CONTENTS

1. INTRODUCTION

Volatility is a measure of risk, obtained by calculating the standard deviation of the annualized return over a given period of time. It is the rate at which the price of a security increases or decreases for a given set of returns. If the prices of a security fluctuate rapidly in a short time span, it is termed to have high volatility. If the prices of a security fluctuate slowly in a longer time span, it is termed to have a low volatility. The ability to forecast financial market volatility is important for portfolio selection, market timing and trading strategies, financial modeling and asset management. Accurate predictions of volatility enable market participants to make informed decisions, assess risk exposure, and develop effective trading strategies.

Firstly, volatility forecasts play an important role in portfolio construction and optimization. Incorporating volatility forecasts into portfolio models allow risk-adjusted returns, and enables one to identify the opportunities for portfolio rebalancing.

Secondly, volatility forecasts can be used by traders to determine entry and exit points, and adjust their trading positions based on expected market conditions. It also provides the traders with additional information to implement volatility based trading strategies such as volatility arbitrage or volatility breakout strategies.

Finally, volatility forecasts are an integral part of various financial models, including asset pricing models, Monte Carlo Simulations. The accuracy of these models is based on the volatility estimates chosen, which in turn affects the financial decision and risk management practices.

Hence, there has been a continuous need to find out accurate measurement of volatility. Early studies by Fama (1965), and Black (1976), in addition to numerous recent studies, all have observed that financial time series exhibits both volatility clustering and leptokurtosis. Furthermore, studies have shown that stock return and volatility changes are negatively correlated.

In line with the previously stated, estimation of volatility drives us away from linear models to statistical models. A study by Engle (1982) proposed to model time varying conditional variance through Auto Regressive Conditional Heteroskedasticity Process (ARCH), which is expected to capture the time varying nature of volatility. The key idea is to incorporate lagged squared error terms into the variance equations, creating a feedback mechanism between past volatility and current volatility. The model recognizes that extreme events in the past can influence the likelihood of extreme events in the future.

Bollerslev (1986) suggested a Generalized Autoregressive Conditional Heteroscedasticity (GARCH) model to overcome the number of parameters in ARCH. GARCH and ARCH both capture volatility clustering and leptokurtosis. But fail to capture the leverage effect. Hence, Nelson (1991) proposed an extension of ARCH model by using Exponential Generalized Autoregressive Conditional Heteroscedasticity (EGARCH) model, in order to track the asymmetric shocks of conditional variance.

The accuracy of these models plays a major role in selecting the appropriate one to forecast the volatility. This paper majorly focuses on evaluating the forecasting performance of ARCH, GARCH, GJR-GARCH, and EGARCH models.

With recent developments in Machine Learning Algorithms, there has been an increase in interest to combine the traditional models with learning algorithms. Upon investigating the forecasting ability of the GARCH models, we will discuss the scope of SVR based GARCH models, and different kernels that enhance the performance of the model.

In conclusion, this paper aims to provide a comprehensive analysis of forecasting volatility using GARCH models. Through theoretical explanations, and performance comparisons, it will contribute to the existing literature on volatility forecasting and offer valuable insights for both academic researchers and industry practitioners.

2. PROBLEM DEFINITION

The primary objective of this paper is to address the problem of accurately forecasting volatility in financial markets using GARCH models. Volatility forecasting, as mentioned in the previous section, plays a crucial role in risk management, option pricing, portfolio optimization, and financial models. Therefore, it is very important that the models chosen to forecast volatility have high accuracy scores, since the applications are less tolerant to error.

The paper answers questions related to selecting a GARCH model specification that adequately represents the volatility dynamics in the financial time series. The forecasting performance of GARCH models is essential to validate their effectiveness. Performance metrics such as Root Mean Square Error (RMSE) is used to evaluate the reliability of the models.

In order to improve the forecasting ability of the traditional models, we will discuss the scope of combining machine learning algorithms to GARCH models. We will be making use of the Support Vector Regression model along with GARCH in this paper. Since the data is not linearly separable, we will study various kernels and how the accuracy score can be enhanced.

Addressing these questions will contribute to the advancing field of volatility forecasting using GARCH models. The findings of this research have practical implications for financial practitioners and risk managers. Accurate volatility forecasting is essential for asset allocation decisions, risk management strategies, and derivative pricing. By identifying the most effective GARCH model specification for volatility forecasting, this study will contribute to improving decision-making processes in various financial applications.

3. TIME SERIES VOLATILITY MODELS

3.1. ARCH

The ARCH (Autoregressive Conditionally Heteroscedastic) model was introduced by Engle (1982). It is used to describe a changing variance, and assumes that the variance of the current error term is related to the size of the previous period error terms.

An ARCH model process can be written as $y_t = a_0 + \sum_{i=1}^{q} y_{t-q} + \epsilon_t$, where ϵ_t denotes the error terms. These ϵ_t are split into stochastic piece z_t and a time-dependent standard deviation σ_t characterizing the typical size of the terms so the $\epsilon_t = z_t \sigma_t$. The random variable z_t is a strong white noise process. The series σ_t^2 is modeled by $\sigma_t^2 = a_0 + \sum_{i=1}^{q} a_i \epsilon_{t-i}^2$, where $a_0 > 0$ and $a_i \geq 0,\ i > 0$ to ensure positive variance.

It is to be noted that the data should be properly formatted for estimation as a time-series. ARCH models are commonly used in modeling financial time series that exhibit time-varying variance and volatility clustering.

3.2. GARCH

GARCH is an extension of the ARCH model that incorporates a moving average component together with an autoregressive component. The introduction of a moving average component allows the model to model both the conditional change in variance over time as well as changes in the time-dependent variance.

GARCH (p,q) model, where p is the order of the GARCH terms σ^2 and q is the order of the ARCH terms ϵ^2, is a model with ϵ_t, the error terms, can be split into a scholastic part z_t and a time-dependent standard deviation σ_t characterizing the typical size of the terms so that $\epsilon_t = z_t \sigma_t$. The random variable z_t is standard gaussian. While σ_t^2 is modeled by $\sigma_t^2 = a_0 + \sum_{i=1}^{q} a_i \epsilon_{t-i}^2 + \sum_{i=1}^{p} \beta_i \sigma_{t-i}^2$, where a_0, a_i and β_i are non-negative constants with $a_i + \beta_j < 1$, it should be closer to unity for an accurate model.

3.3. EGARCH

The empirical results of studies applying ARCH/GARCH models to different countries' stock markets found that the assumption that; all of the shock effects on volatility have a symmetric distribution, which the ARCH and GARCH models are based on, is not true. The effect of the shock is solely dependent on characteristics of each market. As a result, the GARCH model has failed to detect properties such as leverage effect and heavy tailedness.

Thus, modified models were presented depending on non-linear distribution, so that they can also incorporate the relation that; negative shocks have stronger impact on increasing volatility in comparison to the effect of positive shocks. The issue of asymmetric condition was first proposed by Black (1976), following which there have been many empirical studies that supported that Black proposal, this includes the Exponential GARCH by Nelson (1991) and GJR-GARCH (1993).

Consider a return time series $r_t = \mu + \epsilon_t$, where μ is the expected return and ϵ_t is a zero mean white noise, $\epsilon_t = z_t \sigma_t$, where z_t is standard gaussian and;

$$ln\,(\sigma_t^2) = \omega + \sum_{i=1}^{p} a_i (|\epsilon_{i-1}| + \gamma_i \epsilon_{t-i}) + \sum_{j=1}^{q} \beta_j\, ln(\sigma_{t-j}^2)$$

3.4. GJR - GARCH

GJR-GARCH models (Glosten-Jagannathan-Runkle GARCH) capture the asymmetry in the response of volatility to negative and positive shocks. This model allows for different estimates for impact of negative and positive shocks in volatility. There is an additional parameter that captures the asymmetry effect.

$$\sigma_t^2 = \omega + \sum_{i=1}^{q} a_i \epsilon_{t-i}^2 + \sum_{j=1}^{s} \gamma_j \epsilon_{t-j}^2 I_{t-j} + \sum_{k=1}^{p} \beta_k \sigma_{t-k}^2$$

Where

$$I_{t-j} := \begin{cases} 0 \text{ , if } \epsilon_{t-j} \geq \mu \\ 1 \text{ , otherwise} \end{cases}$$

4. IMPLEMENTATION

4.1. INTRODUCTION

In this chapter we will study the forecasting ability of volatility forecasting models including ARCH, GARCH, EGARCH and GJR-GARCH. The comparative study is done with respect to the Root Mean Square Error (RMSE) values of the models.

This research is based on S&P 500 data over a period of twelve years, from 01/01/2010 to 01/01/2022. The period is intentionally chosen to incorporate significant fluctuations caused by the European Sovereign Debt Crisis (2010 - 2012) and the COVID-19 Pandemic. The data is retrieved with the help of yfinance library in python.

In the following sections, we will look at how volatility can be estimated for the S&P 500 data over the mentioned timeline in Python.

4.2. ARCH Model

Firstly, we will compute the return volatility in python:

```python
import numpy as np
from scipy.stats import norm
import scipy.optimize as opt
import yfinance as yf
import pandas as pd
import datetime
import time
from arch import arch_model
import matplotlib.pyplot as plt
from numba import jit
from sklearn.metrics import mean_squared_error as mse
import warnings
warnings.filterwarnings('ignore')
```

```
stocks = '^GSPC'
start = datetime.datetime(2010, 1, 1)
end = datetime.datetime(2022, 1, 1)
s_p500 = yf.download(stocks, start=start, end = end,
interval='1d')
```

```
[*********************100%***********************]  1
of 1 completed
```

```
ret = 100 * (s_p500.pct_change()[1:]['Adj Close'])
realized_vol = ret.rolling(5).std()

plt.figure(figsize=(10,6))
plt.plot(realized_vol.index, realized_vol)
plt.ylabel('Volatility')
plt.xlabel('Date')
plt.show()
```

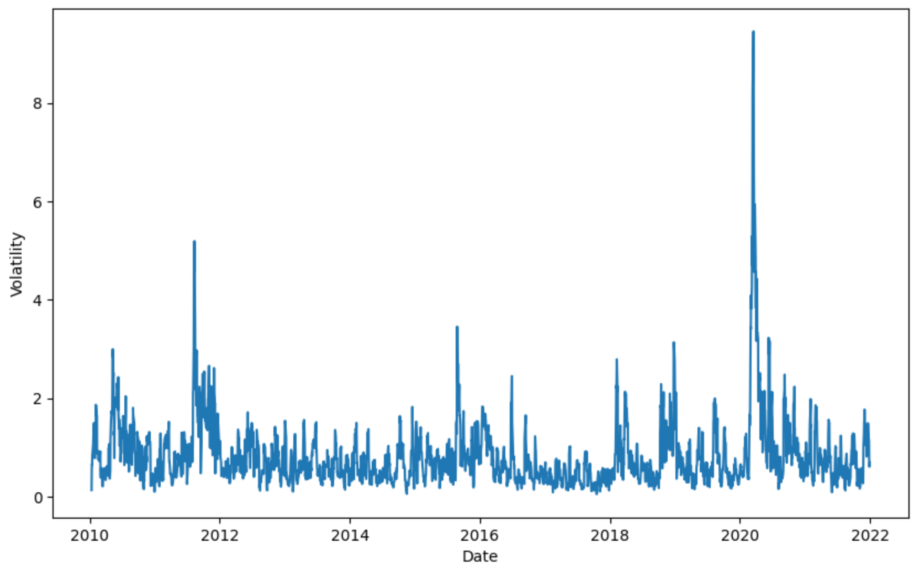

Figure 1: Realized Volatility - S&P 500

```
retv = ret.values

n = 252
split_date = ret.iloc[-n:].index

sgm2 = ret.var()
K = ret.kurtosis()
alpha = (-3.0 * sgm2 + np.sqrt(9.0 * sgm2 ** 2 - 12.0 *
                                     (3.0 * sgm2 - K) * K)) /
(6 * K)
omega = (1 - alpha) * sgm2
initial_parameters = [alpha, omega]
omega, alpha
```

```
(0.6220166329614314, 0.46836884362969905)
```

```
@jit(nopython = True, parallel = True)
def arch_likelihood(initial_parameters, retv):
    omega = abs(initial_parameters[0])
    alpha = abs(initial_parameters[1])
    T = len(retv)
    logliks = 0
    sigma2 = np.zeros(T)
    sigma2[0] = np.var(retv)
    for t in range(1, T):
        sigma2[t] = omega + alpha * (retv[t - 1]) ** 2
    logliks = np.sum(0.5 * (np.log(sigma2)+retv ** 2 /
sigma2))
    return logliks
```

```
logliks = arch_likelihood(initial_parameters, retv)
logliks
```

```
1479.7044188932502
```

```python
def opt_params(x0, retv):
    opt_result = opt.minimize(arch_likelihood, x0=x0, args =
(retv),
                                   method='Nelder-Mead',
                                   options={'maxiter':
5000})
    params = opt_result.x
    print('\nResults of Nelder-Mead minimization\n{}\n{}'
               .format(''.join(['-'] * 28), opt_result))
    print('\nResulting params = {}'.format(params))
    return params

params = opt_params(initial_parameters, retv)
```

```
Results of Nelder-Mead minimization
----------------------------
 final_simplex: (array([[0.69276167, 0.38924185],
       [0.69271891, 0.38929287],
       [0.69268202, 0.38920695]]),
array([1415.84846924, 1415.84846951, 1415.84847368]))
           fun: 1415.8484692408201
       message: 'Optimization terminated successfully.'
          nfev: 61
           nit: 33
        status: 0
       success: True
             x: array([0.69276167, 0.38924185])

Resulting params = [0.69276167 0.38924185]
```

```
def arch_apply(ret):
    omega = params[0]
    alpha = params[1]
    T = len(ret)
    sigma2_arch = np.zeros(T + 1)
    sigma2_arch[0] = np.var(ret)
    for t in range(1, T):
        sigma2_arch[t] = omega + alpha * ret[t - 1] ** 2
    return sigma2_arch
sigma2_arch = arch_apply(ret)
```

```
bic_arch = []
for p in range(1, 5):
    arch = arch_model(ret, mean='zero', vol='ARCH', p=p)\
                         .fit(disp='off')
    bic_arch.append(arch.bic)
    if arch.bic == np.min(bic_arch):
        best_param = p
        arch = arch_model(ret, mean='zero', vol='ARCH',
p=best_param)\
                 .fit(disp='off')
        print(arch.summary())
        forecast = arch.forecast(start=split_date[0])
        forecast_arch = forecast
```

```
rmse_arch = np.sqrt(mse(realized_vol[-n:] / 100,
                                 np.sqrt(forecast_arch\

.variance.iloc[-len(split_date):]
                                 / 100)))
print('The RMSE value of ARCH model is
{:.4f}'.format(rmse_arch))
```

```
The RMSE value of ARCH model is 0.0814
```

```
plt.figure(figsize=(10, 6))
plt.plot(realized_vol / 100, label='Realized Volatility')
plt.plot(forecast_arch.variance.iloc[-len(split_date):] /
100,
                  label='Volatility Prediction-ARCH')
plt.legend()
plt.show()
```

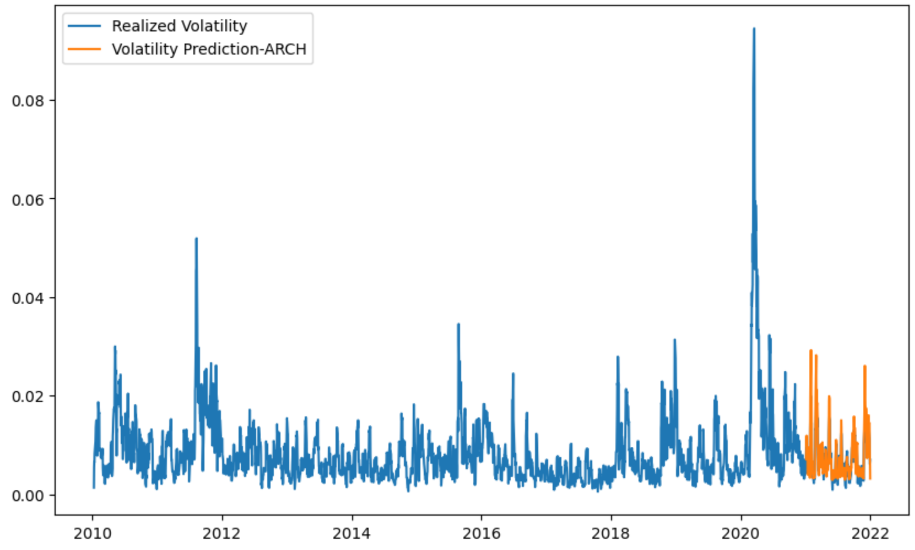

Figure 2: Volatility Prediction using ARCH model

4.3. GARCH Model

```
a0 = 0.0001
sgm2 = ret.var()
K = ret.kurtosis()
h = 1 - alpha / sgm2
alpha = np.sqrt(K * (1 - h ** 2) / (2.0 * (K + 3)))
beta = np.abs(h - omega)
omega = (1 - omega) * sgm2
initial_parameters = np.array([omega, alpha, beta])
print('Initial parameters for omega, alpha, and beta are
\n{}\n{}\n{}'
               .format(omega, alpha, beta))
```

```
Initial parameters for omega, alpha, and beta are
0.442246355322695
0.5167622671734421
0.022326619466433 66
```

```
retv = ret.values

@jit(nopython=True, parallel=True)
def garch_likelihood(initial_parameters, retv):
    omega = initial_parameters[0]
    alpha = initial_parameters[1]
    beta = initial_parameters[2]
    T =  len(retv)
    logliks = 0
    sigma2 = np.zeros(T)
    sigma2[0] = np.var(retv)
    for t in range(1, T):
        sigma2[t] = omega + alpha * (retv[t - 1]) ** 2 +
beta * sigma2[t-1]
    logliks = np.sum(0.5 * (np.log(sigma2) + retv ** 2 /
sigma2))
    return logliks
```

```
logliks = arch_likelihood(initial_parameters, retv)
logliks
```

```
1421.1085
```

```
def garch_constraint(initial_parameters):
    alpha = initial_parameters[0]
    gamma = initial_parameters[1]
    beta = initial_parameters[2]
    return np.array([1 - alpha - beta])

bounds = [(0.0, 1.0), (0.0, 1.0), (0.0, 1.0)]

def opt_paramsG(initial_parameters, retv):
    opt_result = opt.minimize(garch_likelihood,

x0=initial_parameters,

constraints=np.array([1 - alpha - beta]),
                                       bounds=bounds, args =
(retv),
                                       method='Nelder-Mead',
                                       options={'maxiter':
5000})
    params = opt_result.x
    print('\nResults of Nelder-Mead minimization\n{}\n{}'\
               .format('-' * 35, opt_result))
    print('-' * 35)
    print('\nResulting parameters = {}'.format(params))
    return params
params = opt_paramsG(initial_parameters, retv)
```

```
Results of Nelder-Mead minimization
-----------------------------------
 final_simplex: (array([[0.04069199, 0.17298474,
0.78804527],
       [0.04069778, 0.17306089, 0.78799501],
       [0.04067943, 0.17295978, 0.78811383],
       [0.04068475, 0.17301634, 0.78802469]]),
array([1005.13350622, 1005.13350888, 1005.13351164,
1005.13351301]))
           fun: 1005.1335062159609
       message: 'Optimization terminated successfully.'
          nfev: 171
           nit: 97
        status: 0
       success: True
             x: array([0.04069199, 0.17298474,
0.78804527])
-----------------------------------

Resulting parameters = [0.04069199 0.17298474 0.78804527]
```

```python
def garch_apply(ret):
    omega = params[0]
    alpha = params[1]
    beta = params[2]
    T = len(ret)
    sigma2 = np.zeros(T + 1)
    sigma2[0] = np.var(ret)
    for t in range(1, T):
        sigma2[t] = omega + alpha * ret[t - 1] ** 2 + beta *
sigma2[t-1]
    return sigma2
```

```
bic_garch = []
for p in range(1, 5):
    garch = arch_model(ret, mean='zero', vol='GARCH', p=p)\
                     .fit(disp='off')
    bic_garch.append(garch.bic)
    if garch.bic == np.min(bic_garch):
        best_param = p
        garch = arch_model(ret, mean='zero', vol='GARCH',
p=best_param)\
                 .fit(disp='off')
        print(garch.summary())
        forecast = garch.forecast(start=split_date[0])
        forecast_garch = forecast
```

```
rmse_garch = np.sqrt(mse(realized_vol[-n:] / 100,
                                 np.sqrt(forecast_garch\

.variance.iloc[-len(split_date):]
                                 / 100)))
print('The RMSE value of GARCH model is
{:.4f}'.format(rmse_garch))
```

```
The RMSE value of GARCH model is 0.0784
```

```
plt.figure(figsize=(10, 6))
plt.plot(realized_vol / 100, label='Realized Volatility')
plt.plot(forecast_garch.variance.iloc[-len(split_date):] /
100,
                  label='Volatility Prediction-GARCH')
plt.legend()
plt.show()
```

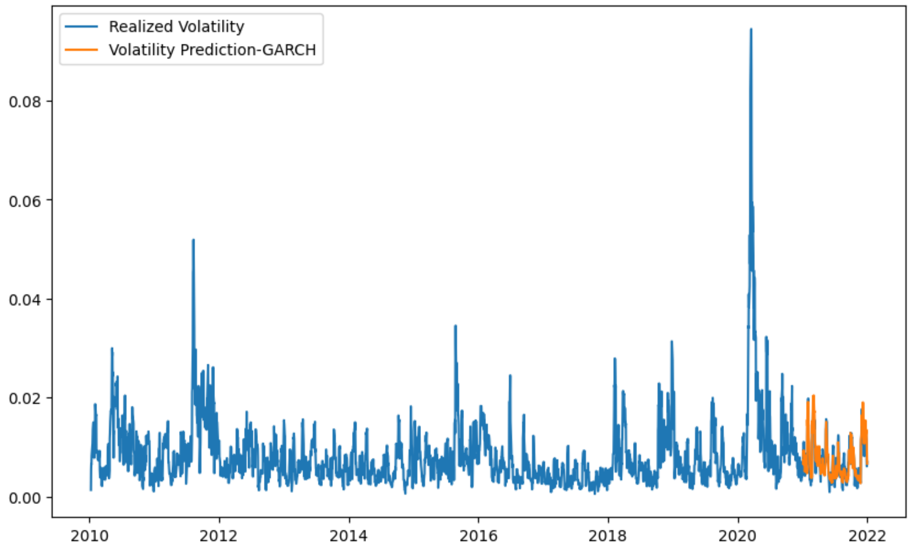

Figure 3: Volatility Prediction using GARCH Model

4.4. E-GARCH Model

```
bic_egarch = []
for p in range(1, 5):
    for q in range(1, 5):
        egarch = arch_model(ret, mean='zero', vol='EGARCH',
p=p, q=q)\
                          .fit(disp='off')
        bic_egarch.append(egarch.bic)
        if egarch.bic == np.min(bic_egarch):
            best_param = p, q
egarch = arch_model(ret, mean='zero', vol='EGARCH',
                             p=best_param[0],
q=best_param[1])\
                  .fit(disp='off')
print(egarch.summary())
forecast = egarch.forecast(start=split_date[0])
forecast_egarch = forecast
```

```
rmse_egarch = np.sqrt(mse(realized_vol[-n:] / 100,

np.sqrt(forecast_egarch.variance\
                                   .iloc[-len(split_date):]
/ 100)))
print('The RMSE value of EGARCH models is
{:.4f}'.format(rmse_egarch))
```

```
The RMSE value of EGARCH models is 0.0816
```

```
plt.figure(figsize=(10, 6))
plt.plot(realized_vol / 100, label='Realized Volatility')
plt.plot(forecast_egarch.variance.iloc[-len(split_date):] /
100,
                  label='Volatility Prediction-EGARCH')
plt.legend()
plt.show()
```

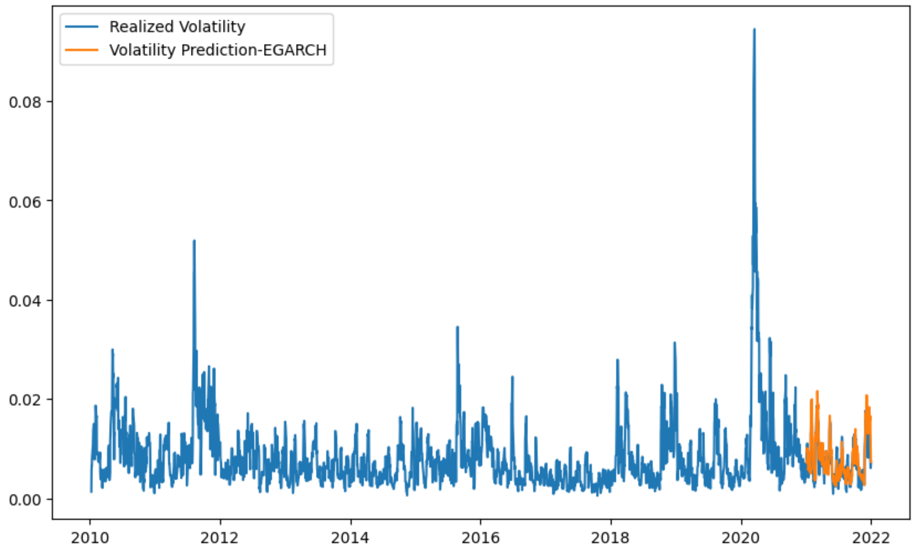

Figure 5: Volatility Prediction using EGARCH Model

4.5. GJR-GARCH Model

```
bic_gjr_garch = []
for p in range(1, 5):
    for q in range(1, 5):
        gjrgarch = arch_model(ret, mean='zero', p=p, o=1,
q=q)\
                            .fit(disp='off')
        bic_gjr_garch.append(gjrgarch.bic)
        if gjrgarch.bic == np.min(bic_gjr_garch):
            best_param = p, q
gjrgarch = arch_model(ret,mean='zero', p=best_param[0], o=1,

q=best_param[1]).fit(disp='off')
print(gjrgarch.summary())
forecast = gjrgarch.forecast(start=split_date[0])
forecast_gjrgarch = forecast
```

```
rmse_gjr_garch = np.sqrt(mse(realized_vol[-n:] / 100,
                                 np.sqrt(forecast_gjrgarch\

.variance.iloc[-len(split_date):]
                                 / 100)))
print('The RMSE value of GJR_GARCH model is
{:.4f}'.format(rmse_gjr_garch))
```

```
The RMSE value of GJR_GARCH model is 0.0779
```

```
plt.figure(figsize=(10, 6))
plt.plot(realized_vol / 100, label='Realized Volatility')
plt.plot(forecast_gjrgarch.variance.iloc[-len(split_date):]
/ 100,
                  label='Volatility Prediction-GJR-GARCH')
plt.legend()
plt.show()
```

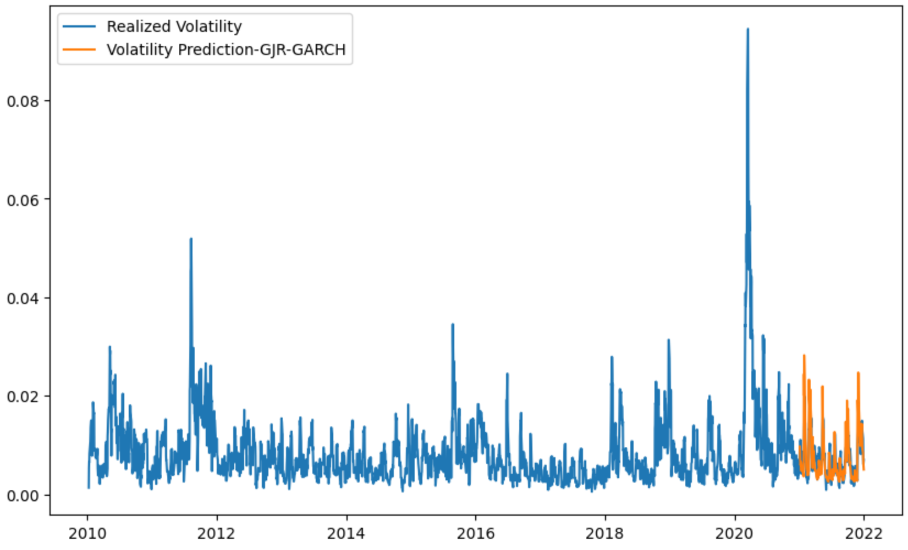

Figure 4: Volatility Prediction using GJR-GARCH Model

4.6. SVR-GARCH Model

Support vector machine (SVM) is a supervised learning algorithm that can be applied to both classification and regression problems. In regression, the aim is to find an optimal line called the hyperplane, that maximizes the margin and minimizes the error. This is called Support Vector Regression (SVR). In this section, we will apply SVR to the GARCH model and estimate the volatility of S&P 500 data over the period of twelve years.

Since the data chosen is not linearly separable, we will make use of kernel functions. The kernel function is used to map the initial nonlinear observations into a higher dimensional space so a linear classifier can be applied. In this study we will be applying three kernel functions; Linear kennel, Polynomial Kernel, and Radial Basis Function (RBF)

Estimating volatility in Python:

```
from sklearn.svm import SVR
from scipy.stats import uniform as sp_rand
from sklearn.model_selection import RandomizedSearchCV

realized_vol = ret.rolling(5).std()
realized_vol = pd.DataFrame(realized_vol)
realized_vol.reset_index(drop=True, inplace=True)

returns_svm = ret ** 2
returns_svm = returns_svm.reset_index()
del returns_svm['Date']

X = pd.concat([realized_vol, returns_svm], axis=1,
ignore_index=True)
X = X[4:].copy()
X = X.reset_index()
X.drop('index', axis=1, inplace=True)

realized_vol = realized_vol.dropna().reset_index()
realized_vol.drop('index', axis=1, inplace=True)
```

```python
svr_poly = SVR(kernel='poly', degree=2)
svr_lin = SVR(kernel='linear')
svr_rbf = SVR(kernel='rbf')

para_grid = {'gamma': sp_rand(),
                      'C': sp_rand(),
                      'epsilon': sp_rand()}
clf = RandomizedSearchCV(svr_lin, para_grid)
clf.fit(X.iloc[:-n].values,

realized_vol.iloc[1:-(n-1)].values.reshape(-1,))
predict_svr_lin = clf.predict(X.iloc[-n:])

predict_svr_lin = pd.DataFrame(predict_svr_lin)
predict_svr_lin.index = ret.iloc[-n:].index

rmse_svr_lin = np.sqrt(mse(realized_vol.iloc[-n:] / 100,
                                predict_svr_lin / 100))
print('The RMSE value of SVR with Linear Kernel is {:.6f}'
               .format(rmse_svr_lin))
```

```
The RMSE value of SVR with Linear Kernel is 0.000481
```

```python
realized_vol.index = ret.iloc[4:].index

plt.figure(figsize=(10, 6))
plt.plot(realized_vol / 100, label='Realized Volatility')
plt.plot(predict_svr_lin / 100, label='Volatility
Prediction-SVR-GARCH')
plt.legend()
plt.show()
```

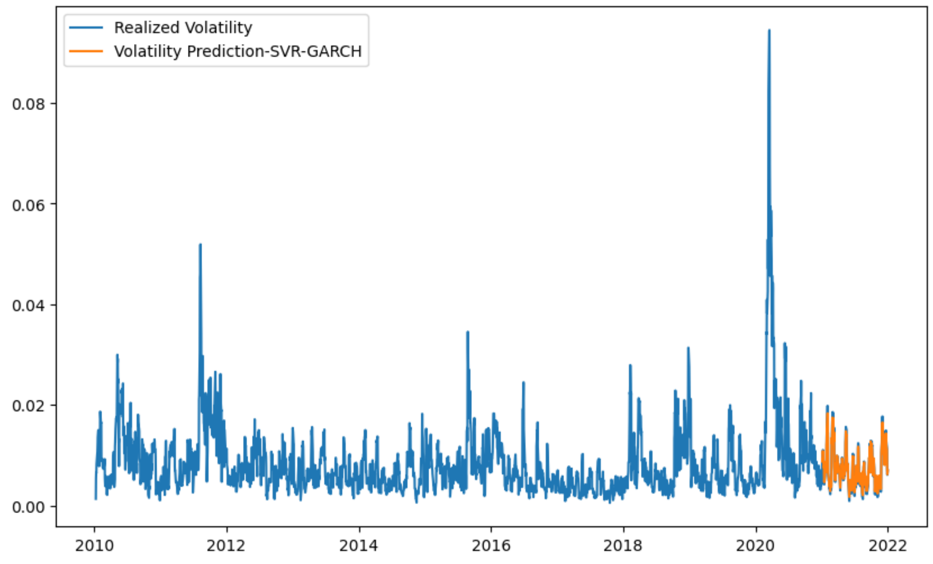

Figure 6: Volatility Prediction using SVR-GARCH (Linear Kernel)

```
para_grid = {'gamma': sp_rand(),
                     'C': sp_rand(),
                     'epsilon': sp_rand()}
clf = RandomizedSearchCV(svr_poly, para_grid)
clf.fit(X.iloc[:-n].values,

realized_vol.iloc[1:-(n-1)].values.reshape(-1,))
predict_svr_poly = clf.predict(X.iloc[-n:])


predict_svr_poly = pd.DataFrame(predict_svr_poly)
predict_svr_poly.index = ret.iloc[-n:].index

rmse_svr_poly = np.sqrt(mse(realized_vol.iloc[-n:] / 100,
                                     predict_svr_poly / 100))
print('The RMSE value of SVR with Polynomial Kernel is
{:.6f}'\
               .format(rmse_svr_poly))
```

```
The RMSE value of SVR with Polynomial Kernel is
0.002674
```

```
plt.figure(figsize=(10, 6))
plt.plot(realized_vol/100, label='Realized Volatility')
plt.plot(predict_svr_poly/100, label='Volatility
Prediction-SVR-GARCH')
plt.legend()
plt.show()
```

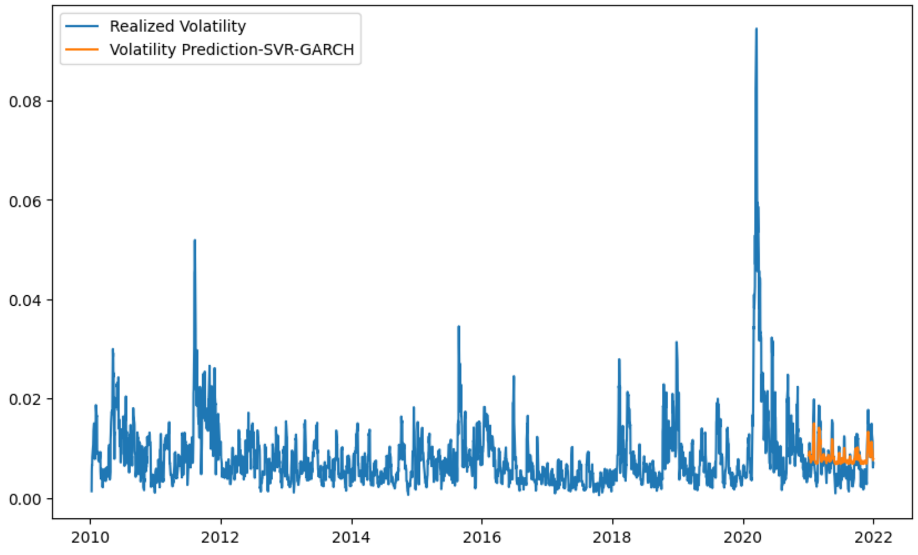

Figure 7: Volatility Prediction using SVR-GARCH (Polynomial kernel)

```
para_grid ={'gamma': sp_rand(),
                     'C': sp_rand(),
                     'epsilon': sp_rand()}
clf = RandomizedSearchCV(svr_rbf, para_grid)
clf.fit(X.iloc[:-n].values,

realized_vol.iloc[1:-(n-1)].values.reshape(-1,))
predict_svr_rbf = clf.predict(X.iloc[-n:])

predict_svr_rbf = pd.DataFrame(predict_svr_rbf)
predict_svr_rbf.index = ret.iloc[-n:].index

rmse_svr_rbf = np.sqrt(mse(realized_vol.iloc[-n:] / 100,
                                    predict_svr_rbf / 100))
```

```
print('The RMSE value of SVR with RBF Kernel is  {:.6f}'
                 .format(rmse_svr_rbf))
```

```
The RMSE value of SVR with RBF Kernel is  0.000378
```

```
plt.figure(figsize=(10, 6))
plt.plot(realized_vol / 100, label='Realized Volatility')
plt.plot(predict_svr_rbf / 100, label='Volatility
Prediction-SVR_GARCH')
plt.legend()
plt.show()
```

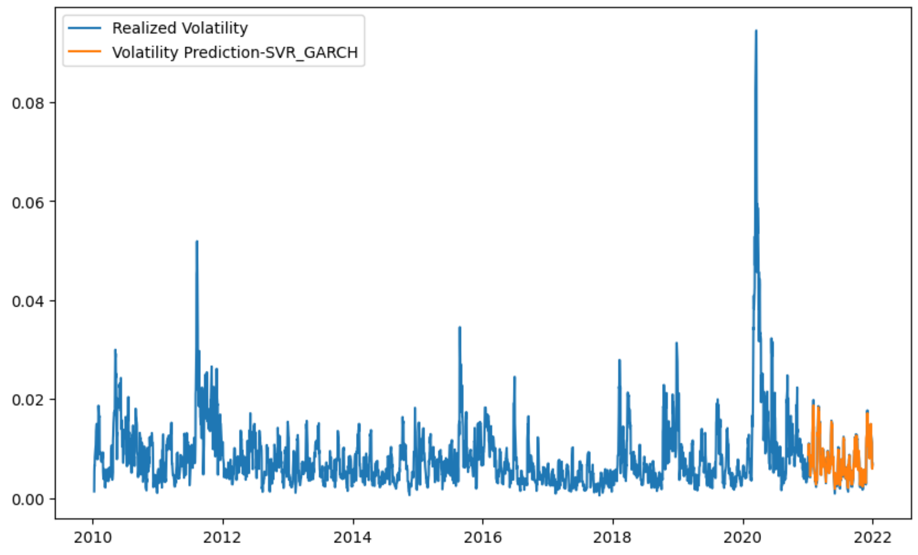

Figure 8: Volatility Prediction using SVR-GARCH (RBF)

5. RESULTS

In this section, we will compare the RMSE values of ARCH, GARCH, EGARCH, GJR-GARCH, and SVR-GARCH. The RMSE measures the average between the predicted values and the actual values. Lower the RMSE, the better the model's forecasting ability.

Table 1: RMSE values of volatility models

Model	RMSE
ARCH	0.0814
GARCH	0.0784
EGARCH	0.0816
GJR-GARCH	0.0779

From the above table, the best performing model is GJR-GARCH, while the worst performing model is EGARCH. The difference in the performance of EGARCH and GJR-GARCH could be due to asymmetry in the market. Otherwise, there is no big difference in the performance of these models. The values of these models are still high knowing that these models are used in making important decisions like those concerned with risk management and portfolio optimization. We shall now study the forecasting ability of Support Vector Regression with the GARCH model.

Table 2: RMSE values of SVR-GARCH with different kernel functions

Model	RMSE
Linear	0.000481
Polynomial	0.002674
RBF	0.000378

From the RMSE values mentioned in Table 2, it is evident that SVR-GARCH with Radial Basis Function (RBF) outperforms SVR-GARCH with linear and Polynomial kernels, and Polynomial Kernel is the worst performing kernel with the highest RMSE (0.002674).

6. CONCLUSION

Measuring and modeling volatility is vital in making investment decisions, trading strategies , and also affects the risk management process. Thus , this study attempts to model the volatility of S&P 500 data over the period of twelve years from 01/01/2010 to 01/01/2022 using ARCH, GARCH, EGARCH, GJR-GARCH and SVR-GARCH. And also study the aforementioned models' forecasting ability by conducting a comparative study of their RMSE values.

Though the traditional volatility models gave satisfactory results, it is important to work towards improving the accuracy further. On the same note, we further investigate the possibility of deploying Machine Learning algorithm, Support Vector Regression in specific, along with GARCH model for better modeling. The RMSE values in Table 2 prove that SVR-GARCH outperforms all the traditional models, with a value as low as 0.000378 observed in the case of SVR-GARCH (RBF).

In conclusion, this study lays emphasis on the forecasting ability of traditional volatility models, and the importance of application of Machine Learning Algorithm to further increase the performance of the aforementioned models.