CREDIT RISK MODELING

GROUP 16

ASWATHY CHANDRASEKAR
JAHNAVI CHENNAMSETTY
TUSHARA TUMMALA

EXECUTIVE SUMMARY

Project Overview:

This project focuses on the development of a Credit Risk Prediction Model to effectively balance the need for minimizing credit defaults while optimizing revenue generation. The model predicts the Probability of Default (PD) for applicants, enabling data-driven decisions to accept low-risk applicants and reject high-risk ones.

Objective:

The goal was to establish a robust model with a 10% default rate threshold, ensuring that applicants with a PD below the threshold are accepted, while those above the threshold are rejected.

Impact on the Company:

- Increased Profitability: By identifying and accepting only low-risk customers, the model minimizes defaults and optimizes revenue.
- Streamlined Operations: The computational efficiency of XGBoost ensures faster credit assessments, reducing processing times and costs.

.

	TRAIN			TEST 1			TEST 2		
	# Total	Default Rate	Revenue	# Total	Default Rate	Revenue	# Total	Default Rate	Revenue
Conservative Strategy	47277	0.0508	6535.72	9773	0.0597	1331.17	9718	0.0661	1331.73
Aggressive Strategy	51467	0.0965	6767.00	10750	0.0978	1405.62	10199	0.0829	1370.31

DATA

- For our credit risk model, we've decided to focus on data from March 1, 2017 to March 31, 2018, a 13-month period. This timeframe offers us with enough historical data to capture a wide range of customer behaviors and economic situations.

- Selected 20% Dataset- 1107082
- Unique Customers- 91783

- We performed key data manipulation tasks, including one-hot encoding of categorical variables to convert them into numerical format, and aggregation of customer-level data(i.e average, min, max, sum etc.) helps create meaningful features.

Features

S_ features

B_ features

P_ features

D_ features

R_ features

Category	Observations	Default Rate
All Applications	1107082	0.256507
Applications with 13 Months of historical data	1005524	0.229405
Applications with 12 Months of historical data	25380	0.378723
Applications with 11 Months of historical data	12749	0.440897
Applications with 10 Months of historical data	12290	0.465764
Applications with 9 Months of historical data	11502	0.435837
Applications with 8 Months of historical data	9352	0.450813
Applications with 7 Months of historical data	7322	0.414914
Applications with 6 Months of historical data	6654	0.412985

Category	Observations	Default Rate
Applications with 5 Months of historical data	4665	0.394427
Applications with 4 Months of historical data	3752	0.430704
Applications with 3 Months of historical data	3474	0.357513
Applications with 2 Months of historical data	2434	0.312243
Applications with 1 Months of historical data	984	0.331301

FEATURES

- **Spending Variables (S_ features)**: Captures customers' spending patterns to assess overspending risks.
- **Balance Variables (B_ features)**: Reflects account balances to evaluate financial stability.
- **Payment Variables (P_ features)**: Tracks repayment behavior to gauge consistency in meeting obligations.
- **Delinquency Variables (D_ features)**: Identifies missed payments to flag early signs of potential defaults.
- **Risk Variable (R_ features):** Credit scores and credit usage ratios that determine the amount of risk involved with lending to a certain borrower.

Category	# of features
Delinquency variables	96
Balance variables	40
Risk variables	28
Spend variables	22
Payment variables	3

FEATURE ENGINEERING

Numerical Features

- Mean, Sum, Min, Max, SD
- Mean, Min, Max, SD for the last 3,6,9 and 12 months.
- Rate of Change in the last 1 year
- Spend to Balance Ratio
- Payment to Spend Ratio
- Spend Volatility
- Balance Volatility
- Payment Volatility
- Days since last transaction

Categorical Features

- Response Rate for the last 3,6,9 and 12 months.
- Ever Response for the last 3,6,9 and 12 months.

Columns	Mean	MIN	1%	5%	Median	95%	99%	MAX	%Missing
P_2_min_3m	0.6143	-0.42058	-0.10185	0.110415	63855	0.958747	1.001349	1.009947	0.611692
P_2_mean_3m	0.6388	-0.3745	-0.04081	0.165658	63855	0.972166	1.004345	1.009947	0.611692
P_2_max_3m	0.66273	-0.35184	-0.00381	0.209769	63855	0.98996	1,008467	1.01	0.611692
B_1_mean_3m	0.13679	-0.24415	0.002226	0.00395	64248	0.6462	1.050152	1.323839	0
D_42_min	0.163685	-0.00028	0.000931	0.003642	15816	0.522016	0.9373	4.1853	75.38289

ONE-HOT ENCODING

- Applied one-hot encoding to transform 11 categorical features.
- Utilized the get_dummies function to generate new binary variables.
- Created 34 indicator features, each named with the original feature and a suffix indicating the category value.

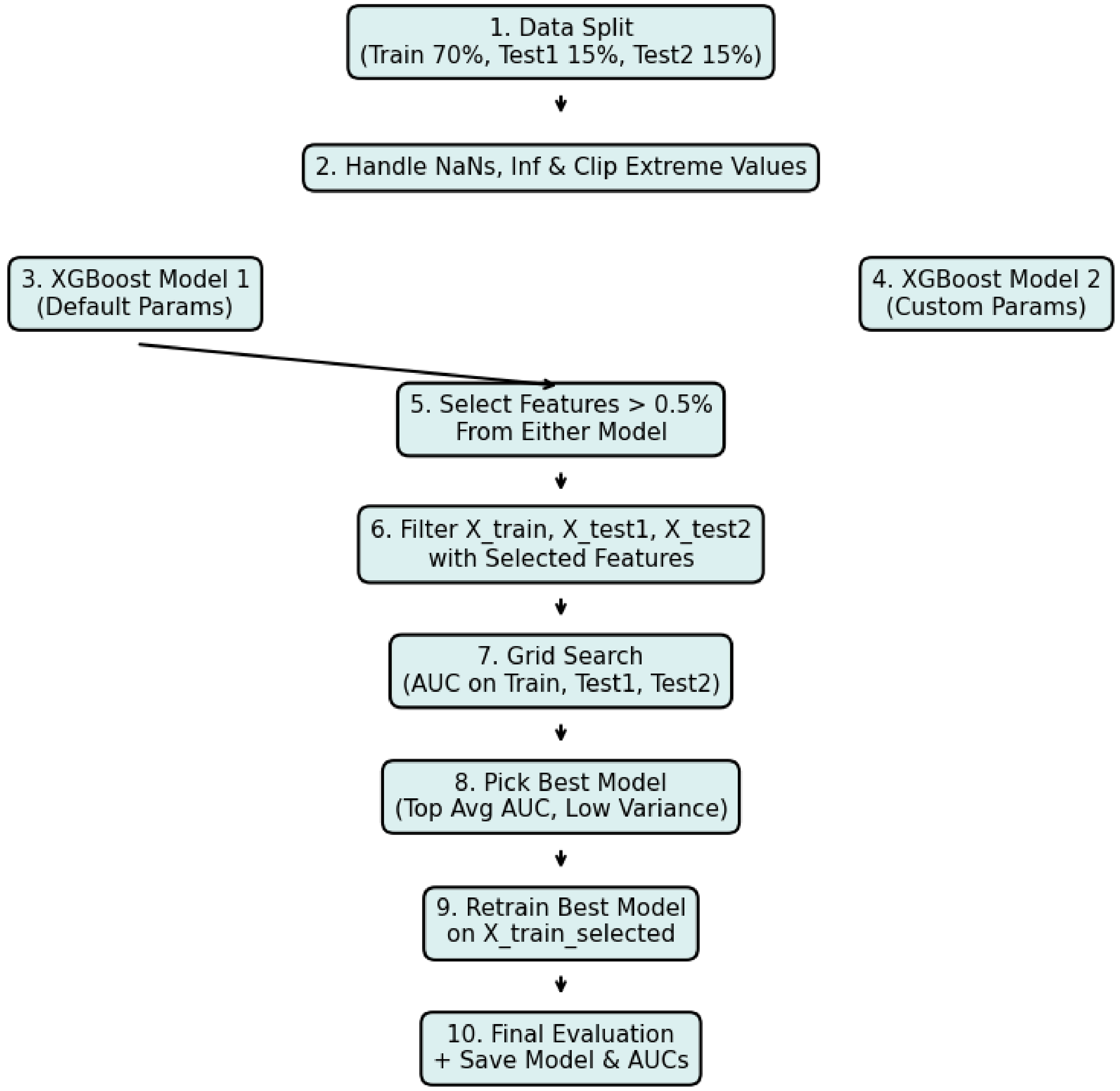
```
categorical_cols = ['B_30', 'B_38', 'D_114', 'D_116', 'D_117', 'D_120', 'D_126', 'D_63', 'D_64', 'D_66', 'D_68']
df_encoded = pd.get_dummies(df, columns=categorical_cols, drop_first=True)
def one_hot_encode(df, categorical_columns, drop_first=True):

    df_encoded = pd.get_dummies(df, columns=categorical_columns, drop_first=drop_first)
    encoded_cols = [col for col in df_encoded.columns if any(c in col for c in categorical_columns)]
    return df_encoded, encoded_cols
categorical_cols = ['B_30', 'B_38', 'D_114', 'D_116', 'D_117', 'D_120', 'D_126', 'D_63', 'D_64', 'D_66', 'D_68']
df_encoded, encoded_cols = one_hot_encode(df, categorical_cols)

# Convert boolean or category dtype to int if needed
df_encoded[encoded_cols] = df_encoded[encoded_cols].astype(int)
```

CODE

List of users

Features	n_uniques
1 B_30	[0.0, 2.0, 1.0, nan]
2 B_38	[2.0, 1.0, 5.0, 3.0, 7.0, 6.0, 4.0, nan]
3 D_114	[1.0, nan, 0.0]
4 D_116	[0.0, nan, 1.0]
5 D_117	[4.0, nan, 2.0, 3.0, 5.0, 6.0, -1.0, 1.0]
6 D_120	[0.0, nan, 1.0]
7 D_126	[1.0, nan, 0.0, -1.0]
8 D_63	[CR, CO, CL, XZ, XL, XM]
9 D_64	[O, nan, R, U, -1]
10 D_66	[nan, 1.0, 0.0]
11D_68	[6.0, nan, 2.0, 3.0, 4.0, 5.0, 1.0, 0.0]

Feature Selection



Feature Selection Process

- The features selected for the model exhibit a feature importance greater than 0.5%. Variables with higher importance are regarded as more informative and contribute more significantly to the prediction process. Prioritizing such features enhances the model’s accuracy and overall effectiveness by concentrating on the most relevant information.
- 16 Features were selected from 4174 features.

```
selected_features = set(fi_default[fi_default['Importance'] > 0.005]['Feature']).union(
    set(fi_custom[fi_custom['Importance'] > 0.005]['Feature'])
)
```

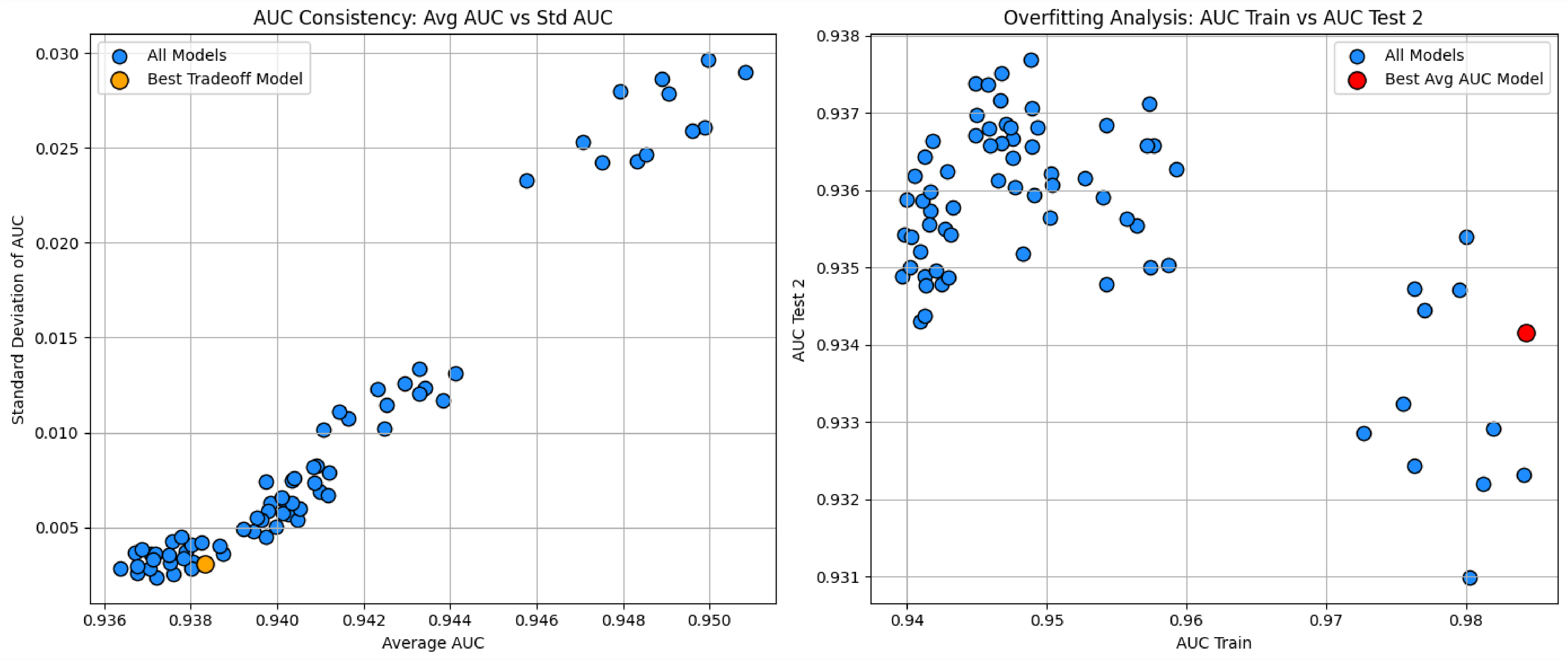
```
selected_features
```

```
{'B_11_min_3m',
 'B_1_mean_3m',
 'B_1_min_3m',
 'B_2_mean_3m',
 'B_2_min_3m',
 'D_42_min',
 'D_44_max_3m',
 'D_44_max_9m',
 'D_51_mean_12m',
 'P_2_max_3m',
 'P_2_mean_3m',
 'P_2_min_3m',
 'P_2_min_6m',
 'R_1_max_6m',
 'R_1_mean_3m',
 'R_2_mean_3m'}
```

XGBOOST - GRID SEARCH

```python
import pandas as pd
import numpy as np
from xgboost import XGBClassifier
from sklearn.metrics import roc_auc_score
import itertools

# Define the parameter grid
param_grid_xgb = {
    'n_estimators': [50, 100, 300],        # Number of trees
    'learning_rate': [0.01, 0.1],          # Learning rate
    'subsample': [0.5, 0.8],               # Percentage of observations used in each tree
    'colsample_bytree': [0.5, 1.0],        # Percentage of features used in each tree
    'scale_pos_weight': [1, 5, 10]         # Weight of default observations
}

# Load previous results if available (avoid repeating completed iterations)
csv_filename = "grid_search_results_xgb.csv"
try:
    results_df = pd.read_csv(csv_filename)
    completed_combinations = set(tuple(row) for row in results_df.iloc[:, :5].values)
except FileNotFoundError:
    results_df = pd.DataFrame(columns=[
        "Trees", "LR", "Subsample", "% Features", "Weight of Default",
        "AUC Train", "AUC Test 1", "AUC Test 2"
    ])
    completed_combinations = set()

# Iterate through all parameter combinations
for n_estimators, learning_rate, subsample, colsample_bytree, scale_pos_weight in itertools.product(
    param_grid_xgb['n_estimators'],
    param_grid_xgb['learning_rate'],
    param_grid_xgb['subsample'],
    param_grid_xgb['colsample_bytree'],
    param_grid_xgb['scale_pos_weight']
):
    param_tuple = (n_estimators, learning_rate, subsample, colsample_bytree, scale_pos_weight)
    if param_tuple in completed_combinations:
        continue  # Skip already completed combinations

    # Define model
    xgb_model = XGBClassifier(
        n_estimators=n_estimators,
        learning_rate=learning_rate,
        subsample=subsample,
        colsample_bytree=colsample_bytree,
        scale_pos_weight=scale_pos_weight,
        objective='binary:logistic',
        use_label_encoder=False,
        eval_metric='auc',
        random_state=42
    )

    # Train model
    xgb_model.fit(X_train_selected, y_train)

    # Evaluate using AUC
    auc_train = roc_auc_score(y_train, xgb_model.predict_proba(X_train_selected)[:, 1])
    auc_test1 = roc_auc_score(y_test1, xgb_model.predict_proba(X_test1_selected)[:, 1])
    auc_test2 = roc_auc_score(y_test2, xgb_model.predict_proba(X_test2_selected)[:, 1])

    # Store results
    results_df.loc[len(results_df)] = [
        n_estimators, learning_rate, subsample, colsample_bytree, scale_pos_weight,
        auc_train, auc_test1, auc_test2
    ]

    # Save progress after each iteration
    results_df.to_csv(csv_filename, index=False)
```

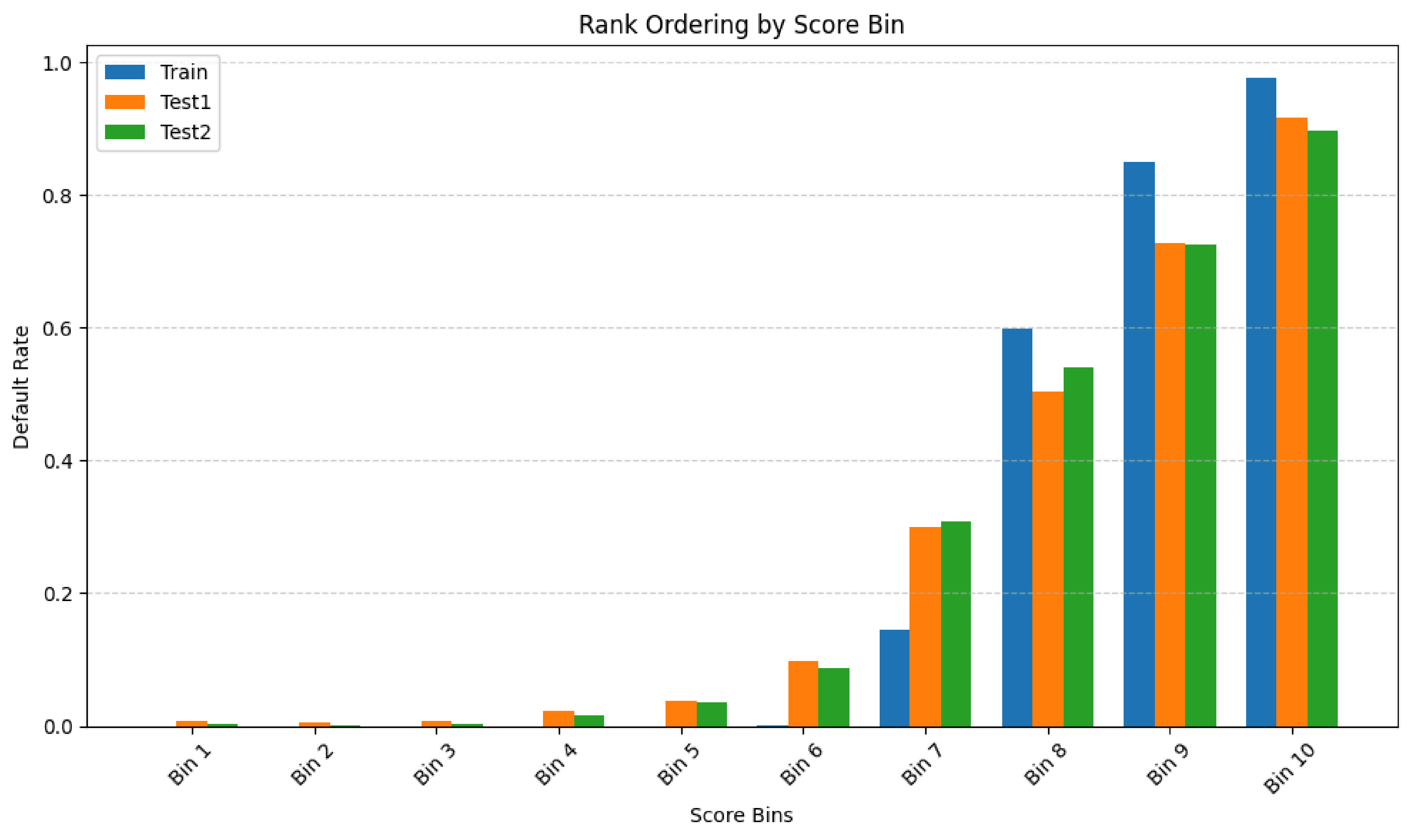
XGBOOST - GRID SEARCH



- Best tradeoff model has a high average AUC and a low standard deviation (orange point in first plot).
- Overfitting check: The red-marked model has high train AUC but still maintains a good Test 2 AUC.
- Low variance models are preferable to ensure stability across different datasets.
- More trees (~250+) increase train AUC, but may risk overfitting if test AUC drops.
- Final choice should balance high AUC and minimal overfitting, prioritizing Test 2 AUC consistency.
- The orange-highlighted model in the first plot seems to be the best tradeoff between performance and generalization.

XGBOOST – FINAL MODEL

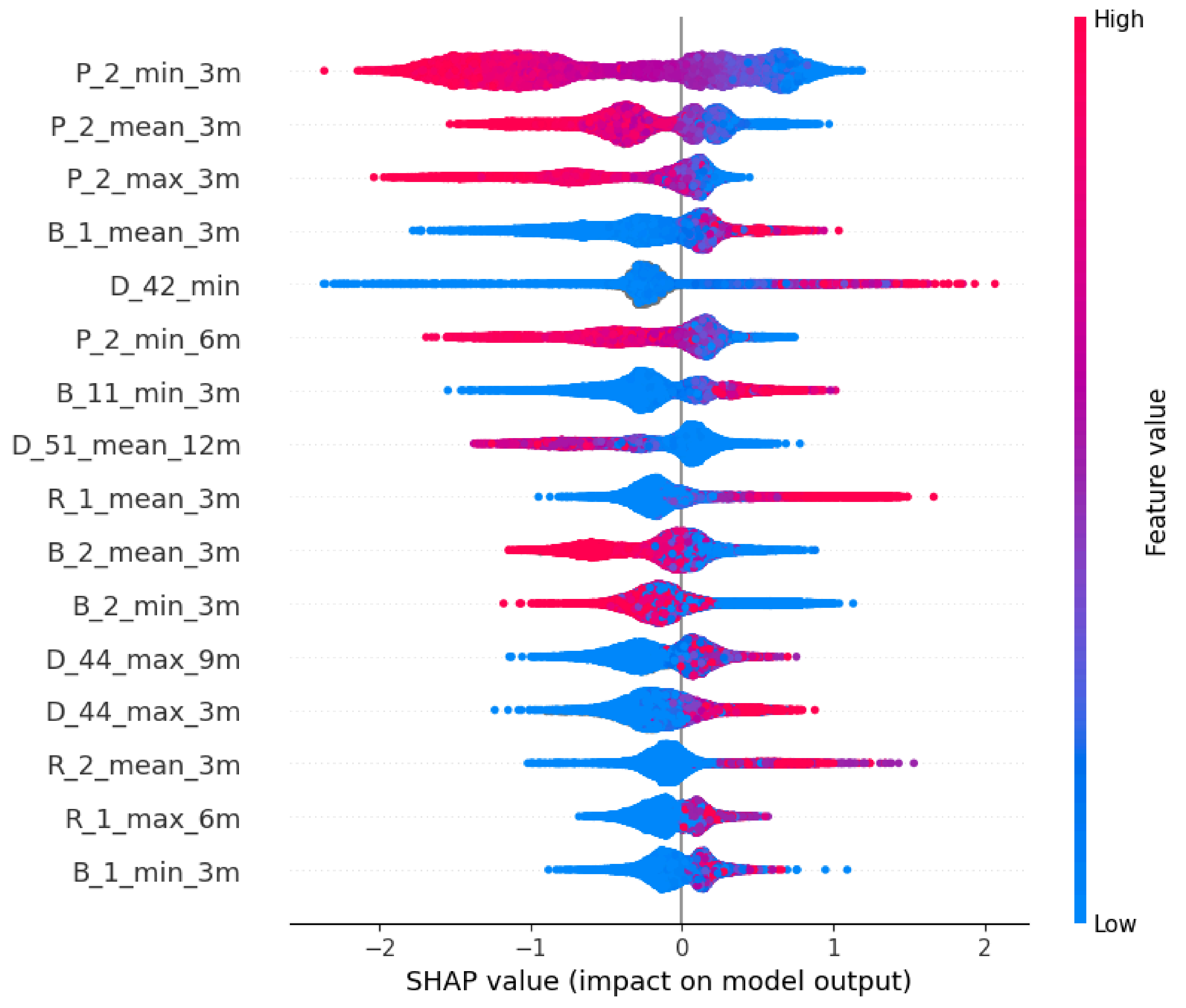
```
Best Model Parameters:
Trees                100.000000
LR                     0.100000
Subsample              0.800000
% Features             0.500000
Weight of Default      1.000000
AUC Train              0.955013
AUC Test 1             0.942248
AUC Test 2             0.940117
avg_auc_test           0.941182
Name: 42, dtype: float64
AUC Train: 0.9550
AUC Test 1: 0.9422
AUC Test 2: 0.9401
```



Final Results showing AUC value

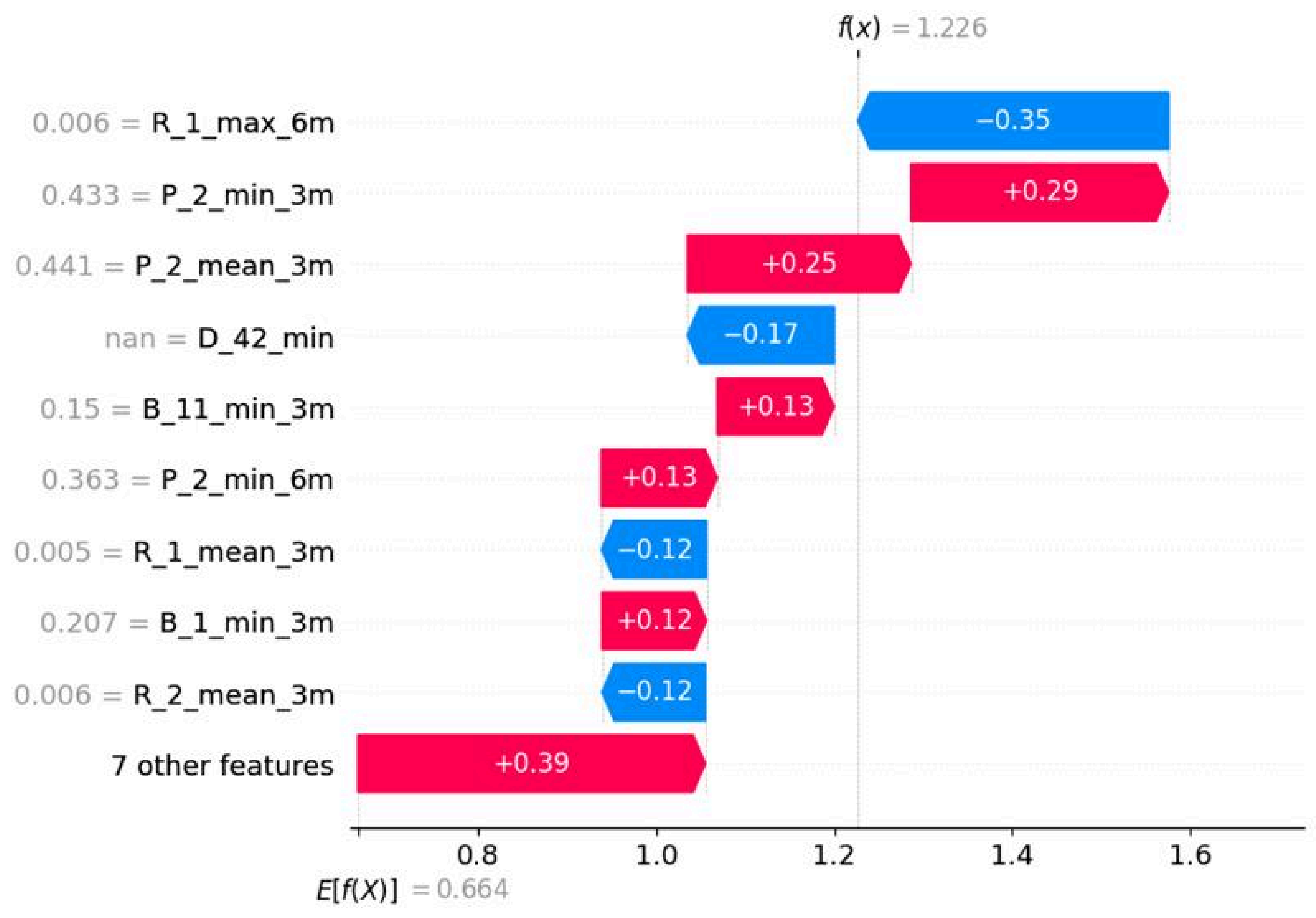
XGBOOST – SHAP ANALYSIS

- **Key Risk Drivers**: P_2_min_3m, P_2_mean_3m, and P_2_max_3m are the most influential features in credit risk prediction.
- **Feature Impact**: Higher P_2-related values increase risk, while higher B_1_mean_3m and R_1_mean_3m reduce risk.
- **Risk Patterns**: D_42_min (lower values) is linked to lower risk, while P_2_min_3m (higher values) signals higher risk.
- **Nonlinear Effects**: Some features impact risk both positively and negatively, indicating complex interactions.
- **Feature Importance**: Wider SHAP spreads (e.g., P_2_min_3m) show stronger predictive power.



SHAP Analysis Beewarm

XGBOOST – SHAP ANALYSIS



SHAP Analysis -Waterfalll

- **Most influential feature:** The "7 other features" group contributes +0.39 to the prediction, indicating multiple features play a key role in credit risk assessment.
- **Largest individual impact:** R_1_max_6m has the most significant negative impact (-0.35), suggesting higher values reduce risk.
- **Positive risk indicators:** P_2_mean_3m (+0.25) and B_11_min_3m (+0.13) contribute positively to risk, meaning higher values in these features may indicate higher default probability.
- **Negative risk indicators:** D_42_min (-0.17) and R_1_mean_3m (-0.12) reduce predicted risk, implying lower values in these features may be associated with lower risk.
- **Feature interactions matter**: The combination of multiple factors influences risk, highlighting the need for a well-balanced feature selection strategy.

NEURAL NETWORK – DATA PROCESSING

Code:

```python
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler

# Step 1: Impute Missing Values with 0
imputer = SimpleImputer(strategy='constant', fill_value=0)
X_train_imputed = imputer.fit_transform(X_train_selected)
X_test1_imputed = imputer.transform(X_test1_selected)
X_test2_imputed = imputer.transform(X_test2_selected)

# Step 2: Outlier Treatment (Cap at 1st and 99th percentiles)
def cap_and_floor(X):
    lower = np.percentile(X, 1, axis=0)
    upper = np.percentile(X, 99, axis=0)
    return np.clip(X, lower, upper)

X_train_clipped = cap_and_floor(X_train_imputed)
X_test1_clipped = cap_and_floor(X_test1_imputed)
X_test2_clipped = cap_and_floor(X_test2_imputed)

# Step 3: Normalize with StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_clipped)
X_test1_scaled = scaler.transform(X_test1_clipped)
X_test2_scaled = scaler.transform(X_test2_clipped)
```

Summary Statistics:

```python
import pandas as pd

# Assuming X_train_selected is a DataFrame and column names are preserved
feature_names = X_train_selected.columns if hasattr(X_train_selected, 'columns') else [f'Var_{i}' for i in range(X_train_scaled.shape[1])]

# Convert scaled data back into DataFrame
X_train_scaled_df = pd.DataFrame(X_train_scaled, columns=feature_names)

# Summary statistics
summary_table = X_train_scaled_df.describe().T[['mean', 'std', 'min', 'max']]
summary_table.to_csv("nn_data_summary_table.csv")

print("Summary table saved as 'nn_data_summary_table.csv'")
summary_table.head()
```

[190] ✓ 0.0s

```
Summary table saved as 'nn_data_summary_table.csv'
```

	mean	std	min	max
R_2_mean_3m	6.237488e-17	1.000008	-0.320241	4.383465
R_1_mean_3m	1.548313e-17	1.000008	-0.437572	4.924374
P_2_max_3m	2.603377e-16	1.000008	-2.606098	1.372813
P_2_min_3m	5.857045e-16	1.000008	-2.628908	1.439420
D_48_mean_3m	3.317813e-18	1.000008	-1.068821	1.979503

Feature Range:

```python
X_train_clipped_df = pd.DataFrame(X_train_clipped, columns=feature_names)
range_table = pd.DataFrame({
    'Min (Clipped)': X_train_clipped_df.min(),
    'Max (Clipped)': X_train_clipped_df.max()
})
range_table.to_csv("nn_feature_ranges_post_clipping.csv")

print("Clipped feature range table saved as 'nn_feature_ranges_post_clipping.csv'")
range_table.head()
```

✓ 0.0s

```
Clipped feature range table saved as 'nn_feature_ranges_post_clipping.csv'
```

	Min (Clipped)	Max (Clipped)
R_2_mean_3m	0.001280	1.005843
R_1_mean_3m	0.001364	1.169727
P_2_max_3m	-0.003139	1.008454
P_2_min_3m	-0.101383	1.001340
D_48_mean_3m	0.000000	1.016596

NEURAL NETWORK - GRID SEARCH

Code:

```
Best Neural Network Hyperparameters Chosen:
HL                              4
# Node                          4
Activation Function          tanh
Dropout                        0%
Batch Size                    100
AUC Train                0.943532
AUC Test1                0.944757
AUC Test2                0.943101
Avg AUC                  0.943797
AUC Variance             0.000001
Name: 22, dtype: object
```

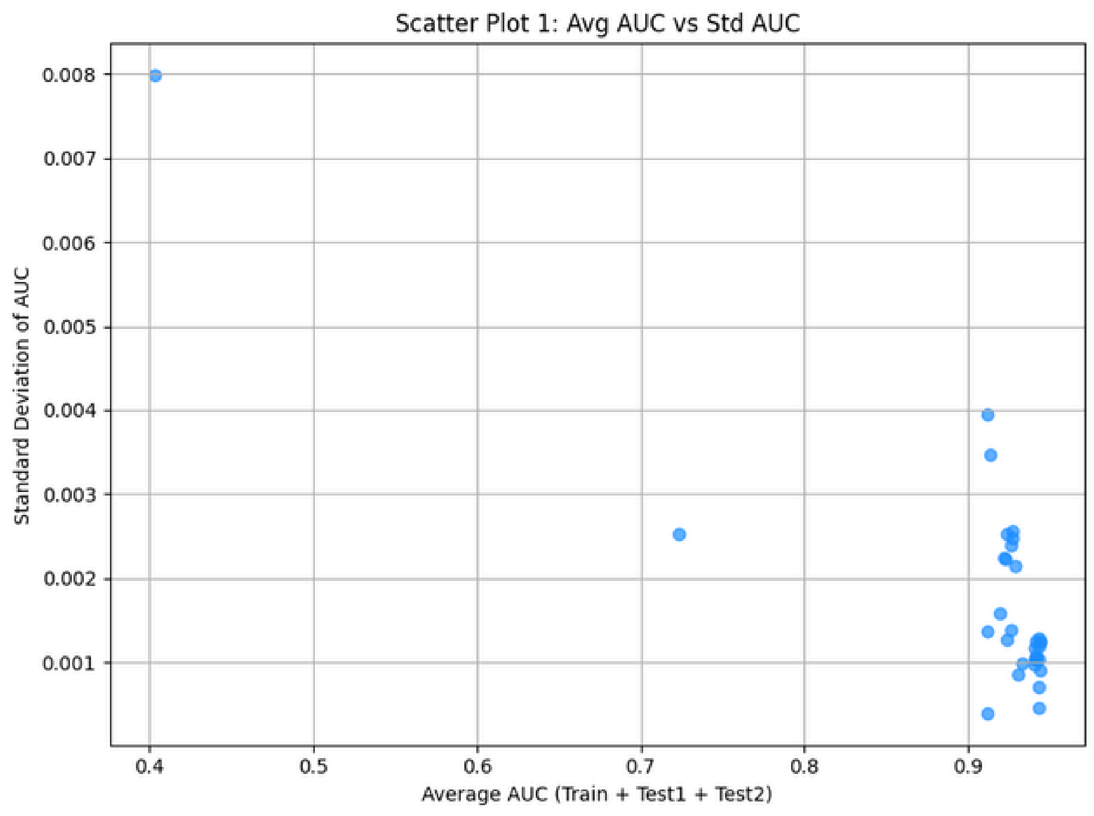
Best Parameter Selection

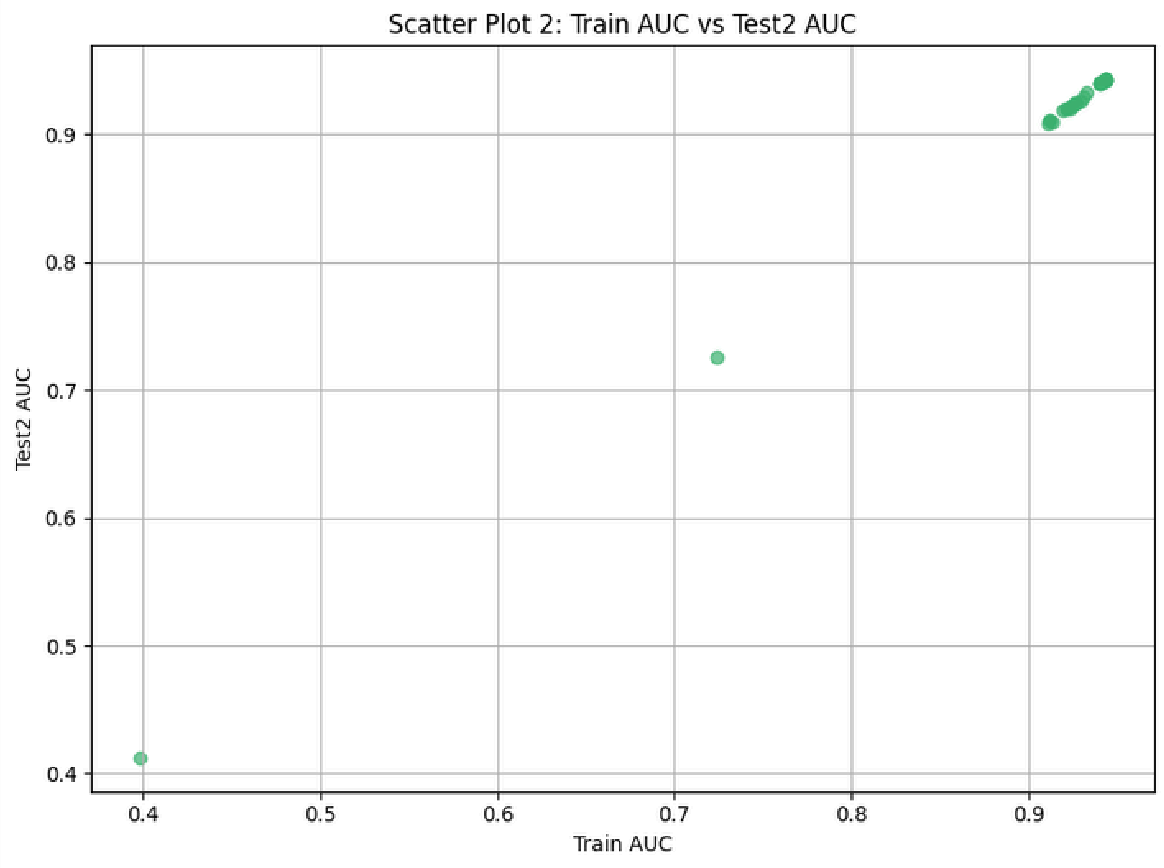
```python
import numpy as np
import pandas as pd
import os
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import BinaryCrossentropy
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import roc_auc_score

# === Grid Search Settings ===
nn_hidden_layers_list = [2, 4]
nn_nodes_list = [4, 6]
nn_activation_list = ['relu', 'tanh']
nn_dropout_list = [0.5, 0.0]  # 50%, 100% (no dropout)
nn_batch_size_list = [100, 10000]
nn_epochs = 20
nn_results_path = 'nn_grid_results.csv'
nn_detailed_log_path = 'nn_grid_detailed_log.csv'

# === Load Previous Summary if it Exists ===
if os.path.exists(nn_results_path):
    result_df_nn = pd.read_csv(nn_results_path)
else:
    result_df_nn = pd.DataFrame(columns=[
        'HL', '# Node', 'Activation Function', 'Dropout', 'Batch Size',
        'AUC Train', 'AUC Test1', 'AUC Test2'
    ])

# === Create Detailed Log File if Not Exists ===
if not os.path.exists(nn_detailed_log_path):
    pd.DataFrame(columns=result_df_nn.columns).to_csv(nn_detailed_log_path, index=False)

# === Function to Build NN ===
def build_model_nn(n_hidden, n_nodes, activation, dropout, input_dim):
    model = Sequential()
    model.add(Dense(n_nodes, activation=activation, input_dim=input_dim))
    if dropout > 0:
        model.add(Dropout(dropout))
    for _ in range(n_hidden - 1):
        model.add(Dense(n_nodes, activation=activation))
        if dropout > 0:
            model.add(Dropout(dropout))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer=Adam(), loss=BinaryCrossentropy(), metrics=[])
    return model

# === Grid Search Loop ===
for nn_hl in nn_hidden_layers_list:
    for nn_node in nn_nodes_list:
        for nn_act in nn_activation_list:
            for nn_drop in nn_dropout_list:
                for nn_bs in nn_batch_size_list:

                    # Skip already trained combos
                    if ((result_df_nn['HL'] == nn_hl) &
                        (result_df_nn['# Node'] == nn_node) &
                        (result_df_nn['Activation Function'] == nn_act) &
                        (result_df_nn['Dropout'] == f"{int(nn_drop*100)}%") &
                        (result_df_nn['Batch Size'] == nn_bs)).any():
                        continue

                    print(f"Training NN | HL:{nn_hl}, Nodes:{nn_node}, Act:{nn_act}, Dropout:{nn_drop}, Batch:{nn_bs}")

                    # === Build + Train ===
                    model_nn = build_model_nn(nn_hl, nn_node, nn_act, nn_drop, X_train_scaled.shape[1])
                    early_stop = EarlyStopping(monitor='loss', patience=3, restore_best_weights=True)

                    model nn.fit(
```

```python
                    print(f"Training NN | HL:{nn_hl}, Nodes:{nn_node}, Act:{nn_act}, Dropout:{nn_drop}, Batch:{nn_bs}")

                    # === Build + Train ===
                    model_nn = build_model_nn(nn_hl, nn_node, nn_act, nn_drop, X_train_scaled.shape[1])
                    early_stop = EarlyStopping(monitor='loss', patience=3, restore_best_weights=True)

                    model_nn.fit(
                        X_train_scaled, y_train,
                        batch_size=nn_bs,
                        epochs=nn_epochs,
                        verbose=0,
                        callbacks=[early_stop]
                    )

                    # === Evaluate ===
                    try:
                        y_train_pred = model_nn.predict(X_train_scaled)
                        y_test1_pred = model_nn.predict(X_test1_scaled)
                        y_test2_pred = model_nn.predict(X_test2_scaled)

                        auc_train = roc_auc_score(y_train, y_train_pred)
                        auc_test1 = roc_auc_score(y_test1, y_test1_pred)
                        auc_test2 = roc_auc_score(y_test2, y_test2_pred)
                    except Exception as e:
                        print(f"Error computing AUC: {e}")
                        auc_train, auc_test1, auc_test2 = np.nan, np.nan, np.nan

                    # === Store Results ===
                    new_row = pd.DataFrame([{
                        'HL': nn_hl,
                        '# Node': nn_node,
                        'Activation Function': nn_act,
                        'Dropout': f"{int(nn_drop * 100)}%",
                        'Batch Size': nn_bs,
                        'AUC Train': auc_train,
                        'AUC Test1': auc_test1,
                        'AUC Test2': auc_test2
                    }])

                    # Save to results
                    result_df_nn = pd.concat([result_df_nn, new_row], ignore_index=True)
                    new_row.to_csv(nn_detailed_log_path, mode='a', header=False, index=False)

 === Final Save ===
esult_df_nn.to_csv(nn_results_path, index=False)
rint("✅ Grid Search Complete. Results saved:")
rint(f"Summary: {nn_results_path}")
rint(f"Details: {nn_detailed_log_path}")
```

NEURAL NETWORK - GRID SEARCH

- **Extensive Grid Search:** Explored 32 combinations using 2–4 hidden layers, 4–6 neurons, two activations (relu, tanh), dropout options (0%, 50%), and batch sizes (100, 10,000).
- **Performance Metrics:** Evaluated models on AUC across Train, Test1, and Test2 datasets. Selected the best model using average AUC and variance.
- **Best Model Selected**: The final NN model had low variance and high average AUC, indicating good generalization across samples.
- **Bias-Variance Tradeoff:** Top-performing models were not always the ones with highest training AUC-models with lower variance across datasets were preferred.
- **Lessons Learned:** Large batch sizes occasionally sped up training but didn't always improve performance. Dropout was helpful in reducing overfitting.
- **Comparison:** Despite good NN performance, XGBoost slightly outperformed NN in terms of average AUC, confirmed via SHAP analysis.





NEURAL NETWORK – FINAL MODEL

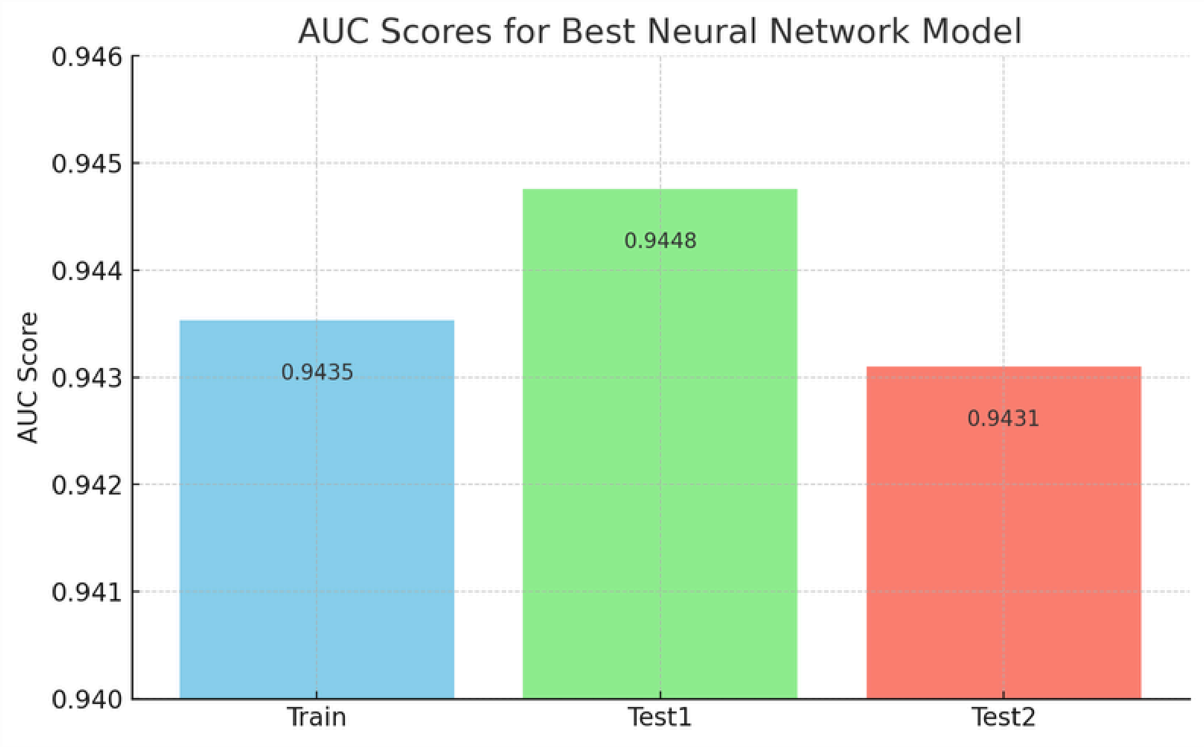
```
Best Neural Network Hyperparameters Chosen:
HL                            4
# Node                        4
Activation Function        tanh
Dropout                      0%
Batch Size                  100
AUC Train              0.943532
AUC Test1              0.944757
AUC Test2              0.943101
Avg AUC                0.943797
AUC Variance           0.000001
Name: 22, dtype: object
```

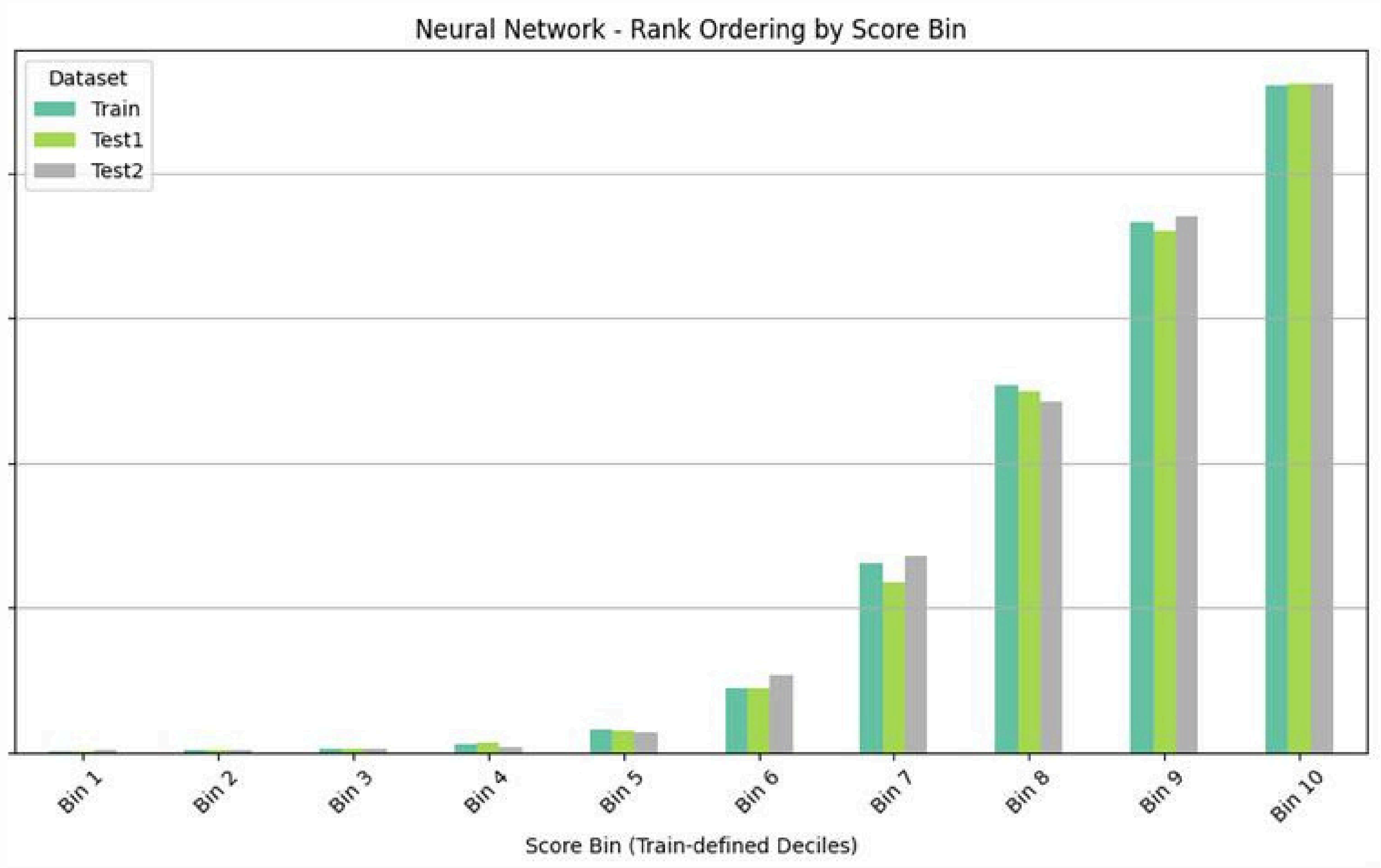
- **Optimal Architecture**: The best-performing model has 4 hidden layers with 4 neurons per layer, using the tanh activation function.
- **Regularization & Training Setup:** The model does not use dropout (0%), has a batch size of 100, and was trained with a fixed 20 epochs.
- **Stability of Model**: The low variance in AUC across train and test sets suggests the model generalizes well without overfitting.

.

Final AUC result

Graphical representation of best model





FINAL MODEL

- For our final model, we evaluated XGBoost and Neural Networks to determine the most effective approach for predicting credit risk.
- After comparing both models based on bias and variance, we selected **XGBoost** as the best-performing model.

Reasons:

- Although both XGBoost and the Neural Network demonstrated similar AUC scores, XGBoost outperformed the Neural Network on both the Train and Test 1 datasets. A higher AUC reflects better capability in correctly classifying positive and negative instances.
- XGBoost also exhibited a more favorable bias-variance tradeoff and proved to be computationally more efficient, especially on large datasets. Unlike Neural Networks, XGBoost can handle missing values natively, eliminating the need for imputation and reducing preprocessing time.
- Considering its speed, efficiency, and ability to simplify the process of evaluating customer acceptance and rejection, XGBoost is the preferred model for this task.

Model	AUC
Neural Net	TRAIN: 0.9435 TEST 1: 0.9448 TEST 2: 0.9431
XGBoost	TRAIN: 0.9550 TEST 1: 0.9456 TEST 2: 0.9429

```
xgb_results_df
```

	trees	lr	subsample	%_features	weight_of_default	auc_train	auc_test_1	auc_test_2
0	300	0.1	0.8	0.5	5.0	0.976477	0.945653	0.942955

```
best_model_params
```

```
Trees                  300.000000
LR                       0.100000
Subsample                0.800000
% Features               0.500000
Weight of Default        5.000000
AUC Train                0.976477
AUC Test 1               0.945653
AUC Test 2               0.942955
Avg AUC Test             0.955028
AUC Test Variance        0.000347
Name: 67, dtype: float64
```

```python
# === Calculate Avg AUC for NN and XGB ===
nn_avg_auc = nn_results_df["avg_auc"].iloc[0]
xgb_avg_auc = xgb_results_df['avg_auc_test'].iloc[0]

# === Compare Models Based on AUC (or any other metric you choose) ===
if nn_avg_auc > xgb_avg_auc:
    best_model = "Neural Network (NN)"
    best_model_details = nn_results_df
else:
    best_model = "XGBoost (XGB)"
    best_model_details = xgb_results_df

# === Save Best Model's Information ===
best_model_details.to_csv("best_model_details.csv", index=False)

# === Print the Best Model ===
print(f"The best model is: {best_model}")
print(f"Best model details saved to 'best_model_details.csv'")
```

```
The best model is: XGBoost (XGB)
Best model details saved to 'best_model_details.csv'
```

Codes showing the best model

STRATEGY

- **Conservative Strategy** (Threshold = **0.05**): Prioritizes financial stability by minimizing defaults, ensuring long-term sustainability, especially in uncertain economic conditions.The projected revenue is around
- **Aggressive Strategy** (Threshold = **0.08**): Focuses on maximizing loan approvals and revenue, accepting higher risk to expand the customer base, often suitable for growth-oriented market conditions.

Final Thoughts:

- At the conservative threshold of 0.05, the projected revenue is around $6,500, while at the 0.08 threshold, it's only about $230 more.(Based on train).
- Therefore, choosing the conservative approach makes sense as it balances risk and revenue effectively. This way, we can prioritize minimizing losses while still making substantial revenue, ensuring financial stability.

	TRAIN			TEST 1			TEST 2		
Threshold	**Default Rate**	**Revenue**	**#Total**	**Default Rate**	**Revenue**	**#Total**	**Default Rate**	**Revenue**	**#Total**
0.1	0	4669.81	31942	0.0163	1128.55	7932	0.0193	1138.21	7920
0.2	8.43E-05	5190.52	35591	0.0293	1222.66	8715	0.0327	1229.25	8675
0.3	0.0009	5507.63	37799	0.0436	1283.35	9265	0.0489	1283.25	9210
0.4	0.0025	5740.60	39458	0.0597	1331.17	9773	0.0661	1331.73	9718
0.5	0.0060	5944.33	40995	0.0761	1367.94	10221	**0.0829**	1370.31	10199
0.6	0.0129	6122.66	42552	**0.0978**	1405.62	10750	0.1048	1403.13	10713
0.7	0.0264	6322.94	44546	0.1251	1439.03	11348	0.1322	1438.68	11318
0.8	0.0508	6535.72	47277	0.1552	1467.87	11971	0.1650	1460.91	11961
0.85	0.0713	6662.6	49287	0.1745	1475.56	12320	0.1835	1468.87	12303
0.89	**0.0965**	6767.00	51467	0.1887	1482.21	12580	0.1993	1475.00	12598
0.9	0.1049	6793.28	52157	0.1920	1483.01	12641	0.2032	1475.63	12666

Thank you!