In [1]:

```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 plotly.express as px



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 [2]:

```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&interval
    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)
```

```
08-17-2017
just added:  ethereum
```

date	ethereum	ethereum_volume
2017-08-17	296.114635	5.904704e+08
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
...	...	...
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	1883.678469	1.465477e+10

1826 rows × 2 columns

In [3]:

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

In [4]:

```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
2017-08-17	296.114635	5.904704e+08
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
...	...	...
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	1883.678469	1.465477e+10

1826 rows × 2 columns

In [27]:

```python
fig = px.line(my_data, y="ethereum", title='Ethereum Historical Price')

fig.show()
```

In [28]:

Train Test Split Sliding Window

Bull LSTM

In [6]:

```python
#bull
bull = pd.DataFrame(my_data)

bull = bull.reset_index()
display(bull)
bull['date'] = pd.to_datetime(bull['date'])
mask = bull['date'] <= '2021-12-01'
bull = bull.loc[mask]
bull = bull.set_index('date')


fig = px.line(bull, y="ethereum", title='Ethereum ending in Bull Market')
fig.show()

#stag
stag = pd.DataFrame(my_data)

stag = bull.reset_index()
display(bull)
stag['date'] = pd.to_datetime(stag['date'])
mask = stag['date'] <= '2020-07-17'
stag = stag.loc[mask]
stag = stag.set_index('date')

fig = px.line(stag, y="ethereum", title='Ethereum ending in stag Market')
fig.show()

#bear
bear = pd.DataFrame(my_data)

bear = bear.reset_index()
display(bear)
bear['date'] = pd.to_datetime(bear['date'])
mask = bear['date'] <= '2022-07-12'
bear = bear.loc[mask]
bear = bear.set_index('date')


fig = px.line(bear, y="ethereum", title='Ethereum ending in Bear Market')
fig.show()
```

	date	ethereum	ethereum_volume
0	2017-08-17	296.114635	5.904704e+08
1	2017-08-18	296.622090	5.537022e+08
2	2017-08-19	295.171577	3.428230e+08
3	2017-08-20	322.201220	1.743910e+09
4	2017-08-21	312.174471	8.983443e+08
...	...	...	...
1821	2022-08-12	1959.330925	1.584926e+10
1822	2022-08-13	1982.411828	1.481675e+10
1823	2022-08-14	1936.701164	1.217221e+10
1824	2022-08-15	1908.277642	1.831087e+10
1825	2022-08-16	1883.678469	1.465477e+10

1826 rows × 3 columns

date	ethereum	ethereum_volume
2017-08-17	296.114635	5.904704e+08
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
...	...	...
2021-11-27	4084.088486	1.650051e+10
2021-11-28	4290.091863	1.740861e+10
2021-11-29	4444.528280	2.015092e+10
2021-11-30	4637.121617	3.153114e+10
2021-12-01	4589.610618	2.795793e+10

1568 rows × 2 columns

	date	ethereum	ethereum_volume
0	2017-08-17	296.114635	5.904704e+08
1	2017-08-18	296.622090	5.537022e+08
2	2017-08-19	295.171577	3.428230e+08
3	2017-08-20	322.201220	1.743910e+09
4	2017-08-21	312.174471	8.983443e+08
...	...	...	...
1821	2022-08-12	1959.330925	1.584926e+10
1822	2022-08-13	1982.411828	1.481675e+10
1823	2022-08-14	1936.701164	1.217221e+10
1824	2022-08-15	1908.277642	1.831087e+10
1825	2022-08-16	1883.678469	1.465477e+10

1826 rows × 3 columns

In [50]:

```python
display(bull)
bull_test = bull.tail(365)
stag_test = stag.tail(365)
bear_test = bear.tail(175)
display(stag)
display(bear)
plt.figure(figsize=(16,8))
plt.title('Historical Price of Ethereum in USD')
plt.xlabel('Date')
plt.ylabel('Close Price USD ($)')
plt.plot(my_data['ethereum'], color='k', label='price_data')
plt.plot(bull_test['ethereum'], color='g', label='bull_test')
plt.plot(stag_test['ethereum'], color='b', label='stag_test')
plt.plot(bear_test['ethereum'], color='r', label='bear_test')
# plt.legend(['Train', 'Val', 'Predictions'], loc='lower right')
plt.legend()
plt.show()
bull_T_first = bull_test.head(1).values[0][0]
bull_T_last = bull_test.tail(1).values[0][0]

bear_T_first = bear_test.head(1).values[0][0]
bear_T_last = bear_test.tail(1).values[0][0]

stag_T_first = stag_test.head(1).values[0][0]
stag_T_last = stag_test.tail(1).values[0][0]

print('change in bull test: ', (bull_T_last-bull_T_first)/bull_T_first)

print('change in bear test: ', (bear_T_last-bear_T_first)/bear_T_first)


print('change in stag test: ', (stag_T_last-stag_T_first)/stag_T_first)
```

date	ethereum	ethereum_volume
2017-08-17	296.114635	5.904704e+08
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
...	...	...
2021-11-27	4084.088486	1.650051e+10
2021-11-28	4290.091863	1.740861e+10
2021-11-29	4444.528280	2.015092e+10
2021-11-30	4637.121617	3.153114e+10
2021-12-01	4589.610618	2.795793e+10

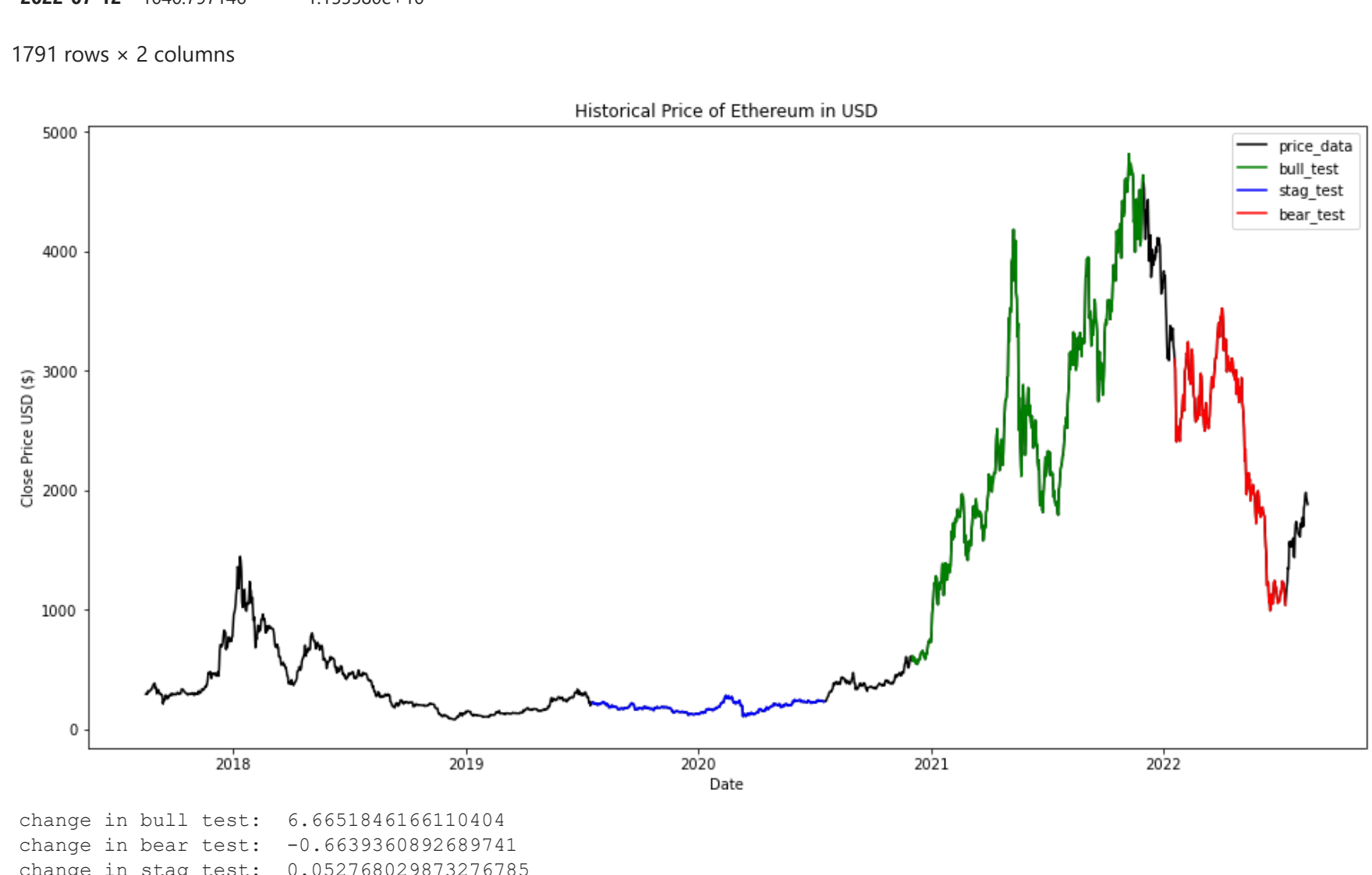
1568 rows × 2 columns

date	ethereum	ethereum_volume
2017-08-17	296.114635	5.904704e+08
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
...	...	...
2020-07-13	239.638787	5.773854e+09
2020-07-14	240.251471	4.794731e+09
2020-07-15	238.663597	4.622070e+09
2020-07-16	233.837075	4.836661e+09
2020-07-17	232.781477	4.344880e+09

1066 rows × 2 columns

date	ethereum	ethereum_volume
2017-08-17	296.114635	5.904704e+08
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
...	...	...
2022-07-08	1233.514679	1.534660e+10
2022-07-09	1216.849707	7.870255e+09
2022-07-10	1169.012708	1.037447e+10
2022-07-11	1097.449438	1.169316e+10
2022-07-12	1040.797146	1.155580e+10

1791 rows × 2 columns




```
change in bull test:  6.6651846166110404
change in bear test:  -0.6639360892689741
change in stag test:  0.052768029873276785
```

In [7]:

```python
seq=list(mydf['ethereum'])
# display(seq)
# scaler = StandardScaler()
scaler = MinMaxScaler(feature_range = (0, 1))
window_size = 15
my_target = window_size-1
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=[11,12,13])
display(sliding_df)

dataset = sliding_df

# We want 80% of the data to be used for training, and 20% for testing
n_train_rows = int(dataset.shape[0]*.8)-1

# Split into train and test sets but keep all 9 columns
train = dataset.iloc[:n_train_rows, :]
test = dataset.iloc[n_train_rows:, :]

# The total rows of the two datasets should equal the total amount of rows in your CSV
print(train.shape)
print(test.shape)

# Normalise features
sc = MinMaxScaler(feature_range = (0, 1))
training_set_scaled = sc.fit_transform(train.values)
test_set_scaled = sc.fit_transform(test.values)
```

	0	1	2	3	4	5	6	7	8	9	
0	296.114635	296.622090	295.171577	322.201220	312.174471	316.788920	321.785298	329.025281	329.865783	343.341337	344.
1	296.622090	295.171577	322.201220	312.174471	316.788920	321.785298	329.025281	329.865783	343.341337	344.201131	366.
2	295.171577	322.201220	312.174471	316.788920	321.785298	329.025281	329.865783	343.341337	344.201131	366.809590	378.
3	322.201220	312.174471	316.788920	321.785298	329.025281	329.865783	343.341337	344.201131	366.809590	378.906331	382.
4	312.174471	316.788920	321.785298	329.025281	329.865783	343.341337	344.201131	366.809590	378.906331	382.934451	386.
...	...	...	...	...	...	...	...	...	...	...	...
1807	1739.042177	1696.785098	1682.011374	1636.018556	1638.378683	1621.341324	1612.418907	1725.382627	1693.296592	1699.006418	1775.
1808	1696.785098	1682.011374	1636.018556	1638.378683	1621.341324	1612.418907	1725.382627	1693.296592	1699.006418	1775.701356	1698.
1809	1682.011374	1636.018556	1638.378683	1621.341324	1612.418907	1725.382627	1693.296592	1699.006418	1775.701356	1698.966129	1852.
1810	1636.018556	1638.378683	1621.341324	1612.418907	1725.382627	1693.296592	1699.006418	1775.701356	1698.966129	1852.878555	1881.
1811	1638.378683	1621.341324	1612.418907	1725.382627	1693.296592	1699.006418	1775.701356	1698.966129	1852.878555	1881.427405	1959.

1812 rows × 12 columns

```
(1448, 12)
(364, 12)
```

In [8]:

```python
display(bull)
```

date	ethereum	ethereum_volume
2017-08-17	296.114635	5.904704e+08
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
...	...	...
2021-11-27	4084.088486	1.650051e+10
2021-11-28	4290.091863	1.740861e+10
2021-11-29	4444.528280	2.015092e+10
2021-11-30	4637.121617	3.153114e+10
2021-12-01	4589.610618	2.795793e+10

1568 rows × 2 columns

In [9]:

```python
# close_prices = stock_data['Close']
# values = close_prices.values
values = bull['ethereum'].values
# print(values)
training_data_len = math.ceil(len(values)- 365)
print('training_data: ', training_data_len)
print('len values', len(values))
scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(values.reshape(-1,1))
train_data = scaled_data[0: training_data_len, :]

x_train = []
y_train = []

for i in range(60, len(train_data)):
    x_train.append(train_data[i-60:i, 0])
    y_train.append(train_data[i, 0])

x_train, y_train = np.array(x_train), np.array(y_train)
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))

print(y_train)
# print(y_train)

test_data = scaled_data[training_data_len-60: , : ]
x_test = []
y_test = values[training_data_len:]

for i in range(60, len(test_data)):
    x_test.append(test_data[i-60:i, 0])

x_test = np.array(x_test)
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
print(x_test)
```

```
training_data:  1203
len values 1568
[0.04948244 0.04875501 0.04749067 ... 0.10377139 0.11170015 0.10690605]
[[[0.0554205 ]
  [0.05678724]
  [0.05702938]
  ...
  [0.10377139]
  [0.11170015]
  [0.10690605]]

 [[0.05678724]
  [0.05702938]
  [0.05428412]
  ...
  [0.11170015]
  [0.10690605]
  [0.10884609]]

 [[0.05702938]
  [0.05428412]
  [0.05445908]
  ...
  [0.10690605]
  [0.10884609]
  [0.11259693]]

 ...

 [[0.61922889]
  [0.68086497]
  [0.69963756]
  ...
  [0.83795046]
  [0.84551208]
  [0.88905337]]

 [[0.68086497]
  [0.69963756]
  [0.70649897]
  ...
  [0.84551208]
  [0.88905337]
  [0.92169536]]

 [[0.69963756]
  [0.70649897]
  [0.69887373]
  ...
  [0.88905337]
  [0.92169536]
  [0.96240228]]]
```

In [10]:

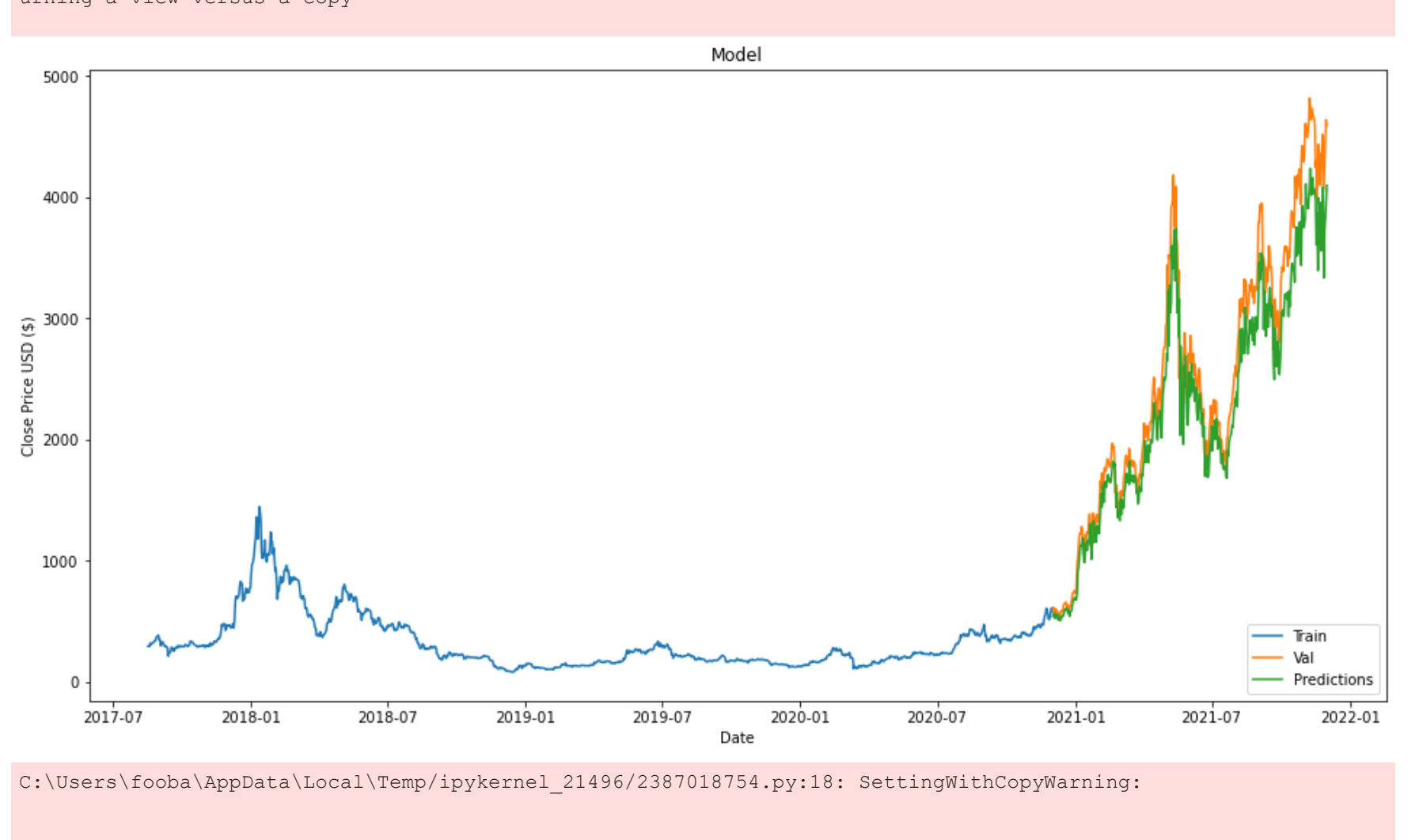
```python
model = keras.Sequential()
model.add(keras.layers.LSTM(100, return_sequences=True, input_shape=(x_train.shape[1], 1)))
model.add(keras.layers.LSTM(100, return_sequences=False))
# model.add(keras.layers.Dense(12, activation="relu", name="layer2"))
# model.add(keras.layers.Dense(1, activation='sigmoid')) # Produces probabilities


model.add(keras.layers.Dense(25))
model.add(keras.layers.Dense(1))

model.summary()

model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(x_train, y_train, batch_size= 1, epochs=9)


predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)
predictions_train = model.predict(x_train)
predictions_train = scaler.inverse_transform(predictions_train)
y_train = np.reshape(y_train, (y_train.shape[0], 1))

y_train = scaler.inverse_transform(y_train)

rmse_train = np.sqrt(np.mean(predictions_train - y_train)**2)
rmse = np.sqrt(np.mean(predictions - y_test)**2)
print('rmse train: ', rmse_train)
print('rmse test: ', rmse)

print('r2_score train: ', r2_score(y_train, predictions_train))
print('r2_score test: ', r2_score(y_test, predictions))
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 lstm (LSTM)                 (None, 60, 100)           40800     
                                                                 
 lstm_1 (LSTM)               (None, 100)               80400     
                                                                 
 dense (Dense)               (None, 25)                2525      
                                                                 
 dense_1 (Dense)             (None, 1)                 26        
                                                                 
=================================================================
Total params: 123,751
Trainable params: 123,751
Non-trainable params: 0
_________________________________________________________________
Epoch 1/9
1143/1143 [==============================] - 18s 14ms/step - loss: 2.9711e-04
Epoch 2/9
1143/1143 [==============================] - 18s 15ms/step - loss: 1.7430e-04
Epoch 3/9
1143/1143 [==============================] - 18s 16ms/step - loss: 1.1109e-04
Epoch 4/9
1143/1143 [==============================] - 18s 15ms/step - loss: 9.0508e-05
Epoch 5/9
1143/1143 [==============================] - 17s 15ms/step - loss: 8.5226e-05
Epoch 6/9
1143/1143 [==============================] - 17s 15ms/step - loss: 9.8158e-05
Epoch 7/9
1143/1143 [==============================] - 18s 15ms/step - loss: 6.6127e-05
Epoch 8/9
1143/1143 [==============================] - 18s 16ms/step - loss: 9.1762e-05
Epoch 9/9
1143/1143 [==============================] - 17s 15ms/step - loss: 5.7573e-05
12/12 [==============================] - 1s 19ms/step
36/36 [==============================] - 1s 18ms/step
rmse train:  33.27516914049641
rmse test:  244.58259406276989
r2_score train:  0.9660980570054355
r2_score test:  0.914156609554722
```

In [11]:

```python
data = bull.filter(['ethereum'])
train = data[:training_data_len]
validation = data[training_data_len:]
validation['Predictions'] = predictions
plt.figure(figsize=(16,8))
plt.title('Model')
plt.xlabel('Date')
plt.ylabel('Close Price USD ($)')
plt.plot(train)
plt.plot(validation[['ethereum', 'Predictions']])
plt.legend(['Train', 'Val', 'Predictions'], loc='lower right')
plt.show()


data = bull.filter(['ethereum'])
train = data[:training_data_len]
validation = data[training_data_len:]
validation['Predictions'] = predictions
plt.figure(figsize=(16,8))
plt.title('Model')
plt.xlabel('Date')
plt.ylabel('Close Price USD ($)')
# plt.plot(train)
plt.plot(validation[['ethereum', 'Predictions']])
plt.legend(['Val', 'Predictions'], loc='lower right')
plt.show()
```

```
C:\Users\fooba\AppData\Local\Temp/ipykernel_21496/2387018754.py:4: SettingWithCopyWarning: 


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```




```
C:\Users\fooba\AppData\Local\Temp/ipykernel_21496/2387018754.py:18: SettingWithCopyWarning: 


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

In [12]:

```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'], 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'])
```

```
C:\Users\fooba\AppData\Local\Temp/ipykernel_21496/2261137071.py:10: SettingWithCopyWarning: 


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

C:\Users\fooba\AppData\Local\Temp/ipykernel_21496/2261137071.py:12: SettingWithCopyWarning: 


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

C:\Users\fooba\AppData\Local\Temp/ipykernel_21496/2261137071.py:13: SettingWithCopyWarning: 


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

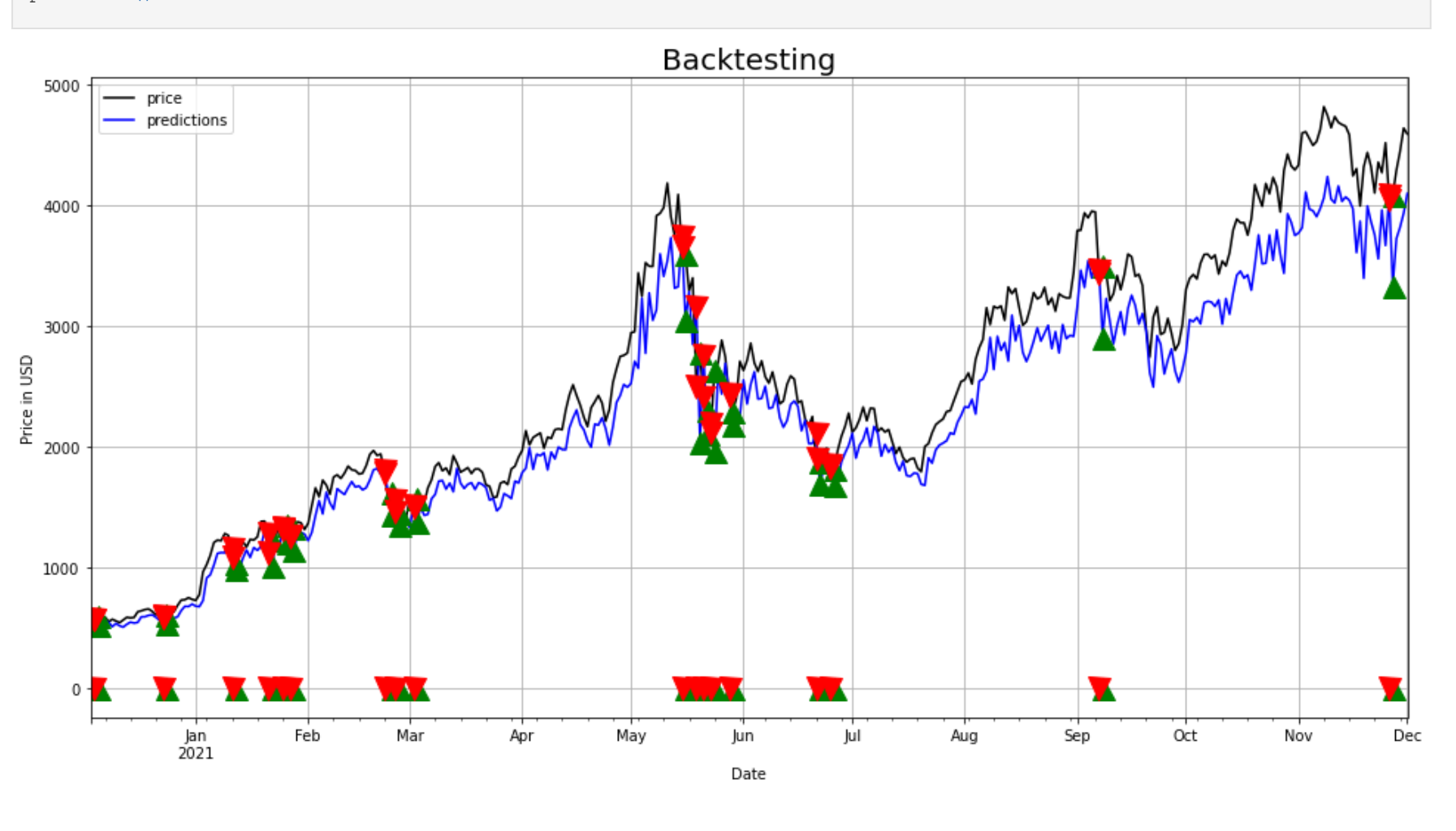
date	ethereum	Predictions	buy	signal	money	eth
2020-12-03	616.506662	556.392212	1	0.0	1000	616.506662
2020-12-04	571.190432	572.232422	0	-1.0	1000	616.506662
2020-12-05	595.919414	524.712402	1	1.0	1000	616.506662
2020-12-06	601.968935	558.290588	1	0.0	1000	616.506662
2020-12-07	592.386507	558.533508	1	0.0	1000	616.506662

date	ethereum	Predictions	buy	signal	money	eth
2020-12-03	616.506662	556.392212	1	0.0	1000.000000	1.622042
2020-12-04	571.190432	572.232422	0	-1.0	926.495150	1.622042
2020-12-05	595.919414	524.712402	1	1.0	926.495150	1.554732
2020-12-06	601.968935	558.290588	1	0.0	926.495150	1.554732
2020-12-07	592.386507	558.533508	1	0.0	926.495150	1.554732
...	...	...	...	...	...	...
2021-11-27	4084.088486	3335.356201	1	1.0	4578.928778	1.121163
2021-11-28	4290.091863	3723.946777	1	0.0	4578.928778	1.121163
2021-11-29	4444.528280	3816.771729	1	0.0	4578.928778	1.121163
2021-11-30	4637.121617	3934.184570	1	0.0	4578.928778	1.121163
2021-12-01	4589.610618	4096.252441	1	0.0	4578.928778	1.121163

364 rows × 6 columns

In [13]:

```python
validation = validation[['ethereum', 'Predictions', 'signal']]
plt.figure(figsize = (16, 8))
validation['ethereum'].plot(color='k', label = 'price')
validation['Predictions'].plot(color='b', label='predictions')
plt.plot(validation[validation['signal']==1].index, validation[validation['signal']==1],'^', markersize = 15, c
plt.plot(validation[validation['signal']==-1].index, validation[validation['signal']==-1],'v', markersize = 15,
plt.ylabel('Price in USD')
plt.xlabel('Date')
plt.title('Backtesting', fontsize = 20)
plt.legend()
plt.grid()
plt.show()
```



LSTM STAGNANT

In [14]:

```python
# close_prices = stock_data['Close']
# values = close_prices.values
values = stag['ethereum'].values
# print(values)
training_data_len = math.ceil(len(values)- 365)
print('training_data: ', training_data_len)
print('len values', len(values))
scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(values.reshape(-1,1))
train_data = scaled_data[0: training_data_len, :]

x_train = []
y_train = []

for i in range(60, len(train_data)):
    x_train.append(train_data[i-60:i, 0])
    y_train.append(train_data[i, 0])

x_train, y_train = np.array(x_train), np.array(y_train)
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))

print(y_train)
# print(y_train)

test_data = scaled_data[training_data_len-60: , : ]
x_test = []
y_test = values[training_data_len:]

for i in range(60, len(test_data)):
    x_test.append(test_data[i-60:i, 0])

x_test = np.array(x_test)
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
print(x_test)
```

```
training_data:  701
len values 1066
[1.71586967e-01 1.69064486e-01 1.64680218e-01 1.60196039e-01
 1.58977439e-01 1.55162121e-01 1.50710164e-01 1.57556946e-01
 1.57357761e-01 1.55323076e-01 1.56951106e-01 1.55877973e-01
 1.61210873e-01 1.63049294e-01 1.61065011e-01 1.52847753e-01
 1.48991925e-01 1.62751588e-01 1.58435854e-01 1.56649283e-01
 1.59473592e-01 1.54976592e-01 1.65016853e-01 1.74834789e-01
 1.59461967e-01 1.70385064e-01 1.68851625e-01 1.69714887e-01
 1.87007303e-01 1.83598687e-01 1.80472306e-01 1.82912905e-01
 1.94197005e-01 1.97633340e-01 2.06685773e-01 2.03218304e-01
 2.16669868e-01 2.40965307e-01 2.86732437e-01 2.81194723e-01
 2.82332011e-01 2.93593072e-01 2.86132083e-01 2.52536620e-01
 2.71457501e-01 2.82966271e-01 2.82961559e-01 2.77115324e-01
 2.84717419e-01 2.80879765e-01 2.67182520e-01 2.69745826e-01
 2.81513746e-01 2.91887476e-01 2.65982599e-01 3.11651089e-01
 4.23671841e-01 4.58882973e-01 4.55026792e-01 4.46579981e-01
 4.53284458e-01 4.72674646e-01 5.23550658e-01 5.46911795e-01
 5.34755777e-01 5.34375797e-01 4.29121263e-01 4.63160237e-01
 4.41531033e-01 4.76487382e-01 5.05956617e-01 4.99255375e-01
 4.78640353e-01 4.92202119e-01 4.78691031e-01 5.06386154e-01
 5.07400579e-01 5.81286802e-01 6.44439736e-01 6.55697739e-01
 6.75104791e-01 7.09646414e-01 7.72408015e-01 8.07052705e-01
 9.36425688e-01 9.24807295e-01 8.04654189e-01 9.02842349e-01
 1.00000000e+00 9.72016977e-01 9.13141894e-01 7.21796167e-01
 6.87909186e-01 6.97955476e-01 7.05890342e-01 7.98252664e-01
 7.10826310e-01 6.71800878e-01 6.65408435e-01 7.13205410e-01
 7.14991011e-01 7.07283128e-01 7.41718639e-01 8.45458210e-01
 7.95846505e-01 7.90184201e-01 7.14097249e-01 7.49308570e-01
 6.87106020e-01 6.08953722e-01 6.28863209e-01 5.50503110e-01
 4.40142160e-01 5.14479607e-01 4.89669308e-01 5.30665425e-01
 5.77209259e-01 5.64766370e-01 5.37604604e-01 5.70922138e-01
 5.49689572e-01 6.09274524e-01 6.18961592e-01 6.23639998e-01
 6.45334460e-01 6.09285887e-01 6.18959189e-01 6.00400631e-01
 5.31623293e-01 5.34054558e-01 5.72330075e-01 5.45984055e-01
 5.56175118e-01 5.71582873e-01 5.75696764e-01 5.58914199e-01
 5.65740356e-01 5.61902909e-01 5.57367756e-01 5.57002263e-01
 5.54359638e-01 5.31124518e-01 4.90186228e-01 4.51045578e-01
 4.59622099e-01 4.40597308e-01 4.61310524e-01 4.48204980e-01
 4.40382740e-01 3.84498275e-01 3.87466912e-01 3.84121799e-01
 3.49109881e-01 3.33411453e-01 3.45421882e-01 3.44785906e-01
 3.49870000e-01 3.32689801e-01 3.32985212e-01 3.19335616e-01
 3.19256786e-01 2.95921569e-01 2.67713180e-01 2.63972948e-01
 2.21007353e-01 2.28336003e-01 2.28680565e-01 2.16460586e-01
 2.20640074e-01 2.43280344e-01 2.17309641e-01 2.20375123e-01
 2.09710618e-01 2.19628913e-01 2.29549357e-01 2.30862466e-01
 2.41061859e-01 2.51832647e-01 2.96702778e-01 2.96986195e-01
 3.01472239e-01 3.23763841e-01 3.10265446e-01 3.03930542e-01
 3.17788001e-01 3.53714600e-01 3.71216756e-01 3.78537041e-01
 3.90954810e-01 4.05001159e-01 4.55324214e-01 3.88407392e-01
 4.20165861e-01 4.09124925e-01 4.31554904e-01 4.37069116e-01
 4.24325174e-01 4.26152395e-01 4.34454086e-01 5.11318398e-01
 5.08739867e-01 5.30111757e-01 5.14592352e-01 4.87415746e-01
 4.84126142e-01 4.83624080e-01 4.64401672e-01 4.33530004e-01
 4.39304373e-01 4.71334446e-01 4.72016632e-01 4.53952204e-01
 4.53437277e-01 4.26446845e-01 4.41534668e-01 4.41266040e-01
 4.54435191e-01 4.43513533e-01 4.08470235e-01 3.64597895e-01
 3.76770026e-01 3.65358818e-01 3.64042536e-01 3.53373042e-01
 3.13567693e-01 3.50305451e-01 3.45620947e-01 3.57624782e-01
 3.59168645e-01 3.65983456e-01 3.85159026e-01 3.68507900e-01
 3.78843846e-01 3.77039249e-01 3.74774539e-01 3.71382560e-01
 3.64509989e-01 3.21909974e-01 3.26404856e-01 3.01445445e-01
 2.86627701e-01 3.17253058e-01 2.94377458e-01 2.99973364e-01
 2.98751150e-01 3.14163542e-01 3.28510614e-01 3.26868224e-01
 3.20201615e-01 2.78625406e-01 2.85248445e-01 2.72598996e-01
 2.74949761e-01 2.54190352e-01 2.59522208e-01 2.47051901e-01
 2.55586895e-01 2.67990074e-01 2.66707824e-01 2.84760605e-01
 2.75516256e-01 2.78098710e-01 2.80779220e-01 2.81300062e-01
 2.91512441e-01 2.93183664e-01 2.82848086e-01 2.55042503e-01
 2.63295965e-01 2.52131167e-01 2.54677861e-01 2.55057245e-01
 2.65922839e-01 2.89082559e-01 3.02985462e-01 2.87456673e-01
 2.79247744e-01 2.66358028e-01 2.74329140e-01 2.72463237e-01
 2.66884334e-01 2.88350193e-01 2.82141005e-01 2.76374227e-01
 2.81569641e-01 2.80060714e-01 2.80441890e-01 2.74330527e-01
 2.55471751e-01 2.44423102e-01 2.39768821e-01 2.44481726e-01
 2.36806963e-01 2.38478726e-01 2.35292156e-01 2.15454562e-01
 1.97787542e-01 2.05145429e-01 1.82283452e-01 1.71159133e-01
 1.71629404e-01 1.47444715e-01 1.42158377e-01 1.44646790e-01
 1.48442635e-01 1.69311262e-01 1.53468141e-01 1.58054888e-01
 1.37259159e-01 1.44085020e-01 1.35930252e-01 1.40738236e-01
 1.45081744e-01 1.42320129e-01 1.39048863e-01 1.48912354e-01
 1.55754237e-01 1.50044181e-01 1.47550408e-01 1.45439806e-01
 1.55277860e-01 1.54500611e-01 1.50298847e-01 1.47831034e-01
 1.09066584e-01 1.06705968e-01 9.64869848e-02 8.28788563e-02
 8.21546898e-02 8.25095684e-02 7.35097489e-02 7.26766644e-02
 9.38444126e-02 9.35196049e-02 1.02072501e-01 9.99385842e-02
 8.26672040e-02 9.26160058e-02 9.18770920e-02 1.03045750e-01
 1.19924333e-01 1.15844657e-01 1.18000239e-01 1.05948088e-01
 9.94715376e-02 9.61550740e-02 1.06593831e-01 1.01468685e-01
 1.08436344e-01 1.09202627e-01 1.08002314e-01 1.04082016e-01
 1.00182561e-01 1.01444211e-01 1.05724662e-01 1.03549878e-01
 1.03981968e-01 1.06815655e-01 1.05675437e-01 1.03984656e-01
 7.93136821e-02 8.30273684e-02 8.59484333e-02 8.24781741e-02
 9.41202434e-02 9.42970377e-02 9.32990904e-02 8.83374419e-02
 8.80906023e-02 9.02203653e-02 8.98009324e-02 8.88685977e-02
 8.81695260e-02 8.77578072e-02 8.69008876e-02 8.76572276e-02
 8.75124217e-02 8.86374225e-02 8.20623896e-02 8.33061981e-02
 8.41788194e-02 8.45332318e-02 8.68493881e-02 8.52761408e-02
 9.01707689e-02 9.18524289e-02 9.92198093e-02 9.91134527e-02
 9.50031005e-02 9.30850222e-02 9.46792890e-02 9.43183206e-02
 9.37199484e-02 9.14664846e-02 7.34660134e-02 7.23121556e-02
 6.76024923e-02 6.73423114e-02 6.88530723e-02 4.88166867e-02
 3.49537693e-02 3.91287824e-02 3.09884252e-02 2.95840720e-02
 2.23556877e-02 2.47284231e-02 1.87959078e-02 1.95317326e-02
 2.83352603e-02 2.48604292e-02 2.17070129e-02 2.51502085e-02
 2.35696708e-02 1.82431783e-02 1.91044158e-02 1.37240985e-02
 5.36149780e-03 6.72971809e-03 5.05154165e-03 7.14175200e-03
 4.44721569e-03 2.97894009e-03 4.09847797e-03 1.59247586e-03
 0.00000000e+00 3.49017741e-05 4.98304727e-04 7.53968802e-03
 1.19203555e-02 1.15110370e-02 2.24592153e-02 1.76394916e-02
 2.29111654e-02 3.32898979e-02 4.04368906e-02 3.21396211e-02
 3.35133455e-02 2.22189473e-02 3.68177906e-02 4.02877023e-02
 3.94649388e-02 3.49406095e-02 3.98410099e-02 5.06264936e-02
 4.61337644e-02 5.07702308e-02 5.12065450e-02 5.19742388e-02
 4.84557461e-02 4.71725157e-02 4.76056855e-02 3.13976654e-02
 2.99333118e-02 2.92975465e-02 2.27061103e-02 3.19697294e-02
 2.64061610e-02 2.81880253e-02 2.79547556e-02 2.59344248e-02
 2.86883480e-02 2.47889594e-02 2.34836118e-02 2.49423725e-02
 2.39154191e-02 2.37903937e-02 2.29874515e-02 2.31246998e-02
 2.10551529e-02 1.58065981e-02 1.52535719e-02 1.79472572e-02
 1.65275070e-02 1.70211785e-02 1.83174452e-02 1.71607560e-02
 1.74673011e-02 1.72465388e-02 1.52724667e-02 1.50098873e-02
 2.60537932e-02 2.61680934e-02 3.01259524e-02 2.72533418e-02
 2.84053749e-02 2.80770539e-02 2.69762314e-02 2.76945755e-02
 2.86959943e-02 3.67163024e-02 4.51043165e-02 4.38797491e-02
 4.77006396e-02 4.50821796e-02 4.70946678e-02 5.41556373e-02
 3.70348021e-02 3.96043220e-02 3.82674438e-02 3.74260295e-02
 3.78957374e-02 3.75029074e-02 3.56359379e-02 3.42675071e-02
 3.06642483e-02 3.86723194e-02 3.92413660e-02 3.88452127e-02
 3.64672714e-02 3.91456083e-02 3.79911045e-02 3.56213581e-02
 3.62809121e-02 3.50942453e-02 3.54445059e-02 3.85972803e-02
 4.19981887e-02 4.01384519e-02 3.94841825e-02 4.01178467e-02
 4.05234894e-02 3.75771542e-02 3.83419811e-02 3.91471243e-02
 3.83111593e-02 3.66694321e-02 3.68250941e-02 4.09343923e-02
 3.95970821e-02 4.35068429e-02 4.28788788e-02 4.23634386e-02
 4.23107897e-02 5.88280022e-02 5.61764923e-02 5.39626067e-02
 5.96254115e-02 5.97621963e-02 6.61865652e-02 7.05265139e-02
 6.70687146e-02 6.85574306e-02 5.95510806e-02 5.92101539e-02
 5.87701432e-02 6.14426853e-02 5.62495302e-02 6.06119405e-02
 6.04966709e-02 6.56904511e-02 6.55545918e-02 6.55091691e-02
 6.31796381e-02 6.39533013e-02 6.38576496e-02 6.00146954e-02
 5.14107156e-02 5.39358369e-02 5.46053598e-02 5.33637792e-02
 5.15205386e-02 5.64050579e-02 5.54849590e-02 5.65608018e-02
 6.07201039e-02 5.79380644e-02 5.75963875e-02 6.52614699e-02
 6.32588001e-02 6.32696002e-02 6.36103090e-02 6.51542886e-02
 8.27504336e-02 7.58476266e-02 8.23003769e-02 9.85775627e-02
 1.20902780e-01 1.32743286e-01 1.17110250e-01 1.11068522e-01
 1.29589404e-01 1.23260233e-01 1.26281017e-01 1.17827758e-01
 1.17898583e-01 1.20756384e-01 1.22423865e-01 1.33638966e-01
 1.38976797e-01 1.37381055e-01 1.35591169e-01 1.25763808e-01
 1.35107645e-01 1.32616638e-01 1.36436286e-01 1.23107107e-01
 1.14668825e-01 1.19496708e-01 1.21799080e-01 1.22289394e-01
 1.17886643e-01 1.08660408e-01 1.19808720e-01 1.18223476e-01
 1.29567160e-01 1.25575493e-01 1.32031187e-01 1.35097497e-01
 1.35800740e-01 1.38770552e-01 1.32661059e-01 1.35860185e-01
 1.38192275e-01 1.54419941e-01 1.66278227e-01 1.64293619e-01
 1.66581095e-01 1.72127671e-01 1.85837062e-01 1.55421140e-01
 1.65760806e-01 1.74177558e-01 1.53220327e-01 1.54991825e-01
 1.51843009e-01 1.61525600e-01 1.47635866e-01 1.49663462e-01
 1.50829714e-01 1.62379980e-01 1.67605828e-01 1.64170834e-01
 1.51146946e-01 1.35480832e-01 1.40738840e-01 1.36711816e-01
 1.05054770e-01 1.07296146e-01 8.51327028e-02 9.31201422e-02
 1.04454841e-01]
[[[0.12326023]
  [0.12628102]
  [0.11782776]
  ...
  [0.0851327 ]
  [0.09312014]
  [0.10445484]]

 [[0.12628102]
  [0.11782776]
  [0.11789858]
  ...
  [0.09312014]
  [0.10445484]
  [0.10065111]]

 [[0.11782776]
  [0.11789858]
  [0.12075638]
  ...
  [0.10445484]
  [0.10065111]
  [0.10640536]]

 ...

 [[0.08560388]
  [0.09013739]
  [0.09569939]
  ...
  [0.11579173]
  [0.1142286 ]
  [0.11467765]]

 [[0.09013739]
  [0.09569939]
  [0.09520765]
  ...
  [0.1142286 ]
  [0.11467765]
  [0.11351385]]

 [[0.09569939]
  [0.09520765]
  [0.0924475 ]
  ...
  [0.11467765]
  [0.11351385]
  [0.10997637]]]
```

In [15]:

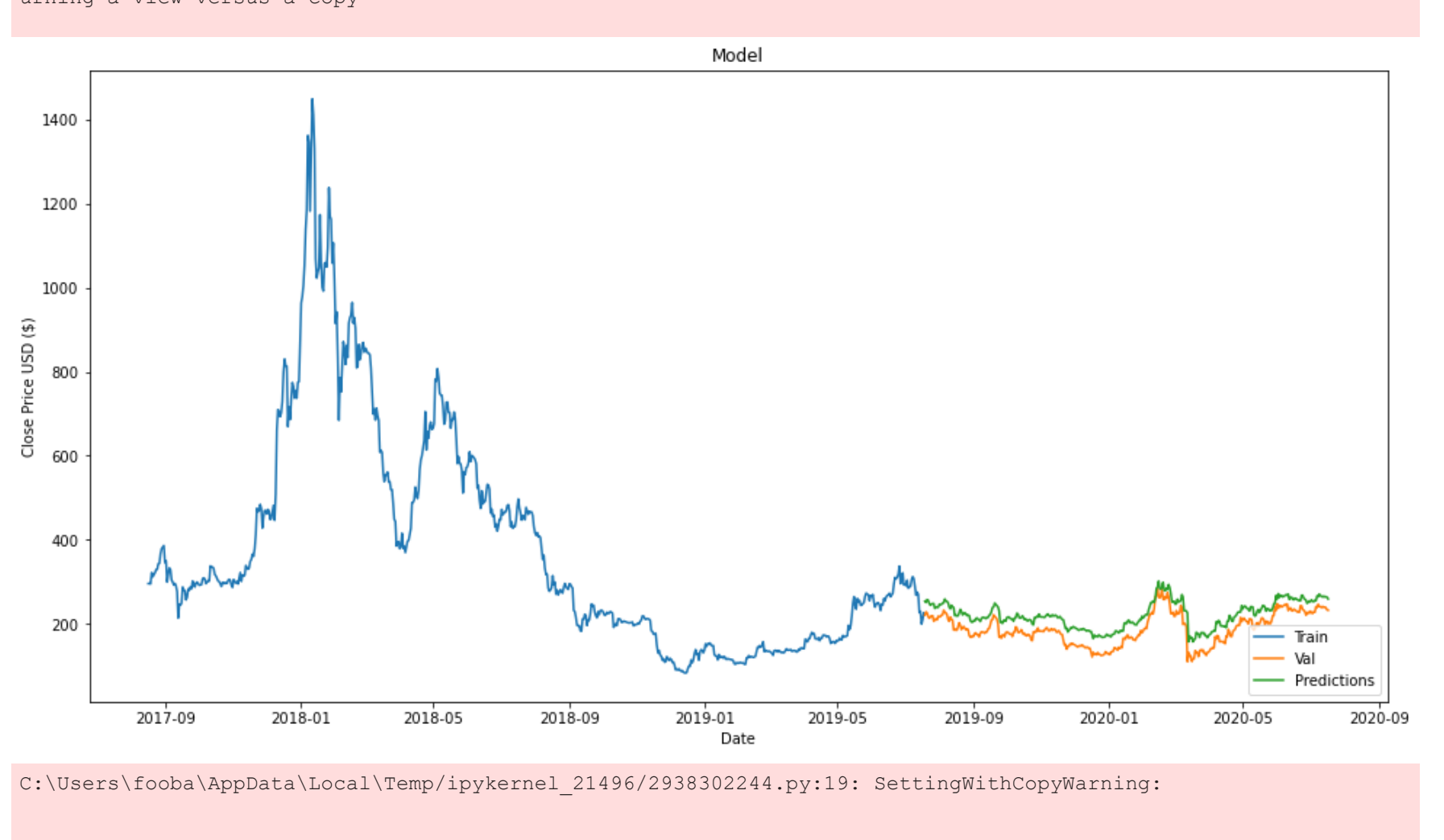
```python
#stag
model = keras.Sequential()
model.add(keras.layers.LSTM(100, return_sequences=True, input_shape=(x_train.shape[1], 1)))
model.add(keras.layers.LSTM(100, return_sequences=False))
model.add(keras.layers.Dense(25))
model.add(keras.layers.Dense(1))
model.summary()

model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(x_train, y_train, batch_size= 1, epochs=9)


predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)
predictions_train = model.predict(x_train)
predictions_train = scaler.inverse_transform(predictions_train)
y_train = np.reshape(y_train, (y_train.shape[0], 1))

y_train = scaler.inverse_transform(y_train)

rmse_train = np.sqrt(np.mean(predictions_train - y_train)**2)
rmse = np.sqrt(np.mean(predictions - y_test)**2)
print('rmse train: ', rmse_train)
print('rmse test: ', rmse)

print('r2_score train: ', r2_score(y_train, predictions_train))
print('r2_score test: ', r2_score(y_test, predictions))
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 lstm_2 (LSTM)               (None, 60, 100)           40800     
                                                                 
 lstm_3 (LSTM)               (None, 100)               80400     
                                                                 
 dense_2 (Dense)             (None, 25)                2525      
                                                                 
 dense_3 (Dense)             (None, 1)                 26        
                                                                 
=================================================================
Total params: 123,751
Trainable params: 123,751
Non-trainable params: 0
_________________________________________________________________
Epoch 1/9
641/641 [==============================] - 12s 15ms/step - loss: 0.0034
Epoch 2/9
641/641 [==============================] - 10s 15ms/step - loss: 0.0022
Epoch 3/9
641/641 [==============================] - 10s 15ms/step - loss: 0.0020
Epoch 4/9
641/641 [==============================] - 10s 15ms/step - loss: 0.0012
Epoch 5/9
641/641 [==============================] - 10s 15ms/step - loss: 0.0013
Epoch 6/9
641/641 [==============================] - 10s 16ms/step - loss: 0.0013
Epoch 7/9
641/641 [==============================] - 12s 18ms/step - loss: 0.0011
Epoch 8/9
641/641 [==============================] - 10s 16ms/step - loss: 9.5057e-04
Epoch 9/9
641/641 [==============================] - 10s 15ms/step - loss: 0.0010
12/12 [==============================] - 1s 20ms/step
21/21 [==============================] - 0s 17ms/step
rmse train:  1.3154710220975507
rmse test:  33.112058401187674
r2_score train:  0.9575670187228563
r2_score test:  0.14013706817077487
```

In [16]:

```python
#stag
data = stag.filter(['ethereum'])
train = data[:training_data_len]
validation = data[training_data_len:]
validation['Predictions'] = predictions
plt.figure(figsize=(16,8))
plt.title('Model')
plt.xlabel('Date')
plt.ylabel('Close Price USD ($)')
plt.plot(train)
plt.plot(validation[['ethereum', 'Predictions']])
plt.legend(['Train', 'Val', 'Predictions'], loc='lower right')
plt.show()


data = stag.filter(['ethereum'])
train = data[:training_data_len]
validation = data[training_data_len:]
validation['Predictions'] = predictions
plt.figure(figsize=(16,8))
plt.title('Model')
plt.xlabel('Date')
plt.ylabel('Close Price USD ($)')
# plt.plot(train)
plt.plot(validation[['ethereum', 'Predictions']])
plt.legend(['Val', 'Predictions'], loc='lower right')
plt.show()
```

```
C:\Users\fooba\AppData\Local\Temp/ipykernel_21496/2938302244.py:5: SettingWithCopyWarning: 


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```




```
C:\Users\fooba\AppData\Local\Temp/ipykernel_21496/2938302244.py:19: SettingWithCopyWarning: 


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

In [17]:

```python
#stag

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



validation['buy'] = np.where(validation['ethereum'] > validation['Predictions'], 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'])
```

```
C:\Users\fooba\AppData\Local\Temp/ipykernel_21496/3781293657.py:9: SettingWithCopyWarning: 


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

C:\Users\fooba\AppData\Local\Temp/ipykernel_21496/3781293657.py:11: SettingWithCopyWarning: 


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

C:\Users\fooba\AppData\Local\Temp/ipykernel_21496/3781293657.py:12: SettingWithCopyWarning: 


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

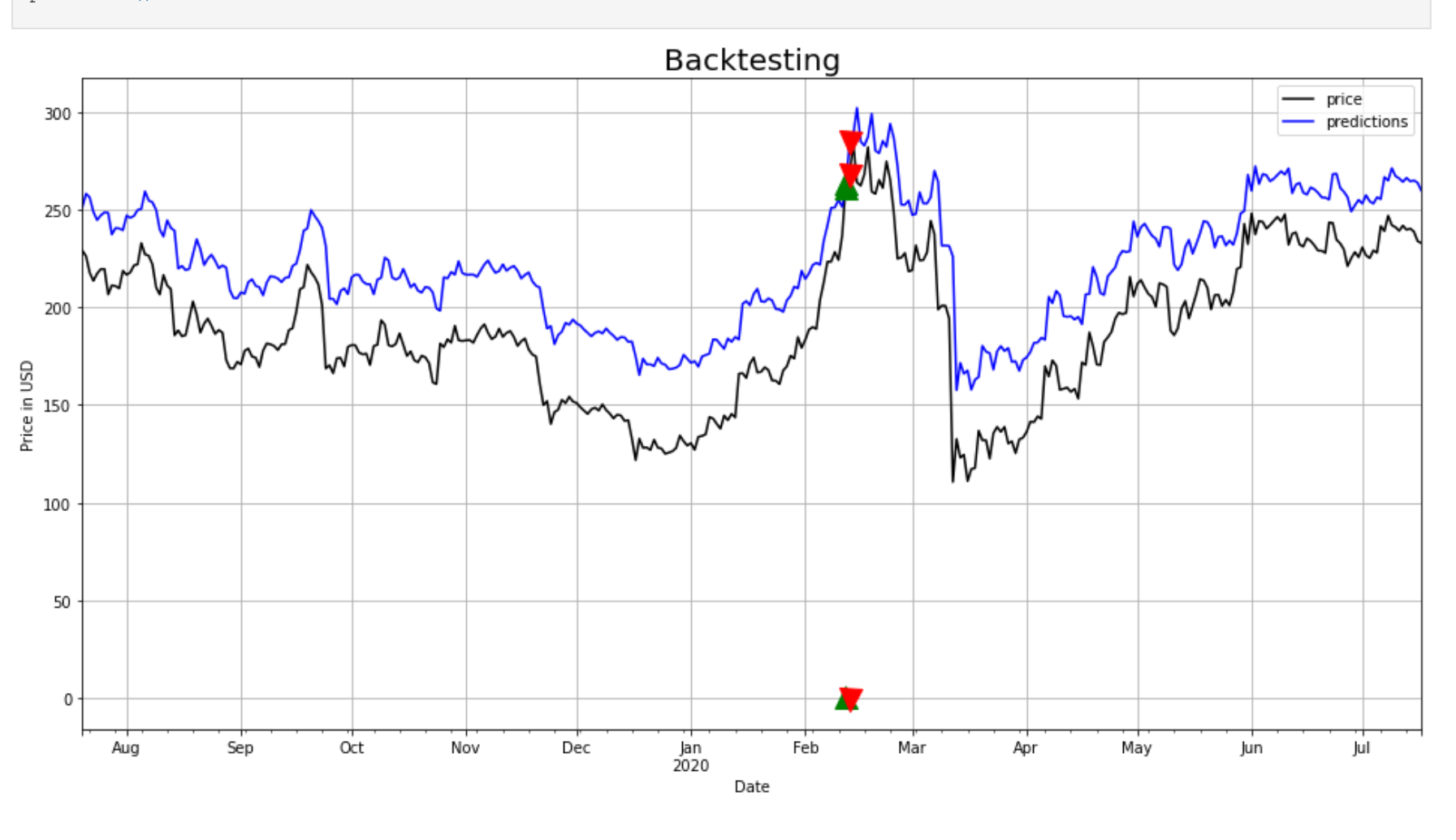
date	ethereum	Predictions	buy	signal	money	eth
2019-07-20	228.964804	251.210114	0	0.0	1000	228.964804
2019-07-21	226.153408	258.115814	0	0.0	1000	228.964804
2019-07-22	217.652562	256.174744	0	0.0	1000	228.964804
2019-07-23	213.502152	248.780853	0	0.0	1000	228.964804
2019-07-24	217.197778	244.596375	0	0.0	1000	228.964804

date	ethereum	Predictions	buy	signal	money	eth
2019-07-20	228.964804	251.210114	0	0.0	1000.000000	4.367483
2019-07-21	226.153408	258.115814	0	0.0	1000.000000	4.367483
2019-07-22	217.652562	256.174744	0	0.0	1000.000000	4.367483
2019-07-23	213.502152	248.780853	0	0.0	1000.000000	4.367483
2019-07-24	217.197778	244.596375	0	0.0	1000.000000	4.367483
...	...	...	...	...	...	...
2020-07-13	239.638787	266.173553	0	0.0	1013.777367	3.787409
2020-07-14	240.251471	264.406189	0	0.0	1013.777367	3.787409
2020-07-15	238.663597	264.891296	0	0.0	1013.777367	3.787409
2020-07-16	233.837075	263.668793	0	0.0	1013.777367	3.787409
2020-07-17	232.781477	259.708771	0	0.0	1013.777367	3.787409

364 rows × 6 columns

In [18]:

```python
#stag
validation = validation[['ethereum', 'Predictions', 'signal']]
plt.figure(figsize = (16, 8))
validation['ethereum'].plot(color='k', label = 'price')
validation['Predictions'].plot(color='b', label='predictions')
plt.plot(validation[validation['signal']==1].index, validation[validation['signal']==1],'^', markersize = 15, c
plt.plot(validation[validation['signal']==-1].index, validation[validation['signal']==-1],'v', markersize = 15,
plt.ylabel('Price in USD')
plt.xlabel('Date')
plt.title('Backtesting', fontsize = 20)
plt.legend()
plt.grid()
plt.show()
```



LSTM Bear

In [19]:
```python
# close_prices = stock_data['Close']
# values = close_prices.values
values = bear['ethereum'].values
# print(values)
training_data_len = math.ceil(len(values)- 175)
print('training_data: ', training_data_len)
print('len values', len(values))
scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(values.reshape(-1,1))
train_data = scaled_data[0: training_data_len, :]

x_train = []
y_train = []

for i in range(60, len(train_data)):
    x_train.append(train_data[i-60:i, 0])
    y_train.append(train_data[i, 0])

x_train, y_train = np.array(x_train), np.array(y_train)
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))

print(y_train)
# print(y_train)

test_data = scaled_data[training_data_len-60: , : ]
x_test = []
y_test = values[training_data_len:]

for i in range(60, len(test_data)):
    x_test.append(test_data[i-60:i, 0])

x_test = np.array(x_test)
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
print(x_test)
```

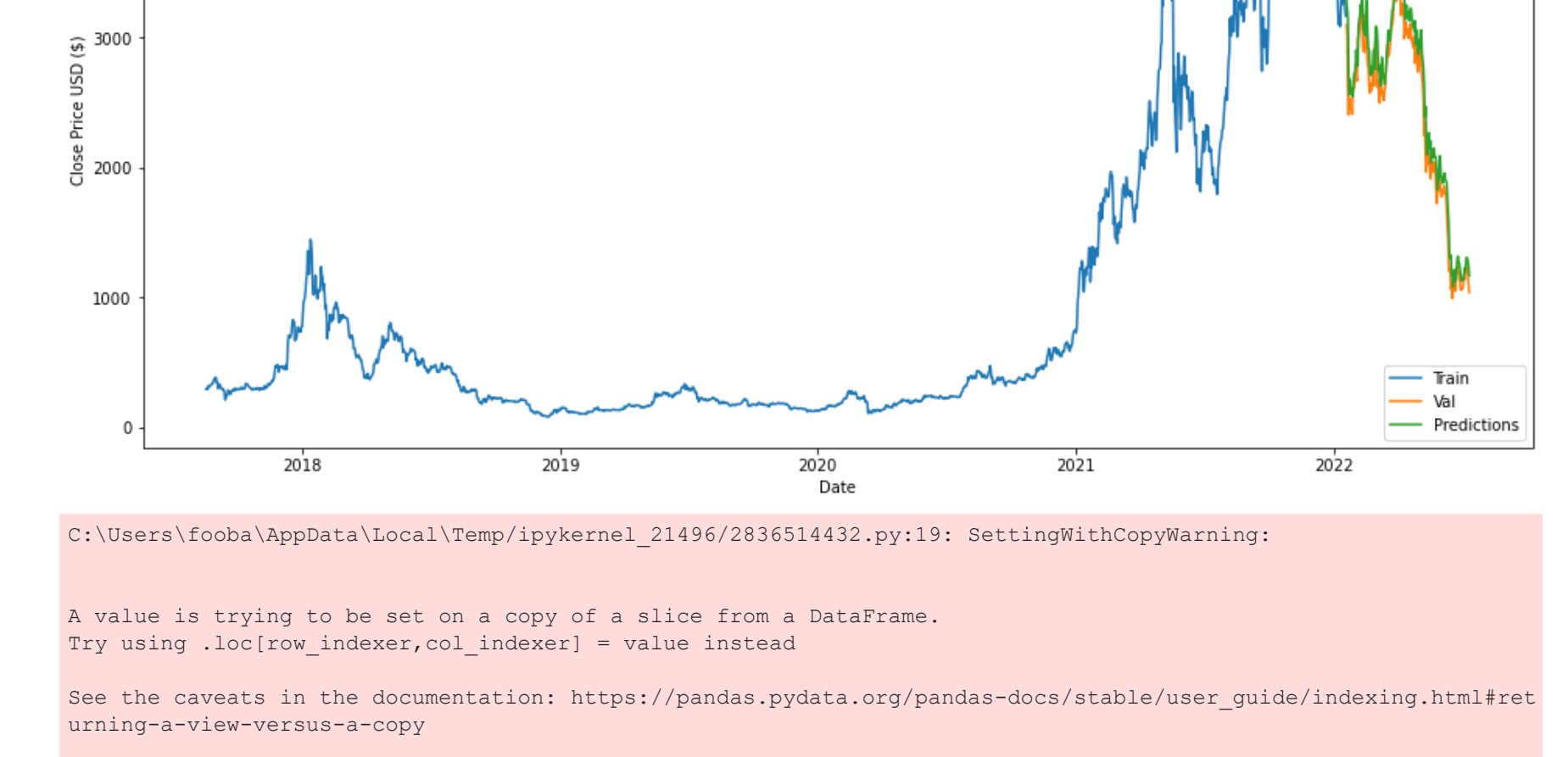
```
training_data:  1616
len values 1791
[0.04948244 0.04875501 0.04749067 ... 0.69168426 0.66215646 0.65155897]
[[[0.91993355]
  [0.89523987]
  [0.84909982]
  ...
  [0.69168426]
  [0.66215646]
  [0.65155897]]

 [[0.89523987]
  [0.84909982]
  [0.90286146]
  ...
  [0.66215646]
  [0.65155897]
  [0.63688358]]

 [[0.84909982]
  [0.90286146]
  [0.88468776]
  ...
  [0.65155897]
  [0.63688358]
  [0.61967181]]

 ...

 [[0.42211625]
  [0.39797636]
  [0.40717376]
  ...
  [0.24445973]
  [0.243009  ]
  [0.23948666]]

 [[0.39797636]
  [0.40717376]
  [0.41859054]
  ...
  [0.243009  ]
  [0.23948666]
  [0.22937573]]

 [[0.40717376]
  [0.41859054]
  [0.43609514]
  ...
  [0.23948666]
  [0.22937573]
  [0.21424997]]]
```

In [20]:
```python
#bear
model = keras.Sequential()
model.add(keras.layers.LSTM(100, return_sequences=True, input_shape=(x_train.shape[1], 1)))
model.add(keras.layers.LSTM(100, return_sequences=False))
model.add(keras.layers.Dense(25))
model.add(keras.layers.Dense(1))
model.summary()

model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(x_train, y_train, batch_size= 1, epochs=9)


predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)
predictions_train = model.predict(x_train)
predictions_train = scaler.inverse_transform(predictions_train)
y_train = np.reshape(y_train, (y_train.shape[0], 1))

y_train = scaler.inverse_transform(y_train)

rmse_train = np.sqrt(np.mean(predictions_train - y_train)**2)
rmse = np.sqrt(np.mean(predictions - y_test)**2)
print('rmse train: ', rmse_train)
print('rmse test: ', rmse)

print('r2_score train: ', r2_score(y_train, predictions_train))
print('r2_score test: ', r2_score(y_test, predictions))
```

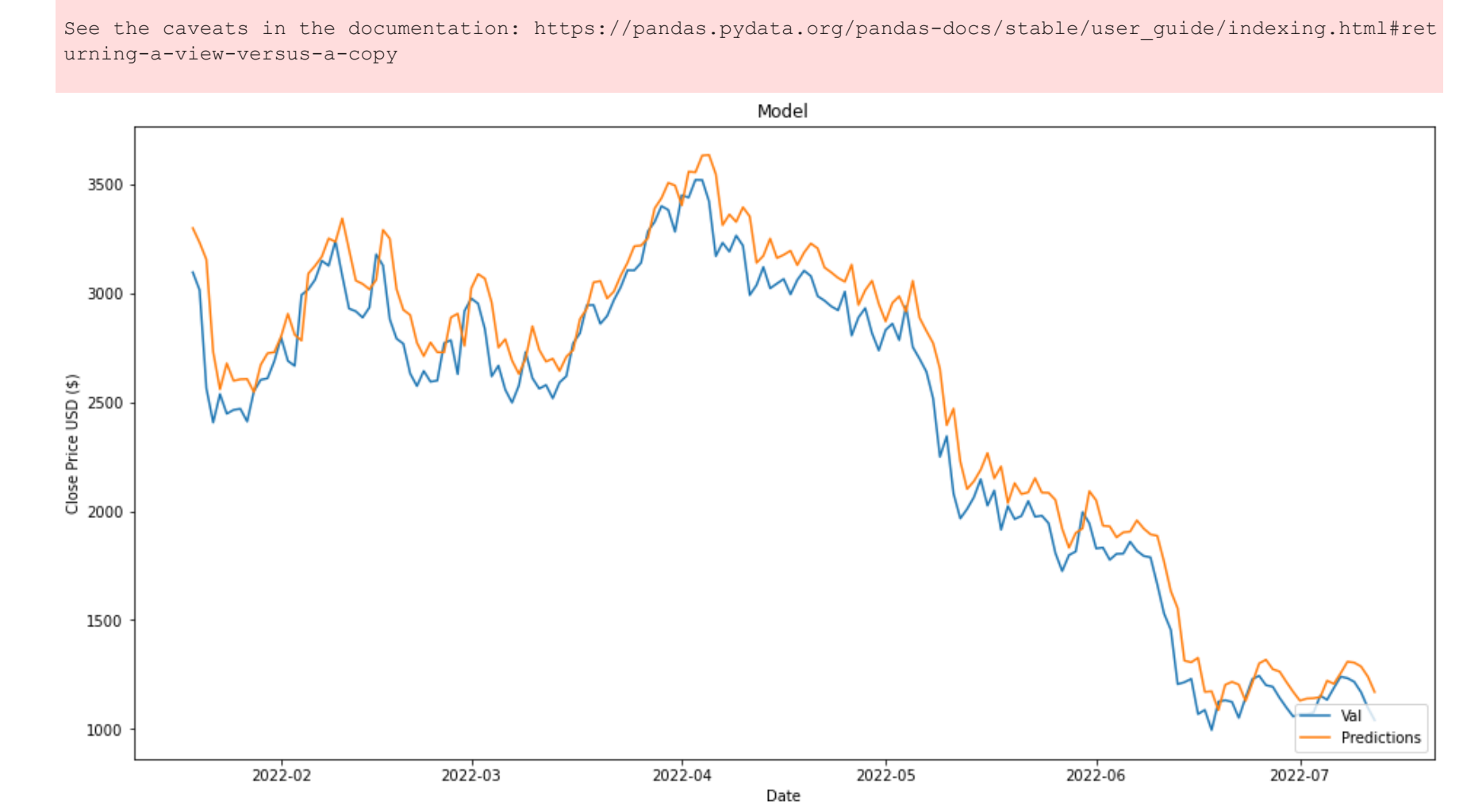
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_4 (LSTM)               (None, 60, 100)           40800

 lstm_5 (LSTM)               (None, 100)               80400

 dense_4 (Dense)             (None, 25)                2525

 dense_5 (Dense)             (None, 1)                 26

=================================================================
Total params: 123,751
Trainable params: 123,751
Non-trainable params: 0
_________________________________________________________________
Epoch 1/9
1556/1556 [==============================] - 27s 16ms/step - loss: 0.0022
Epoch 2/9
1556/1556 [==============================] - 24s 15ms/step - loss: 0.0015
Epoch 3/9
1556/1556 [==============================] - 24s 15ms/step - loss: 9.3127e-04
Epoch 4/9
1556/1556 [==============================] - 24s 15ms/step - loss: 7.9737e-04
Epoch 5/9
1556/1556 [==============================] - 24s 15ms/step - loss: 6.8517e-04
Epoch 6/9
1556/1556 [==============================] - 24s 15ms/step - loss: 0.0015
Epoch 7/9
1556/1556 [==============================] - 24s 15ms/step - loss: 5.1217e-04
Epoch 8/9
1556/1556 [==============================] - 24s 15ms/step - loss: 6.1351e-04
Epoch 9/9
1556/1556 [==============================] - 24s 15ms/step - loss: 5.8642e-04
6/6 [==============================] - 1s 20ms/step
49/49 [==============================] - 1s 19ms/step
rmse train:  49.46605953850972
rmse test:  126.33174959569926
r2_score train:  0.9934862629041088
r2_score test:  0.9473391030862127
```

In [21]:
```python
#bear
data = bear.filter(['ethereum'])
train = data[:training_data_len]
validation = data[training_data_len:]
validation['Predictions'] = predictions
plt.figure(figsize=(16,8))
plt.title('Model')
plt.xlabel('Date')
plt.ylabel('Close Price USD ($)')
plt.plot(train)
plt.plot(validation[['ethereum', 'Predictions']])
plt.legend(['Train', 'Val', 'Predictions'], loc='lower right')
plt.show()


data = bear.filter(['ethereum'])
train = data[:training_data_len]
validation = data[training_data_len:]
validation['Predictions'] = predictions
plt.figure(figsize=(16,8))
plt.title('Model')
plt.xlabel('Date')
plt.ylabel('Close Price USD ($)')
# plt.plot(train)
plt.plot(validation[['ethereum', 'Predictions']])
plt.legend(['Val', 'Predictions'], loc='lower right')
plt.show()
```

```
C:\Users\fooba\AppData\Local\Temp/ipykernel_21496/2836514432.py:5: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```




```
C:\Users\fooba\AppData\Local\Temp/ipykernel_21496/2836514432.py:19: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```



In [22]:
```python
#bear

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



validation['buy'] = np.where(validation['ethereum'] > validation['Predictions'], 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'])
```

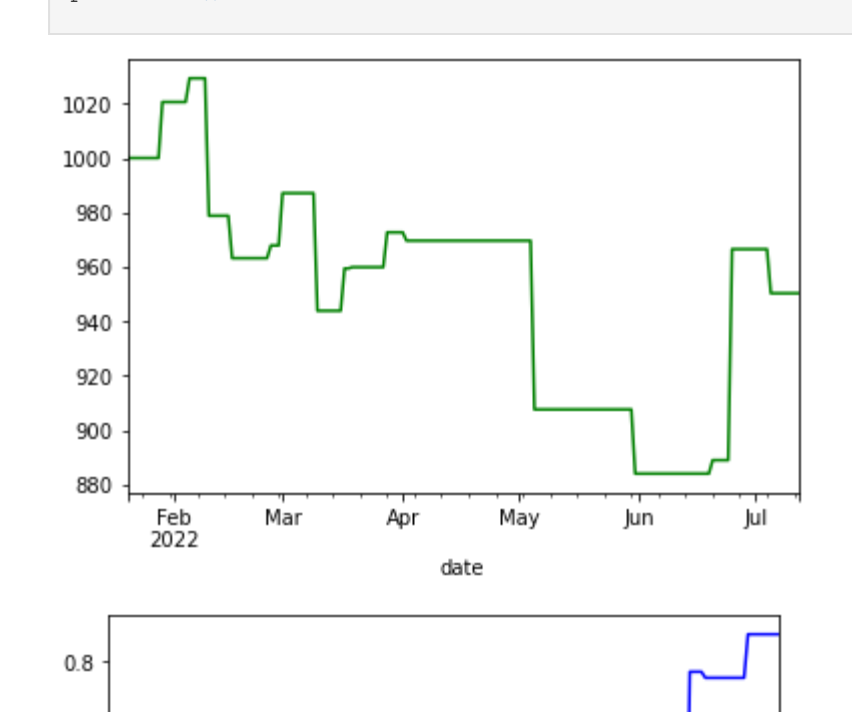
```
C:\Users\fooba\AppData\Local\Temp/ipykernel_21496/2362416721.py:9: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

C:\Users\fooba\AppData\Local\Temp/ipykernel_21496/2362416721.py:11: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

C:\Users\fooba\AppData\Local\Temp/ipykernel_21496/2362416721.py:12: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

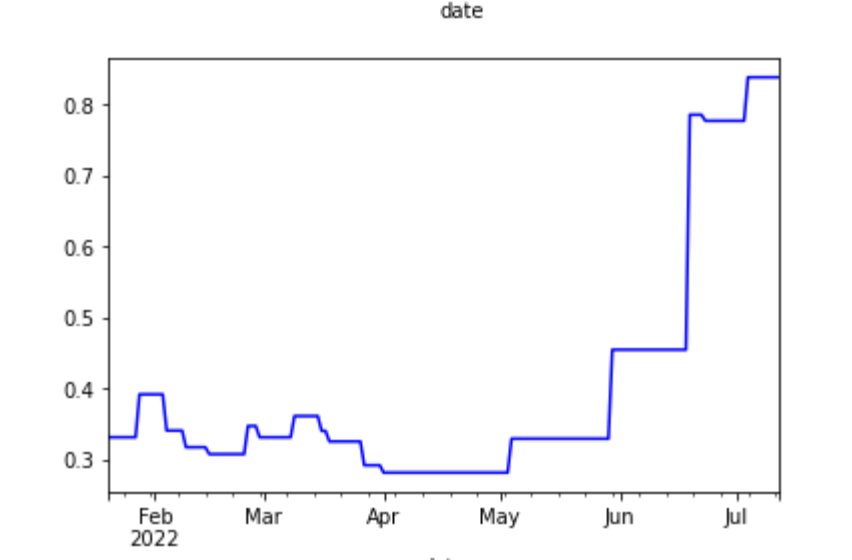
date	ethereum	Predictions	buy	signal	money	eth
2022-01-20	3015.588778	3233.729492	0	0.0	1000	3015.588778
2022-01-21	2564.343342	3155.584961	0	0.0	1000	3015.588778
2022-01-22	2407.377852	2731.641357	0	0.0	1000	3015.588778
2022-01-23	2537.836728	2559.915283	0	0.0	1000	3015.588778
2022-01-24	2447.831512	2678.958740	0	0.0	1000	3015.588778

date	ethereum	Predictions	buy	signal	money	eth
2022-01-20	3015.588778	3233.729492	0	0.0	1000.000000	0.331610
2022-01-21	2564.343342	3155.584961	0	0.0	1000.000000	0.331610
2022-01-22	2407.377852	2731.641357	0	0.0	1000.000000	0.331610
2022-01-23	2537.836728	2559.915283	0	0.0	1000.000000	0.331610
2022-01-24	2447.831512	2678.958740	0	0.0	1000.000000	0.331610
...	...	...	...	...	...	...
2022-07-08	1233.514679	1309.292603	0	0.0	950.322108	0.838014
2022-07-09	1216.849707	1304.855225	0	0.0	950.322108	0.838014
2022-07-10	1169.012708	1287.182495	0	0.0	950.322108	0.838014
2022-07-11	1097.449438	1240.715454	0	0.0	950.322108	0.838014
2022-07-12	1040.797146	1169.993530	0	0.0	950.322108	0.838014

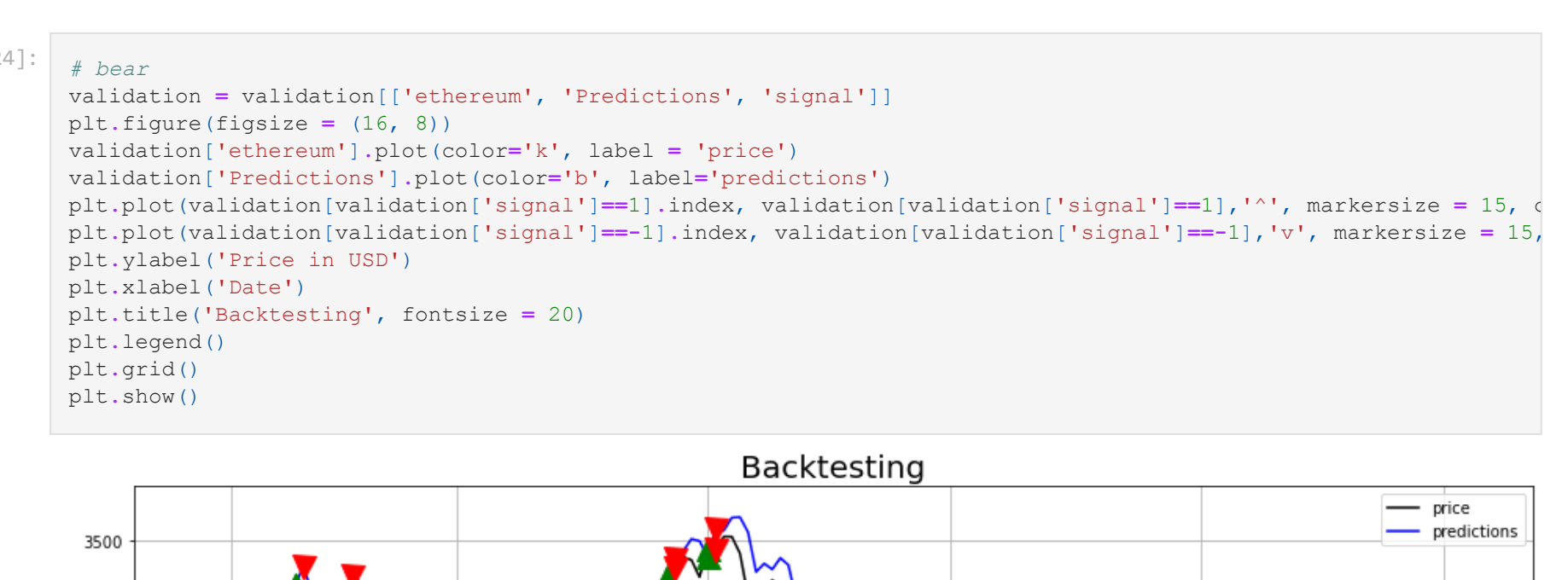
174 rows × 6 columns

In [23]:
```python
validation['money'].plot(color='g', label = 'money')
plt.show()

validation['eth'].plot(color='b', label = 'eth')
plt.show()
```





In [24]:
```python
# bear
validation = validation[['ethereum', 'Predictions', 'signal']]
plt.figure(figsize = (16, 8))
validation['ethereum'].plot(color='k', label = 'price')
validation['Predictions'].plot(color='b', label='predictions')
plt.plot(validation[validation['signal']==1].index, validation[validation['signal']==1],'^', markersize = 15, c
plt.plot(validation[validation['signal']==-1].index, validation[validation['signal']==-1],'v', markersize = 15,
plt.ylabel('Price in USD')
plt.xlabel('Date')
plt.title('Backtesting', fontsize = 20)
plt.legend()
plt.grid()
plt.show()
```



In [ ]:

In [ ]: