Chapter 3: Advance approach to Solve Time Series of Stock Market via AI & ML

Introduction: We will Build a Stock Market prediction Model with Machine Learning Algorithms (Ridge & Lasso) & Deep Learning Algorithms with LSTM & along side we will cover ARIMA Modelling to predict

**Structure:**

* Problem Statement
* Python Dependency Libraries
* Classical Statistics ARIMA Modelling
* Hybrid Machine Learning with Time Series
* LSTM Architecture

**Objectives**

We will cover the following recipes:

* Various Statistical Analysis & Visualization using one of the best tool (Plotly)
* We have learnt complete end to end analysis on ARIMA/ARMA model of classical statistics with its prediction
* How to implement Machine learning Ridge and LASSO to predict the stocks and evaluated thos ewith various parameter s and grid search for beetr accuracy and MAPE.
* Creation of new Deep learning Architecture LSTM to predict the stocks and various techniques to create the LSTM Framework.

**3.1 Problem Statement**

There are a lot of complicated financial indicators and also the fluctuation of the stock market is highly violent. However, as the technology is getting advanced, the opportunity to gain a steady fortune from the stock market is increased and it also helps experts to find out the most informative indicators to make a better prediction. The prediction of the market value is of great importance to help in maximizing the profit of stock option purchase while keeping the risk low. Recurrent neural networks (RNN) have proved one of the most powerful models for processing sequential data. Long Short-Term memory is one of the most successful RNNs architectures. LSTM introduces the memory cell, a unit of computation that replaces traditional artificial neurons in the hidden layer of the network. With these memory cells, networks are able to effectively associate memories and input remote in time, hence suit to grasp the structure of data dynamically over time with high prediction capacity.

In the obtained historical prices, we have the following information for each of the asset:

* ****Date****: Date
* ****Open****: Open price within a date
* ****High****: The highest price within a date
* ****Low****: The lowest price within a date
* ****Close****: Close price within a date
* **NAME*Adj***close****: Adjusted close price in the end of a date
* ****Volume****: Trading volume

**3.2 Python Dependency Libraries**

**from** scipy.stats **import** skew

**from** scipy.stats **import** kurtosis

**from** statsmodels.tsa.stattools **import** kpss

**from** sklearn.linear\_model **import** Ridge, Lasso

**from** keras.layers **import** Dense

**from** keras.layers **import** LSTM

**from** keras.layers **import** Dropout

**from** pandas\_datareader **import** data as pdr

**import** fix\_yahoo\_finance as yf

**from** sklearn.preprocessing **import** MinMaxScaler

**from** sklearn.metrics **import** mean\_squared\_error

**from** sklearn.metrics **import** mean\_absolute\_error

**from** keras.callbacks **import** EarlyStopping

|  |
| --- |
| Rest all the imports I have showed in my Jupyter Notebook, which I gave hyperlink of my Github Account of this chapter.Note:plotly==4.0.0 plotly-express==0.4.1 Anaconda Package Python 2.x/3.x, TensorFlow, Keras.  **Note**  CodeRepository: <https://github.com/aniruddhachoudhury/Artificial-Intelligence-Projects-/tree/master/Chapter3> |

We extracted Daterange from yahoo finance API and we can provide our own range its better to take 6 years data.

stocks\_start = datetime.datetime(2014, 1, 1)

stocks\_end = datetime.datetime(2019, 1, 1)

tickers=['AMZN','GE','F','CAT']

**def get**(tickers, startdate, enddate):

**def data**(ticker):

return (pdr.get\_data\_yahoo(ticker, start=startdate, end=enddate))

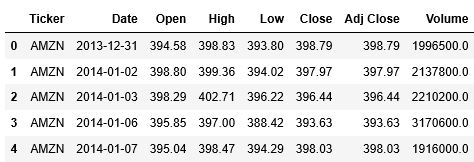
datas = **map**(data, tickers)

**return**(pd.concat(datas, keys=tickers, names=['Ticker', 'Date']))

all\_data = get(tickers, stocks\_start, stocks\_end)

all\_data=all\_data.reset\_index()

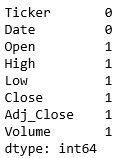
all\_data.head()

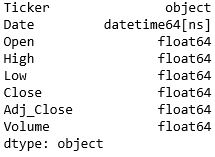


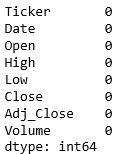
**Table:** *Table 3.1:* *Stock Market Data-set Creation table*

We checked the Datatype and Null Value Analysis and filled those with forward fill method.

all\_data.dtypes

all\_data.isna().sum()



all\_data.fillna(method='ffill',inplace=**True**)

all\_data.isna().sum()

Created a function for adding log-returns column to dataframes.

A return can be expressed nominally as the change in dollar value of an investment over time. A return can also be expressed as a percentage derived from the ratio of profit to investment. Returns can also be presented as net results (after fees, taxes, and inflation) or gross returns that do not account for anything but the price change.

**def add\_log\_returns**(df):

adj\_closing\_price = df['Adj\_Close']

log\_array = np.log(np.array(adj\_closing\_price))

log\_return\_array = log\_array - np.append(log\_array[1:], np.nan)

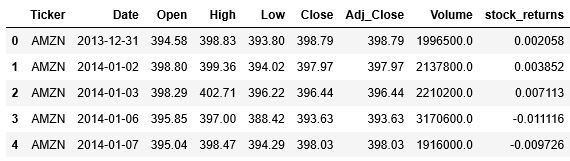
log\_return\_df = pd.DataFrame(log\_return\_array,columns=['stock\_returns'])

df['stock\_returns']=log\_return\_df['stock\_returns']

return df

all\_data=add\_log\_returns(all\_data)

all\_data.head()

**Table:** *Table 3.2:* *Stock Market Data-set with Returns*

We present two functions. One to create a dataframe consisting of only adjusted close prices of our assets, while the other one is to create a dataframe consisting of only asset returns.

**def asset\_adj\_close**(df,tickers):

f=pd.DataFrame()

f=df[df['Ticker']== 'AMZN']

f=f[['Date']]

**for** i **in** tickers:

x =df[df['Ticker']== **i**]

x= x[['Date','Close']]

x.rename(columns={'Close': '%sClose' % **i**}, inplace=True)

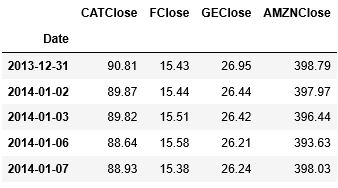
x.reset\_index(drop=**True**)

f = pd.merge(x, f, on='Date',how="left")

**return** f

adj\_close = asset\_adj\_close(all\_data, tickers)

adj\_close.head()



**Table:** *Table 3.3:* *Stock Market Adjusted Close Table*

**def asset\_returns**(df, tickers):

f=pd.DataFrame()

f=df[df['Ticker']== 'AMZN']

f=f[['Date']]

for i in tickers:

x =df[df['Ticker']== i]

x= x[['Date','stock\_returns']]

x.rename(columns={'stock\_returns': '%sreturns' % **i**}, inplace=**True**)

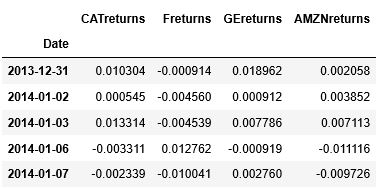
x.reset\_index(drop=**True**)

f = pd.merge(x, f, on='Date')

**return** f

returns = asset\_returns(all\_data, tickers)

returns.head()



**Table:** *Table 3.4:* *Stock Market Returns Table*

**3.3  Classical Statistics ARIMA Modelling**

Here we provide a correlation table and a scatter plot against each other for both prices and returns of General Electric(GE), Ford Motors(F),Caterpillar(CAT), Amazon (AMZN), . Comparison of Pearson and Spearman correlations let us know the affection of large outliers on general picture of the assets movement.

**3.3.1  Asset Returns & Asset Prices**

**def correlation**(df,lis,days):

**for** i **in** lis:

correlation = df[-days:].corr(method=i)

matrix\_cols = correlation.columns.tolist()

corr\_array = np.array(correlation)

trace = go.Heatmap(z = corr\_array,x = matrix\_cols,y = matrix\_cols,

xgap = 2,ygap = 2,colorscale='Viridis',colorbar = dict())

layout = go.Layout(dict(title = '{} correlation'.format(i),

autosize = **False**,height = 720,width = 800,

margin = **dict**(r = 0 ,l = 210,

t = 25,b = 210),

yaxis = **dict**(tickfont = **dict**(size = 9)),

xaxis = **dict**(tickfont = **dict**(size = 9)),

))

fig = go.Figure(data = [trace],layout = layout)

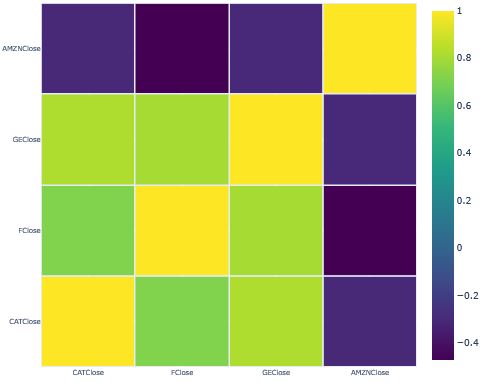
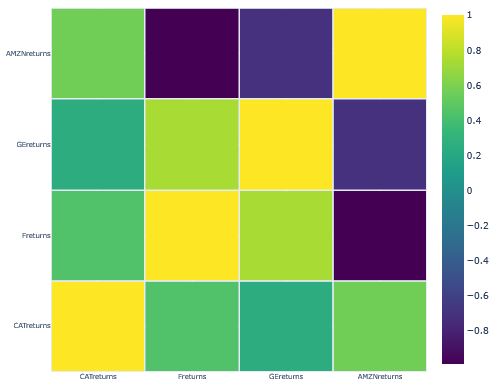
fig.show()

lis=['pearson','spearman']

correlation(adj\_close,lis,days=252)

correlation(returns,lis,days=252)

So below matrix is for pearson for Adj\_clos & Returns resp.

****

*Figure 3.1: Figure 3.2:*

From the figure right above one can see that correlation between asset returns does not change much between two types of correlations. The returns tend to comoving within 252 days time interval.

**def pairplot**(df, days=252, width=9, height=9):

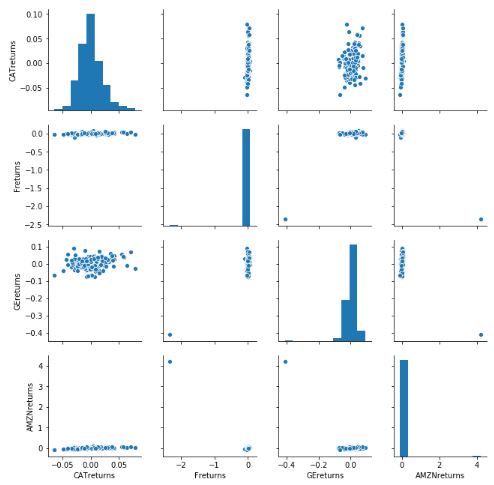
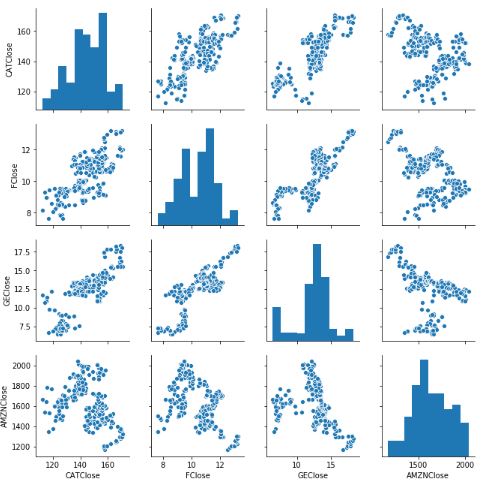
g = sns.pairplot(df[-days:]);

g.fig.set\_size\_inches(width,height)

plt.show()

pairplot(returns, width=9, height=9)

pairplot(returns, width=9, height=9)

****

*Figure 3.3: Figure 3.4:*

The above is the scatter plot for adjusted close prices. This allows us to see general tendency of asset pairs.

According to the correlation analysis, the assets considered here tend to comoving. However, we should also remember what time-interval is considered. Here we take into account 252 days interval corresponding to about 1 trading year. If some other time interval is of interest, then correlation coefficients for all of the assets change appropriately. It is not possible to know in advance whether the correlation between the assets will be higher, remain the same or lower with changing a time-period. But such an analysis demonstrates an estimation of their possible correlation values.

**3.3.1.1 Asset Prices**

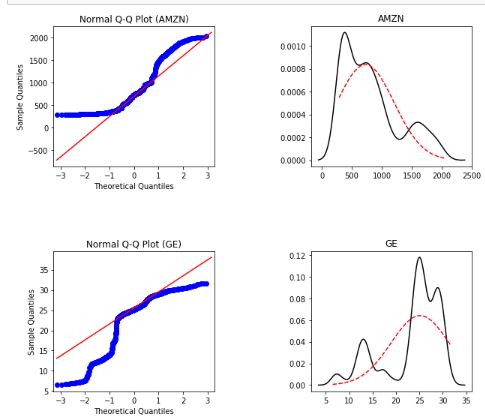
We have created a function and called it here to qqplot\_returns in you can find out in Github the code so don’t get confuse if you see this method below.

**def norm\_function**(x, mu, sigma):

**return** (1**/**(sigma\*np.sqrt(2\*np.pi))) \* np.exp(-(x-mu)\*\*2 / (2\*sigma\*\*2))

qqplot\_returns(adj\_close, tickers, option='Close',

size=(10,20), sharex=**False**, sharey=**False**, wspace=0.6, hspace=0.6)

On the left-hand side the normal quantile-quantile plots are shown, while on the right-hand side kernel density estimate of returns and their closest normal distribution are illustrated. Red lines correspond to normal distribution.

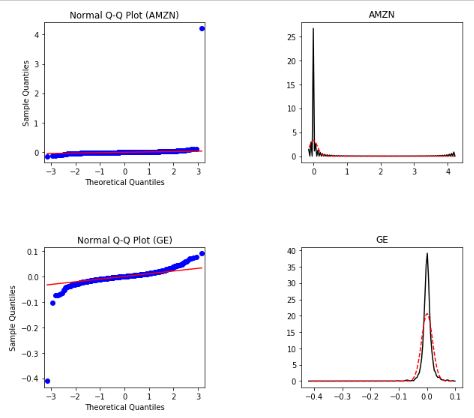
We see that some of adjusted close prices have bi-modal distributions while some others have even more complex structure which does not appear to be normal even close.

*Figure 3.5*

**3.3.1.2  Asset Returns***:*

qqplot\_returns(returns, tickers, option='returns',

size=(10,20), sharex=**False**, sharey=**False**, wspace=0.6, hspace=0.6)

****Red lines correspond to normal distribution. Note that the distribution is not normal as demonstrated by both kinds of plots demonstrating fatter tails and higher kurtosis. However, their structure is closer to the normal distribution than that of adjusted close price.

On this stage of the work we may conclude that fat tails will become a problem for our ARIMA or modelling since it may be that we will not encompass all the time-series information due to that. Let's keep this notion in mind and move on.

*Figure 3.6*

**3.3.2  Skewness, Kurtosis, Max value, Min value, Mean and Variance**

let's gather more statistics about target values. Precisely speaking, their Skewness, Kurtosis, Max returns, Max loss, Mean and Variances.

**def stats**(df, symbols):

stat = pd.DataFrame(index=symbols, columns=['Skewness','Kurtosis','Max value',

'Min value','Mean','Variance'])

stat['Skewness'] = skew(df, axis=0)

stat['Kurtosis'] = kurtosis(df, axis=0)

stat['Max value'] = df.agg('max').values

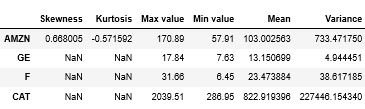
stat['Min value'] = df.agg('min').values

stat['Mean'] = df.agg('mean').values

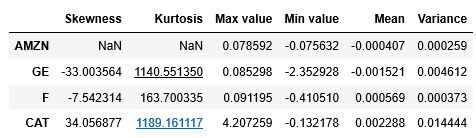
stat['Variance'] = df.agg('var').values

return stat

stats(adj\_close.loc[:, ['CATClose', 'FClose', 'GEClose', 'AMZNClose']], tickers)



stats(returns.loc[:, [ 'CATreturns', 'Freturns', 'GEreturns', 'AMZNreturns']], tickers)

****

**3.3.3  Time-series plots**

**Asset Prices**

**def plot\_prices**(df, symbols, width=1000, height=400):

traces = []

for asset in symbols:

trace = go.Scatter(

x=df.index,

y=df['%sClose' % asset],

name = '%s price' % asset)

traces.append(trace)

layout = go.Layout(title='Adj close price history',

autosize=False,

width=width,

height=height)

fig = go.Figure(data=traces, layout=layout)

return iplot(fig, show\_link=False)

plot\_prices(adj\_close, tickers)



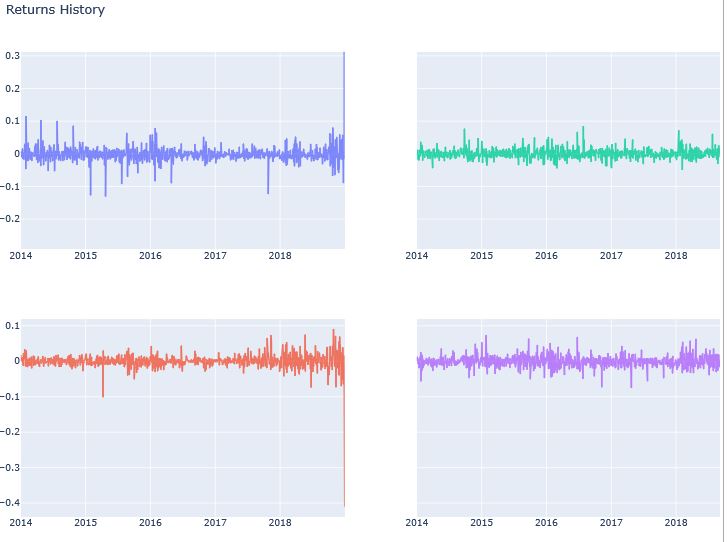
*Figure 3.7*

**Asset Prices**

It might be useful to see all the return values separately.

We have created a function and called it here to plot\_returns\_indiv in plolty you can find out in Github the code so don’t get confuse if you see this method below.

plot\_returns\_indiv(returns, tickers);

**

**

*Figure 3.8*

From the figures of the return time-series one can see that The assets do not have long-term deviations from the mean and are mainly oscillating over some constant values that is almost zero. This is a property of stationary process. There are periods of high volatility followed by periods of relatively tranquility (volatility clustering). One can also notice that volatility clustering does not indicate a lack of stationarity but rather can be viewed as a type of dependence in the conditional variance of each series.

**3.3.3.1  Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF)**

When a time series is observed, a natural question is whether it appears to be stationary. In this section, we plot ACF and PACF in order to check the series on the presence of stationarity. We also perform Augmented Dickey-Fuller test as well as Kwiatkowski-Phillips-Schmidt-Shin test. The former tests null hypothesis that timeseries is not stationary against an alternative hypothesis that the time-series is stationary, while the latter test is designed to test null stationarity against an alternative of non-stationarity.

**def stationary\_analysis**(df, symbols, lags, option='price', figsize=(10,20), wspace=0.3, hspace=0.5):

nrows = df.shape[1]

fig, axes = plt.subplots(nrows=nrows, ncols=2, figsize=figsize)

plt.subplots\_adjust(wspace=0.3, hspace=0.5)

row = 0

**for** asset **in** symbols:

if option == 'price':

smt.graphics.plot\_acf(df['%sClose' % asset], lags=lags, ax=axes[row,0])

smt.graphics.plot\_pacf(df['%sClose' % asset], lags=lags, ax=axes[row,1])

axes[row,0].set\_title('%s Adj close (ACF)' % asset)

axes[row,1].set\_title('%s Adj close (PACF)' % asset)

row += 1

**elif** option == 'return':

smt.graphics.plot\_acf(df['%sreturns' % asset], lags=lags, ax=axes[row,0])

smt.graphics.plot\_pacf(df['%sreturns' % asset], lags=lags, ax=axes[row,1])

axes[row,0].set\_title('%s returns (ACF)' % asset)

axes[row,1].set\_title('%s returns (PACF)' % asset)

row += 1

**return** plt.show()

def **adf\_kpss**(df, symbols, option='price'):

adf\_kpss = pd.DataFrame(index=asset\_names, columns=['ADF','KPSS'])

**if** option == 'price':

adf\_price = []

kpss\_price = []

for i in range(**len**(symbols)):

adf\_price.append(ts.adfuller(adj\_close.iloc[:,i])[1])

kpss\_price.append(kpss(adj\_close.iloc[:,i])[1])

adf\_kpss['ADF'] = np.array(adf\_price)

adf\_kpss['KPSS'] = np.array(kpss\_price)

elif option == 'returns':

adf\_returns = []

kpss\_returns = []

**for** i in range(len(asset\_names)):

adf\_returns.append(ts.adfuller(returns.iloc[:,i])[1])

kpss\_returns.append(kpss(returns.iloc[:,i])[1])

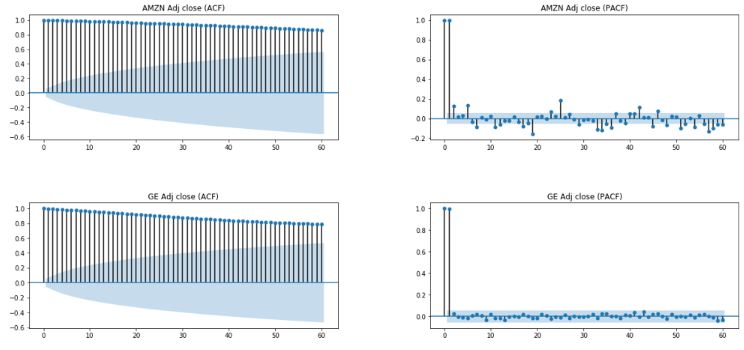
adf\_kpss['ADF'] = np.array(adf\_returns)

adf\_kpss['KPSS'] = np.array(kpss\_returns)

**return** adf\_kpss

**Asset Prices**

stationary\_analysis(adj\_close, tickers, option='price', lags=60, figsize=(20,20))

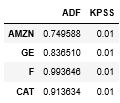
****

*Figure 3.9*

adf\_kpss\_price['ADF'] = np.array(adf\_price)

adf\_kpss\_price['KPSS'] = np.array(kpss\_price)

adf\_kpss\_price

****

Auto-correlation Function (ACF) and Partial Auto-correlation Function (PACF) for four-price time-series General Electric(GE), Ford Motors(F),Caterpillar(CAT), Amazon (AMZN), .One can see that each ACF shows a very slow (barely noticeable), linear decay pattern which is typical of a non-stationary time series. Besides, PACF has one significant spike at lag 1 (lag 0 is not accounted for) meaning that (almost) all the higher-order autocorrelations are effectively explained by the lag 1 autocorrelation.

Similarly we have done it for Asset Returns.

**3.3.4  Metrics and Model selection**

Here we build ARIMA model for stationary time-series of asset returns following the steps: Identify p and q values of ARIMA by using grid search incorporating Akaike information criterion metrics.The function to fit ARIMA(p, d, q) model & Selection of best order and final model based on aic

**def arima\_model**(df, symbol):

asset = df['%sreturns' % symbol]

print(df.columns)

best\_aic = np.inf

best\_order = None

best\_mdl = None

pq\_rng = **range**(6)

d\_rng = **range**(1)

**for** p in pq\_rng:

**for**d in d\_rng:

**for** q in pq\_rng:

**try**:

tmp\_mdl = smt.ARIMA(asset, order=(p,d,q)).fit(method='mle', trend='nc')

tmp\_aic = tmp\_mdl.aic

if tmp\_aic < best\_aic:

best\_aic = tmp\_aic

best\_order = (p, d, q)

best\_mdl = tmp\_mdl

**except**: **continue**

**return** best\_aic, best\_order, best\_mdl

def **tsplot**(y, lags=**None**, figsize=(10, 8), style='bmh', name='asset'):

Residual=pd.DataFrame(y)

Residual=Residual[:-1]

**import** plotly.express as px

fig = px.line(y, x=Residual.index, y=Residual.values)

fig.update\_layout(title='Time Series Analysis Plots: %s' % name,

xaxis\_title='Date',

yaxis\_title='Residual Error',

height=300, width=1000,

legend=**dict**(x=-.1, y=1.5))

fig.show()

**if** not isinstance(y, pd.Series):

y = pd.Series(y)

**with** plt.style.context(style):

fig = plt.figure(figsize=figsize)

layout = (2, 2)

acf\_ax = plt.subplot2grid(layout, (0, 0))

pacf\_ax = plt.subplot2grid(layout, (0, 1))

qq\_ax = plt.subplot2grid(layout, (1, 0))

pp\_ax = plt.subplot2grid(layout, (1, 1))

smt.graphics.plot\_acf(y, lags=lags, ax=acf\_ax, alpha=0.5)

smt.graphics.plot\_pacf(y, lags=lags, ax=pacf\_ax, alpha=0.5)

sm.qqplot(y, line='s', ax=qq\_ax)

qq\_ax.set\_title('QQ Plot')

scs.probplot(y, sparams=(y.mean(), y.std()), plot=pp\_ax)

plt.tight\_layout()

We will remove the last rows & will fit the model to find optimal p,d,q with Low AIC value as the returns cant differenced with the next row.

returns=returns[:-1]

aic\_dict = {}

order\_dict = {}

mdl\_dict = {}

**for** symbol **in** tickers:

aic\_dict[symbol], order\_dict[symbol], mdl\_dict[symbol] = arima\_model(returns, symbol)

**print**(aic\_dict)

**print**(order\_dict)

**Capture**

We fitted the parameters of p,d,q to the dataset and will set the residuals plot

mdl\_dict['GE'] = smt.ARIMA(returns.GEreturns, order=(5,1,2)).fit(method='mle', trend='nc')

mdl\_dict['CAT'] =smt.ARIMA(returns.CATreturns, order=(1,0,0)).fit(method='mle', trend='nc')

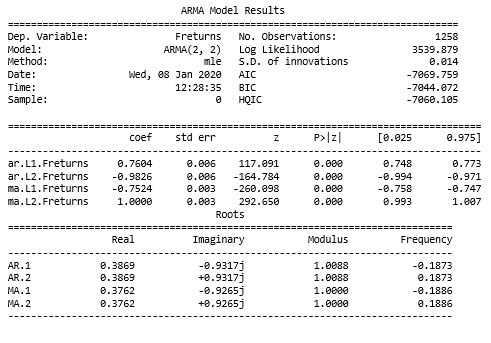
mdl\_dict['F'] = smt.ARIMA(returns.Freturns, order=(2,0,2)).fit(method='mle', trend='nc')

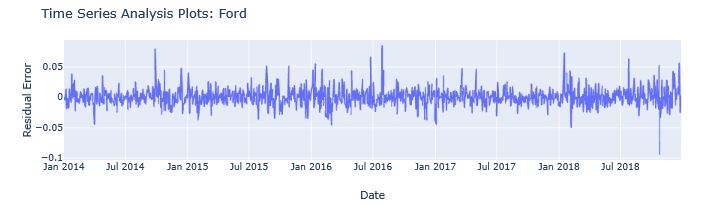
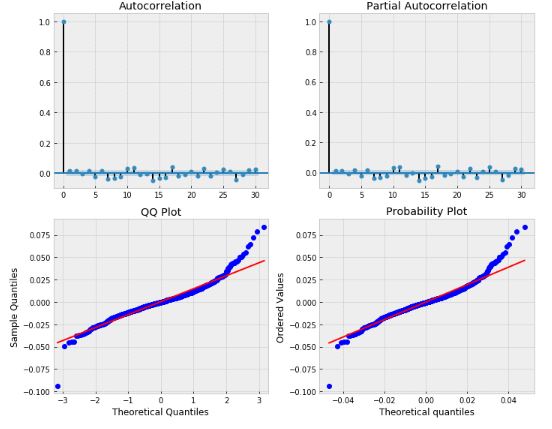
mdl\_dict['AMZN'] = smt.ARIMA(returns.AMZNreturns, order=(3,0,2)).fit(method='mle', trend='nc')

Here we will take one Asset case rest I have showed in my Notebook. We will take Ford into our consideration.

**print**(mdl\_dict['F'].summary())

tsplot(mdl\_dict['F'].resid, lags=30, name='Ford')



****

*Figure 3.10*

**ARMA predictions:**

**for** symbol **in** ['GE','AMZN','F','CAT']:

fig = go.Figure()

ts = returns['%sreturns' % symbol].iloc[-255:-1].copy()

pred = mdl\_dict[symbol].predict()[-255:-1]

n\_steps=30

f, err95, ci95 = mdl\_dict[symbol].forecast(steps=n\_steps) # 95% CI

\_, err99, ci99 = mdl\_dict[symbol].forecast(steps=n\_steps, alpha=0.01) # 99% CI

idx = pd.date\_range(returns.index[-1], periods=n\_steps, freq='D')

fc\_95 = pd.DataFrame(np.column\_stack([f, ci95]), index=idx, columns=['forecast', 'lower\_ci\_95', 'upper\_ci\_95'])

fc\_99= pd.DataFrame(np.column\_stack([ci99]), index=idx, columns=['lower\_ci\_99', 'upper\_ci\_99'])

fc\_all = fc\_95.combine\_first(fc\_99)

fc\_all.index = pd.to\_datetime(fc\_all.index)

fig.add\_trace(go.Scatter(x=ts.index,y=ts.values,fill=**None**,mode='lines',line \_color='blue',name='%sReturns'% symbol))

fig.add\_trace(go.Scatter(x=pred.index, y=pred.values,fill=**None**,mode='lines',line\_color='yellow',name='Sample Prediction'))

fig.add\_trace(go.Scatter(x=fc\_all.index, y=fc\_all.upper\_ci\_95,fill=**None**,line\_color='black',name='Upper\_CI\_95'))

fig.add\_trace(go.Scatter(x=fc\_all.index,y=fc\_all.lower\_ci\_95,fill='tonexty', mode='lines', line\_color='black',name='Lower\_CI\_95'))

fig.add\_trace(go.Scatter(x=fc\_all.index, y=fc\_all.upper\_ci\_99,fill=**None**,mode='lines',line\_color='gray',name='Upper\_CI\_99'))

fig.add\_trace(go.Scatter(x=fc\_all.index,y=fc\_all.lower\_ci\_99,fill='tonexty',mode='lines', line\_color='gray',name='Lower\_CI\_99'))

fig.add\_trace(go.Scatter(x=fc\_all.index,y=fc\_all.forecast,fill=**None**,mode='li nes',line\_color='red',name='Forecast'))

fig.update\_layout(title='{} Day {} Return Forecast\nARIMA{}'.format(n\_steps, symbol, order\_dict[symbol]),

xaxis\_title='Date Range',

yaxis\_title='Returns',

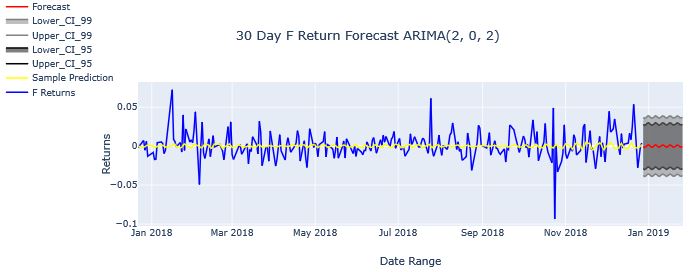
height=400, width=1000,

legend=dict(x=-.25, y=1.6))

fig.update\_layout(title={‘y':0.85,

'x':0.5,'xanchor': 'center','yanchor': 'top'})

fig.show()

*Figure 3.11*

So the returns of ford 30 steps predicting pretty Good. And similarly we can check for Amazon,Caterpillar , GE. And we gave a confidence interval between 95% to 99% range.

**3.4  Hybrid Machine Learning with Time Series**

**3.4.1  Data preprocessing, cross-validation, adjustment of model hyperparameters**

**We will take Adjusted close price predictions with Ridge and Lasso.We have a function to split the dataset into train and test.**

****def timeseries\_train\_test\_split**(X, y, test\_size):**

**test\_index = **int**(len(X)\*(1-test\_size))**

**X\_train = X.iloc[:test\_index]**

**y\_train = y.iloc[:test\_index]**

**X\_test = X.iloc[test\_index:]**

**y\_test = y.iloc[test\_index:]**

****return** X\_train, X\_test, y\_train, y\_test**

****def prepareData**(series, lag\_start=1, lag\_end=20, test\_size=0.2):**

**data = pd.DataFrame(series.copy())**

**data.columns = ["y"]**

**# lags of series**

****for** i in **range**(lag\_start, lag\_end):**

**data["lag\_{}".format(i)] = data.y.shift(i)**

**# datetime features**

**data['year'] = data.index.year**

**data['month'] = data.index.month**

**data['day'] = data.index.day**

**data['weekday'] = data.index.weekday**

**data['season'] = data['month'].apply(lambda x: '1' **if** x **in** [12,1,2] else\**

**('2' **if** x **in** [3,4,5] else ('3' **if** x in [6,7,8] **else** '4')))**

**time\_feat = ['year','month','day','weekday','season']**

**time\_feat\_df = pd.DataFrame(data[time\_feat], index=data.index)**

**time\_feat\_df[time\_feat] = time\_feat\_df[time\_feat].astype('str')**

**time\_feat\_df = pd.get\_dummies(time\_feat\_df)**

**data = pd.concat((data, time\_feat\_df), axis=1)**

**data.drop(columns=time\_feat, inplace=True)**

**y = data.dropna().y**

**X = data.dropna().drop(['y'], axis=1)**

**X\_train, X\_test, y\_train, y\_test = timeseries\_train\_test\_split(X, y, test\_size=test\_size)**

****return** X\_train, X\_test, y\_train, y\_test**

**We scaled our features to same scale for better prediction.**

****def feature\_scaling**(series, lag\_start=1, lag\_end=20, test\_size=0.2):**

**X\_train, X\_test, y\_train, y\_test = prepareData(series, lag\_start=lag\_start, \**

**lag\_end=lag\_end, test\_size=test\_size)**

**scaler = StandardScaler()**

**X\_train\_scaled = scaler.fit\_transform(X\_train)**

**X\_test\_scaled = scaler.transform(X\_test)**

**return X\_train\_scaled, X\_test\_scaled, y\_train, y\_test**

**time\_split = TimeSeriesSplit(n\_splits=3)**

**We will do the grid search finding the beat parameter for our model to predict best outcome.**

****def grid\_search**(estimator, X, y, grid\_param, scoring='neg\_mean\_absolute\_error', idd=False, cv=time\_split):**

**gsearch = GridSearchCV(estimator = estimator,**

**param\_grid = grid\_param,**

**scoring=scoring,**

**iid=idd,**

**cv=cv,**

**n\_jobs=-1,**

**verbose=**True**)**

**gsearch.fit(X, y)**

****return** gsearch**

**To Calculate the MAPE below is the function to judge how good our model is predicting.**

****def** mean\_absolute\_percentage\_error(y\_true, y\_pred):**

****return** np.mean(np.abs((y\_true - y\_pred) / y\_true)) \* 100**

**Hyperparameters for grid search**

param\_ridge = {'alpha': [25,10,4,2,1.0,0.8,0.5,0.3,0.2,0.1,0.05,0.02,0.01]}

param\_lasso = {'alpha': [25,10,4,2,1.0,0.8,0.5,0.3,0.2,0.1,0.05,0.02,0.01]}

**3.4.2  FORD Prediction**

**FORD RIDGE**

X\_F\_train\_scaled,X\_F\_test\_scaled,y\_F\_train,y\_F\_test=feature\_scaling(

adj\_close.FClose)

ridge\_F = grid\_search(estimator = Ridge(),X=X\_F\_train\_scaled,

y=y\_F\_train, grid\_param=param\_ridge)

**Capture**

We have created a function and called it here to prediction\_plot in plotly you can find out in Github the code so don’t get confuse if you see this method below.

prediction\_plot(ridge\_F, X\_F\_test\_scaled, y\_F\_test, asset\_name='Ford Ridge')

****

*Figure 3.12*

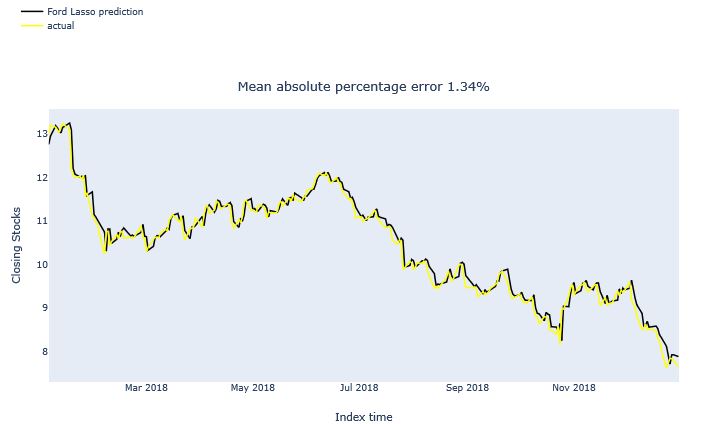
**FORD LASSO**

lasso\_F = grid\_search(estimator = Lasso(),X=X\_F\_train\_scaled,

y=y\_F\_train, grid\_param=param\_lasso, cv=time\_split)

**Capture**

prediction\_plot(lasso\_F, X\_F\_test\_scaled, y\_F\_test,asset\_name='Ford Lasso')

****

*Figure 3.13*

So from above Lasso and Ridge prediction for the asset ford we can see Lasso is predicting pretty good in comparison to the Ridge is capturing the results with good regularization effect.

Similarly we have done for the rest of the assets GE,AMZN,CAT vice-versa.

**3.5  LSTM Architecture**

**3.5.1  Close Volume insights and library upload**

Here we selected Ford asset to predict is closing stocks.

print( 'Kurtosis of normal distribution: {}'.format(scs.kurtosis(adj\_close.FClose)))

print( 'Skewness of normal distribution: {}'.format(scs.skew(adj\_close.FClose)))

Capture

v\_features=list(adj\_close.columns)

for i, cn in **enumerate**(adj\_close[v\_features[:]]):

group\_labels = ['Stocks']

colors = ['magenta']

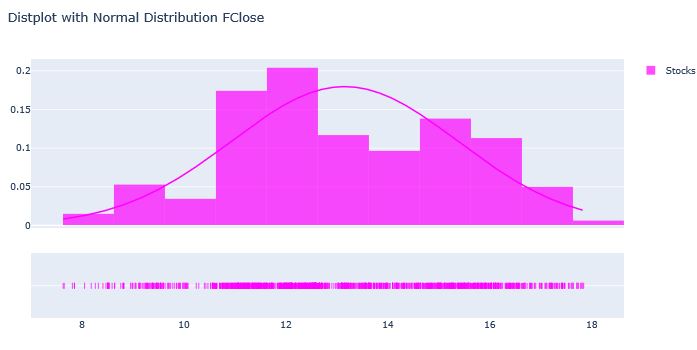
x1 = adj\_close[cn]

fig = ff.create\_distplot([x1], group\_labels, bin\_size=1,

curve\_type='normal', colors=colors)

fig.update\_layout(title\_text='Distplot with Normal Distribution '+str(cn))

fig.show()

****

*Figure 3.14*

**3.5.2  Data Creation for Training[¶](http://localhost:8888/notebooks/Downloads/Book BPP/Time Series/Deep Learning Time Series.ipynb" \l "Data-Creation-for-Training)**

dataset = adj\_close.FClose.values #numpy.ndarray

dataset = dataset.astype('float32')

dataset = np.reshape(dataset, (-1, 1))

scaler = MinMaxScaler(feature\_range=(0, 1))

dataset = scaler.fit\_transform(dataset)

train\_size = int(len(dataset) \* 0.80)

test\_size = len(dataset) - train\_size

train, test = dataset[0:train\_size,:], dataset[train\_size:len(dataset),:]

**We can use TimeSeries Generator to make the datasets for LSTM or convert an array of values into a dataset matrix via numpy**

****def** create\_dataset(dataset, look\_back=1):**

**X, Y = [], []**

****for** i in **range**(**len**(dataset)-look\_back-1):**

**a = dataset[i:(i+look\_back), 0]**

**X.append(a)**

**Y.append(dataset[i + look\_back, 0])**

****return** np.array(X), np.array(Y)**

**We reshaped into X=t and Y=t+1 and** here lookback means it will see previous 12 monsth data to predict 13th month to capture Seasonality

**look\_back = 12**

**X\_train, Y\_train = create\_dataset(train, look\_back)**

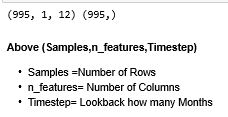
**X\_test, Y\_test = create\_dataset(test, look\_back)**

print(X\_train.shape,Y\_train.shape)

Capture

We reshaped input to be [samples, time steps, features]

X\_train = np.reshape(X\_train, (X\_train.shape[0], 1, X\_train.shape[1]))

X\_test = np.reshape(X\_test, (X\_test.shape[0], 1, X\_test.shape[1]))

**3.5.3  Model Training**

model = Sequential()

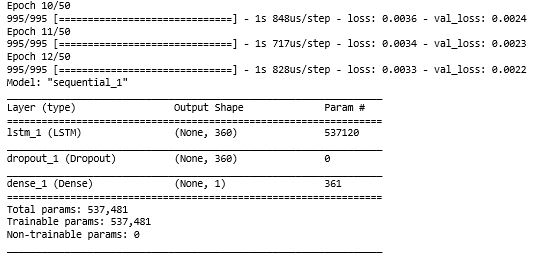
model.add(LSTM(360,activation='relu',input\_shape=(X\_train.shape[1], X\_train.shape[2])))

model.add(Dropout(0.30))

model.add(Dense(1))

model.compile(loss='mean\_squared\_error', optimizer='adam')

history = model.fit(X\_train, Y\_train, epochs=50, batch\_size=32, validation\_data=(X\_test, Y\_test),callbacks=[EarlyStopping(monitor='val\_loss', patience=10)], verbose=1, shuffle=False)

****

# Training Phase

model.summary()

**3.5.4  LSTM Model Evaluation & Visualization**

train\_predict = model.predict(X\_train)

test\_predict = model.predict(X\_test)

# invert predictions

train\_predict = scaler.inverse\_transform(train\_predict)

Y\_train = scaler.inverse\_transform([Y\_train])

test\_predict = scaler.inverse\_transform(test\_predict)

Y\_test = scaler.inverse\_transform([Y\_test])

**print**('Train Mean Absolute Error:', mean\_absolute\_error(Y\_train[0], train\_predict[:,0]))

**print**('Train Root Mean Squared Error:',np.sqrt(mean\_squared\_error(Y\_train[0], train\_predict[:,0])))

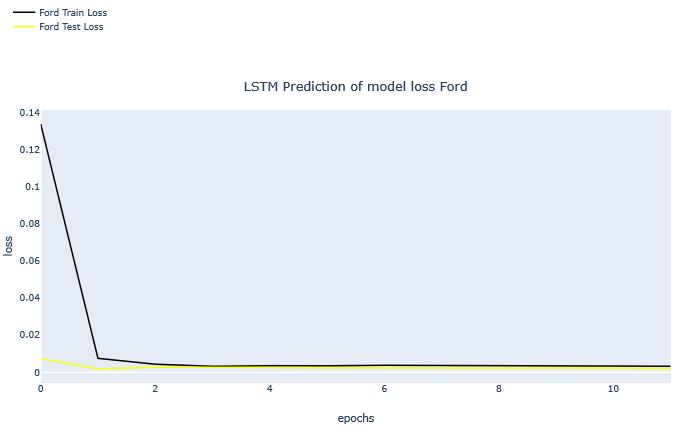
**print**('Test Mean Absolute Error:', mean\_absolute\_error(Y\_test[0], test\_predict[:,0]))

**print**('Test Root Mean Squared Error:',np.sqrt(mean\_squared\_error(Y\_test[0], test\_predict[:,0])))

Capture

We have created a function and called it here to Model\_Loss\_plot in plotly you can find out in Github the code so don’t get confuse if you see this method below.

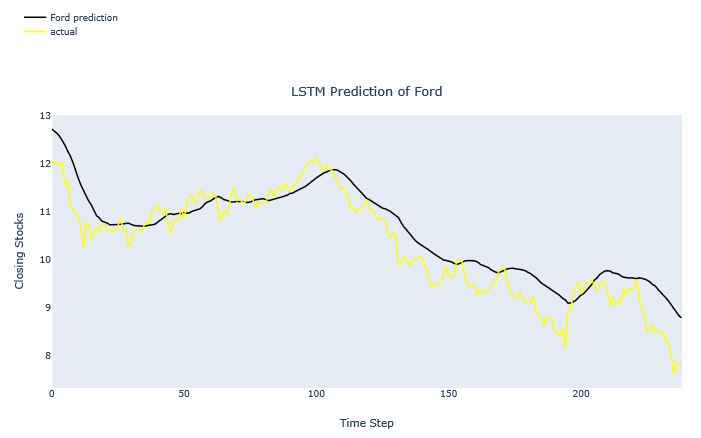
Model\_Loss\_plot(history,asset\_name = 'Ford')



*Figure 3.15*

We have created a function and called it here to prediction\_LSTM in plotly you can find out in Github the code so don’t get confuse if you see this method below.

prediction\_LSTM(test\_predict, Y\_test,asset\_name = 'Ford')

*Figure 3.16*

The Model Predicting pretty good with .47 ****RMSE**** consider that we can predict for Similar for other Company. So LSTM

**3.6 Summary**

* Here we have learnt complete end to end analysis on ARIMA/ARMA model of classical statistics with its prediction
* Then we have learnt how to implement Machine learning Ridge and LASSO to predict the stocks and evaluated thos ewith various parameter s and grid search for beetr accuracy and MAPE.
* Agarin we have learnt a new Deep learning Architecture LSTM to predict the stocks and various techniques to create the LSTM Framework.
* We built the various advanced plots to analyse the stocks and returns.