CryptoWall

cs50 final project by Atousa Toghyani

in this project:

1. Received ´API from https://coinmarketcap.com/ (https://coinmarketcap.com/) to display the top 100 coins.
2. Received dataset from **yfinance**
3. Analyze the Closing prices
4. Real-Time Crypto price
5. Predicting the closing crypto price with Long Short Term Memory ***(LSTM)***

Import the Libraries

In [1]:

```
# Import the Libraries
import pandas as pd
import numpy as np

# For time stamps
from datetime import datetime

# To draw a diagram
import matplotlib.pyplot as plt
plt.style.use("fivethirtyeight")
%matplotlib inline

# To draw a hit map
import seaborn as sns
sns.set_style('whitegrid')

# For Get API from coinmarketcap.com
import time
import json
from bs4 import BeautifulSoup
from requests import Request, Session

# To get information from Yahoo
import yfinance as yf

# For the LSTM
import math
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM
```

Download stock data from yahoo then export as CSV

In [2]:

```
# Set up End and Start times for data grab
end = datetime.now()
start = datetime(end.year - 1, end.month, end.day)

# Download stock data then export as CSV
df_btc = yf.download("BTC-USD", start, end)
df_btc.to_csv('bitcoin.csv')

df_eth = yf.download("ETH-USD", start, end)
df_eth.to_csv('ethereum.csv')

df_usdt = yf.download("USDT-USD", start, end)
df_usdt.to_csv('tether.csv')

df_ada = yf.download("ADA-USD", start, end)
df_ada.to_csv('cardano.csv')

df_bnb = yf.download("BNB-USD", start, end)
df_bnb.to_csv('binance-coin.csv')
```

```
[**********************100%**********************]  1 of 1 compl
eted
[**********************100%**********************]  1 of 1 compl
eted
[**********************100%**********************]  1 of 1 compl
eted
[**********************100%**********************]  1 of 1 compl
eted
[**********************100%**********************]  1 of 1 compl
eted
```

Read dataset

In [3]:

```
# read google dataset
df = pd.read_csv('bitcoin.csv')
df.head()
```

Out[3]:

	Date	Open	High	Low	Close	Adj Close
0	2020-10-01	10785.010742	10915.843750	10493.552734	10623.330078	10623.330078
1	2020-10-02	10624.390625	10662.813477	10440.311523	10585.164062	10585.164062
2	2020-10-03	10583.806641	10614.091797	10527.978516	10565.493164	10565.493164
3	2020-10-04	10567.919922	10700.791016	10531.342773	10684.428711	10684.428711
4	2020-10-05	10688.034180	10804.000977	10646.443359	10804.000977	10804.000977

Analyze the closing prices from dataset:

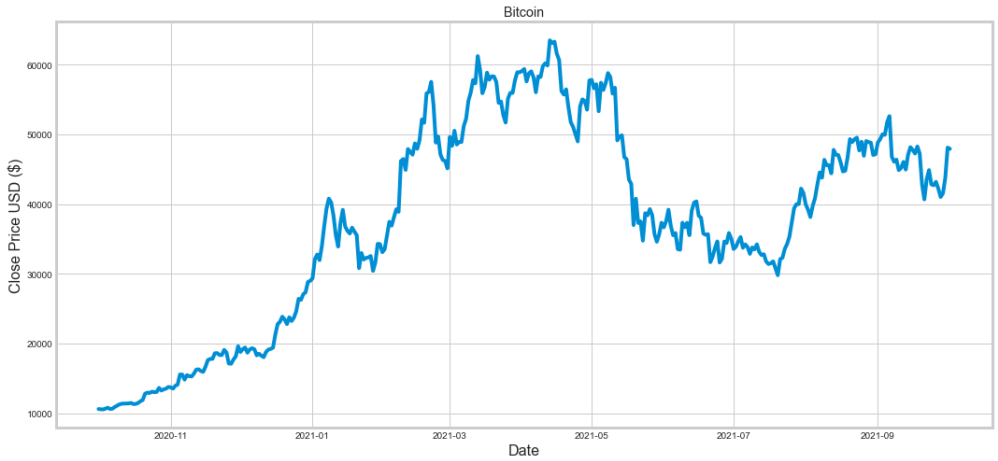
In [4]:

```
# Plot stock company['Close']
def figure_close(stockname , name):
    df = pd.read_csv(stockname)
    df["Date"] = pd.to_datetime(df.Date,format="%Y-%m-%d")
    df.index = df['Date']

    fig = plt.figure(figsize=(16,8))
    ax = fig.add_subplot()
    ax.set_title(name)
    ax.set_xlabel('Date', fontsize=16)
    ax.set_ylabel('Close Price USD ($)', fontsize=16)
    plt.plot(df["Close"],label='Close Price history')
```

Bitcoin

In [5]:

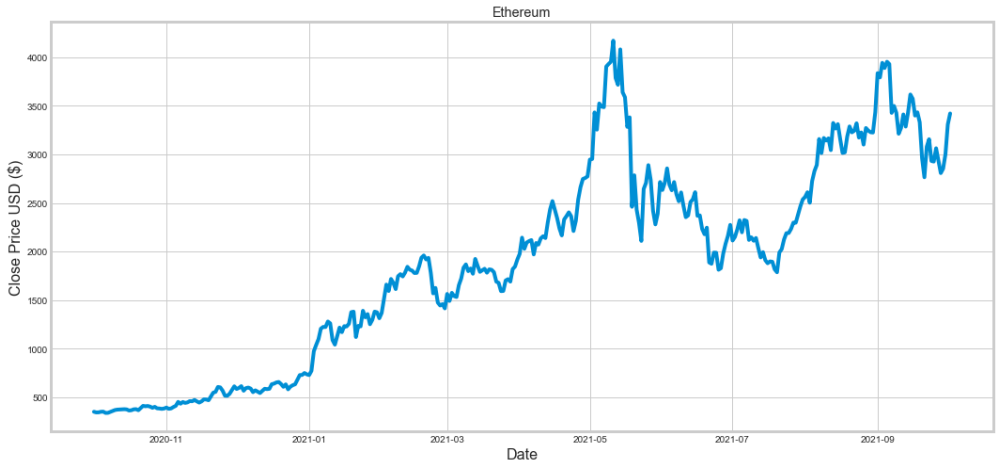
```
figure_close('bitcoin.csv' , 'Bitcoin')
```



Ethereum

In [6]:

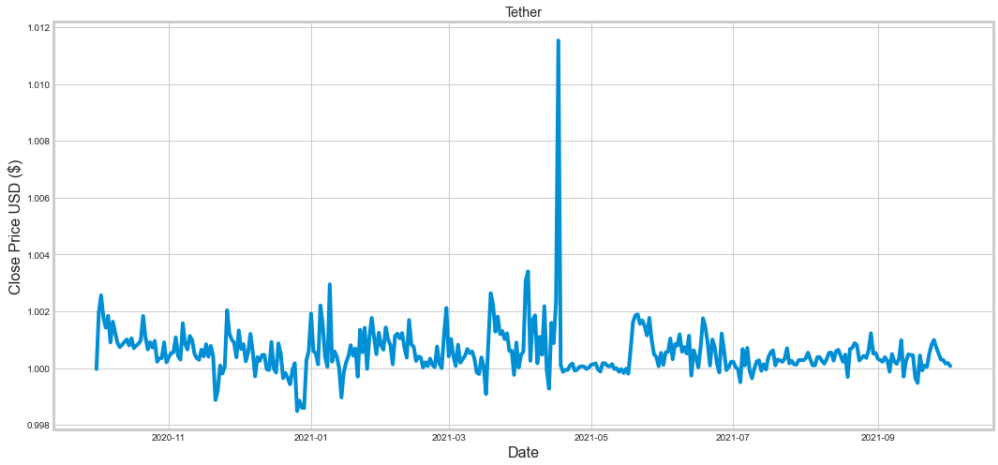
```
figure_close('ethereum.csv' , 'Ethereum')
```



Tether

In [7]:

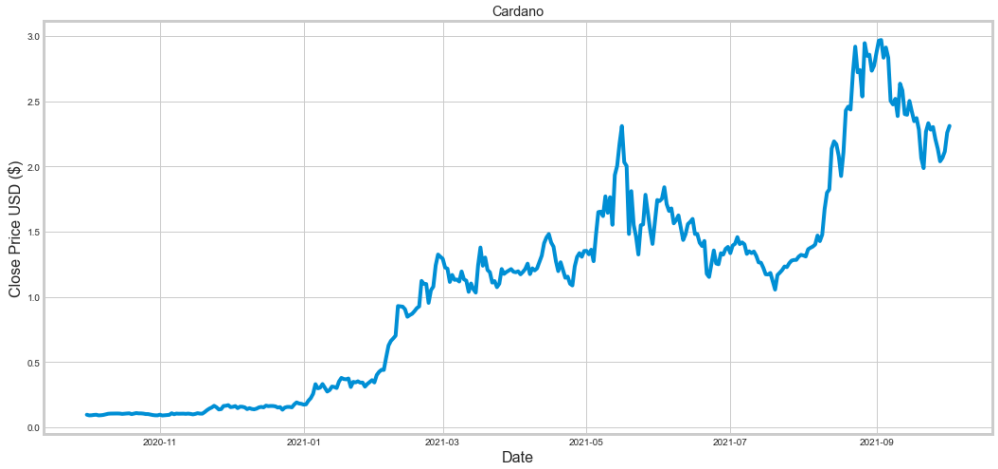
```
figure_close('tether.csv' , 'Tether')
```



Cardano

In [8]:

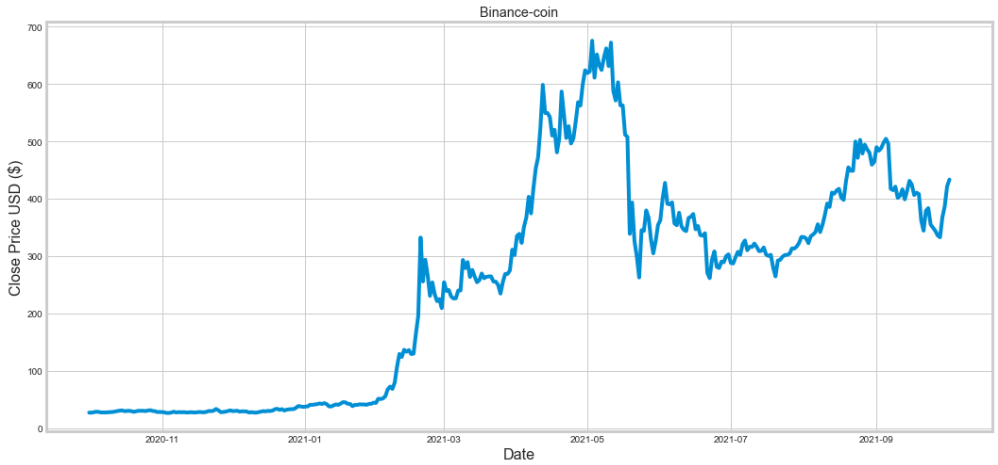
```
figure_close('cardano.csv' , 'Cardano')
```



Binance-coin

In [9]:

```
figure_close('binance-coin.csv' , 'Binance-coin')
```



Use sebron for a quick correlation plot for the daily returns

In [10]:

```
# The tech stocks we'll use for this analysis
coin_list = ['BTC-USD', 'ETH-USD', 'USDT-USD', 'ADA-USD', 'BNB-USD']

# Download stock data then export as CSV
df_close = yf.download(coin_list, start, end)['Adj Close']
df_close.to_csv('closing.csv')
```

```
[**********100%**********]  5 of 5 compl
eted
```

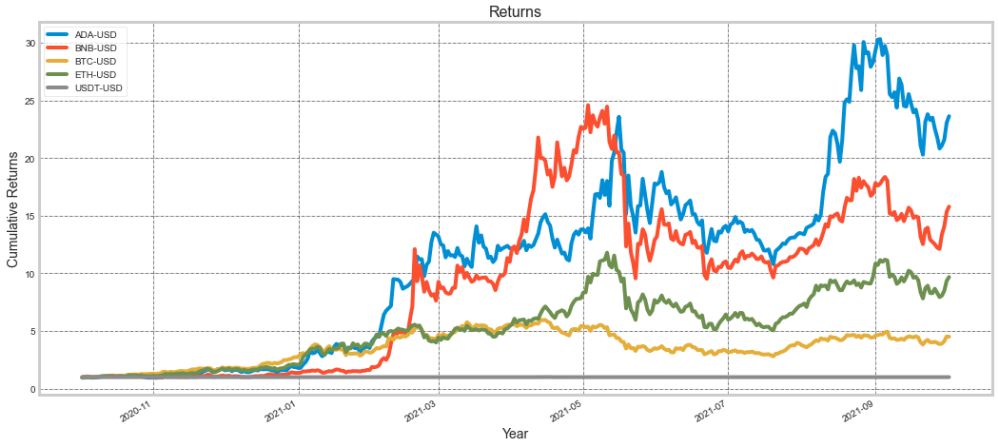
In [11]:

```
print(df_close.head())
```

```
             ADA-USD    BNB-USD       BTC-USD     ETH-USD  USDT-
USD
Date
2020-10-01  0.097878  27.470755  10623.330078  353.231293  0.999
962
2020-10-02  0.092913  27.320980  10585.164062  346.532654  1.001
953
2020-10-03  0.093684  28.271854  10565.493164  347.321594  1.002
561
2020-10-04  0.096301  29.064274  10684.428711  353.121918  1.001
763
2020-10-05  0.097687  28.658798  10804.000977  354.277100  1.001
419
```

In [12]:

```
# Plot all the close prices
((df_close.pct_change()+1).cumprod()).plot(figsize=(16,8))

# Show the legend
plt.legend()

# Define the label for the title of the figure
plt.title("Returns", fontsize=16)
plt.ylabel('Cumulative Returns', fontsize=14)
plt.xlabel('Year', fontsize=14)

# Plot the grid lines
plt.grid(which="major", color='k', linestyle='-.', linewidth=0.5)
plt.show()
```



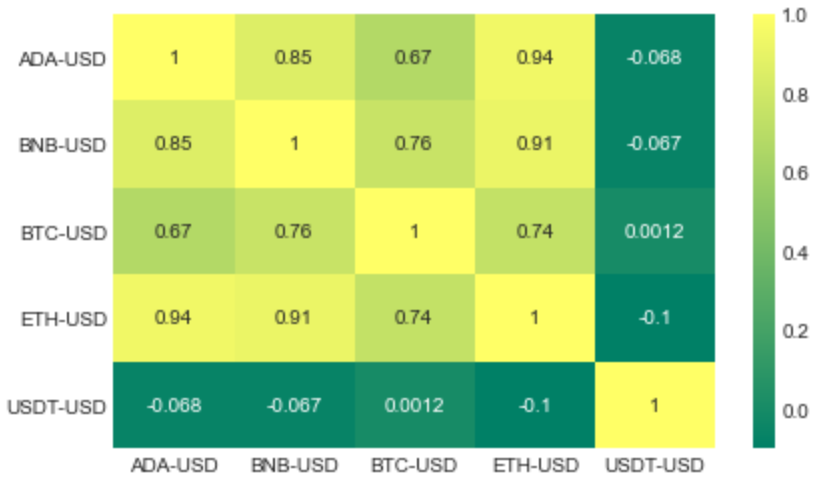
Heatmap

In [13]:

```
sns.heatmap(df_close.corr(), annot=True, cmap='summer')
```

Out[13]:

<AxesSubplot:>



Get API from coinmarketcap.com

In [14]:

```
url = 'https://pro-api.coinmarketcap.com/v1/cryptocurrency/listings/latest'
parameters = {
  'start':'1',
  'limit':'100',
  'convert':'USD'
}
headers = {
  'Accepts': 'application/json',
  'X-CMC_PRO_API_KEY': 'd50d92c4-c400-4e06-9ab7-b630f977e94c',
}

session = Session()
session.headers.update(headers)

response = session.get(url, params=parameters)
json = json.loads(response.text)
data = json['data']

coins ={}
for x in data:
    coins[str(x['id'])] = x['slug']
```

In [15]:

```
print(data)
```

```
[{'id': 1, 'name': 'Bitcoin', 'symbol': 'BTC', 'slug': 'bitcoi
n', 'num_market_pairs': 8563, 'date_added': '2013-04-28T00:00:
00.000Z', 'tags': ['mineable', 'pow', 'sha-256', 'store-of-val
ue', 'state-channels', 'coinbase-ventures-portfolio', 'three-a
rrows-capital-portfolio', 'polychain-capital-portfolio', 'bina
nce-labs-portfolio', 'arrington-xrp-capital', 'blockchain-capi
tal-portfolio', 'boostvc-portfolio', 'cms-holdings-portfolio',
'dcg-portfolio', 'dragonfly-capital-portfolio', 'electric-capi
tal-portfolio', 'fabric-ventures-portfolio', 'framework-ventur
es', 'galaxy-digital-portfolio', 'huobi-capital', 'alameda-res
earch-portfolio', 'a16z-portfolio', '1confirmation-portfolio',
'winklevoss-capital', 'usv-portfolio', 'placeholder-ventures-p
ortfolio', 'pantera-capital-portfolio', 'multicoin-capital-por
tfolio', 'paradigm-xzy-screener'], 'max_supply': 21000000, 'ci
rculating_supply': 18832712, 'total_supply': 18832712, 'platfo
rm': None, 'cmc_rank': 1, 'last_updated': '2021-10-02T16:49:0
2.000Z', 'quote': {'USD': {'price': 47944.447615552024, 'volum
e_24h': 31487065096.713127, 'percent_change_1h': 0.07146898,
'percent_change_24h': 0.94591301, 'percent_change_7d': 12.3103
```

Convert json data to CSV file

In [16]:

```
coin_name = []
coin_symbol = []
market_cap = []
percent_change_1h = []
percent_change_24h = []
percent_change_7d = []
price = []
volume_24h = []

for i in data:
    coin_name.append(i['slug'])
    coin_symbol.append(i['symbol'])
    price.append(i['quote']['USD']['price'])
    percent_change_1h.append(i['quote']['USD']['percent_change_1h'])
    percent_change_24h.append(i['quote']['USD']['percent_change_24h'])
    percent_change_7d.append(i['quote']['USD']['percent_change_7d'])
    market_cap.append(i['quote']['USD']['market_cap'])
    volume_24h.append(i['quote']['USD']['volume_24h'])

df = pd.DataFrame(columns=['coin_name', 'coin_symbol', 'market_cap', 'percent_c
df['coin_name'] = coin_name
df['coin_symbol'] = coin_symbol
df['price'] = price
df['percent_change_1h'] = percent_change_1h
df['percent_change_24h'] = percent_change_24h
df['percent_change_7d'] = percent_change_7d
df['market_cap'] = market_cap
df['volume_24h'] = volume_24h

df.to_csv( "cryptocurrencies.csv", index=False, encoding='utf-8-sig')
```

Read cryptocurrencies dataset

In [17]:

```
df = pd.read_csv("cryptocurrencies.csv")
df
```

Out[17]:

	coin_name	coin_symbol	market_cap	percent_change_1h	percent_change_
0	bitcoin	BTC	9.029240e+11	0.071469	0.945
1	ethereum	ETH	4.028647e+11	0.792412	4.887
2	cardano	ADA	7.413698e+10	2.060271	3.869
3	binance-coin	BNB	7.295901e+10	0.175055	4.176
4	tether	USDT	6.802815e+10	-0.004800	-0.010
...	...	...	...	...	
95	basic-attention-token	BAT	1.048329e+09	0.147966	4.769
96	iostoken	IOST	1.044709e+09	-0.584339	2.568
97	audius	AUDIO	9.989462e+08	2.359198	4.11'
98	paxos-standard	USDP	9.450382e+08	-0.042433	-0.043
99	telcoin	TEL	9.445797e+08	1.289837	3.688

100 rows × 8 columns

Real-Time Crypto Price

In [18]:

```
def live_price(coin_name):
    for i in data:
        name = i['slug']
        if name == coin_name:
            price = i['quote']['USD']['price']
            #change_price = i['quote']['USD']['percent_change_1h']
            print(coin_name,"live crypto price: " , price)

print(datetime.now())
print()
live_price('bitcoin')
live_price('ethereum')
live_price('tether')
live_price('cardano')
live_price('binance-coin')
```

```
2021-10-02 20:20:33.671098

bitcoin live crypto price:  47944.447615552024
ethereum live crypto price:  3420.929875935483
tether live crypto price:  1.0000599748936
cardano live crypto price:  2.31402538926412
binance-coin live crypto price:  433.9258646740517
```

Percent Change Crypto Price

In [19]:

```
def delta(coin_name):
    for i in data:
        name = i['slug']
        if name == coin_name:
            change_price = i['quote']['USD']['percent_change_1h']
            print(coin_name,"Percent Change 1h: " , change_price)

print(datetime.now())
print()
delta('bitcoin')
delta('ethereum')
delta('tether')
delta('cardano')
delta('binance-coin')
```

```
2021-10-02 20:20:37.967229

bitcoin Percent Change 1h:  0.07146898
ethereum Percent Change 1h:  0.79241204
tether Percent Change 1h:  -0.00480015
cardano Percent Change 1h:  2.06027124
binance-coin Percent Change 1h:  0.17505472
```

Get dataset from yfinance for historical data of cryptocurrencies

In [20]:

```
# create empty dataframe
coin_final = pd.DataFrame()
Symbols = ['BTC-USD', 'ETH-USD', 'USDT-USD', 'ADA-USD', 'BNB-USD']

# Set up End and Start times for data grab
end = datetime.now()
start = datetime(end.year - 1, end.month, end.day)

for i in Symbols:
    # print the symbol which is being downloaded
    print( str(Symbols.index(i)) + str(' : ') + i, sep=',', end=',', flush=True

    try:
        # download the stock price
        coin = []
        coin = yf.download(i,start=start, end=end, progress=False)

        # append the individual stock prices
        if len(coin) == 0:
            None
        else:
            coin['Name']=i
            coin_final = coin_final.append(coin,sort=False)
    except Exception:
        None
# convert to csv file
coin_final.to_csv('5coins.csv')
```

```
0 : BTC-USD,1 : ETH-USD,2 : USDT-USD,3 : ADA-USD,4 : BNB-USD,
```

Read 5coins dataset

In [21]:

```
df = pd.read_csv('5coins.csv')
df
```

Out[21]:

	Date	Open	High	Low	Close	Adj Clo
0	2020-10-01	10785.010742	10915.843750	10493.552734	10623.330078	10623.330
1	2020-10-02	10624.390625	10662.813477	10440.311523	10585.164062	10585.164
2	2020-10-03	10583.806641	10614.091797	10527.978516	10565.493164	10565.493
3	2020-10-04	10567.919922	10700.791016	10531.342773	10684.428711	10684.428
4	2020-10-05	10688.034180	10804.000977	10646.443359	10804.000977	10804.000
...	...	...	...	...	...	
1815	2021-09-28	335.807556	344.275574	330.073853	333.032593	333.032
1816	2021-09-29	333.397919	375.006866	331.448517	367.989594	367.989
1817	2021-09-30	367.786011	388.268768	366.644257	387.057343	387.057
1818	2021-10-01	387.635986	423.344269	382.131378	421.643188	421.643
1819	2021-10-02	420.537048	434.209656	411.038330	434.209656	434.209

1820 rows × 8 columns

Predicting the closing crypto price with LSTM

Build and train the LSTM model

In [24]:

```
# For Warning
pd.options.mode.chained_assignment = None  # default='warn'

def LSTM_Model(stockname):
    # read the stock Price
    df = pd.read_csv(stockname)
    df["Date"] = pd.to_datetime(df.Date,format="%Y-%m-%d")
    df.index = df['Date']

    # Create a new dataframe with only the 'Close' column
    data = df.filter(['Close'])

    # Converting the dataframe to a numpy array
    dataset = data.values

    # Get /Compute the number of rows to train the model on
    training_data_len = math.ceil( len(dataset) *.8)

    # Scale the all of the data to be values between 0 and 1
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled_data = scaler.fit_transform(dataset)

    # Create the scaled training data set
    train_data = scaled_data[0:training_data_len  , : ]

    # Split the data into x_train and y_train data sets
    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])

    # Convert x_train and y_train to numpy arrays
    x_train, y_train = np.array(x_train), np.array(y_train)

    # Reshape the data into the shape accepted by the LSTM
    x_train = np.reshape(x_train, (x_train.shape[0],x_train.shape[1],1))

    # Build the LSTM network model
    model = Sequential()
    model.add(LSTM(units=50, return_sequences=True,input_shape=(x_train.shape[1
    model.add(LSTM(units=50, return_sequences=False))
    model.add(Dense(units=25))
    model.add(Dense(units=1))

    # Compile the model
    model.compile(optimizer='adam', loss='mean_squared_error')
```

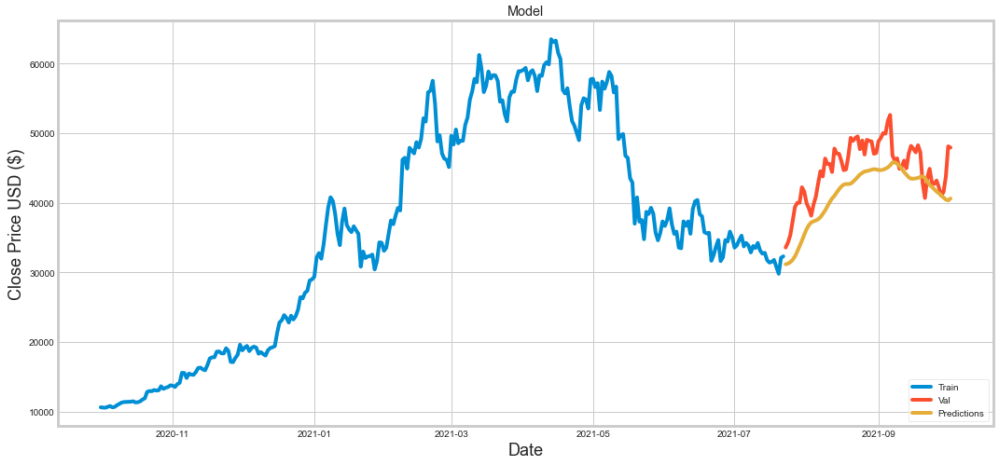
```
# Train the model
model.fit(x_train, y_train, batch_size=1, epochs=1)

# Test data set
test_data = scaled_data[training_data_len - 60: , : ]

# Create the x_test and y_test data sets
x_test = []


y_test =  dataset[training_data_len : , : ]

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

# Convert x_test to a numpy array
x_test = np.array(x_test)

# Reshape the data into the shape accepted by the LSTM
x_test = np.reshape(x_test, (x_test.shape[0],x_test.shape[1],1))

# Getting the models predicted price values
predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)#Undo scaling

# Calculate/Get the value of RMSE
rmse = np.sqrt(np.mean(((predictions- y_test)**2)))
print("rmse: ",rmse)

# Plot/Create the data for the graph
train = data[:training_data_len]
valid = data[training_data_len:]
valid['Predictions'] = predictions

#Visualize the data
plt.figure(figsize=(16,8))
plt.title('Model')
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price USD ($)', fontsize=18)
plt.plot(train['Close'])
plt.plot(valid[['Close', 'Predictions']])
plt.legend(['Train', 'Val', 'Predictions'], loc='lower right')
plt.show()
```

Bitcoin cryptocurrency forecast

In [25]:

```
LSTM_Model('bitcoin.csv')
```

```
232/232 [==============================] - 43s 56ms/step - loss:
0.0325
rmse:  4419.873605656252
```

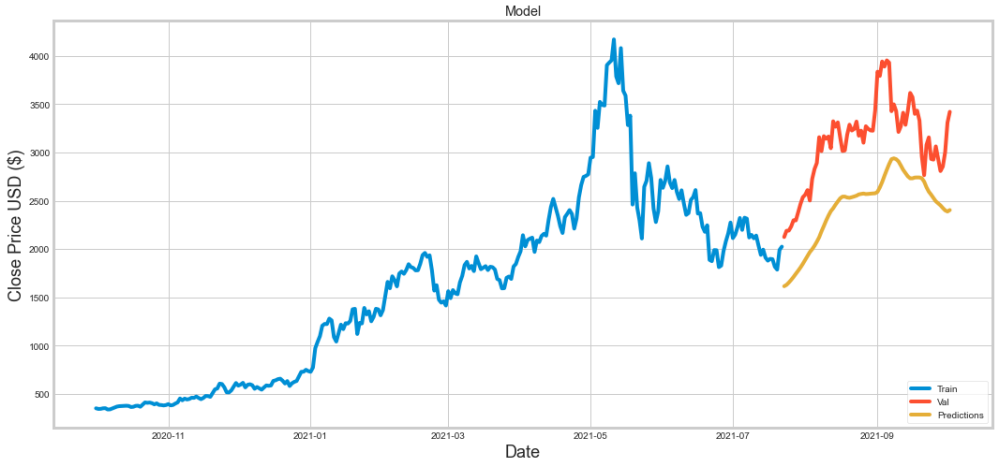


Ethereum cryptocurrency forecast

In [26]:

```
LSTM_Model('ethereum.csv')
```

```
232/232 [==============================] - 18s 45ms/step - loss:
0.0280
rmse:  712.5095492754953
```

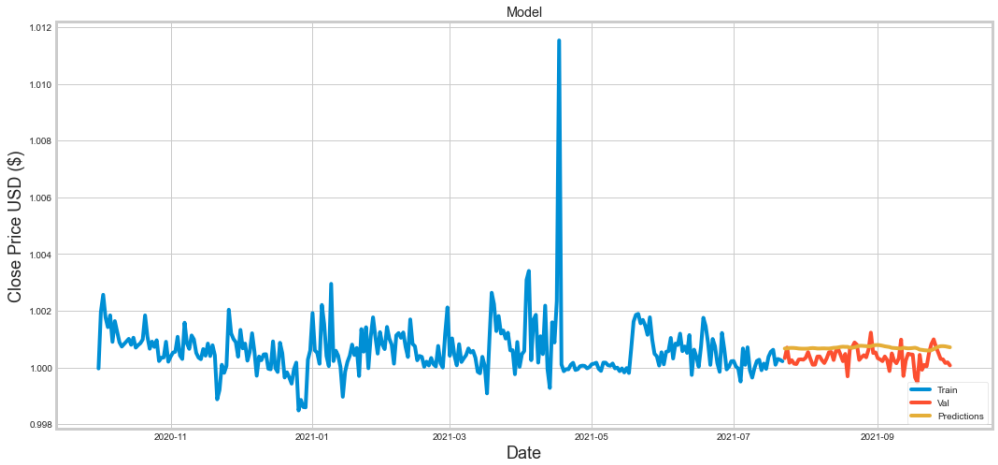


Tether cryptocurrency forecast

In [27]:

```
LSTM_Model('tether.csv')
```

```
232/232 [==============================] - 17s 47ms/step - loss:
0.0056
rmse:  0.0004613554433591186
```

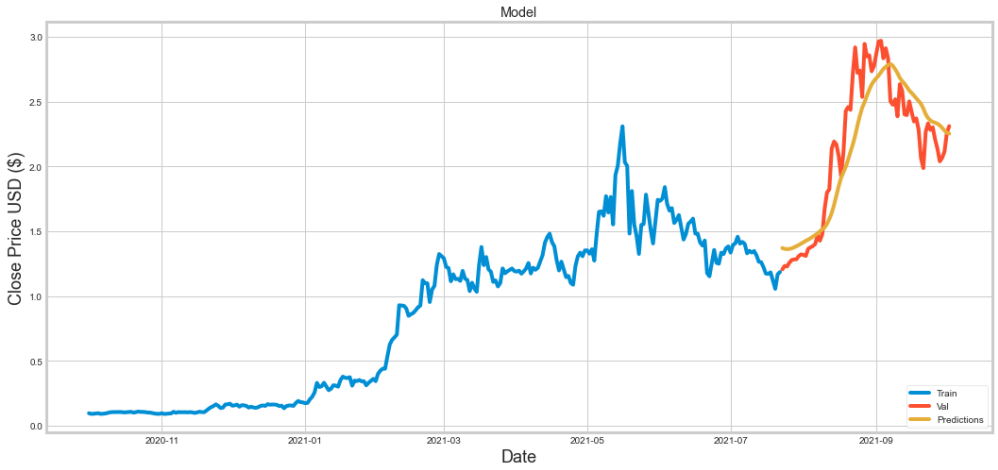


Cardano cryptocurrency forecast

In [28]:

```
LSTM_Model('cardano.csv')
```

```
232/232 [==============================] - 17s 47ms/step - loss:
0.0195
rmse:  0.24471896244942554
```



Binance-coin cryptocurrency forecast

In [29]:

```
LSTM_Model('binance-coin.csv')
```

```
232/232 [==============================] - 17s 47ms/step - loss:
0.0262 0s - loss:  - ETA: 0s - loss: 0.0
WARNING:tensorflow:5 out of the last 13 calls to <function Mode
l.make_predict_function.<locals>.predict_function at 0x000002361
6DE2CA0> triggered tf.function retracing. Tracing is expensive a
nd the excessive number of tracings could be due to (1) creating
@tf.function repeatedly in a loop, (2) passing tensors with diff
erent shapes, (3) passing Python objects instead of tensors. For
(1), please define your @tf.function outside of the loop. For
(2), @tf.function has experimental_relax_shapes=True option that
relaxes argument shapes that can avoid unnecessary retracing. Fo
r (3), please refer to https://www.tensorflow.org/guide/function
#controlling_retracing (https://www.tensorflow.org/guide/functio
n#controlling_retracing) and https://www.tensorflow.org/api_doc
s/python/tf/function (https://www.tensorflow.org/api_docs/pytho
n/tf/function) for  more details.
rmse:  48.450938437571494
```