PREDICTING ANALYST RECOMMENDATION FOR S&P500 STOCKS

University of Colorado Boulder

Master of Data Science

DTSA 5510 Unsupervised Learning -- Week 5 -- Final Project --

Introduction and Key Objectives:

Investors big and small rely on a host of financial information published by listing corporations to build and manage their financial portfolios. Financial Analysts rely on the same sets of publications to rate stocks and make recommendations, usually in the form of advise "buy", "strong buy", "hold", "underperform". Would it be possible to gather financial data from a group of stocks and make a prediction of what the avergae of analyst recommendations would be? In this final project I will attempt to do exactly that. I will first gather the necessary financial data, clean it, and then fit models to it with the main objective of predicting the average analyst recommendation.
On the more technical front, I source the data set consisting of financial information, metrics, and enginered features --available through Python Yahoo Finance Library-- and first fit a K-Means Unsupervised Model on training and testing sets. Since the download contains labels (for Analyst Recommendations), I can also fit a Supervised Logistic Regression Model, measure scores on both models and compare them.

Import Libraries:

In [123…

```
import yfinance as yf
import pandas as pd
import numpy as np
import os
import os.path
#from datetime import datetime
#import datetime
import time
import warnings
import seaborn as sns
import matplotlib.pyplot as plt
import itertools

# PCA decomposition:
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, recall_score, precision_score, confusio

# Combine requests_cache with rate-limiting to avoid triggering Yahoo's rate-limite
from requests import Session
from requests_cache import CacheMixin, SQLiteCache
from requests_ratelimiter import LimiterMixin, MemoryQueueBucket
from pyrate_limiter import Duration, RequestRate, Limiter
class CachedLimiterSession(CacheMixin, LimiterMixin, Session):
    pass

session = CachedLimiterSession(
    limiter=Limiter(RequestRate(2, Duration.SECOND*5)), # max 2 requests per 5 seco
    bucket_class=MemoryQueueBucket,
    backend=SQLiteCache("yfinance.cache")
)



# This function builds a path to the data folder where we save the data files.
def buildFilePath( tickerSymbol ):
    cwd = os.getcwd()
    pathToData = cwd +"/data/"
    fileName = pathToData + tickerSymbol + ".csv"
    return fileName

# Download a list of 500 tickers from wikipedia:
def list_wikipedia_sp500() -> pd.DataFrame:
    # Ref: https://stackoverflow.com/a/75845569/
    url = 'https://en.m.wikipedia.org/wiki/List_of_S%26P_500_companies'
    sp500ListWiki = pd.read_html(url, attrs={'id': 'constituents'}, index_col='Symb
    sp500ListWiki.reset_index(inplace=True)
    fileName = buildFilePath( "sp500ListWiki" )
    sp500ListWiki.to_csv(fileName, header=True, index=False)
    return sp500ListWiki
```

Data Sources, Directories, and Code high level Structure:

The main data source is Yahoo Finance (YF), from which more than 150 financial features are available for each of the 500 constituents of the S&P500. A secondary data source with a list of S&P500 tickers is available in Wikipedia (and not available through Yahoo Finance).
The ticker list and the downloads from YF are saves in the "/data" subdirectory of the directory in which this notebook is saved. It essential that the "/data" subdirectory exist prior to running the code.

The code is divided in two main sections and files: the first section is dedicated solely at sourcing the data, which is done only once. The second section is dedicated to cleaning, EDA, and modelling. Each section is saved in its own .py file.

In [124…

```
#=============================================
# Download a list of 500 tickers from wikipedia:
# Run this code only once:
```

```
#================================================
#sp500WikiList = list_wikipedia_sp500()
#print(sp500WikiList)

# Load all other SP500 tickers as rows in the DF:
tickerNames = pd.read_csv(buildFilePath("sp500ListWiki"))["Symbol"]
#tickerNames = ["XOM", "MSFT", "AAPL", "GOOGL"] # used for testing.
print("\nLine 74: S&P500 Ticker Names:\n", tickerNames)
```

```
Line 74: S&P500 Ticker Names:
 0       MMM
1        AOS
2        ABT
3       ABBV
4        ACN
       ...
498      XYL
499      YUM
500     ZBRA
501      ZBH
502      ZTS
Name: Symbol, Length: 503, dtype: object
```

The download is performed for each security, one at the time, only once. The code is written such that if the file is present in "/data", it will not be downloaded again.

In [125…

```
tickerList = []
for tckr in tickerNames:
    if tckr == "BKR.A":
           tckr = "BRK-A"
    if tckr == "BKR.B":
           tckr = "BKR-B"
    #print("\nLine 111: ticker=", tckr)

    # If file does not exist, do:
    tempFileName = buildFilePath(tckr)
    if not(os.path.isfile(tempFileName)):
        #tempTicker = yf.Ticker(tckr, session = session)
        tempTicker = yf.Ticker(tckr)
        tempMetricsD = tempTicker.info
        tempDF = pd.DataFrame([tempMetricsD])
        tempDF.to_csv(tempFileName, index=False)

        # Save the succesful list of tickers downloaded:
        tickerList.append(tckr)
        tickerDF = pd.DataFrame(tickerList, columns=["tickerName"])
        tempFileName = buildFilePath("tickerDF")
        tickerDF.to_csv(tempFileName, index=False)

        # Delay the next call randomly to avoid locking at yf.
        randNum = random.randint(2, 10)
        time.sleep(randNum)
    #else:
        #print("\nLine 133: ticker=", tckr, "\nfile exists.")
```

The code script also compiles a data frame with all 500 constituent stocks, and saves the full data set in a .csv file.

Exploratory Data Analysis (EDA), Cleaning, and Pre-processing:

Each stock has 179 features, with each feacture describing a piece of information about the listing corporation. While we may find non-financial information in the download, most of the features are financial in nature. This is a fairly large amount of features (in relation to the size of the 500 sample), and as a result I plan to apply a dimensionality reduction strategy prior to modelling. The first reduction is in data type. I will keep int and float data types and eliminate the rest, save for the analyst recommendation, which is a string and strictly related to the objective of this project.

In [126…

```
# Compile a single financial metrics data frame with all SP500 stocks:
# Initialize the sp500FinMetricsDF with 3M.
tempFileName = buildFilePath("MMM")
fullStockFinMetrics = pd.read_csv(tempFileName)
featureNames = list(fullStockFinMetrics.columns)
print("\nLine 31:", tempFileName, "\n", fullStockFinMetrics)

for j in range( len(featureNames)):
    tempDt = fullStockFinMetrics[featureNames[j]].dtype
    print("featureName:", featureNames[j], "; data type:", tempDt)
```

```
Line 31: /mnt/c/Users/LRT/LRTData/MCS/ColoradoBoulder/CUMSDS/Core/DTSA5510MLUnsuperv
ised/week5/data/MMM.csv
     address1        city state         zip        country         phone  \
0  3M Center  Saint Paul    MN  55144-1000  United States  651 733 1110

              website       industry    industryKey   industryDisp  ...  \
0  https://www.3m.com  Conglomerates  conglomerates  Conglomerates  ...

  cryptoTradeable hasPrePostMarketData firstTradeDateMilliseconds  \
0           False                 True              -252322200000

  postMarketChangePercent   shortName    longName  regularMarketChangePercent  \
0               -0.615857  3M Company  3M Company                    -1.97443

   regularMarketPrice  marketState  trailingPegRatio
0              138.02         POST            3.2761

[1 rows x 179 columns]
featureName: address1 ; data type: object
featureName: city ; data type: object
featureName: state ; data type: object
featureName: zip ; data type: object
featureName: country ; data type: object
featureName: phone ; data type: object
featureName: website ; data type: object
featureName: industry ; data type: object
featureName: industryKey ; data type: object
featureName: industryDisp ; data type: object
featureName: sector ; data type: object
featureName: sectorKey ; data type: object
featureName: sectorDisp ; data type: object
featureName: longBusinessSummary ; data type: object
featureName: fullTimeEmployees ; data type: int64
featureName: companyOfficers ; data type: object
featureName: auditRisk ; data type: int64
featureName: boardRisk ; data type: int64
featureName: compensationRisk ; data type: int64
featureName: shareHolderRightsRisk ; data type: int64
featureName: overallRisk ; data type: int64
featureName: governanceEpochDate ; data type: int64
featureName: compensationAsOfEpochDate ; data type: int64
featureName: irWebsite ; data type: object
featureName: executiveTeam ; data type: object
featureName: maxAge ; data type: int64
featureName: priceHint ; data type: int64
featureName: previousClose ; data type: float64
featureName: open ; data type: float64
featureName: dayLow ; data type: float64
featureName: dayHigh ; data type: float64
featureName: regularMarketPreviousClose ; data type: float64
featureName: regularMarketOpen ; data type: float64
featureName: regularMarketDayLow ; data type: float64
featureName: regularMarketDayHigh ; data type: float64
featureName: dividendRate ; data type: float64
featureName: dividendYield ; data type: float64
featureName: exDividendDate ; data type: int64
featureName: payoutRatio ; data type: float64
featureName: fiveYearAvgDividendYield ; data type: float64
featureName: beta ; data type: float64
```

featureName: trailingPE ; data type: float64
featureName: forwardPE ; data type: float64
featureName: volume ; data type: int64
featureName: regularMarketVolume ; data type: int64
featureName: averageVolume ; data type: int64
featureName: averageVolume10days ; data type: int64
featureName: averageDailyVolume10Day ; data type: int64
featureName: bid ; data type: float64
featureName: ask ; data type: float64
featureName: bidSize ; data type: int64
featureName: askSize ; data type: int64
featureName: marketCap ; data type: int64
featureName: fiftyTwoWeekLow ; data type: float64
featureName: fiftyTwoWeekHigh ; data type: float64
featureName: priceToSalesTrailing12Months ; data type: float64
featureName: fiftyDayAverage ; data type: float64
featureName: twoHundredDayAverage ; data type: float64
featureName: trailingAnnualDividendRate ; data type: float64
featureName: trailingAnnualDividendYield ; data type: float64
featureName: currency ; data type: object
featureName: tradeable ; data type: bool
featureName: enterpriseValue ; data type: int64
featureName: profitMargins ; data type: float64
featureName: floatShares ; data type: int64
featureName: sharesOutstanding ; data type: int64
featureName: sharesShort ; data type: int64
featureName: sharesShortPriorMonth ; data type: int64
featureName: sharesShortPreviousMonthDate ; data type: int64
featureName: dateShortInterest ; data type: int64
featureName: sharesPercentSharesOut ; data type: float64
featureName: heldPercentInsiders ; data type: float64
featureName: heldPercentInstitutions ; data type: float64
featureName: shortRatio ; data type: float64
featureName: shortPercentOfFloat ; data type: float64
featureName: impliedSharesOutstanding ; data type: int64
featureName: bookValue ; data type: float64
featureName: priceToBook ; data type: float64
featureName: lastFiscalYearEnd ; data type: int64
featureName: nextFiscalYearEnd ; data type: int64
featureName: mostRecentQuarter ; data type: int64
featureName: earningsQuarterlyGrowth ; data type: float64
featureName: netIncomeToCommon ; data type: int64
featureName: trailingEps ; data type: float64
featureName: forwardEps ; data type: float64
featureName: lastSplitFactor ; data type: object
featureName: lastSplitDate ; data type: int64
featureName: enterpriseToRevenue ; data type: float64
featureName: enterpriseToEbitda ; data type: float64
featureName: 52WeekChange ; data type: float64
featureName: SandP52WeekChange ; data type: float64
featureName: lastDividendValue ; data type: float64
featureName: lastDividendDate ; data type: int64
featureName: quoteType ; data type: object
featureName: currentPrice ; data type: float64
featureName: targetHighPrice ; data type: float64
featureName: targetLowPrice ; data type: float64
featureName: targetMeanPrice ; data type: float64
featureName: targetMedianPrice ; data type: float64
featureName: recommendationMean ; data type: float64

```
featureName: recommendationKey ; data type: object
featureName: numberOfAnalystOpinions ; data type: int64
featureName: totalCash ; data type: int64
featureName: totalCashPerShare ; data type: float64
featureName: ebitda ; data type: int64
featureName: totalDebt ; data type: int64
featureName: quickRatio ; data type: float64
featureName: currentRatio ; data type: float64
featureName: totalRevenue ; data type: int64
featureName: debtToEquity ; data type: float64
featureName: revenuePerShare ; data type: float64
featureName: returnOnAssets ; data type: float64
featureName: returnOnEquity ; data type: float64
featureName: grossProfits ; data type: int64
featureName: freeCashflow ; data type: int64
featureName: operatingCashflow ; data type: int64
featureName: earningsGrowth ; data type: float64
featureName: revenueGrowth ; data type: float64
featureName: grossMargins ; data type: float64
featureName: ebitdaMargins ; data type: float64
featureName: operatingMargins ; data type: float64
featureName: financialCurrency ; data type: object
featureName: symbol ; data type: object
featureName: language ; data type: object
featureName: region ; data type: object
featureName: typeDisp ; data type: object
featureName: quoteSourceName ; data type: object
featureName: triggerable ; data type: bool
featureName: customPriceAlertConfidence ; data type: object
featureName: corporateActions ; data type: object
featureName: postMarketTime ; data type: int64
featureName: regularMarketTime ; data type: int64
featureName: exchange ; data type: object
featureName: messageBoardId ; data type: object
featureName: exchangeTimezoneName ; data type: object
featureName: exchangeTimezoneShortName ; data type: object
featureName: gmtOffSetMilliseconds ; data type: int64
featureName: market ; data type: object
featureName: esgPopulated ; data type: bool
featureName: postMarketPrice ; data type: float64
featureName: postMarketChange ; data type: float64
featureName: regularMarketChange ; data type: float64
featureName: regularMarketDayRange ; data type: object
featureName: fullExchangeName ; data type: object
featureName: averageDailyVolume3Month ; data type: int64
featureName: fiftyTwoWeekLowChange ; data type: float64
featureName: fiftyTwoWeekLowChangePercent ; data type: float64
featureName: fiftyTwoWeekRange ; data type: object
featureName: fiftyTwoWeekHighChange ; data type: float64
featureName: fiftyTwoWeekHighChangePercent ; data type: float64
featureName: fiftyTwoWeekChangePercent ; data type: float64
featureName: dividendDate ; data type: int64
featureName: earningsTimestamp ; data type: int64
featureName: earningsTimestampStart ; data type: int64
featureName: earningsTimestampEnd ; data type: int64
featureName: earningsCallTimestampStart ; data type: int64
featureName: earningsCallTimestampEnd ; data type: int64
featureName: isEarningsDateEstimate ; data type: bool
featureName: epsTrailingTwelveMonths ; data type: float64
```

```
featureName: epsForward ; data type: float64
featureName: epsCurrentYear ; data type: float64
featureName: priceEpsCurrentYear ; data type: float64
featureName: fiftyDayAverageChange ; data type: float64
featureName: fiftyDayAverageChangePercent ; data type: float64
featureName: twoHundredDayAverageChange ; data type: float64
featureName: twoHundredDayAverageChangePercent ; data type: float64
featureName: sourceInterval ; data type: int64
featureName: exchangeDataDelayedBy ; data type: int64
featureName: averageAnalystRating ; data type: object
featureName: cryptoTradeable ; data type: bool
featureName: hasPrePostMarketData ; data type: bool
featureName: firstTradeDateMilliseconds ; data type: int64
featureName: postMarketChangePercent ; data type: float64
featureName: shortName ; data type: object
featureName: longName ; data type: object
featureName: regularMarketChangePercent ; data type: float64
featureName: regularMarketPrice ; data type: float64
featureName: marketState ; data type: object
featureName: trailingPegRatio ; data type: float64
```

In [127…

```
# I will keep all numerical columns to apply PCA in the next step. Except analyst r
columnsDF = ["sector", "fullTimeEmployees","auditRisk","boardRisk","compensationRis
             "regularMarketDayLow", "regularMarketDayHigh","dividendRate","dividend
             "averageVolume10days", "averageDailyVolume10Day", "bid", "ask","bidSiz
             "trailingAnnualDividendRate", "trailingAnnualDividendYield", "enterpri
             "sharesPercentSharesOut", "heldPercentInsiders", "heldPercentInstituti
             "trailingEps", "forwardEps", "enterpriseToRevenue","enterpriseToEbitda
             "recommendationMean", "recommendationKey", "numberOfAnalystOpinions",
             "returnOnEquity","grossProfits","freeCashflow", "operatingCashflow", "
             "averageDailyVolume3Month","fiftyTwoWeekLowChange","fiftyTwoWeekLowCha
             "priceEpsCurrentYear", "fiftyDayAverageChange", "fiftyDayAverageChange
             "trailingPegRatio"]

print( "\nLine 38: feature list size:", len(columnsDF))
```

```
Line 38: feature list size: 112
```

With the list of features now reduced from 179 to 112, I will compile all 500 stocks into one single data frame, and save the file for further cleaning and pre-processing. Not all stocks report the same information, and for many of them, the set is incomplete. I take "MMM" as a reference since it is a long established, and reports a very complete set of financial metrics. I initialize the dataframe with 3M, and its reported features become the features all other stocks will follow (if and when reported).

In [128…

```
# Compile a single financial metrics data frame with all SP500 stocks:
# Initialize the sp500FinMetricsDF with 3M.
tempFileName = buildFilePath("MMM")
sp500FinMetricsDF = pd.read_csv(tempFileName)[columnsDF]
#print("\nLine 31:", tempFileName, "\n\n", fullStockFinMetrics, "\n\n", featureName

# List of all tickers listed on the SP500:
tickerList = pd.read_csv(buildFilePath("tickerDF"))["tickerName"]
#tickerList = ["XOM", "AAPL"]
#print("\nLine 64:", tickerList)

# Go over each ticker and concatenate its information in the sp500FinMetricsDF:
if not(os.path.isfile(buildFilePath("sp500FinMetricsDF"))):
```

```
    columnsS = set(columnsDF.copy())
    for tckr in tickerList:
            tempDF = pd.DataFrame([ [None] * len(columnsDF)], columns=columnsDF)
            if( not tckr=="MMM"): # 3M is already there since the intialization.
                    print("\nLine 74 ticker=\n", tckr)
                    tempStock = pd.read_csv( buildFilePath(tckr))
                    tempColNames = tempStock.columns.tolist()
                    #print(tempColNames)
                    # Go over the Column Names and assign the values into the tempo
                    for j in range( len(tempColNames)):
                            if( tempColNames[j] in columnsS ):
                                    tempDF[ tempColNames[j] ] = tempStock.iloc[0,j]
                    if not tempDF.empty:
                            warnings.simplefilter(action='ignore', category=FutureW
                            sp500FinMetricsDF = pd.concat( [sp500FinMetricsDF, temp
    # This is the full SP500 dataframe with all numerical values.
    print("\nLine 85 sp500FinMetricsDF:\n", sp500FinMetricsDF)
    sp500FinMetricsDF.to_csv( buildFilePath("sp500FinMetricsDF"))
else: # File exists. Read this saved file from the yf data download:
    sp500FinMetricsDF = pd.read_csv(buildFilePath("sp500FinMetricsDF"))

pcaDF = sp500FinMetricsDF.copy()
print(pcaDF.shape)
```

```
(503, 117)
```

This results in a complete set of 503 stocks with 117 features.

While it is true that the financial picture of a company is multi faceted, it could be totally possible that the 117 or so financial line items, metrics, and engineered metrics that the set brings may be overlapping and reporting about similar financial dimensions.

I will explore this in the next step. If there is indeed a degree of overlapping, we can discover that through Principal Components Analysis (PCA), and further reduce the set of features, prior to modelling. PCA can help make clusters (of analyst recommendations in this case) more clearly separated.

PCA does not accept NaN or missing values in the set. We know we have a few, since not all stocks report on every metric. A quick fix is to eliminate incomplete stocks data sets altogether. This is simple, and fast. However, it reduces the 500 list to stocks to about half its size.

I chose to perform cleaning more surgically by looking at features first, particularly at those for which many stocks show missing inputs. We can look at them in groups. Not all stocks pay dividens. In fact there are 109 of them not reporting dividend related metrics. This can be solved by assign zero to the missing value (as opposed to deleting the stock altogether).

In [129…

```
# PCA does not accept NaN values. We need to remove features with NaN or stocks wit
# Let's look at features high number of NaN:
# We can see that the dividend related features have 94 and 109 NaN.
# This may be due to the fact that not all stocks pay dividends.
# I will assign zero to those NaNs.
pcaDF["dividendYield"] = pcaDF["dividendYield"].fillna(0)
pcaDF["dividendRate"] = pcaDF["dividendRate"].fillna(0)
pcaDF["fiveYearAvgDividendYield"] = pcaDF["fiveYearAvgDividendYield"].fillna(0)
```

```
pcaDF["lastDividendValue"] = pcaDF["lastDividendValue"].fillna(0)
```

Something similar happends with PE related measures. When earnings are zero, or negative, these measures don't offer any meaningfull piece of information. This is often the case with companies with volatile, declining, or just no earnings. I will set them to zero.

In [130…
```
# There are 85 (out of 500) stocks in this situation.
# I will set these values to zero, indicating that there is no earning growth.
pcaDF["trailingPegRatio"] = pcaDF["trailingPegRatio"].fillna(0)
# Same thing happens with (quaterly) earnings growth. 64 stocks show NaN, meaning n
pcaDF["earningsQuarterlyGrowth"] = pcaDF["earningsQuarterlyGrowth"].fillna(0)
pcaDF["earningsGrowth"] = pcaDF["earningsGrowth"].fillna(0)
```

Debt to equity ratio, free cash flow, return on equity, quick ratio, operating cash flow, current ratio, return on assets will be set to zero when missing for the same reasons explained above.

In [131…
```
# Debt to Equity ratio:
pcaDF["debtToEquity"] = pcaDF["debtToEquity"].fillna(0)

# Free cash flow: 60 NaN
pcaDF["freeCashflow"] = pcaDF["freeCashflow"].fillna(0)

# return On Equity: 43 NaN
pcaDF["returnOnEquity"] = pcaDF["returnOnEquity"].fillna(0)

# quick ratio:
pcaDF["quickRatio"] = pcaDF["quickRatio"].fillna(0)
pcaDF["operatingCashflow"] = pcaDF["operatingCashflow"].fillna(0)
pcaDF["currentRatio"] = pcaDF["currentRatio"].fillna(0)
pcaDF["enterpriseToEbitda"] = pcaDF["enterpriseToEbitda"].fillna(0)
pcaDF["ebitda"] = pcaDF["ebitda"].fillna(0)
pcaDF["trailingPE"] = pcaDF["trailingPE"].fillna(0)
pcaDF["returnOnAssets"] = pcaDF["returnOnAssets"].fillna(0)

# Short as a percent of float: 1 stock. May be because there is no shorts on it.
pcaDF["shortPercentOfFloat"] = pcaDF["shortPercentOfFloat"].fillna(0)
```

This process has so far allowed us to replace missing NaN values with zeros, and keep the stock on the list in doing so.

The next step is to look at stocks, along with features. We can see that there are 14 stocks missing beta values. This happends with stocks which have been recently listed and do not have a sufficciently long track record established to compute beta. Removing these type of stocks from the list is not a big loss. Same thing goes with stocks which do not have a recommendation reported on them. This happens to stocks that get no coverage, and are mostly not interesting. And the same thing happens with companies which are too small to take the costs of risk assessments.

In [132…
```
# Drop the stocks which do not have beta history.
pcaDF = pcaDF.dropna(subset=["beta"])

# Drop the stocks which do not have Recommendation Mean:
pcaDF = pcaDF.dropna(subset=["recommendationMean"])
```

```
# audit risk: 4 stocks
pcaDF = pcaDF.dropna(subset=["auditRisk"])

# enterprise value: 4 stocks
pcaDF = pcaDF.dropna(subset=["enterpriseValue"])

# Full time employees: 3 stocks
pcaDF = pcaDF.dropna(subset=["fullTimeEmployees"])

# Earnings per share not reported: 1 stock.
pcaDF = pcaDF.dropna(subset=["epsCurrentYear"])

# governanceEpochDate and compensationAsOfEpochDate have the same value for all sto
pcaDF = pcaDF.drop(columns=["governanceEpochDate"])
pcaDF = pcaDF.drop(columns=["compensationAsOfEpochDate"])

# This leads to a dataframe of 465 stocks with 114 features.
#print("\nLine 194: \n", "\npcaDF=\n", pcaDF.head(), pcaDF.shape)

pcaDF.to_csv("pcaDFClean.csv")
pcaFullCleanDF = pcaDF.copy()

# Remove non numerical features to apply PCA.
recKeys = pd.unique(pcaDF["recommendationKey"])
print("\nLine 191: recKeys=", recKeys) # total 4 =  4clusters.


pcaDF.drop(columns=["Unnamed: 0", "sector", "symbol", "recommendationKey" ], inplac
# I will later use KMeans to predict the buy/hold recommendation.
# I will remove this feature and recommendation Mean from the training model.
print("\nLine 199: \n", "\npcaDF=\n", pcaDF.head(), pcaDF.shape)
```

```
Line 191: recKeys= ['buy' 'hold' 'strong_buy' 'underperform']

Line 199:

pcaDF=
    fullTimeEmployees  auditRisk  boardRisk  compensationRisk  \
0             61500.0        1.0        7.0               6.0
1             12700.0        4.0       10.0               7.0
2            114000.0        9.0        7.0               9.0
3             55000.0        6.0        5.0               2.0
4            801000.0        1.0        4.0               4.0

   shareHolderRightsRisk  overallRisk   maxAge  priceHint  previousClose  \
0                    4.0          4.0  86400.0          2         140.80
1                   10.0         10.0  86400.0          2          68.34
2                    2.0          7.0  86400.0          2         132.60
3                    9.0          7.0  86400.0          2         196.07
4                    4.0          2.0  86400.0          2         307.71

     open  ...  epsCurrentYear  priceEpsCurrentYear  fiftyDayAverageChange  \
0  139.13  ...        7.67242            17.989109              -5.044403
1   67.70  ...        3.77489            17.873370               1.807602
2  131.95  ...        5.15265            25.823605               2.409790
3  195.50  ...       12.22512            15.308643             -10.585999
4  303.86  ...       12.72383            23.876457              -8.364624

   fiftyDayAverageChangePercent  twoHundredDayAverageChange  \
0                     -0.035260                    2.587997
1                      0.027529                   -6.196701
2                      0.018445                   13.499649
3                     -0.053536                   -1.854462
4                     -0.026796                  -39.355225

   twoHundredDayAverageChangePercent  postMarketChangePercent  \
0                           0.019109                -0.615857
1                          -0.084118                -0.681783
2                           0.112911                 0.000000
3                          -0.009812                 0.721350
4                          -0.114686                 1.020410

   regularMarketChangePercent  regularMarketPrice  trailingPegRatio
0                   -1.974430              138.02            3.2761
1                   -1.273040               67.47            1.7836
2                    0.346902              133.06            0.0000
3                   -4.549400              187.15            0.4192
4                   -1.270680              303.80            2.4046

[5 rows x 111 columns] (465, 111)
```

At this stage of the cleaning process we have a dataframe with 465 (down from 502 originally available) and 111 features (from the initial 179), with no NaN or missing data, ready to pass it on to PCA algorithm.

Lets verify that's indeed the case.

In [134…

```
countMax = 0
colNameMax=None
pcaColList = pcaDF.columns.to_list()
```

```
for j in range(len(pcaColList)):
    tempNaNCount = pcaDF[pcaColList[j]].isna().sum()
    if(tempNaNCount > countMax):
            colNameMax = pcaColList[j]
            countMax = tempNaNCount
            print("\nLine 110: col Name=", pcaColList[j], "tempNaNCount=", tempNaNC
#print("\nLine 110: colNameMax=", colNameMax, "tempNaNCountMax=", countMax, "symbol
print("\nLine 110: colNameMax=", colNameMax, "tempNaNCountMax=", countMax)
```

```
Line 110: colNameMax= None tempNaNCountMax= 0
```

Dimensionality Reduction through PCA:

Further reduction may be possible through PCA for example. PCA works best with scaled variables. Scaling will be the next step in the process, followed by splitting in a training and testing sets, and the creation of a PCA object.

In [135…

```
#Standardize the data
scaler = StandardScaler()
pcaDFScaled = scaler.fit_transform(pcaDF)
print("\nLine 93: \n", "\nmainDF=\n", pcaDFScaled)

# Split train test:
y = pcaFullCleanDF["recommendationKey"].copy() # buy, strong buy, hold, underperfor
pcaDFScaledTrain, pcaDFScaledTest, yTrain, yTest = train_test_split(pcaDFScaled, y,

# Create a PCA instance for the traininset:
pca = PCA(n_components=0.95) # 95% of explained variance.
```

```
Line 93:

mainDF=
 [[ 0.02167972 -1.54772444  0.5433384  ... -0.8098067  -0.16446862
  -0.02305446]
 [-0.32107593 -0.49297329  1.60490293 ... -0.37463312 -0.31424913
  -0.08713424]
 [ 0.39042298  1.26494528  0.5433384  ...  0.63045102 -0.1749989
  -0.16371225]
 ...
 [-0.34074224 -1.54772444  0.89719324 ...  0.45234256  0.07507405
  -0.11625672]
 [-0.2908741  -0.84455701 -1.22593581 ...  1.64266158 -0.26159773
  -0.11576298]
 [-0.31334988 -0.49297329 -1.57979065 ... -2.79966535 -0.13931059
  -0.04377037]]

Principal components:
 [[-0.00290929  0.01234078  0.01262538 ... -0.02052629  0.17622321
  -0.00433348]
 [ 0.12126851  0.02372993  0.01463914 ...  0.00710312  0.00818944
  -0.00101122]
 [-0.00593624 -0.06561754 -0.02855123 ...  0.10164481  0.01500927
   0.01445421]
 ...
 [ 0.24634314  0.19546739  0.09098784 ...  0.18017517  0.00314721
  -0.03499878]
 [-0.07447606  0.0332074  -0.10791076 ... -0.19553031 -0.00337003
  -0.05135144]
 [ 0.05994501 -0.12325018  0.19980494 ... -0.25934509 -0.01896985
  -0.00763408]]

Explained variance ratio:
 [0.27430603 0.10877064 0.07250461 0.04572351 0.04077042 0.03448287
 0.02908077 0.02345398 0.02340235 0.02209655 0.02077375 0.02010036
 0.01834456 0.01691502 0.01666842 0.01478879 0.0133511  0.01313759
 0.01264615 0.01154205 0.01071271 0.01042362 0.00990456 0.00942922
 0.00917249 0.00857903 0.00797595 0.0076368  0.00686846 0.00636407
 0.00594086 0.00571966 0.00496942 0.00479745 0.00461442 0.00436405]

Cumulative explained variance:
 [0.27430603 0.38307667 0.45558128 0.50130478 0.5420752  0.57655807
 0.60563883 0.62909281 0.65249516 0.67459171 0.69536546 0.71546582
 0.73381038 0.7507254  0.76739382 0.78218261 0.79553371 0.8086713
 0.82131745 0.8328595  0.84357221 0.85399583 0.86390039 0.87332961
 0.88250209 0.89108113 0.89905707 0.90669387 0.91356233 0.9199264
 0.92586726 0.93158693 0.93655634 0.94135379 0.94596821 0.95033227]
```

The results of fitting the data to PCA is shown below. A total of 38 components explain 95% of the variance. This is an amazing result, and one in line with objective. We will be working with 38 out of 179 original features.

In [136…

```
pcaDFTransfTrain = pca.fit_transform(pcaDFScaledTrain)
print("\nPrincipal components:\n", pca.components_)
print("\nExplained variance ratio:\n", pca.explained_variance_ratio_)
print("\nCumulative explained variance:\n", pca.explained_variance_ratio_.cumsum())

# Create a PCA instance for the test set:
```

```
pcaDFTransfTest = pca.transform(pcaDFScaledTest)
```

```
Principal components:
 [[-0.00290929  0.01234078  0.01262538 ... -0.02052629  0.17622321
  -0.00433348]
 [ 0.12126851  0.02372993  0.01463914 ...  0.00710312  0.00818944
  -0.00101122]
 [-0.00593624 -0.06561754 -0.02855123 ...  0.10164481  0.01500927
   0.01445421]
 ...
 [ 0.24634314  0.19546739  0.09098784 ...  0.18017517  0.00314721
  -0.03499878]
 [-0.07447606  0.0332074  -0.10791076 ... -0.19553031 -0.00337003
  -0.05135144]
 [ 0.05994501 -0.12325018  0.19980494 ... -0.25934509 -0.01896985
  -0.00763408]]

Explained variance ratio:
 [0.27430603 0.10877064 0.07250461 0.04572351 0.04077042 0.03448287
 0.02908077 0.02345398 0.02340235 0.02209655 0.02077375 0.02010036
 0.01834456 0.01691502 0.01666842 0.01478879 0.0133511  0.01313759
 0.01264615 0.01154205 0.01071271 0.01042362 0.00990456 0.00942922
 0.00917249 0.00857903 0.00797595 0.0076368  0.00686846 0.00636407
 0.00594086 0.00571966 0.00496942 0.00479745 0.00461442 0.00436405]

Cumulative explained variance:
 [0.27430603 0.38307667 0.45558128 0.50130478 0.5420752  0.57655807
 0.60563883 0.62909281 0.65249516 0.67459171 0.69536546 0.71546582
 0.73381038 0.7507254  0.76739382 0.78218261 0.79553371 0.8086713
 0.82131745 0.8328595  0.84357221 0.85399583 0.86390039 0.87332961
 0.88250209 0.89108113 0.89905707 0.90669387 0.91356233 0.9199264
 0.92586726 0.93158693 0.93655634 0.94135379 0.94596821 0.95033227]
```

Principal Components from PCA may be hard to interprete. The components are newly engineered variables with no direct intuitive interpretation beyond being combinations of the original features. They explain a large majority of the original variance with the least amount of components.

In [140…

```
nComp = pca.components_.shape[0]
compName=[]
for j in range(nComp):
    tempName = "PC" + str(j+1)
    compName.append(tempName)
print(compName)

pcaWeightsDF = pd.DataFrame(
    pca.components_.T, # shape: (n_features, n_components)
    columns = compName,
    index=pcaDF.columns
)
# Result: 113 rows, each a feature, mapped with weights to each 36 principal compon
print("\nLine 243: pcaWeights=\n", pcaWeightsDF)
```

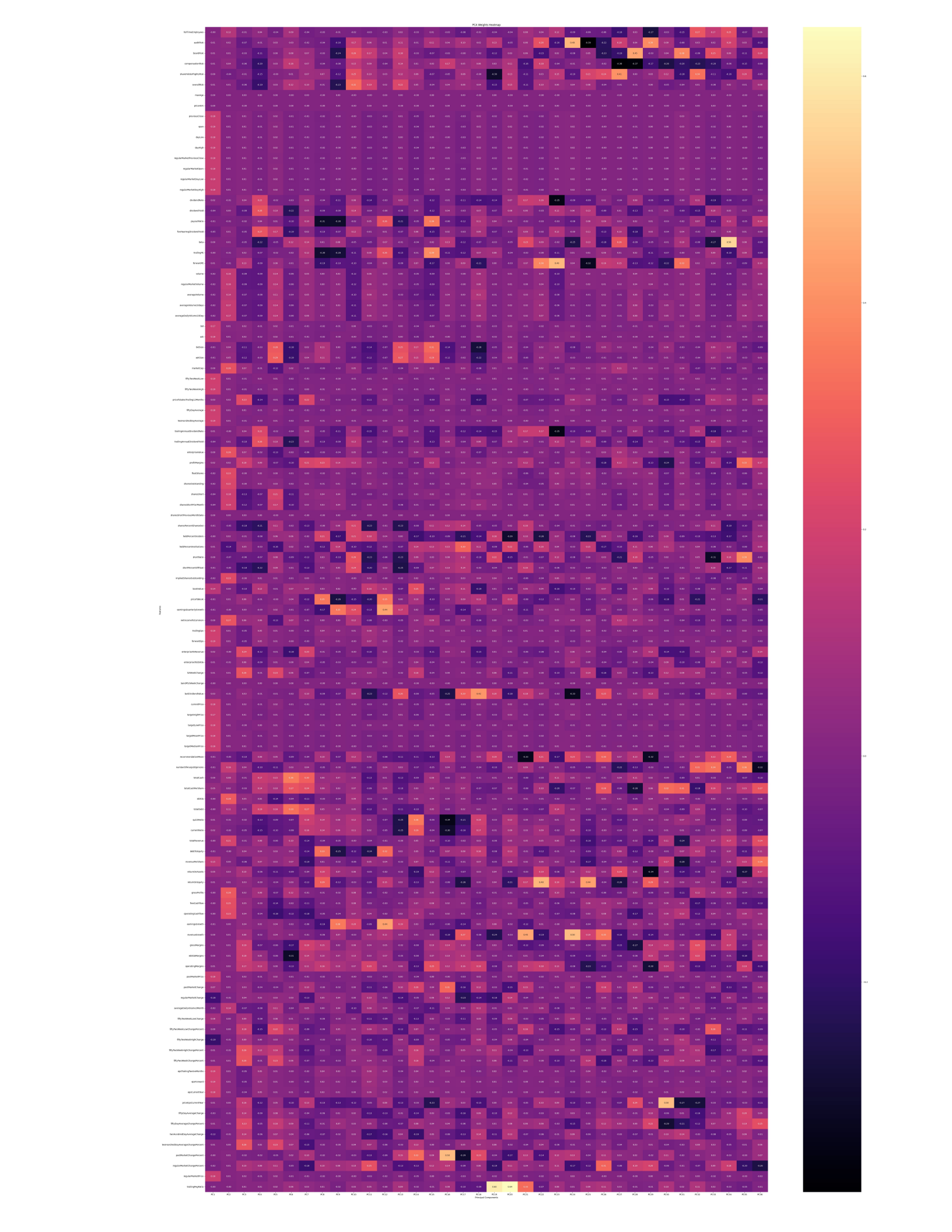
```
['PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6', 'PC7', 'PC8', 'PC9', 'PC10', 'PC11', 'PC1
2', 'PC13', 'PC14', 'PC15', 'PC16', 'PC17', 'PC18', 'PC19', 'PC20', 'PC21', 'PC22',
'PC23', 'PC24', 'PC25', 'PC26', 'PC27', 'PC28', 'PC29', 'PC30', 'PC31', 'PC32', 'PC3
3', 'PC34', 'PC35', 'PC36']

Line 243: pcaWeights=
                                        PC1       PC2       PC3       PC4  \
fullTimeEmployees                  -0.002909  0.121269 -0.005936  0.040404
auditRisk                           0.012341  0.023730 -0.065618 -0.011114
boardRisk                           0.012625  0.014639 -0.028551 -0.106623
compensationRisk                    0.007434  0.035819 -0.057923 -0.195279
shareHolderRightsRisk               0.004645 -0.041490 -0.014091 -0.148264
...                                      ...       ...       ...       ...
twoHundredDayAverageChangePercent   0.006194  0.003235  0.262408  0.018458
postMarketChangePercent            -0.002028  0.010416 -0.015975 -0.019916
regularMarketChangePercent         -0.020526  0.007103  0.101645  0.082593
regularMarketPrice                  0.176223  0.008189  0.015009 -0.014412
trailingPegRatio                   -0.004333 -0.001011  0.014454  0.013707

                                        PC5       PC6       PC7       PC8  \
fullTimeEmployees                  -0.039531  0.091039 -0.042840 -0.025864
auditRisk                           0.029590  0.034282 -0.024146 -0.041582
boardRisk                           0.004982  0.078691  0.066676 -0.027042
compensationRisk                    0.034361  0.155553  0.073429 -0.038086
shareHolderRightsRisk              -0.003142  0.009725  0.071565  0.065738
...                                      ...       ...       ...       ...
twoHundredDayAverageChangePercent   0.225097  0.074350 -0.150888 -0.017348
postMarketChangePercent            -0.051013  0.019604  0.100821 -0.024642
regularMarketChangePercent          0.107925 -0.048737 -0.177534  0.102465
regularMarketPrice                  0.021256 -0.013311 -0.011192 -0.019502
trailingPegRatio                    0.020425  0.061288  0.031276  0.014225

                                        PC9      PC10  ...      PC27  \
fullTimeEmployees                  -0.012527 -0.019036  ... -0.179681
auditRisk                          -0.164186  0.167256  ...  0.177604
boardRisk                          -0.238218  0.257403  ... -0.191438
compensationRisk                   -0.076577  0.147637  ... -0.384578
shareHolderRightsRisk              -0.124896  0.245100  ...  0.408903
...                                      ...       ...  ...       ...
twoHundredDayAverageChangePercent -0.004267  0.035978  ... -0.043517
postMarketChangePercent            -0.015290 -0.031992  ...  0.025001
regularMarketChangePercent          0.080641  0.096163  ... -0.081964
regularMarketPrice                 -0.004084 -0.000608  ... -0.001862
trailingPegRatio                   -0.022296  0.034644  ...  0.025123

                                       PC28      PC29      PC30      PC31  \
fullTimeEmployees                  0.033716 -0.266879 -0.032991 -0.146924
auditRisk                          0.038019  0.391957  0.090143 -0.078200
boardRisk                          0.449062 -0.033140  0.040945  0.381917
compensationRisk                  -0.371230 -0.170735 -0.257784 -0.202134
shareHolderRightsRisk              0.003703  0.034185  0.122036 -0.177239
...                                     ...       ...       ...       ...
twoHundredDayAverageChangePercent  0.025887  0.012616 -0.038203 -0.025085
postMarketChangePercent            0.023714 -0.018074  0.108787 -0.024038
regularMarketChangePercent         0.185136  0.202745 -0.060641 -0.005063
regularMarketPrice                -0.004793  0.001592  0.001243  0.027871
trailingPegRatio                  -0.012927 -0.008082  0.104947  0.018225

                                       PC32      PC33      PC34      PC35  \
```

```
fullTimeEmployees                  0.272194  0.168754  0.246343 -0.074476
auditRisk                          0.029506  0.015927  0.195467  0.033207
boardRisk                         -0.093515  0.246080  0.090988 -0.107911
compensationRisk                  -0.230907 -0.204556 -0.062306 -0.147966
shareHolderRightsRisk              0.335004 -0.107882 -0.157530  0.196198
...                                     ...       ...       ...       ...
twoHundredDayAverageChangePercent  0.078492  0.034287 -0.053738 -0.014334
postMarketChangePercent            0.011021  0.029254 -0.028858 -0.002389
regularMarketChangePercent        -0.070602  0.092719  0.180175 -0.195530
regularMarketPrice                 0.003125 -0.017467  0.003147 -0.003370
trailingPegRatio                   0.111036  0.024028 -0.034999 -0.051351

                                        PC36
fullTimeEmployees                   0.059945
auditRisk                          -0.123250
boardRisk                           0.199805
compensationRisk                   -0.079059
shareHolderRightsRisk              -0.053896
...                                      ...
twoHundredDayAverageChangePercent   0.063370
postMarketChangePercent             0.066086
regularMarketChangePercent         -0.259345
regularMarketPrice                 -0.018970
trailingPegRatio                   -0.007634

[111 rows x 36 columns]
```

The weight of each feature is the contribution of that feature to each principal component. We may visualize it on heatmap.

In [150…

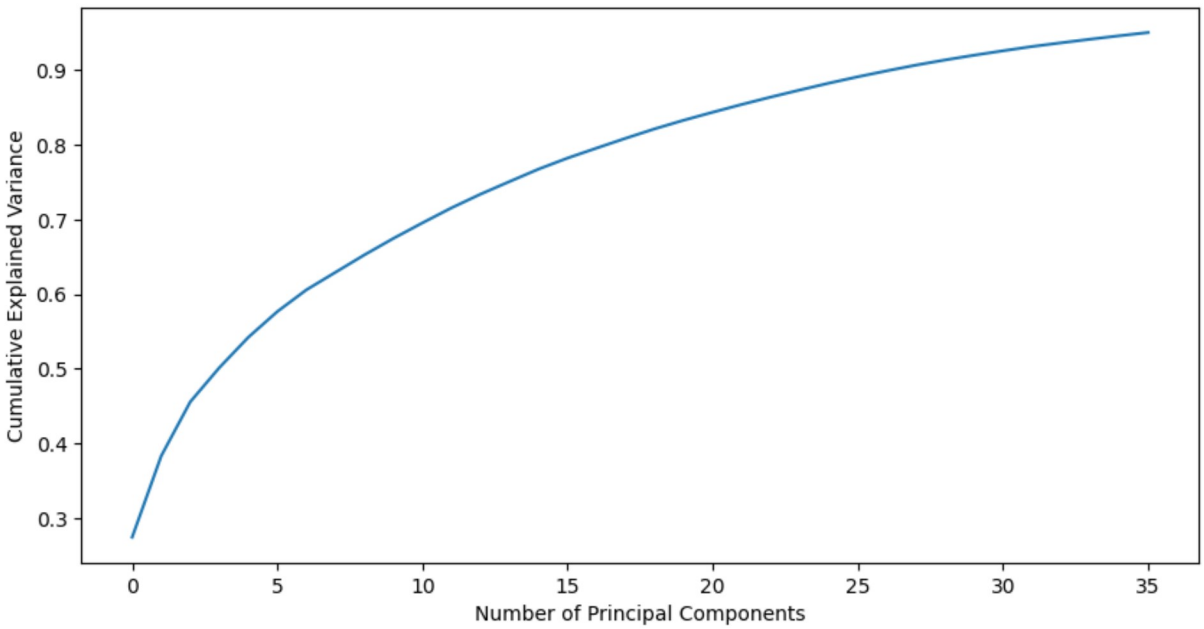
```
xLabels = []
for i in range(pcaWeightsDF.shape[1]):
    xLabels.append( "PC" + str(i+1)) ,
plt.figure(figsize=(60, 100))
#plt.subplots_adjust(left=0.1, right=0.15, top=0.15, bottom=0.1)
sns.heatmap(pcaWeightsDF, cmap= "magma", xticklabels= xLabels, yticklabels= pcaWeig
plt.ylabel("Features")
plt.xlabel("Principal Components")
plt.title("PCA Weights Heatmap")
plt.show()
#plt.savefig( "PCAWeightsHeatmap.png")
```


PCA Weights Heatmap
Features
Principal Components

The component explaining the largest amiount of variance is listed first. The cumulative variance explained is shown below to a limit of 95%.

The proportion of variance each component explains represents the amount of information from the original data that is captured in a given component.

In [151…
```
# Cumulative Explained variance:
# Plot
plt.figure(figsize=(10, 5))
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance')
plt.show()
#plt.savefig( "PCACumVarPlot.png")
```



Fit K-Means (Unsupervised Model), and Logistic Rgeression (Supervised):

We will first not use the labels we have, and fit a K-Means model to discover the 4 clusters [buy, strong buy, hold, underperform] that we know analyst use to rate stocks. Since we have the labels available, I will be able to measure a accuracy once the model is built. I will split the data set into training and testing sets, fit a K-Means model, fine tune hyperparameters, and compare the results to a supervised Logistic Regression Model.

In [156…
```
bestTrainAcc = 0
bestTrainDic = None
bestTrainCombi = None
bestTrainRs = 0
bestTrainAccPct = 0

bestTestAcc = 0
bestTestDic = None
bestTestCombi = None
bestTestRs = 0
bestTestAccPct = 0
bestRecKeyRealTest = None
```

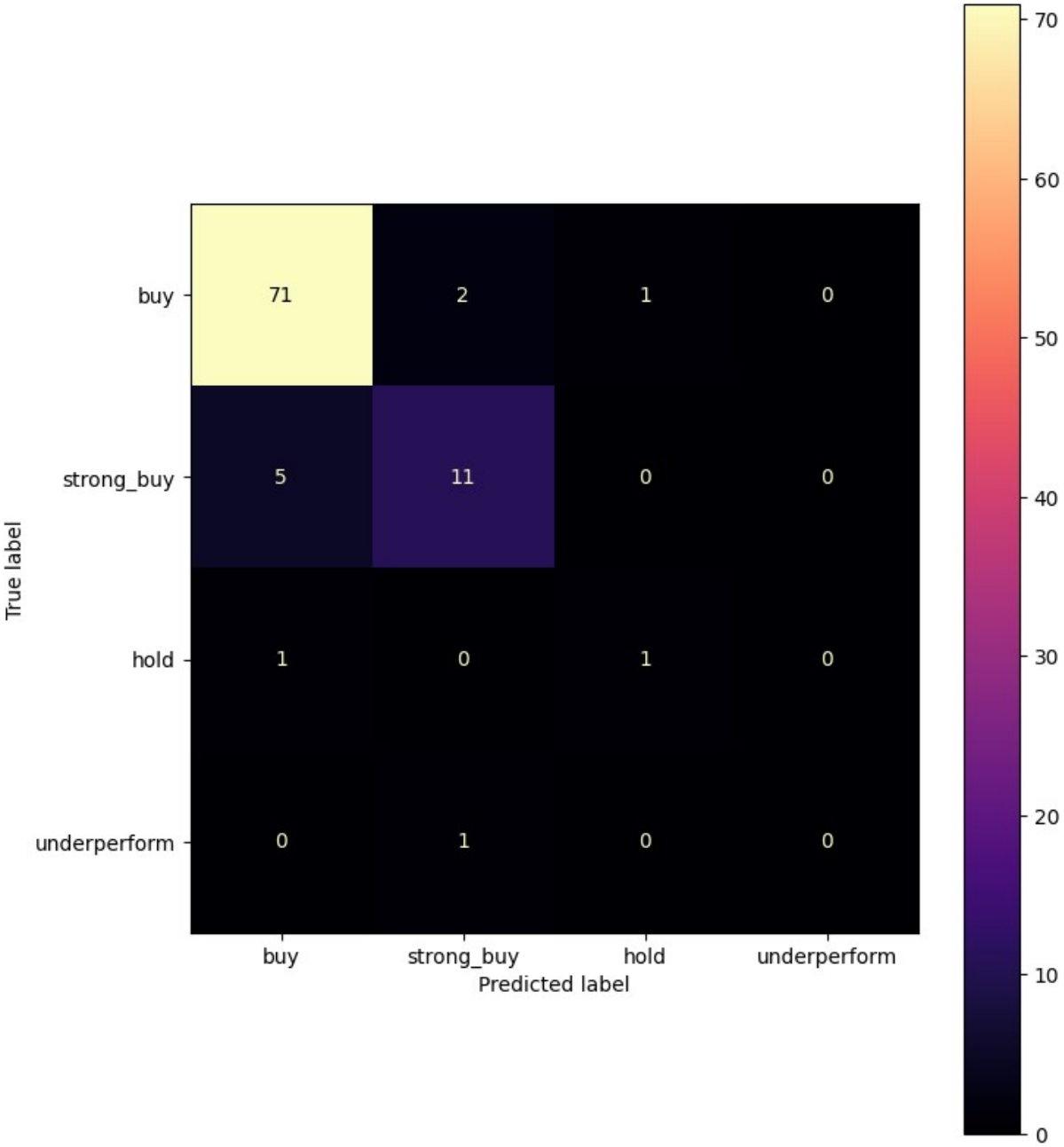
```
supModelBestAccPct = 0
supModelBestRs = 0
for rs in range(1):
    rs=29109
    print("\nLine295 Rs=", rs)
    kmeans = KMeans(n_clusters=len(recKeys), random_state=rs, max_iter=10000,
                    n_init="auto", init="k-means++", algorithm="elkan")

    #kmeans.fit(X_pca)
    kmeans.fit(pcaDFTransfTrain)

    # Predict cluster for training data:
    XTrainpca = pca.transform(pcaDFScaledTrain)
    #cluster_labels = kmeans.predict(X_new_pca)
    clustLabelsTrain = kmeans.predict(XTrainpca)

    # Predict cluster for testing data:
    XTestpca = pca.transform(pcaDFScaledTest)
    clustLabelsTest = kmeans.predict(XTestpca)

    #print(cluster_labels)
    #print(pcaFullCleanDF["recommendationKey"])

    # Numerical keys are assigned randomly.
    # There are 4 recommendation keys: buy, strong buy, hold, underperformed.
    # These keys will map to some combination of 0,1,2,3. There are 24 options.

    clusterNum = [0, 1, 2, 3]
    combiL = list(itertools.permutations(clusterNum))
    #print(combiL)

    recKeyDic = {"buy":0, "strong_buy":1, "hold":2, "underperform":3}
    recKeysL = list(recKeyDic.keys())

    recKeyTrain = list(yTrain.copy()) # original string recommendation buy, hold,..
    recKeyTest = list(yTest.copy()) # original string recommendation buy, hold,....

    for combi in combiL:
            # Build the dictionary for that combination:
            for i in range(len(combi)):
                    recKeyDic[recKeysL[i]] = combi[i]

            # Assign a numerical label to the original train string based on this c
            recKeyRealTrain = [] # numerical version of the string.
            for j in range( len(recKeyTrain)):
                    tempTrainkey = recKeyTrain[j]
                    recKeyRealTrain.append(recKeyDic[tempTrainkey])
            # Evaluate the accuracy of this combination in the training set:
            tempTrainAcc = 0
            for k in range( len(recKeyTrain)):
                    if(recKeyRealTrain[k] == clustLabelsTrain[k]):
                            tempTrainAcc += 1
            if(tempTrainAcc > bestTrainAcc):
                    bestTrainAcc = tempTrainAcc
                    bestTrainDic = recKeyDic.copy()
                    bestTrainCombi = combi
                    bestTrainRs = rs
                    bestTrainAccPct = round(bestTrainAcc/len(recKeyTrain)*100,2)
```

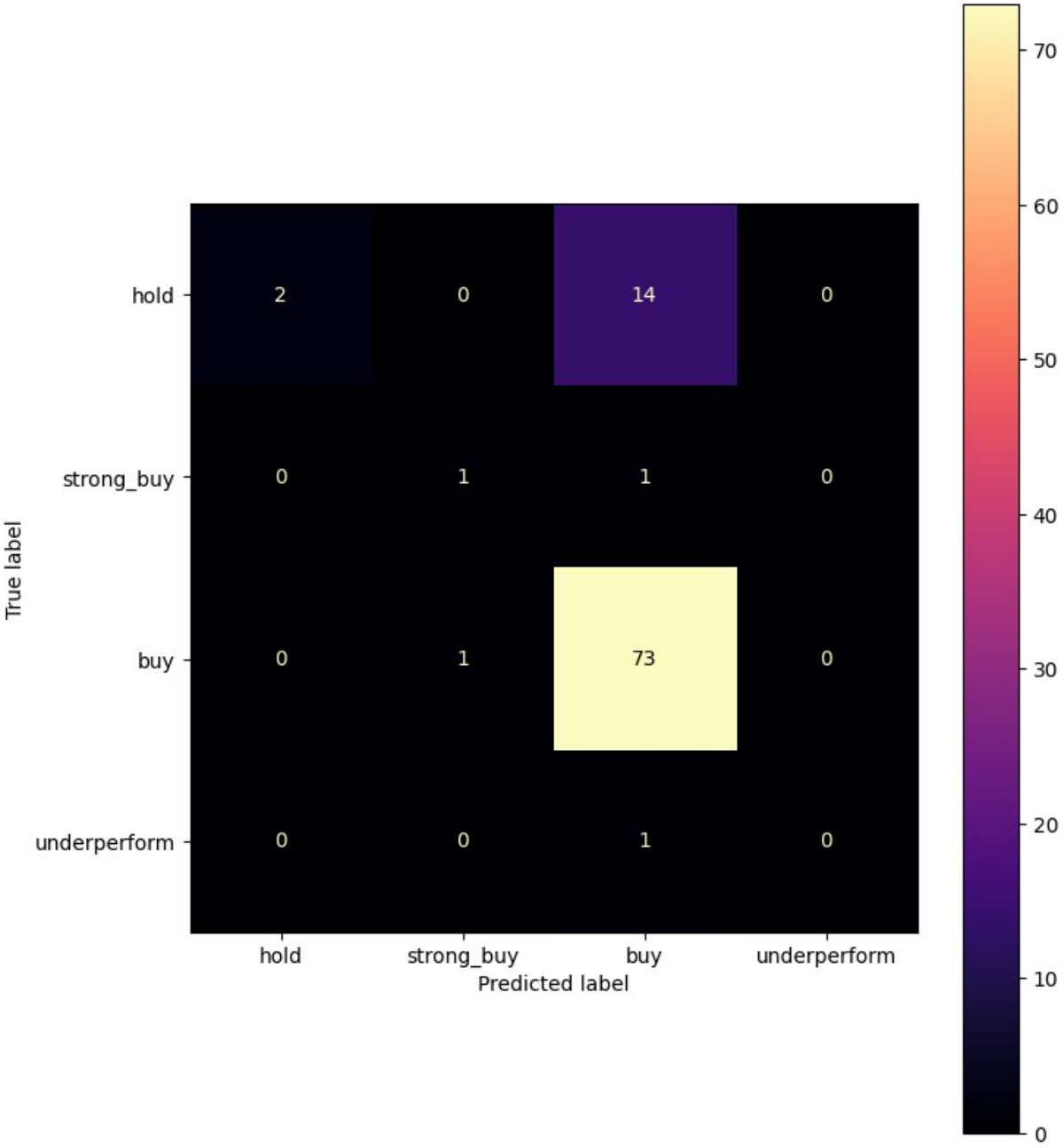
```
        # Assign a numerical label to the original TEST string based on this co
        recKeyRealTest = [] # numerical version of the string.
        for j in range( len(recKeyTest)):
                tempTestkey = recKeyTest[j]
                recKeyRealTest.append(recKeyDic[tempTestkey])
        # Evaluate the accuracy of this combination in the TEST set:
        tempTestAcc = 0
        for k in range( len(recKeyTest)):
                if(recKeyRealTest[k] == clustLabelsTest[k]):
                        tempTestAcc += 1
        if(tempTestAcc > bestTestAcc):
                bestTestAcc = tempTestAcc
                bestTestDic = recKeyDic.copy()
                bestTestCombi = combi
                bestTestRs = rs
                bestTestAccPct = round(bestTestAcc/len(recKeyTest)*100,2)
                bestRecKeyRealTest = recKeyRealTest
        if (combi == combiL[ len(combiL)-1]): # if this is the last combination
                # Confusion Matrix and metrics on the test set:
                confMatk = confusion_matrix(y_true = bestRecKeyRealTest, y_pred
                accuracyk = accuracy_score(y_true = bestRecKeyRealTest, y_pred
                recallk = recall_score(y_true = bestRecKeyRealTest, y_pred = cl
                #precisionk = precision_score(y_true = recKeyRealTest, y_pred =
                print("\nLine 392: Unsupervised KMeans Model Confusion Matrix:\
                print("\nLine 393: Unsupervised KMeans Model accuracy (testing
                #"\nRecall and precision calculated by category. \nCategories:\
                #"\nModel Precision =", precisionk,
                print("\nKmeans Unsupervised KMeans Model Recall %=", [round(x,

                #precMeank = precisionk.mean()
                recallMeank = recallk.mean()
                #print("\nLine 398: Model mean Precision:", round(precisionk.me
                print("\nLine 402: Kmeans Unsupervised Model Recall %: ", round

                bestTestDicSorted = dict(sorted(bestTestDic.items(), key=lambda
                confMatKMeansDisplay = ConfusionMatrixDisplay(confMatk, display
                figk, axk = plt.subplots(figsize=(8,10))
                confMatKMeansDisplay.plot(ax=axk, cmap='magma')
                figk.show
                #figk.savefig("confMatKMeansModel.png")

        # SUPERVISED MODEL:
        #====================
        supModel = LogisticRegression(max_iter=10000, random_state=rs)
        supModel.fit(pcaDFTransfTrain, yTrain)
        yPred = supModel.predict(pcaDFTransfTest)
        supAccPct = round(accuracy_score(yTest, yPred) *100,2)
        if(supAccPct > supModelBestAccPct):
                supModelBestAccPct = supAccPct
                supModelBestRs = rs
                # Confusion Matrix and metrics on the test set:
                confMatSup = confusion_matrix(y_true = yTest, y_pred = yPred)
                accuracySup = accuracy_score(y_true = yTest, y_pred = yPred)
                recallSup = recall_score(y_true = yTest, y_pred = yPred, averag
                precisionSup = precision_score(y_true = yTest, y_pred = yPred,
                print("\nLine 416 Supervised Learning Model: Logistic Regressio
                print("\nLine 417: Supervised LR Model accuracy (testing set) %
                "\nRecall and precision calculated by category. \nCategories:\n
```

```
                    "\nSupervised LR Model Precision %=", [round(x,2) for x in prec
                    "\nSupervised LR Model Recall %=", [round(x,2) for x in recallS

                    precMeanSup = precisionSup.mean()
                    recallMeanSup = recallSup.mean()
                    print("\nLine 424: Supervised LR Model mean Precision %:", roun
                    print("\nLine 425: Supervised LR Model mean Recall %:", round(r

                    confMatSupDisplay = ConfusionMatrixDisplay(confMatSup, display_
                    figSup, axSup = plt.subplots(figsize=(8,10))
                    confMatSupDisplay.plot(ax=axSup, cmap='magma')
                    figSup.show
                    #figSup.savefig("confMatSupLRModel.png")

print( "\nLine 429:\n", "\nTraining Set: \nbestAcc=", bestTrainAcc, "\nbestAccPct="

print( "\nLine 436:\n", "\nTesting Set: \nbestAcc=", bestTestAcc, "\nbestAccPct=",

print("\nLine443: Supervised LR model Testing Set accuracy %:", supModelBestAccPct,
```

```
Line295 Rs= 29109

Line 416 Supervised Learning Model: Logistic Regression. Confusion Matrix:
 [[71  2  1  0]
 [ 5 11  0  0]
 [ 1  0  1  0]
 [ 0  1  0  0]]

Line 417: Supervised LR Model accuracy (testing set) %= 89.25 %.

Line 424: Supervised LR Model mean Precision %: 55.19

Line 425: Supervised LR Model mean Recall %: 53.67
/home/lrt/.local/lib/python3.12/site-packages/sklearn/metrics/_classification.py:156
5: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels w
ith no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
Line 392: Unsupervised KMeans Model Confusion Matrix:
 [[ 2  0 14  0]
 [ 0  1  1  0]
 [ 0  1 73  0]
 [ 0  0  1  0]]

Line 393: Unsupervised KMeans Model accuracy (testing set) %= 81.72 .

Kmeans Unsupervised KMeans Model Recall %= [12.5, 50.0, 98.65, 0.0]

Line 402: Kmeans Unsupervised Model Recall %:  0.4

Line 429:

Training Set:
bestAcc= 273
bestAccPct= 73.39
bestDic:
 {'buy': 2, 'strong_buy': 1, 'hold': 0, 'underperform': 3}
bestCombi:
 (2, 1, 0, 3)
bestRs:
 29109

Line 436:

Testing Set:
bestAcc= 76
bestAccPct= 81.72
bestDic:
 {'buy': 2, 'strong_buy': 1, 'hold': 0, 'underperform': 3}
bestCombi:
 (2, 1, 0, 3)
bestRs:
 29109

Line443: Supervised LR model Testing Set accuracy %: 89.25 supModelBestRs= 29109
```


buy
strong_buy
hold
underperform
True label
71
2
1
0
5
11
0
0
1
0
1
0
0
1
0
0
buy
strong_buy
hold
underperform
Predicted label
70
60
50
40
30
20
10
0



Conclussions and Final Comments:

The confusion matrices for the K-Means (Unsupervised) and Logistic Regression (LR) (Supervised) are presented above. The K-Means unsupervised method finds the four clusters with 82% accuracy. While this is perhpas a bit low a threshhold to issue a sound recommendation, I find this an amazing result. In the absence of anything else but this data set, we would be able to make an informed, better, decision if faced with the question of rating a security. The supervised LR method does even better, wuth almost 90% accuracy. Dimensionality reduction through PCA was an important step to be able to get the algorithms to deliver results in reasonable time.

Future Developments:

The SP500 list of stocks is about 10% of the total of 6000 securities in the US, and a small fraction of all securities listed worldwide.

Future extensions of this work would include all US stocks first with international markets added progressively.

List of References:

https://pypi.org/project/yfinance/

https://www.geeksforgeeks.org/get-financial-data-from-yahoo-finance-with-python/

https://github.com/shilewenuw/get_all_tickers/blob/master/get_all_tickers/get_tickers.py

https://www.geeksforgeeks.org/principal-component-analysis-pca/

In [ ]:

In [ ]:

In [ ]:

In [ ]: