**Technical Appendix: Group A**

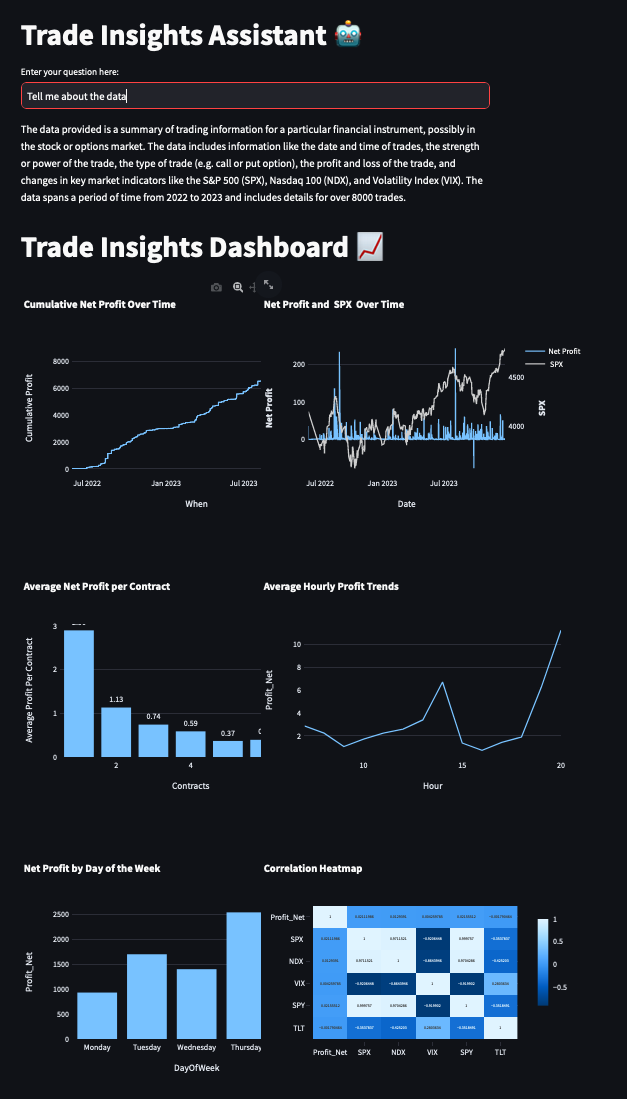
ECON 611/588

By: Jack R, Ammaar M, Colton L, Diego J, Prince K, and Andy H

Friday, April 5th, 2024

**Additions to GUI:**

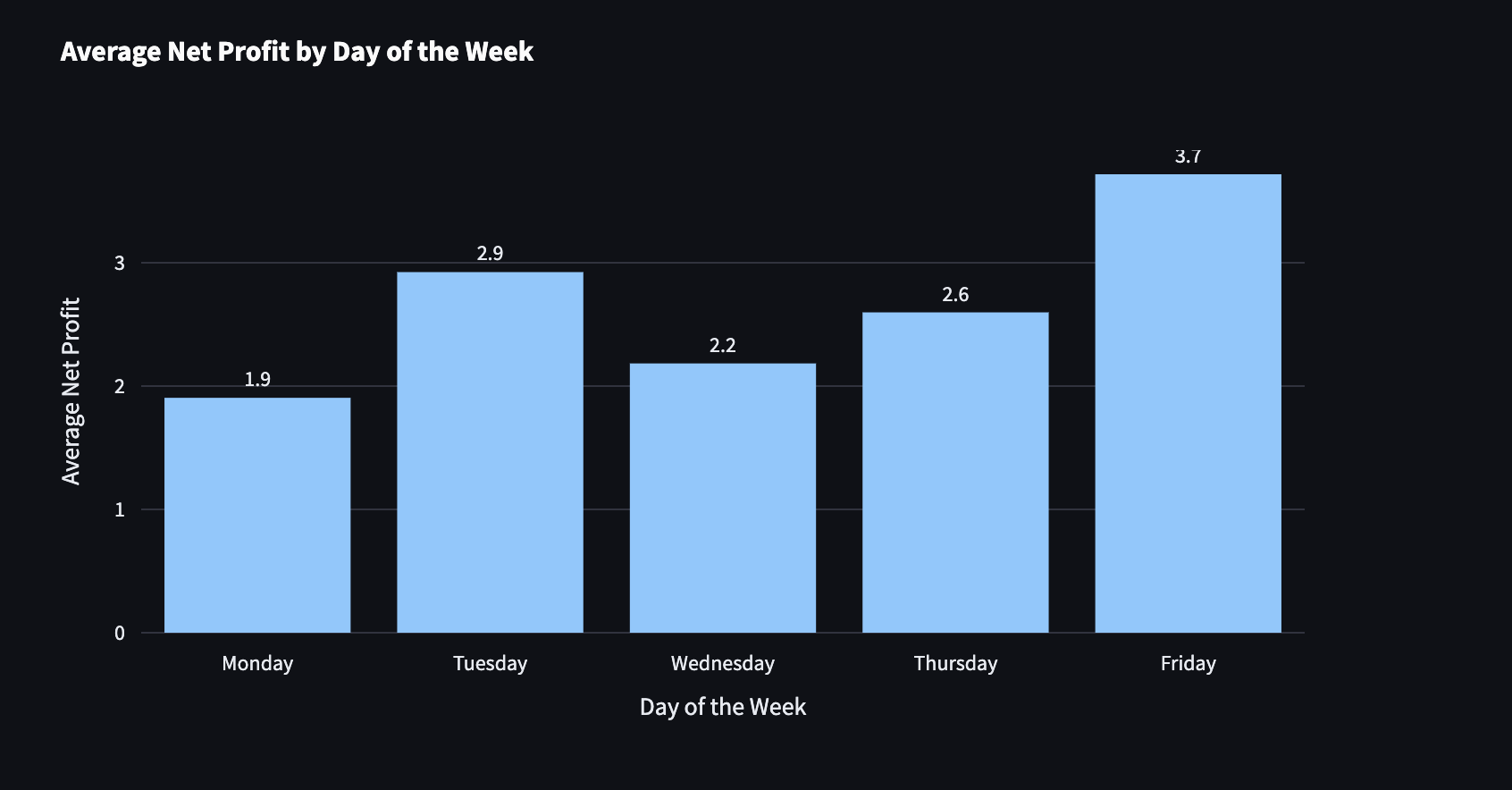
1. OpenAI powered chatbot with access to selected trade data.
2. Minor layout changes to optimize space usage.

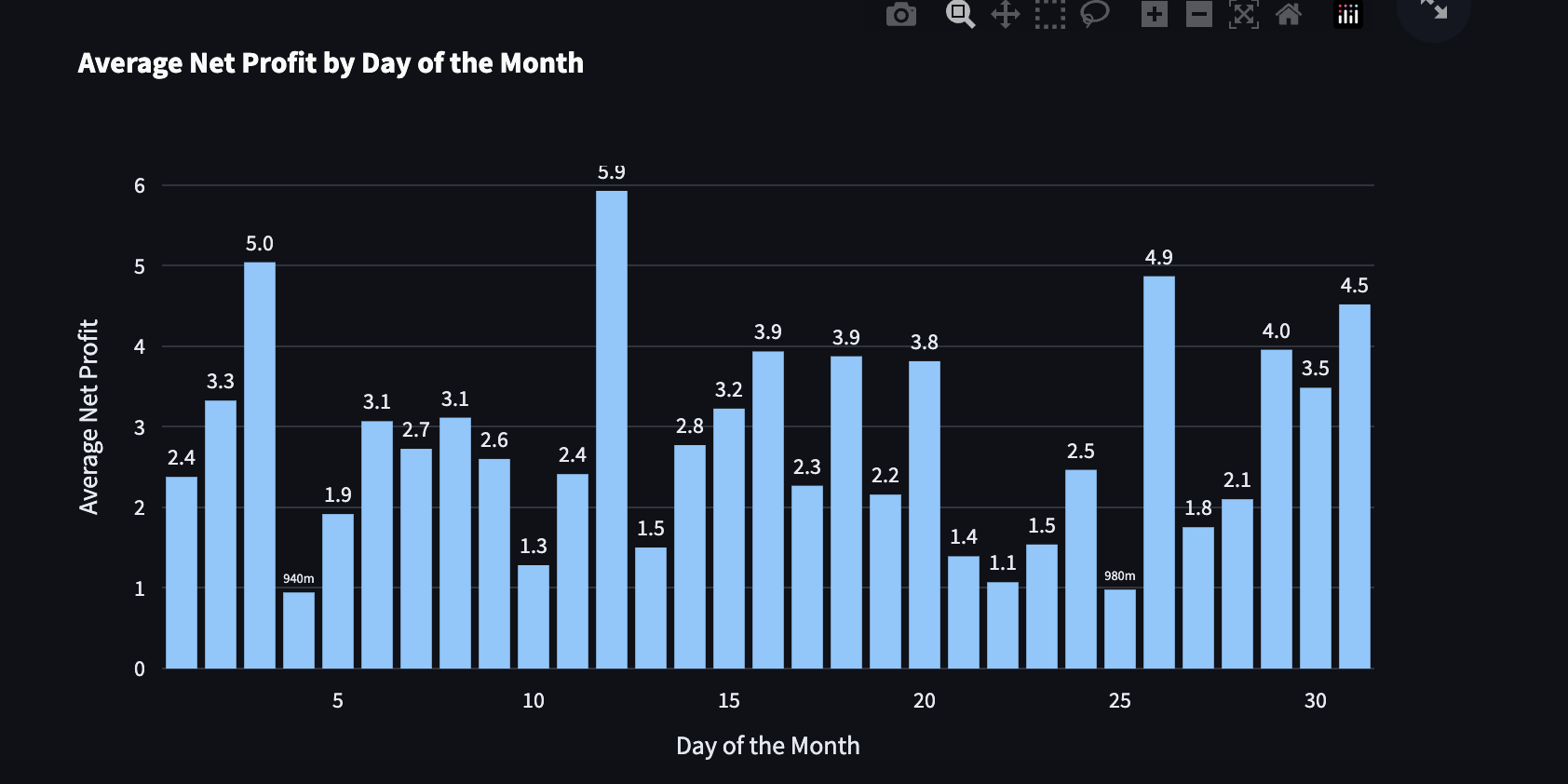


Pseudocode (AI Chatbot):

// Initialize OpenAI API key (ensure to comment this out before deploying to view the GUI without an OpenAI API key)  
Set openai.api\_key to an empty string  
  
// Display the title of the application as 'Trade Insights Assistant 🤖'  
Display title "Trade Insights Assistant 🤖"  
  
// Prompt the user to enter their query into a text input field  
user\_query = Display text input field with prompt "Enter your question here:"  
  
// Define a function to call the OpenAI API with a question and their trade dataframe  
Function ask\_openai(question, dataframe)  
Try  
// Create a prompt combining the user's question with a data summary  
prompt = "Question: " + question + "\n\nData Summary:\n" + dataframe + "\n\nAnswer:"  
  
// Call the OpenAI API with the specified parameters  
response = Call OpenAI completions API with:  
 model set to "gpt-3.5-turbo-instruct"  
 prompt set to the created prompt  
 max\_tokens set to 150  
 n set to 1  
  
// Extract and return the text from the API's response  
answer = Extract text from response and trim whitespace  
Return answer  
Catch any exception as e  
// Return the exception message if an error occurs  
Return the exception message as a string  
  
// Display the answer to the user's query if a query was submitted  
If user\_query is not empty  
Display spinner with text "Getting insights from Data..."  
answer = Call ask\_openai with user\_query and dataframe  
 Display answer

**Additional GUI Visualizations:**



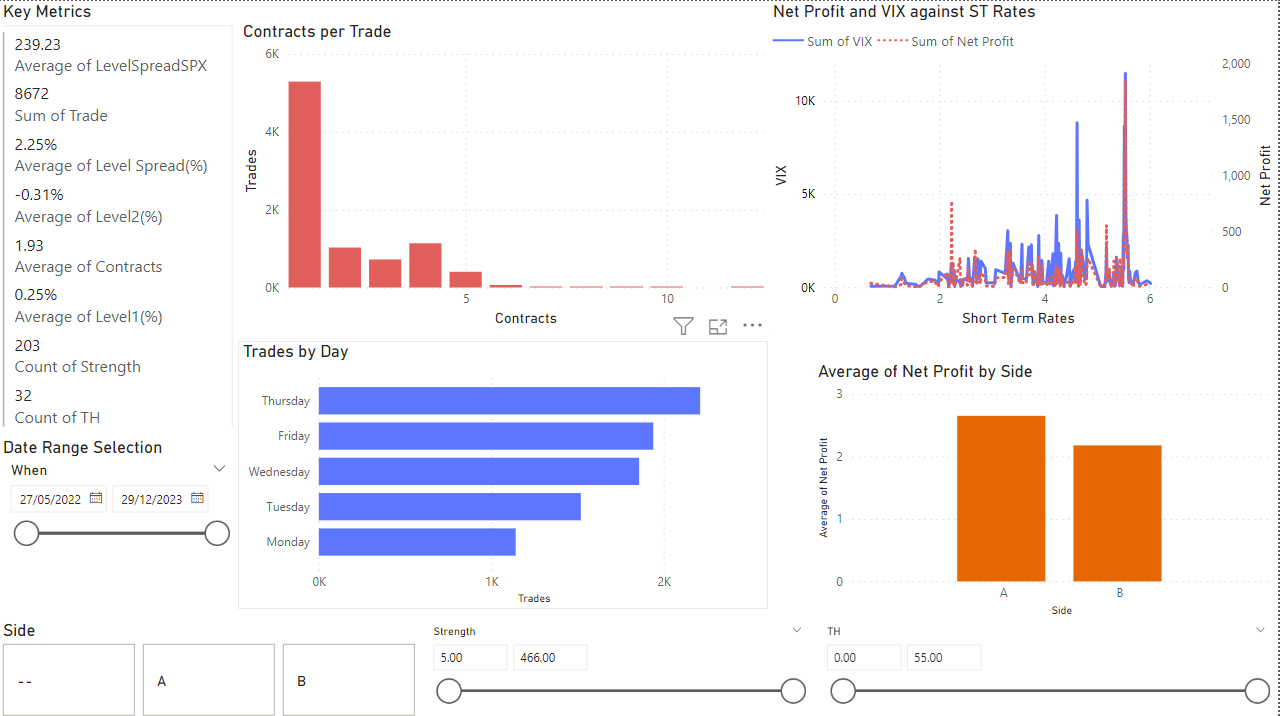


Code for Additional Visualizations:

# 6. Visualization: Average Profit by Day of the Week  
# Extract the day of the week from 'When' (0 = Monday, 6 = Sunday)  
filtered\_data['Day of the Week'] = filtered\_data['When'].dt.dayofweek  
# Calculate the average profit for each day of the week  
average\_profit\_weekday = filtered\_data.groupby('Day of the Week')['Profit\_Net'].mean().reset\_index()  
# Map the integers to weekday names for clearer visualization  
weekday\_mapping = {0: 'Monday', 1: 'Tuesday', 2: 'Wednesday', 3: 'Thursday', 4: 'Friday', 5: 'Saturday', 6: 'Sunday'}  
average\_profit\_weekday['Day of the Week'] = average\_profit\_weekday['Day of the Week'].map(weekday\_mapping)  
# Plot  
fig\_weekday = px.bar(average\_profit\_weekday, x='Day of the Week', y='Profit\_Net', text='Profit\_Net',  
 title='Average Net Profit by Day of the Week')  
fig\_weekday.update\_traces(texttemplate='%{text:.2s}', textposition='outside')  
fig\_weekday.update\_layout(xaxis\_title="Day of the Week", yaxis\_title="Average Net Profit")  
st.plotly\_chart(fig\_weekday)  
  
# 7. Visualization: Average Profit by Day of the Month  
filtered\_data['Day of the Month'] = filtered\_data['When'].dt.day  
# Calculate the average profit for each day of the month  
average\_profit\_day = filtered\_data.groupby('Day of the Month')['Profit\_Net'].mean().reset\_index()  
# Plot  
fig\_day = px.bar(average\_profit\_day, x='Day of the Month', y='Profit\_Net', text='Profit\_Net',  
 title='Average Net Profit by Day of the Month')  
fig\_day.update\_traces(texttemplate='%{text:.2s}', textposition='outside')  
fig\_day.update\_layout(xaxis\_title="Day of the Month", yaxis\_title="Average Net Profit")  
st.plotly\_chart(fig\_day)

**Power BI (PBI) updates:**

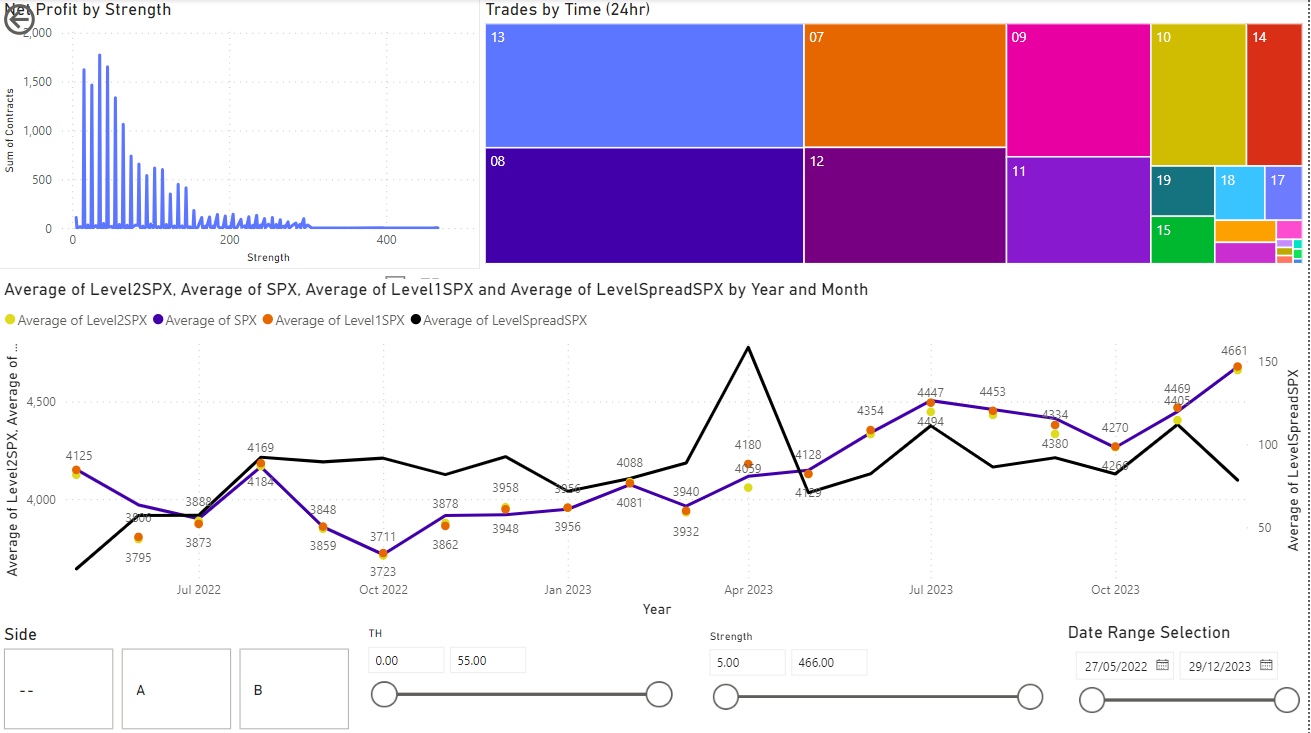
*\*Page 1*



**PBI Graphs/Charts:**

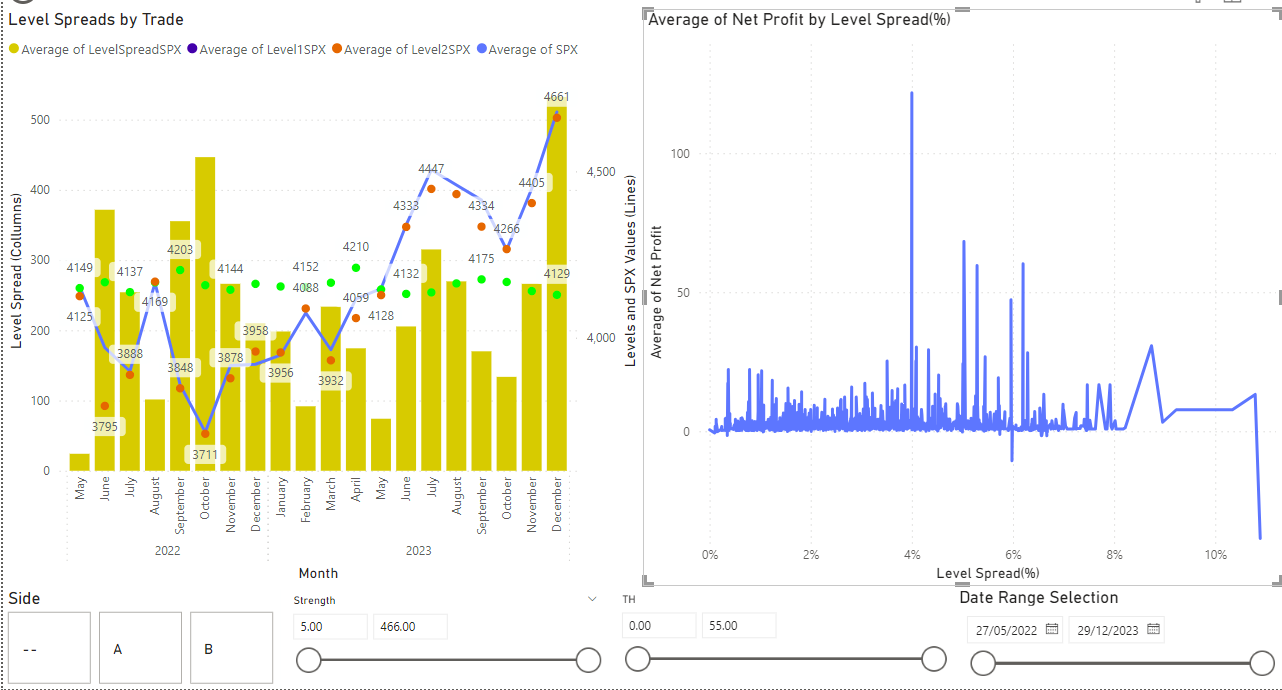
1. (Top Left) Shows the contracts per trade in a clustered column format. As the firm is still quite new, it shows that often, fewer contracts per trade occur. As the firm has expanded, it has been trading more contracts per trade which is slowly bringing the average up
2. (Top Right) Shows relationship between short term interest rates (US) and the response from the VIX. Net profit is also on there to see its relationship with the VIX as it had no direct relationship with the rates. As found, the VIX becomes more volatile as rates increase. Being a firm, whose strategy is to capitalize on market volatility, their net profit spikes and fluctuates with the VIX
3. (Bottom Left) Shows trades by day of the week. Thursday being when the trades are happening the most (therefore profit). Possible explanations for this obvious advantage this day brings are that earning reports and some key economic reports are released on Thursdays which has a short impact on volatility which could be when their system chooses to trade
4. (Bottom Right) Shows profit per side of trade. Side A trades are more profitable for the firm overall. However there seems to be increased risk for taking this side as occasionally the firm loses money while taking this side. Side B, while its profits are less, seldom loses money and when it does, the losses are a fraction of what they are when taking a poor performing side A trade.
5. (Left side) Key metrics table has some key metrics about the data the partner may find useful
6. (Bottom) Slicers allow for user to customize variables that the partner said were in their control (Most of these slicers repeat on subsequent pages)

*\*Page 2*



1. (Top Left) Strength of trades against the firm’s net profit. Every 5 units of strength yields profit spikes/troughs. The number of contracts traded at the trough values is likewise low, which impacts this graph. When filtering for sides, A has much more variation in the strength values of trade than side B.
2. (Top Right) Tree map of the times (24hr) at which trades are occurring. The most frequent times are 8AM and 1PM. The hours surrounding these times take up most of the trades being done. The early trading times (7,8,9) correlate heavily with the NYSE open time. In MST that exchange opens at 7:30. This could explain why trades at 6 are much less common. It is much the same for after 2PM MST where a sharp trading falloff occurs. More than 90% of trades happen during the times when the NYSE is open.
3. (Bottom) Tracking of the SPX along with the spread between levels (aiming to fix the SPX at a value for easier viewing and more insight)

*\*Page 3*



1. (Left) This combination chart attempts to do much of what #9 did. Showing the levels along with the SPX. The difference between the levels is in the form of columns. Levels, since they are a function of SPX, are expected to be correlated with it. This is not the case for level 1, which is currently mislabeled in bright green.
2. (Right) The line chart shows the profit against the level spread. While always fluctuating, the profit is nearly always positive at spreads less than 5% in difference. After that point, observations are limited yet does appear to become less predictable and potentially costly to the firm

**SQL / Code Updates:**

TTG expressed to us that if possible, they would like to be able to visualize what the relationship is between how much the market (denominated by an important index like the SPX, or a currency pair, like USD.CAD) moved in absolute terms (i.e. the absolute sum of how much the price fluctuated) and net profit.

In response, we created new tables in SQL that did performed a left join between their trades table and tables with 5 second price data for a given security. The result was multiple very large tables (about 1.9 million rows), with trades occurring for approximately 7,000 of those rows, all matched up by time. We then downloaded the new table(s) into python as dataframes, created a function to calculate the absolute sum of price movements between trades (where the running sum restarts after every trade is made),and then create a smaller dataframe where each row had a trade occurrence and also showed how much the price moved before the trade was made.

The pseudocode for the python operations above is as follows:

1. Initialize database connection parameters:

1.1. Set server address.

1.2. Set database name.

1.3. Set username.

1.4. Set password.

2. Establish connection to the SQL database using provided parameters.

3. Fetch column names from a specified table:

3.1. Execute SQL query to select all from a designated table.

3.2. Retrieve column names from the query result.

3.3. Close database cursor.

4. Define SQL queries to extract necessary data from database.

5. Load data into DataFrame using pandas for further analysis:

5.1. Execute defined SQL queries.

5.2. Close database connection.

6. Data preprocessing:

6.1. Convert specific columns to datetime or numeric formats as required.

6.2. Sort DataFrame by datetime column in ascending order.

6.3. Identify and count NaN values in selected columns.

7. Feature engineering:

7.1. Add a column to indicate presence of trades based on conditions.

7.2. Calculate absolute price differences and store in new column.

7.3. Initialize columns for movement metrics and calculate running totals.

8. Create a filtered DataFrame including only rows where trades occurred:

8.1. Filter original DataFrame based on 'Trades' column.

8.2. Optionally, reset index of the new DataFrame.

9. Prepare data for database insertion:

9.1. Establish parameters for database connection.

9.2. Define target table name for data insertion.

10. Write DataFrame back to the SQL database using SQLAlchemy:

10.1. Create database engine with connection parameters.

10.2. Execute data write operation to specified table, replacing if exists.

**Gantt Chart:**

A screenshot of a computer

Description automatically generated