Bitcoin Price Prediction, Historical Analysis and Future Trends

June 23, 2024

1 Introduction

This project aims to provide insights into Bitcoin price dynamics and equip investors and traders with tools to make informed decisions in the cryptocurrency market.

Key Objectives:

- Explore the historical data of Bitcoin prices spanning several years.
- Visualize trends and patterns in Bitcoin price movements.
- Identify potential bullish trends and their underlying reasons.
- Forecast future Bitcoin prices using time-series forecasting methods.
- Develop a machine learning model to generate buy/sell signals based on historical price data.

1.0.1 Importing necessary libraries and uploading the data

[1]:
```
# Importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
# Loading the dataset
file_path = 'BTC-USD.csv'
df = pd.read_csv(file_path)
```

[3]:
```
df.head()
```

[3]:
```
         Date        Open        High         Low       Close   Adj Close  \
0  2014-09-17  465.864014  468.174011  452.421997  457.334015  457.334015
1  2014-09-18  456.859985  456.859985  413.104004  424.440002  424.440002
2  2014-09-19  424.102997  427.834991  384.532013  394.795990  394.795990
3  2014-09-20  394.673004  423.295990  389.882996  408.903992  408.903992
4  2014-09-21  408.084991  412.425995  393.181000  398.821014  398.821014

     Volume
0  21056800
1  34483200
2  37919700
3  36863600
```

```
4   26580100
```

[4]:
```
# Checking for missing values
print(df.isnull().sum())
```

```
Date          0
Open          0
High          0
Low           0
Close         0
Adj Close     0
Volume        0
dtype: int64
```

[5]:
```
df.describe()
```

[5]:
```
               Open          High           Low         Close     Adj Close  \
count   2788.000000   2788.000000   2788.000000   2788.000000   2788.000000
mean   12114.051628  12432.075536  11764.920824  12126.416572  12126.416572
std    16612.538889  17044.777808  16119.346993  16615.381435  16615.381435
min      176.897003    211.731003    171.509995    178.102997    178.102997
25%      612.573471    618.876495    609.665756    613.742477    613.742477
50%     6457.810059   6549.650147   6353.985107   6466.239990   6466.239990
75%    11024.040039  11388.611572  10722.320557  11056.325195  11056.325195
max    67549.734375  68789.625000  66382.062500  67566.828125  67566.828125

             Volume
count  2.788000e+03
mean   1.504640e+10
std    1.988339e+10
min    5.914570e+06
25%    8.317548e+07
50%    5.401853e+09
75%    2.558002e+10
max    3.509679e+11
```

1.1 Calculating Percent Change

In this notebook I calculate the percent change in Bitcoin's closing price and prepare the data by dropping rows with NaN values resulting from this calculation.

[6]:
```
df['Percent Change'] = df['Close'].pct_change() * 100

df.head()
```

[6]:
```
         Date        Open        High         Low       Close   Adj Close  \
0  2014-09-17  465.864014  468.174011  452.421997  457.334015  457.334015
```

```
1  2014-09-18  456.859985  456.859985  413.104004  424.440002  424.440002
2  2014-09-19  424.102997  427.834991  384.532013  394.795990  394.795990
3  2014-09-20  394.673004  423.295990  389.882996  408.903992  408.903992
4  2014-09-21  408.084991  412.425995  393.181000  398.821014  398.821014

     Volume  Percent Change
0  21056800             NaN
1  34483200       -7.192558
2  37919700       -6.984264
3  36863600        3.573492
4  26580100       -2.465855
```

[7]:
```python
# Droping the first row as it will have NaN value for percent change
df = df.dropna(subset=['Percent Change'])

df.head()
```

[7]:
```
         Date        Open        High         Low       Close   Adj Close  \
1  2014-09-18  456.859985  456.859985  413.104004  424.440002  424.440002
2  2014-09-19  424.102997  427.834991  384.532013  394.795990  394.795990
3  2014-09-20  394.673004  423.295990  389.882996  408.903992  408.903992
4  2014-09-21  408.084991  412.425995  393.181000  398.821014  398.821014
5  2014-09-22  399.100006  406.915985  397.130005  402.152008  402.152008

     Volume  Percent Change
1  34483200       -7.192558
2  37919700       -6.984264
3  36863600        3.573492
4  26580100       -2.465855
5  24127600        0.835210
```

[8]:
```python
# Ensuring the date column is in datetime format
df['Date'] = pd.to_datetime(df['Date'])
```

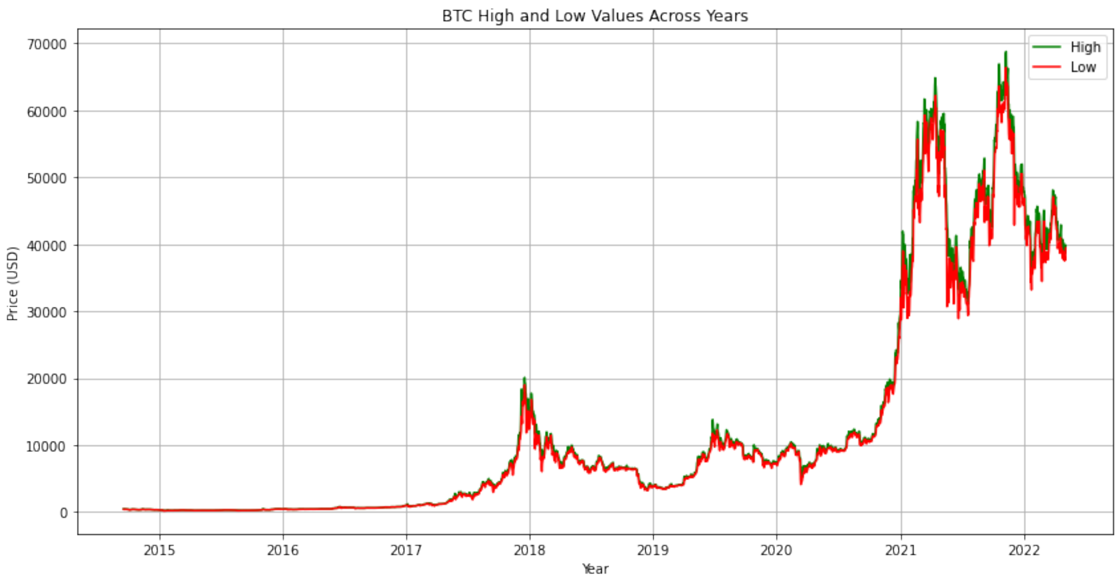
[9]:
```python
# Setting the date column as the index
df.set_index('Date', inplace=True)
```

1.1.1 Plotting High and Low Values Across Years

In this notebook I will visualize Bitcoin's high and low values across years to observe their trends.

[10]:
```python
plt.figure(figsize=(14, 7))
plt.plot(df['High'], label='High', color='g')
plt.plot(df['Low'], label='Low', color='r')
plt.title('BTC High and Low Values Across Years')
plt.xlabel('Year')
plt.ylabel('Price (USD)')
```

```
plt.legend()
plt.grid(True)
plt.show()
```



1.2 Analyzing Trends Based on Days of the Week

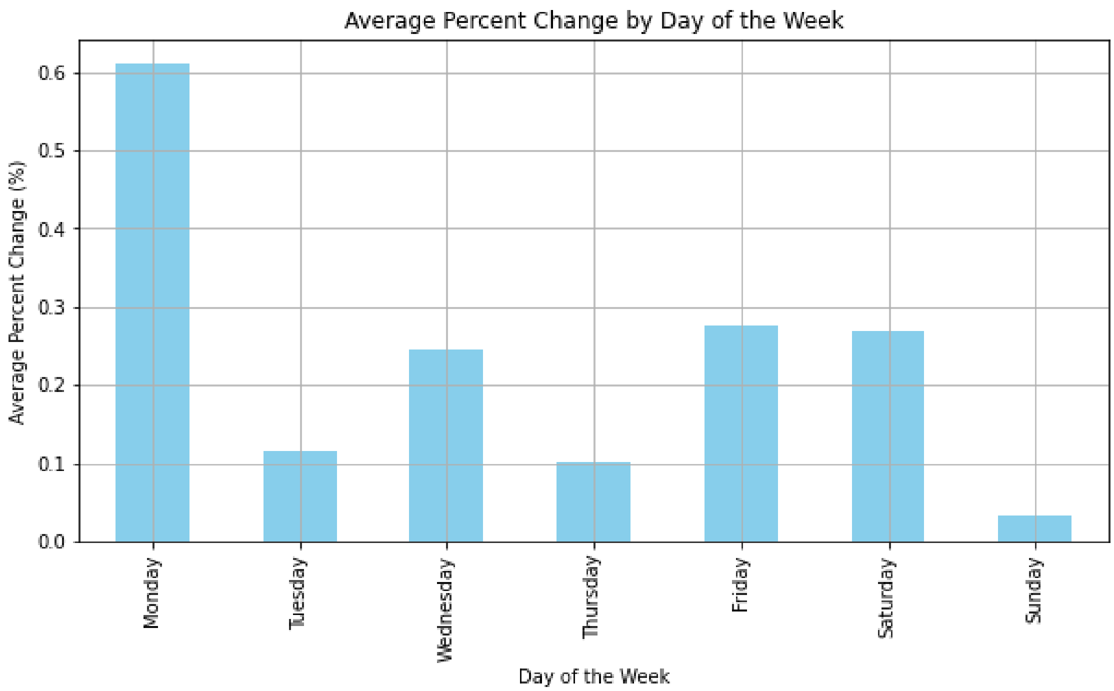
In this notebook I analyze Bitcoin trends based on the days of the week to identify patterns and average percent changes.

[11]:
```
# Extracting day of the week from the date
df['Day of Week'] = df.index.dayofweek

# Mapping day of the week numbers to actual names
day_of_week_mapping = {0: 'Monday', 1: 'Tuesday', 2: 'Wednesday', 3:␣
↪'Thursday', 4: 'Friday', 5: 'Saturday', 6: 'Sunday'}
df['Day of Week Name'] = df['Day of Week'].map(day_of_week_mapping)
```

[12]:
```
# Calculating average percent change for each day of the week
avg_percent_change_by_day = df.groupby('Day of Week Name')['Percent Change'].
↪mean()

# Sorting the days of the week for better visualization
avg_percent_change_by_day = avg_percent_change_by_day.loc[['Monday', 'Tuesday',␣
↪'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']]
```

1.3 Visualizing Average Percent Change by Day of the Week

[13]:
```
plt.figure(figsize=(10, 5))
avg_percent_change_by_day.plot(kind='bar', color='skyblue')
plt.title('Average Percent Change by Day of the Week')
plt.xlabel('Day of the Week')
plt.ylabel('Average Percent Change (%)')
plt.grid(True)
plt.show()
```



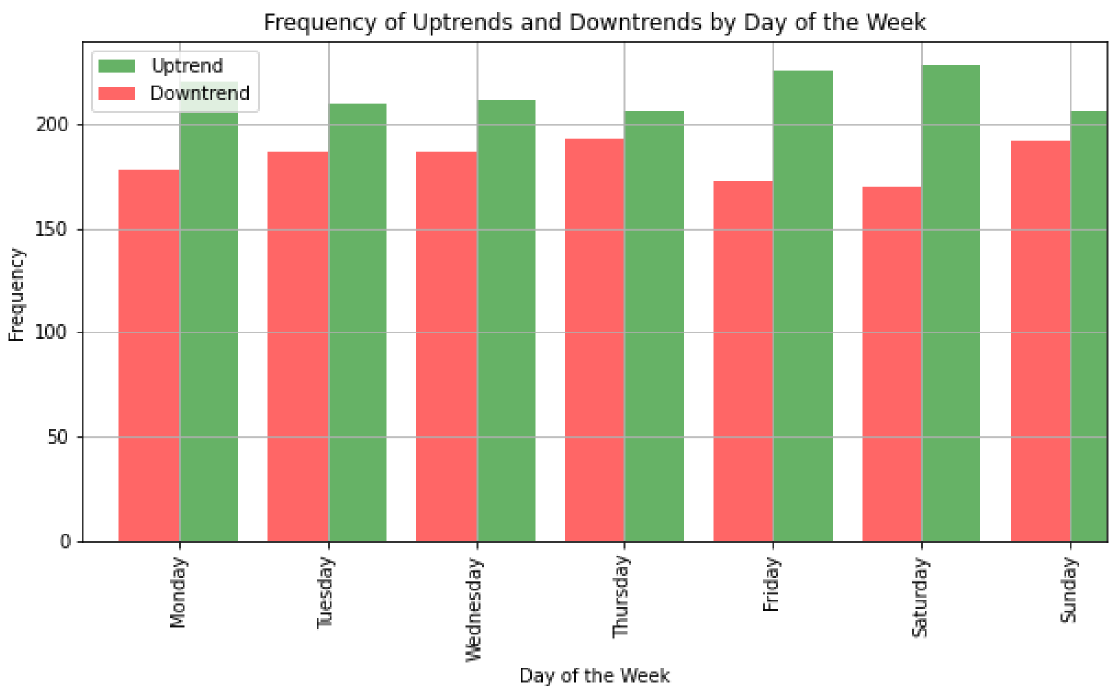
1.4 Analyzing Uptrends and Downtrends by Day of the Week

[14]:
```
uptrend_frequency = df[df['Percent Change'] > 0].groupby('Day of Week␣
↪Name')['Percent Change'].count()
downtrend_frequency = df[df['Percent Change'] < 0].groupby('Day of Week␣
↪Name')['Percent Change'].count()
```

[15]:
```
uptrend_frequency = uptrend_frequency.reindex(['Monday', 'Tuesday',␣
↪'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'])
downtrend_frequency = downtrend_frequency.reindex(['Monday', 'Tuesday',␣
↪'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'])
```

1.5 Visualizing Frequency of Uptrends and Downtrends by Day of the Week

[16]:
```
# Plotting the frequency of uptrends and downtrends
plt.figure(figsize=(10, 5))
uptrend_frequency.plot(kind='bar', color='green', alpha=0.6, position=0,␣
↪width=0.4, label='Uptrend')
downtrend_frequency.plot(kind='bar', color='red', alpha=0.6, position=1,␣
↪width=0.4, label='Downtrend')
plt.title('Frequency of Uptrends and Downtrends by Day of the Week')
plt.xlabel('Day of the Week')
plt.ylabel('Frequency')
plt.legend()
plt.grid(True)
plt.show()
```



1.6 Identifying Dramatic Bullish Trends

[17]:
```
threshold = 10
```

[18]:
```
bullish_days = df[df['Percent Change'] > threshold]

bullish_days
```

[18]:

Date	Open	High	Low	Close \
2014-11-12	367.984985	429.717987	367.984985	423.561005
2015-01-15	176.897003	229.067001	176.897003	209.843994
2015-11-02	325.941986	365.359985	323.209015	361.188995
2015-11-03	361.872986	417.899994	357.647003	403.416992
2016-01-20	379.739990	425.266998	376.598999	420.230011
2016-05-28	473.028992	533.473022	472.699005	530.039978
2016-06-12	609.684021	684.843994	607.039001	672.783997
2016-06-16	696.523010	773.721985	696.523010	766.307983
2017-01-04	1044.400024	1159.420044	1044.400024	1154.729980
2017-07-17	1932.619995	2230.489990	1932.619995	2228.409912
2017-07-20	2269.889893	2900.699951	2269.889893	2817.600098
2017-08-05	2897.629883	3290.010010	2874.830078	3252.909912
2017-09-15	3166.300049	3733.449951	2946.620117	3637.520020
2017-09-18	3591.090088	4079.229980	3591.090088	4065.199951
2017-10-12	4829.580078	5446.910156	4822.000000	5446.910156
2017-11-13	5938.250000	6811.189941	5844.290039	6559.490234
2017-11-15	6634.759766	7342.250000	6634.759766	7315.540039
2017-12-06	11923.400391	14369.099609	11923.400391	14291.500000
2017-12-07	14266.099609	17899.699219	14057.299805	17899.699219
2017-12-16	17760.300781	19716.699219	17515.300781	19497.400391
2017-12-26	14036.599609	16461.199219	14028.900391	16099.799805
2018-01-05	15477.200195	17705.199219	15202.799805	17429.500000
2018-01-20	11656.200195	13103.000000	11656.200195	12899.200195
2018-02-06	7051.750000	7850.700195	6048.259766	7754.000000
2018-02-14	8599.919922	9518.540039	8599.919922	9494.629883
2018-04-12	6955.379883	7899.229980	6806.509766	7889.250000
2018-11-28	3822.469971	4385.899902	3822.469971	4257.419922
2018-12-20	3742.195068	4191.228516	3728.974609	4134.441406
2019-04-02	4156.919434	4905.954590	4155.316895	4879.877930
2019-05-11	6379.666992	7333.002930	6375.698730	7204.771484
2019-05-13	6971.178223	8047.413086	6898.282227	7814.915039
2019-05-19	7267.962891	8261.941406	7267.962891	8197.689453
2019-06-26	11778.581055	13796.489258	11755.597656	13016.231445
2019-06-28	11162.167969	12445.174805	10914.495117	12407.332031
2019-07-03	10818.156250	11968.078125	10818.156250	11961.269531
2019-07-18	9698.502930	10736.842773	9376.798828	10666.482422
2019-10-25	7490.703125	8691.540039	7479.984375	8660.700195
2020-03-13	5017.831055	5838.114746	4106.980957	5563.707031
2020-03-19	5245.416504	6329.735840	5236.968750	6191.192871
2020-03-23	5831.374512	6443.934570	5785.004395	6416.314941
2020-04-29	7806.712402	8871.753906	7786.049316	8801.038086
2020-07-27	9905.217773	11298.221680	9903.969727	10990.873047
2020-11-05	14133.733398	15706.404297	14102.088867	15579.848633
2021-01-13	33915.121094	37599.960938	32584.667969	37316.359375
2021-02-08	38886.828125	46203.929688	38076.324219	46196.464844

2021-04-26	49077.792969	54288.003906	48852.796875	54021.753906
2021-05-20	36753.667969	42462.984375	35050.617188	40782.738281
2021-05-24	34700.363281	39835.140625	34551.082031	38705.980469
2021-06-09	33416.976563	37537.371094	32475.865234	37345.121094
2022-02-04	37149.265625	41527.785156	37093.628906	41500.875000
2022-02-28	37706.000000	43760.457031	37518.214844	43193.234375

	Adj Close	Volume	Percent Change	Day of Week \
Date				
2014-11-12	423.561005	45783200	15.193570	2
2015-01-15	209.843994	81773504	17.821709	3
2015-11-02	361.188995	101918000	10.987888	0
2015-11-03	403.416992	206162000	11.691385	1
2016-01-20	420.230011	121720000	10.543504	2
2016-05-28	530.039978	181199008	11.949375	5
2016-06-12	672.783997	277084992	10.887435	6
2016-06-16	766.307983	271633984	10.344449	3
2017-01-04	1154.729980	344945984	10.623277	2
2017-07-17	2228.409912	1201760000	15.472426	0
2017-07-20	2817.600098	2249260032	23.936087	3
2017-08-05	3252.909912	1945699968	12.328508	5
2017-09-15	3637.520020	4148069888	15.295649	4
2017-09-18	4065.199951	1943209984	13.461798	0
2017-10-12	5446.910156	2791610112	12.854714	3
2017-11-13	6559.490234	6263249920	10.242240	0
2017-11-15	7315.540039	4200880128	10.244359	2
2017-12-06	14291.500000	12656300032	19.928334	2
2017-12-07	17899.699219	17950699520	25.247169	3
2017-12-16	19497.400391	12740599808	10.111877	5
2017-12-26	16099.799805	13454300160	14.780490	1
2018-01-05	17429.500000	23840899072	11.733293	4
2018-01-20	12899.200195	11801700352	11.129105	5
2018-02-06	7754.000000	13999800320	11.483810	1
2018-02-14	9494.629883	7909819904	10.424378	2
2018-04-12	7889.250000	8906250240	13.215957	3
2018-11-28	4257.419922	7280280000	11.429782	2
2018-12-20	4134.441406	8927129279	10.370951	3
2019-04-02	4879.877930	21315047816	17.356014	1
2019-05-11	7204.771484	28867562329	12.947827	5
2019-05-13	7814.915039	28677672181	12.084030	0
2019-05-19	8197.689453	25902422040	12.741782	6
2019-06-26	13016.231445	45105733173	10.392020	2
2019-06-28	12407.332031	35087757766	10.950072	4
2019-07-03	11961.269531	30796494294	10.735293	2
2019-07-18	10666.482422	25187024648	10.034036	3
2019-10-25	8660.700195	28705065488	15.576342	4
2020-03-13	5563.707031	74156772075	11.928067	4

2020-03-19	6191.192871	51000731797	18.187756	3
2020-03-23	6416.314941	46491916000	10.052049	0
2020-04-29	8801.038086	60201052203	12.731805	2
2020-07-27	10990.873047	35359749590	10.961007	0
2020-11-05	15579.848633	40856321439	10.231863	3
2021-01-13	37316.359375	69364315979	10.003250	2
2021-02-08	46196.464844	101467222687	18.746474	0
2021-04-26	54021.753906	58284039825	10.238907	0
2021-05-20	40782.738281	88281943359	10.216344	3
2021-05-24	38705.980469	67359584098	11.318184	0
2021-06-09	37345.121094	53972919008	11.569118	2
2022-02-04	41500.875000	29412210792	11.697807	4
2022-02-28	43193.234375	35690014104	14.541184	0

Date	Day of Week Name
2014-11-12	Wednesday
2015-01-15	Thursday
2015-11-02	Monday
2015-11-03	Tuesday
2016-01-20	Wednesday
2016-05-28	Saturday
2016-06-12	Sunday
2016-06-16	Thursday
2017-01-04	Wednesday
2017-07-17	Monday
2017-07-20	Thursday
2017-08-05	Saturday
2017-09-15	Friday
2017-09-18	Monday
2017-10-12	Thursday
2017-11-13	Monday
2017-11-15	Wednesday
2017-12-06	Wednesday
2017-12-07	Thursday
2017-12-16	Saturday
2017-12-26	Tuesday
2018-01-05	Friday
2018-01-20	Saturday
2018-02-06	Tuesday
2018-02-14	Wednesday
2018-04-12	Thursday
2018-11-28	Wednesday
2018-12-20	Thursday
2019-04-02	Tuesday
2019-05-11	Saturday
2019-05-13	Monday

```
2019-05-19           Sunday
2019-06-26        Wednesday
2019-06-28           Friday
2019-07-03        Wednesday
2019-07-18         Thursday
2019-10-25           Friday
2020-03-13           Friday
2020-03-19         Thursday
2020-03-23           Monday
2020-04-29        Wednesday
2020-07-27           Monday
2020-11-05         Thursday
2021-01-13        Wednesday
2021-02-08           Monday
2021-04-26           Monday
2021-05-20         Thursday
2021-05-24           Monday
2021-06-09        Wednesday
2022-02-04           Friday
2022-02-28           Monday
```

1.7 Examining Context for Dramatic Bullish Days

[19]:
```
for index, row in bullish_days.head(5).iterrows():
    print(f"Date: {index}")
    print(f"Percent Change: {row['Percent Change']:.2f}%")
    print(f"Close Price: {row['Close']}")
    print(f"High: {row['High']}")
    print(f"Low: {row['Low']}")
    print()
```

```
Date: 2014-11-12 00:00:00
Percent Change: 15.19%
Close Price: 423.56100499999997
High: 429.717987
Low: 367.984985

Date: 2015-01-15 00:00:00
Percent Change: 17.82%
Close Price: 209.84399399999998
High: 229.06700099999998
Low: 176.897003

Date: 2015-11-02 00:00:00
Percent Change: 10.99%
Close Price: 361.18899500000003
High: 365.359985
```

```
Low: 323.20901499999997

Date: 2015-11-03 00:00:00
Percent Change: 11.69%
Close Price: 403.416992
High: 417.899994
Low: 357.647003

Date: 2016-01-20 00:00:00
Percent Change: 10.54%
Close Price: 420.23001100000005
High: 425.26699800000006
Low: 376.59899900000005
```

[20]:
```
# Printing out the dates for cross-referencing with historical news articles
print("Significant Bullish Days for Manual Research:")
for date in bullish_days.index[:10]:
    print(date)
```

```
Significant Bullish Days for Manual Research:
2014-11-12 00:00:00
2015-01-15 00:00:00
2015-11-02 00:00:00
2015-11-03 00:00:00
2016-01-20 00:00:00
2016-05-28 00:00:00
2016-06-12 00:00:00
2016-06-16 00:00:00
2017-01-04 00:00:00
2017-07-17 00:00:00
```

1.8 Time-Series Forecasting with ARIMA

[21]:
```
from statsmodels.tsa.arima_model import ARIMA
```

[22]:
```
df = df.sort_index()
```

[23]:
```
ts_data = df['Close']
```

1.9 Spliting the data into training and test sets (80-20 split)

[24]:
```
train_size = int(len(ts_data) * 0.8)
train, test = ts_data[:train_size], ts_data[train_size:]
```

[25]:
```
print(f"Training set length: {len(train)}")
print(f"Test set length: {len(test)}")
```

```
Training set length: 2229
Test set length: 558
```

[26]:
```
# Fiting the ARIMA model
model = ARIMA(train, order=(5, 1, 0))
model_fit = model.fit(disp=0)
```

```
/opt/conda/lib/python3.8/site-packages/statsmodels/tsa/base/tsa_model.py:524:
ValueWarning: No frequency information was provided, so inferred frequency D
will be used.
  warnings.warn('No frequency information was'
/opt/conda/lib/python3.8/site-packages/statsmodels/tsa/base/tsa_model.py:524:
ValueWarning: No frequency information was provided, so inferred frequency D
will be used.
  warnings.warn('No frequency information was'
```

[27]:
```
print(model_fit.summary())
```

```
                              ARIMA Model Results
==============================================================================
Dep. Variable:                D.Close   No. Observations:                 2228
Model:                 ARIMA(5, 1, 0)   Log Likelihood              -15742.403
Method:                       css-mle   S.D. of innovations            283.373
Date:                Mon, 27 May 2024   AIC                          31498.807
Time:                        22:02:04   BIC                          31538.769
Sample:                    09-19-2014   HQIC                         31513.400
                         - 10-24-2020
================================================================================
=
                    coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
-
const             5.7316      6.381      0.898      0.369      -6.776
18.239
ar.L1.D.Close     0.0177      0.021      0.837      0.402      -0.024
0.059
ar.L2.D.Close    -0.0045      0.021     -0.212      0.832      -0.046
0.037
ar.L3.D.Close    -0.0108      0.021     -0.513      0.608      -0.052
0.031
ar.L4.D.Close    -0.0276      0.021     -1.304      0.192      -0.069
0.014
ar.L5.D.Close     0.0845      0.021      3.999      0.000       0.043
0.126
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
```

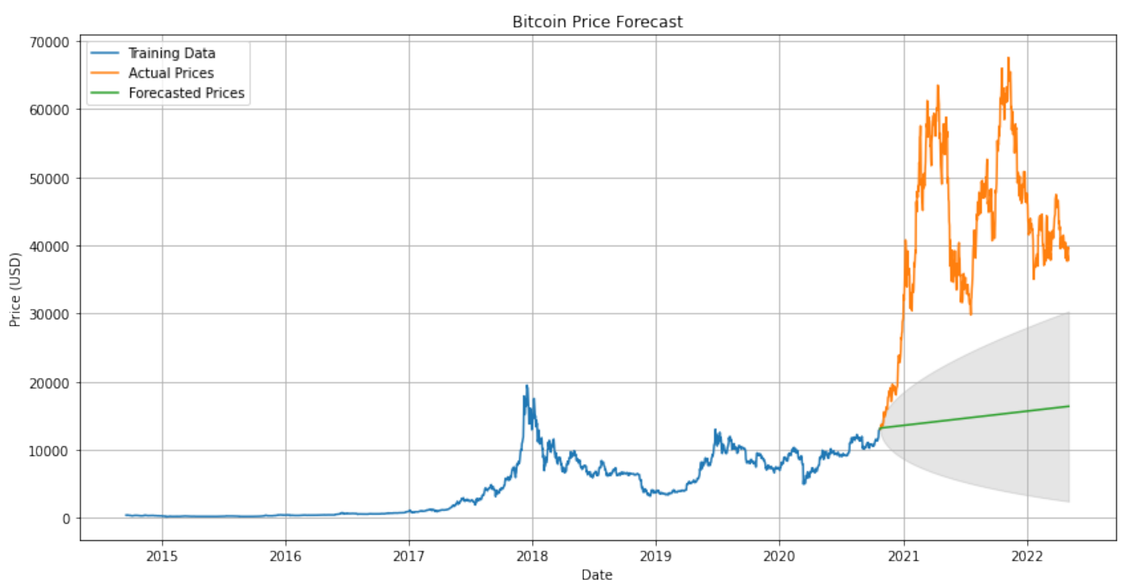
```
================================================================================
AR.1            -1.2783           -0.9653j            1.6018           -0.3971
AR.2            -1.2783           +0.9653j            1.6018            0.3971
AR.3             0.5802           -1.5299j            1.6363           -0.1923
AR.4             0.5802           +1.5299j            1.6363            0.1923
AR.5             1.7222           -0.0000j            1.7222           -0.0000
--------------------------------------------------------------------------------
```

[28]:
```python
# Forecasting the next steps in the test set
forecast, stderr, conf_int = model_fit.forecast(steps=len(test))
```

[29]:
```python
forecast_series = pd.Series(forecast, index=test.index)
```

1.10 Ploting the actual vs forecast values

[30]:
```python
plt.figure(figsize=(14, 7))
plt.plot(train, label='Training Data')
plt.plot(test, label='Actual Prices')
plt.plot(forecast_series, label='Forecasted Prices')
plt.fill_between(forecast_series.index, conf_int[:, 0], conf_int[:, 1],␣
 ↪color='k', alpha=0.1)
plt.title('Bitcoin Price Forecast')
plt.xlabel('Date')
plt.ylabel('Price (USD)')
plt.legend()
plt.grid(True)
plt.show()
```



[31]:
```
# Fitting the ARIMA model on the entire dataset to forecast future prices
model_full = ARIMA(ts_data, order=(5, 1, 0))
model_full_fit = model_full.fit(disp=0)
```

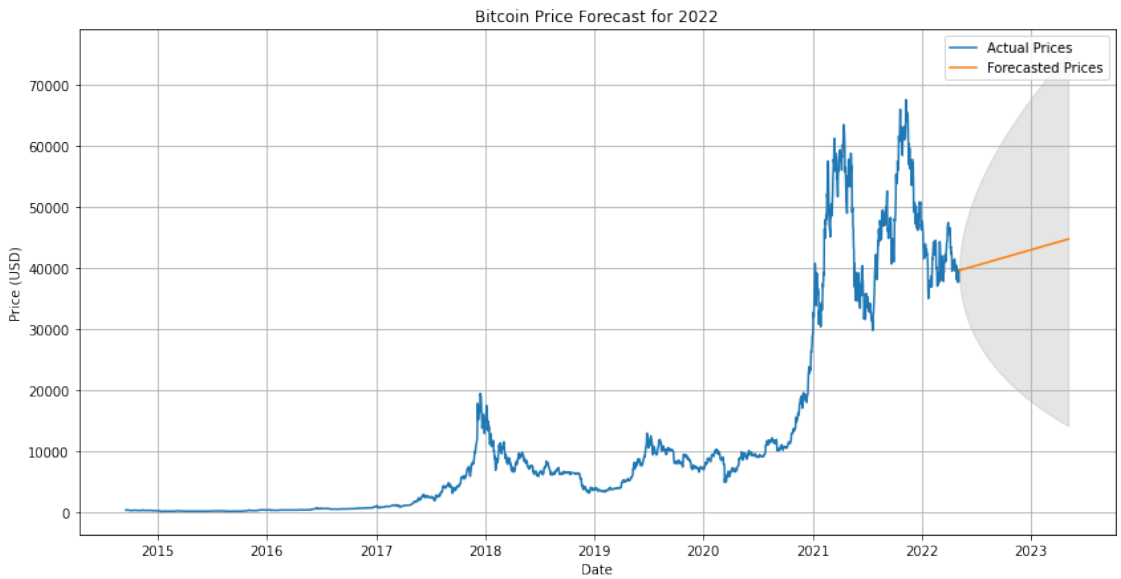
```
/opt/conda/lib/python3.8/site-packages/statsmodels/tsa/base/tsa_model.py:524:
ValueWarning: No frequency information was provided, so inferred frequency D
will be used.
  warnings.warn('No frequency information was'
/opt/conda/lib/python3.8/site-packages/statsmodels/tsa/base/tsa_model.py:524:
ValueWarning: No frequency information was provided, so inferred frequency D
will be used.
  warnings.warn('No frequency information was'
```

[32]:
```
# Forecast for the next year (365 days)
forecast_full, stderr_full, conf_int_full = model_full_fit.forecast(steps=365)
```

[33]:
```
forecast_full_series = pd.Series(forecast_full, index=pd.
 ↪date_range(start=ts_data.index[-1] + pd.Timedelta(days=1), periods=365))
```

1.11 Plotting the actual data and forecast

[34]:
```
# Plot the actual data and forecast
plt.figure(figsize=(14, 7))
plt.plot(ts_data, label='Actual Prices')
plt.plot(forecast_full_series, label='Forecasted Prices')
plt.fill_between(forecast_full_series.index, conf_int_full[:, 0],␣
 ↪conf_int_full[:, 1], color='k', alpha=0.1)
plt.title('Bitcoin Price Forecast for 2022')
plt.xlabel('Date')
plt.ylabel('Price (USD)')
plt.legend()
plt.grid(True)
plt.show()
```



[35]:
```
forecast_full_series
```

[35]:
```
2022-05-06    39572.237308
2022-05-07    39590.888764
2022-05-08    39658.320649
2022-05-09    39692.798270
2022-05-10    39703.585634
                  …
2023-05-01    44710.212686
2023-05-02    44724.263072
2023-05-03    44738.313458
2023-05-04    44752.363845
2023-05-05    44766.414231
Freq: D, Length: 365, dtype: float64
```

2 Training a Random Forest Classifier

[36]:
```
# Import necessary libraries
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score
```

[47]:
```
# features and target
df['Target'] = (df['Percent Change'] > 0).astype(int)
features = ['Open', 'High', 'Low', 'Close', 'Volume']
X = df[features]
y = df['Target']
```

[54]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪shuffle=False)
```

[48]:
```
#training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[55]:
```
# shapes of the datasets
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(2229, 5) (558, 5) (2229,) (558,)
```

[56]:
```
# Initializing the Random Forest Classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)
```

[57]:
```
# Training the model
rf.fit(X_train, y_train)
```

[57]:
```
RandomForestClassifier(random_state=42)
```

[58]:
```
# Predicting on the test set
y_pred = rf.predict(X_test)
```

[59]:
```
# classification report and accuracy score
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Accuracy Score:", accuracy_score(y_test, y_pred))
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.48      0.97      0.64       266
           1       0.59      0.04      0.08       292

    accuracy                           0.48       558
   macro avg       0.54      0.51      0.36       558
weighted avg       0.54      0.48      0.35       558

Accuracy Score: 0.4838709677419355
```

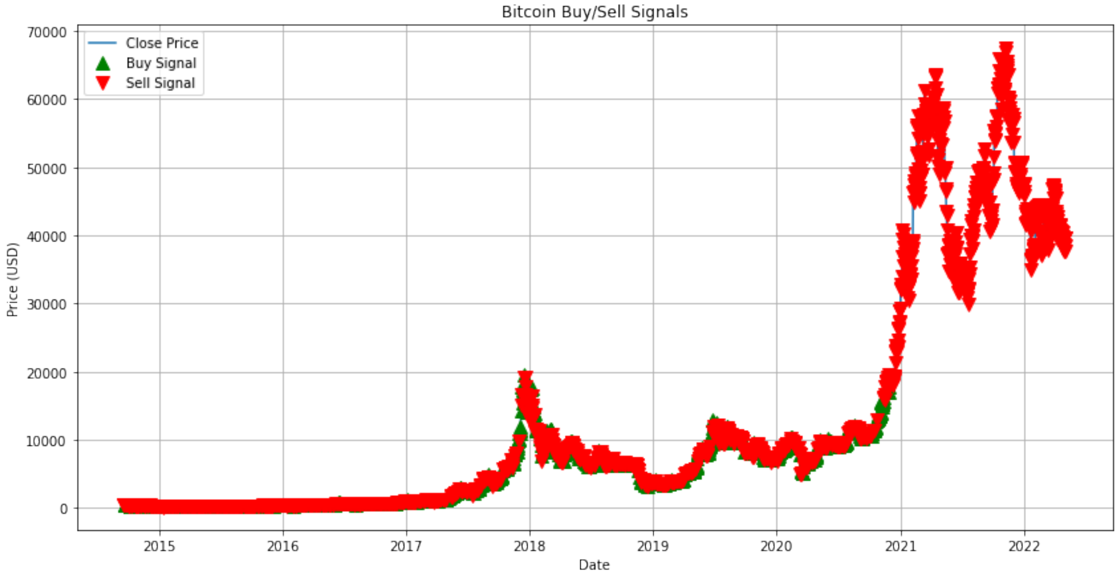
[44]:
```
#predictions for the entire dataset
df['Signal'] = rf.predict(X)
```

[45]:
```
# the first few rows for verifying
df[['Open', 'High', 'Low', 'Close', 'Volume', 'Percent Change', 'Target',␣
↪'Signal']].head()
```

[45]:
```
                  Open        High         Low       Close    Volume  \
Date
2014-09-18  456.859985  456.859985  413.104004  424.440002  34483200
2014-09-19  424.102997  427.834991  384.532013  394.795990  37919700
2014-09-20  394.673004  423.295990  389.882996  408.903992  36863600
2014-09-21  408.084991  412.425995  393.181000  398.821014  26580100
2014-09-22  399.100006  406.915985  397.130005  402.152008  24127600

            Percent Change  Target  Signal
Date
2014-09-18       -7.192558       0       0
2014-09-19       -6.984264       0       0
2014-09-20        3.573492       1       1
2014-09-21       -2.465855       0       0
2014-09-22        0.835210       1       1
```

2.1 Plotting the closing price with buy/sell signals

[46]:
```python
# Plot the closing price with buy/sell signals
plt.figure(figsize=(14, 7))
plt.plot(df['Close'], label='Close Price')
plt.plot(df[df['Signal'] == 1].index, df[df['Signal'] == 1]['Close'], '^',␣
 ↪markersize=10, color='g', label='Buy Signal')
plt.plot(df[df['Signal'] == 0].index, df[df['Signal'] == 0]['Close'], 'v',␣
 ↪markersize=10, color='r', label='Sell Signal')
plt.title('Bitcoin Buy/Sell Signals')
plt.xlabel('Date')
plt.ylabel('Price (USD)')
plt.legend()
plt.grid(True)
plt.show()
```



2.1.1 Summary:

This project conducts a comprehensive analysis of historical Bitcoin price data to uncover trends, identify potential bullish movements, and predict future price changes. Through data visualization, trend analysis, and machine learning techniques, the project provides insights into Bitcoin price dynamics and generates actionable buy/sell signals. The key outputs include visualizations of Bitcoin price trends, identification of significant bullish trends, forecasted Bitcoin prices for 2022, and generation of buy/sell signals based on historical data.

Author: Bahraleloom Abdalrahem **Email:** bahraleloom@gmail.com **GitHub:** https://github.com/Bahraleloom **Kaggle:** https://www.kaggle.com/bahraleloom **Date:** 26/05/2024