EE475 Project - Cryptocurrency Price Prediction

December 6, 2021

Project Presentation Youtube Link: https://youtu.be/CiKeDgDzUzQ

Introduction

With the primary motivation to define a trustworthy exchange currency used as peer-to-peer electronic cash systems, the cryptocurrency attracted criticism from the economists and supervision regulations but attracted worldwide investors with rapidly growing market capitalization. The cryptocurrency consisting of binary data has the most attractive property of decentralization to secure the transaction records and verify the transfer of ownership, without any physical form.

With the prices as well as the market size of cryptocurrencies increasing substantially, people are considering them as investment options such as Bitcoin which leads to the study in providing good predictions considering its trading markets. By extracting the structure and dimensional features from different data modeling methods, machine learning techniques are able to deliver acceptable timely prediction results with optimized investment strategy.

Dataset

We collect the below dataset from Nasdaq and further combine them together based on `Date`.

1. Bitcoin Market Price USD
2. ETH/USD Exchange Rate
3. Bitcoin Total Transaction Fees USD
4. Bitcoin Cost Per Transaction
5. Bitcoin Number of Transactions
6. Bitcoin Number of Transaction per Block
7. Bitcoin Average Block Size
8. Bitcoin Number of Unique Bitcoin Addresses Used
9. Bitcoin Hash Rate
10. Bitcoin Difficulty
11. NASDAQ Composite (COMP)

We also apply different log function to better reveal the internal relationship. Finally choose factors of `log_BTC_PerTxFee`, `log_ETH_Price`, `log_BTC_HashRate` for their high correlation score with `log_BTC_Price`.

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

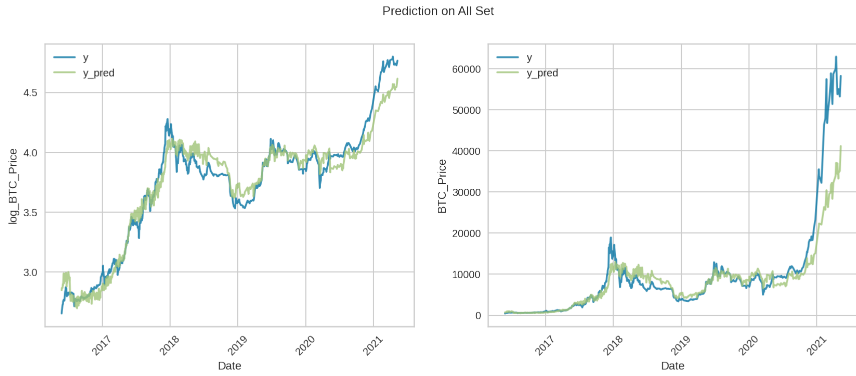
```
                  log_BTC_Price  ...  log_BTC_HashRate
log_BTC_HashRate       0.890897  ...          1.000000
log_ETH_Price          0.924772  ...          0.741251
log_BTC_PerTxFee       0.957572  ...          0.795576
log_BTC_Price          1.000000  ...          0.890897

[4 rows x 4 columns]
```

Models

We apply many models trying to predict Bitcoin Price, which includes `Linear Regression`, `Polynominal Features + Linear Regression`, `Stochastic Gradient Descent (SGD)`, `Support Vector Machine using RBF kernel`, `Decision Tree`, `Random Forest`, `Neural Network`, `Long Short-Term Memory (LSTM)`.

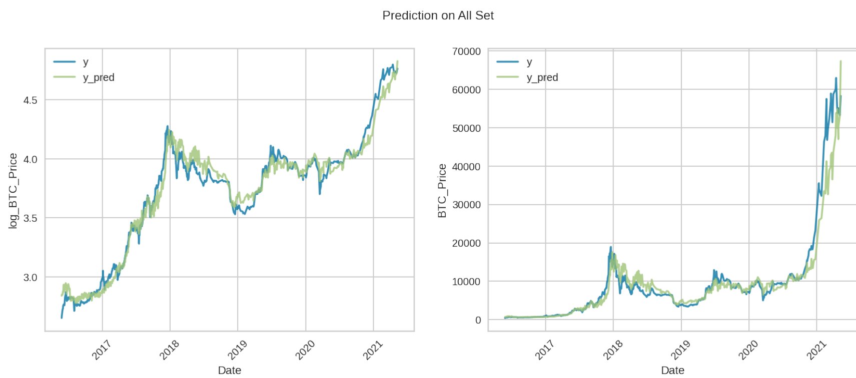
Linear Regression

```
[ ]: name = 'LR'
```



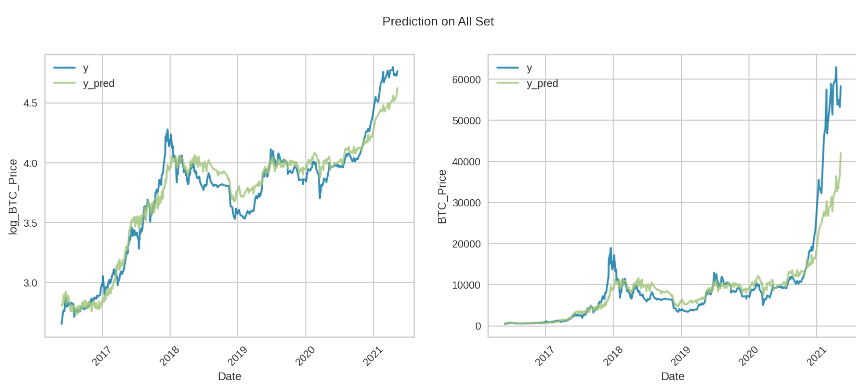
Polynominal Features + Linear Regression

```
[ ]: name = 'Poly+LR'
```



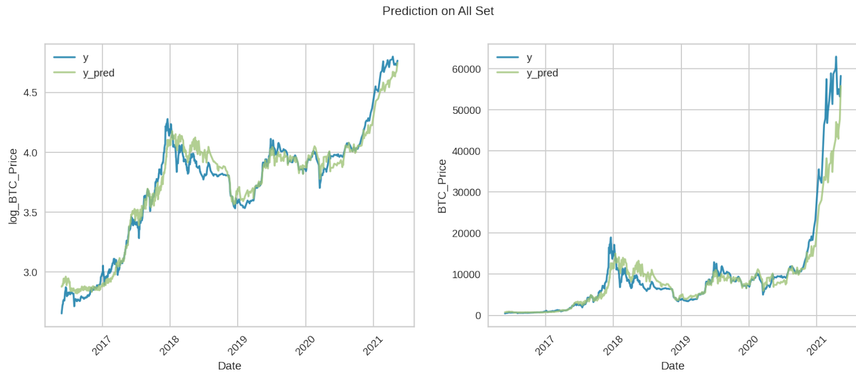
Stochastic Gradient Descent (SGD)

```
[ ]: name = 'SGD'
```



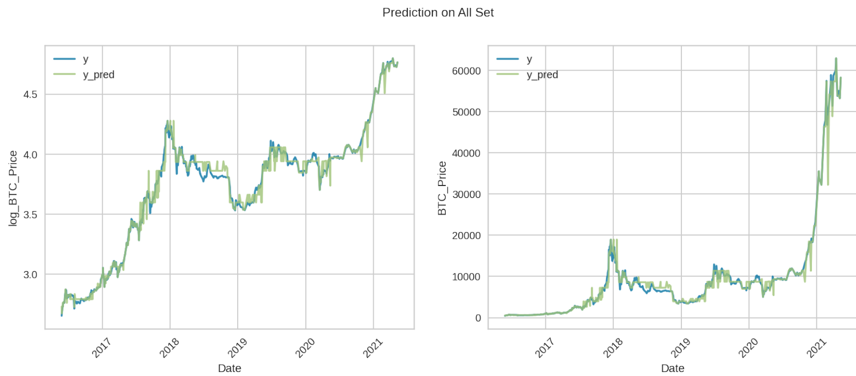
Support Vector Machine using RBF kernel

[ ]:
```
name = 'SVM+RBF'
```



Decision Tree

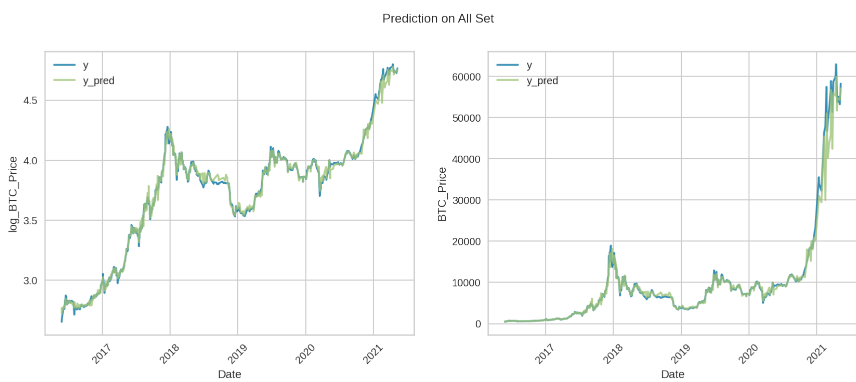
[ ]:
```
name = 'Decision Tree'
```



Random Forest

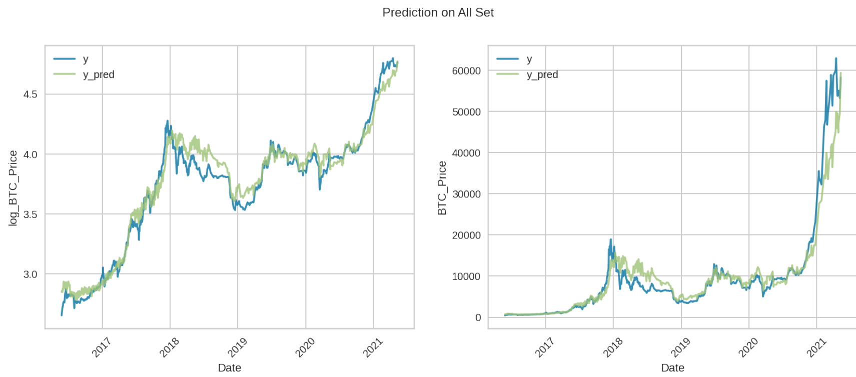
[ ]:
```
name = 'Random Forest'
```

```
[0.21507476 0.08088493 0.70404031]
['log_BTC_PerTxFee' 'log_ETH_Price' 'log_BTC_HashRate']
```



Neural Network

[ ]:
```
name = 'NN'
```



LSTM

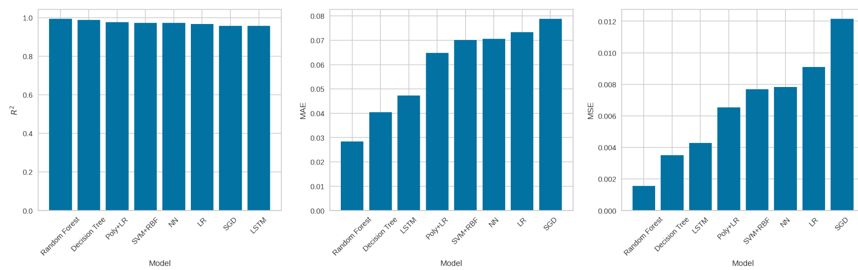
[ ]:
```
name = 'LSTM'
```

Result

Finally, we compare the performace of each model based on R^2, MAE and MSE.

[ ]:
```
plt.figure(figsize = (20, 5))

plt.subplot(1, 3, 1)
model, R2 = zip(*sorted(R2s.items(), key = lambda k : k[1], reverse = True))
plt.bar(model, R2)
```

```
plt.xlabel('Model')
plt.ylabel(r'$R^2$')
plt.xticks(rotation = 45)

plt.subplot(1, 3, 2)
model, MAE = zip(*sorted(MAEs.items(), key = lambda k : k[1]))
plt.bar(model, MAE)
plt.xlabel('Model')
plt.ylabel('MAE')
plt.xticks(rotation = 45)

plt.subplot(1, 3, 3)
model, MSE = zip(*sorted(MSEs.items(), key = lambda k : k[1]))
plt.bar(model, MSE)
plt.xlabel('Model')
plt.ylabel('MSE')
plt.xticks(rotation = 45)
plt.show()
```



Conclusion

During analyzing the original dataset, we find out that transforming the original data by applying log function can better reveal the internal correlationship between different factors and the base of 10 achieves the best performance among 2, 10 and e. After evaluating by three metrics: R^2, MAE and MSE on all the previous models, we can conclude that Random Forest is the best model for the Bitcoin Price Prediction. Besides, most Nonlinear Models achieves better performace than Linear Models, which may caused by the complexity of Bitcoin Price variety. Most factors related to Bitcoin is not ideally linear to the Bitcoin Price.

Because of the inflation brought by the policy and manufacture conditions during pandemic time, the prices of cryptocurrencies grown exponentially. More factors like google search trend and social media sentiment in twitter and facebook can be taken into consideration, which can better illustrate the external influence on human when trading Bitcoin. Apart from that, more advanced technologies like Deep Learning, RNN can be adopted because they are more effective in analyzing large Bitcoin Prices Dataset.

Appendix

Dataset

[ ]:
```
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.pipeline import 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.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.neural_network import MLPRegressor

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

Define Auxiliary Functions

[ ]:
```
def preprocess_df(df):
    df.sort_values(by='Date', inplace=True)
    df.set_index('Date', inplace=True)
    return df

def log(data):
    return np.log10(data)
```

Combine Dataset

[ ]:
```
# Bitcoin Market Price USD (https://data.nasdaq.com/data/BCHAIN/MKPRU)
df_BCHAIN_MKPRU = preprocess_df(pd.read_csv('./datasets/BCHAIN-MKPRU.csv'))
df_BCHAIN_MKPRU = df_BCHAIN_MKPRU.rename(columns = {'Value' : 'BTC_Price'})
df_BCHAIN_MKPRU.drop(df_BCHAIN_MKPRU[df_BCHAIN_MKPRU['BTC_Price'] == 0].index, inplace = True)

# ETH/USD Exchange Rate (https://data.nasdaq.com/data/GDAX/ETH_USD)
df_GDAX_ETH_USD = preprocess_df(pd.read_csv('./datasets/GDAX-ETH_USD.csv'))
df_GDAX_ETH_USD = df_GDAX_ETH_USD.loc[:, ['Open', 'Volume']].rename(columns = {'Open' :␣
↪'ETH_Price','Volume' : 'ETH_Vol'})

# Bitcoin Total Transaction Fees USD (https://data.nasdaq.com/data/BCHAIN/TRFUS)
df_BCHAIN_TRFUS = preprocess_df(pd.read_csv('./datasets/BCHAIN-TRFUS.csv'))
```

```
df_BCHAIN_TRFUS = df_BCHAIN_TRFUS.rename(columns = {'Value' : 'BTC_TotalTxFee'})
df_BCHAIN_TRFUS.drop(df_BCHAIN_TRFUS[df_BCHAIN_TRFUS['BTC_TotalTxFee'] == 0].index, inplace = True)

# Bitcoin Cost Per Transaction (https://data.nasdaq.com/data/BCHAIN/CPTRA)
df_BCHAIN_CPTRA = preprocess_df(pd.read_csv('./datasets/BCHAIN-CPTRA.csv'))
df_BCHAIN_CPTRA = df_BCHAIN_CPTRA.rename(columns = {'Value' : 'BTC_PerTxFee'})

# Bitcoin Number of Transactions (https://data.nasdaq.com/data/BCHAIN/NTRAN)
df_BCHAIN_NTRAN = preprocess_df(pd.read_csv('./datasets/BCHAIN-NTRAN.csv'))
df_BCHAIN_NTRAN = df_BCHAIN_NTRAN.rename(columns = {'Value' : 'BTC_TxNum'})

# Bitcoin Number of Transaction per Block (https://data.nasdaq.com/data/BCHAIN/NTRBL)
df_BCHAIN_NTRBL = preprocess_df(pd.read_csv('./datasets/BCHAIN-NTRBL.csv'))
df_BCHAIN_NTRBL = df_BCHAIN_NTRBL.rename(columns = {'Value' : 'BTC_TxNumPerBlk'})

# Bitcoin Average Block Size (https://data.nasdaq.com/data/BCHAIN/AVBLS)
df_BCHAIN_AVBLS = preprocess_df(pd.read_csv('./datasets/BCHAIN-AVBLS.csv'))
df_BCHAIN_AVBLS = df_BCHAIN_AVBLS.rename(columns = {'Value' : 'BTC_AvgBlkSize'})

# Bitcoin Number of Unique Bitcoin Addresses Used (https://data.nasdaq.com/data/BCHAIN/NADDU)
df_BCHAIN_NADDU = preprocess_df(pd.read_csv('./datasets/BCHAIN-NADDU.csv'))
df_BCHAIN_NADDU = df_BCHAIN_NADDU.rename(columns = {'Value' : 'BTC_UsedAddrNum'})

# Bitcoin Hash Rate (https://data.nasdaq.com/data/BCHAIN/HRATE)
df_BCHAIN_HRATE = preprocess_df(pd.read_csv('./datasets/BCHAIN-HRATE.csv'))
df_BCHAIN_HRATE = df_BCHAIN_HRATE.rename(columns = {'Value' : 'BTC_HashRate'})

# Bitcoin Difficulty (https://data.nasdaq.com/data/BCHAIN/DIFF)
df_BCHAIN_DIFF = preprocess_df(pd.read_csv('./datasets/BCHAIN-DIFF.csv'))
df_BCHAIN_DIFF = df_BCHAIN_DIFF.rename(columns = {'Value' : 'BTC_DD'})

# NASDAQ Composite (COMP) (https://data.nasdaq.com/data/NASDAQOMX/COMP)
df_NASDAQOMX_COMP = pd.read_csv('./datasets/NASDAQOMX-COMP.csv')
df_NASDAQOMX_COMP = df_NASDAQOMX_COMP.rename(columns={'Trade Date' : 'Date', 'Index Value' :␣
↪'NASDAQ_COMP'})
df_NASDAQOMX_COMP = df_NASDAQOMX_COMP[['Date', 'NASDAQ_COMP']]
df_NASDAQOMX_COMP = preprocess_df(df_NASDAQOMX_COMP)

df = df_BCHAIN_MKPRU.merge(df_GDAX_ETH_USD, on = 'Date')
df = df.merge(df_BCHAIN_TRFUS, on = 'Date')
df = df.merge(df_BCHAIN_CPTRA, on = 'Date')
df = df.merge(df_BCHAIN_NTRAN, on = 'Date')
df = df.merge(df_BCHAIN_NTRBL, on = 'Date')
df = df.merge(df_BCHAIN_AVBLS, on = 'Date')
df = df.merge(df_BCHAIN_NADDU, on = 'Date')
df = df.merge(df_BCHAIN_HRATE, on = 'Date')
df = df.merge(df_BCHAIN_DIFF, on = 'Date')
df = df.merge(df_NASDAQOMX_COMP, on = 'Date')
df
```

[ ]:

```
            BTC_Price  ETH_Price  …        BTC_DD  NASDAQ_COMP
Date                              …
2016-05-26     448.70      12.61  …  1.993121e+11      4901.77
2016-05-27     470.29      12.47  …  1.993121e+11      4933.50
2016-05-31     531.15      12.86  …  1.993121e+11      4948.05
2016-06-01     531.97      14.01  …  1.993121e+11      4952.25
2016-06-02     539.99      14.00  …  1.993121e+11      4971.36
…                 …          …  …            …            …
2021-04-22   53808.80    2332.18  …  2.358198e+13     13818.41
```

```
2021-04-27   54056.64    2322.43  …  2.358198e+13     14090.22
2021-04-28   55071.46    2533.99  …  2.358198e+13     14051.03
2021-05-05   53241.72    3261.40  …  2.060885e+13     13582.42
2021-05-10   58280.73    3917.26  …  2.060885e+13     13401.86

[771 rows x 12 columns]
```

[ ]:
```python
# add log columns
df['log_BTC_Price'] = df['BTC_Price'].apply(log)
df['log_ETH_Price'] = df['ETH_Price'].apply(log)
df['log_ETH_Vol'] = df['ETH_Vol'].apply(log)
df['log_BTC_TotalTxFee'] = df['BTC_TotalTxFee'].apply(log)
df['log_BTC_PerTxFee'] = df['BTC_PerTxFee'].apply(log)
df['log_BTC_TxNum'] = df['BTC_TxNum'].apply(log)
df['log_BTC_TxNumPerBlk'] = df['BTC_TxNumPerBlk'].apply(log)
df['log_BTC_AvgBlkSize'] = df['BTC_AvgBlkSize'].apply(log)
df['log_BTC_UsedAddrNum'] = df['BTC_UsedAddrNum'].apply(log)
df['log_BTC_HashRate'] = df['BTC_HashRate'].apply(log)
df['log_BTC_DD'] = df['BTC_DD'].apply(log)
df['log_NASDAQ_COMP'] = df['NASDAQ_COMP'].apply(log)
```

[ ]:
```python
df = df[['BTC_Price','log_BTC_Price',
         'ETH_Price', 'log_ETH_Price',
         'ETH_Vol', 'log_ETH_Vol',
         'BTC_TotalTxFee', 'log_BTC_TotalTxFee',
         'BTC_PerTxFee', 'log_BTC_PerTxFee',
         'BTC_TxNum', 'log_BTC_TxNum',
         'BTC_TxNumPerBlk', 'log_BTC_TxNumPerBlk',
         'BTC_AvgBlkSize', 'log_BTC_AvgBlkSize',
         'BTC_UsedAddrNum', 'log_BTC_UsedAddrNum',
         'BTC_HashRate', 'log_BTC_HashRate',
         'BTC_DD', 'log_BTC_DD',
         'NASDAQ_COMP', 'log_NASDAQ_COMP'
         ]]
df.corr().sort_values('log_BTC_Price')
```

[ ]:

	BTC_Price	log_BTC_Price	…	NASDAQ_COMP	log_NASDAQ_COMP
log_BTC_TxNumPerBlk	0.210567	0.262373	…	0.362489	0.373202
log_BTC_TxNum	0.216261	0.284791	…	0.328246	0.347040
BTC_TxNumPerBlk	0.229741	0.307184	…	0.391879	0.408571
BTC_TxNum	0.229301	0.318581	…	0.336355	0.360548
ETH_Vol	0.272817	0.331090	…	0.226396	0.243881
BTC_TotalTxFee	0.634433	0.497964	…	0.370935	0.349032
log_ETH_Vol	0.336453	0.518246	…	0.366140	0.413989
log_BTC_AvgBlkSize	0.505034	0.592179	…	0.679024	0.692695
BTC_AvgBlkSize	0.543207	0.619754	…	0.723628	0.733400
BTC_UsedAddrNum	0.601288	0.637918	…	0.577797	0.583898
log_BTC_UsedAddrNum	0.568860	0.643835	…	0.585320	0.600219
BTC_DD	0.690710	0.703096	…	0.928082	0.915264
ETH_Price	0.875747	0.705947	…	0.637080	0.609242
BTC_HashRate	0.699041	0.710048	…	0.929709	0.917531
BTC_Price	1.000000	0.747872	…	0.818652	0.769146
log_BTC_TotalTxFee	0.621191	0.769759	…	0.567689	0.601105
NASDAQ_COMP	0.818652	0.823709	…	1.000000	0.986054
BTC_PerTxFee	0.867972	0.832515	…	0.655264	0.662575
log_NASDAQ_COMP	0.769146	0.881054	…	0.986054	1.000000
log_BTC_DD	0.600291	0.885081	…	0.861362	0.922227
log_BTC_HashRate	0.603033	0.890897	…	0.859766	0.921889
log_ETH_Price	0.624700	0.924772	…	0.665410	0.734219

```
log_BTC_PerTxFee      0.672420      0.957572  …     0.676373        0.744960
log_BTC_Price         0.747872      1.000000  …     0.823709        0.881054

[24 rows x 24 columns]
```

Save Dataset

```
[ ]: df = df[np.isfinite(df).all(1)]
     df.to_pickle('./datasets/dataset_latest.pkl')
```

Code

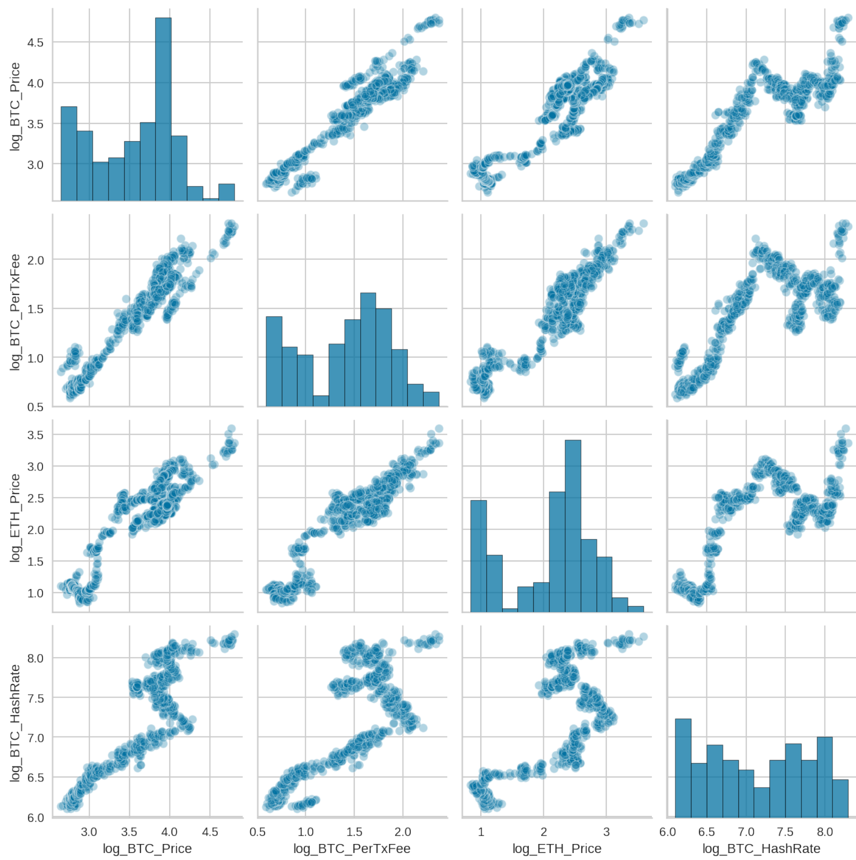
Import Dataset

```
[ ]: df = pd.read_pickle('./datasets/dataset_latest.pkl')
     df.corr(method='spearman').sort_values('log_BTC_Price')[-10:]
```

```
                  BTC_Price  log_BTC_Price  ...  NASDAQ_COMP  log_NASDAQ_COMP
log_BTC_HashRate   0.864497       0.864497  ...     0.979512         0.979512
BTC_HashRate       0.864497       0.864497  ...     0.979512         0.979512
BTC_DD             0.866481       0.866481  ...     0.979628         0.979628
log_BTC_DD         0.866484       0.866484  ...     0.979632         0.979632
log_NASDAQ_COMP    0.888968       0.888968  ...     1.000000         1.000000
NASDAQ_COMP        0.888968       0.888968  ...     1.000000         1.000000
log_BTC_PerTxFee   0.918578       0.918578  ...     0.765783         0.765783
BTC_PerTxFee       0.918578       0.918578  ...     0.765783         0.765783
log_BTC_Price      1.000000       1.000000  ...     0.888968         0.888968
BTC_Price          1.000000       1.000000  ...     0.888968         0.888968

[10 rows x 24 columns]
```

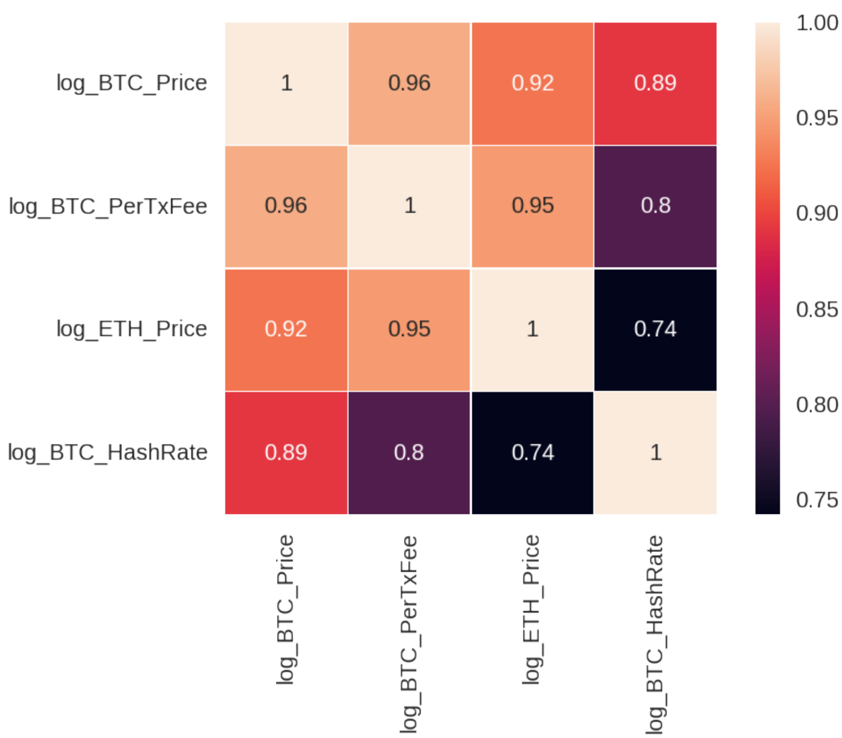
```
[ ]: df_key = df[['log_BTC_Price', 'log_BTC_PerTxFee', 'log_ETH_Price', 'log_BTC_HashRate']]
     sns.pairplot(df_key, plot_kws = {'alpha': 0.3})
```

```
<seaborn.axisgrid.PairGrid at 0x7f8efba43490>
```




```
[ ]: sns.heatmap(df_key.corr(), linewidths = 0.1, vmax = 1.0, square = True, linecolor = 'white', annot =
     True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff0cc147810>
```



Split Dataset

[ ]:
```
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, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.1, random_state = 1, shuffle␣
 ↪= False)
X_train = pd.DataFrame(y_train[:-1])
X_train = np.reshape(X_train, (len(X_train), 1))
Y_train = y_train[1:]

X_test = pd.DataFrame(y_test[:-1])
X_test = np.reshape(X_test, (len(X_test), 1))
Y_test = y_test[1:]

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, 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)
```

Define Auxiliary Functions

[ ]:
```
def evaluate(model, name, x, y):
    y_pred = model.predict(x)
    y_pred = pd.DataFrame(y_pred)
    y_pred.index = y.index

    # Evaluate The Model
    R2 = metrics.r2_score(y, y_pred)
    MAE = metrics.mean_absolute_error(y, y_pred)
    MSE = metrics.mean_squared_error(y, y_pred)

    R2s[name] = R2
    MAEs[name] = MAE
    MSEs[name] = MSE
    print('R^2 Score: {}'.format(R2))
    print('MAE Score: {}'.format(MAE))
    print('MSE Score: {}'.format(MSE))
```

```
[ ]: def plot_prediction(model, x, y, x_train, y_train, x_test, y_test):
         y_train_pred = model.predict(x_train)
         y_train_pred = pd.DataFrame(y_train_pred)
         y_train_pred.index = y_train.index

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

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

         # 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, np.power(10, y_test), alpha = 0.8, label = 'y_test')
         plt.plot(np.power(10, y_test), alpha = 0.8, label = 'y_test')

         # plt.scatter(y_test_pred.index, np.power(10, y_test_pred), alpha = 0.8, label = 'y_test_pred')
         plt.plot(np.power(10, 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(10, y), alpha = 0.8, label = 'y')
```

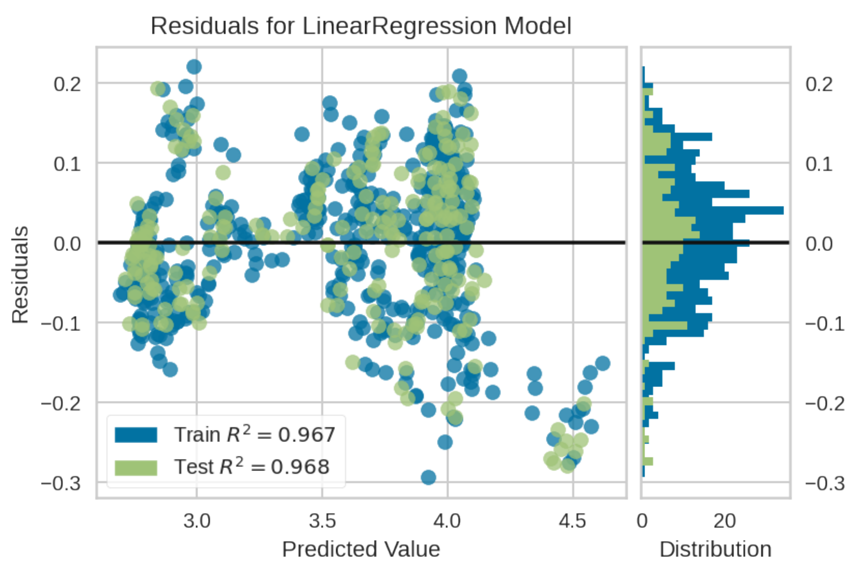
```
    # plt.scatter(y_pred.index, np.power(10, y_pred), alpha = 0.8, label = 'y_pred')
    plt.plot(np.power(10, y_pred), alpha = 0.8, label = 'y_pred')

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

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

[ ]:
```
R2s = {}
MAEs = {}
MSEs = {}
```

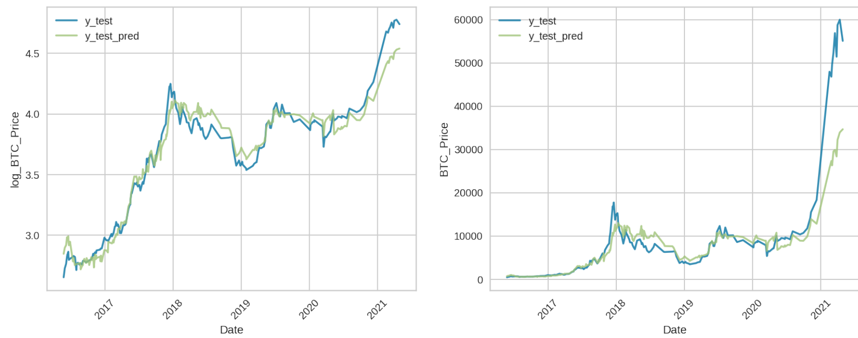
Models

Linear Regression

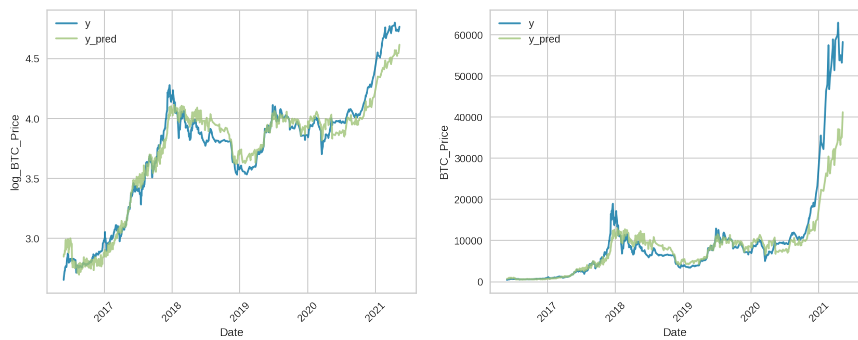
[ ]:
```
name = 'LR'
lr = Pipeline([
               ("std_scaler", StandardScaler()),
               ("lin_reg", LinearRegression())
               ])

viz = residuals_plot(lr, x_train, y_train_np, x_test, y_test_np)

lr.fit(x_train, y_train)
evaluate(lr, name, x_test, y_test)
plot_prediction(lr, x, y, x_train, y_train, x_test, y_test)
```



Prediction on Test Set

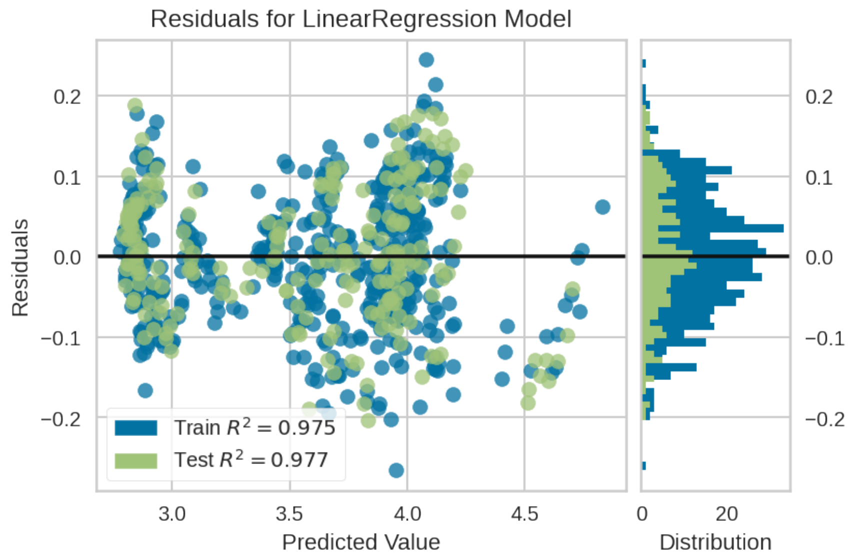


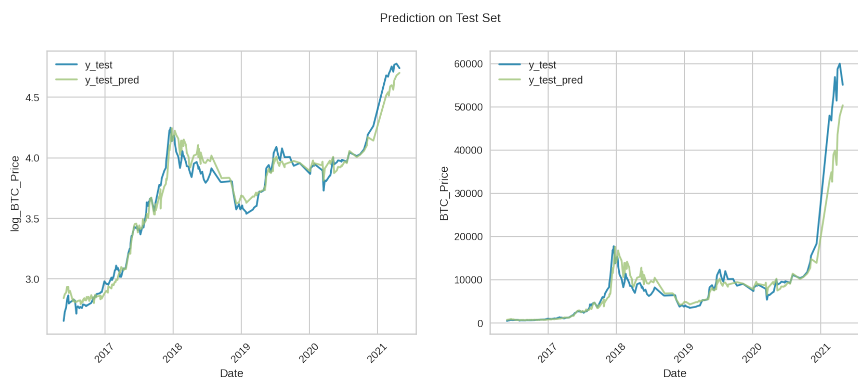
Prediction on All Set

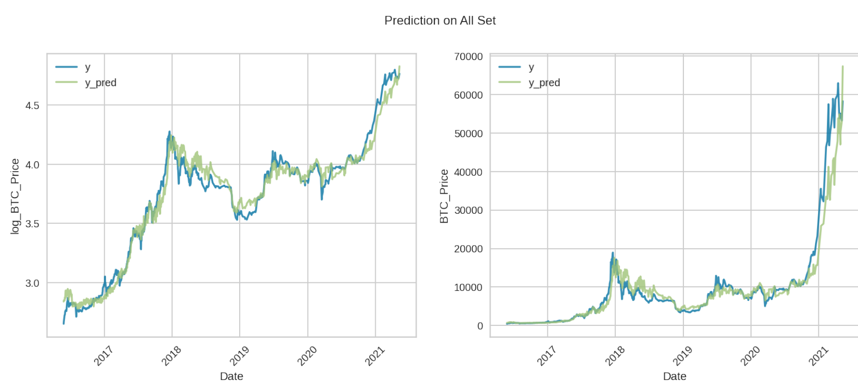


Polynominal Features + Linear Regression

```
[ ]: name = 'Poly+LR'
poly = Pipeline([
                 ("poly", PolynomialFeatures(degree = 2)),
                 ("std_scaler", StandardScaler()),
                 ("lin_reg", LinearRegression())
                 ])
viz = residuals_plot(poly, x_train, y_train_np, x_test, y_test_np)

poly.fit(x_train, y_train)
evaluate(poly, name, x_test, y_test)
plot_prediction(poly, x, y, x_train, y_train, x_test, y_test)
```

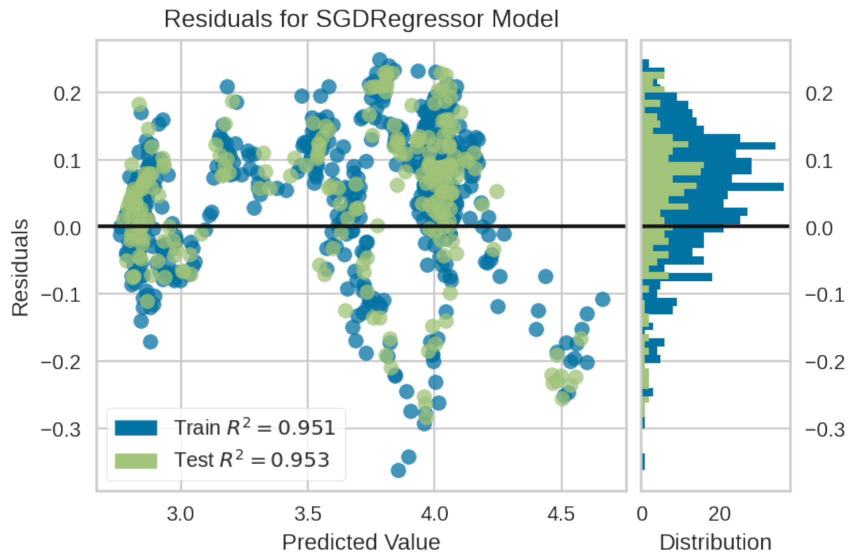


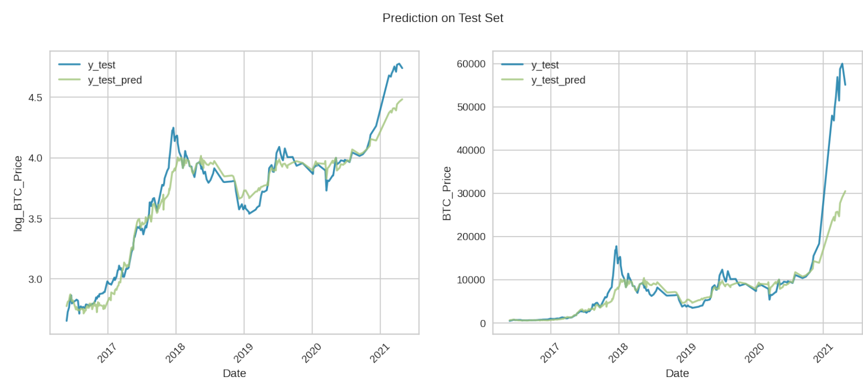


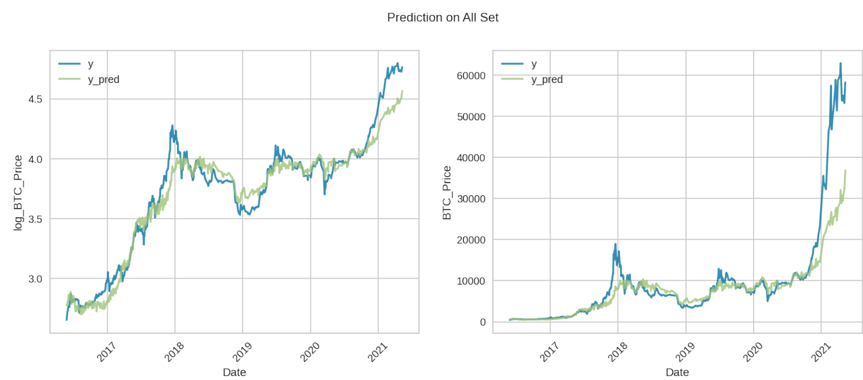


Stochastic Gradient Descent (SGD)

[ ]:
```
name = 'SGD'
SGD = SGDRegressor(max_iter = 1000, tol = 1e-3)
viz = residuals_plot(SGD, x_train, y_train_np, x_test, y_test_np)

SGD.fit(x_train, y_train_np)
evaluate(SGD, name, x_test, y_test)
plot_prediction(SGD, x, y, x_train, y_train, x_test, y_test)
```



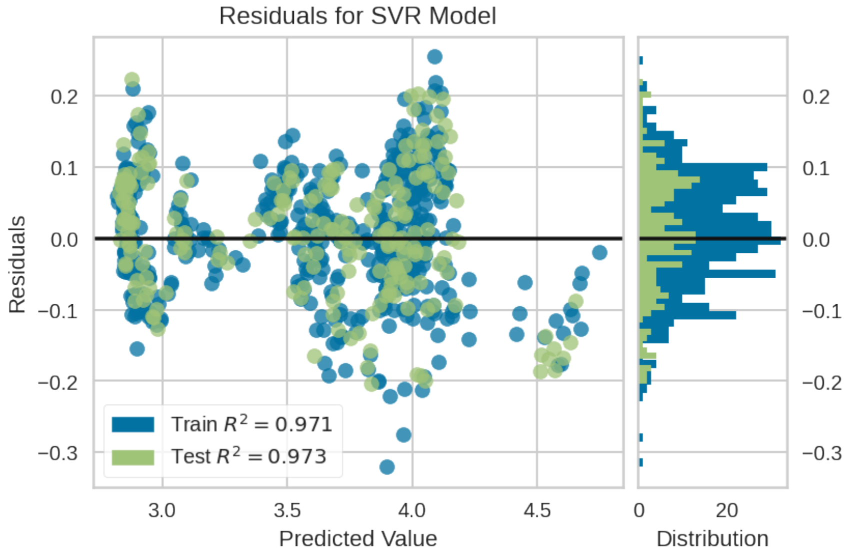


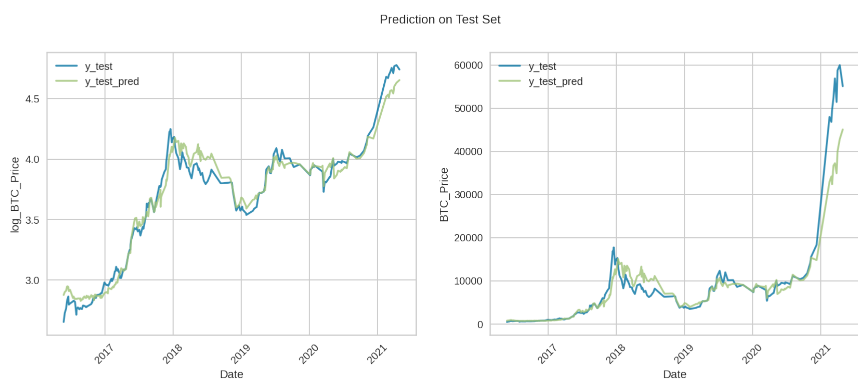


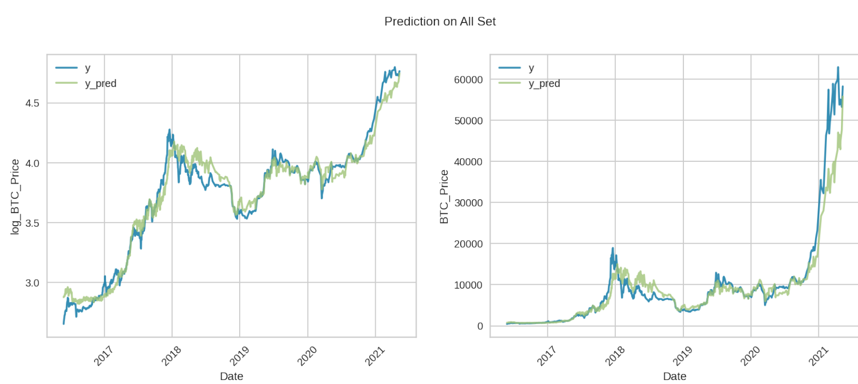
Support Vector Machine using RBF kernel

[ ]:
```
name = 'SVM+RBF'
SVR_rbf = Pipeline([
                    # ("poly", PolynomialFeatures(degree = 2)),
                    # ("std_scaler", StandardScaler()),
                    ("rbf", svm.SVR(kernel = 'rbf'))
                    ])
viz = residuals_plot(SVR_rbf, x_train, y_train_np, x_test, y_test_np)
```

```
SVR_rbf.fit(x_train, y_train_np)
evaluate(SVR_rbf, name, x_test, y_test)
plot_prediction(SVR_rbf, x, y, x_train, y_train, x_test, y_test)
```

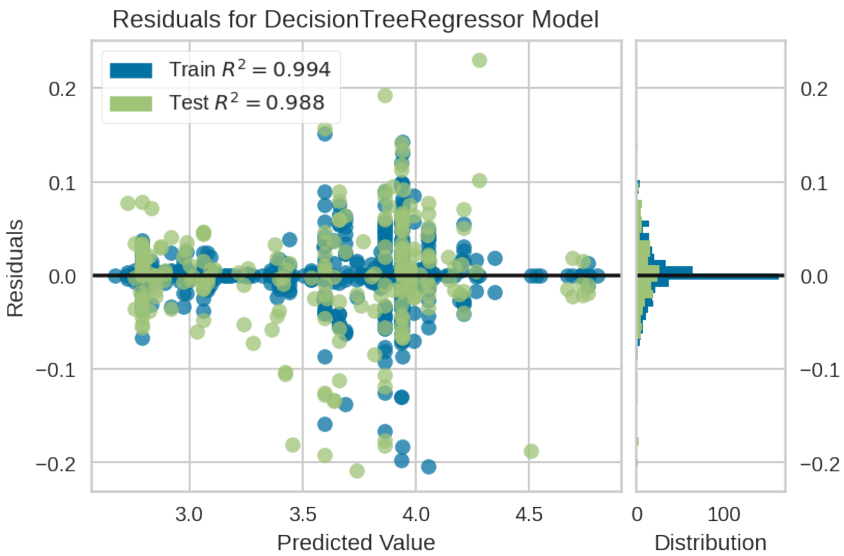


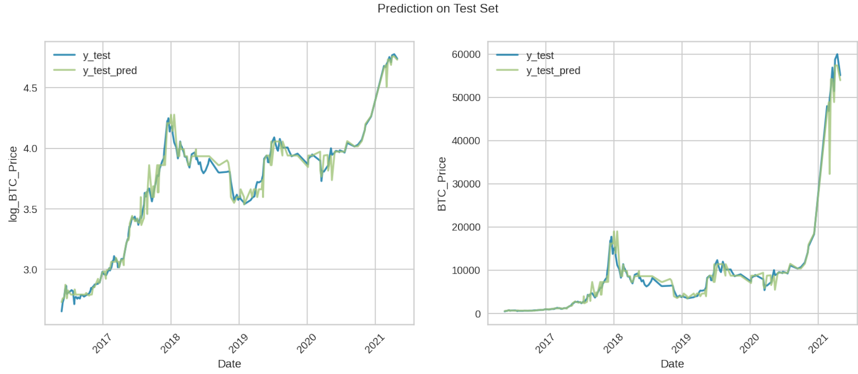


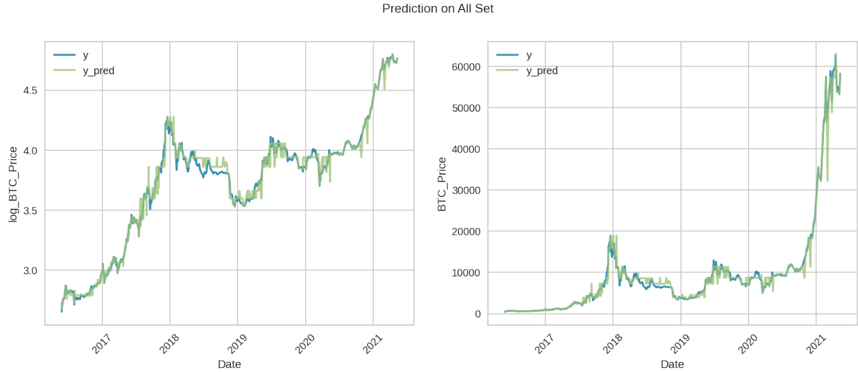


Decision Tree

[ ]:
```
name = 'Decision Tree'
DecisionTree = DecisionTreeRegressor(max_depth = 8, random_state = 0)
viz = residuals_plot(DecisionTree, x_train, y_train_np, x_test, y_test_np)

DecisionTree.fit(x_train, y_train)
evaluate(DecisionTree, name, x_test, y_test)
plot_prediction(DecisionTree, x, y, x_train, y_train, x_test, y_test)
```

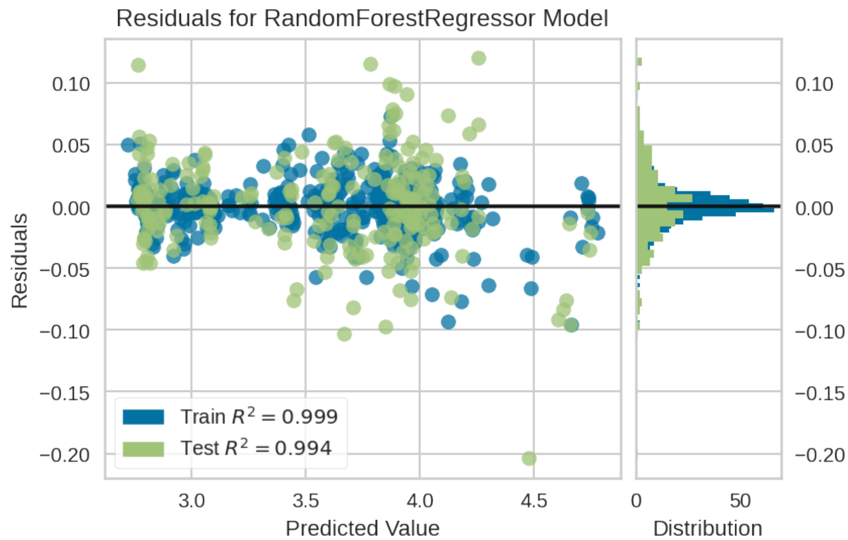




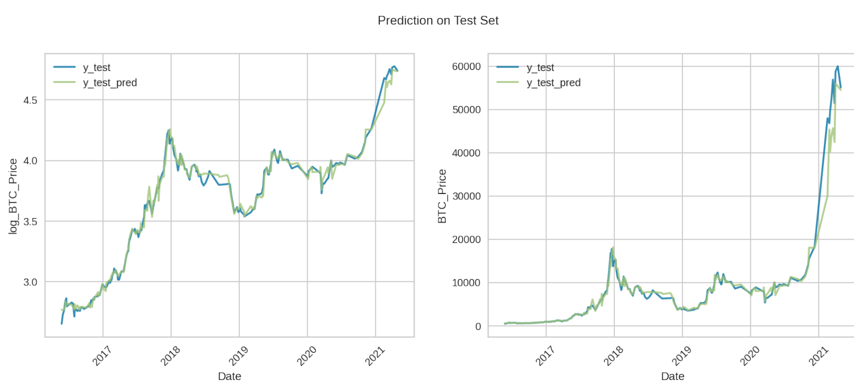


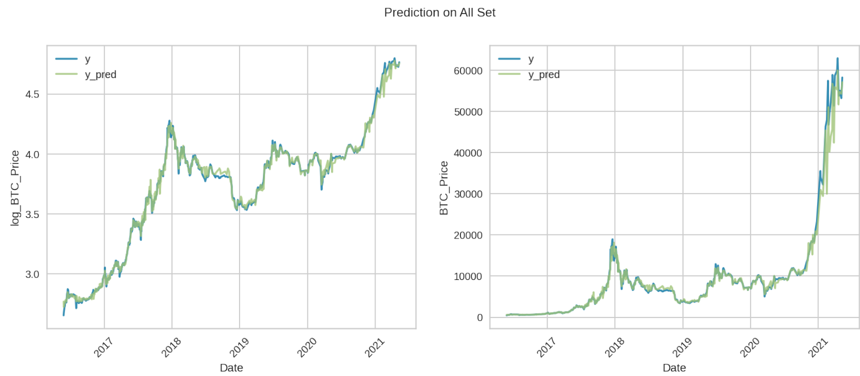
Random Forest

```
[ ]: name = 'Random Forest'
RandomForest = RandomForestRegressor(n_estimators = 1000, random_state = 0)
viz = residuals_plot(RandomForest, x_train, y_train_np, x_test, y_test_np)

RandomForest.fit(x_train, y_train_np)
print(RandomForest.feature_importances_)
print(RandomForest.feature_names_in_)
evaluate(RandomForest, name, x_test, y_test)
plot_prediction(RandomForest, x, y, x_train, y_train, x_test, y_test)
```



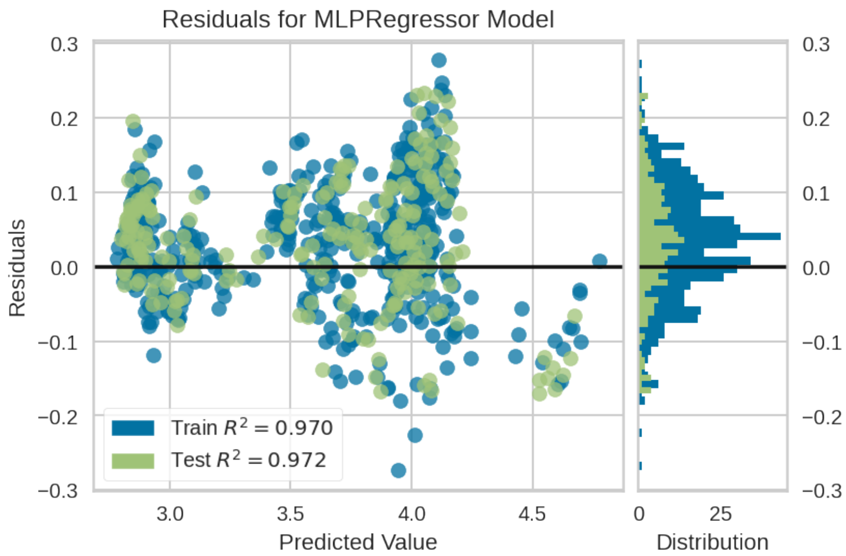

```
[0.21507476 0.08088493 0.70404031]
['log_BTC_PerTxFee' 'log_ETH_Price' 'log_BTC_HashRate']
```

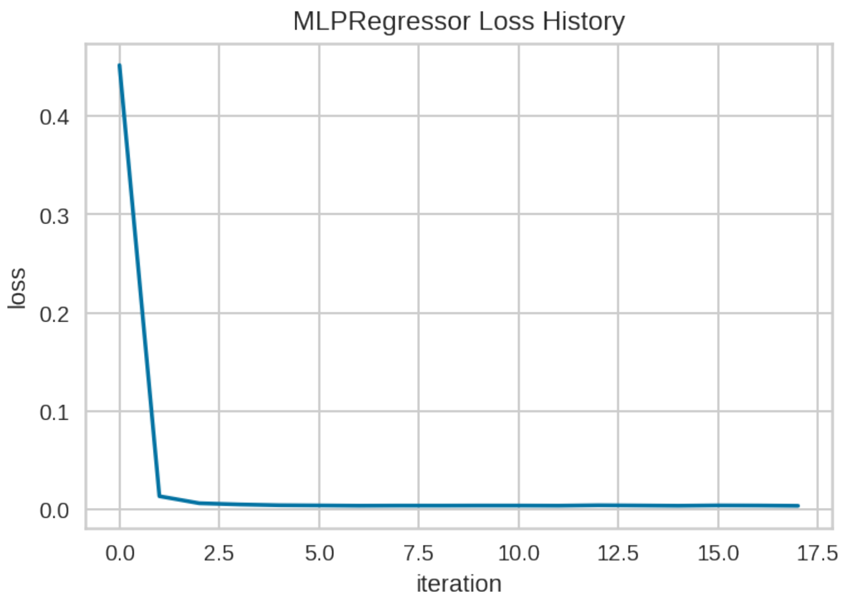


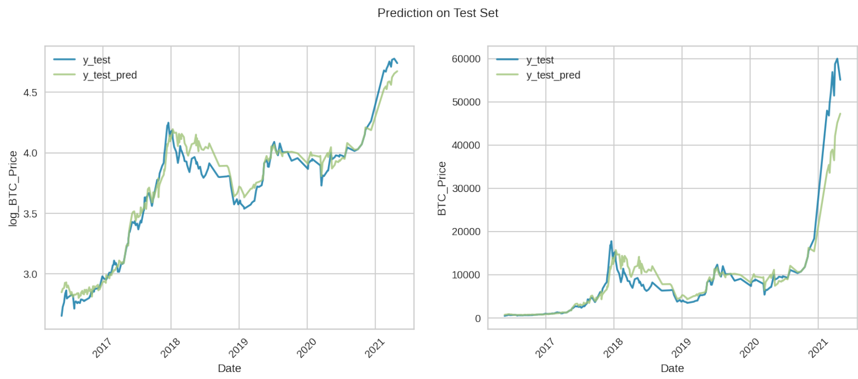


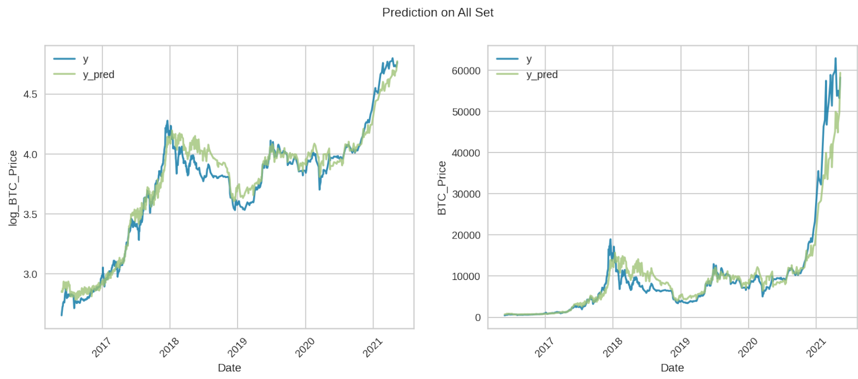
Neural Network

```
[ ]: name = 'NN'
MLP = MLPRegressor(hidden_layer_sizes=(100, 50), batch_size=10, max_iter = 500, learning_rate =␣
 ↪'adaptive', solver = 'adam', random_state = 1)
viz = residuals_plot(MLP, x_train, y_train_np, x_test, y_test_np)

MLP.fit(x_train, y_train_np)
# plot loss curve
plt.plot(MLP.loss_curve_)
plt.title('MLPRegressor Loss History')
plt.xlabel('iteration')
plt.ylabel('loss')
plt.show()

evaluate(MLP, name, x_test, y_test)
plot_prediction(MLP, x, y, x_train, y_train, x_test, y_test)
```







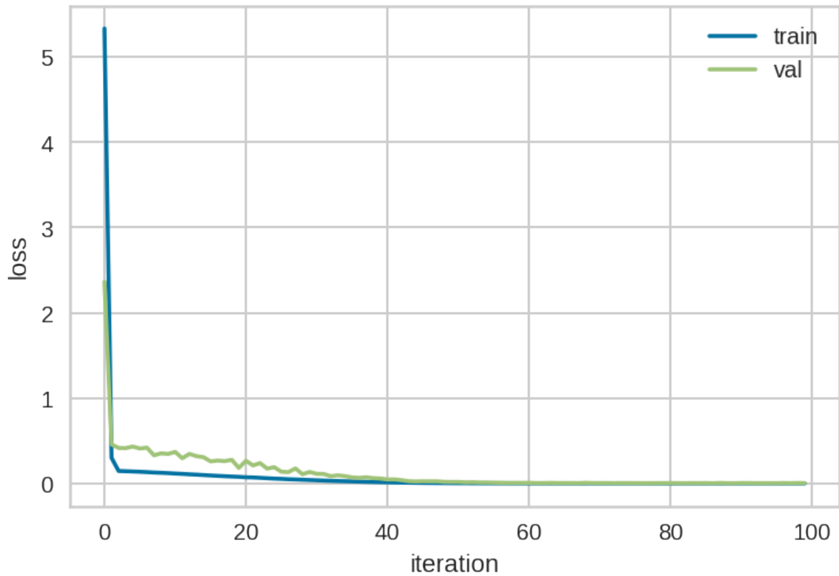


Long Short-Term Memory (LSTM)

[ ]:
```
model = Sequential()
model.add(LSTM(units = 128, activation = 'sigmoid', input_shape = (None, 1)))
model.add(Dense(units = 1))
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

```
history = model.fit(X_train, Y_train, batch_size = 10, epochs = 100, verbose = 1, validation_data =␣
 ↪(X_test, Y_test))
```

[ ]:
```
plt.plot(history.history['loss'], label = 'train')
plt.plot(history.history['val_loss'], label = 'val')
plt.xlabel('iteration')
plt.ylabel('loss')
plt.legend()
plt.show()
```



[ ]:
```
Y_test_pred = model.predict(X_test)
Y_test_pred = pd.DataFrame(Y_test_pred)
Y_test_pred.index = Y_test.index

plt.figure(figsize=(14, 5))
plt.plot(Y_train, label = 'Y_train')
plt.plot(Y_test_pred, label = 'Y_test_pred')
plt.plot(Y_test, label = 'Y_test')
plt.xlabel('Date')
plt.ylabel('log_BTC_Price')
plt.legend()
plt.show()

plt.figure(figsize=(14, 5))
plt.plot(np.power(10, Y_train), label = 'Y_train')
plt.plot(np.power(10, Y_test_pred), label = 'Y_test_pred')
plt.plot(np.power(10, Y_test), label = 'Y_test')
plt.xlabel('Date')
plt.ylabel('BTC_Price')
plt.legend()
plt.show()
evaluate(model, 'LSTM', X_test, Y_test)
```

Y_train
Y_test_pred
Y_test
4.5
4.0
3.5
3.0
log_BTC_Price
2017
2018
2019
2020
2021
Date

Y_train
Y_test_pred
Y_test
60000
50000
40000
30000
20000
10000
0
BTC_Price
2017
2018
2019
2020
2021
Date