Palace Law Prediction Algorithm – Brandon DeAvilla

**Background:**

I took an existing algorithm created for Palace Law that was utilizing data input from Airtables and outputting a QScore for clients. I adjusted the algorithm to utilize Google Sheets as the source of the data, and worked toward a solution to automate the process where the QScore will automatically appear on the same Google sheet.

The algorithm uses data from past cases, and whether Palace Law took the case or not, to determine if they should take future cases, explained via a QScore. This model will run automatically in the background and will run overnight for any new client data input into the google sheet.

**Creation Process**

**Framing the Problem:**

The problem that is being solved with this project is to adapt a previously created algorithm to be used with Google Sheets and actually be able to run data through the model and have it posted directly to Google Sheets.

Due to some issues with Google Sheets, the plan is to have any new data which is input into the sheet, run through the model at a scheduled time (during non-operating hours).

The data that is written to the spreadsheet is written to a separate sheet in the workbook and therefore, the model will never touch the actual live data.

The idea of this process is to allow Palace Law to quickly determine if they wish to take on a client, and the likelihood that the case is a case worth taking.

Palace Law also mentioned wanting to include financial information in the model, which was unable to be done at this time, but definitely can be used in the future to further grow the model.

**Research, Ideation & Prototyping:**

Discussions with Jordan from Palace Law helped us come up with a game plan to make this model something useable for them. We discussed via email, and over the phone the process in which to execute this plan.

We came to the decision to migrate everything over to Google Sheets, as the Law Firm already uses Google Sheets. A lot of research went into using the Google API in order to accomplish the read/write commands that we needed for the model.

**User Testing & Refinement:**

**STEP ONE:**

The first step in this process was me speaking with Palace Law and figuring out exactly what they wanted. I spoke to them about the previous model and what they wanted in order to make the model that I was creating more useable. We discussed moving the model to Google from Airtable, and also make the model more automated.

To start this, I needed to get the login information to Airtable and to the Palace Law Google account. Once I attained access, I began to migrate the tables used in the previous project over to Google Sheets (with up to date data).

After actually moving the data to Google Sheets, I had to discover a method to read data from Google Sheets to Python. This required using a package called “GSpread”. Unfortunately, there were a lot of issues regarding the authentication process that this package required. I had to learn to setup projects with Google and allow authentication with a .json file. After figuring all of this out, I applied it to the current project and was able to get the google sheets to appear inside python. (Two weeks after working through all of this, we actually learned how to do this in class).

Once the data was finally loaded into python, I needed to properly merge the data. I merged two of the sheets at first, and after noticing a very poor F1 Score (explained below), I noticed there was a third spreadsheet that needed to be incorporated. The addition of a third spreadsheet created a lot more columns that were not properly cleaned, this required redoing all of the data cleaning. Other efficiency additions were added after noticing the poor F1 Score, such as dropping unnecessary columns before merging the data. This helped stop the loss of data while doing a 3-way merge. This method resulted in keeping the correct amount of data and a lack of data loss from the merge.

**STEP TWO:**

Once the data was properly merged, it needed to be cleaned. Like stated above, I did a lot of the cleaning of the data before the merge in order to make a cleaner merge. The merge required a three way merge due to how the data was split. Data 1 had all of the client data for phone calls, the data was then split into Data 2 and Data 3, depending on the type of case it was. To properly merge the data, I cleaned it as necessary, and merge Data 2 into Data 1 on the ID column, and then merged Data 3 into Data 1, also on the ID column. Once I had these two separate merged dataframes, I merged them into each other, creating a merged and cleaned copy of the data.

We then needed to identify all of the columns that contained dates, and to properly mark them as such. This was fairly easy, and I was able to adapt previous code to work with my new data. Similar to the Gut Challenge and the existing code, we calculated out the age of the client, time since injury and the report delay into a day counter, so that it could properly be input into the model. Once these numbers were generated, the originating columns were dropped.

Some more data cleaning was required after the initial cell drops. More specifically, was reducing a lot of redundancies in the injury types (hand, hand(s), left hand, right hand, etc.). Once these were all reduced down, it created a much cleaner, and easier way of reading the data. We then split the Body Part columns into individual body part columns, to make the model specific on injury type.

I then turned variables that had two types of answers into 1’s and 0’s, such as Male or Female, or Yes/No type questions. These were done like this because the model will only accept numbers.

**STEP THREE:**

Once the majority of the cleaning was done, it now came time to push the data into the model. This required verifying the columns were all necessary columns in determining whether a case should/shouldn’t be accepted. (At the time of writing this, further cleaning of data is recommended with the ‘Line of Work’ and ‘Nature of Injury’ columns, similar to what was done with the body part columns).

Once the columns and features were verified, they were ready to be input into different models to create a specific model for Palace Law. There were a lot of NaN values (due to no dates being input) that if removed would drastically reduce the amount of data that the model had to learn.

A workaround I did for this was generate a mean within each individual column to input the data. This process was done because though it introduces manufactured data, the benefit of being able to include the remaining data vastly outweighed the added data.

**STEP FOUR:**

We then input the features into the model, like stated above, when this was first done, the F1 score output was very low. This required a lot of restructuring of the data because of a missing spreadsheet, which we were able to eventually get working. This was input into multiple types of models, the one with the higher F1 score was selected to be used. This created the model to which we are going to be able to input new data through and generate a QScore, or a likelihood to take a case!

**STEP FIVE:**

I then worked on the next segment of the code, which is referred to as ‘Pickle’. This required the same processes as above, with receiving the data from google sheets, and then cleaning it to match the features that are in the model. The key part of this though is to have the input data only run the data on the spreadsheet that does not currently have a QScore. We did this by only displaying rows that had “NOT CALCULATED” (explanations of the automation for this will be explained below). I at first had the filter run at the beginning of the model, but soon discovered that it will not have the same amount of features (because obviously one case will not have every type of injury). The workaround for this required me to run all of the data through the cleaning process again, and filtering out the actual data right before it is input into the model.

**STEP SIX:**

Next, I had to prep the spreadsheet for the above. This involved adding a QScore column on Data 1 and also creating a new sheet within the workbook for the generated QScores and ID numbers to be input.

In order to properly display the QScore on the main sheet, I used a VLookup to place the QScore onto Data 1 based on the ID number that is input into the newly created sheet. This allowed an accurate display of QScore from the model onto the google sheet. If no QScore was there it would input “NOT CALCULATED”. I dragged this VLookup formula down about 5000 cells to future proof the workbook for a while.

After creating this formula and going through testing, I discovered that the input data from the google sheet was now around 10,000 rows. While investigating this, I discovered it was because the future proofing that I did above caused “Not Calculated” to be placed along every single row.

I did research on Vlookup methods and eventually found a way to use multiple if statements for the formula to check if there is an ID number first, and if not, put a blank entry, otherwise, it would run the vlookup formula.

**STEP SEVEN:**

Once the spreadsheet was properly prepared, it was time to work on running the data through the model. A lot of meticulous issues arose from this, from not matching the feature count (due to blank columns), to other errors. Once worked through, we were able to get the model to generate a QScore and pair it up with the corresponding ID number.

Now that we have this data, we need to find a way to post the data to the Google Sheet in order to utilize our Vlookup formula. I did research regarding this and found a package library that was perfect for this. I installed the package and it wrote all of the data to the google sheet in one post, which was perfect.

I then began restarting the kernel to run a full test on the model, and I ran into an issue with the authentication of the gspread package we use to read/write to Google Sheets. This broke all of the models, and required a lot of research. I worked on uninstalling and reinstalling packages, assuming it was one of the new packages that were installed. This did not resolve the issue, and I was at a loss for the resolution. Luckily, I met with Prof. Colarusso, who assisted me with deleting packages from the root in pythonanywhere, and reinstalling. This resolved the issue with the authentication but required finding a new method to write to Google Sheets.

Upon researching this, and looking into an old class project that we did, I was able to find a method to write to Google Sheets. It required a lot of adjustments off of the initial gspread code, but I was finally able to get it to work. The only difference that this write had compared to the old write package was that this wrote each row individually. This became an issue as the Google API would begin to block the model whenever it exceeded more than 100 writes in 100 seconds.

To bypass this, I put a 1.2sec delay on each row write (which if there are ever bulk uploads, this will be a non-issue as the plan is to run this script via scheduler at night).

We ran a test and were able to get the data to properly write to the google sheet. The vlookup is working as intended!

**Product:**

**Intro Pitch:**

During class I provided a pitch of my final project. During this, I used a PowerPoint to outline my plan for my final project to my classmates. The PowerPoint can be found at <https://bdeavilla.github.io/ctl/>Palace%20Law%20Presentation.pptx

**Impact & Efficiencies:**

The idea of this project is to adapt an already existing project to be easier to use within the firm and more automated. The methods that I designed will run their data through the model, like before, but will allow them to move all of their information to Google Sheets (where all of their other data is). This model now will allow them to input client data into their spreadsheet (like previous), but when they come in the next day, the data will have a calculated out QScore for them to compare to and see if they would like to listen to the score.

The beauty of this as well, is that it is all located within one spreadsheet. So as they add to the spreadsheet and it generates QScores, and they determine if they Accept the matter or not, it is also adding to the model to make it more accurate in the future.

**Real World Viability**

At this point in the project, I believe it is semi viable for real world use. I believe that the algorithm needs some more tweaks to make it more accurate to how Palace Law actually makes decisions. It also requires a more uniform style of data entry on Palace Law’s behalf.

A suggestion I would make is to create dropdown windows within the Google Sheet in order to have consistent decisions and in order to guarantee that the model will not break. I’d suggest these drop downs on body parts, line of work, and nature of injury. Having consistent data will allow this model to be more accurate in the future, and also future proof, so that incorrect entries will not break the model.

I would suggest that we have Jordan and Palace Law switch over their data entry into the Google Sheet data, and begin to see the QScore and make notes on the scores that are generated and their actual decisions. This will allow for the next years student to adapt the model and make suggested changes as mentioned in this write up.

**Sustainability:**

As stated throughout this write-up, the model is near autonomous and will need to be set on a schedule through Pythonanywhere. The sustainability of this model is based on Palace Law switching over to Google Sheets and inputting new data onto the sheet as necessary. This model is also wholely dependent on the PythonAnywhere subscription being paid and continued in use.

As stated above, the model can definitely be tweaked and adjusted and I believe the best method of sustainability for this model and process is to have uniform answers via dropdowns so that the model can easily be adapted and grown.