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,1

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

```r
#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(aggrega

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

Importing Stock Data (Python Portion)

```python
### 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}')
```

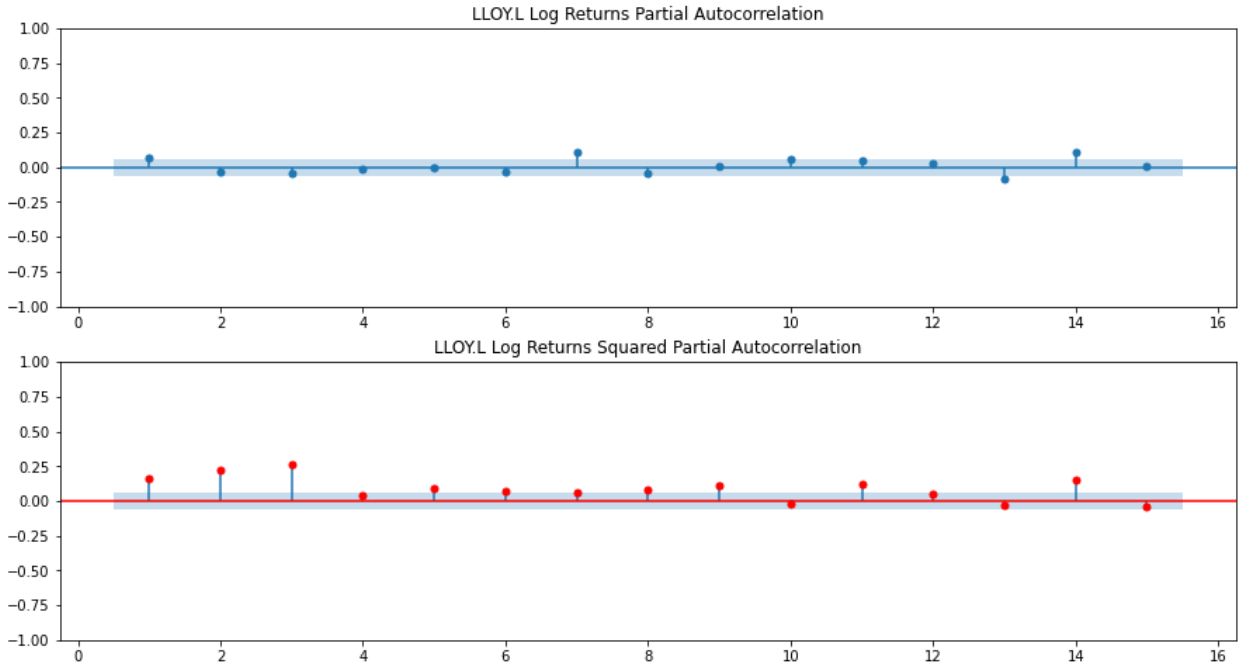

Figure 1: png

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,
```

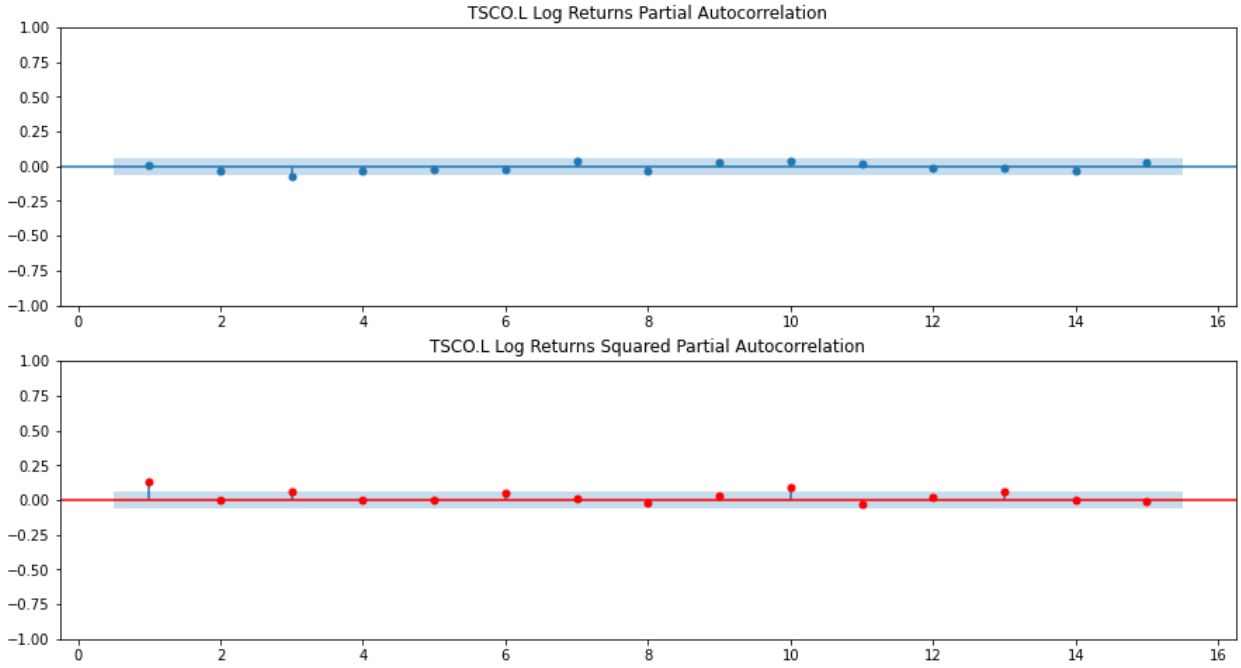

Figure 2: png

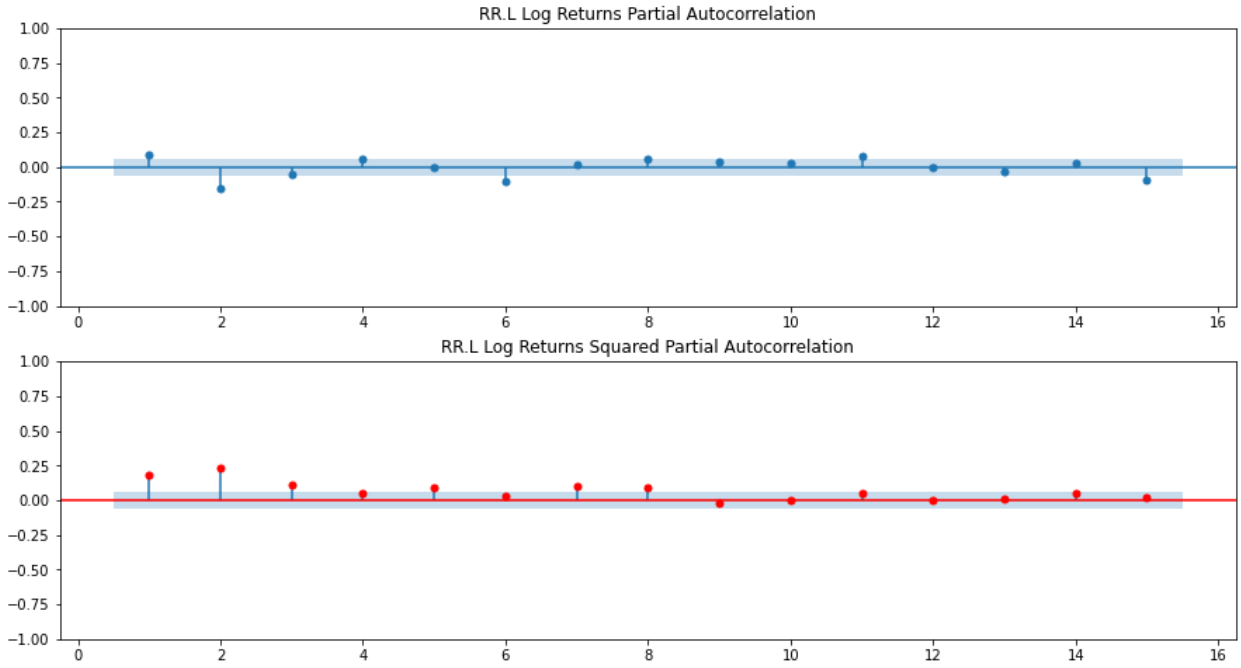

Figure 3: png

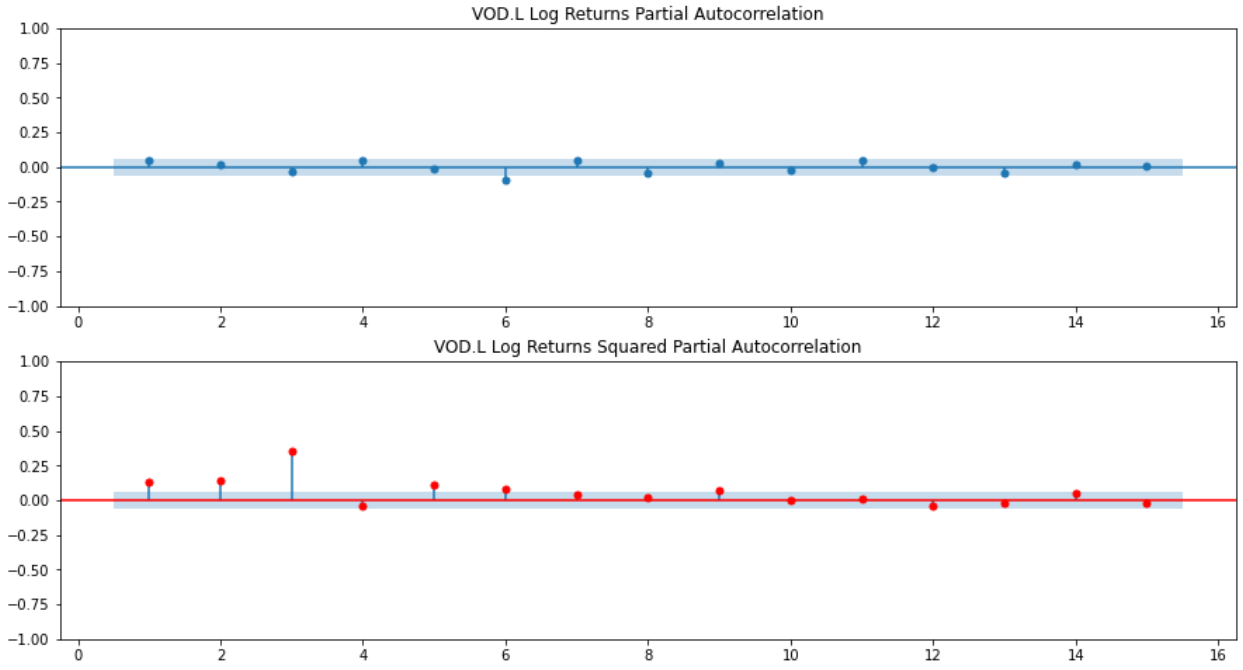

Figure 4: png

```
                                        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()
```

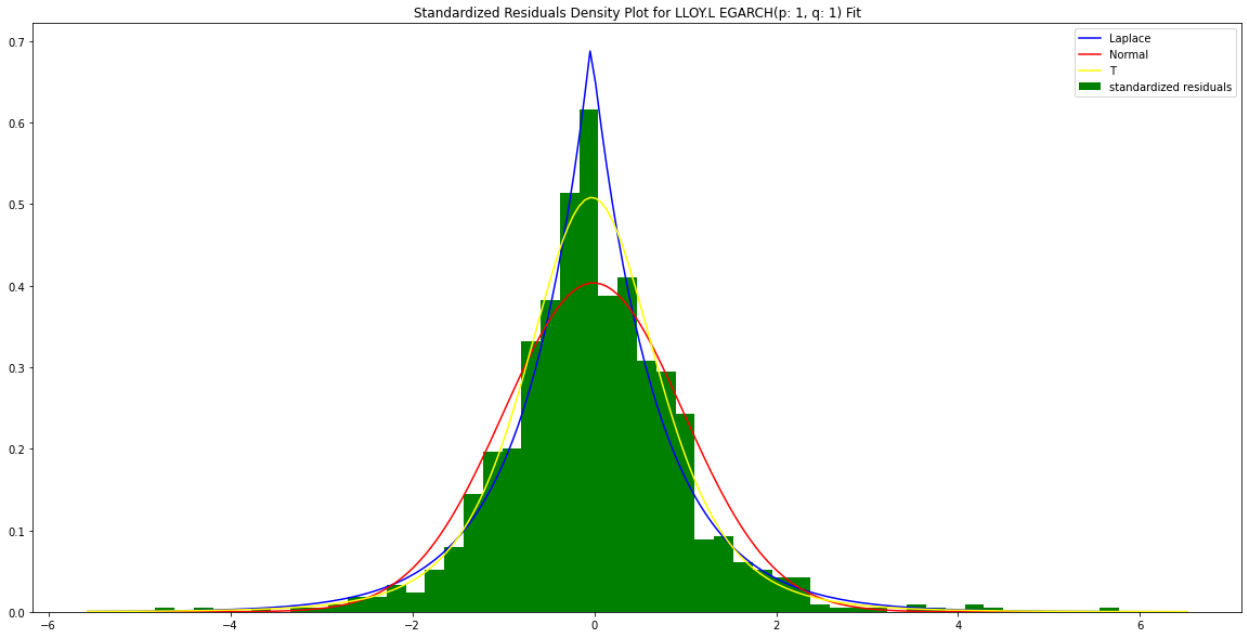

Figure 9: png

Lloyds, Tesco, Rolls Royce, Vodafone

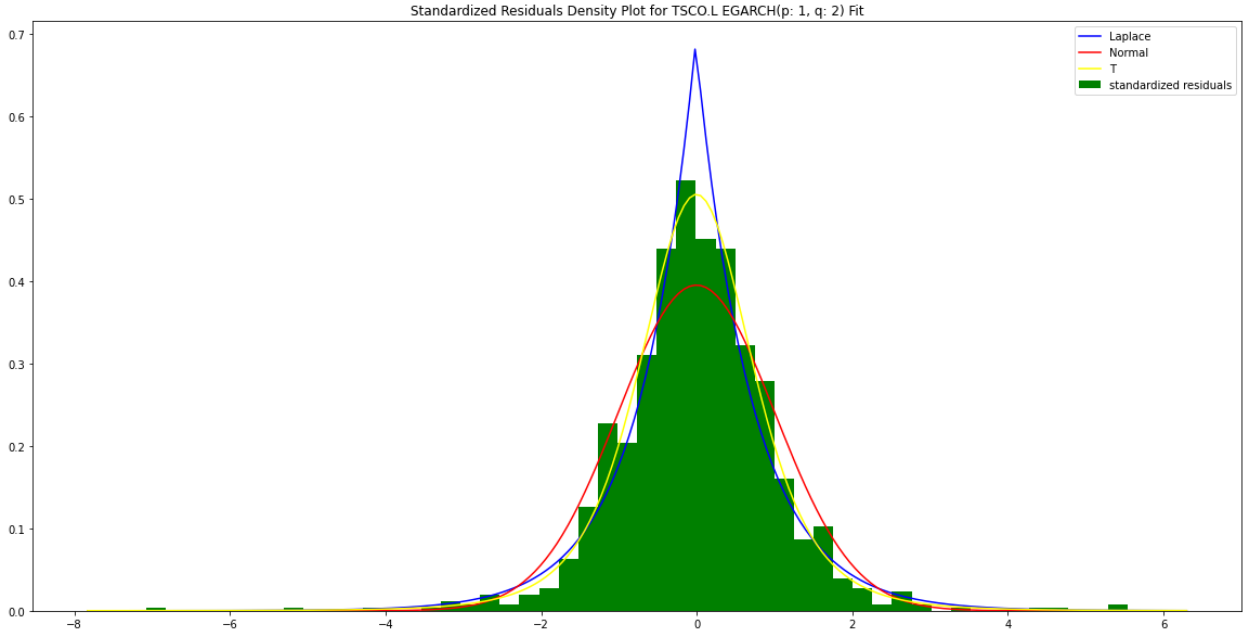

Figure 10: png

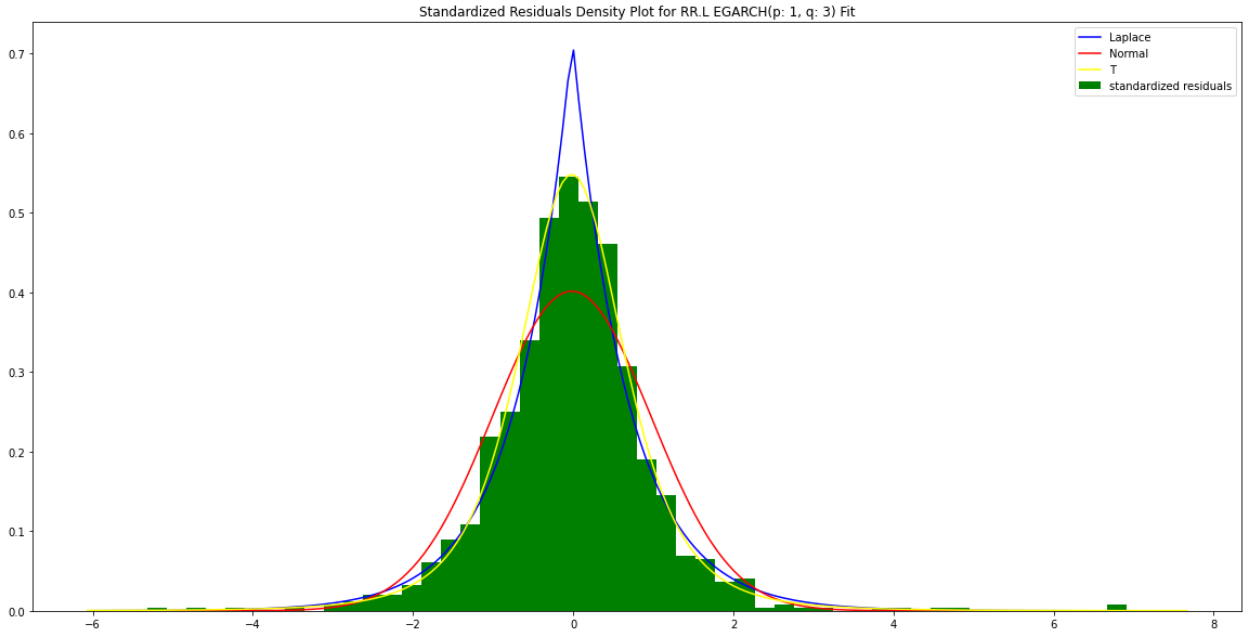

Figure 11: png

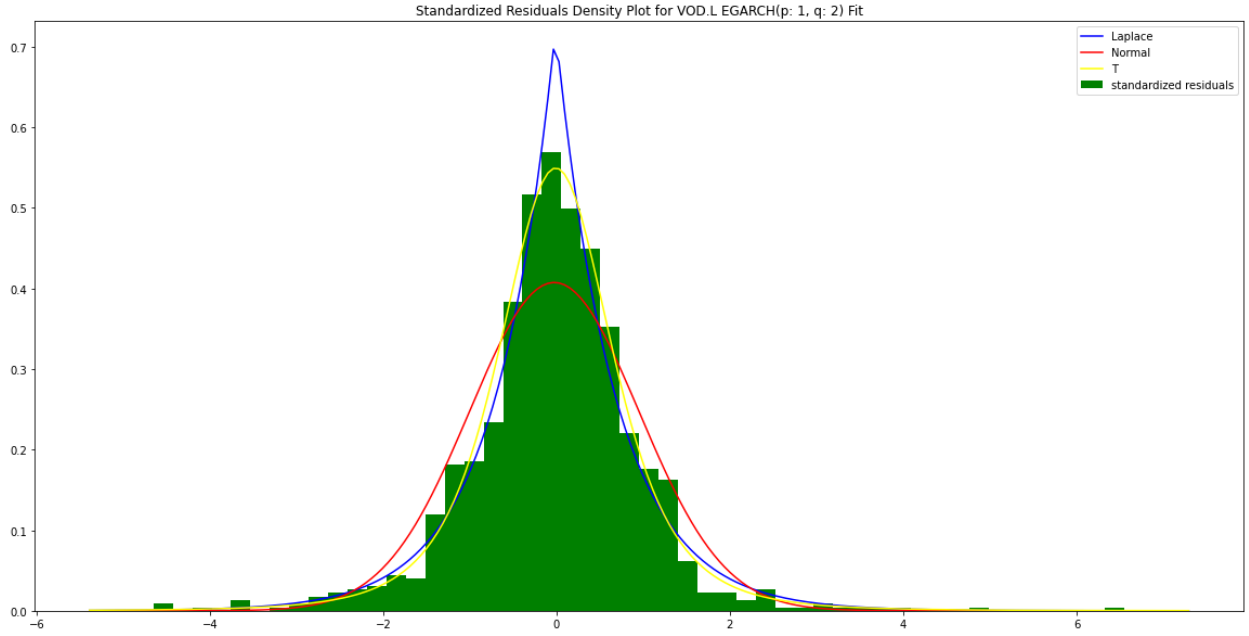

Figure 12: png

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
```

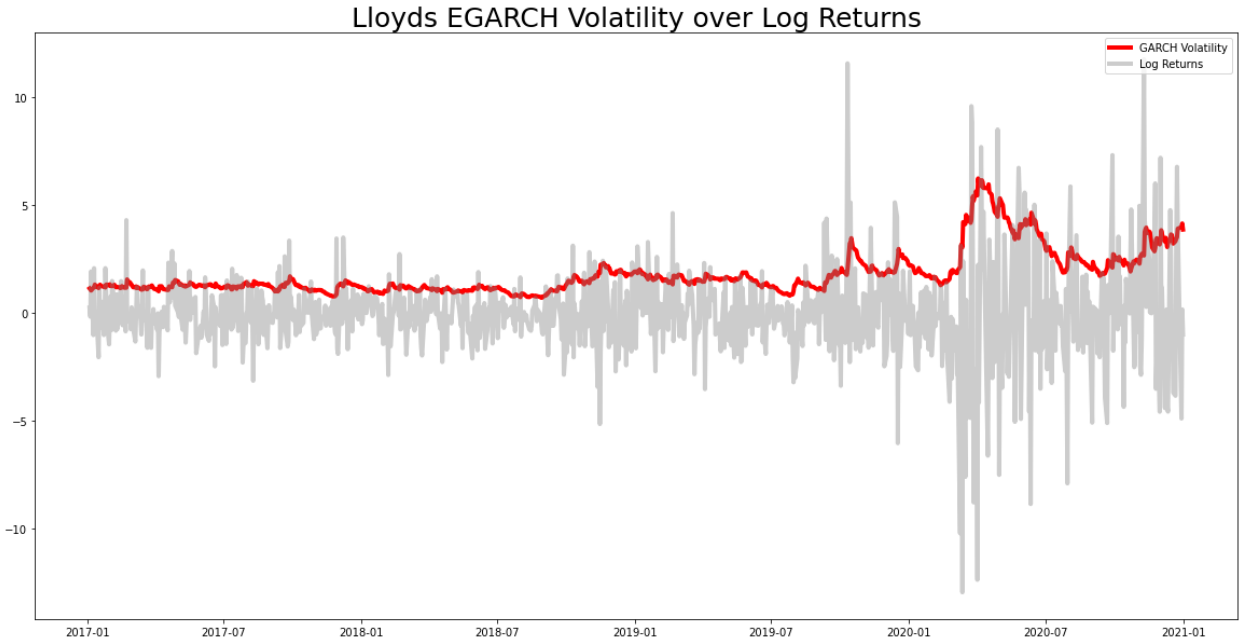

Figure 13: png

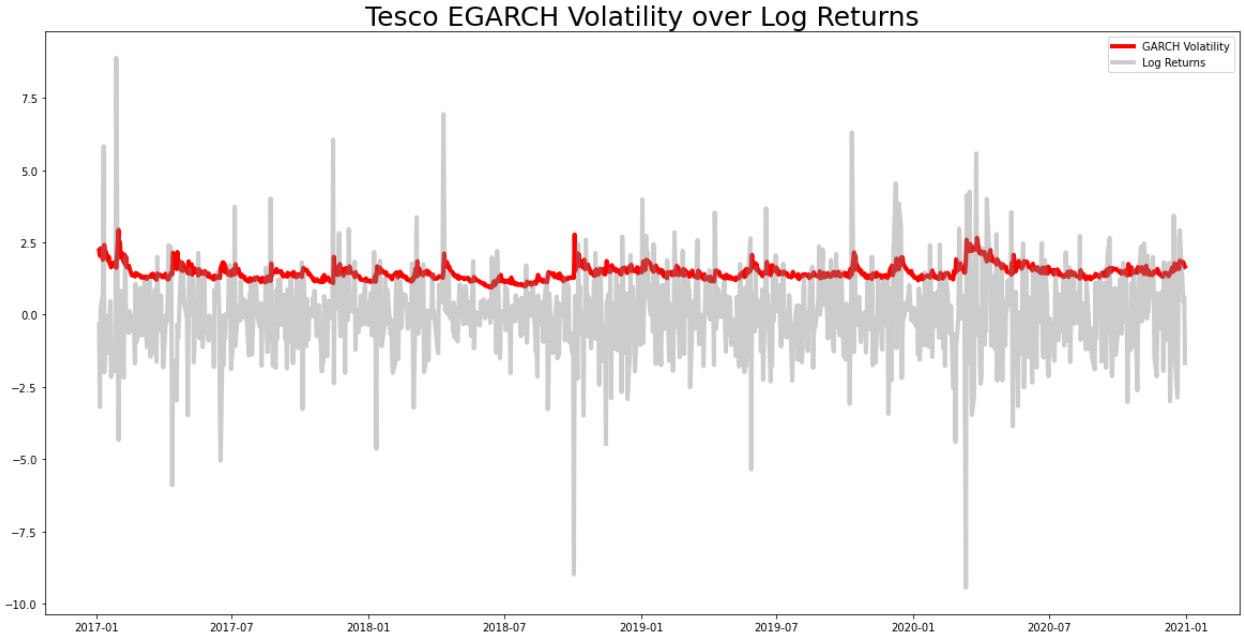

Figure 14: png

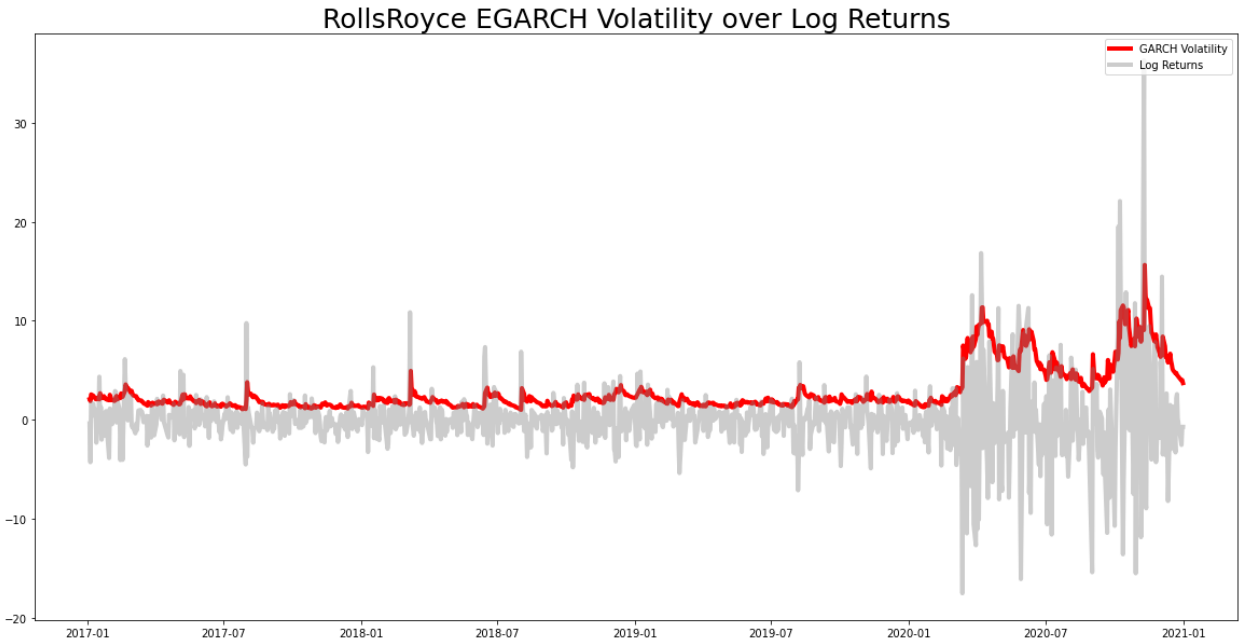

Figure 15: png

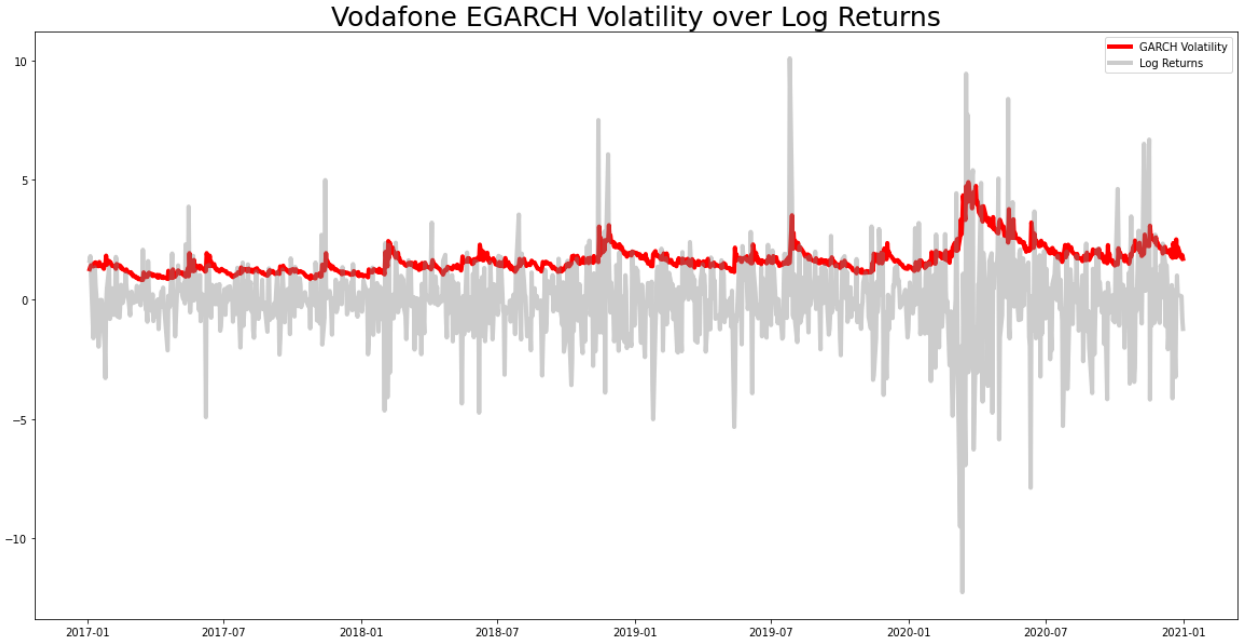

Figure 16: png

```
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
```

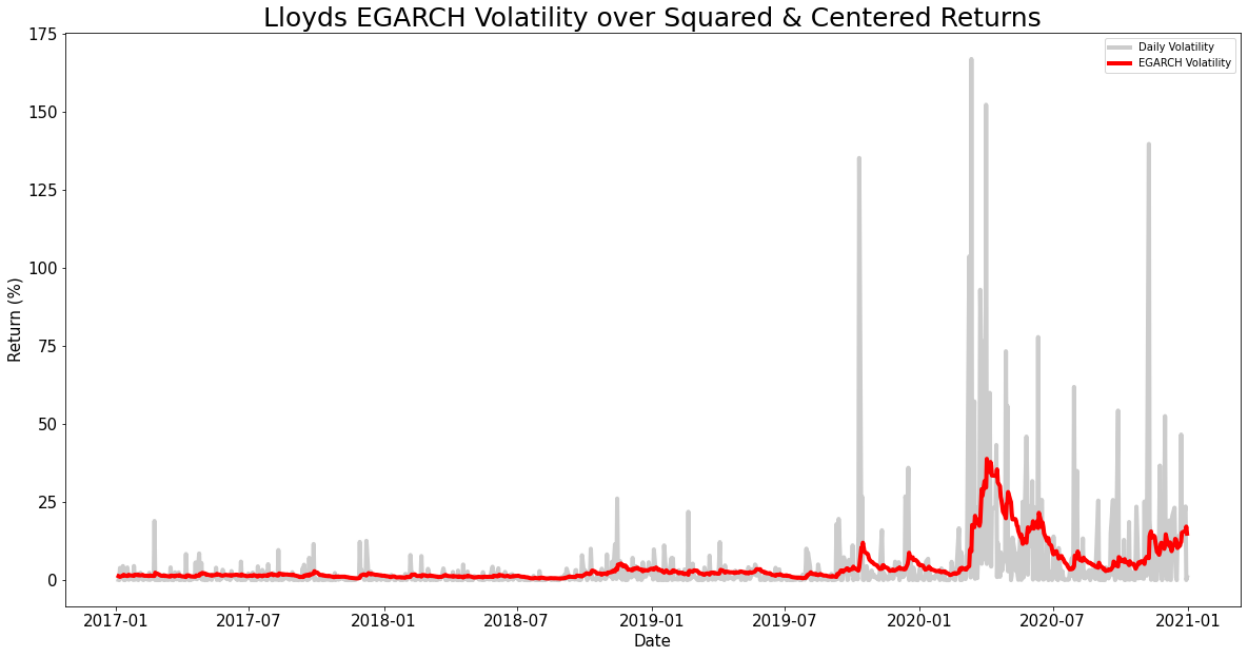

Figure 17: png

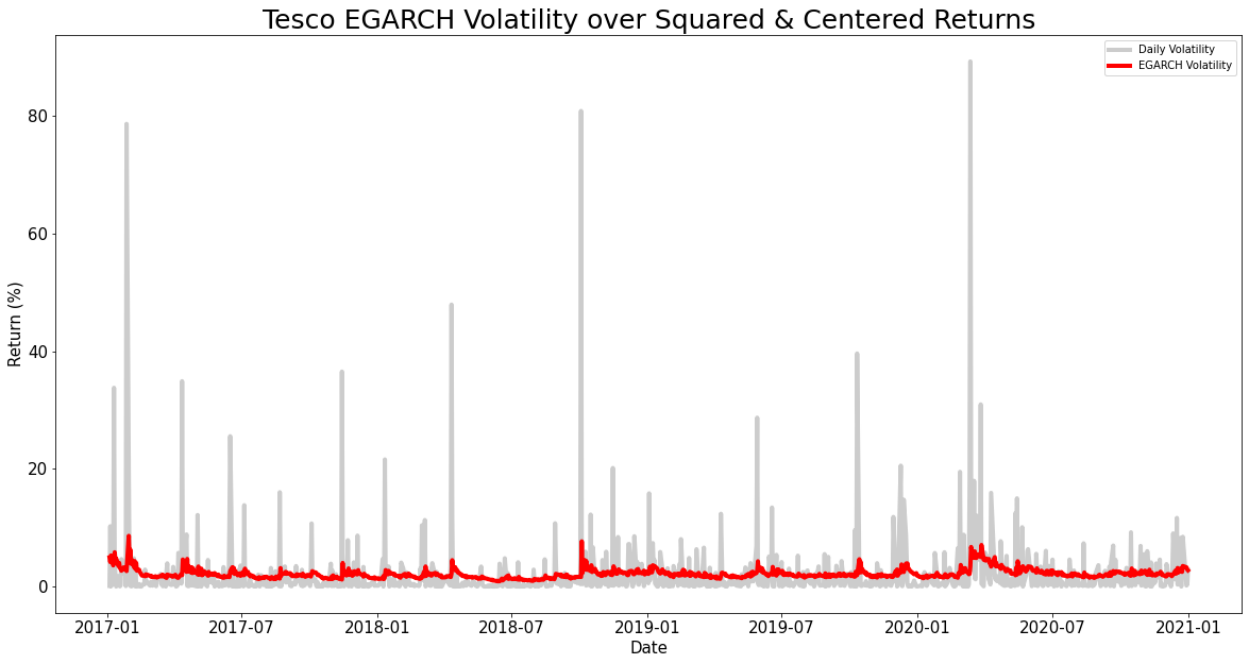

Figure 18: png

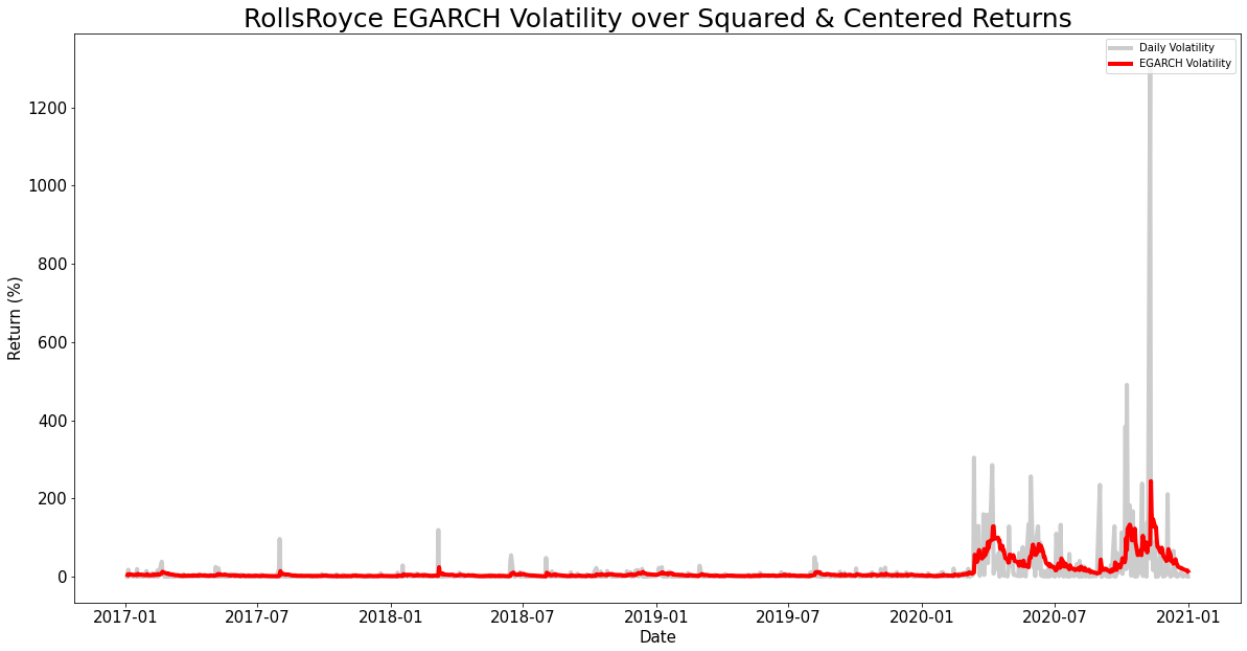

Figure 19: png

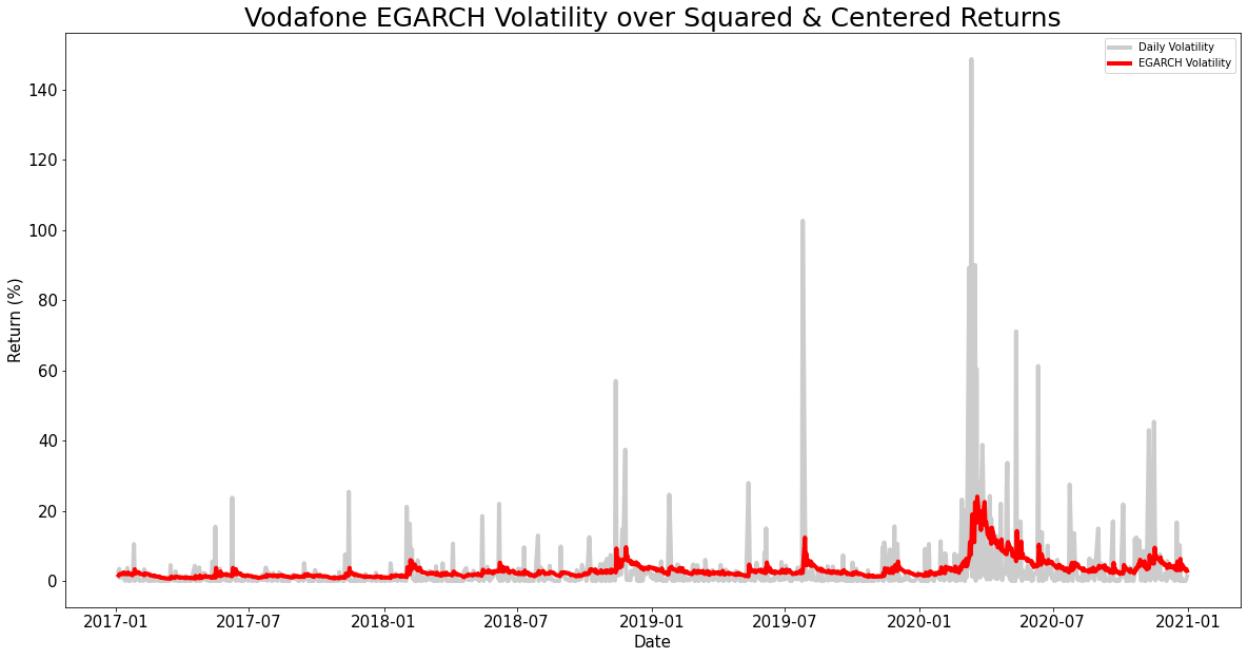

Figure 20: png

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
```

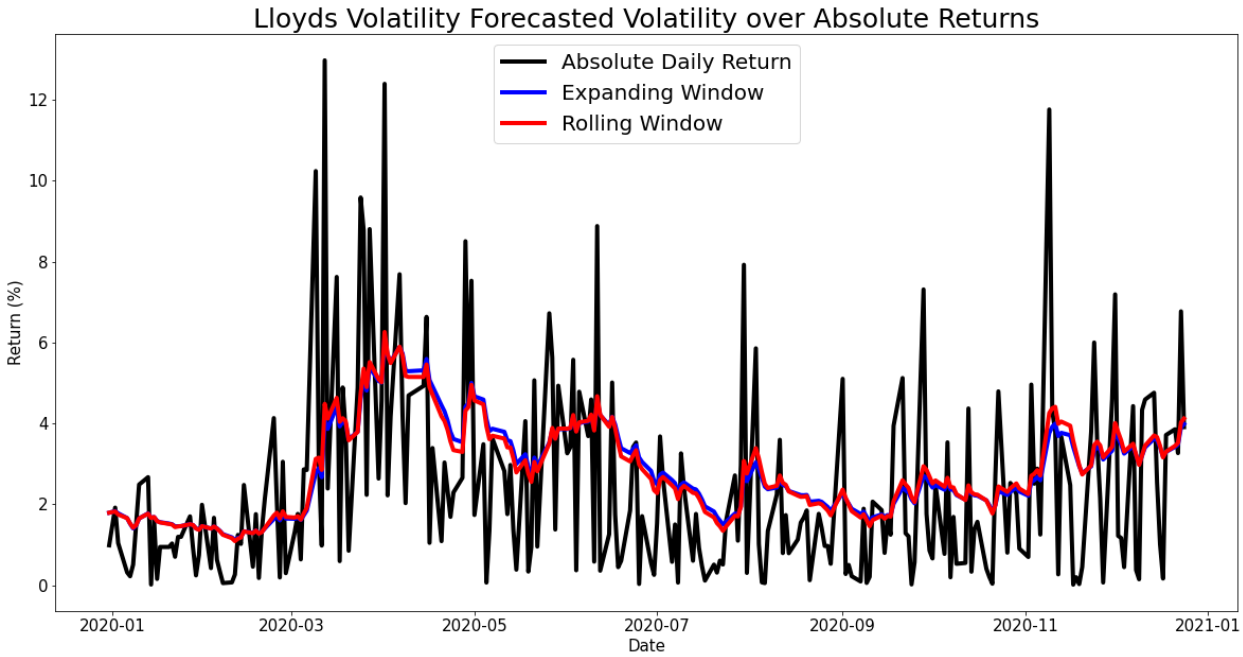

Figure 21: png

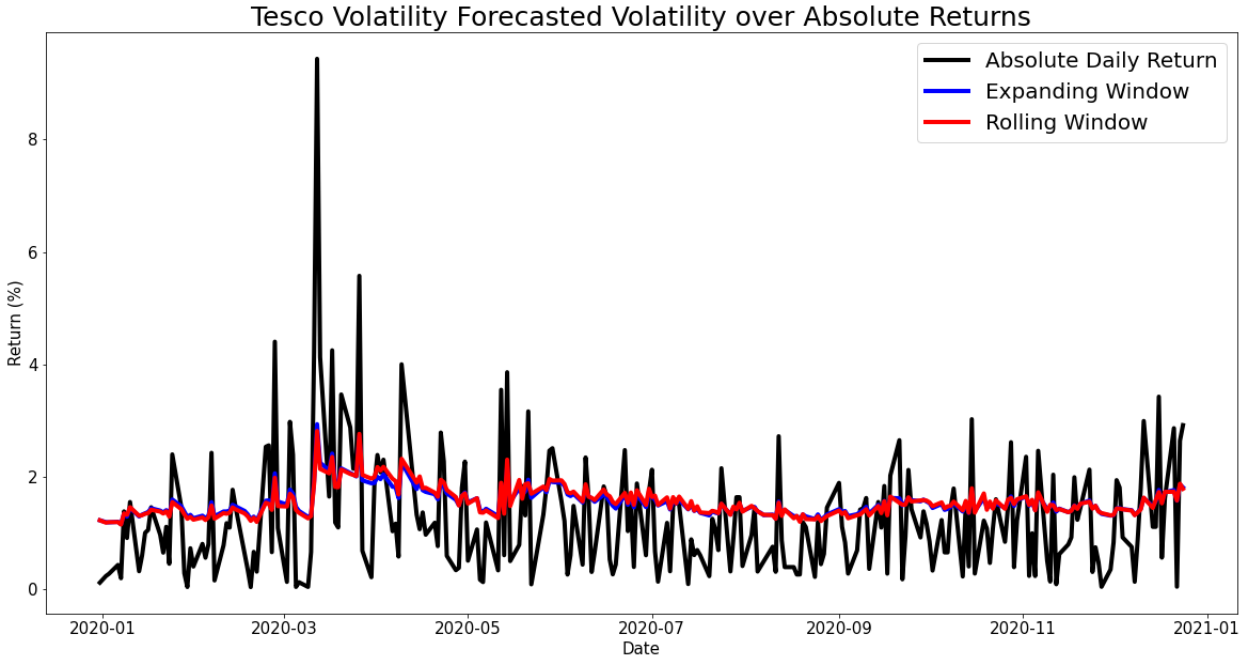

Figure 22: png

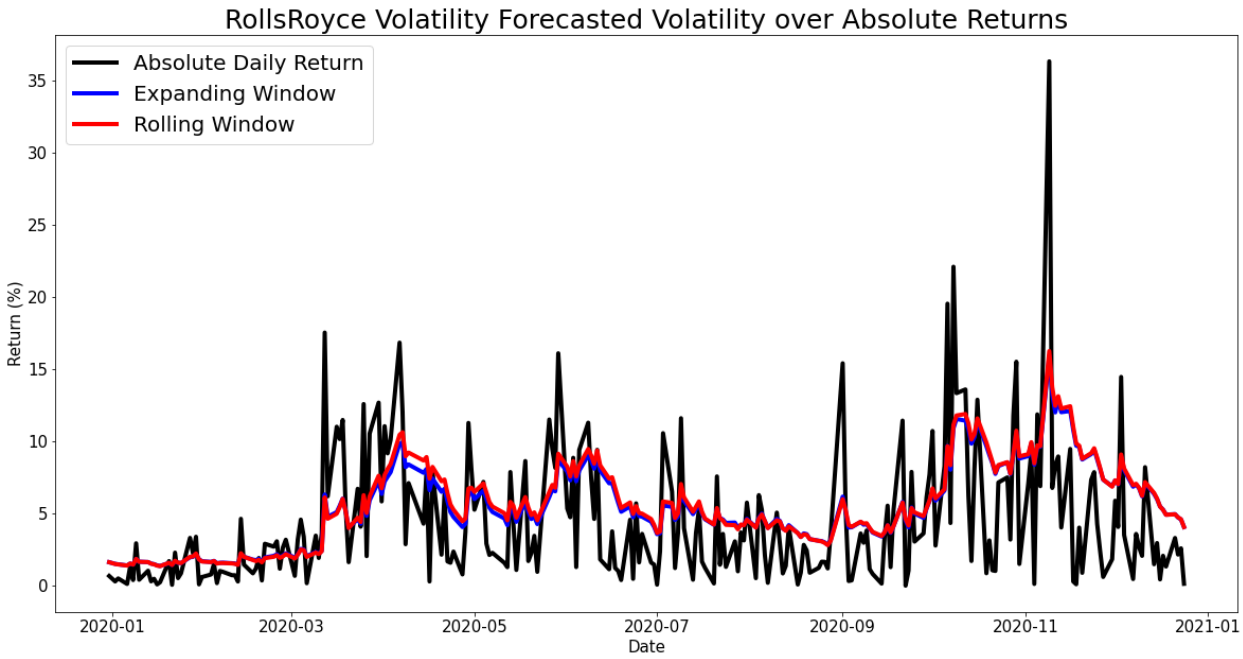

Figure 23: png

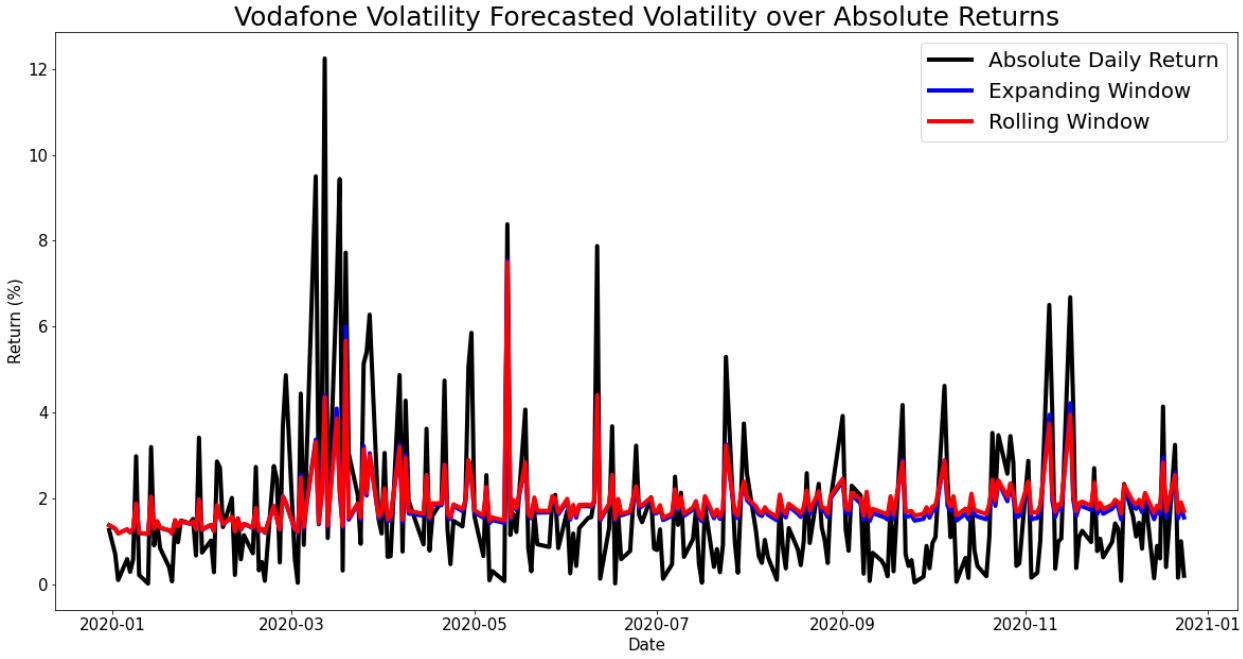

Figure 24: png

```
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)
```

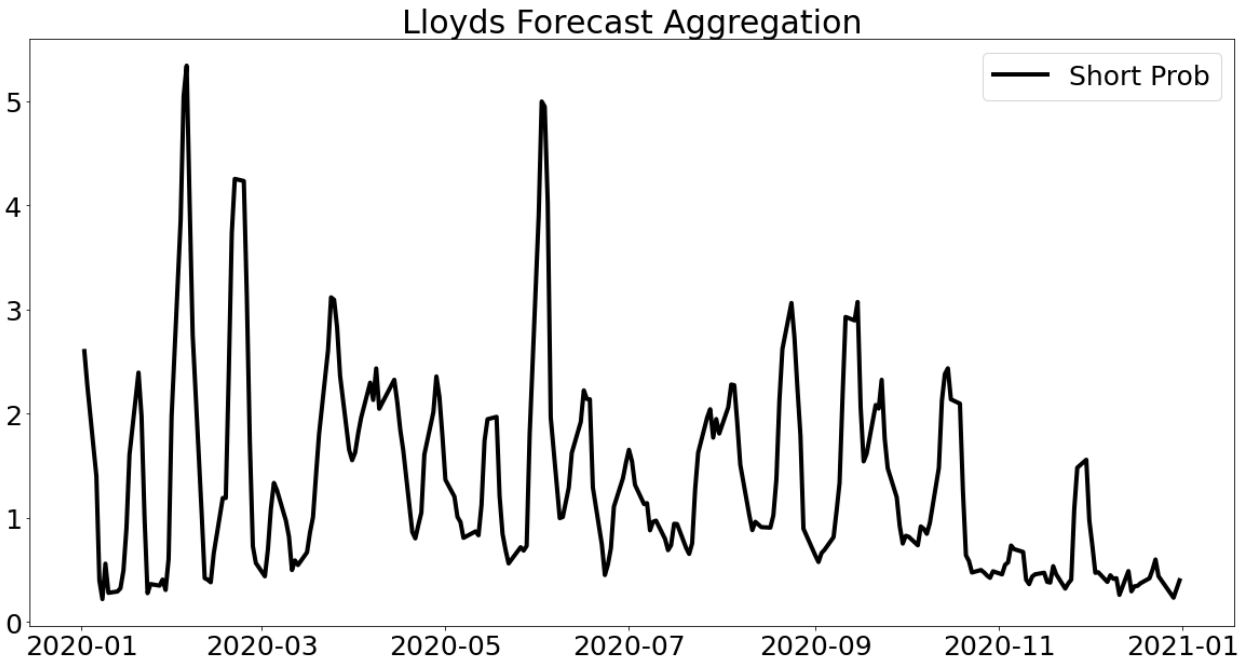

Figure 25: png

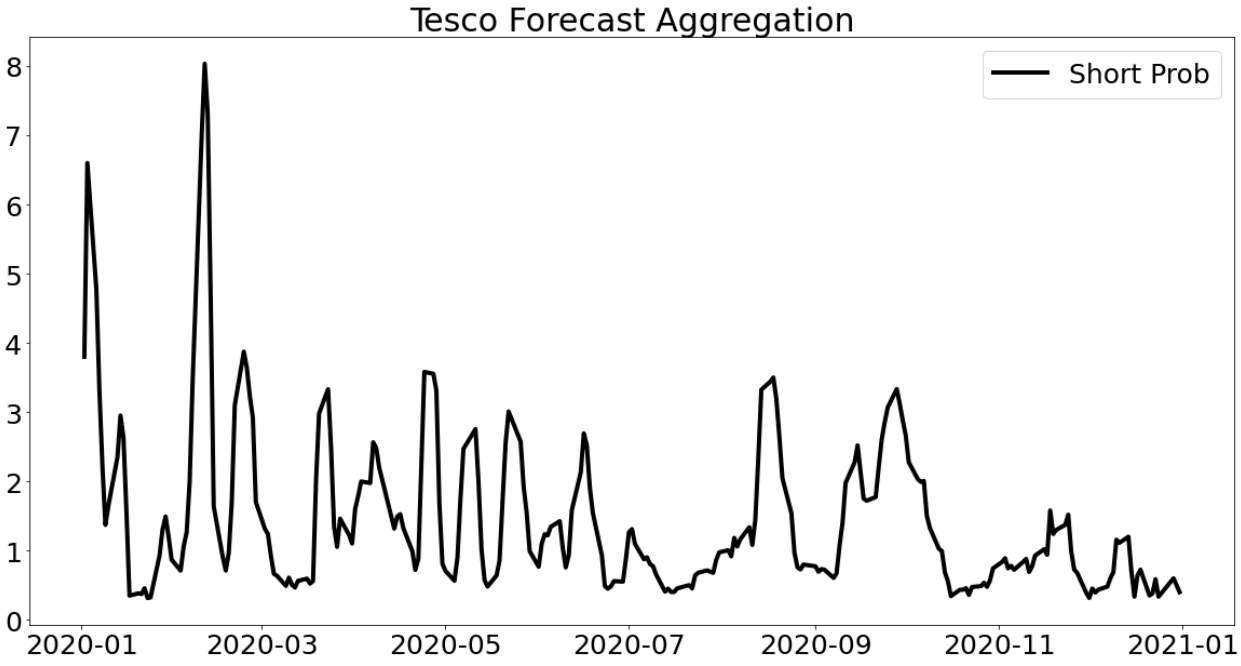

Figure 26: png

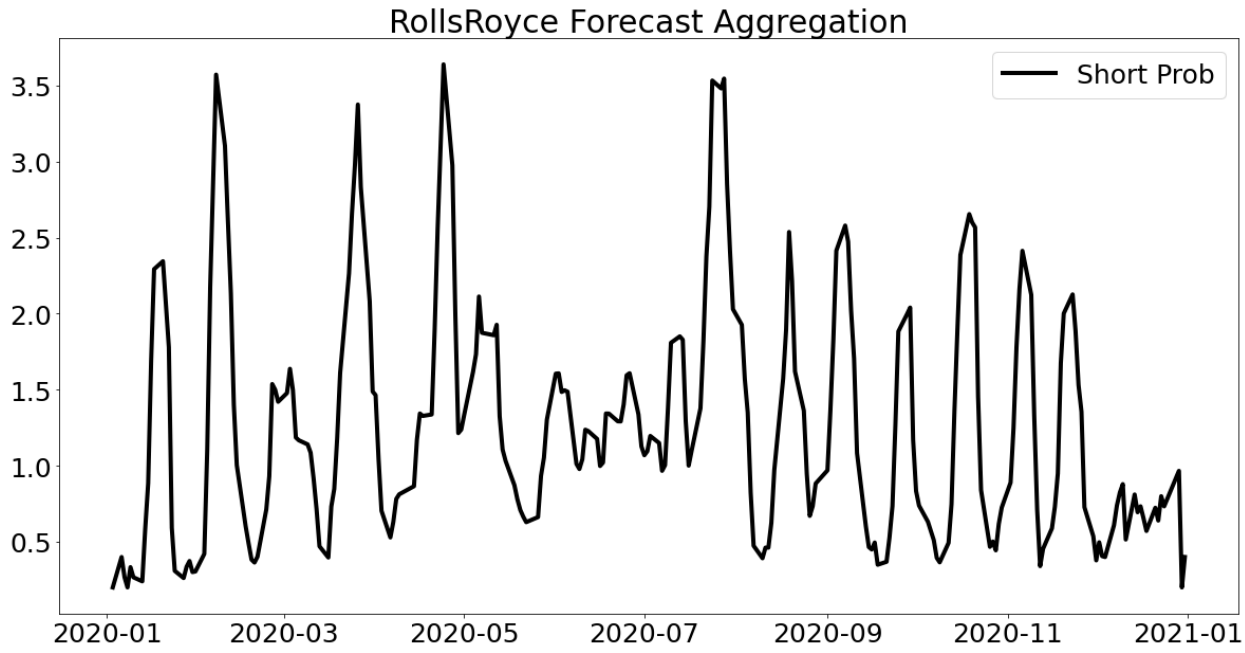

Figure 27: png

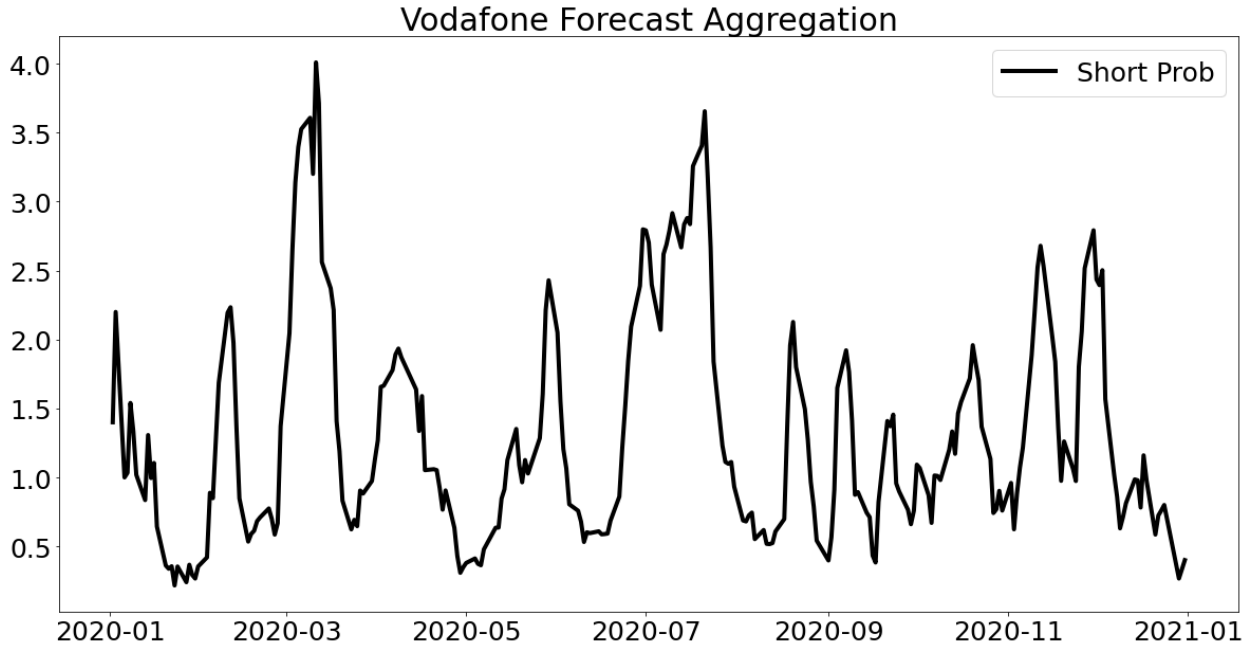

Figure 28: png

```
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()
```

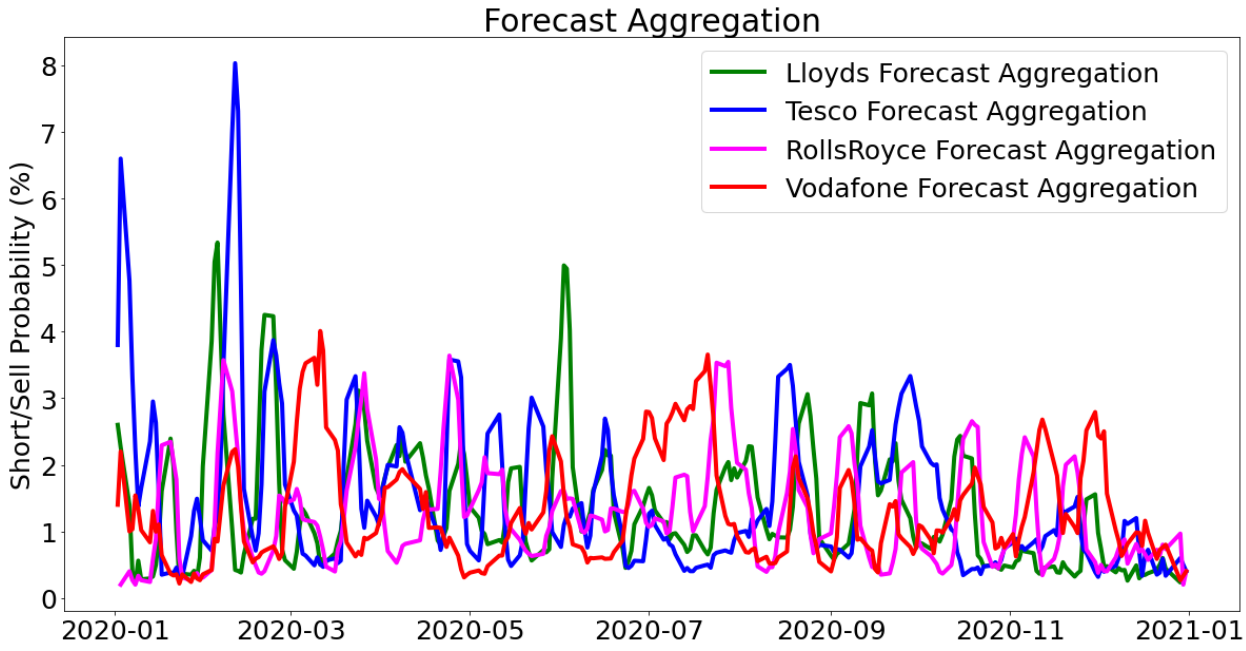

Figure 29: png

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 , n

    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, c
    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))
```