Clustering and Deep Learning in Portfolio Construction

Nitesh Kumar - Janice Pham

</br>

CS-82 Advanced Machine Learning - Final Project

Part II: Long Short-Term Memory Modeling

Set Up Notebook

Install Packages

In [ ]:
```
!pip install wordcloud nltk transformers spacy umap-learn hdbscan yfinance --quiet
```

```
     |████████████████████████████████| 5.8 MB 36.0 MB/s
     |████████████████████████████████| 88 kB 8.5 MB/s
     |████████████████████████████████| 5.2 MB 82.1 MB/s
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
    Preparing wheel metadata ... done
     |████████████████████████████████| 182 kB 73.0 MB/s
     |████████████████████████████████| 7.6 MB 72.8 MB/s
     |████████████████████████████████| 1.1 MB 67.6 MB/s
     |████████████████████████████████| 62 kB 128 kB/s
  Building wheel for umap-learn (setup.py) ... done
  Building wheel for pynndescent (setup.py) ... done
  Building wheel for hdbscan (PEP 517) ... done
```

Import Packages

In [ ]:
```
# System
import time
import os
import requests
from datetime import datetime
from collections import Counter
import itertools
import random as rn
import re
import string
from itertools import product
import yfinance as yf

# Libraries
import numpy as np
```

```
import pandas as pd
import urllib.request
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
%matplotlib inline

# Visualization
from sklearn.manifold import TSNE
from sklearn.manifold import MDS
from sklearn.manifold import Isomap
import umap

# Libraries
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error, mean_absolute_error, mean_absolute_percentage_error
from sklearn.cluster import KMeans, DBSCAN, MeanShift
from sklearn.neighbors import NearestNeighbors, NearestCentroid
from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.cluster import AgglomerativeClustering
import hdbscan
from sklearn.mixture import GaussianMixture as GMM


# Scipy
import scipy.cluster.hierarchy as hac
from scipy.spatial.distance import pdist
from scipy.cluster.hierarchy import fcluster
from scipy.cluster.hierarchy import dendrogram, linkage

import tensorflow as tf
from tensorflow.keras.models import Model, Sequential, save_model
from tensorflow.keras import layers
from tensorflow.keras import activations
from tensorflow.keras import losses
from tensorflow.keras import optimizers
from tensorflow.keras import initializers
from tensorflow.keras import regularizers

# NLP
import nltk
from nltk.corpus import stopwords
from transformers import pipeline
from transformers import BertTokenizer, TFBertForSequenceClassification
from transformers import InputExample, InputFeatures
from sklearn.feature_extraction.text import CountVectorizer
import spacy
```

The data collection in this notebook is similar to Part I Clustering notebook. However, we also added some data pre-processing particularly for LSTM model in this notebook

Data Collection & Processing

Fundamental Data

In [ ]:
```
################################################
### SET UP WORKING DIRECTORY & ACCESS TO DATA ###
################################################

# Get data from github to bypasss Google Drive Authentication
!git clone https://github.com/mlp2501/capstone_hes22.git

# Change working directory to data folder
os.chdir('capstone_hes22/cs82/final')
```

```
Cloning into 'capstone_hes22'...
remote: Enumerating objects: 44388, done.
remote: Counting objects: 100% (348/348), done.
remote: Compressing objects: 100% (343/343), done.
remote: Total 44388 (delta 31), reused 303 (delta 4), pack-reused 44040
Receiving objects: 100% (44388/44388), 292.02 MiB | 15.68 MiB/s, done.
Resolving deltas: 100% (871/871), done.
Checking out files: 100% (43524/43524), done.
```

In [ ]:
```
###------ List of all stocks collected in the dataset ------###
stock_list = []

for file in os.listdir("data/fundamental/"):
  stock_list.append(file.split('.')[0])

for stock in stock_list:
    if stock =='':
        stock_list.remove(stock)


###------ Function to combine all stock data across quarters 2002-2022 ------###
def add_key_and_combine_df(df,stock_list):
  """
  Combine all stocks and add key to each row i.e. 2022_3 representing the year and quarter
  """
  for stock in stock_list[1:]:
    temp = pd.read_excel(f"data/fundamental/{stock}.xlsx",index_col='date')
    temp.index = pd.date_range(start='03/31/2002', end='09/30/2022', periods =83)
    temp['stock'] = stock
    temp['key'] = [f"{temp.index[i].year}_{temp.index[i].quarter}" for i in range(temp.shape[0])]
    temp['date'] = temp.index.strftime("%Y-%m-%d")

    temp.drop('quarter', axis=1, inplace=True)

    df = pd.concat([df,temp], axis=0,ignore_index=True)

  return df
```

In [ ]:
```
###------ Dataframe contain all stock data across quarters ------###
```

```
stock = stock_list[0]
clustering_all_data = pd.read_excel(f"data/fundamental/{stock}.xlsx",index_col='date')
clustering_all_data.index = pd.date_range(start='03/31/2002', end='09/30/2022', periods =83)
clustering_all_data['stock'] = stock
clustering_all_data['key'] = [f"{clustering_all_data.index[i].year}_{clustering_all_data.index[i].quarter}" for i in range(clustering_all_data
clustering_all_data['date'] = clustering_all_data.index.strftime("%Y-%m-%d")
clustering_all_data.drop('quarter', axis=1,inplace=True)

clustering_all_data = add_key_and_combine_df(clustering_all_data.copy(),stock_list)

# print result
print(f"Number of stock tickers before cleaning: {len(clustering_all_data['stock'].unique())}")
clustering_all_data.head(3)
```

```
Number of stock tickers before cleaning: 283
```

Out[ ]:

	bal_sharesBasic	bal_acctPay	bal_totalLiabilities	bal_accoci	bal_equity	bal_retainedEarnings	bal_investmentsCurrent	bal_investmentsNonCurrent	bal_assetsNon
0	66477529.0	548200000.0	4.077200e+09	-516600000.0	6.139900e+09	2.051700e+09	0.0	0.0	7.2029
1	67004382.0	652800000.0	4.142800e+09	-332200000.0	6.276700e+09	1.996600e+09	0.0	0.0	7.3610
2	67542208.0	641300000.0	4.047300e+09	-175600000.0	6.383700e+09	1.927500e+09	0.0	0.0	7.4987

3 rows × 51 columns

We checked for NaN in the data. There were a few features that contained a lot missing values, so we excluded these features from the final dataset.

In [ ]:

```
# Check for NaN
clustering_all_data.isna().sum()
```

Out[ ]:

```
bal_sharesBasic                   3
bal_acctPay                       4
bal_totalLiabilities              4
bal_accoci                        4
bal_equity                        4
bal_retainedEarnings            891
bal_investmentsCurrent          315
bal_investmentsNonCurrent       315
bal_assetsNonCurrent            315
bal_deposits                      4
bal_debtCurrent                 317
bal_acctRec                       4
bal_debtNonCurrent              390
bal_ppeq                          4
bal_liabilitiesNonCurrent       394
bal_cashAndEq                     4
bal_liabilitiesCurrent          394
bal_taxLiabilities                4
bal_investments                   4
bal_intangibles                   4
bal_inventory                     4
bal_deferredRev                   4
```

```
bal_debt                        4
bal_taxAssets                   4
bal_totalAssets                 4
bal_assetsCurrent             315
inc_netIncComStock              0
inc_ebit                        0
inc_consolidatedIncome          0
inc_epsDil                      2
inc_ebitda                      1
inc_nonControllingInterests     0
inc_opex                        0
inc_eps                         1
inc_rnd                         0
inc_taxExp                      0
inc_prefDVDs                    0
inc_netIncDiscOps               0
inc_intexp                      0
inc_grossProfit                 0
inc_costRev                     0
inc_shareswaDil              3198
inc_netinc                      0
inc_ebt                         0
inc_opinc                       0
inc_sga                         0
inc_revenue                     0
inc_shareswa                    0
stock                           0
key                             0
date                            0
dtype: int64
```

In [ ]:

```
###---- Remove features with a lot of NaN ------###
missing_features = ['bal_investmentsNonCurrent', 'bal_investmentsCurrent', 'bal_assetsCurrent', \
                    'bal_liabilitiesNonCurrent', 'bal_debtNonCurrent', 'bal_debtCurrent', \
                    'bal_liabilitiesCurrent', 'bal_assetsNonCurrent', 'bal_retainedEarnings', \
                    'inc_shareswaDil'

]


for feature in missing_features:
  if feature in clustering_all_data.columns.tolist():
      clustering_all_data.drop(feature, axis=1, inplace=True)

# Check NaN again
clustering_all_data.isna().sum()
```

Out[ ]:

```
bal_sharesBasic               3
bal_acctPay                   4
bal_totalLiabilities          4
bal_accoci                    4
bal_equity                    4
bal_deposits                  4
bal_acctRec                   4
bal_ppeq                      4
bal_cashAndEq                 4
```

```
bal_taxLiabilities             4
bal_investments                4
bal_intangibles                4
bal_inventory                  4
bal_deferredRev                4
bal_debt                       4
bal_taxAssets                  4
bal_totalAssets                4
inc_netIncComStock             0
inc_ebit                       0
inc_consolidatedIncome         0
inc_epsDil                     2
inc_ebitda                     1
inc_nonControllingInterests    0
inc_opex                       0
inc_eps                        1
inc_rnd                        0
inc_taxExp                     0
inc_prefDVDs                   0
inc_netIncDiscOps              0
inc_intexp                     0
inc_grossProfit                0
inc_costRev                    0
inc_netinc                     0
inc_ebt                        0
inc_opinc                      0
inc_sga                        0
inc_revenue                    0
inc_shareswa                   0
stock                          0
key                            0
date                           0
dtype: int64
```

There were still a few missing values. We observed each of these missing values and their corresponding stock tickers.

In [ ]:

```
###------ Deaaling with stocks with small number of NaN ------###
# Find stocks with missing values
search_features = ['bal_ppeq', 'bal_deferredRev', 'bal_sharesBasic', 'bal_acctRec', \
                   'bal_equity', 'bal_acctPay', 'bal_totalAssets', 'bal_totalLiabilities', \
                   'bal_cashAndEq', 'bal_deposits', 'bal_deposits', 'bal_investments', \
                   'bal_accoci', 'bal_taxAssets', 'bal_debt', 'bal_inventory', \
                   'bal_taxLiabilities', 'bal_intangibles', 'inc_ebitda', 'inc_epsDil', \
                   'inc_eps',

    ]

temp1 = clustering_all_data.iloc[:,10:][clustering_all_data['bal_totalAssets'].isna()]
temp2 = clustering_all_data.iloc[:,10:][clustering_all_data['bal_sharesBasic'].isna()]
temp3 = clustering_all_data.iloc[:,10:][clustering_all_data['inc_ebitda'].isna()]
temp4 = clustering_all_data.iloc[:,10:][clustering_all_data['inc_epsDil'].isna()]
print('='*50 + '\n1st List of NaN Stocks\n' + '='*50)
display(temp1)
print('='*50 + '\n2nd List of NaN Stocks\n' + '='*50)
display(temp2)
print('='*50 + '\n3rd List of NaN Stocks\n' + '='*50)
```

```
display(temp3)
print('='*50 + '\n4th List of NaN Stocks\n' + '='*50)
display(temp4)
```

```
==================================================
1st List of NaN Stocks
==================================================
```

	bal_investments	bal_intangibles	bal_inventory	bal_deferredRev	bal_debt	bal_taxAssets	bal_totalAssets	inc_netIncComStock	inc_ebit	inc_consolidatedIncome
7767	NaN	NaN	NaN	NaN	NaN	NaN	NaN	3339000	8330000	3339000
7768	NaN	NaN	NaN	NaN	NaN	NaN	NaN	26314000	34343000	26314000
7769	NaN	NaN	NaN	NaN	NaN	NaN	NaN	37920000	43032000	37920000
15182	NaN	NaN	NaN	NaN	NaN	NaN	NaN	106000000	192000000	106000000

4 rows × 31 columns

```
==================================================
2nd List of NaN Stocks
==================================================
```

	bal_investments	bal_intangibles	bal_inventory	bal_deferredRev	bal_debt	bal_taxAssets	bal_totalAssets	inc_netIncComStock	inc_ebit	inc_consolidatedInco
4064	105000000.0	217000000.0	935000000.0	86000000.0	579000000.0	115000000.0	9.046000e+09	127000000	194000000	132000(
4065	104000000.0	220000000.0	900000000.0	87000000.0	575000000.0	113000000.0	8.885000e+09	141000000	258000000	159000(
4066	87000000.0	258000000.0	859000000.0	0.0	655000000.0	116000000.0	8.231000e+09	144000000	246000000	153000(

3 rows × 31 columns

```
==================================================
3rd List of NaN Stocks
==================================================
```

	bal_investments	bal_intangibles	bal_inventory	bal_deferredRev	bal_debt	bal_taxAssets	bal_totalAssets	inc_netIncComStock	inc_ebit	inc_consolidatedIncome
7769	NaN	NaN	NaN	NaN	NaN	NaN	NaN	37920000	43032000	37920000

1 rows × 31 columns

```
==================================================
4th List of NaN Stocks
==================================================
```

	bal_investments	bal_intangibles	bal_inventory	bal_deferredRev	bal_debt	bal_taxAssets	bal_totalAssets	inc_netIncComStock	inc_ebit	inc_consolidatedI
21242	3.481900e+08	1.806453e+09	367358000.0	9.563800e+07	1.066192e+10	102968000.0	1.988948e+10	63223000	336555000	63
21248	7.857000e+09	1.443830e+11	0.0	2.757000e+09	9.762300e+10	0.0	2.543080e+11	-4733000000	-2759000000	-4665(

2 rows × 31 columns

Based on the above tables, these stock contained many missing values across the features. Therefore we remove them from the list instead of imputation every single entry.

In [ ]:
```
###--- Stocks to remove from dataset ------###
stocks_with_missing_features =  ['TTWO','LUV','MGA','CNP', 'CMCSA']
for stock in stocks_with_missing_features:
  clustering_all_data = clustering_all_data[clustering_all_data['stock'] != stock]
  stock_list.remove(stock)
```

Lastly, we checked the missing values again and confirmed there was no more missing values.

In [ ]:
```
# Final check for missing value
clustering_all_data.isna().sum()
```

Out[ ]:
```
bal_sharesBasic                0
bal_acctPay                    0
bal_totalLiabilities           0
bal_accoci                     0
bal_equity                     0
bal_deposits                   0
bal_acctRec                    0
bal_ppeq                       0
bal_cashAndEq                  0
bal_taxLiabilities             0
bal_investments                0
bal_intangibles                0
bal_inventory                  0
bal_deferredRev                0
bal_debt                       0
bal_taxAssets                  0
bal_totalAssets                0
inc_netIncComStock             0
inc_ebit                       0
inc_consolidatedIncome         0
inc_epsDil                     0
inc_ebitda                     0
inc_nonControllingInterests    0
inc_opex                       0
inc_eps                        0
inc_rnd                        0
inc_taxExp                     0
inc_prefDVDs                   0
inc_netIncDiscOps              0
```

```
inc_intexp                  0
inc_grossProfit             0
inc_costRev                 0
inc_netinc                  0
inc_ebt                     0
inc_opinc                   0
inc_sga                     0
inc_revenue                 0
inc_shareswa                0
stock                       0
key                         0
date                        0
dtype: int64
```

Since there were features that mostly overlap with one another based on reporting categories, we sorted and selected the features that we thought would be helpful according to our domain knowledge.

In [ ]:

```
###------ Remove overlapped features ------###
dropped_features = [
    'bal_ppeq', 'bal_deferredRev', 'bal_sharesBasic', 'bal_acctRec', 'bal_acctPay', 'bal_cashAndEq', \
    'bal_deposits', 'bal_accoci', 'bal_taxAssets', 'bal_taxLiabilities', 'bal_intangibles', 'inc_shareswa', \
    'inc_rnd', 'inc_sga', 'inc_epsDil', 'inc_netIncDiscOps', 'inc_ebt',
    'inc_nonControllingInterests', 'inc_taxExp', 'inc_prefDVDs', 'inc_intexp'
]

shortlist_modified_clustering_agg = clustering_all_data.copy()

for feature in dropped_features:
  if feature in shortlist_modified_clustering_agg.columns.tolist():
      shortlist_modified_clustering_agg.drop(feature, axis=1, inplace=True)

print(f'Number of features in modified cluster aggregate data: {shortlist_modified_clustering_agg.shape[1]}')
```

```
Number of features in modified cluster aggregate data: 20
```

The next step was to load the daily and quarterly stock prices and returns.

In [ ]:

```
                            ##################################
                            ### DAILY STOCK DATA PROCESSING ###
                            ##################################

###------ Function to combine all daily stock prices data ------###
def stock_price_df(df, stock_price_list):
  for stock in stock_price_list[1:]:
    temp = pd.read_excel(f"data/daily/{stock}.xlsx",index_col='date').iloc[:,0]
    df = pd.concat([df,temp], axis=1,ignore_index=True)
  return df

###------ Stock prices data frame ------###
stock_price_first_i = stock_list[0]
stock_daily_df = pd.read_excel(f"data/daily/{stock_price_first_i}.xlsx", index_col='date').iloc[:,0]
stock_daily_df = stock_price_df(stock_daily_df.copy(), stock_list)
stock_daily_df.columns = stock_list
```

```
# Display result
stock_daily_df.head(3)
```

Out[ ]:

date	RRX	AES	LRCX	PNR	WAB	HOLX	TTC	RL	ATR	AME	...	GRMN	DRI	CVS	AMD	SO	
2002-01-02	15.473745	11.011985	21.313821	8.358264	5.739346	2.450	2.254840	20.814473	12.938435	4.072210	...	5.526697	12.576474	10.696285	16.39	9.677001	15.72
2002-01-03	16.178699	10.821784	22.979101	8.268439	5.715996	2.500	2.242098	21.213065	12.901415	4.131678	...	5.537100	12.957689	10.696285	19.37	9.540384	15.95
2002-01-04	16.672167	11.444856	23.296298	8.107216	5.720666	2.675	2.259251	22.647996	12.856991	4.050233	...	5.555306	13.720117	10.707126	20.00	9.415153	15.13

3 rows × 278 columns

In [ ]:

```
###------ Gettting quarterly stock returns ------###
stock_quarterly_returns = pd.read_excel('stock_quarterly_values.xlsx')
stock_quarterly_returns.index = stock_quarterly_returns['quarter']
#stock_quarterly_returns.index = pd.to_datetime(stock_quarterly_returns.index)
stock_quarterly_returns = stock_quarterly_returns.drop(columns=['quarter'])
print(f'Shape of data frame: {stock_quarterly_returns.shape}')
stock_quarterly_returns.head()
```

```
Shape of data frame: (83, 278)
```

Out[ ]:

quarter	NOV	JBL	BIIB	STZ	EXPD	FIS	MSI	DVN	ALV	PPL	...	LRCX	TAP	RMD	COO	
2002_1	0.108288	-0.000185	-0.025255	0.109111	0.029684	0.052248	-0.037931	0.102078	0.086801	0.060331	...	0.083528	0.106691	-0.122034	-0.012552	
2002_2	-0.092556	-0.054273	-0.266956	0.067796	0.040049	-0.041063	0.002300	0.000373	0.016417	-0.072402	...	-0.220446	-0.031041	-0.132734	-0.000144	-
2002_3	-0.033830	-0.122138	0.105040	-0.127750	-0.075957	-0.263006	-0.146182	-0.008808	-0.063070	0.009341	...	-0.277468	-0.027970	0.011541	0.058347	
2002_4	0.038875	0.070638	-0.098974	0.004049	0.058614	0.067419	-0.081526	-0.025264	-0.020854	0.027749	...	0.081601	0.027824	0.017687	-0.026577	-
2003_1	-0.003862	-0.033653	-0.001520	-0.035439	0.029877	-0.001205	-0.044652	0.006988	-0.031078	0.009255	...	-0.009802	-0.103902	0.012817	0.066076	-

5 rows × 278 columns

In [ ]:

```
###------ Gettting daily stock returns ------###
stock_daily_returns = pd.read_csv('stock_daily_rets.csv')
stock_daily_returns.index = stock_daily_returns['date']
stock_daily_returns.index = pd.to_datetime(stock_daily_returns.index)
stock_daily_returns = stock_daily_returns.drop(columns=['date'])
print(f'Shape of data frame: {stock_daily_returns.shape}')
stock_daily_returns.head()
```

```
Shape of data frame: (5266, 278)
```

Out[ ]:

date	NOV	JBL	BIIB	STZ	EXPD	FIS	MSI	DVN	ALV	PPL	...	LRCX	TAP	RMD	COO

date	NOV	JBL	BIIB	STZ	EXPD	FIS	MSI	DVN	ALV	PPL	...	LRCX	TAP	RMD	COO	
date																
2002-01-03	-0.015452	0.021944	-0.027693	0.003843	0.001370	0.001724	0.009946	-0.009076	0.014135	0.000997	...	0.032672	-0.005862	-0.007833	-0.013011	-0.(
2002-01-04	0.004536	0.004537	0.005601	0.008969	0.019830	-0.004199	-0.009667	0.008051	0.009793	-0.033085	...	0.005954	-0.001165	-0.004432	-0.008334	-0.
2002-01-07	0.000676	0.005176	-0.016816	0.018537	-0.004451	-0.013484	-0.010745	0.011360	0.000412	0.007192	...	-0.004956	-0.000333	-0.013661	0.011351	-0.
2002-01-08	-0.011411	-0.007264	0.013383	0.012759	0.003943	-0.002439	-0.010724	-0.004127	-0.008522	-0.001986	...	0.000498	-0.000501	0.010116	0.003357	-0.
2002-01-09	-0.005352	0.014747	-0.007155	0.003929	-0.003576	0.009422	-0.010093	0.002347	-0.001682	0.005799	...	0.000332	0.004403	0.001269	0.005836	-0.(

5 rows × 278 columns

We also retrieved S&P 500 daily prices using Yahoo Finance API and computed daily returns of the S&P 500. This would be used later as the benchmark for our model comparison.

In [ ]:

```
                                ###################################
                                ### DAILY S&P500 DATA PROCESSING ###
                                ###################################

###------ Get data for S&P500 ------###
sp_start = datetime(2002, 1, 1)
sp_end = datetime(2022, 9, 30)

sp500_ticker = yf.Ticker('^GSPC')

sp500_daily_prices = pd.DataFrame(sp500_ticker.history(start=sp_start, \
                                        end=sp_end)['Close'])
print('='*40 + '\nS&P 500 Daily Prices\n' + '='*40)
display(sp500_daily_prices.head())


###------ Compute S&P 500 Returns ------###
sp500_daily_rets = pd.DataFrame(np.log(sp500_daily_prices.Close) - np.log(sp500_daily_prices.Close.shift(1)))
sp500_daily_rets.columns = ['S&P500']
sp500_daily_rets = sp500_daily_rets.iloc[1:,:]
sp500_daily_rets.head(3)
```

```
========================================
S&P 500 Daily Prices
========================================
```

Date	Close
2002-01-02 00:00:00-05:00	1154.670044

	Close
Date	
2002-01-03 00:00:00-05:00	1165.270020
2002-01-04 00:00:00-05:00	1172.510010
2002-01-07 00:00:00-05:00	1164.890015
2002-01-08 00:00:00-05:00	1160.709961

Out[ ]:

	S&P500
Date	
2002-01-03 00:00:00-05:00	0.009138
2002-01-04 00:00:00-05:00	0.006194
2002-01-07 00:00:00-05:00	-0.006520

In [ ]:

```
###------ Create dict to split data by quarters ------###
by_quarter_grouping = shortlist_modified_clustering_agg.groupby(shortlist_modified_clustering_agg['key'])

by_quarter_dict = dict(list(by_quarter_grouping))

# Print result
print(f'Length of dict: {len(by_quarter_dict)}')
print(f"Shape of quarterly data frame: {by_quarter_dict['2002_1'].shape}")
display(by_quarter_dict['2002_1'].head(3))
```

```
Length of dict: 83
Shape of quarterly data frame: (278, 20)
```

	bal_totalLiabilities	bal_equity	bal_investments	bal_inventory	bal_debt	bal_totalAssets	inc_netIncComStock	inc_ebit	inc_consolidatedIncome	inc_eb
0	4.077200e+09	6.139900e+09	0.000000e+00	1.361500e+09	2.315400e+09	1.024900e+10	119800000	174400000	121900000	2.522000e
83	3.231400e+10	3.409000e+09	1.792000e+09	9.980000e+08	2.220500e+10	3.777500e+10	421000000	842000000	446000000	1.108000e
166	1.130872e+10	7.412567e+09	1.205510e+08	4.360325e+09	5.003473e+09	1.872128e+10	1425879000	1654745000	1425879000	1.730496e

In [ ]:

```
###------ Data grouped by quarter ------###
by_quarter_dict_clean = {}
for i in by_quarter_dict.keys():
  by_quarter_dict_clean[i] = by_quarter_dict[i].drop(columns=['stock', 'key', 'date'])
```

In [ ]:

```
###------ Group daily S&P500 returns by quarter ------###
sp500_daily_rets_by_quarter = {}
for k, v in by_quarter_dict_clean.items():
  daily_ls = []
```

```
  if k[-1] == '1':
    for i in sp500_daily_rets.T.columns:
      if (str(i).startswith(k[:4]+'-01')==True) or \
        (str(i).startswith(k[:4]+'-02')==True) or \
        (str(i).startswith(k[:4]+'-03')==True):
        col = sp500_daily_rets.T[i]
        daily_ls.append(col)
        daily_df = pd.DataFrame(daily_ls)
      else:
        pass
    sp500_daily_rets_by_quarter[k] = daily_df

  elif k[-1] == '2':
    for i in sp500_daily_rets.T.columns:
      if (str(i).startswith(k[:4]+'-04')==True) or \
         (str(i).startswith(k[:4]+'-05')==True) or \
         (str(i).startswith(k[:4]+'-06')==True):
        col = sp500_daily_rets.T[i]
        daily_ls.append(col)
        daily_df = pd.DataFrame(daily_ls)
      else:
        pass
    sp500_daily_rets_by_quarter[k] = daily_df

  elif k[-1] == '3':
    for i in sp500_daily_rets.T.columns:
      if (str(i).startswith(k[:4]+'-07')==True) or \
         (str(i).startswith(k[:4]+'-08')==True) or \
         (str(i).startswith(k[:4]+'-09')==True):
        col = sp500_daily_rets.T[i]
        daily_ls.append(col)
        daily_df = pd.DataFrame(daily_ls)
      else:
        pass
    sp500_daily_rets_by_quarter[k] = daily_df

  elif k[-1] == '4':
    for i in sp500_daily_rets.T.columns:
      if (str(i).startswith(k[:4]+'-10')==True) or \
         (str(i).startswith(k[:4]+'-11')==True) or \
         (str(i).startswith(k[:4]+'-12')==True):
        col = sp500_daily_rets.T[i]
        daily_ls.append(col)
        daily_df = pd.DataFrame(daily_ls)
      else:
        pass
    sp500_daily_rets_by_quarter[k] = daily_df

  else:
    pass
```

In [ ]:

```
###------ Display result ------###
print('='*50 + '\nDaily S&P 500 returns grouped by quarter - 2022Q3\n' + '='*50)
print('\nFirst 3 days\n')
display(sp500_daily_rets_by_quarter['2022_3'].head(3))
```

```
print('\nLast 3 days\n')
display(sp500_daily_rets_by_quarter['2022_3'].tail(3))
```

```
================================================
Daily S&P 500 returns grouped by quarter - 2022Q3
================================================

First 3 days
```

	S&P500
2022-07-01 00:00:00-04:00	0.010499
2022-07-05 00:00:00-04:00	0.001583
2022-07-06 00:00:00-04:00	0.003567

```
Last 3 days
```

	S&P500
2022-09-27 00:00:00-04:00	-0.002123
2022-09-28 00:00:00-04:00	0.019481
2022-09-29 00:00:00-04:00	-0.021353

Long Short-term Memory (LSTM)

The first step before training the model was data processing. We performed splitting, scaling, and reshaping to match the input data structure to that of the LSTM model architect.

Data Processing

In [ ]:

```
###------ Split aggregate data ------###
groupby_shortlist = list(shortlist_modified_clustering_agg.groupby(shortlist_modified_clustering_agg['date']<='2018-01-01'))
train_data_agg = groupby_shortlist[1][1]
test_data_agg = groupby_shortlist[0][1]

###------ Scale data ------###
lstm_scaler = StandardScaler()
scaled_train_agg = lstm_scaler.fit_transform(train_data_agg.iloc[:,:-3])
scaled_train_agg_df = pd.DataFrame(scaled_train_agg)
scaled_train_agg_df.columns = train_data_agg.columns[:-3]
scaled_train_agg_df = pd.concat([scaled_train_agg_df, train_data_agg.iloc[:,-3:].reset_index(drop=True)], axis=1)

scaled_test_agg = lstm_scaler.transform(test_data_agg.iloc[:,:-3])
scaled_test_agg_df = pd.DataFrame(scaled_test_agg)
scaled_test_agg_df.columns = test_data_agg.columns[:-3]
scaled_test_agg_df = pd.concat([scaled_test_agg_df, test_data_agg.iloc[:,-3:].reset_index(drop=True)], axis=1)

print('='*40 + '\nTrain data\n' + '='*40)
```

```
display(scaled_train_agg_df.head(3))
print('='*40 + '\nTest data\n' + '='*40)
display(scaled_test_agg_df.head(3))
```

```
========================================
Train data
========================================
```

	bal_totalLiabilities	bal_equity	bal_investments	bal_inventory	bal_debt	bal_totalAssets	inc_netIncComStock	inc_ebit	inc_consolidatedIncome	inc_ebitda	inc_ope
0	-0.337292	-0.034478	-0.185263	0.014317	-0.285800	-0.264079	-0.162136	-0.224020	-0.166862	-0.263867	-0.30253
1	-0.334924	-0.022877	-0.185263	0.021658	-0.288104	-0.258536	-0.133865	-0.201989	-0.140222	-0.238782	-0.29949
2	-0.338372	-0.013804	-0.185263	0.007604	-0.294896	-0.258202	-0.154750	-0.227692	-0.160358	-0.258987	-0.29100

```
========================================
Test data
========================================
```

	bal_totalLiabilities	bal_equity	bal_investments	bal_inventory	bal_debt	bal_totalAssets	inc_netIncComStock	inc_ebit	inc_consolidatedIncome	inc_ebitda	inc_opex
0	-0.458004	-0.492927	-0.185263	-0.291348	-0.413372	-0.496466	-0.276826	-0.348452	-0.282127	-0.409339	-0.418974
1	-0.458651	-0.495291	-0.185263	-0.293920	-0.414554	-0.497678	-0.272281	-0.343139	-0.277663	-0.405595	-0.421421
2	-0.458426	-0.498381	-0.185263	-0.293977	-0.412736	-0.498477	-0.284406	-0.358117	-0.289571	-0.417246	-0.419202

In [ ]:

```
                        ####################################
                        ### DATA PROCESSING FOR LSTM MODEL ###
                        ####################################


###------ Grouped train/test data ------###
grouped_train = dict(list(scaled_train_agg_df.groupby(scaled_train_agg_df['stock'])))
grouped_test = dict(list(scaled_test_agg_df.groupby(scaled_test_agg_df['stock'])))

lstm_grouped_train = {}
for k, v in grouped_train.items():
  lstm_grouped_train[k] = v.iloc[:,:]

lstm_grouped_test = {}
for k, v in grouped_test.items():
  lstm_grouped_test[k] = v.iloc[:-1,:]


###------ Reshape x values for LSTM ------###
rs_train = {}
for k, v in lstm_grouped_train.items():
  rs_train[k] = np.array(v.iloc[:,:-3]).reshape(v.shape[0], 1, v.shape[1]-3)
```

```
rs_test = {}
for k, v in lstm_grouped_test.items():
  rs_test[k] = np.array(v.iloc[:,:-3]).reshape(v.shape[0], 1, v.shape[1]-3)

###------ Organize aggregate stock quarterly returns data to dict of stocks rets ------###
shift_stock_quarterly_returns = stock_quarterly_returns.iloc[1:, :]
stock_quarterly_rets_dict = {}
for i in shift_stock_quarterly_returns.columns:
  stock_quarterly_rets_dict[i] = shift_stock_quarterly_returns[i]

###------ Split and reshape y values for LSTM ------###
rs_y_train_dict = {}
rs_y_test_dict = {}
for k, v in stock_quarterly_rets_dict.items():
  rs_y_train_dict[k] = np.array(v[:64]).reshape(64,1,1)

for k, v in stock_quarterly_rets_dict.items():
  rs_y_test_dict[k] = np.array(v[64:]).reshape(18,1,1)
```

Modeling

The LSTM model architecture includes 2 layers of LSTM (with dropout layer in between) followed by TimeDistributed layer wrapped around output dense layer. We did not go deeper because of the limited dataset (82 quarters for each stock across 17 features) and we would risk overfitting if we add more layers into the architecture.

In [ ]:

```
                        ##############################
                        ### LSTM MODEL ARCHITECTURE ###
                        ##############################

tf.random.set_seed(2022)
loss_fn = losses.MeanSquaredError()
lr = 0.01
lstm_node = 64
dropout_rate = 0.3
metrics = 'mse'
epochs = 50
batch_size = 32

def lstm_model(train_x, test_x, train_y, test_y):
  early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, verbose=1, restore_best_weights=True)

  lstm_model = tf.keras.models.Sequential(
        [
            layers.InputLayer(input_shape=[train_x.shape[1], \
                                           train_x.shape[2]]),
            layers.LSTM(lstm_node, return_sequences=True),
            layers.Dropout(dropout_rate),
            layers.LSTM(int(lstm_node/2), return_sequences=True),
            layers.Dropout(dropout_rate),
            layers.TimeDistributed(layers.Dense(activation='linear', units=train_y.shape[2]))

        ]
    )
```

```
    # Compile model
  lstm_model.compile(
      optimizer = optimizers.Adam(learning_rate=lr),
      loss = loss_fn,
      metrics = metrics
  )


  model_hist = lstm_model.fit(
      train_x,
      train_y,
      epochs=epochs,
      batch_size = batch_size,
      validation_data = (test_x, test_y),
      callbacks = early_stopping,
      verbose = 0,
      shuffle = False
  )

  return lstm_model, model_hist
```

We performed LSTM model separately on each of the stock to obtain the predicted returns for 278 stock across all the quarters. Based on the quarterly returns of these stocks, we ranked the the stock and selected the top N stocks to buy and the bottom N stocks to sell. These were the long-short portfolio and would be used to compare with the benchmark S&P 500.

In [ ]:

```
                    ############################################################
                    ### LSTM ACROSS ALL STOCKS TO PREDICT QUARTERLY RETURNS ###
                    ############################################################

tf.random.set_seed(2022)
lstm_all_stocks_model = {}
for k in rs_train.keys():
  lstm_all_stocks_model[k] = lstm_model(rs_train[k], rs_test[k], \
                                        rs_y_train_dict[k], rs_y_test_dict[k])
```

Modeling Result

Based on the result of each model, we predicted the returns in train and test period to get the stock returns for the entire period.

In [ ]:

```
###------ Model predictions for all stocks ------###
train_preds_dict = {}
test_preds_dict = {}
for k, v in lstm_all_stocks_model.items():
  train_preds_dict[k] = pd.DataFrame(v[0].predict(rs_train[k]).reshape(-1,1), columns=[k])
  test_preds_dict[k] = pd.DataFrame(v[0].predict(rs_test[k]).reshape(-1,1), columns=[k])
```

In [ ]:

```
###------ Concatenate all predictions across all stocks ------###
all_preds_dict = {}
for (k, v_train, v_test) in zip(train_preds_dict.keys(), train_preds_dict.values(), test_preds_dict.values()):
  all_preds_dict[k] = pd.concat([v_train, v_test], axis=0)
```

```
  all_preds_dict[k].index = shift_stock_quarterly_returns.index

all_preds_df = pd.DataFrame()
for k, v in all_preds_dict.items():
  all_preds_df = pd.concat([all_preds_df, v], axis=1)

print('='*100 + '\nLSTM Model Predictions Across All Stocks\n' + '='*100)
all_preds_df.head()
```

```
====================================================================================================
LSTM Model Predictions Across All Stocks
====================================================================================================
```

Out[ ]:

quarter	A	AAP	ABC	ABMD	ADBE	ADP	AEE	AEP	AES	AGCO	...	WM	WMB	WOLF	WSO	WST
2002_2	0.008466	0.035112	0.180928	-0.029335	-0.010527	0.049566	-0.001419	-0.011185	0.013202	0.003040	...	0.010214	-0.025973	0.022563	0.009848	0.019952
2002_3	0.008640	0.035753	0.179609	-0.014503	-0.009112	0.049884	0.007447	-0.009008	0.021588	0.007858	...	0.011231	-0.019925	0.022101	0.007264	0.018269
2002_4	0.008203	0.039974	0.172275	-0.016013	-0.006689	0.068764	0.006214	-0.009749	0.015888	0.009143	...	0.012550	-0.018016	0.022799	0.011522	0.018936
2003_1	0.008282	0.032382	0.176952	-0.011989	-0.006768	0.052047	0.011534	-0.010324	0.020667	-0.002589	...	0.012531	-0.024705	0.022908	0.014037	0.019508
2003_2	0.008288	0.034711	0.176454	-0.015157	-0.005065	0.051405	-0.000371	-0.008473	0.012185	0.005376	...	0.012377	-0.014624	0.022952	0.009749	0.018903

5 rows × 278 columns

The position table was built upon the rankings of the stock returns per quarter. We selected the top N stocks to long and the bottom N stocks to short. In this case, we used N = 30, 50, and 100.

In [ ]:

```
###------ Function to create position table ------###

def pos_table(all_preds_df, n):
  pos_table = {}
  for i in range(len(all_preds_df)):
    df_ls = []
    for j in all_preds_df.columns:
      bottom_ = sorted(all_preds_df.iloc[i])[:n]
      top_ = sorted(all_preds_df.iloc[i])[-n:]
      if all_preds_df[j][i] in top_:
        df_ls.append(1)
      elif all_preds_df[j][i] in bottom_:
        df_ls.append(-1)
      else:
        df_ls.append(0)

    pos_table[i] = df_ls
    # Convert position dict to data frame
  pos_df = pd.DataFrame(pos_table).T
  pos_df.columns = all_preds_df.columns
  pos_df.index = stock_quarterly_returns.index[:-1] # Position at the end of the month 01/31/04

  return pos_df
```

In [ ]:
```
###------ Long-Short Portfolio of Top/Bottom 30 Stocks ------###
long_short30_pos = pos_table(all_preds_df, 30)
long_short30_pos.head(3)
```

Out[ ]:

quarter	A	AAP	ABC	ABMD	ADBE	ADP	AEE	AEP	AES	AGCO	...	WM	WMB	WOLF	WSO	WST	WTRG	WY	XEL	ZBH	ZBRA
2002_1	0	0	1	-1	0	1	0	0	0	0	...	0	-1	0	0	0	0	0	-1	0	0
2002_2	0	0	1	0	0	1	0	0	0	0	...	0	0	0	0	0	0	0	-1	0	0
2002_3	0	0	1	-1	0	1	0	0	0	0	...	0	-1	0	0	0	0	0	-1	0	1

3 rows × 278 columns

In [ ]:
```
###------ Long-Short Portfolio of Top/Bottom 50 Stocks ------###
long_short50_pos = pos_table(all_preds_df, 50)
long_short50_pos.head(3)
```

Out[ ]:

quarter	A	AAP	ABC	ABMD	ADBE	ADP	AEE	AEP	AES	AGCO	...	WM	WMB	WOLF	WSO	WST	WTRG	WY	XEL	ZBH	ZBRA
2002_1	0	1	1	-1	0	1	0	0	0	0	...	0	-1	0	0	0	0	0	-1	0	1
2002_2	0	1	1	-1	0	1	0	0	0	0	...	0	-1	0	0	0	0	0	-1	0	0
2002_3	0	1	1	-1	0	1	0	-1	0	0	...	0	-1	0	0	0	0	0	-1	0	1

3 rows × 278 columns

In [ ]:
```
###------ Long-Short Portfolio of Top/Bottom 100 Stocks ------###
long_short100_pos = pos_table(all_preds_df, 100)
long_short100_pos.head(3)
```

Out[ ]:

quarter	A	AAP	ABC	ABMD	ADBE	ADP	AEE	AEP	AES	AGCO	...	WM	WMB	WOLF	WSO	WST	WTRG	WY	XEL	ZBH	ZBRA
2002_1	0	1	1	-1	-1	1	-1	-1	0	0	...	0	-1	1	0	1	0	1	-1	1	1
2002_2	0	1	1	-1	-1	1	0	-1	1	0	...	0	-1	1	0	1	0	1	-1	1	-1
2002_3	0	1	1	-1	-1	1	0	-1	0	0	...	0	-1	1	0	1	0	1	-1	1	1

3 rows × 278 columns

The Sharpe ratio was computed to evaluate and compare the performances among the long-short portfolios and with the benchmark S&P 500.

In [ ]:
```
###------ Function to compute long-short portfolio Sharpe ratio------###
def annualized_sharpe(pos_tbl):
```

```
  whole_rets = (all_preds_df.reset_index(drop=True)*pos_tbl.reset_index(drop=True)).sum(axis=1)
  whole_yr_rets = (whole_rets.sum()/len(whole_rets))*4
  whole_vol = (whole_rets.std())*(4**0.5)
  whole_annualized_sharpe = whole_yr_rets/whole_vol

  train_rets = whole_rets[:64]
  train_yr_rets = (train_rets.sum()/len(train_rets))*4
  train_vol = (train_rets.std())*(4**0.5)
  train_annualized_sharpe = train_yr_rets/train_vol

  test_rets = whole_rets[64:]
  test_yr_rets = (test_rets.sum()/len(test_rets))*4
  test_vol = (test_rets.std())*(4**0.5)
  test_annualized_sharpe = test_yr_rets/test_vol


  return whole_annualized_sharpe, train_annualized_sharpe, test_annualized_sharpe
```

In [ ]:

```
###------ Annualized S&P 500 Performance ------###
# Start 2002 Q2 at iloc 58 end 2018 Q1 at iloc 4087
train_period_sp500 = sp500_daily_rets[58:4088]
test_period_sp500 = sp500_daily_rets[4088:]

annualized_sp500_rets_whole = (np.product(1 + sp500_daily_rets[58:]))-1
annualized_sp500_vol_whole = (sp500_daily_rets[58:].std())*(252**0.5)
annualized_sp500_sharpe_whole = annualized_sp500_rets_whole/annualized_sp500_vol_whole

annualized_sp500_rets_train = (np.product(1 + train_period_sp500))-1
annualized_sp500_vol_train = (train_period_sp500.std())*(252**0.5)
annualized_sp500_sharpe_train = annualized_sp500_rets_train/annualized_sp500_vol_train

annualized_sp500_rets_test = (np.product(1 + test_period_sp500))-1
annualized_sp500_vol_test = (test_period_sp500.std())*(252**0.5)
annualized_sp500_sharpe_test = annualized_sp500_rets_test/annualized_sp500_vol_test
```

In [ ]:

```
###------ Long-Short 30 Stocks Portfolio ------###
portfolio_30_sharpe = annualized_sharpe(long_short30_pos)

###------ Long-Short 50 Stocks Portfolio ------###
portfolio_50_sharpe = annualized_sharpe(long_short50_pos)

###------ Long-Short 100 Stocks Portfolio ------###
portfolio_100_sharpe = annualized_sharpe(long_short100_pos)

lstm_portfolio_df = pd.DataFrame({
  'Whole Period': [portfolio_30_sharpe[0], portfolio_50_sharpe[0], portfolio_100_sharpe[0], annualized_sp500_sharpe_whole[0]],
  'Train Period': [portfolio_30_sharpe[1], portfolio_50_sharpe[1], portfolio_100_sharpe[1], annualized_sp500_sharpe_train[0]],
  'Test Period': [portfolio_30_sharpe[2], portfolio_50_sharpe[2], portfolio_100_sharpe[2], annualized_sp500_sharpe_test[0]],

})
lstm_portfolio_df.index=['Long-Short 30', 'Long-Short 50', 'Long-Short 100', 'S&P 500']
print('='*80 + '\nLSTM Long-Short Portfolio Performance\n' + '='*80)
lstm_portfolio_df
```

```
================================================================================
LSTM Long-Short Portfolio Performance
================================================================================
```

Out[ ]:

	Whole Period	Train Period	Test Period
Long-Short 30	5.237905	5.721372	29.379903
Long-Short 50	5.699035	6.204037	36.285026
Long-Short 100	6.276307	6.812937	42.562016
S&P 500	5.793181	3.846231	1.078904

The above result indicated that the long-short portfolio outperformed the benchmark S&P 500 in both train and test period. For the whole period, the long-short portfolio of the top and bottom 100 stocks was the only portfolio that outperformed the benchmark. In addition, the LSTM model seems to perform better in the test period as the Sharpe ratios in this period across all portfolios are higher than in the train period. This could be an indication that the model can generalize well. However, the abnormally high Sharpe ratios in the test period raise some concerns because it could reflect certain special events during this period that affected the Sharpe ratio. In other words, we may not be able to replicate this same pattern in the future.

We also see the effect of diversification here: the higher the number of stocks, the higher the Sharpe ratio.

In [ ]: