In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
from datetime import datetime
import tkinter as tk
from tkinter import messagebox, font
import joblib
from prophet import Prophet
from PIL import Image, ImageTk

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')
```

In [3]:
```
data=pd.read_csv(r"C:\Users\grmus\Downloads\bitcoin.csv")
```

In [4]:
```
data.head()
```

Out[4]:

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

In [5]:
```
data.shape
```

Out[5]:
```
(2713, 7)
```

In [6]:
```
data.describe()
```

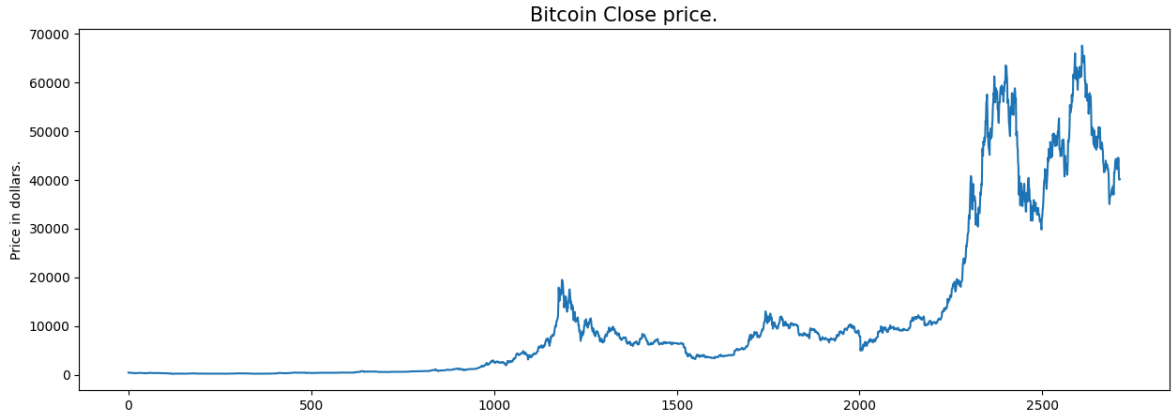
Out[6]:

	Open	High	Low	Close	Adj Close	Volu
count	2713.000000	2713.000000	2713.000000	2713.000000	2713.000000	2.713000e
mean	11311.041069	11614.292482	10975.555057	11323.914637	11323.914637	1.470462e
std	16106.428891	16537.390649	15608.572560	16110.365010	16110.365010	2.001627e
min	176.897003	211.731003	171.509995	178.102997	178.102997	5.914570e
25%	606.396973	609.260986	604.109985	606.718994	606.718994	7.991080e
50%	6301.569824	6434.617676	6214.220215	6317.609863	6317.609863	5.098183e
75%	10452.399414	10762.644531	10202.387695	10462.259766	10462.259766	2.456992e
max	67549.734375	68789.625000	66382.062500	67566.828125	67566.828125	3.509679e

In [7]:
```
data.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
```

In [8]:
```
plt.figure(figsize=(15, 5))
plt.plot(data['Close'])  # Use the correct column name here
plt.title('Bitcoin Close price.', fontsize=15)
plt.ylabel('Price in dollars.')
plt.show()
```



In [9]:
```
data[data['Close'] == data['Close']].shape, data.shape
```

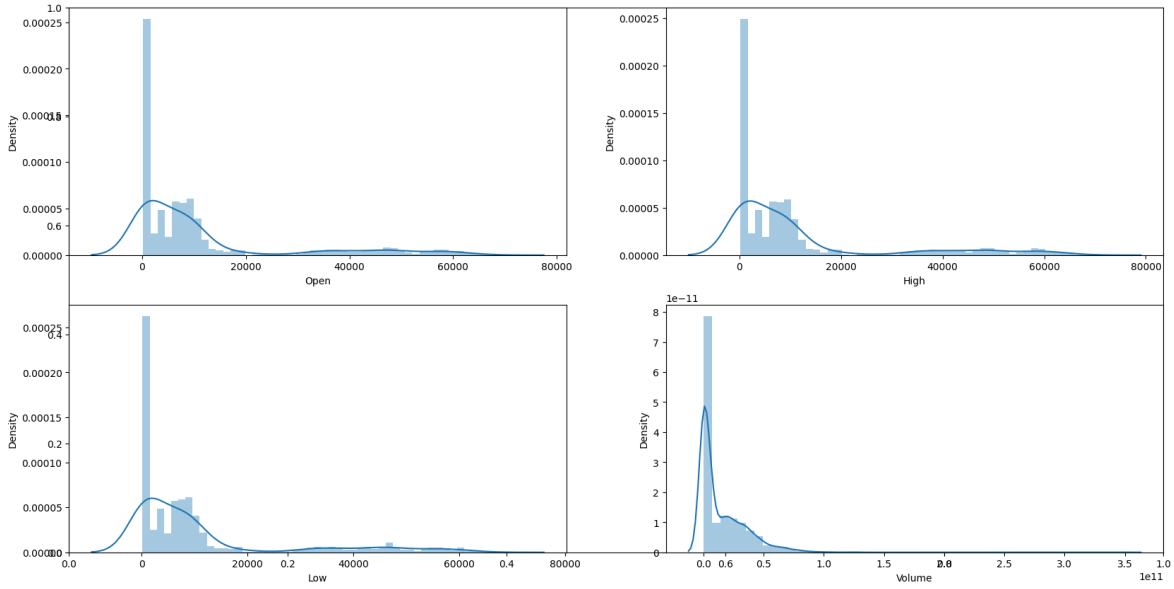
Out[9]: ((2713, 7), (2713, 7))

In [10]:
```
data = data.drop(['Close'], axis=1)
```

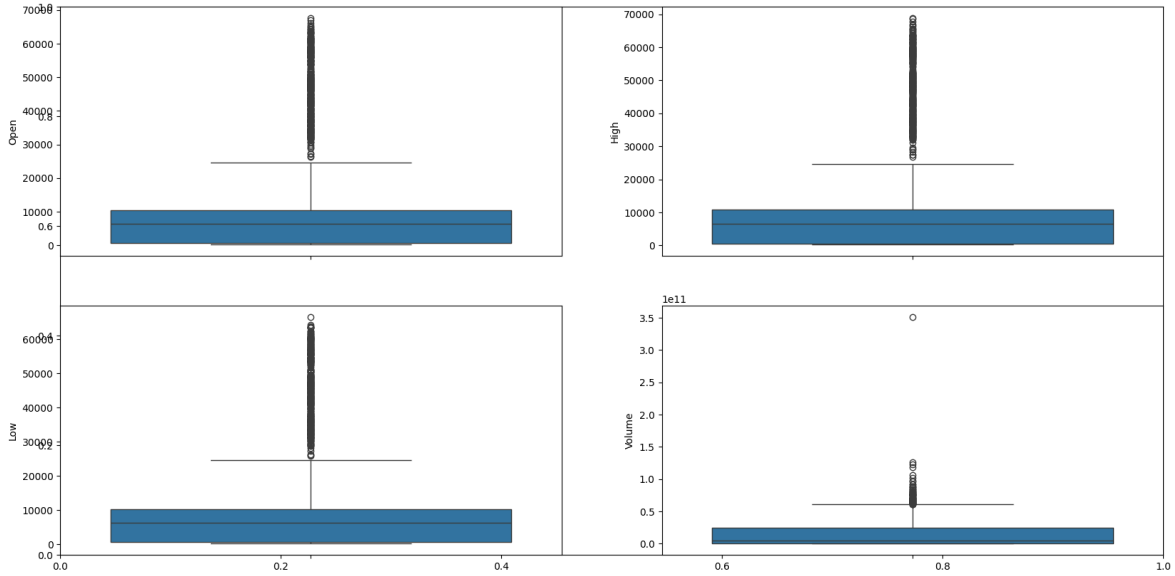
In [11]:
```
data.isnull().sum()
```

Out[11]:
```
Date         0
Open         0
High         0
Low          0
Adj Close    0
Volume       0
dtype: int64
```

In [12]:
```
features = ['Open', 'High', 'Low', 'Volume']

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



In [13]:
```
plt.subplots(figsize=(20,10))
for i, col in enumerate(features):
  plt.subplot(2,2,i+1)
  sb.boxplot(data[col])
plt.show()
```



In [14]:
```
print(data.columns)
```

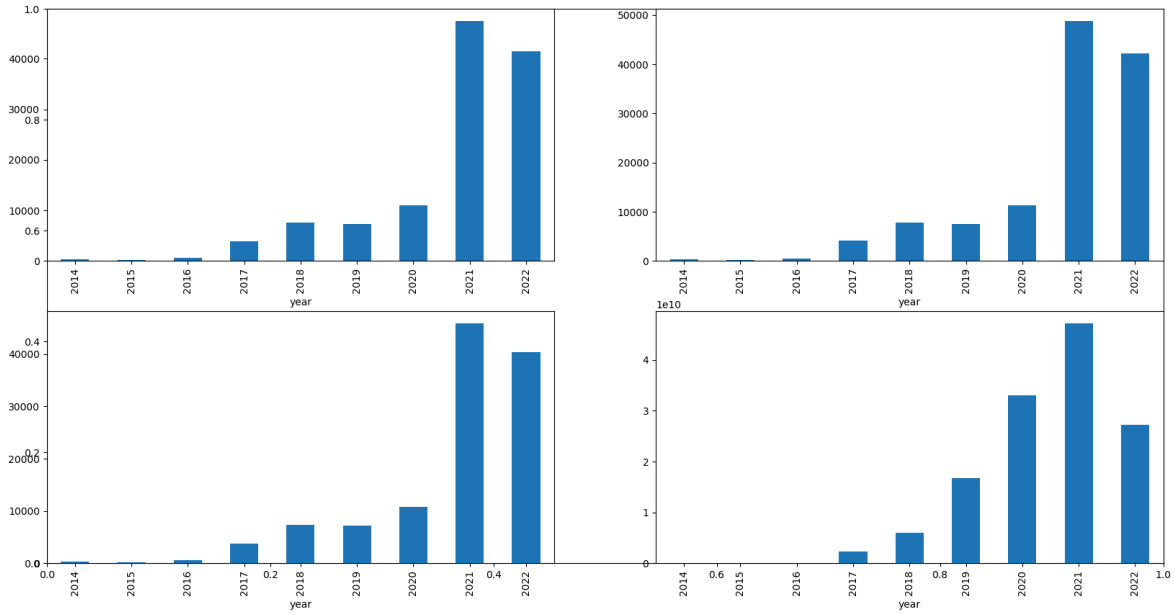
```
Index(['Date', 'Open', 'High', 'Low', 'Adj Close', 'Volume'], dtype='object')
```

In [15]:
```
# Convert the 'timestamp' column to datetime
data['Date'] = pd.to_datetime(data['Date'] )# Use 'ms' if the timestamp is in mi

# Extract year, month, and day from the 'date' column
data['year'] = data['Date'].dt.year
data['month'] = data['Date'].dt.month
data['day'] = data['Date'].dt.day

# Drop the 'timestamp' column (optional)
#data = data.drop('Date', axis=1)

# Display the updated DataFrame
data.head()
```

Out[15]:

	Date	Open	High	Low	Adj Close	Volume	year	month	day
0	2014-09-17	465.864014	468.174011	452.421997	457.334015	21056800	2014	9	17
1	2014-09-18	456.859985	456.859985	413.104004	424.440002	34483200	2014	9	18
2	2014-09-19	424.102997	427.834991	384.532013	394.795990	37919700	2014	9	19
3	2014-09-20	394.673004	423.295990	389.882996	408.903992	36863600	2014	9	20
4	2014-09-21	408.084991	412.425995	393.181000	398.821014	26580100	2014	9	21

In [16]:
```
data_grouped = data.groupby('year').mean()
plt.subplots(figsize=(20,10))
for i, col in enumerate(['Open', 'High', 'Low', 'Volume']):
  plt.subplot(2,2,i+1)
  data_grouped[col].plot.bar()
plt.show()
```



In [17]:
```
data['is_quarter_end'] = np.where(data['month']%3==0,1,0)
data.head()
```

Out[17]:

	Date	Open	High	Low	Adj Close	Volume	year	month	day
0	2014-09-17	465.864014	468.174011	452.421997	457.334015	21056800	2014	9	17
1	2014-09-18	456.859985	456.859985	413.104004	424.440002	34483200	2014	9	18
2	2014-09-19	424.102997	427.834991	384.532013	394.795990	37919700	2014	9	19
3	2014-09-20	394.673004	423.295990	389.882996	408.903992	36863600	2014	9	20
4	2014-09-21	408.084991	412.425995	393.181000	398.821014	26580100	2014	9	21

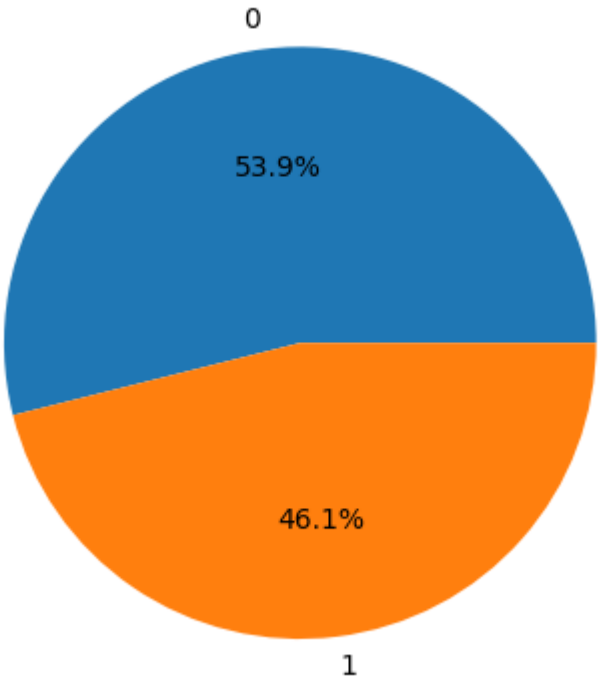
In [18]:
```
print(data.columns)
```

```
Index(['Date', 'Open', 'High', 'Low', 'Adj Close', 'Volume', 'year', 'month',
       'day', 'is_quarter_end'],
      dtype='object')
```

In [19]:
```
 # Ensure the DataFrame is sorted by date (if not already sorted)
data = data.sort_values(by='Date')

# Create the 'low-high' feature (difference between 'low' and 'high')
data['low-high'] = data['Low'] - data['High']

# Optional: Create a target variable based on the 'open' column
data['target'] = np.where(data['Open'].shift(-1) > data['Open'], 1, 0)

# Drop rows with NaN values in 'target'
data = data.dropna(subset=['target'])

# Display the updated DataFrame
print(data.head())
```

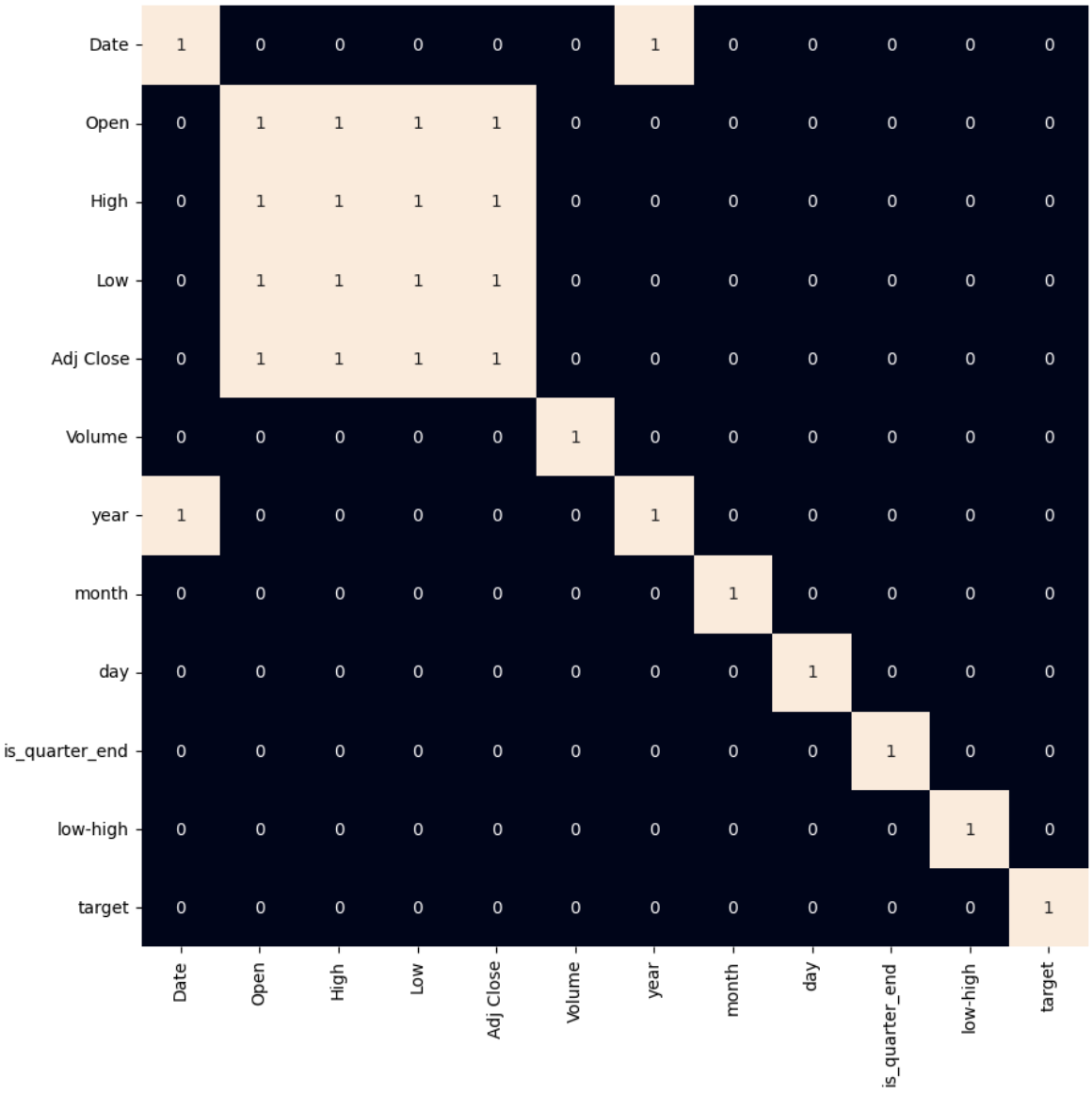
```
        Date        Open        High         Low   Adj Close    Volume  year  \
0 2014-09-17  465.864014  468.174011  452.421997  457.334015  21056800  2014
1 2014-09-18  456.859985  456.859985  413.104004  424.440002  34483200  2014
2 2014-09-19  424.102997  427.834991  384.532013  394.795990  37919700  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   low-high  target
0      9   17               1 -15.752014       0
1      9   18               1 -43.755981       0
2      9   19               1 -43.302978       0
3      9   20               1 -33.412994       1
4      9   21               1 -19.244995       0
```

In [20]:
```
plt.pie(data['target'].value_counts().values,
        labels=[0, 1], autopct='%1.1f%%')
plt.show()
```



In [21]:
```
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(data.corr() > 0.9, annot=True, cbar=False)
plt.show()
```



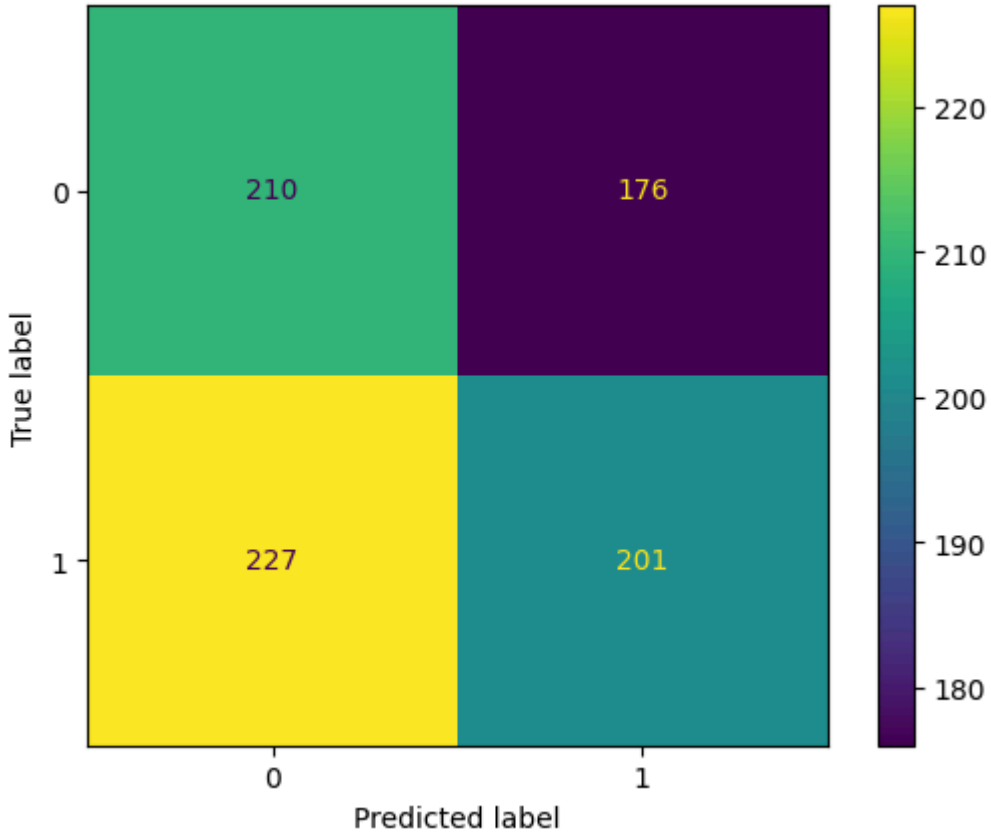
In [22]:
```
# Select features and target
features = data[['low-high', 'is_quarter_end']]  # Removed 'open-close' since it
target = data['target']

# Scale the features
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)

# Split the data into training (first 70%) and validation (last 30%)
split_index = int(0.7 * len(features))  # 70% for training, 30% for testing
X_train, X_valid = features_scaled[:split_index], features_scaled[split_index:]
Y_train, Y_valid = target[:split_index], target[split_index:]

# Print the shapes of the splits
print("Training features shape:", X_train.shape)
print("Validation features shape:", X_valid.shape)
print("Training target shape:", Y_train.shape)
print("Validation target shape:", Y_valid.shape)
```

```
Training features shape: (1899, 2)
Validation features shape: (814, 2)
Training target shape: (1899,)
Validation target shape: (814,)
```

In [23]:
```python
models = [LogisticRegression(), SVC(kernel='poly', probability=True), XGBClassif

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
  print('Validation Accuracy : ', metrics.roc_auc_score(Y_valid, models[i].predi
  print()
```

```
LogisticRegression() :
Training Accuracy :  0.5163317453927357
Validation Accuracy :  0.5234068568108082

SVC(kernel='poly', probability=True) :
Training Accuracy :  0.5116959652889604
Validation Accuracy :  0.530960970413055

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 :  0.7607376095902667
Validation Accuracy :  0.5322411747615128
```

In [24]:
```python
from sklearn.metrics import ConfusionMatrixDisplay

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



In [25]:

```python
# Global variables
model = None
scaler = None
data = None
prophet_model = None  # For time-series forecasting

# Function to validate the date
def validate_date(year, month, day):
    try:
        # Check if the date is valid
        datetime(year=year, month=month, day=day)
        return True
    except ValueError:
        return False

# Function to train and save the model
def train_and_save_model():
    global model, scaler, data, prophet_model
    # Load the dataset
    data = pd.read_csv(r"C:\Users\grmus\Downloads\bitcoin.csv")  # Update the pa

    # Preprocess the data
    data['Date'] = pd.to_datetime(data['Date'])
    data['year'] = data['Date'].dt.year
    data['month'] = data['Date'].dt.month
    data['day'] = data['Date'].dt.day
    data['is_quarter_end'] = np.where(data['month'] % 3 == 0, 1, 0)
    data['low-high'] = data['Low'] - data['High']
    data['target'] = np.where(data['Open'].shift(-1) > data['Open'], 1, 0)
    data = data.dropna(subset=['target'])

    # Select features and target
    features = data[['low-high', 'is_quarter_end']]
```

```
    target = data['target']

    # Scale features
    scaler = StandardScaler()
    features_scaled = scaler.fit_transform(features)

    # Train the model (using Logistic Regression as an example)
    model = LogisticRegression()
    model.fit(features_scaled, target)

    # Save the model and scaler
    joblib.dump(model, 'bitcoin_price_model.pkl')
    joblib.dump(scaler, 'bitcoin_price_scaler.pkl')

    # Train Prophet model for time-series forecasting
    prophet_data = data[['Date', 'Close']].rename(columns={'Date': 'ds', 'Close'
    prophet_model = Prophet()
    prophet_model.fit(prophet_data)

    # Save the Prophet model
    joblib.dump(prophet_model, 'prophet_model.pkl')

    messagebox.showinfo("Success", "Models trained and saved successfully!")

# Function to load the model
def load_model():
    global model, scaler, prophet_model
    try:
        model = joblib.load('bitcoin_price_model.pkl')
        scaler = joblib.load('bitcoin_price_scaler.pkl')
        prophet_model = joblib.load('prophet_model.pkl')  # Load Prophet model
        messagebox.showinfo("Success", "Models loaded successfully!")
    except FileNotFoundError:
        messagebox.showwarning("Error", "Models not found. Please train the mode

# Function to predict Bitcoin price
def predict_price():
    global model, scaler, data, prophet_model
    if model is None or scaler is None or prophet_model is None:
        messagebox.showwarning("Error", "Please load the models first!")
        return

    # Get user input
    try:
        year = int(entry_year.get())
        month = int(entry_month.get())
        day = int(entry_day.get())
    except ValueError:
        messagebox.showwarning("Input Error", "Please enter valid numbers for ye
        return

    # Validate the date
    if not validate_date(year, month, day):
        messagebox.showwarning("Input Error", "Invalid date. Please enter a vali
        return

    # Check if the date is realistic (e.g., year between 2009 and 2030)
    if year < 2009 or year > 2030:
        messagebox.showwarning("Input Error", "Year must be between 2009 and 203
        return
```

```
    # Create a DataFrame for the input
    input_date = pd.to_datetime(f"{year}-{month}-{day}")

    # Use Prophet for future predictions
    if input_date > data['Date'].max():
        try:
            # Calculate the number of days between the last date in the dataset
            last_date = data['Date'].max()
            days_ahead = (input_date - last_date).days

            # Ensure the forecast covers the input date
            if days_ahead <= 0:
                result = "Error: The input date is not in the future."
            else:
                future = prophet_model.make_future_dataframe(periods=days_ahead)
                forecast = prophet_model.predict(future)

                # Check if the input date is in the forecast
                if input_date in forecast['ds'].values:
                    predicted_price = forecast[forecast['ds'] == input_date]['yh
                    result = f"The predicted Bitcoin price on {input_date.date()
                else:
                    result = "Error: The input date is not in the forecast range
        except Exception as e:
            result = f"Error in prediction: {str(e)}"
    else:
        # Use Logistic Regression for historical dates
        input_data = pd.DataFrame({
            'year': [year],
            'month': [month],
            'day': [day],
            'is_quarter_end': [1 if month % 3 == 0 else 0],
            'low-high': [data[data['Date'] == input_date]['Low'].values[0] - dat
        })
        input_scaled = scaler.transform(input_data[['low-high', 'is_quarter_end'
        prediction = model.predict(input_scaled)
        result = "The model predicts that the Bitcoin price will increase." if p

    result_label.config(text=result)

# Create the main window
root = tk.Tk()
root.title("Bitcoin Price Prediction")
root.geometry("800x600")
root.state('zoomed')  # Start in full-screen mode

# Load background image
bg_image_path = (r"C:\Users\grmus\Downloads\pexels-alesiakozik-6770513.jpg")
bg_image = Image.open(bg_image_path)
bg_image = bg_image.resize((root.winfo_screenwidth(), root.winfo_screenheight()))
bg_image = ImageTk.PhotoImage(bg_image)

# Create a canvas for the background
canvas = tk.Canvas(root, width=root.winfo_screenwidth(), height=root.winfo_scree
canvas.pack(fill="both", expand=True)
canvas.create_image(0, 0, image=bg_image, anchor="nw")

# Stylish fonts
title_font = font.Font(family="Helvetica", size=24, weight="bold")
```

```
label_font = font.Font(family="Helvetica", size=16)
button_font = font.Font(family="Helvetica", size=14, weight="bold")
result_font = font.Font(family="Helvetica", size=18, weight="bold")

# Title
canvas.create_text(root.winfo_screenwidth() // 2, 50, text="Bitcoin Price Predic

# Frame for input fields
input_frame = tk.Frame(root, bg="#4CAF50", bd=0)
input_frame.place(relx=0.5, rely=0.4, anchor="center")

# Year Input
label_year = tk.Label(input_frame, text="Year:", font=label_font, bg="#4CAF50",
label_year.grid(row=0, column=0, padx=5, pady=5)
entry_year = tk.Entry(input_frame, font=label_font)
entry_year.grid(row=0, column=1, padx=5, pady=5)

# Month Input
label_month = tk.Label(input_frame, text="Month:", font=label_font, bg="#4CAF50"
label_month.grid(row=1, column=0, padx=5, pady=5)
entry_month = tk.Entry(input_frame, font=label_font)
entry_month.grid(row=1, column=1, padx=5, pady=5)

# Day Input
label_day = tk.Label(input_frame, text="Day:", font=label_font, bg="#4CAF50", fg
label_day.grid(row=2, column=0, padx=5, pady=5)
entry_day = tk.Entry(input_frame, font=label_font)
entry_day.grid(row=2, column=1, padx=5, pady=5)

# Train and Save Model Button
train_button = tk.Button(root, text="Train and Save Model", font=button_font, bg
train_button.place(relx=0.3, rely=0.6, anchor="center")

# Load Model Button
load_button = tk.Button(root, text="Load Model", font=button_font, bg="#2196F3",
load_button.place(relx=0.5, rely=0.6, anchor="center")

# Predict Button
predict_button = tk.Button(root, text="Predict Price", font=button_font, bg="#FF
predict_button.place(relx=0.7, rely=0.6, anchor="center")

# Result Label
result_label = tk.Label(root, text="", font=result_font, bg="#4CAF50", fg="gold"
result_label.place(relx=0.5, rely=0.8, anchor="center")

# Run the application
root.mainloop()
```

```
09:44:17 - cmdstanpy - INFO - Chain [1] start processing
09:44:19 - cmdstanpy - INFO - Chain [1] done processing
```