The purpose of this Juputer notebook is to centralized the codes made to extract, prepare, transform the data obtained from the sources to get the final dataframe used to train and test the machine learning algorithms utilized in the project.

There are 2 sources to get all variables in the dataset.

1. Yahoo Finance
2. Provided by Albert

Data Source for demographics

The data source for demographic data used in this project was Yahoo Finance.

We will be using the yfinance library to retrieve the necessary demographics for each company from Yahoo Finance

In [ ]:
```
import yfinance as yf
import pandas as pd
import numpy as np
```

We create a list of the tickers of the stocks that we wish to make API calls to. This simplifies the process of retrieving the values

In [ ]:
```
tickers = ['aapl', 'amd', 'amzn', 'ba', 'baba', 'bac', 'c', 'csco', 'cvx', 'dis', 'f', 'ge',
           'googl', 'ibm', 'intc', 'jnj', 'jpm', 'ko', 'mcd', 'meta', 'msft', 'nflx', 'nvda',
           'pfe', 'pltr', 't', 'tsla', 'vz', 'wmt', 'xom']
```

Since we will be calling all the available information from yfinance using .info, we will need to fitler it down. We can achieve this by creating a function to help us

In [ ]:
```
def extract_demographics(x, to_extract):
    # Given a dictionary and a list of keys to extract, will return as a dictionary
    # if the value does not exist on the yahoo finance site, np.nan will fill the value
    result_dict = {key: x.get(key, np.nan) for key in to_extract}
    return result_dict
```

Define the ratios and demographics we wish to extract from the yfinance .info function

In [ ]:
```
ratios_to_extract = ['symbol', 'sector', 'trailingPE' , 'priceToBook', 'debtToEquity', 'freeCashflow',
                     'pegRatio' , 'returnOnEquity']
```

Prepare the results dataframe to concat each result onto

In [ ]:
```
demographic_df = pd.DataFrame()
```

We create a for loop now to complete the task of retrieving all the ratios and demographics for the companies we wish to know about:

- Step 1: Retrieve the yf.Ticker info for the stock (This will retrieve all available information)
- Step 2: Apply the extract_demographics function to keep only the ratios and demographics we are interested in
- Step 3: Convert the dictionary of values we wish to keep into a dataframe
- Step 4: Concatenate the results of this onto our final demographic_df dataframe

In [ ]:
```
for i in range(len(tickers)):
    temp_ticker = yf.Ticker(tickers[i]).info
    temp_row = extract_demographics(temp_ticker, ratios_to_extract)
    temp_df  = pd.DataFrame([temp_row])
    demographic_df = pd.concat([demographic_df, temp_df],ignore_index=True)
# As for 4-11-2023 the demographic scarpping is unavaiable due to an error with the yahoo finance API, no soluti
```

In [ ]:
```
demographic_df
```

Out[ ]:

	symbol	sector	trailingPE	priceToBook	debtToEquity	freeCashflow	pegRatio	returnOnEquity
0	AAPL	Technology	29.134003	45.153164	181.305	9.068050e+10	4.46	1.60093
1	AMD	Technology	NaN	2.813442	5.187	3.586375e+09	4.64	-0.00045
2	AMZN	Consumer Cyclical	107.774994	7.911060	103.371	2.114013e+10	0.84	0.08714
3	BA	Industrials	NaN	NaN	NaN	9.339625e+09	-0.73	NaN
4	BABA	Consumer Cyclical	19.291285	0.211622	17.115	1.046624e+11	46.89	0.06978
5	BAC	Financial Services	8.060345	0.875168	NaN	NaN	2.33	0.10782
6	C	Financial Services	6.610143	0.426165	NaN	NaN	2.11	0.06676
7	CSCO	Technology	17.374594	4.889989	21.311	1.807362e+10	2.43	0.29986
8	CVX	Energy	10.474953	1.948202	13.505	2.339550e+10	-1.72	0.19263
9	DIS	Communication Services	67.260160	1.551023	42.535	5.497875e+09	1.27	0.02721
10	F	Consumer Cyclical	12.079207	1.117933	332.179	-8.953750e+08	-3.69	0.09125
11	GE	Industrials	12.215135	3.942703	73.773	5.923250e+09	1.69	0.30892
12	GOOGL	Communication Services	28.418300	6.166501	11.017	6.825513e+10	1.29	0.23329
13	IBM	Technology	62.714897	6.047599	270.774	1.007137e+10	5.26	0.10396
14	INTC	Technology	NaN	1.436508	46.947	-9.475375e+09	8.17	-0.00911
15	JNJ	Healthcare	32.724697	5.589130	59.681	2.150512e+10	3.22	0.17090
16	JPM	Financial Services	9.456298	1.499760	NaN	NaN	-2.27	0.15982
17	KO	Consumer Defensive	23.875519	9.564495	156.776	9.308375e+09	3.96	0.40157
18	MCD	Consumer Cyclical	25.020294	NaN	NaN	5.987112e+09	2.97	NaN
19	META	Communication Services	34.750880	5.732202	27.589	2.024775e+10	0.83	0.17356
20	MSFT	Technology	32.975235	11.515425	38.522	4.726900e+10	2.32	0.38824
21	NFLX	Communication Services	41.800873	7.455894	74.428	1.786123e+10	1.55	0.20265
22	NVDA	Technology	99.074880	37.122818	39.831	9.954625e+09	0.57	0.40217
23	PFE	Healthcare	8.736702	1.872755	66.142	1.235237e+10	-0.79	0.23068
24	PLTR	Technology	NaN	10.181818	8.105	4.359732e+08	0.78	-0.01611
25	T	Communication Services	NaN	1.074160	142.906	1.331925e+10	23.83	-0.05967
26	TSLA	Consumer Cyclical	73.688760	15.873115	11.136	3.028125e+09	8.23	0.27139
27	VZ	Communication Services	6.658000	1.470212	188.910	1.147488e+10	16.57	0.23394
28	WMT	Consumer Defensive	31.138462	5.478970	78.146	1.874350e+10	3.94	0.16325
29	XOM	Energy	9.180800	2.308033	20.047	3.487800e+10	-1.34	0.27202

These values are a snapshot of the current values on yfinance and not a historical capture of the metrics.

Data Source for commodities

The code for extracting commodity data is currently unavailable. However, the CSV files were directly provided by Albert Wong. You can find these CSV files in the following link: https://www.dropbox.com/scl/fo/n638k3vvnic2pss80zyvq/h?rlkey=3ilmw5fiehotnlqbeneiu249t&dl=0

In [ ]:
```
import pandas as pd
import numpy as np
import holidays
from functools import reduce
```

Merging the stocks csv files into one table

In [ ]:
```
#reading stock data
# The csv'S can be found in the next link : https://www.dropbox.com/scl/fo/n638k3vvnic2pss80zyvq/h?rlkey=3ilmw5f
aapl = pd.read_csv('stocks/AAPL.csv', delimiter=";")
amd = pd.read_csv('stocks/AMD.csv', delimiter=";")
amzn = pd.read_csv('stocks/AMZN.csv', delimiter=";")
ba = pd.read_csv('stocks/BA.csv', delimiter=";")
baba = pd.read_csv('stocks/BABA.csv', delimiter=";")
bac = pd.read_csv('stocks/BAC.csv', delimiter=";")
c = pd.read_csv('stocks/C.csv', delimiter=";")
csco = pd.read_csv('stocks/CSCO.csv', delimiter=";")
cvx = pd.read_csv('stocks/CVX.csv', delimiter=";")
dis = pd.read_csv('stocks/DIS.csv', delimiter=";")
f = pd.read_csv('stocks/F.csv', delimiter=";")
ge = pd.read_csv('stocks/GE.csv', delimiter=";")
googl = pd.read_csv('stocks/GOOGL.csv', delimiter=";")
ibm = pd.read_csv('stocks/IBM.csv', delimiter=";")
intc = pd.read_csv('stocks/INTC.csv', delimiter=";")
jnj = pd.read_csv('stocks/JNJ.csv', delimiter=";")
jpm = pd.read_csv('stocks/JPM.csv', delimiter=";")
ko = pd.read_csv('stocks/KO.csv', delimiter=";")
mcd = pd.read_csv('stocks/MCD.csv', delimiter=";")
meta = pd.read_csv('stocks/META.csv', delimiter=";")
msft = pd.read_csv('stocks/MSFT.csv', delimiter=";")
nflx = pd.read_csv('stocks/NFLX.csv', delimiter=";")7
nvda = pd.read_csv('stocks/NVDA.csv', delimiter=";")
pfe = pd.read_csv('stocks/PFE.csv', delimiter=";")
pltr = pd.read_csv('stocks/PLTR.csv', delimiter=";")
t = pd.read_csv('stocks/T.csv', delimiter=";")
tsla = pd.read_csv('stocks/TSLA.csv', delimiter=";")
vz = pd.read_csv('stocks/VZ.csv', delimiter=";")
wmt = pd.read_csv('stocks/WMT.csv', delimiter=";")
xom = pd.read_csv('stocks/XOM.csv', delimiter=";")
```

In [ ]:
```
# list of stocks to allow for easier cleaning
stocks = [aapl, amd, amzn, ba, baba, bac, c, csco, cvx, dis, f, ge, googl, ibm, intc, jnj, jpm, ko, mcd, meta, r
          nvda, pfe, pltr, t, tsla, vz, wmt, xom]
tickers = ["AAPL", "AMD", "AMZN", "BA", "BABA", "BAC", "C", "CSCO", "CVX", "DIS", "F", "GE", "GOOGL", "IBM", "IN
           "JPM", "KO", "MCD", "META", "MSFT", "NFLX", "NVDA", "PFE", "PLTR", "T", "TSLA", "VZ", "WMT", "XOM"]
```

In [ ]:
```
for stock, ticker in zip(stocks, tickers):
    """
    Renames the 'Date' column of the stock DataFrame to 'Time' and adds a 'stock_ID' column.

    Args:
        stock (DataFrame): The stock data to be processed.
        ticker (str): The ticker symbol associated with the stock.

    Returns:
        None. The 'stock' DataFrame is modified in-place.

    Example:
        stocks = [stock1, stock2, stock3]
        tickers = ['AAPL', 'GOOGL', 'MSFT']
        for stock, ticker in zip(stocks, tickers):
            rename_and_add_columns(stock, ticker)

    """
    stock.rename(columns={'Date': 'Time'}, inplace=True)
    stock["stock_ID"] = ticker
```

In [ ]:
```
for stock in stocks:
    stock['DATETIME'] = pd.to_datetime(stock['Time'], format ='%Y-%m-%d %H:%M:%S')
```

In [ ]:
```
aapl.head()
```

In [ ]:
```
stocks_reduced = []
for i in range(len(stocks)):
    """
    Filters specific columns from the stock DataFrame and creates a reduced DataFrame.

    Args:
        stocks (list): A list of stock DataFrames.
```

```
    Returns:
        list: A list of reduced stock DataFrames.

    Example:
        stocks = [stock1, stock2, stock3]
        stocks_reduced = filter_columns(stocks)

    """
    stocks_reduced.append(stocks[i].filter(["stock_ID", "Volume", "Close", 'DATETIME']))
```

In [ ]:
```
for stock in stocks_reduced:
    stock["Volume"] = stock["Volume"].shift(1)
```

In [ ]:
```
stocks_reduced[0].head()
```

Clean Commodities and Bonds

In [ ]:
```
# The csv'S can be found in the next link : https://www.dropbox.com/scl/fo/n638k3vvnic2pss80zyvq/h?rlkey=3ilmw5f
us2y = pd.read_csv('commodities/2YTBond.csv', delimiter=";")
us5y= pd.read_csv('commodities/5YTBond.csv', delimiter=";")
us10y = pd.read_csv('commodities/10YTBond.csv', delimiter=";")
dowjones = pd.read_csv('commodities/DowJones.csv', delimiter=";")
nasdaq = pd.read_csv('commodities/NASDAQ.csv', delimiter=";")
sp500 = pd.read_csv('commodities/S&P.csv', delimiter=";")
gold = pd.read_csv('commodities/Gold.csv', delimiter=";")
oil = pd.read_csv('commodities/Oil.csv', delimiter=";")
```

In [ ]:
```
commodities = [us2y, us5y, us10y, dowjones, nasdaq, sp500, gold, oil]
```

In [ ]:
```
#rename column Date to Time
for commodity in commodities:
    commodity.rename(columns={'Date': 'Time'}, inplace=True)
```

In [ ]:
```
# change data type
for commodity in commodities:
    commodity['DATETIME'] = pd.to_datetime(commodity['Time'], format ='%Y-%m-%d %H:%M:%S')
```

In [ ]:
```
us2y.head()
```

In [ ]:
```
#move time period back by 15 minutes
for commodity in commodities:
    commodity['DATETIME'] = commodity['DATETIME'] - pd.Timedelta(minutes=15)
```

In [ ]:
```
us2y.head()
```

In [ ]:
```
Tickers = ["US2Y", "US5Y", "US10Y", 'DJ', "NQ", "SP", "Gold", "Oil"]
commodities_reduced = []
for i in range(len(commodities)):
    commodities_reduced.append(commodities[i].filter(['Close', 'DATETIME']).\
                    rename(columns = {'Close' : '{}_PP'.format(Tickers[i])}))
```

In [ ]:
```
#use reduce function to merge list of dataframes into singular dataframe on DATETIME_ADJUSTED
merge_commodities = reduce(lambda df1,df2: pd.merge(df1,df2,on='DATETIME'), commodities_reduced)
```

In [ ]:
```
merge_commodities
```

In [ ]:
```
merged_df = []
for stock in stocks_reduced:
    merged_df.append(pd.merge(stock, merge_commodities, on="DATETIME"))
```

In [ ]:
```
merged_df[0].head()
```

In [ ]:
```
df = reduce(lambda df1,df2: pd.concat([df1,df2]), merged_df)
```

In [ ]:
```
df.sort_values(by='DATETIME')
```

In [ ]:
```
df['Sector'] = df['stock_ID'].map(lambda x: 'Technology' if x in ("AAPL", "AMD", "CSCO", "IBM", "INTC", "MSFT",
                                    else "Consumer Cyclical" if x in ("AMZN", "BABA", "F", "MCD", "TSLA")
                                    else "Industrial" if x in ("BA", "GE")
                                    else "Financial Services" if x in ("BAC", "C", "JPM")
                                    else 'Energy' if x in ("CVX", 'XOM')
                                    else "Communication Services" if x in ("DIS", "GOOGL", "META", "NFLX", "T", "V
                                    else "Healthcare" if x in ("JNJ", "PFE")
                                    else "Consumer Defensive" if x in ("KO", "WMT")
                                    else '')
```

In [ ]:
```
df
```

In [ ]:
```
df['Sector'].value_counts()
```

In [ ]:
```
#create columns for each categorical datetime variable of interest
df['DATE']        = df.DATETIME.dt.date
df['MONTH']       = df.DATETIME.dt.month
df['DAY']         = df.DATETIME.dt.day
df['HOUR']        = df.DATETIME.dt.hour
df['MINUTE']      = df.DATETIME.dt.minute
df['WEEK_DAY']    = df.DATETIME.dt.dayofweek
```

In [ ]:
```
# Create holiday flag
holiday_days = []
for holiday in holidays.US(state = 'NY', years=[2020, 2021, 2022]).items(): holiday_days.append(str(holiday[0]))
df['HOLIDAY']=[1 if str(value) in holiday_days else 0 for value in df['DATE']]
```

In [ ]:
```
# Create pre - post holiday flag
df['POSTHOLIDAY_MORNING'], df['PREHOLIDAY_AFTERNON'] = 0, 0

id = df.loc[(df.HOLIDAY == 1),:].index
POSTHOLIDAY = pd.to_datetime((df.loc[id,'DATE'] + pd.DateOffset(days=1)).unique())
PREHOLIDAY  = pd.to_datetime((df.loc[id,'DATE'] - pd.DateOffset(days=1)).unique())

for i in range(len(PREHOLIDAY)) : df.loc[(df.DATE == PREHOLIDAY[i]) & (df.HOUR > 12)    , 'PREHOLIDAY_AFTERNON']
for i in range(len(POSTHOLIDAY)): df.loc[ (df.DATE == POSTHOLIDAY[i]) & (df.HOUR <= 12), 'POSTHOLIDAY_MORNING']
```

In [ ]:
```
# Create monday morning flag
df['MONDAY_MORNING']=0
df.loc[(df.WEEK_DAY == 0) & (df.HOUR <= 12), 'MONDAY_MORNING'] =1
```

In [ ]:
```
# Create friday afternoon flag
df['FRIDAY_AFTERNOON']=0
df.loc[(df.WEEK_DAY == 4) & (df.HOUR > 12), 'FRIDAY_AFTERNOON'] =1
```

In [ ]:
```
# Delete holiday days
df.drop(df.index[df['HOLIDAY']==1], inplace=True)
df.drop(columns=['HOLIDAY'], inplace=True)
```

Hot encoding

In [ ]:
```
def one_hot(og_df, feature_to_encode):
    "Function that takes a dataframe, and a feature to one-hot encode and returns the dataframe with that featur
    dummies = pd.get_dummies(og_df[feature_to_encode], prefix = feature_to_encode, prefix_sep = "_")
    df = pd.concat([og_df, dummies], axis=1)
    df = df.drop([feature_to_encode], axis=1)
    return(df)
```

In [ ]:
```
to_encode = ['Sector', 'MONTH', 'DAY']
```

In [ ]:
```
#encode the time features of interest
encoded_df = df
for encode in to_encode:
    encoded_df = one_hot(encoded_df, encode)
```

In [ ]:
```
encoded_df.rename(columns={"Volume":"Volume_PP"}, inplace=True)
```

In [ ]:
```
encoded_df
```

Final result

In [ ]:
```
encoded_df.to_csv('df.csv', index=False)
```