📘 Detailed Guide: Claims Optimization System

This document explains **every file** in the project, its **functionality**, and the **step-by-step process** of building and running the Claims Optimization System.

📂 Project Structure

```
claims-optimization/
├─ app.py                      # Main Streamlit app (UI & dashboards)
├─ train.py                    # ML pipeline: preprocess + train + save model
├─ generate_sample_data.py     # Creates synthetic claim data for testing
├─ requirements.txt            # Dependencies list
├─ Dockerfile (optional)       # Containerize the app
├─ data/
│  └─ claims.csv               # Dataset (real or synthetic)
├─ models/
│  └─ claim_approval.joblib    # Saved trained model
└─ logs/ (optional)            # Audit logs of predictions
```

🗂️ File-by-File Explanation

1. `requirements.txt`

- Contains the Python dependencies.
- Makes it easy to recreate the environment.
- Key libraries:
- **pandas, numpy** → data processing.
- **scikit-learn** → ML model training & preprocessing.
- **joblib** → saves the model.
- **streamlit** → web interface.
- **altair** → charts for dashboards.
- **pyarrow** → handles Parquet/CSV efficiently.

👉Install with:

```
pip install -r requirements.txt
```

---

2. `generate_sample_data.py`

- Creates a **synthetic dataset** (claims.csv) for testing.
- Uses random number generation to simulate insurance claims.
- Columns:
- claim_id, claim_amount, patient_age, patient_gender, insurance_provider, diagnosis_code, procedure_code, in_network, preauth_obtained, prior_denials_count, days_since_service, status, denial_reason
- Saves file at `data/claims.csv`.

👉Run with:

```
python generate_sample_data.py
```

---

3. `train.py`

- **Core ML pipeline**:
- Loads `data/claims.csv`.
- Splits into **features (X)** and **target (y)**.
  - Target = `status` (Approved = 1, Denied = 0).
- Defines **feature types**:
  - Numeric → claim_amount, age, etc.
  - Boolean → in_network, preauth_obtained.
  - Categorical → gender, provider, diagnosis, procedure.
- Uses **ColumnTransformer**:
  - Numeric → impute median.
  - Boolean → passthrough.
  - Categorical → impute most frequent + OneHotEncoder.
- Trains **Logistic Regression** with class balancing.
- Wraps in **CalibratedClassifierCV** → improves probability outputs.
- Evaluates model:
  - ROC-AUC, PR-AUC, precision/recall.
- Saves model to `models/claim_approval.joblib` with metadata (features, threshold).

👉Run with:

```
python train.py
```

---

4. `app.py`

- **Streamlit UI application**.
- Two main tabs:

📊**Dashboards**

- Metrics:
- Total claims
- Approved vs Denied counts
- Average claim amount
- Charts:
- Status distribution (Approved/Denied)
- Claim amount histogram
- Insurance provider trends
- Top denial reasons
- Option to **upload a CSV** (instead of using default `data/claims.csv`).

🧮**Predictor**

- **Single Claim Form** → enter details and get prediction (Approved/Denied + probability).
- **Batch Scoring** → upload CSV, app predicts for all rows, adds:
- `p_approved` (probability)
- `predicted_status` (Approved/Denied)
- Option to **download results** as a new CSV.

👉Run with:

```
streamlit run app.py
```

Open browser at `http://localhost:8501`

---

5. `Dockerfile` (optional)

- Lets you containerize the app for deployment.
- Uses Python slim image.
- Installs dependencies.
- Runs `streamlit run app.py` inside container.

👉Build & run with:

```
docker build -t claims-optimization .
docker run -p 8501:8501 claims-optimization
```

---

6. `data/claims.csv`

- The dataset (real or synthetic).
- Used both for **training** and for **dashboards**.

- Format: one row per claim.

7. `models/claim_approval.joblib`

- Trained ML pipeline.
- Contains:
- Preprocessing steps.
- Trained classifier.
- Calibration wrapper.
- Metadata (features, threshold).

8. `logs/` (optional)

- Can store audit logs.
- Each prediction = claim_id, probability, decision, timestamp.
- Useful for compliance in healthcare.

⚙️Step-by-Step Process (Building the App)

1. **Set up project**

```
mkdir claims-optimization && cd claims-optimization
python -m venv .venv
# activate venv (Windows)
.venv\Scripts\activate
# or (Linux/Mac)
source .venv/bin/activate
```

2. **Create requirements.txt** (as shown above) and install deps:

```
pip install -r requirements.txt
```

3. **Prepare dataset**

4. Option A: Run `generate_sample_data.py` to create synthetic claims.

5. Option B: Place your real dataset at `data/claims.csv`.

6. **Train model**

```
python train.py
```

7. Saves `models/claim_approval.joblib`.

8. **Run app**

```
streamlit run app.py
```

9. Open in browser → view dashboards + make predictions.

10. **Test with other CSVs**

11. Upload in sidebar for dashboards.
12. Upload in **Batch Scoring** for predictions.

13. Download results as CSV.

14. **(Optional) Deployment**

15. Build Docker image.
16. Deploy to cloud (Streamlit Community Cloud, AWS, GCP, Azure).

🔗 Summary

- `generate_sample_data.py` → creates test data.
- `train.py` → builds + saves ML model.
- `app.py` → interactive dashboards + prediction interface.
- `requirements.txt` → dependencies.
- `data/` → CSV datasets.
- `models/` → saved ML models.
- `Dockerfile` → deployment option.

This setup gives you a **complete end-to-end system** for healthcare claim prediction and fraud/denial insights.