Name: Shivam.R.Prajapati **Div: D15C** **Roll No: 41**

Experiment No: 1

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

THEORY:

Data Science:

Data Science is an interdisciplinary field that **uses statistics, machine learning, and programming to extract insights from data**. It involves collecting, cleaning, analyzing, and visualizing data to make informed decisions. Data scientists use tools like Python, SQL, and AI models to solve real-world problems in industries such as healthcare, finance, and marketing. With the increasing availability of big data, Data Science plays a crucial role in driving business strategies, automation, and innovation.

Pandas:

Pandas is a powerful Python library used for data manipulation and analysis. It provides data structures like **Series** (1D) and **DataFrame** (2D) that make handling structured data easy. **With Pandas, you can load datasets, clean missing values, filter data, perform aggregations, and visualize trends efficiently**. It is widely used in data science, machine learning, and finance for working with large datasets

STEPS:

Step 1: Load data in Pandas.

Code:

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

```
[ ] df = pd.read_csv('/content/Traffic_Collision_Data_from_2010_to_Present (2).csv')
```

The code imports essential libraries like **Pandas for data manipulation, NumPy for numerical operations, Matplotlib and Seaborn for visualization**. It then loads the **"Traffic_Collision_Data_from_2010_to_Present (2).csv"** file into a **Pandas DataFrame (df) using pd.read_csv()**, allowing further analysis and processing

Step 2: Description of the dataset.

Code 1: print(df.head())

Output:

```
   DR Number Date Reported Date Occurred  Time Occurred  Area ID   Area Name  \
0  190319651    08/24/2019    08/24/2019            450        3   Southwest
1  190319680    08/30/2019    08/30/2019           2320        3   Southwest
2  190413769    08/25/2019    08/25/2019            545        4  Hollenbeck
3  190127578    11/20/2019    11/20/2019            350        1     Central
4  190319695    08/30/2019    08/30/2019           2100        3   Southwest

   Reporting District  Crime Code Crime Code Description  \
0                 356         997      TRAFFIC COLLISION
1                 355         997      TRAFFIC COLLISION
2                 422         997      TRAFFIC COLLISION
3                 128         997      TRAFFIC COLLISION
4                 374         997      TRAFFIC COLLISION

                          MO Codes  Victim Age Victim Sex Victim Descent  \
0           3036 3004 3026 3101 4003        22.0          M              H
1  3037 3006 3028 3030 3039 3101 4003        30.0          F              H
2           3101 3401 3701 3006 3030         NaN          M              X
3      0605 3101 3401 3701 3011 3034        21.0          M              H
4      0605 4025 3037 3004 3025 3101        49.0          M              B

   Premise Code Premise Description                           Address  \
0         101.0              STREET  JEFFERSON                      BL
1         101.0              STREET  JEFFERSON                      BL
2         101.0              STREET                        N  BROADWAY
3         101.0              STREET                                1ST
4         101.0              STREET            MARTIN LUTHER KING JR
```

The output of **df.head() displays the first five rows** of the dataset, showing various columns such as **DR Number, Date Reported, Date Occurred, Time Occurred, Area ID, Area Name, Reporting District, Crime Code, Crime Code Description, MO Codes, Victim Age, Victim Sex, Victim Descent, Premise Code, Premise Description,** and **Address.** This provides an overview of traffic collision incidents, including location details, victim demographics, and crime codes.

Code 2: print(df.info())

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 619595 entries, 0 to 619594
Data columns (total 18 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   DR Number               619595 non-null  int64
 1   Date Reported           619595 non-null  object
 2   Date Occurred           619595 non-null  object
 3   Time Occurred           619595 non-null  int64
 4   Area ID                 619595 non-null  int64
 5   Area Name               619595 non-null  object
 6   Reporting District      619595 non-null  int64
 7   Crime Code              619595 non-null  int64
 8   Crime Code Description  619595 non-null  object
 9   MO Codes                532293 non-null  object
 10  Victim Age              531691 non-null  float64
 11  Victim Sex              608958 non-null  object
 12  Victim Descent          608007 non-null  object
 13  Premise Code            618636 non-null  float64
 14  Premise Description     618635 non-null  object
 15  Address                 619595 non-null  object
 16  Cross Street            590242 non-null  object
 17  Location                619595 non-null  object
dtypes: float64(2), int64(5), object(11)
memory usage: 85.1+ MB
None
```

The output of **df.info() provides a summary of the dataset**, showing that it has **619,595 rows** and **18 columns.** It **displays the column names, the number of non-null values in each column**, and their **data types** (**int64, float64, and object**). Some columns, like **MO Codes, Victim Age, Victim Sex, and Cross Street,** contain missing values. The memory usage of the dataset is **85.1+ MB**, indicating a relatively large dataset.

Code 3: print(df.describe())

Output:

```
            DR Number  Time Occurred        Area ID  Reporting District  \
count  6.195950e+05  619595.000000  619595.000000       619595.000000
mean   1.611640e+08    1352.441509      11.074290         1153.331095
std    3.724420e+07     605.329745       5.883848          589.513393
min    1.001000e+08       1.000000       1.000000          100.000000
25%    1.309219e+08     930.000000       6.000000          666.000000
50%    1.612121e+08    1430.000000      11.000000         1162.000000
75%    1.906157e+08    1824.000000      16.000000         1653.000000
max    2.521041e+08    2359.000000      21.000000         2199.000000

       Crime Code     Victim Age   Premise Code
count    619595.0  531691.000000  618636.000000
mean        997.0      41.386678     102.431370
std           0.0      16.718899      23.535171
min         997.0      10.000000     101.000000
25%         997.0      28.000000     101.000000
50%         997.0      38.000000     101.000000
75%         997.0      51.000000     101.000000
max         997.0      99.000000     970.000000
```

The dataset mainly records **traffic collisions (Crime Code 997)**, with **victim ages ranging from 10 to 99 (avg: 41.38)**. **Incidents occur mostly in the afternoon (avg. Time: 1352)**. **Area IDs range from 1 to 21**, and **Premise Codes mostly around 101**.

Code 4: print(df.isnull().sum())

Output:

```
DR Number                    0
Date Reported                0
Date Occurred                0
Time Occurred                0
Area ID                      0
Area Name                    0
Reporting District           0
Crime Code                   0
Crime Code Description       0
MO Codes                 87302
Victim Age               87904
Victim Sex               10637
Victim Descent           11588
Premise Code               959
Premise Description        960
Address                      0
Cross Street             29353
Location                     0
dtype: int64
```

The output shows missing values in various columns of the DataFrame. Columns like **MO Codes (87,302), Victim Age (87,904), Victim Sex (10,637), Victim Descent (11,588), Premise Code (959), Premise Description (960), and Cross Street (29,353)** have null values, while others like **DR Number, Date Reported, Crime Code, and Location** have none. This indicates the need for data cleaning and imputation before further analysis.

Step 3: Drop columns that aren't useful.

Code:

```
columns_to_drop = ['Crime Code Description', 'MO Codes', 'Address', 'Cross Street']
df.drop(columns=columns_to_drop, inplace=True)
print(df.head())
```

Output:

```
   DR Number Date Reported Date Occurred  Time Occurred  Area ID   Area Name  \
0  190319651    08/24/2019    08/24/2019            450        3   Southwest
1  190319680    08/30/2019    08/30/2019           2320        3   Southwest
2  190413769    08/25/2019    08/25/2019            545        4  Hollenbeck
3  190127578    11/20/2019    11/20/2019            350        1     Central
4  190319695    08/30/2019    08/30/2019           2100        3   Southwest

   Reporting District  Crime Code  Victim Age Victim Sex Victim Descent  \
0                 356         997        22.0          M              H
1                 355         997        30.0          F              H
2                 422         997         NaN          M              X
3                 128         997        21.0          M              H
4                 374         997        49.0          M              B

   Premise Code Premise Description              Location
0         101.0              STREET  (34.0255, -118.3002)
1         101.0              STREET  (34.0256, -118.3089)
2         101.0              STREET  (34.0738, -118.2078)
3         101.0              STREET  (34.0492, -118.2391)
4         101.0              STREET  (34.0108, -118.3182)
```

The **dataset has been cleaned by dropping Crime Code Description, MO Codes, Address, and Cross Street columns.** The updated **head() output confirms the presence of essential fields** like DR Number, Date Reported, Date Occurred, Time, Area, Reporting District, Crime Code, Victim Details, and Location

Step 4: Drop rows with maximum missing values.

Code:

```
threshold = df.shape[1] * 0.7
df.dropna(thresh=threshold, inplace=True)
```

This command removes rows with **more than 30% missing values** based on the total number of columns. Now, the dataset contains only rows where at least **70% of the columns have valid (non-null) data.**

Step 5: Take care of missing data.

Code:

```
df.fillna({'Victim Age': df['Victim Age'].median()}, inplace=True)
categorical_columns = ['Victim Sex', 'Victim Descent', 'Premise Description', 'Premise Code']
for col in categorical_columns:
   df[col].fillna(df[col].mode()[0], inplace=True)
print(df.isnull().sum())
```

Output:

```
  df[col].fillna(df[col].mode()[0], inplace=True)
DR Number              0
Date Reported          0
Date Occurred          0
Time Occurred          0
Area ID                0
Area Name              0
Reporting District     0
Crime Code             0
Victim Age             0
Victim Sex             0
Victim Descent         0
Premise Code           0
Premise Description    0
Location               0
dtype: int64
```

The **provided code fills missing values in the DataFrame. The Victim Age column is filled with its median value, while categorical columns (Victim Sex,**

Victim Descent, Premise Description, and Premise Code) are filled with their mode (most frequent value). Finally, **print(df.isnull().sum())** confirms that all missing values have been handled.

Step 6: Create dummy variables.

Code:

```
df = pd.get_dummies(df, columns=['Area Name', 'Victim Sex', 'Premise Description'],
drop_first=True)
```

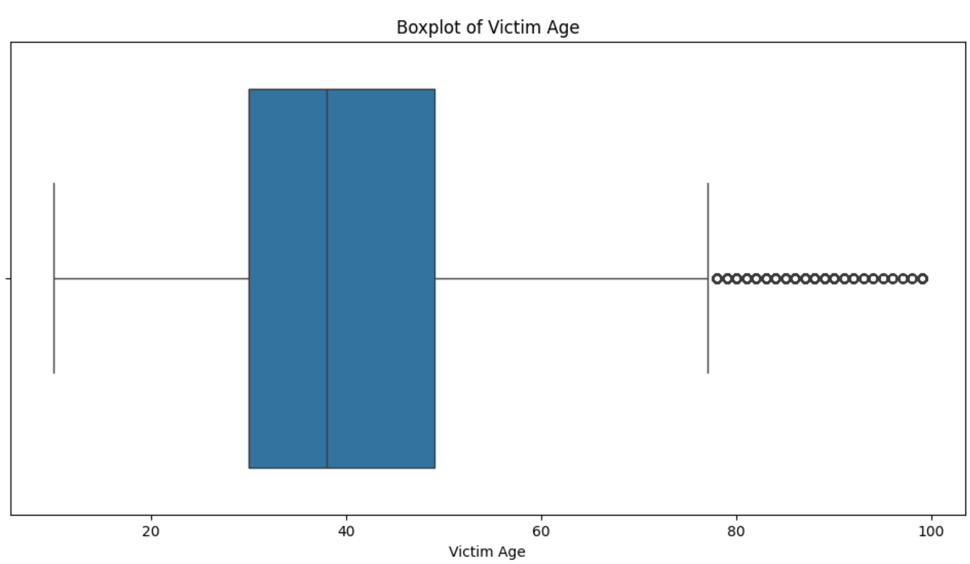
This **code applies one-hot encoding to the categorical columns** 'Area Name', 'Victim Sex', and 'Premise Description' using **pd.get_dummies()**, converting them into numerical format for machine learning models. The drop_first=True argument removes the first category from each column to avoid a dummy variable trap, reducing multicollinearity.

Step 7: Find out outliers (manually)

Code 1:

```
plt.figure(figsize=(12,6))
sns.boxplot(x=df['Victim Age'])
plt.title('Boxplot of Victim Age')
plt.show()
```

Output:



This boxplot visualizes the distribution of 'Victim Age', highlighting the median, interquartile range (IQR), and outliers. The **box represents the middle 50% of the data (Q1 to Q3)**, while the **whiskers extend to the non-outlier minimum and maximum values.** The **circles beyond the whiskers indicate outliers**, which are significantly high ages in this case.

Code 2:

```
Q1 = df['Victim Age'].quantile(0.25)
Q3 = df['Victim Age'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
outliers = df[(df['Victim Age'] < lower_bound) | (df['Victim Age'] > upper_bound)]
print("Number of Outliers in Victim Age:", len(outliers))
```

Output:

```
Number of Outliers in Victim Age: 16813
```

There are **16,813 outliers** in the 'Victim Age' column based on the IQR method, indicating a significant number of extreme values.

Code 3:

```
Q1 = df['Victim Age'].quantile(0.25)
Q3 = df['Victim Age'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
outliers = df[(df['Victim Age'] < lower_bound) | (df['Victim Age'] > upper_bound)]
print(outliers[['Victim Age']])
```

Output:

```
        Victim Age
100           84.0
101           99.0
141           99.0
146           88.0
152           90.0
...            ...
619509        99.0
619543        78.0
619566        83.0
619573        99.0
619591        99.0

[16813 rows x 1 columns]
```

Step 8: Standardization

Code:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df['Victim Age Standardized'] = scaler.fit_transform(df[['Victim Age']])
print(df[['Victim Age', 'Victim Age Standardized']].head())
```

Output:

```
   Victim Age  Victim Age Standardized
0        22.0                -1.217182
1        30.0                -0.702145
2        38.0                -0.187107
3        21.0                -1.281562
4        49.0                 0.521069
```

The **output shows the 'Victim Age' column standardized using StandardScaler**, where **values are transformed into z-scores** (mean = 0, standard deviation = 1). Negative values indicate ages below the mean, while positive values indicate ages above it.

Step 9: Normalization

Code:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df['Victim Age Normalized'] = scaler.fit_transform(df[['Victim Age']])
print(df[['Victim Age', 'Victim Age Normalized']].head())
```

Output:

	Victim Age	Victim Age Normalized
0	22.0	0.134831
1	30.0	0.224719
2	38.0	0.314607
3	21.0	0.123596
4	49.0	0.438202

The **output shows the 'Victim Age' column normalized using MinMaxScaler,** which **scales values between 0 and 1.** This ensures that the smallest age is mapped to 0 and the largest to 1, with all other values proportionally adjusted in between.

<u>CONCLUSION</u>:

We analyzed the 'Victim Age' data to understand its distribution and identify unusual values (outliers). First, we used a **boxplot** to visualize the spread of ages and detect outliers. Then, we calculated the **Interquartile Range (IQR)** to find and count the extreme values. Next, we **standardized** the ages using StandardScaler, which transforms the data to have a mean of 0 and a standard deviation of 1. Finally, we applied **MinMaxScaler to normalize the data**, scaling values between 0 and 1. These steps help in preparing the data for further analysis by making it more consistent and removing biases caused by extreme values.