Code

November 27, 2021

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[ ]:
```
import os
os.chdir("/content/drive/MyDrive/EE 475/Project")
!ls
```

```
Code.ipynb  Dataset.ipynb  datasets
```

[ ]:
```
import numpy as np
import pandas as pd
pd.options.mode.chained_assignment = None
import seaborn as sns
from datetime import date
import matplotlib.pyplot as plt
from scipy.ndimage.interpolation import shift
from yellowbrick.regressor import residuals_plot

from sklearn import svm
from sklearn import metrics
from sklearn import preprocessing
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import SGDRegressor
from sklearn.linear_model import Perceptron
from sklearn.preprocessing import PolynomialFeatures
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor

%matplotlib inline
# %matplotlib qt
# plt.style.use('ggplot')
```

```
%config InlineBackend.figure_format = 'retina'
```

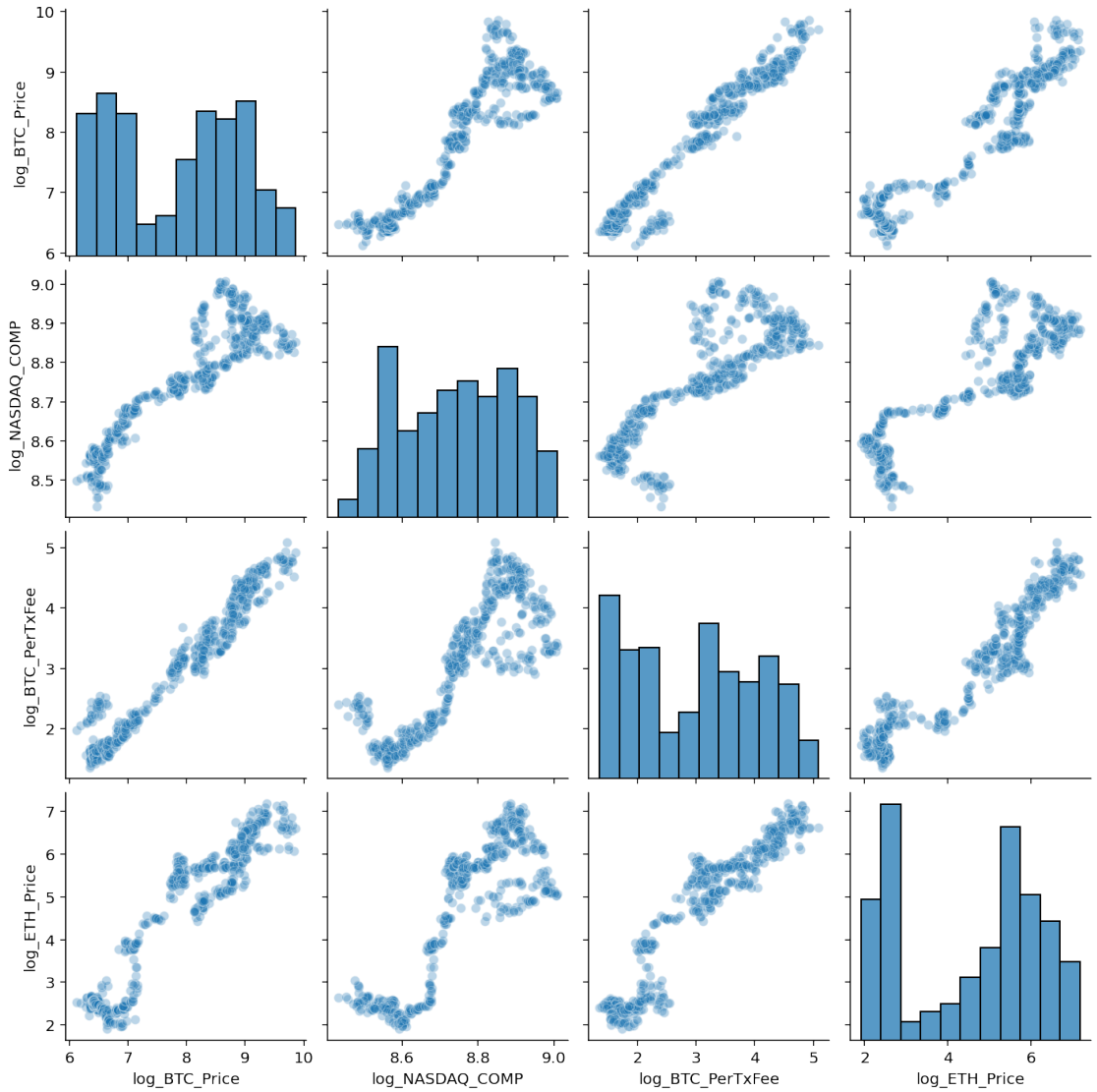
[ ]:
```
df = pd.read_pickle('./datasets/dataset.pkl')
```

[ ]:
```
df_key =␣
↪df[['log_BTC_Price','log_NASDAQ_COMP','log_BTC_PerTxFee','log_ETH_Price']]
df_key.corr().sort_values('log_BTC_Price')
```

[ ]:
```
                 log_BTC_Price  …  log_ETH_Price
log_NASDAQ_COMP       0.911698  …       0.857295
log_ETH_Price         0.949702  …       1.000000
log_BTC_PerTxFee      0.974659  …       0.954123
log_BTC_Price         1.000000  …       0.949702

[4 rows x 4 columns]
```

[ ]:
```
sns.pairplot(df_key,plot_kws={'alpha':0.3})
```

[ ]:
```
<seaborn.axisgrid.PairGrid at 0x7f40c476c750>
```



[ ]:
```
df_key.index = pd.to_datetime(df_key.index)
x = df_key[[col for col in df_key.columns if col not in ['log_BTC_Price']]]
y = df_key['log_BTC_Price']

# x_train = df_key[['log_BTC_PerTxFee', 'log_NASDAQ_COMP', 'log_ETH_Price']][:
 ↪-50]
# x_test = df_key[['log_BTC_PerTxFee', 'log_NASDAQ_COMP', 'log_ETH_Price']][-50:
 ↪]
# y_train = df_key['log_BTC_Price'][:-50]
# y_test = df_key['log_BTC_Price'][-50:]

# Split Dataset into Train Set and Test Set
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.1,␣
↪random_state = 1, shuffle = True)
y_train = pd.DataFrame(y_train)
y_test = pd.DataFrame(y_test)

# Sort Train Set and Test Set by Date
x_train.sort_index(inplace = True)
x_test.sort_index(inplace = True)
y_train.sort_index(inplace = True)
y_test.sort_index(inplace = True)

# Convert DataFrame to NumPy Array
x_np = np.array(x).astype(float)
x_train_np = np.array(x_train).astype(float)
x_test_np = np.array(x_test).astype(float)

y_np = y.values.ravel().astype(float)
y_train_np = y_train.values.ravel().astype(float)
y_test_np = y_test.values.ravel().astype(float)

print('x_train shape is: {}'.format(x_train.shape))
print('x_test shape is: {}'.format(x_test.shape))
print('y_train shape is: {}'.format(y_train.shape))
print('y_test shape is: {}'.format(y_test.shape))
```

```
x_train shape is: (522, 3)
x_test shape is: (59, 3)
y_train shape is: (522, 1)
y_test shape is: (59, 1)
```

[ ]:
```
def evaluate(y_test, y_test_pred):
    # Evaluate The Model
    R2 = metrics.r2_score(y_test, y_test_pred)
    MAE = metrics.mean_absolute_error(y_test, y_test_pred)
    MSE = metrics.mean_squared_error(y_test, y_test_pred)
    print('R^2 Score: {}'.format(R2))
    print('MAE Score: {}'.format(MAE))
    print('MSE Score: {}'.format(MSE))
```

[ ]:
```
def plot_prediction(x, y, y_pred, x_train, y_train, y_train_pred, x_test,␣
↪y_test, y_test_pred):
    # Prediction on Test Set
    plt.figure(figsize=(14, 5))
    plt.subplot(1, 2, 1)
    # plt.scatter(y_test.index, y_test, alpha = 0.8, label = 'y_test')
    plt.plot(y_test, alpha = 0.8, label = 'y_test')
```

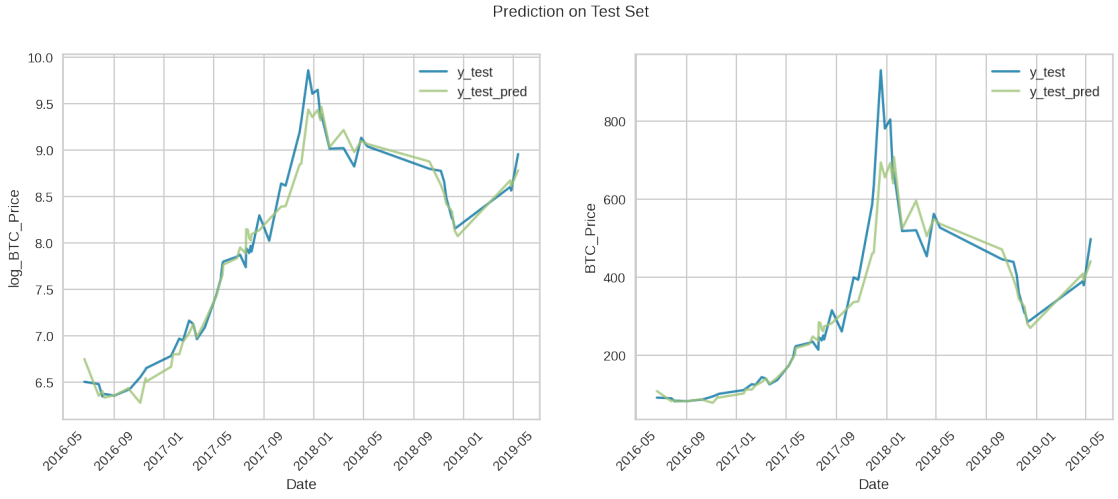
```
    # plt.scatter(y_test_pred.index, y_test_pred, alpha = 0.8, label =␣
↪'y_test_pred')
    plt.plot(y_test_pred, alpha = 0.8, label = 'y_test_pred')
    plt.xlabel('Date')
    plt.xticks(rotation=45)
    plt.ylabel('log_BTC_Price')
    plt.legend()

    plt.subplot(1, 2, 2)
    # plt.scatter(y_test.index, y_test, alpha = 0.8, label = 'y_test')
    plt.plot(np.power(2, y_test), alpha = 0.8, label = 'y_test')

    # plt.scatter(y_test_pred.index, y_test_pred, alpha = 0.8, label =␣
↪'y_test_pred')
    plt.plot(np.power(2, y_test_pred), alpha = 0.8, label = 'y_test_pred')

    plt.xlabel('Date')
    plt.xticks(rotation = 45)
    plt.ylabel('BTC_Price')
    plt.legend()

    plt.suptitle('Prediction on Test Set')
    plt.show()



    # Prediction on All Set
    plt.figure(figsize=(14, 5))
    plt.subplot(1, 2, 1)
    # plt.scatter(y.index, y, alpha = 0.8, label = 'y')
    plt.plot(y, alpha = 0.8, label = 'y')

    # plt.scatter(y_pred.index, y_pred, alpha = 0.8, label = 'y_pred')
    plt.plot(y_pred, alpha = 0.8, label = 'y_pred')
    plt.xlabel('Date')
    plt.xticks(rotation=45)
    plt.ylabel('log_BTC_Price')
    plt.legend()

    plt.subplot(1, 2, 2)
    # plt.scatter(y.index, y, alpha = 0.8, label = 'y')
    plt.plot(np.power(2, y), alpha = 0.8, label = 'y')

    # plt.scatter(y_pred.index, y_pred, alpha = 0.8, label = 'y_pred')
    plt.plot(np.power(2, y_pred), alpha = 0.8, label = 'y_pred')

    plt.xlabel('Date')
```

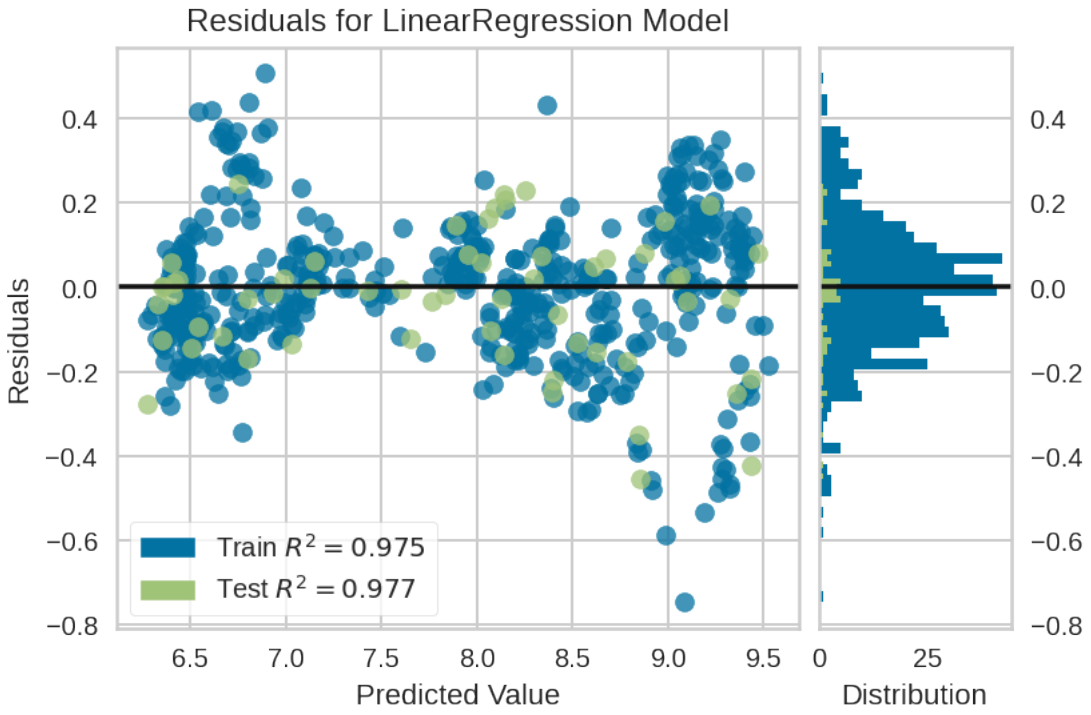
```
    plt.xticks(rotation = 45)
    plt.ylabel('BTC_Price')
    plt.legend()

    plt.suptitle('Prediction on All Set')
    plt.show()
```

Linear Regression

[ ]:
```
lr = LinearRegression()
lr.fit(x_train, y_train)

y_train_pred = lr.predict(x_train)
y_train_pred = pd.DataFrame(y_train_pred)
y_train_pred.index = y_train.index

y_test_pred = lr.predict(x_test)
y_test_pred = pd.DataFrame(y_test_pred)
y_test_pred.index = y_test.index

y_pred = lr.predict(x)
y_pred = pd.DataFrame(y_pred)
y_pred.index = y.index

evaluate(y_test, y_test_pred)
plot_prediction(x, y, y_pred, x_train, y_train, y_train_pred, x_test, y_test,␣
 ↪y_test_pred)
viz = residuals_plot(LinearRegression(), x_train_np, y_train_np, x_test_np,␣
 ↪y_test_np)
```

```
R^2 Score: 0.9769435871199582
MAE Score: 0.11637793776054353
MSE Score: 0.02439590450187253
```





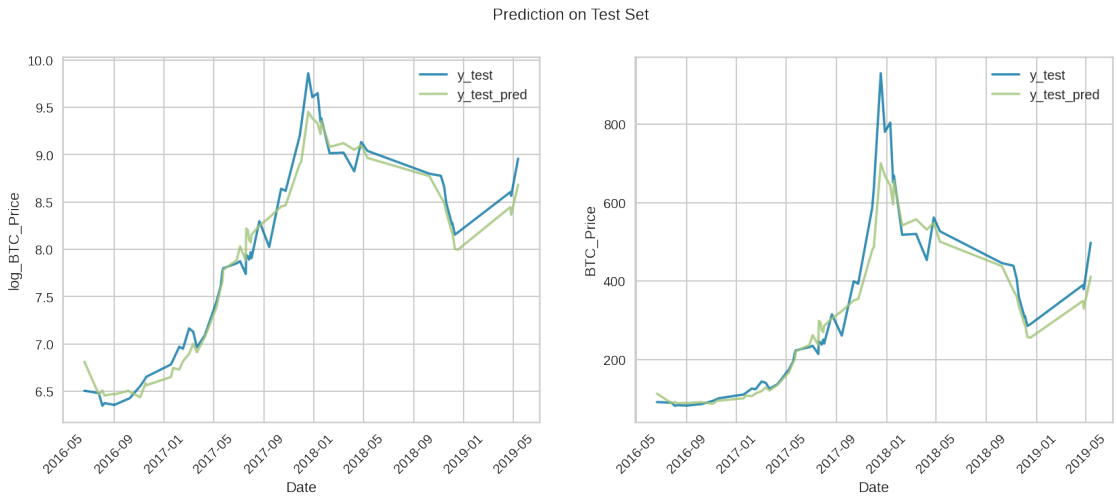
[ ]:
```
x_train_poly = PolynomialFeatures(interaction_only=True).fit_transform(x_train).
↪astype(float)
clf = Perceptron(fit_intercept=False, max_iter=10, tol=None, shuffle=False)
clf.fit(x_train_poly, y_train)
```

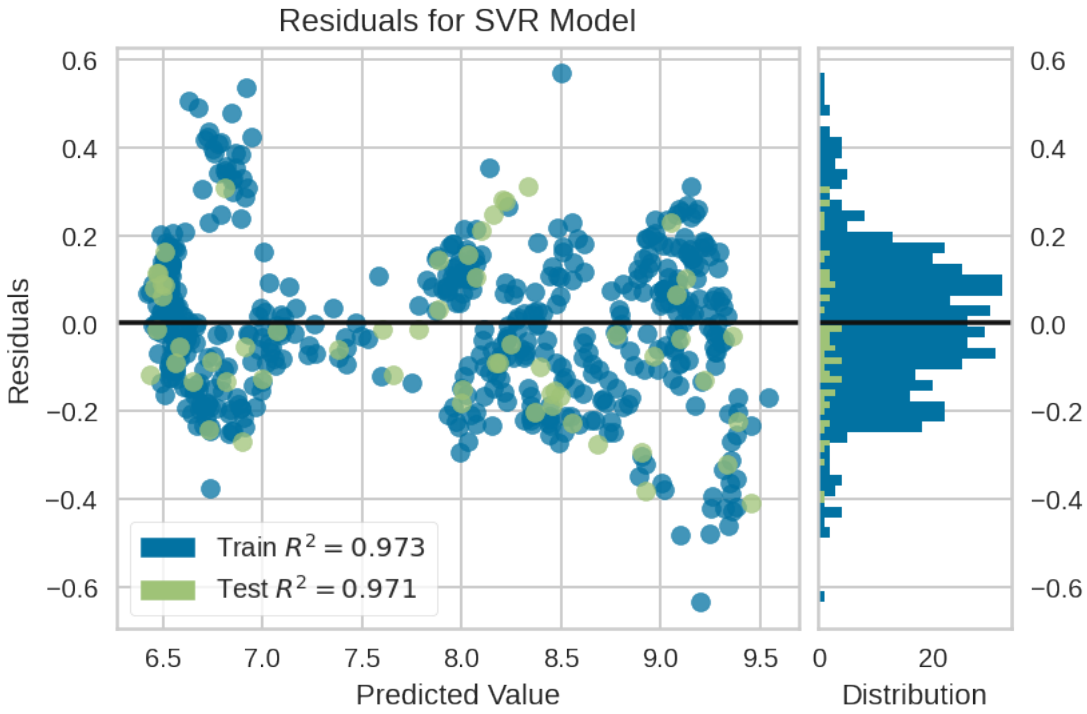
```
print(clf.score(x_train, y_train))
```

Epsilon-Support Vector Regression using RBF kernel

[ ]:
```
SVR_rbf = svm.SVR(kernel = 'rbf')
SVR_rbf.fit(x_train, y_train_np)

y_train_pred = SVR_rbf.predict(x_train)
y_train_pred = pd.DataFrame(y_train_pred)
y_train_pred.index = y_train.index

y_test_pred = SVR_rbf.predict(x_test)
y_test_pred = pd.DataFrame(y_test_pred)
y_test_pred.index = y_test.index

y_pred = SVR_rbf.predict(x)
y_pred = pd.DataFrame(y_pred)
y_pred.index = y.index

evaluate(y_test, y_test_pred)
plot_prediction(x, y, y_pred, x_train, y_train, y_train_pred, x_test, y_test,␣
↪y_test_pred)
viz = residuals_plot(svm.SVR(kernel='rbf'), x_train_np, y_train_np, x_test_np,␣
↪y_test_np)
```

```
R^2 Score: 0.9707137828896395
MAE Score: 0.14674298081504575
MSE Score: 0.030987637129968142
```

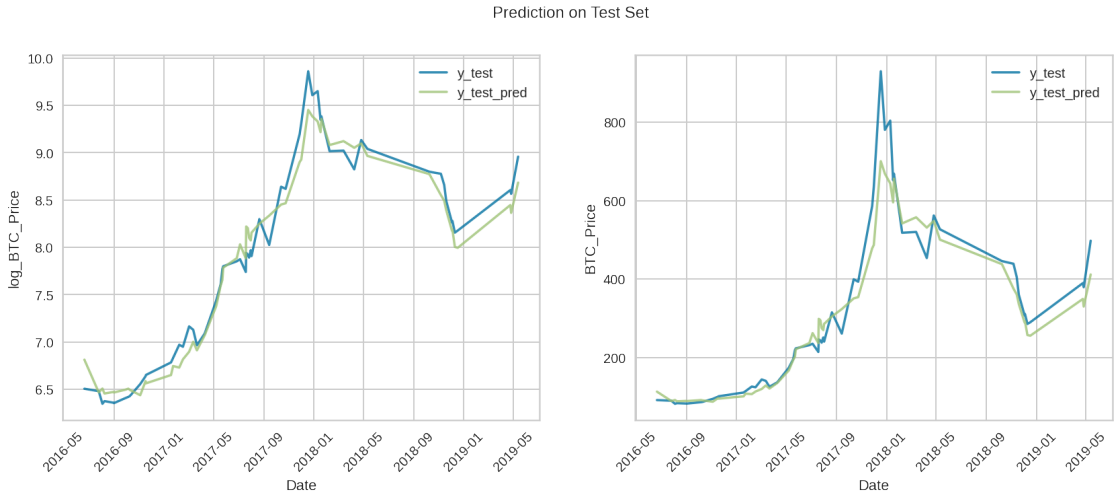





```
[ ]: SGD = SGDRegressor(max_iter = 1000, tol = 1e-3)
     SGD.fit(x_train, y_train_np)

     y_train_pred = SVR_rbf.predict(x_train)
     y_train_pred = pd.DataFrame(y_train_pred)
     y_train_pred.index = y_train.index
```

```
y_test_pred = SVR_rbf.predict(x_test)
y_test_pred = pd.DataFrame(y_test_pred)
y_test_pred.index = y_test.index

y_pred = SVR_rbf.predict(x)
y_pred = pd.DataFrame(y_pred)
y_pred.index = y.index

evaluate(y_test, y_test_pred)
plot_prediction(x, y, y_pred, x_train, y_train, y_train_pred, x_test, y_test,␣
↪y_test_pred)
viz = residuals_plot(SGDRegressor(max_iter = 1000, tol = 1e-3), x_train,␣
↪y_train.values.ravel(), x_test, y_test.values.ravel())
```

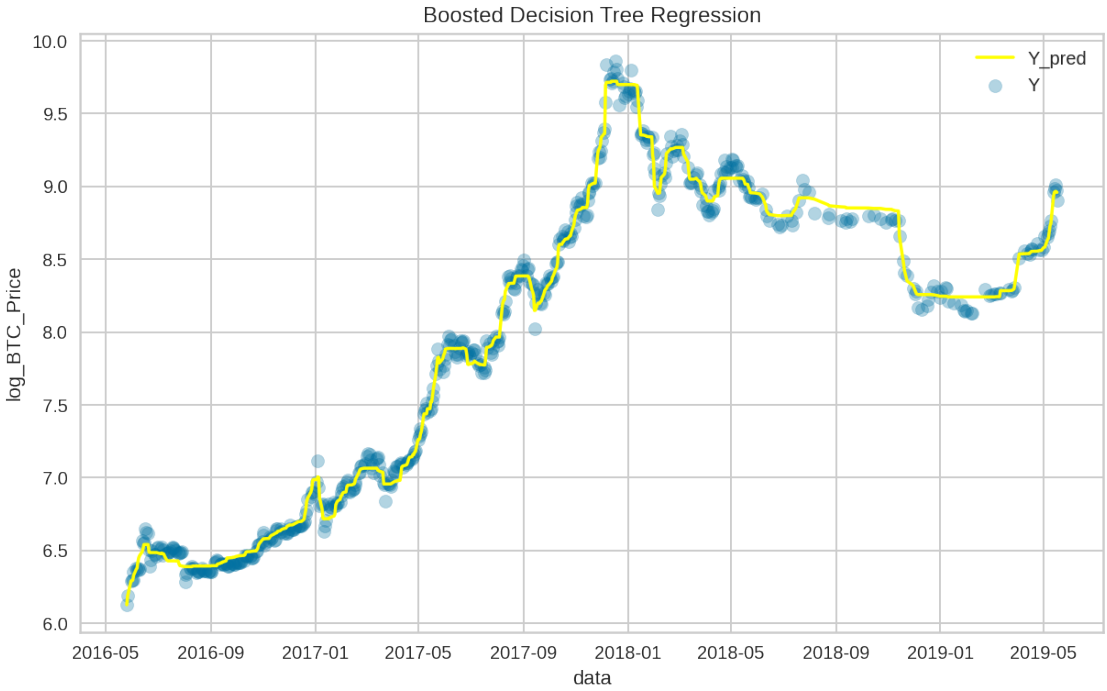
```
R^2 Score: 0.9707137828896395
MAE Score: 0.14674298081504575
MSE Score: 0.030987637129968142
```



Decision Tree Regressor

[ ]:
```
DecisionTree = AdaBoostRegressor(DecisionTreeRegressor(max_depth=5),␣
↪n_estimators=300, random_state=0)

X_train = np.arange(0, y_train_np.size)[:, np.newaxis]
```

```
Y_train = y_train_np.ravel()
X_test = np.arange(0, y_test_np.size)[:, np.newaxis]
Y_test = y_test_np.ravel()
X = np.arange(0, y_np.size)[:, np.newaxis]
Y = y_np.ravel()
DecisionTree.fit(X, Y)

Y_test_pred = DecisionTree.predict(X_test)
Y_test_pred = pd.DataFrame(Y_test_pred)
Y_test_pred.index = y_test.index

Y_train_pred = DecisionTree.predict(X_train)
Y_train_pred = pd.DataFrame(Y_train_pred)
Y_train_pred.index = y_train.index

Y_pred = DecisionTree.predict(X)
Y_pred = pd.DataFrame(Y_pred)
Y_pred.index = y.index

plt.figure(figsize=(10, 6))
plt.scatter(x.index, Y, alpha = 0.3, label = "Y")
plt.plot(Y_pred, c = 'yellow', label = "Y_pred")
plt.xlabel("data")
plt.ylabel("log_BTC_Price")
plt.title("Boosted Decision Tree Regression")
plt.legend()
plt.show()

# cross_val_score(DecisionTree, X, Y, cv=10)
# print(DecisionTree.score(Y_test, Y_test_pred))
evaluate(Y, Y_pred)
plot_prediction(x, y, Y_pred, x_train, y_train, Y_train_pred, x_test, y_test,␣
 ↪Y_test_pred)
```




```
R^2 Score: 0.9978021042546645
MAE Score: 0.039494577844040125
MSE Score: 0.00249902675336626
```


Prediction on All Set
y
y_pred
log_BTC_Price
BTC_Price
Date
Date