Importing modules

In [1]:

```
import mysql.connector
from mysql.connector import errorcode
import pandas as pd
import numpy as np
from datetime import datetime
import numpy_financial
import dateutil
```

Loading personal infomation as variable and Automated_Authentication

In [3]:

```
%run ./Functions_Folder/Personal_Folder/Personal_Information.ipynb
%run ./Functions_Folder/Automated_Authentication.ipynb
```

```
Loaded Personal Variables
Loaded decoded_content
```

Creating an function to collect basic information from MySQL such as Ticker, Sector, and Industy

In [2]:

```
def SectorID(symbol):
    cnx = mysql.connector.connect(user= MySQL_username, password = MySQL_password, data
    cursor = cnx.cursor()

    query = ("select Ticker, Sector, Industy "
        "from company_info "
        "where Ticker = '{}'").format(symbol)

    cursor.execute(query)
    for (Ticker, Sector, Industy) in cursor:
        x =1

    return Sector

# symbol = 'PAYC'
# SectorID(symbol)
```

Loading more functions shared with other files

In [4]:

```
%run ./Functions_Folder/Functions.ipynb
%run ./Functions_Folder/Forecasting_Functions.ipynb
```

```
Creating table Company_Info: already exists.
Creating table CalcTable: already exists.
Creating table Yahoo_Forecast: already exists.
Creating table futureEarningsDate: already exists.
```

Creating another function that is calcualting the future value of the a given ticker based how much the user has spent on its purchase of the material

In [5]:

```
# symbol = 'TSLA'
# longQuantity = 2

def FV_function(symbol, longQuantity):
    today = datetime.today()
    count = longQuantity
    Current_FV = 0
    sixM_FV = 0
    one_and_a_half_YR_FV = 0
    for single_transaction in transaction_dict[symbol]:
        amount = transaction_dict[symbol][single_transaction]['amount']
        if count > 0:
            count = count-amount
            amount = transaction_dict[symbol][single_transaction]['amount']
            price = transaction_dict[symbol][single_transaction]['price']


            Open_date = datetime.strptime(single_transaction, '%Y-%m-%d')
            change = dateutil.relativedelta.relativedelta(today, Open_date)
            Years = change.years
            Months_in_Years = change.months/12
            Days_in_Years = change.days/365.25
            Total = Years+Days_in_Years+Months_in_Years
            single_transation_FV = round(numpy_financial.fv(0.13, Total, 0, -price*amou
            Current_FV = Current_FV+single_transation_FV
            sixM_transation_FV = round(numpy_financial.fv(0.13, Total+.5, 0, -price*amo
            sixM_FV = sixM_transation_FV + sixM_FV
            one_and_a_half_YR_transation_FV = round(numpy_financial.fv(0.13, Total+1, 0
            one_and_a_half_YR_FV =one_and_a_half_YR_FV + one_and_a_half_YR_transation_F

    Current_FV = Current_FV/longQuantity
    sixM_FV = sixM_FV/longQuantity
    one_and_a_half_YR_FV = one_and_a_half_YR_FV/longQuantity


    return (Current_FV, sixM_FV, one_and_a_half_YR_FV, Total, change)

# FV_function(symbol, longQuantity)
```

Collecting the history of the trasactions of the user from their account

In [6]:

```
Resource_URL = 'https://api.tdameritrade.com/v1/accounts/{}/transactions'.format(accoun
Query_Parameters = {'type': 'BUY_ONLY'}
Header_Parameters = {'Authorization':decoded_content['Authorization']}

transactions = requests.get(r'{}'.format(Resource_URL), headers = Header_Parameters, pa
```

The positions and active orders in the current account

In [7]:

```
Resource_URL = 'https://api.tdameritrade.com/v1/accounts/{}'.format(account_num)
Query_Parameters = {'fields':'positions,orders'}
Header_Parameters = {'Authorization':decoded_content['Authorization']}

balance_positions_orders = requests.get(r'{}'.format(Resource_URL), headers = Header_Pa
```

In [8]:

```
# Creating Transaction History into Dictionary
transaction_dict = {}
num = 0
for num in range(len(transactions)):
    BuyorSell = transactions[num]['description']
    if BuyorSell =='BUY TRADE':
        settlementDate = transactions[num]['settlementDate']
        orderId = transactions[num]['orderId']
        transactionId = transactions[num]['transactionId']

        symbol = transactions[num]['transactionItem']['instrument']['symbol']
        amount = transactions[num]['transactionItem']['amount']
        price =  transactions[num]['transactionItem']['price']

        # if more than one order for the symbol
        try:
            x = transaction_dict[symbol]
        except:
            transaction_dict[symbol] = {}

        # if more than one order exist on the same settlement date
        try:
            x = transaction_dict[symbol][settlementDate]
        except:
            transaction_dict[symbol][settlementDate] = {}

        transaction_dict[symbol][settlementDate]['amount'] = amount
        transaction_dict[symbol][settlementDate]['price'] = price
        transaction_dict[symbol][settlementDate]['orderId'] = orderId
        transaction_dict[symbol][settlementDate]['transactionId'] = transactionId

# transaction_dict
```

Updating what the future 5 yr forecast from analysts on Yahoo

In [9]:

```
# Updating forecasting and prices for postions and orders
Tickers_lists = []
positions_list = balance_positions_orders['securitiesAccount']['positions']
orders_list = balance_positions_orders['securitiesAccount']['orderStrategies']

for num in range(len(positions_list)):
    symbol = positions_list[num]['instrument']['symbol']
    if symbol != 'MMDA1':
        Tickers_lists.append(symbol)
for num in range(len(orders_list)):
    instruction = orders_list[num]['orderLegCollection'][0]['instruction']
    status = orders_list[num]['status']
    Type = 'orders'
    if status == 'PENDING_ACTIVATION'and instruction == 'BUY':
        symbol = orders_list[num]['orderLegCollection'][0]['instrument']['symbol']
        Tickers_lists.append(symbol)

Updating_Yahoo_Table(Tickers_lists)
```

```
We have 0 companies remaining! We are 100.0% done!
```

Creating a database that hold all the users current positions and orders

In [10]:

```
# Creating the Master Database
Header = ['Type', 'Symbol', 'Price', 'Quantity', 'Total Cost', 'Market Value', 'Profit_
Dataset = []

for num in range(len(positions_list)):
    try:
        symbol = positions_list[num]['instrument']['symbol']
        Type = 'positions'

        if symbol != 'MMDA1':
            NPV_Data = NPV_Function(symbol)
            averagePrice = positions_list[num]['averagePrice']
            longQuantity = positions_list[num]['longQuantity']
            marketValue = positions_list[num]['marketValue']
            totalCost = averagePrice*longQuantity
            Profit_Lost_perc =  round(((marketValue-totalCost)/totalCost)*100, 2)
            Sector = Identifying_Sector_and_Industry(symbol)[1]
            Price = NPV_Data[4]
            FMV = NPV_Data[5]
            FMV_perc = round((((FMV-averagePrice)/averagePrice)*100),2)
            #Compounding Interest Rate
            FV_Data = FV_function(symbol, longQuantity)
            Current_FV = FV_Data[0]
            Current_FV_perc = round((((marketValue-(Current_FV*longQuantity))/(Current_
            sixM_FV = FV_Data[1]
            sixM_FV_perc = round((((marketValue-(sixM_FV*longQuantity))/(sixM_FV*longQu
            one_and_a_half_YR_FV = FV_Data[2]
            one_and_a_half_YR_FV_perc = round((((marketValue-(one_and_a_half_YR_FV*long
            Time = round(FV_Data[3],2)

        else:
            symbol = 'Remaining Cash'
            averagePrice = None
            longQuantity = None
            marketValue = positions_list[num]['marketValue']
            totalCost = positions_list[num]['marketValue']
            Sector = 'Remaining Cash'
            FMV = None
            No_CIR_perc = None
            Profit_Lost_perc =  None
            Current_FV = None
            sixM_FV = None
            one_and_a_half_YR_FV = None
            Current_FV_perc = None
            sixM_FV_perc = None
            Time = None
            FMV_perc =None
            one_and_a_half_YR_FV_perc = None


        Dataset.append([Type,symbol,averagePrice,longQuantity,totalCost,marketValue, Pr

    except:
        print(positions)

for num in range(len(orders_list)):

    instruction = orders_list[num]['orderLegCollection'][0]['instruction']
    status = orders_list[num]['status']
    Type = 'orders'

    if status == 'PENDING_ACTIVATION'and instruction == 'BUY':
        longQuantity = orders_list[num]['orderLegCollection'][0]['quantity']
        symbol = orders_list[num]['orderLegCollection'][0]['instrument']['symbol']
        averagePrice = orders_list[num]['price']
        Sector = SectorID(symbol)
        totalCost = longQuantity*averagePrice
        marketValue = longQuantity*averagePrice
        No_CIR_perc = None
        Profit_Lost_perc =  None
        Current_FV = None
        sixM_FV = None
        one_and_a_half_YR_FV = None
        Current_FV_perc = None
        sixM_FV_perc = None
        Time = None
        one_and_a_half_YR_FV_perc = None


        NPV_Data = NPV_Function(symbol)
        FMV = NPV_Data[5]
        FMV_perc = round((((FMV-averagePrice)/averagePrice)*100),2)




        Dataset.append([Type,symbol,averagePrice,longQuantity,totalCost,marketValue, Pr
Dataset = pd.DataFrame(Dataset, columns = Header)
Dataset
```

Out[10]:

	Type	Symbol	Price	Quantity	Total Cost	Market Value	Profit_Lost_perc	Sector	FMV	FMV (%)
0	positions	PAYC	294.420	1.0	294.42	293.14	-0.43	Industrials	176.35	-40.10
1	positions	LTHM	22.040	4.0	88.16	82.44	-6.49	Basic Materials	32.52	47.55
2	positions	PLMR	57.790	2.0	115.58	99.90	-13.57	Financial Services	63.68	10.19
3	positions	DDOG	65.205	2.0	130.41	195.32	49.77	Technology	29.14	-55.31
4	positions	ENPH	204.855	2.0	409.71	246.06	-39.94	Technology	87.13	-57.47
5	positions	AMPH	54.460	2.0	108.92	104.06	-4.46	Healthcare	68.62	26.00
6	positions	CRWD	118.390	1.0	118.39	166.23	40.41	Technology	76.22	-35.62
7	positions	HRMY	34.720	3.0	104.16	109.92	5.53	Healthcare	52.78	52.02
8	positions	TSLA	129.420	1.0	129.42	251.70	94.48	Consumer Cyclical	32.70	-74.73
9	positions	ZS	88.100	1.0	88.10	158.30	79.68	Technology	51.48	-41.57
10	positions	Remaining Cash	NaN	NaN	577.49	577.49	NaN	Remaining Cash	NaN	NaN
11	orders	HALO	32.510	3.0	97.53	97.53	NaN	Healthcare	63.67	95.85
12	orders	OWL	9.960	10.0	99.60	99.60	NaN	Financial Services	13.65	37.05

Categorizing all active positons into each sector to see how the portfolio is diversified

In [11]:

```
Positions = Dataset[Dataset['Type'] == 'positions']
Cash_Allocated_df = Positions[['Sector', 'Market Value']]
Cash_Allocated_df = Cash_Allocated_df.groupby(Positions['Sector']).sum()
Total_Cash = Cash_Allocated_df['Market Value'].sum()
Cash_Allocated_df['Percentage (%)'] = (Cash_Allocated_df['Market Value']/Total_Cash) *1
Cash_Allocated_df['Max Cash'] = (.13*Total_Cash)-Cash_Allocated_df['Market Value']
Remaining_Cash = Cash_Allocated_df.at['Remaining Cash','Market Value']
print("Remaining Cash: {}".format(Remaining_Cash))
print("Unlisted Sectors can hold up to: ${} (per stock -> ${})".format(round(.13*Total_
Cash_Allocated_df = Cash_Allocated_df.drop(index = 'Remaining Cash')
Cash_Allocated_df
```

```
Remaining Cash: 577.49
Unlisted Sectors can hold up to: $296.99 (per stock -> $99.0)
```

Out[11]:

	Market Value	Percentage (%)	Max Cash
Sector			
Basic Materials	82.44	3.608572	214.5528
Consumer Cyclical	251.70	11.017439	45.2928
Financial Services	99.90	4.372833	197.0928
Healthcare	213.98	9.366355	83.0128
Industrials	293.14	12.831355	3.8528
Technology	765.91	33.525493	-468.9172

List of stocks the customer is recommended to place a closing limit order on

In [12]:

```
# Removing the Remaining Cash
Negative_Outlook = Dataset[Dataset['FMV (%)']<0]
Negative_Outlook = Negative_Outlook.sort_values(by = 'FMV (%)')
Negative_Outlook = Negative_Outlook[Negative_Outlook['1_1/2YR_FV%']<0]
Negative_Outlook = Negative_Outlook[['Symbol', 'Price', 'Profit_Lost_perc', 'FMV', 'FMV
Negative_Outlook
```

Out[12]:

	Symbol	Price	Profit_Lost_perc	FMV	FMV (%)	Time (yrs)	1_1/2YR_FV	1_1/2YR_FV%
4	ENPH	204.855	-39.94	87.13	-57.47	0.63	246.595	-25.05
0	PAYC	294.420	-0.43	176.35	-40.10	0.33	346.360	-15.37