PROJECT REPORT

"COMPARING MACHINE LEARNING AND DEEP LEARNING MODELS TO FORECAST THE STOCK PRICE OF FAST GROWING COMPANIES IN INDIA"

Submitted towards the partial fulfillment of the criteria for award of Post Graduate Program in Data Analytics by Imarticus

Submitted By:

*VENUJAYAKANTH. B (*IL021097*)*

Course and Batch: *PGA-04-MARCH-2021*

ABSTRACT

Stock Market is a platform where millions of investors invest their money with an indentation to get better returns in future. Since changes in stock price are inevitable, it is significant to forecast its price to have a clear decision in investing one's own money in any company. Stakeholders and Investors are more likely to invest in top fast growing companies in India, so for forecasting purpose Top three fast growing companies Reliance, Future Retail limited and Avenue Supermarket was taken into consideration. Two famous Machine learning models (ARIMA and GARCH) and one Deep learning models (RNN) where taken for predicting the Stock prices. Root Mean Square Error was taken for model accuracy purpose. On Comparing the above three models on the stock price of Reliance, Future Retails and Avenue Supermarket, ARIMA model outperformed well in most of the cases on testing dataset.

ACKNOWLEDGEMENTS

We are using this opportunity to express our gratitude to everyone who supported us throughout the course of this group project. We are thankful for their aspiring guidance, invaluably constructive criticism and friendly advice during the project work. We are sincerely grateful to them for sharing their truthful and illuminating views on a number of issues related to the project.

Further, we were fortunate to have great teachers who readily shared their immense knowledge in data analytics and guided us in a manner that the outcome resulted in enhancing our data skills. We wish to thank, all the faculties, as this project utilized knowledge gained from every course that formed the PGA program.

We certify that the work done by us for conceptualizing and completing this project is original and authentic.

Date: March, 2021 Venujayakanth B.

Place: Coimbatore

CERTIFICATE OF COMPLETION

I hereby certify that the project titled ***"COMPARING MACHINE LEARNING AND DEEP LEARNING MODELS TO FORECAST THE STOCK PRICE OF FAST GROWING COMPANIES IN INDIA"*** was undertaken and completed under my supervision by Venujayakanth B from the batch of PGA-04 (March 2021)

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1. Title & Objective of the study

The Capstone project I worked upon was **"Comparing Machine Learning and Deep Learning models to forecast the stock price of fast growing companies in India".** This project comes under Finance Domain. In this project I considered the top three fast growing retail companies such as Reliance, Future Retail limited, and Avenue Supermarkets. It assists the stakeholders/policy makers to make decision in investment process to get better returns in future. The objective of this study is as follows

1. To forecast the stock price of retail companies by using **ARIMA** model.
2. To forecast the stock price of retail companies by using **AR-GARCH** model
3. To forecast the stock price of retail companies by using **RNN**
4. To ascertain the better model to forecast the retail companies.

1.1. Need of the Study

Stock market is highly uncertain, it is necessary to forecast the stock price of big companies to make long term investment without risk. Many machine learning and deep learning models are available in the literature to forecast price, most famous one is ARIMA, since stock price are highly volatile in nature so GARCH model is also taken into consideration. Neural networks outperform well in most of the situation so it is also considered in stock market prediction. Thus for our problem ARIMA, ARMA-GARCH and RNN will be considered.

1.2. Data Sources

The Data for above study was taken from Yahoo Finance (https://in.finance.yahoo.com). For analytical purpose last five year data was taken.

1.3. Tools & Techniques

- Machine learning models – ARIMA, AR-GARCH model.
- Deep learning models – Recurrent Neural Network (RNN).
- Python programming language was used for building models.
- Jupyter Notebook and Google Colab platform was used for programming.

CHAPTER 2: DATA CLEANING AND PREPROCESSING

The Data for analysis as taken from Yahoo finance official website, last five years daily data as taken into consideration for analysis. Before staring analysis it is significant to preprocess the data, clean the data for efficient analysis. The steps involved in Data cleaning and Pre-processing are as follows:

2.1. Data Extraction:

The secondary data collected from yahoo finance is as follows

```
In [2]: Data = pd.read_csv('DMART.BO.csv',index_col=0)
        Data
```

Out[2]:

Date	Open	High	Low	Close	Adj Close	Volume
2017-03-21	604.400024	650.000000	558.750000	640.750000	640.750000	14566653.0
2017-03-22	644.000000	657.500000	625.299988	639.599976	639.599976	3073734.0
2017-03-23	639.400024	648.000000	616.500000	634.150024	634.150024	1163968.0
2017-03-24	637.000000	640.000000	612.799988	616.000000	616.000000	1202009.0
2017-03-27	615.000000	623.000000	603.000000	613.700012	613.700012	876105.0
...	...	...	...	...	...	...
2021-02-26	3086.600098	3103.949951	2950.000000	2993.649902	2993.649902	47641.0
2021-03-01	3004.000000	3019.000000	2897.399902	2908.399902	2908.399902	28831.0
2021-03-02	2947.000000	3106.350098	2887.050049	3077.399902	3077.399902	29373.0
2021-03-03	3080.000000	3177.399902	3080.000000	3168.899902	3168.899902	50644.0

The Comma Separated values (CSV) file contains Daily stock open price (Day Starting price), Peak, Low, Close and Adjusted Close Price and also the volume. A stock price for a particular day is indicated by its closing price so closing stock price was considered for analytical purpose. The column containing Closing stock price and Date is extracted as Follows:

```
In [3]: Data1 = Data['Close']
        Data1
```

```
Out[3]: Date
        2017-03-21     640.750000
        2017-03-22     639.599976
        2017-03-23     634.150024
        2017-03-24     616.000000
        2017-03-27     613.700012
                          ...
        2021-02-26    2993.649902
        2021-03-01    2908.399902
        2021-03-02    3077.399902
        2021-03-03    3168.899902
        2021-03-04    3180.300049
        Name: Close, Length: 979, dtype: float64
```

2.2. Removing Null Values

Null values in the dataset is identified and deleted as follows:

```
In [5]: Data2.isnull().sum()

Out[5]: 2

In [6]: Data3 = Data2.dropna()
        Data3

Out[6]: Date
        2017-03-21     640.750000
        2017-03-22     639.599976
        2017-03-23     634.150024
        2017-03-24     616.000000
        2017-03-27     613.700012
                          ...
        2021-02-26    2993.649902
        2021-03-01    2908.399902
        2021-03-02    3077.399902
        2021-03-03    3168.899902
        2021-03-04    3180.300049
        Name: Close, Length: 977, dtype: float64
```

2.3. Data Sorting and Pattern Identification

Though the data is collected in chronological order it is necessary to sort the data in ascending order (Recent Date at the end) for model building and forecasting purpose. The Python code to sort the data is as follows:

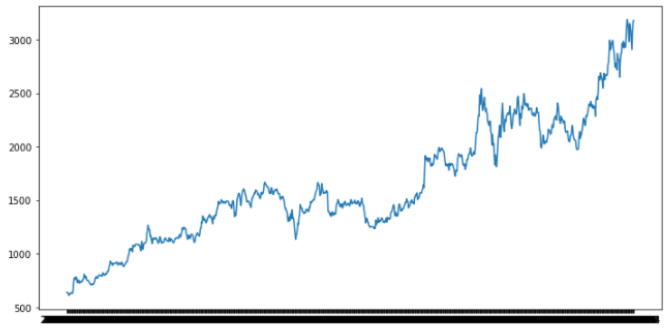
```
In [4]: Data2 = Data1.sort_index()
        Data2

Out[4]: Date
        2017-03-21     640.750000
        2017-03-22     639.599976
        2017-03-23     634.150024
        2017-03-24     616.000000
        2017-03-27     613.700012
                          ...
        2021-02-26    2993.649902
        2021-03-01    2908.399902
        2021-03-02    3077.399902
        2021-03-03    3168.899902
        2021-03-04    3180.300049
        Name: Close, Length: 979, dtype: float64
```

Before starting analysis, it is significant to identify the pattern in the dataset i.e. to estimate trend in last five years. The trend of Avenue Supermarket Stock is given as follows,

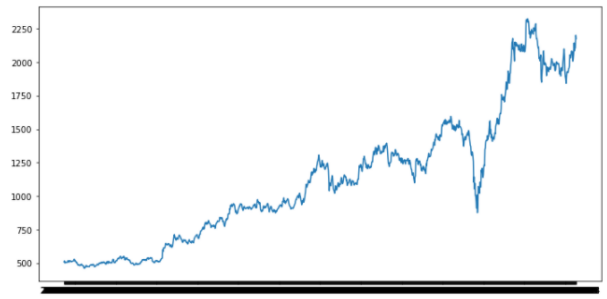
```
In [7]: plt.figure(figsize=(12,6))
        plt.plot(Data3)

Out[7]: [<matplotlib.lines.Line2D at 0x97bbad8>]
```



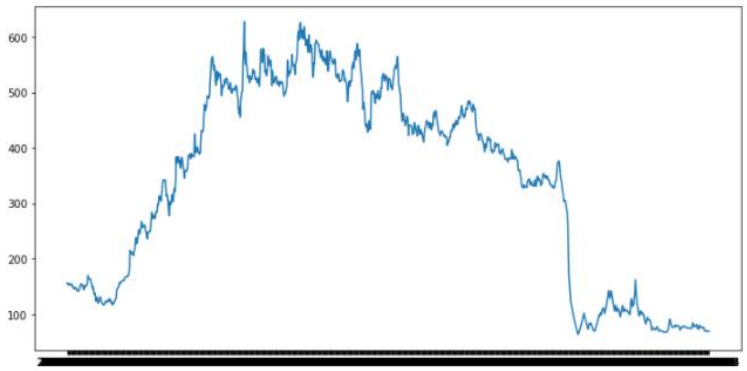
From the above plot it is easily predictable that the stock price of Avenue Supermarkets increased drastically in last five years, though there may be a short term fluctuation but in long run it will give better returns to the investors. The trend of Reliance Company Stock price is given as follows:

```
In [7]: plt.figure(figsize=(12,6))
        plt.plot(Data3)
Out[7]: [<matplotlib.lines.Line2D at 0xa48fb38>]
```



The stock price of this market is also exhibited a good growth in last five years, a sudden recursion had happened during the corona period but the company as improved its stock and moving at faster rate. The trend of Future Retail Private Limited is given as follows:

```
In [7]: plt.figure(figsize=(12,6))
        plt.plot(Data3)
Out[7]: [<matplotlib.lines.Line2D at 0x930db38>]
```



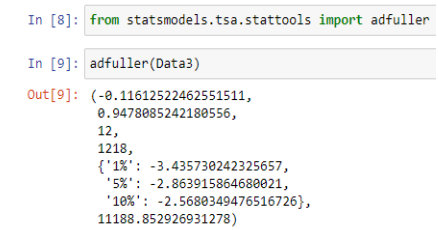
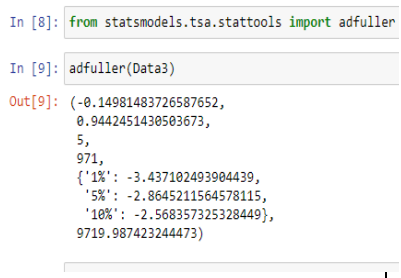
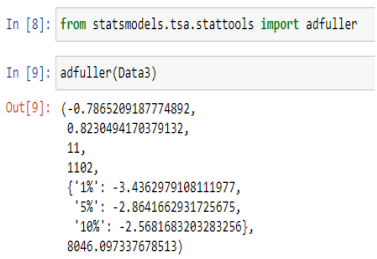
This company had good growth in the fast but recently the stock price got reduced, it makes the investors in little risk i.e. the stock may go beyond the margin level.

2.4. Checking Stationary

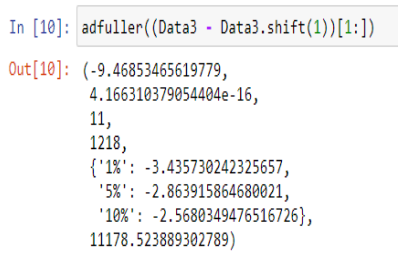
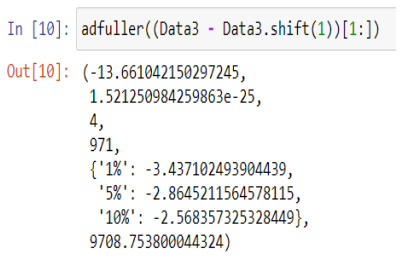
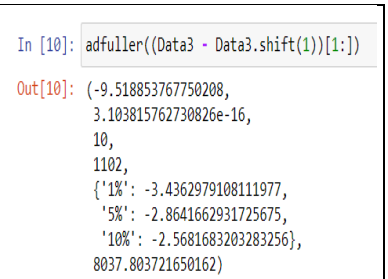
In most of the Time series data the mean and variance changes over time, this pose a problem in forecasting. If these statistic is not constant, while forecasting predicted value may either explode or degenerate leading to give an erroneous results. It is always a mandatory procedure to check stationary of time series data available in the literature. The most commonly used test is Augmented Dickey Fuller Test (ADF), it identifies the stationary

of a series by observing the presence of Unit Root (Non-Stationary) or absence of Unit Root (Stationary). If the p-value of ADF test statistic is greater than level of significance (5 or 1 per cent) then the series will be Non-stationary and vice-versa.

The original time series data collected from Internet is called as Levels, if the given series is non-stationary at levels it may be stationary at first difference or second difference, most of the time series attains stationary at first difference itself. The ADF test run on the Reliance, Avenue Super markets and Future Retails are given as follow, the first value is test statistic, second value is significance level of the test statistic, third is number of lags.

		
P-value of ADF test statistic is 0.947 (> 0.05)	P-value of ADF test statistic is 0.944 (> 0.05)	P-value of ADF test statistic is 0.82 (> 0.05)
RELAINCE	**AVENUE SUPERMARKETS**	**FUTURE RETAILS LTD**
ADF TEST ON LEVELS OF THREE COMPANIES		

Since for all the three stock data ADF test statistic is non-significant at levels so first difference is taken on all the three series and again ADF test is run which is given as follows:

		
P-value of ADF test statistic is 4.16e-16 (< 0.05)	P-value of ADF test statistic is 1.52e-25 (< 0.05)	P-value of ADF test statistic is 3.10E-16 (<0.05)
RELAINCE	**AVENUE SUPERMARKETS**	**FUTURE RETAILS LTD**
ADF TEST ON FIRST DIFFERNCE OF THREE COMPANIES		

On first differencing all the three series become stationary, so for model building purpose only the first difference was taken.

CHAPTER 3: MODEL BUILDING AND TESTING

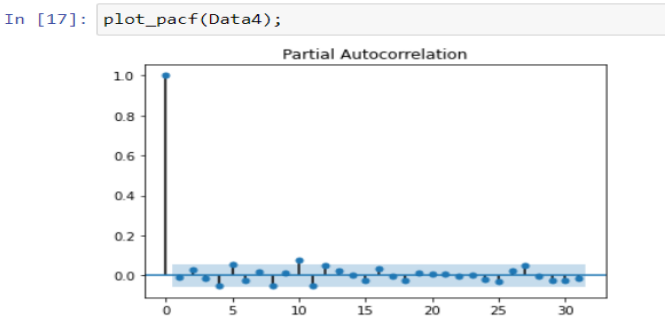
Three models ARIMA, ARMA-GARCH and RNN were selected to build on the Stock price, the first two are machine learning models and RNN is a Deep learning model. First difference data was used to build machine leaning models.

3.1. ARIMA model

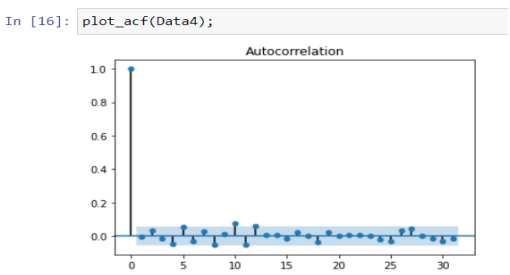
ARIMA (Autoregressive Integrated Moving Average) is a famous and efficient model available in the time series literature, the parameters involved in this model are p, d and q, where p – Number of AR terms, q – Number of moving average terms and d – Number of times a series is differenced to attain stationary. Parameter 'p' is identified with the help of PACF – Partial Auto Correlation function and 'q' is identified with the help of ACF – Auto Correlation Function.

3.1.1. Fitting ARIMA model for Reliance Stock Price

The parameter 'p' in ARIMA refers to number of Autoregressive term which means how much a current lags depends on previous lags for example 't' but neglecting the effect of remaining t-1 lags, this is identified by plotting PACF for specific Lag period. The PACF for Reliance Stock price is as follows:



The parameter 'q' in ARIMA refers to number of moving average term which means how much a current lags depends on previous lags for example 't' but including the effect of remaining t-1 lags, this is identified by plotting ACF for specific Lag period. The ACF for Reliance Stock price is as follows:



The most of the lags in ACF and PACF are within the Standard error limits, so for this model ARIMA (0, 1, and 0) would be appropriate. On training different parameters of ARIMA (1,1,1),ARIMA(1,1,0), ARIMA(0,1,1). The model with parameter p = 0, d=1and q=1 is found to be better with significant coefficients. The fitted parameters are as follows:

```
In [38]: modelfit = model.fit()
         modelfit.summary()
```

Out[38]:

ARIMA Model Results

Dep. Variable:	D.Close	**No. Observations:**	983
Model:	ARIMA(0, 1, 1)	**Log Likelihood**	-4179.014
Method:	css-mle	**S.D. of innovations**	16.926
Date:	Sun, 21 Mar 2021	**AIC**	8364.027
Time:	19:42:09	**BIC**	8378.699
Sample:	1	**HQIC**	8369.608

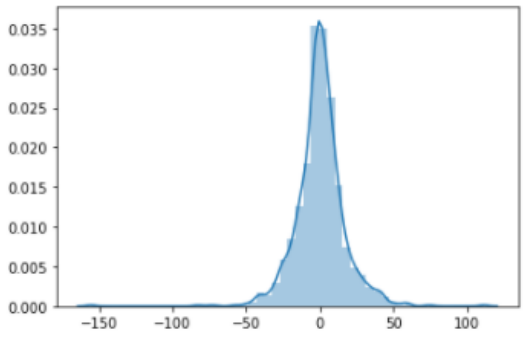
	coef	std err	z	P>|z|	[0.025	0.975]
const	-0.0016	0.002	-0.853	0.394	-0.005	0.002
ma.L1.D.Close	-1.0000	0.004	-226.043	0.000	-1.009	-0.991

Roots

	Real	Imaginary	Modulus	Frequency
MA.1	1.0000	+0.0000j	1.0000	0.0000

The moving average parameter is found to be significant and characteristic roots are less than one. The distribution of the residual of the fitted model was Normal as follows:

```
In [40]: sb.distplot(modelfit.resid)
Out[40]: <matplotlib.axes._subplots.AxesSubplot at 0x79360d0>
```



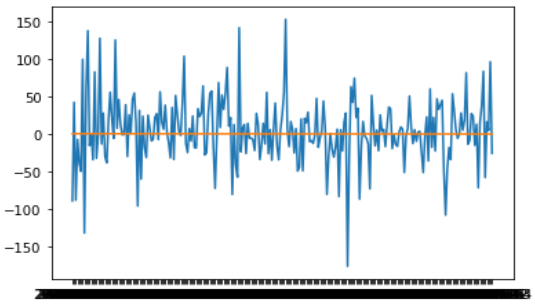
The root mean square error of the fitted model is 43.73 which are pretty good for forecasting.

```
In [28]: from sklearn.metrics import mean_squared_error

In [43]: math.sqrt(mean_squared_error(Test,modelfit.forecast(steps=245)[0]))
Out[43]: 43.73663001681425
```

The forecasted (RED) and actual value (BLUE) is plotted as follows which states that in long run the forecast value approaches to a mean value i.e. constant.

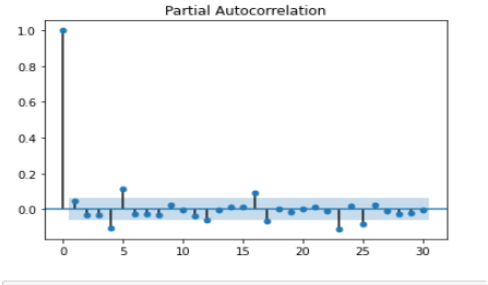
```
In [44]: plt.plot(Test)
         plt.plot(modelfit.forecast(steps=245)[0])
Out[44]: [<matplotlib.lines.Line2D at 0xd30b568>]
```



3.1.2. Fitting ARIMA model for Avenue Supermarkets

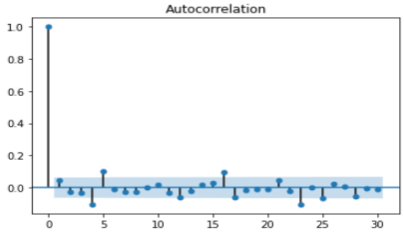
The parameter ‘p’ in ARIMA refers to number of Autoregressive term which means how much a current lags depends on previous lags for example ‘t’ but neglecting the effect of remaining t-1 lags, this is identified by plotting PACF for specific Lag period. The PACF for Avenue Supermarket Stock price is as follows:

```
In [17]: plot_pacf(Data4);
```



The parameter ‘q’ in ARIMA refers to number of moving average term which means how much a current lags depends on previous lags for example ‘t’ but including the effect of remaining t-1 lags, this is identified by plotting ACF for specific Lag period. The ACF for Avenue Supermarket Stock price is as follows:

```
In [16]: plot_acf(Data4);
```



The most of the lags in ACF and PACF are within the Standard error limits, so for this model ARIMA (0, 1, 0) would be appropriate. On training different parameters of ARIMA (1,1,1),ARIMA(1,1,0), ARIMA(0,1,1). The model with parameter p = 0, d=1and q=1 is found to be better with significant coefficients. The fitted parameters are as follows:

```
In [24]: modelfit = model.fit()
         modelfit.summary()
```

Out[24]:

ARIMA Model Results

Dep. Variable:	D.Close	**No. Observations:**	779
Model:	ARIMA(0, 1, 1)	**Log Likelihood**	-3874.258
Method:	css-mle	**S.D. of innovations**	34.819
Date:	Thu, 18 Mar 2021	**AIC**	7754.515
Time:	19:56:18	**BIC**	7768.489
Sample:	1	**HQIC**	7759.890

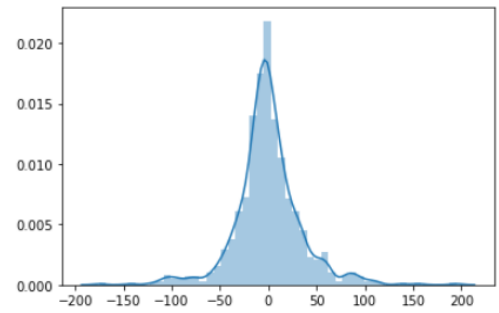
	coef	std err	z	P>|z|	[0.025	0.975]
const	-0.0003	0.006	-0.055	0.956	-0.011	0.011
ma.L1.D.Close	-1.0000	0.004	-271.191	0.000	-1.007	-0.993

Roots

	Real	Imaginary	Modulus	Frequency
MA.1	1.0000	+0.0000j	1.0000	0.0000

The moving average parameter is found to be significant and characteristic roots are less than one. The distribution of the residual of the fitted model was Normal as follows:

```
In [26]: sb.distplot(modelfit.resid)
Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0xced550>
```



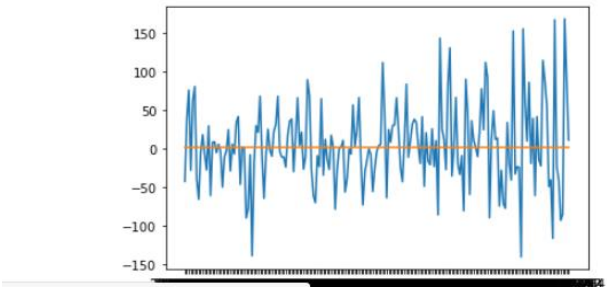
The root mean square error of the fitted model is 53.79 which are pretty good for forecasting.

```
In [28]: from sklearn.metrics import mean_squared_error

In [51]: math.sqrt(mean_squared_error(Test,modelfit.forecast(steps=195)[0]))
Out[51]: 53.79338668715609
```

The forecasted (RED) and actual value (BLUE) is plotted as follows which states that in long run the forecast value approaches to a mean value i.e. constant.

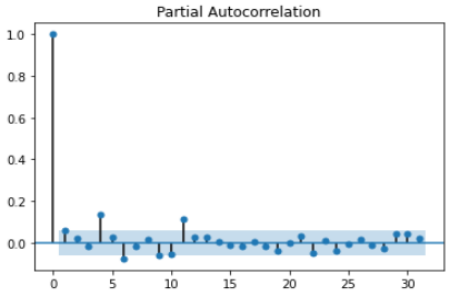
```
In [30]: plt.plot(Test)
         plt.plot(modelfit.forecast(steps=195)[0])
Out[30]: [<matplotlib.lines.Line2D at 0xc932c88>]
```



3.1.3. Fitting ARIMA model for Future Retails Limited

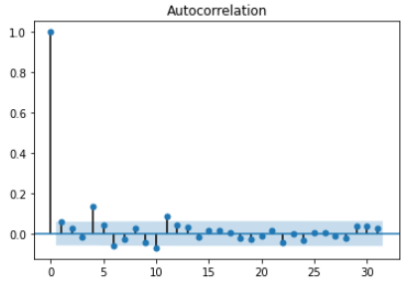
The parameter ‘p’ in ARIMA refers to number of Autoregressive term which means how much a current lags depends on previous lags for example ‘t’ but neglecting the effect of remaining t-1 lags, this is identified by plotting PACF for specific Lag period. The PACF for Future Retail Stock price is as follows:

```
In [17]: plot_pacf(Data4);
```



The parameter ‘q’ in ARIMA refers to number of moving average term which means how much a current lags depends on previous lags for example ‘t’ but including the effect of remaining t-1 lags, this is identified by plotting ACF for specific Lag period. The ACF for Future Retail Stock price is as follows:

```
In [16]: plot_acf(Data4);
```



The fourth lag of ACF and PACF are outside the Standard error limits, so for this model ARIMA (4, 1, 4) would be appropriate. On training different parameters of ARIMA from p=1 to 4 and q=1 to 4. The model with parameter p = 4, d=1and q=4 was found to be better with significant coefficients. The fitted parameters are as follows:

```
In [36]: modelfit = model.fit()
         modelfit.summary()
```

Out[36]:

ARIMA Model Results

Dep. Variable:	D.Close	**No. Observations:**	891
Model:	ARIMA(4, 1, 4)	**Log Likelihood**	-3350.638
Method:	css-mle	**S.D. of innovations**	10.359
Date:	Sun, 21 Mar 2021	**AIC**	6721.275
Time:	19:50:28	**BIC**	6769.199
Sample:	1	**HQIC**	6739.591

	coef	std err	z	P>|z|	[0.025	0.975]
const	-0.0042	0.002	-2.634	0.008	-0.007	-0.001
ar.L1.D.Close	0.0541	0.182	0.297	0.766	-0.303	0.411
ar.L2.D.Close	-0.4772	0.116	-4.114	0.000	-0.705	-0.250
ar.L3.D.Close	0.2618	0.166	1.576	0.115	-0.064	0.587
ar.L4.D.Close	0.1313	0.038	3.483	0.000	0.057	0.205
ma.L1.D.Close	-1.0096	0.183	-5.527	0.000	-1.368	-0.652
ma.L2.D.Close	0.5220	0.167	3.128	0.002	0.195	0.849
ma.L3.D.Close	-0.7998	0.143	-5.583	0.000	-1.081	-0.519
ma.L4.D.Close	0.2874	0.172	1.674	0.094	-0.049	0.624

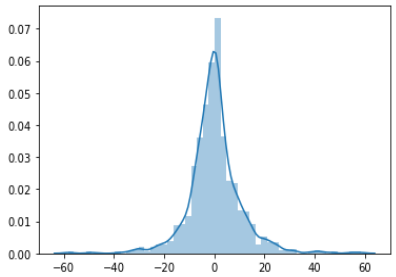
Roots

	Real	Imaginary	Modulus	Frequency
AR.1	-0.1674	-1.1584j	1.1705	-0.2728
AR.2	-0.1674	+1.1584j	1.1705	0.2728
AR.3	1.6700	-0.0000j	1.6700	-0.0000
AR.4	-3.3296	-0.0000j	3.3296	-0.5000
MA.1	-0.2996	-1.1709j	1.2086	-0.2899
MA.2	-0.2996	+1.1709j	1.2086	0.2899
MA.3	1.0000	-0.0000j	1.0000	-0.0000
MA.4	2.3822	-0.0000j	2.3822	-0.0000

The most of the autoregressive and moving average parameters was found to be significant and characteristic roots are less than one. The distribution of the residual of the fitted model was Normal as follows:

```
In [38]: sb.distplot(modelfit.resid)
```

```
Out[38]: <matplotlib.axes._subplots.AxesSubplot at 0xc4d54c0>
```



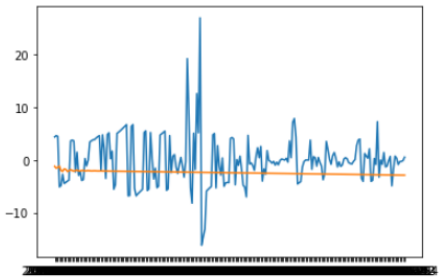
The root mean square error of the fitted model is 5.19 which are good for forecasting.

```
In [40]: from sklearn.metrics import mean_squared_error

In [43]: math.sqrt(mean_squared_error(Test,modelfit.forecast(steps=220)[0]))
Out[43]: 5.198406022374731
```

The forecasted (RED) and actual value (BLUE) is plotted as follows which states that in long run the forecast value approaches to a mean value i.e. constant.

```
In [44]: plt.plot(Test)
         plt.plot(modelfit.forecast(steps=220)[0])
Out[44]: [<matplotlib.lines.Line2D at 0xc56ac28>]
```



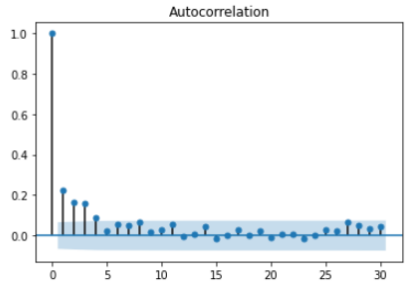
3.2. GARCH model

Stock price data is subjected to volatility i.e. the error variance is not constant over time. In Statistical terminology heterocedasticity is a major problem in stock price data so it is necessary to model the heterocedasticity of the data set. GARCH (Generalized Autoregressive Conditional Heterocedasticity) model the error of the fitted model, this model will capture the non-constant error variance. Before fitting GARCH model, ARMA model is fitted to the dataset and with the help of residuals obtained tat ARMA model, GARCH model is fitted.

3.2.1. Fitting GARCH model for Reliance Stock Price

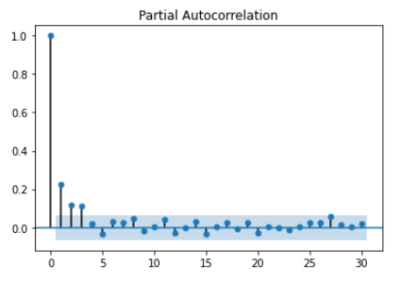
To fit GARCH model it is mandatory to check whether the data suffers from heterocedasticity. This is checked with the help of error obtained from the fitted ARMA model. If the ACF of squared residual of the fitted model is outside the standard error limits then the dataset suffers from heterocedasticity.

```
In [47]: plot_acf(Rsd_Squre);
```



ACF of Squared Residuals

```
In [63]: plot_pacf(Rsd_Squre);
```



PACF of Squared Residuals

The first few lags of the ACF of squared residuals are outside the standard error limits which indicates that the squared error (variance) is depend on time period i.e. presence of heterocedasticity. Parameter in GARCH model is 'p' and 'q', where p refers to number of previous squared residual terms and q refers to number previous variance terms, this can be identified by PACF and ACF of the squared residuals, since both show an exponential decay at lag one. So GARCH(1, 1) model will be appropriate and the fitted model is as follows:

```
In [51]: model1fit.summary()
```

Out[51]:

Zero Mean - GARCH Model Results

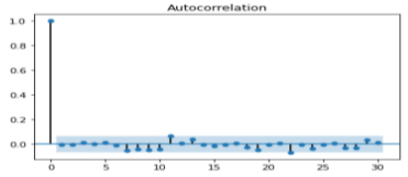
Dep. Variable:	None	**R-squared:**	0.000
Mean Model:	Zero Mean	**Adj. R-squared:**	0.001
Vol Model:	GARCH	**Log-Likelihood:**	-3263.27
Distribution:	Normal	**AIC:**	6532.53
Method:	Maximum Likelihood	**BIC:**	6546.91
		No. Observations:	891
Date:	Sun, Mar 21 2021	**Df Residuals:**	891
Time:	19:52:02	**Df Model:**	0

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	0.7503	0.512	1.464	0.143	[-0.254, 1.755]
alpha[1]	0.0380	2.399e-02	1.585	0.113	[-8.993e-03,8.505e-02]
beta[1]	0.9572	2.365e-02	40.480	0.000	[0.911, 1.004]

The sum of alpha and beta is less than one which is a good sign of better model and the coefficient was also found to be significant and the fitted model residuals are within the standard error limit which is as follows:

```
In [55]: plot_acf(model1fit.resid);
```



GARCH model the variance, ARMA model the dependent variable value, on combining the ARMA-GARCH forecast value one gets the predicted stock price value i.e. the error in ARMA model follows GARCH model so square root of forecasted variance in GARCH model is the error of ARMA model. On combing these two models and applying the RMSE of test and predicted value one gets the results as follows:

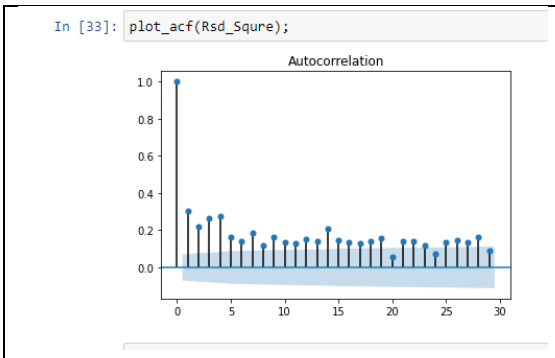
```
In [69]: yfor = modelfit.forecast(steps=245)[0] + error

In [70]: math.sqrt(mean_squared_error(Test,yfor))
Out[70]: 52.34413061082825
```

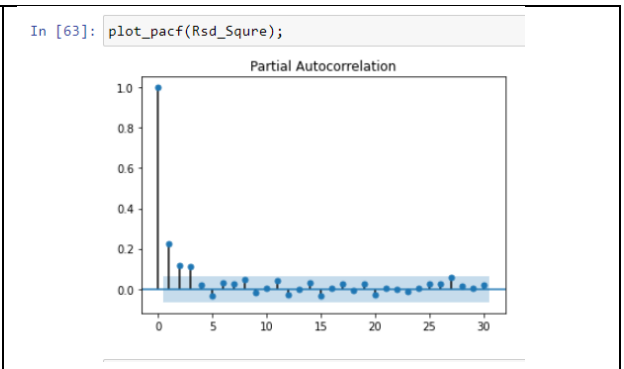
The RMSE of the fitted ARMA (0, 1, 1) – GARCH (1, 1) is found to be 52.34 which pretty good for forecasting purpose.

3.2.2. Fitting GARCH model for Avenue Supermarkets

The error obtained from the fitted ARMA model is checked for the presence of heterocedaticity. If the ACF of squared residual of the fitted model is outside the standard error limits then the dataset suffers from heterocedasticity.



ACF of Squared Residuals



PACF of Squared Residuals

Most of the lags in ACF of squared residuals are outside the standard error limits which indicates that the squared error (variance) is depend on time period for longer period i.e. presence of heterocedasticity. Parameter in GARCH model is ‘p’ and ‘q’, where p refers to number of previous squared residual terms and q refers to number previous variance terms, this can be identified by PACF and ACF of the squared residuals, since both show an exponential decay at lag one. So GARCH(1, 1) model will be appropriate and the fitted model is as follows:

```
In [37]: model1fit.summary()
```

Out[37]:

Zero Mean - GARCH Model Results

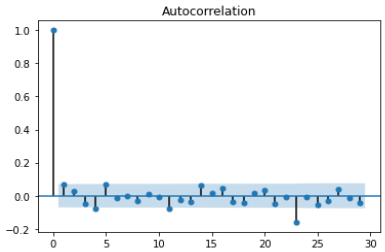
Dep. Variable:	None	**R-squared:**	0.000
Mean Model:	Zero Mean	**Adj. R-squared:**	0.001
Vol Model:	GARCH	**Log-Likelihood:**	-3757.97
Distribution:	Normal	**AIC:**	7521.93
Method:	Maximum Likelihood	**BIC:**	7535.91
		No. Observations:	779
Date:	Thu, Mar 18 2021	**Df Residuals:**	779
Time:	19:56:56	**Df Model:**	0

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	29.2247	20.881	1.400	0.162	[-11.701, 70.150]
alpha[1]	0.1209	4.222e-02	2.864	4.184e-03	[3.817e-02, 0.204]
beta[1]	0.8632	5.357e-02	16.116	1.985e-58	[0.758, 0.968]

The sum of alpha and beta is less than one which is a good sign of better model and the coefficient was also found to be highly significant and the fitted model residuals are within the standard error limit which is as follows:

```
In [55]: plot_acf(model1fit.resid);
```



GARCH model the variance, ARMA model the dependent variable value, on combining the ARMA-GARCH forecast value one gets the predicted stock price value i.e. the error in ARMA model follows GARCH model so square root of forecasted variance in GARCH model is the error of ARMA model. On combing these two models and applying the RMSE of test and predicted value one gets the results as follows:

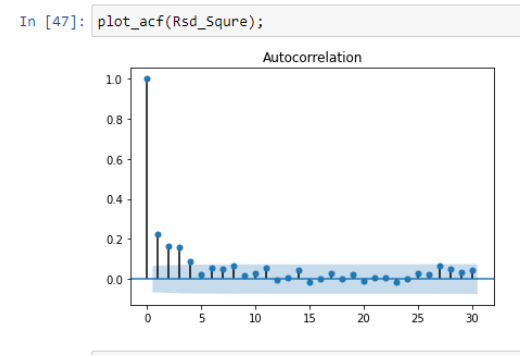
```
In [49]: yfor = modelfit.forecast(steps=195)[0] + error

In [52]: math.sqrt(mean_squared_error(Test,yfor))
Out[52]: 73.72388279019805
```

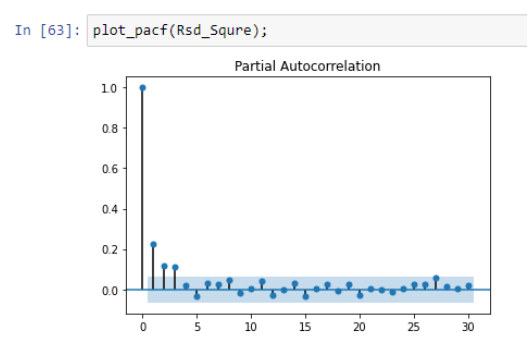
The RMSE of the fitted ARMA (0, 1, 1) – GARCH (1, 1) is found to be 73.72 which pretty good for forecasting purpose.

3.2.3. Fitting GARCH model for Future Retail

The error obtained from the fitted ARMA model is checked for the presence of heterocedaticity. If the ACF of squared residual of the fitted model is outside the standard error limits then the dataset suffers from heterocedasticity.



ACF of Squared Residuals



PACF of Squared Residuals

First few lags in ACF of squared residuals are outside the standard error limits which indicates that the squared error (variance) is depend on time period for longer period i.e. presence of heterocedasticity. Parameter in GARCH model is 'p' and 'q', where p refers to number of previous squared residual terms and q refers to number previous variance terms, this can be identified by PACF and ACF of the squared residuals, since both show an exponential decay at lag one. So GARCH(1, 1) model will be appropriate and the fitted model is as follows:

```
In [51]: model1fit.summary()
```

Out[51]:

Zero Mean - GARCH Model Results

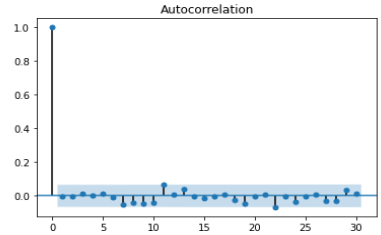
Dep. Variable:	None	**R-squared:**	0.000
Mean Model:	Zero Mean	**Adj. R-squared:**	0.001
Vol Model:	GARCH	**Log-Likelihood:**	-3263.27
Distribution:	Normal	**AIC:**	6532.53
Method:	Maximum Likelihood	**BIC:**	6546.91
		No. Observations:	891
Date:	Sun, Mar 21 2021	**Df Residuals:**	891
Time:	19:52:02	**Df Model:**	0

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	0.7503	0.512	1.464	0.143	[-0.254, 1.755]
alpha[1]	0.0380	2.399e-02	1.585	0.113	[-8.993e-03,8.505e-02]
beta[1]	0.9572	2.365e-02	40.480	0.000	[0.911, 1.004]

The sum of alpha and beta is less than one which is a good sign of better model and the coefficient was also found to be highly significant and the fitted model residuals are within the standard error limit which is as follows:

```
In [55]: plot_acf(model1fit.resid);
```



GARCH model the variance, ARMA model the dependent variable value, on combining the ARMA-GARCH forecast value one gets the predicted stock price value i.e. the error in ARMA model follows GARCH model so square root of forecasted variance in GARCH model is the error of ARMA model. On combing these two models and applying the RMSE of test and predicted value one gets the results as follows:

```
In [61]: yfor = modelfit.forecast(steps=220)[0] + error

In [62]: math.sqrt(mean_squared_error(Test,yfor))
Out[62]: 9.84333933558522
```

The RMSE of the fitted ARMA (0, 1, 1) – GARCH (1, 1) is found to be 9.84 which good for forecasting purpose.

3.3. RNN model

RNN – Recurrent Neural Network is a deep learning model which is mainly used in sequential data, Since Time series is a sequential in nature so it is appropriate to apply RNN model. There is several architecture available for RNN such as Vanilla RNN (Traditional RNN), GRU (Grated Recurrent Unit) and LSTM (Long Short Term Memory). The last five years data is used for analysis so more than 500 lag is available, memory is very high a traditional RNN does not able to remember that much memory so it is good to use LSTM which use a CELL consist three gates (Input Gate, Forget Gate and Output Gate), Sigmoidal and Tanh function is used as a Activation function in RNN.

3.3.1. Fitting RNN model for Reliance Stock Price

Unlike ARIMA model it is not necessary to check stationary in RNN but several preprocessing is required to before build this RNN model. The first step is to standardize the variable and convert the whole value between 0 to 1, minmaxscaler is employed for this

purpose. The next step is to divide the whole dataset into train and test, 80 per cent data (984) is taken as train size and 20 per cent data (247) is taken as test size.

```
In [66]: from sklearn.preprocessing import MinMaxScaler
         scaler=MinMaxScaler(feature_range=(0,1))
         df1=scaler.fit_transform(np.array(df1).reshape(-1,1))

In [67]: df1.shape

Out[67]: (1231, 1)

In [68]: training_size=int(len(df1)*0.80)
         test_size=len(df1)-training_size
         train_data,test_data=df1[0:training_size,:],df1[training_size:len(df1),:1]

In [69]: training_size,test_size

Out[69]: (984, 247)
```

After splitting into train and test, train data is separated into xtrain and ytrain for example if ytrain is value of current time period, the xtrain is the value of previous time period it can be of any dimension 1 to n, where n is number of time steps.

```
In [70]: import numpy
         # convert an array of values into a dataset matrix
         def create_dataset(dataset, time_step=1):
             dataX, dataY = [], []
             for i in range(len(dataset)-time_step-1):
                 a = dataset[i:(i+time_step), 0]   ###i=0, 0,1,2,3-----99   100
                 dataX.append(a)
                 dataY.append(dataset[i + time_step, 0])
             return numpy.array(dataX), numpy.array(dataY)

In [72]: # reshape into X=t,t+1,t+2,t+3 and Y=t+4
         time_step = 60
         X_train, y_train = create_dataset(train_data, time_step)
         X_test, ytest = create_dataset(test_data, time_step)
         X_train

Out[72]: array([[0.02577615, 0.02968243, 0.02172371, ..., 0.00697549, 0.007879  ,
                 0.00659018],
                [0.02968243, 0.02172371, 0.02245448, ..., 0.007879  , 0.00659018,
                 0.00665662],
                [0.02172371, 0.02245448, 0.02384957, ..., 0.00659018, 0.00665662,
                 0.00640417],
                ...,
                [0.57924588, 0.58865279, 0.58299271, ..., 0.48961396, 0.45900143,
                 0.45235812],
                [0.58865279, 0.58299271, 0.58323188, ..., 0.45900143, 0.45235812,
                 0.46654825],
```

LSTM taken input in three dimensions but the split xtrain and ytrain is in 2D, so it is necessary reshape the xtrain and ytrain into three dimension as follows:

```
In [73]: print(X_train.shape), print(y_train.shape)

         (923, 60)
         (923,)

Out[73]: (None, None)

In [74]: # reshape input to be [samples, time steps, features] which is required for LSTM
         X_train =X_train.reshape(X_train.shape[0],X_train.shape[1] , 1)
         X_test = X_test.reshape(X_test.shape[0],X_test.shape[1] , 1)

In [75]: X_train.shape

Out[75]: (923, 60, 1)
```

Tensor flow is used to model RNN, for this data 60 time steps is used and three LSTM layers had been applied in the model, in each layer 10 memory cell is used to remember the information. Adam optimizer is used for back propagation.

```
In [76]: ### Create the Stacked LSTM model
         from tensorflow.keras.models import Sequential
         from tensorflow.keras.layers import Dense
         from tensorflow.keras.layers import LSTM

In [77]: model=Sequential()
         model.add(LSTM(10,return_sequences=True,input_shape=(60,1)))
         model.add(LSTM(10,return_sequences=True))
         model.add(LSTM(10))
         model.add(Dense(1))
         model.compile(loss='mean_squared_error',optimizer='adam')
```

The results of the fitted model are as follows, 10 epochs is applied in the model, loss function degenerate at faster rate.

```
In [79]: model.fit(X_train,y_train,epochs=10,batch_size=4,verbose=1)

         Epoch 1/10
         231/231 [==============================] - 17s 51ms/step - loss: 0.0169
         Epoch 2/10
         231/231 [==============================] - 13s 56ms/step - loss: 8.1773e-04
         Epoch 3/10
         231/231 [==============================] - 13s 58ms/step - loss: 7.6504e-04
         Epoch 4/10
         231/231 [==============================] - 14s 59ms/step - loss: 8.7546e-04
         Epoch 5/10
         231/231 [==============================] - 14s 60ms/step - loss: 8.1863e-04
         Epoch 6/10
         231/231 [==============================] - 14s 60ms/step - loss: 6.8017e-04
         Epoch 7/10
         231/231 [==============================] - 14s 61ms/step - loss: 5.9407e-04
         Epoch 8/10
         231/231 [==============================] - 14s 61ms/step - loss: 5.3704e-04
         Epoch 9/10
         231/231 [==============================] - 14s 61ms/step - loss: 5.6162e-04
         Epoch 10/10
         231/231 [==============================] - 14s 60ms/step - loss: 6.2570e-04

Out[79]: <tensorflow.python.keras.callbacks.History at 0x7fad5cb9aed0>
```

After fitting the model, RMSE is used to check the adequacy of the fitted model. The value of RMSE is 164.07 which are not bad in predicting the stock price of Reliance markets.

```
In [84]: testScore = math.sqrt(mean_squared_error(ytest[0], y_pred[:,0]))
         print('Test Score: %.2f RMSE' % (testScore))

         Test Score: 164.07 RMSE
```

3.3.2. Fitting RNN model for Avenue Supermarket Stock Price

The first step is to standardize the variable and convert the whole value between 0 to 1, minmaxscaler is employed for this purpose. The next step is to divide the whole dataset into train and test, 80 per cent data (781) is taken as train size and 20 per cent data (196) is taken as test size.

```
In [13]: from sklearn.preprocessing import MinMaxScaler
         scaler=MinMaxScaler(feature_range=(0,1))
         df1=scaler.fit_transform(np.array(df1).reshape(-1,1))

In [15]: df1.shape

Out[15]: (977, 1)

In [16]: training_size=int(len(df1)*0.80)
         test_size=len(df1)-training_size
         train_data,test_data=df1[0:training_size,:],df1[training_size:len(df1),:1]

In [17]: training_size,test_size

Out[17]: (781, 196)
```

After splitting into train and test, train data is separated into xtrain and ytrain for example if ytrain is value of current time period, the xtrain is the value of previous time period it can be of any dimension 1 to n, where n is number of time steps.

```
In [18]: import numpy
         # convert an array of values into a dataset matrix
         def create_dataset(dataset, time_step=1):
             dataX, dataY = [], []
             for i in range(len(dataset)-time_step-1):
                 a = dataset[i:(i+time_step), 0]   ###i=0, 0,1,2,3-----99   100
                 dataX.append(a)
                 dataY.append(dataset[i + time_step, 0])
             return numpy.array(dataX), numpy.array(dataY)

In [19]: # reshape into X=t,t+1,t+2,t+3 and Y=t+4
         time_step = 50
         X_train, y_train = create_dataset(train_data, time_step)
         X_test, ytest = create_dataset(test_data, time_step)
         X_train

Out[19]: array([[0.01050873, 0.01006195, 0.00794468, ..., 0.04388027, 0.05023212,
                 0.06229482],
                [0.01006195, 0.00794468, 0.00089353, ..., 0.05023212, 0.06229482,
                 0.06384879],
                [0.00794468, 0.00089353, 0.        , ..., 0.06229482, 0.06384879,
                 0.06773372],
                ...,
                [0.61593594, 0.63143686, 0.6321944 , ..., 0.65744643, 0.66919834,
                 0.71457433],
                [0.63143686, 0.6321944 , 0.59249045, ..., 0.66919834, 0.71457433,
```

LSTM taken input in three dimensions but the split xtrain and ytrain is in 2D, so it is necessary reshape the xtrain and ytrain into three dimension as follows:

```
In [20]: print(X_train.shape), print(y_train.shape)

         (730, 50)
         (730,)

Out[20]: (None, None)

In [21]: # reshape input to be [samples, time steps, features] which is required for LSTM
         X_train =X_train.reshape(X_train.shape[0],X_train.shape[1] , 1)
         X_test = X_test.reshape(X_test.shape[0],X_test.shape[1] , 1)

In [22]: X_train.shape

Out[22]: (730, 50, 1)
```

Tensor flow is used to model RNN, for this data 60 time steps is used and three LSTM layers had been applied in the model, in each layer 10 memory cell is used to remember the information. Adam optimizer is used for back propagation.

```
In [47]: ### Create the Stacked LSTM model
         from tensorflow.keras.models import Sequential
         from tensorflow.keras.layers import Dense
         from tensorflow.keras.layers import LSTM

In [48]: model=Sequential()
         model.add(LSTM(10,return_sequences=True,input_shape=(50,1)))
         model.add(LSTM(10,return_sequences=True))
         model.add(LSTM(10))
         model.add(Dense(1))
         model.compile(loss='mean_squared_error',optimizer='adam')
```

The results of the fitted model are as follows, 10 epochs is applied in the model, loss function degenerate at faster rate.

```
In [50]: model.fit(X_train,y_train,epochs=10,batch_size=4,verbose=1)

         Epoch 1/10
         183/183 [==============================] - 13s 45ms/step - loss: 0.0380
         Epoch 2/10
         183/183 [==============================] - 9s 48ms/step - loss: 0.0023
         Epoch 3/10
         183/183 [==============================] - 9s 47ms/step - loss: 0.0022
         Epoch 4/10
         183/183 [==============================] - 9s 47ms/step - loss: 0.0019
         Epoch 5/10
         183/183 [==============================] - 9s 47ms/step - loss: 0.0015
         Epoch 6/10
         183/183 [==============================] - 9s 47ms/step - loss: 0.0017
         Epoch 7/10
         183/183 [==============================] - 9s 47ms/step - loss: 0.0013
         Epoch 8/10
         183/183 [==============================] - 9s 47ms/step - loss: 0.0012
         Epoch 9/10
         183/183 [==============================] - 9s 47ms/step - loss: 0.0011
         Epoch 10/10
         183/183 [==============================] - 9s 47ms/step - loss: 0.0012

Out[50]: <tensorflow.python.keras.callbacks.History at 0x7fad5f2cca50>
```

After fitting the model, RMSE is used to check the adequacy of the fitted model. The value of RMSE is 158.09 which fair in predicting the stock price of Reliance markets.

```
In [58]: testScore = math.sqrt(mean_squared_error(ytest[0], y_pred[:,0]))
         print('Test Score: %.2f RMSE' % (testScore))

         Test Score: 158.09 RMSE
```

3.3.3. Fitting RNN model for Future Retail Stock Price

The first step is to standardize the variable and convert the whole value between 0 to 1, minmaxscaler is employed for this purpose. The next step is to divide the whole dataset into train and test, 80 per cent data (891) is taken as train size and 20 per cent data (223) is taken as test size.

```
In [94]: from sklearn.preprocessing import MinMaxScaler
         scaler=MinMaxScaler(feature_range=(0,1))
         df1=scaler.fit_transform(np.array(df1).reshape(-1,1))

In [95]: df1.shape

Out[95]: (1114, 1)

In [96]: training_size=int(len(df1)*0.80)
         test_size=len(df1)-training_size
         train_data,test_data=df1[0:training_size,:],df1[training_size:len(df1),:1]

In [97]: training_size,test_size

Out[97]: (891, 223)
```

After splitting into train and test, train data is separated into xtrain and ytrain for example if ytrain is value of current time period, the xtrain is the value of previous time period it can be of any dimension 1 to n, where n is number of time steps.

```
In [98]: import numpy
         # convert an array of values into a dataset matrix
         def create_dataset(dataset, time_step=1):
             dataX, dataY = [], []
             for i in range(len(dataset)-time_step-1):
                 a = dataset[i:(i+time_step), 0]   ###i=0, 0,1,2,3-----99   100
                 dataX.append(a)
                 dataY.append(dataset[i + time_step, 0])
             return numpy.array(dataX), numpy.array(dataY)

In [100]: # reshape into X=t,t+1,t+2,t+3 and Y=t+4
          time_step = 55
          X_train, y_train = create_dataset(train_data, time_step)
          X_test, ytest = create_dataset(test_data, time_step)
          X_train

Out[100]: array([[0.1647986 , 0.16005118, 0.16229543, ..., 0.11110956, 0.10653477,
                  0.09937048],
                 [0.16005118, 0.16229543, 0.15772064, ..., 0.10653477, 0.09937048,
                  0.10532633],
                 [0.16229543, 0.15772064, 0.15823852, ..., 0.09937048, 0.10532633,
                  0.11715174],
                 ...,
                 [0.50424944, 0.50495876, 0.49423002, ..., 0.01214738, 0.00594069,
                  0.        ],
                 [0.50495876, 0.49423002, 0.49440737, ..., 0.00594069, 0.        ,
                  0.00425602],
                 [0.49423002, 0.49440737, 0.48961938, ..., 0.        , 0.00425602,
                  0.01010805]])
```

LSTM taken input in three dimensions but the split xtrain and ytrain is in 2D, so it is necessary reshape the xtrain and ytrain into three dimension as follows:

```
In [101]: print(X_train.shape), print(y_train.shape)

          (835, 55)
          (835,)
Out[101]: (None, None)

In [102]: # reshape input to be [samples, time steps, features] which is required for LSTM
          X_train =X_train.reshape(X_train.shape[0],X_train.shape[1] , 1)
          X_test = X_test.reshape(X_test.shape[0],X_test.shape[1] , 1)

In [103]: X_train.shape
Out[103]: (835, 55, 1)
```

Tensor flow is used to model RNN, for this data 60 time steps is used and three LSTM layers had been applied in the model, in each layer 10 memory cell is used to remember the information. Adam optimizer is used for back propagation.

```
In [104]: ### Create the Stacked LSTM model
          from tensorflow.keras.models import Sequential
          from tensorflow.keras.layers import Dense
          from tensorflow.keras.layers import LSTM

In [105]: model=Sequential()
          model.add(LSTM(10,return_sequences=True,input_shape=(55,1)))
          model.add(LSTM(10,return_sequences=True))
          model.add(LSTM(10))
          model.add(Dense(1))
          model.compile(loss='mean_squared_error',optimizer='adam')
```

The results of the fitted model are as follows, 10 epochs is applied in the model, loss function degenerate at faster rate.

```
In [107]: model.fit(X_train,y_train,epochs=10,batch_size=4,verbose=1)

          Epoch 1/10
          209/209 [==============================] - 15s 48ms/step - loss: 0.1450
          Epoch 2/10
          209/209 [==============================] - 11s 52ms/step - loss: 0.0071
          Epoch 3/10
          209/209 [==============================] - 11s 53ms/step - loss: 0.0055
          Epoch 4/10
          209/209 [==============================] - 11s 55ms/step - loss: 0.0042
          Epoch 5/10
          209/209 [==============================] - 12s 55ms/step - loss: 0.0036
          Epoch 6/10
          209/209 [==============================] - 12s 55ms/step - loss: 0.0034
          Epoch 7/10
          209/209 [==============================] - 12s 55ms/step - loss: 0.0035
          Epoch 8/10
          209/209 [==============================] - 11s 55ms/step - loss: 0.0020
          Epoch 9/10
          209/209 [==============================] - 11s 55ms/step - loss: 0.0019
          Epoch 10/10
          209/209 [==============================] - 11s 54ms/step - loss: 0.0016
Out[107]: <tensorflow.python.keras.callbacks.History at 0x7fad5aab4790>
```

After fitting the model, RMSE is used to check the adequacy of the fitted model. The value of RMSE is 25.42 which good in predicting the stock price of Reliance markets.

```
In [114]: testScore = math.sqrt(mean_squared_error(ytest[0], y_pred[:,0]))
          print('Test Score: %.2f RMSE' % (testScore))

          Test Score: 25.42 RMSE
```

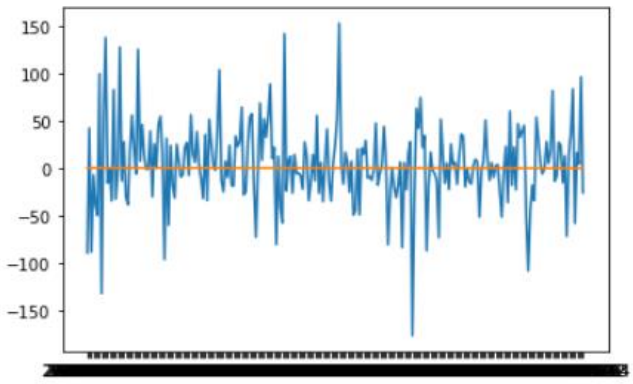
CHAPTER 4: FINAL MODEL AND CONCLUSION

Machine learning and Deep learning models were used for forecasting stock price of Reliance, Avenue Supermarkets and Future retails. The RMSE score of the entire three models was given as follows:

COMPANY	MODEL	RMSE
Reliance	ARIMA(0, 1, 1)	43.73
	ARMA(0, 1, 1) – GARCH (1, 1)	52.34
	RNN	164.07
Avenue Supermarket (DMART)	ARIMA(0, 1, 1)	53.79
	ARMA(0, 1, 1) – GARCH (1, 1)	73.72
	RNN	158.09
Future Retails Ltd.	ARIMA(4, 1, 4)	5.19
	ARMA(0, 1, 1) – GARCH (1, 1)	9.84
	RNN	25.42

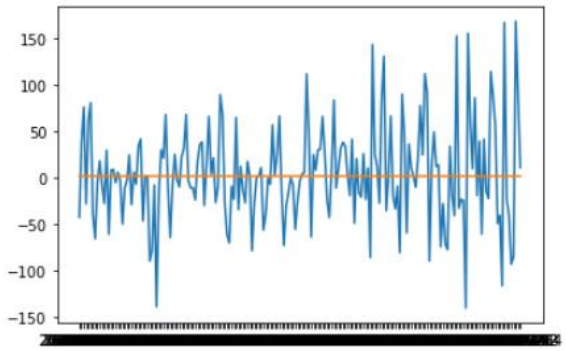
ARIMA model outperformed all the two models in forecasting the stock price for all the three companies. ARIMA(0, 1, 1) was found to better in forecasting Reliance and DMART stock price and ARIMA(4, 1, 4) is better to forecast Future Retails Ltd. The out of sample forecast with it test value is given as follows:

```
In [44]: plt.plot(Test)
         plt.plot(modelfit.forecast(steps=245)[0])
Out[44]: [<matplotlib.lines.Line2D at 0xd30b568>]
```



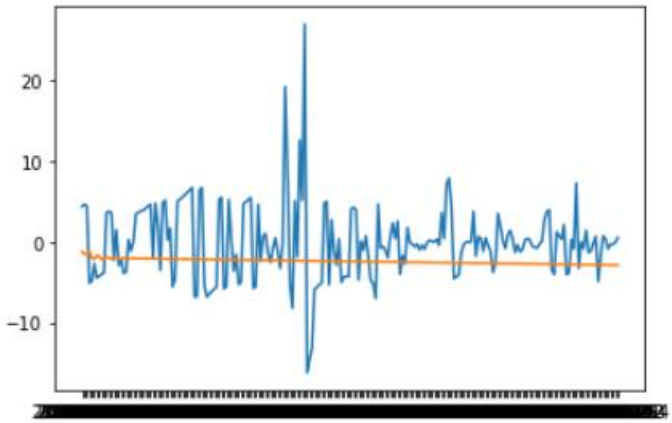
Reliance Stock price out of sample forecast(RED) vs Test value(BLUE)

```
In [30]: plt.plot(Test)
         plt.plot(modelfit.forecast(steps=195)[0])
Out[30]: [<matplotlib.lines.Line2D at 0xc932c88>]
```



Avenue Supermarket Stock price out of sample forecast(RED) vs Test value(BLUE)

```
In [44]: plt.plot(Test)
         plt.plot(modelfit.forecast(steps=220)[0])
Out[44]: [<matplotlib.lines.Line2D at 0xc56ac28>]
```



Future Retail Ltd Stock price out of sample forecast(RED) vs Test value(BLUE)

The above analysis helps the stakeholders and investors to forecast the future stock price the top three fast growing companies. It helps them to make a better decision and reduce risk in investment process. It also helps to formulate policy by the company to identify changes and fluctuations happening around the stock price.

THANK YOU