Assignment 2: Data Analysis with pandas and Visualization Libraries

Name: Navneet Yadav

Roll no. 2301560044

Program: MCA-I

Subject: AI-ML

Question 2

- GitHub Link: https://github.com/navYadav20/AI-ML- (https://github.com/navYadav20/AI-ML-)

---

1. Data type changes(astype)

In [1]:
```
import pandas as pd

data = {
    'Product_ID': [101, 102, 103, 104, 105],
    'Product_Name': ['iPhone', 'MacBook Pro', 'iPad', 'Apple Watch', 'AirPo
    'Release_Year': ['2020', '2019', '2018', '2021', '2019'],
    'Price ($)': [999, 1299, 329, 399, 159],
    'Units_Sold_Millions': ['75', '20', '40', '25', '30']
}

apple_df = pd.DataFrame(data)
apple_df
```

Out[1]:

	Product_ID	Product_Name	Release_Year	Price ($)	Units_Sold_Millions
0	101	iPhone	2020	999	75
1	102	MacBook Pro	2019	1299	20
2	103	iPad	2018	329	40
3	104	Apple Watch	2021	399	25
4	105	AirPods	2019	159	30

In [3]:
```
apple_df.dtypes
```

Out[3]:
```
Product_ID              int64
Product_Name           object
Release_Year           object
Price ($)               int64
Units_Sold_Millions    object
dtype: object
```

How would you convert the 'Release_Year' column from string type to integer type?

In [4]:
```
apple_df.astype({'Release_Year': 'int64'}).dtypes
```

Out[4]:
```
Product_ID              int64
Product_Name           object
Release_Year            int64
Price ($)               int64
Units_Sold_Millions    object
dtype: object
```

Can you use astype to change the data type of the 'Price ($)' column to a float?

In [5]:
```
apple_df.astype({'Price ($)': 'float64'}).dtypes
```

Out[5]:
```
Product_ID              int64
Product_Name           object
Release_Year           object
Price ($)             float64
Units_Sold_Millions    object
dtype: object
```

How can you change the 'Units_Sold_Millions' column from string to integer data type?

In [6]:
```
apple_df.astype({'Units_Sold_Millions': 'int64'}).dtypes
```

Out[6]:
```
Product_ID              int64
Product_Name           object
Release_Year           object
Price ($)               int64
Units_Sold_Millions     int64
dtype: object
```

If you wanted the 'Product_ID' column to be of string type for joining with another dataset, how would you use astype to achieve this?

In [7]:
```
apple_df.astype({'Product_ID': 'str'}).dtypes
```

Out[7]:
```
Product_ID             object
Product_Name           object
Release_Year           object
Price ($)               int64
Units_Sold_Millions    object
dtype: object
```

Can you transform the 'Units_Sold_Millions' column into a float type and ensure that the column represents the actual number of units (i.e., not in millions) using a single line of code?

In [8]:
```
apple_df.loc[:, ['Units_Sold_Millions']].astype('float64').apply(lambda x:
```

Out[8]:

	Units_Sold_Millions
0	75000000.0
1	20000000.0
2	40000000.0
3	25000000.0
4	30000000.0

How would you convert all string columns in the DataFrame to uppercase while changing their data type to a category?

In [9]:
```
apple_df.astype('category').dtypes
```

Out[9]:
```
Product_ID             category
Product_Name           category
Release_Year           category
Price ($)              category
Units_Sold_Millions    category
dtype: object
```

In [10]:
```
apple_df.select_dtypes(include=['object']).applymap(lambda x: x.upper() if
```

```
C:\Users\msuse\AppData\Local\Temp\ipykernel_1232\630351609.py:1: FutureWar
ning: DataFrame.applymap has been deprecated. Use DataFrame.map instead.
  apple_df.select_dtypes(include=['object']).applymap(lambda x: x.upper()
if isinstance(x, str) else x).astype('category')
```

Out[10]:

	Product_Name	Release_Year	Units_Sold_Millions
0	IPHONE	2020	75
1	MACBOOK PRO	2019	20
2	IPAD	2018	40
3	APPLE WATCH	2021	25
4	AIRPODS	2019	30

2. Between Method

In [2]:
```
import pandas as pd
import datetime

from random import randint
```

In [13]:
```
data = {
    'Coffee_Shop': ['Brewed Awakening', 'Coffee Cloud', 'Bean Dream', 'Espr
    'Mocha Magic', 'Cafe Comfort', 'Seattle Sip', 'Drip Drop', 'Grind Groun
    'Location': ['Downtown', 'Capitol Hill', 'Green Lake', 'Ballard', 'West
    'Fremont', 'Queen Anne', 'Belltown', 'University District', 'Magnolia']
    'Avg_Rating': [4.5, 4.2, 5.0, 4.8, 4.6, 4.3, 4.9, 4.0, 4.7, 4.6],
    'Coffee_Type': ['Espresso', 'Latte', 'Cappuccino', 'Americano', 'Cold B
    'Macchiato', 'Espresso', 'Latte', 'Drip Coffee', 'Americano'],
    'Tables_Available': [5, 8, 6, 7, 5, 9, 7, 8, 6, 5]
}
```

In [3]:
```
coffee_shops_df = pd.DataFrame(data)
coffee_shops_df
```

Out[3]:

	Product_ID	Product_Name	Release_Year	Price ($)	Units_Sold_Millions
0	101	iPhone	2020	999	75
1	102	MacBook Pro	2019	1299	20
2	103	iPad	2018	329	40
3	104	Apple Watch	2021	399	25
4	105	AirPods	2019	159	30

How would you use the between method to filter rows from the coffee_shops_df where Avg_Rating is between 4.5 and 5.0, inclusive?

In [16]:
```
coffee_shops_df.loc[coffee_shops_df['Avg_Rating'].between(4.5, 5.0)]
```

Out[16]:

	Coffee_Shop	Location	Avg_Rating	Coffee_Type	Tables_Available
0	Brewed Awakening	Downtown	4.5	Espresso	5
2	Bean Dream	Green Lake	5.0	Cappuccino	6
3	Espresso Express	Ballard	4.8	Americano	7
4	Latte Love	West Seattle	4.6	Cold Brew	5
6	Cafe Comfort	Queen Anne	4.9	Espresso	7
8	Drip Drop	University District	4.7	Drip Coffee	6
9	Grind Ground	Magnolia	4.6	Americano	5

Which coffee shops from the DataFrame are located in areas between 'Ballard' and 'Fremont' alphabetically using the between method?

In [17]:
```
coffee_shops_df.loc[coffee_shops_df['Location'].between('Ballard', 'Fremont
```

Out[17]:

	Coffee_Shop	Location	Avg_Rating	Coffee_Type	Tables_Available
0	Brewed Awakening	Downtown	4.5	Espresso	5
1	Coffee Cloud	Capitol Hill	4.2	Latte	8
3	Espresso Express	Ballard	4.8	Americano	7
5	Mocha Magic	Fremont	4.3	Macchiato	9
7	Seattle Sip	Belltown	4.0	Latte	8

How can you use the between method to identify coffee shops that have between 5 and 7 tables available?

In [18]:
```
coffee_shops_df.loc[coffee_shops_df['Tables_Available'].between(5, 7)]
```

Out[18]:

	Coffee_Shop	Location	Avg_Rating	Coffee_Type	Tables_Available
0	Brewed Awakening	Downtown	4.5	Espresso	5
2	Bean Dream	Green Lake	5.0	Cappuccino	6
3	Espresso Express	Ballard	4.8	Americano	7
4	Latte Love	West Seattle	4.6	Cold Brew	5
6	Cafe Comfort	Queen Anne	4.9	Espresso	7
8	Drip Drop	University District	4.7	Drip Coffee	6
9	Grind Ground	Magnolia	4.6	Americano	5

Given a DataFrame df with a column 'Names', how would you filter rows where the name length is between 3 and 7 characters using the between method?

In [20]:
```
coffee_shops_df['Names'] = ['Sip','Mocha','Java','Café Gemstone','Sugar','C
coffee_shops_df
```

Out[20]:

	Coffee_Shop	Location	Avg_Rating	Coffee_Type	Tables_Available	Names
0	Brewed Awakening	Downtown	4.5	Espresso	5	Sip
1	Coffee Cloud	Capitol Hill	4.2	Latte	8	Mocha
2	Bean Dream	Green Lake	5.0	Cappuccino	6	Java
3	Espresso Express	Ballard	4.8	Americano	7	Café Gemstone
4	Latte Love	West Seattle	4.6	Cold Brew	5	Sugar
5	Mocha Magic	Fremont	4.3	Macchiato	9	Coffee
6	Cafe Comfort	Queen Anne	4.9	Espresso	7	Velvet
7	Seattle Sip	Belltown	4.0	Latte	8	Bliss
8	Drip Drop	University District	4.7	Drip Coffee	6	Cappuccino
9	Grind Ground	Magnolia	4.6	Americano	5	Grace

If coffee_shops_df has a 'Barista' column with names of baristas, how would you use between to select only the rows where the barista's name length is between 4 and 8 characters?

In [22]:
```
barista_names = [
    "Eva",
    "Liam",
    "Zara",
    "Jake",
    "Maya",
    "Lucas",
    "Nina",
    "Alex",
    "Sophia",
    "Henry"
]

coffee_shops_df['Barista'] = barista_names
coffee_shops_df
```

Out[22]:

	Coffee_Shop	Location	Avg_Rating	Coffee_Type	Tables_Available	Names	Barista
0	Brewed Awakening	Downtown	4.5	Espresso	5	Sip	Eva
1	Coffee Cloud	Capitol Hill	4.2	Latte	8	Mocha	Liam
2	Bean Dream	Green Lake	5.0	Cappuccino	6	Java	Zara
3	Espresso Express	Ballard	4.8	Americano	7	Café Gemstone	Jake
4	Latte Love	West Seattle	4.6	Cold Brew	5	Sugar	Maya
5	Mocha Magic	Fremont	4.3	Macchiato	9	Coffee	Lucas
6	Cafe Comfort	Queen Anne	4.9	Espresso	7	Velvet	Nina
7	Seattle Sip	Belltown	4.0	Latte	8	Bliss	Alex
8	Drip Drop	University District	4.7	Drip Coffee	6	Cappuccino	Sophia
9	Grind Ground	Magnolia	4.6	Americano	5	Grace	Henry

In [23]:
```
coffee_shops_df[coffee_shops_df['Barista'].str.len().between(4, 8)]
```

Out[23]:

	Coffee_Shop	Location	Avg_Rating	Coffee_Type	Tables_Available	Names	Barista
1	Coffee Cloud	Capitol Hill	4.2	Latte	8	Mocha	Liam
2	Bean Dream	Green Lake	5.0	Cappuccino	6	Java	Zara
3	Espresso Express	Ballard	4.8	Americano	7	Café Gemstone	Jake
4	Latte Love	West Seattle	4.6	Cold Brew	5	Sugar	Maya
5	Mocha Magic	Fremont	4.3	Macchiato	9	Coffee	Lucas
6	Cafe Comfort	Queen Anne	4.9	Espresso	7	Velvet	Nina
7	Seattle Sip	Belltown	4.0	Latte	8	Bliss	Alex
8	Drip Drop	University District	4.7	Drip Coffee	6	Cappuccino	Sophia
9	Grind Ground	Magnolia	4.6	Americano	5	Grace	Henry

In [ ]:
```
```

3. Unique and Duplicate

In [24]:
```
data = {
    'Product_ID': [1, 2, 3, 4, 5, 6, 1, 2, 3, 4],
    'Product_Name': ['Toothpaste', 'Shampoo', 'Laptop', 'TV', 'Coffee Maker
    'Category': ['Personal Care', 'Personal Care', 'Electronics', 'Electron
    'Price': [5.0, 10.0, 1000.0, 600.0, 50.0, 30.0, 5.0, 10.0, 1000.0, 600.
    'Units_Sold': [100, 50, 20, 15, 30, 25, 110, 55, 22, 17]
}
df = pd.DataFrame(data)
df
```

Out[24]:

	Product_ID	Product_Name	Category	Price	Units_Sold
0	1	Toothpaste	Personal Care	5.0	100
1	2	Shampoo	Personal Care	10.0	50
2	3	Laptop	Electronics	1000.0	20
3	4	TV	Electronics	600.0	15
4	5	Coffee Maker	Kitchen Appliances	50.0	30
5	6	Blender	Kitchen Appliances	30.0	25
6	1	Toothpaste	Personal Care	5.0	110
7	2	Shampoo	Personal Care	10.0	55
8	3	Laptop	Electronics	1000.0	22
9	4	TV	Electronics	600.0	17

Duplicates

- How would you identify if there are any duplicate rows in the DataFrame? What function would you use?

In [25]:
```
df.duplicated()
```

Out[25]:
```
0    False
1    False
2    False
3    False
4    False
5    False
6    False
7    False
8    False
9    False
dtype: bool
```

If the DataFrame has a column for 'Product_ID', how would you find out if there are any duplicate 'Product_ID' entries?

In [26]:
```
df.loc[df['Product_ID'].duplicated(), ['Product_ID']]
```

Out[26]:

	Product_ID
6	1
7	2
8	3
9	4

If the DataFrame has columns 'Product_ID' and 'Product_Category', how would you remove rows where only 'Product_ID' is duplicated but keep the first occurrence?

In [31]:
```
df1.drop_duplicates(subset=['Product_ID'], keep='first')
```

Out[31]:

	Product_ID	Product_Name	Category	Price	Units_Sold
0	1	Toothpaste	Personal Care	5.0	100
1	2	Shampoo	Personal Care	10.0	50
2	3	Laptop	Electronics	1000.0	20
3	4	TV	Electronics	600.0	15
4	5	Coffee Maker	Kitchen Appliances	50.0	30
5	6	Blender	Kitchen Appliances	30.0	25

Is it possible to keep the last occurrence of a duplicate 'Product_ID' and remove the rest? If so, how would you do it?

In [32]:
```
df1.drop_duplicates(subset=['Product_ID'], keep='last')
```

Out[32]:

	Product_ID	Product_Name	Category	Price	Units_Sold
4	5	Coffee Maker	Kitchen Appliances	50.0	30
5	6	Blender	Kitchen Appliances	30.0	25
6	1	Toothpaste	Personal Care	5.0	110
7	2	Shampoo	Personal Care	10.0	55
8	3	Laptop	Electronics	1000.0	22
9	4	TV	Electronics	600.0	17

DROP_DUPLICATES

How would you use the drop_duplicates function to remove all rows that have identical values across all columns?

In [34]:
```
df1.drop_duplicates()
```

Out[34]:

	Product_ID	Product_Name	Category	Price	Units_Sold
0	1	Toothpaste	Personal Care	5.0	100
1	2	Shampoo	Personal Care	10.0	50
2	3	Laptop	Electronics	1000.0	20
3	4	TV	Electronics	600.0	15
4	5	Coffee Maker	Kitchen Appliances	50.0	30
5	6	Blender	Kitchen Appliances	30.0	25
6	1	Toothpaste	Personal Care	5.0	110
7	2	Shampoo	Personal Care	10.0	55
8	3	Laptop	Electronics	1000.0	22
9	4	TV	Electronics	600.0	17

If your DataFrame contains columns 'Product_ID', 'Product_Name', and 'Price', how would you use drop_duplicates to keep only the unique 'Product_ID' rows, while retaining the first occurrence of each 'Product_ID'?

In [35]:
```
df1.drop_duplicates(subset=['Product_ID', 'Product_Name', 'Price'], keep='f
```

Out[35]:

	Product_ID	Product_Name	Category	Price	Units_Sold
0	1	Toothpaste	Personal Care	5.0	100
1	2	Shampoo	Personal Care	10.0	50
2	3	Laptop	Electronics	1000.0	20
3	4	TV	Electronics	600.0	15
4	5	Coffee Maker	Kitchen Appliances	50.0	30
5	6	Blender	Kitchen Appliances	30.0	25

4. Null values operations

In [39]:
```
import pandas as pd
import numpy as np
data = {
    'Product_ID': [1, 2, 3, 4, 5, np.nan],
    'Product_Name': ['Laptop', 'Phone', 'TV', 'Headphones', None, 'Microwav
    'Stock': [20, 15, np.nan, 30, 25, 12],
    'Price': [999.99, 799.99, 399.99, np.nan, 59.99, 99.99],
    'Discounted': [True, False, True, False, np.nan, True],
}

target_df = pd.DataFrame(data)
target_df
```

Out[39]:

	Product_ID	Product_Name	Stock	Price	Discounted
0	1.0	Laptop	20.0	999.99	True
1	2.0	Phone	15.0	799.99	False
2	3.0	TV	NaN	399.99	True
3	4.0	Headphones	30.0	NaN	False
4	5.0	None	25.0	59.99	NaN
5	NaN	Microwave	12.0	99.99	True

Identify Null Values

1. How would you identify rows where the 'Stock' column has null values in the provided target_df DataFrame?

In [40]:
```
target_df.loc[target_df.loc[:, 'Stock'].isna() == True]
```

Out[40]:

	Product_ID	Product_Name	Stock	Price	Discounted
2	3.0	TV	NaN	399.99	True

2. Can you list the columns in target_df that contain at least one null value?

In [41]:
```
target_df.isna().any()
```

Out[41]:
```
Product_ID      True
Product_Name    True
Stock           True
Price           True
Discounted      True
dtype: bool
```

3. If you wanted to identify rows where both 'Stock' and 'Price' have null values, how would you do it?

In [42]:
```
target_df.loc[(target_df.loc[:, "Stock"].isna() == True) & (target_df.loc[:
```

Out[42]:

	Product_ID	Product_Name	Stock	Price	Discounted

In [43]:
```
target_df
```

Out[43]:

	Product_ID	Product_Name	Stock	Price	Discounted
0	1.0	Laptop	20.0	999.99	True
1	2.0	Phone	15.0	799.99	False
2	3.0	TV	NaN	399.99	True
3	4.0	Headphones	30.0	NaN	False
4	5.0	None	25.0	59.99	NaN
5	NaN	Microwave	12.0	99.99	True

In [44]:
```
target_df.loc[6] = [6.0, "Monitor", np.nan, np.nan, True]
target_df
```

Out[44]:

	Product_ID	Product_Name	Stock	Price	Discounted
0	1.0	Laptop	20.0	999.99	True
1	2.0	Phone	15.0	799.99	False
2	3.0	TV	NaN	399.99	True
3	4.0	Headphones	30.0	NaN	False
4	5.0	None	25.0	59.99	NaN
5	NaN	Microwave	12.0	99.99	True
6	6.0	Monitor	NaN	NaN	True

In [45]:
```
target_df.loc[(target_df.loc[:, "Stock"].isna() == True) & (target_df.loc[:
```

Out[45]:

	Product_ID	Product_Name	Stock	Price	Discounted
6	6.0	Monitor	NaN	NaN	True

Filtering

- How would you filter out rows where the 'Stock' column has null values in the target_df DataFrame?

In [46]:
```
target_df.loc[target_df.loc[:, "Stock"].isna() == True]
```

Out[46]:

	Product_ID	Product_Name	Stock	Price	Discounted
2	3.0	TV	NaN	399.99	True
6	6.0	Monitor	NaN	NaN	True

- What method would you use to remove any row that has at least one null value in target_df?

In [47]:
```
temp = target_df.copy()
temp
```

Out[47]:

	Product_ID	Product_Name	Stock	Price	Discounted
0	1.0	Laptop	20.0	999.99	True
1	2.0	Phone	15.0	799.99	False
2	3.0	TV	NaN	399.99	True
3	4.0	Headphones	30.0	NaN	False
4	5.0	None	25.0	59.99	NaN
5	NaN	Microwave	12.0	99.99	True
6	6.0	Monitor	NaN	NaN	True

In [48]:
```
temp = temp.dropna(axis=0)
temp
```

Out[48]:

	Product_ID	Product_Name	Stock	Price	Discounted
0	1.0	Laptop	20.0	999.99	True
1	2.0	Phone	15.0	799.99	False

- If you only want to drop rows where both 'Stock' and 'Price' have null values, how would you go about doing this in target_df?

In [49]:
```
temp = target_df.copy()
temp
```

Out[49]:

	Product_ID	Product_Name	Stock	Price	Discounted
0	1.0	Laptop	20.0	999.99	True
1	2.0	Phone	15.0	799.99	False
2	3.0	TV	NaN	399.99	True
3	4.0	Headphones	30.0	NaN	False
4	5.0	None	25.0	59.99	NaN
5	NaN	Microwave	12.0	99.99	True
6	6.0	Monitor	NaN	NaN	True

In [50]:
```
temp.dropna(subset=['Stock', 'Price'])
```

Out[50]:

	Product_ID	Product_Name	Stock	Price	Discounted
0	1.0	Laptop	20.0	999.99	True
1	2.0	Phone	15.0	799.99	False
4	5.0	None	25.0	59.99	NaN
5	NaN	Microwave	12.0	99.99	True

Filling

- How would you fill null values in the 'Stock' column with the median value of that column?

In [51]:
```
temp
```

Out[51]:

	Product_ID	Product_Name	Stock	Price	Discounted
0	1.0	Laptop	20.0	999.99	True
1	2.0	Phone	15.0	799.99	False
2	3.0	TV	NaN	399.99	True
3	4.0	Headphones	30.0	NaN	False
4	5.0	None	25.0	59.99	NaN
5	NaN	Microwave	12.0	99.99	True
6	6.0	Monitor	NaN	NaN	True

In [52]:
```
stock_median = np.median(temp.loc[temp['Stock'].notna(), ["Stock"]])
stock_median
```

Out[52]:
```
20.0
```

In [53]:
```
temp.loc[temp['Stock'].isna(), ["Stock"]] = stock_median
temp
```

Out[53]:

	Product_ID	Product_Name	Stock	Price	Discounted
0	1.0	Laptop	20.0	999.99	True
1	2.0	Phone	15.0	799.99	False
2	3.0	TV	20.0	399.99	True
3	4.0	Headphones	30.0	NaN	False
4	5.0	None	25.0	59.99	NaN
5	NaN	Microwave	12.0	99.99	True
6	6.0	Monitor	20.0	NaN	True

5. Concat, extend and merge

In [55]:
```
data = {
    'Shop_ID' : [1, 2, 3, 4, 2, 5, 6],
    'Shop_Name': ['Starbucks', 'Blue Bottle', 'Verve Coffee', 'Stumptown',
    'Location': ['Hollywood', 'San Diego', 'Hollywood', 'San Diego', 'New Y
    'Rating': [4.5, 4.8, 4.4, 4.3, 4.6, 4.2, 4.7],
    'Revenue': [50000, 30000, 25000, 28000, 52000, 20000, 24000]
}
coffee_shops = pd.DataFrame(data)
coffee_shops
```

Out[55]:

	Shop_ID	Shop_Name	Location	Rating	Revenue
0	1	Starbucks	Hollywood	4.5	50000
1	2	Blue Bottle	San Diego	4.8	30000
2	3	Verve Coffee	Hollywood	4.4	25000
3	4	Stumptown	San Diego	4.3	28000
4	2	Blue Bottle	New York	4.6	52000
5	5	Gregorys Coffee	New York	4.2	20000
6	6	Cafe Grumpy	New York	4.7	24000

In [56]:
```
suppliers_data = {
'Shop_ID': [1, 2, 3, 4, 5, 6, 7],
'Shop_Name': ['Starbucks', 'Blue Bottle', 'Verve Coffee', 'Stumptown', 'Gre
'Supplier_Name': ['Beans R Us', 'Premium Beans', 'South Coffee Suppliers',
'Delivery_Days': [7, 5, 7, 4, 6,9, 10]
}

suppliers_df = pd.DataFrame(suppliers_data)
coffee_shops
```

Out[56]:

	Shop_ID	Shop_Name	Location	Rating	Revenue
0	1	Starbucks	Hollywood	4.5	50000
1	2	Blue Bottle	San Diego	4.8	30000
2	3	Verve Coffee	Hollywood	4.4	25000
3	4	Stumptown	San Diego	4.3	28000
4	2	Blue Bottle	New York	4.6	52000
5	5	Gregorys Coffee	New York	4.2	20000
6	6	Cafe Grumpy	New York	4.7	24000

Basic Merge:

- How would you merge coffee_shops with suppliers_df based on a common column named Shop_ID?

In [57]:
```
coffee_shops.merge(suppliers_df, on='Shop_ID')
```

Out[57]:

	Shop_ID	Shop_Name_x	Location	Rating	Revenue	Shop_Name_y	Supplier_Name	Deliv
0	1	Starbucks	Hollywood	4.5	50000	Starbucks	Beans R Us	
1	2	Blue Bottle	San Diego	4.8	30000	Blue Bottle	Premium Beans	
2	2	Blue Bottle	New York	4.6	52000	Blue Bottle	Premium Beans	
3	3	Verve Coffee	Hollywood	4.4	25000	Verve Coffee	South Coffee Suppliers	
4	4	Stumptown	San Diego	4.3	28000	Stumptown	San Diego Beans	
5	5	Gregorys Coffee	New York	4.2	20000	Gregorys Coffee	NY Fresh Beans	
6	6	Cafe Grumpy	New York	4.7	24000	Blue Bottle	Mexico tea	

Left Merge:

- If you wanted to retain all records from coffee_shops and only the matching records from suppliers_df based on the Shop_ID column, which type of merge would you use?

In [58]:
```
suppliers_df.loc[5] = [7, 'Chaayos', 'Tata Coffee', 9]
coffee_shops
```

Out[58]:

	Shop_ID	Shop_Name	Location	Rating	Revenue
0	1	Starbucks	Hollywood	4.5	50000
1	2	Blue Bottle	San Diego	4.8	30000
2	3	Verve Coffee	Hollywood	4.4	25000
3	4	Stumptown	San Diego	4.3	28000
4	2	Blue Bottle	New York	4.6	52000
5	5	Gregorys Coffee	New York	4.2	20000
6	6	Cafe Grumpy	New York	4.7	24000

In [59]:
```
suppliers_df
```

Out[59]:

	Shop_ID	Shop_Name	Supplier_Name	Delivery_Days
0	1	Starbucks	Beans R Us	7
1	2	Blue Bottle	Premium Beans	5
2	3	Verve Coffee	South Coffee Suppliers	7
3	4	Stumptown	San Diego Beans	4
4	5	Gregorys Coffee	NY Fresh Beans	6
5	7	Chaayos	Tata Coffee	9
6	7	Gregorys Coffee	Chayyos	10

In [60]:
```
coffee_shops.merge(suppliers_df, on='Shop_ID', how='left')
```

Out[60]:

	Shop_ID	Shop_Name_x	Location	Rating	Revenue	Shop_Name_y	Supplier_Name	Deliv
0	1	Starbucks	Hollywood	4.5	50000	Starbucks	Beans R Us	
1	2	Blue Bottle	San Diego	4.8	30000	Blue Bottle	Premium Beans	
2	3	Verve Coffee	Hollywood	4.4	25000	Verve Coffee	South Coffee Suppliers	
3	4	Stumptown	San Diego	4.3	28000	Stumptown	San Diego Beans	
4	2	Blue Bottle	New York	4.6	52000	Blue Bottle	Premium Beans	
5	5	Gregorys Coffee	New York	4.2	20000	Gregorys Coffee	NY Fresh Beans	
6	6	Cafe Grumpy	New York	4.7	24000	NaN	NaN	

Right Merge:

- Assuming that suppliers_df has some Shop_ID entries that don't exist in coffee_shops, how would you merge such that you retain all supplier records and only matching coffee shop records?

In [61]:
```
coffee_shops.merge(suppliers_df, on='Shop_ID', how='right')
```

Out[61]:

	Shop_ID	Shop_Name_x	Location	Rating	Revenue	Shop_Name_y	Supplier_Name	Deliv
0	1	Starbucks	Hollywood	4.5	50000.0	Starbucks	Beans R Us	
1	2	Blue Bottle	San Diego	4.8	30000.0	Blue Bottle	Premium Beans	
2	2	Blue Bottle	New York	4.6	52000.0	Blue Bottle	Premium Beans	
3	3	Verve Coffee	Hollywood	4.4	25000.0	Verve Coffee	South Coffee Suppliers	
4	4	Stumptown	San Diego	4.3	28000.0	Stumptown	San Diego Beans	
5	5	Gregorys Coffee	New York	4.2	20000.0	Gregorys Coffee	NY Fresh Beans	
6	7	NaN	NaN	NaN	NaN	Chaayos	Tata Coffee	
7	7	NaN	NaN	NaN	NaN	Gregorys Coffee	Chayyos	

Outer Merge:

- If you wanted to retain all records from both coffee_shops and suppliers_df, including those without matching Shop_ID, which merge strategy would you employ?

In [62]:
```
coffee_shops.merge(suppliers_df, on='Shop_ID', how='outer')
```

Out[62]:

	Shop_ID	Shop_Name_x	Location	Rating	Revenue	Shop_Name_y	Supplier_Name	Deliv
0	1	Starbucks	Hollywood	4.5	50000.0	Starbucks	Beans R Us	
1	2	Blue Bottle	San Diego	4.8	30000.0	Blue Bottle	Premium Beans	
2	2	Blue Bottle	New York	4.6	52000.0	Blue Bottle	Premium Beans	
3	3	Verve Coffee	Hollywood	4.4	25000.0	Verve Coffee	South Coffee Suppliers	
4	4	Stumptown	San Diego	4.3	28000.0	Stumptown	San Diego Beans	
5	5	Gregorys Coffee	New York	4.2	20000.0	Gregorys Coffee	NY Fresh Beans	
6	6	Cafe Grumpy	New York	4.7	24000.0	NaN	NaN	
7	7	NaN	NaN	NaN	NaN	Chaayos	Tata Coffee	
8	7	NaN	NaN	NaN	NaN	Gregorys Coffee	Chayyos	

Suffix Handling:

- If both coffee_shops and suppliers_df have a column named Rating and you're merging them on Shop_ID, how would you differentiate between the Rating columns in the

In [65]:
```
coffee_shops
```

Out[65]:

	Shop_ID	Shop_Name	Location	Rating	Revenue
0	1	Starbucks	Hollywood	4.5	50000
1	2	Blue Bottle	San Diego	4.8	30000
2	3	Verve Coffee	Hollywood	4.4	25000
3	4	Stumptown	San Diego	4.3	28000
4	2	Blue Bottle	New York	4.6	52000
5	5	Gregorys Coffee	New York	4.2	20000
6	6	Cafe Grumpy	New York	4.7	24000

In [64]:
```
suppliers_df
```

Out[64]:

	Shop_ID	Shop_Name	Supplier_Name	Delivery_Days
0	1	Starbucks	Beans R Us	7
1	2	Blue Bottle	Premium Beans	5
2	3	Verve Coffee	South Coffee Suppliers	7
3	4	Stumptown	San Diego Beans	4
4	5	Gregorys Coffee	NY Fresh Beans	6
5	7	Chaayos	Tata Coffee	9
6	7	Gregorys Coffee	Chayyos	10

In [67]:
```
suppliers_df['Rating'] = [4.6, 3.4, 4.2, 4.3, 4.1, 4.5,4.8]
suppliers_df
```

Out[67]:

	Shop_ID	Shop_Name	Supplier_Name	Delivery_Days	Rating
0	1	Starbucks	Beans R Us	7	4.6
1	2	Blue Bottle	Premium Beans	5	3.4
2	3	Verve Coffee	South Coffee Suppliers	7	4.2
3	4	Stumptown	San Diego Beans	4	4.3
4	5	Gregorys Coffee	NY Fresh Beans	6	4.1
5	7	Chaayos	Tata Coffee	9	4.5
6	7	Gregorys Coffee	Chayyos	10	4.8

In [68]:
```
coffee_shops.merge(suppliers_df, on='Shop_ID', suffixes=["_coffee", "_suppl
```

Out[68]:

	Shop_ID	Shop_Name_coffee	Location	Rating_coffee	Revenue	Shop_Name_supplier	Su
0	1	Starbucks	Hollywood	4.5	50000	Starbucks	
1	2	Blue Bottle	San Diego	4.8	30000	Blue Bottle	Pr
2	2	Blue Bottle	New York	4.6	52000	Blue Bottle	Pr
3	3	Verve Coffee	Hollywood	4.4	25000	Verve Coffee	
4	4	Stumptown	San Diego	4.3	28000	Stumptown	
5	5	Gregorys Coffee	New York	4.2	20000	Gregorys Coffee	