```
#Important packages
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier, export_text, plot_tree
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, ConfusionMatrixDisplay
from imblearn.over_sampling import SMOTE  # For handling class imbalance
from sklearn.ensemble import RandomForestClassifier
```

```
#TASK 1

# Load dataset
df = pd.read_csv("customer_churn.csv")

# Check first few rows
df.head()
```

	CustomerID	Age	Subscription_Length_Months	Watch_Time_Hours	Number_of_Logins	Preferred_Content_Type	Membership_Typ
0	1	56	35	62.579266	73	TV Shows	Bas
1	2	69	15	159.714415	1	Sports	Bas
2	3	46	25	41.119547	36	Movies	Premiu
3	4	32	28	183.961735	35	Movies	Standa
4	5	60	10	87.782848	66	Movies	Standa

```
# Summary statistics
df.describe()
```

	CustomerID	Age	Subscription_Length_Months	Watch_Time_Hours	Number_of_Logins	Payment_Issues	Number_of_Co
count	1000.000000	1000.00000	1000.000000	1000.000000	1000.000000	1000.000000	10
mean	500.500000	43.81900	18.218000	100.794546	50.387000	0.154000	
std	288.819436	14.99103	10.177822	56.477606	28.224171	0.361129	
min	1.000000	18.00000	1.000000	5.036738	1.000000	0.000000	
25%	250.750000	31.00000	9.000000	50.383080	26.000000	0.000000	
50%	500.500000	44.00000	18.000000	100.234954	51.000000	0.000000	
75%	750.250000	56.00000	27.000000	150.445885	75.000000	0.000000	
max	1000.000000	69.00000	35.000000	199.944192	99.000000	1.000000	

```
# Check missing values
df.isnull().sum()
```

	0
CustomerID	0
Age	0
Subscription_Length_Months	0
Watch_Time_Hours	0
Number_of_Logins	0
Preferred_Content_Type	0
Membership_Type	0
Payment_Method	0
Payment_Issues	0
Number_of_Complaints	0
Resolution_Time_Days	0
Churn	0

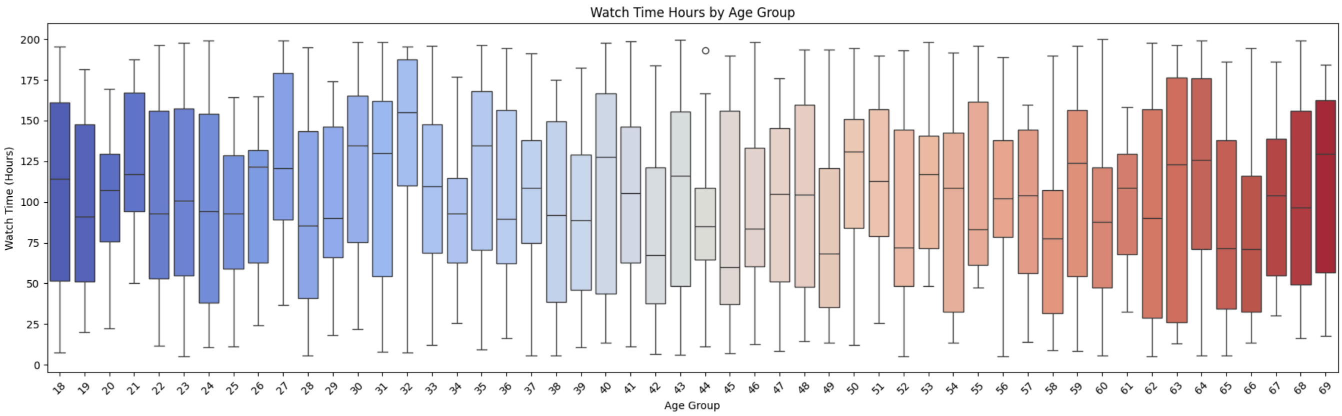
dtype: int64

```
# Check dataset info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 12 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   CustomerID                  1000 non-null   int64  
 1   Age                         1000 non-null   int64  
 2   Subscription_Length_Months  1000 non-null   int64  
 3   Watch_Time_Hours            1000 non-null   float64
 4   Number_of_Logins            1000 non-null   int64  
 5   Preferred_Content_Type      1000 non-null   object 
 6   Membership_Type             1000 non-null   object 
 7   Payment_Method              1000 non-null   object 
 8   Payment_Issues              1000 non-null   int64  
 9   Number_of_Complaints        1000 non-null   int64  
 10  Resolution_Time_Days        1000 non-null   int64  
 11  Churn                       1000 non-null   int64  
dtypes: float64(1), int64(8), object(3)
memory usage: 93.9+ KB
```

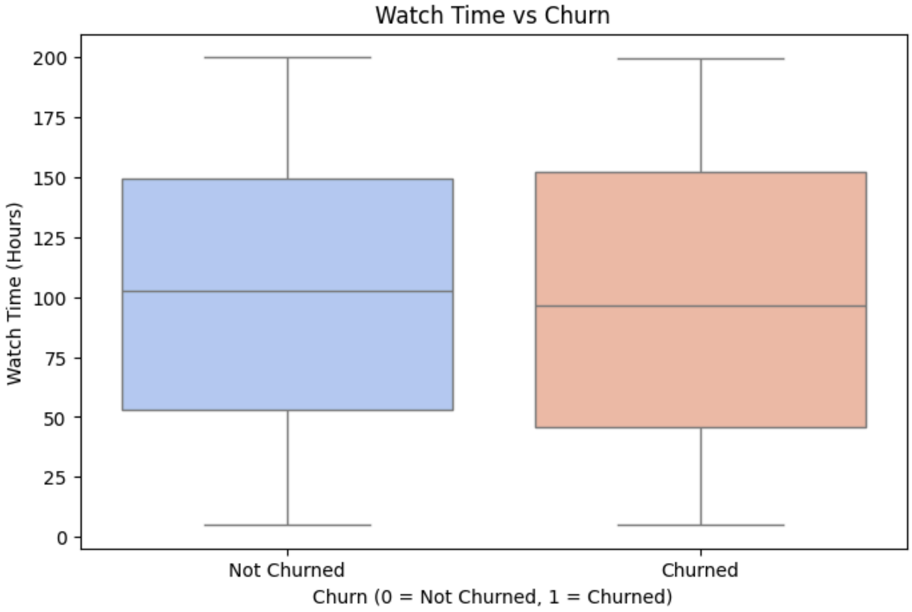
```
#Visualize data distributions

plt.figure(figsize=(22, 6))
sns.boxplot(x=df["Age"], y=df["Watch_Time_Hours"], palette="coolwarm")

plt.title("Watch Time Hours by Age Group")
plt.xlabel("Age Group")
plt.ylabel("Watch Time (Hours)")
plt.xticks(rotation=45)
plt.show();
```

```
<ipython-input-33-e5adb9a3225d>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue`

  sns.boxplot(x=df["Age"], y=df["Watch_Time_Hours"], palette="coolwarm")
```



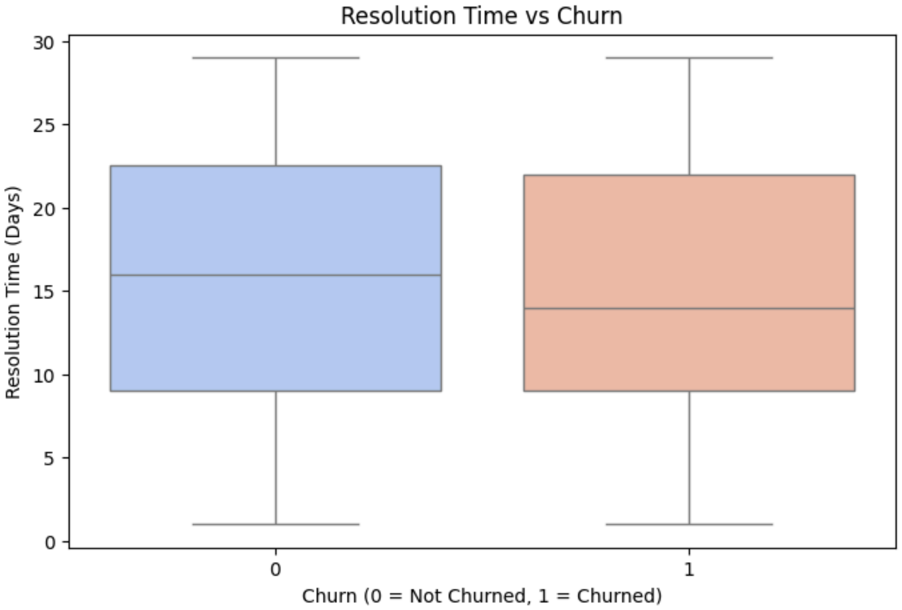

```
plt.figure(figsize=(8, 5))
sns.boxplot(x=df["Churn"], y=df["Watch_Time_Hours"], palette="coolwarm")

plt.title("Watch Time vs Churn")
plt.xlabel("Churn (0 = Not Churned, 1 = Churned)")
plt.ylabel("Watch Time (Hours)")
plt.xticks([0, 1], ["Not Churned", "Churned"])
plt.show()
```

```
<ipython-input-34-7786c40664a3>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue`

  sns.boxplot(x=df["Churn"], y=df["Watch_Time_Hours"], palette="coolwarm")
```

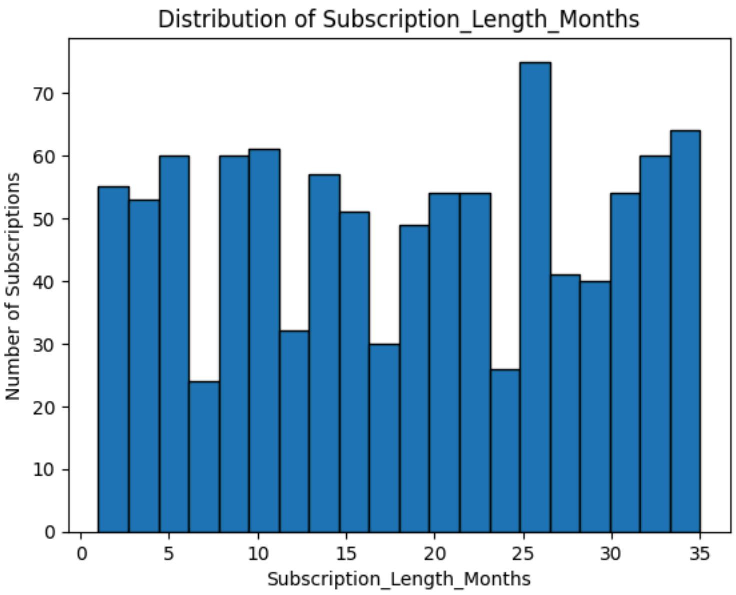



```
plt.figure(figsize=(8, 5))
sns.boxplot(x=df["Churn"], y=df["Resolution_Time_Days"], palette="coolwarm")

plt.title("Resolution Time vs Churn")
plt.xlabel("Churn (0 = Not Churned, 1 = Churned)")
plt.ylabel("Resolution Time (Days)")
plt.show()
```

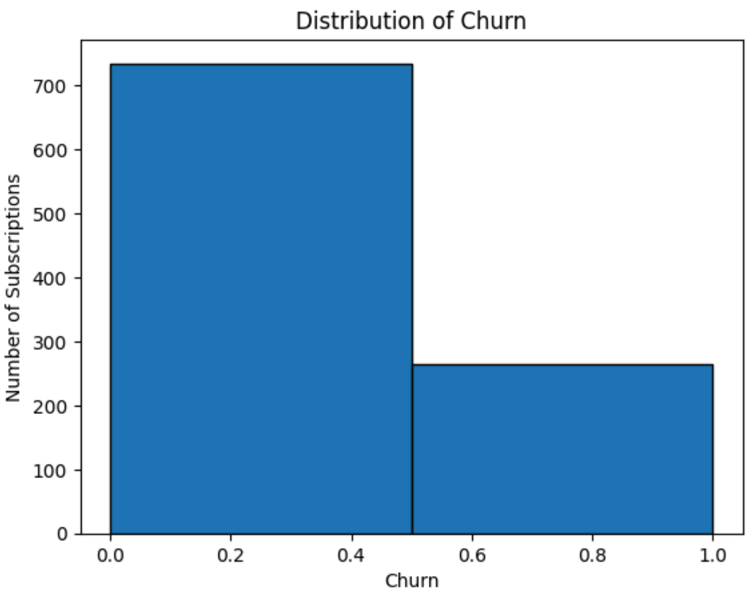
```
<ipython-input-35-914ed4838e07>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue`

  sns.boxplot(x=df["Churn"], y=df["Resolution_Time_Days"], palette="coolwarm")
```



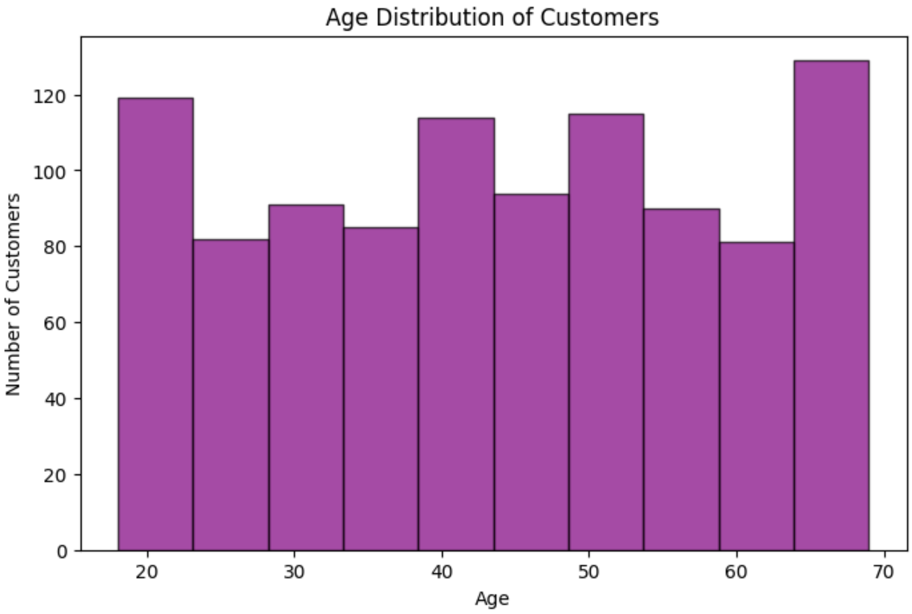

```
# Histogram for Subscription Length
plt.hist(df['Subscription_Length_Months'], bins=20, edgecolor='k')
plt.title('Distribution of Subscription_Length_Months')
plt.xlabel('Subscription_Length_Months')
plt.ylabel('Number of Subscriptions')
plt.show()
```



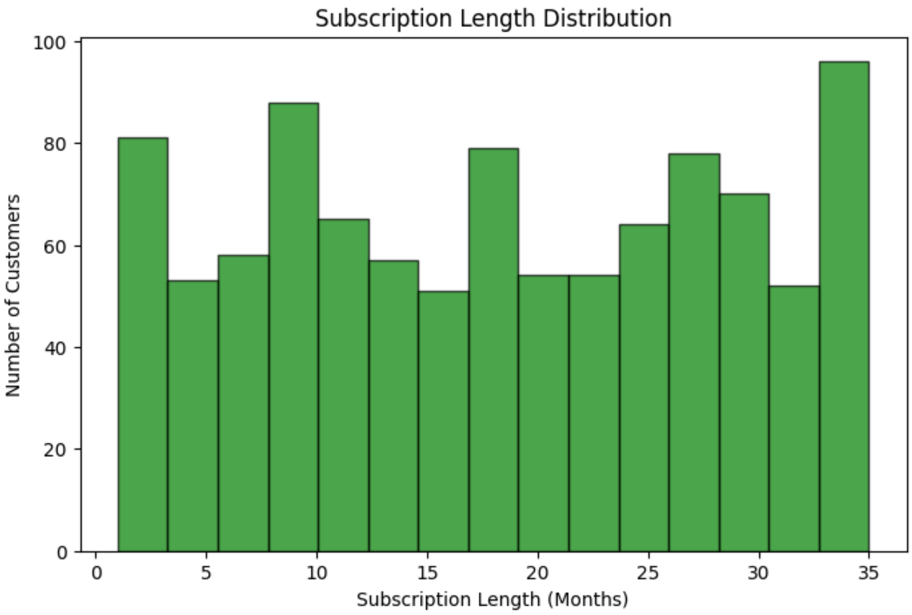

```
# Histogram for Churn - checking for class imbalance
plt.hist(df['Churn'], bins=2, edgecolor='k')
plt.title('Distribution of Churn')
plt.xlabel('Churn')
plt.ylabel('Number of Subscriptions')
plt.show()
```




```
plt.figure(figsize=(8, 5))
plt.hist(df["Age"], bins=10, edgecolor="black", color="purple", alpha=0.7)
plt.title("Age Distribution of Customers")
plt.xlabel("Age")
plt.ylabel("Number of Customers")
plt.show()
```

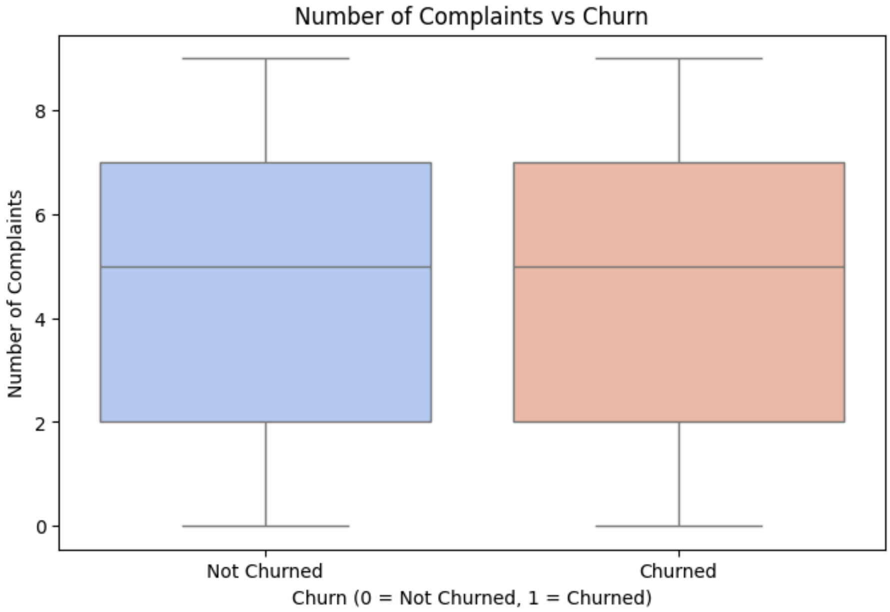



```
plt.figure(figsize=(8, 5))
plt.hist(df["Subscription_Length_Months"], bins=15, edgecolor="black", color="green", alpha=0.7)
plt.title("Subscription Length Distribution")
plt.xlabel("Subscription Length (Months)")
plt.ylabel("Number of Customers")
plt.show()
```

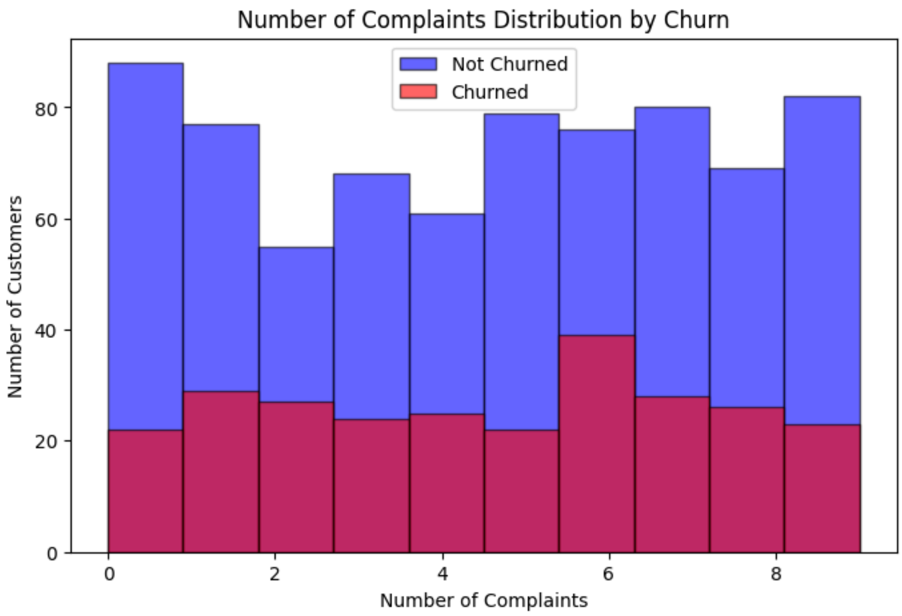



```
import seaborn as sns

plt.figure(figsize=(8, 5))
sns.boxplot(x=df["Churn"], y=df["Number_of_Complaints"], palette="coolwarm")
plt.title("Number of Complaints vs Churn")
plt.xlabel("Churn (0 = Not Churned, 1 = Churned)")
plt.ylabel("Number of Complaints")
plt.xticks([0, 1], ["Not Churned", "Churned"])
plt.show()
```

```
<ipython-input-40-1d58590b928b>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue`

  sns.boxplot(x=df["Churn"], y=df["Number_of_Complaints"], palette="coolwarm")
```




```
plt.figure(figsize=(8, 5))
plt.hist(df[df["Churn"] == 0]["Number_of_Complaints"], bins=10, alpha=0.6, label="Not Churned", color="blue", edgecolor="bla
plt.hist(df[df["Churn"] == 1]["Number_of_Complaints"], bins=10, alpha=0.6, label="Churned", color="red", edgecolor="black")
plt.title("Number of Complaints Distribution by Churn")
plt.xlabel("Number of Complaints")
plt.ylabel("Number of Customers")
plt.legend()
plt.show()

churn_counts = df["Churn"].value_counts()
print(churn_counts, "\n")
correlation = df["Number_of_Complaints"].corr(df["Churn"])
print(f"Correlation between Complaints and Churn: {correlation:.2f}")
```

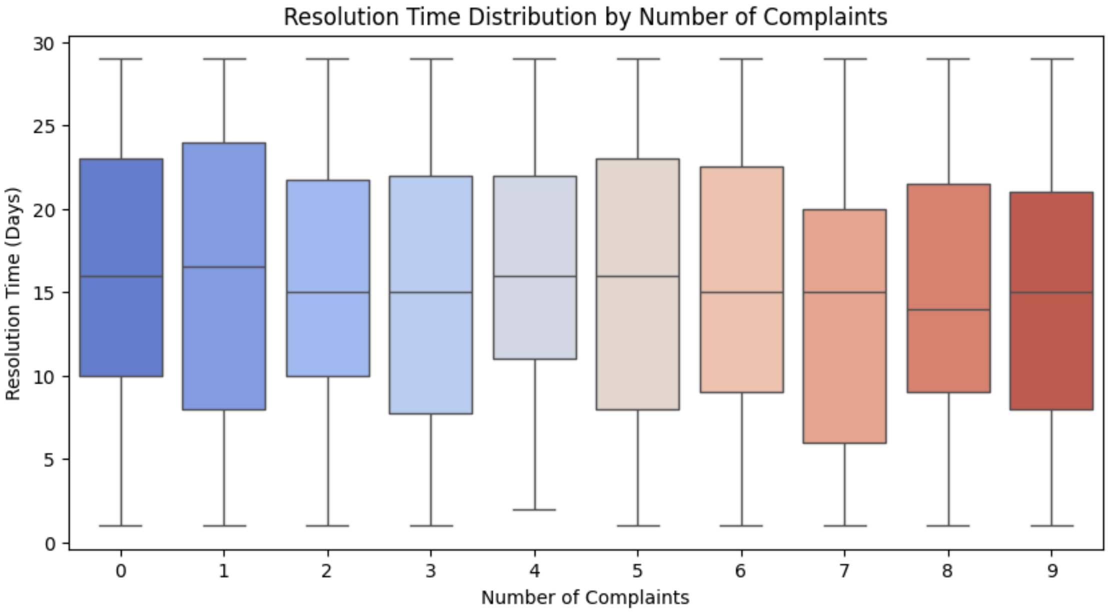



```
Churn
0    735
1    265
Name: count, dtype: int64

Correlation between Complaints and Churn: 0.00
```

```
plt.figure(figsize=(10, 5))
sns.boxplot(x=df["Number_of_Complaints"], y=df["Resolution_Time_Days"], palette="coolwarm")

plt.title("Resolution Time Distribution by Number of Complaints")
```

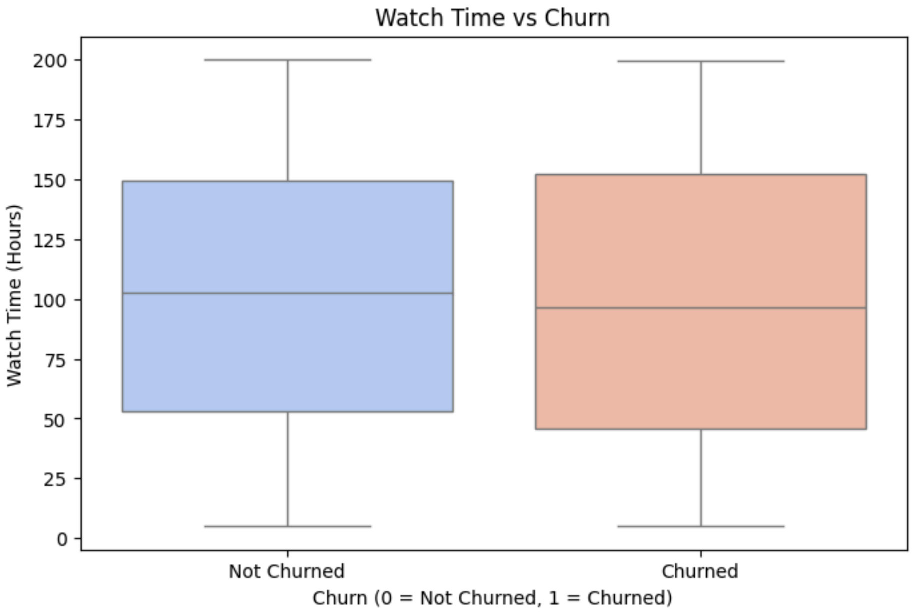
```
plt.xlabel("Number of Complaints")
plt.ylabel("Resolution Time (Days)")
plt.show()
```

```
<ipython-input-42-7a713d04199a>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue`

  sns.boxplot(x=df["Number_of_Complaints"], y=df["Resolution_Time_Days"], palette="coolwarm")
```




```
plt.figure(figsize=(8, 5))
sns.boxplot(x=df["Churn"], y=df["Watch_Time_Hours"], palette="coolwarm")

plt.title("Watch Time vs Churn")
plt.xlabel("Churn (0 = Not Churned, 1 = Churned)")
plt.ylabel("Watch Time (Hours)")
plt.xticks([0, 1], ["Not Churned", "Churned"])
plt.show()
```

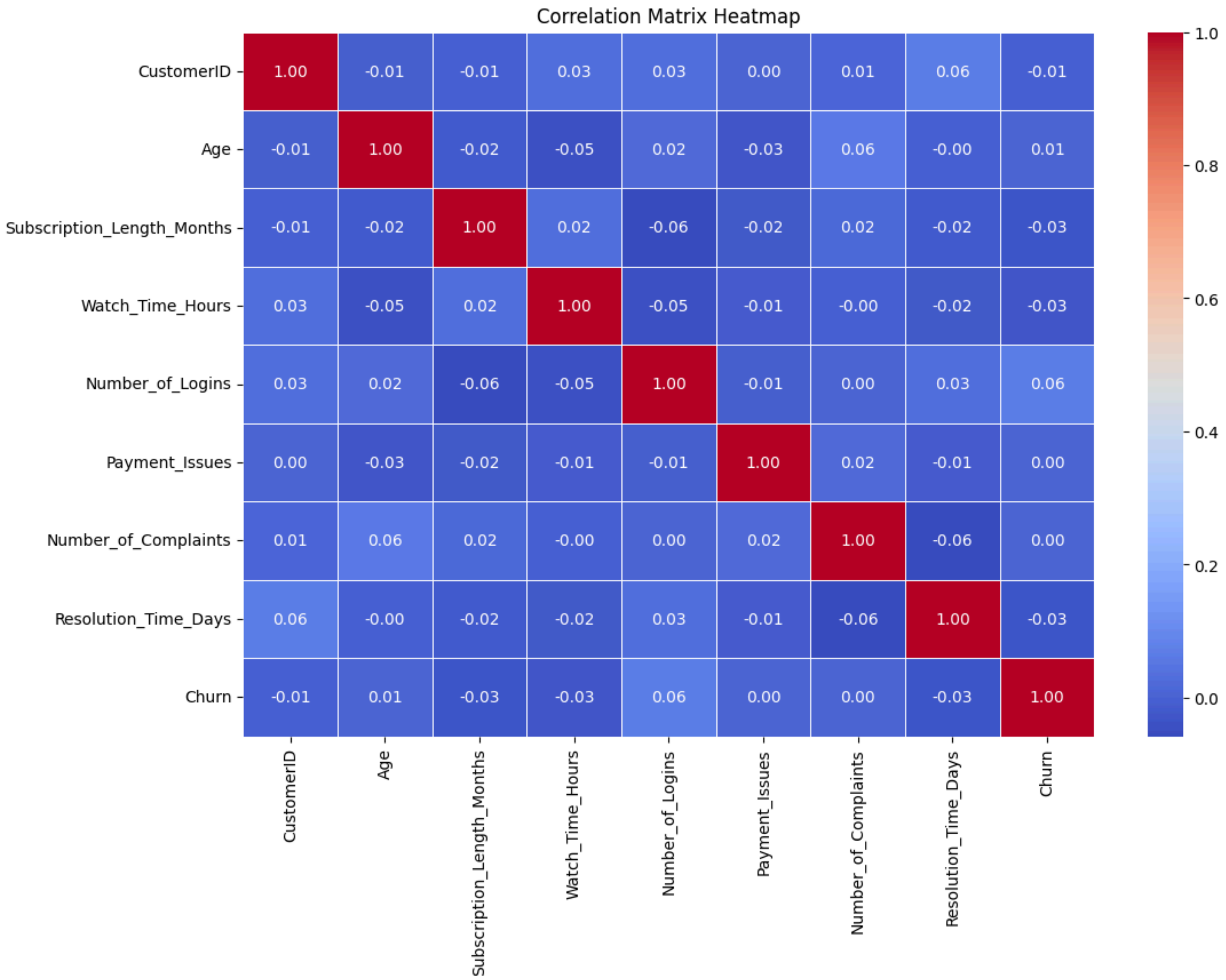
```
<ipython-input-43-7786c40664a3>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue`

  sns.boxplot(x=df["Churn"], y=df["Watch_Time_Hours"], palette="coolwarm")
```




```
#CORRELATION BETWEEN VARIABLE

# Drop non-numeric columns for correlation analysis
numeric_df = df.select_dtypes(include=['float64', 'int64'])
```

```
# Calculate the correlation matrix
correlation_matrix = numeric_df.corr()

# Display the correlation matrix
print("Correlation Matrix:")
print(correlation_matrix)

# Visualize the correlation matrix using a heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title("Correlation Matrix Heatmap")
plt.show()
```

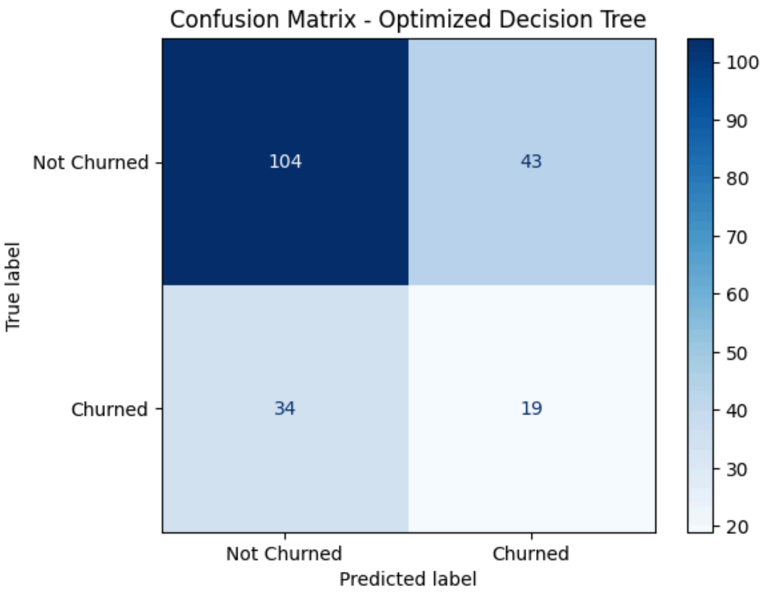
```
Correlation Matrix:
                           CustomerID       Age  Subscription_Length_Months  \
CustomerID                   1.000000 -0.011816                   -0.006179
Age                         -0.011816  1.000000                   -0.018938
Subscription_Length_Months  -0.006179 -0.018938                    1.000000
Watch_Time_Hours             0.032126 -0.046360                    0.024732
Number_of_Logins             0.025595  0.018846                   -0.056609
Payment_Issues               0.003177 -0.026649                   -0.024939
Number_of_Complaints         0.008697  0.059649                    0.015968
Resolution_Time_Days         0.062073 -0.001181                   -0.020476
Churn                       -0.005506  0.005136                   -0.033582

                           Watch_Time_Hours  Number_of_Logins  \
CustomerID                         0.032126          0.025595
Age                               -0.046360          0.018846
Subscription_Length_Months         0.024732         -0.056609
Watch_Time_Hours                   1.000000         -0.047729
Number_of_Logins                  -0.047729          1.000000
Payment_Issues                    -0.013810         -0.009978
Number_of_Complaints              -0.001512          0.004564
Resolution_Time_Days              -0.021412          0.025105
Churn                             -0.025224          0.062204

                           Payment_Issues  Number_of_Complaints  \
CustomerID                       0.003177              0.008697
Age                             -0.026649              0.059649
Subscription_Length_Months      -0.024939              0.015968
Watch_Time_Hours                -0.013810             -0.001512
Number_of_Logins                -0.009978              0.004564
Payment_Issues                   1.000000              0.023658
Number_of_Complaints             0.023658              1.000000
Resolution_Time_Days            -0.014245             -0.058042
Churn                            0.001193              0.004123

                           Resolution_Time_Days     Churn
CustomerID                             0.062073 -0.005506
Age                                   -0.001181  0.005136
Subscription_Length_Months            -0.020476 -0.033582
Watch_Time_Hours                      -0.021412 -0.025224
Number_of_Logins                       0.025105  0.062204
Payment_Issues                        -0.014245  0.001193
Number_of_Complaints                  -0.058042  0.004123
Resolution_Time_Days                   1.000000 -0.034181
Churn                                 -0.034181  1.000000
```



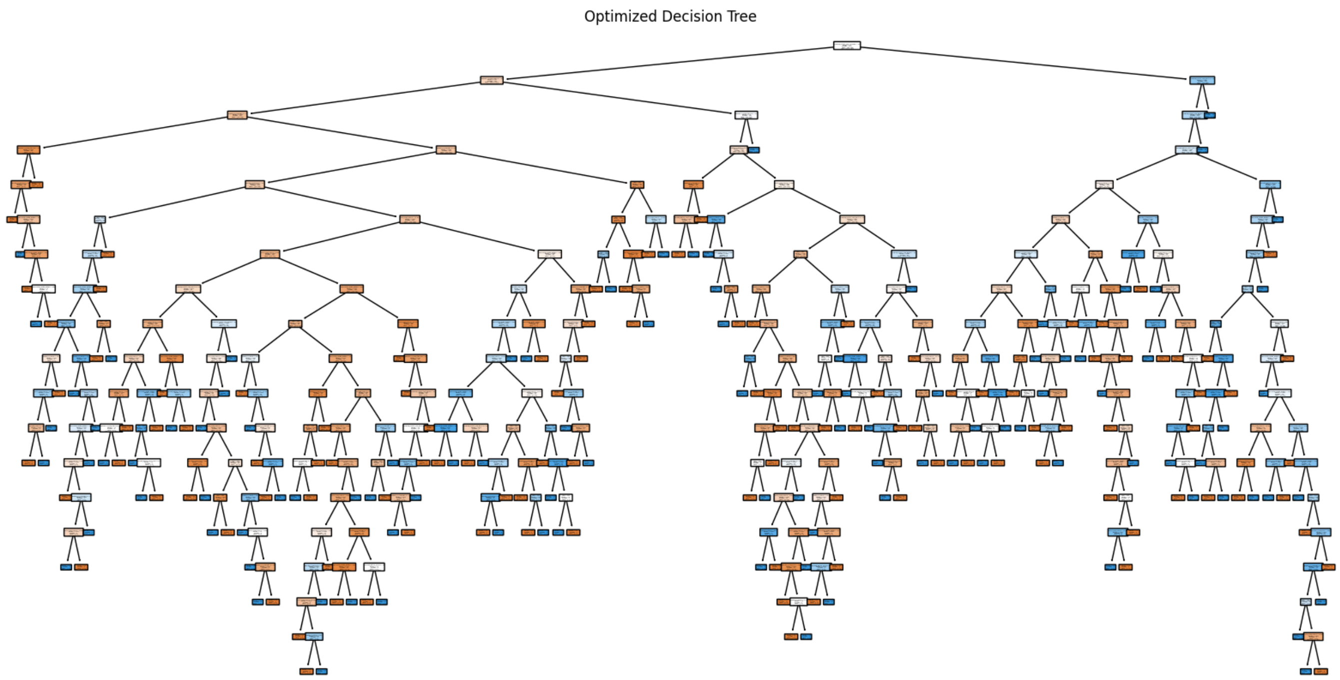
```
# TASK 2
# ---  Decision Tree  ---

# Features (X) and Target (y)
X = df.drop(["Churn", "CustomerID"], axis=1)
y = df["Churn"]

# One-Hot Encode Categorical Variables
X = pd.get_dummies(X, drop_first=True)

# Split 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, stratify=y)

# Address Class Imbalance Using SMOTE
smote_dt = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote_dt.fit_resample(X_train, y_train)

# Train a Decision Tree Classifier with Hyperparameter Tuning
param_grid_dt = {
    'max_depth': [3, 5, 10, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'criterion': ['gini', 'entropy']  # Add criterion as a parameter
}

# Perform GridSearchCV with F1 scoring
grid_search = GridSearchCV(DecisionTreeClassifier(random_state=42), param_grid_dt, cv=5, scoring='f1')
grid_search.fit(X_train_resampled, y_train_resampled)

# Get the Best Model
best_dt_model = grid_search.best_estimator_

# Predict on the Test Set
best_dt_prediction = best_dt_model.predict(X_test)

# Evaluate the Optimized Model
best_dt_accuracy = accuracy_score(y_test, best_dt_prediction)
best_dt_precision = precision_score(y_test, best_dt_prediction)
best_dt_recall = recall_score(y_test, best_dt_prediction)
best_dt_f1 = f1_score(y_test, best_dt_prediction)

# Print Evaluation Metrics
print("\nOptimized Decision Tree Performance Metrics:")
print("Accuracy:", best_dt_accuracy)
print("Precision:", best_dt_precision)
print("Recall:", best_dt_recall)
print("F1 Score:", best_dt_f1)
```

```
Optimized Decision Tree Performance Metrics:
Accuracy: 0.615
Precision: 0.3064516129032258
Recall: 0.3584905660377358
F1 Score: 0.33043478260869563
```

```
# Confusion Matrix - Decision Tree
conf_matrix_dt = confusion_matrix(y_test, best_dt_prediction)
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix_dt, display_labels=["Not Churned", "Churned"])
disp.plot(cmap="Blues")
plt.title("Confusion Matrix - Optimized Decision Tree")
plt.show()
```




```
# Visualize the Decision Tree
plt.figure(figsize=(20, 10))
plot_tree(best_dt_model, feature_names=X.columns, class_names=["Not Churned", "Churned"], filled=True, rounded=True)
plt.title("Optimized Decision Tree")
plt.show()
```

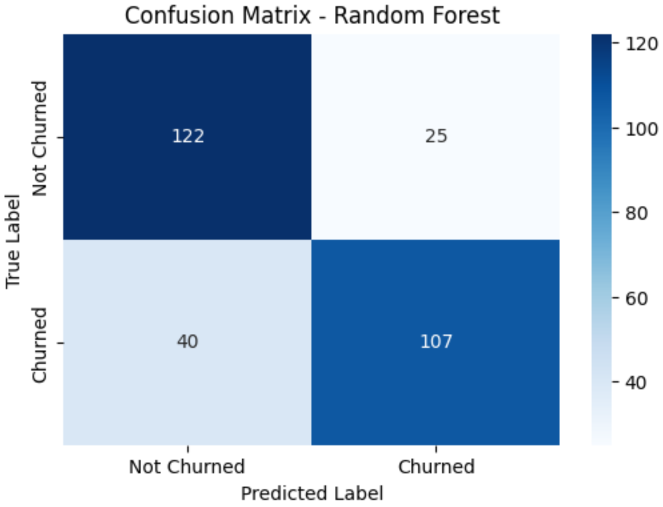



```
# TASK 3
# ---  Random Forest  ---
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from imblearn.over_sampling import SMOTE  # For handling class imbalance

# Handle class imbalance using SMOTE
sm = SMOTE(random_state=42)
X_res, y_res = sm.fit_resample(X, y)

# Split resampled data
X_train, X_test, y_train, y_test = train_test_split(
    X_res, y_res,
```

```
        test_size=0.2,
        stratify=y_res,
        random_state=42
    )

    # Hyperparameter Tuning
    param_grid = {
        'n_estimators': [100, 200],
        'max_depth': [None, 10, 20],
        'min_samples_split': [2, 5],
        'class_weight': ['balanced', None]
    }

    rf = RandomForestClassifier(random_state=42)
    grid_search = GridSearchCV(rf, param_grid, cv=5, scoring='recall', n_jobs=-1)
    grid_search.fit(X_train, y_train)

    best_rf = grid_search.best_estimator_

    # Evaluation the model
    y_pred_rf = best_rf.predict(X_test)
    y_proba_rf = best_rf.predict_proba(X_test)[:,1]

    accuracy_rf = accuracy_score(y_test, y_pred_rf)
    precision_rf = precision_score(y_test, y_pred_rf)
    recall_rf = recall_score(y_test, y_pred_rf)
    f1_rf = f1_score(y_test, y_pred_rf)

    print("Random Forest Performance Metrics:")
    print("Accuracy: ", accuracy_rf)
    print("Precision: ", precision_rf)
    print("Recall: ", recall_rf)
    print("F1 Score: ", f1_rf)
```

```
Random Forest Performance Metrics:
Accuracy:  0.7789115646258503
Precision:  0.8106060606060606
Recall:  0.7278911564625851
F1 Score:  0.7670250896057348
```

```
# Confusion matrix - Random Forest
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)

# Create heatmap
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix_rf, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Churned', 'Churned'], yticklabels=['Not Chu
plt.title('Confusion Matrix - Random Forest')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.show()
```




```
# PERFORMANCE COMPARISON

# Performance Metrics for Decision Tree
dt_metrics = {
    "Accuracy": best_dt_accuracy,
    "Precision": best_dt_precision,
    "Recall": best_dt_recall,
    "F1 Score": best_dt_f1
```

```
}

# Performance Metrics for Random Forest
rf_metrics = {
    "Accuracy": accuracy_rf,
    "Precision": precision_rf,
    "Recall": recall_rf,
    "F1 Score": f1_rf
}

# Create a bar chart for comparison
labels = list(dt_metrics.keys())
dt_values = list(dt_metrics.values())
rf_values = list(rf_metrics.values())

x = np.arange(len(labels))  # Label locations
width = 0.35  # Width of the bars

fig, ax = plt.subplots(figsize=(10, 6))
rects1 = ax.bar(x - width/2, dt_values, width, label='Decision Tree', color='skyblue'
rects2 = ax.bar(x + width/2, rf_values, width, label='Random Forest', color='orange')

# Add some text for labels, title, and custom x-axis tick labels
ax.set_ylabel('Scores')
ax.set_title('Comparison of Decision Tree and Random Forest Performance Metrics')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()

# Add value labels on top of each bar
def add_labels(rects):
    for rect in rects:
        height = rect.get_height()
        ax.annotate(f'{height:.2f}',
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')

add_labels(rects1)
add_labels(rects2)

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