```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn import metrics

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

```
df = pd.read_csv('/content/coin_Bitcoin.csv')
df.head()
```

	SNo	Name	Symbol	Date	High	Low	Open	Close	Volume	
0	1	Bitcoin	BTC	2013-04-29 23:59:59	147.488007	134.000000	134.444000	144.539993	0.0	1.
1	2	Bitcoin	BTC	2013-04-30 23:59:59	146.929993	134.050003	144.000000	139.000000	0.0	1.

```
df.shape
```

```
(2991, 10)
```

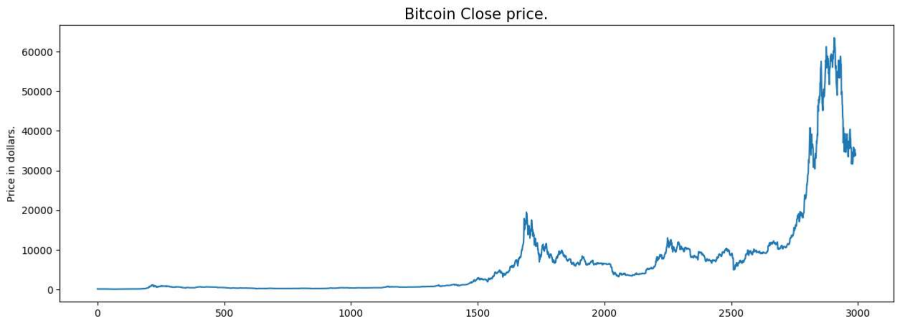
```
df.describe()
```

	SNo	High	Low	Open	Close	Volume
count	2991.000000	2991.000000	2991.000000	2991.000000	2991.000000	2.991000e+03
mean	1496.000000	6893.326038	6486.009539	6700.146240	6711.290443	1.090633e+10
std	863.571653	11642.832456	10869.032130	11288.043736	11298.141921	1.888895e+10
min	1.000000	74.561096	65.526001	68.504997	68.431000	0.000000e+00
25%	748.500000	436.179001	422.879486	430.445496	430.569489	3.036725e+07
50%	1496.000000	2387.610107	2178.500000	2269.889893	2286.409912	9.460360e+08
75%	2243.500000	8733.926948	8289.800459	8569.656494	8576.238715	1.592015e+10
max	2991.000000	64863.098908	62208.964366	63523.754869	63503.457930	3.509679e+11

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2991 entries, 0 to 2990
Data columns (total 10 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   SNo        2991 non-null   int64  
 1   Name       2991 non-null   object 
 2   Symbol     2991 non-null   object 
 3   Date       2991 non-null   object 
 4   High       2991 non-null   float64
 5   Low        2991 non-null   float64
 6   Open       2991 non-null   float64
 7   Close      2991 non-null   float64
 8   Volume     2991 non-null   float64
 9   Marketcap  2991 non-null   float64
dtypes: float64(6), int64(1), object(3)
memory usage: 233.8+ KB
```

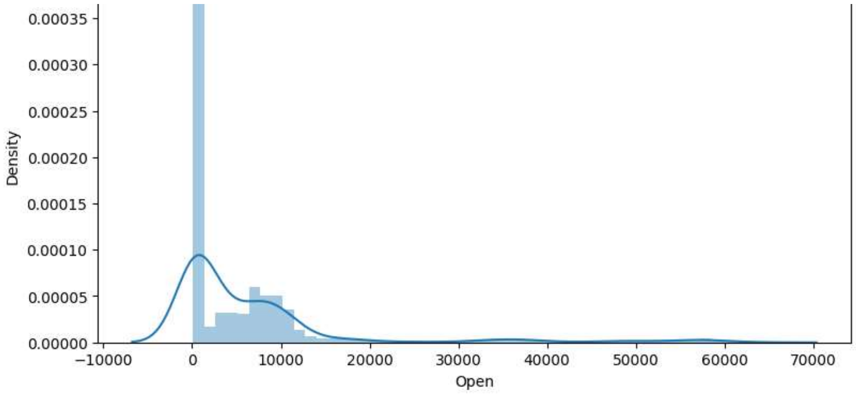
```
plt.figure(figsize=(15, 5))
plt.plot(df['Close'])
plt.title('Bitcoin Close price.', fontsize=15)
plt.ylabel('Price in dollars.')
plt.show()
```

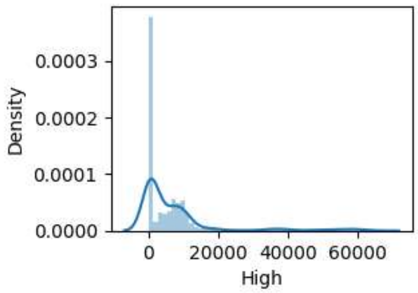


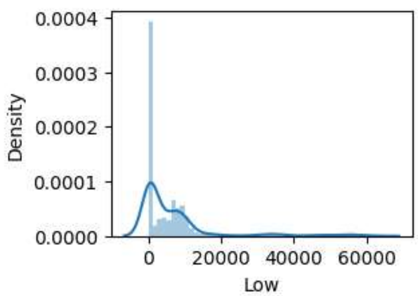

```
df.isnull().sum()
```

```
SNo          0
Name         0
Symbol       0
Date         0
High         0
Low          0
Open         0
Close        0
Volume       0
Marketcap    0
dtype: int64
```

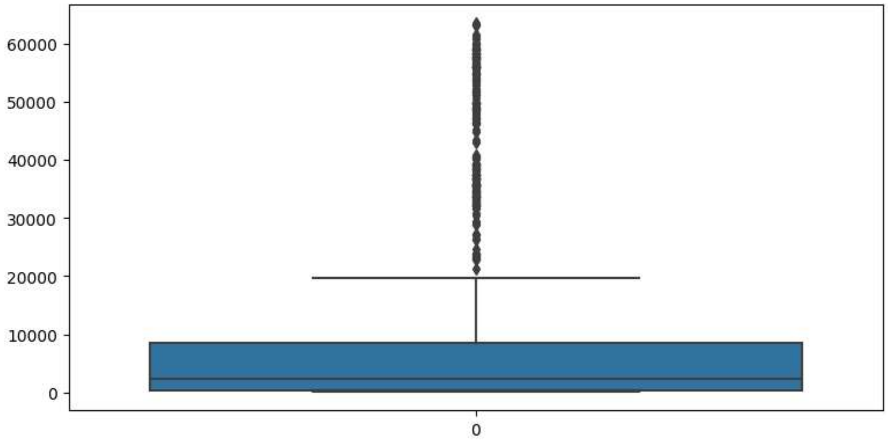
```
features = ['Open', 'High', 'Low', 'Close']

plt.subplots(figsize=(20,10))
for i, col in enumerate(features):
  plt.subplot(2,2,i+1)
  sb.distplot(df[col])
  plt.show()
```

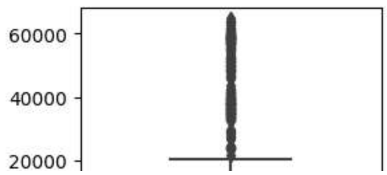






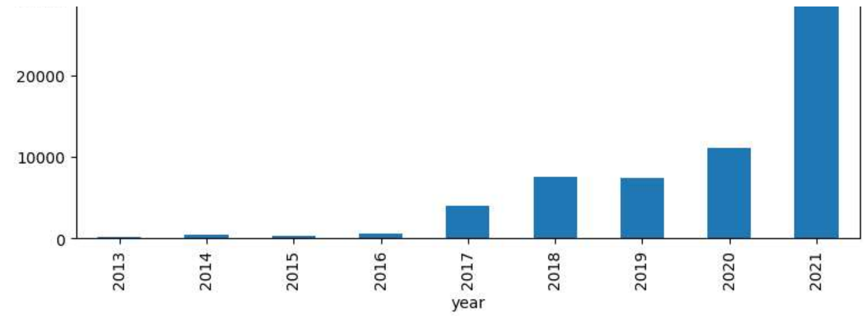

```
plt.subplots(figsize=(20,10))
for i, col in enumerate(features):
  plt.subplot(2,2,i+1)
  sb.boxplot(df[col])
  plt.show()
```

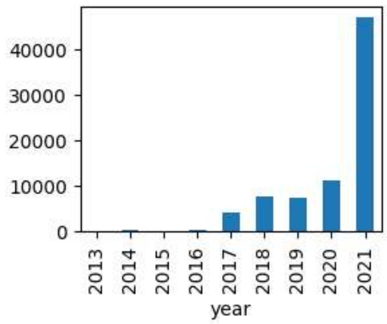





```
df['Date'] = pd.to_datetime(df['Date'], format='%Y-%m-%d %H:%M:%S')
df['year'] = df['Date'].dt.year
df['month'] = df['Date'].dt.month
df['day'] = df['Date'].dt.day
```

```
data_grouped = df.groupby('year').mean()
plt.subplots(figsize=(20,10))
for i, col in enumerate(['Open', 'High', 'Low', 'Close']):
  plt.subplot(2,2,i+1)
  data_grouped[col].plot.bar()
  plt.show()
```



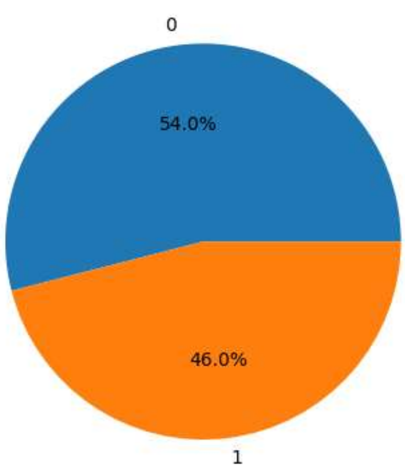



```
df['is_quarter_end'] = np.where(df['month']%3==0,1,0)
df.head()
```

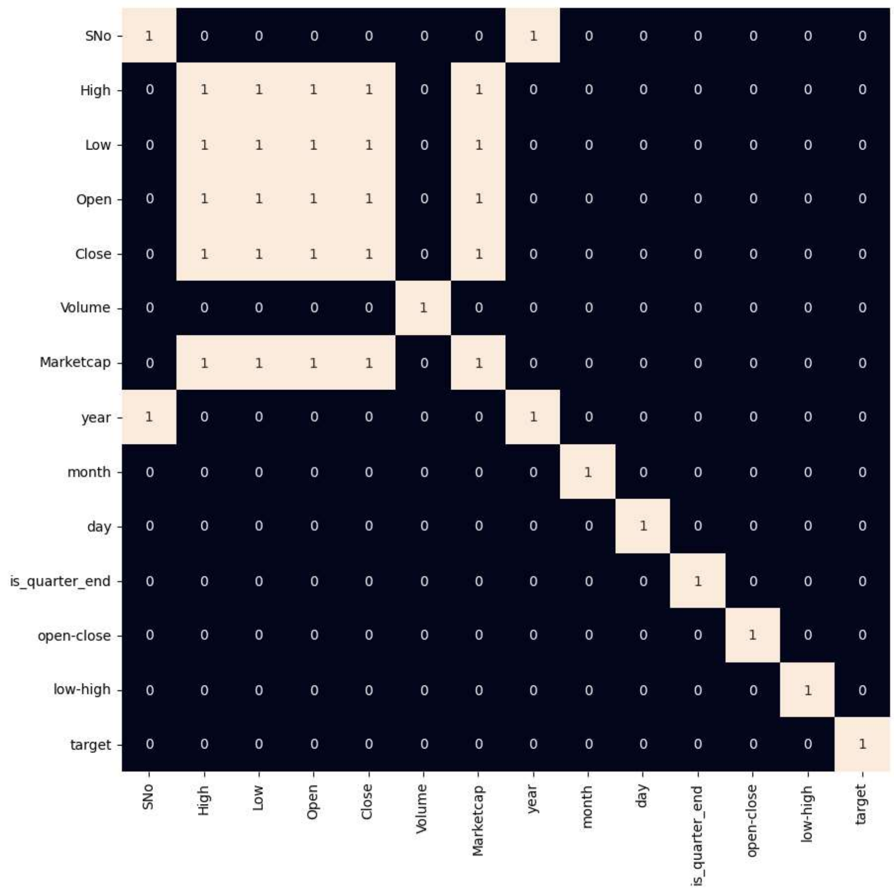
	SNo	Name	Symbol	Date	High	Low	Open	Close	Volume	
0	1	Bitcoin	BTC	2013-04-29 23:59:59	147.488007	134.000000	134.444000	144.539993	0.0	1.
1	2	Bitcoin	BTC	2013-04-30 23:59:59	146.929993	134.050003	144.000000	139.000000	0.0	1.

```
df['open-close'] = df['Open'] - df['Close']
df['low-high'] = df['Low'] - df['High']
df['target'] = np.where(df['Close'].shift(-1) > df['Close'], 1, 0)
```

```
plt.pie(df['target'].value_counts().values,
    labels=[0, 1], autopct='%1.1f%%')
plt.show()
```



```
plt.figure(figsize=(10, 10))

# As our concern is with the highly
# correlated features only so, we will visualize
# our heatmap as per that criteria only.
sb.heatmap(df.corr() > 0.9, annot=True, cbar=False)
plt.show()
```




```
features = df[['open-close', 'low-high', 'is_quarter_end']]
target = df['target']

scaler = StandardScaler()
features = scaler.fit_transform(features)

X_train, X_valid, Y_train, Y_valid = train_test_split(
    features, target, test_size=0.1, random_state=2022)
print(X_train.shape, X_valid.shape)
```

```
(2691, 3) (300, 3)
```

```
models = [LogisticRegression(), SVC(kernel='poly', probability=True), XGBClassifier()]

for i in range(3):
  models[i].fit(X_train, Y_train)

  print(f'{models[i]} : ')
```

```
  print('Training Accuracy : ', metrics.roc_auc_score(Y_train, models[i].predict_proba(X_train)[:,1]))
  print('Validation Accuracy : ', metrics.roc_auc_score(Y_valid, models[i].predict_proba(X_valid)[:,1]))
  print()
```

```
LogisticRegression() :
Training Accuracy :  0.5348847672849073
Validation Accuracy :  0.48994986223406656

SVC(kernel='poly', probability=True) :
Training Accuracy :  0.5231565265210218
Validation Accuracy :  0.5378291702425584

XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...) :
Training Accuracy :  0.9101990257017003
Validation Accuracy :  0.4579475134378247
```

```
!pip install --upgrade scikit-learn
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.3.2)
Requirement already satisfied: numpy<2.0,>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.23.5)
Requirement already satisfied: scipy>=1.5.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.11.3)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.2.0)
```

```
"""Conclusion:
We can observe that the accuracy achieved by the state-of-the-art ML model is no better than simply guessing with a probability of 50%. Possil
```

```
'Conclusion:\nWe can observe that the accuracy achieved by the state-of-the-art ML model is no better than simply guessing with a proba
bility of 50%. Possible reasons for this may be the lack of data or using a very simple model to perform such a complex task as Stock M
arket prediction.'
```

```
metrics.plot_confusion_matrix(models[0], X_valid, Y_valid)
plt.show()
```