MAA107 Project
Time-Series Forecasting: Testing The Combination of the Arma and Garch Model For The Price of the S&P500 Index

Lorenzo De Giovanni, Andrei Bornea, Simone Spallacci

May 2022

1 Introduction

1.1 Time-series and forecasting models

Time-series forecasting models are models that analyze a sequence of non-stationary data points collected over an interval of time. Non-stationary data is data whose statistical features, such as the mean and standard deviation, do not remain constant throughout time. The non-stationary input data used as input to these models are commonly referred to as time-series. Temperature values over time, heights of ocean tides over time, and the daily closing value of the S&P500 Index over time, are all examples of time-series. As a result, the input is a signal (time-series) made up of observations gathered in a particular order across time.

The ability to depict how variables change over time distinguishes time-series data from other types of data. In other words, time is an important variable since it reveals how the data changes through time as well as the final outcomes. It provides an additional source of data as well as a predetermined order of data dependencies.

To maintain consistency and dependability, time series analysis often requires a high number of data points. A large data collection ensures that your sample size is representative and that your analysis can cut through noisy data (anomalies). It also guarantees that any trends or patterns found aren't outliers and that seasonal variation is taken into account. Time series data can also be used for forecasting or anticipating future data based on previous data.

1.2 The ARMA Model

ARMA is a forecasting model in which the methods of autoregression (AR) analysis and moving average (MA) are both used on well-behaved time-series data. The time series is believed to be stationary in ARMA, and when it fluctuates, it does so uniformly around a specific time.

Moving average models account for the premise that returns are influenced not just by present information, but also by signals received in the past. This could occur if new information is processed slowly or reaches market participants at various times. As a result, any new signal has an instantaneous effect as well as a delayed one.

Autoregressive models presume that current returns and their own history have a linear connection. When (some) investors base their judgments on recent price movements, this type of model can be used: in a bull market, profits attract more buyers, driving up the price even higher; and dropping prices are viewed as a sell signal, prolonging the downward trend.

These two concepts can be merged, and the resulting model is known as an autoregressive moving average model: The model is often referred to as the ARMA(p,q) model, where p is the order of the autoregressive polynomial and q is the order of the moving average polynomial.

The equation is given by:

$$X_t = c + \epsilon_t + \sum_{i=1}^{p} \phi_i X_{t-i} + \sum_{i=1}^{q} \theta_i \epsilon_{t-i}$$

Where,

1. ϕ = the autoregressive model's parameters
2. θ = the moving average model's parameters
3. c = a constant
4. ϵ = error term (white noise)

Using the ARMA model in econometric analysis usually begins with identification, meaning determining p and q. The parameters can be estimated using one of three methods: minimizing residual squares, maximizing likelihood, or utilizing an information criterion. The latter takes into account not only the residual sum of squares, but also the number of parameters in the model and the number of available data, and can thus help with identification.

1.3 The Arch and Garch Model

1.3.1 Introduction

These two models are used as standard tools when econometrician are asked to forecast and analyze the size of the errors of the model they are working on. In most of the models developed by econometrician, the use of the least squares model seems to be a must. The most basic version of the least squares model implies that when all error components are squared, the expected value is the same at every given position. Homoskedasticity is the term for this assumption, and it is this assumption that ARCH/GARCH is concerned with. Heteroskedasticity is defined as data in which the variances of the error terms are not equal, in which the error terms may fairly be expected to be bigger for certain points or ranges of the data than for others. The traditional caution is that even if the regression coefficients for an ordinary least squares regression are still unbiased in the presence of heteroskedasticity, the standard errors and confidence intervals generated by conventional approaches will be excessively narrow, creating a misleading feeling of accuracy. Instead of treating heteroskedasticity as a problem to be solved, ARCH and GARCH models regard it as a variance to be modeled. As a consequence, not only are the flaws of least squares rectified, but a forecast for the variance of each error term is generated as well. This forecast is frequently interesting, especially in financial applications. The accuracy of the model's predictions is a normal concern that arises from time to time for applied econometricians. The accuracy of the model's predictions is sometimes a natural question for an applied econometrician.
The variation of the error terms and what causes them to be significant are the main concerns in many econometrics scenarios. This subject frequently arises in financial applications where the dependent variable is the return on an asset or portfolio, and the variation of the return indicates the amount of risk associated with those returns. Despite the fact that they are time series applications, heteroskedasticity is likely to be a problem. Even a basic examination of financial data indicates that certain time periods are riskier than others; that is, the anticipated value of the magnitude of error terms is bigger at some times than at others. Furthermore, these high-risk periods are not evenly distributed among quarterly or yearly data. The riskiness of financial returns, on the other hand, exhibits some autocorrelation. These concerns are addressed by the ARCH and GARCH models, which stand for autoregressive conditional heteroskedasticity and generalized autoregressive conditional heteroskedasticity, respectively. They are now commonly used to deal with time series heteroskedastic models. The purpose of these models is to give a volatility measure, similar to a standard deviation, that may be utilized in risk analysis, portfolio selection, and derivative pricing.

1.3.2 The Arch and Garch Models

Using financial notation, let's call the dependent variable r_t, which might represent the return on an asset or a portfolio. The variance h and the mean value m will be determined in relation to a previous data set. The mean value of r (that is, the anticipated value of r based on prior information) plus the standard deviation of r (that is, the square root of the variance) times the error term for the current period equals the return r in the present.

The econometric issue is to define how the data is utilized to forecast the mean and variation of the return based on previous data. While numerous specifications for the mean return have been developed and applied in attempts to estimate future returns, there were essentially no methodologies for the variance before the emergence of ARCH models. The rolling standard deviation was the most important descriptive tool. This is the standard deviation derived from a predetermined number of recent observations. This formulation is known as the first ARCH model because it assumes that the variance of tomorrow's return is an evenly weighted average of the squared residuals from the previous observations. The premise of equal weights appears unappealing, because current occurrences would seem to be more significant and hence should have larger weights. Engle (1982) created the ARCH model, which allowed these weights to be computed as parameters. As a result, the data was able to establish the optimal weights to apply in forecasting the variance using the model.

The GARCH parameterization developed by Bollerslev is a helpful extension of this model (1986). This model is similarly a weighted average of past squared residuals, but the weights are decreasing and never reach zero. It produces simple, parsimonious models that are straightforward to estimate, and it has been remarkably successful in forecasting conditional variances even in its most basic version. The most generally used GARCH specification claims that a weighted average of the long-run average variance, the variance forecast for this period, and the new information acquired by the most recent squared residual is the best predictor of the variance in the next period. This type of updating rule is known as Bayesian updating and is a simplified representation of adaptive or learning behavior.
We can define h_t to be the variance of the residuals of a regression of the type $r_t = m_t + \sqrt{h_t}\epsilon_t$. In particular, we assume the variance of ϵ to be 1. Then, the Garch model writes:

$$h_{t+1} = \omega + \alpha(r_t - m_t)^2 + \beta h_t = \omega + \alpha h_t {\epsilon_t}^2 + \beta h_t$$

Where α, β, and ω are the constants to be estimated. Updating, then, only requires knowing the previous forecast h and residual. In particular the weighs are $(1 - \alpha - \beta, \beta, \alpha)$, while the long-run variance is given by $\sqrt{\omega/(1 - \alpha - \beta)}$. The shown equation works only if $\alpha + \beta < 1$ and if $\alpha > 0, \beta > 0, \omega > 0$.
The GARCH model just presented is usually defined as GARCH(1,1) model. The first 1 of (1,1) refers to how many autoregressive lags appear in the equation (called ARCH terms), while the second one to how many moving average lags are specified (called GARCH terms).
Although this model is directly set up to forecast for just one period, it turns out that based on the one-period forecast, a two-period forecast can be made. Thus, the GARCH models are mean reverting and conditionally heteroskedastic, but have a constant unconditional variance.
To estimate the shown equation for the GARCH(1,1) when we only have information on r_t, is to use the maximum likelihood by substituting h_t for σ^2 in the normal likelihood and then maximizing with respect to the parameters. However, the procedure isn't all that mysterious. It is simple to compute the variance forecast for the second observation given any combination of parameters v, a, b, and a starting estimate for the variance of the first observation, which is commonly considered to be the observed variance of the residuals. The GARCH updating formula estimates the variance of the second observation using the weighted average of the unconditional variance, the squared residual for the first observation, and the beginning variance. This information is used to forecast the third variance, and so on.

2 Goal of the Project

Many economists and investors have fantasized about being able to forecast the Stock Market's next move, or volatility, in order to respond appropriately and earn a guaranteed profit. Isn't that something we'd all like? Unfortunately, due to the randomness of the market, this remains undiscovered. What we can do, though, is act based on the information we've accumulated over the years. Meaning that from the analysis of historical data we can model the state of tomorrows market. Because of its stability, we selected to investigate fluctuations in returns for the SP 500 market index, which is a popular gauge of the status of the US economy since only the stock prices (and their fluctuations) of 500 of the country's top companies are included. A return is defined as the difference between two index values separated by a specified time interval, in this example seven days. As a result, we'll be evaluating and comparing the accuracy of both

the ARMA model and the Arch and Garch model using historical data from the S&P 500 market index. In conclusion, the aim of the project is to test out the validity of these two models, using a combinations of the two so to predict future values and reduce the confidence interval of these predictions. As said, to do this we have taken price levels from the past (starting from 1994 and ending at the end of 2018), and we construct a prediction of last year's level so that we can compare the results of the model with the actual values.
Our study stops at the end of August of 2019. We have decided to proceed down this path due to the outbreak of COVID-19. This pandemic has affected consumer confidence, hence affecting the economy worldwide. This shock has caused an unpredictable anomaly, that would create bias in our results, hence we have chosen to exclude it. We are aware that it is impossible to predict the market due to its randomness, however we aim to study the accuracy of the ARMA and GARCH model.

3 Data Analysis

3.1 Estimating parameters of ARMA model

As stated above the ARMA model has two parameters; p and q The order of p, which affects the effect of previous values on current values, is determined by the number of significant delays in the PACF plot (figure 2). The order of q, which affects the effect of prior residuals on the current value, is determined by the number of significant delays in the ACF plot (figure 1).

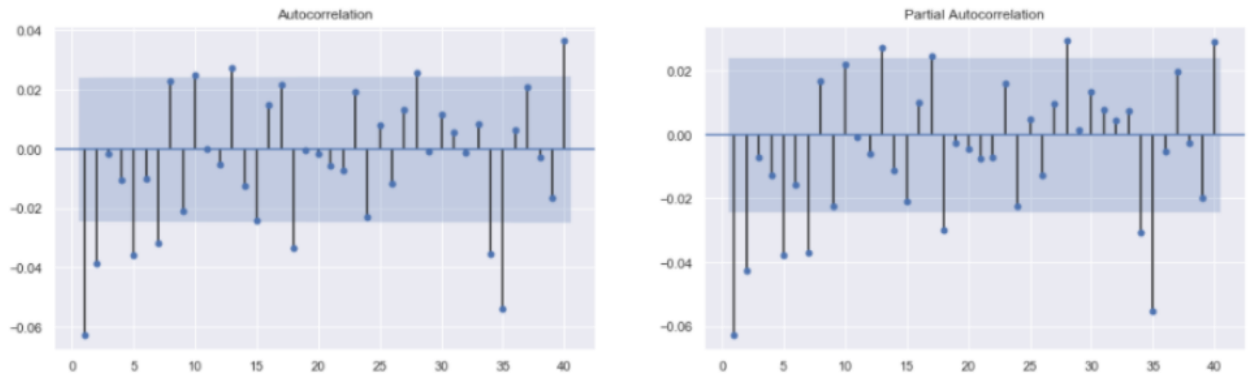


Figure 1 and 2: Plot of ACF and PACF.

It is evident from looking at the graphs that the first two lags are significant in both plots. The relevance level quickly slows off after these and then surges up again. For simplicity's sake let us set the parameters as follows:

1. q = 1 or q = 2
2. p = 1 or p = 2

3.2 ARMA Model Predictions and Confidence Interval

The Returns for the Test set are forecasted using the ARMA(1, 1) model. The confidence intervals are also generated using the same model.

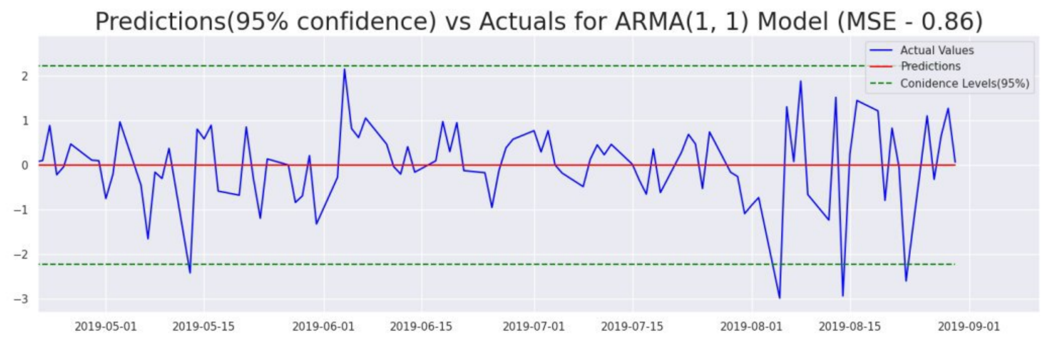


Figure 3: ARMA(1,1).

The get_forecast() method is used to create a forecasts object, which can then be used with the conf_int() function to calculate confidence intervals. The test set predictions are obtained using the predict() method. To calculate the accuracy of the predictions in relation to the actual returns in the test set, the RMSE (Root Mean Squared Error) metric is utilized. To visually confirm the model's correctness, the predictions and confidence intervals are plotted against the actual returns from the test set.

When looking at the plot (given in the image), the forecasts are sometimes spot on, and sometimes they are way off. The confidence intervals offer no insight into how the predictions will perform over the course of the test set's various time periods. The confidence intervals appear to be overly conservative in several circumstances, and the returns overrun the borders. The confidence intervals aren't quite cautious enough in other circumstances. To solve this problem, we will examine the model's residuals in the next part and attempt to forecast when the ARMA predictions will be off by a small margin and when they will be off by a significant margin.

3.3 ARMA Model Residuals

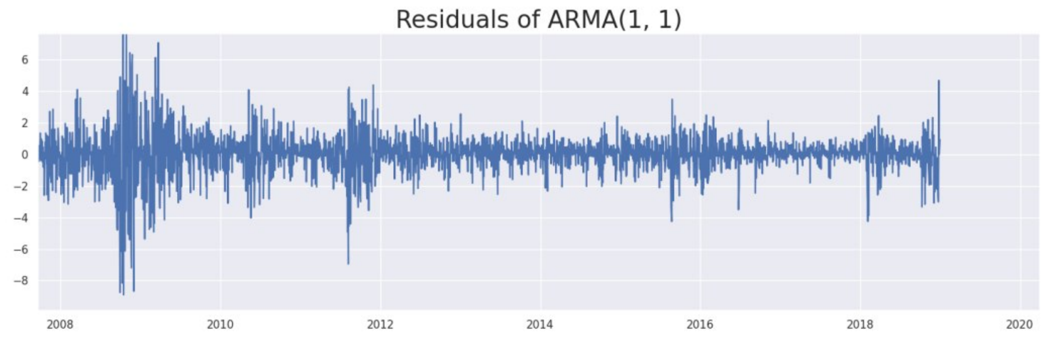


Figure 4: Residuals of the Arma model.

The fitted model's resid attribute can be used to obtain the residuals generated by the ARMA(1, 1) model. The residuals clearly reveal the phenomenon of volatility clustering when viewed in this way. This means that if the series has a lot of volatility (high variance) at one time step, it will have a lot of volatility in the next time step, and vice versa. This phenomena may be seen in the years around 2004 and 2008 (high volatility periods) as well as the years in between (low volatility periods).

The GARCH model can successfully represent series that demonstrate such volatility clustering (as seen in part 4 linked at the end). We'll start estimating the parameters needed to fit the GARCH model to the residuals of the ARMA(1, 1) model in the next section.

3.4 GARCH Parameter Estimation

The GARCH model comprises two parameters: GARCH and GARCH (p, q). The number of significant lags in the PACF graphic is used to estimate these parameters. The code for making such a plot is not displayed because it is extremely similar to the code for making the PACF plot above. The only changes that need to be made are in the data that has been supplied. The model_results.resid series will be passed in this scenario.

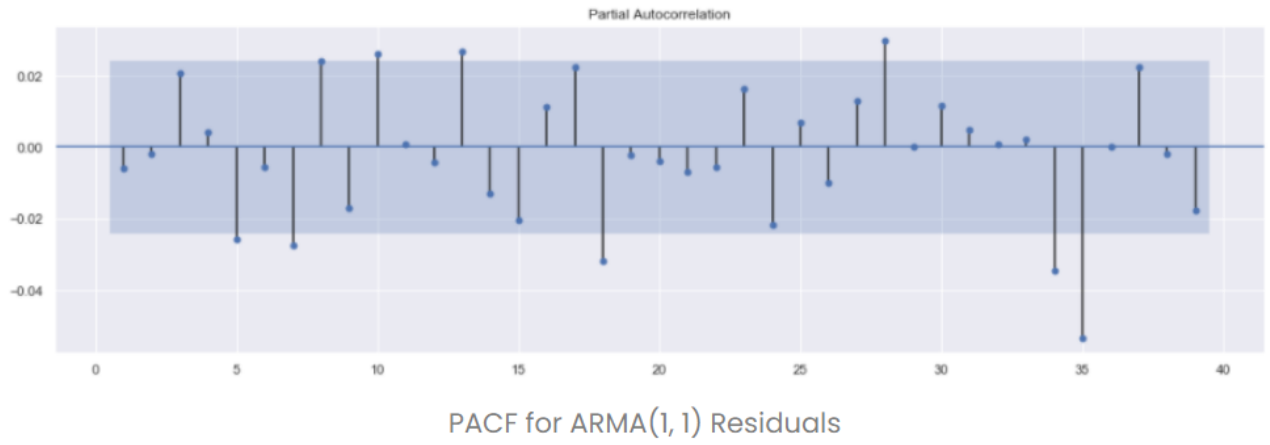


Figure 5: PACF.

There are no substantial lags in the residuals, as shown in the graph. As a result, we'll fit multiple GARCH models, such as GARCH(1, 1), GARCH(1, 2), GARCH(2, 1), GARCH(2, 2), and so on, until we find one with significant coefficients and the best accuracy.

3.5 GARCH Model on ARMA Residuals

The PACF figure, as described in the preceding section, provided little insight into the initial settings. Various models should be fitted in this scenario in order to identify one with significant coefficients and the best accuracy. This ensures that we have the simplest model with the highest accuracy possible.
In our case, after having tried to run the code with the various GARCH models, we observed that the GARCH(2, 2) satisfied this condition best.

A new dataframe is created before fitting the model. This dataframe contains all of the original dataset's time steps (before train-test split). The Residuals from the ARMA(1, 1) model occupy the training time steps. These are the ones that are utilized to train the GARCH model. The actual returns seen one time step prior are used to fill the testing periods. This essentially means that the model will forecast tomorrow's return volatility based on today's returns.
A GARCH model is defined using the arch_model() function in the arch library. To train the provided model, use the fit() function. The last_obs option is used to determine when the model should begin predicting. As illustrated in the image, the summary() function prints out the summary table. The summary table demonstrates that all of the model's coefficients are significant.

3.6 Predictions of ARMA+GARCH model

To predict the volatility of the residuals in the test set, we employ the GARCH(2, 2) model fitted on ARMA(1, 1) residuals in this section.

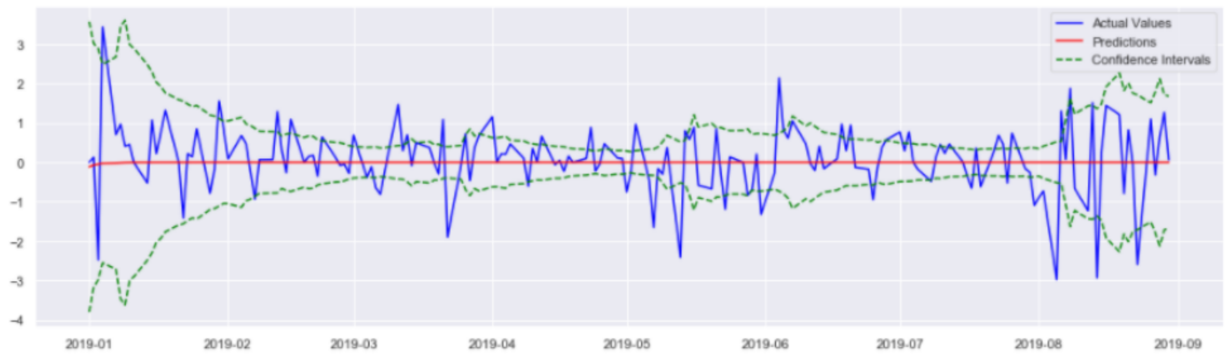


Figure 6: Final prediction of the S&P500 index on returns.

On the fitted model, resid_model_results, the forecast() technique is utilized. This generates an ARCH Model Forecast object with predictions for both the mean and volatility models. The residual_variance attribute is then used to get the volatility forecasts. The lower (and upper) confidence intervals of the model are calculated by subtracting (and adding) the volatility model predictions from the ARMA model predictions stored in the arma_predictions_df.
The confidence intervals (from GARCH(2, 2)) and predictions (from ARMA(1, 1) are then displayed against the actual S&P 500 Returns. When the returns are stable and the predictions are near to actual returns, the confidence intervals reflect this by being close, as shown in the graph. The same is true when the Returns are volatile and the projections are off by a large margin. The confidence intervals shift and become widely apart in this situation as well.
As we can see in the previous graph, our model predicts a constant 0% throughout the first nine months of 2019. Within the 95% confidence interval, our prediction is close to the actual values we have collected. If we have to reason in favor of our prediction, we should consider the nature of the data we have. We have considered the S&P500 index, whose value represents the first 500 hundred companies in the US. This means that this index is tendentiously stable, so large fluctuations are very unlikely. We can clearly see this, using the graph picturing the concentration of our values.

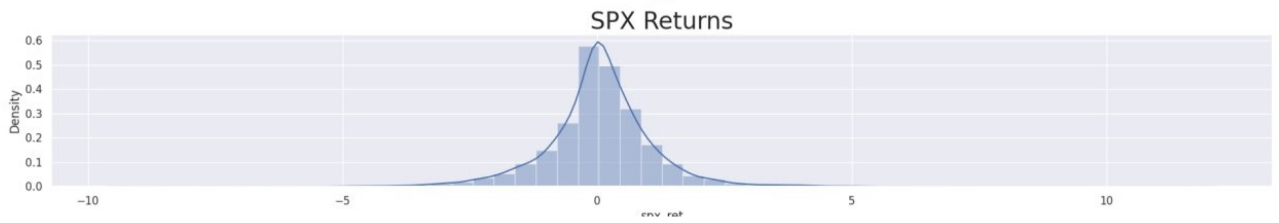


Figure 7: Density of dataset for returns.

In the same way, the next graph confirms what we have already explained. The percentage of returns is on average 0 throughout the entire period of study. Furthermore, we can also clearly see that the values are very close to 0. The only very noticeable exceptions (where we have large peaks in magnitude) are in 2001 and 2008. In 2001, the market crash followed the terrorist attacks and the event of 9/11, causing a $1.4 trillion loss in market value. The stock market crash of 2008 was a result of defaults on consolidated mortgage-backed securities. Subprime housing loans comprised most MBS. Banks offered these loans to almost everyone, even those who weren't creditworthy. When the housing market fell, many homeowners defaulted on their loans.

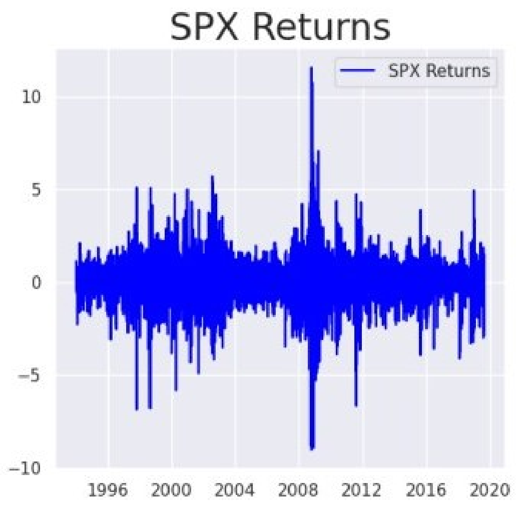


Figure 8: Returns of S&P500 from 1994 to 2019.

This market crashes are clearly visible on the trend of the price of this index as well, where in 2008 we have a fall from $1500 to about $700, which is more than 50%.

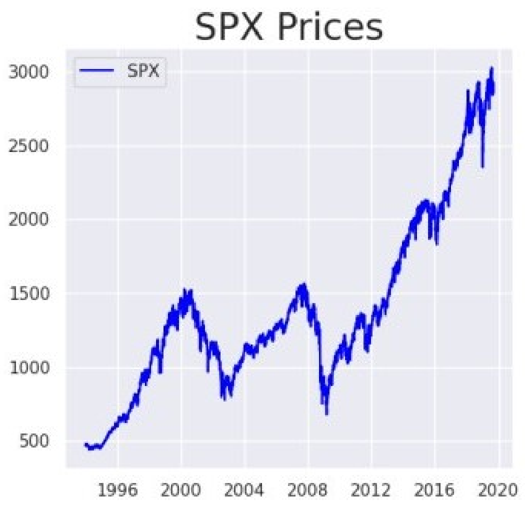


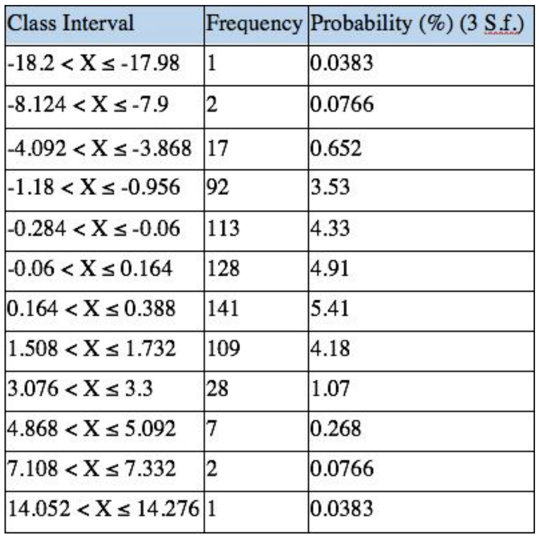
Figure 8: Price levels.

4 Conclusion

As a result, we can create significantly more meaningful forecasts by combining the ARMA and GARCH models. We may comprehend the ballpark figure for future returns as well as market stability by looking at the projections and confidence intervals.

5 Further Considerations

By dividing the frequency within a class interval with the total amount of samples in the data set. I have been able to calculate the probability of the occurrence of fluctuations within the interval, a few randomly chosen intervals are represented below in Table 1.1:



Class Interval	Frequency	Probability (%) (3 S.f.)
-18.2 < X ≤ -17.98	1	0.0383
-8.124 < X ≤ -7.9	2	0.0766
-4.092 < X ≤ -3.868	17	0.652
-1.18 < X ≤ -0.956	92	3.53
-0.284 < X ≤ -0.06	113	4.33
-0.06 < X ≤ 0.164	128	4.91
0.164 < X ≤ 0.388	141	5.41
1.508 < X ≤ 1.732	109	4.18
3.076 < X ≤ 3.3	28	1.07
4.868 < X ≤ 5.092	7	0.268
7.108 < X ≤ 7.332	2	0.0766
14.052 < X ≤ 14.276	1	0.0383

Table 1.1: Gaussian probability distribution of S&P500 market index fluctuations within interval groups.

This table demonstrates that, the closer the class interval is to the mean (0.162%) the larger the frequency and subsequently the probability. Consequently, it has been shown that in the S&P 500 index, smaller fluctuations are much more probable and frequent than larger ones. This is usually the case, but the specific index's stability helps amplify this pattern, because the index measures the stock performance of 500 large firms listed on stock exchanges in the United States, implying that the index's stability is guaranteed, therefore, for there to be a large fluctuation, a truly exceptional event must occur.

These large negative fluctuations are market crashes: -11.6% and -18.2% which happened in 2001 and 2008 respectively. Although it is clear these large fluctuations are rare, one might ask themselves how rare? And subsequently, what is the probability of these detrimental fluctuations occurring again?

Models such as the Gaussian model, which have been relied on and used the bell-curve, also known as the Gaussian model which represents probability with normal distribution: which is a type of continuous probability distribution for a real-valued random variable. The 'bell- curve' is a symmetrical graph, which is used to describe the probability distribution in a symmetrical manner adding and subtracting 3 standard deviations on either side of the arithmetic mean, where the concave quadratic function encompasses around 99.73% of the data within these first 3 standard deviations. Much like the Gaussian model, the ARMA+GARCH MODEL tends to underestimate the 'extreme' scenarios, due to its 'thin tailed' property.

To more accurately model the extremes, we can use the power law which follows a probability distribution that is "fat tailed", where, it can be perceived as similar to the Gaussian for smaller fluctuations but with pumped up tails enabling it to encompass and more accurately predict and even over predict the probability of larger variations. Which is crucial as economists shouldn't discredit these cases, as although they do rarely occur, their occurrence has incredibly significant implications on the market.

6 Appendix A

In this appendix, we insert the code we used to build the model and to construct all the graphs that are present in the file.

[4]:
```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
```

```python
import seaborn as sns
sns.set()

import yfinance

raw_data = yfinance.download (tickers = "^GSPC", start = "1994-01-07",
                              end = "2019-09-01", interval = "1d")

# Create a new dataframe with one column - "spx"
data = pd.DataFrame(columns = ["spx"])
# Copy closing prices of S&P 500 to this new column
data["spx"] = raw_data["Close"]
# Ensure that the dates are ordered in business week fashion (5 days a week)
data = data.asfreq("b")

print("Null values - ",data.spx.isnull().sum())

print("\nStatistical Description of the series - ")
print(data.describe())

data.spx = data.spx.fillna(method='ffill')
print("\nNull values - ",data.spx.isnull().sum())

# Calculating returns and volatility based on previous formulas
data["spx_ret"] = data.spx.pct_change(1).mul(100)
data["spx_vol"] = data.spx_ret.abs()



#part2
# Importing Dataset
data = pd.read_csv("data.csv")
data.Date = pd.to_datetime(data.Date)
data.set_index("Date", inplace = True)

# Displaying first 5 rows of "data" DataFrame
data.head()

# Setting the figure size
plt.rcParams["figure.figsize"] = (18, 5)

# Subdividing the figure into 3 figures stacked in 1 row
fig, ax = plt.subplots(1, 3)

# First Plot - S&P 500 prices/ against time
ax[0].plot(data.spx, color = "blue", label = "SPX")
ax[0].set_title("SPX Prices", size = 24)
ax[0].legend()

# Second Plot - S&P 500 returns against time
ax[1].plot(data.spx_ret, color = "blue", label = "SPX Returns")
ax[1].set_title("SPX Returns", size = 24)
ax[1].legend()
```

```python
# Third Plot - S&P 500 volatility against time
ax[2].plot(data.spx_vol, color = "blue", label = "SPX Volatility")
ax[2].set_title("SPX Volatility", size = 24)
ax[2].legend()

# Used to display the plot free from any additional text in Jupyter notebooks
plt.show()

train_df = data.loc[:"2018-12-31"]
test_df = data.loc["2019-01-01":]

print("Training Set Shape - ", train_df.shape)
print("Testing Set Shape - ", test_df.shape)

# Adding another column in train_df storing the "Year" of each observation
train_df["Year"] = train_df.index.year

# Setting the size of the figure
plt.rcParams["figure.figsize"] = 24, 21

# Defining 3 subplots
fig, axes = plt.subplots(3, 1)

# First Boxplot: Yearly S&P 500 Prices
train_df.boxplot(by ='Year', column =['spx'], ax = axes[0])
axes[0].set_title("SPX Prices", size = 24)

# Second Boxplot: Yearly S&P 500 Returns
train_df.boxplot(by ='Year', column =['spx_ret'], ax = axes[1])
axes[1].set_title("SPX Returns", size = 24)

# Third Boxplot: Yearly S&P 500 Volatility
train_df.boxplot(by ='Year', column =['spx_vol'], ax = axes[2])
axes[2].set_title("SPX Volatility", size = 24)

# Displaying plots
plt.show()

# Setting the figure size
plt.rcParams["figure.figsize"] = 24, 18

# Defining 3 subplots one below the other
fig, axes = plt.subplots(3, 1)

# Plotting the distributions in the respective subplots
sns.distplot(train_df.spx, ax = axes[0])
sns.distplot(train_df.spx_ret, ax = axes[1])
sns.distplot(train_df.spx_vol, ax = axes[2])

# Setting the title for each subplot
axes[0].set_title("SPX Prices", size = 24)
axes[1].set_title("SPX Returns", size = 24)
```

```
axes[2].set_title("SPX Volatility", size = 24)

# Displaying the plot
plt.show()

# Import the required package
from statsmodels.tsa.seasonal import seasonal_decompose

# Set Plot size
plt.rcParams["figure.figsize"] = 18, 20

# Call the seasonal_decompose method to decompose the data using the "additive" model
result = seasonal_decompose(train_df.spx, model='additive')

# Un-comment these lines for additive decomposition of S&P 500 Returns and Volatility␣
 ↪respectively
# result = seasonal_decompose(train_df.spx_ret[1:], model='additive')
# result = seasonal_decompose(train_df.spx_vol[1:], model='additive')

# Un-comment these lines for multiplicative decomposition of S&P 500 Prices, Returns␣
 ↪and Volatility respectively
# result = seasonal_decompose(train_df.spx, model='multiplicative')
# result = seasonal_decompose(train_df.spx_ret[1:], model='multiplicative')
# result = seasonal_decompose(train_df.spx_vol[1:], model='multiplicative')

# Plot the result
result.plot()

# Display the plot
plt.show()

result.seasonal[:20].plot(marker = "o")
plt.show()

#part3
train_df = data.loc[:"2018-12-31"]
test_df = data.loc["2019-01-01":]

print("Training Set Shape - ", train_df.shape)
print("Testing Set Shape - ", test_df.shape)

# Importing the necessary package
from statsmodels.tsa.stattools import adfuller

# ADF test in S&P 500 Returns
adfuller(train_df["spx_ret"][1:])

# Importing Required Package
import statsmodels.graphics.tsaplots as sgt
# Fixing plot size
plt.rcParams["figure.figsize"] = 18, 5

# Defining Subplots
```

```
fig, axes = plt.subplots(1, 2)

# Plotting ACF and PACF for S&P 500 Returns
sgt.plot_acf(train_df.spx_ret[1:], zero = False, lags = 40, ax = axes[0])
sgt.plot_pacf(train_df.spx_ret[1:], zero = False, lags = 40, ax = axes[1])

# Display the Plot
plt.show()

# MODEL FITTING
# Importing Required Package
from statsmodels.tsa.statespace.sarimax import SARIMAX

# Defining the Model
model = SARIMAX(train_df["spx_ret"][1:], order = (1, 0, 1))
# Fitting the Model
model_results = model.fit()

# Printing the model summary
print(model_results.summary())

# EVALUATING RESIDUALS
# Defining the figure size
plt.rcParams["figure.figsize"] = 18, 5

# Defining the subplots
fig, axes = plt.subplots(1, 2)

# ACF and PACF for residuals of ARIMA(1, 0, 1)
sgt.plot_acf(model_results.resid[1:], zero = False, lags = 40, ax = axes[0])
sgt.plot_pacf(model_results.resid[1:], zero = False, lags = 40, ax = axes[1])

# Displaying the plots
plt.show()

# MODEL FITTING
# Importing Required Package
from statsmodels.tsa.statespace.sarimax import SARIMAX
# Defining the Model
seas_model = SARIMAX(train_df["spx_ret"][1:], order = (1, 0, 1), seasonal_order = (1,␣
 ↪0, 1, 5))
# Fitting the Model
seas_model_results = seas_model.fit()

# Printing the model summary
print(seas_model_results.summary())


# EVALUATING RESIDUALS
# Defining the figure size
plt.rcParams["figure.figsize"] = 18, 5

# Defining the subplots
```

```
fig, axes = plt.subplots(1, 2)

# ACF and PACF for residuals of SARIMA(1, 0, 1)(1, 0, 1, 5)
sgt.plot_acf(seas_model_results.resid[1:], zero = False, lags = 40, ax = axes[0])
sgt.plot_pacf(seas_model_results.resid[1:], zero = False, lags = 40, ax = axes[1])

# Displaying the plots
plt.show()


# Importing Required Packages
from sklearn.metrics import mean_squared_error

# GENERATING FORECASTS AND CALCULATING ACCURACY
# Forecasts of ARIMA model
pred = model_results.predict(start = test_df.index[0], end = test_df.index[-1])
# Forecasts of SARIMA model
seas_pred = seas_model_results.predict(start = test_df.index[0], end = test_df.
 ↪index[-1])

# RMSE of ARIMA model
arima_rmse = np.sqrt(mean_squared_error(y_true = test_df["spx_ret"].values, y_pred =␣
 ↪pred.values))
# RMSE of SARIMA model
sarima_rmse = np.sqrt(mean_squared_error(y_true = test_df["spx_ret"].values, y_pred =␣
 ↪seas_pred.values))


# FORECAST vs ACTUALS PLOT
# Setting the size of the figure
plt.rcParams["figure.figsize"] = 18, 5

# Defining the subplots
fig, ax = plt.subplots(1, 2)

# Actuals vs Predictions for ARIMA(1, 0, 1)
ax[0].plot(test_df["spx_ret"], color = "blue", label = "Actuals")
ax[0].plot(pred, color = "red", label = "ARIMA(1, 0, 1) Predictions")
ax[0].set_title(f"ARIMA(1, 0, 1) Predictions (RMSE: {np.round(arima_rmse, 3)})", size␣
 ↪= 16)

# Actuals vs Predictions for SARIMA(1, 0, 1)(1, 0, 1, 5)
ax[1].plot(test_df["spx_ret"], color = "blue", label = "Actuals")
ax[1].plot(seas_pred, color = "red", label = "SARIMA(1, 0, 1)(1, 0, 1, 5) Predictions")
ax[1].set_title(f"SARIMA(1, 0, 1)(1, 0, 1, 5) Predictions (RMSE: {np.
 ↪round(sarima_rmse, 3)})", size = 16)

# Displaying the plots
plt.show()

# FORECASTING AND CONFIDENCE INTERVALS
# Generating Forecast object
forecasts = model_results.get_forecast(len(test_df.index))
```

```
# Generating confidence intervals
forecasts_df = forecasts.conf_int(alpha = 0.05)  # Confidence Interval of 95%
# Actual predictions
forecasts_df["Predictions"] = model_results.predict(start = test_df.index[0], end =␣
 ↪test_df.index[-1])

# Displaying first 5 rows of the forecasts_df
print(forecasts_df.head())

# RMSE of the forecasts
arima_rmse = np.sqrt(mean_squared_error(y_true = test_df["spx_ret"].values, y_pred =␣
 ↪forecasts_df["Predictions"].values))



# PLOTTING THE FORECASTS AND CONFIDENCE INTERVALS
# Setting the figure size
plt.rcParams["figure.figsize"] = 18, 5

# Actual values of the S&P 500 returns in the test set
plt.plot(test_df["spx_ret"], color = "blue", label = "Actual Values")

# Predictions from the model and confidence intervals
plt.plot(forecasts_df["Predictions"], color = "red", label = "Predictions")
plt.plot(forecasts_df["upper spx_ret"], color = "green", linestyle = "--", label =␣
 ↪"Conidence Levels(95%)")
plt.plot(forecasts_df["lower spx_ret"], color = "green", linestyle = "--")

# Title of the plot
plt.title(f"Predictions vs Actuals for ARIMA(1, 0, 1) Model (RMSE - {round(arima_rmse,␣
 ↪2)})", size = 24)

# Display the labels
plt.legend()
# Display the plot
plt.show()

#part3

print("Training Set Shape - ", train_df.shape)
print("Testing Set Shape - ", test_df.shape)

# Importing Required Package
import statsmodels.graphics.tsaplots as sgt

# Setting the figure size
plt.rcParams["figure.figsize"] = 12, 5

# PCF Plot for S&P 500 Returns
sgt.plot_pacf(train_df.spx_ret[1:], zero = False, lags = 40)

# Displaying the plot
plt.show()
```

```
# Importing required package
from arch import arch_model


 # Building Dataframe for fitting GARCH model
garch_df = pd.DataFrame(data["spx_ret"].shift(1).loc[data.index])
garch_df.at[train_df.index, "spx_ret"] = train_df["spx_ret"]

 # Instantating the model with the full dataset, parameters and specifying the model␣
 ↪to be a GARCH model
model = arch_model(garch_df["spx_ret"][1:], p = 2, q = 2, vol = "GARCH")
 # Fitting the model on all the data just before the date specified in "last_obs"␣
 ↪argument
model_results = model.fit(last_obs = test_df.index[0], update_freq = 5)
 # Printing the Summary table of the fitted model
model_results.summary()

 # FORECASTING
 # Building Predictions Data
predictions_df = test_df.copy()
 # Predictions
predictions_df["Predictions"] = model_results.forecast().residual_variance.loc[test_df.
 ↪index]

 # PLOTTING FORECASTS

 # Setting the Figure Size
plt.rcParams["figure.figsize"] = 18, 5

 # Plotting the Predictions and the test data
plt.plot(predictions_df["spx_vol"], label = "Actuals")
plt.plot(predictions_df["Predictions"], label = "Predictions")

 # Setting the Title
plt.title("Actuals vs Predictions for SPX Volatility", size = 24)

 # Displaying the labels and the plot respectively
plt.legend()
plt.show()

#part5

print("Training Set Shape - ", train_df.shape)
print("Testing Set Shape - ", test_df.shape)

# Importing Required Package
import statsmodels.graphics.tsaplots as sgt

# Fixing plot size
plt.rcParams["figure.figsize"] = 18, 5

# Defining Subplots
fig, axes = plt.subplots(1, 2)
```

```
# Plotting ACF and PACF for S&P 500 Returns
sgt.plot_acf(train_df.spx_ret[1:], zero = False, lags = 40, ax = axes[0])
sgt.plot_pacf(train_df.spx_ret[1:], zero = False, lags = 40, ax = axes[1])

# Display the Plot
plt.show()

# MODEL FITTING
# Importing Required Package
from statsmodels.tsa.statespace.sarimax import SARIMAX

# Defining the Model
model = SARIMAX(train_df["spx_ret"][1:], order = (1, 0, 1))
# Fitting the Model
model_results = model.fit()

# Printing the model summary
print(model_results.summary())

# FORECASTING
# Building Forecast Object to generate Confidence Intervals
arma_forecast = model_results.get_forecast(len(test_df.index))
arma_predictions_df = arma_forecast.conf_int(alpha = 0.05) # Confidence level of 95%
# Predictions
arma_predictions_df["Predictions"] = model_results.predict(start = test_df.index[0],␣
↪end = test_df.index[-1])

# RMSE for the Predictions
arma_rmse = np.sqrt(mean_squared_error(test_df["spx_ret"].values,␣
↪arma_predictions_df["Predictions"]))


# PLOTTING FORECASTS

# Set the Size of the figure
plt.rcParams["figure.figsize"] = 18, 5

# Plot the Actuals
plt.plot(test_df["spx_ret"], color = "blue", label = "Actual Values")

# Plot the Forecasts and the Confidence Intervals
plt.plot(arma_predictions_df["Predictions"][test_df.index], color = "red", label =␣
↪"Predictions")
plt.plot(arma_predictions_df["upper spx_ret"][test_df.index], color = "green",␣
↪linestyle = "--", label = "Conidence Levels(95%)")
plt.plot(arma_predictions_df["lower spx_ret"][test_df.index], color = "green",␣
↪linestyle = "--")

# Set the Title of the Plot
plt.title(f"Predictions(95% confidence) vs Actuals for ARMA(1, 1) Model (MSE -␣
↪{round(arma_rmse, 2)})", size = 24)
```

```
# Display the plot with appropriate labels
plt.legend()
plt.show()

# Set the figure size
plt.rcParams["figure.figsize"] = 18, 5

# Plotting residuals
plt.plot(model_results.resid, label = "Residuals")

# Setting Title
plt.title("Residuals of ARMA(1, 1)", size = 24)

# Display the plot
plt.show()

# Importing required package
from arch import arch_model

 # Building Residuals DataFrame
resid_df = data.copy()
resid_df["spx_ret_resid"] = resid_df["spx_ret"].shift(1).loc[resid_df.index]
resid_df.at[train_df.index[1]:train_df.index[-1], "spx_ret_resid"] = model_results.
 ↪resid

 # Defining GARCH(2, 2) model
resid_model = arch_model(resid_df["spx_ret_resid"][1:], p = 2, q = 2, vol = "GARCH")
 # Fitting (Training) the model
resid_model_results = resid_model.fit(last_obs = test_df.index[0], update_freq = 5)
 # Displaying the model summary
resid_model_results.summary()

 # FORECASTING AND CONFIDENCE INTERVALS
 # Forecasting volatility of test set
resid_forecasts = resid_model_results.forecast().residual_variance.loc[test_df.index].
 ↪values

 # New Confidence Intervals
arma_garch_predictions_df = arma_predictions_df.copy()
arma_garch_predictions_df["lower spx_ret"] = arma_garch_predictions_df["Predictions"]␣
 ↪- resid_forecasts.reshape(-1,)
arma_garch_predictions_df["upper spx_ret"] = arma_garch_predictions_df["Predictions"]␣
 ↪+ resid_forecasts.reshape(-1,)


 # PLOTTING THE FORECASTS AND CONFIDENCE INTERVALS

 # Setting the Figure size
plt.rcParams["figure.figsize"] = 18, 5

 # Plotting the Actual S&P 500 Returns
plt.plot(test_df["spx_ret"], color = "blue", label = "Actual Values")
```

```
 # Plot the Forecasted Returns from ARMA(1, 1)
plt.plot(arma_garch_predictions_df["Predictions"][test_df.index], color = "red", label␣
 ↪= "Predictions")
 # Plot the new confidence intervals generated by the GARCH model
plt.plot(arma_garch_predictions_df["lower spx_ret"][test_df.index], color = "green",␣
 ↪linestyle = "--", label = "Confidence Intervals")
plt.plot(arma_garch_predictions_df["upper spx_ret"][test_df.index], color = "green",␣
 ↪linestyle = "--")

 # Display the plot and the labels
plt.legend()
plt.plot()

#part6
print("Training Set Shape - ", train_df.shape)
print("Testing Set Shape - ", test_df.shape)

# SETTING THE PLOT SIZE AND SUBPLOTS
plt.rcParams["figure.figsize"] = 18, 3
fig, axes = plt.subplots(1, 3)


# ACF AND PACF PLOT FOR THE TRANSFORMED DATA
sgt.plot_acf(train_df["spx"].dropna(), zero = False, lags = 40, ax = axes[0])
axes[0].set_title("ACF", size = 24)
sgt.plot_pacf(train_df["spx"].dropna(), zero = False, lags = 40, ax = axes[1])
axes[1].set_title("PACF", size = 24)


# LINE PLOT OF THE TRANSFORMED DATA
axes[2].plot(train_df["spx"].dropna())
axes[2].set_title("S&P 500 Prices", size = 24)
plt.show()


# ADF TEST ON THE TRANSFORMED DATA
from statsmodels.tsa.stattools import adfuller
adfuller(train_df["spx"].dropna())

# LOG TRANSFORMATION
train_df["spx"] = np.log(train_df["spx"].values)
# DIFFERENCING AFTER LOG TRANSFORM
train_df["spx"] = train_df["spx"].diff(1)

# Importing Required Package
from statsmodels.tsa.statespace.sarimax import SARIMAX

# Model Definition
model = SARIMAX(train_df["spx_log_diff"].dropna(), order = (1, 0, 1))
# Model Training
model_fit = model.fit()
 # Summary of the model built
model_fit.summary()
```

```
 # Building a predictions dataframe for storing all prediction data
pred_df = pd.DataFrame(columns = ["spx", "spx_lag_1",
                                    "model_preds", "model_preds_lower",␣
 ↪"model_preds_upper",
                                    "model_preds_exp", "model_preds_lower_exp",␣
 ↪"model_preds_upper_exp",
                                    "spx_preds", "spx_preds_lower", "spx_preds_upper"],
                         index = data.index)




 # Storing the original series and one lagged version (y(t) and y(t-1))
pred_df["spx"] = data["spx"]
pred_df["spx_lag_1"] = pred_df["spx"].shift(1)

# # Predictions on transformed data over the full span of the dataset.
pred_df["model_preds"] = model_fit.predict(start = train_df.index[1], end = test_df.
 ↪index[-1])

 # Getting Confidence Intervals for the transformed predictions on test set
forecast = model_fit.get_forecast(len(test_df.index))
forecast_df = forecast.conf_int(alpha = 0.05) # Confidence level of 95%

pred_df.at[test_df.index, "model_preds_lower"] = forecast_df["lower spx_log_diff"]
pred_df.at[test_df.index, "model_preds_upper"] = forecast_df["upper spx_log_diff"]




 # Taking Exponent to invert logarithmic effect - exp(y_new(t))
pred_df["model_preds_exp"] = np.exp(pred_df["model_preds"].values)
pred_df["model_preds_lower_exp"] = np.exp(list(pred_df["model_preds_lower"].values))
pred_df["model_preds_upper_exp"] = np.exp(list(pred_df["model_preds_upper"].values))

 # Multiplying with past lags to get the forecast and the confidence intervals -␣
 ↪y(t-1) * exp(y_new(t))
pred_df["spx_preds"] = pred_df["model_preds_exp"] * pred_df["spx_lag_1"]
pred_df.at[test_df.index, "spx_preds_lower"] = pred_df.loc[test_df.
 ↪index]["model_preds_lower_exp"] * pred_df.loc[test_df.index]["spx_lag_1"]
pred_df.at[test_df.index, "spx_preds_upper"] = pred_df.loc[test_df.
 ↪index]["model_preds_upper_exp"] * pred_df.loc[test_df.index]["spx_lag_1"]

 # RMSE Metric Calculation
arma_rmse = np.round(np.sqrt(
                     mean_squared_error(y_true = pred_df.loc[test_df.index, "spx"].
 ↪values,
                                        y_pred = pred_df.loc[test_df.index,␣
 ↪"spx_preds"].values)
                 ), 2)
 # Plotting Predictions

 # Setting Figure Size
plt.rcParams["figure.figsize"] = 18, 5

 # Plottting Actual Test Set Values
plt.plot(pred_df.loc[test_df.index, "spx"], color = "blue", label = "Actual Values")

 # Plotting Predictions and Confidence Intervals
plt.plot(pred_df.loc[test_df.index, "spx_preds"], color = "red", label = "Predictied␣
 ↪Values")
plt.plot(pred_df.loc[test_df.index, "spx_preds_lower"], color = "green", linestyle =␣
 ↪"--", label = "Confidence Intervals")
plt.plot(pred_df.loc[test_df.index, "spx_preds_upper"], color = "green", linestyle =␣
 ↪"--")

 # Setting the Title
plt.title(f"ARMA(1, 1) Predictions vs Actual S&P 500 Prices || RMSE = {arma_rmse}",␣
 ↪size = 24)

 # Displaying the labels and plot
plt.legend()
plt.show()
```