1: Data Loading

In [9]:

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

# Create a sample dataset
data = {
    "ID": [1, 2, 2, 3, 4, 5, 6, 7, 8, 9],
    "Name": ["Amit", "Priya", "Priya", "Rahul", "Sneha", "Vikram", "Raj", "A
    "Age": [25, 30, 30, 34, -5, 40, 35, 29, 150, 28],
    "Salary (INR)": [500000, 540000, 540000, 620000, 720000, np.nan, 800000,
    "City": ["Mumbai", "Delhi", "Delhi", "Bangalore", "Chennai", "Chennai",
    "Joining Date": ["2022-01-15", "2021-08-20", "2021-08-20", "2020-06-30",
    "Department": [" HR ", "Finance", "Finance", "IT", "HR", "HR", "IT", "Fi
}

df = pd.DataFrame(data)
df
df.dtypes
df.info()
df
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   ID            10 non-null     int64  
 1   Name          10 non-null     object 
 2   Age           10 non-null     int64  
 3   Salary (INR)  9 non-null      float64
 4   City          10 non-null     object 
 5   Joining Date  10 non-null     object 
 6   Department    10 non-null     object 
dtypes: float64(1), int64(2), object(4)
memory usage: 688.0+ bytes
```

Out[9]:

	ID	Name	Age	Salary (INR)	City	Joining Date	Department
0	1	Amit	25	500000.0	Mumbai	2022-01-15	HR
1	2	Priya	30	540000.0	Delhi	2021-08-20	Finance
2	2	Priya	30	540000.0	Delhi	2021-08-20	Finance
3	3	Rahul	34	620000.0	Bangalore	2020-06-30	IT
4	4	Sneha	-5	720000.0	Chennai	2019-11-25	HR
5	5	Vikram	40	NaN	Chennai	2018-03-14	HR
6	6	Raj	35	800000.0	Kolkata	2017-09-10	IT
7	7	Ananya	29	920000.0	Pune	2016-07-04	Finance
8	8	Kiran	150	1030000.0	Hyderabad	2015-05-21	IT
9	9	Neha	28	1140000.0	Ahmedabad	2014-12-11	HR

2. Find Basic Descriptive statistics

In [5]:

```
df.describe()
```

Out[5]:

	ID	Age	Salary (INR)
count	10.000000	10.00000	9.000000e+00
mean	4.700000	39.60000	7.566667e+05
std	2.750757	40.65355	2.318405e+05
min	1.000000	-5.00000	5.000000e+05
25%	2.250000	28.25000	5.400000e+05
50%	4.500000	30.00000	7.200000e+05
75%	6.750000	34.75000	9.200000e+05
max	9.000000	150.00000	1.140000e+06

3. Check for duplicate rows and remove the duplicate values

In [8]:

```
df.duplicated().sum()
```

Out[8]:

```
1
```

In [15]:

```
# we found there is one duplicate row,we can remove it
df=df.drop_duplicates() # To remove the duplicate values
df.duplicated().sum()
cp=df.copy()
cp.head()
```

Out[15]:

	ID	Name	Age	Salary (INR)	City	Joining Date	Department
0	1	Amit	25	500000.0	Mumbai	2022-01-15	HR
1	2	Priya	30	540000.0	Delhi	2021-08-20	Finance
3	3	Rahul	34	620000.0	Bangalore	2020-06-30	IT
4	4	Sneha	-5	720000.0	Chennai	2019-11-25	HR
5	5	Vikram	40	NaN	Chennai	2018-03-14	HR

4. Check for no.of missing values and handle them

In [12]:

```
#Before removing missing value
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9 entries, 0 to 9
Data columns (total 7 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   ID             9 non-null      int64  
 1   Name           9 non-null      object 
 2   Age            9 non-null      int64  
 3   Salary (INR)   8 non-null      float64
 4   City           9 non-null      object 
 5   Joining Date   9 non-null      object 
 6   Department     9 non-null      object 
dtypes: float64(1), int64(2), object(4)
memory usage: 576.0+ bytes
```

In [14]:

```
#first check for null values in the dataframe
df.isnull().sum()#.sum()

```

Out[14]:

```
ID                0
Name              0
Age               0
Salary (INR)      1
City              0
Joining Date      0
Department        0
dtype: int64
```

There are two ways to handle missing values 1. To remove 2. To fill missing values

DataFrame.dropna(axis=0, how='any', thresh=None, subset=None, inplace=False)

df.dropna():Dropping rows with NaN values (default behavior)

df.dropna(axis=1):Dropping columns with NaN values

df.dropna(subset=col_name): Drop rows based on given column conatins Nan values

df.dropna(how=any/all):Drop if any row contain single Nan or all Nan values

df.dropna(thresh=2): drop rows with more than 2 missing values

In [16]:
```python
# Handling Missing Values
# Drop rows where ID is missing
df.dropna(subset=["ID"], inplace=True) #specific column has missing values
df
```

Out[16]:

	ID	Name	Age	Salary (INR)	City	Joining Date	Department
0	1	Amit	25	500000.0	Mumbai	2022-01-15	HR
1	2	Priya	30	540000.0	Delhi	2021-08-20	Finance
3	3	Rahul	34	620000.0	Bangalore	2020-06-30	IT
4	4	Sneha	-5	720000.0	Chennai	2019-11-25	HR
5	5	Vikram	40	NaN	Chennai	2018-03-14	HR
6	6	Raj	35	800000.0	Kolkata	2017-09-10	IT
7	7	Ananya	29	920000.0	Pune	2016-07-04	Finance
8	8	Kiran	150	1030000.0	Hyderabad	2015-05-21	IT
9	9	Neha	28	1140000.0	Ahmedabad	2014-12-11	HR

In [17]:
```python
# Fixing Incorrect Values
df=df[df["Age"] > 0]  # Remove negative Age values
df.loc[df["Age"] > 100, "Age"] = df["Age"].mean()  # Replace unrealistic age
df
```

```
c:\users\vamsi2001\appdata\local\programs\python\python39\lib\site-packages\pan
das\core\indexing.py:1817: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
  self._setitem_single_column(loc, value, pi)
```

Out[17]:

	ID	Name	Age	Salary (INR)	City	Joining Date	Department
0	1	Amit	25.000	500000.0	Mumbai	2022-01-15	HR
1	2	Priya	30.000	540000.0	Delhi	2021-08-20	Finance
3	3	Rahul	34.000	620000.0	Bangalore	2020-06-30	IT
5	5	Vikram	40.000	NaN	Chennai	2018-03-14	HR
6	6	Raj	35.000	800000.0	Kolkata	2017-09-10	IT
7	7	Ananya	29.000	920000.0	Pune	2016-07-04	Finance
8	8	Kiran	46.375	1030000.0	Hyderabad	2015-05-21	IT
9	9	Neha	28.000	1140000.0	Ahmedabad	2014-12-11	HR

In [19]:

```
print(df['Age'].median())
# Fill missing Age with median
df["Age"].fillna(df["Age"].median(), inplace=True)  #inplace is to modify th
df
df["Age"].median()
```

```
32.0

c:\users\vamsi2001\appdata\local\programs\python\python39\lib\site-packages\pan
das\core\generic.py:6392: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
  return self._update_inplace(result)
```

Out[19]:

```
32.0
```

In [22]:

```
 # Fill missing Salary with mean
df["Salary (INR)"].fillna(df["Salary (INR)"].mean(), inplace=True)
```

```
c:\users\vamsi2001\appdata\local\programs\python\python39\lib\site-packages\pan
das\core\generic.py:6392: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
  return self._update_inplace(result)
```

In [23]:

```
df
```

Out[23]:

	ID	Name	Age	Salary (INR)	City	Joining Date	Department
0	1	Amit	25.000	5.000000e+05	Mumbai	2022-01-15	HR
1	2	Priya	30.000	5.400000e+05	Delhi	2021-08-20	Finance
3	3	Rahul	34.000	6.200000e+05	Bangalore	2020-06-30	IT
5	5	Vikram	40.000	7.928571e+05	Chennai	2018-03-14	HR
6	6	Raj	35.000	8.000000e+05	Kolkata	2017-09-10	IT
7	7	Ananya	29.000	9.200000e+05	Pune	2016-07-04	Finance
8	8	Kiran	46.375	1.030000e+06	Hyderabad	2015-05-21	IT
9	9	Neha	28.000	1.140000e+06	Ahmedabad	2014-12-11	HR

In [24]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8 entries, 0 to 9
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   ID            8 non-null      int64  
 1   Name          8 non-null      object 
 2   Age           8 non-null      float64
 3   Salary (INR)  8 non-null      float64
 4   City          8 non-null      object 
 5   Joining Date  8 non-null      object 
 6   Department    8 non-null      object 
dtypes: float64(2), int64(1), object(4)
memory usage: 512.0+ bytes
```

In [73]:

```
df
```

Out[73]:

	ID	Name	Age	Salary (INR)	City	Joining Date	Department
0	1.0	Amit	25.000000	500000.0	Mumbai	2022-01-15	HR
1	2.0	Priya	30.000000	540000.0	Delhi	2021-08-20	Finance
5	5.0	Vikram	40.000000	826000.0	Chennai	2018-03-14	HR
7	7.0	Ananya	29.000000	920000.0	Pune	2016-07-04	Finance
8	8.0	Kiran	50.333333	1030000.0	Hyderabad	2015-05-21	IT
9	9.0	Neha	28.000000	1140000.0	Ahmedabad	2014-12-11	HR

5.Fixing inconsistent data

In [ ]:

```

```

6. Renaming columns

In [74]:

```python
# Renaming Columns
df.rename(columns={"Joining Date": "Join_Date", "Department": "Dept"}, inpla
df
```

```
c:\users\vamsi2001\appdata\local\programs\python\python39\lib\site-packages\pan
das\core\frame.py:5039: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
  return super().rename(
```

Out[74]:

	ID	Name	Age	Salary (INR)	City	Join_Date	Dept
0	1.0	Amit	25.000000	500000.0	Mumbai	2022-01-15	HR
1	2.0	Priya	30.000000	540000.0	Delhi	2021-08-20	Finance
5	5.0	Vikram	40.000000	826000.0	Chennai	2018-03-14	HR
7	7.0	Ananya	29.000000	920000.0	Pune	2016-07-04	Finance
8	8.0	Kiran	50.333333	1030000.0	Hyderabad	2015-05-21	IT
9	9.0	Neha	28.000000	1140000.0	Ahmedabad	2014-12-11	HR

In [75]:

```python
# Save to CSV
df.to_csv("cleaned.csv", index=False)
```

7. Handling Outliers

Interquartile Range (IQR)

IQR (Interquartile Range) is a statistical measure used to detect outliers in a dataset. It focuses on the middle 50% of the data and helps identify values that are significantly higher or lower than the rest.

7.1. Understanding Quartiles

To understand IQR, you need to know about quartiles. Quartiles divide sorted data into four equal parts:

- **Q1 (First Quartile - 25th Percentile)**: The median (middle) of the lower half of the data.
- **Q2 (Second Quartile - 50th Percentile or Median)**: The middle value of the dataset.
- **Q3 (Third Quartile - 75th Percentile)**: The median of the upper half of the data.

Example Data (Sorted)

```
[5, 7, 9, 12, 15, 18, 21, 25, 30, 35]
```

- **Q1 (25th percentile) = 9**
- **Q2 (50th percentile / median) = 15**
- **Q3 (75th percentile) = 25**

7.2. How to Calculate IQR

Formula:

[IQR = Q3 - Q1]

Using our example: [IQR = 25 - 9 = 16]

7.3. Detecting Outliers Using IQR

Outliers are values that are too far from the middle range. We define the lower and upper bounds to detect them:

Outlier Boundaries:

{Lower Bound} = Q1 - 1.5 * IQR

{Upper Bound} = Q3 + 1.5 * IQR

Applying it to Our Example:

{Lower Bound} = 9 - (1.5 * 16) = 9 - 24 = -15

{Upper Bound} = 25 + (1.5 * 16) = 25 + 24 = 49

Outliers:

Any value less than -15 or greater than 49 is considered an outlier.
Since our dataset `[5, 7, 9, 12, 15, 18, 21, 25, 30, 35]` has no values outside these bounds, there are **no outliers**.

7.4. Visualizing IQR with a Box Plot

A **box plot** (or **box-and-whisker plot**) is a graphical way to show IQR and outliers.

Box Plot Components:

- **Box** → Shows Q1, Q2 (Median), and Q3.
- **Whiskers** → Extend to the min and max values within the **IQR range**.
- **Dots (Outliers)** → Any values **outside** the lower and upper bounds.

Here’s how we plot a box plot:

```python
import matplotlib.pyplot as plt
import seaborn as sns

data = [5, 7, 9, 12, 15, 18, 21, 25, 30, 35, 100]  # 100 is an outlier
sns.boxplot(data=data)
plt.title("Box Plot Example")
plt.show()
```

7.5. Why Use IQR?

Better than Mean & Standard Deviation: Works well for **skewed** data and **non-normal distributions**.

Robust to Outliers: Unlike standard deviation, it **focuses on the middle 50%** of data, ignoring extreme values.

Widely Used in Data Science: Helps in **data cleaning**, **preprocessing**, and **anomaly detection**.

Lets us remove outliers from dataset

In [1]:

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


# Sample real-world-like dataset (House Prices)
data = {
    "Price": [250000, 270000, 275000, 300000, 500000, 260000, 290000, 310000
    "Size_sqft": [1500, 1600, 1700, 1800, 2200, 1550, 1650, 1750, 5000, 1580
    "Bedrooms": [3, 3, 3, 4, 5, 13, 3, 4, 10, 3, 4, 4, 3, 12]
}

df = pd.DataFrame(data)

df

```

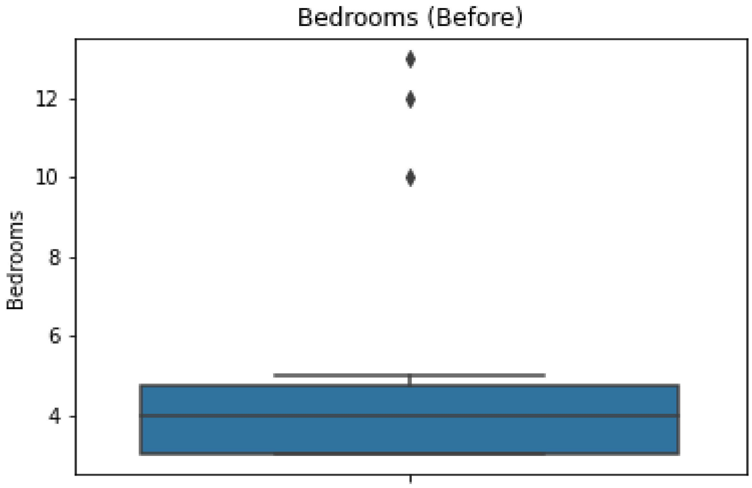
```
c:\users\vamsi2001\appdata\local\programs\python\python39\lib\site-packages\num
py\_distributor_init.py:30: UserWarning: loaded more than 1 DLL from .libs:
c:\users\vamsi2001\appdata\local\programs\python\python39\lib\site-packages\num
py\.libs\libopenblas.EL2C6PLE4ZYW3ECEVIV3OXXGRN2NRFM2.gfortran-win_amd64.dll
c:\users\vamsi2001\appdata\local\programs\python\python39\lib\site-packages\num
py\.libs\libopenblas.XWYDX2IKJW2NMTWSFYNGFUWKQU3LYTCZ.gfortran-win_amd64.dll
  warnings.warn("loaded more than 1 DLL from .libs:"
```

Out[1]:

	Price	Size_sqft	Bedrooms
0	250000	1500	3
1	270000	1600	3
2	275000	1700	3
3	300000	1800	4
4	500000	2200	5
5	260000	1550	13
6	290000	1650	3
7	310000	1750	4
8	1000000	5000	10
9	270000	1580	3
10	320000	1850	4
11	340000	1900	4
12	255000	1520	3
13	6000000	7000	12

In [2]:

```
# Plot box plots before removing outliers
import matplotlib.pyplot as plt
import seaborn as sns

sns.boxplot(y=df["Bedrooms"])
plt.title("Bedrooms (Before)")
plt.show()
```

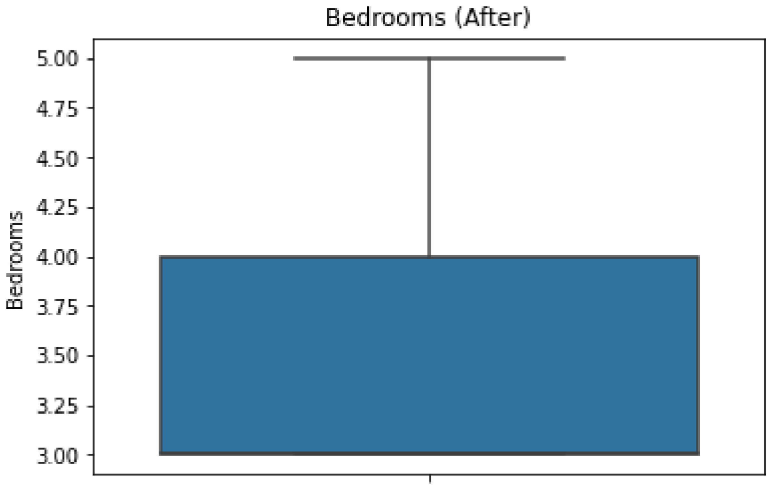


In [3]:

```
# Function to remove outliers using IQR

Q1 = df['Bedrooms'].quantile(0.25)
Q3 = df['Bedrooms'].quantile(0.75)
IQR = Q3 - Q1

lb = Q1 - 1.5* IQR
ub = Q3 + 1.5 * IQR
df=df[(df['Bedrooms'] >= lb) & (df['Bedrooms'] <= ub)]


```

In [4]:

```
#plt.subplot(1, 3, 3)
sns.boxplot(y=df["Bedrooms"])
plt.title("Bedrooms (After)")

#plt.tight_layout()
plt.show()

```



In [5]:

```
#After removing outliers
df
```

Out[5]:

	Price	Size_sqft	Bedrooms
0	250000	1500	3
1	270000	1600	3
2	275000	1700	3
3	300000	1800	4
4	500000	2200	5
6	290000	1650	3
7	310000	1750	4
9	270000	1580	3
10	320000	1850	4
11	340000	1900	4
12	255000	1520	3

Concatination

In [80]:

```
import pandas as pd

# Creating two DataFrames with the same columns
df1 = pd.DataFrame({'ID': [1, 2,3,4,5], 'Name': ['Ali', 'Bobby','Ramesh','sa
df2 = pd.DataFrame({'ID': [5, 6,7,8], 'Name': ['Cherry', 'Mahesh','Preethi',
df1

```

Out[80]:

	ID	Name	Age
0	1	Ali	25
1	2	Bobby	30
2	3	Ramesh	23
3	4	sakshi	20
4	5	sahil	24

In [81]:

```
# Concatenating along rows (axis=0)
df_concat = pd.concat([df1, df2],ignore_index=True)

print("Concatenated DataFrame (Vertical):")
print(df_concat)

```

```
Concatenated DataFrame (Vertical):
   ID     Name  Age
0   1      Ali   25
1   2    Bobby   30
2   3   Ramesh   23
3   4   sakshi   20
4   5    sahil   24
5   5   Cherry   35
6   6   Mahesh   30
7   7  Preethi   26
8   8  Santosh   28
```

In [82]:

```python
df3 = pd.DataFrame({'City': ['HYD', 'BEN','VIJ','CHE'], 'Salary': [50000, 60

# Concatenating along columns (axis=1)
df_concat = pd.concat([df1, df3], axis=1)

print("Concatenated DataFrame (Horizontal):")
print(df_concat)

```

```
Concatenated DataFrame (Horizontal):
   ID    Name  Age City   Salary
0   1     Ali   25  HYD  50000.0
1   2   Bobby   30  BEN  60000.0
2   3  Ramesh   23  VIJ  70000.0
3   4  sakshi   20  CHE  34000.0
4   5   sahil   24  NaN      NaN
```

8. Standardization of Data

In [7]:

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

# Sample Data
data = {'Age': [18, 25, 35, 60], 'Salary': [30000, 50000, 100000, 200000]}
df = pd.DataFrame(data)

df
```

Out[7]:

	Age	Salary
0	18	30000
1	25	50000
2	35	100000
3	60	200000

In [111]:

```
# Calculate mean and standard deviation
mean = df.mean()
std_dev = df.std()

# Apply Z-score formula
df= (df - mean) / std_dev
df_std
df

```

Out[111]:

	Age	Salary
0	-0.897925	-0.855955
1	-0.516987	-0.592584
2	0.027210	0.065843
3	1.387702	1.382697

In [13]:

```
from sklearn.preprocessing import MinMaxScaler
sc=MinMaxScaler()
df=sc.fit_transform(df)
df
```

Out[13]:

```
array([[0.        , 0.        ],
       [0.16666667, 0.11764706],
       [0.4047619 , 0.41176471],
       [1.        , 1.        ]])
```

9. Type Conversions

In [2]:

```
import pandas as pd

data = {
    'ID': ['101', '102', '103', '104'],  # Should be int
    'Price': ['100.5', '200.0', 'invalid', '400.75'],  # Should be float
    'Date': ['2024-01-01', '2024-02-15', '2024-03-20', '2024-04-10'],  # Sho
    'Category': ['A', 'B', 'C', 'D']  # Should remain as object (string)
}

df = pd.DataFrame(data)


df.head()
```

Out[2]:

	ID	Price	Date	Category
0	101	100.5	2024-01-01	A
1	102	200.0	2024-02-15	B
2	103	invalid	2024-03-20	C
3	104	400.75	2024-04-10	D

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   ID        4 non-null      object
 1   Price     4 non-null      object
 2   Date      4 non-null      object
 3   Category  4 non-null      object
dtypes: object(4)
memory usage: 256.0+ bytes
```

In [4]:

```
#'ID' is stored as an object, we convert it to an integer.
df['ID'] = df['ID'].astype(int)
df['ID'].dtype
```

Out[4]:

```
dtype('int32')
```

In [6]:

```
#some values are non-numeric ('invalid')
df['Price'] = pd.to_numeric(df['Price'], errors='coerce')  # Converts 'inval
df['Price'].dtype
df.dtypes
```

Out[6]:

```
ID             int32
Price        float64
Date          object
Category      object
dtype: object
```

In [8]:

```
#convert date column from object to date
df['Date']=pd.to_datetime(df['Date'])
df['Date'].dtype
```

Out[8]:

```
dtype('<M8[ns]')
```

In [10]:

```
#convert
df['Category'] = df['Category'].astype('category')
df['Category'].dtype
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype         
---  ------    --------------  -----         
 0   ID        4 non-null      int32         
 1   Price     3 non-null      float64       
 2   Date      4 non-null      datetime64[ns]
 3   Category  4 non-null      category      
dtypes: category(1), datetime64[ns](1), float64(1), int32(1)
memory usage: 416.0 bytes
```

In [12]:

```python
import pandas as pd

data = {
    'Product': ['A', 'B', 'C', 'D'],
    'Price': ['$1,000', '$2,500', '$3,750', '$4,100']
}

df = pd.DataFrame(data)

df.dtypes
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   Product  4 non-null      object
 1   Price    4 non-null      object
dtypes: object(2)
memory usage: 192.0+ bytes
```

In [15]:

```python
#df['Price']=pd.to_numeric(df['Price'])
#df['Price']=df['Price'].astype(int)
```

In [17]:

```python
# Remove the dollar sign and commas, then convert to float
df['Price'] = df['Price'].replace({'\$': '', ',': ''}, regex=True).astype(fl

# After conversion: Check the data types
df.dtypes
df
```

Out[17]:

	Product	Price
0	A	1000.0
1	B	2500.0
2	C	3750.0
3	D	4100.0

In [ ]:

```python

```