DPP-Enabled Reverse Logistics Prototype — Project Documentation

This document explains your folder structure, each file's role, and what happens "behind the scenes" when you run the app.

1) Project Structure

Recommended folder layout:

```
dpp-rl-dashboard/
■■■ data/
■   ■■■ washing_machine_returns.csv
■■■ src/
■   ■■■ kpi_engine.py
■   ■■■ decision_engine.py
■■■ app.py
■■■ requirements.txt
```

2) What Each File Does

2.1 data/washing_machine_returns.csv

Role: This is your mock Digital Product Passport (DPP)-like dataset. Each row represents one returned washing machine. It includes product condition, DPP-like attributes (repairability, disassembly time, CO■ footprint), market values, and an info_completeness field that represents how complete/structured the product information is (0–1).

2.2 src/kpi_engine.py

Role: The KPI Engine (the "calculator"). It takes the raw dataset and creates new columns that represent economic and ecological KPIs. This file should contain only reusable calculation functions (no Streamlit UI code).

What it computes in the starter version:

```
• disassembly_hours  = disassembly_time_min / 60
• disassembly_cost   = disassembly_hours * LABOR_RATE
• processing_cost    = PROCESSING_ENERGY * ENERGY_COST
• net_value_reman    = market_value_reman - disassembly_cost - processing_cost
• avoided_co2        = co2_mfg_kg * 0.8   (simple substitution assumption)
```

Behind the scenes: app.py imports calculate_kpis(df), passes the dataframe to it, and receives a modified dataframe back with extra KPI columns. Later you can expand this to include transport cost, inspection cost, recycling yields, avoided virgin material, etc.

2.3 src/decision_engine.py

Role: The Decision Engine (the "recommender"). It chooses the best End-of-Life option per return (e.g., Remanufacture, Reuse, Recycle) using a transparent scoring model. This is where you operationalize your thesis idea: better product data (higher info_completeness) reduces decision risk and improves outcomes.

Starter logic (simplified):

```
score = 0.6 * net_value_reman + 0.4 * avoided_co2 - (1 - info_completeness) * 50

if score > 150:  Remanufacture
elif score > 80: Reuse
else:            Recycle
```

Behind the scenes: app.py applies recommend_option(row) to each row. The returned label is stored in a new column recommended_option. Later you can add: feasibility gates (hazards/parts missing), normalized scoring, explainability (top drivers), and scenario toggles.

2.4 app.py

Role: The Streamlit Application (the “front-end”). It loads the dataset, calls the KPI Engine, calls the Decision Engine, and displays everything in a dashboard.

Flow inside app.py:

```
1) Import libraries and your modules (kpi_engine, decision_engine)
2) Read CSV from data/
3) df = calculate_kpis(df)             # add KPI columns
4) df["recommended_option"] = ...      # add decision recommendation
5) Display table and KPI summary cards in Streamlit
```

Behind the scenes: when you run 'streamlit run app.py', Streamlit executes app.py from top to bottom to build the page. Every time you interact (filters/toggles), Streamlit reruns the script to update the outputs. This is why we keep heavy logic in src/ modules—so the app stays clean and maintainable.

2.5 requirements.txt

Role: A reproducible dependency list. It lets any recruiter/company run your project with one command. Example:

```
pandas
numpy
streamlit
plotly
```

3) How the Whole System Works (End-to-End)

Think of your prototype as a mini “data pipeline + decision support tool”. Here is the end-to-end chain:

```
A) Input (DPP-like data)
   washing_machine_returns.csv

B) Processing (Python modules)
   KPI Engine       -> adds economic + ecological KPI columns
   Decision Engine  -> selects best EoL option per return

C) Output (Streamlit UI)
   • Table: each return + KPIs + recommendation
   • Metrics: average net value, total avoided CO■, etc.
   • Later: charts + scenario (With vs Without DPP)
```

4) Recommended Next Enhancements (Short List)

1) Add 'With DPP vs Without DPP' toggle: artificially reduce info_completeness and/or blank out fields in the 'Without' scenario. 2) Add explainability: store 2–3 reasons for the recommendation per row. 3) Add Plotly charts: distribution of recommended options, net value vs repairability, avoided CO■ totals. 4) Add exports: allow saving the processed dataframe (CSV) for Power BI. 5) Improve scoring: normalize econ_score and eco_score separately before weighting.