SyriaTel-Customer-Churn-phase-3project- / phase-3-project.ipynb

dennismwau-1 Add files via upload 7e73af8 · 1 minute ago

2360 lines (2360 loc) · 1.48 MB

SyriaTel-Customer-Churn-phase-3project- / phase-3-project.ipynb Top

Preview Code Blame Raw

SyriaTel Customer Churn Project

Please fill out:

- Student name: Dennis mwania
- Student pace: Remote
- Scheduled project review :Feb 17 at 12am - Feb 23 at 11:59pm
- Instructor name: faith rotich
- Blog post URL:

INTRODUCTION

In this project, we focus on the analysis of customer churn data from SyriaTel, a leading telecommunications provider. Customer churn refers to the phenomenon of customers discontinuing their services, and understanding the factors behind churn is critical for telecom companies like SyriaTel. By predicting churn, companies can take proactive measures to retain customers, improve customer satisfaction, and optimize resources.

The SyriaTel Customer Churn dataset contains information about the usage, demographic, and service-related details of customers. The main objective of this project is to identify the key factors influencing customer churn and to build a predictive model to forecast the likelihood of customer attrition.

Problem Statement:

SyriaTel, a prominent telecommunications company, seeks to minimize customer churn—the rate at which customers stop doing business with them. High churn rates can be costly, as acquiring new customers is more expensive than retaining existing ones. This project aims to develop a predictive model to identify customers who are at high risk of discontinuing their services (churning) in the near future. By predicting churn, SyriaTel can implement proactive strategies to retain these customers and reduce revenue loss.

Business Understanding

Main Objective:

The primary goal of this project is to predict customer churn for SyriaTel, a telecommunications company. Customer churn refers to the phenomenon where customers stop using a company's services. Predicting churn is crucial for SyriaTel to implement retention strategies, improve customer satisfaction, and ultimately increase revenue.

Specific objectives:

1. To do exploratory data analysis on the data
2. To Identify the key factors or variables that contribute most to customer churn.
3. To fit different classification algorithm models
4. To determine which one works best for churn prediction
5. To Evaluate the effectiveness of churn prediction models using metrics like accuracy, precision, recall, and F1 score.

Data Source:

The dataset used for this analysis is the SyriaTel Customer Churn dataset, which contains information about customers and whether they have churned. https://www.kaggle.com/becksddf/churn-in-telecoms-dataset

Importing Libraries:

Before loading the dataset, it's essential to import the required libraries. These libraries provide functionality for handling data, performing analysis, and visualizing the results.

In [2]:

```
#importing 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.metrics import classification_report, confusion_matrix, accura
```

Data import and Overview

The "Data Import and Overview" involves loading the dataset into your working environment and performing basic exploratory tasks to understand the dataset's structure, the types of variables present.

In [3]:

```
#importing churn_in_telecomms dataset
df = pd.read_csv('bigml_59c28831336c6604c800002a.csv')
```

In [4]:

```
#reviewing the first ten columns
df.head(10)
```

Out[4]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls
0	KS	128	415	382-	no	yes	25	265.1	110

				4657					
1	OH	107	415	371-7191	no	yes	26	161.6	123
2	NJ	137	415	358-1921	no	no	0	243.4	114
3	OH	84	408	375-9999	yes	no	0	299.4	71
4	OK	75	415	330-6626	yes	no	0	166.7	113
5	AL	118	510	391-8027	yes	no	0	223.4	98
6	MA	121	510	355-9993	no	yes	24	218.2	88
7	MO	147	415	329-9001	yes	no	0	157.0	79
8	LA	117	408	335-4719	no	no	0	184.5	97
9	WV	141	415	330-8173	yes	yes	37	258.6	84

10 rows × 21 columns

In [5]:

```
#checking the data type
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   state                   3333 non-null   object 
 1   account length          3333 non-null   int64  
 2   area code               3333 non-null   int64  
 3   phone number            3333 non-null   object 
 4   international plan      3333 non-null   object 
 5   voice mail plan         3333 non-null   object 
 6   number vmail messages   3333 non-null   int64  
 7   total day minutes       3333 non-null   float64
 8   total day calls         3333 non-null   int64  
 9   total day charge        3333 non-null   float64
 10  total eve minutes       3333 non-null   float64
 11  total eve calls         3333 non-null   int64  
 12  total eve charge        3333 non-null   float64
 13  total night minutes     3333 non-null   float64
 14  total night calls       3333 non-null   int64  
 15  total night charge      3333 non-null   float64
 16  total intl minutes      3333 non-null   float64
 17  total intl calls        3333 non-null   int64  
 18  total intl charge       3333 non-null   float64
 19  customer service calls  3333 non-null   int64  
 20  churn                   3333 non-null   bool   
dtypes: bool(1), float64(8), int64(8), object(4)
memory usage: 524.2+ KB
```

In [6]:

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

Out[6]:

	account length	area code	number vmail messages	total day minutes	total day calls	total da charg
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.0000(
mean	101.064806	437.182418	8.099010	179.775098	100.435644	30.5623(
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.2594:
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.0000(
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.4300(
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.5000(
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.7900(
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.6400(

Data Cleaning

In this section, we will be looking at the missing values in the dataset as well as the duplicate records.

In [7]:

```
# Checking for missing values
df.isnull().sum()
```

Out[7]:

```
state                     0
account length            0
area code                 0
phone number              0
international plan        0
voice mail plan           0
number vmail messages     0
total day minutes         0
total day calls           0
total day charge          0
total eve minutes         0
total eve calls           0
total eve charge          0
total night minutes       0
total night calls         0
total night charge        0
total intl minutes        0
total intl calls          0
total intl charge         0
customer service calls    0
churn                     0
dtype: int64
```

In [8]:

```
#checking for duplicates
df.duplicated().sum()
```

Out[8]: 0

In [9]:

```
print('This data has no duplicates and missing values')
```

```
This data has no duplicates and missing values
```

Inspecting the Target Variable:

In a binary classification problem like customer churn prediction, we will look at the distribution of the target variable (e.g., Churn column) to understand how balanced the classes are. This can help you assess whether any class balancing techniques might be necessary.

In [10]:

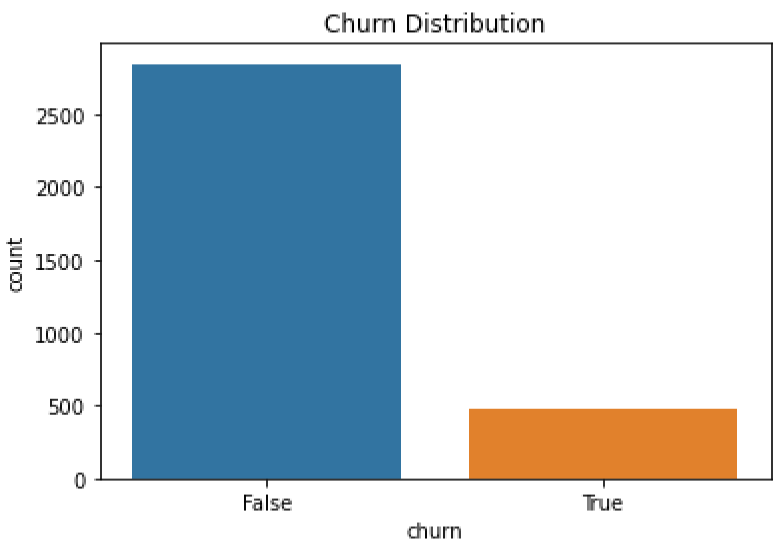
```
# Check the distribution of the target variable (Churn)
# Check the distribution of the target variable
# Replace 'Churn' with the actual column name in your dataset
df['churn'].value_counts()
```

Out[10]:

```
False    2850
True      483
Name: churn, dtype: int64
```

In [11]:

```
# Visualize the distribution of churn (target variable)
sns.countplot(x='churn', data=df) # Changed 'Churn' to 'churn'
plt.title('Churn Distribution')
plt.show()
```



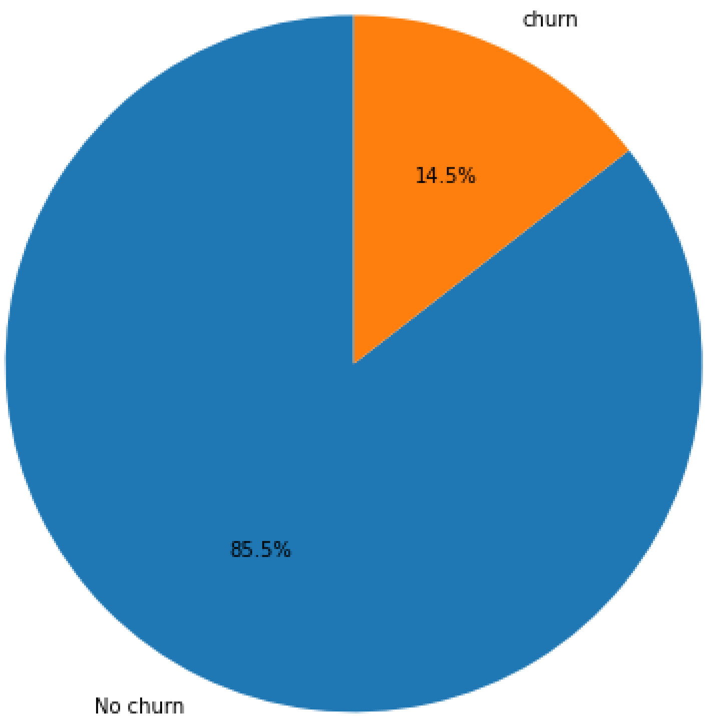
The dataset is imbalanced, with more customers not churning than churning.

In [12]:

```
#Create a pie chart using pyplot to visulaize churn rate
plt.figure(figsize=(8, 8))
plt.pie(df['churn'].value_counts(), labels=['No churn', 'churn'], autopct='
plt.title('churn Rate')
```

Out[12]: Text(0.5, 1.0, 'churn Rate')

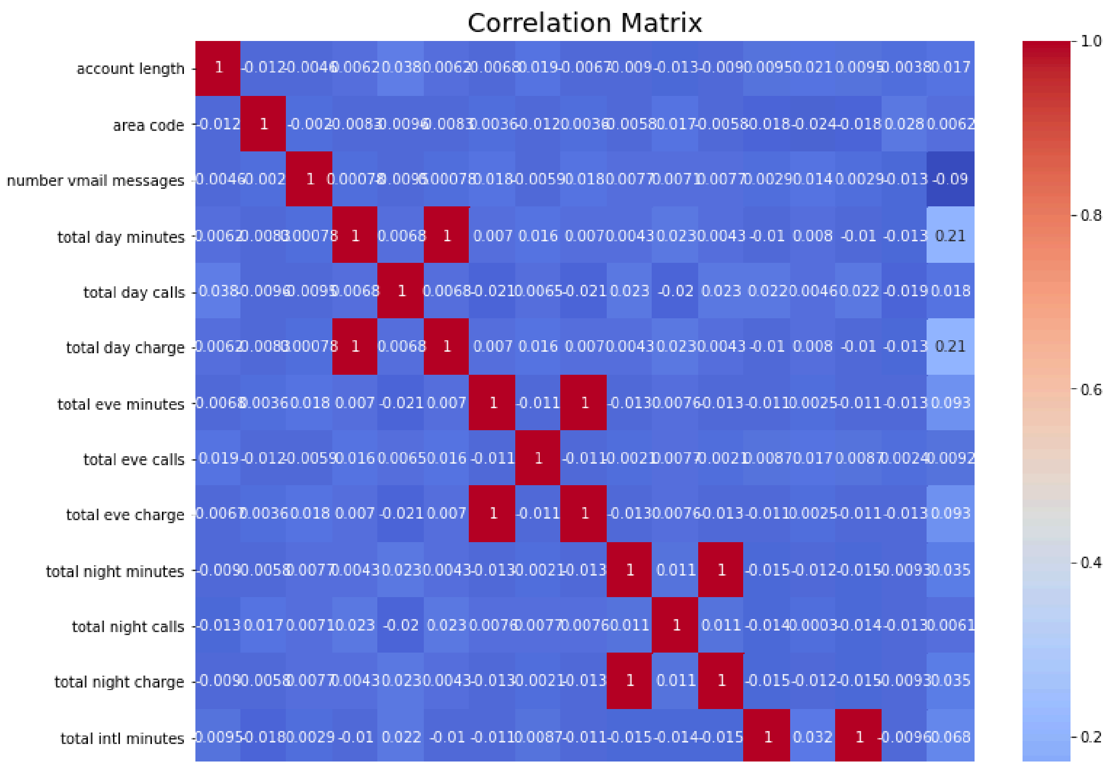
churn Rate

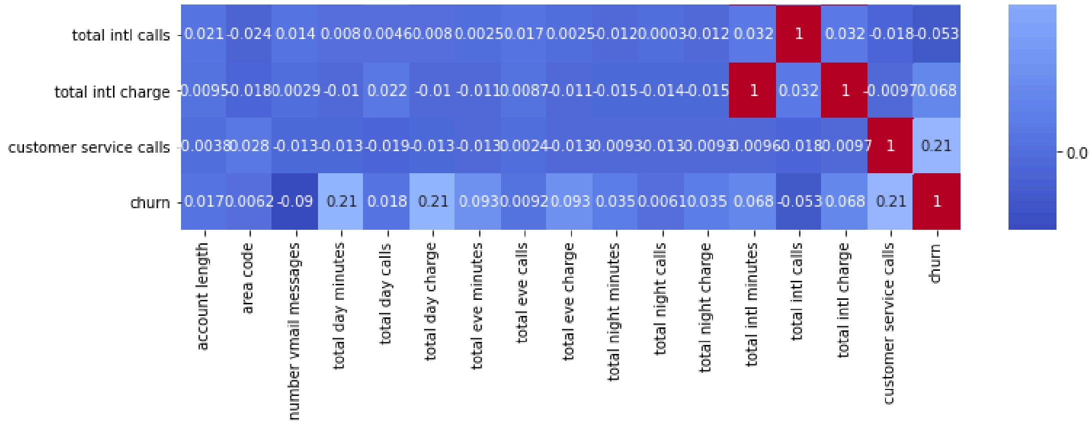


The chart clearly shows that the majority of customers (85.5%) have not churned, while a smaller portion (14.5%) has churned. This suggests a relatively low churn rate, which is generally positive for a business.

In [13]:

```
#correllation matrix
corr_matrix = df.corr()
fig, ax = plt.subplots(figsize=(12,12))   # Set the figure size to 12 inche
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', ax=ax)
plt.title('Correlation Matrix', fontsize=18)
plt.show();
```





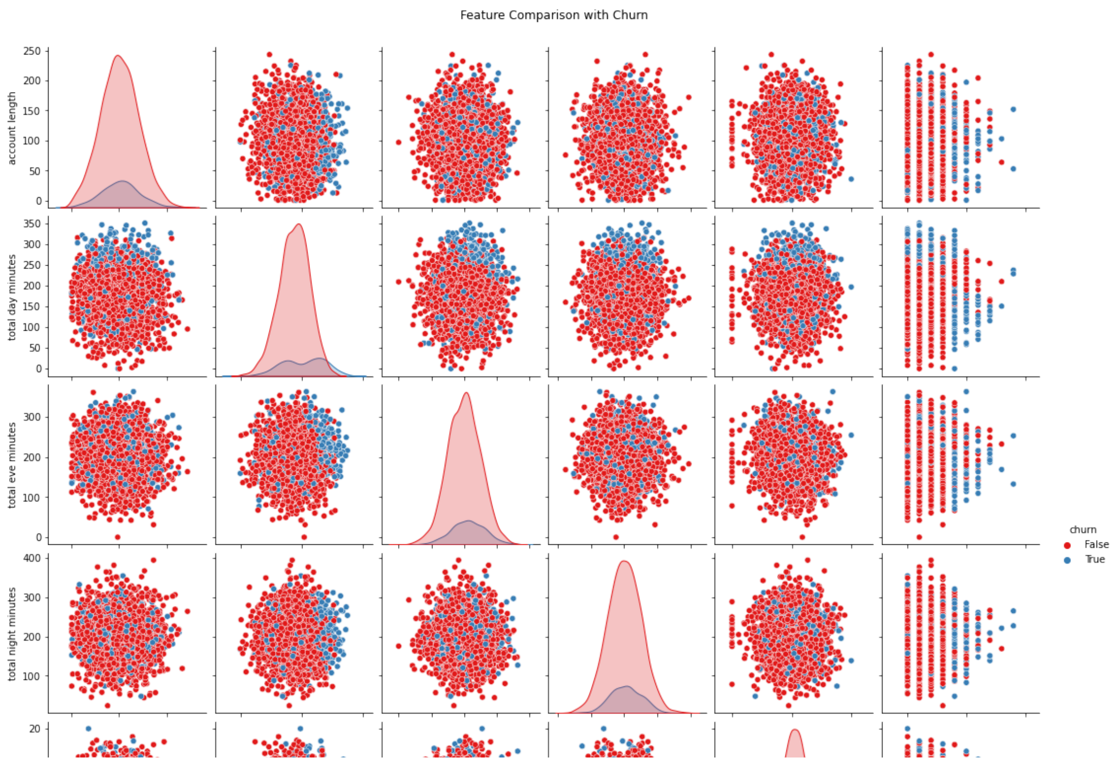
Observation:

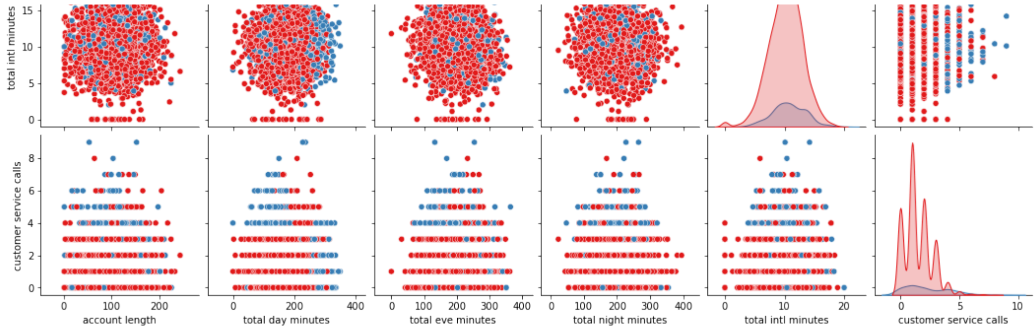
Data: The matrix examines correlations between variables like account length, call usage (day, evening, night, international), customer service calls, and churn (whether a customer left).

- Strong Correlations: Expectedly, there are strong positive correlations between call minutes and corresponding charges (day minutes vs. day charge, etc.).
- Churn Indicators: The matrix suggests a moderate positive correlation between customer service calls and churn, indicating that customers who call customer service more often are more likely to churn. There's also a weak negative correlation between account length and churn, meaning longer-term customers are slightly less likely to leave

In [14]:

```
# Select relevant features for pairplot
features = ['account length', 'total day minutes', 'total eve minutes', 'tc
sns.pairplot(df[features], hue='churn', palette='Set1')
plt.suptitle('Feature Comparison with Churn', y=1.02)
plt.show()
```



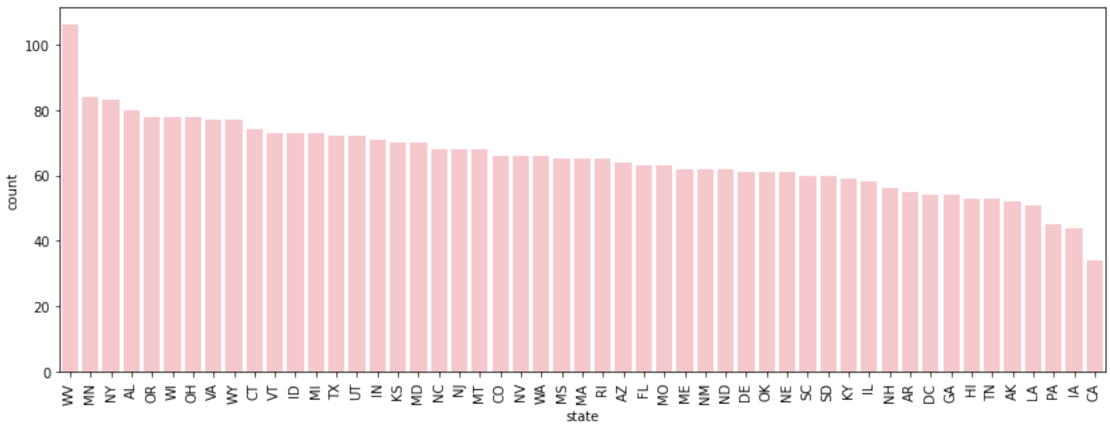


In [15]:

```
#Checking the distribution of categorical features
def plot_categorical_distribution(df, feature):
    """
    Plots the distribution of a categorical feature in the given data.
    """
    plt.figure(figsize=(14, 5))
    sns.countplot(x=feature, data=df,color='pink', order=df[feature].value_
    plt.xticks(rotation=90)
    plt.show()
```

In [16]:

```
#state
plot_categorical_distribution(df, 'state')
```

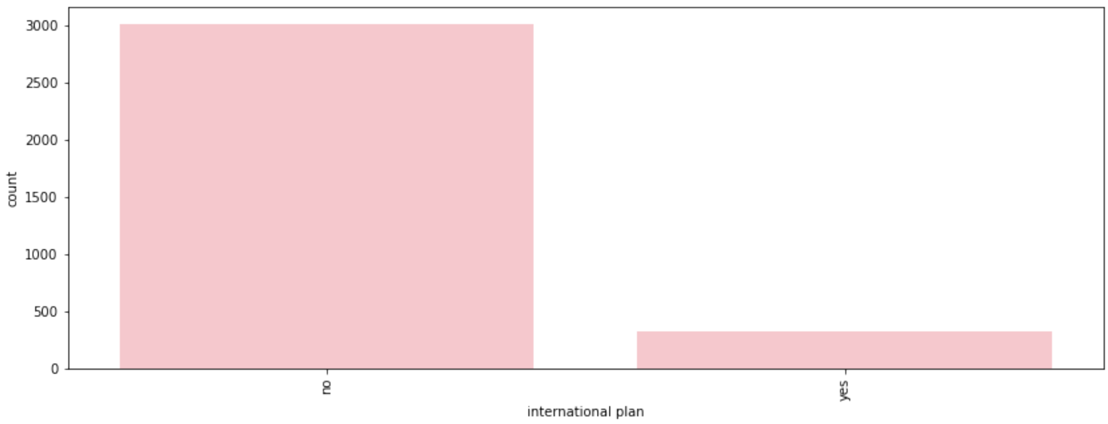


Most customers are from West Virginia,Minnesota,NewYork,Alabama and alabama

international plan

In [17]:

```
plot_categorical_distribution(df,'international plan')
```

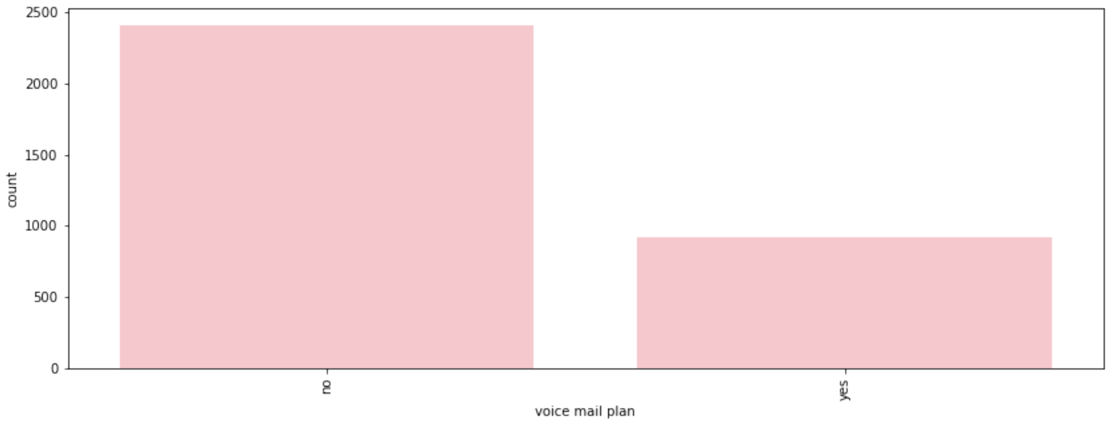


In [18]:
```
df['international plan'].value_counts()
```

Out[18]:
```
no     3010
yes     323
Name: international plan, dtype: int64
```

voice mail plan

In [19]:
```
plot_categorical_distribution(df, 'voice mail plan')
```

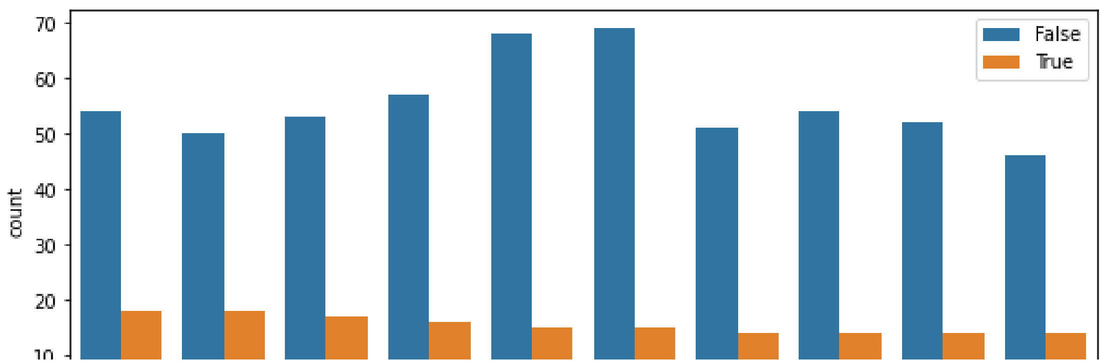


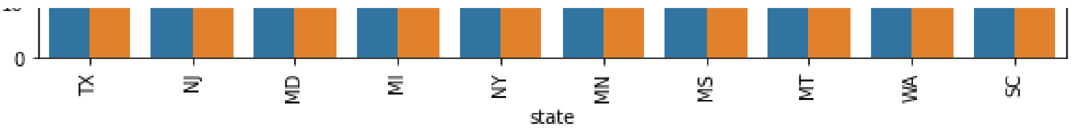
In [20]:
```
df['voice mail plan'].value_counts()
```

Out[20]:
```
no     2411
yes     922
Name: voice mail plan, dtype: int64
```

In [21]:
```
#Checking the distribution of categorical features based on churn rate
def plot_categorical_distribution(df, feature):
    """
    Plots the distribution of a categorical feature in the given data.
    """
    plt.figure(figsize=(10, 4))
    churn_counts = df.groupby(feature)["churn"].sum().sort_values(ascending
    top_10_categories = churn_counts.head(10).index.tolist()
    sns.countplot(x=feature, hue="churn", data=df, order=top_10_categories)
    plt.xticks(rotation=90)
    plt.legend(loc="upper right")
    plt.show()
```

In [22]:
```
plot_categorical_distribution(df, 'state')
```

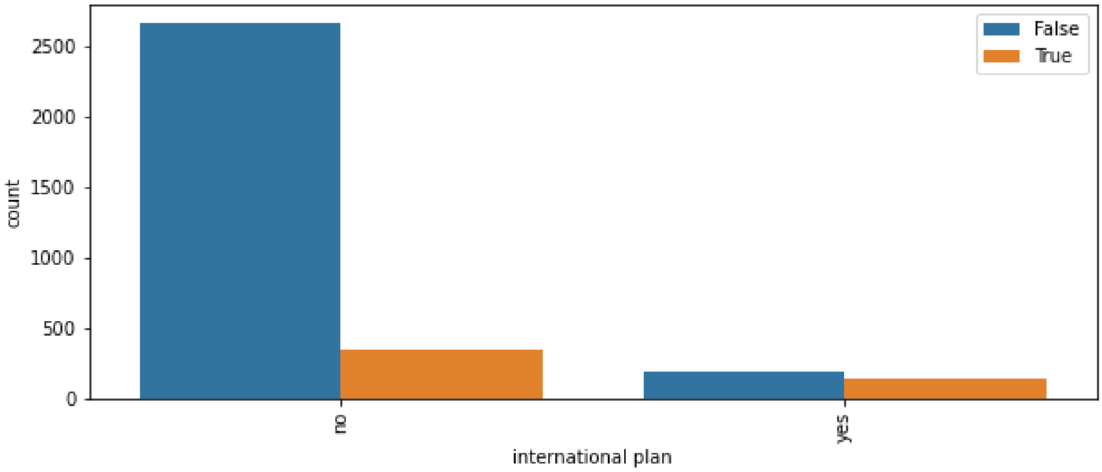




Texas, New Jersey, Maryland,Miami and NewYork has the highest churn rate among the top 10 states, as indicated by the highest orange bar

In [23]:

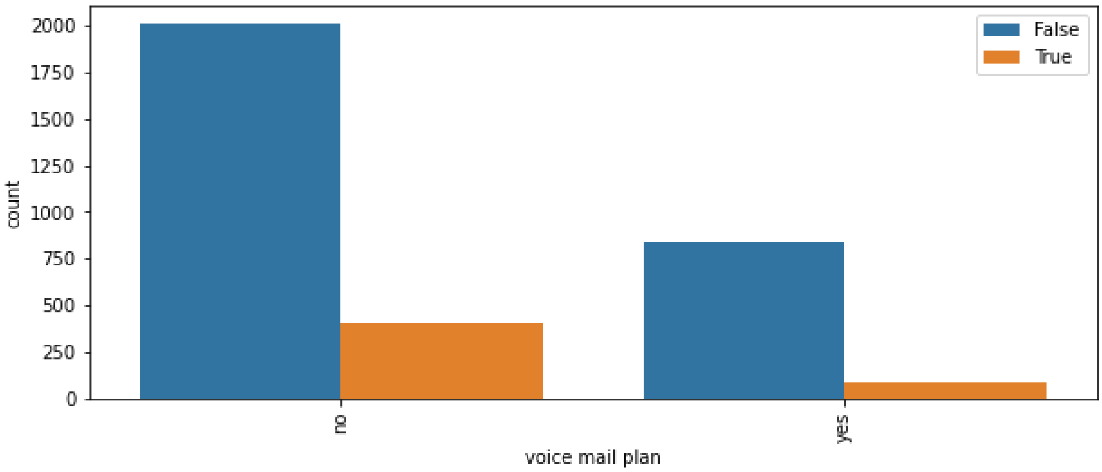
```
#International plan
plot_categorical_distribution(df, 'international plan')
```



Dominance of "no": The chart clearly shows that the majority of customers in the dataset do not have an international plan ("no" bar is significantly taller than the "yes" bar).

In [24]:

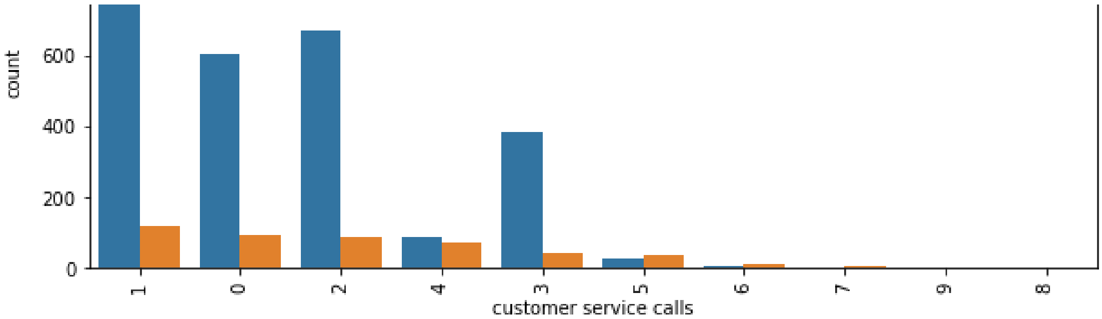
```
#voice mail plan
plot_categorical_distribution(df, 'voice mail plan')
```



most customers who churned did not have an voicemail plan.

In [25]:

```
#customer service calls
plot_categorical_distribution(df, 'customer service calls')
```



the chart suggests that the number of customer service calls is a potential predictor of customer churn. Customers who make more service calls are more likely to churn, while those who make fewer calls are more likely to remain customers. Additionally, the absence of a voicemail plan is observed to be associated with churn

Feature Engineering

In this phase, we'll perform Label Encoding, One Hot Encoding and Scaling the data.

1. Label Encoding -It is a technique used to convert categorical variables into numerical values.

In [26]:

```
# Convert columns with 'yes' or 'no' to binary using LabelEncoder
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
df['churn'] = label_encoder.fit_transform(df['churn'])
```

2. One Hot Encoding -This is a technique used to convert categorical variables into a set of binary features

In [27]:

```
df = pd.get_dummies(df,columns = ['state', 'area code','international plan'
df.head()
```

Out[27]:

	account length	phone number	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	total eve charge	m
0	128	382-4657	25	265.1	110	45.07	197.4	99	16.78	
1	107	371-7191	26	161.6	123	27.47	195.5	103	16.62	
2	137	358-1921	0	243.4	114	41.38	121.2	110	10.30	
3	84	375-9999	0	299.4	71	50.90	61.9	88	5.26	
4	75	330-6626	0	166.7	113	28.34	148.3	122	12.61	

5 rows × 75 columns

Scaling the data

Data scaling refers to the process of transforming the features (or variables) in a dataset to a common scale, without distorting differences in the ranges of values. This process is important when working with machine learning algorithms

In [28]:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

def scaling(columns):
    return scaler.fit_transform(df[columns].values.reshape(-1,1))

for i in df.select_dtypes(include=[np.number]).columns:
    df[i] = scaling(i)
df.head()
```

Out[28]:

	account length	phone number	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls
0	0.524793	382-4657	0.490196	0.755701	0.666667	0.755701	0.542755	0.582353
1	0.438017	371-7191	0.509804	0.460661	0.745455	0.460597	0.537531	0.605882
2	0.561983	358-1921	0.000000	0.693843	0.690909	0.693830	0.333242	0.647059
3	0.342975	375-9999	0.000000	0.853478	0.430303	0.853454	0.170195	0.517647
4	0.305785	330-6626	0.000000	0.475200	0.684848	0.475184	0.407754	0.717647

5 rows × 75 columns

MODELING

Modeling data involves applying machine learning algorithms or statistical methods to make predictions or uncover patterns within the data. It's one of the key steps in data science and analytics. we will build a model that can predict the customer churn based on the features in our dataset.

we will be using the following algorithms:

-Logistic Regression -Decision Tree -Random Forest

In [29]:

```
#convert categorical variables to dumy/indicator variables
df=pd.get_dummies(df, drop_first=True)
```

In [30]:

```
#Defining X and y
X = df.drop("churn", axis=1)
y = df["churn"]
```

In [31]:

```
#Train-Test Split
#Splitting data into train and test sets using a test_size of 0.25

#splitting the data in to train and test sets
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size=0.25, randc
```

Logistic Regression

Logistic Regression is a popular machine learning algorithm used for binary classification tasks, where the goal is to predict one of two possible outcomes (e.g., yes/no, true/false, 1/0).

In [32]:

```
# Standardize features
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [33]:

```
# Train logistic regression model with class_weight='balanced'
model = LogisticRegression(random_state=42, max_iter=1000, class_weight='ba
model.fit(X_train, y_train)
```

Out[33]:

```
LogisticRegression(class_weight='balanced', max_iter=1000, random_state=42)
```

In [34]:

```
# Predictions
y_pred = model.predict(X_test)
```

In [35]:

```
# Evaluate model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
```

In [36]:

```
# Display results
print(f"Accuracy: {accuracy:.4f}")
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)
```

```
Accuracy: 0.8201
Confusion Matrix:
 [[610  99]
 [ 51  74]]
Classification Report:
               precision    recall  f1-score   support

         0.0       0.92      0.86      0.89       709
         1.0       0.43      0.59      0.50       125
```

```
    accuracy                           0.82       834
   macro avg       0.68      0.73      0.69       834
weighted avg       0.85      0.82      0.83       834
```

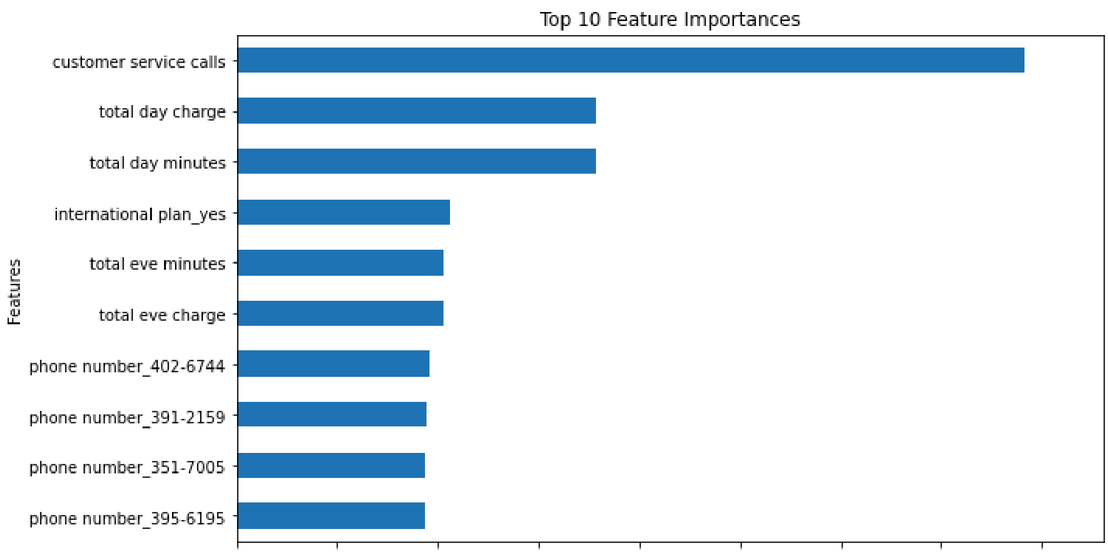
interpretaction

Accuracy: 0.8201 (82.01%) - This means that about 82% of the predictions made by the model are correct. The model performs well in predicting non-churn instances (Class 0.0) with high precision and recall.

However, it struggles with predicting churn instances (Class 1.0), as evidenced by the lower precision and recall. This means the model is more likely to miss churners and has a higher rate of false positives.

In [37]:

```
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt
import pandas as pd

# Fit your logistic regression model first
logreg = LogisticRegression()
logreg.fit(X_train, y_train)

# Access coefficients for the feature importances
importance = logreg.coef_[0]
feature_names = X_train.columns
feature_importances = pd.Series(importance, index=feature_names)

# Sort the feature importances in descending order
feature_importances = feature_importances.sort_values(ascending=False)

# Plot the top 10 features
plt.figure(figsize=(10, 6))
top_features = feature_importances[:10]  # Select the top 10 features
top_features.sort_values().plot(kind='barh')
plt.xlabel('Importance')
plt.ylabel('Features')
plt.title('Top 10 Feature Importances')
plt.xlim(0, max(top_features) * 1.1)  # Set the xlim to the maximum importa
plt.show()
```



According to the model,total day charge, customer service calls,total day minutes are the top three most important features.

Decision Tree model

In [38]:

```
# Train Decision Tree model
from sklearn.tree import DecisionTreeClassifier
dt_model = DecisionTreeClassifier(random_state=42, class_weight='balanced')
dt_model.fit(X_train, y_train)
```

Out[38]:

```
DecisionTreeClassifier(class_weight='balanced', random_state=42)
```

In [39]:

```
# Predictions
y_pred = dt_model.predict(X_test)

# Evaluate model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Display results
print(f"Accuracy: {accuracy:.4f}")
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)
```

```
Accuracy: 0.9209
Confusion Matrix:
 [[674  35]
 [ 31  94]]
Classification Report:
               precision    recall  f1-score   support

         0.0       0.96      0.95      0.95       709
         1.0       0.73      0.75      0.74       125

    accuracy                           0.92       834
   macro avg       0.84      0.85      0.85       834
weighted avg       0.92      0.92      0.92       834
```

Accuracy: 0.9209 (92.09%) - This means that about 92% of the predictions made by the model are correct. The model performs exceptionally well in predicting non-churn instances (Class 0.0) with high precision and recall.

It also shows significant improvement in predicting churn instances (Class 1.0) with better precision and recall compared to the previous model

Random Forest model

In [40]:

```
# Train Random Forest model
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier(random_state=42, class_weight='balanced',
rf_model.fit(X_train, y_train)
```

```
rf_model.fit(X_train, y_train)

# Predictions
y_pred = rf_model.predict(X_test)
```

In [41]:

```
# Evaluate model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Display results
print(f"Accuracy: {accuracy:.4f}")
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)
```

```
Accuracy: 0.8861
Confusion Matrix:
 [[709   0]
 [ 95  30]]
Classification Report:
               precision    recall  f1-score   support

         0.0       0.88      1.00      0.94       709
         1.0       1.00      0.24      0.39       125

    accuracy                           0.89       834
   macro avg       0.94      0.62      0.66       834
weighted avg       0.90      0.89      0.85       834
```

Accuracy: 0.8861 (88.61%) - This means that about 89% of the predictions made by the model are correct. The model performs exceptionally well in predicting non-churn instances (Class 0.0) with high recall and a good balance between precision and recall.

However, it struggles significantly in predicting churn instances (Class 1.0), with a high precision but very low recall. This means the model correctly identifies churners when it predicts them, but misses a lot of actual churners.

Model Evaluation

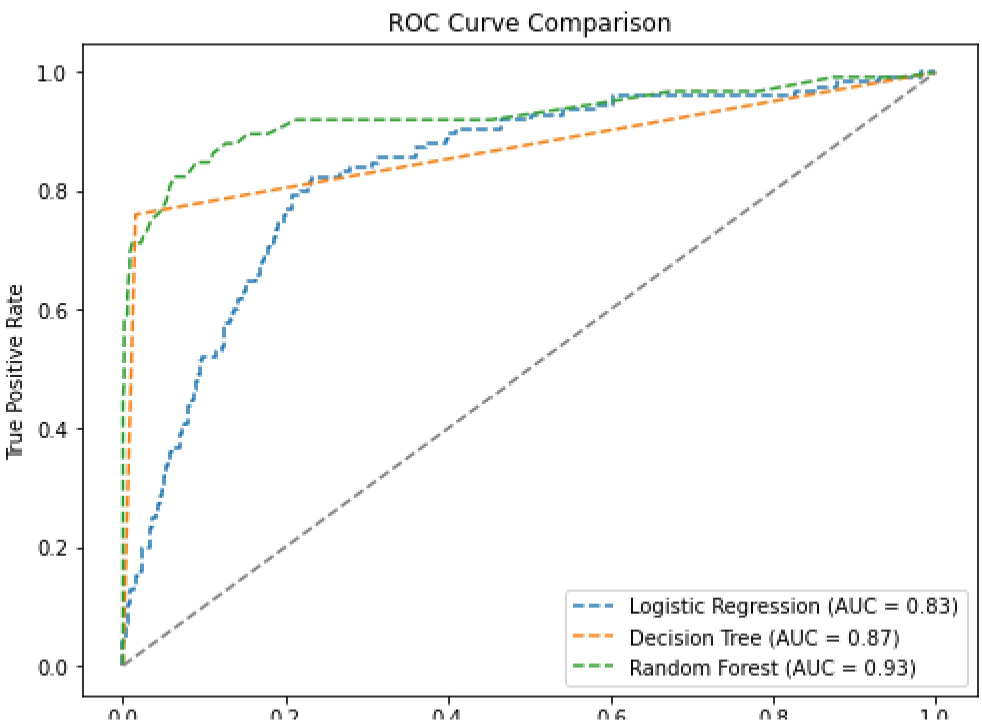
Model evaluation refers to the process of assessing how well a machine learning model performs on a given task. This involves measuring the model's ability to make accurate predictions and generalize to new, unseen data

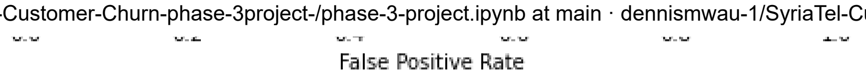
Models Comparison - ROC Curve

In [42]:

```
# Compute ROC Curves
# Import necessary libraries
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Initialize models
logreg = LogisticRegression()
dt = DecisionTreeClassifier()
rf = RandomForestClassifier()

# Train models (ensure you have your X_train, y_train)
logreg.fit(X_train, y_train)
dt.fit(X_train, y_train)
rf.fit(X_train, y_train)

# Get predicted probabilities for the positive class (class 1)
log_probs = logreg.predict_proba(X_test)[:, 1]
dt_probs = dt.predict_proba(X_test)[:, 1]
rf_probs = rf.predict_proba(X_test)[:, 1]

# Compute ROC Curves
log_fpr, log_tpr, _ = roc_curve(y_test, log_probs)
dt_fpr, dt_tpr, _ = roc_curve(y_test, dt_probs)
rf_fpr, rf_tpr, _ = roc_curve(y_test, rf_probs)

# Compute AUC Scores
log_auc = auc(log_fpr, log_tpr)
dt_auc = auc(dt_fpr, dt_tpr)
rf_auc = auc(rf_fpr, rf_tpr)

# Plot ROC Curves
plt.figure(figsize=(8, 6))
plt.plot(log_fpr, log_tpr, linestyle='--', label=f'Logistic Regression (AUC
plt.plot(dt_fpr, dt_tpr, linestyle='--', label=f'Decision Tree (AUC = {dt_a
plt.plot(rf_fpr, rf_tpr, linestyle='--', label=f'Random Forest (AUC = {rf_a
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')  # Diagonal line for
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Comparison')
plt.legend()
plt.show()
```





AUC (Area Under the Curve) Scores:

Logistic Regression (AUC = 0.83): Performs well but is outperformed by tree-based models. Decision Tree (AUC = 0.86): Better than logistic regression but still not the best. Random Forest (AUC = 0.93): The best-performing model, with the highest AUC, meaning it has the strongest ability to differentiate between classes. Model Performance:

The Random Forest model (green curve) is closest to the top-left corner, meaning it has the highest True Positive Rate (TPR) while keeping a low False Positive Rate (FPR). The Decision Tree model (orange curve) performs better than Logistic Regression but is still lower than Random Forest. Logistic Regression (blue curve), while decent, has a more gradual rise, meaning it struggles more with classification. Best Model: Since higher AUC values indicate better performance, the Random Forest model (AUC = 0.93) is the best model for this dataset.

RECOMMENDATIONS

-Most customers do not have an international plan or voicemail plan. The business should think about promoting these services to customers who might benefit from them but do not currently have them. -increase marketing in states with higher churn rates, such as Texas, New Jersey, Maryland, Miami, and New York. -Total day minutes, total eve charge, total night minutes, and total international calls and charges were identified as influential predictors the company can