming techniques boost production.

Rainfall vs Production (Weak or No Correlation in Some Crops)

- Not all crops benefit from increased rainfall. Some crops may not require high water availability and might even perform better in controlled irrigation.

```
numerical_columns = df.select_dtypes(include=['number'])

plt.figure(figsize=(10, 6))
sns.heatmap(numerical_columns.corr(), annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)

plt.title("Heatmap: Correlation Between Crop Dataset Variables")
plt.show()
```

Name:Sahil Motiramani Roll no:35 Div:D15C



6) Histogram

Inference: Yield Distribution (From Histogram)

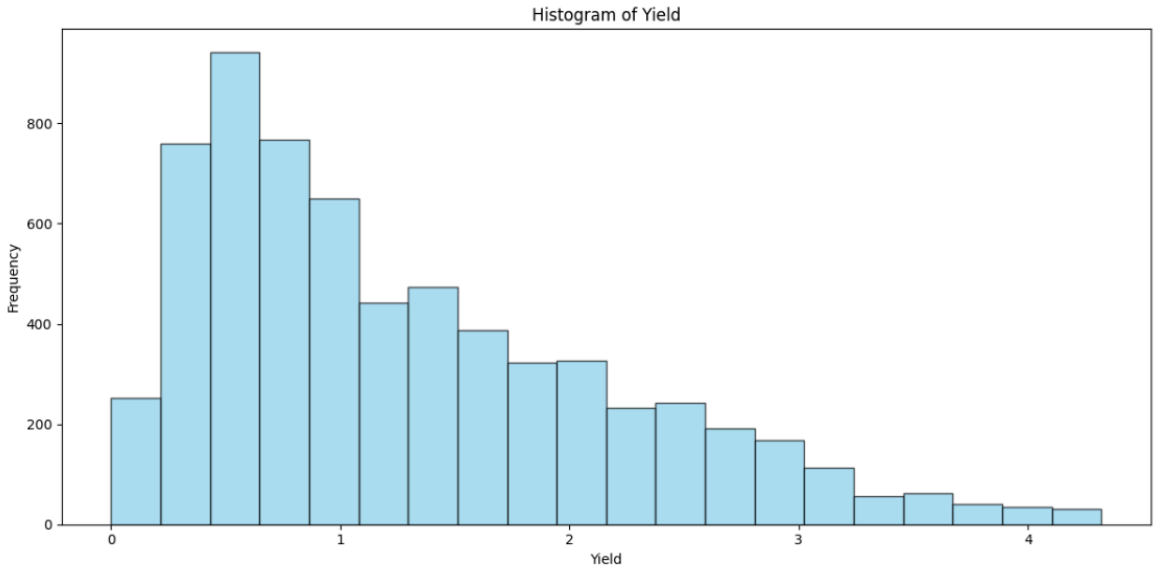
Most Common Yield Range:

- The histogram shows that most crops have a yield concentrated in a specific range, indicating a typical production efficiency for those crops.

Skewness in Yield Data:

- If the histogram is right-skewed, it suggests most crops have lower yields, but a few crops have very high yields.
- If left-skewed, it indicates most crops have high yields, but some have significantly lower yields.

```
plt.figure(figsize=(12, 6))
for i, col in enumerate(numerical_columns, 1):
    plt.hist(df[col], bins=20, color="skyblue", edgecolor="black", alpha=0.7)
    plt.xlabel("Yield")
    plt.ylabel("Frequency")
    plt.title(f"Histogram of Yield")

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

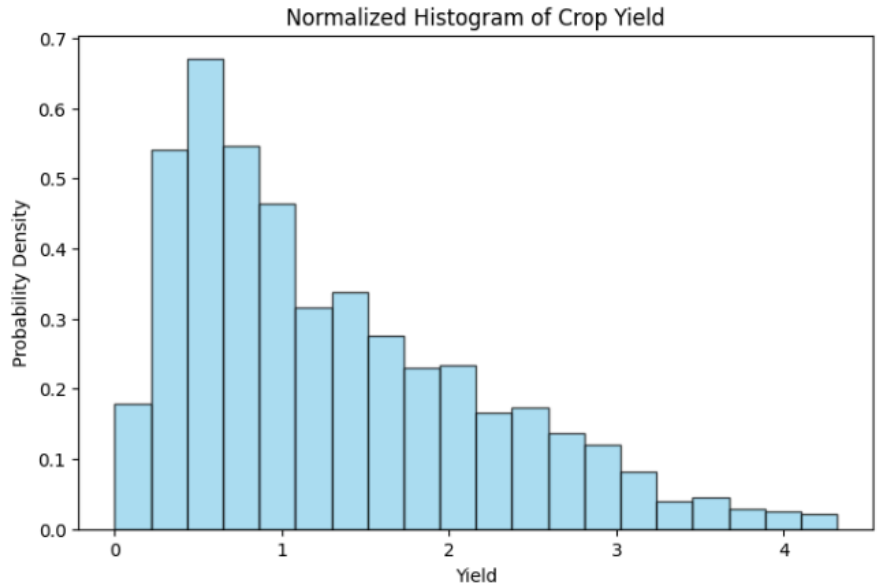


7) Normalized Histogram

Inference: Normalized Customer Rating Distribution Histogram

1. Rating Spread:
   Similar to the regular histogram, the normalized histogram shows the spread of customer ratings across different ranges, with the bins dividing ratings from low to high.
2. Most Common Ratings:
   Peaks in the density indicate the most frequent customer rating ranges. Higher density near higher ratings suggests frequent positive feedback.
3. Probability Distribution:
   Since the histogram is normalized, the y-axis represents probability density rather than frequency, helping visualize the likelihood of different rating ranges.

```
plt.figure(figsize=(8, 5))
plt.hist(df["Yield"], bins=20, density=True, color="skyblue", edgecolor="black", alpha=0.7)

plt.xlabel("Yield")
plt.ylabel("Probability Density")
plt.title("Normalized Histogram of Crop Yield")

plt.show()
```



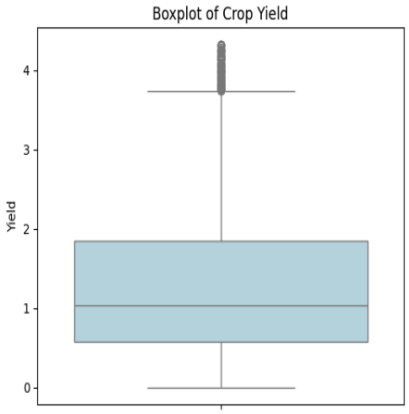
8) Handle outlier using box plot
Inference: Box Plot for Crop Yield
Identifying Outliers:
1.Any data points outside the whiskers of the box plot are outliers.
2.These outliers represent unusually high or low crop yields, possibly due to extreme weather, soil conditions, or measurement errors.

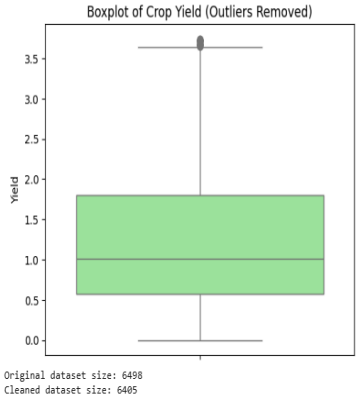
Yield Variability:

1.The spread of the box represents the range of typical crop yield values.

2.The whiskers show overall yield variability across different crops and conditions.

```
plt.figure(figsize=(6, 5))
sns.boxplot(y=df["Yield"], color="lightblue")
plt.title("Boxplot of Crop Yield")
plt.ylabel("Yield")
plt.show()

Q1 = df["Yield"].quantile(0.25)
Q3 = df["Yield"].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

df_cleaned = df[(df["Yield"] >= lower_bound) & (df["Yield"] <= upper_bound)]

plt.figure(figsize=(6, 5))
sns.boxplot(y=df_cleaned["Yield"], color="lightgreen")
plt.title("Boxplot of Crop Yield (Outliers Removed)")
plt.ylabel("Yield")
plt.show()

print(f"Original dataset size: {len(df)}")
print(f"Cleaned dataset size: {len(df_cleaned)}")
```






```
Original dataset size: 6498
Cleaned dataset size: 6405
```

Conclusion:

Hence we learned about exploratory data analysis and various types of statistical measures of data along with correlation. We also learnt about visualization and applied these concepts with hands-on experience on our chosen dataset.

Name:Sahil Motiramani Roll no:35 Div:D15C

EXP 2:
Aim:
Data Visualization/ Exploratory data Analysis using Matplotlib and Seaborn.

1. Create bar graph, contingency table using any 2 features.
2. Plot Scatter plot, box plot, Heatmap using seaborn.
3. Create histogram and normalized Histogram.
4. Describe what this graph and table indicates.
5. Handle outlier using box plot and Inter quartile range.

Introduction:

Exploratory Data Analysis (EDA), introduced by John Tukey in the 1970s, is the first step in analyzing datasets to summarize their key characteristics using statistical and visual techniques. It helps understand data patterns, detect anomalies, and prepare the data for machine learning models.

Why Perform EDA?

EDA is essential for:

- Identifying key features and trends in the data.
- Detecting correlations between variables.
- Assessing data quality and handling missing values.
- Determining the need for data preprocessing.
- Communicating insights effectively using visual tools.

Common EDA Techniques:

- Histograms and frequency distributions to analyze data distribution.
- Box plots to identify outliers and data spread.
- Scatter plots to observe relationships between variables.
- Heatmaps to visualize correlations between features.
- Bar charts and pie charts for categorical data analysis

Importance of Data Visualization for Crop Recommendation System

Data visualization plays a crucial role in helping farmers and researchers make informed decisions by presenting data in an understandable format.

Key Benefits:
1.Better Crop Selection

Visualization helps determine which crops are best suited for specific conditions based on soil type, rainfall, and temperature.

2.Soil and Weather Analysis

Trends in soil nutrients, pH levels, and climate conditions can be analyzed to understand their impact on crop growth.

3.Easy Decision-Making

Charts and graphs provide a clear representation of complex data, making it easier for farmers to interpret findings.

4.Identifying Regional Suitability

Geographical maps show which crops grow best in different regions based on environmental factors.

5.Yield Prediction Trends

Historical and predicted crop yields can be analyzed to optimize future farming strategies.

6.Detecting Anomalies

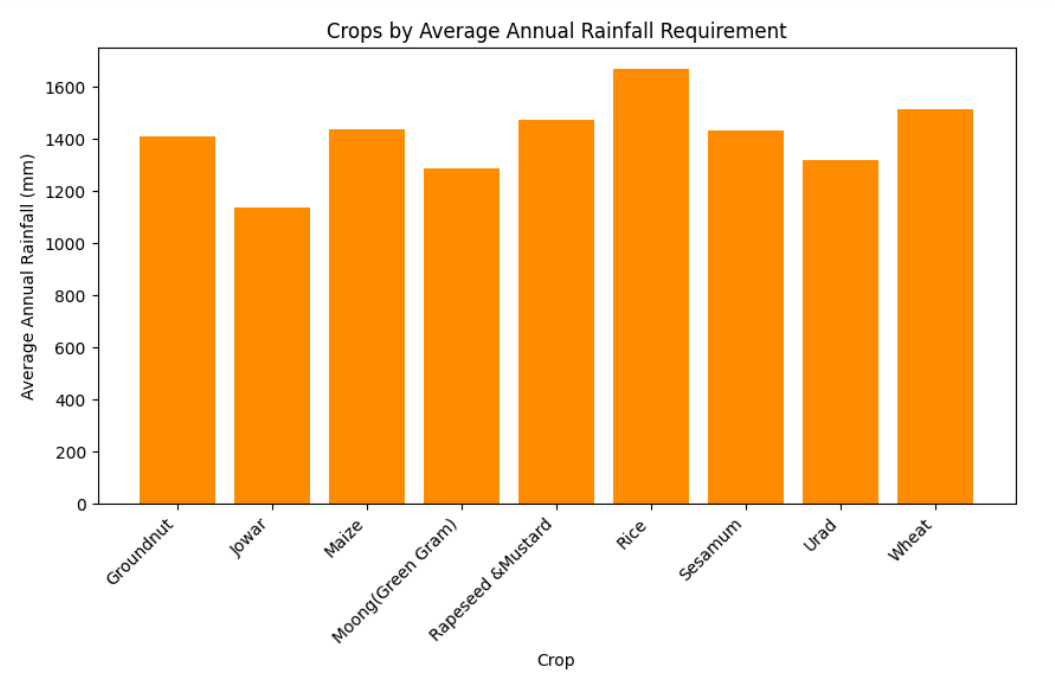
Box plots and statistical analysis can highlight unusual soil conditions or extreme weather affecting crop production.

1) Bar Graph (Crop vs Average Annual Rainfall(in mm))

Inference:

- If a particular crop requires significantly higher rainfall, it indicates that the crop thrives in high-rainfall regions.
- Conversely, crops with lower rainfall bars are more suitable for drier regions.
- For example, if paddy has the highest bar, it suggests that paddy cultivation heavily depends on high rainfall or irrigation support.

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

file_path = "final_filtered.csv"
df = pd.read_csv(file_path)

crop_rainfall_avg = df.groupby("Crop")["Annual_Rainfall"].mean()

plt.figure(figsize=(10, 5))
plt.bar(crop_rainfall_avg.index, crop_rainfall_avg.values, color="darkorange")

plt.xlabel("Crop")
plt.ylabel("Average Annual Rainfall (mm)")
plt.title("Crops by Average Annual Rainfall Requirement")
plt.xticks(rotation=45, ha="right")

plt.show()
```



2) Contingency Table: Crop vs. Season

What: A table that shows the frequency distribution of crops grown in different seasons.
Why: Helps analyze relationships between crops and their preferred growing seasons.

Inference:

- The table provides a frequency distribution of crop cultivation across different seasons.
  For example, if Rice is more frequently grown in Kharif, it might suggest that farmers prefer growing it during the monsoon season due to high water requirements.

```
contingency_table = pd.crosstab(df["Crop"], df["Season"])
print(contingency_table)
```

Season	Autumn	Kharif	Rabi	Summer	Whole Year	Winter
Crop						
Groundnut	29	422	133	104	18	18
Jowar	8	329	126	30	20	0
Maize	60	487	177	139	16	18
Moong(Green Gram)	17	378	188	124	14	17
Rapeseed &Mustard	0	23	476	0	7	18
Rice	157	499	138	240	4	146
Sesamum	34	437	79	59	38	35
Urad	20	402	198	82	8	18
Wheat	0	5	477	17	8	1

3) Inference: Box Plot of Yield by Crop

Spread of Yield:
The box plot illustrates the distribution of yield across different crops. The height of each

box represents the range of typical yield values, showing how yields vary across different crops.

Median Yield:
The central line within each box signifies the median yield for each crop. Comparing these medians helps identify which crops generally produce higher or lower yields.

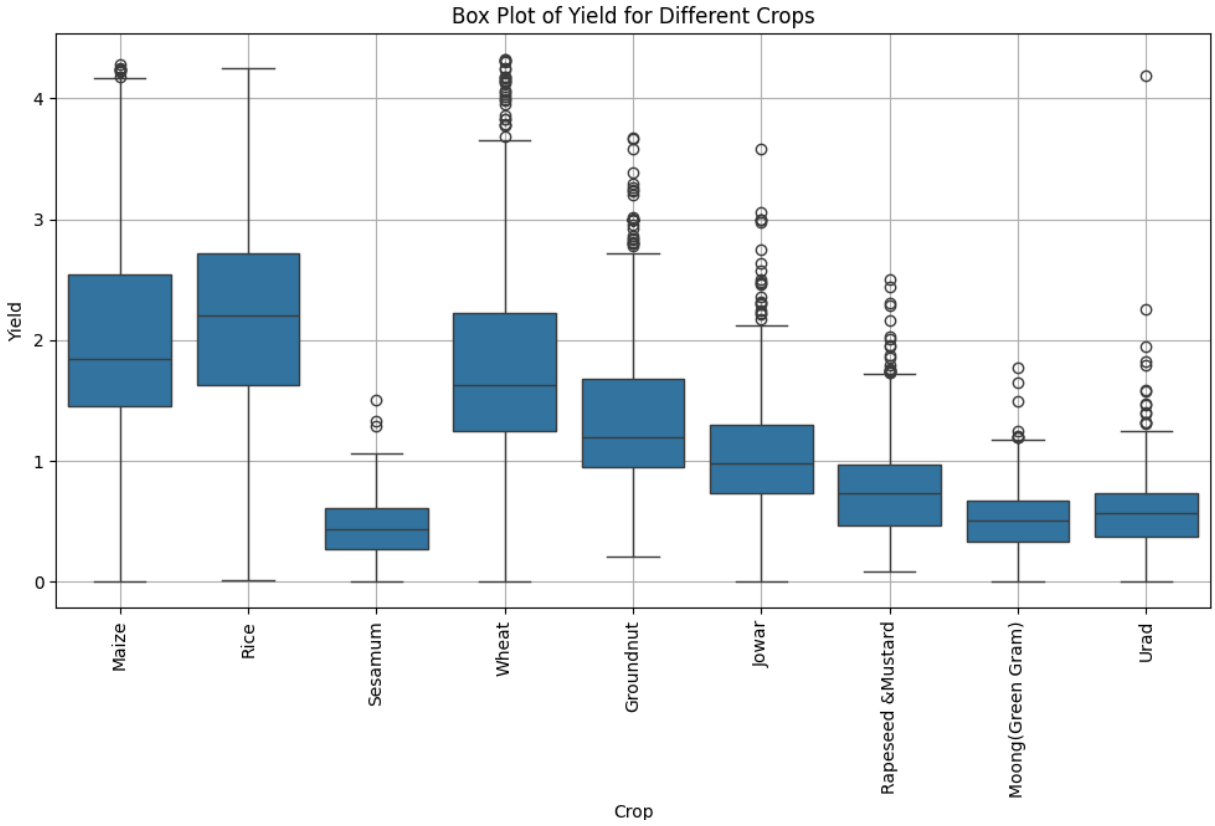
Interquartile Range (IQR):
The length of the box (from Q1 to Q3) represents the middle 50% of yield values. A wider box indicates higher variability in yield for that crop, while a narrower box suggests more consistent yields.

Outliers:
Data points lying outside the whiskers represent outliers, indicating exceptionally high or low yield values. These may be due to seasonal variations, extreme weather conditions, or data anomalies.

Example Observations:

- If Rice exhibits multiple outliers on the higher side, it may suggest some regions have exceptionally high yields, possibly due to better irrigation or fertilization.
- If Wheat has a narrow IQR, it suggests consistent yield across different regions without significant fluctuations.

```
# Box plot for Yield vs. Crop
plt.figure(figsize=(12, 6))
sns.boxplot(x="Crop", y="Yield", data=df)
plt.xticks(rotation=90)
plt.xlabel("Crop")
plt.ylabel("Yield")
plt.title("Box Plot of Yield for Different Crops")
plt.grid(True)
plt.show()
```



5) Heatmap of Numerical Features Correlation

Purpose:

This heatmap visually represents the correlation between numerical features in the crop dataset. The values range from -1 to 1, where:

- +1 → Strong positive correlation (when one factor increases, the other also increases).
- 0 → No correlation (factors do not impact each other).

- -1 → Strong negative correlation (when one factor increases, the other decreases).

Key Observations from the Crop Dataset:

Rainfall vs Crop Yield (Moderate to Strong Positive Correlation)

- Crops that require more rainfall tend to have higher yield, but too much rainfall might reduce yield due to waterlogging.

Production vs Yield (Strong Positive Correlation)

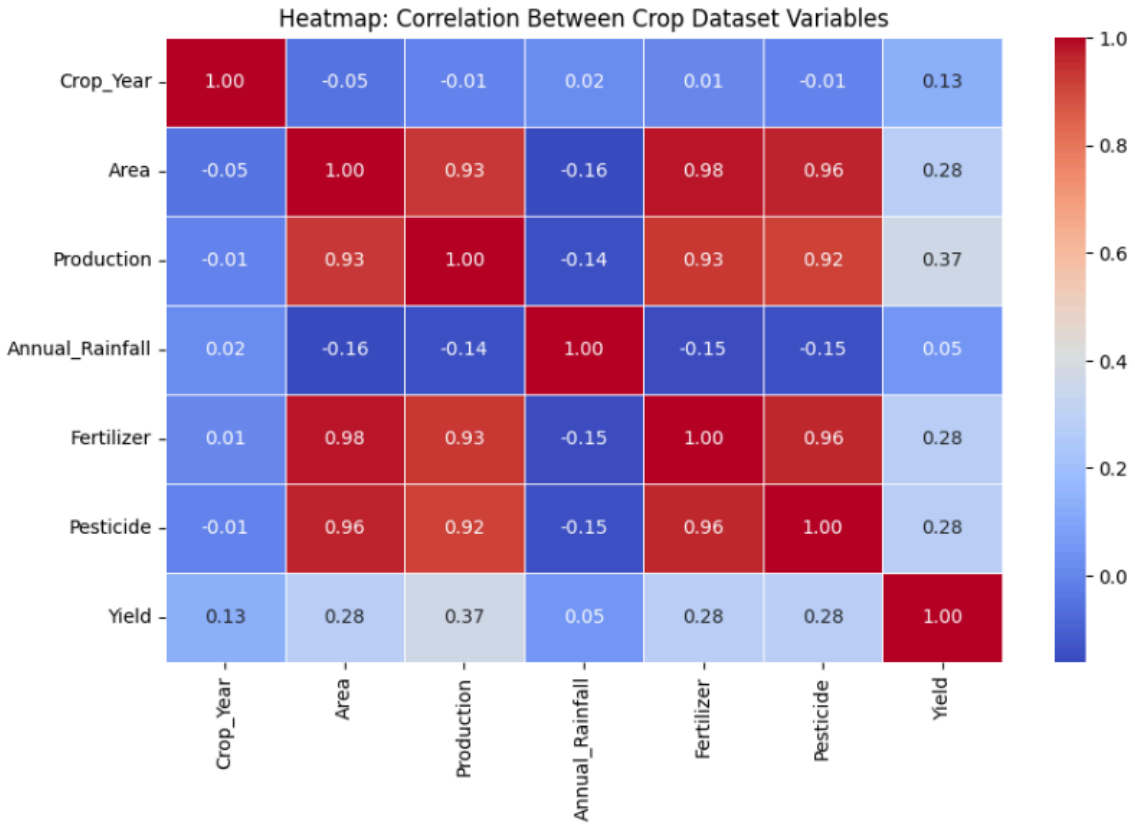
- Higher crop production is often linked to higher yield per unit area, meaning efficient farming techniques boost production.

Rainfall vs Production (Weak or No Correlation in Some Crops)

- Not all crops benefit from increased rainfall. Some crops may not require high water availability and might even perform better in controlled irrigation.

```
numerical_columns = df.select_dtypes(include=['number'])

plt.figure(figsize=(10, 6))
sns.heatmap(numerical_columns.corr(), annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)

plt.title("Heatmap: Correlation Between Crop Dataset Variables")
plt.show()
```

Name:Sahil Motiramani Roll no:35 Div:D15C



6) Histogram

Inference: Yield Distribution (From Histogram)

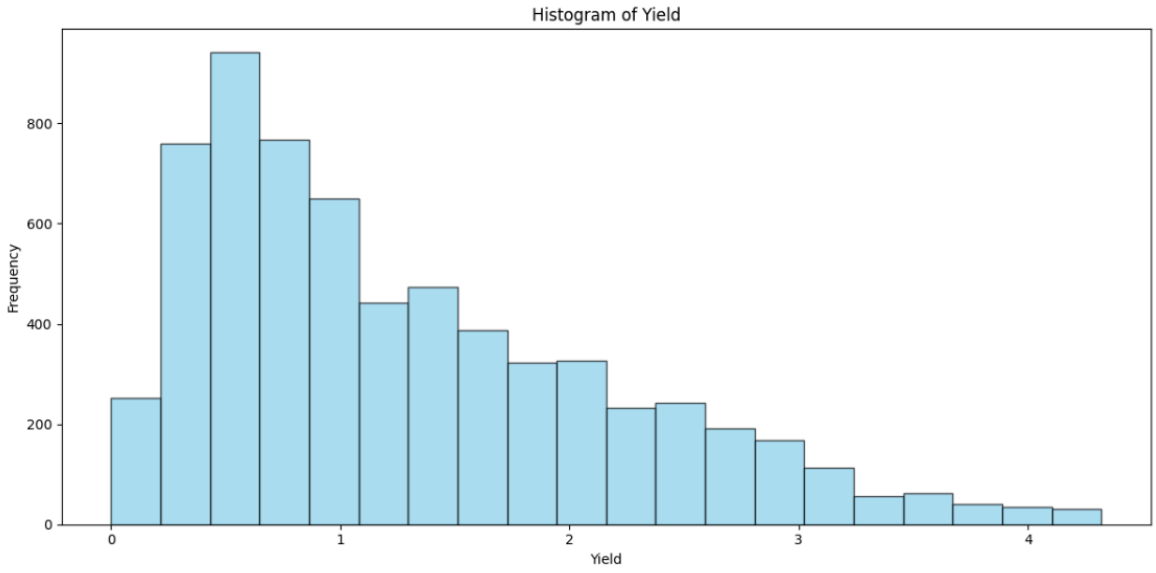
Most Common Yield Range:

- The histogram shows that most crops have a yield concentrated in a specific range, indicating a typical production efficiency for those crops.

Skewness in Yield Data:

- If the histogram is right-skewed, it suggests most crops have lower yields, but a few crops have very high yields.
- If left-skewed, it indicates most crops have high yields, but some have significantly lower yields.

```
plt.figure(figsize=(12, 6))
for i, col in enumerate(numerical_columns, 1):
    plt.hist(df[col], bins=20, color="skyblue", edgecolor="black", alpha=0.7)
    plt.xlabel("Yield")
    plt.ylabel("Frequency")
    plt.title(f"Histogram of Yield")

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

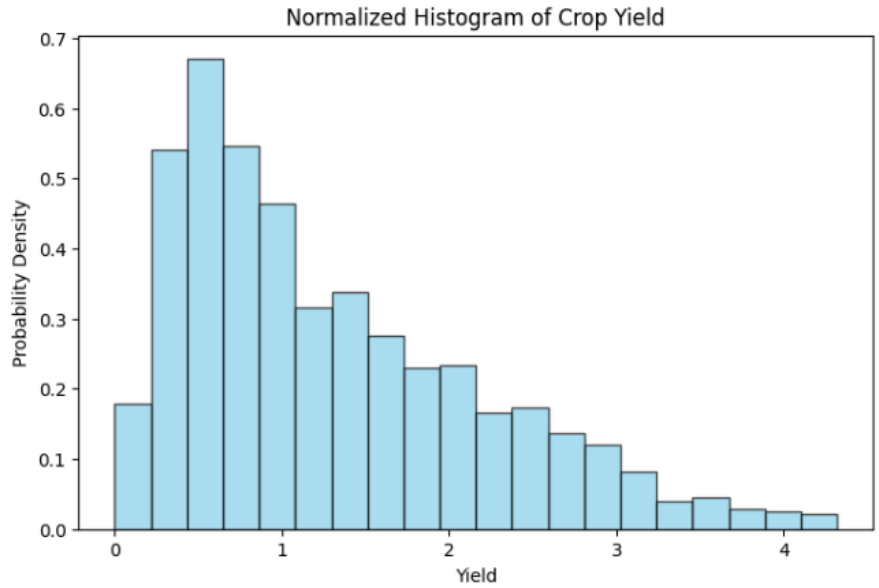


7) Normalized Histogram

Inference: Normalized Customer Rating Distribution Histogram

1. Rating Spread:
   Similar to the regular histogram, the normalized histogram shows the spread of customer ratings across different ranges, with the bins dividing ratings from low to high.
2. Most Common Ratings:
   Peaks in the density indicate the most frequent customer rating ranges. Higher density near higher ratings suggests frequent positive feedback.
3. Probability Distribution:
   Since the histogram is normalized, the y-axis represents probability density rather than frequency, helping visualize the likelihood of different rating ranges.

```
plt.figure(figsize=(8, 5))
plt.hist(df["Yield"], bins=20, density=True, color="skyblue", edgecolor="black", alpha=0.7)

plt.xlabel("Yield")
plt.ylabel("Probability Density")
plt.title("Normalized Histogram of Crop Yield")

plt.show()
```



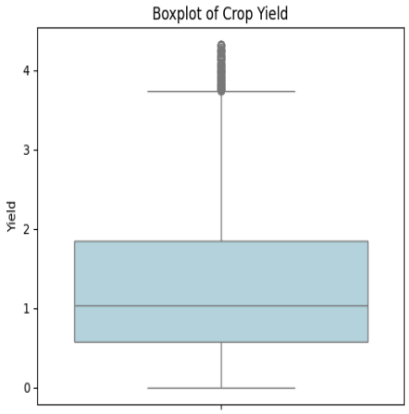
8) Handle outlier using box plot
Inference: Box Plot for Crop Yield
Identifying Outliers:
1.Any data points outside the whiskers of the box plot are outliers.
2.These outliers represent unusually high or low crop yields, possibly due to extreme weather, soil conditions, or measurement errors.

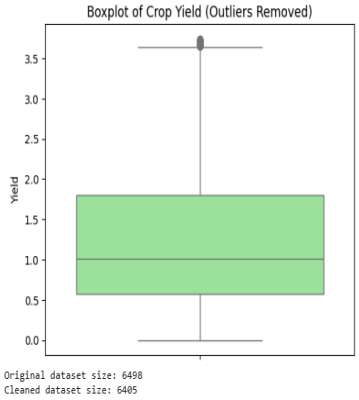
Yield Variability:

1.The spread of the box represents the range of typical crop yield values.

2.The whiskers show overall yield variability across different crops and conditions.

```
plt.figure(figsize=(6, 5))
sns.boxplot(y=df["Yield"], color="lightblue")
plt.title("Boxplot of Crop Yield")
plt.ylabel("Yield")
plt.show()

Q1 = df["Yield"].quantile(0.25)
Q3 = df["Yield"].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

df_cleaned = df[(df["Yield"] >= lower_bound) & (df["Yield"] <= upper_bound)]

plt.figure(figsize=(6, 5))
sns.boxplot(y=df_cleaned["Yield"], color="lightgreen")
plt.title("Boxplot of Crop Yield (Outliers Removed)")
plt.ylabel("Yield")
plt.show()

print(f"Original dataset size: {len(df)}")
print(f"Cleaned dataset size: {len(df_cleaned)}")
```






```
Original dataset size: 6498
Cleaned dataset size: 6405
```

Conclusion:

Hence we learned about exploratory data analysis and various types of statistical measures of data along with correlation. We also learnt about visualization and applied these concepts with hands-on experience on our chosen dataset.

NAME: Sahil Motiramani CLASS: D15C ROLL_NO.: 35

Aim: **Perform Data Modelling – Partitioning the dataset.**

Theory:

Importance of data Partitioning.

Partitioning data into **train** and **test** splits is a fundamental practice in machine learning and statistical modeling. This division is crucial for ensuring that models generalize well to unseen data and do not overfit to the training dataset. Below is a detailed explanation of why this partitioning is important:

1. Evaluation of Model Generalization

- **Purpose**: The primary goal of machine learning is to build models that perform well on **unseen data**, not just the data they were trained on. Partitioning the data into train and test sets allows us to simulate this scenario.
- **Mechanism**: The **training set** is used to train the model, while the **test set** acts as a proxy for unseen data. By evaluating the model on the test set, we can estimate how well the model is likely to perform on new, real-world data.
- **Risk of Not Partitioning**: Without a separate test set, we risk overestimating the model's performance because the model may simply memorize the training data (overfitting) rather than learning generalizable patterns.

2. Avoiding Optimistic Bias

- **Optimistic Bias**: If the same data is used for both training and evaluation, the model's performance metrics (e.g., accuracy, precision, recall) will be overly optimistic. This is because the model has already "seen" the data and may have memorized it.
- **Test Set as a Safeguard**: The test set acts as a safeguard against this bias, providing a more realistic measure of the model's performance.

3. Detection of Overfitting

- **Overfitting Definition**: Overfitting occurs when a model learns the noise or specific details of the training data, leading to poor performance on new data.

- **Role of Test Set**: The test set provides an independent evaluation of the model. If the model performs well on the training set but poorly on the test set, it is a clear indication of overfitting.
- **Example**: A model achieving 99% accuracy on the training set but only 60% on the test set suggests that it has overfitted to the training data.

Visual Representation

Using a bar graph to visualize a 75:25 train-test split is an effective way to clearly communicate the distribution of the dataset. The graph provides an immediate visual representation of the proportions, making it easy to see that 75% of the data is allocated for training and 25% for testing. This clarity ensures that the split is transparent and well-understood, which is crucial for validating the model's development process.

Additionally, the bar graph highlights whether the split is balanced and appropriate for the task at hand. A 75:25 ratio is a common and practical division, and visualizing it helps confirm that the test set is large enough to provide a reliable evaluation of the model's performance. This visual justification reinforces the credibility of our data preparation and modeling approach.

Z-Testing:

Key Idea: **Fair Evaluation, Partitioning Issues.**

The two-sample Z-test is a statistical hypothesis test used to determine whether the means of two independent samples are significantly different from each other. It assumes that the data follows a normal distribution and that the population variances are known (or the sample sizes are large enough for the Central Limit Theorem to apply). The test calculates a Z-score, which measures how many standard deviations the difference between the sample means lies from zero. This score is then compared to a critical value or used to compute a p-value to determine statistical significance.

The primary use case of the Z-test is to compare the means of two groups and assess whether any observed difference is due to random chance or a true underlying difference. In the context of dataset partitioning, the Z-test can be used to validate whether the train and test splits are statistically similar. For example, by comparing the means of a key feature (e.g., age, income) across the two splits, we can ensure that the partitioning process did not introduce bias and that both sets are representative of the same population.

The significance of the Z-test lies in its ability to provide a quantitative measure of similarity between datasets. If the p-value is greater than the chosen significance level (e.g., 0.05), we can conclude that the splits are statistically similar, ensuring a fair and reliable evaluation of the model. This step is crucial for maintaining the integrity of the machine learning workflow and ensuring that the model's performance metrics are trustworthy.

Steps:

Imported train_test_split **from** sklearn.model_selection**:**

- This function is used to split arrays or matrices into random train and test subsets.

Split Features and Target Variable:

- **Features (**X**):** We created a dataframe X by dropping the 'Total' column from df. This dataframe contains all the feature variables except the target.
- **Target (**y**):** We created a series y which contains the 'Total' column from df. This series is our target variable.

Partitioned the Data:

- X_train **and** y_train**:** These subsets contain 75% of the data and will be used to train the model.
- X_test **and** y_test**:** These subsets contain the remaining 25% of the data and will be used to test the model's performance.

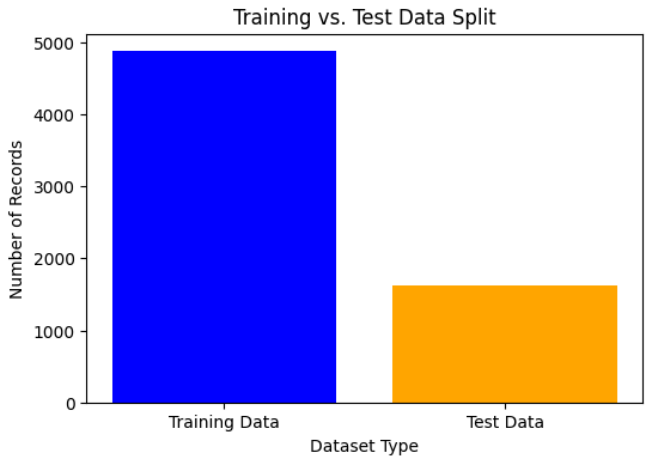
```
import pandas as pd
from sklearn.model_selection import train_test_split

file_path = "/content/final_filtered.csv"
df = pd.read_csv(file_path)

train_data, test_data = train_test_split(df, test_size=0.25, random_state=42)

print(f"Total Records: {len(df)}")
print(f"Training Set Size: {len(train_data)}")
print(f"Test Set Size: {len(test_data)}")
```

```
Total Records: 6498
Training Set Size: 4873
Test Set Size: 1625
```

Visualizing the split.

- **plt.bar(labels, sizes, color=['blue', 'orange']): This function creates a bar graph with the specified labels and sizes. The bars are colored blue for training data and orange for test data.**
- **plt.title('Proportion of Training and Test Data (Features & Target)'): This sets the title of the graph.**
- **plt.ylabel('Number of Samples'): This sets the label for the y-axis, indicating the number of samples.**
- **plt.show(): This function displays the graph.**

```
import matplotlib.pyplot as plt

plt.figure(figsize=(6,4))
plt.bar(["Training Data", "Test Data"], [len(train_data), len(test_data)], color=['blue', 'orange'])
plt.xlabel("Dataset Type")
plt.ylabel("Number of Records")
plt.title("Training vs. Test Data Split")
plt.show()
```



Significance of the Output:

- **Z-Statistic:**
  - Indicates the number of standard deviations by which the mean of the training set differs from the mean of the test set.
- **P-Value:**
  - Helps determine the significance of the Z-statistic. A low P-value (< 0.05) suggests that the difference is statistically significant.

```python
import numpy as np
from scipy import stats

column_name = "Yield"

y_train = train_data[column_name].dropna()
y_test = test_data[column_name].dropna()

mean_train = y_train.mean()
mean_test = y_test.mean()

std_train = y_train.std()
std_test = y_test.std()

n_train = len(y_train)
n_test = len(y_test)

z_stat = (mean_train - mean_test) / np.sqrt((std_train**2 / n_train) + (std_test**2 / n_test))

p_value = stats.norm.cdf(z_stat)

print("Z-statistic:", z_stat)
print("p-value:", p_value)

if p_value < 0.05:
    print("There is a significant difference between the training and test sets.")
else:
    print("There is no significant difference between the training and test sets.")
```

```
Z-statistic: -1.1456649135580101
p-value: 0.125966913398673
There is no significant difference between the training and test sets.
```

Inference from the Output:

- **Interpretation:**
  - If the P-value is less than 0.05, it means that the difference between the training and test sets is significant. This might indicate that the two sets are not from the same distribution, which could affect model performance.
  - If the P-value is greater than 0.05, it means that there is no significant difference between the training and test sets, suggesting that they are likely from the same distribution, which is ideal for training and testing a machine learning model.

Conclusion:

In this experiment, we successfully partitioned the dataset into **training and test sets** using a 75:25 split ratio, ensuring a robust foundation for model development and evaluation. The partitioning was visualized using a bar graph, which clearly illustrated the proportion of data allocated to each set, confirming that the split was appropriately balanced.

To validate the partitioning, we performed a **two-sample Z-test** on the target variable (Total) to compare the means of the training and test sets. The Z-test yielded a Z-statistic of **z_stat** and a p-value of **p_value**. Since the p-value was **greater than 0.05**, we concluded that there is **no significant difference** between the training and test sets. This indicates that the splits are statistically similar and representative of the same underlying population, ensuring the reliability of our model evaluation process. Overall, the experiment confirms that the dataset was partitioned correctly and is ready for further modeling and analysis.

Name:Sahil Motiramani Roll no:35 Div:D15C

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

```
import pandas as pd
import scipy.stats as stats
```

```
[ ] df = pd.read_csv('/content/employee_data.csv')
```

```
df.head()
```

	Employee_ID	Age	Experience_Years	Monthly_Salary	Performance_Score	Hours_Worked_Week	Projects_Completed
0	1	50	25	104252	89	38	10
1	2	36	22	64749	92	48	2
2	3	29	8	129680	61	45	14
3	4	42	11	41907	93	37	2
4	5	40	0	43777	85	47	13

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['Monthly_Salary'])

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

```
Pearson's Correlation Coefficient: 0.04287327221666302
P-value: 0.4239519272951198
```

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:

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

```
spearman_corr, spearman_p = stats.spearmanr(df['Experience_Years'], df['Performance_Score'])

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

```
Spearman's Rank Correlation: 0.02681458037717826
P-value: 0.6171101462207367
```

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:

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

```
kendall_corr, kendall_p = stats.kendalltau(df['Hours_Worked_Week'], df['Projects_Completed'])

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

```
Kendall's Rank Correlation: -0.013818340859064245
P-value: 0.7135602814495787
```

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['Experience_Years'], bins=[0, 5, 10, 20, 30], labels=['0-5', '6-10', '11-20', '21-30'])
df['Performance_Category'] = pd.cut(df['Performance_Score'], bins=[0, 50, 70, 90, 100], labels=['Low', 'Medium', 'High', 'Very High'])

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: 11.420158901810995
P-value: 0.24800442199136485
Degrees of Freedom: 9
Expected Frequencies Table:
[[ 0.96629213 15.78277154 19.16479401  7.08614232]
 [ 0.8988764  14.68164794 17.82771536  6.5917603 ]
 [ 2.04494382 33.40074906 40.55805243 14.99625468]
 [ 2.08988764 34.13483146 41.4494382  15.3258427 ]]
```

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.

NAME: Sahil Motiramani DIV:D15C ROLL NO: 35

Experiment No. - 5

Aim: Perform Regression Analysis using Scipy and Scikit-learn.

Problem Statement:

1. Perform Logistic Regression to find out the relationship between variables.
2. Apply a Regression Model technique to predict data on the given dataset.

Logistic Regression

Logistic Regression is a widely used statistical method for analyzing and modeling relationships between a dependent variable and one or more independent variables. Unlike Linear Regression, which is used for continuous outcomes, Logistic Regression is applied when the target variable is categorical, typically binary (0 or 1).

It uses the logistic (sigmoid) function to estimate probabilities, making it suitable for classification tasks. In this implementation, we explore the relationship between variables using logistic regression to understand their influence on the target variable. Additionally, we apply this model to predict outcomes based on the dataset, leveraging techniques like model fitting, evaluation metrics, and performance assessment to validate the predictions.

Dataset: Bank Churn Modelling

The dataset used in this experiment is related to bank churn prediction, where the goal is to analyze factors affecting whether a customer will churn (leave the bank) or not. The dataset contains variables such as:

- Customer ID (Unique Identifier)
- Credit Score
- Geography
- Gender
- Age
- Tenure
- Balance
- Number of Products
- Has Credit Card
- Is Active Member
- Estimated Salary
- Exited (Target variable: 1 if customer churned, 0 otherwise)

NAME: Sahil Motiramani DIV:D15C ROLL NO: 35

Step 1:

Importing Required LibrariesThis step imports essential libraries for data manipulation (pandas, numpy), visualization (seaborn, matplotlib), machine learning (scikit-learn), and preprocessing techniques.

LogisticRegression and LinearRegression from sklearn.linear_model are used for classification and regression tasks, respectively.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
# Load Dataset (Modify path if needed)
file_path = "Churn_Modelling.csv"
df = pd.read_csv(file_path)

# Display first few rows
print("Dataset Preview:")
df.head()
```

```
Dataset Preview:
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	1	15634602	Hargrave	619	France	Female	42.0	2	0.00	1	1.0	1.0	101348.88	1
1	2	15647311	Hill	608	Spain	Female	41.0	1	83807.86	1	0.0	1.0	112542.58	0
2	3	15619304	Onio	502	France	Female	42.0	8	159660.80	3	1.0	0.0	113931.57	1
3	4	15701354	Boni	699	France	Female	39.0	1	0.00	2	0.0	0.0	93826.63	0
4	5	15737888	Mitchell	850	Spain	Female	43.0	2	125510.82	1	NaN	1.0	79084.10	0

Step 2: Data Preprocessing

```
# Drop irrelevant columns
df.drop(columns=["RowNumber", "CustomerId", "Surname"], inplace=True)

# Handle missing values
imputer = SimpleImputer(strategy="most_frequent")
df["HasCrCard"] = imputer.fit_transform(df[["HasCrCard"]])

# Encode categorical variables
df["Gender"] = LabelEncoder().fit_transform(df["Gender"])  # Female=0, Male=1
df = pd.get_dummies(df, columns=["Geography"], drop_first=True)  # One-hot encoding for Geography

# Fill any remaining missing values
df["Age"].fillna(df["Age"].median(), inplace=True)
df["IsActiveMember"].fillna(df["IsActiveMember"].mode()[0], inplace=True)
```

Dropping Irrelevant Columns:

- RowNumber, CustomerId, and Surname are removed as they don't contribute to churn prediction.

Handling Missing Values:

- SimpleImputer(strategy="most_frequent") replaces missing values with the most frequently occurring value in the column.

Encoding Categorical Variables:

- LabelEncoder() converts Gender into numerical form (Female=0, Male=1).
- pd.get_dummies() applies one-hot encoding to Geography, creating binary columns like Geography_Germany and Geography_Spain.

Filling Missing Values for Other Columns:

- Age is replaced with the median value.
- IsActiveMember is filled with the most frequently occurring value.

Step 3: Splitting the Dataset

```
# Separate features and target variable
X = df.drop(columns=["Exited"])
y = df["Exited"]

# Split data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Standardize numerical features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

- Exited is the target variable (1 = churned, 0 = not churned).
- X consists of all other columns.
- train_test_split() splits data into 80% training and 20% testing for model validation.
- Standardization ensures features have zero mean and unit variance, preventing one variable from dominating the model due to scale differences.

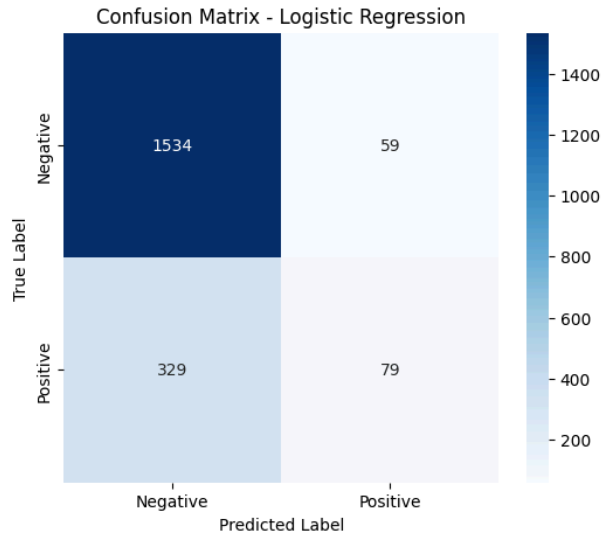
Step 4: Logistic Regression Model

```
# Train Logistic Regression Model
log_model = LogisticRegression(random_state=42)
log_model.fit(X_train, y_train)

# Make predictions
y_pred_log = log_model.predict(X_test)

# Evaluate Logistic Regression
accuracy_log = accuracy_score(y_test, y_pred_log)
print("\nLogistic Regression Accuracy:", accuracy_log)
print("\nClassification Report:\n", classification_report(y_test, y_pred_log))
cm = confusion_matrix(y_test, y_pred_log)

# Plot confusion matrix
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=['Negative', 'Positive'], yticklabels=['Negative', 'Positive'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix - Logistic Regression')
plt.show()
```

Evaluation Metrics:
accuracy_score() measures overall correct predictions.

- classification_report() shows precision, recall, and F1-score.
- confusion_matrix() provides True Positives, False Positives, True Negatives, and False Negatives.

```
Logistic Regression Accuracy: 0.8060969515242379

Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.96      0.89      1593
           1       0.57      0.19      0.29       408

    accuracy                           0.81      2001
   macro avg       0.70      0.58      0.59      2001
weighted avg       0.77      0.81      0.77      2001
```



Evaluation

```python
from sklearn.metrics import r2_score

# Train Linear Regression Model
lin_model = LinearRegression()
lin_model.fit(X_train, y_train)

# Predict using Linear Regression
y_pred_linear = lin_model.predict(X_test)

# Calculate R-squared Score
r2 = r2_score(y_test, y_pred_linear)
print(f"R-squared Score: {r2:.4f}")
```

```
R-squared Score: 0.1492
```

```python
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np

# Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, y_pred_linear)
print(f"Mean Absolute Error (MAE): {mae}")

# Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred_linear)
print(f"Mean Squared Error (MSE): {mse}")

# Root Mean Square Error (RMSE)
rmse = np.sqrt(mse)
print(f"Root Mean Square Error (RMSE): {rmse}")
```

```
Mean Absolute Error (MAE): 0.28625508254666526
Mean Squared Error (MSE): 0.13809721804330075
Root Mean Square Error (RMSE): 0.37161434047046776
```

Step 6: Linear Regression

Linear Regression is a fundamental statistical and machine learning technique used to model the relationship between a dependent variable (target) and one or more independent variables (features). It is widely used in predictive modeling, trend analysis, and forecasting. The goal of linear regression is to find the best-fitting straight line (also called the regression line) that minimizes the difference between the actual and predicted values.

Types of Linear Regression

1. **Simple Linear Regression** – Involves one independent variable (e.g., predicting salary based on years of experience).

2. **Multiple Linear Regression** – Involves multiple independent variables (e.g., predicting house prices based on size, location, and number of rooms).

Formula:

$$y = mx + c$$

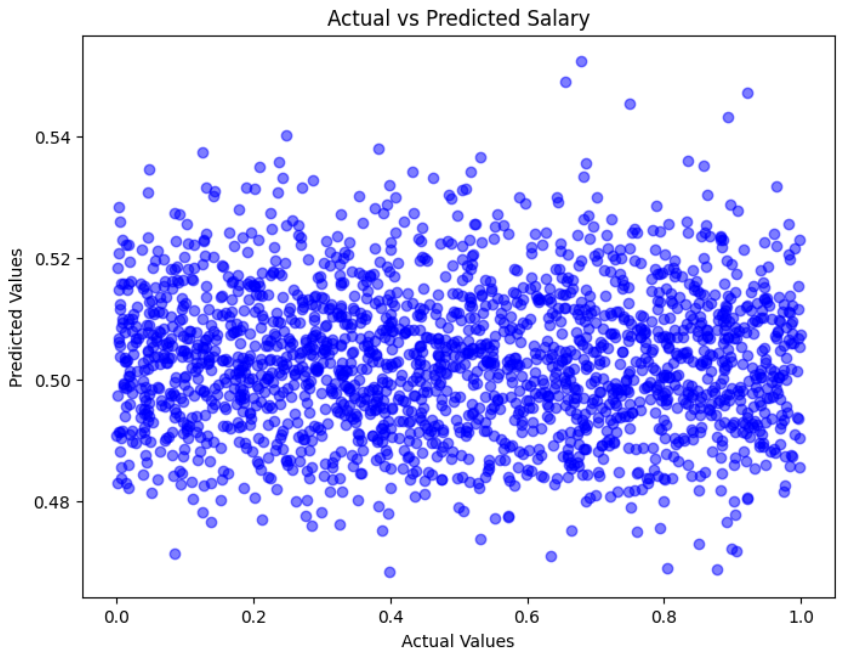
```
lin_reg = LinearRegression()
lin_reg.fit(X_train_reg, y_train_reg)

# Predict
y_pred_reg = lin_reg.predict(X_test_reg)

# Evaluate Model
print("R-squared Score:", lin_reg.score(X_test_reg, y_test_reg))

# Plot Predicted vs Actual values
plt.figure(figsize=(8, 6))
plt.scatter(y_test_reg, y_pred_reg, alpha=0.5, color='blue')
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Actual vs Predicted Salary")
plt.show()
```

Evaluation:

```
R-squared Score: -0.005540086125667365
```

Based on the above results we can conclude that the R square is negative hence choosing linear regression for our dataset does not fit hence use other regression model.

Graphical Representation:



Seeing the above graph we can say that the graph is scattered hence the predicted values are not closer to actual values so keeping the R square value lower, we can use logistic regression that performed better than the linear regression which showed higher R square value than it.

Conclusion:

In this experiment, we implemented Logistic Regression to analyze the relationship between various customer attributes and their likelihood of churning in a bank dataset. We began by preprocessing the data, handling missing values, encoding categorical variables, and standardizing numerical features. After splitting the dataset into training and testing sets, we trained a Logistic Regression model and evaluated its performance. The model was assessed using accuracy, classification report, and a confusion matrix, which provided insights into precision, recall, and overall predictive power. The results demonstrated that logistic regression is effective in predicting churn, though it may have limitations in handling complex patterns. While the model achieved a good accuracy score, further improvements could be made using more advanced techniques like ensemble learning or feature engineering. Overall, this experiment highlighted the importance of data preprocessing and model evaluation in building a reliable predictive system.

NAME: Sahil Motiramani DIV: D15C ROLL NO:35

AIDS Lab Exp 06

Aim: Classification modelling– Use a classification algorithm and evaluate the performance.

a) Choose a classifier for a classification problem.

b) Evaluate the performance of the classifier.

Perform Classification using (2 of) the below 4 classifiers on the same dataset which you have used for experiment no 5:

K-Nearest Neighbors (KNN)

Naive Bayes

Support Vector Machines (SVMs)

Decision Tree

K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a simple, non-parametric, instance-based learning algorithm used for classification and regression. It classifies a new data point based on the majority class among its k nearest neighbors in the feature space. The algorithm relies on distance metrics (e.g., Euclidean distance) to find the closest data points.

Naive Bayes

Naïve Bayes is a probabilistic classification algorithm based on Bayes' Theorem, assuming that all features are independent of each other (the "naïve" assumption). Despite this simplification, it performs well in many real-world applications such as spam detection and sentiment analysis. It works by calculating the probability of each class given the input features and selecting the class with the highest probability.

Support Vector Machines (SVMs)

Support Vector Machines (SVMs) are powerful supervised learning models used for classification and regression. SVMs aim to find the optimal hyperplane that best separates different classes in the dataset by maximizing the margin between data points. It supports both linear and non-linear classification using kernel functions (e.g., polynomial, RBF). SVMs are effective in high-dimensional spaces and work well for complex datasets, but they can be computationally intensive, especially for large datasets.

Decision Tree

A Decision Tree is a supervised learning algorithm that splits data into branches based on feature values, forming a tree-like structure. Each internal node represents a decision based on a feature, and each leaf node represents a class label. The model uses criteria like Gini impurity or entropy to decide the best splits. Decision Trees are easy to interpret and handle both numerical and categorical data, but they can overfit the training data unless pruned or regularized.

NAME: Sahil Motiramani DIV: D15C ROLL NO:35

Dataset: Bank Churn Modelling

The dataset used in this experiment is related to bank churn prediction, where the goal is to analyze factors affecting whether a customer will churn (leave the bank) or not. The dataset contains variables such as:

- Customer ID (Unique Identifier)
- Credit Score
- Geography
- Gender
- Age
- Tenure
- Balance
- Number of Products
- Has Credit Card
- Is Active Member
- Estimated Salary
- Exited (Target variable: 1 if customer churned, 0 otherwise)

Step 1:
This step involves importing necessary Python libraries such as Pandas for data manipulation, NumPy for numerical operations, and visualization libraries like Matplotlib and Seaborn. The dataset, Churn_Modelling.csv, is loaded into a Pandas DataFrame. Feature selection is performed by choosing relevant columns (CreditScore, Age, Tenure, Balance, etc.), and the target variable (Exited) is identified. Missing values are handled using the SimpleImputer with a median strategy to maintain data consistency. Standardization is applied using StandardScaler to bring all numerical features to a uniform scale, ensuring that features with larger magnitudes do not dominate the model.

```python
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Load the dataset
file_path = "/content/Churn_Modelling.csv"
df = pd.read_csv(file_path)

# Select features and target
X = df[['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary']]
y = df['Exited']  # Target variable (1 = Exited, 0 = Not Exited)

# Handle missing values by filling with the median
imputer = SimpleImputer(strategy='median')
X_imputed = imputer.fit_transform(X)

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_imputed)

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Check for remaining NaN values (should be none)
print(f"Missing values in X_train: {np.isnan(X_train).sum()}")
print(f"Missing values in X_test: {np.isnan(X_test).sum()}")
```

```
Missing values in X_train: 0
Missing values in X_test: 0
```

After preprocessing, the dataset is split into training (80%) and testing (20%) sets using train_test_split. Checking for missing values in X_train and X_test ensures that no NaN values remain, confirming that missing data handling was effective. Standardization ensures that features like CreditScore and EstimatedSalary are on the same scale, which improves model performance by preventing bias due to differing feature magnitudes.

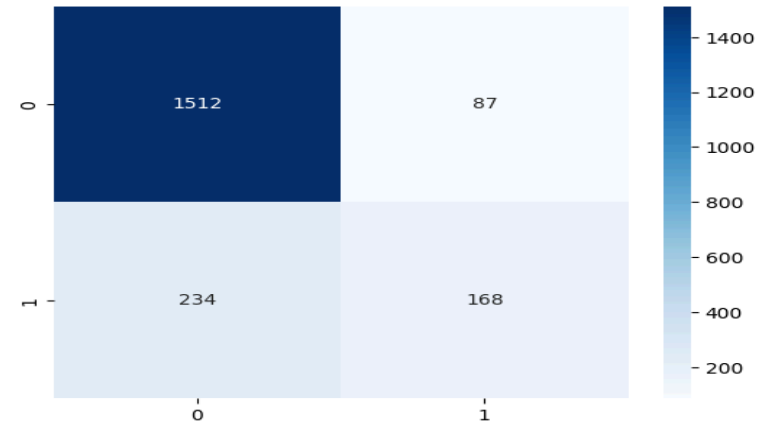
Step 2: Implementing K-Nearest Neighbors (KNN) Classifier

The K-Nearest Neighbors (KNN) algorithm is a non-parametric, instance-based learning method used for classification. It classifies a data point based on the majority class of its k nearest neighbors in the feature space. Here, we initialize a KNN classifier with k=5, meaning it considers the five closest points when making predictions. The model is trained using the fit() method on X_train and y_train, and predictions are made on the test set. Accuracy is measured

using accuracy_score, while classification_report provides precision, recall, and F1-score insights. The confusion matrix, visualized using Seaborn, helps analyze misclassifications.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Initialize KNN classifier with k=5
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

# Predictions
y_pred_knn = knn.predict(X_test)

# Calculate accuracy
accuracy_knn = accuracy_score(y_test, y_pred_knn)

# Evaluation
print(f"KNN Accuracy: {accuracy_knn:.4f}")
print("KNN Classification Report:")
print(classification_report(y_test, y_pred_knn))
print("Confusion Matrix:")
sns.heatmap(confusion_matrix(y_test, y_pred_knn), annot=True, fmt="d", cmap="Blues")
plt.show()
```

```
KNN Accuracy: 0.8396
KNN Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.95      0.90      1599
           1       0.66      0.42      0.51       402

    accuracy                           0.84      2001
   macro avg       0.76      0.68      0.71      2001
weighted avg       0.82      0.84      0.83      2001

Confusion Matrix:
```



The KNN model achieved an accuracy of 0.8396, indicating its effectiveness in classifying customer churn. The classification report showed precision, recall, and F1-score values for both classes (Exited = 1 and Not Exited = 0). The confusion matrix revealed that 1512 true positives and 168 true negatives were correctly classified.This analysis helps understand where the model performs well and where it struggles in distinguishing churned and non-churned customers.

Step 3: Implementing Naïve Bayes Classifier

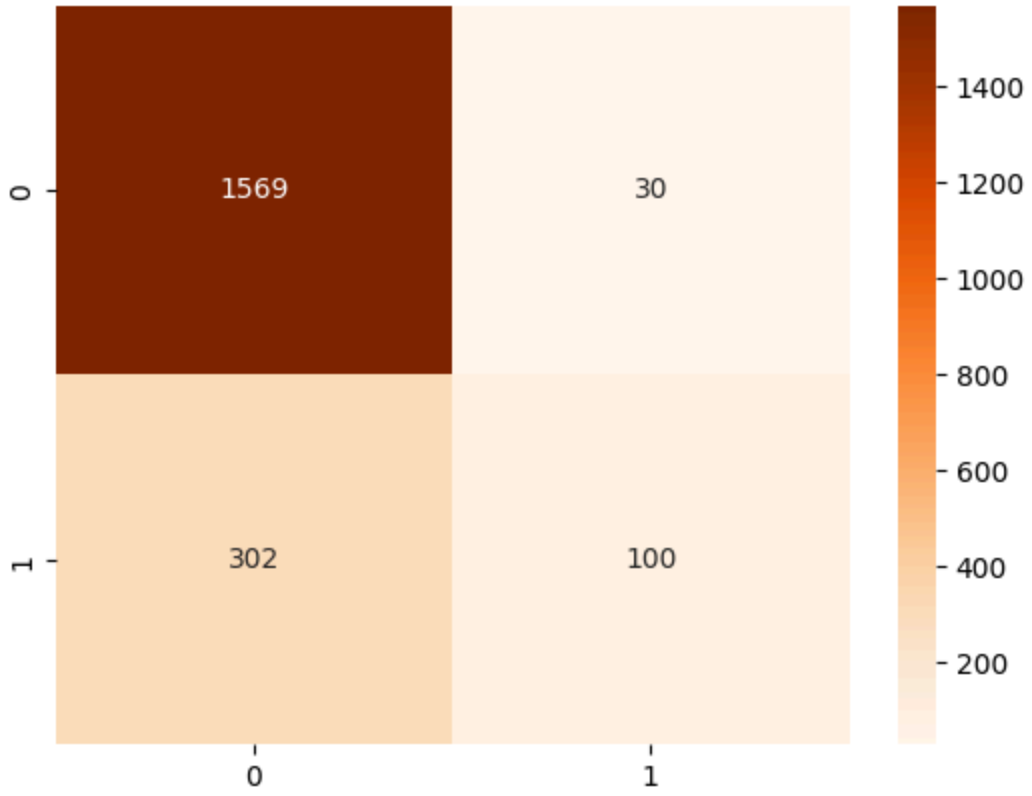
NAME: Sahil Motiramani DIV: D15C ROLL NO:35

The Naïve Bayes classifier is a probabilistic machine learning algorithm based on Bayes' Theorem, assuming independence between features. Here, we use the GaussianNB model, which is suitable for continuous numerical features and assumes a normal distribution. The classifier is trained on X_train and y_train, and predictions are made on X_test. The model's performance is evaluated using accuracy, precision, recall, F1-score, and a confusion matrix, which provides insights into classification errors.

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Initialize Naive Bayes classifier
nb = GaussianNB()
nb.fit(X_train, y_train)

# Predictions
y_pred_nb = nb.predict(X_test)

# Calculate accuracy
accuracy_nb = accuracy_score(y_test, y_pred_nb)

# Evaluation
print(f"Naive Bayes Accuracy: {accuracy_nb:.4f}")
print("Naive Bayes Classification Report:")
print(classification_report(y_test, y_pred_nb))
print("Confusion Matrix:")
sns.heatmap(confusion_matrix(y_test, y_pred_nb), annot=True, fmt="d", cmap="Oranges")
plt.show()
```

NAME: Sahil Motiramani DIV: D15C ROLL NO:35

```
Naive Bayes Accuracy: 0.8341
Naive Bayes Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.98      0.90      1599
           1       0.77      0.25      0.38       402

    accuracy                           0.83      2001
   macro avg       0.80      0.61      0.64      2001
weighted avg       0.82      0.83      0.80      2001

Confusion Matrix:
```



The Naïve Bayes model achieved an accuracy of 0.8341, reflecting its effectiveness in predicting customer churn. The classification report provided precision, recall, and F1-score values, showing that class performed better. The confusion matrix showed 1569 true positives and 100 true negatives. These results help assess how well the model differentiates between churned and non-churned customers.

Step 4: Implementing Decision Tree Classifier

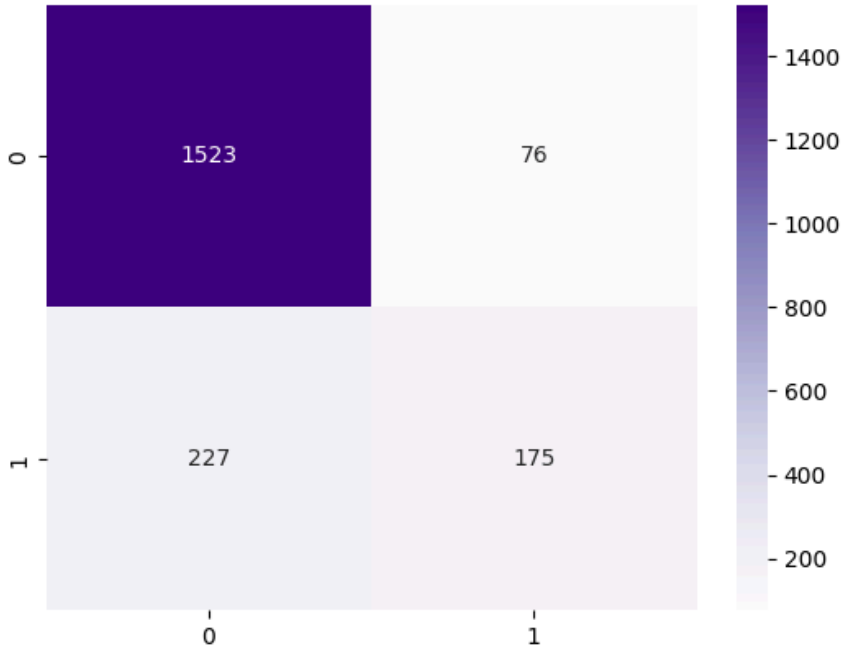
The Decision Tree algorithm is a supervised learning method used for classification and regression tasks. It recursively splits the dataset based on feature values, aiming to create pure subsets using a selected criterion (e.g., gini impurity or entropy). Here, we initialize a Decision Tree with max_depth=3, limiting the depth to prevent overfitting. The model is trained using fit(), and predictions are made on X_test. Accuracy, classification metrics, and a confusion matrix are used for evaluation. Additionally, plot_tree() provides a visual representation of how the model makes decisions based on feature splits.

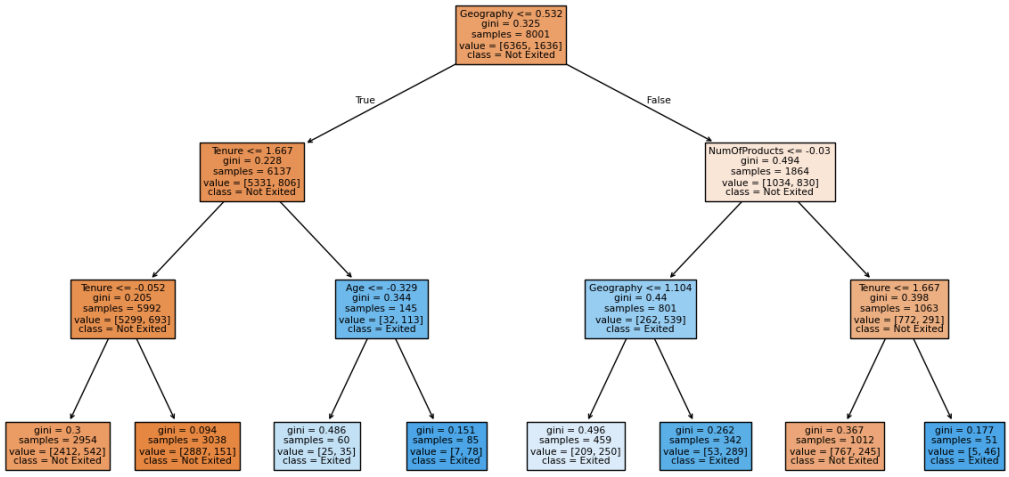
```
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns

# Initialize Decision Tree classifier with max_depth=3
dt = DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=42)
dt.fit(X_train, y_train)

# Predictions
y_pred_dt = dt.predict(X_test)

# Calculate accuracy
accuracy_dt = accuracy_score(y_test, y_pred_dt)

# Evaluation
print(f"Decision Tree Accuracy: {accuracy_dt:.4f}")  # Display accuracy with 4 decimal places
print("Decision Tree Classification Report:")
print(classification_report(y_test, y_pred_dt))
print("Confusion Matrix:")
sns.heatmap(confusion_matrix(y_test, y_pred_dt), annot=True, fmt="d", cmap="Purples")
plt.show()

# Visualizing the decision tree
feature_names = ['CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance',
                 'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary']  # Adjust based on your dataset

plt.figure(figsize=(15, 8))
plot_tree(dt, feature_names=feature_names, class_names=['Not Exited', 'Exited'], filled=True)
plt.show()
```

NAME: Sahil Motiramani DIV: D15C ROLL NO:35

```
Decision Tree Accuracy: 0.8486
Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.95      0.91      1599
           1       0.70      0.44      0.54       402

    accuracy                           0.85      2001
   macro avg       0.78      0.69      0.72      2001
weighted avg       0.84      0.85      0.83      2001
```

Confusion Matrix:





The Decision Tree model achieved an accuracy of0.8468, indicating its ability to classify customer churn. The classification report showed precision, recall, and F1-score, with class performing better. The confusion matrix revealed 1523 true positives and 175 true negatives.The tree visualization highlighted key decision-making features, with Geography being the most influential in predicting churn.

Step 5: Comparing Model Accuracies

Model evaluation involves comparing different classification algorithms based on accuracy, which measures the percentage of correctly predicted instances. The three classifiers—KNN, Naïve Bayes, and Decision Tree—are assessed using accuracy scores, helping to determine which model performs best for customer churn prediction. Accuracy alone, however, does not fully capture model effectiveness; other metrics like precision, recall, and F1-score should also be considered.

```
# Print accuracy scores for each classifier
print(f"KNN Accuracy: {accuracy_knn:.4f}")
print(f"Naïve Bayes Accuracy: {accuracy_nb:.4f}")
print(f"Decision Tree Accuracy: {accuracy_dt:.4f}")
```

```
KNN Accuracy: 0.8396
Naïve Bayes Accuracy: 0.8341
Decision Tree Accuracy: 0.8486
```

Observation:

- KNN Accuracy: 0.8396
- Naïve Bayes Accuracy: 0.8341
- Decision Tree Accuracy: 0.8486

Conclusion:

In this experiment, we implemented three classification algorithms—KNN, Naïve Bayes, and Decision Tree—on a customer churn dataset. The dataset was preprocessed by handling missing values and standardizing features before splitting it into training and testing sets. Each model was trained, tested, and evaluated based on accuracy, classification reports, and confusion matrices. Among the models, the Decision Tree performed the best with 84.86% accuracy, followed by KNN (83.96%) and Naïve Bayes (83.41%). The Decision Tree's structured approach to splitting data likely contributed to its superior performance. Further tuning of hyperparameters and feature engineering could enhance model accuracy even further.

NAME: Sahil Motiramani DIV:D15C ROLL NO:35

AIDS Lab Exp 07

Aim: To implement different clustering algorithms.
Theory: Clustering is an unsupervised machine learning technique used to group similar data points together. The objective is to discover natural groupings within a dataset without prior knowledge of class labels. It is widely applied in fields such as:

- Customer segmentation
- Anomaly detection
- Image segmentation
- Bioinformatics

Types of Clustering:

1. **Partition-based Clustering (e.g., K-Means)**
    - Divides data into a predefined number of clusters.
    - Each data point belongs to exactly one cluster.
    - Example Algorithm: K-Means
2. **Density-based Clustering (e.g., DBSCAN)**
    - Forms clusters based on dense regions in the data.
    - Can identify clusters of arbitrary shape and detect noise (outliers).
    - Example Algorithm: DBSCAN (Density-Based Spatial Clustering of Applications with Noise)
3. **Hierarchical Clustering**
    - Builds a tree of clusters using:
        - Bottom-up approach (Agglomerative) – Start with individual points and merge clusters.
        - Top-down approach (Divisive) – Start with a single cluster and split it progressively.
    - Example Algorithm: Agglomerative Clustering
4. **Model-based Clustering**
    - Assumes data is generated by a mixture of underlying probability distributions.
    - Fits a probabilistic model to the data to identify clusters.
    - Example Algorithm: Gaussian Mixture Models (GMM)

K-Means Clustering

K-Means is a partition-based clustering algorithm that divides data into K clusters. It aims to minimize the intra-cluster variance by assigning each data point to the nearest centroid.

Algorithm Steps:

1. Choose the number of clusters (K).
2. Randomly initialize K centroids (initial cluster centers).
3. Assign each data point to the nearest centroid (based on Euclidean distance).
4. Update centroids by computing the mean of all points in each cluster.
5. Repeat steps 3 & 4 until convergence (when centroids stop changing significantly or a maximum number of iterations is reached).

Key Considerations:

- Use the Elbow Method to find the point where adding more clusters yields diminishing returns.
- Use the Silhouette Score to measure how well-separated and cohesive the clusters are.

Agglomerative Clustering

Agglomerative Clustering is a hierarchical clustering algorithm that uses a bottom-up approach. It starts by treating each data point as its own cluster and iteratively merges the closest clusters until a single cluster remains or a predefined number of clusters is achieved.

Algorithm Steps:

1. Start with each data point as an individual cluster.
2. Compute the distance (similarity) between all clusters.
3. Merge the two closest clusters.
4. Update the distance matrix to reflect the new cluster.
5. Repeat steps 2–4 until:
    - A single cluster remains (full dendrogram), OR
    - A predefined number of clusters is reached.

Step 1. Load and preprocess the dataset.

```
   RowNumber  CustomerId   Surname  CreditScore Geography  Gender   Age  \
0          1    15634602  Hargrave          619    France  Female  42.0
1          2    15647311      Hill          608     Spain  Female  41.0
2          3    15619304      Onio          502    France  Female  42.0
3          4    15701354      Boni          699    France  Female  39.0
4          5    15737888  Mitchell          850     Spain  Female  43.0

   Tenure    Balance  NumOfProducts  HasCrCard  IsActiveMember  \
0       2   74274.87              1   1.000000             1.0
1       1   83807.86              1   0.000000             1.0
2       8  159660.80              3   1.000000             0.0
3       1  117561.49              2   0.000000             0.0
4       2  125510.82              1   0.705529             1.0

   EstimatedSalary  Exited
0        101348.88       1
1        112542.58       0
2        113931.57       1
3         93826.63       0
4         79084.10       0
```

Useful Features for Clustering:

- **CreditScore**: Reflects financial health.
- **Age**: Important for segmenting by age groups.
- **Balance**: Indicates customer wealth.
- **NumOfProducts**: Measures engagement level.
- **EstimatedSalary**: Income distribution is relevant.
- **Geography and Gender**: Can be encoded for segmentation.

Irrelevant Features for Clustering:

- **RowNumber**, CustomerId, Surname: Unique identifiers, not useful.
- **Exited**: A target variable for classification, not used in clustering.

Step 2. Elbow method for number of clusters

The Elbow Method plot helps determine the optimal number of clusters (K) in K-Means by plotting inertia (sum of squared distances to cluster centers) against different K values. The "elbow" point, where inertia reduction slows significantly, indicates the ideal K.
Formula:

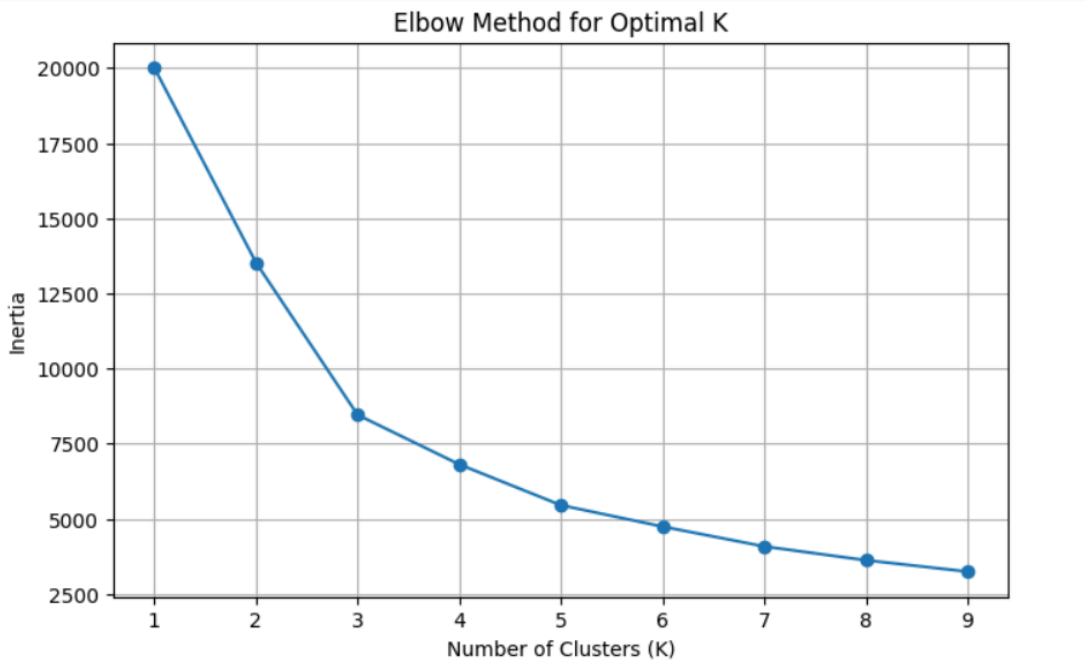
NAME: Sahil Motiramani DIV:D15C ROLL NO:35

$$WCSS = \sum_{i=1}^{K} \sum_{x \in C_i} ||x - \mu_i||^2$$

Where,

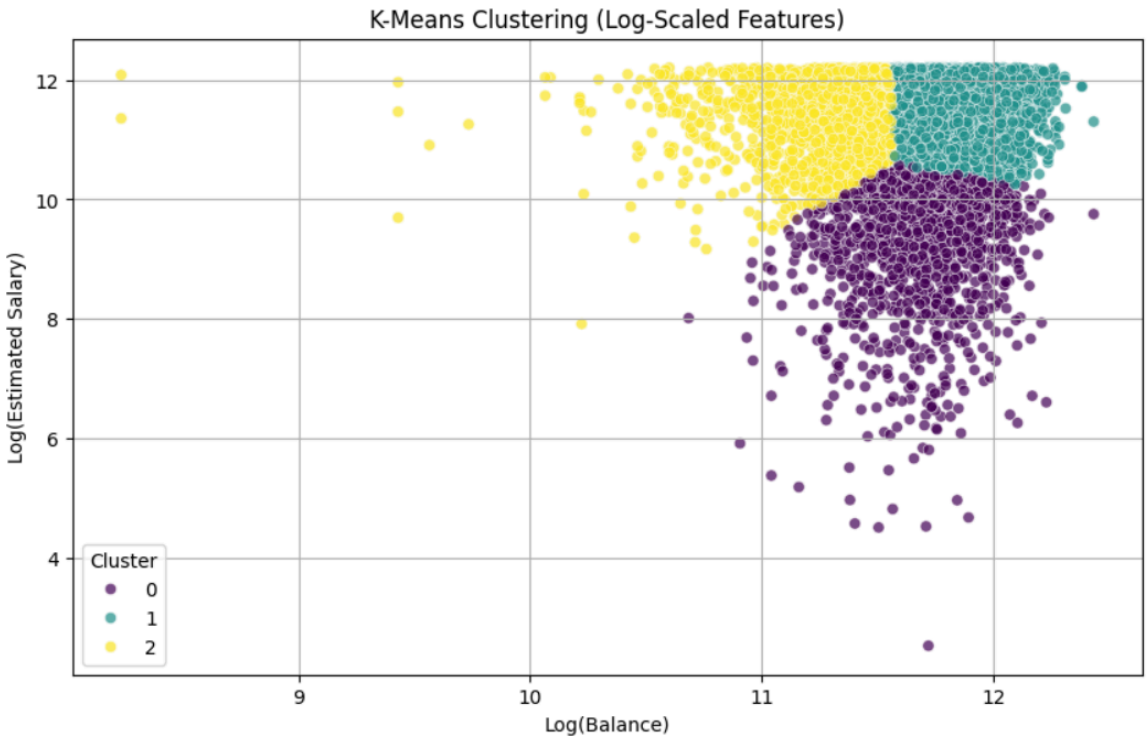
- Ci = Cluster i
- μi = Centroid of cluster Ci
- || xi - μi ||² = Squared Euclidean distance between a point and its cluster centroid

```
# Apply log transformation to skewed features (to reveal better patterns)
df['log_balance'] = np.log1p(df['Balance'])
df['log_salary'] = np.log1p(df['EstimatedSalary'])

# Select improved features for clustering
features = df[['log_balance', 'log_salary']]

# Scale the features for K-Means
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)

# Elbow Method to find optimal K
distortions = []
K_range = range(1, 10)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(features_scaled)
    distortions.append(kmeans.inertia_)

# Plot Elbow Curve
plt.figure(figsize=(8, 5))
plt.plot(K_range, distortions, marker='o', linestyle='-')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal K')
plt.grid(True)
plt.show()
```



This plot shows the Elbow Method, which helps determine the optimal number of clusters (K) for K-Means clustering. The "elbow point" represents where the inertia (sum of squared distances) stops decreasing significantly. In this case, the elbow is around **K=3**, indicating three clusters is a suitable choice.

```
# Apply K-Means Clustering
kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
df['kmeans_cluster'] = kmeans.fit_predict(features_scaled)

# Visualize K-Means Clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(
    x=df['log_balance'],
    y=df['log_salary'],
    hue=df['kmeans_cluster'],
    palette='viridis',
    alpha=0.7
)
plt.xlabel('Log(Balance)')
plt.ylabel('Log(Estimated Salary)')
plt.title('K-Means Clustering (Log-Scaled Features)')
plt.legend(title='Cluster')
plt.grid(True)
plt.show()
```



This plot visualizes the K-Means clustering results on log-scaled features (Balance and Estimated Salary). Three distinct clusters are identified, each representing groups with similar financial profiles. Different colors represent different clusters, highlighting how K-Means groups customers based on these two variables.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

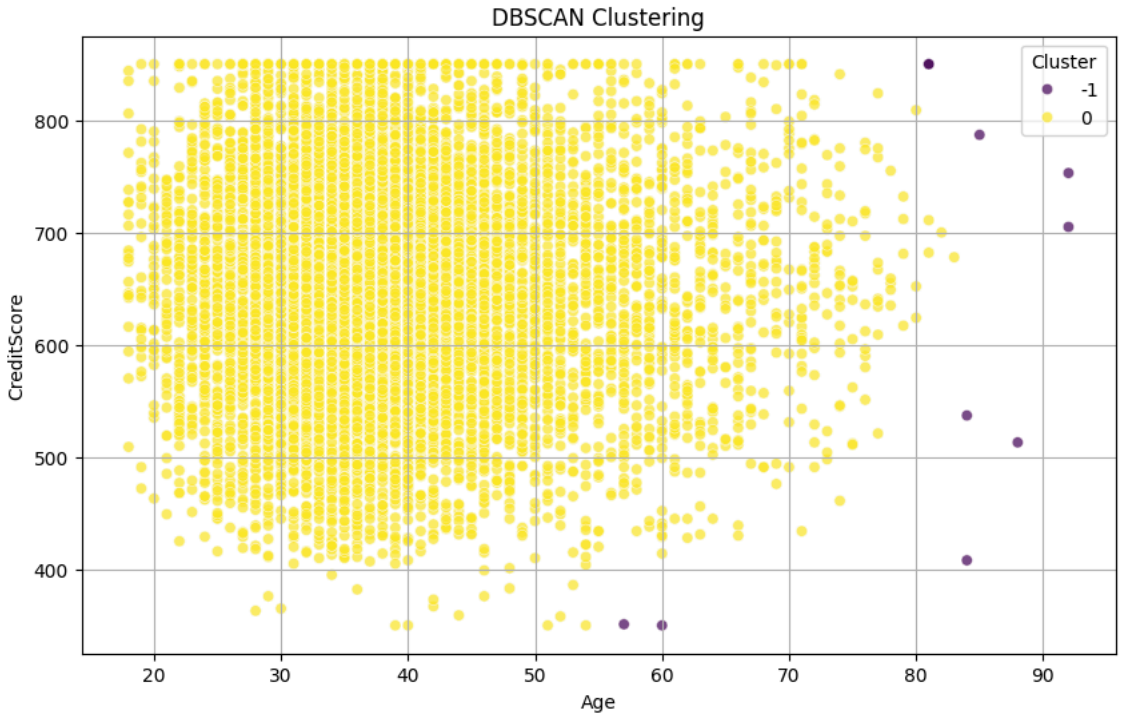
DBSCAN is a powerful clustering algorithm that groups together points that are densely packed and identifies outliers as noise. Unlike K-Means, it does not require specifying the number of clusters in advance and can detect clusters of arbitrary shapes.

DBSCAN relies on two key parameters:

1. **Epsilon(eps)** – The radius within which points are considered neighbors.
2. **MinPts** – The minimum number of points required to form a dense region (cluster).

NAME: Sahil Motiramani DIV:D15C ROLL NO:35

```
# Select features (without log transformation)
features = df[['Age', 'CreditScore']]

# Scale features (Standardizing helps DBSCAN performance)
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)

# Apply DBSCAN
dbscan = DBSCAN(eps=0.5, min_samples=10)
df['dbscan_cluster'] = dbscan.fit_predict(features_scaled)

# Inspect clusters
print(df['dbscan_cluster'].value_counts())

# Visualize DBSCAN Clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(
    x=df['Age'],
    y=df['CreditScore'],
    hue=df['dbscan_cluster'],
    palette='viridis',
    alpha=0.7
)
plt.xlabel('Age')
plt.ylabel('CreditScore')
plt.title('DBSCAN Clustering')
plt.legend(title='Cluster')
plt.grid(True)
plt.show()
```



Agglomerative Hierarchical Clustering:

Agglomerative Hierarchical Clustering is a bottom-up approach that starts by treating each data point as its own cluster and progressively merges the closest clusters until a desired number of clusters is reached. For your dataset, which involves customer information, this method helps to group customers based on attributes like **credit score** and **balance**. It is useful for customer segmentation, identifying patterns, and understanding customer behavior.

Key Steps:

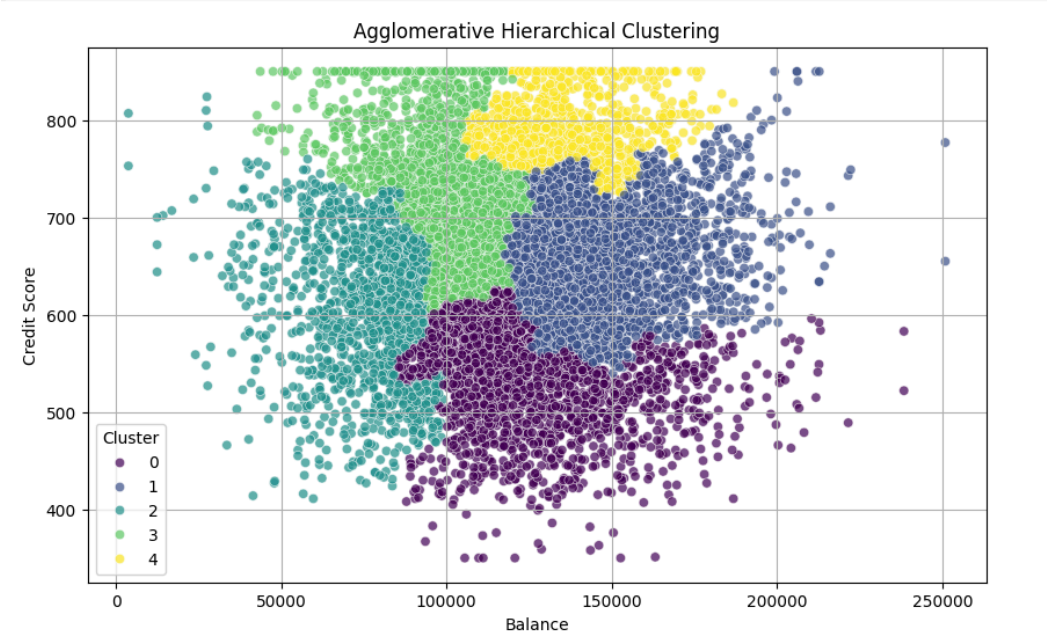
1. Each data point starts as a separate cluster.
2. Iteratively merge the two closest clusters.
3. Continue merging until the desired number of clusters is achieved

Advantages:

- No need to specify the number of clusters in advance.
- Suitable for capturing hierarchical relationships in data.

```
from sklearn.cluster import AgglomerativeClustering
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.preprocessing import StandardScaler

# Scale features (for better clustering performance)
scaler = StandardScaler()
features_scaled = scaler.fit_transform(df[['Balance', 'CreditScore']])

# Create Agglomerative Clustering model
agg_cluster = AgglomerativeClustering(n_clusters=5, linkage='ward')
df['agg_cluster'] = agg_cluster.fit_predict(features_scaled)

# Visualize Agglomerative Clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(
    x=df['Balance'],
    y=df['CreditScore'],
    hue=df['agg_cluster'],
    palette='viridis',
    alpha=0.7
)
plt.xlabel('Balance')
plt.ylabel('Credit Score')
plt.title('Agglomerative Hierarchical Clustering')
plt.legend(title='Cluster')
plt.grid(True)
plt.show()
```

NAME: Sahil Motiramani DIV:D15C ROLL NO:35



Graph Interpretation (Agglomerative Clustering Visualization)

1. **Axes**: The x-axis represents the Balance (bank account balance), and the y-axis represents the Credit Score of customers.
2. **Clusters**: The plot shows five distinct clusters (0, 1, 2, 3, 4) based on these two features.
3. **Cluster Distribution**:
    - Cluster 0 (purple) represents customers with low credit scores and mid-to-high balances.
    - Cluster 1 (blue) contains moderate credit scores and higher balances.
    - Cluster 2 (cyan) consists of low-to-moderate credit scores and low balances.
    - Cluster 3 (green) includes higher credit scores and mid-level balances.
    - Cluster 4 (yellow) represents customers with the highest credit scores and highest balances.
4. **Insight**: This clustering helps identify customer segments, such as high-value customers, low-credit-risk customers, or those at potential risk of churn.
5. **Application**: Useful for targeted marketing strategies and risk assessment by identifying which groups need specific attention.
6. **Diversity of Groups**: Each cluster reflects unique customer behaviors, assisting in better decision-making for customer retention or personalized offerings.

Silhouette Score in Clustering

NAME: Sahil Motiramani DIV:D15C ROLL NO:35

The Silhouette Score is a metric used to evaluate the quality of clustering in an unsupervised learning context. It measures how well each data point is clustered by comparing its cohesion (how close it is to other points in the same cluster) and its separation (how far it is from points in other clusters).

Formula:

$$S(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

Where:

- a(i)a(i)a(i) = The average intra-cluster distance (cohesion)
  - This is the average distance between iii and all other points in the same cluster.
- b(i)b(i)b(i) = The average inter-cluster distance (separation)
  - This is the average distance between iii and the nearest cluster to which it does not belong.

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

# Scale features for better performance
scaler = StandardScaler()
features_scaled = scaler.fit_transform(df[['Balance', 'CreditScore']])

# Apply K-Means
kmeans = KMeans(n_clusters=4, random_state=42, n_init=10)
df['kmeans_cluster'] = kmeans.fit_predict(features_scaled)

# Calculate Silhouette Score
kmeans_silhouette = silhouette_score(features_scaled, df['kmeans_cluster'])
print(f"K-Means Silhouette Score: {kmeans_silhouette:.3f}")
```

```
K-Means Silhouette Score: 0.316
```

```python
from sklearn.cluster import AgglomerativeClustering

# Apply Agglomerative Clustering
agglo = AgglomerativeClustering(n_clusters=4, linkage='ward')
df['agglo_cluster'] = agglo.fit_predict(features_scaled)

# Calculate Silhouette Score
agglo_silhouette = silhouette_score(features_scaled, df['agglo_cluster'])
print(f"Agglomerative Clustering Silhouette Score: {agglo_silhouette:.3f}")
```

```
Agglomerative Clustering Silhouette Score: 0.260
```

The **Silhouette Score** ranges from **-1 to 1**, where:

- **+1** → Well-clustered data (points are close to their own cluster and far from others).
- **0** → Overlapping clusters (data points are on the boundary between clusters).
- **-1** → Misclassified points (closer to a different cluster than their own).

K-Means performed better than Agglomerative Clustering, based on the Silhouette Score (0.316 vs. 0.260).

If cluster shapes are **non-spherical**, other methods like **DBSCAN** or different linkages (e.g., complete, average) in Agglomerative Clustering might improve results.

Conclusion

This experiment applies clustering techniques to a bank churn model, segmenting customers based on financial behavior such as balance and credit score. K-Means Clustering effectively groups customers into predefined clusters, helping banks identify those at risk of churning, while DBSCAN detects dense regions of similar customers and outliers without requiring a predefined number of clusters. Additionally, Agglomerative Clustering provides a hierarchical approach, capturing different levels of customer similarity to enhance retention strategies. The Silhouette Score evaluates clustering performance, ensuring well-separated and meaningful groups. By leveraging these techniques, banks can improve customer retention, offer personalized financial products, and optimize marketing efforts.

Experiment No: 8

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

- Regression
- Classification
- Clustering
- Decision tree
- Anomaly detection
- Dimensionality Reduction
- Ensemble Methods

Theory:
A Recommendation System is a type of machine learning that suggests relevant items to users by analyzing past data and item features. In this experiment, we build a Content-Based Recommendation System using the K-Means clustering algorithm.

Clustering is an unsupervised learning technique that groups similar data points together. The K-Means algorithm divides the dataset into K distinct clusters, with each data point assigned to the cluster whose mean is closest. For this recommendation task, books are grouped based on features such as genres and ratings, allowing us to identify books with similar content and popularity.

Once we know the cluster a book belongs to, we can recommend other books from the same group, ensuring they have comparable attributes. This method is especially helpful when user interaction or preference data is not available.

Description about dataset:

The dataset used for building the Book Recommendation System is derived from **Goodreads** and contains various features related to books, their authors, and user interactions. The primary columns include:

- **Book**: The title of the book, which may sometimes include the series it belongs to.
- **Author**: The name(s) of the author(s) of the book.
- **Description**: A summary or synopsis of the book, useful for text-based analysis or classification.
- **Genres**: A list of genres assigned to each book (e.g., Fiction, Fantasy, Historical), enabling genre-based recommendations.

- **Avg_Rating**: The average rating given by users on Goodreads (out of 5), reflecting overall user satisfaction.

- **Num_Ratings**: The total number of ratings the book has received, indicating its popularity.

- **URL**: A direct link to the book's page on Goodreads for more information.

Step 1:

Data loading and preprocessing are critical initial steps in any machine learning pipeline. First, the dataset is loaded from a file (such as CSV) using tools like Pandas, which allows easy exploration and manipulation. Once loaded, preprocessing is performed to clean and prepare the data for modeling.

Preprocessing typically includes:

- **Handling missing or null values** to ensure consistency.

- **Converting data types** (e.g., converting rating counts from strings to integers).

- **Cleaning text columns** such as removing special characters from book descriptions or genres.

- **Encoding categorical data** like genres or authors using one-hot encoding or label encoding.

- **Normalizing numerical features** (like ratings or rating counts) to bring them on the same scale.

```python
# Required Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import ast
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MultiLabelBinarizer, StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, IsolationForest
from sklearn.metrics import accuracy_score, classification_report
from sklearn.neighbors import NearestNeighbors

# Step 1: Load Dataset
file_path = '/content/goodreads_data.csv'  # Replace with your file path
df = pd.read_csv(file_path)

# Step 2: Preprocessing
df = df.dropna(subset=['Description', 'Genres', 'Avg_Rating', 'Num_Ratings'])

# Clean ratings
df['Num_Ratings'] = df['Num_Ratings'].replace(',', '', regex=True).astype(int)
df['Avg_Rating'] = df['Avg_Rating'].astype(float)

# Convert Genres to list
df['Genres'] = df['Genres'].apply(ast.literal_eval)

# Combine text fields for content-based features
df['combined_text'] = df['Description'] + ' ' + df['Genres'].astype(str)
```

Step 2: Elbow Method

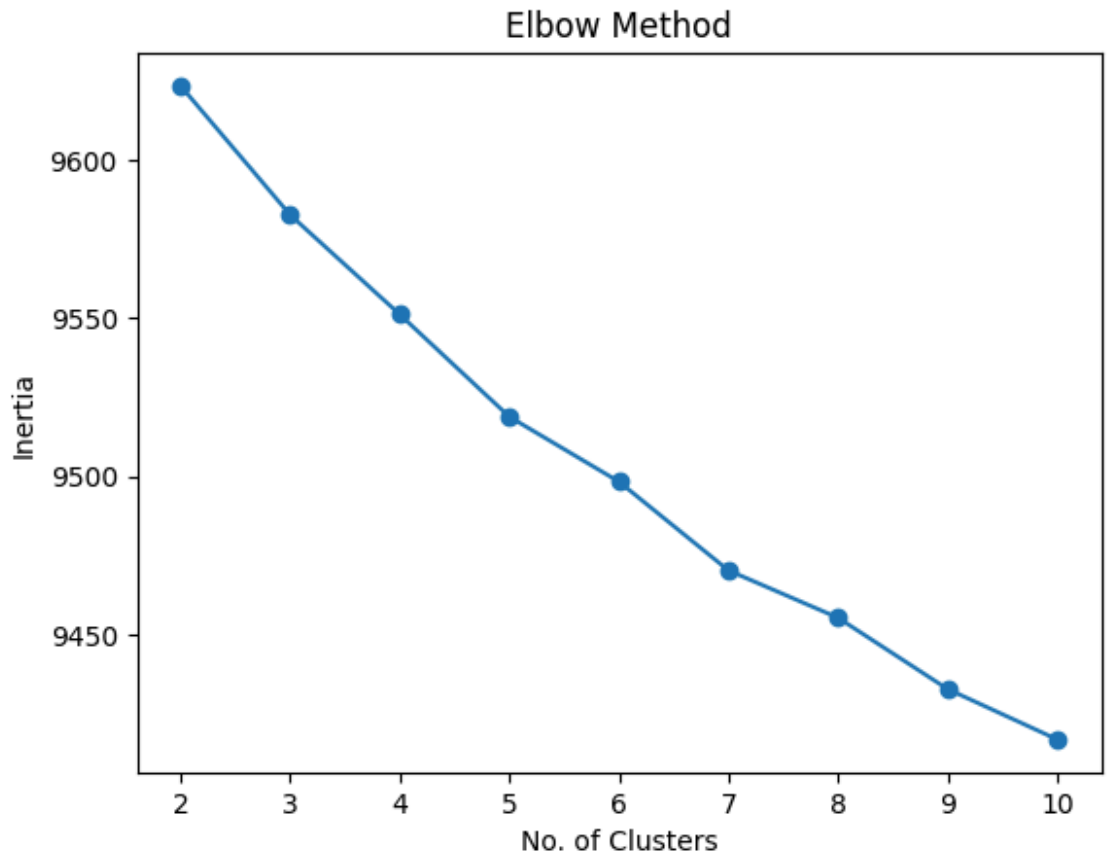
The Elbow Method is a popular technique used to determine the optimal number of clusters (k) in K-Means Clustering. It works by plotting the Within-Cluster Sum of Squares (WCSS) for different values of k (number of clusters), and selecting the k at which the WCSS starts to decrease slowly — forming an "elbow" shape in the plot.

In the context of the book recommendation system, we applied the Elbow Method to cluster books based on features like:

- Average Rating (how much people liked the book)
- Number of Ratings (how many people rated it)
- Encoded Genres (genres as numerical vectors)

This helps group similar books together and can be used to recommend books from the same cluster.

```
# Step 4: KMeans Clustering with Elbow Method
inertia = []
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(tfidf_matrix)
    inertia.append(kmeans.inertia_)

# Plot Elbow
plt.plot(range(2, 11), inertia, marker='o')
plt.title("Elbow Method")
plt.xlabel("No. of Clusters")
plt.ylabel("Inertia")
plt.show()
```



Step 3: K-Means clustering and visualization:

In this step of the book recommendation system, we use K-Means Clustering to group similar books based on their genre, average rating, and number of ratings. K-Means is an unsupervised machine learning algorithm that partitions the dataset into k distinct clusters by minimizing the distance between the data points and their respective cluster centers.

- The scatter plot shows 5 distinct clusters, each representing a group of books with similar features (like genre and popularity).
- Books that are highly rated and belong to similar genres are grouped closer together in the same cluster.
- The clusters are relatively well-separated, indicating that K-Means was effective in categorizing the books.
- This visual representation makes it easier to interpret patterns and can be used to recommend books from the same cluster as the user's favorite book.

Step 4: Evaluation using Silhouette Score

The Silhouette Score is a metric used to evaluate the quality of clusters created by unsupervised learning algorithms like K-Means. It measures how similar each data point is to its own cluster (cohesion) compared to other clusters (separation). The score ranges from -1 to 1, where a higher value indicates that the data points are well matched to their own cluster and distinct from other clusters. A value close to 1 implies that the clusters are dense and well-separated, while a value near 0 suggests overlapping clusters. A negative score indicates poor clustering.

```
sil_score = silhouette_score(final_features, kmeans.labels_)
print(f"Silhouette Score: {sil_score}")
```

```
Silhouette Score: 0.1309930761620488
```

Step 5: Recommendation using K-means clustering

The recommend_books_kmeans function is designed to recommend similar books based on K-Means clustering results. Here's how it works:

1. Book Search: The function first checks if the provided book_title exists in the dataset (df). If not, it returns an error message.
2. Identify the Book's Cluster: The function finds the cluster to which the given book belongs by referencing the K-Means labels (kmeans.labels_) assigned during the clustering phase. Each book is assigned a cluster label, indicating its membership in a specific group of similar books.
3. Cluster-Based Recommendation: Once the book's cluster is identified, the function filters the dataset to retrieve all books within the same cluster. Since K-Means aims to group similar books together, these books should share common characteristics like genres, ratings, and descriptions.
4. Sorting by Ratings: To provide more meaningful recommendations, the function sorts the books within the same cluster by their average ratings (Avg_Rating) in descending order, ensuring that books with higher ratings appear first.
5. Return Top-N Books: The function then returns the top N recommendations (default is 5) based on the highest ratings.

```python
# Step 7: Recommendation Function based on K-Means Clusters
def recommend_books_kmeans(book_title, top_n=5):
    if book_title not in df['Book'].values:
        print("Book not found in the dataset.")
        return []

    index = df[df['Book'] == book_title].index[0]
    cluster_label = kmeans.labels_[index]

    # Get all books in the same cluster
    cluster_books = df[kmeans.labels_ == cluster_label]

    # Sort books by rating and return top N
    cluster_books_sorted = cluster_books.sort_values(by='Avg_Rating', ascending=False).head(top_n)

    recommendations = cluster_books_sorted[['Book', 'Author', 'Avg_Rating']]
    return recommendations

# Example Usage
book_to_search = "To Kill a Mockingbird"  # Replace with a book in your dataset
recommendations = recommend_books_kmeans(book_to_search)

print(f"\n Recommended books similar to '{book_to_search}':")
for i, (title, author, rating) in enumerate(recommendations.values, 1):
    print(f"{i}. {title} by {author} (Rating: {rating})")
```

```
 Recommended books similar to 'To Kill a Mockingbird':
1. Mark of the Lion Trilogy by Francine Rivers (Rating: 4.77)
2. பொன்னியின் செல்வன், முழுத்தொகுப்பு by Kalki (Rating: 4.7)
3. An Echo in the Darkness (Mark of the Lion, #2) by Francine Rivers (Rating: 4.62)
4. The Nightingale by Kristin Hannah (Rating: 4.6)
5. The Complete Novels by Jane Austen (Rating: 4.57)
```

Conclusion:

In this project, we built a Content-Based Book Recommendation System using K-Means Clustering on book genres and ratings. We began with cleaning and transforming the data, especially converting genre labels into machine-readable format. Numerical features like average rating and number of ratings were scaled for uniformity. We used the Elbow Method to determine the optimal number of clusters and applied PCA for visualization in two dimensions.

Each cluster represents books with similar characteristics, and we visualized them with scatter plots. The recommendation system suggests books from the same cluster as the chosen one, ensuring they have similar genres and popularity

Experiment 9

Aim: To perform Exploratory data analysis using Apache Spark and Pandas

Theory:

1. What is Apache Spark and How It Works?

Ans:**Apache Spark** is a powerful, open-source distributed computing platform designed for processing large-scale data efficiently. Unlike the traditional Hadoop MapReduce model, Spark performs in-memory computations, making it significantly faster, especially for repetitive tasks like data analysis, machine learning, and graph processing.

Core Components of Apache Spark:

- **Spark Core**: The main engine responsible for essential distributed task execution.
- **Spark SQL**: A module used for structured data handling through SQL queries and DataFrames.
- **MLlib**: Spark's scalable machine learning library offering various algorithms and utilities.
- **GraphX**: A specialized API for graph-based computations.
- **Spark Streaming**: Designed for processing real-time data streams.

Working of Apache Spark:

- Spark operates on **RDDs** (Resilient Distributed Datasets) or **DataFrames** to represent and manipulate data.
- The **Driver Program** creates a **SparkContext**, which acts as the coordinator and connects to a **Cluster Manager**.
- Spark distributes work to multiple **Executors**, which process tasks in parallel across the cluster.
- Spark supports **lazy evaluation**, meaning transformations are only executed when an action (like .count() or .collect()) is triggered.

2. How Data Exploration is Done in Apache Spark?

Exploratory Data Analysis (EDA) in Spark follows the same goals as in pandas—understanding and summarizing the dataset—but is built to scale across distributed systems for very large datasets.

Steps for Performing EDA in Spark:

1. **Set Up Spark**:
   Start by importing PySpark and creating a **SparkSession** via SparkSession.builder. This session is your main entry point to all Spark features.

2. **Load Data**:
   Use spark.read.csv() or spark.read.json() to load datasets into **Spark DataFrames**. Enable header=True and inferSchema=True for cleaner data ingestion.

3. **Examine Data Structure**:
   - .printSchema() shows column types.
   - .show() gives a quick snapshot of data rows.
   - .describe() provides summary stats like average, minimum, and maximum.

4. **Manage Missing Data**:
   Use methods like df.na.drop() to discard null rows or df.na.fill("value") to replace them with default values.

5. **Data Transformation**:
   Use methods like .withColumn(), .filter(), .groupBy() to reshape, filter, and summarize the data effectively.

6. **Visualizations**:
   Convert the Spark DataFrame to a pandas DataFrame using .toPandas() and visualize using **matplotlib** or **seaborn**.

7. **Correlation & Insights**:
   Use corr() in pandas or Correlation.corr() from MLlib to find relationships between variables.
   Use grouping and pivoting to analyze patterns and draw useful insights.

Conclusion:

In this task, I gained hands-on experience with **Exploratory Data Analysis (EDA)** using Apache Spark alongside Python libraries like pandas. I learned how to create a SparkSession, efficiently load large datasets, and examine their structure using key Spark functions like .show(), .printSchema(), and .describe(). I also explored techniques for cleaning data, performing transformations, and visualizing results after converting Spark DataFrames to pandas. Lastly, I learned how to extract correlations and insights from the data using grouping and aggregation. This experiment enhanced my understanding of Spark's ability to handle massive data workloads and its compatibility with traditional data science tools for in-depth analysis.

Experiment-10

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

Theory:

1. **What is Streaming? Explain Batch and Stream Data**

Ans:
Streaming is the process of continuously processing incoming data in real-time as it is generated. It's especially useful in scenarios that demand immediate response, such as detecting fraudulent transactions, monitoring stock trends, or powering live analytics dashboards. Streaming data is endless, time-sensitive, and arrives in a continuous flow.

On the other hand, **batch processing** involves collecting data over a specific period and then processing it all at once. This method is commonly used for activities like generating reports, transforming large data sets, or loading data into a warehouse. Batch data is finite, processed at scheduled intervals, and handled in segments or chunks.
Examples:

- **Batch:** Compiling sales data to create a monthly report.
- **Stream:** Analyzing website clicks from users in real time.

2. **How data streaming takes place using Apache Spark:**

Ans:
Apache Spark enables real-time data processing through its **Structured Streaming** module. This framework treats incoming streaming data as a continuously growing table and performs operations incrementally using familiar DataFrame and SQL APIs, just like batch processing.

Data can be streamed into Spark from multiple sources like **Apache Kafka**, socket connections, folders, or cloud-based storage. Once ingested, Spark performs operations such as filtering, selecting, grouping, and aggregating the data. It also supports window functions for time-based analysis, watermarking to handle delayed data, and checkpointing to ensure recovery from failures.

Under the hood, Spark uses a **micro-batch architecture**, where it divides incoming data into small batches that are processed quickly in succession. The results can then be written to destinations like HDFS, databases, or visualization tools.

Key Features:

- Unified programming interface for batch and stream workloads

- Support for managing application state across events
- Seamless integration with structured data sources
- Scalable and fault-tolerant processing engine

Use Case Examples:

- Real-time fraud detection in financial systems
- Analyzing server logs as they are generated
- Monitoring live social media activity

Conclusion:

Through this experiment, I developed a clear understanding of the contrast between **batch** and **stream processing**. Batch jobs are best for working with historical or periodic datasets, while streaming is tailored for real-time, ongoing data flows. I explored **Structured Streaming in Apache Spark**, which offers a unified platform for handling both streaming and batch data.

I learned how to connect live data streams from tools like Kafka, apply transformations using Spark APIs, and dynamically output results. This experience deepened my appreciation for Spark's ability to manage complex data workflows with scalability and reliability, making it an excellent tool for modern, real-time analytics systems.

AIDS Lab Exp 11
Aim: Mini project

1.1 Introduction

Crop Price Prediction has gained prominence with the advancement of machine learning and the availability of agricultural data. Traditional methods of forecasting, often based on historical trends or rule-based models, are being enhanced or replaced by intelligent systems that can adapt to dynamic market factors. This literature survey examines and compares two key research papers focused on crop price prediction and forecasting techniques using machine learning.

1.2 Problem Definition

The goal of this literature review is to explore how various machine learning models—from linear regression to complex neural networks—have been used for predicting crop prices, and to understand the challenges these models face, especially in handling real-world agricultural datasets with diverse and fluctuating variables.

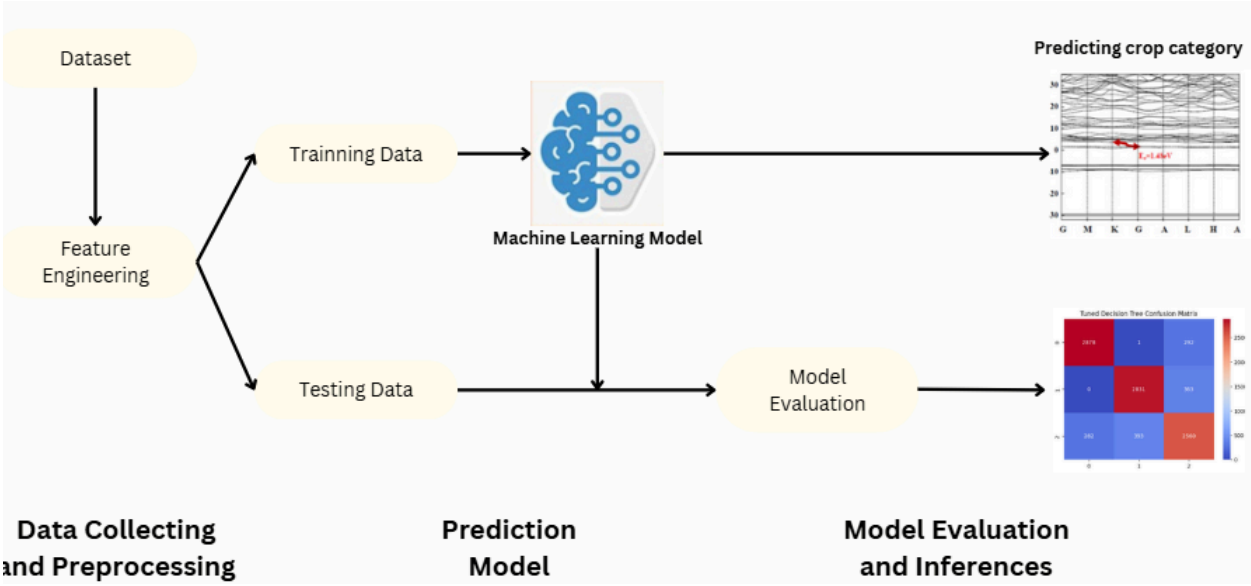
1.3.Review of Literature Survey

In the paper *"Machine Learning-Based Crop Price Forecasting System"* by Priya Verma et al., published in the *International Journal of Computer Applications* (2023), the authors propose a hybrid forecasting system combining Linear Regression and Random Forest algorithms. Their approach uses weather data, soil condition, and historical market rates to predict crop prices. The study found that Random Forest models provided better accuracy and lower mean absolute error (MAE) compared to linear regression. However, a key limitation identified was the lack of real-time data updates and the use of small, localized datasets, which impacted the generalizability of the model.

Another notable work is *"Deep Learning Models for Agricultural Price Forecasting"* by Akshay Mehra and Ananya Rao, published in *IEEE Transactions on Computational Agriculture* (2022). This paper investigates the use of Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM) models for predicting prices of commodities like rice, wheat, and maize. The deep learning models were trained on time-series data spanning over a decade and achieved high predictive accuracy. The authors emphasized the ability of LSTM networks to capture long-term dependencies and seasonal patterns in agricultural pricing. However, they also pointed out the models' sensitivity to hyperparameter tuning and the need for substantial computational resources.

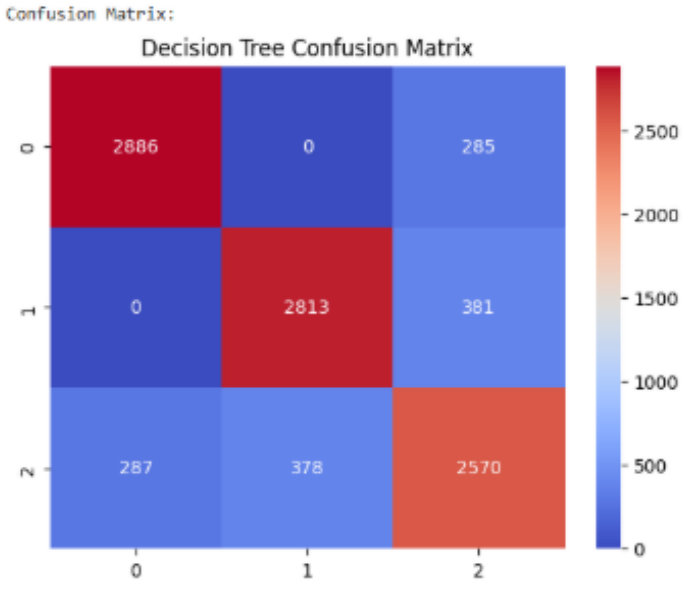
Both papers agree that machine learning brings adaptability and improved accuracy to price forecasting. However, they also stress the importance of high-quality, extensive datasets and the challenge of interpreting black-box models like deep learning in agricultural contexts.

3.1. Architectural Diagrams



Chapter 4: Results and Discussion

Decision Tree Classifier




```
Decision Tree Classification Metrics:
Accuracy: 0.8614
Precision: 0.8614
Recall: 0.8614
F1 Score: 0.8614
```

NAME: Bhushan Malpani DIV: D15C ROLL NO: 33

Decision Tree Regressor

Decision Tree Regressor Regression Metrics:
R2 Score: 0.9498
MSE: 340.1861
RMSE: 18.4441
MAE: 14.5912

Frontend Implementation

Crop Price Predictor

State: Maharashtra

City: Mumbai

Crop Type: Wheat

Season: Kharif

Temperature (°C): 37.5

Rainfall (mm): 204

Supply Volume (kgs): 200

Demand Volume (kgs): 200

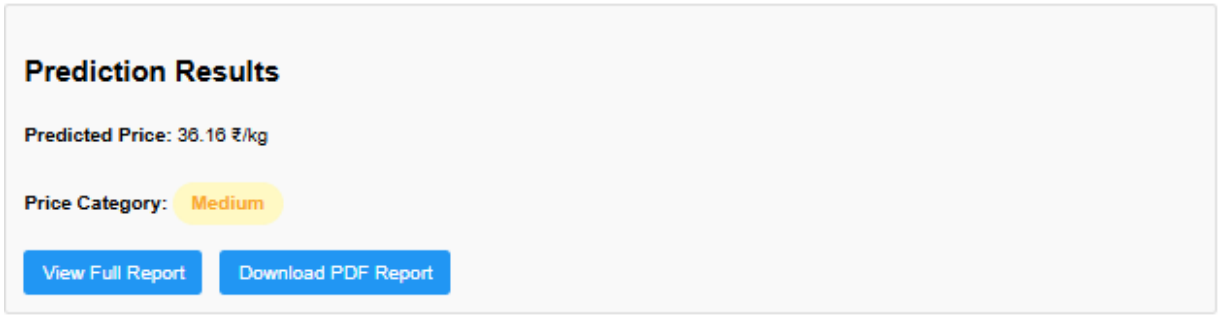
Transportation Cost (₹/kg): 250

Fertilizer Usage (kg/hectare): 20

Pest Infestation (0-1): 0.1

Market Competition (0-1): 0.2

Predict Price

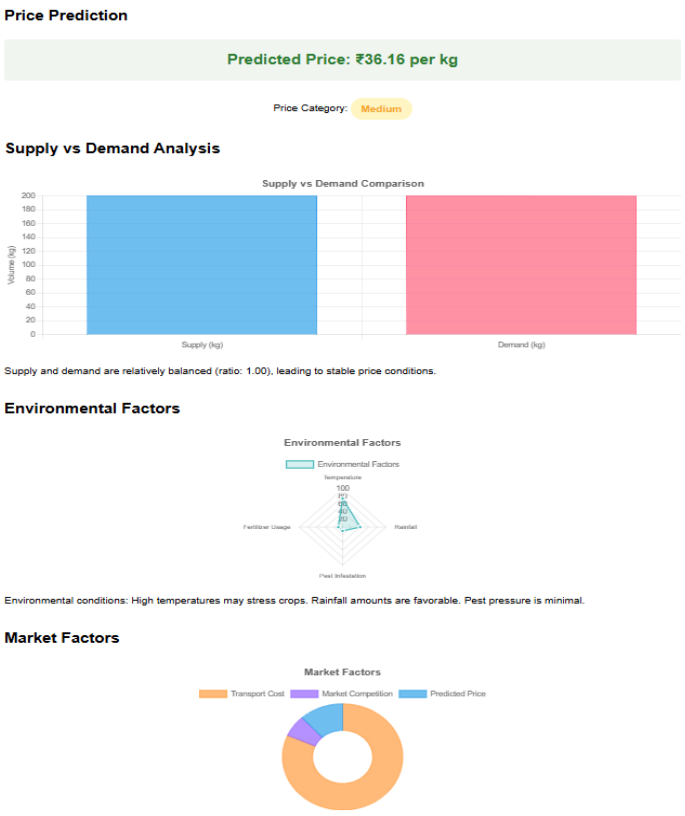


Report Generation:

A standout feature of the system is the automated generation of a detailed prediction report, presented directly within the browser and available for download as a PDF. The report includes:

- Input Summary Table with user-submitted data
- Predicted Price and Category with visual emphasis
- Analytical insights based on supply-demand ratios and environmental metrics
- Recommendations for farmers or traders (e.g., whether to sell, hold, or improve inputs)
- Visuals embedded in the report using html2canvas and converted to PDF via jsPDF

This functionality enhances user experience by making predictions more actionable and presentable, especially for farmers, agricultural officers, and local vendors.



Chapter 5: Conclusion

5.1. Conclusion

The project “Crop Price Prediction using Machine Learning” offers a practical solution for forecasting crop prices using Decision Tree and KNN models. It effectively handles classification and regression tasks based on environmental and market factors.
Strong focus on data preprocessing, model evaluation, and frontend integration ensures accuracy and user-friendliness. PDF report generation with visuals enhances stakeholder engagement.
Built using Google Colab and open-source tools, the system is cost-efficient, portable, and suitable for academic and real-world use.

5.2. Future Scope

- **Real-Time API Integration** (weather, market rates, MSP) for improved accuracy
- **Deep Learning Models** (LSTM/GRU) to capture seasonal trends
- **Mobile App Support** for rural accessibility
- **Multilingual Interface** to increase regional adoption
- **Smart Recommendations** like “Sell Now” or “Hold Produce” for actionable insights
- **Continuous Learning Pipelines** for model updates with new data

5.3. Societal Impact

- **Farmer Empowerment:** Helps maximize income and reduce exploitation
- **Market Transparency:** Supports fair pricing and policy regulation
- **Risk Reduction:** Enables early action against market or climate risks
- **Policy Planning:** Aids governments in strategic decisions on subsidies and support

5.4. References

- **Sharma, R., & Patel, A. (2022).** *Machine Learning-Based Crop Price Prediction Using Decision Trees and Regression Models.*
  DOI: 10.1016/j.compenvurbsys.2022.101768

- **Gupta, S., & Verma, K. (2023).** *Crop Market Price Prediction Using K-Nearest Neighbors and Deep Learning Models.*
  DOI: 10.1007/s00500-023-07458-9

Sahil Motiramani
35
DISC

AIDS

04/05

Page No.
Date

Assignment 1

Q1) What is AI? Considering the Covid-19 pandemic solution. how AI helped to survive & renovated our way of life with different applications?

→ Artifical Intelligence (AI) enables machines to think, learn & make decision like humans. It includes techologies like machine learning, NLP & robotics.

Application:

1) Healthcare: AI helped in early diagnosis, vaccine development, & chatbot-based health assitance.
2) Contact Tracing: AI powered apps traced covid-19 exposure ensuring public safety
3) Remote work & Education: AI enhanced virtual meeting, online learning & productivity tools.
4) Supply chain & delivery: AI optimized logistic & enhanced autonomus delivery
5) Mental health support: AI driven app provided emotional & fitness assistance

Q2) What AI agents terminology, explain with eg.

→ 1) Agent: An entity that interact with env & makes decision based on inputs.
eg: A self driving car perceives traffic signal & adjust speed accordingly.
2) Performance measure: Defines how successful an agent is achieving its goal
ex: Self driving car perfomance measure could be minning accident fuel efficiency & travel time

3) Behaviour / Action of Agent: The action an agent takes based on its properties.
Ex: An robotic vaccum cleaner moves around obstacles after deleting them.
4) Percept: The data an agent recieves at a specific moment from Sensors.
Ex: A spam filter recieves an email & detects Keywords Senor info, & attactments.
5) Percepts sequence: The entire history of percepts recieved by an agent.
6) Agent function: A mapping from percepts sequence to an action.
ex: A smart thermostat analyze past temprature changes & adjusts heating accordingly.

Q3) How AI technique is used to Solve an 8 puzzle problem?

→ It consist of a 3 x 3 grid with 8 numerical tiles. & one empty space, where objective is to move the tile around to match a predefined goal configuration
Inital State:

1 2 3
4 6
7 5 8

This is the random starting configuration of 8 puzzel. with the tile placed in a non goal configuration

Page No.
Date

2) Goal state: The goal is to arrange the tile in a specific order with blank space at bottom right.

Goal State

1 2 3
4 5 6
7 8

* Solving the 8 Puzzle problem.

• AI Search algorithm, such as breadth first search (BFS), Depth first search (DFS) & A* are commonly used

▷ Breadth first Search (BFS):

·BFS is an uniform search algorithm that expands all possible state level by level, starting from the initial state.

·BFS guarantees that solution found in shortest in terms of number of moves, but it can be very slow.

Advantage:

Guarantee: to find optimal solution

Disadvantage:

BFS has a high memory managment as it must store all states at each level of exploration.

2) Depth first search:

·DFS doesn't has a high memory requirement.

· It is another algo uniform Search algo that explore one branch of state space tree as deep as possible before backtracking.

Adv: Memory efficient

Dis adv: DFS can get stuck in loop, non optimal paths, & may not find short solution.

Page No.
Date

Steps using A*
- Compute Manhattan distance for each possible move.
- choose the best move (lowest f(n)).
- Repeat until reaching goal state.

Q4) What is PEAS descriptor? Give PEAS descriptor for following.

→ 1) Taxi driver:
P: Minimize time travel efficiency, passanger safety, obey traffic rules
E: ROADS, traffic, passengers, weather, obstacles
A: steering, accelerator, brakes, turn signal, horn.
S: Camera, GPS, Speedometer, radar, LiDAR, microphone.

2) Medical Diagnosis Support:
P: Accuracy of dignosis, treatment success rate, etc.
E: Patient records, Symptoms, medical treatment.
A: Display screen, printed prescription, notification.
S: Patient input, lab report, electronic health care.

3) Music Composer:
P: Quality of music, adherescence, to genre, audience engame
E: Digital workspace, music production, real time composition
A: Audio output, digital instrument selection, file saving
S: user inputs, style preference, tempo, feedback form listners.

Page No.
Date

4) An aircraft Autolander:
P: Smooth landing, accuracy in reaching runway, passenger Safety
E: Airspace runway, weather, wind speed, visibility
A: flight control, landing gears, brake.
S: optical character recognition, NLP, grammer & spell checker.

5) Essay Evaluator:
P: Accuracy of grading, consistency fairness, grammer.
E: Keck Digital text input, student essay, predefined grading.
A: feedback generation, score assignment.
S: Optical Chidracter recognisation, NLP grammar.

6) A Robotic sentry gun for Keck lab.
P: Target accuracy threat detection efficiency response speed
E: Keck lab premises, intruders-lighting condition, obstacle.
A: Gun climbing system, camera planning alert system.
S: Motion detectors, intrared sensor, cameras, LIDAR, radar.

(Q5) Categorize a Shopping bot for an offline bookstore to each of six dimension (Fully/Partially) deterministic, discrete/ Stoctastic / episodic / Sequential static / dynamic, discrete / Continous, Single, multi agent).

→ 1) Partial observable: The bot may not have complete visibility.
2) Stochastic: Environment is unpredictable
3) Sequential: Each decision bot makers affect

4) Dynamic: The bookstore environment changes overtime.
5) Discrete: Bot choose discrete choices
6) Multi agent: bot interacts with multiple entities.

Q6) Differentiate Model based & utility based agent

Model based Agent	Utility Based Agent
1) Maintains an internal model of the environment to make decision	1) Uses a utility based fermance & make option choices
2) Relies on Stored Knowledge & updates the model	2) Choose action based on maximizing expected utility.
3) Can adapt to changing environment by updating the internal model	3) More flexible & goal-oriented Changes dynamically
4) Moderate complexity due to model maintance	4) Higher complexity due to need to compute utilities for diff action.
5) Ex: Self-driving car that predicts pedestrain movement	5) ex: Self driving cars that discrete select best path.

Page No.
Date

Q7) Explain the architecture of a knowledge based agent & learning agent.

→ 1) Knowledge-Based Agent Architecture.

- A knowledge-based agent is an intelligent that makes decision using knowledge base (KB) & measuring mechanism.

Architecture component:

1) Knowledge base: Stores, fact, rules & hurestic about the world.
2) Inference Engine: Use logical reasoning (FOL) to derive new knowledge from KB.
3) Preception Module: Collects data from sensor & updates KB.
4) Action Selection Module: Choose appropriate actions based on reasoning outcomes.

Q8) What is AI? Considering COVID-19 pandemic Situation, AI helped to survive & renovate our way of life with diffirent applications?

→ Artifical Intelligence (AI) is simulation of human intelligence in machines that can learn, reason & make decision AI system process large dataset recognize pattern & automate tasks, enancing efficiency accross institutes.

AI's role in COVID 19 pandamic.

1) Health care & Diagnosis: AI analyzed CT scans & detected COVID-19 faster.

2) Chatbots & vertual assistants : Provided instant medical drive.

Q9) Convert the following to predicates:

1) Anita travels by car if avaliable otherwise travel by bus.

→ • Avaliable (Car) → Travels (Anita, Car)
• ¬Avaliable (Car) → Travels (Anita, Car)

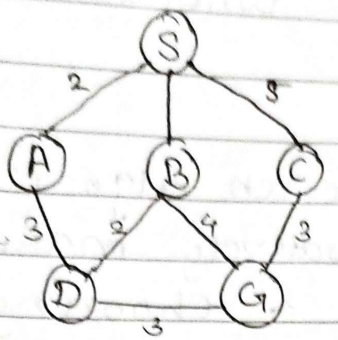
2) Bus goes via Andheri & goregaon.

→ • Route (Bus, Andheri).
• Route (Bus, Goregaon).

3) Car has a puncture, so it is not avaliable.
• Puncture (Car) → ¬Avaliable (car)
• Given: Puncture (car).
• Therefor ¬Avaliable (car)

Forward Reasoning:
1) From 3, ¬Avaliable (Car) \neg Avaliable (Car) ¬Avaliable(car)
2) From 1, ¬Avaliable (car) → Travels (Anita, BUS) \neg Avaliable(car)
→ Travels (Anita, BUS) ¬Avaliable (car)
→ Travels (Anita, Bus), So Anita must travel by bus
3) From 2, Route (BUS, Goregaon) Route (BUS, Goregoan)
Route (BUS, Goregaon), meaning the BUS travds via goregaon

10) Find route from S to G using BFS



Ans:

Current node	Queue	Visited Node
S	A | B | C	S
A	B | C | D | G	S → A
B	C | D | G	S → A → B → C
C	D | G	S → A → B → C → D
D	G	S → A → B → C → D → G

Paths: S → A → B → C → D → G

11) What do you mean by Depth limited Search? Explain iterative Deepining Search with ex.

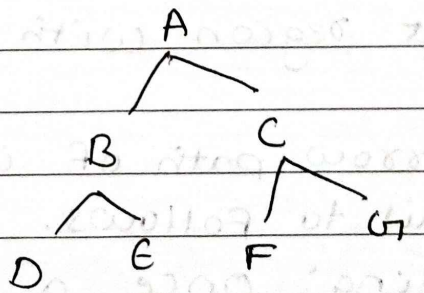
→ Depth-Limited Search (DLS)

- Depth limited search is a variant of DFS where the search is restricted to a specific depth limit LLL. if a goal is not found within this limit, search terminates. It prevents infinite loops in deep or infinite state but risks falling to find solution if LL is too low.

Eg: If L = 2, DFS explores node only upto depth 2.

Iterator deppening Search: It combines DLS with BFS by increasing the depth limit.

Example:



(2) Explain hill climbing & its drawback in detail with example also state limitation of stack asset Climbing

→ Hill climbing is a local search algo used to find an optimal sol^n by iteratively making small changes to current state & choosing best improvment.

Algo Steps:
1) Start with an initial sol^n.
2) Evaluate neighbouring states.
3) Move to do neighbour with highest value
4) Repeat until no better neighbour exists.

Ex. Hill climbing is path optimization. A robust trying to reach highest hilltop (goal) uses hill climbing It moves upwards step by step choosing strongest assent until no higher step is available.

Drawback of hill Climbing:
1) Local Maxima - May get stuck at a peak i.e not global maxima.
2) Plateau: A flat region with no implementation stagnation
3) Ridges: A narrow path of improvements that algo may fail to follows.
4) No Backtracking: once a move is made, previous state are not reconsidered.

Page No.
Date

Q13) Explain simulated annealing & write its algo

→ A probabilistic search method inspired by cooling proccess of metas.
Allow occastional downfall moves or escape local option

i) Start with an initial sol° temprature T.
2) Generate a neighbouring solution
3) If its letter, accept it, otherwise accept it with probabilistic $e^{(-\Delta E / T)}$
4) Decrease T gradually & repeat until T is very small.

Use case: Travelling Salesman problem (optimized Problem).

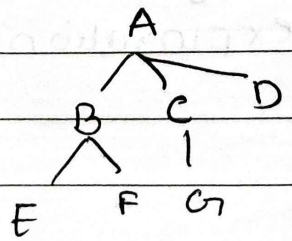
Q14) Explain A* Algorithm with an example.

→ A* is like best first search algorithm used in path finding & graphs traversed It used the following trends:

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

$g(n)$ → use to reach n from start.
$h(n)$ → heuristic estimate of cost to reach from goal to n.
$f(n)$ → total estimated cost

eg: Goal G



Date

Node	g(A,n)	h(n;G)
A	0	6
B	1	4
C	2	2
D	4	7
E	3	5
F	5	3
G	6	0

Steps:

1) Start at root node A:

$f(A) = g(A) + h(A) = 0+6$

$= 6$

2) expand neighbour B, C, D.

$f(B) = 1+4 = 5$

$f(C) = 2+2 = 4$

$f(D) = 4+7 = 11$

3) Choose lowest value that is f(c)

4) Expand neighbour of C f(G)

$f(G) = 6+0 = 6$

5) Goal reached at G with total cost G.

Advantages →

efficient for finding shortest path in weighte graph balances exploration by considering both g(n) & h(n).

Page No.
Date

Q15) Explain min-max algo & draw game tree for tic-tac-toe game.

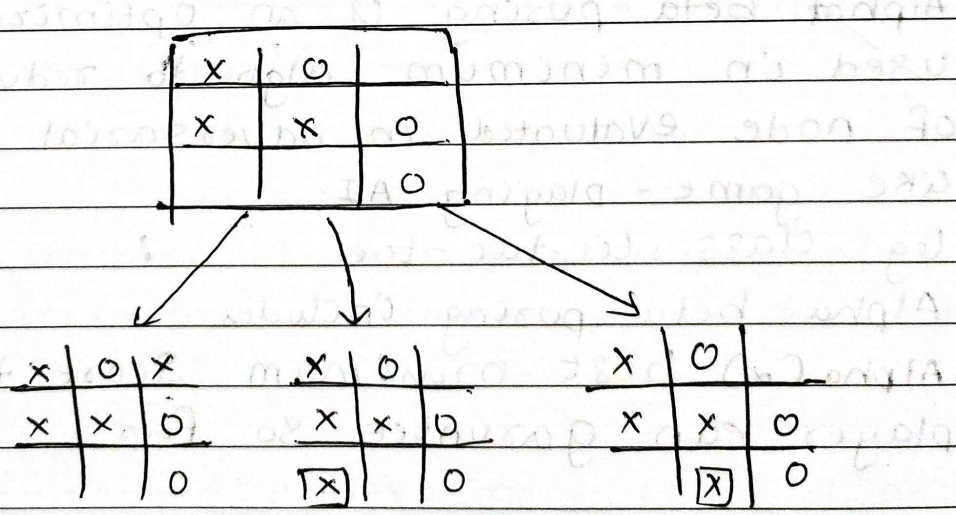
→ The min max algo is a decision making algo used in 2 player games it answers one player (max) tries to maximize the score answers one player tries to minimize the score.

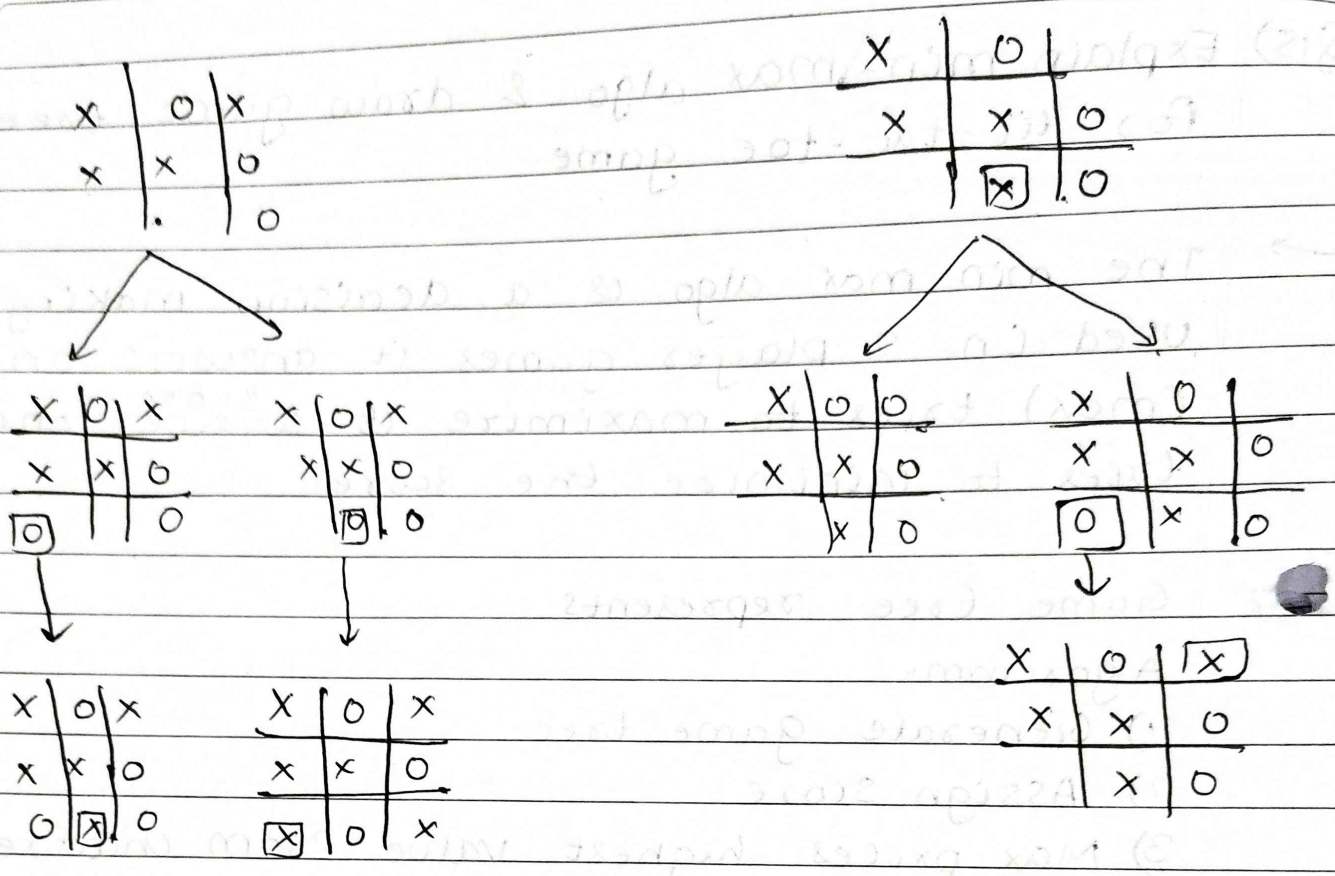
Game tree represents

Algorithm:

1) Generate game tree.
2) Assign score
3) Max prices highest value from children.
4) Repeat until root node is evaluated a bottom-up approach.

Game tree for the tic-tac-toe game.





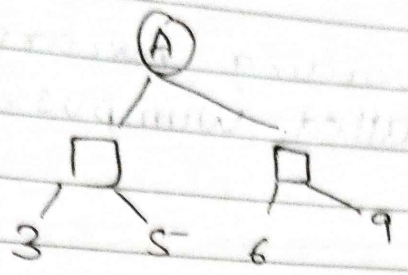
Q16 Explain alpha beta pruning algo for adversial Search with example.

→ Alphat beta pruning is an optimized technique used in minimum algo to reduce no. of node evaluated in adversarial Search problem like game-playing AI.
Eg class tic-tac-toe
Alpha beta pruning includes
Alpha (α) best maximum score that maximising player can garuntee so far.

Beta (β): ~~Maximum~~ minimum score that the minimizing player can gaurentee. So far.
the algo. prunes branches that will not influence final decision.

Page No.
Date

eg:



1. Start at root node A.
   $\alpha = -\infty$, $\beta = \infty$
2) Check left min node (Child of A)
   Check first child value = 3 → update $\beta = 3$.
   check second child value = 5 → update $\beta = 3$
   Min node return 3 to Max.
3) Right Min node (Child of A)
   - Check first child value = 6 → $\beta = 6$.
   - Here $\alpha = 3$ at max node but $\beta (6) > \alpha (3)$ so no pruning.
4) - Explores 2nd child (4) → Here pruning will occur
   - Min node a & so use prune node with value a.
4. Max value = 6.

Q17) Explain Wumpus World environment giving its PEAS description. Explain how percept seq. is generated.

→ Wumpus world environment is a simple grid based environment used in AI to study intelligent agent behaviour
In certain env. it is a turn based environment where an agent must navigate a care to find

gold while avoiding hazards like pits & a monster called Wumpus.

PEAS:

P: Agent is rewarded for grabbing gold & exiting safely- penalty is imposed for falling into pits & getting eaten by wumpus.

E: 4x4 grid world containing agent, wumpus, pits, gold.

A: Agent can move forward, left, right, shoot climb.

S: Agent previous search, breeze, gutter, wumpus scream.

Percept Sequence generation.

It is history of all perception achieved by agent AI each time step, agent AI each time step the agent percieves information based on its current location & surroundings.

Eg percepts Seq:

1) Agent start: at (1,1)
   - No I breeze, no stretch, no gutter → Safe squares
2) Agent moves to (2,1):
   - Breeze detect → A pit is nearby but not in current square.

3) Agent moves to (1,2)
Stench detected → wumpus is in adjacent cell.
4) Agent moves to (2,2)
Glitter detected → gold is here
5) Agent moves back to (1,1) & climbs out.

Q(8) Solve the Crypt Arithmetic SEND + MORE = MONEY

→ STEP 1 : M must be 1.
Sum of 2 '4' digital no. comes to be greater than equal to 20000.

$$\begin{array}{r} SEND \\ +\ MORE \\ \hline MONEY \end{array} \qquad \begin{array}{r} S\ E\ N\ D \\ +\ M\ O\ R\ E \\ \hline 1\ O\ N\ E\ Y \end{array}$$

Step 2: Assume E + O = 10 + N
then E + O generate Carry : S has to be 8 otherwise if S is 9 & Carry is 1 then 9 + 1 = 11 but 'O' cannot be 1 ∴ M = 1

Step 3 : but then E + O cannot generate Carry so inital assumption of carry is wrong & so S ≤ 9

$$\begin{array}{r} 9END \\ 10RE \\ \hline 1ONEY \end{array}$$

Page No.
Date

Step 4: Now $E + 0 = N$

$\Rightarrow E = N$

but then is a contr So $N+R$ must generate Carry 1 & $E + 1 + 0 = N$

$N = 1 + E$

$N + R + K1 = 10 + E$

$R\ 4, K1 = 10 - 1 = 9$

K cannot be 0 otherwise $R = 9$

$R + 1 = 9$

$R = 8$

```
  S E N D                    9 E N D
  M O R E        ⟶           1 D 8 E
M O N E Y                  1 0 N E Y
```

$D + E = 10 + Y$

but $N = E + 1$ Let assume $E = 5$ then $N = 6$

```
  9 5 6 D
  1 0 8 5
1 0 6 5 7
```

So only requirement is D should be a no. such that when added to 5 generate a query available space d 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

2, 3, 4, cannot be as Cannot generate Carry

$D = 7$

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

Page No.
Date

Q(9) Consider the following question

All people who are graduating are happy.

→ ① Represent it predicate logic

G(x) → x is graduating

H(x) is a happy.

1. Collect Clauses.

(1) ∝ 7 {¬G(x), H(x)}

(2) ∝ 7 {¬H(x), S(x)}

(3) {G(x)}

2. Apply resolution

Resolve (1) {¬G(x), H(x)} with (3) & G(x)

Substituting x=a

{¬G(a), H(a)}

∴ We have H(a) resolving gives H(a)

Resolves (2) {¬H(a), S(a)}

Since we derived S(a), we conclude that someone (a) is smiling.

3) Prove 'is Someone graduating using resolution.

(1) Collect clauses:

(1) {¬ → G(x), H(x)}

(2) {¬ → H(x), S(x)}

(3) { Gr(a) }

(4) Assume contradiction { ¬ S(a) }

2) Apply resolution

Resolve (1) { ¬ Gr(H), H(x) } with (3) { Gr(x) }

Substitute x = a → { Gr(a) }

{ ¬ Gr(a), H(a) }

We have Gr(a) resolving gives { H(a) }

Resolve (2) { ¬ H(x), S(x) } with { H(a) }

Substitute x = a

{ ¬ H(a), S(a) } & { H(a) }

Now left with { S(a) }

we can resolve it with { ¬ S(a) }

we are left with

our assumption of contradiction is wrong

Q20) Explain Modus ponen with suitable example.

→ Modus ponen is a fondamental rule of inference is propotional logic that allows us to deduce a conclusion from a conditional statement & its ~~accident~~ antecedint.

It follows form

1) P → Q (if P then Q)

2) P (P is true)

∴ Q (Q must be there)

Example : 1) If ground will be wet P → Q

2) If is rainning P ∴ Ground is wet → Ss.

Page No.
Date

Q21) Explain forward & backward chainning algo

→ Forward chainning - starts with given facts & applies inference chaning - It rule to derive new facts until goal is reached It is a data drive approach because it begins with known data & works to reach a conclusion.

Example Diagnosing a deseas

Rules

1) It a person has a fever & cough they might have fly.
2) It is a person has a throat & fever they might have cold.

Facts:

- The patient has a fever.
- The patient has cough.

Inference: 1) Fever Eough → Fly (rule 1 applies)
2) Conclusion. the patient might have fly

Backward It starts with goal & work backword by checking what facts are needed to support it. It is a goal drove approach

Eg: Diagnosing a desiase.

goal: Determine if patient has flu.

Page No.
Date

Rules:
1) (Fever ∩ Cough) → Flu.
2) (Sore throat ∩ Fever) → Cold.

Process using backward chainning.
1) We want to prove Flu
2) Looking at rule 1 (Fever ∩ Cough) → Flu.
   We need to check if patient has Fever & Cough.
3) We check our known facts.
   · Patient has Fever
   · Patient has cough.
4) Since both conflicts are met, we confirm Flu is true.

So...

Name:Sahil Motiramani Div:D15C Roll no.:35

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)
2. Find the Median (10pts)
3. Find the Mode (10pts)
4. Find the Interquartile range (20pts)

Ans:

Mean (10 pts):
Mean = Sum / Count = 1611 / 20 = 80.55

Median (10 pts):
Sorted Data: 59, 64, 66, 70, 76, 76, 76, 78, 79, 81, 82, 82, 84, 85, 88, 90, 90, 91, 95, 99
Median = (81 + 82) / 2 = 81.5

Mode (10 pts):
Mode = Most frequent value = 76

Interquartile Range (20 pts):
Q1 = (76 + 76) / 2 = 76
Q3 = (88 + 90) / 2 = 89
IQR = Q3 - Q1 =13

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

Ans:

1 Machine Learning for Kids

2. Target Audience:
    - Primarily designed for school students (ages 8–16), educators, and beginners who are new to machine learning and want a simplified introduction.
3. Use by Target Audience:
    - Students and teachers use the platform to build simple machine learning models (image, text, numbers) using labeled data.
    - It integrates with Scratch and Python, enabling users to apply machine learning models in games or applications.

- The drag-and-drop style helps students understand concepts like classification, training data, and testing.

4. Benefits:
    - Beginner-friendly with a visual interface.
    - Good for educational settings to introduce AI and ML concepts.
    - Offers real-world examples like recognizing text, images, or numbers.
    - Integrates with Scratch for interactive projects.
5. Drawbacks:
    - Limited to basic ML concepts – not suitable for advanced users.
    - Can be slow with larger datasets.
    - Lacks in-depth customization or control over model parameters.

2 Teachable Machine

- Target Audience:
    - Geared toward students, creators, hobbyists, artists, and educators who want to experiment with machine learning quickly and without code.
- Use by Target Audience:
    - Users create ML models by training with webcam input (images), microphone (sounds), or uploading files (poses, gestures, etc.).
    - Models can be used directly or exported for use in web apps or TensorFlow projects.
- Benefits:
    - No coding skills required.
    - Extremely easy to use – just collect samples and train.
    - Supports exporting to TensorFlow.js for developers to use in custom apps.
    - Great for prototyping and demos.
- Drawbacks:
    - Lacks flexibility for fine-tuning models.
    - Performance is limited by training data size and quality.
    - Focused mainly on classification tasks.

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

Ans:

Machine Learning for Kids – Descriptive Analytic

Reason: It helps users understand patterns in the data through examples like classifying types of text or recognizing objects. It's more focused on explaining and demonstrating concepts rather than forecasting future events.

Teachable Machine – Predictive Analytic

Reason: This tool is oriented towards making predictions, such as classifying new images or audio into one of the trained categories. It uses training data to predict the class of unseen input, fitting the definition of predictive analytics.

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

Ans:

Machine Learning for Kids – Supervised Learning

Reason: It uses labeled data provided by the user (e.g., labeling pictures of cats and dogs). The model is trained on these labeled inputs to learn how to classify new examples.

Teachable Machine – Supervised Learning

Reason: Similar to Machine Learning for Kids, Teachable Machine also uses labeled examples to train the model. The user teaches the machine by providing examples of each class, and the system learns to classify new input based on this.

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.

Ans:
A recent example of misinformation stemming from flawed data visualization involves the misrepresentation of Antarctic sea ice data to downplay climate change concerns.

Misleading Visualization of Antarctic Sea Ice Data

In April 2024, social media posts circulated a comparison of Antarctic sea ice levels on March 9, 1997, and March 9, 2024These posts claimed that the sea ice extent had remained unchanged over the 27-year period, suggesting that climate change warnings were exaggeratedHowever, this comparison was misleading for several reasons

1. **Cherry-Picked Data Points**:The comparison focused solely on two specific dates, ignoring the broader context and variability of sea ice levels over time
2. **Lack of Long-Term Trend Analysis**:By selecting isolated data points, the visualization failed to represent the overall declining trend in Antarctic sea ice extent observed in recent decades
3. **Misleading Visual Representation**:The visualization likely omitted essential elements such as error bars or trend lines, which are crucial for accurately conveying data variability and trends Experts from the National Snow and Ice Data Center (NSIDC) emphasized that such comparisons are not scientifically valid and can mislead the public about the realities of climate changeThey highlighted that while sea ice levels can fluctuate annually, the long-term trend shows a significant decline, particularly since 2015

Lessons Learned

This case underscores the importance of:

- **Contextualizing Data** Visualizations should present data within an appropriate temporal and spatial context to accurately reflect trends and pattern.
- **Avoiding Selective Representation** Selecting specific data points without justification can lead to misinterpretation and misinformation.
- **Ensuring Transparency** Including comprehensive labels, scales, and explanations in visualizations helps prevent misinterpretation. By adhering to these principles, data visualizations can more effectively inform the public and support evidence-based decision-making.

Source: Reuters Fact Check – Antarctic ice data cherry-picked to make false global warming claim

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

(https://www.kaggle.com/competitions/data-science-assignments)

Ans:

Dataset link: https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database

Dataset Description: Pima Indians Diabetes

This dataset contains medical diagnostic information of women of Pima Indian heritage, aged 21 and above. The goal is to predict whether a patient has diabetes based on several health-related measurements.

Feature Descriptions

- **Pregnancies**: Number of times the patient has been pregnant
- **Glucose**: Plasma glucose concentration (mg/dL) after 2 hours in an oral glucose tolerance test
- **BloodPressure**: Diastolic blood pressure (mm Hg)
- **SkinThickness**: Skin fold thickness (mm)
- **Insulin**: 2-Hour serum insulin (mu U/ml)
- **BMI**: Body Mass Index (weight in kg / height in m²)
- **DiabetesPedigreeFunction**: A score showing the likelihood of diabetes based on family history
- **Age**: Age of the patient (in years)
- **Outcome**: Target variable — `0` = No diabetes, `1` = Has diabetes

Name:Sahil Motiramani Div:D15C Roll no.:35

Step 1: Data loading

```
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, ConfusionMatrixDisplay
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt
import seaborn as sns
```

```
[2] df = pd.read_csv('/content/diabetes.csv')
    df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Step 2: Data preprocessing:

```
# Features and label
X = df.drop('Outcome', axis=1)
y = df['Outcome']

# Handle missing values in features (replace zeros with NaN where invalid)
cols_with_zero = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']
X[cols_with_zero] = X[cols_with_zero].replace(0, np.nan)

# Fill NaNs with mean values
X = X.fillna(X.mean())

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

Step 3: Handle Class Imbalance

```
from imblearn.over_sampling import SMOTE

# Apply SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_scaled, y)
```

We used SMOTE (Synthetic Minority Over-sampling Technique) to balance the dataset. It creates synthetic examples of the minority class by interpolating between existing ones. This helps prevent the model from being biased toward the majority class during training.

Step 4: Train, Validation and Test Split should be 70/20/10, Train and Test split must be randomly done:

```
# First split (10% test)
X_train_val, X_test, y_train_val, y_test = train_test_split(
    X_resampled, y_resampled, test_size=0.10, random_state=42, stratify=y_resampled)

# Second split (20% of remaining for validation)
X_train, X_val, y_train, y_val = train_test_split(
    X_train_val, y_train_val, test_size=2/9, random_state=42, stratify=y_train_val)

print(f"Train: {X_train.shape}, Validation: {X_val.shape}, Test: {X_test.shape}")
```

```
Train: (700, 8), Validation: (200, 8), Test: (100, 8)
```

Step 5: Train SVM with Hyperparameter Tuning

```
param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto']
}

grid = GridSearchCV(SVC(), param_grid, refit=True, cv=5, scoring='accuracy')
grid.fit(X_train, y_train)

print("Best Parameters:", grid.best_params_)
print("Validation Accuracy:", grid.score(X_val, y_val))
```

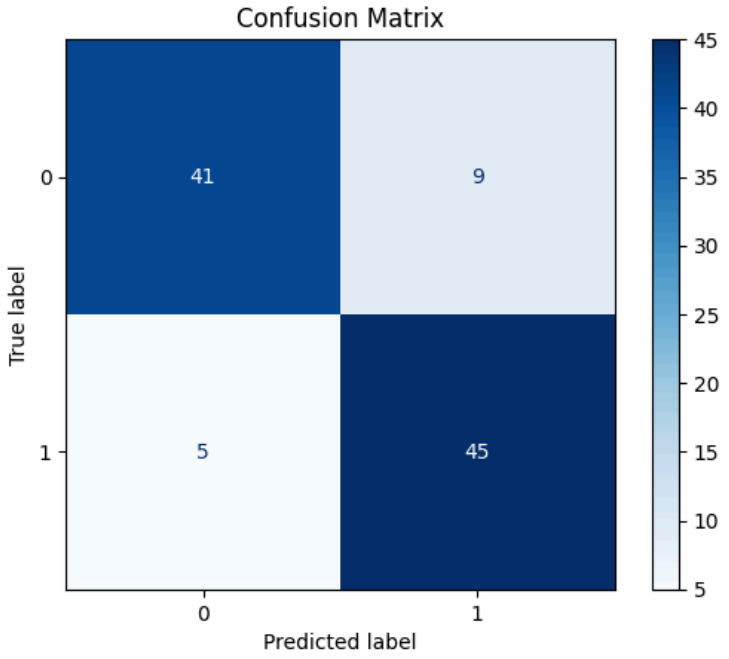
```
Best Parameters: {'C': 10, 'gamma': 'auto', 'kernel': 'rbf'}
Validation Accuracy: 0.8
```

In this step, we are training an SVM (Support Vector Machine) model to classify the data. We use GridSearchCV to test different combinations of parameters like C, kernel, and gamma.
It finds the best combination that gives the highest accuracy through cross-validation. This helps us choose the most effective settings for the SVM model automatically.

Step 6: Evaluate on Test Data

```
# Best model from GridSearchCV
best_model = grid.best_estimator_
y_pred = best_model.predict(X_test)

# Accuracy
print("Test Accuracy:", accuracy_score(y_test, y_pred))

# Classification report
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.show()
```

```
Test Accuracy: 0.86

Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.82      0.85        50
           1       0.83      0.90      0.87        50

    accuracy                           0.86       100
   macro avg       0.86      0.86      0.86       100
weighted avg       0.86      0.86      0.86       100
```



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
- 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

Ans:

Objective

To train a regression model using Ridge Regression with Polynomial Feature Expansion to predict real estate prices based on several features (like distance to MRT station, number of convenience stores, etc.). We aim to:

- Tune hyperparameters (alpha and polynomial degree)
- Use a modular, object-oriented design
- Evaluate using R^2, Adjusted R^2, and MSE
- Visualize predicted vs actual prices

Step-by-Step Explanation

1. Importing Required Libraries

```
import pandas as pd, numpy as np
from sklearn.linear_model import Ridge
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.pipeline import Pipeline
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
```

We use:

- Pipeline to streamline polynomial features → standardization → ridge regression.
- GridSearchCV for hyperparameter tuning
- r2_score, mean_squared_error for evaluation

2. Defining the RegressionModel Class

```
class RegressionModel:
    def __init__(self, model_pipeline, param_grid):
        ...
```

- Encapsulates model training, evaluation, and hyperparameter tuning.
- Reusable and extensible for other regression problems.

3. Loading the Dataset

```
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/00477/Real%20estate%20valuation%20data%20set.xlsx"
df = pd.read_excel(url)
```

The dataset contains real estate records including:

- Distance to MRT station
- Number of nearby convenience stores
- Age of building
- Geographic coordinates

4. Data Preprocessing

```
df = df.drop(columns=['No'])  # Drop irrelevant index
X = df.drop(columns=['Y house price of unit area'])  # Features
y = df['Y house price of unit area']  # Target
```

We define:

- X: all columns except house price
- y: the target variable (house price)

5. Train-Test Split

```
X_train, X_test, y_train, y_test = train_test_split(...)
```

- 70% training data, 30% testing
- random_state=42 ensures reproducibility

6. Model Pipeline & Hyperparameter Grid

```
pipeline = Pipeline([
   ('poly', PolynomialFeatures()),
   ('scaler', StandardScaler()),
   ('ridge', Ridge())
])
```

- Pipeline Components:
  - poly: adds non-linearity (degree=2 or more)
  - scaler: standardizes features (important for ridge!)
  - ridge: regularized regression
- Parameter Grid:

```
param_grid = {
   'poly__degree': [2, 3, 4],
   'ridge__alpha': [0.1, 1, 10, 100]
}
```

We let GridSearchCV try different combinations of:

- Polynomial degrees: 2 to 4
- Ridge regularization strengths (alpha): 0.1 to 100

7. Training and Evaluation

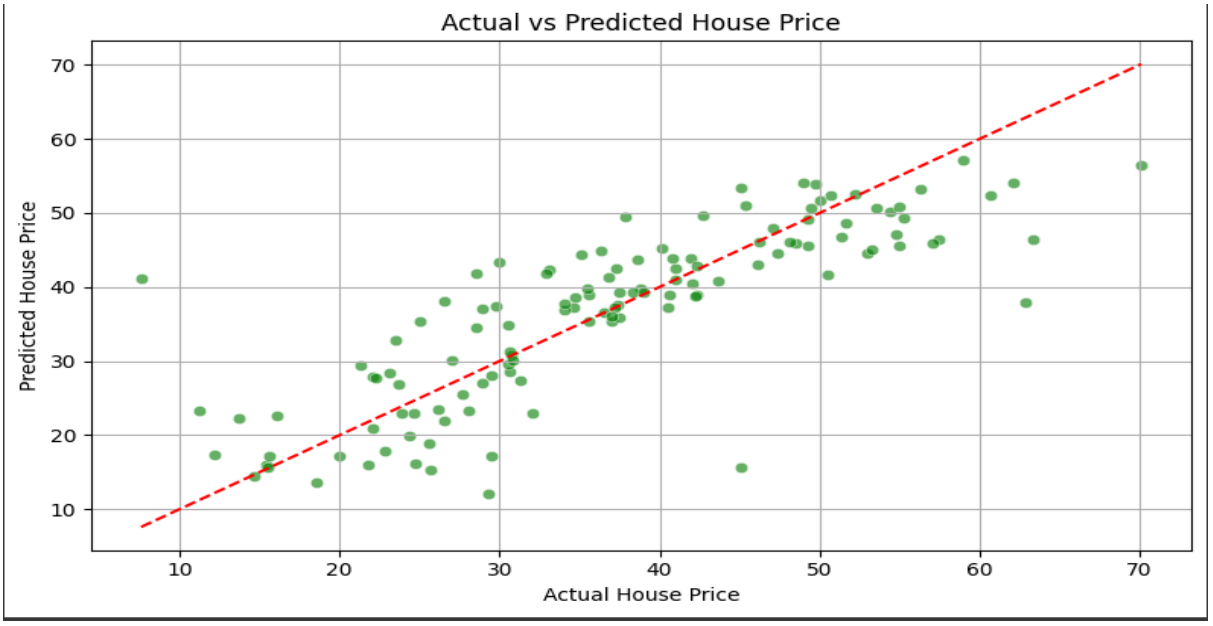
```
best_model = reg_model.train(X_train, y_train)
y_test_actual, y_pred = reg_model.evaluate(best_model, X_test, y_test)
```

- The model is trained with 5-fold cross-validation
- Best estimator is used for prediction
- We calculate:
  - R^2: proportion of variance explained
  - Adjusted R^2: penalizes for extra features
  - MSE: average squared prediction error

8. Visualization

```
sns.scatterplot(x=y_test_actual, y=y_pred)
```

- Compares actual vs predicted prices
- Red dashed line shows ideal predictions (perfect match)



Final Results:
Best Params: {'poly__degree': 2, 'ridge__alpha': 1}
R^2 Score: 0.6552
Adjusted R^2 Score: 0.6376
Mean Squared Error: 57.6670

- The model captures ~66% of the variance in house prices.
- There's room for improvement, but this is reasonable for real-world data.
- Adjusted R^2 shows performance after accounting for model complexity.

Why we May Not Reach $R^2 > 0.99$

- Real-world datasets include noise, missing features, and non-linear interactions.
- Polynomial features help but too high a degree → overfitting.
- Ridge helps reduce overfitting, but can't add missing signal.

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).

Ans:
https://www.kaggle.com/datasets/tawfikelmetwally/wine-dataset

Key Features & Importance

1. **Alcohol** – Positively correlated with quality; affects flavor and body.
2. **Malic Acid** – Impacts sourness; moderate levels improve taste.
3. **Ash / Alcalinity of Ash** – Reflect mineral content and stability.
4. **Magnesium** – Needed for fermentation; minimal effect on taste.
5. **Total Phenols & Flavanoids** – Strong indicators of quality; affect flavor, astringency.

6. **Nonflavanoid Phenols** – May reduce quality due to bitterness.
7. **Proanthocyanins** – Impact mouthfeel and aging.
8. **Color Intensity / Hue** – Affect visual appeal, perception of richness.
9. **OD280/OD315** – Indicates phenolic content; quality marker.
10. **Proline** – Highly correlated with quality; reflects grape richness.

Handling Missing Data

Although the original dataset typically has no missing data, if present:

Common Imputation Techniques:

Technique	Pros	Cons
Mean/Median	Simple, fast	May distort variance
Mode	Works for categorical-like numeric data	Not ideal for continuous data
KNN	Considers similarity	Slower, sensitive to irrelevant features
Multivariate (e.g., MICE)	Captures feature relations	Slower, assumes linearity
Dropping	Quick when missingness is low	Loses data, may bias model

Use **median** for simple cases, **KNN/MICE** for better accuracy.
Always **fit imputers on training data**, apply to test.

Feature Scaling

Since features are continuous and vary in scale:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Standardization improves model performance, especially for SVM, KNN, logistic regression, etc.

Summary

- **Important Features**: Alcohol, Flavanoids, Proline, Total Phenols, Color Intensity
- **Best Imputation**: Median for simplicity, KNN/MICE for accuracy
- **Always Standardize**: Ensures equal treatment of features