**Project Title: Telecom Customer Churn Prediction**

**Objective:**

The objective is to build a machine learning model that predicts whether a customer will churn (leave the service) based on their usage patterns, demographic information, and past interaction history.

**Project Steps**

**1. Problem Definition and Data Collection**

* **Define the Problem**:
  + Predict customer churn to help the telecom company proactively retain customers.
  + Identify features such as customer demographics, account information, service usage, and billing history to inform the model.
* **Data Collection**:
  + Obtain historical data on customers, including:
    - **Customer Information**: Customer ID, demographic details, etc.
    - **Account Information**: Tenure, contract type, billing method, etc.
    - **Usage Patterns**: Data on minutes used, number of calls, SMS, data usage.
    - **Churn Label**: Indicates whether the customer has churned or not.
* **Data Source**:
  + The data can be obtained from internal databases (e.g., using SQL) or external sources if available.
  + Example sources might include customer transaction logs or subscription details stored in an Azure SQL Server or a CSV file.

**2. Data Exploration and Preprocessing (Python)**

* **Data Loading**:
  + Load the data using Python libraries like pandas.

import pandas as pd

# Load data

data = pd.read\_csv("telecom\_customer\_data.csv")

* **Exploratory Data Analysis (EDA)**:
  + Use Python (pandas, matplotlib, seaborn) to explore data distribution, spot missing values, and identify potential features.
  + Plot churn distribution to understand the proportion of churned vs. non-churned customers.

import seaborn as sns

import matplotlib.pyplot as plt

# Churn distribution

sns.countplot(x='churn', data=data)

plt.title('Churn Distribution')

plt.show()

* **Data Cleaning**:
  + Handle missing values (imputation or removal).
  + Convert categorical variables (e.g., contract type, payment method) into numerical format using one-hot encoding or label encoding.

data = pd.get\_dummies(data, columns=['contract\_type', 'payment\_method'], drop\_first=True)

**3. Feature Engineering**

* **Feature Selection**:
  + Use correlation analysis or feature importance scores to select relevant features.
  + Drop features that are highly correlated with each other to reduce redundancy.
* **New Feature Creation**:
  + Generate new features that might be useful, such as average\_monthly\_charges (total charges divided by tenure).
* **Scaling and Normalization**:
  + Use StandardScaler or MinMaxScaler to normalize continuous features.

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

continuous\_features = ['monthly\_charges', 'total\_charges']

data[continuous\_features] = scaler.fit\_transform(data[continuous\_features])

**4. Data Splitting**

* **Train-Test Split**:
  + Split the data into training and testing sets (e.g., 80/20 split).

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

X = data.drop(['churn'], axis=1)

y = data['churn']

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

**5. Model Selection and Training**

* **Choose Models**:
  + Try several machine learning models such as:
    - **Logistic Regression** (baseline model)
    - **Decision Trees**
    - **Random Forest**
    - **Gradient Boosting** (e.g., XGBoost or LightGBM)
* **Train Models**:
  + Train each model and tune hyperparameters using GridSearchCV or RandomizedSearchCV.

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import GridSearchCV

rf = RandomForestClassifier(random\_state=42)

param\_grid = {

'n\_estimators': [50, 100, 200],

'max\_depth': [10, 20, 30]

}

grid\_search = GridSearchCV(rf, param\_grid, cv=5)

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

**6. Model Evaluation**

* **Evaluate Models on Test Data**:
  + Use metrics like **accuracy**, **precision**, **recall**, **F1 score**, and **ROC-AUC** to evaluate performance.

from sklearn.metrics import accuracy\_score, classification\_report, roc\_auc\_score

y\_pred = grid\_search.best\_estimator\_.predict(X\_test)

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

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

print("ROC-AUC Score:", roc\_auc\_score(y\_test, grid\_search.best\_estimator\_.predict\_proba(X\_test)[:, 1]))

* **Select Best Model**:
  + Choose the model with the best performance across key metrics, focusing on recall if prioritizing correct churn predictions.

**7. Model Deployment**

* **Save the Model**:
  + Use joblib or pickle to save the model for later deployment.

import joblib

joblib.dump(grid\_search.best\_estimator\_, 'churn\_model.pkl')

* **Deploy the Model in a Web Application**:
  + Use Flask or FastAPI to create a simple API that serves predictions.
  + The API will load the model and take customer data as input to predict churn likelihood.

from flask import Flask, request, jsonify

import joblib

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

model = joblib.load('churn\_model.pkl')

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

def predict():

data = request.get\_json()

# Convert input data into DataFrame, then predict

prediction = model.predict(pd.DataFrame(data))

return jsonify({'churn': int(prediction[0])})

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

app.run()

**8. Monitoring and Maintenance**

* **Model Monitoring**:
  + Track model performance over time to ensure prediction accuracy.
  + Use monitoring tools or logs to check model metrics on live data, flagging model drift or significant changes.
* **Model Retraining**:
  + Regularly retrain the model with new data to improve accuracy and adapt to changes in customer behavior.

**Summary of Tools and Libraries Used**

1. **Data Storage and Collection**:
   * SQL or CSV files for data storage.
2. **Data Processing and Model Training (Python)**:
   * **Pandas**, **NumPy**: For data manipulation and cleaning.
   * **Scikit-learn**: For model training and evaluation.
   * **Matplotlib** and **Seaborn**: For data visualization.
3. **Model Deployment (Flask/FastAPI)**:
   * **Flask/FastAPI**: For creating an API to serve model predictions.
   * **Joblib**: For saving the model to be loaded in the web application.
4. **Monitoring and Maintenance**:
   * Track model performance using logs or monitoring tools like **Prometheus** or **Grafana** if hosted on the cloud.

**Resume Points**

1. **Data Collection and Exploration**:
   * Designed and implemented an end-to-end churn prediction model pipeline using Python, SQL, and machine learning, reducing churn rate predictions by identifying high-risk customers.
   * Conducted exploratory data analysis (EDA) using Python libraries (pandas, matplotlib, seaborn) to understand data patterns, identify data quality issues, and inform feature engineering.
   * Utilized SQL and Azure SQL Server for data extraction and transformation, optimizing data preparation processes to facilitate model training.
2. **Data Preprocessing and Feature Engineering**:
   * Applied data cleaning, encoding, and scaling techniques to preprocess data, ensuring high-quality inputs for model training and improving model interpretability.
   * Engineered key features such as customer tenure and average monthly charges to enhance predictive model accuracy, improving AUC scores by approximately 10%.
   * Leveraged correlation analysis and feature importance scores to identify and select high-impact features, optimizing model efficiency and interpretability.
3. **Model Training and Hyperparameter Tuning**:
   * Developed and trained multiple machine learning models (Logistic Regression, Decision Trees, Random Forests, and Gradient Boosting) to predict customer churn, achieving up to 85% accuracy and a 0.90 ROC-AUC score.
   * Conducted hyperparameter tuning using GridSearchCV and RandomizedSearchCV, enhancing model performance and minimizing false positives in churn predictions.
   * Leveraged Scikit-learn’s pipeline to streamline model training, testing, and tuning workflows, improving team efficiency in model development.
4. **Model Evaluation and Selection**:
   * Evaluated models using various metrics, including accuracy, precision, recall, F1 score, and ROC-AUC, ensuring optimal performance for real-world deployment.
   * Selected the best-performing model based on recall to prioritize minimizing churn false negatives, optimizing resource allocation for customer retention.
   * Validated model effectiveness using cross-validation techniques, ensuring robustness and minimizing overfitting in predictive accuracy.
5. **Model Deployment**:
   * Deployed the final model as an API endpoint using Flask, enabling real-time predictions to assist business teams in proactive churn management.
   * Integrated the model into a Flask-based web application, allowing cross-functional teams to easily access churn predictions.
   * Saved and managed model versions using joblib, ensuring seamless deployment and consistent performance in production environments.
6. **Monitoring and Maintenance**:
   * Implemented monitoring mechanisms to track model performance metrics over time, identifying drift and retraining requirements as customer behavior evolved.
   * Developed a retraining strategy using new customer data, ensuring the model remains relevant and accurate over time.
   * Established a feedback loop from business teams to incorporate real-world insights and continuously improve model predictions.