Part 2 — Workshop 4

TECH2: Introduction to Programming, Data, and Information Technology

Richard Foltyn
Norwegian School of Economics (NHH)

October 25, 2024

See GitHub repository for notebooks and data:

`https://github.com/richardfoltyn/TECH2-H24`

Contents

1 Exercise: 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_19X0.csv` where `X` identifies the decade (`X` takes on the values 5, 6, 7, 8, 9, 0, 1). 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()`, see also the last exercise in workshop 3), 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?

2 Exercise: 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`.

3 Exercise: 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). Write a loop to import these and merge them on `DATE` into a single `DataFrame` using *outer joins* (recall that `merge()` and `join()` operate on only two DataFrames at a time).

   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. Your friend is a pandas guru and tells you that you don't need to iteratively merge many files but can instead directly use `pd.concat()` for merging many DataFrames in a single step. Repeat the previous part using `pd.concat()` instead, and verify that you get the same result (you can do this using `compare()`).

3. 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. Write a function `num_missing(x)` which takes as argument `x` a `Series` and returns the number of missing values in this `Series`.

   3. Compute the number of missing values by decade for each variable using a `groupby()` operation and the function `num_missing` you wrote.

   4. Aggregate this data across all variables to create an indicator for each decade whether there are any missing values. This can be done in many ways but will require aggregation across columns, e.g., with `sum(..., axis=1)`.

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

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

5. Compute the decade average for each variable.

Challenge

- Your pandas guru friend claims that all the steps in 3.2 to 3.5 can be done with a single one-liner using `transform()`. Can you come up with a solution?

4 Exercise: Mering the 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.