University Cover Page

Introduction

There has been a growing demand for small loans from financial companies in recent years. This is because more and more people are finding themselves needing extra money to cover unexpected expenses or make ends meet (AREAS, 2018). While banks and other traditional

lenders have typically been the go-to source for loans, they are often unwilling to provide loans for smaller amounts. This is where financial companies specializing in small loans can come in handy.

Several financial companies provide small loans to consumers in need. Some of these companies include CashNetUSA, Prosper, etc. Each of these companies has different loan terms and requirements, so it is essential to do your research before you decide which one to use.

The current economic environment is favorable for companies like CashNetUSA. With unemployment remaining relatively low and consumer confidence high, more people are willing to take out loans for small purchases or emergencies (Morduch & Schneider, 2017). This has resulted in increased demand for the company's products and services. The company has an automated system that makes getting approved for a loan easy. The system checks your credit score and income to see if you qualify for a loan. If you do, you can get approved for a loan in as little as minutes.

Significant changes in some of the businesses providing financial services have been seen (Farquhar & Meidan, 2017). Prosperity comes in the category. In response to the new regulatory environment, Prosper began offering personal loans instead of peer-to-peer loans. This allowed them to skirt some of the new regulations while still providing financing to borrowers. However, this shift meant that Prosper competed with traditional banks and other online lenders (Freedman & Jin, 2008).

The primary problems to be addressed are those related to the manual loan application process: skills shortages on the loan team, longer loan approval times, and increased potential operational and control risk. Other issues that may need to be considered include gender and marital status factors potentially influencing the approval decision, the number of dependents an applicant has, and their education and income levels.

In order to address these problems, it is essential to streamline the loan application process so that it is quicker and easier for both applicants and loan officers. It is also essential to consider other factors influencing the approval decision. Division of the high-risk and lower-risk groups is one of the main factors to be considered. For example, married couples are generally seen as more stable and responsible than single individuals, making them another low-risk group. On the other hand, people with multiple dependents may be viewed as high-risk because they have a greater financial responsibility. As such, lenders need to consider all of these factors when making lending decisions.

If an automated system were implemented to streamline the loan application process, it would positively impact both applicants and loan officers. The process would be quicker and easier for both groups, and the potential for errors would be significantly reduced. This would ultimately lead to more successful loan approvals and a better overall experience for everyone involved. The risk of missing critical information or making errors would be significantly reduced. However, If the current state of the loan application process remains unchanged, it will continue

to be a lengthy and complicated process. Suppose the company does not take action to improve its loan approval process. In that case, it risks losing customers to competitors, missing out on revenue growth opportunities, and incurring operational and control risks (Berry & Parasuraman, 1997). The current manual process is time-consuming and requires skilled staff, which may be in short supply. A well-functioning loan approval process is critical for any financial institution. A lengthy or complicated process can dissuade potential borrowers from applying for a loan, resulting in lost business opportunities (Dinh & Kleimeier, 2007).

In order to streamline the loan application process, several different human resources would be needed. First and foremost, loan officers would need to be responsible for assessing applications and making decisions. They would need to have access to all of the necessary information in one place and be able to assess each application quickly. Additionally, there would need to be customer service representatives who could help applicants with any questions or concerns. Finally, there would need to be IT staff who could develop and maintain the automated system itself. Having all of these different human resources available makes the loan application process much more streamlined and efficient.

Approach

Loan companies have several data, including comprehensive information on loan applicants. Data is an essential part of any enterprise or business. Collecting, processing, and analyzing data flow in a timely and accurate manner is critical to discover valuable information for decision-making (Bose, 2009). Data volume can be large, making information handling difficult and time-consuming. Python language is used with the Jupyter Notebook tool for loan data analytics (Andrews et al., 2014).

Python is a popular programming language in scientific computing because it has many data-oriented feature packages that can speed up and simplify data processing, thus saving time (McKinney, 2010). It has many completely free libraries that are open to the public. That critical factor makes Python essential for data analysis and data science. Jupyter Notebooks is one of the leading open-source tools for developing and managing data analytics (Mendez et al., 2019). It produces documents (notebooks) that combine inputs (code) and outputs into a single file. This single document approach enables users to develop, visualize the results and add information, charts, and formulas that make work more understandable, repeatable, and shareable. Jupyter notebooks support more than 40 programming languages, with a significant focus on Python (Nagar, 2018). Since it is a free and open-source tool, anyone can use it freely for their data science projects.

Our code uses 3 libraries: NumPy, Pandas and Tabula. NumPy includes features of mathematical operations like average, mode, the sum of array and functions like sin, cos, and least integer. Pandas library can read data from excel and CSV files and make a data frame (Wade, 2020), while Tabula library can make a data frame from PDF files (Corrêa & Zander, 2017). After getting the whole data frame, we can perform various analyses and operations on data with the help of the Pandas library.

While reading data from PDF, some data points are changed (it became a decimal number from an integer value). Also, there are 14 pages in PDF; each page contains a data table. Except for the 1st page, the 1st row of each page's data table became a column header. We need to make it a data row and replace the header with the actual column header.

Out[104]:

	2219	1	1.1	3	1.2	0	8750	4996	130	360	1.3	3.1	Y
0	2223	1	1	0	1	0	4310	0	130	360	0	2	Y
1	2224	1	0	0	1	0	3069	0	71	480	1	1	N

For example, in the above picture, 1 becomes 1.1, and 3 becomes 3.1. Also, the row with Loan ID 2219 itself is a row, but the library reads as a column header.

'df.columns' reads the column's heading (titles) from data frame 'df' and stores it in an array. By this, we read columns of 1st page's table from PDF. This should be the actual header in all data frames from all pages.

```
df2 = pd.concat([df2,df1[0]],ignore_index = True)
for i in range(1,len(df1)):
    df1[i].loc[-1] = df1[i].columns
    df1[i].index = df1[i].index + 1
    df1[i].sort_index(inplace = True)
    df1[i].columns = header
    df2 = pd.concat([df2,df1[i]],ignore_index = True)
```

'df=pd.concat([df1, df2], ignore_index=True)' syntax combines data frame df1 and df2 and make one data frame df. Here we are ignoring the index of df2, which means the index of 1st row of df2 will be the index of the last row of df1 + 1. After that, we have to combine the data frame of 14 pages so we will take the help of for loop running from 1 to the length of data frame 1. In each loop take out the header and make it the first row.

```
for i in range(len(header)-1):
    df2[header[i]] = pd.to_numeric(df2[header[i]])
    df2[header[i]] = df2[header[i]].apply(np.floor)
    df2[header[i]] = df2[header[i]].astype(int)

df2 = df2.sort_values(by=['Loan_ID'], ignore_index = True)
```

After combining all data frames, we have to clean the loan data. We get our whole data together but still, some values are wrong or their data types are wrong (Data type float or string instead of int). First, we make all numbers that are in string format into numeric data type using

‘pd.to_numeric(data)’. Then there are certain columns that should have only integer values but while reading it came out to float values, so using the floor function from the Numpy library we convert them into integers. Finally, we will sort all rows according to Loan ID in ascending using ‘df.sort_values(by=[‘Loan_ID’])’. Now we got rid of the correction of duplicates, missing values, outliers etc.

Now, we move towards basic Exploratory Data Analysis (EDA) of the data. To see the numbers of rows and columns in data, ‘df.shape’ is used. We have 645 rows and 13 columns.

```
1  df2.shape
```

```
(645, 13)
```

df.info() gives basic information like column names with their data type, memory usage etc.

```
1  df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 645 entries, 0 to 644
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            645 non-null    int32
 1   Gender             645 non-null    int32
 2   Married            645 non-null    int32
 3   Dependents         645 non-null    int32
 4   Graduate           645 non-null    int32
 5   Self_Employed      645 non-null    int32
 6   ApplicantIncome    645 non-null    int32
 7   CoapplicantIncome  645 non-null    int32
 8   LoanAmount         645 non-null    int32
 9   Loan_Amount_Term   645 non-null    int32
 10  Credit_History     645 non-null    int32
 11  Property_Area      645 non-null    int32
 12  Loan_Status        645 non-null    object
dtypes: int32(12), object(1)
memory usage: 35.4+ KB
```

```
1  df2.describe()
```

	Loan_ID	Gender	Married	Dependents	Graduate	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
count	645.000000	645.000000	645.000000	645.000000	645.000000	645.000000	645.000000	645.000000	645.000000	645.000000
mean	1996.812403	1.186047	0.648062	0.731783	0.784496	0.130233	5365.589147	1608.558140	148.150388	334.837209
std	553.465890	0.389446	0.477945	1.002015	0.411491	0.336820	6021.940563	2867.005153	86.122299	63.206969
min	1002.000000	1.000000	0.000000	0.000000	0.000000	0.000000	150.000000	0.000000	9.000000	12.000000
25%	1562.000000	1.000000	0.000000	0.000000	1.000000	0.000000	2875.000000	0.000000	101.000000	333.000000
50%	1955.000000	1.000000	1.000000	0.000000	1.000000	0.000000	3813.000000	1229.000000	128.000000	360.000000
75%	2448.000000	1.000000	1.000000	1.000000	1.000000	0.000000	5726.000000	2306.000000	168.000000	360.000000
max	2990.000000	2.000000	1.000000	3.000000	1.000000	1.000000	81000.000000	41667.000000	700.000000	480.000000

To analyze the statistical details of each column ‘df.describe()’ is useful. It gives total count, mean, std deviation, max, min and median of 25%, 50% and 75% for each column.

‘df.Column_Name.value_counts()’ gives the total count of each entry in the column named ‘Column_Name’ in the data frame ‘df’. Here, we can see that from the Loan_Status column only 444 applicants’ loans are approved and 201 applicants’ loan is rejected.

```
1  df2.Loan_Status.value_counts()
```

```
Y    444
N    201
Name: Loan_Status, dtype: int64
```

‘df=df1[df1['Column_Name']==key]’ this syntax makes new data frame df from old data frame df1 such that it takes only rows with the entry ‘key’ in a column of name ‘Column_Name’. For example, the self_df data frame contains only those rows with the value ‘1’ in the ‘Self_Employed’ column. ‘df[‘Column_Name’].mean()’ gives the mean value of that column.

```
11  self_df = df2[df2['Self_Employed'] == 1]
12  avg_self_income = self_df['ApplicantIncome'].mean()
```

With using above features, we can calculate below statistical data:

```
The percentage of female applicants that had their loan approved:  66.67
The average income of all applicants:  5365.59
The average income of all applicants that are self-employed:  7284.29
The average income of all applicants that are not self-employed:  5078.3
The average income of all graduate applicants:  5809.92
The percentage of graduate applicants that had their loan status approved:  71.15
```

Recommendations

Research and development of an automated system can be used to streamline the loan application process. This system should be designed to make the process quicker and easier for applicants and loan officers, reducing the potential for errors.

One way to streamline the process is to create a centralized database where all required documentation can be stored electronically (Figorilli et al., 2018). This would allow loan officers to quickly access an applicant's file and eliminate the need for paper file folders. The database should be secure and only accessible by authorized personnel (Alruwaili, 2012).

Finally, we need to create a set of standard operating procedures (SOPs) that everyone involved in the loan application process must follow. These SOPs should be designed to minimize errors and ensure that all steps in the process are completed on time (Manghani,

2011). By following a set of standardized procedures, we can help ensure that each loan application is processed efficiently and correctly.

The automated system should be tested internally to ensure it is functioning correctly. Once satisfied, the system can be launched live for all users. There are a few ways to test the system:

- Ø Use test data. This is data that is not real but simulates actual user data. This can be used to see if the system can correctly handle different input and output types (Maurer, 1990).

- Ø Use beta testers. These are actual users who try out the system and give feedback on its performance. Beta testing can help identify issues using the system in a real-world setting.

Based on the data provided, it is possible to develop a predictive model that could streamline the loan application process. The model would consider various factors, including the applicant's credit score, employment history, and income. By considering these factors, the model would be able to generate a prediction of whether or not an applicant is likely to be approved for a loan. This would allow loan officers to focus their time on the most likely approved applications, ultimately leading to more successful loans. In order to further improve the accuracy of the model, additional data points could be collected and used. For example, information on the applicant's debts and assets could be considered. Additionally, data on previous loan applications could be used to understand an applicant's creditworthiness better, thus significantly improving the loan process.

There are a variety of techniques that can be used to evaluate the precision of a predictive model. One standard method is cross-validation, which involves partitioning the data into training and test sets (Refaeilzadeh et al., 2009). The model is then fit on the training set and evaluated on the test set. This process can be repeated multiple times to get an estimate of the model's generalization error. Additionally, various metrics can be used to assess the performance of a predictive model (Madeyski & Jureczko, 2015). For example, accuracy, precision, recall, and F1 score are all commonly used measures. In general, the goal is to maximize the model's predictive power while minimizing overfitting.

A variety of libraries, tools, and objective functions can be used to optimize a predictive model. For example, scikit-learn is a popular Python library for machine learning that includes a wide range of algorithms that can be used for predictive modeling (Kramer, 2016). Additionally, many commercial software packages offer data mining and machine learning tools. These tools typically include a variety of optimization methods that can be used to improve the performance of a predictive model. Finally, various objective functions can be used to measure the success of a predictive model. For example, one typical objective function is log-loss, which measures the negative log-likelihood of predicting the correct class label. Minimizing log loss makes it possible to improve a predictive model's accuracy.

Conclusions

Understanding the complex and ever-changing contours of the business environment of companies providing financial services is illuminating. Companies providing fast and easy-to-access services can adapt to the environment. The study of Zappy Financial Services highlights the need to develop a program to streamline the lending process, keeping in mind various other factors such as labor shortage, gender, etc. In the case discussed above, self-learning software using python was made as a part of the solution. The generation of the software required in-depth analysis and knowledge of the data and its effects on the process of the company.

In conclusion, an automated loan application system would positively impact the business. The process would be quicker and easier for various groups, and the potential for errors would be significantly reduced. This would ultimately lead to more successful loan approvals and a better overall experience for everyone involved.

References

Alruwaili, A. H. (2012). Security in database systems. *Global Journal of Computer Science and Technology*, *12*(17), 9–13.

Andrews, R. W., Stein, J. S., Hansen, C., & Riley, D. (2014). Introduction to the open source PV LIB for python Photovoltaic system modelling package. *2014 IEEE 40th Photovoltaic Specialist Conference (PVSC)*, 0170–0174.

AREAS, B. (2018). Financial analysis. *Growth*, *30*, 10.

Berry, L. L., & Parasuraman, A. (1997). Listening to the customer–the concept of a service-quality information system. *MIT Sloan Management Review*, *38*(3), 65.

Bose, R. (2009). Advanced analytics: Opportunities and challenges. *Industrial Management & Data Systems*.

Corrêa, A. S., & Zander, P.-O. (2017). Unleashing tabular content to open data: A survey on pdf table extraction methods and tools. *Proceedings of the 18th Annual International Conference on Digital Government Research*, 54–63.

Dinh, T. H. T., & Kleimeier, S. (2007). A credit scoring model for Vietnam's retail banking market. *International Review of Financial Analysis*, *16*(5), 471–495.

Farquhar, J., & Meidan, A. (2017). *Marketing financial services*. Bloomsbury Publishing.

Figorilli, S., Antonucci, F., Costa, C., Pallottino, F., Raso, L., Castiglione, M., Pinci, E., Del Vecchio, D., Colle, G., & Proto, A. R. (2018). A blockchain implementation prototype for the electronic open source traceability of wood along the whole supply chain. *Sensors*, *18*(9), 3133.

Freedman, S., & Jin, G. Z. (2008). *Do social networks solve information problems for peer-to-peer lending? Evidence from Prosper. com*.

Kramer, O. (2016). Scikit-learn. In *Machine learning for evolution strategies* (pp. 45–53). Springer.

Madeyski, L., & Jureczko, M. (2015). Which process metrics can significantly improve defect prediction models? An empirical study. *Software Quality Journal*, *23*(3), 393–422.

Manghani, K. (2011). Quality assurance: Importance of systems and standard operating procedures. *Perspectives in Clinical Research*, *2*(1), 34.

Maurer, P. M. (1990). Generating test data with enhanced context-free grammars. *Ieee Software*, *7*(4), 50–55.

McKinney, W. (2010). Data structures for statistical computing in python. *Proceedings of the 9th Python in Science Conference*, *445*(1), 51–56.

Mendez, K. M., Pritchard, L., Reinke, S. N., & Broadhurst, D. I. (2019). Toward collaborative open data science in metabolomics using Jupyter Notebooks and cloud computing. *Metabolomics*, *15*(10), 1–16.

Morduch, J., & Schneider, R. (2017). *The financial diaries: How American families cope in a world of uncertainty*. Princeton University Press.

Nagar, S. (2018). IPython. In *Introduction to Python for Engineers and Scientists* (pp. 31–45). Springer.

Refaeilzadeh, P., Tang, L., & Liu, H. (2009). Cross-validation. *Encyclopedia of Database Systems*, *5*, 532–538.

Wade, R. (2020). Reading CSV Files. In *Advanced Analytics in Power BI with R and Python* (pp. 151–175). Springer.

Appendix 1 – Code

```
import pandas as pd
import tabula as tb
import numpy as np

df1 = tb.read_pdf('PDA - Loans_Database_Table.pdf', pages = 'all')  # reading pdf dataframe
df2 = pd.read_excel('PDA - Zappy Loan Data.xlsx') # reading excel dataframe

df1[0]
```

```
'''
here we can see that during the reading of the pdf file 1st row of every page became a header
also their integer values became the float we need to clean this data frame made of only one
page of pdf and finally combine it to excel data frame
'''
df1[-1]

header = df1[0].columns        # actual header
header

df2 = pd.concat([df2,df1[0]],ignore_index = True)
for i in range(1,len(df1)):
        # here we first take out the header row from all pdf pages and append below that page
    df1[i].loc[-1] = df1[i].columns
    df1[i].index = df1[i].index + 1                  # shifting the index
    df1[i].sort_index(inplace = True)
    df1[i].columns = header                          # make actual header in all pdf pages
        df2 = pd.concat([df2,df1[i]],ignore_index = True)   #attach this data frame in excel file

for i in range(len(header)-1):
        # changing all numeric string into a numerical value
    df2[header[i]] = pd.to_numeric(df2[header[i]])
        # correcting 1st row of every page pdf disturbed integer into float while reading pdf
    df2[header[i]] = df2[header[i]].apply(np.floor)
    df2[header[i]] = df2[header[i]].astype(int)             # making all values integer as before

df2 = df2.sort_values(by=['Loan_ID'], ignore_index = True) # sorting according to Loan ID
# saving as an excel file, this helps to analyze datatype of cells and whole data at once

df2.to_excel('TotalData.xlsx')

df2.head()

df2.shape

df2.info()

df2.describe()

df2.Property_Area.value_counts()     # type of values of property area and their frequency

df2.Loan_Status.value_counts()

df2.Dependents.value_counts()
```

```
#The percentage of female applicants that had their loan approved
female_df = df2[df2['Gender'] == 2]
fe_percent = 100*(female_df.Loan_Status.value_counts()['Y']/len(female_df))
print('The percentage of female applicants that had their loan approved: ', f'{fe_percent:.4}')

#The average income of all applicants
avg_income = df2['ApplicantIncome'].mean()
print('The average income of all applicants: ', f'{avg_income:.6}')

#The average income of all applicants that are self-employed
self_df = df2[df2['Self_Employed'] == 1]
avg_self_income = self_df['ApplicantIncome'].mean()
print('The average income of all applicants that are self-employed: ', f'{avg_self_income:.6}')

#The average income of all applicants that are not self-employed
nonself_df = df2[df2['Self_Employed'] == 0]
avg_nonself_income = nonself_df['ApplicantIncome'].mean()
print('The average income of all applicants that are not self-employed: ', f'{avg_nonself_income:.6}')

#The average income of all graduate applicants
graduate_df = df2[df2['Graduate'] == 1]
avg_grad_income = graduate_df['ApplicantIncome'].mean()
print('The average income of all graduate applicants: ', f'{avg_grad_income:.6}')

#The percentage of graduate applicants that had their loan status approved
grad_percent = 100*(graduate_df.Loan_Status.value_counts()['Y']/len(graduate_df))
print('The percentage of graduate applicants that had their loan status approved: ', f'{grad_percent:.4}')
```