In [ ]:

```
'''Stock market prices are very unpredictable as they are volatile, dynamic and non-linear
factors, such as politics, global economic conditions, unexpected events, a company's finan
impacting it and thus making it a very challenging task.

Traditionally analysts have relied on quantitative finance methodology for stock price pred
with the onset of recent advancements in machine learning applications, the field has evolv
solutions that learn what is going on to make accurate predictions. For this project I have
algorithms to predict the stock prices and evaluate their performance.'''
```

In [ ]:

```
'''In general, apart from the historical price of the stock itself, the features that are g
stock price prediction are as below:

•   Correlated Assets: An organization depends on and interacts with many external factors,
the global economy, the geopolitical situation, fiscal and monetary policies, access to cap
price may be correlated not only with the stock price of other companies but also with othe
FX, broad-based indices, or even fixed income securities;

•   Technical indicators: A lot of investors follow technical indicators. Moving average, e
momentum are the most popular indicators;

•   Other Factors:
    -   Performance reports: Annual and quarterly reports of companies can be used to extra
    ROE (Return on Equity) and P/E (Price-to-Earnings);
    -   News: News can indicate upcoming events that can potentially move the stock price i

I have used various supervised learning-based models to predict the stock price of Microsof
historical data. Visualization of the data using different kinds of charts (i.e., density,
Used deep learning (LSTM) models for time series forecasting.Interpretation of the results
and underfitting of the data across the models.In the supervised regression framework, the
the predicted variable. We need to understand what affects Microsoft stock price and incorp
model. Out of correlated assets, technical indicators, and other factors, I have focused on
important features for the prediction.

For this project, other than the historical data of Microsoft, the independent variables us
correlated assets: * Stocks : IBM (IBM) and Alphabet (GOOGL); * Currency : USD/JPY and GBP/
Indices : S&P 500, Dow Jones, and VIX;
'''
```

In [5]:

```
import numpy as np
import pandas as pd
import pandas_datareader.data as web
from matplotlib import pyplot
from pandas.plotting import scatter_matrix
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from pandas.plotting import scatter_matrix
from statsmodels.graphics.tsaplots import plot_acf
```

In [6]:

```
#Function and modules for the supervised regression models

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.neural_network import MLPRegressor
```

In [10]:

```
#Function and modules for data analysis and model evaluation

from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2, f_regression
```

In [14]:

```
#Function and modules for deep learning models

from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.layers import LSTM
from tensorflow.keras.wrappers.scikit_learn import KerasRegressor
```

In [15]:

```
#Function and modules for time series models
from statsmodels.tsa.arima.model import ARIMA

#from statsmodels.tsa.statespace.sarimax import SARIMAX
import statsmodels.api as sm

#Libraries for Saving the Model
from pickle import dump
from pickle import load

#Diable the warnings
import warnings
warnings.filterwarnings('ignore')
```

In [16]:

```
# Load the data

stk_tickers = ['MSFT', 'IBM', 'GOOGL']
ccy_tickers = ['DEXJPUS', 'DEXUSUK']
idx_tickers = ['SP500', 'DJIA', 'VIXCLS']
stk_data = web.DataReader(stk_tickers, 'yahoo')
ccy_data = web.DataReader(ccy_tickers, 'fred')
idx_data = web.DataReader(idx_tickers, 'fred')
```

In [ ]:

```
return_period = 5
```

return_period = 5 ''' we choose to predict using weekly returns. Hence have approximated this by using the 5 business day period returns.'''

```
# We now define our Y series and our X series
Y: MSFT Future Returns (Dependent variable)
X: (Independent variables)
a. GOOGL 5 Business Day Returns
b. IBM 5 Business DayReturns
c. USD/JPY 5 Business DayReturns
d. GBP/USD 5 Business DayReturns
e. S&P 500 5 Business DayReturns
f. Dow Jones 5 Business DayReturns
g. MSFT 5 Business Day Returns
h. MSFT 15 Business Day Returns
i. MSFT 30 Business Day Returns
j. MSFT 60 Business Day Returns
```

In [19]:

```
Y = np.log(stk_data.loc[:, ('Adj Close', 'MSFT')]).diff(return_period).shift(-return_period
Y.name = Y.name[-1]+'_pred'

X1 = np.log(stk_data.loc[:, ('Adj Close', ('GOOGL', 'IBM'))]).diff(return_period)
X1.columns = X1.columns.droplevel()

X2 = np.log(ccy_data).diff(return_period)

X3 = np.log(idx_data).diff(return_period)

X4 = pd.concat([np.log(stk_data.loc[:, ('Adj Close', 'MSFT')]).diff(i) for i in[return_peri

X4.columns = ['MSFT_DT', 'MSFT_3DT', 'MSFT_6DT', 'MSFT_12DT']

X = pd.concat([X1, X2, X3, X4], axis=1)

dataset = pd.concat([Y, X], axis=1).dropna().iloc[::return_period, :]

Y = dataset.loc[:, Y.name]
X = dataset.loc[:, X.columns]
```

In [20]:

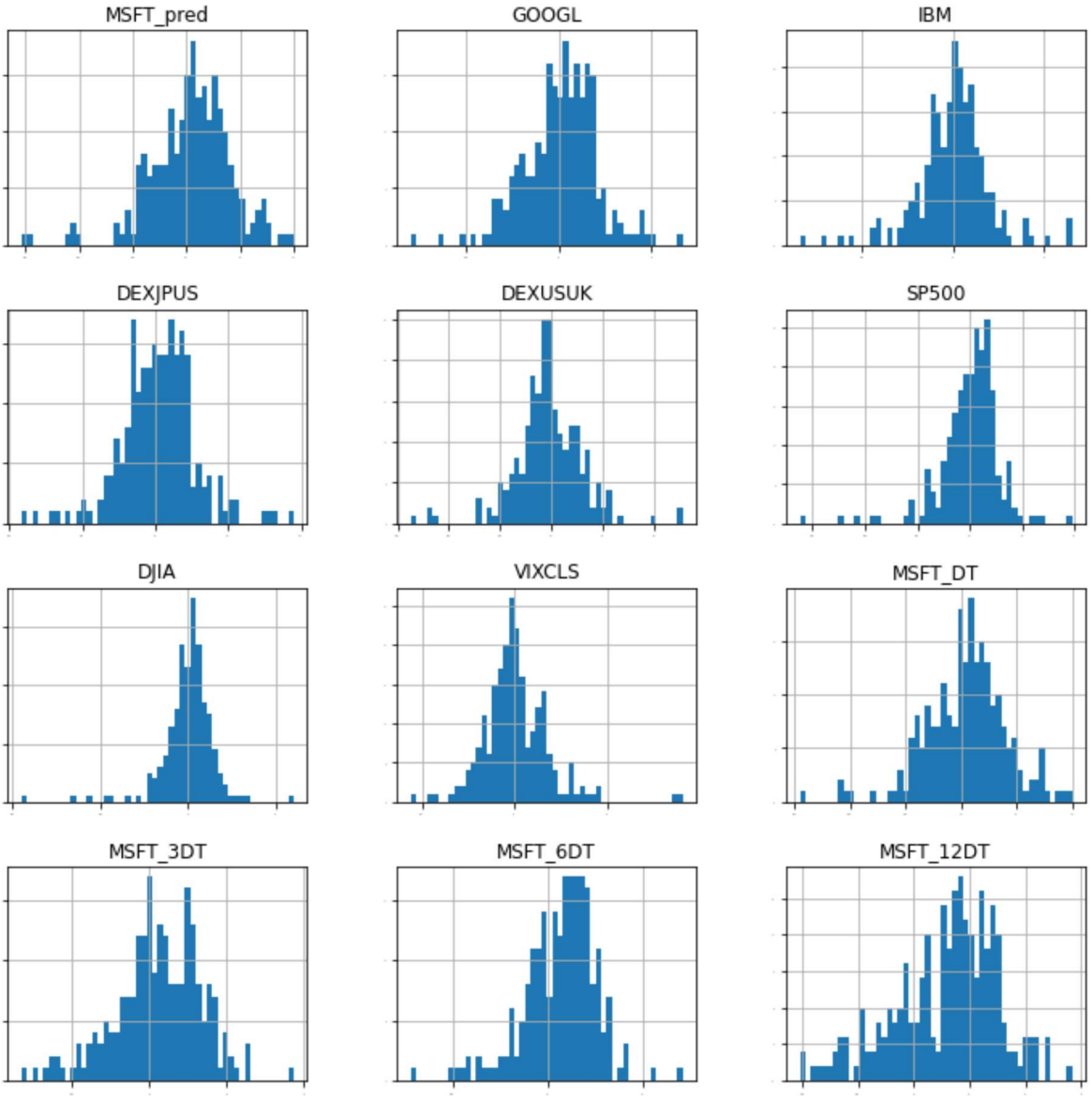
```
pd.set_option('precision', 3)
dataset.describe()
```

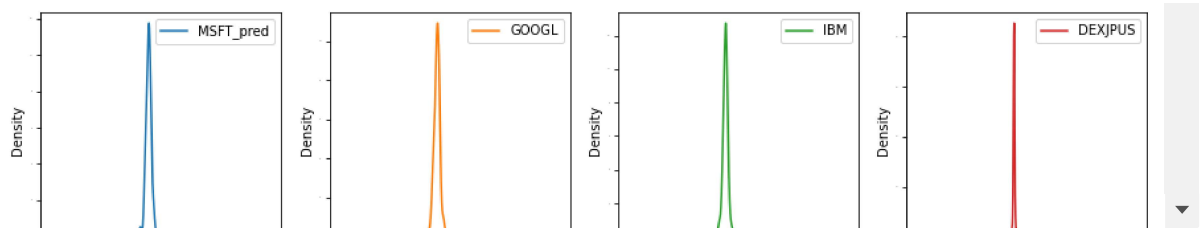
Out[20]:

	MSFT_pred	GOOGL	IBM	DEXJPUS	DEXUSUK	SP500	DJIA	VIXCLS	M
count	225.000	225.000	2.250e+02	2.250e+02	225.000	2.250e+02	2.250e+02	225.000	
mean	0.004	0.002	4.692e-05	8.817e-04	-0.001	5.134e-04	1.224e-04	0.008	
std	0.038	0.040	3.886e-02	1.025e-02	0.013	2.864e-02	2.950e-02	0.178	
min	-0.153	-0.159	-1.683e-01	-3.665e-02	-0.055	-1.623e-01	-1.900e-01	-0.559	
25%	-0.017	-0.022	-1.904e-02	-5.229e-03	-0.008	-1.072e-02	-1.235e-02	-0.086	
50%	0.007	0.006	2.792e-03	1.031e-03	-0.001	5.060e-03	3.284e-03	-0.007	
75%	0.027	0.028	1.967e-02	6.823e-03	0.006	1.639e-02	1.431e-02	0.095	
max	0.100	0.134	1.304e-01	3.800e-02	0.051	9.770e-02	1.208e-01	0.910	

In [21]:

```
dataset.hist(bins=50, sharex=False, sharey=False, xlabelsize=1, ylabelsize=1,figsize=(12,12
pyplot.show()

dataset.plot(kind='density', subplots=True, layout=(4,4), sharex=True,legend=True, fontsize
pyplot.show()
```



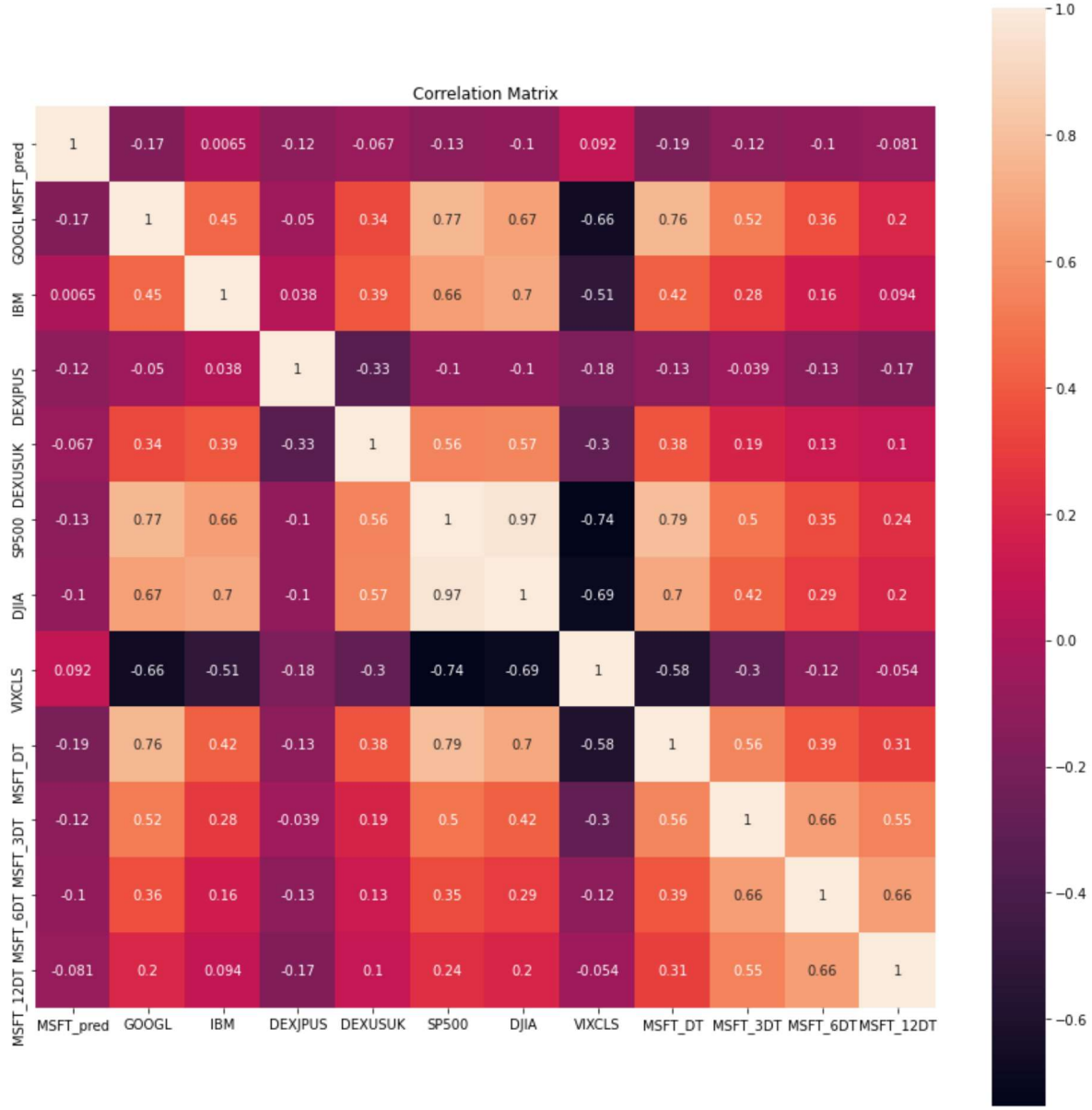


In [22]:

```
correlation = dataset.corr()
pyplot.figure(figsize=(15,15))
pyplot.title('Correlation Matrix')
sns.heatmap(correlation, vmax=1, square=True,annot=True)
```

Out[22]:

```
<AxesSubplot:title={'center':'Correlation Matrix'}>
```



In [ ]:

```
#From the correlation matrix, we can see some correlation of the predicted variable with th
# and 60-days returns of MSFT. Also, there is negative correlation of many asset returns ve
```

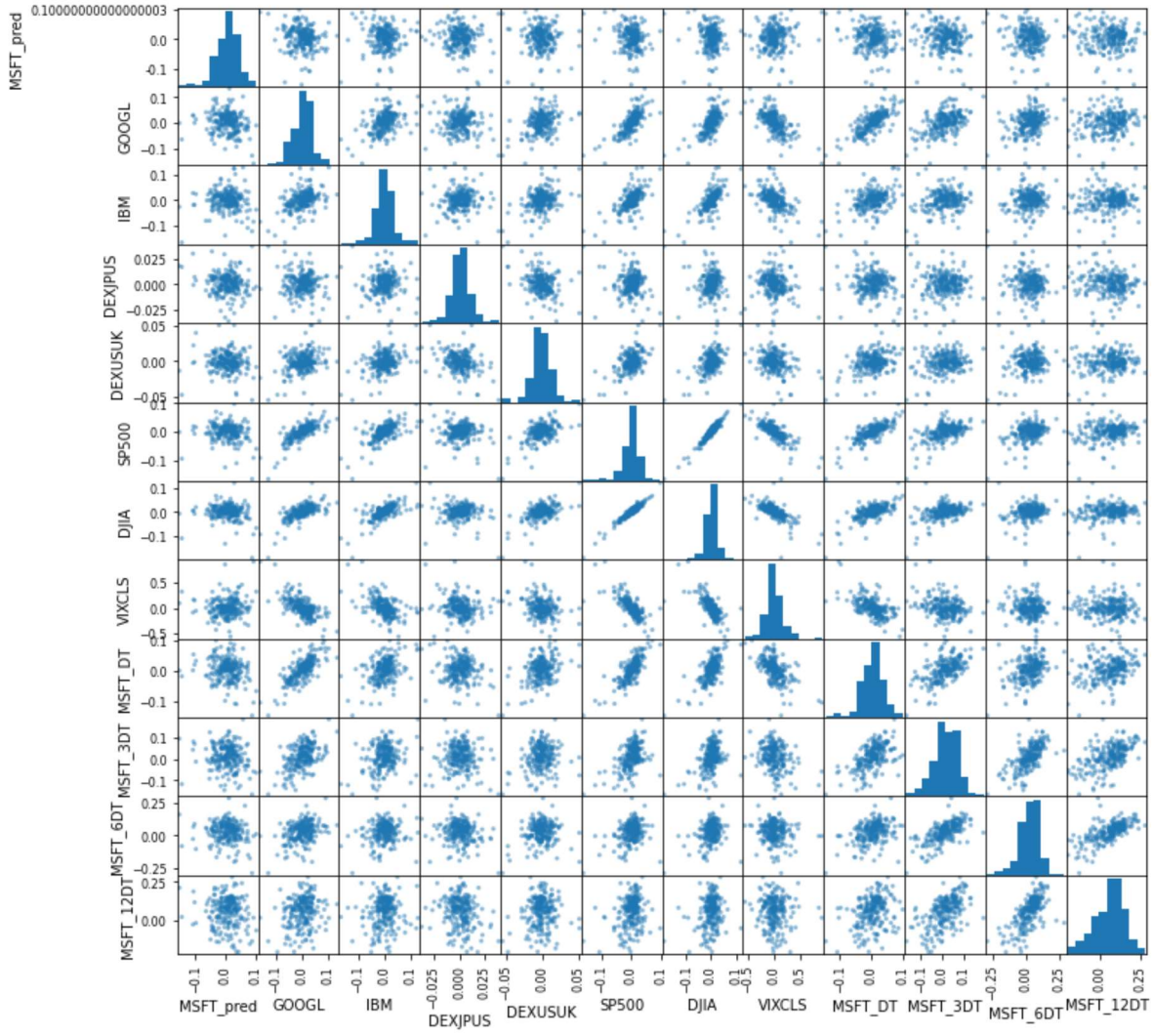
In [ ]:

```
# to visualize the relationship between all the variables in the regression, we will use th
```

In [23]:

```
pyplot.figure(figsize=(15,15))
scatter_matrix(dataset,figsize=(12,12))
pyplot.show()
```

```
<Figure size 1080x1080 with 0 Axes>
```

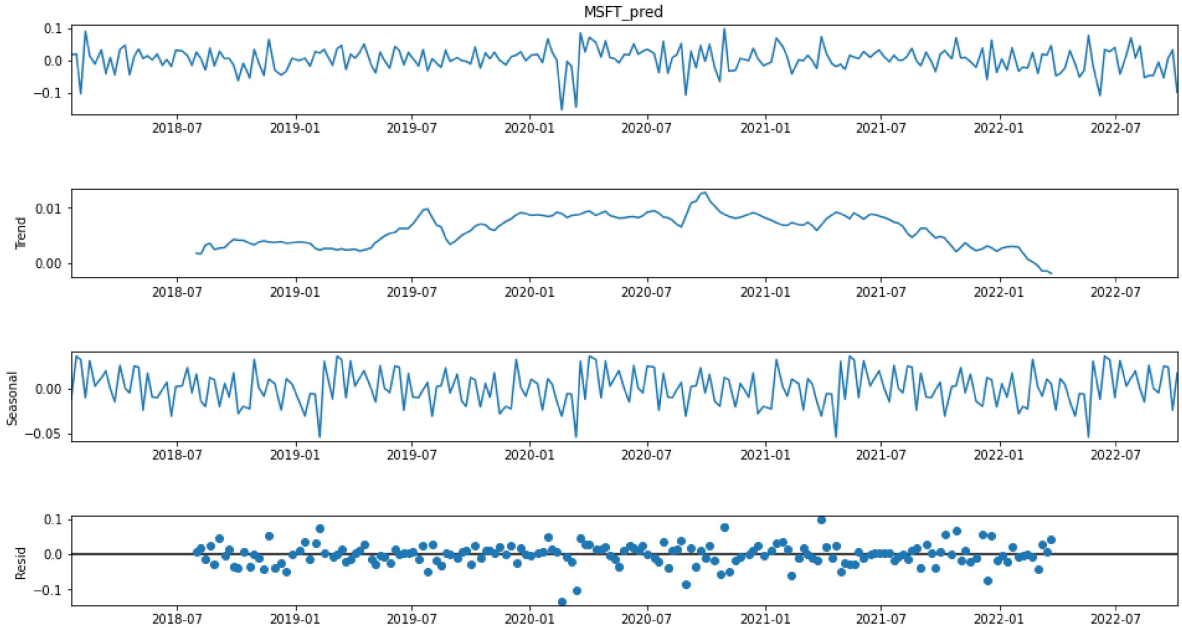


In [ ]:

```
# With a time series data, it is very important to look at the trend, seasonality and cycle
# using the seasonal decomposition of our time series.
```

In [26]:

```
res = sm.tsa.seasonal_decompose(Y, period=52)
fig = res.plot()
fig.set_figheight(8)
fig.set_figwidth(15)
pyplot.show()
```



In [ ]:

```
''' We can see that for MSFT there has been a general upward trend in the return series unt
July 2020. This may be due to the strong rise in MSFT over the last few years, which has re
in more positive than negative weekly return data points. Then there is a general downward
until January 2022. This new trend may be due to a decline in MSFT (which is probably relat
to internal or external events not yet known here).
The trend may show up in the constant/bias terms in our models. The residual (or white nois
term is relatively small over the entire time series.'''
```

In [ ]:

```
''' Data preprocessing is one of the key steps in building a robust model. The model in its
the analysis,but we might not get the optimum results if we fail to prep our data for the m
inferences.

Data preprocessing typically involves data processing, data cleaning, looking at feature im
feature reduction. The data we are using is relatively clean and doesn't require further pr

Feature reduction is also not required, as we have relatively small number of predictor var
use all of these for the prediction. However, to understand few of the important features f
we would use the SelectKBest function'''
```

In [31]:

```
bestfeatures = SelectKBest(k=6, score_func=f_regression)
fit = bestfeatures.fit(X,Y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)
#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Specs','Score'] #naming the dataframe columns
featureScores.nlargest(12,'Score').set_index('Specs') #print 12 best features
```

Out[31]:

Specs	Score
MSFT_DT	8.321
GOOGL	6.356
SP500	3.711
MSFT_3DT	3.464
DEXJPUS	3.063
DJIA	2.307
MSFT_6DT	2.288
VIXCLS	1.921
MSFT_12DT	1.483
DEXUSUK	1.010
IBM	0.010

In [ ]:

```
''' we would divide our data into training and test set. Unlike the random division of data
and test, for the time series data, we would select arbitrary split point in the ordered li
and create two new datasets'''
```

In [32]:

```
validation_size = 0.3
train_size = int(len(X) * (1-validation_size))
X_train, X_test = X[0:train_size], X[train_size:len(X)]
Y_train, Y_test = Y[0:train_size], Y[train_size:len(X)]
```

In [33]:

```
num_folds = 10
seed = 7
# scikit is moving away from mean_squared_error.
# In order to avoid confusion, and to allow comparison with other models, we invert the fin
scoring = 'neg_mean_squared_error'
```

In [76]:

```
models = []
models.append(('LR', LinearRegression()))
models.append(('LASSO', Lasso()))
models.append(('EN', ElasticNet()))
models.append(('KNN', KNeighborsRegressor()))
models.append(('SVR', SVR()))
models.append(('MLP', MLPRegressor())) # Neural network algorith
```

In [77]:

```
# Boosting methods
models.append(('ABR', AdaBoostRegressor()))
models.append(('GBR', GradientBoostingRegressor()))

# Bagging methods
models.append(('RFR', RandomForestRegressor()))
```

In [ ]:

```
''' We loop over our selected models and run the k-fold analysis.
Next, we run the model on the entire training and testing dataset.'''
```

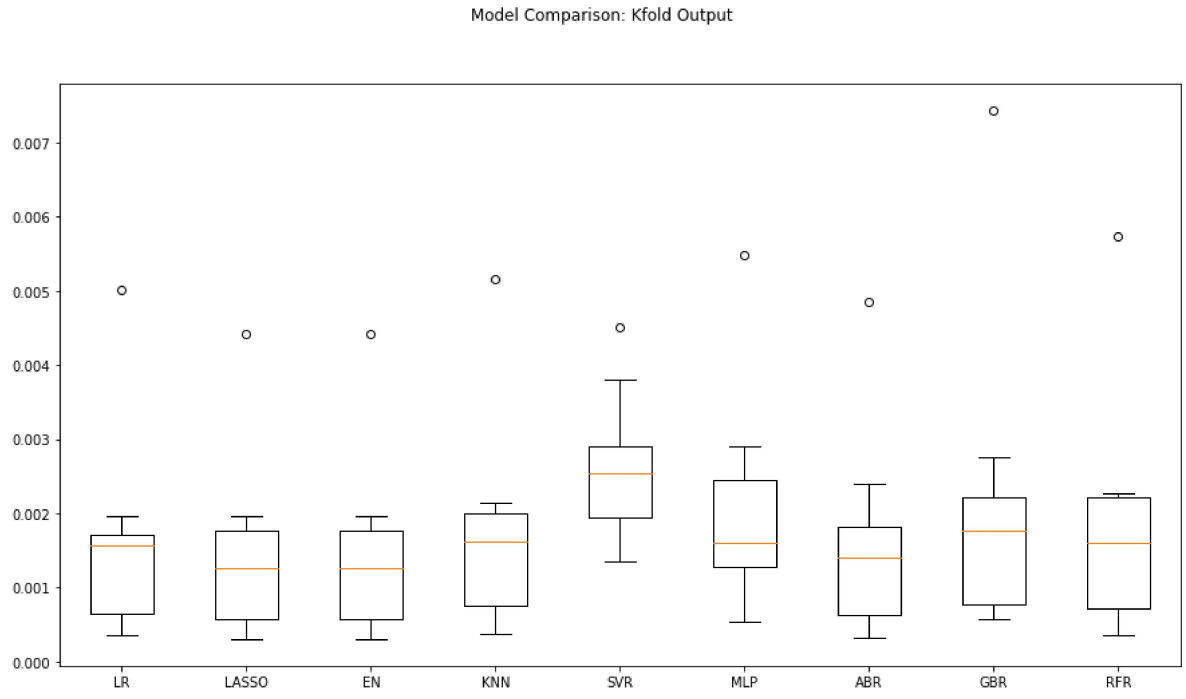
In [78]:

```
names = []
kfold_results = []
train_results = []
test_results = []

print('name\t\t', 'cv_results.mean\t', 'cv_results.std\t\t','train_result\t', 'test_result'

for name, model in models:
    names.append(name)
    kfold = KFold(n_splits=num_folds)

    # converting mean square error to positive. The lower the beter
    cv_results = -1* cross_val_score(model, X_train, Y_train, cv=kfold,scoring=scoring)
    kfold_results.append(cv_results)

    # Full Training period
    res = model.fit(X_train, Y_train)
    train_result = mean_squared_error(res.predict(X_train), Y_train)
    train_results.append(train_result)

    # Test results
    test_result = mean_squared_error(res.predict(X_test), Y_test)
    test_results.append(test_result)


    msg = "%s:\t\t %f\t\t    %f\t\t   %f\t  %f\t" % (name, cv_results.mean(), cv_results.st
    print(msg)
```

```
name            cv_results.mean         cv_results.std          train_resul
t       test_result
LR:             0.001594                  0.001270              0.001201
0.001761
LASSO:          0.001428                  0.001141              0.001418
0.001432
EN:             0.001428                  0.001141              0.001418
0.001432
KNN:            0.001697                  0.001316              0.001221
0.001715
SVR:            0.002600                  0.000925              0.002503
0.002227
MLP:            0.001990                  0.001371              0.002373
0.002192
ABR:            0.001584                  0.001270              0.000630
0.001650
GBR:            0.002089                  0.001923              0.000090
0.002041
RFR:            0.001782                  0.001499              0.000251
0.001667
```

In [79]:

```
fig = pyplot.figure()
fig.suptitle('Model Comparison: Kfold Output')
ax = fig.add_subplot(111)
pyplot.boxplot(kfold_results)
ax.set_xticklabels(names)
fig.set_size_inches(15,8)
pyplot.show()
```

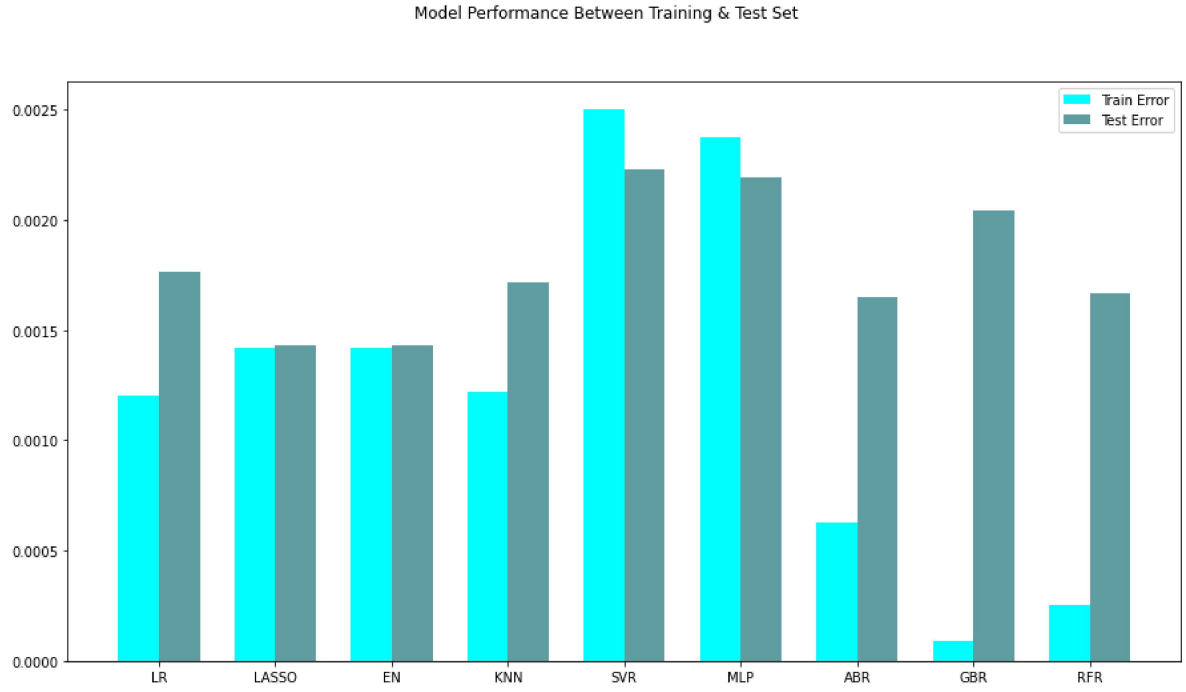


In [ ]:

```
''' We can clear;y see that the Lasso and the Elastic Net model perform best.
We can now evaluate the training and test errors to see which model fits best to the test d
This would help us evaluate overfitting by any of the models'''
```

In [84]:

```
fig = pyplot.figure()
ind = np.arange(len(names)) # the x locations for the groups
width = 0.35 # the width of the bars
fig.suptitle('Model Performance Between Training & Test Set')
ax = fig.add_subplot(111)
pyplot.bar(ind - width/2, train_results, width=width, label='Train Error', color='cyan')
pyplot.bar(ind + width/2, test_results, width=width, label='Test Error', color = 'cadetblue
fig.set_size_inches(15,8)
pyplot.legend()
ax.set_xticks(ind)
ax.set_xticklabels(names)
pyplot.show()
```



In [ ]:

```
''' We can see that the models like Linear regression, Lasso and Elastic Net are the best p
```

In [ ]:

```
''' I would now try and fit an LSTM model for the prediction and evaluate its performance w
```

In [90]:

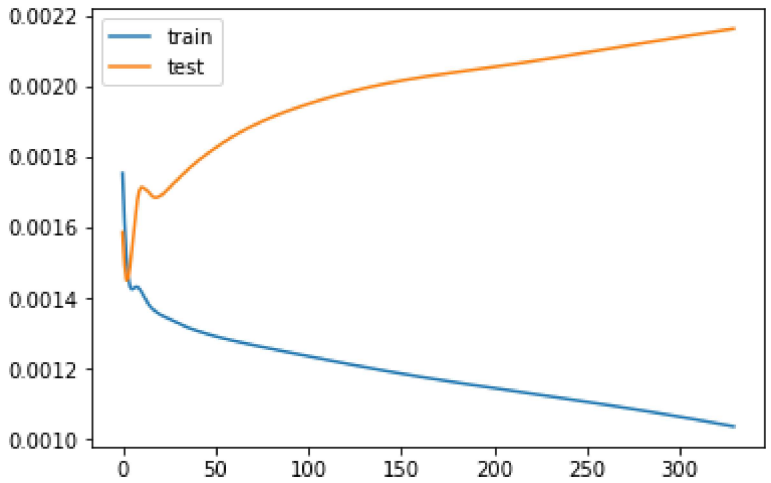
```
seq_len = 2 #Length of the seq for the LSTM
Y_train_LSTM, Y_test_LSTM = np.array(Y_train)[seq_len-1:], np.array(Y_test)
X_train_LSTM = np.zeros((X_train.shape[0]+1-seq_len, seq_len, X_train.shape[1]))
X_test_LSTM = np.zeros((X_test.shape[0], seq_len, X.shape[1]))

for i in range(seq_len):
    X_train_LSTM[:, i, :] = np.array(X_train)[i:X_train.shape[0]+i+1-seq_len, :]
    X_test_LSTM[:, i, :] = np.array(X)[X_train.shape[0]+i-1:X.shape[0]+i+1-seq_len, :]
```

In [91]:

```
def create_LSTMmodel(neurons=12, learn_rate = 0.01, momentum=0):
    model = Sequential()
    model.add(LSTM(50, input_shape=(X_train_LSTM.shape[1], X_train_LSTM.shape[2])))
    model.add(Dense(1))
    optimizer = SGD(lr=learn_rate, momentum=momentum)
    model.compile(loss='mse', optimizer='adam')
    return model

LSTMModel = create_LSTMmodel(12, learn_rate = 0.01, momentum=0)

LSTMModel_fit = LSTMModel.fit(X_train_LSTM, Y_train_LSTM,validation_data=(X_test_LSTM, Y_te
```

In [92]:

```
#Visual plot to check if the error is reducing
pyplot.plot(LSTMModel_fit.history['loss'], label='train')
pyplot.plot(LSTMModel_fit.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
```



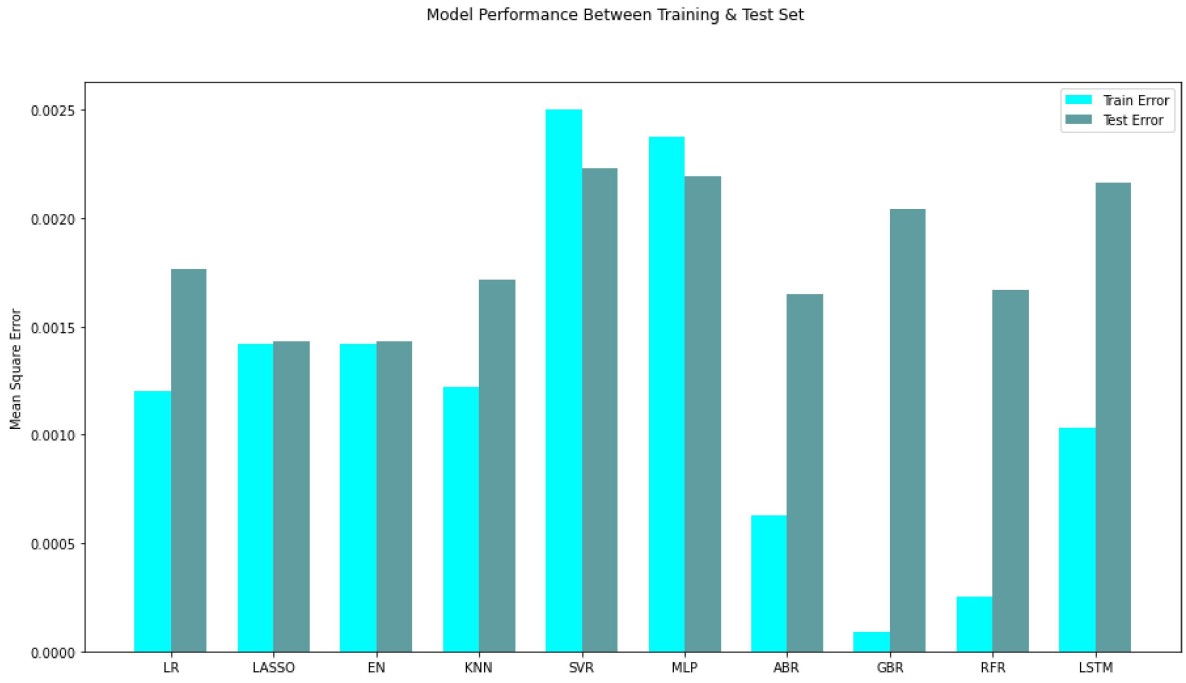
In [94]:

```
error_Training_LSTM = mean_squared_error(Y_train_LSTM, LSTMModel.predict(X_train_LSTM))
predicted = LSTMModel.predict(X_test_LSTM)
error_Test_LSTM = mean_squared_error(Y_test,predicted)

test_results.append(error_Test_LSTM)
train_results.append(error_Training_LSTM)
names.append("LSTM")
```

```
5/5 [==============================] - 0s 2ms/step
3/3 [==============================] - 0s 6ms/step
```

In [99]:

```
# compare algorithms
fig = pyplot.figure()
ind = np.arange(len(names)) # the x locations for the groups
width = 0.35 # the width of the bars
fig.suptitle('Model Performance Between Training & Test Set')
ax = fig.add_subplot(111)
pyplot.bar(ind - width/2, train_results, width=width, label='Train Error', color = 'cyan')
pyplot.bar(ind + width/2, test_results, width=width, label='Test Error', color = 'cadetblue
fig.set_size_inches(15,8)
pyplot.legend()
ax.set_xticks(ind)
ax.set_xticklabels(names)
pyplot.ylabel('Mean Square Error')
pyplot.show()
```



In [ ]:

```
''' We can see that the simple linear models - linear regression, regularized regression (i
net) and other models like LSTM. These models can enable financial practitioners to model t
flexible approach. It is essential that we also test our models by including other set of r
ratio, trading volume, technical indicators or news data, which might lead to better result
```