VOLUME - 10, ISSUE - 02, FEBRUARY - 2021 • PRINT ISSN No. 2277 - 8160 • DOI : 10.36106/gjra

Original Research Paper

Management

STOCK MARKET PREDICTION: USING ECONOMETRIC MODELS AND NEURAL NETWORKS

Parichay Pothepalli

ABSTRACT Stock market trading involves buying and selling of shares or stocks, which represents ownership of business. This research paper will focus on capturing the algorithmic trading based on historical data and compare present day algorithms to find the best fit model to understand the underlying patterns in stock market trading. A comparative analysis of closing stock price for 12 companies from three different sectors has been considered to understand the efficacy of the models in order to predict the future stock prices with minimal errors. Stock market was earlier predicted using traditional econometric models like the ARIMA and SARIMA, however, in this paper, Machine Learning, a part of Artificial Intelligence will be incorporated in the stock data collected from Yahoo Finance to train models and provide predictions/decisions without being explicitly programmed to do so. Models such as OLS, SARIMA, Convolutional Neural Networks and Recursive Neural Networks (LSTM) will also be used to analyze the historical stock data and will be compared for accuracy using testing parameters like Mean Squared Error (MSE).

KEYWORDS : Stock market, OLS, SARIMA, CNN, RNN, LSTM, MSE

INTRODUCTION

Stock market buyers and sellers do not invest in a particular company and wait for the returns, rather the main motive is to diversify the portfolio of the stocks to ensure minimal risk. Stock market predictions not only consider the returns but also the ways in which one manages risk.

Neural networks are an enhanced way to predict the stock market as they can deal with time series data. The primary factor to be accommodated in the financial models used is 'Time'. Traditional econometrics literature suggests that models such as Autoregressive Integrated Moving Average (ARIMA) and Seasonal Autoregressive Integrated Moving (SARIMA) are good for stock market prediction.

This paper aims to understand the differences between various predictive models and suggest a best fit model to predict the closing price of the stocks, given the situation that stock market is very volatile.

DATA PRE-PROCESSING

The data used for this research paper is taken from Yahoo Finance that dates from Jan 1972 to Jan 2021 and a snapshot of the stock data is presented below in Figure 1. The data is preprocessed by reformatting, dealt with missing data, checked for stationarity, lagging, split data into training, test and validation and then used for data normalization and rescaling.

Out[5]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	1972-06-01	0.000000	0.815346	0.802993	0.815346	0.184002	2458800
1	1972-06-02	0.815346	0.817817	0.802993	0.805463	0.181772	1613900
2	1972-06-05	0.805463	0.807934	0.798051	0.802993	0.181214	2585300
3	1972-06-06	0.802993	0.825229	0.800522	0.820288	0.185117	2347500
4	1972-06-07	0.820288	0.820288	0.807934	0.820288	0.185117	1032100
...	...	...	...	...	...	...	...
12255	2021-01-05	36.720001	37.369999	36.599998	37.189999	37.189999	29909400
12256	2021-01-06	36.830002	37.480000	36.770000	36.869999	36.869999	34962500
12257	2021-01-07	37.040001	37.150002	36.669998	37.060001	37.060001	27809500
12258	2021-01-08	37.160000	37.529999	36.900002	37.130001	37.130001	33462400
12259	2021-01-11	37.220001	37.830002	37.150002	37.770000	37.770000	47286900

12260 rows × 7 columns

Figure 1: Snapshot of the Pfizer stock dataset

a) Reformatting and sorting the data

To input the data into various models, the time variable has been converted to a datetime variable. For understanding the time series data, the dataset has been sorted by the "Date" variable and the data on the weekends is dropped, if any, because the stock trading is usually closed on the weekends.

b) Dealing with missing data using linear interpolation

For the missing data, linear interpolation can be done to ensure that there is minimal data loss and also the missing values can be fit by using linear polynomials in order to construct various data points. It allows the datapoint to obtain a discrete value akin to the known set of values available in the dataset obtained. (Pfizer stock market)

c) Checking for stationarity of data

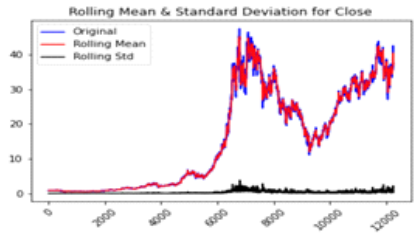
Stationarity is a property of time series data stating that the distributional properties (mean and standard deviation) of the data series has not changed across time. The data that we have today is representative of the data that we may have in the future. For time series forecasting and for the purpose of stock predictions the dataset has to be stationary. A Dickey Fuller test is performed to understand the stationarity of the closing price column .Packages like "statsmodels. tsa. stattools" and "statsmodels. graphics.tsaplots" have been used to perform the Dickey Fuller test. (Figure 2).

```
import matplotlib.pyplot as plt
import statsmodels.tsa.stattools
from statsmodels.tsa.stattools import adfuller
import statsmodels.tsa.x13
from statsmodels.tsa.x13 import x13_arima_select_order, _find_x12
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.stattools import acf, pacf
import statsmodels.graphics.tsaplots as tsaplots

def test_stationarity(df, ts):

    # Determing rolling statistics
    rolmean = df[ts].rolling(window = 12, center = False).mean()
    rolstd = df[ts].rolling(window = 12, center = False).std()


    orig = plt.plot(df[ts], color = 'blue',label = 'Original')
    mean = plt.plot(rolmean, color = 'red',label = 'Rolling Mean')
    std = plt.plot(rolstd, color = 'black', label = 'Rolling Std')
    plt.legend(loc = 'best')
    plt.title('Rolling Mean & Standard Deviation for %s' %(ts))
    plt.xticks(rotation = 45)
    plt.show(block = False)
    plt.close()


    print ('Results of Dickey-Fuller Test:')
    dftest = statsmodels.tsa.stattools.adfuller(df[ts], autolag='AIC') #Add Lags

    dfoutput = pd.Series(dftest[0:4], index = ['Test Statistic','p-value', '# Lags Used','Number of Observations Used'])
    for key, value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print (dfoutput)


test_stationarity(df, 'Close')
```

Figure 2: Dickey Fuller test

Upon performing the Dickey-Fuller test we gain insight into the p-Value and use hypothesis testing to understand if the stationarity in the data is due to random chance variation or due to the intrinsic property of the data.

Null Hypothesis (H_0): Time series is not stationary. Alternate Hypothesis (H_1): Time series is stationary. The output shown in Figure 3, indicates a p-Value of 0.83 which means that we choose to reject the Null hypothesis of the data being stationary.




```
Results of Dickey-Fuller Test:
Test Statistic                  -0.757096
p-value                          0.831401
# Lags Used                     40.000000
Number of Observations Used  12219.000000
Critical Value (1%)             -3.430885
Critical Value (5%)             -2.861777
Critical Value (10%)            -2.566896
dtype: float64
```

Figure 3: Results of the Dickey Fuller Test

The graph in Figure 3 indicates that there is a huge variation in the stock closing which is usually the case with big pharmaceutical companies.

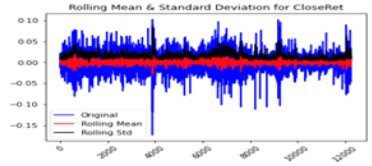
a) Fixing the non-stationarity of the data

Percentage change is used to make the dataset stationary in the place of the first difference. This allows the data to have equal ground and can be directly used in the ML and Neural Networks. We therefore use the below function named "percentChange" to calculate the percentage change for the closing stock price as in Figure 4. The inputs include the column for the closing stock price and the number of lags. "NumLags" here is used to calculate the percentage change over time. For a daily change we can take 1 day, for a monthly change we can consider 30 days and so on. Since the data has only level variables, no segregation of level/non level variables is required.

```
def percentChange(x,numLags):
    y = (x - x.shift(numLags))/x.shift(numLags)
    return y

dataForMl = pd.DataFrame()
dataForMl['Date'] = df['Date']
levelVars = df.columns[:-1]
for levelVar in levelVars:
    dataForMl[f'{levelVar}Ret'] = percentChange(df[levelVar],1)

dataForMl = dataForMl[1:]
```

Figure 4: "PercentChange" function

The formula is: Percentage change (for yesterday) = [value of stock (today) – value of stock (yesterday)]/ value of stock (yesterday) The graph in Figure 5 depicts a flatness as compared to Figure 3 which means that the percentage change function has converted the data to stationary data. Thereby the p-value of 0 (an approximation) makes us accept the Null Hypothesis.




```
Results of Dickey-Fuller Test:
Test Statistic                -23.580731
p-value                         0.000000
# Lags Used                    25.000000
Number of Observations Used  12233.000000
Critical Value (1%)             -3.430885
Critical Value (5%)             -2.861776
Critical Value (10%)            -2.566896
dtype: float64
```

Figure 5: Results of the Dickey Fuller test after fixing non-stationarity.

e) Lagging the data

Time series has a sequential nature to the data and is autoregressive in nature. The value for today's data is heavily dependent on yesterday's data which again is heavily dependent on day before yesterday's data. Therefore, one needs to create lagged versions of the independent variable (Pfizer closing stock price). The number of maximum lags to be used to predict any data is calculated using Auto Correlation Function (ACF) and the Partial Autocorrelation Function (PACF), as shown in Figure 6.

Autocorrelation function tells us the correlation between today's value and every lag that has been created. Partial correlation function is a secondary measure to understand the residual correlation in the data after the ACF (Figure 7) has been taken into account.

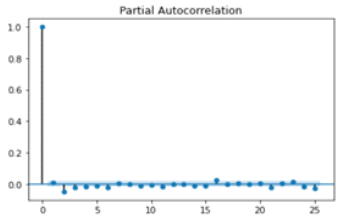


Figure 6: PACF Plot

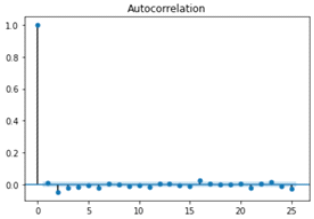


Figure 7: ACF Plot

The partial blue area in the above figures is the area of significance. And as per the graphs we can see that the number of blue dots above the area of significance tells us the nature of stock market. The closing price for the given stock is highly corelated to the previous day and then the correlation slowly dies down due to the volatility in the stock market. However as per prior stock market predictions and literature studies, the number of max lags has been set to 10, as seen in Figure 8.

```
minLagNum = 1
maxLagNum = 10
dataForMl = dataForMl.sort_values(['Date'])
for column in dataForMl.columns:
    for lag in range(minLagNum,maxLagNum+1):
        dataForMl[f'{column}Lag_{lag}'] = dataForMl[f'{column}'].shift(lag)
```

Figure 8: Function for lagging

This leads to lagging of every variable in the dataset up to 10 lags to ensure that the stock price predictions are not done on the basis of present-day data but on the same data lagged by 10 as shown below in Figure 9.

VolumeRetLag_2	VolumeRetLag_3	VolumeRetLag_4	VolumeRetLag_5	VolumeRetLag_6	VolumeRetLag_7	VolumeRetLag_8
NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN
-0.343623	NaN	NaN	NaN	NaN	NaN	NaN
0.601896	-0.343623	NaN	NaN	NaN	NaN	NaN
-0.091982	0.601896	-0.343623	NaN	NaN	NaN	NaN

Figure 9: Snapshot of data after lagging.

f) Training, Testing and Validation

The objective of a machine learning method is to predict the dependent variable as accurately as possible i.e., to minimize the predicted error (the difference between the actual value and predicted value). To measure the predictive accuracy of a model, it is important that the forecast accuracy be measured out - of sample, as the training accuracy can be made

arbitrarily high through overfitting. Therefore, to protect against 'data leakage', the out-of-sample data has been split into two parts: validation data and testing data. The validation set allows the evaluation of the model on unseen data to select the best model architecture, while still holding out a subset of data for final evaluation after finding the best model, seen in Figure 10.

```
test_percent = 0.10
no_test_obs =  int(np.round(test_percent*len(dataForMl)))
training = dataForMl[:-no_test_obs]
testing = dataForMl[-no_test_obs:]

#breaking the testing data into validation and out of sample data
validation_percent = 0.70
no_validation_obs = int(np.round(validation_percent*len(testing)))
validation = testing[:-no_validation_obs]
outOfSample = testing[-no_validation_obs:]
```

Figure 10: Splitting the data

g) Data Normalization and Re-scaling

Using packages like "sklearn.preprocessing" the "Minmacscaler" is imported in order to ensure that the data has been normalized i.e.(within a scale of -1 to +1). As seen in Figure 11, this normalization along with rescaling is done because ML models and Neural Networks are very sensitive to the scale of the data and therefore a common normalized data will help in the comparative analysis of the models one can fit.

```
from sklearn.preprocessing import MinMaxScaler

min_max_scaler = MinMaxScaler(feature_range=(-1, 1))
trainMinmax = min_max_scaler.fit_transform(training.values)
valMinmax = min_max_scaler.transform(validation.values)
outSampleMinmax = min_max_scaler.transform(outOfSample.values)
```

Figure 11: Normalization and rescaling

METHODOLOGY:

Part A: Prediction using OLS (Ordinary Least Squared Error)

Time series data has very unique properties that the OLS method or the simple Linear regression cannot cater to. These properties include that the data is sequentially ordered i.e., there is a time component related to it and that the data shows multicollinearity which is again not supported by OLS. Refer to Table 1.

Table1: Fitting OLS in time series data

Property of Time series data	OLS?	Reason for not considering OLS
Sequentially ordered data	NO	Order of the data does not matter in OLS. OLS displays no auto-correlation.
Displays trends/seasonality	NO	OLS does not display multicollinearity.
Data may not be stationary	NO	Assumptions of OLS include only homoskedasticity.

Part B: Prediction using Seasonal Autoregressive Integrated Moving Average (SARIMA)

Time series data can be dealt with traditional econometric models like ARIMA, which is used to simply predict the values of the closing stock price using the previous historical data. A variation of ARIMA is the seasonal ARIMA that has been used in this paper, taking into consideration some of the seasonal components. The seasonal ARIMA (SARIMA) is capable of modelling the seasonal components in a univariate time series in addition to the autoregressive, moving average and trend components typically modelled by ARIMA.

The variables based on which, the model has been built is mentioned in Table 2.

Table 2: Variables for the SARIMA

Variable	Significance	Value used
p	Determined from the PACF. It is the trend autoregressive order.	1 (refer PACF plot)
d	Determines the stationarity of the data.	0(Stationary data)
q	Determined from the ACF plot. It is the trend of the moving average order.	1 (refer ACF plot)
P	Number of seasonal autoregressive terms	2
D	Number of seasonal difference terms	2
Q	Number of seasonal moving average terms	2
M	Number of timesteps for a seasonal period	3

Using the above-mentioned parameters in SARIMA model on the training data, we calculate the MSE (Mean-Squared Error) of the validation data and the Testing data, as seen in Figure 12.

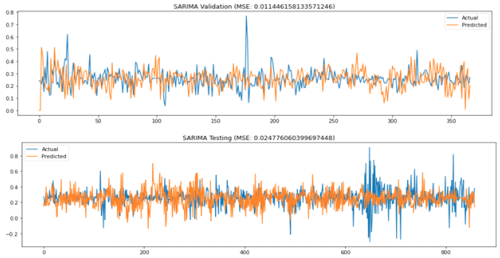


Figure 12: MSE Values for SARIMA

We use the SARIMA model as the benchmark model for understanding how the Neural Networks perform better/worse using metrics like MSE. These values will be taken into consideration for suggesting the best fit model for predictions, as mentioned in the conclusion.

Disadvantages of SARIMA model:

- Potential for error propagation.
- Leads to mean reverting i.e., as the number of steps forecasted ahead in the future increases, the forecast converges to the mean.
- Poor at predicting turning points in any given data.

Though there are a few disadvantages to SARIMA model, the econometric literature deems SARIMA to be a good model for predicting stock markets.

PART C: Prediction using Convolutional Neural Networks

Neural Networks are composite functions which are basically universal function approximators. Every neural network is comprised of three types of layers i.e. the input layer, the computational/hidden layer and the output layer.

Each of the 3 layers is comprised of multiple nodes and is connected to the subsequent layer through weights.

Below (Table 3) is a comparison of various machine learning models and their compatibility with time series data.

Table 3: Models suitable for Time Series Data

Method	Type	Suitable for time series?
Elastic net	Linear	No
Random Forests	Non-Linear	No

Xgboost	Non-Linear	No
CNN	Non-Linear	Yes
LSTM (Long short term memory)	Non-Linear	Yes

A feed forward neural network such as Random forests and XgBoost are not considered to be ideal for time series data as they cannot gauge information from the spatial and sequential nature of time series data. Convolutional Neural Networks (CNN) and Long Short Term Memory (LSTM) can learn an internal representation of the time series data and ideally achieve comparable performance to models fit on a version of the dataset with engineered features. The reason why these specific models have been considered in this research paper is because the input layer preserves the temporal structure of data and the different computational nodes used in a CNN, LSTM network result in two types of networks processing the data differently.

In order to preserve the temporal structure of the data, a function called split sequences has been used to split the multivariate sequence into samples and find the pattern that drives the temporal structure of the data. This function includes the "steps in" i.e., the number of observations from the past that are assumed to be relevant across time and "steps out" indicates the number of units ahead that needs to be forecast into the future, Figure 13.

```
from numpy import array

def split_sequences(sequences, n_steps_in, n_steps_out):
    X, y = list(), list()
    for i in range(len(sequences)):
        end_ix = i + n_steps_in
        out_end_ix = end_ix + n_steps_out-1
        if out_end_ix > len(sequences):
            break
        seq_x, seq_y = sequences[i:end_ix, :-1], sequences[end_ix-1:out_end_ix, -1]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

n_steps_in =30
n_steps_out = 1

trainSeq_x, trainSeq_y = split_sequences(trainMinmax, n_steps_in,n_steps_out)

validationSeq_x, validationSeq_y= split_sequences(valMinmax, n_steps_in,n_steps_out)

outSampleSeq_x, outSampleSeq_y= split_sequences(outSampleMinmax, n_steps_in,n_steps_out)
```

Figure 13: Function to split sequences

The "trainMinmax" consists of the order of a csv file i.e. only rows and columns. However, on using split sequences, as seen in Figure 14, one can see the three dimensions to the "trainSeq_x". These three dimensions here are the number of samples, size of the timesteps and the number of independent variables.

```
trainMinmax.shape
(11013, 61)

trainSeq_x.shape
(10984, 30, 60)
```

Figure 14: Training data temporal shape

For the CNN model, the keras library has been used and the TensorFlow library has been used for closing the price prediction. The CNN extracts information using filters that look for patterns in spatially adjacent data. Usually, CNN filters can be moved along multiple dimensions but as we are dealing with a timeseries data, the filter moves only in one direction i.e., time, to create a feature map. While training the model it is important that we use the MSE error of the validation set to decide when to stop training our network. If we use the MSE of the training set, it does not produce good predictions in the test set due to over fitting. However, unlike the error in the training set, the error in the validation set does not reduce with every passing epoch. Sometimes, it increases for a while before it starts declining. The patience argument in Earlystop allows us to decide how many times we want the validation error to keep increasing before we stop training the neural network. Then we initialize the keras sequential model.

```
model = Sequential() #initializing keras Sequential mode

#convolutional layer starts
model.add(Conv1D(filters=5,#number of filters
                 kernel_size=2,#size of the filte racros
                 strides=2, #number of rows that the fil
                 activation='linear',#transformation
                 input_shape=(n_steps_in, n_features)))
                #kernel_regularizer=l2(0.009),, bias_reg
#convolutional layer ends
```

Figure 15: Initializing Keras Sequential Model

After performing the batch normalization, a regression layer is added as seen in Figure 16.

```
#model.add(BatchNormalization())
model.add(Dropout(0.1)) #reduces overfitting by dropping sc

#regression layer begins
model.add(Flatten())
model.add(Dense(1,
                activation='tanh',
                kernel_regularizer=l2(0.01)))

model.compile(loss='mean_squared_error', optimizer='sgd')
```

Figure 16: Batch Normalization and regression layers

The model. Summary () in Figure 17 tells us the number of weights in each layer. Convolutional layer has 605 weights, Maxpooling and dropout layers have no weights as they are not computational layers.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_1 (Conv1D)            (None, 15, 5)             605
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 15, 5)             0
_________________________________________________________________
dropout_1 (Dropout)          (None, 15, 5)             0
_________________________________________________________________
flatten_1 (Flatten)          (None, 75)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 76
=================================================================
Total params: 681
Trainable params: 681
Non-trainable params: 0
```

Figure 17: CNN model Summary

A depiction of the MSE for the CNN Validation set and the Testing set is as seen below in Figure 18. One can notice that the MSE errors have reduced significantly from the SARIMA benchmark for both the testing and the validation data set.

CNN models are widely used for feed forward and backpropagation algorithms and have widely been used.

However, a shortcoming in the CNN models is that for time series data, they do not draw information from the sequential nature of independent data.

Hence, a special type of Recurrent Neural Network i.e., the LSTM is analyzed in PART D.

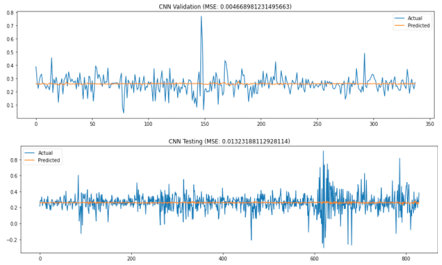


Figure 18: MSE Values for CNN model

PART D PREDICTION USING LSTM (LONG SHORT TERM MEMORY)

LSTM networks draw information from the sequential nature of the data because they have memory. LSTMs build short and long term memories by revealing data to the hidden nodes in a sequential fashion. Each LSTM node is comprised of LSTM

cells and each LSTM cell has 4 gates/ vectors. 1) The Forget Gate / Remember Vector, 2) Candidate Gate, 3) Input Gate / Save Vector, 4) Output gate / Focus Vector. The updated working memory at the end of the sequence is the output of an LSTM node.

Figures 20,21 indicate the code for LSTM model and their respective MSE values for the validation and the testing data set. We realize that the Validation and the Testing set MSE scores are almost similar in the Pfizer closing price dataset. LSTM is considered to be the This paper utilizes the early stopping method (Figure 19) as discussed earlier and imports the model sequential.

best fit model but for all practical and heuristic purposes the best fit model may vary and, in this case, the Convolutional Neural networks have performed at par with the LSTM network.

```
EarlyStop = EarlyStopping(monitor='val_loss', patience=10, verbose=0, mode='auto',restore_best_weights=True)
epochs = 100000
lr =0

sgd = SGD(lr=lr)
bs = 100

n_steps_in =30
n_steps_out = 1
```

Figure 19: Function for Earlystopping

```
model = Sequential()
model.add(LSTM(300, #number of LSTM nodes
               input_shape=(n_steps_in, n_features),
               activation = 'tanh'))
model.add(Dropout(0.1))
model.add(Dense(1,activation = 'linear'))
model.compile(loss='mean_squared_error', optimizer='sgd')

#model
model.fit(trainSeq_x, trainSeq_y,batch_size=bs,epochs=epochs, callbacks= [EarlyStop] ,verbose=2, shuffle=False,
                    validation_data =(validationSeq_x, validationSeq_y))
```

Figure 20: LSTM Model

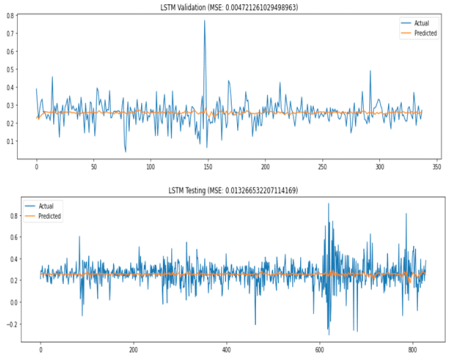


Figure 21: MSE values for LSTM mod

Table 4: Comparison of the models used for prediction

Nature of time series data	OLS	SARIMA	CNN	LSTM
Data may not be stationary	No	Yes	Yes	Yes
Requires no. of observations to be less than no. of independent variables	No	No	Yes	Yes
Temporal structure of data	No	No	Yes	Yes
New features created	No	No	Yes	Yes
Sequentially ordered	No	Yes (only for Y variable)	Yes	Yes
Displays trends	No	Yes (only for Y variable)	Yes	Yes
MSE (Mean Squared Error values)	NA	Validation: 0.0114 Test:0.0247	Validation: 0.0046 Test:0.0132	Validation:0.0047 Test:0.0131

Table 5: Comparison of models across various sectors and companies.

		SARIMA		CNN		LSTM	
	Companies	Validation	Testing	Validation	Testing	Validation	Testing
Pharma	Pfizer	0.0114	0.0247	0.0046	0.0132	0.0047	0.0131
	Sanofi	0.01	0.021	0.0099	0.0221	0.0087	0.021
	Roche	0.0067	0.0276	0.007	0.0254	0.0065	0.02
	GlaxoSmithKline	0.0034	0.0063	0.0036	0.0064	0.0036	0.0066
Automobile	Toyota	0.0057	0.0086	0.0043	0.0082	0.0041	0.0081
	Tesla	0.1051	0.054	0.0966	0.0522	0.0972	0.05
	Honda	0.0029	0.0091	0.0027	0.0087	0.0027	0.0085
	Nissan	0.0037	0.019	0.0035	0.019	0.0035	0.0199
Tech companies	Google	0.0105	0.0216	0.0088	0.019	0.0087	0.013
	Amazon	0.0077	0.0052	0.007	0.004	0.007	0.005
	Facebook	0.0144	0.0103	0.0077	0.0097	0.0077	0.0103
	Apple	0.0026	0.0034	0.0025	0.0032	0.0024	0.003

RESULT:

After analysis of the data of Pfizer stock one can predict the stock price not only by using the traditional model such as Seasonal Autoregressive Integrated Moving Average (SARIMA) but also advanced models such as CNN and LSTM. All these models can be used to predict the future and understand the algorithmic trading behind the stock market. Similar analysis using the above-mentioned modelling was applied to three different sectors namely Pharmaceuticals, Automobiles and Tech companies across 12 different companies. Taking into consideration, the MSE scores observed in the above Table 4 and Table 5, the comparison of stock prices of companies across varied sectors showed that more than 90% of the times the LSTM model produced far better predictions and lesser MSE scores than CNN and the SARIMA models. Therefore, LSTM is deemed a better fit model than the traditional OLS, SARIMA and CNN, due to its lesser MSE score and the way in which it caters to the time series data. These observations help us to sum up that the Neural Networks are the most efficient in predicting stock market.

CONCLUSION:

Machine Learning and AI are extensively being used to understand the core of algorithmic trading and predict future outcomes. Increasingly, today consulting companies are hired by hedge fund companies and investment broking services to understand the subtle nuances in this very volatile stock market. This paper has analyzed stock market data by using traditional as well as advanced models and their practical application with reference to stocks. The analysis and stock market predictions using these models have produced very low MSE scores, but in real life there are numerous extrinsic factors other than historical data, that should be taken into consideration while predicting the stock market. Keeping in view the innumerable extrinsic factors, the LSTM model is the nearest possible solution for predictions.

REFERENCES:

[1] Ayodele A. Adebiyi., 2 Aderemi O. Adewumi (2014), "Stock market prediction using the ARIMA model." UKSim-AMSS 16th International Conference on Computer Modelling and Simulation

[2] Lokesh Kumar Shrivastav; Ravinder Kumar (2019)," An Empirical Analysis of Stock Market Price Prediction using ARIMA and SVM", IEEE

[3] Mohankumari C, Vishukumar M (2019)," ANALYSIS OF DAILY STOCK TREND PREDICTION USING ARIMA MODEL", International Journal of Mechanical Engineering and Technology (IJMET)

[4] M Mallikarjuna, R. Prabhakar Rao (2019)," Evaluation of forecasting methods from selected stock market returns", Springer Open.

[5] Bijesh Dhyani, Manish Kumar, P (2020)," Stock Market Forecasting Technique using Arima Model", International Journal of Recent Technology and Engineering (IJRTE)