Web Applications with Shiny

Solve the following exercises and upload your solutions to Moodle by the due date.

Important Information!

Please try to *exactly match the outputs* provided in the examples.

The apps in this assignment have to be submitted differently than in other assignments! For each of the apps create a directory (e.g. `a4_ex1`) which contains your app script, that has the name `app.py`. This is so you can run it most easily directly in VS Code as it automatically then recognizes the script as an app (make sure you have the Shiny extension by Posit.co installed). You can also run the app from the terminal with this command: `shiny run --reload --launch-browser --port=0 app.py`. The exact directory names/file names are shown below as usual. The two directories have to be compressed into a file `a4.zip` which you can upload to Moodle.

Use Shiny Express to solve the exercises, do not use Shiny Core. There are screenshots so you can see how the app should behave, but there are also demo videos on Moodle.

Unless explicitly stated otherwise, you can assume correct user input and correct arguments. You may use **only standard libraries**, plus modules covered in the lecture (including Programming in Python 1).

Exercise 1 – Submission: `a4_ex1/app.py` 60 Points

Create a web app with the title **Data Cleaner** that enables simple data pre-processing. The app should have the following capabilities:

- Allow users to upload a .csv file containing any number of columns (for initial state of the app check Figure 1). The uploaded data should be shown as a data frame (check Figure 2).
- After uploading, users can click an "Analyze" button to display information about the data set (check Figure 3). The results should be shown in a table (data frame) sorted by number of missing values in descending order.
- Users should be able to select and remove specific columns from the dataset using a multi-select dropdown menu.
- Users can choose how to handle missing values:
  - No change (selected default)
  - Replace with 0
  - Replace with column mean (only for numeric columns)
  - Replace with column median (only for numeric columns)
  - Drop rows with missing values
- Users can choose to transform the numerical variables and either apply normalization (between 0 and 1) or standardization (0 mean and unit variance).

- When the user clicks the "Clean" button, the selected operations should be applied to the dataset and the display should update (order of execution: dropping of columns → handling of missing values → transformations of columns).
- Check Figure 4 to see how the dataset should look like after the clicking "Clean".
- Allow users to download the cleaned data as a .csv file (file name: `cleaned_data.csv`).
- The "Reset" button should restore the original uploaded data, clear all modifications and reset the user interface.
- There should be a dark mode switcher.
- Use a sidebar layout that contains a sidebar and a pill navset (that in turn contains the two navpanels for "Data" and "Analysis").

A few additional important points:

- Use reactive values to manage the internal state of the data (raw data and cleaned data).
- Use pandas for data manipulation.
- Use `BytesIO` to create the downloadable .csv file.
- In general, ensure that your app does not produce unexpected behavior or crashes. For example:
  - If anything goes wrong with the file upload (e.g. wrong file type), the app should display an error message (check out `ui.notification_show()`) with the text "Error loading file: *error_message*" for a duration of 5 seconds. The app should continue to run and the user can then try another file.
  - "Reset" and "Clean" should do nothing if no data set is loaded.
  - "Clean" should always use the modified data. So for example, it should be possible to delete one column first and delete another one with a second click.
  - "Replace with mean" and "Replace with median" should only do something if there are in fact numerical columns.
  - In the "Columns to transform" selector, only numerical columns should be shown.
  - Ensure that columns that should be removed actually exist in the table.
  - If no data is uploaded, the download button should produce a file with an empty data frame.
  - It's not necessary to dynamically determine the columns that can be transformed. So for example, if there is a column that is supposed to be transformed, but that very same column is also dropped in the same step, then the deletion is done (as it is first in the execution order), and the transformation is just ignored for that column.

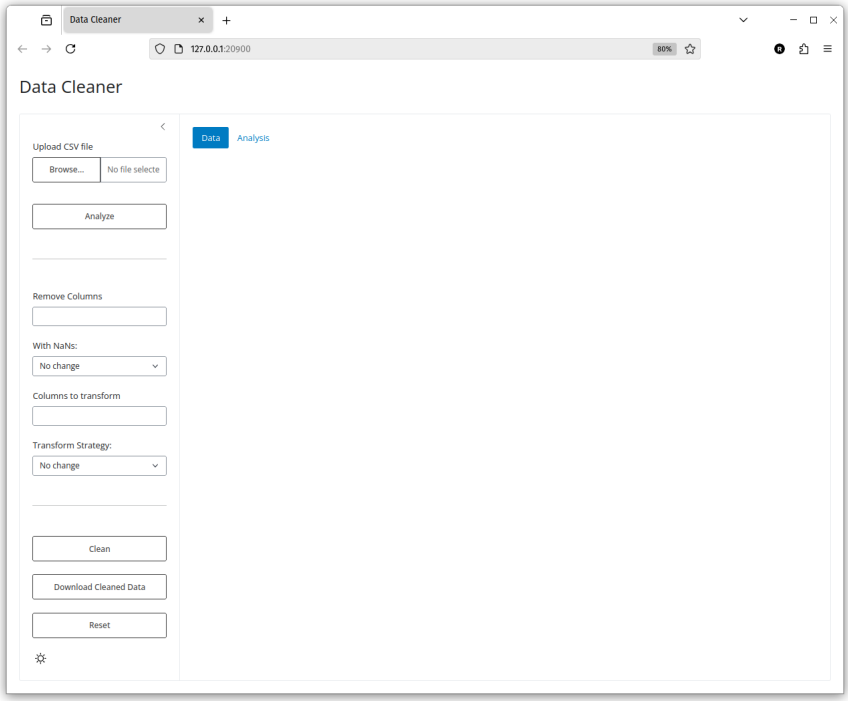

Figure 1: Initial state

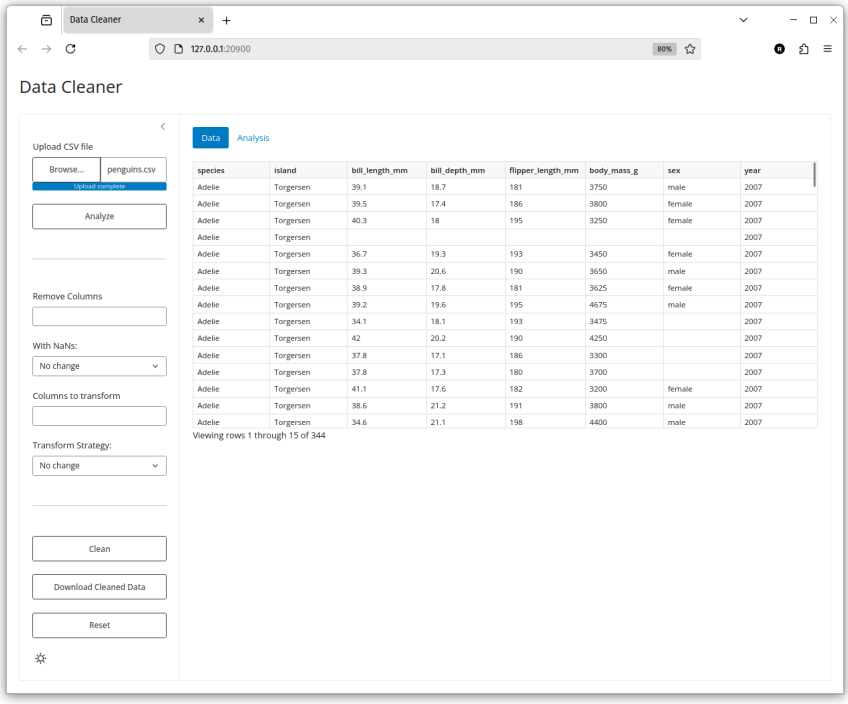

Figure 2: After uploading the data

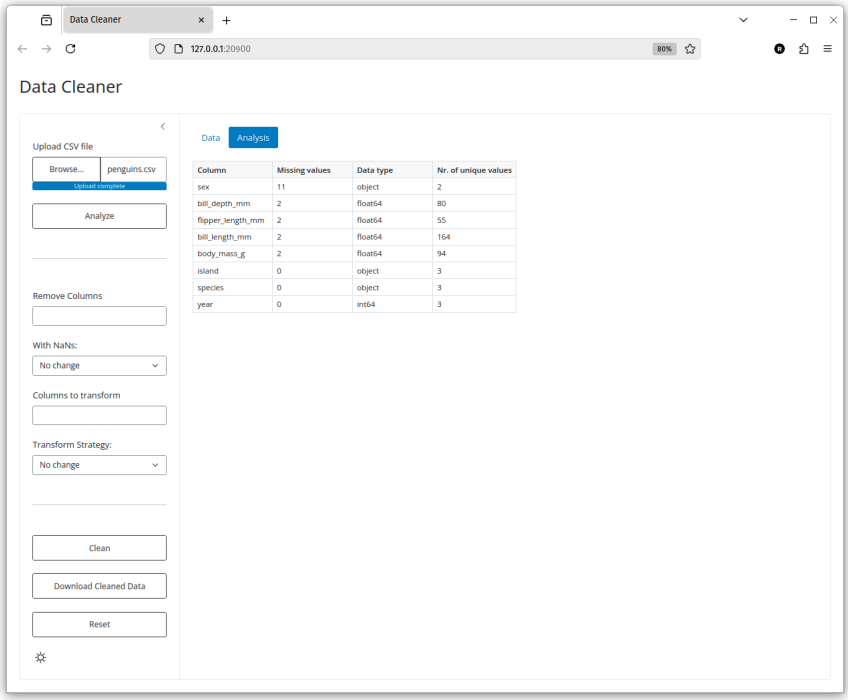

Figure 3: After clicking "Analyze" in the analysis tab

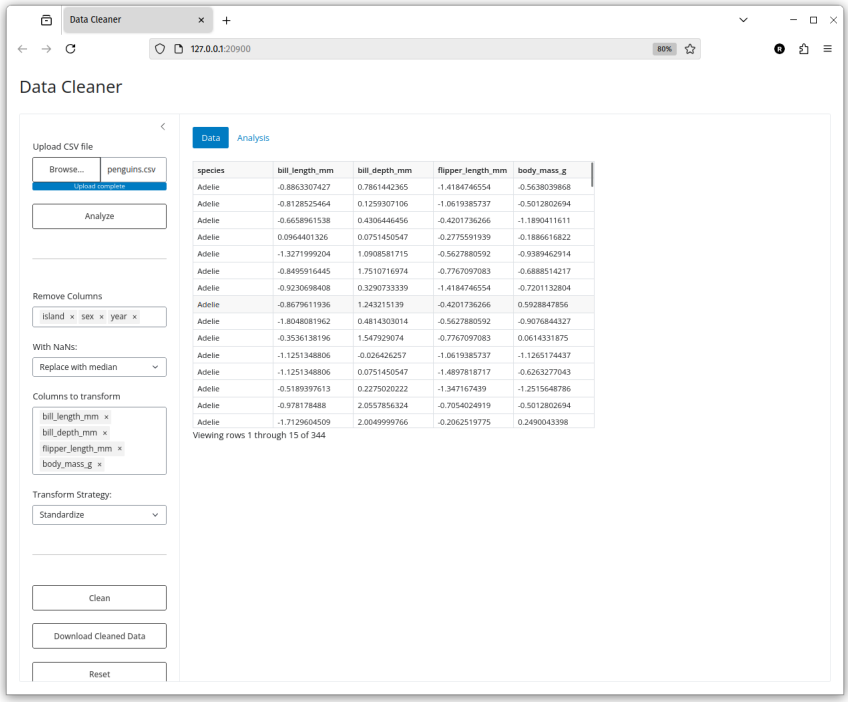

Figure 4: After applying modifications

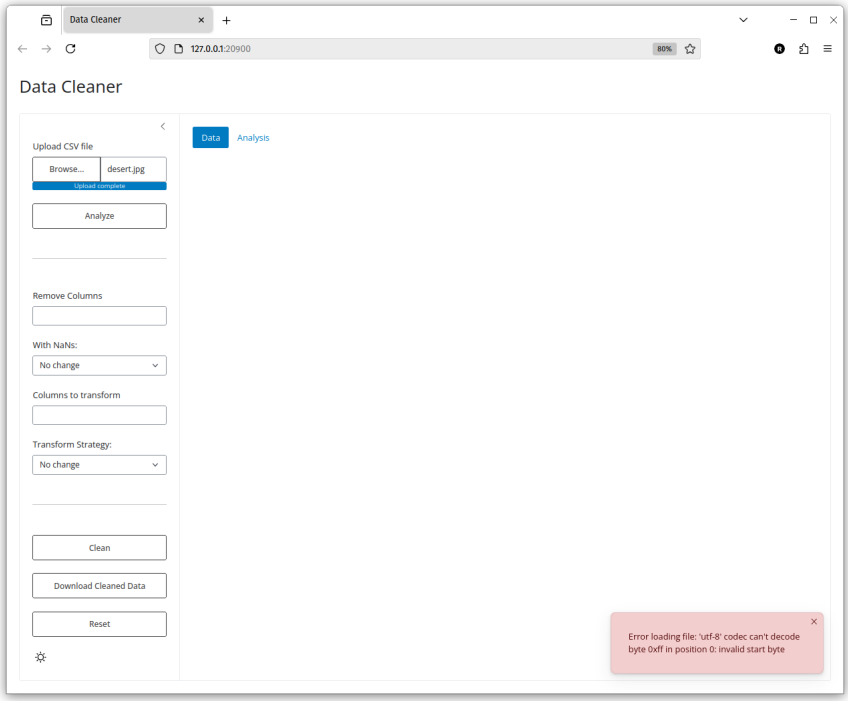

Figure 5: Trying to upload a jpg file

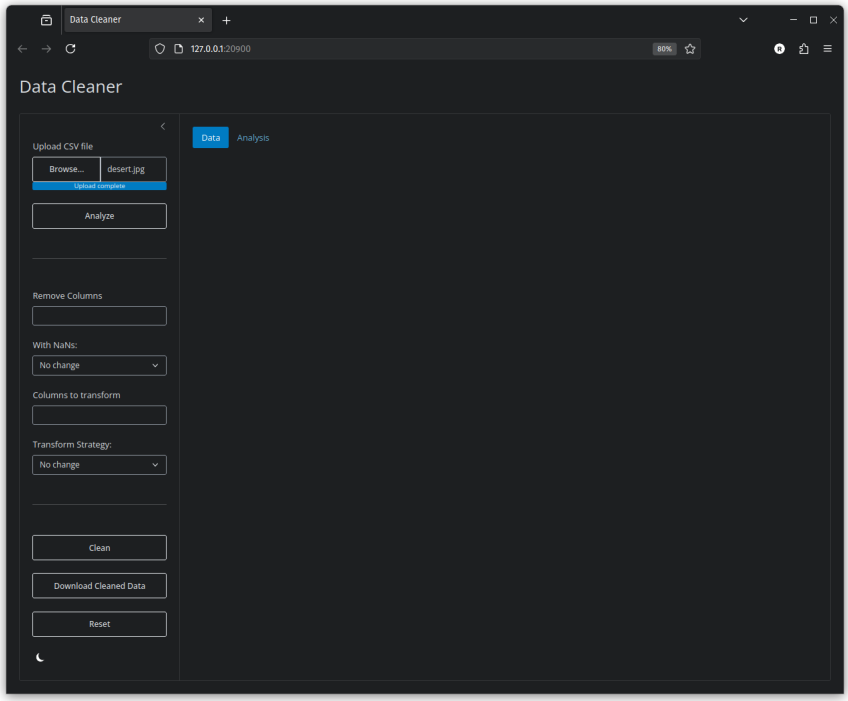

Figure 6: Dark mode

Exercise 2 – Submission: `a4_ex2/app.py` 40 Points

Create an app called **CO2 Dashboard** that allows to analyze CO2 emissions for the countries of the world. The app should work as follows:

- Use a navset bar to show the two navpanels for the time series analysis and the world map. Within the navpanels, use a sidebar layout to arrange the controls in the side bar and the plots on the main pane.
- The data for the app is available at:
  `https://raw.githubusercontent.com/owid/co2-data/master/owid-co2-data.csv`
- Pre-process the data as follows:
  - Only the columns `"country", "iso_code", "year", "co2"` are needed.
  - Coerce `co2` to numeric (in case there are stray non-numeric values).
  - Drop any rows where `iso_code` or `co2` is `NaN` (parameter `subset`).
  - Filter the data: we only want rows in the final data where `co2 > 0` and the length of the `iso_code` is exactly 3.
- In the time series tab, the user should be able to select a country from the dropdown menu. This will show the CO2 emissions of this country over the years as a `plotly.express` line plot. In addition, a smoothed line should be displayed (rolling mean over a window that is set by the slider). Also consider:
  - The selector should show all countries from the data set (ensure that the list only contains unique items), with Austria as default selection.
  - The slider should go from 1 to 20 with 5 as default.
  - Make sure the x/y axis labels, plot title and legend look as in Figure 7.
  - Hover mode should be `x unified` (check out `update_layout()`).
- In the world map tab the user should be able to set a year with the slider and the map should show the map where the user can zoom in and out. By hovering with the mouse over a country, the name of the country, its `iso_code` and the `co2` value should be shown (check Figure 8). The map should be color coded according to the intensity of the CO2 emissions. Also note:
  - For the map, check out `choropleth()` (from `plotly.express`). Color scale should be `Reds`. The primary information when hovering should be the country name.
  - Set the top margin for the plot to 40, so that the title appears correctly.

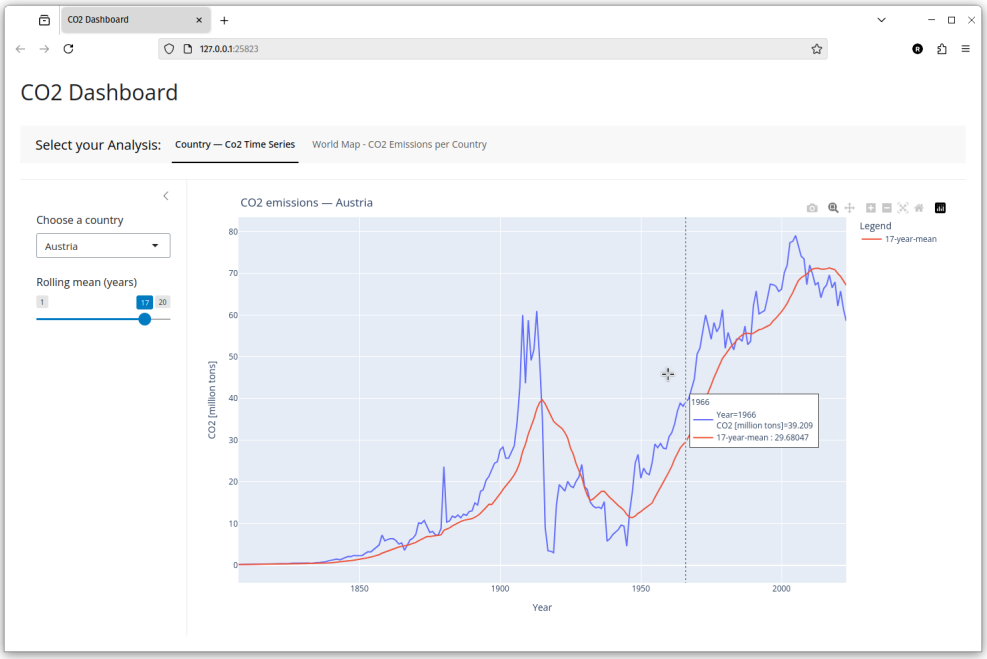

Figure 7: Time series tab

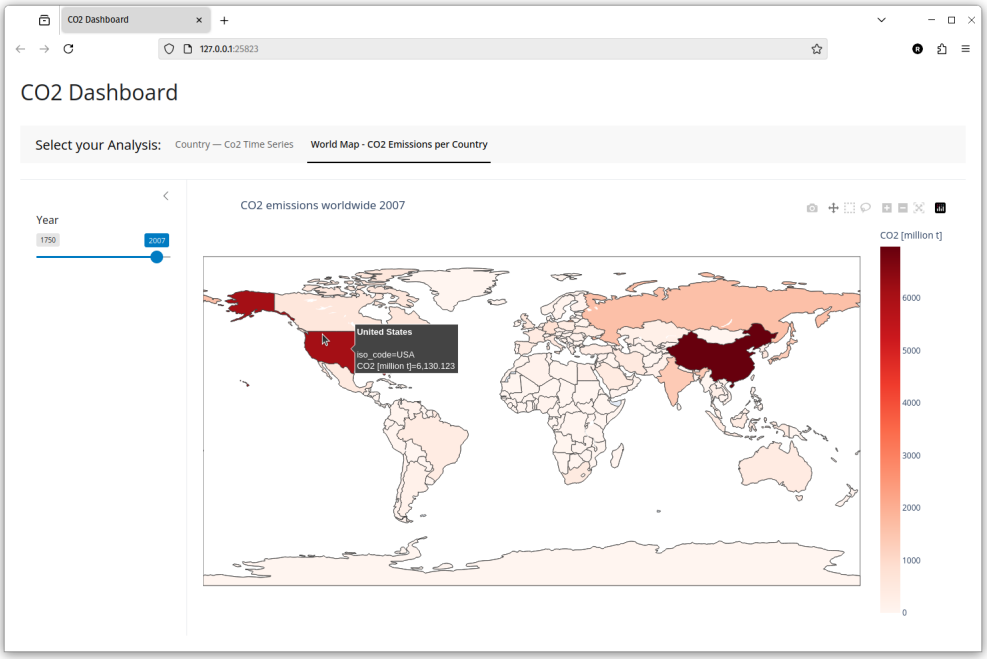

Figure 8: World map tab