I worked on the automated data-cleaning prompt. The tool I built – **AnaLazy** – explicitly provides data scientists with a comprehensive understanding of how their data cleaning procedures might impact the skew of their data, which we've learned in class can directly influence the fairness of models built on that data if a hidden skew/bias is accidentally introduced. My tool also provides suggestions for the user to manually investigate certain issues with the data if the tool determines that there might be some existing trend in erroneous data which needs to be addressed.

AnaLazy is a web app built using the Python Flask framework. **I have attached a video demo to my Canvas submission**, and I will briefly discuss the implementation of the various features shown in the demo here. First, there is a landing page that allows you to upload data, go through the automated data fixing process, or go to the data quality terminal (which I discuss more in the last paragraph). You can drag and drop upload .csv files, and the app stores them on the Flask server in a "uploads" folder. You can then navigate to the "Fix" page, where the automated data cleaning/fixing process takes place. You are guided through the process step-by-step. First, you chose the dataset you want to work with. Then you chose which columns of the dataset to include in the quality evaluation. In this step (Step 2), you can also provide Python expressions for each column which will be used by the tool to evaluate whether the data in the column is valid or not (you can leave this empty too).

The tool then automatically runs data quality checks along four criteria: (1) missing values (2) outlier values (3) values not conforming to the validation criteria provided by the user (4) duplicate values. The tool evaluates each column on these four criteria and then lists out each issue it detects and the column where it was detected in Step 3. The user can then click through each of these identified issues. As they do so, they are presented with three possible fixes for each column's erroneous values: (1) multiple imputation of the average value of the column (2) multiple imputation of "-1" (3) deletion of rows with erroneous values. For each of these proposed fixes, the tool provides a graphical representation (histogram) of the distribution of the column values currently and what they would look like if that specific proposed fix was implemented. This allows the user to see whether choosing one data cleaning strategy over another will result in un-biased and un-skewed data which may create the potential for biased models or otherwise incorrect applications of the data.

The tool also automatically detects whether erroneous values in each column are correlated with values in other columns. For the criteria of missing values, this is effectively identifying "Missing at Random" (MAR). However, the tool does this analysis for all four types of data quality issues it considers, not just missing values. That is, the tool identifies if erroneous values of X are correlated with values of Y. It does so by turning non-missing values to 0's and missing values to 1's, running a chi-square test against each of the other columns, creating a contingency table, and then extracting a p-value (using the Pandas crosstab and chi2contingency functions) and seeing if it's below the alpha = 0.2 threshold I set (I found this to be the most accurate threshold empirically). If the tool identifies a correlation, it alerts the user and provides them with the column(s) which are correlated with erroneous values and the specific values in those other columns which are present in rows with erroneous values.

For each issue that AnaLazy identifies, it allows the user to select one of the three proposed fixes for each of the issues and add them to a “FixQueue”. Users can choose to not address any of the issues which AnaLazy detects if they want. Then, users give AnaLazy the order in which to execute the various fixes they’ve queued (in case the order of execution is important) and executes all of the fixes.

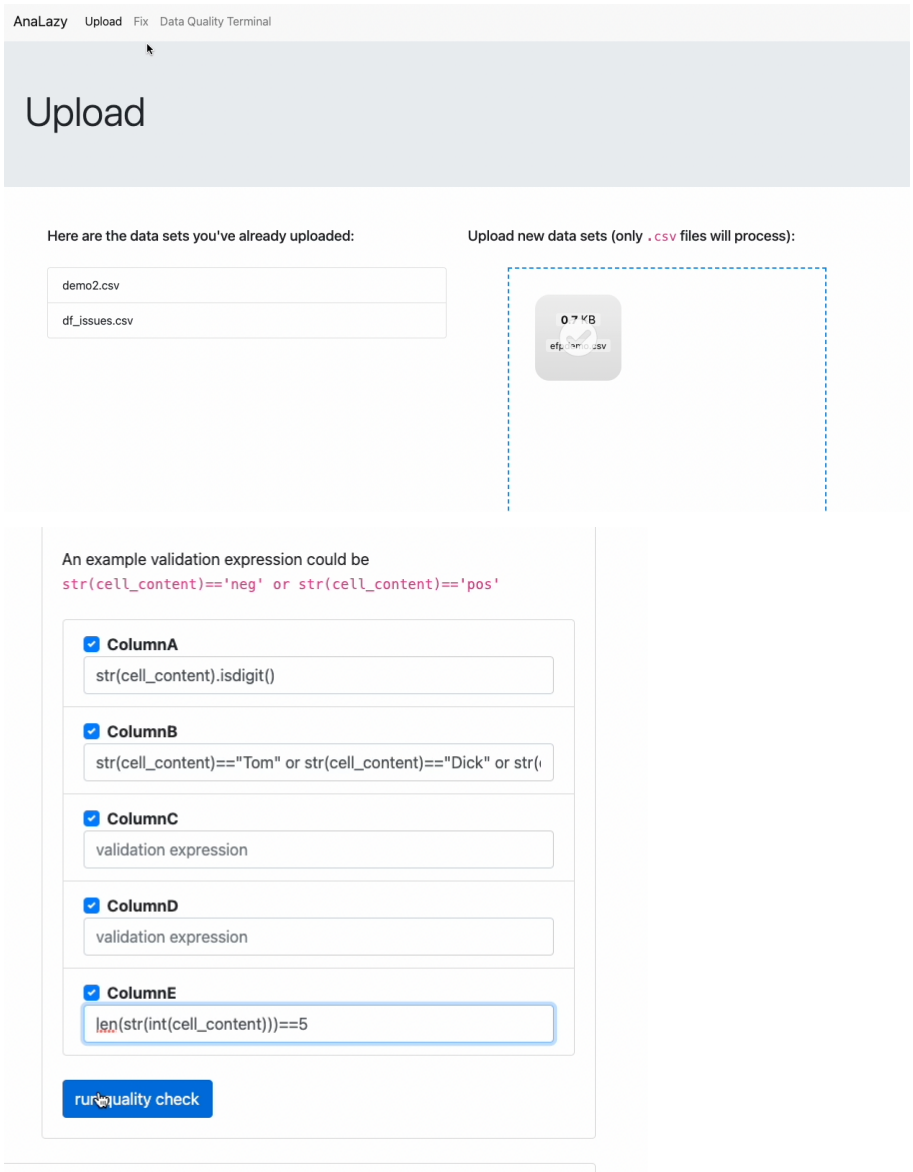
AnaLazy then automatically downloads the fixed data into a new “out.csv” file, and also generates a PDF report which summarizes each of the fixes the user asked it to make, along with the impact it had on the distribution of the data (through similar graphs as earlier).

The other component of AnaLazy which is important to highlight is the Data Quality Terminal. Before meeting with Blase about my project and focusing on a few specific data quality issues, I had built out a border data quality evaluation toolkit, which included functions to evaluate the accuracy, completeness, consistency, currency, and uniqueness of a dataset. Since I didn’t build this into the automated data cleaning process, I still wanted to allow users to use these functions to get a better understanding of their data. To facilitate this, I built a Data Quality Terminal, where you can run Python code to use all of these functions by typing the “/run” command in a terminal-like interface within the web app. I send the inputs to the “terminal” (which is just a form) to my Flask server, which evaluates the code using the Python eval() function, and then displays the result in the web app. I have attached a screenshot of this interface in the images section at the end.

Because I had to build a front-end web app to drive this entire process, as well as build out all of the data operations which enable the issue identification and cleaning itself, and figure out how to transfer data between the two, this project was incredibly time-consuming for me. I spent upwards of 30 hours total on this project and more >1,500 lines of code. I learned a lot about the Flask framework and also how to embed data-relevant content like tables, graphs, etc. in a Python web app. I have attached a .zip file of all of my code if you would like to look through it. To run the code, you can unzip all of the code, ‘cd’ into the root of the project, and run “python3 index.py --web --debug” (or equivalent). Including the --web and --debug flags is necessary, I was trying to get it to work as a desktop app (but couldn’t get that to function properly), so to run it in the proper web app form, you need the flags.

**Images**

I’ve included various screenshots on the following pages highlighting important AnaLazy functionality in case the demo video is inaccessible:


AnaLazy
Upload
Fix
Data Quality Terminal
Upload
Here are the data sets you've already uploaded:
demo2.csv
df_issues.csv
Upload new data sets (only .csv files will process):
An example validation expression could be
str(cell_content)=='neg' or str(cell_content)=='pos'
ColumnA
str(cell_content).isdigit()
ColumnB
str(cell_content)=="Tom" or str(cell_content)=="Dick" or str(
ColumnC
validation expression
ColumnD
validation expression
ColumnE
len(str(int(cell_content)))==5
quality check

### 3. Explore Detected Quality Issues and Proposed Fixes

**Select Issue to Review**

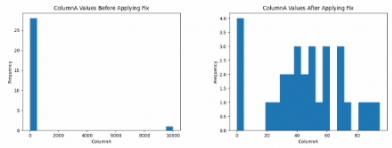
- ○ Outliers in `ColumnA`
- ○ Data validation issue in `ColumnB`
- ○ Missing data in `ColumnC`
- ○ Missing data in `ColumnD`
- ○ Outliers in `ColumnE`
- ○ Data validation issue in `ColumnE`

review this issue and fix

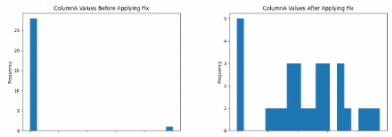
**Review Fix for:**

**Outliers in `ColumnA`**

**Detected Issue**
We detected numerical outliers at indices **[11]**
**Proposed Fix #1**
Fill all erroneous cells with the average `ColumnA` value
**Impact of Proposed Fix #1**




**Proposed Fix #2**
Fill all erroneous cells with -1
**Impact of Proposed Fix #2**



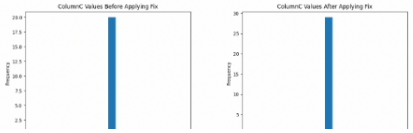

**Review Fix for:**

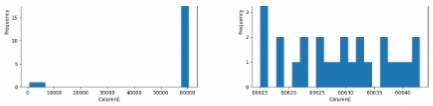
**Missing data in `ColumnC`**

**Detected Issue**
Data was missing in this column at indices **[0, 3, 6, 12, 15, 18, 21, 24, 27]**

We found a correlation between erroneous values in `ColumnC` and the values of the following column(s): `['ColumnB']`. `['ColumnB']` values of **['Tom']** were associated with erroneous `ColumnC` values. You may want to review the erroneous indices and fix manually.

**Proposed Fix #1**
Fill all erroneous cells with the average `ColumnC` value
**Impact of Proposed Fix #1**




Fix #1

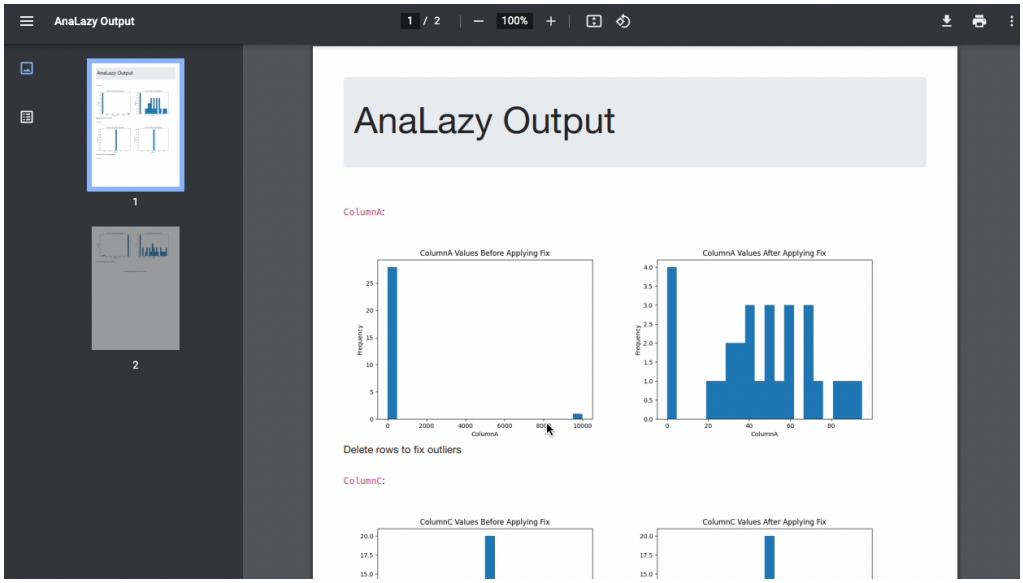
Add to FixQueue

### 4. Confirm Fixes

Enter the order in which you want these fixes to be executed. If you would like any of these fixes not to be run, enter `-1`. If you would like to add more fixes, do so in Step **3**. Please ensure **every box is filled**.

| | |
|---|---|
| 1 | `ColumnA`: Delete rows to fix outliers |
| 3 | `ColumnC`: Delete rows to fix missing data |
| 2 | `ColumnE`: Fill with average to fix outliers |

run fixes




## Data Quality

You can run normal Python in the below console — to run something, use the `/run` command followed by the snippet to be run
To store a variable, use `session['var_name'] = var_val`
For more information about all of the functions included in the data quality analysis toolkit and usage examples, see the AnaLazy Data Quality Toolkit docs

```
In >
Out > Data Quality > command request sent > 'session['qual_check_df'] = pandas.read_csv('uploads/testfile6.csv')'
Out > Returns:
Out > completed
 In >
Out > Data Quality > command request sent > 'empty_counts(session['qual_check_df'])'
Out > Returns:
Out > {'X1': 4, 'X2': 2, 'Y': 2}
 In >
Out > Data Quality > command request sent > 'col_empty_locs(session['qual_check_df'], 'X1')'
Out > Returns:
Out > [7, 29, 33, 46]
 In >
Out > Data Quality > command request sent > 'col_form_check(session['qual_check_df'], 'Y', "str(cell_content)=='neg' or
      str(cell_content)=='pos'")'
Out > Returns:
Out > [9, 32]
Enter > /run col_empty_locs(session['qual_check_df'], 'Y')
```