**Scott Schmidt; Data Science Finalist; October 2021**

**Python Data extrapolation, analysis:**

At Illinois State University, I invented a Python data science pandas dataFrame automation program that

compares thousands of data transactions to database records which detects unbalanced accounts, new transactions, and human error: <https://github.com/ScottFrederickSchmidt/AccountingAutomation>

This project involved a hundred lines of Python code with most of it involving data manipulation and algorithms. For instance, the hyphens in the real data had to be temporarily dropped to match the Access database records. Accounting records had to be grouped together. This was an important task because it involves accounting. If the program is not 100% accurate, one could get involved in an audit that could cost the company thousands of dollars and a bad reputation. In accounting, one must be exact. There are no guesses. Accuracy, attention to detail and perfection is of highest importance. My accounting software can verify if an account is not balanced with 100% accuracy. If my accounting software was implemented by a company doing accounting by paper/pencil, it would save a company up to 12% a year in accounting expenses. It would also not automate jobs but make the job of an accountant more enjoyable (increasing employee retention rates).

Likewise, at Douglas Capital Management I programmed Python financial automation using pandas, ib\_insync, and operating systems (os). One slight mistake with the trading bot could mean a loss of money if the bot gathered incorrect information using the ib\_insync package. It could also mean accidently selling something for a gain and then a customer having to pay unwanted taxes! Also, Python was used for webpage data scrapping using BeautifulSoup (bs4), requests, selenium, webdriver, and to\_csv. For example, I developed a weather bot to help get the average temperature across major cities in the United States to help determine the fair value of the price for natural gas depending on the weather during winter. Also, one must make sure that the data is correct. For instance, a stock split could easily distort data if the data does not count for stock split adjustments. Another example of understanding data would be if a job title changed titles ten years ago. To gather the most data, one would have to know this information to use that previous job title data. Otherwise, the data will not be using lots of datapoints that could have been collected in the predictive model.

Coding efficiency, queries, and data manipulation is my strength as I have over one thousand hours in Python/SQL data manipulation. For instance, my first solutions to problem 44 took 86 seconds for my Python code to get the correct solution. However, all solutions should be solved in under ten seconds. After putting many hours of changing the logic (i.e. when for a greater number is searched and adding in a function), I was able to get the program code to run and find the correct solution in under two seconds with a new, revised solution. My data manipulation training which be viewed on both GitHub and YouTube:

<https://github.com/ScottFrederickSchmidt/LeetCode> <https://github.com/ScottFrederickSchmidt/ProjectEulerPython>

**Predictive Python Models:**

My data science predictions are in Python using pandas, numpy, and sklearn packages. Classification models are logistic (linear) regression, KNN, random forest, and SVM: <https://github.com/ScottFrederickSchmidt/PredictivePythonModeling>

Predicting stock market returns was the entire goal at Douglas Capital Management. A database from Portfolio123.com was used with around a hundred finance formulas to predict future returns. For instance, OperCashFlTTM - CapExTTM + IntExpTTM\*(1-TaxRate%TTMInd/100) would be inserted into a database with an optimization process to see how this one formula can affect future returns. More important formulas such as annual growth rate would have a higher significance for returns. Therefore, we would weight growth rates higher than per say dividend growth rate. We also had to account for missing data. On occasion, formulas had to be completely deleted if there were too many missing values for analyzing microcap stocks. But most of the time, missing data was filled in using an estimate by a regression line. Prebuilt basic MATLAB functions would be used to help find the statistic relevance for each formula. In 2019, our aggressive trading hedge fund return was in the top ten. During this time, I also designed Python financial automation using pandas and os that would teach an AI bot when to place a trade.

During my MBA program as an ISU graduate assistant, I generated a 95% r-squared using data analytics (depends on adjustment to overfitting). This identified the best candidates that would most likely successfully graduate at ISU. This was the final MBA capstone course, MQM497. In conclusion, Pell Grants and number of initiated contacts are the two biggest factors in whether a student enrolled at the university or not. These two factors alone can make a good prediction on whether a student enrolls at the university or not. In conclusion, neural networks show that undecided majors, Pell Grant, distance from ISU, # of initiated events, family income median, total consumer expenditures, high school low-income percentage, and high school GPA are the major factors in whether a student enrolls at Illinois State University or not. The test confirms that ACT, GPA, major, Pell Grant, and # of initiated events make a significant difference in students enrolling at Illinois State University.

**SQL Database Query:**

To write an effective SQL query, one first needs to understand the data. Once the data is understood, the criteria must then be identified. Then, one can now create an efficient SQL query to find the desired results. Here are some ideas of producing an efficient SQL result:

* The larger the table, the longer it takes to read the data.
* Using a JOIN statement will increase the runtime. As such, I am careful to use only the tables needed. One should try to only use the columns needed. Using inner join can limit the rows brought into the data to only what is needed.
* Using subqueries should be used whenever possible before using a JOIN statement if the subquery is returning a single value.
* One should never use the \* operator. Including specific column names enhances performance. It also makes the code more reusable in the future (if more columns get added later). Using the \* function is also a security issue. SQL can accidentally gather confidential information such as addresses, studentID, age, birthday, and confidential information into an csv file when it should not be there.
* One should not repeat code. In my automation project, I made sure to never use a line twice. Temporary tables should be used to speed up efficiency instead of having multiple subqueries referring to the same table.

On my “Amazon” example website, I built an entire SQL database that was on the backend connected to the frontend using PHP. In addition, it has a very basic SQL search and recommendation section. All these SQL queries happened with high performance speed: <https://github.com/ScottFrederickSchmidt/AmazonSite>

SQL coding data manipulation can be viewed here:

<https://github.com/ScottFrederickSchmidt/threeSQLcourses>

<https://github.com/ScottFrederickSchmidt/LeetCode>

**Treating Missing Values:**

The simple, easy way to treat missing values is to simply delete any data row that has a missing value using a simple Python script such as df.dropna(). This is my preferred method because there are no assumptions made. Another way is to find the mean of the data and assume that the missing value is that number. KNN nearest algorithm can be used to impute data. A bad choice would be to fill the missing data with a 0 because this would distort the data. Lastly, if there is not a lot of data points with lots of missing data, one should consider removing that column completely.

**Data Science Python Coding Sample**

**EXECUTIVE SUMMARY**

In quick summary, linear regression and KNN were the best two models I found. In this exercise, the KNN generated an AUC of 0.9595. The logistic regression produced an AUC of 0.6216. I additionally performed a linear regression which produced a 0.9189 AUC. Therefore, the KNN and linear regression provided better results than the logistic regression. However, I believe I could improve the logistic regression results by finding certain columns that might not be relevant and giving the better columns a higher value. In addition, I added a decision tree, support vector, and an automation project. These models will be discussed individually later and how they could be improved for better AUC later.

**LOGISTIC REGRESSION (GLM):**

Logistic regression requires a lot of datasets. In this case, it was not an issue as there were 10,000 rows for less than 100 features. Some believe that there should be 10x more rows than features. This fit this description. Overfitting is an issue if there are more features than datapoints. One disadvantage of logistic regression is having to sometimes scale features.

Logistic regression code here:

<https://github.com/ScottFrederickSchmidt/PredictivePythonModeling/blob/main/InterviewLogisticRegression>

**KNN Classification (SML):**

The major disadvantage of KNN is slow runtime since KNN has to calculate distance between points.

When there are more than 50,000 rows, KNN can become slow. With only 10,000 rows, I knew that KNN would not have a runtime issue in this case. This made KNN the perfect second choice for my classification model.

Also, KNN is sensitive to missing data points and outliers. The data was successfully cleaned using simple techniques for now. My error\_rate said that 3 was the best performer for n\_neighbors. KNN initial precision was 0.894, however, this number was a bit decieving because I had used model = KNeighborsClassifier(n\_neighbors = 5.) When I tried model = KNeighborsClassifier(n\_neighbors = 7), precision dropped to 0.82. KNN code here: <https://github.com/ScottFrederickSchmidt/PredictivePythonModeling/blob/main/InterviewKNN>

**LINEAR REGRESSION:**

As expected, the linear regression performed better than logistic with a 0.9189 AUC. Linear regression is a simple, popular, and powerful model. One disadvantage of linear regression is it is prone to overfitting and outliers. However, the model performed well even without removing any outliers. Linear regression code is found here: <https://github.com/ScottFrederickSchmidt/PredictivePythonModeling/blob/main/InterviewLinearRegression>

**RANDOM FOREST:**

Generally, random forest is one of the best models to use. The decision tree AUC is 0.3788 but the precision was .93. This is a very rare occurrence that should not happen. After looking at the Python code, there was no real answer of why the AUC would be so low. However, underfitting can often occur with decision trees. Therefore, one must carefully select the features carefully when using a decision tree. Using specific features would likely increase the AUC to a more natural probability. Random Forest code here: <https://github.com/ScottFrederickSchmidt/PredictivePythonModeling/blob/main/InterviewRandomForest>

**SUPPORT VECTOR MACHINE (SVM):**

The AUC for SVM was .50 which is not a very good result for this exercise.

Normally, SVM is a very powerful and complex model that works well if there are sufficient data points. One disadvantage with SVM (and neural networks) is that it is much more complex and be extremely difficult to explain to others. Often, doing a simpler linear (logistic) regression, KNN, or random forest is better simply because its simpler.

<https://github.com/ScottFrederickSchmidt/PredictivePythonModeling/blob/main/InterviewSVM>

**AUTOMATION PROJECT:**

I invented an automation data science Python project to find the best feature variables (columns x1 to x100) that have the highest significance by taking each column individually to compare it to df['y'], the independent variable:

For this project, I used r-squared because it is my favorite metric since it combines probability with number of samples.

At the end, I stored the results into a dictionary with a key and value. Then at the very end, I print out the five highest r\_squared values within the dictionary. An additional cool project to do would be to use a Python intertools permutations script to calculate every combination for each column to detect which combo provides the best outcome.

<https://github.com/ScottFrederickSchmidt/PredictivePythonModeling/blob/main/InterviewAutomation>