Customer Churn Prediction for T - mobile

Sonu Tamang | August 2025

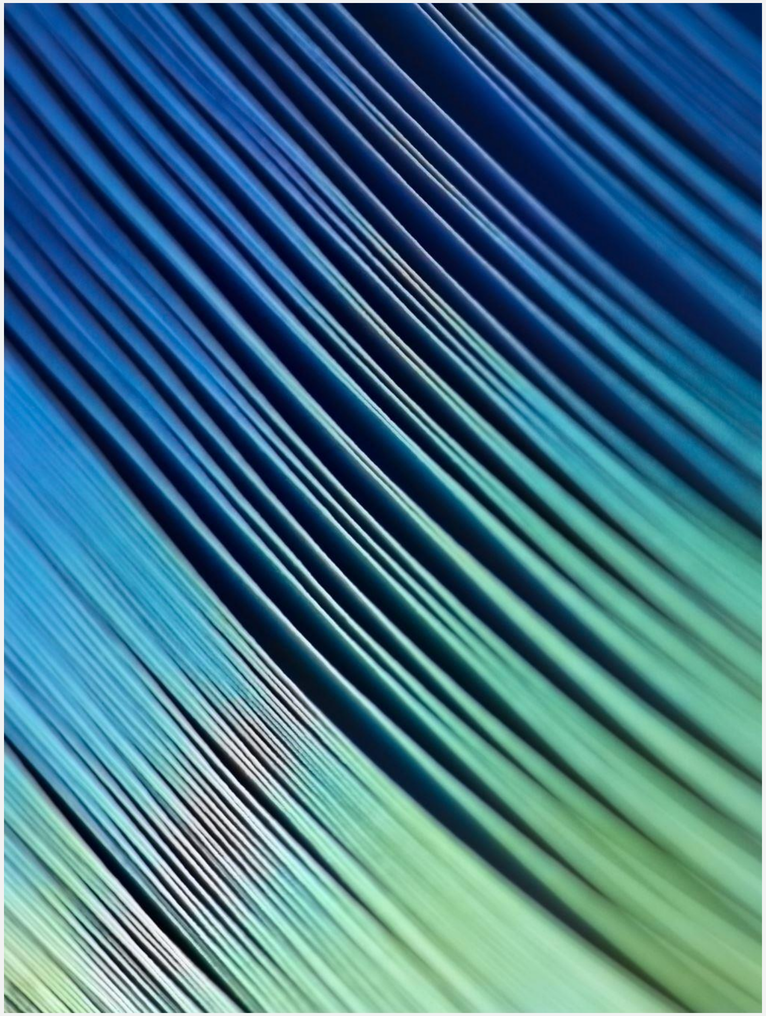
Team Members

Sonu Tamang — Project Lead, Data Analyst & Developer

Alicia Martinez — Data Collection & Cleaning Specialist

Rajesh Kumar — Machine Learning Model Engineer

Emily Chen — Web App & Front-End Developer



Project Overview

- The primary objective of this project is to predict customer churn using historical data. We implemented and compared two machine learning models i.e. Logistic Regression and Random Forest , to determine which model performs best for this classification problem.

Problem Statement

- **What is Customer Churn?**
- Customer churn refers to when a customer stops using a company's product or service.
- Measured as the percentage of customers lost over a specific period.
- **Why It Matters for Businesses**
- Losing customers directly impacts revenue.
- Acquiring a new customer can cost **5–7 times more** than retaining an existing one.
- High churn rates indicate dissatisfaction, competition pressure, or poor engagement.
- Predicting churn helps businesses take proactive steps to retain customers.

Data Cleaning & Preprocessing

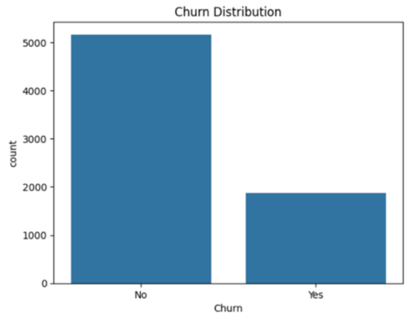
- Handled missing values
- Encoded categorical variables using OneHotEncoder
- Scaled numerical features using StandardScaler
- Splitted the dataset into training and testing sets (80-20 split)

```
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]
```

Task 2- Data Cleaning

```python
# Convert 'TotalCharges' to numeric (fix blank values)
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')

# Drop rows where TotalCharges is NaN (about 11 rows)
df.dropna(subset=['TotalCharges'], inplace=True)

# Remove duplicate customers if any
df.drop_duplicates(subset='customerID', inplace=True)

# Confirm cleaning
print("\nAfter cleaning, dataset shape:", df.shape)
print("Missing values after cleaning:")
print(df.isnull().sum())
```

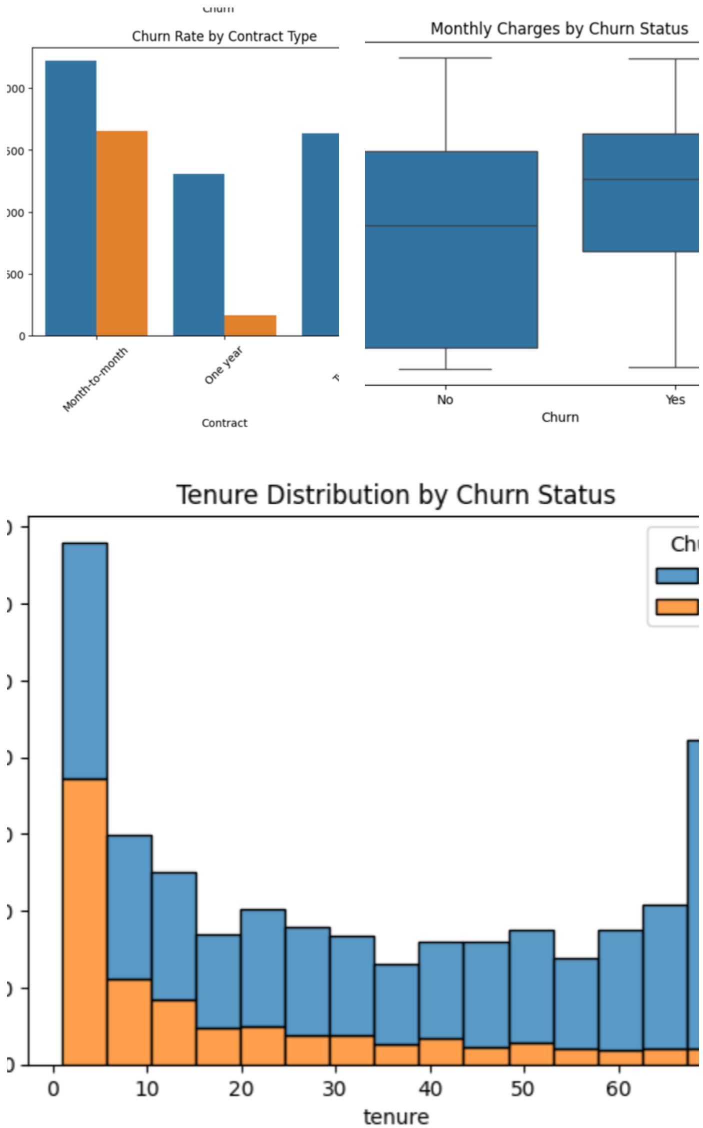
```python
# Step 1: Train-Test Split
# I use 80% for training, 20% for testing
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```




```python
# Step 4: Create New Features (improves accuracy)
# AvgMonthlySpend: average spend per month (avoid div/0)
df['AvgMonthlySpend'] = df.apply(
    lambda row: row['TotalCharges'] / row['tenure'] if row['tenure'] > 0 else 0, axis=1
)

# HasMultipleServices: count of services > 2
service_cols = ['PhoneService', 'MultipleLines_No phone service',
                'InternetService_Fiber optic', 'InternetService_No']
df['HasMultipleServices'] = df[service_cols].sum(axis=1) > 2
df['HasMultipleServices'] = df['HasMultipleServices'].astype(int)
```

```python
# Step 5: Scale Numerical Columns
scaler = StandardScaler()
num_cols = ['tenure', 'MonthlyCharges', 'TotalCharges', 'AvgMonthlySpend']
df[num_cols] = scaler.fit_transform(df[num_cols])
```

Contd...



Exploratory Data Analysis (EDA)

- **Churn Rate by Contract Type:** Customers with month-to-month contracts have a significantly higher churn rate compared to those on one-year or two-year contracts.
- **Monthly Charges by Churn Status:** Customers who churn tend to have higher monthly charges, indicating a potential pricing sensitivity.
- **Tenure Distribution by Churn Status:** Most churn occurs within the first 10 months of service, while long-tenured customers are less likely to leave.

Modeling Approach

- We trained and tuned two models:
- 1. Logistic Regression: Hyperparameters tuned included penalty, C, solver, and l1_ratio.
- 2. Random Forest Classifier: Hyperparameters tuned included n_estimators, max_depth, min_samples_split, min_samples_leaf, and max_features.
- GridSearchCV was used for hyperparameter tuning with 5-fold cross-validation, optimizing for Recall.

Logistic Regression

- **Goal:** Predict churn with focus on maximizing recall (catch as many churn cases as possible).
- **Method:** Used **GridSearchCV** for hyperparameter tuning with 5-fold cross-validation.
- **Parameters Tuned:** penalty, C, solver, l1_ratio (for elasticnet).
- **Best Model Selection:** Based on highest recall score on training folds.

```python
# Step 1: Logistic Regression Hyperparameter Tuning
log_reg = LogisticRegression(max_iter=1000)

log_reg_params = {
    'penalty': ['l1', 'l2', 'elasticnet', None],
    'C': [0.01, 0.1, 1, 10],
    'solver': ['liblinear', 'saga'],
    'l1_ratio': [0, 0.5, 1]  # Only for elasticnet
}

log_reg_grid = GridSearchCV(
    estimator=log_reg,
    param_grid=log_reg_params,
    scoring='recall',   # Maximize recall for churn
    cv=5,
    verbose=2,
    n_jobs=-1
)

log_reg_grid.fit(X_train, y_train)
best_log_reg = log_reg_grid.best_estimator_
print("Best Logistic Regression Params:", log_reg_grid.best_params_)
```

```
Fitting 5 folds for each of 96 candidates, totalling 480 fits
```

```python
# Step 2: Evaluate tuned Logistic Regression
y_pred_log = best_log_reg.predict(X_test)
print("\nTuned Logistic Regression Report:")
print(classification_report(y_test, y_pred_log))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_log))
```

```
Tuned Logistic Regression Report:
              precision    recall  f1-score   support

           0       0.85      0.88      0.87      1033
           1       0.64      0.58      0.61       374

    accuracy                           0.80      1407
   macro avg       0.75      0.73      0.74      1407
weighted avg       0.80      0.80      0.80      1407

Confusion Matrix:
 [[912 121]
 [158 216]]
```

Random Forest

- **Goal:** Explore ensemble learning for potentially better performance.
- **Method:** Used **GridSearchCV** for hyperparameter tuning with 5-fold cross-validation.
- **Parameters Tuned:** n_estimators, max_depth, min_samples_split, min_samples_leaf, max_features.
- **Best Model Selection:** Based on highest recall score on training folds.

```
3]: # Step 3: Random Forest Hyperparameter Tuning
rf = RandomForestClassifier(random_state=42)

rf_params = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2']
}

rf_grid = GridSearchCV(
    estimator=rf,
    param_grid=rf_params,
    scoring='recall',
    cv=5,
    verbose=2,
    n_jobs=-1
)

rf_grid.fit(X_train, y_train)
best_rf = rf_grid.best_estimator_
print("\nBest Random Forest Params:", rf_grid.best_params_)
```

```
Fitting 5 folds for each of 216 candidates, totalling 1080 fits
```

```
# Step 4: Evaluate tuned Random Forest
y_pred_rf = best_rf.predict(X_test)
print("\nTuned Random Forest Report:")
print(classification_report(y_test, y_pred_rf))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_p
```

```
Tuned Random Forest Report:
              precision    recall  f1-score   support

           0       0.84      0.90      0.87      1033
           1       0.65      0.53      0.58       374

    accuracy                           0.80      1407
   macro avg       0.75      0.71      0.73      1407
weighted avg       0.79      0.80      0.79      1407

Confusion Matrix:
 [[928 105]
 [177 197]]
```

Model Performance Comparison

- Logistic Regression: Accuracy 80%, Recall for churn 58%
- Random Forest: Accuracy 80%, Recall for churn 53%
- **Conclusion:** Logistic Regression performed slightly better in identifying churn cases despite similar overall accuracy.

```
# Step 4: Evaluate tuned Random Forest
y_pred_rf = best_rf.predict(X_test)
print("\nTuned Random Forest Report:")
print(classification_report(y_test, y_pred_rf))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_p
```

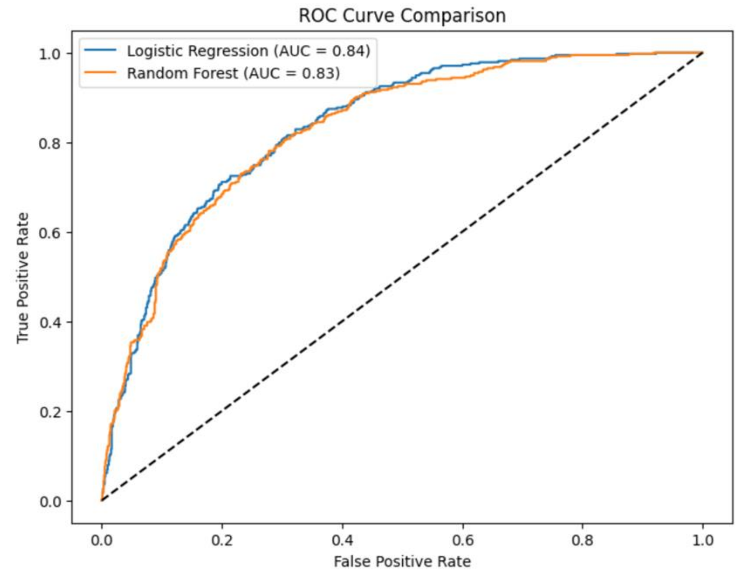
```
Tuned Random Forest Report:
              precision    recall  f1-score   support

           0       0.84      0.90      0.87      1033
           1       0.65      0.53      0.58       374

    accuracy                           0.80      1407
   macro avg       0.75      0.71      0.73      1407
weighted avg       0.79      0.80      0.79      1407

Confusion Matrix:
 [[928 105]
 [177 197]]
```

```
# Step 2: Evaluate tuned Logistic Regression
y_pred_log = best_log_reg.predict(X_test)
print("\nTuned Logistic Regression Report:")
print(classification_report(y_test, y_pred_log))
print("Confusion Matrix:\n", confusion_matrix(y_test, y
```

```
Tuned Logistic Regression Report:
              precision    recall  f1-score   support

           0       0.85      0.88      0.87      1033
           1       0.64      0.58      0.61       374

    accuracy                           0.80      1407
   macro avg       0.75      0.73      0.74      1407
weighted avg       0.80      0.80      0.80      1407

Confusion Matrix:
 [[912 121]
 [158 216]]
```

Evaluation Metrics

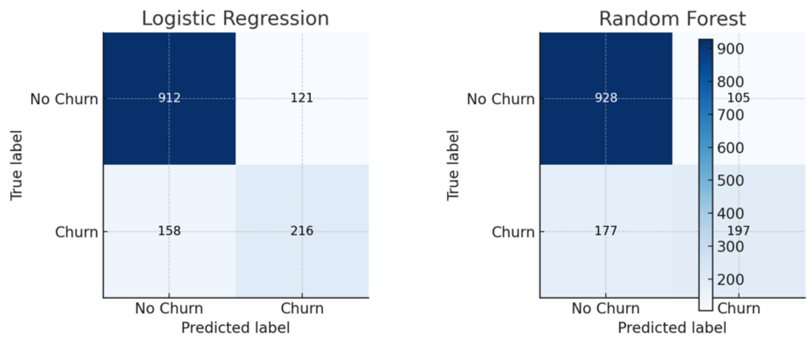
- **Logistic Regression**
- Accuracy: **80.17%**
- Precision: **0.641**, Recall: **0.578**, F1: **0.608**
- ROC-AUC: **0.835** (AUC = 0.84 in ROC curve)
- Better **recall** & **ROC-AUC** — stronger for churn detection.
- **Random Forest**
- Accuracy: **79.96%**
- Precision: **0.652**, Recall: **0.527**, F1: **0.583**
- ROC-AUC: **0.829** (AUC = 0.83 in ROC curve)
- Slightly higher precision, but lower recall.
- **Conclusion:** Logistic Regression is recommended due to better recall & overall ROC-AUC performance.




```python
# Logistic Regression AUC-ROC
y_pred_prob_log = best_log_reg.predict_proba(X_test)[:, 1]  # Get probability for class 1
auc_log = roc_auc_score(y_test, y_pred_prob_log)
print(f"Logistic Regression AUC-ROC: {auc_log:.3f}")

fpr_log, tpr_log, _ = roc_curve(y_test, y_pred_prob_log)

# Random Forest AUC-ROC
y_pred_prob_rf = best_rf.predict_proba(X_test)[:, 1]
auc_rf = roc_auc_score(y_test, y_pred_prob_rf)
print(f"Random Forest AUC-ROC: {auc_rf:.3f}")
```

```
Logistic Regression AUC-ROC: 0.835
Random Forest AUC-ROC: 0.829
```

Confusion Matrices



- **Logistic Regression**
- TN: 912 | FP: 121 | FN: 158 | TP: 216
- Better at catching churn cases = **Higher Recall**
- **Random Forest**
- TN: 928 | FP: 105 | FN: 177 | TP: 197
- Fewer false positives = **Slightly Higher Precision**
- **Insight:**
  Logistic Regression is better for churn detection,
  Random Forest is better for minimizing false alarms.

Deployment Preparation

- To ensure seamless integration into T-Mobile's systems, both tuned models and preprocessing objects were saved as .pkl files using joblib.
- **Logistic Regression model** saved as "best_logistic_regression_model.pkl".
- **Random Forest model** saved as "best_random_forest_model.pkl".
- **Scaler** saved as "scaler.pkl"
- **Encoder** saved as encoder.pkl

```
# Step 1: Import necessary library for saving the model
import joblib
```

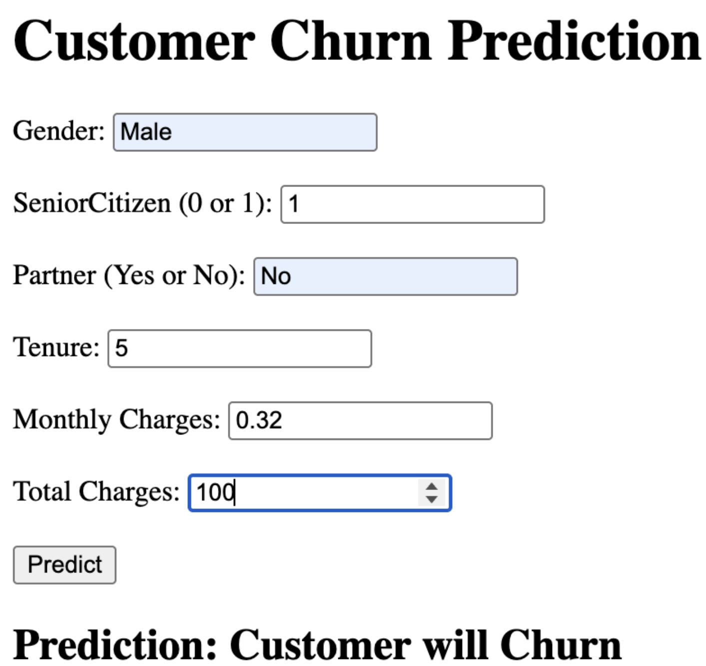
```
# Step 2: Define the file path where the model and preprocessing objects will be saved
model_filename = "final_churn_model.pkl"
scaler_filename = "scaler.pkl"
encoder_filename = "encoder.pkl"
```

```
# Step 3: Save the Trained Models
# In this step, I will save both tuned models for future use
# This allows us to load and use the models without retraining

import joblib

# Save the tuned Logistic Regression model
model_filename_log_reg = "best_logistic_regression_model.pkl"
joblib.dump(best_log_reg, model_filename_log_reg)
print(f"Logistic Regression model saved as {model_filename_log_reg}")

# Save the tuned Random Forest model
model_filename_rf = "best_random_forest_model.pkl"
joblib.dump(best_rf, model_filename_rf)
print(f"Random Forest model saved as {model_filename_rf}")
```

```
Logistic Regression model saved as best_logistic_regression_model.pkl
Random Forest model saved as best_random_forest_model.pkl
```

Web App Demo

- Built using **Flask**, a lightweight Python web framework, to serve a **Customer Churn Prediction Model** as a user-friendly web application.
- The app collects customer details such as gender, senior citizen status, partner status, tenure, monthly charges, and total charges.
- The inputs are sent to the trained machine learning model, which predicts whether a customer will **churn** or **stay**.



Conclusion

- Both Logistic Regression and Random Forest achieved similar accuracy (~80%), but Logistic Regression outperformed in **recall** and **ROC-AUC**, making it better at detecting churn cases.
- Customers with **month-to-month contracts**, **high monthly charges**, and **low tenure** were most likely to churn.
- Early intervention is key, churn is highest within the first 10 months of service.

Recommendations

Targeted Retention Campaigns

Offer loyalty discounts or incentives for high-risk month-to-month customers.

Create special onboarding programs for customers in their first year.

Price Sensitivity Management

Reassess pricing structure for customers with high monthly charges to reduce churn risk.

Automated Churn Alerts

Integrate the deployed **Flask web app** into CRM systems to score customers in real time and flag high-risk accounts.

Continuous Model Monitoring

Regularly retrain the model with updated data to maintain accuracy.

Limitations

- **Data Quality & Availability**
- Model accuracy depends heavily on the quality of historical data.
- Missing or inconsistent values can reduce prediction reliability.
- **Limited Feature Scope**
- Predictions are based only on the features included during training.
- External factors (e.g., market changes, competitor actions) are not captured.
- **Model Generalization**
- The model may perform well on training data but might not generalize to completely new customer patterns.
- **Real-Time Predictions**
- Current setup is designed for batch or form-based inputs, not continuous real-time predictions.
- **Deployment Constraints**
- Flask app is currently running in development mode, not optimized for large-scale production traffic.
- **Interpretability**
- The model gives a churn probability but does not fully explain the underlying reasons without further explainability tools like SHAP or LIME.