Lab5

October 1, 2024

1 STOR 320: Introduction to Data Science

2 Lab 5

[ ]:
```
# Just run this cell
import numpy as np
import pandas as pd
```

2.1 Combining Datasets

1. You are given the following Series and DataFrames that represent sales and returns data for different products over various months.

[ ]:
```
# Sales data for different products
sales_data = pd.DataFrame({
    'Product_A': [100, 150, 200],
    'Product_B': [80, 90, 120]
}, index=['Jan', 'Feb', 'Mar'])

# Returns data for different products
returns_data = pd.DataFrame({
    'Product_A': [5, 7, 6],
    'Product_B': [2, 3, 4]
}, index=['Jan', 'Feb', 'Mar'])

# Additional sales data for Product_C
additional_sales = pd.Series([50, 60, 70], index=['Feb', 'Mar', 'Apr'],
                             name='Product_C')
```

1.1 Concatenate `sales_data` and `returns_data` along the columns, ensuring that the resulting DataFrame contains all data. Use the `keys` parameter to create a heirarchical column index to differentiate between sales and returns. Display the result.

[ ]:
```
pd.concat((sales_data, returns_data), axis=1, keys=("Sales", "Returns"))
```

[ ]:
```
        Sales               Returns
    Product_A Product_B Product_A Product_B
Jan       100        80         5         2
```

```
Feb       150       90        7         3
Mar       200      120        6         4
```

1.2. Concatenate `sales_data` and `additional_sales` along the columns, ensuring that the resulting DataFrame contains all months and products, including months where data might be missing. Display the result.

```
[ ]: pd.concat((sales_data, additional_sales), axis=1)
```

```
[ ]:      Product_A  Product_B  Product_C
     Jan      100.0       80.0        NaN
     Feb      150.0       90.0       50.0
     Mar      200.0      120.0       60.0
     Apr        NaN        NaN       70.0
```

1.3. If you were the data scientist working on a project with these three objects, what do you think is the best way to join `sales_data`, `returns_data`, and `additional_sales`? Specify the axis, join type, and if you would add any heirarchical indexes or suffixes. Then, implement your join or reference the question that has already completed the join if you think it is one of the joins we have completed.

- The best way to join the three dataframes is to first group the sales data and the additional sales together. I'd combine them by Column, since it creates a new column for `Product_C`. Then, I'd outer join the combined sales data with the returns data, using `sales` and `returns` as a hierarchy. Both joins were completed by 1.1 and 1.2. I'd also replace all NaN values with zero, implying that there's no sale or return data for that month or year.

```
[ ]: combined_sales = pd.concat((sales_data, additional_sales), axis=1)
     sales_and_returns = pd.concat((combined_sales, returns_data), axis=1,␣
      ↪keys=("Sales", "Returns"))
     sales_and_returns = sales_and_returns.fillna(0.0)
     sales_and_returns
```

```
[ ]:         Sales                          Returns
        Product_A Product_B Product_C Product_A Product_B
     Jan     100.0      80.0       0.0       5.0       2.0
     Feb     150.0      90.0      50.0       7.0       3.0
     Mar     200.0     120.0      60.0       6.0       4.0
     Apr       0.0       0.0      70.0       0.0       0.0
```

2. True or False: By default, the `pd.concat` function performs an inner join, taking the intersection of the input columns.
If the statement is True, just state True. If the statement is False, please provide the corrected statement.

False, it provides an outer join by default.

3. What are the three different types of joins implemented by the `pd.merge` function?

Answer: one-to-one, many-to-one, many-to-many

4. True or False: Many-to-one joins are joins in which one of the two key columns contains duplicate entries.
If the statement is True, just state True. If the statement is False, please provide the corrected statement.

[ ]:
```
True
```

[ ]:
```
True
```

5. You are given the following DataFrames and Series representing customer orders, customer details, and product information. Perform the following tasks using merge and join.

[ ]:
```
# Customer Orders
orders = pd.DataFrame({
    'OrderID': [1, 2, 3, 4],
    'CustomerID': [101, 102, 101, 103],
    'ProductID': ['P001', 'P002', 'P005', 'P001'],
    'Quantity': [5, 3, 2, 1],
    'OrderDate': ['2023-01-15', '2023-01-16', '2023-01-17', '2023-01-18']
})

# Customer Details
customers = pd.DataFrame({
    'CustomerID': [101, 102, 103, 104],
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston']
})

# Product Information
products = pd.DataFrame({
    'ProductID': ['P001', 'P002', 'P003', 'P004'],
    'ProductName': ['Laptop', 'Smartphone', 'Tablet', 'Monitor'],
    'Price': [1200, 800, 300, 400]
})

# Customer Ratings
ratings = pd.Series(
    [4.5, 4.0, 3.5, 5.0],
    index=[101, 102, 103, 104],
    name='Rating'
)
```

5.1. Merge the orders DataFrame with the customers DataFrame on the CustomerID column to get a combined DataFrame of orders with customer details.

[ ]:
```
pd.merge(orders, customers, on="CustomerID")
```

```
[ ]:    OrderID  CustomerID ProductID  Quantity   OrderDate     Name         City
     0        1         101      P001         5  2023-01-15    Alice     New York
     1        3         101      P005         2  2023-01-17    Alice     New York
     2        2         102      P002         3  2023-01-16      Bob  Los Angeles
     3        4         103      P001         1  2023-01-18  Charlie      Chicago
```

5.2. Merge the result of 5.1 with the products DataFrame on the ProductID column to include product details for each order.

```
[ ]: # storing 5.1 result
     df = pd.merge(orders, customers, on="CustomerID")
     pd.merge(df, products, on='ProductID')
```

```
[ ]:    OrderID  CustomerID ProductID  Quantity   OrderDate     Name         City  \
     0        1         101      P001         5  2023-01-15    Alice     New York
     1        4         103      P001         1  2023-01-18  Charlie      Chicago
     2        2         102      P002         3  2023-01-16      Bob  Los Angeles

       ProductName  Price
     0      Laptop   1200
     1      Laptop   1200
     2  Smartphone    800
```

5.3. Join the ratings Series with the customers DataFrame on the CustomerID to add the ratings to the customer details.

```
[ ]: customers.join(ratings, on="CustomerID")
```

```
[ ]:    CustomerID     Name         City  Rating
     0         101    Alice     New York     4.5
     1         102      Bob  Los Angeles     4.0
     2         103  Charlie      Chicago     3.5
     3         104    David      Houston     5.0
```

5.4. Perform a left join of the orders DataFrame with the products DataFrame to include product details for each order, ensuring that all orders are included even if product details are missing.

```
[ ]: pd.merge(orders, products, how="left")
```

```
[ ]:    OrderID  CustomerID ProductID  Quantity   OrderDate ProductName   Price
     0        1         101      P001         5  2023-01-15      Laptop  1200.0
     1        2         102      P002         3  2023-01-16  Smartphone   800.0
     2        3         101      P005         2  2023-01-17         NaN     NaN
     3        4         103      P001         1  2023-01-18      Laptop  1200.0
```

5.5. Merge the customers DataFrame with the orders DataFrame using an outer join to ensure that all customers and all orders are included, regardless of whether there is a matching entry in both DataFrames.

```
[ ]: pd.merge(customers, orders, how="outer")
```

```
[ ]:    CustomerID     Name         City  OrderID ProductID  Quantity   OrderDate
     0         101    Alice     New York      1.0      P001       5.0  2023-01-15
     1         101    Alice     New York      3.0      P005       2.0  2023-01-17
     2         102      Bob  Los Angeles      2.0      P002       3.0  2023-01-16
     3         103  Charlie      Chicago      4.0      P001       1.0  2023-01-18
     4         104    David      Houston      NaN       NaN       NaN         NaN
```