UCDPA_ShivankGarg

September 17, 2022

1 Real World Scenerio to study cryptocurrency and importing the required Dataset from https://www.cryptodatadownload.com

[1]:
```
import pandas as pd
import functools as ft
import seaborn as sns
import numpy as np
import re

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error,␣
 ↪explained_variance_score, r2_score
```

2 Import a CSV file into a Pandas DataFrame

[2]:
```
#Importing data
dataFrame_ada = pd.read_csv('./Dataset/Binance_ADAUSDT_1h.csv')
dataFrame_bnb = pd.read_csv('./Dataset/Binance_BNBUSDT_1h.csv')
dataFrame_btc = pd.read_csv('./Dataset/Binance_BTCUSDT_1h.csv')
dataFrame_eth = pd.read_csv('./Dataset/Binance_ETHUSDT_1h.csv')
dataFrame_xrp = pd.read_csv('./Dataset/Binance_XRPUSDT_1h.csv')
```

3 Fetch/Importing Data from API

4 Empty List - Using Dictionary or Lists

5 Regex to find all the States starting with M

6 Make use of iterators for listing all the States in Data

[3]:
```
# Fetch/Importing Data from API

import requests
import json
```

```
import re

#Empty List - Using Dictionary or Lists
thislist = []


response = requests.get("https://api.covid19india.org/state_district_wise.json")

print('API Status Response: ', response)
#print("--------------------------------------------------------------------------------")
#print('API Data Response: ', response.json())
response_data = json.loads(response.text)


txt = "The rain in Spain"
x = re.search("^The.*Spain$", txt)

#Make use of iterators for listing all the States in Data
print("--------------------------------List of States in␣
↪Data---------------------------------------")
for i in response_data:
    print(i)

    #Regex to find all the States starting with M
    x = re.findall("^M", i)
    if x:
      thislist.append(i)
   ␣
  ↪#print("--------------------------------------------------------------------------------")
```

```
API Status Response:  <Response [200]>
--------------------------------List of States in
Data---------------------------------------
State Unassigned
Andaman and Nicobar Islands
Andhra Pradesh
Arunachal Pradesh
Assam
Bihar
Chandigarh
Chhattisgarh
Delhi
Dadra and Nagar Haveli and Daman and Diu
Goa
Gujarat
Himachal Pradesh
Haryana
Jharkhand
```

```
Jammu and Kashmir
Karnataka
Kerala
Ladakh
Lakshadweep
Maharashtra
Meghalaya
Manipur
Madhya Pradesh
Mizoram
Nagaland
Odisha
Punjab
Puducherry
Rajasthan
Sikkim
Telangana
Tamil Nadu
Tripura
Uttar Pradesh
Uttarakhand
West Bengal
```

[4]:
```
#Print list filtered using regex to find the state name starting with M
print(thislist)
```

```
['Maharashtra', 'Meghalaya', 'Manipur', 'Madhya Pradesh', 'Mizoram']
```

[5]:
```
ada_head = dataFrame_ada.head()
print("Cardano Data :: {}".format(ada_head))
print("--------------------------------------------------------------------------------")
bnb_head = dataFrame_bnb.head()
print("BNB Data :: {}".format(bnb_head))
print("--------------------------------------------------------------------------------")
btc_head = dataFrame_btc.head()
print("Bitcoin Data :: {}".format(btc_head))
print("--------------------------------------------------------------------------------")
eth_head = dataFrame_eth.head()
print("Ethereum Data :: {}".format(eth_head))
print("--------------------------------------------------------------------------------")
xrp_head = dataFrame_xrp.head()
print("Ripple Data :: {}".format(xrp_head))
```

```
Cardano Data ::
https://www.CryptoDataDownload.com
unix          date                symbol    open       high       low
close      Volume ADA       Volume USDT                              tradecount
1660176000000 2022-08-11 00:00:00 ADA/USDT 0.53740000 0.54470000 0.53700000
0.54470000 6894009.50000000 3728764.58708000                               8040
```

```
1660172400000 2022-08-10 23:00:00 ADA/USDT 0.53520000 0.53750000 0.53490000
0.53740000 2351540.80000000 1261238.77354000                              2176
1660168800000 2022-08-10 22:00:00 ADA/USDT 0.53610000 0.53730000 0.53430000
0.53520000 3508178.60000000 1879794.96216000                              3675
1660165200000 2022-08-10 21:00:00 ADA/USDT 0.53470000 0.53660000 0.53150000
0.53610000 6149815.00000000 3285007.45185000                              5959
-------------------------------------------------------------------------------
BNB Data ::
https://www.CryptoDataDownload.com
unix          date                symbol   open         high         low
close        Volume BNB     Volume USDT                              tradecount
1660176000000 2022-08-11 00:00:00 BNB/USDT 328.80000000 330.00000000
328.50000000 329.90000000 7048.58000000  2320515.08690000
3750
1660172400000 2022-08-10 23:00:00 BNB/USDT 327.20000000 328.90000000
327.00000000 328.80000000 11312.88100000 3708339.38320000
6223
1660168800000 2022-08-10 22:00:00 BNB/USDT 329.30000000 329.70000000
326.80000000 327.30000000 13160.50600000 4317169.56490000
7762
1660165200000 2022-08-10 21:00:00 BNB/USDT 327.70000000 330.50000000
326.10000000 329.20000000 20409.95500000 6699728.36470000
12387
-------------------------------------------------------------------------------
Bitcoin Data ::
https://www.CryptoDataDownload.com
unix          date                symbol   open           high           low
close          Volume BTC    Volume USDT
tradecount
1660521600000 2022-08-15 00:00:00 BTC/USDT 24305.25000000 24316.56000000
24164.10000000 24261.60000000 3638.40299000 88222299.34225000
124977
1660518000000 2022-08-14 23:00:00 BTC/USDT 24257.90000000 24353.10000000
24234.07000000 24305.24000000 4931.96334000 119823412.83479350
166290
1660514400000 2022-08-14 22:00:00 BTC/USDT 24344.88000000 24364.40000000
24172.40000000 24258.68000000 6704.19848000 162788277.62054590
221345
1660510800000 2022-08-14 21:00:00 BTC/USDT 24313.51000000 24435.00000000
24273.94000000 24343.67000000 4309.37601000 104928470.60300340
156977
-------------------------------------------------------------------------------
Ethereum Data ::
https://www.CryptoDataDownload.com
unix          date                symbol   open          high          low
close         Volume ETH     Volume USDT
tradecount
1660176000000 2022-08-11 00:00:00 ETH/USDT 1853.58000000 1865.00000000
```

```
1850.32000000 1859.76000000 12898.12260000 23948851.55936400
14377
1660172400000 2022-08-10 23:00:00 ETH/USDT 1850.00000000 1855.47000000
1846.56000000 1853.57000000 13972.14190000 25860915.10931200
20804
1660168800000 2022-08-10 22:00:00 ETH/USDT 1860.69000000 1869.27000000
1845.70000000 1849.99000000 29681.34170000 55101016.17500600
40892
1660165200000 2022-08-10 21:00:00 ETH/USDT 1842.50000000 1885.00000000
1828.88000000 1860.68000000 63307.62550000 117645449.39901800
98796
------------------------------------------------------------------------------
Ripple Data ::
https://www.CryptoDataDownload.com
unix          date                 symbol   open       high       low
close      Volume XRP      Volume USDT                               tradecount
1660176000000 2022-08-11 00:00:00 XRP/USDT 0.38140000 0.38600000 0.38120000
0.38460000 7277015.00000000 2787123.22760000                               5295
1660172400000 2022-08-10 23:00:00 XRP/USDT 0.38010000 0.38150000 0.37940000
0.38140000 5532299.00000000 2104535.66780000                               3851
1660168800000 2022-08-10 22:00:00 XRP/USDT 0.37970000 0.38130000 0.37920000
0.38020000 7117137.00000000 2706793.66710000                               4765
1660165200000 2022-08-10 21:00:00 XRP/USDT 0.37880000 0.38070000 0.37730000
0.37970000 9665012.00000000 3661335.30550000                               6288
```

7 Define a custom function to create reusable code

[6]:
```
#Define a custom function to create reusable code

def remove_columns(df):

    df.reset_index(inplace=True)

    df.drop(columns=['level_0', 'level_7', 'level_8', 'https://www.
 ↪CryptoDataDownload.com'], axis=1,inplace=True)

    dict = {'level_1':'date',
        'level_2':'symbol',
        'level_3':'open',
        'level_4':'high',
        'level_5':'low',
        'level_6':'close'}
    df.rename(columns = dict, inplace = True)

    df.drop(df.index[0], inplace = True)
```

8 Checking/Removing missing values or drop duplicates

[7]:
```
#Checking/Removing missing values or drop duplicates
def data_check(df):

    print('If Dataframe has any null values :: {}' .format(df.isnull().values.
 ↪any()))
    print('Number of duplicate values in Dataframe :: {}' .format(df.
 ↪duplicated().sum()))
    if(df.duplicated().sum() > 0):
        df.drop_duplicates(inplace=True)
        print('Number of duplicate values after removing in Dataframe :: {}' .
 ↪format(df.duplicated().sum()))
```

[8]:
```
remove_columns(dataFrame_ada)
remove_columns(dataFrame_bnb)
remove_columns(dataFrame_btc)
remove_columns(dataFrame_eth)
remove_columns(dataFrame_xrp)
```

[9]:
```
print('-----ADA Data Check-----')
data_check(dataFrame_ada)
print('-----BNB Data Check-----')
data_check(dataFrame_bnb)
print('-----BTC Data Check-----')
data_check(dataFrame_btc)
print('-----ETH Data Check-----')
data_check(dataFrame_eth)
print('-----XRP Data Check-----')
data_check(dataFrame_xrp)
```

```
-----ADA Data Check-----
If Dataframe has any null values :: False
Number of duplicate values in Dataframe :: 0
-----BNB Data Check-----
If Dataframe has any null values :: False
Number of duplicate values in Dataframe :: 0
-----BTC Data Check-----
If Dataframe has any null values :: False
Number of duplicate values in Dataframe :: 1
Number of duplicate values after removing in Dataframe :: 0
-----ETH Data Check-----
If Dataframe has any null values :: False
Number of duplicate values in Dataframe :: 0
-----XRP Data Check-----
If Dataframe has any null values :: False
Number of duplicate values in Dataframe :: 0
```

[10]:

```python
print("************* HEAD Data **************")
print("ADA Data :: {}".format(dataFrame_ada.head()))
print("------------------------------------------------------------------------------")
print("BNB Data :: {}".format(dataFrame_bnb.head()))
print("------------------------------------------------------------------------------")
print("BTC Data :: {}".format(dataFrame_btc.head()))
print("------------------------------------------------------------------------------")
print("ETH Data :: {}".format(dataFrame_eth.head()))
print("------------------------------------------------------------------------------")
print("XRP Data :: {}".format(dataFrame_xrp.head()))
```

```
************* HEAD Data **************
ADA Data ::                      date    symbol        open        high         low  \
1  2022-08-11 00:00:00  ADA/USDT  0.53740000  0.54470000  0.53700000
2  2022-08-10 23:00:00  ADA/USDT  0.53520000  0.53750000  0.53490000
3  2022-08-10 22:00:00  ADA/USDT  0.53610000  0.53730000  0.53430000
4  2022-08-10 21:00:00  ADA/USDT  0.53470000  0.53660000  0.53150000
5  2022-08-10 20:00:00  ADA/USDT  0.52920000  0.53530000  0.52920000

        close
1  0.54470000
2  0.53740000
3  0.53520000
4  0.53610000
5  0.53490000
------------------------------------------------------------------------------
BNB Data ::                      date    symbol          open          high
low  \
1  2022-08-11 00:00:00  BNB/USDT  328.80000000  330.00000000  328.50000000
2  2022-08-10 23:00:00  BNB/USDT  327.20000000  328.90000000  327.00000000
3  2022-08-10 22:00:00  BNB/USDT  329.30000000  329.70000000  326.80000000
4  2022-08-10 21:00:00  BNB/USDT  327.70000000  330.50000000  326.10000000
5  2022-08-10 20:00:00  BNB/USDT  327.00000000  328.10000000  326.00000000

          close
1  329.90000000
2  328.80000000
3  327.30000000
4  329.20000000
5  327.70000000
------------------------------------------------------------------------------
BTC Data ::                      date    symbol            open            high  \
1  2022-08-15 00:00:00  BTC/USDT  24305.25000000  24316.56000000
2  2022-08-14 23:00:00  BTC/USDT  24257.90000000  24353.10000000
3  2022-08-14 22:00:00  BTC/USDT  24344.88000000  24364.40000000
4  2022-08-14 21:00:00  BTC/USDT  24313.51000000  24435.00000000
5  2022-08-14 20:00:00  BTC/USDT  24284.07000000  24352.22000000
```

```
              low          close
1  24164.10000000  24261.60000000
2  24234.07000000  24305.24000000
3  24172.40000000  24258.68000000
4  24273.94000000  24343.67000000
5  24251.22000000  24312.41000000
------------------------------------------------------------------------------
ETH Data ::                   date    symbol          open          high
low  \
1  2022-08-11 00:00:00  ETH/USDT  1853.58000000  1865.00000000  1850.32000000
2  2022-08-10 23:00:00  ETH/USDT  1850.00000000  1855.47000000  1846.56000000
3  2022-08-10 22:00:00  ETH/USDT  1860.69000000  1869.27000000  1845.70000000
4  2022-08-10 21:00:00  ETH/USDT  1842.50000000  1885.00000000  1828.88000000
5  2022-08-10 20:00:00  ETH/USDT  1818.24000000  1844.20000000  1817.72000000

           close
1  1859.76000000
2  1853.57000000
3  1849.99000000
4  1860.68000000
5  1842.50000000
------------------------------------------------------------------------------
XRP Data ::                   date    symbol        open        high         low
\
1  2022-08-11 00:00:00  XRP/USDT  0.38140000  0.38600000  0.38120000
2  2022-08-10 23:00:00  XRP/USDT  0.38010000  0.38150000  0.37940000
3  2022-08-10 22:00:00  XRP/USDT  0.37970000  0.38130000  0.37920000
4  2022-08-10 21:00:00  XRP/USDT  0.37880000  0.38070000  0.37730000
5  2022-08-10 20:00:00  XRP/USDT  0.37570000  0.37910000  0.37570000

        close
1  0.38460000
2  0.38140000
3  0.38020000
4  0.37970000
5  0.37870000
```

[11]:
```python
print("************* INFO Data **************")
print("ADA Data ::")
dataFrame_ada.info()
print("-------------------------------------------------------------------------------")
print("BNB Data ::")
dataFrame_bnb.info()
print("-------------------------------------------------------------------------------")
print("BTC Data ::")
dataFrame_btc.info()
```

```python
print("------------------------------------------------------------------------------------")
print("ETH Data ::")
dataFrame_eth.info()
print("------------------------------------------------------------------------------------")
print("XRP Data ::")
dataFrame_xrp.info()
```

```
************* INFO Data **************
ADA Data ::
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17720 entries, 1 to 17720
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   date    17720 non-null  object
 1   symbol  17720 non-null  object
 2   open    17720 non-null  object
 3   high    17720 non-null  object
 4   low     17720 non-null  object
 5   close   17720 non-null  object
dtypes: object(6)
memory usage: 830.8+ KB
------------------------------------------------------------------------------
BNB Data ::
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17720 entries, 1 to 17720
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   date    17720 non-null  object
 1   symbol  17720 non-null  object
 2   open    17720 non-null  object
 3   high    17720 non-null  object
 4   low     17720 non-null  object
 5   close   17720 non-null  object
dtypes: object(6)
memory usage: 830.8+ KB
------------------------------------------------------------------------------
BTC Data ::
<class 'pandas.core.frame.DataFrame'>
Int64Index: 43752 entries, 1 to 43752
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   date    43752 non-null  object
 1   symbol  43752 non-null  object
 2   open    43752 non-null  object
 3   high    43752 non-null  object
```

```
 4   low     43752 non-null  object
 5   close   43752 non-null  object
dtypes: object(6)
memory usage: 2.3+ MB
-------------------------------------------------------------------------------
ETH Data ::
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 43656 entries, 1 to 43656
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   date    43656 non-null  object
 1   symbol  43656 non-null  object
 2   open    43656 non-null  object
 3   high    43656 non-null  object
 4   low     43656 non-null  object
 5   close   43656 non-null  object
dtypes: object(6)
memory usage: 2.0+ MB
-------------------------------------------------------------------------------
XRP Data ::
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17720 entries, 1 to 17720
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   date    17720 non-null  object
 1   symbol  17720 non-null  object
 2   open    17720 non-null  object
 3   high    17720 non-null  object
 4   low     17720 non-null  object
 5   close   17720 non-null  object
dtypes: object(6)
memory usage: 830.8+ KB
```

[12]:

```
print("************* Tail Data **************")
print("-------------------------------------------------------------------------------")
print("ADA Data :: {}".format(dataFrame_ada.tail()))

print("-------------------------------------------------------------------------------")
print("BNB Data :: {}".format(dataFrame_bnb.tail()))

print("-------------------------------------------------------------------------------")
print("BTC Data :: {}".format(dataFrame_btc.tail()))

print("-------------------------------------------------------------------------------")
print("ETH Data :: {}".format(dataFrame_eth.tail()))
```

```python
print("--------------------------------------------------------------------------------")
print("XRP Data :: {}".format(dataFrame_xrp.tail()))
```

```
************* Tail Data **************
-------------------------------------------------------------------------------
ADA Data ::                       date    symbol        open        high
low  \
17716  2020-08-02 03:00:00  ADA/USDT  0.14435000  0.14826000  0.14418000
17717  2020-08-02 02:00:00  ADA/USDT  0.14372000  0.14490000  0.14334000
17718  2020-08-02 01:00:00  ADA/USDT  0.14495000  0.14503000  0.14312000
17719  2020-08-02 00:00:00  ADA/USDT  0.14415000  0.14527000  0.14313000
17720  2020-08-01 23:00:00  ADA/USDT  0.14482000  0.14678000  0.14404000

            close
17716  0.14759000
17717  0.14432000
17718  0.14372000
17719  0.14495000
17720  0.14410000
-------------------------------------------------------------------------------
BNB Data ::                       date    symbol         open         high
low  \
17716  2020-08-02 03:00:00  BNB/USDT  21.84860000  22.18320000  21.68510000
17717  2020-08-02 02:00:00  BNB/USDT  21.95000000  21.98730000  21.70000000
17718  2020-08-02 01:00:00  BNB/USDT  21.56480000  22.00000000  21.54450000
17719  2020-08-02 00:00:00  BNB/USDT  21.62040000  21.72160000  21.43100000
17720  2020-08-01 23:00:00  BNB/USDT  21.59000000  21.64990000  21.40000000

             close
17716  22.09640000
17717  21.84930000
17718  21.94800000
17719  21.56040000
17720  21.61680000
-------------------------------------------------------------------------------
BTC Data ::                    date    symbol      open      high      low
close
43748  2017-08-17 08-AM  BTC/USDT   4349.99   4377.85  4333.32  4360.69
43749  2017-08-17 07-AM  BTC/USDT   4324.35   4349.99  4287.41  4349.99
43750  2017-08-17 06-AM  BTC/USDT   4315.32   4345.45  4309.37  4324.35
43751  2017-08-17 05-AM  BTC/USDT   4308.83   4328.69  4291.37  4315.32
43752  2017-08-17 04-AM  BTC/USDT  16199.91  16199.91  4261.32  4308.83
-------------------------------------------------------------------------------
ETH Data ::                    date    symbol    open    high    low   close
43652  2017-08-17 08-AM  ETH/USDT  307.96  309.97    307  308.62
43653  2017-08-17 07-AM  ETH/USDT  302.68  307.96  302.6  307.96
43654  2017-08-17 06-AM  ETH/USDT   303.1  304.44  301.9  302.68
```

```
43656  2017-08-17 04-AM  ETH/USDT  652.74  652.74    298  300.79
43655  2017-08-17 05-AM  ETH/USDT  301.61  303.28    300   303.1
-------------------------------------------------------------------------------
XRP Data ::                   date    symbol        open        high
low  \
17716  2020-08-02 03:00:00  XRP/USDT  0.31217000  0.32562000  0.30800000
17717  2020-08-02 02:00:00  XRP/USDT  0.29464000  0.31500000  0.29374000
17718  2020-08-02 01:00:00  XRP/USDT  0.29084000  0.29509000  0.29018000
17719  2020-08-02 00:00:00  XRP/USDT  0.29089000  0.29423000  0.28927000
17720  2020-08-01 23:00:00  XRP/USDT  0.29387000  0.29388000  0.28839000

            close
17716  0.32452000
17717  0.31217000
17718  0.29465000
17719  0.29085000
17720  0.29089000
```

9 NumPy - slicing

[13]:
```
#NumPy - slicing

def get_OHLC(df):
    df['date']= pd.to_datetime(df['date'], errors='coerce')
    df[['open','high','low','close']]=df[['open','high','low','close']].
 ↪astype(float)
    name =[x for x in globals() if globals()[x] is df][0]
    df['OHLC'+name[9:]] = (df[['open','high', 'low', 'close']]).mean(axis=1)
    df.drop(columns=['symbol', 'open', 'high', 'low', 'close'],␣
 ↪axis=1,inplace=True)
```

[14]:
```
get_OHLC(dataFrame_ada)
get_OHLC(dataFrame_bnb)
get_OHLC(dataFrame_btc)
get_OHLC(dataFrame_eth)
get_OHLC(dataFrame_xrp)
```

[15]:
```
print("************* HEAD Data **************")
print("ADA Data :: {}".format(dataFrame_ada.head()))
print("-------------------------------------------------------------------------------")
print("BNB Data :: {}".format(dataFrame_bnb.head()))
print("-------------------------------------------------------------------------------")
print("BTC Data :: {}".format(dataFrame_btc.head()))
print("-------------------------------------------------------------------------------")
print("ETH Data :: {}".format(dataFrame_eth.head()))
print("-------------------------------------------------------------------------------")
print("XRP Data :: {}".format(dataFrame_xrp.head()))
```

```
************* HEAD Data **************
ADA Data ::                  date  OHLC_ada
1 2022-08-11 00:00:00  0.540950
2 2022-08-10 23:00:00  0.536250
3 2022-08-10 22:00:00  0.535725
4 2022-08-10 21:00:00  0.534725
5 2022-08-10 20:00:00  0.532150
------------------------------------------------------------------------------
BNB Data ::                  date  OHLC_bnb
1 2022-08-11 00:00:00   329.300
2 2022-08-10 23:00:00   327.975
3 2022-08-10 22:00:00   328.275
4 2022-08-10 21:00:00   328.375
5 2022-08-10 20:00:00   327.200
------------------------------------------------------------------------------
BTC Data ::                  date    OHLC_btc
1 2022-08-15 00:00:00  24261.8775
2 2022-08-14 23:00:00  24287.5775
3 2022-08-14 22:00:00  24285.0900
4 2022-08-14 21:00:00  24341.5300
5 2022-08-14 20:00:00  24299.9800
------------------------------------------------------------------------------
ETH Data ::                  date   OHLC_eth
1 2022-08-11 00:00:00  1857.1650
2 2022-08-10 23:00:00  1851.4000
3 2022-08-10 22:00:00  1856.4125
4 2022-08-10 21:00:00  1854.2650
5 2022-08-10 20:00:00  1830.6650
------------------------------------------------------------------------------
XRP Data ::                  date  OHLC_xrp
1 2022-08-11 00:00:00  0.383300
2 2022-08-10 23:00:00  0.380600
3 2022-08-10 22:00:00  0.380100
4 2022-08-10 21:00:00  0.379125
5 2022-08-10 20:00:00  0.377300
```

[16]:

```python
print("************* INFO Data **************")
print("ADA Data ::")
dataFrame_ada.info()
print("------------------------------------------------------------------------------")
print("BNB Data ::")
dataFrame_bnb.info()
print("------------------------------------------------------------------------------")
print("BTC Data ::")
dataFrame_btc.info()
print("------------------------------------------------------------------------------")
print("ETH Data ::")
```

```python
dataFrame_eth.info()
print("--------------------------------------------------------------------------------")
print("XRP Data ::")
dataFrame_xrp.info()
```

```
************* INFO Data **************
ADA Data ::
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17720 entries, 1 to 17720
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   date      17720 non-null  datetime64[ns]
 1   OHLC_ada  17720 non-null  float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 277.0 KB
------------------------------------------------------------------------------
BNB Data ::
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17720 entries, 1 to 17720
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   date      17720 non-null  datetime64[ns]
 1   OHLC_bnb  17720 non-null  float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 277.0 KB
------------------------------------------------------------------------------
BTC Data ::
<class 'pandas.core.frame.DataFrame'>
Int64Index: 43752 entries, 1 to 43752
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   date      17816 non-null  datetime64[ns]
 1   OHLC_btc  43752 non-null  float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 1.0 MB
------------------------------------------------------------------------------
ETH Data ::
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 43656 entries, 1 to 43656
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   date      17720 non-null  datetime64[ns]
 1   OHLC_eth  43656 non-null  float64
dtypes: datetime64[ns](1), float64(1)
```

```
memory usage: 682.3 KB
--------------------------------------------------------------------------------
XRP Data ::
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17720 entries, 1 to 17720
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   date      17720 non-null  datetime64[ns]
 1   OHLC_xrp  17720 non-null  float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 277.0 KB
```

[17]:

```
dataFrame_btc = dataFrame_btc.dropna(how='any',axis=0)
dataFrame_eth = dataFrame_eth.dropna(how='any',axis=0)
```

[18]:

```
print("BTC Data ::")
dataFrame_btc.info()
print("--------------------------------------------------------------------------------")
print("ETH Data ::")
dataFrame_eth.info()
```

```
BTC Data ::
<class 'pandas.core.frame.DataFrame'>
Int64Index: 17816 entries, 1 to 17816
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   date      17816 non-null  datetime64[ns]
 1   OHLC_btc  17816 non-null  float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 417.6 KB
--------------------------------------------------------------------------------
ETH Data ::
<class 'pandas.core.frame.DataFrame'>
Int64Index: 17720 entries, 1 to 17720
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   date      17720 non-null  datetime64[ns]
 1   OHLC_eth  17720 non-null  float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 415.3 KB
```

10 Merge DataFrames

[19]:
```
#Merge DataFrames

dataFrames_combine = [dataFrame_ada, dataFrame_bnb, dataFrame_btc,␣
↪dataFrame_eth, dataFrame_xrp]
df = ft.reduce(lambda left, right: pd.merge(left, right, on='date'),␣
↪dataFrames_combine)
```

[20]:
```
df.head()
```

[20]:

	date	OHLC_ada	OHLC_bnb	OHLC_btc	OHLC_eth	OHLC_xrp
0	2022-08-11 00:00:00	0.540950	329.300	24002.8725	1857.1650	0.383300
1	2022-08-10 23:00:00	0.536250	327.975	23896.1400	1851.4000	0.380600
2	2022-08-10 22:00:00	0.535725	328.275	23907.6075	1856.4125	0.380100
3	2022-08-10 21:00:00	0.534725	328.375	23962.5025	1854.2650	0.379125
4	2022-08-10 20:00:00	0.532150	327.200	23773.6775	1830.6650	0.377300

[21]:
```
df.info()
```

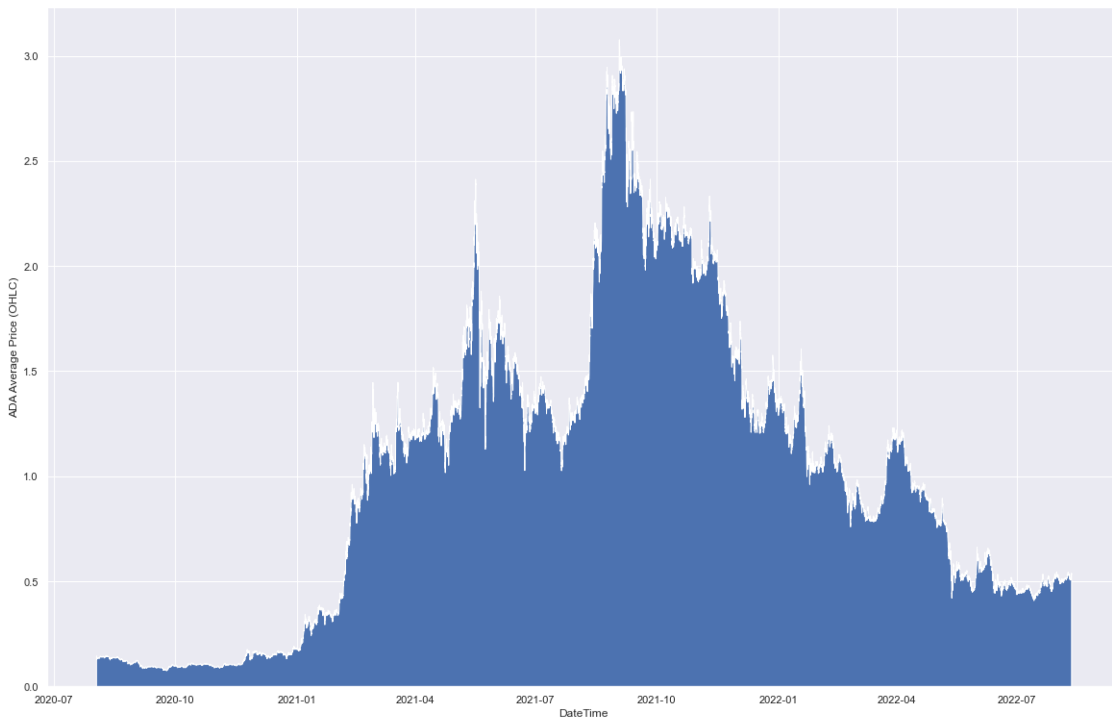
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 17720 entries, 0 to 17719
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   date      17720 non-null  datetime64[ns]
 1   OHLC_ada  17720 non-null  float64
 2   OHLC_bnb  17720 non-null  float64
 3   OHLC_btc  17720 non-null  float64
 4   OHLC_eth  17720 non-null  float64
 5   OHLC_xrp  17720 non-null  float64
dtypes: datetime64[ns](1), float64(5)
memory usage: 969.1 KB
```

[22]:
```
sns.set(rc={'figure.figsize':(20,13)})
```

11 Visualise using matplotlib

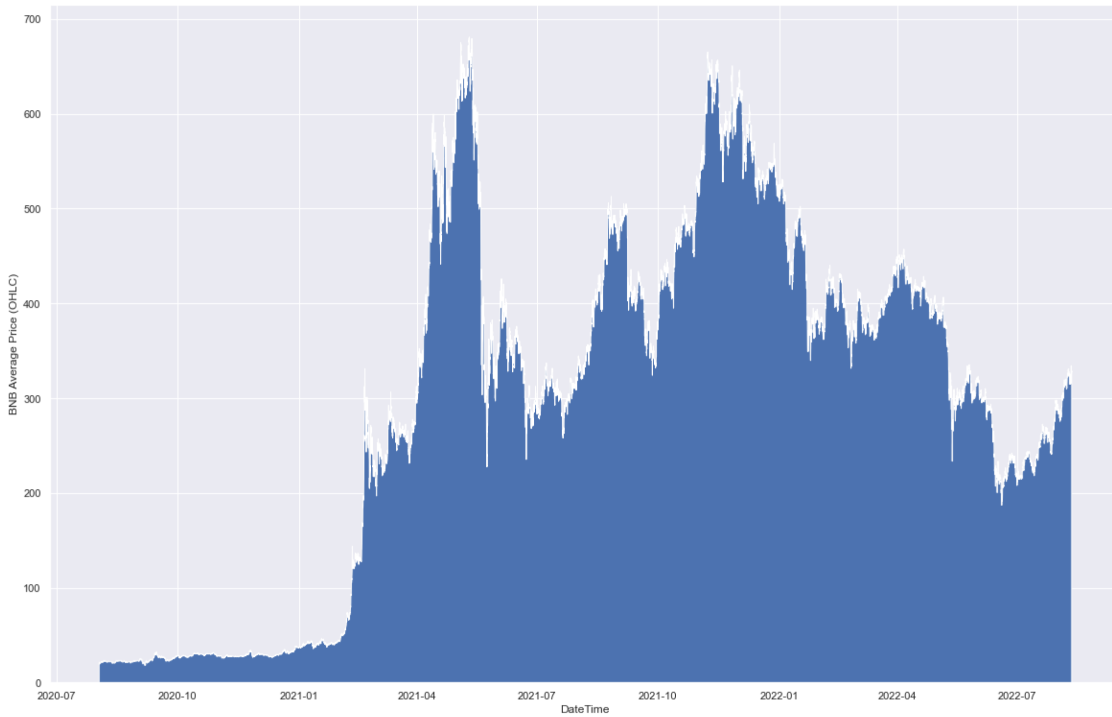
[23]:
```
#Visualise
print("************* ADA Chart **************")
import matplotlib.pyplot as plt
sns.set_theme()
plt.stackplot( dataFrame_ada['date'], dataFrame_ada['OHLC_ada'], labels=['ADA'])
plt.xlabel("DateTime")
plt.ylabel("ADA Average Price (OHLC)")
plt.show()
```

```
************* ADA Chart **************
```



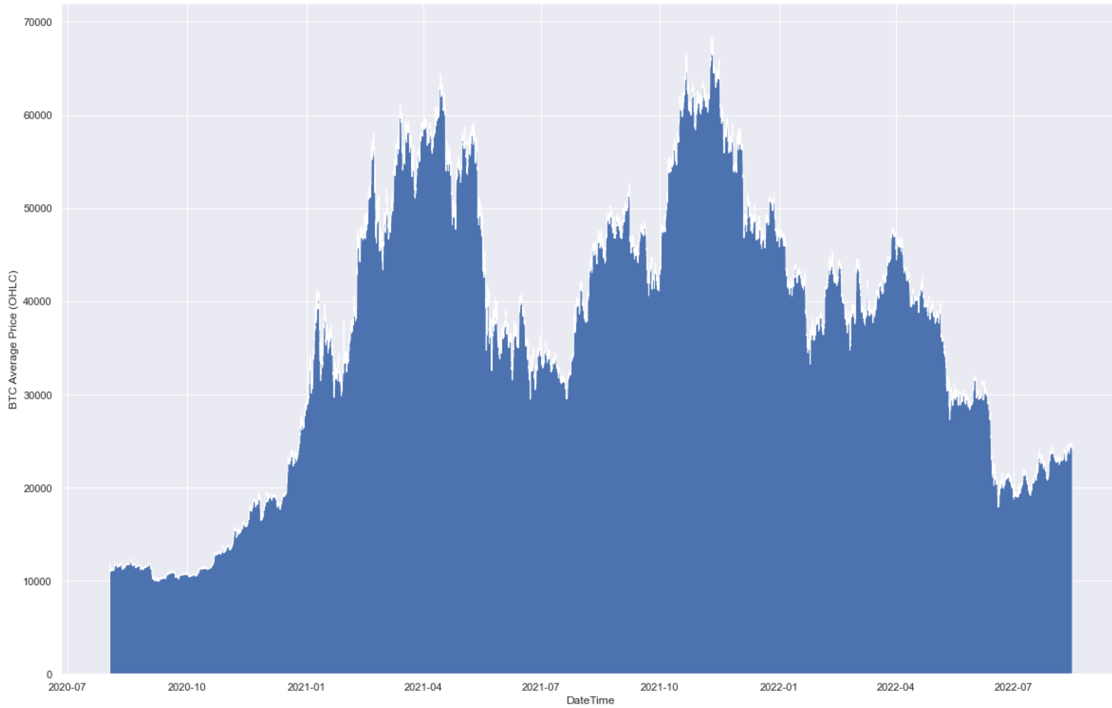
[24]:
```
print("************* BNB Chart **************")
plt.stackplot(dataFrame_bnb['date'], dataFrame_bnb['OHLC_bnb'], labels=['BNB'])
plt.xlabel("DateTime")
plt.ylabel("BNB Average Price (OHLC)")
plt.show()
```

```
************* BNB Chart **************
```



[25]:
```
print("************* BTC Chart **************")
plt.stackplot(dataFrame_btc['date'], dataFrame_btc['OHLC_btc'], labels=['BTC'])
plt.xlabel("DateTime")
plt.ylabel("BTC Average Price (OHLC)")
plt.show()
```

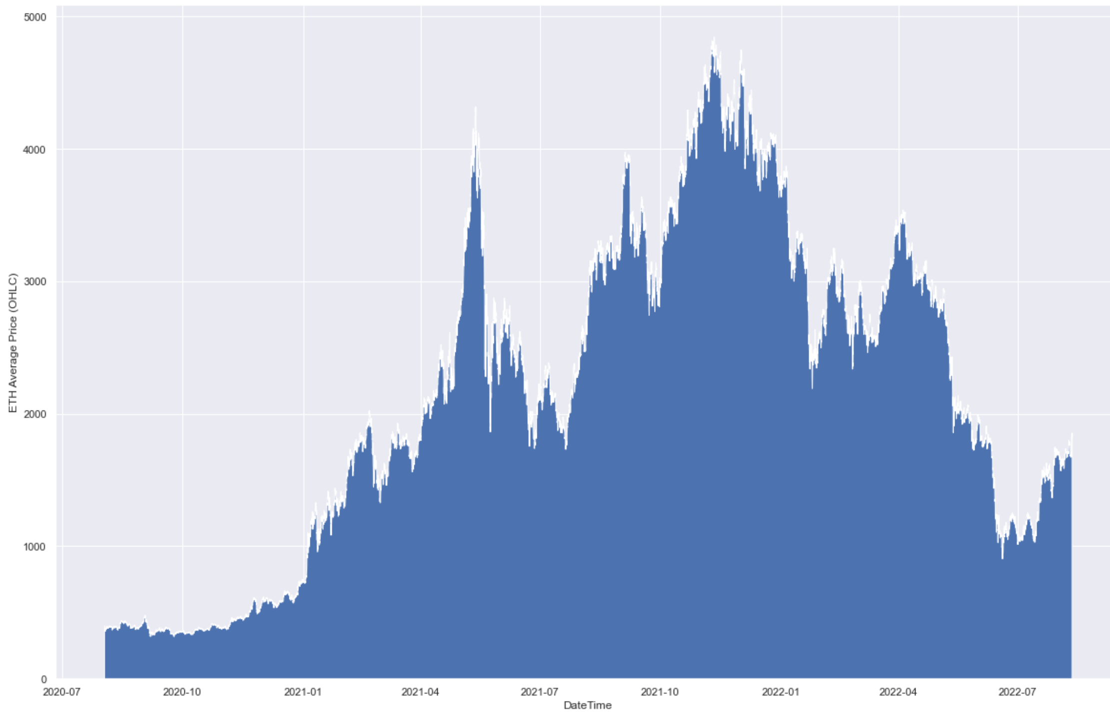
```
************* BTC Chart **************
```



[26]:

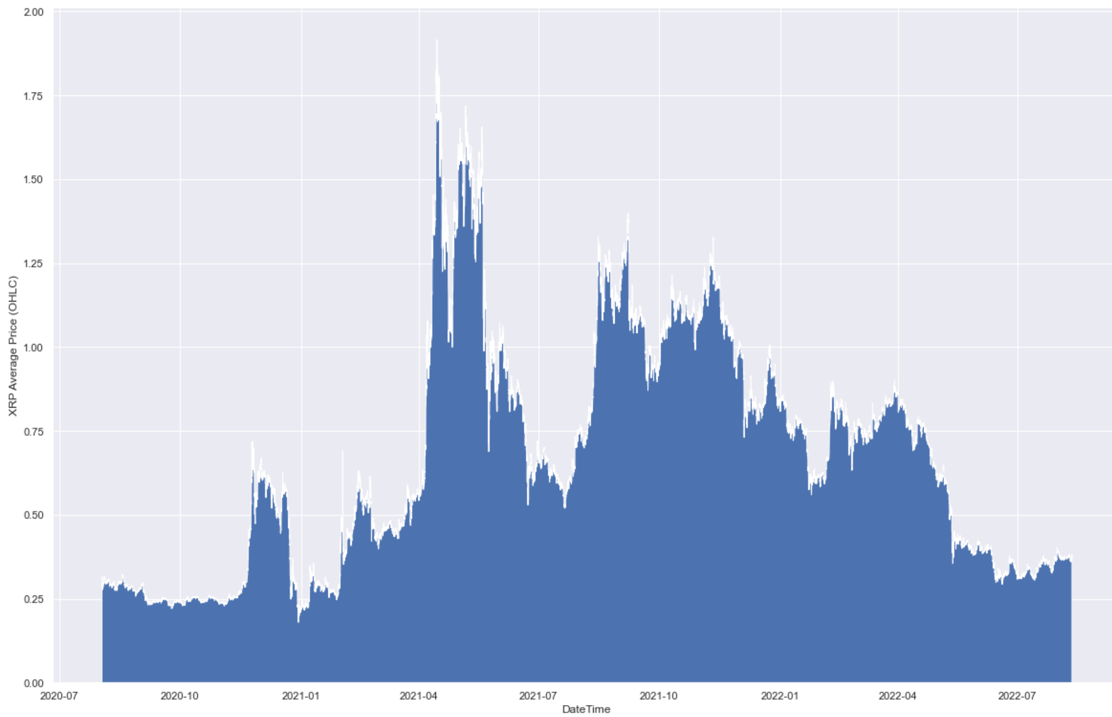
```
print("************* ETH Chart **************")
plt.stackplot(dataFrame_eth['date'], dataFrame_eth['OHLC_eth'], labels=['ETH'])
plt.xlabel("DateTime")
plt.ylabel("ETH Average Price (OHLC)")
plt.show()
```

```
************* ETH Chart **************
```



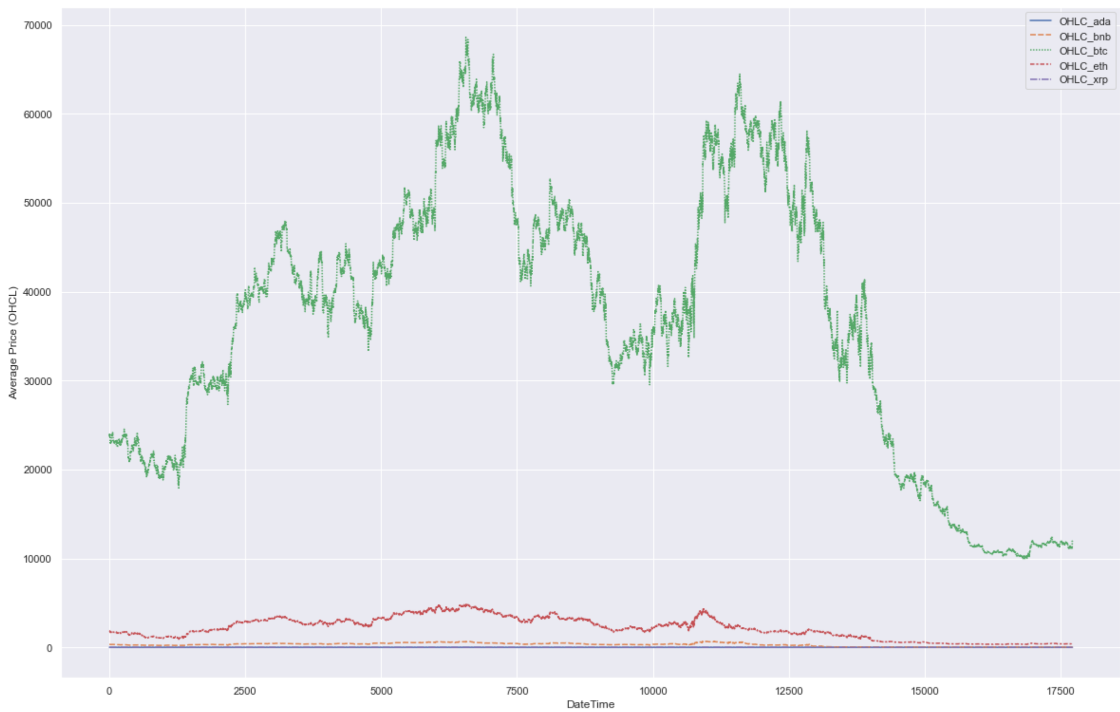
[27]:
```
print("************* XRP Chart **************")
plt.stackplot(dataFrame_xrp['date'], dataFrame_xrp['OHLC_xrp'], labels=['ETH'])
plt.xlabel("DateTime")
plt.ylabel("XRP Average Price (OHLC)")
plt.show()
```

```
************* XRP Chart **************
```




```
[28]: plt.xlabel("DateTime")
      plt.ylabel("Average Price (OHCL)")
      sns.lineplot(data = df)
```

```
[28]: <AxesSubplot:xlabel='DateTime', ylabel='Average Price (OHCL)'>
```



[29]:
```
sns.heatmap(df.corr(), cmap="YlGnBu", annot=True)
```

[29]: <AxesSubplot:>

12 Machine Learning - Supervised

[30]:
```
X = df[['OHLC_ada','OHLC_bnb','OHLC_btc','OHLC_eth']]
y = df['OHLC_xrp']

X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(X, y,␣
↪test_size = 0.3, random_state=42)

reg_all = LinearRegression()

reg_all.fit(X_train_reg, y_train_reg)
```

[30]:
```
LinearRegression()
```

[31]:
```
train_predict_reg = reg_all.predict(X_train_reg)
test_predict_reg = reg_all.predict(X_test_reg)
```

[32]:
```
import math
print("Regression Metrics - Multiple Linear Regression Model")
print("----------------------------------------------------------------------------------")
print("R^2 Train Data: {}".format(reg_all.score(X_train_reg, y_train_reg)))
```

```
print("Train data RMSE: ", math.
 ↪sqrt(mean_squared_error(y_train_reg,train_predict_reg)))
print("Train data MSE: ", mean_squared_error(y_train_reg,train_predict_reg))
print("Train data MAE: ", mean_absolute_error(y_train_reg,train_predict_reg))
print("-------------------------------------------------------------------------------")
print("R^2 Test Data: {}".format(reg_all.score(X_test_reg, y_test_reg)))
print("Test data RMSE: ", math.
 ↪sqrt(mean_squared_error(y_test_reg,test_predict_reg)))
print("Test data MSE: ", mean_squared_error(y_test_reg,test_predict_reg))
print("Test data MAE: ", mean_absolute_error(y_test_reg,test_predict_reg))
```

```
Regression Metrics - Multiple Linear Regression Model
--------------------------------------------------------------------------------
-----
R^2 Train Data: 0.8029019704136469
Train data RMSE:  0.15341378997393745
Train data MSE:  0.023535790954167392
Train data MAE:  0.10979972103477147
--------------------------------------------------------------------------------
-----
R^2 Test Data: 0.7936598650551228
Test data RMSE:  0.15920613703629086
Test data MSE:  0.025346594070018224
Test data MAE:  0.11357372366014024
```

[33]:
```
# Import the model we are using
from sklearn.ensemble import RandomForestRegressor
# Instantiate model with 1000 decision trees
rf = RandomForestRegressor(n_estimators = 1000, random_state = 42)
# Train the model on training data
rf.fit(X_train_reg, y_train_reg);

train_predict_rf=rf.predict(X_train_reg)
test_predict_rf=rf.predict(X_test_reg)

print("Regression Metrics - Random Forest Regression Model")
print("-------------------------------------------------------------------------------")
print("R^2 Train Data: {}".format(reg_all.score(X_train_reg, y_train_reg)))
print("Train data RMSE: ", math.
 ↪sqrt(mean_squared_error(y_train_reg,train_predict_rf)))
print("Train data MSE: ", mean_squared_error(y_train_reg,train_predict_rf))
print("Train data MAE: ", mean_absolute_error(y_train_reg,train_predict_rf))
print("-------------------------------------------------------------------------------")
print("R^2 Test Data: {}".format(reg_all.score(X_test_reg, y_test_reg)))
print("Test data RMSE: ", math.
 ↪sqrt(mean_squared_error(y_test_reg,test_predict_rf)))
print("Test data MSE: ", mean_squared_error(y_test_reg,test_predict_rf))
```

```
print("Test data MAE: ", mean_absolute_error(y_test_reg,test_predict_rf))
```

```
Regression Metrics - Random Forest Regression Model
--------------------------------------------------------------------------------
-----
R^2 Train Data: 0.8029019704136469
Train data RMSE:  0.007619603159219505
Train data MSE:  5.8058352303987855e-05
Train data MAE:  0.003114538660512747
--------------------------------------------------------------------------------
-----
R^2 Test Data: 0.7936598650551228
Test data RMSE:  0.019238822345849758
Test data MSE:  0.00037013228525516797
Test data MAE:  0.00833114461813389
```

13 Boosting - AdaBoost Regressor

[34]:
```
from sklearn.ensemble import AdaBoostRegressor

ab_regr = AdaBoostRegressor(random_state=43, n_estimators=100)
ab_regr.fit(X_train_reg, y_train_reg)
```

[34]:
```
AdaBoostRegressor(n_estimators=100, random_state=43)
```

[35]:
```
train_predict_abr=ab_regr.predict(X_train_reg)
test_predict_abr=ab_regr.predict(X_test_reg)

print("Train data prediction:", train_predict_abr.shape)
print("Test data prediction:", test_predict_abr.shape)
```

```
Train data prediction: (12404,)
Test data prediction: (5316,)
```

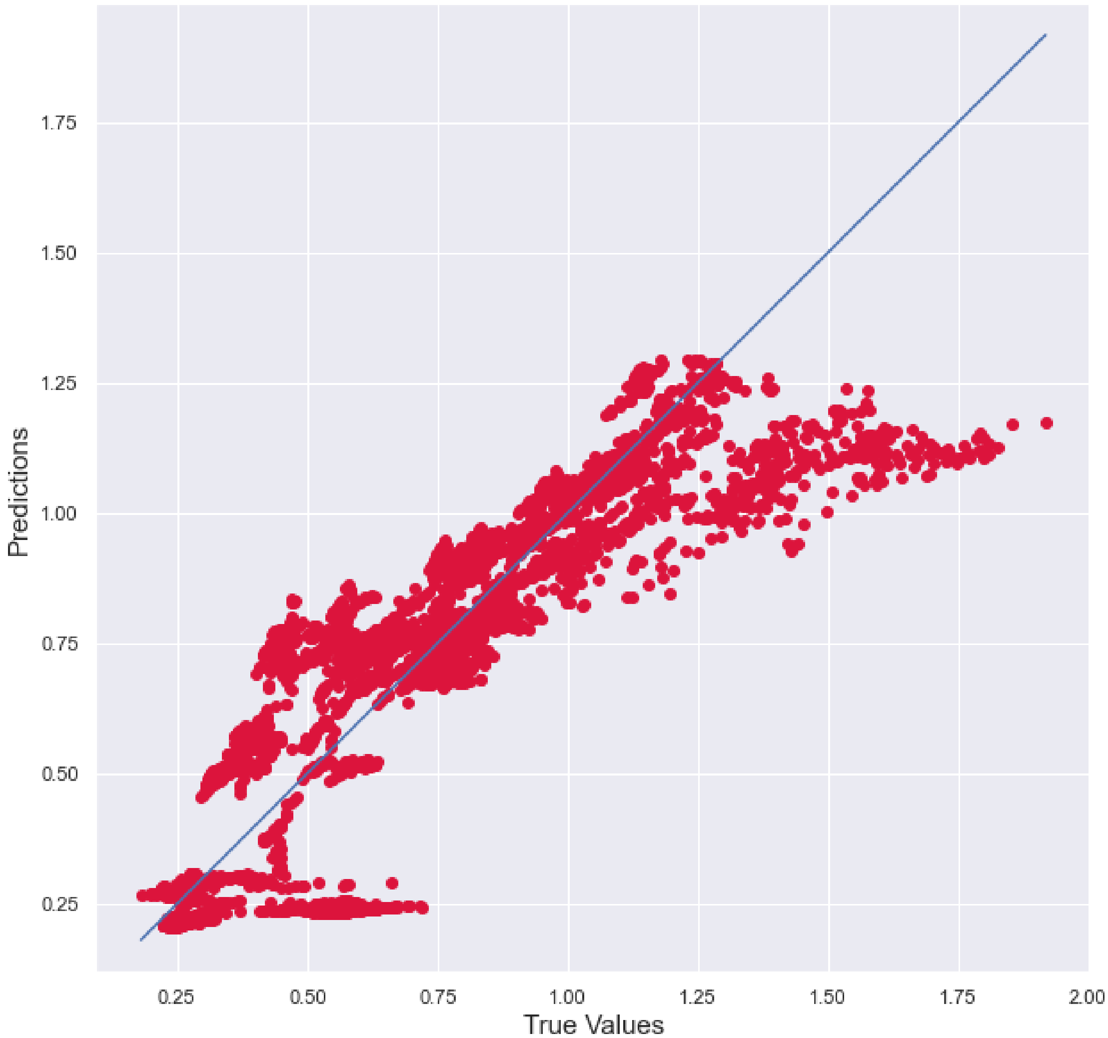
[36]:
```
print("Regression Metrics - AdaBoost Regressor")
print("--------------------------------------------------------------------------------")
print("Train data R2 score:", r2_score(y_train_reg,train_predict_abr))
print("Train data RMSE: ", math.
 ↪sqrt(mean_squared_error(y_train_reg,train_predict_abr)))
print("Train data MSE: ", mean_squared_error(y_train_reg,train_predict_abr))
print("Train data MAE: ", mean_absolute_error(y_train_reg,train_predict_abr))
print("--------------------------------------------------------------------------------")
print("Test data R2 score:", r2_score(y_test_reg,test_predict_abr))
print("Test data RMSE: ", math.
 ↪sqrt(mean_squared_error(y_test_reg,test_predict_abr)))
print("Test data MSE: ", mean_squared_error(y_test_reg,test_predict_abr))
print("Test data MAE: ", mean_absolute_error(y_test_reg,test_predict_abr))
```

```
print("------------------------------------------------------------------------------------")
```

```
Regression Metrics - AdaBoost Regressor
--------------------------------------------------------------------------------
-----
Train data R2 score: 0.8795675853595181
Train data RMSE:  0.11992092083309258
Train data MSE:  0.014381027253456857
Train data MAE:  0.1049030474587174
--------------------------------------------------------------------------------
-----
Test data R2 score: 0.8828198950263948
Test data RMSE:  0.11997614160206813
Test data MSE:  0.014394274553719501
Test data MAE:  0.10484497191218466
--------------------------------------------------------------------------------
-----
```

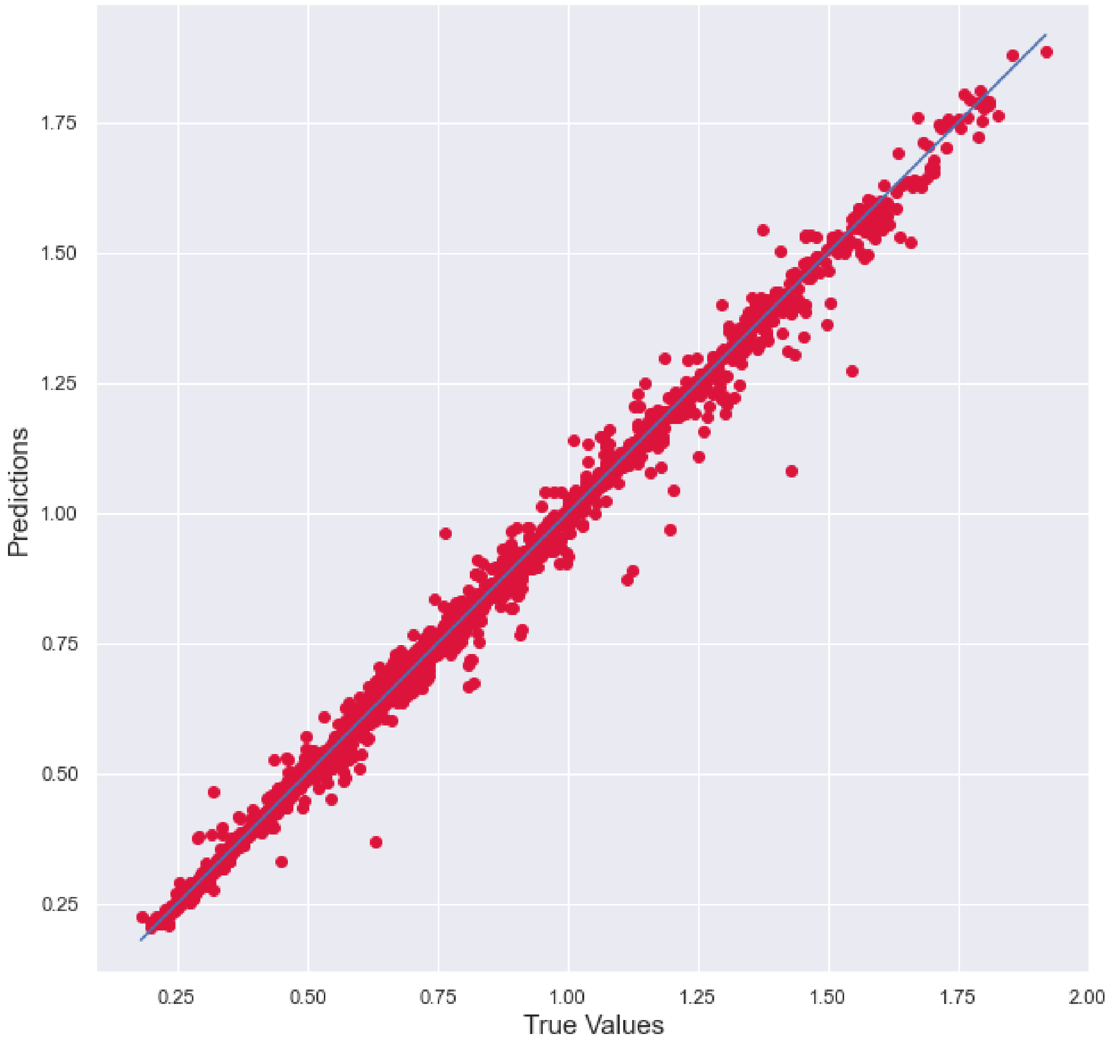
[37]:

```
# Linear Regression

plt.figure(figsize=(10,10))
plt.scatter(y_test_reg,test_predict_reg, c='crimson')
p1 = max(max(test_predict_reg), max(y_test_reg))
p2 = min(min(test_predict_reg), min(y_test_reg))
plt.plot([p1, p2], [p1, p2], 'b-',alpha=1)
plt.xlabel('True Values', fontsize=15)
plt.ylabel('Predictions', fontsize=15)
plt.axis('equal')
plt.show()
```

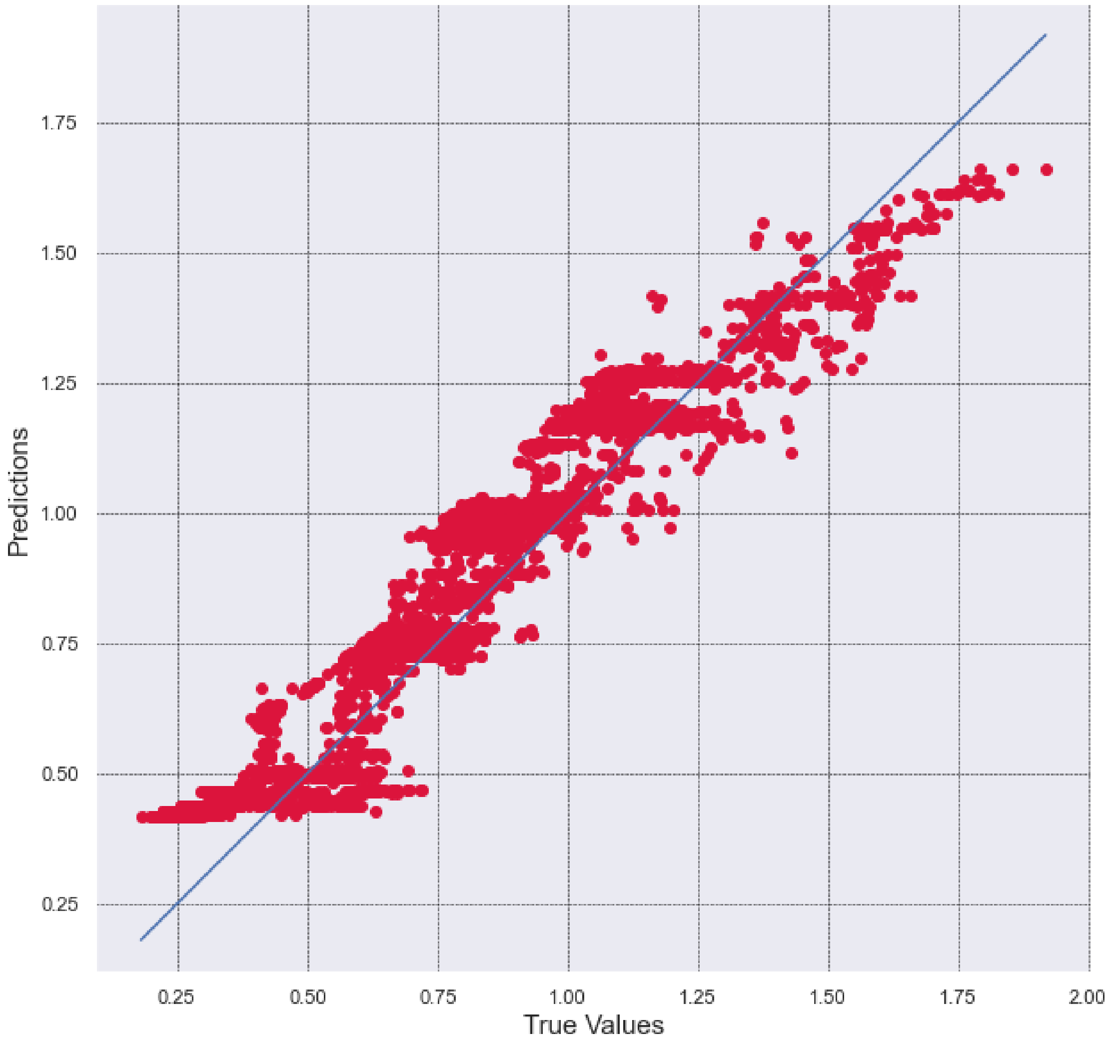


[38]:

```
# Random Forest Regression

plt.figure(figsize=(10,10))
plt.scatter(y_test_reg,test_predict_rf, c='crimson')
p1 = max(max(test_predict_rf), max(y_test_reg))
p2 = min(min(test_predict_rf), min(y_test_reg))
plt.plot([p1, p2], [p1, p2], 'b-',alpha=1)
plt.xlabel('True Values', fontsize=15)
plt.ylabel('Predictions', fontsize=15)
plt.axis('equal')
plt.show()
```



[39]:
```
# AdaBoost Regressor

plt.figure(figsize=(10,10))
plt.scatter(y_test_reg,test_predict_abr, c='crimson')

y_test_reg,test_predict_reg
p1 = max(max(test_predict_abr), max(y_test_reg))
p2 = min(min(test_predict_abr), min(y_test_reg))
plt.plot([p1, p2], [p1, p2], 'b-')
plt.grid(color = 'black', linestyle = '--', linewidth = 0.5)
plt.xlabel('True Values', fontsize=15)
plt.ylabel('Predictions', fontsize=15)
plt.axis('equal')
plt.show()
```



1.75
1.50
1.25
1.00
0.75
0.50
0.25
Predictions
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
True Values


```
[ ]:
```

Project Report

GitHub URL

https://github.com/shivankgarg/UCDPA_ShivankGarg

Abstract

This project has been done as a part of project submission for UCD Specialist Certificate in Data Analytics. The objective of this project is to predict price of cryptocurrency Ripple (XRP) based on the prices of 4 cryptocurrencies (Bitcoin (BTC), Ethereum (ETH), Binance Coin (BNB) and Cardano (ADA)). For the project, we use averaged price action or an average of OHLC (open, high, low, close) values) on an hourly interval and use machine learning models namely Multiple Linear Regression, Random Forest Regression and AdaBoost Regression to determine the price of XRP. The best performing model with a high R^2 value of 0.88 on the test data was the AdaBooster Regression model.

Introduction

Stocks and Cryptocurrencies always seems lucrative for me as they are good source of secondary income. However, given the variability in cryptocurrency prices within a day (they can move as much as 10% in a day) and Unlike trading stocks and commodities, the cryptocurrency market is open 24/7. I wanted to explore the idea of a data-driven trading strategy, where I would base the price of a cryptocurrency on the prices of 4 other (larger) cryptocurrencies.

I wanted to base the prediction model on the prices of other currencies and no other factors such as DateTime or trading volume as the price of one cryptocurrency strongly affects the price of the other.

Dataset

Source - https://www.cryptodatadownload.com/data/binance

The datasets for different cryptocurrencies were downloaded from the website. It contains historical data from different exchanges across the world and I choose the prices from the BINANCE exchange which has the largest daily trading volume in the world.
To have large sample space I had choose hourly data. I chose this source since the data was free and easy to access (no sign-up required) and reliable (as per reviews).

Dataset downloaded from website:

1. Binance_BTCUSDT_1h.csv
2. Binance_ADAUSDT_1h.csv
3. Binance_XRPUSDT_1h.csv
4. Binance_ETHUSDT_1h.csv
5. Binance_BNBUSDT_1h.csv

And each dataset contains the following columns:

- Unix Timestamp - This is the unix timestamp or also known as "Epoch Time". Use this to convert to your local timezone
- Date - This timestamp is in UTC datetime
- Symbol - The symbol for which the timeseries data refers
- Open - This is the opening price of the time period
- High - This is the highest price of the time period
- Low - This is the lowest price of the time period
- Close - This is the closing price of the time period
- Volume (Crypto) - This is the volume in the transacted Ccy. Ie. For BTC/USDT, this is in BTC amount
- Volume Base Ccy - This is the volume in the base/converted ccy. Ie. For BTC/USDT, this is in USDT amount
- Trade Count - This is the unique number of trades for the given time period

The aim of the cleaning process would be to get 2 columns each from the 5 datasets, the timestamp and an averaged value of the ‘Open’, ‘High’, ‘Low’ and ‘Close’ values for each cryptocurrency and then merge them along the timestamp.
We would be plotting open-high-low-close chart which can be used to illustrate movements in the price of different cryptocurrency over the time.

Implementation Process

Five Major Task were carried out in this project as below:

1. **Data Importing**
    - Fetching the Data from the API.
    - Importing the Data from CSV file into Dataframe.

2. **Data cleaning and merging**
    - Using .head() and .info() method, to know more about dataset imported.
    - Remove multi-index: Datasets came through as a multi-index and only one column where the column name was the data source (https://www.CryptoDataDownload.com).
    - Function remove_columns(): Custom function were created to reset the multi-index, rename columns using a dictionary.
    - Function data_check(): Check and remove the missing/duplicate values from dataset Using methods like isnull(), duplicated(), sum(), drop_duplicates().
    - Get average of open, high, low, close column: Combine open, high, low, close into one column OHLC that can be used for analysis using get_OHLC() function. All columns are objects, 'date' will be converted into datetime object while 'open', 'high', 'low', and 'close' columns will be converted into floats.
    - Dropping null values: During conversion, 25,936 'date' values did not convert to datetime object in dataFrame_btc (BTC) and dataframe_eth (ETH) datasets. These returned 'NaT' null value. The remaining values (17720) also happen to be the exact number of rows that are found in the dataFrame_bnb (BNB), dataFrame_ada (ADA) and dataFrame_xrp (XRP) datasets. This is likely due to the datasource changing the datatime format at the moment. Since all datasets would require equal values to merge and any NaT values would drop anyway, we drop these values using function dropna().
    - Merging datasets: Merge all 5 datasets into a single dataset namely df

3. **Exploratory Data Analysis**

- Plot datasets using matplotlib library: Plot the 5 datasets having datetime on the x-axis and average asset price (OHCL) on the y-axis.
- Comparative plot: Ploting all 5 datasets together with datetime on the x-axis and average asset price(OHCL) on the y-axis.
- Correlation matrix: Develop a correlation matrix for the 5 assets.

4. **Data preparation and model training**

- Creating feature and target variables: ‘df’ columns of 'OHLC_ada','OHLC_bnb','OHLC_btc','OHLC_eth' were added as feature variables ‘X’ and the target variable of ‘OHLC_xrp’ was assigned the target variable ‘y’.
- The feature and target variable ‘X’ and ‘y’ are split using a 70:30 ratio train-test split.
- Multiple Linear Regression: The multiple regression model is called and fit to the training data, before being used to predict the test ‘X’ dataset. Then, regression metrics are calculated to evaluate the model.
- Random Forest Regression: The Random Forest regression model is called and fit to the training data, before being used to predict the test ‘X’ dataset. Then, regression metrics are calculated to evaluate the model accuracy.
- AdaBoost Regression: The AdaBoost regression model is called and fit to the training data, before being used to predict the test ‘X’ dataset. Then, regression metrics are calculated to evaluate the model accuracy.

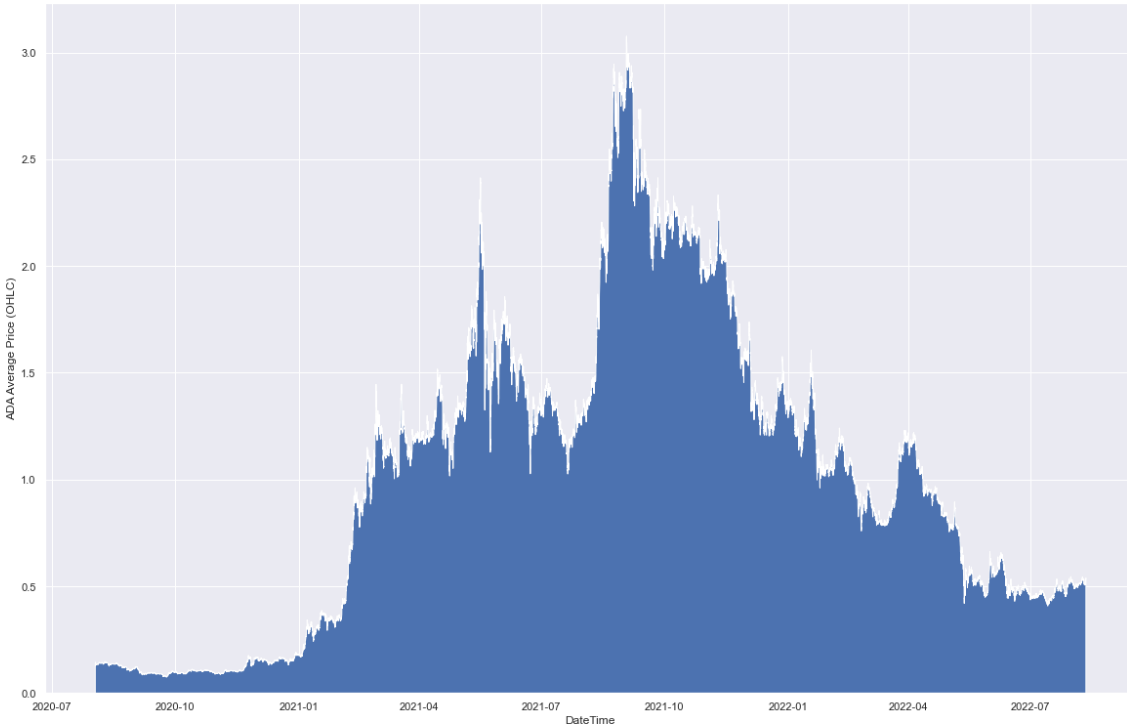
5. **Developing Insights**

- Correlation Matrix: Pearson correlation method was called on the dataframe to get an understanding of the correlation between the 5 cryptocurrencies and heatmap was plotted for the same.
- Multiple Regression Plot: The true values of the test data were plotted on the x-axis with the predicted values on the y-axis on a scatter plot to look at the differences between the two values. The regression metrics are added below the plot to have them available to develop insights over the model.
- Random Forest Regression Plot: The true and predicted values of the test data were charted on the scatter plot to look at the differences between the two values. Regression metrics were added as above.
- AdaBooster Regression Plot: The true and predicted values of the test data were charted to the scatter plot to look at the differences between the two values. Regression metrics were added as above.

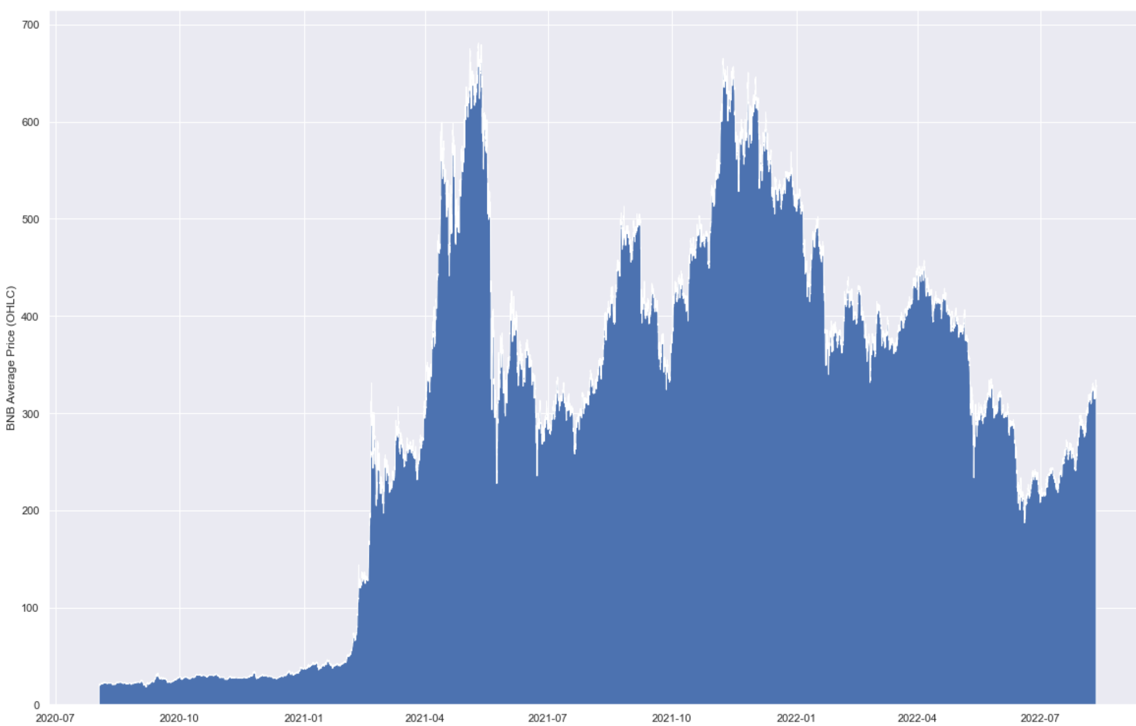
To conclude, data was loaded onto the Jupyter notebook, it was then cleaned to get one dataframe that contained the timestamp and the averaged OHLC values of the 5 assets before fitting 3 different machine learning models and then measuring them on various metrics. Finally, insights were developed on the correlation

Results

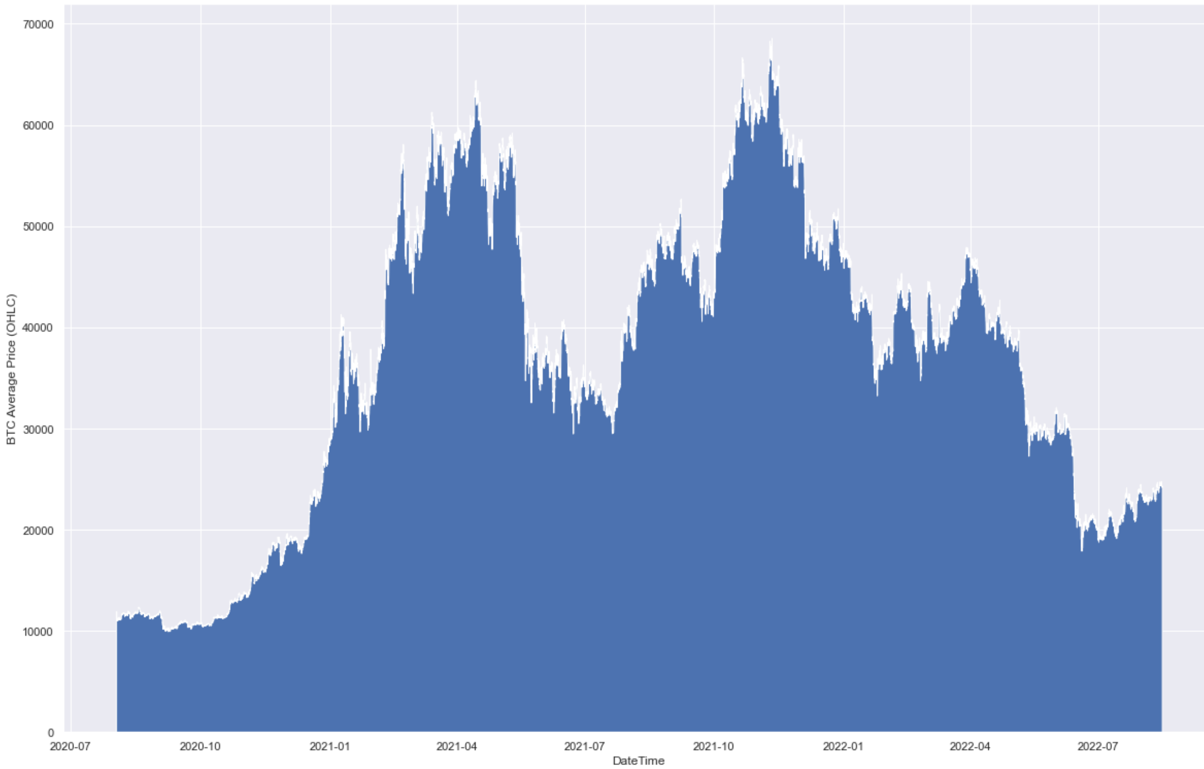
ADA Chart



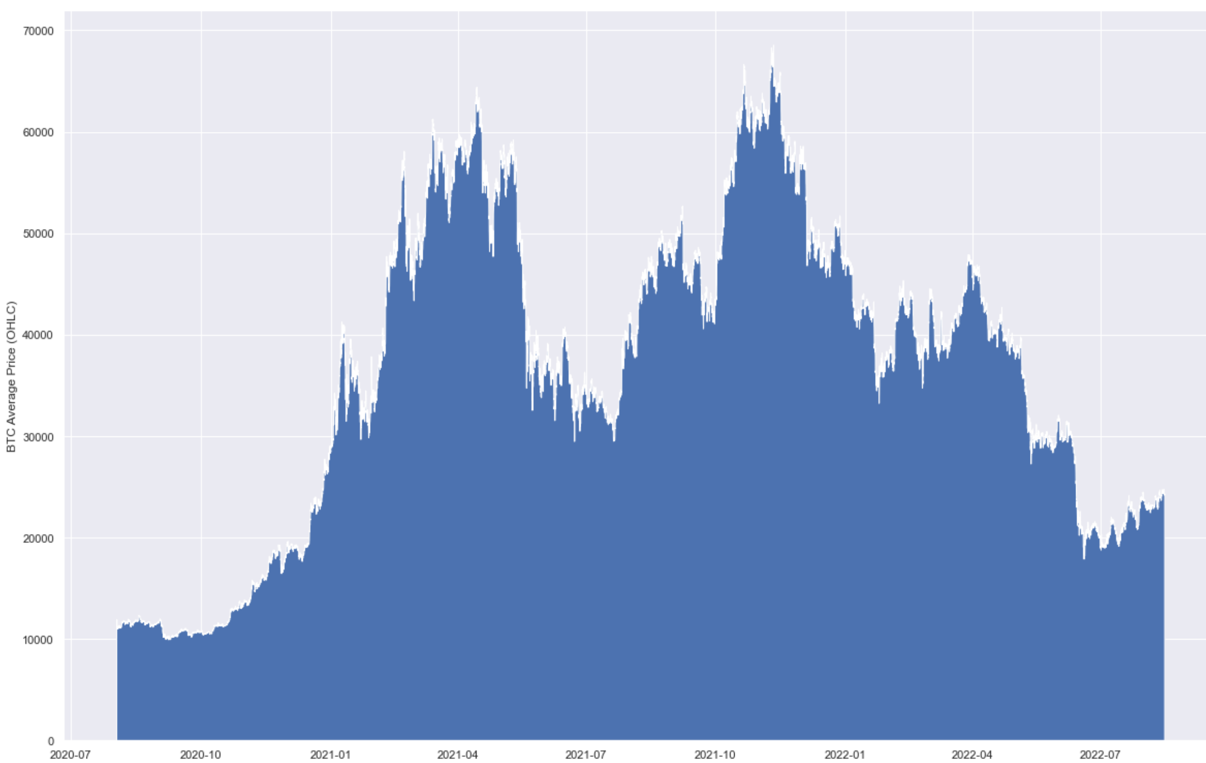
BNB Chart



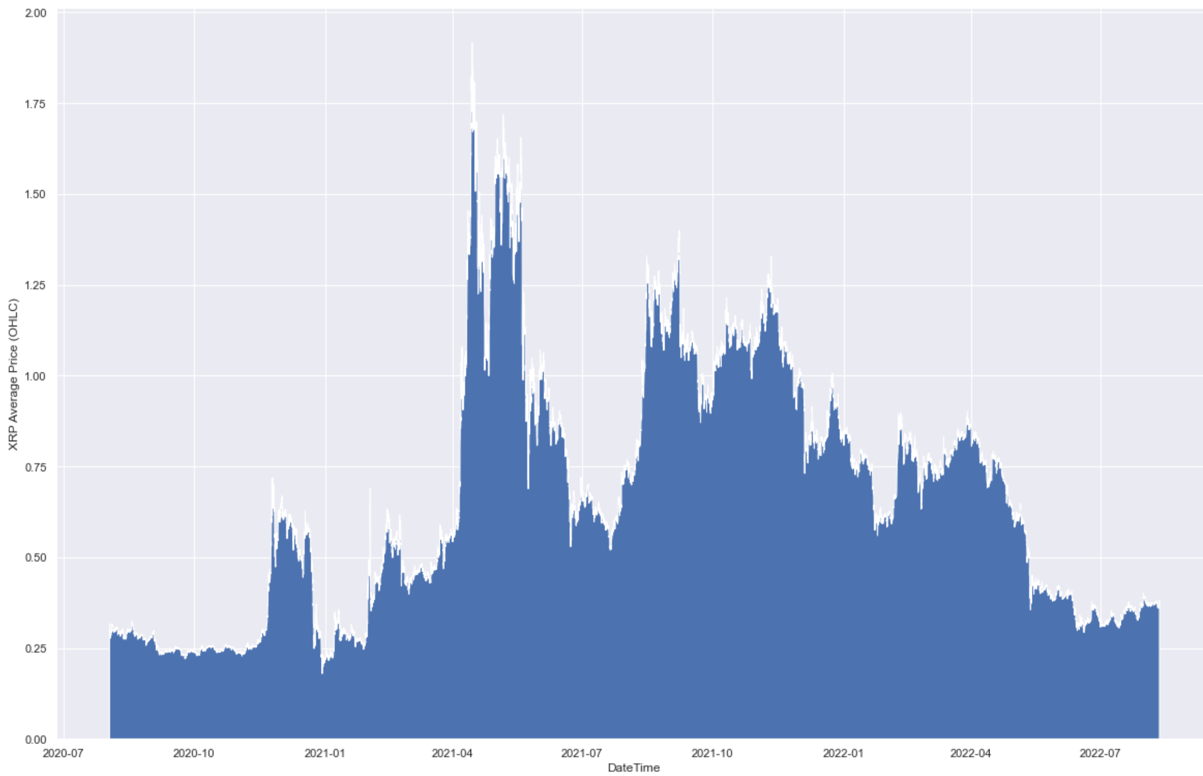
BTC Chart



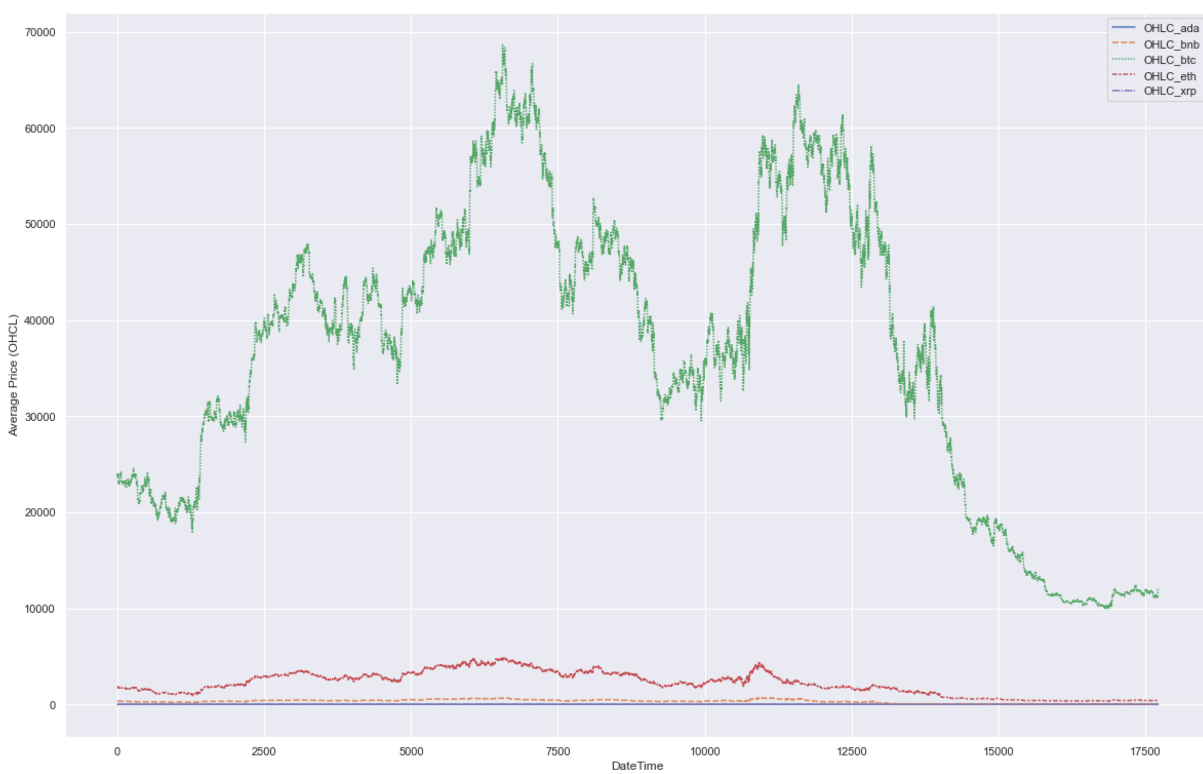
ETH Chart



XRP Chart

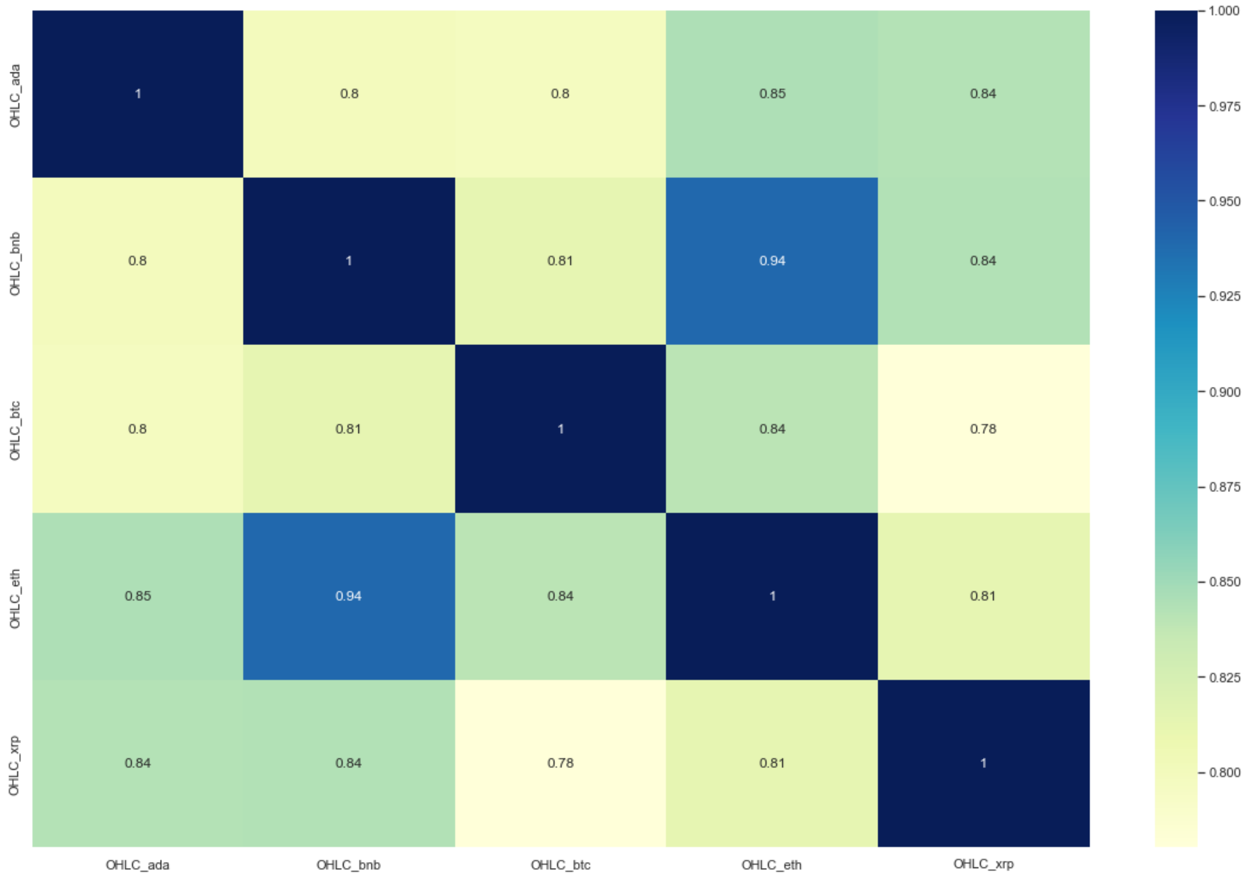


All Combined



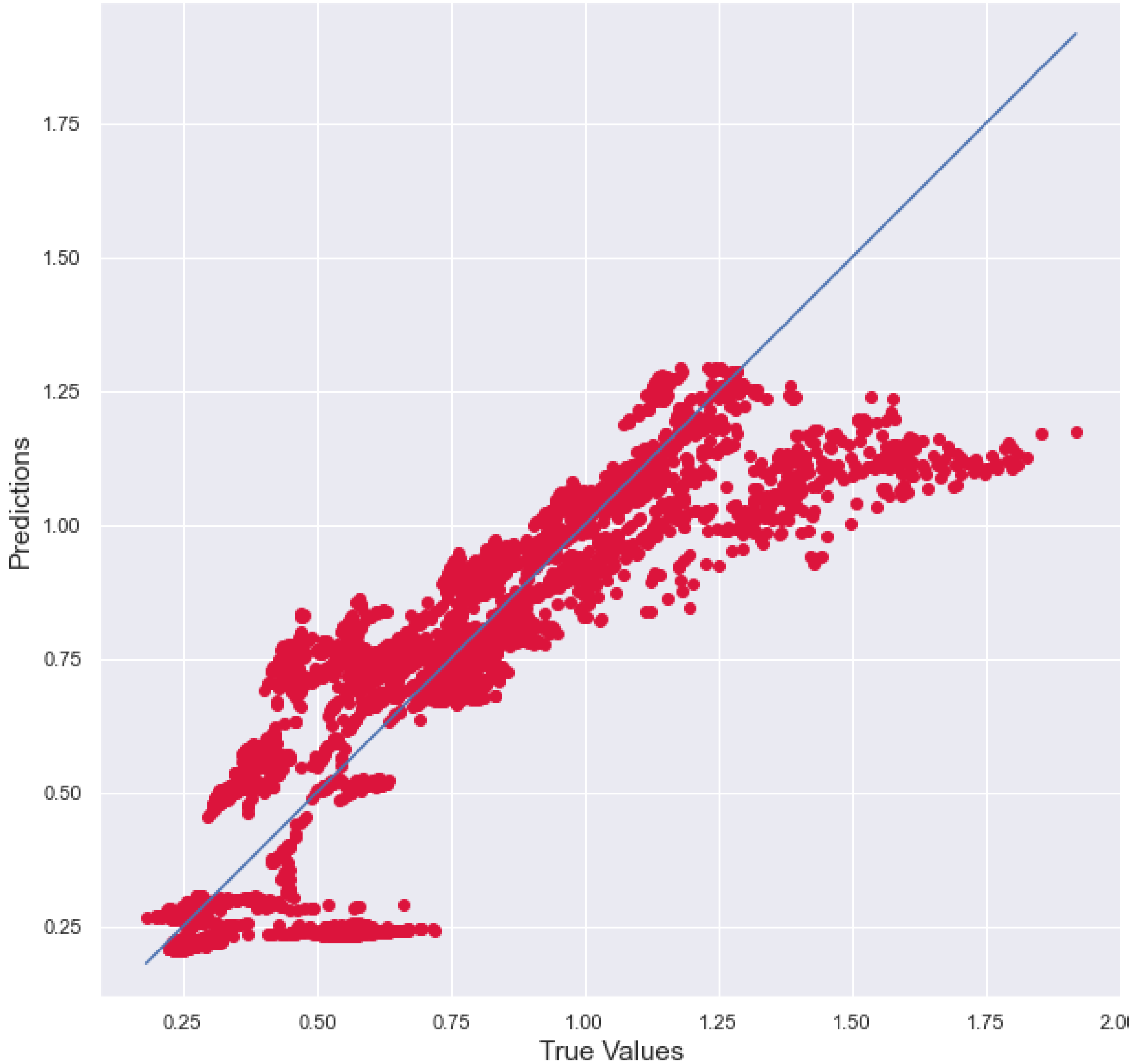
The results from the exploratory data analysis and prediction modelling are below:

1. Correlation Matrix



‘OHLC_xrp’ prices are positively correlated to all of the other cryptocurrencies, with it being most correlated to the ‘OHLC_ada’, ‘OHLC_bnb’ price (0.84) and least correlated to the ‘OHLC_btc’ price (0.78).

2. Multiple Linear Regression (MLR)
    a. Graph



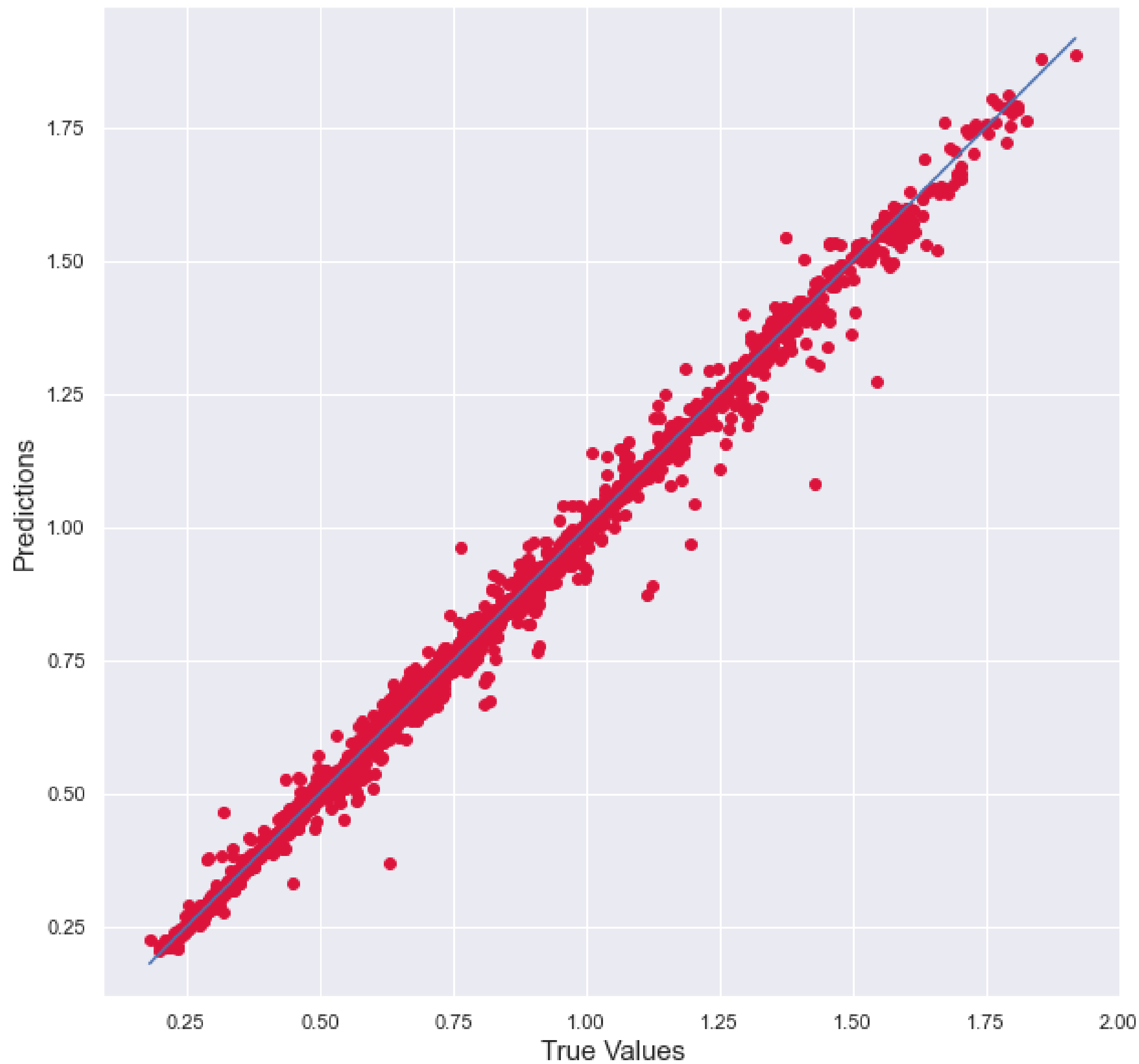
The multiple linear regression scatter plot shows the True Values of the test on the x-axis and the predicted values on the y-axis. We can see that the model is poor at higher true values of ‘OHLC_xrp’ but overall, the model undervalues the ‘OHLC_xrp’ price consistently.

b. Metrics

```
Regression Metrics - Multiple Linear Regression Model
-----------------------------------------------------------------------------------
R^2 Train Data: 0.8029019704136469
Train data RMSE:  0.15341378997393745
Train data MSE:  0.023535790954167392
Train data MAE:  0.10979972103477147
-----------------------------------------------------------------------------------
R^2 Test Data: 0.7936598650551228
Test data RMSE:  0.15920613703629086
Test data MSE:  0.025346594070018224
Test data MAE:  0.11357372366014024
```

The multiple linear regression model gives has an R^2 score of 0.79 on the test data, a Root Mean Squared Error of 0.15, Mean Squared Error of 0.025 and Mean Absolute Error of 0.11.
Scores on the Training Data were marginally better with slightly higher R^2 and slightly lower RMSE, MSE and MAE values.

3. Random Forest Regression
    a. Graph



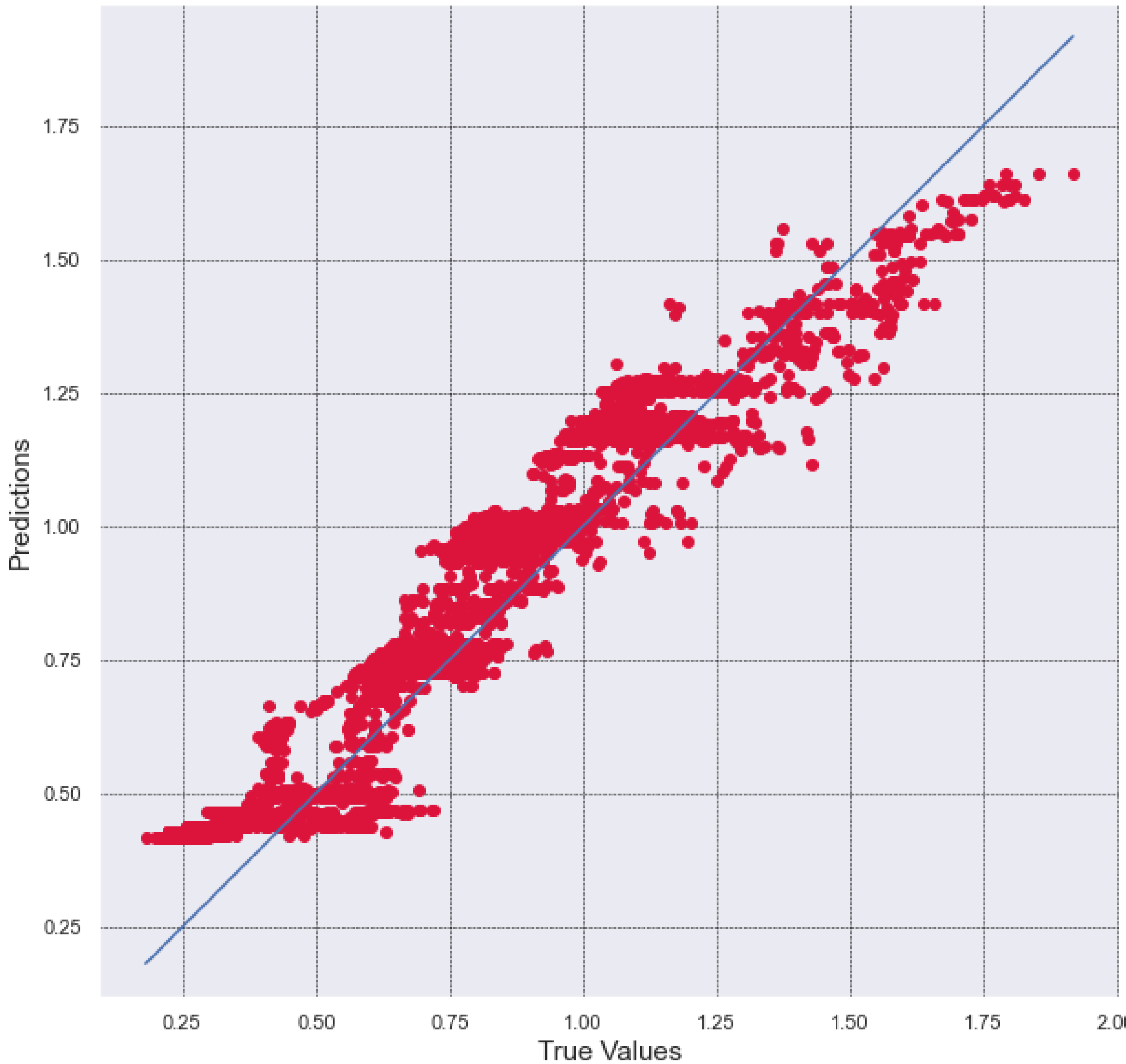
The Random Forest Regression scatterplot gives values that are more closely located around the line of fit, and the differences between the true values and the predicted values are less in comparison to the Multiple Linear Regression model.

b. Metrics

```
Regression Metrics - Random Forest Regression Model
-----------------------------------------------------------------------------------
R^2 Train Data: 0.8029019704136469
Train data RMSE:  0.007619603159219505
Train data MSE:  5.8058352303987855e-05
Train data MAE:  0.003114538660512747
-----------------------------------------------------------------------------------
R^2 Test Data: 0.7936598650551228
Test data RMSE:  0.019238822345849758
Test data MSE:  0.00037013228525516797
Test data MAE:  0.00833114461813389
```

The model has a test data R^2 score of 0.79, with an RMSE of 0.019, MSE of 0.0003 and MAE of 0.008. Scores on the random forest regression model is similar in comparison to the baseline model. The training data scores marginally better.

4. Adaboost Linear Regression (ABR)
    a. Graph



The Adaboost Regressor scatterplot gives values that are closely located around the line of fit.

b. Metrics

```
Regression Metrics - AdaBoost Regressor
-------------------------------------------------------------------------------------
Train data R2 score: 0.8795675853595181
Train data RMSE:  0.11992092083309258
Train data MSE:  0.014381027253456857
Train data MAE:  0.1049030474587174
-------------------------------------------------------------------------------------
Test data R2 score: 0.8828198950263948
Test data RMSE:  0.11997614160206813
Test data MSE:  0.014394274553719501
Test data MAE:  0.10484497191218466
-------------------------------------------------------------------------------------
```

The model has a test data R^2 score of 0.88, with an RMSE of 0.11, MSE of 0.014 and MAE of 0.10. This scores the Adaboost Regression model was best among all 3 models.

Insights

- Overall, the AdaBooster Regression model performs the best against the accuracy metrics in comparison to the other two models. It provides the most accurate predictions between the three models.

- Linear Regression and Random Forest Regression have the same R2 score for test data.

- OHLC_xrp prices are positively correlated to all of the other cryptocurrencies, with it being most correlated to the ‘OHLC_ada’, ‘OHLC_bnb’ price (0.84) and least correlated to the ‘OHLC_btc’ price (0.78). Other cryptocurrencies share a greater correlation, with the ‘OHLC_eth’ showing the most positive correlation using the Pearson method.

- All cryptocurrencies are very volatile in nature. As we can see in the graph in a single day there is a change of more than 20% in price of crypto.

- In early 2021, we saw cryptocurrency market saw boom and matured. The same can be seen in the OHLC graph for all 5 crypto as the price increased exponentially.

- As seen in the combined chart of all crypto, Bitcoin has the most volatility among all 5 crypto used for visualization.

- After so many up and down over a period of age in average price of XRP currency, price in 2020 and 2022 are almost same.