PHASE 3 PROJECT: SYRIATEL CUSTOMER CHURN

Business Understanding

Data Loading and Initial Inspection

Data Cleaning

Exploratory Data Analysis

Prepare for Data Modelling

Build and Evaluate Models

Model Comparison

Feature Importance(for best model if tree based)

Business Impact

Recommendations

1. BUSINESS UNDERSTANDING

""" Business Problem: SyriaTel is a telecommunications company facing customer churn issues. Churn refers to customers leaving the service, which directly impacts revenue.

Stakeholder: SyriaTel Retention Department Business Goal: Identify customers likely to churn 30-60 days in advance Success Metric: Reduce churn rate by 15% within 6 months """

2. DATA LOADING AND INITIAL INSPECTION

```
!pip install kagglehub
```

```
Requirement already satisfied: kagglehub in c:\users\user\anaconda3\lib\site-pac
Requirement already satisfied: packaging in c:\users\user\anaconda3\lib\site-pac
Requirement already satisfied: pyyaml in c:\users\user\anaconda3\lib\site-packag
Requirement already satisfied: requests in c:\users\user\anaconda3\lib\site-pack
Requirement already satisfied: tqdm in c:\users\user\anaconda3\lib\site-packages
Requirement already satisfied: charset_normalizer<4,>=2 in c:\users\user\anacond
Requirement already satisfied: idna<4,>=2.5 in c:\users\user\anaconda3\lib\site-
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\user\anaconda3\lib
```

```
Requirement already satisfied: certifi>=2017.4.17 in c:\users\user\anaconda3\lib
Requirement already satisfied: colorama in c:\users\user\anaconda3\lib\site-pack
```

```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("becksddf/churn-in-telecoms-dataset")

print("Path to dataset files:", path)
```

```
Warning: Looks like you're using an outdated `kagglehub` version (installed: 0.3
Path to dataset files: C:\Users\USER\.cache\kagglehub\datasets\becksddf\churn-in
```

```
# Import 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, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_sc
from imblearn.over_sampling import SMOTE
import warnings
```

```
#Load the data
df = pd.read_csv('syriatel_churn.csv')
print(f"Dataset shape: {df.shape}")
```

```
Dataset shape: (3333, 21)
```

```
df.head()
```

	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	382-4657	no	yes	25	265.1	110
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 rows × 21 columns

```
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
```

```
df.shape
```

```
(3333, 21)
```

```
df.describe()
```

	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	3333.00000
mean	101.064806	437.182418	8.099010	179.775098	100.435644	30.56230
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.25943
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.00000
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.43000
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.50000
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.79000
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.64000

3. DATA CLEANING

```
# Make a copy of the original dataframe for preprocessing
df_clean = df.copy()
```

```
# Remove phone number (unique identifier)
if 'phone number' in df_clean.columns:
    df_clean = df_clean.drop('phone number', axis=1)
    print("✓ Dropped 'phone number' column")
```

```
✓ Dropped 'phone number' column
```

```
# Handle missing values
missing_before = df_clean.isnull().sum().sum()
if missing_before > 0:
    print(f"\nHandling {missing_before} missing values...")
    # Fill numeric with median
    for col in df_clean.select_dtypes(include=[np.number]).columns:
        if df_clean[col].isnull().sum() > 0:
            df_clean[col] = df_clean[col].fillna(df_clean[col].median())

             # Fill categorical with mode
    for col in df_clean.select_dtypes(include=['object']).columns:
        if df_clean[col].isnull().sum() > 0:
```

```
            df_clean[col] = df_clean[col].fillna(df_clean[col].mode()[0])

    print("✓ Missing values filled")
```

```
# Check for duplicates
duplicates = df_clean.duplicated().sum()
if duplicates > 0:
    print(f"\nRemoving {duplicates} duplicate rows...")
    df_clean = df_clean.drop_duplicates()
    print("✓ Duplicates removed")

print(f"\nFinal dataset shape: {df_clean.shape}")
```

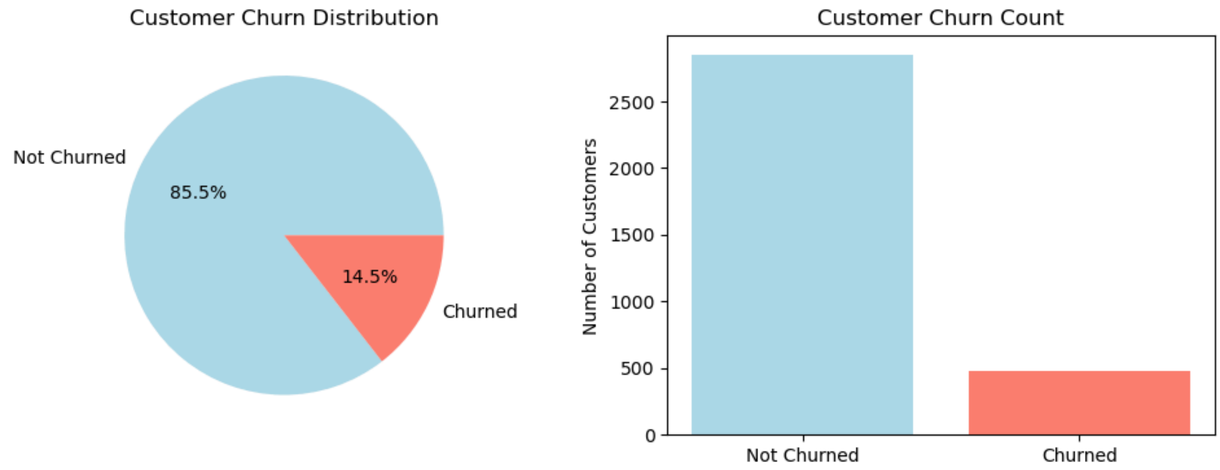
```
Final dataset shape: (3333, 20)
```

4. EXPLORATORY DATA ANALYSIS

```
# First, get the churn counts from cleaned data
churn_counts = df_clean['churn'].value_counts()
print(f"Churn distribution in cleaned data: {dict(churn_counts)}")
```

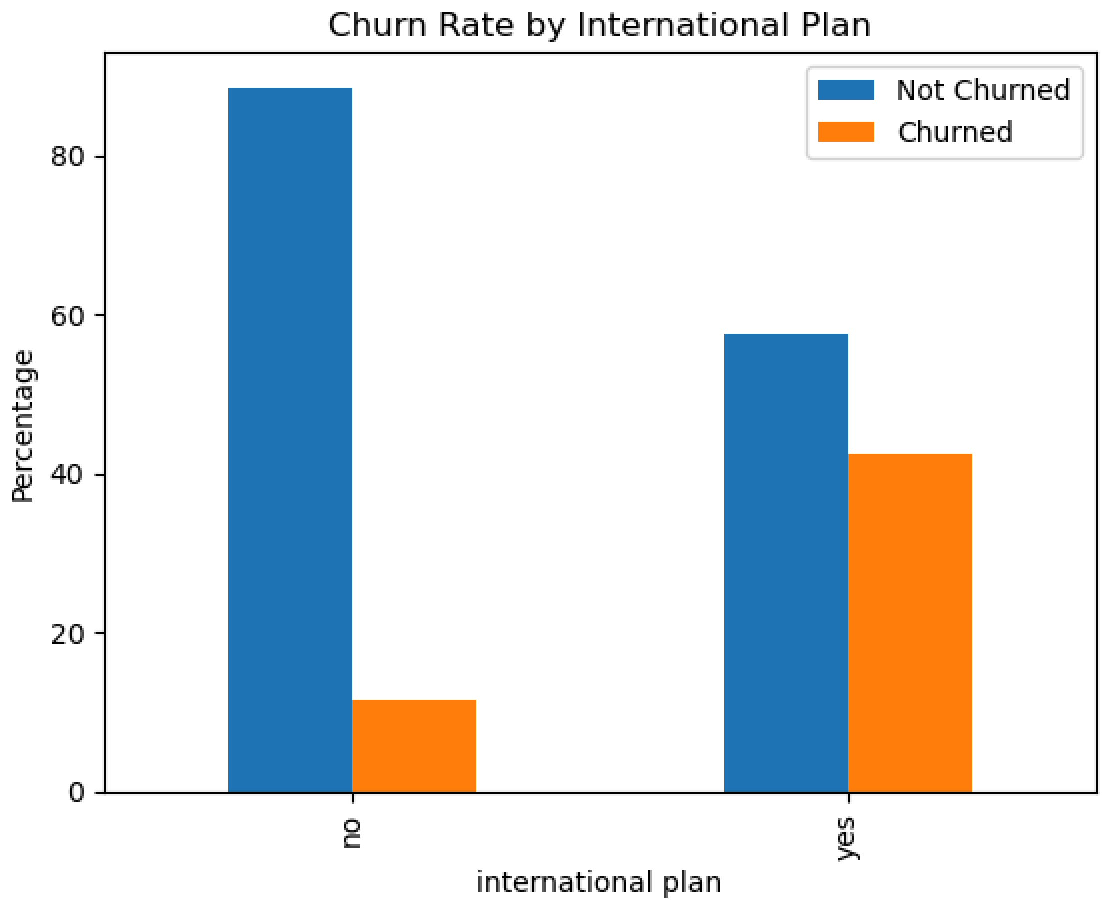
```
Churn distribution in cleaned data: {False: 2850, True: 483}
```

```
# Visual 1: Churn distribution
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.pie(churn_counts, labels=['Not Churned', 'Churned'],
        autopct='%1.1f%%', colors=['lightblue', 'salmon'])
plt.title('Customer Churn Distribution')

plt.subplot(1, 2, 2)
plt.bar(['Not Churned', 'Churned'], churn_counts, color=['lightblue', 'salmon']
plt.title('Customer Churn Count')
plt.ylabel('Number of Customers')
plt.show()
```

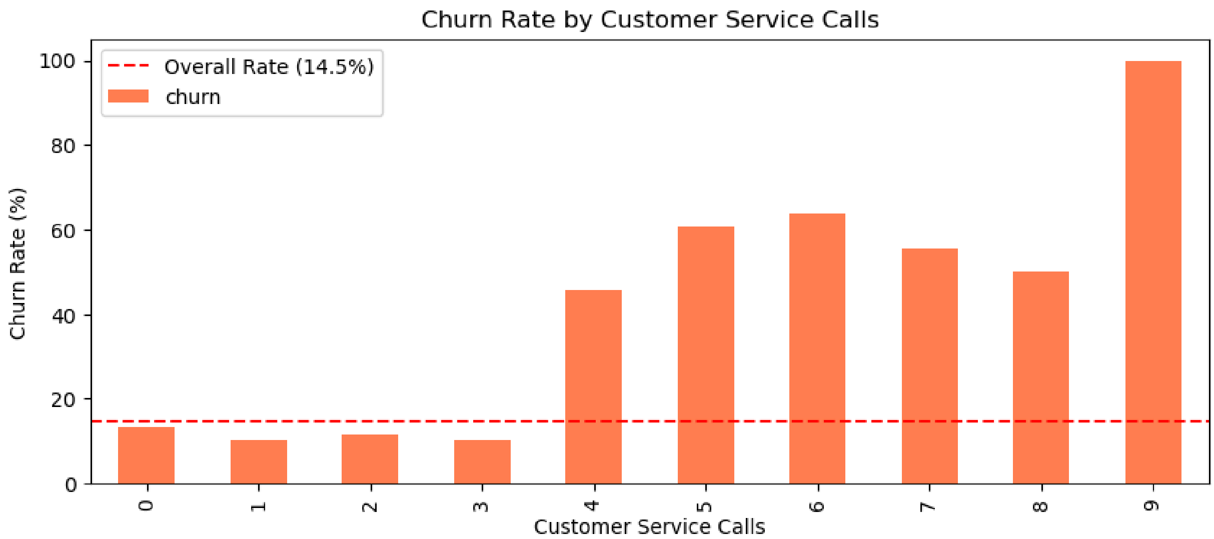



```
# Visual 2: Churn by international plan
if 'international plan' in df_clean.columns:
    plt.figure(figsize=(8, 4))
    churn_by_plan = pd.crosstab(df_clean['international plan'], df_clean['churr
    churn_by_plan.plot(kind='bar')
    plt.title('Churn Rate by International Plan')
    plt.ylabel('Percentage')
    plt.legend(['Not Churned', 'Churned'])
    plt.show()
```

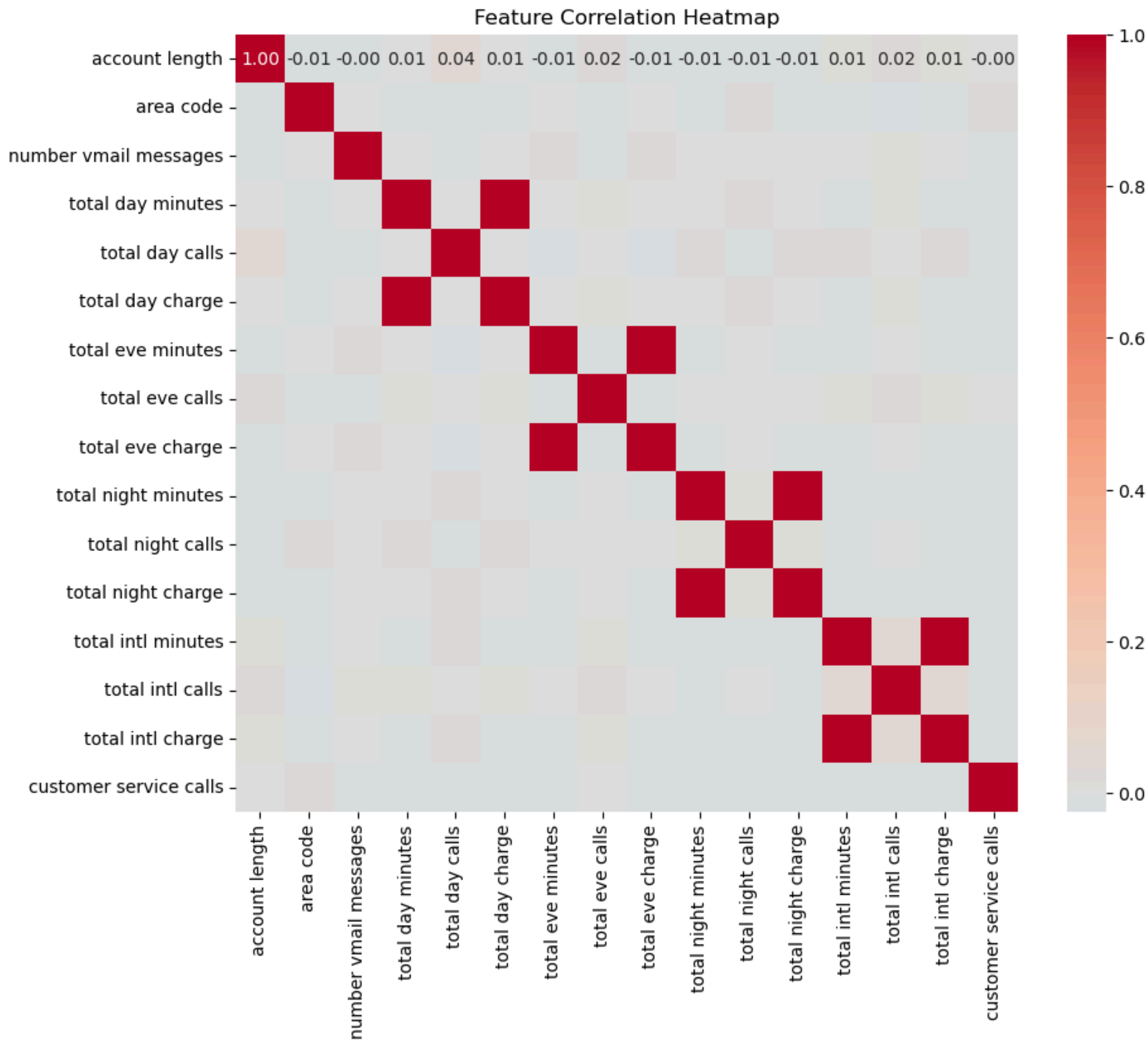
```
<Figure size 800x400 with 0 Axes>
```




```
# Visual 3: Churn by customer service calls
if 'customer service calls' in df_clean.columns:
    plt.figure(figsize=(10, 4))
    service_calls_churn = df_clean.groupby('customer service calls')['churn'].m
    service_calls_churn.plot(kind='bar', color='coral')
    plt.title('Churn Rate by Customer Service Calls')
    plt.xlabel('Customer Service Calls')
    plt.ylabel('Churn Rate (%)')
    plt.axhline(y=df_clean['churn'].mean() * 100, color='red', linestyle='--',
                label=f'Overall Rate ({df_clean["churn"].mean()*100:.1f}%)')
    plt.legend()
    plt.show()
```




```
# Visual 4: Correlation heatmap
plt.figure(figsize=(10, 8))
numeric_df = df_clean.select_dtypes(include=[np.number])
sns.heatmap(numeric_df.corr(), annot=True, cmap='coolwarm', center=0, fmt='.2f'
plt.title('Feature Correlation Heatmap')
plt.show()
```



5. PREPARE DATA FOR MODELING

```
# Create copy for modeling
df_model = df_clean.copy()
```

```
# Encode categorical variables
cat_cols = df_model.select_dtypes(include=['object']).columns
for col in cat_cols:
    if df_model[col].nunique() == 2:
        # Binary encoding
        df_model[col] = df_model[col].map({'yes':1, 'no':0, 'Yes':1, 'No':0})
        print(f"✓ Binary encoded '{col}'")
    else:
        # One-hot encoding
        dummies = pd.get_dummies(df_model[col], prefix=col, drop_first=True)
        df_model = pd.concat([df_model.drop(col, axis=1), dummies], axis=1)
        print(f"✓ One-hot encoded '{col}'")
```

```
✓ One-hot encoded 'state'
✓ Binary encoded 'international plan'
✓ Binary encoded 'voice mail plan'
```

```
# Features and target
X = df_model.drop('churn', axis=1)
y = df_model['churn']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random

# SMOTE to balance classes
smote = SMOTE(random_state=42)
X_train_bal, y_train_bal = smote.fit_resample(X_train, y_train)
print(f"\nBefore SMOTE: {np.bincount(y_train)}")
print(f"After SMOTE:  {np.bincount(y_train_bal)}")
```

```

Before SMOTE: [2280  386]
After SMOTE:  [2280 2280]
```

```
# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_bal)
X_test_scaled = scaler.transform(X_test)
print(f"\nScaled training data: {X_train_scaled.shape}")
print("✓ Data preparation complete")
```

```

Scaled training data: (4560, 68)
✓ Data preparation complete
```

6. BUILD AND EVALUATE MODELS

```
#  evaluation function (no external accuracy_score needed)
def evaluate_model(model, X_train, X_test, y_train, y_test, model_name):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    # Compute accuracy directly
    acc = (y_pred == y_test).mean()
    recall = recall_score(y_test, y_pred)

    # AUC if available
    auc = None
    if hasattr(model, 'predict_proba'):
        y_proba = model.predict_proba(X_test)[:, 1]
        auc = roc_auc_score(y_test, y_proba)
    print(f"\n{model_name}")
    print("-" * 30)
    print(f"Test Accuracy:  {acc:.4f}")
    print(f"Test Recall:    {recall:.4f}")
    if auc:
        print(f"Test AUC:       {auc:.4f}")
    print("\nClassification Report:")
    print(classification_report(y_test, y_pred, target_names=['Not Churned', 'C

    return {'model': model, 'accuracy': acc, 'recall': recall, 'auc': auc, 'y_p
```

```
# ---------- Model 1: Logistic Regression (baseline) ----------
lr = LogisticRegression(random_state=42, max_iter=1000)
lr_res = evaluate_model(lr, X_train_scaled, X_test_scaled, y_train_bal, y_test,

# ---------- Model 2: Logistic Regression (tuned) ----------
lr_tuned = LogisticRegression(random_state=42, max_iter=1000, class_weight='bal
lr_tuned_res = evaluate_model(lr_tuned, X_train_scaled, X_test_scaled, y_train_

# ---------- Model 3: Random Forest ----------
rf = RandomForestClassifier(random_state=42, n_estimators=100, class_weight='ba
rf_res = evaluate_model(rf, X_train_scaled, X_test_scaled, y_train_bal, y_test,

# ---------- Model 4: Decision Tree (simple) ----------
dt = DecisionTreeClassifier(random_state=42, max_depth=5)
dt_res = evaluate_model(dt, X_train_scaled, X_test_scaled, y_train_bal, y_test,
# Collect results
results = {
    'Logistic Regression (baseline)': lr_res,
    'Logistic Regression (tuned)': lr_tuned_res,
    'Random Forest': rf_res,
```

```
    'Decision Tree': dt_res
}
```

```
1. Logistic Regression (baseline)
------------------------------
Test Accuracy:  0.8576
Test Recall:    0.2887
Test AUC:       0.7898

Classification Report:
              precision    recall  f1-score   support

 Not Churned       0.89      0.95      0.92       570
     Churned       0.52      0.29      0.37        97

    accuracy                           0.86       667
   macro avg       0.70      0.62      0.65       667
weighted avg       0.83      0.86      0.84       667


2. Logistic Regression (tuned)
------------------------------
Test Accuracy:  0.8576
Test Recall:    0.3196
Test AUC:       0.7932

Classification Report:
              precision    recall  f1-score   support

 Not Churned       0.89      0.95      0.92       570
     Churned       0.52      0.32      0.39        97

    accuracy                           0.86       667
   macro avg       0.70      0.63      0.66       667
weighted avg       0.84      0.86      0.84       667


3. Random Forest
------------------------------
Test Accuracy:  0.8936
Test Recall:    0.5876
Test AUC:       0.8645

Classification Report:
              precision    recall  f1-score   support

 Not Churned       0.93      0.95      0.94       570
     Churned       0.65      0.59      0.62        97

    accuracy                           0.89       667
   macro avg       0.79      0.77      0.78       667
weighted avg       0.89      0.89      0.89       667


4. Decision Tree
```

```
------------------------------
Test Accuracy:  0.8621
Test Recall:    0.6392
Test AUC:       0.7791
```

8. MODEL COMPARISON

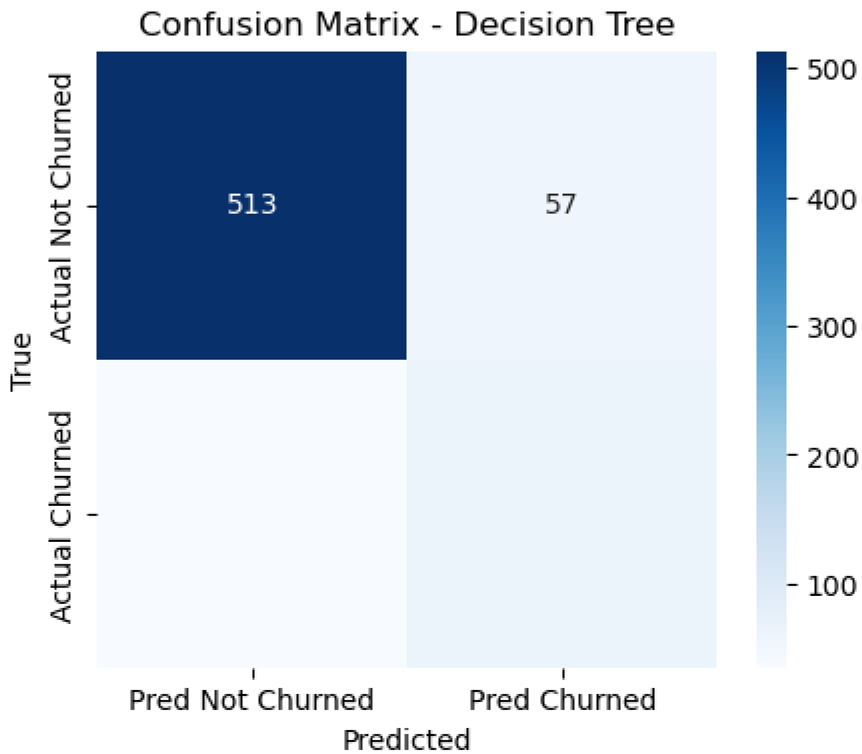
```
comp_df = pd.DataFrame([
    {'Model': name, 'Accuracy': res['accuracy'], 'Recall': res['recall'], 'AUC'
    for name, res in results.items()
]).sort_values('Recall', ascending=False)

print(comp_df.to_string(index=False))
```

```
                        Model  Accuracy   Recall      AUC
                Decision Tree  0.862069 0.639175 0.779146
                Random Forest  0.893553 0.587629 0.864478
   Logistic Regression (tuned) 0.857571 0.319588 0.793163
Logistic Regression (baseline) 0.857571 0.288660 0.789799
```

```
# Select best model based on recall
best_idx = comp_df['Recall'].idxmax()
best_model_name = comp_df.loc[best_idx, 'Model']
best_model = results[best_model_name]['model']
best_y_pred = results[best_model_name]['y_pred']
print(f"\n✅ Best model: {best_model_name} (Recall = {comp_df.loc[best_idx, 'R
```

```

✅ Best model: Decision Tree (Recall = 0.6392)
```

```
# Show confusion matrix only for the best model
cm = confusion_matrix(y_test, best_y_pred)
plt.figure(figsize=(5,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Pred Not Churned', 'Pred Churned'],
            yticklabels=['Actual Not Churned', 'Actual Churned'])
plt.title(f'Confusion Matrix - {best_model_name}')
plt.ylabel('True')
plt.xlabel('Predicted')
plt.show()
```



9. FEATURE IMPORTANCE (for best model if tree-based)

```
if hasattr(best_model, 'feature_importances_'):
    imp = best_model.feature_importances_
    feat_names = X.columns
    imp_df = pd.DataFrame({'feature': feat_names, 'importance': imp}).sort_valu
    print("\nTop 10 important features:")
    print(imp_df.head(10).to_string(index=False))

    # Plot
    plt.figure(figsize=(10,5))
    top10 = imp_df.head(10)
    plt.barh(range(len(top10)), top10['importance'])
    plt.yticks(range(len(top10)), top10['feature'])
    plt.xlabel('Importance')
    plt.title(f'Top 10 Feature Importances ({best_model_name})')
    plt.gca().invert_yaxis()
    plt.tight_layout()
    plt.show()
else:
    print("Feature importance not available for this model.")
```

```
Top 10 important features:
                feature  importance
         total day charge    0.284936
   customer service calls    0.261715
           voice mail plan    0.103382
         total eve charge    0.075338
         total intl calls    0.067741
        total day minutes    0.059565
       international plan    0.058011
        total intl charge    0.052572
       total night charge    0.010891
                 state_WY    0.006641
```