In [6]:

```python
# !pip install plotly
# !pip install pandas
import pandas as pd
# !pip install pandas_ta
# import pandas_ta as ta
import numpy as np
# !pip install seaborn
import seaborn as sns
import datetime
from datetime import timedelta
import math
import requests
import plotly as py
import plotly.express as px
from plotly.subplots import make_subplots
import plotly.graph_objects as go
import matplotlib.pyplot as plt


import sklearn as sk
import sklearn.preprocessing
from sklearn import metrics
from matplotlib.pyplot import figure

from sklearn.preprocessing import  (StandardScaler, MinMaxScaler, LabelBinarizer)
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn import linear_model
from sklearn.linear_model import LinearRegression, LogisticRegression
import xgboost as xgb
from sklearn.metrics import *
# from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn import tree

from sklearn import decomposition
from sklearn.pipeline import Pipeline
from sklearn.metrics import r2_score
# !pip install graphviz
# import graphviz
# import chart_studio.tools as tls
```

Creating DataFrame

In [7]:

```python
#manipulatable variables
numDaysBack = str(365*5) #for daily you can go back multiple years worth, for daily you can only go back 90 day
myInterval = 'daily' # options are daily or hourly
theCoins = ['ethereum'] #can add more than one coin if you like
window_length = 14
mycom = 0.4
lower_macd_ema = 12
upper_macd_ema = 26
trigger_macd_ema = 9


def df_builder_clean(days, interval, coins, window, mycom, lower_macd_ema, upper_macd_ema, trigger_macd_ema):
    #manipulatable variables
    numDaysBack = days #for daily you can go back multiple years worth, for daily you can only go back 90 days
    myInterval = interval # options are daily or hourly
    theCoins = coins
    window_length = window

    #builds initial dataframe with ethereum as first market but just to log the dates we are working with
    geckoReq = 'https://api.coingecko.com/api/v3/coins/ethereum/market_chart?vs_currency=usd&days='+numDaysBack
    r = requests.get(geckoReq).json()
    ts = r['prices'][0][0]
    ts = ts/1000
    HistPricesList = []
    for i in range(len(r['prices'])):
        currentUnix = r['prices'][i][0]
        price = r['prices'][i][1]
        currentUnix = currentUnix/1000
#         currentTS = datetime.datetime.fromtimestamp(currentUnix).strftime("%d-%m-%Y %H:%M:%S") #adding dd-mm-
        currentTS = datetime.datetime.fromtimestamp(currentUnix).strftime("%m-%d-%Y") #just the date mm-dd-yyy
        HistPricesList.append([currentTS])
    global df
    df = pd.DataFrame(HistPricesList, columns = ['date'])


    #looping through each coin and adding in all data points and input variables
    for coin in theCoins:
        price_data(coin)
        add_ewm(coin, mycom)
        add_rsi(coin, window_length)
        add_macd(coin, lower_macd_ema, upper_macd_ema, trigger_macd_ema)
        print('just added: ', coin)
    df['date'] = pd.to_datetime(df['date'])
    df = df.set_index('date')
#     print('r')
#     display(r)
    return df


def price_data(coin):
    global df
    # geckoReq = 'https://api.coingecko.com/api/v3/coins/ethereum/market_chart?vs_currency=usd&days=60&interva
    geckoReq = 'https://api.coingecko.com/api/v3/coins/'+coin+'/market_chart?vs_currency=usd&days='+numDaysBack

    r = requests.get(geckoReq).json()
    ts = r['prices'][0][0]
    # print(ts)
    ts = ts/1000
    print(datetime.datetime.fromtimestamp(ts).strftime("%m-%d-%Y"))
    # print('prices length',len(r['prices']))
    HistPricesList = []
    for i in range(len(r['prices'])):
        currentUnix = r['prices'][i][0]
        price = r['prices'][i][1]
        volume = r['total_volumes'][i][1]
        currentUnix = currentUnix/1000
        currentTS = datetime.datetime.fromtimestamp(currentUnix).strftime("%m-%d-%Y")
    #     print('price: ', price, 'TS: ', currentTS)
        HistPricesList.append([currentTS, price, volume])
    #     currentUnix = currentTS
#         print(HistPricesList)
    dfCoin = pd.DataFrame(HistPricesList, columns = ['date', coin, coin+'_volume'])
#     print('dfCoin')
#     display(dfCoin)
    df = pd.merge(df, dfCoin[coin], left_on = df['date'], right_on=dfCoin['date']).drop(['key_0'], axis = 1)
    df = pd.merge(df, dfCoin[coin+'_volume'], left_on = df['date'], right_on=dfCoin['date']).drop(['key_0'], ax

def add_ewm(coin, mycom):
    df[coin+'_ewm'] = df[coin].ewm(com=mycom).mean()

def add_rsi(coin, window_length):
    global df
    df['diff'] = df[coin].diff(1)
    df['gain'] = df['diff'].clip(lower=0).round(2)
    df['loss'] = df['diff'].clip(upper=0).abs().round(2)

    # Get initial Averages
    df['avg_gain'] = df['gain'].rolling(window=window_length, min_periods=window_length).mean()[:window_length+
    df['avg_loss'] = df['loss'].rolling(window=window_length, min_periods=window_length).mean()[:window_length+

    # Get WMS averages
    # Average Gains
    for i, row in enumerate(df['avg_gain'].iloc[window_length+1:]):
        df['avg_gain'].iloc[i + window_length + 1] =\
            (df['avg_gain'].iloc[i + window_length] *
             (window_length - 1) +
             df['gain'].iloc[i + window_length + 1])\
            / window_length
    # Average Losses
    for i, row in enumerate(df['avg_loss'].iloc[window_length+1:]):
        df['avg_loss'].iloc[i + window_length + 1] =\
            (df['avg_loss'].iloc[i + window_length] *
             (window_length - 1) +
             df['loss'].iloc[i + window_length + 1])\
            / window_length

    df['rs'] = df['avg_gain'] / df['avg_loss']

    df['rsi'] = 100 - (100 / (1.0 + df['rs']))
    df = pd.DataFrame(df)
    df = df.drop(['gain', 'loss', 'avg_loss', 'avg_gain', 'rs'], axis = 1)

    #renaming diff and rsi columns
    dict = {'diff': coin+'_diff',
        'rsi': coin+'_rsi'}
    df.rename(columns=dict,
              inplace=True)

def add_macd(coin, lower_macd_ema, upper_macd_ema, trigger_macd_ema):
    global df
    #get ema for lower
    k = df[coin].ewm(span=lower_macd_ema, adjust = False, min_periods = lower_macd_ema).mean()

    #get ema for upper
    d = df[coin].ewm(span=upper_macd_ema, adjust=False, min_periods=upper_macd_ema).mean()

    macd = k-d

    # Get the 9-Day EMA of the MACD for the Trigger line
    macd_s = macd.ewm(span=trigger_macd_ema, adjust=False, min_periods=trigger_macd_ema).mean()
    # Calculate the difference between the MACD - Trigger for the Convergence/Divergence value
    macd_h = macd - macd_s
    # Add all of our new values for the MACD to the dataframe
    df['macd'] = df.index.map(macd)
    df['macd_h'] = df.index.map(macd_h)
    df['macd_s'] = df.index.map(macd_s)



mydf = pd.DataFrame(df_builder_clean(numDaysBack, myInterval, theCoins, window_length, mycom, lower_macd_ema, u
display(mydf)
```

```
07-22-2017
C:\Users\fooba\anaconda3\lib\site-packages\pandas\core\indexing.py:1732: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  self._setitem_single_block(indexer, value, name)
just added:  ethereum
```

date	ethereum	ethereum_volume	ethereum_ewm	ethereum_diff	ethereum_rsi	macd	macd_h	macd_s
2017-07-22	226.718599	2.044678e+08	226.718599	NaN	NaN	NaN	NaN	NaN
2017-07-23	224.913407	1.442842e+08	225.314561	-1.805192	NaN	NaN	NaN	NaN
2017-07-24	206.810505	3.580211e+08	211.781744	-18.102902	NaN	NaN	NaN	NaN
2017-07-25	200.328556	2.912707e+08	203.546013	-6.481949	NaN	NaN	NaN	NaN
2017-07-26	203.189322	1.557632e+08	203.290748	2.860766	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...
2022-07-20	1488.975794	2.168190e+10	1492.140810	0.000000	61.868846	69.262089	41.973093	27.288996
2022-07-20	1527.413931	2.172813e+10	1517.335896	38.438137	64.019527	73.845298	37.245041	36.600256
2022-07-20	1527.413931	2.168190e+10	1524.534492	0.000000	64.019527	76.594589	31.995466	44.599123
2022-07-20	1488.975794	2.172813e+10	1499.135422	-38.438137	60.088919	74.809429	24.168245	50.641184
2022-07-20	1488.975794	2.168190e+10	1491.878545	0.000000	60.088919	72.558271	17.533669	55.024601

1832 rows × 8 columns

In [8]:

```python
mydf.dropna(inplace=True)
myCols = ['ethereum', 'ethereum_volume', 'ethereum_ewm', 'ethereum_rsi', 'macd', 'macd_h', 'macd_s', 'ethereum_
```

In [9]:

```python
my_data = mydf[myCols]
# display(my_data.tail(10))
my_data.drop(my_data.tail(10).index,
        inplace = True)
display(my_data)
my_data.corr()
```

```
C:\Users\fooba\anaconda3\lib\site-packages\pandas\core\frame.py:4906: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  return super().drop(
```

date	ethereum	ethereum_volume	ethereum_ewm	ethereum_rsi	macd	macd_h	macd_s	ethereum_diff
2017-08-24	329.025281	4.519507e+08	326.457443	69.654547	20.372867	0.419856	19.953011	7.239983
2017-08-25	329.865783	3.338393e+08	328.891971	69.894479	20.626894	0.539106	20.087788	0.840502
2017-08-26	343.341337	3.700521e+08	339.212947	73.513669	21.665826	1.262431	20.403396	13.475555
2017-08-27	344.201131	5.198744e+08	342.775936	73.730643	22.301488	1.518474	20.783014	0.859794
2017-08-28	366.809590	7.671665e+08	359.942832	78.676407	24.348890	2.852701	21.496189	22.608458
...	...	...	...	...	...	...	...	...
2022-07-13	1112.920783	1.793689e+10	1098.951561	41.988562	-67.788266	17.479501	-85.267768	72.123637
2022-07-14	1191.130837	1.647422e+10	1164.793901	48.359970	-58.631347	21.309137	-79.940484	78.210054
2022-07-15	1234.099139	1.681719e+10	1214.297642	51.511001	-47.361288	26.063357	-73.424645	42.968302
2022-07-16	1355.045640	1.902583e+10	1314.831927	59.079833	-28.343586	36.064847	-64.408433	120.946501
2022-07-17	1344.720284	1.579765e+10	1336.180753	58.243628	-13.944348	40.371268	-54.315616	-10.325356

1789 rows × 8 columns

Out[9]:

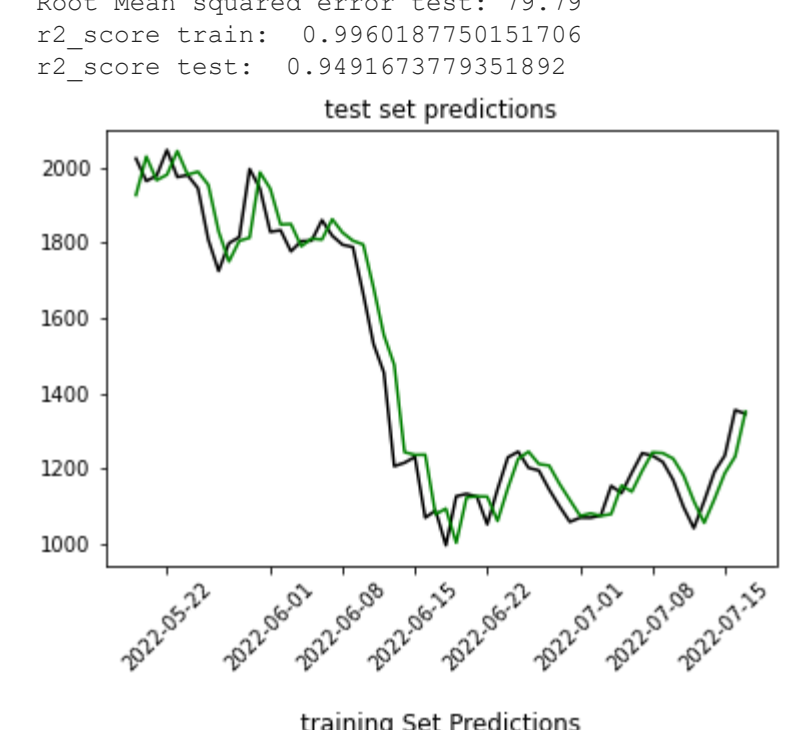
	ethereum	ethereum_volume	ethereum_ewm	ethereum_rsi	macd	macd_h	macd_s	ethereum_diff
ethereum	1.000000	0.532233	0.999824	0.091851	0.185069	-0.002743	0.196870	0.030610
ethereum_volume	0.532233	1.000000	0.533278	0.235115	0.278609	-0.115095	0.330872	-0.041297
ethereum_ewm	0.999824	0.533278	1.000000	0.086098	0.182843	-0.008451	0.196287	0.012615
ethereum_rsi	0.091851	0.235115	0.086098	1.000000	0.604187	0.394895	0.517163	0.233843
macd	0.185069	0.278609	0.182843	0.604187	1.000000	0.331733	0.956027	0.073065
macd_h	-0.002743	-0.115095	-0.008451	0.394895	0.331733	1.000000	0.040476	0.191527
macd_s	0.196870	0.330872	0.196287	0.517163	0.956027	0.040476	1.000000	0.017845
ethereum_diff	0.030610	-0.041297	0.012615	0.233843	0.073065	0.191527	0.017845	1.000000

Linear Regression Calculated Features

In [10]:

```python
# X = my_data[['ethereum_volume', 'ethereum_ewm', 'ethereum_rsi', 'macd', 'ethereum_diff']]
X_raw = my_data[['ethereum_volume', 'ethereum_ewm', 'ethereum_rsi', 'macd', 'macd_h', 'macd_s', 'ethereum_diff'
y = my_data['ethereum']
scaler=StandardScaler()
# Drop first row of prices
y.drop(index=y.index[0],
       axis=0,
       inplace=True)
#dropping last row of features
X_raw.drop(index=X_raw.index[-1],
       axis=0,
       inplace=True)
X=scaler.fit_transform(X_raw)
display(X)
# X = X.shift(1, freq='d')
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .1, random_state=42)
X_train = X[:-60]
X_test = X[-60:]
y_train = y[:-60]
y_test = y[-60:]
print('X_train: ')
print(X_train.shape)
print('X_test: ')
print(X_test.shape)
print('y_train: ')
print(y_train.shape)
print('y_test: ')
print(y_test.shape)


# print('linear regression with no scaling')
# Create linear regression object
regr = linear_model.LinearRegression()

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

y_pred_train = regr.predict(X_train)
# Make predictions using the testing set
y_pred_test = regr.predict(X_test)

# The mean squared error
print("Root Mean squared error train: %.2f" % math.sqrt(mean_squared_error(y_train, y_pred_train)))
print("Root Mean squared error test: %.2f" % math.sqrt(mean_squared_error(y_test, y_pred_test)))

print('r2_score train: ', r2_score(y_train, y_pred_train))
print('r2_score test: ', r2_score(y_test, y_pred_test))

y_testc = pd.DataFrame(y_test)
# print(y_check1)
y_testc['pred_linear_reg']= y_pred_test
# print(y_check1)
plt.plot(y_testc.index,y_testc['ethereum'], color = 'black')
plt.plot(y_testc.index,y_testc['pred_linear_reg'], color = 'green')
plt.xticks(rotation=45)
plt.title('test set predictions')
plt.show()

y_trainc = pd.DataFrame(y_train)
# print(y_check1)
y_trainc['pred_linear_reg']= y_pred_train
# print(y_check1)
plt.plot(y_trainc.index,y_trainc['ethereum'], color = 'black')
plt.plot(y_trainc.index,y_trainc['pred_linear_reg'], color = 'green')
plt.xticks(rotation=45)
plt.title('training Set Predictions')
plt.show()
```

```
C:\Users\fooba\anaconda3\lib\site-packages\pandas\core\generic.py:4153: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  self._update_inplace(obj)
C:\Users\fooba\anaconda3\lib\site-packages\pandas\core\frame.py:4906: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  return super().drop(
array([[-0.90048046, -0.60472109,  1.30264522, ...,  0.02338732,
         0.19060082,  0.08494027],
       [-0.90902945, -0.60273321,  1.31964148, ...,  0.02797176,
         0.19221073,  0.00334829],
       [-0.90640834, -0.59430577,  1.57601716, ...,  0.05577918,
         0.19598067,  0.16444245],
       ...,
       [ 0.2592226 ,  0.07981032, -0.205817  , ...,  0.82645283,
        -1.00262915,  0.98979317],
       [ 0.28404708,  0.12023188,  0.01739533, ...,  1.0092236 ,
        -0.92479731,  0.54046853],
       [ 0.44390971,  0.20232169,  0.55355517, ...,  1.39371993,
        -0.81709847,  1.53467352]])
X_train:
(1728, 7)
X_test:
(60, 7)
y_train:
(1728,)
y_test:
(60,)
Root Mean squared error train: 78.41
Root Mean squared error test: 79.79
r2_score train:  0.9960187750151706
r2_score test:  0.9491673779351892
```

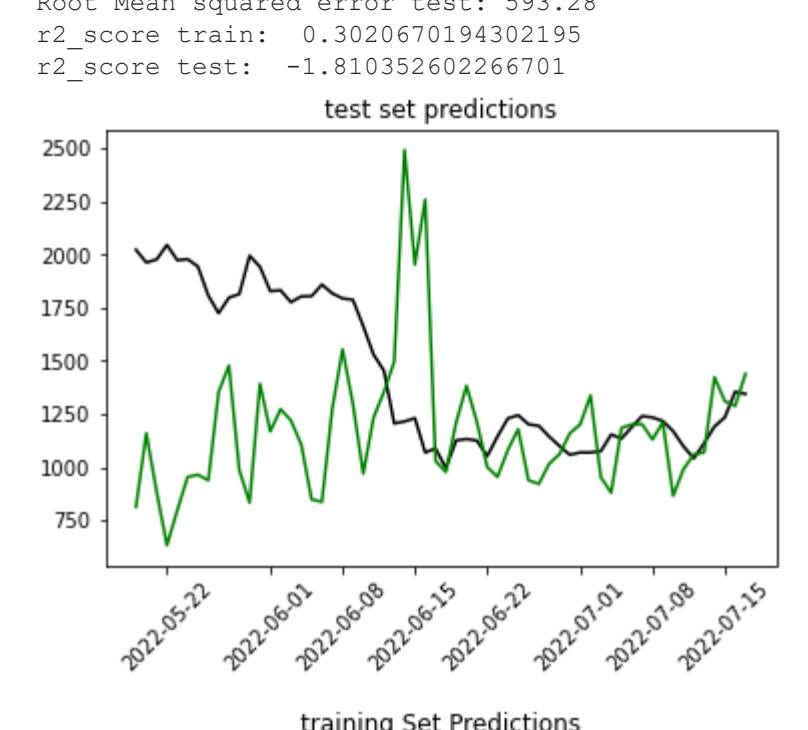


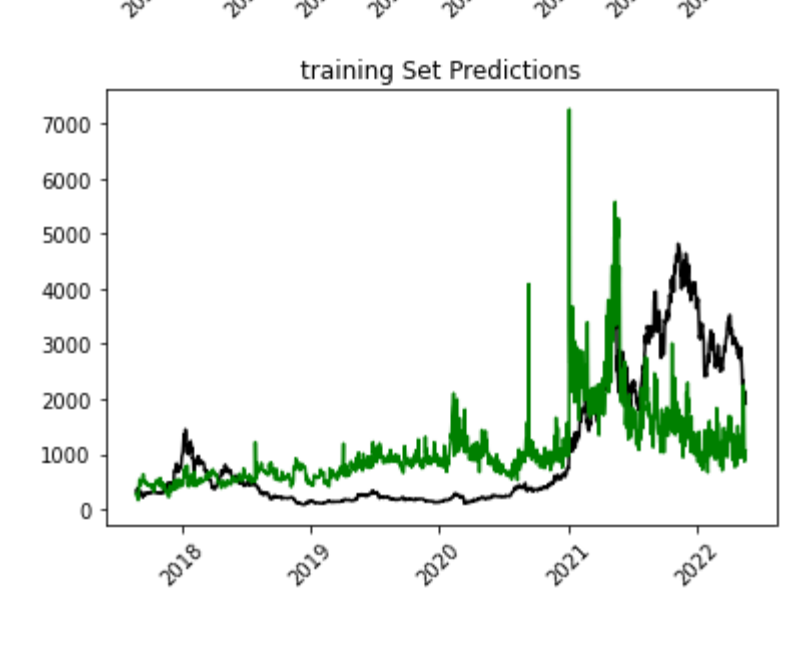
Dropping ewm for Linear Regression

In [11]:

```python
# X = my_data[['ethereum_volume', 'ethereum_ewm', 'ethereum_rsi', 'macd', 'ethereum_diff']]
X_raw = my_data[['ethereum_volume', 'ethereum_rsi', 'macd', 'macd_h', 'macd_s', 'ethereum_diff']]
y = my_data['ethereum']
scaler=StandardScaler()
# Drop first row of prices
y.drop(index=y.index[0],
       axis=0,
       inplace=True)
#dropping last row of features
X_raw.drop(index=X_raw.index[-1],
       axis=0,
       inplace=True)
X=scaler.fit_transform(X_raw)
display(X)
# X = X.shift(1, freq='d')
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .1, random_state=42)
X_train = X[:-60]
X_test = X[-60:]
y_train = y[:-60]
y_test = y[-60:]
print('X_train: ')
print(X_train.shape)
print('X_test: ')
print(X_test.shape)
print('y_train: ')
print(y_train.shape)
print('y_test: ')
print(y_test.shape)


# print('linear regression with no scaling')
# Create linear regression object
regr = linear_model.LinearRegression()

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

y_pred_train = regr.predict(X_train)
# Make predictions using the testing set
y_pred_test = regr.predict(X_test)

# The mean squared error
print("Root Mean squared error train: %.2f" % math.sqrt(mean_squared_error(y_train, y_pred_train)))
print("Root Mean squared error test: %.2f" % math.sqrt(mean_squared_error(y_test, y_pred_test)))

print('r2_score train: ', r2_score(y_train, y_pred_train))
print('r2_score test: ', r2_score(y_test, y_pred_test))

y_testc = pd.DataFrame(y_test)
# print(y_check1)
y_testc['pred_linear_reg']= y_pred_test
# print(y_check1)
plt.plot(y_testc.index,y_testc['ethereum'], color = 'black')
plt.plot(y_testc.index,y_testc['pred_linear_reg'], color = 'green')
plt.xticks(rotation=45)
plt.title('test set predictions')
plt.show()

y_trainc = pd.DataFrame(y_train)
# print(y_check1)
y_trainc['pred_linear_reg']= y_pred_train
# print(y_check1)
plt.plot(y_trainc.index,y_trainc['ethereum'], color = 'black')
plt.plot(y_trainc.index,y_trainc['pred_linear_reg'], color = 'green')
plt.xticks(rotation=45)
plt.title('training Set Predictions')
plt.show()
```

```
C:\Users\fooba\anaconda3\lib\site-packages\pandas\core\generic.py:4153: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  self._update_inplace(obj)
C:\Users\fooba\anaconda3\lib\site-packages\pandas\core\frame.py:4906: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  return super().drop(
array([[-0.90048046,  1.30264522,  0.18679283,  0.02338732,  0.19060082,
         0.08494027],
       [-0.90902945,  1.31964148,  0.18965735,  0.02797176,  0.19221073,
         0.00334829],
       [-0.90640834,  1.57601716,  0.20137282,  0.05577918,  0.19598067,
         0.16444245],
       ...,
       [ 0.2592226 , -0.205817  , -0.70409452,  0.82645283, -1.00262915,
         0.98979317],
       [ 0.28404708,  0.01739533, -0.57700823,  1.0092236 , -0.92479731,
         0.54046853],
       [ 0.44390971,  0.55355517, -0.362556  ,  1.39371993, -0.81709847,
         1.53467352]])
X_train:
(1728, 6)
X_test:
(60, 6)
y_train:
(1728,)
y_test:
(60,)
Root Mean squared error train: 1038.12
Root Mean squared error test: 593.28
r2_score train:  0.3020670194302195
r2_score test:  -1.810352602266701
```



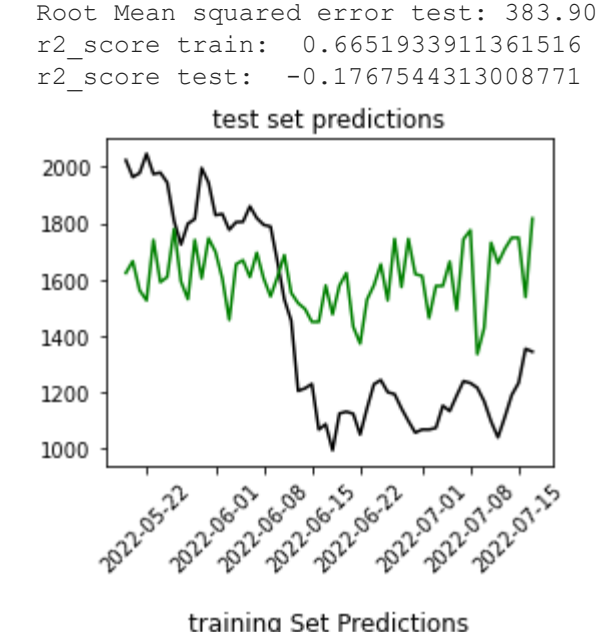


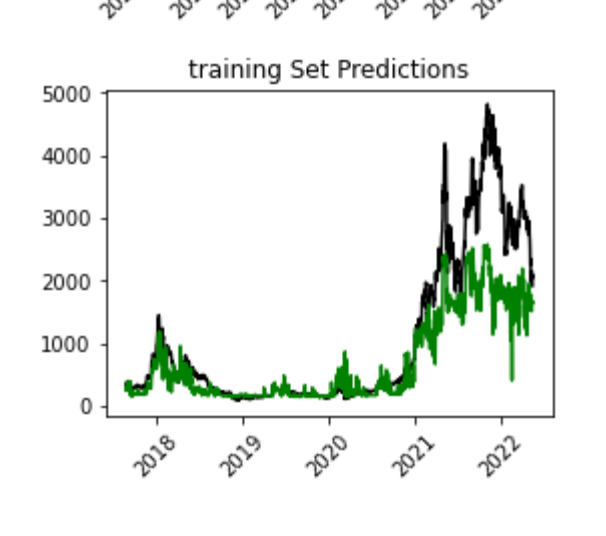
XGBOOST Calculated Features

In [12]:

```python
# X = my_data[['ethereum_volume', 'ethereum_ewm', 'ethereum_rsi', 'macd', 'ethereum_diff']]
X_raw = my_data[['ethereum_volume', 'ethereum_ewm', 'ethereum_rsi', 'macd', 'macd_h', 'macd_s', 'ethereum_diff'
y = my_data['ethereum']
scaler=StandardScaler()
# Drop first row of prices
y.drop(index=y.index[0],
       axis=0,
       inplace=True)
#dropping last row of features
X_raw.drop(index=X_raw.index[-1],
       axis=0,
       inplace=True)
X=scaler.fit_transform(X_raw)
display(X)
# X = X.shift(1, freq='d')
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .1, random_state=42)
X_train = X[:-60]
X_test = X[-60:]
y_train = y[:-60]
y_test = y[-60:]
print('X_train: ')
print(X_train.shape)
print('X_test: ')
print(X_test.shape)
print('y_train: ')
print(y_train.shape)
print('y_test: ')
print(y_test.shape)

xg_reg=xgb.XGBRegressor(objective='reg:squarederror',colsample_bytree=0.3, learning_rate=0.1,
                        max_depth=5,alpha=10,n_estimators=10)
xg_reg.fit(X_train,y_train)
y_pred_test=xg_reg.predict(X_test)
y_pred_train=xg_reg.predict(X_train)
# test_rmse=np.sqrt(mean_squared_error(y_test,y_pred_test))
# train_rmse=np.sqrt(mean_squared_error(y_train,y_pred_train))

print("Root Mean squared error train: %.2f" % math.sqrt(mean_squared_error(y_train, y_pred_train)))
print("Root Mean squared error test: %.2f" % math.sqrt(mean_squared_error(y_test, y_pred_test)))

print('r2_score train: ', r2_score(y_train, y_pred_train))
print('r2_score test: ', r2_score(y_test, y_pred_test))

y_testc = pd.DataFrame(y_test)
# print(y_check1)
y_testc['pred_linear_reg']= y_pred_test
# print(y_check1)
figure(figsize=(4, 3))
plt.plot(y_testc.index,y_testc['ethereum'], color = 'black')
plt.plot(y_testc.index,y_testc['pred_linear_reg'], color = 'green')
plt.xticks(rotation=45)
plt.title('test set predictions')
plt.show()


y_trainc = pd.DataFrame(y_train)
# print(y_check1)
y_trainc['pred_linear_reg']= y_pred_train
# print(y_check1)
figure(figsize=(4, 3))
plt.plot(y_trainc.index,y_trainc['ethereum'], color = 'black')
plt.plot(y_trainc.index,y_trainc['pred_linear_reg'], color = 'green')
plt.xticks(rotation=45)
plt.title('training Set Predictions')

plt.show()
```

```
C:\Users\fooba\anaconda3\lib\site-packages\pandas\core\generic.py:4153: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  self._update_inplace(obj)
C:\Users\fooba\anaconda3\lib\site-packages\pandas\core\frame.py:4906: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  return super().drop(
array([[-0.90048046, -0.60472109,  1.30264522, ...,  0.02338732,
         0.19060082,  0.08494027],
       [-0.90902945, -0.60273321,  1.31964148, ...,  0.02797176,
         0.19221073,  0.00334829],
       [-0.90640834, -0.59430577,  1.57601716, ...,  0.05577918,
         0.19598067,  0.16444245],
       ...,
       [ 0.2592226 ,  0.07981032, -0.205817  , ...,  0.82645283,
        -1.00262915,  0.98979317],
       [ 0.28404708,  0.12023188,  0.01739533, ...,  1.0092236 ,
        -0.92479731,  0.54046853],
       [ 0.44390971,  0.20232169,  0.55355517, ...,  1.39371993,
        -0.81709847,  1.53467352]])
X_train:
(1728, 7)
X_test:
(60, 7)
y_train:
(1728,)
y_test:
(60,)
Root Mean squared error train: 719.02
Root Mean squared error test: 383.90
r2_score train:  0.6651933911361516
r2_score test:  -0.1767544313008771
```





XGBoost Grid Search Hyperparameter Tuning on Calculated Features Set

In [13]:

```python
parameters = {
    'max_depth': range (2, 10, 1),
    'booster': ['gbtree', 'gblinear'],
#     'n_estimators': range(0, 1000, 50),
    'n_estimators': [10,50,60,200,500,1000],
    'learning_rate': [0.1, 0.01, 0.05]
#     'eval_metric': ['rmse', 'mae', 'mape']
}
grid_search = GridSearchCV(
    estimator=xg_reg,
    param_grid=parameters,
#     scoring = 'roc_auc',
    n_jobs = 10,
    cv = 5,
    verbose=True
)

grid_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 288 candidates, totalling 1440 fits
```

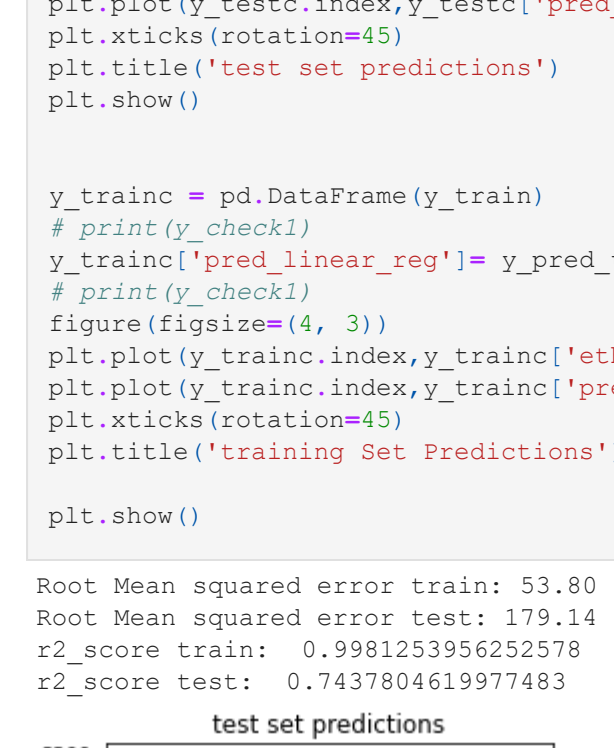
Out[13]:

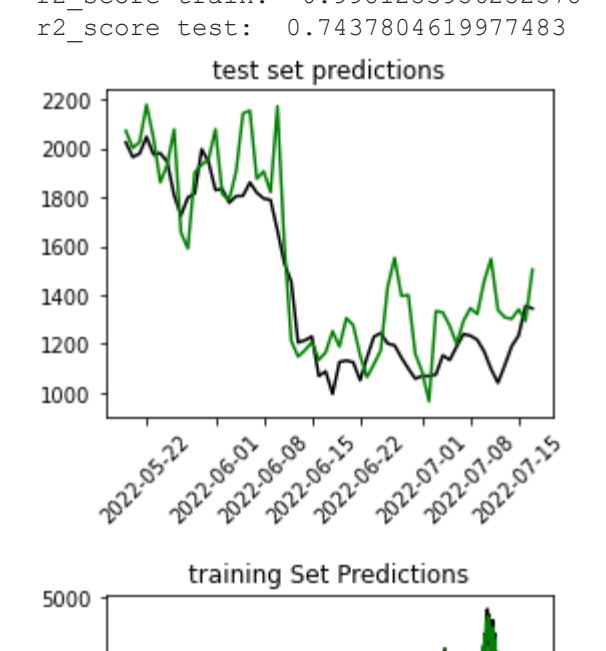
```
GridSearchCV(cv=5,
             estimator=XGBRegressor(alpha=10, base_score=0.5, booster='gbtree',
                                    callbacks=None, colsample_bylevel=1,
                                    colsample_bynode=1, colsample_bytree=0.3,
                                    early_stopping_rounds=None,
                                    enable_categorical=False, eval_metric=None,
                                    gamma=0, gpu_id=-1, grow_policy='depthwise',
                                    importance_type=None,
                                    interaction_constraints='',
                                    learning_rate=0.1, max_bin=256,
                                    max_...ta_step=0,
                                    max_depth=5, max_leaves=0,
                                    min_child_weight=1, missing=nan,
                                    monotone_constraints='()', n_estimators=10,
                                    n_jobs=0, num_parallel_tree=1,
                                    predictor='auto', random_state=0,
                                    reg_alpha=10, ...),
             n_jobs=10,
             param_grid={'booster': ['gbtree', 'gblinear'],
                         'learning_rate': [0.1, 0.01, 0.05],
                         'max_depth': range(2, 10),
                         'n_estimators': [10, 50, 60, 200, 500, 1000]},
             verbose=True)
```

In [14]:

```python
grid_search.best_estimator_
grid_search.best_estimator_.fit(X_train, y_train)
y_pred_train = grid_search.best_estimator_.predict(X_train)

y_pred_test = grid_search.best_estimator_.predict(X_test)
# rmse=np.sqrt(mean_squared_error(y_test,y_pred))

# print('xgb rmse: ', rmse)

print("Root Mean squared error train: %.2f" % math.sqrt(mean_squared_error(y_train, y_pred_train)))
print("Root Mean squared error test: %.2f" % math.sqrt(mean_squared_error(y_test, y_pred_test)))

print('r2_score train: ', r2_score(y_train, y_pred_train))
print('r2_score test: ', r2_score(y_test, y_pred_test))

y_testc = pd.DataFrame(y_test)
# print(y_check1)
y_testc['pred_linear_reg']= y_pred_test
# print(y_check1)
figure(figsize=(4, 3))
plt.plot(y_testc.index,y_testc['ethereum'], color = 'black')
plt.plot(y_testc.index,y_testc['pred_linear_reg'], color = 'green')
plt.xticks(rotation=45)
plt.title('test set predictions')
plt.show()


y_trainc = pd.DataFrame(y_train)
# print(y_check1)
y_trainc['pred_linear_reg']= y_pred_train
# print(y_check1)
figure(figsize=(4, 3))
plt.plot(y_trainc.index,y_trainc['ethereum'], color = 'black')
plt.plot(y_trainc.index,y_trainc['pred_linear_reg'], color = 'green')
plt.xticks(rotation=45)
plt.title('training Set Predictions')

plt.show()
```

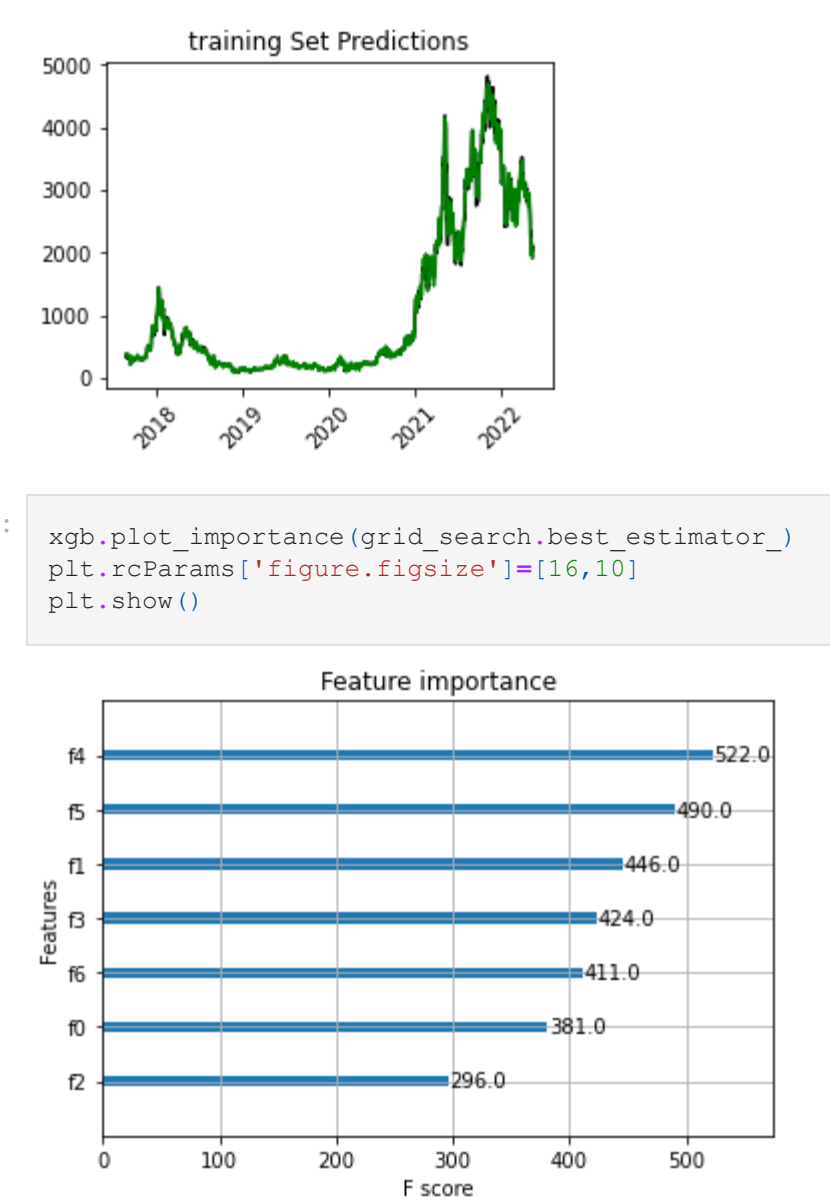
```
Root Mean squared error train: 53.80
Root Mean squared error test: 179.14
r2_score train:  0.9981253956252578
r2_score test:  0.7437804619977483
```





In [15]:

```python
xgb.plot_importance(grid_search.best_estimator_)
plt.rcParams['figure.figsize']=[16,10]
plt.show()
```



sliding window

train test split and create sliding window

In [16]:

```python
seq=list(mydf['ethereum'])
# display(seq)

window_size = 15
sliding_window=[]
for i in range(len(seq) - window_size + 1):
    sliding_window.append(seq[i: i + window_size])

# print(sliding_window)

sliding_df = pd.DataFrame(sliding_window)

sliding_df = sliding_df.drop(columns=[1,2,3,4,5])
display(sliding_df)
y = sliding_df[0]
X = sliding_df.drop(columns=[0])
X=scaler.fit_transform(X)
# print('X')
# display(X)
# print('y')
# display(y)
# X = X.shift(1, freq='d')
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .2, random_state=42)
X_train = X[:-360]
X_test = X[-360:]
y_train = y[:-360]
y_test = y[-360:]
print('X_train: ')
print(X_train.shape)
print('X_test: ')
print(X_test.shape)
print('y_train: ')
print(y_train.shape)
print('y_test: ')
print(y_test.shape)
```

	0	6	7	8	9	10	11	12	13	14
0	329.025281	382.934451	386.343057	346.398775	349.929240	299.717332	318.972804	333.206330	329.415663	304.475108
1	329.865783	386.343057	346.398775	349.929240	299.717332	318.972804	333.206330	329.415663	304.475108	299.694444
2	343.341337	346.398775	349.929240	299.717332	318.972804	333.206330	329.415663	304.475108	299.694444	292.779657
3	344.201131	349.929240	299.717332	318.972804	333.206330	329.415663	304.475108	299.694444	292.779657	296.475399
4	366.809590	299.717332	318.972804	333.206330	329.415663	304.475108	299.694444	292.779657	296.475399	290.586532
...	...	...	...	...	...	...	...	...	...	...
1780	1216.849707	1234.099139	1355.045640	1344.720284	1570.658959	1542.629821	1527.413931	1527.413931	1488.975794	1488.975794
1781	1169.012708	1355.045640	1344.720284	1570.658959	1542.629821	1527.413931	1527.413931	1488.975794	1488.975794	1527.413931
1782	1097.449438	1344.720284	1570.658959	1542.629821	1527.413931	1527.413931	1488.975794	1488.975794	1527.413931	1527.413931
1783	1040.797146	1570.658959	1542.629821	1527.413931	1527.413931	1488.975794	1488.975794	1527.413931	1527.413931	1488.975794
1784	1112.920783	1542.629821	1527.413931	1527.413931	1488.975794	1488.975794	1527.413931	1527.413931	1488.975794	1488.975794

1785 rows × 10 columns

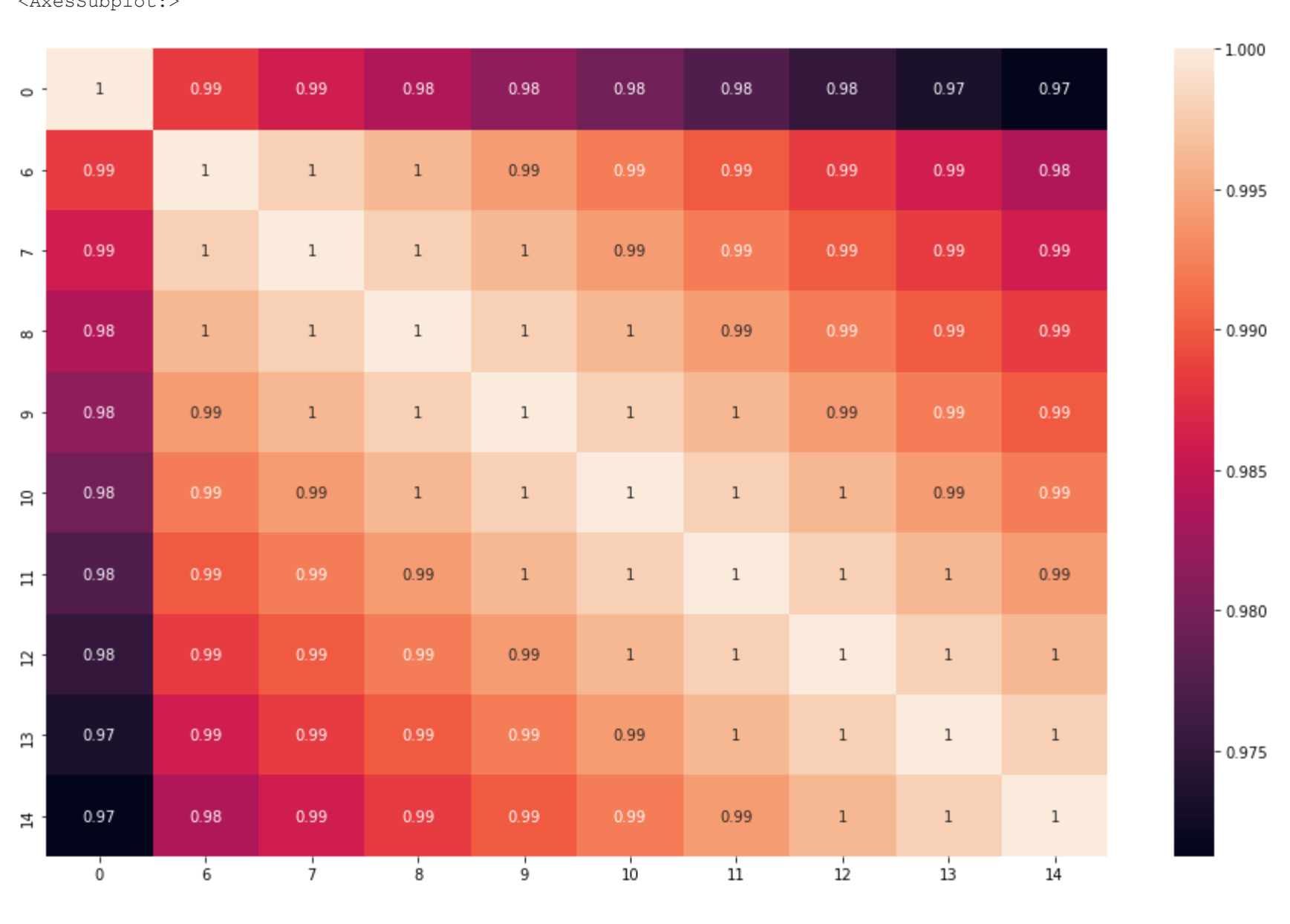
```
X_train:
(1425, 9)
X_test:
(360, 9)
y_train:
(1425,)
y_test:
(360,)
```

In [17]:

```python
sns.heatmap(sliding_df.corr(), annot=True)
```

Out[17]:

```
<AxesSubplot:>
```

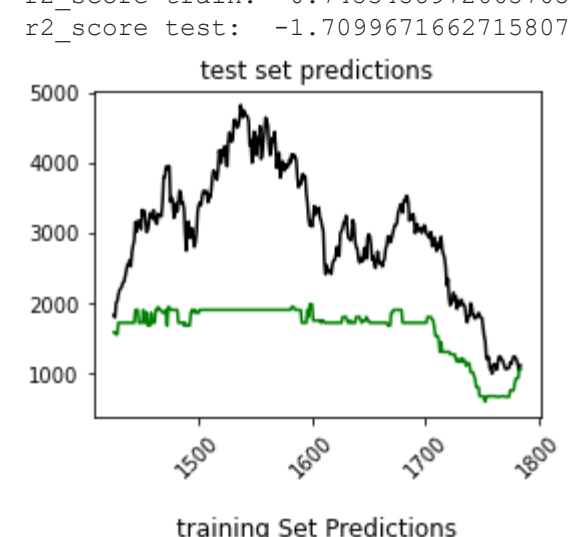


xgboost on sliding window

In [18]:

```python
xg_reg=xgb.XGBRegressor(objective='reg:squarederror',colsample_bytree=0.3, learning_rate=0.1,
                        max_depth=5,alpha=10,n_estimators=10)
xg_reg.fit(X_train,y_train)
y_pred_test=xg_reg.predict(X_test)
y_pred_train=xg_reg.predict(X_train)
# rmse=np.sqrt(mean_squared_error(y_test,preds))
# print(preds)
# print(y_check1)
# print('xgb rmse: ', rmse)

print("Root Mean squared error train: %.2f" % math.sqrt(mean_squared_error(y_train, y_pred_train)))
print("Root Mean squared error test: %.2f" % math.sqrt(mean_squared_error(y_test, y_pred_test)))

print('r2_score train: ', r2_score(y_train, y_pred_train))
print('r2_score test: ', r2_score(y_test, y_pred_test))


y_testc = pd.DataFrame(y_test)
# print(y_check1)
y_testc['pred_linear_reg']= y_pred_test
# print(y_check1)
figure(figsize=(4, 3))
plt.plot(y_testc.index,y_testc[0], color = 'black')
plt.plot(y_testc.index,y_testc['pred_linear_reg'], color = 'green')
plt.xticks(rotation=45)
plt.title('test set predictions')
plt.show()


y_trainc = pd.DataFrame(y_train)
# print(y_check1)
y_trainc['pred_linear_reg']= y_pred_train
# print(y_check1)
figure(figsize=(4, 3))
plt.plot(y_trainc.index,y_trainc[0], color = 'black')
plt.plot(y_trainc.index,y_trainc['pred_linear_reg'], color = 'green')
plt.xticks(rotation=45)
plt.title('training Set Predictions')

plt.show()
```

```
Root Mean squared error train: 346.77
Root Mean squared error test: 1519.32
r2_score train:  0.7453458972665705
r2_score test:  -1.7099671662715807
```



hyperparameter tuning on sliding windows feature set

In [19]:

```python
parameters = {
    'max_depth': range (2, 10, 1),
    'booster': ['gbtree', 'gblinear'],
#     'n_estimators': range(0, 1000, 50),
    'n_estimators': [10,50,60,200,500,1000],
    'learning_rate': [0.1, 0.01, 0.05]
#     'eval_metric': ['rmse', 'mae', 'mape']
}
grid_search = GridSearchCV(
    estimator=xg_reg,
    param_grid=parameters,
#     scoring = 'roc_auc',
    n_jobs = 10,
    cv = 5,
    verbose=True
)

grid_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 288 candidates, totalling 1440 fits
```

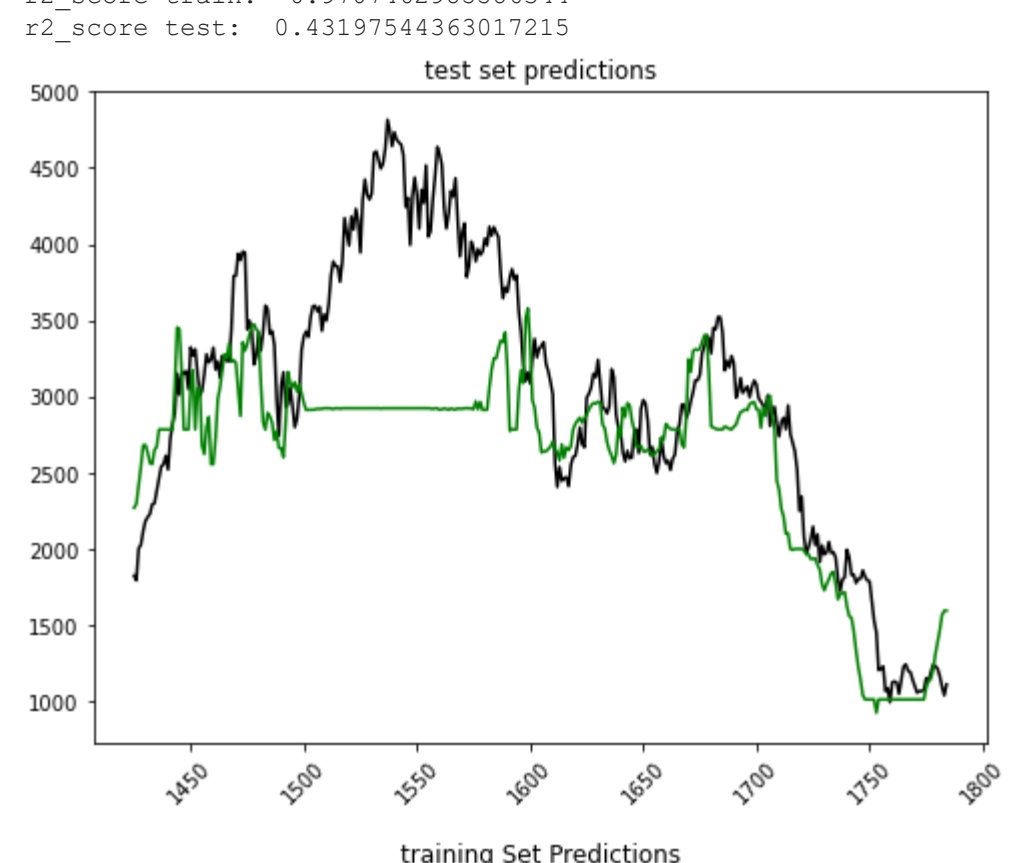
Out[19]:

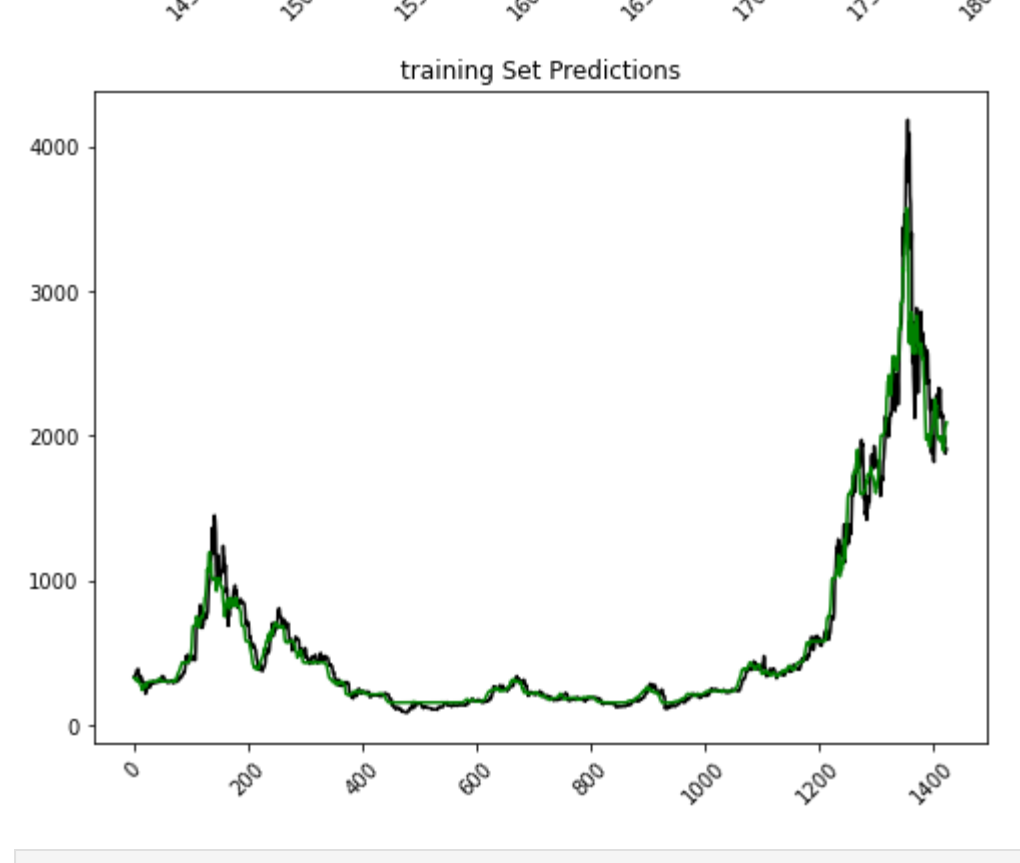
```
GridSearchCV(cv=5,
             estimator=XGBRegressor(alpha=10, base_score=0.5, booster='gbtree',
                                    callbacks=None, colsample_bylevel=1,
                                    colsample_bynode=1, colsample_bytree=0.3,
                                    early_stopping_rounds=None,
                                    enable_categorical=False, eval_metric=None,
                                    gamma=0, gpu_id=-1, grow_policy='depthwise',
                                    importance_type=None,
                                    interaction_constraints='',
                                    learning_rate=0.1, max_bin=256,
                                    max_...ta_step=0,
                                    max_depth=5, max_leaves=0,
                                    min_child_weight=1, missing=nan,
                                    monotone_constraints='()', n_estimators=10,
                                    n_jobs=0, num_parallel_tree=1,
                                    predictor='auto', random_state=0,
                                    reg_alpha=10, ...),
             n_jobs=10,
             param_grid={'booster': ['gbtree', 'gblinear'],
                         'learning_rate': [0.1, 0.01, 0.05],
                         'max_depth': range(2, 10),
                         'n_estimators': [10, 50, 60, 200, 500, 1000]},
             verbose=True)
```

In [20]:

```python
grid_search.best_estimator_
grid_search.best_estimator_.fit(X_train, y_train)
y_pred_test = grid_search.best_estimator_.predict(X_test)
y_pred_train = grid_search.best_estimator_.predict(X_train)
# rmse=np.sqrt(mean_squared_error(y_test,y_pred))

print("Root Mean squared error train: %.2f" % math.sqrt(mean_squared_error(y_train, y_pred_train)))
print("Root Mean squared error test: %.2f" % math.sqrt(mean_squared_error(y_test, y_pred_test)))

print('r2_score train: ', r2_score(y_train, y_pred_train))
print('r2_score test: ', r2_score(y_test, y_pred_test))

y_testc = pd.DataFrame(y_test)
# print(y_check1)
y_testc['pred_linear_reg']= y_pred_test
# print(y_check1)
figure(figsize=(8, 6))
plt.plot(y_testc.index,y_testc[0], color = 'black')
plt.plot(y_testc.index,y_testc['pred_linear_reg'], color = 'green')
plt.xticks(rotation=45)
plt.title('test set predictions')
plt.show()


y_trainc = pd.DataFrame(y_train)
# print(y_check1)
y_trainc['pred_linear_reg']= y_pred_train
# print(y_check1)
figure(figsize=(8, 6))
plt.plot(y_trainc.index,y_trainc[0], color = 'black')
plt.plot(y_trainc.index,y_trainc['pred_linear_reg'], color = 'green')
plt.xticks(rotation=45)
plt.title('training Set Predictions')

plt.show()
```

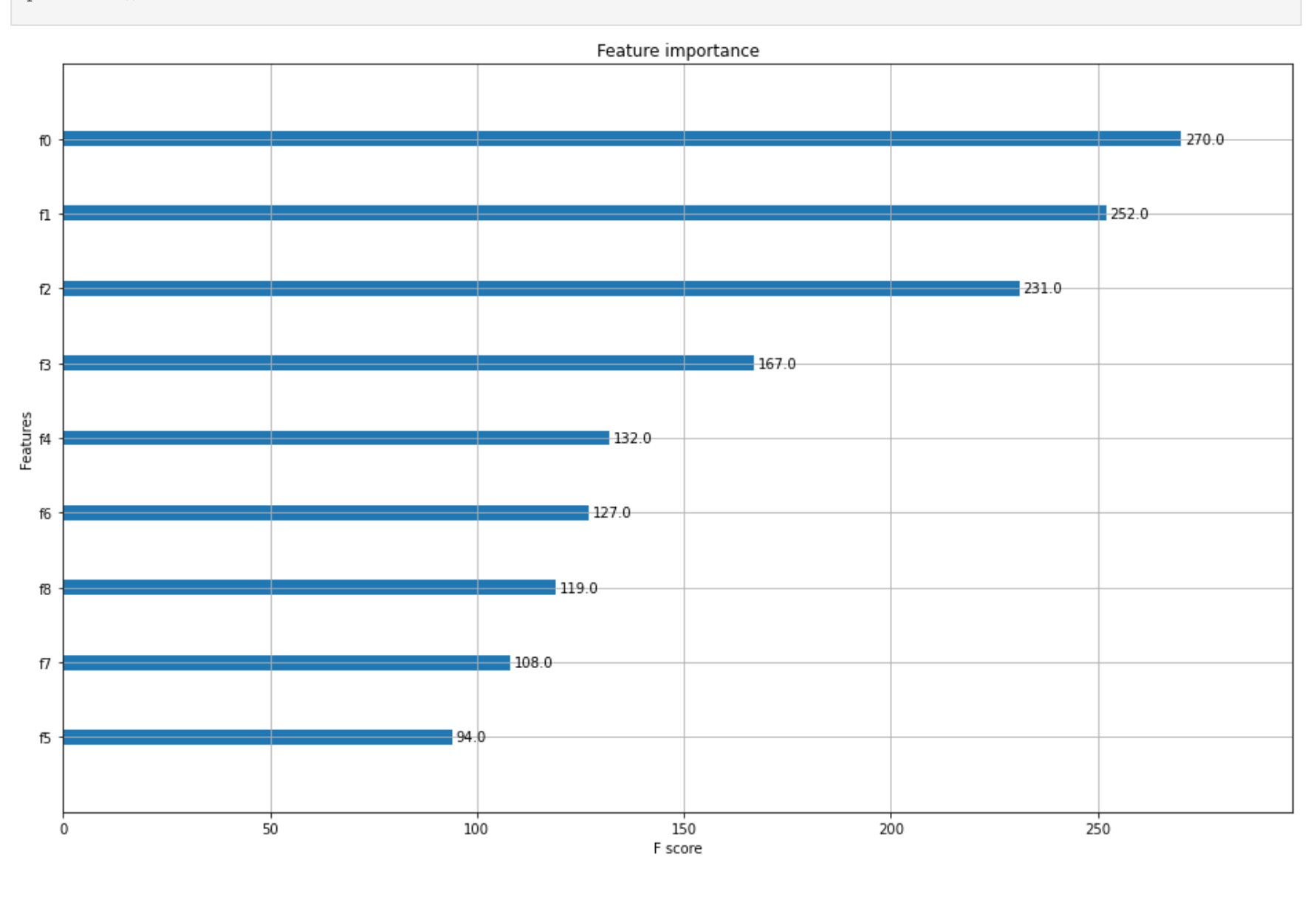
```
Root Mean squared error train: 117.53
Root Mean squared error test: 695.59
r2_score train:  0.9707462985586344
r2_score test:  0.43197544363017215
```





In [21]:

```python
xgb.plot_importance(grid_search.best_estimator_)
plt.rcParams['figure.figsize']=[16,10]
plt.show()
```



linear regression on sliding window

In [22]:

```python
# print('linear regression with no scaling')
# Create linear regression object
regr = linear_model.LinearRegression()

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

# Make predictions using the testing set
y_pred_test = regr.predict(X_test)
y_pred_train = regr.predict(X_train)
# The mean squared error
# print("Root Mean squared error: %.2f" % math.sqrt(mean_squared_error(y_test, y_pred)))

print("Root Mean squared error train: %.2f" % math.sqrt(mean_squared_error(y_train, y_pred_train)))
print("Root Mean squared error test: %.2f" % math.sqrt(mean_squared_error(y_test, y_pred_test)))

print('r2_score train: ', r2_score(y_train, y_pred_train))
print('r2_score test: ', r2_score(y_test, y_pred_test))
print('adjusted r2_score test:', 1-(1-r2_score(y_test, y_pred_test))*((len(X_test)-1)/(len(X_test)-len(X_test[0

y_testc = pd.DataFrame(y_test)
# print(y_check1)
y_testc['pred_linear_reg']= y_pred_test
# print(y_check1)
figure(figsize=(8, 4))
plt.plot(y_testc.index,y_testc[0], color = 'black')
plt.plot(y_testc.index,y_testc['pred_linear_reg'], color = 'green')
plt.xticks(rotation=45)
plt.title('test set predictions')
plt.show()


y_trainc = pd.DataFrame(y_train)
# print(y_check1)
y_trainc['pred_linear_reg']= y_pred_train
# print(y_check1)
figure(figsize=(8, 4))
plt.plot(y_trainc.index,y_trainc[0], color = 'black')
plt.plot(y_trainc.index,y_trainc['pred_linear_reg'], color = 'green')
plt.xticks(rotation=45)
plt.title('training Set Predictions')

plt.show()
```

```
Root Mean squared error train: 137.45
Root Mean squared error test: 328.49
r2_score train:  0.9599911252662963
r2_score test:  0.8733209124304121
adjusted r2_score test: 0.8700634501786226
```