Titanic Pro Systems

This project is a comprehensive Machine Learning and Data Engineering application designed to predict passenger survival probabilities on the Titanic. Going beyond simple static modelling, the system integrates a full-stack data pipeline that combines historical data analysis, interactive AI predictions, and real-time streaming capabilities.

The core technology stack includes **XG Boost** for high-performance classification, **Stream lit** for the interactive web interface, and **Apache Kafka** (containerized via Docker) to handle real-time data ingestion. The application features a "Neon Deep-Ocean" themed UI, offering users three distinct modules: an Analytics Dashboard for historical insights, a Neural Survival Predictor for custom scenarios, and a Live Kafka Stream for processing passenger data in real-time.

---

ii. A Walkthrough of the EDA, Model Building, and Final Predictions

The project utilizes a robust machine learning pipeline defined in train_model.py to process raw data and generate accurate predictions.

- **Exploratory Data Analysis (EDA):** The system begins by loading the training dataset (train.csv). The analysis phase is visualized in the Stream lit application, which calculates key metrics such as total passenger count, overall survival rates, average fares, and age distribution.
- **Feature Engineering & Preprocessing:** Before training, the raw data undergoes significant transformation:
    - **Title Extraction:** Passenger titles (e.g., Mr., Mrs., Dr.) are extracted from names to categorize social status.
    - **Family Grouping:** A Family Size feature is created by combining siblings (SibSp) and parents/children (Parch), alongside an Is Alone binary flag.
    - **Handling Missing Data:** The system imputes missing values using median strategies for numerical data and "most frequent" strategies for categorical data.
- **Model Building:** The prediction engine relies on an **XG Boost Classifier** (XGB Classifier) with 100 estimators. This model is wrapped in a Scikit-Learn pipeline that automatically handles scaling (Standard Scaler) and One-Hot Encoding for categorical variables, ensuring the model is robust and production-ready.

---

iii. Demonstration of Filtering and Sorting Functionality via the Stream lit UI

The user interface acts as a dynamic control centre, allowing users to "filter" inputs and view "sorted" analytical insights through two main views in app.py:

- **Interactive Parameter Tuning (Filtering):** In the "Neural Survival Predictor" module, users can filter the model's input by manually adjusting specific passenger parameters. The sidebar and main expandable forms allow the selection of:
    - **Class:** 1st, 2nd, or 3rd Class.
    - **Demographics:** Gender and Age (via a slider).

- **Economic Status:** Fare price.

Upon clicking "RUN PREDICTION," the system filters these specific inputs through the trained XG Boost model to output a precise survival probability score displayed on a gauge chart.

- **Visual Data Sorting:** The "Analytics Dashboard" automatically sorts and groups historical data for quick interpretation. For example, it displays a "Survival by Class" bar chart (sorting survival rates by Pclass) and an "Age Distribution" histogram, allowing users to visually inspect how different groups fared compared to one another.

---

iv. Demonstration of the Real-Time Prediction System Using Kafka

The project simulates a modern production environment where passenger data is received continuously rather than in static batches.

- **Infrastructure:** The streaming architecture relies on **Apache Kafka** and **Zookeeper**, orchestrated via docker-compose.yml to ensure a stable message broker service.
- **The Data Producer:** A standalone script, producer.py, reads raw passenger data from test.csv and serializes it into JSON format. It acts as a live data source, pushing individual passenger records to the titanic_stream Kafka topic at one-second intervals to mimic real-time events.
- **The Stream lit Consumer:** The "Live Kafka Stream" page in the main application connects to this topic using the confluent_kafka library. As new passenger records arrive:
    1. The app deserializes the JSON data.
    2. It extracts features on the fly (e.g., calculating title and family size dynamically).
    3. The pre-trained XG Boost model predicts the survival status ("Alive" vs "Deceased") instantly.
    4. The results are logged to a live-updating table on the dashboard, showing the Passenger Name, Predicted Status, and Model Confidence score.