```
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('/content/TSLA.csv')
df.head()
```

| | Date | Open | High | Low | Close | Adj Close | Volume |
|---|---|---|---|---|---|---|---|
| 0 | 2010-06-29 | 19.000000 | 25.00 | 17.540001 | 23.889999 | 23.889999 | 18766300 |
| 1 | 2010-06-30 | 25.790001 | 30.42 | 23.299999 | 23.830000 | 23.830000 | 17187100 |
| 2 | 2010-07-01 | 25.000000 | 25.92 | 20.270000 | 21.959999 | 21.959999 | 8218800 |
| 3 | 2010-07-02 | 23.000000 | 23.10 | 18.709999 | 19.200001 | 19.200001 | 5139800 |
| 4 | 2010-07-06 | 20.000000 | 20.00 | 15.830000 | 16.110001 | 16.110001 | 6866900 |

```
df.shape
```

```
(2416, 7)
```

```
df.describe()
```

| | Open | High | Low | Close | Adj Close | Volume |
|---|---|---|---|---|---|---|
| count | 2416.000000 | 2416.000000 | 2416.000000 | 2416.000000 | 2416.000000 | 2.416000e+03 |
| mean | 186.271147 | 189.578224 | 182.916639 | 186.403651 | 186.403651 | 5.572722e+06 |
| std | 118.740163 | 120.892329 | 116.857591 | 119.136020 | 119.136020 | 4.987809e+06 |
| min | 16.139999 | 16.629999 | 14.980000 | 15.800000 | 15.800000 | 1.185000e+05 |
| 25% | 34.342498 | 34.897501 | 33.587501 | 34.400002 | 34.400002 | 1.899275e+06 |
| 50% | 213.035004 | 216.745002 | 208.870002 | 212.960007 | 212.960007 | 4.578400e+06 |
| 75% | 266.450012 | 270.927513 | 262.102501 | 266.774994 | 266.774994 | 7.361150e+06 |
| max | 673.690002 | 786.140015 | 673.520020 | 780.000000 | 780.000000 | 4.706500e+07 |

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2416 entries, 0 to 2415
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Date       2416 non-null   object 
 1   Open       2416 non-null   float64
 2   High       2416 non-null   float64
 3   Low        2416 non-null   float64
 4   Close      2416 non-null   float64
 5   Adj Close  2416 non-null   float64
 6   Volume     2416 non-null   int64  
dtypes: float64(5), int64(1), object(1)
memory usage: 132.2+ KB
```

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


```
df.head()
```

| | Date | Open | High | Low | Close | Adj Close | Volume |
|---|---|---|---|---|---|---|---|
| 0 | 2010-06-29 | 19.000000 | 25.00 | 17.540001 | 23.889999 | 23.889999 | 18766300 |
| 1 | 2010-06-30 | 25.790001 | 30.42 | 23.299999 | 23.830000 | 23.830000 | 17187100 |
| 2 | 2010-07-01 | 25.000000 | 25.92 | 20.270000 | 21.959999 | 21.959999 | 8218800 |
| 3 | 2010-07-02 | 23.000000 | 23.10 | 18.709999 | 19.200001 | 19.200001 | 5139800 |
| 4 | 2010-07-06 | 20.000000 | 20.00 | 15.830000 | 16.110001 | 16.110001 | 6866900 |

```
df[df['Close'] == df['Adj Close']].shape
```

```
(2416, 7)
```

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

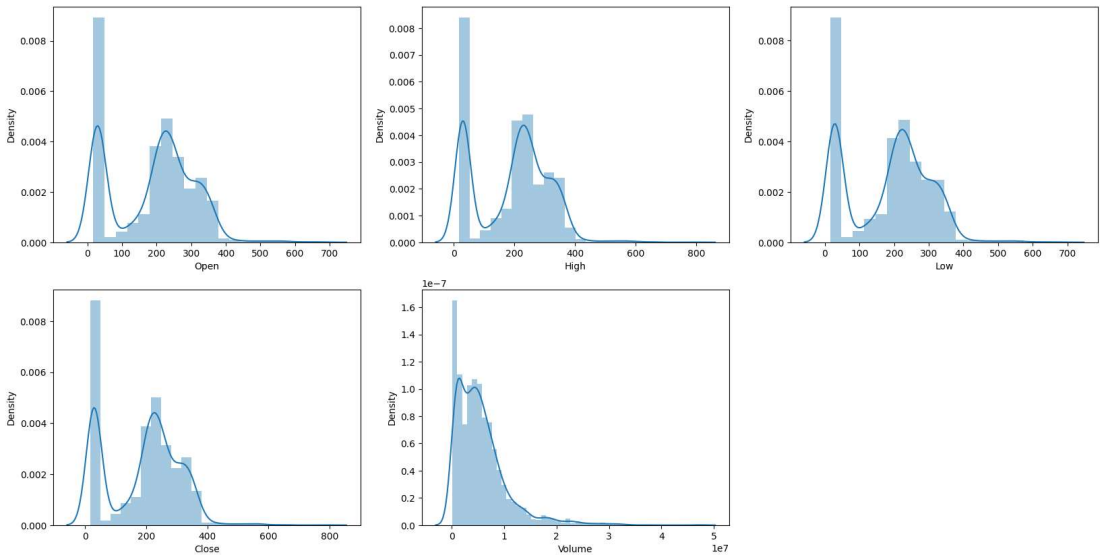
```
df.isnull().sum()
```

```
Date      0
Open      0
High      0
Low       0
Close     0
Volume    0
dtype: int64
```

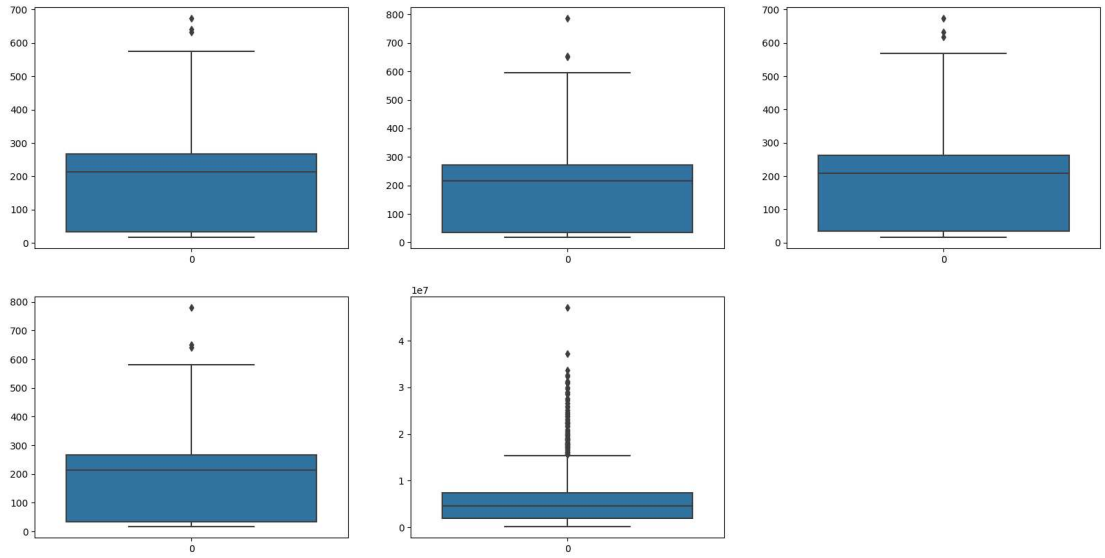
```
features = ['Open', 'High', 'Low', 'Close', 'Volume']

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

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




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



```
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()
```

| | Date | Open | High | Low | Close | Volume | day | month | year |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 2010-06-29 | 19.000000 | 25.00 | 17.540001 | 23.889999 | 18766300 | 6 | 2010 | 29 |
| 1 | 2010-06-30 | 25.790001 | 30.42 | 23.299999 | 23.830000 | 17187100 | 6 | 2010 | 30 |
| 2 | 2010-07-01 | 25.000000 | 25.92 | 20.270000 | 21.959999 | 8218800 | 7 | 2010 | 1 |
| 3 | 2010-07-02 | 23.000000 | 23.10 | 18.709999 | 19.200001 | 5139800 | 7 | 2010 | 2 |
| 4 | 2010-07-06 | 20.000000 | 20.00 | 15.830000 | 16.110001 | 6866900 | 7 | 2010 | 6 |

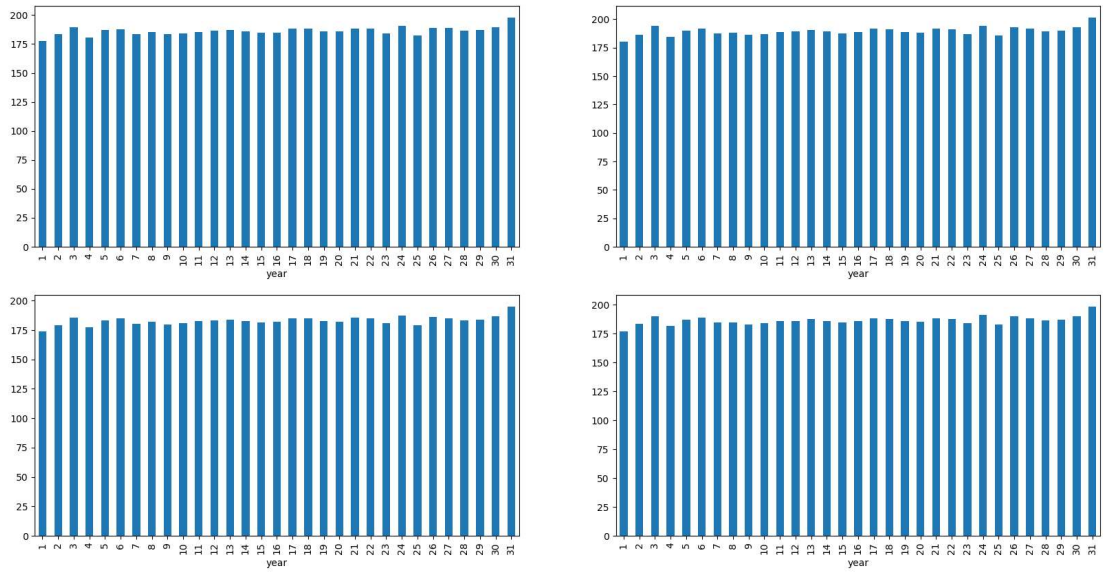
```
df['is_quarter_end'] = np.where(df['month']%3==0,1,0)
df.head()
```

| | Date | Open | High | Low | Close | Volume | day | month | year | is_quarter_e |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2010-06-29 | 19.000000 | 25.00 | 17.540001 | 23.889999 | 18766300 | 6 | 2010 | 29 | |
| 1 | 2010-06-30 | 25.790001 | 30.42 | 23.299999 | 23.830000 | 17187100 | 6 | 2010 | 30 | |
| 2 | 2010-07-01 | 25.000000 | 25.92 | 20.270000 | 21.959999 | 8218800 | 7 | 2010 | 1 | |

```
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()
```

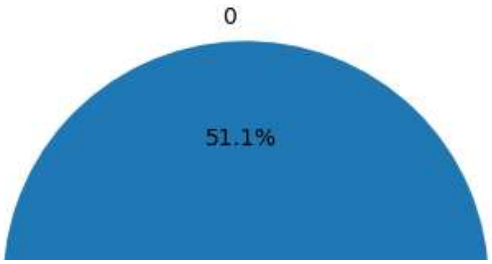



```
df.groupby('is_quarter_end').mean()
```

| is_quarter_end | Open | High | Low | Close | Volume | day | |
|---|---|---|---|---|---|---|---|
| 0 | 195.383850 | 198.830719 | 191.934386 | 195.566477 | 5.012534e+06 | 6.455556 | 20 |
| 1 | 170.534763 | 173.600440 | 167.344233 | 170.580711 | 6.540089e+06 | 6.972912 | 20 |

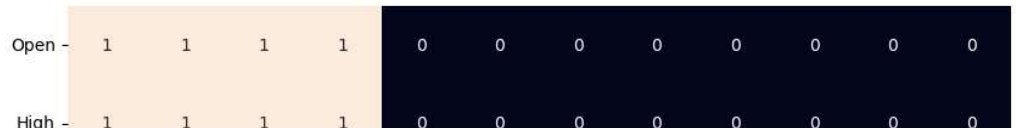
```
df['open-close'] = df['Open'] - df['Close']
df['low-high'] = df['Low'] - df['High']
df['target'] = np.where(df['Close'].shift(-1) > df['Close'], 1, 0)
```

```
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)

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)
```

```
(2174, 3) (242, 3)
```

```
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 :  0.5089337346592329
Validation Accuracy :  0.5124521072796935

SVC(kernel='poly', probability=True) :
Training Accuracy :  0.5165199795103654
Validation Accuracy :  0.4817323481116584

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 :  0.9365469039061541
Validation Accuracy :  0.45583607006020804
```

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

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-34-e4a3478fce11> in <cell line: 1>()
----> 1 metrics.plot_confusion_matrix(models[0], X_valid, Y_valid)
      2 plt.show()
```

```
"""AttributeError: module 'sklearn.metrics' has no attribute 'plot_confusion_matrix'
it is been deprecated from sklearn"""
```

```
'AttributeError: module 'sklearn.metrics' has no attribute 'plot_confusion_matrix'\nit
is been deprecated from sklearn'
```

```
"""Conclusion:
We can observe that the accuracy achieved by the state-of-the-art ML model is no better than simply guessing with a

I have tried to make it more better """
```

```
'Conclusion:\nWe can observe that the accuracy achieved by the state-of-the-art ML mode
l is no better than simply guessing with a probability of 50%. Possible reasons for thi
s may be the lack of data or using a very simple model to perform such a complex task a
s Stock Market prediction.\n\nI have tried to make it more better '
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from xgboost import XGBClassifier
from sklearn import metrics

# Load your dataset
df = pd.read_csv('/content/TSLA.csv')

# Data Preprocessing
# ... (You can include feature engineering, data cleaning, etc.)

# Feature Scaling
scaler = StandardScaler()
features = scaler.fit_transform(features)

# Data Splitting
X_train, X_valid, Y_train, Y_valid = train_test_split(features, target, test_size=0.1, random_state=2022)

# Model Development
model = XGBClassifier(n_estimators=100, max_depth=3, learning_rate=0.1)

model.fit(X_train, Y_train)

# Evaluation
train_accuracy = metrics.roc_auc_score(Y_train, model.predict_proba(X_train)[:, 1])
valid_accuracy = metrics.roc_auc_score(Y_valid, model.predict_proba(X_valid)[:, 1])

print(f'Training Accuracy: {train_accuracy}')
print(f'Validation Accuracy: {valid_accuracy}')
```

```
Training Accuracy: 0.710399759543132
Validation Accuracy: 0.504447181171319
```

```
df.head(5)
```

| | Date | Open | High | Low | Close | Volume | open-close | low-high | target |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 2010-06-29 | 19.000000 | 25.00 | 17.540001 | 23.889999 | 18766300 | -4.889999 | -7.459999 | 0 |
| 1 | 2010-06-30 | 25.790001 | 30.42 | 23.299999 | 23.830000 | 17187100 | 1.960001 | -7.120001 | 0 |
| 2 | 2010- | 25.000000 | 25.92 | 20.270000 | 21.959999 | 8218800 | 3.040001 | -5.650000 | 0 |

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from xgboost import XGBClassifier
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV

# Load your dataset, perform initial data preprocessing, and feature engineering
# Replace this with your data loading and preprocessing steps
df = pd.read_csv('/content/TSLA.csv')
df = df.drop(['Adj Close'], axis=1)

# Feature engineering
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)

# Split the data into features and target
features = df[['open-close', 'low-high']]
target = df['target']

# Normalize the features
scaler = StandardScaler()
features = scaler.fit_transform(features)

# Split the data into training and validation sets
X_train, X_valid, Y_train, Y_valid = train_test_split(
    features, target, test_size=0.1, random_state=2022)

# Hyperparameter tuning using GridSearchCV
param_grid = {
    'max_depth': [3, 5, 7],
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.2]
}

xgb = XGBClassifier()
grid_search = GridSearchCV(xgb, param_grid, cv=3)
grid_search.fit(X_train, Y_train)
best_xgb = grid_search.best_estimator_

# Ensemble multiple models
models = [best_xgb, RandomForestClassifier(), GradientBoostingClassifier()]

for model in models:
    model.fit(X_train, Y_train)
    print(f'{model} : ')
    train_accuracy = metrics.roc_auc_score(Y_train, model.predict_proba(X_train)[:, 1])
    valid_accuracy = metrics.roc_auc_score(Y_valid, model.predict_proba(X_valid)[:, 1])
```

```
    print(f'Training Accuracy: {train_accuracy}')
    print(f'Validation Accuracy: {valid_accuracy}')
    print()

# Combine predictions from multiple models using majority voting or averaging

# Evaluate the ensemble model

# Continue optimizing and experimenting with different techniques to improve accuracy
```

```
    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=0.01, max_bin=None,
                  max_cat_threshold=None, max_cat_to_onehot=None,
                  max_delta_step=None, max_depth=5, max_leaves=None,
                  min_child_weight=None, missing=nan, monotone_constraints=None,
                  multi_strategy=None, n_estimators=50, n_jobs=None,
                  num_parallel_tree=None, random_state=None, ...) :
    Training Accuracy: 0.6684091322809111
    Validation Accuracy: 0.45740968801313625

    RandomForestClassifier() :
    Training Accuracy: 0.9999995766604435
    Validation Accuracy: 0.43045292829775594

    GradientBoostingClassifier() :
    Training Accuracy: 0.772182781086882
    Validation Accuracy: 0.4714354132457581
```

```
from sklearn.metrics import plot_confusion_matrix
import matplotlib.pyplot as plt

# Replace 'best_xgb' with your chosen model
plot_confusion_matrix(best_xgb, X_valid, Y_valid)
plt.show()
```

```
    ---------------------------------------------------------------------------
    ImportError                               Traceback (most recent call last)
    <ipython-input-49-c48be53b5521> in <cell line: 1>()
    ----> 1 from sklearn.metrics import plot_confusion_matrix
          2 import matplotlib.pyplot as plt
          3
          4 # Replace 'best_xgb' with your chosen model
          5 plot_confusion_matrix(best_xgb, X_valid, Y_valid)

    ImportError: cannot import name 'plot_confusion_matrix' from 'sklearn.metrics'
    (/usr/local/lib/python3.10/dist-packages/sklearn/metrics/__init__.py)

    ---------------------------------------------------------------------------
    NOTE: If your import is failing due to a missing package, you can
    manually install dependencies using either !pip or !apt.

    To view examples of installing some common dependencies, click the
    "Open Examples" button below.
    ---------------------------------------------------------------------------
```

OPEN EXAMPLES SEARCH STACK OVERFLOW

```
pip install -U scikit-learn
```

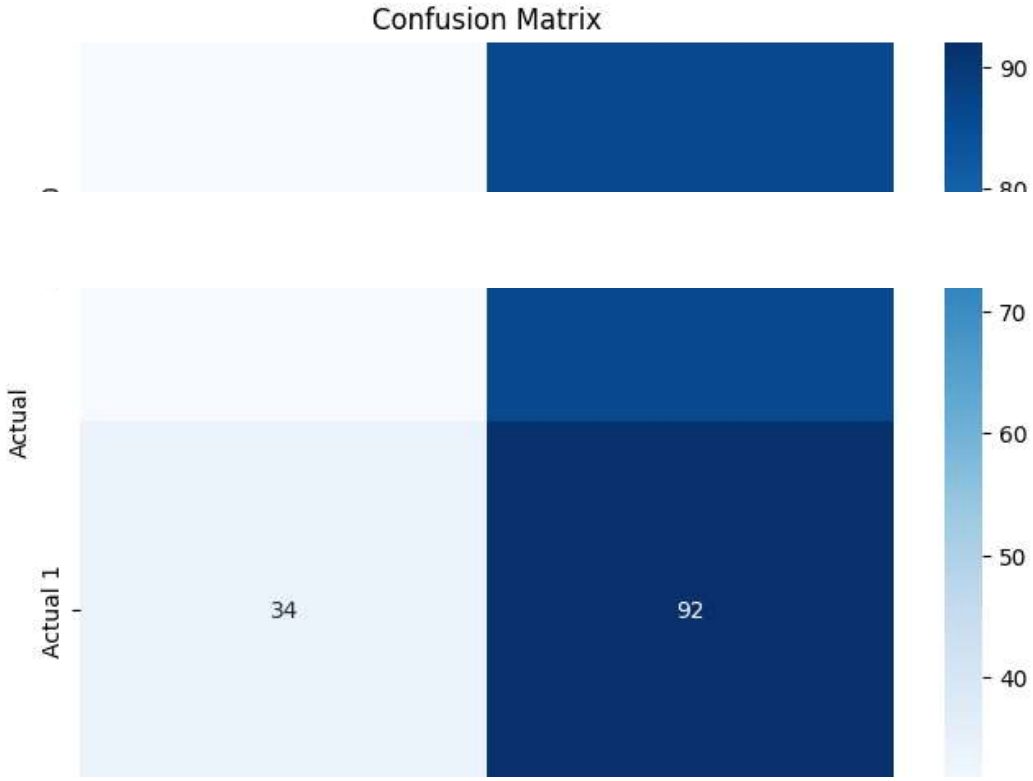
```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (
Collecting scikit-learn
  Downloading scikit_learn-1.3.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10.8/10.8 MB 77.7 MB/s eta 0:00:00
Requirement already satisfied: numpy<2.0,>=1.17.3 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: scipy>=1.5.0 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-pa
Installing collected packages: scikit-learn
  Attempting uninstall: scikit-learn
    Found existing installation: scikit-learn 1.2.2
    Uninstalling scikit-learn-1.2.2:
      Successfully uninstalled scikit-learn-1.2.2
Successfully installed scikit-learn-1.3.2
WARNING: The following packages were previously imported in this runtime:
  [sklearn]
You must restart the runtime in order to use newly installed versions.
```

RESTART RUNTIME

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Replace 'best_xgb' with your chosen model
y_pred = best_xgb.predict(X_valid)
cm = confusion_matrix(Y_valid, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Predicted 0", "Predicted 1"], yticklabels=["Actual
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```


Confusion Matrix
Actual
Actual 1
34
92
90
80
70
60
50
40