Customer Churn Prediction API

1. **Project Title:** Predictive Customer Churn API Service
2. **Project Goal/Objective:** To develop and deploy a machine learning model that predicts the likelihood of a customer churning (discontinuing service), exposed as a REST API for integration into hypothetical business systems.
3. **Problem Statement/Background:** Customer retention is critical for business sustainability. Proactively identifying customers at high risk of churning allows businesses to intervene with targeted retention strategies (e.g., special offers, support outreach), reducing revenue loss and acquisition costs.
4. **Scope:**
    - **In Scope:** Sourcing relevant (likely tabular) customer data; data preprocessing and feature engineering; training and evaluating various classification models (e.g., Logistic Regression, Random Forest, Gradient Boosting); selecting the best performing model; building a REST API endpoint that accepts customer data (e.g., JSON payload) and returns a churn probability score; containerizing the application; deploying the containerized API to a cloud platform.
    - **Out of Scope:** Building a front-end user interface for the API (focus is on the backend service); real-time integration with live CRM systems; implementing the actual customer intervention strategies.
5. **Key Deliverables:**
    - Cleaned dataset and feature engineering code.
    - Trained ML model(s) for churn prediction (code and saved model files).
    - Documented API specification (e.g., OpenAPI/Swagger).
    - A containerized (Docker) REST API application.
    - Deployed API endpoint accessible via URL.
    - Final technical report covering data, modeling, API design, deployment process, and performance evaluation.
    - Final presentation demonstrating API functionality and explaining the project.
6. **Potential Datasets to Get Started:**
    - **Telco Customer Churn Datasets (Kaggle/IBM):** Several popular datasets exist representing fictional telecom customers with demographic info, service usage, and churn status. These are well-suited for standard classification tasks.
        - *Link (IBM sample on Kaggle):* https://www.kaggle.com/datasets/blastchar/telco-customer-churn
        - *Link (Another Kaggle Telco set):* https://www.kaggle.com/datasets/mnassrib/telecom-churn-datasets

    - **Simulated Data:** Students could also generate synthetic data if specific features relevant to a chosen industry (e.g., SaaS, e-commerce subscription) are desired.
7. **Potential Technical Approach:**
    - **ML Models:** Use libraries like Scikit-learn, XGBoost, LightGBM. Focus on model evaluation (Accuracy, Precision, Recall, F1-score, ROC AUC) and interpretation (feature importance).
    - **API Framework:** Python frameworks like Flask or FastAPI are excellent choices for building the REST API.
    - **Containerization:** Use Docker to package the application, model, and dependencies. Create a `Dockerfile`.
    - **Cloud Platform:** AWS (e.g., SageMaker endpoint, Lambda + API Gateway, Elastic Beanstalk, Fargate), GCP (e.g., AI Platform Prediction, Cloud Run, App Engine), or Azure (e.g., Azure ML Endpoints, Azure Functions, App Service).
8. **Deployment Requirements & Tips:**
    - **Requirement:** The final model must be deployed as a containerized REST API endpoint on a chosen cloud platform (AWS, GCP, or Azure). The endpoint URL must be provided and functional for demonstration.
    - **Tips for Students:**
        - **Start Simple:** Build and test the API locally first (Flask/FastAPI). Ensure it works with sample requests (e.g., using `curl` or Postman).
        - **Containerize Early:** Create the `Dockerfile` and test building/running the container locally. This helps catch dependency issues early. Ensure the model file is included in the container.
        - **Choose the Right Cloud Service:**
            - **Serverless Functions (Lambda, Azure Functions, Cloud Functions):** Good for simple APIs, pay-per-use, but might have cold starts or limitations on package size/runtime. Often paired with an API Gateway service.
            - **Container Orchestration (Cloud Run, Fargate, AKS/EKS/GKE):** More flexible, directly run containers. Cloud Run (GCP) or Azure Container Apps are often good starting points with simpler configuration than full Kubernetes.
            - **ML Platform Endpoints (SageMaker, Azure ML, GCP AI Platform):** Specifically designed for ML model deployment, handle scaling, monitoring, potentially A/B testing, but might have a steeper learning curve initially.

    - **Manage Dependencies:** Use a `requirements.txt` file for Python packages. Ensure all necessary libraries (including ML frameworks and the API framework) are listed.
    - **Cloud Credentials & Permissions:** Setting up access correctly (IAM roles/policies) is crucial and often tricky. Follow cloud provider documentation carefully. Use environment variables for sensitive keys, don't hardcode them.
    - **Logging & Monitoring:** Implement basic logging in the API to help debug issues once deployed. Cloud platforms offer monitoring tools.
    - **Resource:** Leverage the cloud provider's documentation, quickstart guides, and tutorials specific to deploying Python web applications or ML models. Utilize the resources from the `Cloud Computing for Machine Learning` course.
9. **Potential Challenges:** Data preprocessing complexities; achieving good model performance; debugging deployment issues (dependencies, permissions, networking); managing cloud costs (use free tiers where possible); API security basics.
10. **Success Metrics:** Deployed API functionality and reliability; predictive performance of the ML model; quality of code and documentation; adherence to deployment requirements.
11. **Team Size / Duration:** 3-4 Students / 14 Weeks