Case Study on Tesla Stock Price

In [40]:
```
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')
```

The dataset we will use here to perform the analysis and build a predictive model is Tesla Stock Price data. We will use OHLC('Open', 'High', 'Low', 'Close') data from 1st January 2010 to 31st December 2017 which is for 8 years for the Tesla stocks.

In [13]:
```
df = pd.read_csv('Tesla.csv')
```

Exploratory Data Analysis

EDA is an approach to analyzing the data using visual techniques. It is used to discover trends, and patterns, or to check assumptions with the help of statistical summaries and graphical representations.

While performing the EDA of the Tesla Stock Price data we will analyze how prices of the stock have moved over the period of time and how the end of the quarters affects the prices of the stock.

In [7]:
```
df.shape
```

Out[7]:
```
(1692, 7)
```

From this, we got to know that there are 1692 rows of data available and for each row, we have 7 different features or columns.

In [8]:
```
df.describe()
```

Out[8]:

	Open	High	Low	Close	Volume	Adj Close
count	1692.000000	1692.000000	1692.000000	1692.000000	1.692000e+03	1692.000000
mean	132.441572	134.769698	129.996223	132.428658	4.270741e+06	132.428658
std	94.309923	95.694914	92.855227	94.313187	4.295971e+06	94.313187
min	16.139999	16.629999	14.980000	15.800000	1.185000e+05	15.800000
25%	30.000000	30.650000	29.215000	29.884999	1.194350e+06	29.884999
50%	156.334999	162.370002	153.150002	158.160004	3.180700e+06	158.160004
75%	220.557495	224.099999	217.119999	220.022503	5.662100e+06	220.022503
max	287.670013	291.420013	280.399994	286.040009	3.716390e+07	286.040009

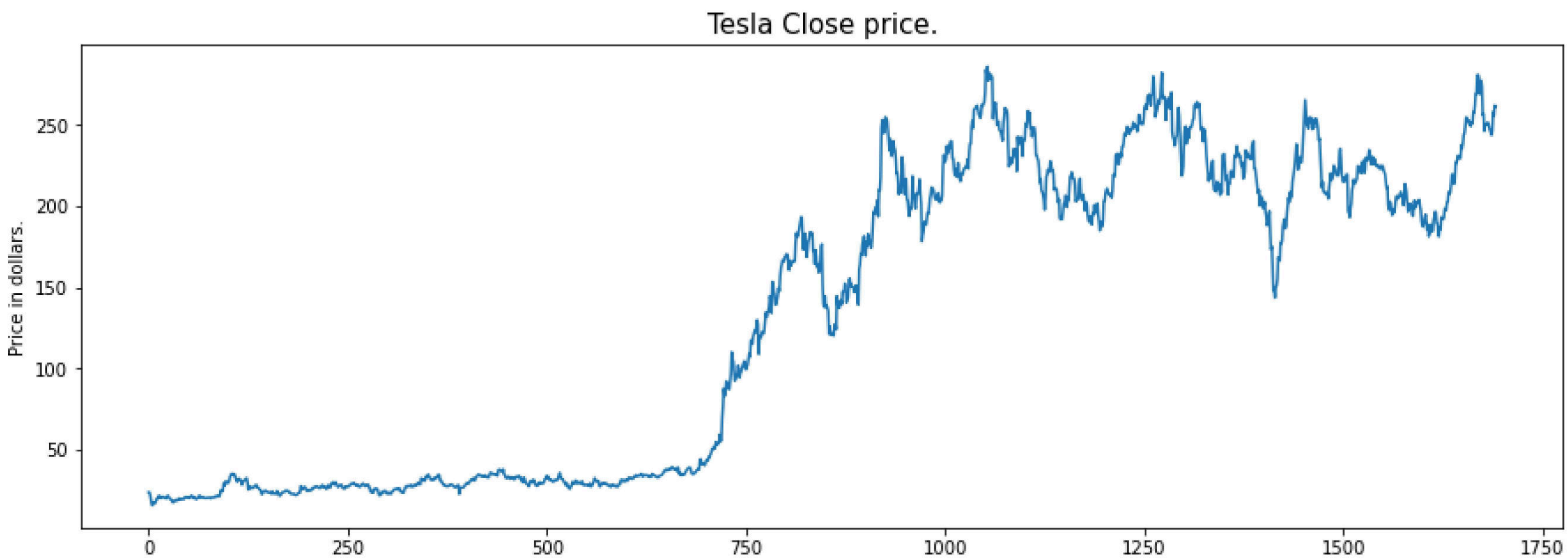
In [9]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1692 entries, 0 to 1691
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Date       1692 non-null   object 
 1   Open       1692 non-null   float64
 2   High       1692 non-null   float64
 3   Low        1692 non-null   float64
 4   Close      1692 non-null   float64
 5   Volume     1692 non-null   int64  
 6   Adj Close  1692 non-null   float64
dtypes: float64(5), int64(1), object(1)
memory usage: 92.7+ KB
```

In [10]:

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



The prices of the tesla stocks are showing an upward trend as depicted by the plot of the closing price of the stocks.

In [12]:
```
df.head()
```

Out[12]:

	Date	Open	High	Low	Close	Volume	Adj Close
0	6/29/2010	19.000000	25.00	17.540001	23.889999	18766300	23.889999
1	6/30/2010	25.790001	30.42	23.299999	23.830000	17187100	23.830000
2	7/1/2010	25.000000	25.92	20.270000	21.959999	8218800	21.959999
3	7/2/2010	23.000000	23.10	18.709999	19.200001	5139800	19.200001
4	7/6/2010	20.000000	20.00	15.830000	16.110001	6866900	16.110001

If we observe carefully we can see that the data in the 'Close' column and that available in the 'Adj Close' column is the same let's check whether this is the case with each row or not.

In [14]:
```
df[df['Close'] == df['Adj Close']].shape
```

Out[14]:
```
(1692, 7)
```

From here we can conclude that all the rows of columns 'Close' and 'Adj Close' have the same data. So, having redundant data in the dataset is not going to help so, we'll drop this column before further analysis.

In [15]:
```
df = df.drop(['Adj Close'], axis=1)
```

Now let's draw the distribution plot for the continuous features given in the dataset.

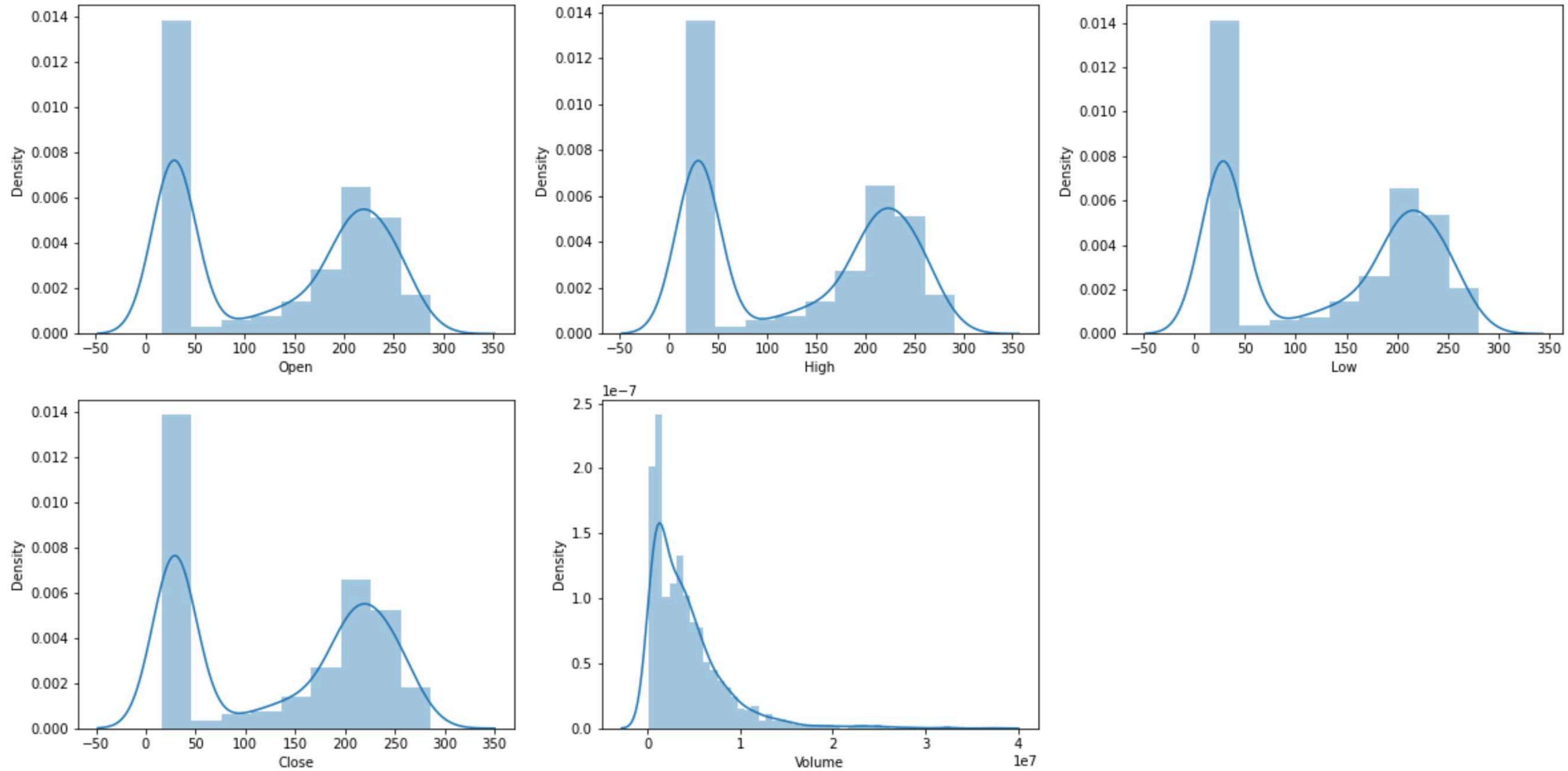
Before moving further let's check for the null values if any are present in the data frame.

In [16]:
```
df.isnull().sum()
```

Out[16]:
```
Date      0
Open      0
High      0
Low       0
Close     0
Volume    0
dtype: int64
```

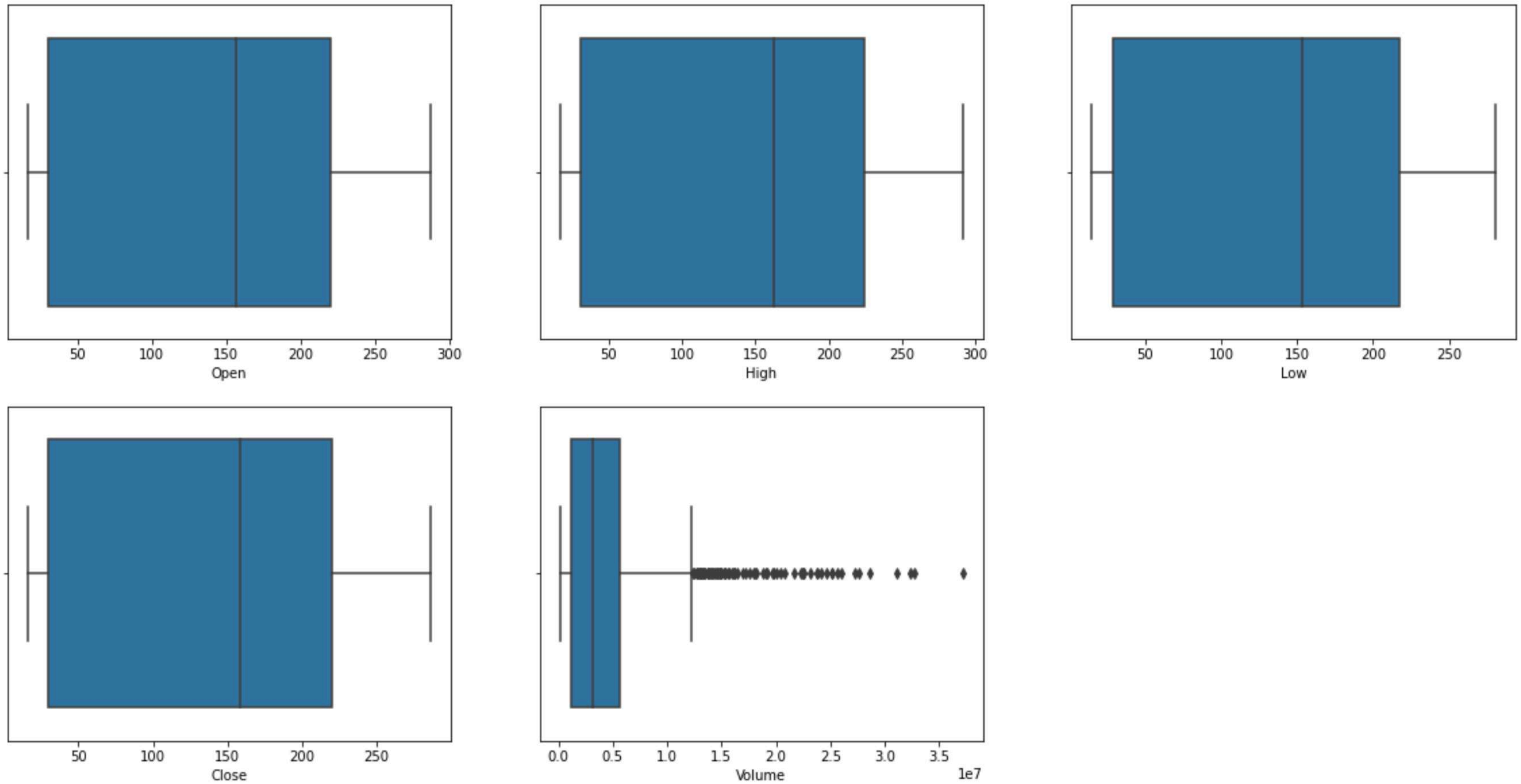
This implies that there are no null values in the data set provided.

In [17]:
```
features = ['Open', 'High', 'Low', 'Close', 'Volume']

plt.subplots(figsize=(20,10))

for i, col in enumerate(features):
  plt.subplot(2,3,i+1)
  sb.distplot(df[col])
plt.show()
```



In the distribution plot of OHLC data, we can see two peaks which means the data has varied significantly in two regions. And the Volume data is left-skewed.

In [18]:
```
plt.subplots(figsize=(20,10))
for i, col in enumerate(features):
    plt.subplot(2,3,i+1)
    sb.boxplot(df[col])
plt.show()
```



From the above boxplots, we can conclude that only volume data contains outliers in it but the data in the rest of the columns are free from any outlier.

Feature Engineering

Feature Engineering helps to derive some valuable features from the existing ones. These extra features sometimes help in increasing the performance of the model significantly and certainly help to gain deeper insights into the data.

In [19]:

```
splitted = df['Date'].str.split('/', expand=True)

df['day'] = splitted[1].astype('int')
df['month'] = splitted[0].astype('int')
df['year'] = splitted[2].astype('int')
```

```
df.head()
```

Out[19]:

	Date	Open	High	Low	Close	Volume	day	month	year
0	6/29/2010	19.000000	25.00	17.540001	23.889999	18766300	29	6	2010
1	6/30/2010	25.790001	30.42	23.299999	23.830000	17187100	30	6	2010
2	7/1/2010	25.000000	25.92	20.270000	21.959999	8218800	1	7	2010
3	7/2/2010	23.000000	23.10	18.709999	19.200001	5139800	2	7	2010
4	7/6/2010	20.000000	20.00	15.830000	16.110001	6866900	6	7	2010

Now we have three more columns namely ‘day’, ‘month’ and ‘year’ all these three have been derived from the ‘Date’ column which was initially provided in the data.

In [20]:

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

Out[20]:

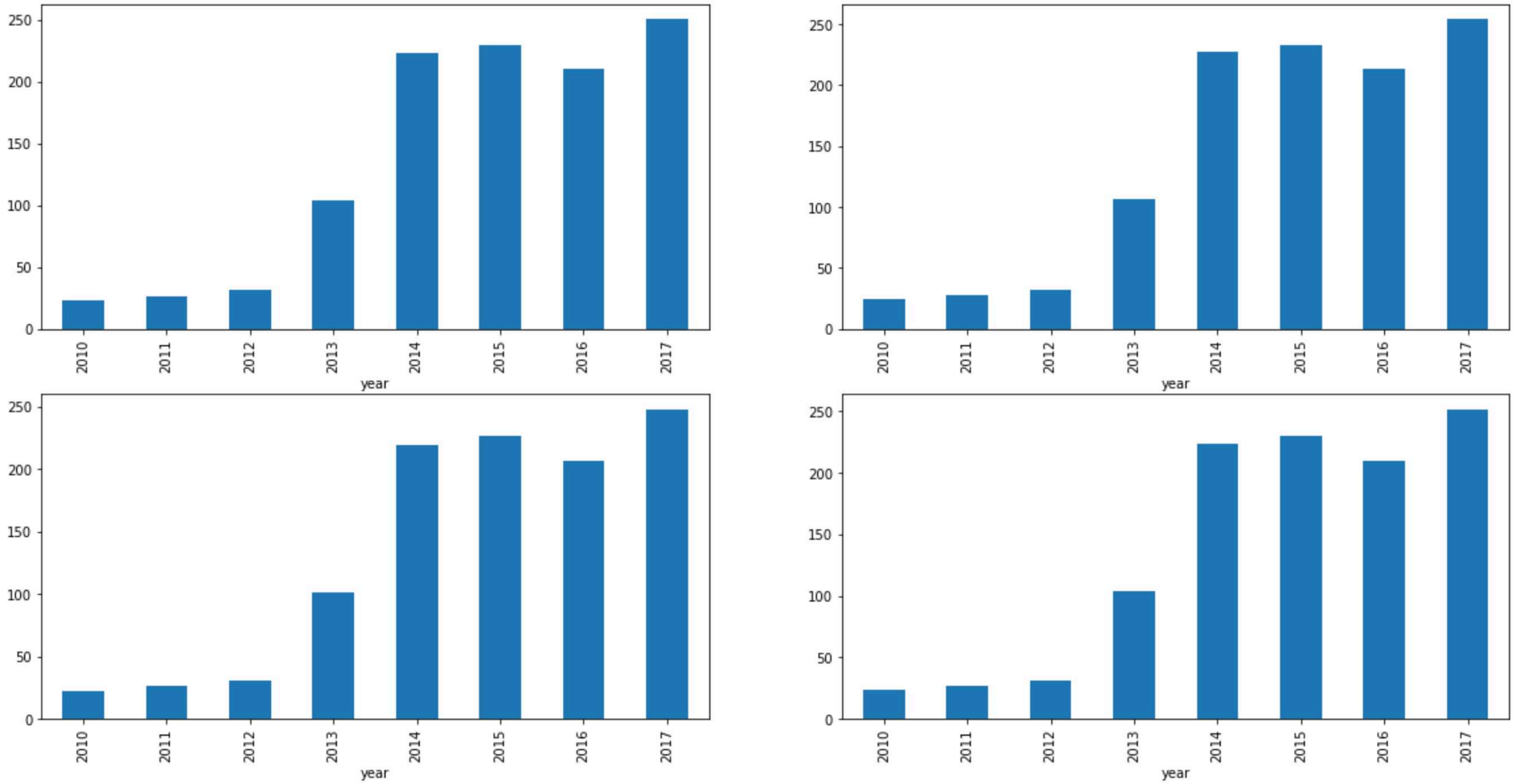
	Date	Open	High	Low	Close	Volume	day	month	year	is_quarter_end
0	6/29/2010	19.000000	25.00	17.540001	23.889999	18766300	29	6	2010	1
1	6/30/2010	25.790001	30.42	23.299999	23.830000	17187100	30	6	2010	1
2	7/1/2010	25.000000	25.92	20.270000	21.959999	8218800	1	7	2010	0
3	7/2/2010	23.000000	23.10	18.709999	19.200001	5139800	2	7	2010	0
4	7/6/2010	20.000000	20.00	15.830000	16.110001	6866900	6	7	2010	0

A quarter is defined as a group of three months. Every company prepares its quarterly results and publishes them publically so, that people can analyze the company’s performance. These quarterly results affect the stock prices heavily which is why we have added this feature because this can be a helpful feature for the learning model.

In [21]:

```
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()
```



From the above bar graph, we can conclude that the stock prices have doubled from the year 2013 to that in 2014.

In [22]:
```
df.groupby('is_quarter_end').mean()
```

Out[22]:

	Open	High	Low	Close	Volume	day	month	year
is_quarter_end								
0	130.813739	133.182620	128.257229	130.797709	4.461581e+06	15.686501	6.141208	2013.353464
1	135.679982	137.927032	133.455777	135.673269	3.891084e+06	15.657244	7.584806	2013.314488

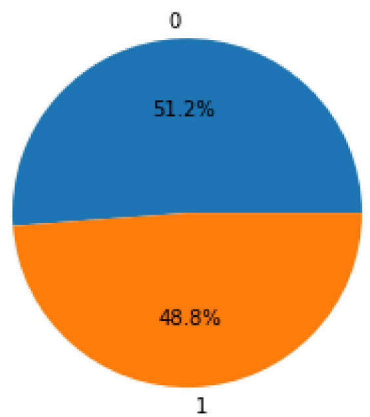
Here are some of the important observations of the above-grouped data:

Prices are higher in the months which are quarter end as compared to that of the non-quarter end months. The volume of trades is lower in the months which are quarter end.

In [23]:
```
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)
```

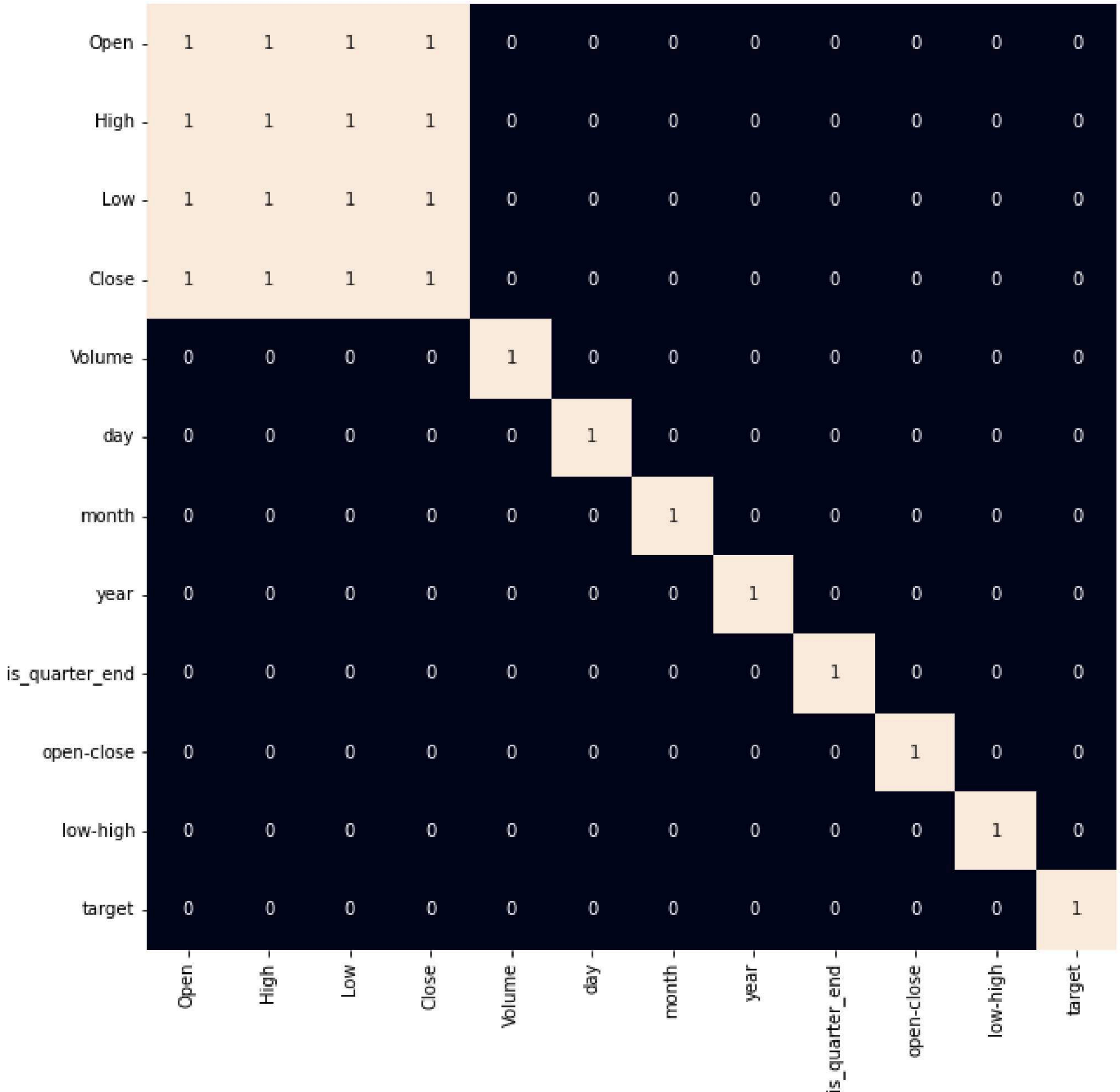
Above we have added some more columns which will help in the training of our model. We have added the target feature which is a signal whether to buy or not we will train our model to predict this only. But before proceeding let's check whether the target is balanced or not using a pie chart.

In [24]:
```
plt.pie(df['target'].value_counts().values,
labels=[0, 1], autopct='%1.1f%%')
plt.show()
```



When we add features to our dataset we have to ensure that there are no highly correlated features as they do not help in the learning process of the algorithm.

In [25]:
```
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()
```



From the above heatmap, we can say that there is a high correlation between OHLC that is pretty obvious and the added features are not highly correlated with each other or previously provided features which means that we are good to go and build our model.

Data Splitting and Normalization

In [26]:
```
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)
```

```
(1522, 3) (170, 3)
```

After selecting the features to train the model on we should normalize the data because normalized data leads to stable and fast training of the model. After that whole data has been split into two parts with a 90/10 ratio so, that we can evaluate the performance of our model on unseen data.

Model Development and Evaluation

Now is the time to train some state-of-the-art machine learning models(Logistic Regression, Support Vector Machine, XGBClassifier), and then based on their performance on the training and validation data we will choose which ML model is serving the purpose at hand better.

For the evaluation metric, we will use the ROC-AUC curve but why this is because instead of predicting the hard probability that is 0 or 1 we would like it to predict soft probabilities that are continuous values between 0 to 1. And with soft probabilities, the ROC-AUC curve is generally used to measure the accuracy of the predictions.

In [56]:
```
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(
```

```python
    Y_valid, models[i].predict_proba(X_valid)[:,1]))
    print()
```

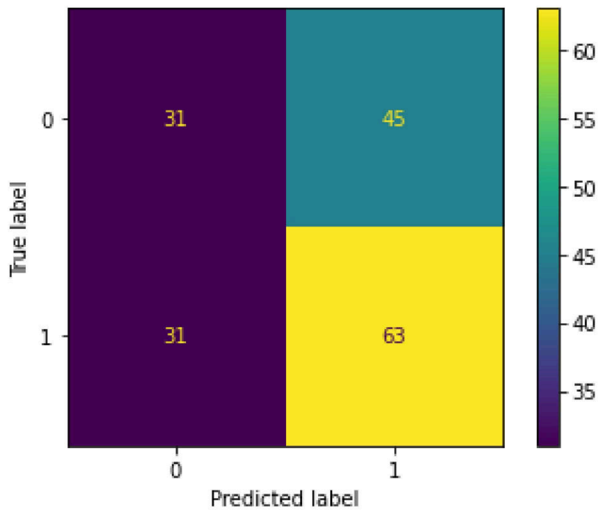
```
LogisticRegression() :
Training Accuracy :  0.5191709844559586
Validation Accuracy :  0.5435330347144457

SVC(kernel='poly', probability=True) :
Training Accuracy :  0.4734758203799654
Validation Accuracy :  0.44260918253079506

XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, ...) :
Training Accuracy :  0.9764784110535405
Validation Accuracy :  0.5187569988801792
```

Among the three models, we have trained XGBClassifier has the highest performance but it is pruned to overfitting as the difference between the training and the validation accuracy is too high. But in the case of the Logistic Regression, this is not the case.

Now let's plot a confusion matrix for the validation data.

In [57]:

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



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%. Possible reasons for this may be the lack of data or using a very simple model to perform such a complex task as Stock Market prediction.