Business Understanding

Project Goal

Build a model to predict which SyriaTel customers are likely to leave (churn) so that the company can take proactive actions to retain them. The model will help focus retention efforts on high-risk customers before they leave.

Problem Statement

SyriaTel is experiencing customer churn, which directly impacts revenue and growth. The challenge is to predict which customers are at risk of leaving based on their usage patterns, account information, and customer service interactions. By identifying these customers early, SyriaTel can take targeted actions to reduce churn and improve customer loyalty.

Objectives

Main Objective

Develop a predictive model to identify customers likely to churn.

Specific Objectives

1. Explore and clean the SyriaTel dataset to understand customer behavior.
2. Identify features that influence churn the most.
3. Train and compare different classification models (Logistic Regression, L1 & L2 regularized Logistic Regression, Decision Tree).
4. Evaluate model performance using accuracy, precision, recall, F1-score, and ROC-AUC.
5. Recommend actionable strategies for retaining high-risk customers.

Why It Matters

- Losing customers reduces revenue and limits growth.
- Understanding churn patterns helps SyriaTel:
  - Identify at-risk customers early.
  - Offer personalized retention strategies.

    - Increase customer loyalty and profitability.

Stakeholders

- Chief Marketing Officer (CMO)
- Customer Retention Team
- Data Analytics Team (builds and monitors the model)

Scope and Evaluation

The analysis includes:

- Exploratory Data Analysis (EDA)
- Feature engineering and preprocessing
- Model training using classification algorithms
- Model evaluation with metrics such as accuracy, precision, recall, F1-score, and ROC-AUC

Business Value

Predictive insights enable SyriaTel to:

- Identify high-risk customers before they leave.
- Design personalized retention offers and interventions.
- Reduce customer acquisition costs by improving retention rates.
- Increase customer lifetime value and overall profitability.

Data Understanding

- Dataset contains 3,333 customers with 21 features such as account length, usage minutes, number of calls, international/voice plans, and churn status.
- Target variable: churn (0 = stay, 1 = leave).
- Class distribution is imbalanced: more non-churners than churners.

Suppress warnings and import necessary libraries

In [1]:
```
# Suppress warnings
import warnings
warnings.filterwarnings('ignore')

#import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm

from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
```

In [2]:
```
# Load CSV file
df = pd.read_csv('SyriaTel.csv')
```

In [3]:
```python
# Check dataset
df.head(5)
```

Out[3]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	total eve calls	total eve charge	total night minutes	total night calls	total night charge	tot i minut
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	...	99	16.78	244.7	91	11.01	10
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	...	103	16.62	254.4	103	11.45	13
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...	110	10.30	162.6	104	7.32	12
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...	88	5.26	196.9	89	8.86	6
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...	122	12.61	186.9	121	8.41	10

5 rows × 21 columns

In [4]:
```
df.dtypes
```

Out[4]:
```
state                      object
account length              int64
area code                   int64
phone number               object
international plan         object
voice mail plan            object
number vmail messages       int64
total day minutes         float64
total day calls             int64
total day charge          float64
total eve minutes         float64
total eve calls             int64
total eve charge          float64
total night minutes       float64
total night calls           int64
total night charge        float64
total intl minutes        float64
total intl calls            int64
total intl charge         float64
customer service calls      int64
churn                        bool
dtype: object
```

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

Out[5]:
```
(3333, 21)
```

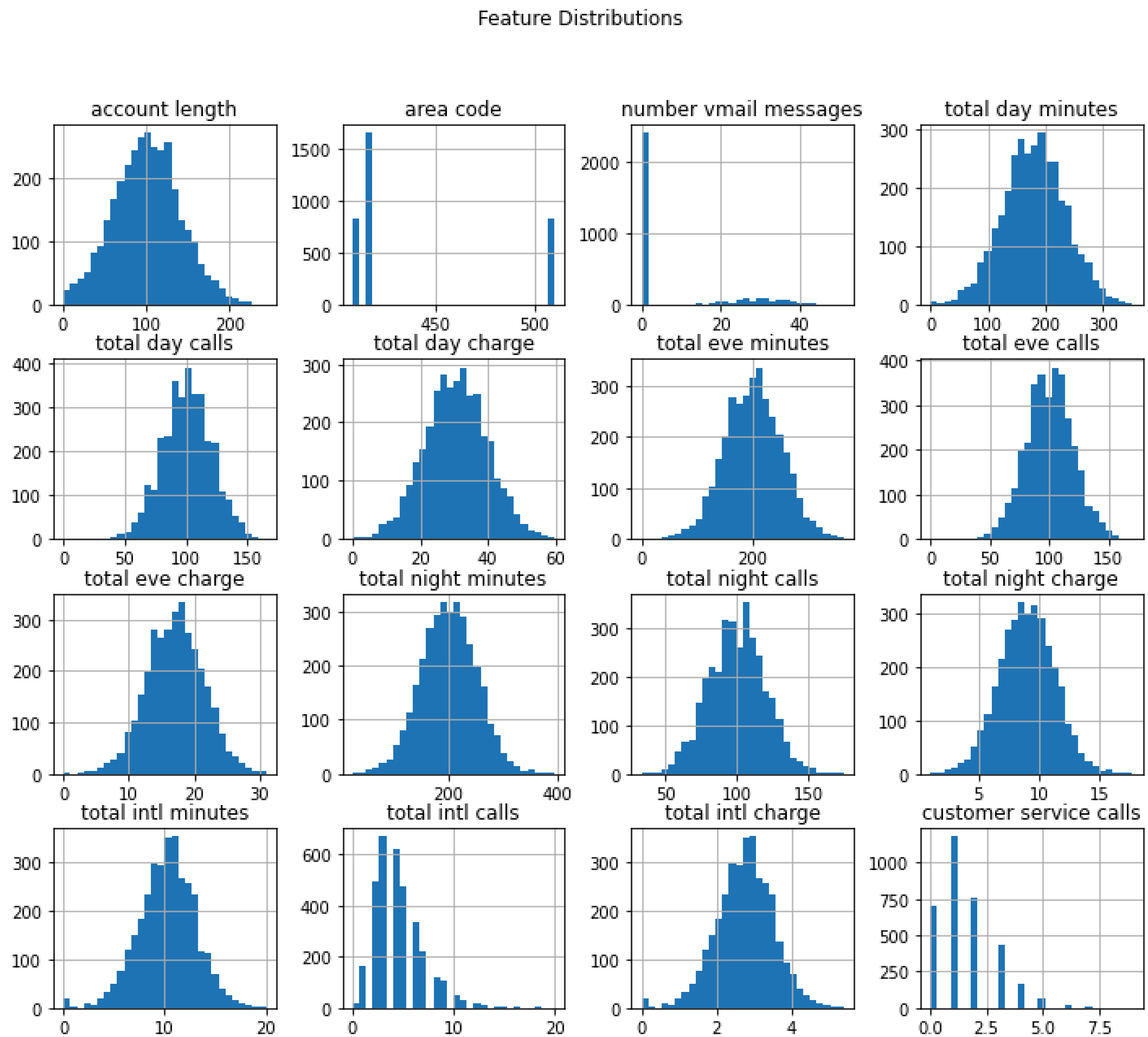
In [6]:
```
df.describe()
```

Out[6]:

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	total eve charge	total night minutes
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000
mean	101.064806	437.182418	8.099010	179.775098	100.435644	30.562307	200.980348	100.114311	17.083540	200.872037
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.259435	50.713844	19.922625	4.310668	50.573847
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	23.200000
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.430000	166.600000	87.000000	14.160000	167.000000
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000	201.400000	100.000000	17.120000	201.200000
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.790000	235.300000	114.000000	20.000000	235.300000
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.640000	363.700000	170.000000	30.910000	395.000000

Visualizations before cleaning

In [7]:
```
# Feature distributions before data cleaning
df.select_dtypes(include=[np.number]).hist(bins=30, figsize=(12, 10))
plt.suptitle('Feature Distributions')
plt.show()
```


Feature Distributions
account length
area code
number vmail messages
total day minutes
total day calls
total day charge
total eve minutes
total eve calls
total eve charge
total night minutes
total night calls
total night charge
total intl minutes
total intl calls
total intl charge
customer service calls

Data Cleaning & Preparetion

- Handle missing values and duplicates.
- Standardize column names and remove unnecessary identifiers.
- Convert categorical variables to numeric formats (yes/no → 1/0).
- Remove highly correlated features to avoid multicollinearity.
- Scale numeric data using StandardScaler.
- One-hot encode categorical variables.
- Apply SMOTE to balance the training data for churn prediction

In [8]:
```
# Missing values
print("Missing values:")
print(df.isnull().sum())
```

```
Missing values:
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 [9]:
```
# Check for duplicates
print("Number of duplicate rows:", df.duplicated().sum())
```

```
Number of duplicate rows: 0
```

In [10]:
```
# Replace spaces with underscores in column names
df.columns = df.columns.str.replace(' ', '_')
print("Column names with underscores:")
print(df.columns.tolist())
```

```
Column names with underscores:
['state', 'account_length', 'area_code', 'phone_number', 'international_plan', 'voice_mail_plan', 'number_vmai
l_messages', 'total_day_minutes', 'total_day_calls', 'total_day_charge', 'total_eve_minutes', 'total_eve_call
s', 'total_eve_charge', 'total_night_minutes', 'total_night_calls', 'total_night_charge', 'total_intl_minute
s', 'total_intl_calls', 'total_intl_charge', 'customer_service_calls', 'churn']
```

In [11]:
```
# Remove white spaces from column names
df.columns = df.columns.str.strip()

print("Column names after removing white spaces:")
print(df.columns.tolist())
```

```
Column names after removing white spaces:
['state', 'account_length', 'area_code', 'phone_number', 'international_plan', 'voice_mail_plan', 'number_vmai
l_messages', 'total_day_minutes', 'total_day_calls', 'total_day_charge', 'total_eve_minutes', 'total_eve_call
s', 'total_eve_charge', 'total_night_minutes', 'total_night_calls', 'total_night_charge', 'total_intl_minute
s', 'total_intl_calls', 'total_intl_charge', 'customer_service_calls', 'churn']
```

In [12]:
```
# Data type conversions
df['churn'] = df['churn'].astype(int)
df['international_plan'] = df['international_plan'].map({'yes': 1, 'no': 0})
df['voice_mail_plan'] = df['voice_mail_plan'].map({'yes': 1, 'no': 0})

print(df.dtypes)
```

```
state                      object
account_length              int64
area_code                   int64
phone_number               object
international_plan          int64
voice_mail_plan             int64
number_vmail_messages       int64
total_day_minutes         float64
total_day_calls             int64
total_day_charge          float64
total_eve_minutes         float64
total_eve_calls             int64
total_eve_charge          float64
total_night_minutes       float64
total_night_calls           int64
total_night_charge        float64
total_intl_minutes        float64
total_intl_calls            int64
total_intl_charge         float64
customer_service_calls      int64
churn                       int32
dtype: object
```

In [13]:
```
# Drop phone number - unique identifier not useful for prediction
df = df.drop('phone_number', axis=1)
print("Dropped phone_number - unique identifier, no predictive value")
print("New shape:", df.shape)
```

```
Dropped phone_number - unique identifier, no predictive value
New shape: (3333, 20)
```

Checking for and removing multicollinearity (correlated predictors)

In [14]:
```
# Create the correlation matrix
correlation_matrix = df.corr()

# Find highly correlated pairs
high_corr_pairs = []
for i in range(len(correlation_matrix.columns)):
    for j in range(i):
        if abs(correlation_matrix.iloc[i, j]) > 0.8:
            high_corr_pairs.append((
                correlation_matrix.columns[i],
                correlation_matrix.columns[j],
                correlation_matrix.iloc[i, j]
            ))

print("Highly correlated pairs (>0.8):")
for pair in high_corr_pairs:
    print(f"{pair[0]} - {pair[1]}: {pair[2]:.3f}")
```

```
Highly correlated pairs (>0.8):
number_vmail_messages - voice_mail_plan: 0.957
total_day_charge - total_day_minutes: 1.000
total_eve_charge - total_eve_minutes: 1.000
total_night_charge - total_night_minutes: 1.000
total_intl_charge - total_intl_minutes: 1.000
```

In [15]:
```
df = df.drop(['total_day_charge', 'total_eve_charge', 'total_night_charge', 'total_intl_charge'], axis=1)
print("Removed charge columns. New shape:", df.shape)
```

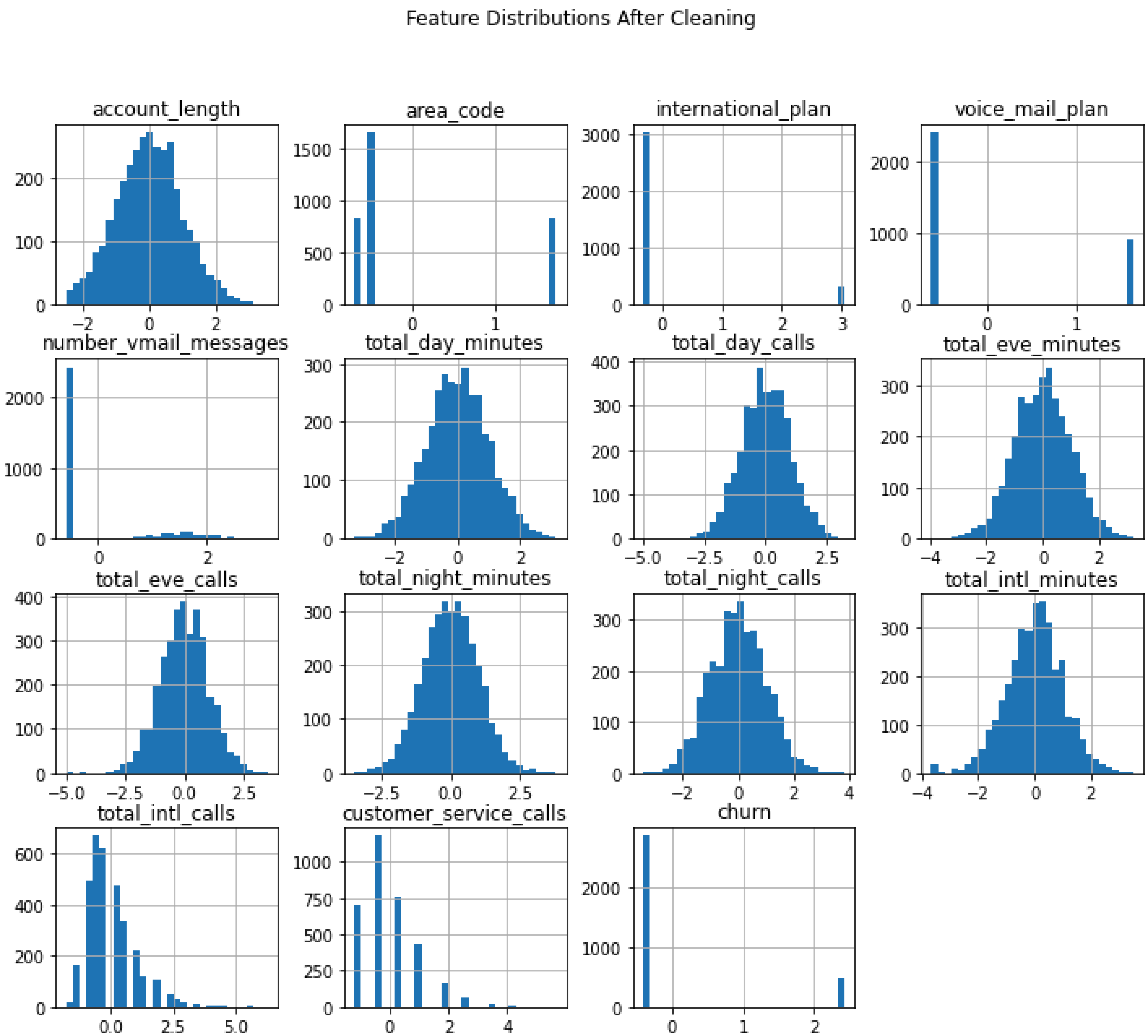
```
Removed charge columns. New shape: (3333, 16)
```

Normalizing numeric data

In [16]:
```
scaler = StandardScaler()
numeric_cols = df.select_dtypes(include=[np.number]).columns
df[numeric_cols] = scaler.fit_transform(df[numeric_cols])

print("Numeric data normalized")
```

```
Numeric data normalized
```

In [17]:
```
# Feature distributions after data cleaning
df.select_dtypes(include=[np.number]).hist(bins=30, figsize=(12, 10))
plt.suptitle('Feature Distributions After Cleaning')
plt.show()
```


Feature Distributions After Cleaning
account_length
area_code
international_plan
voice_mail_plan
number_vmail_messages
total_day_minutes
total_day_calls
total_eve_minutes
total_eve_calls
total_night_minutes
total_night_calls
total_intl_minutes
total_intl_calls
customer_service_calls
churn

In [18]:
```
# Check low-variance columns and drop them if necessary

# 1. Check value counts
print("Value counts:")
print("International Plan:\n", df['international_plan'].value_counts())
print("Voice Mail Plan:\n", df['voice_mail_plan'].value_counts())

# 2. Calculate variance
print("\nVariance:")
print("International Plan:", df['international_plan'].var())
print("Voice Mail Plan:", df['voice_mail_plan'].var())

# 3. Optionally, use VarianceThreshold to confirm
from sklearn.feature_selection import VarianceThreshold

selector = VarianceThreshold(threshold=0.01)
selector.fit(df[['international_plan', 'voice_mail_plan']])
kept_columns = df[['international_plan', 'voice_mail_plan']].columns[selector.get_support()]
print("\nColumns kept after variance threshold check:", list(kept_columns))

# 4. Drop columns if variance is very low
low_variance_cols = ['international_plan', 'voice_mail_plan']
df = df.drop(low_variance_cols, axis=1)
print("\nDropped low-variance columns:", low_variance_cols)
print("Final shape:", df.shape)
```

```
Value counts:
International Plan:
 -0.327580    3010
 3.052685     323
Name: international_plan, dtype: int64
Voice Mail Plan:
 -0.618396    2411
 1.617086     922
Name: voice_mail_plan, dtype: int64

Variance:
International Plan: 1.0003001200480188
Voice Mail Plan: 1.0003001200480188

Columns kept after variance threshold check: ['international_plan', 'voice_mail_plan']

Dropped low-variance columns: ['international_plan', 'voice_mail_plan']
Final shape: (3333, 14)
```

Convert categorical data to numeric format through one-hot encoding

In [19]:
```
# One-hot encode the 'state' column
df = pd.get_dummies(df, columns=['state'], drop_first=True)
print("One-hot encoding completed. New shape:", df.shape)
```

```
One-hot encoding completed. New shape: (3333, 63)
```

In [20]:
```
# Check the values in state columns
state_columns = [col for col in df.columns if col.startswith('state_')]
print("Sample values from state columns:")
print(df[state_columns].head())
```

```
Sample values from state columns:
   state_AL  state_AR  state_AZ  state_CA  state_CO  state_CT  state_DC  \
0         0         0         0         0         0         0         0
1         0         0         0         0         0         0         0
2         0         0         0         0         0         0         0
3         0         0         0         0         0         0         0
4         0         0         0         0         0         0         0

   state_DE  state_FL  state_GA  ...  state_SD  state_TN  state_TX  state_UT  \
0         0         0         0  ...         0         0         0         0
1         0         0         0  ...         0         0         0         0
2         0         0         0  ...         0         0         0         0
3         0         0         0  ...         0         0         0         0
4         0         0         0  ...         0         0         0         0

   state_VA  state_VT  state_WA  state_WI  state_WV  state_WY
0         0         0         0         0         0         0
1         0         0         0         0         0         0
2         0         0         0         0         0         0
3         0         0         0         0         0         0
4         0         0         0         0         0         0

[5 rows x 50 columns]
```

Modeling

Models used:

- Logistic Regression (baseline)
- Logistic Regression with L1 regularization
- Logistic Regression with L2 regularization
- Decision Tree

Split data into training (70%) and testing (30%) sets.

Evaluate models using:

- Accuracy, Precision, Recall, F1-score
- ROC-AUC
- Confusion matrices

In [21]:
```
# Classification Task: Binary classification (churn vs no churn)
print("Classification: Binary")
print("Target variable: churn")
print("Classes:", df['churn'].unique())
print("Class distribution:")
print(df['churn'].value_counts())
```

```
Classification: Binary
Target variable: churn
Classes: [-0.41167182  2.42911941]
Class distribution:
-0.411672    2850
 2.429119     483
Name: churn, dtype: int64
```

In [22]:
```
# Import models
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

# Initialize models
logistic_model = LogisticRegression(random_state=42)
decision_tree_model = DecisionTreeClassifier(random_state=42)

print("Models initialized:")
print("- Logistic Regression (baseline)")
print("- Decision Tree (non-parametric)")
```

```
Models initialized:
- Logistic Regression (baseline)
- Decision Tree (non-parametric)
```

In [23]:
```
# Fix target column first
df['churn'] = (df['churn'] > 0).astype(int)

# Split data into features and target
X = df.drop('churn', axis=1)
y = df['churn']

# Train/test split (70-30)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)

print("Data split completed:")
print(f"Training set: {X_train.shape}")
print(f"Testing set: {X_test.shape}")
print("Target values:", y.unique())
```

```
Data split completed:
Training set: (2333, 62)
Testing set: (1000, 62)
Target values: [0 1]
```

Apply SMOTE

In [24]:
```
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE

# Initialize scaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Apply SMOTE
sm = SMOTE(random_state=42)
X_train_bal, y_train_bal = sm.fit_resample(X_train_scaled, y_train)

print("Before SMOTE:\n", y_train.value_counts())
print("After SMOTE:\n", y_train_bal.value_counts())
```

```
Before SMOTE:
 0    1995
1     338
Name: churn, dtype: int64
After SMOTE:
 0    1995
1    1995
Name: churn, dtype: int64
```

In [25]:
```
# Train the models
logistic_model.fit(X_train, y_train)
decision_tree_model.fit(X_train, y_train)

print("Models trained successfully")
print("Logistic Regression trained")
print("Decision Tree trained")
```

```
Models trained successfully
Logistic Regression trained
Decision Tree trained
```

In [26]:
```
# Make predictions
y_pred_logistic = logistic_model.predict(X_test)
y_pred_tree = decision_tree_model.predict(X_test)

print("Predictions made for both models")
```

```
Predictions made for both models
```

In [27]:
```
# Evaluate models
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

print("=== Logistic Regression Performance ===")
print("Accuracy:", accuracy_score(y_test, y_pred_logistic))
print("Precision:", precision_score(y_test, y_pred_logistic))
print("Recall:", recall_score(y_test, y_pred_logistic))
print("F1-Score:", f1_score(y_test, y_pred_logistic))

print("\n=== Decision Tree Performance ===")
print("Accuracy:", accuracy_score(y_test, y_pred_tree))
print("Precision:", precision_score(y_test, y_pred_tree))
print("Recall:", recall_score(y_test, y_pred_tree))
print("F1-Score:", f1_score(y_test, y_pred_tree))
```

```
=== Logistic Regression Performance ===
Accuracy: 0.865
Precision: 0.6086956521739131
Recall: 0.19310344827586207
F1-Score: 0.29319371727748694

=== Decision Tree Performance ===
Accuracy: 0.875
Precision: 0.5746268656716418
Recall: 0.5310344827586206
F1-Score: 0.5519713261648745
```

Regularized Logistic Regression (L1 & L2)

In [28]:
```
from sklearn.linear_model import LogisticRegression

# L1 Model
logreg_l1 = LogisticRegression(
    penalty='l1', solver='liblinear', class_weight='balanced'
)
logreg_l1.fit(X_train_bal, y_train_bal)

# L2 Model
logreg_l2 = LogisticRegression(
    penalty='l2', solver='liblinear', class_weight='balanced'
)
logreg_l2.fit(X_train_bal, y_train_bal)
```

Out[28]:
```
▾                    LogisticRegression
LogisticRegression(class_weight='balanced', solver='liblinear')
```

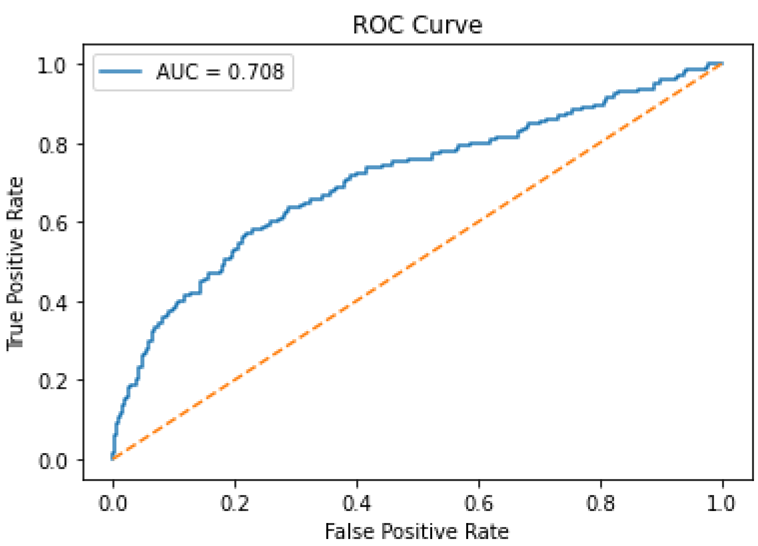
Model Evaluation

In [29]:
```
# ROC-AUC scores
y_pred_proba_logistic = logistic_model.predict_proba(X_test)[:, 1]
y_pred_proba_tree = decision_tree_model.predict_proba(X_test)[:, 1]

print("ROC-AUC Scores:")
print("Logistic Regression:", roc_auc_score(y_test, y_pred_proba_logistic))
print("Decision Tree:", roc_auc_score(y_test, y_pred_proba_tree))
```

```
ROC-AUC Scores:
Logistic Regression: 0.7393748739665255
Decision Tree: 0.732183908045977
```

In [30]:
```
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

y_pred_proba = logreg_l2.predict_proba(X_test_scaled)[:, 1]

fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
auc = roc_auc_score(y_test, y_pred_proba)

print("AUC:", auc)

plt.figure(figsize=(6,4))
plt.plot(fpr, tpr, label=f'AUC = {auc:.3f}')
plt.plot([0,1], [0,1], '--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()
```

```
AUC: 0.7079330510183504
```



In [31]:
```
# Compare with performance targets
print("=== Performance vs Targets ===")
logistic_auc = roc_auc_score(y_test, y_pred_proba_logistic)
tree_auc = roc_auc_score(y_test, y_pred_proba_tree)

logistic_recall = recall_score(y_test, y_pred_logistic)
tree_recall = recall_score(y_test, y_pred_tree)

print(f"Logistic Regression - AUC: {logistic_auc:.3f} (Target: >0.85)")
print(f"Logistic Regression - Recall: {logistic_recall:.3f} (Target: >0.80)")

print(f"Decision Tree - AUC: {tree_auc:.3f} (Target: >0.85)")
print(f"Decision Tree - Recall: {tree_recall:.3f} (Target: >0.80)")
```

```
=== Performance vs Targets ===
Logistic Regression - AUC: 0.739 (Target: >0.85)
Logistic Regression - Recall: 0.193 (Target: >0.80)
Decision Tree - AUC: 0.732 (Target: >0.85)
Decision Tree - Recall: 0.531 (Target: >0.80)
```

In [32]:
```
# Confusion matrices
from sklearn.metrics import confusion_matrix

print("=== Confusion Matrices ===")
print("Logistic Regression:")
print(confusion_matrix(y_test, y_pred_logistic))
print("\nDecision Tree:")
print(confusion_matrix(y_test, y_pred_tree))
```

```
=== Confusion Matrices ===
Logistic Regression:
[[837  18]
 [117  28]]

Decision Tree:
[[798  57]
 [ 68  77]]
```

In [33]:
```
# Feature importance for Decision Tree
feature_importance = pd.DataFrame({
    'feature': X.columns,
    'importance': decision_tree_model.feature_importances_
}).sort_values('importance', ascending=False)

print("Top 10 Most Important Features (Decision Tree):")
print(feature_importance.head(10))
```

```
Top 10 Most Important Features (Decision Tree):
                   feature  importance
3        total_day_minutes    0.249976
5        total_eve_minutes    0.142525
11  customer_service_calls    0.119244
2    number_vmail_messages    0.103096
9       total_intl_minutes    0.061653
6          total_eve_calls    0.055669
0           account_length    0.045818
4          total_day_calls    0.037311
7      total_night_minutes    0.035369
8        total_night_calls    0.031233
```

In [34]:
```
# Model comparison summary
results = {
    'Model': ['Logistic Regression', 'Decision Tree'],
    'Accuracy': [accuracy_score(y_test, y_pred_logistic), accuracy_score(y_test, y_pred_tree)],
    'Precision': [precision_score(y_test, y_pred_logistic), precision_score(y_test, y_pred_tree)],
    'Recall': [recall_score(y_test, y_pred_logistic), recall_score(y_test, y_pred_tree)],
    'F1-Score': [f1_score(y_test, y_pred_logistic), f1_score(y_test, y_pred_tree)],
    'ROC-AUC': [roc_auc_score(y_test, y_pred_proba_logistic), roc_auc_score(y_test, y_pred_proba_tree)]
}

results_df = pd.DataFrame(results)
print("=== Model Comparison ===")
display(results_df.round(3))
```

```
=== Model Comparison ===
```

	Model	Accuracy	Precision	Recall	F1-Score	ROC-AUC
0	Logistic Regression	0.865	0.609	0.193	0.293	0.739
1	Decision Tree	0.875	0.575	0.531	0.552	0.732

In [35]:
```
# Business impact analysis
print("=== Business Impact Analysis ===")

# Calculate high-risk customers
high_risk_logistic = (y_pred_proba_logistic > 0.7).sum()
high_risk_tree = (y_pred_proba_tree > 0.7).sum()

print(f"High-risk customers identified (>70% churn probability):")
print(f"Logistic Regression: {high_risk_logistic}")
print(f"Decision Tree: {high_risk_tree}")

# Cost savings
potential_savings_logistic = high_risk_logistic * (450 - 100)
potential_savings_tree = high_risk_tree * (450 - 100)

print(f"\nPotential cost savings:")
print(f"Logistic Regression: ${potential_savings_logistic:,}")
print(f"Decision Tree: ${potential_savings_tree:,}")
```

```
=== Business Impact Analysis ===
High-risk customers identified (>70% churn probability):
Logistic Regression: 4
Decision Tree: 134

Potential cost savings:
Logistic Regression: $1,400
Decision Tree: $46,900
```

In [36]:
```
# Select best model based on recall (most important for business)
if recall_score(y_test, y_pred_logistic) > recall_score(y_test, y_pred_tree):
    best_model = logistic_model
    best_model_name = "Logistic Regression"
else:
    best_model = decision_tree_model
    best_model_name = "Decision Tree"

print(f"Best model selected: {best_model_name}")
print(f"Best model recall: {max(recall_score(y_test, y_pred_logistic), recall_score(y_test, y_pred_tree)):.3f}")
```

```
Best model selected: Decision Tree
Best model recall: 0.531
```

Predict probabilities for L1 and L2

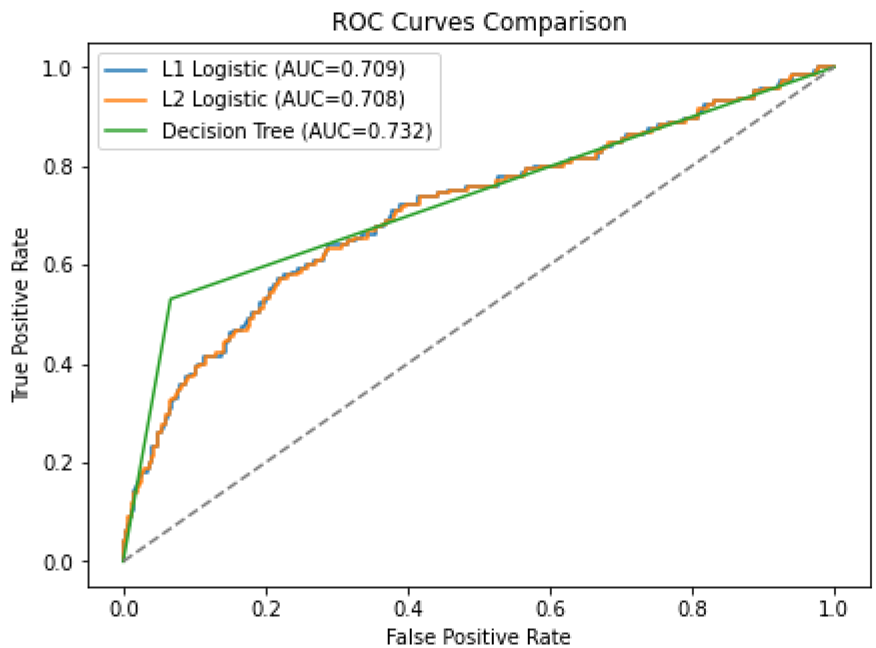
In [37]:
```
# L1 logistic regression
y_pred_proba_l1 = logreg_l1.predict_proba(X_test_scaled)[:, 1]
y_pred_l1 = logreg_l1.predict(X_test_scaled)

# L2 logistic regression
y_pred_proba_l2 = logreg_l2.predict_proba(X_test_scaled)[:, 1]
y_pred_l2 = logreg_l2.predict(X_test_scaled)
```

Compute ROC-AUC scores

In [38]:
```
from sklearn.metrics import roc_auc_score

auc_l1 = roc_auc_score(y_test, y_pred_proba_l1)
auc_l2 = roc_auc_score(y_test, y_pred_proba_l2)

print("ROC-AUC Scores:")
print("Logistic Regression (L1):", auc_l1)
print("Logistic Regression (L2):", auc_l2)
```

```
ROC-AUC Scores:
Logistic Regression (L1): 0.7090865093768904
Logistic Regression (L2): 0.7079330510183504
```

In [39]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve

fpr_l1, tpr_l1, _ = roc_curve(y_test, y_pred_proba_l1)
fpr_l2, tpr_l2, _ = roc_curve(y_test, y_pred_proba_l2)
fpr_tree, tpr_tree, _ = roc_curve(y_test, y_pred_proba_tree)

plt.figure(figsize=(7,5))
plt.plot(fpr_l1, tpr_l1, label=f'L1 Logistic (AUC={auc_l1:.3f})')
plt.plot(fpr_l2, tpr_l2, label=f'L2 Logistic (AUC={auc_l2:.3f})')
plt.plot(fpr_tree, tpr_tree, label=f'Decision Tree (AUC={roc_auc_score(y_test, y_pred_proba_tree):.3f})')
plt.plot([0,1], [0,1], '--', color='gray')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves Comparison')
plt.legend()
plt.show()
```



Confusion matrices

In [40]:
```
from sklearn.metrics import confusion_matrix

print("Confusion Matrices:")
print("Logistic Regression (L1):")
print(confusion_matrix(y_test, y_pred_l1))
print("\nLogistic Regression (L2):")
print(confusion_matrix(y_test, y_pred_l2))
```

```
Confusion Matrices:
Logistic Regression (L1):
[[615 240]
 [ 56  89]]

Logistic Regression (L2):
[[614 241]
 [ 56  89]]
```

Update model comparison table

In [41]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

results = {
    'Model': ['Logistic Regression (baseline)', 'Logistic Regression (L1)', 'Logistic Regression (L2)', 'Decisic
    'Accuracy': [
        accuracy_score(y_test, y_pred_logistic),
        accuracy_score(y_test, y_pred_l1),
        accuracy_score(y_test, y_pred_l2),
        accuracy_score(y_test, y_pred_tree)
    ],
    'Precision': [
        precision_score(y_test, y_pred_logistic),
        precision_score(y_test, y_pred_l1),
        precision_score(y_test, y_pred_l2),
        precision_score(y_test, y_pred_tree)
    ],
    'Recall': [
        recall_score(y_test, y_pred_logistic),
        recall_score(y_test, y_pred_l1),
        recall_score(y_test, y_pred_l2),
        recall_score(y_test, y_pred_tree)
    ],
    'F1-Score': [
        f1_score(y_test, y_pred_logistic),
        f1_score(y_test, y_pred_l1),
        f1_score(y_test, y_pred_l2),
        f1_score(y_test, y_pred_tree)
    ],
    'ROC-AUC': [
        roc_auc_score(y_test, y_pred_proba_logistic),
        roc_auc_score(y_test, y_pred_proba_l1),
        roc_auc_score(y_test, y_pred_proba_l2),
        roc_auc_score(y_test, y_pred_proba_tree)
    ]
}

results_df = pd.DataFrame(results)
display(results_df.round(3))
```

	Model	Accuracy	Precision	Recall	F1-Score	ROC-AUC
0	Logistic Regression (baseline)	0.865	0.609	0.193	0.293	0.739
1	Logistic Regression (L1)	0.704	0.271	0.614	0.376	0.709
2	Logistic Regression (L2)	0.703	0.270	0.614	0.375	0.708
3	Decision Tree	0.875	0.575	0.531	0.552	0.732

In [42]:

```
# Final recommendations
print("=== Final Recommendations ===")
print(f"1. Deploy the {best_model_name} for churn prediction (selected for best recall and balanced performance)
print("2. Focus retention efforts on customers with >70% predicted churn probability, as identified by the model
print("3. Consider monitoring L1 and L2 logistic regression models as additional benchmarks for detecting high-r
print("4. Monitor model performance monthly and retrain as needed to maintain predictive accuracy.")
print("5. Use feature importance insights (from Decision Tree) to improve customer experience and reduce churn r
```

```
=== Final Recommendations ===
1. Deploy the Decision Tree for churn prediction (selected for best recall and balanced performance).
2. Focus retention efforts on customers with >70% predicted churn probability, as identified by the model.
3. Consider monitoring L1 and L2 logistic regression models as additional benchmarks for detecting high-risk c
ustomers.
4. Monitor model performance monthly and retrain as needed to maintain predictive accuracy.
5. Use feature importance insights (from Decision Tree) to improve customer experience and reduce churn risk.
```

In [ ]: