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

AIDS Exp-01

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")

```
======================================================== RESTART: C:
>>> 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:

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

```
# 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:

	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 A	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 AVENU	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:

	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	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 AVENU	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 AVEN	WALLABOUT STREET		0	0	0	0	0	0	
10	2020-08-29	10:30:00	3	11385	0.9950948459	0.005221296338	POINT (-73.8620	METROPOLITAN	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 TREMONT	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 STF	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 AVENU	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.