Exercise 1: Practice your Pandas skills!

This notebook contains exercises on data exploration and transformation using the Pandas library. It does **not** directly follow the book or user guide, but rather contains exercises that are practically motivated and require a wide variety of Pandas functionality.

Learning Goals

- Understand how to think about data manipulation & analysis in Pandas.
- Know how to use *indexing, smart indexing, and aggregations* in Pandas.
- Know how to use *grouping operations* in Pandas.
- Know how to *combine* datasets in Pandas.

Remember that you don't need to learn everything by heart; it's okay to need to look up things, but you should become more proficient in the basic principles & techniques needed to solve exercises like these.

Useful Resources

- *Python Data Science Handbook* by Jake VanderPlas, Ch.3: Data Manipulation with Pandas (https://jakevdp.github.io/PythonDataScienceHandbook/index.html#3.-Data-Manipulation-with-Pandas)
- Pandas User Guide (https://pandas.pydata.org/docs/user_guide/)

The datasets used here contain *synthetically generated* (i.e., fake) data which is based on, but not identical to, the "Coffee Chain" dataset from Kaggle (https://www.kaggle.com/datasets/qusaybtoush1990/coffee-chain).

In [2]:

```
#%load_ext rich
import pandas as pd
```

In [3]:

```
# Load the data into a Pandas DataFrame
data = "data/coffee-chain.csv"
df = pd.read_csv(data)
df
```

Out[3]:

	Ddate	Market	Product	Product Type	Sales	Profit	Expenses
0	9/1/13	Central	Decaf Irish Cream	Coffee	90	36	37
1	5/1/12	East	Decaf Espresso	Espresso	203	56	55
2	1/1/13	West	Caffe Latte	Espresso	524	136	93
3	8/1/12	West	Earl Grey	Tea	273	81	68
4	12/1/13	West	Amaretto	Coffee	201	-3	68
...	...	...	...	...	...	...	...
9995	2/1/12	Central	Darjeeling	Tea	185	59	39
9996	3/1/12	East	Darjeeling	Tea	88	17	36
9997	8/1/13	Central	Chamomile	Herbal Tea	61	21	21
9998	4/1/12	East	Caffe Mocha	Espresso	124	38	84
9999	5/1/12	West	Darjeeling	Tea	524	150	89

10000 rows × 7 columns

Data Exploration

Here are some common questions that you might want to explore on a dataset like this. They all have in common that they ***can be solved with just one line of code*** in Pandas. That doesn't mean they're trivial – they require you to understand how Pandas works and how you can think about data selection and transformation in Pandas. Sometimes there are also several different ways to achieve the same result. If you can implement a solution that needs more than one line of code, that's also fine – but I'd encourage you to try to find the one-line solution as well, simply in order to learn more about the possibilities that Pandas offers.

If you're lost, don't forget you can reference the book (https://jakevdp.github.io/PythonDataScienceHandbook/index.html#3.-Data-Manipulation-with-Pandas) or the user guide (https://pandas.pydata.org/docs/user_guide/).

1. Does the dataset have any missing values? Write an expression to obtain a *single boolean value* (i.e., True or False) that signifies whether the DataFrame has any missing values or not.

In [31]:

```
# Your code here
from numpy import nan
def has_null(df):
    for n in df:
        if n == nan:
            return False
    return True

has_null(df)
```

Out[31]:

```
True
```

In [33]:

```
df([np.nan]).any()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-33-bbff9a0dadf7> in <module>
----> 1 df([np.nan]).any()

NameError: name 'np' is not defined
```

2. What different *product types* are there in the dataset? Produce a list of all unique product types in the dataset, i.e., unique values in the "Product Type" column.

In [25]:

```
# Your code here
df['Product Type'].unique()
```

Out[25]:

```
array(['Coffee', 'Espresso', 'Tea', 'Herbal Tea'], dtype=object)
```

3. What different *"coffee" products* are there in the dataset? Produce a list of all unique coffee products in the dataset, i.e., values in the "Product" column where the "Product Type" is "Coffee".

In [27]:

```
# Your code here
df.groupby("Product Type")["Product"].unique()
#df.Product.unique()
```

Out[27]:

```
Product Type
Coffee                 [Decaf Irish Cream, Amaretto, Colombian]
Espresso      [Decaf Espresso, Caffe Latte, Caffe Mocha, Reg...
Herbal Tea                              [Lemon, Mint, Chamomile]
Tea                          [Earl Grey, Darjeeling, Green Tea]
Name: Product, dtype: object
```

4. What different *"coffee or espresso" products* are there in the dataset? Do the same as above, but this time include all products where the "Product Type" is either "Coffee" or "Espresso". Try to figure out how to do

this in a single Pandas expression — i.e., how can you select all rows in the dataset that match the required condition?

Hint: You might want to consult "Indexing and Selecting Data" in the user guide (https://pandas.pydata.org/docs/user_guide/indexing.html).

In [6]:

```
# Your code here
```

5. Which product type generates the most/the least profit, on average? Find out the average "Profit" values by "Product Type".

In [7]:

```
# Your code here
```

6. What's the *distribution* of profits among all products? Produce a table that has unique "Product" values (e.g., Amaretto, Caffe Latte, ...) as *rows*, and statistics about the distribution of their "Profit" values (e.g., count, mean, min, max, ...) as *columns*.

Hint: The statistics about distributions can be obtained with the `.describe()` method.

In [8]:

```
# Your code here
```

7. What are the average profits by products *and* market region? From the solution to the previous question, we know the mean "Profit" values for each "Product" in the dataset. Now, we want to look at the average profit by *two* variables: "Product" and "Market". Produce a table that has products as *rows*, markets as *columns*, and the mean profits of the product–market combination as *values*.

Hint: This is a classic use case for pivot tables (https://jakevdp.github.io/PythonDataScienceHandbook/03.09-pivot-tables.html).

In [9]:

```
# Your code here
```

8. How many instances are there in the dataset of each product–market combination? If you produced the pivot table for the previous question, you should see that several values in the table are "NaN" (*not a number*). It looks like some combinations of "Product" and "Market" might not exist in the dataset at all. Modify the pivot table from Q7 so that instead of averages, it shows *counts* (i.e., how many values there are in total), and instead of "NaN" it fills in `0` for the missing combinations.

In [10]:

```
# Your code here
```

Data Preprocessing

So far we just looked at the data in different ways, now we're going to modify it.

9. Filter out all products where we don't have data from all markets. In Q8, we saw that some product–market combinations don't appear in the dataset at all. Maybe we want to do an analysis based on products and market regions, and therefore only want to keep products where we have data from *all* of the four markets. In other words, we want to drop all the products where one of the markets has a count of `0`. Make a new DataFrame `df_subset` that consists of only the rows in `df` which fulfill this criterion.

Hint: This is possible to implement in one line, but it might easier to think about if you define a "helper function" first. This helper function should take a Pandas data structure and return `True` if it contains data from four different markets, and `False` otherwise. Afterwards, you can use this helper function to *filter* the dataset.

Hint 2: If you implemented the filtering correctly, the resulting `df_subset` should have exactly 2616 rows.

In [11]:

```
df_subset = ...  # Your code here
```

Check: If you created the column correctly, the following line of code should evaluate to `True`:

In [ ]:

```
len(df_subset) == 7426
```

10. Make a new column "Profit Above Average" that is `True` when the "Profit" is above the average of the dataset, and `False` otherwise. Creating new columns can be a useful preprocessing step for further analyses or visualizations. Work with the original `df` here, select all rows where "Profit" is above average, and assign the result to the new column "Profit Above Average".

In [12]:

```
df["Profit Above Average"] = ...  # Your code here
```

Check: If you created the column correctly, the following line of code should evaluate to `True`:

In [ ]:

```
df["Profit Above Average"].sum() == 3401
```

11. Transform the `Ddate` column into a `datetime` type. The dataset contains a column `Ddate` which stores a date in month-day-year format. However, it's currently just stored as a string. In order to perform operations based on this date column (for example, grouping or filtering by month), it would be helpful to convert it to a proper Python `datetime` expression. *Transform* the `Ddate` column from `string` into a `datetime` format.

In [13]:

```
# Your code here
```

Check: If you transformed the column correctly, the following line of code should work and return `True`:

In [ ]:

```
sorted(df["Ddate"].apply(lambda x: x.month).unique()) == list(range(1, 13))
```

12. Perform min-max scaling on the `Expenses` column. Some machine learning algorithms are sensitive to the absolute distance between data points. It's therefore sometimes necessary to *reshape* your data. Here, we want to perform min-max scaling on a column, which means *transforming* our data to fall within the $[0, 1]$ range. In mathematical terms, this means applying the function:

$$f(x) = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Perform this transformation on the `Expenses` column.

Hint: Don't try to do this one on one line; you probably want to write a helper function here for clarity.

In [14]:

```
# Your code here
```

Check: If you transformed the column correctly, the following line of code should evaluate to `True`:

In [ ]:

```
df["Expenses"].aggregate([min, max]).tolist() == [0.0, 1.0] and (
    0.19 < df["Expenses"].median() < 0.21
)
```

Combining Datasets

The dataset we looked at contains data for the years 2012 and 2013. Let's assume we're getting additional data for the years 2014/2015 later, and want to merge these with the first dataset. Since we modified the original dataset a lot in the exercises above, let's first reload it here, and then load the new "extra" dataset:

In [15]:

```
df = pd.read_csv(data)
df_extra = pd.read_csv("data/coffee-chain-extra.csv")
df_extra
```

Out[15]:

	Ddate	Market	Product	Sales	Profit	Expenses
0	5/1/15	South	Caffe Mocha	231	67	54
1	2/1/15	Central	Decaf Irish Cream	319	136	85
2	7/1/14	Central	Darjeeling	251	133	59
3	5/1/14	East	Decaf Espresso	194	142	55
4	10/1/14	Central	Decaf Irish Cream	192	103	49
...	...	...	...	...	...	...
4995	4/1/14	East	Lemon	77	-8	73
4996	5/1/14	West	Decaf Espresso	105	41	20
4997	2/1/14	West	Caffe Mocha	116	-34	27
4998	2/1/14	Central	Decaf Irish Cream	119	36	42
4999	9/1/15	West	Lemon	144	28	41

5000 rows × 6 columns

13. Concatenate the two datasets `df` and `df_extra` into a DataFrame `df_new`, making sure that the new, concatenated dataset **has no duplicate indices.** The result should be a DataFrame with 15,000 rows that has a numerical index (like our original datasets) from 0 to 14,999.

In [16]:

```
df_new = ...  # Your code here
```

Check: If you created the DataFrame correctly, the following should evaluate to `True`:

In [ ]:

```
(len(df_new) == 15000) and (14999 in df_new.index)
```

Sometimes, we want to combine two datasets that have complementary information, i.e., different columns. For example, let's say we have this tiny dataset containing additional information about "Market Size" for each of our four markets:

In [17]:

```
df_markets = pd.DataFrame.from_dict(
    {
        "Market": ["Central", "East", "West", "South"],
        "Market Size": ["Major", "Major", "Major", "Minor"],
    }
)
df_markets
```

Out[17]:

	Market	Market Size
0	Central	Major
1	East	Major
2	West	Major
3	South	Minor

14. Merge `df_new` with `df_markets`. The resulting DataFrame should look just like `df_new`, but have an extra column "Market Size" whose value depends on the "Market" value of the given row.

In [18]:

```
# Your code here
```

15. Fill in missing data in the `Product Type` column. You might have noticed that the `df_extra` dataset we loaded above is missing a column that the original `df` had: the "Product Type"! Luckily, this information can be easily recovered, as the "Product Type" can be uniquely deduced from the "Product" (e.g., a "Darjeeling" product is always a "Tea"). However, the process to get there may be a little bit tricky.

Hint: First, try to get a DataFrame that has all unique `(Product, Product Type)` combinations. Then, you should be able to merge this with `df_extra`.

In [19]:

```
# Your code here
```

Slightly harder bonus challenges

16. Make a new column "Top within Product" that is `True` when the "Profit" is within the top 25% of its "Product" category.

Hints:

1. Find the cutoff value for the top 25% in each "Product" category, and store this in a variable. (A keyword to look out for when obtaining the "top 25%" is *quantiles*...)
2. Write a function that takes a Pandas Series and returns `True` if its "Profit" value is higher than the cutoff value for its "Product" category.

3. Apply (https://pandas.pydata.org/docs/user_guide/basics.html#row-or-column-wise-function-application) this function to each row of `df` to get your new column.

In [20]:

```
# Your code here
```

Check: If you created the column correctly, the following line of code should evaluate to `True`:

In [ ]:

```
df["Top within Product"].sum() == 2518
```

17. Try to do the same as in Q15, but with `df_new`. In other words, try to fill in the `NaN` values in the "Product Type" after you have already merged `df` and `df_extra` together. This can be quite tricky, and there are many possible ways to approach this.

In [21]:

```
# Your code here
```