pandas

August 27, 2024

1 15-Minutes Pandas Exercise

1.1 Jupyter Notebook Setup

1. Import necessary libraries: `os`, `pandas`, and `matplotlib`

[18]:
```
import os
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

The last line `%matplotlib inline` is a magic command used in Jupyter notebooks. It tells Jupyter to display the plots created by matplotlib directly in the notebook, right below the code cell that produces them, rather than in a separate window. This makes it easier to visualize the results of your plotting commands inline with your code and text.

However, in a non-Jupyter environment, this command would not be necessary or applicable.

2. Check and set working directory

[19]:
```
os.getcwd()
```

[19]:
```
'/Users/YigitAydede/Library/CloudStorage/Dropbox/Documents/Courses/MBAN/NLPBootc
amp/PythonBC'
```

3. Change the current directory to the folder where you want to save the notebook.

[20]:
```
os.chdir('/Users/YigitAydede/Library/CloudStorage/Dropbox/Documents/Courses/
↪MBAN/NLPBootcamp/PythonBC')
os.getcwd()
```

[20]:
```
'/Users/YigitAydede/Library/CloudStorage/Dropbox/Documents/Courses/MBAN/NLPBootc
amp/PythonBC'
```

4. Verify the data file exists

[21]:
```
import os

file_path = 'sales_data.csv'
if os.path.exists(file_path):
    print(f"The file {file_path} exists.")
```

```
else:
    print(f"The file {file_path} does not exist.")
```

```
The file sales_data.csv exists.
```

5. Read the CSV file

[22]:
```
df = pd.read_csv(file_path)
```

6. Use `df.info()` to get an overview of the dataset, including column names, data types, and non-null counts.

[23]:
```
# Display basic information about the dataset
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Date      10000 non-null  object
 1   Product   10000 non-null  object
 2   Quantity  10000 non-null  int64
 3   Price     10000 non-null  float64
 4   Region    10000 non-null  object
dtypes: float64(1), int64(1), object(3)
memory usage: 390.8+ KB
None
```

The output is the result of calling the 'info()" method on a pandas DataFrame. This method gives a concise summary of the DataFrame, including:

1. Class Type: <class 'pandas.core.frame.DataFrame'> indicates that the object is a DataFrame.
2. Range Index: RangeIndex: 10000 entries, 0 to 9999 specifies the range of the index, showing that the DataFrame has 10,000 rows, indexed from 0 to 9999.
3. Data Columns: Data columns (total 5 columns): indicates there are 5 columns in the DataFrame.
4. Column Information:

- #: Sequential number of the columns.
- Column: Name of the column.
- Non-Null Count: Number of non-null (non-missing) entries in each column.
- Dtype: Data type of the entries in each column.

For this DataFrame:

- Date: 10,000 non-null entries, dtype object (typically used for strings).
- Product: 10,000 non-null entries, dtype object.
- Quantity: 10,000 non-null entries, dtype int64 (integer).
- Price: 10,000 non-null entries, dtype float64 (floating-point number).
- Region: 10,000 non-null entries, dtype object.

- Dtype Summary: dtypes: float64(1), int64(1), object(3) provides a count of the different data types present in the DataFrame.

Memory Usage: memory usage: 390.8+ KB indicates the approximate amount of memory used by the DataFrame.

None: This is the return value of the info() method, which is None as the method is used for its side effect (printing the summary) rather than returning a value.

In the context of pandas DataFrames:

`dtype`stands for "data type" and refers to the type of data stored in each column of a DataFrame. It is an attribute of pandas Series and DataFrames that describes the kind of elements contained within. Common data types (dtypes) in pandas include:

- int64: 64-bit integer
- float64: 64-bit floating-point number
- bool: Boolean (True/False)
- datetime64[ns]: Date and time
- timedelta[ns]: Difference between two datetime values
- category: Categorical data
- object: General-purpose data type for text or mixed types

object object is a general-purpose dtype in pandas. It is used to store text data (strings) or mixed types. When pandas encounters data that doesn't fit neatly into one of the more specific dtypes (like integers, floats, or booleans), it uses object as a fallback. For example:

Columns containing strings (e.g., names, addresses) are typically of dtype object. Columns with mixed data types (e.g., a mix of integers, floats, and strings) will also be of dtype object.

7. Use `df.head()` to show us the first few rows of the data.

[24]:
```
# Display the first few rows of the dataset
df.head()
```

[24]:

	Date	Product	Quantity	Price	Region
0	2022-03-31	Laptop	10	398.041660	South
1	2022-06-13	Tablet	9	118.846586	South
2	2022-10-06	Smartphone	8	484.396535	West
3	2022-03-04	Smartphone	10	765.314520	West
4	2022-12-02	Laptop	9	815.330139	East

8. Use `df.describe()` to provide a statistical summary of the numerical columns.

[25]:
```
df.describe()
```

[25]:

	Quantity	Price
count	10000.000000	10000.000000
mean	5.455000	549.258954
std	2.871794	259.564628
min	1.000000	100.039033
25%	3.000000	323.127504

```
50%         5.000000    547.014960
75%         8.000000    772.120157
max        10.000000    999.705039
```

9. Check for missing values using `df.isnull().sum()`.

[26]:
```
df.isnull().sum()
```

[26]:
```
Date        0
Product     0
Quantity    0
Price       0
Region      0
dtype: int64
```

And we can remove the missing values

[ ]:
```
df.dropna(inplace=True)
```

The method `df.dropna(inplace=True)` is used in pandas to remove missing values (NaNs) from the DataFrame df. Here's a detailed explanation of what it does:

This method is used to remove rows or columns that contain missing values (NaNs). inplace=True: This argument modifies the DataFrame in place. Instead of creating a new DataFrame with the missing values removed, it updates the existing DataFrame df.

How It Works When you call df.dropna(inplace=True), it will remove all rows that contain at least one missing value (NaN) if no additional parameters are specified. It also updates the DataFrame df directly without needing to reassign it to a new variable.

9. Calculate total sales by multiplying Quantity and Price.

[27]:
```
df['Total Sales'] = df['Quantity'] * df['Price']
df.describe()
```

[27]:
```
           Quantity         Price   Total Sales
count  10000.000000  10000.000000  10000.000000
mean       5.455000    549.258954   2986.553337
std        2.871794    259.564628   2227.921417
min        1.000000    100.039033    100.039033
25%        3.000000    323.127504   1142.239718
50%        5.000000    547.014960   2404.393648
75%        8.000000    772.120157   4359.697954
max       10.000000    999.705039   9995.667353
```

10. Report the average price and total sales for each product.

[28]:
```
"""
Calculates the mean price and total sales for each product in the DataFrame␣
↪`df`.
```

```
The resulting `product_stats` DataFrame contains the following columns:
- `Price`: the mean price for each product
- `Total Sales`: the total sales for each product
"""
product_stats = df.groupby('Product').agg({'Price': 'mean', 'Total Sales':␣
↪'sum'})
print(product_stats)
```

```
                Price   Total Sales
Product
Headphones  548.321028  5.778406e+06
Laptop      546.433465  5.982554e+06
Smartphone  546.308102  6.084072e+06
Smartwatch  551.088800  6.040751e+06
Tablet      554.176400  5.979750e+06
```

11. Sort the same table by the Total Sales column in ascending order. Save it as a CSV file.

[29]:
```
product_stats_sorted = product_stats.sort_values('Total Sales', ascending=True)
print(product_stats_sorted)
product_stats_sorted.to_csv('product_stats.csv')
```

```
                Price   Total Sales
Product
Headphones  548.321028  5.778406e+06
Tablet      554.176400  5.979750e+06
Laptop      546.433465  5.982554e+06
Smartwatch  551.088800  6.040751e+06
Smartphone  546.308102  6.084072e+06
```

12. Find the correlation between Average Price and Total Sales

[30]:
```
correlation = product_stats['Price'].corr(product_stats['Total Sales'])
print(f"The correlation between Average Price and Total Sales is:␣
↪{correlation}")
```

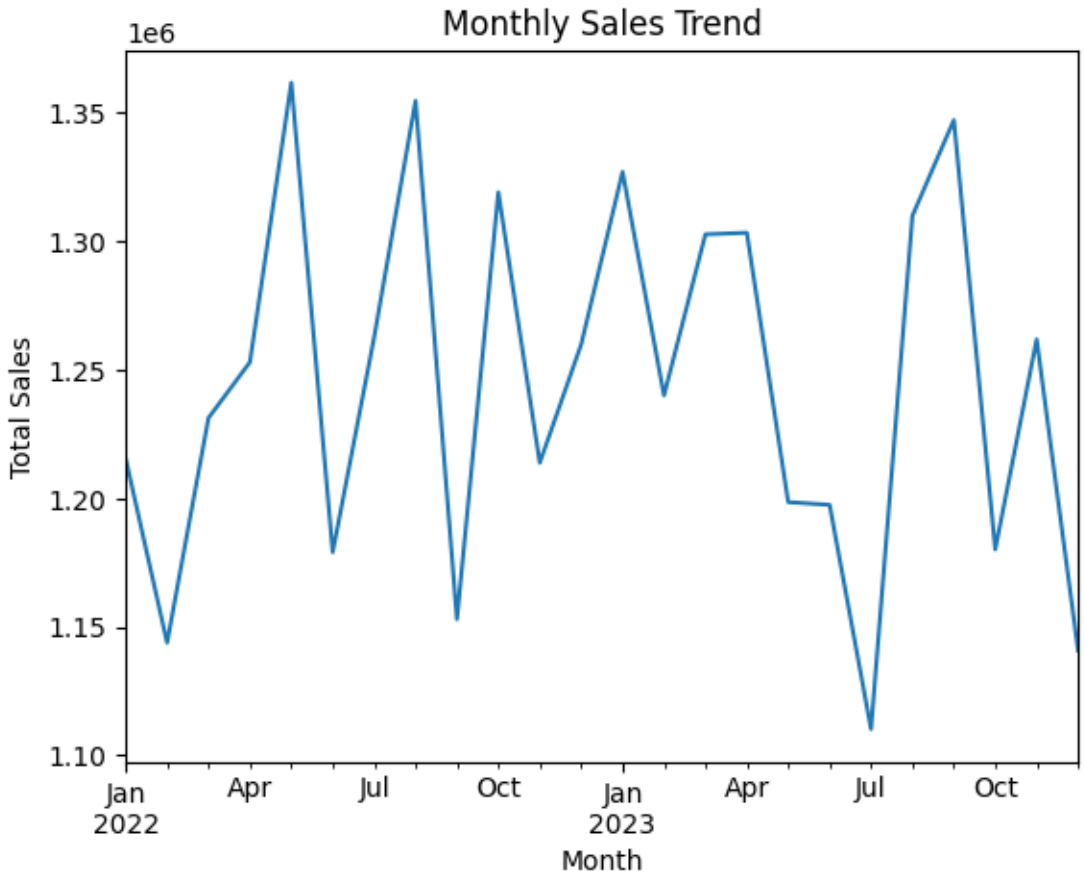
```
The correlation between Average Price and Total Sales is: -0.009575689060696368
```

13. Create a line plot to visualize the monthly sales trend.

[31]:
```
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)
monthly_sales = df.resample('M').sum()
monthly_sales['Total Sales'].plot(kind='line')
plt.xlabel('Month')
plt.ylabel('Total Sales')
plt.title('Monthly Sales Trend')
plt.show()
```

```
/var/folders/b2/gpnsjh9j6bv5prtx7w5lsym80000gp/T/ipykernel_80045/2285257092.py:3
: FutureWarning: 'M' is deprecated and will be removed in a future version,
```

```
please use 'ME' instead.
  monthly_sales = df.resample('M').sum()
```



- The first line converts the 'Date' column in the DataFrame df from a string format to a datetime format using the '`pd.to_datetime()` function. This conversion is essential for performing time series operations like resampling.
- The second line sets the 'Date' column as the index of the DataFrame. By doing this, you make the DataFrame suitable for time series operations, as the index will now represent the dates. `inplace=True` modifies the DataFrame in place, so you don't need to assign it back to df.
- The third line resamples the data to a monthly frequency using the `resample()` method. The `'M'` argument specifies that you want to resample to the end of each month. The `'mean'` argument specifies that you want to calculate the mean of the data within each month.
- The fourth line line creates a line plot of the 'Total Sales' column from the monthly_sales DataFrame. `kind='line'` specifies that the plot should be a line plot.

14. Calculate and display sales by region

[32]:
```
sales_by_region = df.groupby('Region')['Total Sales'].sum()
print(sales_by_region)
```

```
Region
Central    5.959897e+06
East       5.873895e+06
North      6.170071e+06
South      5.688002e+06
West       6.173668e+06
Name: Total Sales, dtype: float64
```

15. Find the day with highest sales

[33]:
```
highest_sales_day = df.groupby('Date')['Total Sales'].sum().idxmax()
print(f"The day with the highest sales is: {highest_sales_day}")
```

```
The day with the highest sales is: 2023-01-13 00:00:00
```

The `idxmax()` method returns the index of the first occurrence of the maximum value in the Series. In this case, it returns the date with the highest total sales.

16. Report the months when the maximum and minumum sales happens by product

The error you're encountering occurs because the Date column has been set as the index of the DataFrame. When you attempt to access df.loc[max_sales_month, ['Product', 'Date']], it cannot find the Date column because it's now an index, not a regular column.

To resolve this, you can reset the index before attempting to access the columns. Here's how you can do it:

[34]:
```
import pandas as pd

# Load the data from a CSV file
df = pd.read_csv(file_path)

# Calculate 'Total Sales' as the product of 'Quantity' and 'Price'
df['Total Sales'] = df['Quantity'] * df['Price']

# Ensure 'Date' column is in datetime format
df['Date'] = pd.to_datetime(df['Date'])

# Set 'Date' column as the index for time series operations
df.set_index('Date', inplace=True)

# Find the index of the row with maximum sales for each product
max_sales_idx = df.groupby('Product')['Total Sales'].idxmax()
# Get the corresponding rows and reset the index to turn 'Date' back into a␣
 ↪column
max_sales_month_df = df.loc[max_sales_idx].reset_index()[['Product', 'Date']]
# Rename the 'Date' column for clarity
max_sales_month_df.rename(columns={'Date': 'Month with Maximum Sales'},␣
 ↪inplace=True)
```

```
# Find the index of the row with minimum sales for each product
min_sales_idx = df.groupby('Product')['Total Sales'].idxmin()
# Get the corresponding rows and reset the index to turn 'Date' back into a␣
 ↪column
min_sales_month_df = df.loc[min_sales_idx].reset_index()[['Product', 'Date']]
# Rename the 'Date' column for clarity
min_sales_month_df.rename(columns={'Date': 'Month with Minimum Sales'},␣
 ↪inplace=True)

# Merge the two DataFrames on the 'Product' column
sales_month = pd.merge(max_sales_month_df, min_sales_month_df, on='Product')

# Display the result
print(sales_month)
```

```
        Product Month with Maximum Sales Month with Minimum Sales
0    Headphones               2023-03-11               2023-06-30
1    Headphones               2023-03-11               2023-01-07
2    Headphones               2023-03-11               2023-01-07
3    Headphones               2023-03-11               2023-01-07
4    Headphones               2023-03-11               2022-02-17
..          …                        …                        …
696  Smartphone               2023-09-17               2022-02-17
697  Smartphone               2023-09-17               2022-02-17
698  Smartphone               2023-09-17               2023-02-11
699  Smartphone               2023-09-17               2023-02-11
700  Smartphone               2023-09-17               2023-02-11

[701 rows x 3 columns]
```