In [1]:
```
#Final Project
#Predicting Customer Churn using Machine Learning
#Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
```

In [7]:
```
#Load Dataset of telecommunication
data=pd.read_csv("telecom_churn.csv")
```

In [8]:
```
data.head()
```

Out[8]:

	Churn	AccountWeeks	ContractRenewal	DataPlan	DataUsage	CustServCalls	DayMins	DayCalls	MonthlyCharge	OverageFee	RoamMins
0	0	128	1	1	2.7	1	265.1	110	89.0	9.87	10.0
1	0	107	1	1	3.7	1	161.6	123	82.0	9.78	13.7
2	0	137	1	0	0.0	0	243.4	114	52.0	6.06	12.2
3	0	84	0	0	0.0	2	299.4	71	57.0	3.10	6.6
4	0	75	0	0	0.0	3	166.7	113	41.0	7.42	10.1

In [9]:
```
data.shape
```

Out[9]:
```
(3333, 11)
```

In [10]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Churn           3333 non-null   int64  
 1   AccountWeeks    3333 non-null   int64  
 2   ContractRenewal 3333 non-null   int64  
 3   DataPlan        3333 non-null   int64  
 4   DataUsage       3333 non-null   float64
 5   CustServCalls   3333 non-null   int64  
 6   DayMins         3333 non-null   float64
 7   DayCalls        3333 non-null   int64  
 8   MonthlyCharge   3333 non-null   float64
 9   OverageFee      3333 non-null   float64
 10  RoamMins        3333 non-null   float64
dtypes: float64(5), int64(6)
memory usage: 286.6 KB
```

In [11]:
```
data.describe()
```

Out[11]:

	Churn	AccountWeeks	ContractRenewal	DataPlan	DataUsage	CustServCalls	DayMins	DayCalls	MonthlyCharge	OverageFee	RoamMins
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000
mean	0.144914	101.064806	0.903090	0.276628	0.816475	1.562856	179.775098	100.435644	56.305161	10.051488	10.237294
std	0.352067	39.822106	0.295879	0.447398	1.272668	1.315491	54.467389	20.069084	16.426032	2.535712	2.791840
min	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	14.000000	0.000000	0.000000
25%	0.000000	74.000000	1.000000	0.000000	0.000000	1.000000	143.700000	87.000000	45.000000	8.330000	8.500000
50%	0.000000	101.000000	1.000000	0.000000	0.000000	1.000000	179.400000	101.000000	53.500000	10.070000	10.300000
75%	0.000000	127.000000	1.000000	1.000000	1.780000	2.000000	216.400000	114.000000	66.200000	11.770000	12.100000
max	1.000000	243.000000	1.000000	1.000000	5.400000	9.000000	350.800000	165.000000	111.300000	18.190000	20.000000

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

Out[12]:
```
Churn             0
AccountWeeks      0
ContractRenewal   0
DataPlan          0
DataUsage         0
CustServCalls     0
DayMins           0
DayCalls          0
MonthlyCharge     0
OverageFee        0
RoamMins          0
dtype: int64
```

In [36]:
```
#Removing irrelevant columns
data.columns = data.iloc[0]
data=data.drop(data.index[0])
```

In [38]:
```
col_names=data.columns.tolist()
col_names
```

Out[38]:
```
[0.0, 128.0, 1.0, 1.0, 2.7, 1.0, 265.1, 110.0, 89.0, 9.87, 10.0]
```

In [31]:
```
#Handle missing values
data.fillna(data.mean())
```

Out[31]:

	0.00	128.00	1.00	1.00	1.00	265.10	110.00	89.00	9.87
1	0	107	1	1	1	161.6	123	82.0	9.78
2	0	137	1	0	0	243.4	114	52.0	6.06
3	0	84	0	0	2	299.4	71	57.0	3.10
4	0	75	0	0	3	166.7	113	41.0	7.42
5	0	118	0	0	0	223.4	98	57.0	11.03
...	...	...	...	...	...	...	...	...	...
3328	0	192	1	1	2	156.2	77	71.7	10.78
3329	0	68	1	0	3	231.1	57	56.4	7.67
3330	0	28	1	0	2	180.8	109	56.0	14.44
3331	0	184	0	0	2	213.8	105	50.0	7.98
3332	0	74	1	1	0	234.4	113	100.0	13.30

3332 rows × 9 columns

In [16]:
```
data= data.dropna()
```

In [4]:
```
# converting categorical variables to numerical values
```

In [3]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

df = pd.read_csv('telecom_churn.csv')

X = df.drop('Churn', axis=1)
y = df['Churn']

# Splitting the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# categorical columns
X_train_encoded = pd.get_dummies(X_train, drop_first=True)
X_test_encoded = pd.get_dummies(X_test, drop_first=True)

# Create and train the model (Random Forest classifier)
model = RandomForestClassifier(random_state=42)
model.fit(X_train_encoded, y_train)

# predictions
y_pred = model.predict(X_test_encoded)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
```

```
Accuracy: 0.9265367316341829
```

In [20]:
```
data = pd.get_dummies(data, drop_first=True)
```

In [23]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC

# Initialize models
models = {
    'Logistic Regression': LogisticRegression(),
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'Support Vector Machine': SVC()
}

# Train models
for name, model in models.items():
    model.fit(X_train, y_train)
    print(f"{name} trained.")
```

```
C:\Users\D E L L\anaconda4\lib\site-packages\sklearn\linear_model\_logistic.py:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Logistic Regression trained.
Decision Tree trained.
Random Forest trained.
Support Vector Machine trained.
```

In [33]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Evaluating models
for name, model in models.items():
    y_pred = model.predict(X_test)

    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)

    print(f"Metrics for {name}:")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1 Score: {f1:.4f}")
    print("\n")
```

```
Metrics for Logistic Regression:
Accuracy: 0.8470764617691154
Precision: 0.4838709677419355
Recall: 0.14814814814814815
F1 Score: 0.22752752752752754


Metrics for Decision Tree:
Accuracy: 0.8725637181409296
Precision: 0.5796331273549137
Recall: 0.5841584158415841
F1 Score: 0.5819078811738


Metrics for Random Forest:
Accuracy: 0.9265367316341751
Precision: 0.8450704225352113
Recall: 0.5940594059405941
F1 Score: 0.6976744186046511


Metrics for Support Vector Machine:
Accuracy: 0.8500749625187406
Precision: 1.0
Recall: 0.009900990099009901
F1 Score: 0.019607843137254949
```

In [32]:
```
new_data = pd.read_csv('telecom_churn.csv')
new_data_encoded = pd.get_dummies(new_data, drop_first=True)

training_columns = X_train.columns

new_data_encoded = new_data_encoded.reindex(columns=training_columns, fill_value=0)

# The best-performing model for predictions
predictions = best_model.predict(new_data_encoded)
print("Predictions using the best-performing model:")
print(predictions)
```

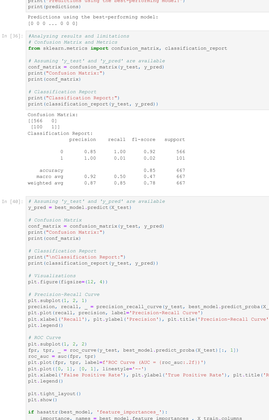
```
Predictions using the best-performing model:
[0 0 0 ... 0 0 0]
```

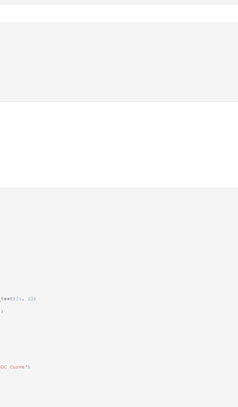
In [34]:
```
#Analyzing results and limitations
# Confusion Matrix and Metrics
from sklearn.metrics import confusion_matrix, classification_report

# Assuming 'y_test' and 'y_pred' are available
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)

# Classification Report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

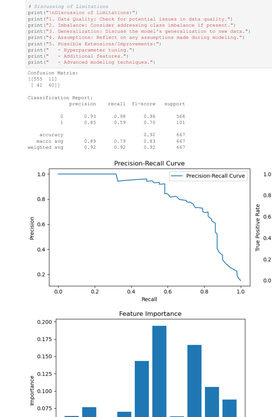
```
Confusion Matrix:
[[566   0]
 [100  12]]
Classification Report:
              precision    recall  f1-score   support

           0       0.85      1.00      0.92       566
           1       1.00      0.01      0.02       101

    accuracy                           0.85       667
   macro avg       0.92      0.50      0.47       667
weighted avg       0.87      0.85      0.78       667
```

In [40]:
```
# Assuming 'y_test' and 'y_pred' are available
y_pred = best_model.predict(X_test)

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)

# Classification Report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Visualizations
plt.figure(figsize=(12, 4))

# Precision-Recall Curve
plt.subplot(1, 2, 1)
precision, recall, _ = precision_recall_curve(y_test, best_model.predict_proba(X_test)[:, 1])
plt.plot(recall, precision, label='Precision-Recall Curve')
plt.xlabel('Recall'), plt.ylabel('Precision'), plt.title('Precision-Recall Curve')
plt.legend()

# ROC Curve
plt.subplot(1, 2, 2)
fpr, tpr, _ = roc_curve(y_test, best_model.predict_proba(X_test)[:, 1])
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate'), plt.ylabel('True Positive Rate'), plt.title('ROC Curve')
plt.legend()

plt.tight_layout()
plt.show()

if hasattr(best_model, 'feature_importances_'):
    feature_importances = best_model.feature_importances_ X_train.columns
    plt.barh(range(len(feature_importances)), feature_importances)
    plt.xlabel('Feature'), plt.ylabel('Importance'), plt.title('Feature Importance')
    plt.xticks(rotation=45, ha='right')
    plt.show()

# Discussing of Limitations
print("\nDiscussion of Limitations:")
print("1. Data Quality: Check for potential issues in data quality.")
print("2. Imbalance: Consider addressing class imbalance if present.")
print("3. Generalization: Discuss the model's generalization to new data.")
print("4. Assumptions: Reflect on any assumptions made during modeling.")
print("5. Possible Extensions/Improvements:")
print("   - Hyperparameter tuning.")
print("   - Additional features.")
print("   - Advanced modeling techniques.")
```

```
Confusion Matrix:
[[555  11]
 [ 41  60]]

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.98      0.96       566
           1       0.85      0.59      0.70       101

    accuracy                           0.92       667
   macro avg       0.89      0.79      0.83       667
weighted avg       0.92      0.92      0.92       667
```








```
Discussion of Limitations:
1. Data Quality: Check for potential issues in data quality.
2. Imbalance: Consider addressing class imbalance if present.
3. Generalization: Discuss the model's generalization to new data.
4. Assumptions: Reflect on any assumptions made during modeling.
5. Possible Extensions/Improvements:
   - Hyperparameter tuning.
   - Additional features.
   - Advanced modeling techniques.
```

In [ ]:
```
```