cleaning

May 1, 2025

1 Bitcoin Price Predictor – AI306

[1]:
```
!pip install kagglehub
```

```
Requirement already satisfied: kagglehub in
/home/mohammed/anaconda3/envs/Crypto/lib/python3.12/site-packages (0.3.4)
Requirement already satisfied: packaging in
/home/mohammed/anaconda3/envs/Crypto/lib/python3.12/site-packages (from
kagglehub) (24.1)
Requirement already satisfied: requests in
/home/mohammed/anaconda3/envs/Crypto/lib/python3.12/site-packages (from
kagglehub) (2.32.3)
Requirement already satisfied: tqdm in
/home/mohammed/anaconda3/envs/Crypto/lib/python3.12/site-packages (from
kagglehub) (4.67.1)
Requirement already satisfied: charset-normalizer<4,>=2 in
/home/mohammed/anaconda3/envs/Crypto/lib/python3.12/site-packages (from
requests->kagglehub) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in
/home/mohammed/anaconda3/envs/Crypto/lib/python3.12/site-packages (from
requests->kagglehub) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/home/mohammed/anaconda3/envs/Crypto/lib/python3.12/site-packages (from
requests->kagglehub) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in
/home/mohammed/anaconda3/envs/Crypto/lib/python3.12/site-packages (from
requests->kagglehub) (2024.8.30)
```

[2]:
```
import kagglehub
import shutil
import os

import numpy as np
import pandas as pd
```

1.1 Data Preprocessing

1.1.1 Downloading the dataset

[3]:

```
# Download latest version
path = kagglehub.dataset_download("mczielinski/bitcoin-historical-data")

print("Path to dataset files:", path)
```

```
Warning: Looks like you're using an outdated `kagglehub` version, please
consider updating (latest version: 0.3.12)
Downloading from
https://www.kaggle.com/api/v1/datasets/download/mczielinski/bitcoin-historical-
data?dataset_version_number=222…

100%|      | 113M/113M [00:12<00:00, 9.65MB/s]

Extracting files…


Path to dataset files:
/home/mohammed/.cache/kagglehub/datasets/mczielinski/bitcoin-historical-
data/versions/222
```

1.1.2 Copying the dataset to the project directory

[4]:

```
# Source and destination paths
source_path = os.path.join(path, 'btcusd_1-min_data.csv')
destination_path = './data/btcusd_dataset.csv'

# Create destination directory if it doesn't exist
os.makedirs('./data', exist_ok=True)

# Only copy if the file doesn't already exist
if not os.path.exists(destination_path):
    shutil.copy(source_path, destination_path)
    print("File copied successfully.")

else:
    print("File already exists. Skipping copy.")
```

```
File already exists. Skipping copy.
```

1.1.3 Reading dataset

[5]:

```
data = pd.read_csv(destination_path)
```

```
/tmp/ipykernel_3253/1927658998.py:1: DtypeWarning: Columns (6) have mixed types.
Specify dtype option on import or set low_memory=False.
  data = pd.read_csv(destination_path)
```

[6]:

```
data.head()
```

[6]:

```
      Timestamp  Open  High   Low  Close  Volume                   datetime
0  1.325412e+09  4.58  4.58  4.58   4.58     0.0  2012-01-01 10:01:00+00:00
1  1.325412e+09  4.58  4.58  4.58   4.58     0.0  2012-01-01 10:02:00+00:00
2  1.325412e+09  4.58  4.58  4.58   4.58     0.0  2012-01-01 10:03:00+00:00
3  1.325412e+09  4.58  4.58  4.58   4.58     0.0  2012-01-01 10:04:00+00:00
4  1.325412e+09  4.58  4.58  4.58   4.58     0.0  2012-01-01 10:05:00+00:00
```

1.1.4 Info about dataset

[7]:

```
print('Shape of the dataset: ', data.shape)
```

```
Shape of the dataset:  (7001004, 7)
```

[8]:

```
print(f"info of the dataset: \n{data.info()}\n")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7001004 entries, 0 to 7001003
Data columns (total 7 columns):
 #   Column     Dtype
---  ------     -----
 0   Timestamp  float64
 1   Open       float64
 2   High       float64
 3   Low        float64
 4   Close      float64
 5   Volume     float64
 6   datetime   object
dtypes: float64(6), object(1)
memory usage: 373.9+ MB
info of the dataset:
None
```

[9]:

```
print(f"describe of the dataset: \n{data.describe()}\n")
```

```
describe of the dataset:
          Timestamp          Open          High           Low         Close  \
count  7.001004e+06  7.001004e+06  7.001004e+06  7.001004e+06  7.001004e+06
mean   1.535443e+09  1.729476e+04  1.730170e+04  1.728760e+04  1.729476e+04
std    1.212619e+08  2.389940e+04  2.390744e+04  2.389117e+04  2.389938e+04
min    1.325412e+09  3.800000e+00  3.800000e+00  3.800000e+00  3.800000e+00
25%    1.430427e+09  4.239100e+02  4.240000e+02  4.237600e+02  4.239300e+02
50%    1.535442e+09  6.575210e+03  6.578515e+03  6.572320e+03  6.575290e+03
75%    1.640457e+09  2.720000e+04  2.720400e+04  2.719600e+04  2.720000e+04
max    1.745542e+09  1.091110e+05  1.093560e+05  1.087940e+05  1.090360e+05
```

```
             Volume
count  7.001004e+06
mean   5.308327e+00
std    2.253495e+01
min    0.000000e+00
25%    1.815710e-02
50%    4.703309e-01
75%    3.039586e+00
max    5.853852e+03
```

[10]:

```
print(f"null values of the dataset: \n{data.isnull().sum()}\n")
```

```
null values of the dataset:
Timestamp         0
Open              0
High              0
Low               0
Close             0
Volume            0
datetime     218724
dtype: int64
```

[11]:

```
print(f"duplicated values of the dataset: \n{data.duplicated().sum()}\n")
```

```
duplicated values of the dataset:
0
```

1.1.5 Cleaning Dataset

[12]:

```
# Drop the 'datetime' column
data = data.drop(columns=['datetime'])

# view the updated DataFrame
print(data.head())
print(f"\n\nnull values of the dataset: \n{data.isnull().sum()}\n")
```

```
      Timestamp  Open  High   Low  Close  Volume
0  1.325412e+09  4.58  4.58  4.58   4.58     0.0
1  1.325412e+09  4.58  4.58  4.58   4.58     0.0
2  1.325412e+09  4.58  4.58  4.58   4.58     0.0
3  1.325412e+09  4.58  4.58  4.58   4.58     0.0
4  1.325412e+09  4.58  4.58  4.58   4.58     0.0


null values of the dataset:
```

```
Timestamp    0
Open         0
High         0
Low          0
Close        0
Volume       0
dtype: int64
```

[13]:
```
# Convert Unix timestamp (seconds since 00:00:00 UTC January 1, 1970) to␣
  ↪datetime
data['datetime'] = pd.to_datetime(data['Timestamp'], unit='s')

print(data.head())
```

```
      Timestamp  Open  High   Low  Close  Volume            datetime
0  1.325412e+09  4.58  4.58  4.58   4.58     0.0 2012-01-01 10:01:00
1  1.325412e+09  4.58  4.58  4.58   4.58     0.0 2012-01-01 10:02:00
2  1.325412e+09  4.58  4.58  4.58   4.58     0.0 2012-01-01 10:03:00
3  1.325412e+09  4.58  4.58  4.58   4.58     0.0 2012-01-01 10:04:00
4  1.325412e+09  4.58  4.58  4.58   4.58     0.0 2012-01-01 10:05:00
```

[14]:
```
# ensure the data is continuous and their are no missing values or rows,
# Reindexes the data to have a row for every minute - even if that minute was␣
  ↪missing in the original data.
continuous_data = data.set_index('datetime').asfreq('min')
print('data Null/NA Values before fill:', continuous_data.isnull().values.sum())

# fill in and interpolate missing values after re-indexing is done
continuous_data.interpolate(method='time', inplace=True)  # Time-based␣
  ↪interpolation
continuous_data.ffill(inplace=True) # forwards fill missing values

continuous_data.reset_index(inplace=True) # Moves 'datetime' back from the␣
  ↪index to a regular column
print('data Null/NA Values after fill:', continuous_data.isnull().values.sum())

data = continuous_data.copy()
```

```
data Null/NA Values before fill: 6960
data Null/NA Values after fill: 0
```

[15]:
```
first_nonzero_row = data[data['Volume'] > 0].head(1)
print(first_nonzero_row)
```

```
               datetime     Timestamp  Open  High   Low  Close  Volume
627 2012-01-01 20:28:00  1.325450e+09  4.84  4.84  4.84   4.84    10.0
```

[16]:
```
# Save cleaned data to csv file
data.to_csv('./data/cleaned_data.csv', index=False)
```

1.1.6 Data Reduction

[17]:
```
data = data.drop(columns=['Volume'])
data.head()
```

[17]:
```
             datetime     Timestamp  Open  High   Low  Close
0 2012-01-01 10:01:00  1.325412e+09  4.58  4.58  4.58   4.58
1 2012-01-01 10:02:00  1.325412e+09  4.58  4.58  4.58   4.58
2 2012-01-01 10:03:00  1.325412e+09  4.58  4.58  4.58   4.58
3 2012-01-01 10:04:00  1.325412e+09  4.58  4.58  4.58   4.58
4 2012-01-01 10:05:00  1.325412e+09  4.58  4.58  4.58   4.58
```

[ ]:
```
def create_resampled_dataframe(data, resample_rule='h'):
    """
    Create a resampled DataFrame for a given time resolution.

    Args:
        data (pd.DataFrame): Input DataFrame
        resample_rule (str): Resampling rule (e.g., 'h', 'D', 'W-MON', 'M')

    Returns:
        pd.DataFrame: Resampled DataFrame

    """
    datetime_column = data['datetime']

    if resample_rule == '1min' or resample_rule is None:
        return data

    df = pd.DataFrame()
    df['Timestamp'] = data.set_index(datetime_column)['Timestamp'].
 ↪resample(resample_rule).first()
    df['Open'] = data.set_index(datetime_column)['Open'].
 ↪resample(resample_rule).first()
    df['High'] = data.set_index(datetime_column)['High'].
 ↪resample(resample_rule).max()
    df['Low'] = data.set_index(datetime_column)['Low'].resample(resample_rule).
 ↪min()
    df['Close'] = data.set_index(datetime_column)['Close'].
 ↪resample(resample_rule).last()
    if 'Volume' in data.columns:
        df['Volume'] = data.set_index(datetime_column)['Volume'].
 ↪resample(resample_rule).sum()
```

```
    print('Null/NA Values in resample dataframe:', df.isnull().values.sum())
    df = df.dropna()
    print('Shape of the dataset: ', df.shape)

    return df
```

[29]:

```
reduced_data = create_resampled_dataframe(data, resample_rule='h')
reduced_data.head()
```

```
Null/NA Values in resample dataframe: 0
Shape of the dataset:  (116703, 5)
```

[29]:

```
                        Timestamp  Open  High   Low  Close
datetime
2012-01-01 10:00:00  1.325412e+09  4.58  4.58  4.58   4.58
2012-01-01 11:00:00  1.325416e+09  4.58  4.58  4.58   4.58
2012-01-01 12:00:00  1.325419e+09  4.58  4.58  4.58   4.58
2012-01-01 13:00:00  1.325423e+09  4.58  4.58  4.58   4.58
2012-01-01 14:00:00  1.325426e+09  4.58  4.58  4.58   4.58
```

1.1.7 Data Transformation (Feature Engineering)

[26]:

```
def add_indicators(data):
    """
    Add technical indicators to the DataFrame.

    Args:
        data (pd.DataFrame): Input DataFrame

    Returns:
        pd.DataFrame: DataFrame with added indicators
    """
    # Calculate moving averages to add to the dataframe
    # moving averages are used to capture the trend of the data by averaging␣
  ↪the past values over a period
    sma_200 = data['Close'].rolling(window=200).mean()


    # Calculate Average True Range (ATR)
    # ATR is a volatility indicator that measures the true range over a period
    # It works by comparing the highest and lowest prices over a period
    # Calculate True Range (TR)
    high_low = data['High'] - data['Low']
    high_close_prev = abs(data['High'] - data['Close'].shift(1))
    low_close_prev = abs(data['Low'] - data['Close'].shift(1))
    true_range = pd.concat([high_low, high_close_prev, low_close_prev], axis=1).
 ↪max(axis=1)

    # Calculate ATR using a rolling average of the True Range
    atr_period = 168  # Or any period of choice
    atr_168 = true_range.rolling(window=atr_period).mean()


    data['SMA_200'] = sma_200
    data['ATR_168'] = atr_168

    print('Null/NA Values in engineered dataframe:', data.isnull().values.sum())
    data = data.dropna()
    print('Shape of the dataset: ', data.shape)

    return data
```

```
[ ]: # this is the cleaned, reduced (minute > hourly) and engineered data
     engineered_data = add_indicators(reduced_data)
     engineered_data.head()
```

```
Null/NA Values in engineered dataframe: 366
Shape of the dataset:  (116504, 7)
```

```
[ ]:                        Timestamp  Open  High  Low  Close  SMA_200   ATR_168
     datetime
     2012-01-09 17:00:00  1.326128e+09   6.9   6.9  6.5    6.5  5.83495  0.032560
     2012-01-09 18:00:00  1.326132e+09   6.5   6.6  6.5    6.5  5.84455  0.033155
     2012-01-09 19:00:00  1.326136e+09   6.5   6.6  6.5    6.6  5.85465  0.033750
     2012-01-09 20:00:00  1.326139e+09   6.6   6.6  6.6    6.6  5.86475  0.033750
     2012-01-09 21:00:00  1.326143e+09   6.6   6.6  6.6    6.6  5.87485  0.033750
```

```
[30]: data = engineered_data.reset_index() # reset the index so that we can use it as␣
       ↪a column
      data.head()
```

```
[30]:              datetime     Timestamp  Open  High  Low  Close  SMA_200   ATR_168
      0 2012-01-09 17:00:00  1.326128e+09   6.9   6.9  6.5    6.5  5.83495  0.032560
      1 2012-01-09 18:00:00  1.326132e+09   6.5   6.6  6.5    6.5  5.84455  0.033155
      2 2012-01-09 19:00:00  1.326136e+09   6.5   6.6  6.5    6.6  5.85465  0.033750
      3 2012-01-09 20:00:00  1.326139e+09   6.6   6.6  6.6    6.6  5.86475  0.033750
      4 2012-01-09 21:00:00  1.326143e+09   6.6   6.6  6.6    6.6  5.87485  0.033750
```

```
[31]: # Save cleaned data to csv file
      data.to_csv('./data/preprocessed_hourly_data.csv', index=False)
```