EXPERIMENT NO. 1

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

Theory:

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

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

The following key steps were carried out in this process:

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

1. Loading data into pandas:

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

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

2. Description of the dataset.

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

3. Drop columns that are not useful:

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

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

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

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

4. Dropping rows with missing values:

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

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

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

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

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

5. Taking care of missing values:

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

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

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

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

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

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

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

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

For, numeric columns like ZIP CODE, LATITUDE & LONGITUDE we do the following
df["ZIP CODE"] = df["ZIP CODE"].fillna(df["ZIP CODE"].mode()[0]) df["LATITUDE"] =
df["LATITUDE"].fillna(df["LATITUDE"].median()) df["LONGITUDE"] =
df["LONGITUDE"].fillna(df["LONGITUDE"].median())

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

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

6. Creating dummy variables:

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

 'CONTRIBUTING FACTOR VEHICLE 2'
]

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

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

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

[5 rows x 389 columns]
```

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

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

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

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

Mentioned below is the code snippet
# Continuous columns to be standardized or normalized
continuous_columns = [

'LATITUDE', 'LONGITUDE',
'NUMBER OF PERSONS INJURED', 'NUMBER OF PERSONS KILLED', 'NUMBER OF PEDESTRIANS INJURED', 'NUMBER OF PEDESTRIANS KILLED', 'NUMBER OF CYCLIST INJURED', 'NUMBER OF CYCLIST KILLED', 'NUMBER OF MOTORIST INJURED', 'NUMBER OF MOTORIST KILLED']
# 1. Standardization (Z-score normalization)
scaler = StandardScaler()
df[continuous_columns] = scaler.fit_transform(df[continuous_columns])

Applying Normalization:
Normalization scales the data between **0 and 1** by using the minimum and maximum values of each feature.
We applied MinMaxScaler() from sklearn.preprocessing:

Its effect on our dataset:

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

Dataset before cleaning and processing:

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