I. Objective

The aim of this project is to demonstrate the knowledge of statistics and econometrics by designing an algorithmic trading using python.

Data Source: Yahoo Finance
***Type of asset*:** Index - S&P500
Timeframe: 1st October 2000 to 1st January 2019
Coding language: Python, with some R function
Model Used: Arima + GARCH

3.3.1 Algorithmic Trading

1. Explain the algorithm step by step
2. Provide R code and/or Excel calculations
3. Provide charts
4. Calculate returns, cumulative returns, standard deviation and forecasts
5. Indicate research papers or books on this topic

3.3.2 Improving Strategy

1. Indicate ways for improving the previous algorithmic trading strategy
2. Indicate research papers or books on the topic

II. Introduction

Packages used:

- pyramid.arima
- arch
- rpy2.robjects.packages
- rpy2.robjects
- numpy
- pandas
- pandas_datareader
- datetime
- matplotlib.pyplot

Algorithmic trading is using computers to generate trading signals, sending orders and managing portfolio using algorithms with or without human initiation. Sophisticated electronic markets/platforms are used by the algorithms to trade in the similar fashion as done in electronic trading. The difference is that in algorithmic trading decisions about volume or size, timing and price are determined by the algorithm. Furthermore, algorithmic trading efficiently increases the universe being traded by an individual trader which is limited in electronic trading environment.

When we start designing strategies for stock trading, biggest obstacle we face is continuous change in volatility (or high variance) of particular stock (which we are targeting). Hence to consider volatility factor in our strategy, we have to choose apt model for stock forecasting. In this submission, our focus will be on combination of such two model, i.e. ARMA and GARCH.

ARIMA (p,d,q) – It stands for AutoRegressive Integrated Moving Average. Let's cover it in step by step.

1. AR (Autoregression): A model that uses the dependent relationship between an observation and some number of lagged observations. p is a parameter of how many lagged observations to be taken in.
2. I (Integrated): A model that uses the differencing of raw observations (e.g. subtracting an observation from the previous time step). Differencing in statistics is a transformation applied to time-series data in order to make it stationary. This allows the properties do not depend on the time of observation, eliminating trend and seasonality and stabilizing the mean of the time series.
3. 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. q is a parameter of how many lagged observations to be taken in. Contrary to the AR model, the finite MA model is always stationary.
4. p (lag order): number of lag observations included in the model
5. d (degree of differencing): number of times that the raw observations are differenced
6. q (order of moving average): size of the moving average window

Several test and trials are done to get the value of p, d and q according to data. It is a very popular statistical method for time series forecasting and can reduce a non-stationary series to a stationary series using a sequence of differences.

GARCH (p,q)- It stands for Generalized Autoregressive Conditional Heteroskedasticity which allows the method to support changes in the time dependent volatility, such as increasing and decreasing volatility in the same series. It is extension of ARCH or Autoregressive Conditional Heteroskedasticity method which provides a way to model a change in variance in a time series that is time dependent, such as increasing or decreasing volatility.

Specifically, the model includes lag variance terms (e.g. the observations if modeling the white noise residual errors of another process), together with lag residual errors from a mean process. The introduction of a moving average component allows the model to both model the conditional change in variance over time as well as changes in the time-dependent variance. Examples include conditional increases and decreases in variance.

- p: The number of lag variances to include in the GARCH model.
- q: The number of lag residual errors to include in the GARCH model.

A generally accepted notation for a GARCH model is to specify the GARCH() function with the p and q parameters GARCH(p, q); for example GARCH(1, 1) would be a first order GARCH model.
A GARCH model subsumes ARCH models, where a GARCH(0, q) is equivalent to an ARCH(q) model.

Now we have discussed our model in details. Let's focus on implementation of them in building trading strategy in python language.
We are using S&P 500 index close price (adjusted for dividends and splits) from period 1 October 2000 to 1 January 2019. We will be using log returns for data analysis and model fitting. Some libraries from R programming are also used in python codes.

The strategy is carried out on a "rolling" basis:

- For each day 'n', the previous 'k' days of the differenced logarithmic returns of a stock market index are used as a window for fitting an optimal ARIMA and GARCH model.
- The combined model is used to make a prediction for the next day returns.
- If the prediction is negative the stock is shorted at the previous close, while if it is positive it is longed.
- If the prediction is the same direction as the previous day then nothing is changed

III. Results

3.3.1 Algorithmic Trading

We will now discuss our strategy step by step:

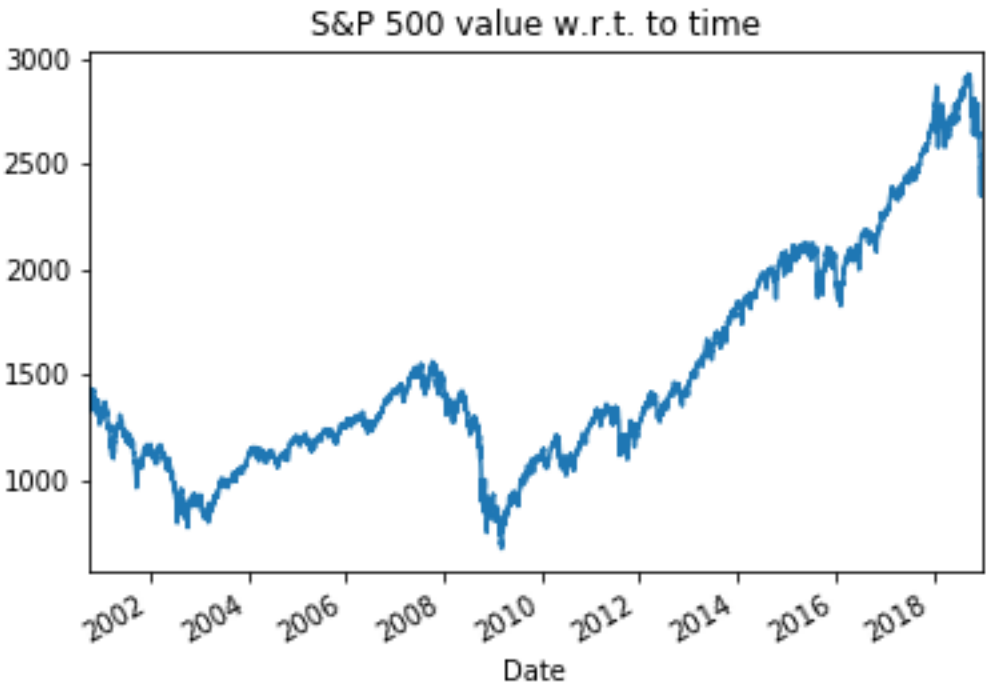
1. Install and Import the libraries mentioned in Introduction part
2. Fetch data S&P 500 index data from Yahoo Finance with help of datareader function for mentioned time frame.
3. Create variable and store the differenced logarithmic returns of the "Closing Price" of the S&P500 and strip out the initial NA value
4. Create a vector, (in our code it is 'forecasted_returns') to store our forecast values on particular dates. We set the 'forecase_length' (lenght of time series to forecast next point) and 'window_length' (number of point we will forecast using previous data)
5. Now we will run the loop for 'forecast_lengths' times on log returns for finding the optimal (best fitting) ARIMA model and as well as GARCH model.
6. Now we will forecast the data using model selected in previous step and store the trading signal into dataframe ('returns_df')
7. Getting actual return by multiplying Stragety signal (i.e. 1 or -1) and Buy and hold strategy return

Following are the screenshots from output of our code:

```
In [2]: gspc = DataReader("^GSPC",  "yahoo", datetime(2000,10,1), datetime(2019,1,1))['Adj Close']

In [4]: gspc.head()

Out[4]: Date
        2000-10-02    1436.229980
        2000-10-03    1426.459961
        2000-10-04    1434.319946
        2000-10-05    1436.280029
        2000-10-06    1408.989990
        Name: Adj Close, dtype: float64
```

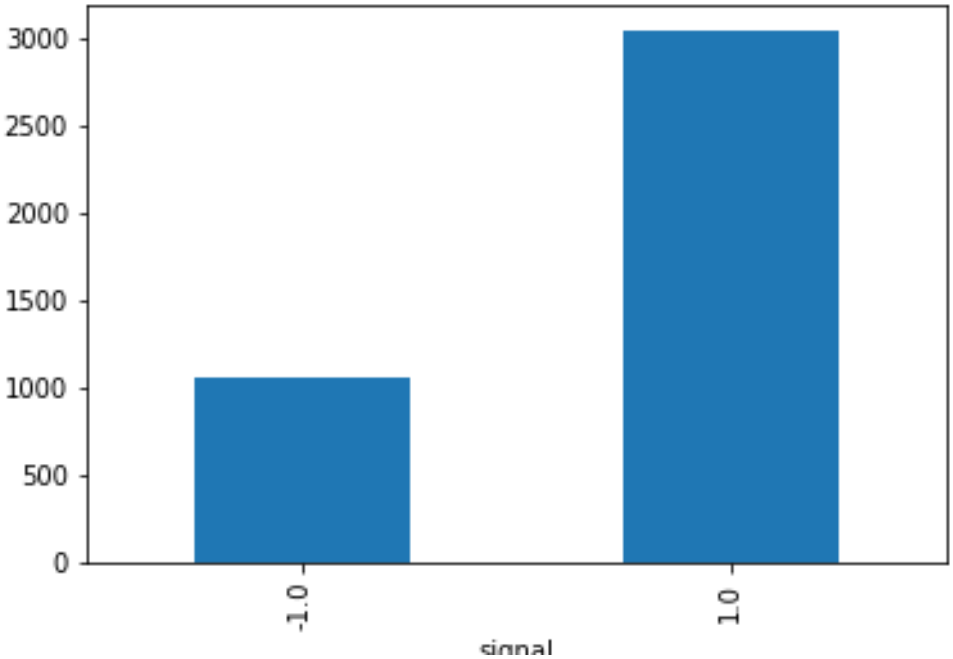


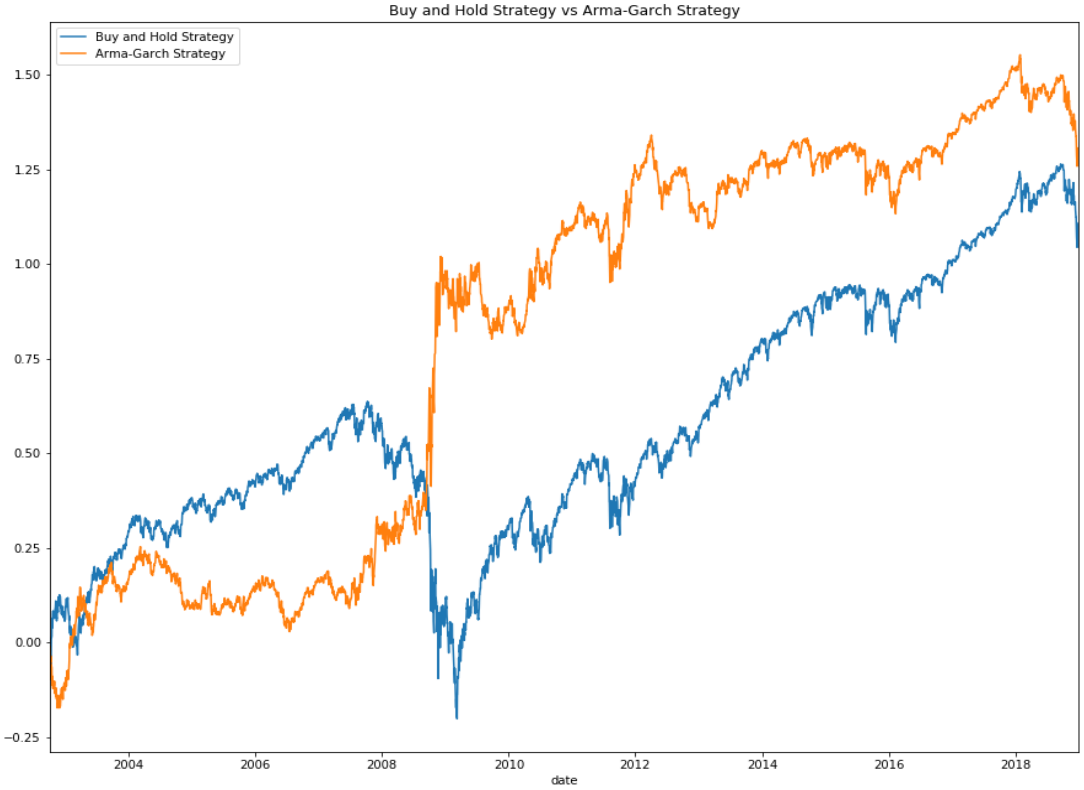
In [5]:
```
gspc_returns = np.log(gspc).diff().dropna()
gspc_returns.head()
```

Out[5]:
```
Date
2000-10-03   -0.006826
2000-10-04    0.005495
2000-10-05    0.001366
2000-10-06   -0.019183
2000-10-09   -0.004952
Name: Adj Close, dtype: float64
```

In [6]:
```
#window length => lenght of time series to forecast next point
#forecast_length => Number of point we will forecast using previous data
window_length = 500
forecast_length = len(gspc_returns) - window_length
forecast_length
```

Out[6]: 4089


3000
2500
2000
1500
1000
500
0
-1.0
1.0
signal


Buy and Hold Strategy vs Arma-Garch Strategy
Buy and Hold Strategy
Arma-Garch Strategy
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
2004
2006
2008
2010
2012
2014
2016
2018
date

Interpretation of Output: As you can see, over a 18 year (approx.) period, the ARIMA+GARCH strategy has significantly outperformed "Buy & Hold". Notice that the volatility of the curve is quite minimal until the early 2008s, at which point the volatility increases significantly and the average returns are less impressive. Equity curve remains below a Buy & Hold strategy for almost 4 years (i.e. between 2004 and 2009) and during the stock market crash of 2008/2009 it does not perform well. This makes sense because there is likely to be a significant serial correlation in this period and it will be well-captured by the ARIMA and GARCH models. However, you can also see that the majority of the gain occurred between 2010 and 2017. Once the market recovered post-2009 and enters what looks to be more a stochastic trend, the model performance begins to suffer once again.

Basic Stats:
1056 times Sell signal
3041 times Buy signal

Stats Buy and Hold
Avg. Daily Returns = 0.000276
std = 0.011605
cumm Returns = 1.132254

Arma Garch
Avg. Daily Returns = 0.000325
std = 0.011604
cumm Returns =1.329533

3.3.2 Improve the Strategy

Indicate ways for improving the previous algorithmic trading strategy

- To improve the above strategy, we could use a deep learning model to predict stock prices on the S&P 500 using time-series historical data.
- A regression model worked best for this as the model would output a numerical value for the future price of a stock, as well as a probability of how confident it felt with the prediction.
- We could use the mean squared error (MSE), to generate a measure of deviation between the model's prediction and actual training targets.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

- The input of the model can be a two-dimensional matrix, while the outputs would then be a single dimensional vector. These placeholders serve the purpose of fitting our model, as X is the input, or the stock prices of all stocks under the S&P 500, while Y is the output or index value of the S&P 500 one minute into the future

```
# Placeholder
X = tf.placeholder(dtype=tf.float32, shape=[batch_size, number_stocks])
Y = tf.placeholder(dtype=tf.float32, shape=[batch_size])
```

Adding Predictors

Predictors are a set of feature variables that must be chosen to train the model and predict our *outcome*. So, forecasting factor choice is crucial

Using Dropout to Prevent Over-Fitting and Improve Accuracy

Dropout is a technique for regularization in machine learning models to prevent neural networks from overfitting. Overfitting is essentially when the model will model training data to well. It learns the detail and noise in the training to an extent that it negatively impacts performance on new data, such as the test and validation set. It works by selecting random neurons which are then ignored, or "dropped out", randomly. Any updates to the weights of the model, as well as the ignored neurons contribution to the forward pass, are not applied to the neuron in backpropagation. This forces other neurons to have to "step in" and handle any representation required to make predictions for the missing neurons. This creates multiple independent internal representations being learned by the model and causes the network to become less sensitive to the specific weights of neurons. The network is able to become more capable of generalization and less likely to overfit.

We can incorporate this into all the hidden layers of the model, which would help lower the test mean squared error. The dropout rate that could be used is 50% or 0.5, meaning through each iteration, half of the neurons are randomly "dropped-out".

```
# Hidden Layer
hidden_layer1 = tf.nn.relu(tf.add(tf.matmul(X, weights_hidden_1), bias_hidden_1))
dropped = tf.nn.dropout(hidden_layer1, keep_prob = 0.5)
hidden_layer2 = tf.nn.relu(tf.add(tf.matmul(hidden_layer1, weights_hidden_2), bias_hidden_2))
dropped = tf.nn.dropout(hidden_layer2, keep_prob = 0.5)
hidden_layer3 = tf.nn.relu(tf.add(tf.matmul(hidden_layer2, weights_hidden_3), bias_hidden_3))
dropped = tf.nn.dropout(hidden_layer3, keep_prob = 0.5)
hidden_layer4 = tf.nn.relu(tf.add(tf.matmul(hidden_layer3, weights_hidden_4), bias_hidden_4))
dropped = tf.nn.dropout(hidden_layer4, keep_prob = 1)
```

After importing the data, defining the placeholders, variables, cost functions and AdamOptimizer, the network can be trained through **mini-batch training**. This is where we take random samples out of the dataset and feed it into the network. A batch of data, Y will flow through the network's hidden layers until it reaches the output layer, where it is compared to the target data, Y , in the current batch. Using the optimizer and cost function, the model updates its parameters, such as it's weights and biases. This repeats with each batch of data. One full iteration throughout all batches is known as an **Epoch**

Through utilizing things such as the dropout rate and having a criteria for stopping the model, the model would yield far better results.

- To improve our forecasts, beyond the above ARIMA GARCH Trading strategy, we can consider ***long-memory processes, state-space models and cointegrated time series.***
- We can use cointegration, which would allow us to determine if we are able to form a mean reverting pair of assets. To achieve this, we need a robust mathematical framework for identifying pairs or baskets of assets that mean revert.

Consider a pair of time series, both of which are non-stationary. If we take a particular linear combination of these series it can sometimes lead to a stationary series. Such a pair of series would then be termed *cointegrated*. Stock price evolution is very difficult to be forecasted but the spread between two correlated stocks is easier to be predicted. The spread can be a stationary time series, and therefore a trader can use an Autoregressive Moving Average (ARMA) model to predict the spread.

- We can try to forecast something different but close prices or returns — volatility, skewness, maybe other characteristics.
- ***Backtesting technique*** – It means simulating an investment strategy based on historical data. It is also used for calibrating and evaluating an investment strategy. Various indicators related to risk and return are calculated. It can be used with ARMA-GARCH Analysis to improve trading strategy.
  -VAR (Value -At-Risk) and Sharpe Ratio calculated can be used for generating longer-term trends considering different macro or financial indicators and increase the scope of ARMA-GARCH modelling from short- term predictions.
  -The in-sample data is used for parameter estimation and to evaluate the performance. The estimated parameters are used in the out-sample data to check their generalization capacity. If the tests using out-sample data provide similar results with in-sample test, then we validate the trading strategy and we can use it in live trading.

- **Multi -Scale Capability**-The essence of multi-scale capability is that it takes snap-shot measures over various time horizons and unifies them into a coherent inter-temporal decision framework. Given that investors do not want to be hampered by tail events along the path to longer term performance, this methodology brings to the fore the time period over which the investor must wait to know with a high degree of certainty that the strategy will cover its costs. The strategy for which this number is the lowest is the best strategy.
  The multi-scale capability methodology protects both the investor and the marketplace with real time monitoring and operational SPC,and verifies the validity of performance with SPC of the input reconciliation data. Thus, this methodology is sufficient to assure the prudence of a trading strategy in the legal sense, and that its quantitative nature lends itself to easy outside verification.

- ***Long Memory Processes***-Many assets have the "long-memory" property, which means that the autocorrelation of the volatility decays slower than exponentially as it does in an ARMA type process. One way to incorporate this property into the Stochastic Volatility(SV) model is by allowing the log-volatility to follow a fractionally integrated ARMA (ARFIMA) process. Such a model is called a long-memory stochastic volatility (LMSV) model.

[Econometrics]

IV. Bibliography/Reference

Daroczi G. et al. Introduction to R for Quantitative Finance. Packt Publishing.

Halls-Moore, M. L. (2017). Successful Algorithmic Trading.

Jeet, P. and Vats, P. Learning Quantitative Finance with R. . Birmingham: Packt Publishing.

Jiang, W. (2012). Using the GARCH model to analyse and predict the different stock markets.

Hultman, H. (2018). Volatility Forecasting.

Scott, M. et al. (2013). Financial Risk Modelling and Portfolio Optimization with R. Wiley.

Chitrananda, S. (Dec 21, 2018). How I Built A Model To Predict Prices on the S&P 500 Using Deep Learning https://medium.com/datadriveninvestor/how-i-built-a-model-to-predict-prices-on-the-s-p-500-using-deep-learning-bde4745e5d86

Nalon, A. The Rise Of Automated Trading: Machines Trading the S&P 500 https://www.toptal.com/machine-learning/s-p-500-automated-trading

Krtek, J and Vosvrda, M. Academy of Sciences of the Czech Republic, Institute of Information Theory and Automation. SVV 305–09/261315 ˇ Supervisor. Comparing Neural Networks and ARMA Models in Artificial Stock Market
https://pdfs.semanticscholar.org/eace/04915feea91069d92658ec55d2192c28bd7a.pdf

Coopera, R. Ongb, M and Van Vliet, B. Stuart School of Business, Illinois Institute of Technology, Chicago, IL, USA. Multi-scale capability: A better approach to performance measurement for algorithmic trading https://content.iospress.com/download/algorithmic-finance/af043?id=algorithmic-finance/af043

Libo Xie. (Jan25, 2006). Long-Memory Stochastic Volatility Models: A New Method for Inference and Applications in Option Pricing
https://www.stat.cmu.edu/proposals/LiboXie.pdf