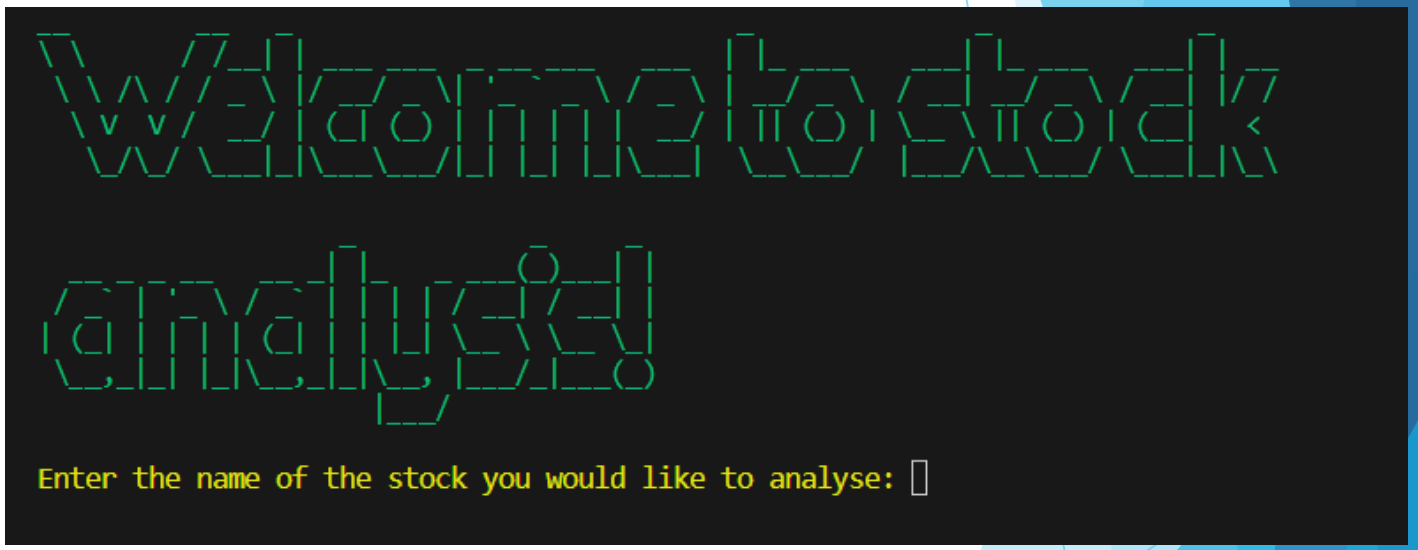
# Stock Analysis Terminal Application

Timothy Lin

# Purpose

- The purpose of the application is to help users better assess companies and stocks.
- It does this by calculating, analysing and comparing 3 key metrics with their respective industry averages.



# Data Structures

- The application utilises a dictionary to store sector averages.
- The Key-Value pair provides optimal storage and retrieval abilities to the code.
- By having a nested dictionary, we can access a sector and simultaneously access another key or ‘metric’.
- This helps with efficiency and is simple to use.

```
SECTOR_AVERAGES = {
        'healthcare': {'pe_ratio': 25, 'debt_to_equity': 0.6},
        'finance': {'pe_ratio': 10, 'debt_to_equity': 0.6},
        'energy': {'pe_ratio': 14, 'debt_to_equity': 0.5},
        'real_estate': {'pe_ratio': 11, 'debt_to_equity': 0.3},
        'retail': {'pe_ratio': 12, 'debt_to_equity': 0.7},
        'tech': {'pe_ratio': 13, 'debt_to_equity': 1},
    }

def sector_averages(sector):
    return SECTOR_AVERAGES.get(sector, {})
```

# Input

- Here, the application allows the user to enter in the financial data of their chosen company.
- Because the financial data must be in the form of a float, the get_float_input function also checks to make sure this is the case and prompts the user to try again if they enter something else.

```
def get_float_input(prompt):
    while True:
        try:
            value = float(input(Fore.YELLOW + prompt))
            return value
        except ValueError:
            print("Please enter a valid number!")
```

```
revenue = get_float_input("Enter the company's revenue: $")
expenses = get_float_input("Enter the company's expense: $")
operating_cash_flow = get_float_input("Enter the company's operating cash flow: $")
capital_expenditure = get_float_input("Enter the company's capital_expenditure: $")
assets = get_float_input("Enter the company's assets: $")
liabilities = get_float_input("Enter the company's liabilities: $")
market_capitalisation = get_float_input("Enter the company's market cap: $")
```

# Calculations

- Once the input is received, the application will calculate profit, free cash flow, profit to earnings ratio and debt to equity ratio.
- The metrics are rounded to two decimal places. This is because the additional decimal points do not assist in the case of analysing stocks with these metrics.
- There is also error handling, to ensure code still runs when the denominators are zero in the functions.

```
# Analysis data is calculated here

profit = revenue - expenses
free_cash_flow = operating_cash_flow - capital_expenditure

try:
    pe_ratio = market_capitalisation / profit
except ZeroDivisionError:
    pe_ratio = float('inf')
    print("As profit equals zero, PE ratio cannot be calculated")

try:
    debt_to_equity = liabilities / (assets - liabilities)
except ZeroDivisionError:
    debt_to_equity = float('inf')
    print("As equity equals zero, Debt to equity ratio cannot be calculated")


analysis_data = {
    'pe_ratio': round(pe_ratio, 2) if pe_ratio != float('inf') else 'N/A',
    'debt_to_equity': round(debt_to_equity, 2) if debt_to_equity != float('inf') else 'N/A',
    'free_cash_flow': round(free_cash_flow, 2)
}

print(Fore.BLUE + f"Here are the company's metrics for your stock: {analysis_data}" )
```

# Comparisons

- By using if statements the application can compare outputted metrics with the industry averages in the dictionary created earlier in sector_averages.
- The if statements also allow for different answers to be printed out depending on the outcome.
- Based on the outcome, the user can form their own opinion and assessment.

```
if sector_averages_data:
    # Handling PE ratio analysis
    if pe_ratio == float('inf'):
        valuation = "N/A"
        expectations = "The company is not profitable, meaning PE rato cannot be calculated"
    elif pe_ratio > sector_averages_data['pe_ratio']:
        valuation = "expensive"
        expectations = "This means that investors expect higher than average growth in the future"
    elif pe_ratio < sector_averages_data['pe_ratio']:
        valuation = "cheap"
        expectations = "This means that investors expect lower than average growth in the future"
    else:
        valuation = "fair"
        expectations = "This means that the stock is a fair price relative to it's future growth prospects"

sector_averages_data = sector_averages(sector)

observation = "0"
debt_levels = "0"

    # Handling debt to equity analysis
if sector_averages_data:
    if debt_to_equity == float('inf'):
        debt_levels = "N/A"
        observation = "The company has zero equity, meaning debt to equity can not be calculated"
    elif debt_to_equity > sector_averages_data['debt_to_equity']:
        debt_levels = "high"
        observation = "Careful! This company is highly leveraged compared to its peers."
    elif debt_to_equity < sector_averages_data['debt_to_equity']:
        debt_levels = "low"
        observation = "This company does not have much debt compared to its peers."
    else:
        debt_levels = "average"
        observation = "This company has a fair amount of debt compared to its peers."

sector_averages_data = sector_averages(sector)

cash_flow_levels = "0"
outlook = "0"

if sector_averages_data:
    if free_cash_flow < 0:
        cash_flow_levels = "negative"
        outlook = "Careful! This company does not produce postive cash flow, meaning it will have to raise capital or increase debt when they run out of cash! "
    elif free_cash_flow >= 0:
        cash_flow_levels = "positive"
        outlook = "Great! this company produces enough cash flow to be self sustaining! It can use this cash to invest, expand or pay shareholders!."

    print(Fore.BLUE + f"The stock is considered {valuation} relative to its industry peers. {expectations}. ")
```

# Results

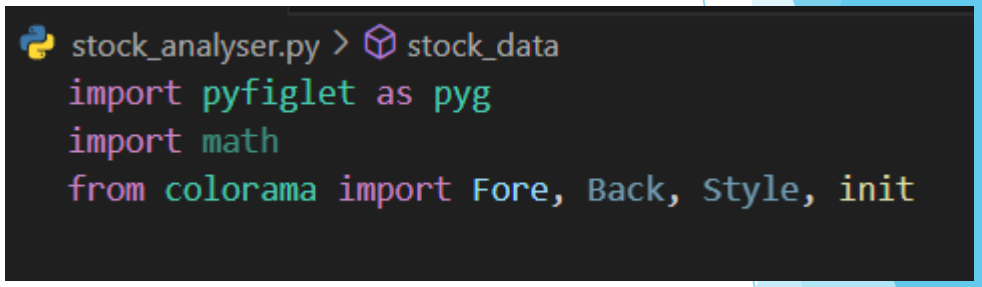
- The user will have an option to analyse additional stocks if they choose to.
- Once the user has finished comparing stocks, the results will all be appended to a text file.
- This text file will then be opened with notepad and can be saved by the user if they choose to.

```
    another_company = input(Fore.CYAN + "Do you want to analyze another company? (y/n): ")
    if another_company.upper() != "Y":
        break
```

```
def save_results_to_file(analysis_data, valuation, expectations, stock_name):
    try:
        with open("analysis_results.txt", "a") as file:
            file.write("\n")
            file.write(f"Stock: {stock_name}\n")
            file.write(f"PE Ratio: {analysis_data['pe_ratio']}\n")
            file.write(f"Debt to equity: {analysis_data['debt_to_equity']}\n")
            file.write(f"Free Cash Flow: {analysis_data['free_cash_flow']}\n")
            file.write(f"Valuation: {valuation}\n")
            file.write(f"Expectations: {expectations}\n")
    except IOError as e:
        print(f"Error: {e}")
```

# Dependencies

- The application does have a couple of dependencies.
- Colorama was imported to assist with the styling and coloring of the application.



# THANK YOU!