Part 2 — Workshop 2: Introduction to pandas

TECH2: Introduction to Programming, Data, and Information Technology

Richard Foltyn
NHH Norwegian School of Economics

October 3, 2025

Exercise 1: Data cleaning

Before doing actual data analysis, we usually first need to clean the data. This might involve steps such as dealing with missing values and encoding categorical variables as integers. In this exercise, you will perform such steps based on the Titanic passenger data.

1. Load the Titanic data set in `titanic.csv` located in the `data/` folder.
2. Report the number of observations with missing `Age`, for example using `isna()`.
3. Compute the average age in the data set. Use the following approaches and compare your results:
    1. Use pandas's `mean()` method.
    2. Convert the `Age` column to a NumPy array using `to_numpy()`. Experiment with NumPy's `np.mean()` and `np.nanmean()` to see if you obtain the same results.
4. Replace the all missing ages with the mean age you computed above, rounded to the nearest integer. Note that in "real" applications, replacing missing values with sample means is usually not a good idea.
5. Convert this updated `Age` column to integer type using `astype()`.
6. Generate a new column `Female` which takes on the value one if `Sex` is equal to `"female"` and zero otherwise. This is called an *indicator* or *dummy* variable, and is preferrable to storing such categorical data as strings. Delete the original column `Sex`.
7. Save your cleaned data set as `titanic-clean.csv` using `to_csv()` with `,` as the field separator. Tell `to_csv()` to *not* write the `DataFrame` index to the CSV file as it's not needed in this example.

Exercise 2: Selecting subsets of data

In this exercise, you are asked to select subsets of macroeconomic data for the United States based on some criteria.

1. Load the annual data from FRED which are located in `FRED_annual.xlsx` in the `data/FRED` folder.
2. Print the list of columns and the number of non-missing observations.
3. Since we are dealing with time series data, set the column `Year` as the DataFrame index.
4. Print all observations for the 1960s decade using at least two different methods.
5. Using the data in the column `GDP`, compute the annual GDP growth in percent and store it in the column `GDP_growth`. Select the years in which
    1. GDP growth was above 5%.

2. GDP growth was negative, but inflation as still above 5% (such episodes are called "stagflation" since usually negative GDP growth is associated with low inflation).

   Use at least two methods to select such years.

Hint: You can compute changes relative to the previous observation using the `pct_change()` method.

Exercise 3: Labor market statistics for the US

In this exercise, you are asked to compute some descriptive statistics for the unemployment rate and the labor force participation (the fraction of the working-age population in the labor force, i.e., individuals who are either employed or unemployed) for the United States.

1. Load the monthly time series from FRED which are located in `FRED_monthly.csv` in the `data/FRED` folder.

   Hint: You can use `pd.read_csv(..., parse_dates=['DATE'])` to automatically parse strings stored in the `DATE` column as dates.

2. Print the list of columns and the number of non-missing observations.

3. Since we are dealing with time series data, set the column `DATE` as the DataFrame index. Using the date index, select all observations from the first three months of the year 2020.

4. For the columns `UNRATE` (unemployment rate) and `LFPART` (labor force participation), compute and report the mean, minimum and maximum values for the whole sample. Round your results to one decimal digit.

   Hint: You can use the DataFrame methods `mean()`, `min()`, and `max()` to compute the desired statistics.

   Hint: You can use the DataFrame method `round()` to truncate the number of decimal digits.

5. You are interested in how the average unemployment rate evolved over the last few decades.

   - Add a new column `Decade` to the DataFrame which contains the starting year for each decade (e.g., this value should be 1950 for the years 1950-1959, and so on).

     Hint: The decade can be computed from the column `Year` using truncated integer division:

     ```
     df['Year'] // 10 * 10
     ```

   - Write a loop to compute and report the average unemployment rate (column `UNRATE`) for each decade.

     Include only the decades from 1950 to 2010 for which you have all observations.

Exercise 4: Working with string data (advanced)

Most of the data we deal with contain strings, i.e., text data (names, addresses, etc.). Often, such data is not in the format needed for analysis, and we have to perform additional string manipulation to extract the exact data we need. This can be achieved using the pandas string methods.

To illustrate, we use the Titanic data set for this exercise.

1. Load the Titanic data and restrict the sample to men. (This simplifies the task. Women in this data set have much more complicated names as they contain both their husband's and their maiden name)

2. Print the first five observations of the `Name` column. As you can see, the data is stored in the format *"Last name, Title First name"* where title is something like Mr., Rev., etc.

3. Split the `Name` column by `,` to extract the last name and the remainder as separate columns. You can achieve this using the `partition()` string method.
4. Split the remainder (containing the title and first name) using the space character `" "` as separator to obtain individual columns for the title and the first name.
5. Store the three data series in the original `DataFrame` (using the column names `FirstName`, `LastName` and `Title`) and delete the `Name` column which is no longer needed.
6. Finally, extract the ship deck from the values in `Cabin`. The ship deck is the first character in the string stored in `Cabin` (A, B, C, ...). You extract the first character using the `get()` string method. Store the result in the column `Deck`.

Hint: Pandas's string methods can be accessed using the `.str` attribute. For example, to partition values in the column `Name`, you need to use

```
df['Name'].str.partition()
```