## **Step 1: Data Cleaning & Preprocessing**

**What I did:**

**Dropped identifiers** (encounter\_id, patient\_nbr) because they are record keys, not predictive features → keeping them would cause overfitting.  
  
 df = df.drop(["encounter\_id", "patient\_nbr"], axis=1, errors="ignore")

**Defined target variable** from the readmitted column:

* + <30 → **1 (readmitted within 30 days)**
  + >30 → **0 (not readmitted)**
  + 30 → removed (ambiguous cases).

df = df[df["readmitted"] != "30"]

df["target"] = df["readmitted"].apply(lambda x: 1 if x == "<30" else 0)

df = df.drop("readmitted", axis=1)

**Handled missing values:**

* + In the raw dataset, missing values were coded as "?".
  + Replaced them with NaN and then "Unknown" to keep information explicit.

df.replace("?", np.nan, inplace=True)

df.fillna("Unknown", inplace=True)

**Categorical encoding:**

* + Used **OneHotEncoder** (inside a pipeline) for categorical features such as race, gender, age group, admission type.

from sklearn.preprocessing import OneHotEncoder

cat\_pipeline = Pipeline([

("onehot", OneHotEncoder(handle\_unknown="ignore"))

])

**Scaling numeric features:**

* + Applied **StandardScaler** so that features like time\_in\_hospital or num\_lab\_procedures are on the same scale.

from sklearn.preprocessing import StandardScaler

num\_pipeline = Pipeline([

("scaler", StandardScaler())

])

**Combined pipelines with ColumnTransformer** to preprocess numeric + categorical features together.  
  
 from sklearn.compose import ColumnTransformer

preprocessor = ColumnTransformer([

("num", num\_pipeline, num\_cols),

("cat", cat\_pipeline, cat\_cols)

])

**Train/test split with stratification** to preserve class balance.  
  
 from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.3, stratify=y, random\_state=42

)

## **🔹 Step 2: Model Training**

**What I did:** I trained **three supervised learning models** using **Pipelines + GridSearchCV** (just like in the textbook).

**Logistic Regression** → Baseline interpretable model.  
  
 from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import GridSearchCV, Pipeline

log\_pipeline = Pipeline([

("preprocess", preprocessor),

("clf", LogisticRegression(max\_iter=1000))

])

log\_params = {"clf\_\_C": [0.01, 0.1, 1, 10]}

log\_grid = GridSearchCV(log\_pipeline, log\_params, cv=5, scoring="f1")

log\_grid.fit(X\_train, y\_train)

**Random Forest** → Handles non-linear interactions & gives feature importance.  
  
 from sklearn.ensemble import RandomForestClassifier

rf\_pipeline = Pipeline([

("preprocess", preprocessor),

("clf", RandomForestClassifier(random\_state=42))

])

rf\_params = {"clf\_\_n\_estimators": [100, 200],

"clf\_\_max\_depth": [5, 10, None]}

rf\_grid = GridSearchCV(rf\_pipeline, rf\_params, cv=5, scoring="f1")

rf\_grid.fit(X\_train, y\_train)

**XGBoost** → State-of-the-art gradient boosting for tabular data.  
  
 import xgboost as xgb

xgb\_pipeline = Pipeline([

("preprocess", preprocessor),

("clf", xgb.XGBClassifier(eval\_metric="logloss", random\_state=42))

])

xgb\_params = {"clf\_\_n\_estimators": [100, 200],

"clf\_\_max\_depth": [3, 5, 7]}

xgb\_grid = GridSearchCV(xgb\_pipeline, xgb\_params, cv=5, scoring="f1")

xgb\_grid.fit(X\_train, y\_train)

## **🔹 Step 3: Model Evaluation**

**What I did:**

* Used **confusion matrix** to check true/false positives & negatives.
* Plotted **ROC curves** and compared AUC scores across models.
* Calculated standard metrics: Accuracy, Precision, Recall, F1, AUC.

Example:

from sklearn.metrics import classification\_report, roc\_auc\_score

y\_pred = xgb\_grid.predict(X\_test)

y\_prob = xgb\_grid.predict\_proba(X\_test)[:, 1]

print(classification\_report(y\_test, y\_pred))

print("XGBoost AUC:", roc\_auc\_score(y\_test, y\_prob))

Also used **SHAP (Shapley values)** to explain which features most influenced predictions.  
  
 import shap

explainer = shap.TreeExplainer(xgb\_grid.best\_estimator\_["clf"])

shap\_values = explainer.shap\_values(X\_test)

* shap.summary\_plot(shap\_values, X\_test, plot\_type="bar")