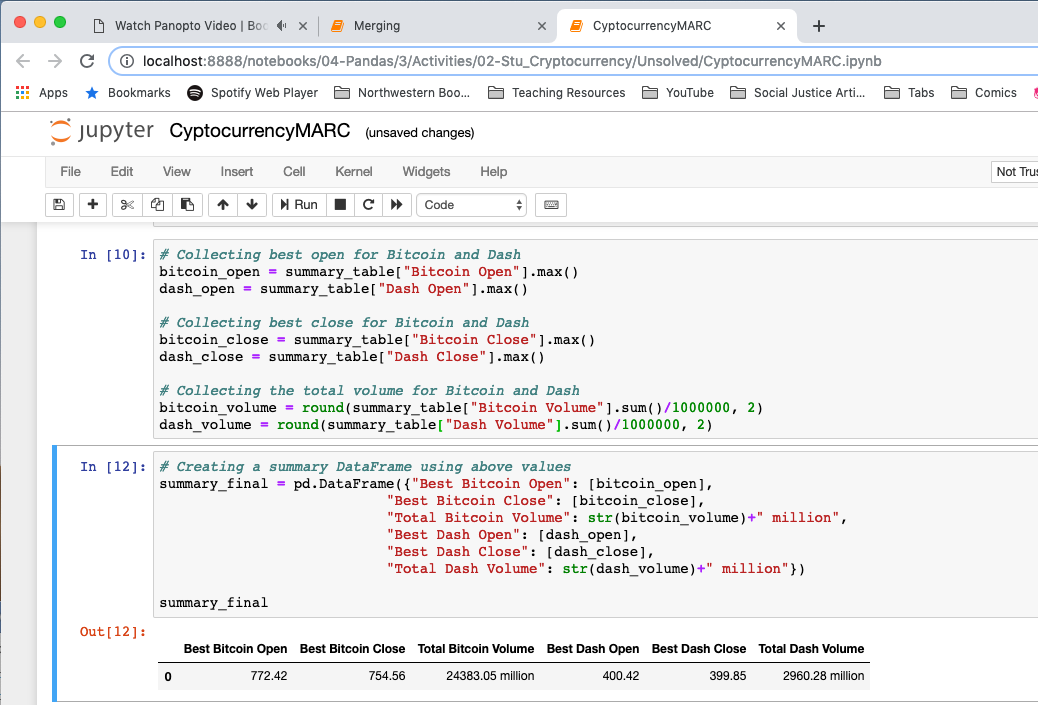
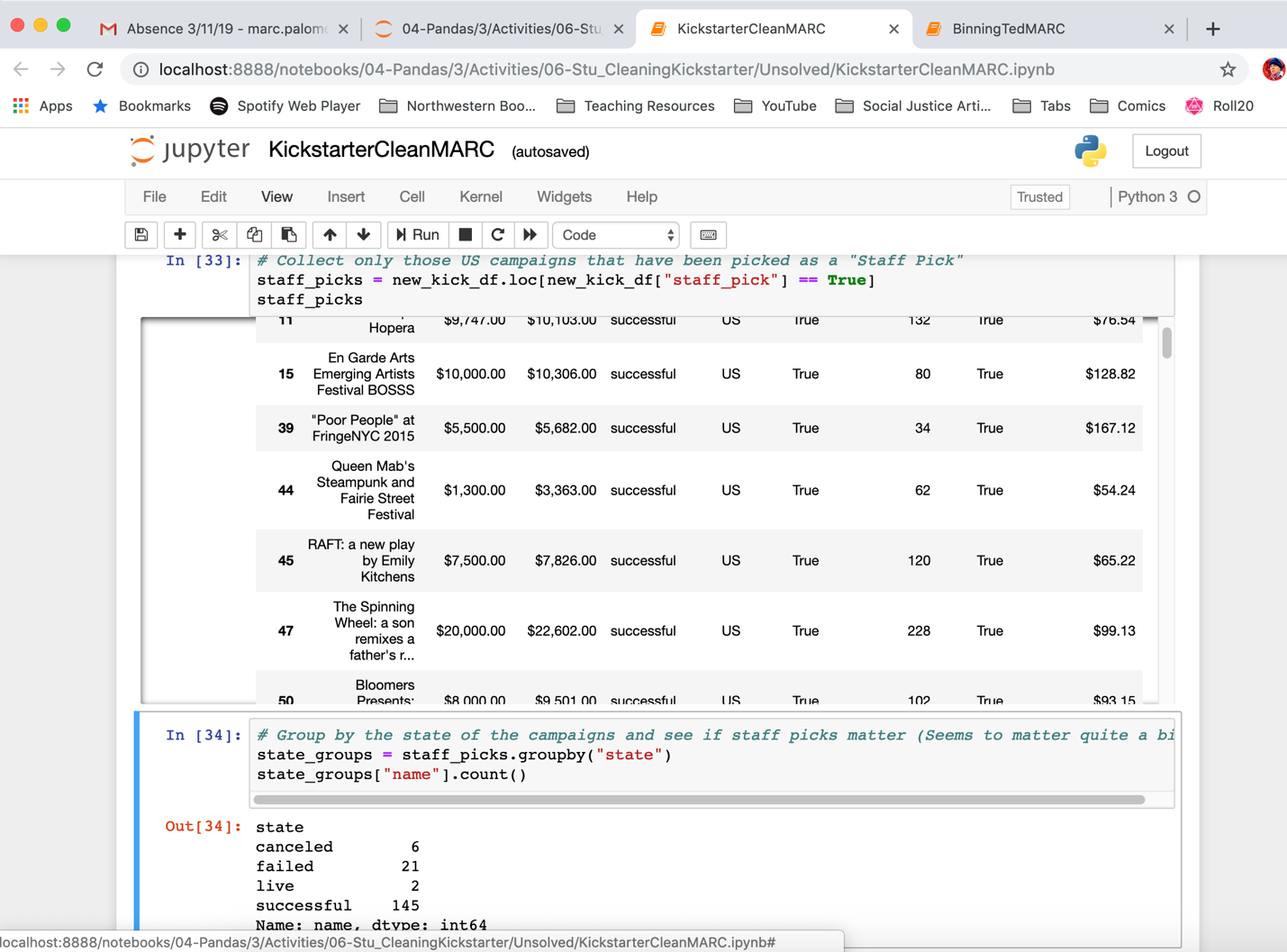
In the first instructor lead activity, Merging, we looked at how to merge two dataframes. The first example showed “inner join”, which dropped pieces that did not have a corresponding data set in both dataframes. The opposite is the “outer join” which took all components of both dataframes and replaces a lack of data with “NaN”. Next we use “left join” and the dataframe on the ‘left’ in the code takes priority: pd.merge(**info\_pd**, items\_pd, on=”customer\_id”, how=”right”). Everything in “info\_pd” will show up in the new dataframe and “NaN” will fill where no corresponding data is found in the second dataframe. The reverse here is the “right join”, which prioritizes data in “items\_pd” in the example listed above. It is important to remember the default merge is “inner join”. Below is my final picture for my Cryptocurrency activity. I did not consider dividing by 1,000,000 and rounding to make the number seem smaller, but added it when I saw the answer:



The second instructor lead activity, Binning, which takes data and assigns it a corresponding new data label within different buckets, or breakpoints that are inclusive coming from the left (the low to high). In this instructor lead activity, the max was correct for the test score, but the corresponding student name was not correct. I think using a loc to keep the indexed score would help pull the correct student name. Before the next student activity, Kerry asked to move on into the next instructor lead activity because the following student activity was more complicated and needed to combine both new skills. Therefore, he went into Instructor Lead Mapping, where we learned that mapping should be done at the end. Once formatting is applied the numbers are changed to strings and you can use the .type to see the new data types. The corresponding activity is KickstarterClean, and below is a screenshot of the final components of my code. I realized I was doing the average incorrectly. I was trying to average the entire the column, but the formula was pledge/backers for what was being asked:



The final activity focused on cleaning code through debugging. This example showed us how to Google Fu problems based on the error messages we were receiving. The student assignment, BugfixBonanza proved to be quite challenging since there was an error that was corrected incorrectly. The incorrect correction did not cause an error but there was a problem in how it grouped the data, which wouldn’t allow the index to be reset. In the video, the instructor attempts to go through the code and fix all the problems. The incorrect fix was in the second to last step:

# Find the percentage of bugs overall fixed by each Assignee

total\_bugs = assignee\_group["Assignee"].count()

bugs\_per\_user = assignee\_group["Assignee"].**value\_counts()**

user\_bug\_percent = pd.DataFrame((bugs\_per\_user/total\_bugs)\*100)

user\_bug\_percent.head()

Needed to be come:

# Find the percentage of bugs overall fixed by each Assignee

total\_bugs = eclipse\_df["Assignee"].count()

bugs\_per\_user = assignee\_group["Assignee**"].count()**

user\_bug\_percent = pd.DataFrame((bugs\_per\_user/total\_bugs)\*100)

user\_bug\_percent.head()

without correcting this component, the code still runs correctly; however, the index is unable to be reset. A picture of my final version is below:

