```python
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('bitcoin.csv')
df.head()
```

```
         Date        Open        High         Low       Close    Adj
Close  \
0  2014-09-17  465.864014  468.174011  452.421997  457.334015
457.334015
1  2014-09-18  456.859985  456.859985  413.104004  424.440002
424.440002
2  2014-09-19  424.102997  427.834991  384.532013  394.795990
394.795990
3  2014-09-20  394.673004  423.295990  389.882996  408.903992
408.903992
4  2014-09-21  408.084991  412.425995  393.181000  398.821014
398.821014

     Volume
0  21056800
1  34483200
2  37919700
3  36863600
4  26580100
```

```python
df.shape
```

```
(2713, 7)
```

```python
df.describe()
```

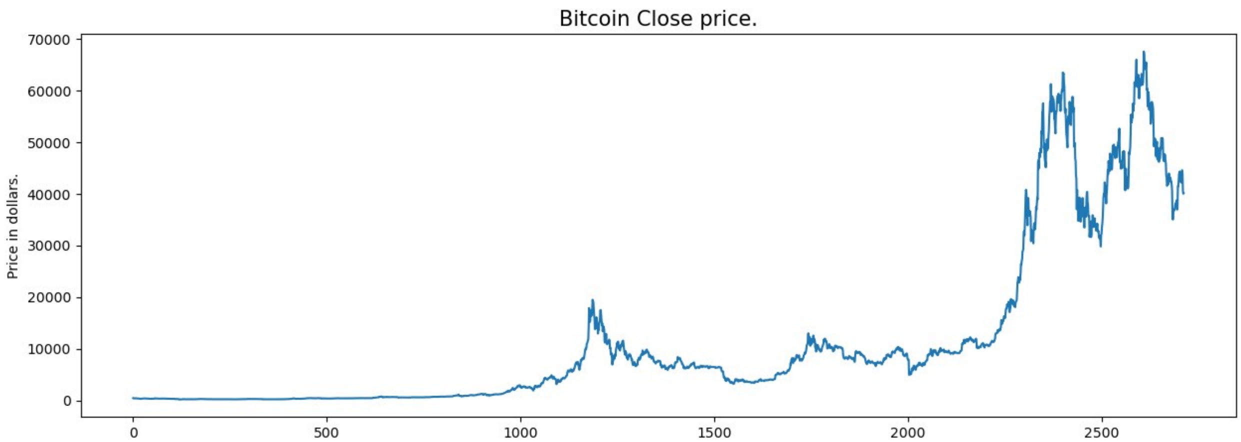
```
               Open          High           Low         Close     Adj
Close  \
count   2713.000000   2713.000000   2713.000000   2713.000000
2713.000000
mean   11311.041069  11614.292482  10975.555057  11323.914637
11323.914637
std    16106.428891  16537.390649  15608.572560  16110.365010
```

```
16110.365010
min      176.897003    211.731003    171.509995    178.102997
178.102997
25%      606.396973    609.260986    604.109985    606.718994
606.718994
50%     6301.569824   6434.617676   6214.220215   6317.609863
6317.609863
75%    10452.399414  10762.644531  10202.387695  10462.259766
10462.259766
max    67549.734375  68789.625000  66382.062500  67566.828125
67566.828125

             Volume
count  2.713000e+03
mean   1.470462e+10
std    2.001627e+10
min    5.914570e+06
25%    7.991080e+07
50%    5.098183e+09
75%    2.456992e+10
max    3.509679e+11
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2713 entries, 0 to 2712
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       2713 non-null   object
 1   Open       2713 non-null   float64
 2   High       2713 non-null   float64
 3   Low        2713 non-null   float64
 4   Close      2713 non-null   float64
 5   Adj Close  2713 non-null   float64
 6   Volume     2713 non-null   int64
dtypes: float64(5), int64(1), object(1)
memory usage: 148.5+ 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[df['Close'] == df['Adj Close']].shape, df.shape
```

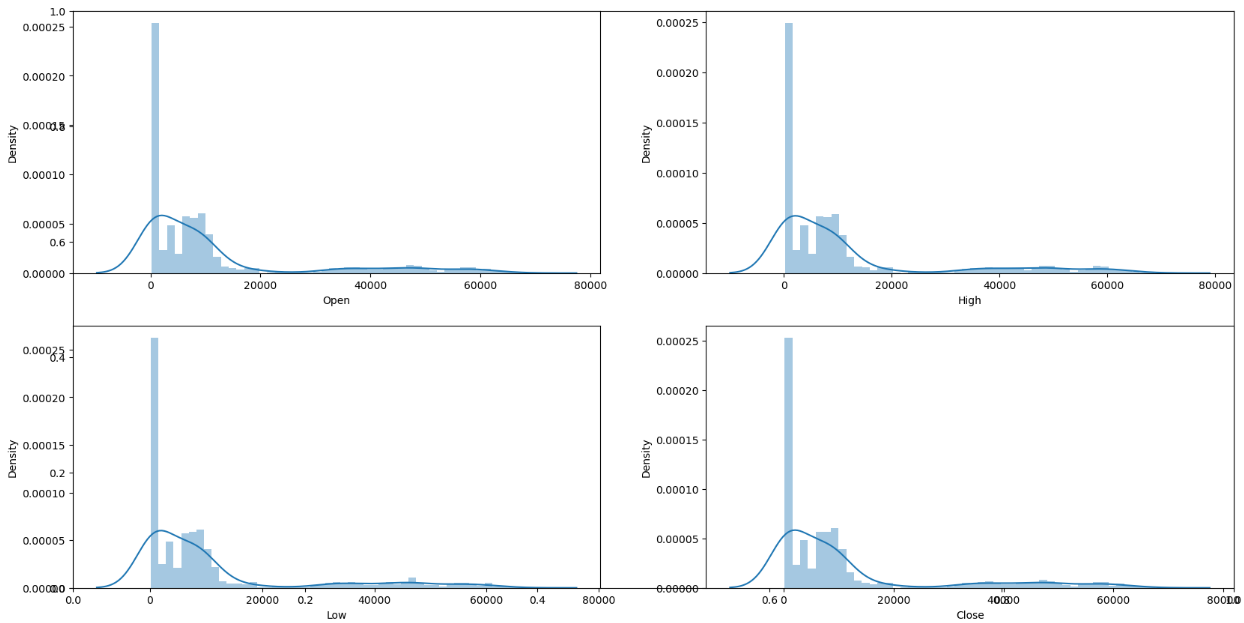
```
((2713, 7), (2713, 7))
```

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

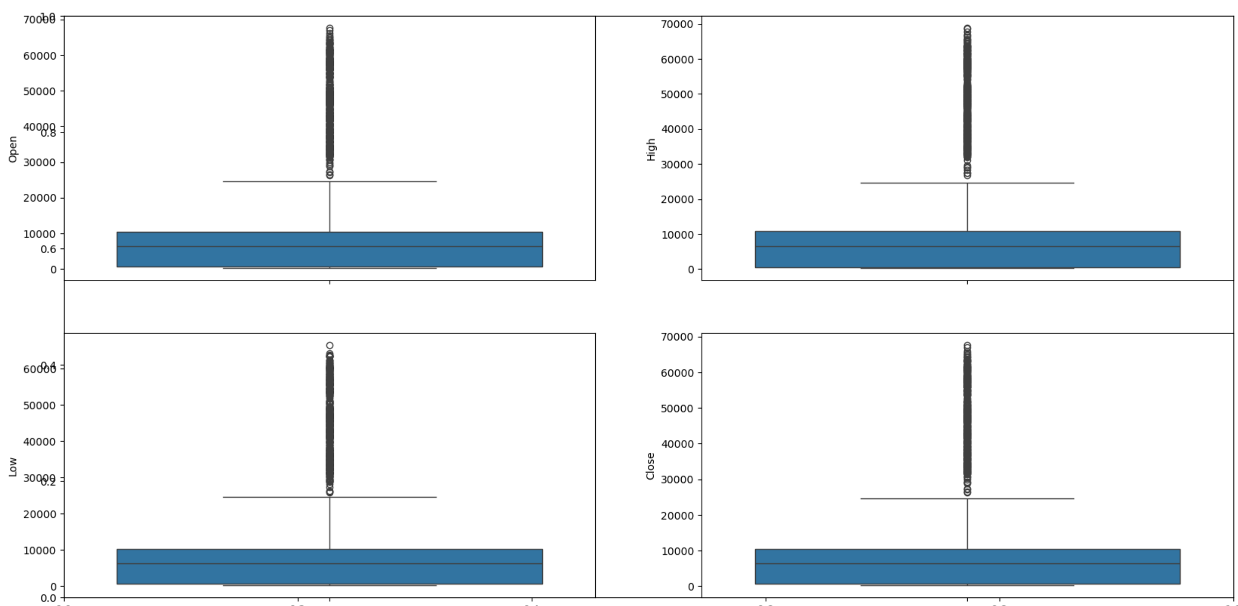
```
df.isnull().sum()
```

```
Date      0
Open      0
High      0
Low       0
Close     0
Volume    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()
```




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

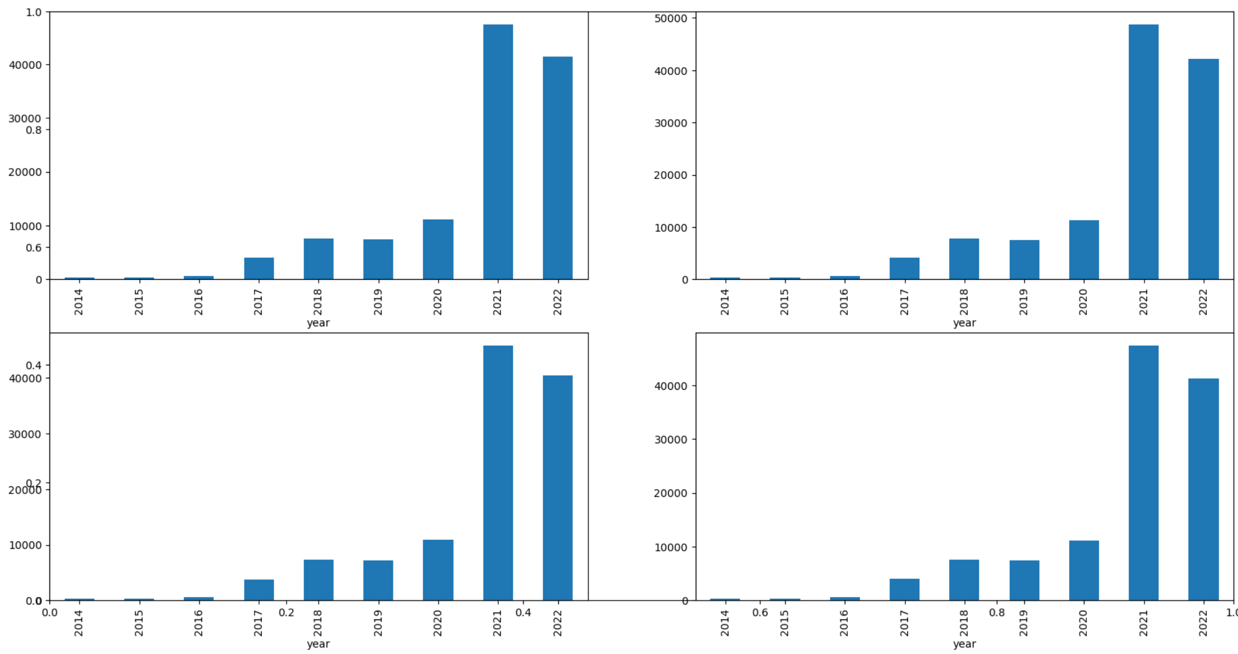



```python
splitted = df['Date'].str.split('-', expand=True)

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

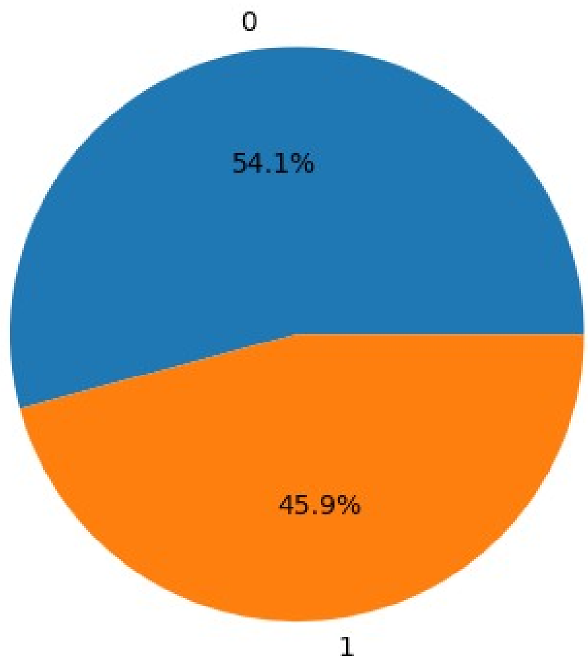
```
# Convert the 'Date' column to datetime objects
df['Date'] = pd.to_datetime(df['Date'])

df.head()

# This code is modified by Susobhan Akhuli
```

```
        Date        Open        High         Low       Close    Volume  
year  \
0 2014-09-17  465.864014  468.174011  452.421997  457.334015  21056800
2014
1 2014-09-18  456.859985  456.859985  413.104004  424.440002  34483200
2014
2 2014-09-19  424.102997  427.834991  384.532013  394.795990  37919700
2014
3 2014-09-20  394.673004  423.295990  389.882996  408.903992  36863600
2014
4 2014-09-21  408.084991  412.425995  393.181000  398.821014  26580100
2014

   month  day
0      9   17
1      9   18
2      9   19
3      9   20
4      9   21
```

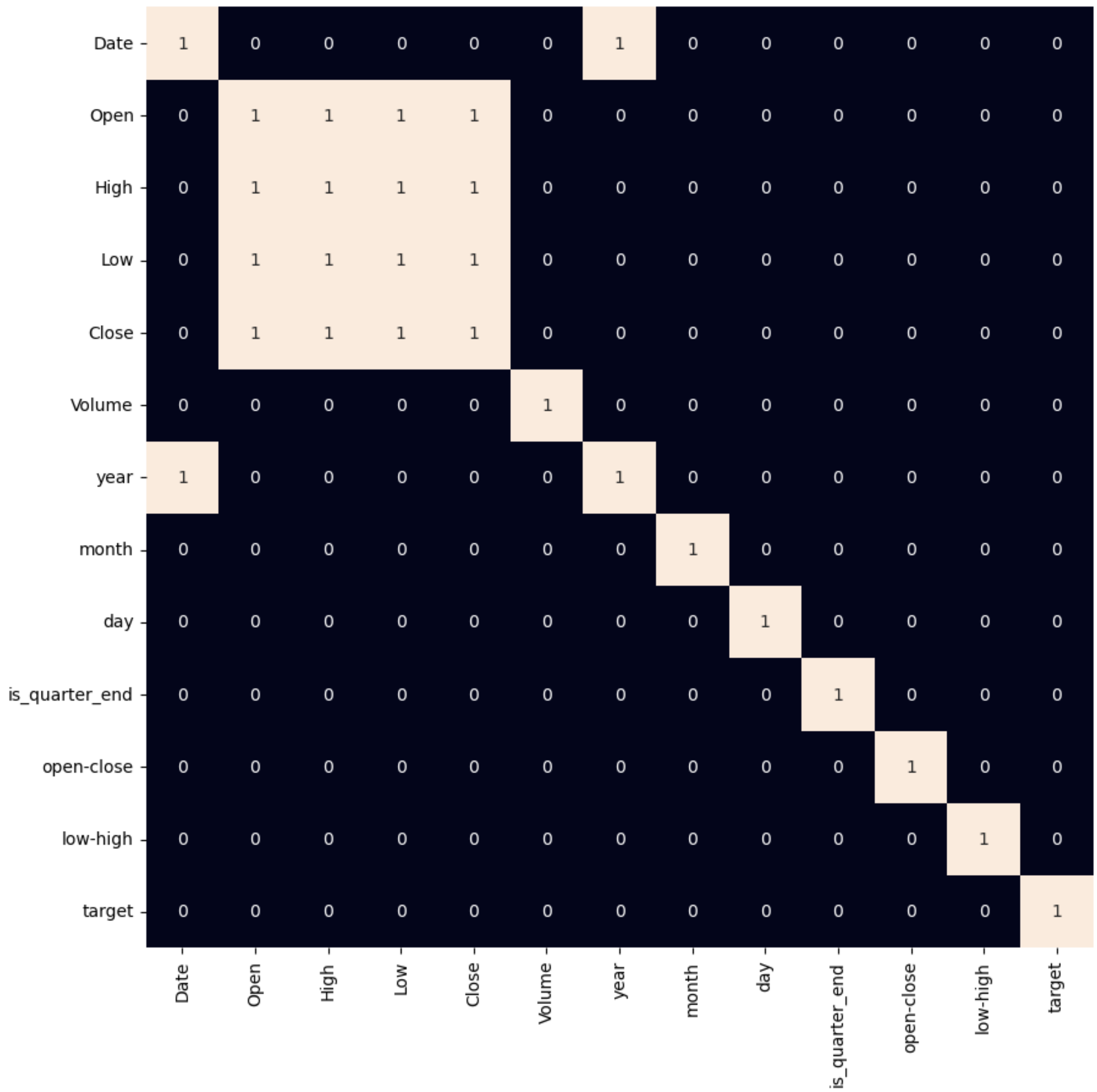
```
data_grouped = df.groupby('year').mean()
plt.subplots(figsize=(20,10))
for i, col in enumerate(['Open', 'High', 'Low', 'Close']):
  plt.subplot(2,2,i+1)
  data_grouped[col].plot.bar()
plt.show()
```




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

         Date        Open        High         Low       Close    Volume
year  \
0 2014-09-17  465.864014  468.174011  452.421997  457.334015  21056800
2014
1 2014-09-18  456.859985  456.859985  413.104004  424.440002  34483200
2014
2 2014-09-19  424.102997  427.834991  384.532013  394.795990  37919700
2014
3 2014-09-20  394.673004  423.295990  389.882996  408.903992  36863600
2014
4 2014-09-21  408.084991  412.425995  393.181000  398.821014  26580100
2014

   month  day  is_quarter_end
0      9   17               1
1      9   18               1
2      9   19               1
3      9   20               1
4      9   21               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()
```

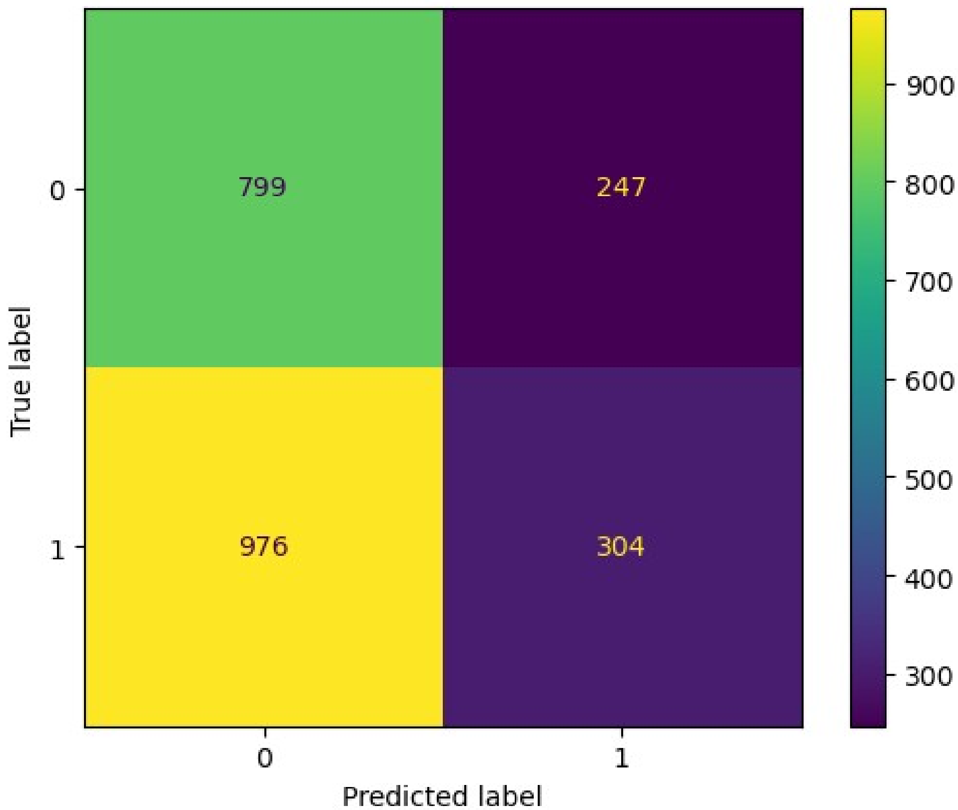



```
plt.figure(figsize=(10, 10))

# 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)
#We do not use train test split, rather use the first 70% data to
train and last 30% to test
X_train, X_valid, Y_train, Y_valid = X_train, X_valid, Y_train,
Y_valid =
features[:len(features)//7],features[len(features)//7:],target[:len(fe
atures)//7],target[len(features)//7:]
```

```
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 :  0.5351397573619796
Validation Accuracy :  0.5170956321701721

SVC(kernel='poly', probability=True) :
Training Accuracy :  0.4620811287477955
Validation Accuracy :  0.4875664734703633

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,
              feature_weights=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, ...) :
Training Accuracy :  0.9993586660253327
Validation Accuracy :  0.5329379780114722


from sklearn.metrics import ConfusionMatrixDisplay

ConfusionMatrixDisplay.from_estimator(models[0], X_valid, Y_valid)
plt.show()

# This code is modified by Susobhan Akhuli
```



799
247
976
304
0
1
0
1
True label
Predicted label
900
800
700
600
500
400
300