

Data manipulation in Pandas

April 2, 2025

1 Data manipulation with Pandas:

In this chapter, you will learn the fundamentals of data manipulation with Pandas, a powerful Python library for working with structured data. You will explore how to :

- Inspect and analyze datasets using Pandas' DataFrame structure
- Perform operations like filtering, grouping, and aggregating data
- Visualize data directly from Pandas

Pandas is one of the most widely used libraries in Python for data manipulation and analysis. It provides powerful and flexible tools to work with structured data, especially in tabular form, which makes it an essential tool in the data science workflow.

Pandas is built on top of both NumPy and Matplotlib, which enhances its functionality for **Data manipulation** and **visualization**

- **NumPy integration:** Pandas uses NumPy arrays under the hood to handle the data in its main data structures, the Series and DataFrame. This integration allows Pandas to perform efficient numerical operations, such as vectorized computations, which significantly improve performance when working with large datasets. Many of the operations in Pandas, like slicing, indexing, and basic math, rely on NumPy's fast array operations.
- **Matplotlib integration:** Pandas also builds on Matplotlib for data visualization. Through Pandas, you can easily generate visualizations directly from a DataFrame or Series without needing to use Matplotlib's functions explicitly. For instance, you can plot data quickly using the `plot()` method in Pandas, which internally utilizes Matplotlib to render the charts.

By leveraging NumPy for performance and Matplotlib for visualizations, Pandas provides a powerful and efficient toolset for working with structured data.

1.1 Inspecting a DataFrame:

In the previous chapter, you learned the basics of DataFrame and how to load it from csv file. After the data is loaded into in a Pandas DataFrame, the first step in understanding it is to inspect it. Below are some essential functions in Pandas that help in inspecting a DataFrame, let's first load *supermarket sales* data set that we will use in this chapter:

```
[13]: import pandas as pd

df = pd.read_csv("https://drive.google.com/uc?
↳export=download&id=1B4w7L2eAehzIN-l4pZ4dD7Evp0cg5Lq8")
```

1.1.1 df.head():

Displays the first 5 rows of the DataFrame. This is useful to get a quick look at the structure of the data

```
[16]: df.head()
```

```
[16]: Invoice ID Branch      City Customer type Gender \
0  750-67-8428      A      Yangon      Member  Female
1  226-31-3081      C  Naypyitaw      Normal  Female
2  631-41-3108      A      Yangon      Normal   Male
3  123-19-1176      A      Yangon      Member   Male
4  373-73-7910      A      Yangon      Normal   Male

      Product line  Unit price  Quantity  Tax 5%  Total  Date \
0  Health and beauty      74.69         7  26.1415  548.9715  1/5/2019
1  Electronic accessories      15.28         5   3.8200   80.2200  3/8/2019
2  Home and lifestyle      46.33         7  16.2155  340.5255  3/3/2019
3  Health and beauty      58.22         8  23.2880  489.0480  1/27/2019
4  Sports and travel      86.31         7  30.2085  634.3785  2/8/2019

      Time      Payment  cogs  gross margin percentage  gross income  Rating
0  13:08      Ewallet  522.83          4.761905         26.1415      9.1
1  10:29      Cash    76.40          4.761905          3.8200      9.6
2  13:23  Credit card  324.31          4.761905         16.2155      7.4
3  20:33      Ewallet  465.76          4.761905         23.2880      8.4
4  10:37      Ewallet  604.17          4.761905         30.2085      5.3
```

You can adjust the number of rows displayed by the `head` function by passing a specific number as an argument in the function call.

1.1.2 df.tail():

Displays the last 5 rows of the DataFrame. This helps you see data at the bottom of the table.

```
[20]: df.tail()
```

```
[20]: Invoice ID Branch      City Customer type Gender      Product line \
995  233-67-5758      C  Naypyitaw      Normal   Male      Health and beauty
996  303-96-2227      B  Mandalay      Normal  Female      Home and lifestyle
997  727-02-1313      A      Yangon      Member   Male      Food and beverages
```

998	347-56-2442	A	Yangon	Normal	Male	Home and lifestyle
999	849-09-3807	A	Yangon	Member	Female	Fashion accessories

	Unit price	Quantity	Tax 5%	Total	Date	Time	Payment \
995	40.35	1	2.0175	42.3675	1/29/2019	13:46	Ewallet
996	97.38	10	48.6900	1022.4900	3/2/2019	17:16	Ewallet
997	31.84	1	1.5920	33.4320	2/9/2019	13:22	Cash
998	65.82	1	3.2910	69.1110	2/22/2019	15:33	Cash
999	88.34	7	30.9190	649.2990	2/18/2019	13:28	Cash

	cogs	gross margin percentage	gross income	Rating
995	40.35	4.761905	2.0175	6.2
996	973.80	4.761905	48.6900	4.4
997	31.84	4.761905	1.5920	7.7
998	65.82	4.761905	3.2910	4.1
999	618.38	4.761905	30.9190	6.6

1.1.3 df.shape:

Returns the shape of the DataFrame, i.e., the number of rows and columns.

```
[23]: df.shape
```

```
[23]: (1000, 17)
```

1.1.4 df.info():

Provides a concise summary of the DataFrame, including the data types of each column and the number of non-null entries.

```
[26]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 17 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Invoice ID             1000 non-null   object
1   Branch                1000 non-null   object
2   City                  1000 non-null   object
3   Customer type         1000 non-null   object
4   Gender                1000 non-null   object
5   Product line          1000 non-null   object
6   Unit price            1000 non-null   float64
7   Quantity              1000 non-null   int64
8   Tax 5%                1000 non-null   float64
```

```

9   Total                1000 non-null    float64
10  Date                 1000 non-null    object
11  Time                 1000 non-null    object
12  Payment              1000 non-null    object
13  cogs                 1000 non-null    float64
14  gross margin percentage 1000 non-null    float64
15  gross income         1000 non-null    float64
16  Rating               1000 non-null    float64
dtypes: float64(7), int64(1), object(9)
memory usage: 132.9+ KB

```

1.1.5 df.describe():

Generates descriptive statistics for numeric columns, including count, mean, min, max, and percentiles (25%, 50%, 75%).

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

```

[29]:      Unit price    Quantity    Tax 5%    Total    cogs \
count  1000.000000  1000.000000  1000.000000  1000.000000  1000.000000
mean    55.672130    5.510000    15.379369    322.966749    307.58738
std     26.494628    2.923431    11.708825    245.885335    234.17651
min     10.080000    1.000000     0.508500     10.678500     10.17000
25%     32.875000    3.000000     5.924875    124.422375    118.49750
50%     55.230000    5.000000    12.088000    253.848000    241.76000
75%     77.935000    8.000000    22.445250    471.350250    448.90500
max     99.960000   10.000000    49.650000   1042.650000    993.00000

      gross margin percentage  gross income    Rating
count                1.000000e+03  1000.000000  1000.000000
mean                  4.761905e+00    15.379369    6.97270
std                   6.131498e-14    11.708825    1.71858
min                   4.761905e+00     0.508500    4.00000
25%                   4.761905e+00     5.924875    5.50000
50%                   4.761905e+00    12.088000    7.00000
75%                   4.761905e+00    22.445250    8.50000
max                   4.761905e+00    49.650000   10.00000

```

1.1.6 df.index:

Returns the index of the dataframe.

```
[32]: df.index
```

```
[32]: RangeIndex(start=0, stop=1000, step=1)
```

1.1.7 df.columns:

Returns the list of column names in the DataFrame, helping you identify all available fields.

```
[35]: df.columns
```

```
[35]: Index(['Invoice ID', 'Branch', 'City', 'Customer type', 'Gender',  
         'Product line', 'Unit price', 'Quantity', 'Tax 5%', 'Total', 'Date',  
         'Time', 'Payment', 'cogs', 'gross margin percentage', 'gross income',  
         'Rating'],  
         dtype='object')
```

1.1.8 df.dtypes:

Returns the data types of each column. This is useful to know what kind of operations can be applied to each column (e.g., numeric operations for floats or integers).

```
[38]: df.dtypes
```

```
[38]: Invoice ID           object  
      Branch           object  
      City             object  
      Customer type    object  
      Gender           object  
      Product line     object  
      Unit price       float64  
      Quantity         int64  
      Tax 5%           float64  
      Total            float64  
      Date             object  
      Time             object  
      Payment          object  
      cogs             float64  
      gross margin percentage float64  
      gross income     float64  
      Rating           float64  
      dtype: object
```

1.2 Sorting a DataFrame :

you can sort a DataFrame by one or more columns using the `sort_values()` function. This function allows you to sort the data in either ascending or descending order, making it easier to organize and analyze your data.

To sort by a single column, specify the column name as an argument to the `sort_values()` function. By default, it sorts in ascending order.

```
[41]: # Sort the DataFrame by the 'Total' column in ascending order
df.sort_values(by='Total').head(3)
```

```
[41]: Invoice ID Branch      City Customer type Gender      Product line \
822  784-21-9238      C  Naypyitaw      Member      Male      Sports and travel
629  308-39-1707      A    Yangon      Normal      Female     Fashion accessories
223  279-62-1445      C  Naypyitaw      Member      Female     Fashion accessories

      Unit price  Quantity  Tax 5%      Total      Date      Time      Payment \
822         10.17         1  0.5085  10.6785  2/7/2019  14:15      Cash
629         12.09         1  0.6045  12.6945  1/26/2019  18:19  Credit card
223         12.54         1  0.6270  13.1670  2/21/2019  12:38      Cash

      cogs  gross margin percentage  gross income  Rating
822  10.17                    4.761905         0.5085      5.9
629  12.09                    4.761905         0.6045      8.2
223  12.54                    4.761905         0.6270      8.2
```

```
[42]: # Sort the DataFrame by the 'Total' column in descending order
df.sort_values(by='Total', ascending=False)
```

```
[42]: Invoice ID Branch      City Customer type Gender      Product line \
350  860-79-0874      C  Naypyitaw      Member      Female     Fashion accessories
167  687-47-8271      A    Yangon      Normal      Male      Fashion accessories
557  283-26-5248      C  Naypyitaw      Member      Female     Food and beverages
699  751-41-9720      C  Naypyitaw      Normal      Male      Home and lifestyle
996  303-96-2227      B  Mandalay      Normal      Female     Home and lifestyle
..      ...      ...      ...      ...      ...      ...
402  236-86-3015      C  Naypyitaw      Member      Male      Home and lifestyle
443  192-98-7397      C  Naypyitaw      Normal      Male      Fashion accessories
223  279-62-1445      C  Naypyitaw      Member      Female     Fashion accessories
629  308-39-1707      A    Yangon      Normal      Female     Fashion accessories
822  784-21-9238      C  Naypyitaw      Member      Male      Sports and travel

      Unit price  Quantity  Tax 5%      Total      Date      Time      Payment \
350         99.30         10  49.6500  1042.6500  2/15/2019  14:53  Credit card
167         98.98         10  49.4900  1039.2900  2/8/2019  16:20  Credit card
557         98.52         10  49.2600  1034.4600  1/30/2019  20:23      Ewallet
699         97.50         10  48.7500  1023.7500  1/12/2019  16:18      Ewallet
996         97.38         10  48.6900  1022.4900  3/2/2019  17:16      Ewallet
..      ...      ...      ...      ...      ...      ...      ...
402         13.98          1   0.6990   14.6790  2/4/2019  13:38      Ewallet
443         12.78          1   0.6390   13.4190  1/8/2019  14:11      Ewallet
223         12.54          1   0.6270   13.1670  2/21/2019  12:38      Cash
629         12.09          1   0.6045   12.6945  1/26/2019  18:19  Credit card
822         10.17          1   0.5085   10.6785  2/7/2019  14:15      Cash
```

	cogs	gross margin percentage	gross income	Rating
350	993.00	4.761905	49.6500	6.6
167	989.80	4.761905	49.4900	8.7
557	985.20	4.761905	49.2600	4.5
699	975.00	4.761905	48.7500	8.0
996	973.80	4.761905	48.6900	4.4
..
402	13.98	4.761905	0.6990	9.8
443	12.78	4.761905	0.6390	9.5
223	12.54	4.761905	0.6270	8.2
629	12.09	4.761905	0.6045	8.2
822	10.17	4.761905	0.5085	5.9

[1000 rows x 17 columns]

```
[43]: # Sort the DataFrame by the 'Rating' and 'Total' column in descending order
df.sort_values(by=['Rating', 'Total'], ascending=False).head(3)
```

```
[43]: Invoice ID Branch      City Customer type Gender \
159  423-57-2993      B Mandalay      Normal   Male
853  866-70-2814      B Mandalay      Normal  Female
62   347-34-2234      B Mandalay      Member  Female

      Product line  Unit price  Quantity  Tax 5%   Total \
159   Sports and travel      93.39         6  28.0170  588.3570
853  Electronic accessories      52.79        10  26.3950  554.2950
62    Sports and travel      55.07         9  24.7815  520.4115

      Date   Time  Payment   cogs  gross margin percentage  gross income \
159  3/27/2019  19:18  Ewallet  560.34          4.761905      28.0170
853  2/25/2019  11:58  Ewallet  527.90          4.761905      26.3950
62   2/3/2019  13:40  Ewallet  495.63          4.761905      24.7815

      Rating
159     10.0
853     10.0
62      10.0
```

You can use the ascending parameter to sort the DataFrame in **descending** order.

If you want to sort by multiple columns, pass a list of column names. You can also specify different sort orders for each column.

1.3 Summary statistics :

In Pandas, you can easily compute **summary statistics** for your DataFrame, which provides valuable insights into the **central tendency**, **dispersion**, and **shape of the data**. These statistics

include measures like mean, median, standard deviation, minimum, maximum, and more. Here's how you can use Pandas to calculate them:

1.3.1 1. Central tendency measures :

Tendency measures describe the central point of a dataset. The most common ones include the **mean** (average of the data), **median** (middle value when data is sorted), and **mode** (most frequent value). These measures help identify where most data points lie.

Mean : The average of the values in a column.

```
[51]: df['Total'].mean()
```

```
[51]: 322.966749
```

Summary statistics can be calculated to more than one column at the same time:

```
[53]: # For more than one column
df[['Total', 'Quantity']].mean()
```

```
[53]: Total          322.966749
Quantity         5.510000
dtype: float64
```

Median : The middle value of the data.

```
[55]: df['Total'].median()
```

```
[55]: 253.848
```

Mode: the most common value(s), it is used for categorical variables where mean and median are not applicable

```
[57]: df['Branch'].mode()
```

```
[57]: 0    A
      Name: Branch, dtype: object
```

1.3.2 2. Measures of dispersion:

Measures of spread describe how much variability or dispersion exists in the data. Key measures include the **range** (difference between the largest and smallest values), **variance** (average squared difference from the mean), and **standard deviation** (how much values deviate from the mean).

Minimum and maximum: The smallest and largest values.

```
[61]: print(df['Total'].min())  
      print(df['Total'].max())
```

```
10.6785  
1042.65
```

Standard deviation : The spread or variability of the data.

```
[63]: df['Total'].std()
```

```
[63]: 245.88533510097207
```

1.3.3 3. Measures of shape:

Measures of shape describe the shape of the distribution of the data and understanding where the values fall within the overall range of the dataset.

Quantiles: Quantiles are used to divide your data into equal parts, providing insights into the distribution of the dataset. The most common quantiles are:

- **0.25 quantile (25th percentile):** 25% of the data lies below this value.
- **0.50 quantile (50th percentile):** 50% of the data lies below this value. The **median** value.
- **0.75 quantile (75th percentile):** 75% of the data lies below this value.

```
[67]: df['Total'].quantile(0.25)
```

```
[67]: 124.422375
```

The 0.25 quantile indicates that 25% of the invoices have a total amount of 124.4 or less.

```
[69]: df['Total'].quantile(0.75)
```

```
[69]: 471.35024999999996
```

The 0.75 quantile indicates that 75% of the invoices have a total amount of 471.4 or less.

```
[71]: df['Total'].quantile(0.30)
```

```
[71]: 147.45465
```

The 0.30 quantile indicates that 30% of the invoices have a total amount of 147.5 or less.

1.3.4 3. Other measures:

Sum: The sum of the values.

```
[75]: df['Quantity'].sum()
```

```
[75]: 5510
```

Cumulative sum: cumulative sum of values where each value is the sum of all previous values up to that point.

```
[77]: df['Quantity'].cumsum()
```

```
[77]: 0          7
      1         12
      2         19
      3         27
      4         34
      ...
      995      5491
      996      5501
      997      5502
      998      5503
      999      5510
      Name: Quantity, Length: 1000, dtype: int64
```

Count: The number of non-missing values.

```
[79]: df['City'].count()
```

```
[79]: 1000
```

1.3.5 Describing categorical data:

- Categorical data represents **distinct** groups or categories. Its values are often labels or names (e.g., “Red,” “Blue,” “Yes,” “No”).
- While non-categorical data is analyzed using statistical measures like mean and variance, categorical data is often analyzed using **counts** and **proportions**

For categorical data, you can find the number of unique values and their frequency:

Unique values: Returns the distinct values in a column.

```
[84]: df['Product line'].unique()
```

```
[84]: array(['Health and beauty', 'Electronic accessories',  
        'Home and lifestyle', 'Sports and travel', 'Food and beverages',  
        'Fashion accessories'], dtype=object)
```

Number of unique values: Returns the number of distinct values in a column:

```
[86]: df['Product line'].nunique()
```

```
[86]: 6
```

Value counts: Shows the frequency of each value:

```
[88]: df['Product line'].value_counts()
```

```
[88]: Product line  
Fashion accessories      178  
Food and beverages      174  
Electronic accessories  170  
Sports and travel       166  
Home and lifestyle      160  
Health and beauty       152  
Name: count, dtype: int64
```

count() vs. nunique() The `count()` returns the number of non-null entries in a series or a dataframe (**including duplicates**), while `nunique()` returns the number of unique values (**excluding duplicates**).

drop_duplicates(): Another way to count unique values in a DataFrame or Series is by using `drop_duplicates()` function. This function is used to **remove duplicate rows** from a DataFrame or Series, retaining only the first occurrence of each unique row.

```
[93]: df['Product line'].drop_duplicates()
```

```
[93]: 0      Health and beauty  
1      Electronic accessories  
2      Home and lifestyle  
4      Sports and travel  
9      Food and beverages  
10     Fashion accessories  
Name: Product line, dtype: object
```

```
[94]: len(df['Product line'].drop_duplicates())
```

```
[94]: 6
```

We used `drop_duplicates()` to create a Series containing only unique values. Then we used `len()` to count the number of entries in the resulting object.

The `drop_duplicates()` function can be utilized to remove duplicates based on specific **combinations of columns** or even **across the entire DataFrame**. For instance, the code below removes rows that share identical values across all columns in the DataFrame:

```
[97]: df.drop_duplicates()
```

```
[97]:
```

	Invoice ID	Branch	City	Customer type	Gender	\
0	750-67-8428	A	Yangon	Member	Female	
1	226-31-3081	C	Naypyitaw	Normal	Female	
2	631-41-3108	A	Yangon	Normal	Male	
3	123-19-1176	A	Yangon	Member	Male	
4	373-73-7910	A	Yangon	Normal	Male	
...	
995	233-67-5758	C	Naypyitaw	Normal	Male	
996	303-96-2227	B	Mandalay	Normal	Female	
997	727-02-1313	A	Yangon	Member	Male	
998	347-56-2442	A	Yangon	Normal	Male	
999	849-09-3807	A	Yangon	Member	Female	

	Product line	Unit price	Quantity	Tax 5%	Total	\
0	Health and beauty	74.69	7	26.1415	548.9715	
1	Electronic accessories	15.28	5	3.8200	80.2200	
2	Home and lifestyle	46.33	7	16.2155	340.5255	
3	Health and beauty	58.22	8	23.2880	489.0480	
4	Sports and travel	86.31	7	30.2085	634.3785	
...	
995	Health and beauty	40.35	1	2.0175	42.3675	
996	Home and lifestyle	97.38	10	48.6900	1022.4900	
997	Food and beverages	31.84	1	1.5920	33.4320	
998	Home and lifestyle	65.82	1	3.2910	69.1110	
999	Fashion accessories	88.34	7	30.9190	649.2990	

	Date	Time	Payment	cogs	gross margin percentage	\
0	1/5/2019	13:08	Ewallet	522.83	4.761905	
1	3/8/2019	10:29	Cash	76.40	4.761905	
2	3/3/2019	13:23	Credit card	324.31	4.761905	
3	1/27/2019	20:33	Ewallet	465.76	4.761905	
4	2/8/2019	10:37	Ewallet	604.17	4.761905	
...	
995	1/29/2019	13:46	Ewallet	40.35	4.761905	
996	3/2/2019	17:16	Ewallet	973.80	4.761905	
997	2/9/2019	13:22	Cash	31.84	4.761905	
998	2/22/2019	15:33	Cash	65.82	4.761905	
999	2/18/2019	13:28	Cash	618.38	4.761905	

	gross income	Rating
0	26.1415	9.1
1	3.8200	9.6
2	16.2155	7.4
3	23.2880	8.4
4	30.2085	5.3
..
995	2.0175	6.2
996	48.6900	4.4
997	1.5920	7.7
998	3.2910	4.1
999	30.9190	6.6

[1000 rows x 17 columns]

The code below remove the duplicates based on **Branch** and **Product line**. The result contains unique combinations of **Branch** and **Product line**:

```
[99]: df.drop_duplicates(['Branch', 'Product line'])
```

```
[99]:
```

	Invoice ID	Branch	City	Customer type	Gender	\
0	750-67-8428	A	Yangon	Member	Female	
1	226-31-3081	C	Naypyitaw	Normal	Female	
2	631-41-3108	A	Yangon	Normal	Male	
4	373-73-7910	A	Yangon	Normal	Male	
6	355-53-5943	A	Yangon	Member	Female	
7	315-22-5665	C	Naypyitaw	Normal	Female	
9	692-92-5582	B	Mandalay	Member	Female	
10	351-62-0822	B	Mandalay	Member	Female	
11	529-56-3974	B	Mandalay	Member	Male	
13	252-56-2699	A	Yangon	Normal	Male	
15	299-46-1805	B	Mandalay	Member	Female	
19	319-50-3348	B	Mandalay	Normal	Female	
21	371-85-5789	B	Mandalay	Normal	Male	
27	189-17-4241	A	Yangon	Normal	Female	
34	183-56-6882	C	Naypyitaw	Member	Female	
35	232-16-2483	C	Naypyitaw	Member	Female	
38	333-73-7901	C	Naypyitaw	Normal	Female	
49	574-22-5561	C	Naypyitaw	Member	Female	

	Product line	Unit price	Quantity	Tax 5%	Total	\
0	Health and beauty	74.69	7	26.1415	548.9715	
1	Electronic accessories	15.28	5	3.8200	80.2200	
2	Home and lifestyle	46.33	7	16.2155	340.5255	
4	Sports and travel	86.31	7	30.2085	634.3785	
6	Electronic accessories	68.84	6	20.6520	433.6920	
7	Home and lifestyle	73.56	10	36.7800	772.3800	

9	Food and beverages	54.84	3	8.2260	172.7460
10	Fashion accessories	14.48	4	2.8960	60.8160
11	Electronic accessories	25.51	4	5.1020	107.1420
13	Food and beverages	43.19	10	21.5950	453.4950
15	Sports and travel	93.72	6	28.1160	590.4360
19	Home and lifestyle	40.30	2	4.0300	84.6300
21	Health and beauty	87.98	3	13.1970	277.1370
27	Fashion accessories	87.67	2	8.7670	184.1070
34	Food and beverages	99.42	4	19.8840	417.5640
35	Sports and travel	68.12	1	3.4060	71.5260
38	Health and beauty	54.92	8	21.9680	461.3280
49	Fashion accessories	82.63	10	41.3150	867.6150

	Date	Time	Payment	cogs	gross margin percentage \
0	1/5/2019	13:08	Ewallet	522.83	4.761905
1	3/8/2019	10:29	Cash	76.40	4.761905
2	3/3/2019	13:23	Credit card	324.31	4.761905
4	2/8/2019	10:37	Ewallet	604.17	4.761905
6	2/25/2019	14:36	Ewallet	413.04	4.761905
7	2/24/2019	11:38	Ewallet	735.60	4.761905
9	2/20/2019	13:27	Credit card	164.52	4.761905
10	2/6/2019	18:07	Ewallet	57.92	4.761905
11	3/9/2019	17:03	Cash	102.04	4.761905
13	2/7/2019	16:48	Ewallet	431.90	4.761905
15	1/15/2019	16:19	Cash	562.32	4.761905
19	3/11/2019	15:30	Ewallet	80.60	4.761905
21	3/5/2019	10:40	Ewallet	263.94	4.761905
27	3/10/2019	12:17	Credit card	175.34	4.761905
34	2/6/2019	10:42	Ewallet	397.68	4.761905
35	1/7/2019	12:28	Ewallet	68.12	4.761905
38	3/23/2019	13:24	Ewallet	439.36	4.761905
49	3/19/2019	17:08	Ewallet	826.30	4.761905

	gross income	Rating
0	26.1415	9.1
1	3.8200	9.6
2	16.2155	7.4
4	30.2085	5.3
6	20.6520	5.8
7	36.7800	8.0
9	8.2260	5.9
10	2.8960	4.5
11	5.1020	6.8
13	21.5950	8.2
15	28.1160	4.5
19	4.0300	4.4
21	13.1970	5.1

27	8.7670	7.7
34	19.8840	7.5
35	3.4060	6.8
38	21.9680	7.6
49	41.3150	7.9

This function is useful for **cleaning** up datasets by ensuring that each entry is unique, which can be essential for data **analysis** and processing. which helps in removing duplicates from the series or the entire dataframe

1.3.6 The agg() function:

The `agg()` (aggregate) function allows you to apply **multiple aggregation** operations simultaneously on your DataFrame or Series. It can be used to compute summary statistics like sum, mean, count, etc., on columns and combine multiple aggregations in a single step.

You can apply the `agg()` function to one columns and specify one or more aggregation functions:

```
[103]: df['Total'].agg(['mean', 'std'])
```

```
[103]: mean    322.966749
      std     245.885335
      Name: Total, dtype: float64
```

Or you can apply it to more than one column:

```
[105]: df[['Total', 'Quantity']].agg(['mean', 'std'])
```

```
[105]:          Total  Quantity
mean  322.966749  5.510000
std   245.885335  2.923431
```

You can also use `agg()` to apply different aggregation functions to different columns of your DataFrame.

```
[107]: df.agg({
      'Total': ['sum', 'mean'],
      'Quantity': ['sum']
    })
```

```
[107]:          Total  Quantity
sum  322966.749000  5510.0
mean   322.966749     NaN
```

1.3.7 The describe() function:

The `describe()` method generates summary statistics for **numeric** columns in the DataFrame. It provides statistics such as **count**, **mean**, **standard deviation**, **minimum**, and **maximum** values, as well as the **25th**, **50th (median)**, and **75th** percentiles.

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

```
[110]:
```

	Unit price	Quantity	Tax 5%	Total	cogs \
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000
mean	55.672130	5.510000	15.379369	322.966749	307.58738
std	26.494628	2.923431	11.708825	245.885335	234.17651
min	10.080000	1.000000	0.508500	10.678500	10.17000
25%	32.875000	3.000000	5.924875	124.422375	118.49750
50%	55.230000	5.000000	12.088000	253.848000	241.76000
75%	77.935000	8.000000	22.445250	471.350250	448.90500
max	99.960000	10.000000	49.650000	1042.650000	993.00000

	gross margin percentage	gross income	Rating
count	1.000000e+03	1000.000000	1000.000000
mean	4.761905e+00	15.379369	6.97270
std	6.131498e-14	11.708825	1.71858
min	4.761905e+00	0.508500	4.00000
25%	4.761905e+00	5.924875	5.50000
50%	4.761905e+00	12.088000	7.00000
75%	4.761905e+00	22.445250	8.50000
max	4.761905e+00	49.650000	10.00000

If your DataFrame contains both numeric and non-numeric data, you can specify `include='all'` to get statistics for all types of data.

```
[112]: # Get summary statistics for all columns, including non-numeric
df.describe(include='all')
```

```
[112]:
```

	Invoice ID	Branch	City	Customer type	Gender	Product line \
count	1000	1000	1000	1000	1000	1000
unique	1000	3	3	2	2	6
top	750-67-8428	A	Yangon	Member	Female	Fashion accessories
freq	1	340	340	501	501	178
mean	NaN	NaN	NaN	NaN	NaN	NaN
std	NaN	NaN	NaN	NaN	NaN	NaN
min	NaN	NaN	NaN	NaN	NaN	NaN
25%	NaN	NaN	NaN	NaN	NaN	NaN
50%	NaN	NaN	NaN	NaN	NaN	NaN
75%	NaN	NaN	NaN	NaN	NaN	NaN
max	NaN	NaN	NaN	NaN	NaN	NaN

	Unit price	Quantity	Tax 5%	Total	Date	Time \
--	------------	----------	--------	-------	------	--------

count	1000.000000	1000.000000	1000.000000	1000.000000	1000	1000
unique	NaN	NaN	NaN	NaN	89	506
top	NaN	NaN	NaN	NaN	2/7/2019	19:48
freq	NaN	NaN	NaN	NaN	20	7
mean	55.672130	5.510000	15.379369	322.966749	NaN	NaN
std	26.494628	2.923431	11.708825	245.885335	NaN	NaN
min	10.080000	1.000000	0.508500	10.678500	NaN	NaN
25%	32.875000	3.000000	5.924875	124.422375	NaN	NaN
50%	55.230000	5.000000	12.088000	253.848000	NaN	NaN
75%	77.935000	8.000000	22.445250	471.350250	NaN	NaN
max	99.960000	10.000000	49.650000	1042.650000	NaN	NaN

	Payment	cogs	gross margin percentage	gross income	Rating
count	1000	1000.000000	1.000000e+03	1000.000000	1000.000000
unique	3	NaN	NaN	NaN	NaN
top	Ewallet	NaN	NaN	NaN	NaN
freq	345	NaN	NaN	NaN	NaN
mean	NaN	307.58738	4.761905e+00	15.379369	6.97270
std	NaN	234.17651	6.131498e-14	11.708825	1.71858
min	NaN	10.17000	4.761905e+00	0.508500	4.00000
25%	NaN	118.49750	4.761905e+00	5.924875	5.50000
50%	NaN	241.76000	4.761905e+00	12.088000	7.00000
75%	NaN	448.90500	4.761905e+00	22.445250	8.50000
max	NaN	993.00000	4.761905e+00	49.650000	10.00000

1.4 Grouped summary statistics:

When analyzing datasets, summary statistics are often calculated for the entire dataset. However, comparing different groups within the data can reveal deeper insights. In Pandas, we can group data based on one or more columns and compute summary statistics for each group. This is particularly useful for understanding trends, differences, or patterns across categories.

```
[115]: df
```

```
[115]:
```

	Invoice ID	Branch	City	Customer type	Gender	\
0	750-67-8428	A	Yangon	Member	Female	
1	226-31-3081	C	Naypyitaw	Normal	Female	
2	631-41-3108	A	Yangon	Normal	Male	
3	123-19-1176	A	Yangon	Member	Male	
4	373-73-7910	A	Yangon	Normal	Male	
...	
995	233-67-5758	C	Naypyitaw	Normal	Male	
996	303-96-2227	B	Mandalay	Normal	Female	
997	727-02-1313	A	Yangon	Member	Male	
998	347-56-2442	A	Yangon	Normal	Male	
999	849-09-3807	A	Yangon	Member	Female	

	Product line	Unit price	Quantity	Tax 5%	Total \
0	Health and beauty	74.69	7	26.1415	548.9715
1	Electronic accessories	15.28	5	3.8200	80.2200
2	Home and lifestyle	46.33	7	16.2155	340.5255
3	Health and beauty	58.22	8	23.2880	489.0480
4	Sports and travel	86.31	7	30.2085	634.3785
..
995	Health and beauty	40.35	1	2.0175	42.3675
996	Home and lifestyle	97.38	10	48.6900	1022.4900
997	Food and beverages	31.84	1	1.5920	33.4320
998	Home and lifestyle	65.82	1	3.2910	69.1110
999	Fashion accessories	88.34	7	30.9190	649.2990

	Date	Time	Payment	cogs	gross margin percentage \
0	1/5/2019	13:08	Ewallet	522.83	4.761905
1	3/8/2019	10:29	Cash	76.40	4.761905
2	3/3/2019	13:23	Credit card	324.31	4.761905
3	1/27/2019	20:33	Ewallet	465.76	4.761905
4	2/8/2019	10:37	Ewallet	604.17	4.761905
..
995	1/29/2019	13:46	Ewallet	40.35	4.761905
996	3/2/2019	17:16	Ewallet	973.80	4.761905
997	2/9/2019	13:22	Cash	31.84	4.761905
998	2/22/2019	15:33	Cash	65.82	4.761905
999	2/18/2019	13:28	Cash	618.38	4.761905

	gross income	Rating
0	26.1415	9.1
1	3.8200	9.6
2	16.2155	7.4
3	23.2880	8.4
4	30.2085	5.3
..
995	2.0175	6.2
996	48.6900	4.4
997	1.5920	7.7
998	3.2910	4.1
999	30.9190	6.6

[1000 rows x 17 columns]

Calculate total sales for each Product line

Returning to our supermarket sales dataset, we previously learned how to calculate the total sales across all invoices. However, what if we want to calculate the total sales for each product line? This would allow us to compare the sales performance across different product lines and identify which ones generate higher or lower sales. To achieve this, we can apply filtering and use the `sum()`

function for each product line separately, as shown below:

```
[118]: print(df[df['Product line'] == 'Health and beauty']['Total'].sum())
print(df[df['Product line'] == 'Electronic accessories']['Total'].sum())
print(df[df['Product line'] == 'Home and lifestyle']['Total'].sum())
print(df[df['Product line'] == 'Sports and travel']['Total'].sum())
print(df[df['Product line'] == 'Food and beverages']['Total'].sum())
print(df[df['Product line'] == 'Fashion accessories']['Total'].sum())
```

```
49193.739
54337.531500000001
53861.913
55122.826500000001
56144.8440000000005
54305.8950000000004
```

The code above is not scalable and may become inconvenient when dealing with a large number of product lines. Additionally, it contains repetitive code, which increases the risk of errors. Instead, we can achieve the same result more efficiently using Pandas' `groupby()` function.

```
[120]: df.groupby('Product line')['Total'].sum()
```

```
[120]: Product line
Electronic accessories    54337.5315
Fashion accessories       54305.8950
Food and beverages        56144.8440
Health and beauty         49193.7390
Home and lifestyle        53861.9130
Sports and travel         55122.8265
Name: Total, dtype: float64
```

1.4.1 Grouping Data with `groupby()`

The `groupby()` function in Pandas allows you to **split your data into different groups** based on the values in one or more columns. You can **then calculate summary statistics for each group**.

`groupby()`: **split-apply-combine process**:

By `groupby()` we are referring to a process involving the following steps: 1. **Split**: Divide the DataFrame into groups based on the values in one or more columns. 2. **Apply**: After grouping, apply aggregation functions like `mean()`, `sum()`, `min()`, or custom function to calculate statistics for each group. 3. **Combine**: Combine the results of each group into a dataframe.

Find the average sales per gender:

Let's look at another example: finding the average sales per gender. In other words, we want to determine whether invoices tend to be higher for males or females. To do this, we can divide the data into two groups—male and female—using the `groupby()` function, and then calculate the mean for each group.

```
[128]: df.groupby('Gender')['Total'].mean()
```

```
[128]: Gender
      Female    335.095659
      Male     310.789226
      Name: Total, dtype: float64
```

From the above result, we can observe that the average invoice **total** is slightly higher for females compared to males.

1.4.2 Applying multiple aggregations with `agg()`

Similar to ungrouped summary statistics, we can use the `agg` method to obtain multiple statistics at once.

```
[131]: df.groupby('Product line')['Total'].agg(['mean', 'sum'])
```

```
[131]:
```

	mean	sum
Product line		
Electronic accessories	319.632538	54337.5315
Fashion accessories	305.089298	54305.8950
Food and beverages	322.671517	56144.8440
Health and beauty	323.643020	49193.7390
Home and lifestyle	336.636956	53861.9130
Sports and travel	332.065220	55122.8265

1.4.3 Group by multiple columns:

In addition to grouping by a single column, Pandas also allows you to group data by multiple columns. This can be useful when you want to analyze data based on more than one factor. For example, you might want to see the total sales grouped by both gender and product line to understand how sales differ across these categories.

```
[134]: df.groupby(['Product line', 'Branch'])['Total'].sum()
```

```
[134]:
```

Product line	Branch	
Electronic accessories	A	18317.1135
	B	17051.4435
	C	18968.9745
Fashion accessories	A	16332.5085
	B	16413.3165
	C	21560.0700
Food and beverages	A	17163.1005
	B	15214.8885
	C	23766.8550
Health and beauty	A	12597.7530

	B	19980.6600
	C	16615.3260
Home and lifestyle	A	22417.1955
	B	17549.1645
	C	13895.5530
Sports and travel	A	19372.6995
	B	19988.1990
	C	15761.9280

Name: Total, dtype: float64

1.4.4 Multiple groups and multiple summaries:

You can also group by multiple columns and apply aggregation functions to multiple columns simultaneously.

```
[137]: df.groupby(['Product line', 'Branch'])[['Total', 'gross income']].sum()
```

```
[137]:
```

Product line	Branch	Total	gross income
Electronic accessories	A	18317.1135	872.2435
	B	17051.4435	811.9735
	C	18968.9745	903.2845
Fashion accessories	A	16332.5085	777.7385
	B	16413.3165	781.5865
	C	21560.0700	1026.6700
Food and beverages	A	17163.1005	817.2905
	B	15214.8885	724.5185
	C	23766.8550	1131.7550
Health and beauty	A	12597.7530	599.8930
	B	19980.6600	951.4600
	C	16615.3260	791.2060
Home and lifestyle	A	22417.1955	1067.4855
	B	17549.1645	835.6745
	C	13895.5530	661.6930
Sports and travel	A	19372.6995	922.5095
	B	19988.1990	951.8190
	C	15761.9280	750.5680

1.4.5 Practice questions:

1. Who made more purchases, male or female?

To address this question, we need to compare the total purchases between males and females. To do so, we will split the data into two groups based on the **Gender** column and apply the `sum()` aggregation to each group. This can be done easily using the `groupby()` function as shown below:

```
[141]: df.groupby('Gender')['Total'].sum()
```

```
[141]: Gender
      Female    167882.925
      Male     155083.824
      Name: Total, dtype: float64
```

From the results above, we can see that females made slightly more purchases compared to males.

2. Do males and females have different preferences when it comes to payment methods?

```
[144]: df.groupby(['Payment', 'Gender'])['Invoice ID'].count()
```

```
[144]: Payment      Gender
      Cash         Female    178
              Male         166
      Credit card  Female    163
              Male         148
      Ewallet      Female    160
              Male         185
      Name: Invoice ID, dtype: int64
```

From this, we can conclude that while females tend to use cash and credit cards slightly more, males show a clear preference for e-wallets over females.

3. Calculate the profit (gross income) for each product line from each branch?

```
[147]: df.groupby(['Branch', 'Product line'])['gross income'].sum()
```

```
[147]: Branch  Product line
      A      Electronic accessories    872.2435
              Fashion accessories     777.7385
              Food and beverages      817.2905
              Health and beauty       599.8930
              Home and lifestyle     1067.4855
              Sports and travel       922.5095
      B      Electronic accessories    811.9735
              Fashion accessories     781.5865
              Food and beverages      724.5185
              Health and beauty       951.4600
              Home and lifestyle     835.6745
              Sports and travel       951.8190
      C      Electronic accessories    903.2845
              Fashion accessories    1026.6700
              Food and beverages     1131.7550
              Health and beauty       791.2060
              Home and lifestyle     661.6930
              Sports and travel       750.5680
      Name: gross income, dtype: float64
```

4. Find the latest 3 invoices in each branch?

```
[195]: df.groupby('Branch').apply(lambda x: x.sort_values(["Date", "Time"],
↪ascending=False).head(3))
```

C:\Users\asabb\AppData\Local\Temp\ipykernel_21444\3912989711.py:1:

DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns. This behavior is deprecated, and in a future version of pandas the grouping columns will be excluded from the operation. Either pass `include_groups=False` to exclude the groupings or explicitly select the grouping columns after groupby to silence this warning.

```
df.groupby('Branch').apply(lambda x: x.sort_values(["Date", "Time"],
ascending=False).head(3))
```

```
[195]:
```

	Invoice ID	Branch	City	Customer type	Gender	\
Branch						
A	326	815-11-1168	A	Yangon	Member	Male
	234	157-13-5295	A	Yangon	Member	Male
	585	127-47-6963	A	Yangon	Normal	Male
B	45	132-32-9879	B	Mandalay	Member	Female
	122	219-22-9386	B	Mandalay	Member	Male
	11	529-56-3974	B	Mandalay	Member	Male
C	73	841-35-6630	C	Naypyitaw	Normal	Female
	424	489-64-4354	C	Naypyitaw	Normal	Male
	611	277-35-5865	C	Naypyitaw	Member	Female

		Product line	Unit price	Quantity	Tax 5%	Total	\
Branch							
A	326	Food and beverages	99.78	5	24.9450	523.8450	
	234	Health and beauty	51.94	10	25.9700	545.3700	
	585	Health and beauty	51.71	4	10.3420	217.1820	
B	45	Electronic accessories	93.96	4	18.7920	394.6320	
	122	Sports and travel	99.96	9	44.9820	944.6220	
	11	Electronic accessories	25.51	4	5.1020	107.1420	
C	73	Electronic accessories	75.91	6	22.7730	478.2330	
	424	Fashion accessories	16.28	1	0.8140	17.0940	
	611	Food and beverages	98.97	9	44.5365	935.2665	

		Date	Time	Payment	cogs	gross margin percentage	\
Branch							
A	326	3/9/2019	19:09	Cash	498.90	4.761905	
	234	3/9/2019	18:24	Ewallet	519.40	4.761905	
	585	3/9/2019	13:53	Credit card	206.84	4.761905	
B	45	3/9/2019	18:00	Cash	375.84	4.761905	
	122	3/9/2019	17:26	Credit card	899.64	4.761905	
	11	3/9/2019	17:03	Cash	102.04	4.761905	
C	73	3/9/2019	18:21	Cash	455.46	4.761905	
	424	3/9/2019	15:36	Cash	16.28	4.761905	

	611	3/9/2019	11:23	Cash	890.73	4.761905
		gross income		Rating		
Branch						
A	326	24.9450	5.4			
	234	25.9700	6.5			
	585	10.3420	9.8			
B	45	18.7920	9.5			
	122	44.9820	4.2			
	11	5.1020	6.8			
C	73	22.7730	8.7			
	424	0.8140	5.0			
	611	44.5365	6.7			

1.5 Dataframe index:

1.5.1 Dataframe structure (Recap):

A Pandas DataFrame is a two-dimensional, labeled data structure that consists of **three** key components:

1. Row Index:

This is the label or index assigned to each row, which helps in identifying and accessing rows easily. It acts like a unique identifier for each row, similar to row numbers in a table. By default, Pandas assigns numeric row indices starting from 0, but you can customize the row index with other labels, such as strings or dates.

```
[48]: df.index
```

```
[48]: RangeIndex(start=0, stop=1000, step=1)
```

2. Column Index

Column Index: Similar to the row index, the column index labels each column, making it easy to reference columns by name. The column index typically represents the names of variables or features in your dataset. Each column label corresponds to a different type of data stored in the DataFrame.

```
[49]: df.columns
```

```
[49]: Index(['Invoice ID', 'Branch', 'City', 'Customer type', 'Gender',
          'Product line', 'Unit price', 'Quantity', 'Tax 5%', 'Total', 'Date',
          'Time', 'Payment', 'cogs', 'gross margin percentage', 'gross income',
          'Rating'],
          dtype='object')
```

3. Data (Numpy Array):

The actual data in the DataFrame is stored in a Numpy array format. This array is structured as a matrix where the rows represent individual data entries (or records) and the columns represent variables (or features). Pandas leverages Numpy's efficient array operations, making data manipulation and computation faster.

```
[4]: df.values
```

```
[4]: array([[ '750-67-8428', 'A', 'Yangon', ..., 4.761904762, 26.1415, 9.1],
          [ '226-31-3081', 'C', 'Naypyitaw', ..., 4.761904762, 3.82, 9.6],
          [ '631-41-3108', 'A', 'Yangon', ..., 4.761904762, 16.2155, 7.4],
          ...,
          [ '727-02-1313', 'A', 'Yangon', ..., 4.761904762, 1.592, 7.7],
          [ '347-56-2442', 'A', 'Yangon', ..., 4.761904762, 3.291, 4.1],
          [ '849-09-3807', 'A', 'Yangon', ..., 4.761904762, 30.919, 6.6]],
          dtype=object)
```

These three components together allow the Pandas DataFrame to efficiently handle, manipulate, and analyze large datasets

1.5.2 Moving column to index:

In Pandas, columns that are part of the main data in a DataFrame can be moved to become part of the DataFrame's index.

When you move a column to the index, you're changing how the DataFrame is structured and accessed. Instead of that column being part of the regular data (where each value is aligned with a row and a column), it now serves as a label for the rows, making it easier to locate and group data.

This can be done using the `set_index()` function in Pandas. Here's an example:

Example: move Invoice ID to be the index

```
[5]: df = df.set_index('Invoice ID')
df
```

```
[5]:
```

	Branch	City	Customer type	Gender	Product line \
Invoice ID					
750-67-8428	A	Yangon	Member	Female	Health and beauty
226-31-3081	C	Naypyitaw	Normal	Female	Electronic accessories
631-41-3108	A	Yangon	Normal	Male	Home and lifestyle
123-19-1176	A	Yangon	Member	Male	Health and beauty
373-73-7910	A	Yangon	Normal	Male	Sports and travel
...
233-67-5758	C	Naypyitaw	Normal	Male	Health and beauty
303-96-2227	B	Mandalay	Normal	Female	Home and lifestyle
727-02-1313	A	Yangon	Member	Male	Food and beverages
347-56-2442	A	Yangon	Normal	Male	Home and lifestyle
849-09-3807	A	Yangon	Member	Female	Fashion accessories

	Unit price	Quantity	Tax 5%	Total	Date	Time \
Invoice ID						
750-67-8428	74.69	7	26.1415	548.9715	1/5/2019	13:08
226-31-3081	15.28	5	3.8200	80.2200	3/8/2019	10:29
631-41-3108	46.33	7	16.2155	340.5255	3/3/2019	13:23
123-19-1176	58.22	8	23.2880	489.0480	1/27/2019	20:33
373-73-7910	86.31	7	30.2085	634.3785	2/8/2019	10:37
...
233-67-5758	40.35	1	2.0175	42.3675	1/29/2019	13:46
303-96-2227	97.38	10	48.6900	1022.4900	3/2/2019	17:16
727-02-1313	31.84	1	1.5920	33.4320	2/9/2019	13:22
347-56-2442	65.82	1	3.2910	69.1110	2/22/2019	15:33
849-09-3807	88.34	7	30.9190	649.2990	2/18/2019	13:28

	Payment	cogs	gross margin percentage	gross income \
Invoice ID				
750-67-8428	Ewallet	522.83	4.761905	26.1415
226-31-3081	Cash	76.40	4.761905	3.8200
631-41-3108	Credit card	324.31	4.761905	16.2155
123-19-1176	Ewallet	465.76	4.761905	23.2880
373-73-7910	Ewallet	604.17	4.761905	30.2085
...
233-67-5758	Ewallet	40.35	4.761905	2.0175
303-96-2227	Ewallet	973.80	4.761905	48.6900
727-02-1313	Cash	31.84	4.761905	1.5920
347-56-2442	Cash	65.82	4.761905	3.2910
849-09-3807	Cash	618.38	4.761905	30.9190

	Rating
Invoice ID	
750-67-8428	9.1
226-31-3081	9.6
631-41-3108	7.4
123-19-1176	8.4
373-73-7910	5.3
...	...
233-67-5758	6.2
303-96-2227	4.4
727-02-1313	7.7
347-56-2442	4.1
849-09-3807	6.6

[1000 rows x 16 columns]

Here, the Invoice ID column becomes the row index, and it's no longer part of the regular data. Instead, it acts as a label for each row, and can be used to reference or locate the data associated with each name more efficiently.

```
[6]: df.loc['303-96-2227']
```

```
[6]: Branch                B
      City                Mandalay
      Customer type        Normal
      Gender              Female
      Product line        Home and lifestyle
      Unit price          97.38
      Quantity            10
      Tax 5%              48.69
      Total               1022.49
      Date                3/2/2019
      Time                17:16
      Payment             Ewallet
      cogs                 973.8
      gross margin percentage 4.761905
      gross income         48.69
      Rating              4.4
      Name: 303-96-2227, dtype: object
```

1.5.3 Removing the index:

If you have a custom index in your DataFrame, and you want to reset or remove it to bring the data back into the main DataFrame (as a regular column), you can use the `reset_index()` function. This operation moves the index back into the body of the DataFrame as a normal column.

```
[7]: df = df.reset_index()
      df
```

```
[7]:
```

	Invoice ID	Branch	City	Customer type	Gender	\
0	750-67-8428	A	Yangon	Member	Female	
1	226-31-3081	C	Naypyitaw	Normal	Female	
2	631-41-3108	A	Yangon	Normal	Male	
3	123-19-1176	A	Yangon	Member	Male	
4	373-73-7910	A	Yangon	Normal	Male	
...	
995	233-67-5758	C	Naypyitaw	Normal	Male	
996	303-96-2227	B	Mandalay	Normal	Female	
997	727-02-1313	A	Yangon	Member	Male	
998	347-56-2442	A	Yangon	Normal	Male	
999	849-09-3807	A	Yangon	Member	Female	

	Product line	Unit price	Quantity	Tax 5%	Total	\
0	Health and beauty	74.69	7	26.1415	548.9715	
1	Electronic accessories	15.28	5	3.8200	80.2200	
2	Home and lifestyle	46.33	7	16.2155	340.5255	

3	Health and beauty	58.22	8	23.2880	489.0480
4	Sports and travel	86.31	7	30.2085	634.3785
..
995	Health and beauty	40.35	1	2.0175	42.3675
996	Home and lifestyle	97.38	10	48.6900	1022.4900
997	Food and beverages	31.84	1	1.5920	33.4320
998	Home and lifestyle	65.82	1	3.2910	69.1110
999	Fashion accessories	88.34	7	30.9190	649.2990

	Date	Time	Payment	cogs	gross margin percentage \
0	1/5/2019	13:08	Ewallet	522.83	4.761905
1	3/8/2019	10:29	Cash	76.40	4.761905
2	3/3/2019	13:23	Credit card	324.31	4.761905
3	1/27/2019	20:33	Ewallet	465.76	4.761905
4	2/8/2019	10:37	Ewallet	604.17	4.761905
..
995	1/29/2019	13:46	Ewallet	40.35	4.761905
996	3/2/2019	17:16	Ewallet	973.80	4.761905
997	2/9/2019	13:22	Cash	31.84	4.761905
998	2/22/2019	15:33	Cash	65.82	4.761905
999	2/18/2019	13:28	Cash	618.38	4.761905

	gross income	Rating
0	26.1415	9.1
1	3.8200	9.6
2	16.2155	7.4
3	23.2880	8.4
4	30.2085	5.3
..
995	2.0175	6.2
996	48.6900	4.4
997	1.5920	7.7
998	3.2910	4.1
999	30.9190	6.6

[1000 rows x 17 columns]

1.5.4 Drop index:

If you don't want the old index to be kept as a new column, you can use the `drop=True` option. This will remove the index entirely and not insert it as a new column:

```
[8]: df = df.set_index('Invoice ID')
      df.reset_index(drop=True)
```

```

[8]:      Branch      City Customer type Gender      Product line \
0         A      Yangon      Member Female      Health and beauty
1         C Naypyitaw      Normal Female Electronic accessories
2         A      Yangon      Normal  Male      Home and lifestyle
3         A      Yangon      Member  Male      Health and beauty
4         A      Yangon      Normal  Male      Sports and travel
..      ...      ...      ...      ...      ...
995      C Naypyitaw      Normal  Male      Health and beauty
996      B  Mandalay      Normal Female      Home and lifestyle
997      A      Yangon      Member  Male      Food and beverages
998      A      Yangon      Normal  Male      Home and lifestyle
999      A      Yangon      Member Female      Fashion accessories

      Unit price  Quantity  Tax 5%      Total      Date      Time      Payment \
0         74.69         7  26.1415  548.9715  1/5/2019  13:08      Ewallet
1         15.28         5   3.8200   80.2200  3/8/2019  10:29      Cash
2         46.33         7  16.2155  340.5255  3/3/2019  13:23      Credit card
3         58.22         8  23.2880  489.0480  1/27/2019  20:33      Ewallet
4         86.31         7  30.2085  634.3785  2/8/2019  10:37      Ewallet
..      ...      ...      ...      ...      ...      ...      ...
995      40.35         1   2.0175   42.3675  1/29/2019  13:46      Ewallet
996      97.38        10  48.6900 1022.4900  3/2/2019  17:16      Ewallet
997      31.84         1   1.5920   33.4320  2/9/2019  13:22      Cash
998      65.82         1   3.2910   69.1110  2/22/2019  15:33      Cash
999      88.34         7  30.9190  649.2990  2/18/2019  13:28      Cash

      cogs  gross margin percentage  gross income  Rating
0     522.83                4.761905      26.1415      9.1
1      76.40                4.761905       3.8200      9.6
2     324.31                4.761905      16.2155      7.4
3     465.76                4.761905      23.2880      8.4
4     604.17                4.761905      30.2085      5.3
..      ...                ...      ...      ...
995    40.35                4.761905       2.0175      6.2
996   973.80                4.761905      48.6900      4.4
997    31.84                4.761905       1.5920      7.7
998    65.82                4.761905       3.2910      4.1
999   618.38                4.761905      30.9190      6.6

```

[1000 rows x 16 columns]

1.5.5 Filtering by index:

```
[9]: df = pd.read_csv("data/supermarket_sales.csv")
df = df.set_index(pd.to_datetime(df['Date']))
df
```

```
[9]:
```

	Invoice ID	Branch	City	Customer type	Gender	\
Date						
2019-01-05	750-67-8428	A	Yangon	Member	Female	
2019-03-08	226-31-3081	C	Naypyitaw	Normal	Female	
2019-03-03	631-41-3108	A	Yangon	Normal	Male	
2019-01-27	123-19-1176	A	Yangon	Member	Male	
2019-02-08	373-73-7910	A	Yangon	Normal	Male	
...	
2019-01-29	233-67-5758	C	Naypyitaw	Normal	Male	
2019-03-02	303-96-2227	B	Mandalay	Normal	Female	
2019-02-09	727-02-1313	A	Yangon	Member	Male	
2019-02-22	347-56-2442	A	Yangon	Normal	Male	
2019-02-18	849-09-3807	A	Yangon	Member	Female	

	Product line	Unit price	Quantity	Tax 5%	Total	\
Date						
2019-01-05	Health and beauty	74.69	7	26.1415	548.9715	
2019-03-08	Electronic accessories	15.28	5	3.8200	80.2200	
2019-03-03	Home and lifestyle	46.33	7	16.2155	340.5255	
2019-01-27	Health and beauty	58.22	8	23.2880	489.0480	
2019-02-08	Sports and travel	86.31	7	30.2085	634.3785	
...	
2019-01-29	Health and beauty	40.35	1	2.0175	42.3675	
2019-03-02	Home and lifestyle	97.38	10	48.6900	1022.4900	
2019-02-09	Food and beverages	31.84	1	1.5920	33.4320	
2019-02-22	Home and lifestyle	65.82	1	3.2910	69.1110	
2019-02-18	Fashion accessories	88.34	7	30.9190	649.2990	

	Date	Time	Payment	cogs	gross margin percentage	\
Date						
2019-01-05	1/5/2019	13:08	Ewallet	522.83	4.761905	
2019-03-08	3/8/2019	10:29	Cash	76.40	4.761905	
2019-03-03	3/3/2019	13:23	Credit card	324.31	4.761905	
2019-01-27	1/27/2019	20:33	Ewallet	465.76	4.761905	
2019-02-08	2/8/2019	10:37	Ewallet	604.17	4.761905	
...	
2019-01-29	1/29/2019	13:46	Ewallet	40.35	4.761905	
2019-03-02	3/2/2019	17:16	Ewallet	973.80	4.761905	
2019-02-09	2/9/2019	13:22	Cash	31.84	4.761905	
2019-02-22	2/22/2019	15:33	Cash	65.82	4.761905	
2019-02-18	2/18/2019	13:28	Cash	618.38	4.761905	

	gross income	Rating
Date		
2019-01-05	26.1415	9.1
2019-03-08	3.8200	9.6
2019-03-03	16.2155	7.4
2019-01-27	23.2880	8.4
2019-02-08	30.2085	5.3
...
2019-01-29	2.0175	6.2
2019-03-02	48.6900	4.4
2019-02-09	1.5920	7.7
2019-02-22	3.2910	4.1
2019-02-18	30.9190	6.6

[1000 rows x 17 columns]

In Pandas, the index of a DataFrame plays a powerful role in filtering and subsetting data. When you set a column (or multiple columns) as an index, you can easily access, filter, and subset your data using that index. This can be especially useful for tasks that involve selecting specific rows or sections of your dataset.

1.5.6 Single index filtering:

```
[10]: df.loc['2019-01-05']
```

```
[10]:
```

	Invoice ID	Branch	City	Customer type	Gender	\
Date						
2019-01-05	750-67-8428	A	Yangon	Member	Female	
2019-01-05	628-34-3388	C	Naypyitaw	Normal	Male	
2019-01-05	217-58-1179	A	Yangon	Member	Male	
2019-01-05	144-51-6085	A	Yangon	Member	Male	
2019-01-05	843-01-4703	B	Mandalay	Member	Female	
2019-01-05	573-98-8548	C	Naypyitaw	Member	Female	
2019-01-05	209-61-0206	A	Yangon	Normal	Female	
2019-01-05	339-12-4827	B	Mandalay	Member	Female	
2019-01-05	841-18-8232	B	Mandalay	Normal	Female	
2019-01-05	801-88-0346	C	Naypyitaw	Normal	Female	
2019-01-05	433-08-7822	C	Naypyitaw	Normal	Female	
2019-01-05	489-82-1237	A	Yangon	Normal	Female	

	Product line	Unit price	Quantity	Tax 5%	Total	\
Date						
2019-01-05	Health and beauty	74.69	7	26.1415	548.9715	
2019-01-05	Fashion accessories	27.38	6	8.2140	172.4940	
2019-01-05	Home and lifestyle	62.65	4	12.5300	263.1300	

2019-01-05	Home and lifestyle	70.74	4	14.1480	297.1080
2019-01-05	Home and lifestyle	35.38	9	15.9210	334.3410
2019-01-05	Fashion accessories	31.90	1	1.5950	33.4950
2019-01-05	Home and lifestyle	42.91	5	10.7275	225.2775
2019-01-05	Fashion accessories	73.96	1	3.6980	77.6580
2019-01-05	Food and beverages	71.20	1	3.5600	74.7600
2019-01-05	Fashion accessories	76.06	3	11.4090	239.5890
2019-01-05	Health and beauty	78.89	7	27.6115	579.8415
2019-01-05	Electronic accessories	93.88	7	32.8580	690.0180

	Date	Time	Payment	cogs	gross margin percentage \
Date					
2019-01-05	1/5/2019	13:08	Ewallet	522.83	4.761905
2019-01-05	1/5/2019	20:54	Credit card	164.28	4.761905
2019-01-05	1/5/2019	11:25	Cash	250.60	4.761905
2019-01-05	1/5/2019	16:05	Credit card	282.96	4.761905
2019-01-05	1/5/2019	19:50	Credit card	318.42	4.761905
2019-01-05	1/5/2019	12:40	Ewallet	31.90	4.761905
2019-01-05	1/5/2019	17:29	Ewallet	214.55	4.761905
2019-01-05	1/5/2019	11:32	Credit card	73.96	4.761905
2019-01-05	1/5/2019	20:40	Credit card	71.20	4.761905
2019-01-05	1/5/2019	20:30	Credit card	228.18	4.761905
2019-01-05	1/5/2019	19:48	Ewallet	552.23	4.761905
2019-01-05	1/5/2019	11:51	Credit card	657.16	4.761905

	gross income	Rating
Date		
2019-01-05	26.1415	9.1
2019-01-05	8.2140	7.9
2019-01-05	12.5300	4.2
2019-01-05	14.1480	4.4
2019-01-05	15.9210	9.6
2019-01-05	1.5950	9.1
2019-01-05	10.7275	6.1
2019-01-05	3.6980	5.0
2019-01-05	3.5600	9.2
2019-01-05	11.4090	9.8
2019-01-05	27.6115	7.5
2019-01-05	32.8580	7.3

When you have a single-column index, you can use it to select specific rows based on the index value.

Index values don't need to be unique !

You can also filter for multiple index values

```
[11]: df.loc[['2019-01-05', '2019-01-06']]
```

[11]:

	Invoice ID	Branch	City	Customer type	Gender	\
Date						
2019-01-05	750-67-8428	A	Yangon	Member	Female	
2019-01-05	628-34-3388	C	Naypyitaw	Normal	Male	
2019-01-05	217-58-1179	A	Yangon	Member	Male	
2019-01-05	144-51-6085	A	Yangon	Member	Male	
2019-01-05	843-01-4703	B	Mandalay	Member	Female	
2019-01-05	573-98-8548	C	Naypyitaw	Member	Female	
2019-01-05	209-61-0206	A	Yangon	Normal	Female	
2019-01-05	339-12-4827	B	Mandalay	Member	Female	
2019-01-05	841-18-8232	B	Mandalay	Normal	Female	
2019-01-05	801-88-0346	C	Naypyitaw	Normal	Female	
2019-01-05	433-08-7822	C	Naypyitaw	Normal	Female	
2019-01-05	489-82-1237	A	Yangon	Normal	Female	
2019-01-06	393-65-2792	C	Naypyitaw	Normal	Male	
2019-01-06	704-11-6354	A	Yangon	Member	Male	
2019-01-06	490-29-1201	A	Yangon	Normal	Female	
2019-01-06	834-25-9262	C	Naypyitaw	Normal	Female	
2019-01-06	662-72-2873	A	Yangon	Normal	Female	
2019-01-06	826-58-8051	B	Mandalay	Normal	Male	
2019-01-06	729-06-2010	B	Mandalay	Member	Male	
2019-01-06	852-82-2749	A	Yangon	Normal	Male	
2019-01-06	846-10-0341	A	Yangon	Normal	Female	

	Product line	Unit price	Quantity	Tax 5%	Total	\
Date						
2019-01-05	Health and beauty	74.69	7	26.1415	548.9715	
2019-01-05	Fashion accessories	27.38	6	8.2140	172.4940	
2019-01-05	Home and lifestyle	62.65	4	12.5300	263.1300	
2019-01-05	Home and lifestyle	70.74	4	14.1480	297.1080	
2019-01-05	Home and lifestyle	35.38	9	15.9210	334.3410	
2019-01-05	Fashion accessories	31.90	1	1.5950	33.4950	
2019-01-05	Home and lifestyle	42.91	5	10.7275	225.2775	
2019-01-05	Fashion accessories	73.96	1	3.6980	77.6580	
2019-01-05	Food and beverages	71.20	1	3.5600	74.7600	
2019-01-05	Fashion accessories	76.06	3	11.4090	239.5890	
2019-01-05	Health and beauty	78.89	7	27.6115	579.8415	
2019-01-05	Electronic accessories	93.88	7	32.8580	690.0180	
2019-01-06	Food and beverages	89.48	10	44.7400	939.5400	
2019-01-06	Home and lifestyle	58.90	8	23.5600	494.7600	
2019-01-06	Sports and travel	15.34	1	0.7670	16.1070	
2019-01-06	Fashion accessories	81.68	4	16.3360	343.0560	
2019-01-06	Food and beverages	40.94	5	10.2350	214.9350	
2019-01-06	Home and lifestyle	62.19	4	12.4380	261.1980	
2019-01-06	Health and beauty	80.47	9	36.2115	760.4415	
2019-01-06	Sports and travel	64.59	4	12.9180	271.2780	
2019-01-06	Fashion accessories	42.57	7	14.8995	312.8895	

	Date	Time	Payment	cogs	gross margin percentage \
Date					
2019-01-05	1/5/2019	13:08	Ewallet	522.83	4.761905
2019-01-05	1/5/2019	20:54	Credit card	164.28	4.761905
2019-01-05	1/5/2019	11:25	Cash	250.60	4.761905
2019-01-05	1/5/2019	16:05	Credit card	282.96	4.761905
2019-01-05	1/5/2019	19:50	Credit card	318.42	4.761905
2019-01-05	1/5/2019	12:40	Ewallet	31.90	4.761905
2019-01-05	1/5/2019	17:29	Ewallet	214.55	4.761905
2019-01-05	1/5/2019	11:32	Credit card	73.96	4.761905
2019-01-05	1/5/2019	20:40	Credit card	71.20	4.761905
2019-01-05	1/5/2019	20:30	Credit card	228.18	4.761905
2019-01-05	1/5/2019	19:48	Ewallet	552.23	4.761905
2019-01-05	1/5/2019	11:51	Credit card	657.16	4.761905
2019-01-06	1/6/2019	12:46	Credit card	894.80	4.761905
2019-01-06	1/6/2019	11:23	Cash	471.20	4.761905
2019-01-06	1/6/2019	11:09	Cash	15.34	4.761905
2019-01-06	1/6/2019	12:12	Cash	326.72	4.761905
2019-01-06	1/6/2019	13:58	Ewallet	204.70	4.761905
2019-01-06	1/6/2019	19:46	Ewallet	248.76	4.761905
2019-01-06	1/6/2019	11:18	Cash	724.23	4.761905
2019-01-06	1/6/2019	13:35	Ewallet	258.36	4.761905
2019-01-06	1/6/2019	11:51	Cash	297.99	4.761905

	gross income	Rating
Date		
2019-01-05	26.1415	9.1
2019-01-05	8.2140	7.9
2019-01-05	12.5300	4.2
2019-01-05	14.1480	4.4
2019-01-05	15.9210	9.6
2019-01-05	1.5950	9.1
2019-01-05	10.7275	6.1
2019-01-05	3.6980	5.0
2019-01-05	3.5600	9.2
2019-01-05	11.4090	9.8
2019-01-05	27.6115	7.5
2019-01-05	32.8580	7.3
2019-01-06	44.7400	9.6
2019-01-06	23.5600	8.9
2019-01-06	0.7670	6.5
2019-01-06	16.3360	9.1
2019-01-06	10.2350	9.9
2019-01-06	12.4380	4.3
2019-01-06	36.2115	9.2
2019-01-06	12.9180	9.3

2019-01-06 14.8995 6.8

Indexes can also be used to subset a range of rows if they are sorted in a meaningful order (e.g., dates or numerical ranges).

```
[13]: df.sort_index().loc['2019-01-05': '2019-01-07']
```

```
[13]: Invoice ID Branch          City Customer type    Gender \
```

Date

2019-01-05	801-88-0346	C	Naypyitaw	Normal	Female
2019-01-05	628-34-3388	C	Naypyitaw	Normal	Male
2019-01-05	750-67-8428	A	Yangon	Member	Female
2019-01-05	841-18-8232	B	Mandalay	Normal	Female
2019-01-05	339-12-4827	B	Mandalay	Member	Female
2019-01-05	489-82-1237	A	Yangon	Normal	Female
2019-01-05	144-51-6085	A	Yangon	Member	Male
2019-01-05	209-61-0206	A	Yangon	Normal	Female
2019-01-05	573-98-8548	C	Naypyitaw	Member	Female
2019-01-05	843-01-4703	B	Mandalay	Member	Female
2019-01-05	217-58-1179	A	Yangon	Member	Male
2019-01-05	433-08-7822	C	Naypyitaw	Normal	Female
2019-01-06	704-11-6354	A	Yangon	Member	Male
2019-01-06	826-58-8051	B	Mandalay	Normal	Male
2019-01-06	729-06-2010	B	Mandalay	Member	Male
2019-01-06	393-65-2792	C	Naypyitaw	Normal	Male
2019-01-06	490-29-1201	A	Yangon	Normal	Female
2019-01-06	662-72-2873	A	Yangon	Normal	Female
2019-01-06	834-25-9262	C	Naypyitaw	Normal	Female
2019-01-06	846-10-0341	A	Yangon	Normal	Female
2019-01-06	852-82-2749	A	Yangon	Normal	Male
2019-01-07	232-16-2483	C	Naypyitaw	Member	Female
2019-01-07	390-31-6381	C	Naypyitaw	Normal	Male
2019-01-07	541-48-8554	A	Yangon	Normal	Male
2019-01-07	889-04-9723	B	Mandalay	Member	Female
2019-01-07	566-19-5475	B	Mandalay	Normal	Male
2019-01-07	526-86-8552	C	Naypyitaw	Member	Female
2019-01-07	109-28-2512	B	Mandalay	Member	Female
2019-01-07	126-54-1082	A	Yangon	Member	Female
2019-01-07	799-71-1548	A	Yangon	Member	Male

```
Product line    Unit price    Quantity    Tax 5%    Total \
```

Date

2019-01-05	Fashion accessories	76.06	3	11.4090	239.5890
2019-01-05	Fashion accessories	27.38	6	8.2140	172.4940
2019-01-05	Health and beauty	74.69	7	26.1415	548.9715
2019-01-05	Food and beverages	71.20	1	3.5600	74.7600
2019-01-05	Fashion accessories	73.96	1	3.6980	77.6580

2019-01-05	Electronic accessories	93.88	7	32.8580	690.0180
2019-01-05	Home and lifestyle	70.74	4	14.1480	297.1080
2019-01-05	Home and lifestyle	42.91	5	10.7275	225.2775
2019-01-05	Fashion accessories	31.90	1	1.5950	33.4950
2019-01-05	Home and lifestyle	35.38	9	15.9210	334.3410
2019-01-05	Home and lifestyle	62.65	4	12.5300	263.1300
2019-01-05	Health and beauty	78.89	7	27.6115	579.8415
2019-01-06	Home and lifestyle	58.90	8	23.5600	494.7600
2019-01-06	Home and lifestyle	62.19	4	12.4380	261.1980
2019-01-06	Health and beauty	80.47	9	36.2115	760.4415
2019-01-06	Food and beverages	89.48	10	44.7400	939.5400
2019-01-06	Sports and travel	15.34	1	0.7670	16.1070
2019-01-06	Food and beverages	40.94	5	10.2350	214.9350
2019-01-06	Fashion accessories	81.68	4	16.3360	343.0560
2019-01-06	Fashion accessories	42.57	7	14.8995	312.8895
2019-01-06	Sports and travel	64.59	4	12.9180	271.2780
2019-01-07	Sports and travel	68.12	1	3.4060	71.5260
2019-01-07	Food and beverages	27.22	3	4.0830	85.7430
2019-01-07	Sports and travel	60.95	9	27.4275	575.9775
2019-01-07	Food and beverages	89.14	4	17.8280	374.3880
2019-01-07	Fashion accessories	47.97	7	16.7895	352.5795
2019-01-07	Home and lifestyle	21.82	10	10.9100	229.1100
2019-01-07	Fashion accessories	97.61	6	29.2830	614.9430
2019-01-07	Home and lifestyle	21.54	9	9.6930	203.5530
2019-01-07	Electronic accessories	77.72	4	15.5440	326.4240

	Date	Time	Payment	cogs	gross margin percentage \
Date					
2019-01-05	1/5/2019	20:30	Credit card	228.18	4.761905
2019-01-05	1/5/2019	20:54	Credit card	164.28	4.761905
2019-01-05	1/5/2019	13:08	Ewallet	522.83	4.761905
2019-01-05	1/5/2019	20:40	Credit card	71.20	4.761905
2019-01-05	1/5/2019	11:32	Credit card	73.96	4.761905
2019-01-05	1/5/2019	11:51	Credit card	657.16	4.761905
2019-01-05	1/5/2019	16:05	Credit card	282.96	4.761905
2019-01-05	1/5/2019	17:29	Ewallet	214.55	4.761905
2019-01-05	1/5/2019	12:40	Ewallet	31.90	4.761905
2019-01-05	1/5/2019	19:50	Credit card	318.42	4.761905
2019-01-05	1/5/2019	11:25	Cash	250.60	4.761905
2019-01-05	1/5/2019	19:48	Ewallet	552.23	4.761905
2019-01-06	1/6/2019	11:23	Cash	471.20	4.761905
2019-01-06	1/6/2019	19:46	Ewallet	248.76	4.761905
2019-01-06	1/6/2019	11:18	Cash	724.23	4.761905
2019-01-06	1/6/2019	12:46	Credit card	894.80	4.761905
2019-01-06	1/6/2019	11:09	Cash	15.34	4.761905
2019-01-06	1/6/2019	13:58	Ewallet	204.70	4.761905
2019-01-06	1/6/2019	12:12	Cash	326.72	4.761905

2019-01-06	1/6/2019	11:51	Cash	297.99	4.761905
2019-01-06	1/6/2019	13:35	Ewallet	258.36	4.761905
2019-01-07	1/7/2019	12:28	Ewallet	68.12	4.761905
2019-01-07	1/7/2019	12:37	Cash	81.66	4.761905
2019-01-07	1/7/2019	12:08	Credit card	548.55	4.761905
2019-01-07	1/7/2019	12:20	Credit card	356.56	4.761905
2019-01-07	1/7/2019	20:52	Cash	335.79	4.761905
2019-01-07	1/7/2019	17:36	Cash	218.20	4.761905
2019-01-07	1/7/2019	15:01	Ewallet	585.66	4.761905
2019-01-07	1/7/2019	11:44	Credit card	193.86	4.761905
2019-01-07	1/7/2019	16:11	Credit card	310.88	4.761905

Date	gross income	Rating
2019-01-05	11.4090	9.8
2019-01-05	8.2140	7.9
2019-01-05	26.1415	9.1
2019-01-05	3.5600	9.2
2019-01-05	3.6980	5.0
2019-01-05	32.8580	7.3
2019-01-05	14.1480	4.4
2019-01-05	10.7275	6.1
2019-01-05	1.5950	9.1
2019-01-05	15.9210	9.6
2019-01-05	12.5300	4.2
2019-01-05	27.6115	7.5
2019-01-06	23.5600	8.9
2019-01-06	12.4380	4.3
2019-01-06	36.2115	9.2
2019-01-06	44.7400	9.6
2019-01-06	0.7670	6.5
2019-01-06	10.2350	9.9
2019-01-06	16.3360	9.1
2019-01-06	14.8995	6.8
2019-01-06	12.9180	9.3
2019-01-07	3.4060	6.8
2019-01-07	4.0830	7.3
2019-01-07	27.4275	6.0
2019-01-07	17.8280	7.8
2019-01-07	16.7895	6.2
2019-01-07	10.9100	7.1
2019-01-07	29.2830	9.9
2019-01-07	9.6930	8.8
2019-01-07	15.5440	8.8

1.5.7 Multi-Level (Hierarchical) index:

Pandas also supports multi-level (or hierarchical) indexing, which allows for more complex data organization. Multi-level indexes are often used when working with multi-dimensional data.

Creating a multi-level index:

```
[14]: df = df.set_index(['Branch', 'Product line'])
      df
```

```
[14]:
```

		Invoice ID	City	Customer type	Gender	\
Branch	Product line					
A	Health and beauty	750-67-8428	Yangon	Member	Female	
C	Electronic accessories	226-31-3081	Naypyitaw	Normal	Female	
A	Home and lifestyle	631-41-3108	Yangon	Normal	Male	
	Health and beauty	123-19-1176	Yangon	Member	Male	
	Sports and travel	373-73-7910	Yangon	Normal	Male	
...		
C	Health and beauty	233-67-5758	Naypyitaw	Normal	Male	
B	Home and lifestyle	303-96-2227	Mandalay	Normal	Female	
A	Food and beverages	727-02-1313	Yangon	Member	Male	
	Home and lifestyle	347-56-2442	Yangon	Normal	Male	
	Fashion accessories	849-09-3807	Yangon	Member	Female	

		Unit price	Quantity	Tax 5%	Total	\
Branch	Product line					
A	Health and beauty	74.69	7	26.1415	548.9715	
C	Electronic accessories	15.28	5	3.8200	80.2200	
A	Home and lifestyle	46.33	7	16.2155	340.5255	
	Health and beauty	58.22	8	23.2880	489.0480	
	Sports and travel	86.31	7	30.2085	634.3785	
...		
C	Health and beauty	40.35	1	2.0175	42.3675	
B	Home and lifestyle	97.38	10	48.6900	1022.4900	
A	Food and beverages	31.84	1	1.5920	33.4320	
	Home and lifestyle	65.82	1	3.2910	69.1110	
	Fashion accessories	88.34	7	30.9190	649.2990	

		Date	Time	Payment	cogs	\
Branch	Product line					
A	Health and beauty	1/5/2019	13:08	Ewallet	522.83	
C	Electronic accessories	3/8/2019	10:29	Cash	76.40	
A	Home and lifestyle	3/3/2019	13:23	Credit card	324.31	
	Health and beauty	1/27/2019	20:33	Ewallet	465.76	
	Sports and travel	2/8/2019	10:37	Ewallet	604.17	
...		
C	Health and beauty	1/29/2019	13:46	Ewallet	40.35	

B	Home and lifestyle	3/2/2019	17:16	Ewallet	973.80
A	Food and beverages	2/9/2019	13:22	Cash	31.84
	Home and lifestyle	2/22/2019	15:33	Cash	65.82
	Fashion accessories	2/18/2019	13:28	Cash	618.38

		gross margin percentage	gross income	Rating
Branch	Product line			
A	Health and beauty	4.761905	26.1415	9.1
C	Electronic accessories	4.761905	3.8200	9.6
A	Home and lifestyle	4.761905	16.2155	7.4
	Health and beauty	4.761905	23.2880	8.4
	Sports and travel	4.761905	30.2085	5.3
...	
C	Health and beauty	4.761905	2.0175	6.2
B	Home and lifestyle	4.761905	48.6900	4.4
A	Food and beverages	4.761905	1.5920	7.7
	Home and lifestyle	4.761905	3.2910	4.1
	Fashion accessories	4.761905	30.9190	6.6

[1000 rows x 15 columns]

Here, the DataFrame is indexed by both Branch and Product line.

Filtering with multi-level index: With a multi-level index, you can filter data based on one or more index levels.

```
[15]: df.loc[("A", "Health and beauty")]
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_2900\2993678514.py:1:
PerformanceWarning: indexing past lexsort depth may impact performance.
df.loc[("A", "Health and beauty")]
```

```
[15]:
```

		Invoice ID	City	Customer type	Gender	\
Branch	Product line					
A	Health and beauty	750-67-8428	Yangon	Member	Female	
	Health and beauty	123-19-1176	Yangon	Member	Male	
	Health and beauty	665-32-9167	Yangon	Member	Female	
	Health and beauty	829-34-3910	Yangon	Normal	Female	
	Health and beauty	656-95-9349	Yangon	Member	Female	
	Health and beauty	848-62-7243	Yangon	Normal	Male	
	Health and beauty	595-11-5460	Yangon	Normal	Male	
	Health and beauty	635-40-6220	Yangon	Normal	Male	
	Health and beauty	877-22-3308	Yangon	Member	Male	
	Health and beauty	382-03-4532	Yangon	Member	Female	
	Health and beauty	238-49-0436	Yangon	Normal	Male	
	Health and beauty	249-42-3782	Yangon	Normal	Male	
	Health and beauty	749-24-1565	Yangon	Normal	Female	

Health and beauty	157-13-5295	Yangon	Member	Male
Health and beauty	448-81-5016	Yangon	Normal	Male
Health and beauty	667-92-0055	Yangon	Member	Male
Health and beauty	565-17-3836	Yangon	Member	Female
Health and beauty	423-64-4619	Yangon	Member	Female
Health and beauty	372-94-8041	Yangon	Normal	Male
Health and beauty	725-56-0833	Yangon	Normal	Female
Health and beauty	885-17-6250	Yangon	Normal	Female
Health and beauty	726-27-2396	Yangon	Normal	Female
Health and beauty	132-23-6451	Yangon	Member	Male
Health and beauty	269-10-8440	Yangon	Member	Male
Health and beauty	568-88-3448	Yangon	Normal	Male
Health and beauty	502-05-1910	Yangon	Normal	Male
Health and beauty	287-83-1405	Yangon	Normal	Male
Health and beauty	797-88-0493	Yangon	Normal	Female
Health and beauty	443-82-0585	Yangon	Member	Female
Health and beauty	127-47-6963	Yangon	Normal	Male
Health and beauty	227-50-3718	Yangon	Normal	Male
Health and beauty	610-46-4100	Yangon	Normal	Male
Health and beauty	689-05-1884	Yangon	Member	Male
Health and beauty	258-92-7466	Yangon	Normal	Female
Health and beauty	635-28-5728	Yangon	Normal	Male
Health and beauty	587-73-4862	Yangon	Member	Female
Health and beauty	787-87-2010	Yangon	Member	Male
Health and beauty	276-75-6884	Yangon	Normal	Female
Health and beauty	528-14-9470	Yangon	Member	Male
Health and beauty	160-22-2687	Yangon	Member	Female
Health and beauty	595-94-9924	Yangon	Member	Female
Health and beauty	397-25-8725	Yangon	Member	Female
Health and beauty	131-70-8179	Yangon	Member	Female
Health and beauty	449-16-6770	Yangon	Normal	Male
Health and beauty	333-23-2632	Yangon	Member	Male
Health and beauty	605-03-2706	Yangon	Normal	Female
Health and beauty	809-46-1866	Yangon	Normal	Male

		Unit price	Quantity	Tax 5%	Total	Date \
Branch	Product line					
A	Health and beauty	74.69	7	26.1415	548.9715	1/5/2019
	Health and beauty	58.22	8	23.2880	489.0480	1/27/2019
	Health and beauty	36.26	2	3.6260	76.1460	1/10/2019
	Health and beauty	71.38	10	35.6900	749.4900	3/29/2019
	Health and beauty	68.93	7	24.1255	506.6355	3/11/2019
	Health and beauty	24.89	9	11.2005	235.2105	3/15/2019
	Health and beauty	96.58	2	9.6580	202.8180	3/15/2019
	Health and beauty	89.60	8	35.8400	752.6400	2/7/2019
	Health and beauty	15.87	10	7.9350	166.6350	3/13/2019
	Health and beauty	18.33	1	0.9165	19.2465	2/2/2019

Health and beauty	32.46	8	12.9840	272.6640	3/27/2019
Health and beauty	70.01	5	17.5025	367.5525	1/3/2019
Health and beauty	23.03	9	10.3635	217.6335	1/3/2019
Health and beauty	51.94	10	25.9700	545.3700	3/9/2019
Health and beauty	59.77	2	5.9770	125.5170	3/11/2019
Health and beauty	99.83	6	29.9490	628.9290	3/4/2019
Health and beauty	47.67	4	9.5340	200.2140	3/12/2019
Health and beauty	15.55	9	6.9975	146.9475	3/7/2019
Health and beauty	15.26	6	4.5780	96.1380	2/15/2019
Health and beauty	32.32	10	16.1600	339.3600	2/20/2019
Health and beauty	79.74	1	3.9870	83.7270	3/6/2019
Health and beauty	77.50	5	19.3750	406.8750	1/24/2019
Health and beauty	20.97	5	5.2425	110.0925	1/4/2019
Health and beauty	53.17	7	18.6095	390.7995	1/21/2019
Health and beauty	25.00	1	1.2500	26.2500	3/3/2019
Health and beauty	65.18	3	9.7770	205.3170	2/25/2019
Health and beauty	25.43	6	7.6290	160.2090	2/12/2019
Health and beauty	64.27	4	12.8540	269.9340	3/26/2019
Health and beauty	77.68	4	15.5360	326.2560	2/1/2019
Health and beauty	51.71	4	10.3420	217.1820	3/9/2019
Health and beauty	14.62	5	3.6550	76.7550	3/4/2019
Health and beauty	28.95	7	10.1325	212.7825	3/3/2019
Health and beauty	48.63	10	24.3150	510.6150	3/4/2019
Health and beauty	35.68	5	8.9200	187.3200	2/6/2019
Health and beauty	56.00	3	8.4000	176.4000	2/28/2019
Health and beauty	10.69	5	2.6725	56.1225	3/26/2019
Health and beauty	55.50	4	11.1000	233.1000	1/20/2019
Health and beauty	68.71	3	10.3065	216.4365	3/4/2019
Health and beauty	91.30	1	4.5650	95.8650	2/14/2019
Health and beauty	95.95	5	23.9875	503.7375	1/23/2019
Health and beauty	27.73	5	6.9325	145.5825	3/26/2019
Health and beauty	39.62	9	17.8290	374.4090	1/13/2019
Health and beauty	92.09	3	13.8135	290.0835	2/17/2019
Health and beauty	50.79	5	12.6975	266.6475	2/19/2019
Health and beauty	10.08	7	3.5280	74.0880	3/28/2019
Health and beauty	15.80	3	2.3700	49.7700	3/25/2019
Health and beauty	58.15	4	11.6300	244.2300	1/23/2019

		Time	Payment	cogs	gross margin percentage \
Branch	Product line				
A	Health and beauty	13:08	Ewallet	522.83	4.761905
	Health and beauty	20:33	Ewallet	465.76	4.761905
	Health and beauty	17:15	Credit card	72.52	4.761905
	Health and beauty	19:21	Cash	713.80	4.761905
	Health and beauty	11:03	Credit card	482.51	4.761905
	Health and beauty	15:36	Cash	224.01	4.761905
	Health and beauty	10:12	Credit card	193.16	4.761905

Health and beauty	11:28	Ewallet	716.80	4.761905
Health and beauty	16:40	Cash	158.70	4.761905
Health and beauty	18:50	Cash	18.33	4.761905
Health and beauty	13:48	Credit card	259.68	4.761905
Health and beauty	11:36	Ewallet	350.05	4.761905
Health and beauty	12:02	Ewallet	207.27	4.761905
Health and beauty	18:24	Ewallet	519.40	4.761905
Health and beauty	12:01	Credit card	119.54	4.761905
Health and beauty	15:02	Ewallet	598.98	4.761905
Health and beauty	14:21	Cash	190.68	4.761905
Health and beauty	13:12	Cash	139.95	4.761905
Health and beauty	18:03	Ewallet	91.56	4.761905
Health and beauty	16:49	Credit card	323.20	4.761905
Health and beauty	10:36	Ewallet	79.74	4.761905
Health and beauty	20:36	Ewallet	387.50	4.761905
Health and beauty	13:21	Cash	104.85	4.761905
Health and beauty	18:01	Cash	372.19	4.761905
Health and beauty	15:09	Ewallet	25.00	4.761905
Health and beauty	20:35	Credit card	195.54	4.761905
Health and beauty	19:01	Ewallet	152.58	4.761905
Health and beauty	13:54	Cash	257.08	4.761905
Health and beauty	19:54	Cash	310.72	4.761905
Health and beauty	13:53	Credit card	206.84	4.761905
Health and beauty	12:23	Cash	73.10	4.761905
Health and beauty	20:31	Credit card	202.65	4.761905
Health and beauty	12:44	Cash	486.30	4.761905
Health and beauty	18:33	Credit card	178.40	4.761905
Health and beauty	19:33	Ewallet	168.00	4.761905
Health and beauty	11:07	Ewallet	53.45	4.761905
Health and beauty	15:48	Credit card	222.00	4.761905
Health and beauty	10:05	Cash	206.13	4.761905
Health and beauty	14:42	Ewallet	91.30	4.761905
Health and beauty	14:21	Ewallet	479.75	4.761905
Health and beauty	20:21	Credit card	138.65	4.761905
Health and beauty	17:54	Credit card	356.58	4.761905
Health and beauty	16:27	Cash	276.27	4.761905
Health and beauty	14:53	Credit card	253.95	4.761905
Health and beauty	20:14	Cash	70.56	4.761905
Health and beauty	18:02	Cash	47.40	4.761905
Health and beauty	17:44	Cash	232.60	4.761905

		gross income	Rating
Branch	Product line		
A	Health and beauty	26.1415	9.1
	Health and beauty	23.2880	8.4
	Health and beauty	3.6260	7.2
	Health and beauty	35.6900	5.7

Health and beauty	24.1255	4.6
Health and beauty	11.2005	7.4
Health and beauty	9.6580	5.1
Health and beauty	35.8400	6.6
Health and beauty	7.9350	5.8
Health and beauty	0.9165	4.3
Health and beauty	12.9840	4.9
Health and beauty	17.5025	5.5
Health and beauty	10.3635	7.9
Health and beauty	25.9700	6.5
Health and beauty	5.9770	5.8
Health and beauty	29.9490	8.5
Health and beauty	9.5340	9.1
Health and beauty	6.9975	5.0
Health and beauty	4.5780	9.8
Health and beauty	16.1600	10.0
Health and beauty	3.9870	7.3
Health and beauty	19.3750	4.3
Health and beauty	5.2425	7.8
Health and beauty	18.6095	8.9
Health and beauty	1.2500	5.5
Health and beauty	9.7770	6.3
Health and beauty	7.6290	7.0
Health and beauty	12.8540	7.7
Health and beauty	15.5360	8.4
Health and beauty	10.3420	9.8
Health and beauty	3.6550	4.4
Health and beauty	10.1325	6.0
Health and beauty	24.3150	8.8
Health and beauty	8.9200	6.6
Health and beauty	8.4000	4.8
Health and beauty	2.6725	7.6
Health and beauty	11.1000	6.6
Health and beauty	10.3065	8.7
Health and beauty	4.5650	9.2
Health and beauty	23.9875	8.8
Health and beauty	6.9325	4.2
Health and beauty	17.8290	6.8
Health and beauty	13.8135	4.2
Health and beauty	12.6975	5.3
Health and beauty	3.5280	4.2
Health and beauty	2.3700	9.5
Health and beauty	11.6300	8.4

Filtering the outer level only:

```
[28]: df.loc[(slice(None), "Health and beauty"), :]
```

[28]:

		Invoice ID	City	Customer type	Gender	\
Branch	Product line					
A	Health and beauty	750-67-8428	Yangon	Member	Female	
	Health and beauty	123-19-1176	Yangon	Member	Male	
	Health and beauty	665-32-9167	Yangon	Member	Female	
	Health and beauty	829-34-3910	Yangon	Normal	Female	
	Health and beauty	656-95-9349	Yangon	Member	Female	
...		
C	Health and beauty	148-41-7930	Naypyitaw	Normal	Male	
B	Health and beauty	764-44-8999	Mandalay	Normal	Female	
	Health and beauty	552-44-5977	Mandalay	Member	Male	
	Health and beauty	430-53-4718	Mandalay	Member	Male	
C	Health and beauty	233-67-5758	Naypyitaw	Normal	Male	

		Unit price	Quantity	Tax 5%	Total	Date	\
Branch	Product line						
A	Health and beauty	74.69	7	26.1415	548.9715	1/5/2019	
	Health and beauty	58.22	8	23.2880	489.0480	1/27/2019	
	Health and beauty	36.26	2	3.6260	76.1460	1/10/2019	
	Health and beauty	71.38	10	35.6900	749.4900	3/29/2019	
	Health and beauty	68.93	7	24.1255	506.6355	3/11/2019	
...		
C	Health and beauty	99.96	7	34.9860	734.7060	1/23/2019	
B	Health and beauty	14.76	2	1.4760	30.9960	2/18/2019	
	Health and beauty	62.00	8	24.8000	520.8000	1/3/2019	
	Health and beauty	75.37	8	30.1480	633.1080	1/28/2019	
C	Health and beauty	40.35	1	2.0175	42.3675	1/29/2019	

		Time	Payment	cogs	gross margin percentage	\
Branch	Product line					
A	Health and beauty	13:08	Ewallet	522.83	4.761905	
	Health and beauty	20:33	Ewallet	465.76	4.761905	
	Health and beauty	17:15	Credit card	72.52	4.761905	
	Health and beauty	19:21	Cash	713.80	4.761905	
	Health and beauty	11:03	Credit card	482.51	4.761905	
...		
C	Health and beauty	10:33	Cash	699.72	4.761905	
B	Health and beauty	14:42	Ewallet	29.52	4.761905	
	Health and beauty	19:08	Credit card	496.00	4.761905	
	Health and beauty	15:46	Credit card	602.96	4.761905	
C	Health and beauty	13:46	Ewallet	40.35	4.761905	

		gross income	Rating
Branch	Product line		
A	Health and beauty	26.1415	9.1
	Health and beauty	23.2880	8.4
	Health and beauty	3.6260	7.2

	Health and beauty	35.6900	5.7
	Health and beauty	24.1255	4.6
...	
C	Health and beauty	34.9860	6.1
B	Health and beauty	1.4760	4.3
	Health and beauty	24.8000	6.2
	Health and beauty	30.1480	8.4
C	Health and beauty	2.0175	6.2

[152 rows x 15 columns]