1. The pandas library provides functions like read_csv(), read_excel(), or read_sql() to read data from different file formats or databases into a Pandas DataFrame

Importing of the dataset

```
[2] import pandas as pd
    df = pd.read_csv("/content/market_data.csv")
```

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	Outlet_Type	Item_Out
0	FDA15	9.300	Low Fat	0.016047	Dairy	249.8092	OUT049	1999	Medium	Tier 1	Supermarket Type1	
1	DRC01	5.920	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009	Medium	Tier 3	Supermarket Type2	
2	FDN15	17.500	Low Fat	0.016760	Meat	141.6180	OUT049	1999	Medium	Tier 1	Supermarket Type1	
3	FDX07	19.200	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010	1998	NaN	Tier 3	Grocery Store	
4	NCD19	8.930	Low Fat	0.000000	Household	53.8614	OUT013	1987	High	Tier 3	Supermarket Type1	

2. You can use the drop() method to eliminate columns that are not useful.

Dropping unuseful columns

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

```
[ ] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8523 entries, 0 to 8522
Data columns (total 11 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Item_Identifier       8523 non-null   object
 1   Item_Weight           7060 non-null   float64
 2   Item_Fat_Content      8523 non-null   object
 3   Item_Visibility       8523 non-null   float64
 4   Item_Type             8523 non-null   object
 5   Item_MRP              8523 non-null   float64
 6   Outlet_Identifier     8523 non-null   object
 7   Outlet_Size           6113 non-null   object
 8   Outlet_Location_Type  8523 non-null   object
 9   Outlet_Type           8523 non-null   object
 10  Item_Outlet_Sales     8523 non-null   float64
dtypes: float64(4), object(7)
memory usage: 732.6+ KB
```

3. You can identify and drop rows with a significant number of missing values using the dropna() method and as well taking the mean, median of the whole data and filling it in that place

Taking Care of Missing Data

```
[3] df['Item_Weight'] = df['Item_Weight'].fillna(df['Item_Weight'].mean())
    df
```

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	Outlet_Type	Item_Outlet_Sales
0	FDA15	9.300	Low Fat	0.016047	Dairy	249.8092	OUT049	1999	Medium	Tier 1	Supermarket Type1	3735.1380
1	DRC01	5.920	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009	Medium	Tier 3	Supermarket Type2	443.4228
2	FDN15	17.500	Low Fat	0.016760	Meat	141.6180	OUT049	1999	Medium	Tier 1	Supermarket Type1	2097.2700
3	FDX07	19.200	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010	1998	NaN	Tier 3	Grocery Store	732.3800
4	NCD19	8.930	Low Fat	0.000000	Household	53.8614	OUT013	1987	High	Tier 3	Supermarket Type1	994.7052

4.. **Find out outliers (manually)**:

In the case of Item_Outlet_Sales calculating the outlier using the IQR method

First Quartile (Q1) - Formula: =QUARTILE(H2:H6113, 1)
Q1 = 974.73
Third Quartile (Q3) - Formula: =QUARTILE(H2:H6113, 3)
Q3 = 3271.08
Interquartile Range (IQR) = Q3 - Q1
IQR = 3271.08 - 974.73 = 2296.34

<u>Outlier Boundaries:</u>
Lower Bound = Q1 - 1.5 * IQR = 974.73 - (1.5 * 2296.34) = **- 2469.79**
Upper Bound = Q3 + 1.5 * IQR = 3271.08 + (1.5 * 2296.34) = **6715.59**

Now with this we can say the values that lie below **- 2469.79** and **6715.59** are considered as outliers and in whole dataset there are 135 outliers for that feature

95	DRG23	8.88	Low Fat	0.086709	Hard Drink	151.7682	OUT013	High	Tier 3	Supermark	762.341	FALSE
96	FDY25	12.85765	Low Fat	0.03381	Canned	180.5976	OUT027	Medium	Tier 3	Supermark	7968.294	TRUE
97	NCH54	13.5	Low Fat	0.072669	Household	160.292	OUT046	Small	Tier 1	Supermark	1438.128	FALSE
98	NCR53	12.85765	Low Fat	0.144338	Health anc	224.4404	OUT027	Medium	Tier 3	Supermark	6976.252	TRUE
99	FDG02	7.855	Low Fat	0.011307	Canned	188.5188	OUT018	Medium	Tier 3	Supermark	952.094	FALSE
100	NCO26	7.235	Low Fat	0.076856	Household	116.0492	OUT046	Small	Tier 1	Supermark	1969.436	FALSE
101	FDH35	12.85765	Low Fat	0.059957	Starchy Fo	165.4526	OUT027	Medium	Tier 3	Supermark	5262.483	FALSE
102	NCC31	8.02	Low Fat	0.019867	Household	154.5972	OUT035	Small	Tier 2	Supermark	1402.175	FALSE
103	NCC30	16.6	Low Fat	0.027622	Household	177.6344	OUT049	Medium	Tier 1	Supermark	2676.516	FALSE
104	FDI16	14	Regular	0	Frozen Foc	53.064	OUT035	Small	Tier 2	Supermark	905.488	FALSE
105	FDP16	18.6	Low Fat	0.039356	Frozen Foc	246.3802	OUT049	Medium	Tier 1	Supermark	7370.406	TRUE

5. **Standardization** converts data to **z-scores** using the mean and standard deviation, showing how many standard deviations a value is from the mean.
Normalization scales data between **0 and 1** using min-max scaling, making different features comparable.

Standardization and Normalization of the data

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


numerical_columns = ['Item_Weight', 'Item_Visibility', 'Item_MRP', 'Item_Outlet_Sales']

# Initialize scalers
standard_scaler = StandardScaler()
minmax_scaler = MinMaxScaler()

# Standardization of the data
standardized_data = standard_scaler.fit_transform(df[numerical_columns])
df_standardized = pd.DataFrame(standardized_data, columns=numerical_columns)

# Normalization of the data
normalized_data = minmax_scaler.fit_transform(df[numerical_columns])
df_normalized = pd.DataFrame(normalized_data, columns=numerical_columns)


print("Standardized Data:")
print(df_standardized.head())

print("\nNormalized Data:")
print(df_normalized.head())
```

```
Standardized Data:
   Item_Weight  Item_Visibility  Item_MRP  Item_Outlet_Sales
0    -0.881033        -0.967450  1.744524           0.811077
1    -1.710793        -0.902945 -1.494387          -1.079139
2     1.131996        -0.953220  0.005804          -0.129443
3    -0.971864        -1.287832 -1.404516          -0.762573
4    -0.612220        -1.287832 -1.444060          -1.014143

Normalized Data:
   Item_Weight  Item_Visibility  Item_MRP  Item_Outlet_Sales
0     0.282525         0.048866  0.927507           0.283550
1     0.081274         0.058705  0.072068           0.031370
2     0.770765         0.051037  0.468288           0.158072
3     0.260494         0.000000  0.095805           0.073604
4     0.347723         0.000000  0.085361           0.040041
```

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

Theory:
Data visualization and exploratory data analysis (EDA) using Matplotlib and Seaborn help uncover patterns, trends, and relationships within data. Matplotlib is a flexible, low-level library for creating static, animated, and interactive plots, while Seaborn is built on top of Matplotlib and provides a high-level interface for visually appealing statistical graphics. EDA involves techniques like histograms, scatter plots, box plots, and heatmaps to understand data distributions, detect outliers, and identify correlations

Matplotlib :
It is a powerful and versatile Python library for data visualization, built around a hierarchical structure of Figures and Axes. It provides a comprehensive set of plotting functions, including line plots, scatter plots, bar charts, and histograms. Users can extensively customize plots by adjusting colors, linestyles, markers, and grid lines, as well as incorporating LaTeX-style expressions for labels and titles.

Seaborn :
It is built on top of Matplotlib, is a specialized statistical visualization library that enhances both aesthetics and usability. It provides built-in themes, color palettes, and high-level abstractions for statistical plots like box plots, violin plots, and scatter plots with regression lines.
Seaborn excels in creating complex visualizations such as pair plots, heatmaps, and cluster maps, making it particularly useful for data analysis and pattern recognition. While Matplotlib offers low-level control for general-purpose plotting, Seaborn simplifies statistical data visualization, making it an essential tool for analysts and researchers aiming to extract insights from data efficiently.

1. **Bar graph and contingency table using any two features:**
   a. Distribution of item type.
      A bar graph is a visual representation of categorical data where each category is represented by a bar. You can use the plt.bar function in matplotlib or sns.barplot in seaborn to create a bar graph.

The bar graph shows that **Fruits and Vegetables** and **Snack Foods** generate the highest total sales, while **Seafood** has the least.

b. Contingency table

```
contingency_table = pd.crosstab(df["Item_Type"], df["Outlet_Type_Supermarket Type1"])
print(contingency_table)
```

```
Outlet_Type_Supermarket Type1    0    1
Item_Type
Baking Goods                   222  426
Breads                          91  160
Breakfast                       42   68
Canned                         223  426
Dairy                          232  450
Frozen Foods                   284  572
Fruits and Vegetables          427  805
Hard Drinks                     69  145
Health and Hygiene             185  335
Household                      313  597
Meat                           168  257
Others                          62  107
Seafood                         24   40
Snack Foods                    415  784
Soft Drinks                    145  300
Starchy Foods                   44  104
```

Supermarket Type 1 serves as a **one-stop shop for daily essentials**, whereas the other outlet type likely caters to a **more limited or specialized consumer base**, possibly smaller grocery stores

2. **Scatter plot**

The scatter plot of Item MRP vs. Item Outlet Sales shows a positive trend, indicating that as the Item MRP increases, the Item Outlet Sales also tend to increase, although with some variability.Additionally, the spread of sales increases with MRP, meaning higher-priced items have a wider range of sales figures, possibly due to variations in demand across different products.

3. **Box plot**

This plot shows that median sales are fairly similar across different outlet sizes, with Medium outlets having slightly higher median sales. The spread of sales is also comparable across all outlet sizes, with many outliers indicating high sales in certain cases. We can conclude that outlet size alone may not be a strong determinant of sales, and other factors like location, promotions, and product variety may play a more significant role

4. **Heat map**

```
plt.figure(figsize=(10, 7))
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap="inferno", fmt=".2f", linewidths=0.5)
plt.title("Heatmap of Feature Correlations")
plt.show()
```

Strong correlation between **Item MRP and Item Outlet Sales**, meaning expensive items often have higher sales. This suggests that higher-priced items contribute significantly to total revenue, possibly due to factors such as better quality, brand reputation, or consumer preference for premium products

5. a. **Histogram**

A histogram is a graphical representation of the distribution of a continuous variable. It divides the range of values into bins and shows the frequency or count of observations within each bin. Histograms provide insights into the shape and spread of the data distribution.

```
plt.figure(figsize=(10, 6))
sns.histplot(df["Item_Outlet_Sales"], bins=30, kde=False, color='red')
plt.xlabel("Item Outlet Sales")
plt.ylabel("Frequency")
plt.title("Histogram of Item Outlet Sales")
plt.show()
```

b. **Normalized Histogram**

A normalized histogram represents the relative frequencies or proportions of observations in each bin, providing a normalized view of the distribution. It is useful for comparing distributions of variables with different scales or sample sizes.

```
plt.figure(figsize=(10, 6))
sns.histplot(df["Item_Outlet_Sales"], bins=30, kde=True, stat="density", color='red')
plt.xlabel("Item Outlet Sales")
plt.ylabel("Density")
plt.title("Normalized Histogram of Item Outlet Sales")
plt.show()
```

Key observations:

1. The **histogram** shows that sales are heavily skewed to the right, meaning most sales values are low, with fewer items generating very high sales.
2. The **normalized histogram** (with KDE) confirms this trend, showing a high density at lower sales values.

6. **Handle outlier using box plot and Inter quartile range**

```
plt.figure(figsize=(8, 6))
sns.boxplot(y=df["Item_Outlet_Sales"])
plt.title("Before Removing Outliers")
plt.show()

# Calculating IQR
Q1 = df["Item_Outlet_Sales"].quantile(0.25)
Q3 = df["Item_Outlet_Sales"].quantile(0.75)
IQR = Q3 - Q1

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

# Removing outliers
df_cleaned = df[(df["Item_Outlet_Sales"] >= lower_bound) & (df["Item_Outlet_Sales"] <= upper_bound)]
plt.figure(figsize=(8, 6))
sns.boxplot(y=df_cleaned["Item_Outlet_Sales"])
plt.title("After Removing Outliers")
plt.show()
```

In the first box plot, a large number of extreme outliers are observed, particularly above **6000** in Item Outlet Sales. These extreme values suggest that while most sales remain within a typical range, certain items experience significantly higher sales, possibly due to promotions, high demand, or premium pricing. **After applying the Interquartile Range (IQR) method**, these extreme outliers are removed, resulting in a more balanced dataset.

Conclusion:
In the end, gained valuable insights into sales trends, pricing impact, and outlet characteristics. The bar graph highlighted that Fruits and Vegetables, along with Snack Foods, contribute the most to total sales, while Seafood sees the least. The contingency table revealed that Supermarket Type 1 serves as a primary shopping destination, while other outlets likely cater to

niche markets.Overall, this analysis provided a clearer understanding of sales distribution, pricing effects, and outlet performance

Aim: Perform data Data Modeling

Theory:

1. **Data Partitioning**

Data partitioning divides large datasets into smaller parts (partitions) to improve performance and scalability. Common types include:

1. List Partitioning: Divides data based on specific column values (e.g., customers from different countries).
2. Hash Partitioning: Uses a hash function on a column to distribute data evenly, useful when no natural partition exists.
3. Hybrid Partitioning: Uses both horizontal and vertical partitioning to split data efficiently

The choice depends on data size, access patterns, and system requirements.

2. **Hypothesis Testing**

Hypothesis testing is a statistical method used to make inferences or draw conclusions about the population based on a sample of data. It is a way to test the validity of a claim or idea, often referred to as a hypothesis, about a population parameter:

- Null Hypothesis (H0): Assumes no difference or effect (e.g., "Men and women are of the same height on average").
- Alternative Hypothesis (Ha): Suggests a difference exists (e.g., "Men are taller than women").
- P-Value: Probability of observing the data if H0 is true. A low p-value (< 0.05) suggests rejecting H0.
- Significance Level (α): The threshold below which the null hypothesis is rejected. Common choices are 0.01, 0.05, and 0.1. If the p-value is less than the significance level, the null hypothesis is rejected.
- One-Tailed & Two-Tailed Tests: One-tailed tests are used when the direction of the difference is known, such as when testing if a new drug is better than a placebo. Two-tailed tests are used when the direction of the difference is unknown, such as when testing if a new drug has a different effect than the current standard.
- Formula :

$$z = \frac{(\bar{x}_1 - \bar{x}_2) - (\mu_1 - \mu_2)}{\sqrt{\left(\sigma_1^2 / n_1 + \sigma_2^2 / n_2\right)}}$$

Steps:

1. Define H0 and Ha.

2. Choose α (e.g., 0.05).
3. Calculate test statistics and p-value.
4. Compare p-value with α and decide.
5. Interpret results:

Type I error occurs when the null hypothesis is rejected when it is actually true, leading to a false positive. A Type II error occurs when the null hypothesis is not rejected when it is actually false, leading to a false negative.

Output :

1. Load the dataset

```
import pandas as pd
df = pd.read_csv("/content/market_data.csv")
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8523 entries, 0 to 8522
Data columns (total 14 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Item_Identifier            8523 non-null   object
 1   Item_Weight                8523 non-null   float64
 2   Item_Fat_Content           8523 non-null   object
 3   Item_Visibility            8523 non-null   float64
 4   Item_Type                  8523 non-null   object
 5   Item_MRP                   8523 non-null   float64
 6   Outlet_Identifier          8523 non-null   object
 7   Outlet_Size                8523 non-null   object
 8   Outlet_Location_Type       8523 non-null   object
 9   Outlet_Type                8523 non-null   object
 10  Item_Outlet_Sales          8523 non-null   float64
 11  Sales_Bin                  8523 non-null   object
 12  MRP_Bin                    8523 non-null   object
 13  Item_Outlet_Sales_Capped   8523 non-null   float64
dtypes: float64(5), object(9)
memory usage: 932.3+ KB
```

2. For the given feature Item_MRP calculating the counts

```
df['Item_MRP'].value_counts()
```

Item_MRP	count
142.0154	6
172.0422	6
113.2834	5
188.1872	5
261.2910	5
...	...
219.5482	1
57.3930	1
105.8622	1
101.2990	1
75.4670	1

4694 rows × 1 columns

dtype: int64

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

```
[5] import matplotlib.pyplot as plt
    import seaborn as sns
    import numpy as np
    from scipy.stats import norm
    from sklearn.model_selection import train_test_split
```

```
[6] train_df, test_df = train_test_split(df, test_size=0.25, random_state=50)

    train_size = len(train_df)
    test_size = len(test_df)

    plt.figure(figsize=(8, 5))
    sns.barplot(x=['Train Set', 'Test Set'], y=[train_size, test_size], palette=['green', 'blue'])
    plt.xlabel('Dataset')
    plt.ylabel('Number of Records')
    plt.title('Train-Test Split Proportion')
    plt.show()
```

4. Visualization using a bar graph to confirm the proportions of the data split into training and test sets

5. Total number of records in the split

```
Total records in Training Set: 4584
Total records in Testing Set: 1529
```

6. Now using a two-sample Z-test to validate whether the split was biased or unbiased

```
def sample_test(sample1, sample2):
  mean1, mean2 = np.mean(sample1), np.mean(sample2)
  std1, std2 = np.std(sample1, ddof=1), np.std(sample2, ddof=1)
  n1, n2 = len(sample1), len(sample2)

  z_score = (mean1 - mean2) / np.sqrt((std1**2 / n1) + (std2**2 / n2))

  p_value = 2 * (1 - norm.cdf(abs(z_score)))
  return z_score, p_value
```

7. Now this test is performed for the comparison of **Item_Mrp** column in training and testing dataset. The Z-statistic was calculated based on the means, standard deviations, and sizes of both samples. If p-value < significance level (α=0.05), we reject the null hypothesis.

```
Z-score: -0.26839427745723915
P-value: 0.7883958439089263
No significant difference found (p > 0.05). The partitioning is valid.
```

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

Conclusion:
A two-sample z-test to verify whether the partitioning of the Item_MRP (Maximum Retail Price) column into training and testing sets was statistically balanced. The results showed a p-value greater than 0.05, indicating that there is no significant difference in the mean MRP between the two partitions. This confirms that the data splitting process maintains a consistent distribution, ensuring that the model is trained and evaluated on representative data without any bias due to MRP variations

AIDS_Exp4 **Avan Shetty** **D15C-52**

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

Problem Statement: Perform the following Tests:Correlation Tests:
a) Pearson's Correlation Coefficient
b) Spearman's Rank Correlation
c) Kendall's Rank Correlation
d) Chi-Squared Test

Theory and Output:
1. Data Overview:
The dataset consists of various attributes related to retail products, including product details, pricing, and sales figures. It contains categorical columns such as `**Item_Type**`, `**Outlet_Identifier**`, and `O**utlet_Location_Type**`, which describe product categories and store locations. Numerical attributes include `**Item_Weight**`, `**Item_MRP**`, and `**Item_Outlet_Sales**`, which represent product weight, pricing, and sales revenue. The dataset is useful for analyzing relationships between pricing, sales performance, and product characteristics. Missing values in certain columns may require handling for accurate analysis.

```
categorical_cols = df.select_dtypes(include=['object', 'bool', 'category']).columns.tolist()
numerical_cols = df.select_dtypes(include=['number']).columns.tolist()


print("Categorical Columns:", categorical_cols)
print("Numerical Columns:", numerical_cols)
```

```
Categorical Columns:
 Item_Identifier
Item_Fat_Content
Item_Type
Outlet_Identifier
Outlet_Size
Outlet_Location_Type
Outlet_Type
Sales_Bin
MRP_Bin

Numerical Columns:
 Item_Weight
Item_Visibility
Item_MRP
Item_Outlet_Sales
Item_Outlet_Sales_Capped
```

2. Understanding what is correlation tests:
Correlation tests measure the strength and direction of the relationship between two variables. They help determine how changes in one variable are associated with changes in another.

3. **Pearson Correlation Coefficient**

Pearson's correlation coefficient, also known as Pearson's (r) or simply correlation coefficient, measures the linear relationship between two continuous variables.

The coefficient assumes that the variables are normally distributed and have a linear relationship.

Formula :

$$r = \frac{\sum_{i=1}^{n}\left(\left(x_i - \bar{x}\right)\left(y_i - \bar{y}\right)\right)}{\sqrt{\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2 \sum_{i=1}^{n}\left(y_i - \bar{y}\right)^2}}$$

Indications:

1. Numerator (Covariance Term):
   - This calculates the sum of the product of deviations of x and y from their respective means.
   - This represents the covariance between x and y showing how they vary together.
2. Denominator (Standard Deviation Term):
   - The first square root term represents the standard deviation of x
   - The second square root term represents the standard deviation of y
   - This ensures normalization, making r independent of the scale of the variables.

- r > 0 → Positive correlation
- r < 0 → Negative correlation
- r = 0 → No linear correlation.

3. The **p-value** in Pearson's correlation indicates statistical significance:
   - **p < 0.05** → Significant correlation, unlikely due to chance.
   - **p > 0.05** → Not significant, insufficient evidence of correlation.

   A **p-value of 0** suggests a very strong relationship, often due to a large dataset, confirming the correlation is highly unlikely to be random

Manual Method:

```
import numpy as np

col1, col2 = 'Item_MRP', 'Item_Outlet_Sales'
data = df[[col1, col2]].dropna()  #if Nan values exist then for that purpose

# Pearson's Correlation Coefficient (Manual)
def pearson_manual(x, y):
    mean_x, mean_y = np.mean(x), np.mean(y)

    # Covariance calculation
    covariance = np.sum((x - mean_x) * (y - mean_y))

    # Sd calculations for denominator
    std_x = np.sqrt(np.sum((x - mean_x) ** 2))
    std_y = np.sqrt(np.sum((y - mean_y) ** 2))

    correlation = covariance / (std_x * std_y)
    return correlation

result = pearson_manual(data[col1], data[col2])
print(f"Pearson's Correlation (Manually) between {col1} and {col2}: {result}")
```

```
Pearson's Correlation (Manually) between Item_MRP and Item_Outlet_Sales: 0.5675744466569193
```

Library Function:

```
#Library method
import pandas as pd
from scipy.stats import pearsonr

col1, col2 = 'Item_MRP', 'Item_Outlet_Sales'
data = df[[col1, col2]].dropna()

pearson_library_result, pearson_p_value = pearsonr(data[col1], data[col2])
print(f"Pearson's Correlation (Library) between {col1} and {col2}: {pearson_library_result}")

print(f"Pearson's Correlation (Library) between {col1} and {col2}: p-value: {pearson_p_value:.4f}")
```

```
Pearson's Correlation (Library) between Item_MRP and Item_Outlet_Sales: 0.5675744466569191
Pearson's Correlation (Library) between Item_MRP and Item_Outlet_Sales:  p-value: 0.0000
```

Analysis:

The Pearson correlation coefficient was calculated using both a manual approach and scipy.stats.pearsonr(). The manual method involved computing the means, covariance, and standard deviations, yielding a correlation of **0.5676**. The library-based approach, after handling missing values, returned the same result.
Since both methods produced identical values, this confirms the correctness of the manual calculation. The **moderate positive correlation** (0.5676) indicates that **higher** Item MRP is generally associated with higher Item Outlet Sales, though other factors may also influence sales.

The **p-value being 0** suggests that the correlation is statistically significant at any common significance level (e.g., 0.05 or 0.01). This means there is very **strong evidence** that the relationship between Item MRP and Item Outlet Sales is not due to random chance

4. **Spearman Rank Correlation :**

Spearman's rank correlation coefficient measures the monotonic relationship between two variables. Unlike Pearson's correlation, it does not assume a linear relationship or normality, making it useful for ordinal and non-normally distributed data.

Formula:
Spearman's correlation is computed using the ranked values of the variables. It is given by:

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

Indications:
1: Convert each dataset into ranks, assigning the average rank in case of ties.
2: Compute (di) the difference between ranks for each pair.
3: Square each (di) and sum all values.
4: Apply the formula to calculate r
5. Interpret the results
- r > 0 → Positive monotonic relationship
- r < 0 → Negative monotonic relationship
- r = 0 → No monotonic relationship

6. The p-value tests the null hypothesis that there is no association between the variables:
- p < 0.05 → Significant correlation, unlikely due to chance.
- p > 0.05 → Not significant, insufficient evidence of correlation.
- A p-value of 0 suggests a highly significant relationship, often due to a large dataset.

Manual Method:

```
col1, col2 = 'Item_MRP', 'Item_Outlet_Sales'
data = df[[col1, col2]].dropna()

# Spearman's Rank Correlation (Manually)
def spearman_manual(x, y):
    rank_x = x.rank(method='average')
    rank_y = y.rank(method='average')

    # Compute differences in ranks
    d = rank_x - rank_y
    d_squared = d ** 2

    # Compute Spearman's correlation
    n = len(x)
    numerator = 6 * np.sum(d_squared)
    denominator = n * (n ** 2 - 1)
    correlation = 1 - (numerator / denominator)

    return correlation

r_value = spearman_manual(data[col1], data[col2])
print(f"Spearman's Correlation (Manual) between {col1} and {col2}: {r_value}")
```

```
Spearman's Correlation (Manual) between Item_MRP and Item_Outlet_Sales: 0.562986519638175
```

Library Function:
Using scipy.stats.spearmanr to calculate the rank for Item_MRP and Item_Oullet_Sales

```
from scipy.stats import spearmanr
col1, col2 = 'Item_MRP', 'Item_Outlet_Sales'
data = df[[col1, col2]].dropna()

spearman_rvalue, p_value = spearmanr(data[col1], data[col2])
print(f"Spearman's Correlation (Library) between {col1} and {col2}: {spearman_rvalue}")
print(f"P-value: {p_value}")
```

```
Spearman's Correlation (Library) between Item_MRP and Item_Outlet_Sales: 0.5629864415335609
P-value: 0.0
```

Analysis:
From code output:

- Manually calculated value: ρ=0.562986519638175
- Library calculated result: ρ=0.5629864415335609

As it was computed manually and using scipy.stats.spearmanr() to analyze the relationship between Item_MRP and Item_Outlet_Sales, the manual method involved ranking values, computing rank differences, and applying the Spearman formula. yielding a correlation of **0.56299**. The library-based method produced the same result, confirming the correctness of the manually calculated result.
The **p-value was 0.0**, indicating that the correlation is statistically significant at any common significance level. This suggests a **strong relationship** between Item MRP and Item Outlet Sales, though other factors may also influence sales

5. **Kendall's Rank Correlation**

Kendall's tau measures the strength and direction of the association between two variables. It is non-parametric and does not assume normality or a linear relationship, making it ideal for **ordinal data** or **nonlinear monotonic relationships**.

Formula :

Kendall's correlation is computed based on the number of concordant and discordant pairs:

$$\tau = \frac{\text{Number of concordant pairs} - \text{Number of discordant pairs}}{n(n-1)/2}$$

Concordant: When the order of values in both variables matches (i.e., if one value increases, the other also increases).
Discordant: When the order of values in one variable is reversed in the other (i.e., if one value increases while the other decreases).

n(n−1)/2 is the total number of unique pairs

<u>Steps for Calculation:</u>
1. Compare all pairs: Count concordant (C) and discordant (D) pairs.
2. Compute tau: Use the formula to calculate Kendall's correlation coefficient.
3. Interpret results:
   - $\tau > 0 \rightarrow$ Positive monotonic relationship
   - $\tau < 0 \rightarrow$ Negative monotonic relationship
   - $\tau = 0 \rightarrow$ No monotonic relationship

Manual Method:

```
def kendall_manual(x, y):
    n = len(x)
    concordant = 0
    discordant = 0

    for i in range(n):
        for j in range(i + 1, n):
            if (x[i] > x[j] and y[i] > y[j]) or (x[i] < x[j] and y[i] < y[j]):
                concordant += 1
            elif (x[i] > x[j] and y[i] < y[j]) or (x[i] < x[j] and y[i] > y[j]):
                discordant += 1

    tau = (concordant - discordant) / ((n * (n - 1)) / 2)
    return tau #here tau represents the rank

# Kendall's using library
tau_value = kendall_manual(data[col1].values, data[col2].values)
print(f"Kendall's Correlation (Manual) between {col1} and {col2}: {tau_value}")
```

```
Kendall's Correlation (Manual) between Item_MRP and Item_Outlet_Sales: 0.40690027341013535
```

Library Function:

```
from scipy.stats import kendalltau

tau_value_lib, p_value = kendalltau(data[col1], data[col2])

print(f"Kendall's Correlation (Library) between {col1} and {col2}: {tau_value_lib}")
print(f"Kendall's Correlation p-value: {p_value}")
```

```
Kendall's Correlation (Library) between Item_MRP and Item_Outlet_Sales: 0.40699111339889754
Kendall's Correlation p-value: 0.0
```

Analysis:
- Manually calculated Kendall's correlation: **0.4069**
- Library-calculated Kendall's correlation: **0.40699**

The Kendall's Tau value of **0.4069** indicates a positive correlation between Item MRP and Item Outlet Sales. This suggests that as Item MRP increases, Item Outlet Sales tend to increase as well. The p-value of 0.0 confirms that the correlation is statistically significant, indicating a reliable relationship, though other factors could also influence sales.

6. **Chi-Squared Test**

AIDS_Exp4 **Avan Shetty** **D15C-52**

The Chi-Squared Test of Independence is used to determine if there is a significant association between two categorical variables. It compares the observed frequency of occurrences in each category with the expected frequency, under the assumption that the variables are independent

Formula:
The Chi-Squared statistic is calculated using the formula:

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

Where:
1. Oi = observed frequency in the i-th category
2. Ei = expected frequency in the i-th category
3. The sum is taken over all categories

Steps for Calculation:
1. Create a Contingency Table:
Organize the observed data into a contingency table (cross-tabulation) showing the frequency of occurrences for each combination of the two categorical variables.
2. Calculate Expected Frequencies:
For each cell in the contingency table, calculate the expected frequency under the assumption that the variables are independent:
3. Compute Chi-Squared Statistic:
Calculate the Chi-Squared statistic using the formula

4. Degrees of Freedom (df):
The degrees of freedom are calculated as: df=(r−1)×(c−1): [r = rows, c = cols]
5. Compare with Critical Value:
The Chi-Squared statistic is compared with a critical value from the Chi-Squared distribution table at the chosen significance level (usually α=0.05) and degrees of freedom. If the statistic is greater than the critical value, we reject the null hypothesis.

Interpret Results:
- $p < 0.05$: Significant relationship, the variables are likely associated.
- $p > 0.05$: Fail to reject the null hypothesis, no significant relationship, the variables are likely independent.

Manual Method:

```
from scipy.stats import chi2
contingency_table = pd.crosstab(df['Item_Type'], df['Item_Fat_Content'])

#Expected frequencies (Ei)
total = contingency_table.sum().sum()
rows = contingency_table.sum(axis=1)
cols = contingency_table.sum(axis=0)

expected = np.outer(rows, cols) / total

#Calculate the Chi-Squared statistic
observed = contingency_table.values  #(Oi)
chi_squared_stat = ((observed - expected) ** 2 / expected).sum()

#(dof) = (rows - 1) * (cols - 1)
df = (contingency_table.shape[0] - 1) * (contingency_table.shape[1] - 1)

alpha = 0.05
critical_value = chi2.ppf(1 - alpha, df)

if chi_squared_stat > critical_value:
    result = 'There is a significant relationship. between Item_Type and Item_Fat_Content'
else:
    result = 'Fail to reject null hypothesis: No significant relationship.'

#final output
print(f"Chi-Squared Statistic: {chi_squared_stat}")
print(f"Critical Value: {critical_value}")
print(result)
```

```
Chi-Squared Statistic: 1555.2321723869677
Critical Value: 79.08194448784874
There is a significant relationship. between Item_Type and Item_Fat_Content
```

Library Function:

```
from scipy.stats import chi2_contingency

contingency_table = pd.crosstab(df['Item_Fat_Content'], df['Item_Type'])

# Apply Chi-Squared Test
chi2_stat, p_value, dof, expected = chi2_contingency(contingency_table)

#Output
print(f"Chi-Squared Statistic: {chi2_stat}")
print(f"P-Value: {p_value}")
print(f"Degrees of Freedom: {dof}")
print(f"Expected Frequencies:\n{expected}")

alpha = 0.05
if p_value < alpha:
    print("There is a significant association between Item_Type and Item_Fat_Content")
else:
    print("Fail to reject the null hypothesis: No significant association.")
```

```
Chi-Squared Statistic: 1555.2321723869677
P-Value: 1.5408537629038592e-285
Degrees of Freedom: 60
Expected Frequencies:
[[ 24.02534319   9.30611287   4.07837616  24.06241934  25.28593218
   31.73718174  45.67781298   7.93429544  19.27959639  33.73929368
   15.75736243   6.26586883   2.3728734   44.49137628  16.49888537
    5.48726974]
 [386.91446674 149.86964684  65.67992491 387.51155696 407.21553444
  511.10923384 735.61515898 127.77730846 310.48691775 543.35210607
  253.76334624 100.90824827  38.21377449 716.50827174 265.70515077
   88.36935351]
 [219.64941922  85.08025343  37.28616684 219.98838437 231.17423442
  290.15417107 417.60506864  72.53854277 176.26187962 308.45828933
  144.06019007  57.28511088  21.6937698  406.75818374 150.83949314
   50.16684266]
 [  8.51531151   3.29836912   1.44550041   8.52845242   8.96210255
   11.24862138  16.1896046    2.81215534   6.83327467  11.95823067
    5.58488795   2.22081427   0.84101842  15.76909539   5.84770621
    1.9448551 ]
 [  8.89545935   3.44561774   1.51003168   8.90918691   9.36219641
   11.75079197  16.9123548    2.93769799   7.13833157  12.49208025
    5.83421331   2.31995776   0.87856389  16.47307286   6.10876452
    2.03167899]]
There is a significant association between Item_Type and Item_Fat_Content
```

Analysis:

The Chi-Squared Statistic calculated manually (1555.23) and using the library function (1555.23) match, confirming the correctness of the calculation. The manual method compared the Chi-Squared Statistic with the critical value of 79.08 (at $\alpha = 0.05$ and 60 degrees of freedom). Since the statistic is much larger than the critical value, we reject the null hypothesis, indicating a significant association between the two categorical variables.

Additionally, the library function provides a p-value of 1.54e-285, which is effectively zero. Since this p-value is far below the significance level of 0.05, it reinforces the rejection of the null hypothesis. Both methods confirm that there is a significant association between the two variables, meaning that '**Item_Fat_Content**' and '**Item_Type**' are not independent, and their distribution is significantly related.

Final Observations :

Test	Relationship between variables
Pearson's Correlation	Moderate positive linear relationship (r = 0.5676)
Spearman's Rank Correlation	Moderate positive monotonic relationship (ρ = 0.56299)
Kendall's Rank Correlation	Moderate positive monotonic relationship (τ = 0.4069)
Chi-Squared Test	Significant association (Chi-Squared Statistic = 1555.23, p-value ≈ 0)

Conclusion:
We applied various statistical hypothesis tests Pearson's correlation, Spearman's rank correlation, Kendall's Tau, and the Chi-Squared test to analyze relationships in a retail dataset. The results revealed a positive association between `**Item_MRP**` and `**Item_Outlet_Sales**`, indicating that higher pricing tends to correlate with higher sales. Additionally, the Chi-Squared test showed a significant relationship between categorical variables like `**Item_Fat_Content**` and `**Item_Type**`, suggesting that these factors are not independent. Rather than depending on the library methods we calculated using the formula manually to understand in depth the consistency this dataset holds.

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

Theory:
Regression analysis is a statistical technique used to model and analyze relationships between variables. It is commonly used for predicting numerical outcomes and identifying trends. There are different types of regression, with Linear Regression being the most fundamental.
Regression analysis helps in understanding:

- The relationship between dependent and independent variables.
- The impact of independent variables on the dependent variable.
- Predicting values based on trends in data.

Mathematically, it follows the equation:

$$y = \beta_0 + \beta_1 x + \epsilon$$

Where:

y is the dependent variable (what we predict).

x is the independent variable (the predictor).

β_0 is the intercept (constant term).

β_1 is the coefficient (slope).

ϵ is the error term (random noise).

There can also exist more than one independent variable, in such a case, regression follows the equation:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n$$

Types of Regression:

- Linear Regression (Simple & Multiple)
- Polynomial Regression (Non-linear relationship)
- Logistic Regression (Classification problems)
- Ridge & Lasso Regression (Regularization techniques)
- Support Vector Regression (SVR)
- Decision Tree & Random Forest Regression

Steps:

a. **Linear Regression**

1. **Load the dataset**

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

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 103904 entries, 0 to 103903
Data columns (total 23 columns):
 #   Column                             Non-Null Count   Dtype
---  ------                             --------------   -----
 0   Gender                             103904 non-null  object
 1   Customer Type                      103904 non-null  object
 2   Age                                103904 non-null  int64
 3   Type of Travel                     103904 non-null  object
 4   Class                              103904 non-null  object
 5   Flight Distance                    103904 non-null  int64
 6   Inflight wifi service              103904 non-null  int64
 7   Departure/Arrival time convenient  103904 non-null  int64
 8   Ease of Online booking             103904 non-null  int64
 9   Gate location                      103904 non-null  int64
 10  Food and drink                     103904 non-null  int64
 11  Online boarding                    103904 non-null  int64
 12  Seat comfort                       103904 non-null  int64
 13  Inflight entertainment             103904 non-null  int64
 14  On-board service                   103904 non-null  int64
 15  Leg room service                   103904 non-null  int64
 16  Baggage handling                   103904 non-null  int64
 17  Checkin service                    103904 non-null  int64
 18  Inflight service                   103904 non-null  int64
 19  Cleanliness                        103904 non-null  int64
 20  Departure Delay in Minutes         103904 non-null  int64
 21  Arrival Delay in Minutes           103904 non-null  float64
 22  satisfaction                       103904 non-null  int64
dtypes: float64(1), int64(18), object(4)
memory usage: 18.2+ MB
```

The dataset consists of 103,904 entries with 23 columns, capturing various aspects of airline passengers' experiences and satisfaction levels. It contains a mix of categorical and numerical variables. The **categorical** attributes include **Gender, Customer Type, Type of Travel, Class, and Satisfaction**, which provide insights into passenger demographics and travel preferences. The **numerical** attributes include **Age, Flight Distance, Departure and Arrival Delays**, and various in-flight service ratings such as WiFi service, food and drink, seat comfort, and baggage handling, all measured on an integer scale. Arrival delay will act as our target variable for the purpose of performing regression.

2. **Perform all the necessary preprocessing steps**
    a. Handling missing values
    b. Drop unnecessary columns
    c. Data transformation

```
le = LabelEncoder()
df["satisfaction"] = le.fit_transform(df["satisfaction"])
```

Convert categorical columns to binary data.

3. **Split data and train the model**

```
# Split data for Linear Regression
X_linear = df[selected_features_linear]
y_linear = df["Arrival Delay in Minutes"]
X_train_lin, X_test_lin, y_train_lin, y_test_lin = train_test_split(X_linear, y_linear, test_size=0.2, random_state=42)
```

Split data into 80% training and 20% testing datasets.

```
# Standardize features for Linear Regression
scaler = StandardScaler()
X_train_lin = scaler.fit_transform(X_train_lin)
X_test_lin = scaler.transform(X_test_lin)

# Train Linear Regression Model
linear_model = LinearRegression()
linear_model.fit(X_train_lin, y_train_lin)
y_pred_lin = linear_model.predict(X_test_lin)
```

Regression is sensitive to sudden changes in ranges of independent variables, to avoid this we normalize our numerical columns. We use the z score transformation.
Then we train our model, and use the model to predict the testing dataset.

4. **Evaluation**

```
# Evaluate Linear Regression
mse_lin = mean_squared_error(y_test_lin, y_pred_lin)
accuracy_lin = 1 - (mse_lin / np.var(y_test_lin))  # R-squared equivalent
print(f"Accuracy (Linear Regression): {accuracy_lin*100:.2f}%")
print("Mean square value :" , mse_lin)

coefficients = linear_model.coef_
intercept = linear_model.intercept_

# Format the equation
equation = "y = " + " + ".join([f"{coeff:.4f}*{feature}" for coeff, feature in zip(coefficients, selected_features_linear)])
equation += f" + {intercept:.4f}"

print("Regression Equation:")
print(equation)
```

We evaluate a linear regression model by calculating the Mean Squared Error (MSE) and accuracy to measure performance. It then extracts the model's coefficients and intercept to construct a regression equation.

```
Accuracy (Linear Regression): 91.75%
Mean square value : 115.98955069314997
```

```
Regression Equation:
y = -0.1475*Flight Distance + 37.3971*Departure Delay in Minutes + -0.0577*Inflight wifi service + -0.0512*Seat comfort
```

The regression model achieves **91.75% accuracy**, indicating a strong fit. The regression equation suggests that Departure Delay in Minutes has the highest impact on the dependent variable, followed by On-board service, Inflight WiFi service, and Seat comfort, while Flight Distance has a small negative effect.
The mean squared error (MSE) of **115.99 suggests some variance in predictions**, but overall, the model performs well in explaining the relationship between these factors and the target variable.

b. Logistic regression :
Its a supervised classification algorithm used to predict binary (0/1) or multi-class outcomes. Despite its name, it is actually a classification model, not a regression model

- Logistic Regression estimates the probability **P(Y=1 | X)** using the **sigmoid (logistic) function**. The output is a probability value between **0 and 1**, which is then classified using a **threshold** (e.g., ≥0.5 as class 1, <0.5 as class 0).

1. Selecting the required feature through feature selection basically uses **RFE (Recursive Feature Elimination)** with Logistic Regression to select the 10 most important features.

```
# Selecting all numerical features for Logistic Regression
selected_features_logistic = df.select_dtypes(include=[np.number]).columns.tolist()
selected_features_logistic.remove("satisfaction")

# Feature Selection using RFE for Logistic Regression
X_logistic = df[selected_features_logistic]
y_logistic = df["satisfaction"]
logistic_selector = RFE(LogisticRegression(max_iter=5000, solver="lbfgs"), n_features_to_select=10)
logistic_selector.fit(X_logistic, y_logistic)
selected_features_logistic = X_logistic.columns[logistic_selector.support_].tolist()
print(selected_features_logistic)
```

2. Splits the selected feature dataset into training (80%) and testing (20%) set

```
# Split data for Logistic Regression
X_logistic = df[selected_features_logistic]
X_train_log, X_test_log, y_train_log, y_test_log = train_test_split(X_logistic, y_logistic, test_size=0.2, random_state=42, stratify=y_logisti
```

3. Removing the outliers to enhance the model and which scales data while reducing the impact of outliers.

```
# Use RobustScaler for better handling of outliers
scaler_log = RobustScaler()
X_train_log = scaler_log.fit_transform(X_train_log)
X_test_log = scaler_log.transform(X_test_log)
```

4. Training and evaluating the model

```
logistic_model = LogisticRegression(max_iter=5000, solver="lbfgs", C=0.5, class_weight="balanced")
logistic_model.fit(X_train_log, y_train_log)
y_pred_log = logistic_model.predict(X_test_log)

# Evaluate Logistic Regression
accuracy_log = accuracy_score(y_test_log, y_pred_log)
f1_log = f1_score(y_test_log, y_pred_log, average="weighted")
conf_matrix_log = confusion_matrix(y_test_log, y_pred_log)
classification_report_log = classification_report(y_test_log, y_pred_log)

print(f"Accuracy : {accuracy_log * 100:.2f}%")
print("F1 Score :", f1_log)
print("Confusion Matrix:\n", conf_matrix_log)
```

The Logistic Regression model is trained with hyperparameter tuning (max_iter=5000, solver="lbfgs", C=0.5, and class_weight="balanced"), ensuring better convergence and handling of imbalanced data. Finally, the model's performance is evaluated using accuracy, F1 score, confusion matrix, and classification report, which help analyze its effectiveness in predicting the target variable.

```
Accuracy : 80.60%
F1 Score : 0.8065822767400087
Confusion Matrix:
 [[9450 2326]
 [1706 7299]]
```

The output shows the selected features used for training the Logistic Regression model, including aspects like "Inflight Wifi Service," "Seat Comfort," and "Online Boarding." The model achieved an **accuracy** of **80.60%** and an F1 score of 0.80, indicating a good balance between precision and recall. The confusion matrix reveals that the model correctly classified **9450 instances as negative and 7299 as positive, while misclassifying 2392 false positives and 1706 false negatives.**

Conclusion:

The Linear Regression model achieved 91.75% accuracy, highlighting a strong relationship between Departure Delay, On-board Service, Inflight WiFi Service, and Seat Comfort with the target variable. Despite an MSE of 115.99, the model provided a reasonably good fit. For classification, Logistic Regression with Recursive Feature Elimination (RFE) was applied, resulting in an 80.60% accuracy and an F1 score of 0.80, indicating balanced classification performance. The confusion matrix further validated the model's effectiveness in distinguishing between classes.

In this we were able to understand how not only by selecting but to use feature engineering is utmost necessary for your model to be working really properly and accurate on unknown data

Aim: To perform Classification modelling on given dataset

Theory:

Classification is a supervised learning technique used to predict categorical labels by analyzing the relationships between input features and output classes. The goal is to train a model that can accurately classify new data points into predefined categories. Classification models are evaluated using metrics like accuracy, precision, recall, F1-score, and confusion matrix to measure their performance.

Types of Classification Algorithms:

1. **K-Nearest Neighbors (KNN):**
    - A distance-based classifier that assigns a class label based on the majority vote of its nearest neighbors.
    - Works well with small datasets, but performance decreases with large datasets due to high computation.
    - Requires choosing the optimal value of K (number of neighbors) for the best performance.
    - Sensitive to feature scaling, so normalization is necessary.

2. **Naive Bayes:**
    - A probabilistic classifier based on Bayes' Theorem, assuming independence between features.
    - Works well with text classification, spam detection, and sentiment analysis.
    - Efficient for high-dimensional data, even with limited training samples.
    - Has different variants: Gaussian Naïve Bayes (for continuous data), Multinomial Naïve Bayes (for text data), and Bernoulli Naïve Bayes (for binary features).

3. **Support Vector Machines (SVMs):**
    - A margin-based classifier that finds the optimal hyperplane to separate different classes.
    - Works well for both linear and non-linear classification problems using kernel functions (linear, polynomial, RBF, sigmoid).
    - Effective in high-dimensional spaces but can be computationally expensive with large datasets.
    - Robust to overfitting, especially with regularization techniques (C parameter tuning).

4. **Decision Tree:**

- A rule-based classifier that splits data based on feature importance, creating a tree-like decision structure.
- Easy to interpret and visualize but prone to overfitting if not pruned.
- Handles both numerical and categorical data well.
- Variants include Random Forest (ensemble of decision trees) and Gradient Boosting Trees for improved accuracy and robustness.

Steps:

1. **Load the dataset**

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

# Display first few rows
df.head()
```

	Unnamed: 0	id	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	...	Inflight entertainment	On-board service	Leg room service	Baggage handling	Checkin service	Inflight service	Cleanliness	Depa Del Mi
0	0	70172	Male	Loyal Customer	13	Personal Travel	Eco Plus	460	3	4	...	5	4	3	4	4	5	5	
1	1	5047	Male	disloyal Customer	25	Business travel	Business	235	3	2	...	1	1	5	3	1	4	1	
2	2	110028	Female	Loyal Customer	26	Business travel	Business	1142	2	2	...	5	4	3	4	4	4	5	
3	3	24026	Female	Loyal Customer	25	Business travel	Business	562	2	5	...	2	2	5	3	1	4	2	
4	4	119299	Male	Loyal Customer	61	Business travel	Business	214	3	3	...	3	3	4	4	3	3	3	

The dataset consists of 103,904 entries with 23 columns, capturing various aspects of airline passengers' experiences and satisfaction levels. It contains a mix of categorical and numerical variables. The categorical attributes include Gender, Customer Type, Type of Travel, Class, and Satisfaction, which provide insights into passenger demographics and travel preferences. The numerical attributes include Age, Flight Distance, Departure and Arrival Delays, and various in-flight service ratings such as WiFi service, food and drink, seat comfort, and baggage handling, all measured on an integer scale. The dataset appears to be complete, with no missing values, making it well-suited for exploratory data analysis and predictive modeling.

2. **Data preprocessing and splitting**

```
# Encode categorical variables
label_encoders = {}
categorical_cols = ["Gender", "Customer Type", "Type of Travel", "Class", "satisfaction"]

for col in categorical_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])  # Convert to numerical labels
    label_encoders[col] = le  # Store encoder for inverse transformation if needed later

# Separate features and target
X = df.drop("satisfaction", axis=1)
y = df["satisfaction"]

# Normalize numerical features for KNN & SVM
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

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

# Check data shapes
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

We preprocess the dataset for classification by encoding categorical variables using LabelEncoder, separating features and the target variable (satisfaction), and normalizing numerical features using StandardScaler—important for models like KNN and SVM. **The dataset is then split into training (80%) and testing (20%**) sets using train_test_split, ensuring balanced class distribution with stratification

3. **KNN model**

```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

y_pred_knn = knn.predict(X_test)

print("KNN Accuracy:", accuracy_score(y_test, y_pred_knn))
print(classification_report(y_test, y_pred_knn))

sns.heatmap(confusion_matrix(y_test, y_pred_knn), annot=True, fmt="d", cmap="Blues")
plt.title("KNN Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

We apply the K-Nearest Neighbors (KNN) classifier with n_neighbors=5 to classify data. The model is trained on X_train and y_train using knn.fit(), then tested on X_test to generate predictions (y_pred_knn).

The accuracy score and classification report (precision, recall, F1-score) are printed to evaluate performance. Additionally, a confusion matrix is computed and visualized using sns.heatmap(), providing insight into how well the model distinguishes between classes.

```
KNN Accuracy: 0.9287810981184736
              precision    recall  f1-score   support

           0       0.92      0.96      0.94     11776
           1       0.95      0.89      0.92      9005

    accuracy                           0.93     20781
   macro avg       0.93      0.92      0.93     20781
weighted avg       0.93      0.93      0.93     20781
```

Accuracy: The model achieved 92.88%, showing strong performance.

Precision & Recall:
Class 0: High recall (0.96) indicates fewer false negatives.
Class 1: Slightly lower recall (0.89) suggests some misclassification.

TP: 7983, TN: 11318.
FP: 458, FN: 1022.

4. **Decision Tree**

```
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)

y_pred_dt = dt.predict(X_test)


print("Decision Tree Accuracy:", accuracy_score(y_test, y_pred_dt))
print(classification_report(y_test, y_pred_dt))

sns.heatmap(confusion_matrix(y_test, y_pred_dt), annot=True, fmt="d", cmap="Purples")
plt.title("Decision Tree Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

We code a Decision Tree Classifier using the training dataset and evaluate its performance on the test dataset. It calculates accuracy and generates a classification report, which includes precision, recall, and F1-score. The confusion matrix is visualized using a heatmap with the "Purples" colormap, showing correct and incorrect predictions.

```
Decision Tree Accuracy: 0.9455752851162119
              precision    recall  f1-score   support

           0       0.96      0.95      0.95     11776
           1       0.93      0.94      0.94      9005

    accuracy                           0.95     20781
   macro avg       0.94      0.95      0.94     20781
weighted avg       0.95      0.95      0.95     20781
```

1. The Decision Tree Classifier achieved an accuracy of 94.56%, performing well in classifying both classes. The confusion matrix shows 11143 true negatives and 8507 true positives, indicating strong predictive capability. **However, 633 false positives and 498 false negatives suggest some misclassification.**
2. The precision and recall values are balanced, indicating a good trade-off between false positives and false negatives. Compared to KNN, the Decision Tree performs slightly better, likely due to its ability to capture complex decision boundaries.
3. We also tried to apply the **SVM model** on the same dataset. But the process was unsuccessful, it always took a very long time to train the model and no favourable outcome could be obtained.
4. Since **SVM solves a quadratic programming problem**, which is computationally expensive compared to other models like Decision Trees or KNN, it is possible that due to the size of our dataset, the computational overhead increased very much. Since the dataset has a large number of instances and features, SVM takes a long time to give favorable outcomes.

Conclusion:

From the previous experiments we understood how regression helps to understand the data, But here the right classification model depends on dataset size, computational efficiency, and feature distribution. **KNN performed well with 92.88% accuracy** but struggled slightly with class imbalance. **Decision Tree outperformed KNN with 94.56% accuracy**, effectively capturing decision boundaries while maintaining a balance between precision and recall. However, **SVM proved impractical due to high running time**, making it unsuitable for large datasets. The experiment highlights the importance of feature selection, preprocessing, and model interpretability in achieving optimal classification results.

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. Clustering is widely used in customer segmentation, anomaly detection, image segmentation, and 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.
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.
3. Hierarchical Clustering
    - Builds a tree of clusters using a bottom-up (agglomerative) or top-down (divisive) approach.
4. Model-based Clustering
    - Assumes data is generated by a mixture of underlying probability distributions (e.g., Gaussian Mixture Models).

K Means Clustering:
K-Means is a partition-based clustering algorithm that divides data into K clusters.

Algorithm Steps:

1. Choose the number of clusters, K.
2. Randomly initialize K centroids.
3. Assign each data point to the nearest centroid.
4. Update centroids by computing the mean of points in each cluster.
5. Repeat steps 3 & 4 until convergence (when centroids stop changing significantly).

Key Considerations:
Choosing K: The Elbow method and Silhouette Score help determine the optimal number of clusters.
Limitations: Sensitive to initial centroid placement and does not work well with non-spherical clusters or varying densities.
DBSCAN (Density-Based Spatial Clustering of Applications with Noise):
DBSCAN is a density-based clustering algorithm that groups points closely packed together while marking low-density points as noise.

Algorithm Steps:

1. Set parameters:
2. ε (epsilon): Maximum distance to be considered a neighbor.
3. MinPts: Minimum points required to form a dense region.
4. Choose an unvisited point and determine its ε-neighborhood.
5. If the point has at least MinPts neighbors, a new cluster is formed.
6. Expand the cluster by adding density-reachable points.
7. Repeat for all points, marking outliers as noise.

Advantages:

- Handles clusters of arbitrary shape.
- Automatically detects noise and outliers.
- No need to specify the number of clusters.

Disadvantages:

- Choosing optimal ε and MinPts is challenging.
- Struggles with varying density clusters.

Steps:

1. **Load and preprocess the dataset.**

	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	Ease of Online booking	Gate location	...	Inflight entertainment	On-board service	Leg room service	Baggage handling
0	1	0	13	1	2	460	3	4	3	1	...	5	4	3	4
1	1	1	25	0	0	235	3	2	3	3	...	1	1	5	3
2	0	0	26	0	0	1142	2	2	2	2	...	5	4	3	4
3	0	0	25	0	0	562	2	5	5	5	...	2	2	5	3
4	1	0	61	0	0	214	3	3	3	3	...	3	3	4	4

Checkin service	Inflight service	Cleanliness	Departure Delay in Minutes	Arrival Delay in Minutes	satisfaction
4	5	5	25	18.0	0
1	4	1	1	6.0	0
4	4	5	0	0.0	1
1	4	2	11	9.0	0
3	3	3	0	0.0	1

The dataset represents airline passenger satisfaction, containing demographic details (Gender, Age, Customer Type, Type of Travel), flight-related attributes (Class, Flight Distance, Gate Location), in-flight service ratings (Seat Comfort, Inflight WiFi, Entertainment, Food & Drink),

and delay metrics (Departure/Arrival Delay). Satisfaction is labeled as 0 (Unsatisfied) or 1 (Satisfied), making it suitable for classification and clustering.

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 to calculate:

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

where,

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

The **elbow point** is where the WCSS starts decreasing at a slower rate.

```
# Determine optimal clusters using Elbow Method
inertia = []
k_values = range(1, 11)
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(df_scaled)
    inertia.append(kmeans.inertia_)

# Plot the Elbow Method
plt.figure(figsize=(8, 5))
plt.plot(k_values, inertia, marker='o')
plt.xlabel("Number of Clusters")
plt.ylabel("Inertia")
plt.title("Elbow Method for Optimal K")
plt.show()
```

In this plot, the elbow appears around K = 4 or 5.The dataset likely has 4 or 5 naturally distinct groups, and choosing K balances segmentation quality and computational efficiency

3. **K means clustering**

```
# Apply K-Means with optimal K (assume 3 based on elbow method)
kmeans = KMeans(n_clusters=4, random_state=42, n_init=10)
df['Cluster'] = kmeans.fit_predict(df_scaled)

# Display cluster distribution
print(df['Cluster'].value_counts())
```

```
Cluster
1    35729
3    25697
0    22216
2    20262
Name: count, dtype: int64
```

We try out clusters for k = 4 and 5. With k = 5, we find that cluster size of last cluster is very small as compared to the other clusters formed. So we use k = 4, we can observe that the clusters formed are similar in size. The four cluster formed are:

0 - 22216
1 - 35729
2 - 20262
3 - 25697

Visualization of clusters:

```
from mpl_toolkits.mplot3d import Axes3D
from sklearn.decomposition import PCA

# Reduce dimensions to 3
pca_3d = PCA(n_components=3)
df_pca_3d = pca_3d.fit_transform(df_scaled)

df_plot_3d = pd.DataFrame(df_pca_3d, columns=['PC1', 'PC2', 'PC3'])
df_plot_3d['Cluster'] = df['Cluster']

# 3D scatter plot
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')

# Scatter plot with cluster colors
scatter = ax.scatter(df_plot_3d['PC1'], df_plot_3d['PC2'], df_plot_3d['PC3'],
                     c=df_plot_3d['Cluster'], cmap='viridis', alpha=0.6)

ax.set_title('K-Means Clustering Visualization (3D PCA)')
ax.set_xlabel('Principal Component 1')
ax.set_ylabel('Principal Component 2')
ax.set_zlabel('Principal Component 3')

plt.legend(*scatter.legend_elements(), title='Cluster')
plt.show()

# Ensure feature names match df_scaled
feature_names = df.select_dtypes(include=[np.number]).columns.tolist()[:df_scaled.shape[1]]

# Create PCA components DataFrame
pca_components = pd.DataFrame(pca_3d.components_, columns=feature_names,
                              index=['PC1', 'PC2', 'PC3'])

# Display how features contribute to each principal component
print(pca_components.T)
```

To visualize the clusters we perform **Principal Component Analysis (PCA)** to reduce the dataset dimensions to three, making it easier to visualize the clusters obtained from K-Means

clustering. The **3D scatter plot** visualizes the clusters in the transformed feature space, where different colors represent different clusters. The clusters appear to be well-defined, indicating that the K-Means algorithm successfully grouped similar data points.
The PCA components **help analyze the contribution** of original features to the new principal components, providing insights into how data is distributed across clusters.

4. **DBSCAN clustering**

DBSCAN is an unsupervised clustering algorithm that groups together densely packed points while marking outliers as noise. Unlike K-Means, it does not require specifying the number of clusters in advance and is useful for identifying arbitrarily shaped clusters.

1. Defines Two Parameters:
   - ε (epsilon): The radius to consider neighboring points.
   - MinPts: Minimum number of points required in an ε-neighborhood to form a cluster.
2. Classifies Points as:
   - Core Points: Points with at least MinPts neighbors within ε.
   - Border Points: Points within ε of a core point but with fewer than MinPts neighbors.
   - Noise Points: Points that are neither core nor border (outliers).
3. Clustering Process:
   - Starts with an unvisited point.
   - Expands a cluster if it's a core point, connecting all reachable points.

Repeats until all points are classified.

```
from sklearn.cluster import DBSCAN

# Apply DBSCAN clustering
dbscan = DBSCAN(eps=2, min_samples=50)  # Adjust eps and min_samples as needed
clusters = dbscan.fit_predict(df_scaled)

# Add cluster labels to the dataset
df["Cluster"] = clusters

# Display cluster distribution
print(df['Cluster'].value_counts())
```

```
Cluster
-1     70501
 1     11596
 0     11462
 2      4136
 3      3677
 4       695
 13      259
 11      219
 8       202
 5       178
 7       162
 14      129
 15      113
 6       107
 12      100
 10       75
 9        52
 19       51
 20       50
 16       49
 17       47
 18       44
Name: count, dtype: int64
```

We apply **DBSCAN clustering** with **eps=2** and **min_samples=50**, grouping data points based on density. It assigns cluster labels using dbscan.fit_predict(df_scaled) and stores them in the dataset. The output shows that **most points (70501)** are classified as **noise (-1)**, while the remaining data forms multiple clusters of varying sizes.
The **high number of noise points (-1)** in the output suggests that the chosen eps=2 is **too small to form dense clusters**, causing many points to be classified as outliers. Additionally, min_samples=50 requires at least 50 points in a neighborhood to form a cluster, which may be too restrictive for some regions of the dataset. As a result, only a few dense clusters are detected, while most points remain unclustered.

Visualization:

The DBSCAN clustering visualization in 3D PCA space shows that the majority of points are assigned to cluster -1 (noise) with only a few points belonging to distinct smaller clusters. This suggests that DBSCAN's parameters (epsilon and min_samples) may not be well-tuned for clear separation, leading to most points being grouped together while only a few scattered regions are identified as separate clusters.

<u>Inference from both the clustering:</u>

Overall we can see that the performance of **K means** is **better** than DBSCAN for clustering the dataset.

K-Means was a better choice for this dataset because the data is well-distributed without high-density regions, and clusters can be separated by distance rather than density.

DBSCAN struggled due to high dimensionality and the lack of naturally dense clusters, causing excessive noise detection.

Conclusion:

In all, we come to know that K-Means and DBSCAN analyze airline passenger satisfaction data. K-Means (K=4) successfully grouped data into well-separated clusters, confirmed by the Elbow Method and PCA visualization, though it is sensitive to outliers. DBSCAN, however, struggled with this dataset, classifying many points as noise due to a lack of well-defined dense regions, making it less effective for this case.

Aim: To implement a recommendation system on your dataset using the following machine learning techniques: Regression,Classification, Clustering, Decision tree, Anomaly detection, Dimensionality Reduction, Ensemble Methods.

Theory:

Types of Recommendation Systems : A Recommendation System suggests relevant items to users based on their preferences, behavior, or other factors. There are several types of recommendation techniques:

1. Collaborative Filtering

1. How it works: Recommends items based on user behavior and preferences, assuming users with similar tastes will like similar items.
2. Subtypes:
    a. User-based: Finds users similar to the target user and recommends items they liked.
    b. Item-based: Recommends items similar to those the user already interacted with.
3. Pros: Effective with large user bases; no need for item metadata.
4. Cons: Cold start problem (new users/items lack data); sparsity issues.
5. Example: Netflix suggesting shows based on what similar users watched.

2. Content-Based Filtering

1. How it works: Recommends items by analyzing their attributes and matching them to a user's preferences or past interactions.
2. Pros: Works well for new items; no reliance on other users' data.
3. Cons: Limited by item metadata quality; may lack diversity in recommendations.
4. Example: Spotify recommending songs with similar genres or artists to ones you've liked.

3. Hybrid Systems

1. How it works: Combines collaborative filtering and content-based methods to leverage strengths of both.
2. Pros: Mitigates cold start and sparsity issues; more robust and accurate.
3. Cons: Complex to implement and maintain.
4. Example: Amazon blending user purchase history with item feature similarity.

4. Knowledge-Based Systems

1. How it works: Uses explicit user requirements or domain knowledge to recommend items, often through rules or constraints.
2. Pros: Ideal for complex or infrequent purchases; no cold start issue.

3. Cons: Requires detailed domain knowledge; less dynamic.
4. Example: A travel site suggesting destinations based on user-specified criteria like budget and interests.

<u>Recommendation System Evaluation Measures</u>

Accuracy Measures:

These metrics evaluate how well the recommended items match the actual preferences or ratings of users.

1. Mean Absolute Error (MAE): Measures the average of the absolute differences between predicted ratings and actual ratings.

- **Formula**: $MAE = \frac{1}{n}\sum_{i=1}^{n}|r_i - \hat{r}_i|$
  - r_i = Actual rating
  - $\hat{r}_i$ = Predicted rating

2. Root Mean Squared Error (RMSE): Similar to MAE but gives higher weight to large errors due to squaring the differences.

Formula: $RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(r_i - \hat{r}_i)^2}$

3. Precision: Measures the fraction of recommended items that are actually relevant to the user.

Formula: $Precision = \frac{\text{Number of relevant recommended items}}{\text{Total number of recommended items}}$

4. Recall: Measures the fraction of relevant items that were actually recommended to the user.

Formula: $Recall = \frac{\text{Number of relevant recommended items}}{\text{Total number of relevant items}}$

5. F1-Score: The harmonic mean of Precision and Recall, balancing both.

Formula: $F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$

Model used : XGBoost

XGBoost (Extreme Gradient Boosting) is a highly efficient and robust machine learning algorithm that falls under the umbrella of gradient boosting frameworks

Implementation :

Dataset Overview : The data spans different years (notably 2001 and 2012, as seen in the sample) and includes detailed counts of various types of **crimes** reported in each district. The dataset is structured in a tabular format, with each row representing a district or a specific region (e**.g., railway police, city-specific data**) within a state/union territory, and each column representing a specific crime category or metadata attribute.

The dataset is likely sourced from official crime records, such as those maintained by the **National Crime Records Bureau (NCRB)** of India, given the detailed breakdown of crime types and the geographical granularity. It includes both aggregated totals (e.g., "TOTAL" rows for states) and individual district-level data, making it suitable for analyzing crime patterns across regions, time periods, and crime categories.

Features :

1. **Murder**: number of murder cases.
2. **Attempt to murder:** number of attempted murder cases.
3. **Culpable homicide not amounting to murder**: cases of culpable homicide not classified as murder.
4. **Rape**: total number of rape cases.
5. **Custodial rape**: rape cases occurring in custody (subset of rape).
6. **Other rap**e: rape cases not classified as custodial rape.
7. **Kidnapping & abduction:** total kidnapping and abduction cases.
8. **Other IPC crimes**: Miscellaneous crimes under the Indian Penal Code (IPC) not covered by the above categories.
9. **Total IPC Crimes**: Sum of all IPC crimes reported in the district for the given year.

1. Importing dataset

	STATE/UT	DISTRICT	YEAR	MURDER	ATTEMPT TO MURDER	CULPABLE HOMICIDE NOT AMOUNTING TO MURDER	RAPE	CUSTODIAL RAPE	OTHER RAPE	KIDNAPPING & ABDUCTION	...	HURT/GREVIOUS HURT	DOWRY DEATHS	ASSAULT ON WOMEN WITH INTENT TO OUTRAGE HER MODESTY	INSULT TO MODESTY OF WOMEN	CRUELTY BY HUSBAND OR HIS RELATIVES	IMPORTATION OF GIRLS FROM FOREIGN COUNTRIES	CAUSING DEATH BY NEGLIGENCE	OTHER IPC CRIMES	TOT I CRIM
0	ANDHRA PRADESH	ADILABAD	2001	101	60	17	50	0	50	46	...	1131	16	149	34	175	0	181	1518	41
1	ANDHRA PRADESH	ANANTAPUR	2001	151	125	1	23	0	23	53	...	1543	7	118	24	154	0	270	754	41
2	ANDHRA PRADESH	CHITTOOR	2001	101	57	2	27	0	27	59	...	2088	14	112	83	186	0	404	1262	58
3	ANDHRA PRADESH	CUDDAPAH	2001	80	53	1	20	0	20	25	...	795	17	126	38	57	0	233	1181	31
4	ANDHRA PRADESH	EAST GODAVARI	2001	82	67	1	23	0	23	49	...	1244	12	109	58	247	0	431	2313	65
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
9012	LAKSHADWEEP	LAKSHADWEEP	2012	0	0	0	0	0	0	0	...	3	0	1	0	1	0	0	32	
9013	LAKSHADWEEP	TOTAL	2012	0	0	0	0	0	0	0	...	3	0	1	0	1	0	0	32	
9014	PUDUCHERRY	KARAIKAL	2012	5	6	2	6	0	6	2	...	186	0	2	0	1	0	44	392	7
9015	PUDUCHERRY	PUDUCHERRY	2012	24	21	10	7	0	7	17	...	632	0	7	2	5	0	219	1668	34
9016	PUDUCHERRY	TOTAL	2012	29	27	12	13	0	13	19	...	818	0	9	2	6	0	263	2060	42

9017 rows × 34 columns

2. Dataset is already preprocessed and we can check by this code :

df.isnull().sum()

STATE/UT	0
DISTRICT	0
YEAR	0
MURDER	0
ATTEMPT TO MURDER	0
CULPABLE HOMICIDE NOT AMOUNTING TO MURDER	0
RAPE	0
CUSTODIAL RAPE	0
OTHER RAPE	0
KIDNAPPING & ABDUCTION	0
KIDNAPPING AND ABDUCTION OF WOMEN AND GIRLS	0
KIDNAPPING AND ABDUCTION OF OTHERS	0
DACOITY	0
PREPARATION AND ASSEMBLY FOR DACOITY	0
ROBBERY	0
BURGLARY	0
THEFT	0
AUTO THEFT	0
OTHER THEFT	0
RIOTS	0
CRIMINAL BREACH OF TRUST	0
CHEATING	0

3. Understanding through graphs the districts with maximum total IPC crimes.

The data, aggregated by state/union territory (UT), shows Madhya Pradesh leading with the highest average at approximately 190,000 crimes, followed closely by Maharashtra with around 180,000, and Tamil Nadu with about 160,000. Other notable states include Andhra Pradesh, Uttar Pradesh, Rajasthan, Karnataka, Kerala, Gujarat, and Bihar, with averages ranging from 100,000 to 140,000 crimes. Integrating this information with the safety scores and crime feature analysis **allows for a more informed ranking of districts,** ensuring recommendations steer users toward regions which **exhibit relatively lower crime averages**, thereby enhancing the reliability and relevance of the safety suggestions.

4. Heatmap of features and safety score

The heatmap uses a color gradient from blue (negative correlation) to red (positive correlation), with values ranging from -1 to 1. It reveals strong **positive correlations** among crime features (e.g., **Murder and Rape at 0.83, Theft and Rape at 0.79**), indicating that districts with high rates of one crime tend to have high rates of others. Conversely, all crime features show moderate negative correlations with the Safety Score (ranging from -0.22 to -0.27), suggesting that higher crime rates are associated with lower safety scores

6. Preparing data for **XGboost model.**

```
def prepare_data(df, features, target_col='TOTAL IPC CRIMES', min_samples=2):
    # Create safety label (1 if unsafe, 0 if safe)
    median_crime = df[target_col].median()
    df['SAFETY_LABEL'] = (df[target_col] > median_crime).astype(int)

    # Count records per district and filter
    district_counts = df.groupby(['STATE/UT', 'DISTRICT']).size()
    valid_districts = district_counts[district_counts >= min_samples].index

    # Filter original dataframe
    df_filtered = df.set_index(['STATE/UT', 'DISTRICT']).loc[valid_districts].reset_index()

    # Aggregate by district
    df_agg = df_filtered.groupby(['STATE/UT', 'DISTRICT'])[features + ['SAFETY_LABEL']].mean().reset_index()

    return df_agg
```

The **prepare_data function** processes a crime dataset by creating a safety label (1 for unsafe, 0 for safe) based on the median of 'TOTAL IPC CRIMES', ensuring districts with above-median

crime rates are marked unsafe. It filters out districts with fewer than min_samples (default 2) records, using a group by operation to count records per district, and retains only valid districts. The function then **aggregates the filtered data by district,** calculating the mean of specified crime features

7. **Training** the recommendation model

```
def train_xgboost_model(X_train, y_train, X_test, y_test):
    # Convert to DMatrix format
    dtrain = xgb.DMatrix(X_train, label=y_train)
    dtest = xgb.DMatrix(X_test, label=y_test)

    # Parameters for binary classification
    params = {
        'objective': 'binary:logistic',
        'eval_metric': ['error', 'auc'],
        'max_depth': 5,
        'eta': 0.1,
        'subsample': 0.8,
        'colsample_bytree': 0.8,
        'seed': 42
    }

    # Train with early stopping
    model = xgb.train(params, dtrain, num_boost_round=100,
                      early_stopping_rounds=10, evals=[(dtest, 'test')],
                      verbose_eval=True)
```

The train_xgboost_model function trains an XGBoost classifier for binary classification using training and test datasets. It converts the input data into XGBoost's DMatrix format and defines parameters for binary logistic regression, including 'objective', 'eval_metric' (error and AUC), 'max_depth' (5), 'eta' (0.1), 'subsample' and 'colsample_bytree' (0.8), and a random 'seed' (42). The **model is trained with 100 boosting rounds**, early stopping after 10 rounds if no improvement, and verbose evaluation output, enabling real-time performance monitoring on the test set.

8. **Giving** recommendations (Final Result of our model)

- Recommendation requires the state name.
- Then we ask the user to rate the crimes, they assign a weight to every crime, higher weight indicated, the particular crime should be considered more important when recommendations are made.

```python
def get_safety_recommendations(model, df_agg, features, top_n=5, state=None, crime_weights=None):
    X_all = df_agg[features]
    dmatrix = xgb.DMatrix(X_all)

    # Get safety probabilities and convert to score (0-100)
    df_agg['SAFETY_PROB'] = model.predict(dmatrix)
    df_agg['SAFETY_SCORE'] = (1 - df_agg['SAFETY_PROB']) * 100

    # Apply state filter if specified
    if state:
        df_agg = df_agg[df_agg['STATE/UT'] == state.upper()]

    # Apply crime weights if specified
    if crime_weights:
        for crime, weight in crime_weights.items():
            if crime in features:
                max_val = df_agg[crime].max()
                if max_val > 0:
                    df_agg[crime + '_WEIGHTED'] = (df_agg[crime] / max_val) * weight
                    df_agg['SAFETY_SCORE'] = df_agg['SAFETY_SCORE'] - df_agg[crime + '_WEIGHTED']


    recommendations = df_agg.sort_values('SAFETY_SCORE', ascending=False).head(top_n)

    results = []
    for _, row in recommendations.iterrows():
        results.append({
            'Rank': len(results) + 1,
            'State': row['STATE/UT'],
            'District': row['DISTRICT'],
            'Safety_Score': round(row['SAFETY_SCORE'], 1),
            'Murder_Rate': round(row['MURDER'], 2),
            'Rape_Rate': round(row['RAPE'], 2),
            'Theft_Rate': round(row['THEFT'], 2),
            'Riots_Rate': round(row['RIOTS'], 2)
        })
```

The get_safety_recommendations function generates safety recommendations by predicting safety probabilities for districts using a trained XGBoost model, converting them to safety scores (0-100), and ranking districts accordingly. It filters the data by a specified state if provided, applies weighted adjustments to crime features (e.g., Murder, Rape) if crime weights are given, and **sorts districts by safety score** in descending order to **select the top n (default 5) recommendations.**

9. Result (Model evaluation)

```
Model Evaluation:
Accuracy: 0.87

Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.84      0.86        80
           1       0.86      0.90      0.88        86

    accuracy                           0.87       166
   macro avg       0.87      0.87      0.87       166
weighted avg       0.87      0.87      0.87       166
```

The "Model Evaluation" output for the XGBoost classifier shows an accuracy of 0.87, indicating that **87%** of the predictions align with the actual safety labels. The classification report provides detailed metrics for the binary classification (0 for safe, 1 for unsafe), with precision, recall, and f1-score all at 0.86 for class 0 and 0.88 for class 1, reflecting balanced performance across both categories. The macro and weighted averages of 0.87 across precision, recall, and f1-score, based on 166 total samples.

The "**Feature Importance**" plot displays the relative significance of crime features in the XGBoost model, measured by F-score, for predicting district safety as of April 11, 2025. Riots emerge as the most influential feature with an **F-score of 215.0**, followed closely by Theft at 197.0 and Rape at 179.0, while Murder has the lowest importance at 125.0. This indicates that riots and theft rates have a **stronger impact on the safety** score compared to murder and rape.

10. User input

```
Enter a state to focus on (or press Enter for all India): MAHARASHTRA

Searching for safe districts in MAHARASHTRA...

Number of recommendations to display (default 5): 5

Would you like to weight specific crime types more heavily?
For example, you might want to weight 'MURDER' or 'RAPE' more heavily.
Enter crime:weight pairs like 'MURDER:3 RAPE:4' or press Enter to skip
Crime weights: 'MURDER:5 RAPE:5 THEFT:5 RIOTS:3'
```

11. Recommendations according to user input

```
=== Recommended Safe Districts ===

#1: PUNE RLY., MAHARASHTRA
Safety Score: 97.7/100
Crime Rates (avg per year):
  Murder: 2.58 | Rape: 0.83
  Theft: 300.0 | Riots: 3.17

#2: SINDHUDURG, MAHARASHTRA
Safety Score: 97.5/100
Crime Rates (avg per year):
  Murder: 12.83 | Rape: 8.17
  Theft: 111.75 | Riots: 41.08

#3: HINGOLI, MAHARASHTRA
Safety Score: 92.0/100
Crime Rates (avg per year):
  Murder: 35.83 | Rape: 14.58
  Theft: 183.33 | Riots: 115.08

#4: NANDURBAR, MAHARASHTRA
Safety Score: 90.2/100
Crime Rates (avg per year):
  Murder: 35.58 | Rape: 18.42
  Theft: 171.42 | Riots: 90.75

#5: NAGPUR RLY., MAHARASHTRA
Safety Score: 88.0/100
Crime Rates (avg per year):
  Murder: 6.58 | Rape: 0.92
  Theft: 933.67 | Riots: 10.25
```

Conclusion:

The experiment successfully implemented a recommendation system for district safety using a variety of machine learning techniques, including regression, classification, clustering, decision trees, anomaly detection, dimensionality reduction, and ensemble methods, with XGBoost as the primary model. Leveraging a comprehensive crime dataset from the National Crime Records Bureau (NCRB) of India, the system effectively processed and analyzed features such as Murder, Rape, Theft, and Riots, creating safety scores and labels based on total IPC crimes. The model achieved an accuracy of 0.87, with feature importance highlighting Riots and Theft

Aim : Exploratory Data Analysis using Apache Spark and Pandas

Theory :

1. What is Apache Spark and How Does It Work?

Apache Spark is an open-source, distributed computing framework designed for big data processing and analytics. It provides an efficient, scalable, and fault-tolerant system to handle large datasets across a cluster of computers. Spark is built around the concept of Resilient Distributed Datasets (RDDs), which are immutable, distributed collections of data that can be processed in parallel.

How it Works:

<u>Architecture Of Apache Spark</u>

1. **Spark Driver**
- Role: The main program that runs the application and creates the SparkContext, which serves as the gateway to all Spark functions.
- Components: Includes DAG Scheduler, Task Scheduler, Backend Scheduler, and Block Manager to translate user code into jobs executed on the cluster.
- Function: Coordinates with the Cluster Manager to control job execution, breaking jobs into tasks and distributing them to worker nodes.
2. **SparkContext**
- Purpose: Acts as the entry point for Spark functionality, connecting the driver to the cluster and queuing tasks for worker nodes.
- Tasks: Monitors jobs, manages Resilient Distributed Datasets (RDDs), and enables caching of results.
3. **Cluster Manager**

- Role: Allocates resources across the cluster for job execution.
- Interaction: Works with the Spark Driver to assign tasks to worker nodes, ensuring efficient resource use.

4. **Executors**

- Function: Run on worker nodes to execute tasks and store data in memory or cache.
- Lifecycle: Register with the driver at the start, operate during the application's lifespan, and are dynamically added or removed based on demand.
- Concurrency: Each executor has time slots to run tasks in parallel, handling data processing and external read/write operations.

5. **Worker Nodes**

- Role: Act as slave nodes that execute tasks assigned by the SparkContext and return results to the driver.
- Scalability: Multiple worker nodes process tasks in parallel, with each handling partitions (units of work) for efficiency.
- Monitoring: Managed by Spark workers to ensure smooth computation.

2. Execution:

a. Spark uses a master-worker architecture. The driver program (master) coordinates tasks, while worker nodes execute them.
b. Data is partitioned across nodes, and operations (transformations and actions) are performed in parallel.

3. In-Memory Computation: Spark keeps data in memory whenever possible, reducing I/O overhead and speeding up processing compared to disk-based systems like Hadoop MapReduce.

4. Lazy Evaluation: Transformations (e.g., map, filter) are not executed immediately; they are queued until an action (e.g., collect, count) triggers computation.

2. How is Data Exploration Done in Apache Spark? Explain Steps.

Data exploration in Apache Spark involves analyzing datasets to uncover patterns, statistics, and insights using its distributed computing capabilities. Below are the key steps:

Steps for Data Exploration in Apache Spark:

1. **Setup Environment**:
   a. Initialize a SparkSession, the entry point for Spark functionality (e.g., SparkSession.builder.appName("EDA").getOrCreate()).
2. **Load Data**:
   a. Read data from various sources (CSV, JSON, Parquet, databases) into a DataFrame or RDD. Example: df = spark.read.csv("data.csv").

3. **Inspect Data**:
    a. View schema (df.printSchema()) to understand column names and data types.
    b. Display sample rows (df.show(5)) to get a quick look at the data.
4. **Summary Statistics**:
    a. Use df.describe() to compute basic statistics (count, mean, min, max, stddev) for numeric columns.
5. **Handle Missing Values**:
    a. Check for nulls (df.filter(df.column.isNull()).count()) and decide whether to drop or fill them (df.na.fill(0)).
6. **Data Cleaning**:
    a. Filter rows, remove duplicates (df.dropDuplicates()), or transform columns using Spark SQL or DataFrame operations.
7. **Explore Distributions**:
    a. Compute aggregations (e.g., df.groupBy("column").count()) or use histograms to analyze value distributions.
8. **Correlations**:
    a. Calculate correlations between numeric columns using df.stat.corr("col1", "col2").
9. **Visualization (Optional)**:
    a. Convert Spark DataFrame to Pandas (df.toPandas()) for plotting with libraries like Matplotlib or Seaborn (note: only for small datasets due to memory constraints).
10. **Iterate**:
    a. Refine analysis by applying filters, joins, or custom transformations based on initial findings

Conclusion:
Apache Spark is a powerful tool for exploratory data analysis (EDA) on large-scale datasets, leveraging its distributed, in-memory processing to handle big data efficiently. It works by partitioning data across a cluster and executing operations in parallel, using RDDs or DataFrames. For EDA, Spark enables loading, inspecting, summarizing, and cleaning data through a series of steps like schema inspection, statistical summaries, and aggregations. While Spark excels at scalability, small-scale visualization often requires integration with Pandas. Combining Spark's distributed capabilities with Pandas' ease of use provides a robust framework for comprehensive data exploration.

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

Theory:
1. What is Streaming? Batch vs. Stream Data

Batch Data Processing
Definition: Processing a large volume of data at once (in batches) at scheduled intervals.

Characteristics:
- Data is collected over time and processed in chunks.
- High latency (delay between data collection and processing).
- Suitable for applications where real-time analysis is not required (e.g., daily sales reports).

Example:
Running an end-of-day report on transactions stored in a database.

Stream Data Processing
Definition: Processing data in real-time as it is generated.

Characteristics:
- Data is processed continuously with minimal latency.
- Used for real-time analytics (e.g., fraud detection, live dashboards).

Example:
Analyzing live Twitter feeds for trending topics.

Feature	**Batch Processing**	**Stream Processing**
Data Input	Collected over time	Continuous, real-time
Latency	High (minutes/hours)	Low (seconds/milliseconds)
Use Cases	Reports, historical analysis	Fraud detection, live monitoring
Tools	Hadoop MapReduce, Spark Batch	Spark Streaming, Kafka, Flink

2. How Data Streaming Works in Apache Spark
Apache Spark provides Spark Streaming (now part of Structured Streaming) for real-time data processing.

Key Concepts in Spark Streaming:

- DStream (Discretized Stream):
  - A sequence of RDDs (Resilient Distributed Datasets) representing data in small time intervals (micro-batches).
- Structured Streaming:
  - A higher-level API built on Spark SQL for real-time processing with DataFrame/Dataset abstractions.

Steps for Batch Processing

1. Initialize Spark Session
```
from pyspark.sql import SparkSession
# Create a SparkSession
spark = SparkSession.builder \
    .appName("BatchProcessingExample") \
    .getOrCreate()
```

2. Read Batch Data (CSV, JSON, Parquet, etc.)
```
# Reading from a CSV file
batch_df = spark.read \
    .format("csv") \
    .option("header", "true") \
    .option("inferSchema", "true") \
    .load("path/to/data.csv")
```

3. Apply Transformations (Filtering, Aggregations, Joins, etc.)
```
from pyspark.sql.functions import col, count, avg
# Example: Filtering and aggregation
filtered_df = batch_df.filter(col("age") > 30)
# GroupBy and Aggregation (e.g., average salary by department)
agg_df = batch_df.groupBy("department") \
    .agg(
        count("*").alias("employee_count"),
        avg("salary").alias("avg_salary")
    )
```

4. Output the Processed Data (Save to Disk, Database, etc.)
```
# Write to CSV
agg_df.write \
    .format("csv") \
    .mode("overwrite") \  # Options: "append", "overwrite", "ignore"
```

```
    .option("header", "true") \
    .save("output/path")
```

5. Stop Spark Session
   spark.stop()

Steps for Stream Processing

1. Initialize Spark Session
   ```
   from pyspark.sql import SparkSession
   spark = SparkSession.builder.appName("StreamingExample").getOrCreate()
   ```

2. Read Streaming Data (from Kafka, Socket, Files, etc.)
   ```
   # Reading from a socket (for testing)
   streaming_df = spark.readStream.format("socket").option("host",
   "localhost").option("port", 9999).load()
   ```

3. Apply Transformations (Filtering, Aggregations, etc.)
   ```
   # Example: Word count on streaming text
   from pyspark.sql.functions import explode, split
   words = streaming_df.select(explode(split("value", " ")).alias("word"))
   word_counts = words.groupBy("word").count()
   ```

4. Output the Stream (Console, Kafka, HDFS, etc.)
   ```
   query = word_counts.writeStream.outputMode("complete").format("console").start()
   ```

5. Start and Manage the Stream
   ```
   query.awaitTermination()  # Keeps the stream running
   ```

Conclusion:

In our Apache Spark experiment, batch and stream processing each showed their strengths. Batch processing handled large, static datasets like historical data perfect for scheduled ETL, analytics, and reports, using steps like `spark.read.csv()` and `df.write.save()`. It's steady and reliable. Stream processing, meanwhile, tackled real-time data like live logs or IoT feeds processing it in micro-batches or events with `spark.readStream` and `writeStream.start()`. It's ideal for instant tasks like fraud detection or alerts, feeling dynamic and alive.

Avan B. Shetty
D15C - 52

05/05

AIDS-1 Assignment

1) 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 is the simulation of human intelligence in machines, enabling them to perform tasks such as learning, reasoning, problem-solving and decision-making. AI systems use techniques like machine learning, deep learning and NLP.

The solution Through AI:

1. Healthcare and Diagnostics
AI powered models were used to detect COVID-19 from medical images like X-rays and CT scans, helping in faster diagnosis. AI assisted chatbots provided initial consultations, reducing the burden on hospitals

2. Vaccine Development
AI accelerated vaccine research by analyzing virus structures, predicting protein interactions and identifying potential drug candidates

3 Contact Tracing and Surveillance:
AI driven contact tracing apps, such as Aarogya Setu, helped track COVID-19 exposure and prevent outbreaks.

4. Supply Chain Optimization
AI helped predict demand for medical supplies, optimize logistics and ensure the timely delivery of essential goods, preventing shortages

Q.2) What are AI agents terminology, explain with examples?

→ Key terminologies in AI Agents.

① Agent: An agent that perceives the environ and takes actions to achieve a specific goal

eg. A self-driving car perceives traffic conditions and adjusts its speed accordingly

② Perception: (sensors)

The agent gathers information about the environment through sensors.

eg: A robot vacuum cleaner uses infrared sensors to detect obstacles

③ Actuators:

Components that allow the agent to interact with the environment

eg: A robotic arm in a factory moves objects using mechanical actuators.

④ Environment:

The surroundings in which an agent operates

It can be fully or partially observable

eg: A chess-playing AI has a fully observable env, while a personal assistant AI like Siri operates in a partially observable one

Q3) How AI Technique is used to solve 8-puzzle problem?

→ It consists of a 3X3 grid with 8 numbered tiles and 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	

3) Solving the 8 Puzzle problem

① Breadth first Search (BFS)

- BFS as 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
Disadvantages: BFS has a high memory require

(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 is like can get stuck in deep non-optimal paths and may not find the shortest solution.

Q.4) What is PEAS descriptor? Give PEAS descriptor for following
→ 1) Toni Driver:
- P: Min time travel, Fuel efficiency, passenger safety, obey traffic rules
- E: roads, traffic, passengers, weather, obstacle pedestrians
- A: Steering, accelerator, brakes, turn signals, ho
- S: Camera, GPS, speedometer, LiDAR, microph

2) A 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, notifica
- S: Patient input, lab reports, electronic health r

Sundaram
FOR EDUCATIONAL USE

3) A music composer:

- P: Quality of music, adherence to genre, audience engagement
- E: digital workspace, music production software, realtime composition settings
- A: Audio output, digital instrument selection, file saving / export
- S: User inputs, style preference, tempo, feedback from listness, music theory constraints

4) An aircraft Autolander

- P: Smooth landing, accuracy in reaching runaway, passenger safety, fuel efficiency
- E: Airspace runaway, 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 inputs, student essays, predefined, grading criteria
- A: Feedback generation, score assignment, highlighting errors, suggesting improvement
- S: Optical character recognition, NLP, & spell checker

6) A robotic sentry gun for Keck lab

→ P: Target accuracy, threat deflection efficiency

E: Keck lab premises, intruders, lighting conditions, obstacles.

A: Gun aiming system, firing mechanism, camera panning alert system

S: Motion detectors, infrared sensors, camera, LIDAR, radar

Q.5) Categorize a shopping bot for an offline bookstore according to each six dimension (fully / partially observable / deterministic / stochastic, sequential, episodic, static / dynamic, discrete / continuous, single / multi agent

→ 1) Partially Observable: The bot may not have complete visibility

2) Stochastic: The environment is unpredictable. There are chances that it may have too many book vendors so choosing becomes difficult

3) Sequential: Each decision The bot makes affects future states. There is a continuous manner The bot reacts to its obstacles

4) Dynamic: The bookstore environment changes overtime. The new-books may arrive and There can be replacement over new versions of b

5) Discrete: Bot choose or make discrete cho

6) Multi agent: The bot interacts with multiple entities.

Q.6) Differentiate Model and Utility Based Agent.

Model Based Agent	Utility Based Agent
1) Maintains an internal model of environment to make decision	1) Uses a utility function to measure performance and make option choices
2) Relies on stored knowledge and updates the model	2) Chooses actions based on maximizing expected utility
3) Can adapt to changing environment by updating internal node	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

Q.7) Explain the architecture of a Knowledge Based Agent and Learning Agent

→ ① Knowledge-Based Agent Architecture.

It makes decisions using knowledge base and measuring mechanism.

Component:

1) KB (Knowledge Base): It's the key component of KBA. These deal with the real facts of world. It's the mixture of sentences which are expl in knowledge representation language

2) Inference Engine - It is knowledge-based system engine used to infer new knowledge in system

Actions done by KB Agent

1) It Tells what is recognized from the environm and what it needs to know the knowledge

2) It asks what action to do? and gets answers from The knowledge base

3) It tells the which action is selected, then agent will execute that action

(2) Learning Based Agent

A learning agent improves its performance over ti by learning from past experiences and interacti with the environment

Architecture components:

① Learning element: analyzes feedback from the environment and improves knowledge
② Performance element: makes & executes decisions
③ critic: evaluates the agent's action and provides feedback
④ Problem generator: suggests explatory actions to improve learning

Actions by learning based agent

1. The performance element selects an action
2. Critic evaluates the action of agent and provides feedback
3. The learning element updates the agents knowledge to improve future decisions
4. The problem generator suggests new strategies to explore better solutions.

Q.9) Convert the following to predicates

ⓐ Anita travels by car if available otherwise travels by bus

→ available(car) → travels (Anita, car) ..(1)
¬ available (car) → travels (anita, bus) ..(2)

ⓑ Bus goes via Andheri and Goregoan
ⓒ Car has puncture so is not available
Will Anita travel via Goregoan? Use forward reasoning

→ (b) route (bus, goregoan) ∧ route (bus, Andheri) ..(3)
(c) puncture (car) → ¬ available (anita, car) ..(4)
{ since car is not available for travelling, Anita

won't use the car)
- so its basically true

- Now since the car is not available Anita travel by bus ...[from (1)]

¬ available (car) → travels (anita , bus)

also · routes (bus, goregoan) · from (3)
and · routes (anita, bus)

→ ∴ Anita will travel via Goregoan.

(10) Find the route from S to G using BFS

→ <u>Solution</u>

BFS explores all the neighbours of a node before moving to the next level.

1. start at S
Queue = [S]

2. Dequeue S and explore it's neighbour
Queue = [A, B, C]

3. Dequeue A and explore neighbours
Queue = [B, C, D]

A —3— D

4. Dequeue B and explore neighbors D, G
{ D already taken, so consider G}
Queue = [C, D, G]

5. Dequeue C and explore next level
Queue = [D, G]

6. Dequeue D
queue = [G]

7. Dequeue G

∴ G is our goal node. BFS will stop here.
Route from S to G : [S → B → G]

(ii) What do you mean by depth limited search? Explain Iterative Deepening Search with example.

→ • Depth Limited Search (DLS) is an uninformed search algorithm that modifies DFS by then introducing a depth limit (L), preventing the further exploration beyond the defined limit. This prevents infinite loops in graphs but risks missing goals beyond L.
• Completeness: Doesn't gurantee to reach the goal node
• Does'nt provide optimality
• DLS is memory efficient
• It has space and time complexity of O(b!).

- Iterative Deepening Search:
- IDS is a combination of both BFS and DFS
BFS guranteed to reach goal node and DFS ou
less memory space
- It gradually increases the depth-limit from
0, 1, 2 and so, on and reach the goal no

Example:

- Goal node is H and initial depth-limit = [0]
iteration 1, Nodes visited = [A], goal not found
- Iteration : 2 limit = 1
Terminate with sequence A → B → C
- Iteration : 3, limit = 4
Will expand the nodes from level 0 to level 3
and search terminate with A → B → D → F → E →
Hence, H is the goal node obtained

- It may or maynot provide the Completeness
- It does not give an optimal solution always
- Time and space complexity is O(bd).

Sundaram

FOR EDUCATIONAL

(12) Explain Hill Climbing and its Drawbacks in detail with example. Also state limitations of steeper ascent hill climbing.

→ • Hill climbing is a local state search optimization algorithm which moves forward toward a better neighbouring solution until it reaches a goal

• It is based on the 'heuristic search technique' where the person who is climbing up on the hill estimates the direction which will lead him to the highest peak

Example:

- Start at root node A (10)
- Compare its children B, C, D
- Move to child with highest value i.e B(12)
- Repeat for B's children E & F
- Terminate at E (18)
- Thus algorithm stops at E(18) and doesn't reach goal node

→ Drawbacks

(1) Local Maxima: The algorithm greedily selects the best immediate child and can thus stuck at the local maxima

(2) plateau: If siblings have equal values then the algorithm can't decide the next step & gets stuck

(3) Ridges: Narrow uphill paths require backtracking which hill climbing algorithm doesn't support

- Limitations of steepest ascent hill climbing:
  1. Computationally expensive: Evaluates all neighbors before selecting the best
  2. Stuck: It can still get stuck in local maxima, plateaus or ridges
  3. No global optimality: It only requires favourable immediate improvements

(Q.13) Explain simulated Annealing. Write its Algorithm.

→ Simulated Annealing is a probabilistic optimization algorithm inspired by metallurgical process of annealing where materials are heated and

Sundaram

FOR EDUCATIONAL USE

to reduce effects and defects. It escapes the local optima by temporarily accepting worse solution with a probability.

Algorithm:

1. Initialize
2. Let an initial solution and define a initial Temperature T
3. Repeat untill stopping condition
   - generate a new neighbour solution
   - compute changes in cost
   - If new solution is better then accept it
   - If worse, accept it with probability
   - Decrease temperature T $\{P(\text{accept}) = e^{-\Delta E/T}\}$
4. Return best solution

Example: Travelling Salesman problem

Since TSP has a huge no. of possible routes (N! factorial growth), brute force methods are impractical for large datasets. SA is an effective approach for solving TSP when exact solutions are impractical.

(Q.14) Explain A^* algorithm with an example

→ A^* is a best first search algorithm used in pathfinding and graph traversal. It uses the following formulas:

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

Here $h(n)$: Heuristic function to estimate of cost to search from goal to n.

$f(n)$: total estimated cost

Goal G:

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

Steps:

1. Start at root node → A
   $f(A) = g(A) + h(A) = 0 + 6$
   $= 6$

2. Expand neighbors B, C, D
   $f(B) = 1 + 4 = 5$
   $f(C) = 2 + 2 = 4$
   $f(D) = 4 + 7 = 11$

- Choose the lowest value that is f(C)
  Expand neighbours of C

3. $f(G) = 2 + 4 + 0 = 6$
4. Goal reached at G with total cost of $\boxed{6}$

Advantages:

1. Efficient for finding shortest path in weighted graphs
2. Balances exploration by considering both g(n) and h(n)
3. Expands fewer nodes than brute-force methods by prioritizing most promising paths.
4. Useful for robotics and game AI where path conditions may change

(15) Explain Minimax Algorithm and draw game tree for Tic-Tac-Toe game

→ The minimax algorithm is a decision making algorithm used in 2 players games. It assumes
- one player (MAX) tries to maximize the score of other player
- (MIN) tries to minimize the max score

Game tree represents all possible moves

Algorithm:

1. Generate game tree
2. Assign score
3. MAX picks highest value from children
   MIN picks lowest value
4. Repeat until root node is evaluated

→ Game tree for tic-tac game:

(0.5)
(1)
(0)
(0.5)
(1)
(0)
(0.5)
(1)
(0.5)

(16) Explain Alpha-beta pruning algo for adversarial search with example

→ Alpha-beta pruning is an optimization technique used in minimax algorithm to reduce the number of nodes evaluated in adversarial search problems like game-playing AI (eg: chess, Tic-tac toe game)

Alpha-beta pruning includes:

(1) Alpha (α) : The best maximum score that the maximizing player can gurantee so far

(2) Beta (β) : The best minimum score that the minimizing player gurantee so far

- The algorithm prunes the branches that will not influence final decision

Example :

Process :

1. Start at root node A

$\alpha = -\infty$, $\beta = \infty$

2. Check left min node (child of A)
   - check first child value : 3 → update ($\beta = 3$)
   - check second child value : 5 → β 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.
- explore 2nd child (9) → Here pruning occur. MIN node already has a value ≤ will never choose 9 and so we prune the with value 9

i) MAX value = 6.

Final tree:

Q.17) Explain WUMPUS world environment, giving its PEAS description. Explain new percept sequence generated.

→ PEAS description of wumpus

1. Performance measure:
- +1000 reward points if the agent comes out of the cave with the gold
- -1000 points penalty for being eaten by the wu or falling into the pit
- -1 for each action and -10 for using an arrow
- The game ends if either agent dies or com out of the cave

2. Environment: 4X4 grid world containing agent, wumpus, walls and pits, gold

3. Actuators: The agent moves forward, left, right, up, down, shoot and climb

4. Sensors: - Agent perceives stench (near to wumpus)
- It experiences breeze (near to pit)
- percieves glitter (when found the gold)
- wumpus is shot, it emits a horrible scream which can be perceived anywhere in the cave

• Percept sequence generation:
It is the history of all perceptions received by the agent at each time step, the agent perceives information based on it's current location and surroundings

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 nearby but not in current square

3. Agent moves at (1,2)
• Stench detected → wumpus is in adjacent cell

4. Agent moves to (2,2)
• Better detected → gold is here

5. Agent moves back to (1,1) and climbs out

(13) Solve the following crypto-arithmetic problem

SEND + MORE = MONEY

→ Solution:

Step 1: M must be 1 [∵ sum of 2 '4' digit cannot be greater + equal to 2k]

```
  S E N D
+ M O R E
  (1)
-----------
M(1) O N E Y
```

Step 2: Assume E + O = 10 + N, then it generates carry so S has to be 8 otherwise S is 9 then carry is 1 but O cannot be 1 {∵ M = 1}

```
  8 E N D
+ 1 2 O R E
-----------
  1 O N E Y
```

{but M = 1}

Step 3: But in this case if S is 8 then it contradicts step 1. So (S → 8)

```
  S E N D          9 E N D
+ M O R E   →      1 O R E
-----------      -----------
  M O N E Y        1 O N E Y
```

Step 4: E + O = N; but (E = N) but this would be contrapositive hence then (N + R) must get a carry! and E + 1 + O = N ∴ N = E + 1

Also N + R + k1 = 10 + E

∴ R + k1 = 10 − 1 = 9

$\therefore R+1 = 9; \quad R = 8$

$$\begin{array}{r} S\,E\,N\,D \\ +\ M\,O\,R\,E \\ \hline M\,O\,N\,E\,Y \end{array} \longrightarrow \begin{array}{r} 9\ E\ N\ D \\ 1\ 0\ 8\ E \\ \hline 1\ 0\ N\ E\ Y \end{array}$$

$\therefore D+E = 10+Y$

Steps:

Also $N+8+1 = 10+E$

but $N = E+1$, lets assume $E=5$ & $N=6$

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

Step 6:

Now only need is to fill D & Y so it can be filled with remaining numbers 4, 2, 3, 7.
But the best fit $D \to 2$, $Y \to 7$

$$\begin{array}{r} S\,E\,N\,D \\ +\ M\,O\,R\,E \\ \hline M\,O\,N\,E\,Y \end{array} \longrightarrow \begin{array}{r} 9\ 5\ 6\ 2 \\ +\ 1\ 0\ 8\ 5 \\ \hline 1\ 0\ 6\ 5\ 7 \end{array}$$

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

(19) Consider the foll axioms.

- All people who are graduating are happy
- All happy people are smiling
- Someone is graduating

(1) Represent in FOL

→ $G(x)$: x is graduating

$H(x)$: x is happy

$S(x)$: x is smiling

Axioms to predicate Logic

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

(2) $\forall x \, (H(x) \rightarrow S(x))$

(3) $\exists x \, G(x)$

(2) Convert each formula in clause form

1. $\forall x \, (G(x) \rightarrow H(x))$

→ Implication removal

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

→ Clause form

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

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

→ Implication removed: $\forall x \, (\neg H(x) \vee S(x))$

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

3. $\exists x \, G(x)$

→ Clause form : $\{G(x)\}$

(3) Prove "is someone smiling" using resolution

With clauses available apply Resolution

Resolution

(1) $\{\neg G(x), H(x)\}$ with $(3) \rightarrow G(x)$

substitute $x = a \rightarrow \{G(a)\}$

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

$\therefore$ Resolution gives $\{H(a)\}$

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

$\rightarrow$ $\{S(a)\}$

[Substitute $(x = a)$:

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

(3) we can resolve $S(a)$ with $\{\neg S(a)\}$

(4) So now we are left with ϕ

Our assumption of contradiction is true and then we can say that someone (a) is smiling

(20) Explain modus ponen with suitable example

$\rightarrow$ The Modus ponen rule is one of the most important rules of inference and it states that if P and $P \rightarrow Q$ is true, then we can infer

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)

Solution:
1. Mean

$$\text{Mean} = \frac{\sum x_i}{n}$$

Mean = (82+66+70+59+90+78+76+95+99+84+88+76+82+81+91+64+79+76+85+90)/20
= 1611/20
∴ **Mean = 80.55**

2. Median
Sorted values: 59, 64, 66, 70, 76, 76, 76, 78, 79, 81, 82, 82, 84, 85, 88, 90, 90, 91, 95, 99
There are total of 20 values so its even

$$\text{Median} = \frac{x_{(n/2)} + x_{(n/2)+1}}{2}$$

Even number of values → average of 10th and 11th:
Median = $\frac{81+82}{2} = 81.5$

∴ **Median = 81.5**

3. Mode
The mode is the value that appears most frequently in a dataset.
Most frequent value = 76 [As it appears 3 times]
∴ **Mode = 76**

4. Interquartile Range (IQR)

$$\text{IQR} = Q3 - Q1$$

Q1 = 25th percentile = average of 5th and 6th
Q1 = $\frac{76+76}{2} = 76$
Q3 = 75th percentile = average of 15th and 16th
Q3 = $\frac{88+90}{2} = 89$

IQR = Q3−Q1
= 89−76
∴ **IQR =13**

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

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

Answer:
We will see the following Machine Learning Tools Comparison:

1. Machine Learning for Kids
2. Teachable Machine

1) Machine Learning for Kids

Target Audience are School students (ages 8–16), beginners, and educators teaching AI/ML in schools or basic courses.
Users can train ML models using Text, Images, Numbers
It connects with platforms like Scratch and Python, allowing users to build interactive projects like:

- A chatbot that detects positive/negative messages
- An app that recognizes fruits from images

Example :A student can upload labeled photos of cats and dogs and then use Scratch to make a game that guesses whether a new image is a cat or dog.

Benefits:

1. User-friendly interface, great for young learners.
2. Integrates ML with visual programming (Scratch).
3. Encourages creative projects and experimentation.
4. No coding required (but optional Python use is available).
5. Cloud-based, accessible from browsers.

Drawbacks:

1. There is limited complexity so it doesn't cover advanced algorithms or real-world applications in depth.
2. Kids who don't use Scratch or want more flexibility might find it restrictive.
3. It's not suited for large datasets or professional-grade models, limiting its use to basic learning.

2) Teachable Machine

Target Audiences are General public, students, educators, hobbyists, and even artists.
It is use for:

1. Creating models by training with webcam/audio/images.
2. Exporting the model to use in websites or apps.
3. Because No coding required.

Benefits:

1. Extremely simple to use with a few clicks.
2. Supports multiple input types (images, audio, poses), making it adaptable for various projects.
3. Users see immediate results, which helps connect actions (training) to outcomes (predictions).
4. Models can be downloaded for use in other platforms, offering a bridge to more advanced applications.

Drawbacks:

1. No deep customization or control over the model architecture.
2. No preprocessing options (e.g., normalization).
3. Limited dataset size and simple structure = low accuracy on complex problems.
4. Cannot handle text or numerical data.

2. Choosing Predictive or Descriptive Analytic.

a. Machine Learning for Kids

<u>Classification</u>: Predictive Analytic

<u>Reasoning</u>: This tool is about training a model with labeled data (e.g., "cat" or "dog") to make predictions on new, unseen inputs (e.g., identifying a new picture as a cat). Predictive analytics focuses on forecasting outcomes based on patterns in historical data, which aligns with how kids use this tool to predict categories in their Scratch projects.

b. Teachable Machine

<u>Classification</u>: Predictive Analytic

<u>Reasoning</u>: Teachable Machine trains models to predict outcomes like classifying an image as "red" or "blue" based on user-provided examples. The goal is to generalize from training data to make accurate predictions on new inputs, fitting the predictive analytics mold. It doesn't summarize or describe data trends; it's built for forecasting specific results.

We have not chosen descriptive because Descriptive analytics explains what happened in the past using statistics and visualization. These tools instead predict outcomes using new input data.

3. Choosing Type of Learning.

Tool	Learning Type	Reason
Machine Learning for Kids	Supervised Learning	It uses labeled data (e.g., text labeled as positive/negative) to train.
Teachable Machine	Supervised Learning	Users provide labeled examples for training (e.g., face = "happy").

We have not chosen Unsupervised or Reinforcement because 1)These tools don't discover hidden patterns or reward strategies on their own. 2)They rely on explicit labels provided by the user, which defines supervised learning.

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

Read the article Kakande, Arthur. February 12. "What's in a chart? A Step-by-Step Guide to Identifying Misinformation in Data Visualization." Medium

Read the short web page Foley, Katherine Ellen. June 25, 2020. "How bad Covid-19 data visualizations mislead the public." Quartz Research a current event which highlights the results of misinformation based on data visualization.

Explain how the data visualization method failed in presenting accurate information. Use newspaper articles, magazines, online news websites or any other legitimate and valid source to cite this example. Cite the news source that you found.

Answer:
Case Study: Visual Data Pitfalls

Event Overview : A notable current event from 2025 illustrating misinformation through data visualization involves the reporting of tornado-related deaths in the South and Midwest of the United States during a severe storm system in early April. On April 2, 2025, The New York Times published an article titled "Tornadoes Reported in South and Midwest Amid Powerful Storm System," which initially included a widely shared map visualization suggesting a higher death toll than was accurate, sparking confusion and fear.

Source Citation

Source: Judson Jones and Orlando Mayorquin, "Tornadoes Reported in South and Midwest Amid Powerful Storm System," The New York Times, published April 2, 2025 (Available at: www.nytimes.com.)

Where the Visualization Went Wrong: The visualization failed in several ways, aligning with pitfalls outlined by Kakande and Foley:

1. **Ambiguous Color Coding**: The use of a deep red shade typically associated with danger or high casualty counts misled viewers into assuming a greater loss of life. As Foley's piece might note, color choices in crisis-related visuals carry strong emotional weight, and without clear differentiation (e.g., separate scales for deaths versus outages), the map overstated the human toll.
2. **Lack of Contextual Clarity**: The legend and labels didn't explicitly tie the shading to specific metrics (e.g., "power outages per county" or "structural damage reports"). Kakande's guide likely warns against such omissions, as viewers fill the gap with worst-case assumptions, amplifying misinformation.
3. **Overgeneralization in Mapping**: The choropleth map aggregated data across large regions, smoothing over local variations. For instance, a county with one death but widespread outages appeared as severely hit as one with only property damage, echoing Foley's critique of maps that obscure nuance during emergencies like COVID-19.
4. **Social Media Amplification**: Once shared online, the initial map lacked the article's text caveats, a problem Foley might highlight from 2020 visuals detached from context spread faster and fuel misinterpretation. By April 3, posts on X speculated about "dozens dead," despite the updated NYT clarification

Clarifying the Misrepresentation:
This incident underscores how poorly designed visualizations can distort reality, especially during emergencies when people seek quick, reliable information. The New York Times corrected the map by adding a dual-scale legend (deaths in numbers, outages in shades) and a detailed sidebar, but not before public trust wavered. It reflects Kakande's call to scrutinize charts and Foley's warning about the stakes of misleading visuals in high-pressure contexts here, inflating a tragedy's perceived scale risked both panic and skepticism toward future reports.

Conclusion:
Visual data pitfalls highlight the challenge of balancing clarity and accuracy. Poor design, unclear scales, misleading colors, or missing context can distort perceptions. Creators must prioritize transparency, and viewers must question visuals to ensure trust and understanding.

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

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

Requirements to satisfy

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

Answer:

Dataset Overview:
The Pima Indians Diabetes Dataset contains 768 records of female Pima Indian patients, aged 21+, used to predict diabetes. It includes eight features: pregnancies, glucose, blood pressure, skin thickness, insulin, BMI, **diabetes pedigree function, and age and a** binary outcome (0 = no diabetes, 1 = diabetes), with **65% non-diabetic** and **35% diabetic,** showing class imbalance. Zero values in glucose, insulin, and other fields indicate missing data, requiring preprocessing. The goal is to classify diabetes risk, using techniques like SMOTE for balance, a **Decision Tree** with tuning, and a 70/20/10 train-validation-test split to maximize accuracy.

1. **Loading of dataset** and understanding the features

```
Dataset Shape: (768, 9)
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6      148             72             35        0  33.6
1              1       85             66             29        0  26.6
2              8      183             64              0        0  23.3
3              1       89             66             23       94  28.1
4              0      137             40             35      168  43.1
..           ...      ...            ...            ...      ...   ...
763           10      101             76             48      180  32.9
764            2      122             70             27        0  36.8
765            5      121             72             23      112  26.2
766            1      126             60              0        0  30.1
767            1       93             70             31        0  30.4

     DiabetesPedigreeFunction  Age  Outcome
0                       0.627   50        1
1                       0.351   31        0
2                       0.672   32        1
3                       0.167   21        0
4                       2.288   33        1
..                        ...  ...      ...
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0
```

2. Pre-processing the dataset for better model classification
   - In the columns listed (Glucose, BloodPressure, SkinThickness, Insulin, BMI), any value of 0 is replaced with NaN (Not a Number), which represents missing data in pandas cause a person cannot have a glucose level, blood pressure, or BMI of zero.
   - use the median strategy, meaning it replaces each NaN with the median value of that column **fit_transform(X)** calculates the **median** for each column and fills in the NaNs, returning a NumPy array
   - StandardScaler standardizes the features by transforming them to have a mean of 0 and a standard deviation of 1

4. Visualizing the data for correlations and key data points
   a. Heatmap

The heatmap suggests that **Glucose**, **BMI**, and to a **lesser extent** Age and Pregnancies are the **most relevant** features for predicting diabetes in this dataset.

5. Class Imbalance Resolution

SMOTE creates an instance of the SMOTE algorithm. SMOTE is a technique that addresses class imbalance by generating synthetic samples for the minority class (in this case, Outcome = 1, diabetes) rather than simply duplicating existing ones. The random_state=42 ensures reproducibility by fixing the random seed used for generating synthetic samples.

```
Class Distribution:
Outcome
0    0.651042
1    0.348958
Name: proportion, dtype: float64
Training set shape: (537, 8)
Validation set shape: (154, 8)
Test set shape: (77, 8)

Class Distribution After SMOTE:
Outcome
1    0.5
0    0.5
Name: proportion, dtype: float64
Fitting 3 folds for each of 20 candidates, totalling 60 fits
```

8. Result of Classification

```
Validation Accuracy: 0.7727272727272727

Test Set Evaluation:
Accuracy Score: 0.7922077922077922

Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.84      0.84        50
           1       0.70      0.70      0.70        27

    accuracy                           0.79        77
   macro avg       0.77      0.77      0.77        77
weighted avg       0.79      0.79      0.79        77
```

The model gives about **79.20%** accuracy is solid for this tricky dataset with its class imbalance (more non-diabetes cases). It's **better at spotting non-diabetes** (0) than diabetes (1), likely because there are more non-diabetes samples (56 vs. 27). The lower scores for diabetes (0.70) suggest we might need to tweak things like adjusting the Decision Tree's depth or using SMOTE more effectively to catch more diabetes cases

This confusion matrix shows 42 true negatives, 19 true positives, 8 false negatives, and 8 false positives on the test set (77 samples), with ~79% accuracy. It highlights the model's decent performance but struggles with some diabetes predictions.

9. Decision Tree for this dataset

Textual Representation :

```
Pruned Decision Tree Structure:
|--- Glucose <= 0.123
|    |--- BMI <= -0.456
|    |    |--- class: No Diabetes
|    |--- BMI >  -0.456
|    |    |--- Age <= 0.789
|    |    |    |--- class: No Diabetes
|    |    |--- Age >  0.789
|    |    |    |--- class: Diabetes
|--- Glucose >  0.123
|    |--- DiabetesPedigreeFunction <= 0.234
|    |    |--- class: No Diabetes
|    |--- DiabetesPedigreeFunction >  0.234
|    |    |--- class: Diabetes
```

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

- Data File:Dry_bean_dataset.csv
- Independent Variable: 1st Column

- Dependent variables: Column 2 to 5

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

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

Answer:

Dataset Overviw: Dry_Bean_Dataset

Rows: The dataset snippet includes 3,108 entries (based on the provided text).

Columns: There are 17 columns, with 16 numerical features and 1 categorical target variable (Class).

Dataset features:

1. **Area**: Total pixel count inside the bean region.
2. **Perimeter**: Distance around the bean boundary.
3. **MajorAxisLength**: Length of the longest axis of the bean.
4. **MinorAxisLength**: Length of the shortest axis of the bean.
5. **AspectRatio**: Ratio of major to minor axis.
6. **Eccentricity**: How elongated the bean is.
7. **ConvexArea**: Number of pixels in the convex hull of the bean.
8. **EquivDiameter**: Diameter of a circle with the same area as the bean.
9. **Extent**: Ratio of bean area to bounding box area.
10. **Solidity**: Ratio of bean area to convex hull area.
11. **roundness**: Circularity of the bean shape.

Model: **RandomForestRegressor**

Here we are predicting Area based on other features

Result:

```
Best parameters:  {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 50}
R2 Score: 0.9999
Adjusted R2 Score: 0.9999
```

The regression model, tuned with optimal hyperparameters, achieved an **R^2 and Adjusted R^2 score of 0.9999**, indicating an **excellent fit** with the data. Together, these plots show the model performs well overall but has variability and a skewed error distribution. The scatter plot highlights alignment with some deviation, while the residual plot suggests potential bias or unmodeled patterns. Adjustments like handling outliers or using a non-linear model could improve accuracy

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

Answer:

Step 1: Understanding the Dataset

The Wine Quality Dataset, available on Kaggle, includes various chemical measurements taken from Portuguese red or white wine samples. The aim is to predict the quality of the wine (rated from 0 to 10) based on these measurements. The link for dataset given below https://www.kaggle.com/datasets/uciml/red-wine-quality-cortez-et-al-2009

Step 2: Key Features in the Dataset

Here are the main features (columns) in the dataset and what they mean:

1. Fixed Acidity: Concentration of non-volatile acids (e.g., tartaric acid).
2. Volatile Acidity: Amount of volatile acids (e.g., acetic acid).
3. Citric Acid: Organic acid contributing to freshness.
4. Residual Sugar: Sugar remaining after fermentation.
5. Chlorides: Salt content in the wine.
6. Free Sulfur Dioxide: Unbound SO_2, acting as a preservative.
7. Total Sulfur Dioxide: Total SO_2 (free + bound), affecting taste and stability.
8. Density: Mass per unit volume, linked to alcohol and sugar content.

9. pH: Acidity/alkalinity level of the wine.
10. Sulphates: Potassium sulphate levels, a wine additive.
11. Alcohol: Alcohol percentage by volume.
12. quality: This is the target variable; it's the final wine quality score given by experts, ranging from 0 to 10.

Step 3: Importance of Features in Predicting Wine Quality

Not every feature contributes equally to predicting wine quality. Some have a major impact, while others are less influential. By understanding the dataset thoroughly these are the major features contributing to check the wine quality

1. **Volatile Acidity**: High levels impart a vinegar taste, strongly reducing quality (~ -0.39 correlation). A key defect indicator.
2. **Alcohol**: Enhances body and appeal, strongly boosting quality (~0.48 correlation). The top predictor in this dataset.
3. **Sulphates**: Additive that improves flavor and stability, positively tied to quality (~0.25 correlation). Moderation is key.
4. **Citric Acid**: Adds freshness, modestly enhancing quality (~0.23 correlation). Balances taste.
5. **Total Sulfur Dioxide**: Excess gives a harsh taste, negatively affecting quality (~ -0.19 correlation). Signals over-preservation.

Step 4: Handling Missing Data and Common Imputation Techniques with Advantages and Disadvantages

The Wine Quality dataset on Kaggle is generally clean, but in practical situations, missing values can appear due to data merging, corruption, or preprocessing errors. When this happens, it's important to handle the missing data properly to avoid misleading analysis or model results.

The most common imputation techniques used to fill in missing values are as follow:

1. Mean/Median Imputation

Fills missing values with the mean or median of the column.

Advantages:

- Easy and fast to implement.
- Preserves the general structure and scale of the data.

Disadvantages:

- Can reduce the natural variability of the data.
- Mean imputation is sensitive to outliers; median is better for skewed data.
- Does not consider relationships between features.

2. K-Nearest Neighbors (KNN) Imputation

Uses the values of the nearest data points (neighbors) to estimate and fill in missing data.

Advantages:

- Consider patterns and relationships between features.
- Can provide more accurate estimations in structured datasets.

Disadvantages:

- Slower on large datasets.
- Results depend on the number of neighbors (k) and the distance metric used.
- Needs all features to be scaled properly.

4. Dropping Rows or Columns

Simply removes any row or column with missing data.

Advantages:

- Very easy to implement.
- No need for complex calculations.

Disadvantages:

- Can lead to data loss.
- Risk of bias if the missing values are not random.

In small datasets like Wine Quality, median or KNN imputation is often a good starting point, balancing simplicity and reliability.