In [ ]:

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

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('BTC_dataset.csv', encoding='utf-8')
df.head(10)
```

Out[ ]:

	timestamp	open	high	low	close	volume	quote_asset_volume	numb
0	2023-08-01 13:19:00	28902.48	28902.49	28902.48	28902.49	4.68658	1.354538e+05	
1	2023-08-01 13:18:00	28902.48	28902.49	28902.48	28902.49	4.77589	1.380351e+05	
2	2023-08-01 13:17:00	28908.52	28908.53	28902.48	28902.49	11.52263	3.330532e+05	
3	2023-08-01 13:16:00	28907.41	28912.74	28907.41	28908.53	15.89610	4.595556e+05	
4	2023-08-01 13:15:00	28896.00	28907.42	28893.03	28907.41	37.74657	1.090761e+06	
5	2023-08-01 13:14:00	28890.40	28896.00	28890.39	28895.99	9.88869	2.857173e+05	
6	2023-08-01 13:13:00	28889.63	28890.40	28889.63	28890.39	17.87871	5.165159e+05	
7	2023-08-01 13:12:00	28881.54	28889.64	28881.53	28889.64	13.48153	3.894235e+05	
8	2023-08-01 13:11:00	28876.00	28881.54	28875.99	28881.54	6.85924	1.980829e+05	
9	2023-08-01 13:10:00	28872.48	28876.00	28870.00	28876.00	10.75734	3.105872e+05	

In [ ]:

```python
# Remove time in timestamp column
df['timestamp'] = pd.to_datetime(df['timestamp'])
```

```python
df['timestamp'] = df['timestamp'].dt.date
df.head(10)
```

Out[ ]:

	timestamp	open	high	low	close	volume	quote_asset_volume	numb
0	2023-08-01	28902.48	28902.49	28902.48	28902.49	4.68658	1.354538e+05	
1	2023-08-01	28902.48	28902.49	28902.48	28902.49	4.77589	1.380351e+05	
2	2023-08-01	28908.52	28908.53	28902.48	28902.49	11.52263	3.330532e+05	
3	2023-08-01	28907.41	28912.74	28907.41	28908.53	15.89610	4.595556e+05	
4	2023-08-01	28896.00	28907.42	28893.03	28907.41	37.74657	1.090761e+06	
5	2023-08-01	28890.40	28896.00	28890.39	28895.99	9.88869	2.857173e+05	
6	2023-08-01	28889.63	28890.40	28889.63	28890.39	17.87871	5.165159e+05	
7	2023-08-01	28881.54	28889.64	28881.53	28889.64	13.48153	3.894235e+05	
8	2023-08-01	28876.00	28881.54	28875.99	28881.54	6.85924	1.980829e+05	
9	2023-08-01	28872.48	28876.00	28870.00	28876.00	10.75734	3.105872e+05	

In [ ]:

```python
# Data's information
df.describe()
```

Out[ ]:

	open	high	low	close	volume	quote_asset
count	3.126000e+06	3.126000e+06	3.126000e+06	3.126000e+06	3.126000e+06	3.126
mean	2.008947e+04	2.010217e+04	2.007666e+04	2.008946e+04	5.290800e+01	1.155
std	1.605896e+04	1.606926e+04	1.604871e+04	1.605896e+04	9.774388e+01	2.335
min	2.830000e+03	2.830000e+03	2.817000e+03	2.817000e+03	0.000000e+00	0.000
25%	7.624747e+03	7.629600e+03	7.620000e+03	7.624798e+03	1.120167e+01	1.122
50%	1.169999e+04	1.170681e+04	1.169249e+04	1.170000e+04	2.387539e+01	3.706
75%	2.989957e+04	2.990724e+04	2.989051e+04	2.989957e+04	5.393630e+01	1.276
max	6.900000e+04	6.900000e+04	6.878670e+04	6.900000e+04	5.877775e+03	1.459

In [ ]:
```
df.info()
```

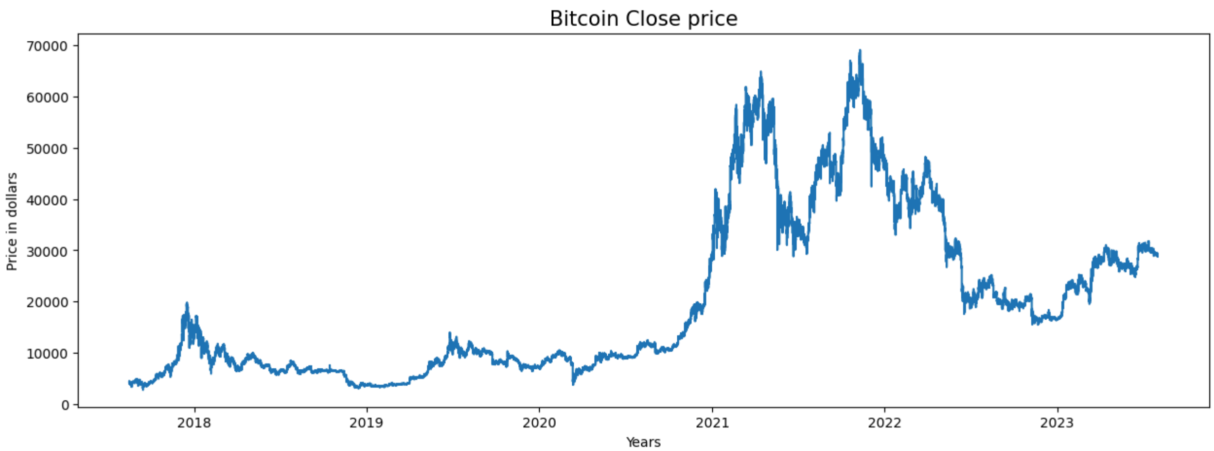
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3126000 entries, 0 to 3125999
Data columns (total 10 columns):
 #   Column                        Dtype  
---  ------                        -----  
 0   timestamp                     object 
 1   open                          float64
 2   high                          float64
 3   low                           float64
 4   close                         float64
 5   volume                        float64
 6   quote_asset_volume            float64
 7   number_of_trades              int64  
 8   taker_buy_base_asset_volume   float64
 9   taker_buy_quote_asset_volume  float64
dtypes: float64(8), int64(1), object(1)
memory usage: 238.5+ MB
```

In [ ]:
```
# EDA process
# Check null Values
df.isnull().sum()
```

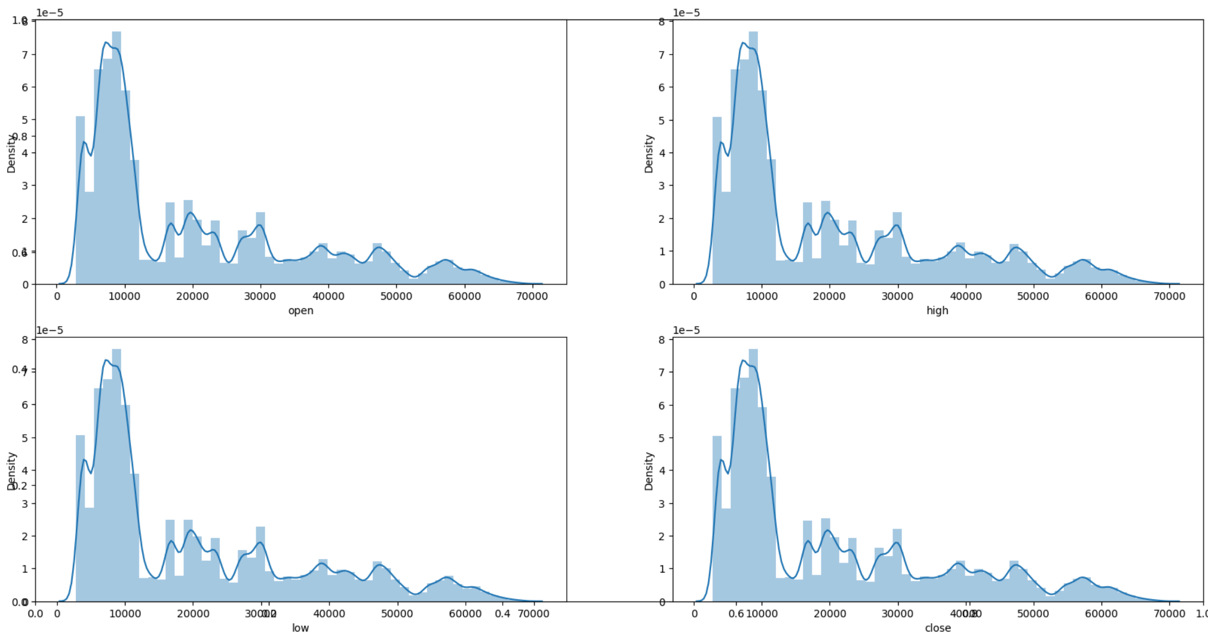
Out[ ]:
```
timestamp                       0
open                            0
high                            0
low                             0
close                           0
volume                          0
quote_asset_volume              0
number_of_trades                0
taker_buy_base_asset_volume     0
taker_buy_quote_asset_volume    0
dtype: int64
```

- There are no null values in the dataset

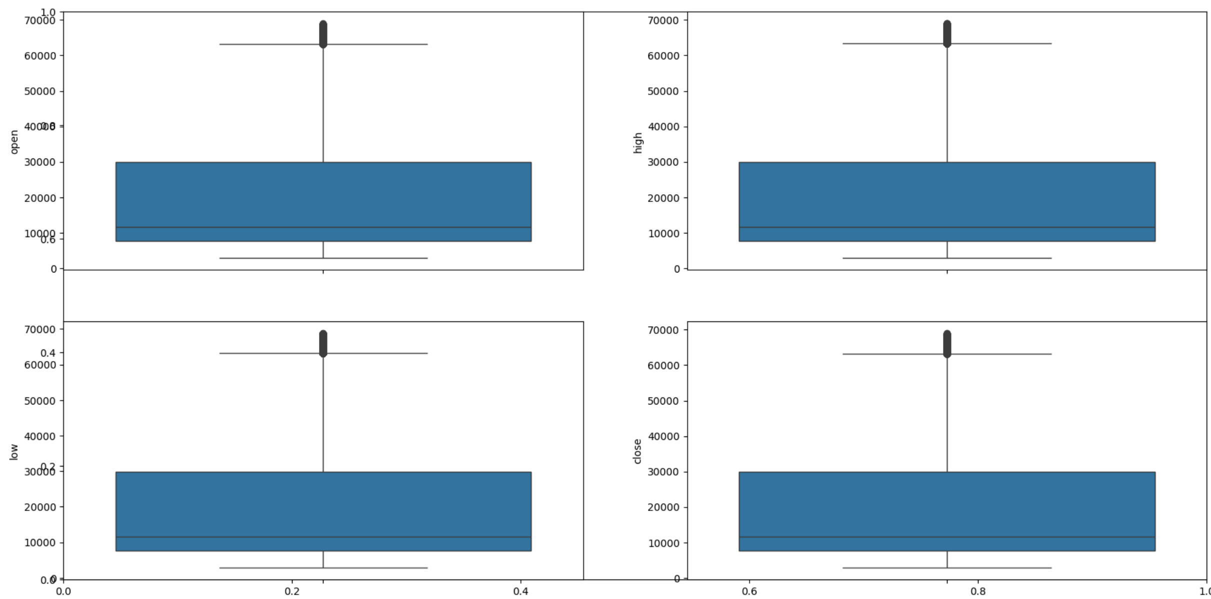
In [ ]:
```
plt.figure(figsize=(15,5))
plt.plot(df['timestamp'], df['close'])
plt.title('Bitcoin Close price', fontsize= 15)
plt.xlabel('Years')
plt.ylabel('Price in dollars')
plt.show()
```



In [ ]:
```python
# Plot graph for open, high, low, close
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()
```



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



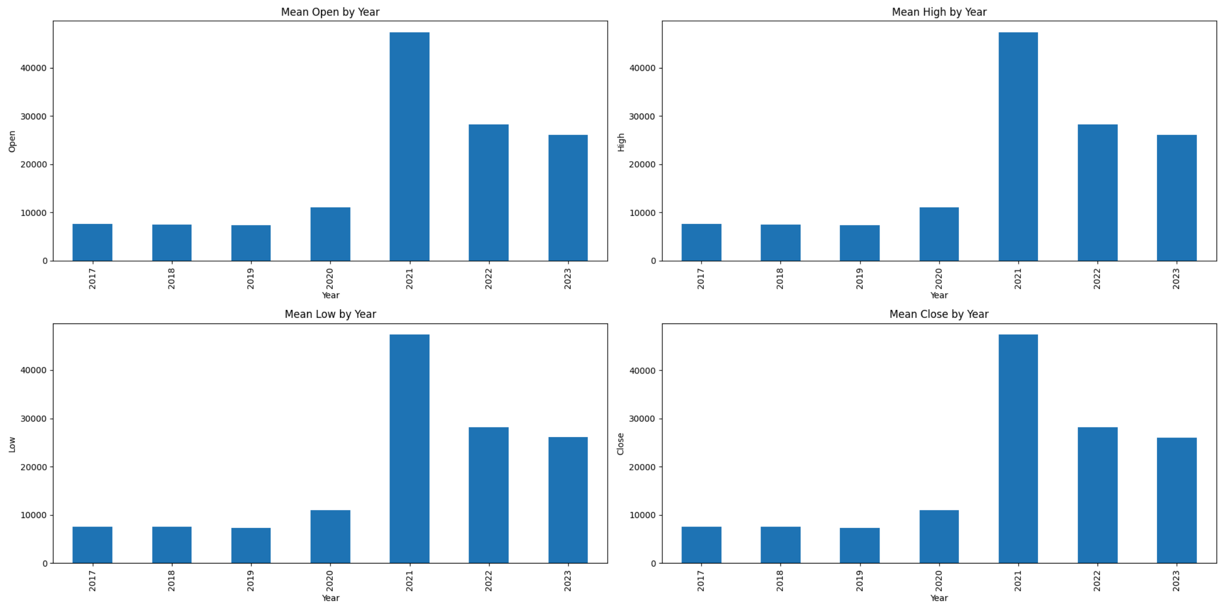
In [ ]:
```
df['timestamp'] = df['timestamp'].astype(str)
splitted = df['timestamp'].str.split('-', expand=True)
df['year'] = splitted[0].astype('int')
df['month'] = splitted[1].astype('int')
df['day'] = splitted[2].astype('int')

df.head()
```

Out[ ]:

	timestamp	open	high	low	close	volume	quote_asset_volume	numb
0	2023-08-01	28902.48	28902.49	28902.48	28902.49	4.68658	1.354538e+05	
1	2023-08-01	28902.48	28902.49	28902.48	28902.49	4.77589	1.380351e+05	
2	2023-08-01	28908.52	28908.53	28902.48	28902.49	11.52263	3.330532e+05	
3	2023-08-01	28907.41	28912.74	28907.41	28908.53	15.89610	4.595556e+05	
4	2023-08-01	28896.00	28907.42	28893.03	28907.41	37.74657	1.090761e+06	

In [ ]:
```
# Select only numeric columns for calculating the mean
numeric_df = df.select_dtypes(include=['number'])

# Group by 'year' and calculate the mean for numeric columns
data_grouped = numeric_df.groupby(df['year']).mean()

# Plotting the data
plt.subplots(2, 2, figsize=(20, 10))
for i, col in enumerate(['open', 'high', 'low', 'close']):
    plt.subplot(2, 2, i + 1)
    data_grouped[col].plot(kind='bar')
```

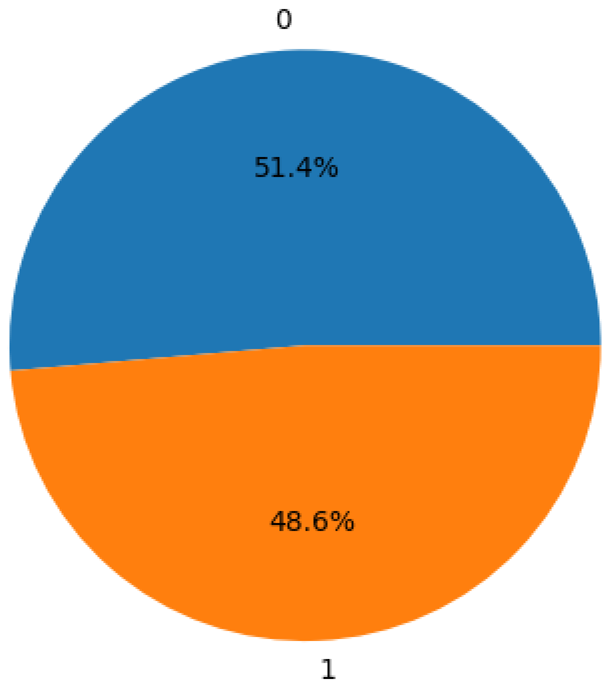
```
    plt.title(f'Mean {col.capitalize()} by Year')  # Added title for clarity
    plt.xlabel('Year')
    plt.ylabel(f'{col.capitalize()}')

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



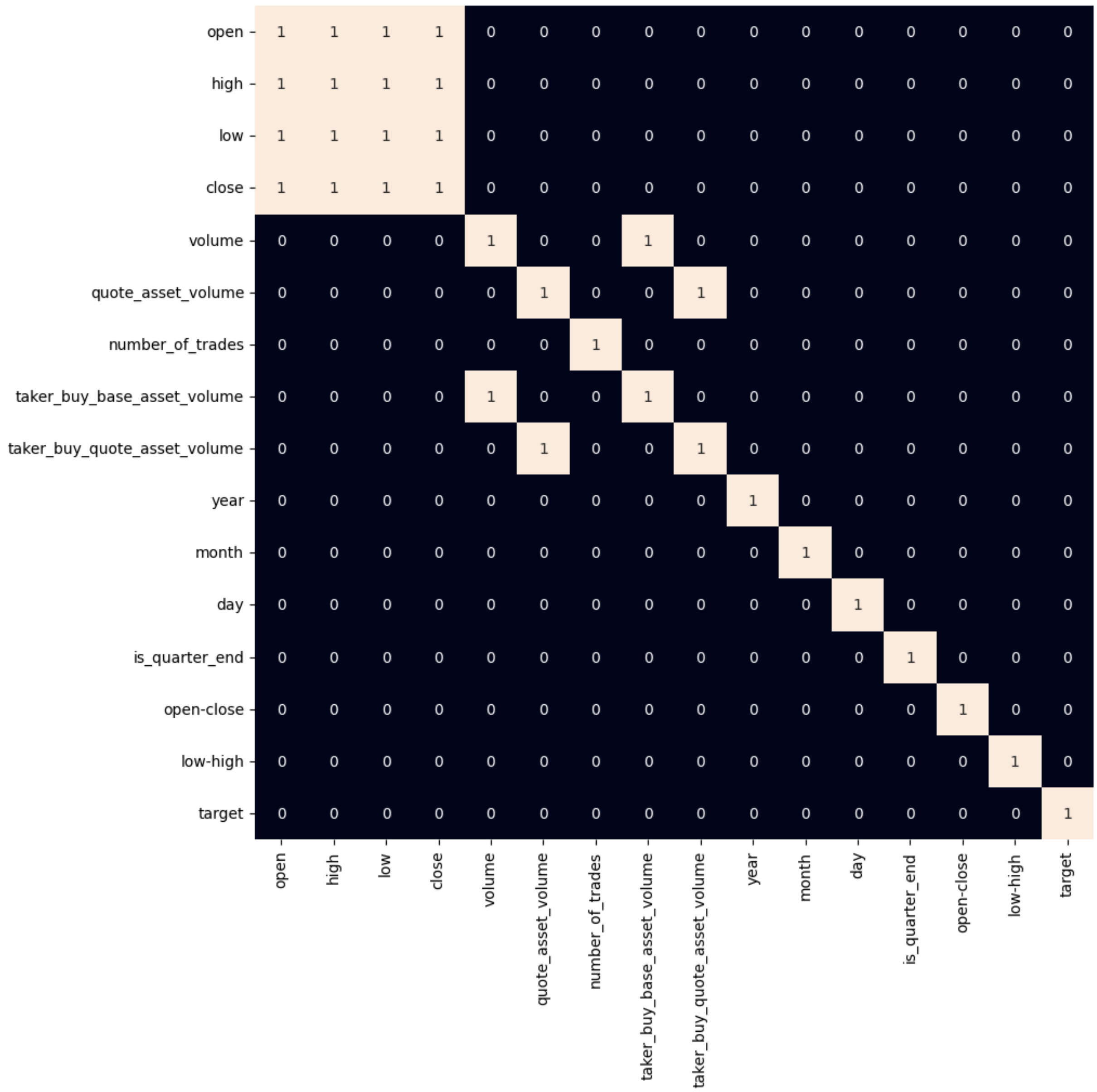
We have added the target feature which is a signal whether to buy or not we will train our model to predict this only. But before proceeding let's check whether the target is balanced or not using a pie chart.

In [ ]:

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



In [ ]:
```
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.
numeric_df = df.select_dtypes(include='number')
sb.heatmap(numeric_df.corr() > 0.9, annot=True, cbar=False)
plt.show()
```



we can say that there is a high correlation between OHLC which 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.

In [ ]:

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

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

X_train, X_test, Y_train, Y_test = train_test_split(
        features, target, test_size=0.2, random_state=2000)
print(X_train.shape, X_test.shape)

models = [LogisticRegression(), XGBClassifier()]

for i in range(2):
        models[i].fit(X_train, Y_train)
```

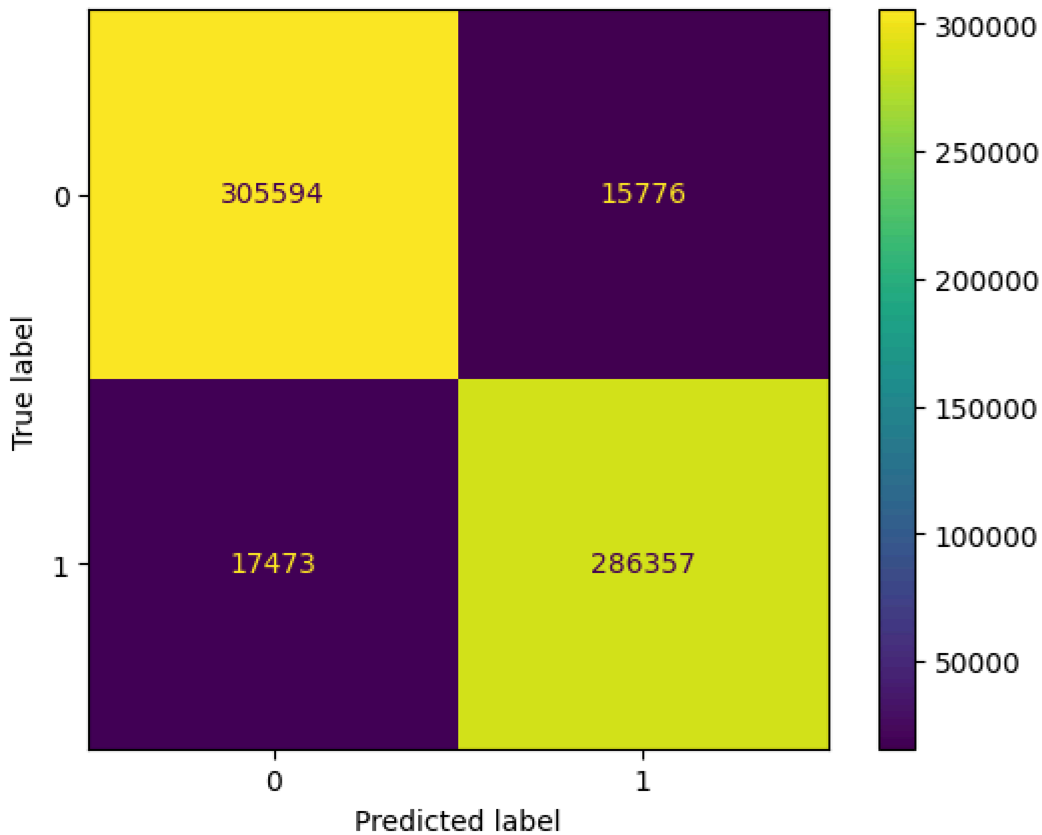
```
        print(f'{models[i]} : ')
        print('Training Accuracy : ', metrics.roc_auc_score(Y_train, models[i].pred
        print('Validation Accuracy : ', metrics.roc_auc_score(Y_test, models[i].pre
        print()
```

```
(2500800, 3) (625200, 3)
LogisticRegression() :
Training Accuracy :  0.9776496212403469
Validation Accuracy :  0.9775479629522494

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 :  0.9805018215863894
Validation Accuracy :  0.979759032963849
```

we have trained XGBClassifier has the highest performance

In [ ]:

```
from sklearn.metrics import ConfusionMatrixDisplay

# Generate the confusion matrix using the model's predictions
y_pred = models[1].predict(X_test)

# Create and plot the confusion matrix display
ConfusionMatrixDisplay.from_estimator(models[1], X_test, Y_test)

plt.show()
```



Summary: The confusion matrix indicates that your model has a relatively high number of correct predictions (both true positives and true negatives). The false positive and false negative rates are relatively low compared to the correct predictions, suggesting that the model is performing well. However, depending on the specific application, the rates of false positives and false negatives may still be significant and warrant further tuning or investigation. Further Analysis: Accuracy Calculation: You can derive the overall accuracy from the confusion matrix by summing the true positives and true negatives and dividing by the total number of instances.

Accuracy = (305594 + 286357)/(305594 + 15776 + 17473 + 286357) = 94.68 %

Precision, Recall, F1-Score: These metrics can be calculated to provide a more nuanced view of your model’s performance, especially in cases where the classes are imbalanced.

This explanation provides a clear understanding of how to interpret the confusion matrix and the implications of the results.