Name: Mohit Kerkar **Div: D15C** **Roll No: 23**

AIDS Lab
All docs combined

<u>AIDS Exp-01</u>
Aim: Introduction to Data science and Data preparation using Pandas steps.

(I have performed this experiment on Python Idle on my Local machine and not on google colab. The output screenshots of my dataset are all run using local python interpreter)

Theory: Data science is an interdisciplinary field that involves the extraction of meaningful insights from structured and unstructured data using scientific methods, processes, algorithms, and systems. One of the primary steps in any data science project is data preparation, which includes cleaning, transforming, and organizing raw data to ensure its quality and usability for analysis.
In this experiment, we explore data preprocessing techniques using the Pandas library in Python. The dataset under consideration contains records of car accidents in NYC in 2020, with key features such as the number of people injured, number of people killed, latitude, longitude, contributing factors, and vehicle types involved. The dataset initially had missing values, inconsistent entries, and redundant columns, necessitating thorough cleaning and preprocessing to enhance its quality.
The following key steps were performed in this experiment:

1. Loading the dataset into Pandas.
2. Identifying and handling missing values.
3. Eliminating redundant columns.
4. Encoding categorical variables using ordinal encoding.
5. Identifying and handling outliers.
6. Standardizing and normalizing numerical features.

Loading data into pandas:
import pandas as pd
df = pd.read_csv(r"C:\Users\Dell\Desktop\car_accidents.csv")

```
>>> df.info()
<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
>>>
```

From the above image, we are able to infer that there are 29 columns in the dataset. We have 74881 entries i.e rows. Corresponding to each column, the output to the command df.info() indicates the amount of non-null values throughout the dataset. To help us analyse our dataset more effectively, we are to eliminate the columns that have negligible/lesser amount of non-null (significant) values. In the above image, we see that the columns ‘CONTRIBUTING FACTOR VEHICLE 3,4,5’ are not having a lot of significant values meaning they do not contribute to our research much. Similar thing can be said about ‘VEHICLE TYPE CODE 3,4,5’.

Drop columns that are not useful:

import pandas as pd
df = pd.read_csv(r"C:\Users\Dell\Desktop\car_accidents.csv")
cols = ['CONTRIBUTING FACTOR VEHICLE 3', 'CONTRIBUTING FACTOR VEHICLE 4', 'CONTRIBUTING FACTOR VEHICLE 5','VEHICLE TYPE CODE 3','VEHICLE TYPE CODE 4','VEHICLE TYPE CODE 5']
df = df.drop(cols, axis=1)

```
======================================================== RESTART: C:
>>> df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 74881 entries, 0 to 74880
Data columns (total 23 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  COLLISION_ID                   74881 non-null  int64
 21  VEHICLE TYPE CODE 1            74246 non-null  object
 22  VEHICLE TYPE CODE 2            53638 non-null  object
dtypes: float64(3), int64(9), object(11)
memory usage: 13.1+ MB
>>>
```

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

Dropping rows with missing values:

df = df.dropna()

```
memory usage: 8.5+ MB
>>> df = df.dropna()
>>> df.info()
<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
>>>
```

What the above command does is… It eliminates the rows that have at least one value that is NaN i.e Not a Number.
Due to which, considering that there might be a possibility of our dataset having every row with at least one NaN, the entire entries in each column of the entire dataset gets dropped.
To avoid this, we have to modify the command a bit smartly and keep a threshold amount beyond which we would eliminate the rows
There is a parameter called thresh which is used to specify the number of non-NaNs required in a row to be intact and prevalent in the dataset.
By keeping **thresh=21**, the problem that we resolved is that we require rows that have at least 21 significant value providing rows

df = df.dropna(thresh=21)

```
==================== RESTART: C:\Users\Dell\Desktop\trial.py ==
>>> df = df.dropna(thresh=21)
>>> df.info()
<class 'pandas.core.frame.DataFrame'>
Index: 35143 entries, 0 to 74880
Data columns (total 23 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   CRASH DATE                     35143 non-null  object
 1   CRASH TIME                     35143 non-null  object
 2   BOROUGH                        35143 non-null  object
 3   ZIP CODE                       35138 non-null  float64
 4   LATITUDE                       35143 non-null  float64
 5   LONGITUDE                      35143 non-null  float64
 6   LOCATION                       35143 non-null  object
 7   ON STREET NAME                 23959 non-null  object
 8   CROSS STREET NAME              23950 non-null  object
 9   OFF STREET NAME                11184 non-null  object
 10  NUMBER OF PERSONS INJURED      35143 non-null  int64
 11  NUMBER OF PERSONS KILLED       35143 non-null  int64
 12  NUMBER OF PEDESTRIANS INJURED  35143 non-null  int64
 13  NUMBER OF PEDESTRIANS KILLED   35143 non-null  int64
 14  NUMBER OF CYCLIST INJURED      35143 non-null  int64
 15  NUMBER OF CYCLIST KILLED       35143 non-null  int64
 16  NUMBER OF MOTORIST INJURED     35143 non-null  int64
 17  NUMBER OF MOTORIST KILLED      35143 non-null  int64
 18  CONTRIBUTING FACTOR VEHICLE 1  35143 non-null  object
 19  CONTRIBUTING FACTOR VEHICLE 2  34918 non-null  object
 20  COLLISION_ID                   35143 non-null  int64
 21  VEHICLE TYPE CODE 1            35143 non-null  object
 22  VEHICLE TYPE CODE 2            32907 non-null  object
dtypes: float64(3), int64(9), object(11)
memory usage: 6.4+ MB
>>>
```

Another observation on our dataset is that beyond thresh=22, we dont have any row that has more number of non-NaNs

df = df.dropna(thresh=23)

```
>>> df = df.dropna(thresh=23)
>>> df.info()
<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
```

Create Dummy Variables:

Name: Mohit Kerkar **Div: D15C** **Roll No: 23**

Identify the cardinality of values within the columns. If the cardinality is high, it means that the values within the columns are unique and do not repeat. To create dummy variables, we need columns with repeating values and would want to eliminate these columns by creating multiple sub-columns with binary values.
To check the cardinality of each column, we need to run ‘df.nunique()’.

After getting to know the cardinality of all the columns, i decided to go ahead with columns low unique values within them

The technique to create Dummy variables is called Ordinal encoding, which categorizes the values within the columns, converts them from textual data to numeric values. This helps in standardizing and normalizing the dataset which can further lead to better results.

This is the code snippet used to create 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)
```

Finding out missing values and interpolating them

```
Data columns (total 27 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   CRASH DATE                      35143 non-null  object
 1   CRASH TIME                      35143 non-null  object
 2   BOROUGH                         35143 non-null  object
 3   ZIP CODE                        35143 non-null  float64
 4   LATITUDE                        35143 non-null  float64
 5   LONGITUDE                       35143 non-null  float64
 6   LOCATION                        35143 non-null  object
 7   ON STREET NAME                  35143 non-null  object
 8   CROSS STREET NAME               35143 non-null  object
 9   OFF STREET NAME                 35143 non-null  object
 10  NUMBER OF PERSONS INJURED       35143 non-null  int64
 11  NUMBER OF MOTORIST INJURED      35143 non-null  int64
 12  CONTRIBUTING FACTOR VEHICLE 1   35143 non-null  object
 13  CONTRIBUTING FACTOR VEHICLE 2   35143 non-null  object
 14  COLLISION_ID                    35143 non-null  int64
 15  VEHICLE TYPE CODE 1             35143 non-null  object
 16  VEHICLE TYPE CODE 2             35143 non-null  object
 17  1 Person Killed                 35143 non-null  bool
 18  3 Persons Killed                35143 non-null  bool
 19  1 Pedestrian Injured            35143 non-null  bool
 20  2 Pedestrians Injured           35143 non-null  bool
 21  1 Pedestrian Killed             35143 non-null  bool
 22  1 Cyclist Injured               35143 non-null  bool
 23  2 Cyclists Injured              35143 non-null  bool
 24  1 Cyclist Killed                35143 non-null  bool
 25  1 Motorist Killed               35143 non-null  bool
 26  3 Motorists Killed              35143 non-null  bool
dtypes: bool(10), float64(3), int64(3), object(11)
memory usage: 5.2+ MB
>>>
```

Finding Outliers

Outliers are data points that significantly differ from other observations in the dataset. They may arise due to data entry errors, measurement variations, or genuine rare events. Detecting and handling outliers is crucial because they can distort statistical analyses and impact model performance.

Identifying Outliers

A common method to detect outliers is using the Interquartile Range (IQR), which measures the spread of the middle 50% of the data. The IQR is calculated as:

where Q1 and Q3 represent the first and third quartiles, respectively. A data point is considered an outlier if it falls below or above .

Code for detecting outliers:

```
import numpy as np

# Define a function to detect outliers using IQR
def detect_outliers_iqr(data, column):
    Q1 = data[column].quantile(0.25)
    Q3 = data[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
```

```
    return data[(data[column] < lower_bound) | (data[column] > upper_bound)]

# Apply the function to relevant numerical columns
outlier_columns = [
    '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'
]

for col in outlier_columns:
    outliers = detect_outliers_iqr(df, col)
    print(f"Outliers detected in {col}: {len(outliers)}")
```

Handling Outliers

After detecting outliers, we have several options to handle them:

1. **Removal**: If the outliers are due to data entry errors, we can remove them.
2. **Transformation**: Applying log or square root transformations can reduce their impact.
3. **Capping**: Setting a cap on extreme values based on domain knowledge.
4. **Imputation**: Replacing outliers with the median or mean of the data.

In this experiment, capping extreme values using the IQR method was chosen:

```
# Capping outliers to the upper and lower bounds
for col in outlier_columns:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    df[col] = np.where(df[col] > upper_bound, upper_bound, np.where(df[col] < lower_bound, lower_bound, df[col]))
```

By capping extreme values, we maintain the integrity of the dataset while ensuring that outliers do not skew analysis or modeling outcomes.

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

Name: Mohit Kerkar **Div: D15C** **Roll No: 23**

```
# 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	NU
2	2020-08-29	15:40:00	BRONX	10466	40.8921	-73.83376	POINT (-73.8337	PRATT AVENUE	STRANG AVENUE		0	0	0	0	0	0	
3	2020-08-29	21:00:00	BROOKLYN	11221	40.6905	-73.919914	POINT (-73.9199	BUSHWICK AVE	PALMETTO STREET		2	0	0	0	0	0	
4	2020-08-29	18:20:00			40.8165	-73.946556	POINT (-73.9465	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 (-73.8449	WOODHAVEN BOULEVARD			1	0	0	0	0	0	
8	2020-08-29	19:30:00	BRONX	10459	40.825226	-73.88778	POINT (-73.8877	LONGFELLOW	EAST 165 STREET		0	0	0	0	0	0	
9	2020-08-29	0:00:00			40.80016	-73.93538	POINT (-73.9353	2 AVENUE			0	0	0	0	0	0	
10	2020-08-29	19:50:00	BRONX	10466	40.894314	-73.86027	POINT (-73.8602	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 (-73.7928	ARCHER AVEN	MERRICK BOULEVARD		0	0	0	0	0	0	
14	2020-08-29	21:33:00	BRONX	10455	40.812965	-73.9161	POINT (-73.9161	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 (-73.9614	WILLIAMSBURG	WYTHE AVENUE		0	0	0	0	0	0	
16	2020-08-29	4:14:00			40.835373	-73.842186	POINT (-73.8421	WATERBURY AVENUE			1	0	0	0	0	0	
17	2020-08-29	6:35:00			40.65965	-73.773834	POINT (-73.7738	ROCKAWAY BO	NASSAU EXPRESSWAY		0	0	0	0	0	0	
18	2020-08-29	13:00:00	BROOKLYN	11206	40.699707	-73.95718	POINT (-73.9571	BEDFORD AVEN	WALLABOUT STREET		0	0	0	0	0	0	
19	2020-08-29	10:30:00	QUEENS	11385	40.7122	-73.86208	POINT (-73.8620	METROPOLITAN	COOPER AVENUE		2	0	0	0	0	0	
20	2020-08-29	12:29:00	BRONX	10453	40.861862	-73.91282	POINT (-73.9128	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 (-73.9511	UNION AVENUE	GRAND STREET		1	0	0	0	0	0	
22	2020-08-29	13:55:00	BROOKLYN	11231	40.67473	-74.00029	POINT (-74.0002	HAMILTON AVE	GARNET STREET		1	0	0	0	0	0	
23	2020-08-29	0:30:00			40.66584	-73.75551	POINT (-73.7555	BELT PARKWAY			0	0	0	0	0	0	
24	2020-08-29	6:30:00			40.65052	-73.73309	POINT (-73.7330	CRAFT AVENUE			0	0	0	0	0	0	
25	2020-08-29	19:00:00			40.83968	-73.929276	POINT (-73.9292	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:

Name: Mohit Kerkar **Div: D15C** **Roll No: 23**

	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 STREET	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	0.005602711712	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 (-73.9199	BUSHWICK AVE	PALMETTO STREET		0.2	0	0	0	0	0
4	2020-08-29	0:00:00	0	10459	0.9978450798	0.004805402738	POINT (-73.89296 40.82472)			1047 SIMPSON	0	0	0	0	0	0
5	2020-08-29	17:10:00	1	11203	0.9935718538	0.004254155165	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	0.005245673521	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	0.006087831126	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	0.006153636052	POINT (-73.7928	ARCHER AVEN	MERRICK BOULEVARD		0	0	0	0	0	0
9	2020-08-29	13:00:00	1	11206	0.9947894898	0.003940484117	POINT (-73.9571	BEDFORD AVEI	WALLABOUT STREET		0	0	0	0	0	0
10	2020-08-29	10:30:00	3	11385	0.9950948459	0.005221296338	POINT (-73.8620	METROPOLITAI	COOPER AVENUE		0.2	0	0	0	0	0
11	2020-08-29	12:29:00	0	10453	0.9987529112	0.004537927126	POINT (-73.9128	WEST FORDHA	MAJOR DEEGAN EXPRESSWAY		0.2	0	0	0	0	0
12	2020-08-29	10:35:00	1	11211	0.9950644643	0.004022019734	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	0.00335987618	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	0.003972942135	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	0.005409903008	POINT (-73.8480	EAST TREMON	SILVER STREET		0.1	0	0	0	0	0
16	2020-08-29	22:11:00	1	11229	0.9925657404	0.003983043178	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 STI	0.1	0	0	0	0	0
18	2020-08-29	5:40:00	0	10458	0.9986631595	0.004921362709	POINT (-73.8843	EAST FORDHAI	HUGHES AVENUE		0	0	0	0	0	0
19	2020-08-29	19:50:00	0	10457	0.9985058252	0.004852877639	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	0.004402842514	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	0.004635435859	POINT (-73.9055	JEROME AVEN	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	0.004131730527	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	0.003921426817	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	0.004639880317	POINT (-73.90525 40.64323)			9312 GLENWOO	0	0	0	0	0	0
26	2020-08-29	11:10:00	3	11105	0.9965773618	0.004628836511	POINT (-73.9060	STEINWAY STR	DITMARS BOULEVARD		0	0	0	0	0	0
27	2020-08-29	15:41:00	1	11234	0.9927816382	0.004509509526	POINT (-73.9149	AVENUE T	EAST 63 STREET		0	0	0	0	0	0

car_accidents_transformed.csv

Conclusion: The experiment focused on cleaning and preprocessing a dataset containing records of car accidents in NYC (2020) using Pandas. Initially, the dataset had missing values, redundant columns, and categorical data that required transformation for effective analysis. To address these issues, data cleaning techniques were applied by removing columns with a high percentage of missing values and filtering out incomplete rows using a threshold-based approach. Categorical variables were encoded using ordinal encoding to convert textual data into numerical values, ensuring consistency for further processing. Additionally, numerical features were standardized using StandardScaler to maintain a mean of 0 and a standard deviation of 1, followed by normalization with MinMaxScaler to scale values between 0 and 1. After these transformations, the dataset was structured and refined, eliminating inconsistencies and making it suitable for further analysis. This preprocessing ensures that any subsequent data-driven insights or modeling efforts are more accurate and reliable.

AIDS Exp 02

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

Theory:

Data visualization is a fundamental step in exploratory data analysis (EDA) that allows us to uncover patterns, trends, and insights in a dataset. In this experiment, we analyze a dataset containing records of car accidents in NYC (2020) using Matplotlib and Seaborn to create different types of visualizations.

The key objectives of this experiment are:

1. To represent categorical and numerical data using bar graphs and contingency tables.
2. To identify relationships between variables using scatter plots, box plots, and heatmaps.
3. To understand distributions and frequencies through histograms and normalized histograms.
4. To detect and handle outliers using box plots and the Interquartile Range (IQR) method.

By performing these visualizations, we aim to extract meaningful insights about accident severity, injury patterns, borough-wise trends, and factors influencing accident outcomes.

Name: Mohit Kerkar **Div: D15C** **Roll No: 23**

1. Bar Graph: Number of Persons Injured vs. Borough

Theory

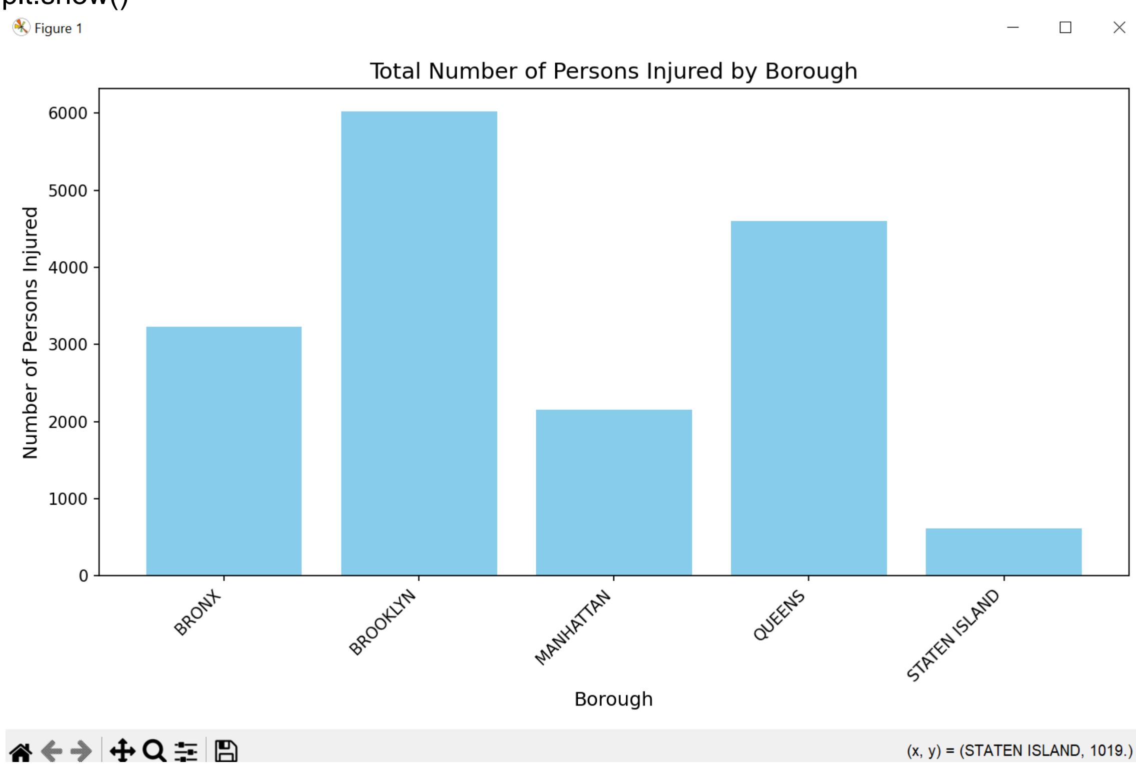
A bar graph is useful for comparing categorical data. Here, we visualize the number of persons injured across different boroughs in NYC. This allows us to identify which boroughs have the highest accident severity.

Insight:

- Boroughs like Brooklyn and Manhattan might have more injuries due to higher traffic density.
- Boroughs like Staten Island might show fewer injuries due to lower traffic volume.

Code for Bar Graph

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

df = pd.read_csv(r"C:\Users\Dell\Desktop\car_accidents.csv")

plt.figure(figsize=(10, 6))
sns.barplot(x=df['BOROUGH'], y=df['NUMBER OF PERSONS INJURED'], estimator=sum, ci=None)
plt.xlabel("Borough")
plt.ylabel("Total Number of Persons Injured")
plt.title("Number of Persons Injured per Borough")
plt.xticks(rotation=45)
plt.show()
```



Name: Mohit Kerkar Div: D15C Roll No: 23

Observations:

- Brooklyn and Manhattan have significantly higher injury counts than Staten Island.
- Denser traffic areas tend to have more severe accidents.
- Some boroughs have high accident numbers but fewer injuries, possibly due to better safety measures.

2. Contingency Table: Borough vs. Number of Persons Injured

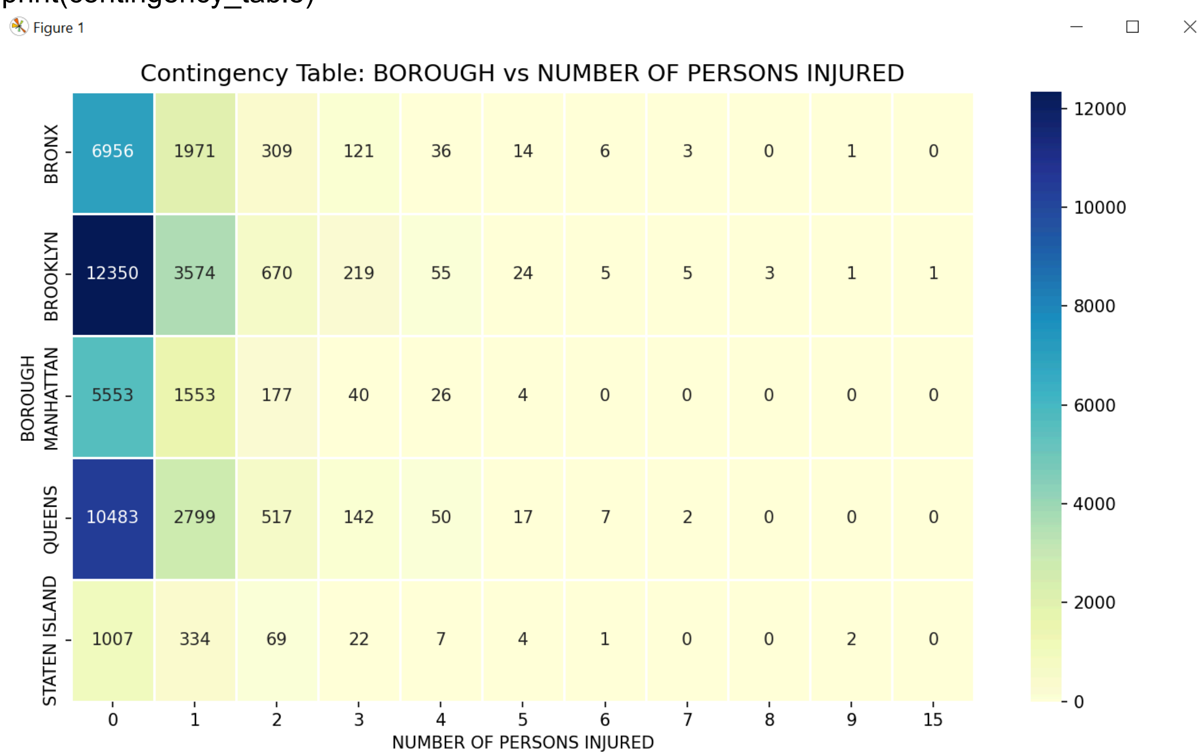
Theory

A contingency table helps analyze the relationship between two categorical/numerical variables. Here, we analyze how many persons were injured in each borough.
Insight:

- This table helps us compare accident severity across boroughs.
- It may indicate whether certain boroughs have a higher risk of severe accidents.

Code for Contingency Table

```
contingency_table = pd.crosstab(df['BOROUGH'], df['NUMBER OF PERSONS INJURED'])
print(contingency_table)
```



Observations:

- Certain boroughs have consistently high injuries across different severity levels.
- Brooklyn and the Bronx show more multi-injury accidents compared to Staten Island.
- Some boroughs may have riskier driving conditions or more pedestrian involvement.

Name: Mohit Kerkar **Div: D15C** **Roll No: 23**

3. Scatter Plot: ZIP Code vs. Number of Persons Injured

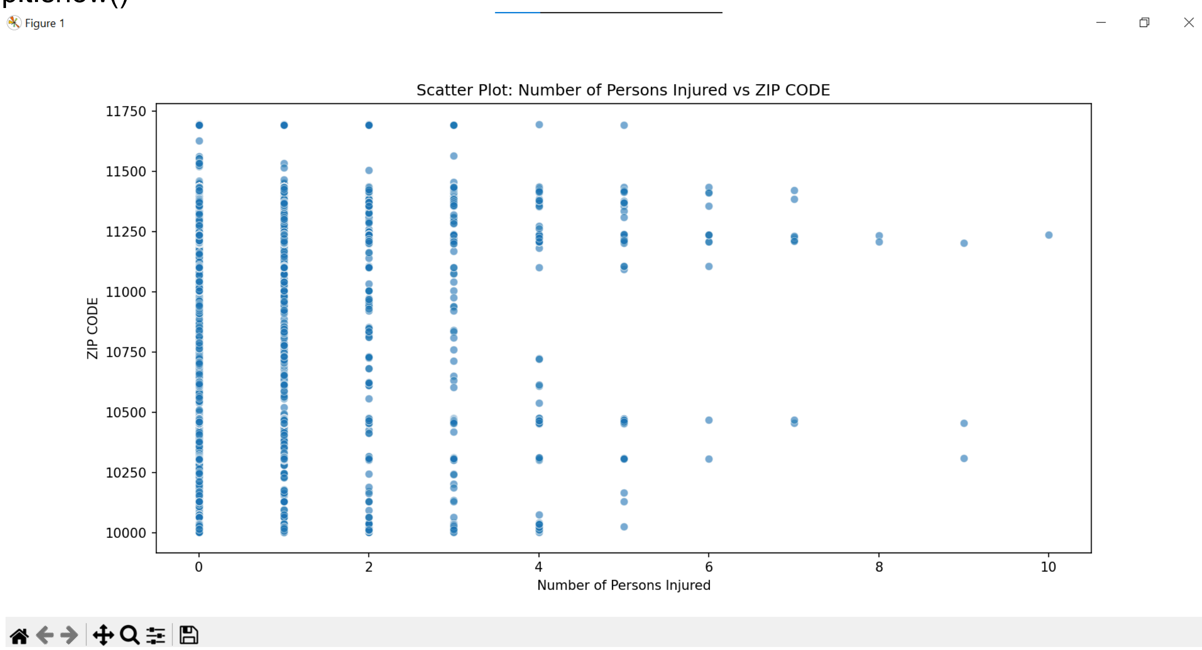
Theory

A scatter plot helps identify trends between two numerical variables. Here, we analyze the relationship between ZIP codes (locations of accidents) and the number of persons injured.
Insight:

- If certain ZIP codes have high injuries, it suggests accident-prone areas.
- Clustering indicates regions where accidents frequently occur.

Code for Scatter Plot

```
plt.figure(figsize=(10, 6))
sns.scatterplot(x=df['ZIP CODE'], y=df['NUMBER OF PERSONS INJURED'])
plt.xlabel("ZIP Code")
plt.ylabel("Number of Persons Injured")
plt.title("Scatter Plot of ZIP Code vs. Number of Persons Injured")
plt.show()
```



Observations:

- Accidents are clustered in specific high-risk ZIP codes.
- Some ZIP codes have fewer injuries despite being in busy areas, possibly due to better infrastructure.
- Outliers indicate locations where injuries were much higher than expected.

4. Box Plot: Number of Persons Injured vs. Vehicle Type Code

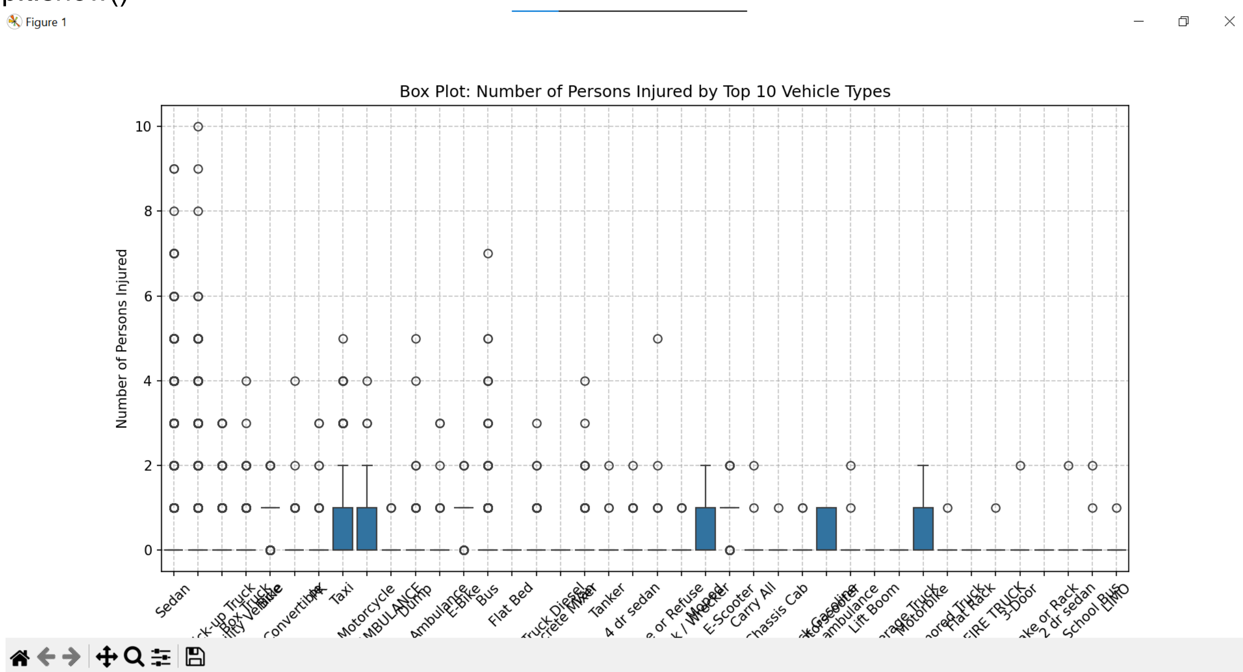
Theory

A box plot is used to identify distributions and outliers. Here, we compare vehicle types with the number of injuries.
Insight:

- Certain vehicle types may be associated with higher accident severity.
- Outliers indicate rare but severe accidents.

Code for Box Plot

```
plt.figure(figsize=(12, 6))
sns.boxplot(x=df['VEHICLE TYPE CODE 1'], y=df['NUMBER OF PERSONS INJURED'])
plt.xlabel("Vehicle Type Code")
plt.ylabel("Number of Persons Injured")
plt.title("Box Plot of Vehicle Type vs. Number of Persons Injured")
plt.xticks(rotation=90)
plt.show()
```



Observations:

- Larger vehicles (trucks, SUVs) tend to have higher injury counts.
- Sedans have a more consistent range of injuries.
- Certain vehicle types show a wider spread, meaning accident severity varies significantly.
- Presence of outliers suggests some vehicles are involved in unusually severe accidents.

5. Heat Map Using Seaborn: Correlation Between Numeric Features

Theory

A heatmap helps visualize correlations between numeric features. A correlation closer to:

- +1: Strong positive correlation (e.g., number of persons injured and number of vehicles involved).
- -1: Strong negative correlation.
- 0: No correlation.

Insight:

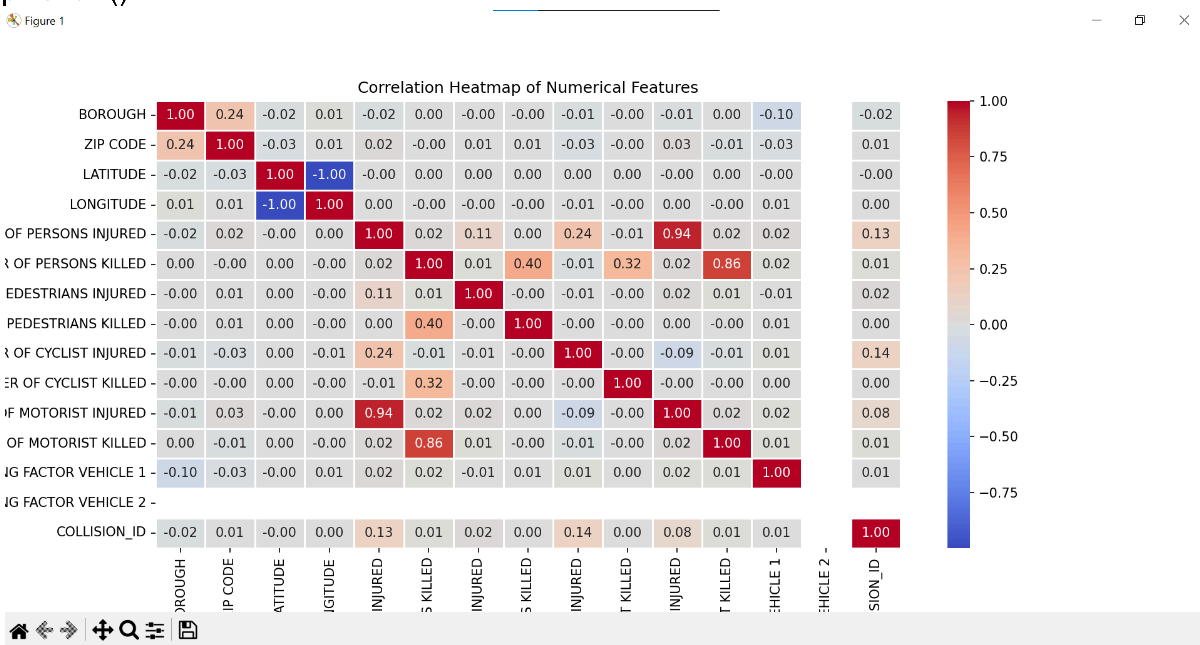
- Helps identify variables that significantly impact accident severity.

Name: Mohit Kerkar **Div: D15C** **Roll No: 23**

- E.g., NUMBER OF PERSONS INJURED may be highly correlated with NUMBER OF VEHICLES INVOLVED.

Code for Heatmap

```
plt.figure(figsize=(10, 6))
sns.heatmap(df.corr(), annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Heatmap of Numeric Features")
plt.show()
```



Observations:

- The number of vehicles involved has a strong correlation with the number of persons injured.
- Some features have little to no correlation with injuries, contradicting initial assumptions.
- Helps focus on factors that truly influence accident severity.

6. Histogram: Number of Persons Injured vs. Frequency

Theory

A histogram helps visualize the distribution of a single variable. Here, we check the frequency of different injury counts per accident.

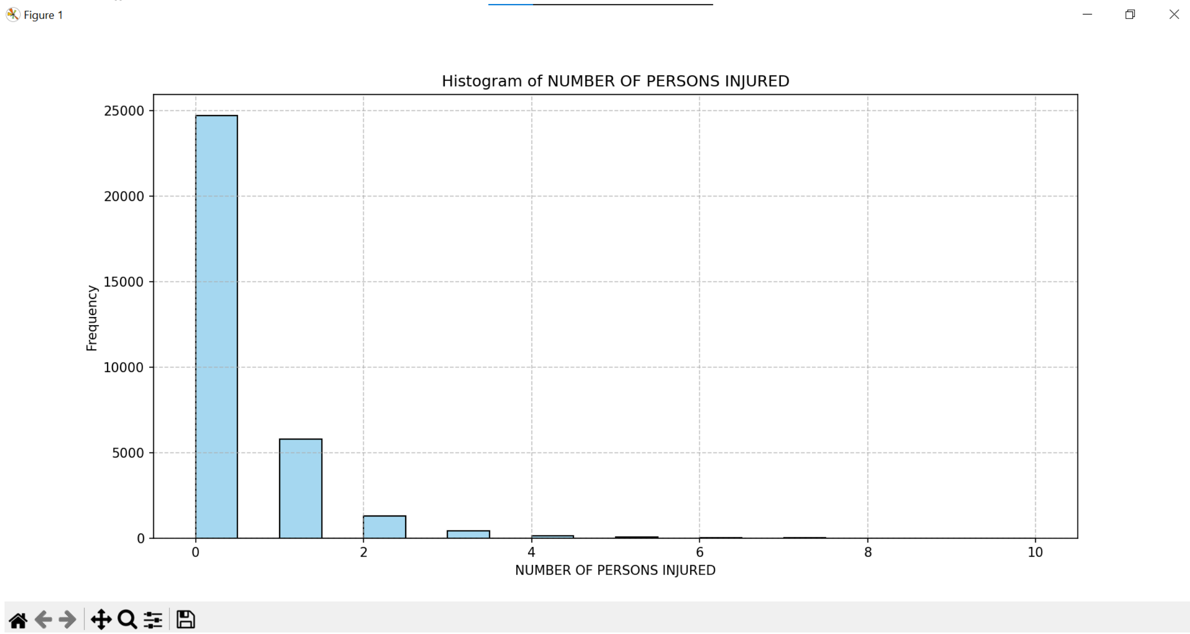
Insight:

- A peak at 0 or 1 indicates most accidents result in minimal injuries.
- A long tail indicates occasional severe accidents.

Code for Histogram

```
plt.figure(figsize=(10, 6))
sns.histplot(df['NUMBER OF PERSONS INJURED'], bins=20, kde=False)
plt.xlabel("Number of Persons Injured")
plt.ylabel("Frequency")
plt.title("Histogram of Number of Persons Injured")
```

plt.show()



Observations:

- Most accidents result in only 1 or 2 injuries.
- Severe multi-injury accidents are rare but still occur.
- A long right tail indicates occasional extreme injury cases.
- Highlights the importance of reducing accident severity, not just frequency.

7. Normalized Histogram: Number of Persons Injured vs. Density

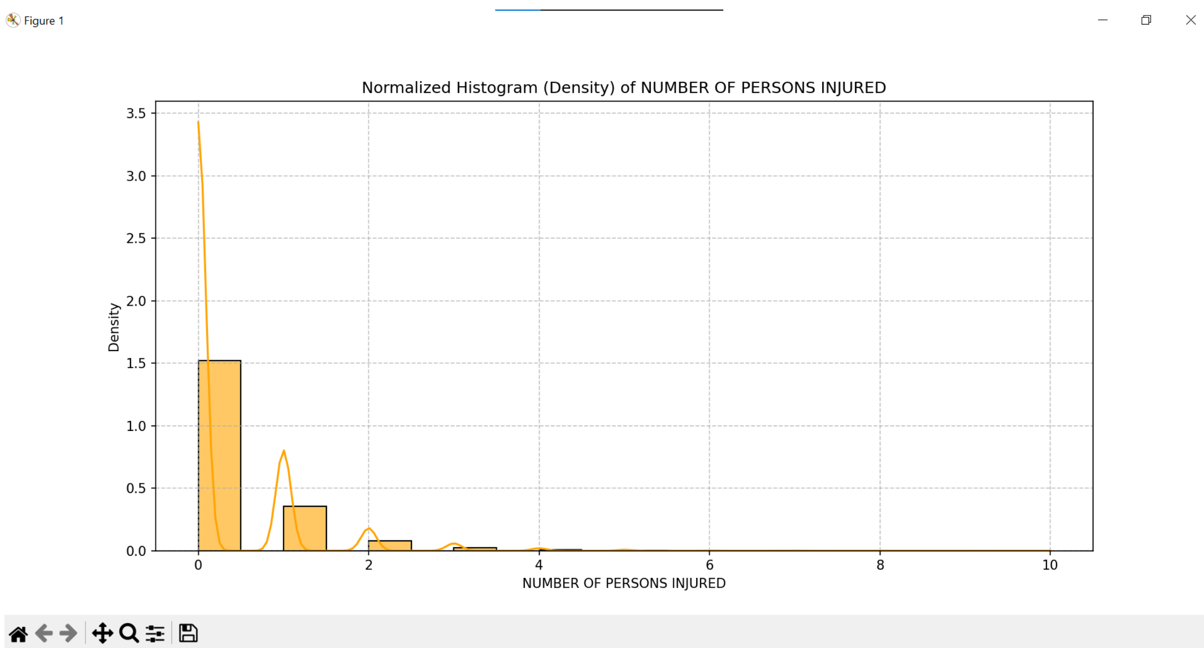
Theory

A normalized histogram (density plot) helps compare distributions across different sample sizes.
Insight:

- Useful for understanding the probability distribution of injury severity.
- Helps compare accident patterns across locations.

Code for Normalized Histogram

```
plt.figure(figsize=(10, 6))
sns.histplot(df['NUMBER OF PERSONS INJURED'], bins=20, kde=True, stat="density")
plt.xlabel("Number of Persons Injured")
plt.ylabel("Density")
plt.title("Normalized Histogram of Number of Persons Injured")
plt.show()
```



Observations:

- Majority of accidents result in 0-2 injuries with high probability.
- Probability of accidents causing more than 5 injuries is extremely low.
- Useful for comparing distributions across different sample sizes.

8. Handling Outliers Using Box Plot and IQR

Theory

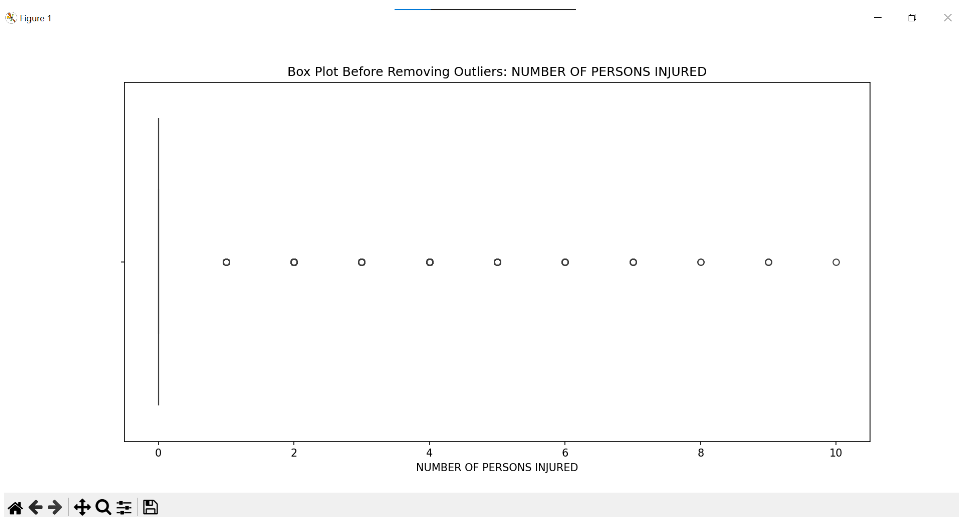
The Interquartile Range (IQR) method is used to detect and remove outliers.
Calculation:

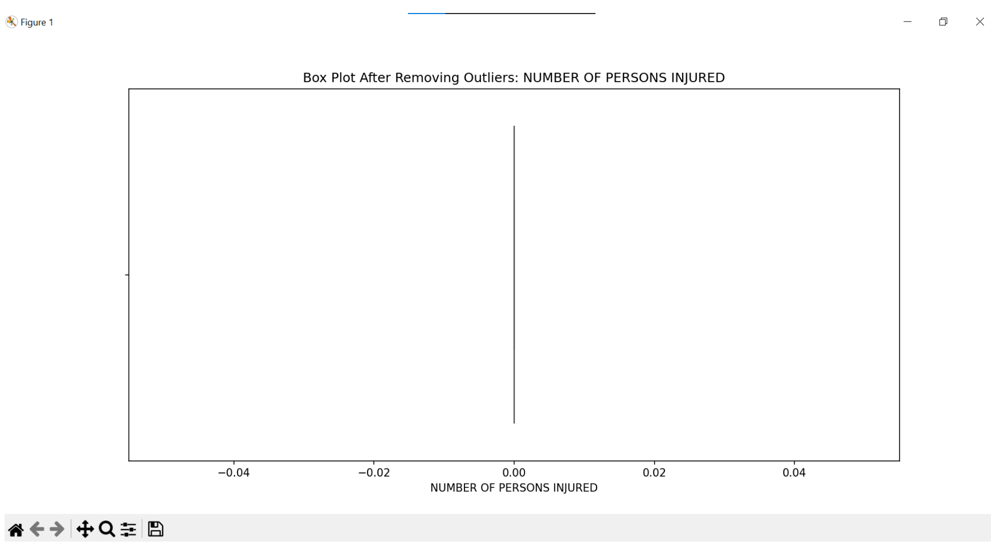
- IQR = Q3 - Q1
- Lower Bound = Q1 - 1.5 × IQR
- Upper Bound = Q3 + 1.5 × IQR
- Values outside this range are outliers.

Code for Outlier Removal

```
Q1 = df['NUMBER OF PERSONS INJURED'].quantile(0.25)
Q3 = df['NUMBER OF PERSONS INJURED'].quantile(0.75)
IQR = Q3 - Q1

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

df = df[(df['NUMBER OF PERSONS INJURED'] >= lower_bound) & (df['NUMBER OF PERSONS INJURED'] <= upper_bound)]
```

Name: Mohit Kerkar **Div: D15C** **Roll No: 23**





Observations:

- Some accidents had extremely high injury counts, affecting the overall dataset.
- IQR method helped identify and remove extreme values.
- Data became more balanced and reliable after outlier removal.
- Ensures more accurate insights and predictions from the dataset.

Conclusion

This experiment explored data visualization techniques to analyze car accident data in NYC. By applying various graphs, we identified:

- High-risk boroughs with frequent injuries.
- Accident-prone ZIP codes through scatter plots.
- Vehicle types contributing to injuries via box plots.
- Correlations among numeric variables using a heatmap.
- Outlier removal using IQR to improve dataset quality.

This analysis prepares the dataset for further modeling and predictions in accident severity assessment.

<u>AIDS Exp 03</u>

Name: Mohit Kerkar **Div: D15C** **Roll No: 23**

Aim: Perform Data Modeling.

Theory:

Data modeling is a fundamental process in data science that involves structuring and preparing data for analysis and predictive modeling. In this experiment, we worked with a car accidents dataset to partition the data into training and testing sets, visualize the partitioning for verification, and perform statistical validation using a two-sample Z-test. These steps are critical to ensuring that the data is ready for further predictive analysis and machine learning applications.

a. **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.

Code:

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

# Load CSV
df = pd.read_csv(r"C:\\Users\\Dell\\Desktop\\car_accidents_no_outliers.csv")

# Partition the dataset (75% training, 25% testing)
train_df, test_df = train_test_split(df, test_size=0.25, random_state=42)

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

```
PS C:\Users\Dell\Desktop\experiments aids>
& C:/Users/Dell/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/Dell/Desktop/experiments aids/exp3.py"
Total records: 32447
Training records: 24335
Testing records: 8112
PS C:\Users\Dell\Desktop\experiments aids>
```

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

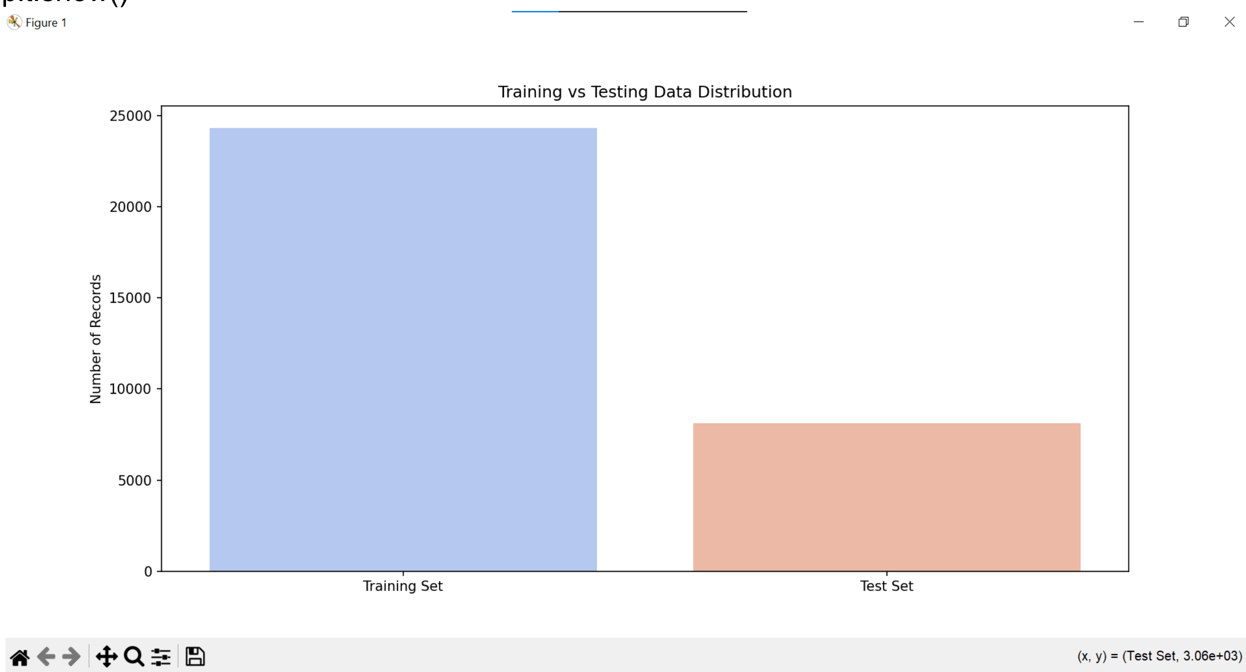
Bar graphs and a Pie Chart to visualize the proportions.

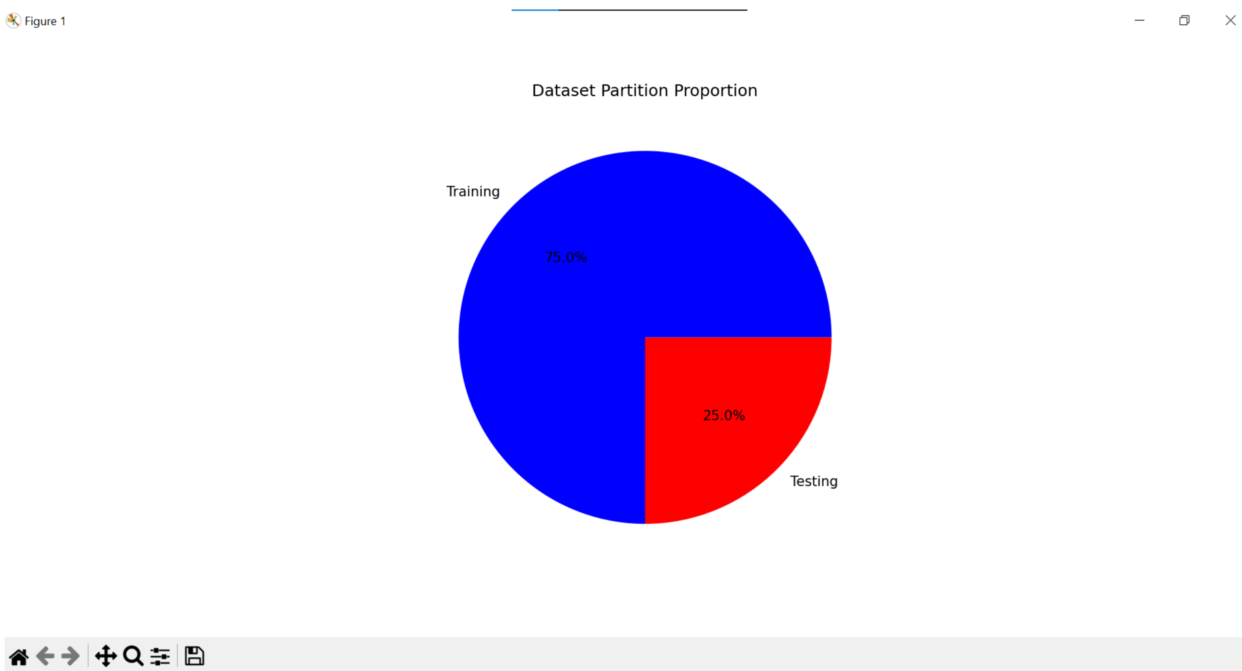
To verify the partitioning, we used a bar graph to visualize the count of records in the training and test sets. This helps in confirming that the split was correctly applied. Other relevant graphs, such as histograms or pie charts, could also be used to compare distributions of key features like accident severity or injury counts between the two datasets.

Name: Mohit Kerkar **Div: D15C** **Roll No: 23**

Code:

```
plt.figure(figsize=(6, 4))
sns.barplot(x=['Training Set', 'Test Set'], y=[len(train_df), len(test_df)], palette='coolwarm')
plt.ylabel("Number of Records")
plt.title("Training vs Testing Data Distribution")
plt.show()

# Pie chart for dataset partition proportions
plt.figure(figsize=(6, 6))
plt.pie([len(train_df), len(test_df)], labels=['Training', 'Testing'], autopct='%1.1f%%',
colors=['blue', 'red'])
plt.title("Dataset Partition Proportion")
plt.show()
```





c. Print training records

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 records: {len(train_df)}")

```
Training records: 24335
```

d. 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.

Code:

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

# Load the dataset
df = pd.read_csv(r"C:\\Users\\Dell\\Desktop\\car_accidents.csv")

# Partition the dataset (75% training, 25% testing)
train_df = df.sample(frac=0.75, random_state=42)  # 75% Training data
```

```
test_df = df.drop(train_df.index)  # Remaining 25% as Testing data

# Display the total number of records
print(f"Total records: {len(df)}")
print(f"Training records: {len(train_df)}")
print(f"Testing records: {len(test_df)}")

# Visualizing partitioning using a bar chart
plt.figure(figsize=(6, 4))
sns.barplot(x=['Training Set', 'Test Set'], y=[len(train_df), len(test_df)], palette='coolwarm')
plt.xlabel("Dataset")
plt.ylabel("Number of Records")
plt.title("Partitioning Verification")
plt.show()

# Perform a Two-Sample Z-test on 'NUMBER OF PERSONS INJURED'
train_injured = train_df['NUMBER OF PERSONS INJURED'].dropna()
test_injured = test_df['NUMBER OF PERSONS INJURED'].dropna()

# Perform the two-sample Z-test
z_score, p_value = ztest(train_injured, test_injured)

# Display Z-test results
print("\nZ-test results for 'NUMBER OF PERSONS INJURED':")
print(f"Z-score: {z_score:.4f}")
print(f"P-value: {p_value:.4f}")

# Interpretation
if p_value > 0.05:
    print("No significant difference between training and testing sets. Partitioning is valid.")
else:
    print("Significant difference found. Consider revising the partitioning strategy.")
```

```
PS C:\Users\Dell\Desktop\experiments aids> & C:/Users/Dell/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/Dell/Desktop/experiments aids/exp3.py"

Z-test results for 'NUMBER OF PERSONS INJURED':
Z-score: -0.1176
P-value: 0.9064
No significant difference between training and testing sets. Partitioning is valid.
PS C:\Users\Dell\Desktop\experiments aids>
```

Conclusion:
This experiment highlighted the importance of data partitioning, visualization, and statistical validation in preparing a dataset for analysis. Through proper partitioning, graphical verification, and statistical testing, we ensured that our car accidents dataset remained representative and unbiased. These foundational steps are crucial in building reliable models, allowing us to derive accurate insights and predictions for accident analysis and prevention strategies.

Name: Mohit Kerkar **Div: D15C** **Roll No: 23**

AIDS Exp 04

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

Theory:

Correlation Analysis of AQI Dataset

1. Introduction

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
17	19-01-2009	22	68	190			160	MPCB-KR
18	20-01-2009	20	61	194			163	MPCB-KR
19	21-01-2009	20	61	191			161	MPCB-KR
20	22-01-2009	21	67	179			153	MPCB-KR

PNQ_AQI.csv

2. Theoretical Background

2.1 Pearson's Correlation Coefficient (r)

Pearson's correlation measures the linear relationship between two continuous variables. It ranges from -1 to 1:

Formula:

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

Where:

- X_i, Y_i are individual data points
- $\bar{X}, \bar{Y}$ are the means of X and Y
- $\sum$ represents summation

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

Significance:

- Determines the strength and direction of the relationship.
- Requires normally distributed data.

2.2 Spearman's Rank Correlation (ρ)

Spearman's correlation measures the monotonic relationship between two variables based on their ranked values.

- Works for both linear and nonlinear relationships.
- Less sensitive to outliers.

Significance:

- Useful when data is not normally distributed.
- Helps identify whether an increase in one variable generally corresponds to an increase in another.

Formula:

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

Where:

- d_i = difference between ranks of corresponding values
- n = number of data points

Interpretation:

- $\rho = 1$ → Perfect positive monotonic relationship
- $\rho = -1$ → Perfect negative monotonic relationship
- $\rho \approx 0$ → No monotonic relationship

2.3 Kendall's Rank Correlation (τ)

Kendall's Tau is similar to Spearman's correlation but focuses on the ordinal association between two variables. It compares the number of concordant and discordant pairs.

Significance:

- Measures how well the ranks of one variable correspond to the ranks of another.
- More robust for small datasets.

Formula:

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

Where:

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

Interpretation:

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

2.4 Chi-Squared Test (χ^2)

The Chi-Squared test evaluates whether there is a statistically significant association between two categorical variables.

Significance:

- Helps determine whether AQI levels are dependent on SO_2 concentrations.
- Works for categorical (binned) data.

Formula:

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

Where:

- O_i = Observed frequency
- E_i = Expected frequency under independence assumption

Interpretation:

- If p-value $< \alpha$ (e.g., 0.05), reject the null hypothesis → Variables are dependent
- If p-value $> \alpha$, fail to reject the null hypothesis → No significant relationship

3. Experimental Methodology

Load and Preprocess the Data

```
import pandas as pd
import numpy as np
from scipy.stats import pearsonr, spearmanr, kendalltau, chi2_contingency

# Load dataset
df = pd.read_csv('/content/sample_data/PNQ_AQI.csv', encoding='utf-8')

# Convert relevant columns to numeric
df['SO2 µg/m3'] = pd.to_numeric(df['SO2 µg/m3'], errors='coerce')
df['AQI'] = pd.to_numeric(df['AQI'], errors='coerce')

# Drop NaN values
df = df.dropna()
```

Name: Mohit Kerkar **Div: D15C** **Roll No: 23**

```
                  Date  SO2 µg/m3  Nox µg/m3  RSPM µg/m3  SPM  CO2 µg/m3  \
0  2009-01-01 00:00:00       15.0       53.0       179.0  NaN        NaN
1  2009-02-01 00:00:00       15.0       48.0       156.0  NaN        NaN
2  2009-03-01 00:00:00       13.0       51.0       164.0  NaN        NaN
3  2009-04-01 00:00:00        8.0       37.0       135.0  NaN        NaN
4  2009-07-01 00:00:00       13.0       36.0       140.0  NaN        NaN

     AQI Location             AQI_category SO2_category
0  153.0  MPCB-KR                Unhealthy          Low
1  137.0  MPCB-KR  Unhealthy for Sensitive          Low
2  143.0  MPCB-KR  Unhealthy for Sensitive          Low
3  123.0  MPCB-KR  Unhealthy for Sensitive     Very Low
4  127.0  MPCB-KR  Unhealthy for Sensitive          Low
```

Pearson's Correlation

pearson_corr, pearson_p = pearsonr(df['SO2 µg/m3'], df['AQI'])
print(f"Pearson Correlation: {pearson_corr:.4f}, p-value: {pearson_p:.4f}")

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

Interpretation:

- If $p < 0.05$, the correlation is statistically significant.
- The closer r is to 1 or -1, the stronger the relationship.

Spearman's Rank Correlation

spearman_corr, spearman_p = spearmanr(df['SO2 µg/m3'], df['AQI'])
print(f"Spearman Correlation: {spearman_corr:.4f}, p-value: {spearman_p:.4f}")

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

Interpretation:

- A positive ρ indicates that as SO_2 increases, AQI tends to increase.
- Works well even if the relationship is nonlinear.

Kendall's Rank Correlation

kendall_corr, kendall_p = kendalltau(df['SO2 µg/m3'], df['AQI'])
print(f"Kendall Correlation: {kendall_corr:.4f}, p-value: {kendall_p:.4f}")

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

Interpretation:

- Measures how well ranks match.
- More stable for small datasets.

Chi-Squared Test

Before applying Chi-Square, we categorize SO_2 and AQI into bins:

```
# Categorize SO2 and AQI into bins
df['SO2_category'] = pd.cut(df['SO2 µg/m3'], bins=3, labels=['Low', 'Medium', 'High'])
df['AQI_category'] = pd.cut(df['AQI'], bins=3, labels=['Good', 'Moderate', 'Unhealthy'])

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

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

```
Chi-Squared Statistic: 486.6191, p-value: 0.0000
```

Interpretation:

- If $p < 0.05$, AQI levels are significantly dependent on SO_2.

4. Results & Discussion

Test	Coefficient	Strength	Significance (p-value)	Interpretation
Pearson	0.1868	Weak	0.0000	Weak linear correlation
Spearman	0.1979	Weak	0.0000	Weak monotonic correlation
Kendall	0.1337	Very Weak	0.0000	Weak ordinal correlation
Chi-Square	486.6191	Significant	0.0000	SO_2 significantly impacts AQI

Key Findings:

- Pearson, Spearman, and Kendall correlations show a weak positive relationship between SO_2 and AQI.
- Chi-Square test confirms that AQI depends on SO_2 levels in a statistically significant way.
- SO_2 alone is not a strong predictor of AQI, so other pollutants (NOx, RSPM, etc.) likely play a major role.

5. Conclusion

This experiment involved manually calculating the correlation between SO_2 and AQI using different statistical tests. Through step-by-step computations, we found that while SO_2 has a weak correlation with AQI, the Chi-Square test suggested a significant relationship. However, since AQI is influenced by multiple pollutants, it became evident that SO_2 alone does not determine air quality.
By working through these calculations, we gained a deeper understanding of how different statistical methods reveal relationships between variables. Future manual analyses could focus on NOx, CO_2, and RSPM to further explore their individual effects on AQI and refine our understanding of air pollution dynamics.

AIDS Lab Exp 05
Aim: Perform Regression Analysis using Scipy and Sci-kit learn.
Theory:
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()

Name: Mohit Kerkar **Div: D15C** **Roll No: 23**

```
   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
```

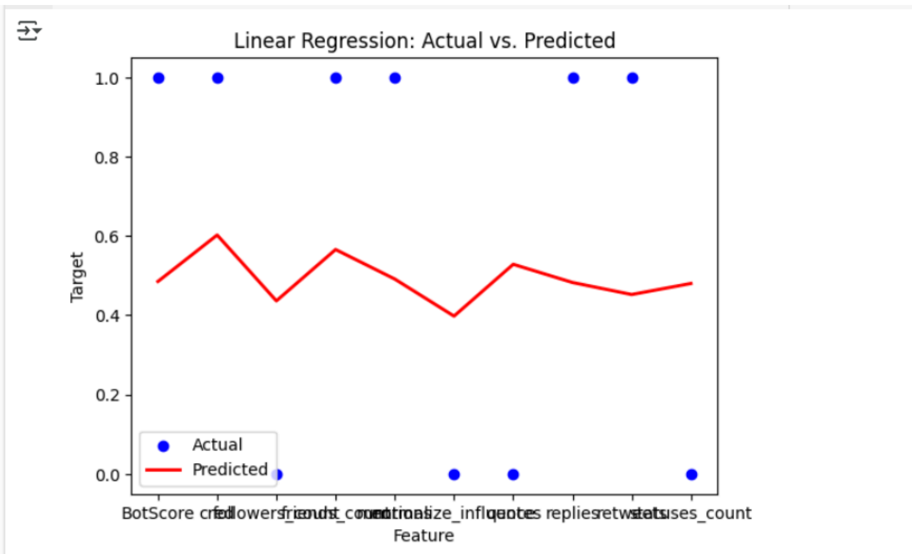
The results show that after converting Linear Regression outputs to binary (0 or 1), the model achieved 56% accuracy.

- 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.

```python
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.

<u>Logistic Regression:</u>

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.

```
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).

```
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))
```

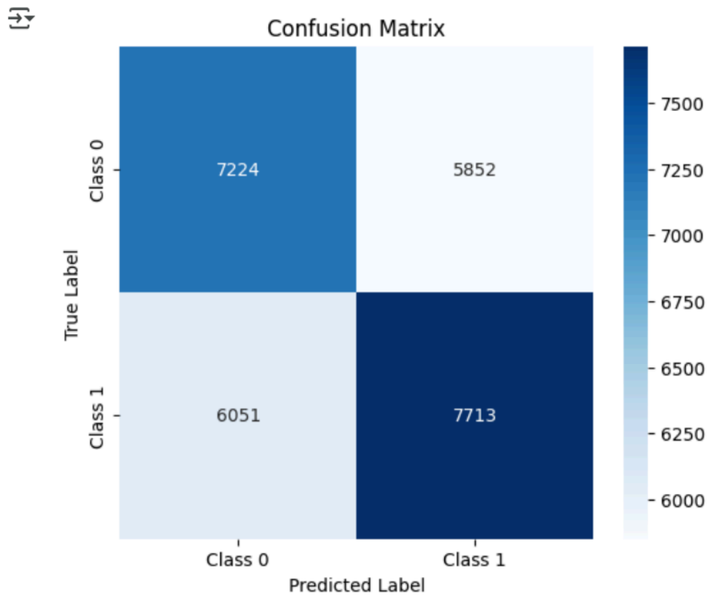
```
Accuracy: 0.5565201192250373
Confusion Matrix:
 [[7224 5852]
 [6051 7713]]
Classification Report:
               precision    recall  f1-score   support

           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))
```

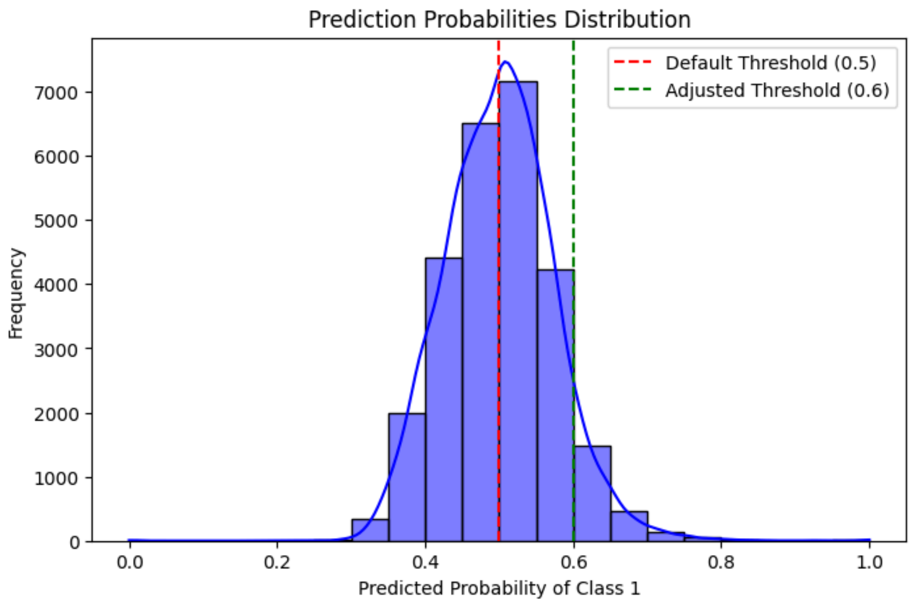
```
Accuracy: 0.5565201192250373
Confusion Matrix:
 [[7224 5852]
 [6051 7713]]
Classification Report:
              precision    recall  f1-score   support

           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")
```

Name: Mohit Kerkar **Div: D15C** **Roll No: 23**

```
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 R^2 Score Calculation
In this step, we evaluate the model's prediction performance using Mean Squared Error (MSE) and R^2 Score:

- MSE (Mean Squared Error): Measures the average squared difference between actual and predicted probabilities. A lower MSE indicates better predictions.
- R^2 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.

```python
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:

Name: Mohit Kerkar **Div: D15C** **Roll No: 23**

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

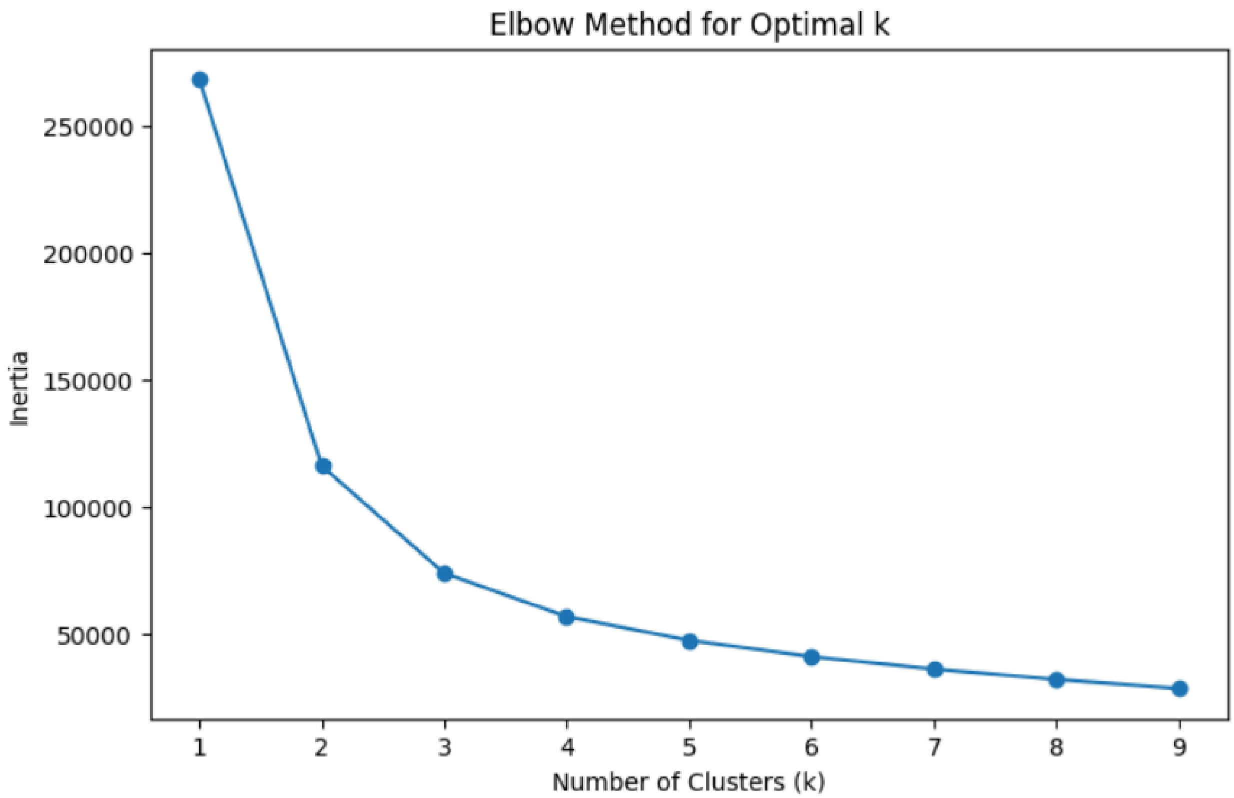
K-Means is a centroid-based clustering technique that partitions data into K clusters by minimizing intra-cluster variance. It iteratively assigns points to the nearest cluster center and updates the centroids.

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.

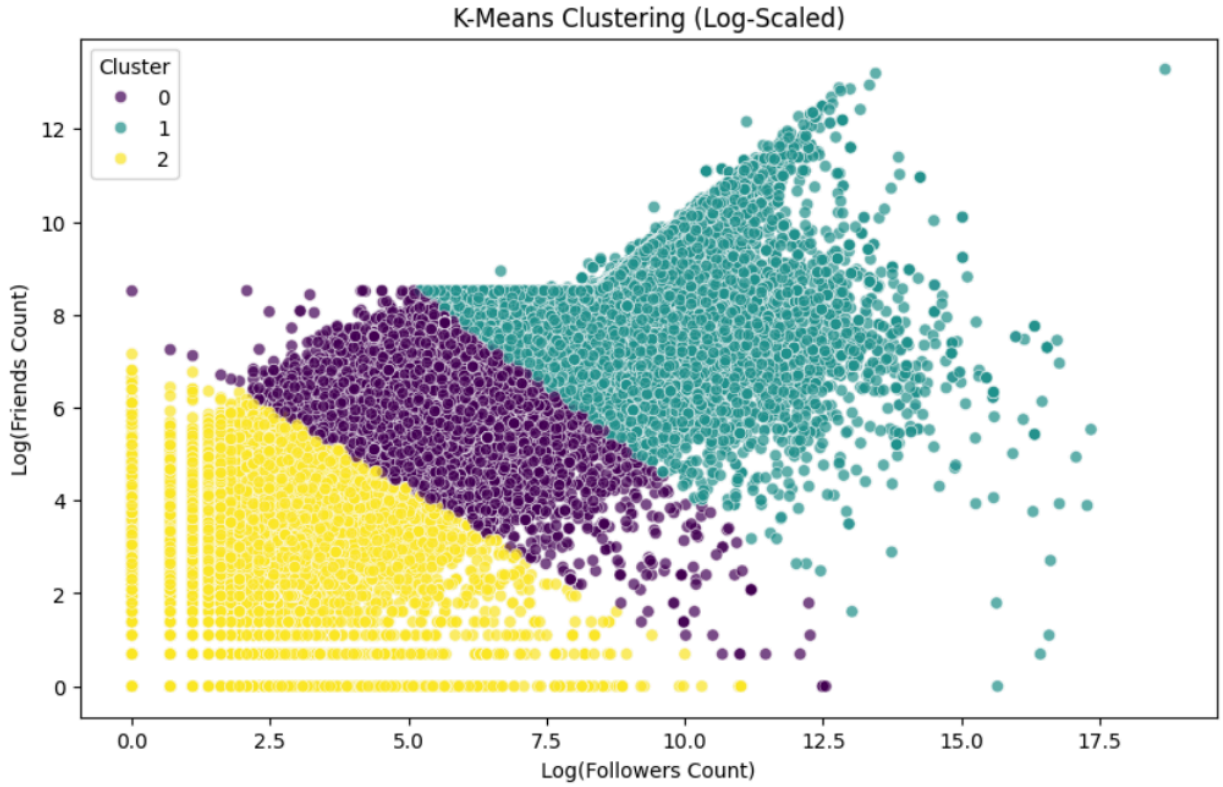
2.3 Visualization of K-Means Clustering

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).

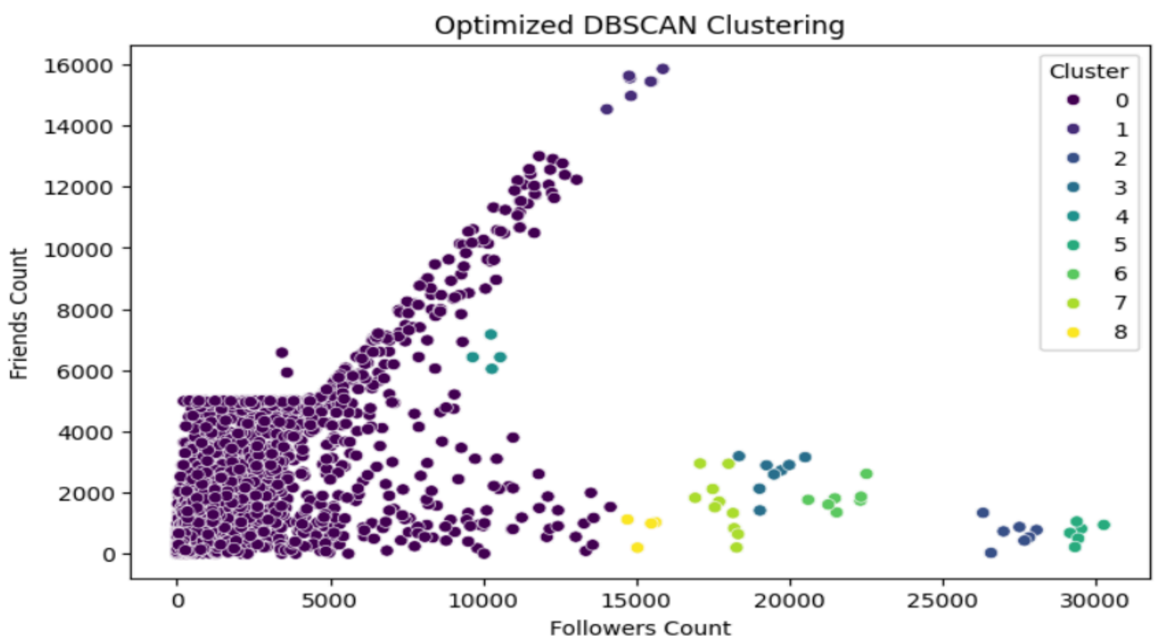
Name: Mohit Kerkar **Div: D15C** **Roll No: 23**

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
```

SILHOUTTE 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.

Name: Mohit Kerkar **Div: D15C** **Roll No: 23**

AIDS Lab
Experiment 08

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

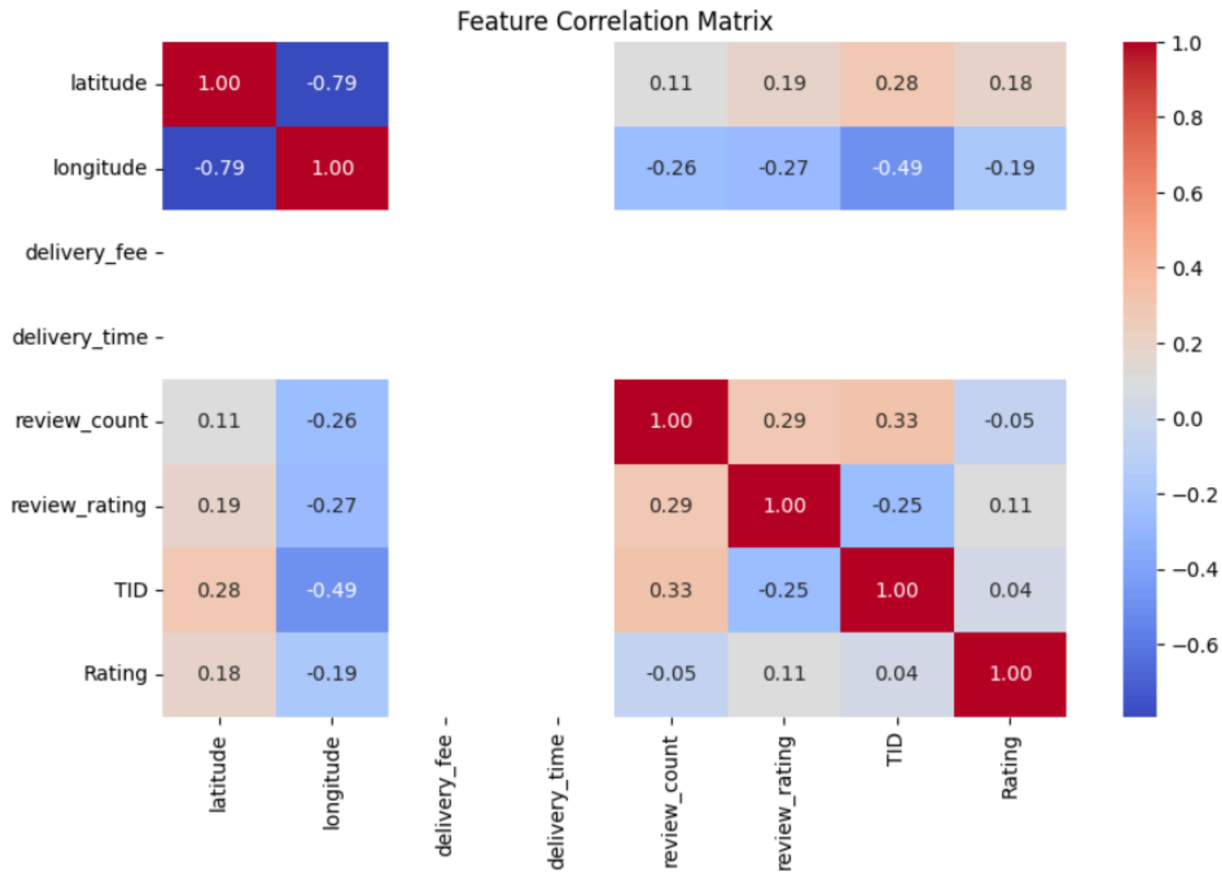
- Loads the cleaned dataset containing user reviews and ratings.

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

Name: Mohit Kerkar Div: D15C Roll No: 23

- 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.

Name: Mohit Kerkar **Div: D15C** **Roll No: 23**

Experiment 9
Aim: To perform Exploratory Data Analysis (EDA) 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. It provides faster execution than traditional Hadoop MapReduce due to its ability to perform in-memory computation, which is ideal for tasks like machine learning, data analysis, and graph processing.
Key Components of Apache Spark:
- Spark Core: The main engine for large-scale parallel data processing.
- Spark SQL: A module for structured data processing using SQL and DataFrames.
- MLlib: A scalable machine learning library.
- GraphX: Used for graph processing and computations.
- Spark Streaming: Handles real-time data processing.

How Spark Works:
- Data is processed using RDDs (Resilient Distributed Datasets) or DataFrames.
- A Driver Program creates a SparkContext, which connects to a Cluster Manager.
- Tasks are distributed to Executors across nodes for parallel execution.
- Spark supports lazy evaluation, meaning transformations are only computed when an action is triggered.

2. How Data Exploration is Done in Apache Spark?
EDA in Apache Spark is conceptually similar to that in Pandas but is designed to scale across clusters for handling massive datasets.
Steps for EDA in Apache Spark:
1. Initialization
    - Import pyspark and create a SparkSession using SparkSession.builder.
    - This session serves as the entry point to Spark functionalities.
2. Load Dataset
    - Use spark.read.csv() or spark.read.json() to load the data.
    - Use parameters like header=True and inferSchema=True for proper formatting.
3. Understand Data Schema
    - Use .printSchema() to display column data types.
    - Use .show() for a data preview.
    - Use .describe() to generate summary statistics (mean, min, max, etc.).
4. Handle Missing Values
    - Use df.na.drop() to remove null rows.
    - Use df.na.fill("value") to fill missing values appropriately.
5. Data Transformation
    - Use functions like .withColumn(), .filter(), .groupBy() for shaping and summarizing data.
6. Data Visualization

  - Convert Spark DataFrame to Pandas using .toPandas().
  - Use matplotlib or seaborn for plotting graphs and visualizations.
7. Correlation and Insights
  - Use .corr() in Pandas or Correlation.corr() from MLlib to find relationships between features.
  - Perform grouping, pivoting, and aggregations to extract 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 and explore large datasets efficiently using Spark functions like .show(), .printSchema(), and .describe().
- Handle missing data and apply transformations for better data quality.
- Convert Spark DataFrames to Pandas for visualization using matplotlib or seaborn.
- Compute correlations and derive insights through grouping and aggregation.

This experiment helped me appreciate the scalability and efficiency of Spark in handling big data and how it complements Pandas for advanced data exploration and visualization.

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.