In [2]:
```
import pandas as pd

# Load the dataset into a pandas dataframe
og_df = pd.read_excel(r"D:\Anshuman\Anshuman\Personal\Summer 2025\Kaggle\Diabetes\d

Diab_df = og_df.copy()

Diab_df.head() # Debugging Statement
```

Out[2]:

	year	gender	age	location	race:AfricanAmerican	race:Asian	race:Caucasian	race:Hisp
0	2020	Female	32.0	Alabama	0	0	0	
1	2015	Female	29.0	Alabama	0	1	0	
2	2015	Male	18.0	Alabama	0	0	0	
3	2015	Male	41.0	Alabama	0	0	1	
4	2016	Female	52.0	Alabama	1	0	0	

In [3]:
```
Diab_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 16 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   year                 100000 non-null  int64  
 1   gender               100000 non-null  object 
 2   age                  100000 non-null  float64
 3   location             100000 non-null  object 
 4   race:AfricanAmerican 100000 non-null  int64  
 5   race:Asian           100000 non-null  int64  
 6   race:Caucasian       100000 non-null  int64  
 7   race:Hispanic        100000 non-null  int64  
 8   race:Other           100000 non-null  int64  
 9   hypertension         100000 non-null  int64  
 10  heart_disease        100000 non-null  int64  
 11  smoking_history      100000 non-null  object 
 12  bmi                  100000 non-null  float64
 13  hbA1c_level          100000 non-null  float64
 14  blood_glucose_level  100000 non-null  int64  
 15  diabetes             100000 non-null  int64  
dtypes: float64(3), int64(10), object(3)
memory usage: 12.2+ MB
```

Scaling the Year Values

In [4]:
```
mu_yr = Diab_df['year'].mean()
sigma_yr = Diab_df['year'].std()
Diab_df['year'] = (Diab_df['year'] - mu_yr) / sigma_yr
```

```
print("AVG Yr =", mu_yr)
print("STDEV Yr =", sigma_yr)
```

```
AVG Yr = 2018.36082
STDEV Yr = 1.3452386495600857
```

In [5]:
```
Diab_df.head()  # Debugging Statement to check the updated dataframe
```

Out[5]:

	year	gender	age	location	race:AfricanAmerican	race:Asian	race:Caucasian	race
0	1.218505	Female	32.0	Alabama	0	0	0	
1	-2.498308	Female	29.0	Alabama	0	1	0	
2	-2.498308	Male	18.0	Alabama	0	0	0	
3	-2.498308	Male	41.0	Alabama	0	0	1	
4	-1.754945	Female	52.0	Alabama	1	0	0	

Scaling Age

In [6]:
```
mu_age = Diab_df['age'].mean()
sigma_age = Diab_df['age'].std()

Diab_df['age'] = (Diab_df['age'] - mu_age) / sigma_age

print("AVG Age =", mu_age)
print("STDEV Age =", sigma_age)
```

```
AVG Age = 41.885856000000004
STDEV Age = 22.51683987161702
```

In [7]:
```
Diab_df.head()  # Debugging Statement to check the updated dataframe
```

Out[7]:

	year	gender	age	location	race:AfricanAmerican	race:Asian	race:Caucasian
0	1.218505	Female	-0.439043	Alabama	0	0	0
1	-2.498308	Female	-0.572276	Alabama	0	1	0
2	-2.498308	Male	-1.060800	Alabama	0	0	0
3	-2.498308	Male	-0.039342	Alabama	0	0	1
4	-1.754945	Female	0.449181	Alabama	1	0	0

Scaling BMI

In [8]:
```
mu_bmi = Diab_df['bmi'].mean()
sigma_bmi = Diab_df['bmi'].std()

Diab_df['bmi'] = (Diab_df['bmi'] - mu_bmi) / sigma_bmi
```

```
print("AVG BMI =", mu_bmi)
print("STDEV BMI =", sigma_bmi)
```

```
AVG BMI = 27.320767099999994
STDEV BMI = 6.636783416648368
```

In [9]:
```
Diab_df.head()  # Debugging Statement to check the updated dataframe
```

Out[9]:

	year	gender	age	location	race:AfricanAmerican	race:Asian	race:Caucasian
0	1.218505	Female	-0.439043	Alabama	0	0	0
1	-2.498308	Female	-0.572276	Alabama	0	1	0
2	-2.498308	Male	-1.060800	Alabama	0	0	0
3	-2.498308	Male	-0.039342	Alabama	0	0	1
4	-1.754945	Female	0.449181	Alabama	1	0	0

Scaling hb1Ac_levels

In [10]:
```
mu_hbA1c = Diab_df['hbA1c_level'].mean()
sigma_hbA1c = Diab_df['hbA1c_level'].std()

Diab_df['hbA1c_level'] = (Diab_df['hbA1c_level'] - mu_hbA1c) / sigma_hbA1c

print("AVG hbA1c =", mu_hbA1c)
print("STDEV hbA1c =", sigma_hbA1c)
```

```
AVG hbA1c = 5.527507
STDEV hbA1c = 1.0706720918835437
```

In [11]:
```
Diab_df.head()  # Debugging Statement to check the updated dataframe
```

Out[11]:

	year	gender	age	location	race:AfricanAmerican	race:Asian	race:Caucasian
0	1.218505	Female	-0.439043	Alabama	0	0	0
1	-2.498308	Female	-0.572276	Alabama	0	1	0
2	-2.498308	Male	-1.060800	Alabama	0	0	0
3	-2.498308	Male	-0.039342	Alabama	0	0	1
4	-1.754945	Female	0.449181	Alabama	1	0	0

Scaling Blood Glucose Levels

In [12]:
```
mu_bgl = Diab_df['blood_glucose_level'].mean()
sigma_bgl = Diab_df['blood_glucose_level'].std()

Diab_df['blood_glucose_level'] = (Diab_df['blood_glucose_level'] - mu_bgl) / sigma_
```

```
print("AVG BGL =", mu_bgl)
print("STDEV BGL =", sigma_bgl)
```

```
AVG BGL = 138.05806
STDEV BGL = 40.70813604870415
```

In [13]:
```
Diab_df.head()  # Debugging Statement to check the updated dataframe
```

Out[13]:

	year	gender	age	location	race:AfricanAmerican	race:Asian	race:Caucasian
0	1.218505	Female	-0.439043	Alabama	0	0	0
1	-2.498308	Female	-0.572276	Alabama	0	1	0
2	-2.498308	Male	-1.060800	Alabama	0	0	0
3	-2.498308	Male	-0.039342	Alabama	0	0	1
4	-1.754945	Female	0.449181	Alabama	1	0	0

One-hot encoding gender

In [14]:
```
# One-hot encoding gender
Diab_df = pd.get_dummies(
    Diab_df,
    columns=['gender'],            # just gender for now
    prefix='gender',
    drop_first=False               # keep all categories
)
print(Diab_df.columns.tolist())
```

```
['year', 'age', 'location', 'race:AfricanAmerican', 'race:Asian', 'race:Caucasian',
'race:Hispanic', 'race:Other', 'hypertension', 'heart_disease', 'smoking_history',
'bmi', 'hbA1c_level', 'blood_glucose_level', 'diabetes', 'gender_Female', 'gender_Ma
le', 'gender_Other']
```

Move gender columns back to where gender originally was

In [15]:
```
# find where 'year' is
year_idx = Diab_df.columns.get_loc('year')

# again grab the gender dummy names
gender_cols = [c for c in Diab_df.columns if c.startswith('gender_')]

# for each dummy, pop it out then insert right after 'year'
for i, col in enumerate(gender_cols, start=1):
    series = Diab_df.pop(col)
    Diab_df.insert(year_idx + i, col, series)

Diab_df.head()  # Debugging Statement to check the updated dataframe
```

Out[15]:

	year	gender_Female	gender_Male	gender_Other	age	location	race:African
0	1.218505	True	False	False	-0.439043	Alabama	
1	-2.498308	True	False	False	-0.572276	Alabama	
2	-2.498308	False	True	False	-1.060800	Alabama	
3	-2.498308	False	True	False	-0.039342	Alabama	
4	-1.754945	True	False	False	0.449181	Alabama	

Convert encoded values from True-False to 1-0

In [16]:
```
gender_cols = [c for c in Diab_df.columns if c.startswith('gender_')]

# Convert in-place to 0/1 ints
Diab_df[gender_cols] = Diab_df[gender_cols].astype(int)

Diab_df.head()  # Debugging Statement to check the updated dataframe
```

Out[16]:

	year	gender_Female	gender_Male	gender_Other	age	location	race:African
0	1.218505	1	0	0	-0.439043	Alabama	
1	-2.498308	1	0	0	-0.572276	Alabama	
2	-2.498308	0	1	0	-1.060800	Alabama	
3	-2.498308	0	1	0	-0.039342	Alabama	
4	-1.754945	1	0	0	0.449181	Alabama	

One-hot encoding Location

In [17]:
```
# One-hot encoding Location
Diab_df = pd.get_dummies(
    Diab_df,
    columns=['location'],          # just location for now
    prefix='location',
    drop_first=False               # keep all categories
)

print(Diab_df.columns.tolist())
```

```
['year', 'gender_Female', 'gender_Male', 'gender_Other', 'age', 'race:AfricanAmerica
n', 'race:Asian', 'race:Caucasian', 'race:Hispanic', 'race:Other', 'hypertension',
'heart_disease', 'smoking_history', 'bmi', 'hbA1c_level', 'blood_glucose_level', 'di
abetes', 'location_Alabama', 'location_Alaska', 'location_Arizona', 'location_Arkans
as', 'location_California', 'location_Colorado', 'location_Connecticut', 'location_D
elaware', 'location_District of Columbia', 'location_Florida', 'location_Georgia',
'location_Guam', 'location_Hawaii', 'location_Idaho', 'location_Illinois', 'location
_Indiana', 'location_Iowa', 'location_Kansas', 'location_Kentucky', 'location_Louisi
ana', 'location_Maine', 'location_Maryland', 'location_Massachusetts', 'location_Mic
higan', 'location_Minnesota', 'location_Mississippi', 'location_Missouri', 'location
_Montana', 'location_Nebraska', 'location_Nevada', 'location_New Hampshire', 'locati
on_New Jersey', 'location_New Mexico', 'location_New York', 'location_North Carolin
a', 'location_North Dakota', 'location_Ohio', 'location_Oklahoma', 'location_Orego
n', 'location_Pennsylvania', 'location_Puerto Rico', 'location_Rhode Island', 'locat
ion_South Carolina', 'location_South Dakota', 'location_Tennessee', 'location_Texa
s', 'location_United States', 'location_Utah', 'location_Vermont', 'location_Virgin
Islands', 'location_Virginia', 'location_Washington', 'location_West Virginia', 'loc
ation_Wisconsin', 'location_Wyoming']
```

In [ ]:
```
# Checking to make sure the one-hot encoding worked and number of encoded columns =
# [attach excel screenshot here in markdown]
```

In [18]:
```
# list out all columns containing 'location'
location_cols = [col for col in Diab_df.columns if 'location' in col]

# how many are there?
print("Number of location dummies:", len(location_cols))

# (optional) see them
print(location_cols)
```

```
Number of location dummies: 55
['location_Alabama', 'location_Alaska', 'location_Arizona', 'location_Arkansas', 'lo
cation_California', 'location_Colorado', 'location_Connecticut', 'location_Delawar
e', 'location_District of Columbia', 'location_Florida', 'location_Georgia', 'locati
on_Guam', 'location_Hawaii', 'location_Idaho', 'location_Illinois', 'location_Indian
a', 'location_Iowa', 'location_Kansas', 'location_Kentucky', 'location_Louisiana',
'location_Maine', 'location_Maryland', 'location_Massachusetts', 'location_Michiga
n', 'location_Minnesota', 'location_Mississippi', 'location_Missouri', 'location_Mon
tana', 'location_Nebraska', 'location_Nevada', 'location_New Hampshire', 'location_N
ew Jersey', 'location_New Mexico', 'location_New York', 'location_North Carolina',
'location_North Dakota', 'location_Ohio', 'location_Oklahoma', 'location_Oregon', 'l
ocation_Pennsylvania', 'location_Puerto Rico', 'location_Rhode Island', 'location_So
uth Carolina', 'location_South Dakota', 'location_Tennessee', 'location_Texas', 'loc
ation_United States', 'location_Utah', 'location_Vermont', 'location_Virgin Island
s', 'location_Virginia', 'location_Washington', 'location_West Virginia', 'location_
Wisconsin', 'location_Wyoming']
```

In [19]:
```
Diab_df.head()  # Debugging Statement to check the updated dataframe
```

Out[19]:

	year	gender_Female	gender_Male	gender_Other	age	race:AfricanAmerican
0	1.218505	1	0	0	-0.439043	0
1	-2.498308	1	0	0	-0.572276	0
2	-2.498308	0	1	0	-1.060800	0
3	-2.498308	0	1	0	-0.039342	0
4	-1.754945	1	0	0	0.449181	1

5 rows × 72 columns

In [20]:
```
age_idx = Diab_df.columns.get_loc('age')

locn_cols = [col for col in Diab_df.columns if 'location' in col]

for i, col in enumerate(locn_cols, start=1):
    series = Diab_df.pop(col)
    Diab_df.insert(age_idx + i, col, series)

Diab_df.head()  # Debugging Statement to check the updated dataframe
```

Out[20]:

	year	gender_Female	gender_Male	gender_Other	age	location_Alabama	loc
0	1.218505	1	0	0	-0.439043	True	
1	-2.498308	1	0	0	-0.572276	True	
2	-2.498308	0	1	0	-1.060800	True	
3	-2.498308	0	1	0	-0.039342	True	
4	-1.754945	1	0	0	0.449181	True	

5 rows × 72 columns

In [21]:
```
locn_cols = [col for col in Diab_df.columns if 'location' in col]

# Convert in-place to 0/1 ints
Diab_df[locn_cols] = Diab_df[locn_cols].astype(int)

Diab_df.head()  # Debugging Statement to check the updated dataframe
```

Out[21]:

	year	gender_Female	gender_Male	gender_Other	age	location_Alabama	loc
0	1.218505	1	0	0	-0.439043	1	
1	-2.498308	1	0	0	-0.572276	1	
2	-2.498308	0	1	0	-1.060800	1	
3	-2.498308	0	1	0	-0.039342	1	
4	-1.754945	1	0	0	0.449181	1	

5 rows × 72 columns

In [22]:
```
# One-hot encoding smoking_history
Diab_df = pd.get_dummies(
    Diab_df,
    columns=['smoking_history'],   # just smoking_history for now
    prefix='smoking',
    drop_first=False               # keep all categories
)

Diab_df.head()  # Debugging Statement to check the updated dataframe
```

Out[22]:

	year	gender_Female	gender_Male	gender_Other	age	location_Alabama	loc
0	1.218505	1	0	0	-0.439043	1	
1	-2.498308	1	0	0	-0.572276	1	
2	-2.498308	0	1	0	-1.060800	1	
3	-2.498308	0	1	0	-0.039342	1	
4	-1.754945	1	0	0	0.449181	1	

5 rows × 77 columns

In [23]:
```
mapping = {}
for col in Diab_df.columns:
    if col.startswith('smoking_'):
        # take the part after the first underscore,
        # lowercase it, replace spaces with underscores
        suffix = col.split('_', 1)[1].lower().replace(' ', '_')
        mapping[col] = f'smoking_history_{suffix}'

Diab_df.rename(columns=mapping, inplace=True)

Diab_df.head()
```

Out[23]:

	year	gender_Female	gender_Male	gender_Other	age	location_Alabama	loc
0	1.218505	1	0	0	-0.439043	1	
1	-2.498308	1	0	0	-0.572276	1	
2	-2.498308	0	1	0	-1.060800	1	
3	-2.498308	0	1	0	-0.039342	1	
4	-1.754945	1	0	0	0.449181	1	

5 rows × 77 columns

In [24]:
```
# this replaces every "race:" prefix with "race_"
Diab_df.rename(
    columns=lambda c: c.replace("race:", "race_") if c.startswith("race:") else c,
    inplace=True
)

Diab_df.head()  # Debugging Statement to check the updated dataframe
```

Out[24]:

	year	gender_Female	gender_Male	gender_Other	age	location_Alabama	loc
0	1.218505	1	0	0	-0.439043	1	
1	-2.498308	1	0	0	-0.572276	1	
2	-2.498308	0	1	0	-1.060800	1	
3	-2.498308	0	1	0	-0.039342	1	
4	-1.754945	1	0	0	0.449181	1	

5 rows × 77 columns

ML Modelling begins:

Creating 70-20-10 splits

In [ ]:
```
from sklearn.model_selection import train_test_split

# Features & target
X = Diab_df.drop('diabetes', axis=1)
y = Diab_df['diabetes']

# 1) Split off 30% (to later carve into val/test)
X_train, X_temp, y_train, y_temp = train_test_split(
    X, y,
    test_size=0.30,         # 30% goes to temp
    random_state=42,
    stratify=y
```

```
)

# 2) Split that temp into 20% val, 10% test = 2/3 of temp vs 1/3 of temp
X_valid, X_test, y_valid, y_test = train_test_split(
    X_temp, y_temp,
    test_size=1/3,          # 1/3 of 30% = 10% overall
    random_state=42,
    stratify=y_temp
)

print("Train:", X_train.shape, y_train.shape)
print("Valid:", X_valid.shape, y_valid.shape)
print("Test: ", X_test.shape,  y_test.shape)
```

```
Train: (70000, 76) (70000,)
Valid: (20000, 76) (20000,)
Test:  (10000, 76) (10000,)
```

In [28]:
```
# right after your 70/20/10 split, before any run_model calls:
feature_cols = X_train.columns.tolist()
```

Helper to Train & Evaluate on Validation (and Test)

In [30]:
```
from sklearn.metrics import roc_auc_score, classification_report, confusion_matrix

# 1) capture once, at the top of your notebook:
feature_cols = X_train.columns.tolist()

def run_model(name, model):
    print(f"\n===== {name} =====")
    # train
    model.fit(X_train, y_train)

    # validate
    yv_pred = model.predict(X_valid)
    yv_prob = model.predict_proba(X_valid)[:,1]
    print("--- Validation ---")
    print("ROC-AUC (val):", roc_auc_score(y_valid, yv_prob))
    print(classification_report(y_valid, yv_pred))

    # test
    yt_prob = model.predict_proba(X_test)[:,1]
    print("\n--- Test ---")
    print("ROC-AUC (test):", roc_auc_score(y_test, yt_prob))

    # append full-data probs **using only the original features**
    col = f"diabetes_probability_{name.lower().replace(' ', '_')}"
    Diab_df[col] = model.predict_proba(Diab_df[feature_cols])[:,1]
    print("Appended:", col)
```

In [31]:
```
from sklearn.linear_model    import LogisticRegression
from sklearn.tree            import DecisionTreeClassifier
from sklearn.ensemble        import RandomForestClassifier
from xgboost                 import XGBClassifier
from lightgbm                import LGBMClassifier
```

```
from sklearn.neural_network  import MLPClassifier
from sklearn.naive_bayes     import GaussianNB

# 2.1 Logistic Regression
run_model(
    "logistic_regression",
    LogisticRegression(max_iter=1000, random_state=42)
)

# 2.2 Decision Tree
run_model(
    "decision_tree",
    DecisionTreeClassifier(random_state=42)
)

# 2.3 Random Forest
run_model(
    "random_forest",
    RandomForestClassifier(n_estimators=100, random_state=42)
)

# 2.4 XGBoost
run_model(
    "xgboost",
    XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)
)

# 2.5 LightGBM
run_model(
    "lightgbm",
    LGBMClassifier(random_state=42)
)

# 2.6 Neural Network
run_model(
    "neural_network",
    MLPClassifier(hidden_layer_sizes=(100,), max_iter=300, random_state=42)
)

# 2.7 Naive Bayes
run_model(
    "naive_bayes",
    GaussianNB()
)
```

```
===== logistic_regression =====
--- Validation ---
ROC-AUC (val): 0.962003600128576
              precision    recall  f1-score   support

           0       0.97      0.99      0.98     18300
           1       0.87      0.62      0.73      1700

    accuracy                           0.96     20000
   macro avg       0.92      0.81      0.85     20000
weighted avg       0.96      0.96      0.96     20000


--- Test ---
ROC-AUC (test): 0.959932369013179
Appended: diabetes_probability_logistic_regression

===== decision_tree =====
--- Validation ---
ROC-AUC (val): 0.8501398264223722
              precision    recall  f1-score   support

           0       0.97      0.98      0.98     18300
           1       0.73      0.72      0.73      1700

    accuracy                           0.95     20000
   macro avg       0.85      0.85      0.85     20000
weighted avg       0.95      0.95      0.95     20000


--- Test ---
ROC-AUC (test): 0.8563645130183222
Appended: diabetes_probability_decision_tree

===== random_forest =====
--- Validation ---
ROC-AUC (val): 0.9592511732561877
              precision    recall  f1-score   support

           0       0.97      1.00      0.98     18300
           1       1.00      0.66      0.79      1700

    accuracy                           0.97     20000
   macro avg       0.98      0.83      0.89     20000
weighted avg       0.97      0.97      0.97     20000


--- Test ---
ROC-AUC (test): 0.9580896817743491
Appended: diabetes_probability_random_forest

===== xgboost =====
```

```
c:\Users\Anshuman\AppData\Local\Programs\Python\Python312\Lib\site-packages\xgboost
\training.py:183: UserWarning: [15:37:33] WARNING: C:\actions-runner\_work\xgboost\x
gboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
--- Validation ---
ROC-AUC (val): 0.9774626486660237
              precision    recall  f1-score   support

           0       0.97      1.00      0.98     18300
           1       0.96      0.69      0.80      1700

    accuracy                           0.97     20000
   macro avg       0.97      0.84      0.89     20000
weighted avg       0.97      0.97      0.97     20000


--- Test ---
ROC-AUC (test): 0.9753882352941174
Appended: diabetes_probability_xgboost

===== lightgbm =====
[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Info] Number of positive: 5950, number of negative: 64050
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing wa
s 0.002143 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 544
[LightGBM] [Info] Number of data points in the train set: 70000, number of used feat
ures: 75
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.085000 -> initscore=-2.376273
[LightGBM] [Info] Start training from score -2.376273
--- Validation ---
ROC-AUC (val): 0.9781022018643524
              precision    recall  f1-score   support

           0       0.97      1.00      0.98     18300
           1       0.98      0.67      0.80      1700

    accuracy                           0.97     20000
   macro avg       0.97      0.84      0.89     20000
weighted avg       0.97      0.97      0.97     20000


--- Test ---
ROC-AUC (test): 0.9770165220186435
Appended: diabetes_probability_lightgbm

===== neural_network =====
c:\Users\Anshuman\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn
\neural_network\_multilayer_perceptron.py:690: ConvergenceWarning: Stochastic Optimi
zer: Maximum iterations (300) reached and the optimization hasn't converged yet.
  warnings.warn(
```

```
--- Validation ---
ROC-AUC (val): 0.9560295403407265
              precision    recall  f1-score   support

           0       0.97      0.98      0.98     18300
           1       0.79      0.70      0.74      1700

    accuracy                           0.96     20000
   macro avg       0.88      0.84      0.86     20000
weighted avg       0.96      0.96      0.96     20000


--- Test ---
ROC-AUC (test): 0.9602322082931534
Appended: diabetes_probability_neural_network

===== naive_bayes =====
--- Validation ---
ROC-AUC (val): 0.8826083895853423
              precision    recall  f1-score   support

           0       0.99      0.59      0.74     18300
           1       0.17      0.92      0.29      1700

    accuracy                           0.61     20000
   macro avg       0.58      0.75      0.51     20000
weighted avg       0.92      0.61      0.70     20000


--- Test ---
ROC-AUC (test): 0.8852560591449695
Appended: diabetes_probability_naive_bayes
```

In [33]:
```
Diab_df.head()  # Final Debugging Statement to check the updated dataframe
```

Out[33]:

	year	gender_Female	gender_Male	gender_Other	age	location_Alabama	loc
0	1.218505	1	0	0	-0.439043	1	
1	-2.498308	1	0	0	-0.572276	1	
2	-2.498308	0	1	0	-1.060800	1	
3	-2.498308	0	1	0	-0.039342	1	
4	-1.754945	1	0	0	0.449181	1	

5 rows × 84 columns

In [42]:
```
import re
import unicodedata
import pandas as pd

# 1️⃣ Load & normalize your saved output
text = open(r"D:\Anshuman\Anshuman\Personal\Summer 2025\Kaggle\Diabetes\ModelRunOut
```

```
# Normalize Unicode (NFKC) and replace all hyphen-like characters with ASCII dash
text = unicodedata.normalize('NFKC', text)
for hy in ['\u2010','\u2011','\u2012','\u2013','\u2014']:
    text = text.replace(hy, '-')

records = []
for block in text.split("Appended:"):
    if "=====" not in block:
        continue

    # Extract model name
    header = block.split("-----")[0].strip()
    model  = re.findall(r"===== (\w+)", header)[0]

    # Extract test accuracy
    acc_match = re.search(r"accuracy\s+([\d\.]+)", block)
    acc       = float(acc_match.group(1)) if acc_match else None

    # Extract test ROC-AUC (catch any dash or space between ROC and AUC)
    # Robustly extract the first float on the ROC line
    roc = None
    for line in block.splitlines():
        if "ROC" in line:
            m = re.search(r"([\d]+\.\d+)", line)
            if m:
                roc = float(m.group(1))
            break

    records.append({
        "Model": model,
        "Test Accuracy": acc,
        "Test ROC-AUC":  roc
    })

# 2️⃣ Build the DataFrame
df = pd.DataFrame(records).set_index("Model")

# Convert to numeric and round
df['Test Accuracy'] = pd.to_numeric(df['Test Accuracy'], errors='coerce').round(3)
df['Test ROC-AUC']  = pd.to_numeric(df['Test ROC-AUC'],  errors='coerce').round(3)

# 3️⃣ Display neatly
from IPython.display import display

print("📊 Ranked by Test Accuracy:")
display(
    df.sort_values("Test Accuracy", ascending=False)
      .style
      .background_gradient(axis=0, cmap="Greens")
      .set_caption("Models sorted by Test Accuracy")
)

print("\n📊 Ranked by Test ROC-AUC:")
display(
    df.sort_values("Test ROC-AUC", ascending=False)
```

```
    .style
    .background_gradient(axis=0, cmap="Blues")
    .set_caption("Models sorted by Test ROC-AUC")
)
```

📊 Ranked by Test Accuracy:

Models sorted by Test Accuracy

Model	Test Accuracy	Test ROC-AUC
random_forest	0.970000	0.959000
lightgbm	0.970000	0.978000
xgboost	0.970000	0.977000
neural_network	0.960000	0.956000
logistic_regression	0.960000	0.962000
decision_tree	0.950000	0.850000
naive_bayes	0.610000	0.883000

📊 Ranked by Test ROC-AUC:

Models sorted by Test ROC-AUC

Model	Test Accuracy	Test ROC-AUC
lightgbm	0.970000	0.978000
xgboost	0.970000	0.977000
logistic_regression	0.960000	0.962000
random_forest	0.970000	0.959000
neural_network	0.960000	0.956000
naive_bayes	0.610000	0.883000
decision_tree	0.950000	0.850000

In [43]:
```
Diab_df.filter(like='diabetes_probability')
```

Out[43]:

	diabetes_probability_logistic_regression	diabetes_probability_decision_tree	diabetes_p
0	0.000254	0.0	
1	0.000094	0.0	
2	0.000592	0.0	
3	0.000344	0.0	
4	0.013924	0.0	
...	...	...	
99995	0.003370	0.0	
99996	0.016394	0.0	
99997	0.146722	0.0	
99998	0.046704	0.0	
99999	0.000021	0.0	

100000 rows × 7 columns

In [44]:
```
# Convert final Diab_df to excel
Diab_df.to_excel(r"D:\Anshuman\Anshuman\Personal\Summer 2025\Kaggle\Diabetes\diabet
print("✅ Final DataFrame saved to diabetes_predictions.xlsx")
```

```
✅ Final DataFrame saved to diabetes_predictions.xlsx
```