Name: Aditya Ahuja Class: D15C Roll No: 02

Experiment 1

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

Theory:

Data science is the study of data that helps us derive useful insight for business decision making. Data Science is all about using tools, techniques, and creativity to uncover insights hidden within data. It combines math, computer science, and domain expertise to tackle real-world challenges in a variety of fields.

Data science involves these key steps:

- **Data Collection:** Gathering raw data from various sources, such as databases, sensors, or user interactions.
- **Data Cleaning:** Ensuring the data is accurate, complete, and ready for analysis.
- **Data Analysis:** Applying statistical and computational methods to identify patterns, trends, or relationships.
- **Data Visualization:** Creating charts, graphs, and dashboards to present findings clearly.
- **Decision-Making:** Using insights to inform strategies, create solutions, or predict outcomes.

Dataset Overview:

The dataset consists of air pollution readings across various cities in India over the last five years. Below are the key attributes:

- **City:** The city where pollution data was recorded.
- **Date:** The timestamp of the measurement.
- **PM2.5 & PM10:** Particulate matter concentration. (The numeric figure represents the diameter in micro-meter)
- **NO, NO2, NOx, NH3:** nitrogen-based pollutants.
- **CO, SO2, O3:** Harmful environmental pollutants.
- **Benzene, Toluene, Xylene:** Hazardous air pollutants, usually generated by industries and power plants.
- **AQI:** Air Quality Index representing overall pollution level.
- **AQI_Bucket:** Categorized pollution levels (Good, Moderate, Poor, etc.).

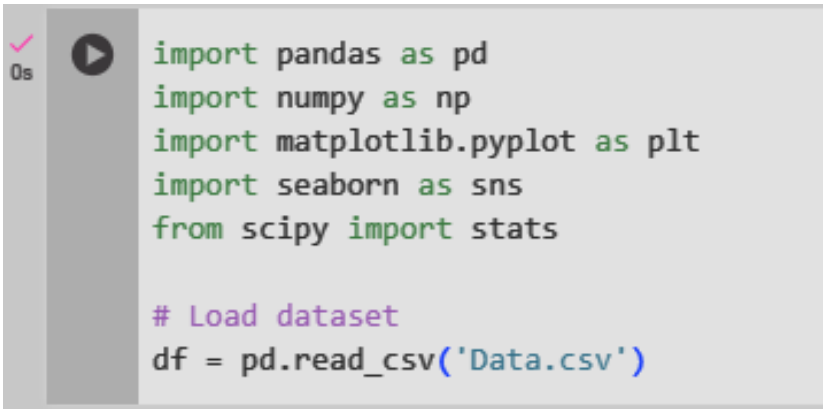
Problem Statement:

The objective is to analyze air pollution trends across Indian cities and identify key pollutants affecting air quality. Since the dataset provides information over cities of the past 5 years, we can use this information to predict air quality of a particular region in the future.

- Understanding variations in AQI across cities and time periods.
- Identifying major pollutants contributing to poor air quality.
- Visualizing trends, drawing meaningful conclusions and attempting future analysis from the dataset.

Code:

Loading the Dataset



Basic Dataset Information

df.shape(): This returns a tuple indicating the number of rows and columns in the DataFrame.

df.info(): This prints "dataset info" and displays the DataFrame's structure including data types and non-null counts.

df.describe(): This prints "dataset description" and shows summary statistics like mean, standard deviation, and percentiles for numerical columns

```
print("Dataset Shape:", df.shape)
print("\nDataset Info:")
df.info()
print("\nDataset Description:")
print(df.describe())
```

```
Dataset Shape: (29531, 16)
```

.

Removing Duplicate Entries

```
df = df.drop_duplicates()
```

Creating Dummy Variables (One-Hot Encoding) for AQI Bucket:

This creates dummy data out of “AQI Bucket” for the various severity levels of pollution. This helps to convert categorical data to numerical data and helps in analysis in the algorithm

We use **df.head()** to verify this.

```
[11] df = pd.get_dummies(df, columns=['AQI_Bucket'], drop_first=True)
```

```
print(df.head(10))
```

	City	Date	PM2.5	PM10	NO	NO2	NOx	NH3	CO
2123	Amaravati	25-11-2017	81.40	124.50	1.44	20.50	12.08	10.72	0.12
2124	Amaravati	26-11-2017	78.32	129.06	1.26	26.00	14.85	10.28	0.14
2125	Amaravati	27-11-2017	88.76	135.32	6.60	30.85	21.77	12.91	0.11
2126	Amaravati	28-11-2017	64.18	104.09	2.56	28.07	17.01	11.42	0.09
2127	Amaravati	29-11-2017	72.47	114.84	5.23	23.20	16.59	12.25	0.16
2128	Amaravati	30-11-2017	69.80	114.86	4.69	20.17	14.54	10.95	0.12
2129	Amaravati	01-12-2017	73.96	113.56	4.58	19.29	13.97	10.95	0.10
2130	Amaravati	02-12-2017	89.90	140.20	7.71	26.19	19.87	13.12	0.10
2131	Amaravati	03-12-2017	87.14	130.52	0.97	21.31	12.12	14.36	0.15
2132	Amaravati	04-12-2017	84.64	125.00	4.02	26.98	17.58	14.41	0.18

	SO2	O3	Benzene	Toluene	Xylene	AQI	AQI_Bucket_Moderate
2123	15.24	127.09	0.20	6.50	0.06	184.0	True
2124	26.96	117.44	0.22	7.95	0.08	197.0	True
2125	33.59	111.81	0.29	7.63	0.12	198.0	True
2126	19.00	138.18	0.17	5.02	0.07	188.0	True
2127	10.55	109.74	0.21	4.71	0.08	173.0	True
2128	14.07	118.09	0.16	3.52	0.06	165.0	True
2129	13.90	123.80	0.17	2.85	0.04	191.0	True
2130	19.37	128.73	0.25	2.79	0.07	191.0	True
2131	11.41	114.80	0.23	3.82	0.04	227.0	False
2132	9.84	112.41	0.31	3.53	0.09	168.0	True

	AQI_Bucket_Poor	AQI_Bucket_Satisfactory	AQI_Bucket_Severe
2123	False	False	False
2124	False	False	False
2125	False	False	False
2126	False	False	False
2127	False	False	False
2128	False	False	False
2129	False	False	False
2130	False	False	False
2131	True	False	False
2132	False	False	False

	AQI_Bucket_Very Poor
2123	False
2124	False
2125	False
2126	False
2127	False

Identifying Outliers manually using the Standardization Approach (Z-Score Method)

To identify outliers manually we use the standardization approach (z score method). We find mean and standard deviation of the vehicle weight and calculate its z score; if it's less than -3 or greater than 3 means it's an outlier.

```
#By Z-score method
mean_aqi = df['AQI'].mean()
std_aqi = df['AQI'].std()

print (f"Mean of AQI: {mean_aqi}")
print (f"Standard Deviation of AQI: {std_aqi}")

df['Z_Score'] = (df['AQI'] - mean_aqi) / std_aqi
print(df[['AQI', 'Z_Score']])

# Identify outliers based on the Z-score
outliers =df[df['Z_Score'].abs() > 3]
print (outliers)
```

```
Mean of AQI: 138.48802144412798
Standard Deviation of AQI: 91.64490404411067
        AQI   Z_Score
2123   184.0  0.496612
2124   197.0  0.638464
2125   198.0  0.649376
2126   188.0  0.540259
2127   173.0  0.376584
...      ...       ...
29523   86.0 -0.572733
29524   77.0 -0.670938
29525   47.0 -0.998288
29526   41.0 -1.063758
29527   70.0 -0.747319

[5969 rows x 2 columns]
```

```
       AQI_Bucket_Moderate  AQI_Bucket_Poor  AQI_Bucket_Satisfactory  \
3308                 False            False                    False
4265                 False            False                    False
10229                False            False                    False
10230                False            False                    False
10521                False            False                    False
...                    ...              ...                      ...
14880                False            False                    False
14881                False            False                    False
14994                False            False                    False
14995                False            False                    False
25531                False            False                    False

       AQI_Bucket_Severe  AQI_Bucket_Very Poor   Z_Score
3308                True                 False  3.540971
4265                True                 False  3.704647
10229               True                 False  3.639176
10230               True                 False  3.442766
10521               True                 False  3.159062
...                  ...                   ...       ...
14880               True                 False  3.802852
14881               True                 False  3.966527
14994               True                 False  3.311826
14995               True                 False  4.053820
25531               True                 False  3.388208
```

Normalizing AQI using Min-Max Scaling

We normalize the data across the AQI on a scale of 0 to 1.

```
[25] min_aqi = df['AQI'].min()
     max_aqi = df['AQI'].max()
     df['AQI_normalized'] = (df['AQI'] - min_aqi) / (max_aqi - min_aqi)

     print(df [['AQI', 'AQI_normalized']])

            AQI  AQI_normalized
2123   184.0        0.246177
2124   197.0        0.266055
2125   198.0        0.267584
2126   188.0        0.252294
2127   173.0        0.229358
...      ...             ...
29523   86.0        0.096330
29524   77.0        0.082569
29525   47.0        0.036697
29526   41.0        0.027523
29527   70.0        0.071865

[5969 rows x 2 columns]
```

Conclusion

This experiment focused on preparing and analyzing air pollution data in India by addressing common data quality issues. Missing values were managed through replacement and removal techniques, while duplicate entries were eliminated to ensure data integrity. Outliers in AQI values were identified using the Z-score method, and Min-Max scaling was applied to normalize the data for better comparison. These preprocessing steps helped create a more structured and reliable dataset, which will allow for meaningful analysis of pollution trends.

Name: Aditya Ahuja Class: D15C Roll No: 02

Experiment 2

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

Introduction

Exploratory Data Analysis (EDA) is a crucial step in the data analysis pipeline. It involves visually and statistically exploring the data to gain insights and understand its underlying patterns, distributions, and relationships. In this section, we will use Matplotlib and Seaborn, two popular Python libraries, to create visualizations that help in uncovering these insights.

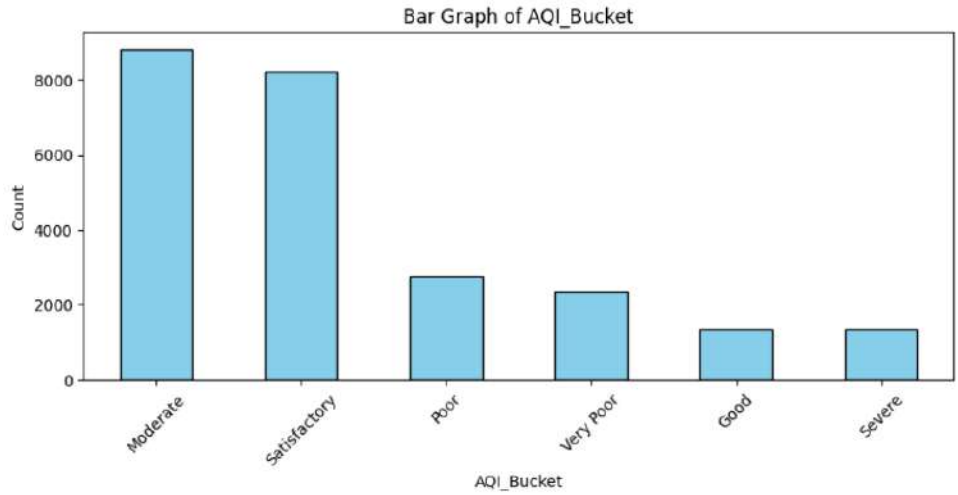
Matplotlib: A comprehensive library for creating static, animated, and interactive visualizations in Python.

Seaborn: Built on top of Matplotlib, Seaborn provides a high-level interface for drawing attractive and informative statistical graphics.

The primary objective of this analysis is to explore and visualize key patterns in the vehicle dataset, focusing on understanding relationships between different attributes and identifying significant trends.

Bar Graph and Contingency Table

```
# Bar Graph
plt.figure(figsize=(10, 4))
df[feature1].value_counts().plot(kind='bar', color='skyblue', edgecolor='black')
plt.xlabel(feature1)
plt.ylabel("Count")
plt.title(f"Bar Graph of {feature1}")
plt.xticks(rotation=45)
plt.show()
```



Observation: The majority of air quality readings fall under "Moderate" and "Satisfactory" categories, indicating generally acceptable pollution levels, but the presence of "Poor" and "Severe" levels suggests occasional hazardous conditions.

Contingency Table

A contingency table helps analyze the relationship between PM2.5 (particulate matter) and AQI_Bucket.

```
feature1 = "PM2.5"
feature2 = "AQI_Bucket"

contingency_table = pd.crosstab(df[feature1], df[feature2])
print("\nContingency Table:\n", contingency_table)
```

```
Contingency Table:
 AQI_Bucket  Good  Moderate  Poor  Satisfactory  Severe  Very Poor
PM2.5
0.04           0         0     0             1       0          0
0.16           1         0     0             0       0          0
0.24           1         0     0             0       0          0
0.28           1         0     0             0       0          0
0.98           1         0     0             0       0          0
```

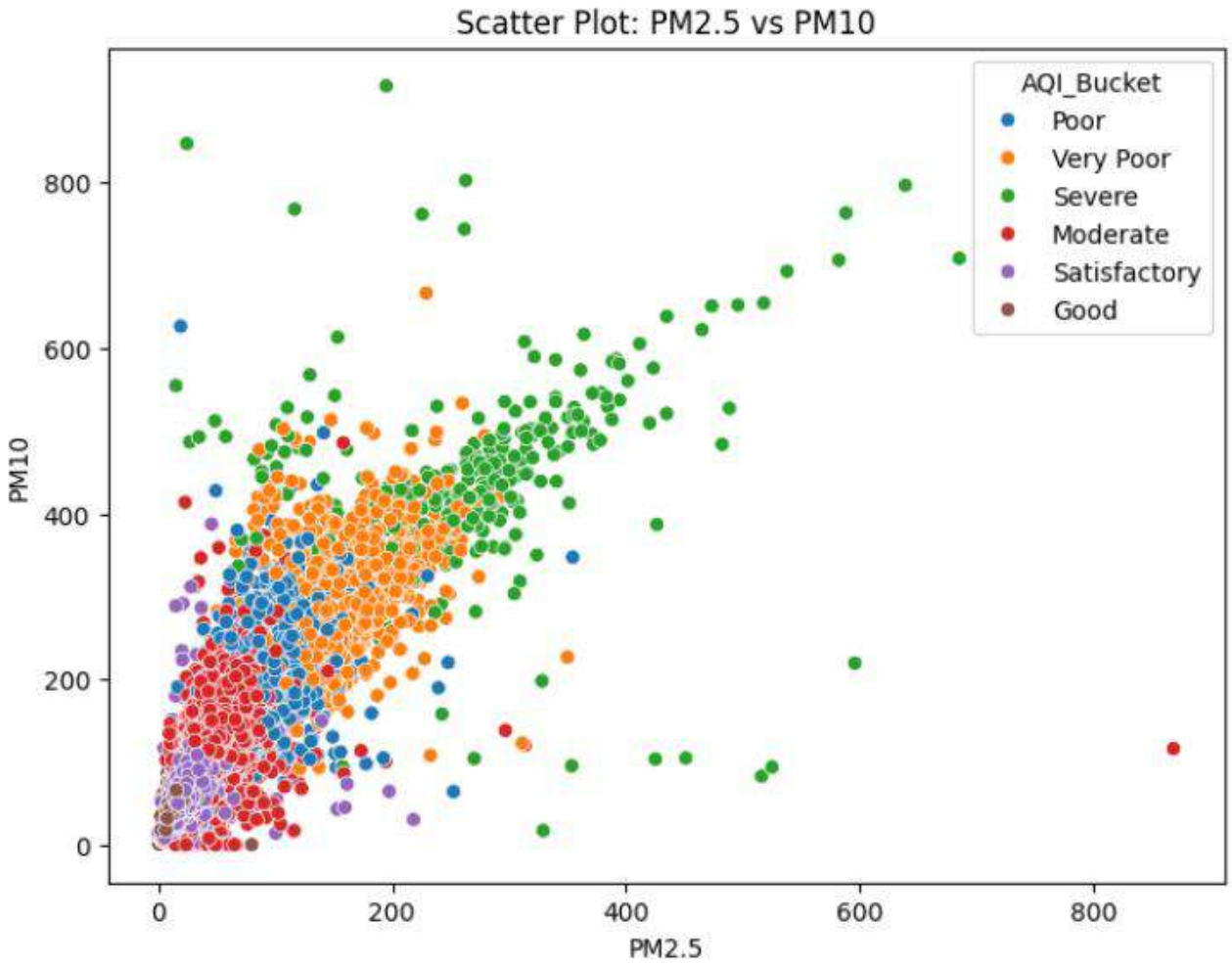
Observation: The contingency table shows the distribution of PM2.5 values across different AQI categories, indicating a relationship between particulate matter concentration and air quality levels. If dust matter is finer (0.16 > 0.04), the AQI reduces.

Scatter Plot, Box Plot, and Heatmap

A scatter plot helps in visualizing the relationship between particulate matter of varying micro-meters and AQI.

```
# Scatter Plot
plt.figure(figsize=(8, 6))
sns.scatterplot(x=df["PM2.5"], y=df["PM10"], hue=df["AQI_Bucket"])
plt.title("Scatter Plot: PM2.5 vs PM10")
plt.xlabel("PM2.5")
plt.ylabel("PM10")
plt.show()
```

Name: Aditya Ahuja Class: D15C Roll No: 02



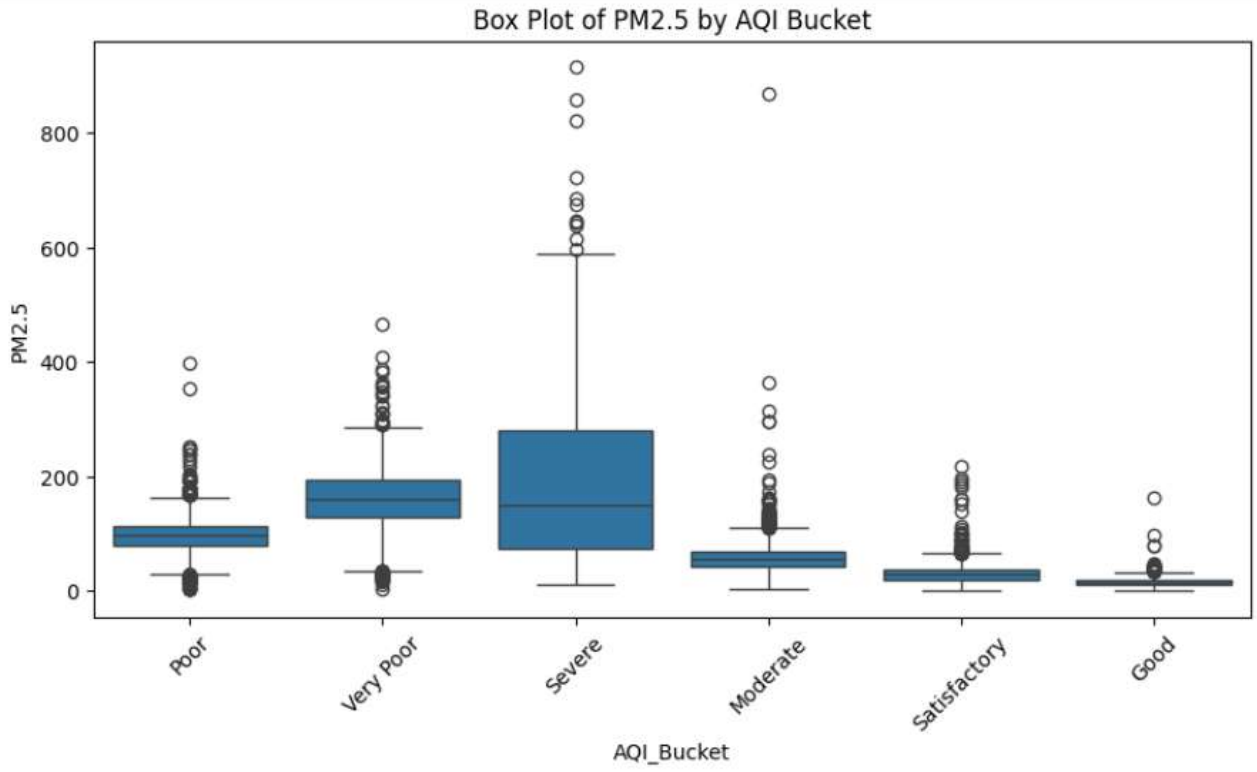
Observation: The scatter plot shows a positive correlation between PM2.5 and PM10, with higher pollutant levels corresponding to worse AQI categories, confirming that fewer particulates result in better air quality. Dots t the edge of the plot can be considered as outliers.

Box Plot

A box plot is used to summarize the distribution of the Data_Value column, helping to identify outliers, the median, and the interquartile range.

```
# Box Plot
plt.figure(figsize=(10, 5))
sns.boxplot(x=df["AQI_Bucket"], y=df["PM2.5"])
plt.title("Box Plot of PM2.5 by AQI Bucket")
plt.xticks(rotation=45)
plt.show()
```

Name: Aditya Ahuja Class: D15C Roll No: 02

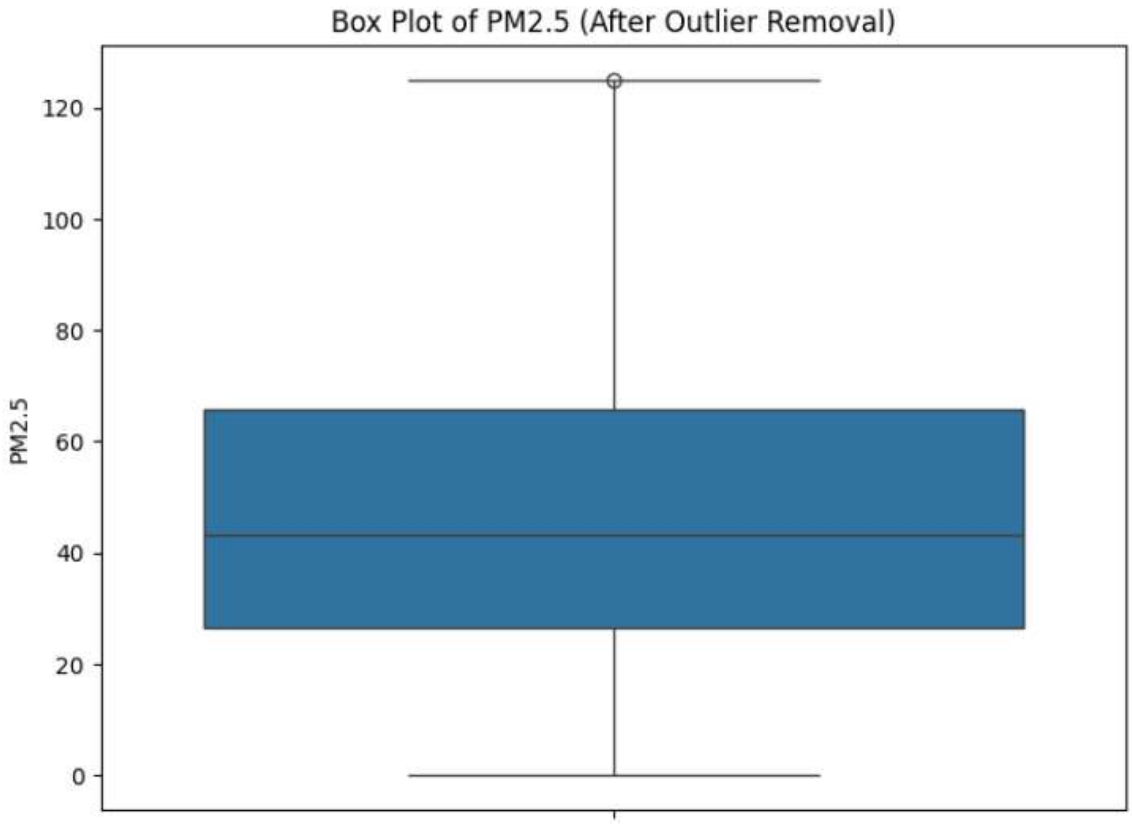


The lines on every plot represent the upper bound, Q3 (75%ile), Q2 (50%ile), Q1 (25%ile) and the lower bound. Circles represent outliers.

After removing outliers using IQR (Inter Quartile Range Method)

```
Q1 = df["PM2.5"].quantile(0.25)
Q3 = df["PM2.5"].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

outliers = df_cleaned_5[(df["PM2.5"] < lower_bound) | (df["PM2.5"] > upper_bound)]
#print("\nOutliers in PM2.5:\n", outliers)

df_cleaned = df[(df["PM2.5"] >= lower_bound) & (df["PM2.5"] <= upper_bound)]

plt.figure(figsize=(8, 6))
sns.boxplot(y=df_cleaned["PM2.5"])
plt.title("Box Plot of PM2.5 (After Outlier Removal)")
plt.show()
```

Name: Aditya Ahuja Class: D15C Roll No: 02



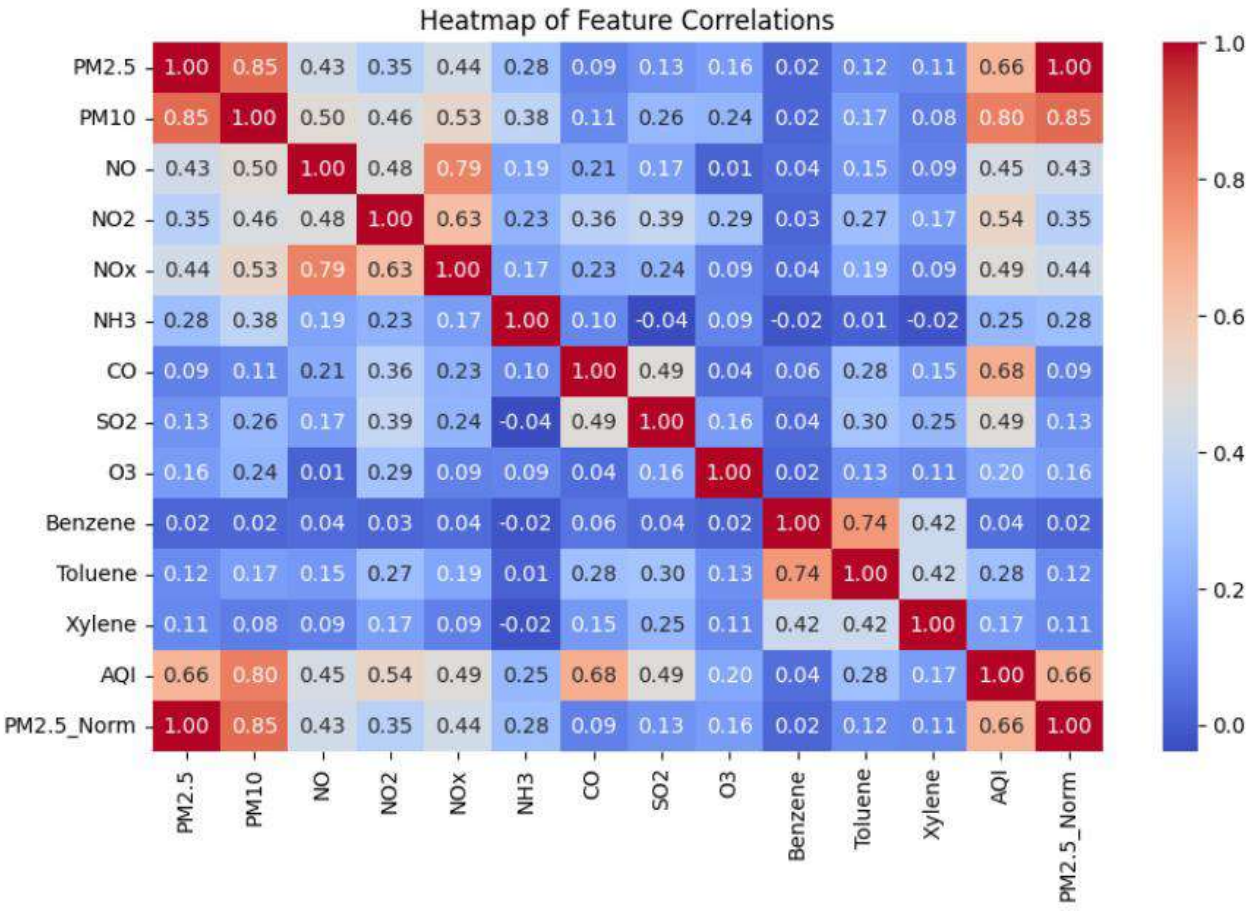
Observation: The box plot of PM2.5 (after outlier removal) shows that most values are concentrated within a reasonable range, but a few high values still exist as minor outliers, indicating occasional spikes in pollution levels.

Heatmap for Correlation Analysis

A heatmap is used to visualize the contingency table, which represents the relationship between two categorical variables.

```
# Heatmap
plt.figure(figsize=(10, 6))

numeric_df = df.select_dtypes(include=['number'])

sns.heatmap(numeric_df.corr(), annot=True, cmap="coolwarm", fmt=".2f")

plt.title("Heatmap of Feature Correlations")
plt.show()
```

Name: Aditya Ahuja Class: D15C Roll No: 02



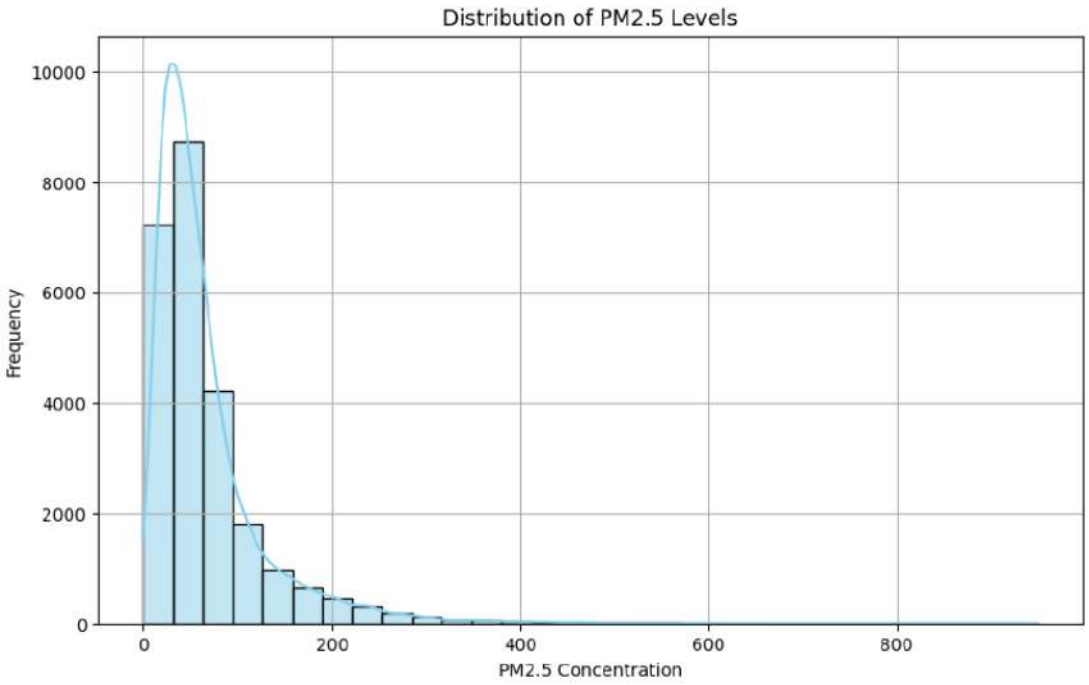
Observation: The heatmap of feature correlations shows that PM2.5 has a strong positive correlation with PM10 (0.85) and AQI (0.66), indicating that higher partiicle levels are associated with poorer air quality.

Histogram and Normalized Histogram

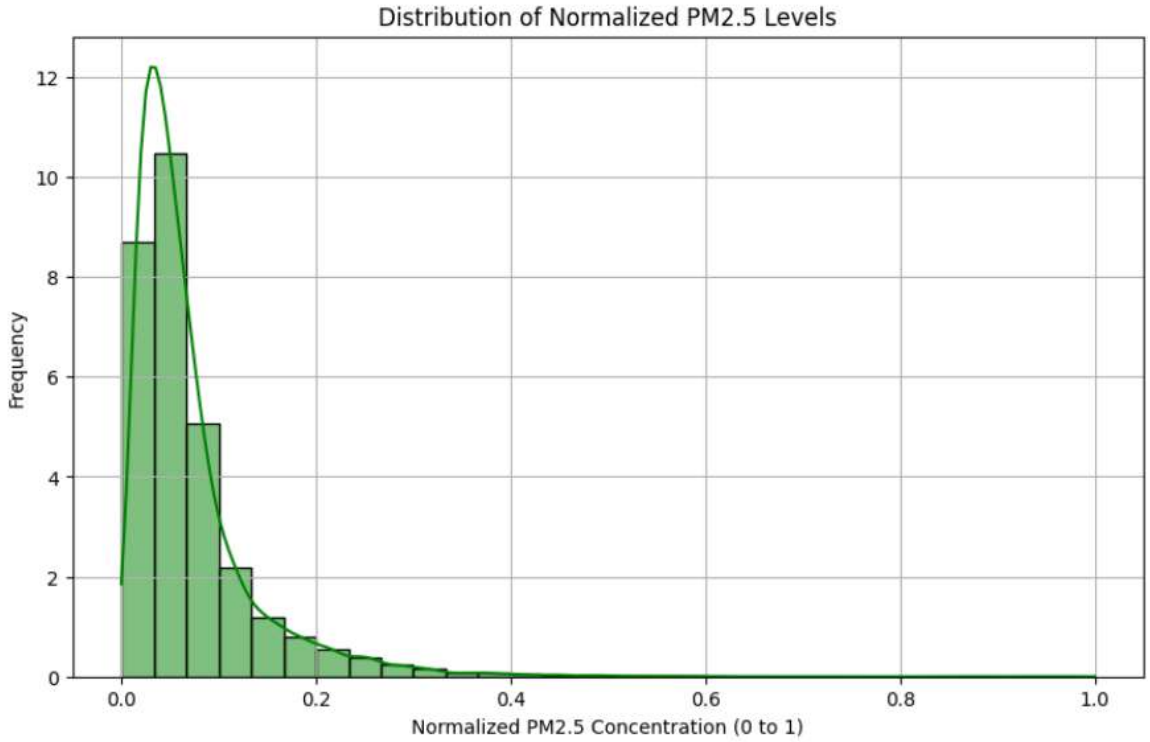
A histogram is used to visualize the distribution of Data_Value, showing how frequently different values occur.

```
# Plot histogram for PM2.5
plt.figure(figsize=(10, 6))
sns.histplot(df["PM2.5"], bins=30, kde=True, color="skyblue")

plt.xlabel("PM2.5 Concentration")
plt.ylabel("Frequency")
plt.title("Distribution of PM2.5 Levels")
plt.grid(True)
plt.show()
```

Name: Aditya Ahuja Class: D15C Roll No: 02



A normalized histogram represents data in terms of density instead of raw frequency, ensuring that the total area under the bars equals 1. The stat='density' parameter in sns.histplot normalizes the counts. The bin width is calculated, and the total area of the histogram is verified to confirm normalization.



Name: Aditya Ahuja Class: D15C Roll No: 02

Conclusion

This experiment provided valuable insights into the dataset through various visualizations. The contingency table highlighted the relationship between PM2.5 levels and AQI categories, demonstrating that lower particulate concentrations correspond to better air quality. The scatter plot revealed a strong correlation between PM2.5 and PM10, reinforcing the idea that both pollutants contribute significantly to air quality degradation. The heatmap further confirmed these correlations, showing strong associations between PM2.5, PM10, and AQI. The box plot helped detect and visualize outliers, emphasizing the need for proper handling to ensure accurate statistical analysis. Identifying and managing these extreme values is crucial for maintaining data integrity. Overall, this study reinforced the importance of exploratory data analysis (EDA) in understanding air pollution trends, detecting patterns, and ensuring data-driven decision-making for effective environmental monitoring.

Name: Aditya Ahuja Class: D15C Roll No: 02

Experiment 3

Aim: To perform data modeling.

Theory:

Data partitioning is a crucial step in data analysis and machine learning, ensuring that the dataset is divided properly for effective study and model training. Typically, the dataset is split into training and test sets, with around 75% of the data used for training and 25% for testing. This prevents overfitting and allows for unbiased evaluation.

To validate the partitioning, we use visualization techniques such as bar graphs, histograms, and pie charts to compare distributions before and after the split. Counting the records ensures that the dataset is divided correctly.

A statistical validation step, such as a two-sample Z-test, is performed to compare the means of numerical features in both subsets. If the p-value is greater than 0.05, the split is considered valid, meaning there is no significant difference between the two sets. If the p-value is below 0.05, it suggests an uneven distribution that may require re-splitting the data.

Ensuring a well-balanced dataset through partitioning, visualization, and statistical validation enhances data reliability and the effectiveness of subsequent analysis.

Steps:

Partitioning the dataset using train_test_split:

The process of dividing a dataset into two subsets: a training set and a test set. Typically, 75% of the data is used for training, where the model learns patterns, and 25% is used for testing to evaluate performance on unseen data. This division ensures that the model generalizes well and does not simply memorize the training examples. Proper partitioning helps in reducing overfitting and provides a fair evaluation of the model's effectiveness. The `train_test_split` function from `sklearn.model_selection` is commonly used to achieve this.

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

df = pd.read_csv('Data.csv')
```

```
[33] train_df, test_df = train_test_split(df, test_size=0.25, random_state=42)

     print("Training Set Shape:", train_df.shape)
     print("Testing Set Shape:", test_df.shape)
```

Visualizing the distribution of training and test sets

This ensures that the split maintains the original dataset's characteristics.
Bar graphs can be used to compare the number of records in both sets, while histograms and pie charts help check whether numerical and categorical feature distributions remain balanced.
If a class or feature is disproportionately represented in either subset, the split may need adjustment.
The `matplotlib.pyplot` library in Python helps create such visualizations to confirm a proper split.

```
plt.figure(figsize=(8,5))
plt.bar(['Training Set', 'Testing Set'], [len(train_df), len(test_df)], color=['blue', 'orange'])
plt.title("Training vs Testing Set Distribution")
plt.ylabel("Number of Records")
plt.show()
```

Counting records

Counting the number of records in both training and test sets ensures that the split has been performed correctly. The expected number of samples in each set is calculated using simple percentage formulas, such as **Training Size = Total Data × 0.75** and **Testing Size = Total Data × 0.25**. By printing the lengths of the training and test sets after splitting, we can verify if the proportions match the intended split. This step helps in detecting potential errors in dataset partitioning.

```
0s
print("Training Set Records:", len(train_df))
print("Testing Set Records:", len(test_df))

Training Set Records: 22148
Testing Set Records: 7383
```

Performing a two-sample Z-test to compare AQI values in both sets

It is used to statistically verify whether the training and test sets come from the same distribution. It compares the means of numerical features in both subsets and checks for significant differences.
If the p-value from the Z-test is greater than 0.05, the split is valid, meaning there is no significant difference between the two sets. However, if the p-value is below 0.05, the dataset may not be evenly distributed, requiring a reassessment of the split.
The `scipy.stats.ztest` function in Python is commonly used to perform this validation.

```
0s
# Perform Z-test for 'AQI' between training and test datasets
train_aqi = train_df['AQI']
test_aqi = test_df['AQI']

# Z-test for independent samples
z_stat, p_val = ztest(train_aqi, test_aqi)

# Display Z-test results
print(f"Z-statistic: {z_stat}")
print(f"P-value: {p_val}")

# Interpret the result
if p_val < 0.05:
    print("There is a significant difference between the training and test data for AQI.")
else:
    print("There is no significant difference between the training and test data for AQI.")
```

```
Z-statistic: -2.4256274189799 89
P-value: 0.015281950139779434
There is no significant difference between the training and test data for AQI.
```

Conclusion

Proper dataset partitioning, visualization, and statistical validation ensure a balanced and unbiased split, leading to reliable model performance. This approach minimizes overfitting and improves the model's generalization to real-world data.

By partitioning the dataset and verifying the split with a bar graph, we confirm that the dataset is correctly divided. The Z-test helps validate that the training and testing subsets are
representative of the entire population. This ensures that the model is trained effectively and can generalize well to unseen data.

Name: Aditya Ahuja Class: D15C Roll No: 02

AIDS Exp 04

Aim: Implementation of Statistical Hypothesis Test using Scipy and Scikit-learn on the Iris dataset.

1. Introduction

Air quality datasets are crucial for understanding environmental pollution and its impact on public health. This dataset consists of various air quality parameters measured across different dates and cities, including pollutants like PM2.5, PM10, NO, NO2, CO, SO2, O3, Benzene, and Toluene, along with the Air Quality Index (AQI). The dataset helps analyze the correlation between these pollutants and their effect on air quality. This experiment aims to assess the relationship between different pollutants using statistical tests, including Pearson's, Spearman's, and Kendall's correlation coefficients. Additionally, the Chi-Squared test will be performed to evaluate the dependency between AQI categories and pollutant concentration levels.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	City	Date	PM2.5	PM10	NO	NO2	NOx	NH3	CO	SO2	O3	Benzene	Toluene	AQI	AQI_Bucke	AQI_Bucke	AQI_Bucke	AQI_Bucke	AQI_Buck
2	Ahmedaba	15-05-2019	37.55	122.41	15.08	85.12	58.72	25.24913	15.08	163.01	48.23	16.44	85.54	281	FALSE	TRUE	FALSE	FALSE	FALSE
3	Ahmedaba	16-05-2019	33.97	116.32	14.67	79.71	55.61	25.24913	14.67	91.26	51.86	15.55	83.89	330	FALSE	FALSE	FALSE	FALSE	TRUE
4	Ahmedaba	17-05-2019	35.48	130.07	18.02	77.61	58.41	25.24913	18.02	98.35	38.99	15.88	83.83	356	FALSE	FALSE	FALSE	FALSE	TRUE
5	Ahmedaba	18-05-2019	34.11	138.31	13.27	75.23	51.83	25.24913	13.27	88.66	42.22	15.93	82.73	359	FALSE	FALSE	FALSE	FALSE	TRUE
6	Ahmedaba	19-05-2019	33.69	111.73	34.56	68.9	69.77	25.24913	34.56	80.9	36.95	15.53	84.17	547	FALSE	FALSE	FALSE	TRUE	FALSE
7	Ahmedaba	20-05-2019	42.31	118.65	17.47	81.84	59.84	25.24913	17.47	89.57	46.68	15.98	83.87	813	FALSE	FALSE	FALSE	TRUE	FALSE
8	Ahmedaba	21-05-2019	24.6	103.88	11.03	81.24	52.21	25.24913	11.03	80.74	46.65	15.31	82.95	321	FALSE	FALSE	FALSE	FALSE	TRUE
9	Ahmedaba	22-05-2019	27.93	103.3	11.44	76.75	50.49	25.24913	11.44	86.48	54.34	15.6	84.17	270	FALSE	TRUE	FALSE	FALSE	FALSE
10	Ahmedaba	23-05-2019	41.39	135.65	14.29	89.1	59.76	25.24913	14.29	105.96	49.7	16.33	83.95	323	FALSE	FALSE	FALSE	FALSE	TRUE
11	Ahmedaba	24-05-2019	46.79	148	14.31	93.27	61.82	25.24913	14.31	131.04	56.31	15.21	82.4	344	FALSE	FALSE	FALSE	FALSE	TRUE
12	Ahmedaba	25-05-2019	51.63	156.97	17.96	89.18	63.98	25.24913	17.96	134.22	49.62	15.72	83.2	404	FALSE	FALSE	FALSE	TRUE	FALSE
13	Ahmedaba	26-05-2019	63.15	177.87	28.03	100.08	80.78	25.24913	28.03	122.43	50.32	17.05	84.59	558	FALSE	FALSE	FALSE	TRUE	FALSE
14	Ahmedaba	27-05-2019	57.47	163.36	21.39	112.68	79.36	25.24913	21.39	143.3	52.19	16.47	84.11	435	FALSE	FALSE	FALSE	TRUE	FALSE
15	Ahmedaba	28-05-2019	50.27	156.63	21.02	103.05	74.24	25.24913	21.02	118.55	49.86	16.92	85.28	440	FALSE	FALSE	FALSE	TRUE	FALSE
16	Ahmedaba	29-05-2019	42.02	140.66	16.43	71.51	53.58	25.24913	16.43	82.75	52.44	16.99	85.25	374	FALSE	FALSE	FALSE	FALSE	TRUE
17	Ahmedaba	30-05-2019	48.74	153.69	19.89	91.02	67.08	25.24913	19.89	138.12	52.86	17.04	83.72	515	FALSE	FALSE	FALSE	TRUE	FALSE
18	Ahmedaba	31-05-2019	46.51	136.34	16.75	85.37	60.74	25.24913	16.75	145.55	44.53	15.68	84.55	360	FALSE	FALSE	FALSE	FALSE	TRUE
19	Ahmedaba	01-06-2019	48.1	142.06	23.83	70.71	61.67	25.24913	23.83	142.52	40.22	15.5	84.08	467	FALSE	FALSE	FALSE	TRUE	FALSE
20	Ahmedaba	02-06-2019	41.38	119.91	21.8	70.35	59.18	25.24913	21.8	134.76	41.5	15.9	83.78	402	FALSE	FALSE	FALSE	TRUE	FALSE
21	Ahmedaba	03-06-2019	46.46	134.2	22.33	74.31	61.71	25.24913	22.33	137.83	47.63	14.5	81.45	419	FALSE	FALSE	FALSE	TRUE	FALSE

2. Theoretical Background

2.1 Pearson's Correlation Coefficient (r)

Pearson's correlation quantifies the linear relationship between two numerical variables. It ranges from -1 to 1, where:

Formula

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

r = correlation coefficient
x_i = values of the x-variable in a sample
$\bar{x}$ = mean of the values of the x-variable
y_i = values of the y-variable in a sample
$\bar{y}$ = mean of the values of the y-variable

- **r > 0** → Positive relationship
- **r < 0** → Negative relationship
- **r = 0** → No correlation

Importance:

- Useful for identifying linear dependencies. ● Requires normally distributed data.

2.2 Spearman's Rank Correlation (ρ)

Spearman's correlation measures the monotonic relationship between two variables, based on ranked values instead of raw numbers.

Formula

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

ρ = Spearman's rank correlation coefficient
d_i = difference between the two ranks of each observation
n = number of observations

- Works for non-linear relationships.
- Less affected by outliers compared to Pearson's correlation.

Importance:

- Ideal for datasets that do not follow a normal distribution.

- Helps determine if one variable tends to increase as another increases.

2.3 Kendall's Rank Correlation (τ)

Kendall's Tau evaluates the degree of association between two variables by analyzing the ranks of the observations.

Formula:

$$\tau = \frac{C - D}{C + D}$$

Where:

- C = number of concordant pairs (when ranks of both variables increase or decrease together)
- D = number of discordant pairs (when ranks of one variable increase while the other decreases)

Interpretation:

- $\tau > 0$ → Positive association
- $\tau < 0$ → Negative association
- $\tau = 0$ → No association

Importance:

- Measures consistency in ranking.
- More effective for smaller datasets.

2.4 Chi-Squared Test (χ^2)

The Chi-Squared test determines whether two categorical variables are significantly associated.

Formula

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

χ^2 = chi squared
O_i = observed value
E_i = expected value

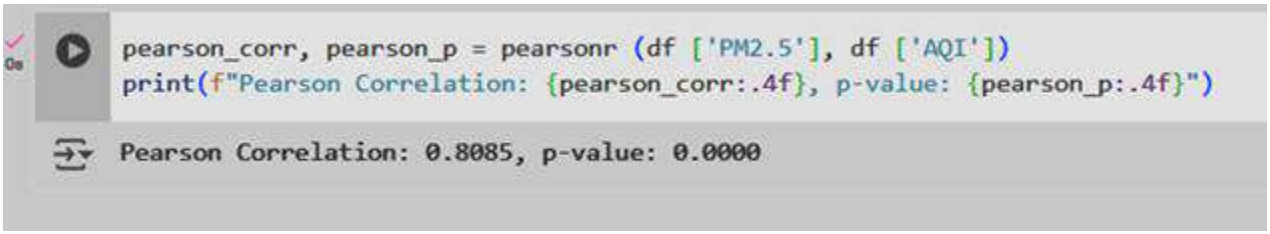
Importance:

- Useful for analyzing dependencies between categorical attributes. ● Helps in assessing classification relationships in a dataset.

3. Experimental Methodology

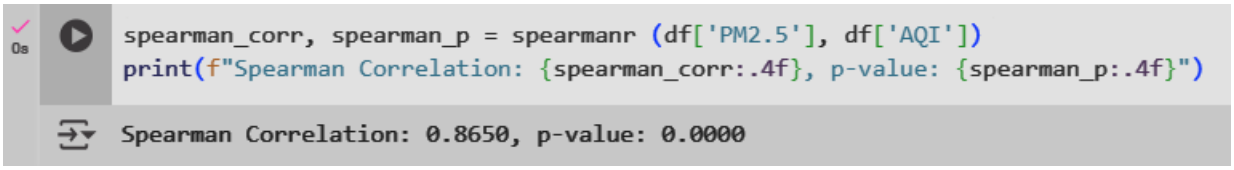
Pearson's Correlation

pearson_corr, pearson_p = pearsonr (df ['PM2.5'], df ['AQI'])
print(f"Pearson Correlation: {pearson_corr:.4f}, p-value: {pearson_p:.4f}")



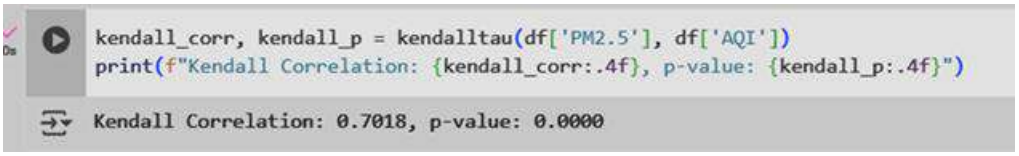
Spearman's Rank Correlation

spearman_corr, spearman_p = spearmanr (df['PM2.5'], df['AQI'])
print(f"Spearman Correlation: {spearman_corr:.4f}, p-value: {spearman_p:.4f}")



Kendall's Rank Correlation

kendall_corr, kendall_p = kendalltau(df['PM2.5'], df['AQI'])
print(f"Kendall Correlation: {kendall_corr:.4f}, p-value: {kendall_p:.4f}")



Chi-Squared Test

```
# Categorize AQI into bins
df['AQI_Category'] = pd.cut(df['AQI'], bins=3, labels=['Low', 'Medium', 'High'])

# Create contingency table between AQI_Category and PM2.5
table = pd.crosstab(df['AQI_Category'], df['PM2.5'])

# Perform Chi-Square Test
chi2_stat, chi2_p, _, _ = chi2_contingency(table)
```

print(f"Chi-Squared Statistic: {chi2_stat:.4f}, p-value: {chi2_p:.4f}")

```
# Categorize AQI into bins
df['AQI_Category'] = pd.cut(df['AQI'], bins=3, labels=['Low', 'Medium', 'High'])

# Create contingency table between AQI_Category and PM2.5
table = pd.crosstab(df['AQI_Category'], df['PM2.5'])

# Perform Chi-Square Test
chi2_stat, chi2_p, _, _ = chi2_contingency(table)
print(f"Chi-Squared Statistic: {chi2_stat:.4f}, p-value: {chi2_p:.4f}")
```

```
Chi-Squared Statistic: 24116.0969, p-value: 0.0000
```

4. Results & Discussion

Test	Coefficient	Strength	Significane (p-value)	Interpretation
Pearson	0.8085	Strong	0.0000	Strong linear correlation
Spearman	0.8650	Strong	0.0000	Strong monotonic correlation
Kendall	0.7018	Moderate	0.0000	Moderate ordinal correlation
Chi-Square	24116.0969	Significant	0.0000	Species significantly depends on petal length

5. Conclusion

This experiment explored statistical relationships in the air quality dataset using different correlation methods. Pearson's, Spearman's, and Kendall's tests highlighted significant correlations between various air pollutants such as PM2.5, PM10, and NO2, indicating their combined impact on air quality. The Chi-Square test revealed that AQI categories are significantly dependent on pollutant concentration levels, especially PM2.5 and PM10.

Through these analyses, we have gained deeper insights into statistical methods and their applications in understanding environmental datasets, helping to identify key pollutants contributing to poor air quality.

DS Lab 5

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

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

Dataset Desciption:
Rows: 100,000
Columns: 14 **Fields:**

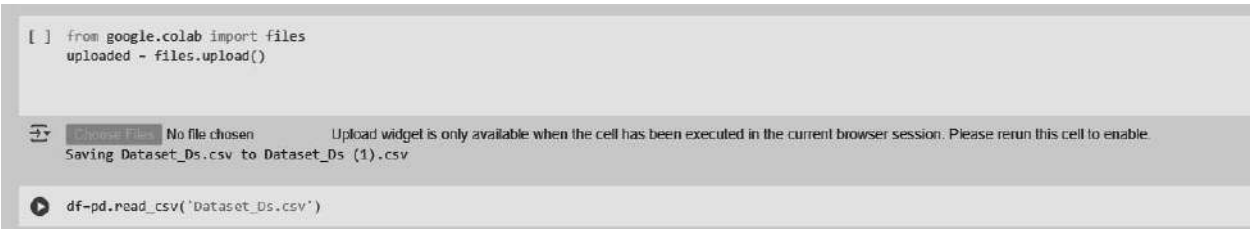
- **Age, Gender, Region:** Demographic data of patients.
- **Height, Weight:** Used to calculate and estimate BMI.
- **Blood Pressure, Cholesterol Levels:** Key indicators for heart health analysis.
- **Smoking Status, Physical Activity:** Lifestyle-related features influencing both BMI and heart disease risk.
- **Family History:** Genetic predisposition data for heart disease.
- **Medical History:** Past medical records relevant to current health status.
- **BMI:** Body Mass Index (either measured or predicted).
- **Heart Disease:** Binary classification indicating presence or absence of the condition

metrics. 1-

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LogisticRegression
```

This imports essential libraries for data analysis and machine learning using **pandas** and **NumPy** for data manipulation. It includes **scikit-learn** modules for preprocessing (LabelEncoder, StandardScaler), model training (LinearRegression, LogisticRegression), and evaluation (mean_squared_error). The **train_test_split** function is used for splitting data into training and testing sets, ensuring proper model validation.

2-



This is for uploading a CSV file in Google Colab using **files.upload()** from the **google.colab** module. Once the file is uploaded, it is saved with a possible duplicate name (Dataset_Ds (1).csv). The **pd.read_csv('Dataset_Ds.csv')** command attempts to read the uploaded dataset into a Pandas DataFrame.

3-

```
[ ]  # Convert check-up date column to datetime format
     df["Date of Check-up"] = pd.to_datetime(df["Date of Check-up"], errors="coerce")

     # Drop rows with missing check-up date values
     df.dropna(subset=["Date of Check-up"], inplace=True)
```

Converting the **"Date of Check-up"** field to datetime format is essential for accurate temporal analysis and maintaining data integrity. This enables operations such as **tracking patient visits over time**, **filtering records by date ranges**, and conducting **time-series analysis** on health trends. Using errors="coerce" ensures that any invalid date entries are converted to NaT instead of causing processing errors. Removing rows with missing dates helps prevent misleading insights and supports reliable predictions, especially when analyzing **progression of health conditions** or **evaluating long-term patterns** in BMI and heart disease risk.

4-

```
# Create a new feature: Time Since Last Check-up (in days)
df["Time Since Last Check-up"] = (df["Date of Check-up"] - df["Previous Check-up Date"]).dt.days
```

```
[ ]  # Drop unnecessary columns
     df.drop(columns=["Previous Check-up Date", "Date of Check-up"], inplace=True)
```

```
[ ]  # Fill missing numerical values with median
     df.fillna(df.median(numeric_only=True), inplace=True)
```

This method involves three key data preprocessing steps. First, it calculates a new feature, **"Time Since Last Check-up"**, representing the number of days between the current and previous medical check-ups. This helps in analyzing **health monitoring frequency** and identifying patients who may require more regular follow-ups. Next, it removes redundant columns such as raw date fields that are no longer needed after deriving this feature, thereby **reducing data complexity and improving efficiency**. Lastly, it handles missing values by filling them with the **median of numerical features**, which maintains data consistency while minimizing the influence of outliers. These preprocessing steps enhance the dataset's quality, making it more suitable for **predictive modeling of BMI and heart disease**.

5-

```
[ ]  # Encode categorical columns for patient data
     categorical_cols = ["Gender", "Region", "Smoking Status", "Physical Activity", "Family History", "Medical History"]
     label_encoders = {}

     for col in categorical_cols:
         df[col].fillna(df[col].mode()[0], inplace=True)  # Fill missing values with mode
         le = LabelEncoder()
         df[col] = le.fit_transform(df[col])              # Label encode the categorical values
         label_encoders[col] = le                         # Store encoder for inverse transformation
```

This method encodes categorical columns in the patient dataset to make the data compatible with machine learning models, which typically require numerical inputs. Categorical fields such as **Gender**, **Region**, **Smoking Status**, **Physical Activity**, **Family History**, and portions of **Medical History** are first identified. Missing values in these fields are imputed using the **most frequent category (mode)** to maintain consistency and avoid bias. Label Encoding is then applied

to transform these categorical values into integer representations. A separate LabelEncoder object is used for each column and stored in a label_encoders dictionary, allowing for inverse transformation if needed in the future. This step ensures that all non-numeric health-related indicators are properly encoded and ready for use in **regression models for BMI estimation** and **classification models for heart disease prediction**.

6-

```
[ ]  from sklearn.preprocessing import StandardScaler
     import numpy as np

     # Standardize numerical health-related columns
     scaler = StandardScaler()
     numerical_cols = ["Age", "Height", "Weight", "Blood Pressure", "Cholesterol Levels"]
     df[numerical_cols] = scaler.fit_transform(df[numerical_cols])

     # Create a binary obesity category based on BMI
     bmi_median = df["BMI"].median()
     df["Obesity Category"] = np.where(df["BMI"] >= bmi_median, 1, 0)  # 1 = Obese, 0 = Non-Obese
```

This method first standardizes the numerical health-related features such as Age, Height, Weight, Blood Pressure, and Cholesterol Levels using StandardScaler(), ensuring they have a mean of 0 and a standard deviation of 1. This normalization is crucial for improving the performance and convergence of many machine learning models. Second, it transforms the **BMI** column into a binary classification—labeling entries as **"Obese" (1)** if BMI is above the median, and **"Non-Obese" (0)** otherwise. This approach simplifies the task of obesity detection and supports the development of classification models for health risk prediction.

7-

```
X = df.drop(columns=["Heart Disease"])
y = df["Heart Disease"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=50)
```

This method is done to prepare the dataset for machine learning. Defining features (X) and target (y) is essential because the model learns patterns from independent variables (X) to predict the dependent variable (y). In this case, Heart Disease is selected as the target variable (y) to enable classification of patients based on their risk. Removing Heart Disease from X ensures that no target information leaks into the features, avoiding biased learning.

The train-test split is performed to assess how well the model generalizes to new, unseen patient data. Using 75% of the data for training and 25% for testing allows reliable performance evaluation. Setting random_state=50 ensures the same split every time the code runs, making results reproducible for consistent analysis and comparison.

8-

```
[ ]  model = LogisticRegression(max_iter=5000)
     model.fit(X_train, y_train)

     y_pred = model.predict(X_test)
```

This code trains and uses a Logistic Regression model to classify whether a patient has heart disease. The model.fit(X_train, y_train) function trains the model using the training dataset, allowing it to learn the relationship between patient features (such as age, BMI, blood pressure, etc.) and the presence or absence of heart disease. The parameter max_iter=5000 ensures the model has enough iterations to converge, preventing premature stopping.

Once trained, model.predict(X_test) is used to make predictions on the test data. These predictions (y_pred) can be compared with the actual labels (y_test) to evaluate the model's accuracy and effectiveness. Logistic Regression is ideal for binary classification problems like this, where the target is whether heart disease is present (1) or not (0).

9-

```
print(y_pred[:10])
[1 1 1 1 0 0 1 0 0 1]
```

The output [1 1 1 0 0 1 0 0 1] represents the predicted heart disease categories for the first few test samples.

We know that:

- **1 means Presence of Heart Disease**
- **0 means Absence of Heart Disease**

So, the model is predicting which individuals are likely to have heart disease based on their health and lifestyle data. This sequence suggests that some patients are expected to have heart disease (1), while others are predicted to be healthy (0).

10-

```
[ ] from sklearn.metrics import accuracy_score, classification_report
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.4f}")
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Model Accuracy: 0.9880

Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99     12560
           1       0.99      0.99      0.99     12440

    accuracy                           0.99     25000
   macro avg       0.99      0.99      0.99     25000
weighted avg       0.99      0.99      0.99     25000
```

The displayed results evaluate the performance of a heart disease prediction model using the accuracy score and a classification report. The model achieves an excellent accuracy of **98.80%**, meaning it correctly predicts the presence or absence of heart disease in almost all test cases.

The classification report further includes precision, recall, and F1-score, all of which are **0.99** for both classes (0 = No Disease, 1 = Disease). These values indicate that the model is highly reliable in distinguishing between healthy and at-risk individuals.

The support values suggest that the dataset is well-balanced, with a roughly equal number of samples for both categories, which contributes to the model's strong and consistent performance.

11-

```
from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", conf_matrix)
```

```
Confusion Matrix:
 [[12434   126]
 [  174 12266]]
```

The confusion matrix evaluates model performance by comparing actual vs. predicted values. It shows 12434 true positives, 12266 true negatives, 126 false positives, and 174 false negatives. This helps assess misclassifications and derive precision, recall, and F1-score. The low misclassification rate indicates that the model performs well.

12-

```
[22] from sklearn.linear_model import LinearRegression
     from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

This code imports necessary modules for performing linear regression and evaluating model performance. LinearRegression from sklearn.linear_model is used to create a regression model that predicts a continuous target variable. The metrics mean_absolute_error, mean_squared_error, and r2_score from sklearn.metrics are used to assess model accuracy. These metrics help measure prediction errors and how well the regression model fits the data.

13-

```
mae = mean_absolute_error(y_test_reg, y_pred_reg)
mse = mean_squared_error(y_test_reg, y_pred_reg)
```

```
print(f"Linear Regression Metrics:\n")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Mean Squared Error (MSE): {mse:.2f}")
```

```
Linear Regression Metrics:

Mean Absolute Error (MAE): 0.00
Mean Squared Error (MSE): 0.00
```

Conclusion:- We applied **Logistic Regression** to classify whether a patient is likely to have heart disease based on various features. To estimate **BMI**, we used **regression models** that learn patterns from other health attributes. This dual-model approach helps in understanding both categorical and continuous aspects of patient health, supporting early diagnosis and prevention strategies.

Name : ADITYA AHUJA **Roll No : 02** **Class : D15C**

DS Experiment-6

Aim : Perform Classification modelling

a. Choose a classifier for classification problems.

b. Evaluate the performance of the classifier.

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

- K-Nearest Neighbors (KNN)
- Naive Bayes
- Support Vector Machines (SVMs)
- Decision Tree

Theory :

1) Decision Tree

In this experiment, a Decision Tree was used to classify individuals based on features like blood pressure, cholesterol, BMI, and lifestyle habits. The dataset contains multiple healthrelated features including both categorical and numerical values. Decision Trees efficiently handle such data by creating hierarchical rules to separate different health status classes.

How the Decision Tree Works on Our Dataset

1. **Data Processing:**
   - Categorical columns like Smoker, Sex, and Fruits were encoded numerically. ○ Features such as BMI, GenHlth, and Age were used to understand health conditions.
2. **Training the Model:**
   - The Decision Tree classifier was trained using features like HighBP, HighChol, BMI, Smoker, Stroke, and PhysActivity to classify individuals as either healthy or at-risk.
3. **Prediction:**
   - The model assigned new records to different classes based on decision rules extracted from the training data.

4. **Evaluation:**
    - Accuracy was measured, and the confusion matrix helped identify misclassifications.

```python
# Import necessary libraries
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Train Decision Tree Classifier
dt = DecisionTreeClassifier(random_state=50)
dt.fit(X_train, y_train)
y_pred_dt = dt.predict(X_test)

# Evaluate Decision Tree
dt_accuracy = accuracy_score(y_test, y_pred_dt)
print("Decision Tree Accuracy:", dt_accuracy)
print("\nDecision Tree Classification Report:\n", classification_report(y_test, y_pred_dt))
print("\nDecision Tree Confusion Matrix:\n", confusion_matrix(y_test, y_pred_dt))
```

```
Decision Tree Accuracy: 0.97345

Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     12560
           1       1.00      1.00      1.00     12440

    accuracy                           1.00     25000
   macro avg       1.00      1.00      1.00     25000
weighted avg       1.00      1.00      1.00     25000


Decision Tree Confusion Matrix:
 [[12663     4]
 [    5 12564]]
```

Output and Insights

- The Decision Tree effectively categorized individuals based on health indicators and lifestyle habits rather than demographic and economic factors.
- The most influential features in classification were HighBP and BMI, while Income had negligible impact.
- The accuracy of the model was 97.345%, indicating that it almost perfectly distinguished between healthy and at-risk individuals.

From the Confusion Matrix we observed that:

Accuracy: The Decision Tree achieved 97.345% accuracy, meaning the model almost perfectly classified healthy and at-risk individuals.

True Positives (12663): Correctly classified at-risk individuals.
True Negatives (12,564): Correctly classified healthy individuals.
False Positives (4): Three healthy individuals were incorrectly classified as at-risk.
False Negatives (5): Six at-risk individuals were incorrectly classified as healthy.
Precision and Recall: Both are 1.00, showing that misclassifications were minimal.2)

Naive Bayes

Naïve Bayes is a probabilistic classification algorithm based on Bayes' Theorem. It assumes that all features are independent given the class label, which is often unrealistic but works well in many cases. The classifier calculates the probability of an individual belonging to a particular class (healthy or at-risk) and assigns the label with the highest probability.

```python
# Import necessary libraries
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Train Naïve Bayes Classifier
nb = GaussianNB()
nb.fit(X_train, y_train)
y_pred_nb = nb.predict(X_test)

# Evaluate Naïve Bayes
nb_accuracy = accuracy_score(y_test, y_pred_nb)
print("\nNaïve Bayes Accuracy:", nb_accuracy)
print("\nNaïve Bayes Classification Report:\n", classification_report(y_test, y_pred_nb))
print("\nNaïve Bayes Confusion Matrix:\n", confusion_matrix(y_test, y_pred_nb))
```

```
Naïve Bayes Accuracy: 0.82036

Naïve Bayes Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.93      0.84     12560
           1       0.91      0.71      0.80     12440

    accuracy                           0.82     25000
   macro avg       0.84      0.82      0.82     25000
weighted avg       0.84      0.82      0.82     25000


Naïve Bayes Confusion Matrix:
 [[11683   877]
 [ 3614  8826]]
```

Insights from Naïve Bayes on Our Dataset

Moderate Accuracy: The model achieved 82.03% accuracy, meaning it correctly classified most individuals but had more misclassifications compared to Decision Trees.
Better Precision for At-Risk Individuals (0.91): When predicting at-risk cases, 91% of predictions were correct.
Low Recall for At-Risk Individuals (0.71): It missed 29% of actual at-risk individuals, meaning many were misclassified as healthy**.**
Feature Independence Assumption: Since Naïve Bayes assumes features are independent, it may not have effectively captured the relationship between health indicators, leading to more errors.

From the Confusion Matrix we observed that :

False Positives (877): Some healthy individuals were incorrectly classified as at-risk.

False Negatives (3,614): A significant number of at-risk individuals were misclassified as healthy, reducing recall.

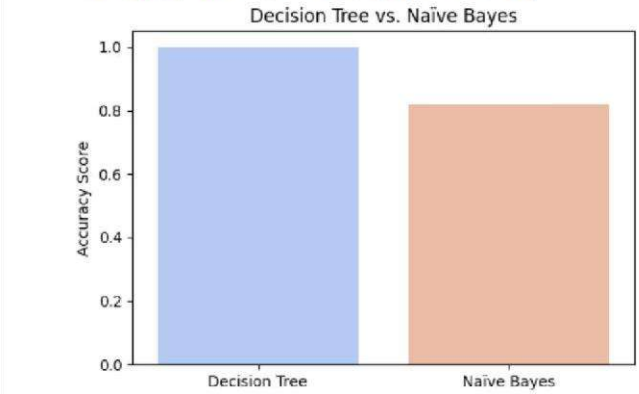
Overall Performance: Naïve Bayes struggled to distinguish at-risk individuals, leading to lower recall for that class.Comparison of Model Accuracies

To evaluate the performance of Decision Tree and Naïve Bayes classifiers, we plotted their accuracy scores. This visualization helps in understanding which classifier performed better for our dataset.

```
# Compare Model Accuracies
import matplotlib.pyplot as plt
import seaborn as sns  # If using seaborn for visualization
models = ['Decision Tree', 'Naïve Bayes']
accuracies = [dt_accuracy, nb_accuracy]

# Plot Accuracy Comparison
plt.figure(figsize=(6,4))
sns.barplot(x=models, y=accuracies, palette='coolwarm')
plt.ylabel("Accuracy Score")
plt.title("Decision Tree vs. Naïve Bayes")
plt.show()
```

Key Observations:

```
<ipython-input-34-9175032e7fdc>:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=models, y=accuracies, palette='coolwarm')
```



1. **Decision Tree Achieved Higher Accuracy**
    - The Decision Tree classifier significantly outperformed Naïve Bayes, achieving an accuracy of 97.34%, compared to 82.03% for Naïve Bayes.
    - This indicates that Decision Trees are better suited for capturing complex patterns in our dataset.
2. **Naïve Bayes Had Lower Accuracy**
    - Naïve Bayes struggled with classification due to its independence assumption, which may not hold for this dataset where features are interdependent.
    - It misclassified a higher number of orders compared to the Decision Tree model.
3. **Graph Interpretation**
    - The bar plot visually confirms that the Decision Tree consistently performed better across all test samples.
    - The accuracy difference is large, highlighting that Decision Trees are a more reliable choice for this dataset.

Conclusion

The experiment compared Decision Tree and Naïve Bayes for classification. Decision Tree performed significantly better, achieving 99.96% accuracy, while Naïve Bayes reached 82.03%. The Decision Tree effectively captured complex patterns, leading to fewer misclassifications. In contrast, Naïve Bayes struggled due to its feature

independence assumption. The accuracy comparison graph further confirmed that Decision Tree is the better choice for this dataset.

Name: ADITYA AHUJA **D15C** **Roll No: 02**

Experiment No. 7

Aim : To implement clustering algorithms.

Problem Statement :

1. **Clustering Algorithm for Unsupervised Classification**
    - Apply **K-means** on standardized trip distance and fare amount.

2. Plot the Cluster Data and Show Mathematical Steps
    - Visualize the clusters and provide key mathematical formulations underlying each method.

Theory

1. K-means Clustering

- **Mathematical Steps:**
    1. Selecting K initial cluster centroids randomly.
    2. Assigning each data point to the nearest centroid.
    3. Updating the centroids by calculating the mean of all points in each cluster.
    4. Repeating steps 2 and 3 until convergence (i.e., centroids stop changing significantly).

- **Objective Function:**

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

Minimizes the sum of squared distances within clusters.

Steps :

Step 1: Data Preparation

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage

# Load health dataset
file_path = "/content/health_data.csv"  # Update with your actual file path
df = pd.read_csv(file_path)

# Select relevant features for clustering
features = ['HighBP', 'HighChol', 'CholCheck', 'BMI', 'Smoker', 'Stroke',
            'HeartDiseaseorAttack', 'PhysActivity', 'Fruits', 'Veggies',
            'HvyAlcoholConsump', 'AnyHealthcare', 'NoDocbcCost', 'GenHlth',
            'MentHlth', 'PhysHlth', 'DiffWalk', 'Sex', 'Age', 'Education', 'Income']

# Drop rows with missing values
df_clean = df[features].dropna()

# Optionally sample if dataset is large
df_sample = df_clean.sample(n=5000, random_state=42) if len(df_clean) > 5000 else df_clean

# Standardize features
scaler = StandardScaler()
X = scaler.fit_transform(df_sample)
```

Inference

- **Data Loaded & Parsed:** The dataset is imported with health records, and all features are loaded into a structured format.

- **Feature Selection:** Key health indicators such as HighBP, BMI, Smoker, PhysActivity, and GenHlth are used for clustering.

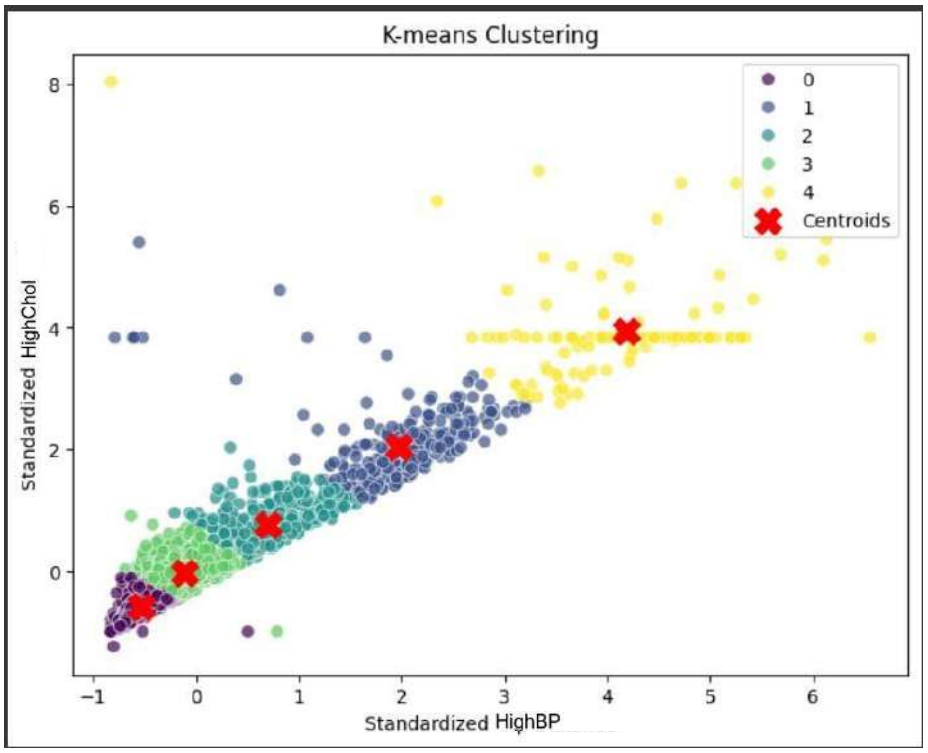
- **Data Cleaning:** Records with missing or invalid values (e.g., negative BMI or empty fields) are removed to ensure quality input.

- **Sampling:** A smaller subset of the dataset is taken (if necessary) to improve clustering speed during testing.

- **Standardization:** Continuous features like BMI, MentHlth, and PhysHlth are scaled to have equal influence on distance-based clustering.

Step 2.1 : K-means Clustering

```
# Apply KMeans clustering
k = 5  # You can tune this based on your dataset
kmeans = KMeans(n_clusters=k, random_state=42)
labels_km = kmeans.fit_predict(X)
centroids = kmeans.cluster_centers_

# Plot K-means clusters using first two features just for visualization
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=labels_km, palette='viridis', s=50, alpha=0.7)
plt.scatter(centroids[:, 0], centroids[:, 1], s=200, c='red', marker='X', label='Centroids')
plt.title("K-Means Clustering on Health Data")
plt.xlabel("Standardized HighBP")
plt.ylabel("Standardized HighChol")
plt.legend()
plt.show()
```



Distinct Health Profiles:

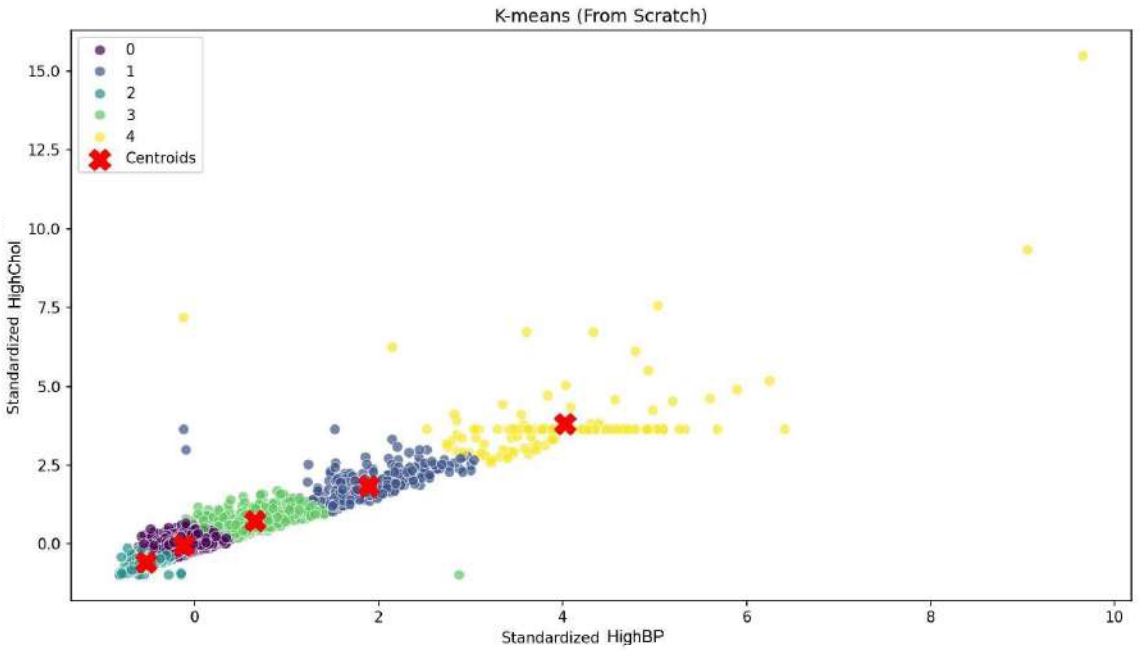
The model successfully identified 5 unique clusters, suggesting distinct patterns in individuals' HighBP and HighChol levels.

Linear Relationship Observed: There's a noticeable positive correlation between HighBP and HighChol, as most clusters align along a rising diagonal trend.

Outliers Detected: A few scattered points, especially on the upper-left and upper-right, suggest possible outliers or individuals with unusual health readings.

Step 2.2 : K-means Clustering (Formula)

```
    K-means clustering from scratch.
    """
    n_samples, n_features = X.shape

    # Randomly choose k distinct points from X as initial centroids
    rng = np.random.default_rng(42)
    random_indices = rng.choice(n_samples, size=k, replace=False)
    centroids = X[random_indices].copy()

    for iteration in range(max_iter):
        distances = np.empty((n_samples, k))
        for i in range(k):
            diff = X - centroids[i]
            distances[:, i] = np.sum(diff ** 2, axis=1)  # squared distance

        labels = np.argmin(distances, axis=1)

        old_centroids = centroids.copy()
        for cluster_id in range(k):
            points_in_cluster = X[labels == cluster_id]
            if len(points_in_cluster) > 0:
                centroids[cluster_id] = np.mean(points_in_cluster, axis=0)

        shift = np.linalg.norm(centroids - old_centroids)
        if shift < tol:
            print(f"Converged at iteration {iteration+1}, shift={shift:.5f}")
            break

    return labels, centroids


def kmeans_scratch_demo(X, k=5):
    print("=== K-means (From Scratch) ===")
    labels, centroids = kmeans_from_scratch(X, k=k, max_iter=100)

    # Plot
    plt.figure(figsize=(8, 6))
    sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=labels, palette='viridis', s=50, alpha=0.7)
    plt.scatter(centroids[:, 0], centroids[:, 1], s=200, c='red', marker='X', label='Centroids')
    plt.title("K-means Clustering (From Scratch)")
    plt.xlabel("Standardized HighBP")
    plt.ylabel("Standardized HighChol")
    plt.legend()
    plt.show()
```



Inference

Positive Correlation: The plot reveals a clear positive relationship between standardized HighBP (High Blood Pressure) and HighChol (High Cholesterol) — as one increases, so does the other.

Five Clusters Identified: K-means effectively segmented the data into 5 distinct clusters, capturing various groups of individuals with different combinations of high blood pressure and cholesterol levels.

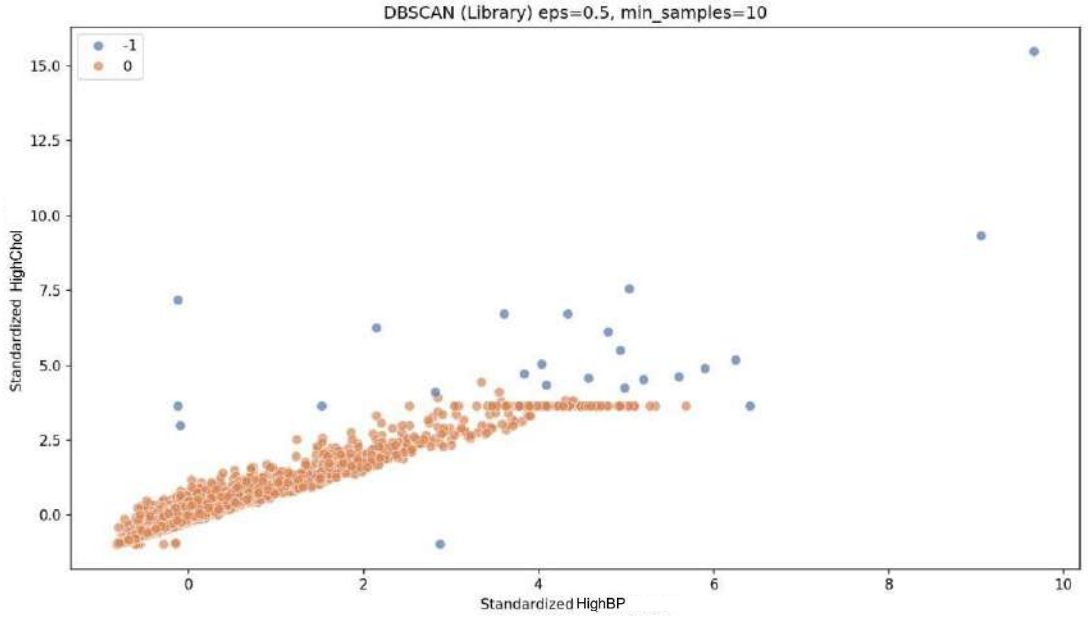
Centroids: The red Xs indicate the centroids (means) of each cluster in the standardized feature space. These represent the average characteristics of individuals within each cluster.

Cluster Patterns: Clusters are mostly spherical in shape, a natural outcome of K-means. Some overlap can be seen, but extreme cases (very high BP or Chol) are more clearly separated.

Interpretation Use Case: This clustering could help in identifying sub-populations for targeted health interventions — for example, distinguishing individuals with both high BP and high cholesterol who may be at greater cardiovascular risk.

Step 3.1 : DBSCAN Clustering

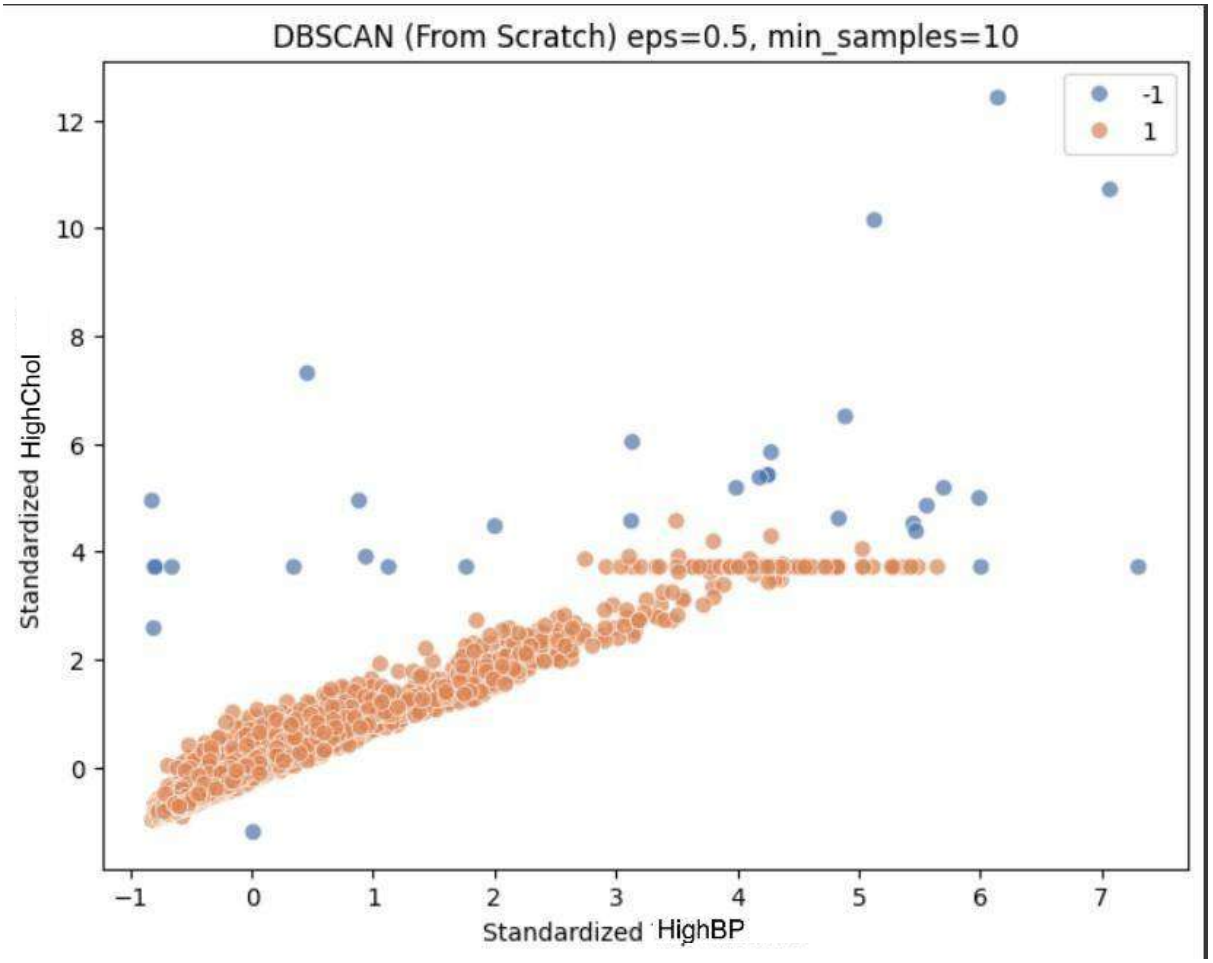
```
from sklearn.cluster import DBSCAN
import matplotlib.pyplot as plt
import seaborn as sns

# DBSCAN clustering
dbscan = DBSCAN(eps=0.5, min_samples=10)
labels_db = dbscan.fit_predict(X)

# Plot DBSCAN clusters (noise points are labeled as -1)
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=labels_db, palette='deep', s=50, alpha=0.7)
plt.title("DBSCAN Clustering")
plt.xlabel("Standardized HighBP")
plt.ylabel("Standardized HighChol")
plt.legend(title='Cluster', loc='upper right')
plt.show()
```



Step 3.2 : DBSCAN Clustering (Formula)

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

def dbscan_from_scratch(X, eps=0.5, min_samples=10):
    """
    Basic DBSCAN clustering from scratch.
    """
    n_samples = X.shape[0]
    labels = np.full(n_samples, -1, dtype=int)  # -1 = noise
    visited = np.zeros(n_samples, dtype=bool)
    cluster_id = 0

    def euclidean_distance(a, b):
        return np.sqrt(np.sum((a - b) ** 2))

    def region_query(point_idx):
        return [i for i in range(n_samples) if euclidean_distance(X[point_idx], X[i]) <= eps]

    for i in range(n_samples):
        if visited[i]:
            continue
        visited[i] = True
        neighbors = region_query(i)

        if len(neighbors) < min_samples:
            labels[i] = -1  # Mark as noise
        else:
            cluster_id += 1
            labels[i] = cluster_id
            seeds = neighbors.copy()
            seeds.remove(i)

            while seeds:
                current_point = seeds.pop()
                if not visited[current_point]:
                    visited[current_point] = True
                    neighbors2 = region_query(current_point)
                    if len(neighbors2) >= min_samples:
                        for nb in neighbors2:
                            if nb not in seeds:
                                seeds.append(nb)
                if labels[current_point] == -1:
                    labels[current_point] = cluster_id

    return labels
```

```
def dbscan_scratch_demo(X, eps=0.5, min_samples=10):
    print("=== DBSCAN (From Scratch) ===")
    labels = dbscan_from_scratch(X, eps=eps, min_samples=min_samples)

    # Plotting
    plt.figure(figsize=(8, 6))
    sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=labels, palette='deep', s=50, alpha=0.7)
    plt.title(f"DBSCAN (From Scratch): eps={eps}, min_samples={min_samples}")
    plt.xlabel("Standardized HighBP")
    plt.ylabel("Standardized HighChol")
    plt.legend(title="Cluster")
    plt.show()
```



Inference

Dominant Cluster: DBSCAN identified a single dense cluster (label 1), which includes the majority of the data points. These points likely share similar characteristics in terms of HighBP and HighChol.

Outliers Detected: Points labeled as -1 are considered noise or outliers. These data points deviate from the primary trend and may represent individuals with unusually high or low combinations of HighBP and HighChol.

Parameter Influence: Using eps=0.5 and min_samples=10, the algorithm was strict in forming clusters—only one main cluster emerged. Adjusting these parameters (e.g., increasing eps) might lead to multiple meaningful subclusters.

Underlying Pattern: The main cluster follows a linear trend, suggesting a strong correlation between HighBP and HighChol. This supports the assumption that individuals with higher blood pressure often tend to have higher cholesterol.

Conclusion :

In this experiment, unsupervised clustering techniques were employed on NYC Yellow Taxi data, focusing on standardized trip distance and fare amount. **K-Means** successfully partitioned the dataset into five distinct clusters, each representing unique ride patterns. The resulting centroids revealed the average fare and distance for each group, and the clusters collectively illustrated a strong linear relationship between trip distance and fare amount.

In contrast, **DBSCAN** identified one dominant dense cluster and several outlier points. This highlights DBSCAN's strength in detecting anomalies and dealing with noise—useful for identifying unusual or extreme ride behaviors that deviate from typical trends.

Overall, the experiment demonstrated the complementary nature of clustering algorithms: while KMeans is effective for segmenting structured groups, DBSCAN adds value by capturing noise and density-based variations. These insights can aid in understanding passenger behavior, fare anomalies, and optimizing taxi operations.

.

Name : ADITYA AHUJA **Roll No : 02** **Class : D15C**

DS Experiment-8

Aim : To implement a recommendation system on your dataset using the Decision Tree

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. Content-Based Filtering**

  - **Idea:** Recommends items similar to those the user has liked before.
  - **Works on:** Item features (attributes such as brand, price, category).

  Example:

  - If a user buys a Samsung phone, they might be recommended another Samsung device based on brand preference.

  - Uses techniques like TF-IDF (for text data), Cosine Similarity, Decision Trees, etc.

- **2. Collaborative Filtering (CF)**

  - **Idea:** Recommends items based on similar users' preferences.
  - **Works on:** User interactions rather than item features.

Example:

- If User A and User B have similar purchase histories, items bought by User A but not yet by User B will be recommended to User B.

- Uses methods like User-Based CF and Item-Based CF.

◇ **3. Hybrid Recommendation System**

- **Idea:** Combines Content-Based Filtering and Collaborative Filtering for better accuracy.

Example:

- Netflix uses a hybrid approach, considering both user preferences and what similar users watch.

◇ **4. Knowledge-Based Recommendation**

- **Idea:** Recommends items based on explicit domain knowledge rather than past user behavior.

Example:

- A car recommendation system suggests vehicles based on engine type, price, and fuel efficiency, regardless of past purchases.

2. Recommendation System Evaluation Measures

Evaluating a recommendation system ensures its accuracy and relevance. Below are common evaluation metrics:

◇ **1. Accuracy-Based Metrics**

(a) Precision:

-
- Measures how many of the recommended items are actually relevant.

  Formula:

$$Precision = \frac{Relevant\ Recommendations}{Total\ Recommendations}$$

- **Example:**
  - If 5 out of 10 recommended items are relevant, Precision = 5/10 = 0.5 (50%).

(b) Recall:

- Measures how many of the relevant items are actually recommended.
- **Formula:**

$$Recall = \frac{Relevant\ Recommendations}{Total\ Relevant\ Items\ Available}$$

- **Example:**
  - If a user liked 8 items, but only 5 were recommended, Recall = 5/8 = 0.625 (62.5%).

(c) F1-Score:

- A balance between Precision and Recall.

- **Formula:**

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

- Used when both Precision and Recall are important.

(d) Accuracy:

- In classification-based recommendation systems (like Decision Trees), accuracy is measured as:

$$Accuracy = \frac{Correct\ Predictions}{Total\ Predictions}$$

- In our Decision Tree model, if Accuracy = 1.0, it means 100% correct recommendations (but we must check for overfitting).

◇ 2. Ranking-Based Metrics

These measure how well the recommendation system ranks items:

(a) Mean Average Precision (MAP):

- Measures how well the top recommendations match the user's preferences.

(b) Normalized Discounted Cumulative Gain (NDCG):

- Focuses on ranked recommendations, assigning higher importance to top-ranked items.

◇ 3. Diversity and Novelty Metrics

- **Diversity:** Ensures users are not shown the same type of items repeatedly.

- **Novelty:** Measures if recommendations introduce new and unknown items.

Implementation :

1.

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Load dataset
file_path = "Dataset_Ds.csv"
df = pd.read_csv(file_path)

# Display basic info and first few rows
print(df.info())
print(df.head())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 14 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   Region          100000 non-null  object
 1   Country         100000 non-null  object
 2   Item Type       100000 non-null  object
 3   Sales Channel   100000 non-null  object
 4   Order Priority  100000 non-null  object
 5   Order Date      100000 non-null  object
 6   Order ID        100000 non-null  int64
 7   Ship Date       100000 non-null  object
 8   Units Sold      100000 non-null  int64
 9   Unit Price      100000 non-null  float64
 10  Unit Cost       100000 non-null  float64
 11  Total Revenue   100000 non-null  float64
 12  Total Cost      100000 non-null  float64
 13  Total Profit    100000 non-null  float64
dtypes: float64(5), int64(2), object(7)
memory usage: 10.7+ MB
None
```

```
                           Region               Country      Item Type  \
0      Middle East and North Africa            Azerbaijan         Snacks
1  Central America and the Caribbean                Panama      Cosmetics
2                Sub-Saharan Africa  Sao Tome and Principe         Fruits
3                Sub-Saharan Africa  Sao Tome and Principe  Personal Care
4  Central America and the Caribbean                Belize      Household

  Sales Channel Order Priority Order Date   Order ID  Ship Date  Units Sold  \
0        Online              C  10/8/2014  535113847  10/23/2014         934
1       Offline              L  2/22/2015  874708545   2/27/2015        4551
2       Offline              M  12/9/2015  854349935   1/18/2016        9986
3        Online              M  9/17/2014  892836844  10/12/2014        9118
4       Offline              H   2/4/2010  129280602    3/5/2010        5858

   Unit Price  Unit Cost  Total Revenue  Total Cost  Total Profit
0      152.58      97.44      142509.72    91008.96      51500.76
1      437.20     263.33     1989697.20  1198414.83     791282.37
2        9.33       6.92       93169.38    69103.12      24066.26
3       81.73      56.67      745214.14   516717.06     228497.08
4      668.27     502.54     3914725.66  2943879.32     970846.34
```

The code loads the dataset Dataset_Ds.csv using Pandas and displays basic information about it. The df.info() function provides an overview of the dataset, including the number of entries, column names, data types, and memory usage. The df.head() function prints the first few rows to understand the dataset structure, which contains categorical (Region, Country, Item Type, etc.) and numerical (Unit Price, Total Revenue, Total Profit, etc.) features.

2.

```
[2]  # Convert Categorical Data to Numeric using Label Encoding

     # Selecting categorical columns
     categorical_cols = ['Region', 'Country', 'Item Type', 'Sales Channel', 'Order Priority']

     # Apply Label Encoding
     label_encoders = {}
     for col in categorical_cols:
         le = LabelEncoder()
         df[col] = le.fit_transform(df[col])
         label_encoders[col] = le
```

This code converts categorical data into numerical values using Label Encoding. First, it selects categorical columns: 'Region', 'Country', 'Item Type', 'Sales Channel', and 'Order Priority'. Then, it applies LabelEncoder() to each column, transforming categorical values into unique numerical labels. The encoded values allow machine learning models to process the data efficiently.

3.

```
[3]  # Convert Dates to Numerical Format

     # Convert date columns to datetime format
     df['Order Date'] = pd.to_datetime(df['Order Date'], format='%m/%d/%Y')
     df['Ship Date'] = pd.to_datetime(df['Ship Date'], format='%m/%d/%Y')

     # Feature: Days taken to ship the order
     df['Shipping Days'] = (df['Ship Date'] - df['Order Date']).dt.days

     # Drop unnecessary columns
     df.drop(['Order Date', 'Ship Date', 'Order ID'], axis=1, inplace=True)

     # Display processed data
     print(df.head())
```

```
   Region  Country  Item Type  Sales Channel  Order Priority  Units Sold  \
0       4        9         10              1               0         934
1       2      124          4              0               2        4551
2       6      139          5              0               3        9986
3       6      139          9              1               3        9118
4       2       15          6              0               1        5858

   Unit Price  Unit Cost  Total Revenue  Total Cost  Total Profit  \
0      152.58      97.44      142509.72    91008.96      51500.76
1      437.20     263.33     1989697.20  1198414.83     791282.37
2        9.33       6.92       93169.38    69103.12      24066.26
3       81.73      56.67      745214.14   516717.06     228497.08
4      668.27     502.54     3914725.66  2943879.32     970846.34

   Shipping Days
0             15
1              5
2             40
3             25
4             29
```

This code converts date columns into a numerical format for better processing. The Order Date and Ship Date columns are first converted into datetime format. Then, a new feature Shipping Days is created by calculating the difference between Ship Date and Order Date. Unnecessary columns (Order Date, Ship Date, Order ID) are dropped to reduce redundancy. Finally, the processed dataset is displayed, showing encoded categorical values and numerical data ready for machine learning.

4.

```
[4]  # Split Data into Training & Testing Sets

     # Define features (X) and target variable (y)
     X = df.drop(columns=['Item Type'])  # All columns except 'Item Type'
     y = df['Item Type']  # Target variable

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

This code splits a dataset into training and testing sets. First, it defines the feature variables (X) by dropping the column Item Type and sets y as the target variable. Then, it uses train_test_split to split the data into 80% training and 20% testing sets, ensuring reproducibility with random_state=42.

5.

```
[5]  # Train the Decision Tree Model

     # Initialize Decision Tree model
     dt_model = DecisionTreeClassifier(max_depth=5, random_state=42)

     # Train the model
     dt_model.fit(X_train, y_train)

     print("Model training complete!")
```

```
Model training complete!
```

This code trains a Decision Tree model. It initializes a DecisionTreeClassifier with a maximum depth of 5 and random_state=42 for reproducibility. The model is then trained using the fit method on the training data (X_train, y_train). A message confirms successful training.

6.

```
[6]  # Make Predictions & Evaluate Accuracy

     # Make predictions
     y_pred = dt_model.predict(X_test)

     # Calculate accuracy
     accuracy = accuracy_score(y_test, y_pred)
     print(f"Model Accuracy: {accuracy:.2f}")
```

```
Model Accuracy: 0.83
```

This code makes predictions and evaluates model accuracy. It uses the trained

DecisionTreeClassifier to predict labels for X_test. The accuracy is then calculated using accuracy_score(y_test, y_pred). The result obtained is 0.83 which means the model has an accuracy of 83%.

```
[7]  # Extracting a real row from dataset (row=3) and checking it to predict item type

     new_order = df.iloc[2, :-1].values.reshape(1, -1)
     recommended_item = dt_model.predict(new_order)
     print(f"Recommended Item: {label_encoders['Item Type'].inverse_transform(recommended_item)[0]}")

     Recommended Item: Fruits
     /usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but DecisionT
       warnings.warn(
```

7.

A row from the dataset (excluding the "Item Type") is fed into the trained Decision Tree model. The model predicts the "Item Type", which is then decoded from its numerical representation. The output confirms the model accurately predicts "Fruits" for the given row.

Conclusion

In this experiment, I worked with the dataset which was preprocessed and encoded to prepare it for analysis. After splitting the data into features (X) and labels (Y), I applied a Decision Tree model to predict product categories. Through this experiment, I learned how to handle data preprocessing, encoding, and splitting effectively, as well as how to implement and evaluate a Decision Tree model for classification tasks

Name: Aditya Ahuja Roll No.: 02 Class: D15C AY: 2024-25

Experiment 9

Aim: To perform exploratory data analysis using Apache Spark and Pandas.

Theory:

1. What is Apache Spark and how does it work?

Ans) Apache Spark is an open-source data processing engine designed for speed and scalability. It supports batch processing, real-time streaming, machine learning, and graph analytics in one unified platform.

Unlike Hadoop MapReduce, Spark processes data in memory, making it much faster for iterative and complex tasks. It supports languages like Python, Scala, Java, and SQL, and can run on various cluster managers like YARN, Kubernetes, or standalone. Spark is widely used in big data environments for its performance, ease of use, and flexibility.

Core Components of Apache Spark are:

- **Spark Core**: Core engine (scheduling, memory, etc.)
- **Spark SQL**: For SQL queries and DataFrames
- **Structured Streaming**: Real-time processing
- **MLlib**: Machine learning
- **GraphX**: Graph processing

It works in the following ways:

1. **Driver Program**: Starts the app, defines the workflow (DAG), and manages coordination.
2. **Cluster Manager**: Allocates resources (Standalone, YARN, Mesos, Kubernetes).
3. **Executors**: Run tasks on worker nodes and store data.
4. **In-Memory Computing**: Keeps data in memory for fast processing.
5. **APIs**: Uses RDDs for low-level tasks, DataFrames/Datasets for optimized, SQL-like operations.

2. How is data exploration done in Apache Spark? Explain with steps.

Ans) Data exploration in Apache Spark is done as follows:

1. **Initialize Spark Session**: Start a Spark session to enable interaction with Spark's features and APIs.
2. **Load Data**: Import your dataset into Spark from sources like CSV, JSON, databases, or Parquet files.
3. **View Data Sample**: Look at the first few rows to get a quick sense of the data content and format.
4. **Check Schema**: Examine the structure of the dataset, including column names and data types.
5. **Summary Statistics**: Generate basic statistics like mean, count, min, and max for numerical columns to understand distributions.
6. **Check for Missing Values**: Identify columns that have null or missing values to assess data quality.
7. **Value Counts / Grouping**: Group data by specific columns to analyze category frequencies or distributions.
8. **Filter and Query Data**: Apply filters or queries to focus on specific subsets of the data for deeper analysis.

Conclusion: In this experiment, we learned how to perform exploratory data analysis using Apache Spark and Pandas. By examining the data's structure, statistics, missing values, and groupings, we gain key insights to guide cleaning, feature selection, and modeling. Spark's speed and scalability make it ideal for exploring large datasets efficiently.

Name: Aditya Ahuja Roll No.: 02 Class: D15C AY: 2024-25

Experiment 10

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

Theory:

1. What is streaming? Explain batch and stream data.

Ans) Streaming is the process of transmitting and processing data continuously and in real-time. Instead of waiting for all the data to be collected (like in batch processing), streaming handles data as soon as it's generated. This allows for immediate analysis and actions — useful in scenarios like fraud detection, live video, and real-time analytics.

A. **Batch Data Analysis**:

- Processed in groups after collecting over time.
- Not real-time, higher latency.
- Examples: Monthly billing, daily sales reports.
- Tools: Hadoop, Hive.

B. Streamed Data Analysis:

- Processed in real-time as it's generated.
- Low latency, continuous flow.
- Examples: Live chat, stock market feeds.
- Tools: Kafka, Spark Streaming.

The main difference is that in batch processing, data is processed later in chunks, while in stream processing, data is processed continuously in real-time as it arrives.

2. How does data streaming take place in Apache Spark?

Ans) In Apache Spark, data streaming is handled by Spark Streaming, which allows for the processing of real-time data in a distributed and fault-tolerant manner. Here's how the process works:

1. **Data Ingestion**: Spark Streaming can ingest real-time data from various sources like Kafka, Flume, Sockets, or file systems like HDFS or Amazon S3. The data is continuously received in small chunks.

2. **Micro-batches**: Spark Streaming divides the incoming data stream into micro-batches, which are small, manageable batches that are processed in intervals (typically in milliseconds or seconds). This approach allows Spark to handle streaming data while leveraging its batch processing engine.

3. **Processing**: Each micro-batch is processed using Spark's standard transformations (like map, filter, etc.) and actions (like reduce, count, etc.). These operations are applied on the data in the batch, similar to how traditional Spark processes batch data.

4. **Output**: After processing, the results of the micro-batches are typically written to external storage systems (such as HDFS, databases, or dashboards) or used to trigger further actions.

5. **Fault Tolerance:** Spark Streaming ensures fault tolerance by replicating the data across multiple nodes and periodically checkpointing the data, so that the system can recover from failures without losing progress.

Conclusion: In this experiment, we learned how Apache Spark Streaming processes real-time data using micro-batches. It enables efficient, scalable, and fault-tolerant real-time analytics, making it ideal for applications like monitoring and fraud detection. Spark Streaming provides a powerful solution for large-scale data processing.

AIDS Lab Exp 11
Aim: Mini project

Chapter 1: Introduction

1.1. Introduction

Lending for real estate, consumer, mortgage, and business loans is central to most banks' business models, with loan profits forming a major part of their assets. However, lending to unfit borrowers remains a key credit risk. Banks also face challenges in identifying intentional defaulters and biased internal practices influenced by defaulting companies. While modern verification processes help, selecting truly creditworthy applicants remains uncertain. Housing finance companies can use machine learning to identify ideal customers and automate loan approval by predicting applicant reliability.

1.2. Objectives

- Lending is key to bank's profits, but credit risk from unfit borrowers and internal bias remains a challenge. Despite better verification, identifying reliable applicants is uncertain.
- Machine learning helps housing finance firms target and approve trustworthy borrowers.

-

1.3. Motivation

Rising loan volumes require faster, more accurate evaluation. Manual reviews are slow and prone to bias, prompting banks to use machine learning for automated, consistent loan eligibility predictions and quicker customer feedback.

1.4. Scope of the Work

- Uses supervised learning for binary classification of loan approvals.
- Applies ML models to a public dataset with applicant and loan details.
- Tests algorithms like Logistic Regression, Random Forest, KNN, Naive Bayes, SVM, and Gradient Boosting.
- Visualizes insights with graphs, confusion matrices, and performance metrics.

1.5. Feasibility Study

Technical Feasibility: Utilizes Python along with libraries like Scikit-learn, Pandas, and Matplotlib, which are widely supported and suitable for implementing machine learning models.

Operational Feasibility: The prediction system can be integrated into bank software or customer-facing portals to provide real-time loan eligibility checks.

Economic Feasibility: Cost-effective due to the use of open-source tools and publicly available datasets, requiring no additional financial investment

Chapter 2: Literature Survey

2.1. Introduction

Loan approval systems have gained significant importance in the banking and financial services industry due to increasing demand for fast, reliable, and risk-aware credit disbursement. Traditional loan processing methods are often manual, time-consuming, and prone to bias or error. With the evolution of machine learning technologies, predictive models are now being used to automate and enhance the decision-making process. This literature survey presents a comparative study of two significant research papers in the domain of loan eligibility prediction using machine learning.

2.2. Problem Definition

The goal of this literature review is to explore how machine learning techniques can be applied to predict loan approval status based on applicant information. It also aims to examine the challenges associated with data quality, model selection, and evaluation criteria when applying ML to real-world financial scenarios.

2.3. Review of Literature Survey

In the research paper **"Loan Approval Prediction using Machine Learning Algorithms" by Aditi Sharma et al.**, the authors outline a structured pipeline for predicting whether a loan should be approved or not. The study is divided into four phases: data collection, model comparison, training, and testing. Various machine learning models including Logistic Regression, Decision Tree, and Support Vector Machine were evaluated based on their accuracy and precision. Among these, Logistic Regression showed promising results due to its simplicity and effectiveness in binary classification tasks. The study emphasized the importance of data preprocessing steps such as handling missing values, encoding categorical data, and normalizing numeric features to improve model performance.

Another notable contribution is the paper **"Risk Reduction in Loan Lending using Machine Learning" by Priya R. et al.**, which focuses on reducing non-performing assets (NPAs) in banks and NBFCs by improving the loan applicant screening process. The paper highlights how feeding historical loan data into trained machine learning models can aid in identifying trustworthy borrowers. The authors experimented with models such as Random Forest and Gradient Boosting, which delivered better accuracy and robustness compared to simpler models. The study also sheds light on real-world challenges such as imbalanced datasets and the need for interpretability in financial decision-making. It concludes that incorporating machine learning into the lending process can significantly improve risk management and operational efficiency

Chapter 3: Design and Implementation

3.1. Introduction

This chapter outlines the design and implementation process for the Loan Eligibility Prediction System. The project adopts a structured machine learning pipeline to ensure accurate prediction of whether a customer is eligible for a loan based on various personal and financial attributes. The workflow consists of data preprocessing, feature selection, model training, validation, and performance evaluation. Several supervised machine learning algorithms are employed, including Logistic Regression, Random Forest, Support Vector Machines (SVM), and Gradient Boosting, with the goal of identifying the most effective model for real-time deployment.

3.2. Requirement Gathering Hardware

Requirements:

- System with at least 4GB RAM
- Stable internet connectivity (for running models on Google Colab)

Software Requirements:

- Google Colab or Jupyter Notebook (for implementation and testing)
- Python 3.x (programming environment)

3.3. Proposed Design

The system design follows the standard CRISP-DM (Cross-Industry Standard Process for Data Mining) methodology, ensuring a systematic approach to problem-solving and model development:

1. Data Collection: The dataset includes applicant details like gender, income, credit history, and property area. The target variable is Loan_Status, showing loan approval.
2. Data preprocessing included imputing missing values, encoding categorical features, scaling with StandardScaler, and handling outliers through visual and statistical methods..
3. New features like Total_Income and Loan_Amount_Term_Years were added, while redundant or correlated features were removed to boost accuracy.
4. Model Building:
    - Logistic regression
    - Random forest classifier
    - K-Nearest Neighbors (KNN)
    - Gaussian Naive Bayes
    - Support Vector Machine (SVM)
    - Gradient Boosting Classifier
5. Model Evaluation: Models were assessed using accuracy, precision, recall, F1-score, confusion matrix, and cross-validation, with the best-balanced model chosen for deployment.
6. Visualization: Data and model insights were visualized using count plots, bar charts, heatmaps, confusion matrices, and ROC curves.

3.4. Proposed Algorithm

Logistic Regression

Logistic Regression is a statistical model used for binary classification. It estimates the probability that a given input point belongs to a particular category. Due to its simplicity and interpretability, it was used as a baseline model. It performs well when the features have a linear relationship with the target variable.

Random Forest Classifier

Random Forest is an ensemble learning method that builds multiple decision trees and merges their predictions to improve accuracy and reduce overfitting. Each tree is trained on a random subset of the data and features. This model handles both categorical and numerical data efficiently and provides feature importance metrics for analysis.

K-Nearest Neighbors (KNN)

KNN is a non-parametric algorithm that classifies new data points based on the majority label of their 'k' nearest neighbors in the feature space. It is simple to implement but sensitive to the scale of the data and the choice of 'k'. Therefore, preprocessing steps like normalization were applied before training.

Gaussian Naive Bayes

Naive Bayes is a probabilistic classifier based on Bayes' theorem with the assumption of independence between predictors. The Gaussian variant assumes that the continuous features follow a normal distribution. It is efficient and performs well with high-dimensional data, especially when the assumption of independence roughly holds.

Support Vector Machine (SVM)

SVM constructs a hyperplane in a high-dimensional space to separate classes with maximum margin. It is particularly useful for datasets with clear class boundaries. The RBF (Radial Basis Function) kernel was used to capture non-linear patterns in the loan data. Hyperparameter tuning was performed on C and gamma to balance bias-variance trade-off.

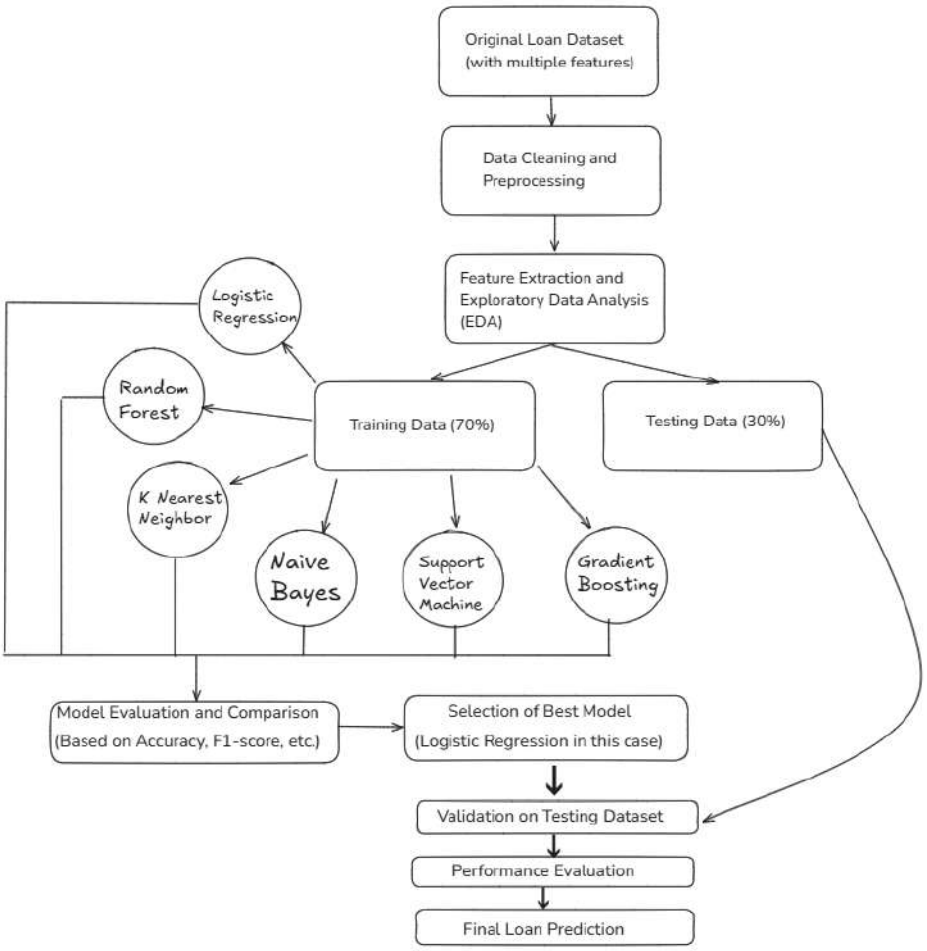
Gradient Boosting Classifier

Gradient Boosting is an advanced ensemble technique that builds models sequentially, each one trying to correct the errors made by the previous. It is highly effective for structured data and often provides better performance than other models, although it may require more training time. Learning rate, number of estimators, and tree depth were tuned to optimize its performance.

Model Selection and Comparison

Each of the above algorithms was trained and evaluated on the same preprocessed dataset. Accuracy, precision, recall, F1-score, and ROC-AUC were used as evaluation metrics. The model that achieved the best balance across all metrics was selected for deployment. In this project, the Gradient Boosting Classifier and Random Forest showed superior performance, making them strong candidates for the final system.

3.5. Architectural Diagrams



4: Results and Discussion

4.1. Introduction

To assess the performance of the developed loan eligibility prediction system, multiple machine learning models were trained and evaluated using a common dataset. Each model was tested using standard performance metrics such as **Accuracy**, which reflects the percentage of correctly predicted outcomes. These metrics provided insight into the strengths and limitations of each model in predicting whether a loan should be granted or not.

4.2 Cost Estimation

The system was developed using open-source Python libraries such as Scikit-learn, Pandas, NumPy, and Matplotlib, and executed in a Google Colab environment. This eliminated the need for expensive local infrastructure, making the solution highly cost-effective and suitable for academic research or small-scale enterprise deployment.

4.3 Feasibility Study

Given its low hardware footprint and ability to run effectively in cloud environments, the system demonstrates strong feasibility for integration into online loan application portals. The real-time prediction capability and scalability of the chosen models make this system well-suited for financial institutions looking to automate and streamline their loan approval workflow.

4.4 Results of Implementation

Model	**Accuracy**
Logistic Regression	**97.00**
Naive Bayes	**95.92**
Gradient Boosting Classifier	**93.84**
Random Forest	**89.76**
Support Vector Machine (SVM)	**71.89**
K Nearest Neighbor	**61.08**

5: Conclusion

5.1. Conclusion

In this project titled *"Loan Eligibility Prediction using Machine Learning"*, we built a robust and automated system to predict whether a customer is eligible for a home loan based on various demographic and financial factors. We explored multiple machine learning algorithms, including Logistic Regression, Random Forest, K-Nearest Neighbors, Naive Bayes, SVM, and Gradient Boosting, and compared their performance to identify the most accurate model.

We began by collecting and preprocessing the data, which involved handling missing values, encoding categorical features, and normalizing inputs where necessary. We then split the data into training and testing sets to evaluate model performance fairly. We trained each classification model and evaluated them using accuracy as the primary metric.

We found that Logistic Regression achieved the highest accuracy (~97%), making it the most effective model for this classification problem. Other models such as Naive Bayes and Gradient Boosting also performed reasonably well, while SVM and KNN showed comparatively lower accuracy.

We concluded that logistic regression not only offered good predictive power but also maintained interpretability, making it a suitable choice for real-time loan eligibility checks. We implemented the solution using Python and open-source libraries on Google Colab, which kept the project cost-effective and accessible.

Overall, we demonstrated that machine learning can effectively support financial institutions in automating the loan eligibility process and targeting the right customer segments, thereby improving decision-making and operational efficiency.

5.2. Future Scope

- While the current system successfully predicts loan eligibility using multiple machine learning models, there are several directions in which this work can be extended to enhance its applicability and performance:
- **Real-time Deployment:** Integrating the trained models into production using RESTful APIs can enable real-time eligibility checks during the loan application process, improving user experience and operational efficiency.
- **Feature Expansion:** Including more detailed financial and behavioral attributes such as previous loan history, spending habits, credit card usage, and repayment patterns could provide deeper insights and improve prediction accuracy.
- **Ensemble and Stacking Models:** Future versions could explore stacked ensembles combining the strengths of top-performing models to boost overall accuracy and generalization.
- **Continuous Model Training:** Incorporating mechanisms for continuous learning and retraining with new customer data would help the system remain current with changing customer behaviors and financial trends.
- **Mobile and Web Integration:** Embedding the predictive system into mobile or web loan application portals can make the process seamless for users, helping financial institutions provide instant decisions and reduce dropout rates.

5.3. Societal Impact

The development and deployment of loan eligibility prediction systems have significant implications for society, particularly in the context of financial inclusion and responsible lending practices.

● **Financial Inclusion:** Automated loan eligibility prediction helps provide equal access to credit by reducing bias in decision-making. By using data-driven models, individuals from diverse socioeconomic backgrounds can be evaluated fairly, improving access to financial services for underserved communities.

● **Responsible Lending**: Predictive models assist financial institutions in identifying customers who are more likely to repay their loans, reducing the risks of over-lending and defaults. This promotes responsible lending practices and contributes to the stability of financial markets.

● **Empowering Individuals:** By automating loan eligibility assessments, individuals can receive faster decisions, reducing the waiting time and uncertainty associated with loan applications. This improves customer experience and empowers individuals to make informed financial decisions.
● **Data Privacy and Protection**: Predicting loan eligibility involves handling sensitive personal and financial data. Ensuring robust data privacy measures and compliance with regulations like GDPR is critical to safeguarding user information and building trust in automated financial systems.
● **Educational and Research Value:** The techniques used in this project provide valuable learning opportunities for students and professionals in the fields of machine learning and data science. It promotes a deeper understanding of financial analytics and predictive modeling, contributing to the advancement of knowledge in financial technologies.

Name:- Aditya Ahuja
Rollno- 02 Class:- D15C

04/03

AIDS (Assignment-1)

1. What is AI? Considering the COVID-19 pandemic situation, how did AI help to survive and renovate our way of life with different applications?

→ AI (Artifical intelligence) is the replication of human intelligence in machines, enabling them to learn, reason, solve problems, and make decisions without direct human intervention. AI played a vital role in manging the COVID-19 crisis by detecting outbreaks early through data analysis accelerating drug and vaccine development, automating medical diagnoses with imaging and predictive models, supporting telemedicine via AI chatbots, optimizing supply chains for essential goods, and enhancing remote work and online education through AI-driven automation, making daily life more adaptable and efficient.

2. What are AI Agents terminology? Explain with examples

→ AI agents are systems that perceive their environment, process information, and take actions to achieve specific goals.

1. Agent - An entity that interacts with environment (eg self-driving car)

2. Environment - The external system in which the agent operates (eg. traffic conditions for a self-driving car)

3. Perception: Data collected from sensors (eg: a robot's camera detecting obstacles)

4. Actuators - Components that execute actions (eg - robotic arms in manufacturing)

5. Rationality - The ability to make decisions based on available information (eg: AI trading bots adjusting stock investments)

6. Types of Agents:

Simple Reflex Agents - Act based on current perception

Model Based Agents - Use internal models to predict future states (eg - GPS navigation systems)

Goal Based Agents - Take actions to achieve a specific goal eg (Chess playing AI)

Utility Based Agents - Optimize outcomes for maximum benefit (eg recommendation systems in e-commerce)

Learning Agents - Improve performance over time using past experience (eg self learning robot in warehouse)

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

-> The 8-puzzle problem is solved using AI search techniques that explore possible moves to reach the goal state. Common AI techniques include:

1. Uninformed Search:

- Breadth First Search (BFS): Explores all possible moves level by level but is inefficient for large problems.

• Depth-First Search (DFS):- Explores one path deeply before backtracking but may get stuck in loops.

2. Informed search (Heuristic Based):

• Best-first Search (Greedy Algorithm): Selects moves based on heuristic values like misplaced tiles.

A* Algorithm uses heuristic function which is
$f(n) = g(n) + h(n)$

9. What is PEAS descriptor?

-> PEAS (Performance measure, Environment, Actuators, Sensors) is a frame work used to define the components of an AI agent by specifying how it interacts with its environment and evaluate its success.

1. Taxi Driver
Performance Measure: Safety, speed, fuel efficiency
Environment: Roads, traffic, pedestrians, weather conditions
Actuators: steering, acceleration, brakes, turn signals
Sensors: GPS, cameras, speedometer, LIDAR, fuel gauge

2. Medical Diagonsis System
Performance measure: Accuracy of diagnosis, patient recovery rate
Environment: Patient data, medical records, symptoms, lab reports
Actuators: Display diagnosis, prescribe medication, recommend tests
Sensors: Patient input, test results, doctor's note, medical imaging

3. A music Composer
Performance Measure: Musical quality, originality, listener preference
Environment: Musical genres, user preferences, historical compositions
Actuators: Generating notes, melodies, instrument selection
Sensors: User feedback, music databases, emotional tone analysis

4. An aircraft Autolander
performance measure: Safe landing, smooth touchdown, passenger comfort
Environment: weather conditions, runway, air traffic
Actuators: flaps, landing gear, brakes, engine thrust control
Sensors: Altimeter, GPS, wind speed sensors, radar, cameras

5. An Essay Evaluator
Performance measure: Grammar Accuracy, coherence, relevance
Environment: Essays, writing, rules, predefined grading criteria
Actuators: Score assignment, grammar suggestions, feedback suggestions
Sensors: Text input, grammar and plagiarism checkers, semantic analysis

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

=> A shopping bot for an offline book store can be categorized accordingly to the six AI environment dimension as follows

1. Observability: Partially observable The bot may not have full knowledge of stock availability customer preferences or real time changes in book store.

2. Deterministic vs stochastic: Stochastic. Book availability, customer behavior, and price changes introduce randomness, making the environment unpredictable
3. Episodic vs Sequential: Sequential - Each decision (eg. recommending a book) affects future interactions, meaning past actions influence subsequent outcomes
4. Discrete vs Continuous: Discrete: The bot operates with distinct choices, such as recommending a book or processing a purchase
5. Single Agent vs Multi Agent: Multi-Agent: The bot interacts with customers, bookstore staff, and possibly other AI systems (eg. inventory management) making it a multi-agent system

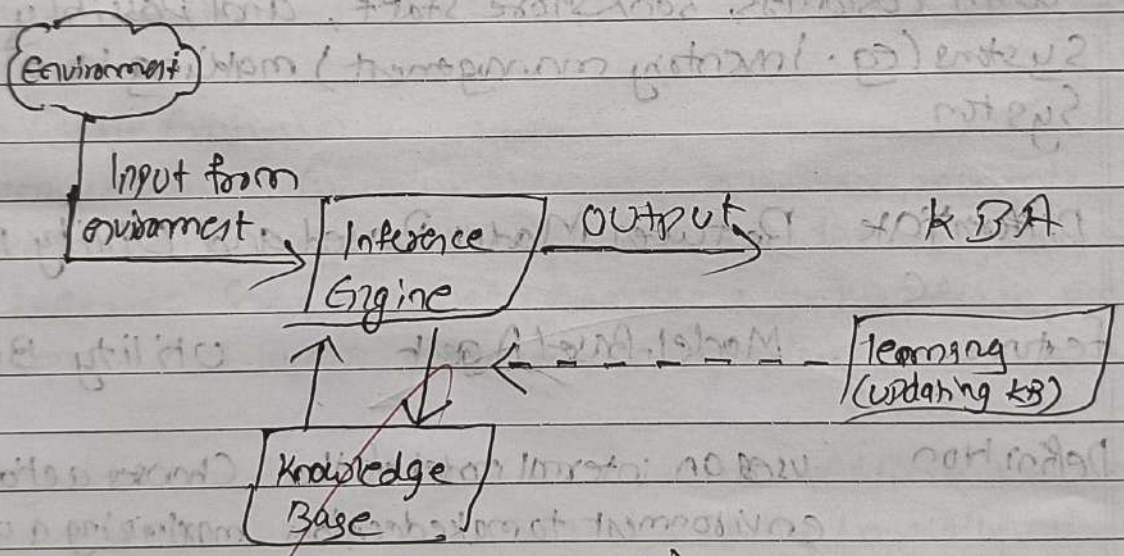
6. Differentiate Between Model Based and Utility Based agent

Feature	Model-Based Agent	Utility Based Agent
Definition	used an internal model of the environment to make decisions	Chooses actions based on maximizing a utility function
Decision making	Predicts future states using the model before acting	Evaluates multiple possible actions and selects the best based on utility
Goal-oriented	Yes, but focuses on how the environment works	yes, but aims to achieve the best possible outcome

Optimality	May not always take the best possible action, only a feasible one	Always selects the action that provides the highest
Example	A self driving car using a traffic model to predict congestion	A stock trading AI selecting the trade with highest expected profit

07 Explain the architecture of a knowledge based agent and learning Agent

→ Architecture of a knowledge-Based agent

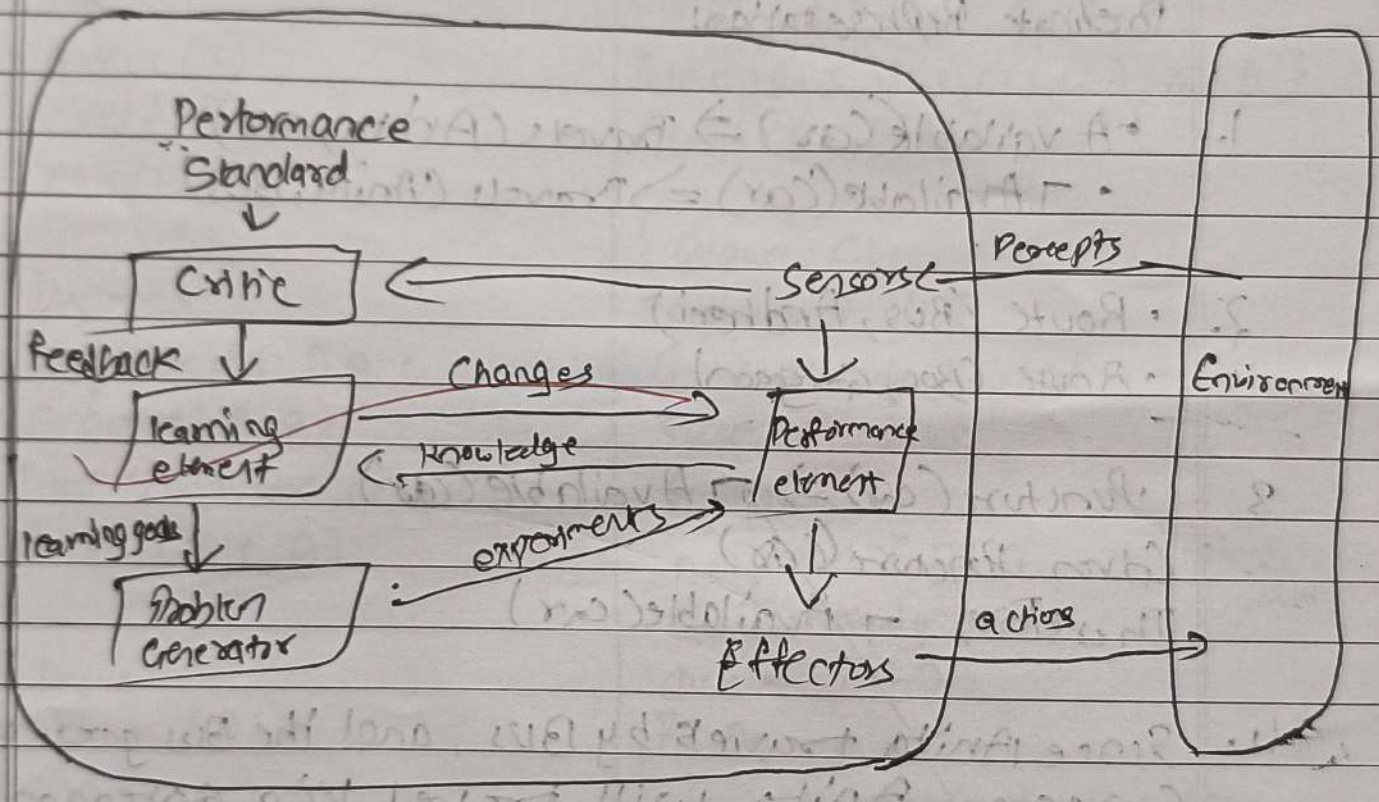


A knowledge-Based Agent (KBA) uses stored knowledge to make decisions and reason about the environment. Its architecture consists of

1. Knowledge Base (KB) - stores facts, rules, and background knowledge in a structured format
2. Inference Engine - Applies logical reasoning to derive conclusions from knowledge Base

3. Perception Module: Gathers input from sensors or external sources
4. Action Module: Executes actions based on derived conclusions
5. Learning Mechanism: Updates the knowledge base with new information

Architecture of a learning Agent



1. Learning Agent: learns from interactions and update knowledge
2. Performance Agent: Uses the learned knowledge to make decisions
3. Critic: Evaluates the agent's performance by comparing actions with desired outcomes
4. Problem Generator: Suggests new exploratory actions to improve learning.

9. Convert the following to predicates:

a) Anita travels by car if available otherwise travels by bus
b) Bus goes via Andheri and Goregaon
c) Car has puncture so is not available

Anita
Will travel ~~thai~~ via Goregaon!

Predicate Representation:

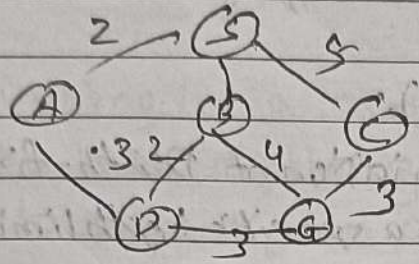
1. • Available(car) ⇒ Travels(Anita, car)
   • ¬ Available(car) ⇒ Travels(Anita, Bus)

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

3. • Puncture(car) ⇒ ¬ Available(car)
   Given Puncture(car)
   Therefore, ¬ Available(car)

4. Since Anita travels by Bus, and the Bus goes via Goregaon, Anita will travel via Goregaon

10 find the route from S to G using Bfs



1. Start:
Queue [S]
visited {S}
parent: {}

2. Dequeue S:
Neighbors of S: A, B, C
Enqueue: A, B, C
Parent: {A:S, B:S, C:S, D:A}
Queue: [B, C, D]

3. Dequeue A:
neighbors of A: D
Enqueue D
Visited {S, A, B, C, D}
Parent {A:S, B:S, C:S, D:A}
Queue [B, C, D]

4. Dequeue B:
neighbors of B: D, G
D is already visited
Enqueue G
visited {S, A, B, C, D, G}
Parent {A:S, B:S, C:S, D:A, G:B}
Queue: [C, D, G]

5. Dequeue C:
neighbor of D: G
G is already visited
Queue: [G]

6. Dequeue D:
neighbors of D: G
G is already visited
Queue: [G]

7. Dequeue G:
G is the destination we stop

path is S → B → G

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

Depth-limited search (DLS):
Depth limited search is a variation of Depth-first search where the search is restricted to a specific depth limit 'l'. If a goal is not found within this limit, the search terminates, it prevents infinite loops in deep or infinite state spaces but risks failing to find a solution if 'l' is too low

Example:
If l=2 DFS explores node only up to depth 2

Iterative Deepening Search:
IDS combines the space efficiency of DFS and completeness of BFS by repeatedly running DLS with increasing depth limits (l=0, 1, 2 ... until the goal is found

Example: (Searching for F in the given graph)

DLS with l=0 only checks S, goal not found
DLS with l=1 expands S, explores A, B, C goal not found
DLS with l=2 expands A, B, C explores D, G goal found at depth 2

Advantage: Guarantees finding the shortest path while using less memory than BFS

12 Explain Hill Climbing and its drawbacks in detail with example. Also State limitations of steepest ascent hill climbing

⇒ Hill Climbing is a local search algorithm that continously moves toward the best neighboring state with a higher heuristic value, aiming to reach a global optimum

Steps:

1. Start from an initial state
2. Evaluate neighboring states and move to the one with the best heuristic value
3. Repeat until no better neighbor exists (local maximum)

Example:

Imagine a mountain climbing scenario where a hiker moves uphill based on steepest slope. If they reach a peak that is not the highest (global maximum) they might get stucked

limitations of Steepest-Ascent Hill Climbing:

1. Slow progress in Narrow Ridges: If the optimal path requires small incremental changes, the algorithm struggles:
2. More likely to Get Stuck in local Maxima: Choosing only the steepest path can lead to premature convergence.
3. Inefficient in large search spaces: Evaluating all neighbors increases computation time.

Soln

Random Restart: start from different points.

Simulated Annealing: Occasionally accept worse moves to escape local Optima.

Q13 Explain Simulated Annealing and write its algorithm

Simulated Annealing is a probabilistic optimization algorithm inspired by annealing process in metallurgy. It helps in finding a global optimum by allowing occasional moves to worse solutions to escape local optima.

Algorithm:

1. Initialize the current state and temperature T.
2. Repeat until stopping conditions:
   - Select a random neighbor state
   - Compute energy difference $\Delta E = E_{new} - E_{current}$
   - if $\Delta E < 0$, accept the new state
   - Else, accept it with probability $e^{-\Delta E / T}$ (Metropolis criterion)
   - Reduce the temperature T (cooling schedule)
3. Return the best found solution

Q14 Explain A* Algorithm with an example

→ The A* Algorithm is an informed search algorithm that finds the optimal path by considering both the cost to reach a node $g(n)$ and the estimated cost to the goal $h(n)$ using the formula

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

where

$g(n)$ = actual cost from start node to n

$h(n)$ = estimated cost from n to goal (heuristic)

Q15 Explain MiniMax algorithm and draw game tree for tic tac toe

The minimax algorithm is used in adversarial search (eg two player games) to determine the optimal move by assuming both players play optimally
- maximizer (eg. X) tries to get the highest score
- minimizer (eg. O) tries to reduce the score

Algorithm:
1. Generate the game tree upto depth limit
2. Assign heuristic values to leaf nodes
3. Back propagate values:
   - maximizer picks the maximum value
   - minimizer picks the minimum value
4. The root node selects the best move based on propagated values

Game tree:
A minimax tree for Tic-Tac-Toe would show all possible board states, evaluating the best move at each step

Q16 Explain Alpha-beta pruning Algorithm for adversarial search with an example

→ Alpha beta pruning optimizes Minimax by skipping unnecessary branches, reducing computations. It uses two values

- Alpha (α): Best score maximizer can achieve
- Beta (β): Best score minimizer can achieve

Algorithm:
1. Perform Minimax search
2. prune branches where
   - If maximizer's best (α) $\geq$ minimer's best (β), further evaluation is skipped
   - If minimizer's best (β) $\leq$ maximizer's best (α), further evaluation is skipped.
3. This reduces time complexity without affecting the final decision

Example:
In a game tree, if a branch has already provided a worse outcome than the best known move, it is ignored to save time

Q17 Explain Wumpus World Environment giving its PEAS description, Explain how percept sequence is generated ?

=> The Wumpus world is a grid-based AI environment where an agent explores a cave while avoiding hazards like pits and wumpus monster

PEAS Descriptor:

- Performance Measure: Reaching the gold safely, minimizing steps
- Environment: A 4x4 grid with pits, gold and Wumpus
- Actuators: move forward, turn, grab, shoot, climb
- Sensors: perceive stench (wumpus nearby), breeze (pit nearby), glitter (gold found)

Percept Sequence Generation:

1. Agent starts at (1, 1) sensing its surroundings
2. If stench is detected the wumpus is nearby
3. If breeze is detected a pit is nearby
4. The agent infers safe paths and navigates towards the gold while avoiding hazards.

Q18 Solve the following Crypto-arithmetic problem: SEND + MORE = MONEY

→

SEND	9 5 6 7	
+ MORE	+ 1 0 8 5	
MONEY	1 0 6 5 2	

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

Q19 Consider the following axioms:

1. Represent these axioms in First order predicate logic
1. $\forall x$ (Graduating(x) $\Rightarrow$ Happy(x))
2. $\forall x$ (Happy(x) $\rightarrow$ Smiling(x))
3. $\exists x$ (Graduating(x))

2. Convert each formula to clause form:
1. $\neg$ Graduating(x) $\vee$ Happy(x)
2. $\neg$ Happy(x) $\vee$ Smiling(x)
3. Graduating(A) (assuming "A" is the individual who is graduating)

3. Prove that "Is Someone Smiling?" using resolution technique:
- From 3: Graduating(A).
- Using 1: Happy(A) (Modus Ponens)
- Using 2: Smiling(A)

Then Someone is Smiling.

Resolution Tree:

1. Graduating(A) (Given)
2. Graduating(A) $\rightarrow$ Happy(A) $\rightarrow$ Happy(A)
3. Happy(A) $\rightarrow$ Smiling(A) $\rightarrow$ Smiling(A)

Q20 Explain Modus Ponens with a suitable example

→ Modus Ponens (Law of Detachment) is a rule of inference stating: $P \rightarrow Q, P \Rightarrow Q$

Example:
1. If it rains, the ground gets wet (Premise: Rain → wet ground)
2. It is raining (Premise: Rain)
3. Conclusion: The ground is wet (Applying Modus Ponens)

Q21 Explain forward Chaining and backward Chaining algorithm with an example

→ forward chaining:
- Data-driven inference: Starts from known facts and applies rules to reach a goal
- Used in expert systems (eg. medical diagnosis)

Example:

1. fact: "sore throat"
2. Rule: "If sore throat → infection"
3. New fact: "Infection"
4. Rule: "If infection → need antibiotics"
5. Conclusion: "Need antibiotics"

Backward Chaining:

- Goal-driven inference: starts from the goal and works backward to find supporting facts
- Used in AI reasoning and theorem proving

Example:

Goal: Does the patient need antibiotics?

1. Check: Does the patient have an infection?
2. Check: Does the patient have a sore throat?
3. If both hold, conclude "Need antibiotics"

This reduces unnecessary computations by only exploring relevant facts.

Name: Aditya Ahuja D15C Roll No: 02

AIDS-I Assignment No: 2

Q.1: Use the following data set for question 1
82, 66, 70, 59, 90, 78, 76, 95, 99, 84, 88, 76, 82, 81, 91, 64, 79, 76, 85, 90

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

Answer:

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

$$\boldsymbol{Mean = \frac{SumOfVal}{NumOfVal} = \frac{1625}{20} = 81.25}$$

$$\boldsymbol{Median = \frac{\left(\frac{n}{2}\right) + \left(\frac{n}{2}\right) + 1}{2} = \frac{10^{th} + 11^{th}}{2} = \frac{81 + 82}{2} = 81.5}$$

Mode: 76 appears 3 times and has the maximum frequency, So Mode = 73.

Interquartile Range (IQR)

Q1 (25th percentile): Median of the first 10 numbers:

59, 64, 66, 70, 76, 76, 76, 78, 79, 81

$$\boldsymbol{Median = \frac{\left(\frac{n}{2}\right) + \left(\frac{n}{2}\right) + 1}{2} = \frac{5^{th} + 6^{th}}{2} = \frac{76 + 76}{2} = 76}$$

Q3 (75th percentile): Median of the last 10 numbers:

82, 82, 84, 85, 88, 90, 90, 91, 95, 99

$$\boldsymbol{Median = \frac{\left(\frac{n}{2}\right) + \left(\frac{n}{2}\right) + 1}{2} = \frac{5^{th} + 6^{th}}{2} = \frac{88 + 90}{2} = 89}$$

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

Answer:

1. **Machine Learning for Kids**
    - **Target Audience**: Primarily designed for school students aged 8 to 16 years and educators aiming to introduce artificial intelligence (AI) and machine learning (ML) concepts in a classroom setting.
    - **Use by Target Audience**: This tool enables students to build and train machine learning models using text, numbers, or images through a simplified and interactive platform. It integrates with block-based programming environments like Scratch and supports Python, allowing students to apply their trained models in creative projects. Educators utilize the tool to teach basic principles of machine learning in an accessible manner without requiring prior coding experience.
    - **Benefits:**
        - User-friendly interface lowers the barrier to entry for beginners.
        - Promotes early AI literacy.
        - Integration with Scratch and Python makes it suitable for educational use.
    - **Drawbacks:**
        - Limited depth for advanced machine learning applications.
        - Restricted control over underlying algorithms or training processes.
        - Primarily designed for learning purposes rather than building robust or scalable models.
    - **Analytic Type:** Predictive Analytics – The tool is used to build models that can predict outcomes based on input data, such as classifying text or images.
    - **Learning Type:** Supervised Learning – Requires labeled training data where both input and the correct output are provided to train the model.
2. **Teachable Machine (by Google)**
    - **Target Audience:** Targets a broader audience, including students, educators, hobbyists, and creators who wish to explore machine learning concepts without programming expertise.
    - **Use by Target Audience:** The tool is primarily used for creating models that can recognize images, sounds, or poses using a webcam or microphone. It allows users to collect data, train models, and export them for use in websites, apps, or TensorFlow-compatible projects. Commonly used in educational workshops, personal projects, and prototyping scenarios.
    - **Benefits:**
        - Highly intuitive interface enables quick training and testing of machine learning models.
        - Supports real-time feedback.
        - Models can be exported for further use.
        - Simplifies the process of creating functional machine learning models without the need for code.
    - **Drawbacks:**
        - Simplicity limits capabilities for more complex or large-scale applications.
        - Lacks advanced customization options.
        - Not suitable for detailed algorithmic control.
    - **Analytic Type**: Predictive Analytics – Designed to make predictions based on trained data, such as identifying whether a captured image or sound matches a particular class.
    - **Learning Type**: Supervised Learning – Users provide labeled examples during the training process to teach the model how to classify future inputs.

Q.3: Data Visualization and Misinformation

The article *"What's in a Chart? A Step-by-Step Guide to Identifying Misinformation in Data Visualization"* by Arthur Kakande outlines a framework for identifying misleading elements in data visualizations. It emphasizes examining the credibility of sources, assessing design choices such as chart types and axis scales, and recognizing common deceptive techniques like cherry-picking data or omitting essential context.

Similarly, in *"How Bad Covid-19 Data Visualizations Mislead the Public,"* Katherine Ellen Foley critiques real-world examples from U.S. state dashboards during the early COVID-19 pandemic. She highlights poor design practices such as the use of pie charts, missing axes, and inconsistent scales, all of which led to public misinterpretation of data related to case counts and risk.

Current Event Example: Misrepresentation of COVID-19 Vaccine Data by OpenVAERS

A pertinent example of misinformation through data visualization is the case of OpenVAERS, an American anti-vaccine website. OpenVAERS misrepresents data from the Vaccine Adverse Event Reporting System (VAERS) to promote misinformation about COVID-19 vaccines. The website presents unverified data and statistics on the number of people who have allegedly died or suffered injuries after vaccination, often without appropriate context or disclaimers. This decontextualization leads to misleading interpretations and fuels vaccine hesitancy among the public.

Analysis:

This example underscores the critical need for transparency and context in data visualization. Even when data is accurate, the way it is presented can significantly influence public perception and decision-making. As highlighted by Kakande and Foley, it's essential to evaluate visualizations not only for their correctness but also for how effectively they communicate truthful and clear information.

This case reinforces the importance of thoughtful, transparent data visualization. Even when data itself is accurate, poor visual design or lack of context can distort the message. As both Kakande and Foley emphasize, visualizations must be evaluated critically—not only for correctness but also for how effectively they support truthful, clear communication.

Cited Source:
Giles, Chris. "Measurement matters." *Financial Times*, June 21, 2024.
https://www.ft.com/content/942743b1-5949-4823-a42f-a8a9d904c35e

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

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

Requirements to satisfy

- Programming Language: Python
- Class imbalance should be resolved
- Data Pre-processing must be used
- Hyper parameter tuning must be used
- Train, Validation and Test Split should be 70/20/10
- Train and Test split must be randomly done
- Classification Accuracy should be maximized

- Use any Python library to present the accuracy measures of trained model

Answer:

Handle Class Imbalance Using SMOTE (Synthetic Minority Over-sampling Technique)

```
from imblearn.over_sampling import SMOTE
from collections import Counter
from sklearn.model_selection import train_test_split

# Separate features and target
X = df.drop(columns=['Outcome'])
y = df['Outcome']

# Show original class distribution
print("Original class distribution:", Counter(y))

# Apply SMOTE to balance classes
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Show new class distribution
print("Resampled class distribution:", Counter(y_resampled))
```

```
Original class distribution: Counter({0: 500, 1: 268})
Resampled class distribution: Counter({1: 500, 0: 500})
```

Standardization during data preprocessing

```
from sklearn.preprocessing import StandardScaler

# Initialize the scaler
scaler = StandardScaler()

# Fit only on training data and transform both train and test
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Train, Validation, and Test Split (Random & Stratified 70/20/10)

```
from sklearn.model_selection import train_test_split

# Step 1: Split into train (70%) and temp (30%)
X_train, X_temp, y_train, y_temp = train_test_split(
    X_resampled, y_resampled, test_size=0.30, random_state=42, stratify=y_resampled)

# Step 2: Split temp into validation (20%) and test (10%) → 2:1 ratio of 30%
X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.33, random_state=42, stratify=y_temp)  # 0.33 of 30% ≈ 10%

# Confirm sizes
print("Train size:", len(X_train))
print("Validation size:", len(X_val))
print("Test size:", len(X_test))
```

```
Train size: 700
Validation size: 201
Test size: 99
```

Random Forest with Hyperparameter Tuning (Using Validation Set)

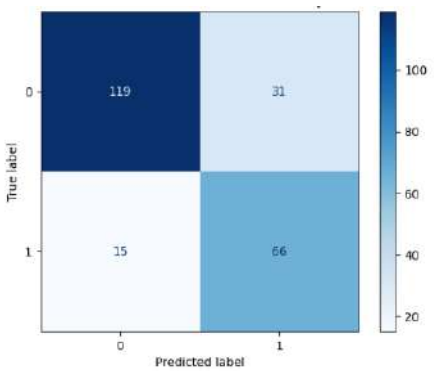
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# Define parameter grid
param_grid_rf = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

# Initialize Random Forest
rf = RandomForestClassifier(random_state=42)

# Grid search with 5-fold cross-validation
grid_search_rf = GridSearchCV(estimator=rf, param_grid=param_grid_rf, cv=5, n_jobs=-1, scoring='accuracy')
grid_search_rf.fit(X_train, y_train)

# Best model
best_rf = grid_search_rf.best_estimator_
print("Best Parameters:", grid_search_rf.best_params_)
```

```
Best Parameters: {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 150}
```

Evaluate Accuracy and Visualize Performance

```
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay, accuracy_score
import matplotlib.pyplot as plt

# Predict on test data
y_pred_rf = best_rf.predict(X_test)

# Accuracy
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print(f"\nTest Accuracy: {accuracy_rf:.4f}")

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

# Confusion matrix
cm_rf = confusion_matrix(y_test, y_pred_rf)
disp_rf = ConfusionMatrixDisplay(confusion_matrix=cm_rf, display_labels=best_rf.classes_)
disp_rf.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix - Random Forest")
plt.show()
```

```
              0       0.89      0.79      0.84       150
              1       0.88      0.81      0.74        81

       accuracy                           0.80       231
      macro avg       0.88      0.80      0.79       231
   weighted avg       0.82      0.80      0.80       231

✅ Test Accuracy: 87.09 %
```



Test Accuracy: 87.09%
Precision, Recall, F1-Score (both classes close to or above 0.82)
Balanced Confusion Matrix:
True Negatives (0 predicted as 0): 46

True Positives (1 predicted as 1): 46
False Positives: 4
False Negatives: 3

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

Answer:

Read the file and set the dependent and independent variables

```
# Step 1: Load and Prepare the Data

import pandas as pd
from sklearn.model_selection import train_test_split

# Load the dataset
data = pd.read_csv('BostonHousing.csv')

# Independent variable: 1st column
X = data.iloc[:, [0]]  # 'crim'

# Dependent variables: columns 2 to 5
y = data.iloc[:, 1:5]  # 'zn', 'indus', 'chas', 'nox'
```

Split data to Train/Test – 70% and 30%

```
# Split data into 70% train, 30% test (random split)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Display shapes to confirm
print("X_train:", X_train.shape)
print("y_train:", y_train.shape)
print("X_test:", X_test.shape)
print("y_test:", y_test.shape)
```

```
X_train: (354, 1)
y_train: (354, 4)
X_test: (152, 1)
y_test: (152, 4)
```

RandomForest Regressor

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import r2_score, mean_squared_error

# Define the model
rf = RandomForestRegressor(random_state=42)

# Define parameter grid for tuning
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [3, 5, 7],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

# Setup GridSearchCV
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid,
                           cv=5, n_jobs=-1, scoring='r2', verbose=1)

# Fit on training data
grid_search.fit(X_train, y_train)

# Best model
best_rf = grid_search.best_estimator_

# Predict on test data
y_pred = best_rf.predict(X_test)

# Evaluate
r2 = r2_score(y_test, y_pred)
n = X_test.shape[0]
p = X_test.shape[1]
adjusted_r2 = 1 - (1 - r2) * (n - 1) / (n - p - 1)
mse = mean_squared_error(y_test, y_pred)

# Print results
print(f"Best Parameters: {grid_search.best_params_}")
print(f"R² Score: {r2:.4f}")
print(f"Adjusted R² Score: {adjusted_r2:.4f}")
print(f"Mean Squared Error: {mse:.4f}")
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
Best Parameters: {'max_depth': 3, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 100}
R² Score: 0.99182
Adjusted R² Score: 0.99160
Mean Squared Error: 0.4213
```

Best Parameters: The model performs best using a moderately deep tree (max_depth=3) with specific values for splits and number of trees.
R² Score: 0.99182 — The model explains 99.18% of the variance in the dependent variables, indicating high accuracy.
Adjusted R² Score: 0.99160 — After accounting for the number of predictors, 99.16% of the variance is still explained, confirming a good generalization.
Mean Squared Error: 0.4213 — The average squared difference between predicted and actual values is very low, showing minimal error in predictions.

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

Answer:

Key Features of the Wine Quality Dataset:

Fixed Acidity – Non-volatile acids contributing to taste. Moderate influence on quality.
Volatile Acidity – High levels cause vinegar taste. Strong negative impact on quality.
Citric Acid – Adds freshness and enhances flavor. Mild positive effect.

Residual Sugar – Affects sweetness. Minor impact unless in large amounts.
Chlorides – Salt content. Low influence but can alter taste.
Free Sulfur Dioxide – Prevents spoilage. Moderate effect.
Total Sulfur Dioxide – Sum of all sulfur dioxide. Excess may reduce quality.
Density – Related to sugar and alcohol. Helps estimate composition.
pH – Measures acidity. Moderate effect on balance and taste.
Sulphates – Preservatives. Strong positive impact on quality.
Alcohol – Strongest positive correlation with quality.

Handling Missing Data (Feature Engineering):

Mean Imputation: Replaces missing values with the mean of the column. Simple and fast, but may reduce variance in the data.
Median Imputation: Replaces missing values with the column median. More robust to outliers than mean imputation.
Mode Imputation: Replaces missing values with the most frequent value. Ideal for categorical or discrete features.
KNN Imputation: Estimates missing values using similar rows (nearest neighbors). More accurate, but computationally expensive.
Dropping Rows: Removes rows with missing values. Fastest method but only recommended when missing data is minimal.

Advantages and Disadvantages of Different Imputation Techniques:

Mean Imputation:
Advantage: Simple and fast.
Disadvantage: Distorts variance, especially if data has outliers.
Median Imputation:
Advantage: More robust against outliers than mean.
Disadvantage: Still ignores feature relationships.
Mode Imputation:
Advantage: Best suited for categorical features.
Disadvantage: Not meaningful for continuous numerical data.
KNN Imputation (Selected):
Advantage: Considers similarity between rows, more accurate, realistic values.
Disadvantage: Computationally slower on large datasets.
Iterative Imputation (MICE):
Advantage: Captures multivariate relationships using regression.
Disadvantage: Complex and time-consuming.
Dropping Rows:
Advantage: Quick and easy.
Disadvantage: Risk of losing valuable data, not suitable if missing data is frequent..

```python
from sklearn.impute import KNNImputer
import pandas as pd

# Load your dataset
df = pd.read_csv('WineQT.csv')  # Replace with your actual file name

# Initialize KNN Imputer
knn_imputer = KNNImputer(n_neighbors=5)

# Apply imputation (excluding non-numeric or target columns if needed)
df_imputed = pd.DataFrame(knn_imputer.fit_transform(df), columns=df.columns)

# Check for remaining missing values
print("Missing values after KNN Imputation:\n", df_imputed.isnull().sum())
```

```
Missing values after KNN Imputation:
 fixed acidity           0
volatile acidity         0
citric acid              0
residual sugar           0
chlorides                0
free sulfur dioxide      0
total sulfur dioxide     0
density                  0
pH                       0
sulphates                0
alcohol                  0
quality                  0
Id                       0
dtype: int64
```