In [ ]:

```
# ETF - Data analysis
```

In [524]:

```
# 1 - Data Cleaning
```

In [523]:

```
# Import libraries
import os

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn.preprocessing as prep
import datetime
import matplotlib.dates as mdates
from sklearn.model_selection import train_test_split
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
import yellowbrick
from sklearn.preprocessing import StandardScaler
from yellowbrick.regressor import ResidualsPlot
from sklearn.linear_model import Ridge

pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
```

In [525]:

```
def symbol_to_path(symbol, base_dir="....""" 
    return os.path.join(base_dir, "{}.csv".format(str(symbol)))
```

In [526]:

```
def get_data(symbol, dates):
    """Read ETF data (adjusted close) from CSV file."""
    df = pd.DataFrame(index=dates)
    # Checking data
    print(" Missing value \n", df.isnull().values.any())
    # Read ETF data
    df_temp = pd.read_csv(symbol_to_path(symbol), index_col='Date', parse_dates=
True,
                          usecols=['Date','Signal','Close', 'Open','High','Low','Adj C
lose'], na_values=['nan'])
    print('df \n', df_temp.tail())
    df = df.join(df_temp)
    # drop dates ETF didn't trade
    df = df.dropna()
    return df
```

In [527]:

```
# Read data
symbol = "ETF"
start_date = "2015-11-19"
end_date = "2020-01-06"
dates = pd.date_range(start_date, end_date)  # date range as index
df = get_data(symbol, dates)  # get data for each symbol
etf_mean = df['Close'].mean()
etf_std = df['Close'].std()
```

```
 Missing value
 False
df
             Signal        Open        High         Low       Close  
Adj Close
Date
2019-12-30      0.0  165.979996  166.210007  164.570007  165.440002  
163.623688
2019-12-31      0.0  165.080002  166.350006  164.710007  165.669998  
163.851135
2020-02-01      0.0  166.740005  166.750000  164.229996  165.779999  
163.959946
2020-03-01      0.0  163.740005  165.410004  163.699997  165.130005  
163.317093
2020-06-01      0.0  163.850006  165.539993  163.539993  165.350006  
163.534668
```

In [528]:

```
# Data Cleaning and Outlier Detection
"""
This step further explores the dataset by checking for null or Not a Number (NA
N) values.
We drop the rows containing null values. After the elimination of null values, w
e look for the outliers.
To detect the outliers, Inter-Quantile Range (IQR) metric is used to measure the
dispersion/spread of data points.
By the rule of thumb is that if a value is larger than Q3 plus 1.5 times the IQ
R, then this value is an outlier.
IQR is the difference in the lower half (Q1) median and the upper half (Q3) medi
an of data given as:

IQR Formula
    IQR = Q3- Q1
    Outliers :
        if datapoint > Q3 + 1.5 x IQR
        if datapoint < Q1 + 1.5 x IQR
    Q1 and Q3 are the first and third quartiles, respectively, of all the values
in that column,
    and IQR = Q3 − Q1 is the interquartile of those values. The symmetry of all
 adjusted and cleaned
    columns can be checked using histograms or statistical tests.

"""

"""
Z-score
The Z-score (or standard score) is obtained by subtracting the mean of the datas
et from each data point and normalizing the result by dividing by the standard d
eviation of the dataset.
In other words, the Z-score of a data point represents the distance in the numbe
r of standard deviations that the data point is away from the mean of all the da
ta points.
For a normal distribution (applicable for large enough datasets) there is a dist
ribution rule of 68-95- 99, summarized as follows:
-       68% of all data will lie in a range of one standard deviation from the m
ean.
-       95% of all data will lie in a range of two standard deviations from the
 mean.
-       99% of all data will lie within a range of three standard deviations fro
m the mean.
"""
```

Out[528]:

```
'\nZ-score \nThe Z-score (or standard score) is obtained by subtract
ing the mean of the dataset from each data point and normalizing the
result by dividing by the standard deviation of the dataset. \nIn ot
her words, the Z-score of a data point represents the distance in th
e number of standard deviations that the data point is away from the
mean of all the data points. \nFor a normal distribution (applicable
for large enough datasets) there is a distribution rule of 68-95- 9
9, summarized as follows: \n-\t68% of all data will lie in a range o
f one standard deviation from the mean.\n-\t95% of all data will lie
in a range of two standard deviations from the mean.\n-\t99% of all
data will lie within a range of three standard deviations from the m
ean. \n'
```

In [529]:

```
# Check for missing values - Quality 1
if df.isnull().values.any():
    print("There are missing values in data.csv\n")
    print(df.isnull().sum())
else:
    print("There are no missing values in data.csv\n")

#Check if ETF Low higher than high - Quality 2
df.drop(df.index[df['Low']> df['High']], inplace=True)
# TEST
filter_1 = df[df['Low'] > df['High']]
print('filter \n', filter_1)
```

```
There are no missing values in data.csv

filter
 Empty DataFrame
Columns: [Signal, Open, High, Low, Close, Adj Close]
Index: []
```

In [530]:

```
# Checking outliers - Quality 3
def indentify_outliers(row, n_sigmas=3):
    x = row['simple_rtn']
    mu = row['mean']
    sigma = row['std']
    if (x > mu + 3 * sigma) | (x < mu - 3 * sigma):
        return 1
    else:
        return 0
```

In [531]:

```
print('ETF Data info \n', df.info())
print('ETF Data description \n',df.describe())
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1024 entries, 2015-11-19 to 2019-12-31
Data columns (total 6 columns):
Signal       1024 non-null float64
Open         1024 non-null float64
High         1024 non-null float64
Low          1024 non-null float64
Close        1024 non-null float64
Adj Close    1024 non-null float64
dtypes: float64(6)
memory usage: 56.0 KB
ETF Data info
 None
ETF Data description
              Signal         Open         High          Low         Cl
ose    Adj Close
count  1024.000000  1024.000000  1024.000000  1024.000000  1024.0000
00  1024.000000
mean     16.843827   141.964209   142.808428   141.020518   141.9594
34   136.468353
std       2.956264    18.384840    18.376066    18.301746    18.3974
36    21.340875
min       0.000000    94.080002    95.400002    93.639999    94.7900
01  -152.277847
25%      14.897133   132.502495   133.912498   131.664997   132.7200
01   127.211576
50%      17.375832   146.810005   148.040001   145.715004   147.0800
02   142.747726
75%      19.036140   155.362500   156.250003   154.340000   155.2750
01   151.794083
max      35.434147   172.789993   173.389999   171.949997   196.2799
99   168.842270
```

In [532]:

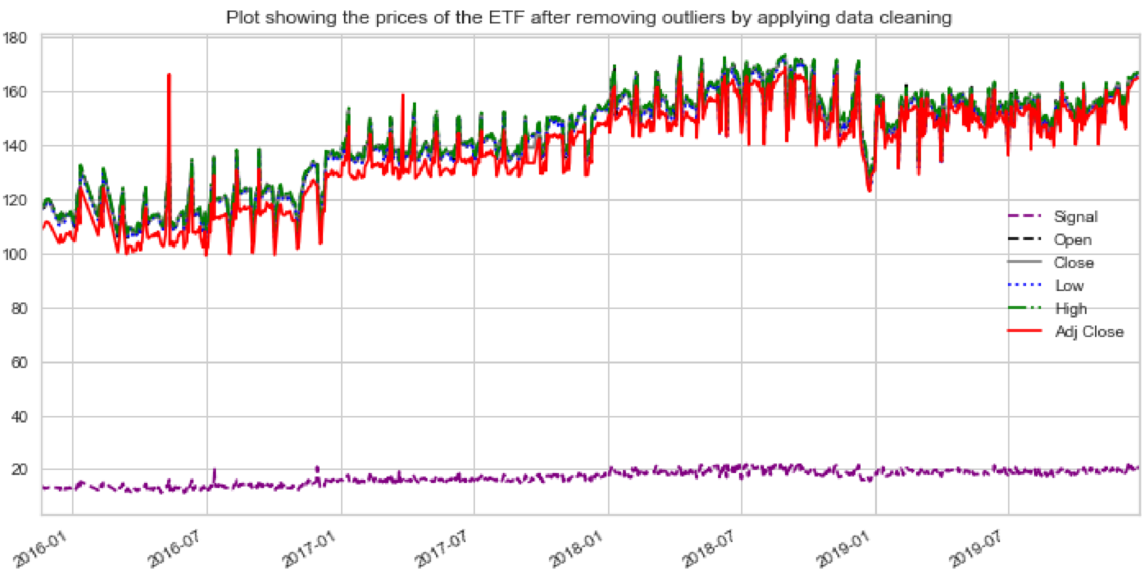
```
# Checking outliers - Quality 3
# Summary of outliers and how to deal with the outliers and correction to be app
lied.

""" Comment - •
The preceding output provides quick summary statistics for every field in our Da
taFrame. Key observations from table above are outlined here:
o       'Adj Close' has a minimum value of -152.27, which is unlikely to be true
for the following reasons:
        The other price fields—Open, High , Low , and Close —all have minimum va
lues around 93/94, so it doesn't make sense for Ajd_Close to have a minimum valu
e of -152.277.
         Given that the 25th percentile for Adj_Close is 125.29, it is unlikely
 that the minimum value would be so much lower than that.
        Two other outliers – 166.17 and 158.57, where the Adj Close is well abov
e the High of the day and that's incorrect.
        The price of an asset should be non-negative.
"""
```

Out[532]:

```
" Comment - •\t\nThe preceding output provides quick summary statist
ics for every field in our DataFrame. Key observations from table ab
ove are outlined here: \no\t'Adj Close' has a minimum value of -152.
27, which is unlikely to be true for the following reasons:\n\uf0a7
\tThe other price fields—Open, High , Low , and Close —all have mini
mum values around 93/94, so it doesn't make sense for Ajd_Close to h
ave a minimum value of -152.277.\n\uf0a7\t Given that the 25th perce
ntile for Adj_Close is 125.29, it is unlikely that the minimum value
would be so much lower than that.\n\uf0a7\tTwo other outliers – 166.
17 and 158.57, where the Adj Close is well above the High of the day
and that's incorrect.\n\uf0a7\tThe price of an asset should be non-n
egative. \n"
```

In [533]:

```
#Libraries
import scipy
import scipy.stats as scs
import statsmodels.api as sm
import statsmodels.tsa.api as smt

# No outlier visualisation - all columns except Signal
no_outlier_prices = df[(np.abs(scipy.stats.zscore(df)) < 2).all(axis=1)]
no_outlier_prices['Signal'].plot(figsize=(12, 6), linestyle='--', color='purple'
, legend='Open')
no_outlier_prices['Open'].plot(figsize=(12, 6), linestyle='--', color='black', l
egend='Open')
no_outlier_prices['Close'].plot(figsize=(12, 6), linestyle='-', color='grey', le
gend='Close')
no_outlier_prices['Low'].plot(figsize=(12, 6), linestyle=':', color='blue', lege
nd='Low')
no_outlier_prices['High'].plot(figsize=(12, 6), linestyle='-.', color='green', l
egend='High')
no_outlier_prices['Adj Close'].plot(figsize=(12, 6), linestyle='solid', color='r
ed', legend='Adj Close')
plt.title('Plot showing the prices of the ETF after removing outliers by applyin
g data cleaning')
plt.savefig('Plot showing the prices of the ETF after removing outliers by apply
ing data cleaning.png', dpi=300)
plt.show()

print('No outlier data describe \n',no_outlier_prices[['Signal','Open', 'Close',
'Low', 'High', 'Adj Close']].describe())
```




```
No outlier data describe
              Signal          Open         Close           Low          High
Adj Close
count   980.000000  980.000000  980.000000  980.000000  980.000000  9
80.000000
mean     17.067660  143.398296  143.357531  142.468347  144.237275  1
38.236435
std       2.574710   16.822460   16.767767   16.711273   16.834933
17.799841
min      11.093390  105.730003  105.669998  104.809998  107.010002
99.015808
25%      15.253656  134.932499  134.880001  134.047501  135.685001  1
28.775867
50%      17.602526  147.630005  147.904998  146.534996  149.014999  1
43.409569
75%      19.075301  155.527504  155.362500  154.479996  156.457497  1
51.897102
max      21.712125  172.789993  172.500000  171.949997  173.389999  1
68.842270
```

In [534]:

```
# Checking outliers - Quality 3
# Summary of outliers and how to deal with the outliers and correction to be app
lied.

"""So, after computing Z-scores of all data points in our dataset, there is an a
pproximately 5% chance of a data point having a Z-score larger than or equal to
 2.
Therefore, we can use this information to filter out all observations with Z-sco
res of 2 or higher to detect and remove outliers.
In our case, we will remove all rows with values whose Z-score is less than -2 o
r greater than 2—that is, three standard deviations away from the mean.
The plot clearly shows that the earlier observation of extreme value for Adj Clo
se has been discarded; there is no longer the dip of -152. Note that while we re
moved the extreme outlier, we were still able to preserve the sharp spikes in pr
ices during 2016 and 2018, thus not leading to a lot of data losses.
"""
```

Out[534]:

```
'So, after computing Z-scores of all data points in our dataset, the
re is an approximately 5% chance of a data point having a Z-score la
rger than or equal to 2. \nTherefore, we can use this information to
filter out all observations with Z-scores of 2 or higher to detect a
nd remove outliers. \nIn our case, we will remove all rows with valu
es whose Z-score is less than -2 or greater than 2—that is, three st
andard deviations away from the mean. \nThe plot clearly shows that
the earlier observation of extreme value for Adj Close has been disc
arded; there is no longer the dip of -152. Note that while we remove
d the extreme outlier, we were still able to preserve the sharp spik
es in prices during 2016 and 2018, thus not leading to a lot of data
losses. \n'
```

In [535]:

```
"""
These statistics look significantly better—as we can see in the following screen
shot,
the min and max values for the Adj Close now looks in line with expectations and
do not
have extreme value, except for the Close.
"""
```

Out[535]:

```
'\nThese statistics look significantly better—as we can see in the f
ollowing screenshot, \nthe min and max values for the Adj Close now
looks in line with expectations and do not \nhave extreme value, exc
ept for the Close.\n'
```

In [536]:

```
# Advanced visualisation techniques
    # Daily close price and signal changes
""" Next, let's compute the daily close price changes, which inspect the summary
statistics for this new DataFrame
to get a sense of how the delta price values are distributed, as follows:
"""
```

Out[536]:

```
" Next, let's compute the daily close price changes, which inspect t
he summary statistics for this new DataFrame \nto get a sense of how
the delta price values are distributed, as follows: \n"
```

In [537]:

```
# Advanced visualisation techniques
    # Daily close price and signal changes
close_prices = no_outlier_prices[['Close', 'Signal']]
delta_close_prices = (close_prices.shift(-1) - close_prices).fillna(0)
delta_close_prices.columns = ["ETF Delta Close", "ETF Delta Signal"]
print('daily returns describe \n',delta_close_prices.describe())
```

```
daily returns describe
        ETF Delta Close  ETF Delta Signal
count       980.000000        980.000000
mean          0.051653          0.006376
std           5.468869          1.127291
min         -27.149994         -7.422303
25%          -1.185004         -0.584704
50%           0.215003          0.036817
75%           1.649994          0.640557
max          23.399994          5.406943
```

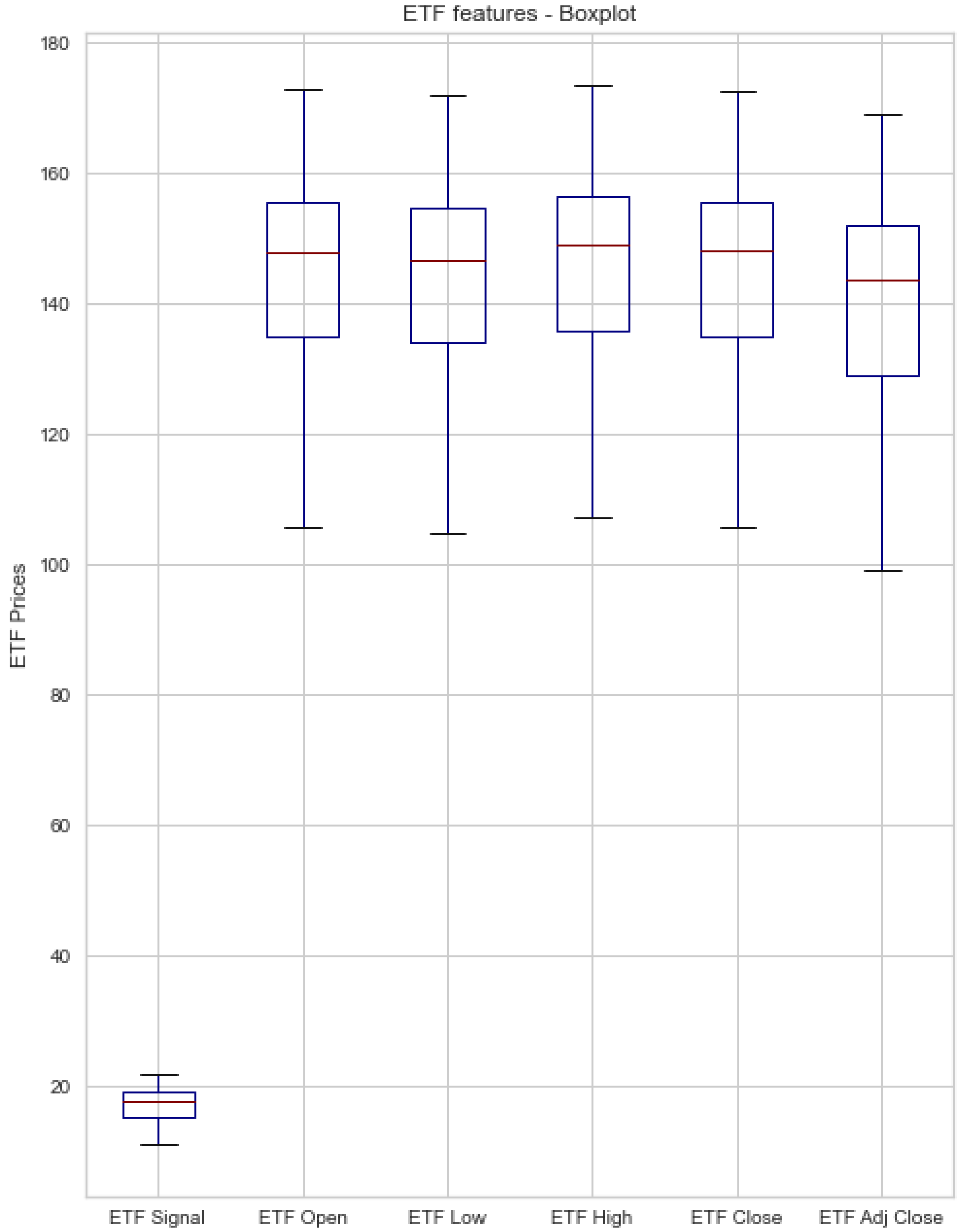
In [538]:

```
print("We can observe from these statistics that both features values’ means are
close to 0, the ETF seems \
to experience small price moves (from the std field). However, the max differenc
e for both is quite large. ")
```

```
We can observe from these statistics that both features values’ mean
s are close to 0, the ETF seems to experience small price moves (fro
m the std field). However, the max difference for both is quite larg
e.
```

In [539]:

```
# Box plot of all features - outliers
df = no_outlier_prices
united_box = pd.concat([df['Signal'],df['Open'], df['Low'], df['High'], df['Clos
e'], df['Adj Close']],
                           axis=1)
united_box.columns = ['ETF Signal','ETF Open', 'ETF Low', 'ETF High', 'ETF Clos
e', 'ETF Adj Close']
united_box.plot(kind='box', figsize=(8, 11), colormap='jet')
plt.ylabel('ETF Prices')
plt.title('ETF features - Boxplot')
plt.savefig('ETF features - Boxplot')
plt.show()
```



In [540]:

```
"""
The data is visualized using a box plot. For the ‘Close’ variable, we got no out
liers, as shown in Fig. 4 below.
For the ‘Volume’ variable, we see there are outliers. We remove the outliers by
 considering the data only in the range
of (Q1−1.5 IQR) and Q3 + 1.5 IQR. Lastly, the feature standardization is done.
The Fig. 7 shows the visualization of the ‘Volume’ feature with outliers, and Fi
g. 8 shows the ‘Close’ and
cleaned ‘Volume’ feature. Figure 9 is the snapshot of the prepared dataset.
"""
```

Out[540]:

```
'\nThe data is visualized using a box plot. For the ‘Close’ variabl
e, we got no outliers, as shown in Fig. 4 below. \nFor the ‘Volume’
variable, we see there are outliers. We remove the outliers by consi
dering the data only in the range \nof (Q1−1.5 IQR) and Q3 + 1.5 IQ
R. Lastly, the feature standardization is done. \nThe Fig. 7 shows t
he visualization of the ‘Volume’ feature with outliers, and Fig. 8 s
hows the ‘Close’ and \ncleaned ‘Volume’ feature. Figure 9 is the sna
pshot of the prepared dataset.\n'
```

In [541]:

```
# Histogram plot
"""
Let's observe the distribution of Close price of the ETF to get more familiar wi
th it, using a histogram plot.
In the following histogram, we can see that the distribution is approximately no
rmally distributed:

"""
```

Out[541]:

```
"\nLet's observe the distribution of Close price of the ETF to get m
ore familiar with it, using a histogram plot. \nIn the following his
togram, we can see that the distribution is approximately normally d
istributed: \n\n"
```

In [542]:

```
# Advanced visualisation techniques
    # Daily close price and signal changes
close_signal_prices = df[['Close','Signal']].dropna()
print('daily returns describe \n', close_signal_prices.describe())
delta_prices = (close_signal_prices.shift(-1) - close_signal_prices).fillna(0)
delta_prices.columns = ["ETF Delta Close", "ETF Delta Signal"]
print('daily returns describe \n', delta_prices.describe())
```

```
daily returns describe
             Close      Signal
count  980.000000  980.000000
mean   143.357531   17.067660
std     16.767767    2.574710
min    105.669998   11.093390
25%    134.880001   15.253656
50%    147.904998   17.602526
75%    155.362500   19.075301
max    172.500000   21.712125
daily returns describe
        ETF Delta Close  ETF Delta Signal
count       980.000000        980.000000
mean          0.051653          0.006376
std           5.468869          1.127291
min         -27.149994         -7.422303
25%          -1.185004         -0.584704
50%           0.215003          0.036817
75%           1.649994          0.640557
max          23.399994          5.406943
```

In [543]:

```
# Comment on the above results
"""
We can observe from these statistics that both features values' means are close
 to 0,
the ETF seems to experience small price moves (from the std field).
"""
```

Out[543]:

```
'\nWe can observe from these statistics that both features values' m
eans are close to 0, \nthe ETF seems to experience small price moves
(from the std field). \n'
```

In [544]:

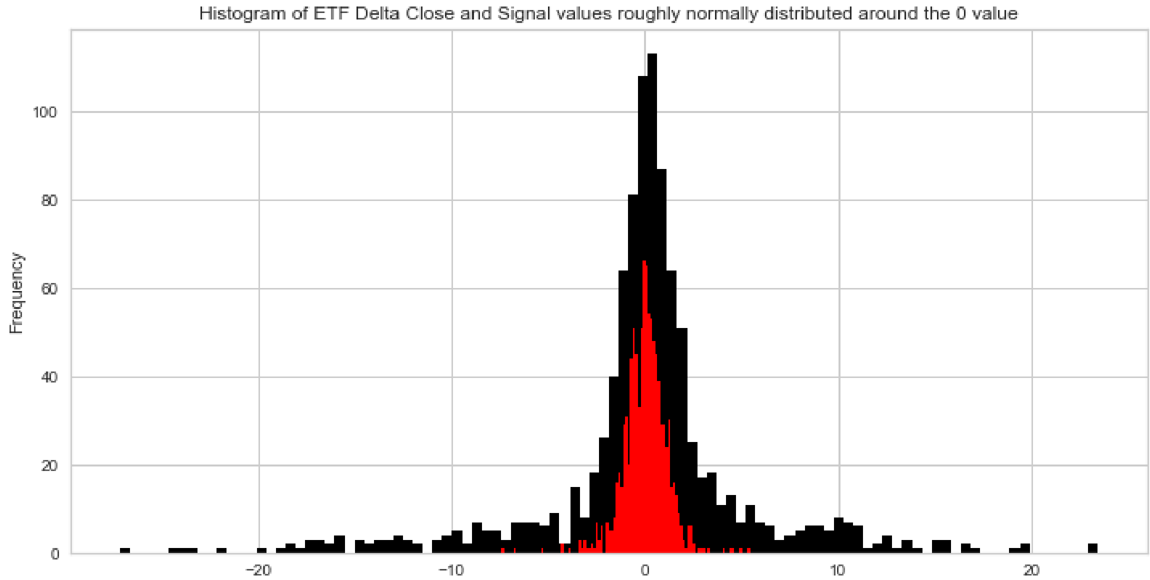
```
# Histogram plot
"""
Let's observe the distribution of Close and signal of the ETF to get more famili
ar with it, using a histogram plot.
"""
```

Out[544]:

```
"\nLet's observe the distribution of Close and signal of the ETF to
get more familiar with it, using a histogram plot. \n"
```

In [545]:

```
# Histogram plot
delta_prices['ETF Delta Close'].plot(kind='hist', bins=100, figsize=(12, 6), col
or='black',grid=True)
delta_prices['ETF Delta Signal'].plot(kind='hist', bins=100, figsize=(12, 6), co
lor='red',grid=True)
plt.title('Histogram of ETF Delta Close and Signal values roughly normally distr
ibuted around the 0 value')
plt.savefig ( 'ETF Delta Close & Signal.png', dpi=300 )
plt.show()
```



In [546]:

```
# Comment on the above histograms
"""
In the above histogram, we can see that the distribution is approximately normal
ly distributed:
"""
```

Out[546]:

```
'\nIn the above histogram, we can see that the distribution is appro
ximately normally distributed: \n'
```

In [547]:

```
# 2 - Signal analysis and return forecasting
```

In [548]:

```
# Feature matrix
x = df[['Signal']]

# Response vector
y = df['Close']

# Split our data
x_train, x_test, y_train, y_test  = train_test_split(x,y)

scaler = StandardScaler()
# Fit to the training data
scaler.fit(x_train)

# Now apply the transformations to the data:
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(x_train, y_train)

y_pred = regr.predict(x_test)

# The mean squared error
print("Mean squared error: %.2f"% mean_squared_error(y_test, y_pred))

# Explained variance score: 1 is perfect prediction
print('R-squared = : %.2f' % r2_score(y_test, y_pred))
```

```
Mean squared error: 29.99
R-squared = : 0.89
```

In [549]:

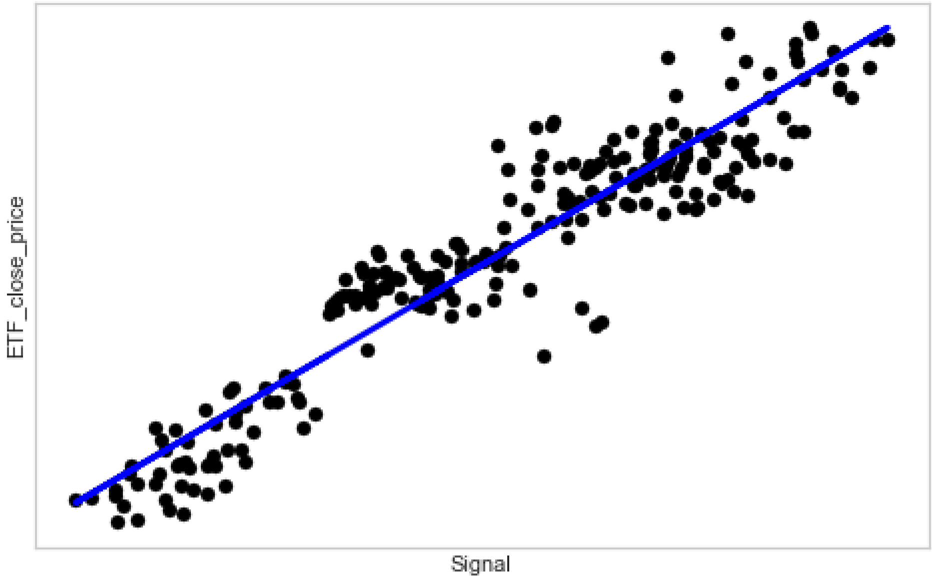
```
"""
We get the value of r-squared to be 0.90 or 90% which indicates that our model i
s a good fit for our data
and the signal has a correct power to predict the ETF_close_price, but it can be
better.
Let's plot our regression line to see how close our ETF_close_price is to our mo
del.
"""
```

Out[549]:

```
"\nWe get the value of r-squared to be 0.90 or 90% which indicates t
hat our model is a good fit for our data \nand the signal has a corr
ect power to predict the ETF_close_price, but it can be better. \nLe
t's plot our regression line to see how close our ETF_close_price is
to our model.\n"
```

In [550]:

```
# Plot outputs
plt.scatter(x_test, y_test,  color='black')
plt.plot(x_test, y_pred, color='blue', linewidth=3)
plt.xlabel("Signal")
plt.ylabel("ETF_close_price")

plt.xticks(())
plt.yticks(())

plt.show()
```



In [551]:

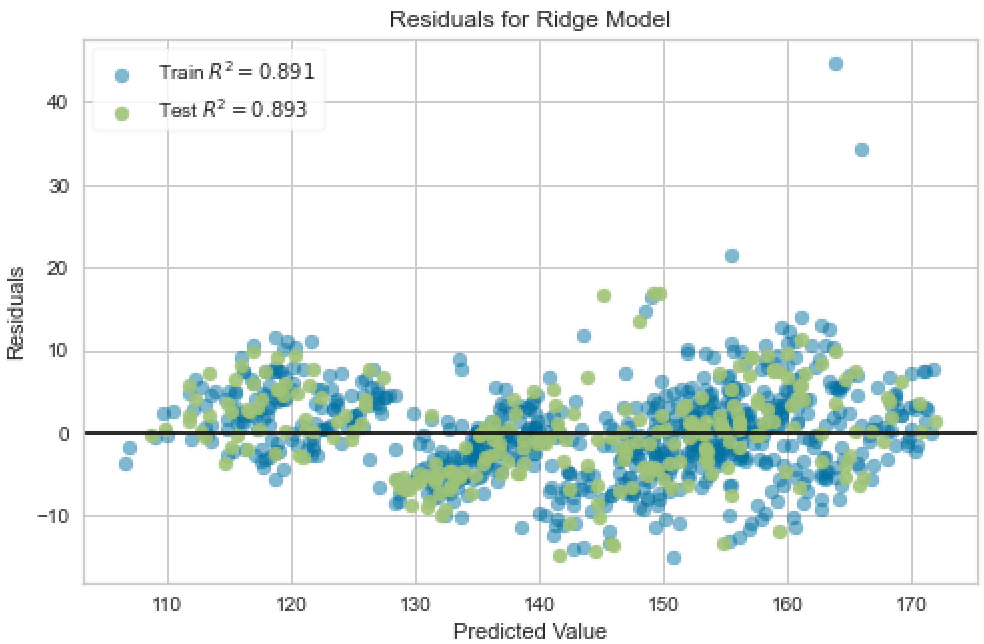
```
# Comment on the plot outputs above
"""
The above plot indicates that the data is scattered along the fitted regression
 line.
Therefore, we will do a residual plot on our data to see if linear regression is
indeed a good fit for our data.
"""
```

Out[551]:

```
'\nThe above plot indicates that the data is scattered along the fit
ted regression line.\nTherefore, we will do a residual plot on our d
ata to see if linear regression is indeed a good fit for our dat
a.\n'
```

In [552]:

```
# Instantiate the linear model and visualizer
from yellowbrick.regressor import ResidualsPlot
from sklearn.linear_model import Ridge

ridge = Ridge()
visualizer = ResidualsPlot(ridge)

visualizer.fit(x_train, y_train)  # Fit the training data to the visualizer
visualizer.score(x_test, y_test)  # Evaluate the model on the test data
g = visualizer.poof()             # Draw/show/poof the data
```



In [553]:

```
# Comment on the "Residuals for Ridge Model plot
# 2 - Please analyze the signal's effectiveness or lack thereof in forecasting E
TF price, using whatever metrics
# you think are most relevant.
"""
After doing a residual plot, we see that our data points are  scattered along th
e horizontal axis.
This indicates that linear regression can be a good model for our data as the R-
squared of our test is about 90%,
In the following section, we will be using different models to compare the resul
ts of this linear regression model
and see if we can considere another model or other features will need to be used
in order to improve the performance
of our model.
"""
```

Out[553]:

```
'\nAfter doing a residual plot, we see that our data points are  sca
ttered along the horizontal axis. \nThis indicates that linear regre
ssion can be a good model for our data as the R-squared of our test
is about 90%,\nIn the following section, we will be using different
models to compare the results of this linear regression model\nand s
ee if we can considere another model.\n'
```

In [555]:

```
print('Other models to compare with the linear regression model - ETF')

# Function and modules for data preparation and visualization
    # pandas, pandas_datareader, numpy and matplotlib
import os
import pandas as pd
import numpy as np

#Libraries for Statistical Models
import seaborn as sns
#import stats
import scipy
import scipy.stats as scs
import statsmodels.api as sm
import statsmodels.tsa.api as smt

#Plotting
from scipy.stats import probplot
import matplotlib.pyplot as pyplot
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf


#Libraries - Scatterplot Matrix
from pandas.plotting import scatter_matrix

# Pipeline
from sklearn.pipeline import Pipeline

# Libraries
#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.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer

# Feature Selection
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2, f_regression

#Diable the warnings
import warnings
```

```
warnings.filterwarnings('ignore')

# Model metrics
from sklearn.metrics import (accuracy_score, mean_absolute_error,explained_varia
nce_score, r2_score)
```

Other models to compare with the linear regression model - ETF

In [556]:

```
# Feature Engineering and Selection
"""
Sometimes training the model using the default features of a dataset result in p
oor predictions.
To increase the quality of input data, we derive more relevant attributes from e
xisting features and extend feature
space by including external data. It is the most critical step as it directly im
pacts the prediction accuracy.
Historical lags or stock price across the last ‘n’ days is an important stock in
dicator that affects stock price.
It could be an essential feature for the prediction models. Lag is used to measu
re the strength of a trend.
The methodology has used four lag-based features to analyze the effect of the si
gnal of the last two months on
the current closing price.

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

We now define our Y series and our X series

Y: ETF Future Returns
X:
    ETF 5 Business Day Returns based on the signal
    ETF 15 Business Day Returns based on the signal
    ETF 30 Business Day Returns based on the signal
    ETF 60 Business Day Returns based on the signal

"""
```

Out[556]:

```
'\nNext, we need a series to predict. We choose to predict using wee
kly returns. \nWe approximate this by using 5 business day period re
turns.\n\nWe now define our Y series and our X series\n\nY: ETF Futu
re Returns\nX:\n    ETF 5 Business Day Returns based on the signal
\n    ETF 15 Business Day Returns based on the signal    \n    ETF 30
Business Day Returns based on the signal    \n    ETF 60 Business Da
y Returns based on the signal\n\n'
```

In [557]:

```
# Machine learning models

return_period = 5
Y = np.log(df['Close']).diff(return_period).shift(-return_period)

Y.name = Y.name+'_pred'
X1 = pd.concat([np.log(df['Signal']).diff(i) for i in [return_period, return_per
iod*3,return_period*6,
                                                          return_period*12]],axi
s=1).dropna()
X1.columns = ['ETF_DT', 'ETF_3DT', 'ETF_6DT', 'ETF_12DT']
X = pd.concat([X1], axis=1)
dataset = pd.concat([Y, X], axis=1).dropna().iloc[::return_period, :]

dataset = dataset.replace([np.inf, -np.inf], np.nan)
dataset = dataset.dropna()
X = X.replace([np.inf, -np.inf], np.nan)
X = X.dropna()

Y = Y.replace([np.inf, -np.inf], np.nan)
Y = Y.dropna()
```

In [558]:

```
print('dataset \n',dataset.head())
```

```
dataset
             Close_pred    ETF_DT   ETF_3DT   ETF_6DT  ETF_12DT
2016-04-14     0.007284  0.068504  0.134967 -0.065016 -0.034639
2016-04-21     0.004504 -0.030265  0.078404 -0.004074  0.001820
2016-04-28     0.001673  0.014533  0.052771  0.070441  0.051372
2016-05-07    -0.023675  0.001483 -0.014250  0.120718  0.042456
2016-05-16    -0.003338 -0.025603 -0.009587  0.068817  0.028177
```

In [559]:

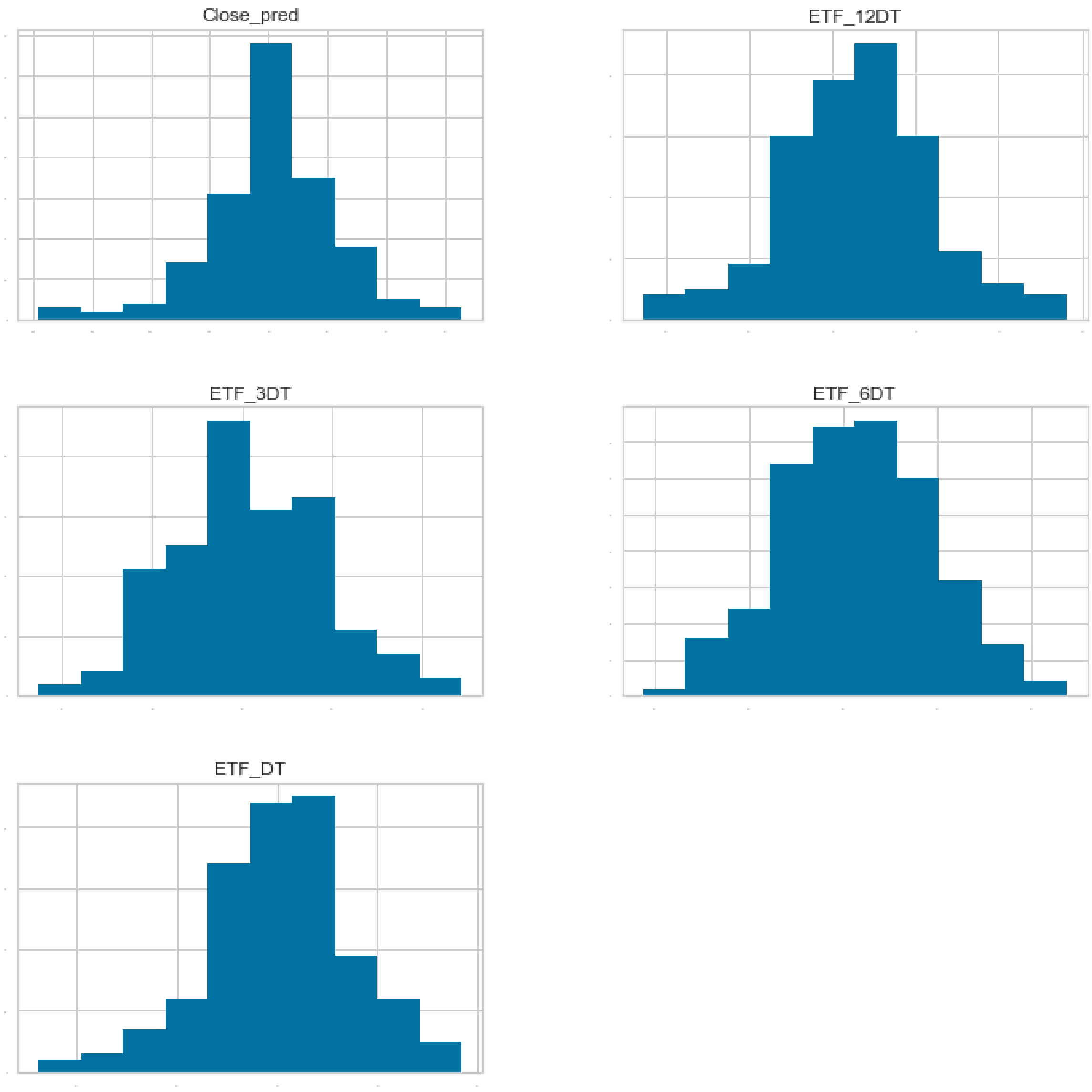
```
"""
The variable Close_pred is the return of ETF stock and is the predicted variabl
e.
The dataset contains the lagged series of the ETF calculated based on the signa
l.
Additionally, it also consists of the lagged historical returns of ETF.

"""
```

Out[559]:

```
'\nThe variable Close_pred is the return of ETF stock and is the pre
dicted variable. \nThe dataset contains the lagged series of the ETF
calculated based on the signal. \nAdditionally, it also consists of
the lagged historical returns of ETF.\n\n'
```

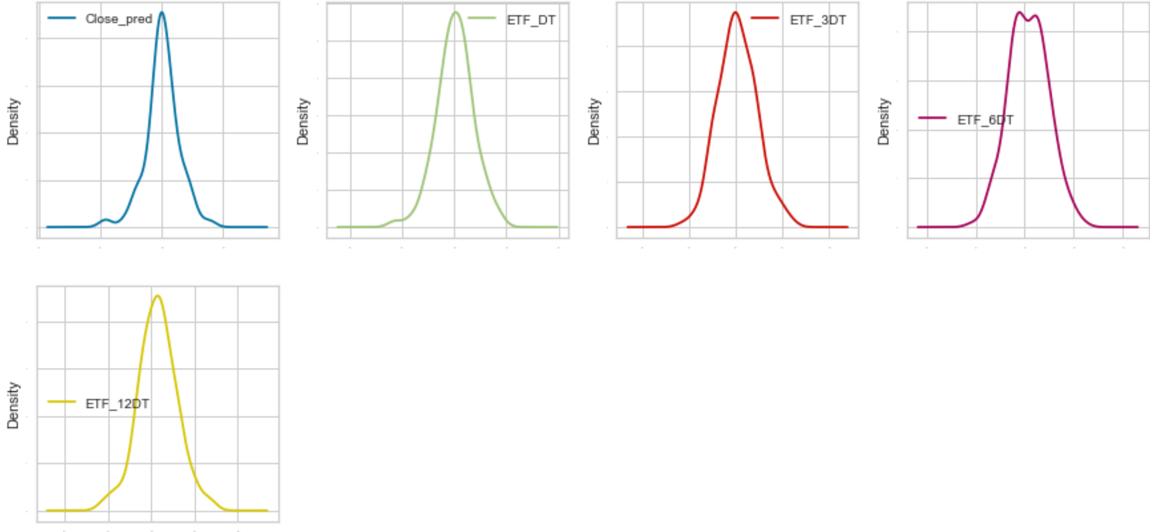
In [560]:

```
# 3 - Data Visualization
# histograms
dataset.hist(sharex=False, sharey=False, xlabelsize=1, ylabelsize=1, figsize=(12
,12))
plt.show()
```



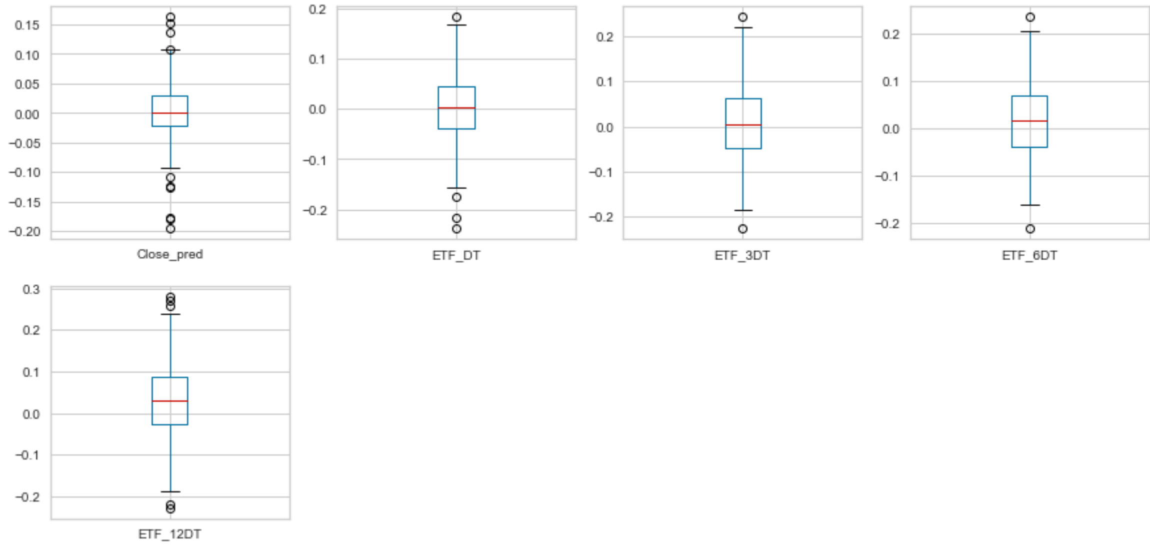
In [561]:

```
# density
dataset.plot(kind='density', subplots=True, layout=(4,4), sharex=False, legend=True, fontsize=1, figsize=(15,15))
plt.show()
```



In [562]:

```
# Box and Whisker Plots
dataset.plot(kind='box', subplots=True, layout=(4,4), sharex=False, sharey=False, figsize=(15,15))
plt.show()
```

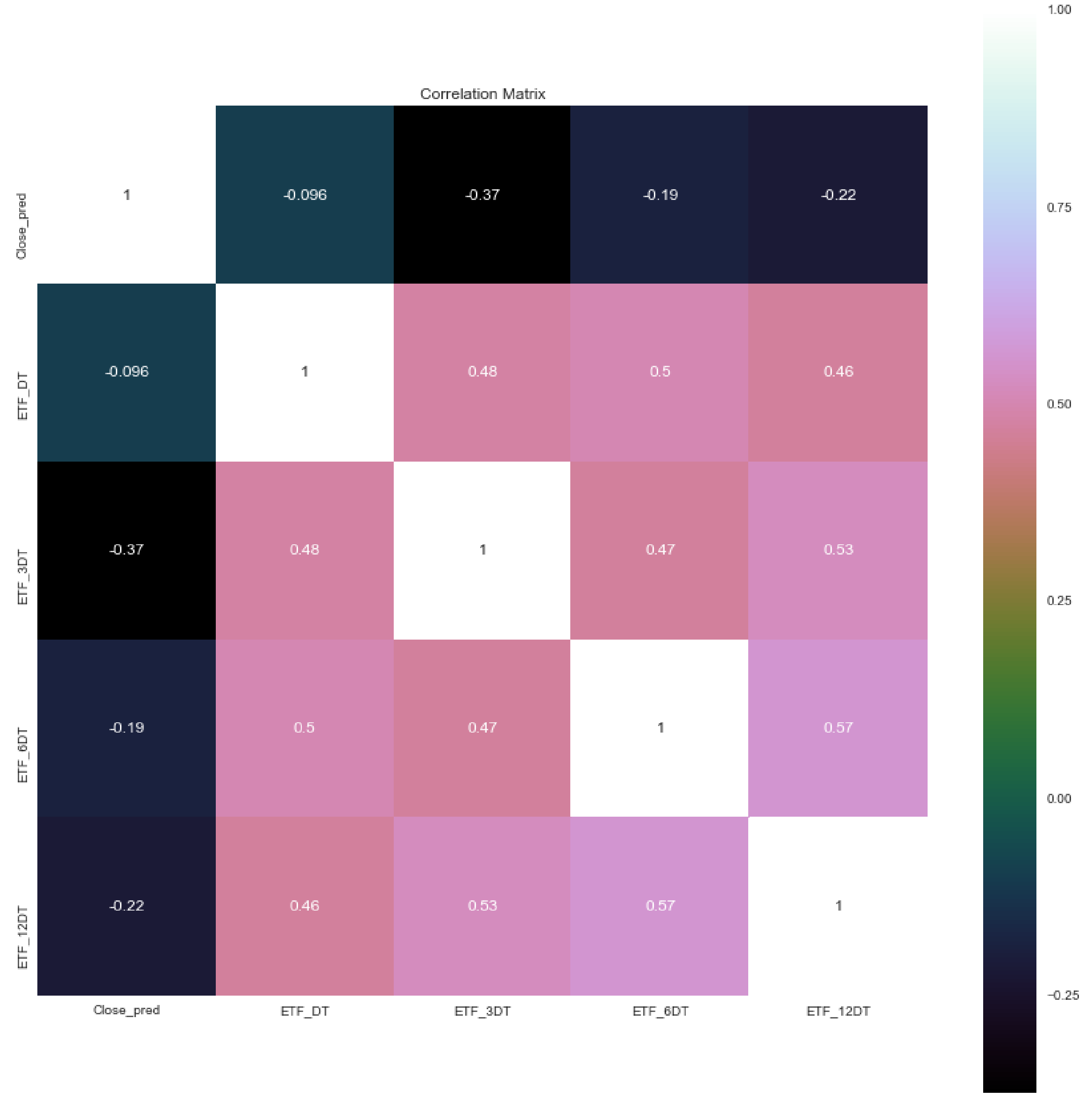


In [563]:

```
# correlation
correlation = dataset.corr()
plt.figure(figsize=(15,15))
plt.title('Correlation Matrix')
plt.savefig('Correlation Matrix.png')
sns.heatmap(correlation, vmax=1, square=True,annot=True,cmap='cubehelix')
```

Out[563]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a22fc6a90>
```



In [564]:

```
# Comment on the heatmap matrix

"""
We see a negative correlation between the closing price return and the return de
termined based on the signal.
used to build predictive trading models with three algorithms: regression analys
is, generalized linear modeling, and chi-square automatic interaction detection.
 An ensemble from the combination of the best two models is then created.
"""
```

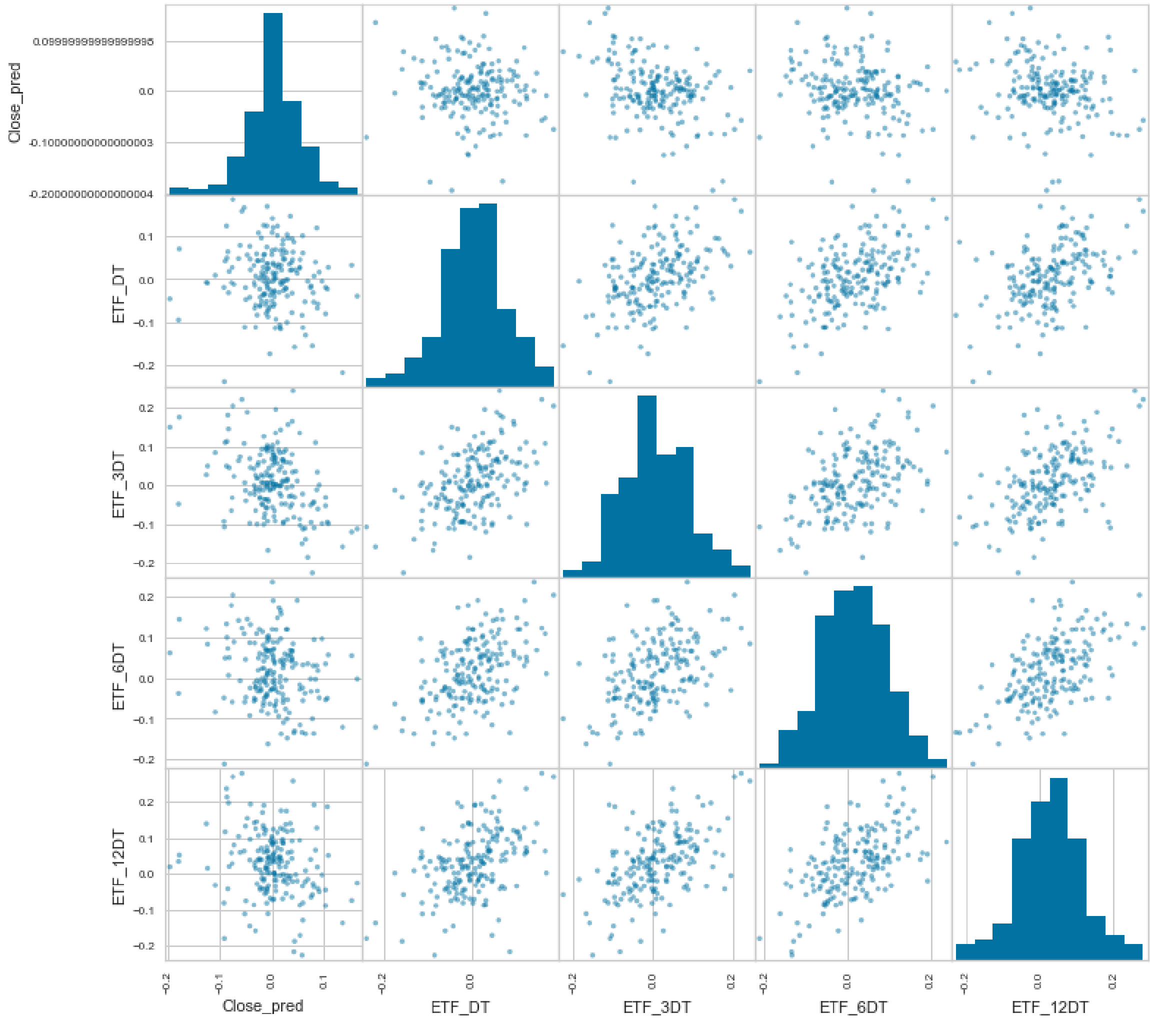
Out[564]:

```
'\nWe see a negative correlation between the closing price return an
d the return determined based on the signal.\nused to build predicti
ve trading models with three algorithms: regression analysis, genera
lized linear modeling, and chi-square automatic interaction detectio
n. \n An ensemble from the combination of the best two models is the
n created.\n'
```

In [565]:

```
# Scatterplot Matrix
from pandas.plotting import scatter_matrix

plt.figure(figsize=(15,15))
scatter_matrix(dataset,figsize=(12,12))
plt.savefig("Scatterplot Matrix.png")
plt.show()
```

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



Close_pred
ETF_DT
ETF_3DT
ETF_6DT
ETF_12DT
0.09999999999999995
0.0
-0.10000000000000003
-0.20000000000000004
0.1
-0.1
-0.2
0.2

In [566]:

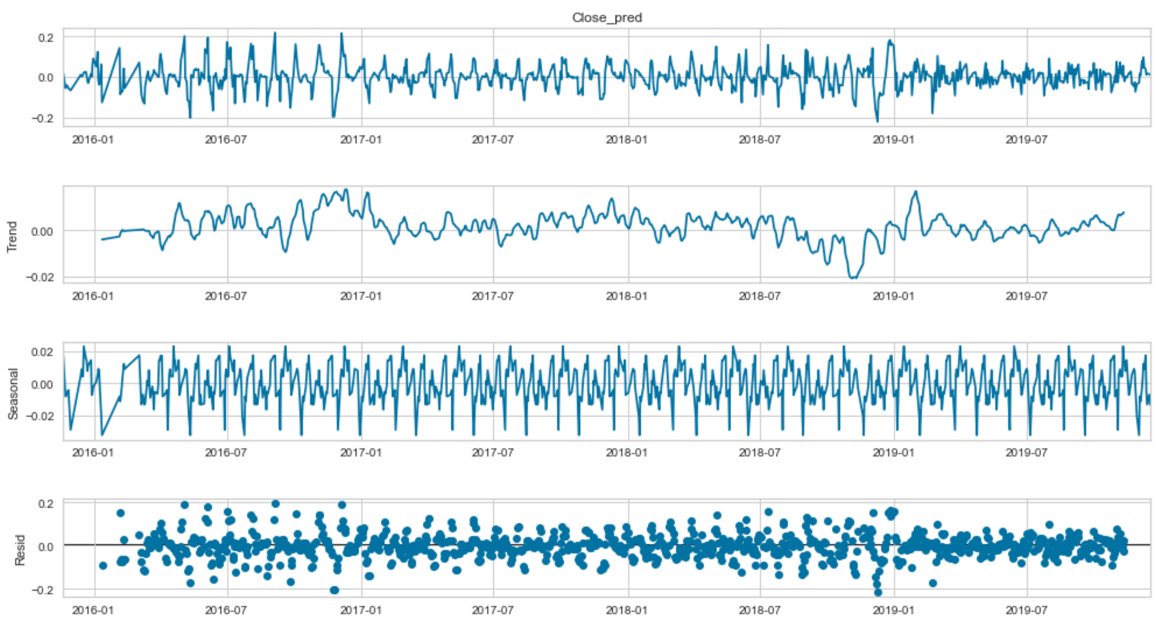
```
# Comment on the above results
"""
By looking at the scatterplots, we can visualize the relationship between all th
e variables in the
regression using the scatter matrix but it's hard to see any special relationshi
p of the predicted variable
with the lagged 15-day, 30-day, and 60-day returns of the ETF.
"""
```

Out[566]:

```
"\nBy looking at the scatterplots, we can visualize the relationship
between all the variables in the \nregression using the scatter matr
ix but it's hard to see any special relationship of the predicted va
riable \nwith the lagged 15-day, 30-day, and 60-day returns of the E
TF.\n"
```

In [567]:

```
# Time series analysis
# Libraries for Statistical Models
import statsmodels.api as sm
res = sm.tsa.seasonal_decompose(Y,freq=52)
fig = res.plot()
fig.set_figheight(8)
fig.set_figwidth(15)
plt.savefig('TSA seasonal.png')
plt.show()
```



In [568]:

```
# Comment on the above results
"""
We can see that for our ETF, there has been a general up and down trend in the r
eturn series, with more negative return
data points in the second half of 2018. The residual(or white noise)term is rela
tively small over the entire time
series.
"""
```

Out[568]:

```
'\nWe can see that for our ETF, there has been a general up and down
trend in the return series, with more negative return\ndata points i
n the second half of 2018. The residual(or white noise)term is relat
ively small over the entire time \nseries. \n'
```

In [569]:

```
# Evaluate models
#1 - Train-test split and evaluation metrics
"""
It is a good idea to partition the original dataset into a training set and a te
st set. The test set is a sample of
the data that we hold back from our analysis and modeling. We use it right at th
e end of our project to confirm
the performance of our final model. It is the final test that gives us confidenc
e on our estimates of accuracy
on unseen data. We will use 70% of the dataset for modeling and use 30% for test
ing because of the length of our dataset,
even though it's generally advised to use 80% for traing and 20% for testing. Wi
th time series data,
the sequence of values is important. So we do not distribute the dataset into tr
aining and test sets in random fashion,
but we select an arbitrary split point in the ordered list of observations and c
reate two new datasets:
"""

validation_size = 0.40
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 [570]:

```
# 2- Test options and evaluation metrics
"""
We use the prebuilt sklearn models to run a k-fold analysis on our training dat
a. We then train the model on the
full training data and use it for prediction of the test data. We will evaluate
 algorithms using the mean
squared error metric. The parameters for the k-fold analysis and evaluation metr
ics are defined as follows:

"""

num_folds = 10
scoring = 'neg_mean_squared_error'
```

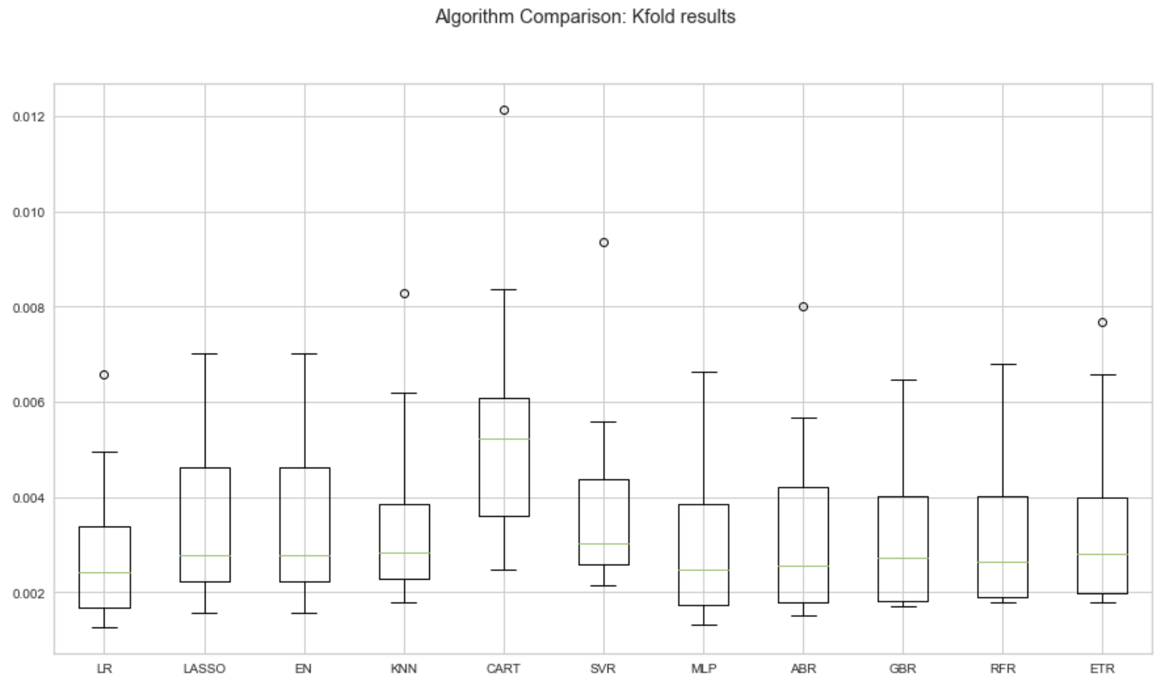
In [571]:

```
# 3. Compare models and algorithms
# 3.1 - Machine learning models from Scikit-learn

#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

# Regression and tree regression algorithms
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
models.append(('MLP', MLPRegressor()))
#Ensemble models
# Boosting methods
models.append(('ABR', AdaBoostRegressor()))
models.append(('GBR', GradientBoostingRegressor()))
# Bagging methods
models.append(('RFR', RandomForestRegressor()))
models.append(('ETR', ExtraTreesRegressor()))
```

In [572]:

```
#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
from sklearn.model_selection import KFold


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

for name, model in models:
    names.append(name)
    # k-fold analysis:
    kfold = KFold(n_splits=num_folds, random_state=None)
    # converted mean squared error to positive. The lower the better
    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)
    #print('res \n',res)
    train_result = mean_squared_error(res.predict(X_train), Y_train)
    #print('train_result \n',train_result)
    train_results.append(train_result)
    #print('train_result \n', 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)

#print('X train \n',X_train)tes
# Cross validation results
fig = plt.figure()
fig.suptitle('Algorithm Comparison: Kfold results')
ax = fig.add_subplot(111)
plt.boxplot(kfold_results)
ax.set_xticklabels(names)
fig.set_size_inches(15,8)
plt.savefig('Algorithm Comparison: Kfold results.png')
plt.show()
```

```
LR: 0.002904 (0.001607) 0.002790 0.003005
LASSO: 0.003468 (0.001755) 0.003465 0.003264
EN: 0.003468 (0.001755) 0.003465 0.003264
KNN: 0.003578 (0.001997) 0.002264 0.003429
CART: 0.005657 (0.002704) 0.000000 0.005363
SVR: 0.003948 (0.002079) 0.003043 0.003537
MLP: 0.003011 (0.001679) 0.002818 0.002916
ABR: 0.003326 (0.002034) 0.002489 0.002935
GBR: 0.003329 (0.001748) 0.001262 0.003144
RFR: 0.003303 (0.001704) 0.000443 0.003112
ETR: 0.003521 (0.001964) 0.000000 0.003306
```



In [575]:

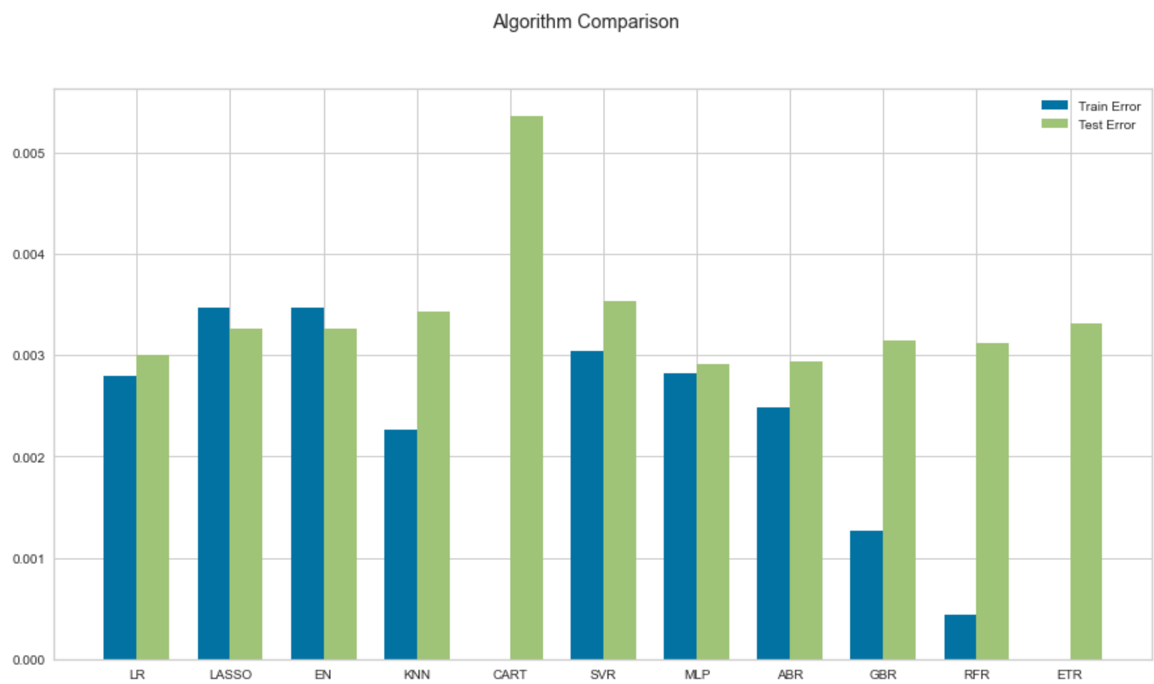
```
"""
Although the results of a couple of the models look good, we see that the linear regression and MLP seem
to perform best.Going back to the exploratory analysis, we saw a good correlation and linear relationship of the target variables
with the different lagged ETF variables.
"""
#This indicates a strong linear relationship between the dependent and independent variables.
```

Out[575]:

```
'\nAlthough the results of a couple of the models look good, we see that the linear regression and MLP seem \nto perform best.Going back to the exploratory analysis, we saw a good correlation and linear relationship of the target variables \nwith the different lagged ETF variables.\n'
```

In [576]:

```
# Errors of the test
# Training and test error
# Compare algorithms
fig = plt.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)
plt.bar(ind - width/2, train_results,  width=width, label='Train Error')
plt.bar(ind + width/2, test_results, width=width, label='Test Error')
fig.set_size_inches(15,8)
plt.legend()
ax.set_xticks(ind)
ax.set_xticklabels(names)
plt.savefig('Algorithm Comparison.png')
plt.show()
```



In [577]:

```
# Comment on the comparaison figure above

"""
Examining the training and test error, we still see a good performance from the
 linear models on the testing sets.
Some of the algorithms, such as the decision tree regressor (CART), overfit on t
he training data and produced
very high error on the test set. Ensemble models such as gradient boosting regre
ssion (GBR) has low bias but high variance,
it won't be a good model to predict the trend.
"""
```

Out[577]:

```
"\nExamining the training and test error, we still see a good perfor
mance from the linear models on the testing sets. \nSome of the algo
rithms, such as the decision tree regressor (CART), overfit on the t
raining data and produced \nvery high error on the test set. Ensembl
e models such as gradient boosting regression (GBR) has low bias but
high variance,\nit won't be a good model to predict the trend. \n"
```

In [579]:

```
"""

# Model Tuning and Grid Search

# Common Regression and Ensemble Grid Search
# 1. Grid search : LinearRegression
'''
fit_intercept : boolean, optional, default True
    whether to calculate the intercept for this model. If set
    to False, no intercept will be used in calculations
    (e.g. data is expected to be already centered).
'''
param_grid = {'fit_intercept': [True, False]}
model = LinearRegression()
kfold = KFold(n_splits=num_folds, random_state=None)
grid = GridSearchCV(estimator=model, param_grid=param_grid, scoring=scoring, cv=
kfold)
grid_result = grid.fit(X_train, Y_train)
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("Mean%f, stdev (%f) with: parameters %r" % (mean, stdev, param))
"""
```

```
Best: -0.002904 using {'fit_intercept': True}
Mean-0.002904, stdev (0.001607) with: parameters {'fit_intercept': T
rue}
Mean-0.002942, stdev (0.001627) with: parameters {'fit_intercept': F
alse}
```

In [591]:

```
# Grid search : MLPRegressor
'''
hidden_layer_sizes : tuple, length = n_layers - 2, default (100,)
The ith element represents the number of neurons in the ith
hidden layer.
'''
param_grid={'hidden_layer_sizes': [(20,), (50,), (20,20), (20, 30, 20)]}
model = MLPRegressor()
kfold = KFold(n_splits=num_folds, random_state=None)
grid = GridSearchCV(estimator=model, param_grid=param_grid, scoring=scoring, cv=
kfold)
grid_result = grid.fit(X_train, Y_train)
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("Mean %f \ stdev(%f) with parameters : %r" % (mean, stdev, param))
```

```
Best: -0.003425 using {'hidden_layer_sizes': (20, 30, 20)}
Mean -0.004039 \ stdev(0.002230) with parameters : {'hidden_layer_si
zes': (20,)}
Mean -0.003602 \ stdev(0.002392) with parameters : {'hidden_layer_si
zes': (50,)}
Mean -0.003719 \ stdev(0.002049) with parameters : {'hidden_layer_si
zes': (20, 20)}
Mean -0.003425 \ stdev(0.001788) with parameters : {'hidden_layer_si
zes': (20, 30, 20)}
```

In [592]:

```
# Finalize the model
# prepare model
model = MLPRegressor(hidden_layer_sizes= (20, 30, 20))
model.fit(X_train, Y_train)
```

Out[592]:

```
MLPRegressor(activation='relu', alpha=0.0001, batch_size='auto', bet
a_1=0.9,
             beta_2=0.999, early_stopping=False, epsilon=1e-08,
             hidden_layer_sizes=(20, 30, 20), learning_rate='constan
t',
             learning_rate_init=0.001, max_fun=15000, max_iter=200,
             momentum=0.9, n_iter_no_change=10, nesterovs_momentum=T
rue,
             power_t=0.5, random_state=None, shuffle=True, solver='a
dam',
             tol=0.0001, validation_fraction=0.1, verbose=False,
             warm_start=False)
```

In [596]:

```
# Results and comparison of Regression and MLP
# estimate accuracy on validation set
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

# prepare model
model = MLPRegressor(hidden_layer_sizes= (20, 30, 20))
model.fit(X_train, Y_train)
predictions = model.predict(X_test)
mse_MLP = mean_squared_error(Y_test, predictions)
r2_MLP = r2_score(Y_test, predictions)

# prepare model
model_2 = LinearRegression()
model_2.fit(X_train, Y_train)
predictions_2 = model_2.predict(X_test)

mse_OLS = mean_squared_error(Y_test, predictions_2)
r2_OLS = r2_score(Y_test, predictions_2)
print("MSE Regression = %f, MSE MLP = %f" % (mse_OLS, mse_MLP))
print("R2 Regression = %f, R2 MLP = %f" % (r2_OLS, r2_MLP))
```

```
MSE Regression = 0.003005, MSE MLP = 0.003026
R2 Regression = 0.077099, R2 MLP = 0.070655
```

In [597]:

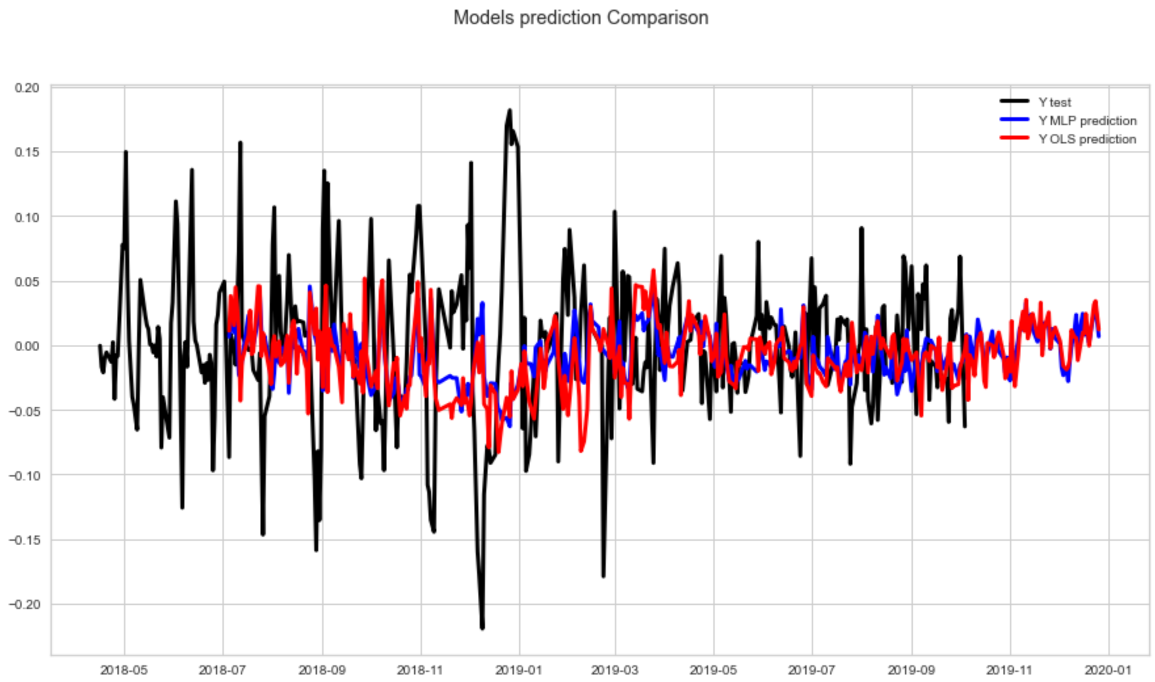
```
# Comment on the metrics obtained with the linear regression model
"""
The Regression linear R2 score (77%) is higher than the one of MLP, but still no
t perfect as it's lower than the
results obtained with the ridge, 90%.
Let us check the prediction shape of the validation set.
"""
```

Out[597]:

```
'\nThe Regression linear R2 score (77%) is higher than the one of ML
P. which is not satisfying and is lower than the results obtained wi
th the ridge, 90%.\nWe may need to considere another model in order
to improve the performance of our predictions.\n'
```

In [623]:

```
# Predictions
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)]

modelMLP = MLPRegressor(hidden_layer_sizes= (20, 30, 20))
modelOLS = LinearRegression()
model_MLP = modelMLP.fit(X_train, Y_train)
model_OLS = modelOLS.fit(X_train, Y_train)

Y_predMLP = pd.DataFrame(model_MLP.predict(X_test), index=X_test.index)
Y_predOLS = pd.DataFrame(model_OLS.predict(X_test), index=X_test.index)

fig = plt.figure()
fig.suptitle('Models prediction Comparison')
plt.plot(Y_test,label = 'Y test', color='black', linewidth=3)
plt.plot(Y_predMLP, label = 'Y MLP prediction', color='blue', linewidth=3)
plt.plot(Y_predOLS,label = 'Y OLS prediction', color='RED', linewidth=3)
ax = fig.add_subplot(111)
fig.set_size_inches(15,8)
plt.legend()
plt.show()

#plt.scatter(X_test, Y_test,  color='black')

plt.show()
```



In [624]:

```
# Comment on the results obtained above
# 3 - Final result and recommendation
"""
Overall, we can see that the linear model provides better results than the MLP m
odel.
Looking at other more sophisticated models seem not being the solution to the im
perfection seen in the performance
of the regression model.
Let us look at the strategy returns based on the OLS before to conclude this wor
k and make some recommendation.
"""
```

Out[624]:

```
"\nOverall, we can see that the linear model provides better results
even though the performance isn't perfect.\n"
```

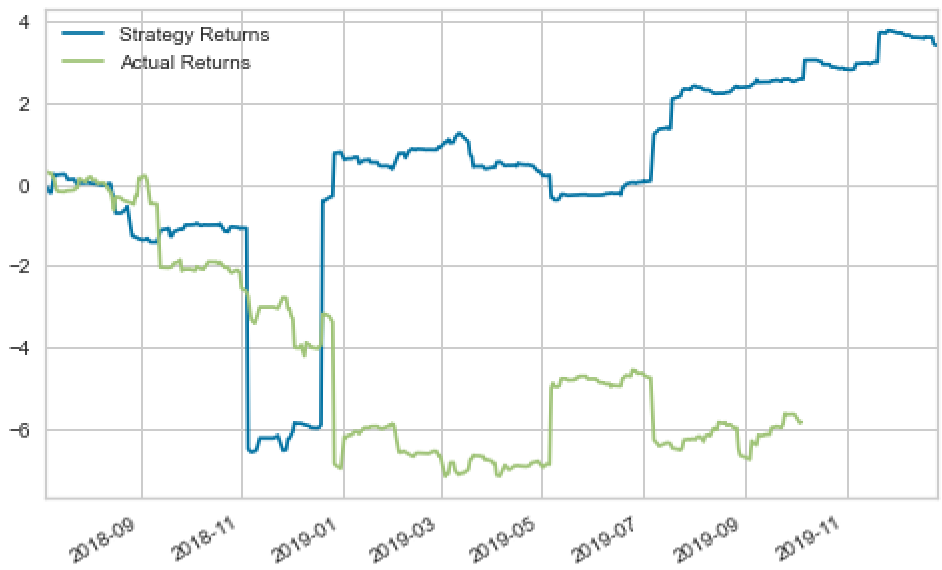
In [582]:

```
#Create column for Strategy Returns by multiplying the daily returns by the posi
tion that was held at close
#of business the previous day
backtestdata = pd.DataFrame(index=X_test.index)
print('X_test \n', X_test.tail())
print('Y_test \n', Y_test.tail())


#backtestdata = pd.DataFrame()
backtestdata['close_pred'] = predictions
backtestdata['close_actual'] = Y_test
backtestdata['Market Returns'] = X_test['ETF_DT'].pct_change()
backtestdata['Actual Returns'] = backtestdata['Market Returns'] * backtestdata[
'close_actual'].shift(1)
backtestdata['Strategy Returns'] = backtestdata['Market Returns'] * backtestdata
['close_pred'].shift(1)
#backtestdata=backtestdata.reset_index()
backtestdata.head()
backtestdata[['Strategy Returns','Actual Returns']].cumsum().hist()
backtestdata[['Strategy Returns','Actual Returns']].cumsum().plot()
```

```
X_test
              ETF_DT   ETF_3DT   ETF_6DT  ETF_12DT
2019-12-19 -0.068950  0.000394  0.023377  0.069285
2019-12-20 -0.013648 -0.006350  0.030012  0.034292
2019-12-23 -0.014804  0.069106  0.067722  0.156281
2019-12-24  0.066885  0.103806  0.040799  0.164798
2019-12-26  0.006773  0.058884 -0.002295  0.071208
Y_test
 2019-09-26    0.027445
2019-09-27    0.003896
2019-09-30    0.028209
2019-10-01    0.068668
2019-10-04   -0.062900
Name: Close_pred, dtype: float64
```

Out[582]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a22f8ef60>
```



In [ ]:

```
# Comment on the results

"""
Looking at the chart, we clearly see a dislocation between the two strategies af
ter few months of testing.
The predicted serie aligns with the actual data for the first few months of the
 test set. A point to note is that
the purpose of the model is to compute the next day’s return given the data obse
rved up to the present day,
and not to predict the stock price several days in the future given the current
 data.
Hence, a deviation from the actual data is expected as we move away from the beg
inning of the test set.
The model seems to perform well for the first few months, with deviation from th
e actual data increasing six
months after the beginning of the test set.
"""

"""
We can conclude that simple models—linear regression are promising modeling appr
oaches for stock price prediction
problems. This approach helps us deal with overfitting and underfitting, which a
re some of the key challenges in
predicting problems in finance. We should also note that we can use a wider set
 of indicators, such as P/E ratio,
trading volume, technical indicators,Relative Strength Indicator, Moving Averag
e, Volatility or news data, which might
lead to more efficient predictions. Overall, we created a supervised-regression
 that allows us to perform stock price
prediction using historical data and signal data and this work can be further ex
tended by including the indicators
above.
"""
```