Project 3

Lin Huan Jhe, Rajiv Wickramaratne and Nick Carr

What is our project about and how did we get started?

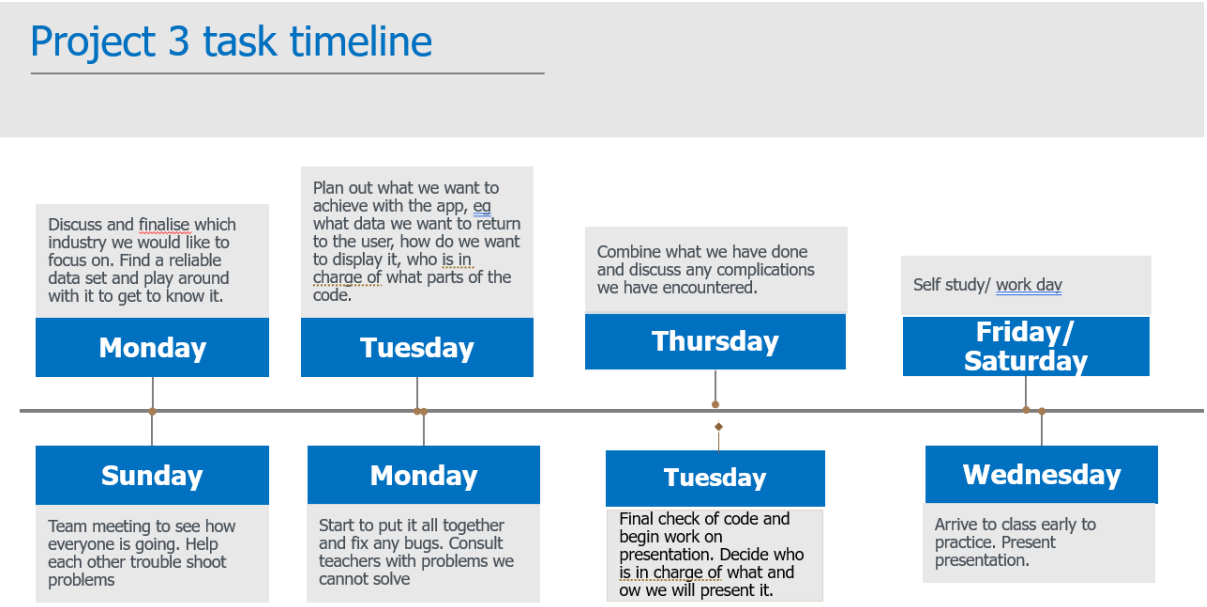
StockInsights is an application which allows you to query the financial information of any Australian company listed. When you query a company by searching through it's ticker, it returns the current share price, it's increase or decrease since the previous closing day, the historic movement of the price which can be changed from 3 months to 5 years, a brief description of the company's details, the assets and liabilities for the past four years, dividend history and cash on hand.

The technologies we used for this assignment include flask, python, JavaScript, html, CSS, a Yahoo finance API called yfinance, a trends API called google trends, web scraping, Plotly, a JavaScript library called anime.js and MongoDB.

The reason we chose this industry as we all have a background and keen interest in finance and wanted to put our freshly learnt skills to building an application similar to ones we have used in the past.

We started this assignment by installing the yfinance and google trend libraries. From there we simply played around with them by reading the documentation. This allowed us to become familiar with the libraries and start cleaning the data by querying the API and deciding which returned data would be relevant for what we want to achieve. Some of the data we decided to return include current share price, share price history, dividends and assets/liabilities on hand.

In order to keep us on track, we created a timeline map in which we planned out the foreseen steps that needed to be taken so we could monitor our progress and see where we can help each other. As we all have full time jobs, this was critical to helping us with time and project management. Below is the timeline we created to this



Installs required

The install requirements to run the application successfully include the following:

- Pillow 9.2.0
- Bs4 0.0.1
- Matplotlib 3.2.2
- Numpy 1.21.5
- Pandas 1.3.5
- Pandas_datareader 0.10.0
- Plotly 5.10.0
- Pymongo 3.12.0
- Pytrends 4.8.0
- Requests 2.28.1
- Wordcloud 1.8.2.2
- Yfinance 0.1.74

How our code operates

- Within app.py, the function dashboard() is defined and within this function we have our dividends variables. The dividends data is obtained by calling the script Fundermentals.py, abbreviated as funds, and sending the stockname we are interested in to the function within funds called get_dividends.

```
@app.route('/dashboard')
def dashboard():
    try:
        stockcode1=request.args['stockcode'] # 'stockcode' is posted from index.html when users key in stock code and click "SUBMIT"

        adjustedName = f'{stockcode1.upper()}.AX' #Standarize stock code to match yfinance requirement

        # dividend plot
        dividend = funds.get_dividends(adjustedName)   # Parsing 'adjustedName' to fundamentals.py, then plotlylayout.py, then return back
        plotlyplot_dividend = json.dumps(dividend, cls=plotly.utils.PlotlyJSONEncoder) # To display the plot as html we have to put into a json format.

        #cash plot
        cash = funds.get_cash(adjustedName)  # Parsing 'adjustedName' to fundamentals.py, then plotlylayout.py, then return back

        plotlyplot_cash = json.dumps(cash, cls=plotly.utils.PlotlyJSONEncoder) # To display the plot as html we have to put into a json format.

        # Asset/Liabilities plot
        AL = funds.get_AL(adjustedName)   # Parsing 'adjustedName' to fundamentals.py, then plotlylayout.py, then return back

        plotlyplot_AL = json.dumps(AL, cls=plotly.utils.PlotlyJSONEncoder) # To display the plot as html we have to put into a json format.
```

- As seen below, get dividends obtains the max historical dividends data, cleans it by only keeping the values of dividends greater than 0, negating any 0 or null fields and finally transforms it via resetting the index which is necessary for it to work with plotly.

```python
def get_dividends(ticker):

    stock = yf.Ticker(ticker)

    ## Dividends

    dividends = stock.history(period='max')['Dividends'] #Source all the records of dividends (daily type)
    dividends = pd.DataFrame(dividends).reset_index() # For plotly function
    dividends = dividends.loc[dividends['Dividends']>0] #companies normally paid dividens twice a year
    dividends

    return ply.create_plotly_bar(dividends)

```

- Finally the data is sent to plotlylayout.py, abbreviated by ply, to generate the plotly bar graph.

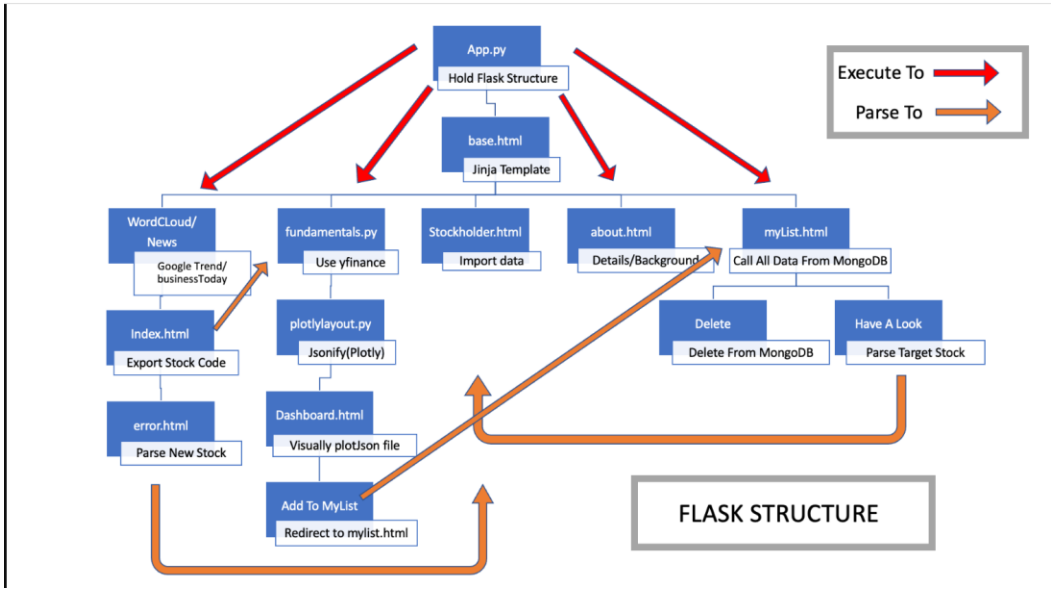
```python
def create_plotly_bar(data):

    fig_dividend = px.bar(data,
                          x="Date",
                          y="Dividends",
                          title="Dividends",
                          barmode='group'
                          )

    fig_dividend.update_layout(title_font=dict(color='navy',family = 'Arial Bold',size=26),yaxis_tickprefix='$')
    return fig_dividend
```

- We decided to use the JavaScript library anime.js. This is a lightweight JavaScript animation library that allows you to create complex animations with a single yet powerful API. It worked well within our project, as we were able to implement it into our homepage, which you will see in our demonstration.

- We have leveraged mongoDB to allow users to store their searched stock information. We chose mongoDB because it met our needs, where we did not require the database to have predefined data structures, rather we wanted to natively structure our data and allow for changes in the future without it affecting the existing database. If we were to change the structure of what we wished to save, through the use of pymongo, initalising a connection to our local mongoDB database, we successfully are able to store the queried stock data when the user requests so.

```python
@app.post('/<id>/addToList/')
def addToList(id):

    # Create a database in MongoDB
    conn = 'mongodb://localhost:27017'

    client = pymongo.MongoClient(conn) #Connect to MongoDB

    db = client.flask_db #Connect to flask_db

    stocks = db.stocks

    # Calling data from yfianace about what we want to store in MongoDB
    lastRecentPrice = yf.Ticker(id).history()['Close'][-1]

    lastRecentPrice =round(lastRecentPrice,2) #round the price

    stockPirce = yf.Ticker(id).history(period="1y")['Close'] #Get the past 1 year stock price

    max52 =round(max(stockPirce),2)

    min52 =round(min(stockPirce),2)

    nowDate = dt.datetime.now().strftime("%d/%m/%Y") # Parse time data type to string type to mark search day

    stocks.insert_one({'StockCode': id, 'Price': lastRecentPrice, 'Weeks52High':max52, 'Weeks52Low':min52, 'LastSearchedDate': nowDate})

    client.close()

    # Stay at the same web page
    return redirect(url_for('mylist'))
```

- The below image is the final code structure and how the whole application operates. Every step is as crucial as the next



Summary

This assignment has been a great venture for us all, as it has allowed us to revise and put into action our newly gained data analytic skills which we have precured over the past few months. It has allowed us to find our strengths and weaknesses within all the new languages and libraries that we have learnt, which then allowed us to turn to each other and ask/ give support and guidance.

We believe that we have filled in many gaps of knowledge needed to make us better data analysts, as well as making us better collaborators within a group environment.

Of course many challenges were met along the way. Coordinating our schedules outside of class time was a challenge, as we all work full time, as well as other life commitments. There was also time pressure, as we had to review and, in some cases, relearn what we have studied to apply it to our assignment. This created a bit of a concern for us as we were not too sure if we were on the right track to deliver a completed final product.

In terms of the technical challenges, the main one was the continual errors in code we encountered and then having to find out where we went wrong and why. However, we know that these errors make us better at understanding the code and what syntax is required.

In conclusion, we are very proud with our end result and the journey of ups and downs we met along the way to get the desired functioning application. We definitely learnt a lot in terms of working in a team, putting our knowledge to the test and time management.