In [41]:

```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 keras
# !pip install tensorflow
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, LSTM, Bidirectional

# !pip install graphviz
# import graphviz
# import chart_studio.tools as tls
```

Creating DataFrame

In [42]:

```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):
    #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

    #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-yyyy
        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))
display(mydf)
```

```
08-18-2017
just added:  ethereum
```

date	ethereum	ethereum_volume
2017-08-18	296.622090	5.537022e+08
2017-08-19	295.171577	3.428230e+08
2017-08-20	322.201220	1.743910e+09
2017-08-21	312.174471	8.983443e+08
2017-08-22	316.788920	4.664746e+08
...	...	...
2022-08-13	1982.411828	1.481675e+10
2022-08-14	1936.701164	1.217221e+10
2022-08-15	1908.277642	1.831087e+10
2022-08-16	1880.600101	1.424023e+10
2022-08-17	1841.674204	1.827104e+10

1826 rows × 2 columns

In [43]:

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

In [44]:

```python
my_data = mydf
display(my_data.tail(10))
my_data.drop(my_data.tail(10).index,
        inplace = True)
display(my_data)
```

date	ethereum	ethereum_volume
2022-08-08	1775.701356	1.556700e+10
2022-08-09	1698.966129	2.428438e+10
2022-08-10	1852.878555	2.169944e+10
2022-08-11	1881.427405	2.183707e+10
2022-08-12	1959.330925	1.584926e+10
2022-08-13	1982.411828	1.481675e+10
2022-08-14	1936.701164	1.217221e+10
2022-08-15	1908.277642	1.831087e+10
2022-08-16	1880.600101	1.424023e+10
2022-08-17	1841.674204	1.827104e+10

date	ethereum	ethereum_volume
2017-08-18	296.622090	5.537022e+08
2017-08-19	295.171577	3.428230e+08
2017-08-20	322.201220	1.743910e+09
2017-08-21	312.174471	8.983443e+08
2017-08-22	316.788920	4.664746e+08
...	...	...
2022-08-03	1621.341324	1.554854e+10
2022-08-04	1612.418907	1.407456e+10
2022-08-05	1725.382627	1.771748e+10
2022-08-06	1693.296592	1.123110e+10
2022-08-07	1699.006418	9.464716e+09

1816 rows × 2 columns

Train Test Split

60 day sliding window predictions

In [45]:

```python
seq=list(mydf['ethereum'])
# display(seq)
scaler = StandardScaler()
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[14]
X = sliding_df.drop(columns=[14])
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	296.622090	329.025281	329.865783	343.341337	344.201131	366.809590	378.906331	382.934451	386.343057	346.398775
1	295.171577	329.865783	343.341337	344.201131	366.809590	378.906331	382.934451	386.343057	346.398775	349.929240
2	322.201220	343.341337	344.201131	366.809590	378.906331	382.934451	386.343057	346.398775	349.929240	299.717332
3	312.174471	344.201131	366.809590	378.906331	382.934451	386.343057	346.398775	349.929240	299.717332	318.972804
4	316.788920	366.809590	378.906331	382.934451	386.343057	346.398775	349.929240	299.717332	318.972804	333.206330
...	...	...	...	...	...	...	...	...	...	...
1797	1527.413931	1438.988038	1636.949181	1723.548521	1739.042177	1696.785098	1682.011374	1636.018556	1638.378683	1621.341324
1798	1576.825239	1636.949181	1723.548521	1739.042177	1696.785098	1682.011374	1636.018556	1638.378683	1621.341324	1612.418907
1799	1536.124922	1723.548521	1739.042177	1696.785098	1682.011374	1636.018556	1638.378683	1621.341324	1612.418907	1725.382627
1800	1552.496489	1739.042177	1696.785098	1682.011374	1636.018556	1638.378683	1621.341324	1612.418907	1725.382627	1693.296592
1801	1601.207967	1696.785098	1682.011374	1636.018556	1638.378683	1621.341324	1612.418907	1725.382627	1693.296592	1699.006418

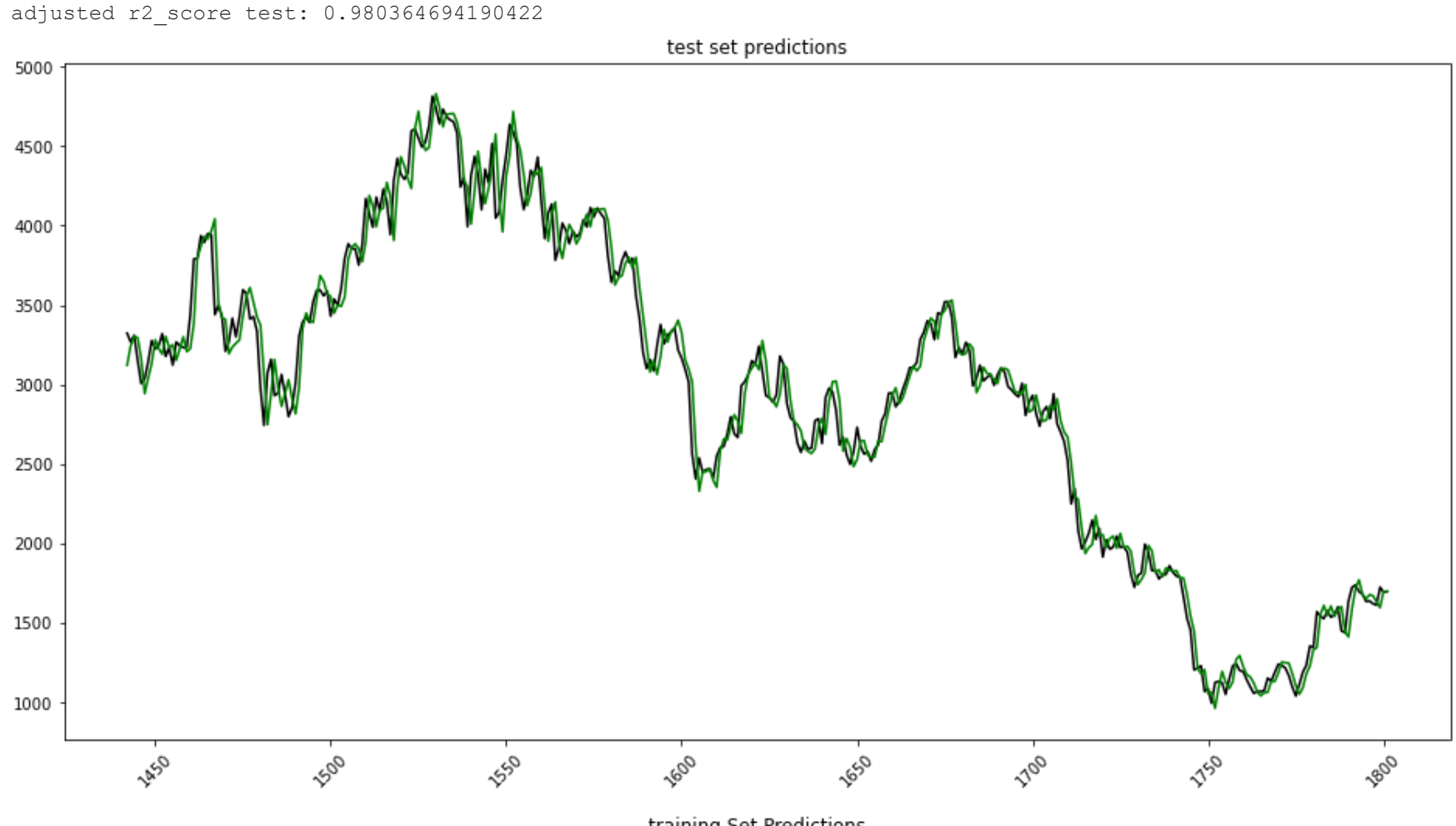
1802 rows × 10 columns

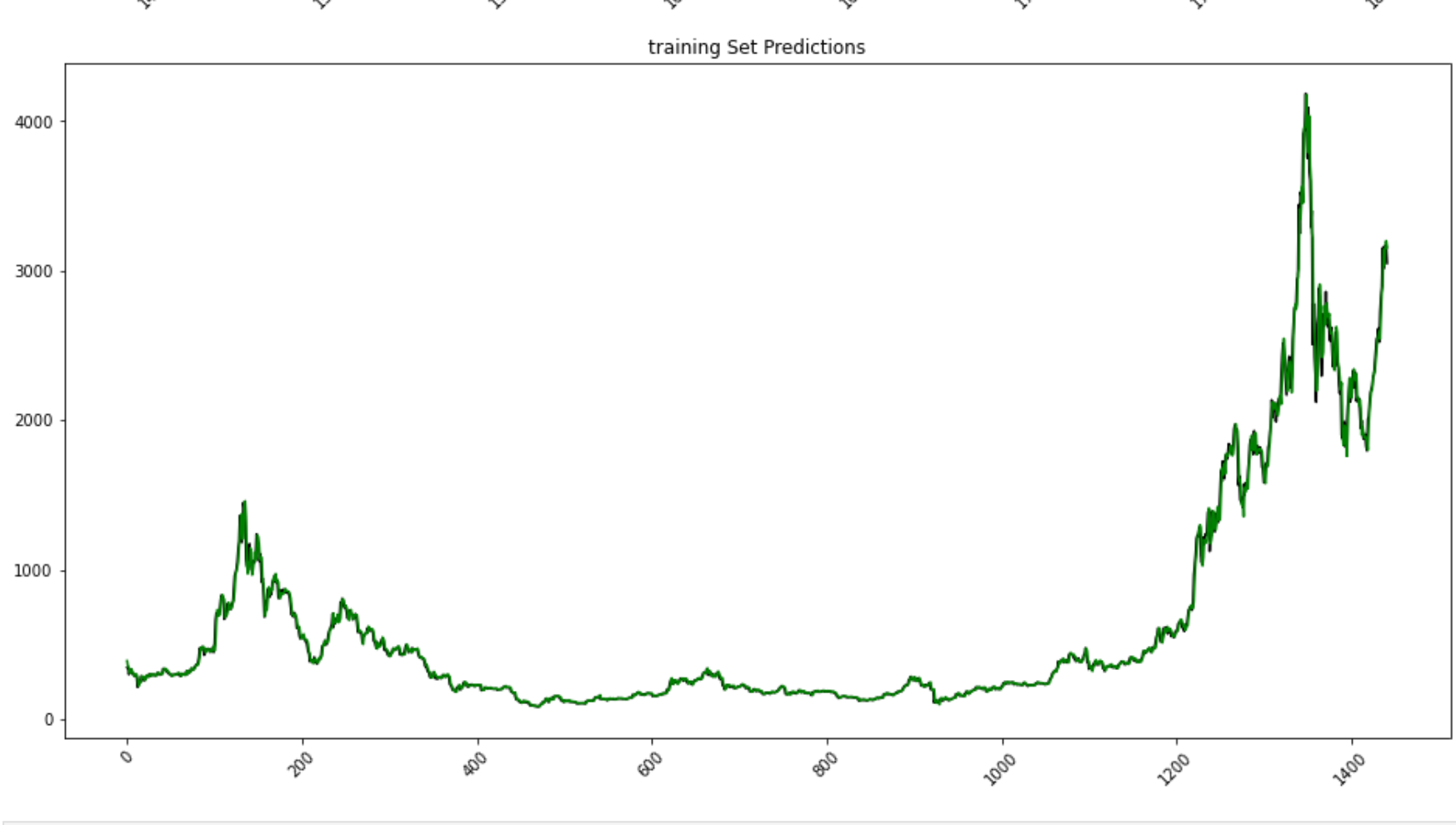
```
X_train:
(1442, 9)
X_test:
(360, 9)
y_train:
(1442,)
y_test:
(360,)
```

In [46]:

```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[(

y_testc = pd.DataFrame(y_test)
# print(y_check1)
y_testc['pred_linear_reg']= y_pred_test
# print(y_check1)
figure(figsize=(16, 8))
plt.plot(y_testc.index,y_testc[14], 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=(16, 8))
plt.plot(y_trainc.index,y_trainc[14], 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: 57.38
Root Mean squared error test: 135.99
r2_score train:  0.9938372416840585
r2_score test:  0.9808569441967903
adjusted r2_score test: 0.980364694190422
```





In [47]:

```python
display(y_testc)
validation = pd.DataFrame(y_testc)
validation =validation.rename(columns={14: "ethereum", 'pred_linear_reg': 'Predictions_test'})
display(validation)
```

	14	pred_linear_reg
1442	3323.197991	3122.139488
1443	3268.548177	3243.123858
1444	3309.754910	3302.236874
1445	3153.583684	3295.786407
1446	3007.144027	3165.503067
...	...	...
1797	1621.341324	1670.653246
1798	1612.418907	1634.880237
1799	1725.382627	1596.804354
1800	1693.296592	1698.710510
1801	1699.006418	1699.512923

360 rows × 2 columns

	ethereum	Predictions_test
1442	3323.197991	3122.139488
1443	3268.548177	3243.123858
1444	3309.754910	3302.236874
1445	3153.583684	3295.786407
1446	3007.144027	3165.503067
...	...	...
1797	1621.341324	1670.653246
1798	1612.418907	1634.880237
1799	1725.382627	1596.804354
1800	1693.296592	1698.710510
1801	1699.006418	1699.512923

360 rows × 2 columns

In [48]:

```python
# buy when prediction is below price, and sell when prediction is above price
# validation['buy'] = if validation['ethereum'] > validation['Predictions']

moneyStart = 1000
money = moneyStart
eth = money/validation['ethereum'].iloc[1]



validation['buy'] = np.where(validation['ethereum'] > validation['Predictions_test'], 1, 0)
# validation['sell'] = np.where(validation['ethereum'] < validation['Predictions'], 1, 0)
validation['signal'] = validation['buy'].diff()
validation['money'] = moneyStart

validation = validation.iloc[1:, :]
validation['eth'] = moneyStart/eth
display(validation.head(5))


for index, row in validation.iterrows():
    if row['signal'] == 1:
#         print('buy')
#         print('money: ', money, 'price: ', row['ethereum'])
        eth = money/row['ethereum']
        validation.loc[index, 'eth'] = eth
        validation.loc[index, 'money'] = money


        row['eth'] = eth
    elif row['signal'] == -1:
#         print('sell')
        money = eth*row['ethereum']
        validation.loc[index, 'eth'] = eth
        validation.loc[index, 'money'] = money
    else:
        validation.loc[index, 'eth'] = eth
        validation.loc[index, 'money'] = money

display(validation)
#     print(moneyStart/row['Close'])
```

	ethereum	Predictions_test	buy	signal	money	eth
1443	3268.548177	3243.123858	1	0.0	1000	3268.548177
1444	3309.754910	3302.236874	1	0.0	1000	3268.548177
1445	3153.583684	3295.786407	0	-1.0	1000	3268.548177
1446	3007.144027	3165.503067	0	0.0	1000	3268.548177
1447	3037.230251	2943.495844	1	1.0	1000	3268.548177

	ethereum	Predictions_test	buy	signal	money	eth
1443	3268.548177	3243.123858	1	0.0	1000.000000	0.305946
1444	3309.754910	3302.236874	1	0.0	1000.000000	0.305946
1445	3153.583684	3295.786407	0	-1.0	964.827046	0.305946
1446	3007.144027	3165.503067	0	0.0	964.827046	0.305946
1447	3037.230251	2943.495844	1	1.0	964.827046	0.317667
...	...	...	...	...	...	...
1797	1621.341324	1670.653246	0	0.0	781.996012	0.477987
1798	1612.418907	1634.880237	0	0.0	781.996012	0.477987
1799	1725.382627	1596.804354	1	1.0	781.996012	0.453230
1800	1693.296592	1698.710510	0	-1.0	767.453643	0.453230
1801	1699.006418	1699.512923	0	0.0	767.453643	0.453230

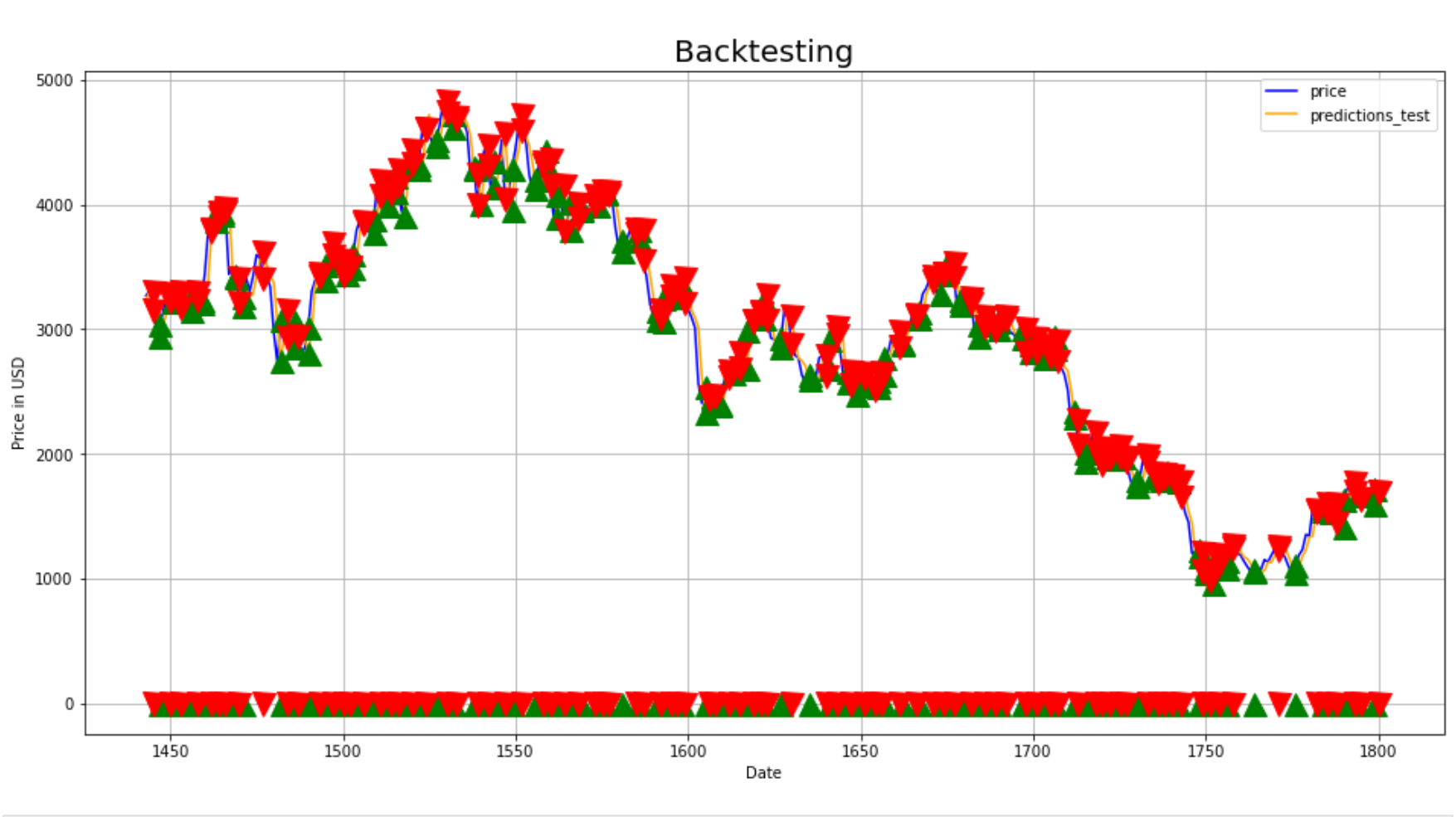
359 rows × 6 columns

In [49]:

```python
backtest = pd.DataFrame(validation)
display(backtest)
backtest = backtest.drop(columns=(['eth', 'money', 'buy']))
plt.figure(figsize = (16, 8))
backtest['ethereum'].plot(color='blue', label = 'price')
backtest['Predictions_test'].plot(color='orange', label='predictions_test')
plt.plot(backtest[backtest['signal']==1].index, backtest[backtest['signal']==1],'^', markersize = 15, color='g'
plt.plot(backtest[backtest['signal']==-1].index, backtest[backtest['signal']==-1],'v', markersize = 15, color='
plt.ylabel('Price in USD')
plt.xlabel('Date')
plt.title('Backtesting', fontsize = 20)
plt.legend()
plt.grid()
plt.show()
```

	ethereum	Predictions_test	buy	signal	money	eth
1443	3268.548177	3243.123858	1	0.0	1000.000000	0.305946
1444	3309.754910	3302.236874	1	0.0	1000.000000	0.305946
1445	3153.583684	3295.786407	0	-1.0	964.827046	0.305946
1446	3007.144027	3165.503067	0	0.0	964.827046	0.305946
1447	3037.230251	2943.495844	1	1.0	964.827046	0.317667
...	...	...	...	...	...	...
1797	1621.341324	1670.653246	0	0.0	781.996012	0.477987
1798	1612.418907	1634.880237	0	0.0	781.996012	0.477987
1799	1725.382627	1596.804354	1	1.0	781.996012	0.453230
1800	1693.296592	1698.710510	0	-1.0	767.453643	0.453230
1801	1699.006418	1699.512923	0	0.0	767.453643	0.453230

359 rows × 6 columns



In [50]:

```python
plt.xlabel('Date')
plt.ylabel('Money in USD')
plt.title('Change in USD After Trades')
validation['money'].plot(color='g', label = 'money')
plt.show()

plt.xlabel('Date')
plt.ylabel('Quantity in eth')
plt.title('Change in ETH After Trades')
validation['eth'].plot(color='g', label = 'eth')
plt.show()
```