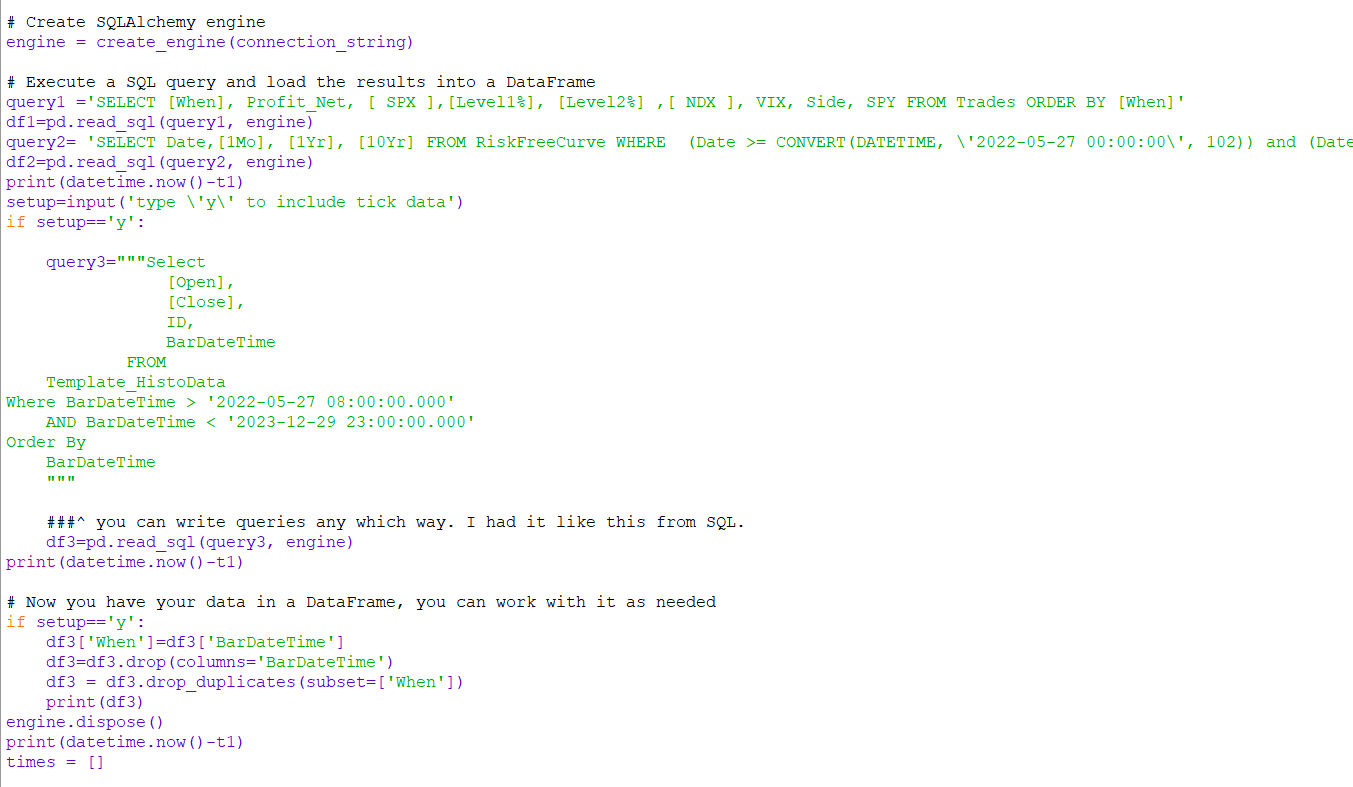
Progress Report 1 Technical Appendix

Group A (Tartigrade)

Colton Code:

**New mechanism for connecting to SQL for Python**:For the ability to efficiently pull and upload data to SQL, we had to switch methods and install ‘SQLalchemy’ instead of ‘pyodbc’ as seen in the previous appendix.

*A1:*



Pseudocode:

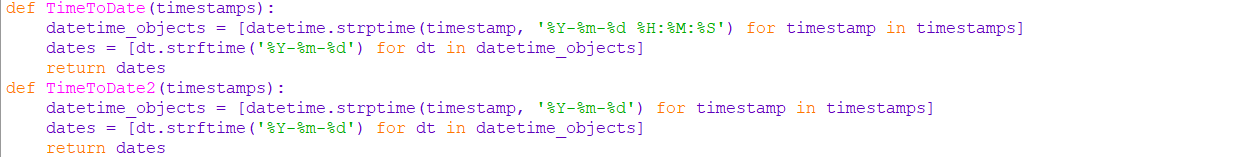
1. Create connection string (server, database, username, password)
2. Create engine using the connection string
3. Create your queries written in SQL and made into a string specifying what data you want from which data base

3a. Select if you wish to import data in the 5s form (this takes a while and using the smaller data is still helpful for finding relationships. Additionally, analysis is skipped for this data. Selecting this option changes how the code functions)

1. Put queries into pandas data frames using the engine

**Various Cleaning Techniques**: The data coming from SQL often had missing values. However, these were not always in the form of NaN. Sometimes the values were filled but wrongly so. Many different approaches to cleaning were needed to make the data ready for analysis. Dropping rows with a missing value was to be avoided if possible. All cleaning techniques were on the smaller data frame. When merging the smaller data frame with the 5s data, all data was forward filled as that data frame had NaN values which are easily fixed with this method.

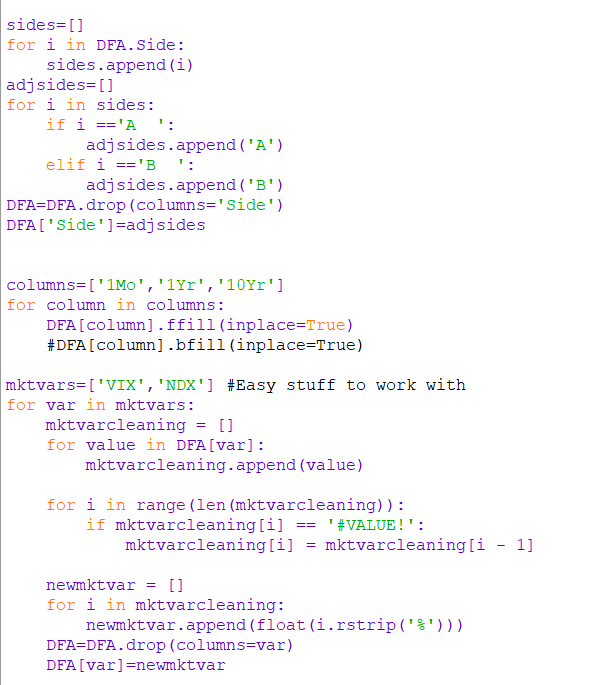
*A2:*



Pseudocode Date Cleaning (Dates from each data base on the server has dates in different formats):

1. Create a list of all times in the first data frame
2. Feed list to TimeToDate function
3. Function converts times to desired format
4. Function returns a list of times/dates
5. Repeat steps with the 2nd data frame instead feeding it in TimeToDate2

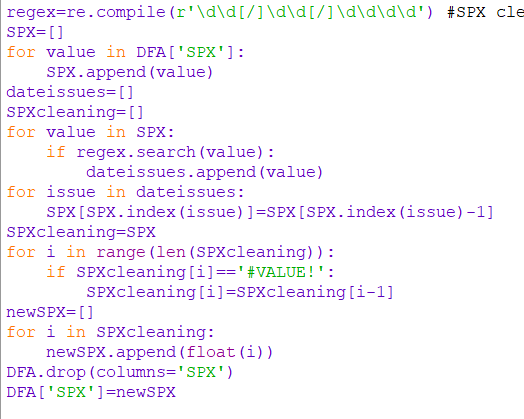
*A3:*



**Pseudocode**: Side and VIX/NDX Cleaning (two spaces were attached to each side value. #VALUE! appeared in a few rows. These are not recognized as NaN values.)

1. Put merged data frame columns into list
2. Create new list for new data frame column
3. Cleans by creating new ‘A’/’B’ values for side or replaces the ‘#VALUE!’ with the previous valid value (this works even with the 5s data where often ‘Side’ takes no value as no trade happened.)
4. Previous data frame column is deleted
5. Re-create the data frame column with list of corrected values while changing their type to ‘float’
6. (interest rates) forward fill the data to remove null values (bfill was initially included afterwards as null values were initially at the start of the dataset (this is done later for the larger data set so all variables can be included in a ‘for’ loop)

*A4:*



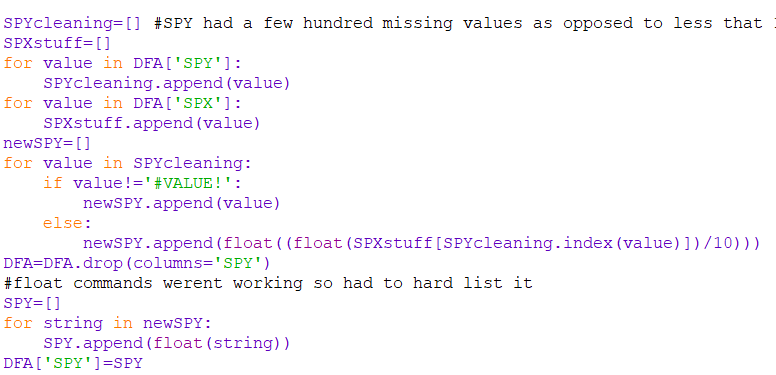
**Pseudocode**: SPX cleaning (There were dates mixed in with the values along with #VALUE!. These values accounted for only a few observations)

1. Define regular expression (all erroneous entries were in the date format written)
2. Create list and append all SPX data to it
3. Establish a list for numerical and date values
4. Iterate over the SPX list

4a. If a date is detected, it is put into the date list

1. Iterate over all dates in the date list and replace them with the previous valid entry
2. Drop old column
3. Convert all types to float and recreate the column in the data frame

*A5:*



**Pseudocode**: SPY Cleaning (SPY had hundreds of missing values amounting to around 10% of what we were given. Instead of filling, had to find another way to make it work)

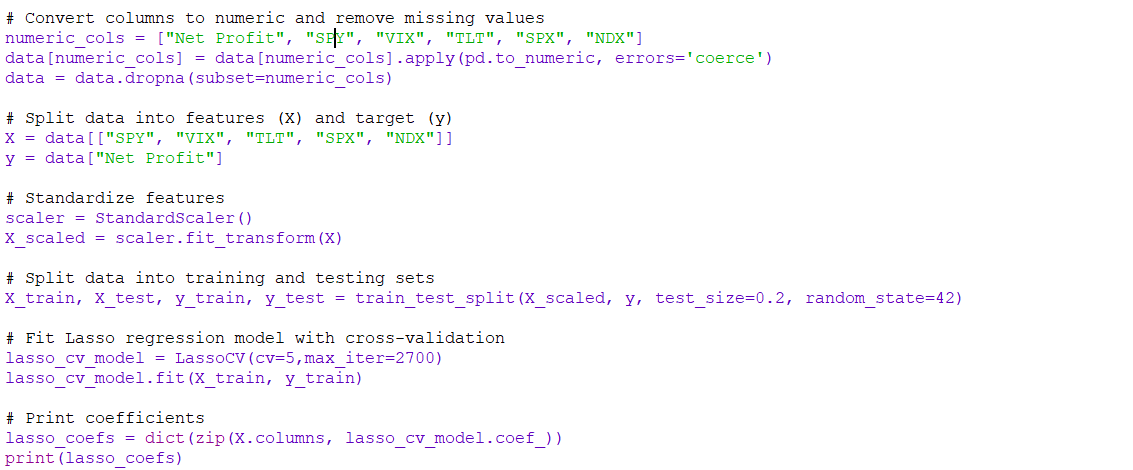
1. Put SPY and SPX values into separate lists
2. Make a new list for the cleaned SPY values
3. Iterate over SPY looking for #VALUE!. Appending to new SPY list.

3a. If issue is found, convert SPX value at the same time to SPY which is usually 10x smaller than SPX

1. Get rid of old SPY column and establish new one

LASSO coefficient usage: LASSO regression was used to find out which variables were most influential on Net Profit and Trading. As we are not creating a predictive model, data was split only into training and test sets. Though the number of variables appears manageable, every additional variable added increases the time it takes to import and export data and increases the likelihood of server errors. We aim to find a balance between accuracy and efficiency.

*A6:*

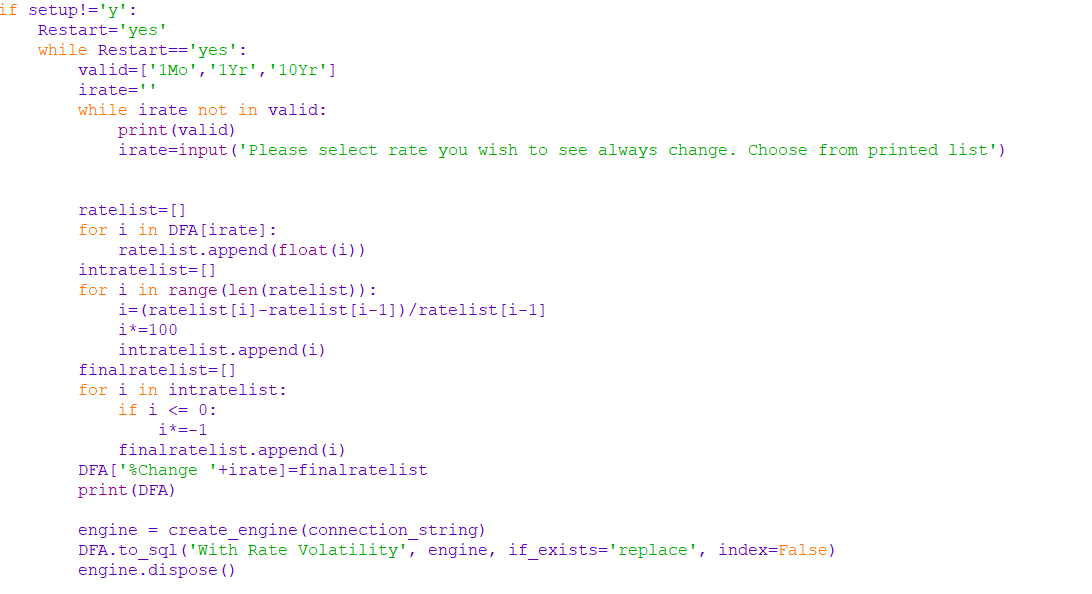


Pseudocode: Market Variables on Net Profit (Originally done in R before being brought over to Python. Then Modified using LASSOCV from sklearn to produce similar results)

1. Split data into X and Y
2. Scale the date
3. Split data into X and Y training/test sets
4. Cross validates with 5 folds (data is large so this is better than 10)
5. Perform LASSO regression
6. Store coefficients in a dictionary and display

**Interest Rate Selection**: If using the smaller data frame, an option will be given to look at the interest rate volatility for a given interest rate. Afterwards, a new table is uploaded to SQL all in one chunk as it is relatively small (~10000 rows)

*A7:*



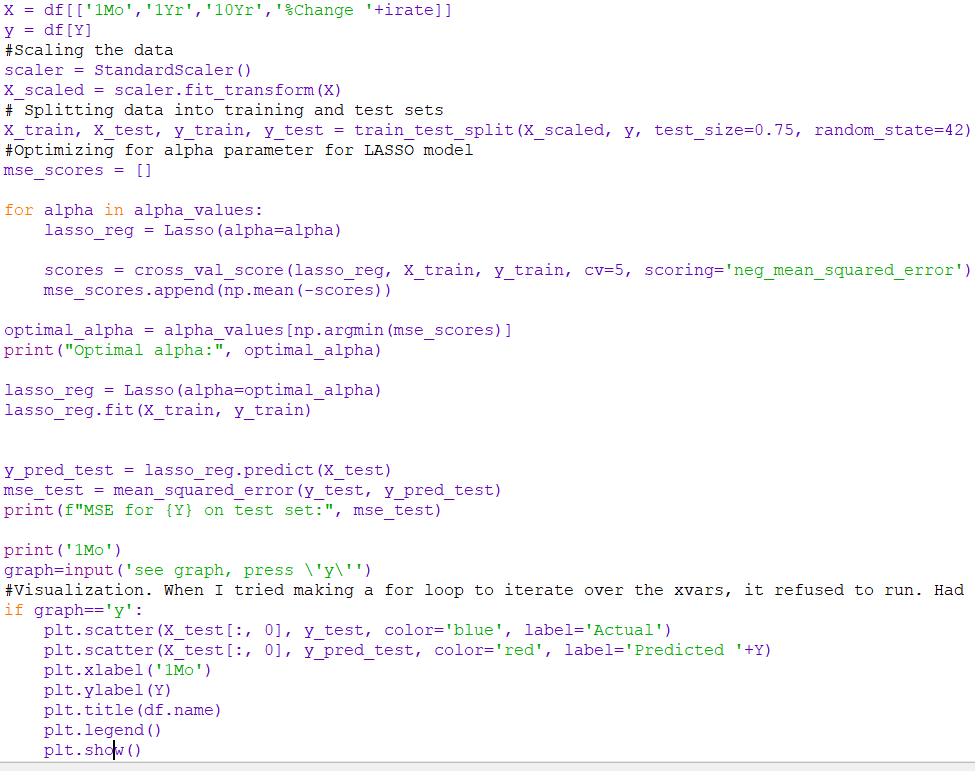
**Pseudocode**: The Restart variable is there as it is here where a loop starts so that after finishing OLS/LASSO regression on the smaller data frame, the user may restart and select a different interest rate/ regression type.

1. Select which interest rate to show volatility for (short term is better)
2. List created for rate volatility
3. Calculates the change in the interest rate for each observation.

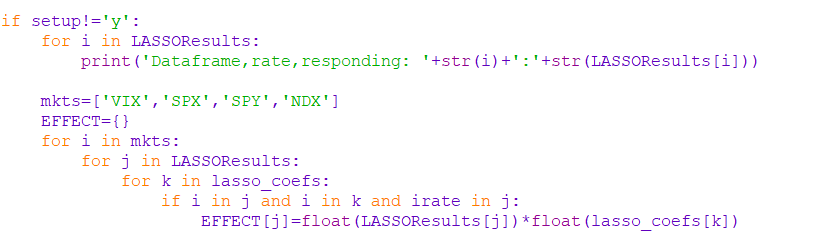
3a. If negative change occurs, multiplied by –1 (looking for absolute changes)

1. Creates new column in data frame using the list
2. Uploads data frame to SQL for visualization

*A8:*



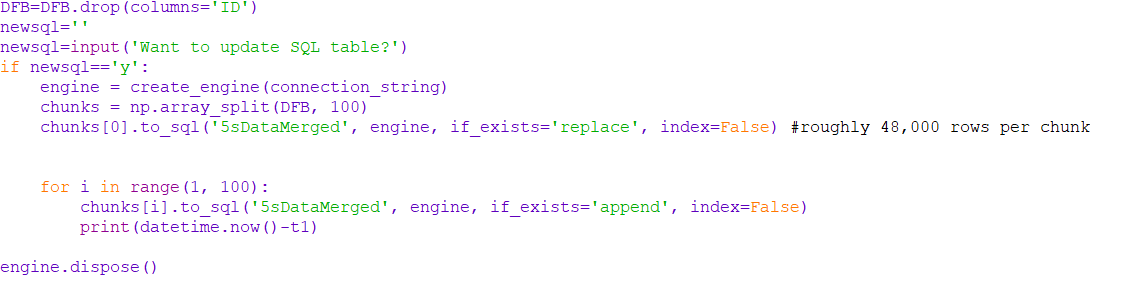
*A9:*



LASSO for interest rates/rate volatility (Lots is the same except it uses LASSO from sklearn, not LASSOCV.) This iterates over a trading data frame and one that only has times when rates change (to look at rate volatility). It also iterates the X variables. Instead of using only cross validation, it optimizes alpha to minimize MSE in addition to the cross validation. Afterwards, an optional graph may be displayed showing the actual values against the LASSO predicted ones (in slide deck). All coefficients are stored in a list. Then, the coefficients that are not related to volatility are dropped from the coefficients done on the data frame where only rate changes are happening. Afterwards, they are multiplied by the respective market variable for an indirect effect which is then added to a dictionary (in slide deck).

**Data frame upload size management**: The large data frame with 5s data is extremely large. Uploading cleaned data was extremely troublesome due to the size. A code had to be made to ensure no errors occurred. This was done by splitting the large data frame into chunks and uploading them one at a time. It takes roughly 90 minutes.

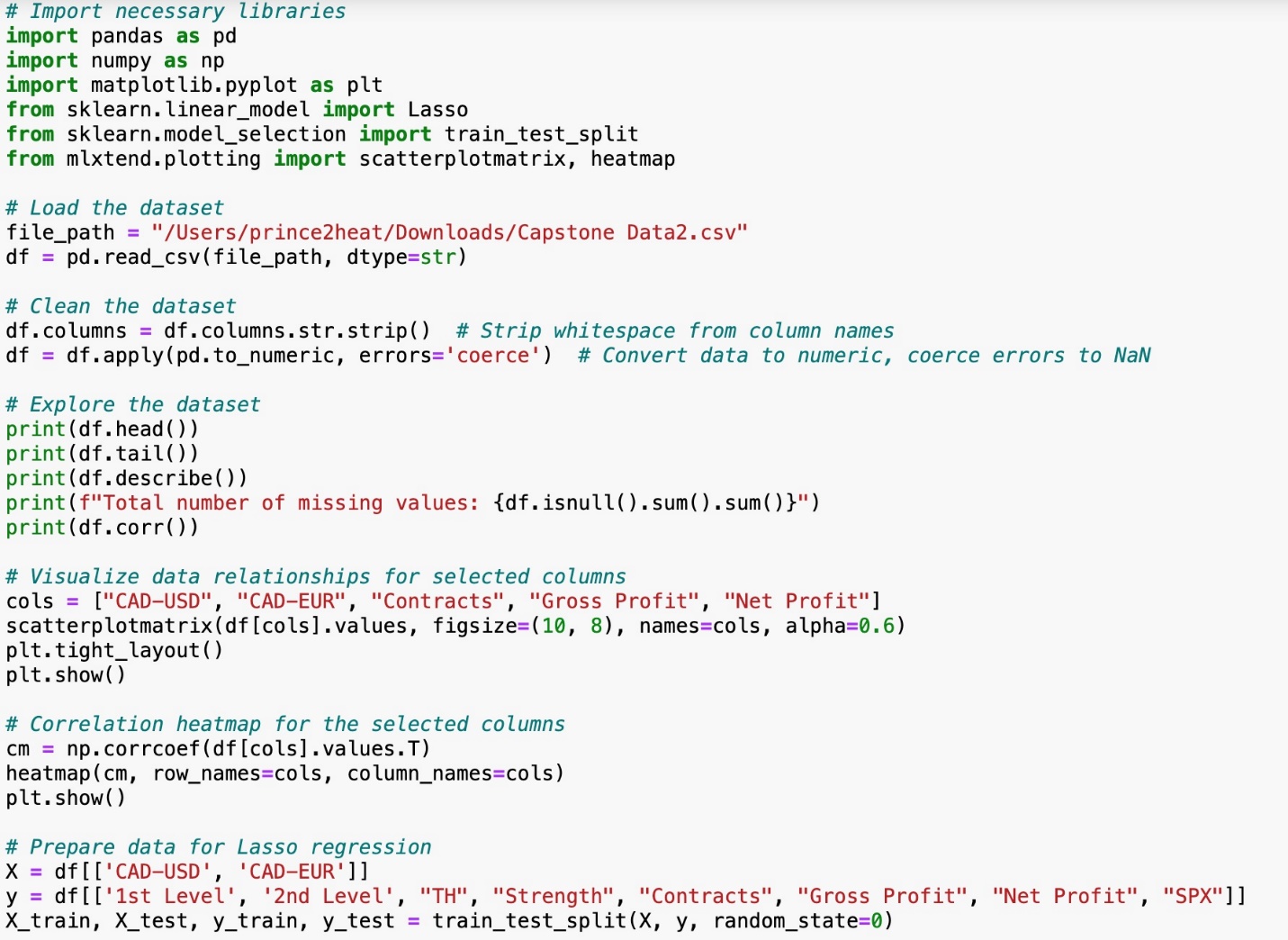
*A10:*

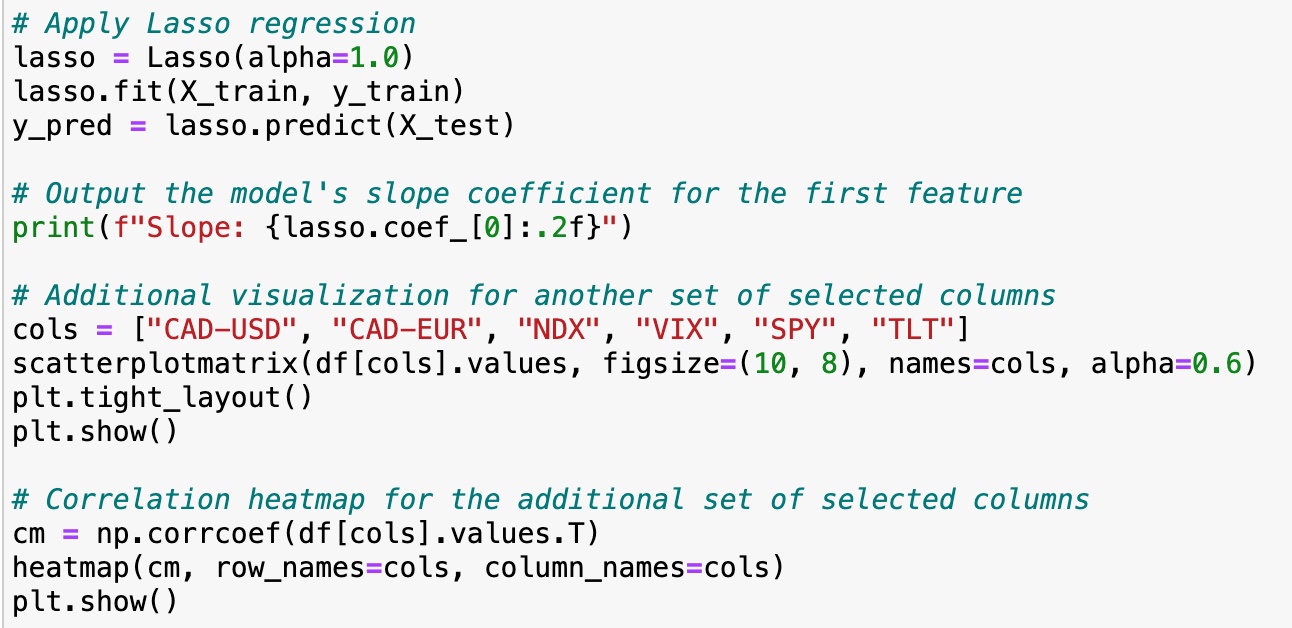


Pseudocode:

1. Drop as many useless columns as possible. Each column dropped saves 3-5 seconds per chunk. (Around 5-8 minutes)
2. Re-connect to SQL server using the same steps as above
3. Split the large data frame into 100 chunks
4. Create/Replace table in SQL with first chunk
5. Iterate over remaining chunks to complete table
6. Close connection to the server to avoid issues when running the code again

Prince:





1. Import necessary libraries:

- pandas for data manipulation

- sklearn.linear\_model for the Lasso regression model

- sklearn.model\_selection for splitting the dataset

- sklearn.metrics for calculating the r2 score

- numpy for numerical operations

- matplotlib.pyplot and mlxtend.plotting for plotting

2. Load the dataset from a CSV file located at a specific path.

3. Clean the dataset:

- Strip whitespace from the column names.

- Convert all data to numeric, coercing errors to NaN (missing values).

4. Explore the dataset:

- Display the first few rows.

- Display the last few rows.

- Generate descriptive statistics.

- Calculate the total number of missing values.

- Compute pairwise correlation of columns.

5. Visualize data relationships:

- Use scatterplot matrix to visualize relationships between selected columns: ["CAD-USD", "CAD-EUR", "Contracts", "Gross Profit", "Net Profit"].

- Plot a heatmap of the correlation coefficients between these selected columns.

6. Prepare data for Lasso regression:

- Define the features (X) as ["CAD-USD", "CAD-EUR"] and the target variables (y) as ["1st Level", "2nd Level", "TH", "Strength", "Contracts", "Gross Profit", "Net Profit", "SPX"].

- Split the data into training and test sets.

7. Apply Lasso regression:

- Train the Lasso model on the training set.

- Predict the target variables for the test set.

- Output the model's slope coefficient for the first feature.

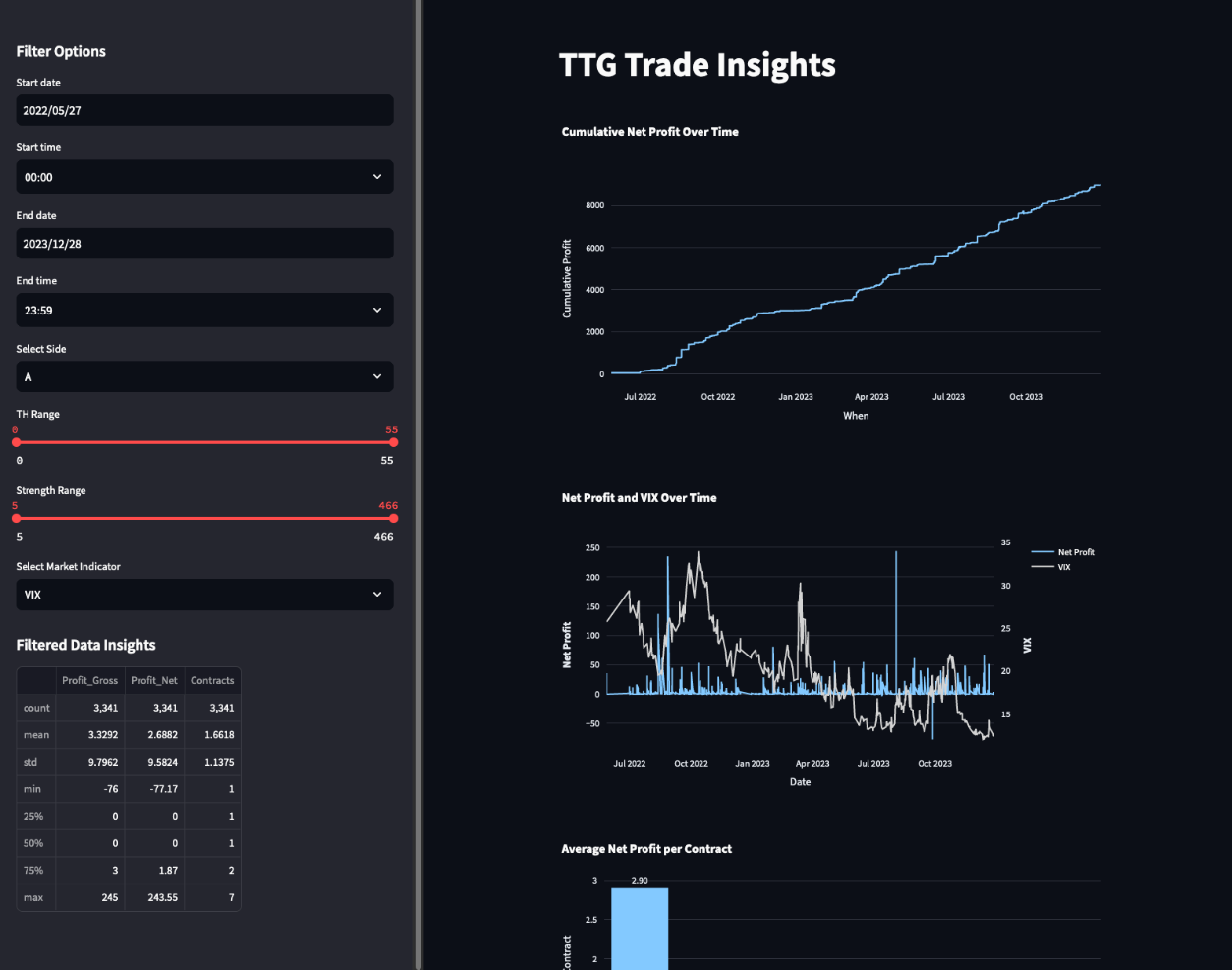
8. Further visualization:

- Use scatterplot matrix to visualize relationships between another set of selected columns: ["CAD-USD", "CAD-EUR", "NDX", "VIX", "SPY", "TLT"].

- Plot a heatmap of the correlation coefficients between these selected columns.

Ammaar

**Trading Insights GUI:**



Visualization Code Samples (Interface done using python STREAMLIT package):

# 2. Visualization: Net Profit and Market Indicator Over Time  
fig = make\_subplots(specs=[[{"secondary\_y": True}]])  
# Adding Profit\_Net trace  
fig.add\_trace(go.Scatter(x=filtered\_data['When'], y=filtered\_data['Profit\_Net'], name='Net Profit'), secondary\_y=False)  
# Adding dynamic market data trace based on user selection  
fig.add\_trace(go.Scatter(x=filtered\_data['When'], y=filtered\_data[market\_indicator], name=market\_indicator, marker\_color='lightgrey'), secondary\_y=True)  
fig.update\_layout(title\_text=f"Net Profit and {market\_indicator} Over Time")  
fig.update\_xaxes(title\_text="Date")  
fig.update\_yaxes(title\_text="<b>Net Profit</b>", secondary\_y=False)  
fig.update\_yaxes(title\_text=f"<b>{market\_indicator}</b>", secondary\_y=True, showgrid=False)  
st.plotly\_chart(fig)

# 3. Visualization: Profit Efficiency per Contract  
filtered\_data['Profit Per Contract'] = filtered\_data['Profit\_Net'] / filtered\_data['Contracts']  
fig = px.bar(filtered\_data.groupby('Contracts')['Profit Per Contract'].mean().reset\_index(),  
 x='Contracts', y='Profit Per Contract', text='Profit Per Contract')  
fig.update\_traces(texttemplate='%{text:.2f}', textposition='outside')  
fig.update\_layout(uniformtext\_minsize=8, uniformtext\_mode='hide',  
 title\_text='Average Net Profit per Contract', xaxis\_title="Contracts", yaxis\_title="Average Profit Per Contract")  
st.plotly\_chart(fig)

Data Pipeline for GUI:

1. Establish connection to SQL Server and run pre-defined queries to select relevant columns from desired table in database
2. Pull data from database into a pandas data frame
3. Clean data frame by handling #VALUEs and #DIV/0, and omitted NaN values where possible
4. Import the pickle library
5. Open a file named 'my\_dataframe.pkl' in write-binary ('wb') mode as 'file':  
    a. Use the pickle library to dump (write) the Trade\_df dataframe into 'file'. This saves the pulled data frame for transferring into the GUI

Building the GUI:

1. Import necessary libraries (streamlit, pickle, plotly, pandas)

2. Load DataFrame 'df' from a pickle file named 'my\_dataframe.pkl'

## Streamlit App ##

3. Set up the Streamlit app with a title 'TTG Trade Insights'

4. Create sidebar for filter options:

a. Add date and time input widgets for start and end datetime selection based on 'When' column in 'df'

b. Add a select box for choosing a trading side from unique values in 'Side' column

c. Add sliders for 'TH' and 'Strength' ranges based on their min and max values

d. Combine date and time inputs to create datetime range filters

5. Filter 'df' based on the sidebar inputs

6. Display statistics of filtered data in the sidebar

7. Visualization 1: Cumulative Net Profit Over Time

a. Sort filtered data by 'When'

b. Calculate cumulative profit and create a line plot with Plotly Express

8. Visualization 2: Net Profit and Market Indicator Over Time

a. Create a subplot with primary and secondary y-axes using Plotly's make\_subplots

b. Add traces for 'Net Profit' and the selected market indicator

c. Customize layout and axis titles

9. Visualization 3: Profit Efficiency per Contract

a. Calculate profit per contract

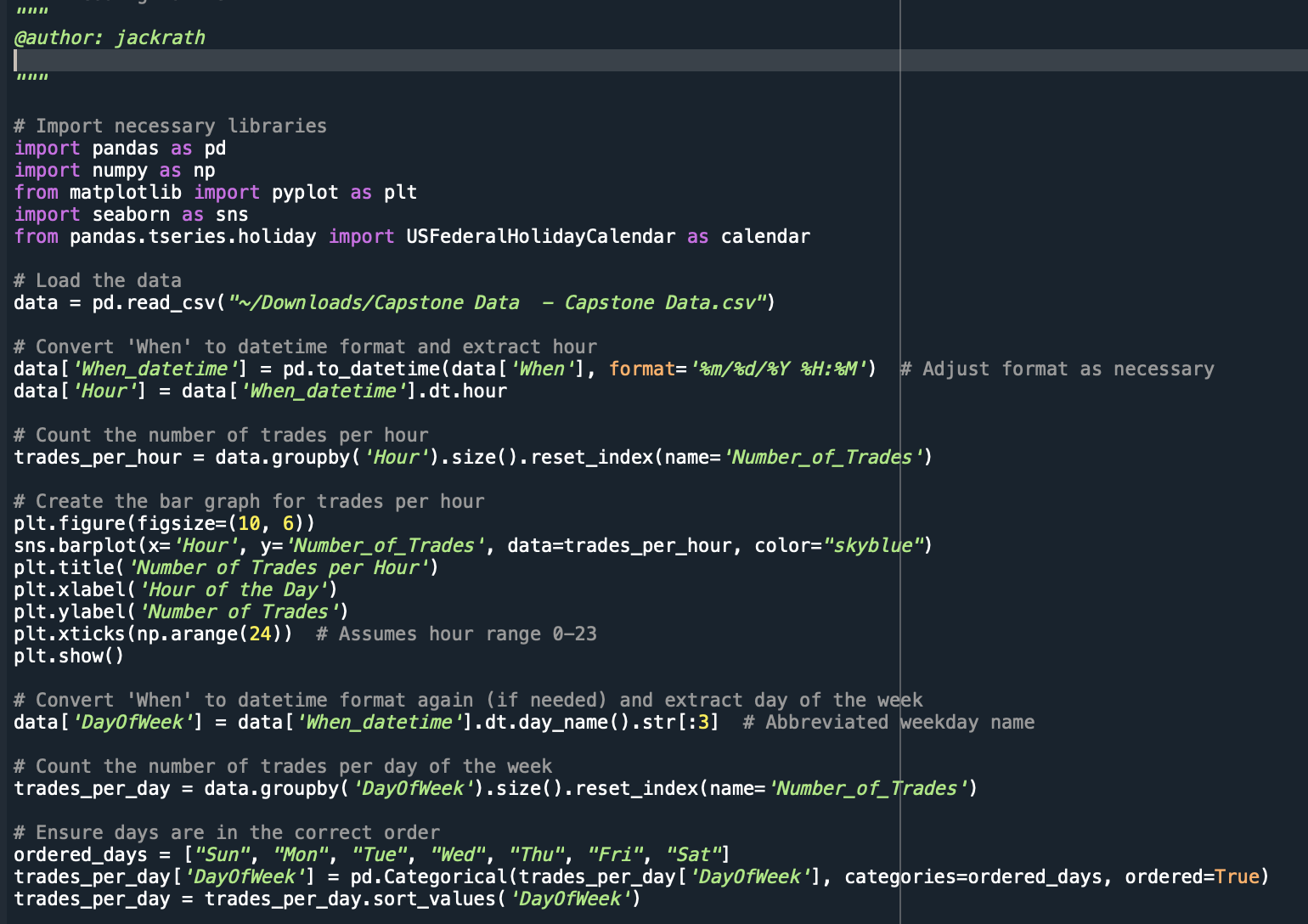
b. Group by 'Contracts', calculate average profit per contract, and create a bar plot with Plotly Express

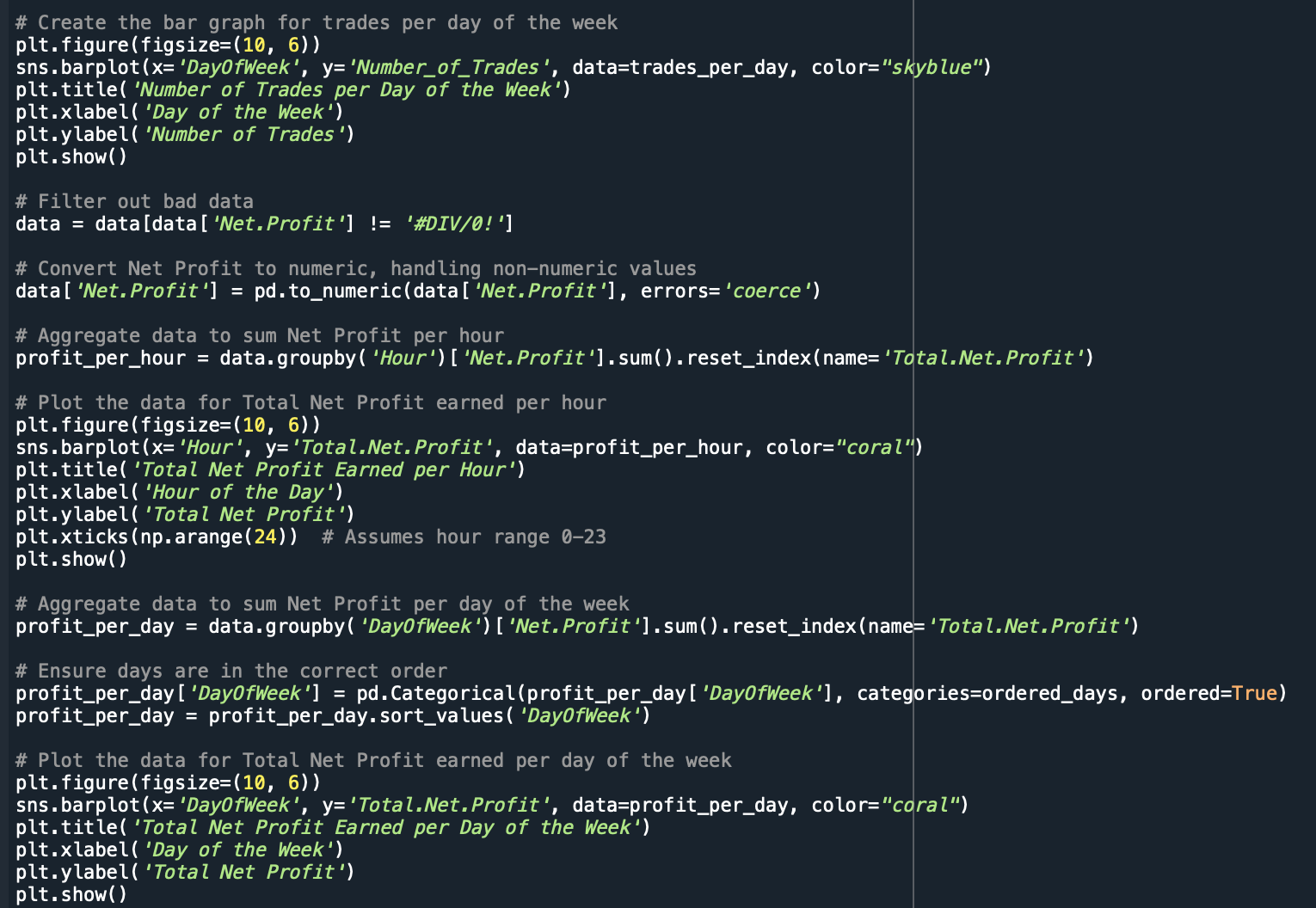
10. Visualization 4: Hourly Profit Trends

a. Extract hour from 'When', group by 'Hour', calculate average 'Profit\_Net', and create a line plot with Plotly Express

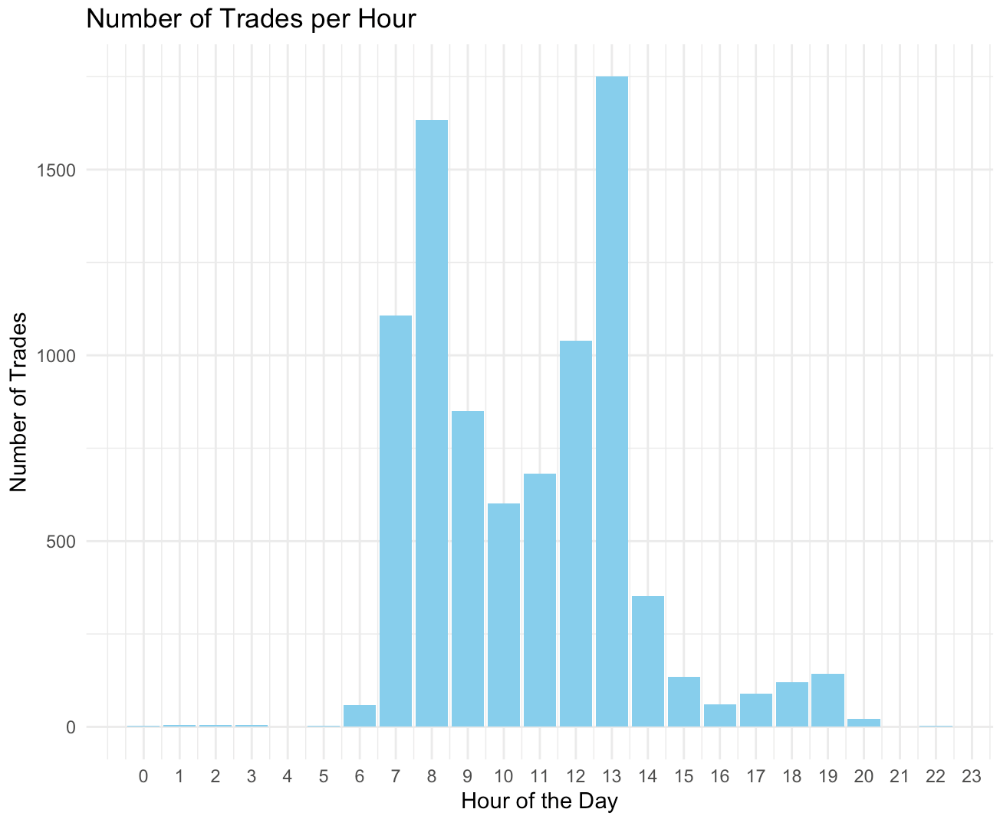
**Extension of GUI:**

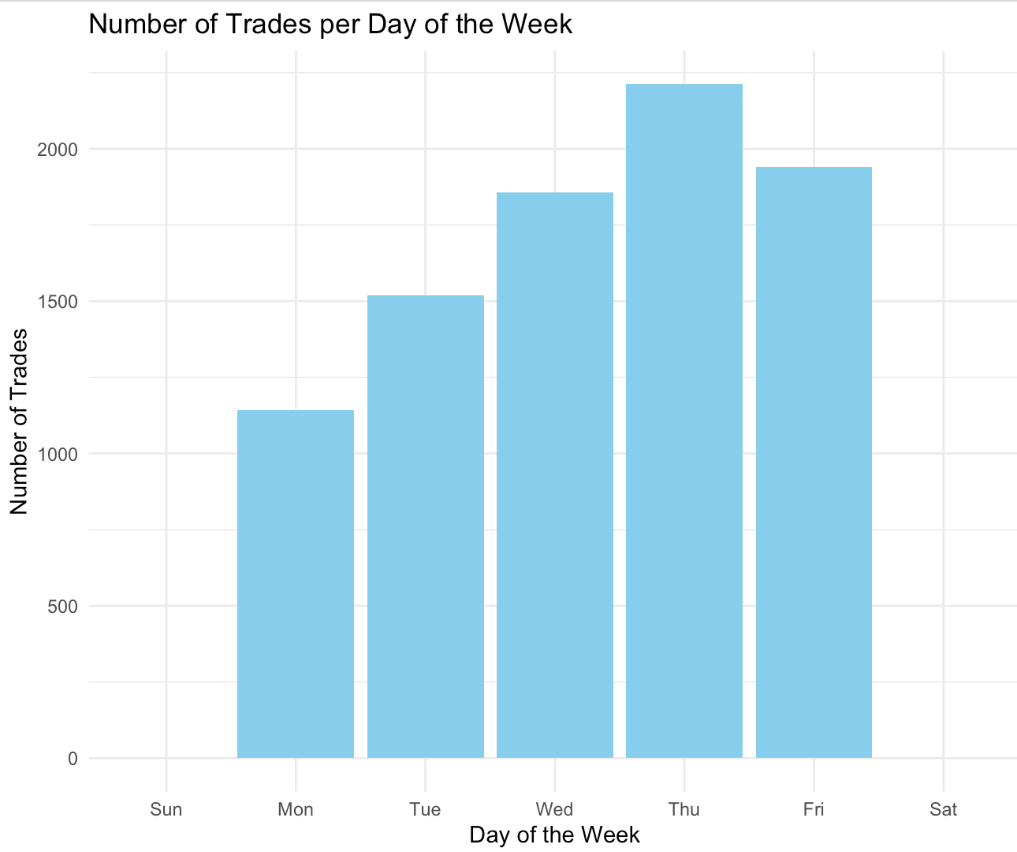
The graphs from the output of the following code will be added into the above GUI. Note that the pseudocode has been added as comments in the code itself.

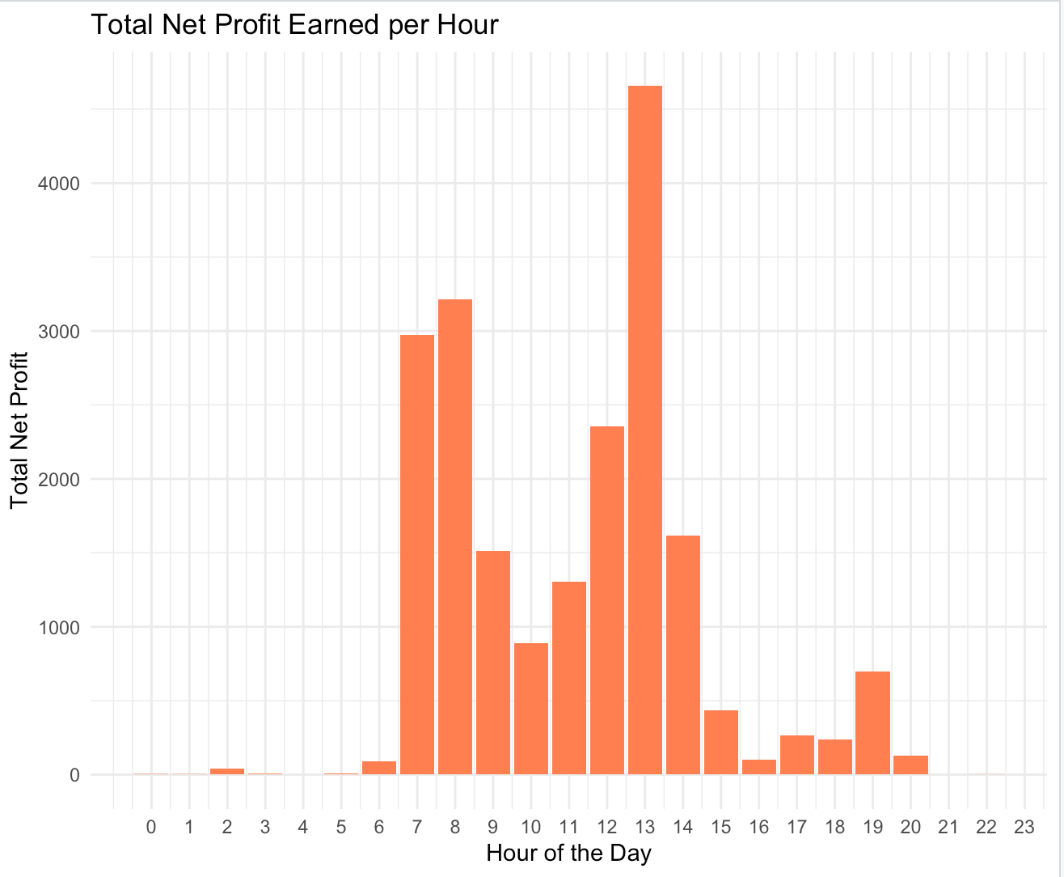
**Code (Jack):**  


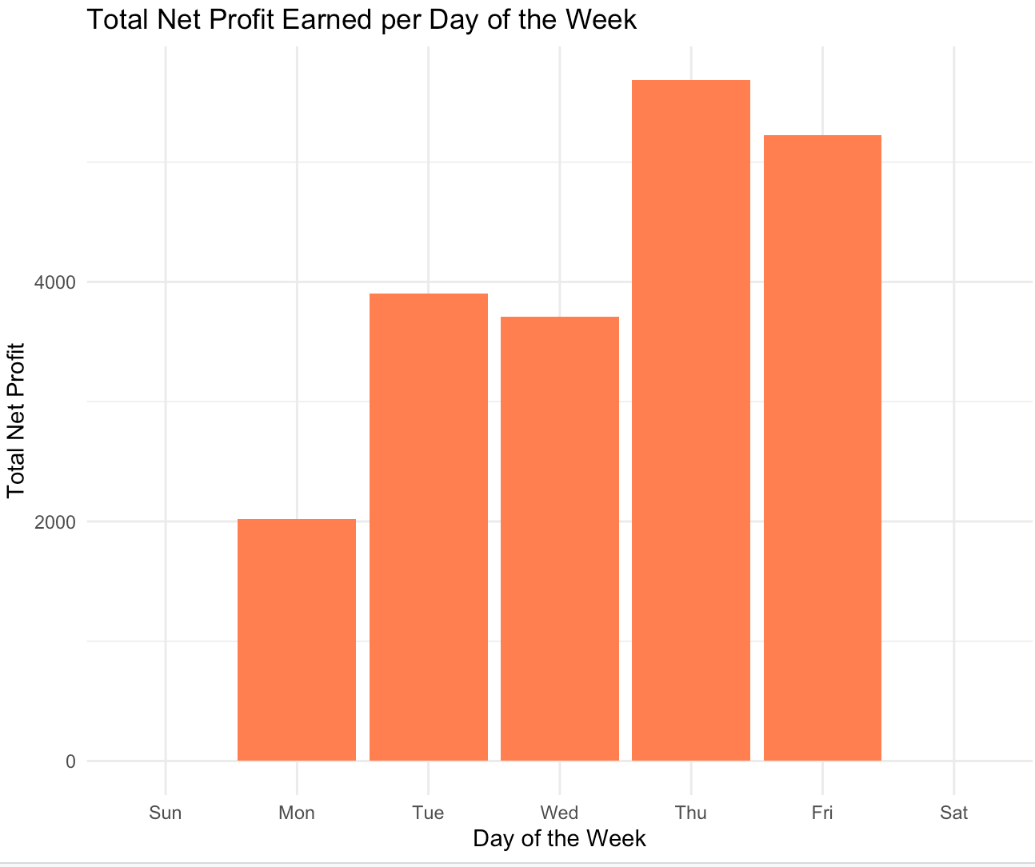


Output:









Diego: Join Partner provided Tables in SQL, run Lasso on net profit and other market variables including international market trading sessions:

A screen shot of a computer program

Description automatically generated

The code above simply does an inner merge (or “join” in SQL terminology) by matching the column “When” with the column “BarDateTime”.

Python Code:

A screen shot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer code

Description automatically generatedA screen shot of a computer

Description automatically generated

A blue background with yellow and blue text

Description automatically generated

Pseudocode:

1. Import necessary libraries (pyodbc, pandas, sklearn)

2. Define connection parameters for SQL database (server, database, username, password)

3. Create a connection string and establish a connection to the database

4. Initialize a cursor for the connection

5. Execute an SQL query to fetch column names from the "dbo.FXandTrades" table

- Store and print the column names

6. Close the cursor

7. Use pandas to execute an SQL query to select all data from "dbo.FXandTrades" table

- Load the results directly into a DataFrame named df1

8. Print the columns of the DataFrame

9. Convert the 'When' column in df1 to datetime format

10. Extract the time part from the 'When' column and assign it to a new column 'Time\_Only'

11. Define a function to determine if a time falls within a given market session

12. Convert 'Time\_Only' from string to datetime.time for comparison

13. Define market open and close times for various markets

14. Apply the market session function for each market to create new columns in df1 indicating if the time falls within the market session

15. Create the 'NY\_LOND' column based on specific conditions relating to 'NY' and 'LOND' columns

16. Iterate over each column in DataFrame to convert values to float if they are strings, else convert to NaN

17. Drop rows with any NaN values

18. Select specific columns from df1 to form the features (X2) and target (y2) for modeling

19. Split the dataset into training and test sets

20. Initialize and fit a StandardScaler to normalize features

21. Define a parameter grid for Lasso regularization strength

22. Initialize a Lasso model and a GridSearchCV to find the optimal regularization strength

23. Fit the GridSearchCV to the training data

24. Predict the target variable for the test set

25. Calculate and print the R-squared score for the test set predictions

A screenshot of a computer

Description automatically generatedUpdated Gantt Chart :