INDIANA UNIVERSITY
BLOOMINGTON

Stock Prediction of Deutsche Börse using AWS

Final project submitted as a part of ECON-M24-Financial Econometrics, M.S. Data Science at Indiana University, Bloomington

Under the guidance of:

Professor Ke-li Xu

Department of Economics

Team 9 - Member Names:

- **Meet Palod** - Preprocessing, EDA, Glue Job/Crawler Creation, QuickSight Visualization
- **Aditya Mhaske** - Data Wrangling, Time Series Analysis, ML model

TABLE OF CONTENTS

1. INTRODUCTION:

The Deutsche Börse Public Eurex Data Set consists of real-time trade data aggregated at one-minute intervals from the Eurex trading systems. It provides the initial, lowest, highest, final price, and volume for every minute of the trading day and tradable security. The contents of the Deutsche Börse Public Dataset, from the Eurex trading engines, are defined in the data dictionary. This Eurex dataset contains trade data relating to derivative security trades. Each row represents one minute of trade activity for each security, following the Open/High/Low/Close (OHLC) format, with the number of trades and traded contracts. The EMR Cluster has been used with a PySpark instance. PySpark on the EMR cluster has been used to propose and develop a model, and this model has been trained on the large dataset. AWS Sagemaker has been used to merge files and carry out exploratory data analysis; it has also been used to perform time series analysis of data. AWS Glue has been used to store the data into the database to connect to AWS QuickSight, this is used to visualize and explore the data, and create an informative data analytics dashboard.

2. DATASET DESCRIPTION:

The dataset is from the AWS Registry. It contains 45.5 million data points with 3230161 rows and 20 columns. The Deutsche Börse Public Data Set consists of real-time trade data aggregated at one-minute intervals from the Eurex trading systems. It provides the initial price, lowest price, highest price, final price, volume for every minute of the trading day, and tradable

security. The contents of the Deutsche Börse Public Dataset, from Eurex trading engines, are defined in the data dictionary. The EUREX dataset contains trade data relating to derivative security trades. Each row represents one minute of trade activity for each security, following the Open/High/Low/Close (OHLC) format, with the number of trades and traded contracts. After cleaning data, the number of columns and rows (3230161, 20), respectively. While using the Linear regression model we have chosen EUR currency data for prediction and modeling which is in data range[date_from = "2021-01-01" date_to = "2022-01-01"]. The EUR data consists of 87409 Data Points with which we are predicting the Start Price of EUR entries of the Deutsche Borse data; based on the current values of Start Price, the future values are predicted based on the best fitting ARCH and GARCH prediction models.

3. PROPOSED ARCHITECTURE :

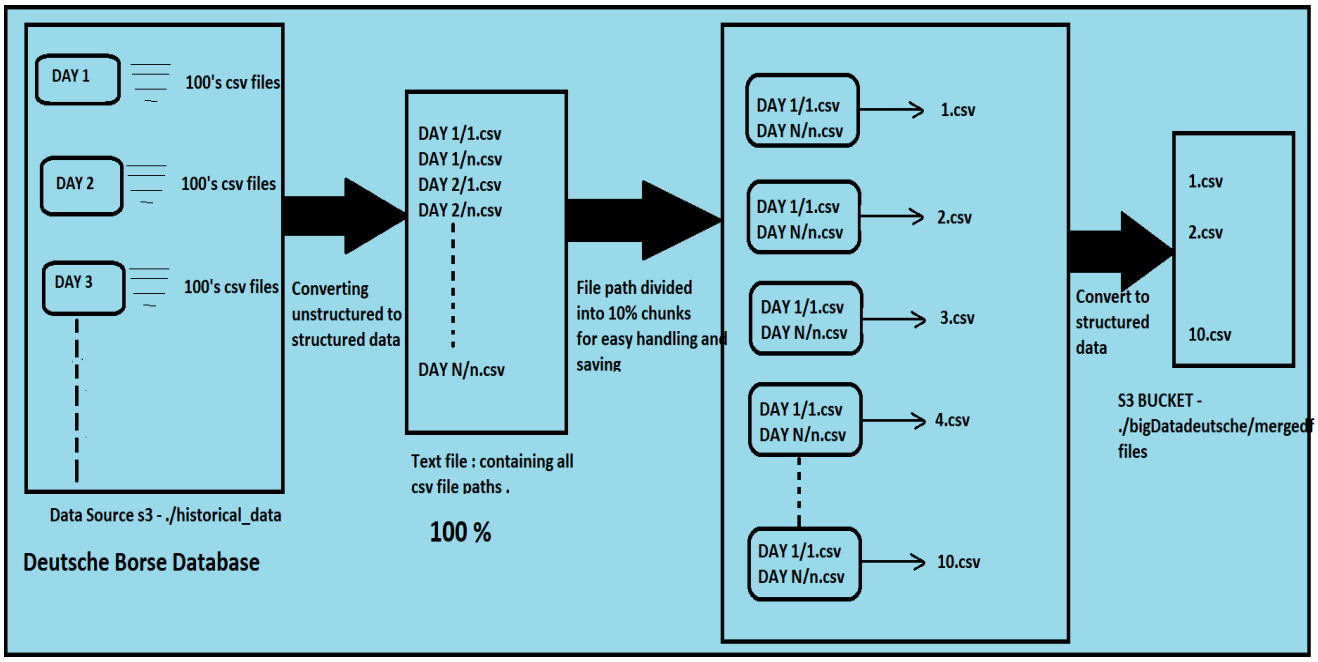


Fig 1.1

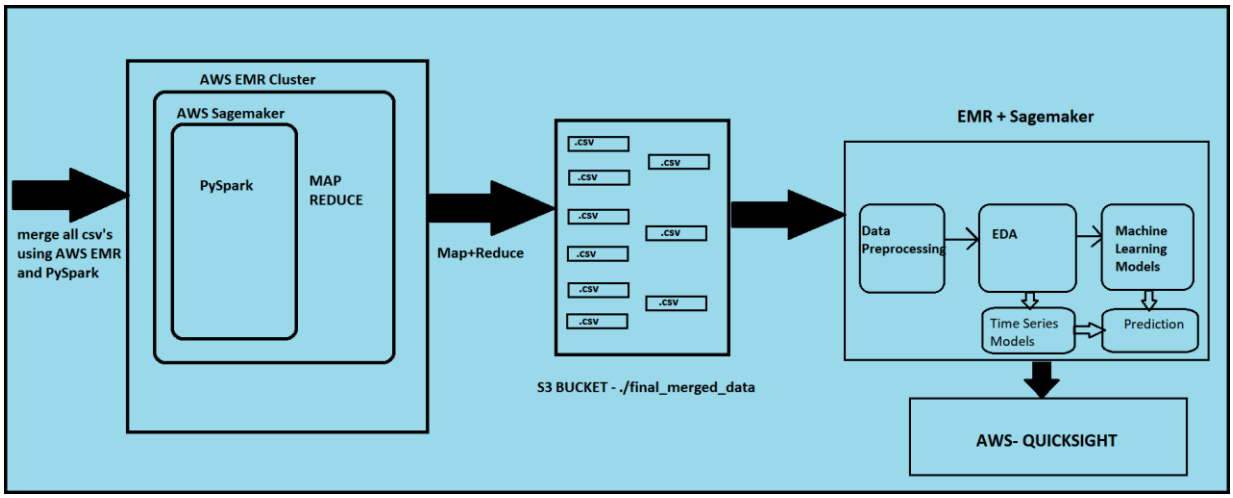


Fig 1.2

Deutsche Borse maintains the stock data in the S3 bucket of the ./historical_data folder. It had multiple folders day-wise which then had hundreds of CSV files. It was hard to read each file and upload it to our S3 bucket. So we came up with the solution of storing all file paths in the text file and then using it to iterate the csvs rather than accessing the S3 bucket. Further, we divided the file paths into batches of size ten and merged all the CSVs. Therefore at the end of processing, we are left with ten CSV files stored in /merged-files S3 bucket. The data from S3 bucket is transformed into AWS EMR, and Pyspark is used to apply the map further and reduce and perform data preprocessing. We can observe that PySpark divided the ten CSV files into multiple files to load across various virtual machines for ease of storage. PySpark has been used by EDA to understand the features and concepts like correlation and time series plot and check how each stock varies for EndPrice, StartPrice, StrikePrice, etc. Machine Learning models and time series models have been used to predict the dataset features like EndPrice and StartPrice and metric evaluation. We have used AWS QuickSight to show visualization and understand the features.

4. ECOSYSTEM :

AWS S3 :

Amazon Simple Storage Service, also called an S3 bucket, is widely used across the project as it is object storage that provides features like data availability, performance, and security. We

created user roles and users to access the S3 bucket using IAM of AWS. The policy was modified to be accessed by the team members only, and individually various preprocessing analyses on the dataset could be performed in Sage maker. S3 bucket also is an object storage service, where each object is stored as a file along with the metadata information. This object storage format is given an object ID, so if used along with Spark, it can incorporate fault-tolerant features and make the data more reliable to work with the loaded dataset. Before preprocessing large amounts of CSVs stored in a folder in the s3 bucket. Further, Py-Spark was used in AWS EMR to load the dataset from the s3 bucket and apply various methodologies of Preprocessing, which will be discussed further in the report.

AWS EMR :

Amazon EMR, also called Elastic MapReduce, is a service for cluster management for applying and programming big data frameworks using tools like Spark on AWS. EMR was chosen for big data processing and analysis as it gives direct access to the Hadoop environment and through which we can create a notebook and write PySpark code. However, EMR also allows writing SQL queries using HIVE and query S3 directly, another way to analyze. However, it was slower than the PySpark data frame, so we adopted this methodology. The project requires features for using clusters and auto-scaling during task execution as the dataset is enormous, so EMR is the best option. However, few team members used Databricks which uses a cluster-based approach and runs on EC2 instances. However, AWS Spark is faster than Databricks, and a convenient way of usage and accessing the s3 bucket was more straightforward, which made us use this mostly.

AWS Sagemaker:

AWS Sage maker is a notebook provided by AWS for trying Machine learning code and analyzing the dataset using PySpark. Sage maker helps connect, create, and manage EMR clusters from the studio. As the Sage maker is integrated within the EMR cluster, it helped us do data cleaning of the tera-byte scale.

AWS Glue:

Aws Glue is a service that allows for data to be extracted, transformed, and loaded (ETL). AWS Glue allows for a fast and easy way to organize, cleanse, validate, and format data and ultimately store this data in databases in Athena that can be queried. Some benefits of Glue include that it offers the ability to run jobs in real-time with Apache spark, it offers serverless computing, and it is effortless to set up. It can be scheduled to run whenever it is needed. Some specific services within AWS Glue include jobs and crawlers. Glue ETL jobs are great for pulling large amounts of data and easy preprocessing. Glue crawlers work on updating the Glue catalog databases.

5. EDA: Data Extraction, Data Wrangling, Preprocessing

The data set used from the S3 bucket of Deutsche Borse contains features which are both Categorical and Numeric in nature. Performed various preprocessing techniques like removing duplicate data points, dropping columns that are not useful for analyzing, converting categorical into numerical with one hot encoding and so on. The dataset has all the fields as a string when read from the s3 bucket and we converted it to respective datatypes , ie. took Numerical data such as Strike Price, Start Price, Max Price, Min Price, and End Price and converted them from strings to float to perform mathematical calculations. Filtered the data frame to date format using the filter function. The remaining values were put in a data frame. Label Encoder is generally used to normalize labels and transform them to non-numerical labels; we used fit_transform() to fit label encoder and return encoded labels. Followed by performing Correlation between the attributes and calculating Correlation coefficients used to measure the strength of the relationship between two variables. Used Heatmap to visualize the correlation between in, 1.0 being the highest correlation and -1 being the least correlated value. Plotted Bar Graph between the highest correlation values (Put or Call and Contract Generation Number) & plotted histogram between Average and StrikePrice of each StartPrice. The correlation between Start Price and Contract Generation Number is 0.25. A correlation 0.50 correlation was found between Start Price and Contract Generation Number.

Fig 5.1

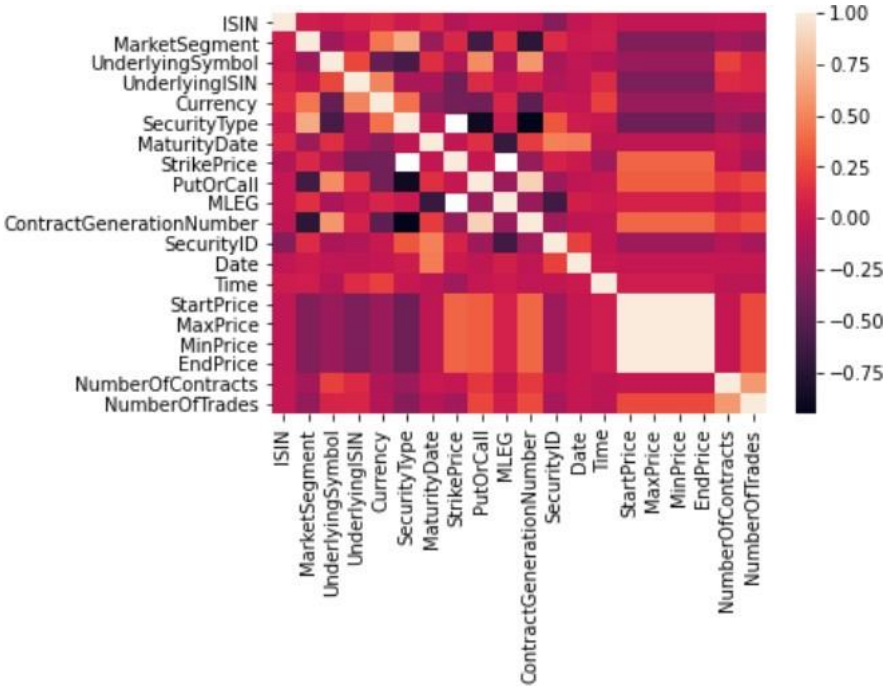


Fig 5.2

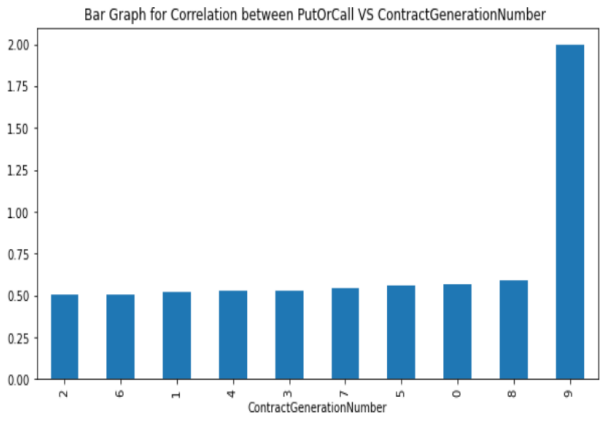


Fig 5.3

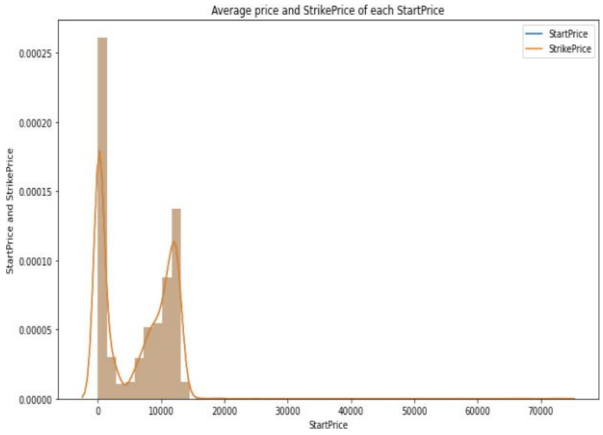
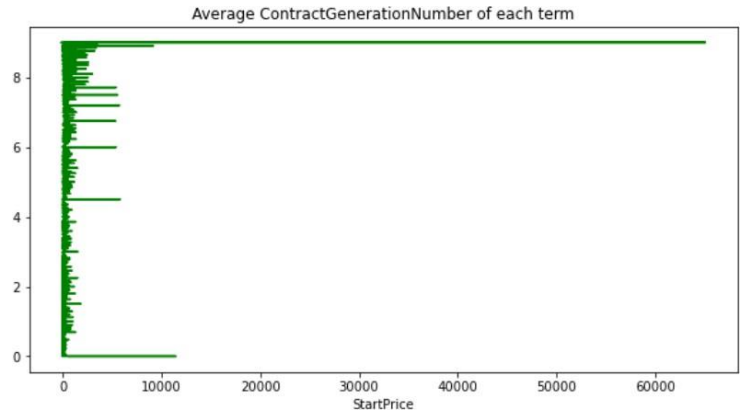


Fig 5.4



5. GLUE JOB SETUP:

In order to be able to deal with a large amount of data in the s3 Deutsche Borse bucket, AWS Glue Crawlers and Jobs were created to send all the files from the s3 bucket to a Glue database ultimately cleaning the code so data so that it could be queried. The first crawler that was created was made to send the data that was pulled from the s3 bucket and send it to a database so that the data could be easily queried when needed. In order to fully set up this crawler, it was essential to configure IAM roles to read from/write to the s3 bucket. The crawler contained the locations of the CSV files, glue database name, and a table that would map all the values. When observing the newly created table, it was evident that there was a problem with formatting the columns, and the query took an extended amount of time. In order to rectify this, a Glue ETL job was created to transform the CSV files to parquet. Besides the advantage that parquet files provide a better query performance, they also take up much less storage than CSV files. The

Glue ETL job contained configurations to map the columns to their names instead of labeled as col0, col1, etc. These new parquet files were written into a new folder where all the parquet files were stored. Finally, the last crawler was created to send the newly created/formatted parquet files into a new table. This data was then used for creating the visualizations using AWS Quicksight, shown later in this report.

6. MODEL SELECTION, PREDICTION AND PERFORMANCE :

We have chosen a Machine Learning Linear Regression model to predict the 'End Price' field and a Time Series ARCH model to predict the 'Start Price' field of the Deutche Borse Eurex dataset.

6.1 Machine Learning Modeling:

Linear Regression

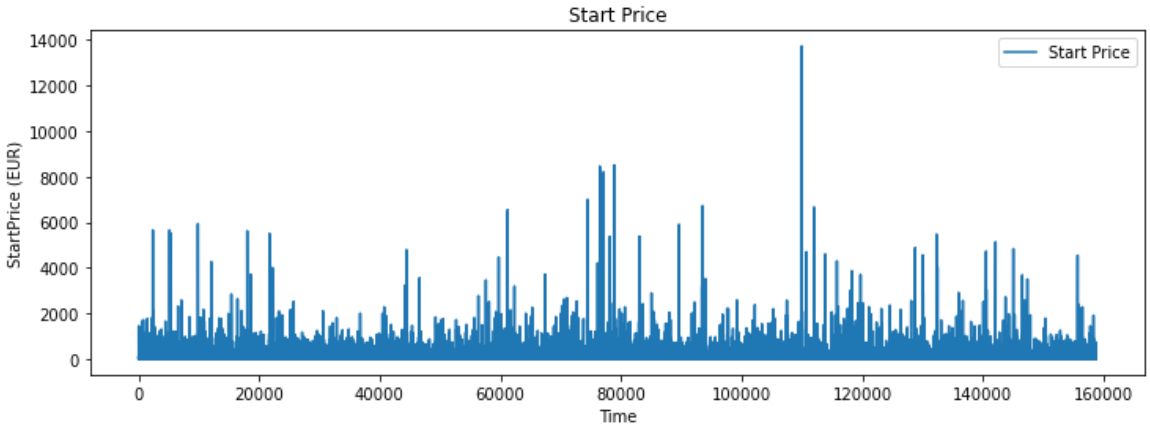
Linear Regression is the process of estimating the optimal line to estimate continuous target values by using the feature set. We used MlLib tool to train the linear regression model to predict the end price of the stock given the other set of features in PySpark. We fed the model with transformed features. We used String Indexer to index the categorical features into numerical values and transformed using encoder. We used a vector assembler to create the final feature set from categorical and numerical features such as 'SecurityType_index', 'StartPrice', 'MaxPrice', 'MinPrice', 'StrikePrice'.

6.2 Time Series Modeling:

ARCH and GARCH Models

ARCH and GARCH models are used in time series analysis for stock prediction due to their ability to handle data volatility. The model compensates for the periods with high volatility by assigning lower weights while modeling. The higher peaks are assigned the lowest values during assignment; thus, the variance prediction model will display a close prediction of the actual values. Since the values in this model are based on values just before prediction, it does not depend much on extremely old data. It assigns lower weights to them during prediction to keep the predicted values current. In the analysis of Deutsche Borse data, several models have been used. The analysis is performed on only the Euro currency transactions so that the uniformity of the prediction is maintained where the target field used is the Start Price field. As per the graph (Fig 6.1) below, the data is volatile. The time series models predict the Start Price of the data for 6 hours based on the current values.

Fig 6.1



Following are the steps performed during the time series modeling:

- The above stock data is made stationary by BoxCox transformations so that the data follows a Gaussian distribution around zero means. For ARCH and GARCH modeling, the current data is subtracted from the previous data and converted to a percentage to determine the percentage difference. The absolute value of the values is taken to ensure better variance prediction.
- The Dickey-Fuller test confirms that the data is now stationary and does not require further transformations. The Dickey-Fuller test displayed a p-value below 0.05, which means the data has become stationary.
- Autocorrelation and Partial-autocorrelation plots are plotted to identify the models; there are no visibly significant correlations between the data points. Thus, the remaining steps are performed.
- GARCH(2,0), also called the ARCH(2) model, is fitted with 87 thousand records; the summary displays that the resulting variance predictions are plotted against the actual values. The ARCH(3) model is also fitted and predicted against the data, similar to the ARCH(2) model. The resulting predictions closely match the valid values, and the model identifies the periods of high and low variances. In the graph (Fig 6.2 and Fig 6.3) below, the Predicted Volatility is plotted against actual values. The pattern generated by the Conditional Variance is seen rising and falling based on the spikes in the data. Comparing the values of mean absolute error: 1487 for ARCH(2) and 1496 for ARCH(3), we choose the ARCH(2) model based on the most negligible value of the mean absolute error.

Fig 6.2

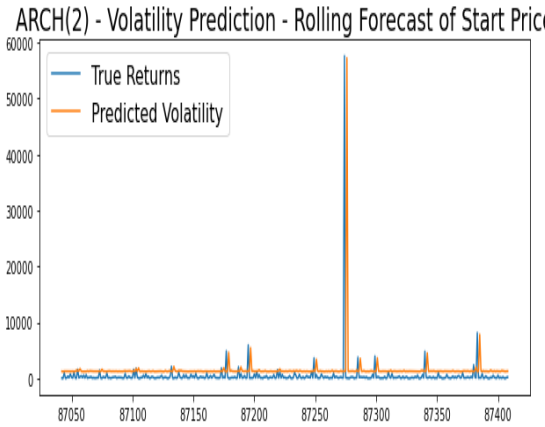
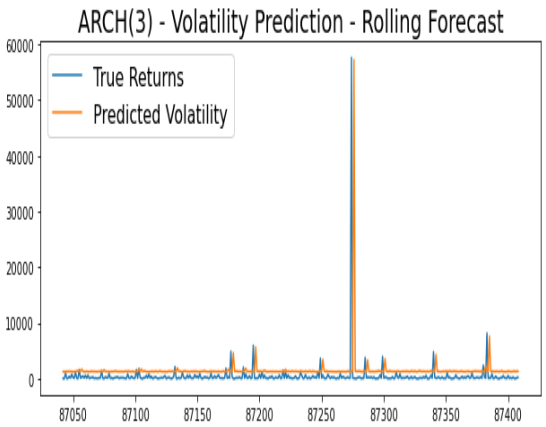


Fig 6.3



- In order to cross-validate, other models are GARCH(1,0), GARCH(1,1), GARCH(2,2), and GARCH(3,3). We compare the generated Volatility Prediction of these models in the graphs below. We observe that the Variance does not match the True values. Thus, the below models are rejected.

Fig 6.4

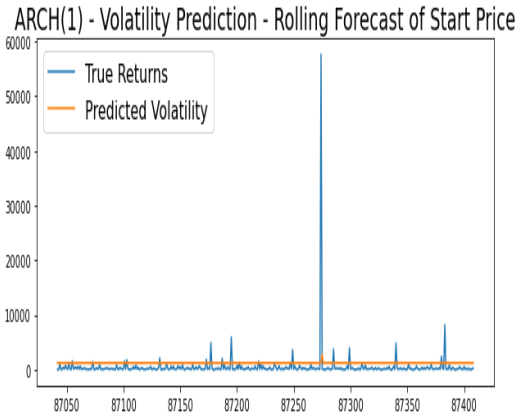


Fig 6.5

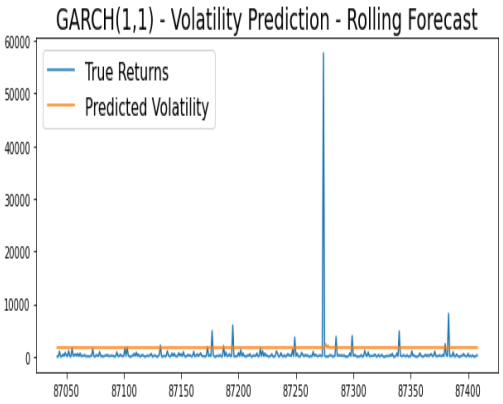


Fig 6.6

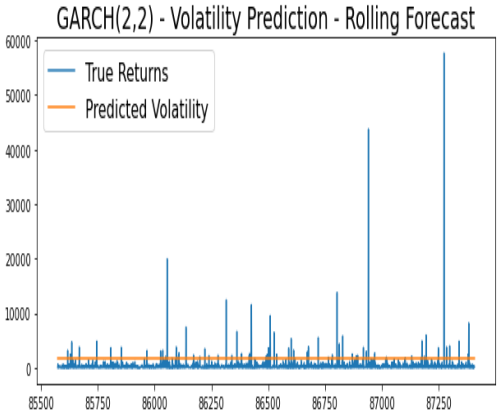
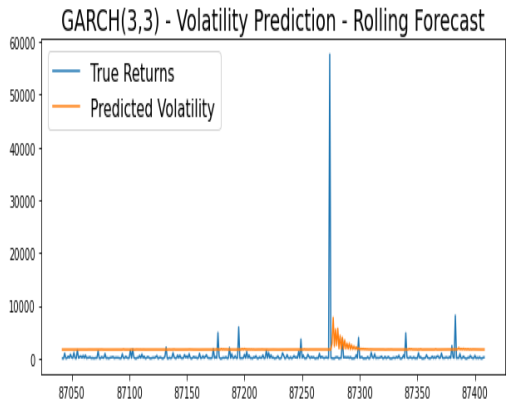


Fig 6.7



We can conclude that the dataset uses ARCH model due to its sudden spikes and volatile pattern that is visible, a GARCH model would be used if the variation remained for a while and then returned to normal, whereas, an ARCH model will notice a sudden jump in the values which return to normal as soon as they are observed. Thus, the ARCH(2) model chosen for this dataset is the best fit for this type of data.

6.3 Performance Observations:

Machine Learning Performance

The size of the data has a tremendous effect on the performance depending on whether we perform parallel or non-parallel processing. The performance of the model significantly increased with the increase in number of data points while the time taken to train the model has increased drastically.

Time Series Performance

Based on the number of records the performance and accuracy vary for the Time series models. Following report illustrates this variation:

Time taken to execute GARCH(2,0) predictions using

- 87409 records is 0:01:36.823132 h:mm:ss
- 60000 records is 0:01:40.172163 h:mm:ss
- 40000 records is 0:01:24.499844 h:mm:ss
- 20000 records is 0:01:27.221510 h:mm:ss
- 100 records is 0:00:25.292846 h:mm:ss

We observe that the execution time is around 2 to 3 minutes for a large dataset, and the execution time is around 30 seconds for a smaller dataset of size 100.

For a GARCH model, the observations prior to the current observation are of utmost importance; therefore, the resulting predictions have little to no bearing in either of the variance prediction graphs of 87409 to 100 records. The number of observations must be above 30 for the normal distribution of the data.

Explained Variance Score, MAE, MSE, RMSE, and R_Squared Scores for each of the different types of models are:

```
Results of sklearn.metrics ARCH(1):
Explained Variance Score is  -0.9373423706916968
MAE: 1496.7449893239113
MSE: 19786792.83359942
RMSE: 4448.234799737916
R-Squared: -1.0434978714910104
```

```
Results of sklearn.metrics ARCH(2):
Explained Variance Score is  -0.9406313835411455
MAE: 1487.6571436550553
MSE: 19802049.971468467
RMSE: 4449.94943470917
R-Squared: -1.045073565390605
```

```
Results of sklearn.metrics ARCH(3):
Explained Variance Score is  -0.9373423706916968
MAE: 1496.7449893239113
MSE: 19786792.83359942
RMSE: 4448.234799737916
R-Squared: -1.0434978714910104
```

```
Results of sklearn.metrics GARCH(1,1):
Explained Variance Score is  -0.0005924616559711549
MAE: 1669.8379155318603
MSE: 11182742.796948161
RMSE: 3344.060824349366
R-Squared: -0.1549072805872198
```

```
Results of sklearn.metrics GARCH(2,2):
Explained Variance Score is  2.220446049250313e-16
MAE: 1657.7410160223017
MSE: 11145946.135833029
RMSE: 3338.5544979576157
R-Squared: -0.15110707409094193
```

```
Results of sklearn.metrics GARCH(3,3):
Explained Variance Score is  -0.026320513940859547
MAE: 1717.2343980863147
MSE: 11578573.152665757
RMSE: 3402.7302497649966
R-Squared: -0.19578699748640838
```

However, the Explained Variance Score, MAE, MSE, RMSE, and R_Squared Scores for all the large datasets has remained almost the same, whereas, the same scores for the small dataset of 100 records shows lower MAE, MSE and RMSE scores due to lower overall variance displayed in the concentrated part of the dataset i.e. the last 100 values of the data:

```
Explained Variance Score is  -0.6422750321805772
Results of sklearn.metrics:
MAE: 1271.096747853803
MSE: 2597558.549993291
RMSE: 1611.694310343401
R-Squared: -1.8045182472314218
```

Visually, the change in the dataset size has little to no bearing on the prediction of the time series ARCH and GARCH models, whereas, the execution time is greatly affected due to big data.

7. VISUALIZATION DASHBOARD- AWS QUICKSIGHT :

Aws quicksight is a tool that allows for easy visualization of data. It provides the opportunity to plot different variables within the dataset against each other and observe any present trends. The databases created via AWS Glue earlier are crucial and are used here. We configured AWS Quicksight to connect to and read the data directly from the tables within these databases.

Fig 7.1

The graph on the top shows the average strike and end prices for the stock “DAX” over a day. It can be observed that the strike price is higher than the end price at all times. It also helps to confirm that our data is correct as there are no values for strike price outside of market hours.

The second and third graphs show how the dataset's data is split up by currency. As observed in the second graph, the most amount of data contained within the files are in Euros. The graph in the bottom right also shows the total money invested in stock compared to other forms of currency.

8. FUTURE SCOPE AND CONCLUSION

Several models have been used to predict stock data in the past, we have managed to predict the Deutche Borse data with great accuracy using a Time Series ARCH model for a time related data field. ARCH models have several applications in the finance industry.

We were faced with a lot of challenges due to the size of the dataset and invalid records in the data. The dataset demanded a tremendous amount of preprocessing and exploratory data analysis to clean the data and understand the pattern of each of the features. For instance, the EUR currency was filtered and used for modeling and predicting in time series. Due to Pyspark's limitations, an external visualization tool called AWS QuickSight was used to gain useful insights from the data. Due to this, several tools and applications were used for modeling and prediction. With the improvement in technology, the complete process can be managed in the single system so as to perform all the operations more efficiently. The models that are demonstrated in the paper can be used for stock prediction in finance related fields, they can also be used in other fields that demand prediction of volatile data.

9. REFERENCES

1. https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/
2. https://towardsdatascience.com/aws-glue-101-all-you-need-to-know-with-a-real-world-example-f34af17b782f
3. https://catalog.us-east-1.prod.workshops.aws/workshops/44c91c21-a6a4-4b56-bd95-56bd443aa449/en-US/lab-guide/visualize
4. https://medium.com/analytics-vidhya/arima-garch-forecasting-with-python-7a3f797de3ff
5. https://medium.com/vedity/python-data-preprocessing-using-pyspark-cc3f709c3c23
6. https://www.datasciencemadesimple.com/get-duplicate-rows-in-pyspark/
7. https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.regression.LinearRegression.html
8. https://towardsdatascience.com/csv-files-for-storage-no-thanks-theres-a-better-option-72c78a414d1d#:~:text=Parquet%20files%20take%20much%20less,to%20store%20in%20Parquet%20format!&text=Once%20again%20%E2%80%94%20yikes!,file%20is%201TB%20in%20size.
9. https://spark.apache.org/docs/2.1.0/ml-tuning.html

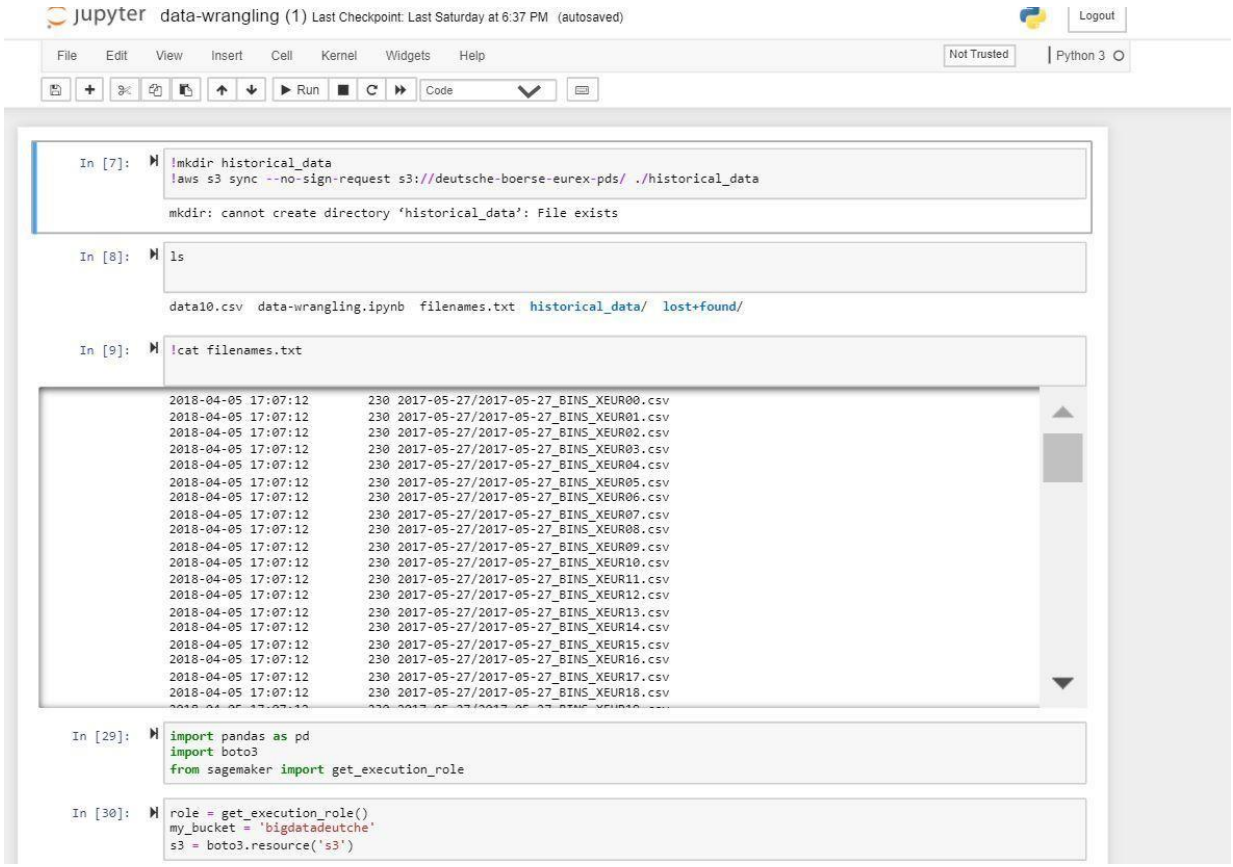
10. APPENDIX

The appendix consists of images of all the code. Firstly we Imported all the files from the S3 bucket, concatenate all the data into ten files and merged it into a single file and placed it in the S3 bucket and performed Pre-processing in Pyspark(EMR), and used ML models and Time Series Models for Prediction in Python and visualized through Quicksight for getting valuable insights.

The appendix has

1.) Data Wrangling Code
2.) Pre-processing (Data cleaning , Heat Map for correlation, Plots for Correlated Attributes)
3.) Glue job
4.) Machine Learning Model code
5.) Time Series Analysis code
6.) Performance Metrics.

Data Wrangling Code



```
2018-04-05 17:07:12        230 2017-05-27/2017-05-27_BINS_XEUR09.csv
2018-04-05 17:07:12        230 2017-05-27/2017-05-27_BINS_XEUR10.csv
2018-04-05 17:07:12        230 2017-05-27/2017-05-27_BINS_XEUR11.csv
2018-04-05 17:07:12        230 2017-05-27/2017-05-27_BINS_XEUR12.csv
2018-04-05 17:07:12        230 2017-05-27/2017-05-27_BINS_XEUR13.csv
2018-04-05 17:07:12        230 2017-05-27/2017-05-27_BINS_XEUR14.csv
2018-04-05 17:07:12        230 2017-05-27/2017-05-27_BINS_XEUR15.csv
2018-04-05 17:07:12        230 2017-05-27/2017-05-27_BINS_XEUR16.csv
2018-04-05 17:07:12        230 2017-05-27/2017-05-27_BINS_XEUR17.csv
2018-04-05 17:07:12        230 2017-05-27/2017-05-27_BINS_XEUR18.csv
```

In [29]:
```
import pandas as pd
import boto3
from sagemaker import get_execution_role
```

In [30]:
```
role = get_execution_role()
my_bucket = 'bigdatadeutche'
s3 = boto3.resource('s3')
```

In [32]:
```
data = pd.read_csv("s3://bigdatadeutche/mergedfiles/*.csv")
data.head()
```

Out[32]:

	ISIN	MarketSegment	UnderlyingSymbol	UnderlyingISIN	Currency	SecurityType	MaturityDate	StrikePrice	PutOrCall	MLEG	ContractGener
0	DE000C0BLY22	FDAX	DAX	DE0008469008	EUR	FUT	20170616.0	NaN	NaN	NaN	
1	DE000C0BLY48	FDXM	DAX	DE0008469008	EUR	FUT	20170616.0	NaN	NaN	NaN	
2	DE000C0BLYQ7	FESX	SX5E	EU0009658145	EUR	FUT	20170616.0	NaN	NaN	NaN	
3	DE000C0BGRF4	FBTP	NaN	NaN	EUR	FUT	20170608.0	NaN	NaN	NaN	
4	DE000C0BGRH0	FGBL	NaN	NaN	EUR	FUT	20170608.0	NaN	NaN	NaN	

In [22]:
```
data.shape
```

Out[22]: (3230161, 20)

In [23]:
```
!ls
```

```
data10.csv  data-wrangling.ipynb  filenames.txt  historical_data  lost+found
```

In [25]:
```
data_list = []
with open('filenames.txt') as f:
    lines = f.readlines()

    data_list = [line.split(" ")[-1] for line in lines]
```

In [ ]:
```
len(data_list)
```

In [ ]:
```
import pandas as pd
data_1 = [pd.read_csv("historical_data/"+file.replace("\n","")) for file in data_list[:int(0.1*len(data_list))]]
```

In [ ]:
```
data_1 = pd.concat(data_1)
data_1.to_csv("data1.csv",index=False)
```

In [ ]:
```
data_1 = [pd.read_csv("historical_data/"+file.replace("\n","")) for file in data_list[int(0.1*len(data_list)):int(0.2*len(da
data_1 = pd.concat(data_1)
data_1.to_csv("data2.csv",index=False)
```

In [ ]:
```
data_1 = [pd.read_csv("historical_data/"+file.replace("\n","")) for file in data_list[int(0.2*len(data_list)):int(0.3*len(da
data_1 = pd.concat(data_1)
data_1.to_csv("data3.csv",index=False)
```

In [ ]:
```
import boto3

from sagemaker import get_execution_role

role = get_execution_role()

my_bucket = 'bigdatadeutche'

orig_file = 'data1.csv'
dest_file = 'mergedfiles/data1.csv'

s3 = boto3.resource('s3')
s3.Bucket(my_bucket).upload_file(orig_file, dest_file)
```

In [ ]:
```
orig_file = 'data2.csv'
dest_file = 'mergedfiles/data2.csv'
s3.Bucket(my_bucket).upload_file(orig_file, dest_file)
orig_file = 'data3.csv'
dest_file = 'mergedfiles/data3.csv'
s3.Bucket(my_bucket).upload_file(orig_file, dest_file)
```

In [ ]:
```
!aws s3 ls --no-sign-request s3//bigdatadeutche//
```

In [ ]:
```
data_1 = [pd.read_csv("historical_data/"+file.replace("\n","")) for file in data_list[int(0.3*len(data_list)):int(0.4*len(da
data_1 = pd.concat(data_1)
data_1.to_csv("data4.csv",index=False)
```

In [ ]:
```
data_1 = [pd.read_csv("historical_data/"+file.replace("\n","")) for file in data_list[int(0.4*len(data_list)):int(0.5*len(da
data_1 = pd.concat(data_1)
data_1.to_csv("data5.csv",index=False)
```

In [ ]:
```
import pandas as pd
data_1 = [pd.read_csv("historical_data/"+file.replace("\n","")) for file in data_list[int(0.5*len(data_list)):int(0.6*len(da
data_1 = pd.concat(data_1)
data_1.to_csv("data6.csv",index=False)
```

In [ ]:
```
data_1 = [pd.read_csv("historical_data/"+file.replace("\n","")) for file in data_list[int(0.6*len(data_list)):int(0.7*len(da
data_1 = pd.concat(data_1)
data_1.to_csv("data7.csv",index=False)
```

```
data_1.to_csv("data8.csv",index=False)
```

In [ ]:
```
import pandas as pd
data_1 = [pd.read_csv("historical_data/"+file.replace("\n","")) for file in data_list[int(0.8*len(data_list)):int(0.9*len(da
data_1 = pd.concat(data_1)
data_1.to_csv("data9.csv",index=False)
```

In [ ]:
```
data_1 = [pd.read_csv("historical_data/"+file.replace("\n","")) for file in data_list[int(0.9*len(data_list)):int(1.0*len(da
data_1 = pd.concat(data_1)
data_1.to_csv("data10.csv",index=False)
```

In [ ]:
```
orig_file = 'data4.csv'
dest_file = 'mergedfiles/data4.csv'
s3.Bucket(my_bucket).upload_file(orig_file, dest_file)
orig_file = 'data5.csv'
dest_file = 'mergedfiles/data5.csv'
s3.Bucket(my_bucket).upload_file(orig_file, dest_file)
```

In [ ]:
```
import boto3

from sagemaker import get_execution_role

role = get_execution_role()

my_bucket = 'bigdatadeutche'
s3 = boto3.resource('s3')


orig_file = 'data6.csv'
dest_file = 'mergedfiles/data6.csv'
s3.Bucket(my_bucket).upload_file(orig_file, dest_file)
```

In [ ]:
```
orig_file = 'data7.csv'
dest_file = 'mergedfiles/data7.csv'
s3.Bucket(my_bucket).upload_file(orig_file, dest_file)
```

In [ ]:
```
orig_file = 'data8.csv'
```

Data Pre-processing

In [1]:
```
import pyspark
from pyspark import SparkConf
from pyspark.sql import SparkSession
```

VBox()

Starting Spark application

ID	YARN Application ID	Kind	State	Spark UI	Driver log	User	Current session?
0	application_1651721918736_0001	pyspark	idle	Link	Link	None	✓

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

SparkSession available as 'spark'.

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

In [2]:
```
spark = SparkSession.builder.appName("BigData").getOrCreate()
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

In [3]:
```
data = spark.read.option("header",True).csv("s3n://big-data-bia678/final_merged_data/")
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

In [4]:
```
data.persist()
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

DataFrame[ISIN: string, MarketSegment: string, UnderlyingSymbol: string, UnderlyingISIN: string, Currency: string, Security Type: string, MaturityDate: string, StrikePrice: string, PutOrCall: string, MLEG: string, ContractGenerationNumber: string, SecurityID: string, Date: string, Time: string, StartPrice: string, MaxPrice: string, MinPrice: string, EndPrice: string, NumberOfContracts: string, NumberOfTrades: string]

In [5]:
```
data.cache()
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

DataFrame[ISIN: string, MarketSegment: string, UnderlyingSymbol: string, UnderlyingISIN: string, Currency: string, Security Type: string, MaturityDate: string, StrikePrice: string, PutOrCall: string, MLEG: string, ContractGenerationNumber: string, SecurityID: string, Date: string, Time: string, StartPrice: string, MaxPrice: string, MinPrice: string, EndPrice: string, NumberOfContracts: string, NumberOfTrades: string]

In [6]:

```
data = data.dropDuplicates()
data.show()
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

```
+------------+-------------+----------------+--------------+--------+------------+------------+----------+---------+------
--------------+-------------------------+----------+----------+-----+----------+--------+--------+--------+----------------
+-------------+
|        ISIN|MarketSegment|UnderlyingSymbol|UnderlyingISIN|Currency|SecurityType|MaturityDate|StrikePrice|PutOrCall|
MLEG|ContractGenerationNumber|SecurityID|      Date| Time|StartPrice|MaxPrice|MinPrice|EndPrice|NumberOfContracts|NumberOfT
rades|
+------------+-------------+----------------+--------------+--------+------------+------------+----------+---------+------
--------------+-------------------------+----------+----------+-----+----------+--------+--------+--------+----------------
+-------------+
|DE000C58X4H8|         FDXM|             DAX|  DE0008469008|     EUR|         FUT|  20211217.0|       null|     null| FDXM
SI 20211217 CS|                     null| 6209447.0|2021-11-09|10:20|   16077.0| 16078.0| 16076.0| 16077.0|             24.0
|          18.0|
|DE000C1W2M60|         ODX2|             DAX|  DE0008469008|     EUR|         OPT|  20211112.0|    16100.0|      Put|ODX2 S
I 20211112 ...|                      1.0| 6905146.0|2021-11-09|10:33|      76.5|    76.5|    76.5|    76.5|              1.0
|           1.0|
|DE000C58X4M8|         FESB|            SX7E|  EU0009658426|     EUR|         FUT|  20211217.0|       null|     null| FESB
SI 20211217 CS|                     null| 6209451.0|2021-11-09|10:36|    104.0|  104.05|   104.0|  104.05|             25.0
|           4.0|
|DE000C58X5Y0|         FSTE|            SXEP|  EU0009658780|     EUR|         FUT|  20211217.0|       null|     null| FSTE
SI 20211217 CS|                     null| 6209496.0|2021-11-09|10:58|    286.1|   286.1|   286.1|   286.1|              1.0
|           1.0|
|DE000C0PLE53|         OESX|            SX5E|  EU0009658145|     EUR|         OPT|  20211217.0|     4200.0|      Put|OESX S
I 20211217 ...|                      1.0| 2267371.0|2021-11-09|11:26|     36.8|    36.8|    36.8|    36.8|            201.0
|           2.0|
|DE000C58EEE5|         FBTP|            null|          null|     EUR|         FUT|  20211208.0|       null|     null| FBTP
SI 20211208 PS|                     null| 6183276.0|2021-11-09|12:31|   152.82|  152.83|  152.82|  152.82|             20.0
|          12.0|
|DE000C1WYZT5|          SZG|             SZG|  DE0006202005|     EUR|         OPT|  20220121.0|       34.0|     Call|SZG SI
20220121 P...|                     4.0| 6940551.0|2021-11-09|13:14|     1.49|    1.49|    1.49|    1.49|             5.0|
1.0|
|DE000C1W2BL4|         OGB2|            FGBL|  DE0009652644|     EUR|         OPT|  20211112.0|      170.5|      Put|OGB2 S
I 20211112 ...|                      1.0| 6906945.0|2021-11-09|13:22|     0.17|    0.17|    0.17|    0.17|              1.0
|           1.0|
|DE000C6CMZQ7|         OKS2|            FKS2|  XC0009664639|     KRW|         OPT|  20211111.0|      397.5|     Call|OKS2 S
I 20211111 ...|                      1.0| 6393198.0|2021-11-09|13:33|     0.29|     0.3|    0.29|     0.3|              3.0
|           3.0|
|DE000C6CMZF0|         OKS2|            FKS2|  XC0009664639|     KRW|         OPT|  20211111.0|      385.0|      Put|OKS2 S
I 20211111 ...|                      1.0| 6393189.0|2021-11-09|13:34|     0.68|    0.68|    0.68|    0.68|              2.0
|           2.0|
|DE000C6CMZQ7|         OKS2|            FKS2|  XC0009664639|     KRW|         OPT|  20211111.0|      397.5|     Call|OKS2 S
I 20211111 ...|                      1.0| 6393198.0|2021-11-09|14:05|      0.3|     0.3|    0.29|     0.3|              6.0
|           3.0|
|DE000C1W2FF7|         OES2|            SX5E|  EU0009658145|     EUR|         OPT|  20211112.0|     3975.0|      Put|OES2 S
I 20211112 ...|                      1.0| 6905288.0|2021-11-09|14:13|      0.3|     0.3|     0.3|     0.3|              5.0
|           1.0|
|DE000C6CVS93|          FVS|            V2TX|  DE000A0C3QF1|     EUR|         FUT|  20220119.0|       null|     null|  FVS
SI 20220119 CS|                     null| 6415656.0|2021-11-09|14:17|     21.4|    21.4|    21.4|    21.4|             19.0
|           1.0|
|DE000C59ESM6|          MRK|             MRK|  DE0006599905|     EUR|         OPT|  20220318.0|      100.0|      Put|MRK SI
20220318 P...|                     5.0| 6230829.0|2021-11-09|14:20|     0.32|    0.32|    0.32|    0.32|             10.0|
1.0|
|DE000C6CMZJ2|         OKS2|            FKS2|  XC0009664639|     KRW|         OPT|  20211111.0|      390.0|     Call|OKS2 S
I 20211111 ...|                      1.0| 6393192.0|2021-11-09|14:53|     2.51|    2.51|    2.45|    2.47|             45.0
|          26.0|
|DE000C4SA6R6|         FESX|            SX5E|  EU0009658145|     EUR|         FUT|  20211217.0|       null|     null| FESX
```

```
I 20211111 ...|                  1.0| 6393189.0|2021-11-09|13:34|     0.68|    0.68|    0.68|    0.68|              2.0
|           2.0|
|DE000C6CMZQ7|      OKS2|          FKS2|  XC0009664639|     KRW|         OPT|  20211111.0|     397.5|     Call|OKS2 S
I 20211111 ...|                  1.0| 6393198.0|2021-11-09|14:05|      0.3|     0.3|    0.29|     0.3|              6.0
|           3.0|
|DE000C1W2FF7|      OES2|          SX5E|  EU0009658145|     EUR|         OPT|  20211112.0|    3975.0|      Put|OES2 S
I 20211112 ...|                  1.0| 6905288.0|2021-11-09|14:13|      0.3|     0.3|     0.3|     0.3|              5.0
|           1.0|
|DE000C6CVS93|       FVS|           V2TX|  DE000A0C3QF1|     EUR|         FUT|  20220119.0|      null|     null|  FVS
SI 20220119 CS|                 null| 6415656.0|2021-11-09|14:17|     21.4|    21.4|    21.4|    21.4|             19.0
|           1.0|
|DE000C59ESM6|       MRK|           MRK|  DE0006599905|     EUR|         OPT|  20220318.0|     100.0|      Put|MRK SI
20220318 P...|                  5.0| 6230829.0|2021-11-09|14:20|     0.32|    0.32|    0.32|    0.32|             10.0|
1.0|
|DE000C6CMZJ2|      OKS2|          FKS2|  XC0009664639|     KRW|         OPT|  20211111.0|     390.0|     Call|OKS2 S
I 20211111 ...|                  1.0| 6393192.0|2021-11-09|14:53|     2.51|    2.51|    2.45|    2.47|             45.0
|          26.0|
|DE000C4SA6R6|      FESX|          SX5E|  EU0009658145|     EUR|         FUT|  20211217.0|      null|     null| FESX
SI 20211217 CS|                 null| 4631963.0|2021-11-09|15:00|   4337.0|  4341.5|  4336.0|  4341.5|           2964.0
|         285.0|
|DE000C6CMZ72|      OKS2|          FKS2|  XC0009664639|     KRW|         OPT|  20211111.0|     375.0|      Put|OKS2 S
I 20211111 ...|                  1.0| 6393181.0|2021-11-09|15:01|     0.07|    0.07|    0.06|    0.06|             43.0
|           7.0|
|DE000C1KBQ55|      NOA3|          NOA3|  FI0009000681|     EUR|         OPT|  20221216.0|       4.0|      Put|NOA3 S
I 20221216 ...|                  4.0| 2867104.0|2021-11-09|15:14|     0.31|    0.31|    0.31|    0.31|             88.0
|           2.0|
|DE000C1SMCK6|       AAR|           ABN|  NL0011540547|     EUR|         OPT|  20211119.0|      12.6|      Put|AAR SI
20211119 P...|                  1.0| 6745084.0|2021-11-09|15:31|     0.31|    0.31|    0.31|    0.31|           2103.0|
5.0|
|DE000C6CMZS3|      OKS2|          FKS2|  XC0009664639|     KRW|         OPT|  20211111.0|     400.0|     Call|OKS2 S
I 20211111 ...|                  1.0| 6393200.0|2021-11-09|15:34|     0.06|    0.06|    0.06|    0.06|              5.0
|           2.0|
+------------+----------+---------------+--------------+--------+------------+------------+----------+---------+------
--------------+------------------------+----------+----------+-----+----------+--------+--------+--------+----------------
+--------------+
only showing top 20 rows
```

In [7]:

```
from pyspark.sql.types import DoubleType
data = data.withColumn("StrikePrice", data["StrikePrice"].cast(DoubleType()))
data= data.withColumn("StartPrice", data["StartPrice"].cast(DoubleType()))
data = data.withColumn("MaxPrice", data["MaxPrice"].cast(DoubleType()))
data = data.withColumn("MinPrice", data["MinPrice"].cast(DoubleType()))
data = data.withColumn("EndPrice", data["EndPrice"].cast(DoubleType()))
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

In [8]:

```
numeric_columns = list()
categorical_column = list()
for col_ in data.columns:
    if data.select(col_).dtypes[0][1] != "string":
        numeric_columns.append(col_)
    else:
        categorical_column.append(col_)

print("Numeric columns",numeric_columns)
print("categorical columns",categorical_columns)
```

VBox()

In [8]:
```
numeric_columns = list()
categorical_column = list()
for col_ in data.columns:
    if data.select(col_).dtypes[0][1] != "string":
        numeric_columns.append(col_)
    else:
        categorical_column.append(col_)

print("Numeric columns",numeric_columns)
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

```
Numeric columns ['StrikePrice', 'StartPrice', 'MaxPrice', 'MinPrice', 'EndPrice']
categorical columns ['ISIN', 'MarketSegment', 'UnderlyingSymbol', 'UnderlyingISIN', 'Currency', 'SecurityType', 'MaturityDa
te', 'PutOrCall', 'MLEG', 'ContractGenerationNumber', 'SecurityID', 'Date', 'Time', 'NumberOfContracts', 'NumberOfTrades']
```

In [9]:
```
from pyspark.sql.functions import *
nan_rows = data.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in data.columns])
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

In [10]:
```
data = data.drop("UnderlyingISIN", "UnderlyingSymbol", "SecurityID")
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

In [11]:
```
data = data.dropna()
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

In [12]:
```
import pyspark.sql.functions as func

date_from = "2021-01-01"
date_to = "2022-01-01"
#data = data.select(func.to_date(data["Date"]).alias("time"))
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

In [13]:
```
sf.take(1)
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

```
[Row(ISIN='DE000C1W2M60', MarketSegment='ODX2', Currency='EUR', SecurityType='OPT', MaturityDate='20211112.0', StrikePrice=
16100.0, PutOrCall='Put', MLEG='ODX2 SI 20211112 CS EU P 16100 0', ContractGenerationNumber='1.0', Date='2021-11-09', Time
='10:33', StartPrice=76.5, MaxPrice=76.5, MinPrice=76.5, EndPrice=76.5, NumberOfContracts='1.0', NumberOfTrades='1.0')]
```

In [14]:
```
#Groupby put or call
sf.groupBy('PutOrCall').count().show()
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

```
+---------+-------+
|PutOrCall|  count|
+---------+-------+
|      Put|2876277|
|     Call|2256072|
+---------+-------+
```

In [16]:
```
#Groupby currency
sf.groupBy('Currency').count().show()
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

```
+--------+-------+
|Currency|  count|
+--------+-------+
|     CHF| 393867|
|     GBX|   1634|
|     USD|    823|
|     EUR|2824787|
|     KRW|1911232|
|     GBP|      6|
+--------+-------+
```

In [59]:
```
from pyspark.ml.feature import StringIndexer
indexer = StringIndexer(inputCol="SecurityType", outputCol="sec")
indexed = indexer.fit(sf).transform(sf)
indexed.head(5)
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

```
[Row(ISIN='DE000C1W2M60', MarketSegment='ODX2', Currency='EUR', SecurityType='OPT', MaturityDate='20211112.0', StrikePrice=
16100.0, PutOrCall='Put', MLEG='ODX2 SI 20211112 CS EU P 16100 0', ContractGenerationNumber='1.0', Date='2021-11-09', Time
='10:33', StartPrice=76.5, MaxPrice=76.5, MinPrice=76.5, EndPrice=76.5, NumberOfContracts='1.0', NumberOfTrades='1.0', sec=
0.0), Row(ISIN='DE000C0PLE53', MarketSegment='OESX', Currency='EUR', SecurityType='OPT', MaturityDate='20211217.0', StrikeP
rice=4200.0, PutOrCall='Put', MLEG='OESX SI 20211217 CS EU P 4200 0', ContractGenerationNumber='1.0', Date='2021-11-09', Ti
me='11:26', StartPrice=36.8, MaxPrice=36.8, MinPrice=36.8, EndPrice=36.8, NumberOfContracts='201.0', NumberOfTrades='2.0',
sec=0.0), Row(ISIN='DE000C1WYZT5', MarketSegment='SZG', Currency='EUR', SecurityType='OPT', MaturityDate='20220121.0', Stri
kePrice=34.0, PutOrCall='Call', MLEG='SZG SI 20220121 PS AM C 34.00 0', ContractGenerationNumber='4.0', Date='2021-11-09',
Time='13:14', StartPrice=1.49, MaxPrice=1.49, MinPrice=1.49, EndPrice=1.49, NumberOfContracts='5.0', NumberOfTrades='1.0',
sec=0.0), Row(ISIN='DE000C1W2BL4', MarketSegment='OGB2', Currency='EUR', SecurityType='OPT', MaturityDate='20211112.0', Str
ikePrice=170.5, PutOrCall='Put', MLEG='OGB2 SI 20211112 PS AM P 170.50 0', ContractGenerationNumber='1.0', Date='2021-11-0
9', Time='13:22', StartPrice=0.17, MaxPrice=0.17, MinPrice=0.17, EndPrice=0.17, NumberOfContracts='1.0', NumberOfTrades='1.
0', sec=0.0), Row(ISIN='DE000C6CMZQ7', MarketSegment='OKS2', Currency='KRW', SecurityType='OPT', MaturityDate='20211111.0',
StrikePrice=397.5, PutOrCall='Call', MLEG='OKS2 SI 20211111 CS EU C 397.5 0', ContractGenerationNumber='1.0', Date='2021-11
-09', Time='13:33', StartPrice=0.29, MaxPrice=0.3, MinPrice=0.29, EndPrice=0.3, NumberOfContracts='3.0', NumberOfTrades='3.
0', sec=0.0)]
```

In [176]:
```
#multiple column featurizer.
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer

indexers = [StringIndexer(inputCol=column, outputCol=column+"_index").fit(sf) for column in list(set(sf.columns)-set(['dat

pipeline = Pipeline(stages=indexers)
indexed = pipeline.fit(sf).transform(sf)
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

In [177]:
```
indexed.take(1)
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

In [177]:
```
indexed.take(1)
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

[Row(ISIN='DE000C1W2M60', MarketSegment='ODX2', Currency='EUR', SecurityType='OPT', MaturityDate='20211112.0', StrikePrice=16100.0, PutOrCall='Put', MLEG='ODX2 SI 20211112 CS EU P 16100 0', ContractGenerationNumber='1.0', Date='2021-11-09', Time='10:33', StartPrice=76.5, MaxPrice=76.5, MinPrice=76.5, EndPrice=76.5, NumberOfContracts='1.0', NumberOfTrades='1.0', PutOrCall_index=0.0, StrikePrice_index=182.0, ContractGenerationNumber_index=0.0, StartPrice_index=965.0, Currency_index=0.0, EndPrice_index=969.0, NumberOfContracts_index=5.0, ISIN_index=8026.0, Date_index=158.0, MLEG_index=7986.0, MarketSegment_index=19.0, MaxPrice_index=969.0, SecurityType_index=0.0, MaturityDate_index=63.0, NumberOfTrades_index=2.0, MinPrice_index=961.0, Time_index=201.0)]

In [72]:
```
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

In [178]:
```
sf.columns
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

['ISIN', 'MarketSegment', 'Currency', 'SecurityType', 'MaturityDate', 'StrikePrice', 'PutOrCall', 'MLEG', 'ContractGenerationNumber', 'Date', 'Time', 'StartPrice', 'MaxPrice', 'MinPrice', 'EndPrice', 'NumberOfContracts', 'NumberOfTrades']

In [179]:
```
assembler = VectorAssembler(
  inputCols=['SecurityType_index',
             #'MLEG_index',
             'StartPrice',
             'MaxPrice',
             'MinPrice',
             #'MarketSegment_index',
             'StrikePrice'
            ],
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

In [182]:
```
output = assembler.transform(indexed)
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

In [193]:
```
output.take(1)
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

[Row(ISIN='DE000C1W2M60', MarketSegment='ODX2', Currency='EUR', SecurityType='OPT', MaturityDate='20211112.0', StrikePrice=16100.0, PutOrCall='Put', MLEG='ODX2 SI 20211112 CS EU P 16100 0', ContractGenerationNumber='1.0', Date='2021-11-09', Time='10:33', StartPrice=76.5, MaxPrice=76.5, MinPrice=76.5, EndPrice=76.5, NumberOfContracts='1.0', NumberOfTrades='1.0', PutOrCall_index=0.0, StrikePrice_index=182.0, ContractGenerationNumber_index=0.0, StartPrice_index=965.0, Currency_index=0.0, EndPrice_index=969.0, NumberOfContracts_index=5.0, ISIN_index=8026.0, Date_index=158.0, MLEG_index=7986.0, MarketSegment_index=19.0, MaxPrice_index=969.0, SecurityType_index=0.0, MaturityDate_index=63.0, NumberOfTrades_index=2.0, MinPrice_index=961.0, Time_index=201.0, features=DenseVector([0.0, 76.5, 76.5, 76.5, 16100.0]))]

In [183]:
```
output.select("features", "EndPrice").show()
final_data = output.select("features", "EndPrice")
```

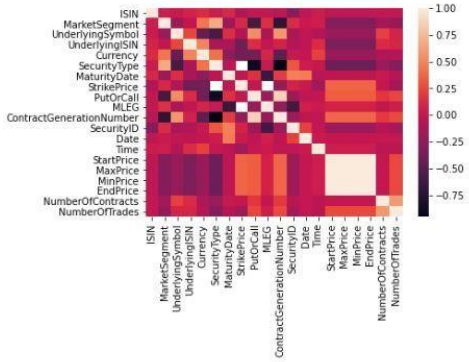
VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

```
+--------------------+--------+
|            features|EndPrice|
+--------------------+--------+
|[0.0,76.5,76.5,76...|    76.5|
|[0.0,36.8,36.8,36...|    36.8|
|[0.0,1.49,1.49,1....|    1.49|
|[0.0,0.17,0.17,0....|    0.17|
|[0.0,0.29,0.3,0.2...|     0.3|
|[0.0,0.68,0.68,0....|    0.68|
|[0.0,0.3,0.3,0.29...|     0.3|
|[0.0,0.3,0.3,0.3,...|     0.3|
|[0.0,0.32,0.32,0....|    0.32|
|[0.0,2.51,2.51,2....|    2.47|
|[0.0,0.07,0.07,0....|    0.06|
|[0.0,0.31,0.31,0....|    0.31|
|[0.0,0.31,0.31,0....|    0.31|
|[0.0,0.06,0.06,0....|    0.06|
|[0.0,0.45,0.46,0....|    0.45|
|[0.0,3.03,3.03,3....|    3.03|
|[0.0,0.86,0.87,0....|    0.87|
|[0.0,0.35,0.35,0....|    0.35|
|[0.0,1.46,1.46,1....|    1.46|
|[0.0,0.02,0.02,0....|    0.02|
```

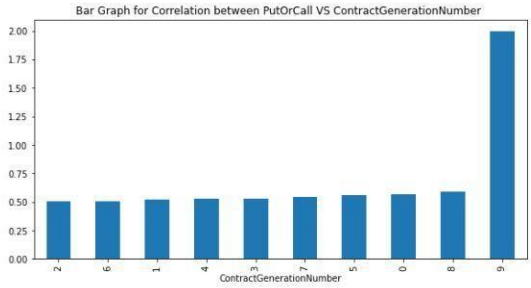
HEAT MAP FOR SHOWING CORRELATION BETWEEN THE ATTRIBUTES

In [50]:
```
import seaborn as sn
import matplotlib.pyplot as plt
#plt.figure(figsize=(10, 5), dpi=1000)
sn.heatmap(corr, annot = False)
plt.show()
```



PLOTTING THE CORRELATED ATTRIBUTES

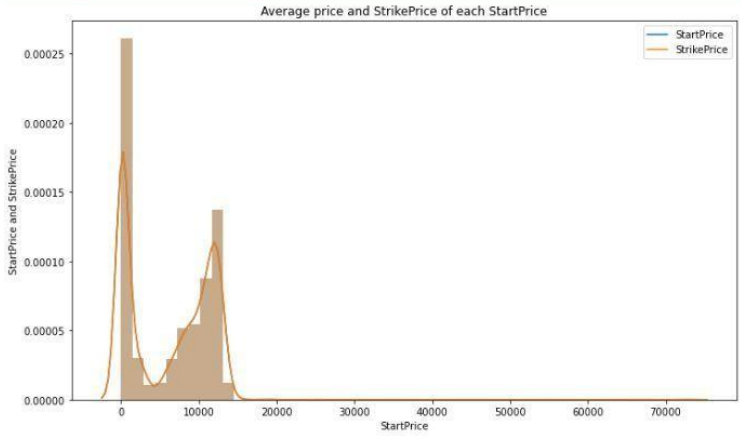
In [51]:
```
plt.figure(figsize=(10,5))
data.groupby('ContractGenerationNumber')['PutOrCall'].mean().sort_values().plot(kind='bar')
plt.title('Bar Graph for Correlation between PutOrCall VS ContractGenerationNumber ')
plt.show()
```



In [52]:

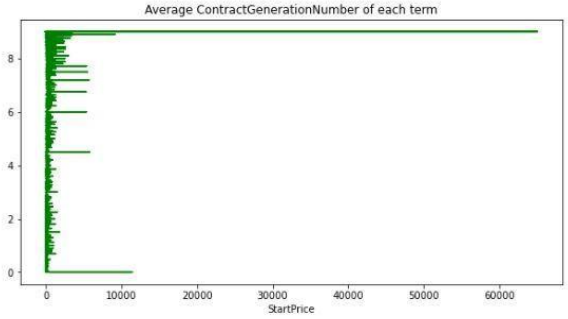
```
plt.figure(figsize=(12,7))
sn.distplot(data.groupby('StartPrice')['StrikePrice'].mean().sort_values())
sn.distplot(data.groupby('StartPrice')['StrikePrice'].mean().sort_values())
plt.title('Average price and StrikePrice of each StartPrice')
plt.xlabel("StartPrice")
plt.ylabel("StartPrice and StrikePrice")
plt.legend(['StartPrice', 'StrikePrice'])
plt.show()
```

```
/home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages/seaborn/distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
/home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages/seaborn/distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



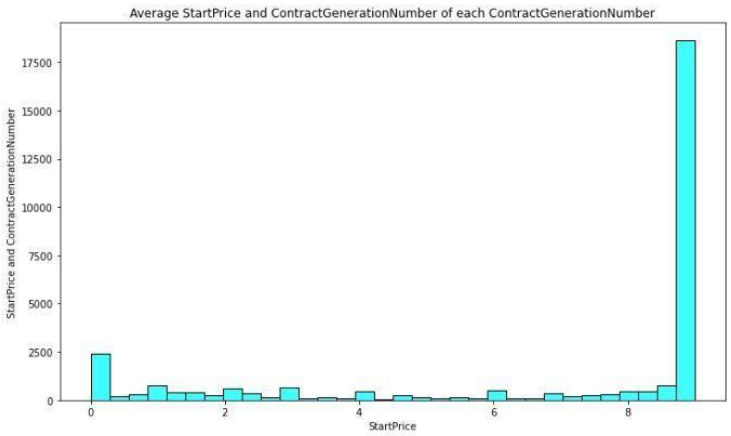
StartPrice

In [55]:
```
plt.figure(figsize=(10,5))
data.groupby('StartPrice')['ContractGenerationNumber'].mean().sort_values().plot( color='g')
plt.title('Average ContractGenerationNumber of each term')
plt.show()
```

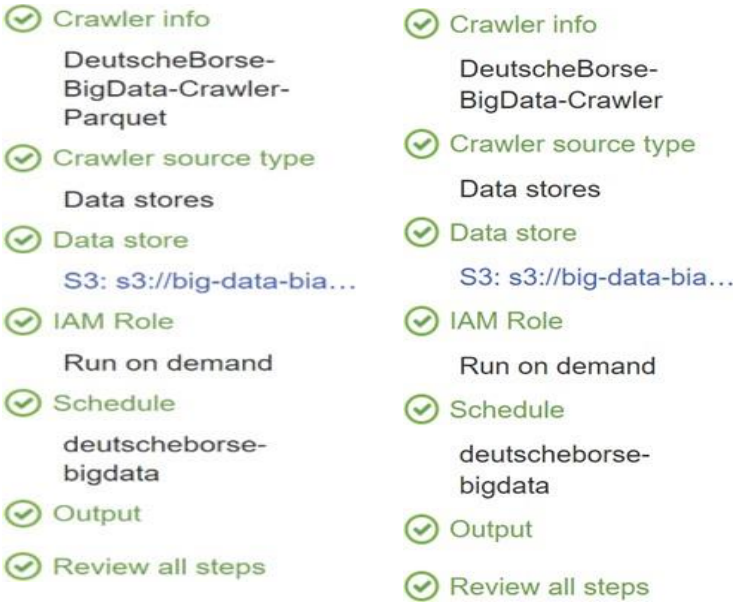


StartPrice

In [61]:
```
plt.figure(figsize=(12,7))
sn.histplot(data.groupby('StartPrice')['ContractGenerationNumber'].mean().sort_values(), color='cyan')
plt.title('Average StartPrice and ContractGenerationNumber of each ContractGenerationNumber')
plt.xlabel("StartPrice")
plt.ylabel("StartPrice and ContractGenerationNumber")
plt.show()
```



Glue Job Code




```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

## @params: [JOB_NAME]
args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)
## @type: DataSource
## @args: [database = "deutscheborse-bigdata", table_name = "big_data_bia678", transformation_ctx = "datasource0"]
## @return: datasource0
## @inputs: []
datasource0 = glueContext.create_dynamic_frame.from_catalog(database = "deutscheborse-bigdata", table_name = "big_data_bia678", transformation_ctx = "datasource0")
applymapping1 = ApplyMapping.apply(frame = datasource0, mappings = [("col0", "string", "ISIN", "string"),
("col1", "string", "MarketSegment", "string"), ("col2", "string", "UnderlyingSymbol", "string"),
("col3", "string", "UnderlyingISIN", "string"), ("col4", "string", "Currency", "string"),
("col5", "string", "SecurityType", "string"), ("col6", "double", "MaturityDate", "double"),
("col7", "double", "StrikePrice", "double"), ("col8", "string", "PutOrCall", "string"),
("col9", "string", "MLEG", "string"), ("col10", "double", "ContractGenerationNumber", "double"),
("col11", "string", "SecurityID", "string"), ("col12", "string", "Date", "string"),
("col13", "string", "Time", "string"), ("col14", "double", "StartPrice", "double"),
("col15", "double", "MaxPrice", "double"), ("col16", "double", "MinPrice", "double"),
("col17", "double", "EndPrice", "double"), ("col18", "double", "NumberOfContracts", "double"),
("col19", "double", "NumberOfTrades", "double"), ("partition_0", "string", "partition_0", "string")], transformation_ctx = "applymapping1")
## @type: ResolveChoice
## @args: [choice = "make_struct", transformation_ctx = "resolvechoice2"]
## @return: resolvechoice2
## @inputs: [frame = applymapping1]
resolvechoice2 = ResolveChoice.apply(frame = applymapping1, choice = "make_struct", transformation_ctx = "resolvechoice2")
## @type: DropNullFields
## @args: [transformation_ctx = "dropnullfields3"]
## @return: dropnullfields3
## @inputs: [frame = resolvechoice2]
dropnullfields3 = DropNullFields.apply(frame = resolvechoice2, transformation_ctx = "dropnullfields3")
## @type: DataSink
## @args: [connection_type = "s3", connection_options = {"path": "s3://big-data-bia678-1/final_parquet_files/"}, format = "parquet", transformation_ctx = "datasink4"]
## @return: datasink4
## @inputs: [frame = dropnullfields3]
datasink4 = glueContext.write_dynamic_frame.from_options(frame = dropnullfields3, connection_type = "s3", connection_options = {"path": "s3://big-data-bia678-1/final_parquet_files
job.commit()
```

Machine Learning Model Code

```
+--------------------+--------+
|[0.0,76.5,76.5,76...|    76.5|
|[0.0,36.8,36.8,36...|    36.8|
|[0.0,1.49,1.49,1....|    1.49|
|[0.0,0.17,0.17,0....|    0.17|
|[0.0,0.29,0.3,0.2...|     0.3|
|[0.0,0.68,0.68,0....|    0.68|
|[0.0,0.3,0.3,0.29...|     0.3|
|[0.0,0.3,0.3,0.3,...|     0.3|
|[0.0,0.32,0.32,0....|    0.32|
|[0.0,2.51,2.51,2....|    2.47|
|[0.0,0.07,0.07,0....|    0.06|
|[0.0,0.31,0.31,0....|    0.31|
|[0.0,0.31,0.31,0....|    0.31|
|[0.0,0.06,0.06,0....|    0.06|
|[0.0,0.45,0.46,0....|    0.45|
|[0.0,3.03,3.03,3....|    3.03|
|[0.0,0.86,0.87,0....|    0.87|
|[0.0,0.35,0.35,0....|    0.35|
|[0.0,1.46,1.46,1....|    1.46|
|[0.0,0.02,0.02,0....|    0.02|
+--------------------+--------+
only showing top 20 rows
```

In [184]:
```
train_data,test_data = final_data.randomSplit([0.8,0.2])
```

```
VBox()
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
```

In [123]:
```
from pyspark.ml.regression import LinearRegression
# Create a Linear Regression Model object
lr = LinearRegression(labelCol='EndPrice')
```

```
VBox()
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
```

In [124]:
```
# Fit the model to the data and call this model lrModel
lrModel = lr.fit(train_data)
```

```
VBox()
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
```

In [125]:
```
# Print the coefficients and intercept for linear regression
print("Coefficients: {} Intercept: {}".format(lrModel.coefficients,lrModel.intercept))
```

```
VBox()
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
```

```
Coefficients: [0.0,-0.7895038597924635,0.6970573469102191,1.0924483373703349,1.1862658397500759e-06] Intercept: 0.0006751078
477741135
```

In [126]:
```
test_results = lrModel.evaluate(test_data)
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

In [127]:
```
print("RMSE: {}".format(test_results.rootMeanSquaredError))
print("MSE: {}".format(test_results.meanSquaredError))
print("R2: {}".format(test_results.r2))
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

```
RMSE: 0.07517837747196177
MSE: 0.0056517884393167675
R2: 0.9999998033353552
```

In [149]:
```
#Strike price prediction
assembler = VectorAssembler(
  inputCols=['SecurityType_index',
             'MLEG_index',
             'PutOrCall_index',
             'StartPrice',
             'MaxPrice',
             'MinPrice',
             'EndPrice',
             'MarketSegment_index'
            ],
  outputCol="features")
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

In [150]:
```
output = assembler.transform(indexed)
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

In [151]:
```
output.select("features", "StrikePrice").show()
final_data = output.select("features", "StrikePrice")
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

In [150]:
```
output = assembler.transform(indexed)
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

In [151]:
```
output.select("features", "StrikePrice").show()
final_data = output.select("features", "StrikePrice")
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

```
+--------------------+-----------+
|            features|StrikePrice|
+--------------------+-----------+
|[0.0,7986.0,0.0,7...|    16100.0|
|[0.0,374.0,0.0,36...|     4200.0|
|[0.0,113921.0,1.0...|       34.0|
|[0.0,9847.0,0.0,0...|      170.5|
|[0.0,47.0,1.0,0.2...|      397.5|
|[0.0,11.0,0.0,0.6...|      385.0|
|[0.0,47.0,1.0,0.3...|      397.5|
|[0.0,17912.0,0.0,...|     3975.0|
|[0.0,87023.0,0.0,...|      100.0|
|[0.0,214.0,1.0,2....|      390.0|
|[0.0,66.0,0.0,0.0...|      375.0|
|[0.0,25766.0,0.0,...|        4.0|
|[0.0,25263.0,0.0,...|       12.6|
|[0.0,17.0,1.0,0.0...|      400.0|
|[0.0,67.0,1.0,0.4...|      395.0|
|[0.0,48.0,0.0,3.0...|      390.0|
|[0.0,167.0,1.0,0....|      392.5|
|[0.0,12481.0,0.0,...|       43.5|
|[0.0,4.0,1.0,1.46...|      402.5|
|[0.0,23022.0,0.0,...|      161.0|
+--------------------+-----------+
only showing top 20 rows
```

In [152]:
```
train_data,test_data = final_data.randomSplit([0.8,0.2])
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

In [152]:
```
train_data,test_data = final_data.randomSplit([0.8,0.2])
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

In [153]:
```
from pyspark.ml.regression import LinearRegression
# Create a Linear Regression Model object
lr = LinearRegression(labelCol='StrikePrice')
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

In [154]:
```
# Fit the model to the data and call this model lrModel
lrModel = lr.fit(train_data)
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

In [155]:
```
# Print the coefficients and intercept for linear regression
print("Coefficients: {} Intercept: {}".format(lrModel.coefficients,lrModel.intercept))
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

```
Coefficients: [0.0,-0.006268962377046943,-14.769061909112818,234.00185753585043,1391.9325096706416,-1389.746396593621,-226.
7172570459919,-6.520406974828376] Intercept: 2288.3953227901006
```

In [156]:
```
test_results = lrModel.evaluate(test_data)
```

VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

In [154]:
```
# Fit the model to the data and call this model lrModel
lrModel = lr.fit(train_data)
```

```
VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
```

In [155]:
```
# Print the coefficients and intercept for linear regression
print("Coefficients: {} Intercept: {}".format(lrModel.coefficients,lrModel.intercept))
```

```
VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

Coefficients: [0.0,-0.006268962377046943,-14.769061909112818,234.00185753585043,1391.9325096706416,-1389.746396593621,-226.7172570459919,-6.520406974828376] Intercept: 2288.3953227901006
```

In [156]:
```
test_results = lrModel.evaluate(test_data)
```

```
VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
```

In [157]:
```
print("RMSE: {}".format(test_results.rootMeanSquaredError))
print("MSE: {}".format(test_results.meanSquaredError))
print("R2: {}".format(test_results.r2))
```

```
VBox()

FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…

RMSE: 4115.592767868261
MSE: 16938103.830929536
R2: 0.1407630371203058
```

Time Series Analysis Code

```
data.shape
```

```
(158723, 17)
```

```
eurdata = data[data['Currency']=='EUR']
```

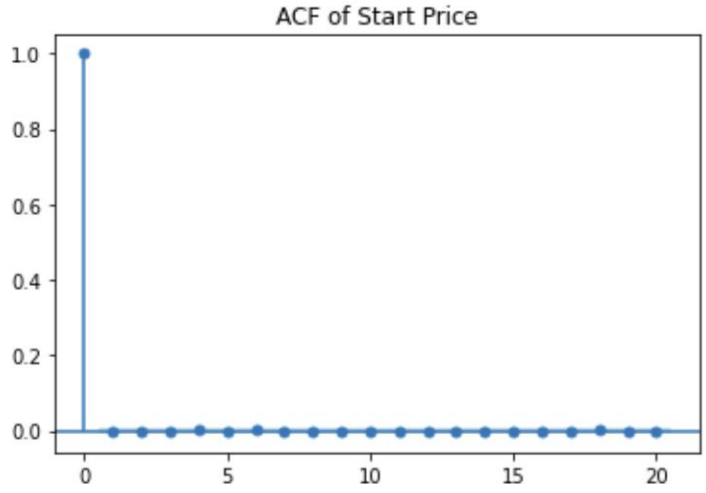
```
eurdata.shape
```

```
(87409, 17)
```

```
eurdata['StrikePrice'] = eurdata['StrikePrice'].astype(float)
eurdata['StartPrice'] = eurdata['StartPrice'].astype(float)
eurdata['MaxPrice'] = eurdata['MaxPrice'].astype(float)
eurdata['MinPrice'] = eurdata['MinPrice'].astype(float)
eurdata['EndPrice'] = eurdata['EndPrice'].astype(float)
```

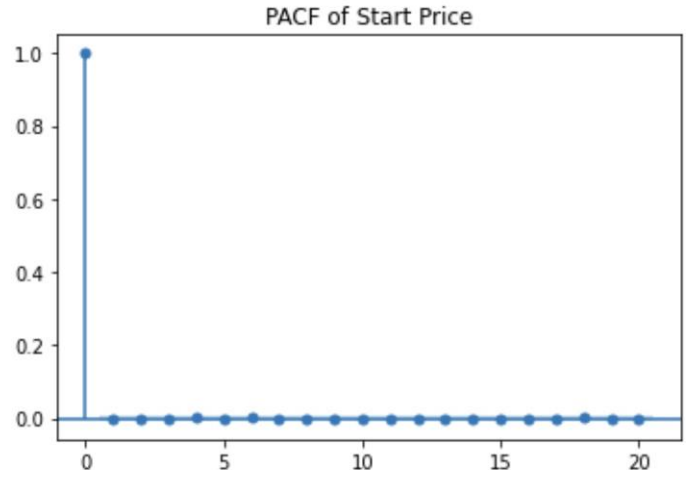
```
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf

# create acf plot
plt.figure(figsize=(1, 1))

plot_acf(eurdata['StartPrice']**2, lags=20)
plt.title("ACF of Start Price")
plt.show()
```

```
<Figure size 72x72 with 0 Axes>
```



```
# create pacf plot
plt.figure(figsize=(1, 1))

plot_pacf(eurdata['StartPrice']**2, lags=20)
plt.title("PACF of Start Price")
plt.show()
```

```
<Figure size 72x72 with 0 Axes>
```

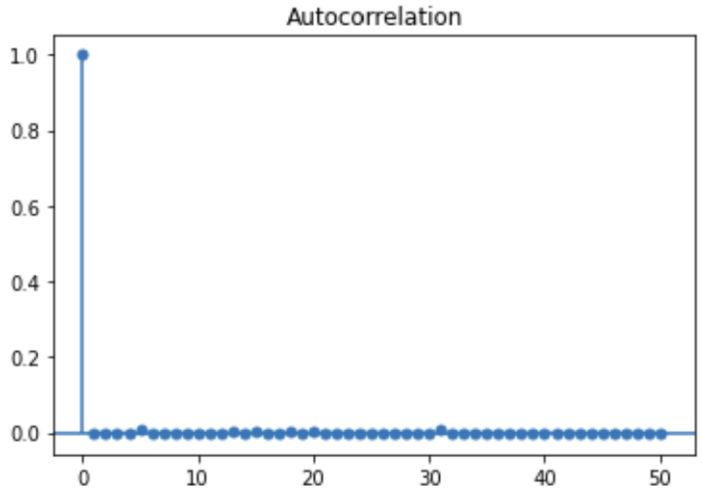



```
from scipy.stats import boxcox

startprice_transformed = boxcox(eurdata.StartPrice, 0)
returns = 100 * pd.DataFrame(startprice_transformed).pct_change().dropna()
returns = abs(returns)
returns = returns.mask(np.isinf(returns))
returns = returns.dropna()
```

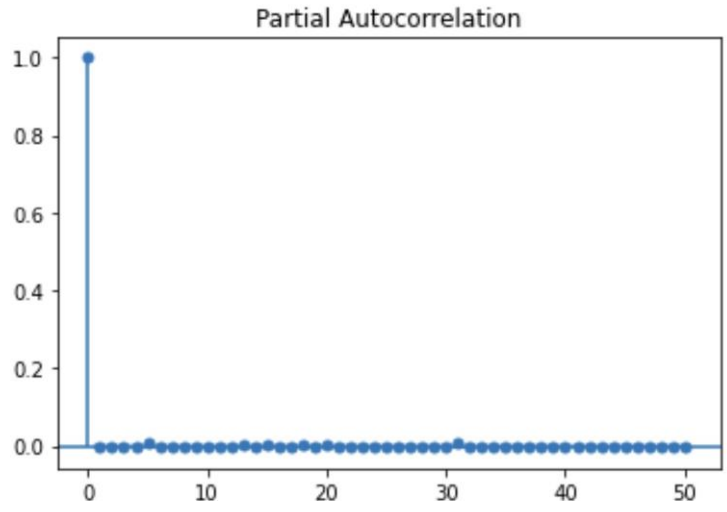
```
series = returns
result = adfuller(series, autolag='AIC')
print(f'ADF Statistic: {result[0]}')
print(f'n_lags: {result[1]}')
print(f'p-value: {result[1]}')
for key, value in result[4].items():
    print('Critial Values:')
    print(f'   {key}, {value}')
```

```
ADF Statistic: -304.10081512162594
n_lags: 0.0
p-value: 0.0
Critial Values:
   1%, -3.4304253397856868
Critial Values:
   5%, -2.861573298949126
Critial Values:
   10%, -2.5667877238751355
```

```
plot_acf(returns**2)
plt.show()
```




```
plot_pacf(returns**2)
plt.show()
```



GARCH(1,0) or ARCH(1) model

```
from arch import arch_model
model10 = arch_model(returns, p=1, q=0)
model10_fit = model10.fit()
model10_fit.summary()
```

```
Iteration:      1,   Func. Count:      5,   Neg. LLF: 783980.5351172888
Iteration:      2,   Func. Count:      8,   Neg. LLF: 783980.5362290283
Optimization terminated successfully    (Exit mode 0)
            Current function value: 783980.5351172888
            Iterations: 6
            Function evaluations: 8
            Gradient evaluations: 2
```

Constant Mean - ARCH Model Results

Dep. Variable:	0	**R-squared:**	0.000
Mean Model:	Constant Mean	**Adj. R-squared:**	0.000
Vol Model:	ARCH	**Log-Likelihood:**	-783981.
Distribution:	Normal	**AIC:**	1.56797e+06
Method:	Maximum Likelihood	**BIC:**	1.56800e+06
		No. Observations:	86801
Date:	Fri, May 06 2022	**Df Residuals:**	86800
Time:	18:02:29	**Df Model:**	1

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	413.3756	5.849	70.677	0.000	[4.019e+02,4.248e+02]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	1.5075e+06	7.321e+04	20.592	3.231e-94	[1.364e+06,1.651e+06]
alpha[1]	0.0000	1.033e-04	0.000	1.000	[-2.025e-04,2.025e-04]

```
warnings.filterwarnings("ignore")
start_time = datetime.now()

rolling_predictions10 = []
test_size10 = 360

for i in range(test_size10):
    train10 = returns[:-(test_size10-i)]
    model10 = arch_model(train10, p=1, q=0)
    model10_fit = model10.fit(disp='off')
    pred10 = model10_fit.forecast(horizon=1)
    rolling_predictions10.append(np.sqrt(pred10.variance.values[-1,:][0]))
end_time = datetime.now()
print("Time taken to execute ARCH(1) model predictions using 87409 records is ",end_time - start_time, " h:mm:ss")
```

```
Time taken to execute ARCH(1) model predictions using 87409 records is  0:01:02.027120  h:mm:ss
```

```
rolling_predictions10 = pd.Series(rolling_predictions10, index=returns.index[-360:])
plt.figure(figsize=(10,4))
true, = plt.plot(returns[-360:])
preds, = plt.plot(rolling_predictions10)
plt.title('ARCH(1) - Volatility Prediction - Rolling Forecast of Start Price', fontsize=20)
plt.legend(['True Returns', 'Predicted Volatility'], fontsize=16)
```

```
<matplotlib.legend.Legend at 0x175832e5b20>
```

GARCH(2,0) or ARCH(2)

```
from arch import arch_model
model20 = arch_model(returns, p=2, q=0)
model20_fit = model20.fit()
model20_fit.summary()
```

```
Iteration:      1,   Func. Count:      6,   Neg. LLF: 781267.8308785993
Iteration:      2,   Func. Count:     11,   Neg. LLF: 781222.0371381209
Iteration:      3,   Func. Count:     16,   Neg. LLF: 781178.6815637584
Iteration:      4,   Func. Count:     20,   Neg. LLF: 781178.6815919019
Optimization terminated successfully    (Exit mode 0)
            Current function value: 781178.6815637584
            Iterations: 8
            Function evaluations: 20
            Gradient evaluations: 4
```

Constant Mean - ARCH Model Results

Dep. Variable:	0	**R-squared:**	0.000
Mean Model:	Constant Mean	**Adj. R-squared:**	0.000
Vol Model:	ARCH	**Log-Likelihood:**	-781179.
Distribution:	Normal	**AIC:**	1.56237e+06
Method:	Maximum Likelihood	**BIC:**	1.56240e+06
		No. Observations:	86801
Date:	Fri, May 06 2022	**Df Residuals:**	86800
Time:	18:03:32	**Df Model:**	1

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	426.6334	57.572	7.410	1.259e-13	[3.138e+02,5.395e+02]

Volatility Model

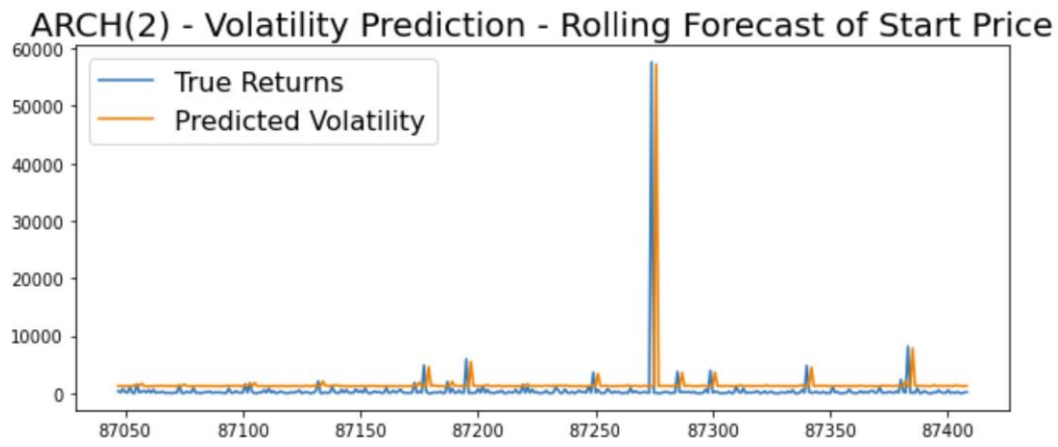
	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	1.5075e+06	9.056e+04	16.646	3.229e-62	[1.330e+06,1.685e+06]
alpha[1]	0.0000	1.294e-04	0.000	1.000	[-2.536e-04,2.536e-04]
alpha[2]	1.0000	0.255	3.917	8.951e-05	[0.500, 1.500]

```
start_time = datetime.now()

rolling_predictions20 = []
test_size20 = 360

for i in range(test_size20):
    train20 = returns[:-(test_size20-i)]
    model20 = arch_model(train20, p=2, q=0, dist = 'normal')
    model20_fit = model20.fit(disp='off')
    pred20 = model20_fit.forecast(horizon=1)
    rolling_predictions20.append(np.sqrt(pred20.variance.values[-1,:][0]))
end_time = datetime.now()
print("Time taken to execute ARCH(2) model predictions using 87409 records is ",end_time - start_time, " h:mm:ss")
```

```
Time taken to execute ARCH(2) model predictions using 87409 records is  0:01:25.408968  h:mm:ss
```

```
rolling_predictions20 = pd.Series(rolling_predictions20, index=returns.index[-360:])
plt.figure(figsize=(10,4))
true, = plt.plot(returns[-360:])
preds, = plt.plot(rolling_predictions20)
plt.title('ARCH(2) - Volatility Prediction - Rolling Forecast of Start Price', fontsize=20)
plt.legend(['True Returns', 'Predicted Volatility'], fontsize=16)
```

```
<matplotlib.legend.Legend at 0x175832e3d30>
```



GARCH(3,0)

```
model30 = arch_model(returns, p=3, q=0)
model30_fit = model30.fit()
model30_fit.summary()
```

```
Iteration:      1,   Func. Count:      7,   Neg. LLF: 781581.2934525444
Iteration:      2,   Func. Count:     13,   Neg. LLF: 781595.633125736
Iteration:      3,   Func. Count:     22,   Neg. LLF: 781210.4854496567
Iteration:      4,   Func. Count:     27,   Neg. LLF: 781210.4854706698
Optimization terminated successfully    (Exit mode 0)
            Current function value: 781210.4854496567
            Iterations: 8
            Function evaluations: 27
            Gradient evaluations: 4
```

Constant Mean - ARCH Model Results

Dep. Variable:	0	**R-squared:**	0.000
Mean Model:	Constant Mean	**Adj. R-squared:**	0.000
Vol Model:	ARCH	**Log-Likelihood:**	-781210.
Distribution:	Normal	**AIC:**	1.56243e+06
Method:	Maximum Likelihood	**BIC:**	1.56248e+06
		No. Observations:	86801
Date:	Fri, May 06 2022	**Df Residuals:**	86800
Time:	18:04:58	**Df Model:**	1

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	425.4549	58.836	7.231	4.786e-13	[3.101e+02,5.408e+02]

Volatility Model

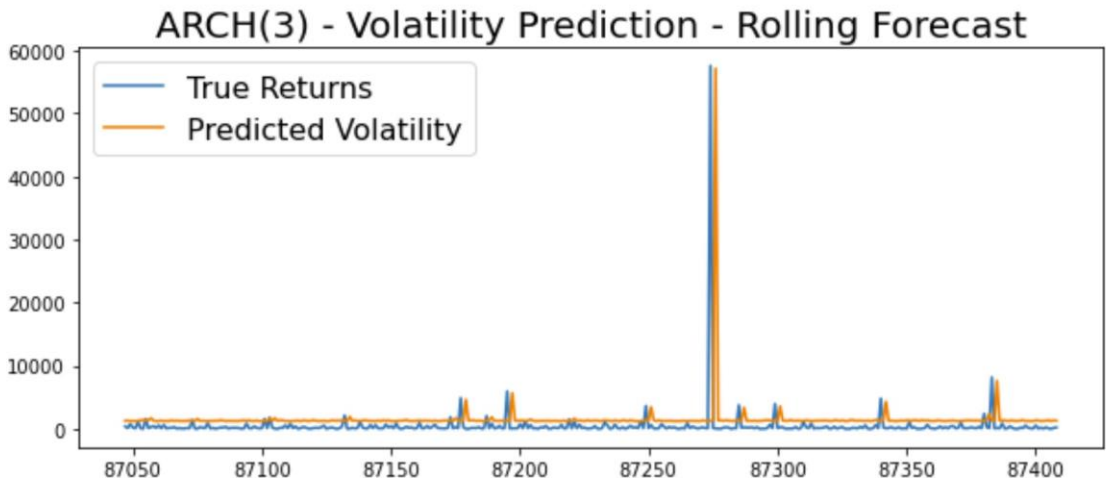
	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	1.5075e+06	9.105e+04	16.557	1.437e-61	[1.329e+06,1.686e+06]
alpha[1]	0.0000	1.298e-04	0.000	1.000	[-2.544e-04,2.544e-04]
alpha[2]	0.9861	0.255	3.870	1.091e-04	[0.487, 1.486]
alpha[3]	2.3636e-03	8.776e-03	0.269	0.788	[-1.484e-02,1.956e-02]

Covariance estimator: robust

```
warnings.filterwarnings("ignore")
start_time = datetime.now()

rolling_predictions30 = []
test_size30 = 360

for i in range(test_size30):
    train30 = returns[:-(test_size30-i)]
    model30 = arch_model(train30, p=3, q=0)
    model30_fit = model30.fit(disp='off')
    pred30 = model30_fit.forecast(horizon=1)
    rolling_predictions30.append(np.sqrt(pred30.variance.values[-1,:][0]))
end_time = datetime.now()
print("Time taken to execute ARCH(3) predictions using 87409 records is ",end_time - start_time, " h:mm:ss")
```

```
rolling_predictions30 = pd.Series(rolling_predictions30, index=returns.index[-360:])
plt.figure(figsize=(10,4))
true, = plt.plot(returns[-360:])
preds, = plt.plot(rolling_predictions30)
plt.title('ARCH(3) - Volatility Prediction - Rolling Forecast', fontsize=20)
plt.legend(['True Returns', 'Predicted Volatility'], fontsize=16)
```

```
<matplotlib.legend.Legend at 0x175837dd610>
```



GARCH(1,1)

```
model11 = arch_model(returns, p=1, q=1)
model11_fit = model11.fit()
model11_fit.summary()
```

```
Iteration:      1,   Func. Count:      6,   Neg. LLF: 1075486.2279215343
Iteration:      2,   Func. Count:     13,   Neg. LLF: 770688.7690133491
Iteration:      3,   Func. Count:     18,   Neg. LLF: 1124813.35250127
Iteration:      4,   Func. Count:     26,   Neg. LLF: 777400.0746668126
Iteration:      5,   Func. Count:     34,   Neg. LLF: 819786.8944707285
Iteration:      6,   Func. Count:     43,   Neg. LLF: 770664.6543735868
Iteration:      7,   Func. Count:     48,   Neg. LLF: 819677.7783889521
Iteration:      8,   Func. Count:     55,   Neg. LLF: 770661.7487263414
Optimization terminated successfully    (Exit mode 0)
            Current function value: 770661.7487264427
            Iterations: 12
            Function evaluations: 55
            Gradient evaluations: 8
```

0]:

Constant Mean - GARCH Model Results

Dep. Variable:	0	**R-squared:**	0.000
Mean Model:	Constant Mean	**Adj. R-squared:**	0.000
Vol Model:	GARCH	**Log-Likelihood:**	-770662.
Distribution:	Normal	**AIC:**	1.54133e+06
Method:	Maximum Likelihood	**BIC:**	1.54137e+06

		No. Observations:	86801
Date:	Fri, May 06 2022	**Df Residuals:**	86800
Time:	18:30:15	**Df Model:**	1

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	413.1322	7.200	57.383	0.000	[3.990e+02,4.272e+02]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	9.0452e+05	1.408e+07	6.424e-02	0.949	[-2.669e+07,2.850e+07]
alpha[1]	0.0000	5.477e-03	0.000	1.000	[-1.073e-02,1.073e-02]
beta[1]	0.7000	4.667	0.150	0.881	[-8.448, 9.848]

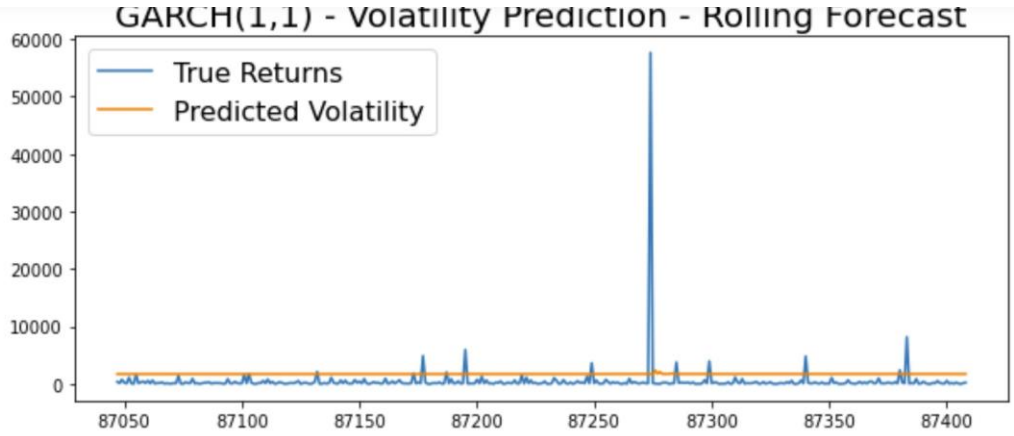
Covariance estimator: robust

```
start_time = datetime.now()

rolling_predictions11 = []
test_size11 = 360

for i in range(test_size11):
    train11 = returns[:-(test_size11-i)]
    model11 = arch_model(train11, p=1, q=1)
    model11_fit = model11.fit(disp='off')
    pred11 = model11_fit.forecast(horizon=1)
    rolling_predictions11.append(np.sqrt(pred11.variance.values[-1,:][0]))
end_time = datetime.now()
print("Time taken to execute GARCH(1,1) model predictions using 87409 records is ",end_time - start_time, " h:mm:ss")
```

```
Time taken to execute GARCH(1,1) model predictions using 87409 records is  0:01:42.050088  h:mm:ss
```

```
rolling_predictions11 = pd.Series(rolling_predictions11, index=returns.index[-360:])

plt.figure(figsize=(10,4))
true, = plt.plot(returns[-360:])
preds, = plt.plot(rolling_predictions11)
plt.title('GARCH(1,1) - Volatility Prediction - Rolling Forecast', fontsize=20)
plt.legend(['True Returns', 'Predicted Volatility'], fontsize=16)
```

```
<matplotlib.legend.Legend at 0x1758330c190>
```



GARCH(2,2)

```
model22 = arch_model(returns, p=2, q=2)
model22_fit = model22.fit()
model22_fit.summary()
```

```
Iteration:      1,   Func. Count:      8,   Neg. LLF: 1034134.2812966239
Iteration:      2,   Func. Count:     17,   Neg. LLF: 770687.4365188936
Iteration:      3,   Func. Count:     24,   Neg. LLF: 813997.8146665256
Iteration:      4,   Func. Count:     33,   Neg. LLF: 770680.0945812084
Optimization terminated successfully    (Exit mode 0)
            Current function value: 770680.0945923924
```

```
Current function value: 770680.0945923924
Iterations: 8
Function evaluations: 33
Gradient evaluations: 4
```

Constant Mean - GARCH Model Results

Dep. Variable:	0	**R-squared:**	0.000
Mean Model:	Constant Mean	**Adj. R-squared:**	0.000
Vol Model:	GARCH	**Log-Likelihood:**	-770680.
Distribution:	Normal	**AIC:**	1.54137e+06
Method:	Maximum Likelihood	**BIC:**	1.54143e+06
		No. Observations:	86801
Date:	Fri, May 06 2022	**Df Residuals:**	86800
Time:	18:31:57	**Df Model:**	1

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	413.0598	6.069	68.065	0.000	[4.012e+02,4.250e+02]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	9.0452e+05	2.024e+06	0.447	0.655	[-3.062e+06,4.871e+06]

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	9.0452e+05	2.024e+06	0.447	0.655	[-3.062e+06,4.871e+06]
alpha[1]	6.0461e-03	3.952e-03	1.530	0.126	[-1.700e-03,1.379e-02]
alpha[2]	1.1214e-03	6.085e-03	0.184	0.854	[-1.081e-02,1.305e-02]
beta[1]	1.3899e-03	0.338	4.112e-03	0.997	[-0.661, 0.664]
beta[2]	0.6931	0.365	1.898	5.769e-02	[-2.261e-02, 1.409]

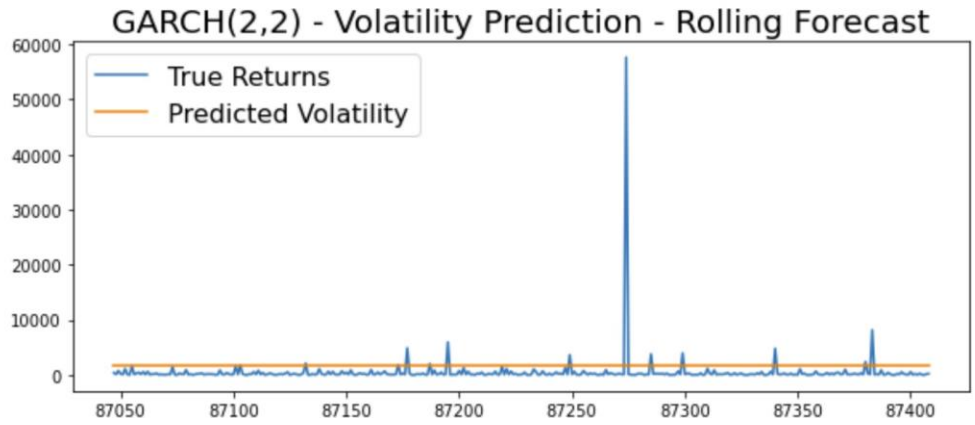
Covariance estimator: robust

```
start_time = datetime.now()
rolling_predictions22 = []
test_size22 = 360

for i in range(test_size22):
    train22 = returns[:-(test_size22-i)]
    model = arch_model(train22, p=2, q=2)
    model_fit = model22.fit(disp='off')
    pred22 = model22_fit.forecast(horizon=1)
    rolling_predictions22.append(np.sqrt(pred22.variance.values[-1,:][0]))
end_time = datetime.now()
print("Time taken to execute GARCH(2,2) model predictions using 87409 records is ",end_time - start_time, " h:mm:ss")
```

```
Time taken to execute GARCH(2,2) model predictions using 87409 records is  0:02:05.212694  h:mm:ss
```

```
rolling_predictions22 = pd.Series(rolling_predictions22, index=returns.index[-360:])
plt.figure(figsize=(10,4))
true, = plt.plot(returns[-360:])
preds, = plt.plot(rolling_predictions22)
plt.title('GARCH(2,2) - Volatility Prediction - Rolling Forecast', fontsize=20)
plt.legend(['True Returns', 'Predicted Volatility'], fontsize=16)
```

```
<matplotlib.legend.Legend at 0x175848e9400>
```



GARCH(3,3)

```
model33 = arch_model(returns, p=3, q=3)
model33_fit = model33.fit()
model33_fit.summary()
```

```
Iteration:      1,   Func. Count:     10,   Neg. LLF: 970466.9470609282
Iteration:      2,   Func. Count:     21,   Neg. LLF: 770630.0754125844
Iteration:      3,   Func. Count:     30,   Neg. LLF: 1095093.0006374482
Iteration:      4,   Func. Count:     42,   Neg. LLF: 810686.5150821721
Iteration:      5,   Func. Count:     55,   Neg. LLF: 846551.3810422946
Iteration:      6,   Func. Count:     66,   Neg. LLF: 770615.9264292385
Iteration:      7,   Func. Count:     76,   Neg. LLF: 811853.1637450479
Iteration:      8,   Func. Count:     88,   Neg. LLF: 819437.6669176227
Iteration:      9,   Func. Count:    100,   Neg. LLF: 770491.2959290533
Optimization terminated successfully    (Exit mode 0)
            Current function value: 770491.2958916024
            Iterations: 13
            Function evaluations: 100
            Gradient evaluations: 9
```

6]:

Constant Mean - GARCH Model Results

Dep. Variable:	0	**R-squared:**	0.000
Mean Model:	Constant Mean	**Adj. R-squared:**	0.000
Vol Model:	GARCH	**Log-Likelihood:**	-770491.
Distribution:	Normal	**AIC:**	1.54100e+06

Method:	Maximum Likelihood	**BIC:**	1.54107e+06
		No. Observations:	86801
Date:	Fri, May 06 2022	**Df Residuals:**	86800
Time:	18:34:04	**Df Model:**	1

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	413.6076	5.846	70.745	0.000	[4.021e+02,4.251e+02]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	9.0452e+05	8.419e+05	1.074	0.283	[-7.456e+05,2.555e+06]
alpha[1]	1.7817e-03	1.542e-03	1.156	0.248	[-1.240e-03,4.804e-03]
alpha[2]	5.9754e-04	2.998e-03	0.199	0.842	[-5.278e-03,6.473e-03]
alpha[3]	0.0112	8.961e-03	1.253	0.210	[-6.331e-03,2.879e-02]
beta[1]	2.8569e-04	0.400	7.143e-04	0.999	[-0.784, 0.784]
beta[2]	0.6919	0.217	3.185	1.447e-03	[0.266, 1.118]
beta[3]	2.8193e-04	0.340	8.299e-04	0.999	[-0.666, 0.666]

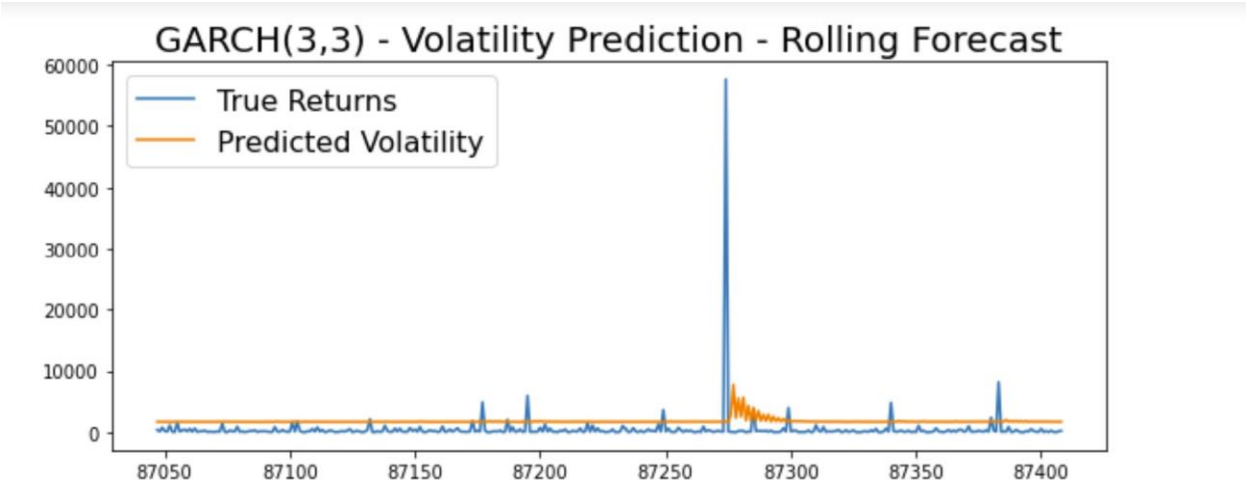
Covariance estimator: robust

```
start_time = datetime.now()
rolling_predictions33 = []
test_size33 = 360

for i in range(test_size33):
    train33 = returns[:-(test_size33-i)]
    model33 = arch_model(train33, p=3, q=3)
    model33_fit = model33.fit(disp='off')
    pred33 = model33_fit.forecast(horizon=1)
    rolling_predictions33.append(np.sqrt(pred33.variance.values[-1,:][0]))
end_time = datetime.now()
print("Time taken to execute GARCH(3,3) model predictions using 87409 records is ",end_time - start_time, " h:mm:ss")
```

```
Time taken to execute GARCH(3,3) model predictions using 87409 records is  0:03:11.070125  h:mm:ss
```

```
rolling_predictions33 = pd.Series(rolling_predictions33, index=returns.index[-360:])
plt.figure(figsize=(10,4))
true, = plt.plot(returns[-360:])
preds, = plt.plot(rolling_predictions33)
plt.title('GARCH(3,3) - Volatility Prediction - Rolling Forecast', fontsize=20)
plt.legend(['True Returns', 'Predicted Volatility'], fontsize=16)
```

```
<matplotlib.legend.Legend at 0x17584ddfa30>
```



Performance Metrics

Performance Metrics

```
print("Results of sklearn.metrics ARCH(1):")
import sklearn.metrics as metrics
mae = metrics.mean_absolute_error(returns[-360:], rolling_predictions10[-360:])
mse = metrics.mean_squared_error(returns[-360:], rolling_predictions10[-360:])
rmse = np.sqrt(mse) # or mse**(0.5)
r2 = metrics.r2_score(returns[-360:],rolling_predictions10[-360:])
ev_score = metrics.explained_variance_score(returns[-360:], rolling_predictions10[-360:])
print("Explained Variance Score is ",ev_score)
print("MAE:",mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R-Squared:", r2)
```

```
Results of sklearn.metrics ARCH(1):
Explained Variance Score is  -0.9373423706916968
MAE: 1496.7449893239113
MSE: 19786792.83359942
RMSE: 4448.234799737916
R-Squared: -1.0434978714910104
```

```
print("Results of sklearn.metrics GARCH(1,1):")
import sklearn.metrics as metrics
mae = metrics.mean_absolute_error(returns[-360:], rolling_predictions11[-360:])
mse = metrics.mean_squared_error(returns[-360:], rolling_predictions11[-360:])
rmse = np.sqrt(mse) # or mse**(0.5)
r2 = metrics.r2_score(returns[-360:],rolling_predictions11[-360:])
ev_score = metrics.explained_variance_score(returns[-360:], rolling_predictions11)
print("Explained Variance Score is ",ev_score)
print("MAE:",mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R-Squared:", r2)
```

```
Results of sklearn.metrics GARCH(1,1):
Explained Variance Score is  -0.0005924616559711549
MAE: 1669.8379155318603
MSE: 11182742.796948161
RMSE: 3344.060824349366
R-Squared: -0.1549072805872198
```

```
print("Results of sklearn.metrics ARCH(2):")
import sklearn.metrics as metrics
mae = metrics.mean_absolute_error(returns[-360:], rolling_predictions20[-360:])
mse = metrics.mean_squared_error(returns[-360:], rolling_predictions20[-360:])
rmse = np.sqrt(mse) # or mse**(0.5)
r2 = metrics.r2_score(returns[-360:],rolling_predictions20[-360:])
ev_score = metrics.explained_variance_score(returns[-360:], rolling_predictions20[-360:])
print("Explained Variance Score is ",ev_score)
print("MAE:",mae)
```

```
print("MAE:",mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R-Squared:", r2)
```

```
Results of sklearn.metrics ARCH(2):
Explained Variance Score is  -0.9406313835411455
MAE: 1487.6571436550553
MSE: 19802049.971468467
RMSE: 4449.94943470917
R-Squared: -1.045073565390605
```

```
print("Results of sklearn.metrics ARCH(3):")
import sklearn.metrics as metrics
mae = metrics.mean_absolute_error(returns[-360:], rolling_predictions30[-360:])
mse = metrics.mean_squared_error(returns[-360:], rolling_predictions30[-360:])
rmse = np.sqrt(mse) # or mse**(0.5)
r2 = metrics.r2_score(returns[-360:],rolling_predictions30[-360:])
ev_score = metrics.explained_variance_score(returns[-360:], rolling_predictions30)
print("Explained Variance Score is ",ev_score)
print("MAE:",mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R-Squared:", r2)
```

```
Results of sklearn.metrics ARCH(3):
Explained Variance Score is  -0.9373423706916968
MAE: 1496.7449893239113
MSE: 19786792.83359942
RMSE: 4448.234799737916
R-Squared: -1.0434978714910104
```

```
print("Results of sklearn.metrics GARCH(2,2):")
import sklearn.metrics as metrics
mae = metrics.mean_absolute_error(returns[-360:], rolling_predictions22[-360:])
mse = metrics.mean_squared_error(returns[-360:], rolling_predictions22[-360:])
rmse = np.sqrt(mse) # or mse**(0.5)
r2 = metrics.r2_score(returns[-360:],rolling_predictions22[-360:])
ev_score = metrics.explained_variance_score(returns[-360:], rolling_predictions22)
print("Explained Variance Score is ",ev_score)
print("MAE:",mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R-Squared:", r2)
```

```
Results of sklearn.metrics GARCH(2,2):
Explained Variance Score is  2.220446049250313e-16
MAE: 1657.7410160223017
MSE: 11145946.135833029
RMSE: 3338.5544979576157
R-Squared: -0.15110707409094193
```

```
print("Results of sklearn.metrics GARCH(3,3):")
import sklearn.metrics as metrics
mae = metrics.mean_absolute_error(returns[-360:], rolling_predictions33[-360:])
mse = metrics.mean_squared_error(returns[-360:], rolling_predictions33[-360:])
rmse = np.sqrt(mse) # or mse**(0.5)
r2 = metrics.r2_score(returns[-360:],rolling_predictions33[-360:])
ev_score = metrics.explained_variance_score(returns[-360:], rolling_predictions33)
print("Explained Variance Score is ",ev_score)
print("MAE:",mae)
print("MSE:", mse)
```

```
print("MSE:", mse)
print("RMSE:", rmse)
print("R-Squared:", r2)
```

```
Results of sklearn.metrics GARCH(3,3):
Explained Variance Score is  -0.026320513940859547
MAE: 1717.2343980863147
MSE: 11578573.152665757
RMSE: 3402.7302497649966
R-Squared: -0.19578699748640838
```

Rerunning the predictions for 60,000 records

```
model10 = arch_model(returns[-60000:], p=2, q=0)
model10_fit = model10.fit()
model10_fit.summary()
```

```
Iteration:      1,   Func. Count:      6,   Neg. LLF: 538589.7948272537
Iteration:      2,   Func. Count:     11,   Neg. LLF: 731317.1706183292
Iteration:      3,   Func. Count:     20,   Neg. LLF: 2254452261.454536
Iteration:      4,   Func. Count:     33,   Neg. LLF: 142178708704.6556
Iteration:      5,   Func. Count:     48,   Neg. LLF: 537101.9398172551
Iteration:      6,   Func. Count:     52,   Neg. LLF: 537101.9398598868
Optimization terminated successfully    (Exit mode 0)
            Current function value: 537101.9398172551
            Iterations: 10
            Function evaluations: 52
            Gradient evaluations: 6
```

5]:

Constant Mean - ARCH Model Results

Dep. Variable:	0	**R-squared:**	0.000
Mean Model:	Constant Mean	**Adj. R-squared:**	0.000
Vol Model:	ARCH	**Log-Likelihood:**	-537102.
Distribution:	Normal	**AIC:**	1.07421e+06

Distribution:	Normal	**AIC:**	1.07421e+06
Method:	Maximum Likelihood	**BIC:**	1.07425e+06
		No. Observations:	60000
Date:	Fri, May 06 2022	**Df Residuals:**	59999
Time:	18:19:36	**Df Model:**	1

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	753.9503	50.019	15.073	2.425e-51	[6.559e+02,8.520e+02]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	1.3648e+06	1.502e+05	9.084	1.043e-19	[1.070e+06,1.659e+06]
alpha[1]	0.0000	3.321e-04	0.000	1.000	[-6.509e-04,6.509e-04]
alpha[2]	1.0000	0.207	4.840	1.298e-06	[0.595, 1.405]

Covariance estimator: robust

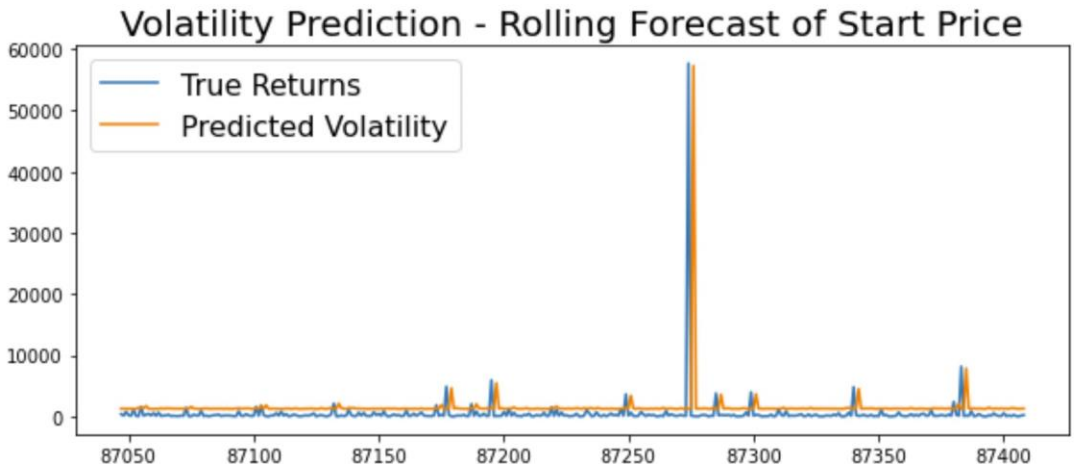
```
start_time = datetime.now()

rolling_predictions10 = []
```

```
start_time = datetime.now()

rolling_predictions10 = []
test_size10 = 360

for i in range(test_size10):
    train10 = returns[:-(test_size10-i)]
    model10 = arch_model(train10, p=2, q=0)
    model10_fit = model10.fit(disp='off')
    pred10 = model10_fit.forecast(horizon=1)
    rolling_predictions10.append(np.sqrt(pred10.variance.values[-1,:][0]))
end_time = datetime.now()
print("Time taken to execute predictions using 60000 records is ",end_time - start_time, " h:mm:ss")
```

```
Time taken to execute predictions using 60000 records is  0:01:40.172163  h:mm:ss
```

```
rolling_predictions10 = pd.Series(rolling_predictions10, index=returns.index[-360:])
plt.figure(figsize=(10,4))
true, = plt.plot(returns[-360:])
preds, = plt.plot(rolling_predictions10)
plt.title('Volatility Prediction - Rolling Forecast of Start Price', fontsize=20)
plt.legend(['True Returns', 'Predicted Volatility'], fontsize=16)
```

```
<matplotlib.legend.Legend at 0x23d01d5f070>
```



Rerunning the predictions for 40,000 records

```
model10 = arch_model(returns[-40000:], p=2, q=0)
model10_fit = model10.fit()
model10_fit.summary()
```

```
Iteration:      1,   Func. Count:      6,   Neg. LLF: 358794.85208549077
Iteration:      2,   Func. Count:     14,   Neg. LLF: 359038.3587055074
Iteration:      3,   Func. Count:     19,   Neg. LLF: 358678.353944883
Iteration:      4,   Func. Count:     24,   Neg. LLF: 358587.2177891553
Iteration:      5,   Func. Count:     29,   Neg. LLF: 358518.22139655624
Iteration:      6,   Func. Count:     33,   Neg. LLF: 358518.22138734185
Positive directional derivative for linesearch    (Exit mode 8)
            Current function value: 358518.22139655624
            Iterations: 10
            Function evaluations: 33
            Gradient evaluations: 6
```

```
C:\Users\Chrisia\anaconda3\lib\site-packages\arch\univariate\base.py:753: ConvergenceWarning: The optimizer returned code 8.
The message is:
Positive directional derivative for linesearch
See scipy.optimize.fmin_slsqp for code meaning.

  warnings.warn(
```

9]:

Constant Mean - ARCH Model Results

Dep. Variable:	0	**R-squared:**	0.000
Mean Model:	Constant Mean	**Adj. R-squared:**	0.000

Vol Model:	ARCH	**Log-Likelihood:**	-358518.
Distribution:	Normal	**AIC:**	717044.
Method:	Maximum Likelihood	**BIC:**	717079.
		No. Observations:	40000
Date:	Fri, May 06 2022	**Df Residuals:**	39999
Time:	18:21:30	**Df Model:**	1

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	477.6238	28.489	16.765	4.377e-63	[4.218e+02,5.335e+02]

Volatility Model

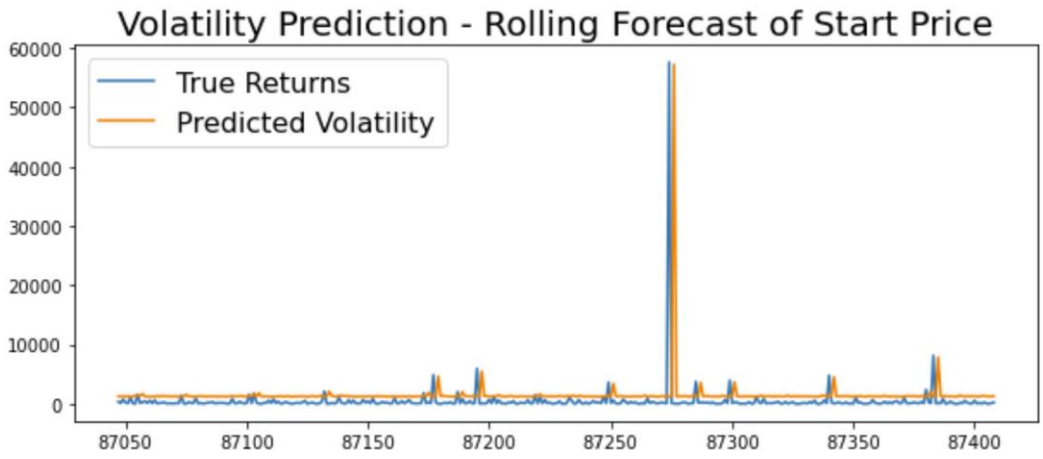
	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	1.3570e+06	1.099e+05	12.343	5.323e-35	[1.142e+06,1.573e+06]
alpha[1]	4.9954e-04	3.166e-04	1.578	0.115	[-1.209e-04,1.120e-03]
alpha[2]	1.0000	0.232	4.311	1.625e-05	[0.545, 1.455]

Covariance estimator: robust
WARNING: The optimizer did not indicate successful convergence. The message was Positive directional derivative for linesearch. See convergence_flag.

```
start_time = datetime.now()

rolling_predictions10 = []
test_size10 = 360

for i in range(test_size10):
    train10 = returns[:-(test_size10-i)]
    model10 = arch_model(train10, p=2, q=0)
    model10_fit = model10.fit(disp='off')
    pred10 = model10_fit.forecast(horizon=1)
    rolling_predictions10.append(np.sqrt(pred10.variance.values[-1,:][0]))
end_time = datetime.now()
print("Time taken to execute predictions using 40000 records is ",end_time - start_time, " h:mm:ss")
```

```
rolling_predictions10 = pd.Series(rolling_predictions10, index=returns.index[-360:])
plt.figure(figsize=(10,4))
true, = plt.plot(returns[-360:])
preds, = plt.plot(rolling_predictions10)
plt.title('Volatility Prediction - Rolling Forecast of Start Price', fontsize=20)
plt.legend(['True Returns', 'Predicted Volatility'], fontsize=16)
```

```
<matplotlib.legend.Legend at 0x23d01e32430>
```



Rerunning the predictions for 20,000 records

```
model10 = arch_model(returns[-20000:], p=2, q=0)
model10_fit = model10.fit()
model10_fit.summary()
```

```
Iteration:      1,   Func. Count:      6,   Neg. LLF: 178967.75426540046
Iteration:      2,   Func. Count:     12,   Neg. LLF: 179066.29034747035
Iteration:      3,   Func. Count:     17,   Neg. LLF: 178971.19693437015
Iteration:      4,   Func. Count:     22,   Neg. LLF: 178961.81290413172
Iteration:      5,   Func. Count:     27,   Neg. LLF: 178950.81388213922
Iteration:      6,   Func. Count:     31,   Neg. LLF: 178950.81388560805
Optimization terminated successfully    (Exit mode 0)
            Current function value: 178950.81388213922
            Iterations: 10
            Function evaluations: 31
            Gradient evaluations: 6
```

3]:

Constant Mean - ARCH Model Results

Dep. Variable:	0	**R-squared:**	0.000
Mean Model:	Constant Mean	**Adj. R-squared:**	0.000
Vol Model:	ARCH	**Log-Likelihood:**	-178951.
Distribution:	Normal	**AIC:**	357910.
Method:	Maximum Likelihood	**BIC:**	357941.
		No. Observations:	20000

Method:	Maximum Likelihood	**BIC:**	357941.
		No. Observations:	20000
Date:	Fri, May 06 2022	**Df Residuals:**	19999
Time:	18:23:51	**Df Model:**	1

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	414.2788	47.769	8.673	4.224e-18	[3.207e+02,5.079e+02]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	1.3346e+06	1.581e+05	8.440	3.187e-17	[1.025e+06,1.644e+06]
alpha[1]	0.0000	2.513e-04	0.000	1.000	[-4.925e-04,4.925e-04]
alpha[2]	1.0000	0.265	3.769	1.640e-04	[0.480, 1.520]

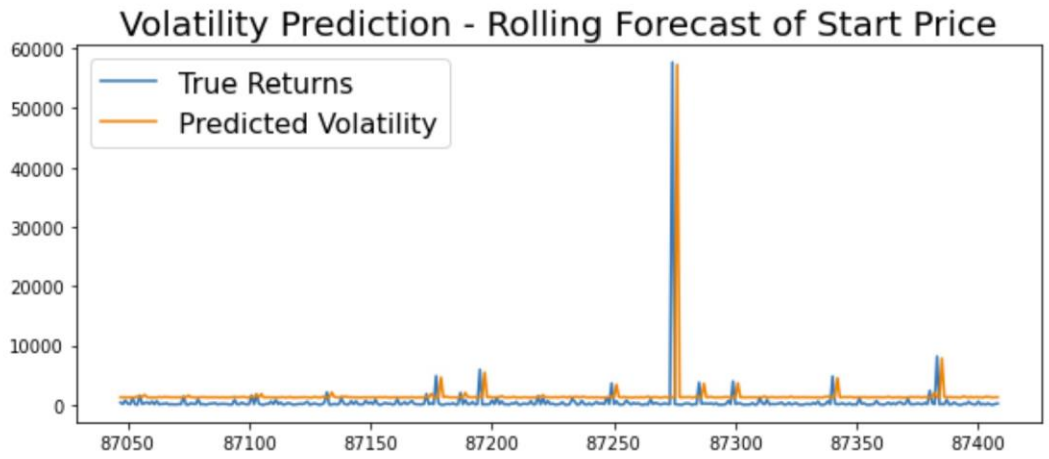
Covariance estimator: robust

```
warnings.filterwarnings("ignore")
start_time = datetime.now()

rolling_predictions10 = []
test_size10 = 360

for i in range(test_size10):
    train10 = returns[:-(test_size10-i)]
    model10 = arch_model(train10, p=2, q=0)
    model10_fit = model10.fit(disp='off')
    pred10 = model10_fit.forecast(horizon=1)
    rolling_predictions10.append(np.sqrt(pred10.variance.values[-1,:][0]))
end_time = datetime.now()
print("Time taken to execute predictions using 20000 records is ",end_time - start_time, " h:mm:ss")
```

```
Time taken to execute predictions using 20000 records is  0:01:27.221510  h:mm:ss
```

```
rolling_predictions10 = pd.Series(rolling_predictions10, index=returns.index[-360:])
plt.figure(figsize=(10,4))
true, = plt.plot(returns[-360:])
preds, = plt.plot(rolling_predictions10)
plt.title('Volatility Prediction - Rolling Forecast of Start Price', fontsize=20)
plt.legend(['True Returns', 'Predicted Volatility'], fontsize=16)
```

```
<matplotlib.legend.Legend at 0x23d03031a60>
```



For 100 records

```
model10 = arch_model(returns[-100:], p=2, q=0)
model10_fit = model10.fit()
model10_fit.summary()
```

```
Iteration:      1,   Func. Count:      6,   Neg. LLF: 844.1790325987181
Iteration:      2,   Func. Count:     10,   Neg. LLF: 844.1790338679338
Optimization terminated successfully    (Exit mode 0)
            Current function value: 844.1790325987181
            Iterations: 3
            Function evaluations: 10
            Gradient evaluations: 2
```

9]:

Constant Mean - ARCH Model Results

Dep. Variable:	0	**R-squared:**	0.000
Mean Model:	Constant Mean	**Adj. R-squared:**	0.000
Vol Model:	ARCH	**Log-Likelihood:**	-844.179
Distribution:	Normal	**AIC:**	1696.36
Method:	Maximum Likelihood	**BIC:**	1706.78
		No. Observations:	100
Date:	Fri, May 06 2022	**Df Residuals:**	99
Time:	18:27:20	**Df Model:**	1

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	353.0877	88.045	4.010	6.064e-05	[1.805e+02,5.257e+02]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	4.6310e+05	2.111e+05	2.194	2.826e-02	[4.934e+04,8.769e+05]
alpha[1]	2.1129e-11	1.144e-02	1.847e-09	1.000	[-2.242e-02,2.242e-02]
alpha[2]	2.4667e-11	9.678e-03	2.549e-09	1.000	[-1.897e-02,1.897e-02]

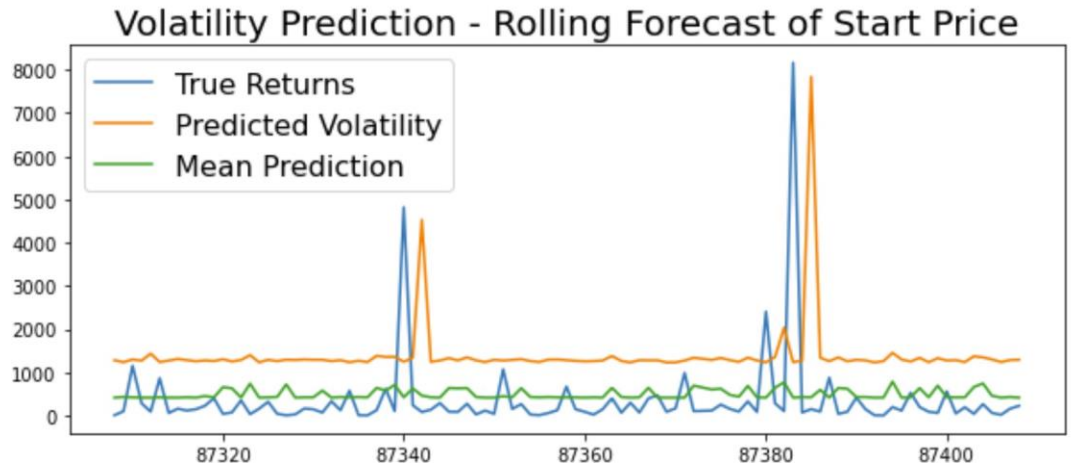
Covariance estimator: robust

Covariance estimator: robust

```
start_time =datetime.now()

rolling_predictions10 = []
predictions10 = []
test_size10 = 100

for i in range(test_size10):
    train10 = returns[:-(test_size10-i)]
    model10 = arch_model(train10, p=2, q=0)
    model10_fit = model10.fit(disp='off')
    pred10 = model10_fit.forecast(horizon=1)
    rolling_predictions10.append(np.sqrt(pred10.variance.values[-1,:][0]))
    predictions10.append(pred10.mean.values[-1,:][0])
end_time = datetime.now()
print("Time taken to execute predictions using 100 records is ",end_time - start_time, " h:mm:ss")
```

```
rolling_predictions10 = pd.Series(rolling_predictions10, index=returns.index[-100:])
predictions10 = pd.Series(predictions10, index=returns.index[-100:])

plt.figure(figsize=(10,4))
true, = plt.plot(returns[-100:])
preds, = plt.plot(rolling_predictions10)
preds_mean, = plt.plot(predictions10)
plt.title('Volatility Prediction - Rolling Forecast of Start Price', fontsize=20)
plt.legend(['True Returns', 'Predicted Volatility', 'Mean Prediction'], fontsize=16)
```

```
<matplotlib.legend.Legend at 0x23d02fea910>
```



```
import sklearn.metrics as metrics
mae = metrics.mean_absolute_error(returns[-100:], rolling_predictions10[-100:])
mse = metrics.mean_squared_error(returns[-100:], rolling_predictions10[-100:])
rmse = np.sqrt(mse) # or mse**(0.5)
r2 = metrics.r2_score(returns[-100:],rolling_predictions10[-100:])
ev_score = metrics.explained_variance_score(returns[-100:], rolling_predictions10)
print("Explained Variance Score is ",ev_score)
print("Results of sklearn.metrics:")
print("MAE:",mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R-Squared:", r2)
```

```
Explained Variance Score is  -0.6422750321805772
Results of sklearn.metrics:
MAE: 1271.096747853803
MSE: 2597558.549993291
RMSE: 1611.694310343401
R-Squared: -1.8045182472314218
```