Part 2 — Workshop 5: Concatenation and merging

TECH2: Introduction to Programming, Data, and Information Technology

Richard Foltyn
NHH Norwegian School of Economics

October 24, 2025

Exercise 1: Business cycle correlations

For this exercise, you'll be using macroeconomic data from the folder `data/FRED`.

1. There are seven decade-specific files named `FRED_monthly_YYYY.csv` where `YYYY` identifies the decade by its first year (`YYYY` takes on the values 1950, 1960, ..., 2010). Write a loop that reads in all seven files as DataFrames and store them in a list.

   Hint: Recall from the lecture that you should use `pd.read_csv(..., parse_dates=['DATE'])` to automatically parse strings stored in the `DATE` column as dates.

2. Use `pd.concat()` to concate these data sets into a single `DataFrame` and set the `DATE` column as the index.

3. You realize that your data does not include GDP since this variable is only reported at quarterly frequency. Load the GDP data from the file `GDP.csv` and merge it with your monthly data using an *inner join*.

4. You want to compute how (percent) changes of the variables in your data correlate with percent changes in GDP.

   1. Create a *new* `DataFrame` which contains the percent changes in CPI and GDP (using `pct_change()`, and the absolute changes for the remaining variables (using `diff()`).

   2. Compute the correlation of the percent changes in GDP with the (percent) changes of all other variables using `corr()`. What does the sign and magnitude of the correlation coefficient tell you?

Exercise 2: Loading many data files

In the previous exercise, you loaded the individual files by specifing an explicit list of file names. This can become tedious or infeasible if your data is spread across many files with varying file name patterns. Python offers the possibility to iterate over all files in a directory (for example, using `os.listdir()`), or to iterate over files that match a pattern, for example using `glob.glob()`.

Repeat parts (1) and (2) from the previous exercise, but now iterate over the input files using `glob.glob()`. You'll need to use a wildcard `*` and make sure to match only the relevant files in `data/FRED`, i.e., those that start with `FRED_monthly_1` or `FRED_monthly_2`.

Exercise 3: Weekly returns of the magnificent seven

In this exercise, you are asked to analyze the weekly stockmarket returns of the so-called magnificent 7 which are some of the most successful tech companies of the last decades years: Apple (AAPL), Amazon (AMZN), Alphabet/Google (GOOGL), Meta (META), Microsoft (MSFT), Nvidia (NVDA), and Tesla (TSLA).

The data for this exercise is located in the folder `data/stockmarket/`.

1. For each of the seven stocks listed above, there is a corresponding CSV file in this directory (based on the ticker symbol).
    1. For each ticker symbol, load the corresponding CSV file and make sure that the `Date` is set as the index.

       The DataFrame has two columns, `Open` and `Close`, which contain the opening and closing price for each trading day.
    2. Use `resample()` to resample the daily data to a weekly frequency by specifying `resample('W')`, and compute the weekly returns in percent:

$$\text{Weekly returns} = \frac{\text{Close price on last day} - \text{Open price on first day}}{\text{Open price on first day}} \times 100$$

       Hint: You can obtain the first and last observation using the `first()` and `last()` methods.
    3. Append these returns to a list so you can merge them into a single DataFrame later.
2. Merge the list of weekly returns you computed into a single DataFrame. Keep only the intersection of dates available for all 7 stocks.

   Hint: This can be achieved using either `pd.concat()`, `pd.merge()`, or `DataFrame.join()`.
3. Finally, you are interested in how the weekly returns are correlated across the 7 stocks.
    1. Compute and report the pairwise correlations using DataFrame.corr().
    2. Create a figure with 7-by-7 subplots showing the pairwise scatter plots of weekly returns for each combination of stocks.

       You can do this either with the `scatter_matrix()` function contained in `pandas.plotting`, or manually build the figure using Matplotlib functions.
    3. **[Advanced]** In each of the subplots, add a text that reports the pairwise correlation for these stocks which you computed earlier. (e.g., the correlation between returns on AAPL and AMZN is about 0.42, so this text should be added to the subplot showing the scatter plot of AAPL vs. AMZN).

Exercise 4: Decade averages of macro time series

For this exercise, you'll be using macroeconomic data from the folder `data/FRED`.

1. There are five files containing monthly observations on annual inflation (INFLATION), the Fed Funds rate (FEDFUNDS), the labor force participation rate (LFPART), the 1-year real interest rate (REALRATE) and the unemployment rate (UNRATE).
    1. Write a loop to import these files and store the individual DataFrames in a list.

       Hint: Recall from the lecture that you should use `pd.read_csv(..., parse_dates=['DATE'], index_col='DATE')` to automatically parse strings stored in the `DATE` column as dates and set the `DATE` column as the index.

    2. Use `pd.concat()` to concatenate this list of DataFrames along the column dimension using an outer join (`join='outer'`) to obtain a merged data set.

2. You want to compute the average value of each variable by decade, but you want to include only decades without *any* missing values for *all* variables.

    1. Create a variable `Decade` which stores the decade (1940, 1950, ...) for each observation.

        Hint: You should have set the `DATE` as the `DataFrame` index. Then you can access the calendar year using the attribute `df.index.year` which can be used to compute the decade.

    2. Create an indicator variable which takes on the value `True` whenever all observations (all columns) for a given date are non-missing, and `False` if at least one variable has a missing observation.

    3. Aggregate this indicator to decades using a `groupby()` so that the indicator takes on the value `True` whenever *all* variables in a given decade have no missing values, and `False` otherwise.

        Hint: You can use the `all()` aggregation for this.

    4. Merge this decade-level indicator data back into the original `DataFrame` (*many-to-one* merge).

3. Using this indicator, drop all observations which are in a decade with missing values.

4. Compute the decade average for each variable.

Challenge

- Your pandas guru friend claims that all the steps in 2.2 to 2.4 can be done with a single one-liner using `transform()`. Can you come up with a solution?

Exercise 5: Merging additional Titanic data

In this exercise, you'll be working with the the original Titanic data set in `titanic.csv` and additional (partly fictitious) information on passengers stored in `titanic-additional.csv`, both located in the `data/` folder.

The goal of the exercise is to calculate the survival rates by country of residence (for this exercise we restrict ourselves to the UK, so these will be England, Scotland, etc.).

1. Load the `titanic.csv` and `titanic-additional.csv` into two DataFrames.

    Inspect the columns contained in both data sets. As you can see, the original data contains the full name including the title and potentially maiden name (for married women) in a single column. The additional data contains this information in separate columns. You want to merge these data sets, but you first need to create common keys in both DataFrames.

2. Since the only common information is the name, you'll need to extract the individual name components from the original DataFrame and use these as merge keys.

    Focusing only on men (who have names that are much easier to parse), split the `Name` column into the tokens `Title`, `FirstName` and `LastName`, just like the columns in the second DataFrame.

    Hint: This is the same task as in the last exercise in Workshop 2. You can just use your solution here.

3. Merge the two data sets based on the columns `Title`, `FirstName` and `LastName` you just created using a *left join* (*one-to-one* merge). Tabulate the columns and the number of non-missing observations to make sure that merging worked.

    Note: The additional data set contains address information only for passengers from the UK, so some of these fields will be missing.

4. You are now in a position to merge the country of residence (*many-to-one* merge). Load the country data from `UK_post_codes.csv` which contains the UK post code prefix (which you can ignore), the corresponding city, and the corresponding country.

   Merge this data with your passenger data set using a *left join* (what is the correct merge key?).

5. Tabulate the number of observations by `Country`, including the number of observations with missing `Country` (these are passengers residing outside the UK).

   Finally, compute the mean survival rate by country.