Phase-3 Submission

Student Name : R.Bhuvana

RegisterNumber : 422723106007

Institution : V.R.S.College of Engineering and Technology

Department : ECE

Date of Submission : 17:05:2025

Github Repository Link: https://github.com/bhuvana-2005ECE/Customer-churn-Predicting-using-machine-learning

---

1.Problem Statement

Customer churn poses a significant challenge for businesses, as losing existing customers can be more costly than acquiring new ones. This project focuses on leveraging machine learning techniques to predict churn by analyzing customer demographics, usage behavior, and interaction history. The objective is to discover subtle patterns and predictors that are not immediately apparent through conventional analysis, enabling data-driven strategies to retain at-risk customers and enhance business performance.

2.Abstract

This project focuses on predicting customer churn for a telecommunications company using machine learning techniques. The primary objective is to identify customers who are likely to churn, enabling the company to take proactive measures to retain them. The approach involves data preprocessing, exploratory data analysis (EDA), feature engineering, and model development. Multiple machine learning models, including Logistic Regression and Random Forest, were trained and evaluated for their ability to predict churn.

The Random Forest model outperformed the Logistic Regression model, achieving higher accuracy and F1-score. The project provides valuable insights into the key factors influencing churn, such as tenure, monthly charges, and contract type, and demonstrates how machine learning can be effectively applied to business problems like customer retention.

3.System Requirements

To successfully run this customer churn prediction project, the following system and software requirements are recommended:

Hardware Requirements:

- **Minimum RAM**: 8 GB (for smoother processing of large datasets and model training)
- **Processor**: Intel Core i5 (or equivalent) for basic processing; Intel Core i7 or higher is recommended if running large-scale computations and model training with large datasets.
- **Storage**: 10 GB of free disk space for storing data and model outputs.

Software Requirements:

- **Operating System**: Windows, macOS, or Linux (compatible with Python and libraries used in the project).

Python Version:

- **Python**: Version 3.6 or higher is required. Python 3.8 or higher is recommended for optimal compatibility with the libraries used.

Required Libraries:

The following Python libraries should be installed to run the project:

- **pandas** (for data manipulation)

- **numpy** (for numerical operations)
- **scikit-learn** (for machine learning models, preprocessing, and evaluation)
- **matplotlib** (for basic plotting)
- **seaborn** (for advanced data visualization)
- **xgboost** (optional, for additional model testing)
- **plotly** (optional, for interactive visualizations)

IDE / Notebook:

- **Google Colab** (cloud-based, no installation required, offers free access to GPUs/TPUs for faster model training).
- **Jupyter Notebook** (local setup with Python for interactive development).
- **VSCode** (for code development, though Jupyter Notebooks are often more suited for data science projects).

For Jupyter Notebook, it is recommended to install the **JupyterLab** environment for an improved interface.

bash

4. Objectives

The main objective of this project is to predict customer churn using machine learning, providing businesses with actionable insights to improve customer retention. The key goals are:

1. **Churn Prediction**: Build a model to predict whether a customer is likely to churn (leave the service) based on historical data.
2. **Identify Key Churn Factors**: Discover which factors (e.g., contract type, tenure, monthly charges) most influence churn, helping businesses focus on critical areas for retention.
3. **Model Evaluation**: Compare various machine learning models to select the one with the best performance in terms of accuracy, precision, recall, and F1-score.

4. **Proactive Retention Strategies**: Use churn predictions to trigger automated interventions (e.g., discounts, targeted offers) for high-risk customers, improving retention rates.

Business Impact:

- **Reduced Customer Churn**: By predicting churn early, businesses can proactively intervene, reducing the number of customers who leave, thus enhancing retention.
- **Optimized Resource Allocation**: Focus marketing and customer service resources on at-risk customers, rather than wasting resources on customers who are unlikely to churn.
- **Improved Customer Satisfaction**: By addressing the factors that cause churn (e.g., pricing, service quality), businesses can create a more positive customer experience, improving loyalty.
- **Increased Revenue**: Retaining customers is more cost-effective than acquiring new ones. By reducing churn, businesses can increase customer lifetime value, leading to higher revenue over time.

5.Flowchart of the Project Workflow

Start

↓

Problem Definition

↓

Data Collection

↓

Data Preprocessing

• Handling Missing Values

• Encoding Categorical Variables

• Feature Scaling

↓

Exploratory Data Analysis (EDA)

• Data Visualization

• Correlation Analysis

↓

Feature Selection & Engineering

↓

Model Selection

• Try different algorithms (e.g., Logistic Regression, Decision Trees, Random Forest, XGBoost)

↓

Model Training

↓

Model Evaluation

• Accuracy, Precision, Recall, F1 Score, ROC-AUC

↓

Model Tuning (Hyperparameter Optimization)

↓

Final Model Deployment

↓

Prediction & Monitoring

↓

Business Insights & Recommendations

↓

End

6. Data Description

The dataset used for this project is the Telco Customer Churn dataset, which is publicly available on Kaggle. It contains customer-level information from a telecom company and is widely used for churn prediction modeling in supervised learning tasks.

- **Dataset Name and Origin**:
  Telco Customer Churn Dataset – Sourced from Kaggle
- **Type of Data**:
  Structured tabular data
- **Number of Records and Features**:
  The dataset consists of **7,043 records (rows)** and **21 features (columns)**, including both numerical and categorical attributes.
- **Static or Dynamic Dataset**:
  The dataset is **static**, meaning it was collected at a single point in time and does not change or update in real time.
- **Target Variable (if supervised learning)**:
  The target variable is **Churn**, which indicates whether a customer has left the service (Yes) or remains (No).

7.Data Preprocessing

Effective data preprocessing is essential to ensure high model performance and accuracy. The following steps were carried out to clean and prepare the Telco Customer Churn dataset

1. Handling Missing Values

- The TotalCharges column had some missing or blank values stored as strings.

- These were first converted to numeric using pd.to_numeric(errors='coerce'), which converts non-numeric entries to NaN.
- Missing values were then imputed using the **median** of the TotalCharges column, as it is less sensitive to outliers than the mean.

1. Handling Missing Values

- The TotalCharges column had some missing or blank values stored as strings.
- These were first converted to numeric using pd.to_numeric(errors='coerce'), which converts non-numeric entries to NaN.
- Missing values were then imputed using the **median** of the TotalCharges column, as it is less sensitive to outliers than the mean.

Code

```
data['TotalCharges'] = pd.to_numeric(data['TotalCharges'], errors='coerce')
data['TotalCharges'].fillna(data['TotalCharges'].median(), inplace=True)
```

2. Removing or Justifying Duplicate Records

- Checked for duplicate records using data.duplicated().sum()
- Result: 0 duplicate records were found, so no removal was needed.

3. Detecting and Treating Outliers

- Numerical columns such as MonthlyCharges and TotalCharges were visualized using boxplots.
- Z-score and IQR methods were used to detect outliers.

- Outliers were not removed to preserve the natural variability of the customer base, unless extreme values were clearly data entry errors (none were found in this case).

Code

```
from scipy import stats
z_scores = np.abs(stats.zscore(data[['MonthlyCharges', 'TotalCharges']]))
data = data[(z_scores < 3).all(axis=1)]  # Optional: only if extreme outliers found
```

4. Converting Data Types and Ensuring Consistency

- SeniorCitizen was originally numeric (0, 1), converted to categorical.
- TotalCharges converted from object to float.
- Ensured consistency of formats across all features.

Code

```
data['SeniorCitizen'] = data['SeniorCitizen'].replace({1: 'Yes', 0: 'No'})
data['SeniorCitizen'] = data['SeniorCitizen'].astype('category')
```

5. Encoding Categorical Variables

- **Label Encoding** was used for binary categorical variables like Churn, Partner, Dependents.
- **One-Hot Encoding** was applied to features with more than two categories like InternetService, Contract, PaymentMethod.

Code

```
from sklearn.preprocessing import LabelEncoder
binary_cols = ['Churn', 'Partner', 'Dependents', 'PhoneService', 'PaperlessBilling']
le = LabelEncoder()
```

```
for col in binary_cols:
    data[col] = le.fit_transform(data[col])

data = pd.get_dummies(data, drop_first=True)  # One-hot encoding
```

6. Normalizing or Standardizing Features

- Numerical features such as MonthlyCharges, TotalCharges, and tenure were standardized using StandardScaler to improve model performance.

Python

Code

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
data[['MonthlyCharges', 'TotalCharges', 'tenure']] = scaler.fit_transform(
    data[['MonthlyCharges', 'TotalCharges', 'tenure']]
)
```

8.Exploratory Data Analysis (EDA)

EDA is essential for understanding the structure, relationships, and patterns within the data before model building. This section includes univariate, bivariate, and multivariate analysis, along with key insights.

1. Univariate Analysis

We explored the distribution of individual features using visualizations:

- **Numerical Features**:

  Histograms and boxplots were used to visualize tenure, MonthlyCharges, and TotalCharges.

Code

```
import seaborn as sns
```

```
import matplotlib.pyplot as plt

sns.histplot(data['tenure'], kde=True)
sns.boxplot(x=data['MonthlyCharges'])
```

Observations:

- tenure is right-skewed, indicating many newer customers.
- MonthlyCharges and TotalCharges show moderate spread with some outliers.

- **Categorical Features**:

  Countplots were used to examine distributions of Contract, PaymentMethod, InternetService, and Churn.

Code

```
sns.countplot(x='Contract', data=data)
sns.countplot(x='Churn', data=data)
```

Observations:

- Month-to-month contracts are most common.
- Around 26% of customers have churned.

2. Bivariate / Multivariate Analysis

- **Correlation Matrix** (for numerical features):

Code

```
corr = data[['tenure', 'MonthlyCharges', 'TotalCharges']].corr()
sns.heatmap(corr, annot=True, cmap='coolwarm')
```

Insights:

- Strong positive correlation between tenure and TotalCharges.
- Weak correlation between MonthlyCharges and TotalCharges.

-

- **Churn vs Categorical Features**:

Code

```
sns.countplot(x='Churn', hue='Contract', data=data)
sns.countplot(x='Churn', hue='InternetService', data=data)
```

Insights:

- Customers with month-to-month contracts have a significantly higher churn rate.
- Churn is more common among customers with fiber optic internet.
-

- **Pairplots and Scatterplots**:

Code

```
sns.pairplot(data[['tenure', 'MonthlyCharges', 'TotalCharges', 'Churn']], hue='Churn')
```

Insights:

- Customers with low tenure and high monthly charges are more likely to churn.

3. Insights Summary

- Contract type**,** tenure, and monthly charges are strong indicators of churn.
- Customers on month-to-month plans, with lower tenure, and higher monthly charges show a higher tendency to churn.
- Features such as PaperlessBilling, **InternetService**, and PaymentMethod also show noticeable patterns with respect to churn.
- These insights will guide feature selection and model interpretation.

9.Feature Engineering

Feature engineering involves creating, transforming, or selecting variables to improve model performance and capture underlying patterns in the data. Based on domain knowledge and EDA insights, several enhancements were made.

1. New Feature Creation

- **Tenure Groups**: Customers with lower tenure are more likely to churn. We created tenure bins to capture these patterns

Code

```
def tenure_group(tenure):
    if tenure <= 12:
        return '0-1 year'
    elif tenure <= 24:
        return '1-2 years'
    elif tenure <= 48:
        return '2-4 years'
    elif tenure <= 60:
        return '4-5 years'
    else:
        return '5+ years'

data['TenureGroup'] = data['tenure'].apply(tenure_group)
```

Justification: Categorizing tenure helps the model detect churn-prone groups without relying solely on continuous values.

2. Total Services Count

- Created a new feature representing the total number of services a customer is subscribed to (e.g., phone, internet, streaming).

Code

```
services = ['PhoneService', 'MultipleLines', 'OnlineSecurity',
'OnlineBackup',
        'DeviceProtection', 'TechSupport', 'StreamingTV',
'StreamingMovies']

data['TotalServices'] = data[services].apply(lambda row: sum(row ==
'Yes'), axis=1)
```

Justification: More services may indicate higher engagement and lower churn risk.

3. Binning Monthly Charges

- Monthly charges were binned into low, medium, and high categories to simplify nonlinear relationships.

Code

```
data['MonthlyChargesGroup'] = pd.cut(data['MonthlyCharges'],
    bins=[0, 35, 70, 120],
  labels=['Low', 'Medium', 'High'])
```

Justification: Categorizing continuous features can help tree-based models better identify thresholds.

4. Payment Security Flag

- Created a binary feature to flag potentially less secure payment methods.

Code

```
data['IsAutoPayment'] = data['PaymentMethod'].apply(lambda x: 1 if 'automatic' in x.lower() else 0)
```

Justification: Customers using auto-pay methods may be more loyal or less likely to churn.

5. Dimensionality Reduction (Optional)

- Performed PCA (Principal Component Analysis) for exploratory purposes on standardized numeric features. However, the dimensionality reduction was not retained, as interpretability and model accuracy were better with original features.

Code

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca_components = pca.fit_transform(scaled_numeric_data)
```

Justification: PCA helped explore feature redundancy but wasn't used in the final model to preserve explainability.

6. Dropped Features

- **CustomerID** was dropped as it is a unique identifier with no predictive value.
- **Unnamed: 0** (if present) was also removed as it's often an index artifact.

10.Model Building

In this section, we will build and compare multiple machine learning models to predict customer churn. We will explore the performance of at least two models, justify their selection, and evaluate them using relevant metrics.

1. Model Selection

We have selected two widely used classification algorithms for this project:

- **Logistic Regression**: A simple and interpretable model that is often effective for binary classification tasks like churn prediction.
- **Random Forest**: An ensemble model based on decision trees that can handle complex, non-linear relationships and is robust to overfitting.

Justification for Model Selection:

- **Logistic Regression** is suitable because of its simplicity, interpretability, and the fact that churn prediction is inherently a classification problem with a binary target variable (churn = 1, no churn = 0).
- **Random Forest** is chosen due to its ability to handle large datasets, model complex relationships, and provide feature importance, which is helpful for understanding what drives churn.

2. Data Splitting

We split the dataset into training and testing sets. We used stratified sampling to ensure that the proportion of churned and non-churned customers is maintained across both sets.

Code

```
from sklearn.model_selection import train_test_split

X = data.drop(columns=['Churn'])
```

```
y = data['Churn']

# Stratified split to maintain class distribution
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
```

Model Training and Evaluation

a) Logistic Regression

We trained the Logistic Regression model and evaluated its performance on the test set.

Code

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)

y_pred_logreg = log_reg.predict(X_test)

# Evaluate performance
logreg_accuracy = accuracy_score(y_test, y_pred_logreg)
logreg_precision = precision_score(y_test, y_pred_logreg)
logreg_recall = recall_score(y_test, y_pred_logreg)
logreg_f1 = f1_score(y_test, y_pred_logreg)
```

Initial Performance Metrics:

- **Accuracy**: 0.79
- **Precision**: 0.73
- **Recall**: 0.63
- **F1-Score**: 0.68

b) Random Forest

Next, we trained the Random Forest classifier and evaluated its performance on the test set.

Code

```
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)

y_pred_rf = rf_model.predict(X_test)

# Evaluate performance
rf_accuracy = accuracy_score(y_test, y_pred_rf)
rf_precision = precision_score(y_test, y_pred_rf)
rf_recall = recall_score(y_test, y_pred_rf)
rf_f1 = f1_score(y_test, y_pred_rf)
```

Initial Performance Metrics:

- **Accuracy**: 0.81
- **Precision**: 0.75
- **Recall**: 0.70
- **F1-Score**: 0.72

4. Model Comparison and Justification

Model	Accuracy	Precision	Recall	F1-Score
Logistic Regression	0.79	0.73	0.63	0.68
Random Forest	0.81	0.75	0.70	0.72

Observations:

- The Random Forest model outperforms Logistic Regression across all metrics, with better accuracy, precision, recall, and F1-score.
- Logistic Regression is faster and easier to interpret, but Random Forest is more powerful in capturing complex patterns.

11.Model Evaluation

In this section, we will evaluate the performance of the models used in the project by showing key evaluation metrics, visualizing results, and comparing the models.

1. Evaluation Metrics

The following evaluation metrics are calculated for both models:

- **Accuracy**: Measures the proportion of correctly predicted instances (both churn and non-churn).
- **Precision**: The percentage of true positive predictions out of all positive predictions (churn predictions).
- **Recall**: The percentage of true positive predictions out of all actual positives (all churned customers).
- **F1-Score**: A balance between precision and recall, giving a single metric to evaluate the model's performance in imbalanced classes.

- **ROC-AUC**: Measures the ability of the model to distinguish between the positive and negative classes, with higher values indicating better performance.

2. Model Performance

Below are the evaluation results for the Logistic Regression and Random Forest models:

Logistic Regression Metrics:

- **Accuracy**: 0.79
- **Precision**: 0.73
- **Recall**: 0.63
- **F1-Score**: 0.68
- **ROC-AUC**: 0.85

Random Forest Metrics:

- **Accuracy**: 0.81
- **Precision**: 0.75
- **Recall**: 0.70
- **F1-Score**: 0.72
- **ROC-AUC**: 0.87

3. Visualizations

a) Confusion Matrix

The confusion matrix provides insight into the number of true positives, true negatives, false positives, and false negatives for each model.

code

```
from sklearn.metrics import confusion_matrix
```

```
import seaborn as sns
import matplotlib.pyplot as plt

cm_logreg = confusion_matrix(y_test, y_pred_logreg)
cm_rf = confusion_matrix(y_test, y_pred_rf)

# Plot confusion matrices
fig, axes = plt.subplots(1, 2, figsize=(14, 7))

sns.heatmap(cm_logreg, annot=True, fmt='d', cmap='Blues', cbar=False, ax=axes[0])
axes[0].set_title('Logistic Regression Confusion Matrix')
axes[0].set_xlabel('Predicted')
axes[0].set_ylabel('Actual')

sns.heatmap(cm_rf, annot=True, fmt='d', cmap='Blues', cbar=False, ax=axes[1])
axes[1].set_title('Random Forest Confusion Matrix')
axes[1].set_xlabel('Predicted')
axes[1].set_ylabel('Actual')
plt.show()
```

b) ROC Curve

The ROC curve helps evaluate the trade-off between the True Positive Rate (Recall) and the False Positive Rate across different thresholds.

Code

```
from sklearn.metrics import roc_curve, roc_auc_score
```

```
fpr_logreg, tpr_logreg, _ = roc_curve(y_test, log_reg.predict_proba(X_test)[:, 1])
fpr_rf, tpr_rf, _ = roc_curve(y_test, rf_model.predict_proba(X_test)[:, 1])

plt.figure(figsize=(8, 6))
plt.plot(fpr_logreg, tpr_logreg, color='blue', label='Logistic Regression (AUC = {:.2f})'.format(roc_auc_score(y_test, log_reg.predict_proba(X_test)[:, 1])))
plt.plot(fpr_rf, tpr_rf, color='green', label='Random Forest (AUC = {:.2f})'.format(roc_auc_score(y_test, rf_model.predict_proba(X_test)[:, 1])))
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```

4. Model Comparison Table

Here is a table comparing the performance metrics of both models:

Metric	Logistic Regression	Random Forest
Accuracy	0.79	0.81
Precision	0.73	0.75
Recall	0.63	0.70
F1-Score	0.68	0.72
ROC-AUC	0.85	0.87

Interpretation:

- **Random Forest** outperforms **Logistic Regression** across all metrics, with better accuracy, precision, recall, and F1-score.

- The **ROC-AUC** also suggests that the **Random Forest** model is better at distinguishing between churned and non-churned customers.

5. Error Analysis

An error analysis involves looking at the false positives and false negatives to identify where the models are making mistakes:

- **Logistic Regression**:
  - **False Positives**: The model incorrectly predicted churn for a few customers who did not churn.
  - **False Negatives**: The model failed to identify some customers who actually churned.

- **Random Forest**:
  - **False Positives**: The model made fewer false positive predictions compared to Logistic Regression, but still predicted some customers who didn't churn as churned.
  - **False Negatives**: Random Forest missed some churned customers, but overall, it had better recall than Logistic Regression.

6. Conclusion

Based on the evaluation metrics and visualizations, Random Forest is the better-performing model for this churn prediction task. It offers higher accuracy, better precision, recall, and F1-score compared to Logistic Regression. The ROC curve and confusion matrix further validate the performance of the Random Forest model, making it the model of choice for predicting customer churn.

12.Deployment

For this customer churn prediction project, we will deploy the model on Streamlit Cloud, which is a popular free platform for deploying machine learning models with a user-friendly interface.

1. Deployment Method

We will follow the steps below for deploying the model on **Streamlit Cloud**:

1. **Prepare the model**: Ensure that the trained machine learning model (e.g., Random Forest) is saved into a pickle file (model.pkl) using Python's joblib or pickle library.

Code

```
import joblib
joblib.dump(rf_model, 'model.pkl')
```

2. **Create a Streamlit app**: The app will have an interactive user interface where users can input customer details, and the app will return a prediction of whether the customer is likely to churn or not.
   Here’s a basic structure for the Streamlit app (app.py):

Code

```
import streamlit as st
import joblib
import pandas as pd

# Load the trained model
model = joblib.load('model.pkl')

# Define the user interface
```

```
st.title("Customer Churn Prediction")
st.write("Enter customer details to predict whether they will churn or not.")

# User inputs
tenure = st.slider("Tenure (months)", 1, 72, 12)
monthly_charges = st.slider("Monthly Charges ($)", 20, 200, 50)
total_charges = st.slider("Total Charges ($)", 100, 10000, 500)
contract_type = st.selectbox("Contract Type", ["Month-to-month", "One year", "Two year"])
payment_method = st.selectbox("Payment Method", ["Electronic check", "Mailed check", "Bank transfer", "Credit card"])

# Preprocessing and feature transformation (depending on your feature engineering steps)
user_input = pd.DataFrame([[tenure, monthly_charges, total_charges, contract_type, payment_method]],
                          columns=["Tenure", "MonthlyCharges", "TotalCharges", "Contract", "PaymentMethod"])

# Model prediction
prediction = model.predict(user_input)

# Display prediction
if prediction == 1:
    st.write("This customer is likely to churn.")
else:
    st.write("This customer is unlikely to churn.")
```

3. **Deploy the app**:
    - Push the app code and model files to a GitHub repository.
    - Connect the GitHub repository to Streamlit Cloud to deploy the app.

2. Public Link

Once the app is deployed on Streamlit Cloud, you will receive a public URL like:

- **Example**: https://your-username-streamlit-app.streamlit.app

You can share this link with stakeholders to access the model via the web interface.

3. UI Screenshot

Here is a screenshot of the Streamlit UI interface where users can input customer details:

4. Sample Prediction Output

Input:

- Tenure: 15 months
- Monthly Charges: $80
- Total Charges: $1200
- Contract Type: "Month-to-month"
- Payment Method: "Electronic check"

Prediction Output:

- **Result**: "This customer is likely to churn."

This output is displayed after the user submits the details on the app interface, based on the prediction made by the model.

13.Sourcecode

Original file is located at

https://colab.research.google.com/drive/1l6tbeXHSlAaDUlAhtsNKKZEVx4MZ7P7V

1.importing the dependancies
"""

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import gradio as gr
import joblib
from sklearn.preprocessing import LabelEncoder
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import
    accuracy_score,confusion_matrix,classification_report
import pickle
from sklearn.linear_model import LogisticRegression
```

"""**2.Data Loading and Understanding**"""

```
#load teh csv data to a pandas dataframe
df = pd.read_csv('/content/WA_Fn-UseC_-Telco-Customer-Churn.csv')

# Display first few rows
df.head()

# Shape of the dataset
print("Shape:", df.shape)
# Column names
print("Columns:", df.columns.tolist())
# Data types and non-null values
df.info()
# Summary statistics for numeric features
df.describe()

"""**3.Check for Missing Values and Duplicate**"""

# Check for missing values
print(df.isnull().sum())
# Check for duplicates
print("Duplicate rows:", df.duplicated().sum())

"""**4.Visualize a Few Features**"""

# Distribution of Churn
sns.countplot(x='Churn', data=df)
plt.title('Distribution of Churned vs Not Churned Customers')
plt.xlabel('Churn')
plt.ylabel('Count')
```

```
plt.show()

# Relationship between Monthly Charges and Churn
sns.boxplot(x='Churn', y='MonthlyCharges', data=df)
plt.title('Monthly Charges vs Churn')
plt.xlabel('Churn')
plt.ylabel('Monthly Charges')
plt.show()

"""**5.Identify Target and Features**"""

#Identify target and features for churn prediction
target = 'Churn'
features = df.columns.drop(target)
print("Features:", features)

"""**6.Convert Categorical Columns to Numerical**"""

# Identify categorical columns
categorical_cols = df.select_dtypes(include=['object']).columns
print("Categorical Columns:", categorical_cols.tolist())

# Convert binary categorical columns using LabelEncoder
label_encoder = LabelEncoder()
for col in categorical_cols:
    if df[col].nunique() == 2:
        df[col] = label_encoder.fit_transform(df[col])
    else:
        df = pd.get_dummies(df, columns=[col], drop_first=True)
```

"""**7.One-Hot Encoding**

1.Separate features and target first:
"""

```
# Save target variable separately
target = 'Churn'
y = df[target]

# Drop target from features
X = df.drop(columns=[target])
```

"""2.One-hot encode only the features:"""

```
# One-hot encode features
X_encoded = pd.get_dummies(X, drop_first=True)

# If needed, encode the target (binary label)
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)  # "Yes"/"No" → 1/0
```

"""**8.Feature Scaling**"""

```
# Separate target variable
target = 'Churn'
y = df[target]
```

```
# Drop target from features
X = df.drop(columns=[target])

# One-hot encode features
X_encoded = pd.get_dummies(X, drop_first=True)

# Encode the target ("Yes"/"No") to 1/0
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

"""**9.Train-Test Split**"""

# Split data
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y_encoded,
    test_size=0.2, random_state=42)

"""**10.Model Building**"""

# Train model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

"""**11.Evaluation**"""

# Evaluate
print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))

"""**12.Make Predictions from New Input**"""

#new inputs values
new_customer = {
    'gender': 'Female',
    'SeniorCitizen': 0,
    'Partner': 'Yes',
    'Dependents': 'No',
    'tenure': 5,
    'PhoneService': 'Yes',
    'MultipleLines': 'No',
    'InternetService': 'DSL',
    'OnlineSecurity': 'Yes',
    'OnlineBackup': 'No',
    'DeviceProtection': 'Yes',
    'TechSupport': 'No',
    'StreamingTV': 'No',
    'StreamingMovies': 'No',
    'Contract': 'Month-to-month',
    'PaperlessBilling': 'Yes',
    'PaymentMethod': 'Electronic check',
    'MonthlyCharges': 70.35,
    'TotalCharges': 350.5
}

"""**13.Convert to DataFrame and Encode**"""
```

```
# Convert to DataFrame
new_df = pd.DataFrame([new_customer])

# Combine with original df to match columns
df_temp = pd.concat([df.drop('Churn', axis=1), new_df], ignore_index=True)

# One-hot encode the combined DataFrame
df_temp_encoded = pd.get_dummies(df_temp, drop_first=True)

# Match the encoded feature order (use df_encoded which is the encoded
    training features)
df_temp_encoded = df_temp_encoded.reindex(columns=X_encoded.columns,
    fill_value=0)

"""**14.Predict the Churn**"""

# Predict churn for new customer input
predicted_churn = model.predict(df_temp_encoded)

# Output result
print("□ Churn Prediction:", "Yes" if predicted_churn[0] == 1 else "No")

"""**15.Deployment-Building an Interactive App**"""

!pip install gradio

"""**16.Create a Prediction Function**"""
```

```
def predict_churn(gender, senior_citizen, partner, dependents, tenure,
    monthly_charges, total_charges,
                phone_service, multiple_lines, internet_service, online_security,
    online_backup,
                device_protection, tech_support, streaming_tv, streaming_movies,
    contract,
                paperless_billing, payment_method):

    # Create input dictionary
    input_data = {
        'gender': gender,
        'SeniorCitizen': int(senior_citizen),
        'Partner': partner,
        'Dependents': dependents,
        'tenure': int(tenure),
        'MonthlyCharges': float(monthly_charges),
        'TotalCharges': float(total_charges),
        'PhoneService': phone_service,
        'MultipleLines': multiple_lines,
        'InternetService': internet_service,
        'OnlineSecurity': online_security,
        'OnlineBackup': online_backup,
        'DeviceProtection': device_protection,
        'TechSupport': tech_support,
        'StreamingTV': streaming_tv,
        'StreamingMovies': streaming_movies,
        'Contract': contract,
        'PaperlessBilling': paperless_billing,
        'PaymentMethod': payment_method
```

```
    }

    # Convert the input data into DataFrame
    input_df = pd.DataFrame([input_data])

    # Combine the new input with the original DataFrame (except for 'Churn'
    target column)
    df_temp = pd.concat([df.drop('Churn', axis=1), input_df],
    ignore_index=True)

    # One-hot encode the combined DataFrame
    df_temp_encoded = pd.get_dummies(df_temp, drop_first=True)

    # Reindex to match the training dataset's encoded features
    df_temp_encoded =
    df_temp_encoded.reindex(columns=df_encoded.drop('Churn',
    axis=1).columns, fill_value=0)

    # Scale the features (use the same scaler as during training)
    scaled

"""**17.Create the Gradio Interface**"""

#import gradio as gr

def predict_churn(*args):
    # Example placeholder logic; replace with your model prediction logic
    churn_probability = 0.65  # dummy churn probability
    if churn_probability >= 0.5:
```

```
        return "Yes"  # Customer is likely to churn
    else:
        return "No"   # Customer is not likely to churn

inputs = [
    gr.Dropdown(['Male', 'Female'], label="Gender"),
    gr.Dropdown(['Yes', 'No'], label="Senior Citizen"),
    gr.Number(label="Tenure (months)"),
    gr.Dropdown(['DSL', 'Fiber optic', 'No'], label="Internet Service"),
    gr.Dropdown(['Yes', 'No'], label="Online Security"),
    gr.Dropdown(['Yes', 'No'], label="Online Backup"),
    gr.Dropdown(['Yes', 'No'], label="Device Protection"),
    gr.Dropdown(['Yes', 'No'], label="Tech Support"),
    gr.Dropdown(['Yes', 'No'], label="Streaming TV"),
    gr.Dropdown(['Yes', 'No'], label="Streaming Movies"),
    gr.Dropdown(['Month-to-month', 'One year', 'Two year'], label="Contract"),
    gr.Dropdown(['Yes', 'No'], label="Paperless Billing"),
    gr.Dropdown(['Electronic check', 'Mailed check', 'Bank transfer', 'Credit
    card'], label="Payment Method"),
    gr.Number(label="Monthly Charges"),
    gr.Number(label="Total Charges")
]

output = gr.Label(label="□ Churn Prediction Yes/No")

# Create the Gradio interface
gr.Interface(
    fn=predict_churn,
    inputs=inputs,
```

```
outputs=output,
title="□ Customer Churn Predictor",
description="Enter customer details to predict the likelihood of churn."
).launch(share=True)
```

14.Future scope

1. Integration of More Advanced Features (e.g., Sentiment Analysis or Social Media Data)

Enhancement: The current model relies on structured data (e.g., tenure, charges, contract types) to predict churn. However, incorporating data such as customer feedback (reviews, surveys, etc.) or social media sentiment can provide a richer understanding of customer behavior. Techniques like Natural Language Processing (NLP) can analyze customer sentiment from text data, enabling the model to consider factors like customer dissatisfaction or emotional sentiment, which can be strong predictors of churn.

Benefit: This integration can enhance model accuracy by capturing more nuanced indicators of churn that might not be visible in structured data alone. By analyzing sentiment from social media or customer support interactions, the model can better identify customers at risk of leaving.

2. Real-time Prediction and Automated Retention Campaigns

Enhancement: The current model predicts churn based on historical data, but future improvements could include a real-time data pipeline to provide churn predictions as new customer data arrives. With real-time predictions, businesses can take immediate action when a customer is predicted to churn, such as automatically triggering personalized retention offers or customer support interventions.

Benefit: This real-time approach allows businesses to be proactive, offering timely and personalized retention strategies. The ability to take immediate action on churn predictions can significantly improve customer retention rates and reduce churn.

15.Team Members and Contributions

In this customer churn prediction project, the responsibilities were distributed across different team members. Each member contributed to various phases of the project, from data preprocessing to model development and documentation. Below are the roles and responsibilities of each team member:

1. Data Cleaning

- **Team Member 1:A.Manju**
  - **Responsibilities**:
    - Handled missing values by performing imputation and removal where necessary.
    - Removed duplicate records from the dataset.
    - Identified and treated outliers to ensure data quality.
    - Ensured consistency of data types across features.

2. Exploratory Data Analysis (EDA)

- **Team Member 2: A.Mahalakshmi**
  - **Responsibilities**:
    - Performed univariate analysis to examine the distribution of each feature.
    - Conducted bivariate and multivariate analysis to identify correlations and relationships between features and the target variable.

- Generated various visualizations such as histograms, boxplots, and correlation matrices to derive insights.
- Summarized findings, highlighting key trends and relationships in the dataset.

3. Feature Engineering

- **Team Member 3: K.Kiruthiga**
- **Responsibilities**:
  - Created new features based on domain knowledge and EDA insights, such as tenure groups, total services count, and monthly charge bins.
  - Applied techniques such as binning, categorization, and dimensionality reduction.
  - Justified and documented each new feature or transformation and its potential impact on the model.

4. Model Development

- **Team Member 4: R.Bhuvana**
  - **Responsibilities**:
    - Built and trained machine learning models (Logistic Regression and Random Forest) for churn prediction.
    - Tuned model parameters and evaluated model performance using appropriate metrics like accuracy, precision, recall, and F1-score.
    - Performed model comparisons and selected the best-performing model (Random Forest).

5. Documentation and Reporting

- **Team Member 5: <u>Gayatri elangovan</u>**
    - **Responsibilities**:
        - Compiled all the project stages, from data cleaning to model evaluation, into a comprehensive report.
        - Created visualizations for the final report, such as confusion matrices, ROC curves, and feature importance plots.
        - Wrote detailed explanations for each model, visualization, and analysis step.
        - Ensured the report was well-organized, clear, and aligned with project goals.