In [1]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, Vot
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
from imblearn.over_sampling import SMOTE
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:

```
# Load the dataset
df = pd.read_csv('C:/Users/srirk/Downloads/archive (1)/WA_Fn-UseC_-Telco-Customer-Ch
```

In [3]:

```
# Display basic information about the dataset
print("Basic Information about the Dataset:")
print(df.info())
print("\nFirst 5 Rows of the Dataset:")
print(df.head())
```

```
Basic Information about the Dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7043 non-null   object 
 1   gender            7043 non-null   object 
 2   SeniorCitizen     7043 non-null   int64  
 3   Partner           7043 non-null   object 
 4   Dependents        7043 non-null   object 
 5   tenure            7043 non-null   int64  
 6   PhoneService      7043 non-null   object 
 7   MultipleLines     7043 non-null   object 
 8   InternetService   7043 non-null   object 
 9   OnlineSecurity    7043 non-null   object 
 10  OnlineBackup      7043 non-null   object 
 11  DeviceProtection  7043 non-null   object 
 12  TechSupport       7043 non-null   object 
 13  StreamingTV       7043 non-null   object 
 14  StreamingMovies   7043 non-null   object 
 15  Contract          7043 non-null   object 
 16  PaperlessBilling  7043 non-null   object 
 17  PaymentMethod     7043 non-null   object 
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object 
 20  Churn             7043 non-null   object 
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
None

First 5 Rows of the Dataset:
   customerID  gender  SeniorCitizen Partner Dependents  tenure PhoneService  \
0  7590-VHVEG  Female              0     Yes         No       1           No   
1  5575-GNVDE    Male              0      No         No      34          Yes   
2  3668-QPYBK    Male              0      No         No       2          Yes   
3  7795-CFOCW    Male              0      No         No      45           No   
4  9237-HQITU  Female              0      No         No       2          Yes   
```

```
      MultipleLines InternetService OnlineSecurity  ... DeviceProtection  \
0  No phone service             DSL             No  ...               No
1                No             DSL            Yes  ...              Yes
2                No             DSL            Yes  ...               No
3  No phone service             DSL            Yes  ...              Yes
4                No     Fiber optic             No  ...               No

  TechSupport StreamingTV StreamingMovies        Contract PaperlessBilling  \
0          No          No              No  Month-to-month              Yes
1          No          No              No        One year               No
2          No          No              No  Month-to-month              Yes
3         Yes          No              No        One year               No
4          No          No              No  Month-to-month              Yes

              PaymentMethod MonthlyCharges  TotalCharges Churn
0          Electronic check          29.85         29.85    No
1              Mailed check          56.95        1889.5    No
2              Mailed check          53.85        108.15   Yes
3  Bank transfer (automatic)          42.30       1840.75    No
4          Electronic check          70.70        151.65   Yes

[5 rows x 21 columns]
```

In [4]:

```python
# Handling missing values
df.replace(' ', np.nan, inplace=True)
print("\nMissing Values before Imputation:")
print(df.isnull().sum())
imputer = SimpleImputer(strategy='median')
df['TotalCharges'] = imputer.fit_transform(df[['TotalCharges']])

print("\nMissing Values after Imputation:")
print(df.isnull().sum())
```

```
Missing Values before Imputation:
customerID           0
gender               0
SeniorCitizen        0
Partner              0
Dependents           0
tenure               0
PhoneService         0
MultipleLines        0
InternetService      0
OnlineSecurity       0
OnlineBackup         0
DeviceProtection     0
TechSupport          0
StreamingTV          0
StreamingMovies      0
Contract             0
PaperlessBilling     0
PaymentMethod        0
MonthlyCharges       0
TotalCharges        11
Churn                0
dtype: int64

Missing Values after Imputation:
customerID          0
gender              0
SeniorCitizen       0
Partner             0
```

```
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges        0
Churn               0
dtype: int64
```

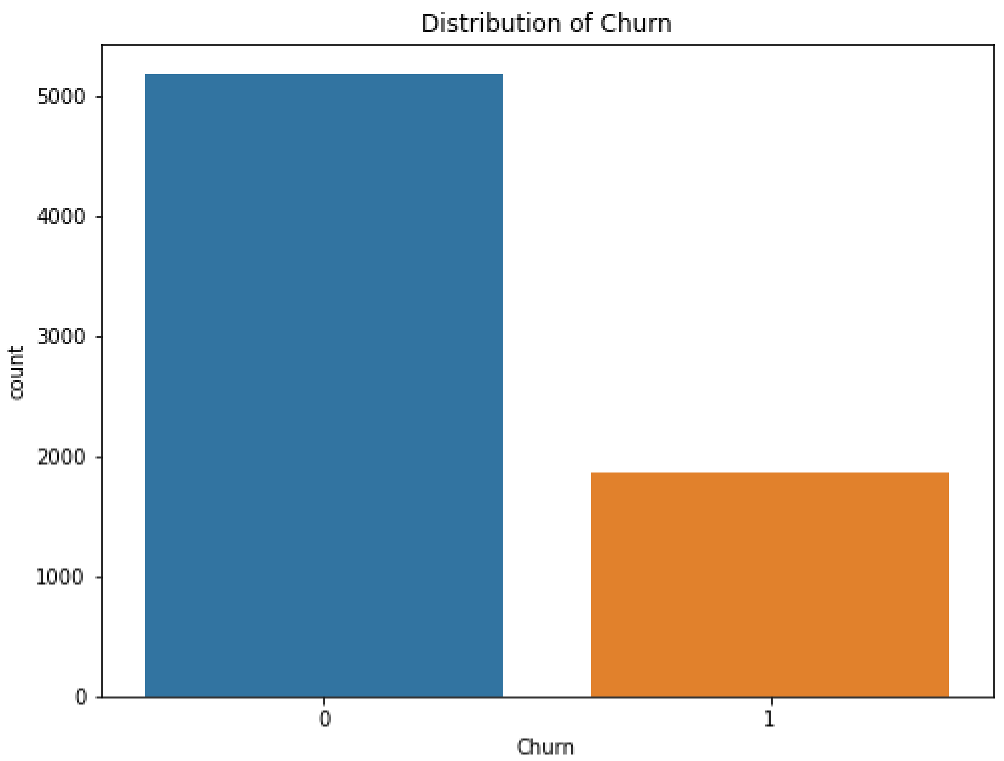
In [5]:

```
# Encode categorical variables
categorical_features = df.select_dtypes(include=['object']).columns
for col in categorical_features:
    if col != 'customerID':
        df[col] = LabelEncoder().fit_transform(df[col])

df.drop('customerID', axis=1, inplace=True)
print("\nDataset after Encoding Categorical Variables and Dropping customerID Column
print(df.head())
```

```
Dataset after Encoding Categorical Variables and Dropping customerID Column:
   gender  SeniorCitizen  Partner  Dependents  tenure  PhoneService  \
0       0              0        1           0       1             0   
1       1              0        0           0      34             1   
2       1              0        0           0       2             1   
3       1              0        0           0      45             0   
4       0              0        0           0       2             1   

   MultipleLines  InternetService  OnlineSecurity  OnlineBackup  \
0              1                0               0             2   
1              0                0               2             0   
2              0                0               2             2   
3              1                0               2             0   
4              0                1               0             0   

   DeviceProtection  TechSupport  StreamingTV  StreamingMovies  Contract  \
0                 0            0            0                0         0   
1                 2            0            0                0         1   
2                 0            0            0                0         0   
3                 2            2            0                0         1   
4                 0            0            0                0         0   

   PaperlessBilling  PaymentMethod  MonthlyCharges  TotalCharges  Churn  
0                 1              2           29.85         29.85      0  
1                 0              3           56.95       1889.50      0  
2                 1              3           53.85        108.15      1  
3                 0              0           42.30       1840.75      0  
4                 1              2           70.70        151.65      1  
```

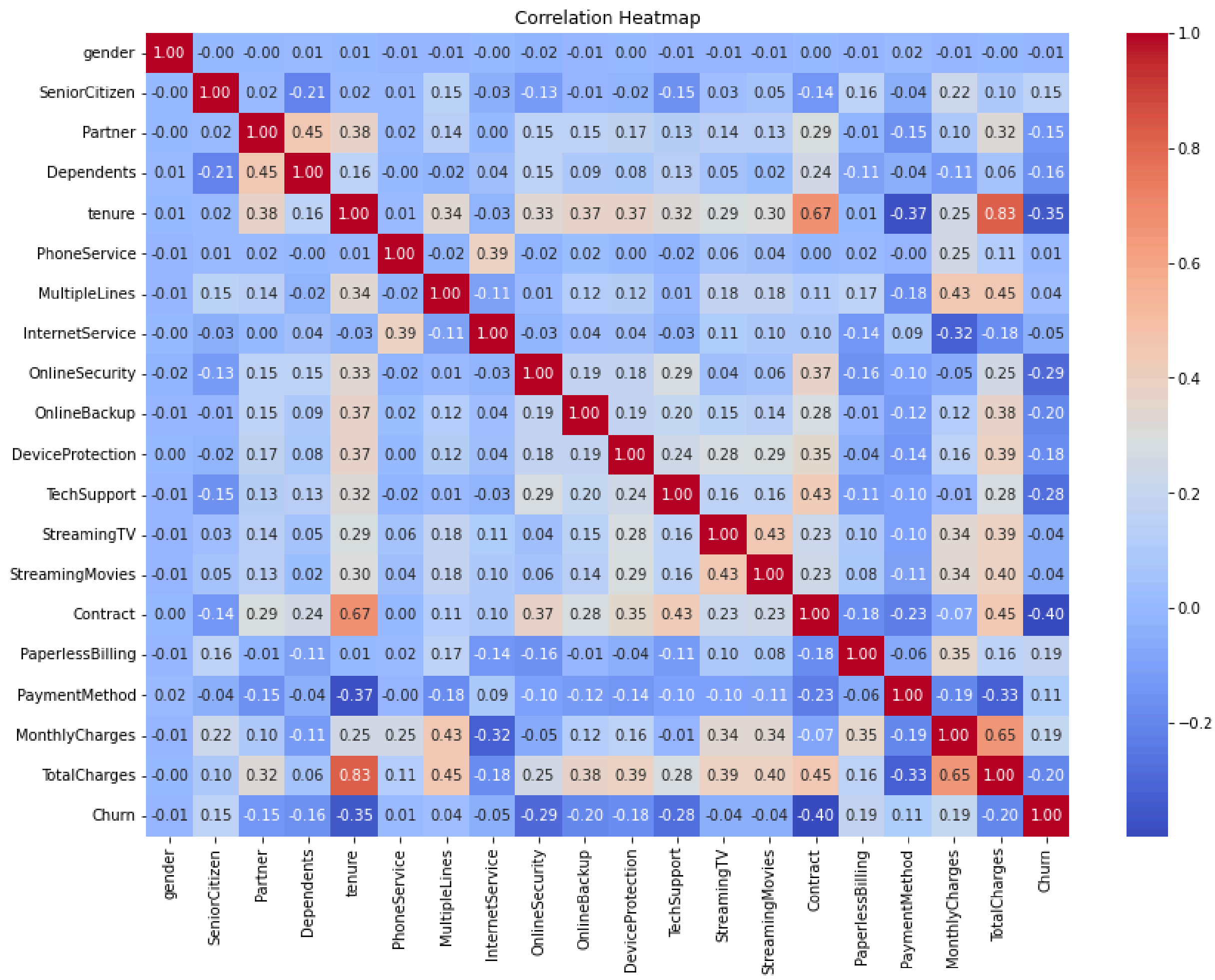
In [6]:

```
# Visualize the distribution of churn
plt.figure(figsize=(8, 6))
sns.countplot(data=df, x='Churn')
plt.title('Distribution of Churn')
plt.show()
```



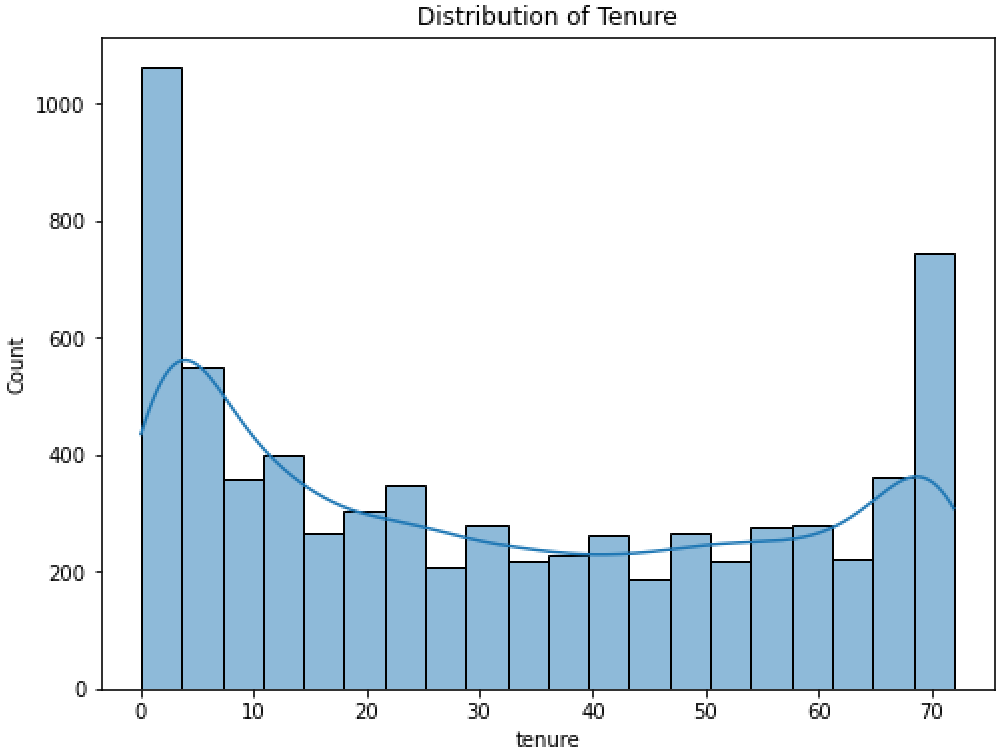

In [7]:

```
# Correlation heatmap
plt.figure(figsize=(14, 10))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap')
plt.show()
```



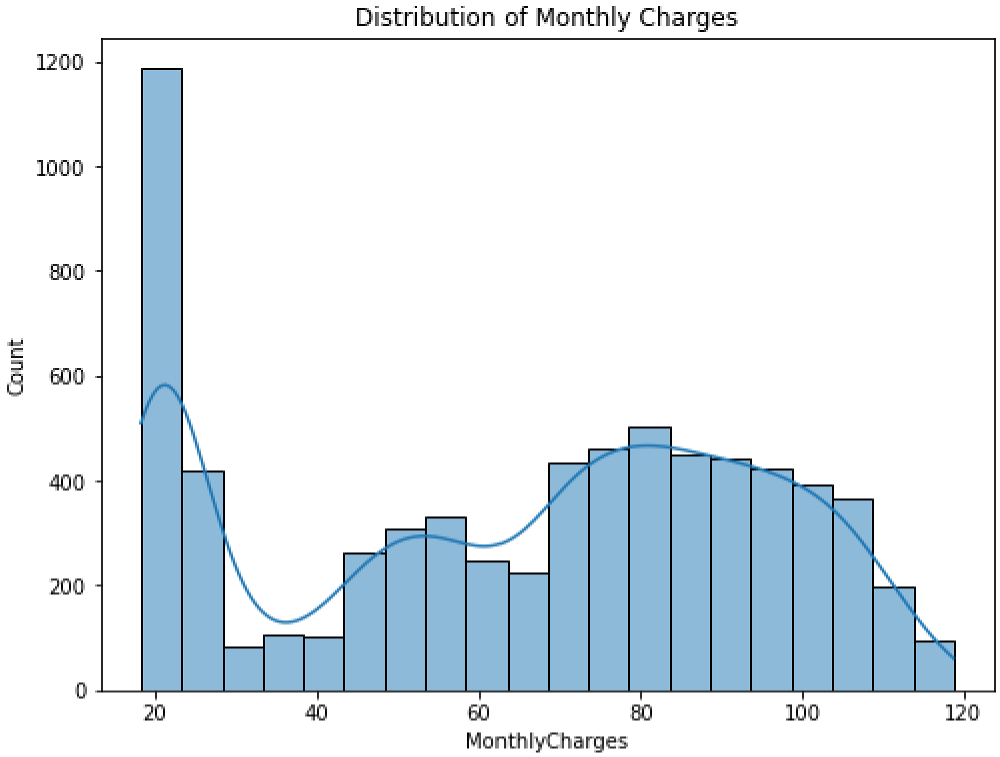
In [8]:

```
# Visualize the distribution of features
plt.figure(figsize=(8, 6))
sns.histplot(df['tenure'], bins=20, kde=True)
plt.title('Distribution of Tenure')
plt.show()
```




In [9]:

```
plt.figure(figsize=(8, 6))
sns.histplot(df['MonthlyCharges'], bins=20, kde=True)
plt.title('Distribution of Monthly Charges')
plt.show()
```



In [10]:
```
# Feature scaling
scaler = StandardScaler()
numerical_features = df.select_dtypes(include=['int64', 'float64']).columns
df[numerical_features] = scaler.fit_transform(df[numerical_features])
```

In [11]:
```
# Splitting data into features and target
X = df.drop('Churn', axis=1)
y = df['Churn']
```

In [12]:
```
# Handle class imbalance with SMOTE
smote = SMOTE(random_state=42)
X_res, y_res = smote.fit_resample(X, y)
```

In [13]:
```
# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size=0.2, ran
```

In [14]:
```
# Initialize models
log_reg = LogisticRegression(random_state=42)
rf_clf = RandomForestClassifier(random_state=42)
gb_clf = GradientBoostingClassifier(random_state=42)
```

In [15]:
```
# Hyperparameter tuning using Grid Search
param_grid_log_reg = {
    'C': [0.01, 0.1, 1, 10, 100],
    'solver': ['lbfgs', 'liblinear']
}

param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10]
}

param_grid_gb = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 4, 5]
}

grid_search_log_reg = GridSearchCV(LogisticRegression(random_state=42), param_grid_l
grid_search_rf = GridSearchCV(RandomForestClassifier(random_state=42), param_grid_rf
grid_search_gb = GridSearchCV(GradientBoostingClassifier(random_state=42), param_gri
```

In [16]:
```
# Train models with best parameters
grid_search_log_reg.fit(X_train, y_train)
grid_search_rf.fit(X_train, y_train)
grid_search_gb.fit(X_train, y_train)

best_log_reg = grid_search_log_reg.best_estimator_
best_rf_clf = grid_search_rf.best_estimator_
best_gb_clf = grid_search_gb.best_estimator_

# Predict on the test set
y_pred_log_reg = best_log_reg.predict(X_test)
y_pred_rf_clf = best_rf_clf.predict(X_test)
```

```
y_pred_gb_clf = best_gb_clf.predict(X_test)

# Function to print evaluation metrics
def evaluate_model(y_true, y_pred, model_name):
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)
    conf_matrix = confusion_matrix(y_true, y_pred)

    print(f"Evaluation Metrics for {model_name}:")
    print(f"Accuracy: {accuracy:.2f}")
    print(f"Precision: {precision:.2f}")
    print(f"Recall: {recall:.2f}")
    print(f"F1 Score: {f1:.2f}")
    print(f"Confusion Matrix:\n{conf_matrix}\n")

    return accuracy
```

In [17]:

```
# Evaluate Logistic Regression
acc_log_reg = evaluate_model(y_test, y_pred_log_reg, "Logistic Regression")
```

```
Evaluation Metrics for Logistic Regression:
Accuracy: 0.80
Precision: 0.77
Recall: 0.86
F1 Score: 0.81
Confusion Matrix:
[[749 272]
 [150 899]]
```

In [18]:

```
# Evaluate Random Forest Classifier
acc_rf_clf = evaluate_model(y_test, y_pred_rf_clf, "Random Forest Classifier")
```

```
Evaluation Metrics for Random Forest Classifier:
Accuracy: 0.85
Precision: 0.83
Recall: 0.88
F1 Score: 0.86
Confusion Matrix:
[[838 183]
 [124 925]]
```

In [19]:

```
# Evaluate Gradient Boosting Classifier
acc_gb_clf = evaluate_model(y_test, y_pred_gb_clf, "Gradient Boosting Classifier")
```

```
Evaluation Metrics for Gradient Boosting Classifier:
Accuracy: 0.84
Precision: 0.84
Recall: 0.86
F1 Score: 0.85
Confusion Matrix:
[[847 174]
 [152 897]]
```

In [20]:

```
# Ensemble method using Voting Classifier
voting_clf = VotingClassifier(estimators=[
    ('log_reg', best_log_reg),
```

```
    ('rf_clf', best_rf_clf),
    ('gb_clf', best_gb_clf)
], voting='soft')

voting_clf.fit(X_train, y_train)
y_pred_voting = voting_clf.predict(X_test)
```

In [21]:

```
# Evaluate Voting Classifier
acc_voting = evaluate_model(y_test, y_pred_voting, "Voting Classifier")
```

```
Evaluation Metrics for Voting Classifier:
Accuracy: 0.85
Precision: 0.82
Recall: 0.90
F1 Score: 0.86
Confusion Matrix:
[[818 203]
 [109 940]]
```

In [22]:

```
# Print accuracies for each model
print("Model Accuracies:")
print(f"Logistic Regression Accuracy: {acc_log_reg:.2f}")
print(f"Random Forest Classifier Accuracy: {acc_rf_clf:.2f}")
print(f"Gradient Boosting Classifier Accuracy: {acc_gb_clf:.2f}")
print(f"Voting Classifier Accuracy: {acc_voting:.2f}")
```

```
Model Accuracies:
Logistic Regression Accuracy: 0.80
Random Forest Classifier Accuracy: 0.85
Gradient Boosting Classifier Accuracy: 0.84
Voting Classifier Accuracy: 0.85
```