Problemset

General instructions

- Carry out an exploratory data analysis of the Titanic dataset, and put your project under version control. **The focus should be on Git-related aspects and not on the analysis.**
- **Commit your changes frequently.** In real life you would not commit every single line of code, but rather group changes that belong together into a single commit. However, for the purpose of this problemset, commit often to get a better understanding of the Git workflow.
- **Experiment with both the command line and the VS Code Git interface.** For instance, you could first carry out the exercises using only the VS Code interface. In a second step, you could repeat the entire problemset using the command line. While the VS Code interface is more user-friendly, the command line will help you understand the underlying Git commands better (and it is also more powerful).

Exercise 1: Standard Workflow

- Create a new project folder for an exploratory analysis of the Titanic dataset and open it in VS Code.
- Initialise this folder as a Git repository.
- Add 3 files to your working directory: Copy the dataset `titanic.csv` into the folder, (2) Create a file `readme.md` and add a project title and a short project description to this file. (3) Create a Python script `titanic-eda.py` that imports pandas and reads in the dataset. Check how the status of the repository changes when you add these files to the repository.
- Commit these changes to the repository. Make sure that a commit only contains changes related to a single task. Add a commit message that describes the changes made in the commit. Reflect: what is the role of the staging area?

Exercise 2: Undo changes

- Carry out a series of EDA steps to the Python script and commit each step separately. For instance, check the first rows of the dataset, the column data types, and the number of missing values.
- Now simulate a situation where you need to undo recent changes: delete some lines of code and commit your changes. Undo this action in such a way that the commit history is preserved.
- Repeat the previous exercise, but this time undo the changes in a way that the commit history is rewritten.
- Delete the readme file and commit this change. Undo this action via the command `git reset --mixed HEAD~1`. Why does the readme file not reappear in the working directory? What do you need to do to get the readme file back?

Exercise 3: Branching and merging

- Create a branch called "git-learnings" and switch to it. Update your readme file, by summarizing the most interesting or surprising things you have learned about Git so far. Also add a subsection where

you note questions you still have about Git. Commit your changes to the git-learnings branch.
- Switch to the main branch and verify that the main branch does not yet contain the changes you made in the git-learnings branch. Merge the git-learnings branch into the main branch. Then delete the git-learnings branch.

Exercise 4: Merge conflicts

- Create two branches from the main branch: `price-analysis` and `age-analysis`.
- Add some changes to the Python script in the `price-analysis` branch and commit your changes. For instance, calculate the average ticket price. Then merge the `price-analysis` branch into the main branch.
- Add some changes to the Python script in the `age-analysis` branch and commit your changes. For instance, calculate the average age of the passengers. Then merge the `age-analysis` branch into the main branch.
- Why does a conflict occur? How can you resolve this conflict?
- Install the VS Code extension "Git Graph" and study the commit graph of your repository. Alternatively, run `git log --graph --oneline --all`

Exercise 5: Handling different file types

- Create a new branch called `filetypes` and switch to this branch.
- Add an empty Word document (`readme.docx`) and commit your change. The n copy some content from the readme markdown file to the corresponding Word document. Commit this change. How does Git handle the Word document? What is the difference between tracking changes in a markdown file and a Word document?
- Add an empty Jupyter Notebook `titanic-eda.ipynb` and commit your change. Then copy some contents from the Python script to the Jupyter Notebook and execute the code cell(s). Commit this change. How does Git handle the Jupyter Notebook? What is the difference between Python scripts and Jupyter Notebooks in terms of version control?
- Add an empty text file to your repository and call it .gitignore. What is the purpose of the .gitignore file? Then add the following line to the .gitignore file: `*.docx`. Does this line have the desired effect? Why or why not? Test whether this line has the desired effect if you add a second Word document to the repository. What is the difference to the previous situation?
- Merge the `filetypes` branch into the main branch and afterwards delete it