Data Manipulation with pandas

Data manipulation refers to cleaning, transforming, and organizing data to make it suitable for analysis. In data science, raw data is often messy (e.g., missing values, incorrect formats, duplicates), and Pandas is the most popular Python library to handle these issues efficiently.

In Python, a data object refers to any entity that holds data and has associated methods and attributes. list, tuple, dictionary.... In Pandas, data objects primarily refer to Series and DataFrames, which store structured data for analysis. Series and dataframe

Installation of Pandas in Python

- Install Pandas Using pip

Working with series

creating series

```
# Pandas library ko import kar rahe hain jo data handling ke liye use hoti hai
import pandas as pd

# Ek list create kar rahe hain jisme kuch numerical values hain
data = [1, 2, 3, 4]
# List ko Pandas Series me convert kar rahe hain jo ek 1-D array jaisa hota hai
s1 = pd.Series(data)
s1
```

```
0    1
1    2
2    3
3    4
dtype: int64
```

creating series using dictionary

```
data = {'a': 100, 'b': 240, 'c':280}
# List ko Pandas Series me convert kar rahe hain jo ek 1-D array jaisa hota hai
s2 = pd.Series(data)
s2
```

```
a    100
b    240
c    280
dtype: int64
```

custom indexing

```
# Ek list create kar rahe hain jisme kuch numerical values hain
data = [1,2,3,4]
# List ko Pandas Series me convert kar rahe hain jo ek 1-D array jaisa
hota hai
s3 = pd.Series(data, index=['aa', 'bb', 'cc', 'dd'])
s3
```

```
aa    1
bb    2
cc    3
dd    4
dtype: int64
```

Accessing Data in a Series (by index, position, slicing)

```
# Series ya uske specific elements ko print kar rahe hain
print(s3['aa':'dd'])
```

```
aa    1
bb    2
cc    3
dd    4
dtype: int64
```

Modifying Data in a Series

```
# Series ke kisi specific index par value ko update kar rahe hain
s3['bb'] = 5
# Series ya uske specific elements ko print kar rahe hain
print(s3)
```

```
aa    1
bb    5
cc    3
dd    4
dtype: int64
```

Operations like Addition, Subtraction, Multiplication

```
# Series ya uske specific elements ko print kar rahe hain
print(s3 + 5)
```

```
aa     6
bb    10
cc     8
dd     9
dtype: int64
```

Using Mathematical Functions

```
# Series ya uske specific elements ko print kar rahe hain
print(s3.apply(lambda x: x ** 2))
```

```
aa     1
bb    25
cc     9
dd    16
dtype: int64
```

Handling Missing Data in series

```
# Pandas library ko import kar rahe hain jo data handling ke liye use hoti hai
import pandas as pd

# Example Pandas Series
# Ek list create kar rahe hain jisme kuch numerical values hain
data = [10, 20, 30, 40]
# List ko Pandas Series me convert kar rahe hain jo ek 1-D array jaisa hota hai
s4 = pd.Series(data)
# Series ya uske specific elements ko print kar rahe hain
print(s4)
```

```
0    10
1    20
2    30
3    40
dtype: int64
```

DataFrames

It is a two-dimensional data structure where data is organized in a tabular format with rows and columns. Widely used for data manipulation and analysis in data science. The data is the core content, which can include various types like integers, floats, strings, or more complex data. This data is stored in a structured format, much like the cells in a spreadsheet. DataFrame is similar to an advanced Excel sheet or SQL table but provides far more flexibility and powerful functionality for handling, analyzing, and manipulating structured data programmatically.

Usages

- Handle structured data (rows and columns) effectively.
- Allow easy data manipulation (sorting, filtering, aggregation).
- It integrates well with libraries for data science in advanced analytics and machine learning.

Creating dataframes

```
# Ek list create kar rahe hain jisme kuch numerical values hain
data = [[1,'Amin', 24],
```

```
[2,'bima', 30], [3,'Sakshi', 27]]
# Series ke kisi specific index par value ko update kar rahe hain
df = pd.DataFrame(data, columns=['ID', 'Name', 'Age'])
df
```

```
   ID    Name  Age
0   1    Amin   24
1   2    bima   30
2   3  Sakshi   27
```

loading dataset

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

Viewing Data

```
df.head(10)
```

```
              area_type   availability                  location
size  \
0  Super built-up  Area         19-Dec  Electronic City Phase II
2 BHK
1            Plot  Area  Ready To Move          Chikka Tirupathi  4
Bedroom
2        Built-up  Area  Ready To Move               Uttarahalli
3 BHK
3  Super built-up  Area  Ready To Move        Lingadheeranahalli
3 BHK
4  Super built-up  Area  Ready To Move                  Kothanur
2 BHK
5  Super built-up  Area  Ready To Move                Whitefield
2 BHK
6  Super built-up  Area         18-May          Old Airport Road
4 BHK
7  Super built-up  Area  Ready To Move              Rajaji Nagar
4 BHK
8  Super built-up  Area  Ready To Move              Marathahalli
3 BHK
9            Plot  Area  Ready To Move              Gandhi Bazar  6
Bedroom

   society total_sqft  bath  balcony   price
0  Coomee        1056   2.0      1.0   39.07
1  Theanmp       2600   5.0      3.0  120.00
2      NaN       1440   2.0      3.0   62.00
3  Soiewre       1521   3.0      1.0   95.00
4      NaN       1200   2.0      1.0   51.00
5  DuenaTa       1170   2.0      1.0   38.00
6  Jaades        2732   4.0      NaN  204.00
```

```
7  Brway G       3300   4.0      NaN  600.00
8      NaN       1310   3.0      1.0   63.25
9      NaN       1020   6.0      NaN  370.00

df.tail()

                  area_type   availability                  location
size  \
13315       Built-up  Area  Ready To Move                Whitefield  5
Bedroom
13316  Super built-up  Area  Ready To Move             Richards Town
4 BHK
13317       Built-up  Area  Ready To Move  Raja Rajeshwari Nagar
2 BHK
13318  Super built-up  Area         18-Jun        Padmanabhanagar
4 BHK
13319  Super built-up  Area  Ready To Move           Doddathoguru
1 BHK

       society total_sqft  bath  balcony  price
13315  ArsiaEx       3453   4.0      0.0  231.0
13316      NaN       3600   5.0      NaN  400.0
13317  Mahla T       1141   2.0      1.0   60.0
13318  SollyCl       4689   4.0      1.0  488.0
13319      NaN        550   1.0      1.0   17.0
```

View a series

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13320 entries, 0 to 13319
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   area_type     13320 non-null  object
 1   availability  13320 non-null  object
 2   location      13319 non-null  object
 3   size          13304 non-null  object
 4   society       7818 non-null   object
 5   total_sqft    13320 non-null  object
 6   bath          13247 non-null  float64
 7   balcony       12711 non-null  float64
 8   price         13320 non-null  float64
dtypes: float64(3), object(6)
memory usage: 936.7+ KB

df.describe()
```

```
                bath       balcony         price
count  13247.000000  12711.000000  13320.000000
mean       2.692610      1.584376    112.565627
std        1.341458      0.817263    148.971674
min        1.000000      0.000000      8.000000
25%        2.000000      1.000000     50.000000
50%        2.000000      2.000000     72.000000
75%        3.000000      2.000000    120.000000
max       40.000000      3.000000   3600.000000
```

```
df.shape
```

```
(13320, 9)
```

```
df[['size','balcony']]
```

```
             size  balcony
0          2 BHK      1.0
1      4 Bedroom      3.0
2          3 BHK      3.0
3          3 BHK      1.0
4          2 BHK      1.0
...          ...      ...
13315  5 Bedroom      0.0
13316      4 BHK      NaN
13317      2 BHK      1.0
13318      4 BHK      1.0
13319      1 BHK      1.0

[13320 rows x 2 columns]
```

Selecting Data (select column..)

```
# Ek list create kar rahe hain jisme kuch numerical values hain
data = [1,'Amin', 24], [2,'bima', 30], [3,'Sakshi', 27]
# Series ke kisi specific index par value ko update kar rahe hain
df = pd.DataFrame(data, columns=['ID', 'Name', 'Age'])
df
```

```
   ID    Name  Age
0   1    Amin   24
1   2    bima   30
2   3  Sakshi   27
```

```
df.iloc[0:2]
```

```
   ID  Name  Age
0   1  Amin   24
1   2  bima   30
```

Select row

```
df
```

```
   ID    Name  Age
0   1    Amin   24
1   2    bima   30
2   3  Sakshi   27
```

Selecting a row and colum

```
df.iloc[0:2, 1:3]
```

```
   Name  Age
0  Amin   24
1  bima   30
```

Adding New Columns

```
# Series ke kisi specific index par value ko update kar rahe hain
df['Mark'] = [27, 42, 35]
df
```

```
   ID    Name  Age  Mark
0   1    Amin   24    27
1   2    bima   30    42
2   3  Sakshi   27    35
```

Deleting a Column

```
df.drop("Mark", axis=1, inplace=True)
#df.drop(2, axis=0)
df
```

```
   ID    Name  Age
0   1    Amin   24
1   2    bima   30
2   3  Sakshi   27
```

Filter Pandas DataFrame

Filtering a Pandas DataFrame by column values is one of the most important tasks in data analysis Data filtering is possible on Pandas based on several conditions. Data can be filtered in Pandas in two primary ways:

- Column name (Labels)
- Actual data inside, by values.

Basic filtering (boolean indexing)

```
# Series ke kisi specific index par value ko update kar rahe hain
df1 = df[df['Age']>25]
df1
```

```
   ID    Name  Age
1   2    bima   30
2   3  Sakshi   27
```

Filtering by Specific Column Values

```
# Series ke kisi specific index par value ko update kar rahe hain
df2 = df[df['Name'] =='Sakshi']
df2
```

```
   ID    Name  Age
2   3  Sakshi   27
```

loc[]: Filtering Rows and Selecting Columns

```
# Series ke kisi specific index par value ko update kar rahe hain
df3 = df.loc[df['Age']>25, ['ID','Name']]
df3
```

```
   ID    Name
1   2    bima
2   3  Sakshi
```

Filtering by Membership (Filter by values)

```
# Series ke kisi specific index par value ko update kar rahe hain
df4 = df[df['Name'].isin(['Sakshi', 'bima'])]
df4
```

```
   ID    Name  Age
1   2    bima   30
2   3  Sakshi   27
```

Using .query():Filter by values

```
df5 = df.query('Age>25 and ID >2')
df5
```

```
   ID    Name  Age
2   3  Sakshi   27
```

Combining Conditions with Logical Operators

```
# Pandas library ko import kar rahe hain jo data analysis ke liye
kaafi powerful hai
```

```
import pandas as pd

# Example Pandas Series
# Ek simple numerical values ki list create kar rahe hain
data = [10, 20, 30, 40]

# List ko Pandas Series me convert kar rahe hain jo ek 1-D array jaisa
hota hai
s4 = pd.Series(data)

# Series ka output print kar rahe hain
print(s4)

# Logical operations perform kar rahe hain (example: values > 20)
print("\nValues greater than 20:\n", s4[s4 > 20])
```

```
0    10
1    20
2    30
3    40
dtype: int64

Values greater than 20:
 2    30
3    40
dtype: int64
```

Sorting Rows by Index

```
# Example DataFrame create kar rahe hain
data = {'ID': [103, 101, 102], 'Name': ['Amit', 'Rohit', 'Sakshi'],
'Age': [25, 30, 27]}

# DataFrame ko Pandas me convert kar rahe hain
df = pd.DataFrame(data)

# Default DataFrame print kar rahe hain
print("\nOriginal DataFrame:\n", df)

# Index ke basis par rows ko sort kar rahe hain
df_sorted = df.sort_index()

print("\nSorted by Index:\n", df_sorted)
```

```
Original DataFrame:
     ID    Name  Age
0  103    Amit   25
1  101   Rohit   30
2  102  Sakshi   27
```

```
Sorted by Index:
     ID    Name  Age
0  103    Amit   25
1  101   Rohit   30
2  102  Sakshi   27
```

Sorting Columns by Index

```
# Columns ko alphabetically sort karne ke liye
df_sorted_col = df.sort_index(axis=1)

print("\nSorted by Column Index:\n", df_sorted_col)
```

```
Sorted by Column Index:
    Age   ID    Name
0   25  103    Amit
1   30  101   Rohit
2   27  102  Sakshi
```

Sorting Rows by Values

```
# Age column ke basis par sorting kar rahe hain
df_sorted_values = df.sort_values(by="Age")

print("\nSorted by Age:\n", df_sorted_values)
```

```
Sorted by Age:
     ID    Name  Age
0  103    Amit   25
2  102  Sakshi   27
1  101   Rohit   30
```

Sorting in Descending Order

```
# Age ke basis par descending order me sort kar rahe hain
df_sorted_desc = df.sort_values(by="Age", ascending=False)

print("\nSorted by Age (Descending Order):\n", df_sorted_desc)
```

```
Sorted by Age (Descending Order):
     ID    Name  Age
1  101   Rohit   30
2  102  Sakshi   27
0  103    Amit   25
```

Sorting with NaN Values

```python
# DataFrame me missing values (NaN) add kar rahe hain
data = {'ID': [103, 101, 102], 'Name': ['Amit', None, 'Sakshi'],
'Age': [25, None, 27]}
df_nan = pd.DataFrame(data)

print("\nDataFrame with NaN Values:\n", df_nan)

# Age ke basis par sorting kar rahe hain (NaN values ko last me rakhne
ke liye `na_position='last'`)
df_nan_sorted = df_nan.sort_values(by="Age", na_position='last')

print("\nSorted with NaN Values:\n", df_nan_sorted)
```

```
DataFrame with NaN Values:
     ID    Name   Age
0  103    Amit  25.0
1  101    None   NaN
2  102  Sakshi  27.0

Sorted with NaN Values:
     ID    Name   Age
0  103    Amit  25.0
2  102  Sakshi  27.0
1  101    None   NaN
```

Grouping in Pandas

It is a process of splitting data into groups based on certain criteria and performing operations on each group separately. The function used for this is groupby()

It is also used with sum(), mean(), count(), or agg() for summarizing the data.

Grouping is useful when analyzing datasets that contain categorical variables and numerical values.

This method is widely used in data analysis and reporting to identify patterns and insights within different categories.

DataFrame.groupby(by, axis=0, level=None, as_index=True, sort=True, group_keys=True, observed=False, dropna=True)

- by Column(s) to group by (string or list)
- axis Axis to group by (default 0, meaning rows)
- level If using a MultiIndex, groups by the specified level
- as_index If True, sets the group column as index (default: True)
- sort If True, sorts the group keys (default: True)
- group_keys True (default) → Includes group labels in output
- observed False (default) → Used for categorical data; True removes unused categories
- dropna True (default) → Ignores NaN values in group keys.

Grouping by column

```
# Ek example DataFrame create kar rahe hain
data = {'Department': ['HR', 'IT', 'HR', 'IT', 'Finance'],
        'Employee': ['Amit', 'Sakshi', 'Ravi', 'Neha', 'Rohit'],
        'Salary': [50000, 60000, 55000, 75000, 72000]}

df_group = pd.DataFrame(data)

# Department wise grouping kar rahe hain aur uska mean salary nikal
rahe hain
grouped_df = df_group.groupby('Department')['Salary'].mean()

print("\nGrouped by Department with Mean Salary:\n", grouped_df)
```

```
Grouped by Department with Mean Salary:
 Department
Finance    72000.0
HR         52500.0
IT         67500.0
Name: Salary, dtype: float64
```

Grouping by Multiple Columns

```
# Department aur Employee ke basis par grouping kar rahe hain
grouped_multiple = df_group.groupby(['Department', 'Employee']).sum()

print("\nGrouped by Department & Employee:\n", grouped_multiple)
```

```
Grouped by Department & Employee:
                     Salary
Department Employee
Finance    Rohit      72000
HR         Amit       50000
           Ravi       55000
IT         Neha       75000
           Sakshi     60000
```

Applying Multiple Aggregations

```
# Grouping karne ke baad multiple aggregation operations apply kar
rahe hain
agg_operations = df_group.groupby('Department')['Salary'].agg(['sum',
'mean', 'count'])

print("\nMultiple Aggregation Functions:\n", agg_operations)
```

```
Multiple Aggregation Functions:
                sum     mean  count
Department
Finance       72000  72000.0      1
HR           105000  52500.0      2
IT           135000  67500.0      2
```

Transformations and Filtering

```
# Salary ko ek naye format me convert kar rahe hain (Transformation)
df_group['Salary_Lakhs'] = df_group['Salary'] / 100000

print("\nTransformed DataFrame (Salary in Lakhs):\n", df_group)

# Filter apply kar rahe hain: Sirf wo employees jinki salary 60000 se
jyada hai
filtered_df = df_group[df_group['Salary'] > 60000]

print("\nFiltered Employees with Salary > 60000:\n", filtered_df)
```

```
Transformed DataFrame (Salary in Lakhs):
   Department Employee  Salary  Salary_Lakhs
0          HR     Amit   50000          0.50
1          IT   Sakshi   60000          0.60
2          HR     Ravi   55000          0.55
3          IT     Neha   75000          0.75
4     Finance    Rohit   72000          0.72

Filtered Employees with Salary > 60000:
   Department Employee  Salary  Salary_Lakhs
3          IT     Neha   75000          0.75
4     Finance    Rohit   72000          0.72
```

Filtering Groups Based on Conditions

```
# Sirf aise departments ko filter kar rahe hain jisme total salary
70000 se jyada hai
filtered_groups = df_group.groupby('Department').filter(lambda x:
x['Salary'].sum() > 70000)

print("\nFiltered Groups where total salary > 70000:\n",
filtered_groups)
```

```
Filtered Groups where total salary > 70000:
   Department Employee  Salary  Salary_Lakhs
0          HR     Amit   50000          0.50
1          IT   Sakshi   60000          0.60
```

```
2         HR     Ravi   55000          0.55
3         IT     Neha   75000          0.75
4    Finance    Rohit   72000          0.72
```

Data Cleaning

- Data Cleaning refers to the process of discovering and correcting errors, inconsistencies, and missing values in a dataset so that it is used correctly in an analysis.
- Raw data usually contain missing values, incorrect formats, and duplicate records.
- Therefore, handling missing and inconsistent data is an important part of data preprocessing in Pandas.
- Pandas offers multiple functions to detect, remove, and replace missing or inconsistent values effortlessly.

Missing values

What is a Missing Value?

- A missing value is an absent data point for a particular variable in a dataset.
- There are different forms of missing data, such as blank cells, NULL, NaN, and special indicators like "NA" or "unknown".
- Generally, missing values are a subset of the challenges of data analysis because they may lead to biased results and computational errors.

Challenges Posed by Missing Values

- Missing values affect data analysis very significantly as shown below:
- The fewer observations, the lesser accuracy and reliability of analysis due to missing data.

Data Cleaning - Handling Missing Values

```
# Missing values ka DataFrame
data = {'ID': [1, 2, 3, 4],
        'Name': ['Amit', None, 'Sakshi', 'Ravi'],
        'Age': [25, None, 27, 30]}

df_missing = pd.DataFrame(data)

print("\nDataFrame with Missing Values:\n", df_missing)

# Missing values ko fill karna (fillna)
df_filled = df_missing.fillna({'Name': 'Unknown', 'Age':
df_missing['Age'].mean()})

print("\nDataFrame after Filling Missing Values:\n", df_filled)
```

```
DataFrame with Missing Values:
    ID    Name   Age
0   1    Amit  25.0
```

```
1   2    None   NaN
2   3  Sakshi  27.0
3   4    Ravi  30.0

DataFrame after Filling Missing Values:
    ID     Name        Age
0   1     Amit  25.000000
1   2  Unknown  27.333333
2   3   Sakshi  27.000000
3   4     Ravi  30.000000
```

Types of Missing Values

1. Missing Completely at Random (MCAR)
2. Missing at Random (MAR)
3. Missing Not at Random (MNAR)

Missing Completely at Random (MCAR) Jab kisi bhi column ya variable me missing value hone ka koi pattern nahi hota, aur ye ek random event hota hai.

Example:

Aapke paas student dataset hai jisme kuch students ke phone numbers missing hain, lekin yeh random hai (koi specific reason nahi hai). Yeh MCAR hoga kyunki kisi particular group ya category ke missing hone ka koi relation nahi hai. Solution:

Listwise Deletion: Jisme missing values wale rows ko hata diya jaye. Mean/Median/Mode Imputation: Missing values ko average ya mode se fill karna. Python Example (MCAR Handling)

```python
import pandas as pd
import numpy as np

# Example dataset with MCAR
data = {'Student': ['Amit', 'Rohit', 'Sakshi', 'Neha', 'Rahul'],
        'Phone': [9876543210, np.nan, 8765432109, np.nan, 9123456789]}
# Some random missing values

df = pd.DataFrame(data)

# Checking Missing Values
print("\nOriginal DataFrame:\n", df)

# Filling missing values with Mean
df['Phone'].fillna(df['Phone'].mean(), inplace=True)

print("\nAfter Handling MCAR (Filled Missing Values with Mean):\n",
df)
```

```
Original DataFrame:
   Student         Phone
0    Amit  9.876543e+09
1   Rohit           NaN
2  Sakshi  8.765432e+09
3    Neha           NaN
4   Rahul  9.123457e+09

After Handling MCAR (Filled Missing Values with Mean):
   Student         Phone
0    Amit  9.876543e+09
1   Rohit  9.255144e+09
2  Sakshi  8.765432e+09
3    Neha  9.255144e+09
4   Rahul  9.123457e+09
```

Missing at Random (MAR)

- Jab ek column ka missing hona kisi doosre column se related hota hai, toh usse MAR kehte hain.

Example:

- Job application dataset me females apni salary details nahi bhar rahi hain, jabki males bhar rahe hain.
Missing data ka pattern gender se related hai, isliye yeh MAR hai.
Solution:

- Regression Imputation: Missing values ko predict karne ke liye models ka use karna.
KNN Imputation: Similar records ka use karke missing values bharna.
Python Example (MAR Handling)

```
  Input In [54]
    - Jab ek column ka missing hona kisi doosre column se related hota hai, toh usse MAR kehte hain.
          ^
SyntaxError: invalid syntax
```

```python
import pandas as pd
import numpy as np

# Example dataset with MAR
data = {'Gender': ['Male', 'Female', 'Male', 'Female', 'Male'],
        'Salary': [50000, np.nan, 60000, np.nan, 70000]}  # Salary missing for Females

df = pd.DataFrame(data)

print("\nOriginal DataFrame:\n", df)
```

```python
# Filling missing values using Group Mean (Mean Salary for each Gender)
df['Salary'] = df.groupby('Gender')['Salary'].transform(lambda x: x.fillna(x.mean()))

print("\nAfter Handling MAR (Filling Missing Values using Group Mean):\n", df)
```

```
Original DataFrame:
    Gender   Salary
0    Male  50000.0
1  Female      NaN
2    Male  60000.0
3  Female      NaN
4    Male  70000.0

After Handling MAR (Filling Missing Values using Group Mean):
    Gender   Salary
0    Male  50000.0
1  Female      NaN
2    Male  60000.0
3  Female      NaN
4    Male  70000.0
```

Missing Not at Random (MNAR)
 Jab data missing hone ka pattern kisi unknown factor se influence hota hai, toh MNAR hota hai.

 Example:

Mental health survey me log depression ka answer nahi dena chahte, toh missing values intentional hain.
Customers apni income ko hide kar rahe hain kyunki wo share nahi karna chahte.
Yeh MNAR hai, kyunki missing values ka ek hidden reason hai.
 Solution:

Domain Knowledge ka use karke missing values ko samajhna aur fill karna
Multiple Imputation technique ka use karna
Dataset ka Exploratory Data Analysis (EDA) karke missing pattern ko samajhna
 Python Example (MNAR Handling)

```
  Input In [56]
    Missing Not at Random (MNAR)
            ^
SyntaxError: invalid syntax
```

```python
import pandas as pd
import numpy as np

# Example dataset with MNAR
data = {'Customer': ['Amit', 'Rohit', 'Sakshi', 'Neha', 'Rahul'],
        'Income': [50000, np.nan, 60000, np.nan, 70000]}  # Some
people are hiding income

df = pd.DataFrame(data)

print("\nOriginal DataFrame:\n", df)

# Using Median Imputation (assuming high-income people are more likely
to hide data)
df['Income'].fillna(df['Income'].median(), inplace=True)

print("\nAfter Handling MNAR (Filling Missing Values with Median):\n",
df)
```

```
Original DataFrame:
   Customer   Income
0     Amit  50000.0
1    Rohit      NaN
2   Sakshi  60000.0
3     Neha      NaN
4    Rahul  70000.0

After Handling MNAR (Filling Missing Values with Median):
   Customer   Income
0     Amit  50000.0
1    Rohit  60000.0
2   Sakshi  60000.0
3     Neha  60000.0
4    Rahul  70000.0
```

```python
#Handling Different Types of Missing Values¶
import numpy as np

# Different types ke missing values add kar rahe hain
data = {'ID': [1, 2, 3, 4],
        'Name': ['Amit', 'Unknown', np.nan, 'Ravi'],
        'Age': [25, None, np.nan, 30]}

df_nan_types = pd.DataFrame(data)

print("\nDataFrame with Different Types of Missing Values:\n",
df_nan_types)

# NaN values ko drop karna
df_cleaned = df_nan_types.dropna()
```

```
print("\nDataFrame after Dropping NaN Values:\n", df_cleaned)
```

```
DataFrame with Different Types of Missing Values:
    ID     Name   Age
0   1     Amit  25.0
1   2  Unknown   NaN
2   3      NaN   NaN
3   4     Ravi  30.0

DataFrame after Dropping NaN Values:
    ID  Name   Age
0   1  Amit  25.0
3   4  Ravi  30.0
```

Dataset (https://www.kaggle.com/datasets/amitabhajoy/bengaluru-house-price-data?resource=download)

```
```