Data Mining Algorithms for Credit Default Risk

Max Boulat, Tanya Neustice, and Beakal Zekaryas

Department of Engineering, San Diego University

AAI-500-01: Probability and Statistics for Artificial Intelligence

Leonid Shpaner

June 23, 2025

Abstract

The final team project uses the analysis of modeling techniques for consumer credit default. The objective was to develop a statistical analysis of Logistic Regression and Naïve Bayesian Classification modeling by using payment behavior to predict the likelihood of default on credit payments. The response variable was a binary indicator of default, and 23 explanatory variables were used to build and compare models. After thorough data analysis, both models were implemented and evaluated. The Logistic Regression model outperformed the Naïve Bayesian Classification approach in terms of prediction accuracy and model interpretability and was therefore selected as the final model. This report documents the probability distribution analysis, validates the final model selection with statistical evidence, and assesses the strengths and weaknesses of both models when analyzing credit default risk.

Keywords: credit default, logistic regression, Naïve Bayesian Classification analysis, binary classification, predictive modeling

Credit Default Predictions

Accurately predicting the credit default rate is critical for financial institutions in helping with risk management. This research is an analysis of evaluating data mining algorithms of a dataset of 30,000 credit card clients with 23 explanatory variables ranging from demographics, age, marital status, education, credit limits, and default indicators (UCI Machine Learning Repository, 2005).

The goal was to analyze the statistical performance of the Logistic Regression versus Naïve Bayesian Classification models in predicting default credit risk using various data mining algorithms and examining which results were clearer interpreting the coefficients.

The final model selected was the Logistic Regression model due to its stronger statistical performance and consistency with binary outcome. As supported by Agresti and Kateri (2022), statistical methods applied to binary and categorical data must be evaluated not only by goodness-of-fit metrics but also by the suitability of the modeling technique to the response variable structure.

Our hypothesis was as follows:

- Null Hypothesis (H_0): There is no statistically significant relationship between the predictor variables and the likelihood of default.
- Alternative Hypothesis (H_1): There exists a statistically significant relationship between one or more predictor variables and the likelihood of default.

Data Cleaning and Preparation

Data cleaning preparation (was applied to Logistic Regression) and utilized for effective statistical modeling. This step was necessary to ensure accurate data mining algorithm interpretations of results. Several preprocessing steps were applied to the original data. The preprocessing of the data was needed to improve clarity and increase the interpretability of explanatory variables. This research employed a binary variable, default payment (Yes = 1, No = 0), as the response variable. This study reviewed the literature and used the following 23 variables as explanatory variables as shown below.

- LIMIT_BAL: Amount of the given credit (NT dollar): it includes both the individual consumer credit and his/her family (supplementary) credit.
- SEX: Gender (1 = male; 2 = female).
- EDUCATION: Education (1 = graduate school; 2 = university; 3 = high school; 4 = others).
- MARRIAGE: Marital status (1 = married; 2 = single; 3 = others).
- AGE: Age (year).

PAY_0-PAY_6: History of past payment. The past monthly payment records (from April to September 2005) are as follows:

- 0 = The Repayment Status in September 2005;
- 2 = The Repayment Status in August 2005;
- 6 = The Repayment Status in April 2005.

The measurement scale for the repayment status is:

- -1 = Pay Duly;

- 1 = Payment Delay for one month;
- 2 = Payment Delay for two months;
- 8 = Payment Delay for eight months;
- 9 = Payment Delay for nine months and above.

BILL_AMT1-BILL_AMT6: Amount of bill statement (NT dollar).

- 1 = Amount of Bill Statement in September 2005;
- 2 = Amount of Bill Statement in August 2005;
- 6 = Amount of Bill Statement in April 2005.

PAY_AMT1-PAY_AMT6: Amount of previous payment (NT dollar).

- 1 = Amount Paid in September 2005;
- 2 = Amount Paid in August 2005;
- 6 = Amount Paid in April 2005.

Consolidation of bill and pay amounts was applied. The original dataset contained one separate column for each pay and bill amount, starting at the most recent and spanning 6 months of history. To quantify the effect of payment amounts and bill amounts as unitary variables, a weighted average was calculated for each subject, using a linear decay factor (highest weight for most recent).

Segmentification of certain continuous variables into contiguous segments. The effect of certain continuous variables was hard to interpret because of the small size of the granularity. The data was derived and adjusted into four equal-sized bins. This transformation enhanced interpretability and mitigated the influence of outliers. The following variables were consolidated into a reduced number of contiguous blocks:

- AGE
- LIMIT_BAL
- WEIGHTED_BILL_AMT
- WEIGHTED_PAY_AMT

The boundaries between the blocks were made to match the quartiles for each series. Quartile ranges were determined as follows:

- AGE: Overall Range: 21-79
    - 21–28;
    - 28–34;
    - 34–41;
    - 41–79.
- LIMIT_BAL: Overall Range: 10,000-1,000,000
    - 10,000–50,000;
    - 50,000–140,000;
    - 140,000–240,000;
    - 240,000–1,000,000.
- WEIGHTED_BILL_AMT / PAY_AMT: Overall Range: -29464.95-873217.38
    - -29464.95-4888.90;
    - 1228.08-2488.14;
    - 2488.14-5696.19;

  - 5696.19-805849.48.

Removal of redundant payment history variables were made. The PAY_2 - PAY-6 variables are duplicative of PAY_0, since they measure payment delay in months for multiple months in a row. Since there is not much additional insights to be gained from them beyond what is already included in PAY_0, and to avoid cannibalizing coefficient capital from other more meaningful variables, a decision was made to remove them altogether.

Replacing -1 with 0 in PAY_0 was changed. -1 means “payed duly”. The next possible value is 1 which indicates “1 month behind”. To ensure a consistent step size from baseline for interpreting the odds ratio in logistic regression, we re-coded -1 to 0.

Separation between the training set and the testing set was applied. To evaluate the effectiveness of the logistic regression, we subdivided the data into two parts:

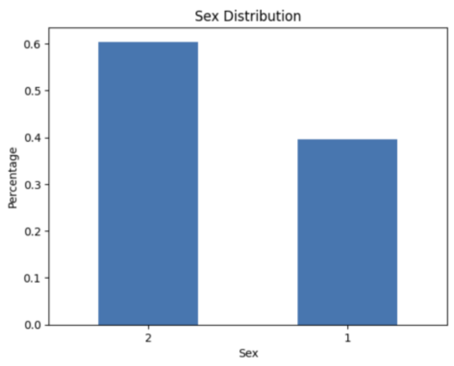
- A training sample containing 70% of samples from the original dataset selected at random
- A testing sample containing the remaining 30% of the samples

This division ensured unbiased model evaluation and reduced the risk of overfitting during training.

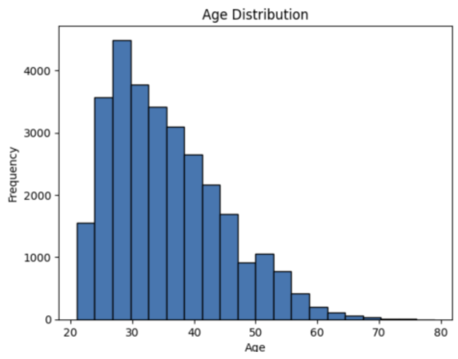
Exploratory Data Analysis

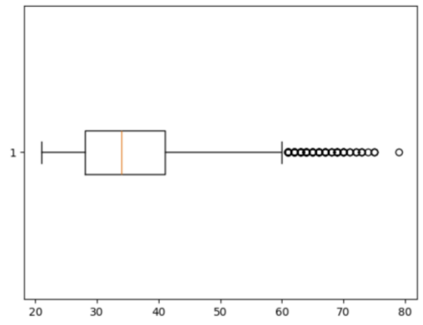
Descriptive statistics and visualizations were run on the dataset as shown below.

Sex Distribution: The categorical variable of SEX was defined as males given the value of 1 and females given the value of 2. A bar chart shows 60.4% were female and 39.6% were male with females being the majority.

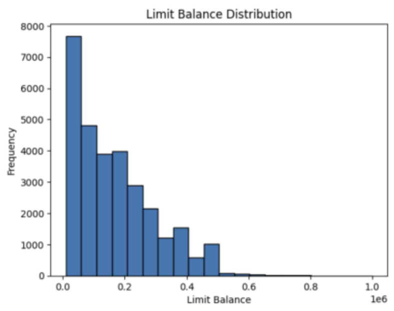


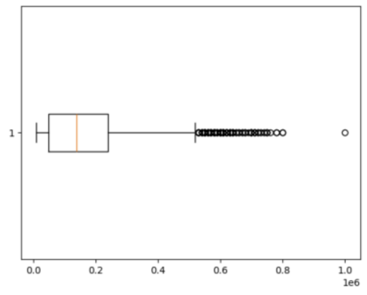
Age Distribution: The continuous variable of AGE showed a right skewed distribution with the majority of ages between 20-40. A box plot showed some outliers.



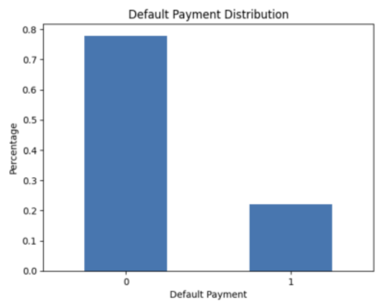


Limit Balance: Showed a heavy right skew distribution on the histogram with a few outliers with a higher balance.

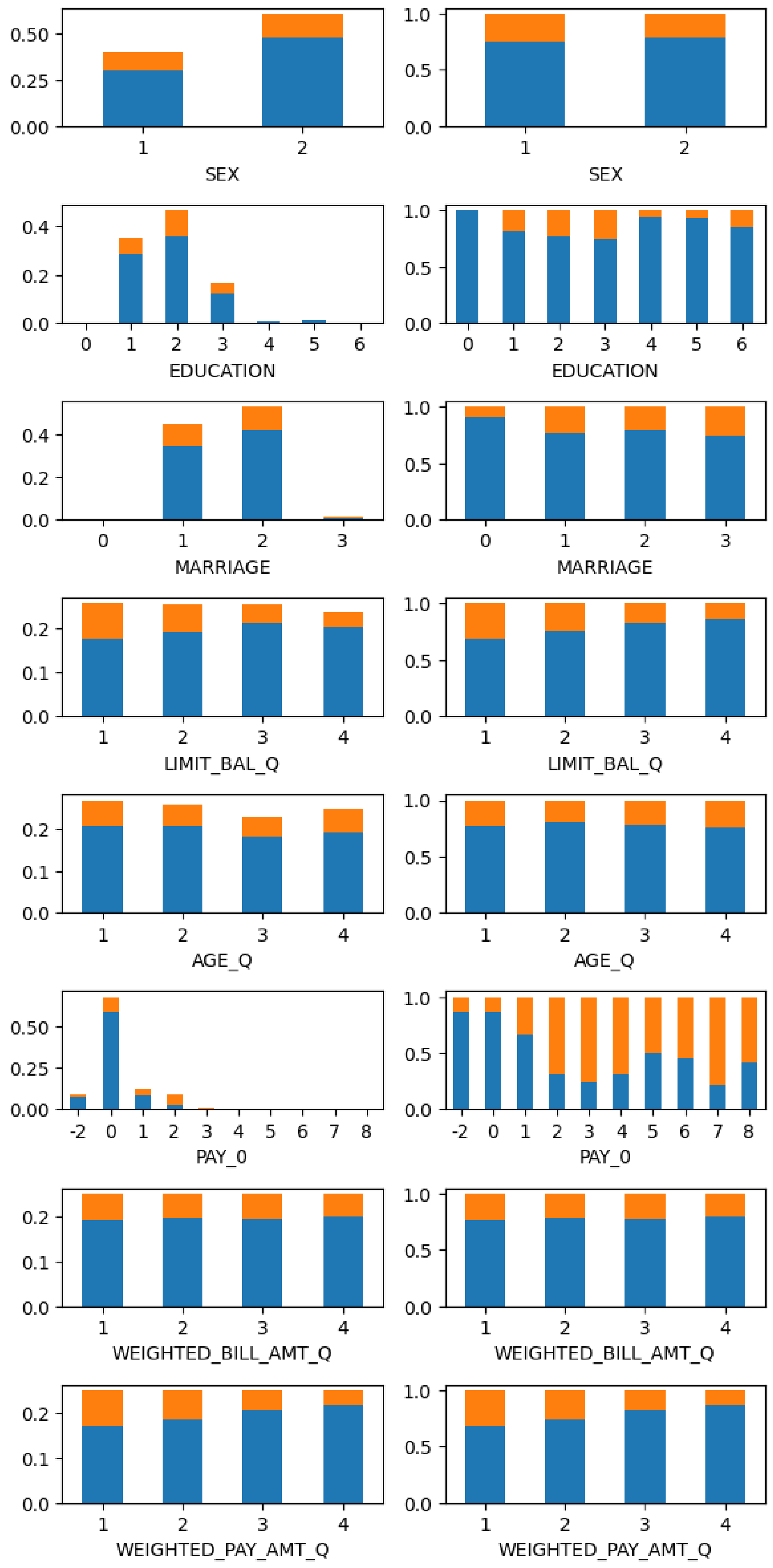




Defaults: A binary variable of 1 was given for default, and a value of 0 was given for no default. A bar chart showed 22% had defaulted, and 78% did not default. A discrepancy was noted.

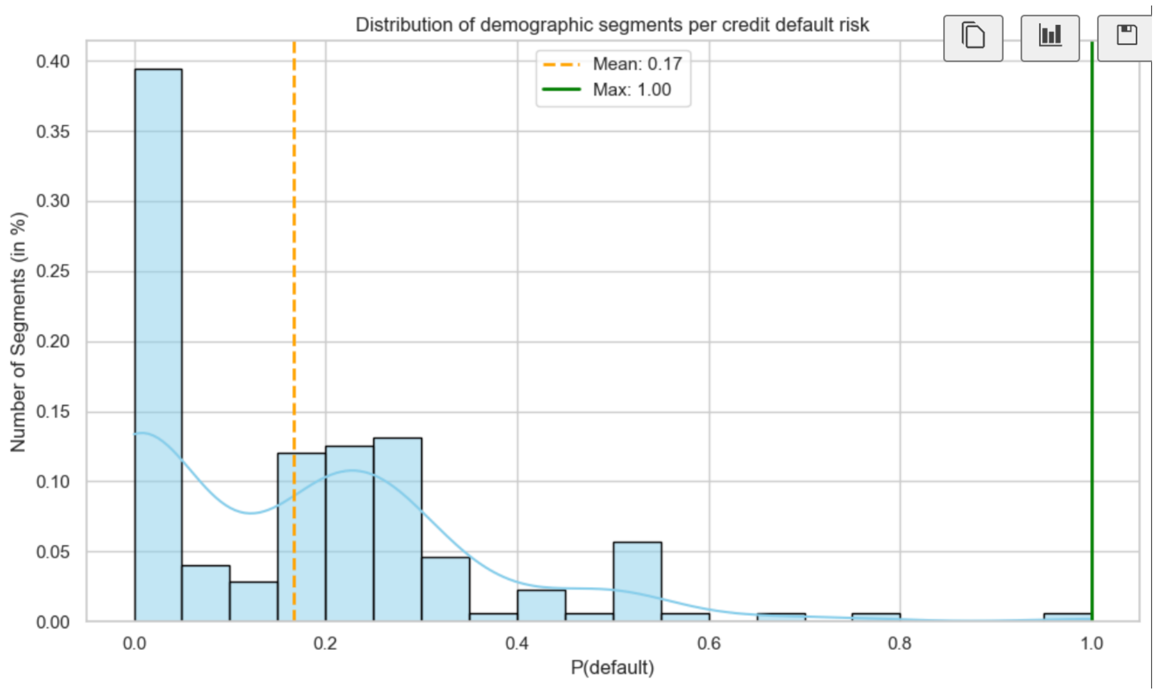


Correlations: The following shows the distribution of the dataset for each variable plotted as a bar graph with percentages on the right and concrete quantities on the left.


SEX
EDUCATION
MARRIAGE
LIMIT_BAL_Q
AGE_Q
PAY_0
WEIGHTED_BILL_AMT_Q
WEIGHTED_PAY_AMT_Q

In this chart we subdivided the samples into distinct demographic segments characterized by unique combinations of Sex, Education, Marriage and Age. We then calculated the credit default rate for each segment and plotted the count of segments for each probability on a histogram.

The resulting distribution is skewed to the right with 40% of the segments having near 0 credit default risk and 12% of the segments having average credit default risk. This shows that demographic variables have a meaningful effect on credit default risk, else we would see a normal distribution centered around the mean.



Model Selection and Analysis

The two statistical models chosen were Logistic Regression and Naïve Bayesian Classification. Both models were used on the dataset and evaluated to see which model would be better at predicting credit default risk for the 23 explanatory variables on the dataset.

Naïve Bayesian Classifier

This report provides an analysis of credit card default prediction using a Naive Bayes classifier applied to a dataset of credit card clients. The model was trained to identify patterns of default based on key demographic and financial variables, with a focus on education level, marital status, sex, and age group. The findings aim to uncover disparities in default risk and inform data-driven credit risk assessment.

Methodology

A Gaussian Naive Bayes model was trained on a dataset consisting of 30,000 clients, using financial and demographic predictors. The dataset was cleaned and transformed, including one-hot encoding of categorical features (SEX, EDUCATION, MARRIAGE) and binning of age into defined groups (AGE_GROUP). The target variable was binary (default: 1 = default, 0 = no default).

The dataset was split into training and testing sets using a 70:30 ratio, and model performance was evaluated using accuracy and a classification report (precision, recall, F1-score).

Model Performance Summary

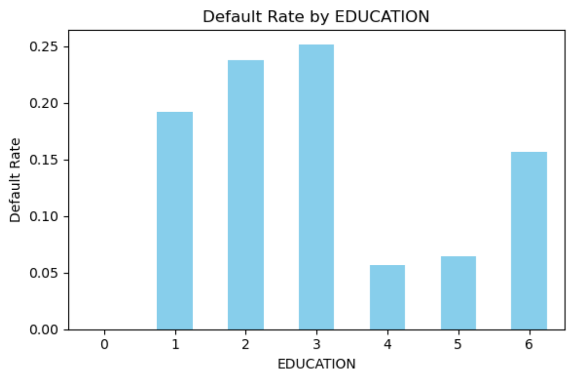
- Overall Accuracy: 0.378 (or 37.8%)
- Classification Report:

Class	Precision	Recall	F1-score
No Default (0)	0.88	0.24	0.27
Default (1)	0.24	0.88	0.38

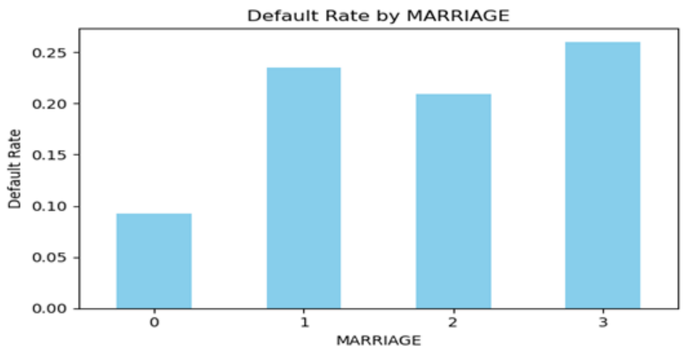
The classifier shows high recall for the default class (1), meaning it captures most of the actual defaulters (88%). However, precision is low (24%), which indicates many false positives. The model performs poorly at predicting non-defaulters. This pattern reflects the conservative nature of Naive Bayes, which prioritizes catching positive cases (defaults) but shows weak ability to detect actual non-defaulters or false alarms.

Demographic Model Analysis

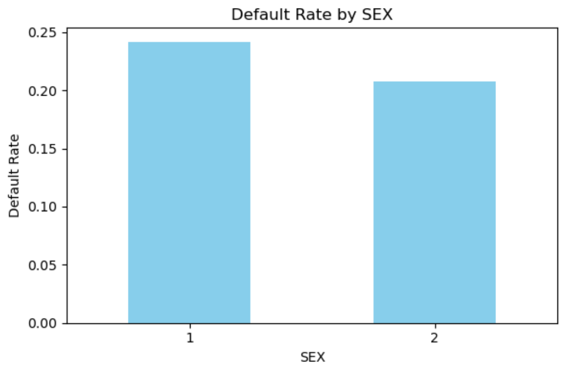
The default rate seems to decrease with higher education levels. Clients with only high school education or lower showed significantly higher default rates than those with university or graduate degrees. This aligns with the hypothesis that financial literacy and income stability may increase with education.



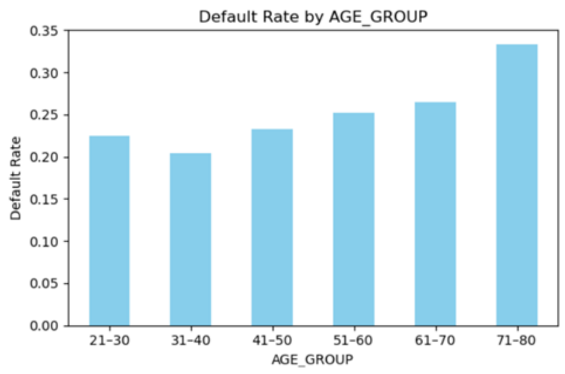
Divorced\Widowed clients showed slightly higher default rates than married or single clients. This could reflect different levels of financial responsibility or dual income benefits in married households.



The difference in default rates between male and female clients looks marginal. However, females had a slightly lower default rate, indicating potential gender-related behavioral or income differences.



Older adults (71–80) show a surprisingly high default rate, which could be due to fixed income, retirement, or small sample bias. Older adults have a moderately high default rate, this is consistent with financial maturity increasing with age, as younger clients may be more credit inexperienced. Younger adults (21–30) have a moderate default rate, possibly due to limited credit experience.



Logistic Regression

Methology

The frequentist approach to credit default risk assessment is through models which identify the effects of each explanatory variable (while adjusting for the others) on credit default risk, resulting in a prediction equation which looks like:

$$E(Y_i) = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}$$

where:

- x_{ij} is the value of explanatory variable j for subject i
- p is the count of explanatory variables

Linear models assume a normal distribution and constant variance in the response variable at each combination of explanatory variables.

In the case of predicting credit defaults, the response variable is a Bernoulli variable and the standard linear regression model is inadequate because:

- the normality assumption is violated because the distribution is binomial
- the constant variance assumption is also violated because its variance depends on its mean: $\mu(1 - \mu)$

- the predicted values are not constrained to [0, 1]

In this case we turn to GLMs (Generalized Linear Models), which are a family of models which extend the linear regression dynamics to fit data which badly violates the assumptions of the normal linear model.

GLMs are characterized by three components:

- the random component (the response variable)
- the systematic component (the explanatory variable)
- the link function (a transformation function which allows the predicted value to be non-linearly related to the explanatory variables)

For Bernoulli variables such as the credit default variable, we need to model the probability that the outcomes fall in one of two categories: success or failure. This makes this problem a classification problem and one for which the logistic regression model is particularly well suited.

The link function of logistic regression models is called **the logit** and can be expressed as:

$$\log\left(\frac{\mu_i}{1-\mu_i}\right)$$

Therefore, the full prediction equation is:

$$\log\left(\frac{\mu_i}{1-\mu_i}\right) = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}$$

The results produced by the logistic regression model would need to be converted back to a probability of default π by reversing the effects of the logit function:

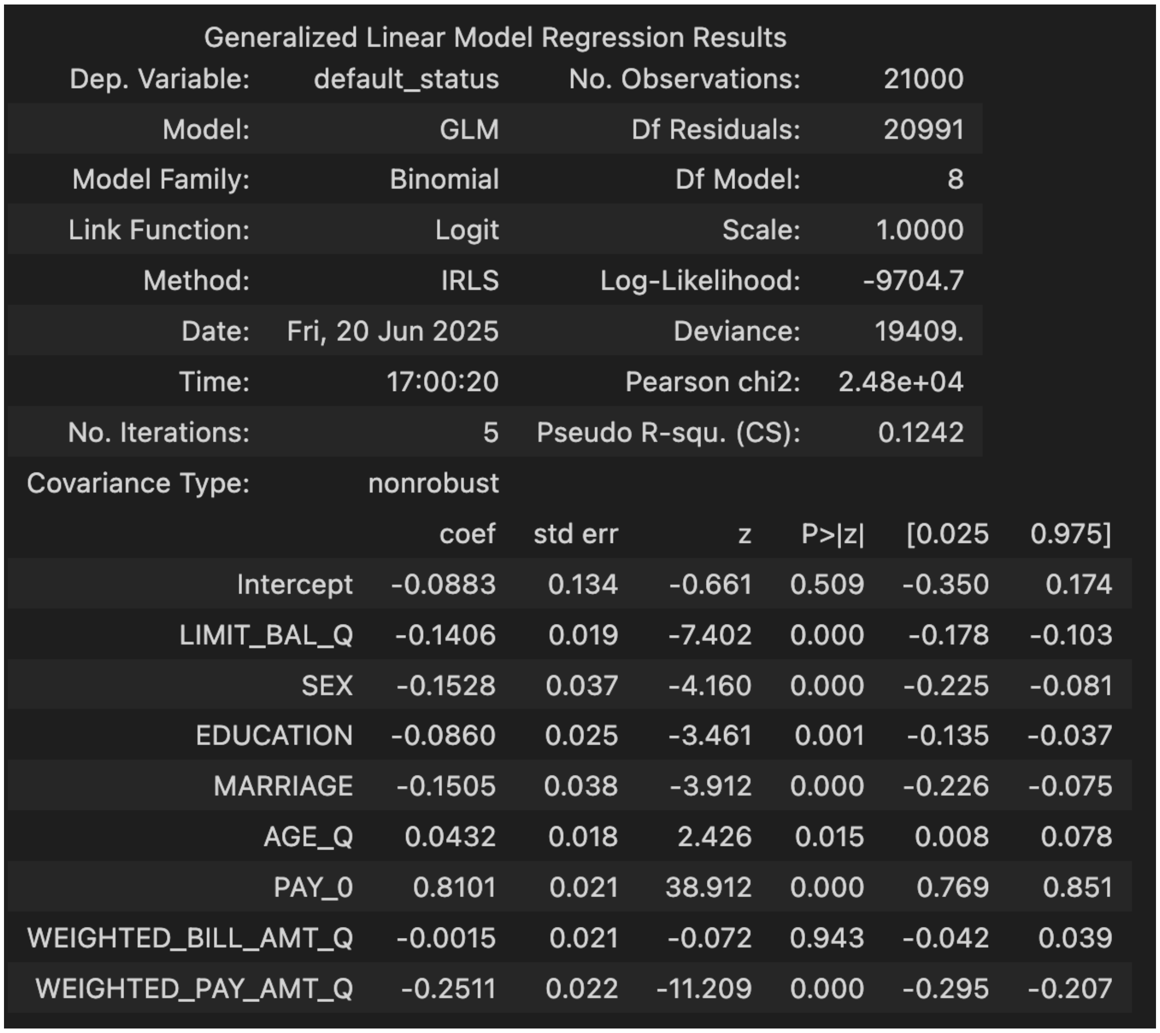
$$\pi_i = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip})}}$$

Another way to read the model is to characterize the change in Y based on a one-unit increment of a given explanatory variable, this is done by looking at the individual coefficients themselves: $\beta_1, \dots, \beta_p$. The coefficients give you the log-odds of changes for each explanatory variable (adjusting for the others). Reversing the transformation gives you the corresponding odds ratio, answering the question "by how much do the odds of default change when that explanatory variable changes?"

$$odds\,ratio_i = e^{\beta_i}$$

Results

Figure ?? Shows the raw results of running the model on the *train* dataset.



```
                 Generalized Linear Model Regression Results
  Dep. Variable:      default_status    No. Observations:      21000
          Model:                 GLM        Df Residuals:      20991
   Model Family:            Binomial            Df Model:          8
  Link Function:               Logit               Scale:     1.0000
         Method:                IRLS      Log-Likelihood:    -9704.7
           Date:    Fri, 20 Jun 2025            Deviance:     19409.
           Time:            17:00:20        Pearson chi2:   2.48e+04
 No. Iterations:                   5  Pseudo R-squ. (CS):     0.1242
Covariance Type:           nonrobust
                           coef   std err        z   P>|z|   [0.025   0.975]
           Intercept    -0.0883     0.134   -0.661   0.509   -0.350    0.174
         LIMIT_BAL_Q    -0.1406     0.019   -7.402   0.000   -0.178   -0.103
                 SEX    -0.1528     0.037   -4.160   0.000   -0.225   -0.081
           EDUCATION    -0.0860     0.025   -3.461   0.001   -0.135   -0.037
            MARRIAGE    -0.1505     0.038   -3.912   0.000   -0.226   -0.075
               AGE_Q     0.0432     0.018    2.426   0.015    0.008    0.078
               PAY_0     0.8101     0.021   38.912   0.000    0.769    0.851
WEIGHTED_BILL_AMT_Q     -0.0015     0.021   -0.072   0.943   -0.042    0.039
 WEIGHTED_PAY_AMT_Q     -0.2511     0.022  -11.209   0.000   -0.295   -0.207
```

The model gives us the estimated effect for each explanatory variable, along with a set of metrics which indicate the statistical significance of the effect and their confidence levels.

These metrics are:

- the standard error
- the z-score
- the p-value
- the confidence interval

The standard error measures the sampling distribution's normalized deviation around the true proportion of defaults if the effect for that variable was 0 (adjusting for the effect of all other variables), a phenomenon denoted as:

$$H_0: \beta_i = 0$$

and the z-score gives us the number of standard errors away from where it would be under H_0. The larger the z-score, the less likely it is due to normal sampling variations.

The p-value gives us the probability of arriving at an effect of that magnitude if the true effect was 0 (adjusting for the effects of all the other variables). In this study, we use $\alpha = 0.05$, therefore, all p-values below 0.05 indicate that the effect is statistically significant.

Out of all the explanatory variables, only WEIGHTED_BILL_AMT_Q was not statistically significant (p-value = 0.990).

Among the remaining variables, we observed the following effects, ranked by strength:

- PAY_0 has the strongest effect on credit default likelihood: for each additional month that a client is late, the odds of defaulting **increase by 133%**.
- PAY_AMT (weighted and split into quartiles) has the second strongest effect. For each increase of one quartile in the adjusted amount paid, the odds of defaulting **decrease by 23%**. This indicates that people who pay small amounts (like the minimum amount) are more likely to default than people who make bigger payments.
- MARRIAGE has the third strongest effect. The odds of defaulting **decrease by 14%** when the client is married compared to being single.

- LIMIT_BAL_Q has the fourth strongest effect. For each quartile-over-quartile increase in credit limits, the odds of defaulting **decrease by 13%**. This indicates that people with small credit limits are more likely to default than people who have higher credit limits.
- SEX has the fifth strongest effect, with the odds of defaulting **decreasing by 11%** when the client is female.
- EDUCATION has the sixth strongest effect, with odds of default **decreasing by 6%** with each increment.
- AGE has the weakest effect on credit defaults, with each quartile-over-quartile increment **increasing the risk of default by 4%**.

To illustrate the practical implementation of the model, we created 5 fictitious individuals and evaluated their risk of default based on their attributes:

Description	Attributes	Credit default probability
Jake is a recent graduate who likes surfing	age: 1 sex: 1 education: 0 marriage: 0 limit_balance: 1 bill_amount: 2 payment_amount: 0 payment_history: 0	39.87%
John is a gambling addict who maxed out all his cards	age: 1 sex: 1 education: 4 marriage: 3 limit_balance: 1 bill_amount: 4 payment_amount: 0 payment_history: 8	99.66%

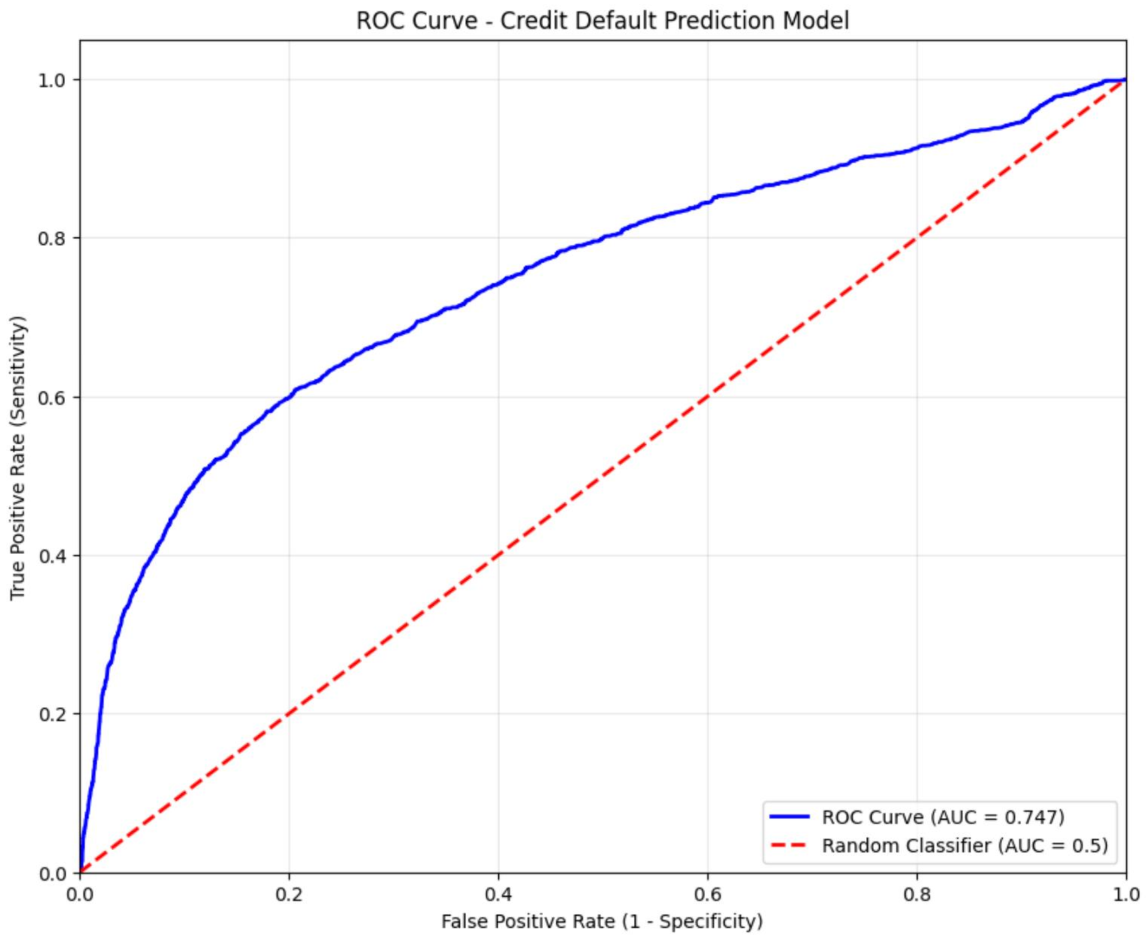
Penelope is a wealthy retiree with pristine financial discipline	age: 4 sex: 2 education: 1 marriage: 1 limit_balance: 4 bill_amount: 1 payment_amount: 3 payment_history: 0	13.98%
Ricardo is a trust fund kid who never learned responsibility	age: 1 sex: 1 education: 1 marriage: 0 limit_balance: 4 bill_amount: 4 payment_amount: 1 payment_history: 6	98.07%
Stella is a struggling single mother working three jobs	age: 2 sex: 2 education: 3 marriage: 2 limit_balance: 1 bill_amount: 1 payment_amount: 1 payment_history: 0	22.67%

Model Validation

Performing credit default classification on the 30% of samples set aside for testing allowed us to assess the performance of the model on previously unseen data. The results were as follows:

- **Accuracy**: 80.89% of the model's predictions were correct.
- S**ensitivity (recall)**: The percentage of actual defaulters that were correctly identified by the model (true positive rate) is 24.5%.
- **Specificity**: The percentage of non-defaulters that were correctly identified by the model (true negative rate) is 97%.

- **Positive Predictive Value**: The probability that a client actually defaulted given that the model predicted a default (precision) is 69.7%.
- **ROC Curve and AUC**: We plotted the ROC (Receiver Operating Characteristic) curve to visualize the model's performance across all classification thresholds. The AUC (Area Under the Curve) was 0.75, which indicates fair discriminatory ability.



Conclusions and Recommendations

According to Logistic Regression, interpretation of model coefficients revealed that PAY_0, WEIGHTED_PAY_AMT_Q, and MARRIAGE were the most influential predictors of credit default risk. Notably, each additional month of delayed payment (PAY_0) increased the odds of default by 133%, while higher average payment amounts and being married reduced the odds of default.

These metrics indicate that the Logistic Regression model was particularly strong in identifying non-defaulters (high specificity). The limitations showed its sensitivity, leaving an area for improvement. Precision was high, suggesting that the model was generally correct when it predicted a default.

Naïve Bayesian Classification model achieved a higher recall (88%) but showed its limitations from lower accuracy (37.8%). This was mainly due to poor precision, and its strong assumptions of conditional independence. According to Agresti and Kateri (2022), Naive Bayes often underperforms when features are highly correlated, which is likely in this financial context.

The recommendations based on the findings and model analysis, are the following:

- Model Variations: Explore more advanced model algorithms.
- Enhanced Variables: Adding variables such as income level, employment stability, or length of credit history to improve results.
- Bias Auditing: Assess the model for potential bias across demographic groups (e.g., sex, age, marital status) to ensure fairness and compliance with ethical AI standards since there were more females than males on the dataset.

- Resampling Techniques: Apply oversampling balancing techniques to address the default class imbalance.

Overall, while the Logistic Regression model provides more consistent results for credit risk prediction, continued refinement and ethical evaluation will ensure ethical and responsible analysis in the future.

References

Agresti, A., & Kateri, M. (2022). *Foundation of statistics for data scientists: With R and Python*. CRC Press.

UCI Machine Learning Repository. (2005). *Default of credit card clients dataset*. https://archive.ics.uci.edu/dataset/350/default+of+credit+card+clients

Appendix

Attach PDF

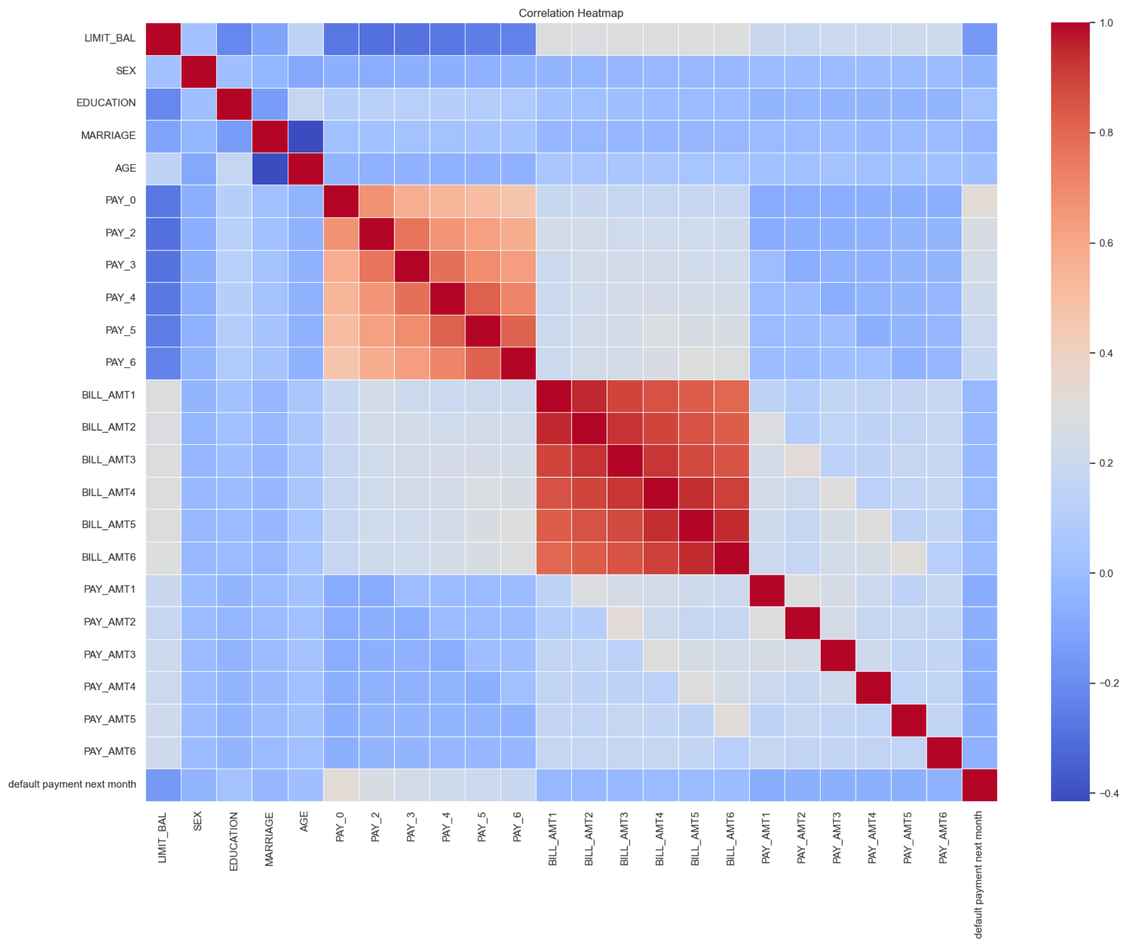
FinalTeamProject

June 21, 2025

```
[240]: import pandas as pd
       import matplotlib.pyplot as plt
       import seaborn as sns
       import numpy as np
```

```
[241]: df = pd.read_csv('Datasets/Credit.csv')
```

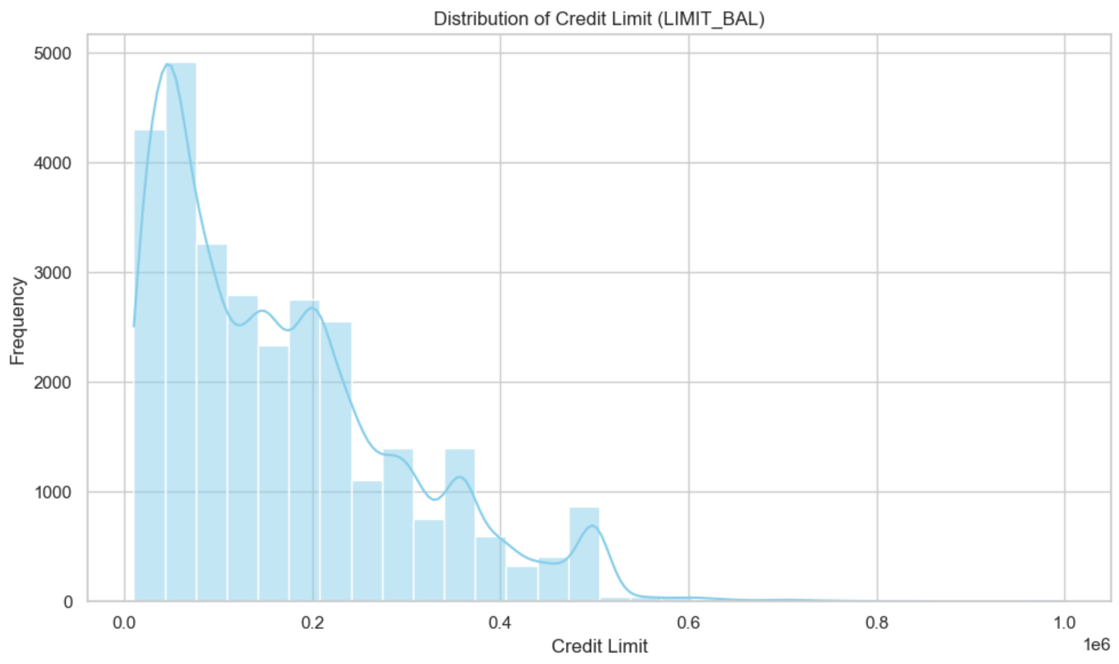
```
[242]: # Load the dataset -- Skip first metadata row
       df = pd.read_csv('Datasets/Credit.csv')

       # Drop the ID column as it's not useful for analysis
       df_cleaned = df.drop(columns=["ID"])

       # Set Seaborn style
       sns.set(style="whitegrid")

       # Generating Correlation Heatmap
       plt.figure(figsize=(18, 14))
       correlation_matrix = df_cleaned.corr()
       sns.heatmap(correlation_matrix, annot=False, cmap="coolwarm", fmt=".2f",␣
        ↪linewidths=0.5)
       plt.title("Correlation Heatmap")
       plt.tight_layout()
       plt.show()

       # Printing my explanation
       print("explanation")
       print()
       print("The correlation heatmap of the dataset reveals relationships between␣
        ↪features such as: High correlation among BILL_AMT variables (e.g.,␣
        ↪BILL_AMT1, BILL_AMT2, etc.)")
       print("The correlation heatmap also indicates a positive correlation between␣
        ↪LIMIT_BAL and PAY_AMT values, a Mmoderate positive correlation between past␣
        ↪payment statuses (PAY_0 to PAY_6) and default likelihood")
```



```
explanation

The correlation heatmap of the dataset reveals relationships between features
such as: High correlation among BILL_AMT variables (e.g., BILL_AMT1, BILL_AMT2,
etc.)
The correlation heatmap also indicates a positive correlation between LIMIT_BAL
and PAY_AMT values, a Mmoderate positive correlation between past payment
statuses (PAY_0 to PAY_6) and default likelihood
```

[243]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset (assuming the file is in the same directory) --Skip first␣
↪metadata row
df = pd.read_csv('Datasets/Credit.csv')

# Drop the ID column as it's not useful for analysis
```

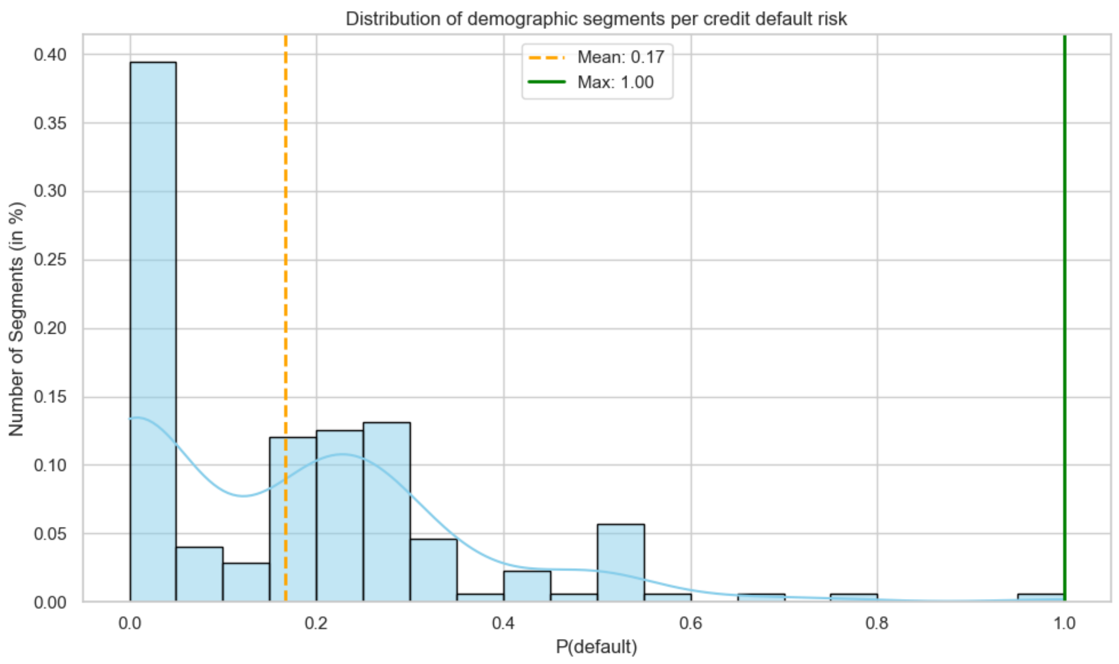
```python
df_cleaned = df.drop(columns=["ID"])

# Set Seaborn style
sns.set(style="whitegrid")

# --- Histogram of Credit Limit ---
plt.figure(figsize=(10, 6))
sns.histplot(df_cleaned["LIMIT_BAL"], bins=30, kde=True, color="skyblue")
plt.title("Distribution of Credit Limit (LIMIT_BAL)")
plt.xlabel("Credit Limit")
plt.ylabel("Frequency")
plt.tight_layout()
plt.show()

# Printing my explanation
print("Explanation -- The histogram of the credit limit (LIMIT_BAL)")
print()
print("Most credit limits are concentrated below 200,000 units.")
print("The distribution is right-skewed, indicating a smaller number of clients␣
 ↪with very high credit limits.")
```



```
Explanation -- The histogram of the credit limit (LIMIT_BAL)

Most credit limits are concentrated below 200,000 units.
The distribution is right-skewed, indicating a smaller number of clients with
very high credit limits.
```

[244]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df2 = pd.read_csv('Datasets/Credit.csv')

# Create AGE groups
age_bins = [20, 30, 40, 50, 60, 70, 80]
age_labels = ['20s', '30s', '40s', '50s', '60s', '70s']
df2['AGE_GROUP'] = pd.cut(df2['AGE'], bins=age_bins, labels=age_labels,␣
 ↪right=False)

# Define population segments
segment_columns = ['SEX', 'EDUCATION', 'MARRIAGE', 'AGE_GROUP']
segment_group = df2.groupby(segment_columns)

# Count total and on-time payments per segment
segment_stats = segment_group['default payment next month'].agg(
    total='count',
    default=lambda x: (x == 1).sum()
).reset_index()

# Calculate Probability
segment_stats['probability'] = segment_stats['default'] / segment_stats['total']

# Plot histogram with KDE
plt.figure(figsize=(10, 6))
sns.histplot(segment_stats['probability'], bins=20, kde=True, color='skyblue',␣
 ↪edgecolor='black', stat='probability')

# Add vertical lines
mean_prob = segment_stats['probability'].mean()
max_prob = segment_stats['probability'].max()

plt.axvline(mean_prob, color='orange', linestyle='--', linewidth=2,␣
 ↪label=f'Mean: {mean_prob:.2f}')
plt.axvline(max_prob, color='green', linestyle='-', linewidth=2, label=f'Max:␣
 ↪{max_prob:.2f}')

# Labels and legend
plt.title("Distribution of demographic segments per credit default risk")
plt.xlabel("P(default)")
plt.ylabel("Number of Segments (in %)")
plt.legend()
plt.tight_layout()
plt.show()
```

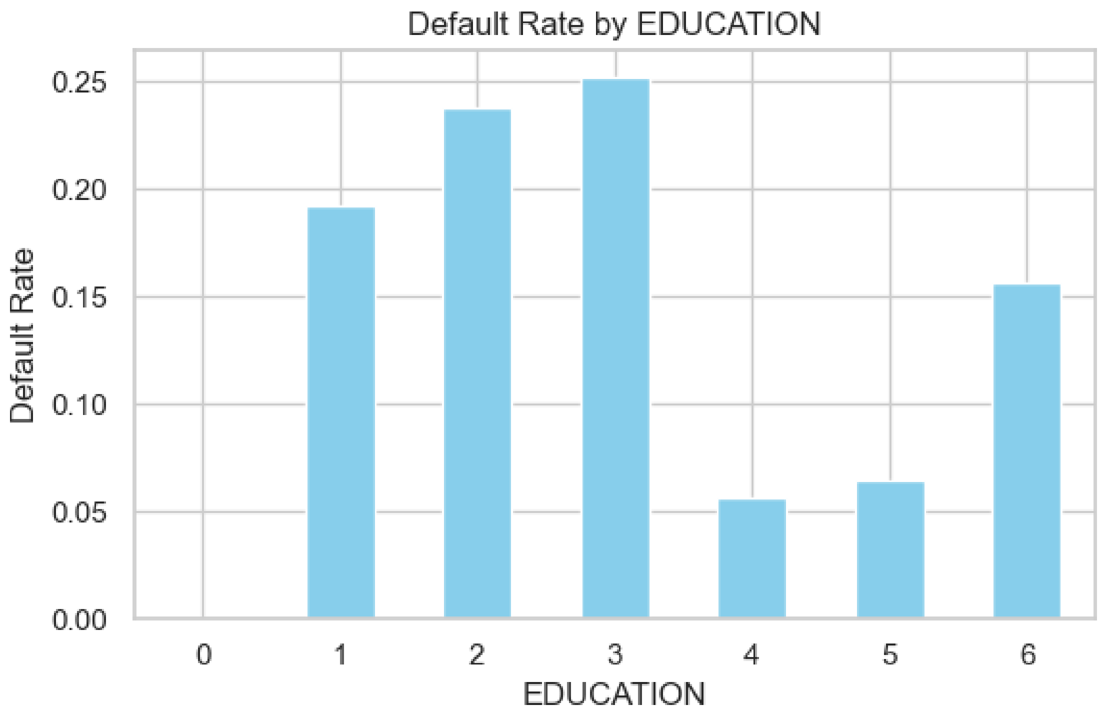
```
/var/folders/ck/sr6gtz6n0jx9dmp9nlplxl_w0000gn/T/ipykernel_65554/2844974901.py:1
4: FutureWarning: The default of observed=False is deprecated and will be
changed to True in a future version of pandas. Pass observed=False to retain
current behavior or observed=True to adopt the future default and silence this
warning.
  segment_group = df2.groupby(segment_columns)
```

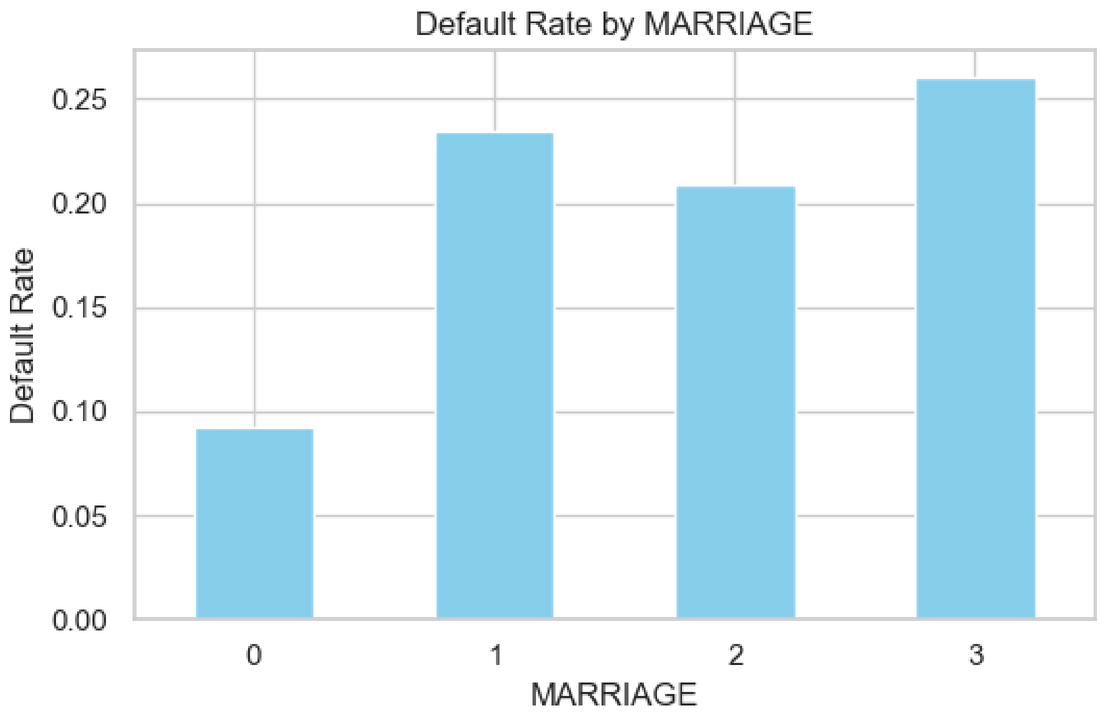


[245]:

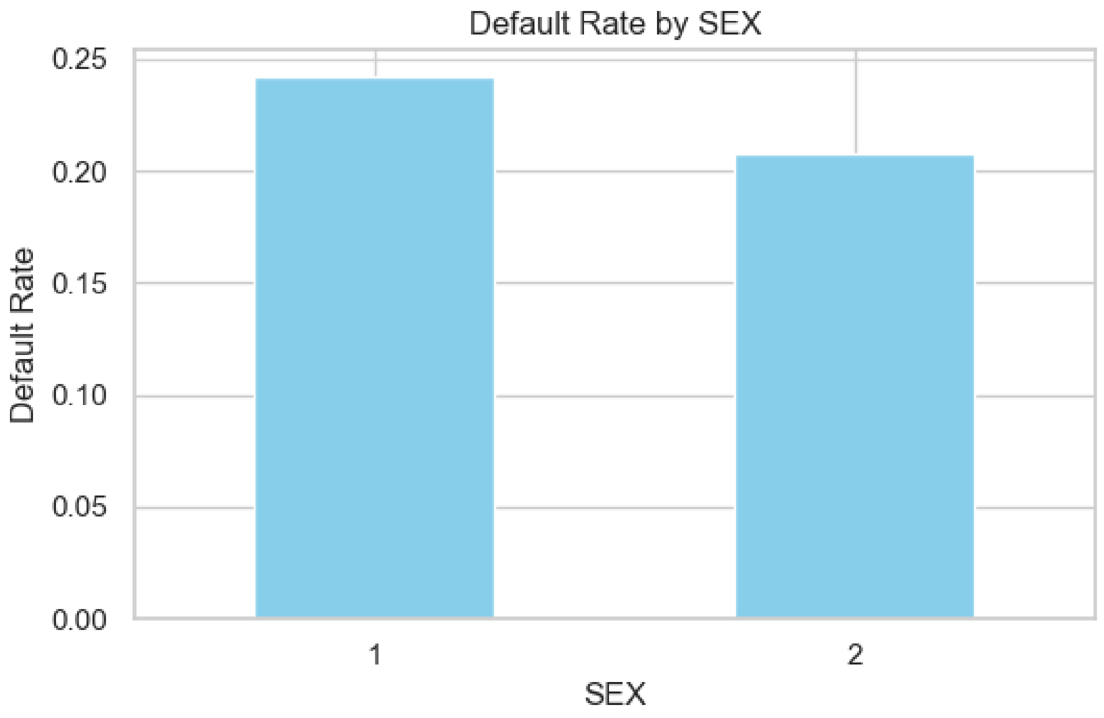
```python
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report, accuracy_score

# Load the data
df_bayes = pd.read_csv('Datasets/Credit.csv')

# Strip any whitespace from column names
df_bayes.columns = df_bayes.columns.str.strip()

# Rename columns for clarity
df_bayes.columns = ['ID', 'LIMIT_BAL', 'SEX', 'EDUCATION', 'MARRIAGE', 'AGE',
                   'PAY_0', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6',
                   'BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5',␣
 ↪'BILL_AMT6',
              'PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5',␣
 ↪'PAY_AMT6', 'default']

# Clean AGE column and create AGE_GROUP
df_bayes['AGE'] = pd.to_numeric(df_bayes['AGE'], errors='coerce')
df_bayes = df_bayes.dropna(subset=['AGE'])

age_bins = [20, 30, 40, 50, 60, 70, 80]
age_labels = ['21-30', '31-40', '41-50', '51-60', '61-70', '71-80']
df_bayes['AGE_GROUP'] = pd.cut(df_bayes['AGE'], bins=age_bins,␣
 ↪labels=age_labels)

# Generating the Plot default rates
for col in ['EDUCATION', 'MARRIAGE', 'SEX', 'AGE_GROUP']:
    plt.figure(figsize=(6, 4))
    df_bayes.groupby(col)['default'].mean().plot(kind='bar', color='skyblue')
    plt.title(f'Default Rate by {col}')
    plt.ylabel('Default Rate')
    plt.xlabel(col)
    plt.xticks(rotation=0)
    plt.tight_layout()
    plt.show()

# Define feature list
features = ['LIMIT_BAL', 'AGE', 'SEX', 'EDUCATION', 'MARRIAGE',
            'PAY_0', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6',
            'BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5',␣
 ↪'BILL_AMT6',
            'PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5',␣
 ↪'PAY_AMT6']

# Preparing features and target
X = df_bayes[features]
y = df_bayes['default']

# One-hot encode categorical variables
X = pd.get_dummies(X, columns=['SEX', 'EDUCATION', 'MARRIAGE'], drop_first=True)

# Splitting the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)

# Training Naive Bayes classifier
model = GaussianNB()
model.fit(X_train, y_train)

# Predict and evaluate
```

```
y_pred = model.predict(X_test)
y_proba = model.predict_proba(X_test)[:, 1]

print("\nAccuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Optional: Show sample predictions
sample = pd.DataFrame({
    'Actual': y_test.values[:5],
    'Predicted Probability': y_proba[:5]
})
print("\nSample Predictions:")
print(sample)

#Printing my explanation of the result-set based on the Naive Bayes classifier

print("Accuracy is 0.377888 -- This means 38%  of the customers were correctly␣
 ↪classified - either as likely to default (1) or not (0).")
print()
print("The report breaks down precision, recall, and F1-score for each class")
print()
print("For Class 0 -- No Default")
print("Precision = 0.88: 88% of those predicted as -- No Default were correct")
print("Recall = 0.24: 24% of the actual -- no default customers correctly␣
 ↪predicted.")
print("F1 = 0.37 -- Weak ability to detect actual non-defaulters.")
print()
print("For Class 1 -- Default")
print("Precision = 0.24: 24% of predicted defaulters were actually defaulters")
print("Recall = 0.88: 88% of actual defaulters -- Postive case of how many␣
 ↪prdicted to be defaulted")
print("F1 = 0.38: Weak ability to detect actual defaulter")
print(" Tha model is too conservative - reluctant to label someone as a␣
 ↪defaulter.")
print("For credit risk, recall on Class 1 is critical - you want to catch as␣
 ↪many defaulters as possible!")

print()
print()
print(" --- Sample Predictions ---")
print("Actual: The true class -- 0 = no default, 1 = default")
print("Predicted Probability: Model's confidence that the customer will␣
 ↪default")

print()
print("Row 0: True label is 0 (no default), model predicts 87% chance of␣
↪default -   correct and confident.")
print("Row 4: True label is 1 (default), model predicts 87% -   somewhat␣
↪confident, borderline.")

print()

print(" --- Recommendations --- ")
print("Improve recall on defaulters: Try different models like (e.g., logistic␣
↪regression, random forest), oversampling (SMOTE), or cost-sensitive learning.
↪")
print("Threshold tuning: Adjust default classification threshold (not just 0.5)␣
↪to balance precision/recall.")
```



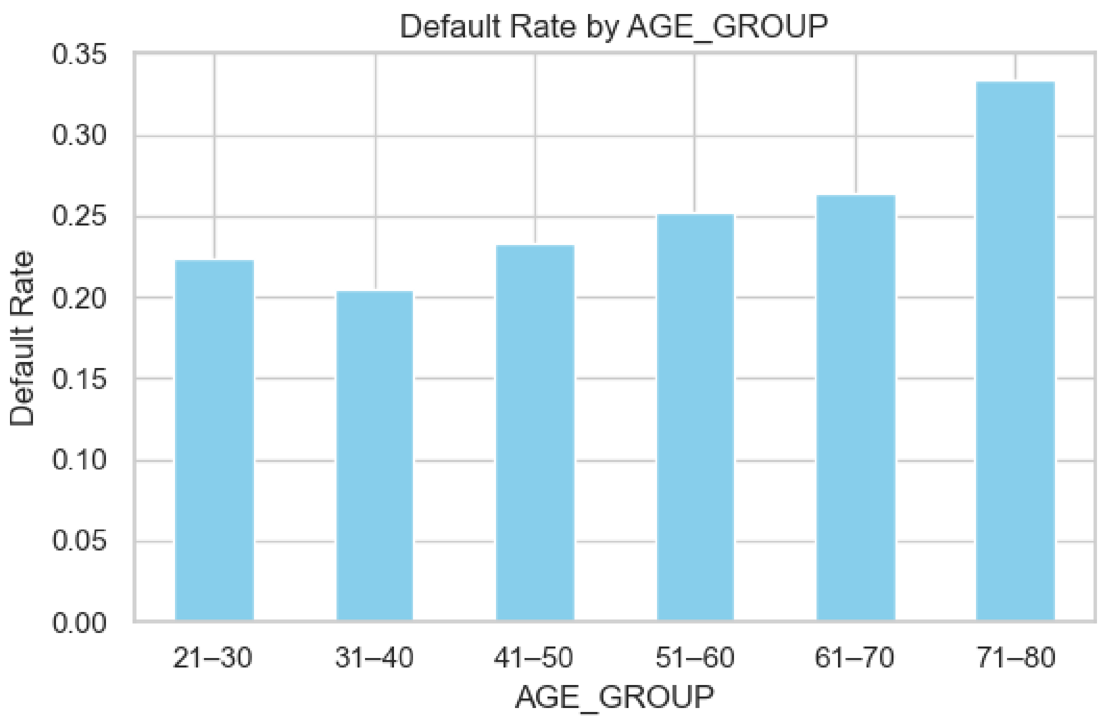


Default Rate by MARRIAGE
Default Rate
0.00
0.05
0.10
0.15
0.20
0.25
0
1
2
3
MARRIAGE



Default Rate by SEX
Default Rate
0.00
0.05
0.10
0.15
0.20
0.25
1
2
SEX

```
/var/folders/ck/sr6gtz6n0jx9dmp9nlplxl_w0000gn/T/ipykernel_65554/88419425.py:30:
FutureWarning: The default of observed=False is deprecated and will be changed
to True in a future version of pandas. Pass observed=False to retain current
behavior or observed=True to adopt the future default and silence this warning.
  df_bayes.groupby(col)['default'].mean().plot(kind='bar', color='skyblue')
```



```
Accuracy: 0.3778888888888889

Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.24      0.37      7040
           1       0.24      0.88      0.38      1960

    accuracy                           0.38      9000
   macro avg       0.56      0.56      0.38      9000
weighted avg       0.74      0.38      0.38      9000


Sample Predictions:
   Actual  Predicted Probability
0       0                 0.879282
1       0                 0.799860
```

```
2       0                0.852085
3       0                0.851283
4       1                0.872585
Accuracy is 0.377888 -- This means 38%  of the customers were correctly
classified - either as likely to default (1) or not (0).

The report breaks down precision, recall, and F1-score for each class

For Class 0 -- No Default
Precision = 0.88: 88% of those predicted as -- No Default were correct
Recall = 0.24: 24% of the actual -- no default customers correctly predicted.
F1 = 0.37 -- Weak ability to detect actual non-defaulters.

For Class 1 -- Default
Precision = 0.24: 24% of predicted defaulters were actually defaulters
Recall = 0.88: 88% of actual defaulters -- Postive case of how many prdicted to
be defaulted
F1 = 0.38: Weak ability to detect actual defaulter
 Tha model is too conservative - reluctant to label someone as a defaulter.
For credit risk, recall on Class 1 is critical - you want to catch as many
defaulters as possible!


 --- Sample Predictions ---
Actual: The true class -- 0 = no default, 1 = default
Predicted Probability: Model's confidence that the customer will default

Row 0: True label is 0 (no default), model predicts 87% chance of default -
correct and confident.
Row 4: True label is 1 (default), model predicts 87% -   somewhat confident,
borderline.

 --- Recommendations ---
Improve recall on defaulters: Try different models like (e.g., logistic
regression, random forest), oversampling (SMOTE), or cost-sensitive learning.
Threshold tuning: Adjust default classification threshold (not just 0.5) to
balance precision/recall.
```

[246]:
```python
df.rename(columns={df.columns[-1]: 'default_status'}, inplace=True)

# Define the columns
bill_columns = ['BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4',␣
 ↪'BILL_AMT5', 'BILL_AMT6']
pay_columns = ['PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5',␣
 ↪'PAY_AMT6']


# Method 1: Linear decay weights (most recent gets highest weight)
# Weights: [6, 5, 4, 3, 2, 1] for [AMT1, AMT2, AMT3, AMT4, AMT5, AMT6]
linear_weights = np.array([6, 5, 4, 3, 2, 1])
linear_weights = linear_weights / linear_weights.sum()  # Normalize to sum to 1

print("Linear weights:", linear_weights)

# Calculate weighted averages
df['WEIGHTED_BILL_AMT'] = np.average(df[bill_columns], weights=linear_weights,␣
 ↪axis=1)
df['WEIGHTED_PAY_AMT'] = np.average(df[pay_columns], weights=linear_weights,␣
 ↪axis=1)

def create_numeric_percentile_bins(df, column_name, num_bins=4):
    """
    Create percentile bins with ascending numeric codes (1, 2, 3, 4)
    """
    # Create percentile bins and assign numeric labels
    binned_column = pd.qcut(df[column_name], q=num_bins, labels=range(1,␣
 ↪num_bins + 1), duplicates='drop')

    # Get the actual bin edges for reference
    _, bin_edges = pd.qcut(df[column_name], q=num_bins, retbins=True,␣
 ↪duplicates='drop')

    return binned_column.astype(int), bin_edges

# Apply numeric percentile binning
variables_to_bin = ['AGE', 'LIMIT_BAL', 'WEIGHTED_BILL_AMT', 'WEIGHTED_PAY_AMT']

print("Creating numeric percentile-based bins (1=lowest quartile, 4=highest␣
 ↪quartile)...")
print("=" * 80)

for var in variables_to_bin:
    # Create numeric bins
    binned_col, edges = create_numeric_percentile_bins(df, var, num_bins=4)

    # Add the binned column to dataframe
    df[f'{var}_Q'] = binned_col

    # Print bin information
    print(f"\n{var}_Q:")
    print(f"  Overall range: {df[var].min():.2f} to {df[var].max():.2f}")
    print(f"  Quartile boundaries and coding:")

    for i in range(len(edges) - 1):
        quartile_num = i + 1
        start_val = edges[i]
        end_val = edges[i + 1]
        count = (df[f'{var}_Q'] == quartile_num).sum()
        percentage = count / len(df) * 100

        print(f"    {quartile_num}: {start_val:8.2f} to {end_val:8.2f} | {count:
↪,} obs ({percentage:.1f}%)")

    # Show the numeric distribution
    print(f"  Value counts: {dict(df[f'{var}_Q'].value_counts().sort_index())}")



df.head()
```

```
Linear weights: [0.28571429 0.23809524 0.19047619 0.14285714 0.0952381
0.04761905]
Creating numeric percentile-based bins (1=lowest quartile, 4=highest
quartile)…
================================================================================

AGE_Q:
  Overall range: 21.00 to 79.00
  Quartile boundaries and coding:
    1:    21.00 to    28.00 | 8,013 obs (26.7%)
    2:    28.00 to    34.00 | 7,683 obs (25.6%)
    3:    34.00 to    41.00 | 6,854 obs (22.8%)
    4:    41.00 to    79.00 | 7,450 obs (24.8%)
  Value counts: {1: np.int64(8013), 2: np.int64(7683), 3: np.int64(6854), 4:
np.int64(7450)}

LIMIT_BAL_Q:
  Overall range: 10000.00 to 1000000.00
  Quartile boundaries and coding:
    1: 10000.00 to 50000.00 | 7,676 obs (25.6%)
    2: 50000.00 to 140000.00 | 7,614 obs (25.4%)
    3: 140000.00 to 240000.00 | 7,643 obs (25.5%)
    4: 240000.00 to 1000000.00 | 7,067 obs (23.6%)
  Value counts: {1: np.int64(7676), 2: np.int64(7614), 3: np.int64(7643), 4:
np.int64(7067)}

WEIGHTED_BILL_AMT_Q:
  Overall range: -29464.95 to 873217.38
  Quartile boundaries and coding:
    1: -29464.95 to  4888.90 | 7,500 obs (25.0%)
    2:  4888.90 to 21980.29 | 7,500 obs (25.0%)
    3: 21980.29 to 60405.44 | 7,500 obs (25.0%)
    4: 60405.44 to 873217.38 | 7,500 obs (25.0%)
  Value counts: {1: np.int64(7500), 2: np.int64(7500), 3: np.int64(7500), 4:
np.int64(7500)}

WEIGHTED_PAY_AMT_Q:
  Overall range: 0.00 to 805849.48
  Quartile boundaries and coding:
    1:      0.00 to  1228.08 | 7,500 obs (25.0%)
    2:   1228.08 to  2488.14 | 7,500 obs (25.0%)
    3:   2488.14 to  5696.19 | 7,500 obs (25.0%)
    4:   5696.19 to 805849.48 | 7,500 obs (25.0%)
  Value counts: {1: np.int64(7500), 2: np.int64(7500), 3: np.int64(7500), 4:
np.int64(7500)}
```

[246]:

```
   ID  LIMIT_BAL  SEX  EDUCATION  MARRIAGE  AGE  PAY_0  PAY_2  PAY_3  PAY_4  \
0   1      20000    2          2         1   24      2      2     -1     -1
1   2     120000    2          2         2   26     -1      2      0      0
2   3      90000    2          2         2   34      0      0      0      0
3   4      50000    2          2         1   37      0      0      0      0
4   5      50000    1          2         1   57     -1      0     -1      0

   …  PAY_AMT4  PAY_AMT5  PAY_AMT6  default_status  WEIGHTED_BILL_AMT  \
0  …         0         0         0               1        1987.809524
1  …      1000         0      2000               1        2639.619048
2  …      1000      1000      5000               0       18487.761905
3  …      1100      1069      1000               0       42508.380952
4  …      9000       689       679               0       16363.571429

   WEIGHTED_PAY_AMT  AGE_Q  LIMIT_BAL_Q  WEIGHTED_BILL_AMT_Q  \
0        164.047619      1            1                    1
1        666.666667      1            2                    1
2       1457.523810      2            2                    2
3       1587.285714      3            1                    3
4      12593.428571      4            1                    2

   WEIGHTED_PAY_AMT_Q
0                   1
1                   1
2                   2
3                   2
4                   4

[5 rows x 31 columns]
```

[247]:

```python
# replace -1 with 0
df['PAY_0'] = df['PAY_0'].replace(-1, 0)

# separate between train and test
```

```
train_df = df.sample(frac=0.7, random_state=42)
test_df = df.drop(train_df.index)

train_df.shape
```

[247]:
```
(21000, 31)
```

[248]:
```
# train logistic regression model

import statsmodels.formula.api as smf
import statsmodels.api as sm



model = smf.glm('default_status ~ LIMIT_BAL_Q + SEX + EDUCATION + MARRIAGE +␣
 ↪AGE_Q + PAY_0 + WEIGHTED_BILL_AMT_Q + WEIGHTED_PAY_AMT_Q', data=train_df,␣
 ↪family=sm.families.Binomial())

results = model.fit()

results.summary()
```

[248]:

Dep. Variable:	default_status	**No. Observations:**	21000
Model:	GLM	**Df Residuals:**	20991
Model Family:	Binomial	**Df Model:**	8
Link Function:	Logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-9560.6
Date:	Sat, 21 Jun 2025	**Deviance:**	19121.
Time:	14:25:17	**Pearson chi2:**	2.57e+04
No. Iterations:	5	**Pseudo R-squ. (CS):**	0.1311
Covariance Type:	nonrobust		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
Intercept	-0.1974	0.134	-1.474	0.140	-0.460	0.065
LIMIT_BAL_Q	-0.1390	0.019	-7.296	0.000	-0.176	-0.102
SEX	-0.1118	0.037	-3.014	0.003	-0.185	-0.039
EDUCATION	-0.0605	0.025	-2.415	0.016	-0.110	-0.011
MARRIAGE	-0.1505	0.039	-3.890	0.000	-0.226	-0.075
AGE_Q	0.0377	0.018	2.105	0.035	0.003	0.073
PAY_0	0.8464	0.021	39.870	0.000	0.805	0.888
WEIGHTED_BILL_AMT_Q	-0.0003	0.021	-0.013	0.990	-0.041	0.040
WEIGHTED_PAY_AMT_Q	-0.2598	0.022	-11.572	0.000	-0.304	-0.216

[249]:
```
# analyze results

summary_df = pd.concat([results.params, results.pvalues], axis=1, keys=['coef',␣
 ↪'pvalue'])
```

```
# absolute value of the coefficients for sorting
summary_df = summary_df.assign(abs_coef=summary_df['coef'].abs())

# get labels of variables with p > 0.05
removed_labels = summary_df.index[summary_df['pvalue'] > 0.05].tolist()

# keep only variables with p <= 0.05
summary_df = summary_df[summary_df['pvalue'] <= 0.05]

# sort by effect size
summary_df = summary_df.sort_values(by='abs_coef', ascending=False)

# rounding
summary_df['pvalue'] = summary_df['pvalue'].map('{:.5f}'.format)

# print labels of variables with p > 0.05
print("p > 0.05: \n\n{}".format(removed_labels))

print("\n---------------------------------\n")

print("Sorted by effect size: \n{}".format(summary_df))
print("\n---------------------------------\n")

# sort by pvalue
summary_df = summary_df.sort_values(by='pvalue', ascending=True)

print("\n---------------------------------\n")

print("Sorted by p-value: \n{}".format(summary_df))
print("\n---------------------------------\n")
```

```
p > 0.05:

['Intercept', 'WEIGHTED_BILL_AMT_Q']

---------------------------------


Sorted by effect size:
                         coef   pvalue  abs_coef
PAY_0                0.846440  0.00000  0.846440
WEIGHTED_PAY_AMT_Q  -0.259782  0.00000  0.259782
MARRIAGE            -0.150458  0.00010  0.150458
LIMIT_BAL_Q         -0.138968  0.00000  0.138968
SEX                 -0.111832  0.00258  0.111832
EDUCATION           -0.060463  0.01573  0.060463
AGE_Q                0.037691  0.03530  0.037691
```

```
--------------------------------


--------------------------------

Sorted by p-value:
                        coef   pvalue  abs_coef
PAY_0               0.846440  0.00000  0.846440
WEIGHTED_PAY_AMT_Q -0.259782  0.00000  0.259782
LIMIT_BAL_Q        -0.138968  0.00000  0.138968
MARRIAGE           -0.150458  0.00010  0.150458
SEX                -0.111832  0.00258  0.111832
EDUCATION          -0.060463  0.01573  0.060463
AGE_Q               0.037691  0.03530  0.037691

--------------------------------
```

[250]:
```
odds_ratios = pd.Series(
    data=round(np.exp(summary_df['coef']), 2),
    index=summary_df.index,
    name='odds_ratio'
)

print(odds_ratios)
```

```
PAY_0                 2.33
WEIGHTED_PAY_AMT_Q    0.77
LIMIT_BAL_Q           0.87
MARRIAGE              0.86
SEX                   0.89
EDUCATION             0.94
AGE_Q                 1.04
Name: odds_ratio, dtype: float64
```

[251]:
```
# Make examples

class Person:

    def __init__(self, age, sex, education, marriage, limit_balance,␣
 ↪bill_amount, payment_amount, payment_history):
        self.age = age
        self.sex = sex
        self.education = education
        self.marriage = marriage
        self.limit_balance = limit_balance
        self.bill_amount = bill_amount
        self.payment_amount = payment_amount
        self.payment_history = payment_history

    def calculate_probability(self):
        intercept = results.params['Intercept']
        age_coef = results.params['AGE_Q']
        sex_coef = results.params['SEX']
        education_coef = results.params['EDUCATION']
        marriage_coef = results.params['MARRIAGE']
        limit_balance_coef = results.params['LIMIT_BAL_Q']
        bill_amount_coef = results.params['WEIGHTED_BILL_AMT_Q']
        payment_amount_coef = results.params['WEIGHTED_PAY_AMT_Q']
        payment_history_coef = results.params['PAY_0']

        probability = 1 / (1 + np.exp(-(intercept + age_coef * self.age +␣
 ↪sex_coef * self.sex + education_coef * self.education + marriage_coef * self.
 ↪marriage + limit_balance_coef * self.limit_balance + bill_amount_coef * self.
 ↪bill_amount + payment_amount_coef * self.payment_amount +␣
 ↪payment_history_coef * self.payment_history)))

        return probability



jake = Person(age=1, sex=1, education=0, marriage=0, limit_balance=1,␣
 ↪bill_amount=2, payment_amount=0, payment_history=0)
print("jake:", round(jake.calculate_probability(), 4))

john = Person(age=1, sex=1, education=4, marriage=3, limit_balance=1,␣
 ↪bill_amount=4, payment_amount=0, payment_history=8)
print("john:", round(john.calculate_probability(), 4))

penelope = Person(age=4, sex=2, education=1, marriage=1, limit_balance=4,␣
 ↪bill_amount=1, payment_amount=3, payment_history=0)
print("penelope:", round(penelope.calculate_probability(), 4))

ricardo = Person(age=1, sex=1, education=1, marriage=0, limit_balance=4,␣
 ↪bill_amount=4, payment_amount=1, payment_history=6)
print("ricardo:", round(ricardo.calculate_probability(), 4))

stella = Person(age=2, sex=2, education=3, marriage=2, limit_balance=1,␣
 ↪bill_amount=1, payment_amount=1, payment_history=0)
print("stella:", round(stella.calculate_probability(), 4))
```

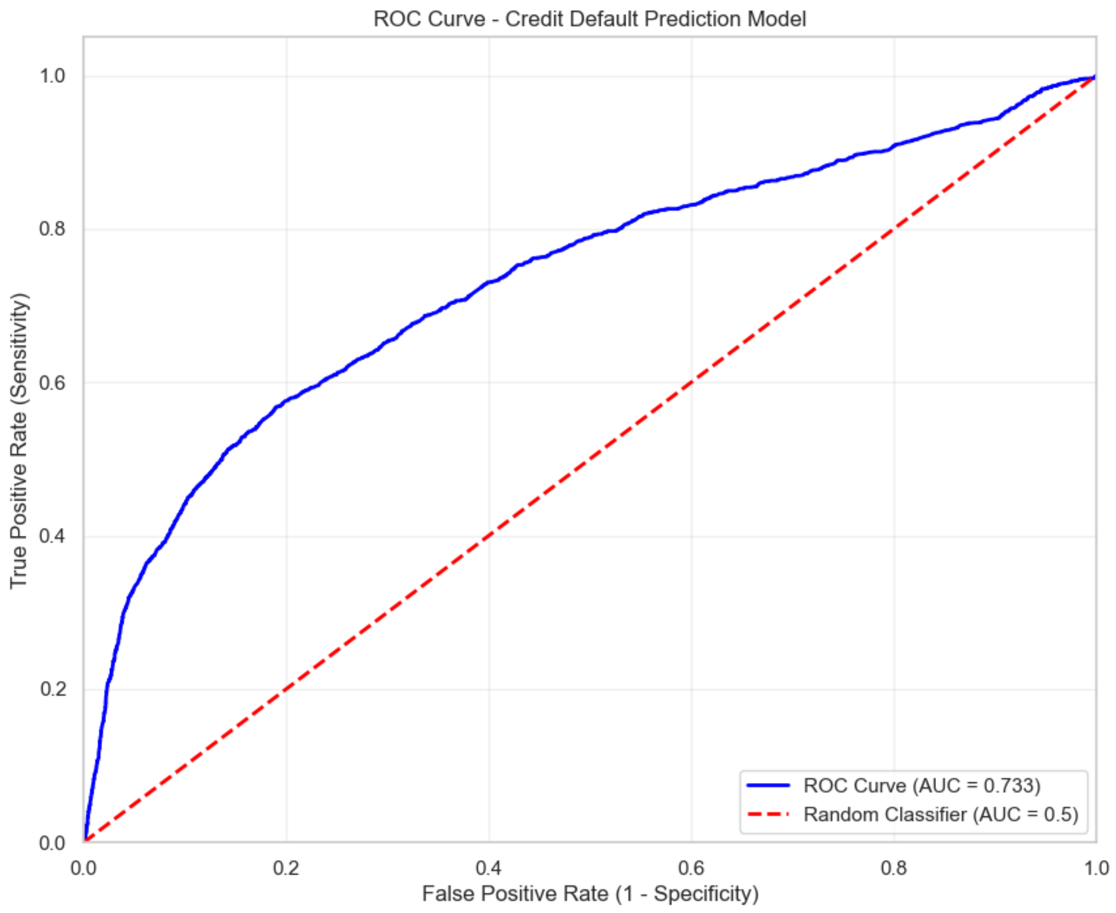
```
jake: 0.3987
john: 0.9966
penelope: 0.1398
```

```
ricardo: 0.9807
stella: 0.2267
```

[252]:
```
# calculate metrics

from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score, confusion_matrix, roc_auc_score, roc_curve
import matplotlib.pyplot as plt
import seaborn as sns


# Generate predictions on test set
# Get predicted probabilities
test_probabilities = results.predict(test_df)

# Convert probabilities to binary predictions using 0.5 threshold
test_predictions = (test_probabilities > 0.5).astype(int)

# Get actual values
test_actual = test_df['default_status'].values

print(f"Test set size: {len(test_df)}")
print(f"Number of actual defaults in test set: {sum(test_actual)}")
print(f"Number of predicted defaults: {sum(test_predictions)}")


# Calculate confusion matrix
cm = confusion_matrix(test_actual, test_predictions)
print("Confusion Matrix:")
print(cm)

# Extract components
tn, fp, fn, tp = cm.ravel()
print(f"\nBreakdown:")
print(f"True Negatives (TN): {tn}")
print(f"False Positives (FP): {fp}")
print(f"False Negatives (FN): {fn}")
print(f"True Positives (TP): {tp}")

# Calculate all performance metrics
accuracy = accuracy_score(test_actual, test_predictions)
precision = precision_score(test_actual, test_predictions)
sensitivity_recall = recall_score(test_actual, test_predictions)  # Same as␣
 ↪sensitivity
f1 = f1_score(test_actual, test_predictions)

# Calculate specificity manually (no direct sklearn function)
```

```python
specificity = tn / (tn + fp)

print("=== MODEL PERFORMANCE METRICS ===")
print(f"Accuracy: {accuracy:.4f} ({accuracy*100:.2f}%)")
print(f"Precision: {precision:.4f} ({precision*100:.2f}%)")
print(f"Sensitivity (Recall): {sensitivity_recall:.4f} ({sensitivity_recall*100:
↪.2f}%)")
print(f"Specificity: {specificity:.4f} ({specificity*100:.2f}%)")
print(f"F1-Score: {f1:.4f}")

print("\n=== METRIC INTERPRETATIONS ===")
print(f"• Accuracy: {accuracy*100:.1f}% of all predictions were correct")
print(f"• Precision: {precision*100:.1f}% of predicted defaults were actually␣
↪defaults")
print(f"• Sensitivity: {sensitivity_recall*100:.1f}% of actual defaults were␣
↪correctly identified")
print(f"• Specificity: {specificity*100:.1f}% of actual non-defaults were␣
↪correctly identified")
print(f"• F1-Score: Harmonic mean of precision and recall = {f1:.3f}")

# Calculate AUC
auc = roc_auc_score(test_actual, test_probabilities)
print(f"AUC-ROC Score: {auc:.4f}")

# Generate ROC curve data
fpr, tpr, thresholds = roc_curve(test_actual, test_probabilities)

# Plot ROC curve
plt.figure(figsize=(10, 8))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC Curve (AUC = {auc:.3f})')
plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--', label='Random␣
↪Classifier (AUC = 0.5)')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.title('ROC Curve - Credit Default Prediction Model')
plt.legend(loc="lower right")
plt.grid(True, alpha=0.3)
plt.show()

print(f"\n=== AUC INTERPRETATION ===")
if auc >= 0.9:
    interpretation = "Excellent"
elif auc >= 0.8:
    interpretation = "Good"
elif auc >= 0.7:
    interpretation = "Fair"
elif auc >= 0.6:
    interpretation = "Poor"
else:
    interpretation = "Very Poor"

print(f"AUC = {auc:.3f} indicates {interpretation} discriminatory ability")
```

```
Test set size: 9000
Number of actual defaults in test set: 2039
Number of predicted defaults: 738
Confusion Matrix:
[[6737  224]
 [1525  514]]

Breakdown:
True Negatives (TN): 6737
False Positives (FP): 224
False Negatives (FN): 1525
True Positives (TP): 514
=== MODEL PERFORMANCE METRICS ===
Accuracy: 0.8057 (80.57%)
Precision: 0.6965 (69.65%)
Sensitivity (Recall): 0.2521 (25.21%)
Specificity: 0.9678 (96.78%)
F1-Score: 0.3702

=== METRIC INTERPRETATIONS ===
• Accuracy: 80.6% of all predictions were correct
• Precision: 69.6% of predicted defaults were actually defaults
• Sensitivity: 25.2% of actual defaults were correctly identified
• Specificity: 96.8% of actual non-defaults were correctly identified
• F1-Score: Harmonic mean of precision and recall = 0.370
AUC-ROC Score: 0.7326
```



```
=== AUC INTERPRETATION ===
AUC = 0.733 indicates Fair discriminatory ability
```