Use Machine Learning to Classify Customer Credit Score

October 16, 2024

1 Introduction

For a bank, classifying customers' credit scores is essential for managing risks and making informed loaning decisions [1]. However, this process can often be multidimensional and complex, presenting significant decision-making challenges [2]. With the development of Machine Learning algorithms, automating credit score classification has become a viable solution to enhance accuracy and efficiency [3]. This Machine Learning project aims to create a categorical classification algorithm that utilizes supervised learning techniques on credit-related factors to classify customers' credit scores into three classes: good, standard, and poor.

The report is structured as follows: Section 1 introduces the motives and lays out the structure. Section 2 formulates the problem by explaining the dataset, data points, features, labels, and data sources. Section 3 discusses the Machine Learning methods utilized to solve the problem. The methods' outcomes are then presented in Section 4, and Section 5 concludes and discusses future scope.

2 Problem Formulation

2.1 Dataset

The Dataset for Credit Score Classification from Kaggle [4] contains basic bank details and credit related information of 100,000 customers with 32 variables. Variables include age, occupation, annual income, monthly inhand salary, number of bank accounts, etc.

2.2 Features and Label

This Machine Learning problem is framed as a supervised learning task, where a machine learning model is trained using labeled data from the dataset. We extract a total of 13 features (Table 1). The label variable takes integer values: 1 for "Good", 0 for "Standard", and -1 for "Poor", which depends on the financial behavior of the customers in the dataset.

Features	**Descriptions**
Monthly_Balance	Monthly balance amount of the customer (in USD). Type: continuous
Annual_Income	Annual income of the customer. Type: continuous
Num_Bank_Accounts	Number of bank accounts the customer holds. Type: discrete
Num_Credit_Card	Number of credit cards the customer holds. Type: discrete
Delay_from_due_date	Average number of days delayed from the payment due date. Type: discrete
Interest_Rate	Interest rate on credit card. Type: continuous
Credit_Mix	Type of the mix of credits that the customer has. '-1' represents 'Bad', '0' represents 'Standard' and '1' represents 'Good'. Type: categorical
Num_of_Loan	Number of loans taken by the customer (the maximum value is 9). Type: discrete
Num_of_Delayed_Payment	Number of payments delayed by customer. Type: discrete
Outstanding_Debt	Remaining debt to be paid by customer (in USD). Type: continuous
Credit_History_Age	Length of the customer's credit history. Type: continuous
Num_Credit_Inquiries	Number of credit card inquiries made by lenders on customer. Type: discrete
Payment_of_Min_Amount	Indicates whether the customer has paid the minimum required amount. '0' represents 'No' and '1' represents 'Yes'. Type: binary

Table 1: Features and Descriptions

3 Methods

3.1 Data pre-processing

The dataset contains 100,000 rows and 32 columns. The column 'Credit_Score' was renamed to 'Label', and columns with string values were mapped to integer values as specified in Table 1. The labels 'Good', 'Standard', and 'Poor' have 17.828, 53.174, and 28.998 data points, respectively, which indicates the need for weighting in future analysis. The dataset has 32 features, which means feature selection is essential. To simplify, the columns 'Customer_ID', 'Occupation', and 'Last_Loan_1' to 'Last_Loan_9' were excluded. We generated a correlation table between the 'Label' column and the remaining features (Table 4), retaining only those with an absolute correlation greater than 0.2. Next, we calculated the correlation among these selected features. For both models, we set a maximum correlation threshold of 0.8. We identified highly correlated pairs using a correlation table (Table 5) and heatmap, then eliminated one feature from each pair to minimize the total number of removed features while maintaining alow correlation between the remaining features. This process resulted in 13 features in the final dataset.

3.2 Dataset splitting

To address the issue of imbalanced datasets (approximately 50:30:20 between classes), we will use Stratified Splits. This approach divides the dataset into subsets while ensuring that the class distribution in each subset matches the overall class distribution of the original dataset. The training-testing size ratio will be 90:10 to optimize the training part, ensuring that the model has sufficient data to learn from while still retaining enough data for effective validation and testing of its performance.

3.3 Method 1: Random Forest Classifier

For our first supervised machine learning algorithm, Random Forest Classification, a model that aggregates the results of multiple decision trees to determine the final prediction, is chosen. By choosing Random Forest over Decision Tree, we can, to some degree, reduce the risk of over-fitting and bias.

In our experiments with the Random Forest Classifier, we evaluated two popular loss functions: Gini impurity and entropy. After conducting two tests, the model using the Gini loss function returned an accuracy of 82.09%, while the entropy-based model achieved a slightly lower accuracy of 81.93%. Both loss functions are approximately equally effective in measuring the quality of splits in decision trees, hence, for simplicity, we decided to use the default loss function of Random Forest Classifier, which is Gini impurity. The function is defined as:

$$Gini = 1 - \sum_{i=1}^{n} p_i^2$$

Where:

- p_i represents the probability of class i,
- n is the number of classes.

3.4 Method 2: Logistic Regression

For our second supervised machine learning algorithm, we select Logistic Regression, a statistical method used for binary classification, predicting the probability that a given input belongs to one of two classes. Although Logistic Regression is normally used for binary classification, we can use the scikit-learn package to apply this algorithm to multi-class classification through multinomial approach [5]. This approach is well-suited for our dataset, which consists of three classes: Good, Standard, and Poor, with an approximate ratio of 5:3:2. This method is also now built-in and automatically applicable for multi-class classification with Logistic Regression. Furthermore, for experimental purposes, we will utilize this parametric method to compare it with the previously discussed non-parametric method.

For our model, the loss function we use is the cross-entropy loss (softmax loss function) specifically for multinomial logistic regression defined as:

$$L(w) = -\sum_{i=1}^{n}\sum_{j=1}^{k} y_{ij} \log\left(\frac{e^{w_j^T x_i}}{\sum_{l=1}^{k} e^{w_l^T x_i}}\right)$$

Where:

- w_j is the weight vector for class j.
- y_{ij} is a binary indicator (0 or 1) that is equal to 1 if the sample i belongs to class j, and 0 otherwise.
- $e^{w_j^T x_i}$ is the exponentiated linear combination of the features for class j.
- The denominator $\sum_{l=1}^{k} e^{w_l^T x_i}$ ensures that the probabilities sum to 1.

[6]

This loss function is chosen because it is the standard loss function for multinomial logistic regression. By penalizing confident incorrect predictions, it encourages the model to improve its accuracy in distinguishing between classes. Additionally, this loss function is readily implemented in the scikit-learn library [5], allowing for seamless integration into our codebase. To evaluate the model's performance, we will use the ratio of the number of correct predictions to the total number of predictions.

For a more comprehensive and visual evaluation, in both of our methods, we will use a confusion matrix, along with accuracy, precision, recall, and F1-scores, all of which are built in scikit-learn library [5].

4 Result

4.1 Evaluation Metrics

The performance of the two models will be evaluated by 4 metrics:

- **Accuracy** calculates the overall correctness of the model by calculating the ratio of correctly predicted instances to the total instances in the dataset.
- **Precision (weighted)** calculates the precision for each class while considering the number of true instances for each class.
- **Recall (weighted)** calculates the recall for each class while considering the number of true instances for each class.
- **F1-score (weighted)** is the harmonic mean of precision and recall while considering the number of true instances for each class.

[5]

After each fold of the model, an analysis was performed to obtain the accuracy, weighted precision, weighted recall, weighted F1 score, and the weighted confusion matrix. The final result of a model is the average of these values across all folds.

4.2 Random Forest Classifier's Outcome

See Figure 3 for the confusion matrix. With all metrics (accuracy, weighted precision, weighted recall, and weighted F1-score) around 82% in the test result as demonstrated by Table 2, Random Forest Classifier demonstrates a more robust and consistent performance. Notably, it classified 83.62% of individuals with bad credit correctly, outperforming that number for the good (78.18%) and standard (82.57%) credit categories. This demonstrates the model's effectiveness in identifying high-risk clients, which is crucial for minimizing potential losses. It also has very few severe misclassifications, with only 0.38% instances of "Poor" being classified as "Good" and 0.28% instances of "Good" being classified as "Poor".

Accuracy	Precision	Recall	F1-Score
82.09%	82.11%	82.09%	82.09%

Table 2: Performance Metrics for Random Forest Classifier

4.3 Logistic Regression's Outcome

To tune the hyperparameters, we used the GridSearchCV algorithm which is built-in in the scikit-learn library [5]. This runs the model using all combinations of hyperparameters related to the machine learning algorithm, each with 5-fold cross-validation. In the end, the best model is chosen and it produces the following results in the test set:

Accuracy	Precision	Recall	F1-Score
65.54%	69.83%	65.54%	65.89%

Table 3: Performance Metrics for Logistic Regression

As we can see from Table 3 and the confusion matrix in Figure 4. All metrics of the model remain below 70%, indicating that there is significant room for improvement. While it correctly classifies 84.86% of individuals with good credit, it only manages to classify 66.79% of individuals with bad credit correctly. Additionally, the model misclassifies 15.34% of individuals with bad credit as having good credit. For a bank or financial organization, this suggests that the model is insufficient for reliable risk assessment.

4.4 Model Comparison

Looking at the average metrics of the two models, it is clear that the Random Forest Classifier outperforms Logistic Regression in all 4 metrics. This indicates the overall efficiency of the Random Forest model in handling the classification task, making it a more robust choice. However, there is one aspect in which Logistic Regression performed better than Random Forest Classifier, which is identifying individual with good credit score correctly. While Random Forest Classifier can identify 78.18% of good credit individuals, that number of Logistic Regression is 84.85%, significantly higher.

After evaluating all these factors, we chose the Random Forest Classifier as our final machine learning method due to its superiority in overall performance compared to Logistic Regression, achieving a final test accuracy of 82%.

5 Conclusion

With the chosen dataset and features, both models can predict the customer credit score with acceptable accuracy. Obviously, there is still room for improvement in both models.

For Random Forest Classifier, due to lack of time and resources, we did not have the chance to tune hyperparameters. Additionally, we did not have the opportunity to implement the customized loss function we discovered in the Science Direct article titled "Entropy-Based Classifier Enhancement to Handle Imbalanced Class Problem." [7].

For Logistic Regression, we have explored various methods to enhance accuracy. We've applied feature scaling (normalization), implemented the One-vs-Rest approach, tuned hyperparameters, and adjusted the correlation threshold for features, etc. However, this is the most effective approach we've found so far. This suggests the need to conduct further research and to search for better a dataset with balanced class distribution and more effective features.

One surprising discovery we made is that, out of 21 initial features, Monthly Inhand Salary and Annual Income do not rank among the top 10 features with the highest absolute correlation to the label. This suggests that other factors, potentially related to spending habits and financial behavior, may play a more significant role in predicting the label than income.

References

[1] Kumar, K. V., & Rao, S. H. (2012). Credit Rating-Role in Modern Financial System. *International Journal of Marketing, Financial Services & Management Research*, 1(8), 126-138.

[2] Bai, C., Shi, B., Liu, F., & Sarkis, J. (2019). Banking credit worthiness: Evaluating the complex relationships. Omega, 83, 26-38.

[3] Kennedy, K. (2013). Credit scoring using machine learning.

[4] Ayush Sharma. Dataset for Credit Score Classification (Kaggle). https://tinyurl.com/nz5jxn32

[5] Pedregosa, F., Varoquaux, Ga"el, Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... others. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12(Oct), 2825–2830.

[6] Ng, Andrew, et al. "Softmax Regression." Unsupervised Feature Learning and Deep Learning. Stanford University. https://tinyurl.com/5fwbtax7

[7] Kirshners, A., Parshutin, S., & Gorskis, H. (2017). Entropy-based classifier enhancement to handle imbalanced class problem. Procedia Computer Science, 104, 586-591.

Appendix

1 Table of Features and Their Correlations with Label

Features	**Correlation**
Credit Utilization Ratio	0.045793
Total EMI per month	0.070619
Spending Behavior	0.090938
Payment Behavior	0.111699
Amount invested monthly	0.154277
Age	0.160356
Changed Credit Limit	0.169506
Monthly Balance	0.208506
Monthly Inhand Salary	0.209921
Annual Income	0.212606
Num of Loan	0.358355
Num of Delayed Payment	0.372963
Outstanding Debt	0.386525
Num Bank Accounts	0.388166
Credit History Age	0.388789
Num Credit Card	0.404428
Delay from due date	0.431591
Num Credit Inquiries	0.435705
Payment of Min Amount	0.443817
Interest Rate	0.485409
Credit Mix	0.498673
Label	1.000000

Table 4: Features and Their Correlations with Label

2 Table and Heat Map of Features Correlations

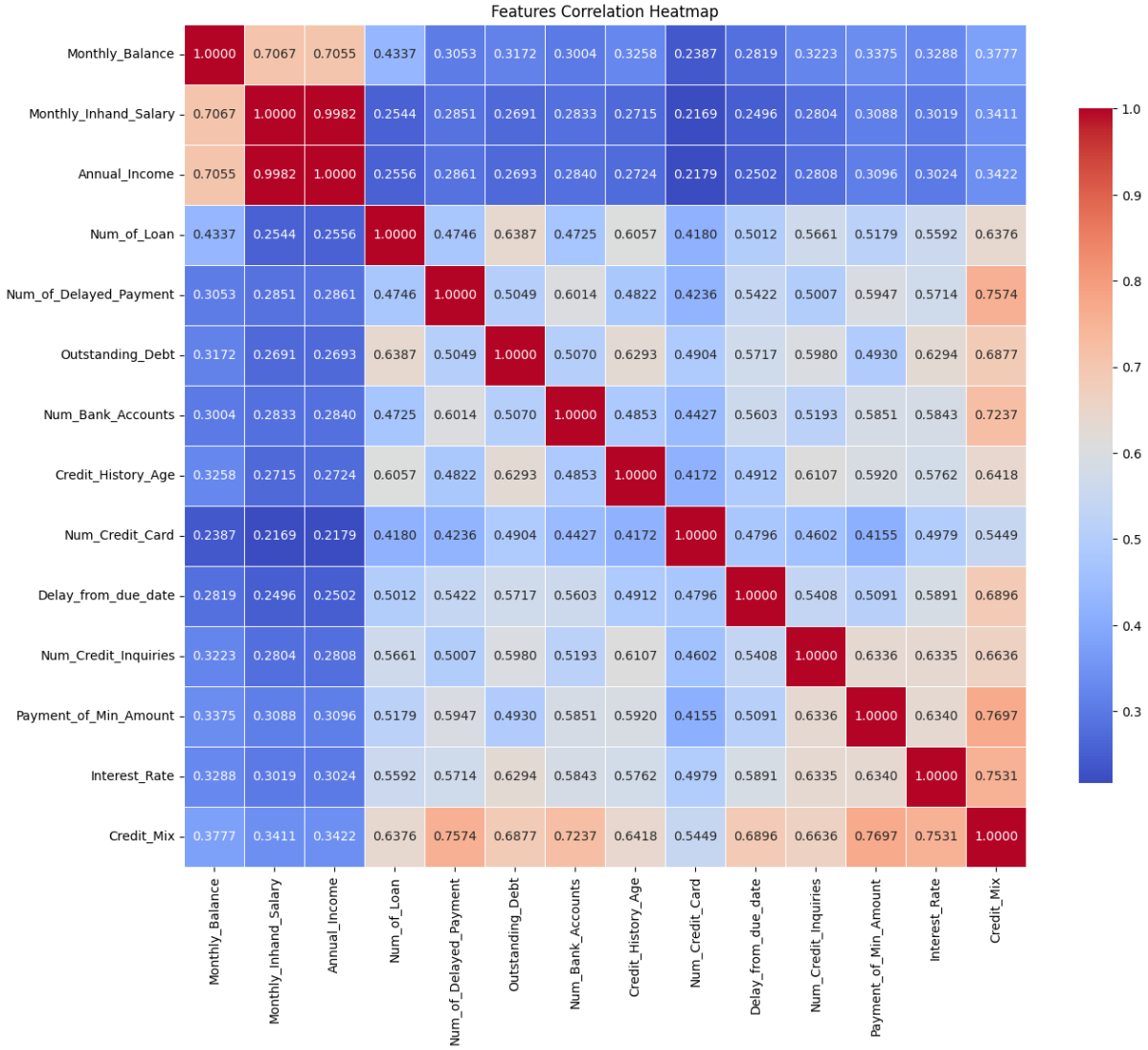

Figure 1: Heat Map of Feature Correlations

Feature 1	**Feature 2**	**Correlation**
Monthly Inhand Salary	Annual Income	0.998201
Credit Mix	Payment of Min Amount	0.769721
Credit Mix	Num of Delayed Payment	0.757443
Credit Mix	Interest Rate	0.753124
Num Bank Accounts	Credit Mix	0.723669
Monthly Balance	Monthly Inhand Salary	0.706741
Annual Income	Monthly Balance	0.705528

Table 5: Feature Correlations

3 Category Frequency of Label

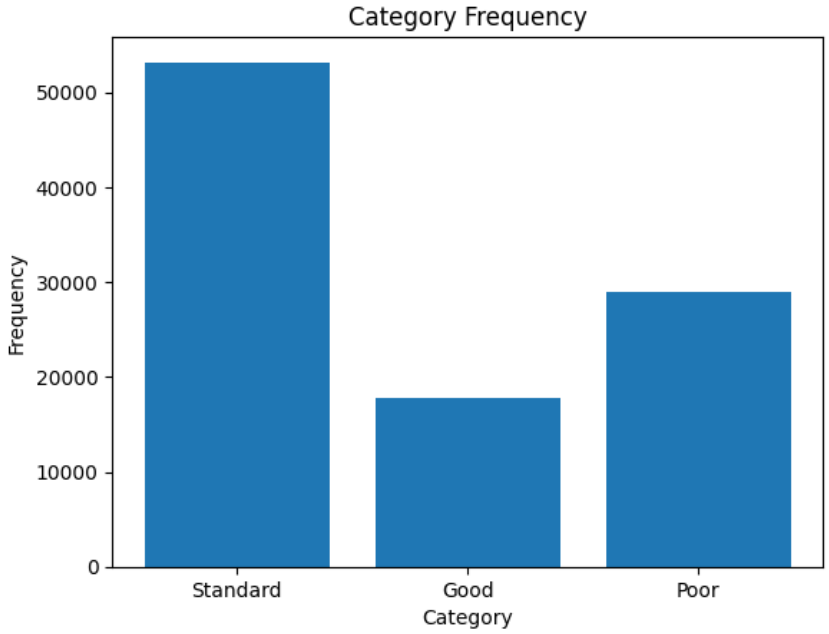

Figure 2: Category frequency of label

4 Confusion Matrices for 2 methods

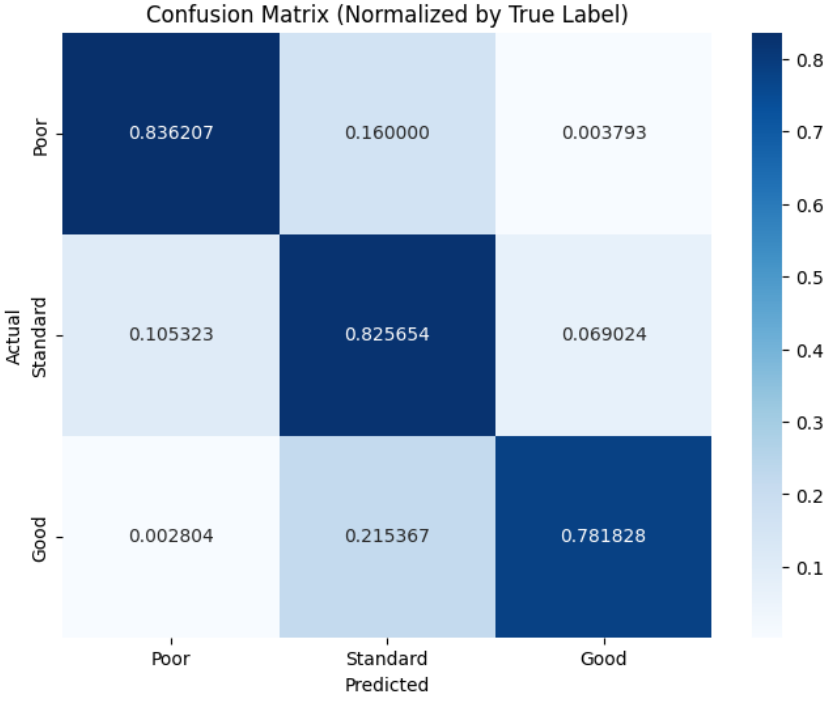

Figure 3: Confusion matrix for Random Forest Classifier

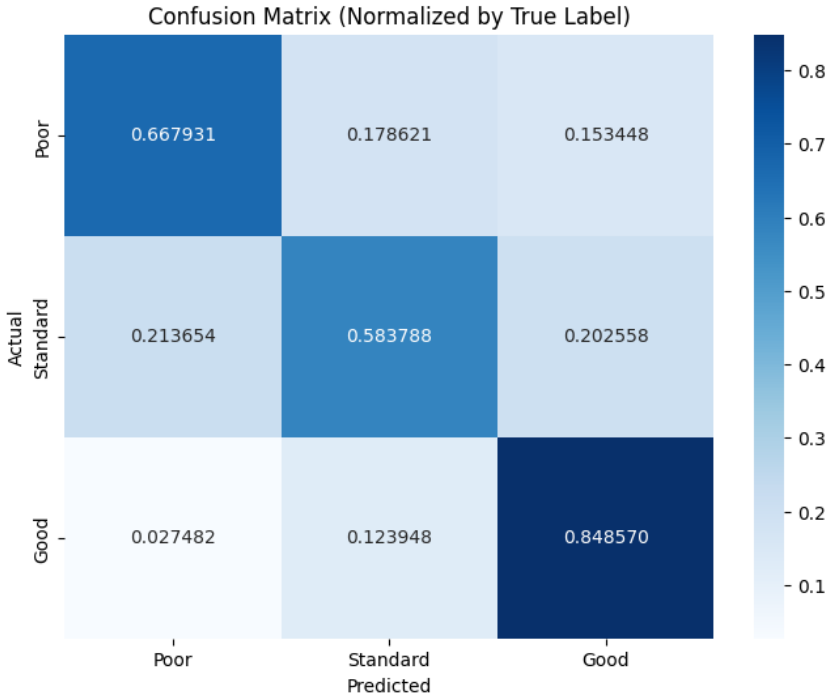

Figure 4: Confusion matrix for Logistic Regression

5 Preprocessing Code

[ ]:

```
# Download and import
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score,
    f1_score
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
```

[ ]:

```
# Download the dataset (add appropriate link for your dataset)
# !wget 'your_download_link_here'

# Load the CSV file into a DataFrame
df = pd.read_csv('/content/download?id=1U3J1DuXPd_7aCJNxY6AvPbd-JyqXfAS9')

# Data cleaning and feature transformation
split_payment_behavior = df['Payment_Behaviour'].str.split('_').str
df.rename(columns={'Credit_Score': 'Label'}, inplace=True)

# Map values to new features
df['Spending_Behavior'] = split_payment_behavior[0].map({'High': 1, 'Low': 0})
df['Payment_Behavior'] = split_payment_behavior[2].map({'Small': 0, 'Medium': 1, 'Large':
    2})
df['Payment_of_Min_Amount'] = df['Payment_of_Min_Amount'].map({'Yes': 1, 'No': 0})
df['Credit_Mix'] = df['Credit_Mix'].map({'Bad': -1, 'Standard': 0, 'Good': 1})
df['Label'] = df['Label'].map({'Poor': -1, 'Standard': 0, 'Good': 1})

# Display the first few rows of the DataFrame
```

```
df.head()

# Calculate correlation between label and features
df_numeric = df.select_dtypes(include=['float64', 'int64'])
label_features_correlation_matrix = df_numeric.corr()
abs_label_features_correlation_df = label_features_correlation_matrix['Label'].map(abs).
    sort_values()

# Select important features with a correlation greater than 0.2
important_features = abs_label_features_correlation_df[abs_label_features_correlation_df >
    0.2]
columns = important_features.index.tolist()

# Filter the DataFrame to keep only important columns
df = df[columns]
df.head()

# Calculate correlation between features
df_numeric = df.select_dtypes(include=['float64', 'int64']).drop(columns=['Label'])
abs_correlation_matrix = df_numeric.corr().abs()

# Plot correlation heatmap
plt.figure(figsize=(15, 12))
sns.heatmap(abs_correlation_matrix, annot=True, fmt=".4f", cmap='coolwarm', square=True,
            cbar_kws={"shrink": .8}, linewidths=.5)
plt.title('Features Correlation Heatmap')
plt.show()

# Identify highly correlated feature pairs (correlation >= 0.7)
features_correlation = abs_correlation_matrix.unstack().sort_values(ascending=False).
    drop_duplicates()
features_correlation = features_correlation[(features_correlation != 1) & (
    features_correlation >= 0.7)]

# Display the highly correlated features
features_correlation

# Calculate label data distribution and plot category frequency
count = Counter(df['Label'])
print(count)
values = count.values()
categories = count.keys()

plt.bar(categories, values)
plt.title('Category Frequency')
plt.xlabel('Category')
plt.ylabel('Frequency')
plt.show()
```

6 Machine Learning Models Code

6.1 Random Forest Classifier

[ ]:

```
# Prepare data for RandomForestClassifier
df_m1 = df.drop(columns=['Monthly_Inhand_Salary']).reset_index(drop=True)
X_m1 = df_m1.drop(columns=['Label'])
y_m1 = df_m1['Label']
X_train_m1, X_test_m1, y_train_m1, y_test_m1 = train_test_split(X_m1, y_m1, test_size=0.1,
random_state=random_state, stratify=y_m1)
```

[ ]:

```
# Train RandomForestClassifier
rfc_model_gini = RandomForestClassifier(random_state=random_state, criterion='gini',
    n_estimators=100)
rfc_model_gini.fit(X_train_m1, y_train_m1)
y_pred_gini_m1 = rfc_model_gini.predict(X_test_m1)
```

```
rfc_model_entropy = RandomForestClassifier(random_state=random_state, criterion='entropy',
    n_estimators=100)
rfc_model_entropy.fit(X_train_m1, y_train_m1)
y_pred_entropy_m1 = rfc_model_entropy.predict(X_test_m1)

accuracy_gini_m1 = accuracy_score(y_test_m1, y_pred_gini_m1)
accuracy_entropy_m1 = accuracy_score(y_test_m1, y_pred_entropy_m1)

print('Accuracy of model with gini as loss function: ', accuracy_gini_m1)
print('Accuracy of model with entropy as loss function: ', accuracy_entropy_m1)
```

[ ]:

```
# RandomForestClassifier metrics
precision_m1 = precision_score(y_test_m1, y_pred_gini_m1, average='weighted')
recall_m1 = recall_score(y_test_m1, y_pred_gini_m1, average='weighted')
f1_m1 = f1_score(y_test_m1, y_pred_gini_m1, average='weighted')
cm_m1 = confusion_matrix(y_test_m1, y_pred_gini_m1, normalize='true')
print("Accuracy:", accuracy_gini_m1)
print("Precision:", precision_m1)
print("Recall:", recall_m1)
print("F1 Score:", f1_m1)

plt.figure(figsize=(8, 6))
sns.heatmap(cm_m1, annot=True, fmt='f', cmap='Blues')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion Matrix')
plt.show()
```

6.2 Logistic Regression with GridSearchCV

[ ]:

```
# Prepare data for LogisticRegression
df_m2 = df.drop(columns=['Monthly_Inhand_Salary']).reset_index(drop=True)
X_m2 = df_m2.drop(columns=['Label'])
y_m2 = df_m2['Label']
# Split into train and test
X_train_m2, X_test_m2, y_train_m2, y_test_m2 = train_test_split(
    X_m2, y_m2, test_size=0.1, stratify=y_m2, random_state=random_state
)
```

[ ]:

```
# Scale features using StandardScaler
scaler = StandardScaler()
X_train_m2_scaled = scaler.fit_transform(X_train_m2)
X_test_m2_scaled = scaler.transform(X_test_m2)

# Define the model and parameter grid for GridSearchCV
lr_model = LogisticRegression(max_iter=2000, class_weight='balanced')
param_grid = {
    'solver': ['saga', 'liblinear'],
    'C': [0.01, 0.1, 1, 10, 100],
    'penalty': ['l1', 'l2'],
}

# Set up GridSearchCV
grid_search = GridSearchCV(estimator=lr_model, param_grid=param_grid,
                           scoring='accuracy', cv=5, n_jobs=-1, verbose=1)

# Fit GridSearchCV
grid_search.fit(X_train_m2_scaled, y_train_m2)

# Get the best model
best_model = grid_search.best_estimator_

# Test the final model
y_pred_test_m2 = best_model.predict(X_test_m2_scaled)
```

[ ]:

```
# Compute metrics for test
accuracy_m2 = accuracy_score(y_test_m2, y_pred_test_m2)
precision_m2 = precision_score(y_test_m2, y_pred_test_m2, average='weighted')
recall_m2 = recall_score(y_test_m2, y_pred_test_m2, average='weighted')
f1_m2 = f1_score(y_test_m2, y_pred_test_m2, average='weighted')
cm_m2 = confusion_matrix(y_test_m2, y_pred_test_m2, normalize='true')

# Print the metrics for final test
print("Accuracy:", accuracy_m2)
print("Precision:", precision_m2)
print("Recall:", recall_m2)
print("F1 Score:", f1_m2)

# Plot the average confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm_m2, annot=True, fmt='f', cmap='Blues')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion Matrix')
plt.show()
```