In [1]:
```
# for importing data set i am using yfinance library

import yfinance as yf
```

In [2]:
```
bitcoin=yf.Ticker("BTC-USD") #Ticker is symbol for stocks
```

In [3]:
```
bitcoin=bitcoin.history(period="max")
```

In [4]:
```
bitcoin
```

Out[4]:

	Open	High	Low	Close	Volume	Dividends	Sto Spl
Date							
2014-09-17 00:00:00+00:00	465.864014	468.174011	452.421997	457.334015	21056800	0.0	(
2014-09-18 00:00:00+00:00	456.859985	456.859985	413.104004	424.440002	34483200	0.0	(
2014-09-19 00:00:00+00:00	424.102997	427.834991	384.532013	394.795990	37919700	0.0	(
2014-09-20 00:00:00+00:00	394.673004	423.295990	389.882996	408.903992	36863600	0.0	(
2014-09-21 00:00:00+00:00	408.084991	412.425995	393.181000	398.821014	26580100	0.0	(
...	...	...	...	...	...	...	
2023-07-12 00:00:00+00:00	30622.246094	30959.964844	30228.835938	30391.646484	14805659717	0.0	(
2023-07-13 00:00:00+00:00	30387.488281	31814.515625	30268.351562	31476.048828	23686079548	0.0	(
2023-07-14 00:00:00+00:00	31474.720703	31582.253906	29966.386719	30334.068359	20917902660	0.0	(
2023-07-15 00:00:00+00:00	30331.783203	30407.781250	30263.462891	30295.806641	8011667756	0.0	(
2023-07-16 00:00:00+00:00	30304.167969	30336.707031	30122.935547	30208.246094	8101786624	0.0	(

3225 rows × 7 columns

In [5]:
```
bitcoin.index
```

Out[5]:

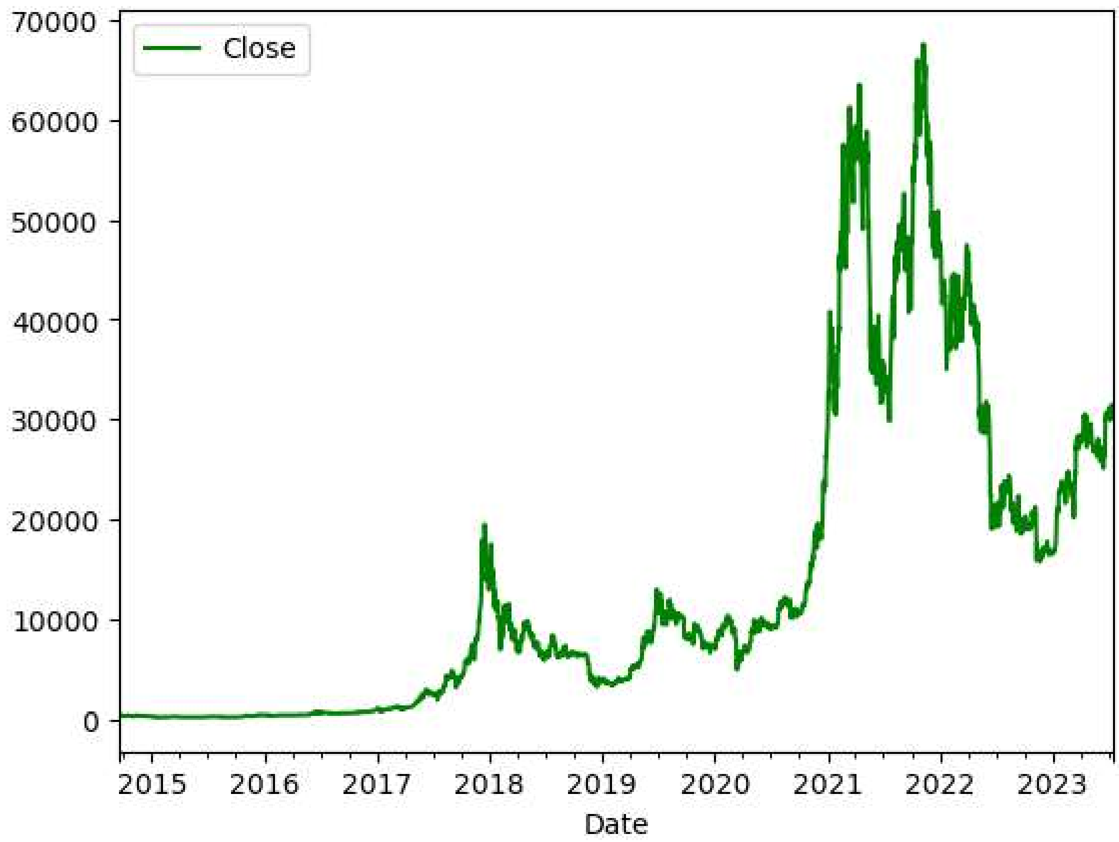
```
DatetimeIndex(['2014-09-17 00:00:00+00:00', '2014-09-18 00:00:00+00:00',
               '2014-09-19 00:00:00+00:00', '2014-09-20 00:00:00+00:00',
               '2014-09-21 00:00:00+00:00', '2014-09-22 00:00:00+00:00',
               '2014-09-23 00:00:00+00:00', '2014-09-24 00:00:00+00:00',
               '2014-09-25 00:00:00+00:00', '2014-09-26 00:00:00+00:00',
               ...
               '2023-07-07 00:00:00+00:00', '2023-07-08 00:00:00+00:00',
               '2023-07-09 00:00:00+00:00', '2023-07-10 00:00:00+00:00',
               '2023-07-11 00:00:00+00:00', '2023-07-12 00:00:00+00:00',
               '2023-07-13 00:00:00+00:00', '2023-07-14 00:00:00+00:00',
               '2023-07-15 00:00:00+00:00', '2023-07-16 00:00:00+00:00'],
              dtype='datetime64[ns, UTC]', name='Date', length=3225, freq=None)
```

In [6]:

```
bitcoin.plot.line(y="Close",use_index=True,color={"Close": "Green"})
```

Out[6]:

```
<AxesSubplot:xlabel='Date'>
```



In [7]:

```
#deleting unwanted columns
del bitcoin["Dividends"]
del bitcoin["Stock Splits"]
```

In [8]:

```
# adding nextdays closing value to make it easy for solving

bitcoin["Tomorrow"]=bitcoin["Close"].shift(-1)  #Here we shifted one value up
```

In [9]:

```
bitcoin
```

Out[9]:

	Open	High	Low	Close	Volume	Tomorrow
Date						
2014-09-17 00:00:00+00:00	465.864014	468.174011	452.421997	457.334015	21056800	424.440002
2014-09-18 00:00:00+00:00	456.859985	456.859985	413.104004	424.440002	34483200	394.795990
2014-09-19 00:00:00+00:00	424.102997	427.834991	384.532013	394.795990	37919700	408.903992
2014-09-20 00:00:00+00:00	394.673004	423.295990	389.882996	408.903992	36863600	398.821014
2014-09-21 00:00:00+00:00	408.084991	412.425995	393.181000	398.821014	26580100	402.152008
...	...	...	...	...	...	...
2023-07-12 00:00:00+00:00	30622.246094	30959.964844	30228.835938	30391.646484	14805659717	31476.048828
2023-07-13 00:00:00+00:00	30387.488281	31814.515625	30268.351562	31476.048828	23686079548	30334.068359
2023-07-14 00:00:00+00:00	31474.720703	31582.253906	29966.386719	30334.068359	20917902660	30295.806641
2023-07-15 00:00:00+00:00	30331.783203	30407.781250	30263.462891	30295.806641	8011667756	30208.246094
2023-07-16 00:00:00+00:00	30304.167969	30336.707031	30122.935547	30208.246094	8101786624	NaN

3225 rows × 6 columns

In [10]:

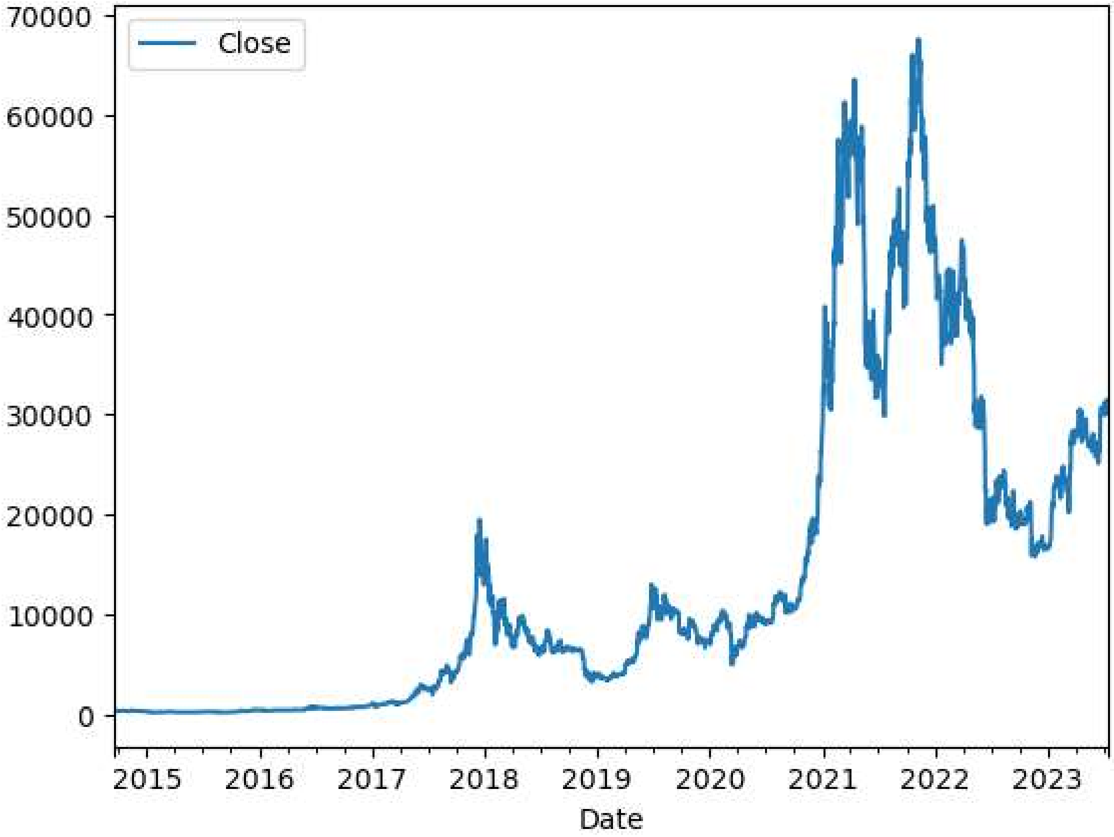
```
#creating one more column namely target here if our tomorrows value is grater
#than closing value it will give output as 1 otherwise 0
# for this we will use astype(int) to get valuse

bitcoin["Target"]=(bitcoin["Tomorrow"]>bitcoin["Close"]).astype(int)
```

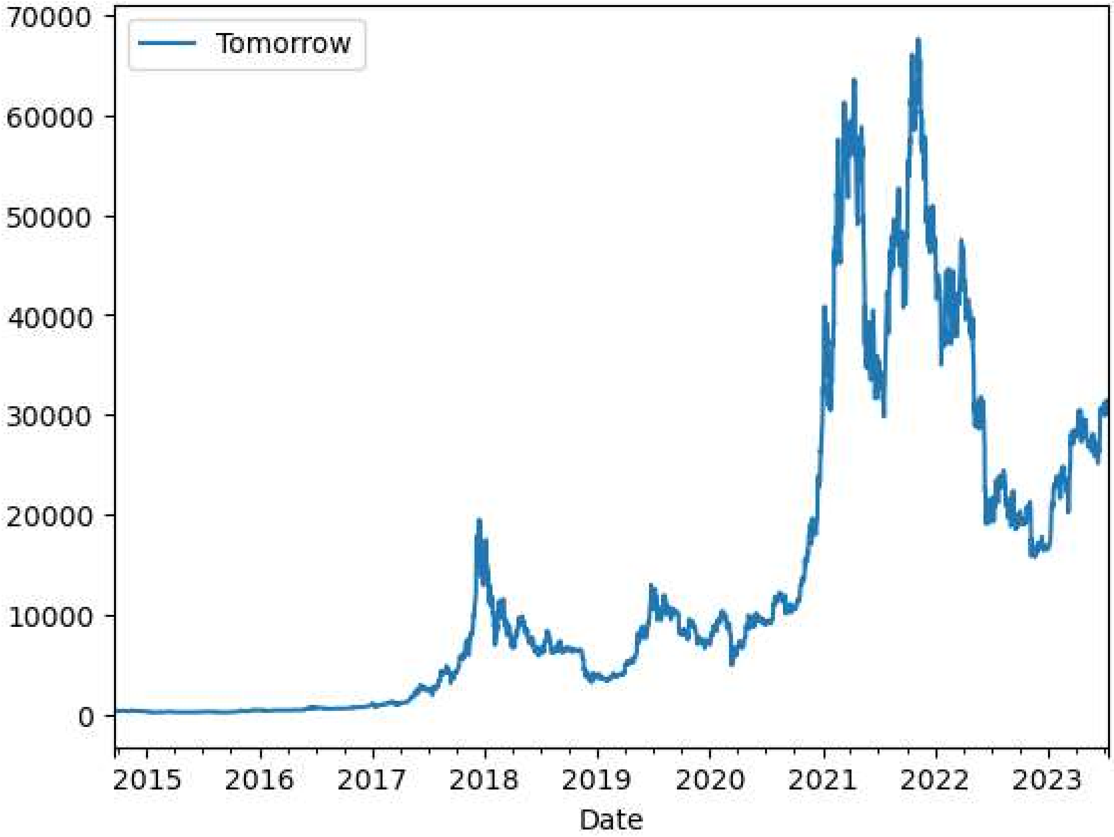
In [11]:

```
bitcoin.plot.line(y="Close",use_index=True)
bitcoin.plot.line(y="Tomorrow",use_index=True)
```

Out[11]:

```
<AxesSubplot:xlabel='Date'>
```





using random forest classifier

In [12]:

```
from sklearn.ensemble import RandomForestClassifier

model=RandomForestClassifier(n_estimators=800,min_samples_split=2,random_state=1)

train=bitcoin.iloc[:-100]
test= bitcoin.iloc[-100:]

predictors = ["Close","Volume","High","Low"]
model.fit(train[predictors],train["Target"])
```

Out[12]:

```
RandomForestClassifier(n_estimators=800, random_state=1)
```

In [13]:

```
from sklearn.metrics import precision_score

preds = model.predict(test[predictors])
```

In [14]:

```
preds
```

Out[14]:

```
array([1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0])
```

In [15]:

```
import pandas as pd

preds = pd.Series(preds,index=test.index)
```

In [16]:

```
precision_score(test["Target"],preds)
```

Out[16]:

```
0.5161290322580645
```

In [17]:

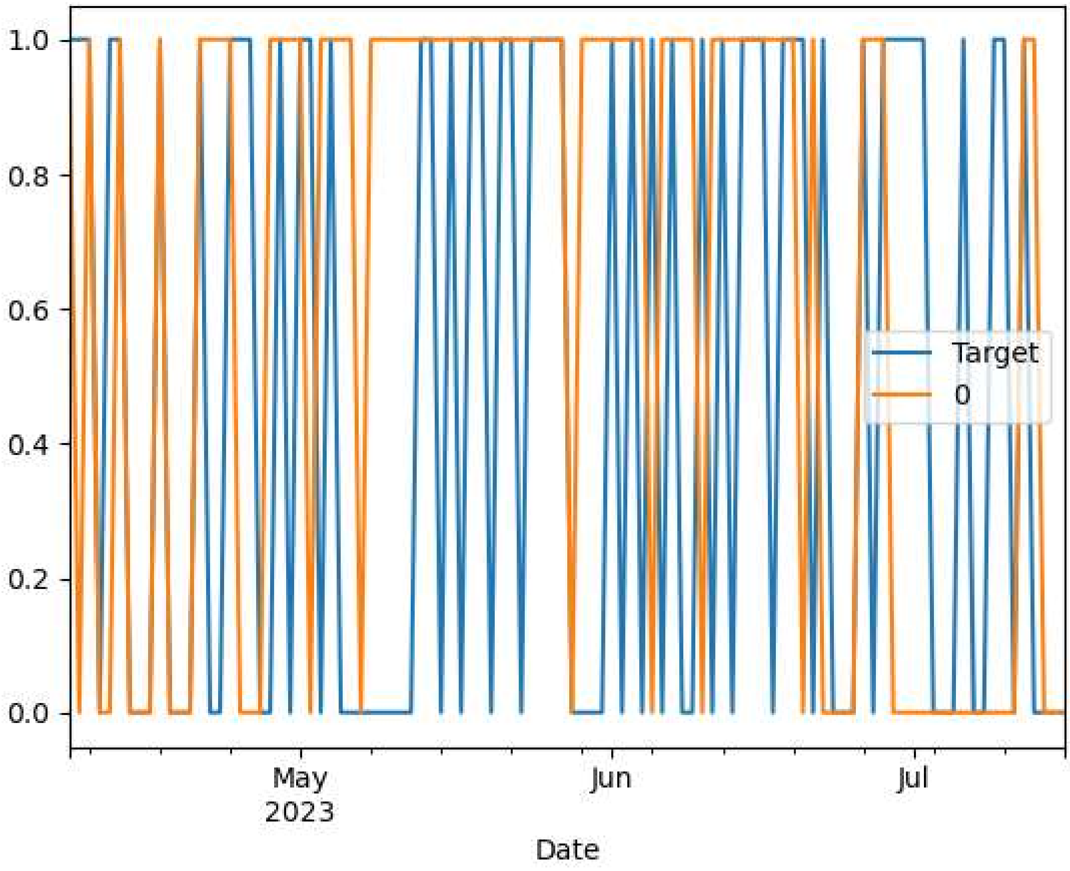
```
combined=pd.concat([test["Target"],preds],axis=1)
```

In [19]:

```
combined.plot()
```

Out[19]:

```
<AxesSubplot:xlabel='Date'>
```



In [20]:
```
# backtesting


def predict(train,test,predictors,model):
    model.fit(train[predictors],train["Target"])
    preds=model.predict(test[predictors])
    preds=pd.Series(preds,index=test.index,name="Predictions")
    combined=pd.concat([test["Target"],preds],axis=1)
    return combined
```

In [21]:
```
def backtest(data,model,predictors,start=2500, step=250):
    all_prediction=[]

    for i in range(start,data.shape[0],step):
        train=data.iloc[0:i].copy()
        test=data.iloc[i:(i+step)].copy()
        predictions=predict(train,test,predictors,model)
        all_prediction.append(predictions)
    return pd.concat(all_prediction)
```

In [22]:
```
predictions=backtest(bitcoin,model,predictors)
```

In [23]:
```
predictions["Predictions"].value_counts()
```

Out[23]:
```
1    423
0    302
Name: Predictions, dtype: int64
```

In [24]:
```
precision_score(predictions["Target"],predictions["Predictions"])
```

```
Out[24]: 0.46808510638297873
```

```
In [25]: predictions["Target"].value_counts()/predictions.shape[0]
```

```
Out[25]: 0    0.513103
         1    0.486897
         Name: Target, dtype: float64
```

```
In [26]: precision_score(test["Target"],preds)
```

```
Out[26]: 0.5161290322580645
```

```
In [ ]:
```