Data 698 - Mid term report¶

Tze Fung Lung, Jim¶

March 17, 2019¶

Topic: Portfolio optimization and Machine learning with visualization analysis for S&P 500¶

10 Keyword:¶

- S&P500
- Stocks
- Return
- Risk
- Strategic
- Linear
- k-Nearest
- Moving-Average
- ARIMA
- LSTM

Introduction¶

Objectives¶

Predicting how the stock market will perform is one of the most difficult things to do. There are so many factors involved in the prediction – physical factors vs. physiological, rational and irrational behavior, etc. All these aspects combine to make share prices volatile and very difficult to predict with a high degree of accuracy.

The S&P 500 is widely regarded as the best single gauge of large-cap U.S. equities. The index includes 500 leading companies and captures approximately 80% coverage of available market capitalization.

1. Description of the Problem¶

We'll look at the S&P 500, an index of the largest US companies. The S&P 500 is an American stock market index based on the market capitalization of 500 large companies having common stock listed on the NYSE, NASDAQ Exchange.

I will load all 500 dataset in S&P 500 for analysis by using portfolio optimization to get the possible several stocks

with higher return and lower risk. And using the machine learning predict the investment trend for S&P 500 index.

- What are the top 20 higher monthly return among all 500 number of stocks in S&P500 by Mathematical programming? The target is to find out the top valuable, higher return with lower risk of stocks.
- Could I invest these top 20 stocks now by analysis for the trend of S&P500 index by Machine learning? It is to determine if I could invest these stocks by choosing the most accuracy model with the trend.

2.Why the problem is interesting¶

Automatic trading without anyone involved will be the trend of stock market near future. I would like to use the data science methods to make a strategic for investment.

I will study which method of machine learning would be more accurate, suitable for prediction by using root-mean-squared error, that the prediction will be more meaningful in use.

3. What other approaches have been tried¶

First of all, I will construct the portfolio optimization in order to achieve a maximum expected return given their risk preferences due to the fact that the returns of a portfolio are greatly affected by nature of the relationship between assets and their weights in the portfolio.

The top 20 monthly return of stocks will be get into the portfolio optimization. Then in order to get the higher return and lower risk of stocks, the portfolio optimization will be conducted to find out which are the best choose of investment and generate the visualization for returns and volatility.

For the next part, I will work with historical data about the S&P500 price index to understand if I can invest in market this moment. I will implement a mix of machine learning algorithms to predict the future stock price of this company, starting with simple algorithms like averaging and linear regression, and then moving on to advanced techniques like Auto ARIMA and LSTM.

And I will compare the models by using root-mean-squared error (RMSE) to measure of how model performed and measure difference between predicted values and the actual values.

4. Discussion on your hypothesis is and how you specific solution will improve¶

Stock market analysis is divided into two parts – Fundamental Analysis and Technical Analysis.

Fundamental Analysis involves analyzing the company’s future profitability on the basis of its current business environment and financial performance. Technical Analysis, on the other hand, includes reading the charts and using statistical figures to identify the trends in the stock market.

We’ll scrape all S&P 500 tickers from Wiki and load all 500 dataset to be in cleaning and appending the adjusted closing price from 2008 to 2018.

Moving Average - The predicted closing price for each day will be the average of a set of previously observed values. Instead of using the simple average, we will be using the moving average technique which uses the latest set of values for each prediction.

Linear Regression - The most basic machine learning algorithm that can be implemented on this data is linear regression. The linear regression model returns an equation that determines the relationship between the independent variables and the dependent variable.

K-Nearest - Neighbors Another interesting ML algorithm that one can use here is kNN (k nearest neighbours). Based on the independent variables, kNN finds the similarity between new data points and old data points.

ARIMA - ARIMA is a very popular statistical method for time series forecasting. ARIMA models take into account the

past values to predict the future values.

Long Short Term Memory (LSTM) - LSTMs are widely used for sequence prediction problems and have proven to be extremely effective.

Literature review¶

1. S&P 500 Index¶

The S&P 500 or Standard & Poor's 500 Index is a market-capitalization-weighted index of the 500 largest U.S. publicly traded companies. The index is widely regarded as the best gauge of large-cap U.S. equities.

The S&P 500 uses a market capitalization weighting method, giving a higher percentage allocation to companies with the largest market capitalizations. The market capitalization of a company is calculated by taking the current stock price and multiplying it by the outstanding shares.

Reference:

- https://www.investopedia.com/terms/s/sp500.asp

2. Time series forecasting¶

Time series is a collection of data points collected at constant time intervals. These are analyzed to determine the long term trend so as to forecast the future or perform some other form of analysis. But what makes a TS different from say a regular regression problem? There are 2 things:

It is time dependent. So the basic assumption of a linear regression model that the observations are independent doesn't hold in this case. Along with an increasing or decreasing trend, most TS have some form of seasonality trends, i.e. variations specific to a particular time frame. For example, if you see the sales of a woolen jacket over time, you will invariably find higher sales in winter seasons.

Reference:

- https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/

3. portfolio construction framework¶

Modern portfolio theory (MPT) provides investors with a portfolio construction framework that maximizes returns for a given level of risk, through diversification. MPT reasons that investors should not concern themselves with an individual investment's expected return, but rather the weighted average of the expected returns of a portfolio's component securities as well as how individual securities move together. Markowitz consequently introduced the concept of covariance to quantify this co-movement.

It proposed that investors should instead consider variances of return, along with expected returns, and choose portfolios offering the highest expected return for a given level of variance. These portfolios were deemed "efficient." For given levels of risk, there are multiple combinations of asset classes (portfolios) that maximize expected return. Markowitz displayed these portfolios across a two-dimensional plane showing expected return and standard deviation, which we now call the efficient frontier.

Reference:

- https://www.windhamlabs.com/insights/modern-portfolio-theory/

4. Monte Carlo Simulation¶

Monte Carlo simulations are used to model the probability of different outcomes in a process that cannot easily be predicted due to the intervention of random variables. It is a technique used to understand the impact of risk and uncertainty in prediction and forecasting models.

The technique was first developed by Stanislaw Ulam, a mathematician who worked on the Manhattan Project. After the war, while recovering from brain surgery, Ulam entertained himself by playing countless games of solitaire. He became interested in plotting the outcome of each of these games in order to observe their distribution and determine the probability of winning. After he shared his idea with John Von Neumann, the two collaborated to develop the Monte Carlo simulation.

Reference:

- https://www.investopedia.com/terms/m/montecarlosimulation.asp

5. Machine learning - Moving average¶

Smoothing is a technique applied to time series to remove the fine-grained variation between time steps.

The hope of smoothing is to remove noise and better expose the signal of the underlying causal processes. Moving averages are a simple and common type of smoothing used in time series analysis and time series forecasting.

Calculating a moving average involves creating a new series where the values are comprised of the average of raw observations in the original time series.

Reference:

- https://machinelearningmastery.com/moving-average-smoothing-for-time-series-forecasting-python/

6. Machine learning - Linear regression¶

Linear regression is a very simple approach for supervised learning. Though it may seem somewhat dull compared to some of the more modern algorithms, linear regression is still a useful and widely used statistical learning method. Linear regression is used to predict a quantitative response Y from the predictor variable X.

Linear Regression is made with an assumption that there’s a linear relationship between X and Y.

Reference:

- https://medium.com/simple-ai/linear-regression-intro-to-machine-learning-6-6e320dbdaf06
- https://www.datascience.com/blog/time-series-forecasting-machine-learning-differences

7. Machine learning - K-Nearest - Neighbors¶

K-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression:

In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The k-NN algorithm is among the simplest of all machine learning

algorithms.

Reference:

- https://machinelearningmastery.com/k-nearest-neighbors-for-machine-learning/
- https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761
- https://tomaszgolan.github.io/introduction_to_machine_learning/markdown/introduction_to_machine_learning_01_knn/introduction_to_machine_learning_01_knn/

8. Machine learning - ARIMA¶

An ARIMA model is a class of statistical models for analyzing and forecasting time series data.

It explicitly caters to a suite of standard structures in time series data, and as such provides a simple yet powerful method for making skillful time series forecasts.

ARIMA is an acronym that stands for AutoRegressive Integrated Moving Average. It is a generalization of the simpler AutoRegressive Moving Average and adds the notion of integration.

This acronym is descriptive, capturing the key aspects of the model itself. Briefly, they are:

- AR: Autoregression. A model that uses the dependent relationship between an observation and some number of lagged observations.
- I: Integrated. The use of differencing of raw observations (e.g. subtracting an observation from an observation at the previous time step) in order to make the time series stationary.
- MA: Moving Average. A model that uses the dependency between an observation and a residual error from a moving average model applied to lagged observations.

Each of these components are explicitly specified in the model as a parameter. A standard notation is used of ARIMA(p,d,q) where the parameters are substituted with integer values to quickly indicate the specific ARIMA model being used.

The parameters of the ARIMA model are defined as follows:

- p: The number of lag observations included in the model, also called the lag order.
- d: The number of times that the raw observations are differenced, also called the degree of differencing.
- q: The size of the moving average window, also called the order of moving average.

Reference:

- https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/

9. Machine learning - Prophet¶

Prophet is a procedure for forecasting time series data. It is based on an additive model where non-linear trends are fit with yearly and weekly seasonality, plus holidays. It works best with daily periodicity data with at least one year of historical data. Prophet is robust to missing data, shifts in the trend, and large outliers.

Prophet is open source software released by Facebook's Core Data Science team.

The Prophet procedure is an additive regression model with four main components:

- A piecewise linear or logistic growth curve trend. Prophet automatically detects changes in trends by selecting changepoints from the data.
- A yearly seasonal component modeled using Fourier series.
- A weekly seasonal component using dummy variables.

- A user-provided list of important holidays.

Reference:

- https://research.fb.com/prophet-forecasting-at-scale/

10. Machine learning - Long Short-Term Memory¶

Long Short-Term Memory usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies. LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is their default behavior.

All recurrent neural networks have the form of a chain of repeating modules of a neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer. LSTMs also have this chain like structure, but the repeating module has a different structure. The key to LSTMs is the cell state which is acting like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It’s very easy for information to flow along it unchanged.

Reference:

- https://machinelearningtutorials.com/long-short-term-memory-lstm/

Methodology section¶

This project are separated 4 part of analysis from data exploration, visualization, correlation and monthly return for data extraction by mathematical programming, portfolio optimization and machine learning.

The following strategic analysis as follow:

Part 1 - Data Exploration (Getting 500 stocks data)¶

1. To use **Beautifulsoup** grap stocks symbol for S&P 500
2. To use API in the function of **"fix_yahoo_finance"** to load all dataset for 500 stocks share from January 2008 until Now, each stocks symbol will be used to load their own CSV dataset file.
3. To select each stocks of adjusted closing price and use the **"join"** function and rename the columns to create the joined CSV
4. To use both **"plotly"** and **"matplotlib"** for data visualization, "iplot" can be used to compare the detail price in certain period of time.

Part 2 - Higher monthly return (Choosing top 10 stocks correlation and monthly return with index from 500 stocks data)¶

1. To use **corr()** function to find out the **top 30 stocks** which are higher correlation with S&P 500 index from **500 stocks share**.
2. The Last part of this project will conduct the prediction of machine learning for S&P 500 index, so the correlation between stocks and index is important reference for the trend affected by index fluctuation. And it will provide the buy and sell signal when the prediction are generated.

3. After getting the top 30 stocks of highest correlation with index, we will calculate and **compare the higher monthly return for this 30 stocks**.
4. To calculate the **top 10 stocks of highest monthly return from the above 30**. we will sort and extract the joined dataset for the next part of Portfolio optimization.

Part 3 - Portfolio Optimization (Testing top 10 stocks from part 2 for portfolio optimization)¶

1. After getting the top 10 stocks from part 2 by using Highest correlation and monthly return without consideration the factor of risk, but this stage we would like to use the portfolio optimization method to decide the investment strategic.
2. Calculating the **top 10 higher monthly return** of stocks share as Medium or long term investment
3. Using **portfolio optimization** to calculate the **top 10 higher monthly return** of stocks share as Medium or long term investment which are higher monthly return and lower risk as investment strategic.

Part 4 - Machine Learning¶

After portfolio optimization analysis, it assume we get the proportion of investment strategic, we will expect to know if it is time to invest this moment or when is the best momnent for buying or selling. Therefore, the machine learning for S&P 500 index will be conducted in machine learning process by comparing different methods and model.

The command process as follow:

```
- Dropping NAN with dropna in **pandas**
- splitting into train and validation
- Measuring root mean square error (RMSE) for the standard deviation of residuals
- Plot the prediction
```

1. Moving average
2. Linear Regression
    - Using the function of **linear_model** in **sklearn**
3. k-Nearest Neighbours
    - Using the function of **neighbors, GridSearchCV, MinMaxScaler** in **sklearn** to find the best parameter
4. Auto ARIMA
    - Using the function of **auto_arima** in **pmdarima** which automatically selects the best combination of (p,q,d) that provides the least error.
5. Prophet
    - Using the function of **Prophet in fbprophet** which Prophet, designed by Facebook, is a time series forecasting library that The input for Prophet is a dataframe with two columns: date and target.
6. Long Short Term Memory
    - Using the function of **MinMaxScaler in sklearn** and **LSTM in keras** to create and fit the LSTM network

Loading libraries¶

In [1]:

```
# Loading libraries

from pandas_datareader import data as pdr
import pandas as pd
import matplotlib.pyplot as plt
import fix_yahoo_finance as yf
from matplotlib import style
import datetime as dt
import numpy as np
import requests
from collections import Counter
from sklearn.model_selection import cross_val_score
from sklearn import svm, neighbors
from sklearn.ensemble import VotingClassifier, RandomForestClassifier
import bs4 as bs
import pickle
import os

import plotly.plotly as py
import plotly.graph_objs as go

#from datetime import datetime
from plotly.offline import download_plotlyjs, init_notebook_mode, iplot
import itertools
init_notebook_mode()

#display tabulate table
from IPython.display import HTML, display
import tabulate
/home/jim/anaconda3/lib/python3.6/site-packages/sklearn/ensemble/weight_boosting.py:29:
DeprecationWarning: numpy.core.umath_tests is an internal NumPy module and should not
be imported. It will be removed in a future NumPy release.
  from numpy.core.umath_tests import inner1d
```

Experimentation and Results¶

Definition of data collection method¶

Part 1 - Data Exploration¶

In order to **get the total 500 stocks shares from January 2008 until Now around 10 years data**. The BeautifulSoup method has been used for scrapying the dataset in web of Nasdaq (http://www.nasdaq.com/symbol/), but it found that the "id" will be changed automatically in certain period, it is not a stable method to get the dataset in our research project, the API of yahoo finance will be the best way to load large range of data.

Getting the stocks symbol¶

- Stock Symbols are scraping by **"BS4 - BeautifulSoup" in Web of Wiki (http://en.wikipedia.org/wiki/)**
- Adding the S&P index symbol into the list for measuring the correlation of stocks share.

Getting the data¶

- Before we analyze stock data, we need to get it into some workable format. Stock data can be obtained from Yahoo! Finance, Google Finance, or a number of other sources. These days I recommend **getting data from Yahoo! Finance**, a provider of community-maintained financial and economic data. Yahoo! Finance used to be the go-to source for good quality stock data.
- Then to **input stocks symbols scraped in API in the function of "fix_yahoo_finance", it loads all dataset for 500 stocks share from January 2008 until Now.**

Detail of dataset¶

- **Open** is the price of the stock at the beginning of the trading day (it need not be the closing price of the previous trading day), **high** is the highest price of the stock on that trading day, **low** the lowest price of the stock on that trading day, and **close** the price of the stock at closing time. **Volume** indicates how many stocks were traded. **Adjusted prices** (such as the adjusted close) is the price of the stock that adjusts the price for corporate actions. While stock prices are considered to be set mostly by traders, stock splits (when the company makes each extant stock worth two and halves the price) and dividends (payout of company profits per share) also affect the price of a stock and should be accounted for.

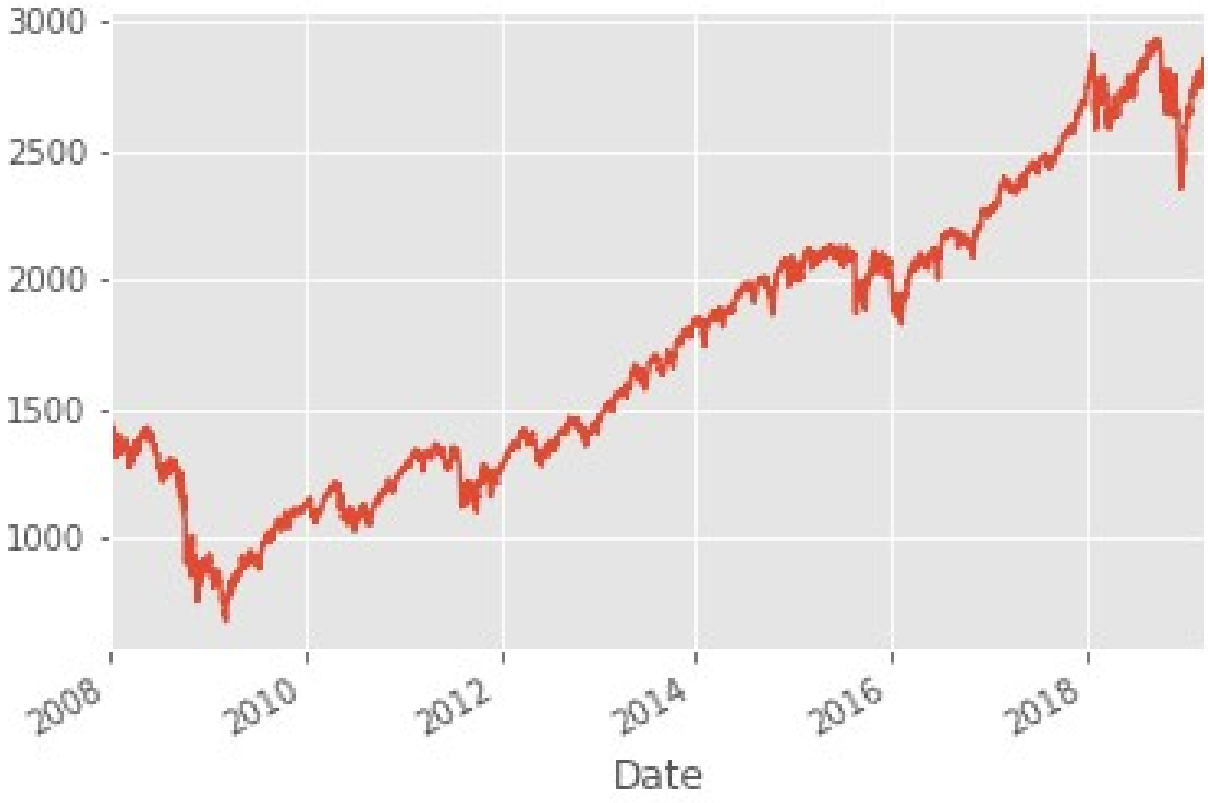
Visualization¶

- We have stock data we would like to visualize it. I will use the **matplotlib** package. And The **plotly** will also be used for visualization for comparing the trend between stocks and index. It will be more easy and clear if many stocks share are plotted at the same graph.

In [ ]:

In [2]:

```
style.use('ggplot')

start = dt.datetime(2008, 1, 1)
end = dt.datetime.now()

df = yf.download('^GSPC',start, end)
#df = yf.download('TSLA',start, end)
df.Close.plot()
plt.show()
[**********************100%**********************]  1 of 1 downloaded
```



In [7]:

```
display(df.tail())
```

	Open	High	Low	Close	Adj Close	Volume
Date						
2019-03-19	2840.760010	2852.419922	2823.270020	2832.570068	2832.570068	3620220000
2019-03-20	2831.340088	2843.540039	2812.429932	2824.229980	2824.229980	3771200000
2019-03-21	2819.719971	2860.310059	2817.379883	2854.879883	2854.879883	3546800000
2019-03-22	2844.520020	2846.159912	2800.469971	2800.709961	2800.709961	4237200000
2019-03-25	2796.010010	2809.790039	2785.020020	2796.929932	2796.929932	1321519941

Description of data¶

Data Preparation¶

To get all company pricing data in the S&P 500, and now we're going to pull stock pricing data on all of them.

Process for data preparation:

1. Using "bs" function to extract the symbol of S&P 500
2. Removing the error symbol or ticket
3. Adding the "^GSPC" index of S&P 500 for comparation purpose

In [8]:

```
def list_print(list_data):
    #display(HTML(tabulate.tabulate(list_data, tablefmt='html')))
    display(pd.DataFrame(list_data.items(), columns=["name", "symbol"]) )
```

In [9]:

```
# Scrapying stocks symbol

def save_sp500_tickers():
    resp = requests.get('http://en.wikipedia.org/wiki/List_of_S%26P_500_companies')
    soup = bs.BeautifulSoup(resp.text, 'lxml')
    table = soup.find('table', {'class': 'wikitable sortable'})
    some_dict = {}
    tickers = ['^GSPC'] # Add S&P index for comparsion
    for row in table.findAll('tr')[1:]:
        dictionary = row.findAll('td')[0].text
        ticker = row.findAll('td')[1].text
        #tickers.append(ticker)
        #tickers = [x for x in tickers if not '.' in x] # remove wrong tickers in list

        some_dict[dictionary] = ticker


    with open("sp500tickers1.pickle","wb") as f:
        pickle.dump(some_dict,f)

    #return some_dict
    #return display(some_dict)
    return list_print(some_dict)

save_sp500_tickers()
```

	name	symbol
0	3M Company	MMM
1	Abbott Laboratories	ABT
2	AbbVie Inc.	ABBV
3	ABIOMED Inc	ABMD
4	Accenture plc	ACN
5	Activision Blizzard	ATVI
6	Adobe Systems Inc	ADBE
7	Advanced Micro Devices Inc	AMD
8	Advance Auto Parts	AAP
9	AES Corp	AES
10	Affiliated Managers Group Inc	AMG
11	AFLAC Inc	AFL
12	Agilent Technologies Inc	A
13	Air Products & Chemicals Inc	APD
14	Akamai Technologies Inc	AKAM
15	Alaska Air Group Inc	ALK
16	Albemarle Corp	ALB
17	Alexandria Real Estate Equities	ARE
18	Alexion Pharmaceuticals	ALXN
19	Align Technology	ALGN
20	Allegion	ALLE
21	Allergan, Plc	AGN
22	Alliance Data Systems	ADS
23	Alliant Energy Corp	LNT
24	Allstate Corp	ALL
25	Alphabet Inc Class A	GOOGL
26	Alphabet Inc Class C	GOOG
27	Altria Group Inc	MO
28	Amazon.com Inc.	AMZN
29	Ameren Corp	AEE
...	...	...
475	Viacom Inc.	VIAB
476	Visa Inc.	V
477	Vornado Realty Trust	VNO
478	Vulcan Materials	VMC
479	Wabtec Corporation	WAB
480	Walmart	WMT
481	Walgreens Boots Alliance	WBA

	name	symbol
482	The Walt Disney Company	DIS
483	Waste Management Inc.	WM
484	Waters Corporation	WAT
485	Wec Energy Group Inc	WEC
486	WellCare	WCG
487	Wells Fargo	WFC
488	Welltower Inc.	WELL
489	Western Digital	WDC
490	Western Union Co	WU
491	WestRock	WRK
492	Weyerhaeuser	WY
493	Whirlpool Corp.	WHR
494	Williams Cos.	WMB
495	Willis Towers Watson	WLTW
496	Wynn Resorts Ltd	WYNN
497	Xcel Energy Inc	XEL
498	Xerox	XRX
499	Xilinx	XLNX
500	Xylem Inc.	XYL
501	Yum! Brands Inc	YUM
502	Zimmer Biomet Holdings	ZBH
503	Zions Bancorp	ZION
504	Zoetis	ZTS

505 rows × 2 columns

After getting the tickers:

1. Then it use datetime to specify dates from 2008 until now for the Pandas datareader, os is to check for, and create, directories.
2. The datasets for each stocks are downloaded by the function of Pandas datareader and use "to_csv" to generate each dataset csv files.

In [10]:

```
# save_sp500_tickers()
def get_data_from_yahoo(reload_sp500=False):
    if reload_sp500:
        tickers = save_sp500_tickers()
        tickers = list(iter(tickers.values()))
        tickers = [x for x in tickers if not '.' in x]
        tickers = ['^GSPC'] + tickers
    else:
        with open("sp500tickers1.pickle", "rb") as f:
```

```
            tickers = pickle.load(f)
            tickers = list(iter(tickers.values()))
            tickers = [x for x in tickers if not '.' in x]
            tickers = ['^GSPC'] + tickers
    if not os.path.exists('stock_dfs'):
        os.makedirs('stock_dfs')

    start = dt.datetime(2008, 1, 1)
    end = dt.datetime.now()
    for ticker in tickers:
        # just in case your connection breaks, we'd like to save our progress!
        if not os.path.exists('stock_dfs/{}.csv'.format(ticker)):
            df = yf.download(ticker,start, end)

            #df = web.DataReader(tickers, 'morningstar', start, end)

            #df.reset_index(inplace=True)
            #df.set_index("Date", inplace=True)
            #df = df.drop("Symbol", axis=1)
            df.to_csv('stock_dfs/{}.csv'.format(ticker))
        else:
            print('Already have {}'.format(ticker))


get_data_from_yahoo()
```

Compiling all the CSV stock data files by using the adjusted close price and rename the columns to each stock tickers and export to a "Joined" csv file.

In [11]:

```
def compile_data():
    with open("sp500tickers1.pickle", "rb") as f:
        tickers = pickle.load(f)
        tickers = list(iter(tickers.values()))
        tickers = [x for x in tickers if not '.' in x]
        tickers = ['^GSPC'] + tickers

    main_df = pd.DataFrame()

    for count, ticker in enumerate(tickers):
        df = pd.read_csv('stock_dfs/{}.csv'.format(ticker))
        df.set_index('Date', inplace=True)

        df.rename(columns={'Adj Close': ticker}, inplace=True)
        df.drop(['Open', 'High', 'Low', 'Close', 'Volume'], 1, inplace=True)
```

```
        if main_df.empty:
            main_df = df
        else:
            main_df = main_df.join(df, how='outer')
            #main_df = main_df.dropna()

        if count % 10 == 0:
            print(count)
    display(main_df.tail())
    #return main_df
    main_df.to_csv('sp500_joined_closes.csv')


compile_data()
```

	^GSPC	MMM	ABT	ABBV	ABMD	ACN	ATVI	
Date								
2019-03-14	2808.479980	207.380005	78.980003	79.949997	325.609985	164.820007	43.320000	267.690002
2019-03-15	2822.479980	208.080002	79.860001	81.339996	333.230011	166.389999	44.630001	257.089996
2019-03-18	2832.939941	208.490005	78.830002	80.650002	333.250000	166.429993	44.970001	257.760010
2019-03-19	2832.570068	209.020004	79.959999	80.820000	339.359985	166.199997	46.419998	260.420013
2019-03-20	2824.229980	208.300003	79.940002	80.430000	339.660004	166.270004	46.160000	259.739990

5 rows × 504 columns

We collected the total 504 number of stock shares with S&P 500 index value from January 2008 until Now (03-14-2019)

Below is a iplot of the comparation stock shares with S&P 500 index.¶

We considered if it plot all 500 stock shares, it can't appear a meaningful comparation graph. Initially, we view the first 5 column including S&P 500 index in the time series from 2008 to 2019 (Now). We will further see the top 10 monthly return, top 10 for higher correlation with S&P 500 index, and top 10 investing stocks strategic from the portfolio optimization in our data time series.

The **offline plotly** has been used for visualization instead of using "matplotlib" function to plot the graph, because it include too many stocks share and also the higher closing price is not equal to earn higher return in investment. It is not a valuable comparing method.

In [12]:

```
def iplot_stocks(df, n):
    data = []
```

```
    tracex = []

    stock_list = list(df.columns.values)
    top5 = itertools.islice(stock_list, n) # grab the first five elements
    citynames = list (top5)

    #citynames = list(df.columns.values)
    #citynames = ['AAPL', 'MMM', 'ABT']


    for i in range(0, len(citynames)):
        tracex =  go.Scatter(
              x=df.index,
              y=df[citynames[i]],
              name=citynames[i])
        data.append(tracex)


    layout = {
        'title' : 'S&P 500 - stocks comparation',
        'xaxis' : {'title' : 'time-series (Date)'},
        'yaxis' : {'title' : 'Adjusted close price (USD)'},
        'showlegend': True,
        #'shapes': shps
    }

    fig = go.Figure(data=data, layout=layout)

    iplot(fig)


df = pd.read_csv('sp500_joined_closes.csv', index_col='Date')

# plot 5 number of stock
iplot_stocks(df, 5)
```

If only checking the adjusted closing price on the graph, it does not get any trend or prediction result. Therefore, the following parts will conduct portfolio optimization and machine learning analysis for getting the best strategic to have a highest return investment.

Correlation¶

Does it have strong relation with S&P 500 index fluctation when we invest some stocks?

Most people think that they will get higher return when the main index going up, so we can calculate the correlation

to check which stocks share are strong relationship and low relationship.

In [13]:

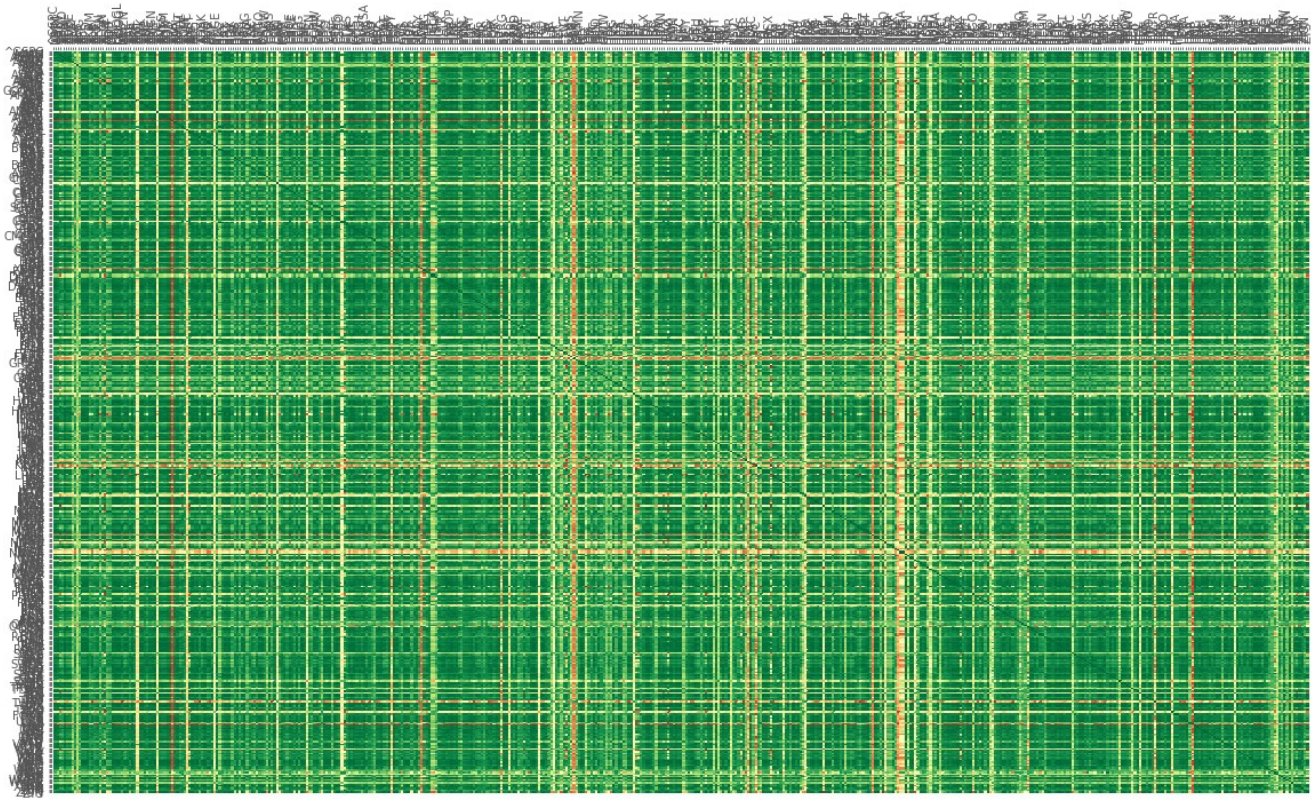
```
df = pd.read_csv('sp500_joined_closes.csv')
#df['AAPL'].plot()
#plt.show()
df_corr = df.corr()
display(df_corr.head())
df_corr.to_csv('sp500corr.csv')
```

	^GSPC	MMM	ABT	ABBV	ABMD	ACN	ATVI	ADBE	AMD	AAP	..
^GSPC	1.000000	0.980137	0.958351	0.951743	0.831823	0.972825	0.903198	0.903110	0.560187	0.827187	...
MMM	0.980137	1.000000	0.936665	0.929043	0.813548	0.972820	0.920370	0.891991	0.548472	0.801029	...
ABT	0.958351	0.936665	1.000000	0.894654	0.883414	0.963480	0.867738	0.933754	0.598889	0.795015	...
ABBV	0.951743	0.929043	0.894654	1.000000	0.887700	0.934496	0.908915	0.921301	0.750295	0.217340	...
ABMD	0.831823	0.813548	0.883414	0.887700	1.000000	0.894063	0.907289	0.978790	0.790333	0.570807	...

5 rows × 504 columns

In [14]:

```
def visualize_data(df_corr):

    data1 = df_corr.values
    fig1 = plt.figure(figsize=(20,10))
    ax1 = fig1.add_subplot(111)

    heatmap1 = ax1.pcolor(data1, cmap=plt.cm.RdYlGn)
    fig1.colorbar(heatmap1)

    ax1.set_xticks(np.arange(data1.shape[1]) + 0.5, minor=False)
    ax1.set_yticks(np.arange(data1.shape[0]) + 0.5, minor=False)
    ax1.invert_yaxis()
    ax1.xaxis.tick_top()
    column_labels = df_corr.columns
    row_labels = df_corr.index
    ax1.set_xticklabels(column_labels)
    ax1.set_yticklabels(row_labels)
    plt.xticks(rotation=90)
    heatmap1.set_clim(-1,1)
    plt.tight_layout()
    #plt.savefig("correlations.png", dpi = (300))
    plt.show()

visualize_data(df_corr)
```



Part 2_1 - Most higher Correlation (Sort out top 30 stocks from all 500 shares)¶

In [18]:

```
# Vertical sort the maximium correlation and find out the large correlation for 30 tickers
df_corr = df_corr.sort_values(by='^GSPC', ascending=False)


# Vertical sort the maximium correlation and find out the large correlation for 30 tickers
df1_corr =df_corr[df_corr.iloc[0,:].sort_values(ascending=False).index]


df1_corr_max = df1_corr.iloc[0:31, 0:31]
df1_corr_max
```

Out[18]:

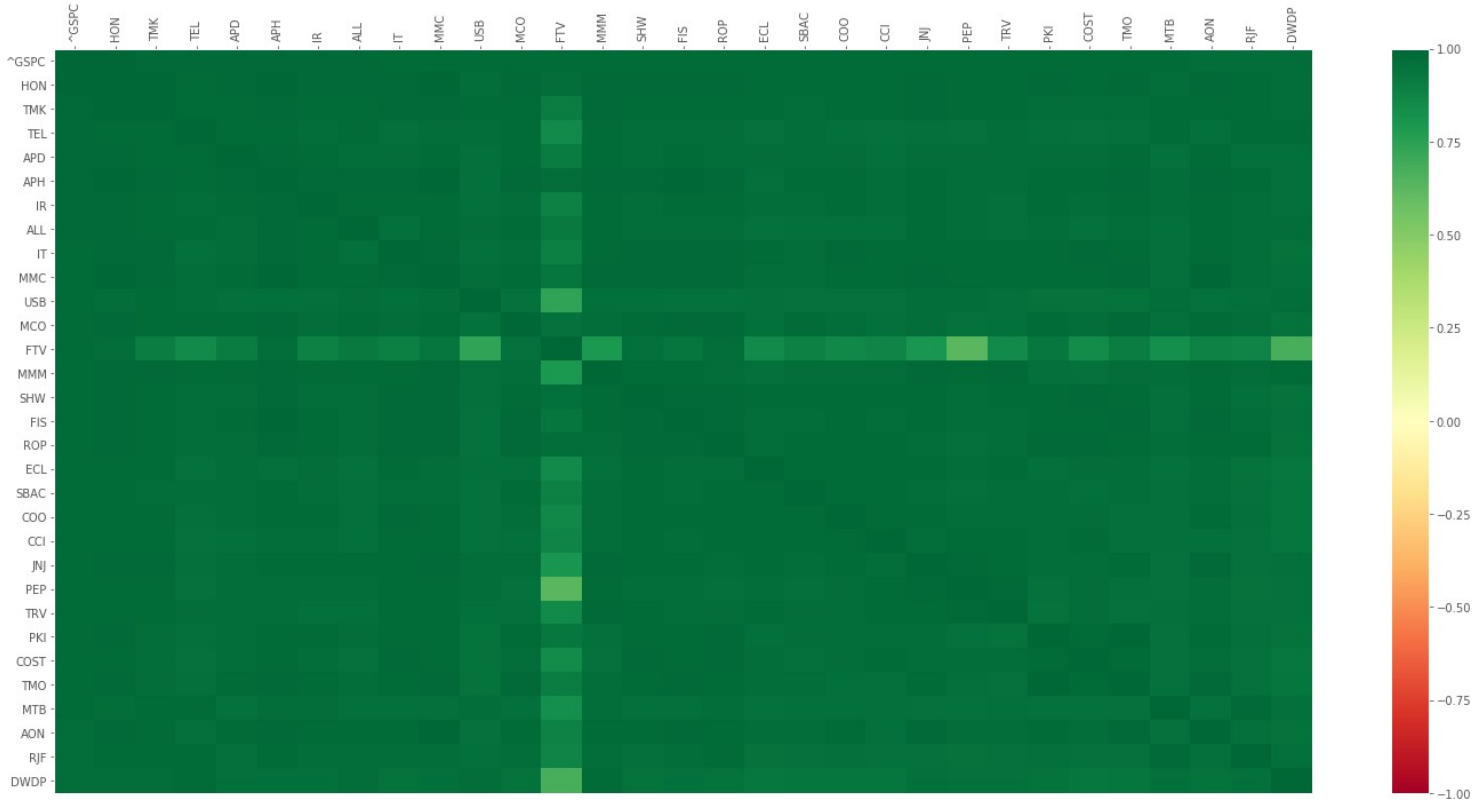
	^GSPC	HON	TMK	TEL	APD	APH	IR	ALL	IT	MMC	..
^GSPC	1.000000	0.992551	0.990752	0.987183	0.986700	0.985939	0.985614	0.984788	0.983039	0.982700	...
HON	0.992551	1.000000	0.992314	0.981840	0.985558	0.993498	0.987669	0.984502	0.989827	0.992853	...

	^GSPC	HON	TMK	TEL	APD	APH	IR	ALL	IT	MMC	...
TMK	0.990752	0.992314	1.000000	0.984330	0.977829	0.988218	0.982516	0.981037	0.985342	0.988789	...
TEL	0.987183	0.981840	0.984330	1.000000	0.978567	0.982212	0.971018	0.979026	0.966386	0.972212	...
APD	0.986700	0.985558	0.977829	0.978567	1.000000	0.984880	0.978630	0.975919	0.974812	0.982543	...
APH	0.985939	0.993498	0.988218	0.982212	0.984880	1.000000	0.986640	0.984509	0.985208	0.992338	...
IR	0.985614	0.987669	0.982516	0.971018	0.978630	0.986640	1.000000	0.976646	0.981976	0.983003	...
ALL	0.984788	0.984502	0.981037	0.979026	0.975919	0.984509	0.976646	1.000000	0.967029	0.980318	...
IT	0.983039	0.989827	0.985342	0.966386	0.974812	0.985208	0.981976	0.967029	1.000000	0.988480	...
MMC	0.982700	0.992853	0.988789	0.972212	0.982543	0.992338	0.983003	0.980318	0.988480	1.000000	...
USB	0.982462	0.971951	0.984264	0.969818	0.961574	0.965808	0.968632	0.975280	0.964901	0.970932	...
MCO	0.981795	0.985110	0.976818	0.977999	0.978058	0.986988	0.971213	0.978389	0.975884	0.981719	...
FTV	0.981125	0.976162	0.907095	0.858886	0.919101	0.975810	0.893746	0.928071	0.905381	0.941682	...
MMM	0.980137	0.987109	0.985588	0.981707	0.981585	0.987476	0.976934	0.978998	0.977108	0.987037	...
SHW	0.979812	0.988960	0.981105	0.970346	0.974533	0.987504	0.974974	0.974846	0.988430	0.987458	...
FIS	0.979797	0.990415	0.983944	0.969427	0.982011	0.992452	0.983156	0.973233	0.988588	0.991964	...
ROP	0.979531	0.989872	0.980541	0.971068	0.971472	0.989255	0.978072	0.969524	0.987521	0.985362	...
ECL	0.979457	0.978053	0.978224	0.959286	0.969070	0.967221	0.970935	0.956306	0.980032	0.974473	...
SBAC	0.979449	0.977689	0.974652	0.971935	0.971130	0.976912	0.970338	0.966398	0.974487	0.973221	...
COO	0.979352	0.982451	0.978848	0.964065	0.970559	0.978365	0.980316	0.961650	0.986059	0.981490	...
CCI	0.979185	0.983028	0.977328	0.959417	0.965532	0.975970	0.970743	0.962469	0.981125	0.976928	...
JNJ	0.978768	0.987492	0.985150	0.964319	0.974861	0.983040	0.977503	0.977450	0.981799	0.990755	...
PEP	0.978457	0.980724	0.979975	0.960204	0.974327	0.973466	0.970111	0.969653	0.978647	0.982371	...
TRV	0.978238	0.982194	0.983483	0.970417	0.973162	0.975849	0.964918	0.966578	0.979501	0.980835	...
PKI	0.977969	0.985693	0.971029	0.962467	0.972779	0.984356	0.981583	0.972025	0.978853	0.978652	...
COST	0.977373	0.983596	0.973368	0.955382	0.97	0.976731	0.968867	0.960830	0.986291	0.979013	...
TMO	0.977149	0.987267	0.972418	0.962758	0.979067	0.985992	0.980992	0.973192	0.978841	0.984912	...
MTB	0.976568	0.974518	0.984323	0.978946	0.957352	0.975295	0.974101	0.966342	0.967230	0.968563	...
AON	0.976367	0.989121	0.979664	0.963727	0.979605	0.989261	0.984156	0.976761	0.983894	0.994783	...
RJF	0.976210	0.977572	0.982986	0.981204	0.961293	0.984196	0.974423	0.970924	0.971340	0.973676	...
DWDP	0.975912	0.969507	0.975121	0.979873	0.966538	0.968163	0.962707	0.968932	0.953095	0.963023	...

31 rows × 31 columns

In [19]:

```
# check the maximium correlation stock share

visualize_data(df1_corr_max)
```



In [20]:

```
# list out the most higher correlation with S&P 500 index

corr_list = list(df1_corr_max.columns.values)
corr_list
```

Out[20]:

```
['^GSPC',
 'HON',
 'TMK',
 'TEL',
 'APD',
 'APH',
 'IR',
 'ALL',
 'IT',
 'MMC',
 'USB',
 'MCO',
 'FTV',
 'MMM',
 'SHW',
 'FIS',
 'ROP',
 'ECL',
```

```
 'SBAC',
 'COO',
 'CCI',
 'JNJ',
 'PEP',
 'TRV',
 'PKI',
 'COST',
 'TMO',
 'MTB',
 'AON',
 'RJF',
 'DWDP']
```

We create a new dataframe for higher correlation stocks share with S&P 500 index to further analysis.

We can believe if choosing the above stocks share for investment, the S&P index will be a higher relative reference.

In [21]:

```
df = pd.read_csv('sp500_joined_closes.csv', index_col='Date')
#df.head()

new_df = df[corr_list]
new_df.head()
```

Out[21]:

	^GSPC	HON	TMK	TEL	APD	APH	IR	ALL	IT
Date									
2008-01-02	1447.160034	43.065563	22.015690	28.644402	65.117157	20.039412	28.689632	38.720806	17.139999
2008-01-03	1447.160034	43.281273	22.256193	28.574295	65.710472	19.921530	28.293470	38.629963	16.830000
2008-01-04	1411.630005	41.900867	22.082300	27.670639	65.009262	19.459085	27.226398	38.667816	15.980000
2008-01-07	1416.180054	41.757069	22.592916	26.689072	65.184586	19.427345	26.242384	38.698090	16.510000
2008-01-08	1390.189941	41.045311	21.997194	25.645185	63.600128	18.747278	25.545910	38.228752	16.790001

5 rows × 31 columns

In [22]:

```
# Here we pick all 30 stocks in iplot, that will be more valuable for comparion.

iplot_stocks(new_df, 30)
```

Assuming we pick the tickers which are the most higher correlation with S&P 500 index (^GSPC), the trend of machine learning will be meaningful and significate for prediction

If the other stock shares which is less correlation with S&P 500 index, it is no prediction value when we build the machine learning model for S&P 500 index.

Part 2_2 - The highest monthly return¶

A “better” solution, though, would be to plot the information we actually want: the stock’s returns. This involves transforming the data into something more useful for our purposes. There are multiple transformations we could apply.

One transformation would be to consider the stock’s return since the beginning of the period of interest.

$$\begin{align} \text{return}_{t,0} = \frac{\text{price}_t}{\text{price}_0} \end{align}$$

In [31]:

```
# new_df - dataset has been selected as top 30 higher correlation with S&P index

df2 = pd.read_csv('sp500_joined_closes.csv')
df2.set_index(pd.DatetimeIndex(df2["Date"]), inplace = True)
# dropping columns
df2.drop(["Date"], axis = 1, inplace = True)

df2 = df2[corr_list]
#print (df2.head())
```

I am using a lambda function, which allows me to pass a small function defined quickly as a parameter to another function or method

In [32]:

```
# Monthly price
by_month = df2.resample('M').agg(lambda x: x[-1])
#print(by_month.head())

# Calculate the monthly percentage changes
monthly_pc = by_month / by_month.shift(1) - 1
# monthly_pc[:5]
#print(monthly_pc.tail())

# Calculate the monthly cumulative returns
monthly_cr = (1 + monthly_pc).cumprod()
#monthly_cr[:5]
#monthly_cr.tail()

# Sort the maximium return at last column and find out the large returns for 10 tickers
monthly_sort =monthly_cr[monthly_cr.iloc[-1,:].sort_values(ascending=False).index]
df_max = monthly_sort.iloc[:, :10]
```

```
df_max.tail(10)
```

Out[32]:

	IT	SHW	COO	SBAC	ROP	MCO	TMO	APH	FIS	COST
Date										
2018-06-30	8.961564	8.374486	6.032505	5.576494	5.252970	5.731960	4.165654	4.780397	5.178272	4.000745
2018-07-31	9.132164	9.055837	6.675123	5.344478	5.756390	5.750779	4.716476	5.129259	5.036642	4.187018
2018-08-31	10.098449	9.379355	6.554177	5.242485	5.689083	5.998048	4.808380	5.187952	5.282784	4.474156
2018-09-30	10.687795	9.371737	7.101767	5.424856	5.647898	5.633489	4.911952	5.169493	5.342351	4.507741
2018-10-31	9.947404	8.100652	6.619007	5.476866	5.401586	4.901674	4.702055	4.920970	5.098915	4.387793
2018-11-30	10.329737	8.748771	7.144816	5.768659	5.682071	5.375525	5.022033	4.835197	5.287492	4.449202
2018-12-31	8.620364	8.117271	6.521378	5.467409	5.088832	4.732435	4.506801	4.466833	5.038099	3.918829
2019-01-31	9.163183	8.696162	7.143824	6.164471	5.417772	5.356600	4.947432	4.847247	5.135373	4.128901
2019-02-28	9.595415	8.960154	7.329108	6.097940	6.190291	5.867519	5.227358	5.180798	5.313217	4.219332
2019-03-31	9.875253	8.878660	7.462882	6.504897	6.325516	6.007496	5.353326	5.261846	5.261552	4.571168

Using 10 years cumulative return for selecting higher return, it may be some problem, like it does not reflect the trend of industry, because new industry may have large return only these few years, it may use only 3 years cumulative return, it would be more realistic. But now in research level, we keep using 10 years sum of return.

In [33]:

```
iplot_stocks(df_max, 10)
```

Apart from the risk consideration, the percentage of increasing return can be 9.8 times over 10 year if just only invest "IT" tickers of stock. Of course, that can be a part of investment strategic we can notice.

In [34]:

```
monthly_list = list(df_max.columns.values)
print (monthly_list)
['IT', 'SHW', 'COO', 'SBAC', 'ROP', 'MCO', 'TMO', 'APH', 'FIS', 'COST']
```

In [35]:

```
# List out the full name of company which are top 10 higher cumulative monthly return
over 10 years and it is higher correlation with S&P index.
def search(a):
```

```
    with open("sp500tickers1.pickle", "rb") as f:
        some_dict = pickle.load(f)

    for i in a:
        search_name = i
        for name, symbol in some_dict.items():
            if symbol == search_name:
                print(name)
```

```
search(monthly_list)
Gartner Inc
Sherwin-Williams
The Cooper Companies
SBA Communications
Roper Technologies
Moody's Corp
Thermo Fisher Scientific
Amphenol Corp
Fidelity National Information Services
Costco Wholesale Corp.
```

Part 3 - portfolio optimization¶

“Modern Portfolio Theory (MPT), a hypothesis put forth by Harry Markowitz in his paper “Portfolio Selection,” (published in 1952 by the Journal of Finance) is an investment theory based on the idea that risk-averse investors can construct portfolios to optimize or maximize expected return based on a given level of market risk, emphasizing that risk is an inherent part of higher reward. It is one of the most important and influential economic theories dealing with finance and investment.

In order to choose from 10 number of stocks share from the result of part 1 which are higher monthly return and lower risk as our final medium or long term investment strategic.

We have 10 stocks in our portfolio. One decision we have to make is how we should allocate our budget to each of stock in our portfolio. If our total budget is 1, then we can decide the weights for each stock, so that the sum of weights will be 1. And the value for weights will be the portion of budget we allocate to a specific stock. For example, if weight is 0.5 for Amazon, it means that we allocate 50% of our budget to Amazon.

“portfolio_annualised_performance” will calculate the returns and volatility, and to make it as an annualised calculation into 252 trading days in one year. “random_portfolios” function will generate portfolios with random weights assigned to each stock, and by giving num_portfolios argument, you can decide how many random portfolios you want to generate.

In [36]:

```
# The new dataframe in daily adjusted closing price are created for top 10 stocks share by above analysis.
```

```
#monthly_list1 = ['^GSPC'] + monthly_list
monthly_list1 = monthly_list
table = df[monthly_list1]
table.head()
```

Out[36]:

	IT	SHW	COO	SBAC	ROP	MCO	TMO	APH	FIS	
Date										
2008-01-02	17.139999	47.881672	37.653690	33.139999	56.963032	29.759460	53.773052	20.039412	19.284296	51
2008-01-03	16.830000	47.856274	38.030224	32.290001	56.580235	29.088890	54.939102	19.921530	19.389524	51
2008-01-04	15.980000	45.790310	38.099590	31.170000	53.872585	28.214615	53.907955	19.459085	19.069073	50
2008-01-07	16.510000	47.001087	37.554592	30.330000	50.791504	29.105852	54.852356	19.427345	18.466436	50
2008-01-08	16.790001	45.714100	37.564507	29.559999	48.606705	28.265533	54.813820	18.747278	18.203388	49

Firstly I download daily price data for each of the stocks in the portfolio, and convert daily stock prices into daily returns. And then it need to calculate annualised portfolio return and annualised portfolio volatility.

In [45]:

```
def portfolio_annualised_performance(weights, mean_returns, cov_matrix):
    #convert daily stock prices into daily returns
    returns = np.sum(mean_returns*weights ) *252
    std = np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights))) * np.sqrt(252)
    return std, returns

def random_portfolios(num_portfolios, mean_returns, cov_matrix, risk_free_rate):
    results = np.zeros((3,num_portfolios))
    weights_record = []
    xrange=range
    for i in xrange(num_portfolios):
        weights = np.random.random(10)
        weights /= np.sum(weights)
        weights_record.append(weights)
        portfolio_std_dev, portfolio_return = portfolio_annualised_performance(weights,
mean_returns, cov_matrix)
        results[0,i] = portfolio_std_dev
        results[1,i] = portfolio_return
        results[2,i] = (portfolio_return - risk_free_rate) / portfolio_std_dev
    return results, weights_record
```

Portfolio standard deviation

The first is the calculation for portfolio's volatility in "portfolio_annualised_performance" function.

$$\sigma_{portfolio} = \sqrt{w_1^2\sigma_1^2 + w_2^2\sigma_2^2 + 2w_1w_2Cov_{1,2}}$$

This formula can be simplified if we make use of matrix notation.

$$\sigma_p^2 = \begin{bmatrix} w_1 & w_2 \end{bmatrix} \begin{bmatrix} \sigma_1^2 & \sigma_{1,2} \\ \sigma_{2,1} & \sigma_2^2 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} w_1\sigma_1^2 + w_2\sigma_{2,1} & w_1\sigma_{1,2} + w_2\sigma_2^2 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$$
$$= w_1^2\sigma_1^2 + w_1w_2\sigma_{2,1} + w_1w_2\sigma_{1,2} + w_2^2\sigma_2^2$$
$$= w_1^2\sigma_1^2 + 2w_1w_2\sigma_{1,2} + w_2^2\sigma_2^2$$

For matrix calculation, we get the part inside the square root in the original formula. Same as the annualised return, I took into account of 252 trading days to calculate the annualised standard deviation of a portfolio.

Sharpe ratio

For the Sharpe ratio, Risk-adjusted return refines an investment's return by measuring how much risk is involved in producing that return, which is generally expressed as a number or rating. There could be a number of different methods of expressing risk-adjusted return, and the Sharpe ratio is one of them.

The ratio describes how much excess return you are receiving for the extra volatility that you endure for holding a riskier asset. The Sharpe ratio can be expressed in below formula.

$$= \frac{\bar{r}_p - r_f}{\sigma_p}$$

Where:

$\bar{r}_p$ = Expected porfolio return

r_f = Risk free rate

σ_p = Portfolio standard deviation

I can get daily returns by calling **pct_change** on the data frame with the price data. And the **mean daily returns**, the covariance matrix of returns are needed to **calculate portfolio returns and volatility**. We will generate **25,000 random portfolios**. Finally, the risk-free rate has been taken from U.S. Department of The Treasury. The rate of **1.78% is the 52week treasury bill rates** at the start of 2018.

In [46]:

```
returns = table.pct_change()
mean_returns = returns.mean()
cov_matrix = returns.cov()
num_portfolios = 25000
risk_free_rate = 0.0178
```

It generates random portfolio and gets the results (portfolio returns, portfolio volatility, portfolio Sharpe ratio) and weights for the corresponding result.

Then by locating the one with the highest Sharpe ratio portfolio, it displays maximum Sharpe ratio portfolio as red star sign.

And does similar steps for minimum volatility portfolio, and displays it as a green star on the plot.

In [47]:

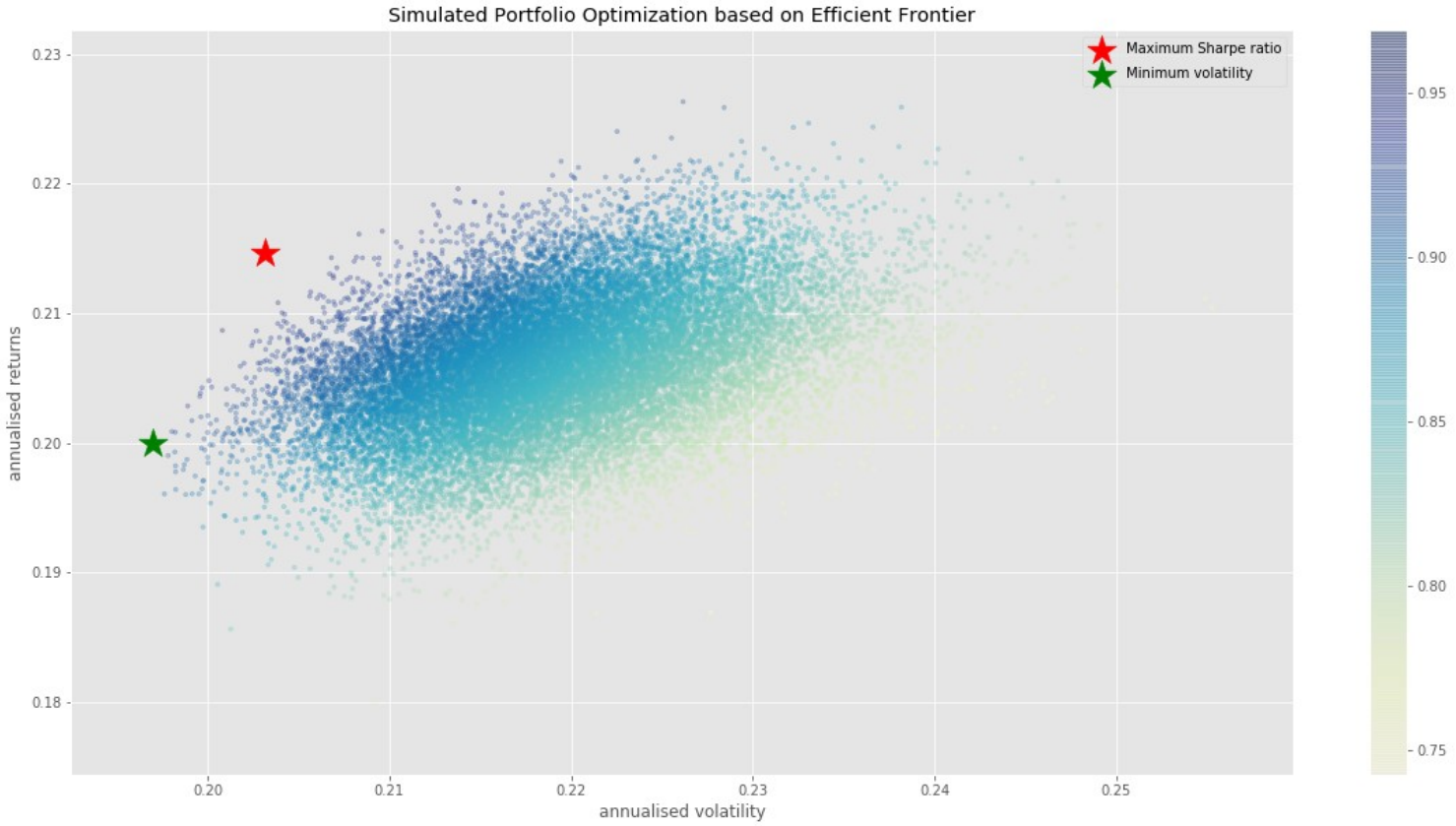
```
def display_simulated_ef_with_random(mean_returns, cov_matrix, num_portfolios,
risk_free_rate):
    results, weights = random_portfolios(num_portfolios,mean_returns, cov_matrix,
risk_free_rate)

    max_sharpe_idx = np.argmax(results[2])
    sdp, rp = results[0,max_sharpe_idx], results[1,max_sharpe_idx]
    max_sharpe_allocation =
pd.DataFrame(weights[max_sharpe_idx],index=table.columns,columns=['allocation'])
    max_sharpe_allocation.allocation = [round(i*100,2)for i in
max_sharpe_allocation.allocation]
    max_sharpe_allocation = max_sharpe_allocation.T

    min_vol_idx = np.argmin(results[0])
    sdp_min, rp_min = results[0,min_vol_idx], results[1,min_vol_idx]
    min_vol_allocation =
pd.DataFrame(weights[min_vol_idx],index=table.columns,columns=['allocation'])
    min_vol_allocation.allocation = [round(i*100,2)for i in
min_vol_allocation.allocation]
    min_vol_allocation = min_vol_allocation.T

    print ("-"*80)
    print ("Maximum Sharpe Ratio Portfolio Allocation\n")
    print ("Annualised Return:", round(rp,2))
    print ("Annualised Volatility:", round(sdp,2))
    print ("\n")
    print (max_sharpe_allocation)
    print ("-"*80)
    print ("Minimum Volatility Portfolio Allocation\n")
    print ("Annualised Return:", round(rp_min,2))
    print ("Annualised Volatility:", round(sdp_min,2))
    print ("\n")
    print (min_vol_allocation)

    plt.figure(figsize=(20, 10))
    plt.scatter(results[0,:],results[1,:],c=results[2,:],cmap='YlGnBu', marker='o',
s=10, alpha=0.3)
    plt.colorbar()
    plt.scatter(sdp,rp,marker='*',color='r',s=500, label='Maximum Sharpe ratio')
    plt.scatter(sdp_min,rp_min,marker='*',color='g',s=500, label='Minimum volatility')
    plt.title('Simulated Portfolio Optimization based on Efficient Frontier')
    plt.xlabel('annualised volatility')
```

```
    plt.ylabel('annualised returns')
    plt.legend(labelspacing=0.8)
```

In [48]:

```
display_simulated_ef_with_random(mean_returns, cov_matrix, num_portfolios,
risk_free_rate)
```

```
--------------------------------------------------------------------------------

Maximum Sharpe Ratio Portfolio Allocation



Annualised Return: 0.21

Annualised Volatility: 0.2




               IT   SHW   COO  SBAC   ROP  MCO  TMO   APH   FIS   COST

allocation  16.24  29.2  18.5  4.08  2.03  0.8  5.5  3.59  4.58  15.47

--------------------------------------------------------------------------------

Minimum Volatility Portfolio Allocation



Annualised Return: 0.2

Annualised Volatility: 0.2




               IT    SHW    COO  SBAC   ROP   MCO   TMO   APH   FIS   COST

allocation  11.56  17.39  10.22  4.33  0.01  2.91  8.76  7.36  9.07  28.38
```



For minimum risk portfolio, we can see around 30% of our budget is allocated to "Cost" - Costco. If you take another look at the daily return plot from earlier, the Costco is the least volatile among these stocks, so allocating a large percentage to Costco for minimum risk portfolio makes sense.

If we are willing to take higher risk for higher return, one that gives us the best risk-adjusted return is the one with maximum Sharpe ratio. In this scenario, we are allocating a significant portion to "SHW" - Sherwin-Williams and "COO" - The Cooper Companies, which are quite volatile stocks from the previous plot of daily returns. And "Cost" - Costco which had around 30% in the case of minimum risk portfolio, has only 15% budget allocated to it.

Coming work process¶

The part of Portfolio optimization and machine learning are still in processing, the proportion of investment strategic will be expected to generate.

For the machine learning part, building the model for S&P 500 index will be implemented including Moving average, Linear regression, k-Nearest Neighbours, auto arima, Long Short Term Memory to generate the prediction and compare the root mean square error in order to get the less error model and most accuracy model.

In [ ]: