COSC 2670: Practical Data Science

Assignment 1: Data Cleaning and Summarising

Due 12.00 pm on 30 March 2017

Submitted by: Casey-Ann Charlesworth (3132392)

Table of Contents

Table of Tables

Table of Figures

1. Data preparation

1.1 Load the CSV data from the file. You need to use an appropriate pandas function to load the csv data, and make use of the correct arguments including sep, decimal, header, names, if needed.

Used the pandas function to read the csv file, including sep and decimal definitions and instructed the system to read from the headers in the csv file. The code looked like this:

```
import pandas as pd
filename = "TeachingRatings.csv"
ratings = pd.read_csv(filename,sep=",", decimal=".", header=0)
```

1.2 Check whether the loaded data is equivalent to the data in the source (CSV) file. That is, you will need to ensure that the loaded data has appropriate data types assigned, or take steps to ensure that the appropriate types are used.

Checking the data types produced this result:

```
minority      object
age          float64
gender        object
credits       object
beauty       float64
eval         float64
division      object
native        object
tenure        object
students       int64
allstudents    int64
prof           int64
dtye: object
```

Looking at the csv file, I felt “age” should be int64 (as we were not dealing with anyone < 1 year old). I therefore changed the data type using the code:

```
ratings["age"] = ratings["age"].astype(int)
```

However, it should be noted that this code could only run after the NaNs were removed (which is detailed in step 1.7 below).

The variable “prof” was identified as an int, however, as it is a unique identifier, we should be careful not to leave it open to accidental mathematical calculations. Therefore, I changed it to a string using the following code:

```
ratings["prof"] = ratings["prof"].astype(str)
```

Which produced the amended data types:

```
minority      object
age            int32
gender        object
credits       object
beauty       float64
eval         float64
division      object
native        object
tenure        object
students       int32[1]
allstudents    int64
prof          object
dtye: object
```

[1] It should be noted that this was altered automatically due to a calculation in step 1.6 below, and had to be converted back manually in the .py script (hence the change to int32)

1.3 Check whether there are typos in the data. If there are any typos, correct them by using masks.

Below in Table 1 I have outlined what actions were required for each variable.

Table 1: Actions required to mask typos within string variables

Variable	Action
minority	Using the .value_counts() function, I checked for typos, then used a mask to locate and correct the typo
gender	No typos (other than upper/lower case issues). No action in this step
credits	Assumption made that “much more” (as there was only one count of this value) should be “more” only. Same process as “minority” above with single/more resulting values
division	Same as “minority” above but only one “lower” typo needed to be corrected
native	Same as “minority” above
tenure	Same as “minority” above
prof	As prof is “a randomly assigned unique identifier” I checked for anomalies using the .value_counts() function. However, as this variable contained unique identifiers, it would be more logical to do a quick double check while the variable is an int, so this is what I did to ensure there were no values > 100 (as this is what the .value_counts() function appeared to reveal) No subsequent action was taken on this column

1.4 Check whether there are instances of extra whitespaces in the data, and if so, demonstrate how to remove them by calling on an appropriate function.

This step was completed in conjunction with step 1.5 below, using the following code:

```
string_variables = ratings.loc[:, ratings.dtypes == object]
for v in string_variables:
    ratings[v] = ratings[v].str.lower()
    ratings[v] = ratings[v].str.strip()
```

1.5 Demonstrate how to cast text data to lower-case, using an appropriate function.

Please refer to step 1.4 above.

1.6 Design and run a small test-suite, consisting of a series of sanity checks to test for the presence of impossible values for each attribute.

Table 2: Actions required to perform sanity checks on numeric variables

Variable	Action

age	Using the below sanity check (age not < 1 nor > 100):
	`bad_lines = ratings.loc[(ratings["age"] < 0) | (ratings["age"] > 100)]`
	`print(bad_lines)`
	the following lines were revealed to be problematic: 104, 112, 127, 132, 133.
	I wrote a script that changed these values in NaNs (to be replaced by the column wise mean in step 1.7)

beauty	As this variable was a calculation, I checked the min() and max() values to ensure there were no errant *after the fact* typos. The range was between 2:-2 so I took this to be accurate. Nothing more was done to this variable for this step

eval	Similar to age variable above, I wrote the below sanity check (eval not < 1 nor > 5 given that the value was expected to be between 1-5 inclusive):
	`bad_lines = ratings.loc[(ratings["eval"] < 1) | (ratings["eval"] > 5)]`
	`print(bad_lines)`
	and the following lines were revealed to be problematic: 34, 83

Variable	Action
	I changed the values <1 and >5 to NaN values to be replaced by the column wise mean at step 1.7 below.
students allstudents	I handled these two variables together, and used a similar sanity check to those used above, however, this time comparing one variable to another. This checked to see whether the students value > allstudents value: *bad_lines = ratings.loc[ratings["students"] > ratings["allstudents"]]* *print(bad_lines)* The following lines were revealed to be problematic: 29, 85 This time replacing them with the column-wise mean would not be appropriate as the mean of "students" was 36.65. Therefore, I made the decision to replace each impossible value with the calculation: allstudents * 2/3 (as this was more scalable than calculating, say, 1/2 the students value) It should be noted that this calculation forced the dtype into *float*, therefore, as the type should be (and was originally) *int*, I wrote a line that returned it to type *int*

1.7 Check whether the loaded data has any missing values. If so, use an appropriate function to replace them with the column-wise mean value.

Now that all steps above had been completed, a final conversion of all NaN values could take place using the following code:

ratings.fillna(ratings.mean(axis=0), inplace=True)

2. Data exploration

2.1 Create a visualization for each column (except prof) by producing an appropriate type of graph.

- **You should explore each column with at least one type of graph, but you can explore with more than one type, including histograms, barcharts, pie graphs, or boxplots.**
- **Format each graph carefully. You need to include appropriate labels on the x-axis and y-axis, a title, and a legend. The fonts should be sized for good readability. Components of the graphs should be coloured appropriately, if applicable.**

Please note that for the graphs created below, I felt they were all clearly labelled without having the extra distraction of a separate legend (as instructed in the spec).

Minority

The following two figures were produced as Minority was a yes/no category. Therefore I chose both pie and bar charts to demonstrate the percentage and counts of the information as shown below in Figure 1 and Figure 2

Figure 1: Minority count by percentage

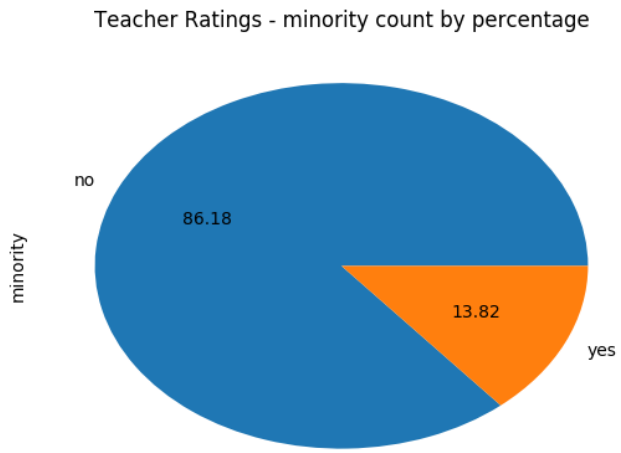
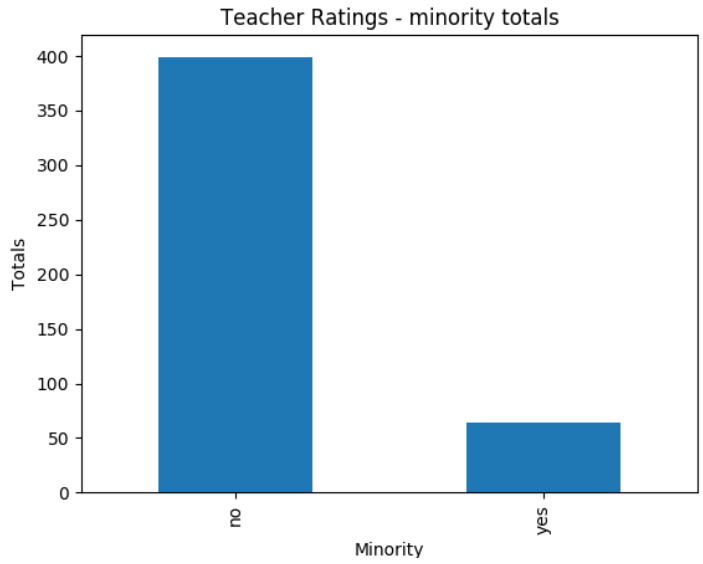


Figure 2: Minority totals



Age

Being a numeric category, it should be shown as a distribution of values – therefore below in Figure 3 and Figure 4 I have demonstrated this in both a histogram and boxplot.

Figure 3: Age distribution

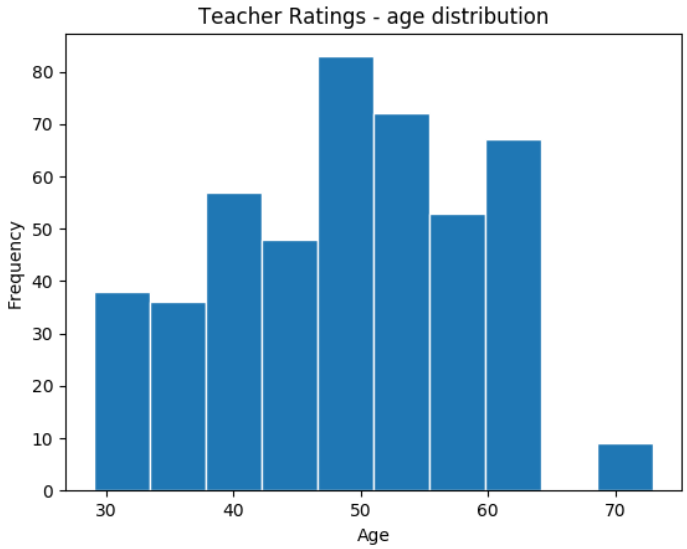
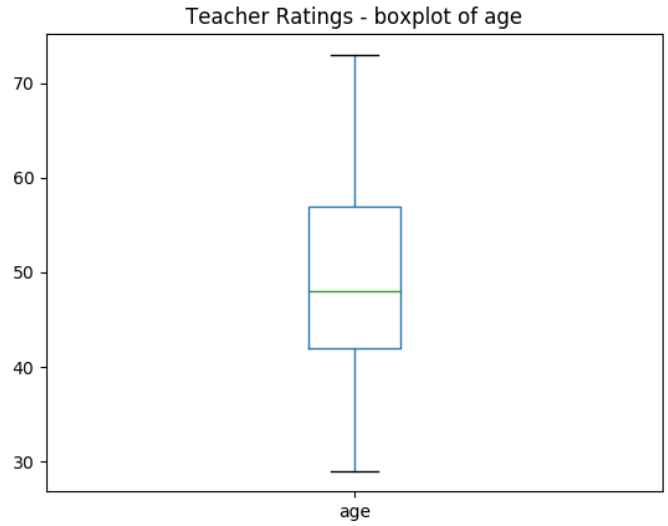


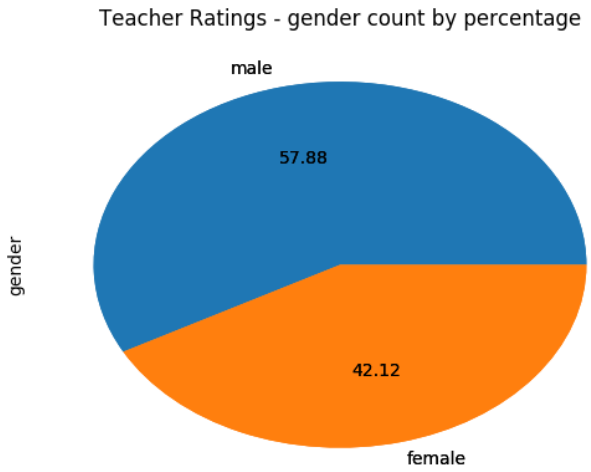
Figure 4: Boxplot of age



Gender

Gender, again, being a 2 value categorical variable would suit either a pie or bar chart. I have produced a pie chart (Figure 5 below) to demonstrate the percentage of each gender count.

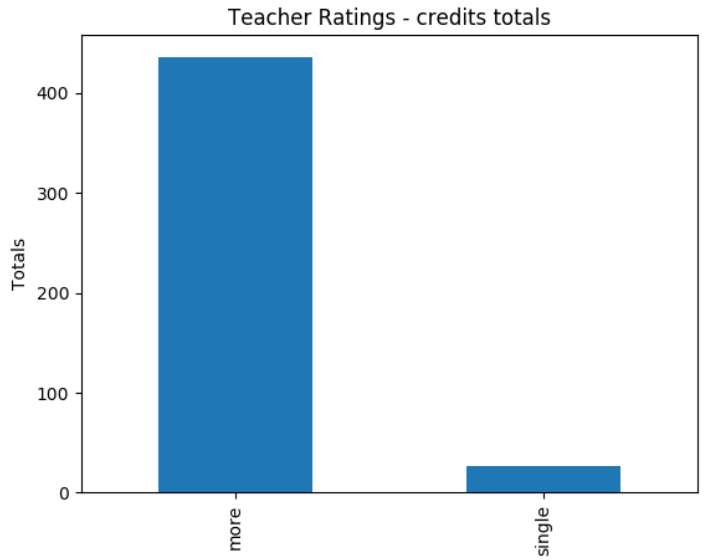
Figure 5: Gender count by percentage



Credits

Same with minority and gender above, credits is a 2 value categorical variable, so I have demonstrated this with a bar chart below in Figure 6.

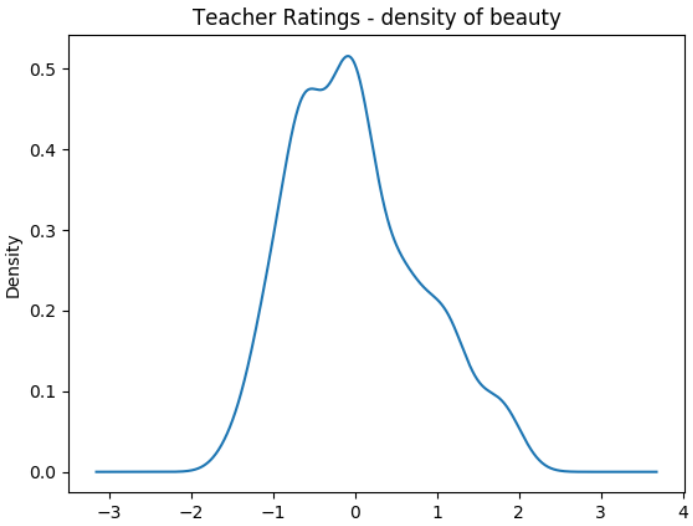
Figure 6: Credits totals



Beauty

I wanted to try something different with beauty, given that its value was a calculation (which, in a basic interpretation, shows negative as “not attractive” rating and positive as “attractive” rating). Therefore, I used a density plot to demonstrate this in Figure 7 below.

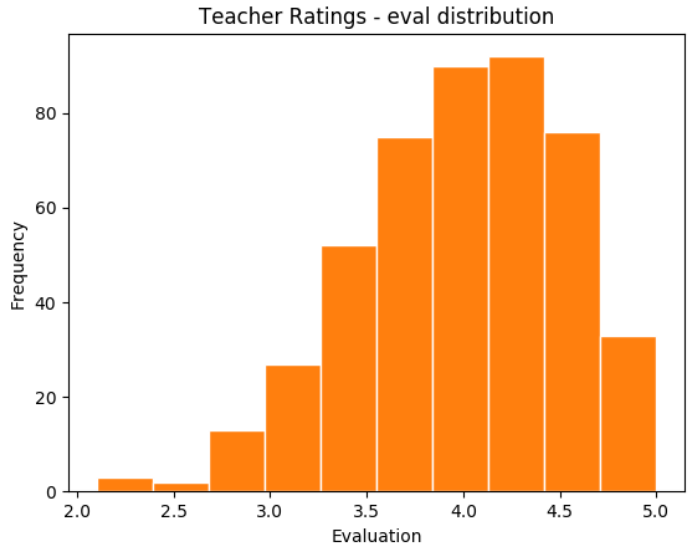
Figure 7: Density of beauty



Eval

Although eval is an ordinal value (where 1: very unsatisfactory, to 5: excellent), as the data supplied contains an average (or mean) of the course evaluation, I am therefore not sure whether it should therefore be grouped into 5 static categories (and shown in a column or bar chart). Therefore, I have plotted it as a histogram to show the frequency of the ratings. This is shown in Figure 8 below.

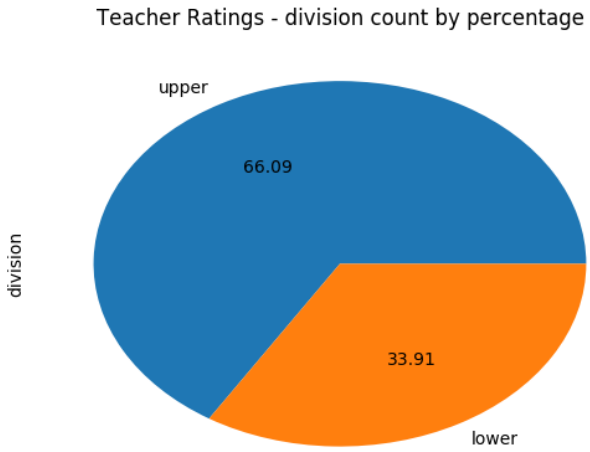
Figure 8: Eval distribution



Division

Again, being a categorical variable, a pie or bar chart would suit best. Therefore below in Figure 9 is a pie chart to represent division.

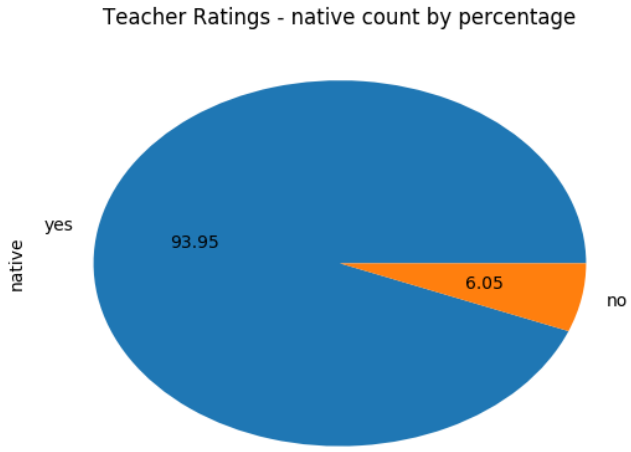
Figure 9: Division count by percentage



Native

Same as above with categorical data, this is represented below in Figure 10 also in a pie chart.

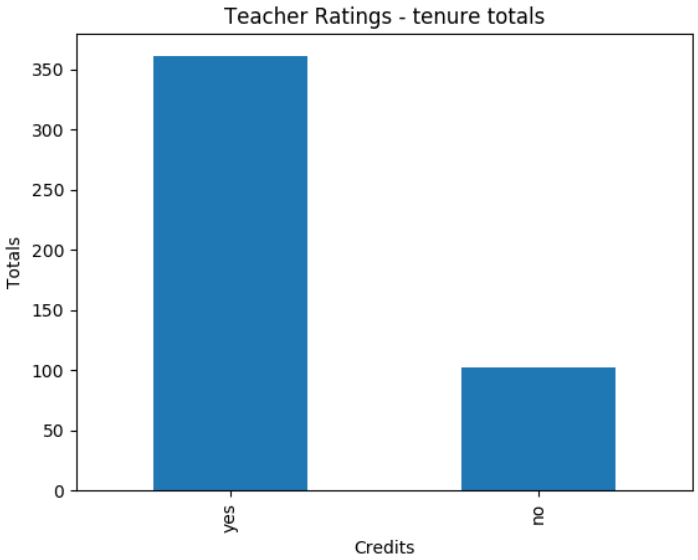
Figure 10: Native count by percentage



Tenure

As with the many categorical variables above, tenure is best show in a pie or bar chart. I have produced a bar chart as can be seen in Figure 11 below.

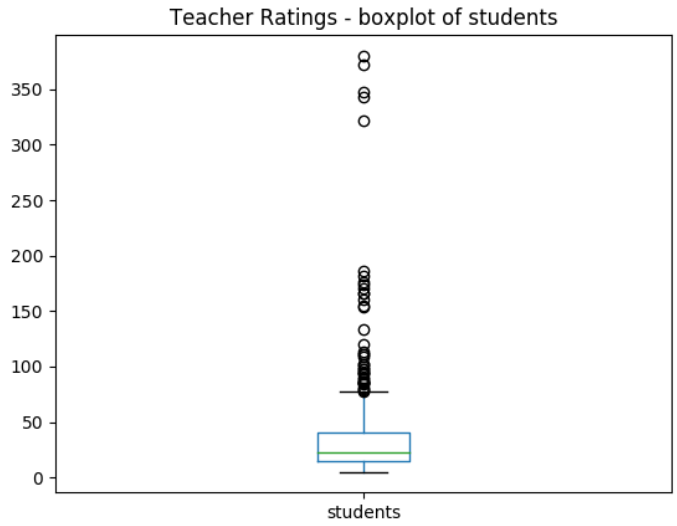
Figure 11: Tenure totals



Students

I chose to use a boxplot to show students – as this numerical category contained values that could be quite disparate. This is represented in Figure 12 below.

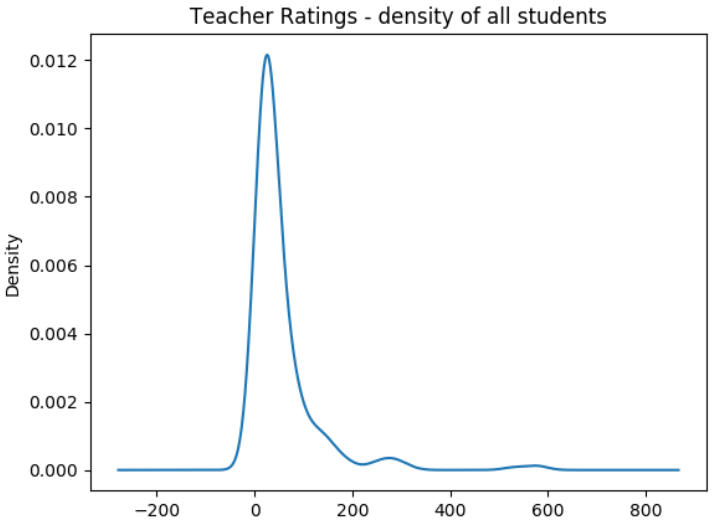
Figure 12: Boxplot of students



Allstudents

Similar to students category above, allstudents contained numerical values that had a large range. This time I chose to use a density plot to demonstrate this (see Figure 13 below).

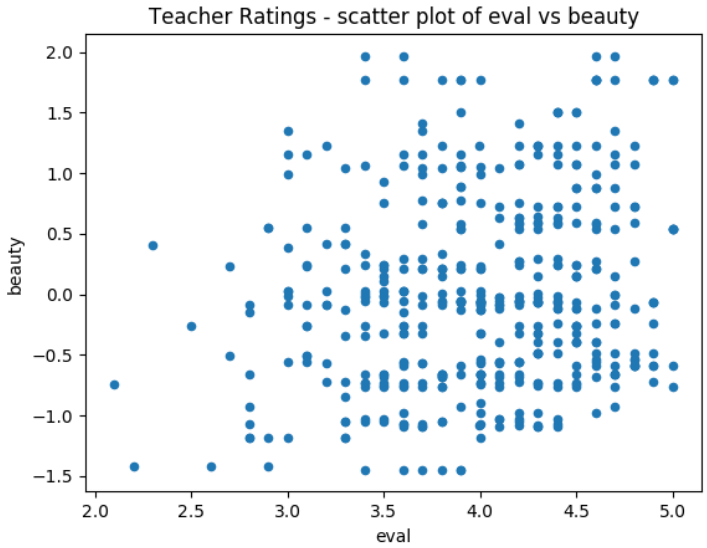
Figure 13: Density of allstudents



2.2 Explore the relationships between columns. You may choose which pairs of columns to focus on, but you need to generate at least 3 visualisations for this subtask. These should address a plausible hypothesis for the data concerned. For example, you might wonder: is there a relationship between the age of an instructor and the course quality as perceived by students? An appropriate visualisation for this could be to graph age against eval scores.

Hypothesis: Is there a correlation between the course evaluation and the perceived attractiveness of the teacher?

Figure 14: Scatter plot of eval vs beauty

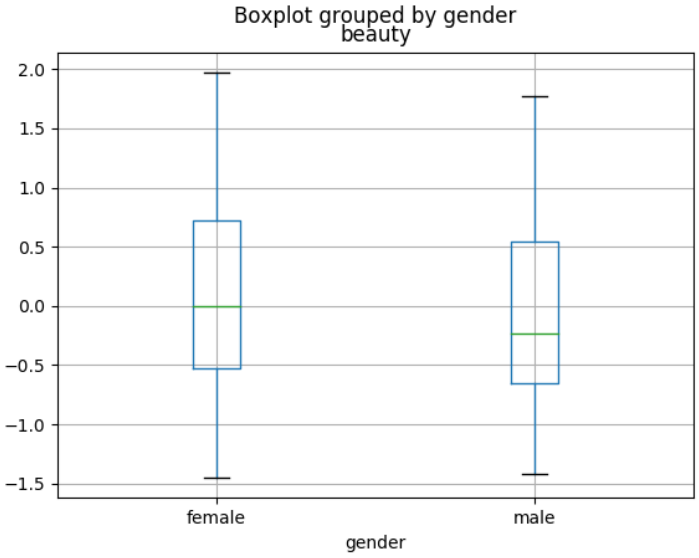


My reasoning behind this hypothesis, was that students may relate a teacher's attractiveness to their ability to teach a good course, thus biasing the perceived attractiveness. The scatter plot in Figure 14 above, I believe, reveals a slight trend towards proving this hypothesis, in

that no teacher was given a *low* course evaluation yet was still perceived to be attractive, whereas, teachers given *high* course evaluations were also perceived to be attractive. Where this data falls down in the hypothesis is that teachers could be awarded a high course evaluation, but also thought to be less attractive.

Hypothesis: Is there a correlation between perceived attractiveness and gender?

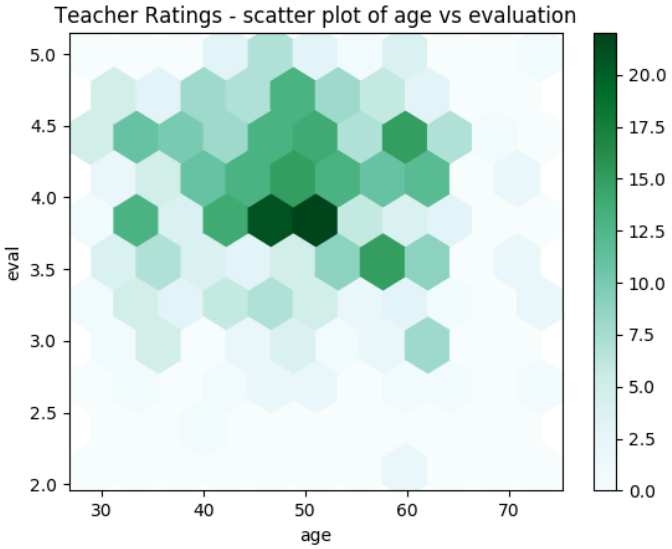
Figure 15: Boxplot grouped by gender



The grouped boxplot in Figure 15 above shows a clear distinction between the attractiveness of a teacher based on whether they are male or female. Female teachers are thought of as more attractive (but with a median of approximately zero), with the interquartile range of male teachers containing more negative data, and demonstrating them to be thought of as less attractive.

Hypothesis: Is there a correlation between age and course evaluation?

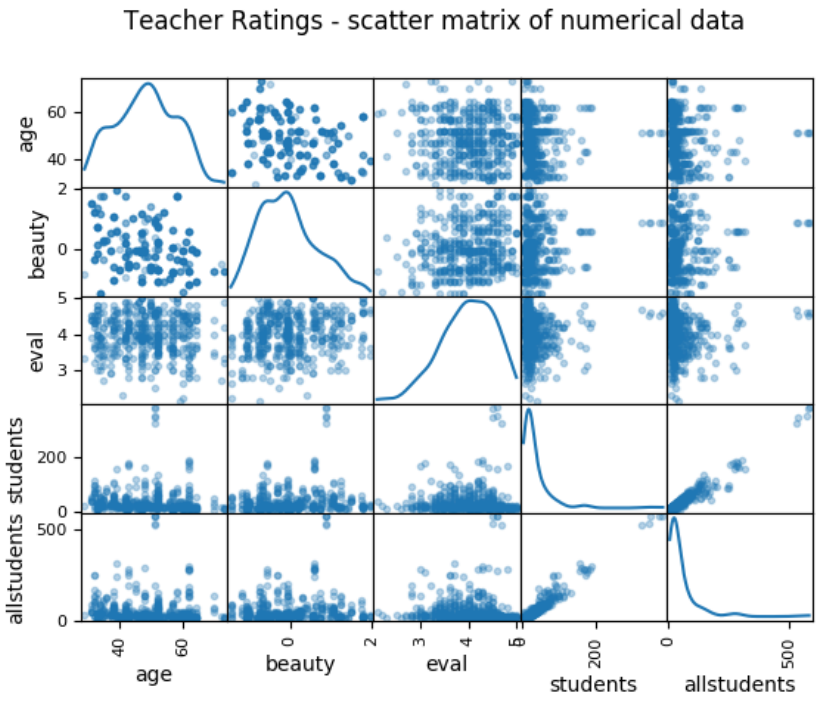
Figure 16: Scatter plot of age vs evaluation



I initially ran this as a scatter plot, but was disappointed with the results. When I changed it to hexbin, the data came to life. Figure 16 above reveals a sweet spot of teachers aged between approximately 45-55 who tend to generate higher evaluation scores.

2.3 Build a scatter matrix for all numerical columns.

Figure 17: Scatter matrix of numerical data



3. Extension

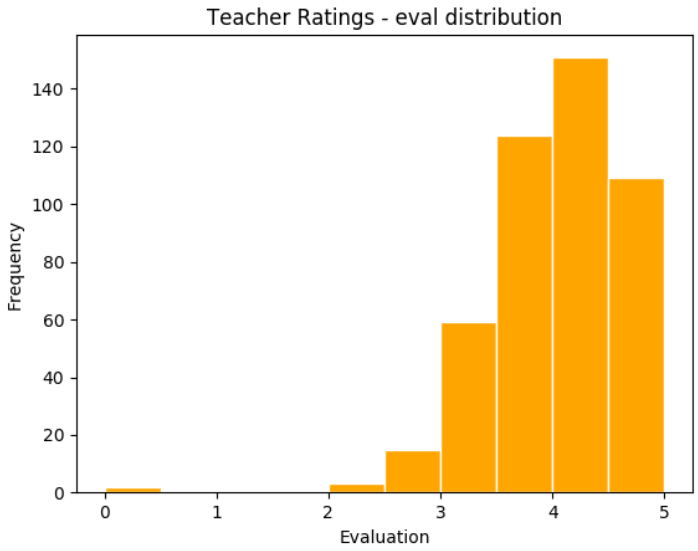
3.1 In your report.pdf file, create a heading called "Extension". In this section, include your three graphs. Under each one, briefly discuss the impact that the different approaches to dealing with missing values have on what you observe from the visualisation.

(A) Replacing them with a fixed value

For this step, I created a new independent .py file (called P3A_s3132392.py) that replaced the NaN values with the fixed value of zero. This was done as the NaNs spanned different numeric variables and hence, "guessing" at a suitable value would not likely fit all variables

The following graph was created for "eval" – matching the graph in section 2 (being Figure 8):

Figure 18: Eval distribution - where NaNs are replaced with zero



It is evident that the distribution changes quite dramatically as the first graph did not contain any values below 2. Therefore, this graph now looks skewed to the left.

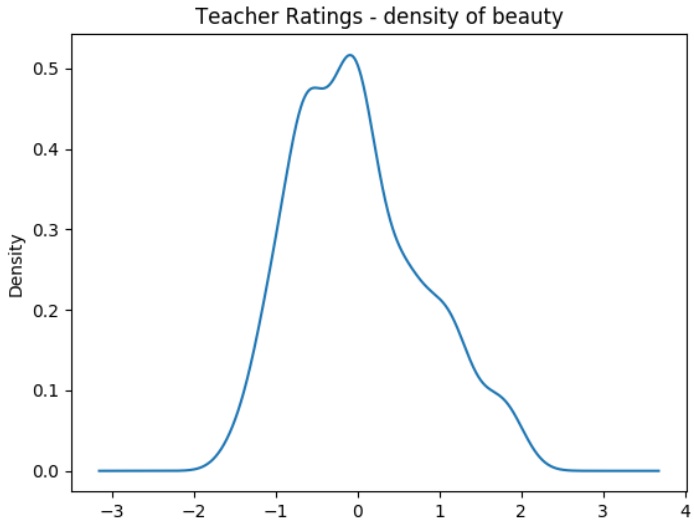
(B) Replacing with the median value (column-wise)

For this step, I created a new independent .py file (called P3B_s3132392.py) that replaced the NaN values with the fixed value of zero.

I tested with graphs the variables of "age", "students", and "beauty" and none really showed any difference to their original graphs. This is to be relatively expected.

Below I present "beauty" where the NaNs have been replaced with the median.

Figure 19: Density of beauty - where NaNs are replaced by the median



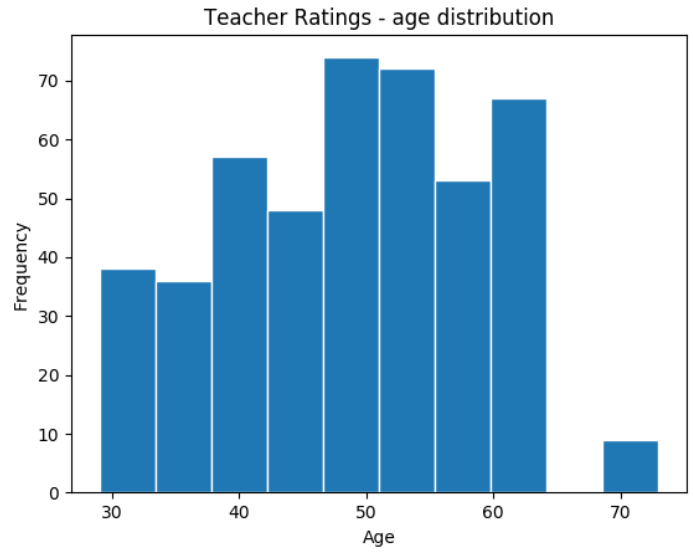
(C) Ignoring all observations containing missing values

For this step, I created a new independent .py file (called P3C_s3132392.py) that ignored all rows with NaN values.

The first issue I incur is the inability to convert the “age” variable to int (as it doesn’t work while NaNs are present – and I plan to use the .dropna() code to create my new graph). Therefore, I have left the “age” variable, in this step only, as a float.

I decided to look at the “age” variable for this graph comparison. I checked both the histogram and box plots and compared against what I did in Figure 3 and Figure 4. While the box plot didn’t appear to change much, the histogram did. See below for comparison.

Figure 20: Age distribution - ignoring NaN values



The 5th column has reduced drastically as this is approximately the mean and was clearly affected by the NaN values being replaced by the column-wise mean.

4. References

Boschetti, A and Massaron, L, 2015, *Python Data Science Essentials*, Packt Publishing Ltd, Birmingham, UK.