Product Requirements and Specification Document

Project Name

TitanicTree - Survival Prediction Web App

Description

TitanicTree is a Flask-based web application that predicts Titanic passenger survival using a Decision Tree model. Users can upload passenger data, view model predictions, visualize the decision tree, and review evaluation metrics. The app is containerized with Docker for straightforward deployment.

---

1. Goals & Objectives

Goal	Description
Predict Survival	Allow users to predict survival of Titanic passengers
Educational Visualization	Visualize the decision tree for learning purposes
Model Evaluation	Display key evaluation metrics (accuracy, precision, recall)
Easy Deployment	Provide Docker support for simple hosting and portability

---

2. Functional Requirements

ID	Requirement
FR1	Users can upload CSV files with passenger data
FR2	Application preprocesses and validates uploaded data
FR3	Predict survival outcomes using a trained Decision Tree model
FR4	Display predictions in a tabular format
FR5	Visualize the trained Decision Tree structure
FR6	Show evaluation metrics: accuracy, precision, recall, F1-score
FR7	Provide sample data for demonstration
FR8	Dockerfile and instructions for containerized deployment

---

3. Non-Functional Requirements

ID	Requirement
NFR1	Responsive and intuitive web UI
NFR2	Secure file upload (size/type validation)

NFR3	Fast prediction and visualization (<2s)
NFR4	Codebase follows PEP8 and best practices
NFR5	Documentation for setup and usage

---

4. User Stories

As a…	I want to…	So that…
Student	Upload Titanic data and get predictions	I can learn about ML and survival factors
Educator	Visualize the decision tree	I can explain model decisions to students
Developer	Deploy the app easily	I can use it in different environments

---

5. Technical Specifications

5.1 Architecture

- **Frontend:** Flask templates (Jinja2), Bootstrap (optional)
- **Backend:** Flask, scikit-learn, pandas, numpy
- **Model:** DecisionTreeClassifier (scikit-learn)
- **Containerization:** Docker

5.2 Data Flow

```
graph TD
A[User Uploads CSV] --> B[Flask Backend]
B --> C[Data Preprocessing]
C --> D[Model Prediction]
D --> E[Display Results & Metrics]
D --> F[Decision Tree Visualization]
```

5.3 Key Endpoints

Endpoint	Method	Description
`/`	GET	Home page, upload form, sample data
`/predict`	POST	Handle file upload, return predictions
`/visualize`	GET	Show decision tree visualization
`/metrics`	GET	Display model evaluation metrics

5.4 Model Training

- Use Titanic dataset (public Kaggle version)
- Preprocess: handle missing values, encode categorical features
- Train DecisionTreeClassifier
- Save model as a serialized file (`.pkl`)

5.5 Visualization

- Use `sklearn.tree.plot_tree` or export as image
- Display in web UI

---

6. UI/UX Requirements

Page/Section	Elements
Home	App description, upload form, sample data link
Prediction Results	Table: input data + predicted survival
Visualization	Image/graph of decision tree
Metrics	Table: accuracy, precision, recall, F1-score
Error Handling	User-friendly error messages for invalid uploads

---

7. Deployment

- **Docker:** Provide `Dockerfile` and `docker-compose.yml` (if needed)
- **Instructions:** README with setup, usage, and deployment steps

---

8. Acceptance Criteria

Criteria	Status
Upload, predict, and display results	
Decision tree visualization is accessible	
Evaluation metrics are shown	
Docker deployment works as documented	
All requirements above are met	

---

9. Out of Scope

- User authentication
- Real-time model retraining
- Advanced model selection/tuning

---

10. Appendix

Technologies

Technology	Purpose

Python	Core language
Flask	Web framework
scikit-learn	ML model and visualization
pandas, numpy	Data processing
Docker	Containerization

End of Document