Test

April 12, 2020

[1]:
```
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt

from sklearn.metrics import mean_squared_error

from scipy import stats

import math

from datetime import datetime

import time

import warnings; warnings.simplefilter('ignore')
```

[2]:
```
# load the data
data = pd.read_csv('btc_ta.csv')
```

[3]:
```
# examine the features
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2638440 entries, 0 to 2638439
Data columns (total 74 columns):
Unnamed: 0                  int64
time                        int64
open                        float64
close                       float64
high                        float64
low                         float64
volume                      float64
volume_adi                  float64
volume_obv                  float64
volume_cmf                  float64
volume_fi                   float64
volume_em                   float64
```

volume_sma_em	float64
volume_vpt	float64
volume_nvi	float64
volatility_atr	float64
volatility_bbm	float64
volatility_bbh	float64
volatility_bbl	float64
volatility_bbw	float64
volatility_bbhi	float64
volatility_bbli	float64
volatility_kcc	float64
volatility_kch	float64
volatility_kcl	float64
volatility_kchi	float64
volatility_kcli	float64
volatility_dcl	float64
volatility_dch	float64
volatility_dchi	float64
volatility_dcli	float64
trend_macd	float64
trend_macd_signal	float64
trend_macd_diff	float64
trend_ema_fast	float64
trend_ema_slow	float64
trend_adx	float64
trend_adx_pos	float64
trend_adx_neg	float64
trend_vortex_ind_pos	float64
trend_vortex_ind_neg	float64
trend_vortex_ind_diff	float64
trend_trix	float64
trend_mass_index	float64
trend_cci	float64
trend_dpo	float64
trend_kst	float64
trend_kst_sig	float64
trend_kst_diff	float64
trend_ichimoku_a	float64
trend_ichimoku_b	float64
trend_visual_ichimoku_a	float64
trend_visual_ichimoku_b	float64
trend_aroon_up	float64
trend_aroon_down	float64
trend_aroon_ind	float64
trend_psar	float64
trend_psar_up	float64
trend_psar_down	float64
trend_psar_up_indicator	float64

```
trend_psar_down_indicator    float64
momentum_rsi                 float64
momentum_mfi                 float64
momentum_tsi                 float64
momentum_uo                  float64
momentum_stoch               float64
momentum_stoch_signal        float64
momentum_wr                  float64
momentum_ao                  float64
momentum_kama                float64
momentum_roc                 float64
others_dr                    float64
others_dlr                   float64
others_cr                    float64
dtypes: float64(72), int64(2)
memory usage: 1.5 GB
```

[4]:
```
data.columns.to_series()[np.isinf(data).any()]
```

[4]:
```
trend_cci    trend_cci
dtype: object
```

[5]:
```
# create the target feature
data['nextClosingPrice'] = data['close'].shift(-1)

# drop the rows with 'None' in target column
data = data.dropna(subset=['nextClosingPrice'])
data = data.drop(['trend_psar_down', 'trend_psar_up', 'trend_cci'], axis=1)

# drop na values from feature extraction
data = data.dropna()

data['time'] = pd.to_datetime(data['time'], unit='ms')

data = data.reset_index()
```

[6]:
```
from xgboost import XGBRegressor
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.tree import DecisionTreeRegressor
from lightgbm import LGBMRegressor
from sklearn.svm import SVR
```

[7]:
```
def testModel(df, windowLength, startingIndex, endingIndex, stepSize,␣
 ↪modelName):

    # lists to store data, will concat to make result data frame
    rmseList = []
```

```
predList = []
realList = []
predTimeList = []
trainingTimeList = []

# extract feature and test data
X = df.drop(['Unnamed: 0', 'time', 'nextClosingPrice'], axis=1)
y = df['nextClosingPrice']

# rolling window test
for i in range(startingIndex, endingIndex, stepSize):

    # split the data
    X_train, X_test = X[i-windowLength:i], X[i:i+1]
    y_train, y_test = y[i-windowLength:i], y[i]

    # start timer
    startTime = time.time()

    # create a new model
    if modelName == 'dt':
        model = DecisionTreeRegressor()
    elif modelName == 'xgb':
        model = XGBRegressor()
    elif modelName == 'lgbm':
        model = LGBMRegressor()
    elif modelName == 'lr':
        model = LinearRegression()
    elif modelName == 'ridge':
        model = Ridge(alpha=0.01)
    elif modelName == 'svr':
        model = SVR(kernel='rbf', C=100, gamma=0.1, epsilon=.1)


    # train the model
    model.fit(X_train, y_train)

    # make a prediction
    y_pred = model.predict(X_test)

    endTime = time.time()

    # record time figures for result data frame
    predTimeList.append(df['time'][i])
    predList.append(y_pred)
    realList.append(y_test)
    trainingTimeList.append(endTime - startTime)
```

```
        # measure the error of this prediction
        squared_error = (y_test - y_pred) **2
        root_squared_error = math.sqrt(squared_error)
        rmseList.append(root_squared_error)

    # result dictionary
    predList = [x[0] for x in predList]
    result_data = pd.DataFrame({'Timestamp': predTimeList,
                                "Real": realList,
                                "Preds": predList,
                                'rmse': rmseList,
                                'timeToTrain': trainingTimeList})

    print("")
    print("RMSE mean:{}".format(result_data['rmse'].mean()))
    print("RMSE std:{}".format(result_data['rmse'].std()))
    print("Time to train mean:{}".format(result_data['timeToTrain'].mean()))
    print("Time to train std:{}".format(result_data['timeToTrain'].std()))

    # result plot
    plt.figure(figsize=(16,10))
    plt.plot('Timestamp', 'Real', data=result_data)
    plt.plot('Timestamp', 'Preds', data=result_data)
    plt.legend()
    plt.show()

    return result_data
```

[8]:
```
param_grid = [60, 90, 120, 150]

# run tests
for p in param_grid:
    print("Window size: {}".format(p))
    dt_results = testModel(data, p, len(data)-50000, len(data), 2, 'dt')
    print("")
```

```
Window size: 60

RMSE mean:4.638939693606274
RMSE std:8.210606002518054
Time to train mean:0.0026932621097564696
Time to train std:0.0004994180707689732
```


```
Window size: 90

RMSE mean:4.679039009028339
RMSE std:7.829549303364206
Time to train mean:0.0036514265823364258
Time to train std:0.000551825201694003
```


```
Window size: 120

RMSE mean:4.647891225381853
RMSE std:7.756624294927457
Time to train mean:0.006153911180496216
Time to train std:0.0020641802393237653
```


```
Window size: 150

RMSE mean:4.6665147087087195
RMSE std:8.597914027793692
Time to train mean:0.005746883535385132
Time to train std:0.0007238185480066518
```

[9]:
```
for p in param_grid:
    print("Window size: {}".format(p))
    lgbm_results = testModel(data, p, len(data)-50000, len(data), 2, 'lgbm')
    print("")
```

```
Window size: 60

RMSE mean:5.673950787085897
RMSE std:8.521156099935272
Time to train mean:0.09505588418960571
Time to train std:0.005546142078377259
```

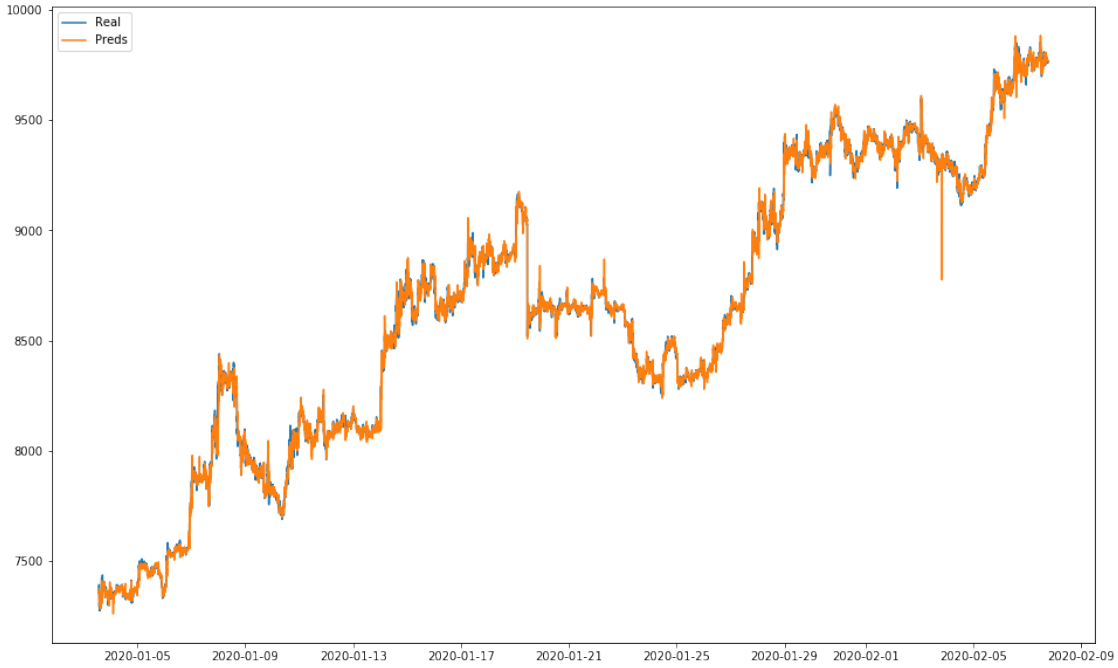

```
Window size: 90

RMSE mean:4.981967444372509
RMSE std:7.708800320047909
Time to train mean:0.10425255514144897
Time to train std:0.006154479464668289
```


```
Window size: 120

RMSE mean:4.788659311347514
RMSE std:7.451486803756344
Time to train mean:0.11256856640815735
Time to train std:0.006769580298384563
```

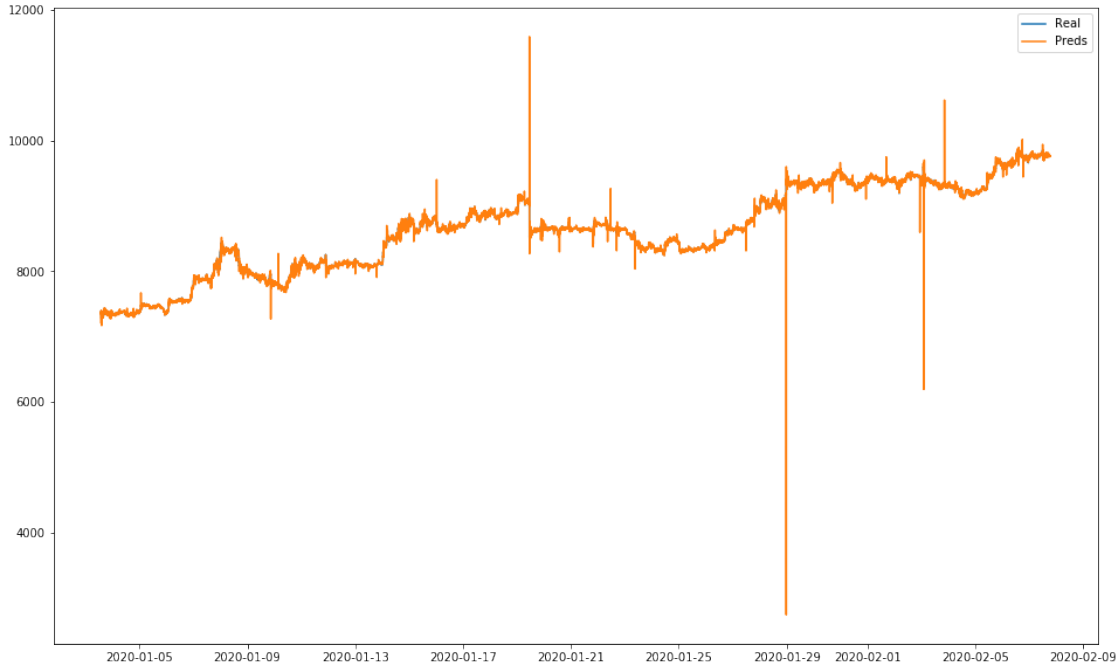

```
Window size: 150

RMSE mean:4.648117379880765
RMSE std:7.347391633327303
Time to train mean:0.12222463409423828
Time to train std:0.008368539252456685
```

[10]:
```
for p in param_grid:
    print("Window size: {}".format(p))
    ridge_results = testModel(data, p, len(data)-50000, len(data), 2, 'ridge')
    print("")
```

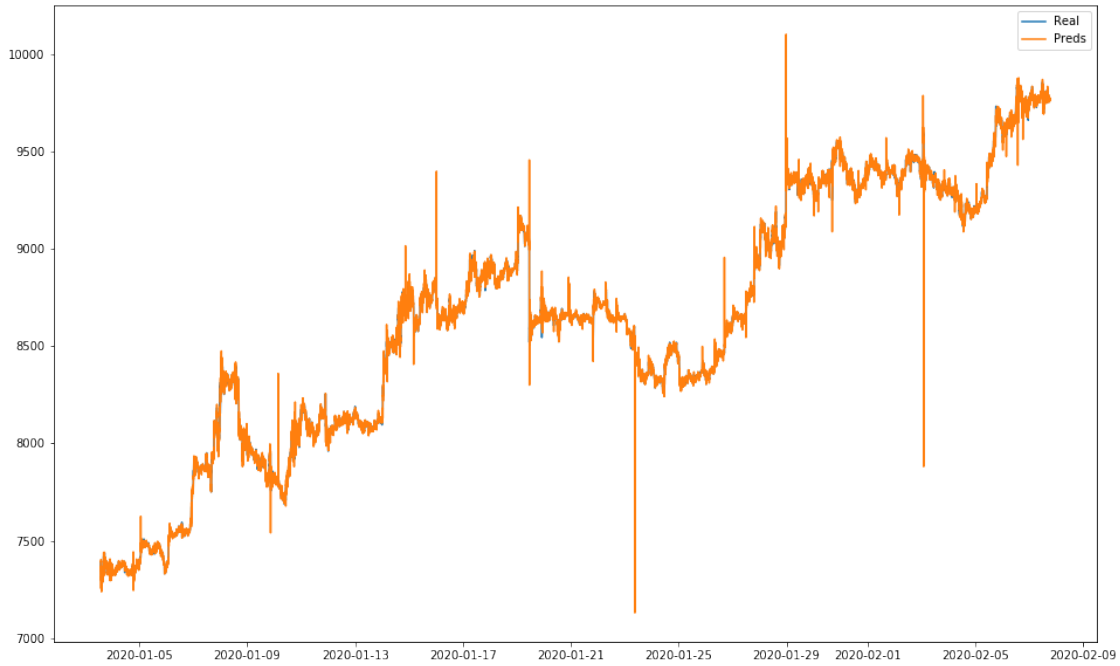
```
Window size: 60

RMSE mean:13.771232988350217
RMSE std:17.312933028130935
Time to train mean:0.003838414611816406
Time to train std:0.0004665987593372468
```




```
Window size: 90

RMSE mean:9.183727604991711
RMSE std:53.92721577340675
Time to train mean:0.003388536958694458
Time to train std:0.0005169001243939896
```




```
Window size: 120

RMSE mean:7.1532984023341655
RMSE std:20.699024180814927
Time to train mean:0.003414637279510498
Time to train std:0.000529677929696571
```

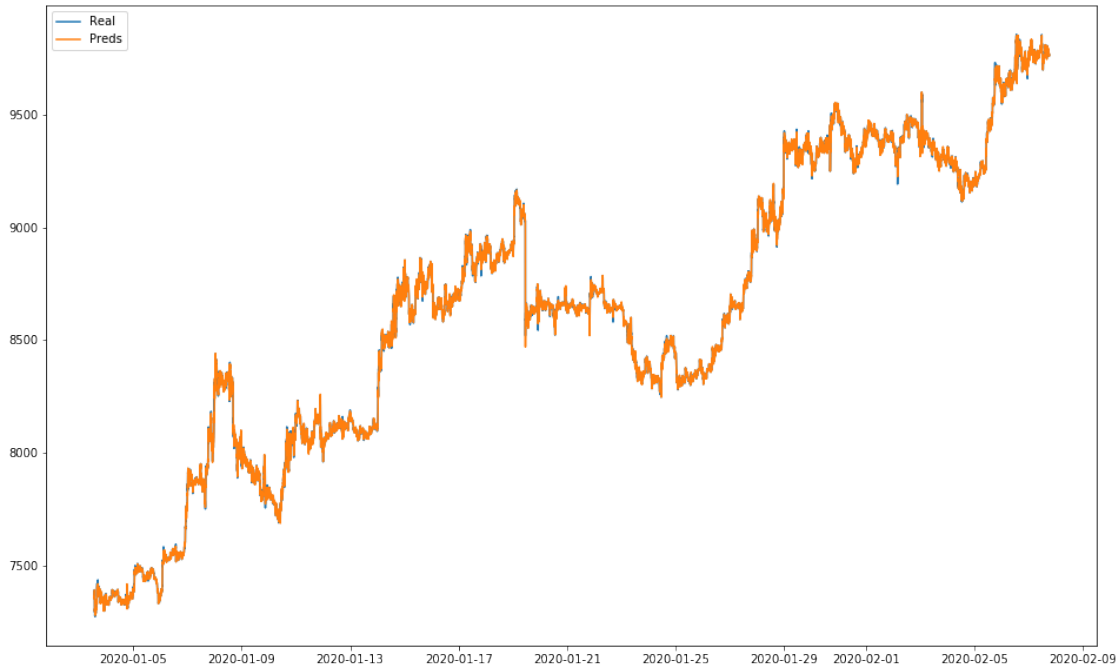



```
Window size: 150

RMSE mean:6.284559509097924
RMSE std:19.825344169559717
Time to train mean:0.0034251598834991455
Time to train std:0.0005287769020462548
```

[11]:
```
for p in param_grid:
    print("Window size: {}".format(p))
    xgb_results = testModel(data, p, len(data)-50000, len(data), 2, 'xgb')
    print("")
```

```
Window size: 60

RMSE mean:3.8783399890893966
RMSE std:5.9272745208089
Time to train mean:0.12389793595314026
Time to train std:0.0113549469012507
```


```
Window size: 90

RMSE mean:3.9752563368236262
RMSE std:5.9358241939401815
Time to train mean:0.13500134643554687
Time to train std:0.008693635354240134
```

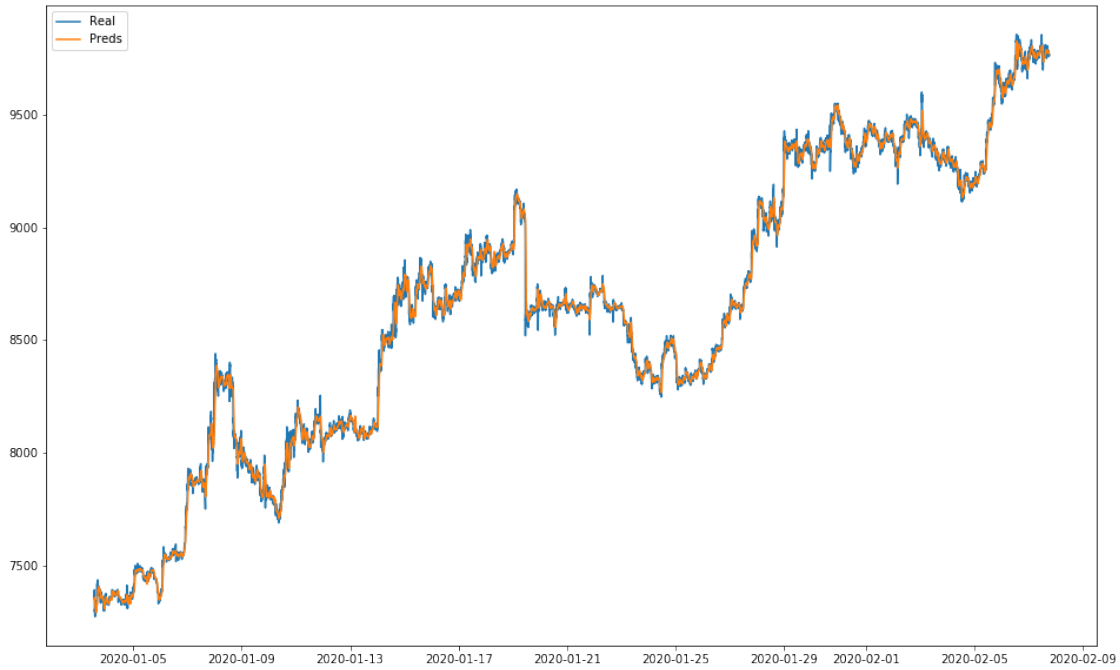



```
Window size: 120

RMSE mean:4.0687625853161755
RMSE std:6.069449951947895
Time to train mean:0.1486352885532379
Time to train std:0.009497299210269039
```

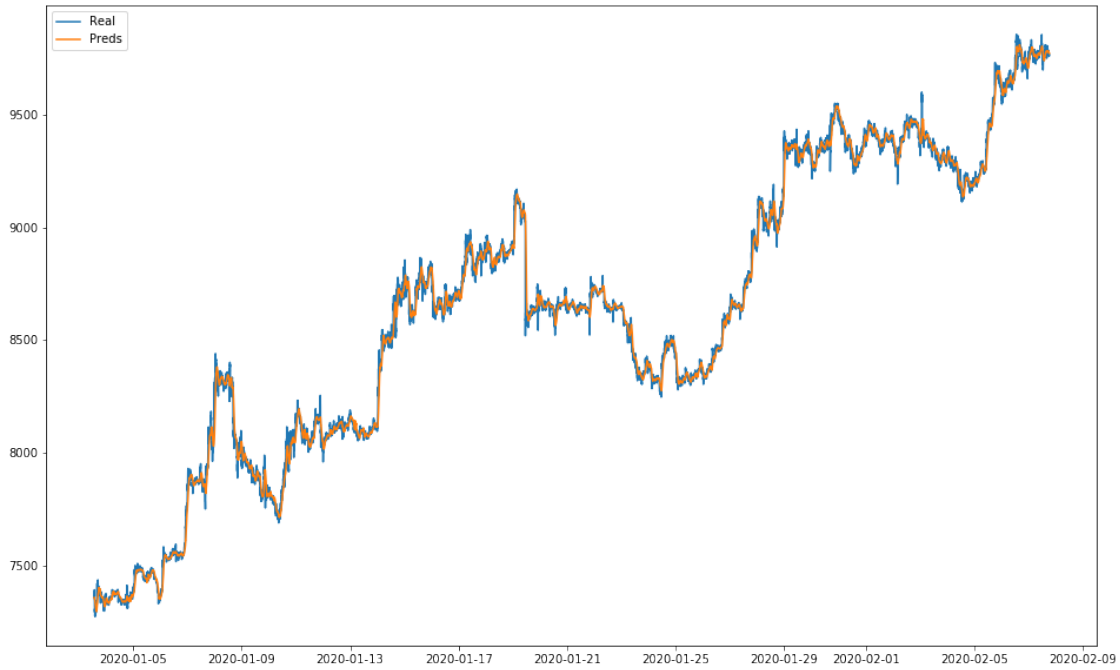

```
Window size: 150

RMSE mean:4.111496452629686
RMSE std:6.108499917927496
Time to train mean:0.16514595043182373
Time to train std:0.014548671601417896
```

[12]:
```
for p in param_grid:
    print("Window size: {}".format(p))
    svr_results = testModel(data, p, len(data)-50000, len(data), 2, 'svr')
    print("")
```

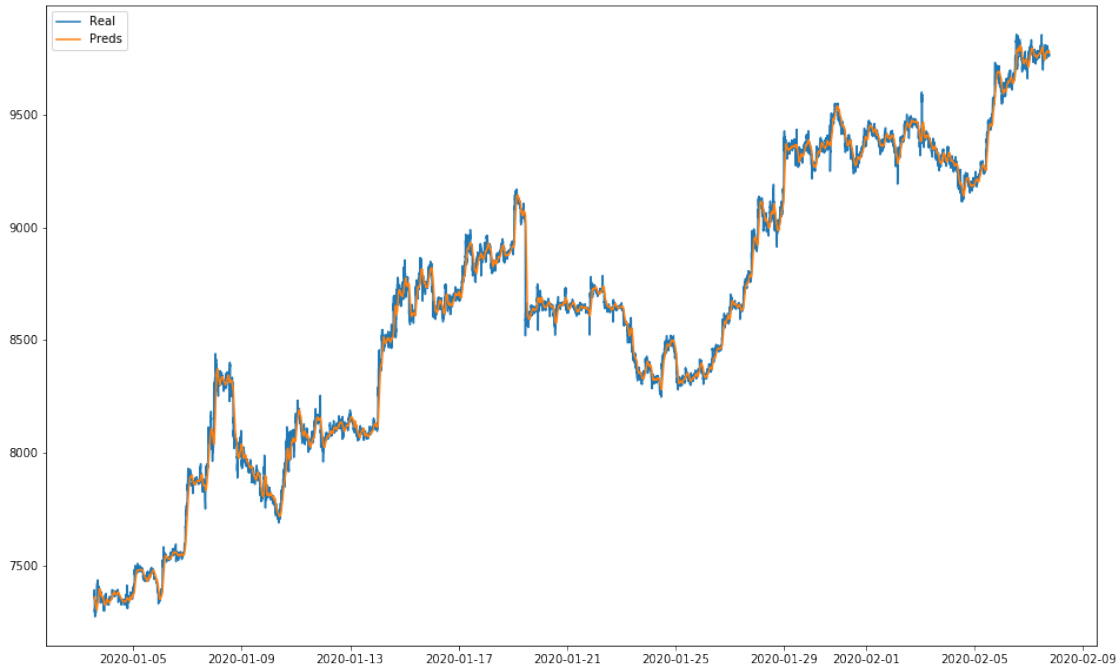
```
Window size: 60

RMSE mean:17.6616931750076
RMSE std:21.092391162230683
Time to train mean:0.0021151193237304686
Time to train std:0.0003705970685899616
```



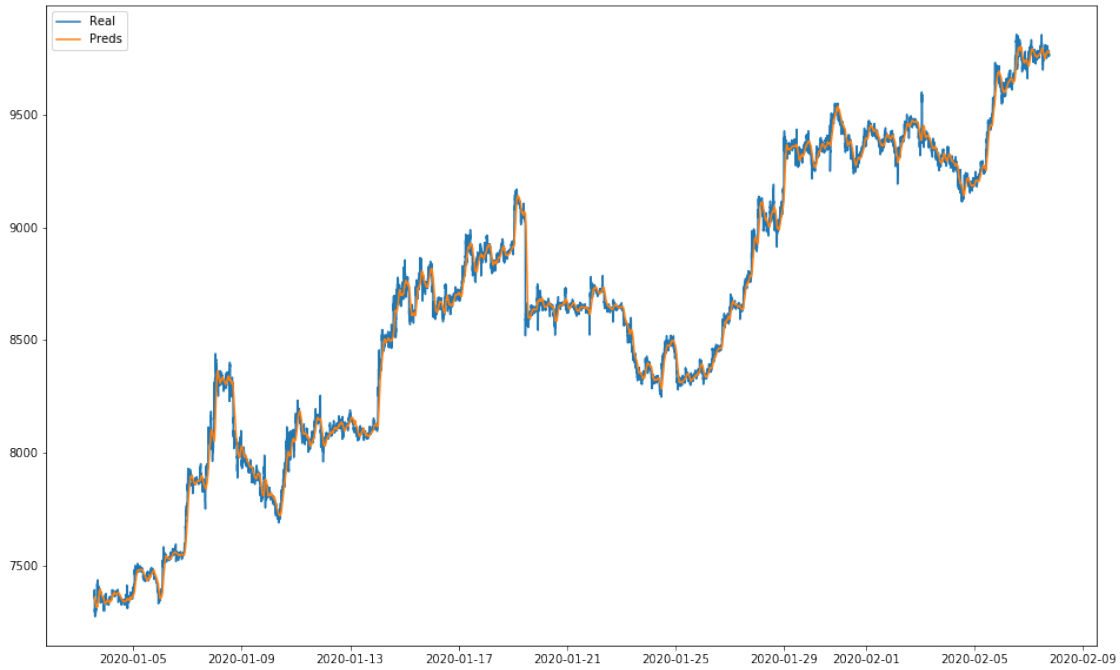

```
Window size: 90

RMSE mean:21.291229795768015
RMSE std:25.010775797952846
Time to train mean:0.002848164873123169
Time to train std:0.00047643691745110576
```




```
Window size: 120

RMSE mean:24.34284362720395
RMSE std:28.366501335063372
Time to train mean:0.0037944900226593017
Time to train std:0.0005224758951400464
```

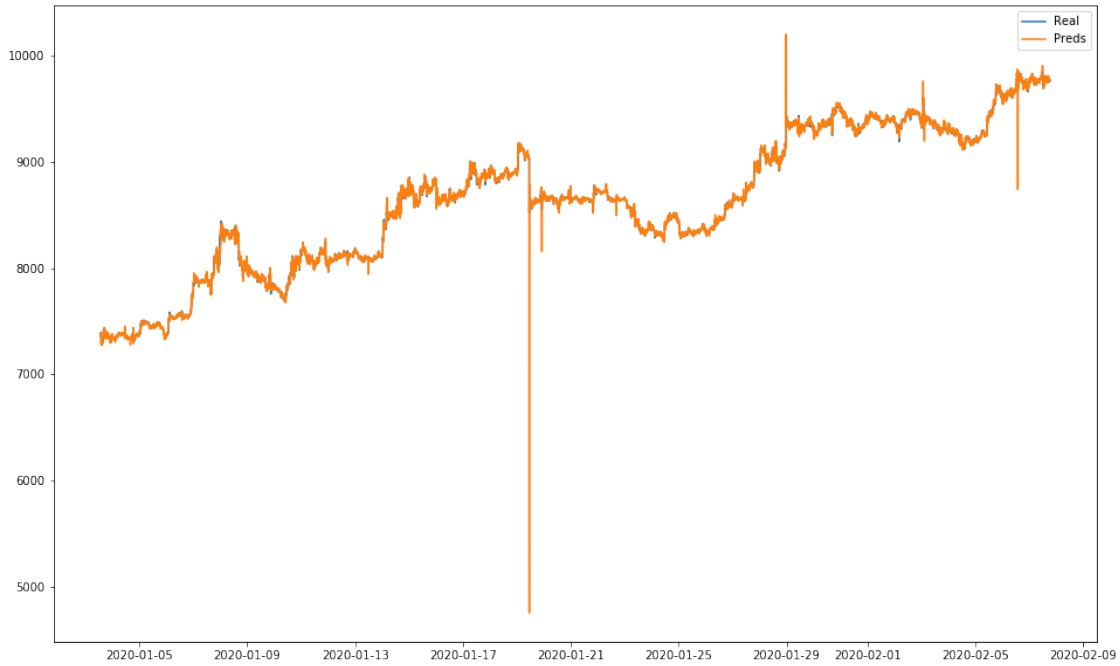



```
Window size: 150

RMSE mean:26.98260098570874
RMSE std:31.26025077433391
Time to train mean:0.005064949865341187
Time to train std:0.0007086567792914443
```

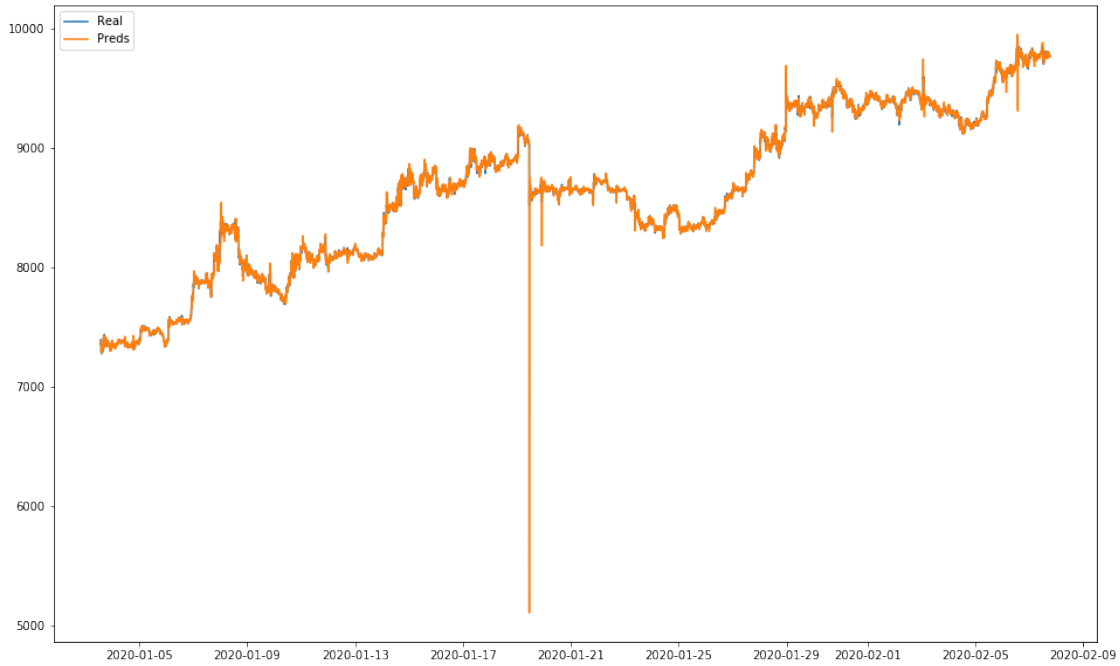


[13]:
```
for p in param_grid:
    print("Window size: {}".format(p))
    lr_results = testModel(data, p, len(data)-50000, len(data), 2, 'lr')
    print("")
```

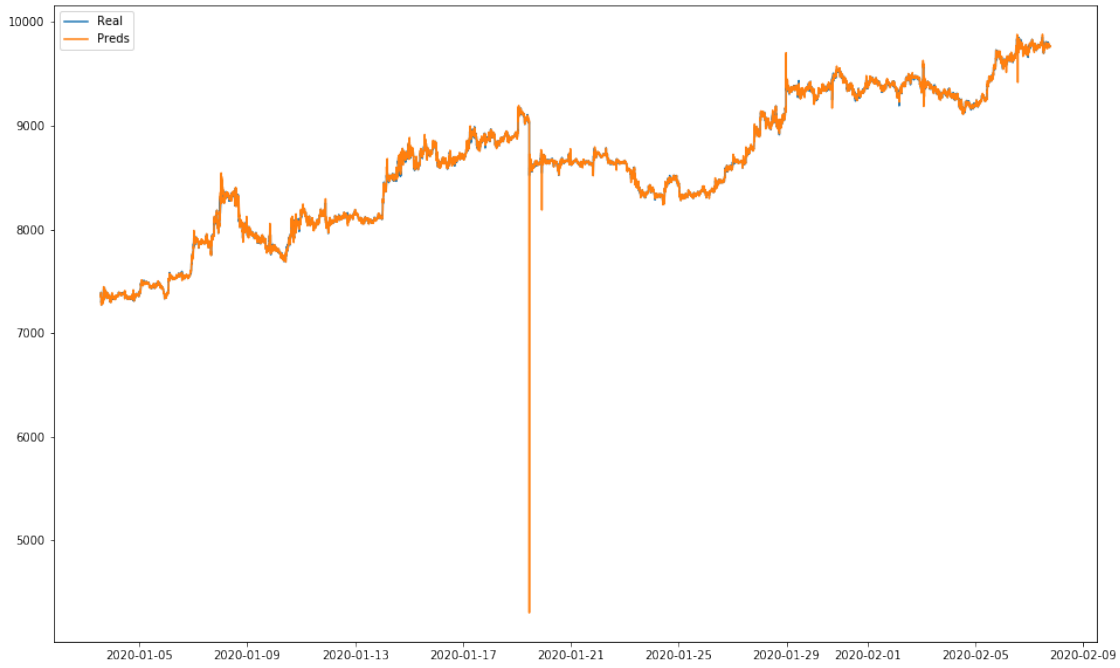
```
Window size: 60

RMSE mean:6.174055205302893
RMSE std:26.625262995811333
Time to train mean:0.0035627897357940674
Time to train std:0.0005801649551287051
```



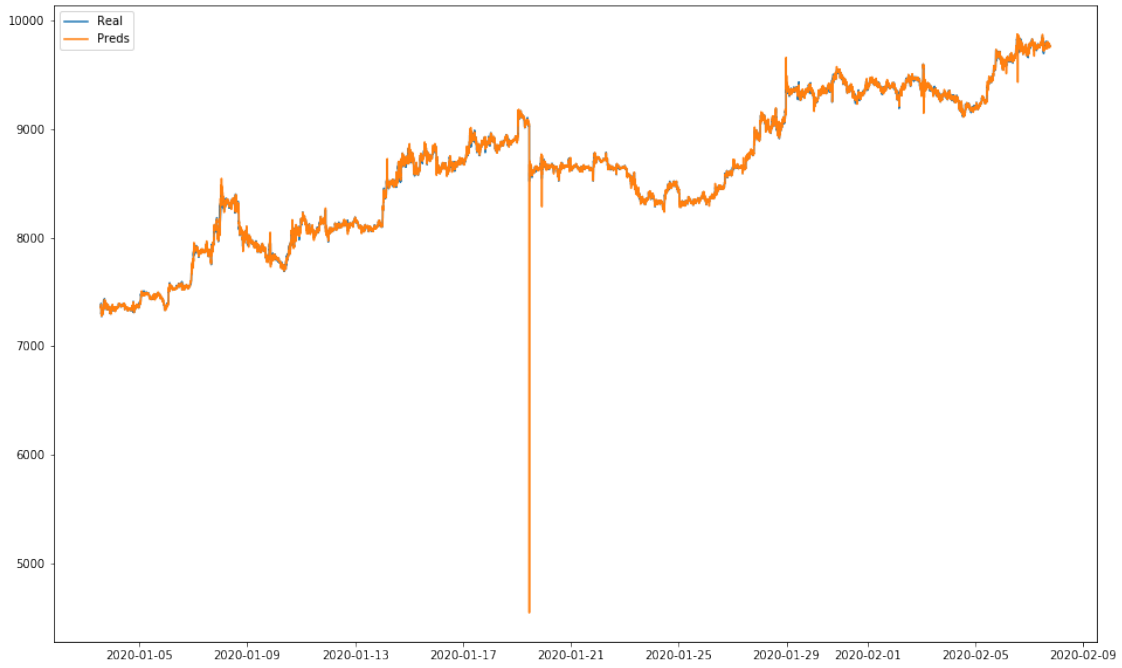

```
Window size: 90

RMSE mean:6.563671480010988
RMSE std:23.529883635638022
Time to train mean:0.0036942616176605225
Time to train std:0.0005710440618869337
```




```
Window size: 120

RMSE mean:7.164818497674767
RMSE std:28.35840171503268
Time to train mean:0.003902086048126221
Time to train std:0.0004396947872672526
```




```
Window size: 150

RMSE mean:7.7766982615291385
RMSE std:27.070628237277443
Time to train mean:0.004488211393356323
Time to train std:0.0005701864723498681
```



[14]:
```
pred_results = pd.DataFrame({'Timestamp': dt_results.Timestamp,
                        'Real': dt_results.Real,
                        'Dt_preds': dt_results.Preds,
                        'Lgbm_preds': lgbm_results.Preds,
                        'Xgb_preds': xgb_results.Preds,
                        'Lr_preds': lr_results.Preds,
                        'Ridge_preds': ridge_results.Preds,
                        'Svr_preds': svr_results.Preds})

rmse_results = pd.DataFrame({'Timestamp': dt_results.Timestamp,
                         'Dt_rmse': dt_results.rmse,
                        'Lgbm_rmse': lgbm_results.rmse,
                        'Xgb_rmse': xgb_results.rmse,
                        'Lr_rmse': lr_results.rmse,
                        'Ridge_rmse': ridge_results.rmse,
                        'Svr_rmse': svr_results.rmse})

time_results = pd.DataFrame({'Timestamp': dt_results.Timestamp,
                        'Dt_timeToTrain': dt_results.timeToTrain,
                        'Lgbm_timeToTrain': lgbm_results.timeToTrain,
                        'Xgb_timeToTrain' : xgb_results.timeToTrain,
                        'Lr_timeToTrain': lr_results.timeToTrain,
                        'Ridge_timeToTrain': ridge_results.timeToTrain,
                        'Svr_timeToTrain': svr_results.timeToTrain})
```

[15]:
```
import seaborn as sns
```

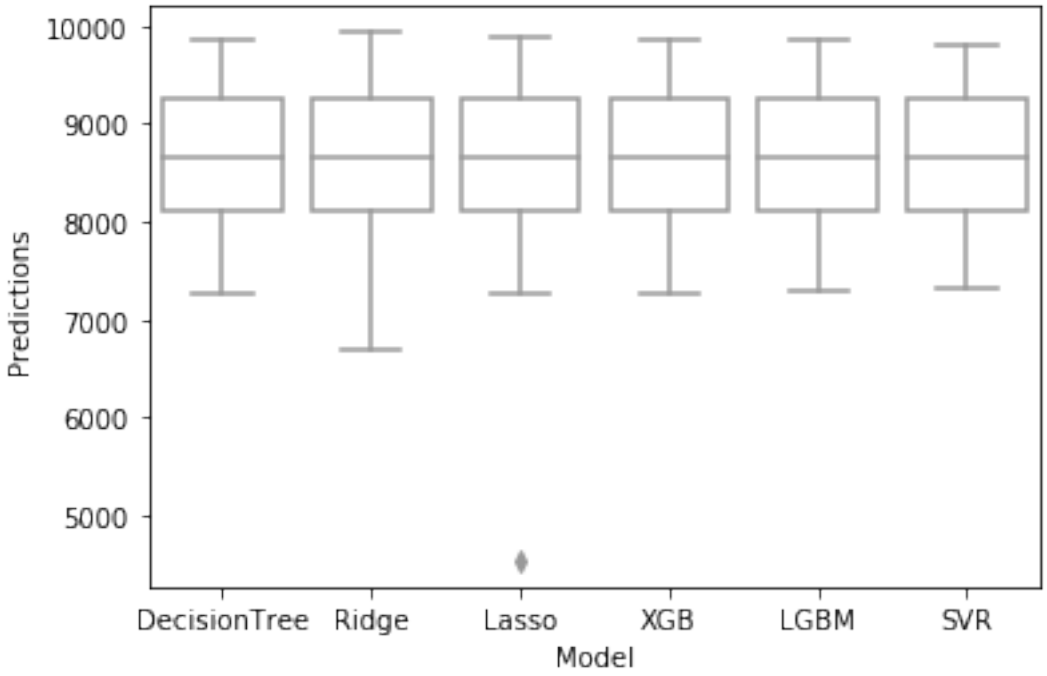
[23]:
```
pred_results.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25000 entries, 0 to 24999
Data columns (total 8 columns):
Timestamp      25000 non-null datetime64[ns]
Real           25000 non-null float64
Dt_preds       25000 non-null float64
Lgbm_preds     25000 non-null float64
Xgb_preds      25000 non-null float64
Lr_preds       25000 non-null float64
Ridge_preds    25000 non-null float64
Svr_preds      25000 non-null float64
dtypes: datetime64[ns](1), float64(7)
memory usage: 1.5 MB
```

[24]:
```
display(pred_results.describe())

# Dataset:
a = pd.DataFrame({ 'Model' : np.repeat('DecisionTree',25000), 'Predictions':␣
 ↪pred_results.Dt_preds})
b = pd.DataFrame({ 'Model' : np.repeat('Ridge',25000), 'Predictions':␣
 ↪pred_results.Ridge_preds})
c = pd.DataFrame({ 'Model' : np.repeat('Lasso',25000), 'Predictions':␣
 ↪pred_results.Lr_preds})
d = pd.DataFrame({ 'Model' : np.repeat('XGB',25000), 'Predictions':␣
 ↪pred_results.Xgb_preds})
e = pd.DataFrame({ 'Model' : np.repeat('LGBM',25000), 'Predictions':␣
 ↪pred_results.Lgbm_preds})
f = pd.DataFrame({ 'Model' : np.repeat('SVR',25000), 'Predictions':␣
 ↪pred_results.Svr_preds})

df=a.append(b).append(c).append(d).append(e).append(f)

# Usual boxplot
sns.boxplot(x='Model', y='Predictions', color='w', data=df)
plt.show()
```

```
                Real      Dt_preds    Lgbm_preds     Xgb_preds      Lr_preds  \
count   25000.000000  25000.000000  25000.000000  25000.000000  25000.000000
mean     8634.667031   8634.582154   8634.338844   8634.046046   8635.058948
std       647.442883    647.386137    647.329022    647.287285    648.356085
min      7275.000000   7276.600000   7283.553240   7279.696777   4541.867634
25%      8124.000000   8123.650000   8123.528385   8122.632324   8123.345848
50%      8650.000000   8649.991514   8649.829074   8649.299805   8650.451832
```

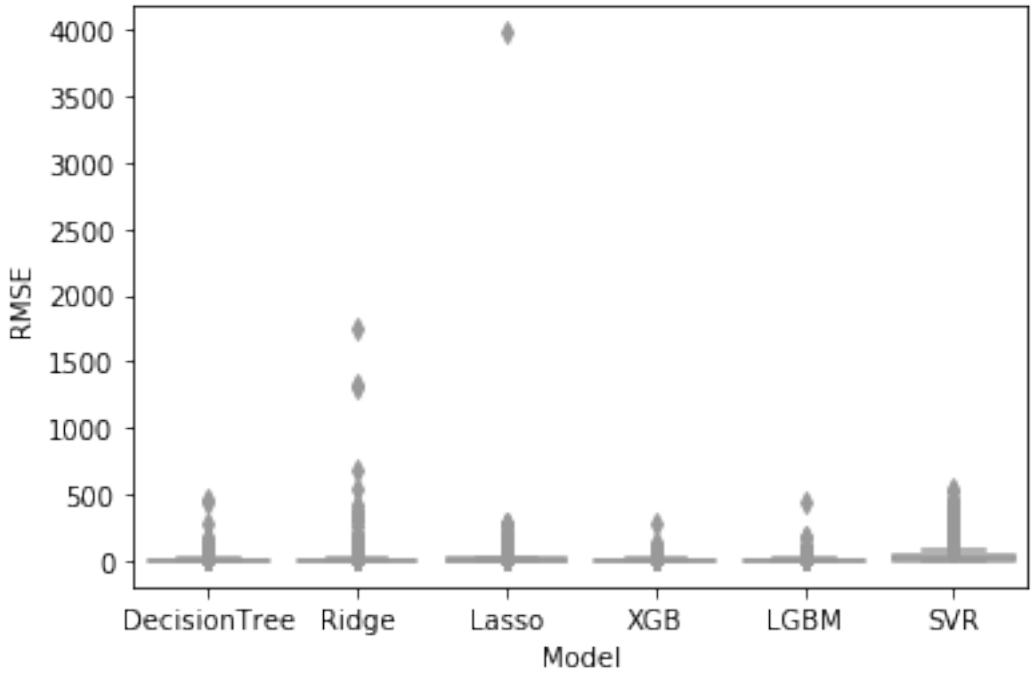
```
75%     9260.000000   9260.000000   9261.104906   9260.102295   9259.336968
max     9855.200000   9858.100000   9851.779984   9854.229492   9874.685383

        Ridge_preds     Svr_preds
count  25000.000000  25000.000000
mean    8634.539770   8631.281587
std      647.641432    647.098565
min     6703.884334   7314.953585
25%     8123.919993   8122.175558
50%     8649.429472   8650.542251
75%     9258.916290   9266.057078
max     9931.389116   9803.443293
```



[25]:
```
display(rmse_results.describe())

# Dataset:
a = pd.DataFrame({ 'Model' : np.repeat('DecisionTree',25000), 'RMSE':␣
 ↪rmse_results.Dt_rmse})
b = pd.DataFrame({ 'Model' : np.repeat('Ridge',25000), 'RMSE': rmse_results.
 ↪Ridge_rmse})
c = pd.DataFrame({ 'Model' : np.repeat('Lasso',25000), 'RMSE': rmse_results.
 ↪Lr_rmse})
d = pd.DataFrame({ 'Model' : np.repeat('XGB',25000), 'RMSE': rmse_results.
 ↪Xgb_rmse})
```

```
e = pd.DataFrame({ 'Model' : np.repeat('LGBM',25000), 'RMSE': rmse_results.
 ↪Lgbm_rmse})
f = pd.DataFrame({ 'Model' : np.repeat('SVR',25000), 'RMSE': rmse_results.
 ↪Svr_rmse})

df=a.append(b).append(c).append(d).append(e).append(f)

# Usual boxplot
sns.boxplot(x='Model', y='RMSE', color='w', data=df)
plt.show()
```

```
           Dt_rmse     Lgbm_rmse      Xgb_rmse       Lr_rmse    Ridge_rmse  \
count  25000.000000  25000.000000  25000.000000  25000.000000  25000.000000
mean       4.666515      4.648117      4.111496      7.776698      6.284560
std        8.597914      7.347392      6.108500     27.070628     19.825344
min        0.000000      0.000093      0.000000      0.000547      0.000073
25%        0.200000      0.861863      0.653320      1.941803      1.471801
50%        2.000000      2.376074      2.116211      4.625999      3.519459
75%        6.071970      5.741862      5.365234      9.647206      7.425046
max      452.479530    438.625990    284.945307   3978.132366   1749.115666

           Svr_rmse
count  25000.000000
mean      26.982601
std       31.260251
min        0.002794
25%        8.006499
50%       17.787803
75%       34.950391
max      540.018127
```

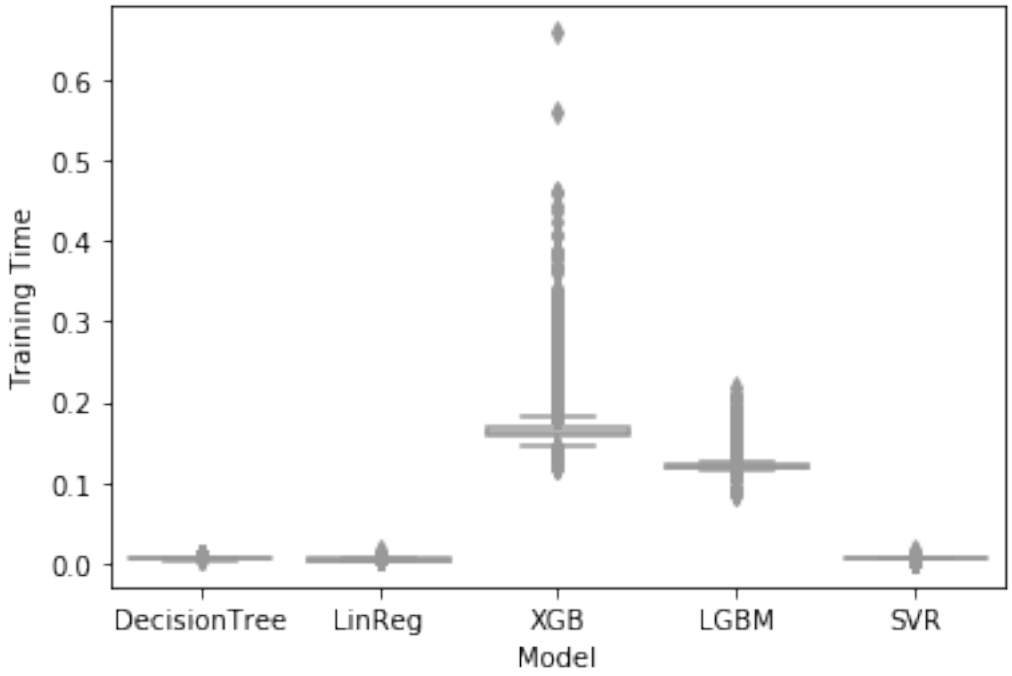


[26]:
```
display(time_results.describe())

# Dataset:
a = pd.DataFrame({ 'Model' : np.repeat('DecisionTree',25000), 'Training Time':␣
 ↪time_results.Dt_timeToTrain})
c = pd.DataFrame({ 'Model' : np.repeat('LinReg',25000), 'Training Time':␣
 ↪time_results.Lr_timeToTrain})
d = pd.DataFrame({ 'Model' : np.repeat('XGB',25000), 'Training Time':␣
 ↪time_results.Xgb_timeToTrain})
e = pd.DataFrame({ 'Model' : np.repeat('LGBM',25000), 'Training Time':␣
 ↪time_results.Lgbm_timeToTrain})
f = pd.DataFrame({ 'Model' : np.repeat('SVR',25000), 'Training Time':␣
 ↪time_results.Svr_timeToTrain})

df=a.append(c).append(d).append(e).append(f)

# Usual boxplot
sns.boxplot(x='Model', y='Training Time', color='w', data=df)
plt.show()
```

```
       Dt_timeToTrain  Lgbm_timeToTrain  Xgb_timeToTrain  Lr_timeToTrain  \
count    25000.000000      25000.000000     25000.000000    25000.000000
mean         0.005747          0.122225         0.165146        0.004488
std          0.000724          0.008369         0.014549        0.000570
min          0.003986          0.086768         0.117686        0.002990
```

25%	0.004987	0.118683	0.158576	0.003989
50%	0.005983	0.119680	0.161568	0.003990
75%	0.005985	0.121675	0.167552	0.004987
max	0.010970	0.217418	0.659237	0.016498

	Ridge_timeToTrain	Svr_timeToTrain
count	25000.000000	25000.000000
mean	0.003425	0.005065
std	0.000529	0.000709
min	0.001982	0.002982
25%	0.002992	0.004984
50%	0.002992	0.004987
75%	0.003989	0.004991
max	0.006982	0.014960



[ ]: