Credit Risk Assessment using Logistic Regression

Building a logistic regression model to assess credit risk for a portfolio of loans, incorporating relevant financial variables.

David Jackson

1. Introduction

Understanding how to manage credit risk in loans and predict whether or not someone will default on a loan is a pivotal role in banking. Accurately forecasting someone's probability of defaulting on a loan significantly impacts profit margins, risk, and growth of banks. Using logistic regression to develop a model that can accurately predict future borrowers, is one of the best ways to do this and is what we will be covering in this project.

2. Collecting the data

To use logistic regression and develop a model, we will need to find a workable data set on lending data. We use the Lending Club data from Kaggle, which is from 2007 through 2019 of Lending Club data on accepted loans. Lending Club also has data on rejected loans, which is not necessary to analyze for this project because there is no data on loan defaults since they were rejected. Please note that this data set is enormous, with over 150 columns and over 2 million rows. Since there are so many column categories to understand what each category means, we need the data dictionary, which was not included on the data card page. We were able to find the data dictionary that someone posted in the comments, but it did not have all of the categories, but most of them. Understanding these categories is crucial to understanding your data so you can start analyzing it. The next thing to do is import the data into Python for analysis.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
accepted_data =
```

```
    pd.read_csv("C:/Users/david/Downloads/accepted_2007_to_2018Q4.sv.gz")
print(accepted_data.columns.tolist())
```

3. Cleaning the data

The first thing we need to do is get rid of columns that are not important or significant for analysis. The first way to get rid of columns is to remove columns that are missing 50% or more of the data in the column. Removing columns that are missing this much data is useful because they are unlikely to provide useful information and may introduce bias and make the analysis hard. The next thing we can do is remove columns with a low variance. This is because columns with low variance will not be statistically significant in the analysis; it is also possible that they are constant, which means that all the elements in the column are the same. The next way we can remove columns is by testing which ones are highly correlated with our target, which is the debt settlement flag, indicating failure to repay a loan as agreed. Debt settlement flag has elements of yes, no, and NaN. Yes for they had a debt settlement flag, no for they did not have a debt settlement flag, and NaN indicates that that spot is missing a response. To analyze this in logistic regression, we need to convert it to binary. For this case we will put yes equal to one and no equal to zero. We will also change the NaN to 0, so everything is numeric. Now that our data is numeric we can remove columns that are not highly correlated with our response variable. Here is the code for that.

```
clean_data = accepted_data
missing_percent = clean_data.isnull().mean()
#fraction of missing per column
threshold = 0.5  # remove columns with >50% missing values
columns_to_drop = missing_percent[missing_percent > threshold].index
```

```
accepted = accepted_data.drop(columns=columns_to_drop)
clean_data_2 = accepted

from sklearn.feature_selection import VarianceThreshold
selector = VarianceThreshold(threshold=0.01) #adjust threshold as needed
accepted_numeric = accepted.select_dtypes(include=['float', 'int'])
#logistic regression only uses numeric
selector.fit(accepted_numeric.fillna(0)) # fill NaNs temporarily
low_variance_cols = accepted_numeric.columns[~selector.get_support()]
accepted_2 = accepted.drop(columns=low_variance_cols)
clean_data_3 = accepted_2

clean_data_3['debt_settlement_flag'].unique()
clean_data_3['debt_settlement_flag_numeric'] =
    clean_data_3['debt_settlement_flag'].map({'Y': 1, 'N': 0})
clean_data_3['debt_settlement_flag_numeric'] =
    clean_data_3['debt_settlement_flag_numeric'].fillna(0)

numeric_data =
    clean_data_3.select_dtypes(include=['float64', 'int64']).copy()
corr_matrix = numeric_data.corr()
corr_with_target = corr_matrix['debt_settlement_flag_numeric']
important_features = corr_with_target[abs(corr_with_target) > 0.05].index
clean_data_4 = numeric_data[important_features]
```

After cleaning we have now gone from 151 columns down to 11 columns. There are still over 2 million rows. We are just going to remove rows that are missing any data in them to make the analysis more accurate. This is how you do that in Python.

```
    threshold = 1  # fraction of data required
min_non_na = int(threshold * clean_data_4.shape[1])
clean_data_5 = clean_data_4.dropna(thresh=min_non_na)
```

After removing the rows with NaN values, we went from 2,260,701 rows to 2,260,668 rows. Although only about 100 rows were removed from the data set, this will make the analysis much easier because we won't have to worry about the missing values, which will cause errors for most statistical functions in Python.

Now that the columns and rows have been cleaned up, we can see what columns we have left using this command.

```
print(clean_data_5.columns.tolist())
['int_rate', 'out_prncp', 'out_prncp_inv', 'total_rec_prncp',
'total_rec_late_fee', 'recoveries', 'collection_recovery_fee',
'last_pymnt_amnt', 'last_fico_range_high', 'last_fico_range_low',
'debt_settlement_flag_numeric']
```

All of the columns that are left are very important for predicting are response variable, debt settlement flag. Before we move into analysis, we want to see what our data looks like to understand what's going on. To do that, we plot a graph of people without a debt settlement flag versus people with a debt settlement flag.

```
status_counts =
clean_data_5['debt_settlement_flag_numeric'].value_counts().sort_index()
plt.figure(figsize=(6,4))
```

```
status_counts.plot(kind='bar', color=['green', 'red'])

plt.xlabel('Debt Settlement Flag')


plt.ylabel('Number of People')

plt.title('Debt Settlement Flag Counts (0 = No, 1 = Yes)')

plt.xticks(ticks=[0,1], labels=['No', 'Yes'], rotation=0)

plt.tight_layout()

plt.show()

plt.savefig(r"C:\Users\david\Downloads\Debt Settlement Flag Counts (0 = No, 1 = Yes)
```

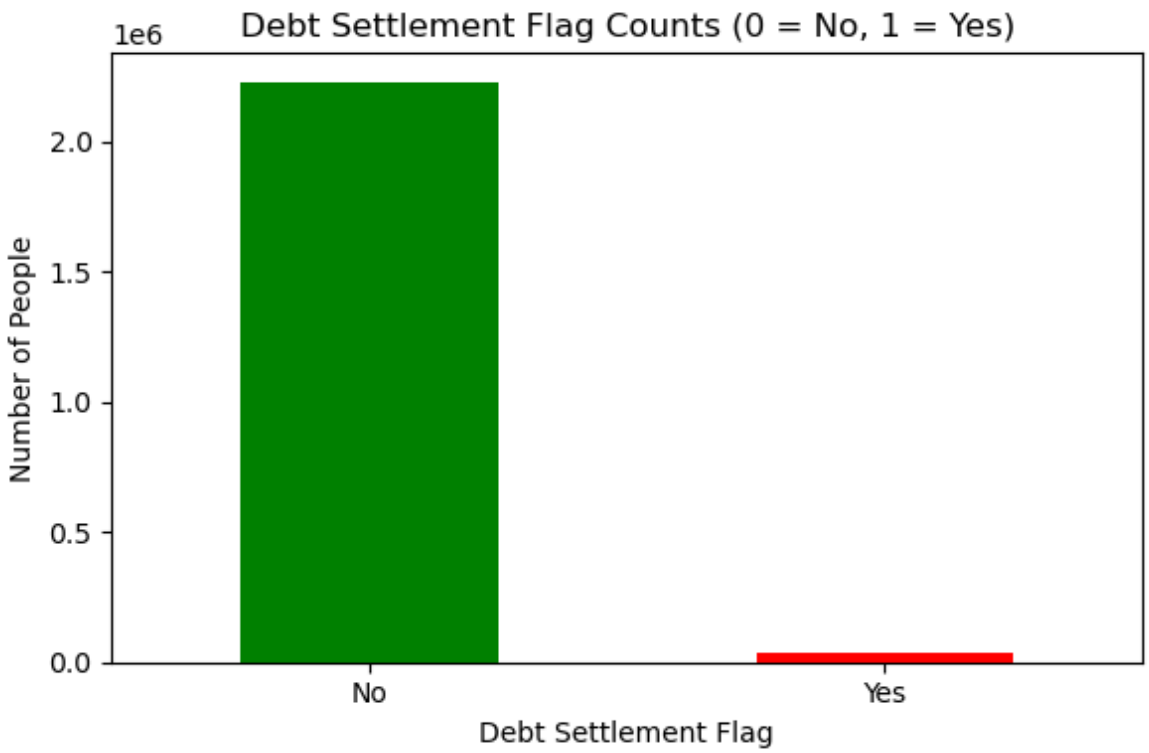

FIGURE 0.0.1.

Note that the number of people who do not have a debt settlement flag is vastly greater than the people who do have a debt settlement flag. This means that our data is highly imbalanced, which can cause a problem for logistic regression because it gives a bias towards the majority class. This can cause errors when trying to predict the minority class. This could cause a big problem with predicting credit risk for loans because there is less data on people who have a debt settlement flag. Some people could get approved for a loan when they shouldn't have because of the lack of data in people who have debt settlement Flags. We will generate synthetic data later on, after analysis of the unbalanced data set, to see the differences between the unbalanced data set and the artificial data set. This will help us understand inaccuracies, errors, and the predictive power in the different data sets.

4. Analyzing the data

The data has been cleaned and is now usable for analysis. The first part of the analysis is making a preliminary model to see how accurate it is, the precision, recall, f1-score, and importance.

```
    Accuracy: 0.9918011032127644
Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00    445282
           1       0.84      0.56      0.68      6852
    accuracy                           0.99    452134
   macro avg       0.92      0.78      0.84    452134
weighted avg       0.99      0.99      0.99    452134
```

```
Confusion Matrix:
 [[444565    717]
 [  2990   3862]]


Top Features:
                    Feature  Importance
6  collection_recovery_fee    0.217688
7          last_pymnt_amnt    0.210901
5               recoveries    0.186241
3          total_rec_prncp    0.140968
0                 int_rate    0.082070
4       total_rec_late_fee    0.051931
9      last_fico_range_low    0.049823
8     last_fico_range_high    0.047758
2            out_prncp_inv    0.006412
1                out_prncp    0.006207
```

Above is the output for some statistics from our preliminary model. As we can see, the accuracy is very high, which is also confirmed by the Confucian matrix by taking the sum of the diagonal divided by the sum of The Matrix, that is $\frac{444565+3862}{444565+717+2990+3862} = 0.9918$.

Precision can be defined as out of everything the model predicted in a predicted class, how many we're correct. This can be calculated using the Confucian Matrix as well. For the predicted class 0, which corresponds to not having a debt settlement flag, the precision is given by the formula $\frac{TN}{TN+FN}$ we can do the same for the predicted class 1, having a debt settlement flag the precision is given by the formula $\frac{TP}{TP+FP}$. (Refer to figure 0.0.2)

Recall can be defined as out of all actual examples in a class how many did the model correctly identify. Using the Confusion Matrix to calculate this is simple. For the actual class 0 the recall is given by the formula $\frac{TN}{TN+FP}$ and for the actual class 1 the formula is given by $\frac{TP}{TP+FN}$. (Refer to figure 0.0.2)

The F1 score is the Harmonic mean of the Precision and recall, which is given by the formula. $\frac{n}{\frac{1}{p}+\frac{1}{r}}$ Where n is the number of data points, since we only have precision and recall, it is two. P represents precision and r represents recall. Using this formula, we can calculate the F1 score for the actual classes zero and one.

The micro average for the Precision score and Recall is just the simple average of both scores across the classes, that is

$$macro_{avg} = \frac{(class0 + class1 + \ldots + classN)}{N}$$

.

The weighted average for the Precision score and recall is the weighted average, which accounts for the support which is given by the formula

$$weighted_{avg} = \frac{(class0 * support0 + class1 * support1 + \ldots)}{total_{support}}$$

.

The important feature comes from the model like Random Forest or XGBoost. We used Random Forest importance tells us which category contributed the most to the predictions in the model. For example, collection recovery fee is the most important feature, and contributes 21.7% to model decisions.

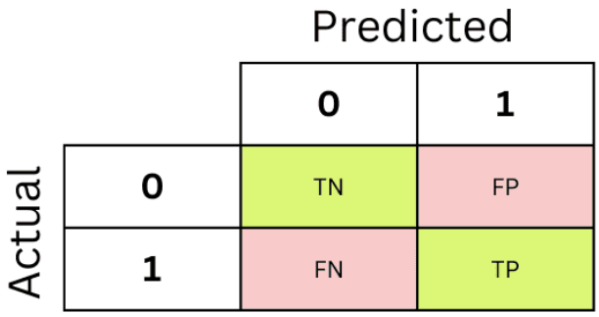

FIGURE 0.0.2. Confusion Matrix

The next analysis technique we used after the preliminary model was the last Lasso (least absolute shrinkage and selection operator) technique. Lasso performs variable selection, which looks for relevant variables, and then performs coefficient shrinkage, which reduces the coefficients of non-relevant variables to zero, so you can remove them to have a simpler model.

```
    from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
lasso = LogisticRegression(penalty='l1', solver='liblinear', C=0.5)
lasso.fit(X_scaled, y)
coef = pd.Series(lasso.coef_[0], index=X.columns)
print(coef.sort_values())
```

```
last_pymnt_amnt          -7.568812
out_prncp                -1.302888
out_prncp_inv            -1.198440
last_fico_range_high     -0.932373
```

recoveries	-0.206746
int_rate	-0.050785
total_rec_prncp	0.060582
total_rec_late_fee	0.134940
last_fico_range_low	0.326455
collection_recovery_fee	0.654289

Looking at the lasso scores, none of them are zero, so all of the variables are relevant to the model. We do not have any justification to remove more variables yet.

Looking for outliers in our data is the next step in analysis. To start, we made a Cook's distance plot to detect any outliers. Note that the data set was too big to make a Cook's distance plot with the whole data set, so we took a sample of 20% of the original data set to make the Cook's distance plot. Because we cannot do the whole data set, we will not be able to detect all of the outliers since we are only looking at 20% of the data, so this Cook's distance plot is a proof of concept. The whole data set could be done very easily on a computer with more memory and RAM.

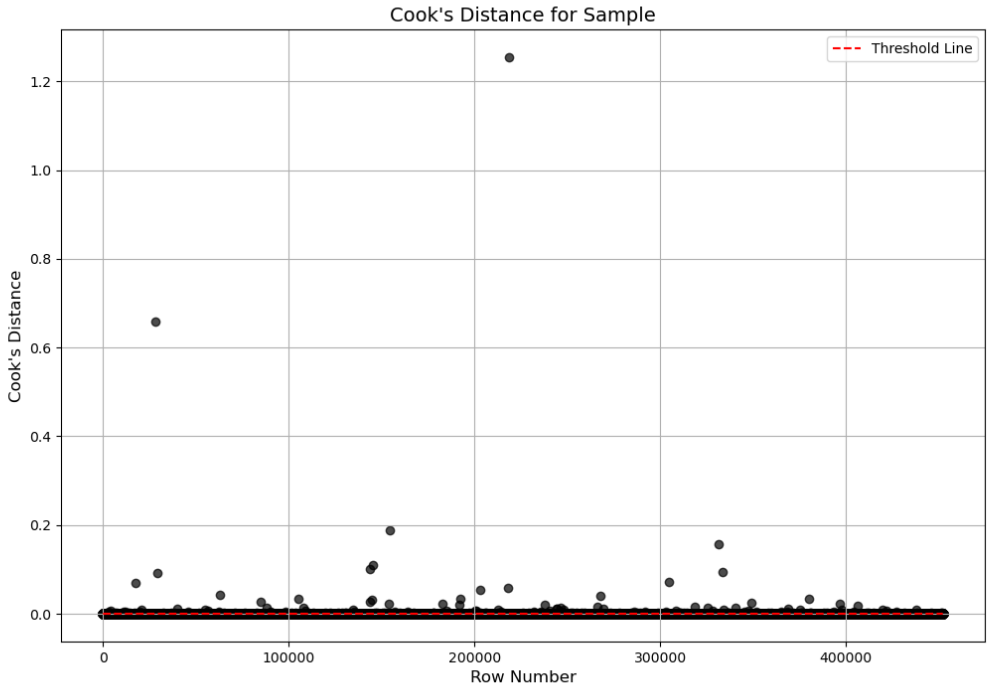

FIGURE 0.0.3.

Looking at the plot, we can see a few points that are way above the threshold, which could be concerning as outliers, but we need more confirmation before we can remove them. To confirm if these are outliers, we will make a residual versus predicted probability plot to identify outliers.

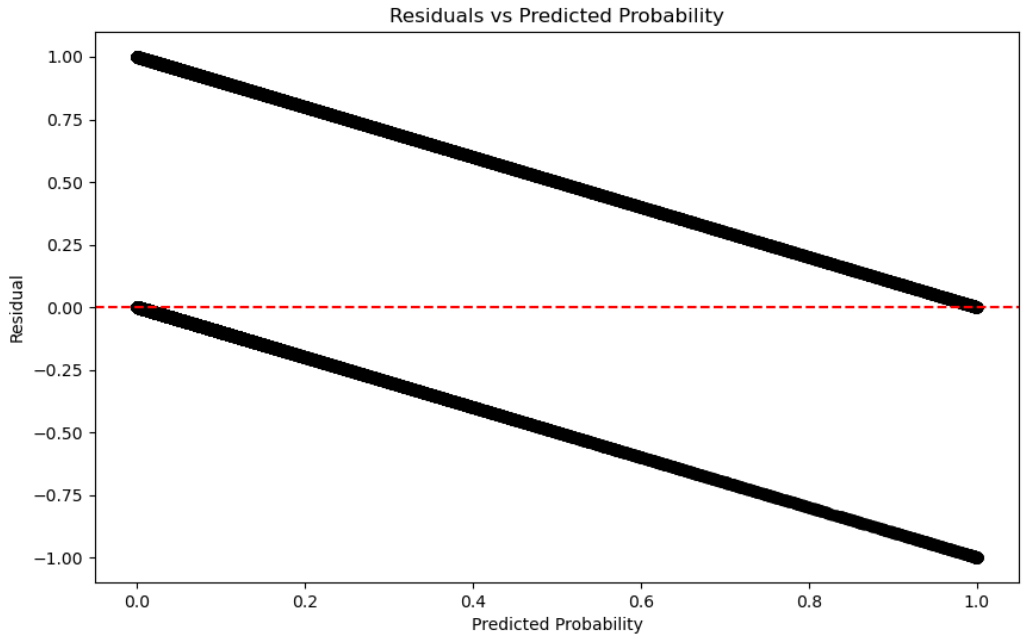

FIGURE 0.0.4.

Looking at the residual plot, it looks a little odd because the dots look like lines, and it's weird how uniform they look, but this isn't abnormal. Residual plots for logistic regression usually look similar to this. Identifying the residuals on The Cook's distance plot was very easy but you cannot see any obvious outliers on the residual plot so we extracted the top 10 Cooks distance values and the top 10 residual plot values and then intersected the two to see if any data points showed up in both the cook's distance plot and the residual plot. That set was empty, meaning that the outliers that showed up in the cook's distance plot didn't show up in the residual plot so we do not have enough evidence to remove any points as outlier. There are other ways to check for outliers, like DF fits plots and DF beta plots, and others, but Cook's distance and residual plots are the main way. Because no outlier showed up in both plots, we will continue with the analysis without removing any data points.

Please remember that for both of these plots, we had to use a sample of our data set for it to run on our computer, so there could be other outliers that were not detected because we only used 20% of the original data for these plots.

Checking for multicollinearity is important for logistic regression if variables are highly correlated with each other the slope coefficients for those variables are unreliable and cannot be used, and essentially mean nothing. This happens because logistic regression assumes that you can hold variables constant, so that one predictor variable can be used while the others are held constant. If variables are highly correlated, you cannot hold one constant while trying to use the other, violating the assumption. To check for multicolinearity we can create a correlation plot heat map.

```
import seaborn as sns
import matplotlib.pyplot as plt
corr_matrix = df_reduced.corr()
plt.figure(figsize=(10, 6))
plt.title('correlation plot')
plt.savefig(r"C:\Users\david\Downloads\correlation plot.png")
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.show()
```

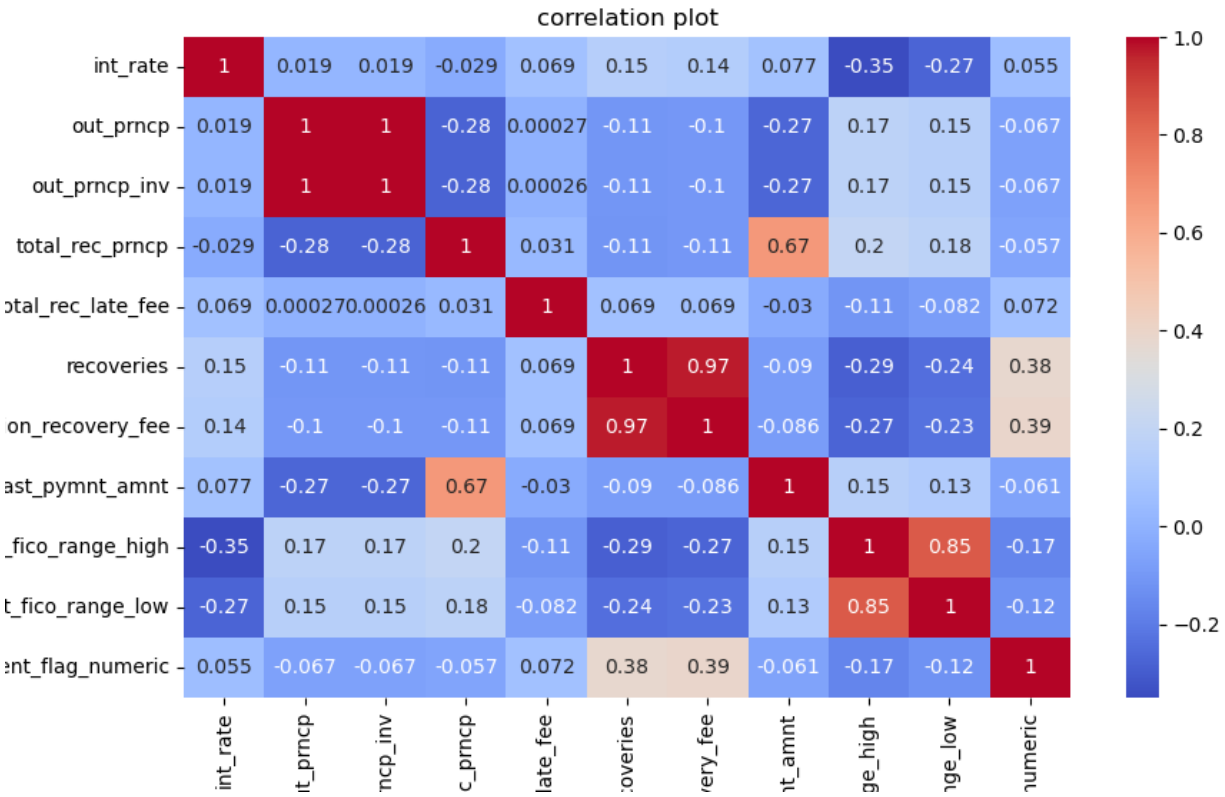

FIGURE 0.0.5.

Looking at the correlation plot, we can see that out_prncp and out_prncp_inv are perfectly correlated with each other, which probably means that there are linear combinations of each other, and they contain the same information. So we are going to get rid of out_prncp_inv. Recoveries and collections recovery fees are also very highly correlated and likely have very similar information, and we are going to remove one of them. We can also see that the last low FICO score and the last High FICO score are highly correlated with each other. We will use principal component analysis to combine these two variables into one. By taking the average of these two, we can combine them into an average FICO score category. By doing this, it allows us to keep the information that is important for predicting, and allows us to avoid multicollinearity issues.

Before we remove any variables or do principal component analysis, we want to get confirmation by using the variance inflation factor to check and verify that these do have a multicollinearity problem.

```
    from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant

# Add a constant column for intercept
X = add_constant(df_reduced)

vif = pd.DataFrame()
vif["Variable"] = X.columns
vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
print(vif)
```

```
 Variable            VIF
0                    const     161.730830
1                 int_rate       1.196802
2                out_prncp  408457.385546
3            out_prncp_inv  408452.709068
4           total_rec_prncp       1.919311
5       total_rec_late_fee       1.027554
6               recoveries      17.539368
7   collection_recovery_fee      17.538874

8           last_pymnt_amnt       1.888300
9     last_fico_range_high       3.948403
```

```
10           last_fico_range_low       3.529668
11  debt_settlement_flag_numeric       1.195325
```

Looking at the variance inflation factors, we can see that out_prncp and out_prncp_inv have a variance inflation factor of 408457.385546 and 408452.709068, respectively. Anything greater than 5 for the variance inflation Factor is considered a problem, and greater than 10 is a serious problem. This gives us a strong statistical reason to get rid of one of these variables; we got rid of out_prncp_inv. Also note that recoveries and collection_recovery_fee have variance inflation factors of around 17, which justifies us in removing one. The high and low FICO scores do not have a very high variance inflation factor, so we may not be justified in removing them, but they still have a high correlation with each other, which is why we will use principal component analysis that keeps information that is important for prediction. The constant term also has a very high variance inflation factor. This is okay and does not need to be worried about because the constant term does not contribute to predictions; it is just there so that the model is not forced to go through the origin. Note that along the diagonal of the plot, they are all perfectly correlated with each other, which is what is supposed to happen since along the diagonal, you are comparing the variable to itself, so its correlation should always be 1.

After removing out_prncp_inv and collection_recovery_fee and can binding the low and high FICO score into an average FICO score, we made another correlation plot to check if we fixed the high colinearity problems.

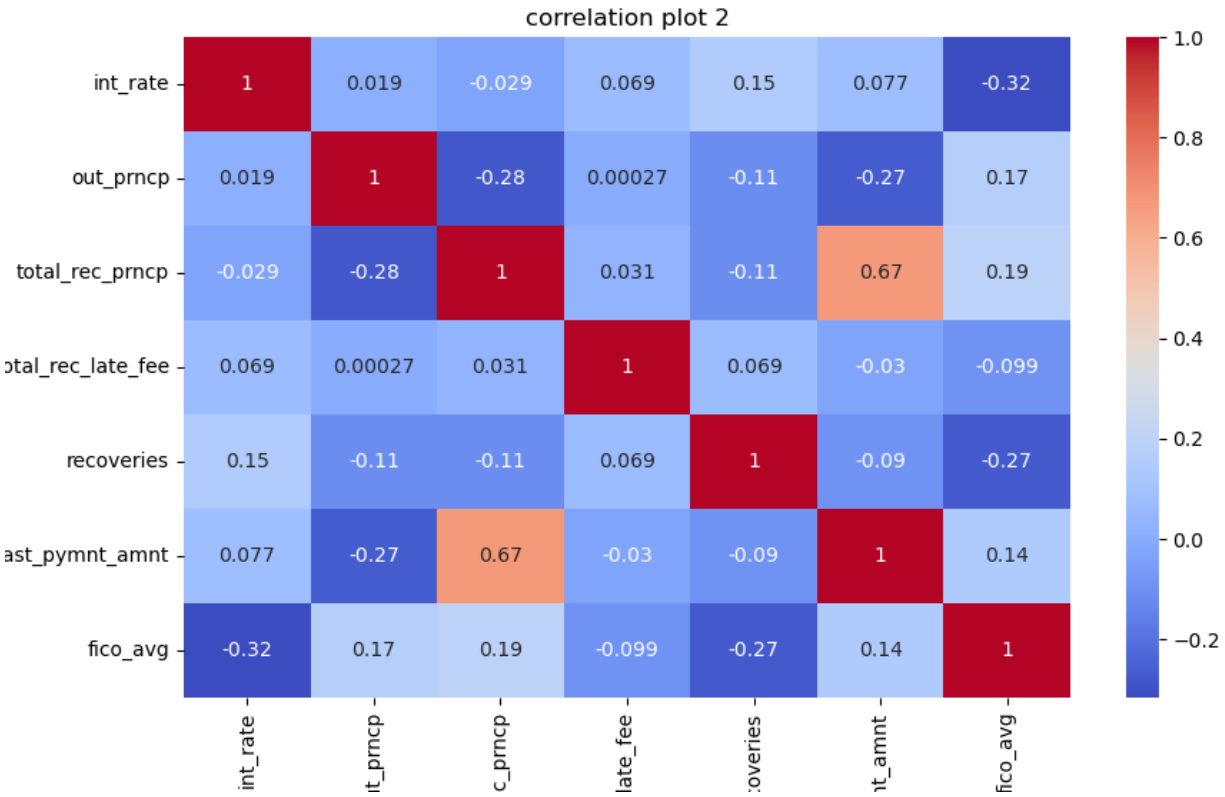

FIGURE 0.0.6.

As we can see, there are no more collinearity problems between any of the variables. The only variables that have a high correlation with each other are total_rec_prncp and last_pymnt_amnt, but their variance inflation factor is small, so there is no justification in removing a variable, and their correlation isn't high enough to be concerning for principal component analysis.

Redoing the VIF to check if there are any variables left with high values, we get the following. All of the variables have a low variance inflation factor, so we have gotten rid of our multicollinearity problem and are ready to start training a model.

	Variable	VIF
0	const	97.263983
1	int_rate	1.160235
2	out_prncp	1.195233
3	total_rec_prncp	1.915648
4	total_rec_late_fee	1.022226
5	recoveries	1.104830
6	last_pymnt_amnt	1.882655
7	fico_avg	1.299012

For linear regression, all of the assumptions must be met. First, we changed the response variable, debt settlement flag, to binary, which is the first assumption. The next assumption is that there is no multicollinearity between predictor variables, which we tested for with the correlation plots and variance inflation factor. Another assumption is that there is a linear relationship between the independent variables and the log odds, looking back to the residual versus predicted probability plots, because there were no weird or unusual trends in the data. The assumption that there is a linear relationship is probably met. Another assumption is having a sufficiently large sample size, which we definitely have because the data set was so large. Lastly another assumption is there's no extreme outliers we looked for outliers using the cook's distance plot and the residual plot and did not see any outliers that overlapped in both plots so we did not have sufficient evidence to remove any although we did have to take a sample of the data set for it to work so there may have been missed outliers but our data set is large enough that outliers should not affect it as much.

Now that all of our assumptions for linear regression are met and our data is clean and ready to be used. We can split the data up into a test group, a training group, and

a validation group, and then make a model to see how our data predicts the response variable. Here is the code for that.

```
    from sklearn.model_selection import train_test_split
predictors = df_final.drop(columns=['debt_settlement_flag_numeric'])
response = df_final['debt_settlement_flag_numeric']
X_train, X_temp, y_train, y_temp =
    train_test_split(predictors, response, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test =
    train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
X_val = scaler.transform(X_val)

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

lg = LogisticRegression(C=0.01, random_state=42)
lg.fit(X_train, y_train)
lg_predictions = lg.predict(X_test)
```

```
print("Logistic Regression Accuracy:",
    accuracy_score(y_test, lg_predictions))
print(classification_report(y_test, lg_predictions))

y_val_pred = lg.predict(X_val)

print(f"Validation Accuracy: {accuracy_score(y_val, y_val_pred)}")
print(classification_report(y_val, y_val_pred))

print("\nConfusion Matrix:\n", confusion_matrix(y_test, lg_predictions))
```

```
Logistic Regression Accuracy: 0.9855883645285622
              precision    recall  f1-score   support

           0       0.99      1.00      0.99    333961
           1       0.64      0.11      0.19      5140

    accuracy                           0.99    339101
   macro avg       0.81      0.56      0.59    339101
weighted avg       0.98      0.99      0.98    339101
```

```
Validation Accuracy: 0.9854379239162488
              precision    recall  f1-score   support

           0       0.99      1.00      0.99    333889
           1       0.65      0.12      0.20      5211

    accuracy                           0.99    339100
   macro avg       0.82      0.56      0.59    339100
weighted avg       0.98      0.99      0.98    339100


Confusion Matrix:
 [[333624    337]
 [  4550    590]]
```

Looking at the training model and the validation model, we can see they have very similar accuracies, indicating the model is not overfitting. We also see that the precision, recall, and F_1 score have gone down from our preliminary model. This is to be expected because we removed more variables, so we are predicting the outcome with less information. The accuracy of the model went down from about 99% to 98%, which means our model is still very accurate, but because we removed the variables, we also avoid multicollinearity, which allows us to use the slope of the variables and interpret what they mean, whereas when we had multicollinearity the slope values cannot be interpreted. Note that multicollinearity does not cause an issue for predicting; it is okay to have multicollinearity for predictions, but if you do have multicollinearity slope values will lose their meaning.

The next thing we tested with this model is getting the p-values for are variables to make sure they are significant in the model, which are given below in the following table. Looking at the p-value column, we see that all of the p-scores are less than 0.01, which means that all of our variables are highly significant.

	coef	std err	z	P>|z|
const	-9.2986	0.074	-125.651	0.000
x1	0.0204	0.008	2.704	0.007
x2	-3.0419	0.054	-56.591	0.000
x3	-0.0590	0.012	-5.087	0.000
x4	0.1514	0.003	44.258	0.000
x5	0.4664	0.003	145.220	0.000
x6	-7.4359	0.133	-55.963	0.000
x7	-0.2253	0.005	-44.401	0.000

To assess the logistic regression model's overall accuracy and to understand the model, you'd usually plot a logistic regression plot, but because we have more than one predictor variable, you cannot do that; it is better to plot a precision-recall curve, which is what we did next. 1.52% of the data is in the positive class that is they have a debt settlement flag. Looking at the average precision score, we have a 0.29 or 29%. Which means our model is well above the baseline of 1.52% and is predicting well beyond random chance. Reading the graph from left to right, we see high precision and low recall, which means the model is predicting a lot of positives but not catching the actual positives as we move left. Recall increases, and precision decreases, which is expected.

As the model tries to predict more positives, the precision decreases; this is expected for a data set that is this imbalanced.

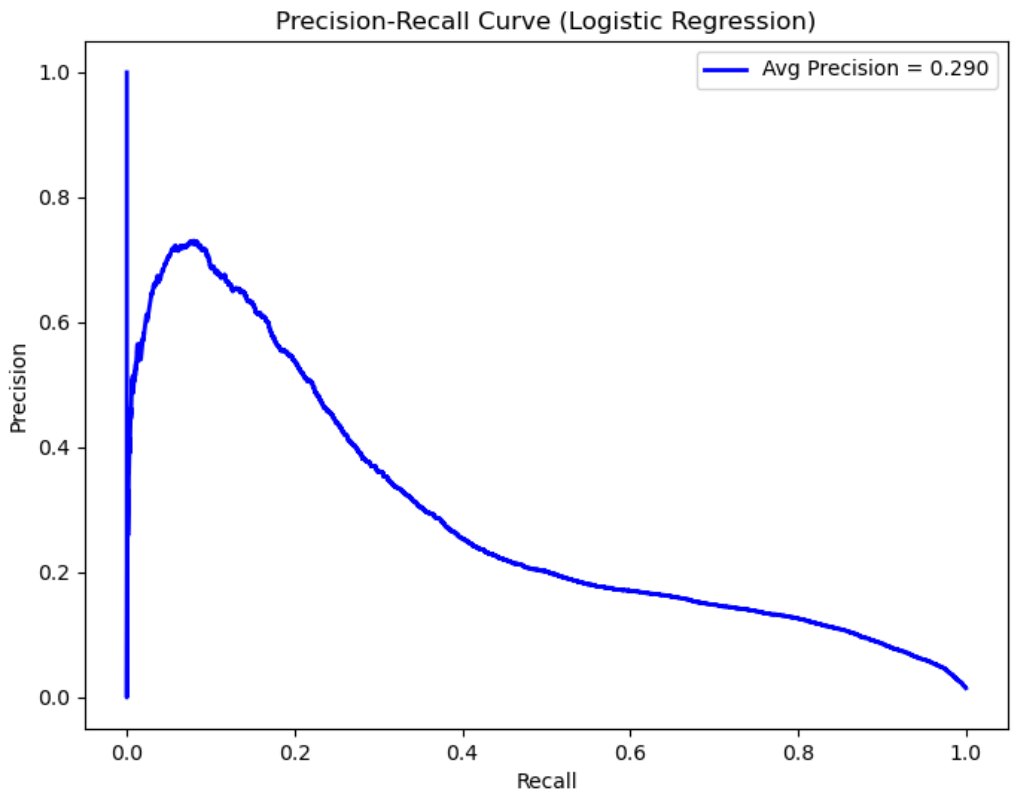

FIGURE 0.0.7.

Because the data set is so unbalanced, the next thing to do is balance the data set. To do this, we will use SMOTE (Synthetic Minority Oversampling Technique). SMOTE targets the minority class for our case, that is, people who have a debt settlement flag, and it finds the nearest neighbor using the k-nearest neighbors algorithm. Then it generates a synthetic data point between the selected point and its nearest neighbor by taking a weighted average where the weights are determined by a random number.

Doing this achieves new data points that are synthetically generated to help it balance the data. This is the Python code below. Looking at the output we can see that SMOTE balanced the data set, and now we are ready for further analysis.

```
    from imblearn.over_sampling import SMOTE
# Oversample training set
smote = SMOTE(random_state=42)
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)
print("Before:", y_train.value_counts())
print("After:", pd.Series(y_train_res).value_counts())


Before: debt_settlement_flag_numeric
0    1558572
1      23895
Name: count, dtype: int64
After: debt_settlement_flag_numeric
0    1558572
1    1558572
Name: count, dtype: int64
```

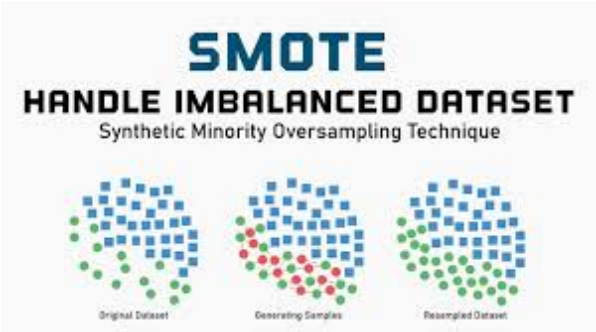

FIGURE 0.0.8.

```
Validation Accuracy: 0.8793335299321734
              precision    recall  f1-score   support

           0       1.00      0.88      0.93    333889
           1       0.11      0.93      0.19      5211

    accuracy                           0.88    339100
   macro avg       0.55      0.91      0.56    339100
weighted avg       0.99      0.88      0.92    339100

Test Accuracy: 0.8793338857744448
              precision    recall  f1-score   support

           0       1.00      0.88      0.93    333961
           1       0.11      0.94      0.19      5140

    accuracy                           0.88    339101
   macro avg       0.55      0.91      0.56    339101
weighted avg       0.99      0.88      0.92    339101


Confusion Matrix:
 [[293363  40598]
 [   320   4820]]
```

Making a logistic regression model for the balance data set, we got the output given above. Looking at the accuracy, we can see that it has gone down from our previous model. This is expected since we added synthetic data; we increased the number of people who have debt settlement flags artificially, so there is more to predict. 87% for model accuracy is still accurate and great for predictions. Although the accuracy went down, the synthetically generated balance data might be better than the original data because it won't have a bias towards people who don't have a debt settlement flag. The precision decreased, and the recall increased. Thinking back to the Precision recall curve, as we increased the recall, the Precision decreased, which allowed us to predict positive debt settlement flags more accurately. This makes sense for our balance data. After all, we increased the number of positive debt settlement flags, which would increase the recall and decrease the precision because we'll be able to predict positive debt settlement flags more easily. You may also notice that the support column for the number of people that do not have a debt settlement flag and the people that do have a debt settlement flag are not even even though we balanced the data using SMOTE this is because although we train the model on the balance data we still test and validate the model using the original data set.

Now we will look at the p-values to see how significant are variables are when the model is trained on balanced data in the following table.

	coef	std err	z	P>|z|
const	6.6628	0.020	326.902	0.000
int_rate	0.0249	0.000	55.997	0.000

```
out_prncp             -0.0002   8.21e-07   -278.467      0.000
total_rec_prncp    -2.536e-05   3.58e-07    -70.848      0.000
total_rec_late_fee     0.0476      0.000    231.664      0.000
recoveries             0.0009   2.22e-06    399.860      0.000
last_pymnt_amnt       -0.0004   1.96e-06   -222.266      0.000
fico_avg              -0.0112   2.94e-05   -379.652      0.000
==============================================================
```

Looking at the p-values, we see that when trained with the balanced data, all of the variables are very significant. The interest rate also dropped to zero on the p-value, whereas before the data was balanced, it was 0.007.

Now that we have balanced data, we are going to use a ROC or receiver operating characteristic curve, which is a plot that visualizes the performance of a binary model at different threshold settings. It plots the true positive rate or sensitivity given by

$$Sensitivity = \frac{TP}{TP + FN}$$

this is the sames as the recall for class 1, against the false positive rate, which is $1 - specificity$ where

$$Specificity = \frac{TN}{TN + FP}$$

this is the sames a the precision for class 1. ROC curves are used to evaluate the trade-off between correctly identifying positive cases and incorrectly identifying negative cases.

Looking at the ROC plot below, we can see that the area under the curve value is 0.946, which indicates very strong separation power between the positive and negative classes. The curve rises steeply toward the top-left corner. This means the model is very effective at distinguishing positives from negatives at most thresholds. We can also

see that the model is above the random guess line, which means our model is not just randomly guessing.

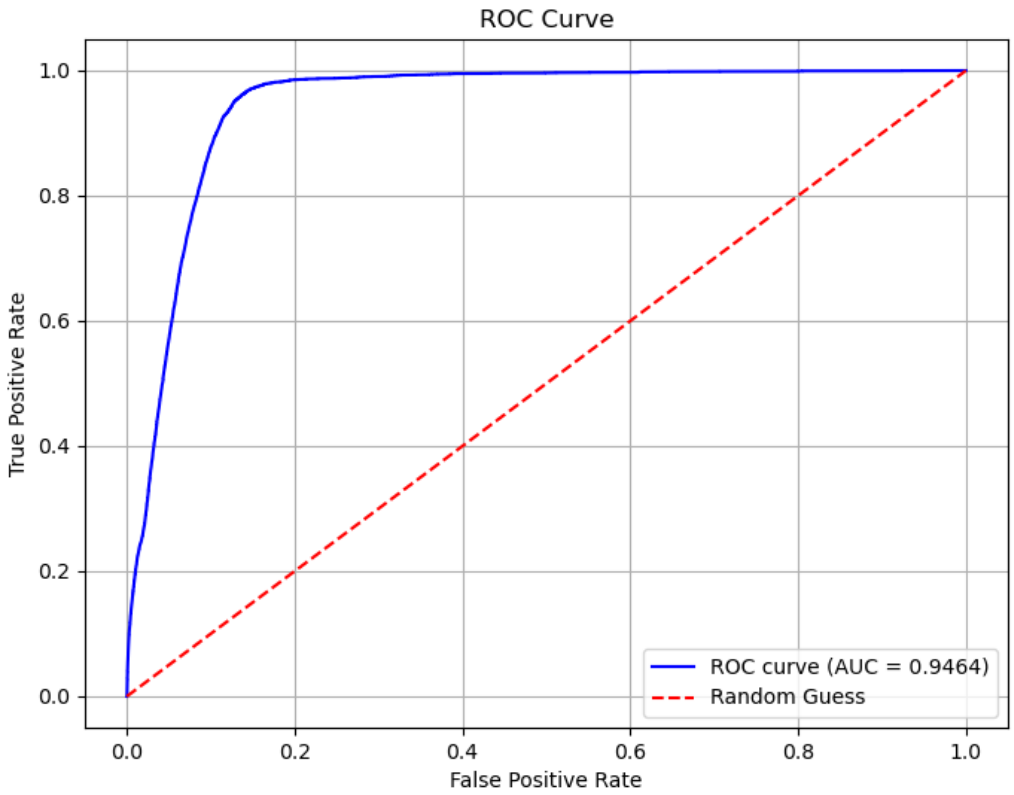

FIGURE 0.0.9.

The next thing we can do is find the maximum threshold using Youden's J statistic. The default threshold is 0.5, and after finding a maximum solution, we got the optimal threshold of 0.4337 and a true positive rate of 0.958, and a false positive rate of 0.136. Recall that the true positive rate before was 0.93, and the false positive rate was 0.11. Increasing the true positive rate it's helpful because it means we will identify more people who are likely to default on a loan, but we also increase the false positive rate, which

means we will classify more people as likely to default on a loan when they are not. For banking, it is more important to identify people who are likely to default on a loan and misclassify some people in exchange. This is especially important for predicting diseases; it is much better to classify someone as having a disease when they don't, rather than misclassify someone who has a disease as not having a disease.

Conclusion

In this project, we used logistic regression to predict the probability of someone defaulting on a loan. Starting with cleaning the data, we remove columns with missing values, low variance, and low correlation with the target variable. Then we removed rows with missing values. This way, we had no missing values in our data set, making it easy to work with. The second thing we did was look at a graph of people with or without debt settlement Flags. Then we made a preliminary model to see how well the data would predict the response variable, before analysis, so we could see what changed after our analysis. The next thing to do was to start analyzing our data. We used lasso to see if we could reduce any variables that were not significant. Then we made a Cook's distance plot to identify outliers alongside the residual versus predicted probability plot so we could detect outliers and check overall model fit. We also made a correlation plot and got the variance inflation factors, and found that some of our variables are highly correlated with each other and had very high variance inflation factors, so we removed some of them and performed principal component analysis on the FICO scores to make an average FICO score. After doing that, we redid the correlation plot and variance inflation factors, and we solved our multicollinearity problem. The 4th part of analyzing the data was to set up another model with a training group test group, and a validation group. Then we found the p-values for this model to check what variables were significant. At this point, we were still working with the unbalanced data, so we

made a precision-recall curve to analyze how well our model performed with unbalanced data. The Next Step was to balance the data using the SMOTE function. After the data was balanced, we set up another model and trained it on the balanced data, then tested and verified the model using our actual data to see how well the artificial data was at predicting loan defaults. We also checked for variable significance using p-values for this model as well. The second-to-last thing we did was an ROC curve since the data was balanced, to see the effectiveness at distinguishing positive from negative classes at different thresholds. Finally, we found the Youden's J statistic in order to find an optimal threshold for predicting in the model.

Python Code

The complete Python source code for this project is available in this GitHub Repository. (If this line does not work, you may need to download it from GitHub.) You may need to download to see the full code, as GitHub will not display all of it or render the code block sometimes. Note that not all of the code was included in the paper, and some of the outputs were simplified for formatting.

Key Takeaways

1. **Logistic Regression is Effective for Binary Outcomes**
   - LR models the probability of an event happening, like success and failure as an output of probabilities between 0 and 1.
2. **Coefficients Provide Interpretability**
   - Model coefficients can be translated into odds ratios, helping explain how predictors increase or decrease the likelihood of the target event.
3. **Class Imbalance**
   - With rare positive cases, accuracy alone can be misleading. Metrics like precision, recall, F1-score, and AUC give a more complete picture of performance.

4. **Threshold Selection is Crucial**
   - Adjusting the classification threshold can optimize trade-offs between sensitivity and specificity.
5. **Model Performance**
   - Using proper scaling, encoding categorical variables, and removing multicollinearity improve the accuracy and interpretability of the model.
6. **Validation is Essential**
   - Splitting data into train, test, and validation sets and using cross-validation ensures results are not just due to chance or overfitting.
7. **ROC and PR Curves**
   - ROC curves highlight the balance between true positives and false positives.
   - Precision-Recall curves are especially informative for imbalanced datasets.
8. **Limitations to Keep in Mind**
   - Logistic regression assumes linear relationships between predictors and log-odds.
   - Outliers, high multicollinearity, and irrelevant features can degrade performance.
9. **Practical Value**
   - The model can be applied to make real-world predictions, such as credit risk, fraud detection, loan default prediction, or medical diagnosis.

References

[1] Cross Validated. (n.d.). Retrieved August 19, 2025, from https://stats.stackexchange.com/

[2] All Lending Club loan data. (n.d.). Kaggle. Retrieved August 19, 2025, from https://www.kaggle.com/datasets/wordsforthewise/lending-club/code

[3] Chawla, N., Angelou, R., & Khan, F. (2024, August 6). Balancing the Scales: How SMOTE Transforms Machine Learning with Imbalanced Data. *Python in Plain English*. Retrieved August 19, 2025, from https://python.plainenglish.io/balancing-the-scales-how-smote-transforms-machine-learning-with-imbalanced-data-a6d3367254cd

[4] Documentation. (n.d.). Overleaf. Retrieved August 19, 2025, from https://www.overleaf.com/learn

[5] George, N. (2019). All Lending Club loan data. Kaggle. Retrieved August 19, 2025, from https://www.kaggle.com/datasets/wordsforthewise/lending-club/data

[6] Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning*. Penn Arts & Sciences. Retrieved August 19, 2025, from https://www.sas.upenn.edu/~fdiebold/NoHesitations/BookAdvanced.pdf

[7] James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An Introduction to Statistical Learning*. UC Berkeley Statistics department. Retrieved August 19, 2025, from https://www.stat.berkeley.edu/~rabbee/s154/ISLR_First_Printing.pdf

[8] Lasso (statistics) - Wikipedia. (n.d.). Wikipedia, the free encyclopedia. Retrieved August 19, 2025, from https://en.wikipedia.org/wiki/Lasso_(statistics)

[9] Newest 'statistics' Questions - Mathematics Stack Exchange. (n.d.). Math Stack Exchange. Retrieved August 19, 2025, from https://math.stackexchange.com/questions/tagged/statistics

[10] Selvaraj, N. (2022, November 9). Confusion Matrix, Precision, and Recall Explained. *KDnuggets*. Retrieved August 19, 2025, from https://www.kdnuggets.com/2022/11/confusion-matrix-precision-recall-explained.html