In [1]:
```
#importing libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
C:\Users\CODEINE\AppData\Local\Temp\ipykernel_8516\4288724001.py:2: DeprecationWa
rning:
Pyarrow will become a required dependency of pandas in the next major release of
pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and better i
nteroperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/54466

  import pandas as pd
```

In [2]:
```
#loading data
df = pd.read_csv("Flyzy Flight Cancellation.csv")
```

In [3]:
```
df.head()
```

Out[3]:

	Flight ID	Airline	Flight_Distance	Origin_Airport	Destination_Airport	Scheduled_Dep
0	7319483	Airline D	475	Airport 3	Airport 2	
1	4791965	Airline E	538	Airport 5	Airport 4	
2	2991718	Airline C	565	Airport 1	Airport 2	
3	4220106	Airline E	658	Airport 5	Airport 3	
4	2263008	Airline E	566	Airport 2	Airport 2	

In [4]:
```
df.tail()
```

Out[4]:

	Flight ID	Airline	Flight_Distance	Origin_Airport	Destination_Airport	Scheduled_I
2995	1265781	Airline D	395	Airport 2	Airport 3	
2996	5440150	Airline E	547	Airport 1	Airport 4	
2997	779080	Airline C	461	Airport 1	Airport 3	
2998	4044431	Airline B	464	Airport 3	Airport 3	
2999	2806578	Airline A	369	Airport 1	Airport 2	

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

Out[5]: (3000, 14)

In [6]:
```
df.columns
```

Out[6]:
```
Index(['Flight ID', 'Airline', 'Flight_Distance', 'Origin_Airport',
       'Destination_Airport', 'Scheduled_Departure_Time', 'Day_of_Week',
       'Month', 'Airplane_Type', 'Weather_Score',
       'Previous_Flight_Delay_Minutes', 'Airline_Rating', 'Passenger_Load',
       'Flight_Cancelled'],
      dtype='object')
```

Observation: Most of the column names consist of multiple words seperated by underscores,but 'Flight ID' does not follow this format, therefore we need to change it to keep consistancy.

In [7]:
```
#Changing column name
df.rename(columns={'Flight ID' : 'Flight_ID'}, inplace =True)
```

In [8]:
```
df.head(2)
```

Out[8]:

	Flight_ID	Airline	Flight_Distance	Origin_Airport	Destination_Airport	Scheduled_Dep
0	7319483	Airline D	475	Airport 3	Airport 2	
1	4791965	Airline E	538	Airport 5	Airport 4	

CHECKING DATA TYPES OF EACH COLUMN

In [9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 14 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Flight_ID                     3000 non-null   int64  
 1   Airline                       3000 non-null   object 
 2   Flight_Distance               3000 non-null   int64  
 3   Origin_Airport                3000 non-null   object 
 4   Destination_Airport           3000 non-null   object 
 5   Scheduled_Departure_Time      3000 non-null   int64  
 6   Day_of_Week                   3000 non-null   int64  
 7   Month                         3000 non-null   int64  
 8   Airplane_Type                 3000 non-null   object 
 9   Weather_Score                 3000 non-null   float64
 10  Previous_Flight_Delay_Minutes 3000 non-null   float64
 11  Airline_Rating                3000 non-null   float64
 12  Passenger_Load                3000 non-null   float64
 13  Flight_Cancelled              3000 non-null   int64  
dtypes: float64(4), int64(6), object(4)
memory usage: 328.3+ KB
```

Observation: This results indicate that all columns have the correct data types according to the data they contain

In [10]:
```
#Checking for duplicates entries
duplicates = df[df.duplicated()]
```

In [11]:
```
duplicates
```

Out[11]:

Flight_ID	Airline	Flight_Distance	Origin_Airport	Destination_Airport	Scheduled_Depa

No duplicates on the dataset

CHECKING FOR MISSING VALUES

In [12]:
```
df.isnull().sum()
```

Out[12]:
```
Flight_ID                        0
Airline                          0
Flight_Distance                  0
Origin_Airport                   0
Destination_Airport              0
Scheduled_Departure_Time         0
Day_of_Week                      0
Month                            0
Airplane_Type                    0
Weather_Score                    0
Previous_Flight_Delay_Minutes    0
Airline_Rating                   0
Passenger_Load                   0
Flight_Cancelled                 0
dtype: int64
```

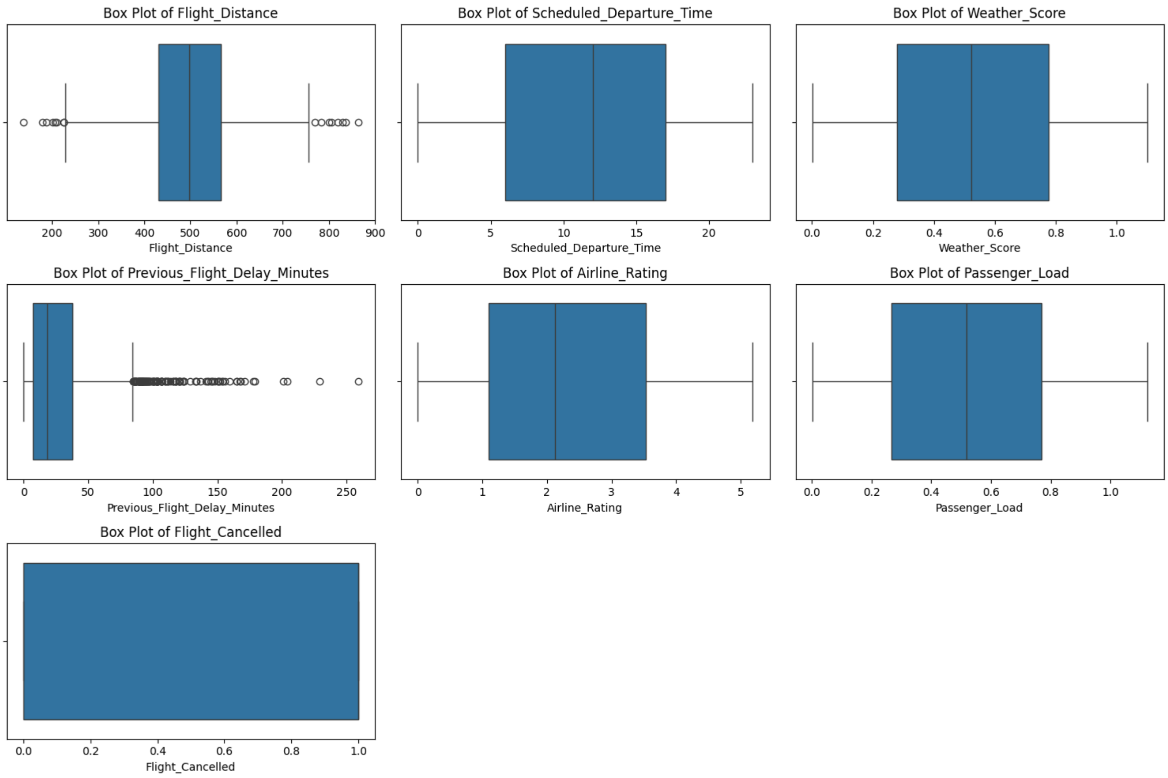
There are no missing values

CHECKING FOR OUTLIERS

In [13]:
```
#Used boxplot to visually check outliers
```

In [14]:
```
columns_to_check =['Flight_Distance',
           'Scheduled_Departure_Time',
           'Weather_Score',
           'Previous_Flight_Delay_Minutes',
           'Airline_Rating', 'Passenger_Load',
           'Flight_Cancelled'
          ]
```

In [15]:
```
plt.figure(figsize =(15,10))
for i, col in enumerate(columns_to_check, 1):
    plt.subplot(3,3, i)
    sns.boxplot(x=df[col])
    plt.title(f'Box Plot of {col}')
plt.tight_layout()
plt.show()
```



This plots shows that the following columns have outliers and have to be handled

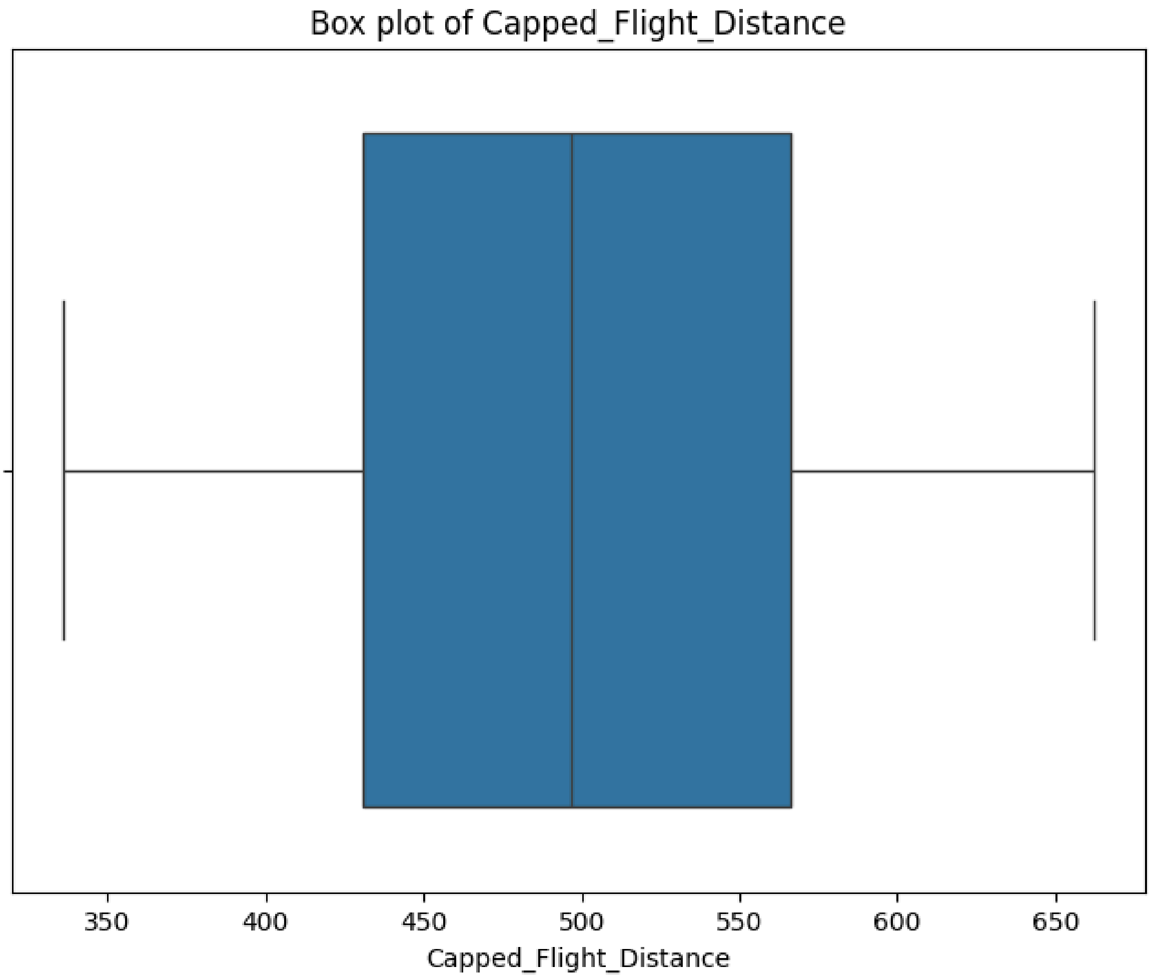
1. Flight_Distance
2. Previous_Flight_Delay_Minutes

1. Handling outliers for Flight_Distance column using Capping method Because it reduces the impact of extreme outliers, which can distort the analysis.

In [16]:
```
#Handling outliers on Flight Distance using Capping approach

#applying threshold
cap_max = df['Flight_Distance'].quantile(0.95)
```

```
cap_min =df['Flight_Distance'].quantile(0.05)
#Apply capping
df['Capped_Flight_Distance'] = np.clip(df['Flight_Distance'], cap_min, cap_max)
```

In [17]:
```
#Plotting transformed column
plt.figure(figsize =(8,6))
sns.boxplot(x=df['Capped_Flight_Distance'])
plt.title('Box plot of Capped_Flight_Distance')
plt.show()
```

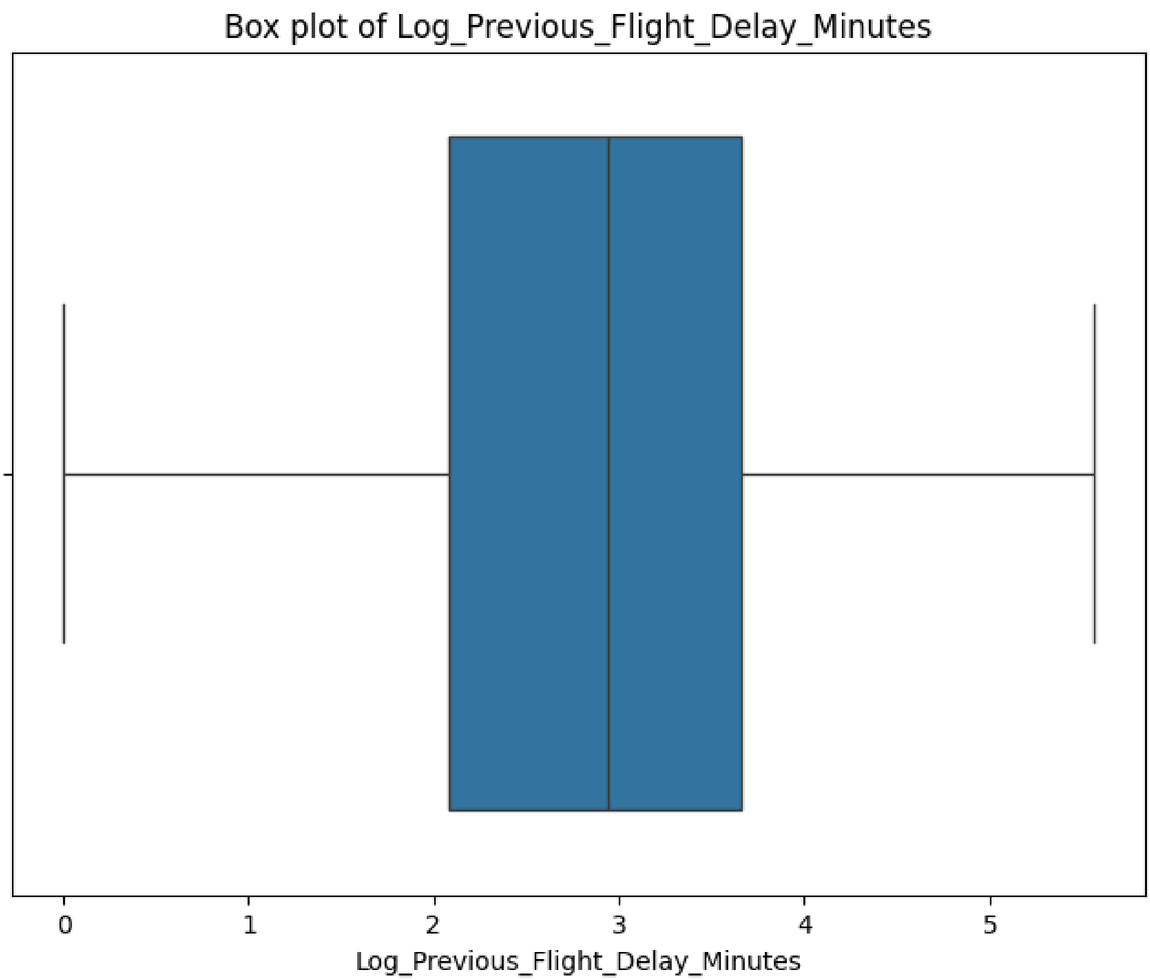


The results shows no more outliers for Flight_Distance

2. Handling Outliers for Previous_Flight_Delay_Minutes Using Log Transformation because data is skewed, compressing the range of delay times, reducing the impact of extreme values.

In [18]:
```
#creating a new column and applying the log
df['Log_Previous_Flight_Delay_Minutes'] = np.log1p(df['Previous_Flight_Delay_Min
```

In [19]:
```
#Plotting transformed column
plt.figure(figsize =(8,6))
sns.boxplot(x=df['Log_Previous_Flight_Delay_Minutes'])
plt.title('Box plot of Log_Previous_Flight_Delay_Minutes')
plt.show()
```



Now the outliers were handled and not showing on the plot

In [20]:
```
df.head(2)
```

Out[20]:

	Flight_ID	Airline	Flight_Distance	Origin_Airport	Destination_Airport	Scheduled_De
0	7319483	Airline D	475	Airport 3	Airport 2	
1	4791965	Airline E	538	Airport 5	Airport 4	

In [21]:
```
df.shape
```

Out[21]: (3000, 16)

EXPLORATORY DATA ANALYSIS

DESCRIPTIVE STATISTICS

In [22]:
```
df.describe()
```

Out[22]:

	Flight_ID	Flight_Distance	Scheduled_Departure_Time	Day_of_Week	Mo
count	3.000000e+03	3000.000000	3000.000000	3000.000000	3000.000
mean	4.997429e+06	498.909333	11.435000	3.963000	6.381
std	2.868139e+06	98.892266	6.899298	2.016346	3.473
min	3.681000e+03	138.000000	0.000000	1.000000	1.000
25%	2.520313e+06	431.000000	6.000000	2.000000	3.000
50%	5.073096e+06	497.000000	12.000000	4.000000	6.000
75%	7.462026e+06	566.000000	17.000000	6.000000	9.000
max	9.999011e+06	864.000000	23.000000	7.000000	12.000

The above shows statistical analysis for various features.

Examples,We can see the longest flight delay of 250 minutes from the previous flight delay minutes and also highest and lowest Airline ratings amongst other observations

DATA DISTRIBUTION

Numerical Columns

In [23]:

```
df.columns
```

Out[23]:

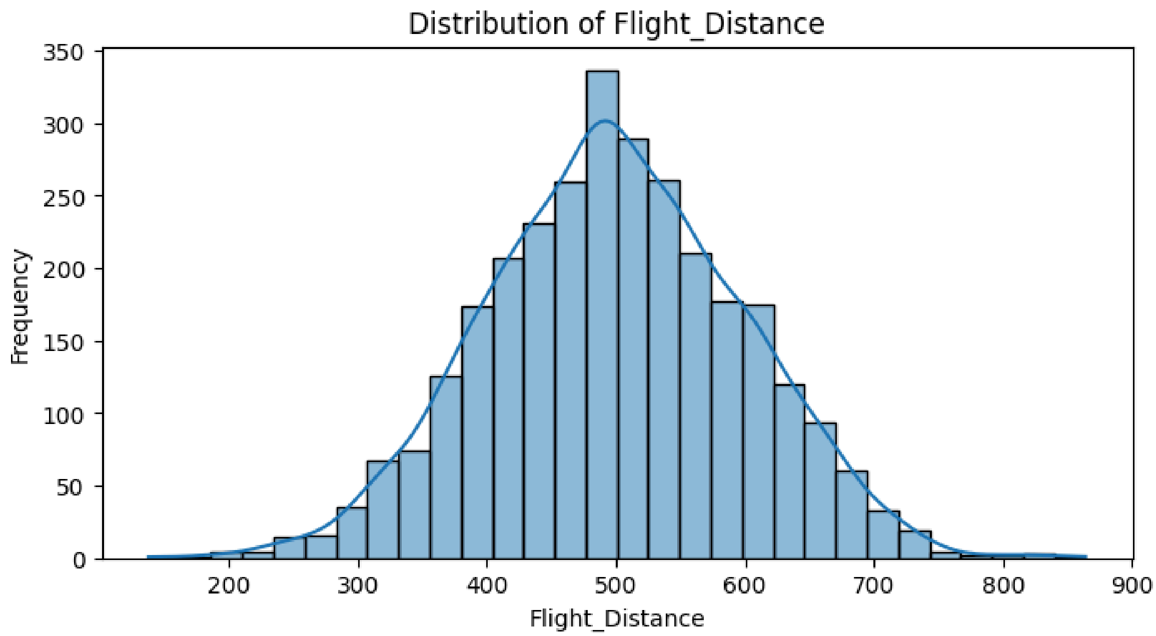
```
Index(['Flight_ID', 'Airline', 'Flight_Distance', 'Origin_Airport',
       'Destination_Airport', 'Scheduled_Departure_Time', 'Day_of_Week',
       'Month', 'Airplane_Type', 'Weather_Score',
       'Previous_Flight_Delay_Minutes', 'Airline_Rating', 'Passenger_Load',
       'Flight_Cancelled', 'Capped_Flight_Distance',
       'Log_Previous_Flight_Delay_Minutes'],
      dtype='object')
```

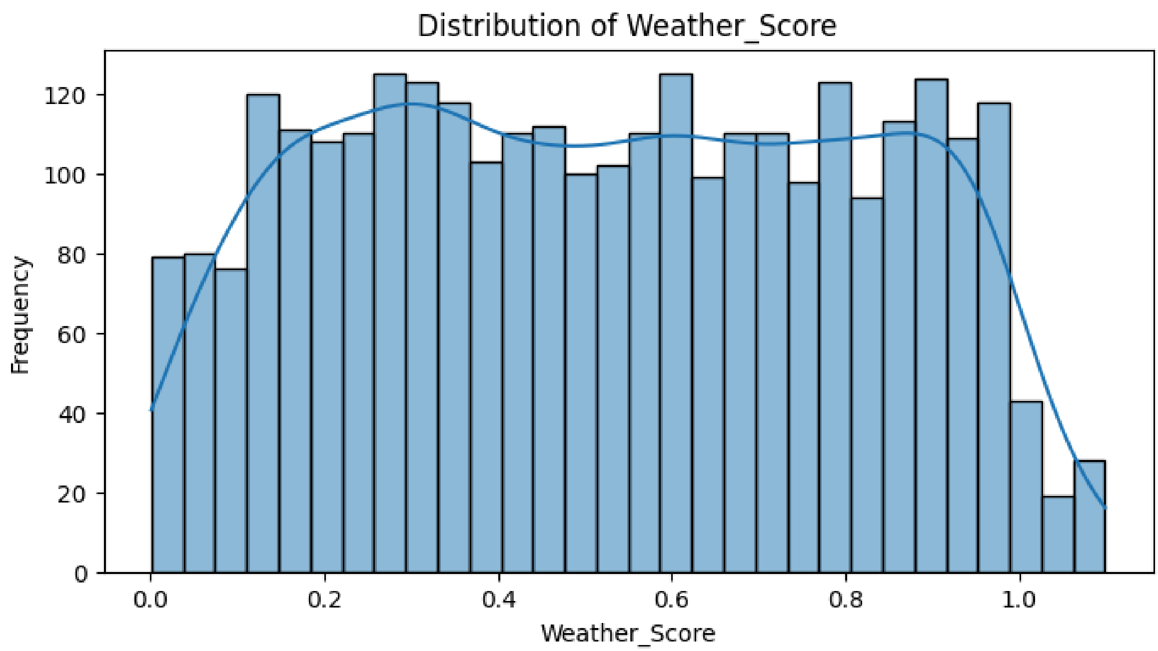
In [24]:

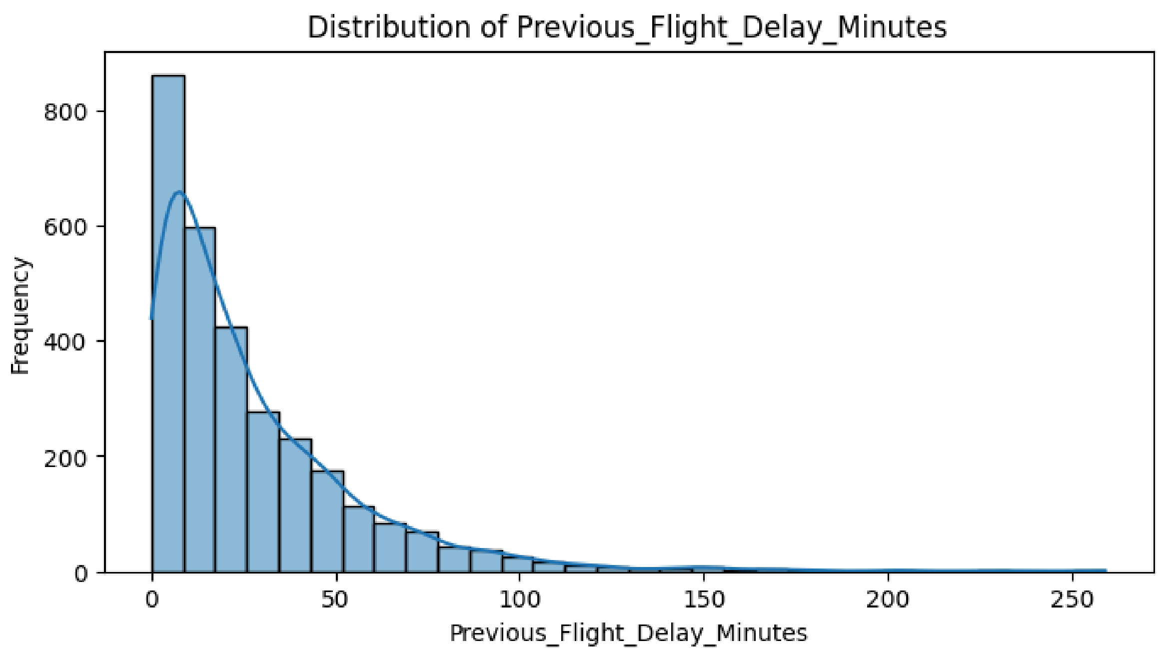
```
#Selecting relevant numerical columns
numerical_columns = [
    'Flight_Distance', 'Weather_Score', 'Previous_Flight_Delay_Minutes',
    'Airline_Rating', 'Passenger_Load', 'Capped_Flight_Distance',
    'Log_Previous_Flight_Delay_Minutes'
]
```

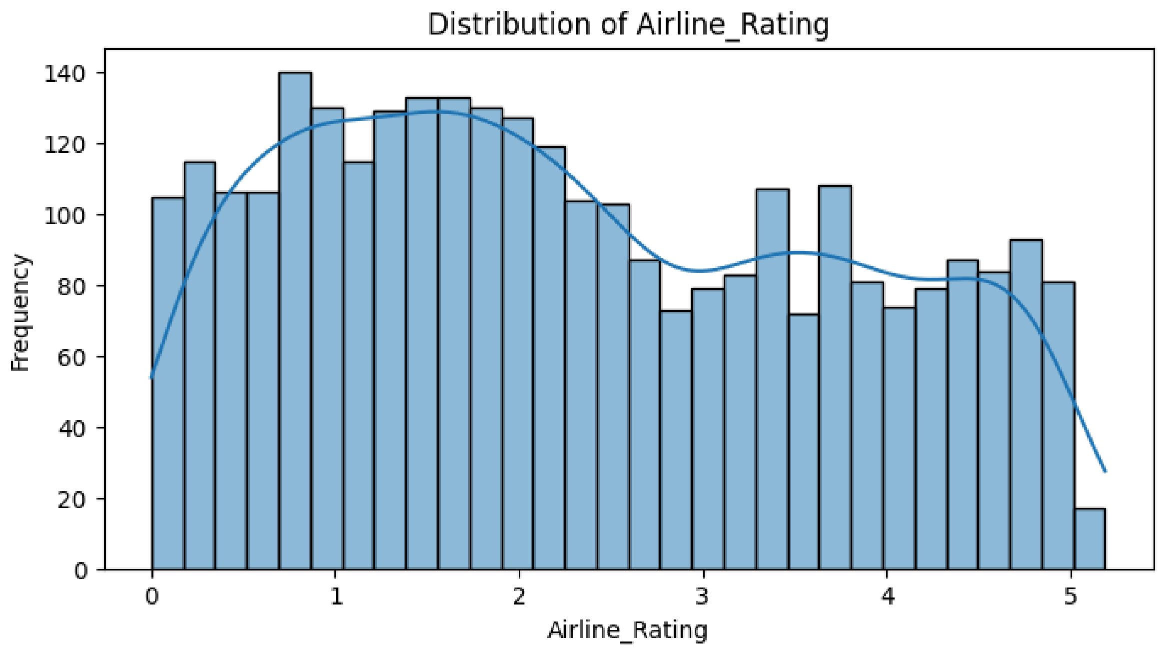
In [25]:

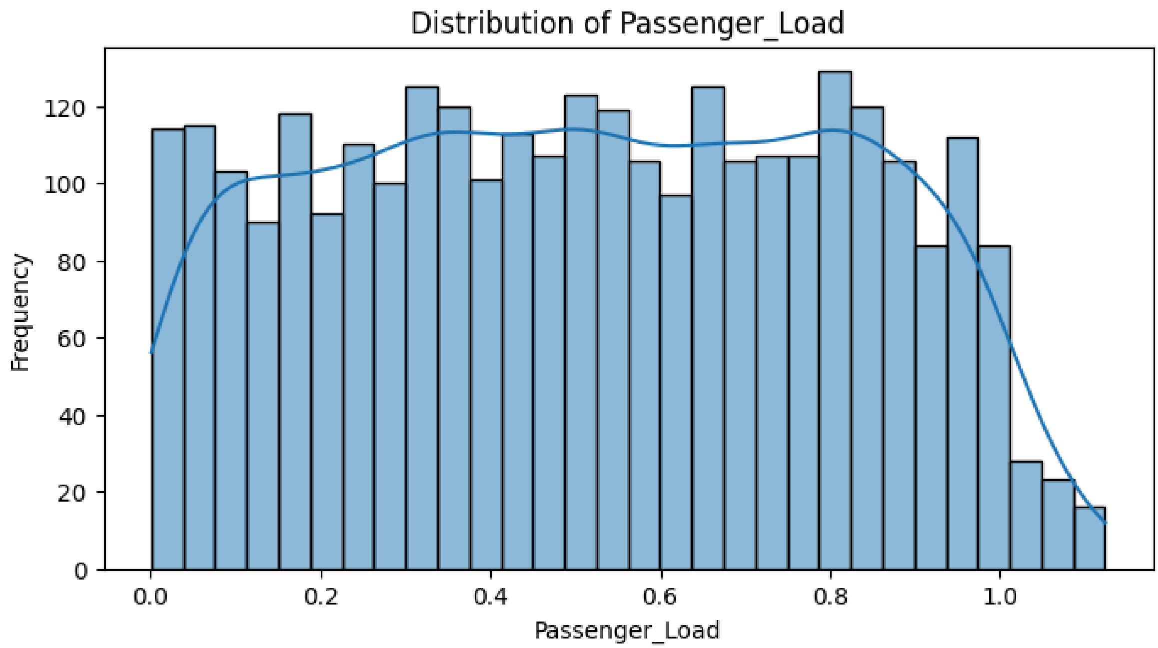
```
# Plotting the distribution for each numerical column
for col in numerical_columns:
    plt.figure(figsize=(8, 4))
    sns.histplot(df[col], kde=True, bins=30)  #
    plt.title(f'Distribution of {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.show()
```

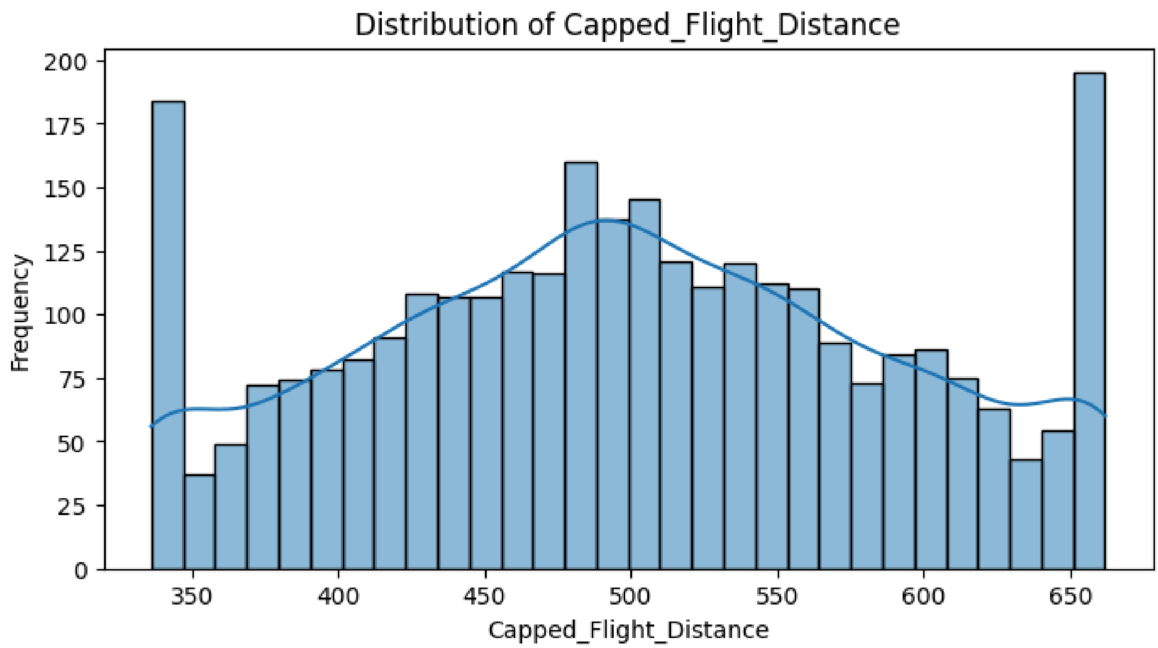

Distribution of Flight_Distance
Frequency
Flight_Distance
0
50
100
150
200
250
300
350
200
300
400
500
600
700
800
900

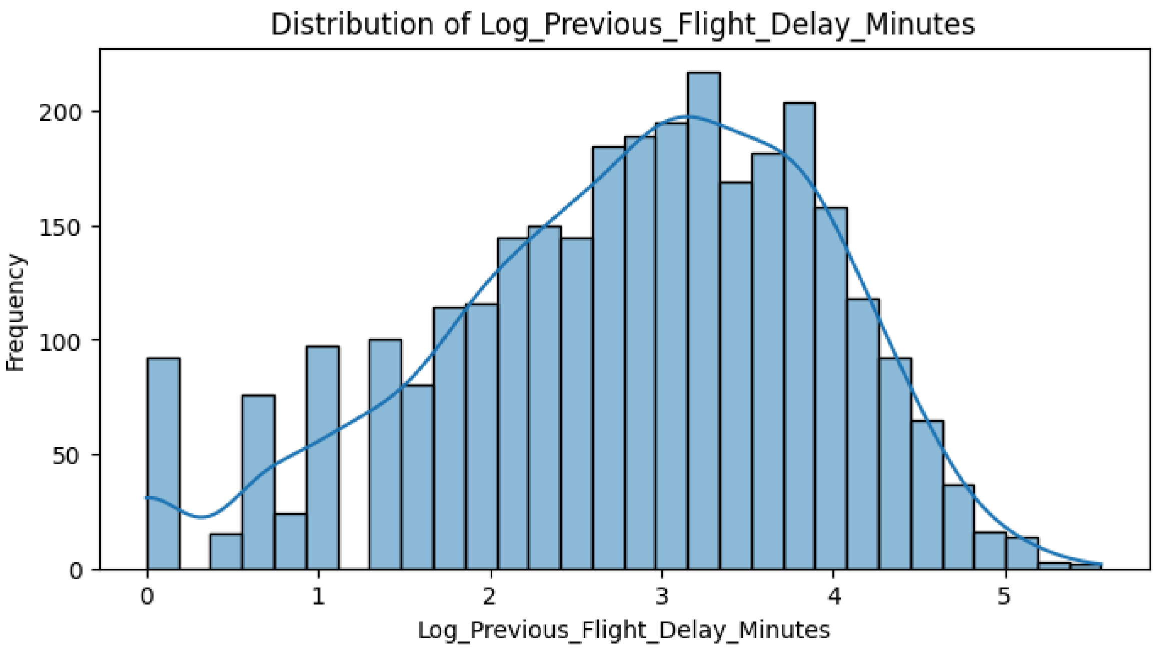

Distribution of Weather_Score
Frequency
Weather_Score
0
20
40
60
80
100
120
0.0
0.2
0.4
0.6
0.8
1.0


Distribution of Previous_Flight_Delay_Minutes
Frequency
Previous_Flight_Delay_Minutes
0
200
400
600
800
0
50
100
150
200
250


Distribution of Airline_Rating
Frequency
0
20
40
60
80
100
120
140
0
1
2
3
4
5
Airline_Rating


Distribution of Passenger_Load
Frequency
0
20
40
60
80
100
120
0.0
0.2
0.4
0.6
0.8
1.0
Passenger_Load

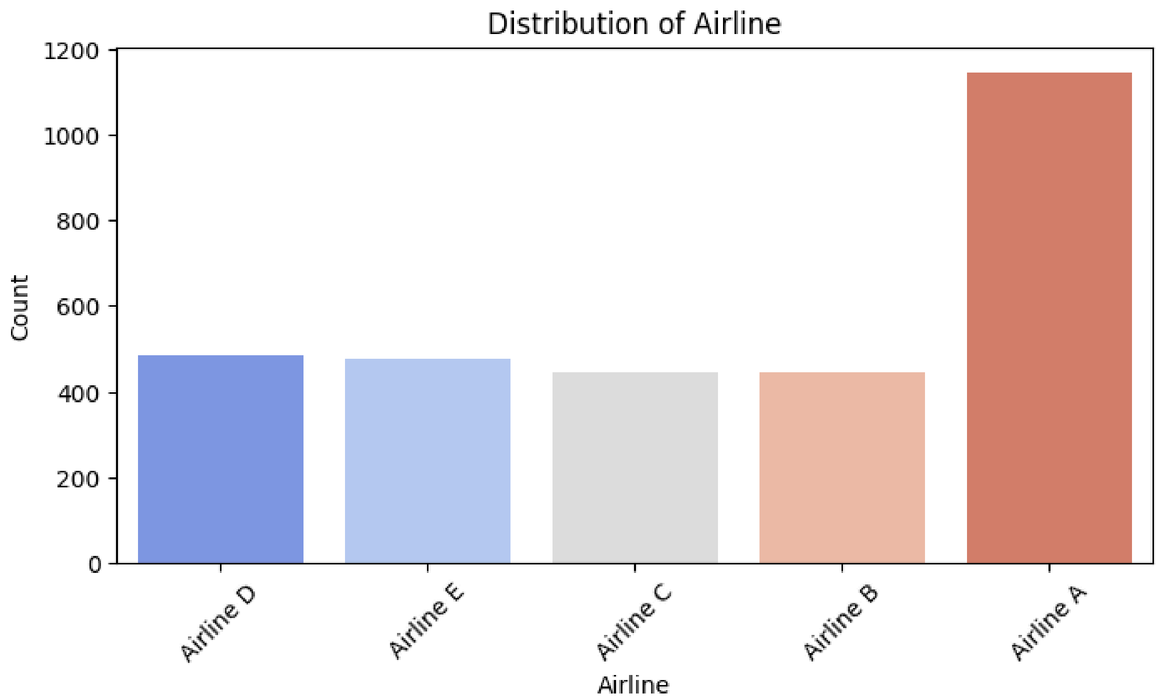

Distribution of Capped_Flight_Distance
Frequency
0
25
50
75
100
125
150
175
200
350
400
450
500
550
600
650
Capped_Flight_Distance

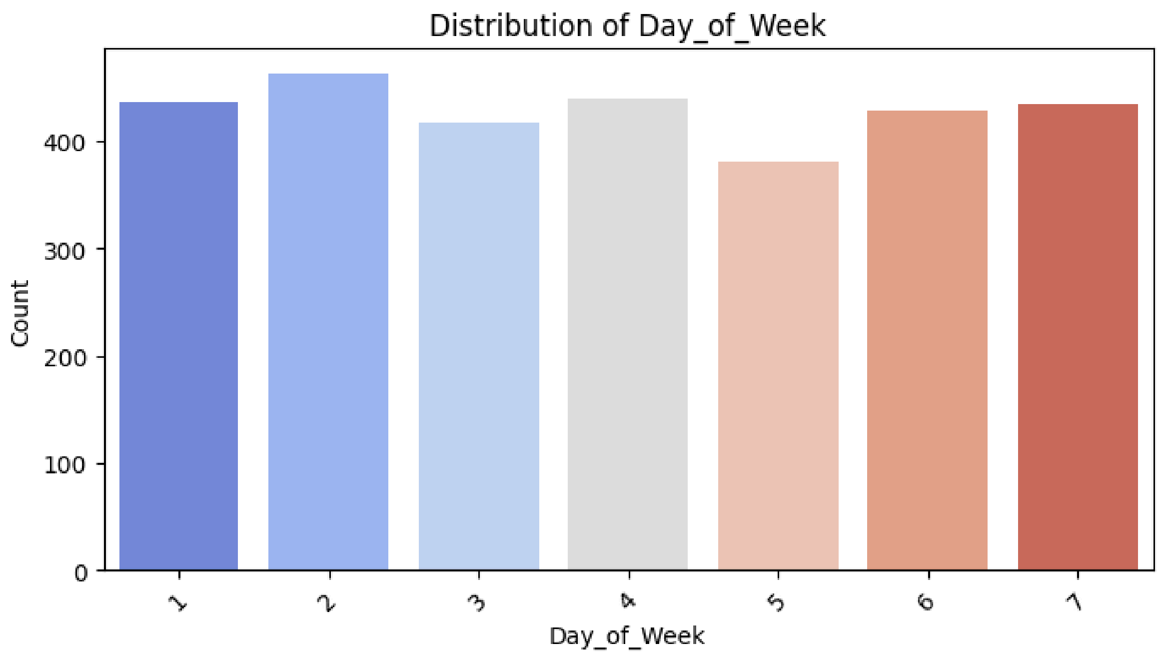


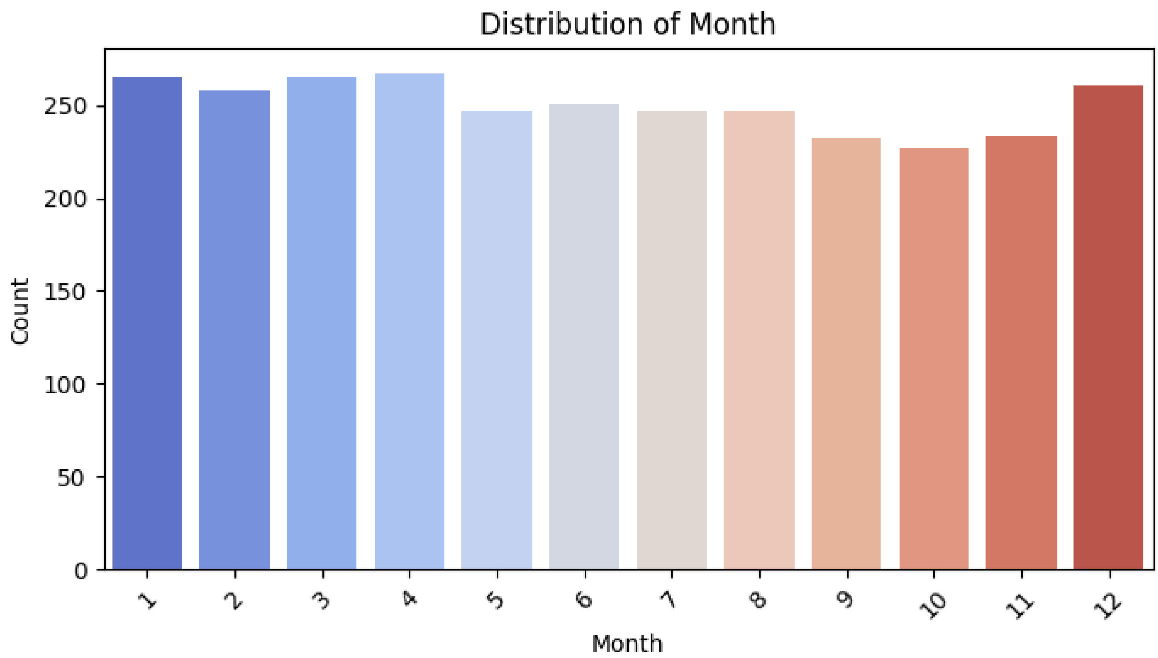
From this , we can see various distribution of the numerical columns

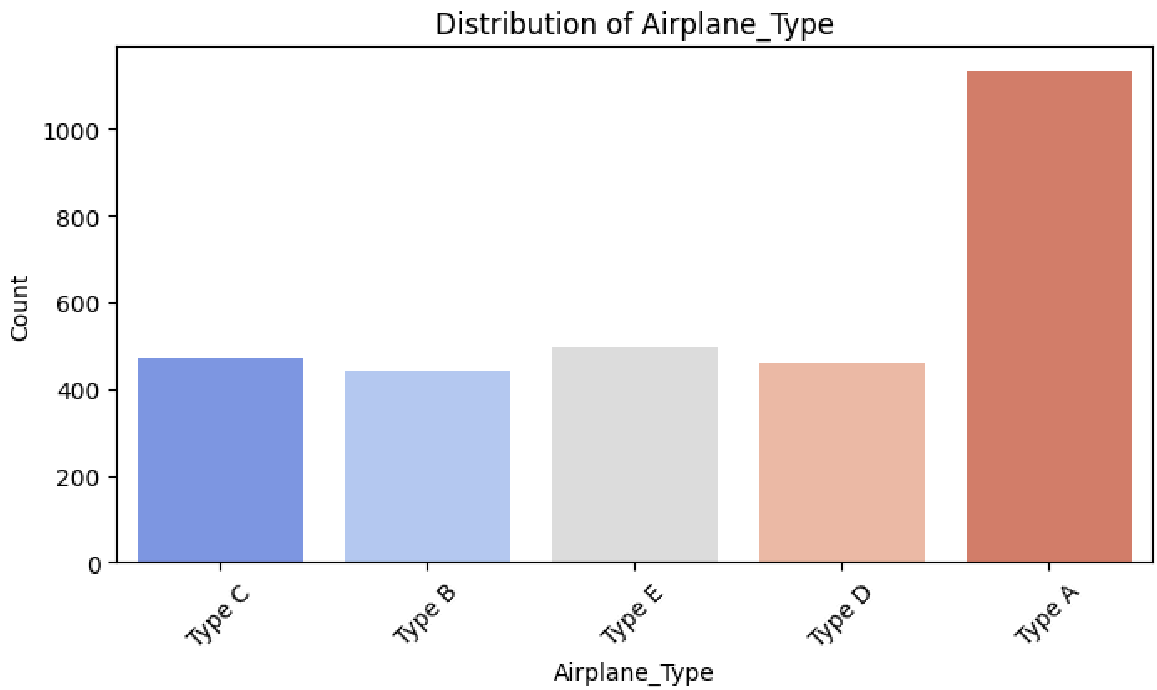
Now checking categorial columns

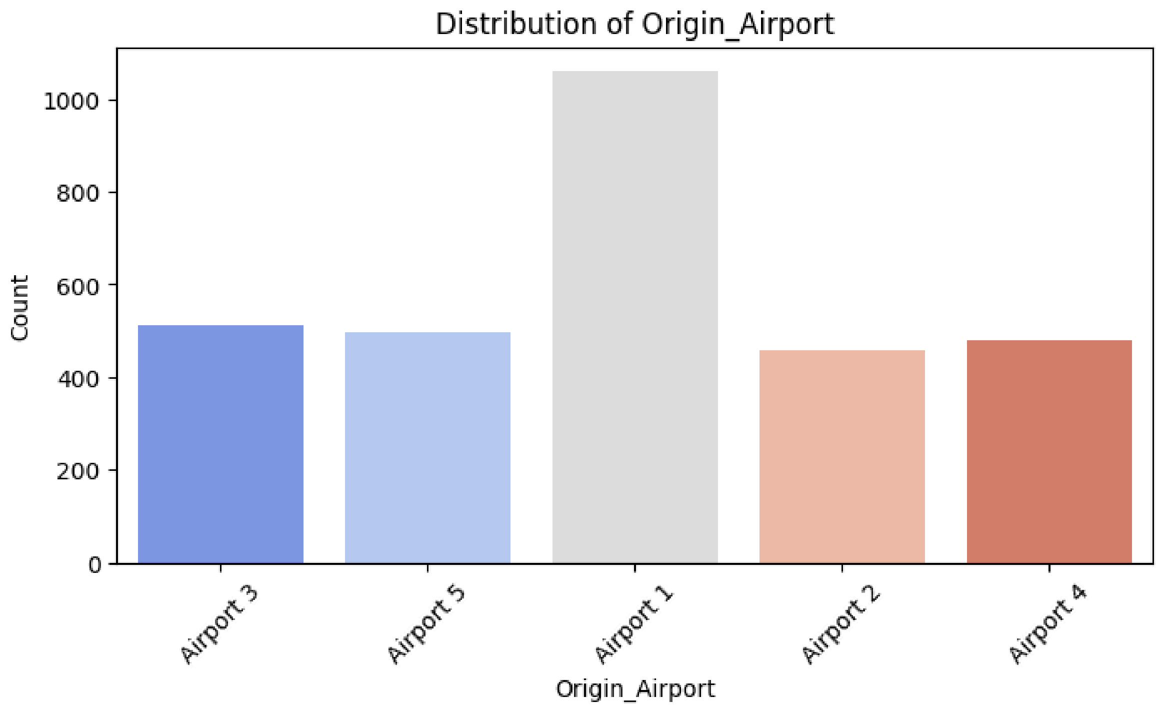
In [26]:
```python
import warnings
warnings.filterwarnings("ignore")
#selecting columns to check
categorical_columns = ['Airline','Day_of_Week', 'Month', 'Airplane_Type','Origin

# Plotting the distribution for each categorical column
for col in categorical_columns:
    plt.figure(figsize=(8, 4))
    sns.countplot(x=df[col], palette='coolwarm')  # Countplot to show the freque
    plt.title(f'Distribution of {col}')
    plt.xlabel(col)
    plt.ylabel('Count')
    plt.xticks(rotation=45)  # Rotate x labels if needed for readability
    plt.show()
```

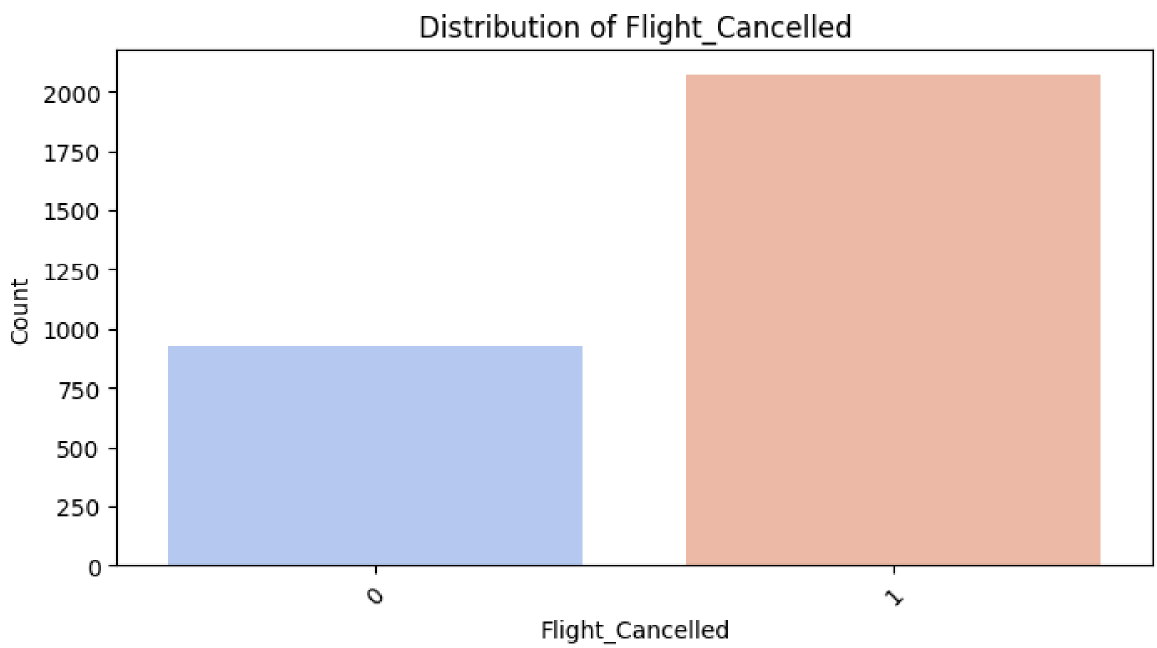

Distribution of Airline
Count
1200
1000
800
600
400
200
0
Airline D
Airline E
Airline C
Airline B
Airline A
Airline


Distribution of Day_of_Week
Count
400
300
200
100
0
1
2
3
4
5
6
7
Day_of_Week


Distribution of Month
Count
0
50
100
150
200
250
1
2
3
4
5
6
7
8
9
10
11
12
Month


Distribution of Airplane_Type
Count
0
200
400
600
800
1000
Type C
Type B
Type E
Type D
Type A
Airplane_Type





Data from Flight_Cancelled target column is highly imbalanced, we have more occurances of cancelled flights than non cancelled.

There is also a noticeable high occurance of :

TypeA flight than other flights

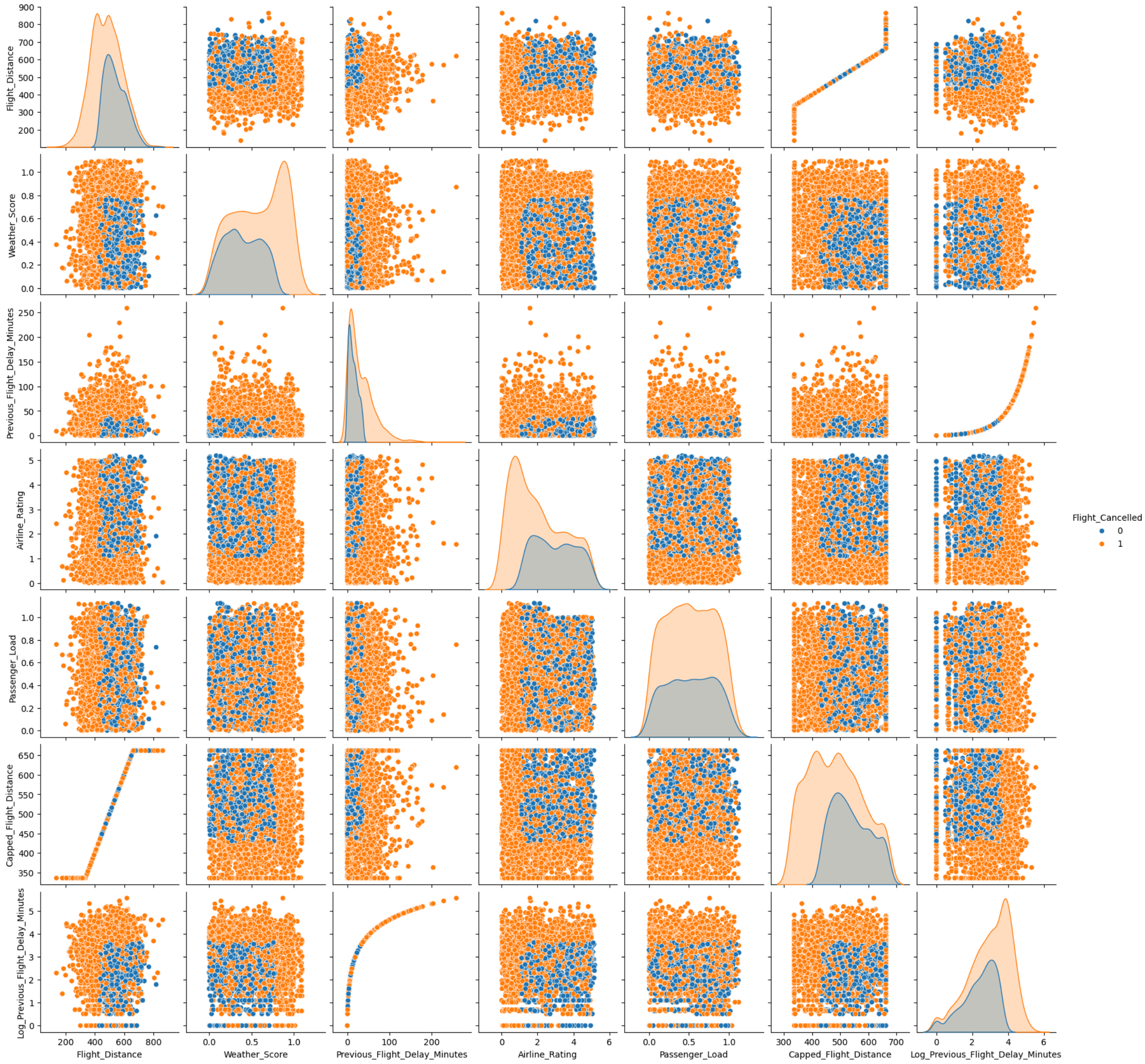
AirlineA than other airlines

Airport1 than other airports

Month and day of the week: there is slight difference between various months and various days of the week

RELATIONSHIP BETWEEN FEATURES:

In [27]:
```
#pairplots
sns.pairplot(df, hue ='Flight_Cancelled', vars=numerical_columns)
plt.show()
```



In [28]:
```
#Correlation matrix to understand the relationships better
df.select_dtypes(include ="number").corr()
```

Out[28]:

	Flight_ID	Flight_Distance	Scheduled_Departure_Time
Flight_ID	1.000000	-0.007541	0.006207
Flight_Distance	-0.007541	1.000000	0.039727
Scheduled_Departure_Time	0.006207	0.039727	1.000000
Day_of_Week	-0.012384	0.024455	-0.011834
Month	-0.025743	0.019573	0.018319
Weather_Score	-0.002007	0.010139	-0.023682
Previous_Flight_Delay_Minutes	0.006172	0.018413	-0.036318
Airline_Rating	0.043170	0.042128	0.040739
Passenger_Load	0.009312	-0.018627	0.046556
Flight_Cancelled	-0.009101	-0.277471	-0.043733
Capped_Flight_Distance	-0.007407	0.988170	0.033209
Log_Previous_Flight_Delay_Minutes	0.011250	0.008974	-0.032594

In [29]:

```
#Visualising the relationships using Heatmap
plt.figure(figsize =(12,10))
sns.heatmap(df.select_dtypes(include ="number").corr(), annot = True)
```

Out[29]: <Axes: >

Relationship Insights:

Flight_Distance & Capped_Flight_Distance are strongly correlated since"Capped_Flight_Distance" is derived from "Flight_Distance" while managing outliers.

Similarly for Previous_Flight_Delay_Minutes & Log_Previous_Flight_Delay_Minutes (0.827)

Flight_Cancelled & Weather_Score (0.306): There's a moderate positive correlation indicating that poor weather may be associated with more cancellations.

Flight_Cancelled & Previous_Flight_Delay_Minutes (0.303): A moderate positive correlation suggesting that flights with previous delays might have a higher chance of being cancelled.

Scheduled_Departure_Time & Passenger_Load (0.047): Slight positive correlation but not strong.

Flight_Distance & Passenger_Load (-0.0186): A weak negative correlation

Airline_Rating & Previous_Flight_Delay_Minutes (-0.0360): A weak negative correlation, indicating that higher delays are not strongly related to airline ratings.

Airline_Rating & Flight_Cancelled (-0.314): A moderate negative correlation, indicating that lower airline ratings might be associated with higher chances of cancellation.

Day_of_Week, Month: weak correlations with other variables, indicating that the day of the week and month might not have strong impacts on flight cancellations.

RELATIONSHIP BETWEEN FEATURES AND TARGET COLUMN

Based on the investigations from the Correlation matrix and Heatmap above, below are the observations:

Flight_Distance - there is a moderate negative correlation between Flight_Distance and Flight_Cancelled, suggesting flight distance might not be highly influencial

Scheduled_Departure_Time , correlation is very close to zero indicating weak correlation with Flight_Cancelled, therefore might not be a significant predictor for flight cancellations

Day_of_Week and Month, also shows weak correlation with the target column , suggesting not much impact on flight cancellations

Weather_Score, shows moderate postive correlation with target column indicating worse weather conditions can influence flight cancellations, this is very important for predicting cancellations

Previous_Flight_Minutes_Minutes, there is moderate postive correlation suggesting previous delays could influence cancellations

Airline_Rating, shows moderate negative correlation with the target, it indicate flights with lower airline rating are likely to be cancelled

Passenger_Load, weak correlation indicating number of passengers not impacting flight cancellations

Capped_Flight_Distance, similar to original Flight_Distance, moderate negative correlation

Log_Transformed_Delay_Minutes, also similar to the original Previous_Flight_Delay_Minutes although slihtly lower than orignal but it still indicates delays are positively associated with cancellations.

TASK 3 : DATA PRE-PROCESSING AND MODEL BUILDING

In [33]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder

#Dropping original columns (to use only the transformed ones) and also target co
#
X =df.drop(['Flight_Cancelled', 'Flight_ID','Flight_Distance','Previous_Flight_D

#Target variable
y = df['Flight_Cancelled']
```

In [38]:
```
# Split the data first into training and test sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

In [44]:
```
#Seperating numerical and categorical features
numerical_columns = ['Capped_Flight_Distance', 'Scheduled_Departure_Time','Weath
                     'Log_Previous_Flight_Delay_Minutes', 'Airline_Rating', 'Pas
categorical_columns = ['Airline', 'Origin_Airport', 'Destination_Airport', 'Airp
```

In [45]:
```
# Scale numerical features only on training data
scaler = StandardScaler()
X_train_numerical_scaled = scaler.fit_transform(X_train[numerical_columns])  # F
X_test_numerical_scaled = scaler.transform(X_test[numerical_columns])        # C
```

In [49]:
```
# Encode categorical features only on training data
encoder = OneHotEncoder(handle_unknown='ignore', sparse_output=False)
X_train_categorical_encoded = encoder.fit_transform(X_train[categorical_columns]
X_test_categorical_encoded = encoder.transform(X_test[categorical_columns])
```

In [52]:
```
# Ensure consistency in feature names for encoding
categorical_feature_names = encoder.get_feature_names_out(categorical_columns)
# Combine scaled numerical and encoded categorical features for training and tes
X_train_processed = pd.DataFrame(
    data=np.hstack((X_train_numerical_scaled, X_train_categorical_encoded)),
    columns=numerical_columns + list(categorical_feature_names)
)

X_test_processed = pd.DataFrame(
    data=np.hstack((X_test_numerical_scaled, X_test_categorical_encoded)),
    columns=numerical_columns + list(categorical_feature_names)
)
```

In [53]:
```
X_train_processed.head()
```

Out[53]:

	Capped_Flight_Distance	Scheduled_Departure_Time	Weather_Score	Log_Previous_Flig
0	-0.152042	-0.822306	0.176550	
1	0.410106	-1.554112	-0.707136	
2	1.159635	0.787667	-0.627524	
3	1.798940	0.348584	-0.670861	
4	0.553398	-0.383222	-1.095607	

5 rows × 44 columns

##Model Building: Logistic Regression

In [55]:
```
#Training the Logistic Regression Model
from sklearn.linear_model import LogisticRegression
#initialize the model
model = LogisticRegression()
```

```
#Train the model
model.fit(X_train_processed,y_train)
```

Out[55]:

```
▾ LogisticRegression
LogisticRegression()
```

In [56]:

```
#making predictions on test set
y_pred = model.predict(X_test_processed)
y_proba = model.predict_proba(X_test_processed)[:, 1]
```

##Model Evaluation

In [57]:

```
#Evaluate the model
from sklearn.metrics import accuracy_score,precision_score,recall_score, confusi
#Accuracy
accuracy = accuracy_score(y_test,y_pred)
print(f'Accuracy: {accuracy: 2f}')

#precision,Recall, F1-score
print(classification_report(y_test,y_pred))

#confusion matrix
conf_matrix = confusion_matrix(y_test,y_pred)
print('Confusion Matrix: ')
print(conf_matrix)
```

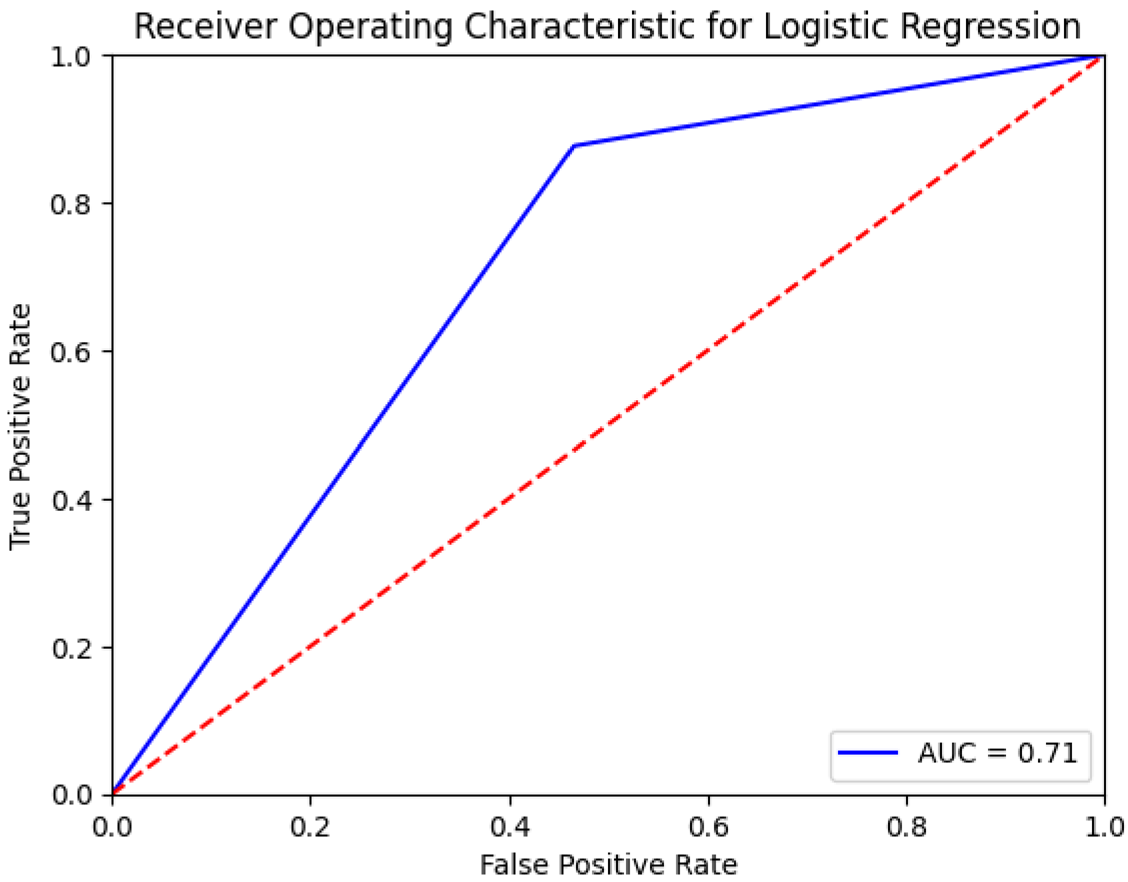
```
Accuracy:  0.770000
              precision    recall  f1-score   support

           0       0.66      0.53      0.59       187
           1       0.81      0.88      0.84       413

    accuracy                           0.77       600
   macro avg       0.73      0.71      0.72       600
weighted avg       0.76      0.77      0.76       600

Confusion Matrix:
[[100  87]
 [ 51 362]]
```

In [58]:

```
import sklearn.metrics as metrics
fpr, tpr, threshold = metrics.roc_curve(y_test, y_pred)
print(fpr)
print(tpr)
print(threshold)
roc_auc = metrics.auc(fpr, tpr)
print(roc_auc)

# method I: plt
plt.title('Receiver Operating Characteristic for Logistic Regression')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
```

```
plt.xlabel('False Positive Rate')
plt.show()
```

```
[0.         0.46524064 1.        ]
[0.         0.87651332 1.        ]
[inf  1.  0.]
0.7056363377400265
```



Our model achived an accuracy of 77%, we can try other classification methods to see how they perform

CHECKING OTHER MODELS

In [81]:
```
#import sklearn.metrics as metrics
#from sklearn.metrics import accuracy_score,precision_score,recall_score, confus
#from sklearn.model_selection import cross_val_score
#from sklearn.metrics import roc_auc_score
```

##Random Forest

In [59]:
```
from sklearn.ensemble import RandomForestClassifier
randm=RandomForestClassifier(max_depth=5)
```

In [60]:
```
# Random Forest model
rf = RandomForestClassifier(random_state=42)

# Fit the model
rf.fit(X_train_processed, y_train)

# Predict on test data
y_pred_rf = rf.predict(X_test_processed)
```

In [61]:
```
#Evaluate the model

#Accuracy
accuracy_rf = accuracy_score(y_test,y_pred_rf)
print(f'Accuracy: {accuracy_rf: 2f}')

#precision,Recall, F1-score
print(classification_report(y_test,y_pred_rf))

#confusion matrix
#conf_matrix = confusion_matrix(y_test,y_pred_rand)
#print('Confusion Matrix: ')
#print(conf_matrix)
```

```
Accuracy:  0.983333
              precision    recall  f1-score   support

           0       0.96      0.99      0.97       187
           1       1.00      0.98      0.99       413

    accuracy                           0.98       600
   macro avg       0.98      0.98      0.98       600
weighted avg       0.98      0.98      0.98       600
```

##Decision Tree

In [63]:
```
#import decision tree classifier
from sklearn.tree import DecisionTreeClassifier
```

In [64]:
```
# Decision Tree model
dt = DecisionTreeClassifier(random_state=42)

# Fit the model
dt.fit(X_train_processed, y_train)

# Predict on test data
y_pred_dt = dt.predict(X_test_processed)
```

In [87]:
```
#Evaluate the model

#Accuracy
accuracy_dt = accuracy_score(y_test,y_pred_dt)
print(f'Accuracy: {accuracy_dt: 2f}')

#precision,Recall, F1-score
print(classification_report(y_test,y_pred_dt))

#confusion matrix
conf_matrix = confusion_matrix(y_test,y_pred_dt)
print('Confusion Matrix: ')
print(conf_matrix)
```

```
Accuracy:  0.966667
              precision    recall  f1-score   support

           0       0.96      0.94      0.95       187
           1       0.97      0.98      0.98       413

    accuracy                           0.97       600
   macro avg       0.96      0.96      0.96       600
weighted avg       0.97      0.97      0.97       600

Confusion Matrix:
[[175  12]
 [  8 405]]
```

##Gradient Boosting model

In [65]:
```
#import gradient boosting classifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

# Gradient Boosting model
gb =  GradientBoostingClassifier(random_state=42)

# Fit the model
gb.fit(X_train_processed, y_train)

# Predict on test data
y_pred_gb = gb.predict(X_test_processed)
```

In [66]:
```
#Evaluate the model

#Accuracy
accuracy_gb = accuracy_score(y_test,y_pred_gb)
print(f'Accuracy: {accuracy_gb: 2f}')

#precision,Recall, F1-score
print(classification_report(y_test,y_pred_gb))

#confusion matrix
conf_matrix = confusion_matrix(y_test,y_pred_gb)
print('Confusion Matrix: ')
print(conf_matrix)
```

```
Accuracy:  0.986667
              precision    recall  f1-score   support

           0       0.96      1.00      0.98       187
           1       1.00      0.98      0.99       413

    accuracy                           0.99       600
   macro avg       0.98      0.99      0.98       600
weighted avg       0.99      0.99      0.99       600

Confusion Matrix:
[[187   0]
 [  8 405]]
```

##SVM Model

In [67]:
```
#import SVM
from sklearn.svm import SVC
# Support Vector Machine model
svm = SVC(random_state=42)

# Fit the model
svm.fit(X_train_processed, y_train)

# Predict on test data
y_pred_svm = svm.predict(X_test_processed)
```

In [68]:
```
#Evaluate the model
#Accuracy
accuracy_svm = accuracy_score(y_test,y_pred_svm)
print(f'Accuracy: {accuracy_svm: 2f}')

#precision,Recall, F1-score
print(classification_report(y_test,y_pred_svm))

#confusion matrix
conf_matrix = confusion_matrix(y_test,y_pred_svm)
print('Confusion Matrix: ')
print(conf_matrix)
```

```
Accuracy:  0.900000
              precision    recall  f1-score   support

           0       0.88      0.79      0.83       187
           1       0.91      0.95      0.93       413

    accuracy                           0.90       600
   macro avg       0.89      0.87      0.88       600
weighted avg       0.90      0.90      0.90       600

Confusion Matrix: 
[[147  40]
 [ 20 393]]
```

Model Comparison Summary

Logistic Regression:

Accuracy: 77%, Precision: 66%, Recall: 53%, F1-Score: 59%

Random Forest:

Accuracy: 98%, Precision: 96%, Recall: 99%, F1-Score: 97%

SVM (Support Vector Machine):

Accuracy: 90%, Precision: 88%, Recall: 79%, F1-Score: 83%

Decision Tree:

Accuracy: 97%, Precision: 96%, Recall: 94%, F1-Score: 95%

Gradient Boosting:

Accuracy: 99%, Precision: 96%, Recall: 100%, F1-Score: 98%,

Analysis

Accuracy: Gradient Boosting performs the best with an accuracy of 99%, closely followed by Random Forest (98%) and Decision Tree (97%). Logistic Regression shows the lowest accuracy at 77%.

Precision: Random Forest and Gradient Boosting both have high precision scores (96%). Logistic Regression has the lowest precision (66%).

Recall: Gradient Boosting achieves a perfect recall of 100%, meaning it correctly identifies all positive instances. Random Forest follows closely with a recall of 99%, while Logistic Regression has the lowest recall (53%).

F1-Score: Gradient Boosting also leads with the highest F1-score of 98%, followed by Random Forest (97%) and Decision Tree (95%). Logistic Regression has the lowest F1-score (59%).

Trade-Offs and Recommendations

Gradient Boosting: With the highest accuracy, recall, and F1-score, Gradient Boosting is the most balanced model.

Random Forest: Offers a high level of precision and recall with a good F1-score.

Decision Tree: Performs well with high accuracy, precision, and recall. However, it might be more prone to overfitting.

SVM: While it has good performance, it doesn't quite match the accuracy and recall of the other methods.

Logistic Regression: Shows the lowest performance metrics.

Conclusion Based on the metrics, Gradient Boosting seems to be the best model overall due to its high accuracy, perfect recall, and strong F1-score. Random Forest is also a strong option.

In [ ]:

```
```