Unleashing the Power of Deep Reinforcement Learning for Trading

Exploring the Intersection of Reinforcement Learning and Neural Networks

Table of Contents

Project Overview

Introduction

Financial markets are a dynamic, complex ecosystem where traditional trading strategies are continually challenged by evolving market conditions, global events, and intricate patterns that define asset prices. In response to this ever-changing landscape, our team has embarked on a pioneering journey to develop a sophisticated DRL-based algorithmic trading system.

Deep Reinforcement Learning, a subset of artificial intelligence, offers the promise of adaptive, self-improving trading strategies. Inspired by the way humans learn through trial and error, our DRL model navigates financial markets, learning and adapting to new information in real-time, all with the goal of maximizing returns while managing risk.

This project represents a convergence of finance, machine learning, and technology. Our DRL model is designed to make informed trading decisions, considering historical data, market indicators, and other relevant factors, ultimately offering the potential to

outperform traditional trading methods. The ability to harness the power of neural networks, paired with reinforcement learning principles, promises a new era in algorithmic trading.

In this project, we will explore the development and implementation of our DRL model, highlighting its capabilities, challenges, and real-world implications in the realm of financial markets. We invite you to journey with us through the intricacies of this innovative approach, where artificial intelligence meets finance, and together, we aim to shape the future of trading.

Objective & Scope

- Collect, clean & analyze Bitcoin crypto market data
- Features Engineering
- Build a DRL trading environment
- Train & improve the agent

Deep Reinformencement Learning Model

- **Reinforcement Learning Algorithm (RLA):** PPO (Proximal Policy Optimization)
- **Neural Network Architecture for Policy (Agent's action):** MLP Policy (Multi-Layer Perceptron Policy)
    - **Architecture Details:**
        - **Input Layer:** The MLP policy takes observations from the environment as input.
        - **Hidden Layers:** The network typically consists of one or more hidden layers, each containing multiple neurons. The specific number of layers and neurons may vary depending on the complexity of the problem.
        - **Activation Functions:** Each layer in the MLP policy may utilize activation functions like ReLU (Rectified Linear Unit) or Tanh.
        - **Output Layer**: The output layer provides the probabilities or values associated with different actions the agent can take.
    - **Training Process:**

- The model is trained using PPO, which involves optimizing the policy network to maximize cumulative rewards while ensuring that policy updates are within a safe margin.

Importing & Downloading Libraries

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
!pip install mplfinance
```

```
Collecting mplfinance
  Downloading mplfinance-0.12.10b0-py3-none-any.whl (75 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 75.0/75.0 kB 1.7 MB/s eta 0:00:00
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from mplfinance) (3.7.1)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from mplfinance) (1.5.3)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->mplfinance
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->mplfinance) (0
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->mplfinanc
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->mplfinanc
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-packages (from matplotlib->mplfinance) (1.
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->mplfinance)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->mplfinance) (
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->mplfinance
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib->mplfin
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->mplfinance) (2023.
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotl
Installing collected packages: mplfinance
Successfully installed mplfinance-0.12.10b0
```

```
!pip install pgmpy
!pip install pandas-ta
```

```
Collecting pgmpy
  Downloading pgmpy-0.1.24-py3-none-any.whl (2.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.0/2.0 MB 24.1 MB/s eta 0:00:00
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from pgmpy) (3.2.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from pgmpy) (1.25.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from pgmpy) (1.11.4)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (from pgmpy) (1.2.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from pgmpy) (1.5.3)
Requirement already satisfied: pyparsing in /usr/local/lib/python3.10/dist-packages (from pgmpy) (3.1.1)
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (from pgmpy) (2.1.0+cu121)
Requirement already satisfied: statsmodels in /usr/local/lib/python3.10/dist-packages (from pgmpy) (0.14.1)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from pgmpy) (4.66.2)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from pgmpy) (1.3.2)
Requirement already satisfied: opt-einsum in /usr/local/lib/python3.10/dist-packages (from pgmpy) (3.3.0)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas->pgmpy) (
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->pgmpy) (2023.4)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->pgmp
Requirement already satisfied: patsy>=0.5.4 in /usr/local/lib/python3.10/dist-packages (from statsmodels->pgmpy) (0.5.6
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from statsmodels->pgmpy) (23
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch->pgmpy) (3.13.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from torch->pgmpy) (4.9.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch->pgmpy) (1.12)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch->pgmpy) (3.1.3)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch->pgmpy) (2023.6.0)
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dist-packages (from torch->pgmpy) (2.1.0)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.4->statsmodels->pgmpy) (
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch->pgmpy) (
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch->pgmpy) (1.3.
Installing collected packages: pgmpy
Successfully installed pgmpy-0.1.24
Collecting pandas-ta
  Downloading pandas_ta-0.3.14b.tar.gz (115 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 115.1/115.1 kB 7.9 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from pandas-ta) (1.5.3)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas->pandas-t
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->pandas-ta) (2023.4
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/dist-packages (from pandas->pandas-ta) (1.25.
```

```
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas
Building wheels for collected packages: pandas-ta
  Building wheel for pandas-ta (setup.py) ... done
  Created wheel for pandas-ta: filename=pandas_ta-0.3.14b0-py3-none-any.whl size=218907 sha256=91d87a0f0437ad983a4ccd5b
  Stored in directory: /root/.cache/pip/wheels/69/00/ac/f7fa862c34b0e2ef320175100c233377b4c558944f12474cf0
Successfully built pandas-ta
Installing collected packages: pandas-ta
Successfully installed pandas-ta-0.3.14b0
```

```
!pip install stable-baselines3
```

```
Collecting stable-baselines3
  Downloading stable_baselines3-2.2.1-py3-none-any.whl (181 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 181.7/181.7 kB 3.0 MB/s eta 0:00:00
Collecting gymnasium<0.30,>=0.28.1 (from stable-baselines3)
  Downloading gymnasium-0.29.1-py3-none-any.whl (953 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 953.9/953.9 kB 35.3 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-packages (from stable-baselines3) (1.25.2)
Requirement already satisfied: torch>=1.13 in /usr/local/lib/python3.10/dist-packages (from stable-baselines3) (2.1.0+c
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.10/dist-packages (from stable-baselines3) (2.2.1)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from stable-baselines3) (1.5.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from stable-baselines3) (3.7.1)
Requirement already satisfied: typing-extensions>=4.3.0 in /usr/local/lib/python3.10/dist-packages (from gymnasium<0.30
Collecting farama-notifications>=0.0.1 (from gymnasium<0.30,>=0.28.1->stable-baselines3)
  Downloading Farama_Notifications-0.0.4-py3-none-any.whl (2.5 kB)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch>=1.13->stable-baselines3
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch>=1.13->stable-baselines3) (
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch>=1.13->stable-baselines3
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch>=1.13->stable-baselines3)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch>=1.13->stable-baselines3)
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dist-packages (from torch>=1.13->stable-basel
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->stable-bas
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->stable-baselin
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->stable-ba
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->stable-ba
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->stable-base
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->stable-baseli
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->stable-bas
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib->stable
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->stable-baselines3)
```

```
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotl
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch>=1.13->st
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch>=1.13->stable
Installing collected packages: farama-notifications, gymnasium, stable-baselines3
Successfully installed farama-notifications-0.0.4 gymnasium-0.29.1 stable-baselines3-2.2.1
```

```
!pip install 'shimmy>=0.2.1'
```

```
Collecting shimmy>=0.2.1
  Downloading Shimmy-1.3.0-py3-none-any.whl (37 kB)
Requirement already satisfied: numpy>=1.18.0 in /usr/local/lib/python3.10/dist-packages (from shimmy>=0.2.1) (1.25.2)
Requirement already satisfied: gymnasium>=0.27.0 in /usr/local/lib/python3.10/dist-packages (from shimmy>=0.2.1) (0.29.
Requirement already satisfied: cloudpickle>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from gymnasium>=0.27.0->s
Requirement already satisfied: typing-extensions>=4.3.0 in /usr/local/lib/python3.10/dist-packages (from gymnasium>=0.2
Requirement already satisfied: farama-notifications>=0.0.1 in /usr/local/lib/python3.10/dist-packages (from gymnasium>=
Installing collected packages: shimmy
Successfully installed shimmy-1.3.0
```

```
import pandas as pd
import gym
import numpy as np
from stable_baselines3 import PPO
from stable_baselines3.common.vec_env import DummyVecEnv
import warnings
warnings.filterwarnings("ignore")
```

```
import numpy as np
import pandas as pd
from pandas.plotting import scatter_matrix
#from pandas_ta import wma, macd, stoch, adx, atr, hma, kama, cmo, zscore, qstick

import calendar
import os
import random
import itertools

from matplotlib.colors import TwoSlopeNorm, LinearSegmentedColormap
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
import matplotlib.cm as cm
from matplotlib.cm import ScalarMappable
from matplotlib.colors import Normalize
import seaborn as sns
import plotly.express as px

from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.seasonal import STL
#from statsmodels.stats.outliers_influence import variance_inflation_factor

from scipy.stats import pearsonr
from scipy import stats
from scipy.stats.mstats import winsorize
from scipy.signal import periodogram

import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter('ignore')

import networkx as nx
```

Data Engineering

```
df = pd.read_csv('/content/drive/MyDrive/IS2/crypto_data.csv')
```

```
df.head()
```

	date	symbol	open	high	low	close	volume usdt	tradecount	token	hour	day
0	2020-12-25 05:00:00	1INCHUSDT	0.2000	3.0885	0.2000	2.5826	35530516	48768	1INCH	5	Friday
1	2020-12-25 06:00:00	1INCHUSDT	2.5824	2.6900	2.2249	2.5059	22440875	31099	1INCH	6	Friday
2	2020-12-25 07:00:00	1INCHUSDT	2.5152	2.8870	2.3609	2.6237	21300426	33001	1INCH	7	Friday
3	2020-12-25 08:00:00	1INCHUSDT	2.6318	2.8247	2.4650	2.6134	17491813	30459	1INCH	8	Friday
4	2020-12-25 09:00:00	1INCHUSDT	2.6104	2.7498	2.5629	2.6365	9919400	21023	1INCH	9	Friday

```
df.shape
```

```
(1120061, 11)
```

```
df.columns
```

```
Index(['date', 'symbol', 'open', 'high', 'low', 'close', 'volume usdt',
       'tradecount', 'token', 'hour', 'day'],
      dtype='object')
```

```
start_date = '2020-08-17 04:00:00'
end_date = '2023-10-19 23:00:00'
```

```
data_df = df.copy(deep=True)
```

```
data_df = data_df[(data_df['token'] == 'BTC') & (data_df['date'] >= start_date) & (data_df['date'] <= end_date)]


data_df.columns
```

```
     Index(['date', 'open', 'high', 'low', 'close', 'volume usdt', 'tradecount',
            'hour', 'day', 'ema_13', 'ema_25', 'ema_32', 'ema_100', 'ema_200',
            'vol_close', 'vol_close_ema_3', 'vol_close_ema_6', 'vol_close_ema_12'],
           dtype='object')
```

Mapping 'day' days to numerical values

```
day_mapping = {
    'Monday': 1,
    'Tuesday': 2,
    'Wednesday': 3,
    'Thursday': 4,
    'Friday': 5,
    'Saturday': 6,
    'Sunday': 7
}

data_df['day'] = data_df['day'].apply(lambda x: day_mapping[x])


first_appearance = df.groupby('token')['date'].min()
token_counts = df['token'].value_counts()
token_info = pd.DataFrame({'Count': token_counts, 'First Appearance': first_appearance})
token_info['Number of Days'] = (token_info['Count'] / 24).astype(int)
token_info = token_info.sort_values(by='Number of Days', ascending=False)

print(token_info)
```

```
             Count     First Appearance  Number of Days
     ETH     54116  2017-08-17 04:00:00            2254
     BTC     54116  2017-08-17 04:00:00            2254
     BNB     52032  2017-11-06 03:00:00            2168
```

```
LTC    51285  2017-12-13 03:00:00           2136
ADA    48198  2018-04-17 04:00:00           2008
ETC    46942  2018-06-12 02:00:00           1955
TRX    46905  2018-06-11 11:00:00           1954
LINK   41665  2019-01-16 10:00:00           1736
XMR    40316  2019-03-15 04:00:00           1679
MATIC  39243  2019-04-26 15:00:00           1635
ATOM   39182  2019-04-29 04:00:00           1632
ALGO   37900  2019-06-22 00:00:00           1579
DOGE   37576  2019-07-05 12:00:00           1565
DENT   36320  2019-08-27 04:00:00           1513
RVN    35616  2019-09-25 12:00:00           1484
HBAR   35528  2019-09-29 04:00:00           1480
BCH    34086  2019-11-28 10:00:00           1420
COMP   29059  2020-06-25 06:00:00           1210
MKR    28382  2020-07-23 14:00:00           1182
SOL    27934  2020-08-11 06:00:00           1163
BAL    27934  2020-08-11 06:00:00           1163
CRV    27840  2020-08-15 04:00:00           1160
DOT    27749  2020-08-18 23:00:00           1156
SUSHI  27445  2020-09-01 11:00:00           1143
UNI    27069  2020-09-17 03:00:00           1127
AVAX   26926  2020-09-22 06:00:00           1121
AAVE   26373  2020-10-15 03:00:00           1098
FIL    26363  2020-10-15 17:00:00           1098
1INCH  24672  2020-12-25 05:00:00           1028
SHIB   21413  2021-05-10 11:00:00            892
ICP    21399  2021-05-11 01:00:00            891
DYDX   18477  2021-09-09 02:00:00            769
```

```
unique_tokens = df['token'].unique()
```

```
unique_tokens
```

```
array(['1INCH', 'AAVE', 'ADA', 'ALGO', 'ATOM', 'AVAX', 'BAL', 'BCH',
       'BNB', 'BTC', 'COMP', 'CRV', 'DENT', 'DOGE', 'DOT', 'DYDX', 'ETC',
       'ETH', 'FIL', 'HBAR', 'ICP', 'LINK', 'LTC', 'MATIC', 'MKR', 'RVN',
       'SHIB', 'SOL', 'SUSHI', 'TRX', 'UNI', 'XMR'], dtype=object)
```

EDA

```
data = pd.read_csv('/content/drive/MyDrive/IS2/crypto_data.csv')


data['date'] = pd.to_datetime(data['date'], format='%Y-%m-%d %H:%M:%S')
data['month'] = data['date'].dt.month
data['week'] = data['date'].dt.isocalendar().week
data['week'] = data['week'].astype(int)
data['year'] = data['date'].dt.year


# new feature 'volatility'(%)
data['volatility'] = (data['high'] - data['low']) / data['low'] * 100
data['hc_change'] = (data['close'] - data['high']) / data['high'] * 100
data['lc_change'] = (data['close'] - data['low']) / data['low'] * 100


#new feature 'price_change_percent' (change in % from close to close)

data = data.sort_values(by=['token', 'date'])
data['price_change_percent'] = (data.groupby('token')['close'].pct_change() * 100).round(2)


data.head()
```

	date	symbol	open	high	low	close	volume usdt	tradecount	token	hour	day	month	week	year	vola
0	2020-12-25 05:00:00	1INCHUSDT	0.2000	3.0885	0.2000	2.5826	35530516	48768	1INCH	5	Friday	12	52	2020	1444.
1	2020-12-25 06:00:00	1INCHUSDT	2.5824	2.6900	2.2249	2.5059	22440875	31099	1INCH	6	Friday	12	52	2020	20.

```
#renaming column

data.rename(columns={'volume usdt': 'volume'}, inplace=True)
data.shape
```

```
(1120061, 18)
```

```
nan_values = data.isna()
any_missing_values = nan_values.any().any()

any_missing_values_in_column = nan_values.any()

missing_value_count = data.isnull().sum()

for column, has_missing in any_missing_values_in_column.items():
    if has_missing:
        count = missing_value_count[column]
        print(f"-----> Column '{column}' has {count} missing values.")

print("\nMissing Values in the Entire DataFrame?")
print(any_missing_values)

print("\nMissing Values in Each Column?")
print(any_missing_values_in_column)

print("\nMissing Value Counts in Each Column:")
print(missing_value_count)
```

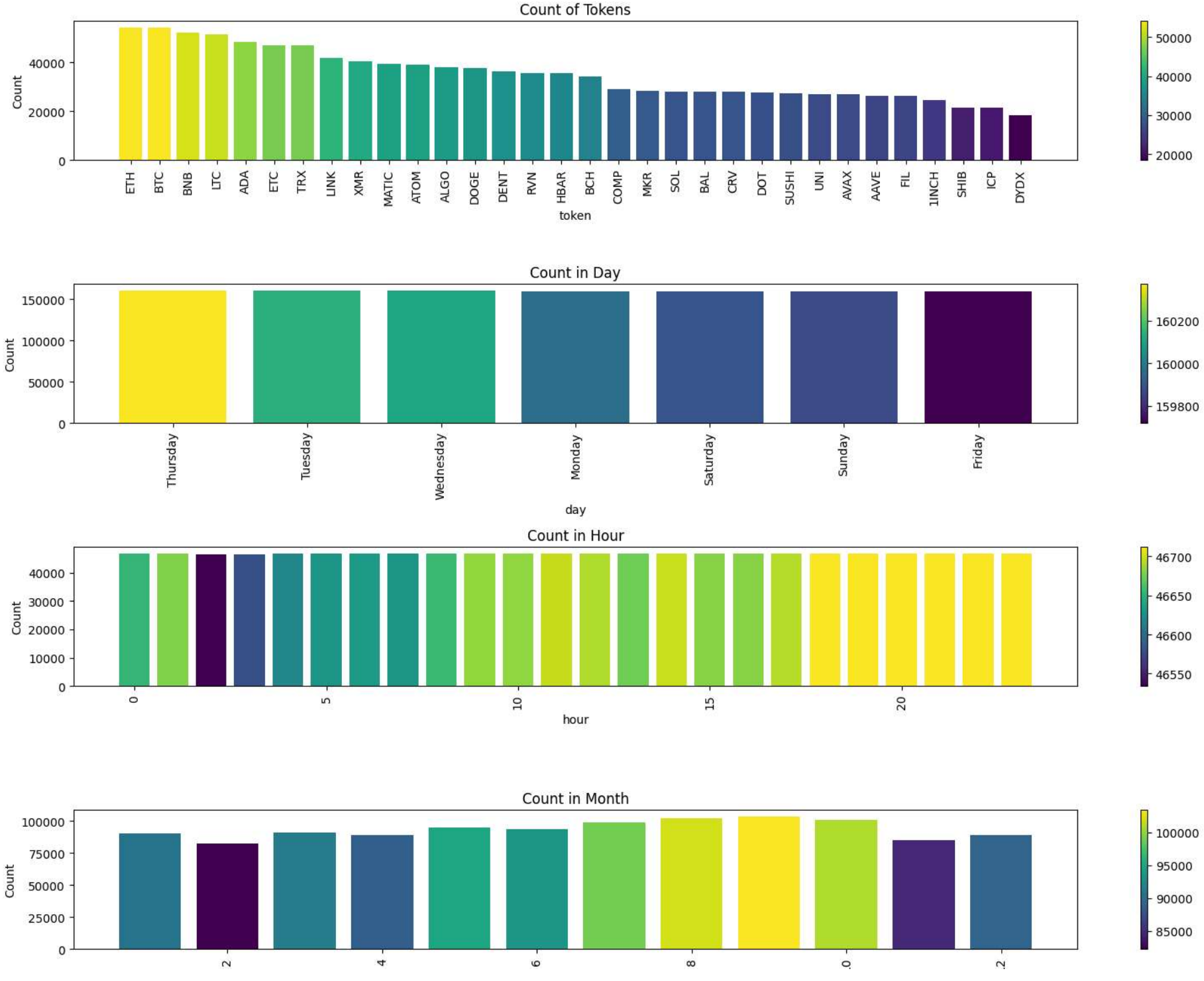
```
-----> Column 'price_change_percent' has 32 missing values.

Missing Values in the Entire DataFrame?
True

Missing Values in Each Column?
date                    False
symbol                  False
open                    False
high                    False
low                     False
```

```
close                  False
volume                 False
tradecount             False
token                  False
hour                   False
day                    False
month                  False
week                   False
year                   False
volatility             False
hc_change              False
lc_change              False
price_change_percent    True
dtype: bool

Missing Value Counts in Each Column:
date                    0
symbol                  0
open                    0
high                    0
low                     0
close                   0
volume                  0
tradecount              0
token                   0
hour                    0
day                     0
month                   0
week                    0
year                    0
volatility              0
hc_change               0
lc_change               0
price_change_percent    32
dtype: int64


#We are expecting 32 unique tokens. Which explains the 32 missing values for price_change

un_token = data['token'].nunique()
un_token
```

```
32


#The first row of each token has a NaN value for price_change'. Normal because there was no input before since it's the fir

data['price_change_percent'] = data.groupby('token')['price_change_percent'].fillna(0)
data.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 1120061 entries, 0 to 1120060
Data columns (total 18 columns):
 #   Column                Non-Null Count    Dtype
---  ------                --------------    -----
 0   date                  1120061 non-null  datetime64[ns]
 1   symbol                1120061 non-null  object
 2   open                  1120061 non-null  float64
 3   high                  1120061 non-null  float64
 4   low                   1120061 non-null  float64
 5   close                 1120061 non-null  float64
 6   volume                1120061 non-null  int64
 7   tradecount            1120061 non-null  int64
 8   token                 1120061 non-null  object
 9   hour                  1120061 non-null  int64
 10  day                   1120061 non-null  object
 11  month                 1120061 non-null  int64
 12  week                  1120061 non-null  int64
 13  year                  1120061 non-null  int64
 14  volatility            1120061 non-null  float64
 15  hc_change             1120061 non-null  float64
 16  lc_change             1120061 non-null  float64
 17  price_change_percent  1120061 non-null  float64
dtypes: datetime64[ns](1), float64(8), int64(6), object(3)
memory usage: 162.4+ MB


data_df = data.copy(deep=True)
df = data.copy(deep=True)
df2 = data.copy(deep=True)
df3 = data.copy(deep=True)
```

```
#Checking all unique tokens in the dataset

u_token = df['token'].unique()
u_token

     array(['1INCH', 'AAVE', 'ADA', 'ALGO', 'ATOM', 'AVAX', 'BAL', 'BCH',
            'BNB', 'BTC', 'COMP', 'CRV', 'DENT', 'DOGE', 'DOT', 'DYDX', 'ETC',
            'ETH', 'FIL', 'HBAR', 'ICP', 'LINK', 'LTC', 'MATIC', 'MKR', 'RVN',
            'SHIB', 'SOL', 'SUSHI', 'TRX', 'UNI', 'XMR'], dtype=object)


t_counts = df['token'].value_counts()
tc_df = t_counts.reset_index()
tc_df.columns = ['token', 'count']
tc_df = tc_df.sort_values(by='count', ascending=False)

d_counts = df['day'].value_counts()
d_df = d_counts.reset_index()
d_df.columns = ['day', 'count']
d_df = d_df.sort_values(by='count', ascending=False)

h_counts = df['hour'].value_counts()
h_df = h_counts.reset_index()
h_df.columns = ['hour', 'count']
h_df = h_df.sort_values(by='count', ascending=False)

m_counts = df['month'].value_counts()
m_df = m_counts.reset_index()
m_df.columns = ['Month', 'count']
m_df = m_df.sort_values(by='count', ascending=False)
```

```
fig, axs = plt.subplots(4, 1, figsize=(16, 12))

for i, (df, title) in enumerate([(tc_df, 'Count of Tokens'),
                                 (d_df, 'Count in Day'),
                                 (h_df, 'Count in Hour'),
                                 (m_df, 'Count in Month')]):
    colormap = plt.cm.get_cmap('viridis')
    norm = Normalize(vmin=df['count'].min(), vmax=df['count'].max())

    bars = axs[i].bar(df[df.columns[0]], df['count'], color=colormap(norm(df['count'])))

    axs[i].set_xlabel(df.columns[0])
    axs[i].set_ylabel('Count')
    axs[i].set_title(title)
    axs[i].tick_params(axis='x', rotation=90)

    sm = ScalarMappable(cmap=colormap, norm=norm)
    sm.set_array([])
    cbar = plt.colorbar(sm, ax=axs[i])

plt.tight_layout()
plt.show()
```


Count of Tokens
Count
token
ETH
BTC
BNB
LTC
ADA
ETC
TRX
LINK
XMR
MATIC
ATOM
ALGO
DOGE
DENT
RVN
HBAR
BCH
COMP
MKR
SOL
BAL
CRV
DOT
SUSHI
UNI
AVAX
AAVE
FIL
1INCH
SHIB
ICP
DYDX
Count in Day
Count
day
Thursday
Tuesday
Wednesday
Monday
Saturday
Sunday
Friday
Count in Hour
Count
hour
Count in Month
Count

```
df2['symbol'] = label_encoder.fit_transform(df2['symbol'])

day_mapping = {
    'Monday': 0,
    'Tuesday': 1,
    'Wednesday': 2,
    'Thursday': 3,
    'Friday': 4,
    'Saturday': 5,
    'Sunday': 6
}

df2['day'] = df2['day'].map(day_mapping)


scat_filt = ['hour', 'day', 'week', 'month', 'volatility', 'tradecount', 'volume']

#btc_scat = df2[(df2['token'] == 'BTC') & (df2['year'] == 2020)]
btc_scat = df2[(df2['token'] == 'BTC')]

btc_scat = btc_scat[scat_filt]
```

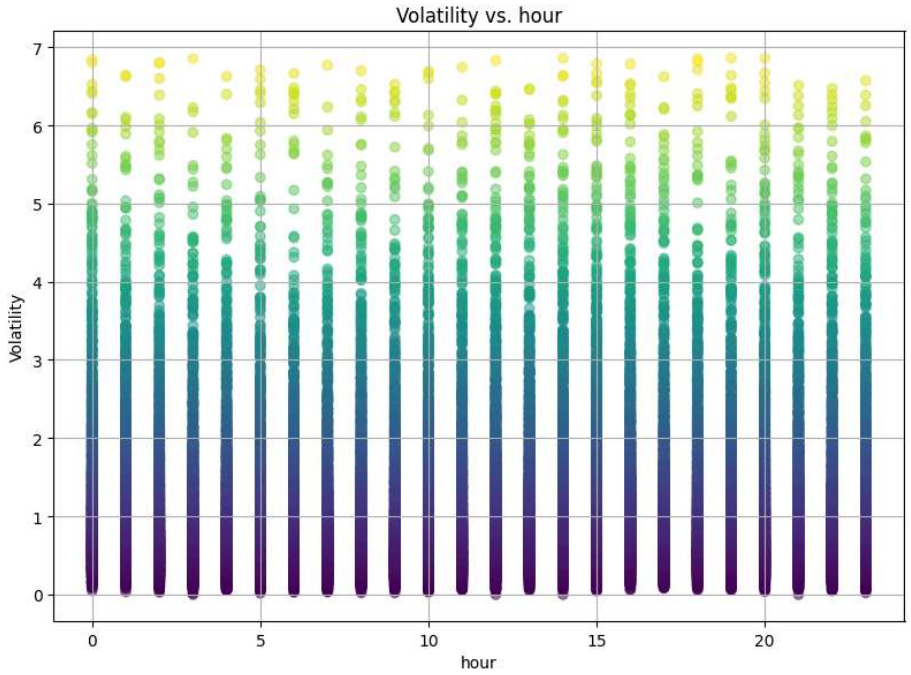
```
btc_scat.info()
```

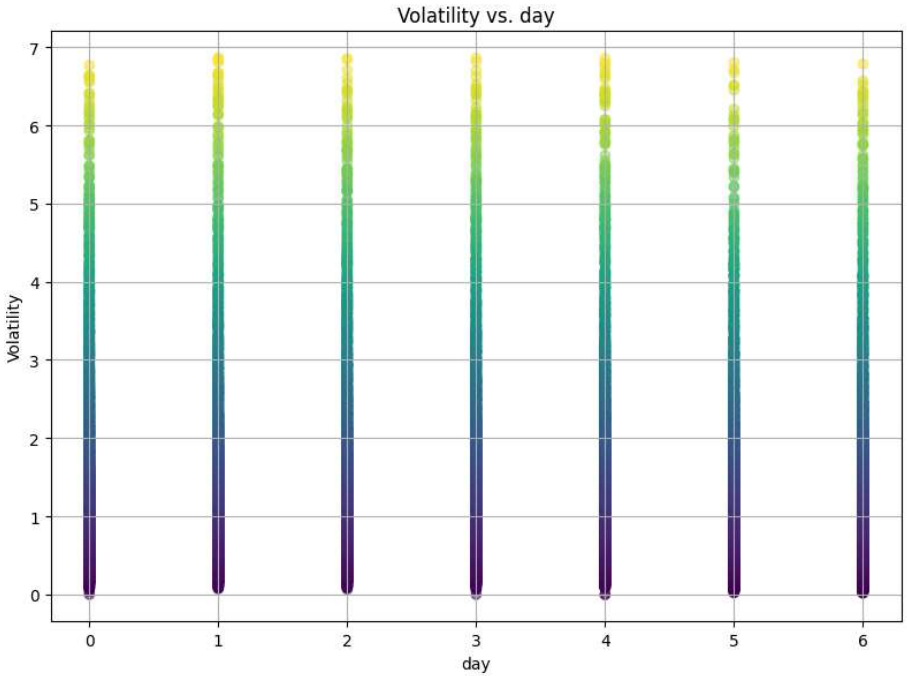
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 54116 entries, 317303 to 371418
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   hour        54116 non-null  int64  
 1   day         54116 non-null  int64  
 2   week        54116 non-null  int64  
 3   month       54116 non-null  int64  
 4   volatility  54116 non-null  float64
 5   tradecount  54116 non-null  int64  
 6   volume      54116 non-null  int64  
dtypes: float64(1), int64(6)
memory usage: 3.3 MB
```

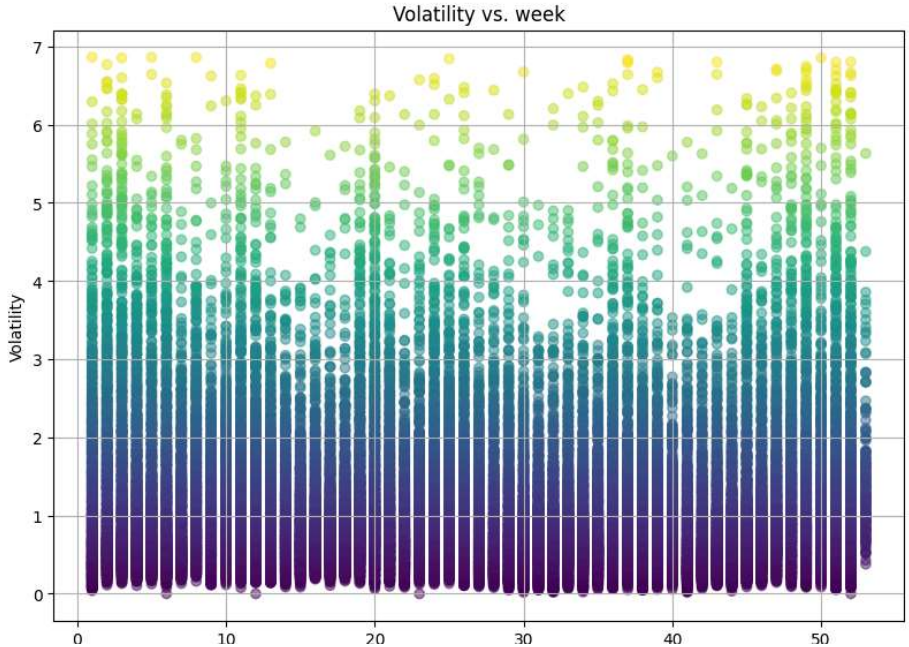
```
percentile_threshold = 99.5
percentile_value = np.percentile(btc_scat['volatility'], percentile_threshold)
btc_scat = btc_scat[btc_scat['volatility'] <= percentile_value]
```

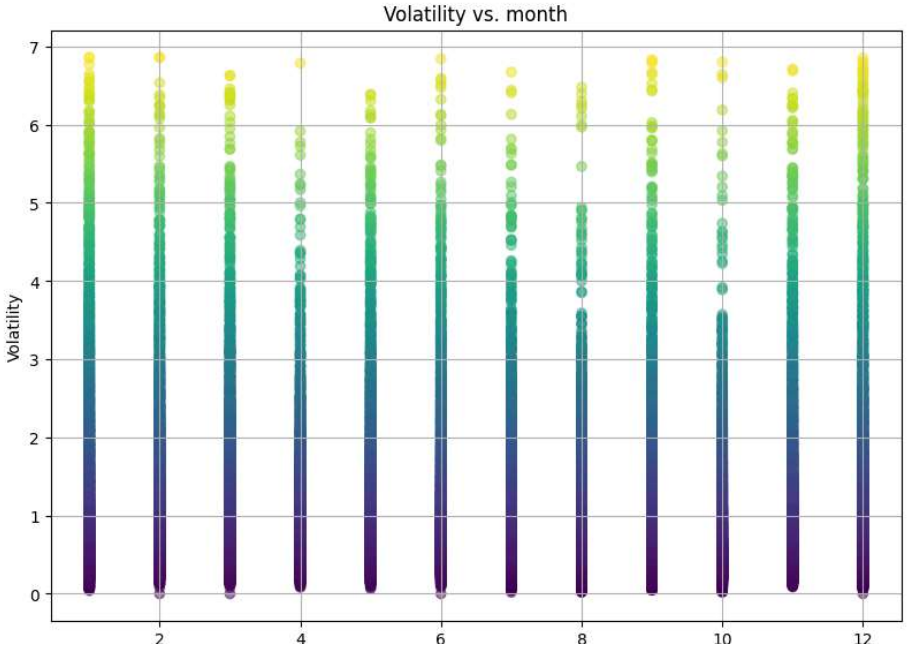
```
scatter_columns = ['hour', 'day', 'week', 'month']

num_rows = len(scatter_columns) // 2 + (len(scatter_columns) % 2 > 0)

fig, axes = plt.subplots(num_rows, 2, figsize=(8 * 2, 6 * num_rows))

for i, column in enumerate(scatter_columns):
    plt.subplot(num_rows, 2, i + 1)

    colors = btc_scat['volatility']

    plt.scatter(btc_scat[column], btc_scat['volatility'], c=colors, cmap=cm.viridis, alpha=0.5)
    plt.xlabel(column)
    plt.ylabel('Volatility')
    plt.title(f"Volatility vs. {column}")
    plt.legend().set_visible(False)
    plt.grid(True)

plt.tight_layout()
plt.show()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that artists whose label start with an u
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that artists whose label start with an u
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that artists whose label start with an u
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that artists whose label start with an u
```







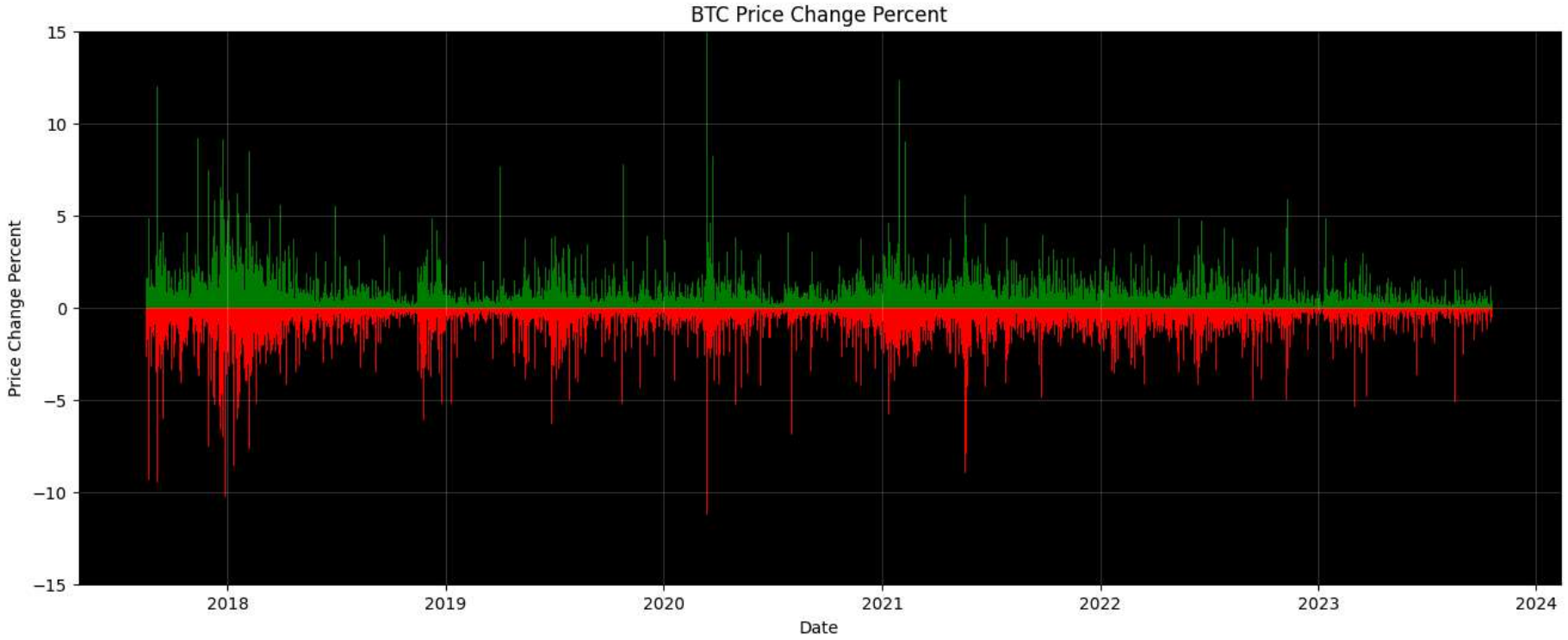



```
btc_df = df2[df2['token'] == 'BTC']
```

```
btc_df_2023_october = btc_df[(btc_df['date'].dt.year == 2023) & (btc_df['date'].dt.month == 10)]

dates = btc_df_2023_october['date']
open_prices = btc_df_2023_october['open']
high_prices = btc_df_2023_october['high']
low_prices = btc_df_2023_october['low']

plt.figure(figsize=(16, 8))
plt.gca().set_facecolor('black')

plt.plot(dates, high_prices, label='High', color='green')
plt.plot(dates, low_prices, label='Low', color='red')

plt.fill_between(dates, low_prices, high_prices, where=(high_prices > low_prices), alpha=0.3, color='white')

plt.xlabel('Date')
plt.ylabel('Price')
plt.title('BTC Price Range (High-Low) for October 2023')
plt.legend()

plt.grid(color='white', alpha=0.2)
plt.show()
```


BTC Price Range (High-Low) for October 2023
High
Low
30000
29500
29000
28500
28000
27500
27000
26500
Price
2023-10-01
2023-10-03
2023-10-05
2023-10-07
2023-10-09
2023-10-11
2023-10-13
2023-10-15
2023-10-17
2023-10-19
Date

```
price_change_percent = btc_df['price_change_percent']

plt.figure(figsize=(16, 6))
plt.gca().set_facecolor('black')
bars = plt.bar(btc_df['date'], price_change_percent, color=['g' if p >= 0 else 'r' for p in price_change_percent])
plt.ylim(-15, 15)

plt.xlabel('Date')
plt.ylabel('Price Change Percent')
plt.title('BTC Price Change Percent')

plt.grid(color='white', alpha=0.2)

plt.show()
```



Creating new columns EMAs (EMA 13 25 32 100 200) - For Moving Average

```
data_df['ema_13'] = data_df['close'].ewm(span=13).mean()
data_df['ema_25'] = data_df['close'].ewm(span=25).mean()
data_df['ema_32'] = data_df['close'].ewm(span=32).mean()
data_df['ema_100'] = data_df['close'].ewm(span=100).mean()
data_df['ema_200'] = data_df['close'].ewm(span=200).mean()
```

Creating Candle Volatility Against Price Close (high - low / close)

```
data_df['vol_close'] = (data_df['high'] - data_df['low']) / data_df['close']
```

Creating EMA for Candle Volatility/Price

```
# EMA for 2 hours (candles)
data_df['vol_close_ema_3'] = data_df['vol_close'].ewm(span=3, adjust=False).mean()

# EMA for 4 hours (candles)
data_df['vol_close_ema_6'] = data_df['vol_close'].ewm(span=6, adjust=False).mean()

# EMA for 8 hours ((candles)
data_df['vol_close_ema_12'] = data_df['vol_close'].ewm(span=12, adjust=False).mean()
```

```
data_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 27812 entries, 343607 to 371418
Data columns (total 20 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   date              27812 non-null  object
 1   symbol            27812 non-null  object
 2   open              27812 non-null  float64
 3   high              27812 non-null  float64
 4   low               27812 non-null  float64
 5   close             27812 non-null  float64
```

```
 6   volume usdt       27812 non-null  int64
 7   tradecount        27812 non-null  int64
 8   token             27812 non-null  object
 9   hour              27812 non-null  int64
 10  day               27812 non-null  int64
 11  ema_13            27812 non-null  float64
 12  ema_25            27812 non-null  float64
 13  ema_32            27812 non-null  float64
 14  ema_100           27812 non-null  float64
 15  ema_200           27812 non-null  float64
 16  vol_close         27812 non-null  float64
 17  vol_close_ema_3   27812 non-null  float64
 18  vol_close_ema_6   27812 non-null  float64
 19  vol_close_ema_12  27812 non-null  float64
dtypes: float64(13), int64(4), object(3)
memory usage: 4.5+ MB
```

```
data_df['date'] = pd.to_datetime(data_df['date'])
```

```
columns_to_drop = ['symbol', 'token']
data_df = data_df.drop(columns=columns_to_drop)
```

```
#data_df = data_df.astype(float)
#data_df.set_index('date', inplace=True)
```

```
data_df.to_csv('data_featured.csv', index=False)
```

```
train_df = data_df.copy(deep=True)
train_df.reset_index(drop=True, inplace=True)
```

```
train_df.head()
```

	date	open	high	low	close	volume usdt	tradecount	hour	day	ema_13	ema_25	ema
0	2020-08-17 04:00:00	11844.72	11858.91	11802.35	11809.38	17196539	28800	4	1	11809.380000	11809.380000	11809.380
1	2020-08-17 05:00:00	11809.39	11836.90	11790.00	11800.01	19958274	28571	5	1	11804.334615	11804.507600	11804.548

```
#df_test = data_df[10000:]
#df_test = df_test.reset_index(drop=True)
```

DRL Environment Architecture

Environment Controls:

- **data**: A DataFrame containing the financial data. (passed when called the function 'env' CryptoTradingEnv(data_df))
- **take_profit_range**: A tuple representing the range for take profit levels.
- **stop_loss_range**: A tuple representing the range for stop loss levels.

- **max_stop_loss**: The maximum allowable stop loss.
- **position_size**: The maximum allowable position size.
- **initial_balance**: The initial capital.
- **balance**: Current balance.
- **position**: The current position (0 for no position, 1 for long position).
- **position_open**: The price at which the position was opened.
- **num_trades**: The number of trades.
- **profit_loss**: The cumulative profit/loss.
- **position_size**: The calculated position size as a percentage of the current balance (controlled within the code).

Obversations:

- open, high, low, close, volume usdt, hour, day, ema_13, ema_25, ema_32, ema_100, ema_200, vol_close_ema_3, vol_close_ema_6, vol_close_ema_12

Initial Reward Calculation:

- When an action is taken, the reward variable is initialized to 0.
- The variable trade_outcome is used to calculate the profit or loss associated with the most recent trade. It is initialized to 0.

Opening and Closing Trades:

- If the agent takes the action to open a position (action code 1), and it is not already in a trade (self.position == 0), it calculates the position_size based on the balance or maximum position size, and records the opening price.
- If the agent takes the action to close a position (action code 2), and it is currently in a trade (self.position == 1), it calculates the trade outcome as the difference between the opening price and the current market price.

- The profit_loss is updated with the trade outcome, representing the cumulative profit or loss across all trades.
- The balance is adjusted based on the trade outcome.

Take Profit and Stop Loss:

- The code checks whether the trade outcome is greater than 0 (a winning trade) and whether it reaches the take profit level defined by self.take_profit_range.
- If the trade reaches the take profit level (>0), a reward of 0.5 is assigned to indicate a successful trade.
- If the trade is closed within the take profit range, an extra reward of 1 is assigned to indicate a successful trade.
- If the trade outcome is less than 0 (a losing trade), the code checks for two conditions:
- If the trade outcome is equal or exceeds the maximum allowable stop loss (self.max_stop_loss * self.balance), a malus reward of -0.7 is assigned, indicating that the trade reached the maximum stop loss.
- If the trade outcome falls within the stop loss range defined by self.stop_loss_range, a reward of -1 is assigned, indicating that the trade reached the stop loss.

Trade Outcome Within Range:

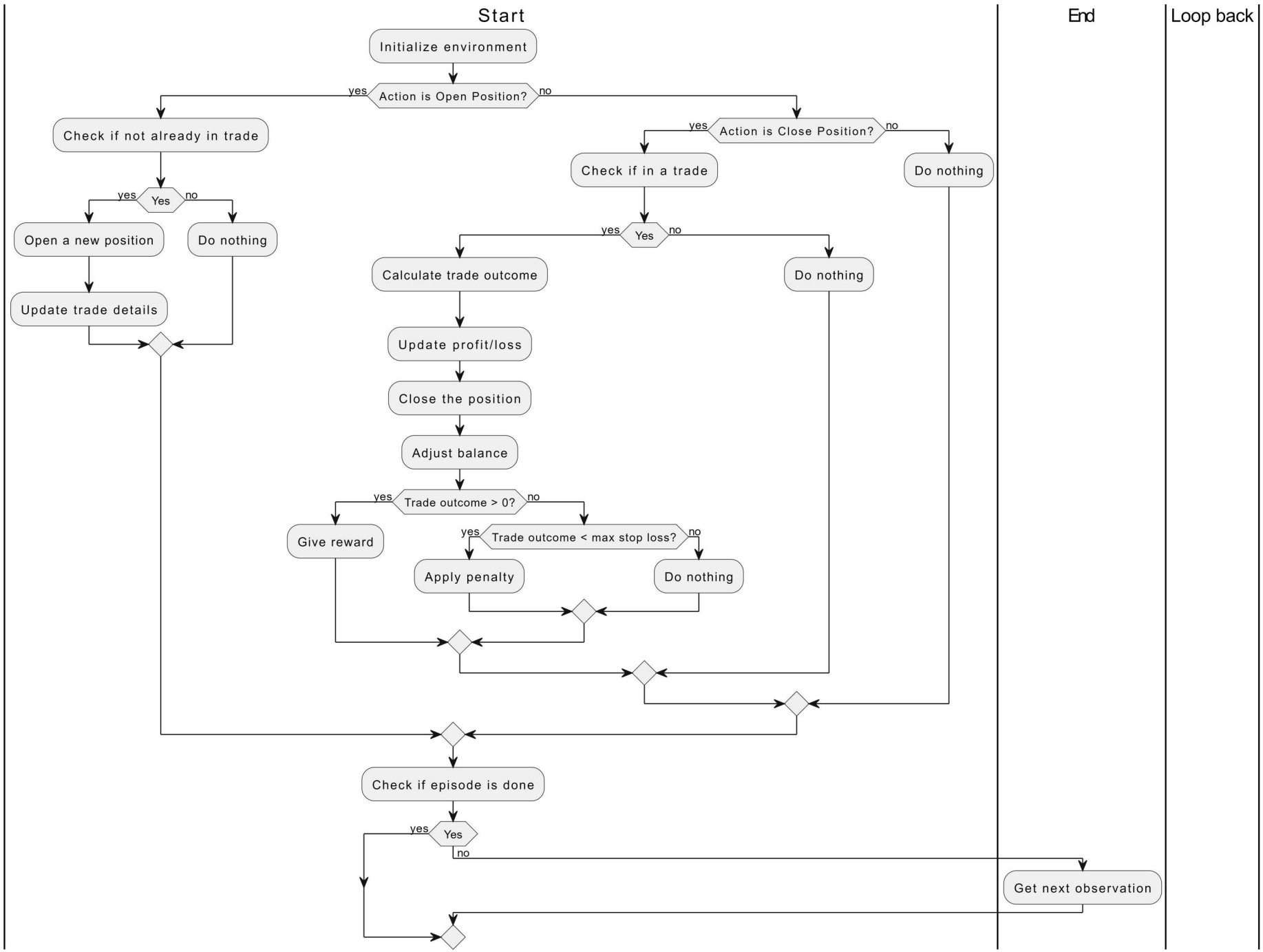
- If trade_within_range is True, it means that the trade outcome fell within the specified take profit range. In this case, self.winning_trade_within_range is incremented to keep track of such trades.

Cumulative Returns and Episode Tracking:

- The reward is also added to the episode_returns list to track rewards for the current episode.
- The cumulative_returns is updated with the reward to keep track of cumulative returns.

Completing an Episode:

- If the current step reaches the end of the trading data (self.current_step >= self.n_steps), the done flag is set to True, indicating that the episode has ended.


Start
End
Loop back
Initialize environment
Action is Open Position?
yes
no
Check if not already in trade
Yes
yes
no
Open a new position
Do nothing
Update trade details
Action is Close Position?
yes
no
Check if in a trade
Do nothing
Yes
yes
no
Calculate trade outcome
Do nothing
Update profit/loss
Close the position
Adjust balance
Trade outcome > 0?
yes
no
Give reward
Trade outcome < max stop loss?
yes
no
Apply penalty
Do nothing
Check if episode is done
Yes
yes
no
Get next observation

DRL Environment v1

It simulates cryptocurrency trading

```
class CryptoTradingEnv(gym.Env):
    def __init__(self, data, take_profit_position_range=(0.10, 0.80), stop_loss_position_range=(0.00, 0.15), max_stop_loss
        super(CryptoTradingEnv, self).__init__()

        self.data = data
        self.n_steps = len(data)
        self.current_step = 0
        self.initial_balance = 10000
        self.balance = self.initial_balance
        self.position = 0
        self.position_open = 0 # Price at which the position was opened
        self.num_trades = 0  # Number of trades
        self.profit_loss = 0  # PnL Profit/Loss
        self.max_stop_loss_position = max_stop_loss_position

        self.take_profit_position_range = take_profit_position_range
        self.stop_loss_position_range = stop_loss_position_range

        # 0=Hold, 1=Open Position, 2=Close Position
        self.action_space = gym.spaces.Discrete(3)

        # Observations - open, high, low, and close prices, trading volume, time indicators (hour and day), and various mo
        n_features = 15
        self.observation_space = gym.spaces.Box(low=0, high=1, shape=(n_features,))

        self.episode_returns = []
        self.cumulative_returns = 0
        self.winning_trades = 0
        self.losing_trades = 0
        self.hourly_returns = []

        self.overall_rewards = 0


    def step(self, action): # This function is responsible for executing an action in the environment and returning the ne
        self.current_step += 1
        done = False

        reward = 0
```

```
trade_outcome = 0
trade_within_range = False

if action == 1:  # Open Position - checks if there is no existing position and opens a new position with a fixed p
    if self.position == 0:  # Only open a position if not already in a trade
        position_size = 0.05 * self.balance
        self.position_open = self.data.loc[self.current_step, 'open']
        self.position = 1
        self.num_trades += 1
        print(f"Opened trade at step {self.current_step} with position size: {position_size:.2f}")

elif action == 2:  # Close Position - checks if there is an existing position, calculates the trade outcome, updat
    if self.position == 1:  # Only close a position if currently in a trade
        position_close = self.data.loc[self.current_step, 'open']
        tr ade_outcome = position_close - self.position_open
        self.profit_loss += trade_outcome
        self.position = 0
        self.position_open = 0
        print(f"Closed trade at step {self.current_step}")
        print(f"-----------------------------> Trade outcome: {trade_outcome}")

        self.balance += trade_outcome

        # Check for take profit and stop loss
        if trade_outcome > 0:
            if trade_outcome >= self.take_profit_position_range[0] * self.position:
                reward = 1  # Trade reached take profit
                self.winning_trades += 1  # +1 winning trades count
                if trade_outcome <= self.take_profit_position_range[1] * self.position and trade_outcome >= self.t
                   # self.trade_within_range = True
                    print("Trade reached take profit")
        elif trade_outcome < 0:
            if abs(trade_outcome) >= self.max_stop_loss_position * self.position:
                reward = -1.7  # Trade reached the maximum stop loss
                self.losing_trades += 1  # +1 losing trades count
                print("Trade reached the maximum stop loss")
            elif abs(trade_outcome) >= self.stop_loss_position_range[0] * self.position:
                reward = -1  # Trade reached the stop loss
                self.losing_trades += 1  # +1 losing trades count
```

```
    print("Trade reached stop loss")
else:
    # No
```