Hospital Readmission ML pipeline — Backend, Frontend, and Multi-model Training

1) Project structure (recommended)

```
hospital-readmit/
├─ data/                # put your CSV(s) here (e.g. patients_30000.csv)
├─ configs/
│  └─ config.yaml
├─ src/
│  ├─ data_loader.py
│  ├─ preprocess.py
│  ├─ features.py
│  ├─ train_models.py
│  ├─ tune_optuna.py
│  ├─ stacking.py
│  ├─ serve/
│  │  └─ api.py           # FastAPI app
│  ├─ explainability.py   # SHAP helpers
│  └─ utils.py
├─ mlflow/                # mlflow server + backend store (optional)
├─ docker/
│  ├─ Dockerfile
│  └─ docker-compose.yml
├─ frontend/
│  └─ app/                # React app scaffold (example component provided)
└─ README.md
```

---

2) Quick start commands (CLI)

```
# 1) Create venv and install
python -m venv venv
source venv/bin/activate
pip install -r requirements.txt

# 2) Train all models (defaults read configs/config.yaml)
python src/train_models.py --config configs/config.yaml

# 3) Start MLflow UI
mlflow ui --port 5000

# 4) Start API (after model saved)
uvicorn src.serve.api:app --host 0.0.0.0 --port 8000

# 5) Start frontend (frontend/app)
cd frontend/app
npm install
npm start
```

---

3) Minimal `configs/config.yaml` (example)

```yaml
data_path: data/patients_30000.csv
target_col: readmitted_30d
id_col: patient_id
```

```
models:
  - name: logistic_regression
  - name: random_forest
  - name: xgboost
  - name: lightgbm
  - name: catboost
  - name: simple_nn
training:
  n_splits: 5
  random_state: 42
  optuna_trials: 60
mlflow:
  tracking_uri: http://localhost:5000
  experiment_name: hospital_readmit
```

---

4) core code highlights (short excerpts)

`data_loader.py` (load & quick checks)

```python
import pandas as pd

def load_data(path):
    df = pd.read_csv(path)
    # quick sanity
    print(df.shape)
    print(df.isnull().mean().sort_values(ascending=False).head())
    return df
```

`preprocess.py` (impute / encode / feature engineering)

```python
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
import category_encoders as ce

# 1. fill missings
imputer = SimpleImputer(strategy='median')
# 2. target encoding for high-cardinality categories
te = ce.TargetEncoder(cols=['diagnosis_code','hospital_id'])
# 3. scaling numeric features
scaler = StandardScaler()
```

`train_models.py` (train multiple models + CV + MLflow)

```python
import mlflow
import joblib
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import roc_auc_score, average_precision_score

# sample model dict
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

models = {
```

```python
  'logistic': LogisticRegression(max_iter=1000),
  'rf': RandomForestClassifier(n_estimators=200, n_jobs=-1),
  'xgb': XGBClassifier(use_label_encoder=False, eval_metric='logloss', n_jobs=-1),
  'lgbm': LGBMClassifier(n_jobs=-1),
  'cat': CatBoostClassifier(verbose=0)
}

# training loop (sketch)
for name, model in models.items():
    scores = []
    skf = StratifiedKFold(n_splits=cfg['training']['n_splits'], shuffle=True, random_state=cfg['training']['random_state'])
    for train_idx, val_idx in skf.split(X, y):
        model.fit(X[train_idx], y[train_idx])
        preds = model.predict_proba(X[val_idx])[:,1]
        scores.append(roc_auc_score(y[val_idx], preds))
    mean_auc = sum(scores)/len(scores)
    mlflow.log_metric(f"{name}_cv_auc", mean_auc)
    # fit on full data and save
    model.fit(X, y)
    artifact = f"models/{name}.pkl"
    joblib.dump(model, artifact)
    mlflow.log_artifact(artifact)
```

`tune_optuna.py` (Optuna example for LightGBM)

```python
import optuna
from lightgbm import LGBMClassifier

def objective(trial):
    params = {
      'num_leaves': trial.suggest_int('num_leaves', 20, 200),
      'learning_rate': trial.suggest_float('learning_rate', 1e-3, 0.3, log=True),
      'n_estimators': trial.suggest_int('n_estimators', 100, 2000)
    }
    model = LGBMClassifier(**params)
    # cross-validate and return mean AUC
    return cv_mean_auc(model, X, y)

study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=cfg['training']['optuna_trials'])
print(study.best_params)
```

`stacking.py` (simple stacking meta-learner)

```python
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
estimators = [(n, models[n]) for n in ['xgb','lgbm','cat']]
meta = LogisticRegression()
stack = StackingClassifier(estimators=estimators, final_estimator=meta, cv=5, n_jobs=-1)
stack.fit(X,y)
joblib.dump(stack, 'models/stack.pkl')
```

`serve/api.py` (FastAPI minimal)

```python
```

```python
from fastapi import FastAPI
import joblib
import pandas as pd
from pydantic import BaseModel

app = FastAPI()

class Patient(BaseModel):
    patient_id: int
    age: int
    diagnosis_code: str
    # add other fields

model = joblib.load('models/stack.pkl')

@app.post('/predict')
def predict(p: Patient):
    df = pd.DataFrame([p.dict()])
    proba = model.predict_proba(df)[:,1].tolist()[0]
    return {'patient_id': p.patient_id, 'risk_30d': float(proba)}
```

`explainability.py` (SHAP explain endpoint helper)

```python
import shap
explainer = shap.Explainer(model.predict_proba, X_sample)
shap_values = explainer(X_patient)
# return top 10 contributing features
```

---

5) Frontend sketch (React) — clinician dashboard

* Show list of patients, risk score (0.0–1.0), top contributing features (SHAP) and suggested actions (e.g., follow-up call, meds reconciliation, home visit).
* Use one main component that calls `/predict` for one patient or batch endpoint `/predict_batch` for lists.

Minimal example component (React)

```jsx
export default function PatientCard({patient}){
  const [risk, setRisk] = useState(null);
  useEffect(()=>{
    fetch('/api/predict', {method:'POST', body: JSON.stringify(patient)})
      .then(r=>r.json()).then(j=>setRisk(j.risk_30d))
  },[])
  return (
    <div className='card'>
      <h3>{patient.patient_id} — Risk {risk && (risk*100).toFixed(1)}%</h3>
    </div>
  )
}
```

---

6) Best-practices & performance tips (implementation checklist)

* Use StratifiedKFold, and report AUC-ROC, PR-AUC, sensitivity at fixed specificity thresholds.
* For imbalanced outcomes prefer PR-AUC and recall (sensitivity) for positive

class.
* Use early stopping and regularization for boosting models.
* Try CatBoost for raw categorical-heavy EHR features (handles categories natively).
* Use Optuna (or Ray Tune) to tune each model; then ensemble the top performers (stacking).
* Calibrate probabilities (Platt / isotonic) before showing clinical risk.
* Log experiments and artifacts in MLflow (or other registry) with versioning.
* Add SHAP outputs and short textual explanations next to the risk score for clinicians.
* Implement logging + request auditing (who requested, when) for HIPAA compliance.

---

7) Scaling to many datasets & multi-model runs

* If you truly have *multiple datasets* (e.g., per hospital), treat each as a separate experiment: run the single-pipeline with different `data_path` and register model version with MLflow.
* For training many models in parallel, use a task queue (Celery) or orchestration (Airflow / Prefect).
* For very large runs, containerize the training job and run on a Kubernetes cluster or in a cloud batch service.

---

8) Monitoring and model drift

* Track input feature distributions and output risk distribution. Alert when distribution shift crosses threshold (Population Stability Index).
* Re-train on rolling windows (e.g., retrain monthly or when model performance drops).

---

9) Security, privacy & governance

* Encrypt PHI in transit and at rest. Use secure storage for models and logs.
* Minimize exposed PHI through the API; prefer PII-less IDs; only show explanations relevant to care.
* Maintain audit logs and access controls.

---

10) Files included in this doc (copy-paste into your project)

* `train_models.py` (full training loop)
* `tune_optuna.py` (Optuna tuning)
* `serve/api.py` (FastAPI server)
* `Dockerfile` & `docker-compose.yml` (examples)
* `README.md` with run instructions

End of document.