Practical 8 - Mini Project

Name : Ansh Sanjay Agrawal

Roll No. : 20

Batch : A2

In [3]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, precision_score, recall_score, f1_score

# Load the dataset
file_path = 'Walmart_Sales.csv'
data = pd.read_csv(file_path)
```

In [5]:
```
# Convert "Date" to datetime format
data['Date'] = pd.to_datetime(data['Date'], format='%d-%m-%Y')

# 1. Exploratory Data Analysis (EDA)
```

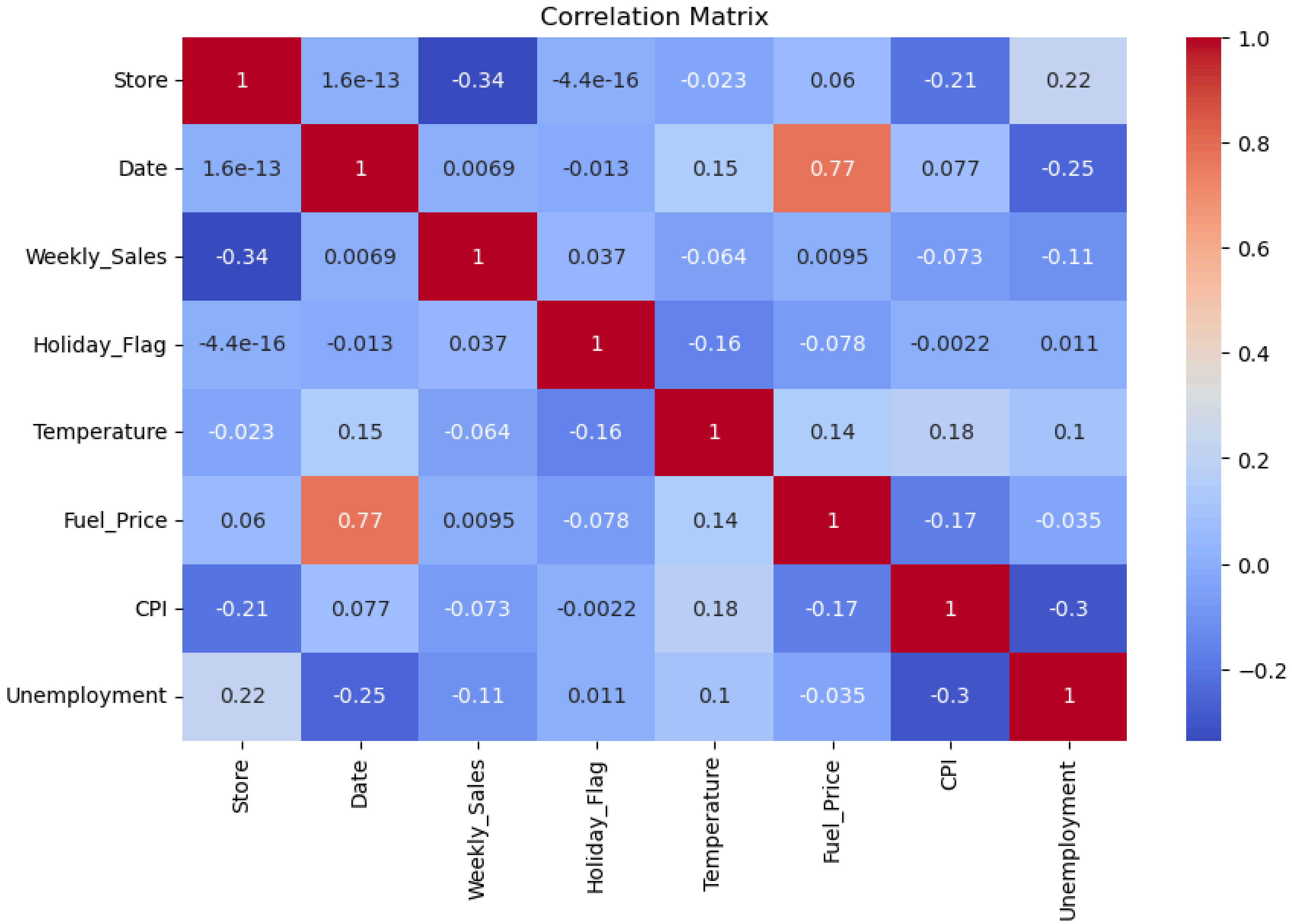
```python
# Summary statistics
print("Summary Statistics:\n", data.describe())
```

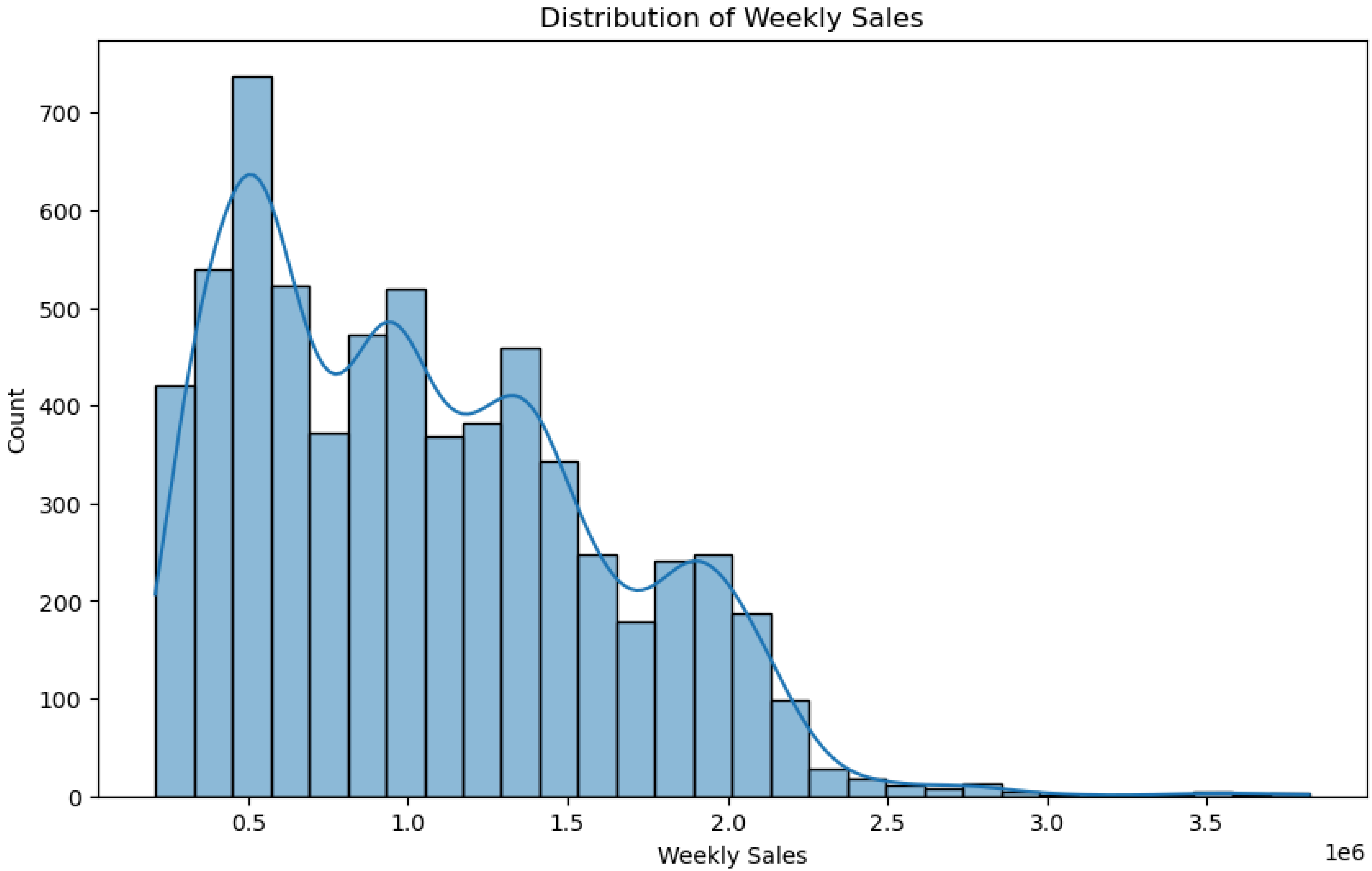
```
Summary Statistics:
              Store                 Date  Weekly_Sales  Holiday_Flag  \
count  6435.000000                 6435  6.435000e+03   6435.000000
mean     23.000000  2011-06-17 00:00:00  1.046965e+06      0.069930
min       1.000000  2010-02-05 00:00:00  2.099862e+05      0.000000
25%      12.000000  2010-10-08 00:00:00  5.533501e+05      0.000000
50%      23.000000  2011-06-17 00:00:00  9.607460e+05      0.000000
75%      34.000000  2012-02-24 00:00:00  1.420159e+06      0.000000
max      45.000000  2012-10-26 00:00:00  3.818686e+06      1.000000
std      12.988182                  NaN  5.643666e+05      0.255049

       Temperature   Fuel_Price          CPI  Unemployment
count  6435.000000  6435.000000  6435.000000   6435.000000
mean     60.663782     3.358607   171.578394      7.999151
min      -2.060000     2.472000   126.064000      3.879000
25%      47.460000     2.933000   131.735000      6.891000
50%      62.670000     3.445000   182.616521      7.874000
75%      74.940000     3.735000   212.743293      8.622000
max     100.140000     4.468000   227.232807     14.313000
std      18.444933     0.459020    39.356712      1.875885
```

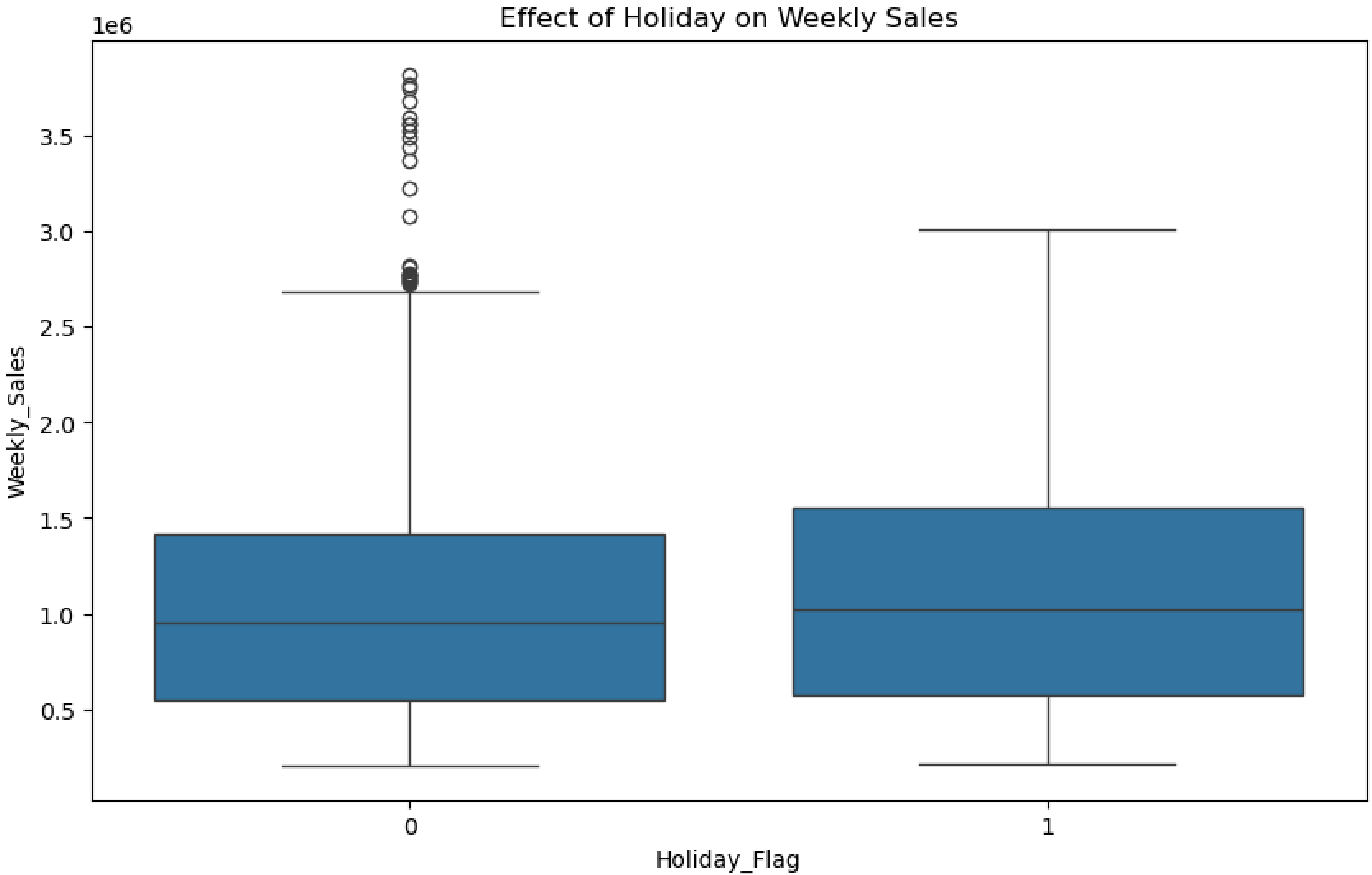
In [7]:

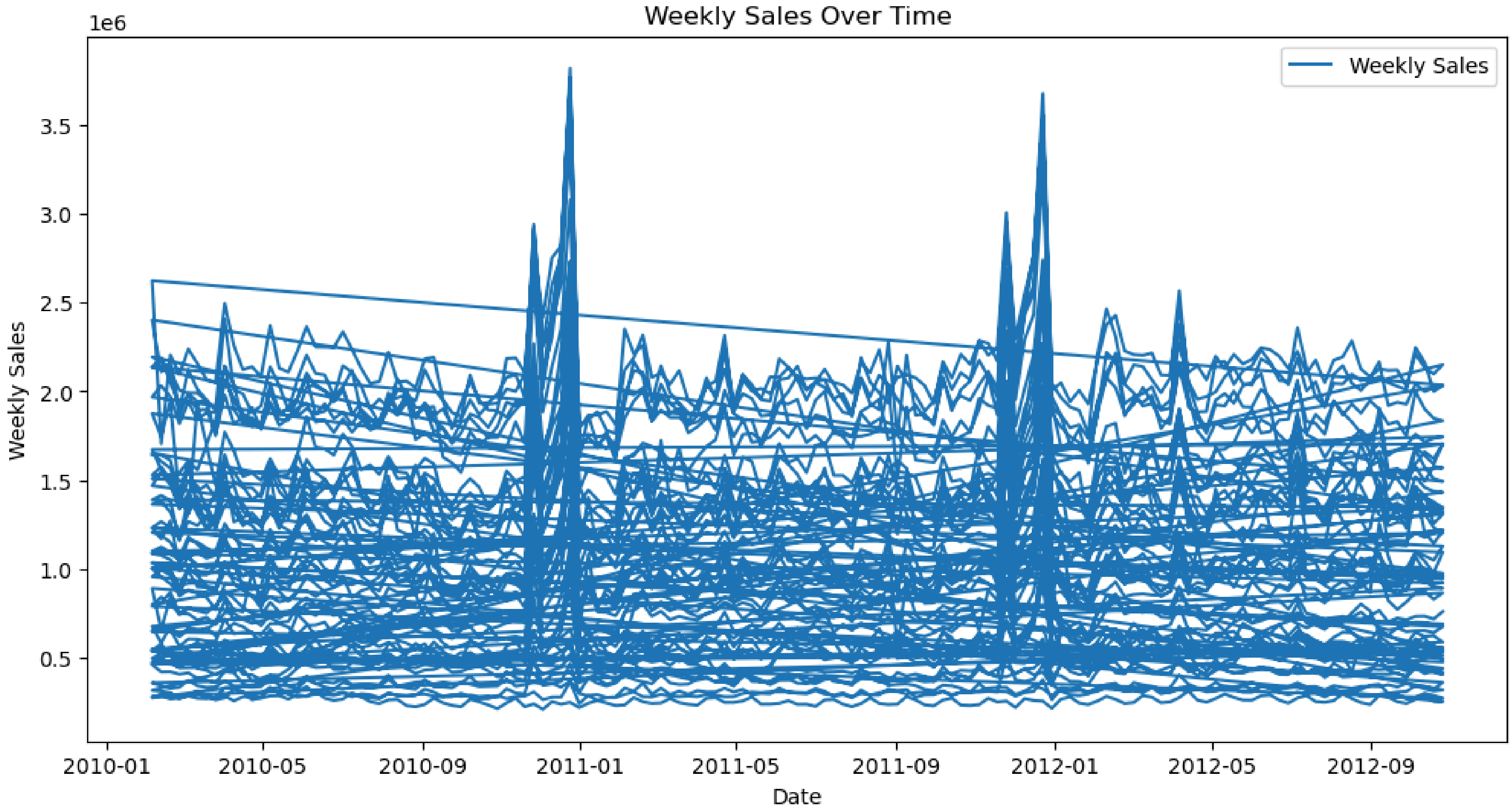
```python
# Correlation matrix visualization
plt.figure(figsize=(10,6))
sns.heatmap(data.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()

# Visualizing the distribution of Weekly Sales
plt.figure(figsize=(10,6))
sns.histplot(data['Weekly_Sales'], bins=30, kde=True)
plt.title('Distribution of Weekly Sales')
plt.xlabel('Weekly Sales')
plt.show()

# Visualizing the effect of Holiday on Weekly Sales
plt.figure(figsize=(10,6))
sns.boxplot(x='Holiday_Flag', y='Weekly_Sales', data=data)
```

```python
plt.title('Effect of Holiday on Weekly Sales')
plt.show()

# Time-series analysis: plotting sales over time
plt.figure(figsize=(12,6))
plt.plot(data['Date'], data['Weekly_Sales'], label='Weekly Sales')
plt.title('Weekly Sales Over Time')
plt.xlabel('Date')
plt.ylabel('Weekly Sales')
plt.legend()
plt.show()
```


Correlation Matrix
Store
Date
Weekly_Sales
Holiday_Flag
Temperature
Fuel_Price
CPI
Unemployment
1 1.6e-13 -0.34 -4.4e-16 -0.023 0.06 -0.21 0.22
1.6e-13 1 0.0069 -0.013 0.15 0.77 0.077 -0.25
-0.34 0.0069 1 0.037 -0.064 0.0095 -0.073 -0.11
-4.4e-16 -0.013 0.037 1 -0.16 -0.078 -0.0022 0.011
-0.023 0.15 -0.064 -0.16 1 0.14 0.18 0.1
0.06 0.77 0.0095 -0.078 0.14 1 -0.17 -0.035
-0.21 0.077 -0.073 -0.0022 0.18 -0.17 1 -0.3
0.22 -0.25 -0.11 0.011 0.1 -0.035 -0.3 1
1.0
0.8
0.6
0.4
0.2
0.0
−0.2


Distribution of Weekly Sales
Count
700
600
500
400
300
200
100
0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
1e6
Weekly Sales


Effect of Holiday on Weekly Sales
1e6
3.5
3.0
2.5
2.0
1.5
1.0
0.5
Weekly_Sales
0
1
Holiday_Flag



In [8]:
```
# 2. Data Preprocessing

# Create a new feature: Extract Year and Month from Date
data['Year'] = data['Date'].dt.year
data['Month'] = data['Date'].dt.month

# Create binary classification target (High Sales vs. Low Sales) based on median
median_sales = data['Weekly_Sales'].median()
data['High_Sales'] = np.where(data['Weekly_Sales'] > median_sales, 1, 0)
```

```
# Drop unnecessary columns for classification
X = data.drop(columns=['Weekly_Sales', 'Date', 'High_Sales'])
y = data['High_Sales']

# Split data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardizing the numeric features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [9]:

```
# 3. Model Training and Evaluation

# Define the classification models
models = {
    "Logistic Regression": LogisticRegression(),
    "Naive Bayes": GaussianNB(),
    "SVM": SVC(),
    "KNN": KNeighborsClassifier(),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),
    "AdaBoost": AdaBoostClassifier()
}

# Train, predict, and evaluate each model
results = {}

for name, model in models.items():
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    results[name] = {
        'Accuracy': accuracy,
        'Precision': precision,
```

```
        'Recall': recall,
        'F1 Score': f1
    }

    # Display the classification report and confusion matrix for each model
    print(f"Model: {name}")
    print(classification_report(y_test, y_pred))
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
    print("\n" + "="*50 + "\n")

# Convert results to DataFrame for comparison
results_df = pd.DataFrame(results).T
```

```
Model: Logistic Regression
              precision    recall  f1-score   support

           0       0.65      0.59      0.62       639
           1       0.63      0.69      0.66       648

    accuracy                           0.64      1287
   macro avg       0.64      0.64      0.64      1287
weighted avg       0.64      0.64      0.64      1287

Confusion Matrix:
 [[374 265]
 [199 449]]

==================================================

Model: Naive Bayes
              precision    recall  f1-score   support

           0       0.64      0.57      0.60       639
           1       0.62      0.69      0.65       648

    accuracy                           0.63      1287
   macro avg       0.63      0.63      0.63      1287
weighted avg       0.63      0.63      0.63      1287

Confusion Matrix:
 [[362 277]
 [200 448]]

==================================================

Model: SVM
              precision    recall  f1-score   support

           0       0.74      0.74      0.74       639
           1       0.74      0.75      0.75       648

    accuracy                           0.74      1287
   macro avg       0.74      0.74      0.74      1287
weighted avg       0.74      0.74      0.74      1287
```

```
Confusion Matrix:
 [[473 166]
 [164 484]]

==================================================

Model: KNN
              precision    recall  f1-score   support

           0       0.80      0.83      0.82       639
           1       0.83      0.80      0.81       648

    accuracy                           0.82      1287
   macro avg       0.82      0.82      0.82      1287
weighted avg       0.82      0.82      0.82      1287

Confusion Matrix:
 [[530 109]
 [129 519]]

==================================================

Model: Decision Tree
              precision    recall  f1-score   support

           0       0.95      0.95      0.95       639
           1       0.95      0.95      0.95       648

    accuracy                           0.95      1287
   macro avg       0.95      0.95      0.95      1287
weighted avg       0.95      0.95      0.95      1287

Confusion Matrix:
 [[607  32]
 [ 33 615]]

==================================================

Model: Random Forest
              precision    recall  f1-score   support
```

```
           0       0.96      0.95      0.95       639
           1       0.95      0.96      0.95       648

    accuracy                           0.95      1287
   macro avg       0.95      0.95      0.95      1287
weighted avg       0.95      0.95      0.95      1287

Confusion Matrix:
 [[605  34]
 [ 28 620]]

==================================================
```

```
C:\Users\Sai\anaconda3\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:519: FutureWarning: The SAMME.R algorithm (the de
fault) is deprecated and will be removed in 1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
```

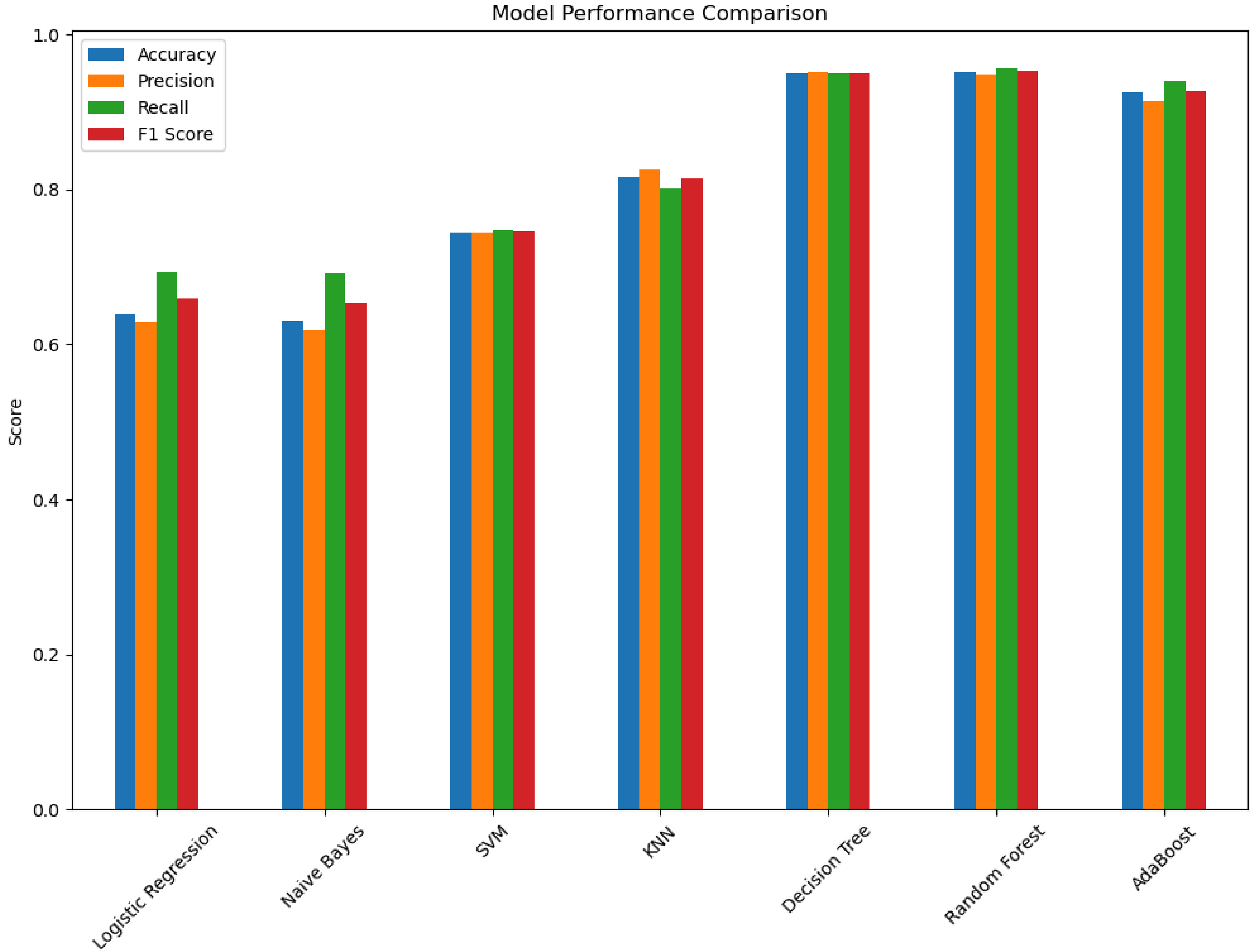
```
Model: AdaBoost
              precision    recall  f1-score   support

           0       0.94      0.91      0.92       639
           1       0.91      0.94      0.93       648

    accuracy                           0.92      1287
   macro avg       0.93      0.92      0.92      1287
weighted avg       0.92      0.92      0.92      1287

Confusion Matrix:
 [[581  58]
 [ 39 609]]

==================================================
```

In [10]:

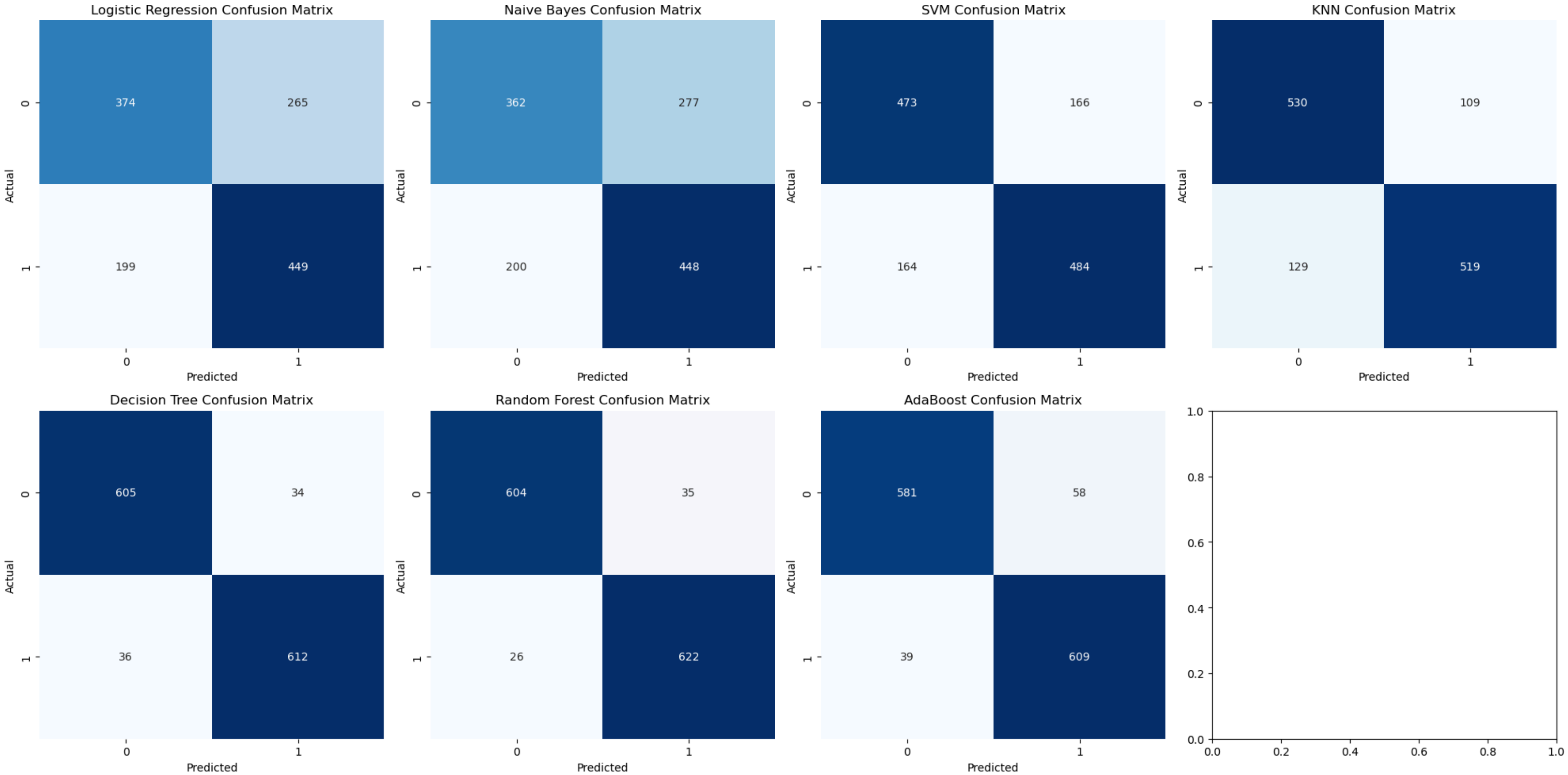
```
# 4. Compare Performance

# Display the results DataFrame
print("Model Performance Comparison:\n", results_df)

# Plot the performance comparison of all models
```

```
results_df.plot(kind='bar', figsize=(12, 8))
plt.title('Model Performance Comparison')
plt.ylabel('Score')
plt.xticks(rotation=45)
plt.show()

# Visualizing Confusion Matrices for deeper insights
fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(20, 10))
axes = axes.ravel()

for idx, (name, model) in enumerate(models.items()):
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)

    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', ax=axes[idx], cmap='Blues', cbar=False)
    axes[idx].set_title(f'{name} Confusion Matrix')
    axes[idx].set_xlabel('Predicted')
    axes[idx].set_ylabel('Actual')

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

```
Model Performance Comparison:
                     Accuracy  Precision    Recall  F1 Score
Logistic Regression  0.639472   0.628852  0.692901  0.659325
Naive Bayes          0.629371   0.617931  0.691358  0.652586
SVM                  0.743590   0.744615  0.746914  0.745763
KNN                  0.815074   0.826433  0.800926  0.813480
Decision Tree        0.949495   0.950541  0.949074  0.949807
Random Forest        0.951826   0.948012  0.956790  0.952381
AdaBoost             0.924631   0.913043  0.939815  0.926236
```


Model Performance Comparison
Accuracy
Precision
Recall
F1 Score
Score
1.0
0.8
0.6
0.4
0.2
0.0
Logistic Regression
Naive Bayes
SVM
KNN
Decision Tree
Random Forest
AdaBoost

```
C:\Users\Sai\anaconda3\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:519: FutureWarning: The SAMME.R algorithm (the de
fault) is deprecated and will be removed in 1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
```




```
In [ ]:
```