Experiment 1

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

Theory:

Data Science is a multidisciplinary field that involves collecting, processing, analyzing, and visualizing data to extract meaningful insights. It uses techniques from statistics, machine learning, and programming to solve real-world problems. One of the essential steps in data science is data preparation, which ensures that raw data is cleaned and structured for analysis.

Key Steps in Data Preparation Using Pandas:

1. Loading Data
2. Exploring Data
3. Handling Missing Values
4. Data Transformation
5. Feature Engineering

Dataset Overview:

The dataset consists of 12 columns, each providing key insights into retail products, their pricing, visibility, and sales performance across different outlets. Below is a breakdown of the dataset's columns and their significance:

- **Item_Identifier:** A unique code assigned to each product, helping distinguish different items in the inventory.
- **Item_Weight:** The weight of a product, which can influence logistics, storage, and customer purchasing decisions.
- **Item_Fat_Content:** Categorizes products as Low Fat or Regular Fat, reflecting their nutritional composition and target consumers.
- **Item_Visibility:** Measures how prominently an item is displayed on store shelves, impacting its likelihood of being purchased.
- **Item_Type:** Classifies products into broad categories like Dairy, Beverages, or Snacks, aiding in trend analysis across different product segments.
- **Item_MRP:** The maximum retail price, which plays a crucial role in determining affordability and customer demand.
- **Outlet_Identifier:** A unique code assigned to each retail outlet, linking products to specific store locations.
- **Outlet_Establishment_Year:** The year an outlet was established, useful for analyzing how store maturity influences sales performance.
- **Outlet_Size:** Defines store sizes as Small, Medium, or Large, affecting customer foot traffic and product demand.

- **Outlet_Location_Type:** Indicates whether a store is located in an Urban, Suburban, or Tier 3 area, capturing demographic influences on sales.
- **Outlet_Type:** Differentiates between store formats, such as Grocery Stores and Supermarkets, highlighting variations in business models.
- **Item_Outlet_Sales:** The target variable, representing the total revenue generated by a product at a particular outlet.

Problem Statement:

The given dataset provides comprehensive details about retail product sales, focusing on the relationship between product attributes, outlet characteristics, and sales performance. This analysis aims to address the following key objectives:

- **Product Performance**: Identifying products or product types that drive the highest sales and those underperforming.
- **Outlet Insights**: Understanding the impact of outlet size, location, and type on overall sales performance.
- **Pricing Analysis**: Investigating how pricing strategies, such as maximum retail price (MRP), influence customer purchasing behavior.
- **Possible sales Prediction**: Developing models to predict item-level sales and provide actionable insights for inventory and pricing strategies.

By preprocessing the dataset and applying statistical analysis, the goal is to extract meaningful patterns that can guide data-driven decisions in retail operations.

Steps:

1. Load Data in Pandas
   The first step is to load our excel sheet or dataset using read_csv method.

Load Data in Pandas

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

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_
0	FDA15	9.300	Low Fat	0.016047	Dairy	249.8092	OUT049	1999	Me
1	DRC01	5.920	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009	Me
2	FDN15	17.500	Low Fat	0.016760	Meat	141.6180	OUT049	1999	Me
3	FDX07	19.200	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010	1998	
4	NCD19	8.930	Low Fat	0.000000	Household	53.8614	OUT013	1987	
...	...	...	...	...	...	...	...	...	
8518	FDF22	6.865	Low Fat	0.056783	Snack Foods	214.5218	OUT013	1987	
8519	FDS36	8.380	Regular	0.046982	Baking Goods	108.1570	OUT045	2002	
8520	NCJ29	10.600	Low Fat	0.035186	Health and Hygiene	85.1224	OUT035	2004	S
8521	FDN46	7.210	Regular	0.145221	Snack Foods	103.1332	OUT018	2009	Me
8522	DRG01	14.800	Low Fat	0.044878	Soft Drinks	75.4670	OUT046	1997	S

2. Description of the dataset
   df.info() provides information about the dataset like what are the different columns or features present in the dataset, what are their respective data types etc.

Description of the dataset

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8523 entries, 0 to 8522
Data columns (total 12 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_Establishment_Year  8523 non-null   int64
 8   Outlet_Size                6113 non-null   object
 9   Outlet_Location_Type       8523 non-null   object
 10  Outlet_Type                8523 non-null   object
 11  Item_Outlet_Sales          8523 non-null   float64
dtypes: float64(4), int64(1), object(7)
memory usage: 799.2+ KB
```

```
df.describe()
```

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Item_Outlet_Sales
count	7060.000000	8523.000000	8523.000000	8523.000000	8523.000000
mean	12.857645	0.066132	140.992782	1997.831867	2181.288914
std	4.643456	0.051598	62.275067	8.371760	1706.499616
min	4.555000	0.000000	31.290000	1985.000000	33.290000
25%	8.773750	0.026989	93.826500	1987.000000	834.247400
50%	12.600000	0.053931	143.012800	1999.000000	1794.331000
75%	16.850000	0.094585	185.643700	2004.000000	3101.296400
max	21.350000	0.328391	266.888400	2009.000000	13086.964800

The describe method helps us understand the instances or the value in every row. It gives information like number of values, mean, max value for every column.

3. Drop columns that aren't useful
   It may not be necessary that all features present in our dataset will contribute to our analysis. There would be columns which are not required and which increase the size of data unnecessarily. In such cases we drop entire such columns. In our dataset the feature "Outlet_Establishment_Year" is not useful, so we drop it.

Drop columns that aren't useful

```
[5]  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
```

4. Drop rows with maximum missing values.
   In our dataset there are instances that have some missing values. We want a proper dataset where there are no missing values. In order to do this, we simply remove the rows with null values in them. To do this we use the dropna method. After deleting all null values rows, we observe that the number of non null values for every feature is now the same.

Drop rows with maximum missing values

```
[7]  df = df.dropna()
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 4650 entries, 0 to 8522
Data columns (total 11 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Item_Identifier       4650 non-null   object
 1   Item_Weight           4650 non-null   float64
 2   Item_Fat_Content      4650 non-null   object
 3   Item_Visibility       4650 non-null   float64
 4   Item_Type             4650 non-null   object
 5   Item_MRP              4650 non-null   float64
 6   Outlet_Identifier     4650 non-null   object
 7   Outlet_Size           4650 non-null   object
 8   Outlet_Location_Type  4650 non-null   object
 9   Outlet_Type           4650 non-null   object
 10  Item_Outlet_Sales     4650 non-null   float64
dtypes: float64(4), object(7)
memory usage: 435.9+ KB
```

5. Take care of missing values

```
print(df["Item_Weight"].isnull().sum())  # Should show 0 missing values
```

```
1463
```

Analysis of our dataset shows that “Item_Weight” has 1463 missing values. This can cause problems in our analysis. So to address this, we will first categorize items by their types(as similar items may have similar weights), then we find their respective mean and fill null values with the mean.

```
df['Item_Weight'] = df.groupby('Item_Type')['Item_Weight'].transform(lambda x: x.fillna(x.mean()))
print(df["Item_Weight"].isnull().sum())  # Should show 0 missing values
```

```
0
```

6. Find out outliers (manually)
   We will use the IQR method to calculate outliers for “Item_Outlet_Sale”:
   First Quartile Q1 = 834.2474
   Second Quartile Q2 = 3101.2964

IQR = Q3-Q1 = 2267.049

Lower Bound = 834.2474 - 1.5x2267.049 = -2566.3261
Upper Bound = 3101.2964 - 1.5x2267.049 = 6501.8699
So any value less than -2566.3261 or greater than 6501.8699 can be considered as outliers. (Last column represents Item Outlet Sales)

Item_Iden	Item_Weig	Item_Fat_	Item_Visib	Item_Type	Item_MRP	Outlet_Ide	Outlet_Est	Outlet_Siz	Outlet_Lo	Outlet_Ty	Item_Outlet_Sales
FDA46	13.6	Low Fat	0.117818	Snack Foo	192.9136	OUT049	1999	Medium	Tier 1	Supermark	2527.377
FDC02	21.35	Low Fat	0.069103	Canned	259.9278	OUT018	2009	Medium	Tier 3	Supermark	6768.523
NCP30	20.5	Low Fat	0.032835	Household	40.2822	OUT045	2002		Tier 2	Supermark	707.0796
FDY25		Low Fat	0.03381	Canned	180.5976	OUT027	1985	Medium	Tier 3	Supermark	7968.294
NCH54	13.5	Low Fat	0.072669	Household	160.292	OUT046	1997	Small	Tier 1	Supermark	1438.128
NCR53		Low Fat	0.144338	Health and	224.4404	OUT027	1985	Medium	Tier 3	Supermark	6976.252
FDY56	16.35	Regular	0.062764	Fruits and	227.6062	OUT017	2007		Tier 2	Supermark	7222.598
FDH19		Low Fat	0.032928	Meat	173.1738	OUT027	1985	Medium	Tier 3	Supermark	7298.5
FDY55	16.75	Low Fat	0.081253	Fruits and	256.4988	OUT013	1987	High	Tier 3	Supermark	7452.965
FDO23	17.85	Low Fat	0.147024	Breads	93.7436	OUT018	2009	Medium	Tier 3	Supermark	1134.523
DRE60	9.395	Low Fat	0.159658	Soft Drinks	224.972	OUT045	2002		Tier 2	Supermark	7696.648
DRP47	15.75	Low Fat	0.141399	Hard Drink	250.5382	OUT017	2007		Tier 2	Supermark	2775.72
FDO55	16.2	Low Fat	0.035984	Fruits and	260.6278	OUT045	2002		Tier 2	Supermark	4425.573
FDN58		Regular	0.056597	Snack Foo	230.9984	OUT027	1985	Medium	Tier 3	Supermark	9267.936
FDI44	16.1	Low Fat	0.100389	Fruits and	76.0328	OUT049	1999	Medium	Tier 1	Supermark	1853.587
FDW56		Low Fat	0.070557	Fruits and	191.2162	OUT027	1985	Medium	Tier 3	Supermark	7504.232
FDE45	12.1	Low Fat	0.040522	Fruits and	178.5002	OUT018	2009	Medium	Tier 3	Supermark	3552.106
FDR35		Low Fat	0.020597	Breads	200.0742	OUT027	1985	Medium	Tier 3	Supermark	8958.339

7. Standardization and Normalization
   Standardization and normalization are crucial in data preprocessing because they bring features to a common scale, improving model performance and convergence. Standardization (Z-score scaling) centers data around zero with unit variance, while Normalization (Min-Max scaling) rescales data to a fixed range (e.g., 0 to 1), ensuring fair weight distribution across features.

Standardization and Normalization of columns

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

df = pd.read_csv("/content/drive/MyDrive/market_data.csv")

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    -0.766217        -0.970732  1.747454           0.910601
1    -1.494175        -0.908111 -1.489023          -1.018440
2     0.999834        -0.956917  0.010040          -0.049238
3     1.365966        -1.281758  0.660050          -0.849103
4    -0.845905        -1.281758 -1.399220          -0.695373

Normalized Data:
   Item_Weight  Item_Visibility  Item_MRP  Item_Outlet_Sales
0     0.282525         0.048866  0.927507           0.283587
1     0.081274         0.058705  0.072068           0.031419
2     0.770765         0.051037  0.468288           0.158115
3     0.871986         0.000000  0.640093           0.053555
4     0.260494         0.000000  0.095805           0.073651
```

Conclusion:

In this experiment, we explored the fundamental steps of data preparation using Pandas in Data Science. We loaded and analyzed a retail dataset, identified missing values, and handled them effectively. We also detected and removed outliers using the IQR method to ensure clean and reliable data. Additionally, we discussed feature selection, data transformation, and the importance of standardization and normalization in preparing data for further analysis.

Experiment 2

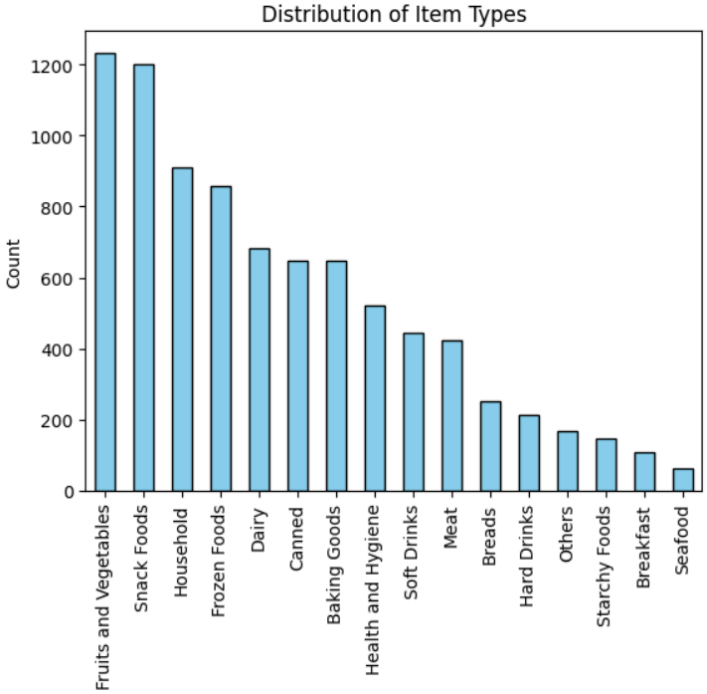
Aim: Data Visualization/ Exploratory data Analysis using Matplotlib and Seaborn.

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.

1. **Bar graph and contingency table using any two features:**

   a. Bar Graph for distribution of item type.

```
import matplotlib.pyplot as plt

df['Item_Type'].value_counts().plot(kind='bar', color='skyblue', edgecolor='black')

plt.xlabel("Item Type")
plt.ylabel("Count")
plt.title("Distribution of Item Types")
plt.xticks(rotation=90)
plt.show()
```



From the graph, we observe that the majority of items fall under the category of "Fruits and Vegetables," with a count of approximately 1,200. Similarly, the "Snack Foods" category also comprises a comparable number of items, indicating a balanced

distribution between these two categories. Rest of the items have a count of less than 1000. “Seafood” accounts for less than approximately 100 items, which is the lowest.

b. Contingency Table for Item type and Outlet type
A contingency table (or cross-tabulation table) displays the frequency distribution of two categorical variables in a dataset. It helps in understanding the relationship between the two features

```
import pandas as pd

contingency_table = pd.crosstab(df['Item_Type'], df['Outlet_Type'])
print(contingency_table)
```

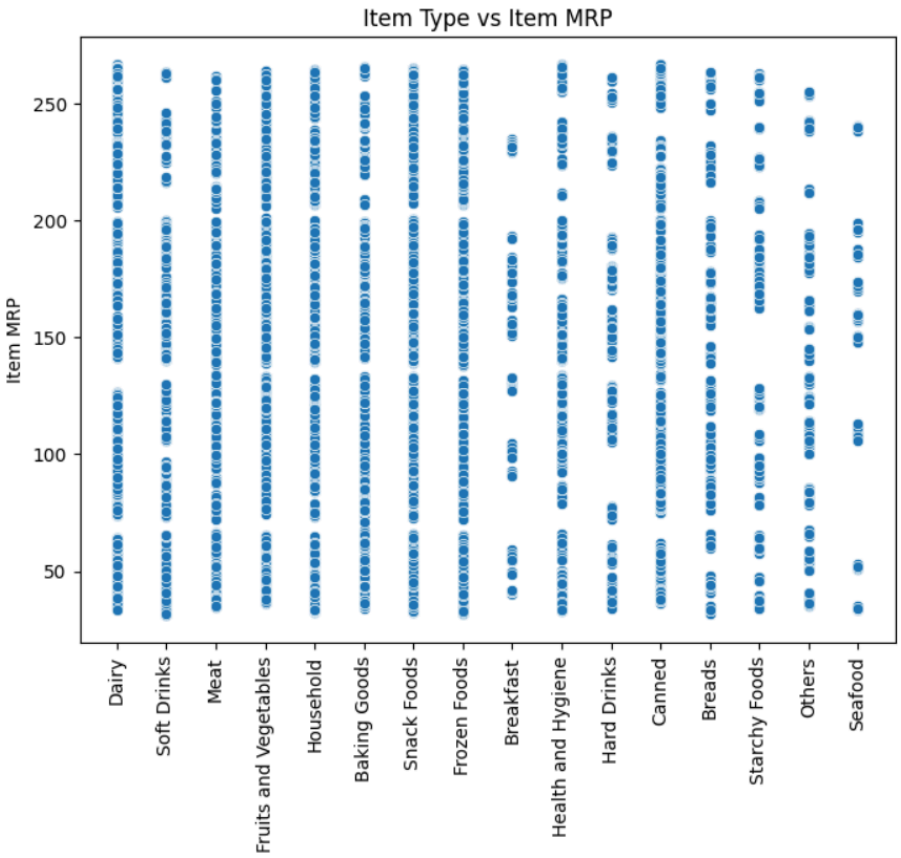
```
Outlet_Type           Grocery Store  Supermarket Type1  Supermarket Type2  \
Item_Type
Baking Goods                     85                426                 68
Breads                           33                160                 27
Breakfast                        19                 68                 12
Canned                           73                426                 78
Dairy                            92                450                 73
Frozen Foods                    103                572                 92
Fruits and Vegetables           152                805                135
Hard Drinks                      24                145                 22
Health and Hygiene               67                335                 58
Household                       119                597                 95
Meat                             66                257                 46
Others                           27                107                 20
Seafood                          10                 40                  7
Snack Foods                     146                785                132
Soft Drinks                      54                300                 46
Starchy Foods                    13                104                 17

Outlet_Type           Supermarket Type3
Item_Type
Baking Goods                         69
Breads                               31
Breakfast                            11
Canned                               72
Dairy                                67
Frozen Foods                         89
Fruits and Vegetables               140
Hard Drinks                          23
Health and Hygiene                   60
Household                            99
Meat                                 56
Others                               15
Seafood                               7
Snack Foods                         137
Soft Drinks                          45
Starchy Foods                        14
```

From the table, it is evident that Supermarket Type 1 has the highest variety and number of items across all categories compared to other outlet types. Fruits and Vegetables and Snack Foods are the dominant categories in most outlets, indicating their popularity. Smaller categories like Seafood and Starchy Foods are less common across all outlet types, suggesting limited demand or availability.

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

a. Scatter Plot for Item Type vs Item MRP

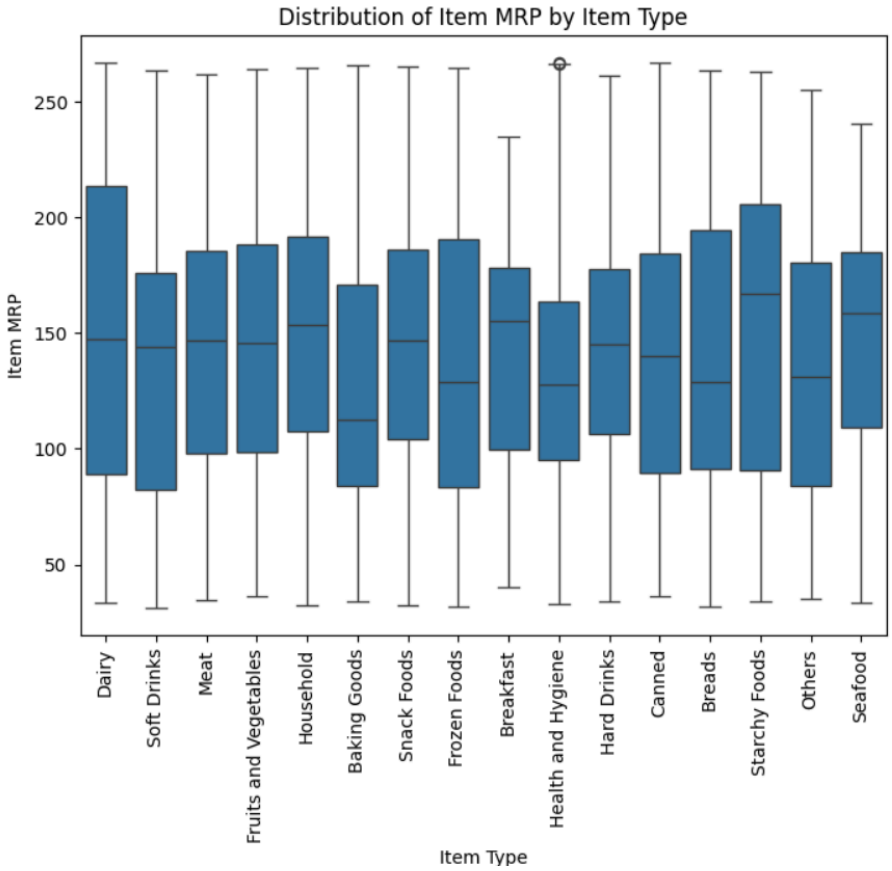
```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x="Item_Type", y="Item_MRP")
plt.title("Item Type vs Item MRP")
plt.xlabel("Item Type")
plt.xticks(rotation=90)
plt.ylabel("Item MRP")
plt.show()
```



From the visualization, it is evident that certain categories, such as Fruits and Vegetables, Snack Foods, and Meat, have a wide range of MRPs, spanning from low to high values. On the other hand, categories like Seafood and Breakfast seem to have fewer data points and a narrower MRP range, suggesting limited representation in the dataset.

b. Box Plot

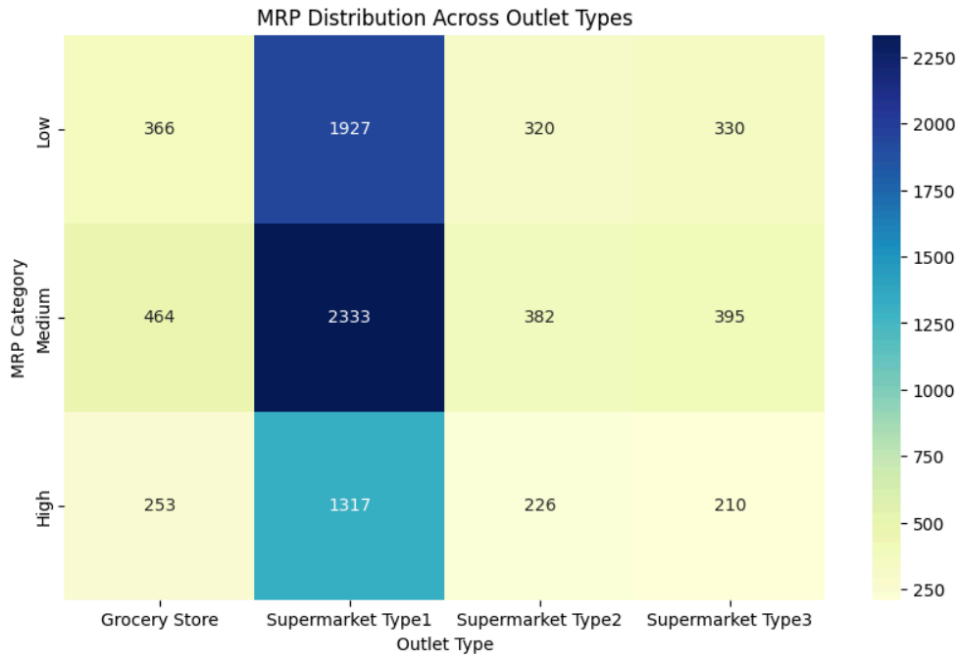
```
plt.figure(figsize=(8, 6))
sns.boxplot(data=df, x="Item_Type", y="Item_MRP")
plt.title("Distribution of Item MRP by Item Type")
plt.xlabel("Item Type")
plt.ylabel("Item MRP")
plt.xticks(rotation=90)
plt.show()
```



The box plot highlights the distribution of Item MRP across various Item Types, with noticeable variations in medians, such as higher values for Dairy and Health and Hygiene compared to Starchy Foods and Baking Goods.

c. Heatmap for Item MRP distribution

```
df["MRP_Bin"] = pd.cut(df["Item_MRP"], bins=3, labels=["Low", "Medium", "High"])
heatmap_data = pd.crosstab(df["MRP_Bin"], df["Outlet_Type"])

plt.figure(figsize=(10, 6))
sns.heatmap(heatmap_data, annot=True, cmap="YlGnBu", fmt="d")
plt.title("MRP Distribution Across Outlet Types")
plt.xlabel("Outlet Type")
plt.ylabel("MRP Category")
plt.show()
```

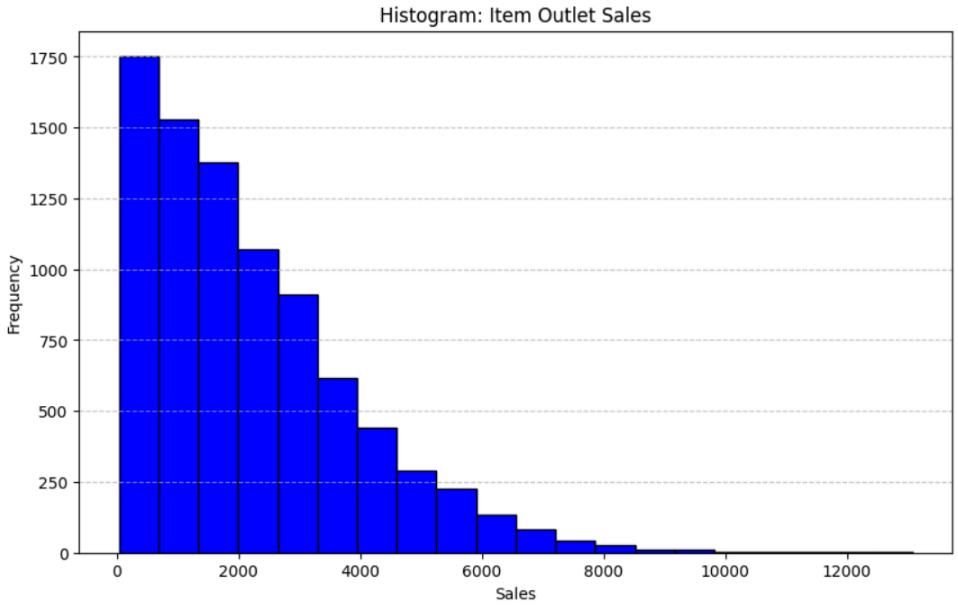


Supermarket Type 1 dominates in all MRP categories, particularly in medium MRP, followed by high MRP. Grocery Stores have a stronger presence in low and medium MRP categories but contribute minimally to high MRP. Supermarkets Types 2 and 3 show a balanced but smaller distribution across all categories, with a slight focus on medium and low MRP.

3. **Histogram and normalized Histogram.**

    a. Histogram for Item Outlet Sales

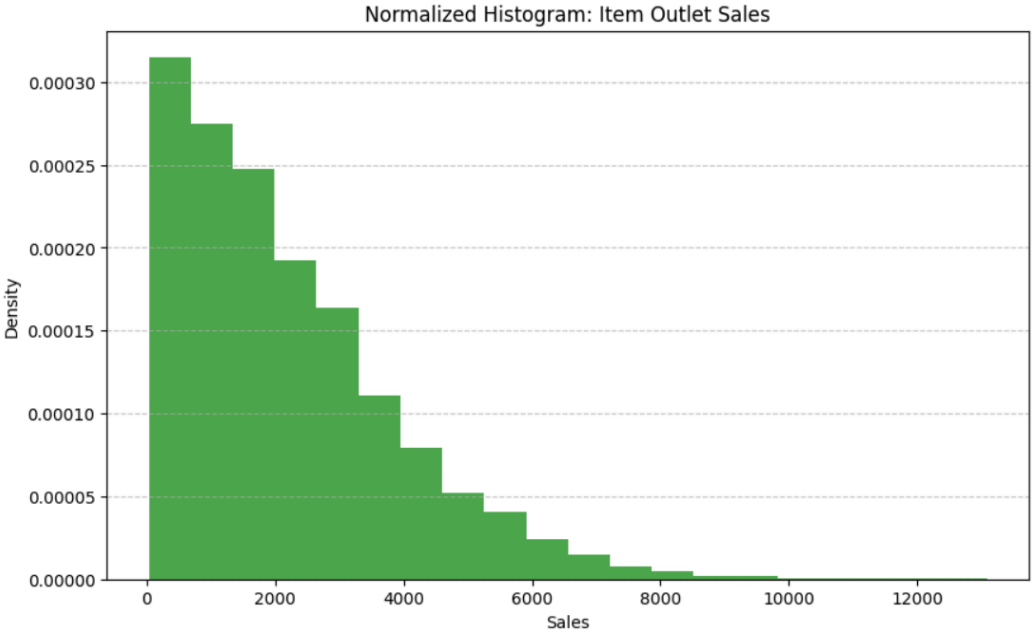
```
sales_data = df["Item_Outlet_Sales"]

plt.figure(figsize=(10, 6))
plt.hist(sales_data, bins=20, color='blue', edgecolor='black')
plt.title("Histogram: Item Outlet Sales")
plt.xlabel("Sales")
plt.ylabel("Frequency")
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



The histogram reveals the distribution of item outlet sales, which is heavily skewed to the right. The majority of sales are concentrated in the lower range, with the highest frequency occurring between 0 and 2,000. As the sales value increases, the frequency decreases significantly, indicating that higher sales amounts are less common.

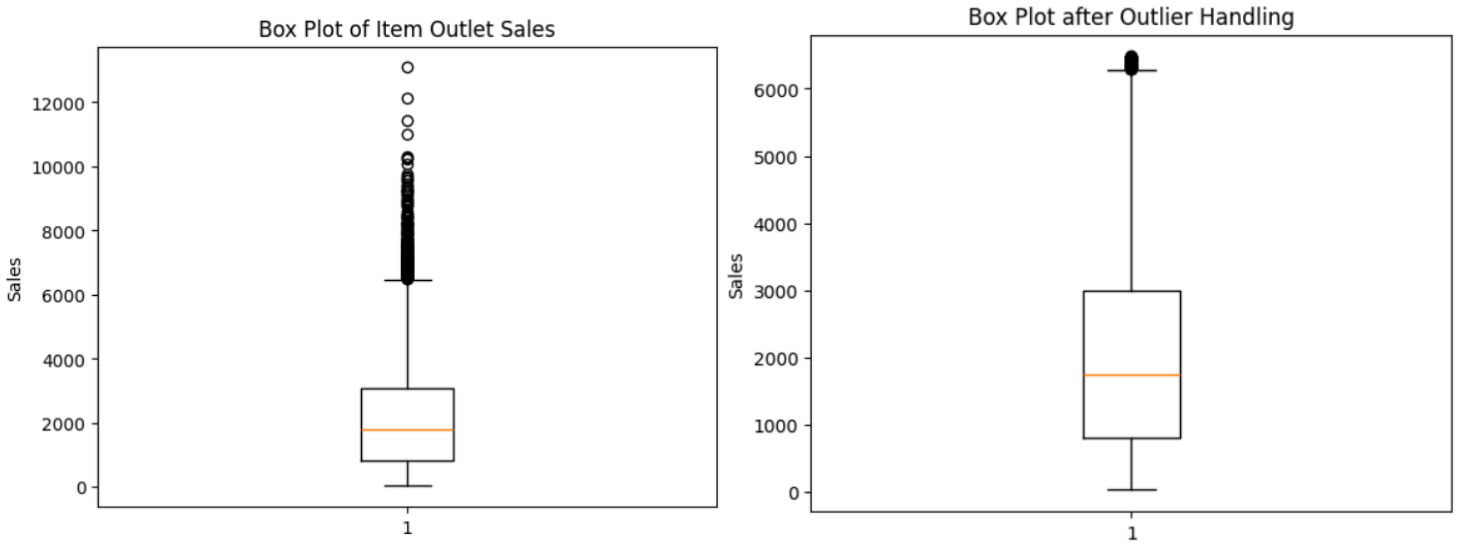
b. Normalized Histogram

```
plt.figure(figsize=(10, 6))
plt.hist(sales_data, bins=20, color='green', alpha=0.7, density=True)
plt.title("Normalized Histogram: Item Outlet Sales")
plt.xlabel("Sales")
plt.ylabel("Density")
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



4. Outliers using box plot and Inter quartile range.

```
sales = df['Item_Outlet_Sales']

plt.boxplot(sales)
plt.title('Box Plot of Item Outlet Sales')
plt.ylabel('Sales')
plt.show()

Q1 = sales.quantile(0.25)
Q3 = sales.quantile(0.75)
IQR = Q3 - Q1

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

filtered_sales = sales[(sales >= lower_bound) & (sales <= upper_bound)]

df['Item_Outlet_Sales_Capped'] = np.where(
    sales < lower_bound, lower_bound,
    np.where(sales > upper_bound, upper_bound, sales)
)

plt.boxplot(filtered_sales)
plt.title('Box Plot after Outlier Handling')
plt.ylabel('Sales')
plt.show()
```



The first box plot highlights that the dataset has several outliers, with Item Outlet Sales extending well beyond the upper whisker (above ~6000). These are visible as individual points above the main plot area.After applying the interquartile range (IQR) method for outlier handling, the second box plot demonstrates that extreme outliers have been removed. The data now falls within a more compact range, capped approximately at the upper whisker (~6000)

Conclusion:

This experiment demonstrated the effective use of Matplotlib and Seaborn for data visualization and exploratory data analysis (EDA). By visualizing data through bar graphs, contingency tables, scatter plots, box plots, and heatmaps, we uncovered patterns, distributions, and relationships within the dataset. Normalized histograms provided deeper insights into data density, while the interquartile range (IQR) method successfully identified and removed outliers for improved data quality.

Experiment 3

Aim: Perform Data Modeling.

Theory:

Data Partitioning:
Data partitioning is an essential step in data modeling and machine learning. It involves dividing the dataset into two subsets: training and testing datasets. The training set is used to build the model, while the testing set is used to evaluate its performance. A common split is 75% training and 25% testing.

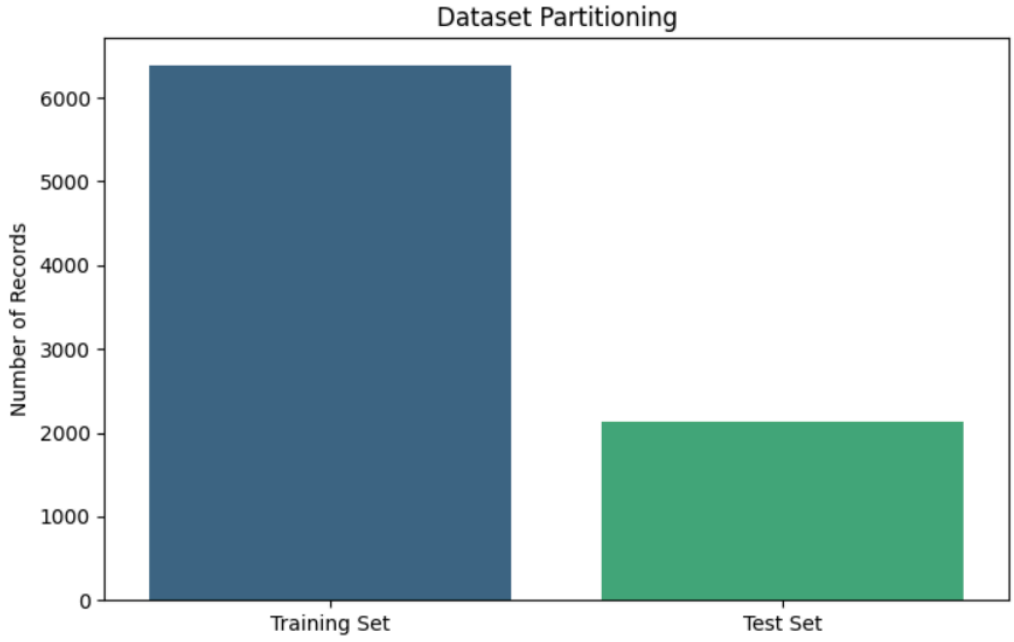
Two-Sample Z-Test:
A two-sample Z-test is a statistical test used to determine if there is a significant difference between the means of two independent samples. This test is useful when comparing training and test datasets to check if their distributions are similar. The Z-test assumes that the data follows a normal distribution and that population variances are known or the sample size is large.

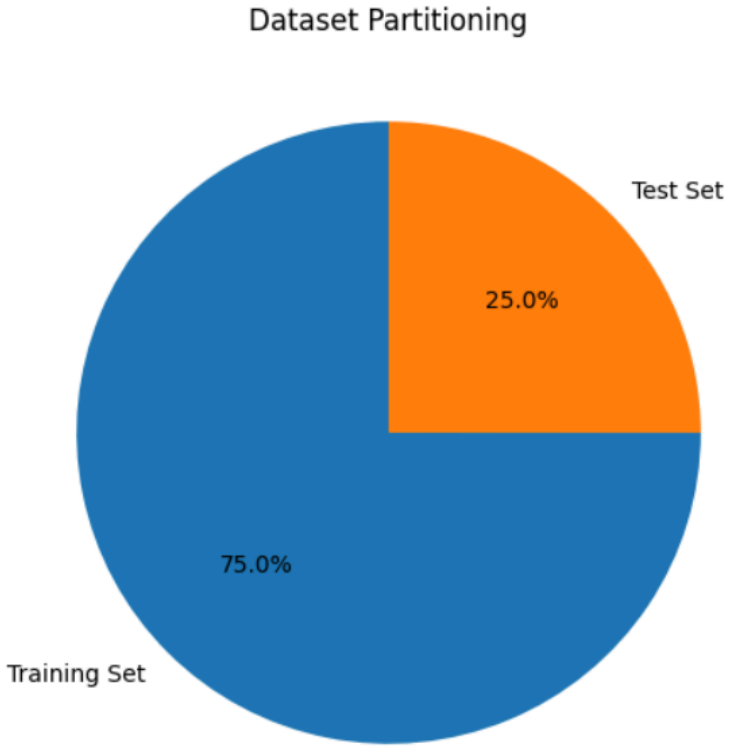
Problem Statement:

a. **Partition the data set**
   The dataset, consisting of 8,523 records, is partitioned into training (75%) and testing (25%) sets using train_test_split, ensuring a balanced distribution for model training and evaluation. The output confirms that 6,392 records are assigned to the training set and 2,131 records to the test set, maintaining the expected proportions. This step is crucial in data modeling to validate the model's performance on unseen data while preventing overfitting.

```
0s
train_df, test_df = train_test_split(df, test_size=0.25, random_state=42)

train_count = len(train_df)
test_count = len(test_df)
total_count = len(df)

print(f"Total records: {total_count}")
print(f"Training set records: {train_count} ({(train_count/total_count)*100:.2f}%)")
print(f"Test set records: {test_count} ({(test_count/total_count)*100:.2f}%)")
```

b. Use a bar graph and other relevant graphs to confirm your proportions.





Bar Graph: The bar graph displays the number of records in training as well as test set. The x axis represents the training and testing dataset while the y axis represents the number of instances in both the sets.

Pie Chart: The pie chart represents the percentage distribution of the data into training as well as testing dataset. It confirms our distribution of 75% for training and 25% for testing.

c. **Identify the total number of records in the training data set.**
We have successfully divided the dataset into training and testing data. The total number of instances in the original dataset were 8523 after preprocessing. Upon division the training data contained 6392 records (75% of original), while the testing data contained 2131 records (25% of original).

```
Total records: 8523
Training set records: 6392 (75.00%)
Test set records: 2131 (25.00%)
```

d. **Validate partition by performing a two-sample Z-test.**

A two-sample Z-test was conducted to compare the mean Item_Outlet_Sales values between the training and test datasets. The test aimed to determine whether there was a significant difference between the two groups. The calculated Z-score (-0.8980) and p-value (0.3692) suggest that the difference is not statistically significant at α = 0.05, indicating that the partitioning of this feature is balanced and does not introduce bias.

```
def sample_test(sample1, sample2):
    mean1, mean2 = np.mean(sample1), np.mean(sample2)
    std1, std2 = np.std(sample1, ddof=0), np.std(sample2, ddof=0)  # Use ddof=0 for Z-test
    n1, n2 = len(sample1), len(sample2)

    # Z-score formula for two-sample Z-test
    z_score = (mean1 - mean2) / np.sqrt((std1**2 / n1) + (std2**2 / n2))

    # Two-tailed p-value calculation
    p_value = 2 * (1 - norm.cdf(abs(z_score)))

    return z_score, p_value

# Call function and store output
z, p = sample_test(train_df["Item_Outlet_Sales"], test_df["Item_Outlet_Sales"])

# Print results
print(f"Z-score: {z:.4f}")
print(f"P-value: {p:.4f}")

# Interpretation
alpha = 0.05  # Significance level
if p < alpha:
    print("There is a significant difference in between training and test datasets.")
else:
    print("No significant difference in between training and test datasets.")
```

```
Z-score: -0.8980
P-value: 0.3692
No significant difference in between training and test datasets.
```

Conclusion:

The experiment successfully demonstrated the process of data partitioning and validation using statistical methods. The dataset was effectively split into 75% training (6,392 records) and 25% testing (2,131 records), ensuring a balanced distribution for

model training and evaluation. Visualization techniques such as bar graphs and pie charts confirmed the correct proportions of the partitioned data.

Furthermore, a two-sample Z-test was performed on Item_Outlet_Sales to validate the partitioning. The Z-score (-0.8980) and p-value (0.3692) indicated that there was no significant difference between the training and testing sets, confirming that the split was unbiased and statistically sound. This ensures that the model can be trained effectively without data distribution discrepancies affecting its performance.

Experiment 4

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

Theory:

1. Pearson's Correlation Coefficient

- Measures linear correlation between two continuous variables.
- Values range from -1 to 1:
  - +1: Perfect positive correlation
  - -1: Perfect negative correlation
  - 0: No correlation

Formula:

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

2. Spearman's Rank Correlation

- Measures monotonic relationships (increasing or decreasing trends).
- Useful for non-linear relationships.
- Based on ranked data rather than raw values.
- No assumption about normality.

Formula:

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

where d_i is the difference between ranks of X and Y.

3. Kendall's Rank Correlation

- Measures the ordinal association between two variables.
- Useful for small datasets or ordinal data.
- Measures the consistency of the rank ordering.

Formula:

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

where C is the number of concordant pairs, and D is the number of discordant pairs.

4. Chi-Squared Test (χ² Test)

- Measures association between two categorical variables.
- Compares observed and expected frequencies in a contingency table.
- Null hypothesis states that there is no association between the variables.
- If the p-value is small (<0.05), we reject the null hypothesis (there is a significant association).

Formula:

$$\chi^2 = \sum \frac{(O - E)^2}{E}$$

where O is the observed frequency, and E is the expected frequency.

Steps:

1. **Pearson's Correlation Coefficient**

```
[23] from scipy.stats import pearsonr

     x = df["Item_MRP"].values
     y = df["Item_Outlet_Sales"].values
     n = len(x)

     mean_x = np.mean(x)
     mean_y = np.mean(y)

     numerator = sum((x - mean_x) * (y - mean_y))
     denominator = np.sqrt(sum((x - mean_x) ** 2) * sum((y - mean_y) ** 2))

     pearson_corr = numerator / denominator

     # Calculate p-value using scipy
     pearson_corr_scipy, p_value = pearsonr(x, y)

     print(f"Pearson Correlation (Manual): {pearson_corr:.4f}")
     print(f"Pearson Correlation (Scipy): {pearson_corr_scipy:.4f}")
     print(f"P-value: {p_value}")

     Pearson Correlation (Manual): 0.5676
     Pearson Correlation (Scipy): 0.5676
     P-value: 0.0
```

- The Pearson correlation coefficient between Item_MRP (Maximum Retail Price) and Item_Outlet_Sales is 0.5676.
- This indicates a moderate positive linear relationship between MRP and sales.
- As the MRP increases, sales tend to increase as well, but the relationship is not perfectly strong.
- A higher MRP is somewhat associated with higher sales, but other factors might also be influencing sales.
- The p-value is 0.0, meaning the correlation is highly significant and unlikely due to random chance.

2. Spearman's Rank Correlation

```
[25] from scipy.stats import spearmanr

     x = df["Item_MRP"].values
     y = df["Item_Outlet_Sales"].values
     n = len(x)

     x_ranks = np.argsort(np.argsort(x))
     y_ranks = np.argsort(np.argsort(y))

     d_squared_sum = sum((x_ranks - y_ranks) ** 2)
     spearman_corr_manual = 1 - (6 * d_squared_sum) / (n * (n**2 - 1))

     #Compute Spearman's Correlation & p-value using SciPy
     spearman_corr_scipy, p_value = spearmanr(x, y)

     print(f"Spearman Correlation (Manual): {spearman_corr_manual:.4f}")
     print(f"Spearman Correlation (SciPy): {spearman_corr_scipy:.4f}")
     print(f"P-value: {p_value}")

Spearman Correlation (Manual): 0.5630
Spearman Correlation (SciPy): 0.5630
P-value: 0.0
```

- The Spearman correlation (0.5630) is very close to the Pearson correlation (0.5676).
- This suggests that the relationship is nearly linear, meaning that higher MRP values are generally associated with higher sales, and the ranking order remains consistent.
- If the Spearman correlation had been significantly different from Pearson's, it would indicate a non-linear relationship.
- The p-value is 0.0, indicating a statistically significant relationship, meaning the correlation is unlikely due to chance.

3. Kendall's Rank Correlation

```python
from scipy.stats import kendalltau

# Define ordinal mapping for Outlet_Size
size_mapping = {"Small": 1, "Medium": 2, "High": 3}
df["Outlet_Size_Ordinal"] = df["Outlet_Size"].map(size_mapping)

# Extract necessary columns
x = df["Outlet_Size_Ordinal"].values  # Now numerical
y = df["Item_Outlet_Sales_Capped"].values
n = len(x)

C = 0  # Concordant pairs
D = 0  # Discordant pairs

# Manual Kendall's Tau calculation
for i in range(n):
    for j in range(i + 1, n):
        if (x[i] - x[j]) * (y[i] - y[j]) > 0:
            C += 1
        elif (x[i] - x[j]) * (y[i] - y[j]) < 0:
            D += 1

kendall_tau_manual = (C - D) / (0.5 * n * (n - 1))

# Compute Kendall's Tau using SciPy
kendall_tau_scipy, p_value = kendalltau(x, y)

print(f"Kendall's Rank Correlation (Manual): {kendall_tau_manual:.4f}")
print(f"Kendall's Rank Correlation (SciPy): {kendall_tau_scipy:.4f}")
print(f"P-value: {p_value:.4f}")
```

```
Kendall's Rank Correlation (Manual): 0.1226
Kendall's Rank Correlation (SciPy): 0.1633
P-value: 9.675939169456887e-82
```

- Kendall's Tau (0.1633) indicates a weak positive ordinal association between Outlet_Size and Item_Outlet_Sales_Capped.
- Compared to Pearson and Spearman correlations, Kendall's Tau is lower, suggesting a less consistent ranking relationship.
- Despite the weak correlation, the result is statistically significant, meaning the relationship is unlikely due to random chance.
- While larger outlet sizes may have slightly higher sales, the effect is weak, implying other factors like location, type, or promotions are more influential.

4. Chi-Square Test:

```python
import numpy as np
import pandas as pd
from scipy.stats import chi2, chi2_contingency

# Create contingency table
contingency_table = pd.crosstab(df["Outlet_Size"], df["Sales_Bin"])
observed = contingency_table.values

row_totals = observed.sum(axis=1).reshape(-1, 1)
col_totals = observed.sum(axis=0)
grand_total = observed.sum()
expected = (row_totals @ col_totals.reshape(1, -1)) / grand_total

chi_squared_manual = np.sum((observed - expected) ** 2 / expected)

# Degrees of Freedom
df_degrees = (observed.shape[0] - 1) * (observed.shape[1] - 1)

# Compute p-value manually
p_value_manual = 1 - chi2.cdf(chi_squared_manual, df_degrees)

# Compute using SciPy
chi2_scipy, p_value_scipy, df_scipy, expected_scipy = chi2_contingency(contingency_table)

# Print results
print(f"Chi-Square Statistic (Manual): {chi_squared_manual:.4f}")
print(f"Chi-Square Statistic (SciPy): {chi2_scipy:.4f}")
print(f"P-value (Manual): {p_value_manual:.4f}")
print(f"P-value (SciPy): {p_value_scipy:.4f}")
print(f"Degrees of Freedom: {df_degrees}")
print("\nExpected Frequencies:")
print(expected)
```

```python
# Hypothesis Decision
alpha = 0.05  # Significance level
if p_value_scipy < alpha:
    print("\nConclusion: Null hypothesis (H₀) is REJECTED.")
    print("There is a significant relationship between Outlet Size and Sales.")
else:
    print("\nConclusion: Null hypothesis (H₀) is NOT rejected.")
    print("There is no significant relationship between Outlet Size and Sales.")
```

```
Chi-Square Statistic (Manual): 356.9009
Chi-Square Statistic (SciPy): 356.9009
P-value (Manual): 0.0000
P-value (SciPy): 0.0000
Degrees of Freedom: 4

Expected Frequencies:
[[ 311.10407134  310.33861316  310.5573155 ]
 [ 932.31080605  930.01689546  930.67229849]
 [1601.58512261 1597.64449138 1598.77038601]]

Conclusion: Null hypothesis (H₀) is REJECTED.
There is a significant relationship between Outlet Size and Sales.
```

- The p-value (0.0000) is less than 0.05, meaning we reject the null hypothesis and conclude that Outlet Size and Sales are related.
- The chi-square statistic (356.9009) indicates a strong deviation from expected values, suggesting a considerable association between Outlet Size and Sales.
- Given the degrees of freedom, the result is statistically valid for the given contingency table structure.
- While the test confirms a relationship, it does not indicate how strong or in which direction the relationship is.

Conclusion:

The experiment analyzed relationships between different variables using statistical hypothesis tests. Pearson's Correlation (0.5676) and Spearman's Rank Correlation (0.5630) showed a moderate positive relationship between Item MRP and Sales, indicating that higher MRP tends to be associated with higher sales. Kendall's Rank Correlation (0.1633) suggested a weak but statistically significant positive association between Outlet Size and Sales, implying that while larger outlets may have slightly higher sales, other factors likely play a more significant role. The Chi-Square Test (p-value = 0.0000) confirmed a strong dependency between Outlet Size and Sales, but it does not indicate the strength or direction of the relationship. Overall, while MRP strongly influences sales, Outlet Size also has an impact, though it may not be the primary determining factor.

Experiment 5

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:

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 + -0.1633*On-board service + 15.1641
```

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.

Logistic regression

```python
# 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)

# 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

# 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)

# Train Logistic Regression Model with hyperparameter tuning
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)
```

This code implements Logistic Regression for classification by selecting numerical features and refining them using Recursive Feature Elimination (RFE) to pick the 10 most important ones. It then splits the data into training and testing sets, ensuring balanced class distribution using stratification. To handle outliers, RobustScaler is applied to normalize the feature values. 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.

```
['Inflight wifi service', 'Departure/Arrival time convenient', 'Ease of Online booking', 'Gate location', 'Online boarding', 'Seat comfort', 'Inflight entertainment', 'On-board service
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:

In this experiment, we applied Linear Regression and Logistic Regression using Scipy and Scikit-Learn. The Linear Regression model achieved 91.75% accuracy, showing a strong relationship between Departure Delay, On-board Service, Inflight WiFi Service, and Seat Comfort with the target variable. The MSE of 115.99 indicates some variance but an overall good fit.

For classification, we used Logistic Regression with Recursive Feature Elimination (RFE), achieving 80.60% accuracy and an F1 score of 0.80. The confusion matrix showed a well-balanced classification. This experiment highlights the importance of feature selection, data preprocessing, and evaluation metrics in building accurate predictive models.

Experiment 6

Aim: Classification modelling

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. **Naïve 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

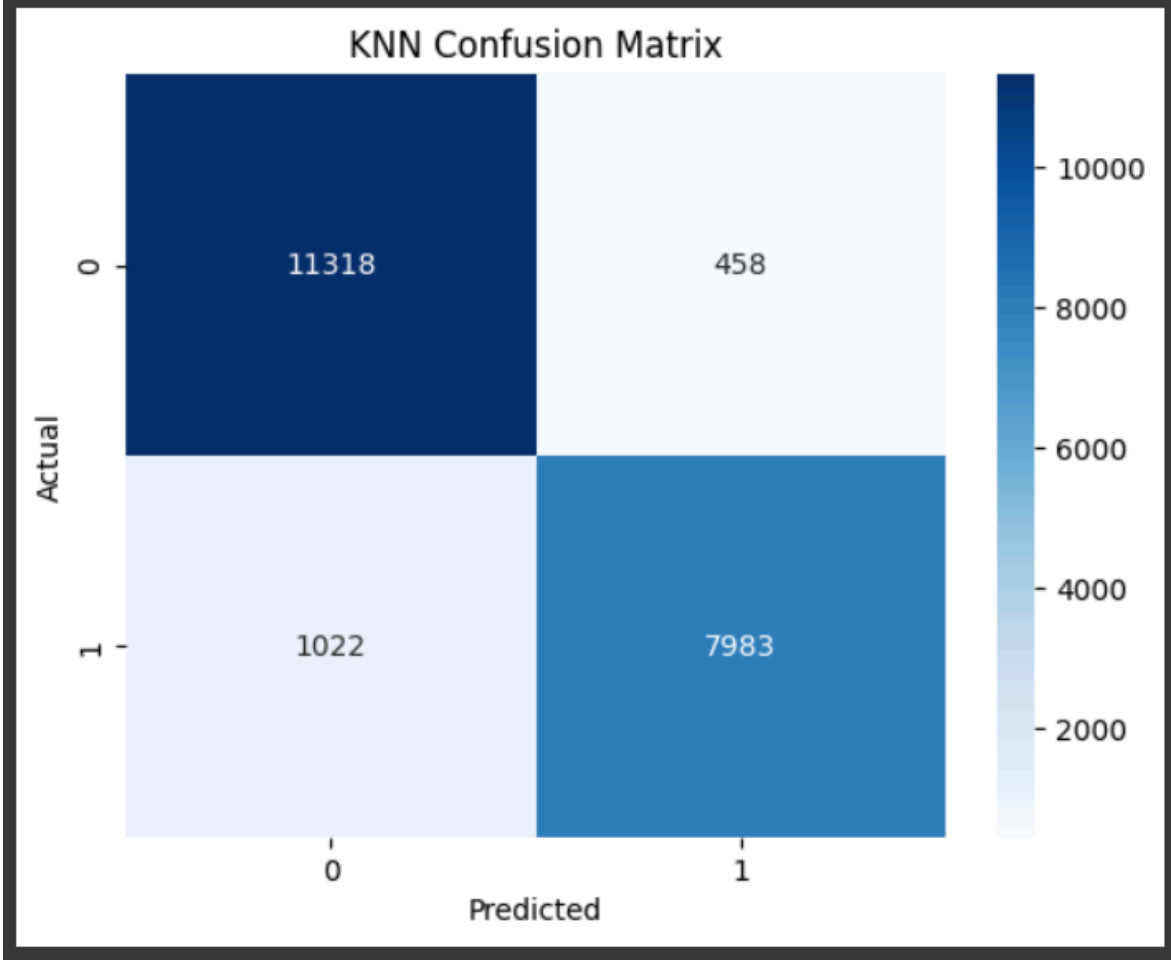
3. **KNN model**

```
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.

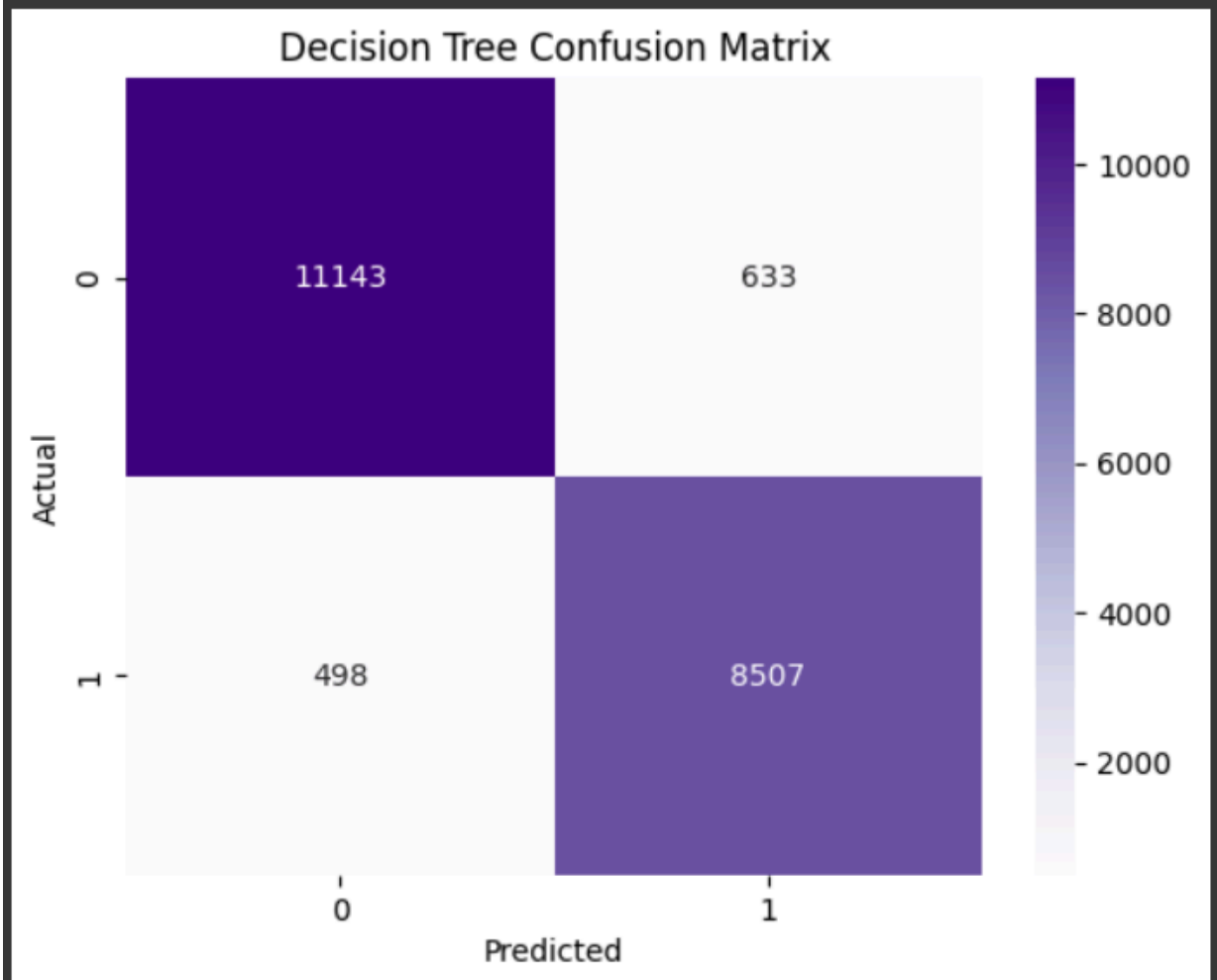
True Positives: 7983, True Negatives: 11318.
False Positives: 458, False Negatives: 1022.

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
```



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. 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.

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. 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 instance and features, SVM takes a long time to give favorable outcomes.

Conclusion:

In this experiment, we explored various classification models, including K-Nearest Neighbors (KNN), Decision Tree, and Support Vector Machine (SVM), to analyze airline passenger satisfaction. The KNN model achieved an accuracy of 92.88%, while the Decision Tree performed slightly better with 94.56% accuracy, demonstrating its ability to capture complex decision boundaries effectively. The confusion matrices and classification reports highlighted that both models performed well, with minor misclassifications.

However, the SVM model was impractical for this dataset due to its high computational cost, as it requires solving a quadratic optimization problem, making it inefficient for large datasets. The results indicate that Decision Trees, with their interpretability and efficiency, are better suited for this classification task.

Experiment 7

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.**

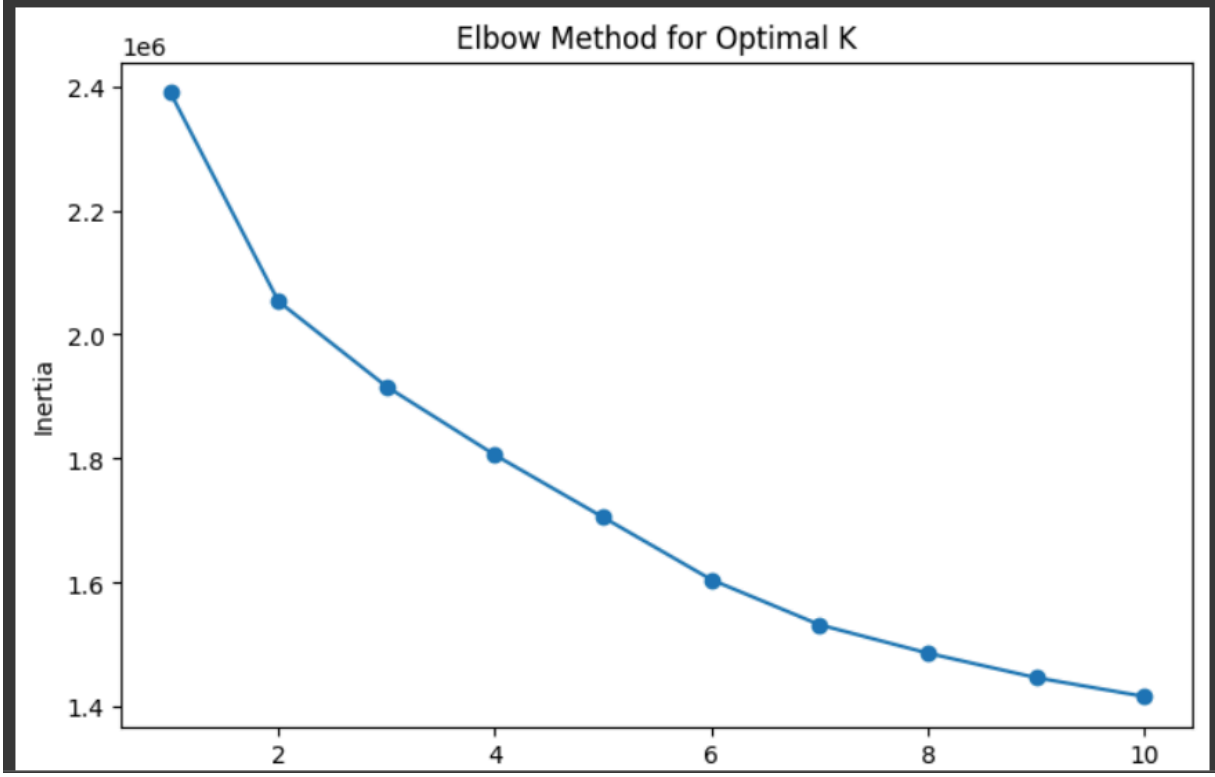
	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	Ease of Online booking	Gate location	...	Inflight entertainment	On-board service	Leg room service	Baggage handling
0	1	0	13	1	2	460	3	4	3	1	...	5	4	3	4
1	1	1	25	0	0	235	3	2	3	3	...	1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

```
# 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()
```



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. In this plot, the elbow appears around K = 4 or 5.

3. **K means clustering**

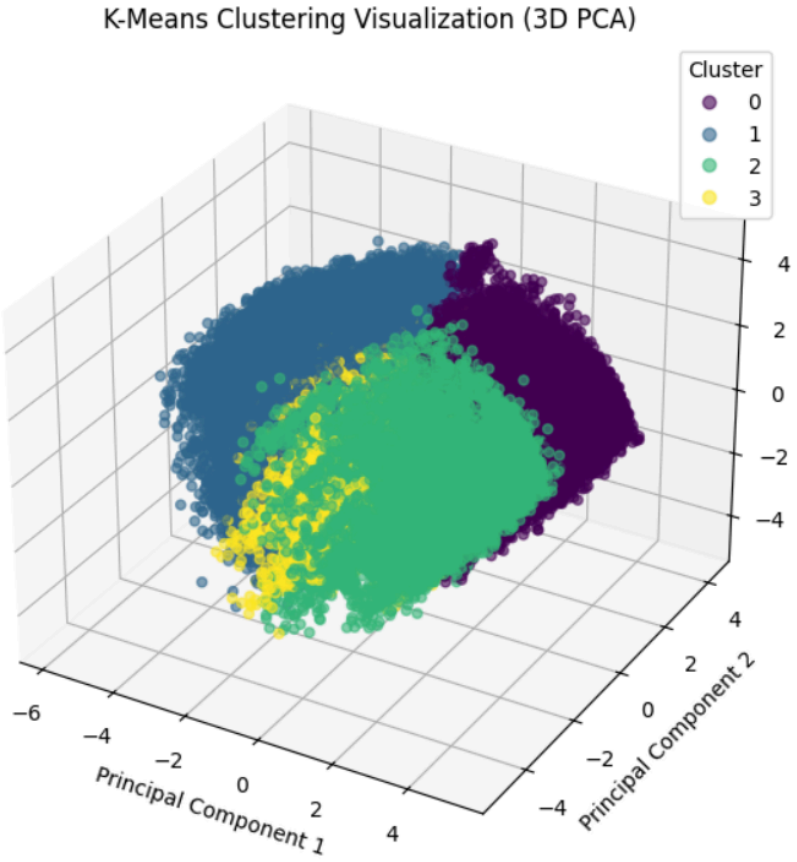
```
# Apply K-Means with optimal K (assume 3 based on elbow method)
kmeans = KMeans(n_clusters=4, random_state=42, n_init=10)
df['Cluster'] = kmeans.fit_predict(df_scaled)

# 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:

```python
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

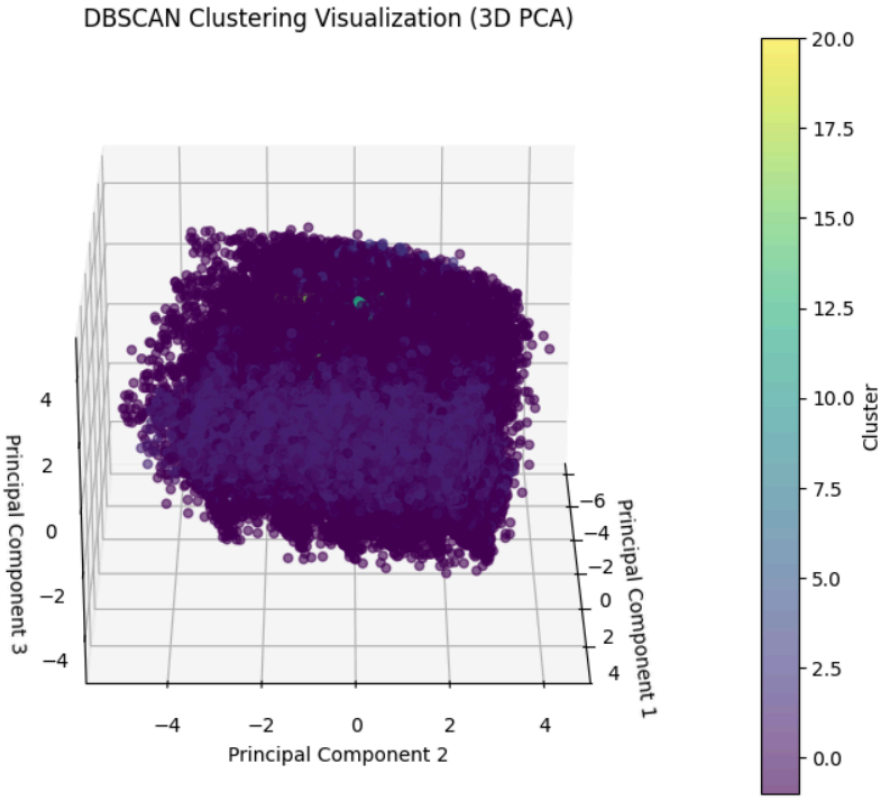
```
from sklearn.cluster import DBSCAN

# 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.

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 this experiment, we implemented and analyzed different clustering techniques, including K-Means and DBSCAN, to identify meaningful patterns in an airline passenger satisfaction dataset.

K-Means clustering successfully grouped the dataset into four well-defined clusters (K=4), ensuring a balanced distribution of data points across clusters. The Elbow Method helped determine the optimal number of clusters, and PCA visualization confirmed that the clusters were well-separated.

DBSCAN clustering, on the other hand, struggled with this dataset. The chosen parameters (eps=2, min_samples=50) resulted in a large number of data points being classified as noise (-1), indicating that DBSCAN was unable to effectively form clusters. This suggests that the dataset does not contain well-defined dense regions, making DBSCAN less suitable for this case.

Experiment 8

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.

Recommendation System Evaluation Measures

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	TO... CRI...
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

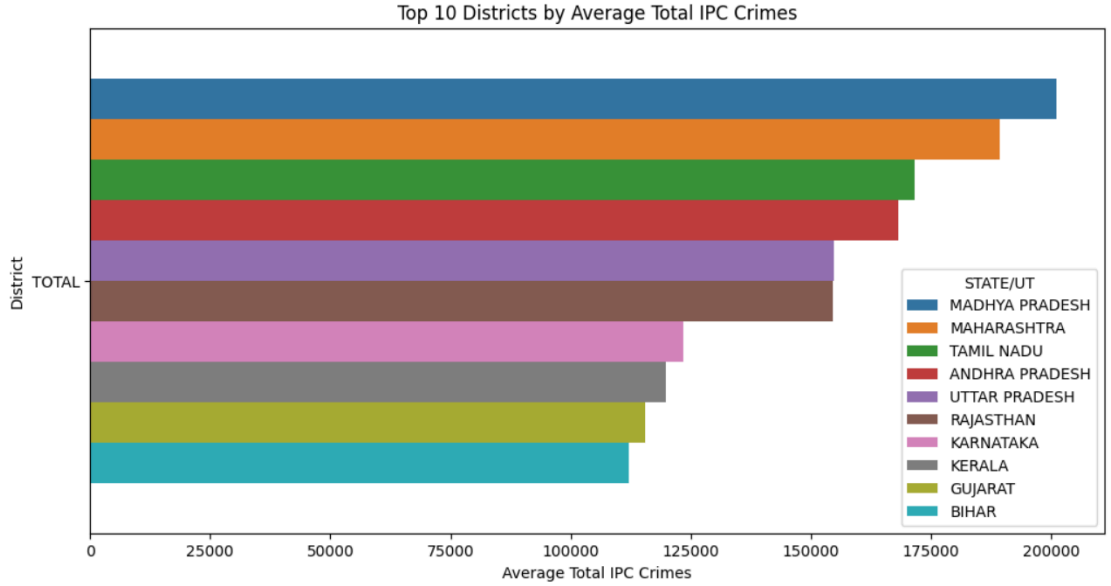
9017 rows × 34 columns

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

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

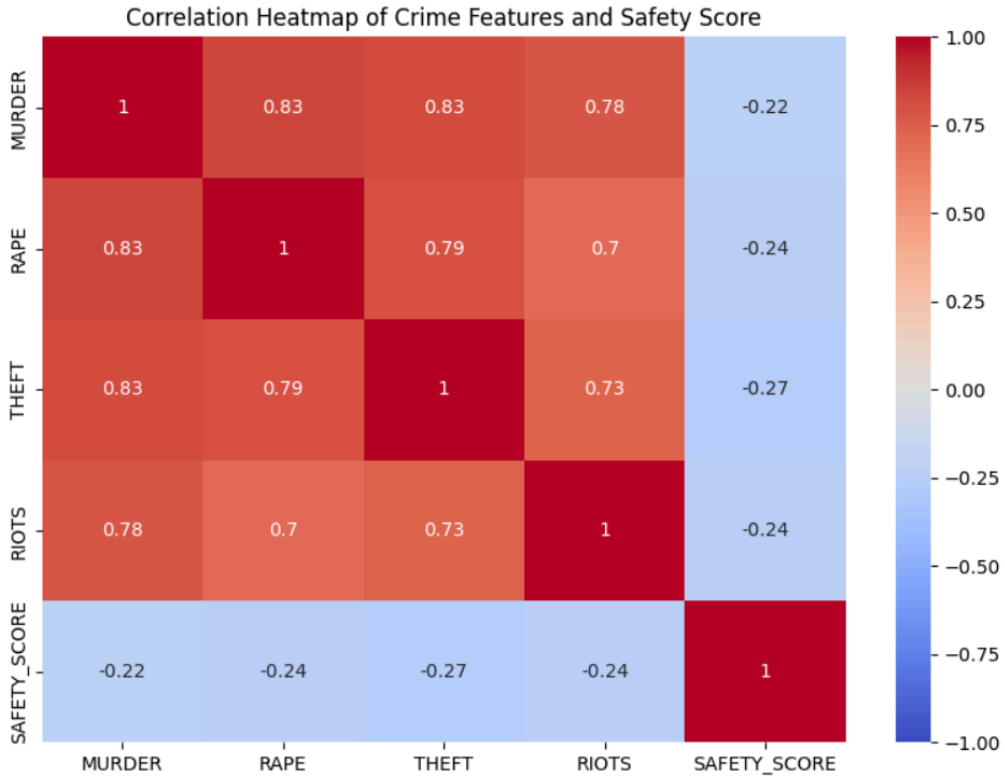
STATE/UT	0
DISTRICT	0
YEAR	0
MURDER	0
ATTEMPT TO MURDER	0
CULPABLE HOMICIDE NOT AMOUNTING TO MURDER	0
RAPE	0
CUSTODIAL RAPE	0
OTHER RAPE	0
KIDNAPPING & ABDUCTION	0
KIDNAPPING AND ABDUCTION OF WOMEN AND GIRLS	0
KIDNAPPING AND ABDUCTION OF OTHERS	0
DACOITY	0
PREPARATION AND ASSEMBLY FOR DACOITY	0
ROBBERY	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.**

```python
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

```python
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.

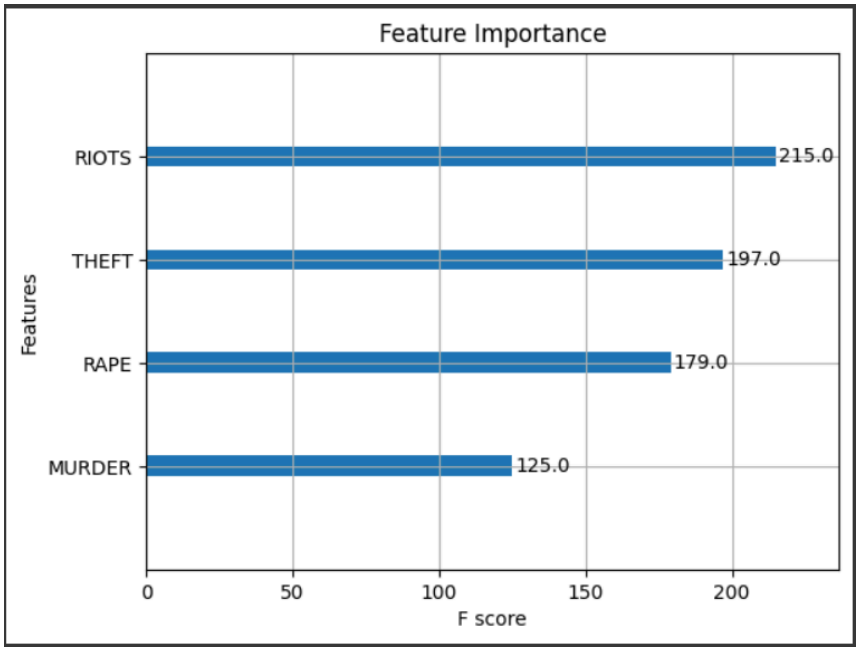
```
def get_safety_recommendations(model, df_agg, features, top_n=5, state=None, crime_weights=None):
    X_all = df_agg[features]
    dmatrix = xgb.DMatrix(X_all)

    # 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

Experiment 9

Aim: To perform 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 interface for programming entire clusters with implicit data parallelism and fault tolerance.

Key Features of Apache Spark:

- In-Memory Processing: Spark stores intermediate data in RAM, making it significantly faster than Hadoop MapReduce.
- Lazy Evaluation: Spark optimizes execution by delaying computation until necessary.
- Fault Tolerance: Uses Resilient Distributed Datasets (RDDs) to recover lost data.
- Multiple Language Support: Works with Python (PySpark), Scala, Java, and R.
- Rich Libraries: Includes Spark SQL (structured data), MLlib (machine learning), GraphX (graph processing), and Spark Streaming (real-time data).

How Apache Spark Works?

- Spark applications run as independent processes coordinated by a SparkContext in the driver program.
- The Cluster Manager (e.g., YARN, Mesos, or Spark Standalone) allocates resources.
- Executors run on worker nodes to perform computations and store data.
- Data is partitioned across nodes for parallel processing.

2. **How is Data Exploration Done in Apache Spark?**

Exploratory Data Analysis (EDA) in Spark involves examining datasets to summarize their main characteristics, often using visual methods and statistical summaries.

Steps for EDA in Apache Spark:

1. Loading the Dataset
    a. Read data from CSV, JSON, Parquet, or other formats using spark.read.

```
df = spark.read.csv("data.csv", header=True, inferSchema=True)
```

2. Viewing Data Structure
    a. Check schema (column names and data types) using printSchema().
    b. Display sample records with show().

```
df.printSchema()
df.show(5)
```

3. Basic Statistics
    a. Compute summary statistics (count, mean, stddev, min, max) using describe().

```
df.describe().show()
```

4. Handling Missing Values
    a. Identify null values:

```
from pyspark.sql.functions import col, isnan, when, count
df.select([count(when(col(c).isNull(), c)).alias(c) for c in df.columns]).show()
```

    b. Drop or fill missing values:

```
df_clean = df.na.drop()  # Drop rows with nulls
df_filled = df.na.fill(0)  # Fill nulls with 0
```

5. Data Aggregation and Grouping
    a. Group data and compute aggregations:

```
df.groupBy("category").agg({"price": "avg", "quantity": "sum"}).show()
```

6. Data Visualization (Using Pandas Integration)
    a. Convert Spark DataFrame to Pandas DataFrame for visualization:

```
import matplotlib.pyplot as plt
pandas_df = df.toPandas()
pandas_df["price"].hist()
plt.show()
```

7. Correlation Analysis
    a. Compute correlations between numerical columns:

```
from pyspark.ml.stat import Correlation
from pyspark.ml.feature import VectorAssembler
assembler = VectorAssembler(inputCols=numeric_cols, outputCol="features")
df_vector = assembler.transform(df).select("features")
matrix = Correlation.corr(df_vector, "features").collect()[0][0]
print(matrix.toArray())
```

8. Handling Categorical Data
    a. Use StringIndexer or OneHotEncoder for categorical variables.

```
from pyspark.ml.feature import StringIndexer
indexer = StringIndexer(inputCol="category", outputCol="categoryIndex")
```

indexed_df = indexer.fit(df).transform(df)

9. Saving Processed Data
    a. Write the cleaned/processed data back to storage:

df.write.parquet("processed_data.parquet")

Conclusion:

In this experiment, we learned about Exploratory Data Analysis (EDA) using Apache Spark and Pandas.

- Spark provides a distributed framework for handling large-scale datasets efficiently.
- Key steps included data loading, schema inspection, missing value handling, statistical summaries, and visualization.
- Pandas integration helped in plotting and further analysis.
- Spark's parallel processing makes it suitable for big data EDA, while Pandas is useful for smaller datasets.

Experiment 10

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 this experiment, we explored Batch and Stream Data Processing using Apache Spark.

Batch Processing

- Used for static, large-scale datasets (e.g., historical data).
- Processes data in fixed intervals (e.g., hourly/daily jobs).
- Ideal for ETL, analytics, and reporting.
- Example: spark.read.csv() → Transformations → df.write.save().

Stream Processing

- Handles real-time, continuous data (e.g., live logs, IoT sensors).
- Processes data in micro-batches or event-by-event.
- Used for fraud detection, live monitoring, alerts.
- Example: spark.readStream → Transformations → writeStream.start().

Name: Rakshit K Sharma
DISC
49

05/05

Assignment - 1

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

→ Artifical Intelligence is a branch of computer science that enables machines to mimic human intelligence, including problem solving, decision making and learning from experience. AI systems utilize algorithms, data, and computational power to perform tasks that typically require human intelligence. During COVID-19 pandemic, AI played a crucial role in helping people, buissenesses and governments survive and adapt to new challenges. It helped in

1. Healthcare & Medical Innovations
2. AI in bussiness and remote works
3. AI in education and E learning
4. AI in supply chain and legistics
5. AI in Mental health and Social well being

Q2. What are AI agents terminology, explain with examples.

→ 1. Environment
Everything external to agent with which it interacts.
Eg. Chatbot - users
2. Sensors
Used by agent to perceive the environment
Eg. Camera

Sundaram
FOR EDUCATIONAL USE

3. Actuators
Allow agent to take action or interact with environment
4. Percept
Input received by agent's sensors
Eg. Camera images
5. Agent function
Maps percept received to actions to be performed
6. Performance
Analyzes how well the agent has performed given the circumstances
Eg. Accuracy

Q3. How AI technique is used to solve 8 puzzle problem?

→ The 8 puzzle problem consists of a 3x3 grid with 8 numbered tiles and one empty space. The goal is to move the tiles to reach a predefined arrangement using valid moves (up, down, left, right)

Initial state	Final State
1 8 3	1 2 3
4 6 2	4 5 6
7 5	7 8

AI can use both uninformed or informed search to find a solution.

1. Uninformed Search
These search space without prior or additional knowledge of the best path.
→ Breadth First Search
- Explores all possible moves level by level

Sundaram
FOR EDUCATIONAL USE

- Guarantees the shortest solution but is memory intensive

→ Deapth First Search

- Explores one path deeply before backtracking
- May not find the optimal solution ad can get stuck in loop.

At every state, AI will create new nodes for every state. If tiles are moved either up, down, left, right depending on the different situations. Every step will create a new state ad these states can be applied to an algorithm to find the goal.

Q4. what is PEAS description ? Give PEAS description for following :

→ PEAS is a framework to describe the structure of an intelligent agent in an AI system. PEAS stand for

→ Performance Measure : Defines how the success of agent is measured

→ Environment : The surrounding in which agent operates.

→ Actuators : The components that allow agent to take action

→ Sensors : The components to perceive environment.

PEAS description for

1. Taxi Driver AI

P : Safety, fuel efficiency, travel time, legal compliance

E : Roads, traffic, pedestrians, other vehicles

A : Steering, accelerator, brakes, horn

S : GPS, cameras, speedometer, fuel gauge

Sundaram
FOR EDUCATIONAL USE

2. Medical Diagnosis System

P: Accuracy, speed, recovery rate

E: patient data, medical records, symptoms

A: Display screen, reports, recommendations

S: user input, medical imaging

3. AI Music Composer

P: Quality, user preference, harmonics, pitch

E: Music database, feedback, instruments

A: Music file, sheet generation, MIDI file

S: genre, music patterns, instrument sound libraries

4. Aircraft autolander

P: Safe landing, smooth touchdown, taxi to alignment runway

E: Runway, weather, altitude

A: landing gear, flaps, thrusters, rudder

S: Altimeter, GPS, gyroscope, airspeed

5. Essay Evaluator

P: Accuracy, grammatical corrections, coherence

E: Submitted essays, grading

A: Score output, feedback

S: Text input, NLP, images.

6. Robotic Sentry Gun for the Keck lab

P: Accuracy in target identification

E: Laboratory surroundings, intruders

A: Gun turrent, warning alarms

S: Motion sensors, infrared cameras.

Sundaram
FOR EDUCATIONAL USE

Q5. Categorize a shopping bot for an online bookstore according to each of the six dimensions

→ Observability – Partially observable
Deterministic/Stochastic – Stochastic
Episodic/Sequential – Sequential
Static/Dynamic – Dynamic
Discrete/Continuous – Discrete
Single/Multi Agent – Multi Agent

Q6. Differentiate Model based and utility based agent.

Model Based Agent	Utility Based Agent
→ Uses an internal model of the environment to make decision	→ Uses a utility function to evaluate and select best option
→ Chooses actions based on predicted future states	→ Chooses actions based on maximum utility
→ Focuses on state representation	→ Focuses on optimizing
→ Low complexity	→ High complexity
→ Eg. Self driving car on predicted road layouts	→ Self driving car finding fastest route

Q7. Explain the architecture of a knowledge based agent and learning agent.

→ A knowledge based agent is an intelligent system that stores, retrieves and applies knowledge to make decisions.

Sundaram
FOR EDUCATIONAL USE

Components of Knowledge Based agent

1. Knowledge Base

Stores facts, rules and heuristics about the environment

2. Inference Engine

Uses logical reasoning to derive new knowledge from existing facts

3. Perception module

Gathers data from sensors or user inputs

4. Query processing and decision making unit

Uses KB and inference rules to generate responses or actions

5. Action Execution

Perform the choosen action based on the inference result

A learning agent improves its performance over time by learning from past experiences and adapting its behaviour.

Components of Learning Agent

1. Learning Element

Improves the agent's Knowledge by updating models or rules based on new experience

2. Performance Element

Executes actions based on the agent's current knowledge

3. Critic

Evaluates agent performance and provides feedback.

4. Problem Generator

Suggests exploratory actions to improve learning

Q9. Convert the following to predicates

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

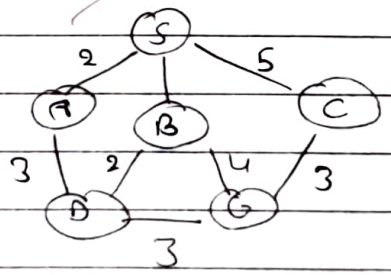
b. Bus goes via Andheri and Goregaon.

Sundaram
FOR EDUCATIONAL USE

c. Car has puncture so it is not available
Will Anita travel via Goregaon ? Use forward reasoning

→ Predicate:
travels (Anita, Vehicle)
available (car) → travels (Anita, car)
¬available (car) → travels (Anita, bus)
via (Bus, Andheri), via (Bus, Goregaon
puncture (car) → ¬available (car)
Forward reasoning :
From 4th we derive
¬available (car)
Using predicate 3 we conclude
travels (Anita, bus)
Since via (Bus, Goregaon) is given and Anita is travelling by Bus
∴ Anita will travel via Goregaon

q10. Find the route from S to G using BFS



S → A (2)
S → B (5)
S → C (5)
A → D (3)
B → D (2)
B → G (4)
C → G (3)
D → G (3)

Sundaram
FOR EDUCATIONAL USE

Using best first search

1. Expand node S

Priority Queue : (A,2) (B,5) (C,5)

Visited : S

2. Take minimum cost, expand A

Priority Queue : (B,5) (C,5) (D,3)

Visited : S, A

3. Expand D

Priority Queue : (B,5) (C,5) (G,3)

Visited : S, A, D

4. ~~Expand~~ G is the goal node

Visited : S, A, D, G

∴ path from S to G

S → A → D → G

Path cost = 2 + 3 + 3 = 8

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

→ Depth limited search is a variation of Depth first search that limits the depth of recursion to a predefined value h. It prevents the algorithm from going too deep in a infinite or large search tree.

Key features of DLS

- Solves the infinite depth problem of DFS
- Uses a depth limit h to control recursion
- Does not guarantee a shortest path if the solution exists at a greater depth than h.

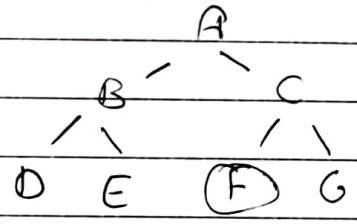
Sundaram
FOR EDUCATIONAL USE

Iterative deepening search is a combination of Depth first search and Breadth first search. It runs multiple Depth limited searches with increasing depth limits until the goal is found.

Features:

- Combines DFS space efficiency with BFS completeness
- Finds the shortest path like BFS but avoids large memory usage.
- Useful in problems where the depth of the solution is unknown.

Eg.



h = 0 → only explores A (goal not found)
h = 1 → explores A, B, C (goal not found)
h = 2 → Explores A, B, C, D, E, F (goal found)

Q12. Explain Hill climbing and its drawbacks in detail with example. Also state limitations of steepest ascent hill climbing.

→ Hill climbing is an informed search algorithm that continuously moves towards the direction of increasing value (or decreasing cost) until it reaches a peak (maximum). It is commonly used for optimization problems.

Principle:

1. Start with an initial solution (state)
2. Evaluate neighbouring states based on an evaluation function
3. Move to the neighbour with the highest value (greedy approach)
4. Repeat until no better neighbour exists.

Sundaram
FOR EDUCATIONAL USE

Eg. Consider a mountain climbing analogy where the goal is to reach the highest peak using only local observations. Suppose we want to find the maximum of the function

$f(x) = -(x-3)^2 + 9$

→ Start at $x = 0$, function value $f(0) = 0$
→ Check neighbours $x = 1$ & $x = -1$
→ Move to $x = 1$ (higher value)
→ Continue moving to $x = 2$, $x = 3$
→ At $x = 3$ we reach the peak where $f(3) = 9$

Draw backs of Hill climbing

1. Local Maxima

The algorithm can get stuck at a local maximum instead of finding the global maximum

2. Plateau

A flat region where all neighbouring states have the same value. The algorithm gets stuck because no better move exist.

3. Ridges

The search can be trapped between two higher peaks but cannot move in the right direction due to its greedy nature.

Limitations of Steepest Ascent hill climbing

Steepest hill climb is a variant where the best possible move is choosen in each step. However it suffers from aditional limitations

1. More computationally expensive.
2. Susceptible to local maxima
3. Slow progress in plateaues

Sundaram
FOR EDUCATIONAL USE

Q13. Explain simulated Annealing and write its algorithm.

→ Simulated Annealing is a probabilistic search algorithm used for optimization problems. It is an improved version of hill climbing that allows downhill moves to escape local maxima and explore better solutions. It is inspired by the annealing process in metallurgy, where materials are heated and slowly cooled to remove defects and reach a stable, low energy state.

Algorithm:

1. Start with an initial solution and temperature T
2. Generate a random neighbouring solution
3. Compare the new solution with the current one
   - If better accept it
   - If worse, accept it with a probility $P = e^{-\Delta E/T}$
4. Gradually decrease the temperature (cooling)
5. Repeat until the system is "frozen"

Q14. Explain A* algorithm with an example

→ A* is a graph / tree search algorithm used in pathfinding and graph traversal. It finds the shortest path from a start node to a goal node by considering both:

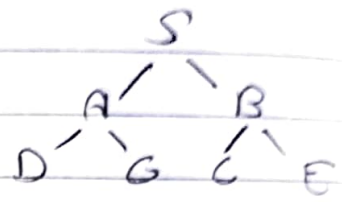
- Cost to reach the node ($g(n)$)
- Estimated cost from the node to the goal ($h(n)$)

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

A* is both complete and optimal, meaning it always finds the shortest path if the heuristic is admissible.

Sundaram
FOR EDUCATIONAL USE

Given cost:



g-value

- $S \to A = 2$, $S \to B = 5$
- $A \to D = 3$, $A \to G = 3$
- $B \to C = 4$, $B \to E = 3$

h-value

- $h(S) = 6$, $h(A) = 4$, $h(B) = 2$, $h(D) = 4$, $h(C) = 2$, $h(G) = 0$, $h(E) = 3$

f(n) for nodes:

$S \to 6$	$G \to 4$
$A \to 6$	$C \to 11$
$B \to 7$	$E \to 6$
$D \to 9$	

Final path = $S \to A \to G$

Q15. Explain Min Max algorithm and draw game tree for TIC TAC TOE game.

→ Minimax is a decision making algorithm used in two player, zero sum games like Tic Tac Toe, chess, and checkers. It assumes both players play optimally and aims to minimize the opponents maximum gain while maximizing its own minimum gain.

Features:

→ Two players
  Maximizer - tries to maximise the score
  Minimizer - tries to minimize the score

→ Every node represents possible moves. Every level represents one players turn. Leaf nodes are game outcomes.

Sundaram
FOR EDUCATIONAL USE

-1 for each move, -10 for shooting arrow.
Environment : 4x4 with wumpus, pits, gold, wall
Actuators : move forward, turn left/right, grab, shoot
Sensors : Breeze, stench, glitter, bump, scream
Percept sequence:
[breeze, stench, glitter, bump, scream]
Eg. [No, Yes, Yes, No, No]
wumpus nearby and gold in current room

q18. Solve the following Crypto arithmetic problem.
SEND + MORE = MONEY

->
```
  SEND
+ MORE
------
MONEY
```

1. Since M comes through carry and maximum carry can be 1
∴ M = 1
O + E = Y or Y + 10
N + R + carry = E or E + 10
E + O + carry = N or N + 10
S + 1 + carry = O or O + 10

2. Since O appears in the second position, and there is no carry affecting this position, O = 0
3. From the thousands place
S + 1 = O
∵ O = 0, S = 9

Sundaram
FOR EDUCATIONAL USE

4. To satisfy the middle column conditions, E = 5 fits correctly when checking the sum of N and R.

5. N = 6

Since N appears at hundreds place, we check the addition

6. From the rightmost column

D + E = Y

∴ 7 + 5 = 12 (carry 1) so Y = 2

7. The only remaining digit that satisfies all conditions without repetition is R = 8

∴

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

$$\begin{array}{r} 9567 \\ +\ 1085 \\ \hline 10652 \end{array}$$

Q19. Consider the following axioms :

All people who are graduating are happy

All happy people are smiling

Someone is graduating

Explain the following

-> 1. All people who are graduating are happy

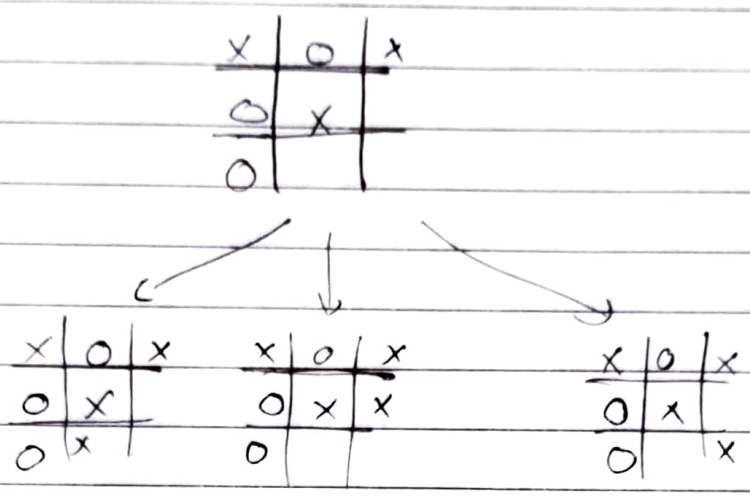
$\forall x$ (graduating(x) → happ(x))

2. All happy people are smiling

$\forall x$ (happ(x) → smiling(x))

3. Someone is graduating

$\exists x$ (graduating(x))



X wins

1. Generate all possible game states from current state
2. Recursively evaluate each state using minimax
3. Assign scores
   - If maximizer score (X) → choose maximum score
   - If minimizer score (O) → choose minimum score
4. Backtrack scores up the tree to decide the best move

Q16. Explain Alpha beta pruning algorithm and draw game tree for adversarial search.

→ Alph beta pruning is an optimization technique for the Minimax algorithm that reduces the number of nodes evaluated in a game tree. It does this by pruning (eliminating) branches that won't effect the final decision, making Minimax faster and more efficient. The minimax evaluates all possible moves in a game tree, which makes it computationally expensive. Alpha-beta pruning skips unnecessary calculations by stopping evaluation for branches that won't change the final decision.

Sundaram
FOR EDUCATIONAL USE

α → Best value found so far for maximizer
β → Best value found so far for the minimizer
Pruning occurs when
$\alpha \geq \beta$, meaning a better move is already available, so further evaluation is unnecessary.

Q17. Explain WUMPUS world environment giving its PEAS description. Explain how percept sequence is generated?

→ The wumpus world is a grid based environment used in AI to demonstrate logical reasoning and agent based problem solving. It is a partially observable, stochastic and sequential environment where an agent navigates a world filled with hazards.

Components :
→ Grid based : 4x4 grid
→ Agent : moves throughout the world and tries to find goal
→ Wumpus : A monster who will eat the agent if it enters its room
→ Pits : Deadly traps resulting in loss
→ Gold : Agent has to collect goald
→ Breeze : indicates nearby pit
→ Stench : indicates nearby wumpus
→ Glitter : indicates gold
→ Arrow : one arrow to shoot the wumpus.

PEAS description :
P : 1000 points for finding gold, -1000 for falling in pit or getting eaten by wumpus.

Sundaram
FOR EDUCATIONAL USE

Convert to clause form

1. ¬ Graduating (x) ∨ happy (x)
2. ¬happy (z) ∨ smiling (x)

~~Resolution~~ : 3. Graduating (A)

Resolution :

From 3

we substitute x < A in clause 1

¬ Graduating (A) ∨ happy (A)

since graduating A is true,

∴ happy (A)

From clause 2

¬happy (A) ∨ smiling (A)

since happ(A) is true

∴ Smiling (A)

Therefore we prove, someone is smiling

Graduating (A)

↓

¬graduating (A) ∨ happ(A)

↓

happy (A)

↓

¬happy (A) ∨ smiling (A)

↓

smiling (A)

∴ someone is smiling.

Sundaram®

FOR EDUCATIONAL USE

Q20. Explain Modus ponen with suitable example

→ Modus Ponens is a fundamental rule of inference in logic. It follows the form:

1. If P, then Q ($P \rightarrow Q$)
2. P is true (P)
3. Therefore, Q is true (Q)

Eg.

~~Fact~~ Rule: If it rains, the ground will be wet.

$P \rightarrow Q$

Fact: It is raining (P is true)

∴ Q is also true

i.e the ground is wet.

This rule is widely used in mathematical proofs, programming and real world decision making scenarios.

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

→ Forward chaining and Backward chaining are reasoning algorithm used in Artificial Intelligence and Expert systems for inference in rule based systems.

1. Forward chaining starts with known fact and applies inference rules to extract more data until a goal is reached.

Process:

start with a given fact

apply rules to derive new fact

Continue until goal is reached or no rule can be applied.

Sundaram
FOR EDUCATIONAL USE

Eg. Diagnosing

1. If a patient has fever and cough, then they might have flu.
2. If a person has a sore throat and a runny nose, then they might have a cold
3. If a person has flu, then they should rest and take fluids

Fever ∧ cough ⟶ Flu

Sore throat ∧ Runny nose ⟶ Cold

Flu → rest & fluids

Fact: patient has fever, patient has cough.

Chaining:

Patient has fever and cough

∴ R1, we deduce patient has flu

From R3, we conclude patient needs to rest and take fluids

2. Backward chaining starts with a goal and works backwards by checking which facts or rules support the goal.

Process:

Start with the goal

Check the rules that can derive goal

If rule's conditions are not known, check for facts or other rules that support those conditions.

repeat until goal is proven or no more rules apply.

Eg. Diagnosing

1. Does the patient have flu can be concluded if the patient has fever and cough.
2. It is known facts that patient has fever and cough
∴ we conclude that patient has flu.

Sundaram®

FOR EDUCATIONAL USE

Assignment 2

Q.1: Use the following data set for question 1
82, 66, 70, 59, 90, 78, 76, 95, 99, 84, 88, 76, 82, 81, 91, 64, 79, 76, 85, 90
1. Find the Mean (10pts)
2. Find the Median (10pts)
3. Find the Mode (10pts)
4. Find the Interquartile range (20pts)

Ans:
1. Find the **Mean**
Mean = Sum of all values / Number of values
Sum = 82+66+70+59+90+78+76+95+99+84+88+76+82+81+91+64+79+76+85+90 = 1611
Number of values = 20
Mean = 1611/20 = 80.55

2. Find the **Median**
First, sort the data in ascending order:
59, 64, 66, 70, 76, 76, 76, 78, 79, 81, 82, 82, 84, 85, 88, 90, 90, 91, 95, 99
The median is the average of the values at positions n/2 and n/2+1

For an even number of observations (20), the median is the average of the 10th and 11th values:
10th value = 81
11th value = 82
Median = (81 + 82)/2 = 81.5

3. Find the **Mode**
The mode is the most frequently occurring value in the dataset.
Looking at the sorted data, the number 76 appears three times, more than any other number.
Mode = 76

4. Find the **Interquartile Range (IQR)**
IQR = Q3 - Q1
First, find Q1 (25th percentile) and Q3 (75th percentile).

For Q1 (position = (20+1)*0.25 = 5.25):
Average of 5th and 6th values = (76 + 76)/2 = 76

For Q3 (position = (20+1)*0.75 = 15.75):
Average of 15th and 16th values = (88 + 90)/2 = 89
IQR = Q3 - Q1 = 89 - 76 = 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

Ans:
1. Tool Analysis
Machine Learning for Kids

- Target Audience:
    a. Primarily designed for school children (ages 7-18) and educators in K-12 settings.
    b. Also useful for beginners who want a simple introduction to AI/ML concepts.
- Use Cases:
    a. Teaches image recognition (e.g., classifying animals, objects).
    b. Introduces text-based AI (e.g., simple chatbots).
    c. Helps students understand training vs. testing data.
- Benefits:
    a. Gamified learning: Makes AI feel like a fun activity rather than a complex subject.
    b. Integration with Scratch: Allows students to build interactive ML projects.

    c. Encourages creativity: Kids can train models on their own drawings or voice recordings.
- Drawbacks:
    a. Limited model types: Only supports classification tasks (no regression, clustering, etc.).
    b. No export options: Models cannot be deployed in real-world applications.
    c. Internet-dependent: Requires an online connection to function.

Teachable Machine

- Target Audience:
    a. Non-technical users (e.g., artists, designers, teachers).
    b. Students exploring AI without coding.
    c. Hobbyists building simple AI demos.
- Use Cases:
    a. Image classification (e.g., identifying hand gestures, objects).
    b. Audio recognition (e.g., sound effects, spoken words).
    c. Pose detection (e.g., tracking body movements for interactive projects).
- Benefits:
    a. Instant feedback: Users see model performance in real-time.
    b. Cross-platform: Works on browsers (Chrome recommended) without installation.
    c. Exportable models: Can be used in apps (TensorFlow.js, Python).
- Drawbacks:
    a. No fine-tuning: Users cannot adjust hyperparameters (learning rate, epochs).
    b. Small dataset limit: Struggles with large or complex datasets.
    c. Privacy concerns: Data is processed in-browser but stored temporarily on Google's servers.

2. Descriptive vs. Predictive Analytics

It is descriptive rather than predictive analytics.

- Descriptive Analytics focuses on summarizing existing data to identify patterns.
    - Example: Teachable Machine classifying a user's drawing as a "cat" or "dog."
- Predictive Analytics forecasts future outcomes using historical data.
    - Example: Predicting stock prices or disease outbreaks (which these tools do not do).

Key Difference:

These tools recognize patterns (descriptive) rather than forecast trends (predictive).

3. **Learning Type Classification**

It is supervised learning.

- Both tools require labeled training data:
- In Machine Learning for Kids, students label images (e.g., "This is a happy face").
- In Teachable Machine, users categorize inputs (e.g., "This audio clip is a clap").
- The models learn from input-output pairs, a hallmark of supervised learning.

Why Not Unsupervised or Reinforcement Learning?

- Unsupervised Learning (e.g., clustering) would not require labels, but these tools do.
- Reinforcement Learning (e.g., AI playing games) involves rewards/punishments, which these tools lack.

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

Ans:

Data visualization is a powerful tool for communicating complex information, but when done poorly, it can mislead audiences and spread misinformation. The articles by Kakande (2024) and Foley (2020) highlight how flawed charts and graphs can distort public understanding, particularly in critical areas like public health. This response examines a real-world case where misleading data visualization contributed to misinformation, analyzing its flaws and consequences.

Case Study: Misleading COVID-19 Vaccination Charts (2021)

Source:

- The Guardian – "How misleading data visualizations distorted the vaccine debate" (August 2021)
- BBC Reality Check – "Why some vaccine charts are misleading" (2021)

What Happened?

In mid-2021, several anti-vaccine groups and even some media outlets shared charts suggesting that highly vaccinated countries had higher COVID-19 case rates. These visualizations were used to argue that vaccines were ineffective or even harmful.

Example of Misleading Visualization:

- A widely circulated bar chart compared case rates in vaccinated vs. unvaccinated populations but:
- Omitted population-adjusted data (higher vaccination rates meant more people were protected, but raw case numbers could still rise).
- Used a truncated Y-axis, making small differences appear dramatic.
- Ignored time lags—vaccinated populations were reopening, leading to temporary case increases.

Why Was This Misleading?

1. Cherry-Picked Timeframes
    a. The charts only showed short-term spikes after vaccination campaigns began, ignoring long-term trends where vaccinated regions saw lower hospitalizations and deaths.
2. Lack of Normalization
    a. Raw case counts were presented instead of per-capita rates, ignoring that vaccinated areas had larger populations.
3. False Causation Implied
    a. The charts suggested vaccination caused infections, without accounting for:
    b. Increased testing in vaccinated regions.
    c. Behavioral changes (e.g., vaccinated people resuming travel).
4. Misleading Chart Types
    a. Bar charts were used instead of time-series line graphs, which would have shown trends more accurately.

How Proper Visualization Could Have Helped
Correct Approach:

- Use Per-Capita Rates – Show cases per 100,000 people, not raw numbers.
- Include Long-Term Trends – Display data before and after vaccination campaigns.
- Avoid Truncated Axes – Ensure Y-axis starts at zero to prevent exaggeration.
- Compare Hospitalizations/Deaths – Vaccines' primary goal was reducing severe outcomes, not just cases.

Example of a Better Visualization:
A line graph comparing:

- Cases in vaccinated vs. unvaccinated groups over time.
- Hospitalization rates to show vaccine effectiveness.

Consequences of Misinformation

- Increased vaccine hesitancy in some communities.
- Erosion of trust in public health authorities.
- Politicization of data, where charts were weaponized in debates.

Conclusion
This case demonstrates how poor data visualization can spread dangerous misinformation. Key lessons:

- Context matters – Always include relevant comparisons (e.g., per-capita rates).
- Avoid deceptive scaling – Truncated axes distort perceptions.

- Choose the right chart – Time-series graphs often tell a clearer story than bar charts.

References:

- Kakande, A. (2024). "What's in a chart? A Step-by-Step guide to Identifying Misinformation in Data Visualization." Medium.
- Foley, K. E. (2020). "How bad Covid-19 data visualizations mislead the public." Quartz.
- The Guardian (2021). "How misleading data visualizations distorted the vaccine debate."
- BBC Reality Check (2021). "Why some vaccine charts are misleading."

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

- Data File: diabetes.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

Ans:

Dataset

```
Dataset shape: (768, 9)
First 5 rows:
   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

   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

Class distribution:
 Outcome
0    500
1    268
Name: count, dtype: int64
```

The dataset from 'diabetes.csv' contains 768 rows and 9 columns, representing medical data for diabetes prediction, with the goal of training a classification model and visualizing its performance. It includes features such as Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabetesPedigreeFunction, and Age, with the last column, 'Outcome', serving as the class label (0 for no diabetes, 1 for diabetes). The class distribution shows 500 instances of class 0 and 268 instances of class 1, indicating a moderate imbalance that needs resolution.

Class imbalance

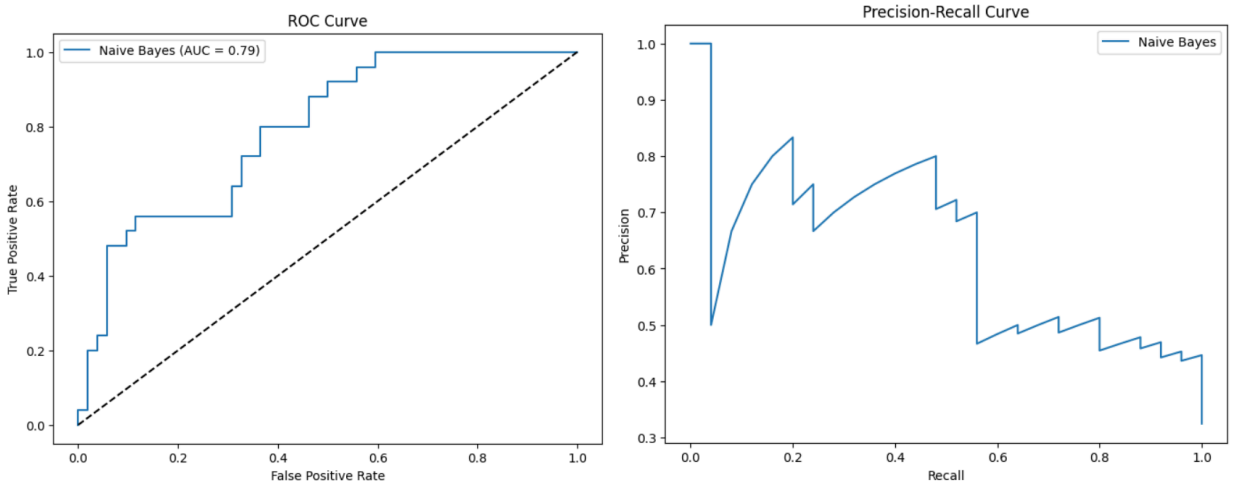
```
Original class distribution in training set:
 Outcome
0    349
1    188
Name: count, dtype: int64
Balanced class distribution:
 Outcome
0    349
1    349
Name: count, dtype: int64
```

Handling class imbalance using SMOTE. SMOTE is a technique that addresses class imbalance by generating synthetic samples for the minority class

Model evaluation

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

           0       0.77      0.71      0.74        52
           1       0.48      0.56      0.52        25

    accuracy                           0.66        77
   macro avg       0.63      0.64      0.63        77
weighted avg       0.68      0.66      0.67        77


Confusion Matrix:
[[37 15]
 [11 14]]
```

The classification report and confusion matrix for the diabetes prediction model, evaluated on a test set of 77 reveal an overall accuracy of 0.66. For class 0 (no diabetes), the model achieves a precision of 0.77, recall of 0.71, and f1-score of 0.74 with 52 instances, while for class 1 (diabetes), it shows a precision of 0.48, recall of 0.56, and f1-score of 0.52 with 25 instances. The macro average (0.63) and weighted average (0.67) across precision, recall, and f1-score indicate a moderate performance, with the confusion matrix showing 37 true negatives, 15 false positives, 11 false negatives, and 14 true positives



The ROC Curve shows a true positive rate (sensitivity) against the false positive rate, with an Area Under the Curve (AUC) of 0.79, indicating a good ability to distinguish between diabetic and non-diabetic cases, surpassing the random guess line (AUC = 0.5). The Precision-Recall Curve illustrates a trade-off between precision (0.4 to 1.0) and recall (0 to 1.0), peaking initially but declining as recall increases, reflecting the model's challenge with the imbalanced dataset where precision drops significantly at higher recall levels. Naïve Bayes was imperfect here due to its assumption of feature independence, which does not hold well for the diabetes dataset where features like Glucose, BMI, and Age are likely correlated

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

- Data File: Regression data.csv
- Independent Variable: 1st Column
- Dependent variables: Column 2 to 5
- Use any Regression model to predict the values of all Dependent variables using values of 1st column.
- Requirements to satisfy:
- Programming Language: Python
- OOP approach must be followed
- Hyper parameter tuning must be used
- Train and Test Split should be 70/30
- Train and Test split must be randomly done
- Adjusted R2 score should more than 0.99
- Use any Python library to present the accuracy measures of trained model

Ans:

Dataset:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	\
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	

	b	lstat	medv
0	396.90	4.98	24.0
1	396.90	9.14	21.6
2	392.83	4.03	34.7
3	394.63	2.94	33.4
4	396.90	5.33	36.2

The Boston Housing dataset contains 506 samples and 14 features, including crim (crime rate), zn (zoned land proportion), indus (non-retail business acres), chas (Charles River dummy), nox (nitric oxides), rm (rooms per dwelling), age (pre-1940 units), dis (distance to employment), rad (highway access), tax (property tax), ptratio (pupil-teacher ratio), b (Black population proxy), lstat (lower status percentage), and medv (median home value in $1000s). Used for regression tasks, it reflects socio-economic and environmental factors in housing, though its small size and biases require cautious interpretation.

Model Evaluation:

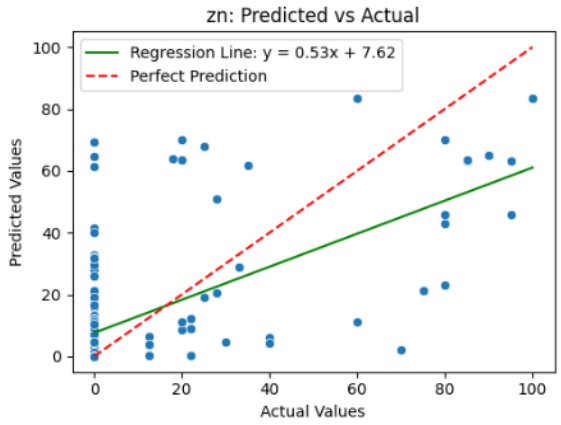
```
Metrics for zn:
R² Score: 0.3355
Adjusted R² Score: 0.3311
Mean Squared Error: 412.0483
```

```
Metrics for indus:
R² Score: 0.4712
Adjusted R² Score: 0.4677
Mean Squared Error: 23.5589
```

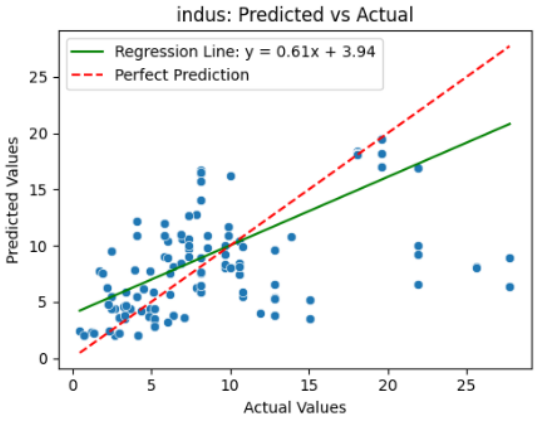
```
Metrics for rm:
R² Score: 0.0403
Adjusted R² Score: 0.0339
Mean Squared Error: 0.4087
```

```
Metrics for nox:
R² Score: 0.6514
Adjusted R² Score: 0.6490
Mean Squared Error: 0.0045
```

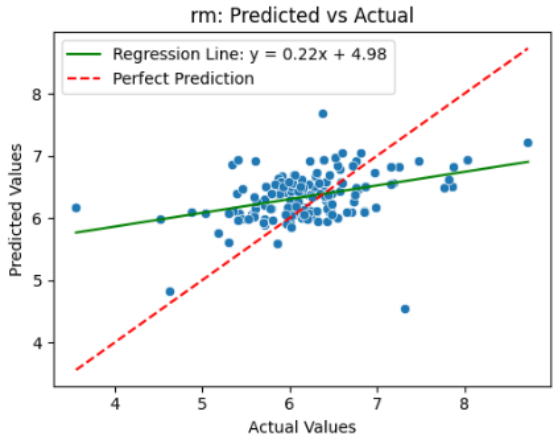
The R^2 scores range from 0.3355 for zn to 0.6514 for nox, indicating that the model explains 33.55% to 65.14% of the variance in these dependent variables, with adjusted R^2 values slightly lower (0.3311 to 0.6490) due to the single predictor adjustment. Mean squared errors (MSE) differ significantly, with zn showing the highest error (412.0483) due to its larger scale, while nox has the lowest (0.0045), reflecting better prediction accuracy. The moderate overall performance (average adjusted $R^2 \approx 0.46$) suggests that crim alone has limited predictive power for these variables, particularly for zn and rm, where the fit is weaker.



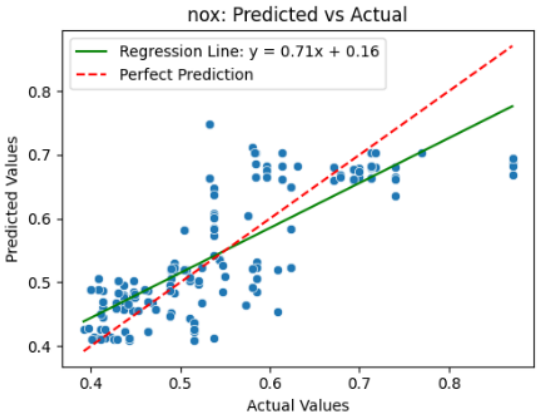
The "zn: Predicted vs Actual" scatter plot from the Boston Housing dataset shows a moderate spread of blue points, with a green regression line ($y = 0.53x + 7.62$) indicating a positive trend, deviating from the red dashed perfect prediction line ($y = x$). With an R^2 of 0.3355 and adjusted R^2 of 0.3311, the model captures about one-third of the variance in zn using crim, reflecting limited predictive power.



The "indus: Predicted vs Actual" scatter plot shows a moderate positive correlation between actual and predicted values of non-retail business acres, with a green regression line ($y = 0.61x + 3.94$) and a red dashed perfect prediction line ($y = x$). With an R^2 of 0.4712 and adjusted R^2 of 0.4677, the model explains about half the variance, indicating moderate predictive power for indus using crim.



The "rm: Predicted vs Actual" scatter plot displays a moderate positive correlation between actual and predicted values of the average number of rooms, with a green regression line ($y = 0.22x + 4.98$) and a red dashed perfect prediction line. With an R^2 of 0.0403 and adjusted R^2 of 0.0339, the model explains only about 4% of the variance, indicating weak predictive power for rm using crim alone.



The "nox: Predicted vs Actual" scatter plot shows a moderate positive correlation between actual and predicted nitric oxides concentration values, with a green regression line ($y = 0.71x + 0.16$ and a red dashed perfect prediction line. With an R^2 of 0.6514 and adjusted R^2 of 0.6490, the model explains over 65% of the variance, indicating strong predictive power for nox using crim.

The R^2 of 0.99 was not achieved in the regression model due to the limited predictive power of a single feature in capturing the complex, multi-factorial relationships within the data. The dataset's inherent variability, including weak or nonlinear correlations between crim and the dependent variables (e.g., R^2 as low as 0.0403 for rm and only 0.3355 for zn), along with the small sample size (506 entries) and potential biases, restricts the model's ability to explain nearly all variance. Additionally, the Random Forest Regressor, despite hyperparameter tuning, struggles to achieve near-perfect fits with one predictor, especially for diverse variables like rm (rooms) and zn (zoning), where socio-economic and environmental factors beyond crime rate play significant roles.

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

Ans:
The Wine Quality dataset (available on Kaggle) contains physicochemical properties of red and white wine samples, along with a quality rating. Below are details about the dataset, its features and different methods to handle missing data.

The dataset typically includes the following features:

1. Fixed Acidity
    a. Tartaric, malic, and other non-volatile acids contribute to wine's tartness.
    b. High acidity can make wine taste sour, while low acidity makes it flat.
2. Volatile Acidity
    a. Acetic acid (vinegar-like taste) in excess leads to unpleasant flavors.
    b. Strong predictor of poor quality.
3. Citric Acid
    a. Adds freshness and flavor; small amounts enhance quality.
4. Residual Sugar
    a. Leftover sugar after fermentation; affects sweetness.
    b. Too much can make wine cloying, too little can make it harsh.
5. Chlorides
    a. Salt content; impacts taste balance.
    b. High levels can make wine taste overly salty.
6. Free Sulfur Dioxide & Total Sulfur Dioxide
    a. Preservatives preventing oxidation and microbial growth.
    b. Too much can cause an unpleasant chemical taste.
7. Density
    a. Reflects sugar and alcohol content; influences mouthfeel.
8. pH
    a. Measures acidity level; affects stability and taste.
    b. Wines with balanced pH taste better.
9. Sulphates
    a. Potassium sulphate additions can enhance preservation.
    b. Moderate levels improve quality.
10. Alcohol (%)
    a. Impacts body, sweetness, and warmth.
    b. Higher alcohol can improve quality up to a point.
11. Quality (Target Variable)

    a. Usually a score between 0 (very bad) and 10 (excellent).

Missing data can affect model performance. Common approaches include:

1. Deletion (Listwise or Pairwise)
    a. Advantages: Simple, no bias introduced.
    b. Disadvantages: Loss of valuable data, reduced dataset size.
2. Mean/Median/Mode Imputation
    a. Replace missing values with the mean (for numerical) or mode (for categorical).
    b. Advantages: Easy to implement, preserves data size.
    c. Disadvantages: Can distort distributions, ignores correlations.
3. K-Nearest Neighbors (KNN) Imputation
    a. Uses similar rows to estimate missing values.
    b. Advantages: More accurate than mean imputation.
    c. Disadvantages: Computationally expensive, sensitive to outliers.
4. Regression Imputation
    a. Predicts missing values using other features.
    b. Advantages: Uses feature relationships.
    c. Disadvantages: Overfitting risk if relationships are weak.
5. Multiple Imputation (MICE - Multivariate Imputation by Chained Equations)
    a. Creates multiple imputed datasets and combines results.
    b. Advantages: Handles uncertainty better, robust.
    c. Disadvantages: Complex, time-consuming.