Dr. Jan Wedigo Radermacher, Ph.D.
Chair of Finance
Goethe University Frankfurt

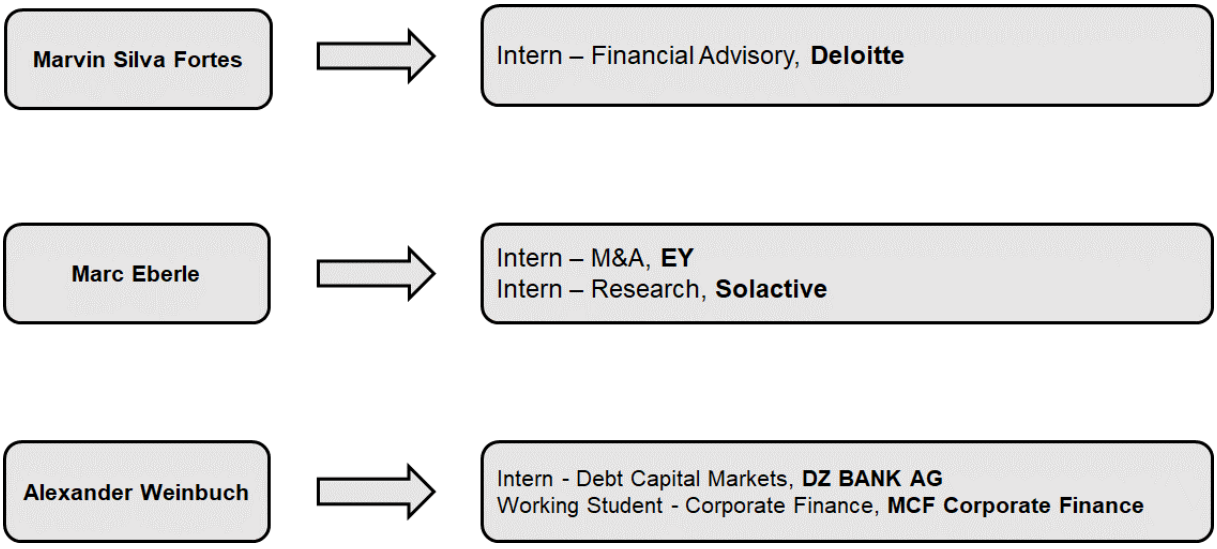
Financial Economics with Python

Submitted on 29 January 2023 by:

Marvin Silva Fortes (7294792)
Marc Eberle (7218978)
Alexander Weinbuch (7240559)

Table of Contents

1 Team



2 Development of Idea

As shown in the team introduction, Alex worked for DZ BANK in Debt Capital Markets. During that time, he was confronted with insider trading for the first time as he was not allowed to publish any confidential information and trade stocks only with permission. Following this experience, made him think about insider trading and wondered whether insiders are better investors than other investors. But with finishing the internship and therefore the end of mandatory reports to the Compliance team in the bank, insider investing was not a frequently appearing topic anymore.
A few days before Prof. Radermacher required the groups to select an idea for their dashboard, Alex was listening to one of his and Marc's favorite podcasts "Doppelgänger TechTalk" by Philipp Klöckner and Philipp Glöckler and Philipp Klöckner is one of the angel investors at Gorillas (now part of Getir). The two podcasters talked about the earning published by Just Eat Takeaway and Delivery Hero and Philipp Klöckner explained to the audience that due to his investment he is not allowed to talk about competitors and developments in the industry. With the topic of insider trading coming to Alex's mind, he asked Marvin and Marc to include the topic in our Python project.

3 Product Development

Based on our understanding, the most important part of developing a dashboard for the first time is a well-defined niche which needs further applications to become accessible to the public or at least the people who are really interest in the topic. Since all group members are interested in capital markets and invest actively in different markets, we agreed on the topic of insider trading and started to look at the current situation of access to information in this niche segment. This was one of the first issues we faced in our project as most websites only display selected information on insider trading or show outdated data that is published in

SEC Form 4 Filings. After several searches and general research, we found some websites that display the information we are interested in. Most of the websites display the information in an extremely messy and confusing format, which makes it almost impossible for “standard” retail investors to understand who is treading how many shares, how many shares does the person hold and most importantly is this information at all relevant.
Due to this situation, we decided to focus our project on making information on insider trading accessible to retail investors, comparing the performance of insider trades to the market, and help investors to make use of this information in a broader context, e.g. understand the market sentiment.
We started looking for further websites we could use to apply Web Scraping and gather relevant information. One main obstacle we faced and still face is that those websites with most sufficient and best-formatted data are only accessible through a subscription. At the same time, these circumstances motivate us to bring (more) transparency into a very important niche segment that provides crucial information as from our perspective (potential) investors have a right to have proper information on what insiders (e.g. C-level executives, x% owners, and founders) are doing with their holdings in the companies.
What all websites, we visited during our initial research and development phase, are missing:

1. Newsletter that informs the (potential) investor on a monthly basis about what insiders of the largest US companies (investor’s companies of interest) are trading.
2. Integrated overview of a firm’s stock price development, news on the firm
3. Comparison of the performance of insider trades with the market in the relevant period (S&P500)
4. Giving information on the most relevant question: Do insiders generate Alpha? Are insider better investors?

To implement these features and give information on these questions, we started to build the tool around our initial idea.
Within the process, we even developed a company/tool name with a slogan:

- Company/Tool name: InsidersInvest
- Slogan: Transparency for all

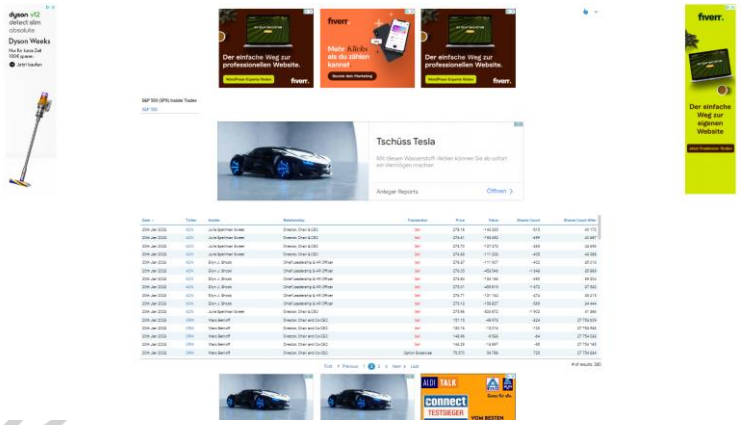
We even implemented another graphical element as we think that this best describes the purpose of our tool. The flower we used is growing from beneath the surface, which is where information on insider trades currently is located. Our tool helps information to spread, grow, and reach sunlight to help investors make for fundamental decisions. Furthermore, investors knowledge is also growing like a flower – first there is a seed, planted beneath the surface but with water (relevant information) the seed develops into a beautiful flower.
In the following sections, we will explain our different tasks, challenges, and results.

4 Application of Ideas and Features (incl. Problems)

In the following sections Alex, Marvin and Marc will describe their individual tasks, problems, learnings and present their results.

4.1 Alex

After developing and formulating the idea, we started to look for websites that provide us with reliable and recent data. As described above, one of the main problems was that most websites that provide promising content are paid services. On the other hand, most free websites are extremely messy and only provide outdated or parts of relevant data. We found a website that seemed to fit our needs and help us to gather relevant data, but we faced several obstacles as there are irregular pop-ups and advertisements on the website which prevented us from scraping data.



https://www.finscreener.org/insider/insider-trades/sp500?o=1002&pg=2

Furthermore, we wanted to use https://finviz.com/insidertrading.ashx but on the website only the 200 most recent trades are displayed, which is not enough data for our analysis. Therefore, we had to start again and spent time on finding the best possible website for our data. After checking dozens of websites, we found http://www.openinsider.com/ , which is the website we use to scrape the data on insider trading.

Following this, we started to think about special features that our tool should provide to differentiate from other tools/websites that only show data without “guiding” the (possible) investor through the information. In this process, we had the idea to implement a newsletter and “develop” the InsidersInvest Alpha as an indicator for insider’s trading performance. The measure is constructed based on the following idea: As our data is focused on companies that are part of S&P500 index, we use the performance of the index as our benchmark.

$$Price\ (Insider\ Purchase) - Price\ (most\ recent))/\ Price\ (most\ recent)$$
$$= Performance\ of\ Insider$$
$$(IndexValue\ (\ at\ Acquisition\ date) - IndexValue\ (most\ recent))/\ IndexValue\ (most\ recent)$$
$$= Performance\ of\ Benchmark$$

Comparing these two values leads us to InsidersInvest Alpha:

$$Performance\ of\ Insider - Performance\ of\ Index = Insider's\ Alpha$$

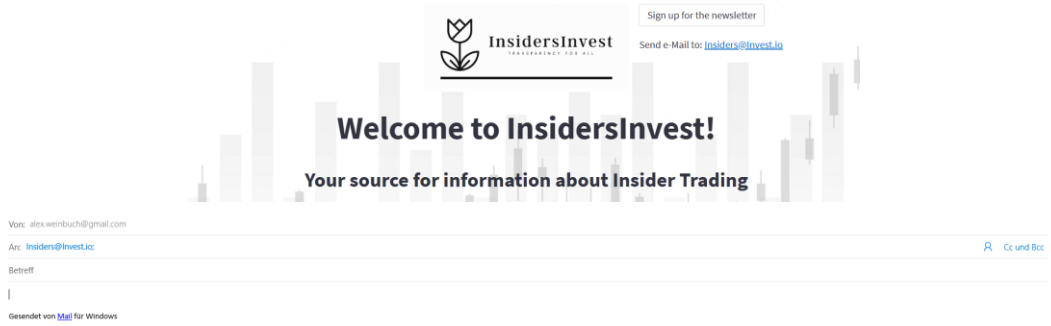
This performance measurement is calculated and displayed for all of the trades we provide data for when filtered through the filter option. Problems, (e.g. data format) arising during the implementation of the InsidersInvest Alpha are described by Marvin.

```python
# Section where we introduce alpha: ==============================
sp500 = pd.read_csv("sp500_index.csv")

df_split = sp500['Date;S&P500'].str.split(";", expand=True)
df_split.columns = ['Date', 'S&P500']
sp500 = pd.concat([sp500, df_split], axis=1)
sp500.drop(columns=['Date;S&P500'], inplace=True)
sp500['Date'] = pd.to_datetime(sp500['Date'], format='%d.%m.%Y').dt.strftime('%Y-%m-%d')

# convert 'Date' column in sp500 to datetime format
sp500['Date'] = pd.to_datetime(sp500['Date'])
sp500['Date'] = pd.to_datetime(sp500['Date'], format='%Y-%m-%d')


# change format of 'Date' column in merged_df to match the format in sp500
merged_df['Date'] = pd.to_datetime(merged_df['Date'])
#merged_df['Date'] = merged_df['Date'].dt.strftime('%Y-%m-%d')
merged_df['Date'] = pd.to_datetime(merged_df['Date'], format='%Y-%m-%d')

merged_df = pd.merge(merged_df, sp500, on='Date')

# ==End SNP500 value formatting and integrating
merged_df['S&P500'] = pd.to_numeric(merged_df['S&P500'], errors='coerce')

# Get the S&P500 value for the newest available date in the sp500 dataframe
latest_sp500_value = sp500.loc[sp500['Date'] == sp500['Date'].max(), 'S&P500'].values[0]
latest_sp500_value = pd.to_numeric(latest_sp500_value, errors='coerce')


# Create a new column in the merged_df dataframe that calculates the percentage change
merged_df['S&P500_PCT_Change'] = ( latest_sp500_value - merged_df['S&P500']) / merged_df['S&P500'] * 100
merged_df["Alpha"] = merged_df["Percent_Change"] - merged_df["S&P500_PCT_Change"]

print(merged_df)

# END OF ALPHA ===================================================

# Transform back before putting into Table:
merged_df['Price'] = merged_df['Price'].astype(str)
merged_df['Closing_Price'] = merged_df['Closing_Price'].astype(str)
merged_df['Price'] = "$"+merged_df['Price']
merged_df['Closing_Price'] = "$"+merged_df['Closing_Price']
merged_df = merged_df.rename(columns={'Closing_Price': 'Closing_Price_Today'})

merged_df['Percent_Change'] = merged_df['Percent_Change'].round(2)
merged_df['Percent_Change'] = merged_df['Percent_Change'].apply('{:.2f}%'.format)



# Take or Drop SnP500 at Trade Date
#merged_df = merged_df.drop("S&P500", axis=1)
merged_df['S&P500'] = merged_df['S&P500'].round(2)

merged_df['S&P500_PCT_Change'] = merged_df['S&P500_PCT_Change'].round(2)
merged_df['S&P500_PCT_Change'] = merged_df['S&P500_PCT_Change'].apply('{:.2f}%'.format)

merged_df['Alpha'] = merged_df['Alpha'].round(2)
merged_df['Alpha'] = merged_df['Alpha'].apply('{:.2f}%'.format)
merged_df.columns = merged_df.columns.str.replace('_', ' ')

st.header("Purchase Performance per Trade")
st.table(merged_df)
```

As our tool needs a structured and clearly laid out landing page/homepage, we used streamlit to create one.
To make our website look professional, we decided to implement a background which was a problem at first. Based on different suggestions out of stackoverflow/github, we had to use a CSS file to implement a background. After working with the CSS file and making first steps with HTML as a language, we found a work-around:

```python
st.set_page_config(page_title="InsidersInvest", page_icon="📈",
                   layout='wide', initial_sidebar_state='expanded')
import base64
def add_bg_from_local(image_file):
    with open(r"C:\Users\Alexander-NB\OneDrive\Desktop\Uni\6. Semester\FEWP\Project\App\{}".format(image_file), "rb") as image_file:
        encoded_string = base64.b64encode(image_file.read())
    st.markdown(
    f"""
    <style>
    .stApp {{
        background-image: url({encoded_string.decode()});
        background-size: cover
    }}
    </style>
    """,
    unsafe_allow_html=True
    )
add_bg_from_local('background.png')
```

After setting a background, we started to work on the structure of our website and thought about first details on our design. Due to that, we agreed on a name for our tool and a slogan. As described above, we created a logo that fits our purpose.
In addition, we created a button to sign-up for our newsletter. After clicking this button our business e-mail pops up. When a user clicks on the e-mail address, the default e-mail application with our mail address already set as a default value opens. Writing an e-mail to us registers users for our monthly newsletter.



When implementing the logo and the newsletter button, we struggled with centering the image and placing the button next to it. We could use a CSS file to solve the problem. But we worked with columns instead:

```
def main():

    # Add a logo + Button and "locate" it correctly
    col1, col2, col3 = st.columns(3)

    with col1:
        st.write(' ')

    with col2:
        st.image("Logo_new.png")

    with col3:
    # Add button
        Newsletter_button = st.button("Sign up for the newsletter")
        if Newsletter_button:
          st.write("Send e-Mail to: Insiders@Invest.io")
        else:
            st.write("")
```

Our homepage gives a quick overview on what our tool is designed for and why an investor should use it.

After “introducing” our product, our homepage gives an overview on our features and what to expect from using our services. Furthermore, we give an insight on the importance of insider trading by implementing a button that opens an article by New York Post about Nancy Pelosi outperforming the S&P500 since 2019. We chose this article as it shows that insiders’ portfolio (not only corporate insiders) deserve to be looked at. In addition, we show a graph that compares the performance of a portfolio based on Congress buys and S&P500 in 2021/22. This portfolio is created by “Quiver Quantitative” whose website can be reached by clicking on the button next to the graph.

Features offered:

- Newsletter with information on insider activities of the largest US companies ✅
- Monitor insider trading activity for all S&P500 companies ✅
- Search and filter through insider trading data ✅
- Analyze stock data and performance ✅
- Compare insider returns to the S&P ✅
- Selected case studies for more fundamental investment decisions ✅

Why is it important to know what insiders are doing?

Example for insider performance

Congress members outperform the market

Checkout Quiver Quantitative

A user might think “Why are they showing data/providing links to information that refers to politicians as insiders?” – we do this because there is substantially more information on insider trading by politicians than on corporate insiders. Our purpose is to bring more transparency into the world of insider trading by managers, owners and other investors connected to the largest companies of the United States. That is why we included the following two sentences:

As you can see, there is more transparency on insider trading by politicians.

InsidersInvest allows you to become an expert on corporate insider trading!

Try it now

The ”Try it now”-button is linked to our tab “Most recent trades”.
Furthermore, we included a sidebar that allows the user to choose from all of the S&P500 companies. The selections made become effective when using the other tabs.

4.2 Marvin

I was trusted by my team members with a challenging but fun task. I had to work with insider trading data we found, scrape it and edit unnecessary parts out, while working the data to provide even more insights.

We started together and time was not plentiful, because soon a team member had to reallocate a lot of his resources into writing his bachelor thesis, while I had to worry about my seminar papers. Nevertheless, we gave our best and started ASAP with the development of our idea.
Since this would be a critical part of our project, us three started together to look for appropriate data while prioritizing recency and information content. As Alex already mentioned in this part, this was one of the first struggles that intervened predominantly with the progress of my task of the “Recent Trades” tab we wanted to include. Problems were:
1. Initially entering the page and being able to scrape after closing the pop-up that tries to incite you to complete a membership. The problem here was that it wasn’t completed with one operation, as taught in our lecture, but it would seemingly unsystematically pop up again every few seconds.
→ Solution here was to implement a line of code that would wait for the popup to appear only to then close it.
2. Second and biggest problem was that the site, even when scraped properly after finding the right HTML body and tbody, only gave us a maximum of 100 trades to monitor at once before having to press on a second page and starting the process anew. The second page was not even accessible through another link, but you’d have to program a mouse click on page 2, 3, 4 etc. So, Data was a Problem (was even not that up to date as other options).
→ Solution: Alex, Marc and I searched the internet for potential alternatives and Alex then found the site “openinsider.com”. A site providing less data at first, but the most important: while offering the largest amount of data.
We switched to openinsider, which provided the ability to navigate with URLs. For example, you could adjust the accessible data on one site with just a link that set the amount to one thousand different trades. As I later worked on integration of the scrape into our code, I found out that the scrape was ridiculously efficient and easily targetable only by using pandas. We left out selenium which was more clunky and inefficient for openinsider.

Another site we tried was finviz. Here the site layout and scraping proved to be difficult again. When checking for the tbody/tr/td tags the td tags were not listed from 1 to 10 as you'd expect for each column but rather the first few needed extra attention by adding an /a at the end. It took some research to realize that.
Below you will find examples of code used for the scraping of the data. The first will be the one used for finviz, the second one for openinsider.

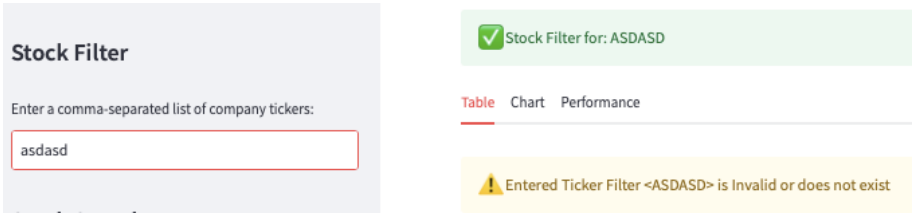
Finviz (left); OpenInsider (Right)

```python
driver = webdriver.Chrome()
driver.get("https://finviz.com/insidertrading.ashx")

#Hide chrome
driver.set_window_position(-12000,0)

time.sleep(10)

# ==== TAKING CARE OF COOKIE NOTIFICATION =====

WebDriverWait(driver, 7).until(EC.element_to_be_clickable((By.XPATH,
                "//div[2]/div/button[3]"))).click()


# ========================================

#Ausführende Personen
owners = driver.find_elements(By.XPATH, "//table[2]/tbody/tr/td[2]/a")

tickers = driver.find_elements(By.XPATH, "//table[2]/tbody/tr/td[1]/a")

relationships = driver.find_elements(By.XPATH, "//table[2]/tbody/tr[position()>1]/td[3]")

#Dates of trade:
dates = driver.find_elements(By.XPATH, "//table[2]/tbody/tr[position()>1]/td[4]")

#Buy, Sell or Option:
transactions = driver.find_elements(By.XPATH, "//table[2]/tbody/tr[position()>1]/td[5]")

# ================ TRANSACTON VALUE ==================
#At Cost per Share:
costs = driver.find_elements(By.XPATH, "//table[2]/tbody/tr[position()>1]/td[6]")

#Number of Shares Transaction:
shares = driver.find_elements(By.XPATH, "//table[2]/tbody/tr[position()>1]/td[7]")

#Value of Transaction
valtransactions = driver.find_elements(By.XPATH, "//table[2]/tbody/tr[position()>1]/td[8]")
# ================ END TRANSACTION VALUE ==============

#Number of shares total:
numshares = driver.find_elements(By.XPATH, "//table[2]/tbody/tr[position()>1]/td[9]")

#When was SEC Form filed?
SECs = driver.find_elements(By.XPATH, "//table[2]/tbody/tr[position()>1]/td[10]")


#Neue Liste erstellen um Daten zu speichern, später dann in Excel import
insiders=[]

for i in range(len(shares)):
    temporary_dict={"Owner" : owners[i].text,
                    "Ticker" : tickers[i].text,
                    "Relationship" : relationships[i].text,
                    "Date" : dates[i].text,
                    "Transaction Type" : transactions[i].text,
                    "Cost per Share" : costs[i].text,
                    "Number of Shares" : shares[i].text,
                    "Value of Transaction in $" : valtransactions[i].text,
                    "Total Number of Shares Outstanding" : numshares[i].text,
                    "SEC filing date" : SECs[i].text}
    insiders.append(temporary_dict)

df = pd.DataFrame(insiders)
```

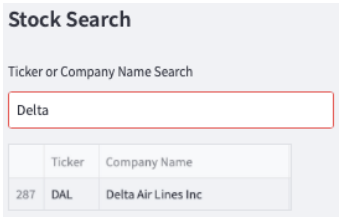
```python
# Scraping & Implementation:
insider = pd.DataFrame()

for company in companies:

    url = f'http://openinsider.com/screener?s={company}&o=&pl=&ph=&ll=&lh=&fd=1461&fdr=&td=0&tdr=&fd

    insider1 = pd.read_html(url)
    try:
        insider1 = insider1[-3]
        insider = pd.concat([insider, insider1])
        insider = insider.drop(columns=["X", "1d", "1w", "1m", "6m"])

        insider = insider.reset_index(drop=True)


        with tab1:
            st.table(insider)
```

(Takeaway: around 25 lines of code for finviz; and 7 for openinsider.)

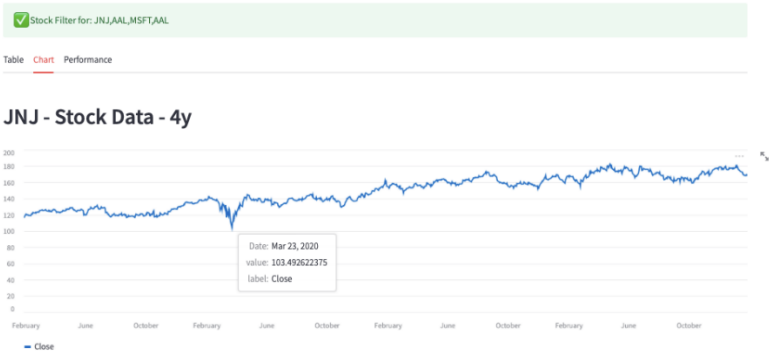
The tool "Most Recent":
We were able to present the tool user with data on the SEC Form 4 Filing Date (YYYY-MM-DD), Trade Date (YYYY-MM-DD), Ticker, Company Name, Insider Name, Title of the Owner, Trade Type, Price, Quantity Traded, Owned Amount of Stocks after the Trade, and Percentage Change in Owned Stock as well as Value of the Transaction.
This data would need a filter function, to serve the user who is interested in just a certain company and not all of them. Applying the filter reveals the last 1000 trades that are classified as insider trades, meaning top 10% holders, management, and company owners, including option exercises, noted as "+ OE" after the transaction type.
The next challenge that we encountered was debugging error codes in the code after implementing the filter function through `st.text_input`. We had previously defined a variable to save the user input and scrape the most recent 1000 trades from the internet. This meant that the Python code in the console, which had previously allowed us to debug more easily in Spyder, was now always giving out an error since no input was given in the IDE (Integrated Development Environment - in our case, Spyder). We had to debug in the app itself, which gave out error messages as well, but it was a challenge to navigate between the IDE and the Streamlit App.
We overcame this challenge by learning and implementing "try; except" functions into our code. This allowed us to ignore certain error messages, assuming values as given, and thus ignoring KeyErrors and NameErrors that would pop up if we tried to debug our code in the console. Additionally, it allowed us to set customized warning messages to inform the user in case of a missed input. For example:



In cases where debugging was needed due to other mistakes on our end, debugging in the Streamlit interface posed one of the more annoying and tedious tasks. Sometimes an error message was only displayed in Streamlit, while other times it was only displayed in Spyder, which accounted for some hours in debugging and compatibility optimization. During the implementation, we experienced some of the limitations of Streamlit. For example, when trying to hide the stock name list in the sidebar menu together with the search text input, using `with st.expander():` resulted in the problem that the search bar itself was left out in the open, while the table was left out in the open.
We overcame this challenge by not trying to hide the search function at all (haha). Now it looks like this and is to be seen as separate from the filter function you see above. An indicator for that is the new `st.header` that has been implemented: "Stock Search".

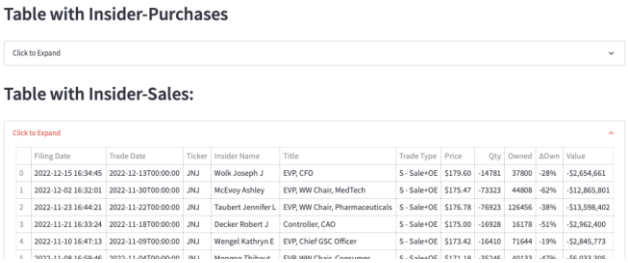


Now, coming to the central part of the tool, we took the data freshly scraped and filtered by the st.text_input and used pandas to separate the sales from the purchases. To clarify: this is taking the last 1000 transactions as a total and not the last 1000 purchases and the last 1000 sales. We found it to be sufficient and always reaching back until about January 2019, which is why we set the stock price visualizations to four years as well. We implemented two different visualizations. One with Streamlit, which gives us a nicely modern and clean looking graph with added interactivity. You can point your mouse at a certain point in the graph and it will show you the date and price taken from Yahoo Finance.



In this example, one can see a relatively stable, up trending course of Johnson&Johnson over the last four years with a visible discrepancy shown at the point in time where the Corona crisis hit (around March 23rd, 2020) with a closing price of $103.49.
After the initial visualization, we are trying to grant the user one of our key elements of the tool: the two separated tables reflecting together the last 1,000 transactions for the company, in this example JNJ. Since these can take up a lot of space, but we wanted to put visible distance between the two stock graphs, we decided to implement the `st.expand` functions to facilitate scrolling through the page without forcing anyone to scroll through the whole table, as these can end up quite lengthy if a company has a lot of insiders and insider trades. Also,

the focus on this page should be put on the charts, which is why this st.tab is also called “Chart”.



	Filing Date	Trade Date	Ticker	Insider Name	Title	Trade Type	Price	Qty	Owned	ΔOwn	Value
0	2022-12-15 16:34:45	2022-12-13T00:00:00	JNJ	Wolk Joseph J	EVP, CFO	S - Sale+OE	$179.60	-14781	37800	-28%	-$2,654,661
1	2022-12-02 16:32:01	2022-11-30T00:00:00	JNJ	McEvoy Ashley	EVP, WW Chair, MedTech	S - Sale+OE	$175.47	-73323	44808	-62%	-$12,865,801
2	2022-11-23 16:44:21	2022-11-22T00:00:00	JNJ	Taubert Jennifer L	EVP, WW Chair, Pharmaceuticals	S - Sale+OE	$176.78	-76923	126456	-38%	-$13,598,402
3	2022-11-21 16:33:24	2022-11-18T00:00:00	JNJ	Decker Robert J	Controller, CAO	S - Sale+OE	$175.00	-16928	16178	-51%	-$2,962,400
4	2022-11-10 16:47:13	2022-11-09T00:00:00	JNJ	Wengel Kathryn E	EVP, Chief GSC Officer	S - Sale+OE	$173.42	-16410	71644	-19%	-$2,845,773

The visualization of these two tables was difficult since the stock data received from yfinance was in a nice format to work with (date-time) and the format of the data had to be found out. This is where I personally started to learn about reformatting and the many ways of reformatting the data in a temporary dataframe to work with the graph. However, as soon as the x-axis (dates) was re-formatted, we managed to read out the data nicely. We first tried to work with Streamlit since it looked the most pleasant. This didn’t seem to work that well and turned out to be tricky, so we decided to work with Matplotlib, with the added downside that it had a rather old look to it. After speaking to Dr. Jan Radermacher about it, he suggested doing some research on the function of styles and stylesheets in Matplotlib. After comparison, we decided to go for the “seaborn-darkgrid” style since it would deliver added visual orientation purposes with the grid, while also not being too hard to “ignore” to see the vertical lines that were done delicately with dotted lines, so if a stock gained a little more sales it wouldn’t completely hide the graph behind a red wall. As a sidenote, this is what we would usually observe. There would always be more sales than purchases. We believe that this has to do with the fact that stock purchase programs are a part of a lot of insiders’ salaries.

Now, for the most important part of my journey with this project, we decided to integrate some sort of performance measure per trade that was done. This had to be done in a separate table, but using the data presented as purchase data from the last 1000 transactions dataset. We just used “Date”, “Ticker”, “Insider Name”, “Title”, “Trade Type”, “Quantity” and at first “Price” from the old “purchase” dataframe. Because this is where the next challenge appeared. We started off with using the old dataframe completely and just adding columns that would calculate the return on the hypothesis that the stock was held until today. For that we introduced a new column we would call “Closing Price Today”; which was taken from Yahoo Finance. Comparing this with the Purchase Price meant that we could find out a hypothetical return on the investment. That was one key measure. No mistakes or problems were realized other than the usual formatting problems, especially with the Yahoo Finance dates and our dates gained from the scraped dataframe.

After extensive collaboration with Alex, we decided that it would make sense to integrate another key metric to display. At this point, it wasn't realized that this would end up being one of the more time-intensive tasks, taking plenty of hours of debugging in the Streamlit UI and re-trying, changing code order, etc. The task is relatively simple to outline:

1. Take the return metric just calculated; Save it.
2. Take an existing database of SNP500 containing Dates and corresponding points.
3. Take the Dates of the Trade, calculate return of the SNP500 in the timeframe that has also been used for the return metrics on the stock.
4. Hereby reflect alpha of the insider trades (or excess returns) and using the s&p500 as market performance.
5. Display as additional metric in the dataframe.

We ran into several challenges. Since the purchase dataframe was displayed elsewhere, we had to take data from there and temporarily format the percentage in a way that it could be used in further calculation (removing the "%" sign and converting with `pd.to_numeric`). The S&P 500 data was in European date metrics, so we had to reformat the data to compare the dates of the insider trades. As everything else went smoothly, we implemented the feature and after trying it out and cleaning up the look of the table (adding percents etc.), it was realized that the data we scraped did not take stock splits into account, rendering the data useless. To fix this, we replaced the "Price" column with other prices we gathered from `yfinance`. The function we used initially didn't work, so after doing some research online, we found that `yf.download` helped tremendously. The only problem we ran into after that was that the data on the S&P 500 points was somehow not roundable by the `round` function, even though it passed without an error. This was not easily fixable and since we deemed it too unimportant to allocate more resources while we were approaching exams, we let it in with two extra 0's included.

Stock Data:

The Stock Data tab was introduced to provide a more in-depth view of the stocks themselves. This allowed for more room to focus on information that is not easily reflected in tables or graphs, as well as a larger space for information that is not solely focused on insider trading. We encountered several issues here. Initially, we included a tick-box system that would allow for more data to be displayed upon click. The goal was to display a large amount of the data that Yahoo Finance delivers. Unfortunately, this worked extremely inconsistently and the only features we found working reliably were "Institutional Investors" and the stock actions containing splits and dividends.

Another feature we wanted to display permanently was "Recent Headlines associated with <ticker>". We found it to be very interesting and relevant to our topic to include a news ticker that outputs general news about the company but could possibly display information on insider trading. There were several display options, but we decided to keep the most important ones: the title of the article, the publisher, and a link to the article.

Unfortunately, there is still a problem without an easy workaround. Even after a lot of trying, we settled to display the links that are given to you in the dataframe as non-clickable. This lies in Streamlit display options. If you try to display a link in an `st.table` or an `st.dataframe`, this link will be automatically rendered unclickable and there is no option to prevent this from happening. You could display it as `st.markup`, but we decided that it looks more clean this way and to just copy out the links and paste them in another browser window.

Recent Headlines associated with AAPL

	Title	Publisher	Link
0	Target, Amazon and 4 More Retailers That Will Reward You for Turning in Your Old Stuff	GOBankingRates	https://finance.yahoo.com/news/target-amazon-4-more-retailers-130111844.html
1	Fed meeting, jobs data, Apple earnings: What to know this week	Yahoo Finance	https://finance.yahoo.com/news/stock-market-week-ahead-big-tech-earnings-federal-reserve-fomc-interest-rates-125335273.html
2	Why This May Be A 'Life Changing' Market Rally; Apple, Fed Meeting Loom As Tesla Run Hits 75%	Investor's Business Daily	https://finance.yahoo.com/m/5a7c7c40-d411-32ea-8ac5-58c115fece40/why-this-may-be-a-%27life.html
3	$50 AirPods Pro? Nope. Here's How to Spot Fake Apple Earbuds.	The Wall Street Journal	https://finance.yahoo.com/m/275eda5b-528f-3810-bdff-173a010a96fc/%2450-airpods-pro%3F-nope.-here%E2%80%99s.html
4	Apple (NASDAQ:AAPL) stock performs better than its underlying earnings growth over last five years	Simply Wall St.	https://finance.yahoo.com/news/apple-nasdaq-aapl-stock-performs-110020106.html
5	5 Top Stocks for February	Motley Fool	https://finance.yahoo.com/m/aa9bc25f-1c22-3ac4-a050-d0718b9f7100/5-top-stocks-for-february.html
6	China's 2022 smartphone shipments the lowest in 10 years - research firm	Reuters	https://finance.yahoo.com/news/chinas-2022-smartphone-shipments-lowest-060925557.html
7	How to Build Great Wealth With the Power of Compounding	TheStreet.com	https://finance.yahoo.com/m/f653bfc5-9ceb-3624-9022-c816f08ac938/how-to-build-great-wealth.html

Stock actions for AAPL

	Dividends	Stock Splits
1987-05-11T00:00:00-04:00	0.0005	0.0000
1987-06-16T00:00:00-04:00	0.0000	2.0000

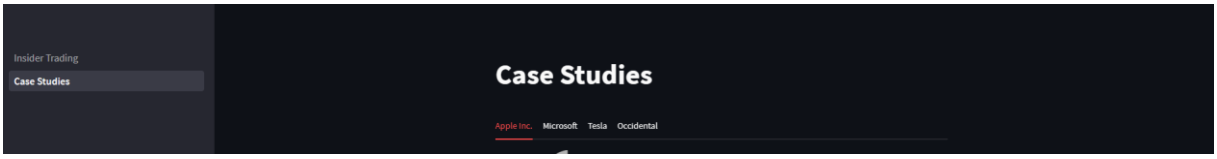
The last challenge that we ran into for the part I was directly responsible for, was the displaying of general insider information directly provided by Yahoo Finance with a table or dataframe. This data was provided to us in the form of a dictionary, and even when converting it into a dataframe through a for-loop, this would not allow us to change the titles of the columns. Therefore, we left it as is. Another annoying part of Streamlit tables and dataframes was that the index couldn't be removed by normal commands.

What my next features or improvements would be:
1. Implement own database that saves the data for the past year that comes in recently.
2. Implement multiple page scraping with „for"-loop, that allows to access double triple or any multiple of 1000 , depending on decision on how often we let the loop run.
3. More visualizations; Integration of a hypothetical insider trading-based portfolio.
4. Integrate Specific person tracker -> see when purchased and sold company

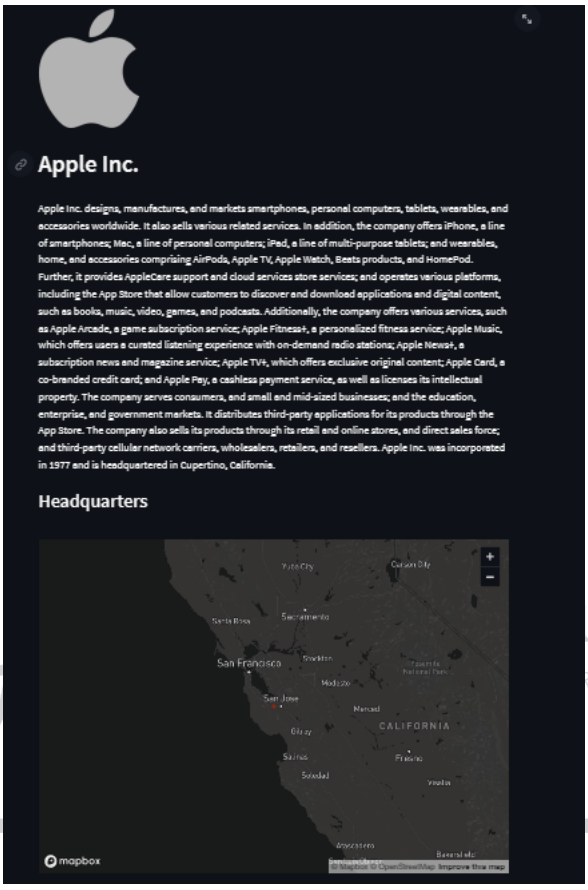
In summary, one could say that it was a journey full of challenges that really forced us to understand more of the logic of Python. Some parts were very tedious and difficult, but it was even more satisfying when we found a solution for a problem, a workaround, or when it finally all worked.

4.3 Marc

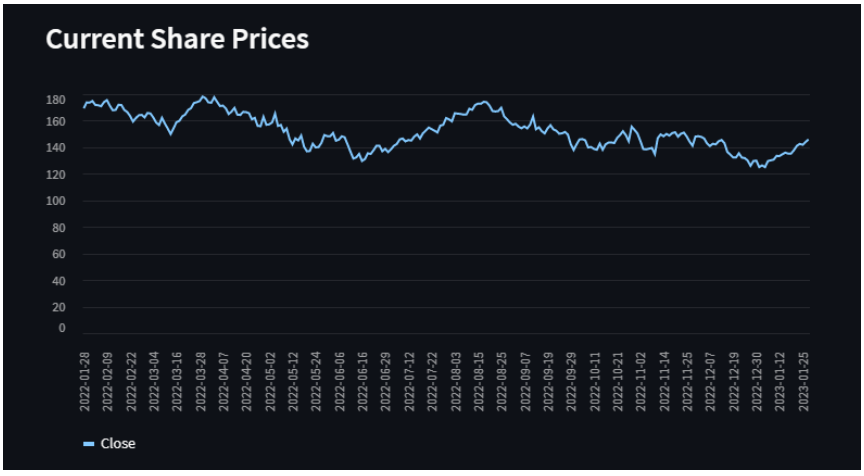
One of our main goals is to give the user a deep insight into the insider trading activities of certain companies. This is the reason why we decided to implement a case studies section in our dashboard. The section is subdivided into different pages, so that the user can choose from interactively. On each page, we give a short introduction to the company and show different graphs, providing different important information on the activity of insiders of a specific company. Finally, we added a conclusive part, where we provide our insight into the specific case to give the user a certain guidance.



As mentioned before, we implemented an additional section for the case studies for the case studies. Using the tools which the streamlit package provides, we create 4 tabs for each company.
To start off with each case study, we give the user an introduction to the company. Each case study starts with a company logo and a short description of the business to give the user an understanding of the business. Additionally, a map with the location of the headquarters is included. The map is interactive and extremely user friendly and by that, increases user attention and interaction with our dashboard.



Another crucial element of each case study section is the share price development over the last 12 months. Providing the user with the share price gives him the chance to compare the share price development with the insider activity, which is also part of our final analysis.



To access the necessary data, we used the yahoo finance API. The API provides us with daily closing prices of the stock over the last year. The data is stored in a pandas DataFrame. An initial problem was the date format, and its took a lot of effort to convert the

dates to the user-friendly format that is now implementation. For the stock prices, we use the line graph provided by the streamlit package. The advantage of the graph is that it is interactive, and users can zoom to specific dates to get a closer look at the share price.

```
driver = webdriver.Chrome()
driver.get("https://www.nasdaq.com/market-activity/stocks/aapl/insider-activity")

driver.minimize_window()

time.sleep(5)

button = driver.find_element(by=By.ID, value="onetrust-accept-btn-handler")
button.click()

wait = WebDriverWait(driver, 10)

table = wait.until(EC.presence_of_element_located((By.CLASS_NAME, "insider-activity__data")))

data = []

rows = table.find_elements(By.TAG_NAME, "tr")
for row in rows:
    cells = row.find_elements(By.TAG_NAME, "td")
    row_data = [cell.text for cell in cells]
    data.append(row_data)

df = pd.DataFrame(data, columns=["Insider Trade", "3 Months", "12 Months"])

df = df.drop(index=0)
```

In the next step, a scrape is necessary to implement and we decided to use the selenium package. The scraping is the process we are most proud of and which was very difficult to implement. In the screenshot provided above, you can see a part of the code responsible for our scraping process. We use the Nasdaq website to scrape our data from, since they provide the latest data on firm specific insider information. After accessing the website, we immediately minimize the window to ensure user-friendliness of our dashboard. Another preparational step is accepting the cookies for the website and our code automatically accepts the cookie settings. To ensure a smooth and reliable scrape, we implement waiting times. We take the hit of longer waiting times to ensure the stability of our dashboard. In the process of creating the code, we struggled several times with errors resulting from missing waiting times.

In the next step, the table that contains the relevant information needs to be identified. The most difficult step was to understand how the table which contains the information is structured and how we can access the data. This process required a lot of sweat and tears. We finally came up with a solution, that finds all the elements with the tag “tr” (table row) in the table element and returns all the matching elements. A loop is started afterwards that iterates through each row element in the table. All elements with the tag “td” (table data) are then found in the current row element. A new list called “row_data” is created and filled with table data which was identified in the previous step. The last line of the loop adds the “row_data” list to the “data” list. This will append the data of each row to the “data” list, so that the final “data” list will contain a list of all the rows in the table, and each row will contain a list of the data in the cells of that specific row.

The data is then transformed into a pandas DataFrame with the column names “Insider Trades”, “3 Months” and “12 Months” since it is way easier to work with the DataFrame. After, the first row of the DataFrame is dropped since it does not contain relevant information. The whole process provides a DataFrame that looks like the following for Apple:

	Insider Trade	3 Months	12 Months
1	Number of Open Market Buys	0	0
2	Number of Sells	1	30
3	Total Insider Trades	1	30

Not only do we scrape the number of insider trades, but also the number of insider shares traded. The overall scraping process remains similar, but the location of the table needs to be redefined. Unfortunately, the class name for both tables is identical, that’s why we need to

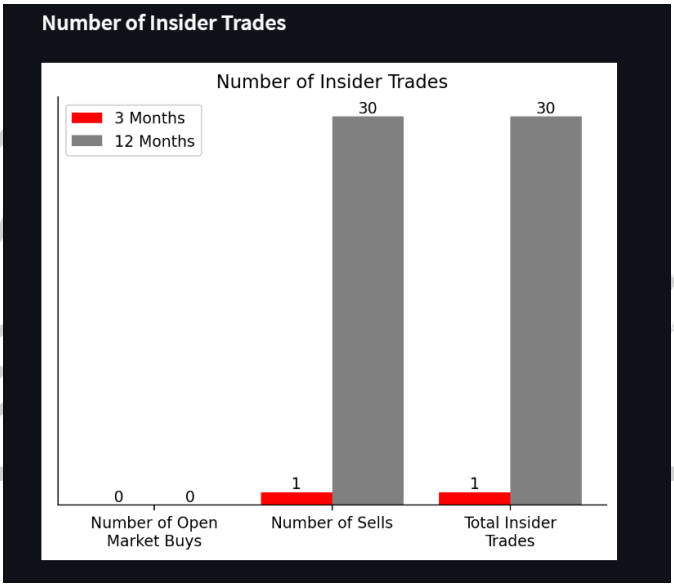
pre-specify the section in which the code is supposed to locate the table. An example of insider shares traded is the following, this time from Tesla Motors:

	Insider Trade	3 Months	12 Months
1	Number of Shares Bought	0	0
2	Number of Shares Sold	41,539,917	59,360,091
3	Total Shares Traded	41,539,917	59,360,091
4	Net Activity	(41,539,917)	(59,360,091)

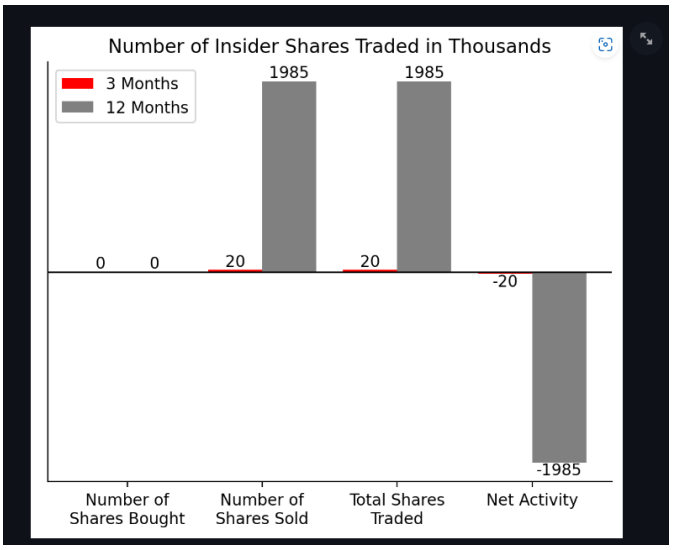
In the following step, we plot the data that we gathered. Again, we encountered another major issue: Python could not recognize the data as integers or floats, but instead as strings. To solve this problem, we defined a new function to convert the strings to integers. It is way more difficult than initially expected, since the parentheses and commas in the numbers result in errors. That's why we first need to get rid of the parentheses and commas, before returning an integer.

```
def remove_parenthesis(val):
    if val.startswith("("):
        return -int(val.replace("(","").replace(")","").replace(",",""))
    else:
        try:
            return int(val.replace(",",""))
        except ValueError:
            return int(val)
```

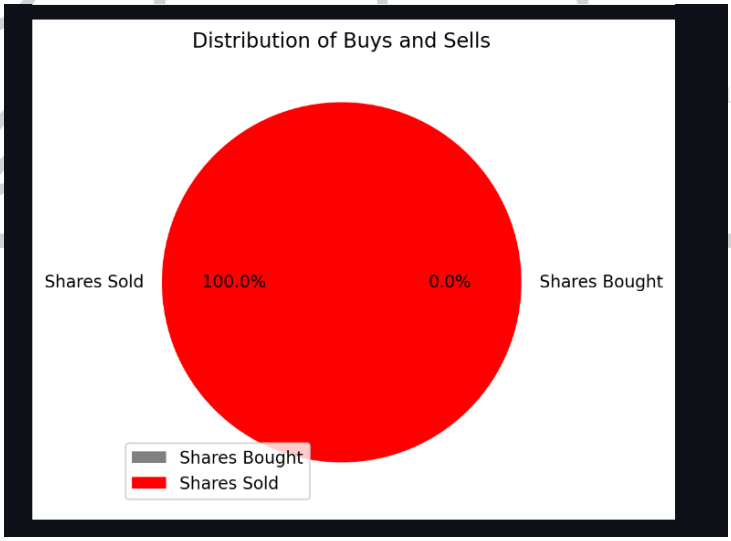
We then apply the function to the DataFrames to ensure the correct data type.



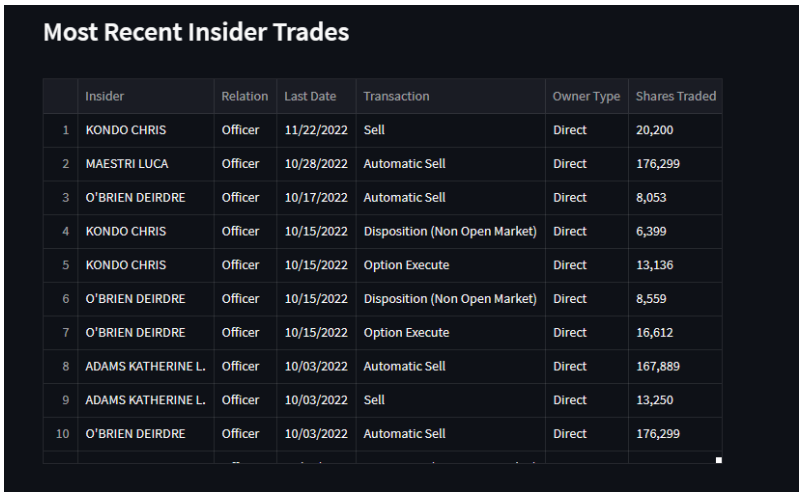
We then use our scraped data to nicely plot the data in our dashboard. We decided to use matplotlib graphs, since they provide the necessary versatility to present our data and bring our intended message across to the user. The advantage of an interactive streamlit graph does not justify the huge decrease in information density. It is important to plot two bars for each category to make users aware that insider trades are usually not equally distributed over time. Another detail of the graphs is the data labels. Data labels are used instead of a y-axis to make the graph more compact and better looking.



The next graph shows information on how many shares were traded by insiders over the last 3 and 12 months. The graph follows a similar structure to the first one, since a bar graph is the best way to present this kind of data. This way, it is easiest to understand for the user. The graph includes four categories, the number of shares bought, number of shares sold, total shares trade and the net activity. In our opinion, it is important to show the difference between the number of trades and number of shares traded to make the user aware of the different sizes of trades. Further, the number of shares traded gives the user an indication of how much volume was actually traded. The net activity is an important indicator to set into relation with the share price over the specific time frame.



Finally, to additionally illustrate the distribution of shares sold to shares bought, we include a pie chart plotting this information. With this graph, we want to make sure that the user can easily get a feeling for the sentiment of the insiders.



	Insider	Relation	Last Date	Transaction	Owner Type	Shares Traded
1	KONDO CHRIS	Officer	11/22/2022	Sell	Direct	20,200
2	MAESTRI LUCA	Officer	10/28/2022	Automatic Sell	Direct	176,299
3	O'BRIEN DEIRDRE	Officer	10/17/2022	Automatic Sell	Direct	8,053
4	KONDO CHRIS	Officer	10/15/2022	Disposition (Non Open Market)	Direct	6,399
5	KONDO CHRIS	Officer	10/15/2022	Option Execute	Direct	13,136
6	O'BRIEN DEIRDRE	Officer	10/15/2022	Disposition (Non Open Market)	Direct	8,559
7	O'BRIEN DEIRDRE	Officer	10/15/2022	Option Execute	Direct	16,612
8	ADAMS KATHERINE L.	Officer	10/03/2022	Automatic Sell	Direct	167,889
9	ADAMS KATHERINE L.	Officer	10/03/2022	Sell	Direct	13,250
10	O'BRIEN DEIRDRE	Officer	10/03/2022	Automatic Sell	Direct	176,299

For the interested users, we include detailed information of the last 15 insider trades. An additional scrape was necessary to generate the table. Again, the Nasdaq website is our source. We generate a pandas DataFrame with the underlying data and then generate an interactive table.

As a last part, we include a short opinion on the company and the insider's activity, as it can be seen above for Tesla. With this part, we want to give users an additional insight into the specific case and help users to put the information into context.

5 Project Summary

	Marvin	Marc	Alex
Main Field of Study	Finance and Accounting	Finance and Accounting	Finance and Accounting
Interests	Chess, Cars, Various Sports	Corporate Finance, Capital Markets, Tennis and Cycling	Football, Hiking, and Capital Markets
Professional Experience	E-Commerce	M&A, Research	Debt Capital Markets, Corporate Finance
Prior Experiences with Programming *On Scale from 1 to 10 (with 1 as lowest)*	3 (Student Initiative, Private Interest)	1	1
Own project started before (not only in the context of Programming)	Yes	Co-founder of a student company	No
Programming Languages (before)	Python, HTML	None	None
Programming Languages (after)	Python, HTML	Python	Python, HTML
Biggest Challenge(s)	Computer Logic, especially the logic of pandas; Different datatypes and formatting; Different Formatting from different Data Providers; Getting Alpha was hell!!; Debugging in streamlit UI	Implementing the web scrape; create user friendly graphs; time management	Understanding Web Scraping; Using CSS flies and understanding the way the code is structured; Start thinking like a project co-manager; Time management (Bachelor Thesis)
Biggest Struggle(s)	Formatting Errors that are dragged through the whole file	Number formats	Understanding CSS and only copying lines of code; General understanding of structure of scraping; Formatting errors
Biggest Learning(s)	I can always make something more efficient, but if it works, it works	Resilience pays off	There is always someone who had the exact same problem before - just google it!
What do you like about the product?	Accessibility, Visualizations of Data (like in the case studies) But just general optical aspects like the Logo; Also, the coverage of data.	Interactive and user-friendly interface that provides unique information	Easy to access and understand, filter options, InsidersInvest Alpha (never seen before)
What will be your next improvement?	More Visualizations; Integration of more Data by scraping more	Add additional case studies, improve the scraping process to make it faster and more efficient	More Case Studies, Weekly Newsletter, UI optimization