Name:Bhushan Malpani Div:D15C Roll no: 33

# Experiment 01

## Aim:

- Load data in Pandas.
- Description of the dataset.
- Drop columns that aren't useful.
- Drop rows with maximum missing values.
- Take care of missing data.
- Create dummy variables.
- Find out outliers (manually)
- standardization and normalization of columns

## Data preprocessing

Data preprocessing involves transforming raw data into a structured and meaningful format, making it suitable for analysis. It is a crucial step in data mining, as raw data often contains inconsistencies, missing values, or noise. Ensuring data quality is essential before applying machine learning or data mining algorithms to achieve accurate and reliable results.

### Why is Data Preprocessing Important?

Data preprocessing is essential for ensuring the quality and reliability of data before analysis. It helps improve the accuracy and efficiency of machine learning and data mining processes. The key aspects of data quality include:

- **Accuracy:** Ensuring the data is correct and free from errors.
- **Completeness:** Checking for missing or unrecorded data.
- **Consistency:** Verifying that data remains uniform across different sources.
- **Timeliness:** Ensuring the data is up-to-date and relevant.
- **Believability:** Assessing whether the data is reliable and trustworthy.
- **Interpretability:** Making sure the data is clear and easy to understand.

Dataset: Crop Yield Dataset

Name:Bhushan Malpani Div:D15C Roll no: 33

1) Loading Data in Pandas

```
import pandas as pd

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

print("Dataset Preview:")
print(df.head().to_string(index=False))
```

```
Dataset Preview:
         Crop  Crop_Year      Season State    Area  Production  Annual_Rainfall  Fertilizer  Pesticide       Yield
     Arecanut       1997 Whole Year  Assam 73814.0       56708           2051.4  7024878.38   22882.34    0.796087
    Arhar/Tur       1997 Kharif      Assam  6637.0        4685           2051.4   631643.29    2057.47    0.710435
  Castor seed       1997 Kharif      Assam   796.0          22           2051.4    75755.32     246.76    0.238333
     Coconut        1997 Whole Year  Assam 19656.0   126905000           2051.4  1870661.52    6093.36 5238.051739
Cotton(lint)       1997 Kharif      Assam  1739.0         794           2051.4   165500.63     539.09    0.420909
```

2)Description of the dataset.

| Attribute/Column Name | Data Type | Description |
|---|---|---|
| **Crop** | String | Name of the crop (e.g., Arecanut, Arhar/Tur, Coconut, etc.). |
| **Crop_Year** | Float | The year in which the crop was grown. |
| **Season** | String | The season in which the crop was cultivated (e.g., Kharif, Whole Year). |
| **State** | String | The state in which the crop was grown. |
| **Area** | Float | The total area (in hectares) used for cultivation. |
| **Production** | Float | The total production of the crop (in metric tons). |
| **Annual_Rainfall** | Float | The annual rainfall (in mm) received in the region. |
| **Fertilizer** | Float | The amount of fertilizer (in kg) used. |
| **Pesticide** | Float | The amount of pesticide (in kg) used. |
| **Yield** | Float | The yield (production per unit area) of the crop. |

df.info(): Provides an overview of the dataset, including:

- Number of rows and columns.
- Data types of each column (e.g., int, float, object).
- Number of non-null (non-missing) values in each column.

df.describe(): Generates summary statistics for numeric columns, such as:

- count: Number of non-missing values.
- mean: Average value.
- std: Standard deviation.

- min, 25%, 50% (median), 75%, and max: Percentile values.

```
import pandas as pd

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

print(df.info())
print(df.describe().to_string(index=False))
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19689 entries, 0 to 19688
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Crop             19689 non-null  object
 1   Crop_Year        19689 non-null  int64
 2   Season           19689 non-null  object
 3   State            19689 non-null  object
 4   Area             19689 non-null  float64
 5   Production       19689 non-null  int64
 6   Annual_Rainfall  19689 non-null  float64
 7   Fertilizer       19689 non-null  float64
 8   Pesticide        19689 non-null  float64
 9   Yield            19689 non-null  float64
dtypes: float64(5), int64(2), object(3)
memory usage: 1.5+ MB
None
   Crop_Year         Area   Production  Annual_Rainfall   Fertilizer    Pesticide        Yield
19689.000000 1.968900e+04 1.968900e+04     19689.000000 1.968900e+04 1.968900e+04 19689.000000
 2009.127584 1.799266e+05 1.643594e+07      1437.755177 2.410331e+07 4.884835e+04    79.954009
    6.498099 7.328287e+05 2.630568e+08       816.909589 9.494600e+07 2.132874e+05   878.306193
 1997.000000 5.000000e-01 0.000000e+00       301.300000 5.417000e+01 9.000000e-02     0.000000
 2004.000000 1.390000e+03 1.393000e+03       940.700000 1.880146e+05 3.567000e+02     0.600000
 2010.000000 9.317000e+03 1.380400e+04      1247.600000 1.234957e+06 2.421900e+03     1.030000
 2015.000000 7.511200e+04 1.227180e+05      1643.700000 1.000385e+07 2.004170e+04     2.388889
 2020.000000 5.080810e+07 6.326000e+09      6552.700000 4.835407e+09 1.575051e+07 21105.000000
```

3) Drop columns that aren’t useful: Columns like Invoice ID may not contribute to analysis (it's often just an identifier). Removing irrelevant columns reduces complexity.

```
import pandas as pd

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

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

| | Crop | Season | State | Area | Production | Annual_Rainfall | Fertilizer | Pesticide | Yield |
|---|---|---|---|---|---|---|---|---|---|
| 0 | Arecanut | Whole Year | Assam | 73814.0 | 56708 | 2051.4 | 7024878.38 | 22882.34 | 0.796087 |
| 1 | Arhar/Tur | Kharif | Assam | 6637.0 | 4685 | 2051.4 | 631643.29 | 2057.47 | 0.710435 |
| 2 | Castor seed | Kharif | Assam | 796.0 | 22 | 2051.4 | 75755.32 | 246.76 | 0.238333 |
| 3 | Coconut | Whole Year | Assam | 19656.0 | 126905000 | 2051.4 | 1870661.52 | 6093.36 | 5238.051739 |
| 4 | Cotton(lint) | Kharif | Assam | 1739.0 | 794 | 2051.4 | 165500.63 | 539.09 | 0.420909 |

4)Drop rows with maximum missing values.

df.dropna(thresh=int(0.5 * len(df.columns))):

- Drops rows where more than half the columns have missing (NaN) values.
- thresh=int(0.5 * len(df.columns)): Ensures that a row must have at least 50% non-null values to remain.

df = ...: Updates the DataFrame after dropping rows.
print(df.info()): Confirms that rows with excessive missing values have been removed.

```
import pandas as pd

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

df = df.dropna(thresh=int(0.5 * len(df.columns)))
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19689 entries, 0 to 19688
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Crop             19689 non-null  object
 1   Crop_Year        19689 non-null  int64
 2   Season           19689 non-null  object
 3   State            19689 non-null  object
 4   Area             19689 non-null  float64
 5   Production       19689 non-null  int64
 6   Annual_Rainfall  19689 non-null  float64
 7   Fertilizer       19689 non-null  float64
 8   Pesticide        19689 non-null  float64
 9   Yield            19689 non-null  float64
dtypes: float64(5), int64(2), object(3)
memory usage: 1.5+ MB
```

5)Take care of missing data.

df.fillna(df.mean()): Replaces missing values (NaN) in numeric columns with the mean of that column.

```
import pandas as pd

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

df = df.dropna(thresh=int(0.5 * len(df.columns)))

numeric_cols = df.select_dtypes(include=['float64', 'int64']).columns
df[numeric_cols] = df[numeric_cols].fillna(df[numeric_cols].mean())

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

```
Crop              0
Crop_Year         0
Season            0
State             0
Area              0
Production        0
Annual_Rainfall   0
Fertilizer        0
Pesticide         0
Yield             0
dtype: int64
```

6)Create dummy variables.

pd.get_dummies(): Converts categorical variables into dummy variables (binary indicators: 0 or 1).

- Example: The Gender column becomes Gender_Male (1 if Male, 0 otherwise).

columns=['...']: Specifies which columns to convert.
drop_first=True: Avoids the "dummy variable trap" by dropping one dummy variable to prevent multicollinearity.

Name:Bhushan Malpani Div:D15C Roll no: 33

```
import pandas as pd

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

df = df.dropna(thresh=int(0.5 * len(df.columns)))

df = pd.get_dummies(df, columns = ['Season'], drop_first = True)
df.head()
```

| Crop_Year | State | Area | Production | Annual_Rainfall | Fertilizer | Pesticide | Yield | Season_Kharif | Season_Rabi | Season_Summer | Season_Whole Year | Season_Winter |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1997 | Assam | 73814.0 | 56708 | 2051.4 | 7024878.38 | 22882.34 | 0.796087 | False | False | False | True | False |
| 1997 | Assam | 6637.0 | 4685 | 2051.4 | 631643.29 | 2057.47 | 0.710435 | True | False | False | False | False |
| 1997 | Assam | 796.0 | 22 | 2051.4 | 75755.32 | 246.76 | 0.238333 | True | False | False | False | False |
| 1997 | Assam | 19656.0 | 126905000 | 2051.4 | 1870661.52 | 6093.36 | 5238.051739 | False | False | False | True | False |
| 1997 | Assam | 1739.0 | 794 | 2051.4 | 165500.63 | 539.09 | 0.420909 | True | False | False | False | False |

7)Find out outliers (manually)

```
import pandas as pd

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

def detect_outliers(df, col):
    Q1 = df[col].quantile(0.25)  # First quartile (25th percentile)
    Q3 = df[col].quantile(0.75)  # Third quartile (75th percentile)
    IQR = Q3 - Q1  # Interquartile range

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

    return df[(df[col] < lower_bound) | (df[col] > upper_bound)]

# Detect outliers in the 'Yield' column
outliers = detect_outliers(df, 'Yield')

print(outliers)

if outliers.empty:
    print("No outliers detected.")
else:
    print(f"Outliers detected:\n{outliers}")

print(f"Number of outliers: {len(outliers)}")
```

Name:Bhushan Malpani Div:D15C Roll no: 33

```
            Crop  Crop_Year       Season      State      Area  Production  \
3       Coconut        1997  Whole Year       Assam   19656.0   126905000
7           Jute       1997  Kharif           Assam   94520.0      904095
14        Potato       1997  Whole Year       Assam   75259.0      671871
21     Sugarcane       1997  Kharif           Assam   31318.0     1287451
54     Sugarcane       1997  Whole Year   Karnataka  308857.0    28999269
...          ...        ...         ...         ...       ...         ...
19618  Sugarcane       2016  Winter           Odisha    5493.0      344294
19636     Potato       2017  Winter           Odisha    3966.0       41812
19647  Sugarcane       2017  Winter           Odisha    3713.0      240245
19665     Potato       2018  Winter           Odisha    4900.0       54455
19676  Sugarcane       2018  Winter           Odisha    6778.0      417672

       Annual_Rainfall   Fertilizer  Pesticide        Yield
3               2051.4   1870661.52    6093.36  5238.051739
7               2051.4   8995468.40   29301.20     9.919565
14              2051.4   7162399.03   23330.29     7.561304
21              2051.4   2980534.06    9708.58    41.896957
54              1266.7  29393920.69   95745.67    91.747368
...                ...         ...        ...         ...
19618           1460.5    841802.25    1922.55    56.160400
19636           1344.4    624407.04    1507.08    11.955455
19647           1344.4    584574.72    1410.94    56.588000
19665           1635.9    794780.00    1715.00    13.017308
19676           1635.9   1099391.60    2372.30    57.584545

[3065 rows x 10 columns]
Outliers detected:
            Crop  Crop_Year       Season      State      Area  Production  \
3       Coconut        1997  Whole Year       Assam   19656.0   126905000
7           Jute       1997  Kharif           Assam   94520.0      904095
14        Potato       1997  Whole Year       Assam   75259.0      671871
21     Sugarcane       1997  Kharif           Assam   31318.0     1287451
54     Sugarcane       1997  Whole Year   Karnataka  308857.0    28999269
...          ...        ...         ...         ...       ...         ...
19618  Sugarcane       2016  Winter           Odisha    5493.0      344294
19636     Potato       2017  Winter           Odisha    3966.0       41812
19647  Sugarcane       2017  Winter           Odisha    3713.0      240245
19665     Potato       2018  Winter           Odisha    4900.0       54455
19676  Sugarcane       2018  Winter           Odisha    6778.0      417672

       Annual_Rainfall   Fertilizer  Pesticide        Yield
3               2051.4   1870661.52    6093.36  5238.051739
7               2051.4   8995468.40   29301.20     9.919565
14              2051.4   7162399.03   23330.29     7.561304
21              2051.4   2980534.06    9708.58    41.896957
54              1266.7  29393920.69   95745.67    91.747368
...                ...         ...        ...         ...
19618           1460.5    841802.25    1922.55    56.160400
19636           1344.4    624407.04    1507.08    11.955455
19647           1344.4    584574.72    1410.94    56.588000
19665           1635.9    794780.00    1715.00    13.017308
19676           1635.9   1099391.60    2372.30    57.584545

[3065 rows x 10 columns]
Number of outliers: 3065
```

8) standardization and normalization of columns

**Standardization** is another scaling technique where the values are centered around the mean with a unit standard deviation. This means that the mean of the attribute becomes zero and the resultant distribution has a unit standard deviation.

Standardization equation

$$X' = \frac{X - \mu}{\sigma}$$

To standardize your data, we need to import the StandardScalar from the sklearn library and apply it to our dataset.

**Normalization** is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1. It is also known as Min-Max scaling.

Normalization equation

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Here, Xmax and Xmin are the maximum and the minimum values of the feature respectively.

- When the value of X is the minimum value in the column, the numerator will be 0, and hence X' is 0
- On the other hand, when the value of X is the maximum value in the column, the numerator is equal to the denominator and thus the value of X' is 1
- If the value of X is between the minimum and the maximum value, then the value of X' is between 0 and 1

To normalize your data, you need to import the MinMaxScalar from the sklearn library and apply it to our dataset.

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

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

cols_to_transform = ['Area', 'Production']

standard_scaler = StandardScaler()
minmax_scaler = MinMaxScaler()

df[cols_to_transform] = standard_scaler.fit_transform(df[cols_to_transform])

df[cols_to_transform] = minmax_scaler.fit_transform(df[cols_to_transform])

print(df[cols_to_transform].head())
```

```
       Area    Production
0  0.001453  8.964274e-06
1  0.000131  7.405944e-07
2  0.000016  3.477711e-09
3  0.000387  2.006086e-02
4  0.000034  1.255138e-07
```

## **Conclusion**:

Thus we have understood how to perform data preprocessing which can further be taken into exploratory data analysis and further in the Model preparation sequence.

Name:Bhushan Malpani Roll no:33 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.

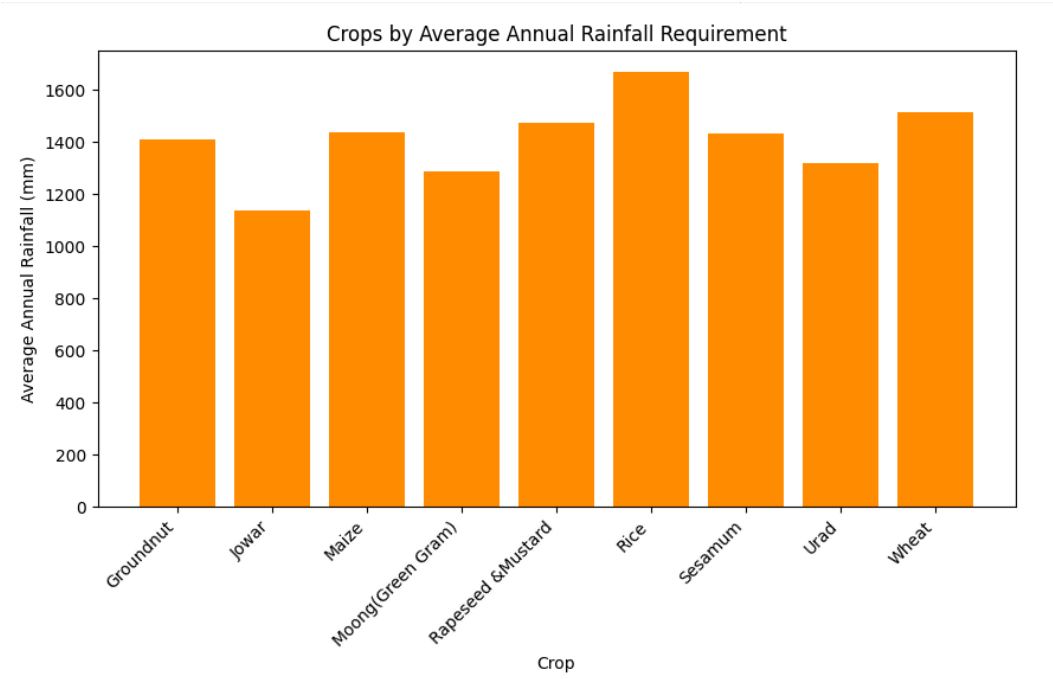
Name:Bhushan Malpani Roll no:33 Div:D15C

## 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.

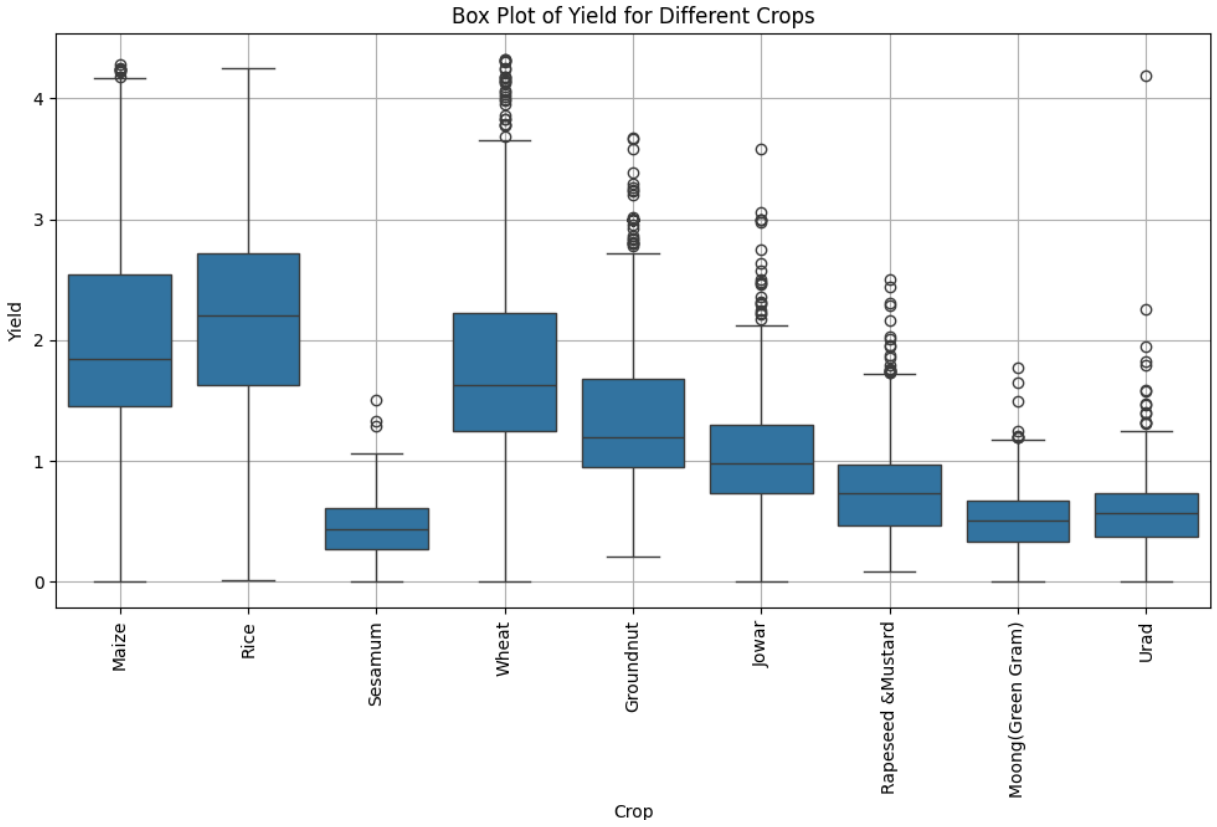
**Outliers:**
Data points lying outside the whiskers represent outliers, indicating exceptionally high or low yield values. These may be due to seasonal variations, extreme weather conditions, or data anomalies.

**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.

Name:Bhushan Malpani Roll no:33 Div:D15C

```
# 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)

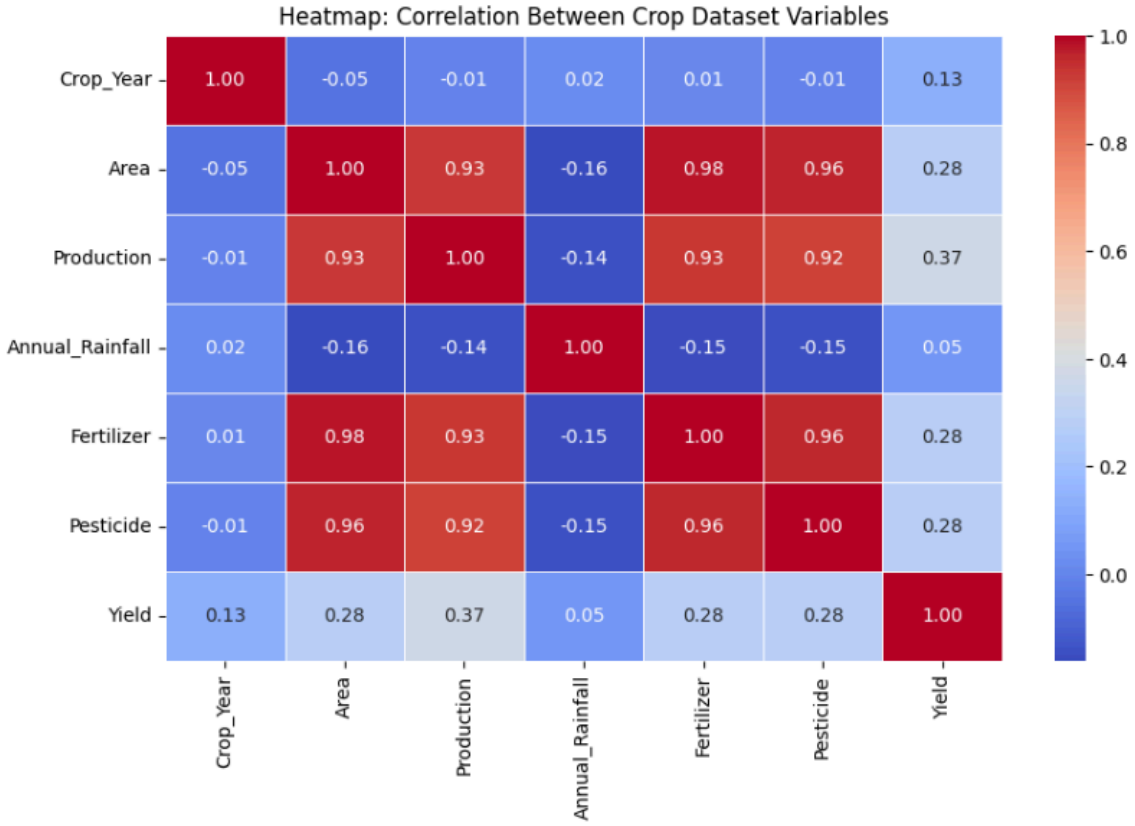
- Higher crop production is often linked to higher yield per unit area, meaning efficient farming techniques boost production.

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.

Name:Bhushan Malpani Roll no:33 Div:D15C

```
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()
```



## 6) Histogram

### Inference: Yield Distribution (From Histogram)

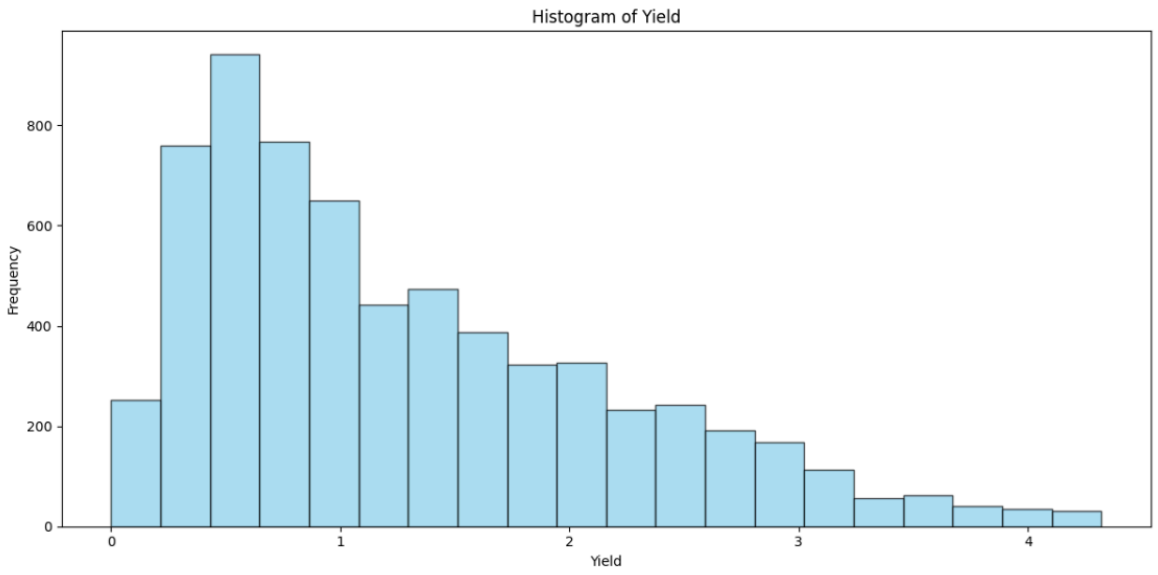
Most Common Yield Range:

- 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()
```



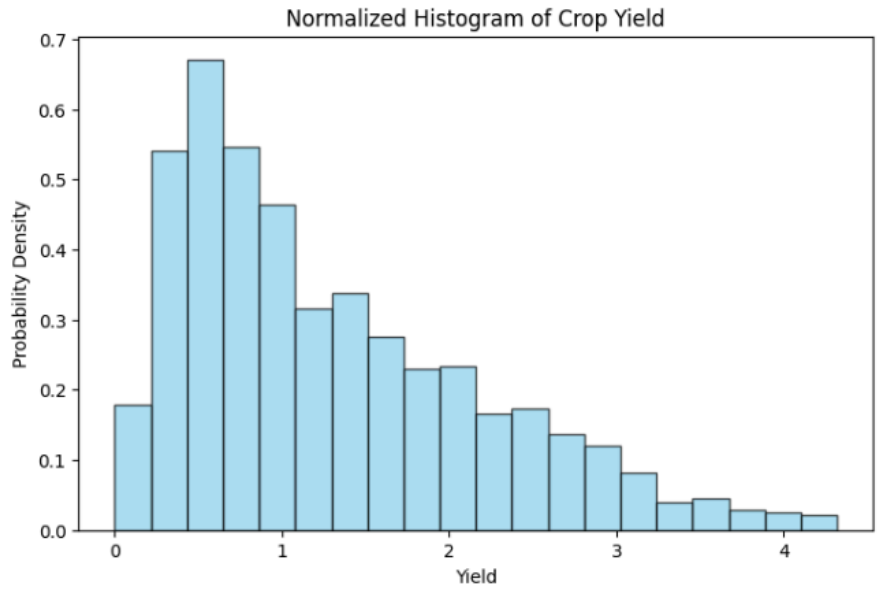
Name:Bhushan Malpani Roll no:33 Div:D15C

**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()
```



Name:Bhushan Malpani Roll no:33 Div:D15C

## 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.

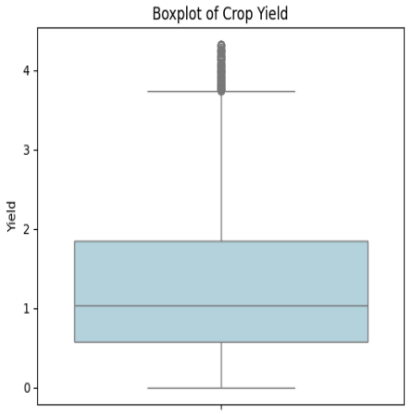
2.These outliers represent unusually high or low crop yields, possibly due to extreme weather, soil conditions, or measurement errors.

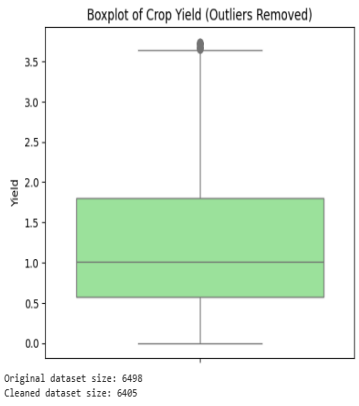
Yield Variability:

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:Bhushan Malpani Roll no:33 Div:D15C

# 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.

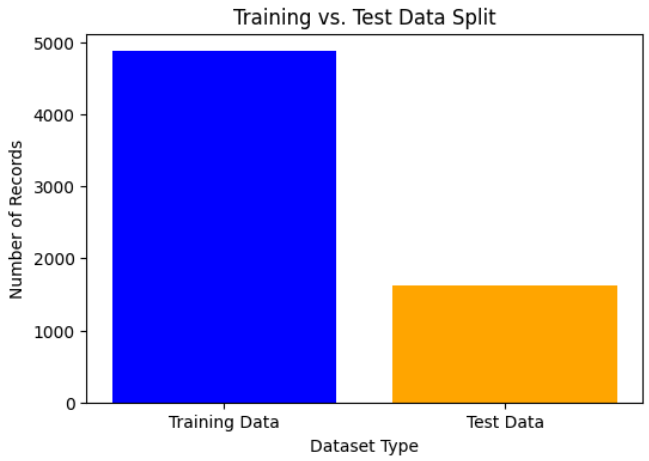
```
import pandas as pd
from sklearn.model_selection import train_test_split

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.

```
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:Bhushan Malpani Roll no:33 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:Bhushan Malpani Roll No:33 Div:D15C

Aim: Perform Regression Analysis using Scipy and Sci-kit learn.

Problem Statement:
a) Perform Logistic regression to find out relation between variables
b) Apply regression model technique to predict the data on above dataset.

## Dataset Description

The dataset contains 100,000 records with 14 attributes, focusing on dietary habits, health conditions, and lifestyle.

Key Features:

- Demographics: Age, Gender, Height (cm), Weight (kg), BMI
- Lifestyle & Health: Activity Level, Health Conditions, Dietary Preferences
- Dietary Data: Recommended Meals, Calories Intake
- Meals: Breakfast, Lunch, Snacks, Dinner

## Step 1: Load the Dataset

Upload your dataset to Google Colab, then read it using pandas.

```
from google.colab import files

file_path =S
df = pd.read_csv(file_path)
print(df.info())
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 14 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   Age                 100000 non-null  int64
 1   Gender              100000 non-null  object
 2   Height_cm           100000 non-null  float64
 3   Weight_kg           100000 non-null  float64
 4   BMI                 100000 non-null  float64
 5   Activity_Level      100000 non-null  object
 6   Health_Conditions   79945 non-null   object
 7   Dietary_Preferences 100000 non-null  object
 8   Recommended_Meals   100000 non-null  object
 9   Calories_Intake     100000 non-null  int64
 10  Breakfast           100000 non-null  object
 11  Lunch               100000 non-null  object
 12  Snacks              100000 non-null  object
 13  Dinner              100000 non-null  object
dtypes: float64(3), int64(2), object(9)
memory usage: 10.7+ MB
None
```

| | Age | Gender | Height_cm | Weight_kg | BMI | Activity_Level | Health_Conditions | Dietary_Preferences | Recommended_Meals | Calories_Intake | Breakfast | Lunch | Snacks | Dinner |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 56 | Other | 189.552819 | 76.388643 | 39.762724 | Sedentary | Heart Disease | Keto | Nuts, Yogurt | 2352 | Grilled Paneer, Vegetables | Oats, Fruits | Nuts, Yogurt | Salad, Brown Rice |
| 1 | 46 | Other | 144.649993 | 101.391668 | 15.588831 | Active | Diabetes | Paleo | Nuts, Yogurt | 2298 | Salad, Brown Rice | Grilled Paneer, Vegetables | Salad, Brown Rice | Oats, Fruits |
| 2 | 32 | Female | 158.142263 | 54.060753 | 19.146735 | Active | Obesity | Vegetarian | Oats, Fruits | 2021 | Oats, Fruits | Oats, Fruits | Oats, Fruits | Grilled Paneer, Vegetables |
| 3 | 60 | Other | 174.077462 | 97.286598 | 37.659330 | Very Active | Heart Disease | Vegetarian | Nuts, Yogurt | 2124 | Oats, Fruits | Oats, Fruits | Grilled Paneer, Vegetables | Salad, Brown Rice |
| 4 | 25 | Female | 144.947456 | 105.377330 | 16.587306 | Sedentary | Heart Disease | Vegetarian | Salad, Brown Rice | 2289 | Nuts, Yogurt | Grilled Paneer, Vegetables | Salad, Brown Rice | Nuts, Yogurt |

## Step 2: Preprocess the Data

Convert categorical variables into numerical form using LabelEncoder.

```
df.info()
df.describe()
df.isnull().sum()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 14 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   Age                  100000 non-null  int64  
 1   Gender               100000 non-null  object 
 2   Height_cm            100000 non-null  float64
 3   Weight_kg            100000 non-null  float64
 4   BMI                  100000 non-null  float64
 5   Activity_Level       100000 non-null  object 
 6   Health_Conditions    79945 non-null   object 
 7   Dietary_Preferences  100000 non-null  object 
 8   Recommended_Meals    100000 non-null  object 
 9   Calories_Intake      100000 non-null  int64  
 10  Breakfast            100000 non-null  object 
 11  Lunch                100000 non-null  object 
 12  Snacks               100000 non-null  object 
 13  Dinner               100000 non-null  object 
dtypes: float64(3), int64(2), object(9)
memory usage: 10.7+ MB
```

| | 0 |
|---|---|
| Age | 0 |
| Gender | 0 |
| Height_cm | 0 |
| Weight_kg | 0 |
| BMI | 0 |
| Activity_Level | 0 |
| Health_Conditions | 20055 |
| Dietary_Preferences | 0 |
| Recommended_Meals | 0 |
| Calories_Intake | 0 |
| Breakfast | 0 |
| Lunch | 0 |
| Snacks | 0 |
| Dinner | 0 |

dtype: int64

## Step 3: Perform Logistic Regression

Logistic Regression helps determine the relationship between Health Conditions and other feature.

```
label_encoders = {}
categorical_cols = ['Gender', 'Activity_Level', 'Health_Conditions', 'Dietary_Preferences']

for col in categorical_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le
```

Classification of dataset

```
X_classification = df[['Age', 'Gender', 'Height_cm', 'Weight_kg', 'BMI', 'Activity_Level', 'Health_Conditions']]
y_classification = df['Dietary_Preferences']
```

Training dataset and testing accuracy

```
X_train, X_test, y_train, y_test = train_test_split(X_classification, y_classification, test_size=0.2, random_state=42)
```

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
log_reg = LogisticRegression(max_iter=500)
log_reg.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(max_iter=500)
```

```
y_pred = log_reg.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

print("Accuracy:", accuracy)
print("Classification Report:\n", classification_rep)
```

```
Accuracy: 0.2016
Classification Report:
              precision    recall  f1-score   support

           0       0.20      0.12      0.15      4000
           1       0.19      0.20      0.20      3953
           2       0.21      0.28      0.24      4118
           3       0.21      0.12      0.16      4008
           4       0.20      0.28      0.23      3921

    accuracy                           0.20     20000
   macro avg       0.20      0.20      0.20     20000
weighted avg       0.20      0.20      0.20     20000
```

## Step 5: Perform Linear Regression

We'll predict Calories_Intake using a regression model.

Name:Bhushan Malpani Roll No:33 Div:D15C

```
X_regression = df[['Age', 'Gender', 'Height_cm', 'Weight_kg', 'BMI', 'Activity_Level', 'Health_Conditions']]
y_regression = df['Calories_Intake']
```

```
X_train, X_test, y_train, y_test = train_test_split(X_regression, y_regression, test_size=0.2, random_state=42)
```

```
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
y_pred = lin_reg.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print(f"MAE: {mae}")
print(f"RMSE: {rmse}")
print(f"R² Score: {r2}")
```

```
MAE: 175.00294674090316
RMSE: 202.0365965061598
R² Score: -0.00024337578128275084
```

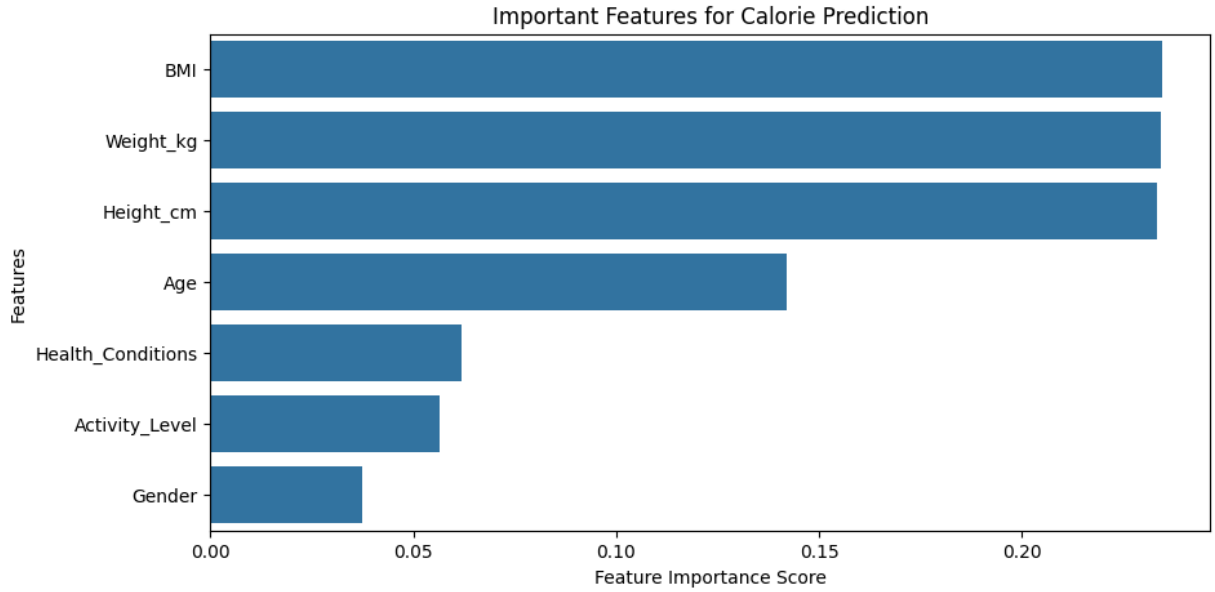
Using random Forest algo:

```
from sklearn.ensemble import RandomForestRegressor

rf_reg = RandomForestRegressor(n_estimators=100, random_state=42)
rf_reg.fit(X_train, y_train)

y_pred_rf = rf_reg.predict(X_test)

mae_rf = mean_absolute_error(y_test, y_pred_rf)
rmse_rf = np.sqrt(mean_squared_error(y_test, y_pred_rf))
r2_rf = r2_score(y_test, y_pred_rf)

print(f"Random Forest MAE: {mae_rf}")
print(f"Random Forest RMSE: {rmse_rf}")
print(f"Random Forest R² Score: {r2_rf}")
```

```
Random Forest MAE: 176.86412249999995
Random Forest RMSE: 205.31399015109758
Random Forest R² Score: -0.032958046218887205
```

Step 6: Visualize the Regression Predictions

```
feature_importance = pd.Series(rf_reg.feature_importances_, index=X_regression.columns).sort_values(ascending=False)

plt.figure(figsize=(10, 5))
sns.barplot(x=feature_importance, y=feature_importance.index)
plt.xlabel("Feature Importance Score")
plt.ylabel("Features")
plt.title("Important Features for Calorie Prediction")
plt.show()
```




## Conclusion

1. We successfully performed Logistic Regression to analyze the relationship between variables and classify Health_Conditions.
2. The accuracy of Logistic Regression was low (20.2%), indicating that the dataset may require better feature selection or more advanced models.
3. We applied Linear Regression to predict Calories_Intake, but the $R^2$ score was nearly zero, suggesting weak predictive power.
4. The Mean Absolute Error (MAE) was 175 calories, meaning the predictions had significant deviations.
5. Feature scaling, transformation, or using non-linear models like Random Forest could improve regression performance.
6. Visualization of predicted vs actual values showed a large variance, confirming the model's limitations.
7. Future work can focus on feature engineering and advanced ML models to enhance prediction accuracy.

NAME: Bhushan Malpani DIV: D15C ROLL NO:33

**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: Bhushan Malpani DIV: D15C ROLL NO:33

## 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.

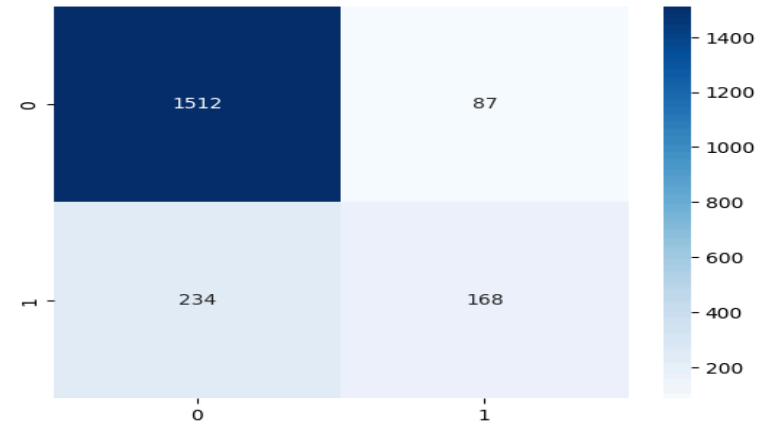
## Step 2: Implementing K-Nearest Neighbors (KNN) Classifier

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: Bhushan Malpani DIV: D15C ROLL NO:33

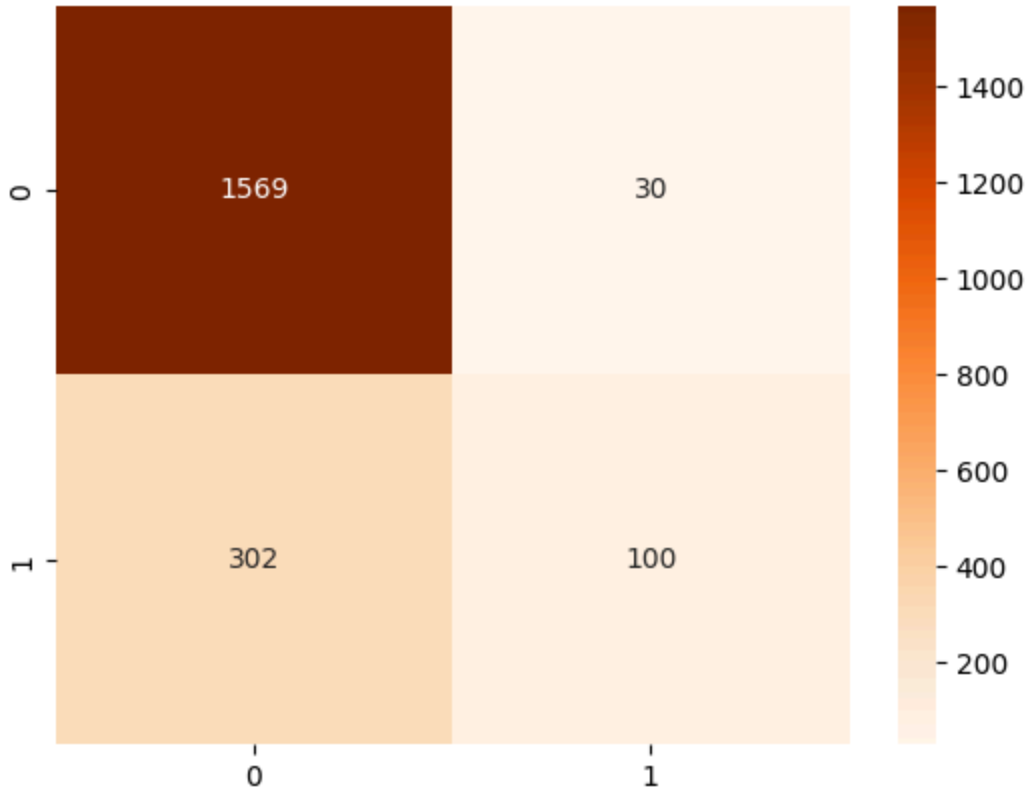
The Naïve Bayes classifier is a probabilistic machine learning algorithm based on Bayes' Theorem, assuming independence between features. Here, we use the GaussianNB model, which is suitable for continuous numerical features and assumes a normal distribution. The classifier is trained on X_train and 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: Bhushan Malpani DIV: D15C ROLL NO:33

```
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

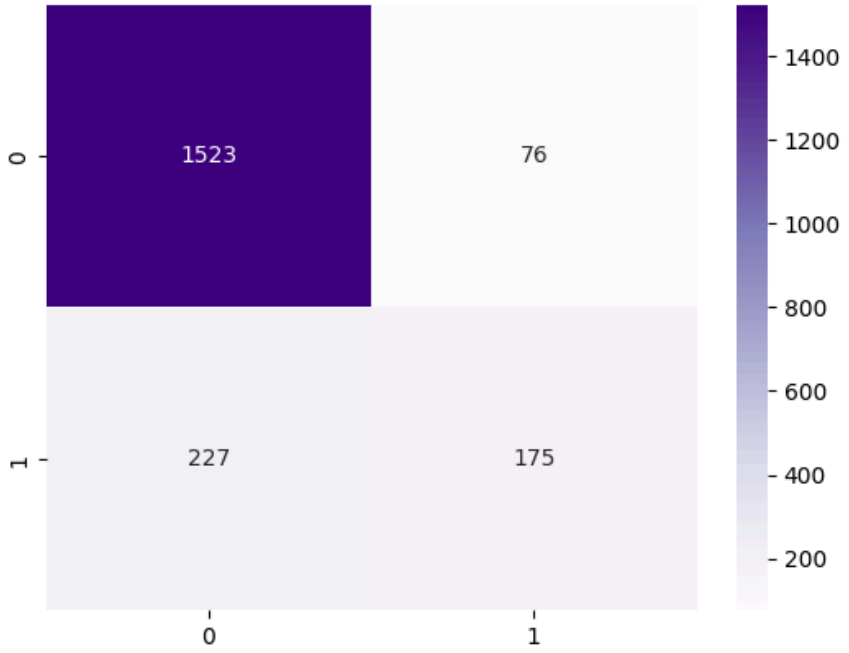
The Decision Tree algorithm is a supervised learning method used for classification and regression tasks. It recursively splits the dataset based on feature values, aiming to create pure subsets using a selected criterion (e.g., gini impurity or entropy). Here, we initialize a Decision Tree with max_depth=3, 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.

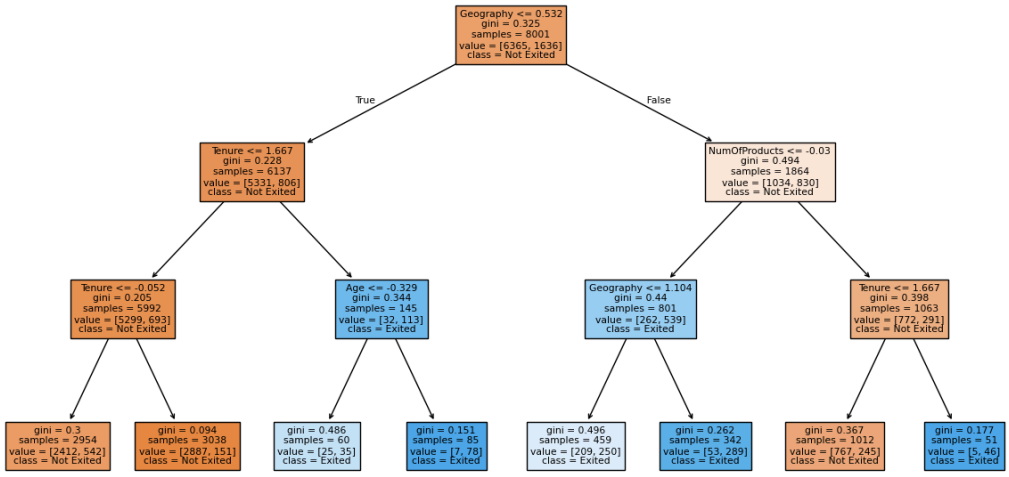
```
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns

# 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: Bhushan Malpani DIV: D15C ROLL NO:33

```
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: Bhushan Malpani DIV:D15C ROLL NO:33

**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)

NAME: Bhushan Malpani DIV:D15C ROLL NO:33

## 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: Bhushan Malpani DIV:D15C ROLL NO:33

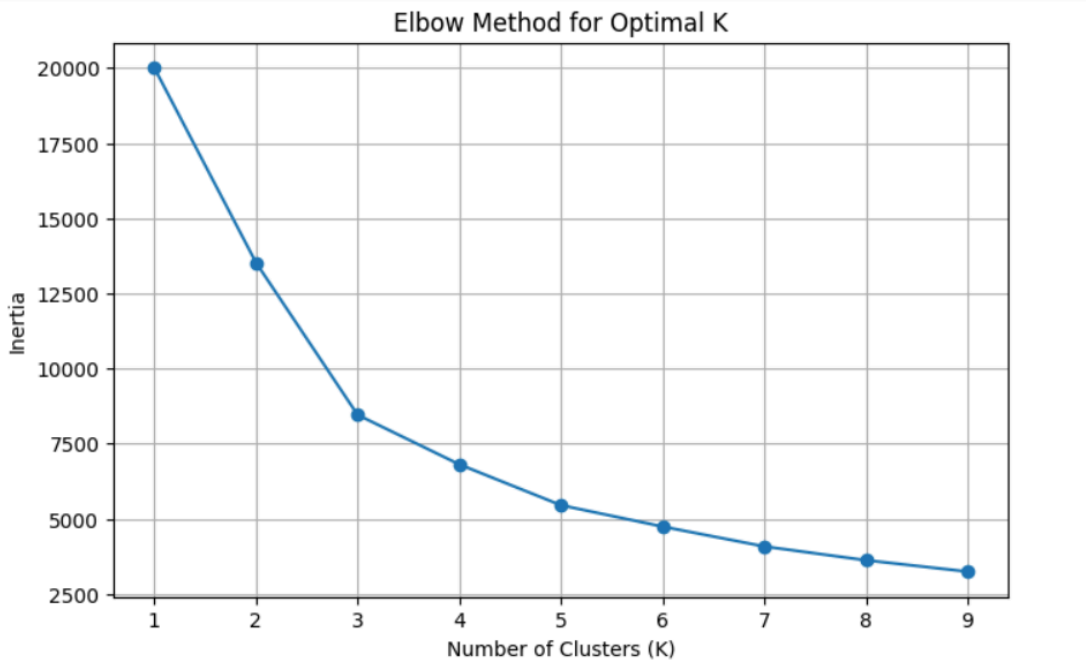
$$WCSS = \sum_{i=1}^{K} \sum_{x \in C_i} ||x - \mu_i||^2$$

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()
```

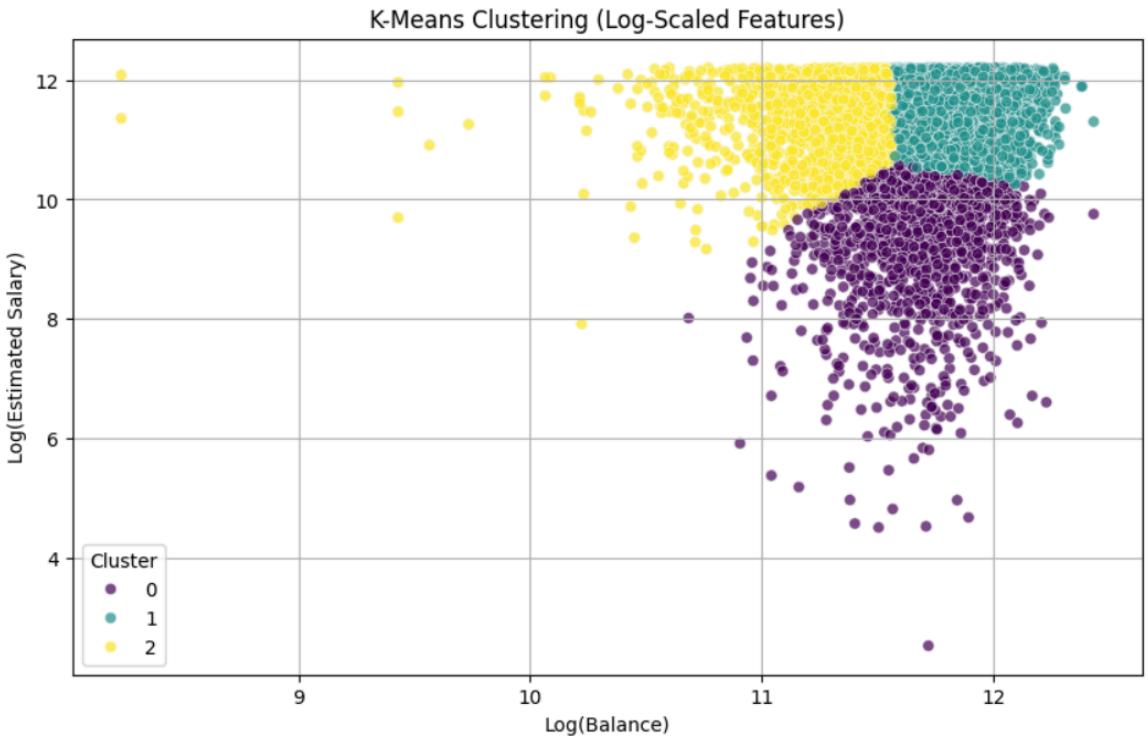
NAME: Bhushan Malpani DIV:D15C ROLL NO:33




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()
```

NAME: Bhushan Malpani DIV:D15C ROLL NO:33




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)

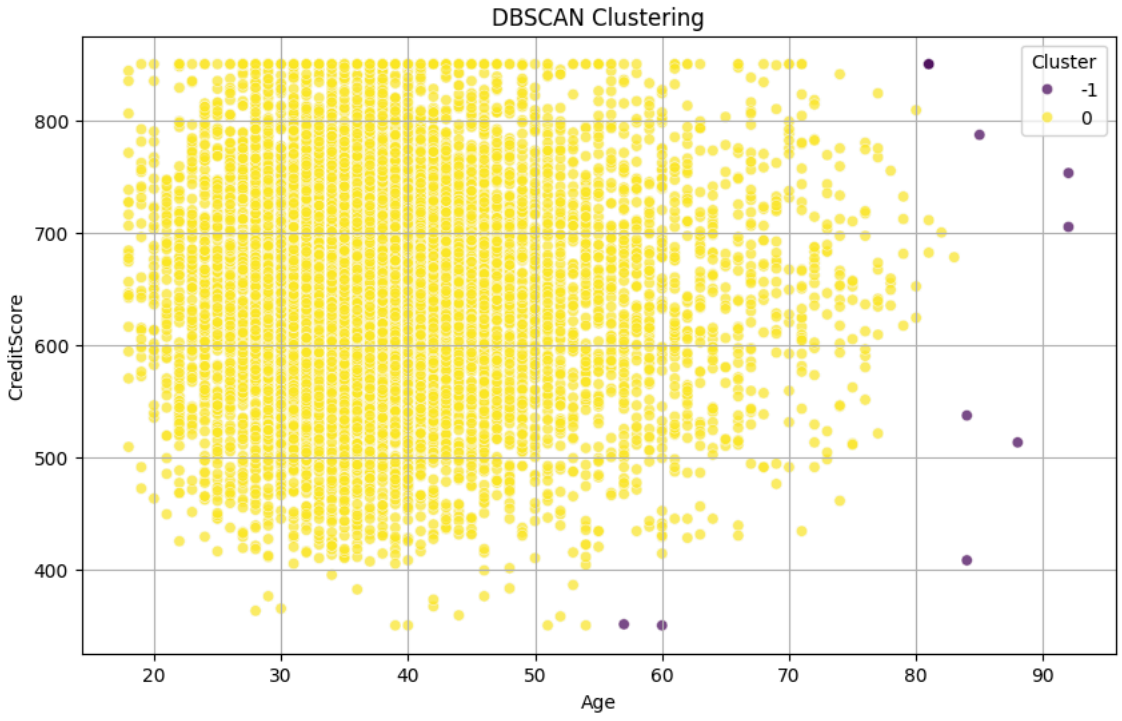
DBSCAN is a powerful clustering algorithm that groups together points that are densely packed and identifies outliers as noise. Unlike K-Means, it does not require specifying the number of clusters in advance and can detect clusters of arbitrary shapes.

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: Bhushan Malpani DIV:D15C ROLL NO:33

```
# 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:**

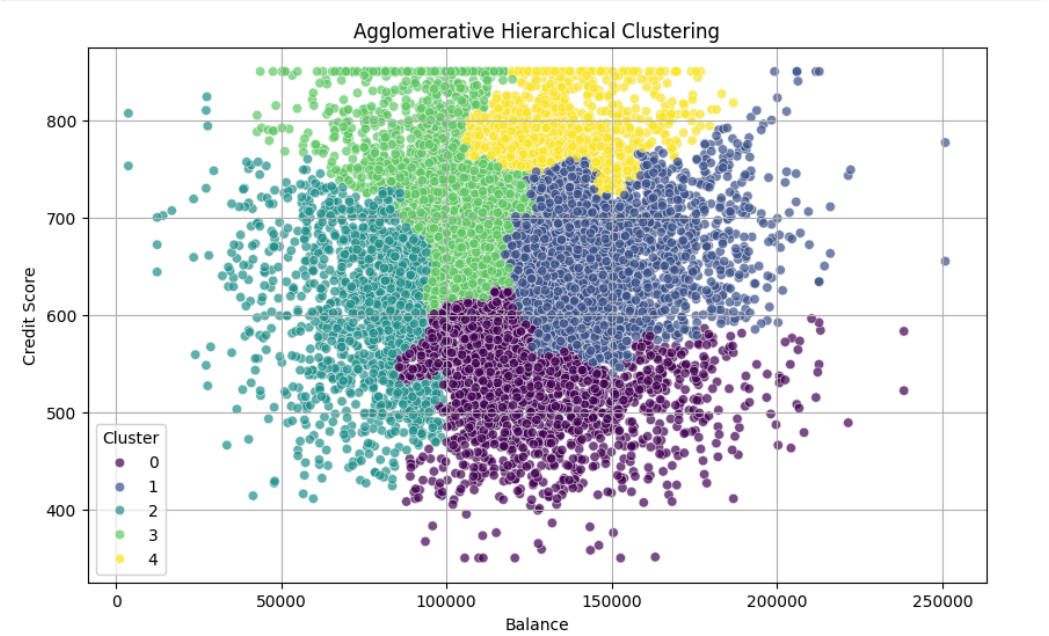
1. Each data point starts as a separate cluster.
2. Iteratively merge the two closest clusters.
3. Continue merging until the desired number of clusters is achieved

**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: Bhushan Malpani DIV:D15C ROLL NO:33




## 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: Bhushan Malpani DIV:D15C ROLL NO:33

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.

NAME: Bhushan Malpani DIV: D15C ROLL NO: 33

# Experiment No: 8

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

o Regression

o Classification

o Clustering

o Decision tree

o Anomaly detection

o Dimensionality Reduction

o Ensemble Methods

**Theory:**
A Recommendation System is a subclass of machine learning that helps suggest relevant items to users by analyzing historical data and item characteristics. In this experiment, we implement a Content-Based Recommendation System using Clustering, specifically the K-Means algorithm.

Clustering is an unsupervised learning technique used to group similar data points together. The K-Means algorithm partitions the dataset into K distinct clusters, where each data point belongs to the cluster with the nearest mean. For recommendation purposes, books are clustered based on features like genres and ratings, helping us find books with similar content and popularity.

By identifying the cluster a book belongs to, we can recommend other books from the same cluster, ensuring they share similar attributes. This method is particularly useful when user interaction or preference data is unavailable.

**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**

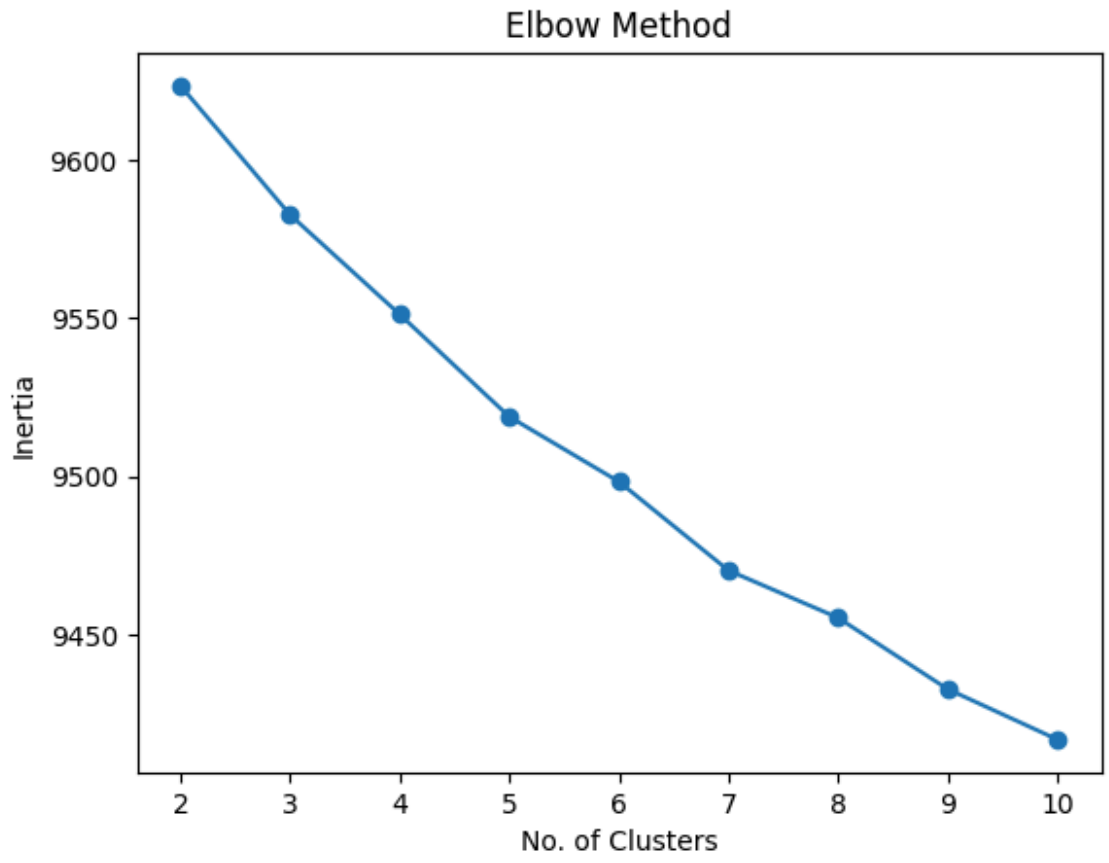
The Elbow Method is a popular technique used to determine the optimal number of clusters (k) in K-Means Clustering. It works by plotting the Within-Cluster Sum of Squares (WCSS) for different values of k (number of clusters), and selecting the k at which the WCSS starts to decrease slowly — forming an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

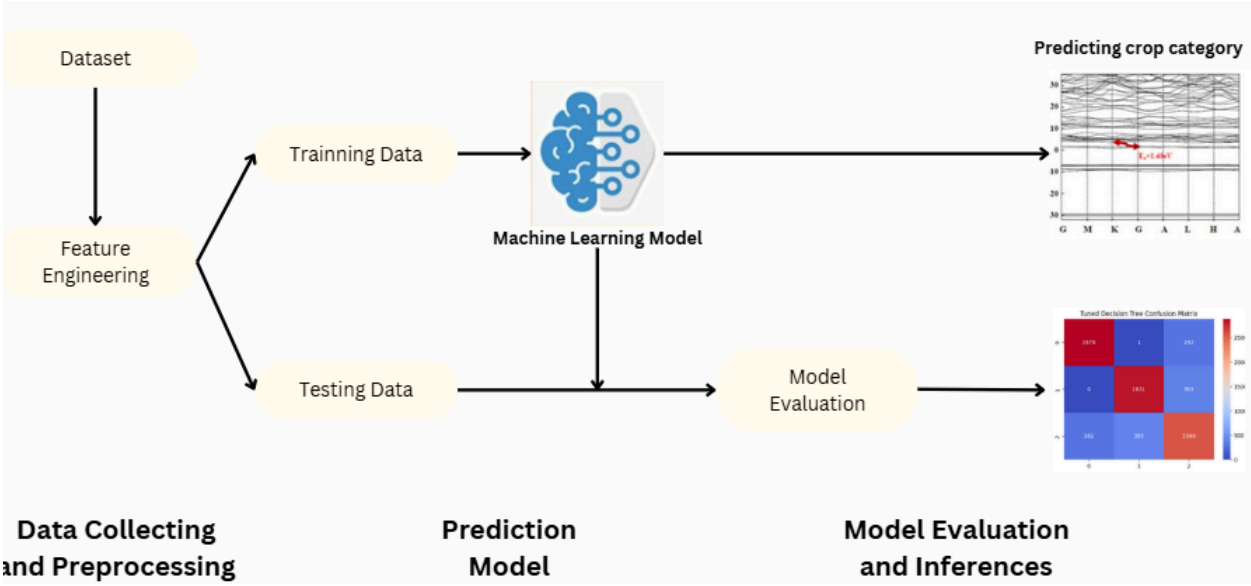
## 1.3.Review of Literature Survey

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.

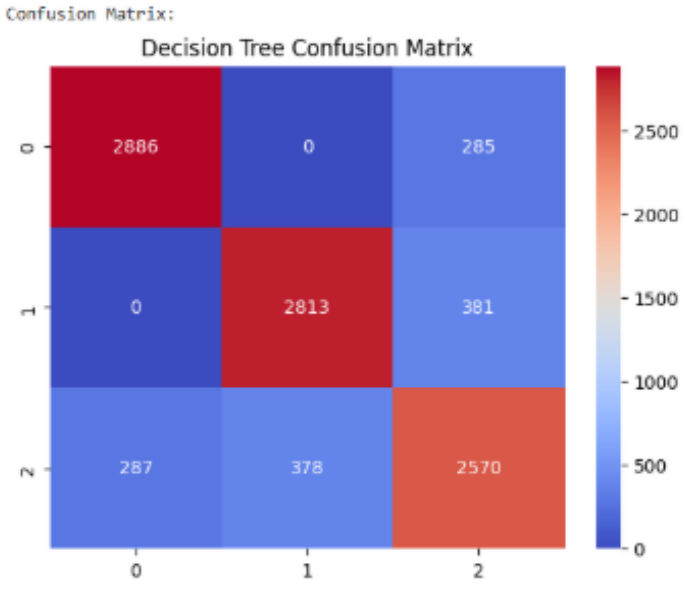
Both papers agree that machine learning brings adaptability and improved accuracy to price forecasting. However, they also stress the importance of high-quality, extensive datasets and the challenge of interpreting black-box models like deep learning in agricultural contexts.

## 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

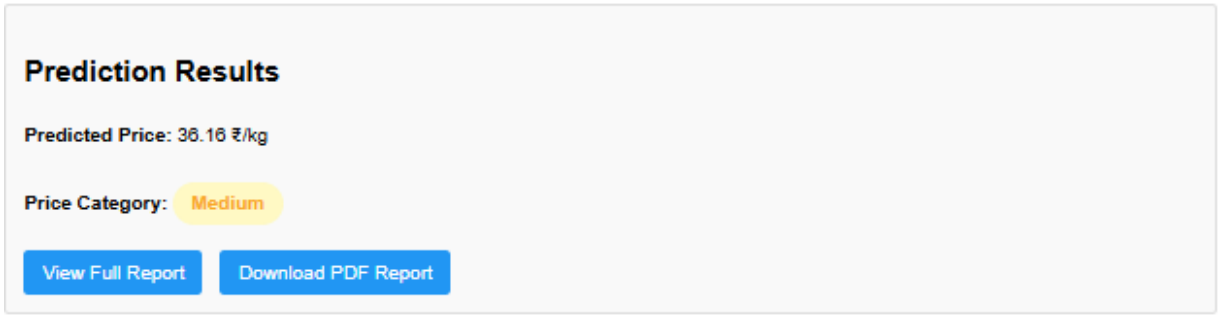
**Transportation Cost (₹/kg):** 250

**Fertilizer Usage (kg/hectare):** 20

**Pest Infestation (0-1):** 0.1

**Market Competition (0-1):** 0.2

Predict Price

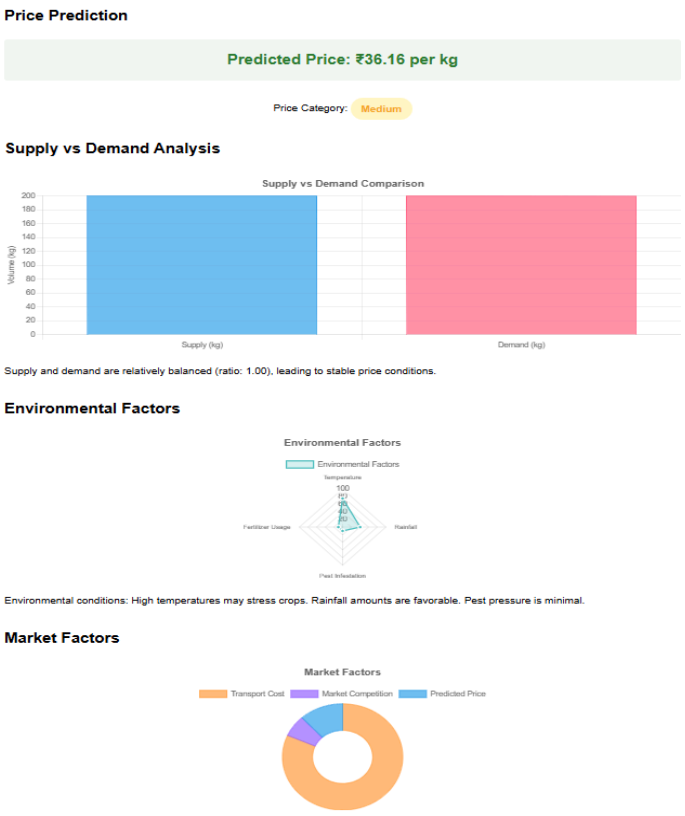



## Report Generation:

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

Name: Bhushan Malpani
Div: D15C
Roll No: 33

05/05

Page No.
Date 30.-

# ASSIGNMENT - I

Q1) What is AI & considering the COVID-19 pandemic situation, how AI helped to survive and renovated our way of life with different applications?

⇒ Artificial Intelligence (AI) enables machines to think, learn and make decisions like humans. It includes technologies like machine learning, NLP & robotics.

## Applications:

1) Healthcare: AI helped in early diagnosis, vaccine development, and chatbot-based health assitance.
2) Contact Tracing: AI-powered apps tracked COVID-19 exposure ensuring public safety.
3) Remote work & Education: AI enhanced virtual meetings, online learning & productivity tools.
4) Supply chain & Delivery: AI optimized logistics & enabled autonomous deliveries.
5) Mental Health Support: AI-driven apps provided emotional & fitness assistance.

Q2) What are AI Agents terminology, explain with examples.

⇒ 1) Agent: An entity that interacts with the environment & makes decision based on inputs.
ex: A self-driving car perceives traffic signals & adjusts speed accordingly.

2) Performance measures: Defines how successful an agent is in achieving its goal.
ex: A self-driving car's performance measures could be minimizing accidents, fuel efficiency & travel time.

3) Behavior/Action of Agent: The action an agent takes based on its percepts.
ex: A robotic vaccuum cleaner moves around obstacles after detecting them.

4) Percept: The data an agent receives at a specific moment from sensors.
ex: A spam filter receives an email & detects keywords, sender info, and attachments.

5) Percept sequence: The entire history of percepts received by an agent.
ex: A chess playing AI remembers all previous moves in the game before making its next move.

6) Agent Function: A mapping from the percept sequence to an action.
ex: A smart thermostat analyzes past tempreture changes & adjusts heating accordingly.

Q3) How AI technique is used to solve 8-puzzle problem?
⇒ It consists of a 3x3 grid with 8 numbered tiles & one empty space, where the objective is to move the tiles around to match a predefined goal configuration.

Initial State:

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | | 6 |
| 7 | 5 | 8 |

This is the random starting configuration of the 8 puzzle with the tiles placed in a non-goal configuration.

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

Goal State

1 2 3
4 5 6
7 8

* Solving the 8 Puzzle problem.

- AI search algorithms, such as breadth-first search (BFS), depth-first search (DFS) and A*, are commonly used :

1> Breadth-First Search (BFS):

- BFS is an uninformed search algorithm that explores all possible state level by level, starting from the initial state.
- BFS guarantees that the solution found is the shortest in terms of number of moves, but it can be very slow.

Advantages:

Guaranteed to find the optimal solution.

Disadvantages:

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

2> Depth-First Search (DFS):

- DFS is another uninformed search algorithm that explores one branch of the state space tree as deep as possible before backtracking.

Advantages:

DFS is more memory-efficient than BFS.

Disadvantages:

DFS can get stuck in deep, non-optimal paths & may not find the shortest solution.

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

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

1) Taxi driver:
- P : Minimize travel time, fuel efficiency, passenger safety, obey traffic rules.
- E : Roads, traffic, passengers, weather, obstacles, pedestrians
- A : Steering, accelerator, brakes, turn signals, horn.
- S : Camera, GPS, speedometer, radar, LiDAR, microphone

2) Medical Diagnosis System:
- P : Accuracy of diagnosis, treatment success rate, response time
- E : Patient records, symptoms, medical tests, hospital database
- A : Display screen, printed prescriptions, notifications
- S : Patient input, lab reports, electronic health records

3) A music Composer:
- P : Quality of music, adherence to genre, audience engagement.
- E : Digital workspace, music production software, real time composition settings.
- A : Audio output, digital instrument selection, file saving/export
- S : User inputs, style preferences, tempo, feedback from listners, music theory contraints.

4) An aircraft Autolander:
P: Smooth landing, accuracy in reaching runway, passenger safety, fuel efficiency
E: Airspace, runway, weather, wind speed, visibility
A: Flight control, landing gears, brakes
S: GPS, airspeed indicator, gyroscope, radar, weather sensors

5) An essay Evaluator
P: Accuracy of grading, consistency, fairness, grammer
E: Digital text input, student essays, predefined grading criteria
A: Feedback generation, score assignment, ~~hand~~ highlighting errors, suggesting improvement.
S: Optical character recognition, NLP, grammer & spell checkers.

6) A robotic sentry gun for the keck lab.
P: Target accuracy, threat detection efficiency, response speed,
E: keck lab premises, intruders, lighting conditions, obstacles
A: Gun aiming system, firing mechanism, camera panning, alert system
S: Motion detectors, infrared sensors, cameras, LIDAR, radar

Q5) Categorize a shopping bot for an offline bookstore according to each of six dimension (fully/partially observable, determinstic/stochastic, episodic/sequential, static/dynamic, discrete/continuous, single/multi agent)

⇒ 1) Partially Observable: The bot may not have complete visibility.
2) Stochastic: The enviroment is unpredictable.
3) Sequential: Each decision bot makes affects future states
4) Dynamic: The bookstore environment changes over time
5) Discrete: Bot choose discrete choices (selecting books)
6) multi agent: The bot interacts with multiple entities.

Page No.
Date

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 function to measure performance & make option choices. |
| 2) Relies on stored knowledge & updates the model. | 2) Chooses actions based on maximizing expected utility. |
| 3) Can adapt to changing environment by updating the internal model. | 3) More flexible & goal-oriented, adapting to changes dynamically. |
| 4) Moderate complexity due to model maintainance. | 4) Higher complexity due to the need to compute utilities for different actions. |
| 5) Ex: Self-driving car that predicts pedestrian movement | 5) A self driving car that evaluates & routes & selects the best one. |

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 decisions using knowledge base (KB) and reasoning machanism.

Architecture Component:

1) knowledge base: Stores fact, rules & heuristics about the world.
2) Inference Engine: Use logical reasoning (FOL) to derive new knowledge from the KB.
3) Perception Module: Collects data from sensor & update the KB.
4) Action selection Module: Chooses appropriate actions based on reasoning outcomes.

5) Communication Module: Allows interaction with other agents

Working Process:

- The agent perceives the environment & updates its KB.
- The interference engine applies logical rules to infer new knowledge.
- The agent decides on action and executes it.
- The KB is continuously updated to improve decision-making.

2) learning Agent Architecture:

- A learning agent improves its performance over time by learning from past experiences & interactions with the environment

Architecture: components:

1) Learning Element: Analyzes feedback from the environment and improves knowledge.
2) Performance Element: Make decisions & executes actions.
3) Critic : Evaluates the agent's action & provides feedback.
4) Problem Generator: Suggests explatory actions to improve learning.

Working Process:

- The performances element selects an action.
- The critic evaluates the action & provides feedback
- The learning element updates the agents knowledge to improve future decisions.
- The problem generator suggests new strategies to explore better solutions.

Page No.
Date

Q8) What is AI? Considering the COVID-19 pandemic situation, how AI helped to survive & renovated our way of life with different applications?

⇒ • Artifical Intelligence (AI) is the simulation of human intelligence in machines that can learn, reason & make decision. AI system process large datasets, recognize patterns & automate tasks, enhancing efficiency across institutes.

AI's role in the COVID-19 pandemic:

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

2) Chatbots & virtual assistants: Provided instant medical advice.

Q9) Convert the Following to predicates:

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

⇒

carAvailable → Travel by Car (Anita)

¬ Car Available → Travel By Bus (Anita)

b) Bus goes via Andheri and Goregaon.

goesVia (Bus, Andheri) ∧ ~~Goregaon~~ goesVia (Bus, Goregaon)

c) Car has punture & is not available

Puncture (car) → ¬ ~~car~~ Available (car)

Q) Will anita travel via Goregaon

From (c):

Punture (car) is true, punture (car) → ¬ Available (car)

For (a):

¬ carAvailable → Travel By Bus (Anita)

From b:

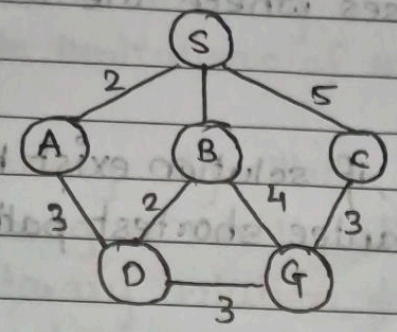
Goes via (Bus, Goregaon)

∴ Anita will travel via goregaon.

Galaxy S24

Page No.
Date

Q10) Find the route from S to G using BFS.




| Current Node | Queue | Visited node |
|---|---|---|
| ~~S~~ | A B C | S |
| ~~A~~ | B C | |

~~Step 1:~~

~~Selecting least cost between (A,2) (B~~

| 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 | D G | S→A→B→C |
| D | G | S→A→B→C→D |
| G | | S→A→B→C→D→G |

Path S→A→B→C→D→G

Q11) What do you mean by depth limited search? Explain iterative Deepening search with example.

→ Depth-Limited Search (DLS):

• Depth-Limited Search (DLS) is a variation of Depth-First Search (DFS) where the search is limited to a predefined depth. This helps prevent the algorithm from going too deep into the search tree

avoiding infinite loops in cases where the tree is unbounded.

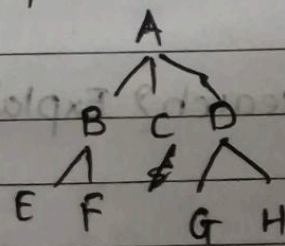
Key features:
1) Completeness: Not complete, if solution exist beyond depth limit.
2) Optimality: Doesn't guarantee shortest path.
3) Time Complexity: $O(b^d)$
4) Space Complexity: $O(b \times L)$

- When the search space is large or infinite
- If you know the appropriate depth where the solution may lie.

* Iterative Deepening Search (IDS):
- Iterative Deepening Search is a combination of DFS & BFS. It performs a series of Depth-limited searches with increase depth limits until a solution is found. This combines the space efficiently of DFS & the completeness of BFS.

Key Features:
1) Completeness: Yes, it will find solution if it exists.
2) Optimality: Yes, if unweighted.
3) Time Complexity: $O(b^d)$
4) Space Complexity: $O(b \times d)$

Example




1) Depth 0: Explore [A] → No solution.
2) Depth 1: Explore [A, B, C, D] → No solution.
3) Depth 2: Explore [A, B, C, D, E, F, G, H] → Solution found at H

Page No.
Date

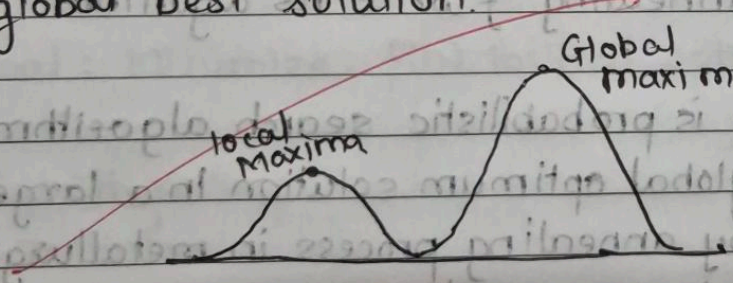
Q.12) Explain Hill Climbing and its drawbacks in detail with example Also state limitations of steepest-ascent hill climbing.

⇒

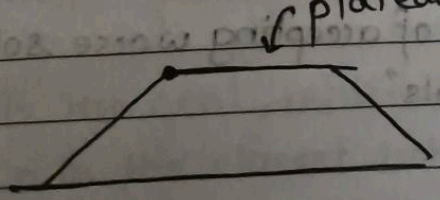
Hill climbing algorithm:
Hill climbing is an iterative search algorithm that starts with an arbitrary solution & makes small changes to improve it. The goal is to reach the optimal solution by continuously moving toward better states.
It is a local search algorithm that evaluates the neighbouring states & moves to the best one.

Types of hill climbing:
1) Simple hill climbing
2) Steepest-Ascent Hill climbing
3) Stochastic Hill climbing.

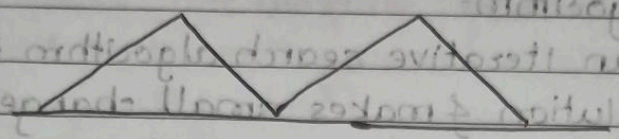
* Drawbacks of Hill Climbing:
1) local Maxima: Gets stuck at a peak that is not the global best solution.




2) Plateau: A flat region where all neighbouring values are the same, preventing progress.



3> Ridges: Steep paths where small moves are ineffective because the goal lies along a narrow ridge.



* Limitation of steepest-Ascent Hill climbing:

1> Increased Time complexity: Evaluating all neighbors requires more computation time.

2> local Optima: It can still stuck in local maxima, just like simple hill climbing.

3> Plateau Issue: Can't move forward if all neighbour have same value.

4> No backtracking: The algorithm can't revisit earlier states, even if a better solution lies there.

Q13> Explain simulated annealing & write its algorithm.

⇒

- Simulated annealing is probabilistic search algorithm used to find an appropriate global optimum solution in a large search space. It is inspired by annealing process in metallurgy, where metals are heated & slowly cooled to reduce defects & improve structure.
- Unlike hill climbing, which only moves to better solution, simulated annealing allows occasional moves to worse solutions to escape local optima. The probability of accepting worse solutions decrease over time as the system "cools".

Page No.
Date

Algorithm:
1) Initial solution: S
2) Initial tempreture: T
3) Cooling rate $\alpha$
4) Cost function f(s)
5) Stopping Condition (min tempreture).

Q14) Explain A* Algorithm with an example.

⇒ A* algorithm is a widely used search algorithm in AI for finding the shortest path between two points. It is an informed search algorithm that combines the strengths of both Uniform Cost Search and Greedy Best-first Search.

A* evaluates each nodes using the following cost function.

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

Where:
f(n) ⇒ Total estimated cost to reach the goal through node n.
g(n) ⇒ Actual cost from the start node to node n.
h(n) ⇒ Heuristics estimate of the cost from node n to the goal.

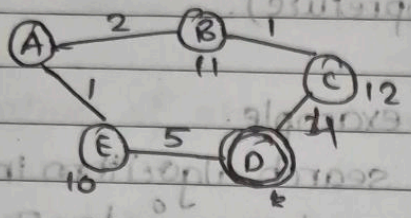
Goal: Minimize f(n) to find optimal path.

Algorithm Steps:
1) Initialize
- Open list (to track nodes to be explored)
- Closed list (to track explored nodes).

2) Add the start node to the open list.
3) Repeat until Goal is found.
- Select the node n from the open list with lowest f(n).
- IF n is the goal, return the path.
- Move n to the closest list.

- For each neighbour of n:
  - If the neighbour is in the closet list, skip it
  - If the neighbour is not in the open list, add it
  - Update the g, h and f values.

4) Return failure if no path is found




Start node: A

$A \rightarrow B \Rightarrow f(n) = g(n) + h(n)$

$= 2 + 11$

$= 13$

$A \rightarrow E \Rightarrow f(n) = g(n) + h(n)$

$= 1 + 10$

$= 11$

Select

$A \rightarrow E \rightarrow D = f(n) = g(n) + h(n)$

$= 1 + 5 + 0$

$= 6$

$A \rightarrow B \rightarrow C \Rightarrow f(n) = g(n) + h(n)$

$= 2 + 1 + 12$

$= 15$

$A \rightarrow B \rightarrow C \rightarrow D \Rightarrow f(n) = g(n) + h(n)$

$= 2 + 1 + 4$

$= 7$

Shortest path $A \rightarrow E \rightarrow D$ cost = 6

Page No.
Date

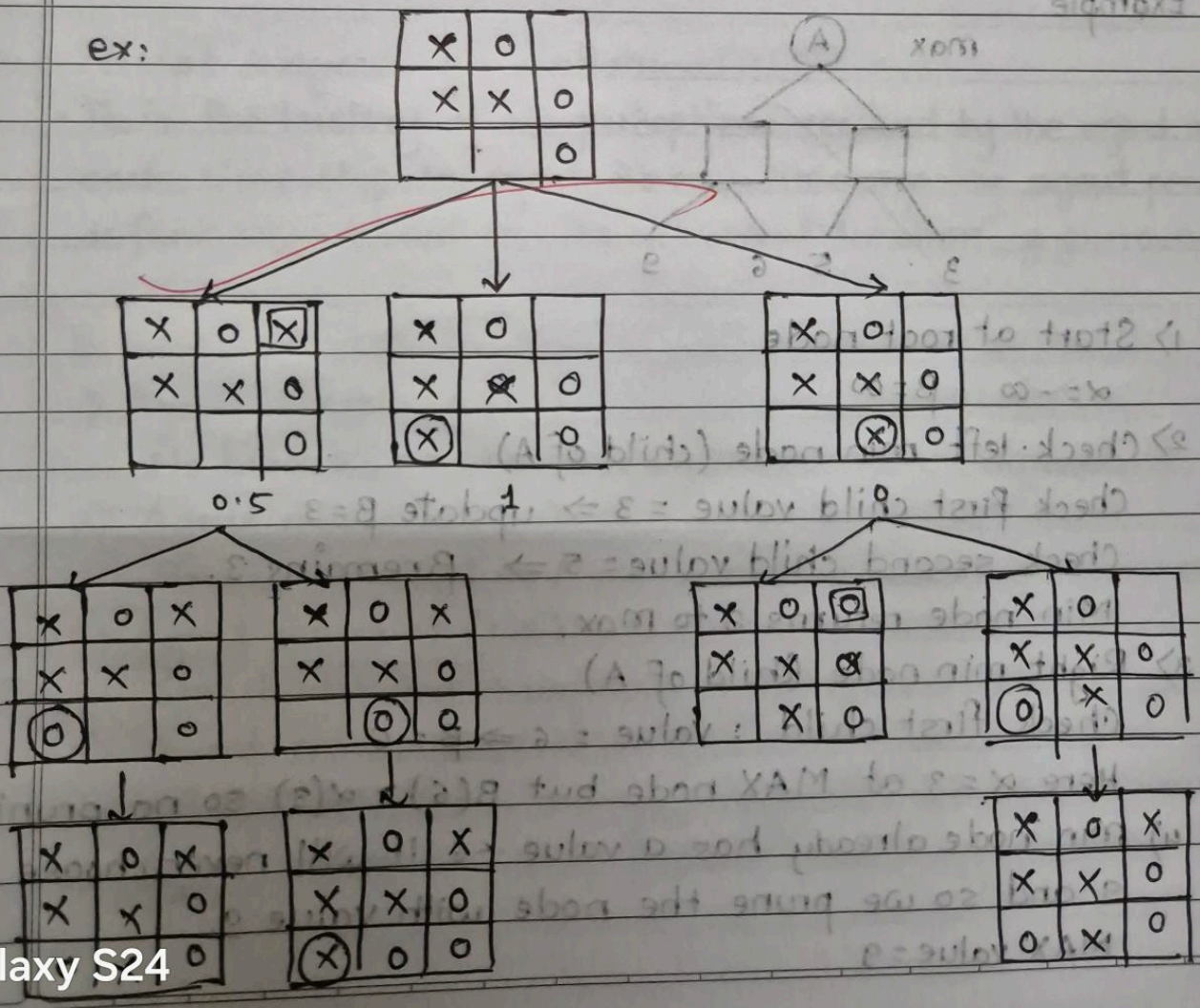
Q15) Explain min-max algorithm & draw game tree for tic-tac-toe game

⇒ • The min-max algorithm is a decision making algorithm used in 2 player games. It assumes one player [MAX] tries to maximize to score & other tries to minimize the score.

Algorithm:

1) Generate game tree.
2) Assign scores
3) MAX picks highest value from children
   MIN picks lowest value.
4) Repeat until root node is evaluated starting a bottom up aproach.

ex:

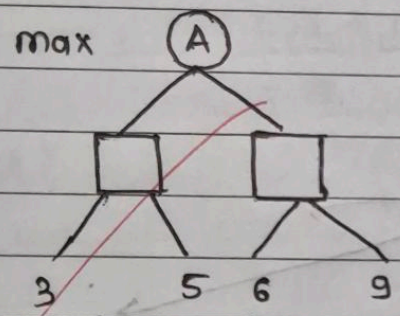


Q16) Explain alpha beta pruning algorithm for adversial search with example

⇒ Alpha beta pruning is an optimization technique used in minimax algorithm to reduce the number of nodes evaluated in adversial search problems like game playing AI (eg: chess).

Alpha beta pruning includes
Alpha (α): The best maximum score that the maximizing player can guarantee so far.
Beta (β): The best minimum score that the minimizing player can guarantee so far.

Example




1) Start at root node
$\alpha = -\infty$ $\beta = \infty$

2) Check left min node (child of A)
Check first child value = 3 ⇒ update $\beta = 3$
Check second child value = 5 ⇒ $\beta$ remains 3.
Min node returns 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) min node already has a value ≤6 it will never choose 9 and so we prune the node with value 9.
MAX value = 9

Page No.
Date

Q17) Explain WUMPAS world environment, giving its PEAS description. Explain how percept sequence is generated.

⇒ • The WUMPUS World environment is a simple grid-based environment used in AI to study intelligent agent behaviour.
• In uncertain environment, it is a turn based environment where an agent must navigate a case to find gold while avoiding hazards like pits & a monster called wumpas.

PEAS:

P: Grabbling gold, exit safely, Do not fall into pit.
E: 4x4 grid, agent, wumpus, pits, gold.
A: Move left, right, shoot, forward
S: Breeze (near pit), glitter (near gold), lump.

Percept sequence generation:

• It is the history of all perceptions received by the agent at each time step, ~~the agent~~ At each time step the agent perceives information based on its ~~as~~ current location & surrounding.

Example:

1) Agent starts at (1,1)
No breeze, no stench, no glitter → safe square.
2) Agent moves to (2,1)
Breeze detected → A pit is near but not on same square.
3) Agent moves at (1,2):
Stench detected → wumpus is in adjacent cell.
4) Agent moves to (2,2):
Glitter detected → ~~wumpus is in adjacent~~ gold is near.
5) Agent moves back to (1,1) & climbs out.

Q18) Solve for crypt Arithematic

SEND + MORE = MONEY

⇒ Step 1:

m=1 (Sum of 4 digit is 5 digit therefore carry from 4th digit).

```
  S E N D
  1 0 R E
---------
1 0 N E Y
```

Step 2:

~~Assume:~~

~~E+0 = 10+N~~ Since carry is generated for 5th digit

~~then~~

S+1 = 0+10

This is possible only if 9=9 ∴ 9+1 = 0+10

0=0

```
  9 E N D
+ 1 0 R E
---------
1 0 N E Y
```

Step 3:

E + 0 = N (This is not possible as E ≠ N)

∴ E + 0 + ~~N = 10~~

lets assume

E = 5

∴ N = 5 + 1 (carry)

N = 6

```
  9 5 6 0
+ 1 0 R 5
---------
1 0 6 5 Y
```

Page No.
Date

$6 + R = E + 5$

$\therefore R = 8$

$6 + 9 = 5 + 10$

$$\begin{array}{r} 9\ 5\ 6\ D \\ +\ 1\ 0\ 8\ 5 \\ \hline 1\ 0\ 6\ 5\ Y \end{array}$$

$D + 5 = Y$

We have $\{2, 3, 4, 7\}$ options left.
Since $(2, 3, 4)$ can't generate carry

$\therefore$ If $D = 7$

$$\begin{array}{r} 9\ 5\ 6\ 7 \\ +\ 1\ 0\ 8\ 5 \\ \hline 1\ 0\ 6\ 5\ 2 \end{array}$$

$\therefore$ Final solution

$S = 9$
$E = 5$
$N = 6$
$D = 7$
$M = 1$
$O = 0$
$R = 8$
$Y = 2$

Q19) Consider the following axioms.

- All people who are graduating are happy, All happy people are smiling, someone is graduating

⇒

Representing these axioms in first order predicate logic.

$G(x)$ = $x$ is graduating.
$H(x)$ = $x$ is happy
$S(x)$ = $x$ is smiling

Translating axiom into predicate logic.

1) All people who are graduating are happy.

$\forall x : G(x) \rightarrow H(x)$

2) All happy people are smiling

$\forall x : H(x) \rightarrow S(x)$

3) Someone is graduating

$\exists (x)\ G(x)$

Convert each formula to clause form

1. Convert implication to clausal form.

$\forall x\ G(x) \rightarrow H(x)$

- Using implication removal

$\forall x\ (G(x) \vee H(x))$

- In clause form

$\{\neg H(x), H(x)\}$

2. $\forall(x) : H(x) \rightarrow S(x)$

- Using implication removal

$\forall x (\neg H(x) \vee S(x))$

- In clausal form

$\{\neg H(x), S(x)\}$

3. $\exists x G(x)$

- In clausal form : $\{G(x)\}$

Prove " is something using resolution

1) Collect clauses

1) $\{\neg G(x), H(x)\}$
2) $\{\neg H(x), S(x)\}$
3) $\{G(x)\}$

2) Apply resolution

- Resolve (1) $\{\neg G(x), H(x)\}$ with $\{G(x)\}$

Substituting $x = a$

$\{\neg G(a), H(a)\}$

$\therefore$ we have $G(a)$, resolving gives $\{H(a)\}$

- Resolve (2) $\{\neg H(x), S(x)$ with $\{H(a)\}$

Substituting $x = a$

$\{\neg H(a), S(a)\}$

- Since, we have $H(a)$, resolving given

$\{S(a)\}$

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

Page No.
Date

Q20) Explain modus ponen with suitable example.

⇒ Modus ponen is a fundamental rule of interference in propositional logic that follows us to deduce a conclusion from a conditional statement & its antecedant.

It follows the form

1) $P \rightarrow Q$ (If P then Q)

2) P (P is true)

$\therefore Q$ (Q must be true)

Example:

1) If it is cold, you wear sweater

$P \rightarrow Q$

2) It is cold $\rightarrow P$

$\therefore$ You wear sweater Q

Q21) Explain forward and backward chaining algorithm with the help of example.

⇒ Forward Chaining: It starts with given facts and applies inference rules to derive new facts until the goal is reached. It is a data driven approach because it begins with known data & work forward to react a conclusion.

Ex: Diagnosing a disease.

Rules:

1) If a person has fever & cough they might have flu.

2) If a person has sore throat and fact, they might have cold

Facts:

1) The patient has a fever

2) The patient has cough.

Page No.
Date

Inference

1) Fever + cough → flu (rule 1 applies)
2) Conclusion, the patient might have flu.

Backward Chaining:

It starts with goal & works to backward by checking what facts are needed to support it. It is a goal driven approach

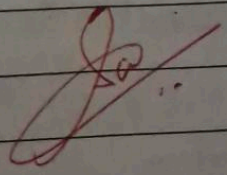
Example : Diagonising a disease
Goal: Determine if patient has flu.

Rules:

1) Fever ∧ cough → flu
2) Sour throat ∧ fever → Cold

Process:

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.
   - Patients has fever
   - Patient has cough.
4) Since, both conditions are met, we confirm flu is true.



NAME: Bhushan Malpani DIV: D15C ROLL NO: 33

**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)**

**Answer:**

1. Mean : Mean is the sum of all numbers divided by the total count.

   Total Sum = 82 + 66 + 70 + 59 + 90 + 78 + 76 + 95 + 99 + 84 + 88 + 76 + 82 + 81 + 91 + 64 + 79 + 76 + 85 + 90
   Total Sum = 1621
   N = 20

   Mean = Total sum / N= 1621 / 20 = 81.05

   Mean = 81.05

2. Median: Median is the middle value in an ordered list.

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

   Since n = 20,
   Therefore, median = average of 10th and 11th number
   Median = (81 + 82) / 2 = 81.5

   **Median = 81.5**

3. Mode: Mode is the number that appears most frequently.

   76 appears 3 times that is maximum than other

   **Mode = 76**

4. Interquartile Range (IQR):

   **Step 1**: Sort the data:
   59, 64, 66, 70, 76, 76, 76, 78, 79, 81, 82, 82, 84, 85, 88, 90, 90, 91, 95, 99

   Divide it into 2 halves

**Step 2**: First half (1st to 10th):
59, 64, 66, 70, 76, 76, 76, 78, 79, 81

Median of the first half:

5th and 6th values: 76 and 76
Q1 = (76 + 76)/2 = 76

**Step 3**: Second half (11th to 20th):
82, 82, 84, 85, 88, 90, 90, 91, 95, 99

Median of the second half:
5th and 6th values: 88 and 90
Q3 = (88 + 90)/2 = 89

**Step 4:**
IQR = Q3 − Q1 = 89 − 76 = 13

**Interquartile Range (IQR) = 13**

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

**Target Audience:**

- Students (typically in middle or high school).
- Teachers and educators introducing artificial intelligence (AI) and machine learning (ML) in classrooms.
- Beginners with little or no coding background.

**Use by Target Audience:**

- Students upload or input examples of text, numbers, or images and assign labels (e.g., happy/sad, dog/cat).
- The tool uses these examples to train a machine learning model.
- Students can then test the model and see how well it predicts new data.
- They can build fun projects using Scratch or Python that include their ML models — like games that respond to emotional messages or quizzes that react to images.

**Benefits**:

- Kid-friendly interface using blocks (Scratch) and easy menus.
- Makes ML interactive, fun, and creative for young learners.
- No need to install anything — it runs on the web.
- Encourages logical thinking and digital creativity.

**Drawbacks**:

- Only covers basic concepts — can't handle complex data science tasks.
- Performance is limited — not suitable for large datasets or real-world deployment.
- Relies on a working internet connection.

**Predictive or Descriptive Analytic:**

**I will choose predictive analytics and reasons for this are:**

- The tool learns from past labeled data and predicts a label for new input (e.g., predicting the emotion of a sentence).
- Example: "I am so excited!" → Model predicts: Happy.
- This prediction of future or unknown input classifies it as predictive, not just summarizing past data.

**Supervised / Unsupervised / Reinforcement Learning:**

**I will chose Supervised Learning because:**

- Users provide labeled training data (e.g., "I love ice cream" → Happy).
- The model learns based on those labels.
- Example: After training, the model is asked to predict the label for a new sentence — and it uses its supervised learning to respond.

**2) Teachable Machine**

**Target Audience:**

- Students, hobbyists, and beginners interested in building AI projects quickly.
- Teachers and presenters who want to demonstrate AI concepts in an interactive way.
- Content creators who want to integrate AI into web or app projects without writing code.

**Use by Target Audience:**

- Users can train models using images, sounds, or poses by providing multiple labeled examples.
- The tool then creates a machine learning model based on these examples.
- Users can test the model live (e.g., showing their face or speaking) and download/export the model for use in apps, websites, or games.

**Benefits**:

- No coding needed — just click, record, and train.
- Great for visual learners — everything happens in real-time and is easy to understand.
- Supports exporting to TensorFlow.js, Unity, or other platforms for real projects.
- Works well for quick experiments and demos.

**Drawbacks**:

- Limited control over how the model is trained (can't tweak algorithms or settings).
- Performance decreases with too many classes or very complex data.
- Requires webcam, mic, and internet access for most features.

**Predictive or Descriptive Analytic:**

**I will choose predictive analytics and reasons for this are:**

- It predicts what a user is doing or saying based on trained data.
- For example: If the model sees you doing a thumbs-up, it predicts "Like".
- The focus is on forecasting/classifying new data, not just summarizing — hence, Predictive Analytic.

---

**Supervised / Unsupervised / Reinforcement Learning:**

**I will chose Supervised Learning because:**

- The user supplies labeled data (each image or sound is tagged).
- The model learns from these examples to make predictions.
- Example: You label 10 "Hello" gestures and 10 "Stop" gestures — the system uses this to recognize them later. That's supervised learning.

NAME: Bhushan Malpani DIV: D15C ROLL NO: 33

**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.

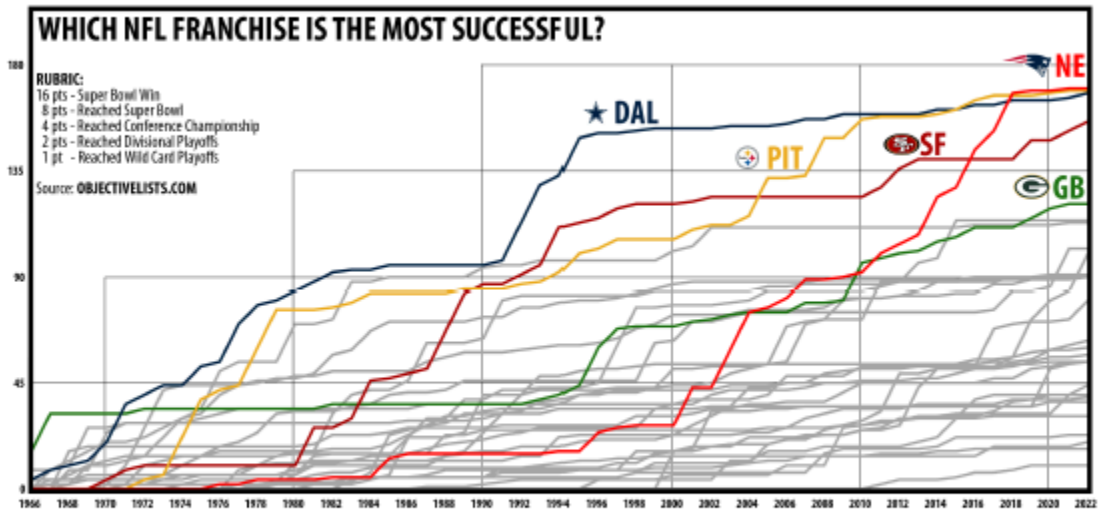



Fig. 8. Contrived metric

Based on the research paper "Misinformative Data Visualizations in the Sports Media Domain" by Drew Scott, I can identify several examples of how data visualizations can mislead readers, specifically in sports media.

One particularly interesting example from the paper is Figure 8, labeled as "Contrived Metric." The author identifies this as a case where a visualization creates a metric without an objective basis, which leads to questionable narratives. This type of misinformation is particularly problematic because it gives the appearance of scientific rigor while actually presenting subjective or arbitrary measurements. The paper categorizes this under "Lie with Statistics" in the Input stage of visualization, and notes that this was one of the most common issues found in their corpus (14 instances).

NAME: Bhushan Malpani DIV: D15C ROLL NO: 33

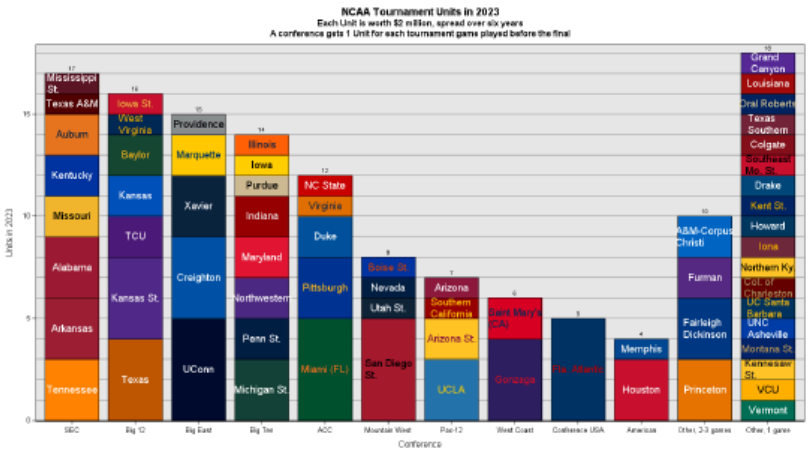



Fig. 3. Stacked bar chart for college basketball conference data

This relates to a current event from March 2023, when ESPN released a "Championship Leverage Index" during the NCAA basketball tournament that purported to show which teams had the most favorable paths to winning. The visualization used a complex formula combining multiple metrics that hadn't been validated, and presented the results as objective analysis. Several sports analysts, including those at The Athletic, criticized the visualization for creating a seemingly scientific metric that actually incorporated significant subjective weighting, leading fans to misunderstand team strengths. The contrived nature of the metric wasn't adequately explained, yet the presentation with precise decimal values and professional graphics gave it an air of authority.

The visualization failed by creating what Scott would categorize as a "Contrived Metric" - a transformation applied to otherwise good data without an apparent or well-explained objective basis. This is particularly misleading because casual viewers typically trust data presented in visual form, especially when it comes from established media outlets. Without proper explanation of methodology or limitations, such visualizations can significantly shape public perception based on questionable analytical foundations.

The paper effectively demonstrates that the sports media domain, despite being considered more "lighthearted" than domains like public health or politics, still exhibits numerous examples of misinformative visualizations that can significantly impact public understanding of the subject matter.

**Cite as:** Drew Scott. Misinformative Data Visualizations in the Sports Media Domain. *TechRxiv.* April 03, 2024.

DOI: 10.36227/techrxiv.171216651.10279711/v1

NAME: Bhushan Malpani DIV: D15C ROLL NO: 33

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

Pima Indians Diabetes Database

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: Bhushan Malpani DIV: D15C ROLL NO: 33

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
```

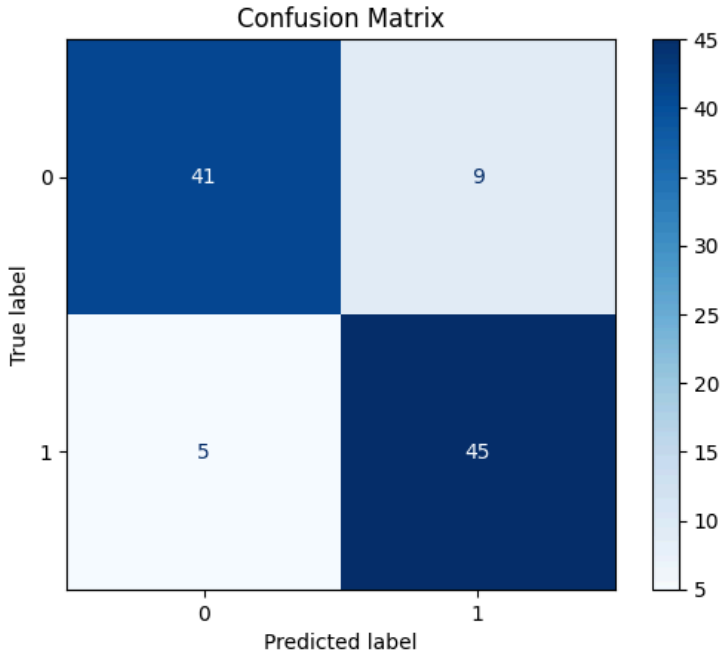
In this step, we are training an SVM (Support Vector Machine) model to classify the data.
We use GridSearchCV to test different combinations of parameters like C, kernel, and gamma.
It finds the best combination that gives the highest accuracy through cross-validation.
This helps us choose the most effective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

NAME: Bhushan Malpani DIV: D15C ROLL NO: 33

Step 6: Evaluate on Test Data

```python
# 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
```



NAME: Bhushan Malpani DIV: D15C ROLL NO: 33

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

Dataset Description:

This dataset is focused on heart disease prediction, where several features (patient demographics, clinical measures, and medical history) are used to predict various health indicators related to the heart. Let's break down the columns:

1. Unnamed: 0:
    - A non-essential index column (likely used for row identification).
2. Age:
    - The age of the patient (numeric).
    - Age plays a significant role in the risk of heart disease.
3. Sex:
    - Gender of the patient (binary: 0 for female, 1 for male).
4. ChestPain:
    - Type of chest pain experienced by the patient (categorical).
    - Values like "typical", "asymptomatic", "nonanginal", and "nontypical" are used to classify the type of chest pain, which can be a symptom of heart disease.
5. RestBP (Resting Blood Pressure):
    - Resting blood pressure (mm Hg).
    - Blood pressure is a key indicator of heart disease risk.
6. Chol (Cholesterol):
    - Serum cholesterol level in mg/dl.
    - High cholesterol is often associated with heart disease.
7. Fbs (Fasting Blood Sugar):
    - Fasting blood sugar (binary: 0 if < 120 mg/dl, 1 if ≥ 120 mg/dl).
    - High fasting blood sugar may indicate a higher risk of heart disease.
8. RestECG (Resting Electrocardiographic Results):
    - Results of the resting electrocardiogram (categorical).
    - Possible values could indicate normal, ST-T wave abnormality, or left ventricular hypertrophy.
9. MaxHR (Maximum Heart Rate):
    - Maximum heart rate achieved during exercise (numeric).
    - A lower maximum heart rate can indicate poor cardiovascular fitness.
10. ExAng (Exercise Induced Angina):
    - Whether or not exercise induced angina (chest pain) occurred (binary: 0 or 1).
    - Induces a measure of the patient's ability to exercise without chest pain.
11. Oldpeak:
    - Depression of the ST segment in the electrocardiogram during exercise (numeric).
    - It's used to evaluate heart ischemia, which indicates potential coronary artery disease.

12. Slope:
    - The slope of the peak exercise ST segment (categorical).
    - This is related to the degree of heart disease, showing if the heart rate response is abnormal during exercise.
13. Ca (Number of Major Vessels Colored by Fluoroscopy):
    - The number of major blood vessels (0 to 3) that have been colored by fluoroscopy during an angiogram (numeric).
    - Used as a measure of coronary artery disease severity.
14. Thal:
    - A blood disorder associated with the heart (categorical).
    - Values like "fixed", "normal", or "reversible" could indicate various stages of the heart disease or the condition of the coronary arteries.
15. AHD (Heart Disease):
    - Whether the patient has heart disease (binary: "Yes" or "No").
    - The target variable for classification, indicating if the patient has a heart condition.

Step 1: 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, GridSearchCV
from sklearn.linear_model import Ridge, LinearRegression
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
```

Step 2: Load the dataset

```
[17] import pandas as pd

    # Load dataset
    df = pd.read_csv("/content/Heart.csv")

    # Drop index column if present
    df.drop(columns=['Unnamed: 0'], errors='ignore', inplace=True)

    # Convert categorical to numerical for correlation analysis
    df_encoded = pd.get_dummies(df, drop_first=True)

    # Display head
    df_encoded.head()
```

| | Age | Sex | RestBP | Chol | Fbs | RestECG | MaxHR | ExAng | Oldpeak | Slope | Ca | ChestPain_nonanginal | ChestPain_nontypical | ChestPain_typical | Thal_normal | Thal_reversable | AHD_Yes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 63 | 1 | 145 | 233 | 1 | 2 | 150 | 0 | 2.3 | 3 | 0.0 | False | False | True | False | False | False |
| 1 | 67 | 1 | 160 | 286 | 0 | 2 | 108 | 1 | 1.5 | 2 | 3.0 | False | False | False | True | False | True |
| 2 | 67 | 1 | 120 | 229 | 0 | 2 | 129 | 1 | 2.6 | 2 | 2.0 | False | False | False | False | True | True |
| 3 | 37 | 1 | 130 | 250 | 0 | 0 | 187 | 0 | 3.5 | 3 | 0.0 | True | False | False | True | False | False |
| 4 | 41 | 0 | 130 | 204 | 0 | 2 | 172 | 0 | 1.4 | 1 | 0.0 | False | True | False | True | False | False |

NAME: Bhushan Malpani DIV: D15C ROLL NO: 33

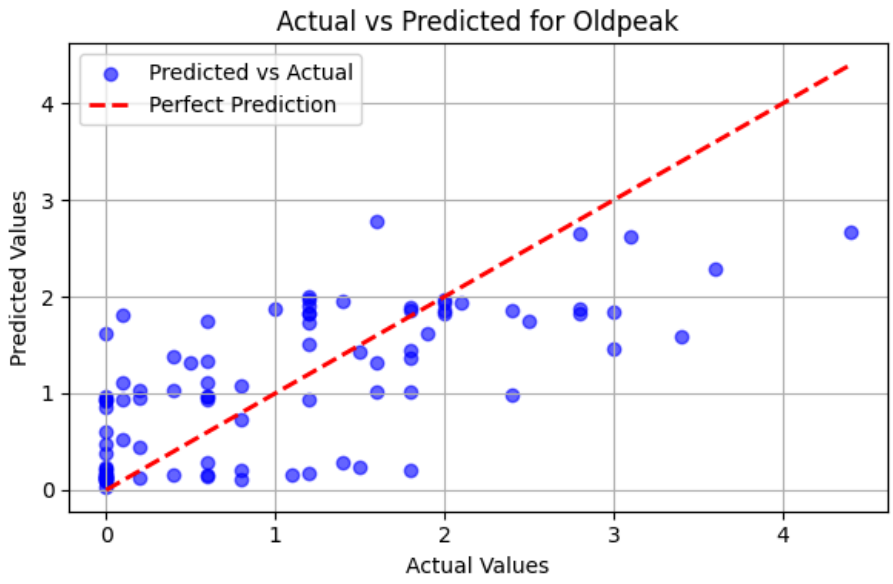
Step 3: Define the Linear Regression Model using OOP

```
# Model definition from previous step
class LinearModelForTarget:
    def __init__(self, target, top_features):
        self.target = target
        self.features = top_features
        self.model = LinearRegression()

    def train_and_evaluate(self, data):
        X = data[self.features]
        y = data[self.target]

        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

        self.model.fit(X_train, y_train)
        y_pred = self.model.predict(X_test)

        r2 = r2_score(y_test, y_pred)
        n, p = X_test.shape
        adj_r2 = 1 - (1 - r2) * (n - 1)/(n - p - 1)
        mae = mean_absolute_error(y_test, y_pred)
        rmse = np.sqrt(mean_squared_error(y_test, y_pred))

        print(f"\nMetrics for {self.target}:")
        print(f"R2 Score: {r2:.4f}")
        print(f"Adjusted R2: {adj_r2:.4f}")
        print(f"MAE: {mae:.4f}")
        print(f"RMSE: {rmse:.4f}")
```

Step 4: Train the model and evaluate the model then plot the predictions:

```
feature_sets = {
    'Oldpeak': ['Slope', 'AHD_Yes', 'MaxHR', 'Thal_normal']
}

import matplotlib.pyplot as plt
import numpy as np

# Step 8: Train and evaluate each target using its best feature set and plot the predictions
for target, features in feature_sets.items():
    print(f"\n📊 Training model for {target} with features: {features}")
    model = LinearModelForTarget(target, features)
    model.train_and_evaluate(df_encoded)

    # Predictions for plotting
    X = df_encoded[features]
    y = df_encoded[target]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
    y_pred = model.model.predict(X_test)

    # Plot Actual vs Predicted
    plt.figure(figsize=(6, 4))
    plt.scatter(y_test, y_pred, color='blue', alpha=0.6, label='Predicted vs Actual')
    plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], 'r--', lw=2, label='Perfect Prediction')
    plt.xlabel("Actual Values")
    plt.ylabel("Predicted Values")
    plt.title(f"Actual vs Predicted for {target}")
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

```
Training model for Oldpeak with features: ['Slope', 'AHD_Yes', 'MaxHR', 'Thal_normal']

Metrics for Oldpeak:
R2 Score: 0.4746
Adjusted R2: 0.4501
MAE: 0.5949
RMSE: 0.7489
```




**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.

Dataset: http://kaggle.com/datasets/yasserh/wine-quality-dataset

Key Features of the Wine Quality Dataset

1. Fixed Acidity
   - Represents non-volatile acids like tartaric acid.
   - Importance: Impacts wine's freshness and stability. Too much or too little affects taste.
2. Volatile Acidity
   - Mainly acetic acid (vinegar-like smell).
   - Importance: High levels lead to unpleasant aroma, reducing quality.
3. Citric Acid
   - Found naturally in wine, adds freshness and flavor.
   - Importance: Enhances flavor, contributes to acidity balance.
4. Residual Sugar
   - Sugar left after fermentation.

    - Importance: Affects sweetness; balance is key for quality perception.
5. Chlorides
    - Amount of salt in wine.
    - Importance: High levels negatively affect taste; small amounts may enhance flavor.
6. Free Sulfur Dioxide
    - Prevents microbial growth and oxidation.
    - Importance: Crucial for wine preservation but must be balanced.
7. Total Sulfur Dioxide
    - Includes both free and bound $SO_2$.
    - Importance: Excess leads to a pungent smell; impacts shelf life and safety.
8. Density
    - Related to sugar and alcohol content.
    - Importance: Indicates fermentation status and body of wine.
9. pH
    - Measures acidity or basicity.
    - Importance: Affects stability, color, and taste.
10. Sulphates
    - Antimicrobial and antioxidant.
    - Importance: Contributes to $SO_2$ levels; affects preservation and flavor.
11. Alcohol
    - Ethanol content in wine.
    - Importance: Strongly correlated with quality. Higher alcohol often perceived as higher quality.

Handling Missing Data During Feature Engineering

1. Detection:
    - Used df.isnull().sum() in Pandas to check missing values.
2. Imputation Techniques Used:
    - Numerical columns (e.g., alcohol, pH):
        - Used mean or median imputation depending on skewness.
        - For example:
          df['alcohol'].fillna(df['alcohol'].mean(), inplace=True)

1. Mean Imputation

Advantages:

- Simple and fast to implement.
- Maintains dataset size.
  Works well with symmetric, normally distributed data.

Disadvantages:

- Affected by outliers.
- Reduces data variability.
- Can introduce bias in skewed distributions.

2. Median Imputation

Advantages:

- Robust to outliers.
- Better suited for skewed data.
- Preserves central tendency better than mean for non-normal data.

Disadvantages:

- Still doesn't account for correlation between features.
- Less effective for normally distributed data.

3. Mode Imputation (for categorical or discrete data)

Advantages:

- Simple to apply.
- Maintains most frequent category.

Disadvantages:

- Not suitable for continuous data.
- Can cause overrepresentation of the mode.

4. Dropping Missing Rows

Advantages:

- Very simple to apply.
- Avoids introducing potential bias from imputation.

Disadvantages:

- Leads to data loss.
- Not suitable if many rows have missing values.
- May reduce model performance due to smaller training set.