Test Plan

SYSTEM TEST PLAN

---

1. Test Plan Identifier
CCP-TP-001 | v1.0 | Date: 2025-04-19

---

2. Introduction
This document outlines the test plan for the **Customer Churn Predictor**, a web application using both Machine Learning and Deep Learning models to predict the likelihood of customer churn based on historical and behavioral data.
The primary objective of this test plan is to validate the backend components, including data preprocessing pipelines, model loading and inference accuracy, and system response to various input scenarios. Emphasis is placed on ensuring that the backend logic functions reliably under different conditions, and that model outputs remain consistent and interpretable.

---

3. Test Items
The components subject to testing include:

- **Web application interface**
- **Catboost.cbm (ML model)**
- **Lightgbm.txt (ML model)**
- **Random_forest.pkl (ML model)**
- **Xgboost.json (ML model)**
- **Node.pt (DL model)**
- **Saint.pt (DL model)**
- **Tabnet.pt (DL model)**
- **Tabtransformer.pt (DL model)**
- **App.py (to host flask application)**
- **Data preprocessing logic**
- **User input forms (age, gender, country, etc.)**
- **Final result generation and display logic**

---

4. Features to be Tested

The system will be tested for the following features:

Feature	Description
User Input Handling	Validation and capturing of all required customer input data from the frontend
Data Preprocessing	Verification of feature encoding, normalization, and missing value handling
Model Inference	Consistent behavior of both ML and DNN models for churn and non-churn cases
Model Integration	Correct loading and switching between multiple models
Backend Logic	Validation of churn probability thresholds, business logic, and decision rules
Output Generation	Accurate JSON or structured output including churn prediction and probability
Error Handling	Proper handling of invalid inputs, model loading failures, or empty fields
Logging & Debugging	Backend logs should capture relevant events and errors clearly

---

5. Features Not to be Tested

The following features are out of scope for this testing process:

- **Business accuracy of churn prediction outcomes**
  Evaluation of real-world business impact or deployment-readiness is not within the scope of this test.
- **Frontend styling and UI responsiveness**
  Testing of layout, CSS design, and mobile responsiveness is excluded, as the focus is on backend logic.
- **Database interactions or long-term data storage**
  Persistent storage mechanisms such as saving input or results to a database are not part of the current version.
- **Scalability and load testing**
  Performance under heavy traffic or simultaneous user requests is not critical for this prototype phase.
- **Security and authentication**
  User authentication, authorization, and data security are not implemented in this minimal viable product (MVP).

---

6. Approach

The system follows a backend-focused web architecture, and the testing approach consists of:

- **Unit Testing**
  Core logic such as data preprocessing, encoding, and model inference will be tested in isolation using Python unit tests.
- **Manual Testing**
  HTML forms and user flows will be manually tested to ensure accurate input capture and expected behavior across the Flask routes.
- **Integration Testing**
  Verifies that the ML and Deep Learning models are properly loaded by the Flask backend, and predictions are correctly processed and returned to the frontend.
- **Smoke Testing**
  Key backend functionalities like model loading, data processing, and prediction generation will be smoke-tested at the end of each sprint.
- **Acceptance Testing**
  A set of predefined customer profiles will be used to validate the system's ability to predict churn outcomes reliably and interpretably.

---

7. Item Pass/Fail Criteria

Criteria	Condition
Pass	All mandatory form fields are validated and result in correct predictions from both ML and DNN models
Pass	Input data is correctly preprocessed, encoded, and passed to models
Fail	Model returns incorrect or inconsistent prediction for predefined input cases
Fail	Invalid or incomplete input is accepted and processed without warning
Fail	Prediction results are not correctly displayed on the HTML page
Fail	Backend fails to load one or both models during form submission

---

8. Suspension Criteria and Resumption Requirements

Suspension	Reason
Model not loading	Missing or corrupted .pt or .txt file
Environment error	Dependency conflicts or version mismatch
Form processing failure	HTML form data not correctly reaching the backend, breaking input-to-output flow
API route failure	Flask routes responsible for prediction or rendering results fail due to misconfiguration or exceptions

Resumption: Issue must be resolved, tested in isolation, and regression tested before continuing.

---

9. Test Deliverables

- Test Plan Document
- Test Cases and Scenarios
- Bug Report and Logs
- Final Test Summary Report
- Model Validation Results
- Screen Recording/Walkthrough for demo testing

---

10. Test Tasks

- Create and validate unit test cases
- Manual UI and flow testing
- Prepare known input-output datasets
- Validate prediction correctness (Churn vs No Churn)
- Document bugs and anomalies
- Regression test after changes

---

11. Environmental Needs

Component	Requirement
OS	Windows/Linux/macOS
Browser	Chrome/Edge/Firefox
Python	3.8+
Flask	1.1.2+ (for backend routing and serving the app)
Local or cloud instance	Can run locally or deployed to cloud platforms (e.g., Heroku, AWS) for web access

12. Responsibilities

Role	Responsibility
Developer/ ML Engineer	Build and maintain the Flask app, implement ML/DNN models, integrate backend components, preprocess data, and fix bugs in the system
QA/Tester	Test backend logic, model predictions, data flow, and API routes; design and execute test cases; validate input/output behavior; report issues

---

13. Schedule

Task	Start Date	End Date
Requirement Gathering & Planning	2025-04-04	2025-04-05
Data Analysis and Preprocessing	2025-04-06	2025-04-07
ML/DNN Model Development	2025-04-08	2025-04-11
Initial Flask App Setup	2025-04-10	2025-04-12
Integration and Testing	2025-04-13	2025-04-14
UI Enhancement and Bug Fixes	2025-04-15	2025-04-16
Final Validation	2025-04-17	2025-04-18
Project Simulation and Demo	2025-04-19	2025-04-20

---

15. Risks and Contingencies

Risk	Contingency
Model not predicting churn cases	Validate input data with historical cases and retrain models if necessary
UI not capturing all fields	Implement mandatory field validation with default error messages and user guidance
Incomplete dependencies	Use requirements.txt and Docker if needed
Inconsistent	Standardize feature preprocessing and debug input pipeline

---

16. Approvals

The following stakeholders must approve this test plan:

- **Customer**
- **Developers**
- **Testers**
- **DevOps Engineer**
- **Project Manager**
- **Scrum Master**
- **Senior Management Team**
