Stock Price Prediction

GUIDJIME ADINSI Ahouahounko Télesphore

August 27, 2022

Case Day: Stock Price Prediction

One of the biggest challenges in finance is predicting stock prices. However, with the onset of recent advancements in machine learning applications, the field has been evolving to utilize nondeterministic solutions that learn what is going on in order to make more accurate predictions. Machine learning techniques naturally lend themselves to stock price prediction based on historical data. Predictions can be made for a single time point ahead or for a set of future time points.

As a high-level overview, other than the historical price of the stock itself, the features that are generally useful for stock price prediction are as follows:

- **Correlated assets**: An organization depends on and interacts with many external factors, including its competitors, clients, the global economy, the geopolitical situation, fiscal and monetary policies, access to capital, and so on. Hence, its stock price may be correlated not only with the stock price of other companies but also with other assets such as commodities, FX, broad-based indices, or even fixed income securities;
- **Technical indicators**: A lot of investors follow technical indicators. Moving average, exponential moving average, and momentum are the most popular indicators;
- **Fundamental analysis**: Two primary data sources to glean features that can be used in fundamental analysis include:
  - Performance reports: Annual and quarterly reports of companies can be used to extract or determine key metrics, such as ROE (Return on Equity) and P/E (Price-to-Earnings);
  - News: News can indicate upcoming events that can potentially move the stock price in a certain direction.

In this case study, we will use various supervised learning–based models to predict the stock price of Microsoft using correlated assets and its own historical data. By the end of this case study, readers will be familiar with a general machine learning approach to stock prediction modeling, from gathering and cleaning data to building and tuning different models.

Furthermore, we will focus on:

- Looking at various machine learning and time series models, ranging in complexity, that can be used to predict stock returns.
- Visualization of the data using different kinds of charts (i.e., density, correlation, scatterplot, etc.).
- Using deep learning (LSTM) models for time series forecasting.

- Implementation of the grid search for time series models (i.e., ARIMA model).
- Interpretation of the results and examining potential overfitting and underfitting of the data across the models.

0.1 Blueprint for Using Supervised Learning Models to Predict a Stock Price

0.1.1 1. Problem definition:

In the supervised regression framework used for this case study, the weekly return of Microsoft stock is predicted variable. We need to undersand what affects Microsoft stock price incorporate as much information into the model. Out of correlated assets, technical indicators, and fundamental analysis. We will focus on correlated assets as features in this case study.

For this case study, other than the historical data of Microsoft, the independent variables used are the following potentially correlated assets: * *Stocks* : IBM (IBM) and Alphabet (GOOGL); * *Currency* : USD/JPY and GBP/USD; * *Indices* : S&P 500, Dow Jones, and VIX;

The dataset used for this case study is extracted from Yahoo Finance and the FRED website. In addition to predicting the stock price accurately, this case study will also demonstrate the infrastructure and framework for each step of time series and supervised regression–based modeling for stock price prediction. We will use the daily closing price of the last 10 years, from 2010 onward.

0.1.2 2. Getting started—loading the data and Python packages

2.1 Loading the Python packages

[38]:
```python
# Function and modules for data preparation and visualization
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

#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
```

```
#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

#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

#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')
```

[39]:
```
keras.__version__
```

[39]:
```
'2.2.4-tf'
```

2.2 Loading the data.

[11]:
```
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')
```

Next, we need a series to predict. We choose to predict using weekly returns. We approximate this by using 5 business day period returns.

[14]:
```
return_period = 5
```

We now define our Y series and our X series

Y: MSFT **Future** Returns

X:

```
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
```

We remove the MSFT past returns when we use the Time series models.

[15]:
```
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_period, return_period*3, return_period*6, return_period*12]], axis=1).
 ↪dropna()
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]
```

0.1.3 3. Exploratory Data Analysis

3.1 Descriptive Statistics

[16]:
```
pd.set_option('precision', 3)
dataset.describe()
```

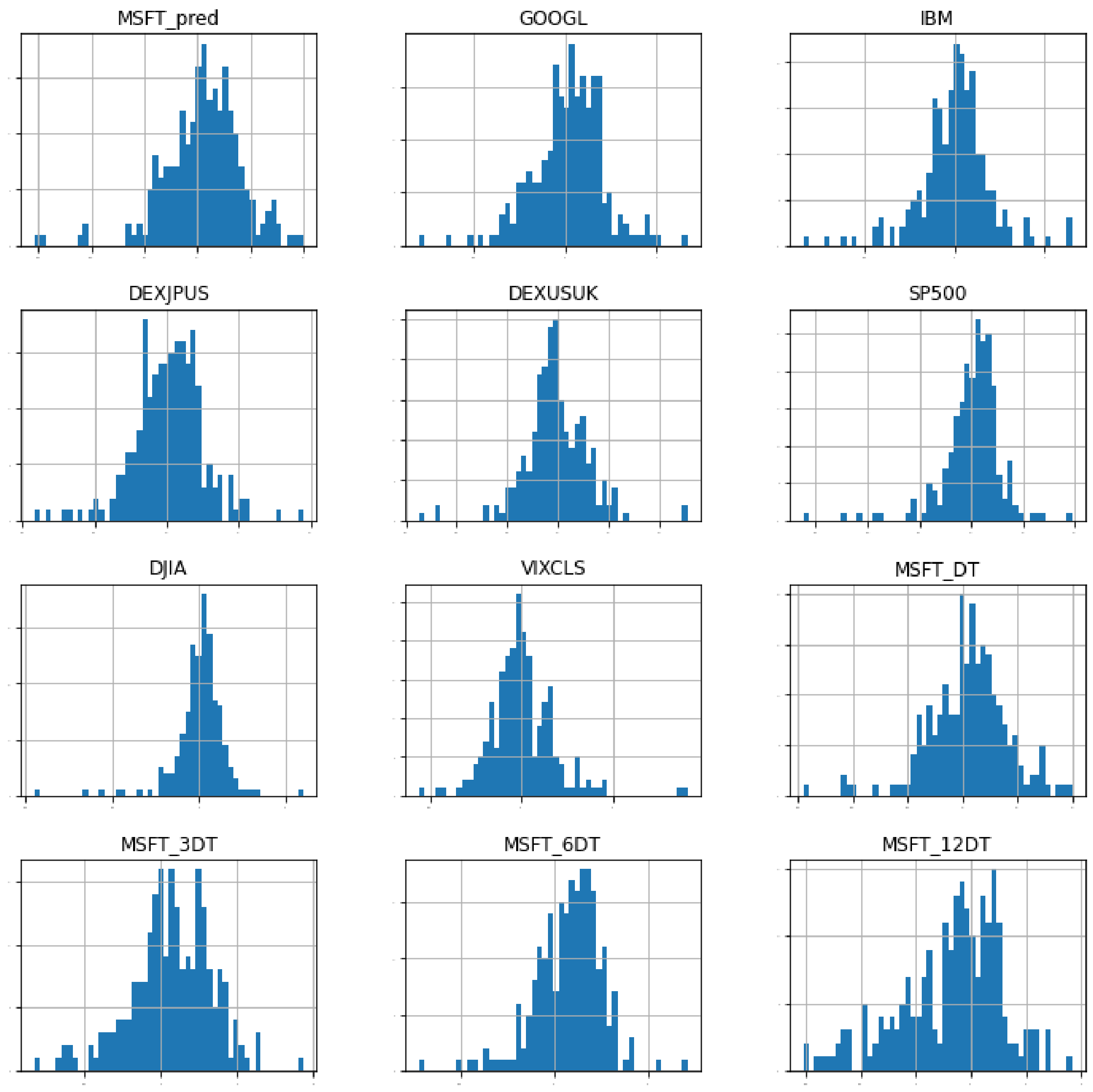
```
[16]:        MSFT_pred     GOOGL        IBM     DEXJPUS     DEXUSUK     SP500  \
count      225.000   225.000  2.250e+02  2.250e+02  2.250e+02  225.000
mean         0.005     0.003  7.839e-04  5.544e-04 -8.392e-04    0.001
std          0.037     0.039  3.891e-02  9.817e-03  1.263e-02    0.028
min         -0.153    -0.159 -1.683e-01 -3.665e-02 -5.492e-02   -0.162
25%         -0.015    -0.016 -1.877e-02 -5.330e-03 -6.905e-03   -0.009
50%          0.007     0.007  3.416e-03  1.013e-03 -1.435e-03    0.006
75%          0.027     0.028  1.994e-02  6.492e-03  6.701e-03    0.016
max          0.100     0.134  1.304e-01  3.800e-02  5.121e-02    0.098

             DJIA    VIXCLS   MSFT_DT   MSFT_3DT   MSFT_6DT   MSFT_12DT
count   2.250e+02   225.000   225.000    225.000    225.000     225.000
mean    9.433e-04     0.005     0.005      0.014      0.031       0.064
std     2.928e-02     0.178     0.035      0.053      0.071       0.092
min    -1.900e-01    -0.559    -0.145     -0.165     -0.288      -0.204
25%    -9.105e-03    -0.093    -0.015     -0.014     -0.006       0.011
50%     4.748e-03    -0.009     0.008      0.016      0.040       0.082
75%     1.444e-02     0.093     0.026      0.051      0.076       0.128
max     1.208e-01     0.910     0.100      0.186      0.285       0.283
```

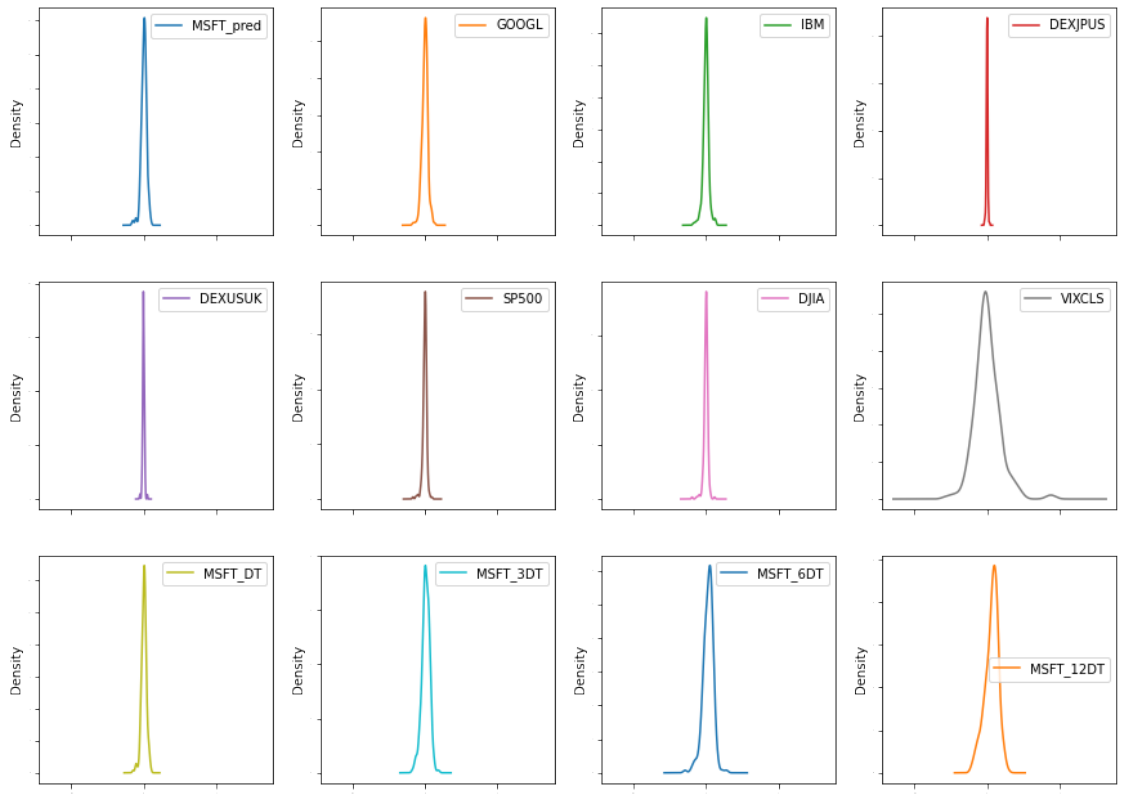
```
[6]: dataset.tail()
```

```
[6]:            MSFT_pred  GOOGL    IBM  DEXJPUS    DEXUSUK  SP500   DJIA  VIXCLS  \
2022-07-18      0.018 -0.059 -0.021    0.007  6.863e-03 -0.006 -0.003  -0.034
2022-07-25      0.071 -0.014 -0.072   -0.011  4.411e-03  0.035  0.029  -0.080
2022-08-01      0.008  0.066  0.027   -0.037  1.941e-02  0.038  0.025  -0.023
2022-08-08      0.046  0.021  0.004    0.022 -1.477e-02  0.005  0.001  -0.070
2022-08-15     -0.053  0.040  0.030   -0.012 -6.615e-04  0.037  0.032  -0.065

            MSFT_DT  MSFT_3DT  MSFT_6DT  MSFT_12DT
2022-07-18   -0.040    -0.052    -0.077     -0.117
2022-07-25    0.018    -0.003    -0.023     -0.088
2022-08-01    0.071     0.050     0.127     -0.040
2022-08-08    0.008     0.098     0.046      0.075
2022-08-15    0.046     0.126     0.123      0.144
```

3.2 Data visualization The distribution of the data over the entire period

```
[17]: 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=1, figsize=(15,15))
      pyplot.show()
```

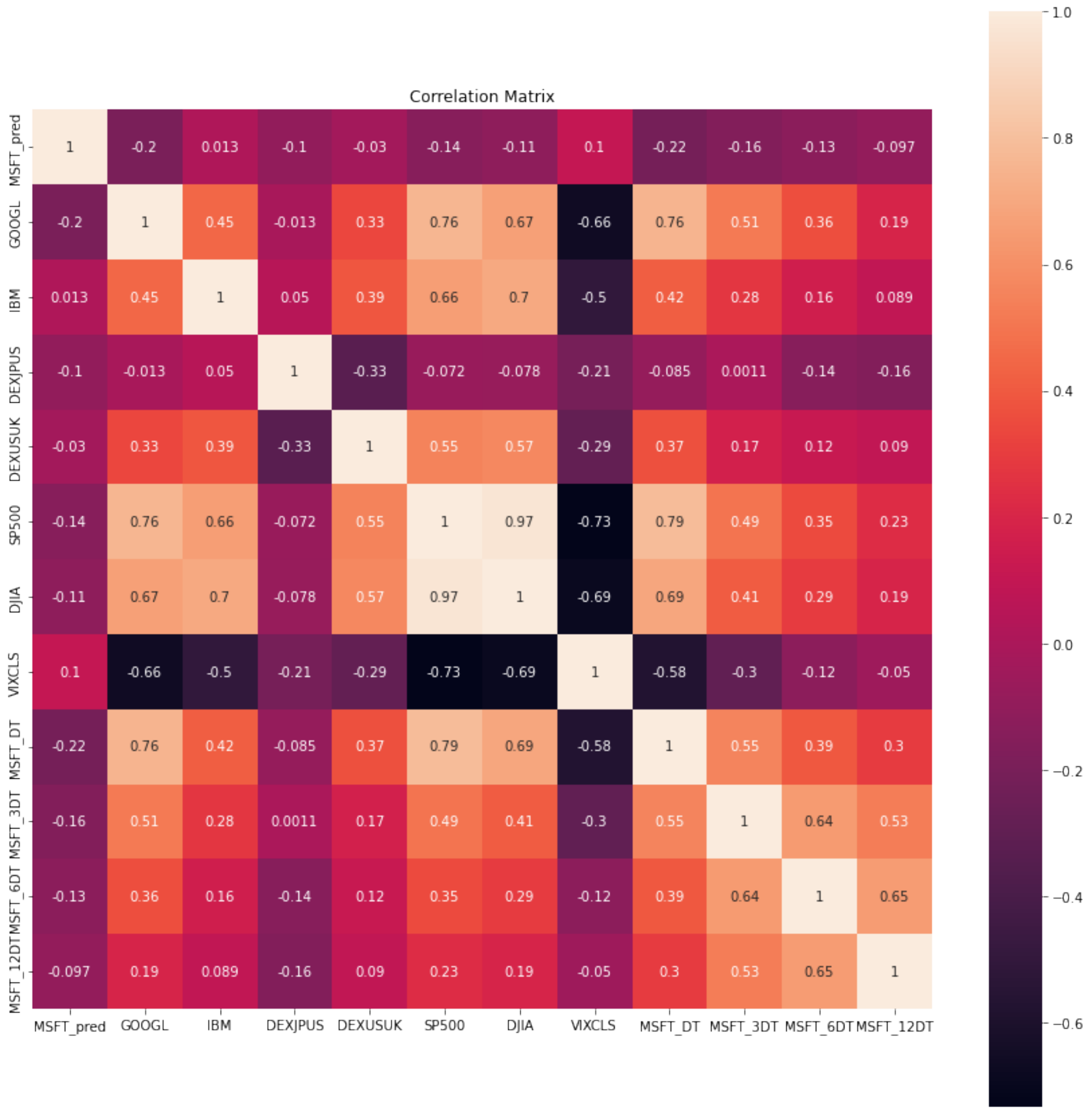

MSFT_pred
GOOGL
IBM
DEXJPUS
DEXUSUK
SP500
DJIA
VIXCLS
MSFT_DT
MSFT_3DT
MSFT_6DT
MSFT_12DT



We can see that the vix has a much larger variance compared to the other distributions.

[19]:
```
correlation = dataset.corr()
pyplot.figure(figsize=(15,15))
pyplot.title('Correlation Matrix')
sns.heatmap(correlation, vmax=1, square=True,annot=True)
```

[19]:
```
<AxesSubplot:title={'center':'Correlation Matrix'}>
```

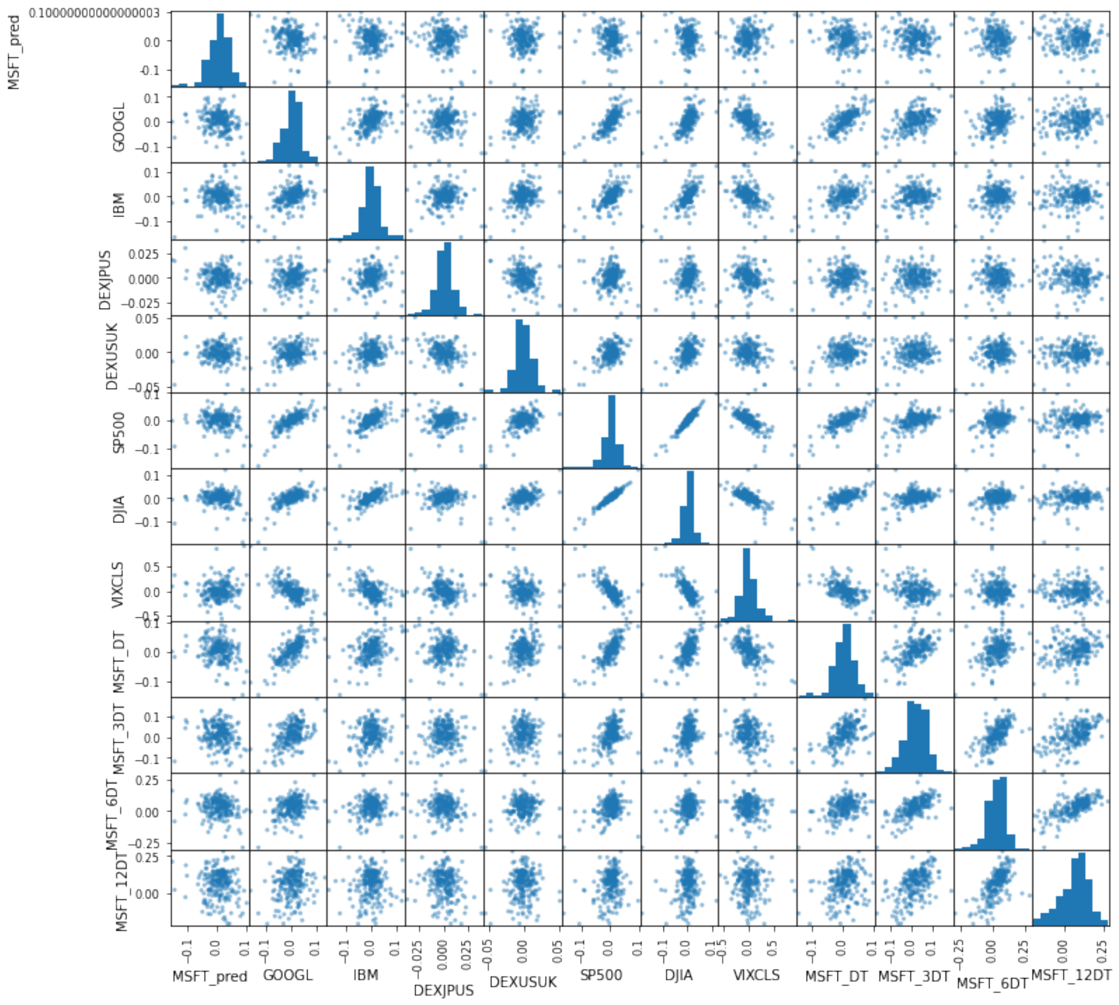


Looking at the correlation plot, we see some correlation of teh predicted variable with the lagged 5-days, 15-days,30-days, and 60-days returns of MSFT. Also, we see a higher negative correlation of many asset returns versus VIX, which is intuitive.

Next, we can visualize the relationship between all the variables in the regression using the scatterplot matrix shown below:

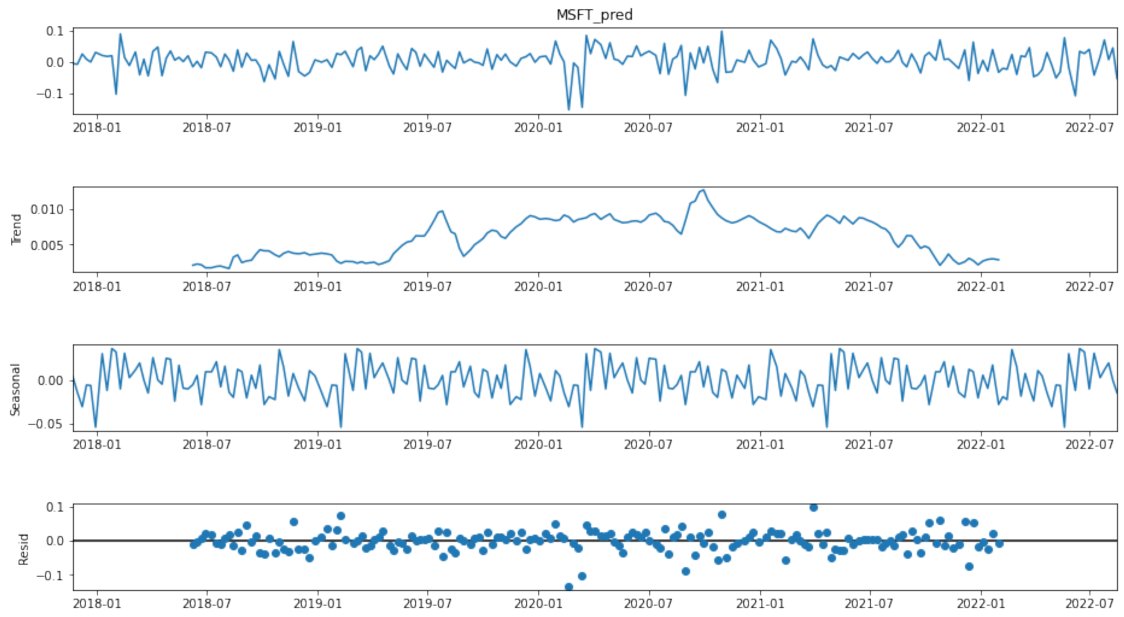
[20]:
```
pyplot.figure(figsize=(15,15))
scatter_matrix(dataset,figsize=(12,12))
pyplot.show()
```

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



3.3 Time Series Analysis Let now look at the seasonal decomposition of our time series.

[21]:
```
res = sm.tsa.seasonal_decompose(Y, period=52)
fig = res.plot()
fig.set_figheight(8)
fig.set_figwidth(15)
pyplot.show()
```



We can see that for MSFT there has been a general upward trend in the return series until just after July 2020. This may be due to the strong rise in MSFT over the last few years, which has resulted in more positive than negative weekly return data points. Then there is a general downward trend until January 2022. This new trend may be due to a decline in MSFT (which is probably related 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 noise) term is relatively small over the entire time series.

0.1.4 4. Data Preparation

This step typically involves data processing, data cleaning, looking at feature importance, an performing feature reduction. The data obtained for this case study is relatively clean and doesn't require further processing. Feature reduction might be useful here, but given the relatively small number of variables considered, we will keep all of them as is.

4.1 Feature Selection We use sklearn's SelectKBest function to get a sense of feature importance.

[22]:
```
bestfeatures = SelectKBest(k=5, 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(10,'Score').set_index('Specs')  #print 10 best features
```

[22]:
```
            Score
Specs
MSFT_DT    10.870
GOOGL       8.976
MSFT_3DT    5.752
SP500       4.544
MSFT_6DT    3.822
DJIA        2.763
VIXCLS      2.346
DEXJPUS     2.270
MSFT_12DT   2.098
DEXUSUK     0.202
```

We see that IBM seems to be the most important feature and vix being the least important.

0.1.5 5. Evaluate Algorithms and Models

5.1. Train Test Split and Evaluation Metrics With time series data, because the sequence of values is important, we do not distribute the dataset into training ant test sets in random fashion. But here, we select arbitrary split point in the ordered list od observations and create two new datasets:

[23]:
```
validation_size = 0.2

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)]
```

5.2. Test options and evaluation metrics.

[29]:
```
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 final scores
scoring = 'neg_mean_squared_error'
```

5.3 Compare Models and Algorithms

5.3.1 Machine Learning models-from scikit-learn Regression and Tree Regression algorithms

[25]:
```
models = []
models.append(('LR', LinearRegression()))
models.append(('LASSO', Lasso()))
models.append(('EN', ElasticNet()))
models.append(('KNN', KNeighborsRegressor()))
models.append(('CART', DecisionTreeRegressor()))
models.append(('SVR', SVR()))
```

Neural Network Algorithms

[33]:
```
models.append(('MLP', MLPRegressor()))
```

Ensemble models

[26]:
```
# Boosting methods
models.append(('ABR', AdaBoostRegressor()))
models.append(('GBR', GradientBoostingRegressor()))
# Bagging methods
models.append(('RFR', RandomForestRegressor()))
models.append(('ETR', ExtraTreesRegressor()))
```

Once we have selected all the models, we loop over each of them. First, we run the k-fold analysis. Next, we run the model on the entire training and testing dataset.

[31]:
```
names = []
kfold_results = []
test_results = []
train_results = []
for name, model in models:
    names.append(name)

    ## K Fold analysis:

    kfold = KFold(n_splits=num_folds)
    #converted 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: %f (%f) %f %f" % (name, cv_results.mean(), cv_results.std(),␣
 ↪train_result, test_result)
    print(msg)
```

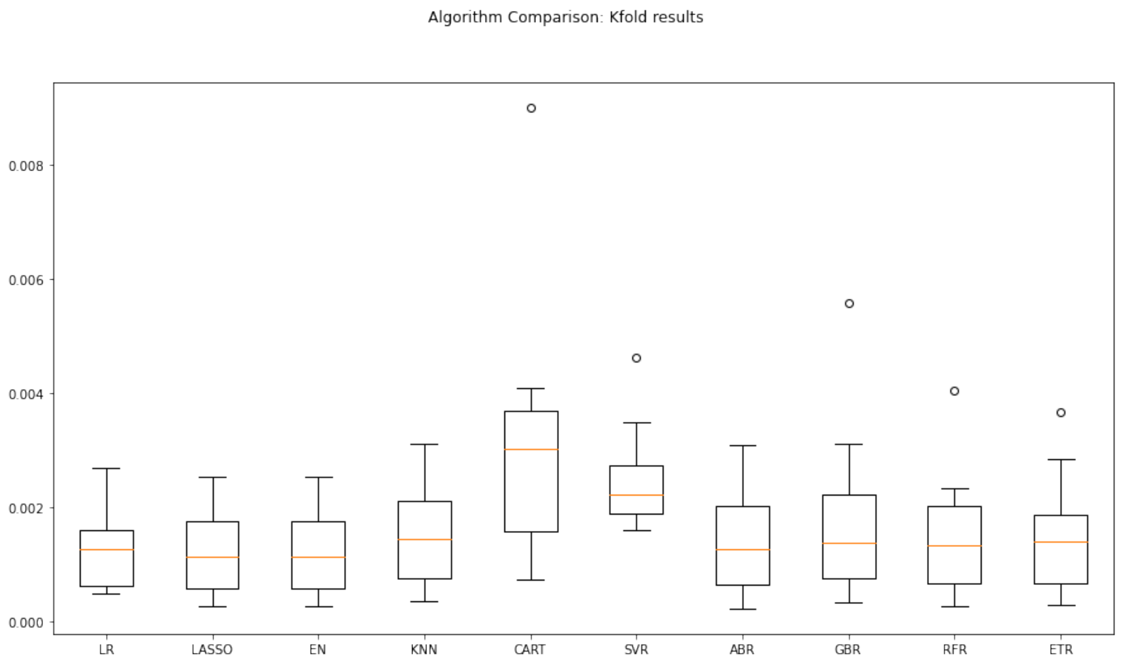
```
LR: 0.001307 (0.000691) 0.001088 0.001985
LASSO: 0.001275 (0.000778) 0.001265 0.001630
EN: 0.001275 (0.000778) 0.001265 0.001630
KNN: 0.001523 (0.000883) 0.001034 0.001832
```

```
CART: 0.003198 (0.002243) 0.000000 0.002812
SVR: 0.002501 (0.000894) 0.002449 0.002335
ABR: 0.001372 (0.000863) 0.000550 0.001926
GBR: 0.001838 (0.001483) 0.000093 0.002494
RFR: 0.001526 (0.001066) 0.000205 0.001978
ETR: 0.001530 (0.001021) 0.000000 0.001929
```

K Fold results

We being by looking at the K Fold results

[32]:
```python
fig = pyplot.figure()
fig.suptitle('Algorithm Comparison: Kfold results')
ax = fig.add_subplot(111)
pyplot.boxplot(kfold_results)
ax.set_xticklabels(names)
fig.set_size_inches(15,8)
pyplot.show()
```

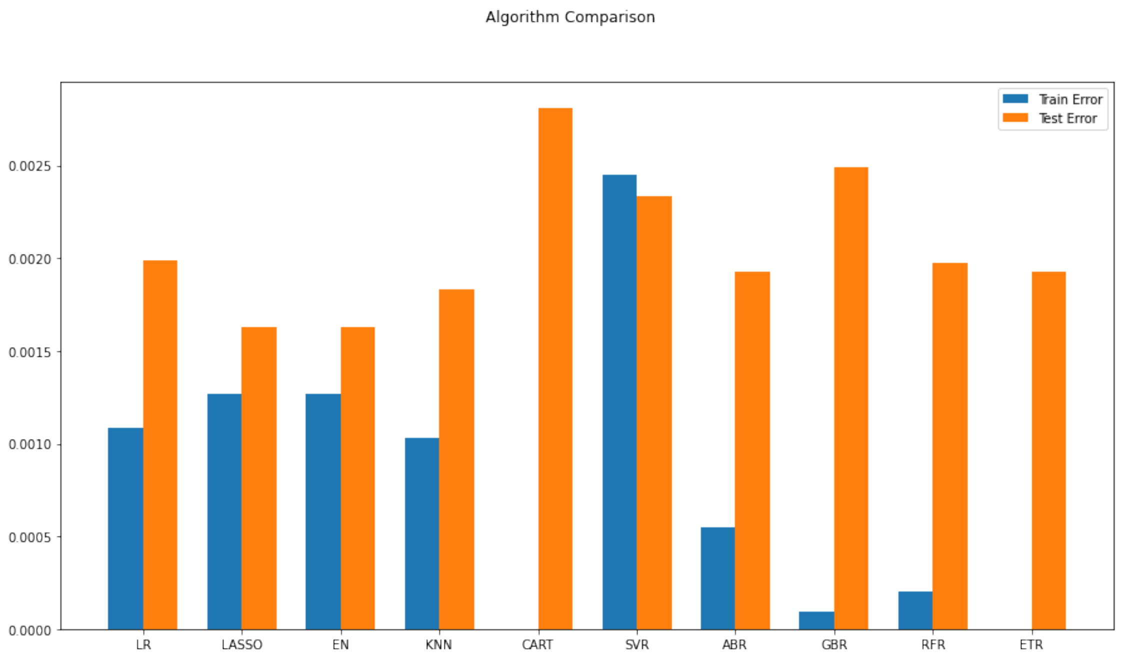


Although the results of a couple of the models look good, we see that the **linear regression** and the regularized regression including the **lasso regression (LASSO)** and **elastic net (EN)** seem to perform best. This indicates a strong linear relationship between the dependent and independent variables. Going back to the exploratory analysis, we saw a good correlation and linear relationship of the target variables with the different lagged MSFT variables.

Let us look at the errors of the test set as well:

Training and test error

[33]:
```
# 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('Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.bar(ind - width/2, train_results,  width=width, label='Train Error')
pyplot.bar(ind + width/2, test_results, width=width, label='Test Error')
fig.set_size_inches(15,8)
pyplot.legend()
ax.set_xticks(ind)
ax.set_xticklabels(names)
pyplot.show()
```



Looking at the training and test error, we still see a better performance of the linear models. Some of the algorithms, such as the decision tree regressor (CART) overfit on the training data and produced very high error on the test set and these models should be avoided. Ensemble models, such as gradient boosting regression (GBR) and random forest regression (RFR) have low bias but high variance. We also see that the artificial neural network (shown as MLP is the chart) algorithm shows higher errors both in training set and test set, which is perhaps due to the linear relationship of the variables not captured accurately by ANN or improper hyperparameters or insufficient training of the model.

We now look at some of the time series and deep learning models that can be used. Once we are done creating these, we will compare their performance against that of the supervised regression–

based models. Due to the nature of time series models, we are not able to run a k-fold analysis. We can still compare our results to the other models based on the full training and testing results.

5.3.2 Time Series based models-ARIMA and LSTM The models used so far already embed the time series component by using a time-delay approach, where the lagged variable is included as one of the independent variables. However, for the time series–based models we do not need the lagged variables of MSFT as the independent variables. Hence, as a first step we remove MSFT's previous returns for these models. We use all other variables as the exogenous variables in these models.

Time Series Model - ARIMA Model Let us first prepare the dataset for ARIMA models by having only the correlated variables as exogenous variables:

[35]:
```
X_train_ARIMA=X_train.loc[:, ['GOOGL', 'IBM', 'DEXJPUS', 'SP500', 'DJIA',␣
 ↪'VIXCLS']]
X_test_ARIMA=X_test.loc[:, ['GOOGL', 'IBM', 'DEXJPUS', 'SP500', 'DJIA',␣
 ↪'VIXCLS']]
tr_len = len(X_train_ARIMA)
te_len = len(X_test_ARIMA)
to_len = len (X)
```

We now configure the ARIMA model with the order (1,0,0) and use the independent variables as the exogenous variables in the model. The version of the ARIMA model where the exogenous variables are also used is known as the ARIMAX model, where "X" represents exogenous variables:

[40]:
```
modelARIMA=ARIMA(endog=Y_train,exog=X_train_ARIMA,order=[1,0,0])
model_fit = modelARIMA.fit()
```

[41]:
```
error_Training_ARIMA = mean_squared_error(Y_train, model_fit.fittedvalues)
predicted = model_fit.predict(start = tr_len -1 ,end = to_len -1, exog =␣
 ↪X_test_ARIMA)[1:]
error_Test_ARIMA = mean_squared_error(Y_test,predicted)
error_Test_ARIMA
```

[41]:
```
0.0018741860656642385
```

Error of this ARIMA model is reasonable.

Now let's prepare the dataset for the LSTM model. We need the data in the form of arrays of all the input variables and the output variables.

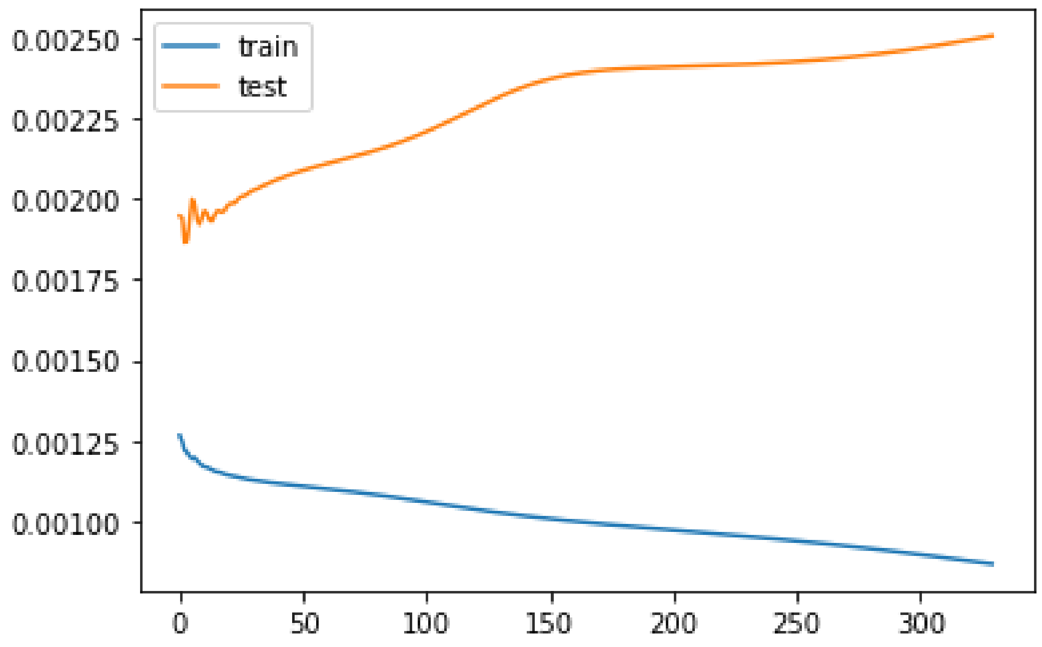
LSTM Model The logic behind the LSTM is that data is taken from the previous day (the data of all the other features for that day—correlated assets and the lagged variables of MSFT) and we try to predict the next day. Then we move the one-day window with one day and again predict the next day. We iterate like this over the whole dataset (of course in batches). The code below will create a dataset in which X is the set of independent variables at a given time (t) and Y is the target variable at the next time (t + 1):

```
[43]: 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, :]
```

```
[44]: # Lstm Network
      def create_LSTMmodel(neurons=12, learn_rate = 0.01, momentum=0):
              # create model
          model = Sequential()
          model.add(LSTM(50, input_shape=(X_train_LSTM.shape[1], X_train_LSTM.
       ↪shape[2])))
          #More number of cells can be added if needed
          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_test_LSTM),epochs=330, batch_size=72,␣
       ↪verbose=0, shuffle=False)
```

```
WARNING:tensorflow:From C:\Users\a\AppData\Local\R-MINI~1\envs\DL\lib\site-
packages\tensorflow\python\ops\init_ops.py:1251: calling
VarianceScaling.__init__ (from tensorflow.python.ops.init_ops) with dtype is
deprecated and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the
constructor
WARNING:tensorflow:From C:\Users\a\AppData\Local\R-MINI~1\envs\DL\lib\site-
packages\tensorflow\python\ops\math_grad.py:1250:
add_dispatch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) is
deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
```

```
[45]: #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()
```



[47]:
```
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)
```

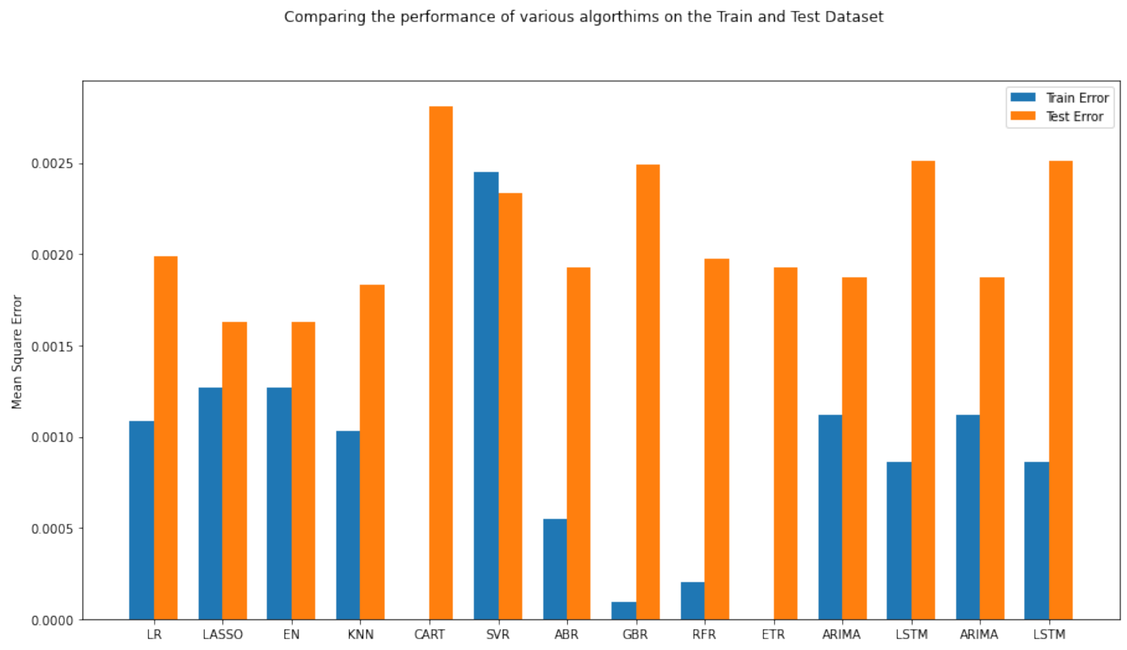
Append to previous results

[49]:
```
test_results.append(error_Test_ARIMA)
test_results.append(error_Test_LSTM)

train_results.append(error_Training_ARIMA)
train_results.append(error_Training_LSTM)

names.append("ARIMA")
names.append("LSTM")
```

0.1.6 Overall Comparison of all the algorithms (including Time Series Algorithms)

[50]:
```
# 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('Comparing the performance of various algorthims on the Train and␣
 ↪Test Dataset')
```

```
ax = fig.add_subplot(111)
pyplot.bar(ind - width/2, train_results,  width=width, label='Train Error')
pyplot.bar(ind + width/2, test_results, width=width, label='Test Error')
fig.set_size_inches(15,8)
pyplot.legend()
ax.set_xticks(ind)
ax.set_xticklabels(names)
pyplot.ylabel('Mean Square Error')
pyplot.show()
```



Looking at the chart above, we find time series based ARIMA model comparable to the linear supervised-regression models such as Linear Regression (LR), Lasso Regression (LASSO) and Elastic Net (EN). This can primarily be due to the strong linear relationship as discussed before. The LSTM model performs decently, however, ARIMA model outperforms the LSTM model in the test set. Hence, we select the ARIMA model for the model tuning.

0.1.7 6. Model Tuning and Grid Search

As shown in the chart above the ARIMA model is one of the best mode, so we perform the model tuning of the ARIMA model. The default order of ARIMA model is [1,0,0]. We perform a grid search with different combination p,d and q in the ARIMA model's order.

[51]:
```
#Grid Search for ARIMA Model
#Change p,d and q and check for the best result

# evaluate an ARIMA model for a given order (p,d,q)
#Assuming that the train and Test Data is already defined before
```

```python
def evaluate_arima_model(arima_order):
    #predicted = list()
    modelARIMA=ARIMA(endog=Y_train,exog=X_train_ARIMA,order=arima_order)
    model_fit = modelARIMA.fit()
    error = mean_squared_error(Y_train, model_fit.fittedvalues)
    return error

# evaluate combinations of p, d and q values for an ARIMA model
def evaluate_models(p_values, d_values, q_values):
    best_score, best_cfg = float("inf"), None
    for p in p_values:
        for d in d_values:
            for q in q_values:
                order = (p,d,q)
                try:
                    mse = evaluate_arima_model(order)
                    if mse < best_score:
                        best_score, best_cfg = mse, order
                    print('ARIMA%s MSE=%.7f' % (order,mse))
                except:
                    continue
    print('Best ARIMA%s MSE=%.7f' % (best_cfg, best_score))

# evaluate parameters
p_values = [0, 1, 2]
d_values = range(0, 2)
q_values = range(0, 2)
warnings.filterwarnings("ignore")
evaluate_models(p_values, d_values, q_values)
```

```
ARIMA(0, 0, 0) MSE=0.0011289
ARIMA(0, 0, 1) MSE=0.0011193
ARIMA(0, 1, 0) MSE=0.0021401
ARIMA(0, 1, 1) MSE=0.0011721
ARIMA(1, 0, 0) MSE=0.0011182
ARIMA(1, 0, 1) MSE=0.0011185
ARIMA(1, 1, 0) MSE=0.0016681
ARIMA(1, 1, 1) MSE=0.0011839
ARIMA(2, 0, 0) MSE=0.0011179
ARIMA(2, 0, 1) MSE=0.0011143
ARIMA(2, 1, 0) MSE=0.0014818
ARIMA(2, 1, 1) MSE=0.0011920
Best ARIMA(2, 0, 1) MSE=0.0011143
```

We see that the ARIMA model with the order (2,0,1) is the best performer out of all the combinations tested in the grid search, although there isn't a significant difference in the mean squared error (MSE) with other combinations. This means that the model with the autoregressive lag of two and moving average of one yields the best result. We should not forget the fact that there are other exogenous variables in the model that influence the order of the best ARIMA model as well.

0.1.8 7. Finalize the model

In the last step we will check the finalized model on the test set.

7.1. Results on the Test Dataset

```
[53]: # prepare model
      modelARIMA_tuned=ARIMA(endog=Y_train,exog=X_train_ARIMA,order=[2,0,1])
      model_fit_tuned = modelARIMA_tuned.fit()
```

```
[54]: # estimate accuracy on validation set
      predicted_tuned = model_fit.predict(start = tr_len -1 ,end = to_len -1, exog =␣
       ↪X_test_ARIMA)[1:]
      print(mean_squared_error(Y_test,predicted_tuned))
```

```
0.0018741860656642385
```

```
[ ]: After tuning the model and picking the best ARIMA model or the order 2,0 and 1
     we select this model and can it can be used for the modeling purpose.

     #### 7.2. Save Model
```

```
[55]: # Save Model Using Pickle
      from pickle import dump
      from pickle import load

      # save the model to disk
      filename = 'finalized_model.sav'
      dump(model_fit_tuned, open(filename, 'wb'))
```

1 Conclusion

We can conclude that simple models - linear regression, regularized regression (i.e. Lasso and elastic net) along with the time series model such as ARIMA are promising modelling approaches for asset price prediction problem. These models can enable financial practitioners to model time dependencies with a very flexible approach. The overall approach presented in this case study may help us encounter overfitting and underfitting which are some of the key challenges in the prediction problem in finance. We should also note that we can use better set of indicators, such as P/E ratio, trading volume, technical indicators or news data, which might lead to better results. We will demonstrate this in some of the case studies in the book. Overall, we created a supervised-regression and time series modelling framework which allows us to perform asset class prediction using historical data to generate results and analyze risk and profitability before risking any actual capital.