

```

# =====
# DIABETES: PREDICT ANY READMISSION AT ALL (<30 or >30 vs NO)
# =====

import os, re, numpy as np, pandas as pd, matplotlib.pyplot as plt, seaborn as sns
import warnings; warnings.filterwarnings("ignore")
plt.rcParams.update({'font.size': 12})

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.base import BaseEstimator, TransformerMixin

from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.calibration import CalibratedClassifierCV

from sklearn.metrics import (
    classification_report, confusion_matrix, roc_auc_score,
    average_precision_score, roc_curve, precision_recall_curve
)

# ----- RareCategoryGrouper -----
class RareCategoryGrouper(BaseEstimator, TransformerMixin):
    def __init__(self, min_freq=50):
        self.min_freq = min_freq
        self.frequent_maps_ = {}
        self.columns_ = []

    def fit(self, X, y=None):
        X = pd.DataFrame(X)
        self.columns_ = X.columns.tolist()
        for c in self.columns_:
            vc = X[c].astype(str).value_counts(dropna=False)
            self.frequent_maps_[c] = set(vc[vc >= self.min_freq].index)
        return self

    def transform(self, X):
        X = pd.DataFrame(X, columns=self.columns_) if not isinstance(X, pd.DataFrame) else X.copy()
        for c in self.columns_:
            keep = self.frequent_maps_[c]
            X[c] = X[c].astype(str).where(X[c].astype(str).isin(keep), "RARE")
        return X

# ----- Load & Preprocess -----
df = pd.read_csv("diabetic_data.csv", na_values="?", low_memory=False)
df = df[df.gender != "Unknown/Invalid"].copy()
df.drop(columns=["encounter_id", "patient_nbr", "weight", "payer_code", "examide", "citoglipton"],

```

```

        errors="ignore", inplace=True)

# TARGET: ANY READMISSION
df["readmitted_any"] = (df["readmitted"] != "NO").astype(int)
print(f"Dataset: {df.shape} | Any readmission rate: {df['readmitted_any'].mean():.1%}")

# Feature Engineering
def age_mid(x):
    m = re.match(r"[(\d+)-(\d+)\]", str(x))
    return (int(m.group(1)) + int(m.group(2))) / 2 if m else np.nan
df["age_num"] = df["age"].apply(age_mid)

def bucket_icd9(code):
    if pd.isna(code): return "missing"
    s = str(code).strip()
    if s.startswith(("V","E")): return s[0].lower()
    try: v = float(s)
    except: return "other"
    if 390 <= v <= 459 or v == 785: return "circulatory"
    if 460 <= v <= 519 or v == 786: return "respiratory"
    if 520 <= v <= 579 or v == 787: return "digestive"
    if 250 <= v < 251: return "diabetes"
    if 800 <= v <= 999: return "injury"
    if 140 <= v <= 239: return "neoplasms"
    return "other"

for c in ["diag_1", "diag_2", "diag_3"]:
    df[f"{c}_bucket"] = df[c].apply(bucket_icd9)

df["max_glu_serum"] = df["max_glu_serum"].map({"None":0, "Norm":1, ">200":2, ">300":3}).fillna(0)
df["A1Cresult"] = df["A1Cresult"].map({"None":0, "Norm":1, ">7":2, ">8":3}).fillna(0)
df["race"] = df["race"].fillna("Other")
df["total_visits"] = df["number_outpatient"] + df["number_emergency"] + df["number_inpatient"]
df["meds_per_day"] = df["num_medications"] / (df["time_in_hospital"] + 1)

X = df.drop(columns=["readmitted", "readmitted_any", "age", "diag_1", "diag_2", "diag_3"])
y = df["readmitted_any"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# ----- Preprocessing -----
num_cols = X.select_dtypes("number").columns.tolist()
cat_cols = X.select_dtypes("object").columns.tolist()

num_pipe = Pipeline([("imp", SimpleImputer(strategy="median"))])
cat_pipe = Pipeline([
    ("rare", RareCategoryGrouper(min_freq=50)),
    ("imp", SimpleImputer(strategy="most_frequent")),
    ("ohe", OneHotEncoder(handle_unknown="ignore", sparse_output=False))
])

preprocess = ColumnTransformer([
    ("num", num_pipe, num_cols),

```

```

("cat", cat_pipe, cat_cols)
], verbose_feature_names_out=False)

# ----- Models -----
models = {
    "Random Forest": RandomForestClassifier(n_estimators=1000, max_features=0.4, n_jobs=-1,
random_state=42),
    "XGBoost": XGBClassifier(n_estimators=1000, max_depth=8, learning_rate=0.05, subsample=0.9,
        colsample_bytree=0.9, random_state=42, n_jobs=-1),
    "LightGBM": LGBMClassifier(n_estimators=1000, max_depth=12, learning_rate=0.05, num_leaves=200,
        subsample=0.9, colsample_bytree=0.9, random_state=42, verbose=-1),
    "Logistic Regression": LogisticRegression(max_iter=1000, n_jobs=-1),
    "Linear SVM": CalibratedClassifierCV(
        LinearSVC(class_weight="balanced", max_iter=20000), cv=3, method="sigmoid"
    )
}

# ----- Train & Large Beautiful Plots -----
results = []
n_models = len(models)
fig = plt.figure(figsize=(24, 10 * n_models)) # Very tall for spacing

plot_idx = 1
row = 0

for name, model in models.items():
    print(f"\n{'*' * 100}")
    print(f" TRAINING: {name.upper()}")
    print(f"{'*' * 100}")

    pipe = Pipeline([("prep", preprocess), ("clf", model)])
    pipe.fit(X_train, y_train)

    pred = pipe.predict(X_test)
    prob = pipe.predict_proba(X_test)[:, 1]

    auc = roc_auc_score(y_test, prob)
    pr_auc = average_precision_score(y_test, prob)
    report = classification_report(y_test, pred, output_dict=True)
    cm = confusion_matrix(y_test, pred)

    results.append({
        "Model": name,
        "AUC": round(auc, 4),
        "PR-AUC": round(pr_auc, 4),
        "Recall": round(report["1"]["recall"], 4),
        "Precision": round(report["1"]["precision"], 4),
        "F1": round(report["1"]["f1-score"], 4),
        "TP": int(cm[1,1])
    })

print(f"\nCLASSIFICATION REPORT — {name}")
print(classification_report(y_test, pred, digits=4))

```

```

print("Confusion Matrix:\n", cm)

# Large, spaced plots
# ROC Curve
ax1 = fig.add_subplot(n_models, 3, plot_idx)
fpr, tpr, _ = roc_curve(y_test, prob)
ax1.plot(fpr, tpr, label=f'AUC = {auc:.4f}', lw=4, color="darkorange")
ax1.plot([0,1],[0,1], 'k--', lw=2)
ax1.set_title(f"{name} — ROC Curve", fontsize=18, pad=20, fontweight="bold")
ax1.set_xlabel("False Positive Rate", fontsize=14)
ax1.set_ylabel("True Positive Rate", fontsize=14)
ax1.legend(fontsize=14)
ax1.grid(True, alpha=0.3)
plot_idx += 1

# PR Curve
ax2 = fig.add_subplot(n_models, 3, plot_idx)
prec, rec, _ = precision_recall_curve(y_test, prob)
ax2.plot(rec, prec, label=f'PR-AUC = {pr_auc:.4f}', lw=4, color="green")
ax2.set_title(f"{name} — Precision-Recall Curve", fontsize=18, pad=20, fontweight="bold")
ax2.set_xlabel("Recall", fontsize=14)
ax2.set_ylabel("Precision", fontsize=14)
ax2.legend(fontsize=14)
ax2.grid(True, alpha=0.3)
plot_idx += 1

# Confusion Matrix
ax3 = fig.add_subplot(n_models, 3, plot_idx)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False, annot_kws={"size": 20},
            xticklabels=["No Readmission", "Readmitted"],
            yticklabels=["No Readmission", "Readmitted"])
ax3.set_title(f"{name} — Confusion Matrix", fontsize=18, pad=20, fontweight="bold")
ax3.set_xlabel("Predicted", fontsize=14)
ax3.set_ylabel("Actual", fontsize=14)
plot_idx += 1

plt.suptitle("PREDICTING ANY DIABETES READMISSION ", fontsize=32, fontweight="bold", y=0.98)
plt.tight_layout(rect=[0, 0.03, 1, 0.95], h_pad=8.0, w_pad=6.0)
plt.show()

# ----- Final Ranking Table -----
results_df = pd.DataFrame(results).sort_values("AUC", ascending=False).reset_index(drop=True)
results_df.index += 1

print("\n" + "="*140)
print(" " * 55 + "FINAL RANKING")
print("="*140)
print(results_df.to_string(index=True))
print("="*140)

winner = results_df.iloc[0]
print(f"\nCHAMPION MODEL: {winner['Model']}")
print(f"    AUC    = {winner['AUC']:.4f}")

```

```

print(f"    Recall    = {winner['Recall']:.4f} → catches {winner['Recall']*100:.1f}% of ALL readmissions")
print(f"    F1-Score   = {winner['F1']:.4f}")
print(f"    True Positives = {winner['TP'],}")

```

```

import joblib, os
from xgboost import XGBClassifier

```

```

final_model = Pipeline([
    ("prep", preprocess),
    ("clf", XGBClassifier(n_estimators=1000, max_depth=8, learning_rate=0.05,
        subsample=0.9, colsample_bytree=0.9,
        random_state=42, n_jobs=-1))
])

```

```

print("Training champion model (this takes ~20–30 seconds)...")
final_model.fit(X_train, y_train)
print("Model trained! Saving...")

```

```

# Save both the model and the column order
joblib.dump(final_model, "anyreadmit_champion_model.pkl")
joblib.dump(X.columns.tolist(), "anyreadmit_feature_columns.pkl")
print("Model saved as 'anyreadmit_champion_model.pkl'")

```

```

import joblib
import pandas as pd
import numpy as np
import re
from IPython.display import display, Markdown, clear_output
import ipywidgets as widgets

```

```

# Load model
model = joblib.load("anyreadmit_champion_model.pkl")
cols = joblib.load("anyreadmit_feature_columns.pkl")
print("Model loaded — live predictions ready!")

```

```

# Mapping for A1C and glu serum
a1c_map = {"None": 0, "Norm": 1, ">7": 2, ">8": 3}
glu_map = {"None": 0, "Norm": 1, ">200": 2, ">300": 3}

```

```

# Blank patient with safe defaults
def blank_patient():
    data = {}
    meds = ["metformin", "repaglinide", "nateglinide", "chlorpropamide", "glimepiride", "glipizide",
        "glyburide", "pioglitazone", "rosiglitazone", "acarbose", "miglitol", "insulin"]
    for c in cols:
        if any(m in c for m in meds): data[c] = "No"
        elif c in ["change", "diabetesMed"]: data[c] = "No"
        elif "bucket" in c: data[c] = "missing"
        elif c in ["A1Cresult", "max_glu_serum"]: data[c] = "None"
        elif c in ["age", "gender", "race", "medical_specialty"]: data[c] = "Other"
        else: data[c] = 0
    return pd.DataFrame([data])[cols]

```

```

# UI
display(Markdown("# Live Readmission Risk Calculator"))
display(Markdown("**Enter patient details → instant clinical decision support**"))

age = widgets.Dropdown(options=["[0-10)","[10-20)","[20-30)","[30-40)","[40-50)","[50-60)","[60-70)","[70-80)","[80-90)","[90-100)"], value="[70-80)", description="Age:")
gender = widgets.Dropdown(options=["Male","Female"], value="Female", description="Gender:")
race = widgets.Dropdown(options=["Caucasian","AfricanAmerican","Hispanic","Asian","Other"],
value="Caucasian", description="Race:")
days = widgets.IntSlider(min=1, max=14, value=4, description="Days in Hosp:")
labs = widgets.IntSlider(min=0, max=130, value=40, description="Lab Tests:")
meds_count = widgets.IntSlider(min=1, max=81, value=16, description="Medications:")
outp = widgets.IntSlider(min=0, max=40, value=0, description="Outpatient:")
er = widgets.IntSlider(min=0, max=76, value=0, description="ER Visits:")
inp = widgets.IntSlider(min=0, max=21, value=1, description="Inpatient Visits:")
diag = widgets.Text(value="250.83", description="Primary ICD9:")
a1c = widgets.Dropdown(options=["None","Norm",">7",">8"], value="None", description="A1C Result:")

button = widgets.Button(description="Predict Risk Now", button_style="success",
layout=widgets.Layout(width="100%", height="70px"))
out = widgets.Output()

def predict_now(b):
    with out:
        clear_output()
        patient = blank_patient().copy()

        # Fill user data
        patient.loc[0, "age"] = age.value
        patient.loc[0, "gender"] = gender.value
        patient.loc[0, "race"] = race.value
        patient.loc[0, "time_in_hospital"] = days.value
        patient.loc[0, "num_lab_procedures"] = labs.value
        patient.loc[0, "num_medications"] = meds_count.value
        patient.loc[0, "number_outpatient"] = outp.value
        patient.loc[0, "number_emergency"] = er.value
        patient.loc[0, "number_inpatient"] = inp.value
        patient.loc[0, "diag_1"] = diag.value
        patient.loc[0, "A1Cresult"] = a1c.value

        patient["A1Cresult"] = patient["A1Cresult"].map(a1c_map)
        patient["max_glu_serum"] = patient["max_glu_serum"].map(glu_map)

        # Engineered features
        patient.loc[0, "total_visits"] = outp.value + er.value + inp.value
        patient.loc[0, "meds_per_day"] = meds_count.value / (days.value + 1)
        m = re.match(r"(\d+)-(\d+)\)", age.value)
        patient.loc[0, "age_num"] = (int(m.group(1)) + int(m.group(2))) / 2 if m else 50

    def icd_bucket(code):
        try: v = float(code);
        except: return "other"
        if 390<=v<=459 or v==785: return "circulatory"

```

```

    if 460<=v<=519 or v==786: return "respiratory"
    if 520<=v<=579 or v==787: return "digestive"
    if 250<=v<=251: return "diabetes"
    if 800<=v<=999: return "injury"
    if 140<=v<=239: return "neoplasms"
    return "other"
patient.loc[0, "diag_1_bucket"] = icd_bucket(diag.value)

# Predict
risk = model.predict_proba(patient)[0,1] * 100
display(Markdown(f"## Readmission Risk: **{risk:.1f}%**"))
if risk > 70:
    display(Markdown("<h2 style='color:red'>HIGH RISK – Urgent intervention required</h2>"))
elif risk > 55:
    display(Markdown("<h2 style='color:orange'>MODERATE RISK – Schedule follow-up</h2>"))
else:
    display(Markdown("<h2 style='color:green'>LOW RISK – Standard discharge plan</h2>"))

button.on_click(predict_now)
display(widgets.VBox([age,gender,race,days,labs,meds_count,outp,er,inp,diag,a1c,button,out]))

```