**Phase-3 Submission Template**

**Student Name:** Thilshan S

**Register Number:** 712523205067

**Institution:** PPG Institute of Technology

**Department:** B.Tech Information Technology

**Date of Submission:** 16-05-2006

**Github Repository Link: [REPO LINK](https://github.com/thilshan-s/NM_Thilshan_DS)**

### **1. Problem Statement**

*Customer churn is a pressing issue across subscription-based industries such as telecom, banking, and streaming services. It refers to when customers stop using a service or switch to a competitor. This project focuses on solving a* ***binary classification problem****—predicting whether a customer will churn (Yes/No) using demographic and service-related features. Reducing churn is essential for sustaining growth, as acquiring new customers is costlier than retaining existing ones. Accurate churn prediction enables businesses to take proactive measures and improve customer retention strategies.*

### **2. Abstract**

*This project addresses the problem of customer churn prediction in the telecom industry using machine learning techniques. The main objective is to build models that can accurately classify customers as likely to churn or not, based on demographic and usage data. We pre process the Telco Customer Churn data set from Kaggle, apply exploratory data analysis (EDA), perform feature engineering, and train several classification models including Logistic Regression, Random Forest, and XG Boost. The models are evaluated using precision, recall, F1-score, and ROC -AUC. Insights from the analysis highlight key churn indicators such as contract type and tenure, making the models valuable tools for business decision-making and potential dashboard integration.*

### **3. System Requirements**

***Hardware:***

* *Minimum RAM: 8 GB*
* *Processor: Intel i5 (or equivalent) or above for smooth model training*

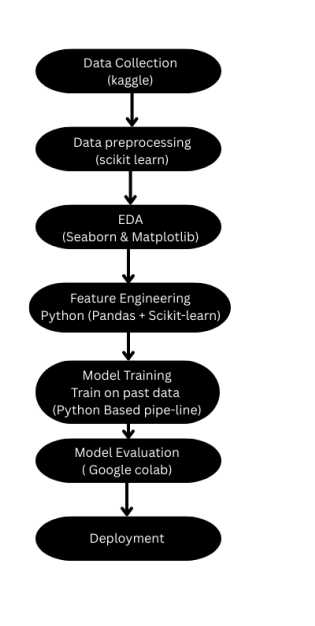
***Software:***

* *Python version: 3.7+*
* *Required Libraries: pandas, numpy, matplotlib, seaborn, scikit-learn, xgboost, plotly, shap*
* *IDE: Google Co-lab or Jupiter Notebook*

### **4. Objectives**

* *Predict customer churn using reliable and interpretable machine learning models.*
* *Identify key features influencing customer churn (e.g., tenure, contract type)*
* *Evaluate and compare model performance based on accuracy, precision, recall, F1-score, and ROC-AUC.*
* *Provide actionable insights to reduce churn and improve business strategies.*
* *Build models that can be integrated with customer dashboards or CRM tools.*

1. **Flowchart of Project Workflow**



### **6.Dataset Description**

***Source****: [Kaggle – Telco Customer Churn](https://www.kaggle.com/datasets/blastchar/telcocustomerchurn" \t "_new)*

***Type****: Public, structured, static dataset*

***Size****:*

*Rows: 7043 (7032 after cleaning)*

*Columns: 21 original features, 31 after feature engineering*

***Target Variable****: Churn (Yes/No)*

### 

### **7.Data Preprocessing**

* *Removed irrelevant column: customerID*
* *Converted TotalCharges to numeric and dropped rows with missing/invalid values3*
* *One-hot encoded categorical variables*
* *Normalized continuous features (MonthlyCharges, TotalCharges)*
* *Ensured all inputs are numeric for compatibility*

### 

### **8.Exploratory Data Analysis (EDA)**

*Uni-variate Analysis:*

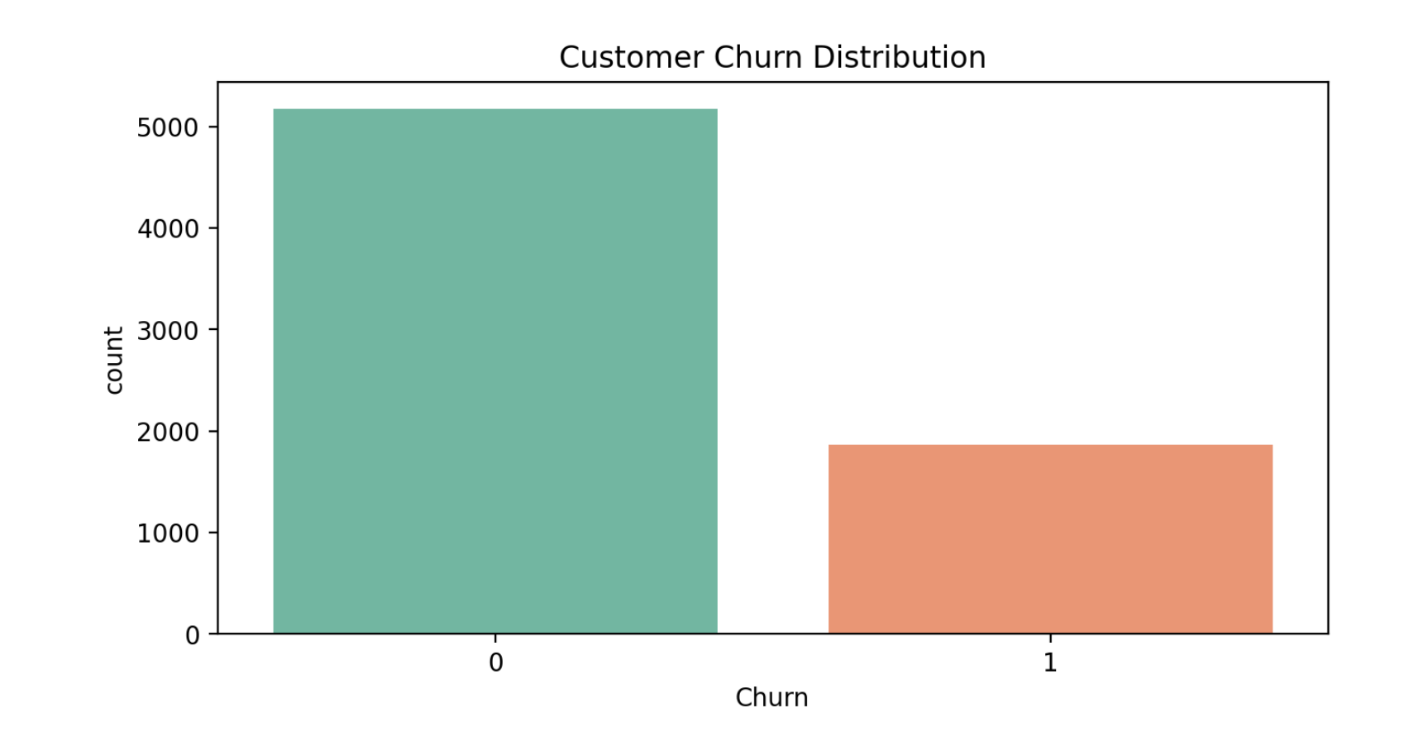
* *tenure and MonthlyCharges have a varied distribution*
* *High count of month-to-month contracts and electronic payment methods*

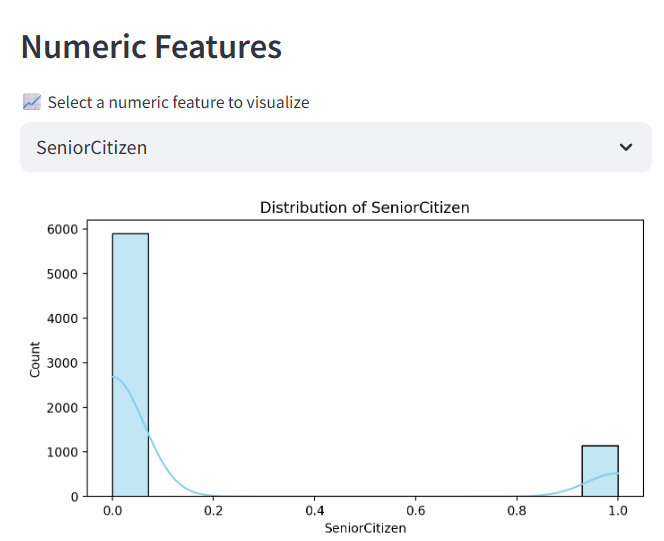
*Bivariate/Multivariate Analysis:*

* *Higher churn among customers with fiber optic internet*
* *Tenure inversely correlated with churn*
* *Contract type and payment method are strong churn indicators*

*Key Insights:*

* *Long-term customers churn less frequently*
* *Month-to-month contracts and electronic check payments are red flags*





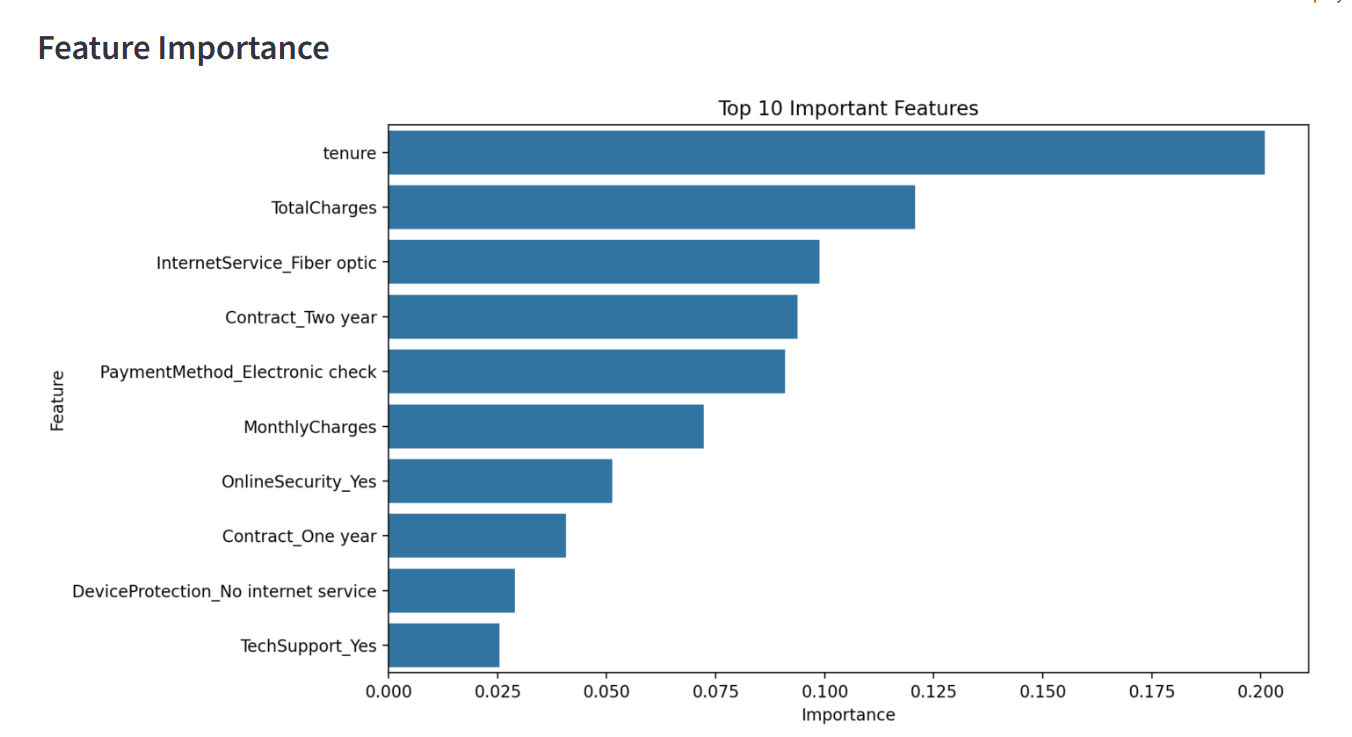
### 

### **9. Feature Engineering**

* *Encoded categorical variables*
* *Scaled features to normalize ranges*
* *Selected features based on correlation and model performance*

***Impact:***

*Proper encoding and scaling enhanced model accuracy, especially for algorithms sensitive to feature scale like SVM and Logistic Regression.*



### 

### **10. Model Building**

### ***Models Tried:*** *o Logistic Regression o Random Forest o XGBoost*

### *Support Vector Machine (SVM)*

### *K-Nearest Neighbors (KNN)*

### 

### ***Training Details:***

### *Train/test split = 80/20*

### *Hyperparameters tuned minimally for speed*

### **11. Model Evaluation**

***Metrics:***

* *Accuracy*
* *F1 Score*
* *Precision*
* *Recall*
* *ROC AUC*

***Comparison Table:***



### **12. Deployment**

** ***Platform:*** *Streamlit Cloud*

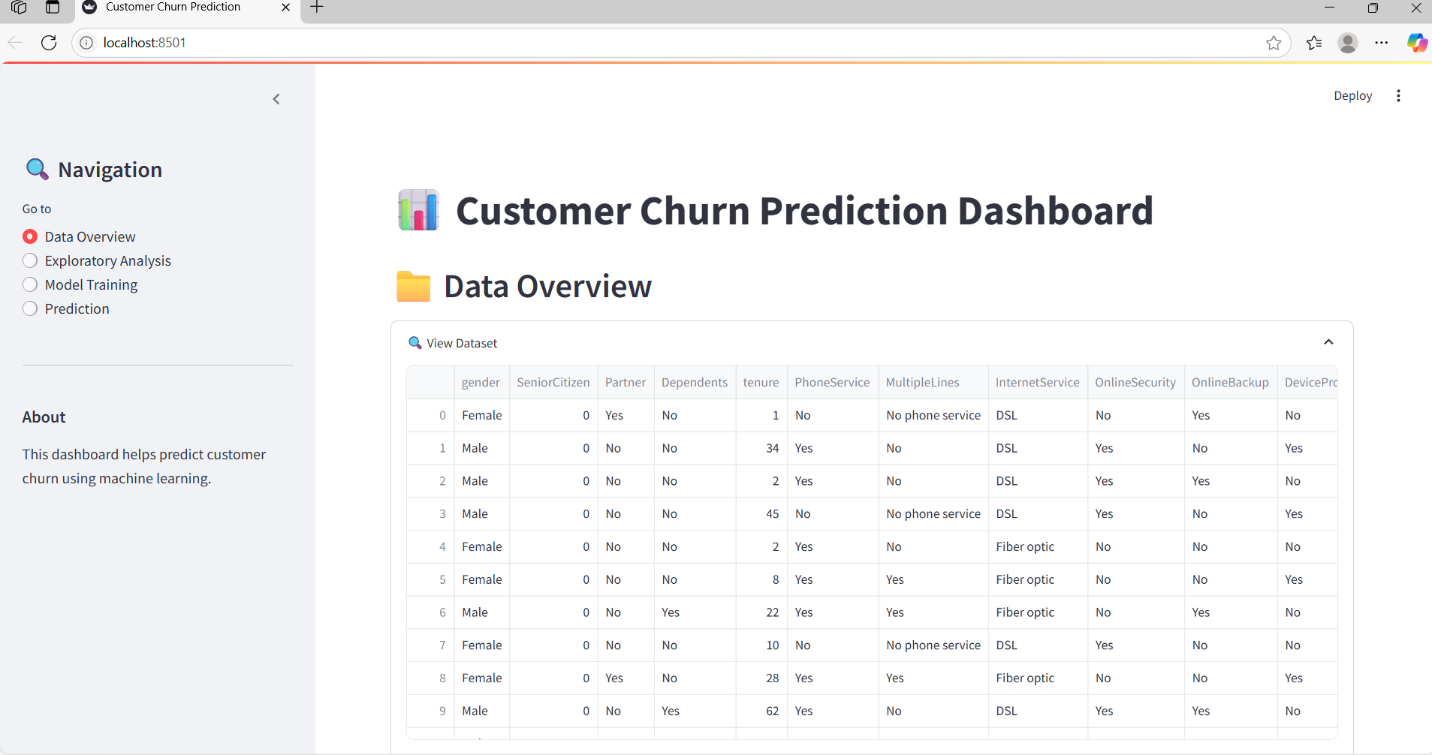
** ***Method:*** *Streamlit Python script*

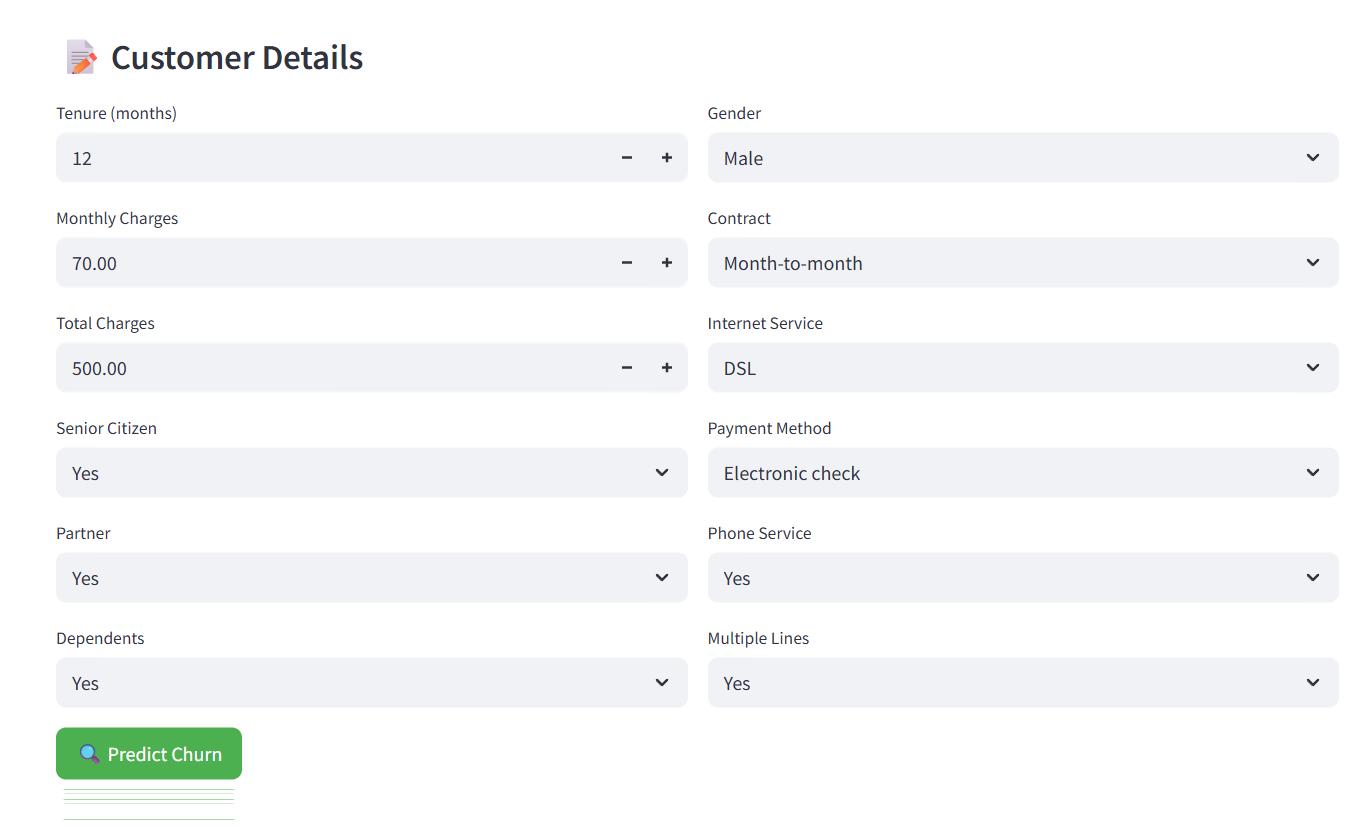
** ***Link:*** *[https://nanmudhalvan-project2adfcuugvxxgel5vd22q6o9.streamlit.app/]*

** ***UI Features:***

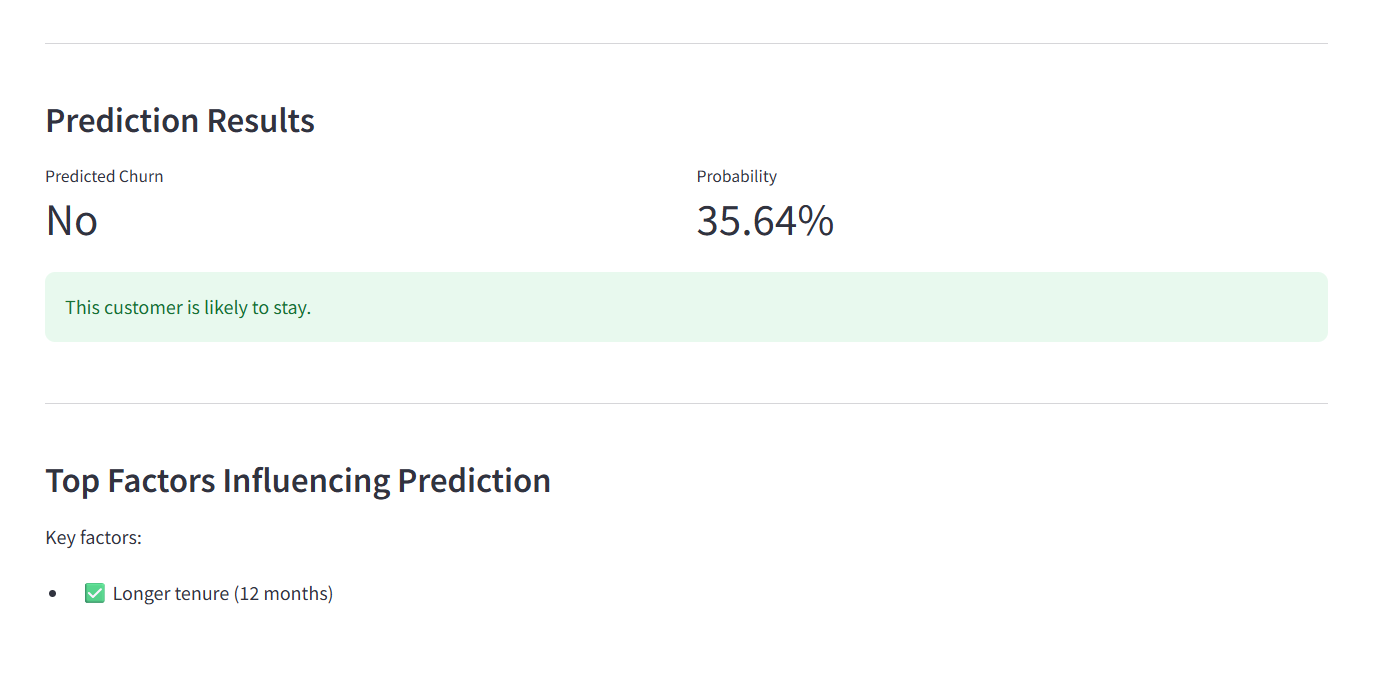
* *Dataset Upload*
* *Model Comparison*
* *SHAP Explainability*

***UI SCREENSHOT:***





***Prediction output:***



**13. Source code**

*import streamlit as st*

*import pandas as pd*

*import numpy as np*

*import matplotlib.pyplot as plt*

*import seaborn as sns*

*import joblib*

*from sklearn.model\_selection import train\_test\_split*

*from sklearn.preprocessing import StandardScaler, LabelEncoder*

*from sklearn.ensemble import RandomForestClassifier*

*from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score*

*from sklearn.linear\_model import LogisticRegression*

*from sklearn.svm import SVC*

*from imblearn.over\_sampling import SMOTE*

*import pickle*

*# Set Streamlit page configuration*

*st.set\_page\_config(*

*page\_title="Customer Churn Prediction",*

*layout="wide",*

*initial\_sidebar\_state="expanded",*

*menu\_items={*

*'Get Help': 'https://example.com/help',*

*'Report a bug': "https://example.com/bug",*

*'About': "# Customer Churn Prediction Dashboard"*

*}*

*)*

*# Custom CSS for modern look*

*st.markdown("""*

*<style>*

*.main {background-color: #f8f9fa;}*

*.stButton>button {border-radius: 8px; background-color: #4CAF50; color: white; border: none; padding: 8px 16px;}*

*.stSelectbox>div>div {background-color: #f0f2f6; border-radius: 8px;}*

*.stSlider>div {color: #0c4a6e;}*

*.css-1d391kg {background-color: #ffffff; border-radius: 8px; padding: 1rem;}*

*.stAlert {border-radius: 8px;}*

*</style>*

*""", unsafe\_allow\_html=True)*

*# Title*

*st.title("📊 Customer Churn Prediction Dashboard")*

*# Initialize session state for model persistence*

*if 'trained\_model' not in st.session\_state:*

*st.session\_state.trained\_model = None*

*if 'feature\_names' not in st.session\_state:*

*st.session\_state.feature\_names = None*

*if 'preprocessor' not in st.session\_state:*

*st.session\_state.preprocessor = None*

*# Sidebar navigation*

*with st.sidebar:*

*st.title("🔍 Navigation")*

*options = st.radio("Go to", ["Data Overview", "Exploratory Analysis", "Model Training", "Prediction"])*

*st.markdown("---")*

*st.markdown("### About")*

*st.markdown("This dashboard helps predict customer churn using machine learning.")*

*# Load data*

*@st.cache\_data*

*def load\_data():*

*try:*

*url = "https://raw.githubusercontent.com/IBM/telco-customer-churn-on-icp4d/master/data/Telco-Customer-Churn.csv"*

*data = pd.read\_csv(url)*

*return data*

*except Exception as e:*

*st.error(f"Error loading data: {str(e)}")*

*return None*

*df = load\_data()*

*if df is None:*

*st.stop()*

*# Preprocessing function*

*@st.cache\_data*

*def preprocess\_data(df):*

*# Make a copy to avoid modifying cached data*

*df = df.copy()*

*# Drop customer ID*

*df = df.drop('customerID', axis=1)*

*# Convert TotalCharges to numeric*

*df['TotalCharges'] = pd.to\_numeric(df['TotalCharges'], errors='coerce')*

*df['TotalCharges'] = df['TotalCharges'].fillna(df['TotalCharges'].median())*

*# Encode target*

*df['Churn'] = LabelEncoder().fit\_transform(df['Churn'])*

*# Get categorical columns (excluding target)*

*categorical\_cols = df.select\_dtypes(include=['object']).columns*

*# One-hot encode*

*df\_encoded = pd.get\_dummies(df, columns=categorical\_cols, drop\_first=True)*

*return df, df\_encoded*

*df, df\_encoded = preprocess\_data(df)*

*# Page: Data Overview*

*if options == "Data Overview":*

*st.header("📁 Data Overview")*

*with st.expander("🔍 View Dataset", expanded=True):*

*st.dataframe(df, use\_container\_width=True)*

*st.subheader("📐 Dataset Shape")*

*st.write(f"\*\*Rows:\*\* {df.shape[0]} | \*\*Columns:\*\* {df.shape[1]}")*

*col1, col2 = st.columns(2)*

*with col1:*

*with st.expander("🧾 Column Data Types"):*

*st.write(df.dtypes)*

*with col2:*

*with st.expander("🚨 Missing Values"):*

*st.write(df.isnull().sum())*

*# Page: Exploratory Analysis*

*elif options == "Exploratory Analysis":*

*st.header("📊 Exploratory Data Analysis")*

*st.subheader("Churn Distribution")*

*fig, ax = plt.subplots(figsize=(8, 4))*

*sns.countplot(x='Churn', data=df, palette='Set2', ax=ax)*

*ax.set\_title('Customer Churn Distribution')*

*st.pyplot(fig)*

*st.markdown("---")*

*col1, col2 = st.columns(2)*

*with col1:*

*st.subheader("Numeric Features")*

*numeric\_cols = df.select\_dtypes(include=['int64', 'float64']).columns*

*selected\_num\_col = st.selectbox("📈 Select a numeric feature to visualize", numeric\_cols)*

*fig, ax = plt.subplots(figsize=(8, 4))*

*sns.histplot(df[selected\_num\_col], kde=True, color="skyblue", ax=ax)*

*ax.set\_title(f'Distribution of {selected\_num\_col}')*

*st.pyplot(fig)*

*with col2:*

*st.subheader("Categorical Features")*

*categorical\_cols = df.select\_dtypes(include=['object']).columns.drop('Churn')*

*selected\_cat\_col = st.selectbox("🗂 Select a categorical feature to visualize", categorical\_cols)*

*fig, ax = plt.subplots(figsize=(10, 6))*

*sns.countplot(x=selected\_cat\_col, hue='Churn', data=df, palette='Set1', ax=ax)*

*plt.xticks(rotation=45)*

*ax.set\_title(f'Churn by {selected\_cat\_col}')*

*st.pyplot(fig)*

*st.markdown("---")*

*st.subheader("📌 Correlation Matrix")*

*fig, ax = plt.subplots(figsize=(12, 8))*

*sns.heatmap(df[numeric\_cols].corr(), annot=True, cmap='coolwarm', ax=ax)*

*ax.set\_title('Feature Correlation Matrix')*

*st.pyplot(fig)*

*# Page: Model Training*

*elif options == "Model Training":*

*st.header("🧠 Model Training")*

*X = df\_encoded.drop('Churn', axis=1)*

*y = df\_encoded['Churn']*

*st.sidebar.subheader("🔧 Model Configuration")*

*test\_size = st.sidebar.slider("Test Size", 0.1, 0.5, 0.2, 0.05)*

*random\_state = st.sidebar.number\_input("Random State", 0, 100, 42)*

*use\_smote = st.sidebar.checkbox("Balance Classes with SMOTE")*

*# Split data*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(*

*X, y,*

*test\_size=test\_size,*

*random\_state=random\_state,*

*stratify=y*

*)*

*# Scale features*

*scaler = StandardScaler()*

*X\_train\_scaled = scaler.fit\_transform(X\_train)*

*X\_test\_scaled = scaler.transform(X\_test)*

*# Handle class imbalance*

*if use\_smote:*

*smote = SMOTE(random\_state=random\_state)*

*X\_train\_scaled, y\_train = smote.fit\_resample(X\_train\_scaled, y\_train)*

*st.subheader("Select and Train Model")*

*model\_type = st.selectbox("Choose Model",*

*["Random Forest", "Logistic Regression", "Support Vector Machine"])*

*if model\_type == "Random Forest":*

*n\_estimators = st.slider("Number of Trees", 50, 300, 100, 25)*

*max\_depth = st.slider("Max Depth", 2, 20, 6)*

*model = RandomForestClassifier(*

*n\_estimators=n\_estimators,*

*max\_depth=max\_depth,*

*random\_state=random\_state*

*)*

*elif model\_type == "Logistic Regression":*

*C = st.slider("C (Regularization Strength)", 0.01, 10.0, 1.0)*

*model = LogisticRegression(*

*C=C,*

*max\_iter=1000,*

*random\_state=random\_state*

*)*

*else:*

*C = st.slider("C (SVM Regularization)", 0.01, 10.0, 1.0)*

*kernel = st.selectbox("SVM Kernel", ["linear", "rbf", "poly"])*

*model = SVC(*

*C=C,*

*kernel=kernel,*

*probability=True,*

*random\_state=random\_state*

*)*

*if st.button("🚀 Train Model"):*

*with st.spinner('Training model...'):*

*model.fit(X\_train\_scaled, y\_train)*

*y\_pred = model.predict(X\_test\_scaled)*

*y\_prob = model.predict\_proba(X\_test\_scaled)[:, 1]*

*# Store model and artifacts in session state*

*st.session\_state.trained\_model = model*

*st.session\_state.feature\_names = X.columns.tolist()*

*st.session\_state.preprocessor = {*

*'scaler': scaler,*

*'use\_smote': use\_smote*

*}*

*st.success("Model trained successfully!")*

*# Evaluation metrics*

*st.subheader("📈 Evaluation Results")*

*col1, col2 = st.columns(2)*

*with col1:*

*st.metric("Accuracy", f"{accuracy\_score(y\_test, y\_pred):.2%}")*

*with col2:*

*st.metric("Positive Class Rate", f"{y\_test.mean():.2%}")*

*st.markdown("---")*

*st.subheader("Classification Report")*

*report = classification\_report(y\_test, y\_pred, output\_dict=True)*

*st.dataframe(pd.DataFrame(report).transpose())*

*st.markdown("---")*

*col1, col2 = st.columns(2)*

*with col1:*

*st.subheader("Confusion Matrix")*

*fig, ax = plt.subplots()*

*sns.heatmap(*

*confusion\_matrix(y\_test, y\_pred),*

*annot=True,*

*fmt='d',*

*cmap='Blues',*

*ax=ax*

*)*

*ax.set\_xlabel('Predicted')*

*ax.set\_ylabel('Actual')*

*st.pyplot(fig)*

*with col2:*

*st.subheader("ROC Curve")*

*from sklearn.metrics import RocCurveDisplay*

*fig, ax = plt.subplots()*

*RocCurveDisplay.from\_estimator(model, X\_test\_scaled, y\_test, ax=ax)*

*ax.plot([0, 1], [0, 1], linestyle='--')*

*st.pyplot(fig)*

*if model\_type == "Random Forest":*

*st.markdown("---")*

*st.subheader("Feature Importance")*

*importance = pd.DataFrame({*

*"Feature": X.columns,*

*"Importance": model.feature\_importances\_*

*}).sort\_values("Importance", ascending=False)*

*fig, ax = plt.subplots(figsize=(10, 6))*

*sns.barplot(*

*x="Importance",*

*y="Feature",*

*data=importance.head(10),*

*ax=ax*

*)*

*ax.set\_title('Top 10 Important Features')*

*st.pyplot(fig)*

*# Option to download feature importance*

*st.download\_button(*

*label="Download Feature Importance",*

*data=importance.to\_csv(index=False),*

*file\_name="feature\_importance.csv",*

*mime="text/csv"*

*)*

*# Page: Prediction*

*elif options == "Prediction":*

*st.header("🔮 Predict Customer Churn")*

*if st.session\_state.trained\_model is None:*

*st.warning("Please train a model first on the Model Training page.")*

*st.stop()*

*st.subheader("📝 Customer Details")*

*col1, col2 = st.columns(2)*

*with col1:*

*tenure = st.number\_input("Tenure (months)", min\_value=0, max\_value=100, value=12)*

*monthly\_charges = st.number\_input("Monthly Charges", min\_value=0.0, max\_value=200.0, value=70.0)*

*total\_charges = st.number\_input("Total Charges", min\_value=0.0, value=500.0)*

*senior\_citizen = st.selectbox("Senior Citizen", ["Yes", "No"])*

*partner = st.selectbox("Partner", ["Yes", "No"])*

*dependents = st.selectbox("Dependents", ["Yes", "No"])*

*with col2:*

*gender = st.selectbox("Gender", ["Male", "Female"])*

*contract = st.selectbox("Contract", ["Month-to-month", "One year", "Two year"])*

*internet\_service = st.selectbox("Internet Service", ["DSL", "Fiber optic", "No"])*

*payment\_method = st.selectbox("Payment Method",*

*["Electronic check", "Mailed check",*

*"Bank transfer", "Credit card"])*

*phone\_service = st.selectbox("Phone Service", ["Yes", "No"])*

*multiple\_lines = st.selectbox("Multiple Lines", ["Yes", "No", "No phone service"])*

*if st.button("🔍 Predict Churn"):*

*try:*

*# Create input DataFrame*

*input\_data = pd.DataFrame({*

*'tenure': [tenure],*

*'MonthlyCharges': [monthly\_charges],*

*'TotalCharges': [total\_charges],*

*'SeniorCitizen\_Yes': [1 if senior\_citizen == "Yes" else 0],*

*'Partner\_Yes': [1 if partner == "Yes" else 0],*

*'Dependents\_Yes': [1 if dependents == "Yes" else 0],*

*'Gender\_Male': [1 if gender == "Male" else 0],*

*'Contract\_One year': [1 if contract == "One year" else 0],*

*'Contract\_Two year': [1 if contract == "Two year" else 0],*

*'InternetService\_Fiber optic': [1 if internet\_service == "Fiber optic" else 0],*

*'InternetService\_No': [1 if internet\_service == "No" else 0],*

*'PaymentMethod\_Credit card (automatic)': [1 if payment\_method == "Credit card" else 0],*

*'PaymentMethod\_Electronic check': [1 if payment\_method == "Electronic check" else 0],*

*'PaymentMethod\_Mailed check': [1 if payment\_method == "Mailed check" else 0],*

*'PhoneService\_Yes': [1 if phone\_service == "Yes" else 0],*

*'MultipleLines\_No phone service': [1 if multiple\_lines == "No phone service" else 0],*

*'MultipleLines\_Yes': [1 if multiple\_lines == "Yes" else 0]*

*})*

*# Ensure all expected features are present*

*missing\_features = set(st.session\_state.feature\_names) - set(input\_data.columns)*

*for feature in missing\_features:*

*input\_data[feature] = 0  # Add missing features with 0 value*

*# Reorder columns to match training data*

*input\_data = input\_data[st.session\_state.feature\_names]*

*# Scale features*

*scaler = st.session\_state.preprocessor['scaler']*

*input\_scaled = scaler.transform(input\_data)*

*# Make prediction*

*model = st.session\_state.trained\_model*

*prediction = model.predict(input\_scaled)*

*probability = model.predict\_proba(input\_scaled)[0][1]*

*# Display results*

*st.markdown("---")*

*st.subheader("Prediction Results")*

*col1, col2 = st.columns(2)*

*with col1:*

*st.metric("Predicted Churn", "Yes" if prediction[0] == 1 else "No")*

*with col2:*

*st.metric("Probability", f"{probability:.2%}")*

*# Show explanation*

*if prediction[0] == 1:*

*st.warning("This customer is likely to churn.")*

*else:*

*st.success("This customer is likely to stay.")*

*# Show feature importance if available*

*if hasattr(model, 'feature\_importances\_'):*

*st.markdown("---")*

*st.subheader("Top Factors Influencing Prediction")*

*# Get feature importances*

*importances = model.feature\_importances\_*

*indices = np.argsort(importances)[::-1][:5]  # Top 5 features*

*# Create explanation*

*explanation = []*

*for i in indices:*

*feature\_name = st.session\_state.feature\_names[i]*

*feature\_value = input\_data.iloc[0][feature\_name]*

*importance = importances[i]*

*# Create human-readable explanation*

*if feature\_name.startswith("Contract\_Two year") and feature\_value == 1:*

*explanation.append(f"✅ Two year contract (reduces churn)")*

*elif feature\_name == "tenure":*

*explanation.append(f"✅ Longer tenure ({tenure} months)")*

*elif feature\_name == "MonthlyCharges" and monthly\_charges > 70:*

*explanation.append(f"⚠️ High monthly charges (${monthly\_charges})")*

*if explanation:*

*st.write("Key factors:")*

*for item in explanation:*

*st.write(f"- {item}")*

*else:*

*st.info("No strong factors identified.")*

*except Exception as e:*

*st.error(f"Error making prediction: {str(e)}")*

**14. Future scope**

* Add live database/API integration to stream customer data.*

* Automate hyperparameter tuning using GridSearchCV or Optuna.*

* Build a feedback system to improve model based on user input.*

* Implement model drift detection and retraining pipeline*

**13. Team Members and Roles**

*Name Contributions*

|  |  |
| --- | --- |
| *Thilshan S* | *Team Leader; coordinated the project workflow, managed deadlines, performed data cleaning and preprocessing, and led exploratory data analysis (EDA).* |
| *Deeksha P* | *Conducted in-depth exploratory data analysis, created visualizations to identify trends and patterns, and helped interpret insights to guide modeling.* |
| *Srimathi B* | *Responsible for feature engineering, including feature creation, transformation, and selection to enhance model accuracy and efficiency.* |
| *Tamilarasan B* | *Carried out model evaluation by training multiple algorithms, comparing their performance, tuning hyperparameters, and selecting the best model.* |
| *Dinesh D* | *Led deployment efforts by developing and deploying the web application using Streamlit, ensuring a smooth user interface and real-time predictions.* |