Final_Project_Test_Dtrigg

September 29, 2024

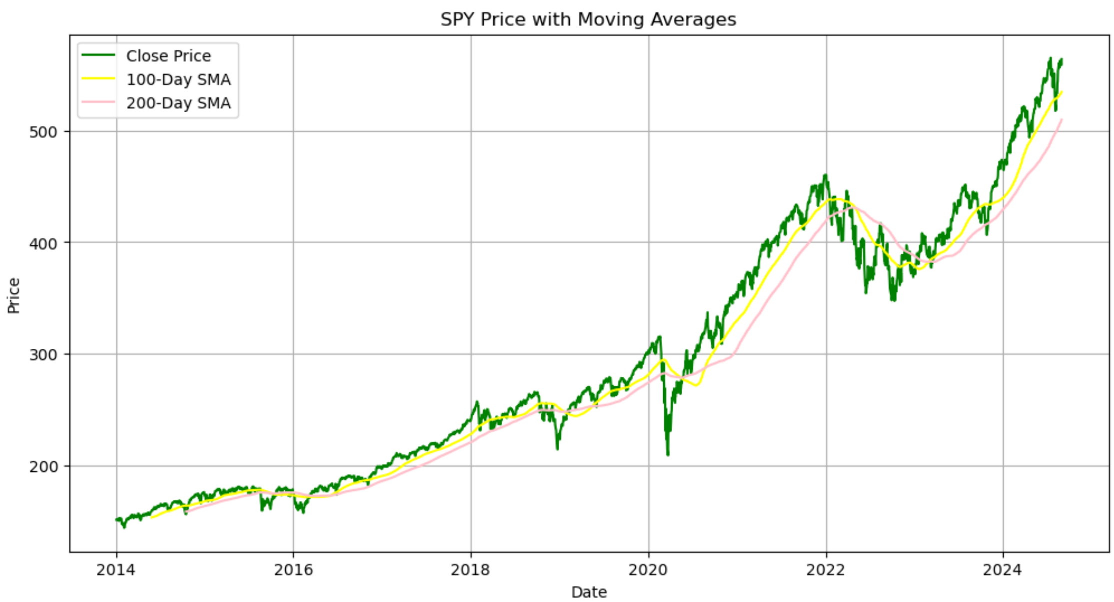
[1]:
```
# Import necessary libraries - common libraries include pandas, numpy,␣
↪matplotlib, and sklearn
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import gridspec
import math
import scipy.stats
from scipy.stats import dgamma
from sklearn.linear_model import LinearRegression
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
```

[2]:
```
# Read in the Carbon West data file
spy = pd.read_csv('spy_since_2014.csv')

# View the first few rows of the dataset
spy.head()
```

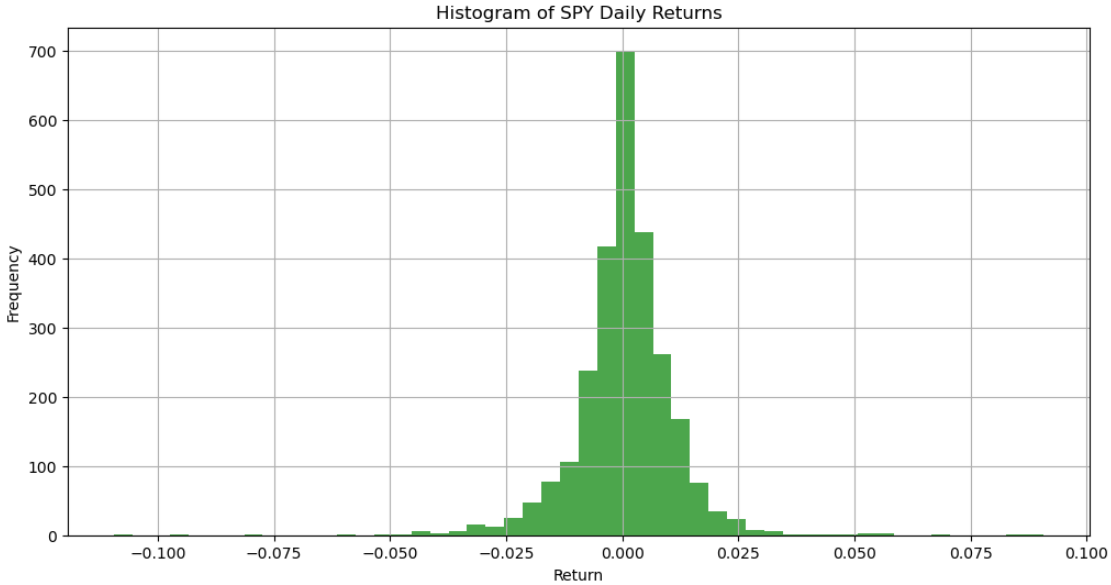
[2]:
```
       Date        Open        High         Low       Close     Volume  Day  \
0  1/2/2014  152.585939  152.660591  151.341897  151.706818  119636900    2
1  1/3/2014  151.963926  152.270798  151.466316  151.681946   81390600    3
2  1/6/2014  152.179553  152.237602  151.010150  151.242371  108028200    6
3  1/7/2014  151.847768  152.428319  151.731658  152.171219   86144200    7
4  1/8/2014  152.146406  152.461568  151.681966  152.204468   96582300    8

   Weekday  Week  Month  Year
0        3     1      1  2014
1        4     1      1  2014
2        0     2      1  2014
3        1     2      1  2014
4        2     2      1  2014
```

[33]:
```
# set Date as the index
spy['Date'] = pd.to_datetime(spy['Date'])  # Convert to datetime if not already
spy.set_index('Date', inplace=True)  # Set 'Date' as the index
# Calculate the moving average
spy['SMA_100'] = spy['Close'].rolling(window=100).mean()
```

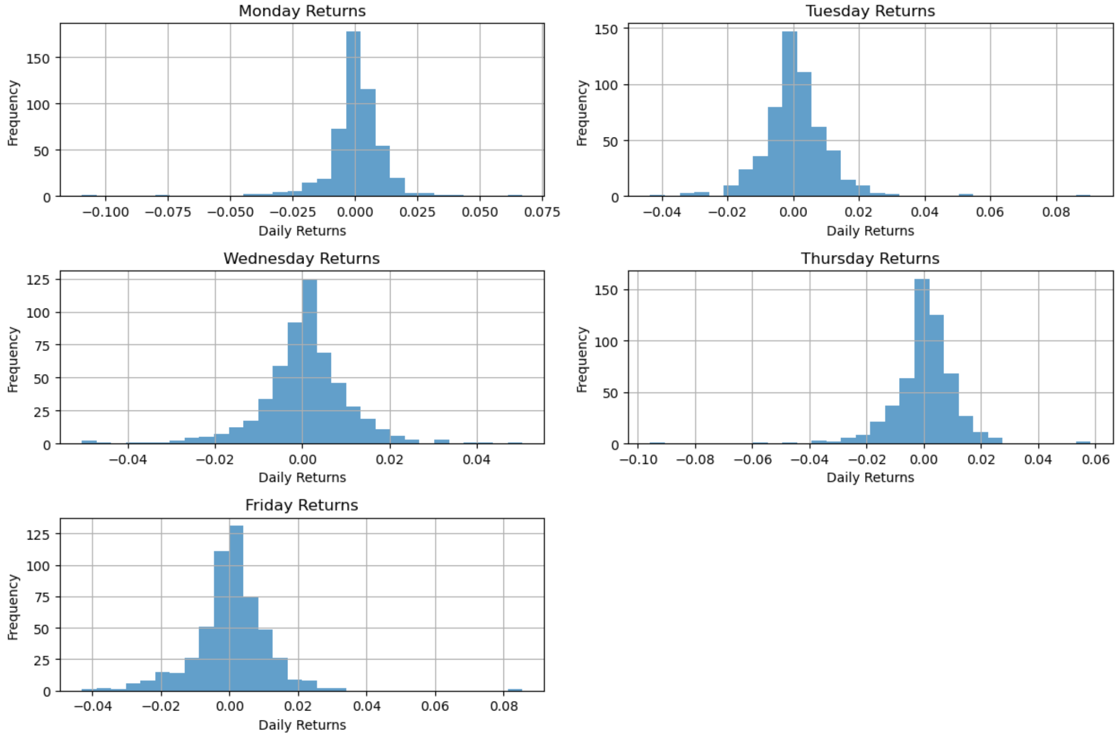
```
spy['SMA_200'] = spy['Close'].rolling(window=200).mean()

# plot spy performance since 2014
plt.figure(figsize=(12, 6))
plt.plot(spy['Close'], label='Close Price', color='green')
plt.plot(spy['SMA_100'], label='100-Day SMA', color='yellow')
plt.plot(spy['SMA_200'], label='200-Day SMA', color='pink')
plt.title('SPY Price with Moving Averages')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.grid()
plt.show()
```



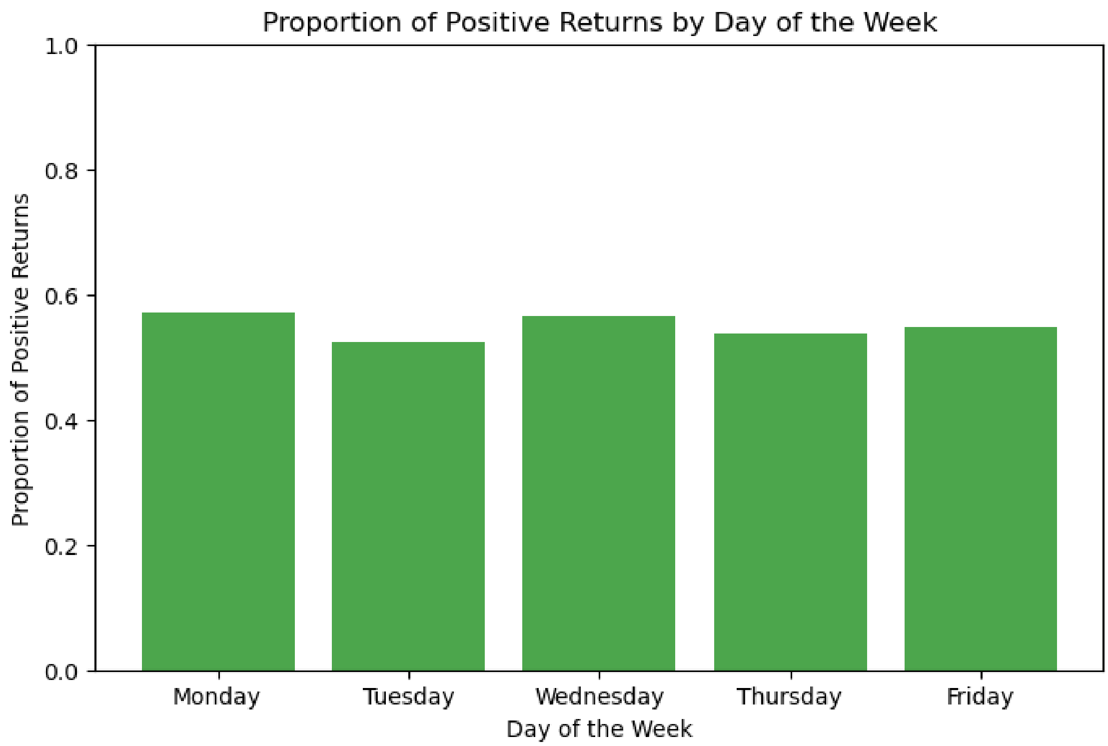
[3]:
```
# plot distribution of returns
spy['Returns'] = spy['Close'].pct_change()

plt.figure(figsize=(12, 6))
plt.hist(spy['Returns'].dropna(), bins=50, color='green', alpha=0.7)
plt.title('Histogram of SPY Daily Returns')
plt.xlabel('Return')
plt.ylabel('Frequency')
plt.grid()
plt.show()
```



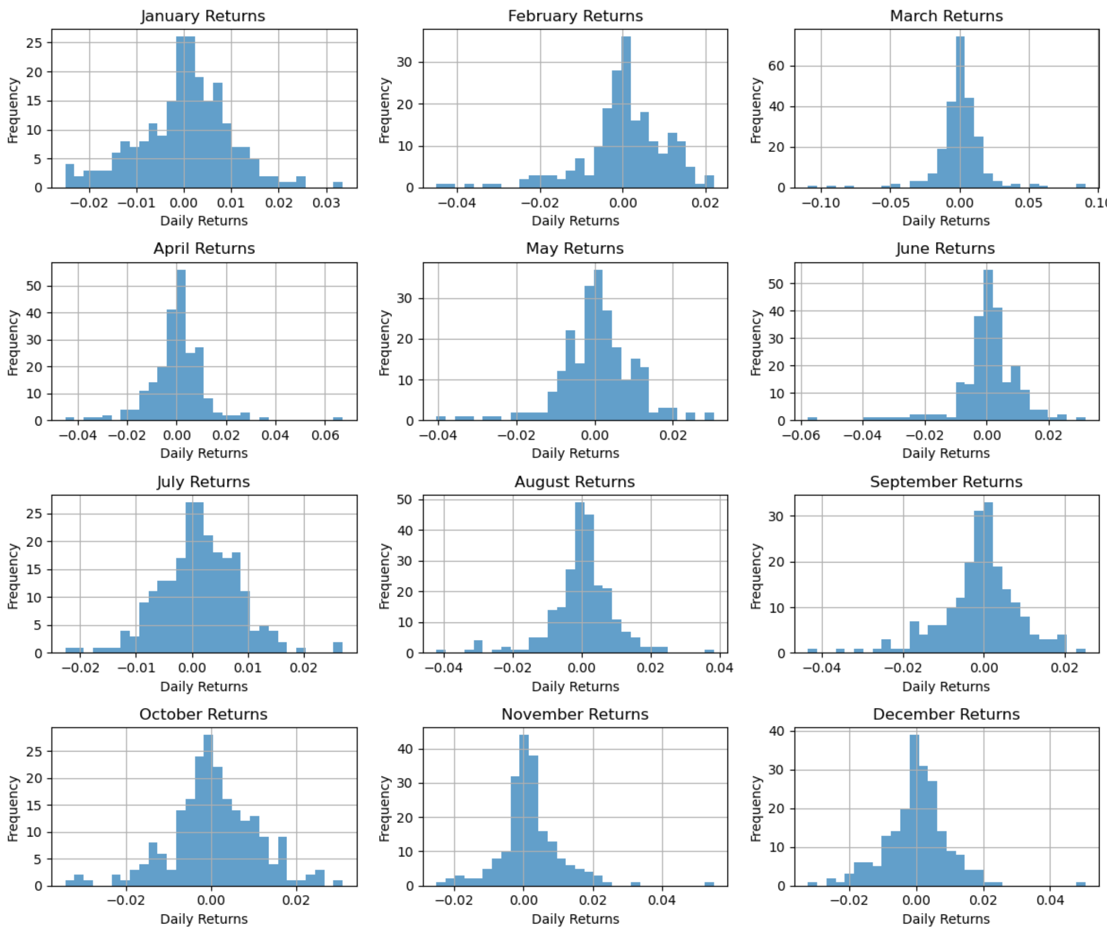
[4]:
```python
# Step 1: Calculate daily returns (if not already calculated)
spy['Daily_Returns'] = spy['Close'].pct_change()

# Step 2: Plot histograms for each day of the week using the existing 'Weekday'␣
↪column (0 to 5)
# 0 = Monday, 5 = Friday
days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']

plt.figure(figsize=(12, 8))
for i, day in enumerate(days):
    plt.subplot(3, 2, i+1)  # Create a subplot for each day
    spy[spy['Weekday'] == i]['Daily_Returns'].hist(bins=30, alpha=0.7)
    plt.title(f'{day} Returns')
    plt.xlabel('Daily Returns')
    plt.ylabel('Frequency')

plt.tight_layout()  # Adjust layout to prevent overlap
plt.show()
```



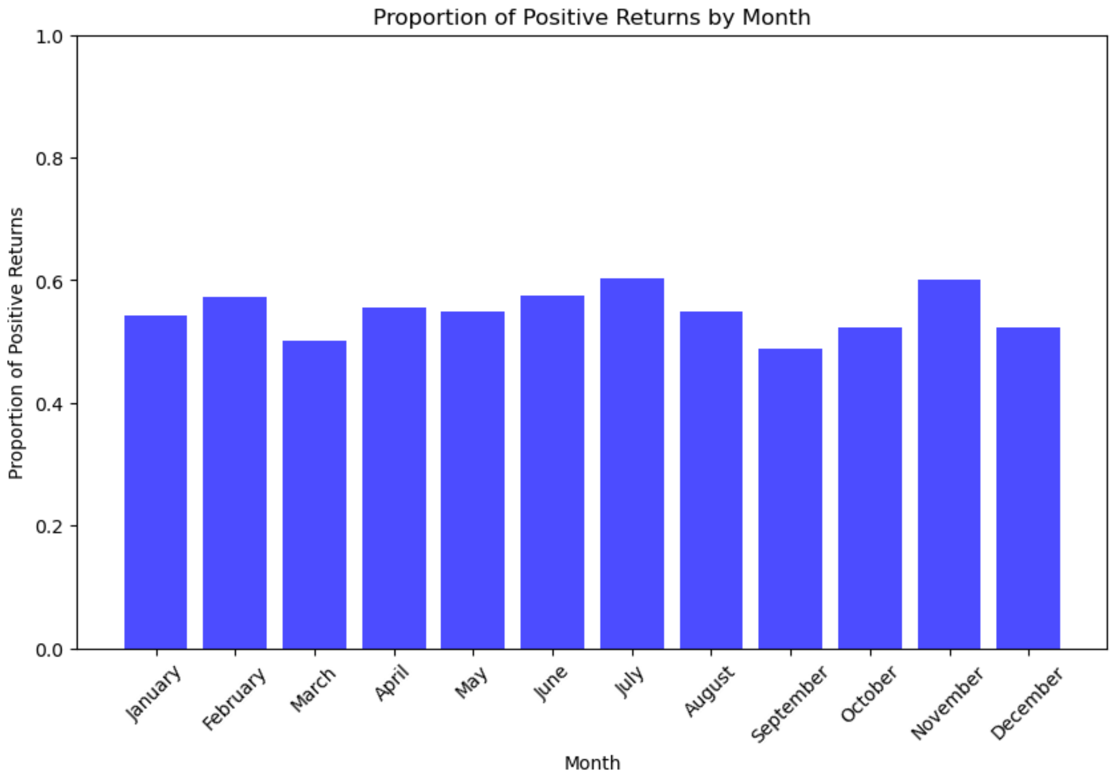
[5]:
```python
# Step 1: Calculate daily returns (if not already calculated)
spy['Daily_Returns'] = spy['Close'].pct_change()

# Step 2: Categorize returns as positive (1) or negative (0)
spy['Positive_Return'] = spy['Daily_Returns'] > 0

# Step 3: Calculate the proportion of positive returns for each day of the week using the existing 'Weekday' column
proportion_positive = spy.groupby('Weekday')['Positive_Return'].mean()

# Step 4: Plot the proportions
days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']

plt.figure(figsize=(8, 5))
plt.bar(days, proportion_positive, color='green', alpha=0.7)
plt.title('Proportion of Positive Returns by Day of the Week')
plt.xlabel('Day of the Week')
plt.ylabel('Proportion of Positive Returns')
plt.ylim(0, 1)  # Proportions are between 0 and 1
plt.show()
```



[6]:
```
# Step 1: Calculate daily returns
spy['Daily_Returns'] = spy['Close'].pct_change()

# Step 3: Plot histograms of daily returns by month
months = ['January', 'February', 'March', 'April', 'May', 'June',
          'July', 'August', 'September', 'October', 'November', 'December']

plt.figure(figsize=(12, 10))
for i, month in enumerate(months):
    plt.subplot(4, 3, i+1)  # Create a subplot for each month
    spy[spy['Month'] == (i+1)]['Daily_Returns'].hist(bins=30, alpha=0.7)
    plt.title(f'{month} Returns')
    plt.xlabel('Daily Returns')
    plt.ylabel('Frequency')

plt.tight_layout()  # Adjust layout to prevent overlap
plt.show()
```

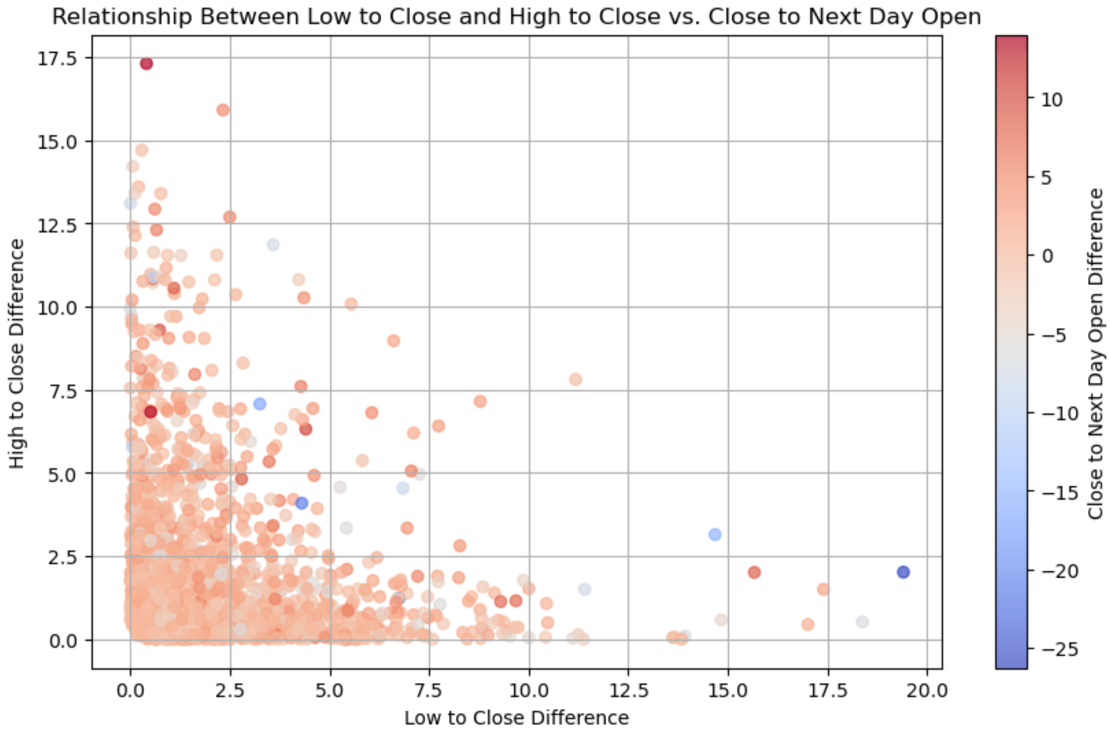


[7]:
```python
# Step 4: Categorize returns as positive (1) or negative (0)
spy['Positive_Return'] = spy['Daily_Returns'] > 0

# Step 5: Calculate the proportion of positive returns for each month
proportion_positive_month = spy.groupby('Month')['Positive_Return'].mean()

# Step 6: Plot the proportion of positive returns by month
plt.figure(figsize=(10, 6))
plt.bar(months, proportion_positive_month, color='blue', alpha=0.7)
plt.title('Proportion of Positive Returns by Month')
plt.xlabel('Month')
plt.ylabel('Proportion of Positive Returns')
plt.ylim(0, 1)  # Proportions are between 0 and 1
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.show()
```

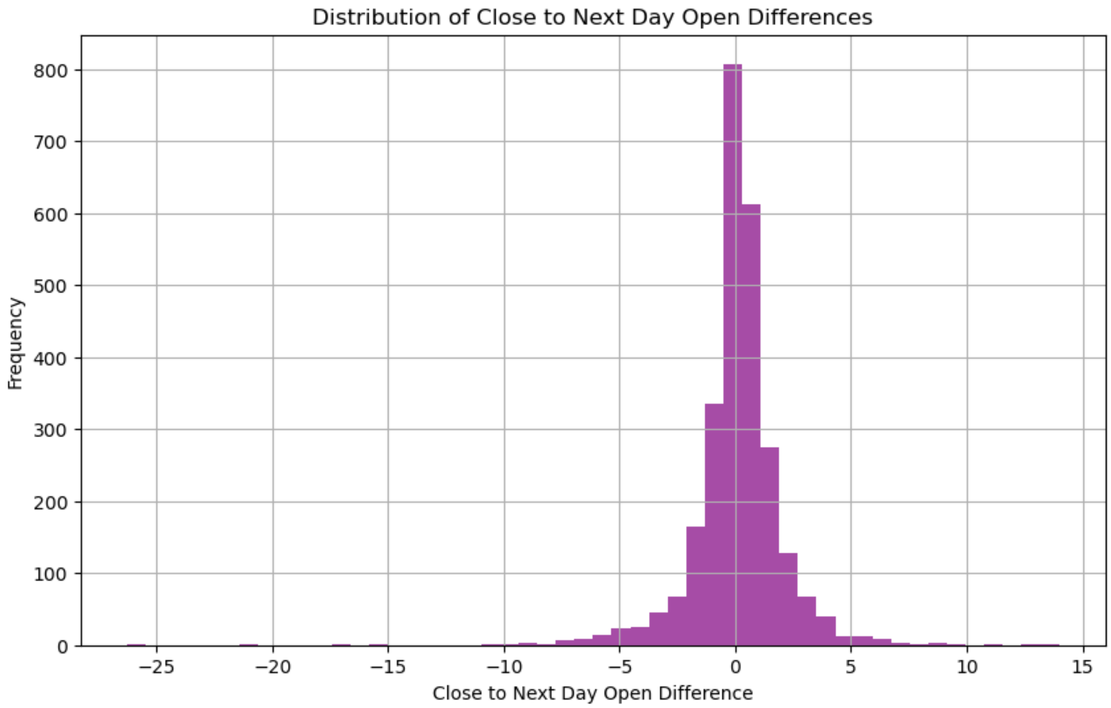



```
[9]: # Step 1: Calculate the necessary columns
spy['Low_to_Close'] = spy['Close'] - spy['Low']
spy['High_to_Close'] = spy['High'] - spy['Close']
spy['Close_to_Next_Open'] = spy['Open'].shift(-1) - spy['Close']  # Shift for␣
↪the next day's open

# Step 2: Plot the scatter plot
plt.figure(figsize=(10, 6))

# Scatter plot where color is based on Close to Next Open difference
scatter = plt.scatter(spy['Low_to_Close'], spy['High_to_Close'],␣
↪c=spy['Close_to_Next_Open'], cmap='coolwarm', alpha=0.7)

# Adding color bar
plt.colorbar(scatter, label='Close to Next Day Open Difference')

# Plot details
plt.title('Relationship Between Low to Close and High to Close vs. Close to␣
↪Next Day Open')
plt.xlabel('Low to Close Difference')
plt.ylabel('High to Close Difference')
plt.grid(True)
```

```
plt.show()
```



[10]:
```
# Step 1: Correlation analysis
correlation_matrix = spy[['Low_to_Close', 'High_to_Close',␣
↪'Close_to_Next_Open']].corr()

# Step 2: Visualize the distribution of Close to Next Day Open differences
plt.figure(figsize=(10, 6))
plt.hist(spy['Close_to_Next_Open'].dropna(), bins=50, alpha=0.7, color='purple')
plt.title('Distribution of Close to Next Day Open Differences')
plt.xlabel('Close to Next Day Open Difference')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()

# Output the correlation matrix
correlation_matrix
```



[10]:

	Low_to_Close	High_to_Close	Close_to_Next_Open
Low_to_Close	1.000000	-0.088490	-0.078369
High_to_Close	-0.088490	1.000000	0.055175
Close_to_Next_Open	-0.078369	0.055175	1.000000

[11]:

```
# Step 1: Create columns for yesterday's high and low
spy['Yesterday_High'] = spy['High'].shift(1)
spy['Yesterday_Low'] = spy['Low'].shift(1)

# Step 2: Create conditions
condition_high = spy['High'] > spy['Yesterday_High']  # Today's high >␣
↪yesterday's high
condition_low = spy['Low'] > spy['Yesterday_Low']     # Today's low >␣
↪yesterday's low

# Step 3: Check if close is higher than open under both conditions
spy['Close_Higher_Than_Open'] = spy['Close'] > spy['Open']
condition_combined = condition_high & condition_low

# Step 4: Calculate the proportion of days where the close is higher than the␣
↪open under these conditions
proportion = spy[condition_combined]['Close_Higher_Than_Open'].mean()

# Output the result
```

```
print(f"Proportion of days where Close is higher than Open when both High and Low are higher than yesterday: {proportion:.2%}")
```

Proportion of days where Close is higher than Open when both High and Low are higher than yesterday: 71.04%

[12]:
```
# Step 1: Filter the dataset where both conditions (high > yesterday's high and low > yesterday's low) are true
valid_days = spy[condition_combined]

# Step 2: Count the number of days where the close price is higher than the open price
num_days = valid_days['Close_Higher_Than_Open'].sum()  # Sum the True values (True = 1, False = 0)

# Step 3: Count the total number of valid days where both conditions (high and low higher than previous day) are true
total_valid_days = valid_days.shape[0]

# Step 4: Calculate the success rate
success_rate = num_days / total_valid_days * 100

# Output the results
num_days, total_valid_days, success_rate
```

[12]: (871, 1226, 71.04404567699837)

[13]:
```
# Calculate the total number of days in the dataset
total_days_in_dataset = spy.shape[0]

# Output the total number of days
total_days_in_dataset
```

[13]: 2684

[14]:
```
# calculate how often higher highs and higher lows occur
scenario_1 = total_valid_days / total_days_in_dataset
print(scenario_1)
```

0.45678092399403875

[15]:
```
# Step 1: Shift the condition_combined to create a condition for the previous day's close
condition_combined_shifted = condition_combined.shift(1)

# Step 2: Check if SPY closes higher than open on the next day when the previous day's condition was met
```

```
next_day_higher_close = spy['Close_Higher_Than_Open'] &␣
 ↪condition_combined_shifted

# Step 3: Calculate the proportion of days where SPY closes higher than open on␣
 ↪the next day
next_day_proportion = next_day_higher_close.mean()

# Output the result
next_day_proportion
```

[15]: 0.2451564828614009

[16]:
```
# Step 1: Check if both conditions are met and SPY closes higher than open
spy['Both_Conditions_Met'] = condition_combined & spy['Close_Higher_Than_Open']

# Step 2: Create a condition where the next day's open is higher than today's␣
 ↪close
next_day_open_higher = spy['Open'].shift(-1) > spy['Close']

# Step 3: Combine the condition with "Both_Conditions_Met"
next_day_open_higher_with_conditions = next_day_open_higher &␣
 ↪spy['Both_Conditions_Met'].shift(1)

# Step 4: Check if the next day closes lower than the next day's open
next_day_close_lower_than_high = spy['Close'].shift(-1) < spy['Open'].shift(-1)

# Step 5: Calculate the proportion of days where the next day closes lower than␣
 ↪the next day's high
next_day_proportion_lower_than_high = (next_day_close_lower_than_high &␣
 ↪next_day_open_higher_with_conditions).mean()

# Output the result
next_day_proportion_lower_than_high
```

[16]: 0.0830849478390462

[17]:
```
# of the 871 days where spy closes higher given the conditions, spy opens␣
 ↪higher than close this number of days
# Step 1: Create a condition where the next day's open is higher than today's␣
 ↪close
next_day_open_higher = spy['Open'].shift(-1) > spy['Close']

# Step 2: Filter for the 395 days where SPY closes higher than open under the␣
 ↪conditions
valid_days_higher_close = valid_days[valid_days['Close_Higher_Than_Open']]
```

```
# Step 3: Count the number of days where the next day's open is higher than␣
↪today's close
num_next_day_open_higher = next_day_open_higher[valid_days_higher_close.index].
↪sum()

# Output the result
num_next_day_open_higher, num_next_day_open_higher /␣
↪len(valid_days_higher_close) * 100
```

[17]: (468, 53.73134328358209)

[18]:
```
# of those 468 day days, how many days does spy close higher than open and␣
↪lower than open?
# Step 1: Filter for the 210 days where the next day's open is higher than␣
↪today's close
valid_next_day_open_higher =␣
↪valid_days_higher_close[next_day_open_higher[valid_days_higher_close.index]]

# Step 2: Check how SPY closes on the next day (Close > Open or Close < Open)
next_day_close_higher = spy['Close'].shift(-1) > spy['Open'].shift(-1)
next_day_close_lower = spy['Close'].shift(-1) < spy['Open'].shift(-1)

# Step 3: Count the number of days where SPY closes higher or lower than open␣
↪on the next day
num_next_day_close_higher = next_day_close_higher[valid_next_day_open_higher.
↪index].sum()
num_next_day_close_lower = next_day_close_lower[valid_next_day_open_higher.
↪index].sum()

# Output the results
num_next_day_close_higher, num_next_day_close_lower
```

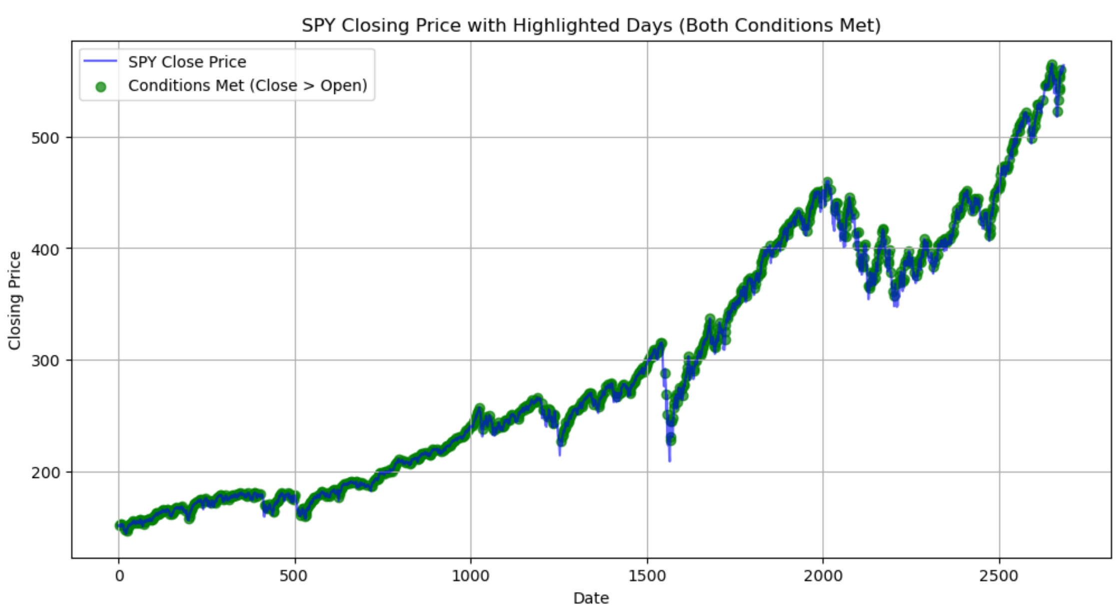
[18]: (250, 214)

[19]:
```
# let's see the proportions where spy closes higher than open and lower than␣
↪open given that the previous day
# spy close higher because the high and low were higher than the prior day's␣
↪high and low
# Calculate the proportions of days where SPY closes higher and lower than open␣
↪given our conditions are met
proportion_close_higher_next_day = num_next_day_close_higher /␣
↪num_next_day_open_higher * 100
proportion_close_lower_next_day = num_next_day_close_lower /␣
↪num_next_day_open_higher * 100

# Output the proportions
```

```
proportion_close_higher_next_day, proportion_close_lower_next_day
```

[19]: (53.41880341880342, 45.72649572649573)

[20]:
```
# Step 1: Filter the dataset to include only the days where both conditions are␣
↪met and SPY closes higher than open
spy['Both_Conditions_Met'] = condition_combined & spy['Close_Higher_Than_Open']

# Step 2: Plot the SPY closing price and highlight the days where both␣
↪conditions are met
plt.figure(figsize=(12, 6))
plt.plot(spy.index, spy['Close'], label='SPY Close Price', color='blue',␣
↪alpha=0.6)

# Highlight days where both conditions are met and SPY closes higher than open
plt.scatter(spy.index[spy['Both_Conditions_Met']],␣
↪spy['Close'][spy['Both_Conditions_Met']], color='green', label='Conditions␣
↪Met (Close > Open)', alpha=0.7)

plt.title('SPY Closing Price with Highlighted Days (Both Conditions Met)')
plt.xlabel('Date')
plt.ylabel('Closing Price')
plt.legend()
plt.grid(True)
plt.show()
```



```
[21]: # Step 1: Calculate how many days SPY closes higher than the previous 2, 3, 4,␣
       ↪and 5 days
      spy['Close_Higher_Than_2_Days'] = (spy['Close'] > spy['Close'].shift(1)) &␣
       ↪(spy['Close'] > spy['Close'].shift(2))
      spy['Close_Higher_Than_3_Days'] = spy['Close_Higher_Than_2_Days'] &␣
       ↪(spy['Close'] > spy['Close'].shift(3))
      spy['Close_Higher_Than_4_Days'] = spy['Close_Higher_Than_3_Days'] &␣
       ↪(spy['Close'] > spy['Close'].shift(4))
      spy['Close_Higher_Than_5_Days'] = spy['Close_Higher_Than_4_Days'] &␣
       ↪(spy['Close'] > spy['Close'].shift(5))

      # Step 2: Count the number of days for each condition
      num_days_2 = spy['Close_Higher_Than_2_Days'].sum()
      num_days_3 = spy['Close_Higher_Than_3_Days'].sum()
      num_days_4 = spy['Close_Higher_Than_4_Days'].sum()
      num_days_5 = spy['Close_Higher_Than_5_Days'].sum()

      # Output the results
      num_days_2, num_days_3, num_days_4, num_days_5
```

```
[21]: (1190, 1043, 950, 887)
```

```
[22]: # Step 1: Check how many days SPY closes higher than open on the days it closes␣
       ↪higher than the previous 2 days
      spy['Close_Higher_Than_Open_On_2_Days_Higher'] =␣
       ↪spy['Close_Higher_Than_2_Days'] & spy['Close_Higher_Than_Open']

      # Step 2: Count the number of days where SPY closes higher than open
      num_days_close_higher_than_open_on_2_days_higher =␣
       ↪spy['Close_Higher_Than_Open_On_2_Days_Higher'].sum()

      # Output the result
      num_days_close_higher_than_open_on_2_days_higher
```

```
[22]: 997
```

```
[23]: from sklearn.model_selection import train_test_split
      from sklearn.linear_model import LogisticRegression
      from sklearn.metrics import accuracy_score, confusion_matrix

      # Step 1: Create the binary target variable (1 if Close > Open, else 0)
      spy['Outcome'] = (spy['Close'] > spy['Open']).astype(int)

      # Step 2: Choose features and target variable
      X = spy[['Open', 'High', 'Low', 'Volume']]
      y = spy['Outcome']
```

```python
# Step 3: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Step 4: Fit the logistic regression model
log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train, y_train)

# Step 5: Make predictions on the test set
y_pred = log_reg.predict(X_test)

# Evaluate model performance
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

accuracy, conf_matrix
```

[23]:
```
(0.5605214152700186,
 array([[ 57, 196],
        [ 40, 244]], dtype=int64))
```

[24]:
```python
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE

# Step 1: Feature Engineering
spy['Range'] = spy['High'] - spy['Low']  # Add the daily range
spy['Prev_Close'] = spy['Close'].shift(1)  # Previous day's close
spy['Return'] = (spy['Close'] - spy['Prev_Close']) / spy['Prev_Close']  #␣
 ↪Previous day's return

# Fill any missing values from shifting
spy.fillna(0, inplace=True)

# Use these features now: Open, High, Low, Volume, Range, Return
X = spy[['Open', 'High', 'Low', 'Volume', 'Range', 'Return']]
y = spy['Outcome']

# Step 2: Feature Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 3: Handle class imbalance using SMOTE (Synthetic Minority Oversampling␣
 ↪Technique)
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_scaled, y)

# Step 4: Split the resampled data into training and testing sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,␣
↪test_size=0.2, random_state=42)

# Step 5: Train the logistic regression model with class weights to balance␣
↪classes
log_reg = LogisticRegression(max_iter=1000, class_weight='balanced')
log_reg.fit(X_train, y_train)

# Step 6: Make predictions on the test set
y_pred = log_reg.predict(X_test)

# Evaluate model performance
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

accuracy, conf_matrix
```

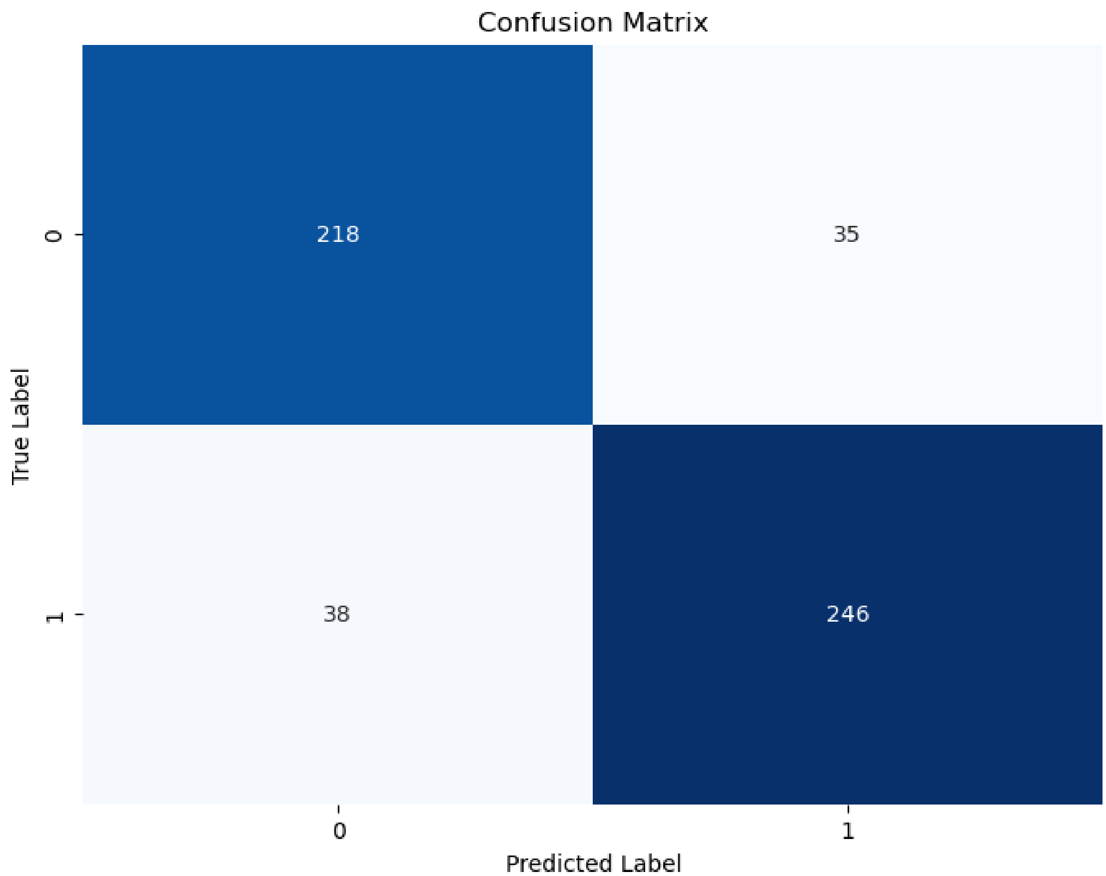
[24]:
```
(0.7986230636833046,
 array([[235,  50],
        [ 67, 229]], dtype=int64))
```

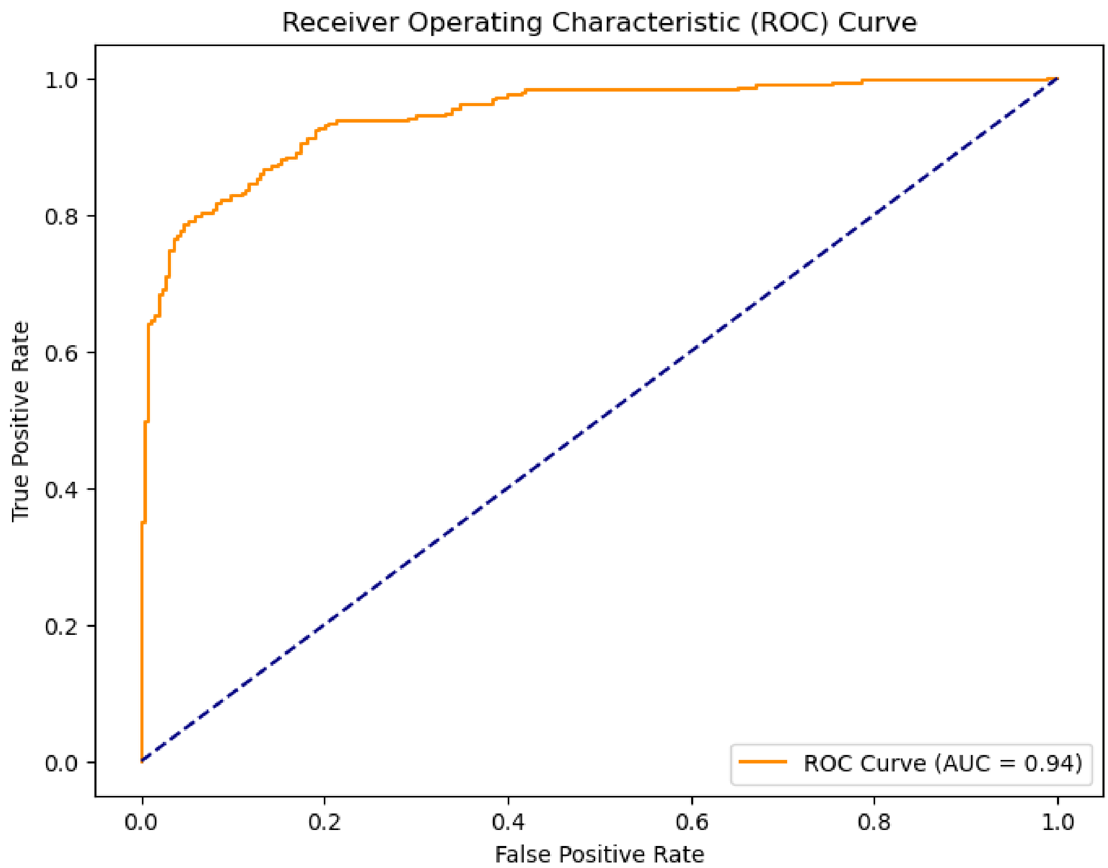
[25]:
```
from sklearn.model_selection import GridSearchCV

# Step 1: Add the day-of-week as a feature (0 for Monday, 4 for Friday, etc.)
spy['Day_of_Week'] = pd.to_datetime(spy['Date']).dt.dayofweek

# Step 2: Use these features, including Day_of_Week, and previously created␣
↪features
X = spy[['Open', 'High', 'Low', 'Volume', 'Range', 'Return', 'Day_of_Week']]
y = spy['Outcome']

# Step 3: Feature Scaling
X_scaled = scaler.fit_transform(X)

# Step 4: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
↪random_state=42)

# Step 5: Set up logistic regression with grid search to tune hyperparameters
param_grid = {'C': [0.01, 0.1, 1, 10, 100]}
log_reg = LogisticRegression(max_iter=1000, class_weight='balanced')
grid_search = GridSearchCV(log_reg, param_grid, cv=5)

# Step 6: Train the model with grid search
grid_search.fit(X_train, y_train)

# Best model from grid search
```

```
best_log_reg = grid_search.best_estimator_

# Step 7: Make predictions with the best model
y_pred = best_log_reg.predict(X_test)

# Evaluate model performance
accuracy_improved = accuracy_score(y_test, y_pred)
conf_matrix_improved = confusion_matrix(y_test, y_pred)

accuracy_improved, conf_matrix_improved, grid_search.best_params_
```

[25]:
```
(0.8640595903165735,
 array([[218,  35],
        [ 38, 246]], dtype=int64),
 {'C': 100})
```

[26]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import roc_curve, auc, roc_auc_score

# Step 1: Confusion Matrix Heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_improved, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

# Step 2: ROC Curve
y_prob = best_log_reg.predict_proba(X_test)[:, 1]  # Probabilities for the␣
 ↪positive class
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {roc_auc:.2f})", color="darkorange")
plt.plot([0, 1], [0, 1], color="navy", linestyle="--")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver Operating Characteristic (ROC) Curve")
plt.legend(loc="lower right")
plt.show()
```


Confusion Matrix
218
35
38
246
0
1
True Label
0
1
Predicted Label



[27]:

```
import numpy as np

# Assuming you have the new day's data for SPY
new_data = {
    'Open': 450.50,   # Example opening price
    'High': 453.00,   # Example high price
    'Low': 449.00,    # Example low price
    'Volume': 75000000,   # Example volume
    'Prev_Close': 448.50   # Previous day's close
}

# Step 1: Preprocess the new data
new_data['Range'] = new_data['High'] - new_data['Low']  # Calculate the range
new_data['Return'] = (new_data['Open'] - new_data['Prev_Close']) /␣
 ↪new_data['Prev_Close']  # Calculate return
new_data['Day_of_Week'] = 2  # Example: Tuesday (day_of_week = 2)

# Convert new data into a DataFrame for consistency
new_data_df = pd.DataFrame([new_data])
```

```
# Step 2: Scale the new data using the existing scaler
new_data_scaled = scaler.transform(new_data_df[['Open', 'High', 'Low',␣
↪'Volume', 'Range', 'Return', 'Day_of_Week']])

# Step 3: Make a prediction using the trained logistic regression model
prediction = best_log_reg.predict(new_data_scaled)
prediction_proba = best_log_reg.predict_proba(new_data_scaled)

# Step 4: Interpret the result
if prediction[0] == 1:
    print(f"The model predicts the SPY close will be higher than the open.")
else:
    print(f"The model predicts the SPY close will be lower than the open.")

print(f"Probability of close > open: {prediction_proba[0][1]:.2f}")
print(f"Probability of close <= open: {prediction_proba[0][0]:.2f}")
```

```
The model predicts the SPY close will be higher than the open.
Probability of close > open: 0.81
Probability of close <= open: 0.19
```

[28]:

```
# Define a function to randomly sample values from the dataset and use them in␣
↪the prediction model
def random_sample_prediction(spy, model, scaler):
    # Step 1: Randomly sample one row from the dataset
    random_sample = spy.sample(1)

    # Step 2: Extract features required for the prediction
    random_sample_data = random_sample[['Open', 'High', 'Low', 'Volume',␣
↪'Range', 'Return', 'Day_of_Week']]

    # Step 3: Scale the features using the existing scaler
    random_sample_scaled = scaler.transform(random_sample_data)

    # Step 4: Make a prediction using the trained logistic regression model
    prediction = model.predict(random_sample_scaled)
    prediction_proba = model.predict_proba(random_sample_scaled)

    # Step 5: Return the prediction and probabilities
    result = {
        'Sampled Data': random_sample_data,
        'Prediction': 'Close > Open' if prediction[0] == 1 else 'Close <= Open',
        'Probability (Close > Open)': prediction_proba[0][1],
        'Probability (Close <= Open)': prediction_proba[0][0]
    }
```

```
    return result

# Run the random sampling prediction using the current dataset, trained model,␣
 ↪and scaler
random_sample_result = random_sample_prediction(spy, best_log_reg, scaler)
random_sample_result
```

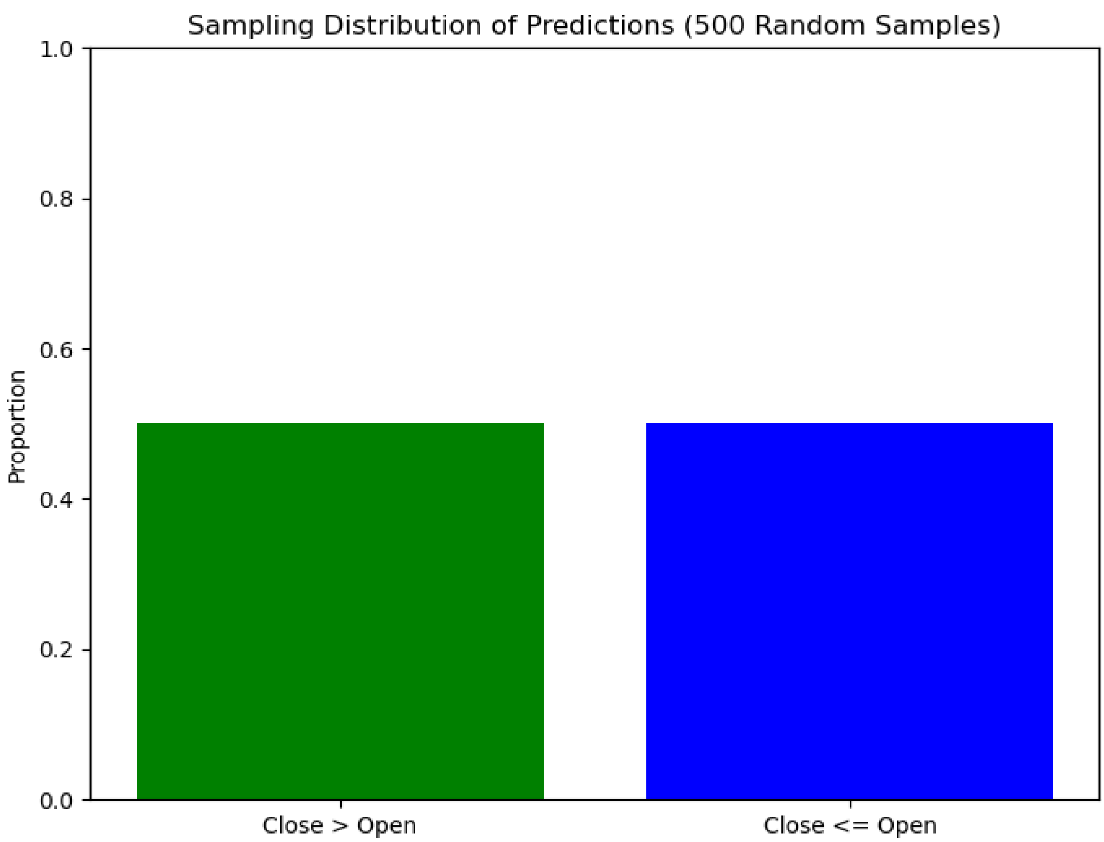
[28]:
```
{'Sampled Data':             Open        High         Low    Volume     Range
Return  \
 82  156.791424  157.307893  156.383238  93019000  0.924655  0.000106

     Day_of_Week
 82            3  ,
 'Prediction': 'Close > Open',
 'Probability (Close > Open)': 0.5010563358966335,
 'Probability (Close <= Open)': 0.49894366410336655}
```

[29]:
```
# Define a function to run multiple random sample predictions and calculate the␣
 ↪sampling distribution
def simulation_prediction_distribution(spy, model, scaler, n_samples=30):
    predictions = []

    for _ in range(n_samples):
        # Get the prediction result for a random sample
        result = random_sample_prediction(spy, model, scaler)
        # Store the prediction ('Close > Open' or 'Close <= Open')
        predictions.append(result['Prediction'])

    # Calculate the distribution of the predictions
    close_greater_than_open = predictions.count('Close > Open') / n_samples
    close_less_than_or_equal_open = predictions.count('Close <= Open') /␣
 ↪n_samples

    distribution = {
        'Close > Open': close_greater_than_open,
        'Close <= Open': close_less_than_or_equal_open
    }

    return distribution

# Run the simulation with 500 random samples
simulation_distribution = simulation_prediction_distribution(spy, best_log_reg,␣
 ↪scaler, n_samples=500)
simulation_distribution
```

[29]:
```
{'Close > Open': 0.5, 'Close <= Open': 0.5}
```

[30]:

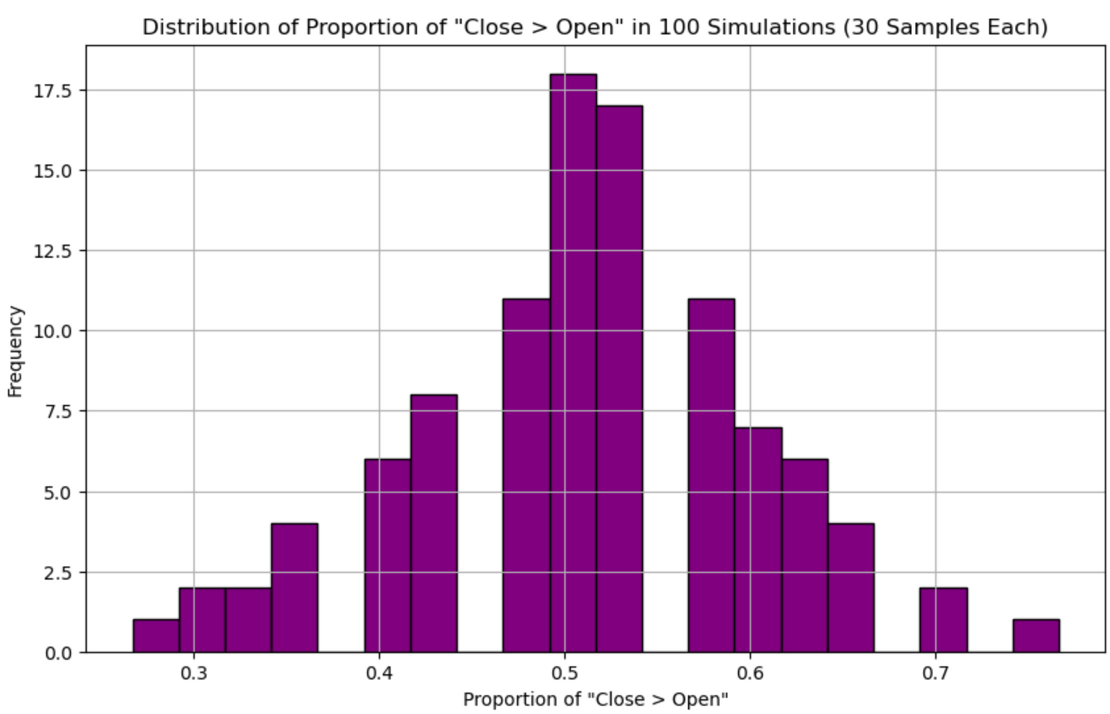
```python
# Import necessary library for visualization
import numpy as np

# Visualize the sampling distribution as a bar chart
labels = ['Close > Open', 'Close <= Open']
values = [simulation_distribution['Close > Open'],␣
 ↪simulation_distribution['Close <= Open']]

plt.figure(figsize=(8, 6))
plt.bar(labels, values, color=['green', 'blue'])
plt.title('Sampling Distribution of Predictions (500 Random Samples)')
plt.ylabel('Proportion')
plt.ylim(0, 1)
plt.show()

# Generate summary statistics
summary_statistics = {
    'Mean (Close > Open)': np.mean(values),
    'Standard Deviation': np.std(values),
    'Variance': np.var(values)
}

summary_statistics
```



[30]: {'Mean (Close > Open)': 0.5, 'Standard Deviation': 0.0, 'Variance': 0.0}

[31]:
```
import numpy as np
import matplotlib.pyplot as plt

# Assuming this function is defined
# def random_sample_prediction(spy, model, scaler): ...

# Define a function to run multiple random sample predictions and calculate the␣
↪sampling distribution
def simulation_prediction_distribution(spy, model, scaler, n_samples=30):
    predictions = []

    for _ in range(n_samples):
        # Get the prediction result for a random sample
        result = random_sample_prediction(spy, model, scaler)
        predictions.append(result['Prediction'])

    # Calculate the distribution of the predictions
    close_greater_than_open = predictions.count('Close > Open') / n_samples
    close_less_than_or_equal_open = predictions.count('Close <= Open') /␣
↪n_samples

    return {
        'Close > Open': close_greater_than_open,
        'Close <= Open': close_less_than_or_equal_open
    }

# Define the multiple_simulations function
def multiple_simulations(spy, model, scaler, n_simulations=100,␣
↪n_samples_per_simulation=30):
    simulation_proportions = []

    for _ in range(n_simulations):
        # Run a single simulation with n_samples_per_simulation random samples
        distribution = simulation_prediction_distribution(spy, model, scaler,␣
↪n_samples=n_samples_per_simulation)
        # Store the proportion of 'Close > Open'
        simulation_proportions.append(distribution['Close > Open'])

    return simulation_proportions

# Run the simulation with fewer simulations and samples for faster execution
```

```python
simulation_proportions_reduced = multiple_simulations(spy, best_log_reg,␣
↪scaler, n_simulations=100, n_samples_per_simulation=30)

# Visualize the distribution of proportions from the 100 simulations
plt.figure(figsize=(10, 6))
plt.hist(simulation_proportions_reduced, bins=20, color='purple',␣
↪edgecolor='black')
plt.title('Distribution of Proportion of "Close > Open" in 100 Simulations (30␣
↪Samples Each)')
plt.xlabel('Proportion of "Close > Open"')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()

# Summary statistics for the reduced number of simulations
summary_stats_reduced_simulations = {
    'Mean Proportion (Close > Open)': np.mean(simulation_proportions_reduced),
    'Standard Deviation': np.std(simulation_proportions_reduced),
    'Variance': np.var(simulation_proportions_reduced)
}

print(f"Summary statistics for reduced simulations:␣
↪{summary_stats_reduced_simulations}")
```



```
Summary statistics for reduced simulations: {'Mean Proportion (Close > Open)':
0.5113333333333334, 'Standard Deviation': 0.09288224324977896, 'Variance':
0.008627111111111111}
```

[ ]:

```

```