

Fully Paid Loan Analysis – Technical Report

Ethan Ericson

University of Arkansas

DASC 1223H – Data Science in Today's World

Dr. Schubert

05/02/2024

Technical Contributions

Overview of Techniques

Hypotheses and Techniques

For our study on fully paid loans and the characteristics of a good borrower for our client, Lending Club, I focused on evaluating the following financial indicator hypotheses:

1. Annual Income is an indicator of loan status.
2. DTI is an indicator of loan status.
3. Employment length is an indicator of loan status.

I investigated these hypotheses through several technical processes, listed below:

1. Creating functions
2. Creating figures with Histplot subplots
3. Creating Boxplots of subplot means.

Using these techniques, I was able to contribute valuable findings to my team and gain interesting insights into the nature of fully paid loans at Lending Club that could be used to enhance their Marketing and Revolving Lines of Credit departments' operations.

Exclusion of Purpose

At the beginning of this project, I had initially planned to evaluate the influence of the "purpose" variable on predicting loan status, but as our team compiled our findings, I found that my investigation of purpose took valuable time away from the explanation of more valuable findings regarding financial indicators; thus, I cut my purpose EDA out of our final product, but kept it in my Jupyter Notebook to serve as documentation of my analytical process.

Methods

All of my techniques were accomplished using Python and the packages: pandas, matplotlib, seaborn, and numpy. I utilized the IDE of Visual Studio to compile my code.

Technical Steps

Creating functions

After performing research on the financial sector, acquiring the dataset from Dr. LaBarr, and performing some surface level cleaning and factorization on the dataset, I was prepared to begin exploring the data using Python. To streamline my development process, I decided to create several functions that I could call to do the majority of my EDA for me.

I first set up specific cleaning functions that I could apply on subsets of the dataset. As a whole, I had already cleaned the majority of the data from duplicate values, null values, and duplicates, but because of the inclusion of variables that referred to either joint or individual applicants, there were many more null values that I had trouble removing at a large scale. To combat this problem, I made a clean feature and groupers function that copied the whole dataset, subset it via the feature and groupers included in the function, got rid of any null values in these variable, checked each of these variables for null values, and then returned the new cleaned subset of the dataset.

Figure 1

Clean Feature and Groupers code.

```

# Pre-Clean Data
def clean_feature_and_groupers(feature, purpose, status):

    # Take out null values from features and groups
    feature_df = cleaned_df.dropna(subset= [feature, purpose, status]).copy()

    # Check
    print(f"Nulls in {feature}: " + str(feature_df[feature].isnull().sum()))
    print(f"Nulls in {purpose}: " + str(feature_df[purpose].isnull().sum()))
    print(f"Nulls in {status}: " + str(feature_df[status].isnull().sum()))

    return(feature_df)

```

Note. This function generates a new cleaned subset of the cleaned dataset.

I also constructed a function that removed outliers from a distribution using interquartile range calculations.

Figure 2

Remove Outliers code.

```

# Remove Outliers function
def remove_outliers(data):
    Q1 = np.percentile(data, 25)
    Q3 = np.percentile(data, 75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    return data[(data >= lower_bound) & (data <= upper_bound)]

```

✓ 0.0s

Note. This function gets rid of outliers from a distribution.

This function was necessary to account for the presence of very high annual incomes that I had identified when looking at the dataset in excel. These large annual incomes heavily skewed the income distributions, so I trimmed them out using this function. As there were only a few of these individuals with very high incomes, I do not believe that their exclusion significantly impacted the validity of my results.

Having created two useful cleaning functions, I next moved onto making functions that would help me to display important characteristics of my analysis and generate the visualizations I planned to use in our team's final presentation. The first of this collection of analysis functions was the calculate total individuals method:

Figure 3

Calculate Total Individuals code

```
def calculate_total_individuals(feature, feature_df, statuses, purposes, trim):

    # Check if purposes is null
    if(purposes == "null"):

        # Total Individuals across all subsets
        total_individuals = 0

        # Use a for loop to go through each element of statuses from earlier
        for status in statuses:

            # For each of the statuses, filter the cleaned income and create a subset dataframe to plot
            subset_df = feature_df[(feature_df['factorized_status'] == status)].copy()

            # Make sure we don't have any empty subsets that we try to graph
            if(len(subset_df) > 0):

                if(trim):
                    # Remove outliers from feature data
                    subset_df[feature] = remove_outliers(subset_df[feature])

                # Calculating the total amount of individuals across all subsets
                total_individuals += len(subset_df[feature])

        return total_individuals

    # Else generate visualizations with purpose
    else:

        # Total Individuals across all subsets
        total_individuals = 0

        # Use a for loop to go through each element of statuses from earlier and purposes
        for purpose in purposes:
            for status in statuses:

                # For each of the statuses, filter the cleaned income and create a subset dataframe to plot
                subset_df = feature_df[(feature_df['factorized_status'] == status) & (feature_df["purpose"] == purpose)].copy()

                # Make sure we don't have any empty subsets that we try to graph
                if(len(subset_df) > 0):

                    if(trim):
                        # Remove outliers from feature data
                        subset_df[feature] = remove_outliers(subset_df[feature])

                    # Calculating the total amount of individuals across all subsets
                    total_individuals += len(subset_df[feature])

        return total_individuals
```

✓ 0.0s

Note. This function calculates the total individuals in a subset of the data.

This function creates all possible subsets of the feature data frame returned by the clean feature and groupers function and counts the total number of individuals within each subset before adding them all up. This function can be used either for both statuses and purposes as groupers, or just statuses. It can also trim outliers, or not trim outliers, depending on user preference. This function helped me create the “% of total borrowers” measure that was included in my final visualizations.

To create these final visualizations, I assembled a very customizable feature visualization function that would generate a figure with subplots of your target variable grouped by status or status and purpose. I have excluded the code for the purpose option of this function as it was too much to capture in a single screenshot, but it is viewable in the Jupyter Notebook I have attached alongside this report.

Figure 4

Feature Visualization code

```

def feature_visualization(feature, feature_df, statuses, purposes, total_individuals, bin_max_x, mean_lower, mean_upper, trim):

    #Check if purposes not given
    if(purposes == "null"):

        # Means List
        mean_feature_values = []

        # Total Individuals within each subset
        subset_individuals = 0

        # Number of Bins
        num_bins = 10

        # Determine the number of plots needed based on the number of statuses
        num_plots = len(statuses)

        # Calculate the number of rows and columns needed for the subplots
        num_cols = 2
        num_rows = int(np.ceil(num_plots / num_cols))

        # Get figure reference and axes for indexing
        fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 5*num_rows))

        # Adjust spacing
        plt.subplots_adjust(wspace=0.5, hspace=0.5)

        # Set plot index (iterator)
        plot_index = 0

        # Use a for loop to go through each element of statuses from earlier and purposes
        for status in statuses:

            # For each of the statuses, filter the cleaned feature and create a subset dataframe to plot
            subset_df = feature_df[(feature_df['factorized_status'] == status)].copy()

            # Make sure we don't have any empty subsets that we try to graph
            if(len(subset_df) > 0):

                if(trim == True):

                    # Remove outliers from income data
                    subset_df[feature] = remove_outliers(subset_df[feature])

                # Calculating the amount of individuals within each subset
                subset_individuals = len(subset_df[feature])

                # Calculate mean feature value
                mean_feature_value = subset_df[feature].mean()

                # Add to a list of mean feature values to graph distribution
                mean_feature_values.append(mean_feature_value)

                # Create a histogram for the distribution of feature values (np.linspace creates evenly spaced bin edges) (ax argument is the location in the subplot)
                plot = sns.histplot(data = subset_df, x = feature, bins = np.linspace(0, bin_max_x, num_bins), ax = axes[plot_index // num_cols, plot_index % num_cols])

                # Rotate x-axis labels and right justify
                plot.tick_params(axis='x', labelrotation=45)

                # Labels and Aesthetics
                axes[plot_index // num_cols, plot_index % num_cols].set_title(f"Loan Status: {status}")
                axes[plot_index // num_cols, plot_index % num_cols].set_xlabel(feature)
                axes[plot_index // num_cols, plot_index % num_cols].set_ylabel("Frequency")

                # Annotate histplot with percentage of individuals
                plot.text(plot.get_xlim()[1]*0.99, plot.get_ylim()[1]*0.8, f"% of Total Borrowers: {100 * (subset_individuals / total_individuals):.2f}%", ha = "right")

                # Annotate histplot with mean income
                plot.axvline(mean_feature_value, color='r', linestyle='--', linewidth=2, label=f'Mean {feature}: {mean_feature_value:.2f}')
                plot.legend(fontsize = "small")

                plot_index += 1

            else:
                print(f"This subset is empty: \nStatus: {status}\n")

        # Plot the boxplot distribution
        sns.boxplot(mean_feature_values)
        plt.ylabel('Mean Values')
        plt.title(f'Mean {feature} Distribution')
        plt.ylim(mean_lower, mean_upper)

        plt.show()

    # Else generate visualizations with purpose
    else:

```

Note. This function generates a figure with subplots for each grouping of a target feature.

This function works by taking in a feature, its corresponding cleaned data frame produced by my cleaning function, a list of statuses to group by, a list of purposes (or null keyword) to group by, the total individuals captured by the feature data frame, the length of the x axis, the upper and lower bounds of the boxplot of the subplot means, and a Boolean signifying if the user wants their data trimmed or not and using these arguments to create a series of feature distributions in subplots with annotations for mean and total percentage of borrowers represented by that subplot.

A notable quirk of this function is that you must provide the same arguments within the calculate total individuals function that you use to generate the total individual's argument if you want your percentages to be accurate.

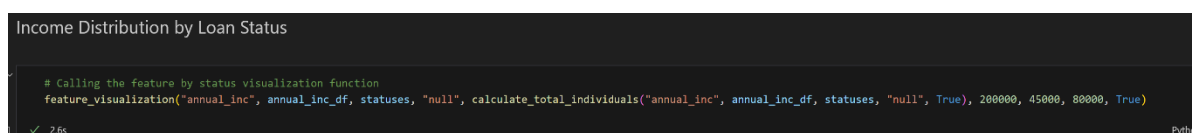
Utilizing this function made creating all my visualizations very easy and increased the efficiency of my work on this project greatly. As you will see in the following figures, I am able to create an entire formatted figure with a single line of code, much preferable to copying and pasting the behemoth Figure 4 repeatedly.

Creating figures with Histplot subplots and Boxplots of subplot means

After constructing all the relevant functions I required for my analysis, I began visualizing important findings regarding financial indicators. I generated three figures with six subplots each – five histplots and one boxplot – that corresponded to the three financial indicator features I was responsible for analyzing. I performed this operation using my feature visualization function as depicted in Figure 5.

Figure 5

Calling the feature visualization function



```

Income Distribution by Loan Status

# Calling the feature by status visualization function
feature_visualization("annual_inc", annual_inc_df, statuses, "null", calculate_total_individuals("annual_inc", annual_inc_df, statuses, "null", True), 200000, 45000, 80000, True)
✓ 2.6s
Python

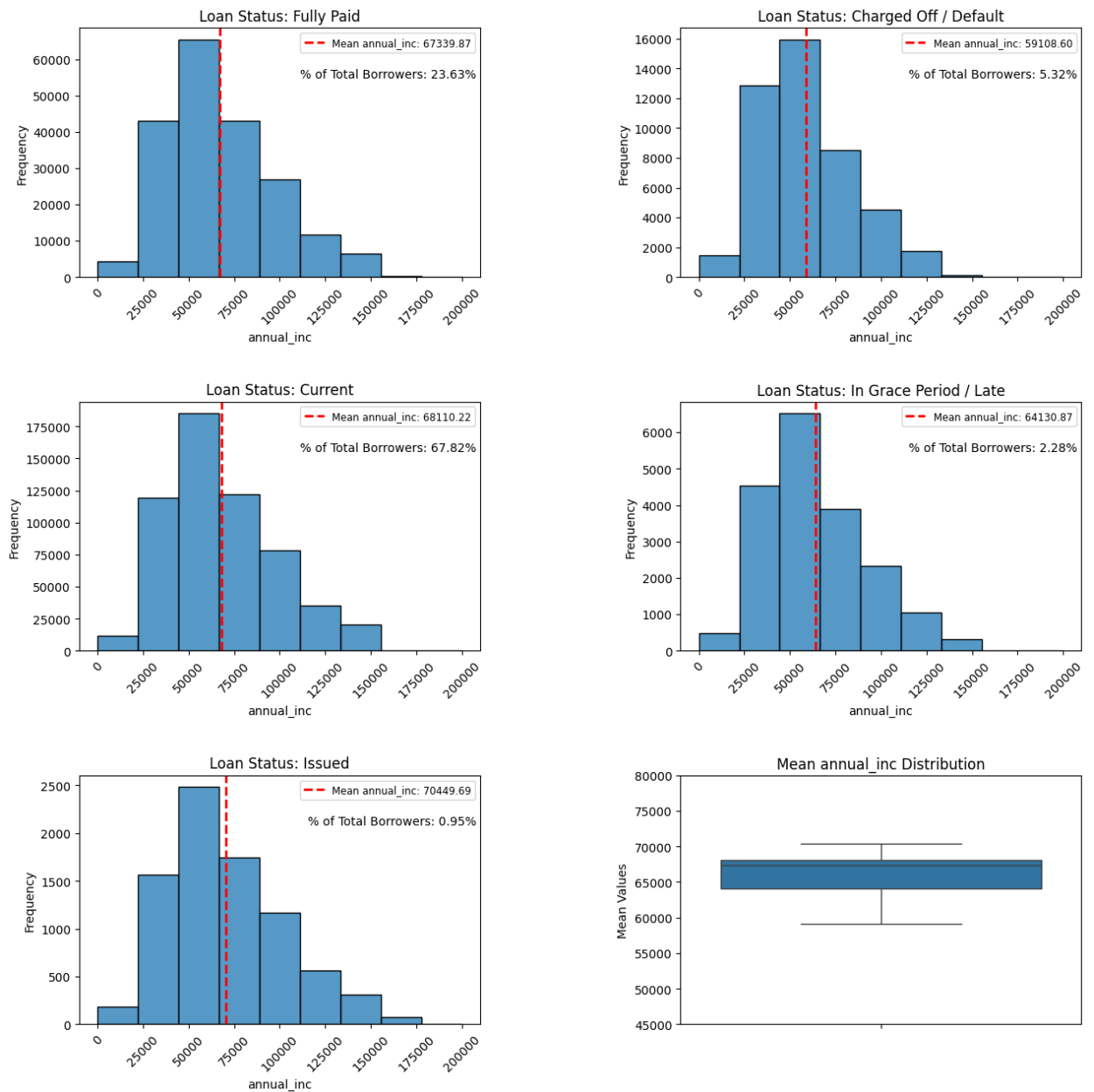
```


Note. This code demonstrates the ability to use my feature visualization function to create custom figures.

By inputting the relevant feature: annual income, and its corresponding data frame, along with several other customization arguments, I was able to generate the following figure:

Figure 6

Income Distribution by Loan Status



Note. This figure proves the effectiveness of my feature visualization function.

I repeated this process of figure generation for my other two features of interest, debt to income (DTI) and employment length. By comparing the subplots within each figure, and by looking at the range of the boxplot of subplots means, I was able to contribute several important findings regarding borrower financial indicators to my team's study.

Value of Contributions

My techniques in function engineering and figure creation advanced my group's understanding of the importance of annual income, DTI, and employment length on predicting a borrower's loan status. Through my analysis, I validated two of the three hypotheses I tested:

1. Annual Income is an indicator of loan status.
2. DTI is an indicator of loan status.

And I invalidated the hypothesis that employment length is an indicator of loan status.

Without the functions I created, the visualization derived from this hypothesis testing would have required many more lines of code to write, and the result would likely have been less accurate. Additionally, by choosing to portray the features I analyzed as distributions and compare their means, I simplified my EDA and avoided the pitfall of overcomplicating my findings to the point where they were difficult to interpret. Overall, I contributed value to my team's analysis by providing efficient and accurate analysis of financial indicators through the implementation of my technical processes.

Bibliography

Lending Club. (2024a). LendingClub_Data LendingClub_Data.csv [[File Acces](#)], Dr. Aric LaBarr. April 15, 2024.

Lending Club. (2024b). LendingClub Data Dictionary [[File Access](#)]. Dr. Aric LaBarr. April 15, 2024.

All images, statistics, metrics, models, graphs, analysis, insights, and calculations used in this research are based and created according to Lending club (2024a) data, the information on this research is for academic purposes only.