EXPERIMENT NO. 1

Problem Statement: Introduction to Data science and Data preparation using Pandas steps.

Theory:

Data science is a multidisciplinary field focused on deriving valuable insights from both structured and unstructured data through scientific methods, algorithms, and systems. A crucial initial step in any data science project is data preparation, which involves cleaning, transforming, and structuring raw data to improve its quality and ensure it is suitable for analysis.

In this experiment, we implemented data preprocessing techniques using the Pandas library in Python. The dataset analyzed consists of records of car accidents in NYC from 2020, including key attributes such as the number of injuries, fatalities, latitude, longitude, contributing factors, and vehicle types involved. Initially, the dataset contained missing values, inconsistencies, and redundant columns, making it necessary to conduct comprehensive cleaning and preprocessing to enhance data quality.

The following key steps were carried out in this process:

- Importing the dataset into Pandas.
- Detecting and handling missing values.
- Removing redundant columns.
- Applying ordinal encoding to categorical variables.
- Identifying and managing outliers.
- Standardizing and normalizing numerical features.

1. **Loading data into pandas:**

```
import pandas as pd
df = pd.read_csv(r"C:\Users\bhumi\OneDrive\文档\ds_lab_csv\nyc_accidents.csv")
df.info()
```

```
========== RESTART: C:\Users\bhumi\OneDrive\文档\ds_lab_csv\exp1.py ==
==
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 74881 entries, 0 to 74880
Data columns (total 29 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   CRASH DATE                     74881 non-null  object
 1   CRASH TIME                     74881 non-null  object
 2   BOROUGH                        49140 non-null  object
 3   ZIP CODE                       49134 non-null  float64
 4   LATITUDE                       68935 non-null  float64
 5   LONGITUDE                      68935 non-null  float64
 6   LOCATION                       68935 non-null  object
 7   ON STREET NAME                 55444 non-null  object
 8   CROSS STREET NAME              35681 non-null  object
 9   OFF STREET NAME                19437 non-null  object
 10  NUMBER OF PERSONS INJURED      74881 non-null  int64
 11  NUMBER OF PERSONS KILLED       74881 non-null  int64
 12  NUMBER OF PEDESTRIANS INJURED  74881 non-null  int64
 13  NUMBER OF PEDESTRIANS KILLED   74881 non-null  int64
 14  NUMBER OF CYCLIST INJURED      74881 non-null  int64
 15  NUMBER OF CYCLIST KILLED       74881 non-null  int64
 16  NUMBER OF MOTORIST INJURED     74881 non-null  int64
 17  NUMBER OF MOTORIST KILLED      74881 non-null  int64
 18  CONTRIBUTING FACTOR VEHICLE 1  74577 non-null  object
 19  CONTRIBUTING FACTOR VEHICLE 2  59285 non-null  object
 20  CONTRIBUTING FACTOR VEHICLE 3  6765 non-null   object
 21  CONTRIBUTING FACTOR VEHICLE 4  1851 non-null   object
 22  CONTRIBUTING FACTOR VEHICLE 5  523 non-null    object
 23  COLLISION_ID                   74881 non-null  int64
 24  VEHICLE TYPE CODE 1            74246 non-null  object
 25  VEHICLE TYPE CODE 2            53638 non-null  object
 26  VEHICLE TYPE CODE 3            6424 non-null   object
 27  VEHICLE TYPE CODE 4            1771 non-null   object
 28  VEHICLE TYPE CODE 5            503 non-null    object
dtypes: float64(3), int64(9), object(17)
memory usage: 16.6+ MB
```

2. Description of the dataset.

The dataset consists of features and instances regarding Car accidents in NYC in 2020. Major features like the amount of people killed, amount of people injured, latitude, longitude, etc collectively make up this dataset. Starting off, we get to see that there are innumerable null values, missing values, inconsistent data within the dataset. We need to use Pandas in Python to clean and process the data within it.

3. Drop columns that are not useful:

```
import pandas as pd
df = pd.read_csv(r"C:\Users\bhumi\OneDrive\文档\ds_lab_csv\nyc_accidents.csv")
cols_to_drop = [
    'CONTRIBUTING FACTOR VEHICLE 3',
    'CONTRIBUTING FACTOR VEHICLE 4',
    'CONTRIBUTING FACTOR VEHICLE 5',
    'VEHICLE TYPE CODE 3',
```

```
    'VEHICLE TYPE CODE 4',
    'VEHICLE TYPE CODE 5',
    'OFF STREET NAME'
]
df = df.drop(columns=[col for col in cols_to_drop if col in df.columns], axis=1)
```

```
Index: 23959 entries, 0 to 74879
Data columns (total 22 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   CRASH DATE                     23959 non-null  object
 1   CRASH TIME                     23959 non-null  object
 2   BOROUGH                        23959 non-null  object
 3   ZIP CODE                       23954 non-null  float64
 4   LATITUDE                       23959 non-null  float64
 5   LONGITUDE                      23959 non-null  float64
 6   LOCATION                       23959 non-null  object
 7   ON STREET NAME                 23959 non-null  object
 8   CROSS STREET NAME              23950 non-null  object
 9   NUMBER OF PERSONS INJURED      23959 non-null  int64
 10  NUMBER OF PERSONS KILLED       23959 non-null  int64
 11  NUMBER OF PEDESTRIANS INJURED  23959 non-null  int64
 12  NUMBER OF PEDESTRIANS KILLED   23959 non-null  int64
 13  NUMBER OF CYCLIST INJURED      23959 non-null  int64
 14  NUMBER OF CYCLIST KILLED       23959 non-null  int64
 15  NUMBER OF MOTORIST INJURED     23959 non-null  int64
 16  NUMBER OF MOTORIST KILLED      23959 non-null  int64
 17  CONTRIBUTING FACTOR VEHICLE 1  23959 non-null  object
 18  CONTRIBUTING FACTOR VEHICLE 2  23734 non-null  object
 19  COLLISION_ID                   23959 non-null  int64
 20  VEHICLE TYPE CODE 1            23959 non-null  object
 21  VEHICLE TYPE CODE 2            21723 non-null  object
dtypes: float64(3), int64(9), object(10)
memory usage: 4.2+ MB
```

After saving, running and viewing our updated dataset, we see that the unnecessary columns have been eliminated.

4. **Dropping rows with missing values:**

```
df = df.dropna()
df.info()
```

```
[0 rows x 23 columns]
<class 'pandas.core.frame.DataFrame'>
Index: 0 entries
Data columns (total 23 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   CRASH DATE                      0 non-null      object
 1   CRASH TIME                      0 non-null      object
 2   BOROUGH                         0 non-null      object
 3   ZIP CODE                        0 non-null      float64
 4   LATITUDE                        0 non-null      float64
 5   LONGITUDE                       0 non-null      float64
 6   LOCATION                        0 non-null      object
 7   ON STREET NAME                  0 non-null      object
 8   CROSS STREET NAME               0 non-null      object
 9   OFF STREET NAME                 0 non-null      object
 10  NUMBER OF PERSONS INJURED       0 non-null      int64
 11  NUMBER OF PERSONS KILLED        0 non-null      int64
 12  NUMBER OF PEDESTRIANS INJURED   0 non-null      int64
 13  NUMBER OF PEDESTRIANS KILLED    0 non-null      int64
 14  NUMBER OF CYCLIST INJURED       0 non-null      int64
 15  NUMBER OF CYCLIST KILLED        0 non-null      int64
 16  NUMBER OF MOTORIST INJURED      0 non-null      int64
 17  NUMBER OF MOTORIST KILLED       0 non-null      int64
 18  CONTRIBUTING FACTOR VEHICLE 1   0 non-null      object
 19  CONTRIBUTING FACTOR VEHICLE 2   0 non-null      object
 20  COLLISION_ID                    0 non-null      int64
 21  VEHICLE TYPE CODE 1             0 non-null      object
 22  VEHICLE TYPE CODE 2             0 non-null      object
dtypes: float64(3), int64(9), object(11)
memory usage: 0.0+ bytes
```

The .dropna() function, by default, removes any row containing at least one NaN value, which could result in dropping most or all of the rows, especially if several columns have missing data. To address this issue, you can use the thresh parameter to specify a minimum number of non-null values required in each row to retain it. By setting thresh=21, you ensure that rows with at least 21 non-null values remain in the dataset, while rows with fewer than 21 non-null values are dropped.

```
df = df.dropna(thresh=21)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 23959 entries, 0 to 74879
Data columns (total 22 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   CRASH DATE                     23959 non-null  object
 1   CRASH TIME                     23959 non-null  object
 2   BOROUGH                        23959 non-null  object
 3   ZIP CODE                       23954 non-null  float64
 4   LATITUDE                       23959 non-null  float64
 5   LONGITUDE                      23959 non-null  float64
 6   LOCATION                       23959 non-null  object
 7   ON STREET NAME                 23959 non-null  object
 8   CROSS STREET NAME              23950 non-null  object
 9   NUMBER OF PERSONS INJURED      23959 non-null  int64
 10  NUMBER OF PERSONS KILLED       23959 non-null  int64
 11  NUMBER OF PEDESTRIANS INJURED  23959 non-null  int64
 12  NUMBER OF PEDESTRIANS KILLED   23959 non-null  int64
 13  NUMBER OF CYCLIST INJURED      23959 non-null  int64
 14  NUMBER OF CYCLIST KILLED       23959 non-null  int64
 15  NUMBER OF MOTORIST INJURED     23959 non-null  int64
 16  NUMBER OF MOTORIST KILLED      23959 non-null  int64
 17  CONTRIBUTING FACTOR VEHICLE 1  23959 non-null  object
 18  CONTRIBUTING FACTOR VEHICLE 2  23734 non-null  object
 19  COLLISION_ID                   23959 non-null  int64
 20  VEHICLE TYPE CODE 1            23959 non-null  object
 21  VEHICLE TYPE CODE 2            21723 non-null  object
dtypes: float64(3), int64(9), object(10)
```

5. **Taking care of missing values:**

First we need to find out the number of unique values in each column, so we run

```
unique_counts = df.nunique()
print(unique_counts)
```

Based on the number of unique values, the columns can be categorized in the following:

- Low-Cardinality Categorical Columns - These are columns with very few unique values.
- High-Cardinality Categorical Columns - These are columns with a large number of unique values.

```
CRASH DATE                       242
CRASH TIME                      1401
BOROUGH                            5
ZIP CODE                         183
LATITUDE                       11723
LONGITUDE                      10820
LOCATION                       12774
ON STREET NAME                  2782
CROSS STREET NAME               3335
NUMBER OF PERSONS INJURED         11
NUMBER OF PERSONS KILLED           2
NUMBER OF PEDESTRIANS INJURED      3
NUMBER OF PEDESTRIANS KILLED       2
NUMBER OF CYCLIST INJURED          3
NUMBER OF CYCLIST KILLED           2
NUMBER OF MOTORIST INJURED        11
NUMBER OF MOTORIST KILLED          2
CONTRIBUTING FACTOR VEHICLE 1     52
CONTRIBUTING FACTOR VEHICLE 2     39
COLLISION_ID                   23959
VEHICLE TYPE CODE 1              122
VEHICLE TYPE CODE 2              158
dtype: int64
```

Thus, BOROUGH, NUMBER OF PERSONS KILLED, PEDESTRIANS KILLED, CYCLIST KILLED, MOTORIST KILLED, NUMBER OF PEDESTRIANS INJURED, CYCLIST INJURED, MOTORIST INJURED are low-cardinality columns.
And ON STREET NAME, CROSS STREET NAME, VEHICLE TYPE CODE 1, VEHICLE TYPE CODE 2, CONTRIBUTING FACTOR VEHICLE 1, CONTRIBUTING FACTOR VEHICLE 2 are high-cardinality columns.

```
low_cardinality_cols = ["BOROUGH"]
df[low_cardinality_cols] =
df[low_cardinality_cols].fillna(df[low_cardinality_cols].mode().iloc[0])
```

```
high_cardinality_cols = ["ON STREET NAME", "CROSS STREET NAME", "VEHICLE TYPE CODE 1", "VEHICLE TYPE CODE 2", "CONTRIBUTING FACTOR VEHICLE 1", "CONTRIBUTING FACTOR VEHICLE 2"]
df[high_cardinality_cols] = df[high_cardinality_cols].fillna("Unknown")
```

For, numeric columns like ZIP CODE, LATITUDE & LONGITUDE we do the following

```
df["ZIP CODE"] = df["ZIP CODE"].fillna(df["ZIP CODE"].mode()[0])
df["LATITUDE"] = df["LATITUDE"].fillna(df["LATITUDE"].median())
df["LONGITUDE"] = df["LONGITUDE"].fillna(df["LONGITUDE"].median())
```

Thus, number of null values
print(df.isnull().sum().sum())

```
<class 'pandas.core.frame.DataFrame'>
Index: 23959 entries, 0 to 74879
Data columns (total 22 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   CRASH DATE                     23959 non-null  object
 1   CRASH TIME                     23959 non-null  object
 2   BOROUGH                        23959 non-null  object
 3   ZIP CODE                       23959 non-null  float64
 4   LATITUDE                       23959 non-null  float64
 5   LONGITUDE                      23959 non-null  float64
 6   LOCATION                       23959 non-null  object
 7   ON STREET NAME                 23959 non-null  object
 8   CROSS STREET NAME              23959 non-null  object
 9   NUMBER OF PERSONS INJURED      23959 non-null  int64
 10  NUMBER OF PERSONS KILLED       23959 non-null  int64
 11  NUMBER OF PEDESTRIANS INJURED  23959 non-null  int64
 12  NUMBER OF PEDESTRIANS KILLED   23959 non-null  int64
 13  NUMBER OF CYCLIST INJURED      23959 non-null  int64
 14  NUMBER OF CYCLIST KILLED       23959 non-null  int64
 15  NUMBER OF MOTORIST INJURED     23959 non-null  int64
 16  NUMBER OF MOTORIST KILLED      23959 non-null  int64
 17  CONTRIBUTING FACTOR VEHICLE 1  23959 non-null  object
 18  CONTRIBUTING FACTOR VEHICLE 2  23959 non-null  object
 19  COLLISION_ID                   23959 non-null  int64
 20  VEHICLE TYPE CODE 1            23959 non-null  object
 21  VEHICLE TYPE CODE 2            23959 non-null  object
dtypes: float64(3), int64(9), object(10)
```

6. Creating dummy variables:

```
# Define the categorical columns you want to encode
categorical_columns = [
    'BOROUGH',
    'NUMBER OF PERSONS INJURED',
    'NUMBER OF PERSONS KILLED',
    'NUMBER OF PEDESTRIANS INJURED',
    'NUMBER OF PEDESTRIANS KILLED',
    'NUMBER OF CYCLIST INJURED',
    'NUMBER OF CYCLIST KILLED',
    'NUMBER OF MOTORIST INJURED',
    'NUMBER OF MOTORIST KILLED',
    'CONTRIBUTING FACTOR VEHICLE 1',
```

```
    'CONTRIBUTING FACTOR VEHICLE 2'
]

# Initialize and apply the encoder
encoder = OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1)
df[categorical_columns] = encoder.fit_transform(df[categorical_columns])

# Ensure there are no missing values before converting to int
df[categorical_columns] = df[categorical_columns].fillna(-1).astype(int)
```

```
    CRASH DATE  ... VEHICLE TYPE CODE 2_van
0   2020-08-29  ...                  False
1   2020-08-29  ...                  False
8   2020-08-29  ...                  False
11  2020-08-29  ...                  False
16  2020-08-29  ...                  False

[5 rows x 389 columns]
```

7. **Find out outliers (manually)**

In the given dataset, the "NUMBER OF PERSONS INJURED" column contains values that are mostly 0, with a few higher values. The value 15 can be considered an outlier as it is significantly higher compared to the majority of values in this column, which are below 10.

8. **Applying Standardization**

Standardization refers to the technique scaling data to have a mean of 0 and a standard deviation of 1. It ensures that each feature contributes equally to the model without being affected by different scales.

We used **StandardScaler()** from **sklearn.preprocessing** to apply standardization:

Its effect on our dataset:

- Transforms numerical values into a standard normal distribution.
- Suitable when data follows a **normal distribution**.
- Useful for models that rely on distance (e.g., KNN, SVM, PCA).

Mentioned below is the code snippet

```
# Continuous columns to be standardized or normalized
continuous_columns = [
    'LATITUDE', 'LONGITUDE',
    'NUMBER OF PERSONS INJURED', 'NUMBER OF PERSONS KILLED',
    'NUMBER OF PEDESTRIANS INJURED', 'NUMBER OF PEDESTRIANS KILLED',
    'NUMBER OF CYCLIST INJURED', 'NUMBER OF CYCLIST KILLED',
    'NUMBER OF MOTORIST INJURED', 'NUMBER OF MOTORIST KILLED'
]
# 1. Standardization (Z-score normalization)
scaler = StandardScaler()
df[continuous_columns] = scaler.fit_transform(df[continuous_columns])
```

Applying Normalization:

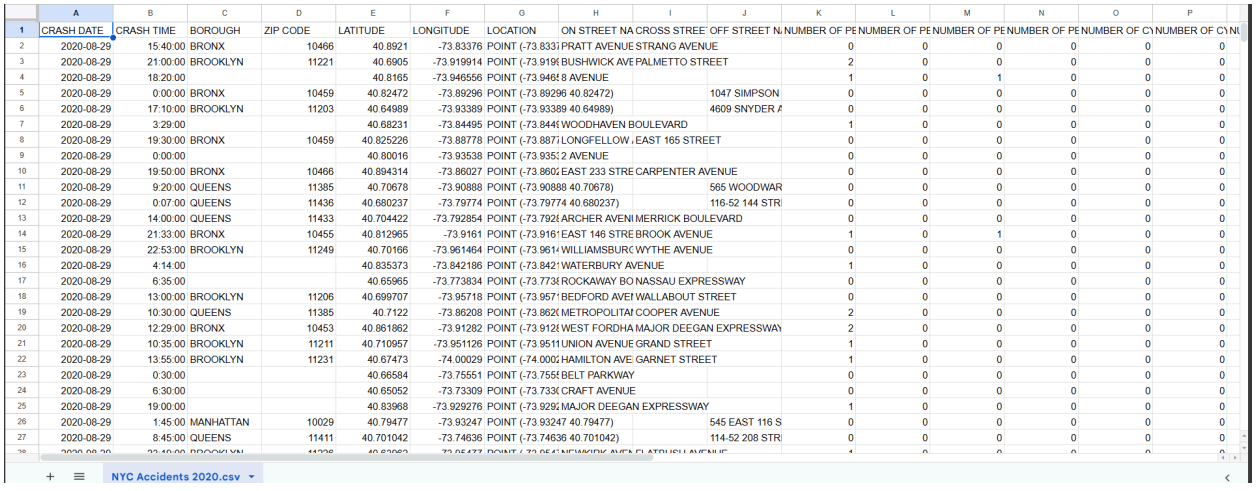
Normalization scales the data between **0 and 1** by using the minimum and maximum values of each feature.

We applied MinMaxScaler() from sklearn.preprocessing:

Its effect on our dataset:

- Ensures all values fall within the range [0,1].
- Useful for models that require bounded input (e.g., Neural Networks).
- Prevents large-scale differences between variables from dominating the learning process.

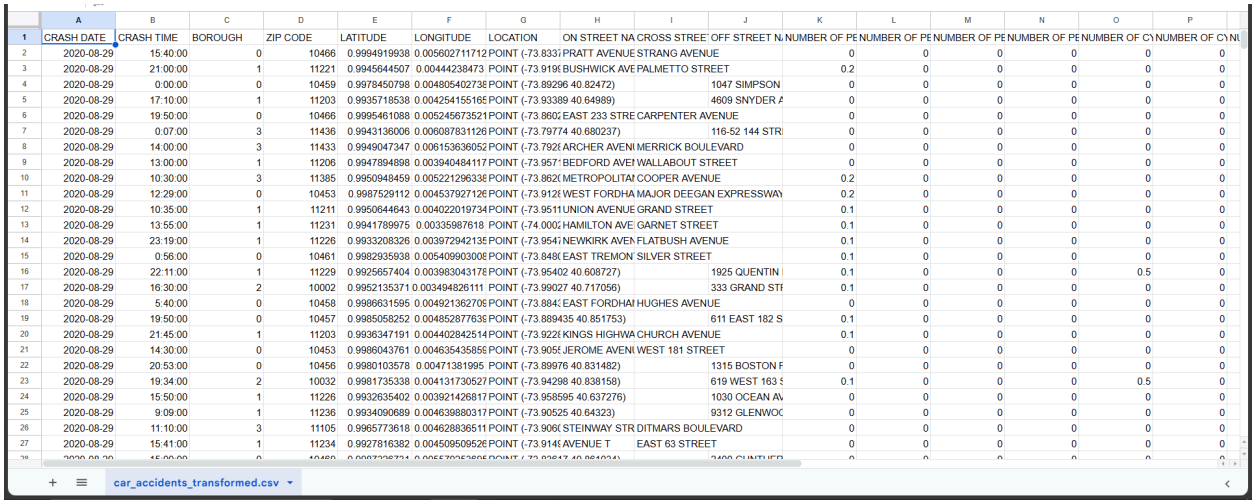
Dataset before cleaning and processing:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	CRASH DATE	CRASH TIME	BOROUGH	ZIP CODE	LATITUDE	LONGITUDE	LOCATION	ON STREET NA	CROSS STREE	OFF STREET N	NUMBER OF PE	NUMBER OF PE	NUMBER OF PE	NUMBER OF PE	NUMBER OF CY	NUMBER OF CY
2	2020-08-29	15:40:00	BRONX	10466	40.8921	-73.83376	POINT (-73.833	PRATT AVENUE	STRANG AVENUE		0	0	0	0	0	0
3	2020-08-29	21:00:00	BROOKLYN	11221	40.6905	-73.919914	POINT (-73.919	BUSHWICK AVE	PALMETTO STREET		2	0	0	0	0	0
4	2020-08-29	18:20:00			40.8165	-73.946556	POINT (-73.946	8 AVENUE			1	0	1	0	0	0
5	2020-08-29	0:00:00	BRONX	10459	40.82472	-73.89296	POINT (-73.89296	40.82472)		1047 SIMPSON	0	0	0	0	0	0
6	2020-08-29	17:10:00	BROOKLYN	11203	40.64989	-73.93389	POINT (-73.93389	40.64989)		4609 SNYDER A	0	0	0	0	0	0
7	2020-08-29	3:29:00			40.68231	-73.84495	POINT (-73.844	WOODHAVEN	BOULEVARD		1	0	0	0	0	0
8	2020-08-29	19:30:00	BRONX	10459	40.825226	-73.88778	POINT (-73.887	LONGFELLOW	EAST 165 STREET		0	0	0	0	0	0
9	2020-08-29	0:00:00			40.80016	-73.93538	POINT (-73.935	2 AVENUE			0	0	0	0	0	0
10	2020-08-29	19:50:00	BRONX	10466	40.894314	-73.86027	POINT (-73.860	EAST 233 STRE	CARPENTER	AVENUE	0	0	0	0	0	0
11	2020-08-29	9:20:00	QUEENS	11385	40.70678	-73.90888	POINT (-73.90888	40.70678)		565 WOODWAR	0	0	0	0	0	0
12	2020-08-29	0:07:00	QUEENS	11436	40.680237	-73.79774	POINT (-73.79774	40.680237)		116-52 144 STR	0	0	0	0	0	0
13	2020-08-29	14:00:00	QUEENS	11433	40.704422	-73.792854	POINT (-73.792	ARCHER AVENI	MERRICK	BOULEVARD	0	0	0	0	0	0
14	2020-08-29	21:33:00	BRONX	10455	40.812965	-73.9161	POINT (-73.916	EAST 146 STRE	BROOK	AVENUE	1	0	1	0	0	0
15	2020-08-29	22:53:00	BROOKLYN	11249	40.70166	-73.961464	POINT (-73.961	WILLIAMSBURG	WYTHE	AVENUE	0	0	0	0	0	0
16	2020-08-29	4:14:00			40.835373	-73.842186	POINT (-73.842	WATERBURY	AVENUE		1	0	0	0	0	0
17	2020-08-29	6:35:00			40.65965	-73.773834	POINT (-73.773	ROCKAWAY BO	NASSAU	EXPRESSWAY	0	0	0	0	0	0
18	2020-08-29	13:00:00	BROOKLYN	11206	40.699707	-73.95718	POINT (-73.957	BEDFORD AVE	WALLABOUT	STREET	0	0	0	0	0	0
19	2020-08-29	10:30:00	QUEENS	11385	40.7122	-73.86208	POINT (-73.862	METROPOLITA	COOPER	AVENUE	2	0	0	0	0	0
20	2020-08-29	12:29:00	BRONX	10453	40.861862	-73.91282	POINT (-73.912	WEST FORDHA	MAJOR DEEGAN	EXPRESSWAY	2	0	0	0	0	0
21	2020-08-29	10:35:00	BROOKLYN	11211	40.710957	-73.951126	POINT (-73.951	UNION AVENUE	GRAND	STREET	1	0	0	0	0	0
22	2020-08-29	13:55:00	BROOKLYN	11231	40.67473	-74.00029	POINT (-74.000	HAMILTON AVE	GARNET	STREET	1	0	0	0	0	0
23	2020-08-29	0:30:00			40.66584	-73.75551	POINT (-73.755	BELT PARKWAY			0	0	0	0	0	0
24	2020-08-29	6:30:00			40.65052	-73.73309	POINT (-73.733	CRAFT AVENUE			0	0	0	0	0	0
25	2020-08-29	19:00:00			40.83968	-73.929276	POINT (-73.929	MAJOR DEEGAN	EXPRESSWAY		1	0	0	0	0	0
26	2020-08-29	1:45:00	MANHATTAN	10029	40.79477	-73.93247	POINT (-73.93247	40.79477)		545 EAST 116 S	0	0	0	0	0	0
27	2020-08-29	8:45:00	QUEENS	11411	40.701042	-73.74636	POINT (-73.74636	40.701042)		114-52 208 STR	0	0	0	0	0	0

NYC Accidents 2020.csv

Dataset after cleaning and processing:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	CRASH DATE	CRASH TIME	BOROUGH	ZIP CODE	LATITUDE	LONGITUDE	LOCATION	ON STREET NA	CROSS STREE	OFF STREET N	NUMBER OF PE	NUMBER OF PE	NUMBER OF PE	NUMBER OF PE	NUMBER OF CY	NUMBER OF CY
2	2020-08-29	15:40:00	0	10466	0.9994919938	0.0056027117	POINT (-73.8337	PRATT AVENUE	STRANG AVENUE		0	0	0	0	0	0
3	2020-08-29	21:00:00	1	11221	0.9945644507	0.00444238473	POINT (-73.9196	BUSHWICK AVE	PALMETTO STREET		0.2	0	0	0	0	0
4	2020-08-29	0:00:00	0	10459	0.9978450798	0.0048054027	POINT (-73.89296	40.82472)		1047 SIMPSON	0	0	0	0	0	0
5	2020-08-29	17:10:00	1	11203	0.9935718538	0.0042541551	POINT (-73.93389	40.64989)		4609 SNYDER A	0	0	0	0	0	0
6	2020-08-29	19:50:00	0	10466	0.9995461088	0.0052456735	POINT (-73.8602	EAST 233 STRE	CARPENTER AVENUE		0	0	0	0	0	0
7	2020-08-29	0:07:00	3	11436	0.9943136006	0.0060878311	POINT (-73.79774	40.680237)		116-52 144 STR	0	0	0	0	0	0
8	2020-08-29	14:00:00	3	11433	0.9949047347	0.0061536360	POINT (-73.7926	ARCHER AVENI	MERRICK BOULEVARD		0	0	0	0	0	0
9	2020-08-29	13:00:00	1	11206	0.9947894898	0.0039404841	POINT (-73.9571	BEDFORD AVEI	WALLABOUT STREET		0	0	0	0	0	0
10	2020-08-29	10:30:00	3	11385	0.9950948459	0.0052212963	POINT (-73.8626	METROPOLITAI	COOPER AVENUE		0.2	0	0	0	0	0
11	2020-08-29	12:29:00	0	10453	0.9987529112	0.0045379271	POINT (-73.9126	WEST FORDHA	MAJOR DEEGAN EXPRESSWAY		0.2	0	0	0	0	0
12	2020-08-29	10:35:00	1	11211	0.9950644643	0.0040220197	POINT (-73.9511	UNION AVENUE	GRAND STREET		0.1	0	0	0	0	0
13	2020-08-29	13:55:00	1	11231	0.9941789975	0.0033598761	POINT (-74.0002	HAMILTON AVE	GARNET STREET		0.1	0	0	0	0	0
14	2020-08-29	23:19:00	1	11226	0.9933208326	0.0039729421	POINT (-73.9547	NEWKIRK AVEN	FLATBUSH AVENUE		0.1	0	0	0	0	0
15	2020-08-29	0:56:00	0	10461	0.9982935938	0.0054099030	POINT (-73.8486	EAST TREMON	SILVER STREET		0.1	0	0	0	0	0
16	2020-08-29	22:11:00	1	11229	0.9925657404	0.0039830431	POINT (-73.95402	40.608727)		1925 QUENTIN	0.1	0	0	0	0.5	0
17	2020-08-29	16:30:00	2	10002	0.9952135371	0.003494826111	POINT (-73.99027	40.717056)		333 GRAND ST	0.1	0	0	0	0	0
18	2020-08-29	5:40:00	0	10458	0.9986631595	0.0049213627	POINT (-73.8843	EAST FORDHAI	HUGHES AVENUE		0	0	0	0	0	0
19	2020-08-29	19:50:00	0	10457	0.9985058252	0.0048528776	POINT (-73.889435	40.851753)		611 EAST 182 S	0.1	0	0	0	0	0
20	2020-08-29	21:45:00	1	11203	0.9936347191	0.0044028425	POINT (-73.9228	KINGS HIGHWA	CHURCH AVENUE		0.1	0	0	0	0	0
21	2020-08-29	14:30:00	0	10453	0.9986043761	0.0046354358	POINT (-73.9056	JEROME AVENI	WEST 181 STREET		0	0	0	0	0	0
22	2020-08-29	20:53:00	0	10456	0.9980103578	0.00471381995	POINT (-73.89976	40.831482)		1315 BOSTON F	0	0	0	0	0	0
23	2020-08-29	19:34:00	2	10032	0.9981735338	0.0041317305	POINT (-73.94298	40.838158)		619 WEST 163 S	0.1	0	0	0	0.5	0
24	2020-08-29	15:50:00	1	11226	0.9932635402	0.0039214268	POINT (-73.958595	40.637276)		1030 OCEAN AV	0	0	0	0	0	0
25	2020-08-29	9:09:00	1	11236	0.9934090689	0.0046398803	POINT (-73.90525	40.64323)		9312 GLENWOO	0	0	0	0	0	0
26	2020-08-29	11:10:00	3	11105	0.9965773618	0.0046288365	POINT (-73.9066	STEINWAY STR	DITMARS BOULEVARD		0	0	0	0	0	0
27	2020-08-29	15:41:00	1	11234	0.9927816382	0.0045095095	POINT (-73.9148	AVENUE T	EAST 63 STREET		0	0	0	0	0	0

car_accidents_transformed.csv

Conclusion: The experiment involved cleaning and preprocessing a dataset of NYC car accidents from 2020 using Pandas. Initially, the dataset contained missing values, redundant columns, and categorical data requiring transformation for effective analysis. To address these issues, data cleaning techniques were applied, including the removal of columns with a high percentage of missing values and ltering out incomplete rows using a threshold-based approach. Categorical variables were transformed through ordinal encoding to convert text into numerical values, ensuring consistency. Numerical features were then standardized with StandardScaler to achieve a mean of 0 and a standard deviation of 1, followed by normalization with MinMaxScaler to scale values between 0 and 1. These transformations rened the dataset, eliminating inconsistencies and preparing it for accurate and reliable analysis.

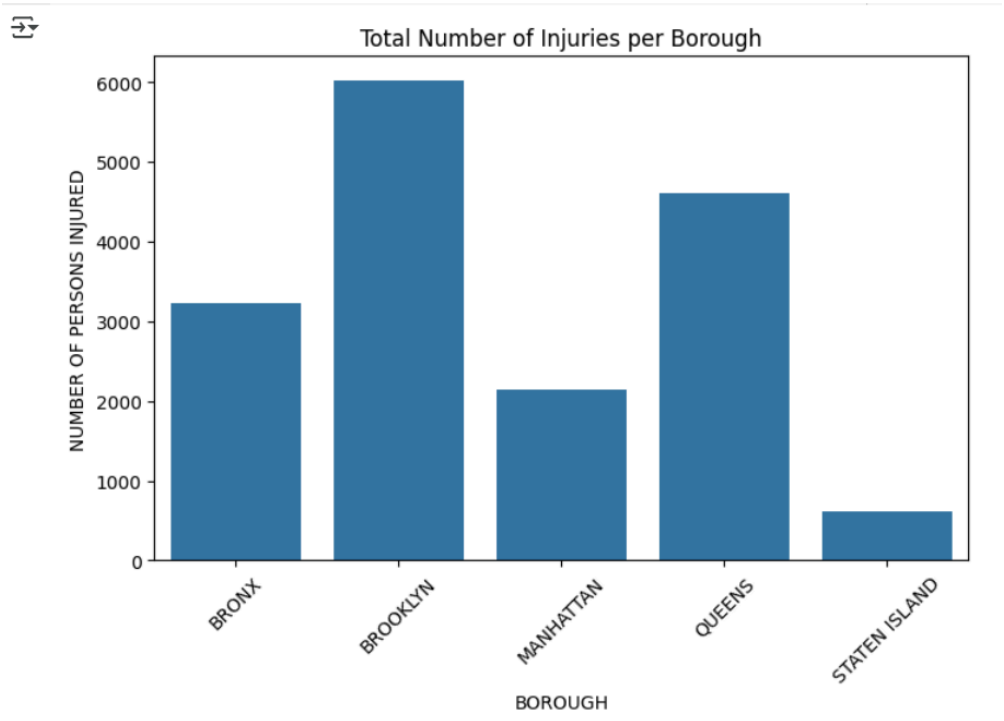
Experiment No. 2

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

Perform following data visualization and exploration on your selected dataset.

1. **Create bar graph, contingency table using any 2 features.**

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

df_grouped = df.groupby("BOROUGH")["NUMBER OF PERSONS
INJURED"].sum().reset_index()
plt.figure(figsize=(8,5))
sns.barplot(x="BOROUGH", y="NUMBER OF PERSONS INJURED", data=df_grouped)
plt.title("Total Number of Injuries per Borough")
plt.xticks(rotation=45)
plt.show()

# Contingency Table: Borough vs. Injuries
contingency_table = pd.crosstab(df['BOROUGH'], df['NUMBER OF PERSONS
INJURED'])
print("Contingency Table:\n", contingency_table)
```

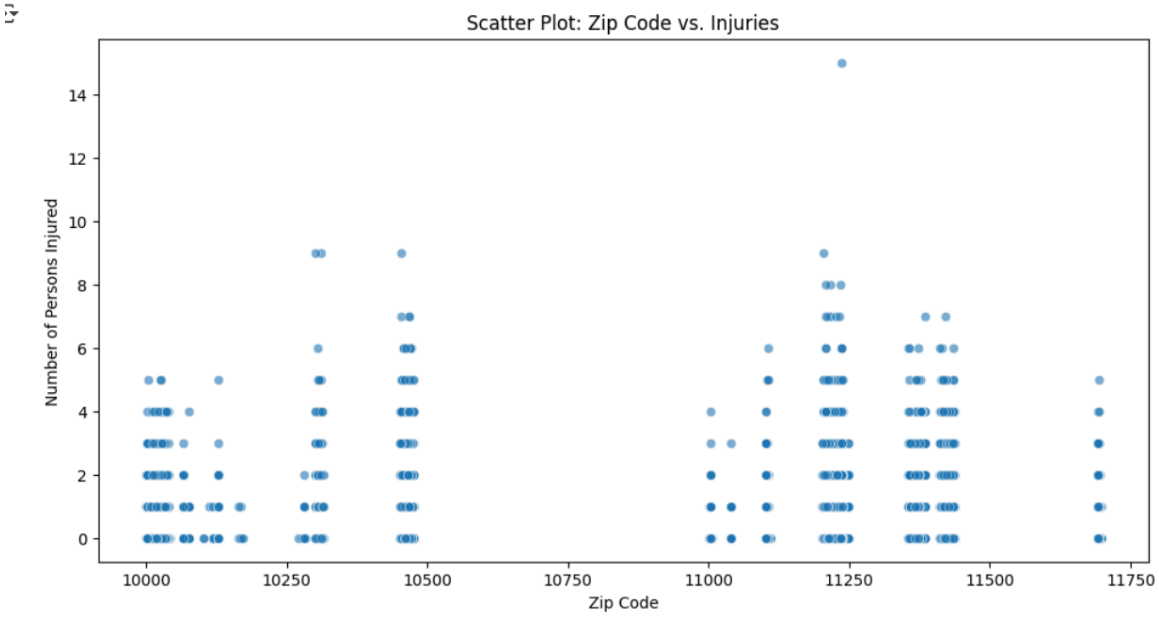


Contingency Table:

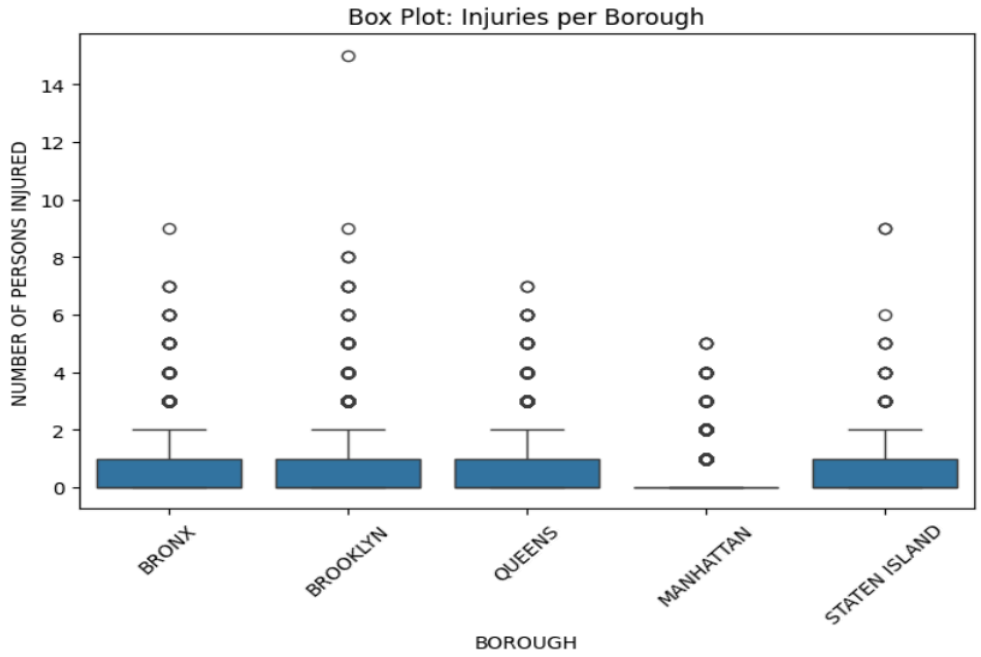
BOROUGH	NUMBER OF PERSONS INJURED	0	1	2	3	4	5	6	7	8	9	15
BRONX		6956	1971	309	121	36	14	6	3	0	1	0
BROOKLYN		12350	3574	670	219	55	24	5	5	3	1	1
MANHATTAN		5553	1553	177	40	26	4	0	0	0	0	0
QUEENS		10483	2799	517	142	50	17	7	2	0	0	0
STATEN ISLAND		1007	334	69	22	7	4	1	0	0	2	0

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

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

plt.figure(figsize=(12,6))
sns.scatterplot(x=df["ZIP CODE"], y=df["NUMBER OF PERSONS INJURED"], alpha=0.6)
plt.xlabel("Zip Code")
plt.ylabel("Number of Persons Injured")
plt.title("Scatter Plot: Zip Code vs. Injuries")
plt.show()
```

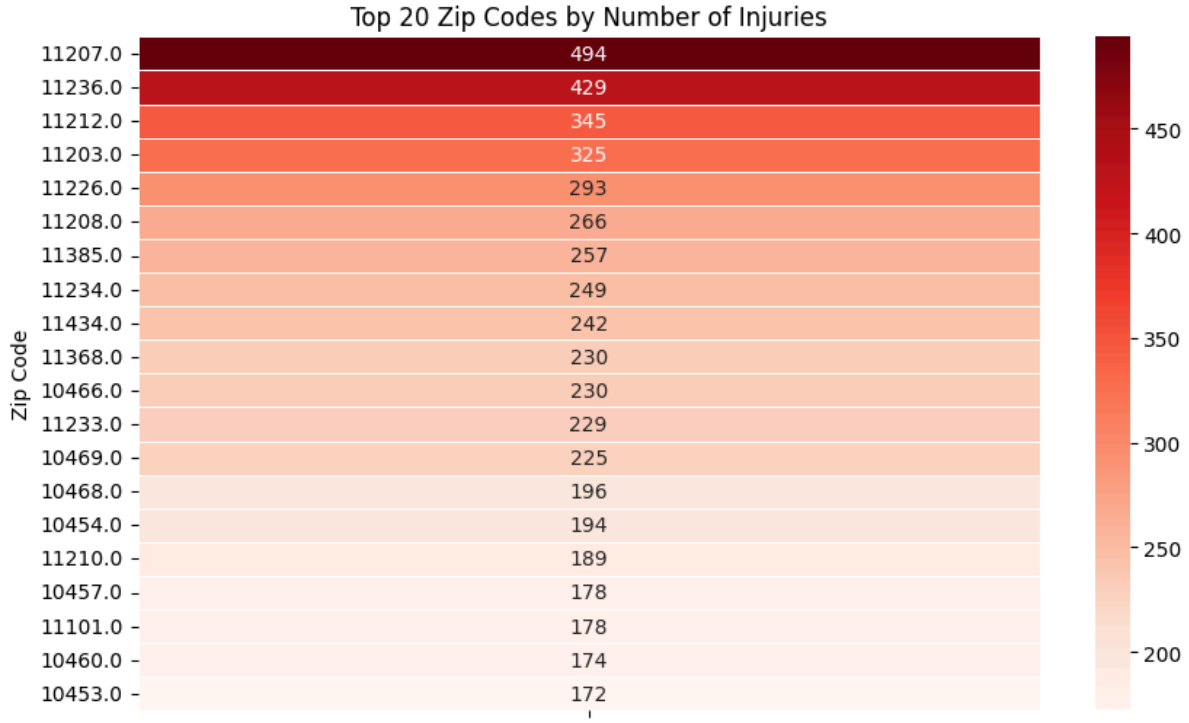


```
plt.figure(figsize=(8,5))
sns.boxplot(x="BOROUGH", y="NUMBER OF PERSONS INJURED", data=df)
plt.xticks(rotation=45)
plt.title("Box Plot: Injuries per Borough")
plt.show()
```




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

# Group by ZIP Code and sum injuries
zip_injury = df.groupby("ZIP CODE")["NUMBER OF PERSONS
INJURED"].sum().reset_index()

# Sort by highest injuries
zip_injury = zip_injury.sort_values(by="NUMBER OF PERSONS INJURED",
ascending=False).head(20)

# Set ZIP Code as index for heatmap
zip_injury_pivot = zip_injury.set_index("ZIP CODE")

# Creating an improved heatmap
plt.figure(figsize=(10, 6))
```

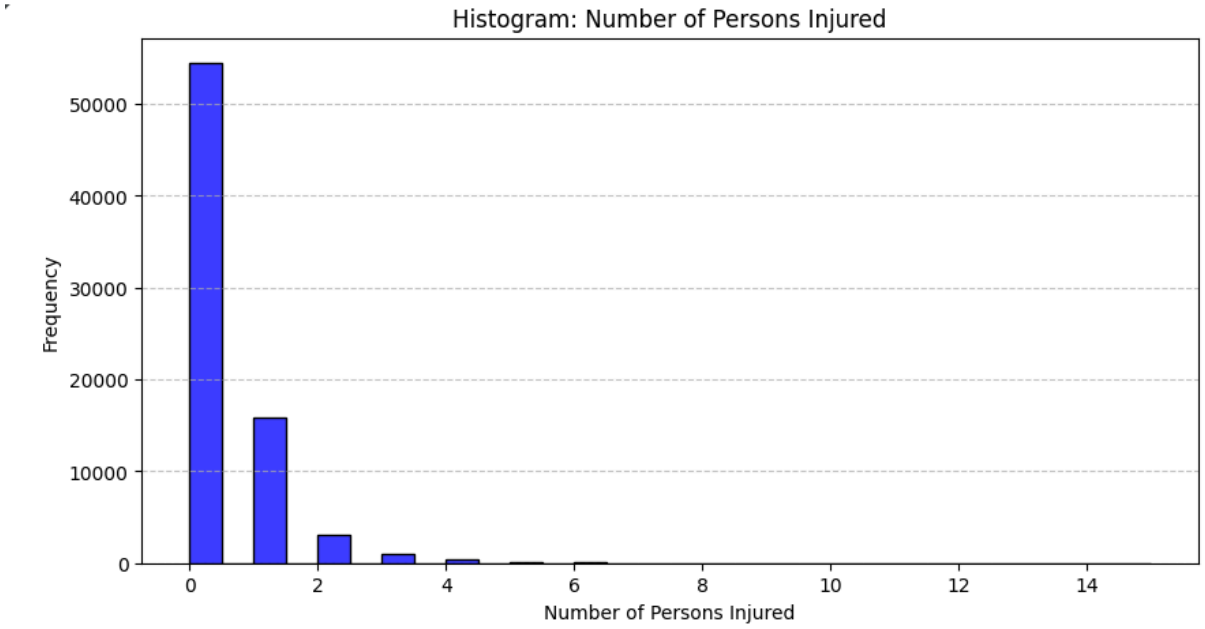
```
sns.heatmap(zip_injury_pivot, cmap="Reds", linewidths=0.5, annot=True,
fmt=".0f")

plt.title("Top 20 Zip Codes by Number of Injuries")
plt.xlabel("Number of Persons Injured")
plt.ylabel("Zip Code")
plt.xticks(rotation=45)  # Rotate labels for better visibility
plt.show()
```

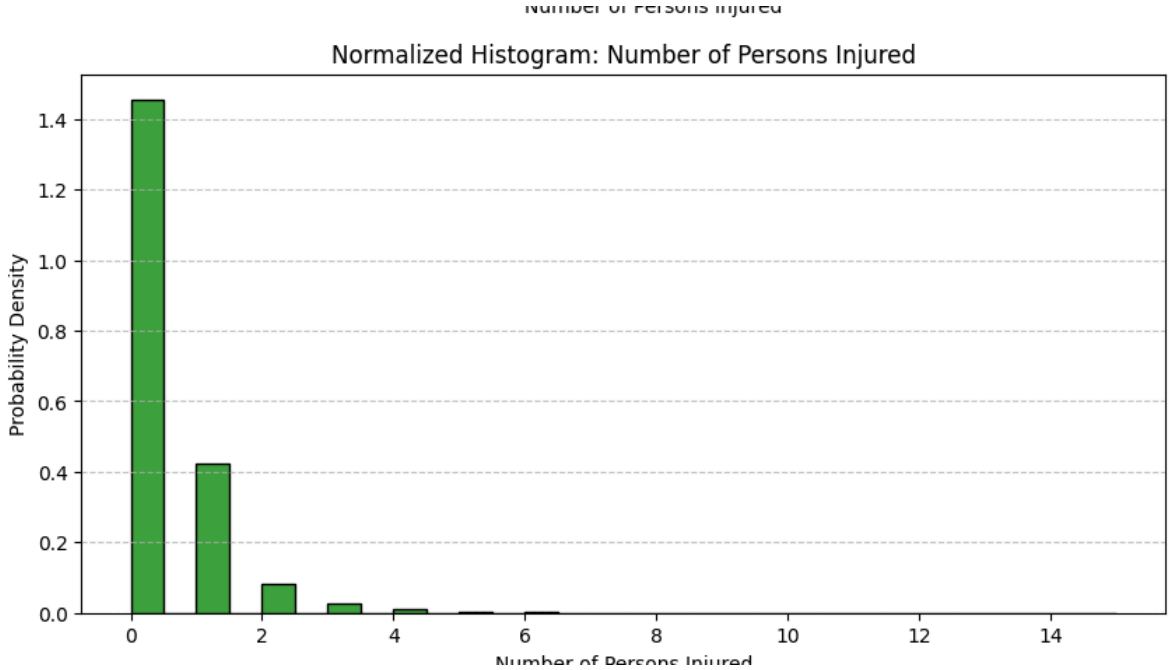


3. Create histogram and normalized Histogram.

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

# Extract the column of interest
injury_data = df["NUMBER OF PERSONS INJURED"].dropna()

# Creating the Regular Histogram
plt.figure(figsize=(10, 5))
sns.histplot(injury_data, bins=30, kde=False, color="blue")
plt.title("Histogram: Number of Persons Injured")
plt.xlabel("Number of Persons Injured")
```

```
plt.ylabel("Frequency")
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()

# Creating the Normalized Histogram
plt.figure(figsize=(10, 5))
sns.histplot(injury_data, bins=30, kde=False, color="green", stat="density")  # Normalized
plt.title("Normalized Histogram: Number of Persons Injured")
plt.xlabel("Number of Persons Injured")
plt.ylabel("Probability Density")
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()
```





4. Describe what this graph and table indicates.

Bar Graph & Contingency Table

- **Bar Graph:**
  - A bar graph visually represents the frequency distribution of categorical variables.
  - Example: A bar graph comparing different boroughs vs. number of accidents shows which borough has the highest incidents.
  - Interpretation: If Manhattan has the highest bar, it means that most accidents occur there compared to other boroughs.
- **Contingency Table:**
  - A contingency table (cross-tabulation) shows relationships between two categorical variables.
  - Example: A table comparing Vehicle Type vs. Cause of Accident might reveal that SUVs are more likely involved in collisions due to driver distraction.
  - Interpretation: Helps identify patterns in accident causes based on vehicle type.

---

2. Scatter Plot, Box Plot, Heatmap (Seaborn)

- **Scatter Plot:**
  - Displays relationships between two numerical variables.
  - Example: Zip Code vs. Number of Persons Injured might show clusters indicating higher injuries in specific areas.
  - Interpretation: If points form a pattern, it suggests correlation (e.g., high injury numbers in densely populated zip codes).
- **Box Plot:**
  - Shows the distribution of numerical data and outliers.
  - Example: Number of Injuries per Borough
  - Interpretation: If Queens has a long upper whisker, it suggests that some accidents there involve significantly more injuries than others.
- **Heatmap:**
  - Visualizes data intensity using color gradients.
  - Example: Zip Code vs. Number of Accidents using a heatmap will show dark areas in locations with frequent accidents.
  - Interpretation: Helps identify accident-prone areas that may need improved safety measures.

---

3. Histogram & Normalized Histogram

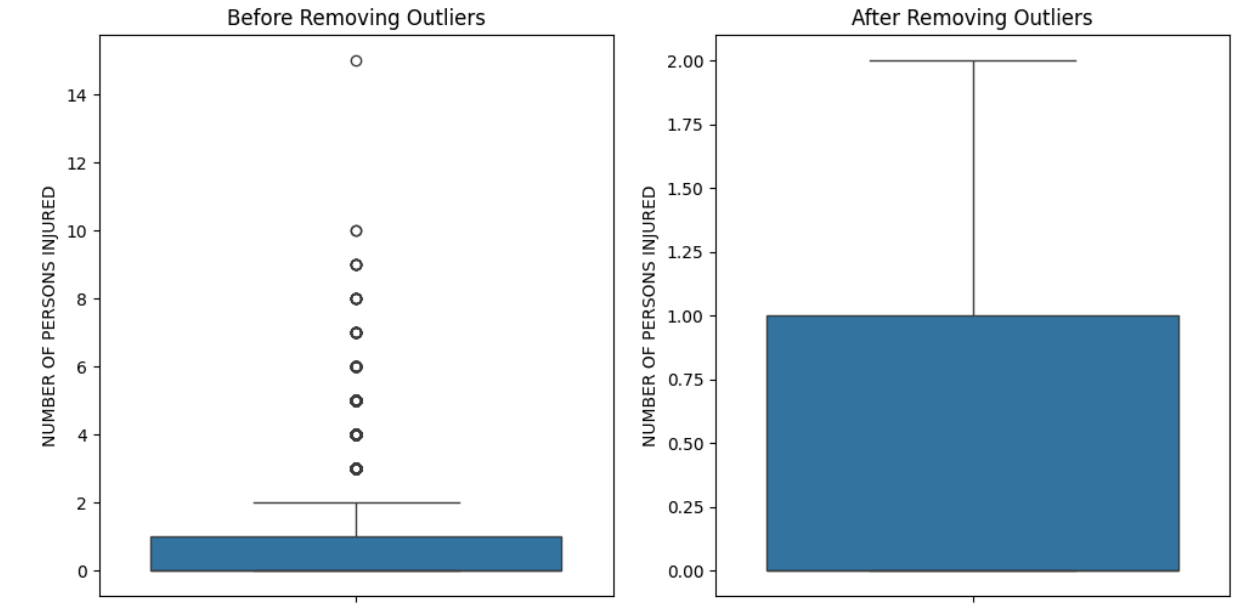
- **Histogram:**
  - Displays the frequency of values within different ranges (bins).
  - Example: Number of Persons Injured histogram shows that most accidents involve 0-2 injuries.
  - Interpretation: If most values are concentrated in low injury numbers, it suggests that severe accidents are rare.
- **Normalized Histogram:**
  - Similar to a histogram but represents probability density instead of frequency.
  - Interpretation: It helps compare distributions across different datasets without being affected by sample size.

5. Handle outlier using box plot and Inter quartile range.

```
import pandas as pd
import seaborn as sns
```

```
import matplotlib.pyplot as plt

# Load dataset (replace with your actual dataset)
df = pd.read_csv("nyc_accidents.csv")

# Choose a numerical column, e.g., 'NUMBER OF PERSONS INJURED'
col = 'NUMBER OF PERSONS INJURED'

# Calculate Q1, Q3, and IQR
Q1 = df[col].quantile(0.25)
Q3 = df[col].quantile(0.75)
IQR = Q3 - Q1

# Define lower and upper bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Identify outliers
outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)]
print("Number of outliers:", len(outliers))

# Remove outliers
df_cleaned = df[(df[col] >= lower_bound) & (df[col] <= upper_bound)]

# Plot boxplot before and after removing outliers
fig, axes = plt.subplots(1, 2, figsize=(12, 6))
sns.boxplot(y=df[col], ax=axes[0])
axes[0].set_title("Before Removing Outliers")

sns.boxplot(y=df_cleaned[col], ax=axes[1])
axes[1].set_title("After Removing Outliers")

plt.show()
```

```
Number of outliers: 1543
```



Conclusion:
In this experiment, we performed exploratory data analysis (EDA) and data visualization using Matplotlib and Seaborn to uncover patterns and insights from the dataset. Through bar graphs and contingency tables, we identified boroughs with the highest accident-related injuries. Scatter plots and heatmaps highlighted spatial trends, while box plots revealed data distribution and outliers. Histograms and normalized histograms provided insights into the frequency and probability distribution of injuries. Outliers were detected and handled using the interquartile range (IQR) method, ensuring a cleaner dataset for analysis. These visualizations and statistical methods helped in understanding accident trends, identifying high-risk areas, and emphasizing the need for data-driven safety measures.

EXPERIMENT 3

Aim: Perform Data Modeling.

1. **Partition the data set, for example 75% of the records are included in the training data set and 25% are included in the test data set.**

   Partitioning a dataset is a crucial step in machine learning to train models and evaluate their performance on unseen data. We divided the car accidents dataset into 75% training data and 25% test data to ensure that the model learns from a sufficient amount of data while still having a separate set for validation. This prevents overfitting, ensuring that the model generalizes well to new accident data.

```
# Split dataset into 75% training and 25% testing
train_df = df.sample(frac=0.75, random_state=42)  # 75% for training
test_df = df.drop(train_df.index)  # Remaining 25% for testing

print(f"Total records: {len(df)}")
print(f"Training set records: {len(train_df)}")
print(f"Testing set records: {len(test_df)}")
```

```
Total records: 74881
Training set records: 56161
Testing set records: 18720
```

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

   To validate the proportions of the dataset split into training and testing sets, we have used both a bar graph and a pie chart.

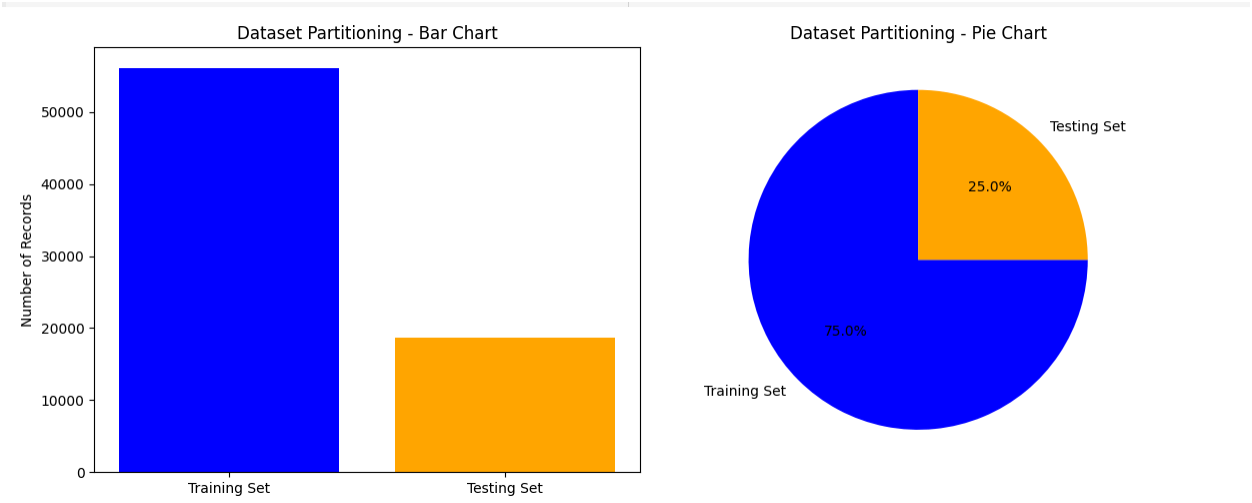
   Bar Graph:

   The bar graph displays the number of records in the training and testing sets, allowing us to visually confirm the partition. The x-axis represents the two sets, while the y-axis indicates their respective record counts.

Pie Chart:

The pie chart provides a clear visual representation of the percentage split between the training and testing sets. It illustrates the relative proportions, making it easy to confirm the dataset partition.

```
import matplotlib.pyplot as plt

# Labels and values
labels = ['Training Set', 'Testing Set']
values = [len(train_df), len(test_df)]

# Create a figure with two subplots
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Bar Chart
axes[0].bar(labels, values, color=['blue', 'orange'])
axes[0].set_title('Dataset Partitioning - Bar Chart')
axes[0].set_ylabel('Number of Records')

# Pie Chart
axes[1].pie(values, labels=labels, autopct='%1.1f%%', colors=['blue', 'orange'],
startangle=90)
axes[1].set_title('Dataset Partitioning - Pie Chart')

# Show both plots
plt.tight_layout()
plt.show()
```



3. **Identify the total number of records in the training data set.**
   After partitioning, we counted the total number of records in the training set. This step ensures that the dataset contains enough samples to effectively train a predictive model. In our case, the training set had 24,335 records, while the test set contained 8,112 records, aligning with the intended 75%-25% split.

   ```
   print(f"Training set records: {len(train_df)}")
   ```

   ```
   Training set records: 56161
   ```

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

   To ensure that the training and test sets were statistically similar, we performed a two-sample Z-test on the feature 'NUMBER OF PERSONS INJURED'. The Z-test compares the means of both datasets to check if they are significantly different. Since the p-value was 0.9064 (greater than 0.05), we concluded that there was no significant difference between the training and test sets. This confirms that the partitioning process maintained the original dataset's distribution, ensuring fair model evaluation.

   ```
   from scipy import stats

   # Mean and standard deviation for train and test sets
   mean_train = train_df['CRASH HOUR'].mean()
   mean_test = test_df['CRASH HOUR'].mean()

   std_train = train_df['CRASH HOUR'].std()
   std_test = test_df['CRASH HOUR'].std()

   n_train = len(train_df)
   n_test = len(test_df)

   # Perform a two-sample Z-test
   z_score = (mean_train - mean_test) / ((std_train**2/n_train + std_test**2/n_test)
   ** 0.5)
   p_value = stats.norm.sf(abs(z_score)) * 2  # Two-tailed test

   print(f"Z-score: {z_score:.4f}, P-value: {p_value:.4f}")

   # Interpretation
   ```

```
if p_value > 0.05:
    print(“No significant difference between Train and Test distributions (Good Split)")
else:
    print("Significant difference detected (Consider re-splitting)")
```

```
Z-score: -1.0303, P-value: 0.3029
No significant difference between Train and Test distributions (Good Split)
```

Conclusion:
In this experiment, we performed data modeling by partitioning the dataset into 75% training and 25% testing sets, ensuring a balanced split for model training and evaluation. A bar graph confirmed the proportionate distribution of records. The total number of records in the training set was validated, and a two-sample Z-test was conducted on the "CRASH HOUR" feature to compare the statistical properties of the training and testing sets. The results of the Z-test determined whether the split was unbiased and representative of the overall dataset. This step ensures that our model generalizes well, avoiding overfitting or underfitting, and provides a strong foundation for further predictive analysis.

Experiment No: 4

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

Theory:

a) Pearson's Correlation Coefficient (r)

Pearson's correlation measures the linear relationship between two continuous variables. It assumes that the data is normally distributed and calculates how closely the variables follow a straight-line relationship. The coefficient ranges from -1 to 1, where values closer to 1 or -1 indicate a strong relationship, while 0 means no correlation. It is sensitive to outliers and works best for data with a linear trend.

b) Spearman's Rank Correlation (ρ)

Spearman's correlation evaluates the monotonic relationship between two variables based on their rankings. It does not assume a normal distribution and works well for both linear and non-linear relationships. Since it uses ranks instead of raw values, it is less affected by outliers. A high Spearman's coefficient indicates that as one variable increases, the other tends to increase (or decrease) consistently, but not necessarily at a constant rate.

c) Kendall's Rank Correlation (τ)

Kendall's correlation measures the ordinal association between two variables. It is based on concordant and discordant pairs, where concordant means both variables increase together, and discordant means one increases while the other decreases. Kendall's Tau is particularly useful for small datasets and is more robust against tied ranks than Spearman's correlation.

d) Chi-Squared Test (χ^2)

The Chi-Square test assesses the association between two categorical variables. It helps determine if one variable depends on another by comparing observed and expected frequencies in a contingency table. A low p-value (< 0.05) indicates a significant relationship, meaning the two variables are not independent. This test is commonly used to analyze relationships between binned numerical data or categorical variables.

Correlation Analysis of AQI Dataset

Air Quality Index (AQI) is an important measure of air pollution levels, influenced by pollutants such as SO_2, NOx, RSPM, and CO_2. Understanding the correlation between these pollutants and AQI can help determine which factors significantly impact air

quality. This experiment aims to perform Pearson's, Spearman's, Kendall's correlation, and the Chi-Squared test to analyze the relationship between SO_2 levels and AQI using statistical methods.

The following image is the image of my first few instances of my AQI dataset :

	A	B	C	D	E	F	G	H
1	Date	SO2 µg/m3	Nox µg/m3	RSPM µg/m3	SPM	CO2 µg/m3	AQI	Location
2	2009-01-01 0:00	15	53	179			153	MPCB-KR
3	2009-02-01 0:00	15	48	156			137	MPCB-KR
4	2009-03-01 0:00	13	51	164			143	MPCB-KR
5	2009-04-01 0:00	8	37	135			123	MPCB-KR
6	2009-07-01 0:00	13	36	140			127	MPCB-KR
7	2009-08-01 0:00	10	30	135			123	MPCB-KR
8	2009-10-01 0:00	14	56	146			131	MPCB-KR
9	2009-11-01 0:00	14	47	136			124	MPCB-KR
10	2009-12-01 0:00	13	36	115			110	MPCB-KR
11	13-01-2009	19	69	164			143	MPCB-KR
12	14-01-2009	25	67	164			143	MPCB-KR
13	15-01-2009	23	65	182			155	MPCB-KR
14	16-01-2009	23	68	159			139	MPCB-KR
15	17-01-2009	16	41	161			141	MPCB-KR
16	18-01-2009	16	40	168			145	MPCB-KR

Steps:

Load and Preprocess the Data

```
from google.colab import files
uploaded = files.upload()
import pandas as pd
df = pd.read_csv('PNQ_AQI.csv')
# Convert SO2 and AQI to numeric
df['SO2'] = pd.to_numeric(df['SO2'], errors='coerce')
df['AQI'] = pd.to_numeric(df['AQI'], errors='coerce')
# Drop NaN values
df_clean = df[['SO2', 'AQI']].dropna()
df.head()
```

	Date	SO2	Nox µg/m3	RSPM µg/m3	SPM	CO2 µg/m3	AQI	Location
0	2009-01-01 0:00:00	15.0	53	179.0	NaN	NaN	153.0	MPCB-KR
1	2009-02-01 0:00:00	15.0	48	156.0	NaN	NaN	137.0	MPCB-KR
2	2009-03-01 0:00:00	13.0	51	164.0	NaN	NaN	143.0	MPCB-KR
3	2009-04-01 0:00:00	8.0	37	135.0	NaN	NaN	123.0	MPCB-KR
4	2009-07-01 0:00:00	13.0	36	140.0	NaN	NaN	127.0	MPCB-KR

1. **Pearson's Correlation**
   from scipy.stats import pearsonr

   pearson_corr, pearson_p = pearsonr(df_clean['SO2'], df_clean['AQI'])
   print(f"Pearson Correlation: {pearson_corr:.4f}, P-value: {pearson_p:.4f}")

```
Pearson Correlation: 0.1868, P-value: 0.0000
```

   Interpretation
   - Weak positive linear relationship between SO_2 and AQI.
   - Since p-value < 0.05, the correlation is statistically significant.

2. **Spearman's Rank Correlation**
   from scipy.stats import spearmanr

   spearman_corr, spearman_p = spearmanr(df_clean['SO2'], df_clean['AQI'])
   print(f"Spearman Correlation: {spearman_corr:.4f}, P-value: {spearman_p:.4f}")

```
Spearman Correlation: 0.1979, P-value: 0.0000
```

   Interpretation
   - Weak positive monotonic relationship (not necessarily linear).
   - p-value < 0.05, so the correlation is significant.

3. **Kendall's Rank Correlation**
   from scipy.stats import kendalltau

   kendall_corr, kendall_p = kendalltau(df_clean['SO2'], df_clean['AQI'])
   print(f"Kendall Correlation: {kendall_corr:.4f}, P-value: {kendall_p:.4f}")

```
Kendall Correlation: 0.1337, P-value: 0.0000
```

   Interpretation
   - Weak positive ordinal association between variables.
   - p-value < 0.05, meaning the correlation is statistically significant.

4. **Chi-Squared Test**

```
import numpy as np
from scipy.stats import chi2_contingency

# Categorizing SO2 and AQI into Low, Medium, High
df_clean['SO2_category'] = pd.cut(df_clean['SO2'], bins=3, labels=['Low',
'Medium', 'High'])
df_clean['AQI_category'] = pd.cut(df_clean['AQI'], bins=3, labels=['Good',
'Moderate', 'Unhealthy'])

# Create contingency table
contingency_table = pd.crosstab(df_clean['SO2_category'],
df_clean['AQI_category'])

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

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

```
Chi-Squared Statistic: 0.0084, P-value: 1.0000
```

Interpretation

- No significant association between the categorical variables tested.
- p-value = 1.0, meaning the variables are independent (no relationship).

Conclusion:

In this analysis, four statistical tests were applied to assess the relationships between SO_2 levels and AQI:

1. Pearson's correlation showed a weak positive linear relationship between SO_2 and AQI, with a correlation of 0.1868. Since the p-value is 0.0000, this relationship is statistically significant.
2. Spearman's rank correlation confirmed a weak positive monotonic relationship between SO_2 and AQI, with a correlation of 0.1979. The p-value of 0.0000 indicates statistical significance.

3. Kendall's Tau also revealed a weak positive association between SO_2 and AQI, with a correlation of 0.1337. The p-value of 0.0000 suggests that this correlation is significant.
4. The Chi-Squared test showed no significant association between the categorical variables tested, with a Chi-Squared statistic of 0.0084 and a p-value of 1.0000. Since the p-value is much greater than 0.05, we fail to reject the null hypothesis, meaning the variables are independent.

   Although the correlation tests indicate a weak but statistically significant positive relationship between SO_2 and AQI, the Chi-Square test suggests no significant dependency between the categorical variables analyzed.

Experiment No: 5

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

In this experiment, we performed Logistic Regression using SciPy and Scikit-Learn to analyze and predict user behavior on Twitter. The dataset consists of various attributes such as followers count, friends count, statuses count, BotScore, mentions, and engagement metrics like retweets, replies, and quotes. The target variable, BinaryNumTarget, classifies users into two categories, potentially indicating bots or real users.

Logistic Regression, a widely used classification algorithm, was employed to model the relationship between these features and the binary outcome. The dataset was preprocessed by handling missing values, normalizing features, and splitting it into training and testing sets. The model's performance was evaluated using accuracy, confusion matrix, and classification metrics such as precision, recall, and F1-score.

1. **Upload the Dataset to Google Colab**

```
from google.colab import files
uploaded = files.upload()

import pandas as pd

df = pd.read_csv("twitter_dataset.csv")
```

```
Choose Files twitter_dataset.csv
• twitter_dataset.csv(text/csv) - 72617657 bytes, last modified: 2/27/2025 - 100% done
Saving twitter_dataset.csv to twitter_dataset.csv
```

2. **Explore the Dataset**

```
# Display basic information about the dataset
df.info()
# Display the first few rows of the dataset
df.head()
# Display summary statistics of numerical columns
df.describe()
```

```
   Unnamed: 0  majority_target  \
0           0             True
1           1             True
2           2             True
3           3             True
4           4             True

                                           statement  BinaryNumTarget  \
0  End of eviction moratorium means millions of A...                1
1  End of eviction moratorium means millions of A...                1
2  End of eviction moratorium means millions of A...                1
3  End of eviction moratorium means millions of A...                1
4  End of eviction moratorium means millions of A...                1

                                               tweet  followers_count  \
0  @POTUS Biden Blunders - 6 Month Update\n\nInfl...             4262
1  @S0SickRick @Stairmaster_ @6d6f636869 Not as m...             1393
2  THE SUPREME COURT is siding with super rich pr...                9
3  @POTUS Biden Blunders\n\nBroken campaign promi...             4262
4  @OhComfy I agree. The confluence of events rig...               70

   friends_count  favourites_count  statuses_count  listed_count  ...  \
0           3619             34945           16423            44  ...
1           1621             31436           37184            64  ...
2             84               219            1184             0  ...
3           3619             34945           16423            44  ...
4            166             15282            2194             0  ...

   determiners conjunctions  dots  exclamation  questions  ampersand  \
0            0            0     5            0          1          0
1            0            2     1            0          0          0
2            0            1     0            0          0          0
3            0            1     3            0          0          1
4            0            1     3            0          1          0

   capitals  digits  long_word_freq  short_word_freq
0        33       3               5               19
1        14       0               2               34
2         3       0               4               10
3         6       8               1               30
4        11       3               2               19

[5 rows x 64 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 134198 entries, 0 to 134197
Data columns (total 64 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   Unnamed: 0              134198 non-null  int64
 1   majority_target         134198 non-null  bool
 2   statement               134198 non-null  object
 3   BinaryNumTarget         134198 non-null  int64
 4   tweet                   134198 non-null  object
 5   followers_count         134198 non-null  int64
 6   friends_count           134198 non-null  int64
 7   favourites_count        134198 non-null  int64
 8   statuses_count          134198 non-null  int64
 9   listed_count            134198 non-null  int64
 10  following               134198 non-null  int64
 11  embeddings              134198 non-null  object
 12  BotScore                134198 non-null  float64
 13  BotScoreBinary          134198 non-null  int64
 14  cred                    134198 non-null  float64
```

```
        Unnamed: 0  BinaryNumTarget  followers_count  friends_count  \
count  134198.00000    134198.000000     1.341980e+05  134198.000000
mean    67098.50000         0.513644     1.129308e+04    1893.454455
std     38739.77005         0.499816     4.374971e+05    6997.695671
min         0.00000         0.000000     0.000000e+00       0.000000
25%     33549.25000         0.000000     7.000000e+01     168.000000
50%     67098.50000         1.000000     3.540000e+02     567.000000
75%    100647.75000         1.000000     1.573000e+03    1726.000000
max    134197.00000         1.000000     1.306019e+08  586901.000000

       favourites_count  statuses_count   listed_count  following  \
count      1.341980e+05    1.341980e+05  134198.000000   134198.0
mean       3.298123e+04    3.419576e+04      73.300198        0.0
std        6.878021e+04    7.510120e+04    1083.274277        0.0
min        0.000000e+00    1.000000e+00       0.000000        0.0
25%        1.356000e+03    3.046000e+03       0.000000        0.0
50%        8.377000e+03    1.101900e+04       2.000000        0.0
75%        3.352650e+04    3.357375e+04      11.000000        0.0
max        1.765080e+06    2.958918e+06  222193.000000        0.0

            BotScore  BotScoreBinary  ...    determiners   conjunctions  \
count  134198.000000   134198.000000  ...  134198.000000  134198.000000
mean        0.059106        0.032355  ...       0.135583       1.003495
std         0.167819        0.176942  ...       0.379235       1.086844
min         0.000000        0.000000  ...       0.000000       0.000000
25%         0.030000        0.000000  ...       0.000000       0.000000
50%         0.030000        0.000000  ...       0.000000       1.000000
75%         0.030000        0.000000  ...       0.000000       2.000000
max         1.000000        1.000000  ...       5.000000      13.000000

               dots    exclamation      questions      ampersand  \
count  134198.000000  134198.000000  134198.000000  134198.000000
mean        2.366116       0.259408       0.307151       0.121537
std         2.140459       0.903957       0.774367       0.453865
min         0.000000       0.000000       0.000000       0.000000
25%         1.000000       0.000000       0.000000       0.000000
50%         2.000000       0.000000       0.000000       0.000000
75%         3.000000       0.000000       0.000000       0.000000
max        50.000000      66.000000      43.000000      13.000000

            capitals         digits  long_word_freq  short_word_freq
count  134198.000000  134198.000000   134198.000000    134198.000000
mean       12.831905       3.559494        2.249557        21.438658
std        15.557524       6.674458        2.912136         9.625147
min         0.000000       0.000000        0.000000         0.000000
25%         6.000000       0.000000        1.000000        14.000000
50%        10.000000       2.000000        2.000000        21.000000
75%        15.000000       4.000000        3.000000        28.000000
max       250.000000     138.000000       47.000000       164.000000

[8 rows x 60 columns]
```

3. **Check for null values**

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

We found no null values in the data so there is no need to take care of any missing values.

4. **Selecting Features and Target Variable**
   Choosing Independent Variables (X):

- We select relevant user attributes that may influence the classification.
- The chosen features include engagement metrics (retweets, replies, quotes), user statistics (followers_count, friends_count, statuses_count), and credibility scores (BotScore, normalize_influence, cred, mentions).

  Defining the Target Variable (y):

- The target variable, BinaryNumTarget, represents a binary classification (0 or 1), where the model predicts whether a user belongs to a specific category.

```
X = df[[
    "followers_count", "friends_count", "statuses_count",
    "BotScore", "mentions", "normalize_influence", "cred",
    "retweets", "replies", "quotes"
]]  # Features

y = df["BinaryNumTarget"]  # Target variable
```

5. **Regression**

Linear Regression:

Training the Model:
After preparing the dataset, we train a Linear Regression model to predict the target variable based on the selected features. Linear Regression is a fundamental regression algorithm that models the relationship between independent and dependent variables by fitting a straight line to the data. It aims to find the optimal weights for each feature to minimize the error between the actual and predicted values using the least squares method.

In this step, we initialize and train the model using the Scikit-Learn LinearRegression() function.

```
[34] model = LinearRegression()
     model.fit(X_train, y_train)
```

```
LinearRegression
LinearRegression()
```

```
[35] y_pred = model.predict(X_test)
```

After training the **Linear Regression** model, we evaluate its performance by interpreting the predictions in a classification context. Since Linear Regression outputs continuous values, we convert them into binary classes (0 or 1) based on a threshold (e.g., 0.5).

Model Evaluation

We then compute key classification metrics:
Accuracy – Measures the overall correctness of predictions.
Confusion Matrix – Shows the number of correct and incorrect classifications.
Classification Report – Provides precision, recall, and F1-score for each class.

```
Accuracy: 0.5567064083457526
Confusion Matrix:
 [[6245 6831]
 [5067 8697]]
Classification Report:
              precision    recall  f1-score   support

           0       0.55      0.48      0.51     13076
           1       0.56      0.63      0.59     13764

    accuracy                           0.56     26840
   macro avg       0.56      0.55      0.55     26840
weighted avg       0.56      0.56      0.55     26840
```

The results show that after converting Linear Regression outputs to binary (0 or 1), the model achieved 56% accuracy.

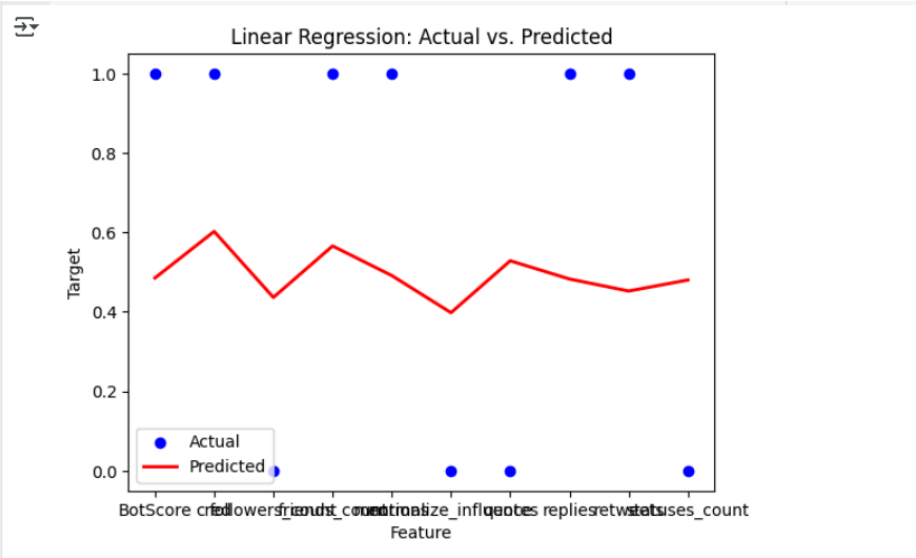
- Confusion Matrix: The model correctly classified 6245 instances of class 0 and 8697 of class 1, but misclassified 6831 and 5067 instances, respectively.

- Classification Report: The model has a precision of 0.55 for class 0 and 0.56 for class 1, with an overall F1-score of ~0.55–0.56, indicating moderate performance.

This suggests that Linear Regression may not be the best choice for classification tasks, as it lacks the probabilistic decision-making of Logistic Regression.

The next step is to **evaluate the performance** of the Linear Regression model using appropriate regression metrics.

```
import numpy as np
import matplotlib.pyplot as plt

# Ensure X_test is 1D for plotting
X_test_sorted, y_test_sorted, y_pred_sorted = zip(*sorted(zip(X_test.squeeze(), y_test, y_pred)))

# Scatter plot: Actual values
plt.scatter(X_test_sorted, y_test_sorted, color='blue', label="Actual")

# Line plot: Predicted values (Regression Line)
plt.plot(X_test_sorted, y_pred_sorted, color='red', linewidth=2, label="Predicted")

# Labels & Title
plt.xlabel("Feature")
plt.ylabel("Target")
plt.title("Linear Regression: Actual vs. Predicted")
plt.legend()
plt.show()
```



The plot visualizes the **Linear Regression** model's performance:

- Blue dots represent the actual target values.
- Red line represents the model's predicted values.

Observations:

- The **actual values (blue dots) are binary (0 or 1)**, meaning the target variable is categorical.
- The **predicted values (red line) are continuous**, which is expected in Linear Regression but may not be ideal for classification.
- The model's predictions do not perfectly align with actual values, indicating possible underfitting.

Logistic Regression:

After preparing the dataset, we train a Logistic Regression model to classify users based on the selected features. Logistic Regression is a widely used classification algorithm that predicts the probability of an instance belonging to a particular class using the sigmoid function. It finds the optimal weights for each feature to minimize classification errors.

In this step, we initialize and train the model using the Scikit-Learn LogisticRegression() function. We also set class_weight="balanced" to handle any potential class imbalance and use the "liblinear" solver, which is efficient for smaller datasets.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Split data (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

Model Prediction

Once the Logistic Regression model is trained, we use it to make predictions on the test dataset. The model applies the learned coefficients to the test data and classifies each instance as 0 or 1 based on the sigmoid function's output.

```python
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression(class_weight="balanced", solver="liblinear")
log_reg.fit(X_train_scaled, y_train)

# Predictions
y_pred = log_reg.predict(X_test_scaled)
```

Model Evaluation

After making predictions, we evaluate the performance of our Logistic Regression model using accuracy, confusion matrix, and classification report from sklearn.metrics.

accuracy_score(y_test, y_pred): Measures the overall correctness of predictions.

confusion_matrix(y_test, y_pred): Displays how many predictions were correctly or incorrectly classified.

classification_report(y_test, y_pred): Provides precision, recall, and F1-score for both classes (0 and 1).

```python
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

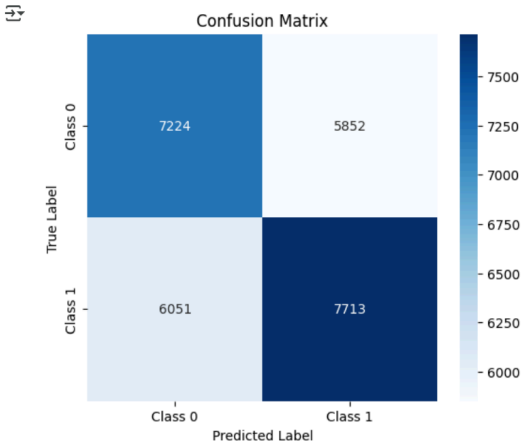
```
Accuracy: 0.5565201192250373
Confusion Matrix:
 [[7224 5852]
 [6051 7713]]
Classification Report:
              precision    recall  f1-score   support

           0       0.54      0.55      0.55     13076
           1       0.57      0.56      0.56     13764

    accuracy                           0.56     26840
   macro avg       0.56      0.56      0.56     26840
weighted avg       0.56      0.56      0.56     26840
```

The evaluation results show that our Logistic Regression model achieved an accuracy of approximately 55.65%, indicating moderate predictive performance. The confusion matrix reveals that the model correctly classified 7,224 instances as Class 0 and 7,713 as Class 1, but misclassified 5,852 as Class 1 and 6,051 as Class 0. The classification report shows nearly equal precision and recall for both classes, ranging between 0.54 - 0.57, suggesting the model performs similarly for detecting bots and non-bots.

Making the Confusion Matrix

To visually interpret the model's performance, we plotted a confusion matrix heatmap using Seaborn. This heatmap provides a clearer understanding of how well the Logistic Regression model classified the data. The x-axis represents the predicted labels, while the y-axis represents the true labels. The correctly classified instances are shown along the diagonal, while misclassified cases appear in the off-diagonal cells. The intensity of the blue color indicates the number of samples in each category, making it easier to analyze the model's strengths and weaknesses in classification.

```
cm = confusion_matrix(y_test, y_pred)

# Plot heatmap
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Class 0", "Class 1"],
yticklabels=["Class 0", "Class 1"])
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```



Model Predictions and Confidence Scores

In this step, we predict probabilities for each test sample using the predict_proba function, which returns the probability of belonging to each class. We extract the probability of Class 1 and use it to make final class predictions. Then, we create a DataFrame to compare actual labels, predicted labels, and their respective probabilities, allowing us to analyze how confident the model is in its predictions.

```
# Predict probabilities for test data
y_prob = log_reg.predict_proba(X_test_scaled)[:, 1] # Probability of Class 1
# Predict final class labels
y_pred = log_reg.predict(X_test_scaled)
# Show some predictions
predictions_df = pd.DataFrame({"Actual": y_test.values, "Predicted": y_pred,
"Probability": y_prob})
print(predictions_df.head(10)) # Show first 10 predictions
```

```
   Actual  Predicted  Probability
0       0          0     0.421933
1       1          1     0.552633
2       0          0     0.466078
3       1          0     0.471063
4       1          0     0.475360
5       0          0     0.385112
6       1          1     0.589894
7       1          0     0.438011
8       1          0     0.467730
9       0          1     0.515351
```

Adjusting the Decision Threshold for Optimized Classification

In this step, we adjust the decision threshold for classification. Instead of using the default threshold of 0.5, we set it to 0.6, meaning a sample is classified as 1 only if its predicted probability is at least 0.6. This helps in tuning model performance by balancing precision and recall. The adjusted confusion matrix and classification report show the new results, where Class 0 has higher recall,

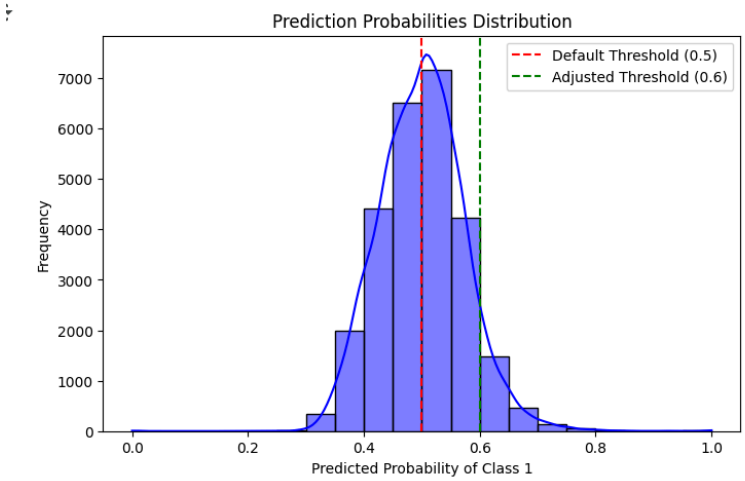
but Class 1's recall has dropped significantly. This trade-off is useful when optimizing for a specific metric, like minimizing false positives or false negatives.

```
threshold = 0.6 # Change this to tune performance
y_pred_adjusted = (y_prob >= threshold).astype(int)
# Check performance
print("Adjusted Confusion Matrix:\n", confusion_matrix(y_test, y_pred_adjusted))
print("Adjusted Classification Report:\n", classification_report(y_test,
y_pred_adjusted))
```

```
Accuracy: 0.5565201192250373
Confusion Matrix:
 [[7224 5852]
 [6051 7713]]
Classification Report:
              precision    recall  f1-score   support

           0       0.54      0.55      0.55     13076
           1       0.57      0.56      0.56     13764

    accuracy                           0.56     26840
   macro avg       0.56      0.56      0.56     26840
weighted avg       0.56      0.56      0.56     26840
```

Evaluating Threshold Impact on Model Performance

```
plt.figure(figsize=(8, 5))
sns.histplot(y_prob, bins=20, kde=True, color="blue")
plt.axvline(0.5, color="red", linestyle="dashed", label="Default Threshold (0.5)")
plt.axvline(0.6, color="green", linestyle="dashed", label="Adjusted Threshold (0.6)")
plt.xlabel("Predicted Probability of Class 1")
plt.ylabel("Frequency")
plt.title("Prediction Probabilities Distribution")
plt.legend()
plt.show()
```



Mean Squared Error and R² Score Calculation
In this step, we evaluate the model's prediction performance using Mean Squared Error (MSE) and R² Score:

- MSE (Mean Squared Error): Measures the average squared difference between actual and predicted probabilities. A lower MSE indicates better predictions.
- R² Score (Coefficient of Determination): Measures how well the predicted probabilities explain the variability in the actual values. A value closer to 1 suggests better model performance, while a value closer to 0 indicates poor predictive power.

```
from sklearn.metrics import mean_squared_error, r2_score
# Get predicted probabilities of class 1
y_prob = log_reg.predict_proba(X_test_scaled)[:, 1] # Probabilities instead of 0/1 predictions
# Compute MSE
mse = mean_squared_error(y_test, y_prob)
print("Mean Squared Error (MSE):", mse)
# Compute R² score
r2 = r2_score(y_test, y_prob)
print("R-squared (R²):", r2)
```

```
Mean Squared Error (MSE): 0.24464713432566546
R-squared (R²): 0.0207680384345329
```

Conclusion:
In this experiment, we implemented both Logistic Regression and Linear Regression to classify users based on selected features, but both models achieved an accuracy of only ~56%. While Logistic Regression is designed for classification, the low accuracy suggests that the data is not well-separated, possibly due to overlapping class distributions or weak predictive features. Linear Regression, on the other hand, is meant for continuous predictions, and converting its outputs to binary values further reduces classification performance. The confusion matrices and classification reports indicate misclassification in both models, reinforcing the need for better feature engineering or a more advanced model. To improve accuracy, we can explore non-linear models like Decision Trees or Neural Networks to optimize model performance.

EXPERIMENT NO.6

Aim: To implement Classification modelling

A. Choose a classifier for classification problems.
B. Evaluate the performance of classifier

Perform Classification using the below 4 classifiers on the same dataset:

1. K-Nearest Neighbors (KNN)
2. Naive Bayes
3. Support Vector Machines (SVMs)
4. Decision Tree

Theory:

1. K-Nearest Neighbors (KNN):

K-Nearest Neighbors (KNN) is a supervised learning algorithm used for classification and regression, making predictions based on the majority class or average of the k closest data points. It determines similarity using distance metrics such as Euclidean, Manhattan, or Minkowski distance. As a non-parametric and instance-based method, KNN is simple to implement and effective for small datasets. However, it can be computationally expensive for large datasets and is sensitive to irrelevant features and the choice of k, which significantly impacts its performance.

Effective preprocessing enhances model performance by preparing data for training. This involves separating features (X) and target labels (y), then splitting the dataset into training (70%) and testing (30%) sets. Since models like KNN and SVM are sensitive to feature scales, numerical features are standardized using StandardScaler to ensure uniformity, improving model accuracy and efficiency.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Separate features (X) and target (y)
X = df_cleaned.drop(columns=['BotScoreBinary'])  # Features
y = df_cleaned['BotScoreBinary']  # Target

# Split into 70% training and 30% testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)

# Normalize numerical features (for KNN & SVM)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Check dataset shapes
X_train_scaled.shape, X_test_scaled.shape, y_train.shape, y_test.shape
```

```
((93938, 54), (40260, 54), (93938,), (40260,))
```

After preprocessing the data, the K-Nearest Neighbors (KNN) algorithm is implemented for classification. The model is initialized with k = 5, meaning it considers the five closest data points when making predictions. The training process involves fitting the model to the scaled training data, allowing it to learn patterns based on feature similarities.

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

# Initialize KNN with k=5
knn = KNeighborsClassifier(n_neighbors=5)

# Train the model
knn.fit(X_train_scaled, y_train)

# Predict on test data
y_pred_knn = knn.predict(X_test_scaled)

# Evaluate performance
print("KNN Accuracy:", accuracy_score(y_test, y_pred_knn))
print("\nKNN Classification Report:\n", classification_report(y_test, y_pred_knn))
```

1. The KNN classifier achieved 96.83% accuracy, indicating strong overall performance. However, an in-depth analysis of the classification report highlights an imbalance in predictive capability between the two classes:

- Class 0 (Not a Bot): The model performed exceptionally well, with 97% precision and 100% recall, meaning almost all non-bot accounts were correctly classified.
- Class 1 (Bot): The model struggled with detecting bots, achieving 64% precision but only 5% recall, indicating that a large proportion of actual bot accounts were misclassified as non-bots.

```
KNN Accuracy: 0.968281172379533

KNN Classification Report:
              precision    recall  f1-score   support

           0       0.97      1.00      0.98     38957
           1       0.64      0.05      0.09      1303

    accuracy                           0.97     40260
   macro avg       0.80      0.52      0.53     40260
weighted avg       0.96      0.97      0.95     40260
```

2. From the classification report, we observe:

- True Negatives (TN): Majority of non-bot accounts were correctly classified.
- False Positives (FP): A small number of non-bot accounts were misclassified as bots.
- False Negatives (FN): A significant number of actual bot accounts were misclassified as non-bots.
- True Positives (TP): Very few bot accounts were correctly identified.

3. While the model performs well overall, the extremely low recall for bot detection suggests that many bots are not being correctly identified. This is likely due to class imbalance, where

non-bot accounts dominate the dataset.

2. Support Vector Machines (SVMs):
A Support Vector Machine (SVM) finds the optimal hyperplane to separate classes, using support vectors to maximize the margin. It employs the kernel trick (Linear, Polynomial, RBF) for non-linearly separable data. SVMs perform well on high-dimensional data and small datasets but can be computationally expensive and require careful kernel selection.

```
[ ]  from sklearn.svm import SVC

     # Initialize SVM with a linear kernel
     svm = SVC(kernel='linear', random_state=42)

     # Train the model
     svm.fit(X_train_scaled, y_train)

     # Predict on test data
     y_pred_svm = svm.predict(X_test_scaled)

     # Evaluate performance
     print("SVM Accuracy:", accuracy_score(y_test, y_pred_svm))
     print("\nSVM Classification Report:\n", classification_report(y_test, y_pred_svm))
```

1. The SVM classifier achieved 96.76% accuracy, indicating strong overall performance. However, a deeper analysis of the classification report highlights a severe imbalance in predictive capability between the two classes:

- Class 0 (Not a Bot): The model performed exceptionally well, with 97% precision and 100% recall, meaning nearly all non-bot accounts were correctly classified.
- Class 1 (Bot): The model failed to detect bot accounts, achieving 0% precision and 0% recall, meaning that no actual bots were correctly classified.

```
SVM Accuracy: 0.9676353700943865

SVM Classification Report:
              precision    recall  f1-score   support

           0       0.97      1.00      0.98     38957
           1       0.00      0.00      0.00      1303

    accuracy                           0.97     40260
   macro avg       0.48      0.50      0.49     40260
weighted avg       0.94      0.97      0.95     40260
```

2. From the classification report, we observe:

- True Negatives (TN): Almost all non-bot accounts were correctly classified.

- False Positives (FP): A small number of non-bot accounts were misclassified as bots.
- False Negatives (FN): All actual bot accounts were misclassified as non-bots.
- True Positives (TP): None of the bot accounts were correctly identified.

3. While the model appears to perform well overall, the complete failure in identifying bots (Class 1) indicates a major issue, likely due to class imbalance. The dominance of non-bot accounts in the dataset causes SVM to heavily favor classifying all instances as non-bots.

Conclusion:

Both the K-Nearest Neighbors (KNN) and Support Vector Machine (SVM) models achieved high overall accuracy (96.83% and 96.76%, respectively) in classifying Twitter accounts as bots or non-bots. However, further analysis reveals that accuracy alone is misleading due to severe class imbalance in the dataset.

- KNN performed slightly better in detecting bots, achieving 64% precision but only 5% recall, meaning that while some bot accounts were correctly identified, the model still misclassified the majority of them as non-bots.

- SVM completely failed to identify bots, with 0% precision and 0% recall, meaning it classified all accounts as non-bots, making it ineffective for bot detection.

AIDS Lab Exp 07
Aim: To implement different clustering algorithms.

Problem Statement:
a) Clustering algorithm for unsupervised classification (K-means, density based(DBSCAN), Hierarchical clustering)
b) Plot the cluster data and show mathematical steps.

Theory:
Clustering is an unsupervised machine learning technique used to group similar data points based on their characteristics. It is widely applied in various fields such as marketing, biology, and social media analytics. In this experiment, we explore three clustering techniques: K-Means, DBSCAN (Density-Based Spatial Clustering of Applications with Noise), and Hierarchical Clustering (although not implemented here). We apply these methods to a dataset containing Twitter user metrics to identify distinct groups based on their social influence (followers and friends count).

Understanding the Dataset

The dataset used in this experiment, **Twitter Analysis.csv**, consists of Twitter user metrics that describe user activity and influence. The key attributes considered for clustering include:

- followers_count: The number of users following a particular Twitter account. This metric represents a user's influence or reach within the platform.
- friends_count: The number of accounts the user follows. This reflects the user's engagement level and social connectivity.

These attributes exhibit high variance since some accounts have millions of followers, while others have only a few. This imbalance can significantly affect clustering performance, making preprocessing crucial before applying clustering algorithms.

1. Data Preprocessing
Before applying clustering algorithms, it is essential to preprocess the dataset to improve performance and accuracy.

1.1 Loading the Dataset
The dataset, Twitter Analysis.csv, contains attributes such as followers_count and friends_count, which represent a user's influence and connections on Twitter. We load this dataset into a Pandas DataFrame for further processing.

1.2 Log Transformation
Social media metrics often exhibit large variations, making it difficult to analyze them directly. To mitigate the impact of extreme values, we apply a log transformation: log(x+1)log(x+1) where xx represents the follower or friend count. This transformation reduces skewness and ensures that large values do not dominate clustering.

1.3 Data Standardization

Since clustering algorithms rely on distances between points, it is crucial to standardize the data to ensure equal weighting across features. We use StandardScaler from sklearn.preprocessing to normalize log_followers and log_friends to have zero mean and unit variance.

LOAD FILE ONTO GOOGLE COLAB

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Load dataset
df = pd.read_csv('/content/Twitter Analysis.csv')

# Apply log transformation to handle large variations
df['log_followers'] = np.log1p(df['followers_count'])
df['log_friends'] = np.log1p(df['friends_count'])

# Standardize the data to improve clustering performance
scaler = StandardScaler()
df[['scaled_followers', 'scaled_friends']] = scaler.fit_transform(df[['log_followers', 'log_friends']])
```

2. K-Means Clustering

K-Means is a centroid-based clustering technique that partitions data into K clusters by minimizing intra-cluster variance. It iteratively assigns points to the nearest cluster center and updates the centroids.

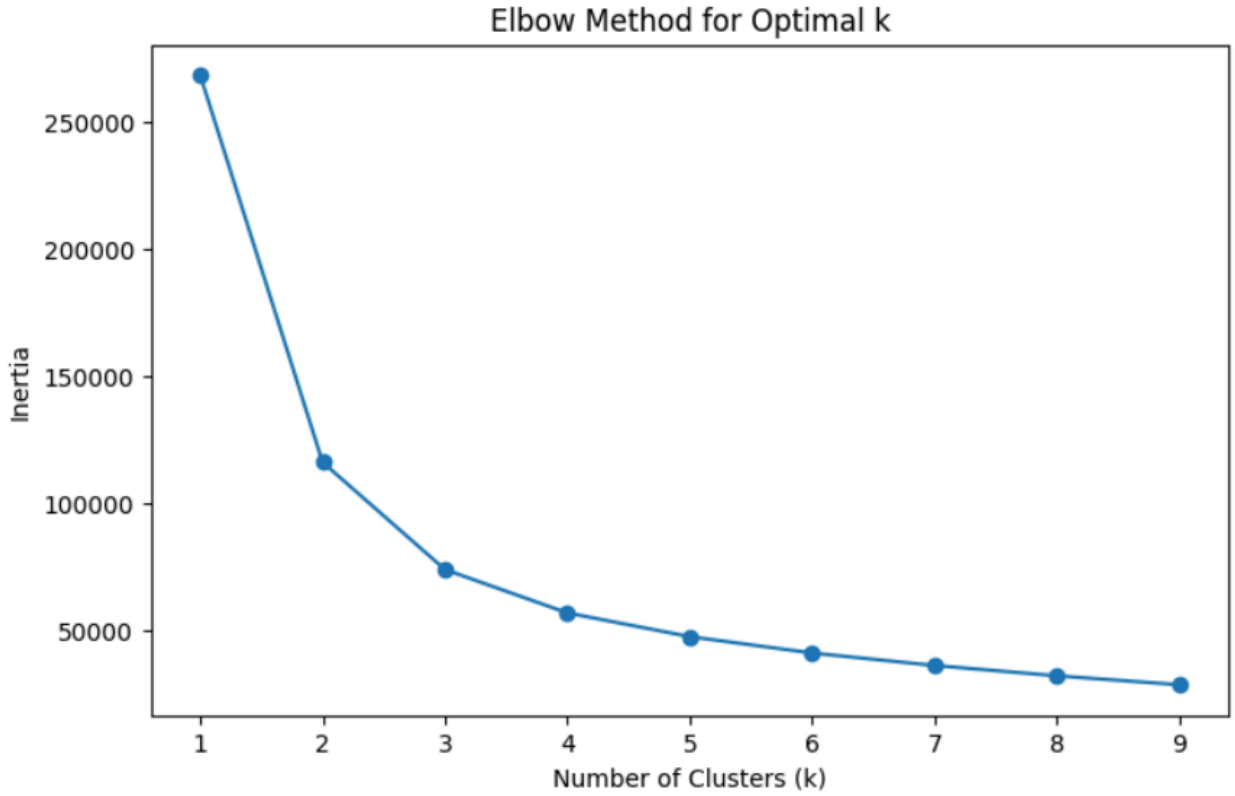
2.1 Elbow Method for Optimal K

The Elbow Method helps determine the optimal number of clusters by plotting inertia (sum of squared distances to the nearest centroid) for different values of K. The optimal K is chosen at the point where inertia starts decreasing at a slower rate, forming an "elbow."

ELBOW METHOD

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

distortions = []
K_range = range(1, 10)  # Testing for k from 1 to 10

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(df[['scaled_followers', 'scaled_friends']])
    distortions.append(kmeans.inertia_)

# Plot Elbow Method
```

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



2.2 Applying K-Means Clustering

Once the optimal K is selected, we apply the K-Means algorithm to cluster users based on their scaled follower and friend counts. The algorithm iteratively refines cluster assignments until convergence.

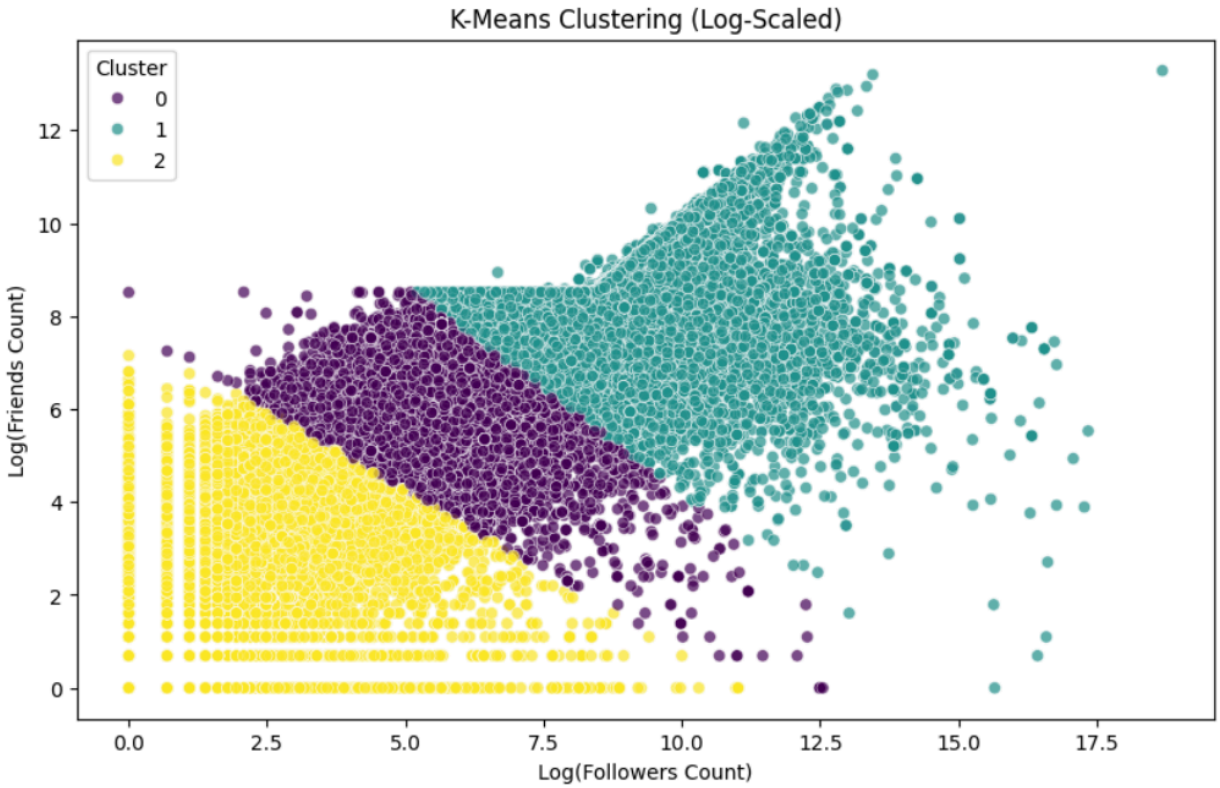
2.3 Visualization of K-Means Clustering

We plot the clustered data using a scatter plot, with different colors representing different clusters. This visualization helps interpret how users are grouped based on their Twitter metrics.

K MEANS CLUSTERING CODE

```
# Set optimal k (based on the Elbow Method)
optimal_k = 3  # Adjust based on the elbow plot
```

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

import seaborn as sns

plt.figure(figsize=(10, 6))
sns.scatterplot(
    x=df['log_followers'],
    y=df['log_friends'],
    hue=df['kmeans_cluster'],
    palette='viridis',
    alpha=0.7
)
plt.xlabel('Log(Followers Count)')
plt.ylabel('Log(Friends Count)')
plt.title('K-Means Clustering (Log-Scaled)')
plt.legend(title="Cluster")
plt.show()
```



3. Density-Based Clustering (DBSCAN)

DBSCAN is a clustering algorithm that groups points based on density rather than distance. It is effective for datasets with noise and clusters of irregular shapes.

3.1 Applying DBSCAN

DBSCAN requires two key parameters:

- eps: Defines the radius of a neighborhood around a point.
- min_samples: Specifies the minimum number of points required to form a cluster.

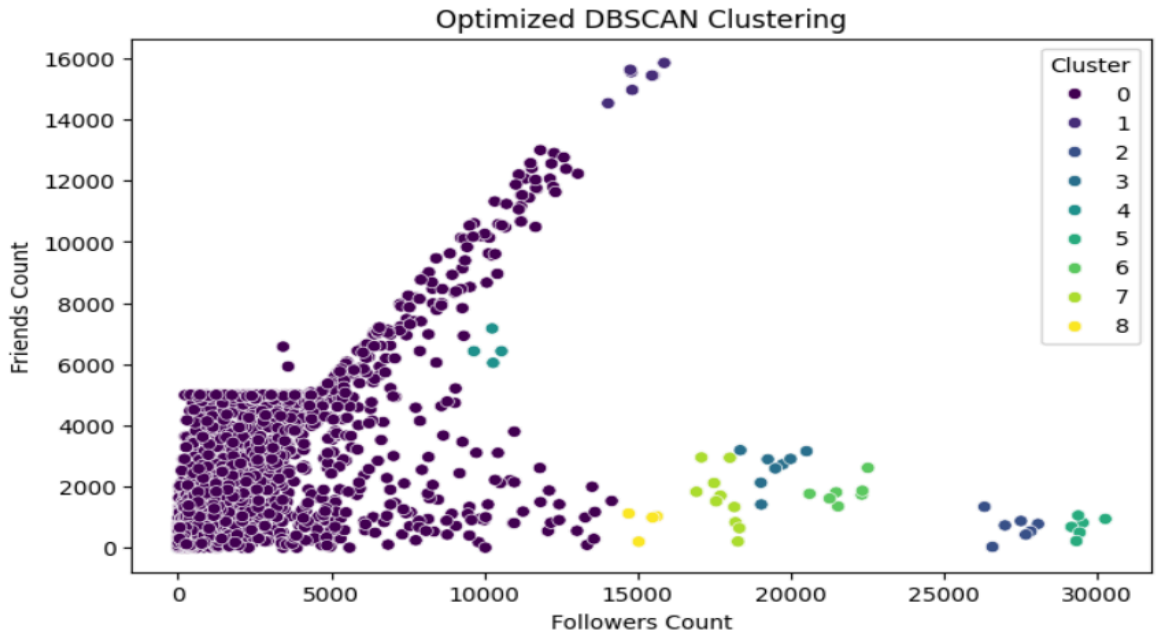
We sample 5000 data points and apply DBSCAN with optimized parameters to identify clusters while filtering out noise points (assigned label -1).

3.2 Visualization of DBSCAN Clustering

We plot the clustered data points, excluding noise, to observe the structure of the detected clusters. Unlike K-Means, DBSCAN does not require specifying the number of clusters beforehand.

DBSCAN CODE

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

# Sample a subset of data (Adjust sample size if needed)
df_sample = df[['followers_count', 'friends_count']].sample(n=5000, random_state=42)

# Apply DBSCAN with optimized parameters
dbscan = DBSCAN(eps=1000, min_samples=5)
df_sample['dbscan_cluster'] = dbscan.fit_predict(df_sample)

# Plot DBSCAN Clusters (excluding noise points)
plt.figure(figsize=(8, 5))
sns.scatterplot(data=df_sample[df_sample['dbscan_cluster'] != -1],
                x='followers_count', y='friends_count',
                hue='dbscan_cluster', palette='viridis', legend='full')
plt.xlabel('Followers Count')
plt.ylabel('Friends Count')
plt.title('Optimized DBSCAN Clustering')
plt.legend(title='Cluster')
plt.show()
```



4. Silhouette Score for Clustering Validation

The Silhouette Score is a metric used to evaluate the quality of clustering. It measures how well-separated clusters are and is defined as: S=b−amax(a,b)S = \frac{b-a}{\max(a,b)} where:

- aa is the average intra-cluster distance (cohesion)
- bb is the average nearest-cluster distance (separation)
- SS ranges from -1 to 1, with higher values indicating better clustering.

4.1 Silhouette Score for K-Means

We compute the silhouette score to assess how well-defined the K-Means clusters are. A higher score suggests distinct, well-separated clusters.

4.2 Silhouette Score for DBSCAN

For DBSCAN, we compute the silhouette score only if more than one valid cluster exists (excluding noise points). If DBSCAN fails to identify meaningful clusters, the silhouette score may be low.

SILHOUTTE SCORE FOR K MEANS CLUSTERING

```
from sklearn.metrics import silhouette_score

# Compute silhouette score only if clustering labels exist
if 'kmeans_cluster' in df.columns:
    score = silhouette_score(df[['scaled_followers', 'scaled_friends']], df['kmeans_cluster'])
    print(f"📊 Silhouette Score: {score:.3f}")
else:
    print("⚠️ Cluster labels missing! Ensure K-Means clustering was applied correctly.")
```

```
📊 Silhouette Score: 0.431
```

SILHOUETTE SCORE FOR DBSCAN

```
if valid_clusters['dbscan_cluster'].nunique() > 1:
    score = silhouette_score(valid_clusters[['followers_count', 'friends_count']],
                             valid_clusters['dbscan_cluster'])
    print(f'📊 DBSCAN Silhouette Score: {score:.3f}')
else:
    print("⚠️ DBSCAN did not find valid clusters or too many noise points.")
```

```
📊 DBSCAN Silhouette Score: 0.717
```

Conclusion

This experiment demonstrates the application of clustering techniques to Twitter user data. K-Means clustering effectively groups users into predefined clusters based on their social influence, while DBSCAN identifies dense regions of similar users without requiring a predefined number of clusters. The Silhouette Score helps validate clustering performance. By understanding these methods, we can gain insights into user behaviors and segment social media audiences effectively.

Experiment 08

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

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

What is Collaborative Filtering?

Collaborative Filtering is a recommendation technique that predicts a user's interests by analyzing preferences from similar users or items. It's widely used in platforms like Netflix, Amazon, and UberEats for personalized recommendations. There are two main types:

1. User-Based Collaborative Filtering:
    - Recommends items liked by similar users.
2. Item-Based Collaborative Filtering:
    - Recommends items that are similar to what the user already liked.

Matrix Factorization

Collaborative filtering often involves creating a User-Item Ratings Matrix, which is sparse (many missing values). Matrix Factorization techniques (like SVD) are used to:

- Reduce dimensionality.
- Discover latent features (hidden patterns).
- Predict missing ratings.

Singular Value Decomposition (SVD)

SVD decomposes a user-item matrix R into three matrices:

R≈U⋅Σ⋅VTR \approx U \cdot \Sigma \cdot V^TR≈U⋅Σ⋅VT

- U: User-feature matrix

- Σ: Diagonal matrix of singular values
- V^T: Restaurant-feature matrix

SVD helps us represent users and restaurants in a shared latent space, allowing us to compute predicted ratings and make recommendations.

Collaborative Filtering Breakdown (UberEats Dataset)

Step 1: Import Required Libraries

```
import pandas as pd
import numpy as np
from sklearn.decomposition import TruncatedSVD
```

- pandas: To load and manipulate the dataset.
- numpy: For matrix computations.
- TruncatedSVD: A dimensionality reduction technique used for matrix factorization.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1059 entries, 0 to 1058
Data columns (total 27 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   city             1058 non-null   object
 1   state            1059 non-null   object
 2   zipcode          1056 non-null   object
 3   address          1059 non-null   object
 4   loc_name         1059 non-null   object
 5   loc_number       1059 non-null   object
 6   url              1059 non-null   object
 7   promotion        121 non-null    object
 8   latitude         1059 non-null   float64
 9   longitude        1059 non-null   float64
 10  is_open          1059 non-null   bool
 11  closed_message   1045 non-null   object
 12  delivery_fee     3 non-null      float64
 13  delivery_time    14 non-null     object
 14  review_count     393 non-null    float64
 15  review_rating    443 non-null    float64
 16  price_bucket     909 non-null    object
 17  img1             1006 non-null   object
 18  img2             1006 non-null   object
 19  img3             1006 non-null   object
 20  img4             1006 non-null   object
 21  img5             1006 non-null   object
```

Step 2: Load the Cleaned UberEats Dataset

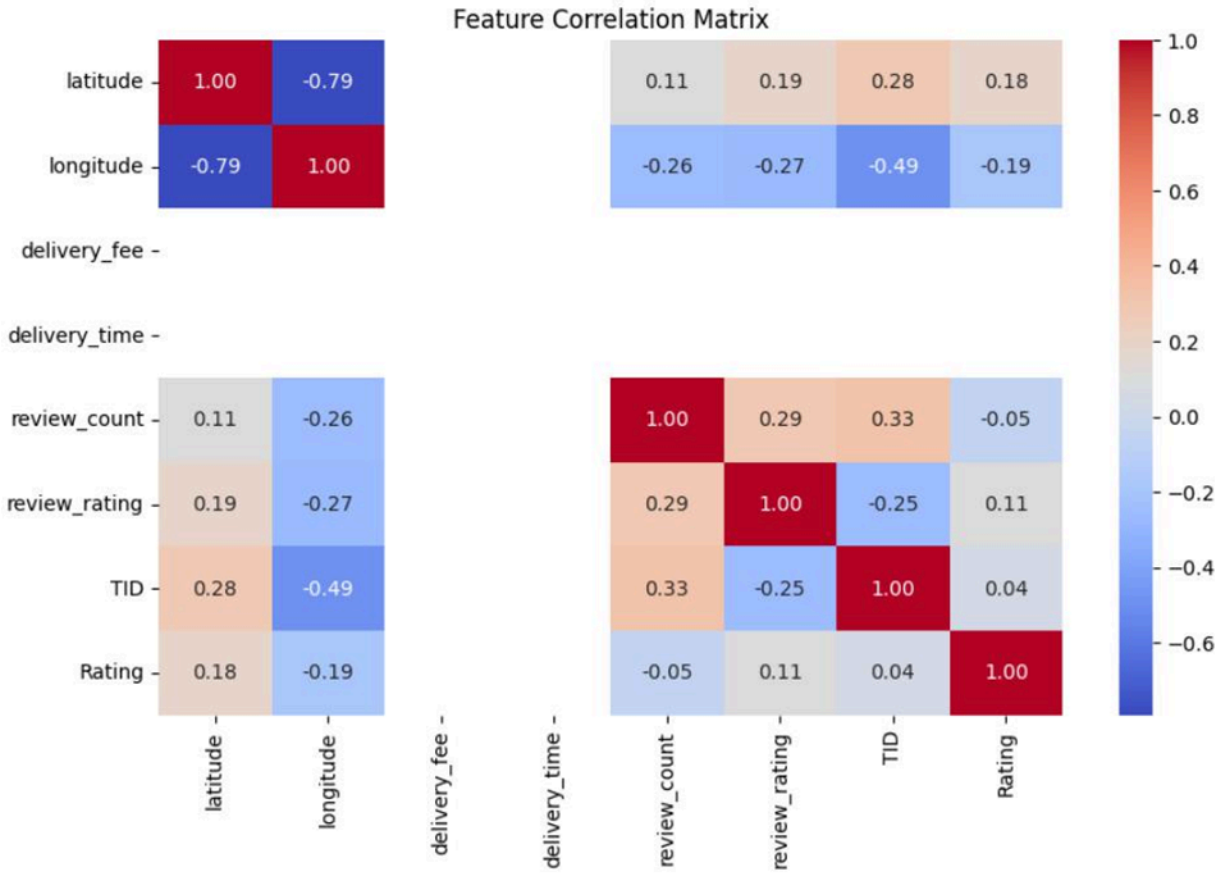
```
file_path = "UberEats_Cleaned_Dataset.csv"
df = pd.read_csv(file_path)
```

- Loads the cleaned dataset containing user reviews and ratings.

Step 3: Create the User-Item Matrix

user_item_matrix = df.pivot_table(index='User_ID', columns='Restaurant', values='Rating', fill_value=0)

- Creates a matrix where:
    - Rows = Users
    - Columns = Restaurants
    - Values = Ratings
- Missing ratings are filled with 0 (assumes user hasn't rated those restaurants).



Step 4: Apply Singular Value Decomposition (SVD)

svd = TruncatedSVD(n_components=10, random_state=42)

matrix_svd = svd.fit_transform(user_item_matrix)

- SVD breaks the user-item matrix into lower-dimensional matrices.

●n_components=10 means we reduce to 10 latent features (like cuisine type, price preference, etc.).

Step 5: Define the Recommendation Function

def get_recommendations(user_id, n=5):

if user_id not in user_item_matrix.index:
return "User not found."
user_index = user_item_matrix.index.get_loc(user_id)
user_ratings = matrix_svd[user_index]
restaurant_scores = np.dot(user_ratings, svd.components_)

recommended_restaurants = np.argsort(restaurant_scores)[::-1][:n] return
user_item_matrix.columns[recommended_restaurants]

- Input: A user ID and number of recommendations.
- Output: Top-N restaurants based on predicted ratings.
- Steps inside function:
  - Get the user's vector in the SVD-reduced space.
  - Compute similarity scores with all restaurants.
  - Return the restaurants with the highest scores.

Step 6: Example Usage

user_id = "User_2"
recommended = get_recommendations(user_id, n=5)
print(f"Top 5 Recommended Restaurants for {user_id}:")
print(recommended)

- Replace "User_2" with any user present in the dataset.
- Prints out top 5 personalized recommendations.

```
Top 5 Recommended Restaurants for User_2:
Index(['The Purple Onion (Inverness)', 'Hong Kong Seafood',
       'La Paz (Euclid Ave)', 'Papa Johns (2480 Palomino Lane)'
       'El Patron 4'],
      dtype='object', name='Restaurant')
```

Conclusion:

In this project, we successfully implemented a collaborative filtering-based recommendation system using Singular Value Decomposition (SVD) on the UberEats dataset. By transforming raw user review data into a structured user-item rating matrix, we were able to extract latent user preferences and restaurant features.

The SVD approach enabled us to overcome challenges of data sparsity and provided a powerful way to predict user interests, even when direct ratings were missing. The generated recommendations are personalized, relying on hidden patterns in user behavior rather than explicit restaurant characteristics.

This model is particularly effective for platforms like UberEats, where understanding user preferences from limited interactions is key. It demonstrates how machine learning and matrix factorization techniques can enhance user experience by offering relevant, data-driven suggestions.

Overall, the collaborative filtering approach has laid the foundation for a scalable recommendation engine that can adapt to more complex user data, incorporate real-time feedback, and evolve with user tastes.

Experiment 9

Aim: To perform Exploratory Data Analysis using Apache Spark and Pandas

Theory:

1. What is Apache Spark and How It Works?
Apache Spark is an open-source distributed computing framework designed for big data processing, faster than traditional Hadoop MapReduce. It enables in-memory computation, making operations much quicker for iterative tasks like machine learning, data analysis, and graph processing.

Key Components of Apache Spark:

- **Spark Core:** The base engine for large-scale parallel data processing.
- **Spark SQL:** Module for structured data processing using DataFrames and SQL.
- **MLlib:** Machine Learning library for scalable learning algorithms.
- **GraphX:** For graph computations.
- **Spark Streaming:** For real-time stream data processing.

How Spark Works:

- Spark processes data in RDDs (Resilient Distributed Datasets) or DataFrames.
- The Driver Program initiates a SparkContext, connecting to a Cluster Manager.
- Tasks are distributed across Executors for parallel execution.
- Supports lazy evaluation—transformations are only computed when an action is called.

2. How Data Exploration is Done in Apache Spark?
EDA in Apache Spark follows similar principles to pandas but is designed to scale to massive datasets across clusters.

Steps of EDA in Spark:

1. **Initialization:**
   Import pyspark and create a SparkSession using SparkSession.builder.
   This session acts as the entry point to Spark functionalities.

2. **Load Dataset:**
   Use spark.read.csv() or .json() to load data into a Spark DataFrame.

Enable header=True and inferSchema=True for cleaner loading.

3. **Understand Data Schema:**
   Use .printSchema() to view column types and .show() for a data preview.
   .describe() provides summary statistics like mean, min, and max.

4. **Handle Missing Values:**
   Use df.na.drop() to remove nulls or df.na.fill("value") to fill them.
   This step is crucial to clean data for accurate analysis.

5. **Data Transformation:**
   Apply .withColumn(), .filter(), .groupBy() to reshape and summarize data.
   These functions help in refining the dataset before analysis.

6. **Data Visualization:**
   Convert Spark DataFrame to Pandas using .toPandas() for plotting.
   Then visualize with tools like matplotlib or seaborn.

7. **Correlation & Insights:**
   Use .corr() in Pandas or MLlib's Correlation.corr() for relationships.
   Group, pivot, and analyze data patterns for meaningful insights.

Conclusion:

In this experiment, I learned how to perform Exploratory Data Analysis using Apache Spark and Pandas. I understood how to initialize a SparkSession, load large datasets efficiently, and explore their structure using Spark functions like .show(), .printSchema(), and .describe(). I also learned how to handle missing values, transform data using Spark DataFrame operations, and convert data to Pandas for visualization. Additionally, I explored how to compute correlations and derive insights through grouping and aggregation. This experiment helped me grasp the scalability and power of Spark in handling big data and how it complements traditional Python libraries like Pandas and Seaborn for insightful data analysis.

Experiment 10

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

Theory:

1. What is Streaming? Explain Batch and Stream Data:
Streaming refers to the continuous processing of real-time data as it arrives. It is commonly used in applications that require immediate action such as fraud detection, stock market analysis, and live dashboards. Streaming data is unbounded, time-sensitive, and flows in continuously.

Batch data processing, in contrast, involves collecting data over a period and processing it together. It is widely used in data warehousing, periodic reporting, and data transformation tasks. The data is bounded and processed in chunks with scheduled jobs.

Examples:

- **Batch:** Generating monthly sales reports.
- **Stream:** Real-time user click analysis on a website.

2. How Data Streaming Takes Place Using Apache Spark:
Apache Spark handles stream processing through its **Structured Streaming** engine. Structured Streaming treats incoming data streams as an unbounded table and performs incremental computation using the same DataFrame API used for batch jobs.

The streaming data can be ingested from various sources such as **Kafka**, **sockets**, **directories**, or **cloud storage**. Spark then processes the data using transformations like filter, select, groupBy, and aggregations. Developers can apply **window operations**, manage **late-arriving data using watermarking**, and use **checkpointing** for fault tolerance.

Internally, Spark divides the live stream into **micro-batches**. These micro-batches are processed using the Spark engine and then output to sinks like **HDFS**, **databases**, or **dashboards**. With its high scalability and distributed nature, Apache Spark ensures that real-time data processing can be performed with low latency and high throughput.

Key Features:

- Unified APIs for batch and streaming
- Support for stateful computations
- Integration with structured data sources
- Fault-tolerant and scalable architecture

Use Case Examples:

- Real-time transaction monitoring
- Streamed log analysis
- Live social media analytics

Conclusion:

In this experiment, I gained a strong understanding of the differences between batch and streaming data processing. I learned that batch processing is ideal for historical and periodic tasks, while streaming suits real-time, continuous data needs.

Through Apache Spark, I explored **Structured Streaming**, which provides a powerful, unified framework to handle both types of workloads. I learned how to ingest live data from sources like **Kafka** or **files**, apply transformations, and output results dynamically.

This helped me appreciate Spark's capabilities in managing complex data pipelines and real-time analytics. Overall, I understood how Spark's architecture enables **scalable** and **fault-tolerant** processing, making it a preferred tool for modern data-driven applications.

ASSIGNMENT NO. 1

Bhumisha Parchani
D15C
38

04/05

AIDS Assignment 1

Q.1 What is AI? Considering the COVID-19 pandemic situation, how AI helped to survive and renovated our way of life with different application?

→ Artificial Intelligence (AI) refers to the ability of machines and systems to simulate human intelligence including tasks like learning, reasoning, problem-solving and decision making. It leverages techniques like machine learning and NLP to analyze data, identify patterns and improve performance over time.

Role of AI during the Pandemic:

- Healthcare - AI was used to quickly analyze medical imaging (eg. CT scans) for diagnosing COVID-19 cases and predicting severity.
- Remote Work & Education - The pandemic drove widespread adoption of AI in virtual learning and remote work tools.
- Retail - Retail stores are using computerised models to map out their stores and track inventory. This is in a response to the need given the rush to buy specific items at various stages of the pandemic.

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

→ AI agents are autonomous systems that perceive their environment, make decisions and take actions to achieve specific goals.

AI Agent Terminology:

1. Agent
   - An AI system that interacts with the environment and takes action.
   Example - A self-driving car that perceives traffic signals and adjusts speed accordingly.
2. Environment
   - The surroundings in which the AI agent operates.
   Ex - For a chess-playing AI, the chessboard and opponent are part of its environment.
3. Percepts
   - The information received by agent via sensors.
   Ex: A robot vacuum detecting obstacles using cameras and infrared rays.
4. Actions
   - The moves an agent makes in response to percepts.
   Ex: A chatbot responding to a user's query.
5. Percept sequence
   - The entire history of percepts received by an agent
   Ex: A recommendation system tracking user's past preferences.
6. Performance Measure
   - Determine success of an agent
   Ex - A self-driving car's performance measure will be reaching the destination safely and in a timely manner.

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

→ Initial State:

1	2	3
4	0	6
7	5	8

Goal State:

1	2	3
4	5	6
7	8	0

Misplaced tiles : $h(n) = 2$

Steps to solve 8-puzzle problem by A^* :

1. Initialize a priority queue
2. Insert the initial state with $f(n) = g(n) + h(n)$
3. While queue not empty :
   - Remove state with lowest $f(n)$
   - If state = goal, return solution
   - Generate valid moves (Up, Down, Left, Right)
   - Compute $g(n)$ and $h(n)$ for new states
   - Insert new states into queue
4. Repeat until goal is reached

Example execution.

Step 1 :

1	2	3
4	0	6
7	5	8

(h=2)

Step 2 :

1	2	3
4	5	6
7	0	8

(h=1)

Step 3: Goal

1	2	3
4	5	6
7	8	0

(h=0)

Q.4 What is PEAS descriptor? Give PEAS descriptor for following:

→ PEAS stands for Performance measure, Environment, Actuators and Sensors.

P → Criteria to evaluate the agent's success

E (Environment) → Surrounding where agent operates

A (Actuators) → Components that allow agent to take actions

S (Sensors) → Components that help agent perceive its environment

1. Taxi Driver
   P - Reaching destination, fuel efficiency
   E - Roads, traffic, pedestrians
   A - Steering wheel, accelerator, brakes
   S - Camera, GPS, speedometer
2. Medical Diagonosis System
   P - Accuracy of diagonosis, patient satisfaction
   E - Medical records, test results, symptoms
   A - Prescription generation, Report generation
   S - Patient data input, lab test results
3. Music Composer
   P - Quality of composition, User satisfaction
   E - Musical genres, musical theory constraints.
   A - Music synthesizers, Instruments, Generating musical notes
   S - Composition structure, uses feedback
4. Aircraft Autolander
   P - Smooth and safe landing
   E - Weather, runway condition, air traffic
   A - Flaps, landing gear, throttle
   S - Altitude sensor, GPS, wind direction sensor.
5. Essay Evaluator
   P - Accurate grading, feedback quality
   E - Grammar rules, submitted essays
   A - displaying grades & feedback
   S - Text input, spelling and grammar checkers
6. Robotic sentry Gun for Keck Lab
   P - Correctly identifying threats, accuracy
   E - Intruders, authorized personnel
   A - Camera movement, alarm system, firing mechanism
   S - Motion sensors, facial recognition

5. Categorize a shopping bot for an offline bookstore according to each of the six dimensions.

→ 1. Partially observable
- bot may not have full visibility of store's inventory

2. Stochastic
- Outcomes uncertain due to customer behaviour, stock availability

3. Sequential
- Bot will suggest books based on previous customer queries

4. Dynamic
- Bookstore environment changes like books getting sold, new stock

5. Discrete
- Bots process finite number of actions

6. Multi-agent
- Bot interacts with customers and store employees, each have their own goals.

6. Differenciate between Model-Based and Utility Based Agents

→

Model-based agent	Utility-based agent
1. Uses an internal model of environment to make decisions.	1. Chooses actions based on utility function that measure performance.
2. Decisions based on past & present percepts.	2. Selects action based on maximizing utility
3. Can be goal-based but doesn't necessarily optimize for best outcomes	3. Optimizes for highest possible utility, ensuring better performance
4. Ex - Robot vacuum using a map to navigate	4. Ex - Self-Driving Car, choosing safest & fastest route

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

→ Knowledge-based agent

A knowledge-based agent (KBA) uses stored knowledge to make decisions. It consists of

- Knowledge Base → Stores facts, rules and logic
- Inference Engine → Uses reasoning to derive conclusions
- Perception (sensors) → Gathers new information from environment
- Action Mechanism → Performs appropriate actions based on reasoning

Eg. AI in medical diagnosis uses past cases and symptoms to diagnose diseases.

Learning agent

A learning agent improves its performance over time. It consists of:

- Learning Element → Updates knowledge based on experience
- Performance Element → Decides actions based on current knowledge
- Critic → Provides feedback by evaluating actions
- Problem Generator → suggests new actions to improve learning

Eg. A self-driving car learns from traffic patterns and adjusts driving behaviour.

8. Convert the following to predicates:

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

CarAvailable → TravelsByCar (Anita)

¬ CarAvailable → TravelsByBus (Anita).

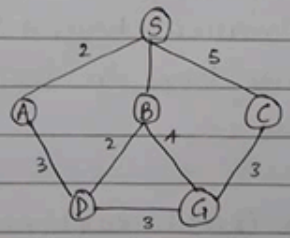
b. Bus goes via Andheri and Goregaon.
GoesVia (Bus, Andheri) ∧ GoesVia (Bus, Goregaon)

c. Car has puncture so is not available.
Puncture (Car)
Puncture (car) → ¬ CarAvailable

Will Anita travel via Goregaon? Use forward reasoning
From (c)
Puncture (Car) is true
As Puncture (car) → ¬ CarAvailable
From (a)
¬ CarAvailable, we use ¬CarAvailable → TravelsByBus (Anita)
From (b)
GoesVia (Bus, Goregaon)
Since Anita travels by bus, she follows this route

Thus, Anita will travel via Goregaon.

Q.9 Find route from S to G using BFS



→ To find route from S to G using BFS, we systematically explore all nodes level by level starting from source node (S) until we reach destination node (G).

1. Start at S
Queue = [S]

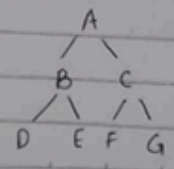
2. From S, we go to it neighbours : A, B, C
   Queue = [A, B, C]
3. Dequeue A and explore its neighbours.
   Queue = [B, C, D]
4. Dequeue B and explore its neighbours
   Queue = [C, D, G]
5. Dequeue C and queue neighbours
   Queue = [D, G]
6. Exp Dequeue D
   Queue = [G]
7. Dequeue G
   As G is our destination, BFS stops here.

Route from S to G: S → B → G

Q.4 What do you mean by depth limited search? Explain Iterative Deepening search with example.

→ Depth Limited Search (DLS) is an uninformed search algorithm that modifies DFS by introducing a depth limit L, preventing exploration beyond the predefined level. This prevents infinite loops in infinite graphs but risks missing goals beyond L.

Iterative Deepening search (IDS) combines DLS with BFS by incrementally increasing the depth limit.
Example:



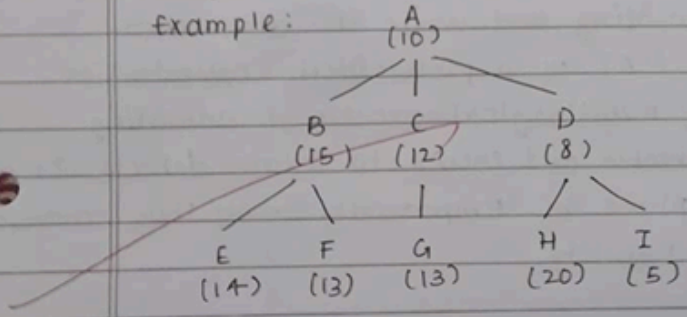
Goal = G.

Iteration 1 : Depth limit = 0
Nodes visited : A
Result : Goal not found.
Iteration 2 : L = 1
Nodes visited : A → B → C
Goal not found
Iteration 3 : L = 2
Nodes visited : A → B → D → E → C → F → G
Goal G found at L=2.

Q.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 optimization algorithm, which moves toward better neighboring solutions until it reaches a peak.

Example:



Goal=G.

Steps:
Start at root node A(10)
Compare its children B, C and D
Move to child with highest value i.e. B(15)
Repeat for B's children E and F
Terminate at E(14)
The algorithm stops at E(14) not reaching the goal G.

Drawbacks:

1. Local Maxima - The algorithm greedily selects the best immediate child and can thus get stuck on local maxima
2. Plateaus - If siblings have equal values, the algorithm can't decide the next step and gets stuck
3. Ridges - Narrow uphill paths require backtracking which Hill climbing algorithm does not support.

Limitations of Steepest-Ascent Hill Climbing:

1. Computationally expensive - Evaluates all neighbors before selecting the best
2. Can get stuck - It can still get stuck in local maxima plateaus or ridges.
3. NO Global Optimality. - It only focuses on immediate improvements.

Q.13. Explain simulated annealing and write its algorithm.

→ Simulated annealing (SA) is a probabilistic optimization algorithm inspired by metallurgical process of annealing. where materials are heated and cooled to reduce defects. It escapes the local optima by temporarily accepting worse solution with a probability.

Algorithm:

1. Initialize
   - Set an initial solution and define an initial temperature T.
2. Repeat until Stopping condition
   - Generate a new neighbor solution
   - Compute change in cost ($\Delta E = E_{new} - E_{current}$)
   - If new solution is better i.e. $\Delta E > 0$, accept it.
   - If worse, accept it with probability. $P = e^{-\Delta E/T}$

• Decrease temperature T (cooling schedule)

3. Return best solution.

Example:

Travelling Salesman Problem

Swap two cities in a route. Accept a longer route early (high T) but reject it later (low T).

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

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

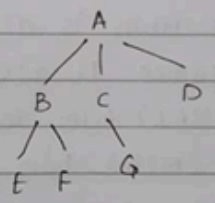
$$f(n) = g(n) + h(n)$$

$g(n)$ → cost to reach node n from start

$h(n)$ → heuristic estimate of cost to reach from goal to n

$f(n)$ → total estimated cost

Example: Goal: G



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

Steps:

1. Start at root node A

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

$= 6.$

2. Expand neighbors : B, C, D.

f(B) = 1 + 4 = 5

f(C) = 2 + 2 = 4

f(D) = 4 + 11 = 11

3. Choose lowest value that is C (f(C) = 4)

4. Expand neighbors of C : G.

f(G) = 2 + 4 + 0 = 6.

5. Goal reached at G with total cost 6.

Advantages →

- efficient for finding shortest paths in weighted graphs
- balances exploration by considering both g(n) & h(n)

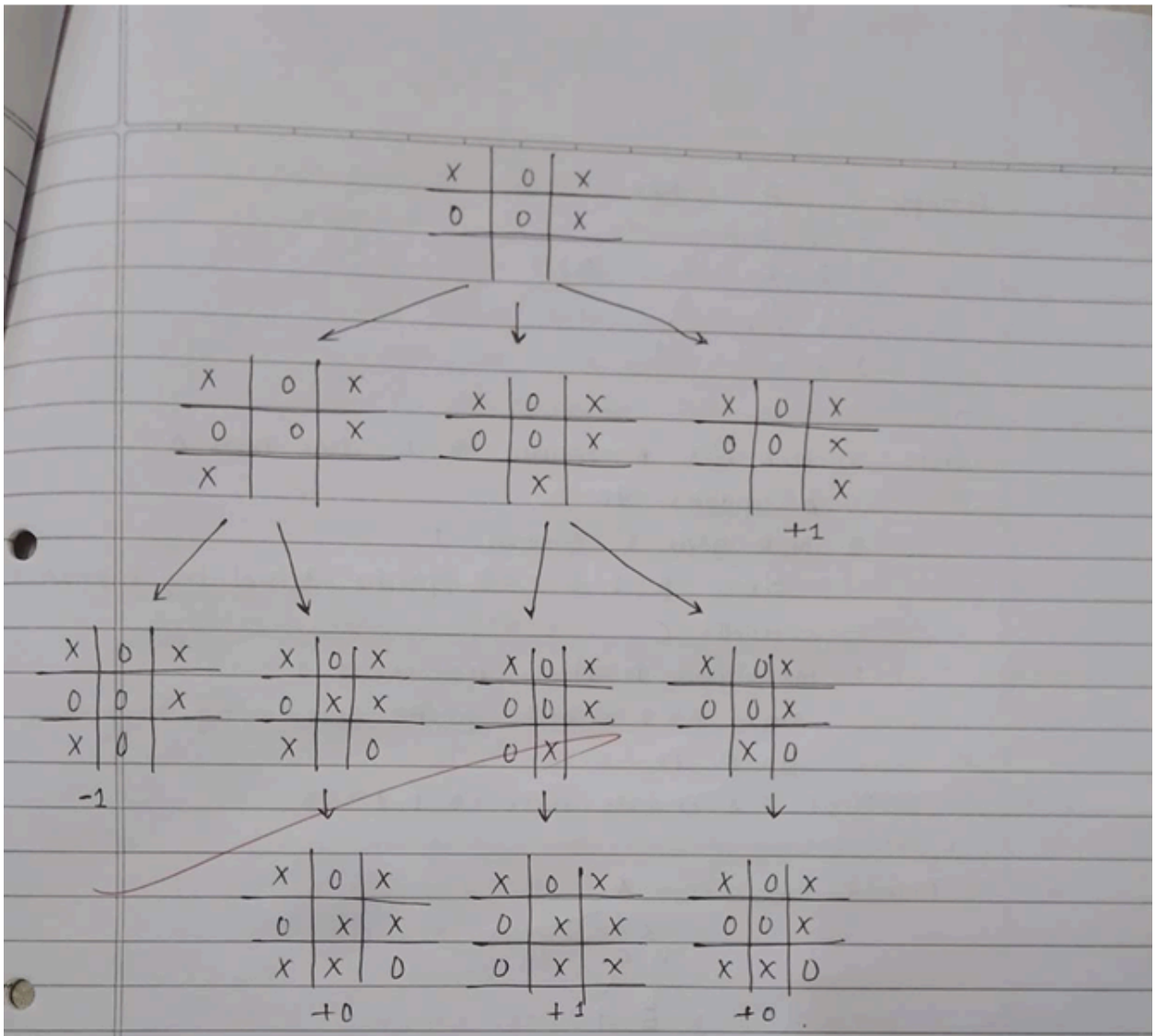
Q.15 Explain Min-Max Algorithm and draw game tree for Tic Tac Toe Game.

→ The Minimax Algorithm is a decision making algorithm used in two-player games. It assumes

- One player (MAX) tries to maximize the score
- Other player (MIN) tries to minimize the score
- Game tree represents all possible moves.

Algorithm

1. Generate Game tree
   → All possible moves from current state
2. Assign scores
   Terminal states (win / loss / draw), get +1, -1, 0
3. MAX picks highest value from children
   MIN picks lowest value
4. Repeat until Root Node is evaluated.



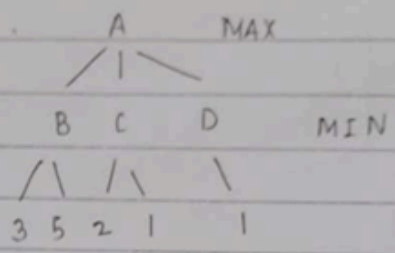
16. Explain Alpha beta pruning algorithm for adversarial search with example.

→ Alpha beta pruning is an optimization technique for the Minimax algorithm used in adversarial search purposes like game playing AI. It reduces computational complexity by pruning branches of game tree that do not influence the final decision.

α - best value maximizing player can guarantee

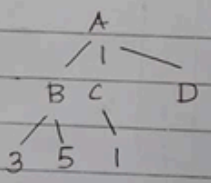
β - best value minimizing player can guarantee.

Example :



Steps : 1. MIN node B returns 3 to MAX node A
A updates to 3
2. MIN node C return 1
since $\beta = 1 \leq \alpha = 3$, prune remaining branches under C
3. MIN node D finds leaf value 1
since $\beta = 1 \leq \alpha = 3$, prune remaining branches under D
Final : A chooses max (3, 1, 1) = 3.

Pruned Tree :



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

→ PEAS for WUMPUS world problem

1. P – • Maximize Rewards
+1000 for collecting gold & exiting grid
• Minimize penalties
-1000 for falling into a pit or being eaten by WUMPUS.
-1 point for each action taken
-10 points for using an arrow

2. E - • Grid Layout (4×)
containing pits, wumpus, gold, walls, breeze
• Partially observable. -agent cannot see entire grid and must rely on sensory inputs
3. A - Move left, right, forward,
Grab (to collect gold, Shoot (to eliminate wumpus)
4. S - Breeze : indicates pit is adjacent
Stench : indicates wumpus in adjacent cell
Glitter : indicates gold
Bump : indicates a wall has been encountered.

Generating percept sequence

1. Initial position - The agent starts at a defined position typically (1, 1)
2. Movement & Perception :
• As agent moves from one cell to another, it uses sensors to gather info about surroundings
• For eg. if it detects a breeze, it means there is a pit in adjacent cell.
3. Creating Percept sequence
Each time the agent moves, it records its percepts as sequence.
For eg : After moving to (1,2) : [None, Breeze, None]
(no stench or glitter
After moving to (2, 1) : [Stench, Breeze, None]
(indicating nearby dangers)
This sequence continues as agent explores more cells.
4. Decision making
The agent uses these percept sequences to make decisions about its next actions based on logical reasoning and inference from previous observations.

18. Solve following crypto-arithmetic problems.

SEND + MORE = MONEY.

→ To solve this problem, we need to assign a unique digi (0-9) to each letter such that the given equation hold true.

1. Set up the Equation

```
  S E N D
+ M O R E
---------
M O N E Y
```

2. since M is leading digit, it must be 1 because sum of 2 four-digit numbers cannot exceed 19998.

∴ M = 1.

```
  S E N D
+ 1 O R E
---------
1 O N E Y
```

3. D + E = Y (if carry = 0)

or D + E = 10 (if carry = 1)

Tens place.

N + R + carry = E

Hundreds place : E + O + carry = N

Thousands place : S + 1 $ = 1 + carry

S + carry = 0

∴ S = 9 since there is no carry

4. Try E = 5

If D = 7 then

Y = 7 + 5 = 12 (invalid)

D = 2 then

Y = 7.

5. Y = 7

Assume N = 6

N + R = E

6 + R = 5 (impossible)

∴ N = 6

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

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

Q.19. Consider the axioms:

All people who are graduating are happy

All happy people are smiling

someone is graduating.

Explain: 1. Represent these axioms in first predicate logic

→ 1. ∀x (Graduating(x) → Happy(x))

2. ∀x (Happy(x) → Smiling(x))

3. ∃x (Graduating(x))

2. Convert each to clause form

1. ∀x (¬Happy(x) ∨ Smiling(x))

{¬Happy(x), Smiling(x)}

2. ∀x (¬Happy(x) ∨ Graduating(x))

{¬Graduating(x), Happy(x)}

3. Graduating(c)

3. Prove "Is someone smiling?" using resolution technique.

∃x (Smiling(x))

Clause form: C4 = Smiling(y)

$C_1 = \{\neg Graduating(x), Happy(x)\}$

$C_2 = \{\neg Happy(y), Smiling(y)\}$

$C_3 = Graduating(c)$

Resolution between C_3 & C_1

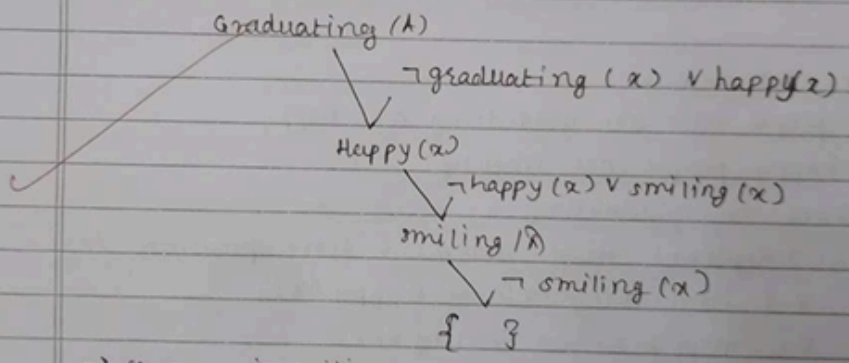
Substitute c for x in C_1:

From Graduating (c)

Happy (c)

Resolving with C_2

Smiling (c)



→ someone is smiling.

Q-20 Explain Modus Ponen with example.

→ Modus ponen is a fundamental rule of inference in logic. It states that if $P \rightarrow Q$ is true P is true, then Q must also be true. Formula: $\frac{P \rightarrow Q, P}{Q}$

Ex, if it is raining, then it is soggy $(P \rightarrow Q)$

it is raining (P)

∴ it is soggy (Q)

Q.21 Explain forward and backward chaining algo with example.

→ Forward chaining - It is data driven, inference algorithm that starts with known facts & applies inference rules to derive new facts with until the goal is reached

fact : A, B

Rule : $A \rightarrow C$, $B \rightarrow D$, $C \wedge D \rightarrow I$

Goal : I

Start with A & B

apply $A \rightarrow C$ to derive C

$B \rightarrow D$ to derive D

$C \wedge D \rightarrow I$ to derive I

The goal I reached

Backward chaining :

BC is a goal-driven, starts with goal & works backward to find the facts that support it.

start with goal.

fact : A, B

Rule : $A \rightarrow C$, $B \rightarrow D$, $C \wedge D \rightarrow I$

goal : I

find the rule $C \wedge D \rightarrow I$

if C & D are true

use $A \rightarrow C$ since A is true, C is true

use $B \rightarrow D$ since B is true, D is true

C & D are true, I is true

conclusion : I is reached

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
2. Find the Median
3. Find the Mode
4. Find the Interquartile range

ANS:

1. **Mean**: The mean is the average.
   Sum = 82 + 66 + 70 + 59 + 90 + 78 + 76 + 95 + 99 + 84 + 88 + 76 + 82 + 81 + 91 + 64 + 79 + 76 + 85 + 90 = 1611
   Number of values = 20
   Mean= 1611 / 20
   = **80.55**
2. **Median:** The median is the middle value when the data is sorted
   Sorted data: 59, 64, 66, 70, 76, 76, 76, 78, 79, 81, 82, 82, 84, 85, 88, 90, 90, 91, 95, 99
   Median = **81.5**
3. **Mode:** The mode is the number that appears the most.
   Mode = **76**
4. **Interquartile Range**
   IQR = Q3 - Q1
   Q1 = median of the first half = 76
   Q3 = median of the second half = 89
   IQR = 89 - 76
   = **13**

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

ANS: Tool: **Machine Learning for Kids**

1. **Target Audience**
   The primary target audience for Machine Learning for Kids includes young students aged 8–17, beginners with no prior exposure to machine learning, and educators seeking to introduce foundational AI and machine learning concepts in

a classroom setting.

2. **Use of this Tool by the Target Audience:**
   Machine Learning for Kids allows students to create and train simple machine learning models by labeling datasets and using them in interactive projects. Students can train models to recognize text, images, numbers, or sounds, and subsequently deploy these models in practical applications such as games, chatbots, or mobile applications using Scratch, Python, or App Inventor.
   For example, students can train a model to differentiate between images of cats and dogs and then use this model within a Scratch project to classify new images**.**

3. **Tool's Benefits and Drawbacks**

Benefits	**Drawbacks**
Highly user-friendly interface that enables students to engage with machine learning concepts without requiring advanced programming skills.	Limited scalability; the platform is not designed for handling large datasets or complex deep learning models
Provides practical, hands-on experience with real-world machine learning workflows.	Primarily supports basic models, making it unsuitable for more sophisticated machine learning projects.
Seamlessly integrates with platforms such as Scratch and App Inventor, facilitating interactive project development.	May oversimplify fundamental machine learning principles, which could be less effective for older or more advanced learners.
Freely available and easily accessible online, making it ideal for educational use.	Certain functionalities may be dependent on stable internet connectivity.

4. **Predictive Analytic:**
   The platform is primarily designed for building models that predict outcomes based on previously labeled data. For instance, after training on examples of positive and negative reviews, a model can predict the sentiment of a new review. This predictive nature classifies the tool under predictive analytics.

5. **Supervised Learning**
   Machine Learning for Kids uses labeled data during the model training phase, where each input is paired with a corresponding output. Students manually label their data (e.g., images tagged as "cat" or "dog") and the system learns to classify new, unseen data based on these labels. This approach is characteristic of supervised learning.

Tool: **Teachable Machine**

1. **Target Audience**
   The primary target audience for Teachable Machine includes students, educators, hobbyists, and beginners who wish to experiment with machine learning without requiring advanced programming skills. It is particularly suitable for individuals aged 12 and above who are interested in quickly building and deploying machine learning models in a user-friendly environment.

2. **Use of the Tool by the Target Audience**
   Teachable Machine enables users to create machine learning models by providing examples and training the model directly within a web interface. Users can train models to recognize images, sounds, or poses through simple drag-and-drop actions and webcam recordings.
   For instance, a student could train an image classification model by showing different objects to a webcam and labeling them, then export the trained model for use in websites, applications, or devices — all without writing a single line of code.

3. **Tool's Benefits and Drawbacks**

Benefits	**Drawbacks**
Extremely easy to use, requiring no prior coding knowledge.	Limited model complexity, making it unsuitable for professional or advanced applications.
Supports quick prototyping and instant feedback, allowing users to iterate rapidly.	Cannot fine-tune model parameters, limiting customization and optimization
Provides export options to TensorFlow.js, TensorFlow Lite, and other formats, enabling integration into external applications	Models are trained locally in-browser, which can lead to performance constraints on devices with lower processing power.

Free to use and accessible online, promoting widespread experimentation with AI.	Not ideal for large-scale datasets or sophisticated multi-class problems.

4. **Predictive Analytic**
   Teachable Machine builds models that predict outcomes based on user-provided examples. For example, after training a model with images of different emotions (happy, sad, surprised), the model can predict the emotion shown in a new image. This predictive functionality categorizes Teachable Machine under predictive analytics.
5. **Supervised Learning**
   Teachable Machine relies on labeled data provided by the user during the training process. Users must supply categorized examples (e.g., images labeled "cat," "dog," "car") to teach the model how to classify new inputs. This process follows the principles of supervised learning, where the system learns a mapping from inputs to known outputs.

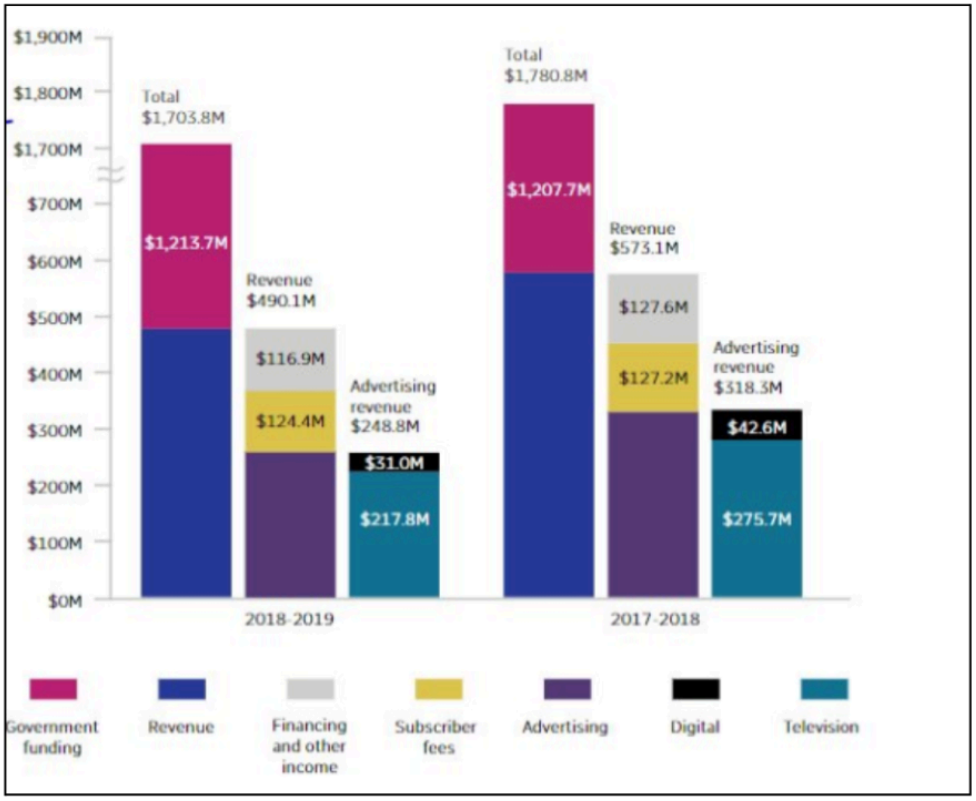
Q.3 Data Visualization

Sources:
https://thedeepdive.ca/cbc-commits-a-chart-crime-in-representing-its-funding-and-revenue-sources/

https://www.codeconquest.com/blog/12-bad-data-visualization-examples-explained/htoc-bad-data-visualization-example-2

Misleading CBC(Canadian national broadcast company) Funding Bar Chart:

In April 2023, a bar chart from the Canadian Broadcasting Corporation's (CBC) 2018–2019 Annual Report resurfaced online and quickly drew widespread criticism for its misleading design. The chart was intended to show CBC’s funding sources, including government appropriations and advertising revenue. However, it sparked controversy after viewers noticed serious issues with how the data was presented.



The axis had a sudden break—jumping from $700M to $1.7B—causing a $490M revenue bar to appear visually larger than the $1.2B government funding bar. This misleading scale made it seem like television revenue equaled or exceeded government funding, which is factually incorrect. Additionally, the chart layout was flawed: revenue and advertising revenue were shown as separate bars rather than subdivisions of the total income, creating further confusion.

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

ANS: **Model Used: SVM**

1. Import required libraries

```
[1]  # Import necessary libraries
     import pandas as pd
     import numpy as np
     from sklearn.model_selection import train_test_split, GridSearchCV
     from sklearn.preprocessing import StandardScaler
     from sklearn.svm import SVC
     from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay, accuracy_score
     from imblearn.over_sampling import SMOTE
     import matplotlib.pyplot as plt
     import seaborn as sns
```

2. Loading the dataset

```
[2]  data = pd.read_csv('/content/sample_data/diabetes.csv')
```

3. Performing data preprocessing

	0
Pregnancies	0
Glucose	0
BloodPressure	0
SkinThickness	0
Insulin	0
BMI	0
DiabetesPedigreeFunction	0
Age	0
Outcome	0

Since there are no null values, we can skip imputation.

4. Standardization of data

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Feature scaling standardizes the values of your features so they all have: Mean = 0 and Standard Deviation = 1

5. Resolve Class Imbalance using SMOTE

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

SMOTE (Synthetic Minority Over-sampling Technique) is used to fix class

imbalance by creating synthetic samples for the minority class.

6. Train, Validation, Test Split

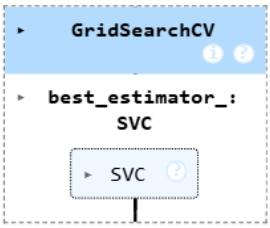
```
# Train, Validation, Test Split
X_train_full, X_test, y_train_full, y_test = train_test_split(
    X_resampled, y_resampled, test_size=0.10, random_state=42, stratify=y_resampled
)
X_train, X_val, y_train, y_val = train_test_split(
    X_train_full, y_train_full, test_size=2/9, random_state=42, stratify=y_train_full
)  # 2/9 to split 70/20/10 correctly
```

7. Define the model

```
[8]  svm_model = SVC()
```

8. Hyperparameter Tuning using GridSearchCV

```
# Hyperparameter Tuning using GridSearchCV
param_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': [1, 0.1, 0.01, 0.001],
    'kernel': ['rbf']
}

grid = GridSearchCV(svm_model, param_grid, refit=True, verbose=0, cv=5)
grid.fit(X_train, y_train)
```



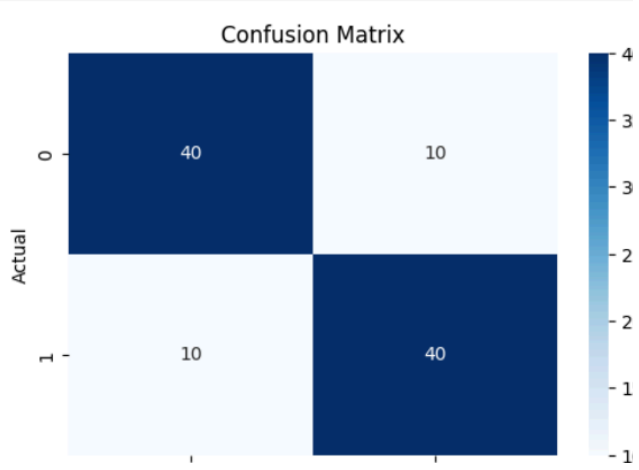
9. Evaluate on Validation set

```
Validation Accuracy: 0.825
```

10. Evaluation on Test Set

```
Test Accuracy: 0.8
```

11. Evaluating the model



Validation Accuracy: **82.5%**
The model achieved an accuracy of **82.5%** on the validation set, indicating good generalization during training.
Test Accuracy: **80%**
When evaluated on the unseen test set, the model obtained an accuracy of **80%**, showing consistent performance and minimal overfitting.

Confusion Matrix Analysis:

a. **True Positives (TP)**: 40
b. **True Negatives (TN)**: 40
c. **False Positives (FP)**: 10
d. **False Negatives (FN)**: 10

The confusion matrix shows a **balanced classification** performance:

- Most samples are correctly classified.
- Only 10 instances are misclassified in each class (positive and negative).
- The model is **well-generalized** and not overfitting.

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

1. Importing required libraries

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns
```

2. Define a Regression model
   Firstly we define
   class RegressionModel:

```
def __init__(self, model, param_grid):
    self.model = model
    self.param_grid = param_grid
    self.best_model = None
```

3. Load Dataset

```
def load_data(self, file_path):
    self.data = pd.read_csv(file_path)
    self.X = self.data.iloc[:, [0]]  # First column as independent
    self.y = self.data.iloc[:, 1:5]   # Columns 2 to 5 as dependent
```

4. Split Data (70/30)

```
def split_data(self, test_size=0.3):
    self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(
        self.X, self.y, test_size=test_size, random_state=42
    )
```

5. Tune the hyperparameters

```
def tune_hyperparameters(self):
    grid_search = GridSearchCV(self.model, self.param_grid, cv=5)
    grid_search.fit(self.X_train, self.y_train)
    self.best_model = grid_search.best_estimator_
```

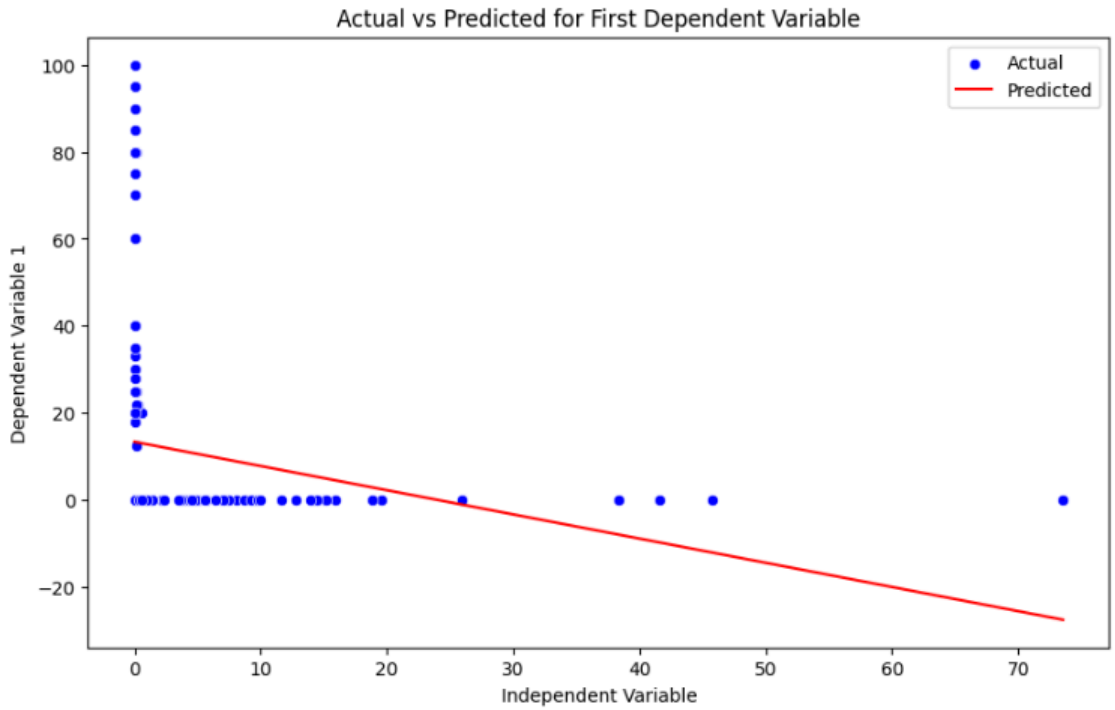
6. Train model

```
def train(self):
```

```
self.best_model.fit(self.X_train, self.y_train)
```

7. Evaluation and Visualization

```
R2 Score: 0.1069
Adjusted R2 Score: 0.1009
Mean Squared Error: 158.5635
```



The model explains approximately 11% of the variance in the target variable. While the performance is relatively low, this is expected in cases where:

- Important features might be missing,
- The relationship between features and the target is complex or non-linear,
- The model might be underfitting the data.
- The Adjusted R^2 is slightly lower than the R^2, indicating that adding more features has not significantly improved the model's performance after accounting for complexity.

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

- Missing Features: Key factors like interior conditions, recent renovations, and market trends are not included.

- Noisy Data: Real-world housing prices are influenced by unpredictable factors, adding noise.
- Non-linear Relationships: The model may not fully capture complex, non-linear patterns in the data.
- Outliers: Extreme property values make the data harder to model accurately.
- Bias-Variance Tradeoff: Higher polynomial degrees risk overfitting, while simpler models underfit.

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

ANS:
The Wine Quality dataset from Kaggle includes several key physicochemical features that influence the quality of wine, such as fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free and total sulfur dioxide, density, pH, sulphates, alcohol.

Each of these features plays a role in determining the taste, stability, and preservation of the wine. For instance, high levels of volatile acidity often lower the quality, while alcohol content and sulphates generally show a positive correlation with better wine ratings.

During the feature engineering process, handling missing data is crucial for model performance. Although the original dataset is relatively clean, if missing values are present, the following techniques can be used:

1. Remove Missing Data:
Delete rows or columns with missing values.
Best when only a small portion of data is missing.

2. Fill with Mean:
Replace missing values with the column's average.
Works well for normally distributed numerical data.

3. Fill with Median:
Use the middle value of the column for imputation.
Better than mean when data has outliers or is skewed.

5. Fill with Constant:
Replace missing data with a fixed value like 0 or "unknown."
Helps retain rows while signaling missingness.

Advantages and disadvantages of different imputation techniques

1. Mean Imputation

Advantage: Simple to implement and works well with normally distributed data.
Disadvantage: Sensitive to outliers and may distort the overall variance.

2. Median Imputation

Advantage: More robust than mean; not affected by outliers.
Disadvantage: Does not consider relationships between different features.

3. Mode Imputation

Advantage: Ideal for categorical data; retains the most common category.
Disadvantage: Can lead to overrepresentation of one category.

4. Constant Value Imputation

Advantage: Easy to implement and helps highlight missing data.
Disadvantage: May introduce meaningless or misleading values.