Analyzing Customer Behavior for Loan Prediction: a Comprehensive Study

Group 6

October 2023

Index

List of Tables

List of Figures

List of Codes

Disclaimer

Given the constraints imposed and the context of our studies, we used Chat GPT to untangle our code and save time on certain methods. However, it was only a tool to help us, and the thinking behind the code, as well as the majority of the code, was not generated by Chat GPT.

Introduction

Our research delves into analyzing the `Loan Prediction Based on Customer Behavior.csv` dataset. Our objective, in essence, is for an organization to refine its consumer loan offerings by utilizing past customer behavior records to proactively pinpoint likely defaulters among prospective clients, consequently segregating those with increased risk. We intend to maximize the potential of this data by employing machine learning models to forecast the outcome.

1 Setup and Data Cleaning

First of all, we imported all the necessary packages and modules for our analysis. We then proceeded to import the dataset and to do a preliminary analysis of its characteristics. In particular, the dataset has 252,000 rows, where each row represents one customer's basic information recorded by the company. The dataset has 12 variables (table 1), all of which were available during the loan application phase. We employed `df.info()`, `df.nunique()`, and `df.isna().sum()` for a comprehensive examination of our variables, revealing 6 integers/categorical and 6 objects/quantitative features, without any missing values. Additionally, 3 of the categorical variables are binary (`Married/Single`, `Car_Ownership`, `Risk_Flag`). Notably, the `risk_flag` indicates whether there has been a default in the past or not.

#	Non-Null Count	Dtype
Income	252000	int64
Age	252000	int64
Experience	252000	int64
Married/Single	252000	object
House_Ownership	252000	object
Car_Ownership	252000	object
Profession	252000	object
CITY	252000	object
STATE	252000	object
CURRENT_JOB_YRS	252000	int64
CURRENT_HOUSE_YRS	252000	int64
Risk_Flag	252000	int64

Table 1: `df.info()` output [4]

However, upon executing `df['STATE'].unique()` and `df['Profession'].unique()` to inspect the distinct state and city names, we identified some anomalous entries like `Uttar_Pradesh[5]` and `Hajipur[31]` that required further cleansing. To streamline our process, we first converted all column names to lowercase for more straightforward manipulation. For tidying up the state and city values, we implemented the following code:

```
df['city'] = df['city'].str.extract("([A-Za-z]+)")                [12]
df['state'] = df['state'].str.extract("([A-Za-z]+)")
```

Post-cleansing, the values were refined to `Uttar` and `Hajipur`. Furthermore, we updated the column name `married/single` to `relationship_status`.

Regarding the rest of the dataset, it appears to be very clean without any outliers, allowing us to proceed with our data analysis confidently.

2 Data Analysis

In this second phase, we started to analyze in depth our data to find interesting insights or any potential problem.

2.1 Univariate Data Analysis

First, we selected only the categorical variables and created a summary table for each column. We also distinguished between binary categories, for which we generated pie charts (fig 1), and non-binary ones, for which we created horizontal bar charts.

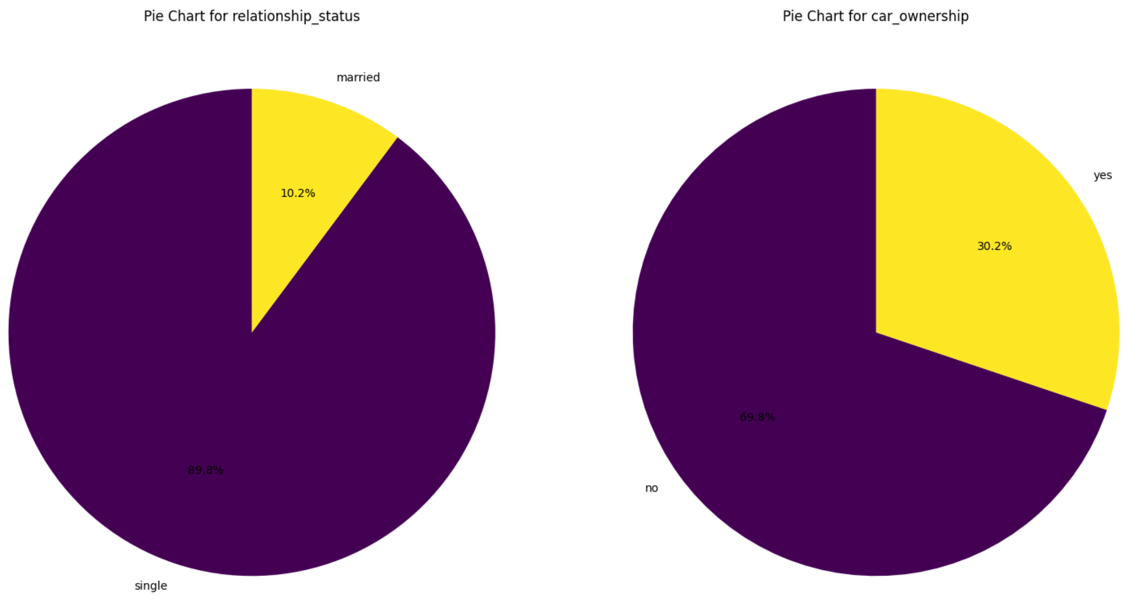

Figure 1: Pie chart for `relationship_status` and `car_ownership` [15]

The pie chart for relationship status indicates that a significant majority of the customers in the dataset are single (89.8%). The bar chart for house ownership reveals that majority of the customers rent their homes. However, the distribution is probably unbalanced. This data imbalance might affect the training and prediction of machine learning models. Unbalanced data can lead the model to over-optimize for the dominant category and neglect the minority class. The result might be that the model predicts very well for the dominant category but performs poorly for the minority category. Similarly, the data on vehicle ownership also shows an imbalance, with 69.8% of people not owning a car and 30.2% owning one.

Then, we looked at the quantitative variables and generated histograms accordingly to visualizes the distribution of data over a continuous interval (fig 2).

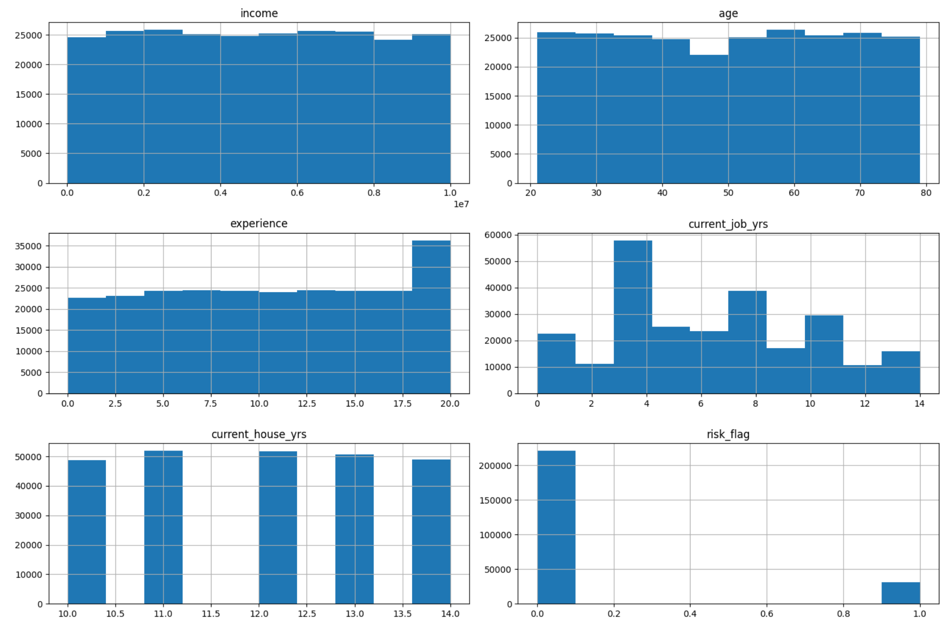

Figure 2: Quantitative variables histograms [16]

Most of the variables seem to be well-balanced, except for `risk_flag` (fig 3), so we decided to look into it.

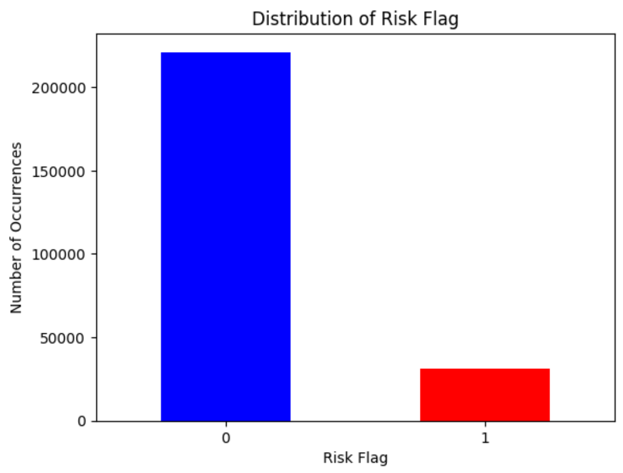

Figure 3: `risk_flag` histogram [17]

We can clearly see a very important class imbalance between the two classes, suggesting that we might need to resampling techniques (e.g., SMOTE) to ensure a balanced dataset, especially when building predictive models. This is because models trained on imbalanced datasets might have a bias towards the majority class.

In the meantime, we also created box pots for each variable to visual the spread of the data, showcasing the median, quartiles, and potential outliers (fig 4).

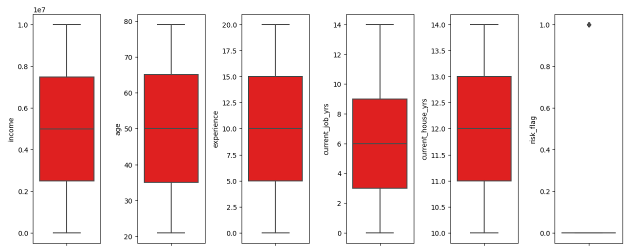

Figure 4: Quantitative variables boxplots [18]

As we can see, the spread of others is relatively even, without significant outliers. As for `risk_flag`, most of the data is centered around 0, with very few instances of the risk flag being 1. This box plot further emphasizes the imbalance in the data regarding the risk flag.

2.2 Bivariate Data Analysis

In order to analyze our variables in pairs, we applied different solutions.

2.2.1 Contingency Table, Chi-square and Cramer's V

We first imported the `itertools.combinations` library to generate combinations of columns and the `chi2_contingency` one to perform the chi-square test of independence. Subsequently, we used a loop iterating through all possible pairs of categorical columns using combinations to get the contingency table, chi-square, p-value and Cramer's V.

	house_ownership		
relationship_status	norent_noown	owned	rented
married	595	923	24,210
single	6,589	11,995	207,688
Chi-square Statistic	174.869089039534		
p-value	1.0657609105047439e-38		
Cramer's V	0.026342455400096765		

Table 2: Contingency table and statistics for `relationship_status` and `house_ownership` [20]

For example, looking at the contingency table for `relationship_status` and `house_ownership` (fig 2), we can see that married people tend to have more owned houses compared to single people in proportion. There are significantly fewer married individuals who don't own or rent compared to single individuals. The obtained p-value, significantly lower than the threshold of 0.05, suggests a statistically significant association between relationship_status" and `house_ownership`. However, it's essential to note that the effect size, as quantified by Cramer's V with a value of 0.0263, indicates a relatively small strength of association between these two variables.

Meanwhile, looking at contingency table for `relationship_status` and `car_ownership` (table 3), we observed that the distribution seems almost proportional between married and single individuals regarding car ownership.

	car_ownership	
relationship_status	no	yes
married	18,011	7,717
single	157,989	68,283
Chi-square Statistic	0.3580171005169928	
p-value	0.5496095547211083	
Cramer's V	0.0011919323721710626	

Table 3: Contingency table for `relationship_status` and `car_ownership` [20]

The calculated p-value, which exceeds the 0.05 threshold, combined with Cramer's V of 0.0012 indicating a very small effect size, leads to the conclusion that there is no statistically significant association between `relationship_status` and `car_ownership`. In other words, these variables are independent of each other.

2.2.2 Correlation Table

Subsequently, we generated a correlation table (table 4) for the quantitative columns and conducted a correlation test to calculate both the correlation coefficient (r) and associated p-values (table 5), allowing us to assess the significance of the observed correlations.

	income	age	experience	current_job_yrs	current_house_yrs	risk_flag
income	1.000000	-0.000652	0.006422	0.007045	-0.002397	-0.003091
age	-0.000652	1.000000	-0.001118	0.002154	-0.020134	-0.021809
experience	0.006422	-0.001118	1.000000	0.646098	0.019309	-0.034523
current_job_yrs	0.007045	0.002154	0.646098	1.000000	0.005372	-0.016942
current_house_yrs	-0.002397	-0.020134	0.019309	0.005372	1.000000	-0.004375
risk_flag	-0.003091	-0.021809	-0.034523	-0.016942	-0.004375	1.000000

Table 4: Correlation matrix for quantitative variables [21]

The correlation table provides insight into the linear relationship between multiple variables. Notably, `experience` and `current_job_yrs` exhibit a strong positive correlation, indicated by a coefficient of approximately 0.6461. This suggests that as experience increases, the years in the current job tend to increase as well, which is an expected trend in many professional settings. Most other variables demonstrate negligible correlations, with values close to zero, suggesting little to no linear relationship. For instance, `income` and `age` have a coefficient of -0.0007, indicating an almost non-existent correlation.

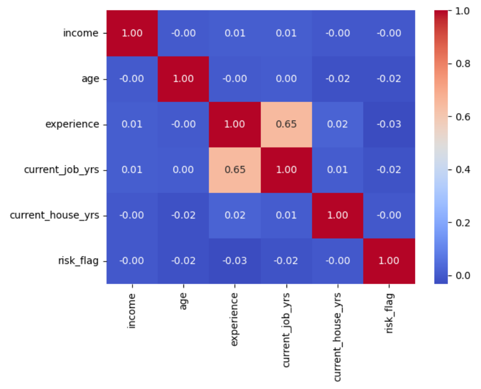

Figure 5: Heatmap of the correlation matrix [22]

The Pearson correlation test results provide a more detailed view, especially concerning the significance of these relationships (table 5). The correlation between `income` and `experience` is 0.0064, with a p-value of 0.0013, signifying that the correlation is statistically significant at conventional levels (typically $p < 0.05$). Similarly, `income` and `current_job_yrs` also have a significant correlation with a p-value of 0.0004. However, correlations like that between `income` and `age` are not statistically significant, given the high p-value of 0.7435.

Variable Pair	Pearson Correlation	p-Value
`income` and `age`	-0.0007	0.7435
`income` and `experience`	0.0064	0.0013
`income` and `current_job_yrs`	0.0070	0.0004
`income` and `current_house_yrs`	-0.0024	0.2288
`income` and `risk_flag`	-0.0031	0.1207
`age` and `experience`	-0.0011	0.5747
`age` and `current_job_yrs`	0.0022	0.2795
`age` and `current_house_yrs`	-0.0201	0.0000
`age` and `risk_flag`	-0.0218	0.0000
`experience` and `current_job_yrs`	0.6461	0.0000
`experience` and `current_house_yrs`	0.0193	0.0000
`experience` and `risk_flag`	-0.0345	0.0000
`current_job_yrs` and `current_house_yrs`	0.0054	0.0070
`current_job_yrs` and `risk_flag`	-0.0169	0.0000
`current_house_yrs` and `risk_flag`	-0.0044	0.0281

Table 5: Pearson correlation coefficients and p-values [23]

2.2.3 Scatterplot Matrix

The scatterplot matrix (fig 6) offers a visually insightful representation of the relationships between various quantitative variables within our dataset. One notable observation is the conspicuous linear relationship observed between the variables `experience` and `current_job_yrs`. This linear pattern reasonably suggests that, as one's years of experience increase, so does their tenure in their current job. However, for most other combinations of variables, discernible linear patterns are notably absent. Instead, we encounter a multitude of scattered data points that do not conform to linear relationships. Furthermore, it is important to highlight the unique behavior of the variable `risk_flag`. Unlike the continuous variables, `risk_flag` exhibits discrete data points, denoting its binary nature. In summary, the scatterplot matrix serves as a valuable tool for visually assessing the associations between quantitative variables, revealing both linear trends and the discrete nature of certain variables, such as `risk_flag`.

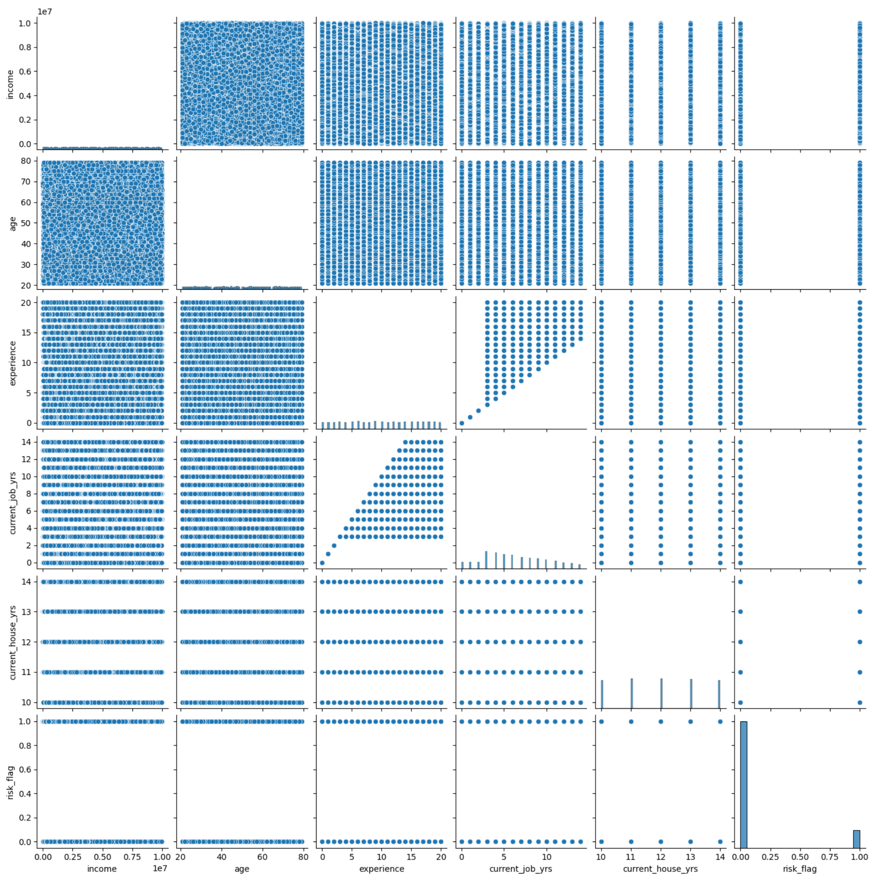

Figure 6: Scatterplot matrix [24]

2.2.4 Anova Table

Lastly, we used OLS (Ordinary Least Squares) method for statistical modeling and created the Anova Table for both quantitative and qualitative columns (table 6). Using a nested loop, the code pairs each quantitative column with each qualitative column. For each pair, an OLS model is constructed using a formula that considers the quantitative column as the dependent variable and the qualitative column as the independent variable. Once the model is fit, an Anova (i.e., analysis of variance) table is produced. This table helps determine if the means of different groups within the qualitative variable are statistically different regarding the quantitative variable.

```
for q_col, qual_col in product(quantitative_columns, qualitative_columns):    [20]
    formula = f"{q_col} ~ C({qual_col})"
    model = ols(formula, data=df).fit()
    anova_table = sm.stats.anova_lm(model, typ=2)

    print(f"ANOVA table for {q_col} by {qual_col}:\n")
    print(anova_table)
    print("\n" + "-"*65 + "\n")
```

Notably, most of the factors show a statistically significant association with the response variables, as indicated by very small p-values (i.e., `PR(>F)`). For instance, income by profession, city, and state have p-values nearly zero, suggesting these factors greatly influence income distribution. Similarly, age's relation with profession, city, and state also indicates strong associations. However, there are a few non-significant relationships as well, like the relationship status's influence on experience and current house years by car ownership, where the p-values are notably higher.

(a) Anova table for `income` by `relationship_status`

Source	sum_sq	df	F	PR(>F)
C(relationship_status)	1.501984×10^{13}	1.0	1.812973	0.178153
Residual	2.087715×10^{18}	251998.0	NaN	NaN

(b) Anova table for `income` by `house_ownership`

Source	sum_sq	df	F	PR(>F)
C(house_ownership)	8.204258×10^{14}	2.0	49.533739	3.104545×10^{-22}
Residual	2.086909×10^{18}	251997.0	NaN	NaN

(c) Anova table for `income` by `car_ownership`

Source	sum_sq	df	F	PR(>F)
C(car_ownership)	3.454915×10^{13}	1.0	4.170301	0.04114
Residual	2.087695×10^{18}	251998.0	NaN	NaN

(d) Anova table for `income` by `profession`

Source	sum_sq	df	F	PR(>F)
C(profession)	8.803626×10^{15}	50.0	21.338565	$2.775902 \times 10^{-190}$
Residual	2.078926×10^{18}	251949.0	NaN	NaN

Table 6: Anova tables [25]

The ANOVA tables test the impact of different qualitative variables (like relationship status, house ownership, car ownership, and profession) on a quantitative variable, in this case, `income`. The F-statistic and the associated p-value are key metrics here. A low p-value (< 0.05 as a common threshold) suggests that there are significant differences in the means of the groups. For instance, the `house_ownership` has a very low p-value, indicating that there's a statistically significant difference in income levels across different house ownership statuses. On the other hand, variables like `relationship_status` and `car_ownership` have higher p-values or missing F-statistics, indicating that the data does not provide strong evidence of significant differences in income based on those categorical variables.

3 Data Preprocessing

In this section, our primary focus was on preparing the data for machine learning. We specifically addressed two critical aspects: feature scaling and handling class imbalance.

3.1 Column Transformer

Initially, we selected numeric features by excluding those of `object` and `category` types, and set up a preprocessing step to scale these numeric features using the `StandardScaler()`. This step standardized the numeric features to have a mean of 0 and a standard deviation of 1. Furthermore, we encoded categorical variables using label encoding, converting each category into a unique numerical identifier.

```
if 'risk_flag' in numeric_features:                                    [26-30]
    numeric_features.remove('risk_flag')

preprocessor = ColumnTransformer([
    ('standard-scaling',StandardScaler(),numeric_features)
])

lb = LabelEncoder()

for col in categorical_features_lb:
    df[col]= lb.fit_transform(df[col])

X = df.drop('risk_flag', axis=1)
y = df['risk_flag']
X = preprocessor.fit_transform(X)
```

3.2 SMOTE

Then, we addressed the potential issue of class imbalance using the Synthetic Minority Over-sampling Technique (SMOTE) as mentioned above. After applying SMOTE, we visualized the distribution of the risk flag classes again using a bar chart, which illustrates that the classes have been made equally distributed (fig 7).

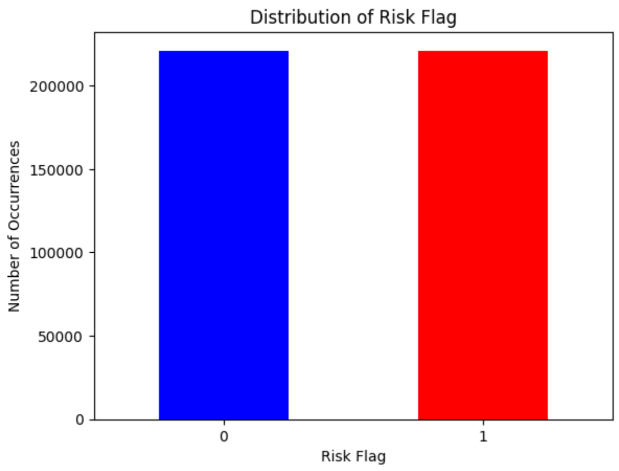

Figure 7: `risk_flag` distribution after applying SMOTE [32]

When dealing with data imbalance, using geometric mean (G-Mean) or Youden's J statistic can be preferable over synthetic oversampling techniques like SMOTE. We decided to use SMOTE for logistical reasons, however we wanted to highlight the differences between these methods. G-Mean and Youden's J statistic do not generate synthetic samples like SMOTE. Instead, they work with the original data points. In some cases, synthetic samples created by SMOTE might be hard to interpret since they are not actual data points. G-Mean and Youden's J statistic provide values that are easier to interpret and relate to the original data. Moreover, SMOTE assumes that the data follows a particular distribution when generating synthetic samples, while G-Mean and Youden's J statistic do not make such assumptions, making them more robust when dealing with data that may not conform to the expected distribution. So, G-Mean and Youden's J statistic are sensitive to class imbalance; they can help balance the trade-off between precision and recall without introducing synthetic data points. In contrast, SMOTE might not always achieve the desired balance.

4 Modeling

In this section, we delved into the heart of our machine learning process, where we explored and evaluated various algorithms for the task at hand. We began by introducing three fundamental algorithms that played a pivotal role in our predictive modeling journey: the K-Nearest Neighbors Classifier, Logistic Regression, and the Random Forest.

We initiated by partitioning the dataset into training and testing subsets, utilizing the `train_test_split` method with a test size of 30%. Stratification is used to ensure that the distribution of the target variable remains consistent in both the training and testing sets. After splitting, the training data was once again resampled using the SMOTE technique to balance the class distribution.

4.1 K-Nearest Neighbors Classifier

For the k-nearest neighbors (KNN) algorithm, we used `GridSearchCV` to find the optimal parameters. The best parameters are found to be `29` neighbors, `distance` weight, and the `euclidean` metric with a cross-validation score of `0.90`.

[35]

```
param_grid = {
    'n_neighbors': list(range(1, 31)),
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan', 'minkowski']
    }

knn = KNeighborsClassifier()
grid_search = GridSearchCV(knn, param_grid, cv=5, n_jobs=-1)
grid_search.fit(X_train_resampled, y_train_resampled)
```

Subsequently, the KNN model is trained with these parameters and tested. The classification report (table 7) shows decent precision and recall scores, especially for class 0.

[36,37]

```
knn_best_param= KNeighborsClassifier(**grid_search.best_params_)
knn_best_param.fit(X_train_resampled,y_train_resampled)
y_pred = knn_best_param.predict(X_test)
```

	Precision	Recall	F1-Score	Support
0	0.96	0.89	0.93	66,301
1	0.50	0.74	0.59	9,299
Accuracy			0.88	75,600
Macro Avg	0.73	0.82	0.76	75,600
Weighted Avg	0.90	0.88	0.89	75,600

Table 7: KNN classification report [38]

4.2 Logistic Regression

For logistic regression, again, a grid search is executed to pinpoint the best hyper-parameters. The parameters in the grid included the inverse of regularization strength, penalty type (`L1` or `L2`), and solver (`liblinear` or `saga`). The best parameters obtained are `C=1000`, `L2` penalty, and the `saga` solver. However, the cross-validation score for this set of parameters is notably lower at 0.53.

```
param_grid = {                                                    [39]
    'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear', 'saga']}

logreg = LogisticRegression(max_iter=10000)
grid_search = GridSearchCV(logreg, param_grid, cv=5, n_jobs=-1)
grid_search.fit(X_train_resampled, y_train_resampled)
```

Upon testing the model (table 8), we noted that while the precision for class 0 is high, the recall for class 1 is quite low, indicating potential issues with false negatives.

```
logreg_best_param = LogisticRegression(max_iter=10000,          [40,41]
                        C=0.01,penalty='l1',solver='saga')
logreg_best_param.fit(X_train_resampled,y_train_resampled)
y_pred = logreg_best_param.predict(X_test)
```

	Precision	Recall	F1-Score	Support
0	0.89	0.50	0.64	66,301
1	0.13	0.54	0.21	9,299
Accuracy			0.50	75,600
Macro Avg	0.51	0.52	0.42	75,600
Weighted Avg	0.79	0.50	0.58	75,600

Table 8: Logistic regression classification report [42]

4.3 Random Forest

For the Random Forest classifier, we employed a randomized search and the parameters being tuned include the number of trees (estimators), maximum depth of trees, minimum samples required at a leaf and to split an internal node, whether to use bootstrap samples, and the function to measure the quality of a split. The optimal parameters derived are `50` estimators, `2` minimum samples to split, `1` minimum sample at each leaf, and using the `entropy` criterion.

```
param_grid = {                                                    [43]
    'n_estimators': [50, 100],
    'max_depth': [None, 20],
    'min_samples_split': [2, 10],
    'min_samples_leaf': [1, 4],
    'bootstrap': [True, False],
    'criterion': ['gini', 'entropy']}

rf = RandomForestClassifier(random_state=42)
random_search = RandomizedSearchCV(rf, param_grid, n_iter=10, cv=3,
                    n_jobs=-1, random_state=42)
random_search.fit(X_train_resampled, y_train_resampled)
```

With these parameters, the Random Forest model's result actually shows a high precision, recall, and f1-score for both classes (table 9).

```
rf_best_param = RandomForestClassifier(**random_search.best_params_)      [44,45]
rf_best_param.fit(X_train_resampled,y_train_resampled)
y_pred = rf_best_param.predict(X_test)
```

	Precision	Recall	F1-Score	Support
0	0.97	0.90	0.93	66,301
1	0.53	0.79	0.63	9,299
Accuracy			0.89	75,600
Macro Avg	0.75	0.84	0.78	75,600
Weighted Avg	0.91	0.89	0.90	75,600

Table 9: Random forest classification report [46]

4.4 Elastic Net Logistic Regression

For Elastic Net Logistic Regression, we mixed the ratio between Lasso (`l1`) and Ridge (`l2`), with hyperparameters defined in the `param_grid`. Using `RandomizedSearchCV` with 3-fold cross-validation, we identified the optimal parameters: a `saga` solver, maximum iteration of 1000, `L1_ratio` of 0.5, and `C` value of 100, resulting in a cross-validation score of 0.52.

```
param_grid = {                                                             [47]
    'C': [0.001, 0.01, 0.1, 1, 10, 100],
    'l1_ratio': [0, 0.5, 1],
    'solver': ['saga']}

log_reg_elastic = LogisticRegression(max_iter = 10000, penalty='elasticnet',
                                     random_state=42)
random_search = RandomizedSearchCV(log_reg_elastic, param_grid, n_iter=10,
                                   cv=3, n_jobs=-1, random_state=42)
random_search.fit(X_train_resampled, y_train_resampled)
```

When trained on the resampled data with these parameters and tested, the model exhibits an overall accuracy of 50% and weighted average precision is 79%, with class 0 having a higher precision (89%) but only half of the actual class 0 instances being correctly predicted (50% recall). In contrast, class 1 has a lower precision (13%) but a comparable recall (55%). The F1-scores, which balance precision and recall, are 0.64 for class 0 and 0.21 for class 1 (table 10).

```
logreg_elastic_best_param = LogisticRegression(penalty='elasticnet',      [48,49]
                                  **random_search.best_params_)
logreg_elastic_best_param.fit(X_train_resampled,y_train_resampled)
y_pred = logreg_elastic_best_param.predict(X_test)
```

	Precision	Recall	F1-Score	Support
0	0.89	0.50	0.64	66,301
1	0.13	0.55	0.21	9,299
Accuracy			0.50	75,600
Macro Avg	0.51	0.52	0.42	75,600
Weighted Avg	0.79	0.50	0.58	75,600

Table 10: Elastic net logistic regression report [50]

4.5 XGBoost

For XGBoost, a gradient boosting framework, we specified hyperparameters within the `param_grid`, encompassing factors such as learning rate, number of estimators, maximum depth, and regularization terms among others. Utilizing `RandomizedSearchCV` with a 3-fold cross-validation, we found the optimal parameters: a subsample rate of 0.5, `reg_lambda` of 3, `reg_alpha` of 0, `n_estimators` of 100, `max_depth` of 10, `learning_rate` of 0.1, `gamma` of 0.5, and `colsample_bytree` of 1.0. This configuration achieved an impressive cross-validation score of 0.91.

[51]

```
param_grid = {
    'learning_rate': [0.001, 0.01, 0.1, 0.3],
    'max_depth': [3, 4, 5, 6, 7, 8, 10],
    'n_estimators': [50, 100, 150, 200],
    'subsample': [0.5, 0.7, 0.9, 1.0],
    'colsample_bytree': [0.5, 0.7, 0.9, 1.0],
    'gamma': [0, 0.25, 0.5, 1.0],
    'reg_alpha': [0, 0.1, 0.5, 1],
    'reg_lambda': [1, 1.5, 2, 3, 4.5]
}

xgb_model = xgb.XGBClassifier(objective='binary:logistic', random_state=42,
                              use_label_encoder=False, eval_metric='logloss')
random_search = RandomizedSearchCV(xgb_model, param_grid, n_iter=10, cv=3,
                                   n_jobs=-1, random_state=42)
random_search.fit(X_train_resampled, y_train_resampled)
```

Upon training with the resampled data, we found an exciting overall accuracy of 88% and a weighted average precision of 91%. Class 0 displayed a precision of 97% and a recall of 90%, leading to an F1-score of 0.93. On the other hand, class 1 presented a precision of 51% and a recall of 78%, resulting in an F1-score of 0.62 (table 11).

[52,53]

```
xgb_best_param = xgb.XGBClassifier(objective='binary:logistic',
                                   random_state=42,use_label_encoder=False,
                                   eval_metric='logloss',
                                   **random_search.best_params_)
xgb_best_param.fit(X_train_resampled,y_train_resampled)
y_pred = xgb_best_param.predict(X_test)
```

	Precision	Recall	F1-Score	Support
0	0.97	0.90	0.93	66,301
1	0.51	0.78	0.62	9,299
Accuracy			0.88	75,600
Macro Avg	0.74	0.84	0.77	75,600
Weighted Avg	0.91	0.88	0.89	75,600

Table 11: XGBoost classification report [54]

4.6 Overall Evaluation

In summary, among the five models, the Random Forest and XGBoost classifiers demonstrate superior performance, with both precision and recall values being substantially high for the two classes. The K-nearest Neighbors classifier also performs well, especially for class 0. In contrast, Logistic Regression and Elastic Net Logistic Regression seems to lag, particularly in terms of recall for class 1, indicating a potential issue in identifying true positives for that class. Given the results, the Random Forest model or XGBoost appears to be the best choice for this dataset and problem.

4.7 Note on Unsupervised Learning

Despite the extensive modeling approaches, we did not employ any unsupervised learning techniques like PCA, as the focus was on prediction rather than clustering, rendering unsupervised methodologies inapplicable for the task at hand. However, we still included it in our coding for practice purposes.

Conclusion

Drawing from a comprehensive analysis of the Loan Prediction dataset, our research stands as a valuable asset for businesses and financial institutions such as insurance companies, private banking, small holding groups, etc. By harnessing past customer behaviors, organizations can significantly enhance their loan offerings, identifying potential defaulters even before they become clients. This proactive risk assessment not only ensures the reduction of financial losses but also enables firms to create tailored loan products for varying risk profiles, driving profitability. Through detailed data processing and the superior performance of the Random Forest classifier, our research paves the way for businesses to harness data-driven insights, optimizing operations for robust growth in competitive landscapes.