***1. Understanding the Problem***

- Objective: Predict customer churn using the Telco Customer Churn dataset.

- Target Variable: Churn (binary classification - 1 for churn, 0 for no churn).

- Features: Various customer-related data such as demographics, service usage, and subscription details.

***2. Data Collection***

- Dataset: Telco Customer Churn dataset.

- Loading Data:

*import pandas as pd*

*df = pd.read\_csv('WA\_Fn-UseC\_-Telco-Customer-Churn.csv')*

*print(df.head())*

***3. Data Preprocessing***

***3.1 Handling Missing Values:***

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

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

***3.2 Encoding Categorical Variables:***

*df['Churn'] = df['Churn'].apply(lambda x: 1 if x == 'Yes' else 0)*

*df = pd.get\_dummies(df, columns=columns\_to\_encode, drop\_first=True)*

***3.3 Feature Scaling:***

*from sklearn.preprocessing import StandardScaler*

*scaler = StandardScaler()*

*df[['tenure', 'MonthlyCharges', 'TotalCharges']] = scaler.fit\_transform(df[['tenure', 'MonthlyCharges', 'TotalCharges']])*

***3.4 Feature Engineering:***

*df['TotalServices'] = df[['PhoneService\_Yes', 'MultipleLines\_Yes', 'InternetService\_Fiber optic',*

*'OnlineSecurity\_Yes', 'OnlineBackup\_Yes', 'DeviceProtection\_Yes',*

*'TechSupport\_Yes', 'StreamingTV\_Yes', 'StreamingMovies\_Yes']].sum(axis=1)*

***4. Exploratory Data Analysis (EDA)***

***4.1 Data Visualization:***

*import matplotlib.pyplot as plt*

*import seaborn as sns*

*df[numerical\_features].hist(bins=20, figsize=(14, 6))*

*plt.show()*

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

*plt.show()*

***4.2 Correlation Analysis:***

*corr\_matrix = df.corr()*

*sns.heatmap(corr\_matrix, annot=True, fmt='.2f', cmap='coolwarm')*

*plt.show()*

***4.3 Boxplots and Categorical Features Analysis:***

*for feature in numerical\_features:*

*sns.boxplot(x='Churn', y=feature, data=df)*

*plt.show()*

*for feature in categorical\_features:*

*sns.countplot(x=feature, hue='Churn', data=df)*

*plt.show()*

***5. Model Selection***

***5.1 Train-Test Split:***

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

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

*y = df['Churn']*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)*

***5.2 Model Initialization:***

*from sklearn.linear\_model import LogisticRegression*

*from sklearn.tree import DecisionTreeClassifier*

*from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier*

*import xgboost as xgb*

*models = {*

*'Logistic Regression': LogisticRegression(max\_iter=1000),*

*'Decision Tree': DecisionTreeClassifier(),*

*'Random Forest': RandomForestClassifier(),*

*'Gradient Boosting': GradientBoostingClassifier(),*

*'XGBoost': xgb.XGBClassifier(use\_label\_encoder=False, eval\_metric='logloss')*

*}*

***6. Model Training***

*import time*

*# Train each model*

*for name, model in models.items():*

*print(f"\nTraining {name}...")*

*start\_time = time.time() # Record the start time*

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

*end\_time = time.time() # Record the end time*

*training\_time = end\_time - start\_time # Calculate the training time*

*print(f"\n{name} training completed.")*

*print(f"Training time: {training\_time:.2f} seconds")*

***7. Model Evaluation***

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

*for name, model in models.items():*

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

*print(f"\n{name} Evaluation:")*

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

*print(classification\_report(y\_test, y\_pred))*

***8. Hyperparameter Tuning***

***8.1 Hyperparameter Tuning for Logistic Regression:***

*from sklearn.model\_selection import GridSearchCV*

*# Define hyperparameters for Logistic Regression*

*param\_grid = {*

*'C': [0.001, 0.01, 0.1, 1, 10, 100], # Regularization strength*

*'solver': ['liblinear', 'newton-cg', 'lbfgs', 'saga'] # Solver options*

*}*

*# Initialize GridSearchCV*

*grid\_search\_lr = GridSearchCV(LogisticRegression(max\_iter=1000), param\_grid, cv=5, scoring='accuracy', n\_jobs=-1)*

*# Fit the GridSearchCVThe*

*grid\_search\_lr.fit(X\_train, y\_train)*

*# Best parameters and best score*

*print("Best parameters for Logistic Regression:", grid\_search\_lr.best\_params\_)*

*print("Best score for Logistic Regression:", grid\_search\_lr.best\_score\_)*

***8.2 Hyperparameter Tuning for Gradient Boosting:***

*from sklearn.model\_selection import GridSearchCV*

*# Define hyperparameters for Gradient Boosting*

*param\_grid\_gb = {*

*'n\_estimators': [50, 100, 200], # Number of boosting stages*

*'learning\_rate': [0.01, 0.1, 0.2], # Learning rate*

*'max\_depth': [3, 4, 5], # Maximum depth of individual trees*

*}*

*# Initialize GridSearchCV*

*grid\_search\_gb = GridSearchCV(GradientBoostingClassifier(), param\_grid\_gb, cv=5, scoring='accuracy', n\_jobs=-1)*

*# Fit the GridSearchCV*

*grid\_search\_gb.fit(X\_train, y\_train)*

*# Best parameters and best score*

*print("Best parameters for Gradient Boosting:", grid\_search\_gb.best\_params\_)*

*print("Best score for Gradient Boosting:", grid\_search\_gb.best\_score\_)*

***8.3 Hyperparameter Tuning for SVM:***

*from sklearn.model\_selection import GridSearchCV*

*# Define hyperparameters for SVM*

*param\_grid\_svm = {*

*'C': [0.1, 1, 10], # Regularization parameter*

*'kernel': ['linear', 'rbf'], # Kernel type*

*'gamma': ['scale', 'auto'] # Kernel coefficient*

*}*

*# Initialize GridSearchCV*

*grid\_search\_svm = GridSearchCV(SVC(probability=True), param\_grid\_svm, cv=5, scoring='accuracy', n\_jobs=-1)*

*# Fit the GridSearchCV*

*grid\_search\_svm.fit(X\_train, y\_train)*

*# Best parameters and best score*

*print("Best parameters for SVM:", grid\_search\_svm.best\_params\_)*

*print("Best score for SVM:", grid\_search\_svm.best\_score\_)*

***9. Model Interpretation***

***9.1 Feature Importance for Gradient Boosting:***

*import matplotlib.pyplot as plt*

*# Train the best model from GridSearchCV for Gradient Boosting*

*best\_gb = grid\_search\_gb.best\_estimator\_*

*# Feature importance*

*importances = best\_gb.feature\_importances\_*

*indices = importances.argsort()[::-1]*

*# Plot feature importances*

*plt.figure(figsize=(12, 8))*

*plt.title('Feature Importances')*

*plt.bar(range(X\_train.shape[1]), importances[indices], align='center')*

*plt.xticks(range(X\_train.shape[1]), X\_train.columns[indices], rotation=90)*

*plt.xlim([-1, X\_train.shape[1]])*

*plt.show()*

*import matplotlib.pyplot as plt*

*# Fit the Gradient Boosting model with the best parameters*

*best\_gb\_model = grid\_search\_gb.best\_estimator\_*

*best\_gb\_model.fit(X\_train, y\_train)*

*# Get feature importances*

*importances = best\_gb\_model.feature\_importances\_*

*# Plot feature importances*

*plt.figure(figsize=(12, 8))*

*plt.barh(X\_train.columns, importances)*

*plt.xlabel('Feature Importance')*

*plt.title('Feature Importance for Gradient Boosting')*

*plt.show()*

***10. Implementation of Retention Strategy***

***10.1 Predict Churn for Test Data***

*# Predict probabilities of churn for the test set using the best Gradient Boosting model*

*y\_pred\_prob = best\_gb\_model.predict\_proba(X\_test)[:, 1]*

*# Add predictions to the original test set DataFrame*

*X\_test['Churn\_Probability'] = y\_pred\_prob*

*X\_test['Churn\_Prediction'] = (y\_pred\_prob >= 0.5).astype(int)*

*# Merge with customer IDs to keep track of who might churn*

*X\_test['customerID'] = df.loc[X\_test.index, 'customerID']*

*results = X\_test[['customerID', 'Churn\_Probability', 'Churn\_Prediction']]*

*# Display the customers most likely to churn*

*likely\_to\_churn = results[results['Churn\_Prediction'] == 1].sort\_values(by='Churn\_Probability', ascending=False)*

*print(likely\_to\_churn.head())*

***10.2: Develop a Retention Strategy***

*# Identify top features that contribute to churn*

*top\_features = X\_train.columns[indices[:5]] # Top 5 important features*

*# Based on these features, develop strategies*

*# Example: If 'Contract\_Two year' is important, offer discounts for two-year contracts.*

*for feature in top\_features:*

*print(f"Consider strategies that address {feature} to reduce churn.")*

*# For instance:*

*# 1. Offer discounts on long-term contracts if 'Contract\_Two year' is a key factor.*

*# 2. Improve service quality if 'TechSupport\_No' is significant.*

*# 3. Offer personalized deals based on the customer's usage patterns and preferences.*

***10.3: Provide Recommendations***

*# Example of providing actionable insights*

*likely\_to\_churn['Action'] = 'Offer a discount on the next bill' # Simplified example action*

*# Show a few examples*

*print(likely\_to\_churn[['customerID', 'Churn\_Probability', 'Action']].head(10))*

*import pickle*

*# Save the best model to a file*

*with open('best\_gb\_model.pkl', 'wb') as f:*

*pickle.dump(best\_gb\_model, f)*

***11. Model Deployment***

***11.1 Set Up a Simple Web API using Flask:***

*from flask import Flask, request, jsonify*

*import pickle*

*with open('best\_gb\_model.pkl', 'rb') as f:*

*model = pickle.load(f)*

*app = Flask(\_\_name\_\_)*

*@app.route('/predict', methods=['POST'])*

*def predict():*

*data = request.get\_json(force=True)*

*prediction = model.predict([data])*

*return jsonify({'prediction': prediction.tolist()})*

*if \_\_name\_\_ == '\_\_main\_\_':*

*app.run(debug=True)*

This outline gives you a structured and complete approach to solving the churn prediction problem, from understanding the problem and preprocessing data to model selection, training, evaluation, interpretation, and deployment.