Session 2B Principles of FOSS and Version Control
Exercise 3 – Git Branches and Merging

For this exercise, you can work individually or as a group, but either way use your support group to ask questions if you get stuck. Refer back to the lecture slides to help you.

- If you're using Windows, you'll need to launch the Git Bash program before starting
- If you're using Linux or Mac OS, you should just need to open a new terminal (assuming Git is installed)

1. Using the repository you set up earlier, rename your *master* branch to another name (we recommend *main* as this is consistent with the default name used on GitHub)

2. Create a new branch called *dev*, and use the *git branch* command to view your branches, as well as the currently active branch

3. Switch to the new *dev* branch

4. Use a *restore* to restore the code from an earlier commit

5. Make some changes to this code. Then save it, stage the changes and commit.

6. Have a look at the log to see the history of commits

7. Merge the changes you have made on *dev* into your *main* branch. Make sure you are in the correct branch before doing this.

8. Have another look at the log to view the commit history again.

9. Switch over to Spyder (ensuring you're in the *main* branch) and confirm the new code is present

10. Merge *main* back into *dev*. Again, make sure you are in the correct branch before doing this.

11. Make a change to some of the lines of code of one of your code files. Then save, stage and commit the changes.

12. Switch back to *main* and make some different changes to the same lines of code in the same file. Save, stage and commit the changes.

13. Try merging the *dev* branch into *main*. You should be informed that there is a conflict. Use a *git status* command to find out in which file(s) the conflict(s) resides.

14. Switch back to Spyder, and you should find that Git has marked the conflict in your code file. Keep the changes that you made in the *dev* branch, and save the file.

15. Stage and commit the changes, and then have a look at the log once again.

16. In the folder of your repository, create a new .csv file called *input_data.csv*, as well as a folder called *raw_data,* which contains another new .csv file called *raw.csv*.

17. Check the status of your repository and confirm that the new files are acknowledged but not being tracked.

18. Create, stage and commit a .gitignore file that tells Git to ignore the input_data.csv file in your repository, as well as anything in the folder *raw_data*.

19. Check the status of your repository again and confirm that the new files are no longer being acknowledged (you should have a clean working tree).

20. Create, stage and commit a fictitious *readme.md* file for your repository.