COMP0034 Coursework 2

In this coursework, I created a dashboard application that uses Dash and the Estates Management Record from HESA (see references). To run the app:

1. Fork this repository: https://github.com/ucl-comp0035/comp0034-cw2i-4jjnaomi
2. Clone the resulting repository locally and to your IDE
3. Create and activate a virtual environment
4. Install the requirements using pip install -r requirements.txt
5. Run the app by using your IDE to run the app.py file in the source folder
6. Open a browser and go to http://127.0.0.1:8051/
7. Go to the various URLs outlined in List of URLs below
8. Stop the app using CTRL+C
9. Run tests using pytest -v or looks at the Github Actions workflows to see previous runs of tests

Application code

The code that creates the dashboard is found in the src.app module of the repo. This dash app can be run in development mode from the command line of a terminal. The URLs for each route can be seen in Table 1.

List of URLs

Table 1 Available urls for dash app and explanation of contents

URL	Explanation
/	Homepage – landing page for user with navigation to all other pages provided and a map of all HEs in England.
/ranking_table	Ranking table of all HEs in the database of various metrics within classes. The user can choose which metrics they'd like to see
/university/<he_name>	Variable route where each university in the database has an overview page allowing the user to analyse that HE's data specifically
/comparison	Users can select a subset of HEs to compare using the bar charts. They can choose which metrics are shown on the bar chart.

Homepage features

The homepage is intended to be the first page that the user sees when they access the app. The page has a navigation bar, some content and a footer. All pages in the app have a navigation bar and a footer. The homepage has three buttons leading to the different pages in the app and the buttons contain a description of the page they lead to.

The key feature of the homepage is an map of Higher Education Institutions (HEIs) in England. Although, efforts were made to show all the HEIs in the database on this map, getting the geo data was not possible for all of them. In the data folder, there is a file called get_lat_lon.py. The original

dataset did not contain latitude and longitude data so a geocoder was needed to do this. In the initial attempt of getting the geodata for each university, the geocoder was not able to get their data. Therefore, alternative university names were assigned to each university where the geocoder wasn't working to improve its success. As mentioned above, the geocoder still was not able to get the data for all the HEIs and some HEIs are mapped at the same point despite being different universities. These universities with geodata problems are: University of London (Institutes and Activities), The University of Northampton, University of St Mark and St John, SOAS University of London and Conservatoire for Dance and Drama). University of London (Institutes and Activities) and SOAS University of London have the exact same latitude and latitude according to the geocoder so their markers on the map overlap. The other three universities have no latitude and longitude data so they don't have markers on the map. If I had more time, I would manually edit these data points so that every university is on the map and has unique geodata.

On the map, when a marker for a university is hovered on, a card is displayed with some key metrics for that University. The name of the university on the card also contains a hyperlink to the overview page for the HEI. The map can be filtered to only show the HEIs within a specific region of England. When a region is selected, that also limits the options for the HEIs that can be selected in the dropdown below. The HEIs shown on the map can also be filtered using the HEI dropdown. This functionality has been implemented using callbacks. I had to use callback contexts (triggered module and prop_id) to simplify the updating the map based on either region or HEIs depending on the user input

Ranking table page features

The ranking table page contains a description of the page, some instructions, a few dropdowns, and the table itself. The values in the 'HE Provider' Column contains a hyperlink leading to the overview of the HE. This was important to me because I wanted my page to be easily navigated.

HEI Overview page

The HEI overview page provides the user with the ability to show data about a specific HEI. This page has a variable route - /university/<he_name>. If the user tries to navigate to a he_name that isn't in the database such as /university/nonexistentuni, then a message is shown to the user to let them know this. I initially wanted to use a 404 page not found error message initially, but I felt like the message shown on the page and allowing the user to still be able access the sidebar would be more useful. The side bar can be toggled using the 'Choose a HEI' button and the user can search through the links in the sidebar to quickly access the HEI they'd like.

To view a line chart of the HEI's environmental performance over four academic years, the user must first choose a class using the dropdown. This will determine the category markers available in the second dropdown. The category marker should then be chosen to obtain the line graph.

For all my graphs in the app, I have used a pastel colour palette in order to have a cohesive and visually appealing theme.

HEI Comparison page

The /comparison page gives the user the ability to compare different HEIs in specific metrics. The instructions at the top give the user guidance on what to do. Similar to other pages, the choices made in one dropdown, then influence the available options in a subsequent dropdown. This was done using callbacks.

Test code

Evidence of tests

As shown in the Figure 2, there were 21 tests and they all passed. The tests can be seen in the tests directory. The tests use parameterisation to access multiple URLs in the navbar. The tests also use multiple fixtures shown in the conftest.py. The main exception in my application code is the 'PreventUpdate' exception. It was not suitable to use patching or simulate this exception as my main use of this exception was to prevent the update of charts if a value wasn't select specific dropdowns otherwise, an error would be shown on the screen. I tested that this exception was being raised by passing empty strings to these dropdowns. An example of this can be seen in the test 'test_comparison_update_category_marker_no_class' in test_comparison.py file.

```
13
14 tests/test_app.py::test_server_live PASSED                                    [  4%]
15 tests/test_app.py::test_navbar_links[0-/] PASSED                             [  9%]
16 tests/test_app.py::test_navbar_links[1-/ranking_table] PASSED                 [ 14%]
17 tests/test_app.py::test_navbar_links[2-/comparison] PASSED                    [ 19%]
18 tests/test_app.py::test_404_page PASSED                                      [ 23%]
19 tests/test_comparison.py::test_comparison_page_layout PASSED                 [ 28%]
20 tests/test_comparison.py::test_comparison_page_callback PASSED               [ 33%]
21 tests/test_comparison.py::test_comparison_update_category_marker_no_class PASSED [ 38%]
22 tests/test_homepage.py::test_homepage_content PASSED                         [ 42%]
23 tests/test_homepage.py::test_map_marker_select_updates_card PASSED           [ 47%]
24 tests/test_homepage.py::test_region_dropdown_map_updates PASSED              [ 52%]
25 tests/test_homepage.py::test_map_card_link_opens PASSED                      [ 57%]
26 tests/test_overview.py::test_overview_page_layout PASSED                     [ 61%]
27 tests/test_overview.py::test_overview_update_line_chart PASSED               [ 66%]
28 tests/test_overview.py::test_toggle_sidebar_button PASSED                    [ 71%]
29 tests/test_overview.py::test_sidebar_search PASSED                           [ 76%]
30 tests/test_overview.py::test_sidebar_link PASSED                             [ 80%]
31 tests/test_overview.py::test_non_existent_university_overview PASSED         [ 85%]
32 tests/test_overview.py::test_update_category_marker_dropdown_no_class_name PASSED [ 90%]
33 tests/test_ranking_table.py::test_ranking_table_layout PASSED                [ 95%]
34 tests/test_ranking_table.py::test_ranking_table_callback PASSED              [100%]
35
```

Figure 1 Evidence of tests being run for dash app

Coverage

Figure 2 shows the output from the pytest coverage. This shows that I achieved 96% coverage which is good for an app with this amount of code.

```
---------- coverage: platform win32, python 3.12.2-final-0 -----------
Name                          Stmts   Miss  Cover   Missing
-----------------------------------------------------------
src\__init__.py                   0      0   100%
src\app.py                       13      1    92%   84
src\figures.py                  122      5    96%   99, 157, 225, 275, 405
src\pages\__init__.py             0      0   100%
src\pages\comparison.py          44      1    98%   70
src\pages\homepage.py            56      2    96%   189-190
src\pages\overview.py            58      2    97%   215-216
src\pages\ranking_table.py       17      0   100%
-----------------------------------------------------------
TOTAL                           310     11    96%
```

Figure 2 Coverage report for tests run for dash app

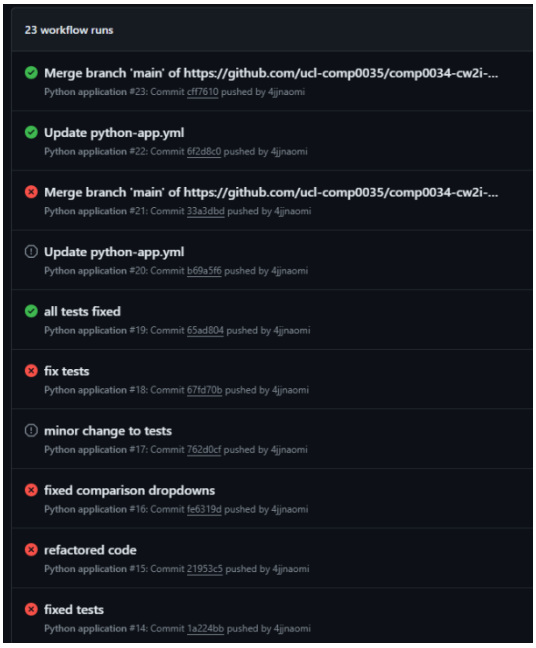

Figure 3 Evidence of GitHub Actions workflow being used

Tools and techniques

Github repository: https://github.com/ucl-comp0035/comp0034-cw2i-4jjnaomi

Continuous integration

This development of this app used a GitHub Actions workflow so that testing was done whenever changes were made to the repository. This meant that I was able to fix issues quickly. Evidence of the workflow being used is shown in Figure 3.

Linting

In writing the code, a linter was used within Visual Studio Code to ensure good code quality through the identification and reporting of potential errors.

In an early commit of the code, there were more than 200 code quality issues. This can be seen by going to the GitHub Actions and looking at the linter tab of workflow 11 or 12. The autopep 8 linter was used to automatically format my files once done to adhere to good code quality. This was able to fix most of the code quality issues but there were still some that had to be fixed manually such as lines being too long. The final linter results are shown in Figure 4.

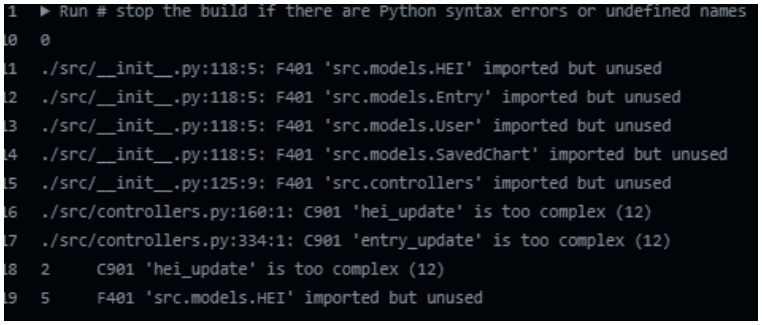

Figure 4 Code quality issues according to flake8 linter in GitHub actions workflow in final commit before submission

References

Acknowledgement of the use of AI

In the development of the dashboard, AI was used to generate, refactor and debug the code used for the application and the tests. Github Copilot v1.159.0 (GitHub, https://github.com/features/copilot) was downloaded as an extension in Visual Studio Code (the IDE I was using) and was able to automatically generate code based on the code already in my files and modules. Copilot was used to generate application code and test code as well as generate comments and doc strings for some functions. Copilot did not influence my code, the AI was influenced by the code I already had.

ChatGPT-3.5 (Open AI, https://chat.openai.com/) was also used for debugging and refactoring of code. ChatGPT influenced the quality of my code as it was used to improve its adherence to DRY and good code design principles. For example, I gave ChatGPT the code in my homepage.py file and gave this prompt: **Refactor this code so that the functions are shorter and the code adheres to DRY**. ChatGPT was able to refactor the code so that they were more efficient and put the relevant lines of code in functions to improve their readability. Some of the code was however not correct so I did have to manually fix them.

For example, my code for create_ranking table() function was initially:

```
def create_ranking_table(ClassName=None,
acedemic_year=None,            selected_regions=None):
    # Load the dataset
    data_path =
Path(__file__).parent.parent.joinpath('data','dataset_prepared.csv')
    data_df = pd.read_csv(data_path)
    cols = ['HE Provider','Region of HE provider', 'Academic Year', 'Class',
'Category', 'Value']
    data_df = data_df[cols]

    # Filter the DataFrame by 'Class' and 'Academic Year'
    data_df = data_df[(data_df['Class'] == ClassName) & (data_df['Academic
Year'] == acedemic_year)]

    if selected_regions:
        data_df = data_df[data_df['Region of HE
provider'].isin(selected_regions)]

    # Convert 'Value' column to numeric, ignoring errors
    data_df['Value'] = pd.to_numeric(data_df['Value'], errors='coerce')

    category_order = data_df['Category'].unique().tolist()
    new_category_order = list(filter(lambda x: x != 'Environmental management
system external verification', category_order))

    # Pivot the DataFrame to have categories as columns
    pivot_df = data_df.pivot_table(index='HE Provider', columns='Category',
values='Value').reset_index()

    pivot_df = pivot_df[['HE Provider'] + new_category_order]

    # Reset index
    pivot_df.reset_index(drop=True, inplace=True)

    # Rename columns
    pivot_df.columns.name = None

    pivot_df['HE Provider'] = pivot_df.apply(lambda row: f"<a
href=/university/{quote(row['HE Provider'])}>{row['HE Provider']}</a>",
axis=1)

    # Converting to DataTable with sorting enabled
    table = dash_table.DataTable(
        id='ranking-table',
        columns=[{'name': col, 'id': col, 'presentation': "markdown"} for col
in pivot_df.columns],
        data=pivot_df.to_dict('records'),
        style_table={'overflowX': 'auto'},
```

Using ChatGpt it refactored the code to use the function load_data() (which is used in other functions also) and also used the filter_data_for_tables function().

```
def create_ranking_table(ClassName=None, academic_year=None,
selected_regions=None):
    """
    Create a ranking table for HE providers based on the given parameters.

    Args:
        ClassName (str, optional): The class name to filter the data. Defaults
to None.
        academic_year (str, optional): The academic year to filter the data.
Defaults to None.
        selected_regions (list, optional): The list of regions to filter the
data. Defaults to None.

    Returns:
        dash_table.DataTable: The ranking table as a Dash DataTable
        object.
    """
    data_path = Path(__file__).parent.parent.joinpath(
        'data', 'dataset_prepared.csv')
    data_df = load_data(data_path, [
                        'HE Provider', 'Region of HE provider', 'Academic
Year', 'Class', 'Category', 'Value'])
    # Filter data based on the given parameters
    data_df = filter_data_for_table(
        data_df, ClassName, academic_year, selected_regions)
    data_df['Value'] = pd.to_numeric(data_df['Value'], errors='coerce')
    # Pivot the data to create the ranking table
    category_order = data_df['Category'].unique().tolist()
    new_category_order = list(filter(
        lambda x: x != 'Environmental management system external
verification', category_order))
    pivot_df = data_df.pivot_table(index='HE Provider', columns='Category',
values='Value').reset_index()[
        ['HE Provider'] + new_category_order]
    # Sort the columns based on the category order
    pivot_df.columns.name = None
    # Change the HE Provider column to a hyperlink in html format
    pivot_df['HE Provider'] = pivot_df['HE Provider'].apply(
        lambda x: f"<a href=/university/{quote(x)}>{x}</a>")
    # Create the ranking table
    table = dash_table.DataTable(
        id='ranking-table',
        columns=[{'name': col, 'id': col, 'presentation': "markdown"}
                 for col in pivot_df.columns],
        data=pivot_df.to_dict('records'),
        style_table={'overflowX': 'auto'},
        style_header={
            'backgroundColor': 'rgb(204, 255, 221)', 'fontWeight': 'bold'},
```

ChatGPT was also used to generate fixtures for my tests. I was initially repeating a lot of the code in my tests, but asked ChatGPT to **suggest some pytest fixtures based on the code I am repeating a lot** and it helped me to generate the dash_app, start_dash_app and wait_for_element fixtures which significantly helped me to simplify the code in my tests to avoid repeating myself. However, ChatGPT did not understand that the WebDriverWait class from selenium did not return the element being 'waited for' so when I asked it to refactor my test code using the fixtures, it kept giving me the wrong code. Therefore, I had to implement the usage of the fixtures myself.

Dataset

The original dataset has the following details:

> Title: Estates management by academic year and HE provider
> Location: UK
> Academic years: 2015/16 to 2021/22
> Data source: HESA
> Data file canonical link: https://www.hesa.ac.uk/data-and-analysis/estates/data.csv
> Licence: Creative Commons Attribution 4.0 International Licence

The data used in the dashboard has been altered from the dataset above.