National University of Singapore
School of Computing
CS1010S: Programming Methodology
Semester I, 2022/2023

Mission 15
Data Visualisation

Release date: 2nd November 2022
Due: 13th November 2022, 23:59

Required Files

- cs1010s.csv
- mission15-template.py

Information

In this mission, we will have fun learning how to apply our Python-fu in the field of data visualisation. We will be using student records from past semesters of CS1010S, and draw (literally!) some interesting insights from the dataset.

We will be using the `seaborn` library[1] for data visualisation, and this will hopefully spark your interest to learn about the wide spectrum of applications Python is capable of!

If you encounter errors importing the libraries, you may not have installed the packages properly in Mission 0.

This mission consists of **seven** tasks.

Background

One fine day while having lunch with your friends, you enthusiastically share about your cool CS1010S experience! You want to convince your friends that programming is cool! Hopefully they will also want to enroll in the module in the next semester.

They are curious to know more about the cool students that take CS1010S, so they ask more about the general CS1010S population. *What faculties are they from? Is it true that hard work pays off in the module? What is the distribution of marks like?*

Luckily for you, your cool TA Keng Hwee has provided you a CSV file containing information from previous CS1010S cohorts that you can use. You excitedly read in the data, only to realise that there are issues in the dataset - missing values and duplicated rows!

"Garbage in—Garbage out" Keng Hwee cautions as he hands over the data, "If you want to answer your friends questions properly, you will need to clean up the data first!" Keng Hwee then proceeds for his 42nd cup of coffee of the day.

[1] `seaborn` `https://seaborn.pydata.org/index.html` is a Python data visualisation library, which is a high-level wrapper around the `matplotlib` library `https://matplotlib.org/`.

Data

The provided dataset **cs1010s.csv** contains the following columns:

student_num	level	participation	midterm	pe	final	total	faculty	is_afast

- **student_num** (`str`): A unique identifier for each student.
- **level** (`int`): Coursemology Level, capped at Level 50.
- **participation** (`float`): Student's participation in tutorials and forums, capped at 5.
- **midterm** (`float`): Midterm Examination score, out of 75 marks.
- **pe** (`float`): Practical Examination score, out of 30 marks.
- **final** (`float`): Final Examination score, out of 100 marks.
- **total** (`float`): Student's total cumulative percentage.
- **faculty** (`str`): Student's Home Faculty.
- **is_afast** (`object`): Student's indication to AFAST.[2]
  Assign `"yes"` to `True`, `"no"` to `False`, and `"nil"` to `None`.

Data Cleaning (6 marks)

As described in the aforementioned background, you are given a dataset which has issues of missing values and duplicated information.

There are several techniques to handle missing data, e.g. imputing with *mean*, *median* or *mode* values. In our case, we simply **drop the entire row if it has any missing data**.

Most times, duplicate data is caused by bad data entry, but there might sometimes be legitimate reasons for duplication of data. In our case, students who retake the module will have their student number appear more than once. We will only **take their best performing instance** (based on the `total` cumulative percentage).

Another common issue when reading in CSV files is that the data is read in as strings, even if they should instead be parsed as other data types. We want to correctly **represent the data in their correct types**.

We will have to resolve these issues, before starting on our data visualisations!

Task 1: Implement the function **`clean(data)`**, that takes in one argument, **`data`**: a *tuple-of-tuples* in the given column format, excluding the header row. Your function should return a *tuple-of-tuples* after applying these steps sequentially:

1. remove rows containing missing values—represented as empty strings,
2. convert values in the each row to the correct type,
3. remove rows with duplicate **`student_num`**, keeping the row with the highest **`total`** (assume no ties),

The remaining rows in the resulting *tuple-of-tuples* should preserve the order of appearance of the rows in the original *tuple-of-tuples*.

[2] **A**lternative **F**inal **A**sse**S**sment**T** is a programme that allows students to enrol in an alternative assessment timeframe, giving students additional time to learn the fundamental concepts more thoroughly.

Example execution:

```
>>> raw_data = read_csv("cs1010s.csv")
>>> headers = raw_data[0]
>>> headers
('student_num', 'level', 'participation', 'midterm', 'pe', 'final',
 'total', 'faculty', 'is_afast')
>>> data = clean(raw_data[1:])
>>> data[:3]
(('A0001414E', 53, 3.0, 31.0, 16.0, 50.5, 62.4, 'FOS', True),
 ('A0000285W', 46, 2.5, 16.0, 9.0, 47.5, 52.2, 'FOS', False),
 ('A0002236Y', 53, 4.0, 63.0, 25.5, 80.0, 86.35, 'MED', False))
```

By making use of the function **`clean`** on the original raw data, we can generate **`data`**: a *tuple-of-tuples* which consists of unique student rows, no missing values and correct types within each column.

Phew, that was a lot of work. We will be using **`data`** to do some exciting data visualisations in the next few tasks! ☺

Countplots: Univariate Categorical Data Visualisation (3 marks)

In the lecture, we extensively (but not exhuastively) covered the use of `matplotlib`, which is a popular Python library for visualising data. In this mission, we introduce another visualisation library built on top of **`matplotlib`** **`seaborn`**—aliased as **`sns`**.

To use these visualisation libraries, we usually run these import statements:

```
import numpy as np              # alias numpy as np
import seaborn as sns           # alias seaborn as sns
import matplotlib.pyplot as plt # alias pyplot as plt
```

For this task, we introduce the concept of *univariate visualisation*—which deals with plots of a single variable—hence, 'univariate'. Bar charts are well-suited for visualising distributions of univariate **categorical** data.

Your friends wish to find out how many students have chosen to AFAST, the distribution of students from each faculty, and so on... Flexing your python-fu, you decide to come up with a function that can tally the number of times a value occurs in a given column.

Task 2: Implement the function **`value_counts(data, headers, col_name)`**, that takes in three arguments, (i) **`data`**: a tuple of tuples, and (ii) **`headers`**: the tuple of column names, and (iii) **`col_name`**: a string representing the column of interest. The function should return a *tuple of two lists*, with the first list containing the unique categories, and the second list containing the respective counts of each category. The lists should be **ordered from the largest counts to smallest counts**.

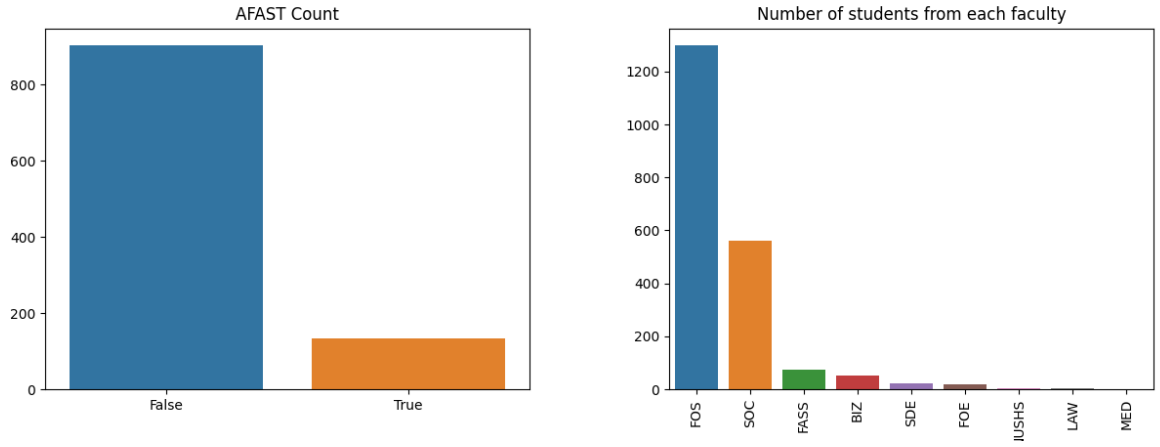
```
>>> afast_counts = value_counts(data, headers, "is_afast")
>>> afast_counts # 132 students chose to AFAST
([None, False, True], [1002, 904, 133])

>>> faculty_counts = value_counts(data, headers, "faculty")
>>> faculty_counts
(['FOS', 'SOC', 'FASS', 'BIZ', 'SDE', 'FOE', 'NUSHS', 'LAW', 'MED'],
 [ 1299,  560,   74,     53,    23,    21,    5,       3      1])
```

Using the result from the **`value_counts`** functions, we can easily visualise the distribution of students from any categorical column:

```
>>> sns.barplot(x=afast_counts[0], y=afast_counts[1])
>>> plt.title("AFAST Count")
>>> plt.show() # display the AFAST counts

>>> sns.barplot(x=faculty_counts[0], y=faculty_counts[1])
>>> plt.title("Number of students from each faculty")
>>> plt.xticks(rotation=90)
>>> plt.show() # display the Faculty counts
```



(a) AFAST count (b) Student count by Faculty

Figure 1: Barplots of Categorical Data

Note: More information at `https://seaborn.pydata.org/generated/seaborn.barplot.html`

Histograms: Univariate Numerical Data Visualisation (10 marks)

After learning how to plot bar charts for univariate categorical data, we now proceed to plot histograms, which are well-suited for univariate **numerical** data.

Note: The `seaborn` library actually works more effectively with data in *wide-form*. You recall that during the lecture, converting data between long and wide forms was discussed. When we use the `read_csv` function, we are returned a *tuple-of-tuples* in *long-form*.

How could we representat data in "*wide-form*" then?

We manipulate convert the data as *tuple-of-tuples* and column names as a *tuple*, into a *dictionary* of key-value pairs: the **keys** are the column name *strings*, and the **values** are the *list* of values belonging to each respective column name **key**.

For example, using a function **`long2wide`** (to-be-defined in this task), we can convert data from *long-form* to *wide-form*:

```
>>> long2wide( (("apple","red"),("banana","yellow"),("banana","green"),
                ("apple","green"),("cherry","red"))
              , ("fruit", "colour") )
{"fruit":  ["apple", "banana", "banana", "apple", "cherry"],
 "colour": ["red",   "yellow", "green",  "green", "red"]}
```

Using the result from `long2wide`, we can easily visualise the distribution of midterm scores:

```
>>> wide_data = long2wide(data, headers)
>>> sns.histplot(data=wide_data, x="midterm")
>>> plt.title("Midterm Distribution")
>>> plt.show()
```

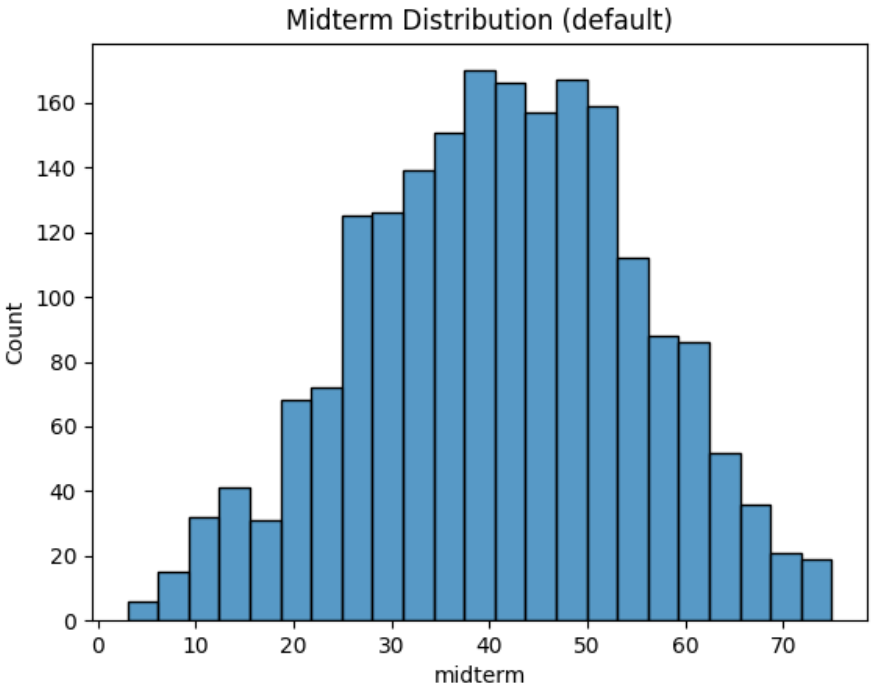


Figure 2: Histplot of midterm score

Task 3: Implement the function **`long2wide(data, headers)`**, that takes in two arguments, (i) **`data`**: a *tuple of tuples*, and (ii) **`headers`**: a *tuple* of column names. The function should return the *dictionary* containing the column names as the keys, and the associated *lists* of column data as the values. [3 marks]

Binning strategies of histograms

Once additional parameter that we might want to adjust is the number of bins. Instead using the default bin-count, there are several popular methods to automatically determine the number of bins. We will try implementing several methods.

Task 4A: Implement the function **`bin_sqrt(values)`**, that uses the square-root to compute and return the number of bins, $\left\lceil \sqrt{N} \right\rceil$, where N is the `len(values)`. [1 mark]
Note: $\lceil x \rceil$ *means* `math.ceil(x)`.

Task 4B: Implement the function **`bin_rice(values)`**, that uses Rice's Rule to compute and return the number of bins, $\left\lceil 2 \times \sqrt[3]{N} \right\rceil$. [1 mark]

Task 4C: Implement the function **`bin_sturge(values)`**, that uses Sturge's Rule to compute and return the number of bins, $\lceil log_2 N \rceil + 1$. [1 mark]

Task 4D: Implement the function **`bin_scott(values)`**, that uses Scott's Rule to compute and return number of bins, $\left\lceil \frac{max(values)-min(values)}{3.49\times\hat{\sigma}} \times \sqrt[3]{N} \right\rceil$, where $\hat{\sigma}$ is the **sample standard deviation** of the values in the given `col`. [1 mark]
Hint: Use `np.std(..., ddof=1)` *to compute* $\hat{\sigma}$*.*

Task 4E: Implement the function **`bin_fd(values)`**, that uses Freedman-Diaconis's Rule to compute and return the number of bins, $\left\lceil \frac{max(values)-min(values)}{2} \times \sqrt[3]{IQR} \right\rceil$, where IQR is the **inter-quartile range** (difference between the 75th and 25th percentile) of the values in the given `col`. [1 mark]
Note: Use `np.percentile(..., ...)` *to compute the respective percentiles.*

Effect of different bin-counts

Plot histograms using various bin strategies and observe the effect of different bin sizes. Then answer the following two questions.

Task 5A: Is it meaningful to have a bin-count of 1? Why or why not? [1 mark]

Task 5B: Is it meaningful to have a bin-count roughly equal to the length of values N? Why or why not? [1 mark]

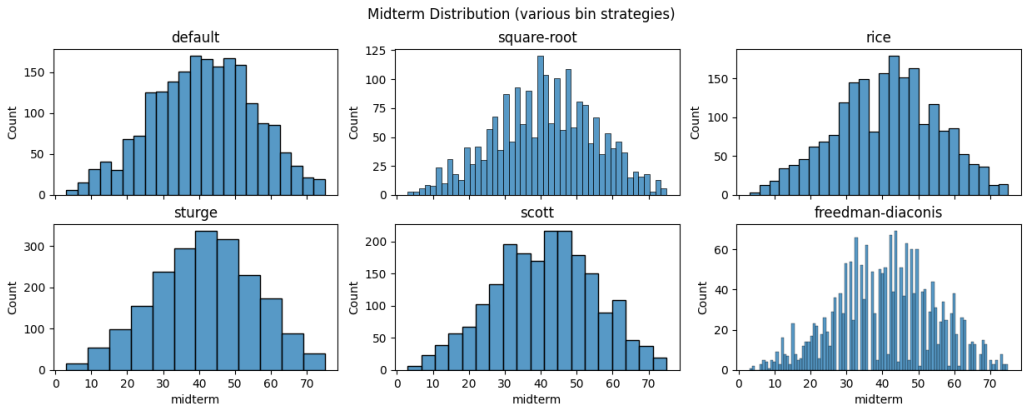


Figure 3: Histplot of varing binning strategies

Note: More information at `https://seaborn.pydata.org/generated/seaborn.histplot.html`

Boxplots: Multivariate Visualisation of Numerical against Categorical Data (5 marks)

We have discussed various forms of univariate visualisations, and now proceed to discuss visualisations involving more than 1 variable. Your friend wants to know how students from different faculties perform for CS1010S, and you realise this question can be suitably answered with *boxplots* of the total percentages grouped according to each faculty. Take note that we are dealing with two variables here, the **faculty** categorical variable, and the **total** numerical variable.

Since we already have the data in *wide-form*, we can immediately use of Seaborn's `boxplot` function. Some example usage of `boxplot` is available in the official documentation at `https://seaborn.pydata.org/generated/seaborn.boxplot.html`.

In our context, we can immediately call `sns.boxplot(x="faculty", y="total", data=wide_data)` to plot the boxplots. However, we notice that the data can be arranged more intuitively if we sort the data according to the median values of each category.

Task 6A: Implement the function **`order_by_median(wide_data, category, numerical)`**, that takes in three arguments, (i) **`wide_data`**: *dictionary* of data in *wide-form*, (ii) **`category`**: a *string* representing the categorical column name, and (iii) **`numerical`**: a *string* representing the numerical column name. The function should return the *list* of unique categories from the **`category`** column, arranged in decreasing order based on the median value of the **`numerical`** column. [4 marks]

Note: The median value is the middle value of the sorted list if there are an odd number of values, otherwise the median is the average of the middle two values.

```
>>> cat_col = "faculty"
>>> num_col = "total"
>>> cat_order = order_by_median(wide_data, cat_col, num_col)}
>>> cat_order
['MED', 'NUSHS', 'LAW', 'SOC', 'FOE', 'FOS', 'BIZ', 'FASS', 'SDE']

>>> sns.boxplot(x=cat_col, y=num_col, data=wide_data, order=cat_order)
>>> plt.title("Distribution of total by faculty")
>>> plt.show()
```

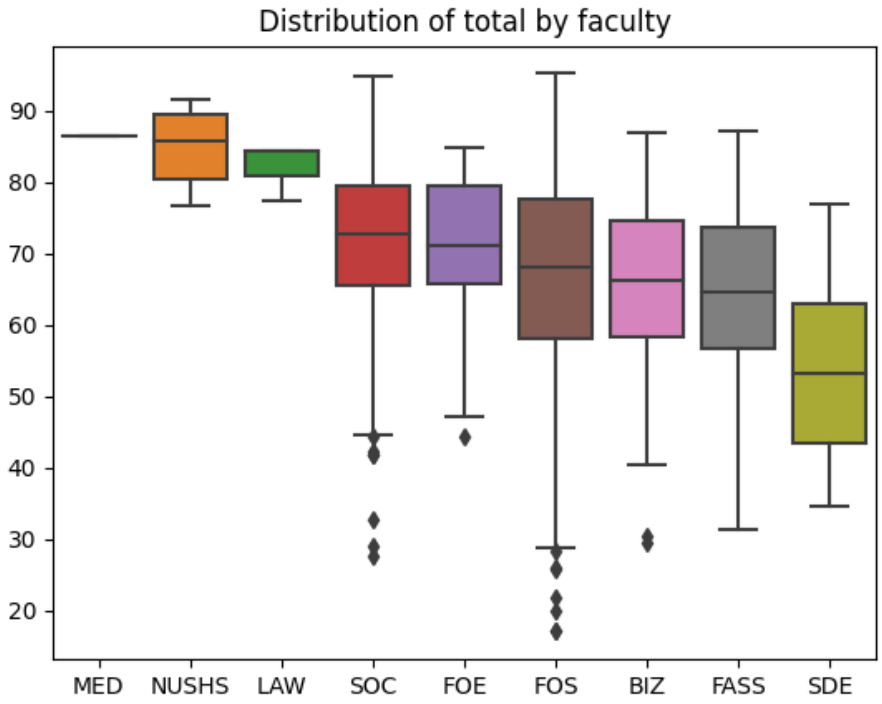


Figure 4: Output of `sns.boxplot(x=cat_col, y=num_col, data=wide_data, order=cat_order)`

Task 6B: Study the data so far. Does the data suggest that an incoming student from Medicine (MED) is more likely to perform better than an incoming student from another faculty? Why, or why not? [1 mark]

Scatterplots: Multivariate Visualisation of Numerical against Numerical Data (3 marks)

Previously, we have looked at understanding relationships between a categorical and numerical variables through boxplots. For the final task, you will use *scatter-plots* to visualise relationships between two numeric variables.

`seaborn` has a function `sns.scatterplot` for plotting scatterplots, that similarly uses data in *wide-form*. Your friends want to know if students who have worked hard on Coursemology have also done well in CS1010S. We can investigate this using scatterplots, since **`level`** and **`total`** are numeric variables provided in the dataset.

Being curious, you also wish to know if how students of various faculties are spread across the scatterplot. We can achieve this by color-coding and styling the points in the scatterplot, using the `hue` and `style` named-arguments in the `sns.scaterplot` function.

```
>>> sns.scatterplot(x='level', y='total', data=wide_data,
                    hue='faculty', style='faculty', s=15)
>>> plt.title("Students' level vs Students' total")
>>> plt.show()
```

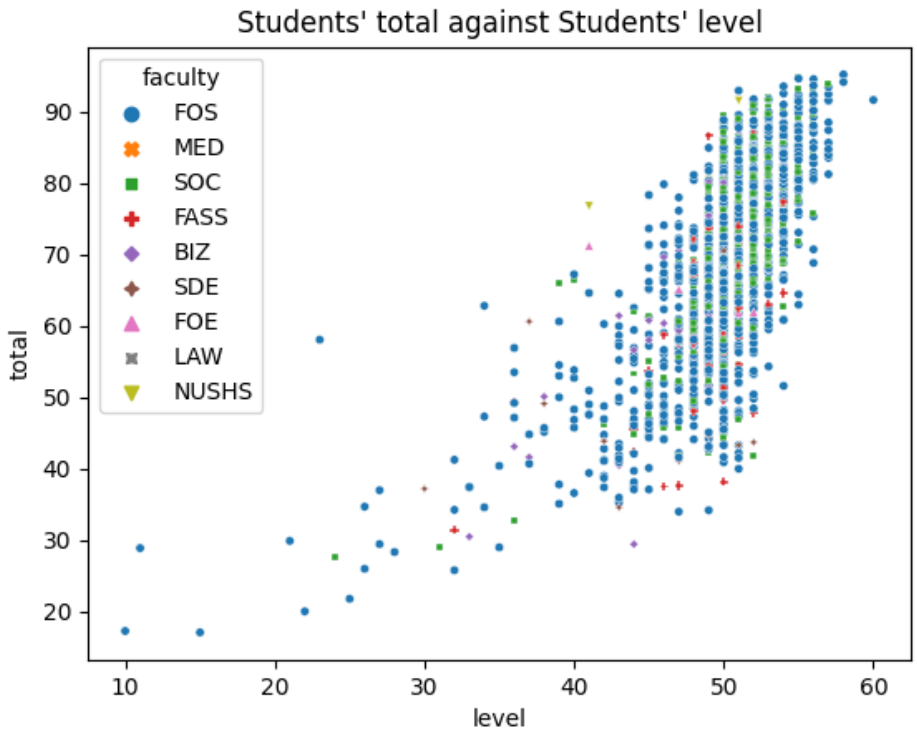


Figure 5: Scatter plot of students' total and their Coursemology level

We can see a positive relationship between Coursemology level and the total score, so the data suggests that completing the exercises on Coursemology should pay off in the long run! Congratulations on reaching this point of the final mission, you are in good hands! ☺

Task 7A: Plot the scatter plot of `final` score (y-axis) against `midterm` score (x-axis) using a similar command in the example above. You are expected to correctly identify arguments for `x`, `y` and `data`. [1 mark]

Task 7B: Analyze and state the trend in the scatter plot in Task 7A (Figure 6). [1 mark]

Task 7C: Your friend claims that since he did badly for the midterm examination, he will surely perform poorly for the final examination as well. Is your friend justified in his conclusion? Why or why not? [1 mark]

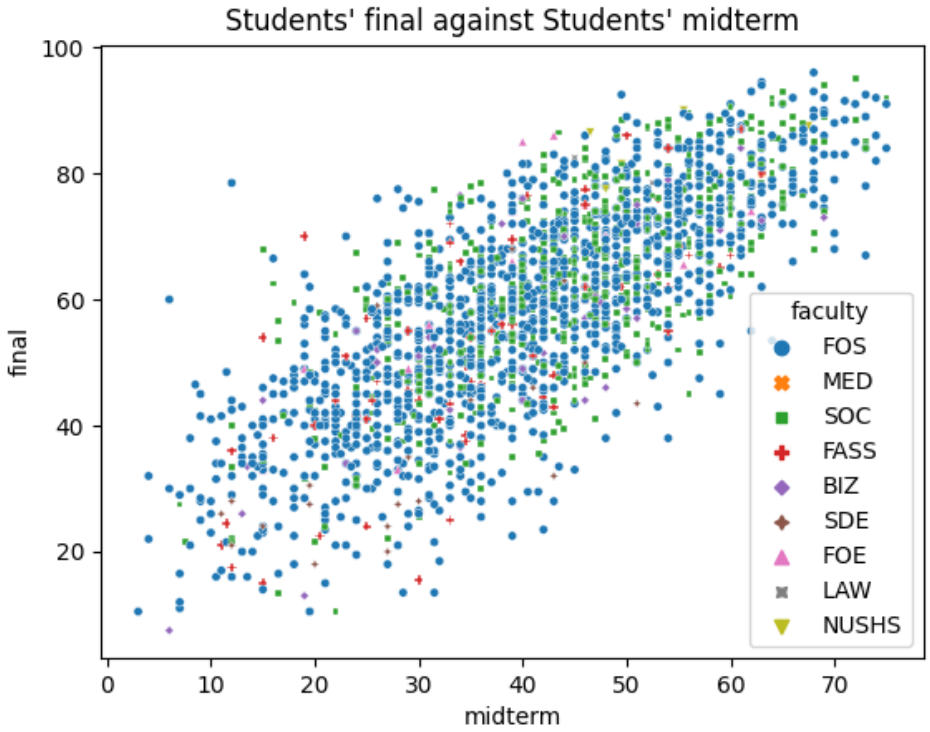


Figure 6: Example output of Task 7A