University of St Andrews | FOUNDED 1413

Can Predicted Fund Management Decisions Improve Stock Volatility Forecasts?

Thesis submitted to the University of St Andrews for the degree of
Master of Science in Statistics, 16th of August 2022.

Zane Hassoun

210031934

Supervised by

Valentin Popov

Signature _____

Date _____ / _____ / _____

Abstract

The United Kingdom alone has around 8.5 Trillion Pounds worth of assets under management, and fund managers must make difficult asset allocation decisions. A British company, Irithmics, built an agent based machine learning model to predict, based upon the prior fund management behavior, the probability of the entire group of funds to short a given stock on a given day. This data was graciously donated for my M.Sc. Dissertation by company founder and St Andrews Alumni Grant Fuller. This paper seeks to determine whether the "Wisdom of the Crowds" holds any merit when trying to forecast the volatility of stock market returns. It specifically looks at four constituents of the FTSE 100 (Lloyd's, Tesco, Rolls Royce, Vodafone) during one of the largest times of uncertainty for the stock market known to date, the year of 2020. The research focuses determining whether the exogenous data provided by Irithmics can increase the 1-day ahead forecast accuracy of an optimized univariate empirical conditional volatility model. The compelling aspect of the research question is the exogenous data may not necessarily be an accurate prediction of the fund managers behavior, yet it still may be an advantageous addition to the model.

Declaration

I, Zane Hassoun, hereby certify that this dissertation, which is approximately 8,600 words in length, has been composed by me, that it is the record of work carried out by me and that it has not been submitted in any previous application for a degree. This project was conducted by me at the University of St Andrews from May/2022 to August/2022 towards fulfilment of the requirements of the University of St Andrews for the degree of Master of Science in Statistics under the supervision of Valentin Popov. [insert signature]

Contents

Chapter 1

Introduction

1.1 Portfolio Management

In financial services, a portfolio is a group of investments including but not limited to stocks, bonds, commodities and real estate. The act of managing said investments is what is called: "Portfolio Management". Formally, this is defined as: "... selecting and overseeing a group of investments that meet the long-term financial objectives and risk tolerance of a client, a company or an institution" (14). Organically, this brings forth the quandary of how does the portfolio manager effectively allocate the assets to best reach its objective? This dissertation will not be focusing specifically on portfolio management theory, it instead will be focused on the concept of volatility in stock returns. To effectively allocate assets, having understanding of volatility is paramount to forecast asset prices. This is defined as: "a statistical measure of the dispersion of returns for a given security or market index ... [it] is often measured as either the standard deviation or variance between returns from that same security or market index" (13).

1.2 Modelling Asset Returns

While it is impossible to obtain the true volatility or instantaneous standard deviation of a stock's return tomorrow, models can be created to **simulate** the future. Returns can be modelled in two ways: discrete or continuous/logarithmic. Discrete returns are simply

$$R_t = \frac{P_t - P_{t-1}}{P_T}$$

or the rate of change between time point $\mathbf{t}$, and time point $\mathbf{t - q}$ where $\mathbf{q}$ is the number of days between price observations. This research will be considering the daily, continuous/logarithmic returns of stocks on the London Stock Exchange. Formally, a continuous/logarithmic return:

$$r_t = ln(\frac{P_t}{P_{t-1}}) = ln(P_t) - ln(P_{t-1})$$

Intuitively, the difference lies in the interval between $\mathbf{t}$ and $\mathbf{t - 1}$, where in a discrete return this is simply 1 period, but in a continuous return, this period would be broken in to $\mathbf{k}$ sub intervals, and over those sub intervals the return is continuously compounded, thus yielding.

$$(1 + \frac{rt}{k})^k || \ \ letting \ \ \mathbf{k} \to \infty \ \ || \quad \lim_{k \to \infty} (1 + \frac{rt}{k})^k = e^{rt}$$

(16)

1.2.1 Time Series

In statistics, observations that are taken across a set of consecutive time points are considered a "Time Series" (16), and therefore when stock market returns are observed, they are classified as a "Financial Time Series". These financial time series observations are considered to have what is called stochastic properties. A stochastic process is a set

of observations that follow a random probability distribution and are indexed by time. Formally defined as:

$$X = (X_t)_{t \subset \mathbf{T}} \mu_x(t) = E(X_t) \sigma_x^2(t) = V(X_t) = E(X_t^2) - [\mu_x(t)]^2$$

The most easily digestible example of a stochastic process is the Gaussian White Noise. This process has a $\mu_x = 0; \sigma_x^2 = 1$ and the points are sampled from a normal distribution. **Figure1.1** is a simulated example of the Gaussian White Noise Stochastic process, and shows the points oscillating around the mean of 0 while being randomly distributed generally within 1 standard deviation of the mean.

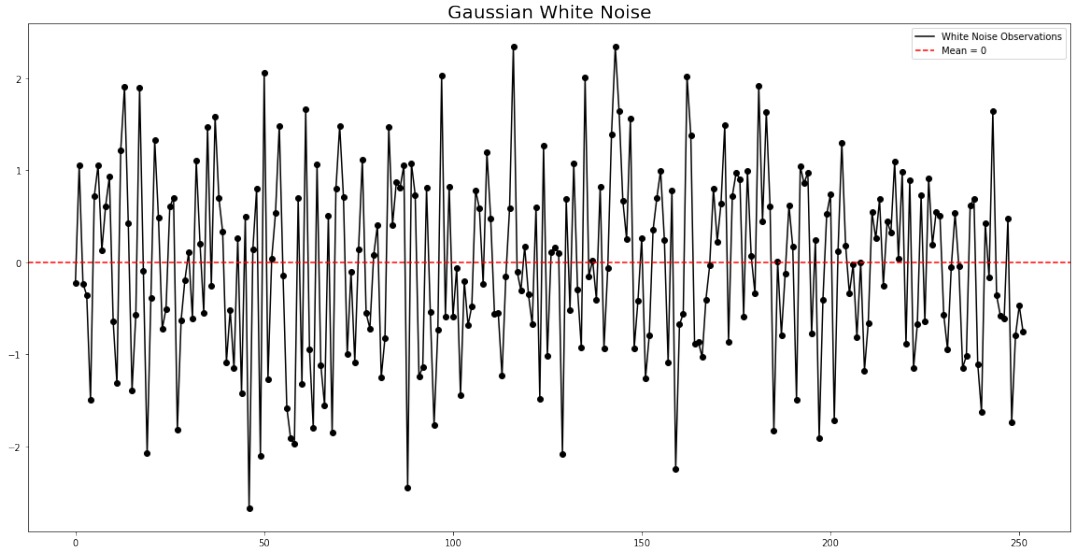


Figure 1.1: Random Gaussian White Noise Example

1.2.2 Properties Financial Data

Unique to financial data, specifically continuous stock returns, there are properties or "stylized facts" (16) that are used as a guide.

1. The distribution of the returns are non-normal in two ways:
   (a) More mass in the tails (fat tails) - greater probability of events further from the mean

   (b) There is a higher peak than a normal distribution (leptokurtic) - greater clusters of data at the mean

2. Raw returns are uncorrelated, but transforming to squared or absolute, significant autocorrelation is observed

3. "Volatility Clustering" - if today's return has a large absolute value it tends to be followed by another return with large absolute value, and visa versa with smaller values oof returns.

(16)

1.3 Motivating Example

The properties of volatility when modelling returns guide us to the types of models we are using. For example, take Lloyds Banking Group, PLC (LLOY.L)'s returns from 2017 - 2020. Using Maximum Likelihood Estimation, I fit the log returns to a distribution and found they fit best to a Cauchy distribution with a location parameter -0.108 and Scale: 0.76 **Figure1.2**. If I assumed mean and volatility to be constant, its expected value is shown by the bottom plot in **Figure1.3**. Intuitively this is illogical, financial time series data are dependent on its location in the series, not random observation, this series assumes complete randomness with fat tails that are causing great changes. Comparing the red line in 1.3 to the true Lloyd's returns in the blue line, it is clear the observations volatility's are clustered and the tails are larger than a normal distribution.

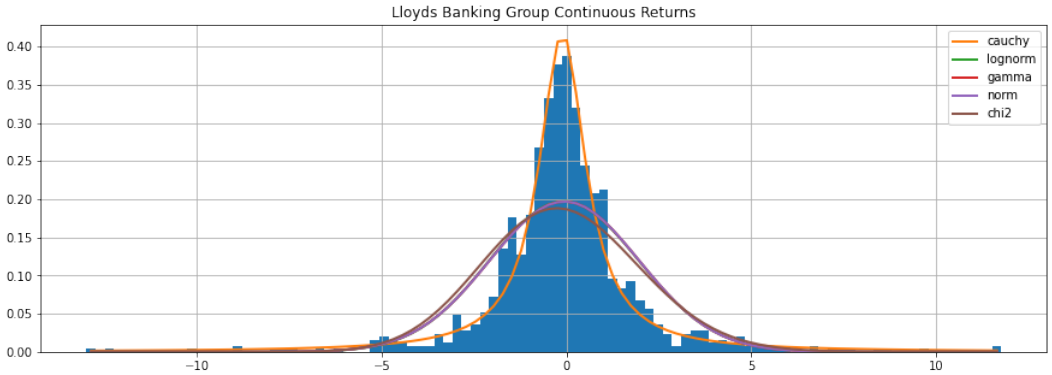


Figure 1.2: Lloyds Returns Fitted Distribution

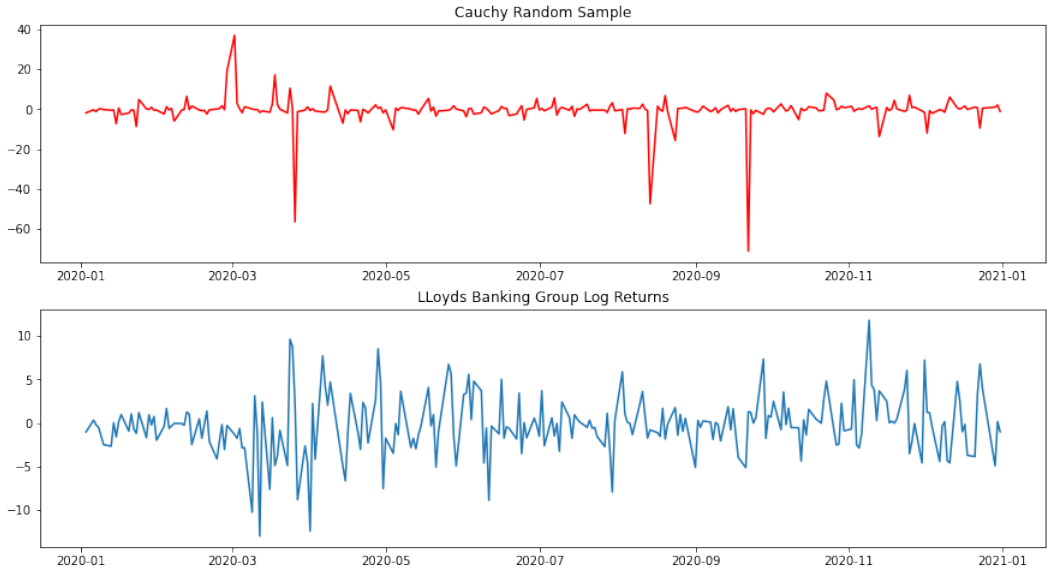


Figure 1.3: Lloyd's Returns vs Cauchy Simulated Returns

Therefore, this dissertation seeks to find a more accurate and efficient way to model stock market returns to extract the volatility/instantaneous standard deviation.

1.4 Irithmics Exogenous Data

In an attempt to approach this problem from a novel direction, I will incorporate an exogenous variable into the volatility forecasting model with the hope of increasing the predictive power. Graciously, Irithmics, a U.K. based financial technology and data company donated data it's deep learning machine generated. These data include over a 170

trading days, each with 75-80 a probability density function indexed on each day with the probability value of how likely the deep learning algorithm believes an aggregated group of 250,000 funds will sell/short a specific stock. The corporation's method: "Our platform uses state-of-the-art deep learning originally designed to help describe and understand the spread and impact of long-term chronic human diseases, like diabetes. We have applied these technologies to learn from over 250,000 global institutional investors and funds, across hundreds of thousands of portfolios with a combined value of many trillions of dollars" (11). The compelling aspect of this exogenous data is it is not objectively correct and therefore the model contains many layers of uncertainty. First, there is the challenge of accurately estimating parameters and hyper parameters, as well as model selection, then whether or not Irithmic's algorithm during the selected time period is an accurate representation of the investors future behavior, and finally if the accurate prediction of the institutions behavior has any significant effect on the volatility of a certain stock's returns. Not only does this problem incorporate the concept of "The Wisdom of Crowds" but also the "Crowd" relationship with stock return volatility.

1.5 Data Selection

1.5.1 2020

There was unprecedented amounts of uncertainty and stock market volatility in 2020 due to the COVID-19 Pandemic. While this was a somber time for humanity, it gives the opportunity to stress test models and model accuracy during extremely uncertain points in time. Generally, forecasting is easy during stable and predictable points in time, but money is made and lost during times where no one knows what to do.

FTSE 100

Specifically, This analysis will be done on four FTSE 100 Companies from 4 different sectors. The FTSE 100 is the Financial Times Stock Exchange is the largest 100 companies listed in the United Kingdom by market capitalization. I will chose one from:

- Banking/Finance - LLoyds Banking Group
- Technology Vodafone
- Automotive - Rolls Royce
- Consumer Goods - Tesco

1.5.2 Forecasting

Unlike predictions, which rely on a set of non time indexed inputs, forecasting a time series requires always looking forward in time. This poses a larger challenge as fitting, training, and testing models must only focus on predicting out of sample data that lies forward in time. This research will specifically be focused on what is called "nowcasting". The phrase comes from the study of meteorology where weather events are forecast in very near future. The concept will be applied to stock market returns to forecast volatility for what is called a 1 day ahead rolling forecast.

1.6 Wisdom of the Crowds

1.6.1 Book

In 2004 James Surowiecki, a staff writer at the New Yorker, wrote the book: "The Wisdom of Crowds: Why the Many Are Smarter Than the Few and How Collective Wisdom

Shapes Business, Economies, Societies and Nations". The book spoke on his belief that effective and accurate decision making can almost always be improved by aggregating individual forecasts/predictions/decisions rather than individuals making decisions on their own. The simplest example from the book was the old story about Sir Francis Galton, being taken back at a silly carnival game for a crowd to guess the weight of an ox. The mean of the guesses was accurate, while the individual guesses themselves were no good. This is what the wisdom of the crowds or aggregation of the crowd intellegence. This isnt a statistical text, but gives an idea into how the concept of forecast aggregation and crowd wisdom can play into this research paper. The irithmics data aggregates their perceived action from the crowd. On day X, there is a Y% chance that the 250,000 funds surveyed will short stock Z. If they are shorting stock Z, they are hoping that the stock will go down as they are in the business of making money of their trades. So, the question now becomes, are the crowds wise? are the fund managers aggregated forecasts better than picking say, one fund at Bridgewater Associates, one trader at Jane Street? This paper hopes to incorporate that information, not for stock prices, but to help with the predictions of volatility within the volatilty forecasts.

The book also acknowledges that all group forecasts aren't created equal. To identify a "wise" crowd there must be: "

1. Diversity of Opinion
2. Independence
3. Decentralization
4. Aggregation
5. Trust

"(20) These ideas for using crowd sourced things can be useful in statistical context, each of these items can be good steps for proper data sampling.

Conversely, there are some integral failures of the crowd. These intuitively are the inverse of a smart crowd: “

1. Homogeneity
2. Centralization
3. Division
4. Imitation
5. Emotionality

” (20) The idea would to be cognizant of the data sampled and the derived forecast one would use. This comes back to the garbage in garbage out modelling, where the data is the most important piece of modelling. One would want to obtain varying forecasts to aggregate from every sense of varying. Socioeconomic Status, , Demographics location time etc etc spatial temporal.

1.6.2 Application

The compelling aspect is a form of forecast aggregation has already happened and this project seeks extend the aggregation further. The difference is there is also a question of whether or not the forecast itself carries any relevance in the topic. Irithmics aggregated their machine's deep learning prediction on when the aggregation of funds will sell a stock, but is this relevant to the volatility and can this aggregation be aggregated with the empirical forecast.

Chapter 2

Literature Review

2.1 Volatility Modeling of Stocks from Selected Sectors of the Indian Economy Using GARCH

2.1.1 Overview

(18) is a recent paper presented at the 2021 Asian Conference on Innovation in Technology (ASIANCON). This paper visits the idea of volatility clustering applied to the Indian Economy and therefore on Indian stocks on the NSE. The paper focuses on applying only the concept of **GARCH** or Generalised AutoRegressive Conditional Heteroscedasticity models to forecast volatility of returns rather than incorporating any exogenous variables or data.

2.1.2 Methodology

As the models the researchers considered were quite straightforward and easy to implement in Python, they took a very structured approach to the problem by beginning with the most simple **GARCH(1,1)** model, and then added complexity. This meant begin-

ning with a constant mean model and normally distributed residuals, followed by skewed -t distributed resiudals. They then fit an Autoregressive Moving Average mean model residuals into the model and took the minimum AIC. Then they fit Asymmetric volatility models on the return series, this for example is the **GJR-GARCH** and **EGARCH** models which asses the impact of a negative shock as more impactful than a positive one.

2.1.3 Results

The results were completed by testing the best model by minimum AIC with some out of sample data that the model has not seen. They used an expanding window and fixed window forecast and backtested the **EGARCH** Model. This was evaluated by a lot of different statistics on the Auto and Banking Sector and **EGARCH** in this scenario was best chosen.

2.2 An Empirical Study of Hang Seng Index Based on GARCH Model

2.2.1 Overview

(5) is a recent paper presented at the 2020 second international conference on Economic Management and Model Engineering (ICEMME). The aim was to give an overview of optimal model fitting on the Hang Sang Index. The HSK is 33 stocks, akin to FTSE100, that look to give an overview of the economic and financial activity and health of the honk kong markets. The power of this index is it includes mainland china based companies in addition to Hong Kong. As discussed, the paper admits that the volatility of stock market returns are time-variant and therefore traditional measurements of standard deviation of the whole sample are useless. The goal of this paper was to evaulate what variant of a **GARCH(p,q)** model would fit best to the data

2.2.2 Methodology

Contrary to R's rugarch, or Python's arch library the researchers used Eviews10 software accompanied by the stock market data provided by the Flush Database. The returns calculated were log or continuous returns ($r_t = ln(P_t) - ln(P_{t-1})$). They did exploratory analysis on the data and determined it was stationary, and it was correlated enough to continue their analysis. To select the model they fit **GARCH(1,1)**, **GARCH(2,1)**, **GARCH(1,2)**, **GARCH(2,2)** and picked the best model from the Akaike Information Criteria or AIC. Using minimum AIC they chose the **GARCH(1,2)**.

2.2.3 Results

The main takeaways the researchers had from this analysis were that the continuous returns (log change in daily prices) of the Hang Seng Index did have volatility clustering. The returns also had from the stylized facts fatter tails and sharper peaks in the distribution (similar to **Figure 1.2**). They also determined there were no unit roots in the returns sequence, therefore inferred signs for stability, while through analysis of autocorrelation there was no monthly correlation. The drawbacks were the lack of sample size, but found as usual that the **GARCH** model's Variance predictions were very adequate for stock market returns.

2.3 A GARCH approach to model short-term interest rates: Evidence from Spanish economy

2.3.1 Overview

(19) was published in September 2020 in the International Journal of Finance and Economics. Instead of talking about stock market returns, this paper gives a derivative

approach, modelling short term interest rates of three year Spanish government issued bonds. Again, this process is completed because the researcher believes that **GARCH** models provide: "a valuable alternative against econometric specifications that imply a homoscedastic error term" (19). The data is taken from January 1995 to December 2000 and is useful as there will be exogenous volatility caused by what they say is the European Central Bank assuming monetary sovereignty.

2.3.2 Methodology

The paper takes into account interest rate theory and the exogenous factors that affect the rate that money is returned as a profit for the buyer of the bond, there is expectations, liquidity preference, institutional approach, habitat hypothesis, market segmentation hypotheses (19). They then applied a mathematical eiquation to the problem through the fisher information equation

$$i_R = i_N - \pi$$

This effectively is the real interest rate is equal to the nominal interest rate minus the expected inflation rate. With this information they fit a **GARCH** model with the interest rates as the input hoping to obtain the conditional volatility of the time series

2.3.3 Results

The researchers found that this method of estimating the interests rates when breaking the traditional assumption of homoscedastic variance, instead with the heteroscedastic that **GARCH** employs was much more efficient and accurate. This model was more flexible as well, but a sub-specification of the **GARCH** model worked better than fitting through the traditional equation. The fisher equation was effective in its estimation of the government debt and they believe this can be expanded into other fields of finance and econometrics as well. (19)

Chapter 3

Volatility Modelling Foundations

3.1 The ARCH & GARCH Processes

Because the aim of this research is to create, train and forecast accurate time series models, they must first be adequately introduced. As illustrated and explained earlier, returns of stock prices are non-linear in nature, stochastic processes, and have time variant instantaneous standard deviations/volatility. Because of this researchers have developed rigorous methods to model these properties.

3.1.1 Foundations

Robert Engle

In the 50th volume of the Econometrica Journal published in July 1982, Robert F. Engle introduced a paper titled "Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflation". This groundbreaking piece of work was extremely influential in leading the change in volatility modelling in econometrics. According to semantic scholar, the paper has been cited 19,974 times! On the 8th of October

2003, The Royal Swedish Academy of Sciences awarded Engle the Bank of Sweden Price in Economic Sciences in Memory of Alfred Nobel. They expalined: "He found that the concept of autoregressive conditional heteroskedasticity (ARCH) accurately captures the properties of many time series and developed methods for statistical modeling of time-varying volatility. His ARCH models have become indispensable tools not only for researchers, but also for analysts on financial markets, who use them in asset pricing and in evaluating portfolio risk" (1). The paper introduced the concept that: " these are mean zero, serially uncorrelated processes with nonconstant variances conditional on the past, but constant unconditional variances. For such processes, the recent past gives information about the one-period forecast variance" (8).

Tim Bollerslev

In 1986 Tim Bollerslev took Engles ARCH Model one step further by creating a Generalized ARCH or GARCH Model. In his paper "Generalization of ARCH process" (3) he seeks to create a "more general class of process, GARCH, ... , allowing for a much more flexible lag structure" (3). He took an imperical example of modelling inflation, by extending one of Engles papers from 1983's example. He compared his model, GARCH(1,1) with an ARCH(8) and found that: "In this light it seems that not only does the GARCH(1,1)model provide a slightly better fit than the ARCH(8) model in Engle and Kraft (1983), but it also exhibits a more reasonable lag structure" (3). Fundamentally, the model extension lead to great gains and found that the research was able to be better explained using GARCH.

3.1.2 Mathematics & Interpretation

ARCH

The Autoregressive Conditional Heteroscedasticity "ARCH" model introduced by Engle considers the volatility or instantaneous standard deviation of a time series independent from its mean function and implied by the name, that volatility is time variant (4). The model takes one parameter as an input and is generally written as **ARCH**(**P**) . The volatility at time point **t** is calculated by considering **p** time periods in the past's value of the series multiplied by some parameter α_1 plus another intercept parameter α_0.

$$\{X_t\}_t \sim ARCH(p) \iff X_t = \sigma_t W_t$$

This function implies that W_t is white noise and σ_t is a positive function of the previous **p** lags (observations in the time series) (4). For example, the function to calculate the standard deviation for an **ARCH**(**1**) process is as follows:

$$\sigma_t^2 = \alpha_0 + \sum_{j=1}^{p} \alpha_j X_{t-j}^2;\ for\ j = 1, 2, ...p$$

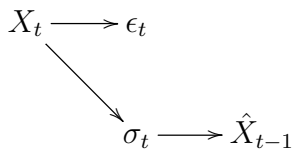
This is not to say the only time period considered is p-behind, interestingly, the σ_t is actually calculated as a weight decaying function of all of the previous lags in the time series, an infinite decay so to speak.

$$\sigma_t = \sqrt{\alpha_0(\sum_{j=0}^{\infty} \alpha_1^j W_{t-1}^2 W_{t-2}^2 ... W_{t-j}^2)}$$

Intuitively this explains the auto regressive nature that the current position relies on all the previous lags with decaying levels of importance, where W_{t-1}^2 is of greatest weight.

To intuitively understand the model, I find it best to consider a causal diagram of what

actually affects the time series at **t**. For an **ARCH**(**p**) process,



Value of the time series at time point **t** t is calculated by white noise, multiplied by the square root of a constant, α_0, plus a constant α_1 multiplied by the value ofs time series squared: $\sigma_{\mathbf{t}}\epsilon_{\mathbf{t}}$. As the diagram shows, the value of the time series, today, is affected by the random error ϵ_t today, as well as the volatility today. But, since the volatility is a function of the time series yesterday, we see that σ_t is a function of σ_{t-1}. The autog regressive nature of the model, is simply an extension of the thought that σ_t is a function of σ_{t-1} because σ_{t-1} is a function of σ_{t-2} and so on and so forth. The weight of σ_{t-n} to σ_t depends on $|\mathbf{t} - \mathbf{n}|$ (17).

ARCH models are thought of as being "bursty" since, from the causal diagram the observation at time point t in the series is dependent only on the last value of the time series it can yield high peaks but values that are not sustained. The idea of volatility clustering is not modelled as well as it could, as the model would identify the peak to the new level of the cluster, but not for 3,5,7 days for example of sustained volatility (17)

GARCH

The GARCH model by nature is just an extension of the **ARCH** with a G. G stands for Generalized and thus **GARCH** has a parameter **p** but also a second **q** ie **GARCH**(**p**, **q**).

$$\{X_t\}_t \sim GARCH(p,q) \iff X_t = \mu_t + \sigma_t W_t$$

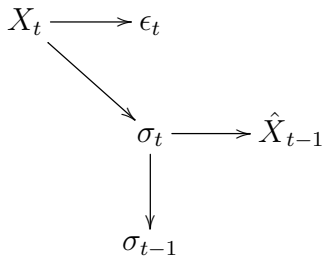
The distinct difference with the **GARCH** model is that the white noise is Gaussian. The

new equation for σ_t^2 the instantaneous or current volatility at the time point now is

$$\sigma_t^2 = \sigma^2 + \sum_{j=1}^{p} \phi_j \sigma_{t-j}^2 + \sum_{j=1}^{q} \theta_j \hat{X}_{t-j}^2;\ for\ j = 1, 2, ...p$$

$$\mathbf{GARCH(1,1)} \quad : \quad \sigma_t^2 = \sigma^2 + \phi_1 \sigma_{t-1}^2 + \theta_1 \hat{X}_{t-1}^2$$

(4). Akin to ARCH, the model relies on the last **p** observation's value in the time series, but also the last **q** standard deviations. For **GARCH**, since there is an extra dependency for **GARCH**$(\mathbf{p}, \mathbf{q})$ not just **ARCH**$(\mathbf{p})$, the causal diagram has another element! For a **GARCH**$(\mathbf{p}, \mathbf{q})$ process:



In the **GARCH** case, the value of the time series at time point **t**, is calcualted the same way **ARCH** is, but now not only is the volatility, at time point **t** a function of the time series $\hat{X}_{t-1}$ yesterday, but it is also a function of the volatility σ_{t-1} yesterday hence the two arrows coming from $\sigma_t(17)$.

Compared to ARCH the GARCH model lends itself to be better at modelling the volatility clusters. The model equation now shows that σ_t the immediate or instantaneous volatility is a function of not only the last value of the time series but also the last value of the standard deviation.

Further Extensions

The traditional **GARCH** model is the typical symmetric model that generally is used as it is extremely powerful and accurate, but over the years since Bollserslev proposed the paper there have been different derivative models. Discussed now are univariate models, where it takes one input, in this case log/continuous returns. There are also

different models such as Exponential **GARCH** (**EGARCH**) Asymmetric (**GARCH**) (**AGARCH**), **GJR − GARCH**, **TGARCH**, (**APGARCH**), **GARCH**. These models derive different variances by modifying the generalized equation. For example, a very useful model, (**EGARCH**) the log of the variance instead of the variance for (**q**). (2)

3.2 Dynamic Conditional Correlation (DCC)

To compare the volatility forecast from a univariate **GARCH**(**p**, **q**) model (modelling a single variable, continuous returns) with a multivariate model (two variables, continuous returns, Irithmics probabilities) I employed what is called a Dynamic Conditional Correlation "DCC" to evaluate the correlation. This model was again founded by the founder of the **ARCH**(**p**) model, Robert Engle. These models are said to be a "Simple Class" of multivariate GARCH models, but their strength is in its flexibility akin to a univariate GARCH process, while still a parameterized model. The models themsevles are not linear in nature, but do use maximum likelihood estimation and are empirically effective (9). The intuition behind this model is similar to that of the **ARCH**/**GARCH**: Like volatility over time, the correlation between the volatilites of two time series' are not static, but instead dynamic and modelling them as an average over time instead of continuously dynamic is inadequate. Statically, conditional correlation between two observations is seen as:

$$\rho_{1,2} = \frac{E_{t-1}(r_{1,t}r_{2_t})}{\sqrt{E_{t-1}(r_{1,t}^2 r_{2_t}^2)}}$$

In order to transform this function into a dynamic and generalisable method, the company $RiskMetrics^{TM}$ has proposed using decaying weights through a λ parameter. This has a drawback of an expanding window where correlations n periods behind are not being omitted, when they are clearly uninformative. The additional issue as found in many smoothing problems is there is no optimal parameter for λ, RM sets it equal to 0.94, but

that is by no means optimal for all problems

$$\hat{\rho}_{1,2} = \frac{\sum_{s=1}^{t-1} \lambda^{t-j-1}(r_{1,s}r_{2_s})}{\sqrt{\sum_{s=1}^{t-1} \lambda^{t-j-1}(r_{1,s}^2 r_{2_s}^2)}}$$

The way Engle remedied this impass was by using a "natural alternative" to exponential smoothing, by instead using the GARCH(1,1) model and incorporating the correlation estimator giving the covariance matrix: $Q_t = |q_{i,j,t}|$ which is " is a weighted average of a positive definite and a positive semidefinite matrix" (9). The mathematics is quite complicated, but it is easily implemented in Python.

3.3 GARCH with Exogenous Regressor

In order to fit this exogenous covariate of Irithmics probabilities, the univariate **GARCH** model must be extended. The choice for this research was, using the probabilities themselves as an exogenous regressor. Letting the next out of sample regressor be equal to π, the multivariate **GARCH** model is:

$$\sigma_t^2 = \sigma^2 + \sum_{j=1}^{p} \phi_j \sigma_{t-j}^2 + \sum_{j=1}^{q} \theta_j \hat{X}_{t-j}^2 + \beta_0 + \beta_1[\pi_1, t];\ for\ j = 1, 2, ...p$$

Chapter 4

Methodology

4.1 Data

4.1.1 Processing

Stock Returns

To obtain stock market returns for Lloyd's, Tesco, Rolls Royce and Vodafone, I used Python 3.9 (21) and the Pandas (15) library to interact with the Yahoo Finance API. I was specifically working with the closing price (price at the end of the trading day) and the data span from the first trading day in 2017 to the last in 2020. To calculate continuous daily returns I used the formula:

$$r_t = ln(close_t) - ln(close_{t-1})$$

I chose the continuous daily returns instead of the discrete returns because of the mathematical ease of temporal aggregations for multi-period continuous returns are simply the sum of the logarithmic returns. Take for example k-periods: $r_t(k) = ln(P_t) - ln(P_{t-k})$. I chose to use a simple tool like the Yahoo Finance API over a real time data feed such as

Refinitiv because I used simple closing prices of years prior historical stock market data. If I was planning analysis of high frequency time periods, I surely would have used it, but this was easier to implement to the models and has greater opportunity for repeatability by others as Yahoo Finance is free.

Irithmics Data

Unlike simple closing prices of a stock, the data provided by Irithmics arrived in a much more complex way. These data were delivered in comma separated value (.csv) format, that for each company in the FTSE 100, and each trading day in 2020, included a derived discrete probability density, indexed with between 75 and 80 days, describing the probability the deep learning algorithm assigned of a short/sell by the group of funds. **Figure4.1** contains a visual example of what the files for date index January, 26th look like. As this is akin to a probability density, the values on each day will sum to one.

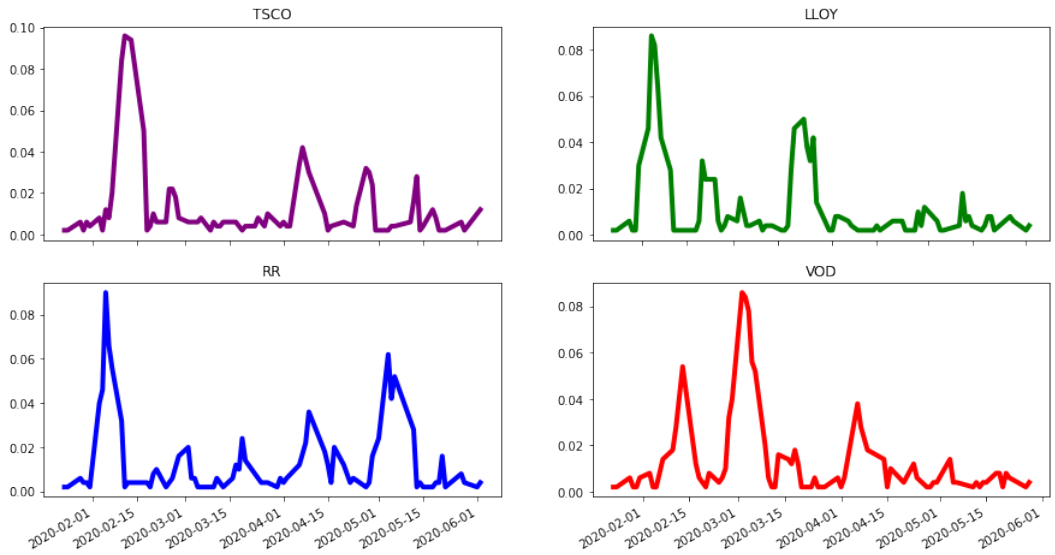
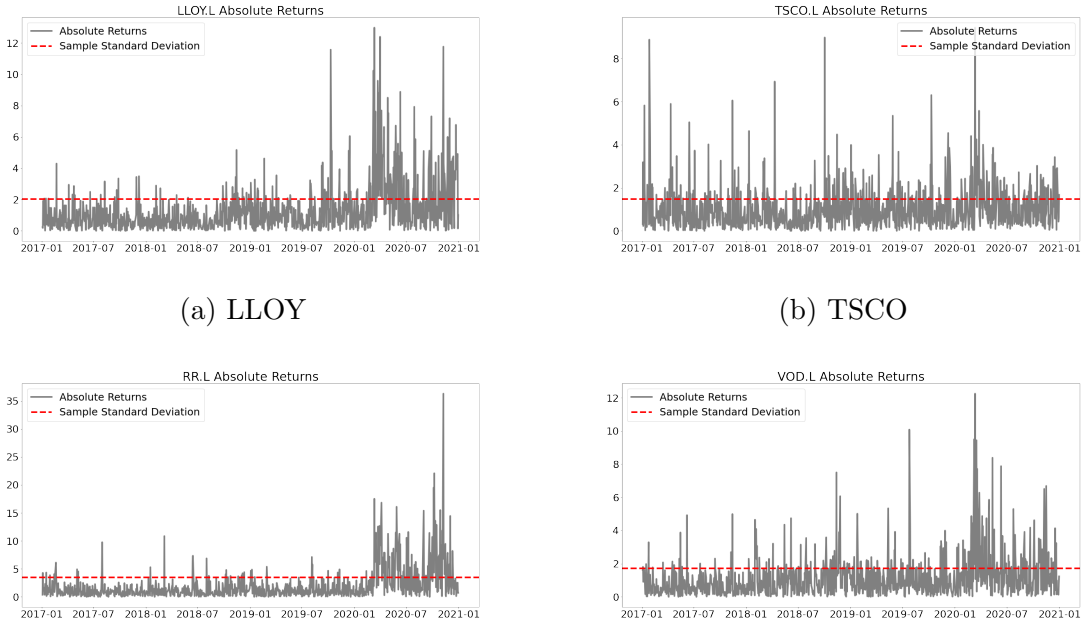


Figure 4.1: January 26th Irithmics Probabilities Example

4.1.2 Exploratory Data Analysis

Stock Returns

Before building and training any time series models, I first needed to understand and summarise the properties of each of the chosen companies continuous returns. This was guided by the properties of financial asset returns introduced in **Section 1.2.2**. Because this research is directly focused on volatility, my first step was to obtain the sample standard deviation of each stock's returns, and plot it over the absolute value of those returns **Figure 4.2**. I assessed the absolute value of the returns rather than the raw value for ease of visualization. The aim of these plots were to evaluate whether or not the returns showed signs of time varying volatility, and volatility clustering. It was immediately apparent that all four stock's returns visually displayed features of both.



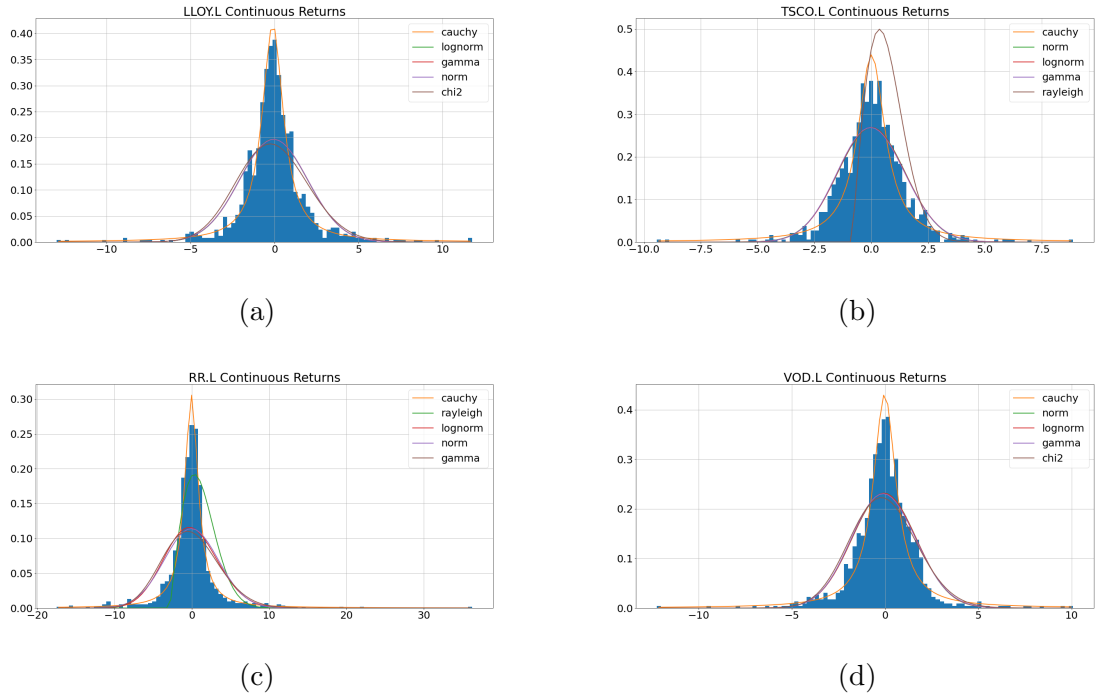
(a) LLOY (b) TSCO

(c) RR (d) VOD

Figure 4.2: Returns v Sample Standard Deviation

The next step in exploration was to identify the distribution of the returns. The first and second properties explained that the distribution of the returns should show more probability mass in the tails, as well as having a higher peak than a normal distribution. I implemented **Figure 4.3** by using the 'Fitter' Python library. This library implements a

kernel density estimation, histogram, as well as overlays other common distributions over the sample data. The distributions are fit using maximum likelihood estimation. Each of the four stocks exhibited the properties of distributions of asset returns differently, following different distributions. Nonetheless, the outcomes were all encouraging and prompted me to continue with the analysis.



(a) (b)

(c) (d)

Figure 4.3: Returns Distribution

The next step was to evaluate the third property of correlation. Since asset returns are said to have minimal to no autocorrelation in continuous returns, but significant autocorrelation in absolute or squared returns (16), I transformed the data into squared returns, and plotted the partial autocorrelation function outcome for Lloyd's as an example in **Figure 4.4**. The plots visually displayed minimal to no autocorrelation between raw returns, but significant partial autocorrelation in squared returns. This is very important to see, as intutively, this is illustrating how, for example, the returns of a stock today, are significantly dependent on the returns of the past 1-3 days. For conditional volatility models such as **GARCH**, this is of utmost necessity, as explained in **Section 3.2.2** the model depends on a specified number of days prior standard deviation and value in the time series.

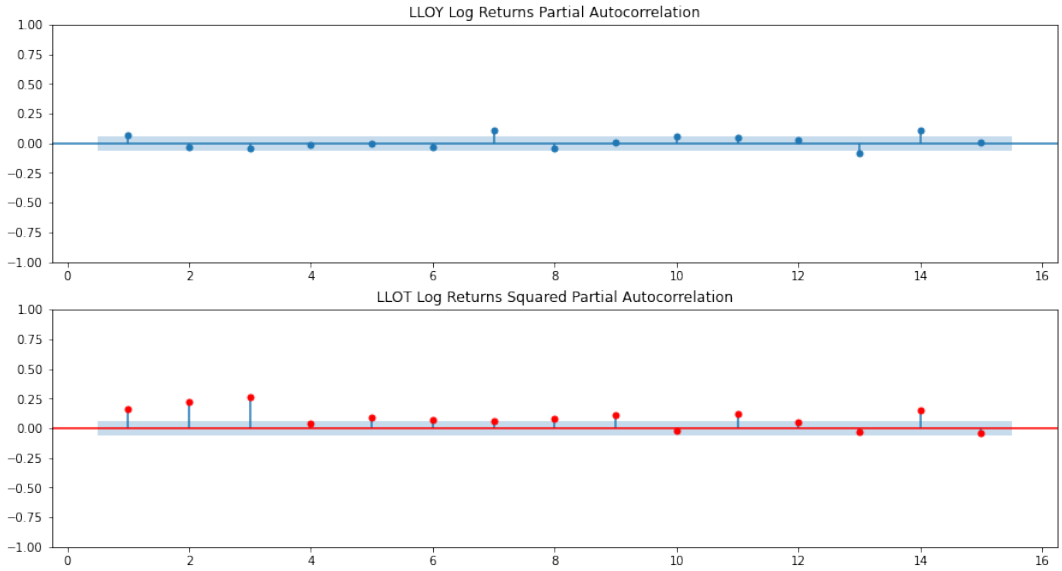


Figure 4.4: PACF: Normal v Squared Returns

Irithmics

The exploration of Irithmics data posed a greater challenge. As these data are non-common and had no prior properties, a more creative approach had to be undertaken. For starters, I returned to **Figure 4.1**, to get a sample visual representation of the properties of the data. It appeared there were signs of seasonality, or recurring external influences such as economic news or portfolio re-balancing. Something I had to take into great account when evaluating these data was trying not to over-fit models or ideas given the shape of one day's data. Each stock contained between 1-100 predictions for the same trading day, therefore keeping the analysis as generalisable as possible was paramount.

Preprocessing these data posed two significant challenges: 1) generalisation, 2)formatting. As generalisation has already been discussed, the formatting issue needed to be remedied as fitting a conditional volatility model with an exogenous covariate works similarly to a regression model. The datasets have to be of the same length, and in time series the indices have to match. This meant I had to find a way to aggregate the already aggregated Irithmics probabilities, such that each stock only had one probability of short selling for each trading day. I completed this by implementing a weight decaying algorithm. This

algorithm grouped the forecasts for each trading day (between 1-100 observations), and assigned a relative weight based on its temporal location. For example: the short/sell probability for March 25th has a prediction from all days January 1st - March 24th. To aggregate these forecasts, my goal was to mimic the concepts used in **GARCH** models, and transform the data into a single, univariate time series: the weight of the probabilities decayed based on its distance from the trading day, i.e. January 10th's prediction carries a lesser weight than March 10th's prediction for March 24th. The outcome of this algorithm is shown in **Figure 4.5** and the code can be found in the appendix.

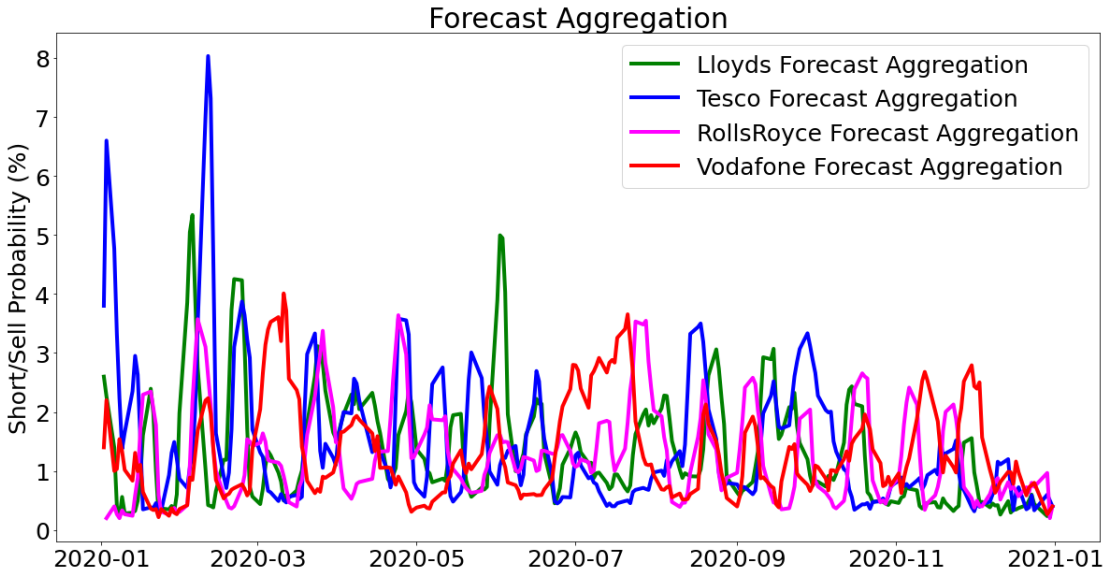


Figure 4.5: Irithmics Forecast Aggregation

4.2 Model Fitting

4.2.1 Strategy

When approaching the fitting and training of models I took a very structured approach. This was because each portion of the project directly built on the outcomes of the prior model. The necessity for consistency across univariate models and models with exogenous covariates cannot be understated as the comparison and impact of the addition of new information had to be the only change, otherwise there would be other factors affecting

the forecasted values. The strategy was as follows:

1. Implement grid search algorithm to search across combinations of parameters and hyper parameters
2. Fit many models through algorithm for each stock without the exogenous covariate of Irithmics data
3. Pick the best model based on Akaike Information Criterion (AIC)
4. Fit a Dynamic Conditional Correlation (DCC) Model to GARCH Volaility and Irithmics Probabilities
5. Fit the same model with an exogenous covariate
6. Forecast last 25 trading days of 2020
7. Compare generalisation error (MSE MAE)

4.2.2 Fitting

Before I fit any models, I first created a holdout data set in order to retain, albeit small, very important out-of-sample (OOS) data. This data is important because the training and fitting of the models cannot be done with a dataset, and then predict already seen data. These models hopefully can be as generalised as possible, and when fed in new data for future events will have predictive accuracy. Since the sample size was quite small, only one year, the holdout or test data was 25 trading days, about the month of December, 2020. This data is interesting because in the United Kingdom at the time, there was great uncertainty around the next lockdown which inevitably came, and therefore had high potential for greater levels of volatility.

Univariate

To search for the best model, I implemented a grid search algorithm in Python. A grid search's goal is to find optimal hyperparameters in a model, based on a specified loss function. In this case, the available hyperparameters were: Volatility (GARCH/EGARCH), the associated lags (p,q), and the residual distribution (students-t, skewed-t). The chosen loss function is the Akaike Information Criterion (AIC) which is calculated by

$$AIC \ = \ 2(K) - 2ln(L)$$

where **K** is the number of parameters and **L** is the likelihood. This loss function penalises more complex models as I look to minimise the AIC value. Based on this specification, the algorithm fit 72 different univariate models and returned a list of the top five and bottom five combinations based on this criterion (**Figure 4.6**).

Model	Volatility	Distribution	AIC
GARCH (1, 3)	EGARCH	skewt	3682.5
GARCH (1, 1)	EGARCH	skewt	3682.96
GARCH (1, 1)	EGARCH	t	3683.19
GARCH (1, 3)	EGARCH	t	3683.26
GARCH (3, 2)	GARCH	skewt	3683.46
...	...	...	...
GARCH (2, 3)	GARCH	normal	3786.54
GARCH (1, 1)	GARCH	normal	3786.69
GARCH (3, 3)	GARCH	normal	3788.47
GARCH (1, 2)	GARCH	normal	3788.69
GARCH (1, 3)	GARCH	normal	3790.69

(a) LLOY

Model	Volatility	Distribution	AIC
GARCH (1, 2)	EGARCH	t	3486.15
GARCH (2, 1)	EGARCH	t	3486.91
GARCH (2, 2)	EGARCH	t	3487.72
GARCH (3, 1)	EGARCH	t	3487.81
GARCH (1, 2)	EGARCH	skewt	3488.09
...	...	...	...
GARCH (1, 1)	GARCH	normal	3630.19
GARCH (2, 3)	GARCH	normal	3630.2
GARCH (1, 3)	GARCH	normal	3630.21
GARCH (3, 3)	GARCH	normal	3630.77
GARCH (1, 2)	GARCH	normal	3632.19

(b) TSCO

Model	Volatility	Distribution	AIC
GARCH (1, 3)	EGARCH	t	4386.07
GARCH (1, 3)	EGARCH	skewt	4387.87
GARCH (2, 3)	EGARCH	t	4388.07
GARCH (1, 1)	GARCH	t	4388.52
GARCH (1, 2)	EGARCH	t	4388.68
...	...	...	...
GARCH (1, 2)	GARCH	normal	4570.86
GARCH (2, 3)	GARCH	normal	4571.31
GARCH (1, 3)	GARCH	normal	4572.86
GARCH (2, 1)	EGARCH	normal	4573.79
GARCH (1, 1)	EGARCH	normal	4589.65

(c) RR

Model	Volatility	Distribution	AIC
GARCH (1, 2)	EGARCH	t	3486.15
GARCH (2, 1)	EGARCH	t	3486.91
GARCH (2, 2)	EGARCH	t	3487.72
GARCH (3, 1)	EGARCH	t	3487.81
GARCH (1, 2)	EGARCH	skewt	3488.09
...	...	...	...
GARCH (1, 1)	GARCH	normal	3630.19
GARCH (2, 3)	GARCH	normal	3630.2
GARCH (1, 3)	GARCH	normal	3630.21
GARCH (3, 3)	GARCH	normal	3630.77
GARCH (1, 2)	GARCH	normal	3632.19

(d) VOD

Figure 4.6: Grid Search

Dynamic Conditional Correlation

While the equations are quite complex, implementing this in Python was rather manageable. Though no available libraries, the researcher by the alias: (6) implemented this in their own project, which I used as a guide. The model took the stock returns and Irithmics probabilities as input, runs a univariate **GARCH** model on each series to obtain a conditional volatility estimate, then takes the conditional volatilities as input, into the DCC equation. For each stock and probability pair, the output is a single series containing the Dynamic Conditional Correlation **Figure 4.7**. The DCC gives information on the dynamic relationship between the two variables, and I hoped to see greater correlation in times of high volatility, which would prompt the model to have an increase in forecast accuracy since I will be fitting the Irithmics data as an exogenous regressor, a linear relationship would show itself here.

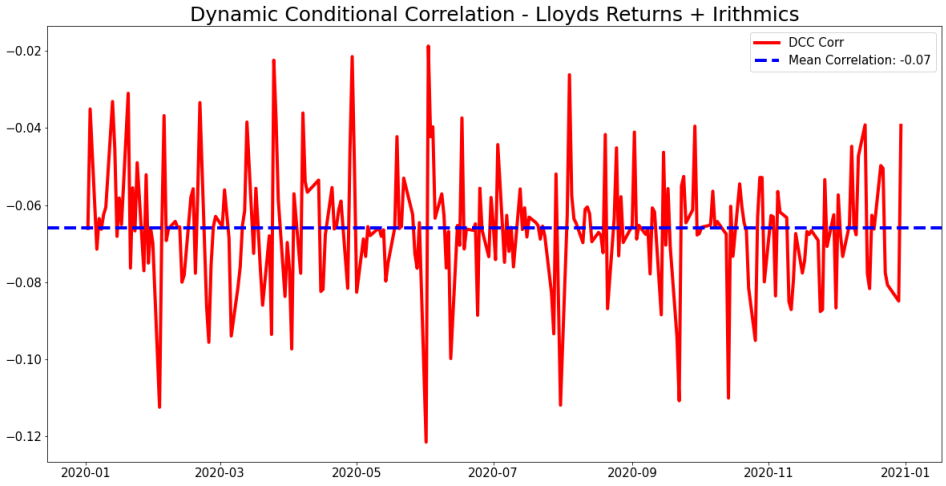


Figure 4.7: Lloyd's vs Irithmics DCC

Exogenous Covariate

In order to include the Irithmics transformed probabilities as an exogenous covariate within the conditional volatility model, I decided it was more advantageous to implement the model using the Rugarch library in R (10). This library allows for further specification

within the model, to include a submodel hyperparameter of "external.regressors". I passed a matrix object, (in this case it is a univariate series so a nx1), which is included in the variance function when the model is fit. In order to keep the comparison between the empirical model with no regressors, and the new model including the probabilities, I ensured the only difference between the two was the newly included data. Shown in **Figure 4.8**, the models fit the data using the training set up to the end of November (solid lines), and began the 1-day ahead rolling forecast using the out-of-sample test set (dashed lines). Because of the rigorous search for optimal hyperparameters in the univariate model, and the experimental constraints of keeping the same model there was no need to re-visit the grid search or selection criterion.

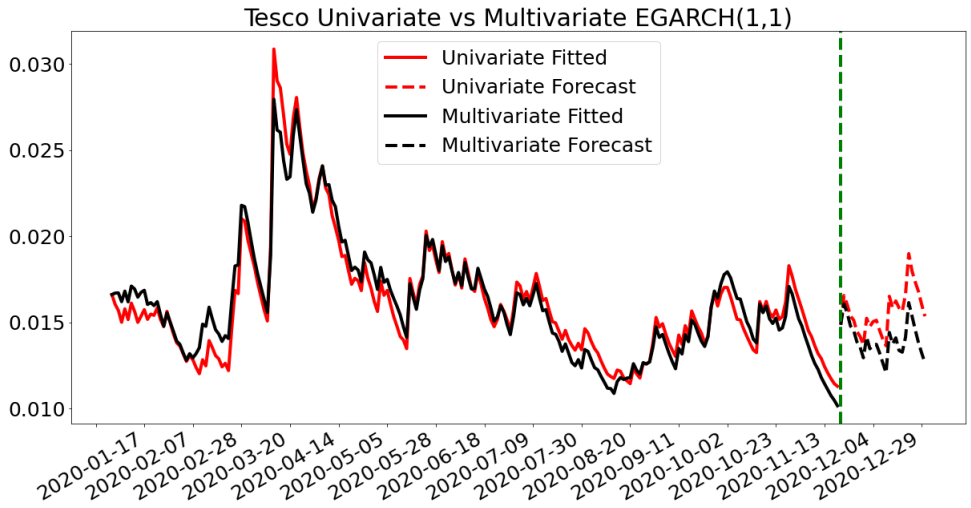


Figure 4.8: TESCO: Univariate vs Multivariate EGARCH(1,1)

4.3 Model Validation

While I did implement automated model selection methods, these are based on numeric criteria that do not necessarily address all components of a holistic research project. For each hyperparameter I included in my algorithm, I needed to validate the assumptions and limitations of including them in the model, as well as their relative impact on the forecast.

4.3.1 Residual Distributions

The first hyperparameter I wanted to validate after viewing the grid search output tables was the residual distribution. It was very apparent that in the Tesco and Vodafone models, the students-t distribution was yielding lower values for AIC. I wanted to ensure not only was this accurate, but also there were no better available alternative distributions I could have used. **Figure 4.9** was a method used for my validation process, where I, similarly to the distribution of returns, I fit using maximum likelihood estimation, the standardized residuals to multiple distributions. For example, here it was clear why the students-t distribution was performing better than the normal distribution as there was more mass in the tails and a taller peak. I repeated this process with all four stocks optimal models to ensure validity in my residual hyperparameter decision.

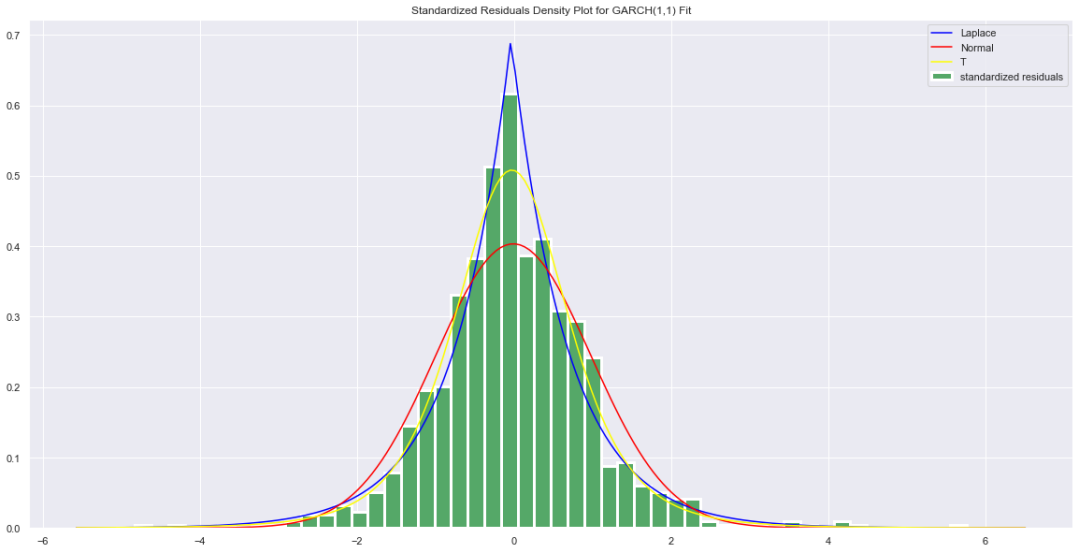


Figure 4.9: GARCH(1,1) Residuals

4.3.2 Volatility Model

I then evaluated why the different volatility models (**EGARCH vs GARCH**) faired differently. Different from the residual distribution, where a visual test was sufficient to gain understanding and confidence, the answer to this is was found through a better theoretical understanding of the **EGARCH** model. According to the creator of the

ARCH, Tim Bollserslev (3), he now speaks on the EGARCH: "There is a stylized fact that the EGARCH model captures that is not contemplated by the GARCH model, which is the empirically observed fact that negative shocks at time t-1 have a stronger impact in the variance at time t than positive shocks" (Engle) Fundamentally, this concept builds on the older theory of a leverage effect found in a $\mathbf{GJR-GARCH}$. Engle further explained: "negative shock is $\gamma - \alpha$, while the effective coefficient associated with a positive shock is $\gamma + \alpha$. In financial time series, we generally find that γ is negative and statistically significant" (Engle). Fundamentally, the **EGARCH** model is able to capture the properties of market returns slightly better than the **GARCH**, and Engle believes this is because the negative shocks have a larger and more significant affect on the next time periods variance than a positive shock. Intuitively this makes sense, as investor behavior tends to be more likely to sell a stock given bad news than buy an equal amount of stock given good news. Given this information I was comfortable with the results of the automated model selection.

4.3.3 Lags (p,q)

Finally, the evaluation of the lag parameters p and q were validated. I approached this problem in two ways, first was through the Partial Autocorrelation Function plots introduced in section one, evaluating the number of lags with significant autocorrelation $mathbfFigure$4.4. This didn't necessarily give an indication of the number of lags the standard deviations had autocorrelation with, but it did give an indicator to the the autocorrelation between the actual time series observations. The second approach was to evaluate the significance of model parameters. In some cases the minimum AIC model would recommend a more complex model, say $\mathbf{EGARCH(3,2)}$, but the parameters for the coefficients of $\alpha_2,\ \alpha_3,\ \beta_2$ were not significant at the 0.05 level. There is also literature on the topic of models extending the general p=1,q=1 framework, such as Peter Hansen and Asger Lunde's (12) which explained not only was the forecast accuracy showing no significant difference, but the model complexity actually hurt the analysis not helped.

For this reason, whenever I saw a minimal difference in AIC, I always chose the simpler model.

4.3.4 Forecasting

Since I am focusing on the 1-day ahead "Nowcast" I evaluated two forecasting methods: rolling window and fixed window. The rolling window has a fixed amount of days used in the model, ex: 100 day rolling the model starts on January 1st, uses 100 days of data and forecasts April 11th, the next forecast data begins on January 2nd uses data until April 11th and forecasts April 12th. Conversely, the expanding window forecast continues to aggregate data and the sample used to make the forecast gets larger with each step. An example of the difference in forecasts is shown in **Figure 4.10**. For this portion of validation there was no objective "better" option, as both have strengths and weaknesses, so I decided to move forward with the forecast method yielding the least generalisation error.

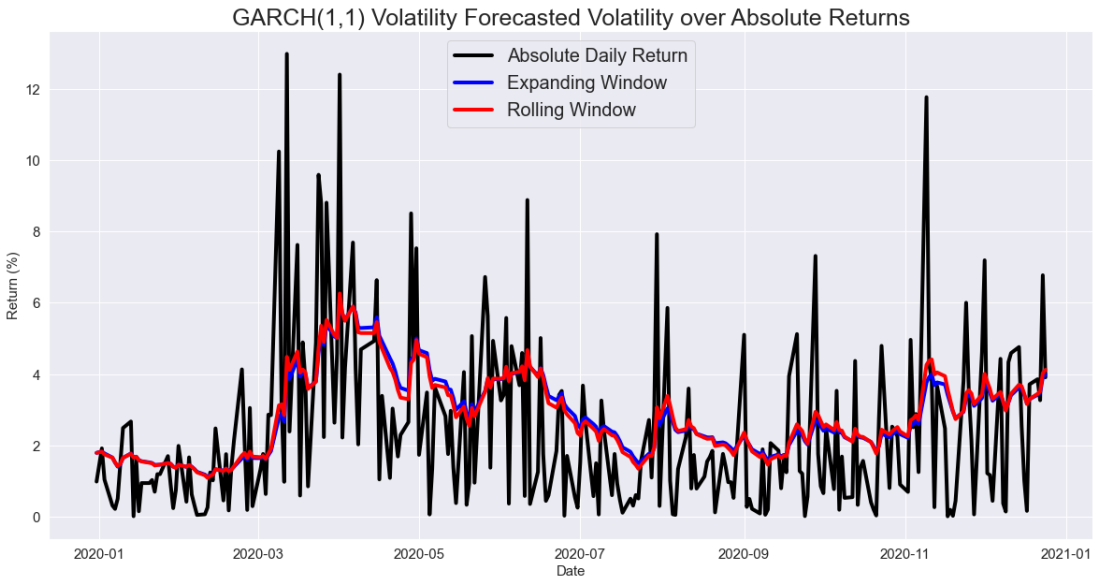


Figure 4.10: Rolling vs Expanding Forecast

4.4 Model Comparison

Finally, I needed to determine a consistent method for comparing the accuracy of the forecasts. As noted prior, since it is not possible to extract the exact instantaneous standard deviation from the stock returns, a generally accepted proxy instead is the squared or absolute value of the selected returns, therefore I decided to compare the forecast conditional volatility to squared returns. The comparison was done using two metrics: Mean Absolute Error and Mean Squared Error.

$$MAE\ (Mean\ Absolute\ Error) = \sum |(\mathbf{X} - \hat{\mathbf{X}})|$$

$$MSE\ (Mean\ Squared\ Error) = \sum (\mathbf{X} - \hat{\mathbf{X}})^2$$

The MAE and MSE simply give a generalisation error that can be compared across models to asses accuracy. These metrics will be used on the 1-day ahead rolling forecasts for the 25 day test dataset.

Chapter 5

Results

5.1 Univariate Model

Using the grid search algorithm for automated model selection, each of the four stocks' returns yielded varying results. **Figure 5.1** shows the fitted model parameters for the two most optimal models, based on the minimization of the selected loss function: Akakie's Information Criterion (AIC). After the automated selection, I manually selected the model I would choose to implement based on significant parameters, model complexity, and residual distribution. As explained in **Section 4.3**, there is more to a "best" model than the minimization of a loss function, so to ensure the best results I took a greater holistic approach to selection.

5.1.1 Lloyd's

The best models for Lloyds were **EGARCH(1,1), skewed-t EGARCH(1,3), skewed-t**. The two had negligible differences in AIC, but looking at the parameters, the added coefficients to the **(1,3)** model not only were insignificant, but it rendered the intercept and β_1 coefficient insignificant as well (p-value < 0.05). For this reason I chose the

EGARCH(1,1) model, simpler and more significant parameters.

5.1.2 Tesco

The best models for Tesco were **EGARCH(1,2), students-t** and **EGARCH(2,1), students-t**. Model selection in this case was quite difficult, though the models themselves wouldn't produce overly different results. Both had 3 significant non-slope parameter, as well as the distribution parameter, while having insignificant intercepts. Because of the negligible difference in parameters, I defaulted to the automated model selection choice of $\mathbf{EGARCH}(\mathbf{1},\mathbf{2})$ $\mathbf{students-t}$ because of its minimum AIC value.

5.1.3 Rolls Royce

The best models for Rolls Royce were **EGARCH(1,3), students-t** and **EGARCH(1,2), skewed-t**. The two had negligible differences in AIC, and no different in model coefficients (barring rounding error). The difference was the distribution of the residuals. To make a decision on which model, I fit using Maximum Likelihood Estimation the residuals to a distribution and found that again there was negligible difference. I chose the $\mathbf{EGARCH}(\mathbf{1},\mathbf{3})$, $\mathbf{students-t}$ model because it had the minimum AIC.

5.1.4 Vodafone

The best models for Vodafone were **EGARCH(3,1), students-t** and **EGARCH(1,2), students-t**. The two had negligible differences in AIC, but looking at the parameters, the added coefficient to the (1,3) was not significant α_3 and neither was ω the intercept. For this reason I chose the $\mathbf{EGARCH}(\mathbf{1},\mathbf{2})$, $\mathbf{students-t}$ model, simpler and more significant parameters.

Lloyds Banking Group: EGARCH(p: 1, q: 1)

	Coefficient	Estimate	P_Value
0	mu	-0.038998	0.32
1	omega	0.025534	0.023
2	alpha[1]	0.186841	0.00017
3	beta[1]	0.990865	0.0
4	eta	4.479581	1.1e-12
5	lambda	0.061294	0.11

Lloyds Banking Group: EGARCH(p: 1, q: 3)

	Coefficient	Estimate	P_Value
0	mu	-0.037789	0.36
1	omega	0.033559	0.075
2	alpha[1]	0.248207	0.038
3	beta[1]	0.799341	0.52
4	beta[2]	0.000000	1.0
5	beta[3]	0.189241	0.92
6	eta	4.449721	1.4e-12
7	lambda	0.062731	0.1

(a) LLOY

Tesco: EGARCH(p: 1, q: 2)

	Coefficient	Estimate	P_Value
0	mu	0.019132	0.61
1	omega	0.081746	0.26
2	alpha[1]	0.232175	0.024
3	beta[1]	0.284122	0.0033
4	beta[2]	0.623739	4.8e-06
5	nu	4.531043	1.3e-11

Tesco: EGARCH(p: 2, q: 1)

	Coefficient	Estimate	P_Value
0	mu	0.022266	0.55
1	omega	0.028161	0.12
2	alpha[1]	0.282378	0.00038
3	alpha[2]	-0.187119	0.019
4	beta[1]	0.970041	0.0
5	nu	4.451065	8.4e-12

(b) TSCO

Rolls Royce: EGARCH(p: 1, q: 3)

	Coefficient	Estimate	P_Value
0	mu	-0.032459	0.5
1	omega	0.067184	0.16
2	alpha[1]	0.361788	0.028
3	beta[1]	0.750370	0.69
4	beta[2]	0.000000	1.0
5	beta[3]	0.232988	0.85
6	nu	3.723359	9e-13

Rolls Royce: EGARCH(p: 1, q: 3)

	Coefficient	Estimate	P_Value
0	mu	-0.042737	0.43
1	omega	0.066426	0.16
2	alpha[1]	0.360949	0.023
3	beta[1]	0.748247	0.68
4	beta[2]	0.000000	1.0
5	beta[3]	0.235371	0.84
6	eta	3.725496	1.3e-12
7	lambda	-0.016992	0.68

(c) RR

Vodafone: EGARCH(p: 3, q: 1)

	Coefficient	Estimate	P_Value
0	mu	-0.013606	0.71
1	omega	0.024493	0.091
2	alpha[1]	0.389048	9.4e-07
3	alpha[2]	-0.320546	0.037
4	alpha[3]	0.053841	0.63
5	beta[1]	0.988683	0.0
6	nu	3.630565	4.2e-18

Vodafone: EGARCH(p: 1, q: 2)

	Coefficient	Estimate	P_Value
0	mu	-0.009822	0.78
1	omega	0.060619	0.034
2	alpha[1]	0.285952	0.00065
3	beta[1]	0.301537	0.041
4	beta[2]	0.665829	1.1e-05
5	nu	3.708528	2e-18

(d) VOD

Figure 5.1: Best Univariate Model Parameters

5.1.5 Forecast Performance

The resultant forecasts from these models yielded different results, and it became apparent that the fit and forecasts for Tesco and and Vodafone in both fixed and expanding window methodology were superior to that of Rolls Royce and Lloyd's. For the most accurate two models, the expanding window was more accurate than the fixed window. A great point of concern in the model fititng was the serious issues with the accuracy of the model for Rolls Royce returns, interestingly, the performance of the **EGARCH(1,3)** was worse than that of a model that simply took the mean of the information set's standard deviations. To interpret the table output, the scale of the chart was in % so the best model, Vodafone, found itself within 2.21% of the volatility proxy of squared centered returns on average. Though the models were simple, only including returns without any other information, a volatility forecast for 25 days with a mean absolute error of less than 2.4% I believe to be very encouraging. For Tesco and Vodafone, the volatility forecasts are quite accurate and building complexity and information on top of these models could be quite advantageous.

To continue work with Lloyd's and Rolls Royce, there is greater tuning to be done, these models are pretty much useless and should not be used moving forward.

	Company	MSE	MAE
0	Lloyds	156.901696	10.445096
1	Tesco	9.283741	2.321513
2	RollsRoyce	1876.738343	37.977320
3	Vodafone	7.298816	2.209078

(a) Expanding Window

	Company	MSE	MAE
0	Lloyds	151.005473	10.378395
1	Tesco	9.444414	2.334610
2	RollsRoyce	1904.449275	38.376101
3	Vodafone	9.316594	2.563449

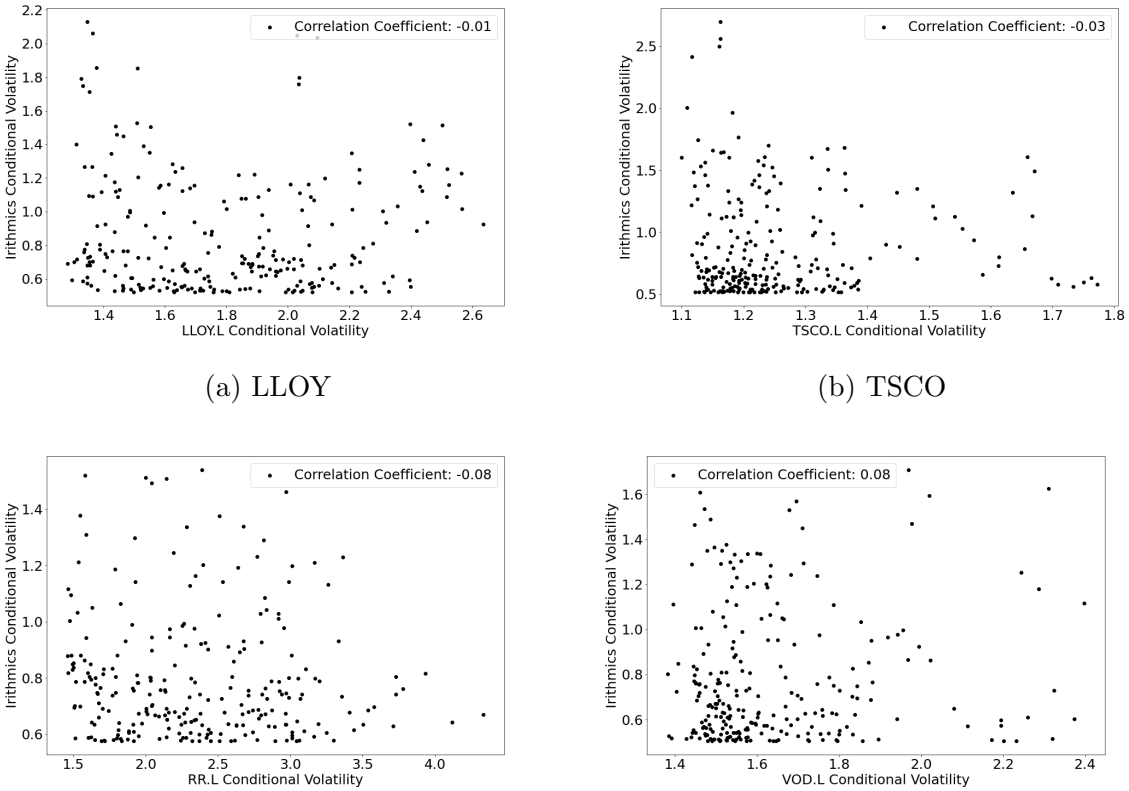
(b) Fixed Window

Figure 5.2: Forecast Generalization Error

5.2 Dynamic Conditional Correlation Model

Implementing the DCC model on the model volatilities and the Irithmics conditional volatility did not immediately lead to a visualization or indication of any encouraging relationships between the Irithmics predicted short/sell probabilities and the stock returns. For the four stocks, the standard Pearson correlation coefficient between the two variables was about 0.07 - 0.09. Then using the dynamic model, I hoped to see if there were any events during the time series that there were significantly higher times of correlation, but unfortunately I never found this to be proven true. The highest correlated stock with Irithmics probabilities was Lloyd's, with its maximum conditional correlation was the beginning of February and June, but this value only reached 0.12. Ideally, this value would be much higher, as the next step in this analysis was to incorporate this value as an exogenous covariate implemented through linear regression. Theoretically all was not lost, as with financial applications any small fraction of increased accuracy can lead to large savings in a risk mitigation setting with investments, but greater correlation is obviously desired. To assist in the visualization over time, **Figure 5.3** plots the **EGARCH**$(\mathbf{1},\mathbf{1})$ volatility with the Irithmics conditional volatility to show the linear relationship in a scatter plot and includes the sample correlation coefficient. Again, these

results were not at all encouraging, but did not necessarily omit the prospect of slight increases in forecast accuracy by incorporating the exogenous data to the conditional volatility models (6).



(a) LLOY

(b) TSCO

(c) RR

(d) VOD

Figure 5.3: DCC Scatter Plot

5.3 Exogenous Covariate Model

5.3.1 Forecast Performance

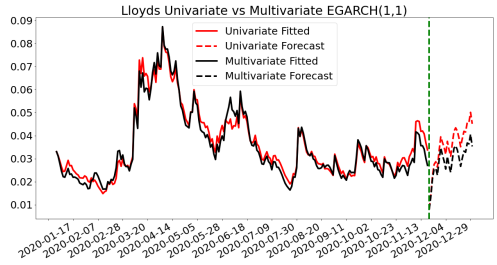
Figure 5.4 highlights the comparative performance of the conditional volatility models forecasts with and without the exogenous covariate.

MAE and MSE for Squared Centered Returns vs Volatility Forecast

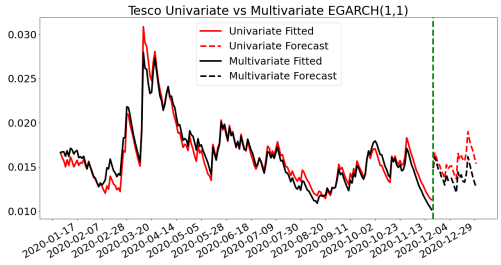
Company	Univariate MAE	Multivariate MAE	Univariate MSE	Multivariate MSE
Lloyds	13.2095	13.2145	360.2926	360.4560
Tesco	2.6225	2.6239	17.7236	17.7347
RollsRoyce	20.2473	20.2422	1912.0652	1911.8713
Vodafone	2.3284	2.3299	19.8681	19.8783

Figure 5.4: Multivariate Model Generalisation Error

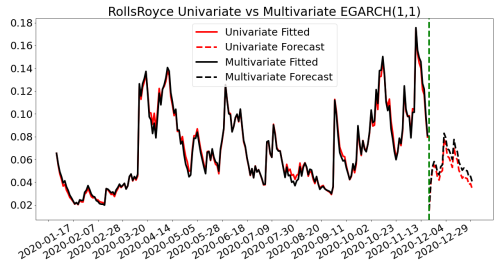
Similar to what would be expected, given the information the Dynamic Conditional Correlation model results, there was no significant change to the model performance when incorporating the exogenous covariate of Irithmics data. In fact, though only by a very small amount, the model's performances were less accurate. The strongest models, Tesco and Vodafone, decreased in accuracy by a very insignificant factor of less than 0.001, and the worst performing models of Rolls Royce and Tesco had impacts slightly greater, but still less than 1% different. **Figure 5.5** illustrates each models fitted values and rolling 1-day ahead nowcast.



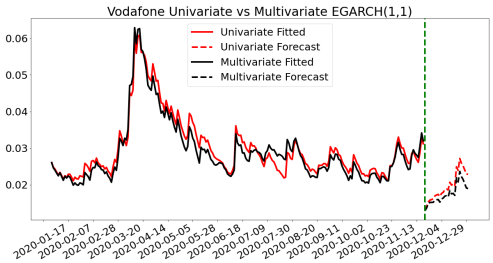
(a) LLOY



(b) TSCO



(c) RR



(d) VOD

Figure 5.5: Univariate vs Exogenous Forecasts

The intuitive interpretation of these results, is the Irithmics short probabilities, when incorporating into the volatility forecast acted almost exclusively as noise rather than

signal of the data generating process. Looking closer at the fitted and forecast plots in **Figure (5.5)**, there are different intensities of impact the exogenous data has on the fitted and forecast values. Sub-figures b and d seem to show more extreme peaks and valleys in the univariate data vs the data incorporating the exogenous data, where sub-figures a and c track the univariate model near identically.

Chapter 6

Discussion

6.1 Overview

In this dissertation, I set out to evaluate if a univariate conditional volatility forecast could be improved by adding an exogenous covariate to the already existing model. First, data was extracted from Yahoo Finance's database and transformed into continuous returns which were then modelled empirically using grid-search selected **EGARCH** models. Then, after receiving Irithmic's data provided, these data were transformed via the weight decaying algorithm. Once the data processing was complete, I trained the optimal models and forecasted and compared outputs. Following the univariate modelling, I implemented a Dynamic Conditional Correlation model to evaluate the relationship between the selected exogenous covariate and the modelled conditional volatility. Though yielding mediocre results, I continued with the final process of fitting an **EGARCH** model with an exogenous covariate. Disappointingly, the results of the model were not improved, though a greater understanding of conditional volatility modelling, exogenous data model integration, and many ideas for future work were derived. This project approached a well researched area of volatility modelling, with a novel approach of taking a tangentially related piece of non-standard data, standardizing the format in a generalisible fashion

and implemented it using empirical methods.

6.2 Future Work

There is great opportunity for further work on this topic. First, I think there is great potential for further incorporating exogenous variables to conditional volatility forecasts. There is research in the area, but has plenty of room for continued work. I think incorporating linear and non-linear models as an extension to **EGARCH** models could be very powerful. Of course there is the balance in statistics between interpretation and prediction, but if one is trying to simply improve forecast accuracy, finding exogenous data sets (could be very large data) to accompany the empirical model could be very powerful. I also think there is opportunity for great model improvements using the Dynamic Conditional Correlation models in a similar study. Given the constraints, I was only able to implement it once, but the possibility for incorporating more data such as interest rates, the FTSE 100 index, or an aggregate of the sector the targeted stock is in. Second, I believe there is much greater opportunity to explore avenues to more effectively use the Irithmics data. Though it was not proved in this project, the power of the Wisdom of the Crowds cannot be understated. Having information about the predicted behavior of trillion's of pounds of assets moving in Britain has many use cases beyond just volatility predictions. I believe a possible greater aggregation of the data across the entire FTSE 100, rather than a selected group of stocks could yield better results, as the greater grouping could smooth out anomalies and increase predictability. Lastly, there is great room to contribute to the Python and R time series open source community. As the research transcends traditional models, there is great opportunity to contribute to development of time series modelling libraries of greater complexity (rugarch in R, arch in Python). I have greatly enjoyed the artistry of programming and developing models and am greatly excited by the opportunity to contribute further to the literature in Financial Time Series.

6.3 Limitations

As this research completed for an M.Sc. Dissertation, there were limitations to completing a fruitful study. Firstly was the datset size. As Irithmics graciously donated these data free of charge, to explore a novel concept, I did not have 1000's trading day history for each stock, instead about 252. This lead to challenges in fitting and training a model with holdout data for out of sample testing, to identify generalization error and tune hyperparameters, while balancing the amount of data for a large sample to train with, though this can also be attributed to the challenges of working with time series data. For example, traditional methods for training neural networks or tree models like bootstrapping or k-fold cross validation are not available as testing data outside the temporal constraint is not useful. Second, the constraint of time. Given this project was completed in 2-3 months, I had to make decisions and pursue ideas very quickly and there was no time to restart or take a step back, read full textbooks and many papers. Lastly, the constraint of the exogenous data itself. The Irithmics data is the corporation's proprietary models prediction of what the aggregate decision of the funds to short/sell a stock. The maximum probability was about 0.07, so even if 7% of the 250,000 funds decided they would sell the stock, depending on the individual portfolio weights, and actions of other institutions, macroeconomic events and other investors, there was no reason to say this will increase the instantaneous volatility of the stock on that day.

Bibliography

(2003). *Press Release*.

Ali, G. (2013). Egarch, gjr-garch, tgarch, avgarch, ngarch, igarch and aparch models for pathogens at marine recreational sites. *Statistical and Econometric Methods*, 2(3):57–73.

Bollerslev, T. (1986). Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31(3):307–327.

Carmona, R. (2014). *Statistical Analysis of Financial Data in R*. Springer, New York Heidelberg Dordrecht London.

Chuge, L. (2020). An empirical study of hang seng index based on garch model. In *2020 2nd International Conference on Economic Management and Model Engineering (ICEMME)*, pages 654–657.

ΔirtyQuant (2021). *Introduction to DCC - Dynamic Conditional Correlation Models*. Manual Python Library.

Engle, R. V-lab: Exponential garch volatility documentation.

Engle, R. F. (1982). Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica*, 50(4):987–1007.

Engle, R. F. (2002). Dynamic conditional correlation – a simple class of multivariate garch models. *Business and Economic Statistics*.

Ghalanos, A. (2022). *rugarch: Univariate GARCH models.* R package version 1.4-8.

Grant, F. (2022). About irithmics.

Hansen, P. R. (2005). A forecast comparison of volatility models: Does anything beat a garch(1,1)? *Journal of Applied Econometrics*, 20(3):873–899.

Hayes, A. (2021). Volatility.

Hayes, A. (2022). Portfolio management definition.

McKinney, W. et al. (2010). Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX.

Popov, V. (2022). Lecture notes in quantitative risk management.

ritvikmath (2020). *GARCH Model : Time Series Talk.* YouTube Video.

Sen, J., Mehtab, S., and Dutta, A. (2021). Volatility modeling of stocks from selected sectors of the indian economy using garch. In *2021 Asian Conference on Innovation in Technology (ASIANCON)*, pages 1–9.

Sánchez García, J. and Cruz Rambaud, S. (2022). A garch approach to model short-term interest rates: Evidence from spanish economy. *International Journal of Finance & Economics*, 27(2):1621–1632.

The Wisdom of Crowds (2004). The wisdom of crowds — Wikipedia, the free encyclopedia. [Online; accessed 06-July-2022].

Van Rossum, G. and Drake, F. L. (2009). *Python 3 Reference Manual.* CreateSpace, Scotts Valley, CA.

Selected Figures

Appendix

Python & R Code

I used a mixture of Python and R code to exploit the strengths of each language. Specifically I found R to be more useful for data processing, as the tidyverse allows similar commands to a SQL environment, as well as rugarch for incorporating exogenous variables. Code can be found on the following pages:

Data Preparation in R

```
read_excel_allsheets <- function(filename, tibble = FALSE) {
  # I prefer straight data.frames
  # but if you like tidyverse tibbles (the default with read_excel)
  # then just pass tibble = TRUE
  sheets <- readxl::excel_sheets(filename)
  x <- lapply(sheets, function(X) readxl::read_excel(filename,
                                                     sheet = X,
                                                     col_names = c(paste0("Date"),
                                                                   paste0("Prob"))))
  if(!tibble) x <- lapply(x, as.data.frame)
  names(x) <- sheets
  x
}
###################### Reading In Data ############################
getFile = function(){
  require(tidyverse)
  require(rvest)
  require(tidyverse)
  link = "https://www.fidelity.co.uk/shares/ftse-100/"
  page = read_html(link)
  ftse = page %>%
    html_nodes("td") %>%
    html_text()


  ftse = do.call(rbind.data.frame,
                 split(ftse, ceiling(seq_along(ftse)/3)))
  colnames(ftse) = c("Symbol", "Name", "Sector")
  ftse$Sector = as.factor(ftse$Sector)

  files = list.files("C:/Users/zaneh/OneDrive/Documents/St_Andrews_2021/Dissertation/Data_Files")
  fileList = sub(".XLON.xlsx", "", files)[-1][1:(length(files)-2)]

  i <- menu(fileList, graphics=TRUE, title="Choose company")

  symb = fileList[i]

  path = paste0("C:/Users/zaneh/OneDrive/Documents/St_Andrews_2021/Dissertation/Data_Files/",
                symb,
                ".XLON.xlsx")
  df = read_excel_allsheets(path)


  return(list("Data" = df,
              "Ticker" = symb,
              "Company" = ftse$Name[which(ftse$Symbol == symb)]
  ))
}
#################### Creating the string of Data ##########################

companySelection = getFile()
```

```
dataString = c()
for(i in 1:length(companySelection$Data)){
  for(j in 1:100){
    if(is.na(companySelection$Data[[i]][[1]][j])){
      break
    }else{
      dataString = c(dataString,
                     companySelection$Data[[i]][[1]][j],
                     companySelection$Data[[i]][[2]][j],
                     names(companySelection$Data[i]))
    }
  }

}


####################Data Cleaning#################################
companyDF =
  do.call(rbind.data.frame,
          split(dataString, ceiling(seq_along(dataString)/3)))


colnames(companyDF) = c("shortDate", "Prob", "forecastDate")

companyDF = companyDF %>%
  mutate(forecastDate = as.Date(companyDF$forecastDate),
         shortDate = as.Date(companyDF$shortDate),
         Prob = as.numeric(companyDF$Prob),
         shortMonth = factor(month.name[lubridate::month(lubridate::floor_date(shortDate, 'month'))],
                             levels = month.name)) %>%
  dplyr::select(forecastDate,shortDate,shortMonth,Prob )


############## Getting Returns ##############################
require(cowplot)
require(gridGraphics)
require(svMisc)
require(quantmod)
require(dplyr)
require(tidyverse)
require(tseries)
require(rugarch)
require(xts)
require(PerformanceAnalytics)
require(fGarch)
require(sgt)
require(MASS)
require(gridExtra)
require(zoo)
require(forecast)
symb = paste0(companySelection$Ticker,".L")
startDate = "2017-01-01"
endDate = "2020-12-31"
```

```
sharePrice = get.hist.quote(symb,
                            start = startDate,
                            end = endDate,
                            quote = 'Close',
                            quiet = TRUE)

returns = (diff(log(sharePrice$Close))) %>%
  na.omit()
numericReturns = as.numeric(returns) %>%
  na.omit()
hmm = companyDF %>%
  dplyr::select(shortDate, Prob) %>%
  arrange(shortDate)

count_obs = c(1)
for(i in 2:nrow(hmm)){
  if(hmm$shortDate[i-1] == hmm$shortDate[i]){
    count_obs = c(count_obs, (count_obs[i-1]+1))
  }else{
    count_obs = c(count_obs, 1)
  }

}


hmm$cumsum = count_obs

totalShorts = hmm %>%
  group_by(shortDate) %>%
  summarise(x = sum(cumsum))


merge_for_weight = merge(x = hmm, y = totalShorts, by = "shortDate")
merge_for_weight$weighted_prob = merge_for_weight$cumsum/merge_for_weight$x
merge_for_weight$new_prob = merge_for_weight$weighted_prob*merge_for_weight$Prob

scaled_observations = merge_for_weight %>%
  group_by(shortDate) %>%
  summarise(probability = sum(new_prob))

ag = 0
laggedProb = c(rep(NA,lag),tail(scaled_observations$probability,(254-lag)))
laggedSD = scaled_observations$sd_check

aggregateVolatility = data.frame('Date' = tail(scaled_observations$shortDate, 253),
                                 'returns' = tail(as.numeric(returns),253),
                                 'volForecast' = tail(numForc,253),
                                 'shortProb' = tail(laggedProb,253),
                                 'shortSD' = factor(tail(laggedSD,253)),
                                 'week' = format(as.Date(zoo::index(tail(scaled_observations$shortDate,

volSD = sd(aggregateVolatility$volForecast)
volMean = mean(aggregateVolatility$volForecast)
```

```
#Daily Aggregated Volatility Incorporating Irithmics into GARCH

aggregateVolatility = aggregateVolatility %>%
  mutate(scaleFactor = ifelse(shortSD == "Mean",
                  -.5*volSD,
                  ifelse(shortSD=='SD',
                         volSD,
                         ifelse(shortSD== "SD*2",
                                2*volSD, 3*volSD
                         ))),
         )
dayOffset = 0
aggregateVolatility$scaleFactor = c(rep(0,dayOffset), tail(aggregateVolatility$scaleFactor,(nrow(aggreg

aggregateVolatility$scaledVolatility = aggregateVolatility$volForecast+aggregateVolatility$scaleFactor
```

Importing Stock Data (Python Portion)

```
### Imports and Settings
import datetime as dt
import sys
import numpy as np
from numpy import cumsum, log, polyfit, sqrt, std, subtract, mean
from numpy.random import randn
import pandas as pd
from pandas_datareader import data as web
import seaborn as sns
from pylab import rcParams
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import matplotlib.dates as mdates
from arch import arch_model
from numpy.linalg import LinAlgError
from scipy import stats
import statsmodels.api as sm
import statsmodels.tsa.api as tsa
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import acf, q_stat, adfuller
from sklearn.metrics import mean_squared_error, mean_absolute_error
from scipy.stats import probplot, moment
from arch import arch_model
from arch.univariate import ConstantMean, GARCH, Normal
from sklearn.model_selection import TimeSeriesSplit
from tabulate import tabulate
import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter('ignore')
import dataframe_image as dfi
from IPython.display import display, HTML
```

```
stockList = ['LLOY.L', 'TSCO.L', 'RR.L', 'VOD.L']
start = pd.Timestamp('2017-01-01')
end = pd.Timestamp('2020-12-31')
prices = [0 for x in range(len(stockList))]
z = 0
for i in stockList:
    priceData = web.DataReader(i, 'yahoo', start, end)\
      [['Close']]
    priceData['logReturn'] = np.log(priceData['Close']).diff().mul(100)
    priceData = priceData.dropna()
    prices[z] = priceData
    z+=1
```

Distributions and Standard Deviations

```
for i in range(4):
    plt.figure(figsize=(20,10), facecolor=(1,1,1))
    plt.plot(abs(prices[i]['logReturn']),
             color = 'grey',
             linewidth = 4,
             label = 'Absolute Returns')
    plt.axhline(y = prices[i]['logReturn'].std(), color = 'red', linestyle = '--', linewidth = 4,
              label = 'Sample Standard Deviation')
    plt.title(f'{stockList[i]} Absolute Returns',
              fontsize = 30)
    plt.legend(fontsize = 25)
    plt.xticks(fontsize = 25)
    plt.yticks(fontsize = 25)
    #plt.savefig('figures\plot {0}.png'.format(i+1))
    plt.close()
```

ACF Plots

```
for i in range(4):
    figure, axis = plt.subplots(2, 1,figsize=(15,8),facecolor=(1, 1, 1))

    plot_pacf(prices[i]['logReturn'],
              zero = False,
              ax = axis[0],
              lags = 15,
             title = f'{stockList[i]} Log Returns Partial Autocorrelation')
    plot_pacf(prices[i]['logReturn']**2,
              zero = False,
              ax = axis[1],
              lags =15,
              color = 'red',
              title = f'{stockList[i]} Log Returns Squared Partial Autocorrelation')
```

```
    plt.close()
    #plt.savefig(f'figures\stock{i}')
```

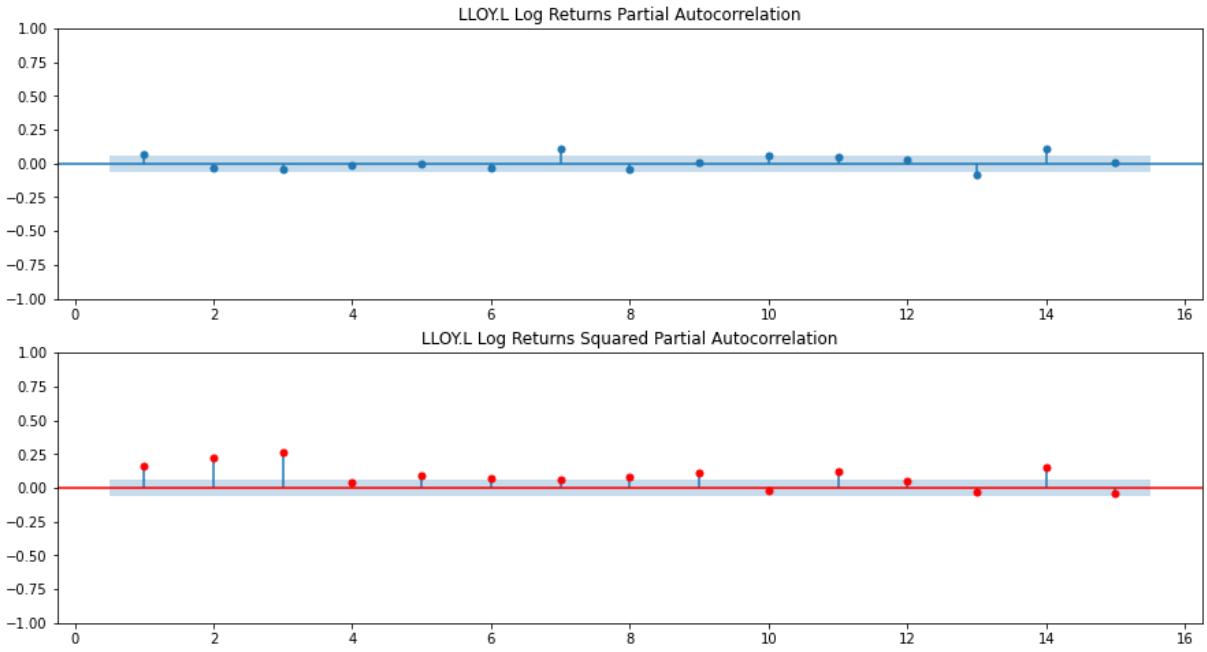


Figure 1: png

Initial Model Fitting

```
col_names = ["Model", "Volatility", "Distribution", "AIC"]

for z in range(len(stockList)):
    aic = [0 for a in range(72)]
    dist = [0 for a in range(72)]
    model = [0 for a in range(72)]
    vol = [0 for a in range(72)]
    x = 0
    for i in range(0,4):
        for j in range(1,4):
            for k in ['normal', 't', 'skewt', 'ged']:
                for l in ['GARCH', 'EGARCH']:
                    if i == 0 & j == 0:
                        next
                    else:
                        mdl = arch_model(prices[z]['logReturn'],
                                         p = i,
                                         q = j,
                                         mean = 'constant',
                                         vol = l,
```

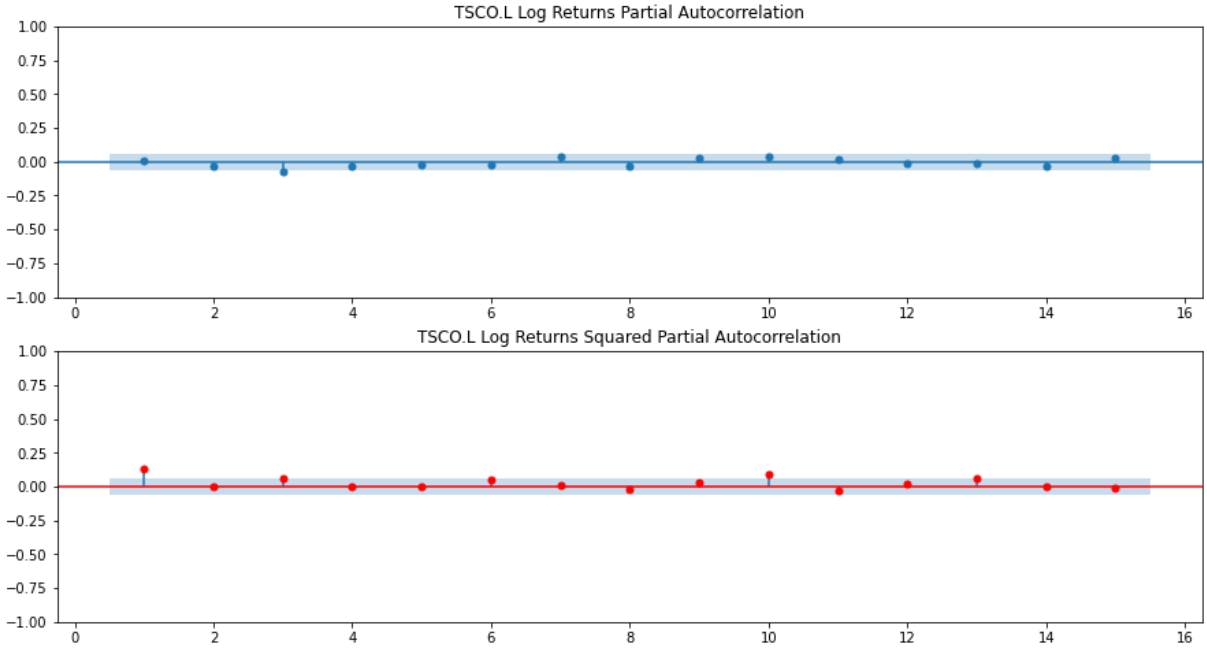


Figure 2: png

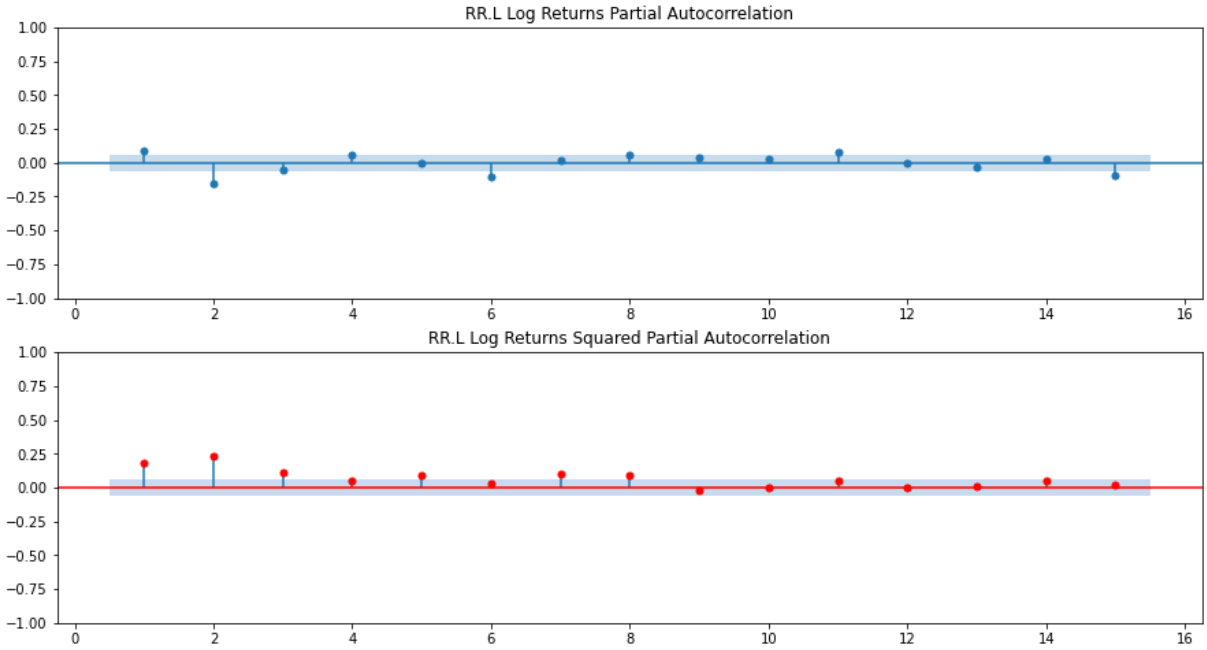


Figure 3: png

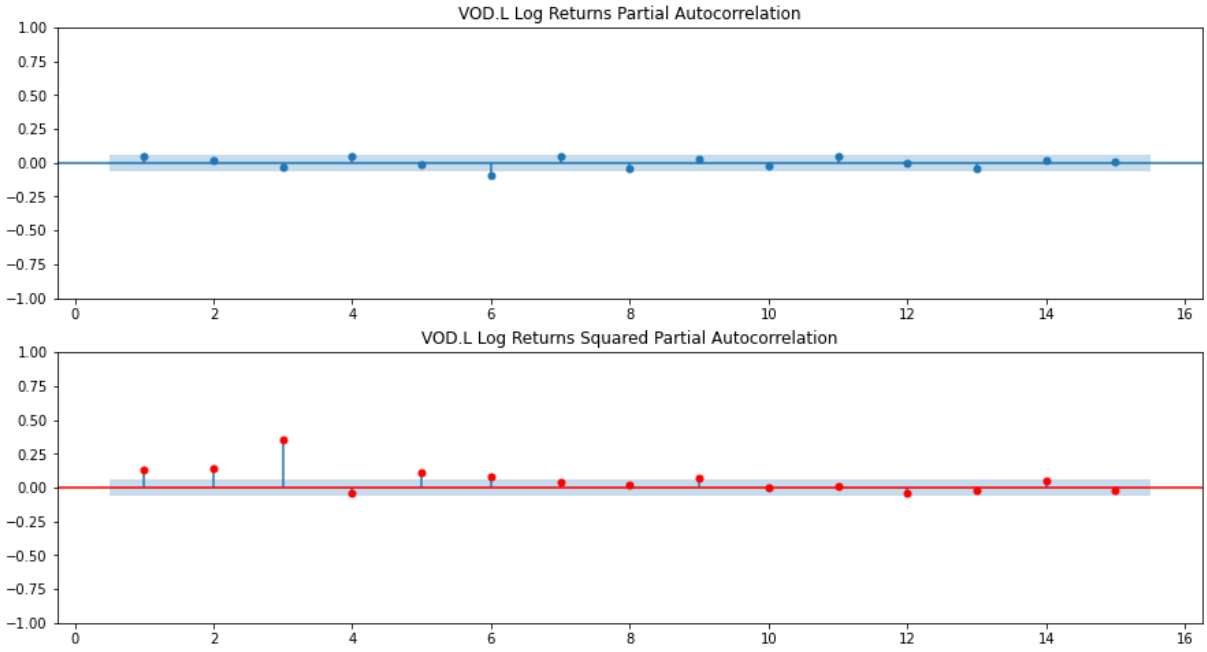


Figure 4: png

```
                                        dist = k)
                    res = mdl.fit(disp = 'off')
                    #print(f'GARCH({j},{i}) {k} AIC: {res.aic}')
                    aic[x] = str(res.aic)
                    vol[x] = str(l)
                    dist[x] = str(k)
                    model[x] ='GARCH (%s, %s)' %(j,i)
                    x = x+1
aic, dist, model, vol = np.array(aic),np.array(dist),np.array(model), np.array(vol)
garchSearch = pd.DataFrame(model, columns = ['Model'])
garchSearch['Volatility'] = vol
garchSearch['Distribution'] = dist
garchSearch['AIC'] = aic.astype(float).round(2)
x = garchSearch.sort_values(by='AIC', ascending=True)
modelTable = pd.concat([x.head(5).round(2),pd.DataFrame({'Model': ["..."],
                    'Volatility': ["..."],
                    'Distribution': ["..."],
             "AIC": ["..."]}), x.tail(5).round(2)])
fig, ax = plt.subplots()
# hide axes
fig.patch.set_visible(False)
ax.axis('off')
ax.axis('tight')
ax.table(cellText=modelTable.values, colLabels=modelTable.columns, loc='center')
fig.tight_layout()
plt.close()
#plt.savefig(f'figures\modelFit\stock{stockList[z]}.png')
```

Model	Volatility	Distribution	AIC
GARCH (1, 3)	EGARCH	skewt	3682.5
GARCH (1, 1)	EGARCH	skewt	3682.96
GARCH (1, 1)	EGARCH	t	3683.19
GARCH (1, 3)	EGARCH	t	3683.26
GARCH (3, 2)	GARCH	skewt	3683.46
...	...	...	...
GARCH (2, 3)	GARCH	normal	3786.54
GARCH (1, 1)	GARCH	normal	3786.69
GARCH (3, 3)	GARCH	normal	3788.47
GARCH (1, 2)	GARCH	normal	3788.69
GARCH (1, 3)	GARCH	normal	3790.69

Figure 5: png

Model	Volatility	Distribution	AIC
GARCH (1, 2)	EGARCH	t	3486.15
GARCH (2, 1)	EGARCH	t	3486.91
GARCH (2, 2)	EGARCH	t	3487.72
GARCH (3, 1)	EGARCH	t	3487.81
GARCH (1, 2)	EGARCH	skewt	3488.09
...	...	...	...
GARCH (1, 1)	GARCH	normal	3630.19
GARCH (2, 3)	GARCH	normal	3630.2
GARCH (1, 3)	GARCH	normal	3630.21
GARCH (3, 3)	GARCH	normal	3630.77
GARCH (1, 2)	GARCH	normal	3632.19

Figure 6: png

Model	Volatility	Distribution	AIC
GARCH (1, 3)	EGARCH	t	4386.07
GARCH (1, 3)	EGARCH	skewt	4387.87
GARCH (2, 3)	EGARCH	t	4388.07
GARCH (1, 1)	GARCH	t	4388.52
GARCH (1, 2)	EGARCH	t	4388.68
...	...	...	...
GARCH (1, 2)	GARCH	normal	4570.86
GARCH (2, 3)	GARCH	normal	4571.31
GARCH (1, 3)	GARCH	normal	4572.86
GARCH (2, 1)	EGARCH	normal	4573.79
GARCH (1, 1)	EGARCH	normal	4589.65

Figure 7: png

Model	Volatility	Distribution	AIC
GARCH (1, 2)	EGARCH	t	3589.73
GARCH (3, 1)	EGARCH	t	3590.45
GARCH (2, 2)	EGARCH	t	3590.93
GARCH (1, 2)	EGARCH	skewt	3591.09
GARCH (3, 1)	EGARCH	skewt	3591.13
...	...	...	...
GARCH (3, 2)	GARCH	normal	3756.1
GARCH (3, 3)	GARCH	normal	3757.84
GARCH (1, 1)	GARCH	normal	3762.53
GARCH (1, 2)	GARCH	normal	3764.53
GARCH (1, 3)	GARCH	normal	3766.53

Figure 8: png

Fitting two models each manually to compare parameters

```
#Implementing two models for Lloyds
optimModel = arch_model(prices[0]['logReturn'],
                        p = 1,
                        q = 1,
                        mean = 'constant',
                        vol = 'EGARCH',
                        dist = 'skewt')
optimRes = optimModel.fit(disp = False)
#Implementing the found model above
optimModel2 = arch_model(prices[0]['logReturn'],
                        p = 1,
                        q = 3,
                        mean = 'constant',
                        vol = 'EGARCH',
                        dist = 'skewt')
optimRes2 = optimModel2.fit(disp = False)
nam = np.array(['mu','omega', 'alpha[1]','beta[1]','eta','lambda'])
c1, p1, c2, p2 = np.array(optimRes.params),np.array(optimRes.pvalues),np.array(optimRes2.params), np.ar
garchParam = pd.DataFrame(nam, columns = ['Coefficient'])
garchParam ['Estimate'] = c1
garchParam['P_Value'] = p1
nam2 = np.array(['mu','omega', 'alpha[1]','beta[1]','beta[2]','beta[3]','eta','lambda'])
garchParam2 = pd.DataFrame(nam2, columns = ['Coefficient'])
garchParam2 ['Estimate'] = c2
garchParam2['P_Value'] = p2
col_names = ["Coefficient", "Estimate", "P-Value"]
```

```
def color_negative_red(value):
  """
  Colors elements in a dateframe
  green if positive and red if
  negative. Does not color NaN
  values.
  """

  if value < 0.05:
    color = 'green'
  else:
    color = 'black'

  return 'color: %s' % color
cm = sns.light_palette("green", as_cmap=True)
# Set CSS properties for th elements in dataframe
th_props = [
  ('font-size', '20px'),
  ('text-align', 'center'),
  ('font-weight', 'bold'),
  ('color', '#6d6d6d'),
```

```
  ('background-color', '#f7f7f9')
  ]

# Set CSS properties for td elements in dataframe
td_props = [
  ('font-size', '20px')
  ]

# Set table styles
styles = [
  dict(selector="th", props=th_props),
  dict(selector="td", props=td_props),
  dict(selector="caption",
           props=[("text-align", "center"),
                  ("font-size", "125%"),
                  ("color", 'black'),
                 ('font-weight', 'bold')])
  ]
```

```
df1_styler = garchParam.style.applymap(color_negative_red, subset=['P_Value']).set_caption(f'Lloyds Banl
df2_styler = garchParam2.style.applymap(color_negative_red, subset=['P_Value']).set_caption(f'Lloyds Ba

#display_html(df1_styler._repr_html_()+df2_styler._repr_html_(), raw=True)
```

```
#Implementing two models for Tesco
optimModel = arch_model(prices[1]['logReturn'],
                        p = 1,
                        q = 2,
                        mean = 'constant',
                        vol = 'EGARCH',
                        dist = 't')
optimRes = optimModel.fit(disp = False)
optimModel2 = arch_model(prices[1]['logReturn'],
                        p = 2,
                        q = 1,
                        mean = 'constant',
                        vol = 'EGARCH',
                        dist = 't')
optimRes2 = optimModel2.fit(disp = False)
nam = np.array(['mu','omega', 'alpha[1]','beta[1]','beta[2]','nu'])
c1, p1, c2, p2 = np.array(optimRes.params),np.array(optimRes.pvalues),np.array(optimRes2.params), np.ar
garchParam = pd.DataFrame(nam, columns = ['Coefficient'])
garchParam ['Estimate'] = c1
garchParam['P_Value'] = p1
nam2 = np.array(['mu','omega', 'alpha[1]','alpha[2]','beta[1]','nu'])
garchParam2 = pd.DataFrame(nam2, columns = ['Coefficient'])
garchParam2 ['Estimate'] = c2
garchParam2['P_Value'] = p2
col_names = ["Coefficient", "Estimate", "P-Value"]
```

```
df1_styler = garchParam.style.applymap(color_negative_red, subset=['P_Value']).set_caption(f'Tesco: {op
df2_styler = garchParam2.style.applymap(color_negative_red, subset=['P_Value']).set_caption(f'Tesco: {o

display_html(df1_styler._repr_html_()+df2_styler._repr_html_(), raw=True)
```

```
#Implementing two models for Tesco
optimModel = arch_model(prices[2]['logReturn'],
                        p = 1,
                        q = 3,
                        mean = 'constant',
                        vol = 'EGARCH',
                        dist = 't')
optimRes = optimModel.fit(disp = False)
optimModel2 = arch_model(prices[2]['logReturn'],
                        p = 1,
                        q = 3,
                        mean = 'constant',
                        vol = 'EGARCH',
                        dist = 'skewt')
optimRes2 = optimModel2.fit(disp = False)
nam = np.array(['mu','omega', 'alpha[1]','beta[1]','beta[2]','beta[3]','nu'])
c1, p1, c2, p2 = np.array(optimRes.params),np.array(optimRes.pvalues),np.array(optimRes2.params), np.ar
garchParam = pd.DataFrame(nam, columns = ['Coefficient'])
garchParam ['Estimate'] = c1
garchParam['P_Value'] = p1
nam2 = np.array(['mu','omega', 'alpha[1]','beta[1]','beta[2]','beta[3]','eta','lambda'])
garchParam2 = pd.DataFrame(nam2, columns = ['Coefficient'])
garchParam2 ['Estimate'] = c2
garchParam2['P_Value'] = p2
col_names = ["Coefficient", "Estimate", "P-Value"]
```

```
df1_styler = garchParam.style.applymap(color_negative_red, subset=['P_Value']).set_caption(f'Rolls Royc
df2_styler = garchParam2.style.applymap(color_negative_red, subset=['P_Value']).set_caption(f'Rolls Roy

#display_html(df1_styler._repr_html_()+df2_styler._repr_html_(), raw=True)
```

```
#Implementing two models for Tesco
optimModel = arch_model(prices[3]['logReturn'],
                        p = 3,
                        q = 1,
                        mean = 'constant',
```

```
                        vol = 'EGARCH',
                        dist = 't')
optimRes = optimModel.fit(disp = False)
optimModel2 = arch_model(prices[3]['logReturn'],
                        p = 1,
                        q = 2,
                        mean = 'constant',
                        vol = 'EGARCH',
                        dist = 't')
optimRes2 = optimModel2.fit(disp = False)
nam = np.array(['mu','omega', 'alpha[1]','alpha[2]','alpha[3]','beta[1]','nu'])
c1, p1, c2, p2 = np.array(optimRes.params),np.array(optimRes.pvalues),np.array(optimRes2.params), np.ar
garchParam = pd.DataFrame(nam, columns = ['Coefficient'])
garchParam ['Estimate'] = c1
garchParam['P_Value'] = p1
nam2 = np.array(['mu','omega', 'alpha[1]','beta[1]','beta[2]','nu'])
garchParam2 = pd.DataFrame(nam2, columns = ['Coefficient'])
garchParam2 ['Estimate'] = c2
garchParam2['P_Value'] = p2
col_names = ["Coefficient", "Estimate", "P-Value"]
```

```
df1_styler = garchParam.style.applymap(color_negative_red, subset=['P_Value']).set_caption(f'Vodafone:
df2_styler = garchParam2.style.applymap(color_negative_red, subset=['P_Value']).set_caption(f'Vodafone:

#display_html(df1_styler._repr_html_()+df2_styler._repr_html_(), raw=True)
```

```
bestModels = [arch_model(prices[0]['logReturn'],
                        p = 1,
                        q = 1,
                        mean = 'constant',
                        vol = 'EGARCH',
                        dist = 'skewt').fit(disp = False)
              ,arch_model(prices[1]['logReturn'],
                        p = 1,
                        q = 2,
                        mean = 'constant',
                        vol = 'EGARCH',
                        dist = 't').fit(disp = False),
              arch_model(prices[2]['logReturn'],
                        p = 1,
                        q = 3,
                        mean = 'constant',
                        vol = 'EGARCH',
                        dist = 't').fit(disp = False)
              ,arch_model(prices[3]['logReturn'],
                        p = 1,
                        q = 2,
                        mean = 'constant',
```

```
                        vol = 'EGARCH',
                        dist = 't').fit(disp=False)]
```

```
fileNames = ['Lloyds', 'Tesco', 'RollsRoyce', 'Vodafone']
```

```
for i in range(len(bestModels)):
    stdRes = bestModels[i].resid/bestModels[i].conditional_volatility
    plt.figure(figsize=(20,10),facecolor=(1, 1, 1))
    sns.distplot(stdRes,fit_kws = {'color':'blue'},
                 fit = stats.laplace,
                 bins = 50,
                 hist_kws={"histtype": "bar",
                           "linewidth": 3,
                           "alpha": 1,
                           "color": "g"},
                 kde = False)
    sns.distplot(stdRes, fit_kws = {'color':'red'},
                 fit = stats.norm, hist = False, kde = False)
    sns.distplot(stdRes,fit_kws = {'color':'yellow'},
                 fit = stats.t, hist = False, kde = False)
    plt.legend(('Laplace', "Normal", "T",'standardized residuals'))
    plt.title(f'Standardized Residuals Density Plot for {stockList[i]} {bestModels[i].model.volatility}
    #plt.savefig('figures\plot {0}.png'.format(i+1))
    plt.close()
```

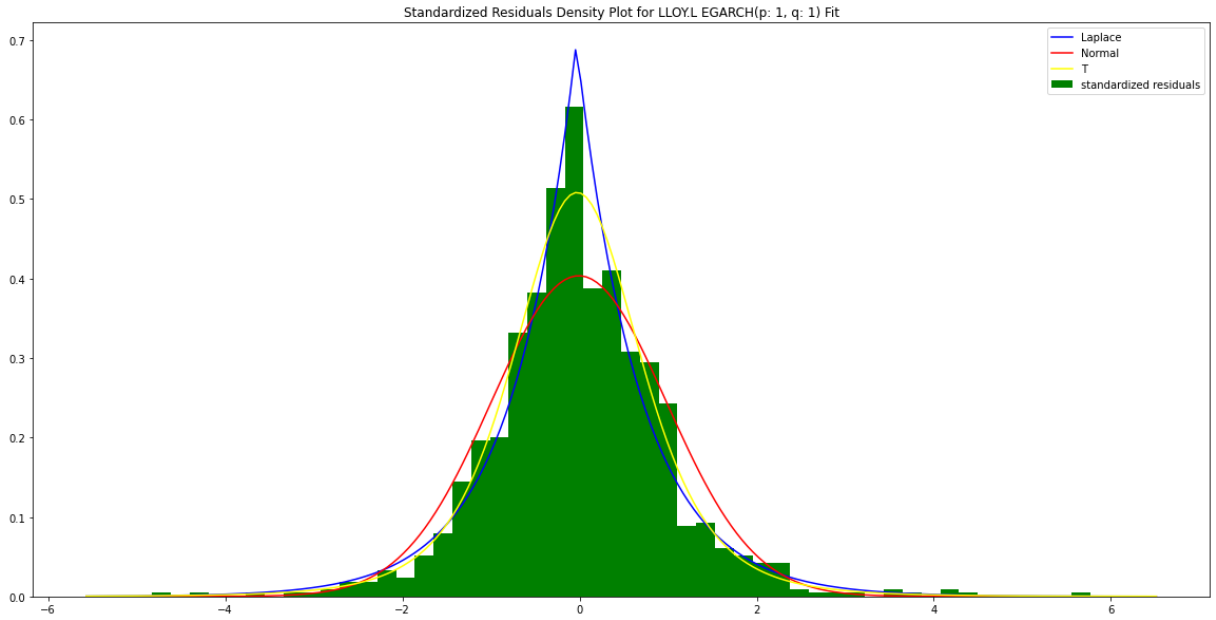


Figure 9: png

Lloyds, Tesco, Rolls Royce, Vodafone

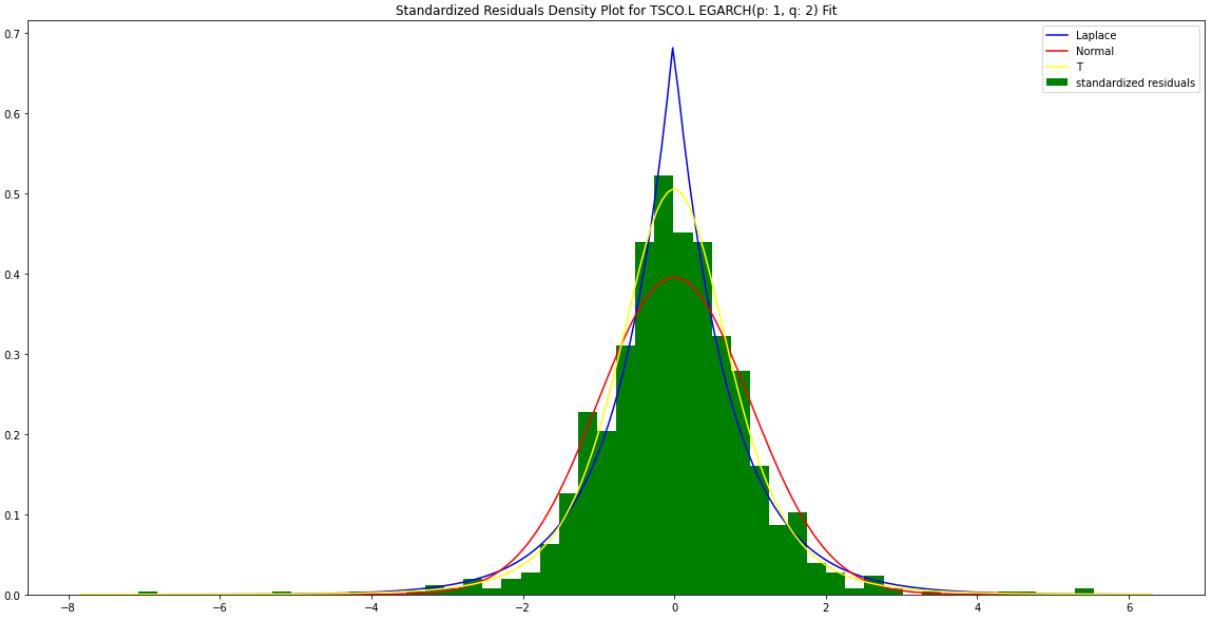


Figure 10: png

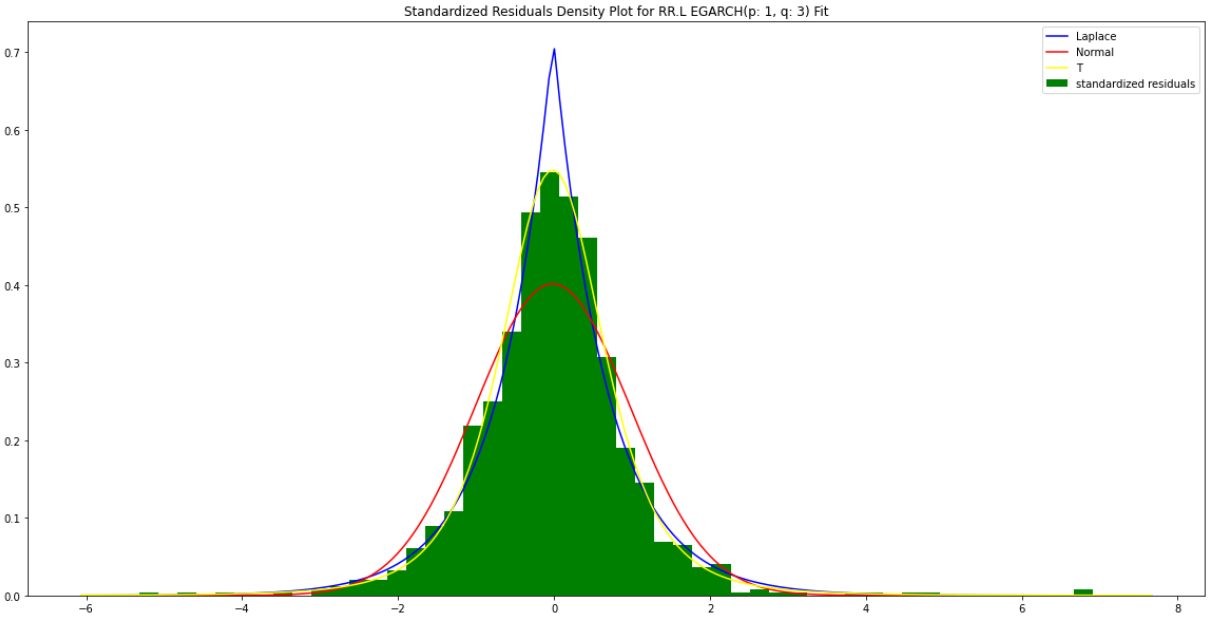


Figure 11: png

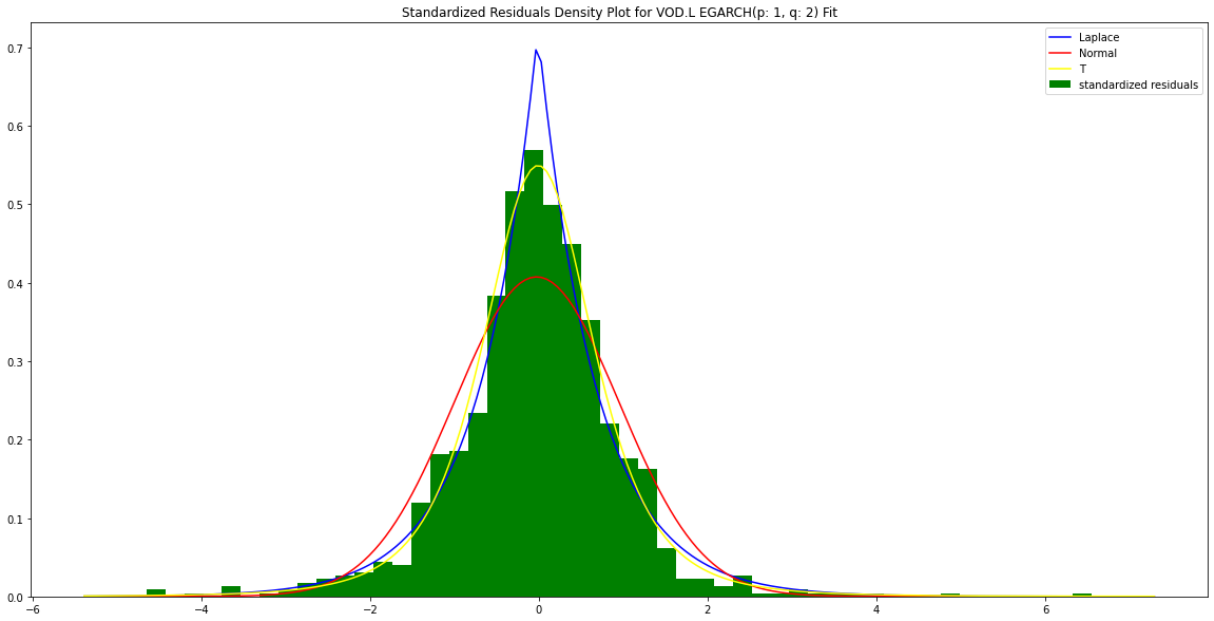


Figure 12: png

Backtesting and Putting in Table

```
mae = [0 for i in range(4)]
mse = [0 for i in range(4)]
for i in range(len(stockList)):
    mae[i] = mean_absolute_error(prices[i]['logReturn'].sub(prices[i]['logReturn'].mean()).pow(2),
                                 bestModels[i].conditional_volatility)
    mse[i] = mean_squared_error(prices[i]['logReturn'].sub(prices[i]['logReturn'].mean()).pow(2),
                                bestModels[i].conditional_volatility)

fileNames, mae, mse = np.array(fileNames), np.array(mae), np.array(mse)
```

```
backTest = pd.DataFrame(fileNames, columns = ['Company'])
backTest['MSE'] = mse
backTest['MAE'] = mae
```

```
for i in range(len(stockList)):
    plt.figure(figsize=(20,10),facecolor=(1, 1, 1))
    plt.plot(bestModels[i].conditional_volatility,
             color = 'red',
             label = 'GARCH Volatility', linewidth = 4)
    plt.plot(prices[i]['logReturn'], color = 'grey',
             label = 'Log Returns', alpha = 0.4, linewidth =4)
    plt.title(f'{fileNames[i]} EGARCH Volatility over Log Returns',
             fontsize = 25)
    plt.legend(loc = 'upper right')
    #plt.savefig('figures\plot {0}.png'.format(i+1))
    plt.close()  # if you do not need to leave the figures open
```

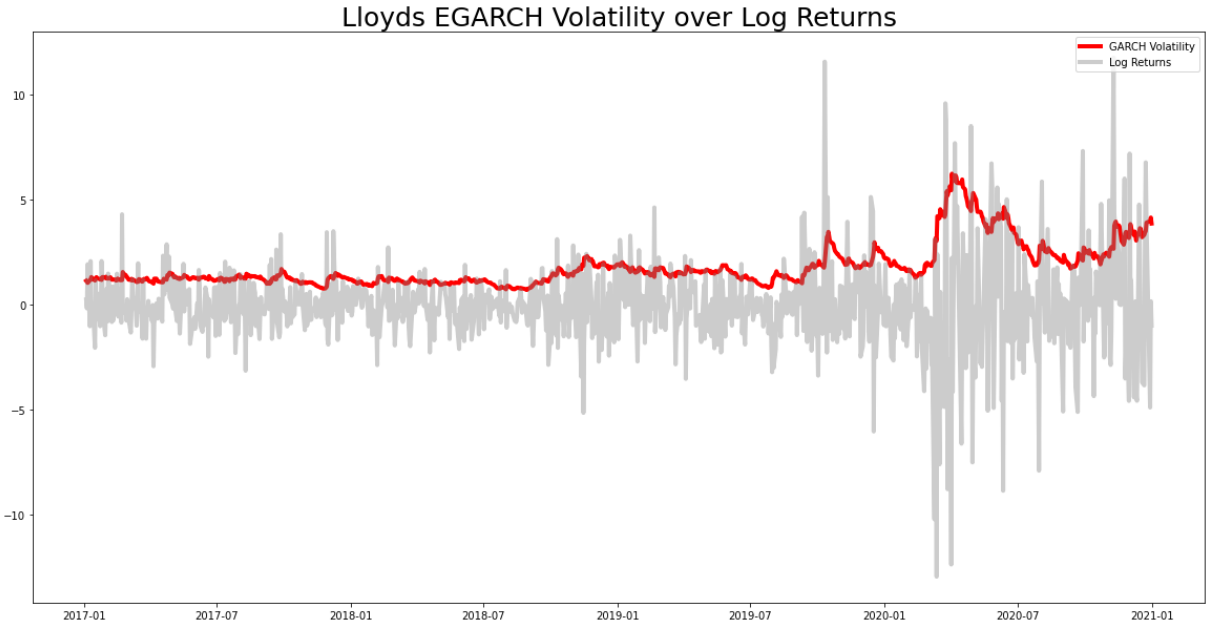


Figure 13: png

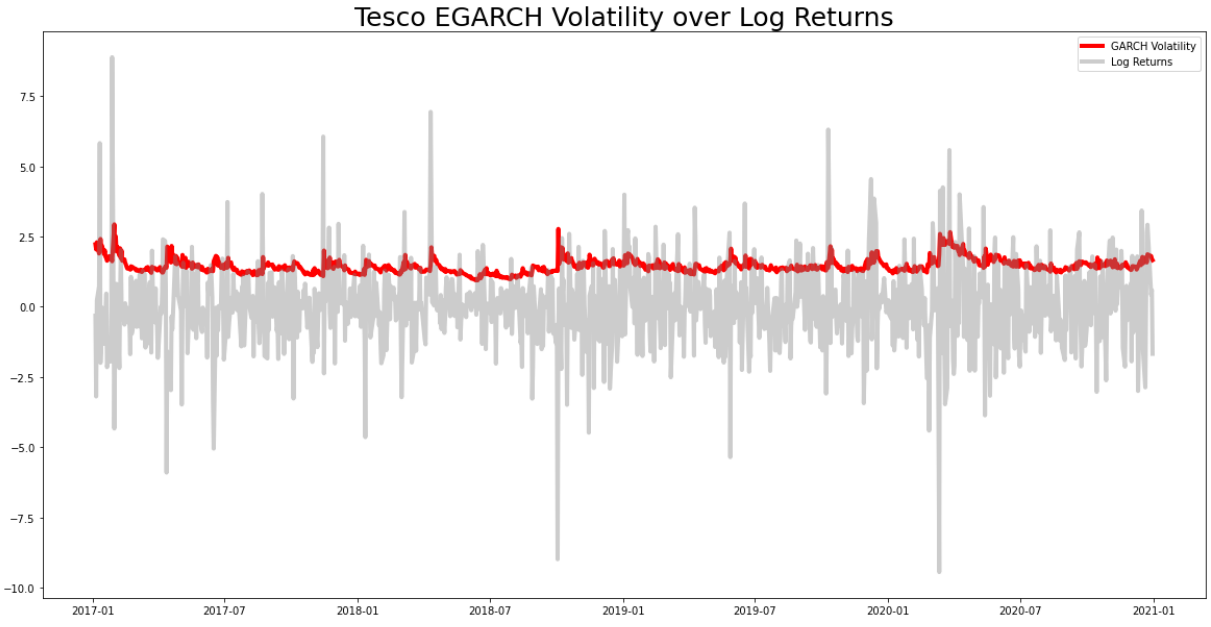


Figure 14: png

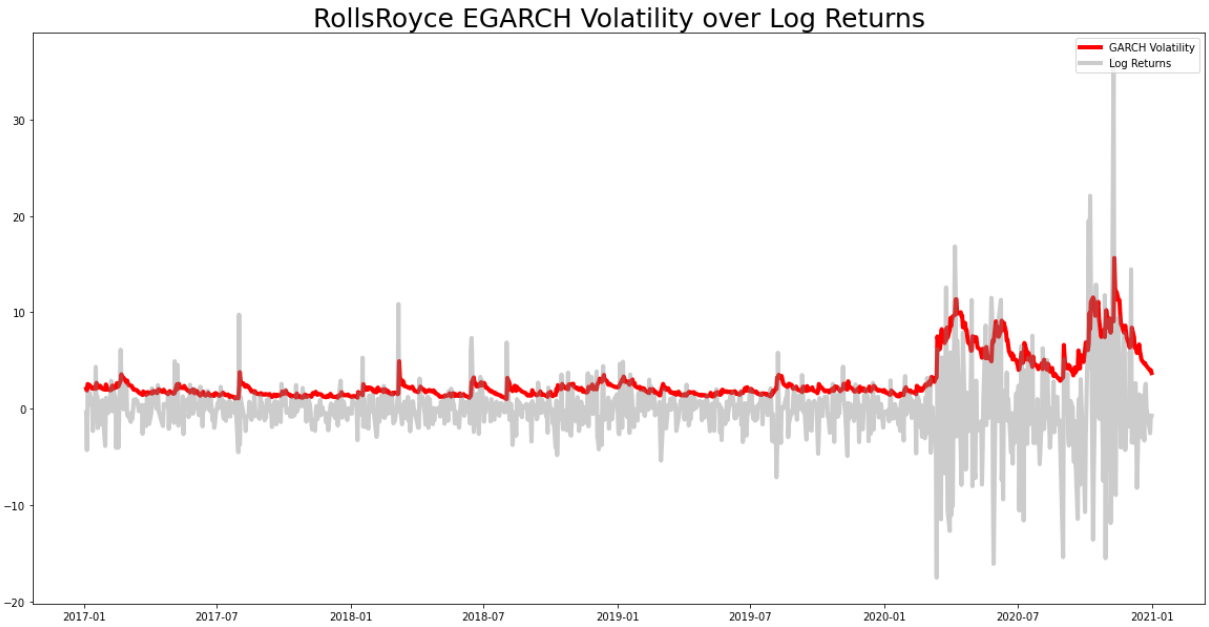


Figure 15: png

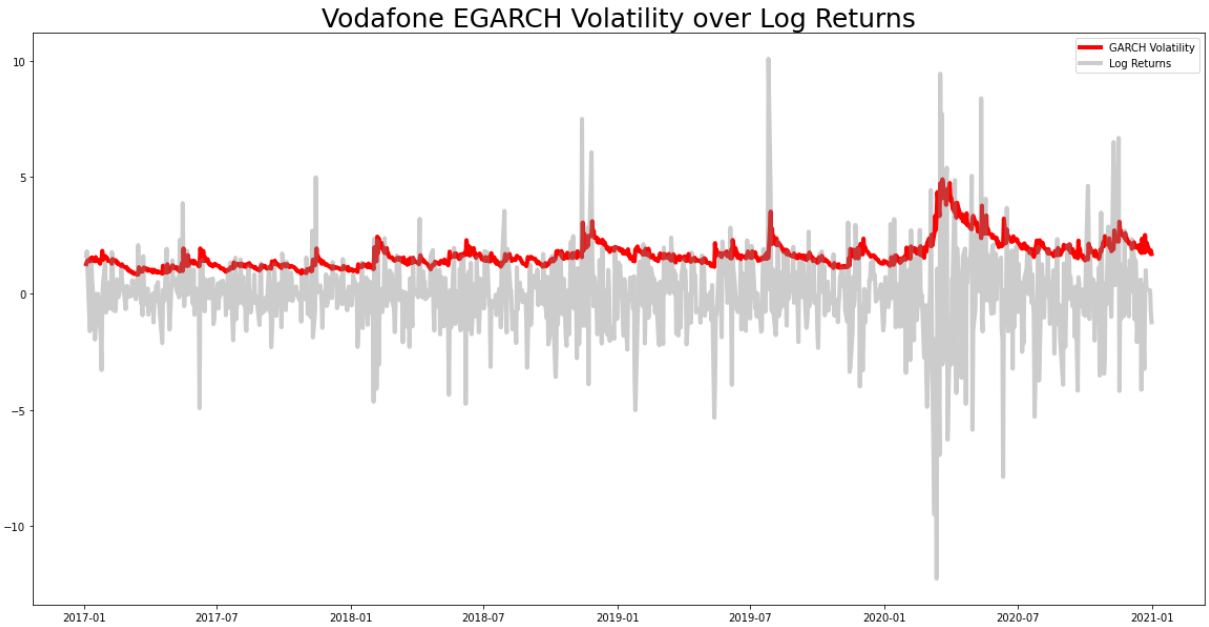


Figure 16: png

```
cm = sns.light_palette("green", as_cmap=True)
#(backTest.style

#    .highlight_min(subset=['MSE','MAE']).set_table_styles(styles))
```

Comparing Volatility Proxy

```
# Plot the actual volatility
for i in range(len(stockList)):
    plt.figure(figsize=(20,10),facecolor=(1, 1, 1))
    plt.plot(prices[i]['logReturn'].sub(prices[i]['logReturn'].mean()).pow(2),
             color = 'grey', alpha = 0.4, label = 'Daily Volatility',
            linewidth = 4)
    # Plot EGARCH  estimated volatility
    plt.plot(bestModels[i].conditional_volatility**2, color = 'red', label = 'EGARCH Volatility', linew
    plt.legend(loc = 'upper right')
    plt.title(f'{fileNames[i]} EGARCH Volatility over Squared & Centered Returns',
             fontsize = 25)
    plt.xticks(fontsize =15)
    plt.yticks(fontsize =15)
    plt.ylabel('Return (%)',fontsize =15)
    plt.xlabel('Date',fontsize =15)
    #plt.savefig('figures\plot {0}.png'.format(i+1))
    plt.close()  # if you do not need to leave the figures open
```

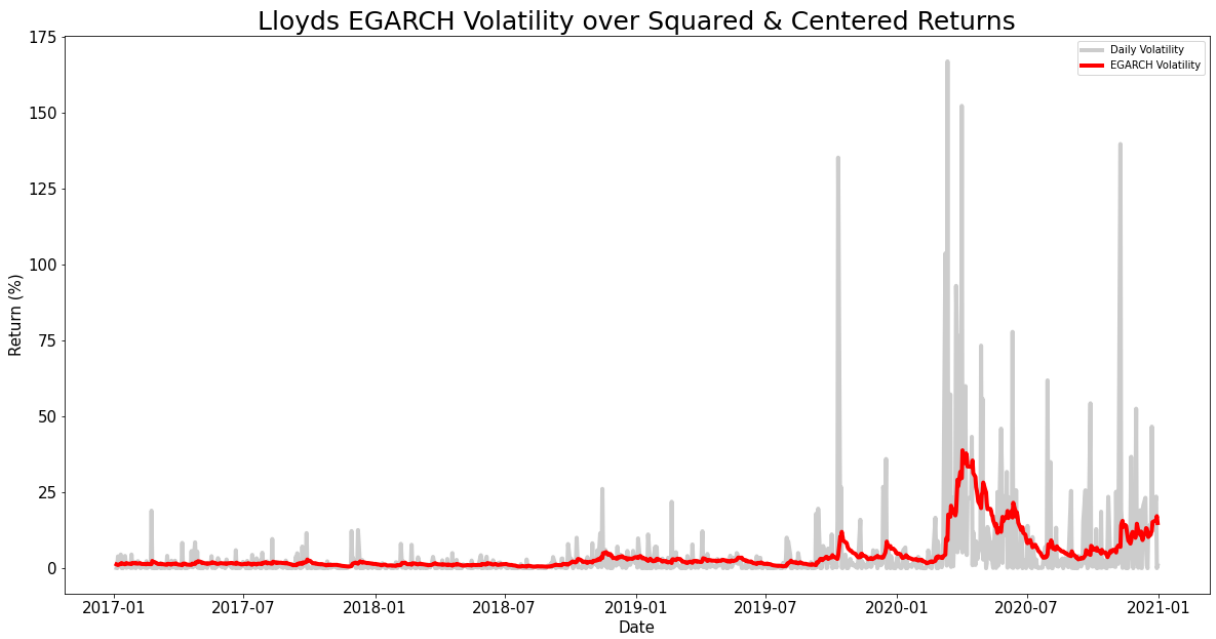


Figure 17: png

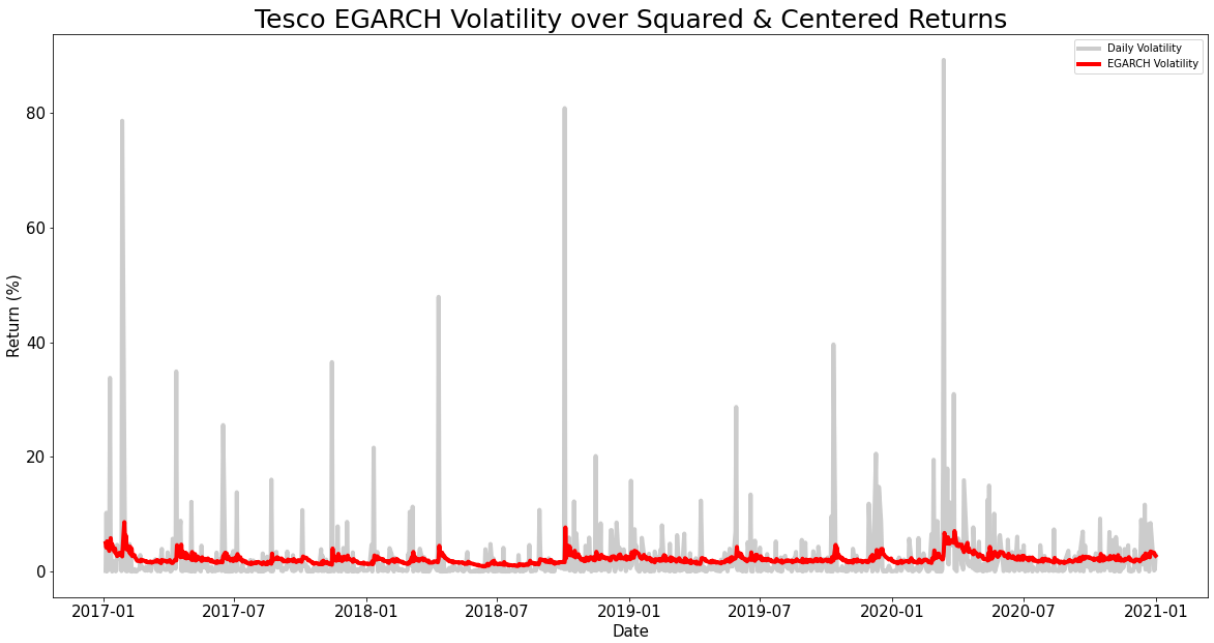


Figure 18: png

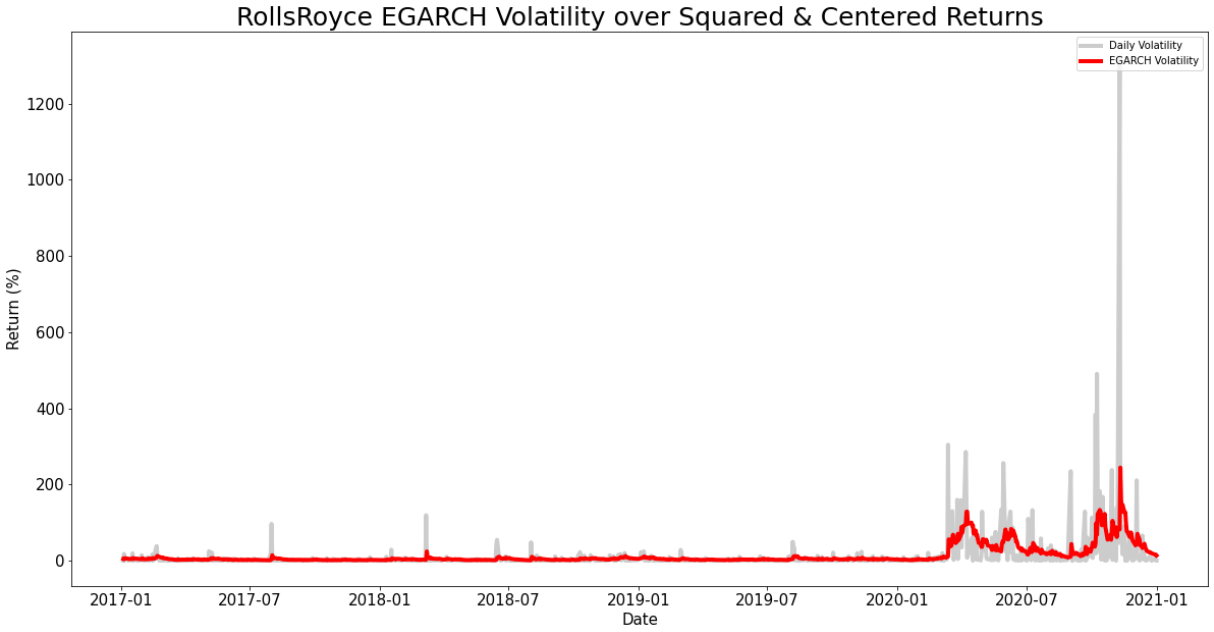


Figure 19: png

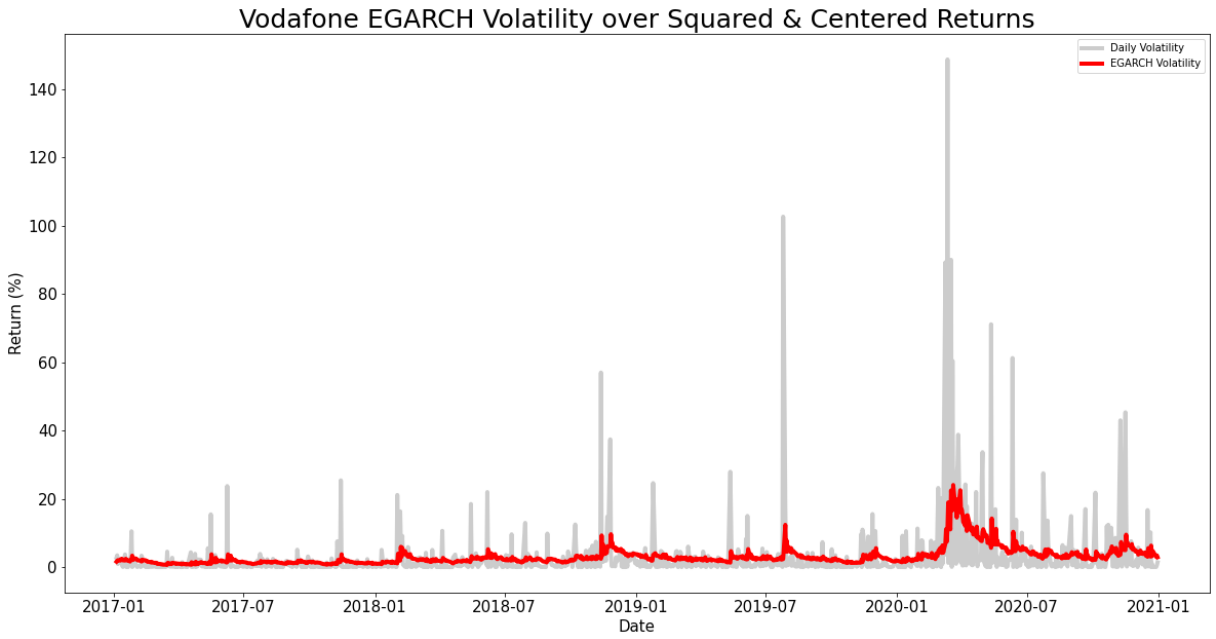


Figure 20: png

GARCH FORECASTING

```
index = prices[0].index
start_loc = 0
end_loc = np.where(index >= '2020-1-1')[0].min()
forecasts = {}
windowLength = 252
```

Cleaning Models

```
forcModels = [arch_model(prices[0]['logReturn'],
                         p = 1,
                         q = 1,
                         mean = 'constant',
                         vol = 'EGARCH',
                         dist = 'skewt')
              ,arch_model(prices[1]['logReturn'],
                         p = 1,
                         q = 2,
                         mean = 'constant',
                         vol = 'EGARCH',
                         dist = 't'),
              arch_model(prices[2]['logReturn'],
                         p = 1,
                         q = 3,
                         mean = 'constant',
```

```
                        vol = 'EGARCH',
                        dist = 't')
                ,arch_model(prices[3]['logReturn'],
                        p = 1,
                        q = 0,
                        mean = 'constant',
                        vol = 'EGARCH',
                        dist = 't')]
```

Fixed Window Variance

```
variance_fixedwin = [0 for i in range(4)]
for j in range(4):
    warnings.filterwarnings('ignore')
    warnings.simplefilter('ignore')
    for i in range(windowLength):
        sys.stdout.write('-')
        sys.stdout.flush()
        res = forcModels[j].fit(first_obs=start_loc + i, last_obs=i + end_loc, disp='off')
        temp = res.forecast(horizon=1).variance
        fcast = temp.iloc[i + end_loc - 1]
        forecasts[fcast.name] = fcast
    print(' Done!')
    variance_fixedwin[j] = pd.DataFrame(forecasts).T
```

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

```
variance_expandwin = [0 for i in range(4)]
for j in range(4):
    for i in range(windowLength):
        warnings.filterwarnings('ignore')
        warnings.simplefilter('ignore')
        sys.stdout.write('-')
        sys.stdout.flush()
        res = forcModels[j].fit(first_obs=start_loc, last_obs=i + end_loc, disp='off')
        temp = res.forecast(horizon=1).variance
        fcast = temp.iloc[i + end_loc - 1]
        forecasts[fcast.name] = fcast
    print(' Done!')
    variance_expandwin[j] = pd.DataFrame(forecasts).T
```

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

```
for i in range(len(stockList)):
    # Calculate volatility from variance forecast with an expanding window
    vol_expandwin = np.sqrt(variance_expandwin[i])

    # Calculate volatility from variance forecast with a fixed rolling window
    vol_fixedwin = np.sqrt(variance_fixedwin[i])

    # Plot results
    plt.figure(figsize=(20,10),facecolor=(1, 1, 1))

    plt.plot(abs(prices[i].logReturn.loc[variance_expandwin[i].index]),
             color = 'black',
             label='Absolute Daily Return', linewidth = 4)
    # Plot volatility forecast with an expanding window
    plt.plot(vol_expandwin, color = 'blue', label='Expanding Window', linewidth = 4)

    # Plot volatility forecast with a fixed rolling window
    plt.plot(vol_fixedwin, color = 'red', label='Rolling Window',linewidth = 4)
    plt.title(f'{fileNames[i]} Volatility Forecasted Volatility over Absolute Returns',
             fontsize = 25)

    plt.xticks(fontsize =15)
    plt.yticks(fontsize =15)
    plt.ylabel('Return (%)',fontsize =15)
    plt.xlabel('Date',fontsize =15)
    plt.legend(loc = 'best',
               fontsize = 20)
    plt.show()
    #plt.savefig('figures\plot {0}.png'.format(i+1))
    #plt.close()  # if you do not need to leave the figures open
```

Backtesting the forecasts

```
mae = [0 for i in range(4)]
mse = [0 for i in range(4)]

for i in range(len(stockList)):
    #Indexing the test data
    testData = prices[i].logReturn.loc[variance_expandwin[i].tail(25).index]
    sqctest = testData.sub(testData.mean()).pow(2)
    #calculating mae/mse
    mae[i] = mean_absolute_error(sqctest,
                                 variance_expandwin[i].tail(25))
    mse[i] = mean_squared_error(sqctest,
                                 variance_expandwin[i].tail(25))

fileNames, mae, mse = np.array(fileNames), np.array(mae), np.array(mse)

backTest = pd.DataFrame(fileNames, columns = ['Company'])
backTest['MSE'] = mse
backTest['MAE'] = mae
```

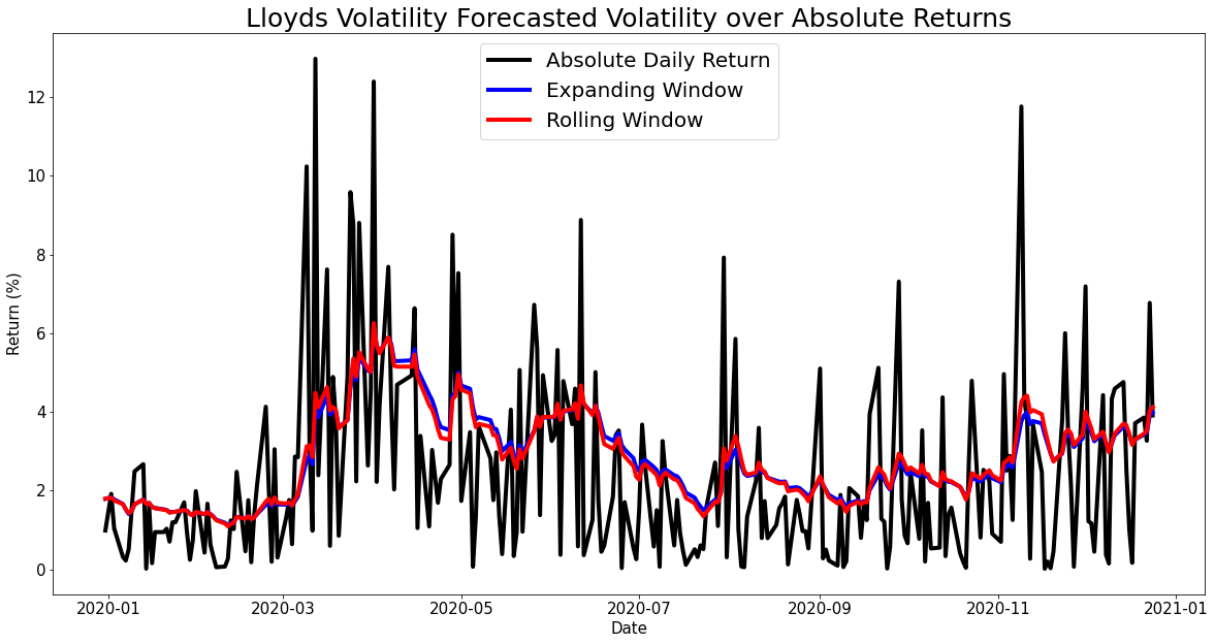


Figure 21: png

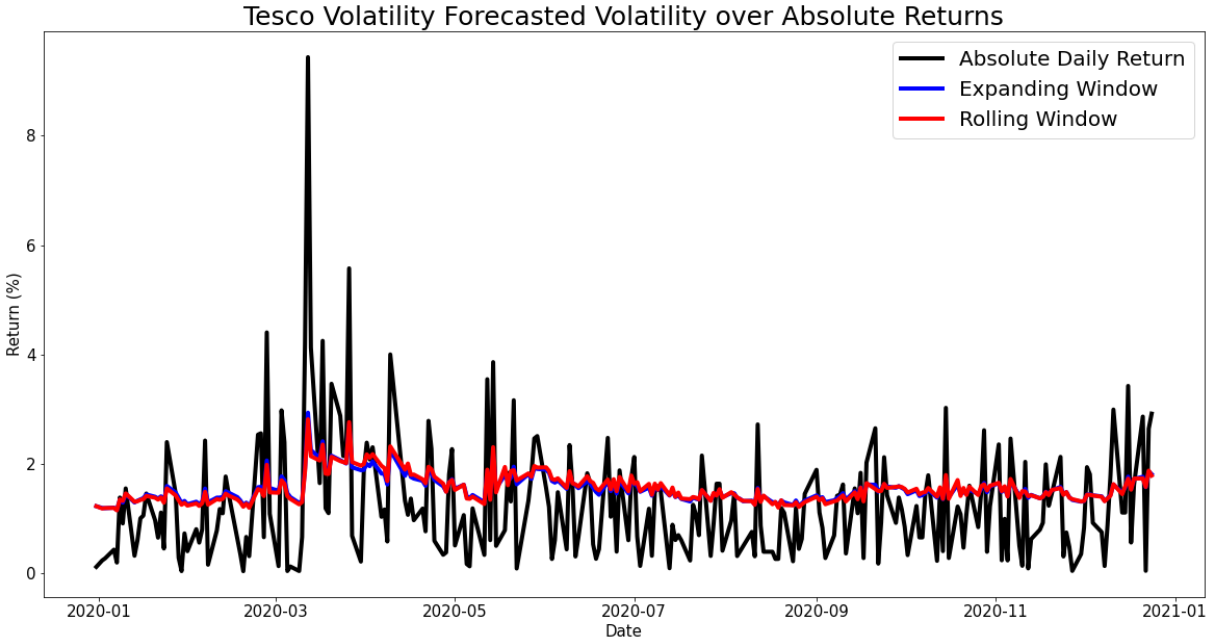


Figure 22: png

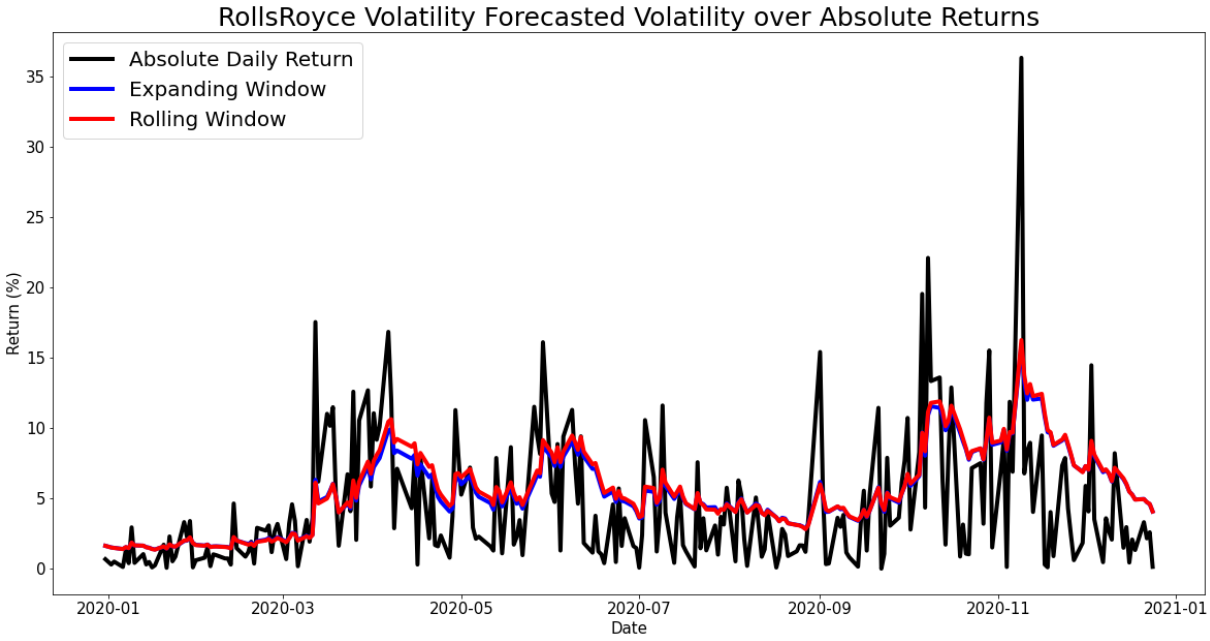


Figure 23: png

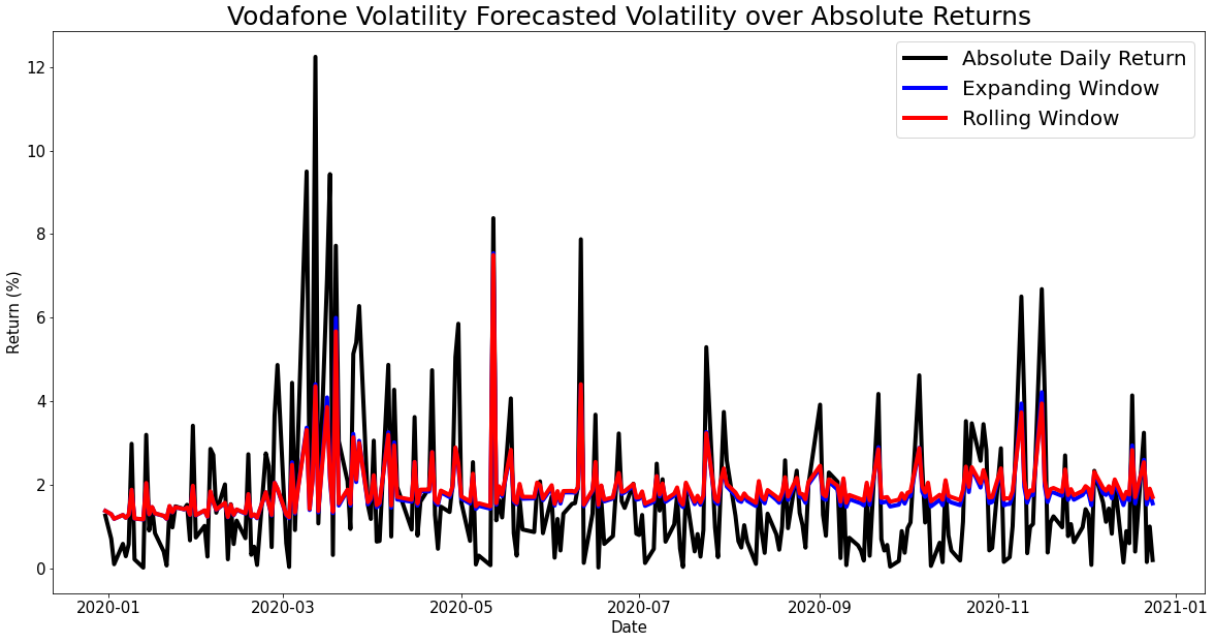


Figure 24: png

```
cm = sns.light_palette("green", as_cmap=True)
(backTest.style
   .highlight_min(subset=['MSE','MAE']).set_table_styles(styles))
```

```
#variance_expandwin[i].tail(25).index
```

```
DatetimeIndex(['2020-04-17', '2020-04-20', '2020-04-21', '2020-04-22',
               '2020-04-23', '2020-04-24', '2020-04-27', '2020-04-28',
               '2020-04-29', '2020-04-30', '2020-05-01', '2020-05-04',
               '2020-05-05', '2020-05-06', '2020-05-07', '2020-05-11',
               '2020-05-12', '2020-05-13', '2020-05-14', '2020-05-15',
               '2020-05-18', '2020-05-19', '2020-05-20', '2020-05-21',
               '2020-05-22'],
              dtype='datetime64[ns]', freq=None)
```

```
mae = [0 for i in range(4)]
mse = [0 for i in range(4)]

for i in range(len(stockList)):
    #Indexing the test data
    testData = prices[i].logReturn.loc[variance_fixedwin[i].tail(25).index]
    sqctest = testData.sub(testData.mean()).pow(2)
    #calculating mae/mse
    mae[i] = mean_absolute_error(sqctest,
                                 variance_fixedwin[i].tail(25))
    mse[i] = mean_squared_error(sqctest,
                                variance_fixedwin[i].tail(25))

fileNames, mae, mse = np.array(fileNames), np.array(mae), np.array(mse)

backTest = pd.DataFrame(fileNames, columns = ['Company'])
backTest['MSE'] = mse
backTest['MAE'] = mae
cm = sns.light_palette("green", as_cmap=True)
(backTest.style
   .highlight_min(subset=['MSE','MAE']).set_table_styles(styles))
```

Irithmics Portion

```
Irithmics = [0 for i in range(4)]
fileList = ['agg.csv','tesco.csv', 'rr.csv', 'vod.csv']
for i in range(4):
    Irithmics[i] = pd.read_csv(fileList[i])
    Irithmics[i] = Irithmics[i][['Date', 'shortProb']]
    Irithmics[i]['Date'] = pd.to_datetime(Irithmics[i]['Date'])
    Irithmics[i] =Irithmics[i].set_index('Date')
```

```
#Fitting GARCH(1,1) to data as the volatility is time variant
irithVol = [0 for i in range(4)]
for i in range(4):
    irithModel = arch_model(Irithmics[i]['shortProb'].mul(100),
                        p = 1,
                        q = 1,
                        mean = 'constant',
                        vol = 'EGARCH',
                        dist = 'normal')
    irithRes = irithModel.fit(disp = False)
    irithResid = irithRes.resid
    irithStd = irithRes.conditional_volatility
    irithStdResid = irithResid/irithStd
    irithVol[i] = irithStd
    plt.figure(figsize=(20,10),facecolor=(1, 1, 1))
    plt.plot(irithStd, color = 'red', label = 'GARCH Volatility', linewidth = 4)
    plt.plot(Irithmics[i]['shortProb'].mul(100), color = 'grey',
             label = 'Short Prob', alpha = 0.4, linewidth =4)
    plt.title(f"GARCH(1,1){fileNames[i]}Volatility over Probability",
             fontsize = 25)
    plt.legend(loc = 'upper right')
    #plt.savefig('figures\plot {0}.png'.format(i+1))
    plt.close()  # if you do not need to leave the figures open
```

```
# Showing a look at the aggregated data

for i in range(4):
    plt.figure(figsize=(20,10),facecolor=(1, 1, 1))
    plt.plot(Irithmics[i]['shortProb'].mul(100), color = 'black',
             label = 'Short Prob', linewidth =4)
    plt.title(f"{fileNames[i]} Forecast Aggregation",
              fontsize = 30)
    plt.legend(fontsize = 25)
    plt.xticks(fontsize = 25)
    plt.yticks(fontsize = 25)
    #plt.show()
    #plt.savefig('figures\plot {0}.png'.format(i+1))
    plt.close()
```

```
# Showing a look at the aggregated data
plt.figure(figsize=(20,10),facecolor=(1, 1, 1))
plt.plot(Irithmics[0]['shortProb'].mul(100), color = 'green',
         label = f"{fileNames[0]} Forecast Aggregation", linewidth =4)
plt.plot(Irithmics[1]['shortProb'].mul(100), color = 'blue',
         label = f"{fileNames[1]} Forecast Aggregation", linewidth =4)
plt.plot(Irithmics[2]['shortProb'].mul(100), color = 'magenta',
         label = f"{fileNames[2]} Forecast Aggregation", linewidth =4)
plt.plot(Irithmics[3]['shortProb'].mul(100), color = 'red',
         label = f"{fileNames[3]} Forecast Aggregation", linewidth =4)
plt.ylabel('Short/Sell Probability (%)', fontsize = 25)

plt.title("Forecast Aggregation",
          fontsize = 30)
```

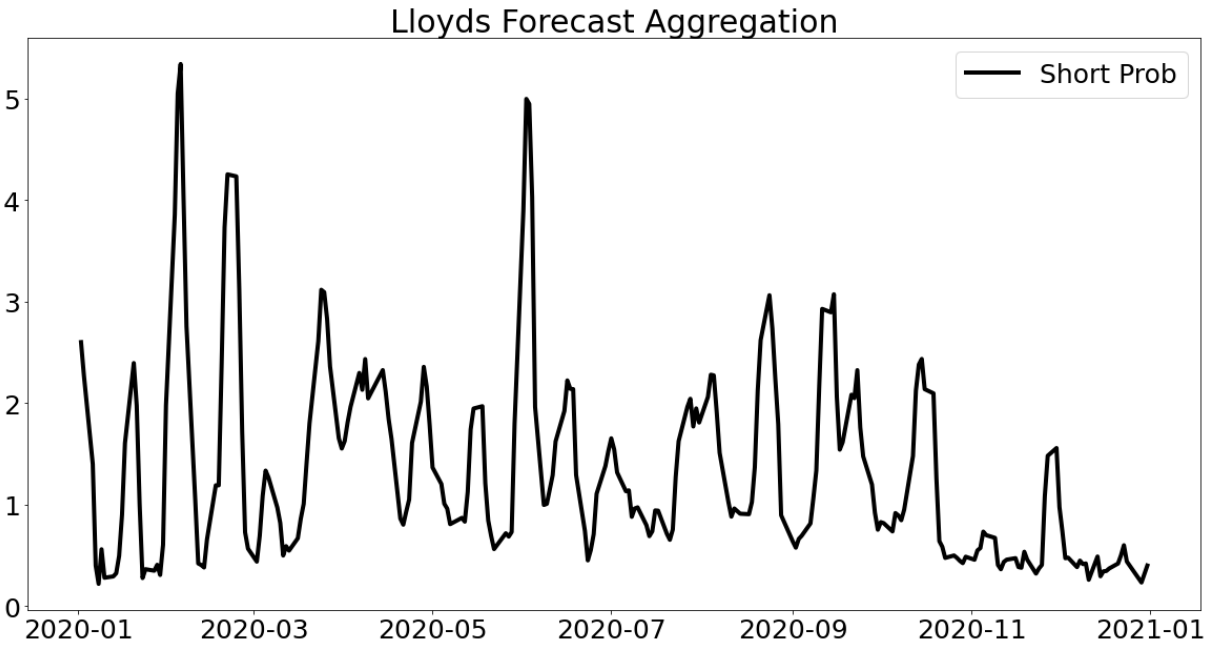


Figure 25: png

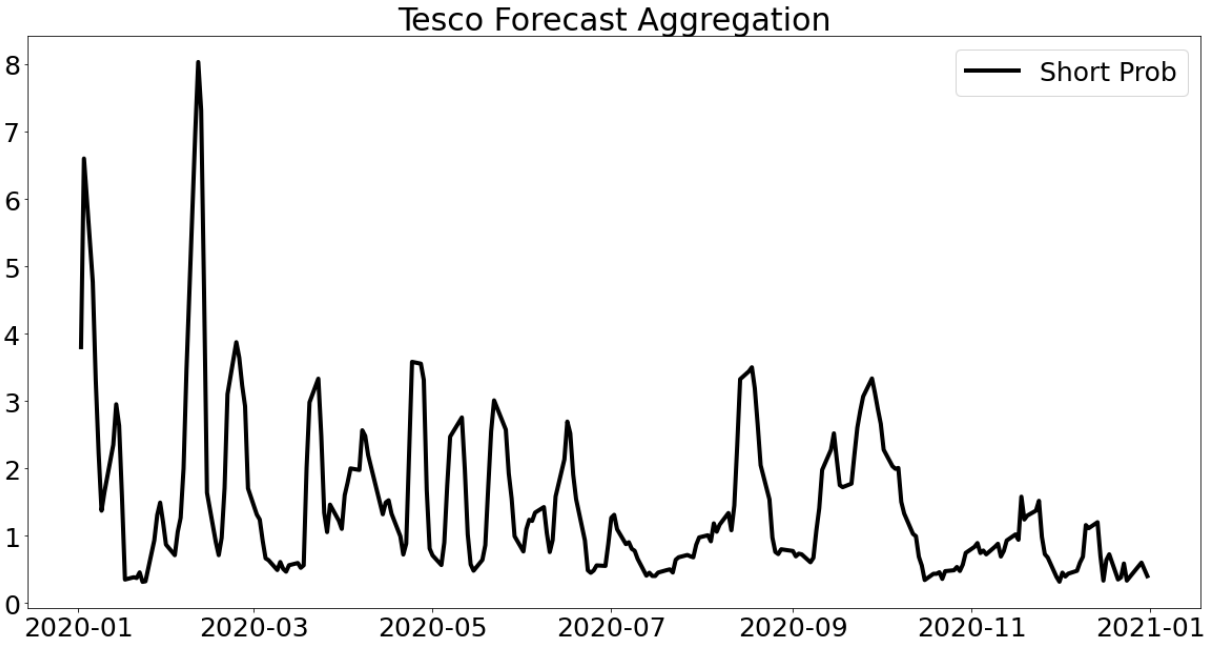


Figure 26: png

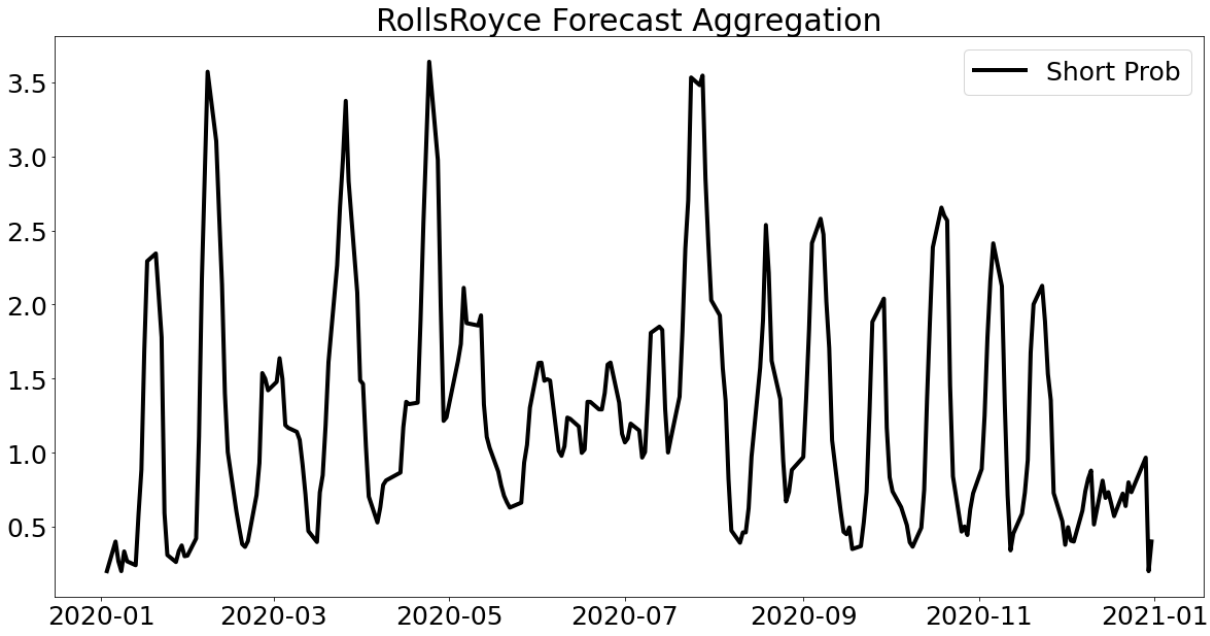


Figure 27: png

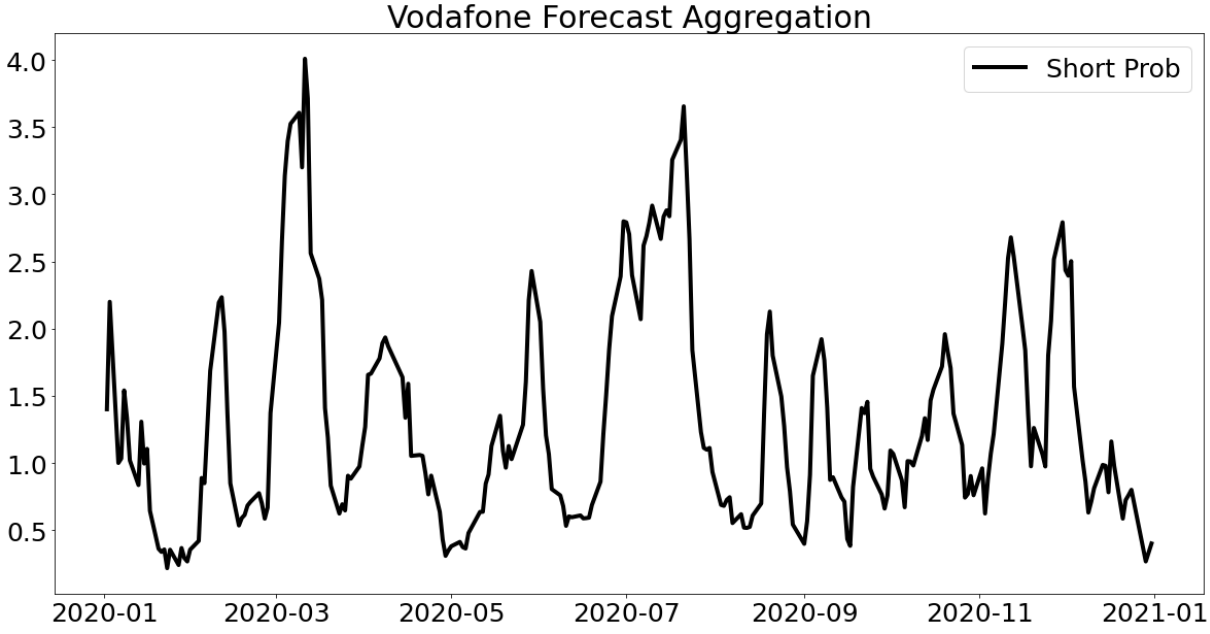


Figure 28: png

```
plt.legend(fontsize = 25)
plt.xticks(fontsize = 25)
plt.yticks(fontsize = 25)
#plt.show()
#plt.savefig('figures\plot {0}.png'.format(i+1))
plt.close()
```

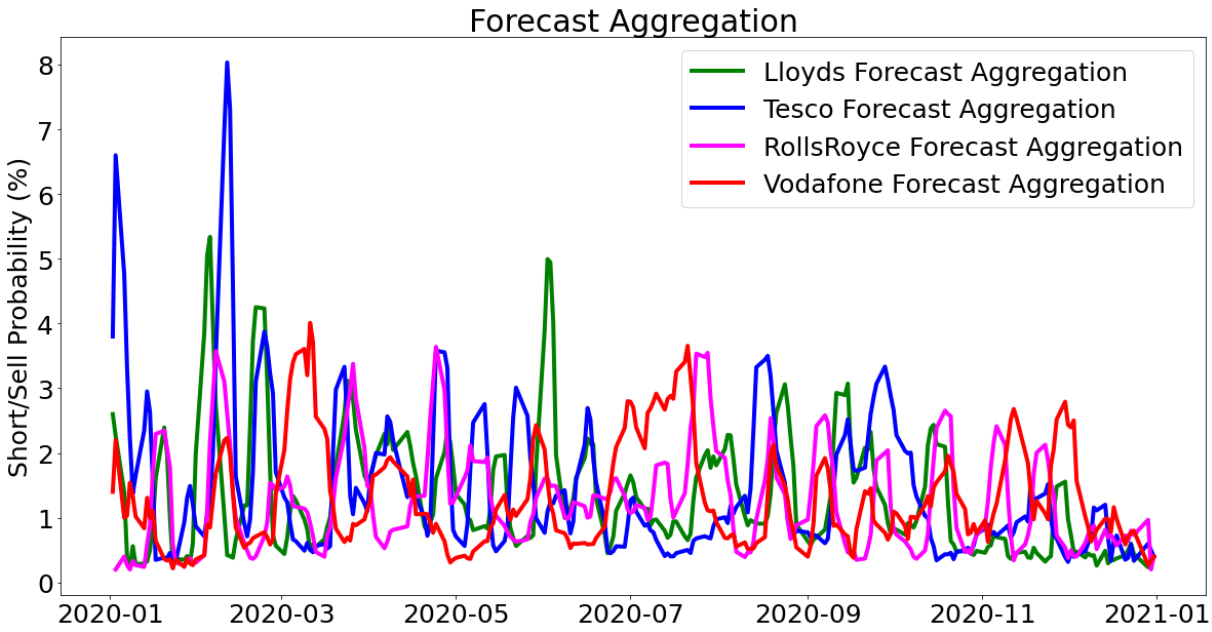


Figure 29: png

Relationship With Data?

DCC GARCH

```
import pandas as pd
import numpy as np
import statsmodels.api as sm
import plotly.express as px
import plotly.figure_factory as ff
from arch import arch_model

from ipywidgets import HBox, VBox, Dropdown, Output
from scipy.optimize import fmin, minimize
from scipy.stats import t
from scipy.stats import norm
from math import inf
from IPython.display import display

import bs4 as bs
```

```
import requests
import yfinance as yf
import datetime
```

```
def vecl(matrix):
    lower_matrix = np.tril(matrix,k=-1)
    array_with_zero = np.matrix(lower_matrix).A1

    array_without_zero = array_with_zero[array_with_zero!=0]

    return array_without_zero
```

```
def garch_t_to_u(rets, res):
    mu = res.params['mu']
    nu = res.params['nu']
    est_r = rets - mu
    h = res.conditional_volatility
    std_res = est_r / h
    # we could also just use:
    # std_res = res.std_resid
    # but it's useful to see what is going on
    udata = t.cdf(std_res, nu)
    return udata
```

```
def loglike_norm_dcc_copula(theta, udata):
    N, T = np.shape(udata)
    llf = np.zeros((T,1))
    trdata = np.array(norm.ppf(udata).T, ndmin=2)


    Rt, veclRt =  dcceq(theta,trdata)

    for i in range(0,T):
        llf[i] = -0.5* np.log(np.linalg.det(Rt[:,:,i]))
        llf[i] = llf[i] - 0.5 *  np.matmul(np.matmul(trdata[i,:] , (np.linalg.inv(Rt[:,:,i]) - np.eye(N
    llf = np.sum(llf)

    return -llf
```

```
def dcceq(theta,trdata):
    T, N = np.shape(trdata)

    a, b = theta

    if min(a,b)<0 or max(a,b)>1 or a+b > .999999:
        a = .9999 - b

    Qt = np.zeros((N, N ,T))

    Qt[:,:,0] = np.cov(trdata.T)

    Rt =  np.zeros((N, N ,T))
    veclRt =  np.zeros((T, int(N*(N-1)/2)))
```

```
    Rt[:,:,0] = np.corrcoef(trdata.T)

    for j in range(1,T):
        Qt[:,:,j] = Qt[:,:,0] * (1-a-b)
        Qt[:,:,j] = Qt[:,:,j] + a * np.matmul(trdata[[j-1]].T, trdata[[j-1]])
        Qt[:,:,j] = Qt[:,:,j] + b * Qt[:,:,j-1]
        Rt[:,:,j] = np.divide(Qt[:,:,j] , np.matmul(np.sqrt(np.array(np.diag(Qt[:,:,j]), ndmin=2)).T ,

    for j in range(0,T):
        veclRt[j, :] = vecl(Rt[:,:,j].T)
    return Rt, veclRt
```

```
model_parameters = {}
udata_list = []

def run_garch_on_return(rets, udata_list, model_parameters):
    for x in rets:
        am = arch_model(rets[x], dist = 't')
        short_name = x.split()[0]
        model_parameters[short_name] = am.fit(disp='off')
        udata = garch_t_to_u(rets[x], model_parameters[short_name])
        udata_list.append(udata)
    return udata_list, model_parameters
```

```
retvol = [0 for i in range(4)]
irithvol = [0 for i in range(4)]
for i in range(4):
    model_parameters = {}
    udata_list = []
    dccData = pd.DataFrame(prices[i]['logReturn'][-254:-1].values,
                       columns = ['return'],
                       index = prices[i][-254:-1].index)
    dccData['irith'] = Irithmics[i]['shortProb'].values*100
    udata_list, model_parameters = run_garch_on_return(dccData, udata_list, model_parameters)
    cons = ({'type': 'ineq', 'fun': lambda x:  -x[0]  -x[1] +1})
    bnds = ((0, 0.5), (0, 0.9997))
    %time opt_out = minimize(loglike_norm_dcc_copula, [0.01, 0.95], args = (udata_list,), bounds=bnds,
    print(opt_out.success)
    print(opt_out.x)
    llf  = loglike_norm_dcc_copula(opt_out.x, udata_list)
    print(llf)
    trdata = np.array(norm.ppf(udata_list).T, ndmin=2)
    Rt, veclRt = dcceq(opt_out.x, trdata)
    stock_names = dccData.columns
    corr_name_list = []

    for j, name_a in enumerate(stock_names):
        if j == 0:
            pass
        else:
            for name_b in stock_names[:j]:
```

```
                corr_name_list.append(name_a + "-" + name_b)
    dcc_corr = pd.DataFrame(veclRt, index = dccData.index, columns= corr_name_list)
    retvol[i] = pd.DataFrame(sqrt(model_parameters['return'].conditional_volatility))
    irithvol[i] = pd.DataFrame(sqrt(model_parameters['irith'].conditional_volatility))
```

```
CPU times: total: 484 ms
Wall time: 514 ms
True
[1.55966853e-02 5.85187905e-13]
-0.48823898453809267
CPU times: total: 922 ms
Wall time: 971 ms
True
[0.03062957 0.33964555]
-0.33737449237679157
CPU times: total: 188 ms
Wall time: 219 ms
True
[4.61696480e-12 2.22424664e-01]
-0.1824765084738777
CPU times: total: 328 ms
Wall time: 380 ms
True
[0.         0.70675387]
-0.21031054448430248
```

Scatter for illustration

```
for i in range(4):
    fff = pd.DataFrame(np.array(retvol[i].values),columns = ['hey'])
    fff['yo'] = np.array(irithvol[i].values)
    corr = fff.corr().iloc[1,0]
    plt.figure(figsize = (15,10), facecolor = (1,1,1))
    plt.scatter(retvol[i],irithvol[i],
                color = 'k',
                label = f'Correlation Coefficient: {corr.round(2)}')
    #xpoints = ypoints = plt.xlim()
    #plt.plot(xpoints, ypoints, linestyle='--', color='r', lw=3, scalex=False, scaley=True,
    #         label = '45 Degree Line')
    plt.xlabel(f'{stockList[i]} Conditional Volatility',
               fontsize = 25)
    plt.ylabel('Irithmics Conditional Volatility',
               fontsize = 25)
    plt.legend(fontsize = 15)
   # plt.title(f'{fileNames[i]} EGARCH Volatility vs Irithmics Volatility',
    #                    fontsize = 30)
    plt.legend(fontsize = 25)
    plt.xticks(fontsize = 25)
    plt.yticks(fontsize = 25)
    #plt.savefig('figures\plot {0}.png'.format(i+1))
    plt.close()
```

Final Test - Volatility With Exogenous Covariate

Rugarch in R

```
## implement with rugarch
require(rugarch)
require(cowplot)
require(gridGraphics)
require(svMisc)
require(quantmod)
require(dplyr)
require(tidyverse)
require(tseries)
require(rugarch)
require(xts)
require(PerformanceAnalytics)
require(fGarch)
require(sgt)
require(MASS)
require(gridExtra)
require(zoo)
require(forecast)
require(tidyverse)

symbols = c('LLOY.L', 'TSCO.L', 'RR.L', 'VOD.L')
irithFiles = c('agg.csv', 'tesco.csv', 'rr.csv', 'vod.csv')

for(i in 1:4){
  startDate = "2020-01-01"
  endDate = "2020-12-31"

  sharePrice = get.hist.quote(symbols[i],
                              start = startDate,
                              end = endDate,
                              quote = 'Close',
                              quiet = TRUE)

  returns = (diff(log(sharePrice$Close))) %>%
    na.omit()
  numericReturns = as.numeric(returns) %>%
    na.omit()

  lloyAgg = read.csv(paste0('C:/Users/zaneh/Dissertation Jupyter/',
                            irithFiles[i]))
  lloyAgg = lloyAgg %>%
    dplyr::select(Date, shortProb)
  lloyAgg = lloyAgg[2:253,]

  test = 1:225
```

```
trainRet = returns[test]
testRet = returns[-test]
testAgg = lloyAgg[-test,]
trainAgg = lloyAgg[test,]

################### Fitting UGARCH SPec ##############################

data = cbind(trainRet,trainAgg$shortProb)



spec1 = ugarchspec(variance.model = list(model = 'eGARCH',
                                         garchOrder = c(1,1),
                                         submodel=NULL,
                                         external.regressors = NULL,
                                         variance.targeting = FALSE),
                   mean.model = list(armaOrder = c(0, 0)),
                   distribution.model = "std",
                   start.pars = list(), fixed.pars = list())

spec2 = ugarchspec(variance.model = list(model = 'eGARCH',
                                         garchOrder = c(1,1),
                                         submodel=NULL,
                                         external.regressors = matrix(data[,2]),
                                         variance.targeting = FALSE),
                   mean.model = list(armaOrder = c(0, 0)),
                   distribution.model = "std",
                   start.pars = list(), fixed.pars = list())

garch1 = ugarchfit(spec = spec1,
                   data = data[,1],
                   solver.control = list(trace = 0))
#garch1

garch2 = ugarchfit(spec = spec2,
                   data = data[,1],
                   solver.control = list(trace = 0))



fittedSigmas = data.frame("Date" = garch2@model$modeldata$index,
                      "Univariate" = garch1@fit$sigma,
                      "Multivariate" = garch2@fit$sigma)
# ggplot(data = fittedSigmas) +
#   geom_line(aes(x = Date,
#                 y = Univariate,
#                 color = 'Univariate'))+
#   geom_line(aes(x = Date,
#                 y = Multivariate,
#                 color = 'Multivariate'))+
#   scale_color_manual(values = c('red', 'black'))
```

```
spec = getspec(garch2)
setfixed(spec) = as.list(coef(garch2))
forecast = ugarchforecast(spec,
                          n.ahead = 1,
                          n.roll = 26,
                          data = testRet,
                          out.sample = 26)



# sigma(forecast)

spec2 = getspec(garch1)
setfixed(spec2) = as.list(coef(garch1))
forcast2 = ugarchforecast(spec2,
                          n.ahead = 1,
                          n.roll = 26,
                          data = testRet,
                          out.sample = 26)
# sigma(forcast2)

sigma_forecasts = data.frame('Date' = forcast2@model$modeldata$index,
                             'Univariate' = sigma(forcast2)[,],
                             'Multivariate' = sigma(forecast)[,])

# ggplot(data = sigma_forecasts) +
#   geom_line(aes(x = Date,
#                 y = Univariate,
#                 color = 'Univariate'))+
#   geom_line(aes(x = Date,
#                 y = Multivariate,
#                 color = 'Multivariate'))+
#   scale_color_manual(values = c('red', 'black'))


fittedfileName = paste0('C:/Users/zaneh/Dissertation Jupyter/',
                 unlist(strsplit(symbols[i], split = '\\.'))[1],"_fitted.csv")
forecastfileName = paste0('C:/Users/zaneh/Dissertation Jupyter/',
                          unlist(strsplit(symbols[i], split = '\\.'))[1],"_forc.csv")
unifileName = paste0('C:/Users/zaneh/Dissertation Jupyter/',
                     unlist(strsplit(symbols[i], split = '\\.'))[1],"_unicoef.csv")
multifileName = paste0('C:/Users/zaneh/Dissertation Jupyter/',
                       unlist(strsplit(symbols[i], split = '\\.'))[1],"_multiforc.csv")

write.csv(fittedSigmas,
          fittedfileName)
write.csv(sigma_forecasts,
          forecastfileName)
write.csv(garch1@fit$matcoef,
          unifileName)
write.csv(garch2@fit$matcoef,
          multifileName)
```

```
}
```

Importing from rugarch results back in python

```
uniCoef = pd.read_csv('LLOY_unicoef.csv')
uniCoef.rename(columns = {'Unnamed: 0':'Parameter'})
multiCoef = pd.read_csv('LLOY_multiforc.csv')
multiCoef.rename(columns = {'Unnamed: 0':'Parameter'})
df1_styler = uniCoef.style.applymap(color_negative_red, subset=['Pr(>|t|)']).set_caption(f'Univariate Pa
df2_styler = multiCoef.style.applymap(color_negative_red, subset=['Pr(>|t|)']).set_caption(f'Multivariat

display_html(df1_styler._repr_html_()+df2_styler._repr_html_(), raw=True)
```

```
uniCoef = pd.read_csv('TSCO_unicoef.csv')
uniCoef.rename(columns = {'Unnamed: 0':'Parameter'})
multiCoef = pd.read_csv('TSCO_multiforc.csv')
multiCoef.rename(columns = {'Unnamed: 0':'Parameter'})
df1_styler = uniCoef.style.applymap(color_negative_red, subset=['Pr(>|t|)']).set_caption(f'Univariate Pa
df2_styler = multiCoef.style.applymap(color_negative_red, subset=['Pr(>|t|)']).set_caption(f'Multivariat

display_html(df1_styler._repr_html_()+df2_styler._repr_html_(), raw=True)
```

```
uniCoef = pd.read_csv('RR_unicoef.csv')
uniCoef.rename(columns = {'Unnamed: 0':'Parameter'})
multiCoef = pd.read_csv('RR_multiforc.csv')
multiCoef.rename(columns = {'Unnamed: 0':'Parameter'})
df1_styler = uniCoef.style.applymap(color_negative_red, subset=['Pr(>|t|)']).set_caption(f'Univariate Pa
df2_styler = multiCoef.style.applymap(color_negative_red, subset=['Pr(>|t|)']).set_caption(f'Multivariat

display_html(df1_styler._repr_html_()+df2_styler._repr_html_(), raw=True)
```

```
uniCoef = pd.read_csv('VOD_unicoef.csv')
uniCoef.rename(columns = {'Unnamed: 0':'Parameter'})
multiCoef = pd.read_csv('VOD_multiforc.csv')
multiCoef.rename(columns = {'Unnamed: 0':'Parameter'})
df1_styler = uniCoef.style.applymap(color_negative_red, subset=['Pr(>|t|)']).set_caption(f'Univariate Pa
df2_styler = multiCoef.style.applymap(color_negative_red, subset=['Pr(>|t|)']).set_caption(f'Multivariat

display_html(df1_styler._repr_html_()+df2_styler._repr_html_(), raw=True)
```

Plot the results

```
fileAbbrev = ['LLOY', 'TSCO', 'RR', 'VOD']
```

```
for i in range(4):
    fitFile = f'{fileAbbrev[i]}_fitted.csv'
```

```
    forcFile = f'{fileAbbrev[i]}_forc.csv'

    fittedVals = pd.read_csv(fitFile)
    fittedVals =fittedVals[['Date','Univariate', 'Multivariate']]
    fittedVals = fittedVals.set_index('Date')
    forcVals = pd.read_csv(forcFile)
    forcVals =forcVals[['Date','Univariate', 'Multivariate']]
    forcVals = forcVals.set_index('Date')
    plt.figure(figsize=(20,10),facecolor=(1, 1, 1))
    plt.plot(fittedVals['Univariate'],
            label = 'Univariate Fitted',
             color = 'red',
            linewidth = 4)
    plt.plot(forcVals['Univariate'],
            label = 'Univariate Forecast',
             color = 'red',
            linewidth = 4,
            linestyle='dashed')
    plt.plot(fittedVals['Multivariate'],
            label = 'Multivariate Fitted',
             color = 'black',
            linewidth = 4)
    plt.plot(forcVals['Multivariate'],
            label = 'Multivariate Forecast',
             color = 'black',
            linewidth = 4,
            linestyle='dashed')

    plt.axvline(x=225, color='green', linestyle='--', linewidth = 4)
    plt.title(f'{fileNames[i]} Univariate vs Multivariate EGARCH(1,1)',
                      fontsize = 30)
    plt.legend(fontsize = 25)
    plt.xticks(fontsize = 25)
    plt.yticks(fontsize = 25)
    plt.gca().xaxis.set_major_locator(mdates.DayLocator(interval=15))
    plt.gcf().autofmt_xdate()
    plt.legend(fontsize = 25)
    #plt.savefig('figures\plot {0}.png'.format(i+1))
    plt.close()
```

Check The MSE, MAE

```
UniMAE = [0 for i in range(4)]
MultiMAE = [0 for i in range(4)]
UniMSE = [0 for i in range(4)]
MultiMSE = [0 for i in range(4)]


for i in range(len(stockList)):
    forcFile = f'{fileAbbrev[i]}_forc.csv'
    forcVals = pd.read_csv(forcFile)
```

```
    forcVals =forcVals[['Date','Univariate', 'Multivariate']]
    forcVals = forcVals.set_index('Date')
    #Indexing the test data

    testData = prices[i].logReturn.loc[forcVals.index]
    sqctest = testData.sub(testData.mean()).pow(2)
    #calculating mae/mse
    UniMAE[i] = mean_absolute_error(sqctest,
                                    forcVals['Univariate'])
    UniMSE[i] = mean_squared_error(sqctest,
                                    forcVals['Univariate'])
    MultiMAE[i] = mean_absolute_error(sqctest,
                                    forcVals['Multivariate'])
    MultiMSE[i] = mean_squared_error(sqctest,
                                    forcVals['Multivariate'])


fileNames, UniMAE, MultiMAE, UniMSE, MultiMSE = np.array(fileNames), np.array(UniMAE), np.array(MultiMA

backTest = pd.DataFrame(fileNames, columns = ['Company'])
backTest['Univariate MAE'] = UniMAE
backTest['Multivariate MAE'] = MultiMAE
backTest['Univariate MSE'] = UniMSE
backTest['Multivariate MSE'] = MultiMSE
```

```
sns.set_theme()
cm = sns.palplot(sns.diverging_palette(150, 275, s=80, l=55, n=9))
```

Figure 30: png

```
(backTest.style
    .highlight_min(subset=['Univariate MAE','Multivariate MAE', 'Univariate MSE', 'Multivariate MSE'],
                   color = 'lightgreen')
 .highlight_max(subset=['Univariate MAE','Multivariate MAE', 'Univariate MSE', 'Multivariate MSE'],
                   color = 'red')
 .set_properties(**{'text-align':'center'})
 .set_table_styles(styles)
 .set_precision(4)
 .set_caption('MAE and MSE for Squared Centered Returns vs Volatility Forecast')
 .hide_index())
 #.background_gradient(cmap = cm))
```