Analysis of Non-Payment Probability for Canadian Households

Oscar Cueva Bravo

Centennial College

Business Analytics Capstone

David Parent

August 12, 2022

Table of Contents

Executive Summary

This analytics project was undertaken with the aim of creating a model that accurately predicts if a Canadian household will skip (or delay) a non-mortgage payment. The Survey of Financial Security 2019 (SFS-2019) from Statistics Canada was used to complete this project. Through its 100 variables, the SFS-2019 describes the financial standing of 100% of the Canadian population prior to the COVID-19 pandemic. As a secondary – longer-term goal, this project aims to find if and how the financial standing of Canadians changed after the economic shock caused by COVID-19.

Most of the biggest challenges faced during the completion of this project revolved around the models' data treatment. Data augmentation (SMOTENC) was implemented to solve the class imbalance found in the target variable. Among 42 models that included Logistic Regressions, Feature Selection Algorithms, Decision Tree Classifiers, and Neural Networks, the best predictor was a tree ensemble model built on SMOTENC data. At 96% and 85%, this AdaBoost Classifier achieved the highest AUC-ROC and Recall metrics, respectively.

As it was found through a profiling exercise performed using the best predictor, although income, debt, and net worth, were all found to be relevant individual factors to explaining if a payment is going to be skipped, it is the combination (and likely the interaction) of multiple factors what ultimately drives that probability. Both high and low-income households had some of the highest (and lowest) probabilities of skipping a payment. However, just as an example, households with relatively low bank account balances (compared to their debts) were among those with highest risk of skipping or delaying a payment.

Assessing the Probability of Non-Payment

The COVID-19 Pandemic

Businesses of all sorts face a very challenging task every day: assessing the probability of non-payment by current and potential customers. Although this already difficult task was added an extra layer of complexity by the economic uncertainty introduced by the COVID-19 pandemic, it also presented a golden opportunity to generate market and consumer insights under such exceptional circumstances. The COVID-19 pandemic is definitely a Black Swan event that generated valuable data to analyze consumer sentiment, consumption patterns, and consumers' ability to meet their financial responsibilities.

In addition to the toll that the COVID-19 pandemic took on the health of millions of Canadians and the global population, labor markets were severely hit by lockdowns (Deloitte, 2022), private consumption expenditure (PCE) shrank (Remes et al., 2021), and supply chains were heavily disrupted (Benigno et al., 2022). According to estimates of the World Bank (2022), advanced economies' Real GDP contracted 4.6% during 2020, and while the global GDP managed to jump back to an estimated 5.7% growth in 2021, as of April of 2022, 100% of the developed economies were experiencing inflation levels above their targets.

At a national level, different countries and sectors of their respective populations were impacted differently. For example, while China only saw a deceleration of its real GDP growth during 2020 (World Bank, 2022), the United States experienced one of its worst-ever GDP contractions during the second quarter of the same year (BEA, 2020) – albeit, the recession only lasted two months according to official figures (NBER, 2021). According to Wang (2022), Canada's GDP per capita contracted 1.3% on average between 2020 and the first quarter of 2022. Lower work intensity among males and reduced participation rates among seniors and young workers were leading factors behind Canada's GDP per capita contraction. These findings

are strong suggestions that not all Canadian households were hit equally by the unprecedented economic shock created by the COVID-19 pandemic.

Within a very short period of time, entire families were forced to change their habits. Companies instituted work-from-home as a regular practice and many workers lost their jobs (Citation). People were forced to reconsider going to the gym, the café, or dining out. Choices related to more significant spending changed too, either out of precaution or due to unexpected changes in purchasing power. Clearly, the pandemic introduced volatility to the Canadian labor and consumer markets, affecting different pockets of the population in unique ways.

Problem Statement: Skipping or Delaying a Non-Mortgage Payment

The present analytics project looks to answer the following questions:

- Will a household skip or delay a non-mortgage payment?
- What are the distinctive characteristics of the Canadian households with the highest and lowest probabilities of skipping or delaying non-mortgage payments?

As a longer-term objective, this project has the potential to be expanded in order to assess how the probability of skipping or delaying a payment during the pandemic changed among groups of interest. Specifically, an extension of this project would look to answer the following question:

- Did the pandemic affect the probability of skipping or delaying a non-mortgage payment among groups of interest? If so, how?

Objective: Maximizing the Probability of Correctly Classifying Positive Instances

The objective of this project is to create a baseline model that accurately and effectively predicts if a non-mortgage payment will be skipped or delayed by Canadians. As this is a classification project – skip/delay versus no-skip/delay – that looks to maximize the potential of correctly classifying positive instances, the highest-ranking measures of success for this baseline

model are AUC-ROC and Recall. Both measures quantify the probability of correctly classifying instances based on their true positive and false positive rates (Google Developers, 2022).

A longer-term and secondary objective of this project is to build an additional model that determines how the economic shock caused by the COVID-19 pandemic changed the probability that a non-mortgage payment will be skipped or delayed. Finding what were the driving factors of such changes would be imperative in the construction of future, complementary, models.

The Distribution of Income and Data Limitations

Data availability issues limited to the elaboration of the present. The initial goal of this project was to analyze the non-payment probabilities among Canadians during the COVID-19 pandemic. Unfortunately, as of July of 2022, it was impossible to find reliable data to complete such a project and it was decided that the best alternative was to construct a baseline 'pre-pandemic' model using the most recent, available, and reliable data. Upon the release of 2020-2021, and possibly 2022 data, an extension model would be constructed and the appropriate assessments would be then conducted. No additional limitations to the completion of this project were found.

The acute concentration of income and wealth among Canadians to led the assumption that the ultra-wealthy, and those found at the top of the distribution of income, behave differently. In other words, it has been assumed that ultra-wealthy and top-earning Canadians skip or delay payments for different reasons when compared to the rest of the population.

Data

Survey of Financial Security 2019

Suggested citation by Statistics Canada (2019):

"This analysis is based on Statistics Canada's Survey of Financial Security Public Use Microdata, 2019, which contains anonymous data collected in the Survey of Financial Security. All computations on these microdata were prepared by Oscar Cueva. The responsibility for the use and interpretation of these data is entirely that of the author."

The Survey of Financial Security (SFS) collects all the data necessary to accurately describe the wealth and overall financial standing of 100% of Canadian households. The SFS contains 10,422 records that represent the entirety of Canadian households; weights are provided in the dataset to make inferences about the true population. The SFS collects information about Canadian households' incomes, assets, liabilities, and many other relevant aspects, all needed to compute their net worth.

Exclusion of the Ultra-Wealthy-Household Records

As it was explained in a previous section of this report, it has been assumed that the wealthiest and top-earning households behave differently from the rest of the population and have different reasons to skip or delay a non-mortgage payment. Based on this premise, 15.18% of the records were excluded from the present analysis based on a multi-dimensional definition of wealth that includes households' net worth, assets, and debt. The record exclusion process will be discussed in further detail in a later section. No additional exclusions were necessary.

Data Dictionary: 100 Variables and 10,422 Records

The SFS 2019 contains 100 variables that describe households' net worth, assets, and debt. For convenience purposes, the variables were categorized into four large groups that include: Real Estate, Mortgage, Resources, and Household, and later further subcategorized into: Assets, Business, Credit & Credit Card, Debt, Family Composition, Funds, Geographic Location, ID, Income, Major income Earner, Mortgage, Net Worth, Payment, Real Estate, RRSP, Version

Date, and Weight. Although variable classification was arbitrary, it was not factored in the analysis process and did not influence its results in any shape or form.

The Real Estate category includes 16 explanatory variables. Refer to Figure 1.

Figure 1. Real Estate Variables

Variable	Description	Python - Data Type
PASRBUYG	Year property was purchased	int64
PASRCNMG	Number of 'other real estate properties' owned (inside Canada)	int64
PASRCON	Property is a condominium development	categorical
PASRCST	Purchase price of the property	int64
PASRDPO1	Down Payment - money in bank account	categorical
PASRDPO2	Down Payment - previous home	categorical
PASRDPO3	Down Payment - investments	categorical
PASRDPO4	Down Payment - borrowing	categorical
PASRDPO5	Down Payment - other source	categorical
PASRDWNG	Down payment in percentage terms	int64
PASRFNMG	Number of 'other real estate properties' (outside Canada)	int64
PASRRNTG	Part of the property is rented out	categorical
PDWTYP	Type of dwelling	categorical
PFTENUR	Principal residence ownership status	categorical
PWAPRVAL	Value of the principal residence	int64
PWASTRST	Real estate other than principal residence	int64

The Household category includes 16 explanatory variables. Refer to Figure 2.

Figure 2. Household Variables

Variable	Description	Python - Data Type
PAGEMIEG	Age of the major income earner in the family unit	int64
PEDUCMIE	Highest level of education of the major income earner	categorical
PFMTYPG	Derived variable, composition of family units	categorical
PFSZ	Number of members in the family unit, all ages	int64
PFSZ0017	Presence of persons ages 0 to 4	categorical
PFSZ1824	Presence of persons in the family of ages 18 to 24	categorical
PFSZ2544	Presence of persons in the family of ages 15 to 44	categorical
PFSZ4564	Presence of persons in the family of ages 45 to 64	categorical
PFSZ65UP	Presence of persons in the family of ages 64 and up	categorical
PLFCHRME	Usually works 30 hours or more per week	categorical
PLFFPTME	Worked either full time or part-time (in 2018)	categorical
PLFPDMEG	Is either family member a paid worker or self employed	categorical
PPVRES	Province of residence for the family unit	categorical
PREGION	Region of residence of the family unit	categorical
PRETIRME	Indicates if the major income earner has ever retired	categorical
PGDRMIE	Gender of the major incomer earner in the family unit	categorical

The Resources category includes 48 explanatory variables. Refer to Figure 3.

Figure 3. Resources Variables

Variable	Description	Python - Data Type
PATTCRC	Have credit cards	categorical
PATTCRLM	Total cedit limit on all credit cards owned	int64
PATTCRR	Haven been refused credit card	categorical
PATTCRU	Credit vard balances usually paid off every month	categorical
PATTDIF	Skipped card payment due to financial difficulties	categorical
PATTLCP	Line of credit balances usually paid off every month	categorical
PATTLCR	Lines of credit	categorical
PATTPAYD	Borrowed money through Pay Day Loans	categorical
PATTRSA	Withdrew RRSP funds to purchase annuities or RRIFs	categorical
PATTRSH	Withdrew RRSP funds through Home Buyer's Plan	categorical
PATTRSL	Withdrew RRSP funds through a Lifelong Learning Plan	categorical
PATTRSP	Have or had money in RRSP	categorical
PATTRSR	Withdrew RRSP funds for any reason other than those listed	categorical
PATTSTIN	Expected change in total income	categorical
PATTLMLC	Total credit limit on all lines of credit	int64
PBUSIND	Have a business	categorical
PEFATINC	After-tax income	int64
PEFGTR	Government transfers, federal & provincial	int64
PEFMJSIF	Major source of income for the family unit	categorical
PEFMTINC	Market income	int64
PFCRN	Number of credit cards used, grouped	int64
PFRSPST	Have or have had RRSPs and made withdrawals	categorical
PINHERT	Total value of inheritances received in 2016 constant dollars	int64
PNBEARG	Number of earners aged 15 or over in the family unit	categorical
PWAOTPEN	Other retirement funds	int64
PWARPPG	Value of all employer pension plans. Going concern basis	categorical
PWARPPT	Value of all employer pension plans. Termination basis	int64
PWARRIF	Registered retirement income funds (RRIFs)	int64
PWARRSPL	RRSP investments including locked in RRSP's	int64
PWASTBND	Bonds, non-registered	int64
PWASTDEP	Money in banks, non-registered	int64
PWASTMUI	Mutual funds, other investment, Income trusts, non-registered	int64
PWASTOIN	Other investments or financial assets	int64
PWASTONF	Other non-financial assets	int64
PWASTSTK	Stock and shares, non registered	int64
PWASTVHE	Vehicles	int64
PWATFS	Tax Free Savings Accounts (TFSA)	int64
PWATOTPG	Total assets including employer pension plans - On going concern basis	categorical
PWATOTPT	Total assets including employer pension plans - Termination basis	int64
PWBUSEQ	Equity value of businesses operated by the family unit	int64
PWDSLOAN	Debt value of student loans	int64
PWDSTCRD	Credit card and installment debt	int64
PWDSTLOC	Line-of-credit debt (home and other line of credit)	int64
PWDSTODB	Other debt (other loans and other money owed)	int64
PWDSTVHN	Debt on vehicles	int64
PWDTOTAL	Total of all debts for the family	int64
PWNETWPG	Net worth of the family unit (On going concern basis)	categorical
PWNETWPT	Net worth of the family unit (Termination basis)	int64

The Mortgage category includes 16 explanatory variables. Refer to Figure 4.

Figure 4. Mortgage Variables

Variable	Description	Python - Data Type
PAS1MRAG	Amount borrowed at time of refinancing	categorical
PAS1MRG1	Refinancing to renovate or make an additional property	categorical
PAS1MRG2	Other reasons for refinancing	categorical
PASR1MFA	Amount of additional mortgage payments	int64
PASR1MR	Mortgage renegotiated for reasons unrelated to home	categorical
PASRCURG	Current length of mortgage term	int64
PASRINT	Type of mortgage interest rate	categorical
PASRINTG	Current interest rate on mortgage	int64
PASRMOAG	Original amortization period for the mortgage	int64
PASRMPFG	Number of years to pay down remaining balance of mortgage	int64
PASRMRYG	Year mortgage up for renewal	int64
PASRSKP	Have skipped or delayed a mortgage payment	categorical
PEXMG1A	Amount of mortgage payments	int64
PEXMG1F	Frequency of mortgage payments	int64
PWDPRMOR	Mortgage on principal residence, fianl value	int64
PWDSTOMR	Mortgage on 'other real estate' in and outside of Canada	int64

PATTSKP was selected as the target variable. Refer to Figure 5.

Figure 5. Target Variable

Variable	Description	Python - Data Type
PATTSKP	Have skipped or delayed a non-mortgage payment	categorical

There are three identification variables, irrelevant for the completion of this project. Refer to Figure 6.

Figure 6. Identification Variables

Variable	Description
PEFAMID	Family unit: economic families and persons not in economic families
PWEIGHT	Survey weights - PUMF
VERDATE	Version Date

The SFS 2019 dataset contains 10,422 records, none of which include missing values. To learn about the survey methodology, accuracy, or how missing values were treated by the creators of the dataset, please consult the 2019 Survey of Financial Security: Public Use Microdata File User Guide from Statistics Canada (2019).

The Diversity Found in Census Data is a Challenge and an Opportunity

Utilizing Statistics Canada survey data to complete this capstone project certainly presented multiple benefits. Using real-life data introduced the opportunity to generate insights about real Canadians; however, it also introduced real-life challenges and complications to this project. For example, a simple Yes/No type question such as *Do you own a house?* is fairly common in the SFS 2019. A binary question like this does not present a challenge on its own; however, follow-up questions like *What year did you purchase your house?* or *What is the value of your property?* create complications because, in this case, only 62.41% of the population own a house – which means that 37.59% of the population skipped the follow-up questions. Therefore, even though the dataset does not include any missing values, a thorough data reconfiguration process had to be implemented before any formal data exploration could be performed.

Continuing with the homeownership example to illustrate the rationale followed in the data reconfiguration process, the next logical question was: what to do with skipped values? Only homeowners answered the follow-up *What year did you purchase your house?* question and non-homeowners were assigned a '96' skip code answer. Since we knew why these questions were skipped and what the '96' code means, we didn't need to calculate an imputation value, but a re-coding was necessary. The following recoding approaches were considered:

- Assigning a '0' code. For this specific question, assigning a zero code did not make sense as it would have generated inaccurate data. Doing so would have meant that a house was purchased more than 2000 years ago and no meaningful calculations could have been performed on the variable.
- Assigning the median or mean purchase year. This seemed a more reasonable option; however, it was decided against it because we were not facing the traditional "missing-

value scenario". It is known who did and did not purchase a house; therefore, a better approach had to be possible.

- Calculate the number of years between the purchase year and 2019. By following this approach and assigning a 2019 purchase year to those who skipped the question, only homeowners would have a value greater than zero, and those who skipped the question would have a zero. Moreover, calculations and descriptive statistics could be applied to the resulting data. Refer to Figure 7 for a visualization of the re-coding of the 'PASRBUYG' – *In what year did you purchase this property?* question.

Figure 7. 'PASRBUYG' reconfiguration code.

```python
# 1 - Recode to years (32)
data['PASRBUYG']=np.where(data['PASRBUYG']==96, (2019-2019),
np.where(data['PASRBUYG']==1, (2019-1960),
np.where(data['PASRBUYG']==2, (2019-1970),
np.where(data['PASRBUYG']==3, (2019-1980),
np.where(data['PASRBUYG']==4, (2019-1990),
np.where(data['PASRBUYG']==5, (2019-2000),
np.where(data['PASRBUYG']==6, (2019-2005),
np.where(data['PASRBUYG']==7, (2019-2010),
np.where(data['PASRBUYG']==8, (2019-2015),
np.where(data['PASRBUYG']==9, (2019-2017), 1)))))))))
```

Still addressing the homeownership example, dealing with the *What is the value of your property?* question introduced a new data problem: aggressive skewness. Even though the skewness topic will be formally addressed in a later section of this report, it is worth mentioning that this problem was clearly identified at a very early stage of this project. Since 37.59% of the answers to the property value question were zero, a variable that was already expected to be skewed (only a few people own very expensive properties when compared to the rest of the population) was skewed even further. Since this was a situation that almost systematically affected other variables related to income, debt, investments, assets, and equity value of owned businesses, it was decided that dummy variables would be generated in a systematic way as an

early option to generate quality information, at the same time that the skewness problem was avoided and/or curbed.

In this case, if the value of the variable 'PASRBUYG' – *value of principal residence*, was greater than zero, the dummy 'D_PASRBUYG' would be given a value of 1; otherwise, the value would be 0. Refer to Figure 8 for a visualization of the code. The same process was followed to create dummies for any variable that included dollar amounts.

Figure 8. 'D_PASRBUYG'

```
# 1 - Create D_PASRBUYG': Home owner IN Canada? (32)
data['D_PASRBUYG']=data['PASRBUYG']
data['D_PASRBUYG']=np.where((data['D_PASRBUYG']!=96),1,0)
print((data[data['D_PASRBUYG']==1]['PWEIGHT'].sum()/data['PWEIGHT'].sum()*100).round(4),
'% of households own a home in CANADA')
```

Python

```
62.4058 % of households own a home in CANADA
```

Data Preparation: Real Estate

PASRBUYG. Property purchase year - As it has been explained, a dummy and a recoding process was implemented of this variable. PASRCNMG. Number of other real estate owned in Canada – 'D_PASRCNMG' was created, skips were recoded as zeroes. Refer to Figure 9.

Figure 9.

```
# 2 - Create D_PASRCNMG: A dummy variable for 'OTHER' real estate IN Canada (34)
data['D_PASRCNMG']=data['PASRCNMG']
data['D_PASRCNMG']=np.where(data['D_PASRCNMG']==6,0,1)
print((data[data['D_PASRCNMG']==1]['PWEIGHT'].sum()/data['PWEIGHT'].sum()*100).round(2),
'% of households have OTHER real estate IN Canada')
```

Python

```
13.12 % of households have OTHER real estate IN Canada
```

```
# 2.1 - Recode missings to zeroes (34)
data['PASRCNMG'].replace(to_replace=[6], value=[0],inplace=True)
data['PASRCNMG'].max()
```

Python

```
4
```

PASRFNMG. Number of other real estate outside of Canada – A dummy was created and skips were recoded as zeroes. Refer to Figure 10.

Figure 10.

```
# 3 - Create D_PASRFNMG: dummy for those who have a property OUTSIDE of Canada / Recoding (53)
data['PASRFNMG'].replace(to_replace=[6],value=[0], inplace=True)
data['D_PASRFNMG']=data['PASRFNMG']
data['D_PASRFNMG']=np.where(data['D_PASRFNMG']==0,0,1)
print((pd.DataFrame(data['D_PASRFNMG'].value_counts())).sort_index())
print((pd.DataFrame(data['PASRFNMG'].value_counts())).sort_index())
```

Python

```
   D_PASRFNMG
0        9877
1         545
   PASRFNMG
0      9877
1       427
2       118
```

PASRCON. Property a condominium development? – The 'Not Stated' class was collapsed with the largest category 'No'. 'Not Stated' only represented 0.1% in a three-category categorical variable. Refer to Figure 11.

Figure 11.

```
# 4 - Recode: Class collapse (35)
data['PASRCON'].replace(to_replace=[9], value=[2], inplace=True)
print((pd.DataFrame(data['PASRCON'].value_counts())).sort_index())
```

Python

```
   PASRCON
1     1045
2     9377
```

PASRCST. Purchase property price? – The 'Not Stated' category '99999999' was collapsed with the skip category '99999996'. A new variable 'IMP_PASRCST' was created. 0 was imputed to skips. A dummy 'Homeowner in Canada' was created before. Refer to Figure 12.

Figure 12.

```
# 5 - Impute: zeroes to those who did not purchase a house (36) / drop original variable
data['PASRCST'].replace(to_replace=[99999999], value=[99999996], inplace=True)
data['IMP_PASRCST']=data['PASRCST']
data['IMP_PASRCST'].replace(to_replace=[99999996], value=[0], inplace=True)
data.drop(columns=['PASRCST'], inplace=True)
print('New mean:',data['IMP_PASRCST'].mean())
print('Max:',data.IMP_PASRCST.max())
print('PASRCST has been dropped')
```

Python

```
New mean: 193677.29658414892
Max: 3800000
PASRCST has been dropped
```

PASRDWNG. Down payment in percentage terms – The 'Valid Skip' category was collapsed with the 0 category. The rest of the categories were recoded to show the middle

percentage value of the class instead of a code. If 1 to 10% of down payment for was in class 01, the category was recoded to 5. Refer to Figure 13.

Figure 13.

```
# 6 - Recode variable to percentages (51)
data['PASRDWNG'].replace(to_replace=[96], value=[0], inplace=True)
data['PASRDWNG']=np.where(data['PASRDWNG']==0, 0,
np.where(data['PASRDWNG']==1, 5,
np.where(data['PASRDWNG']==2, 15,
np.where(data['PASRDWNG']==3, 25,
np.where(data['PASRDWNG']==4, 35,
np.where(data['PASRDWNG']==5, 45,
np.where(data['PASRDWNG']==6, 55,
np.where(data['PASRDWNG']==7, 65,
np.where(data['PASRDWNG']==8, 90, 100)))))))))
```

Python

PASRRNTG. Part of the property rented out – The 'Not Stated' category was collapsed with the 'Valid Skip' category. Not Stated' only represented 0.3% in a three-category variable. Refer to Figure 14.

Figure 14.

```
# 7 - Recode variable - Class collapse (64)
data['PASRRNTG'].replace(to_replace=[9], value=[6], inplace=True)
```

Python

PFTENUR. Principal residence ownership status – a dummy was created to capture homeowners with no mortgage. Homeowners with no mortgage were assigned a value of 1, 0 otherwise. Refer to Figure 15.

Figure 15.

```
# 8 - D_PFTENUR: Dummy for homeowners with no mortgage (138)
data['D_PFTENUR']=data['PFTENUR']
data['D_PFTENUR']=np.where(data['D_PFTENUR']==1,1,0)
data['D_PFTENUR'].value_counts()
```

Python

```
0    6743
1    3679
Name: D_PFTENUR, dtype: int64
```

Data Preparation: Mortgage

PASR1MR. Mortgage renegotiated for reasons unrelated to home – a dummy was created to if mortgage was renegotiated for reasons unrelated to home. Skips were recoded to 0, otherwise 1. Refer to Figure 16.

Figure 16.

```
# 1 - Create D_PASR1MR: A dummy mortgage variable (31); also look to 138
data['D_PASR1MR']=data['PASR1MR']
data['D_PASR1MR']=np.where(data['D_PASR1MR']==6,0,1)
print((data[data['D_PASR1MR']==1]['PWEIGHT'].sum()/data['PWEIGHT'].sum()*100).round(2),
'% of households have at least one mortgage in Canada')
```

Python

```
34.83 % of households have at least one mortgage in Canada
```

PEXMG1A. Amount of mortgage payments – zeroes were imputed to skips. IMP_PEXMG1A was created and the original variable dropped. Refer to Figure 17.

Figure 17.

```
# 2 - Create IMP_PEXMG1A: Zeroes are imputed to those who dont have a mortgage (123)
data['IMP_PEXMG1A']=data['PEXMG1A']
data.loc[data['IMP_PEXMG1A']==99996,['IMP_PEXMG1A']]=0
print('Households ith a 99996 code:',data[data['IMP_PEXMG1A']==99996]['IMP_PEXMG1A'].count())
print('(ALL) Households (INCLUDING THOSE THAT DONT HAVE MORTGAGES) make an average payment of:',
data.IMP_PEXMG1A.mean())
data.drop(columns=['PEXMG1A'], inplace=True)
print('PEXMG1A has been dropped')
```

Python

```
Households ith a 99996 code: 0
(ALL) Households (INCLUDING THOSE THAT DONT HAVE MORTGAGES) make an average payment of: 367.60650546919976
PEXMG1A has been dropped
```

PEXMG1F. Frequency of mortgage payments – ‘Valid Skip’ and ‘Other’ were recoded to 0. Now, skips make zero payments a month. Monthly make 1, and so on. Refer to Figure 18.

```
# 3 - Mortgage payment recode (128)
data['PEXMG1F'].replace(to_replace=[6],value=[0], inplace=True)
data['PEXMG1F'].replace(to_replace=[5],value=[0], inplace=True)
data['PEXMG1F'].replace(to_replace=[4],value=[2], inplace=True)
code=[0,1,2,4]
mort_payment_freq=['Skip','Monthly','Every two weeks','Weekly']
pd.DataFrame({'New Code':code,'Payment Frequency':mort_payment_freq, 'Count':data['PEXMG1F'].value_counts().sort_index()
```

Python

	New Code	Payment Frequency	Count
0	0	Skip	6707
1	1	Monthly	1413
2	2	Every two weeks	1870
3	4	Weekly	432

PASR1MFA. Amount of additional mortgage payments – the dummy 'D_PASR1MFA' was created. Skips were recoded to 0, otherwise, 1. Refer to Figure 19.

Figure 19.

```
# 4.1 - Create D_PASR1MFA': A dummy variable for mortgage payments on additional property(23)
data['D_PASR1MFA']=data['PASR1MFA']
data['D_PASR1MFA']=np.where((data.D_PASR1MFA!=99999996) & (data.D_PASR1MFA>0),1,0)
print((data[data['D_PASR1MFA']==1]['PWEIGHT'].sum()/data['PWEIGHT'].sum()*100).round(2),
'% of households made mortgage payments on additional properties')
```

Python

```
8.01 % of households made mortgage payments on additional properties
```

PASR1MFA. Amount of additional mortgage payments – 'IMP_PASR1MFA' was created. 0 was imputed to skips. The original variable was dropped. Refer to Figure 20.

Figure 20.

```
# 4.2 - Creating a IMP_PASR1MFA: Zeroes are imputed to those who skipped the question (23) -
data['IMP_PASR1MFA']=data['PASR1MFA']
data.loc[data['IMP_PASR1MFA']==99999996,['IMP_PASR1MFA']]=0
data.drop(columns=['PASR1MFA'], inplace=True)
print('Households with a 99999996 code:',data[data['IMP_PASR1MFA']==99999996]['IMP_PASR1MFA'].count())
print('(ALL) Households (INCLUDING THOSE THAT DONT HAVE MORTGAGES ON ADDITIONAL PROPERTIES) make a ADDITIONAL average mo
data.IMP_PEXMG1A.mean())
print('PASR1MFA has been dropped')
```

Python

```
Households with a 99999996 code: 0
(ALL) Households (INCLUDING THOSE THAT DONT HAVE MORTGAGES ON ADDITIONAL PROPERTIES) make a ADDITIONAL average mortgage
payment of: 367.60650546919976
PASR1MFA has been dropped
```

PAS1MRAG. Amount borrowed at time of refinancing – 'D_ PAS1MRAG' was created. Skips were recoded to 0, otherwise, 1. The original variable was dropped, categories other than Valid Skip only represented very small fractions of the total population. On a 5-category variable, the second largest category accounted for 0.8 of the population. Refer to Figure 21.

Figure 21.

```
# 5 Creating a D_PAS1MRAG: Did you refinance your mortgage? (20)
data['D_PAS1MRAG']=np.where(data['PAS1MRAG']==6,0,1)
print(data['D_PAS1MRAG'].value_counts())
data.drop(columns=['PAS1MRAG'],inplace=True)
print('PAS1MRAG has been dropped')
```

Python

```
0    10186
1      236
Name: D_PAS1MRAG, dtype: int64
PAS1MRAG has been dropped
```

PASRCURG. Current length of mortgage term – 'Valid Skip' and 'Other' were recoded to 0. Categories were recoded to the number of years they represented. Refer to Figure 22.

Figure 22.

```
# 6 - Recode skips to 0 and turn codes into years (44)
data['PASRCURG'].replace(to_replace=[96], value=[0], inplace=True)
data['PASRCURG'].replace(to_replace=[8], value=[0], inplace=True)
data['PASRCURG']=np.where(data['PASRCURG']==0,0,
np.where(data['PASRCURG']==1,0.6,
np.where(data['PASRCURG']==2,1,
np.where(data['PASRCURG']==3,2,
np.where(data['PASRCURG']==4,3,
np.where(data['PASRCURG']==5,4,
np.where(data['PASRCURG']==6,5,10)))))))
(pd.DataFrame(data['PASRCURG'].value_counts())).sort_index()
```

Python

	PASRCURG
0.0	6864
0.6	44
1.0	163
2.0	195
3.0	597
4.0	191
5.0	2264
10.0	104

PASRINTG. Current interest rate on mortgage – 'Valid Skip' was recoded to 0. Categories were recoded to the higher band of their class. The recoded values represent the highest interest rates in their categories. Refer to Figure 23.

Figure 23.

```
# 7 - Recoding interest rates (55)
data['PASRINTG']=np.where(data['PASRINTG']==96,0,
np.where(data['PASRINTG']==1,1.99,
np.where(data['PASRINTG']==2,2.49,
np.where(data['PASRINTG']==3,2.99,
np.where(data['PASRINTG']==4,3.49,
np.where(data['PASRINTG']==5,3.99,
np.where(data['PASRINTG']==6,4.99,6)))))))
(pd.DataFrame(data['PASRINTG'].value_counts())).sort_index()
```

Python

	PASRINTG
0.00	6697
1.99	51
2.49	366
2.99	1287
3.49	1108
3.99	530
4.99	244
6.00	139

PASRMOAG. Original amortization period for the mortgage – 'Valid Skip' was recoded to 0. Categories were recoded to the higher band of their class. The recoded values represent the maximum number of years in their categories. Refer to Figure 24.

Figure 24.

```
# 8 - Recoding interest terms codes into years (57)
data['PASRMOAG']=np.where(data['PASRMOAG']==6,0,
np.where(data['PASRMOAG']==1,14,
np.where(data['PASRMOAG']==2,24,
np.where(data['PASRMOAG']==3,29,40))))
(pd.DataFrame(data['PASRMOAG'].value_counts())).sort_index()
```

Python

	PASRMOAG
0	6697
14	128
24	708
29	2298
40	591

PASRMPFG. Number of years to pay down remaining balance of mortgage – 'Valid Skip' was recoded to 0. Categories were recoded to the higher band of their class. The recoded values represent the maximum number of years in their categories. Refer to Figure 25.

Figure 25.

```
# 9 - Recoding codes into number of years reaminig in the mortgage (58)
data['PASRMPFG']=np.where(data['PASRMPFG']==96,0,
np.where(data['PASRMPFG']==1,5,
np.where(data['PASRMPFG']==2,9,
np.where(data['PASRMPFG']==3,14,
np.where(data['PASRMPFG']==4,19,
np.where(data['PASRMPFG']==5,24,30))))))
(pd.DataFrame(data['PASRMPFG'].value_counts())).sort_index()
```

Python

	PASRMPFG
0	6697
5	343
9	389
14	467
19	825
24	1213
30	488

PASRMRYG. Year mortgage up for renewal – 'Valid Skip' was recoded to 0. The remaining categories were coded to display the number of years until 2018. If mortgage is up for renewal in 2019, then 2018-2019 = = 1. If mortgage up for renewal in 2020, then 2018 – 2020 = = 2, and so on. Refer to Figure 26.

Figure 26.

```
# 10 - Recoding codes into number of years remaining for renewal (60)
data['PASRMRYG']=np.where(data['PASRMRYG']==9996,0,
np.where(data['PASRMRYG']==2019,(2019-2018),
np.where(data['PASRMRYG']==2020,(2020-2018),
np.where(data['PASRMRYG']==2021,(2021-2018),
np.where(data['PASRMRYG']==2022,(2022-2018),
np.where(data['PASRMRYG']==2023,(2023-2018),6)))))
(pd.DataFrame(data['PASRMRYG'].value_counts())).sort_index()
```

Python

	PASRMRYG
0	6697
1	159
2	823
3	807
4	739
5	625
6	572

Data Preparation: Credit & Credit Card

PATTLMLC. Total credit limit on all lines of credit – 'Valid Skip' was recoded to 0. There already is a dummy for this variable. PATTLCP. Line of credit balances usually paid off every month – 'Valid Skip' was recoded to 0. PATTCRLM. Total credit limit on all credit cards owned – 'Valid Skip' was recoded to 0. There is already a dummy have credit cards? Refer to Figure 27.

Figure 27.

```
# 1- Recode: impute zeroes to skipped values (86) - There's already a dummy: have lines of credit? (86)
data['PATTLMLC'].replace(to_replace=[99999996], value=[0], inplace=True)
data['PATTLMLC'].skew()
```

Python

```
5.517677265777087
```

```
# 2- Recode: Imputing zeroes to skipped values (76)
data['PATTLCP'].replace(to_replace=[6], value=[0], inplace=True)
```

Python

```
# 3 - Recode: Imputing zeroes to skipped values (67) - There's already a dummy: have credit cards?
data['PATTCRLM'].replace(to_replace=[999996], value=[0], inplace=True)
data.PATTCRLM.skew()
```

Python

```
7.353505973637755
```

PATTCRU. Credit card balances usually paid off every month – 'Valid Skip' was recoded to 0. Now, data is more 'ordinal', the higher the code, the more 'on-time' the payment

was made. A dummy 'D_PATTCRU' was created to capture if a household paid the minimum amount or less. Refer to Figure 28.

Figure 28.

```
# 4 - Recoding skips to zeroes (74)
data['PATTCRU'].replace(to_replace=[6],value=[0], inplace=True)
```

Python

```
# 4.1 - Creating a D_PATTCRU: Did you pay the minimum amount or less? (74)
data['D_PATTCRU']=data['PATTCRU']
for i in data['D_PATTCRU']:
    if i==2:
        data['D_PATTCRU'].replace(to_replace=[i], value=[1], inplace=True)
    elif i==3:
        data['D_PATTCRU'].replace(to_replace=[i], value=[1], inplace=True)
    elif i==4:
        data['D_PATTCRU'].replace(to_replace=[i], value=[0], inplace=True)
    elif i==5:
        data['D_PATTCRU'].replace(to_replace=[i], value=[0], inplace=True)
data.D_PATTCRU.value_counts()
```

PFCRN. Number of credit cards used -grouped – 'Does not own credit cards' was recoded to 0. If 1 or 2 cards, the category was recoded to 1.5, if 3 of 4 cards, then 3.5 and so on. Now, data is more 'ordinal', the higher the code, the more credit cards. Refer to Figure 29.

Figure 29.

```
# 5 - Recoding skips into number of credit cards (129)
for i in data['PFCRN']:
    if i==4:
        data['PFCRN'].replace(to_replace=[i], value=[0], inplace=True)
    elif i==1:
        data['PFCRN'].replace(to_replace=[i], value=[1.5], inplace=True)
    elif i==2:
        data['PFCRN'].replace(to_replace=[i], value=[3.5], inplace=True)
    elif i==3:
        data['PFCRN'].replace(to_replace=[i], value=[5.5], inplace=True)
data.PFCRN.value_counts()
```

Python

```
1.5    5203
3.5    2983
5.5    1290
0.0     946
Name: PFCRN, dtype: int64
```

Data Preparation: Income

PEFGTR. Government transfers, federal & provincial – The dummy 'D1_PEFGTR' was created. 1 if household receives transfers, 0 otherwise. 'D2_PEFGTR' was created. 1 if household is in last decile, 0 otherwise. Refer to Figure 30.

Figure 30.

```
# 1.1 - Creating D1_PEFGTR: Do you receive government transfers? (105)
data['D1_PEFGTR']=data['PEFGTR']
for i in data['D1_PEFGTR']:
    if i>0:
        data['D1_PEFGTR'].replace(to_replace=[i],value=[1],inplace=True)
    elif i==0:
        data['D1_PEFGTR'].replace(to_replace=[i],value=[0],inplace=True)
data['D1_PEFGTR'].value_counts()
```

Python

```
1    9340
0    1082
Name: D1_PEFGTR, dtype: int64
```

```
# 1.2 - Creating D2_PEFGTR: is household in the last decile of amount of transfer income? (105)
data['D2_PEFGTR']=data['PEFGTR']
data['D2_PEFGTR']=np.where(data['D2_PEFGTR']>=data['D2_PEFGTR'].quantile(.9),1,0)
data['D2_PEFGTR'].value_counts()
```

Python

```
0    9318
1    1104
Name: D2_PEFGTR, dtype: int64
```

PNBEARG. Number of earners aged 15 or over in the family unit – ‘Not Stated’ was recoded to 1, the highest cardinality category. Refer to Figure 31.

Figure 31.

```
# 2 - Recoding Not Stated to the mode (151)
data['PNBEARG'].replace(to_replace=[9], value=[1],inplace=True)
```

Python

PINHERT. Total value of inheritances received in 2016 – ‘D_PINHERT’, if received inheritance, 1; 0 otherwise. ‘IMP_PINHERT’, 0 were imputed to skips. Refer to Figure 32.

Figure 32.

```
# 3.1 - Creating D_PINHERT: Did you receive an inheritance?
data['D_PINHERT']=data['PINHERT']
data['D_PINHERT']=np.where(data['D_PINHERT']==999999996,0,1)
data['D_PINHERT'].value_counts()
```

Python

```
0    6985
1    3437
Name: D_PINHERT, dtype: int64
```

```
# 3.2 - Creating IMP_PINHERT: Impute zeroes to those who didnt recieve an inheritance
data['IMP_PINHERT']=data['PINHERT']
data['IMP_PINHERT'].replace(to_replace=[999999996],value=[0], inplace=True)
data[data['IMP_PINHERT']==999999996]['IMP_PINHERT'].count()
```

Python

```
0
```

```
# 3.2.1 - Drop PINHERT
data.drop(columns=['PINHERT'], inplace=True)
```

Python

PEFATINC. After-tax income – 'D_PEFATINC', if household in first decile, 1; 0 otherwise. Refer to Figure 33.

Figure 33.

```
# 4 - Creating D_PEFATINC: is household in the first decile of after tax income?
data['D_PEFATINC']=data['PEFATINC']
data['D_PEFATINC']=np.where(data['D_PEFATINC']<=data['D_PEFATINC'].quantile(.1),1,0)
data['D_PEFATINC'].value_counts()
```

Python

```
0    9379
1    1043
Name: D_PEFATINC, dtype: int64
```

PEFMTINC. Market income – 'D_ PEFMTINC', if household in first decile income, 1; 0 otherwise. 'IMP_ PEFMTINC', 0 were imputed to skips. Refer to Figure 34.

Figure 34.

```
# 5.1 - Creating IMP_PEFMTINC: Impute zeroes to market income
data['IMP_PEFMTINC']=data['PEFMTINC']
data['IMP_PEFMTINC'].replace(to_replace=[99999999],value=[0], inplace=True)
data[data['IMP_PEFMTINC']==99999999]['IMP_PEFMTINC'].count()
```

Python

```
0
```

```
# 5.2 - Creating D_PEFATINC: is household in the first decile of before tax income? (115)
data['D_PEFMTINC']=data['PEFMTINC']
data['D_PEFMTINC']=np.where(data['D_PEFMTINC']<=data['D_PEFMTINC'].quantile(.1),1,0)
data['D_PEFMTINC'].value_counts()
```

Python

```
0    9370
1    1052
Name: D_PEFMTINC, dtype: int64
```

```
data.drop(columns=['PEFMTINC'],inplace=True)
```

Python

Data Preparation: Family Composition & Major Earner

PFSZ. Number of members in the family unit – all ages – 'Not Stated' was recoded to 2. Refer to Figure 35.

Figure 35.

```
# 1 - Recode Not Stated to the mode (131)
data['PFSZ'].replace(to_replace=[9], value=[2], inplace=True)
```

Python

PAGEMIEG. Age of the major income earner – Categories were recoded to the highest value in the class they represent. If class 20 to 24, then 24 and so on. Refer to Figure 36.

Figure 36.

```
for i in data['PAGEMIEG']:
    if i==1:
        data['PAGEMIEG'].replace(to_replace=[i],value=[19],inplace=True)
    elif i==2:
        data['PAGEMIEG'].replace(to_replace=[i],value=[24],inplace=True)
    elif i==3:
        data['PAGEMIEG'].replace(to_replace=[i],value=[29],inplace=True)
    elif i==4:
        data['PAGEMIEG'].replace(to_replace=[i],value=[34],inplace=True)
    elif i==5:
        data['PAGEMIEG'].replace(to_replace=[i],value=[39],inplace=True)
    elif i==6:
        data['PAGEMIEG'].replace(to_replace=[i],value=[44],inplace=True)
    elif i==7:
        data['PAGEMIEG'].replace(to_replace=[i],value=[49],inplace=True)
    elif i==8:
        data['PAGEMIEG'].replace(to_replace=[i],value=[54],inplace=True)
    elif i==9:
        data['PAGEMIEG'].replace(to_replace=[i],value=[59],inplace=True)
    elif i==10:
        data['PAGEMIEG'].replace(to_replace=[i],value=[64],inplace=True)
    elif i==11:
        data['PAGEMIEG'].replace(to_replace=[i],value=[69],inplace=True)
    elif i==12:
        data['PAGEMIEG'].replace(to_replace=[i],value=[74],inplace=True)
    elif i==13:
        data['PAGEMIEG'].replace(to_replace=[i],value=[79],inplace=True)
    elif i==14:
        data['PAGEMIEG'].replace(to_replace=[i],value=[84],inplace=True)
```

Data Preparation: Funds

For all the variables classified as Funds, refer to Figure 37, the dummy variable was created. If the original variable has a value greater than zero, then 'D_*varname*' takes the value 1; 0 otherwise. Refer to Figure 38.

Figure 37.

Variable	Description	Python - Data Type
PWAOTPEN	Other retirement funds	int64
PWARPPT	Value of all employer pension plans. Termination basis	int64
PWASTBND	Bonds, non-registered	int64
PWARRIF	Registered retirement income funds (RRIFs)	int64
PWARRSPL	RRSP investments including locked in RRSP's	int64
PWASTDEP	Money in banks, non-registered	int64
PWASTMUI	Mutual funds, other investment, Income trusts, non-registered	int64
PWASTOIN	Other investments or financial assets	int64
PWASTONF	Other non-financial assets	int64
PWASTSTK	Stock and shares, non registered	int64
PWATFS	Tax Free Savings Accounts (TFSA)	int64

Figure 38.

```python
fund_var=['PWAOTPEN','PWARPPT','PWASTBND','PWARRIF','PWARRSPL','PWASTDEP','PWASTMUI',
'PWASTOIN','PWASTONF','PWASTSTK','PWATFS']
```

```python
for i in data[fund_var]:
    data['D_'+i]=data[i]
    data['D_'+i]=np.where(data['D_'+i]>1,1,0)
```

Data Preparation: Debt

PWDTOTAL. Total of debts for the family – 'D_ PWDTOTAL', if total debts greater than 0, 1; 0 otherwise. PWDSLOAN. Debt value of student loans – 'D_ PWDSLOAN', if student loans greater than 0, 1; 0 otherwise. Refer to Figure 39.

Figure 39.

```python
# 1 Creating D_PWDTOTAL: Has debts? (457)
data['D_PWDTOTAL']=data['PWDTOTAL']
data['D_PWDTOTAL']=np.where(data['D_PWDTOTAL']>0,1,0)
data['D_PWDTOTAL'].value_counts()
```

```
1    7322
0    3100
Name: D_PWDTOTAL, dtype: int64
```

```python
# 2 - Creating D_PWDSLOAN: Has student loans? (385)
data['D_PWDSLOAN']=data['PWDSLOAN']
data['D_PWDSLOAN']=np.where(data['D_PWDSLOAN']>0,1,0)
data['D_PWDSLOAN'].value_counts()
```

```
0    9302
1    1120
Name: D_PWDSLOAN, dtype: int64
```

PWDSTODB. Other debt (other loans and other money owed) – 'D_ PWDSTODB', if other debt greater than 0, 1; 0 otherwise. PWDSTVHN. Debt on vehicles – 'D_ PWDSTVHN', if debt on vehicles greater than 0, 1; 0 otherwise. Refer to Figure 40.

Figure 40.

```python
# 3 - Creating D_PWDSTODB: Has debts with other financial institutions? (421)
data['D_PWDSTODB']=data['PWDSTODB']
data['D_PWDSTODB']=np.where(data['D_PWDSTODB']>0,1,0)
data['D_PWDSTODB'].value_counts()
```

```
0    9510
1     912
Name: D_PWDSTODB, dtype: int64
```

```python
# 4 - Creating D_PWDSTVHN: Has debts on vehicles? (445)
data['D_PWDSTVHN']=data['PWDSTVHN']
data['D_PWDSTVHN']=np.where(data['D_PWDSTVHN']>0,1,0)
data['D_PWDSTVHN'].value_counts()
```

```
0    7157
1    3265
Name: D_PWDSTVHN, dtype: int64
```

Data Preparation: Assets

PWASTVHE. Total of vehicles – 'D_ PWASTVHE', if total of vehicles greater than 0, 1; 0 otherwise. PWATOTPT. Total assets, including employer pension plans – 'D_ PWATOTPT', if total assets value in first decile, 1; 0 otherwise. Refer to Figure 41.

Figure 41.

```
# 1 - Creating D_PWASTVHE: Has vehicles? (313)
data['D_PWASTVHE']=data['PWASTVHE']
data['D_PWASTVHE']=np.where(data['D_PWASTVHE']>0,1,0)
data['D_PWASTVHE'].value_counts()
```

Python

```
1    8888
0    1534
Name: D_PWASTVHE, dtype: int64
```

```
# 2 - Creating D_PWATOTPT: is in the first decile?
data['D_PWATOTPT']=data['PWATOTPT']
data['D_PWATOTPT']=np.where(data['D_PWATOTPT']<=data['D_PWATOTPT'].quantile(.1),1,0)
data['D_PWATOTPT'].value_counts()
```

Python

```
0    9379
1    1043
Name: D_PWATOTPT, dtype: int64
```

Data Preparation: Net Worth

PWNETWPT. Net worth of the family unit – 'D1_ PWNETWPT' finds if a family unit has a negative net worth. If net worth less than 0, 1; 0 otherwise. 'D1_ PWNETWPT' finds if a family unit has a net worth that is located in the first decile. If net worth in first decile. 1; 0 otherwise. Refer to Figure 42.

Figure 42.

```
# 1 - Creating D_PWNETWPT: Has negative equity?
data['D1_PWNETWPT']=data['PWNETWPT']
data['D1_PWNETWPT']=np.where(data['D1_PWNETWPT']<0,1,0)
data['D1_PWNETWPT'].value_counts()
```

Python

```
0    9941
1     481
Name: D1_PWNETWPT, dtype: int64
```

```
data['D2_PWNETWPT']=data['PWNETWPT']
data['D2_PWNETWPT']=np.where(data['D2_PWNETWPT']<=data['D2_PWNETWPT'].quantile(.1),1,0)
data['D2_PWNETWPT'].value_counts()
```

Python

```
0    9377
1    1045
Name: D2_PWNETWPT, dtype: int64
```

Data Exploration & Feature Engineering

Descriptive Statistics – Initial Analysis

Initial descriptive statistics tables were created for the integer and categorical variables in the data and, as it was found, there is a serious case of skewness in most (integer) variables. The maximum skewness, 25.93, was found in PWASTSTK – stocks and shares, non-registered. Only five variables – PASRINT, PASRMOAG, PFCRN, PFSZ, and PAGEMIEG – presented acceptable skewness levels (less than 1). Refer to Figure 43 and Figure 44 for a visualization of the integer data code and descriptive statistics table.

Figure 43. Descriptive Statistics Code – Integer Data

```
# 1. Initial descriptive statistics for integer data
pre_eda_int_data_stats=pd.DataFrame({
'std dev':(data[int_data].std(axis=0)).round(2),
'mean':(data[int_data].mean(axis=0)).round(2),
'min':(data[int_data].min(axis=0)).round(),
'max':data[int_data].max(axis=0),
'skew':(data[int_data].skew(axis=0)).round(2),
'kurtosis':(data[int_data].kurtosis(axis=0)).round(2),
'null values':data[int_data].isnull().sum()}, )
pre_eda_int_data_stats.sort_values(by=['skew'], ascending=False)
```

Given that all the categorical data classes were coded as numbers, integer data statistics like median, mode, min, and max were calculated only as a first step to detect irregularities in the data. For example, if a max value for a variable was 9 in a yes (1) / no (0) question, obviously, that would mean that a piece of code did not work properly in the data reconfiguration process implemented in the previous phase of this project. No other conclusions should be drawn from these statistics.

The '%-mode' and '%-least common value' columns were created to show what are the percentages that the mode and least common value observations take from the total population. In addition to the 'number of classes' column that show how many categories there are in each variable, the '%-mode' and '%-least common value' columns look to provide a glimpse into the distribution of the data's categorical variables. Refer to Figure 45 and Figure 46 for visualizations of the categorical variables' code and the descriptive statistics.

Figure 44. Descriptive Statistics Data Frame – Integer Data

	std dev	mean	min	max	skew	kurtosis	null values
PWASTSTK	429036.85	39434.71	1.0	16250000.0	25.93	842.01	0
PWASTBND	34303.47	2049.53	0.0	1100000.0	24.82	687.50	0
PWASTOIN	124658.80	13665.23	0.0	4600000.0	24.51	759.31	0
PWASTMUI	442391.02	49345.34	0.0	15750000.0	22.86	687.79	0
PWARRIF	197212.37	29524.19	0.0	7250000.0	21.44	657.24	0
PWDSTODB	16624.55	1806.94	0.0	500000.0	20.04	490.55	0
PWAOTPEN	42875.86	4337.22	0.0	1300000.0	18.11	405.17	0
PWASTRST	529246.43	108198.70	0.0	17250000.0	17.12	452.12	0
IMP_PASR1MFA	6400.61	1006.60	0.0	140000.0	11.47	172.30	0
IMP_PEXMG1A	940.41	367.61	0.0	24000.0	10.74	199.06	0
IMP_PINHERT	197833.18	49067.26	0.0	3900000.0	10.74	158.64	0
PWDSTLOC	50561.14	10853.64	0.0	1100000.0	10.40	150.46	0
PWASTONF	80889.80	31541.29	175.0	1550000.0	9.86	134.79	0
IMP_PEFMTINC	124843.03	93597.53	-47000.0	3300000.0	9.18	174.37	0
PWASTDEP	111155.20	33601.90	-3300.0	2200000.0	9.07	117.87	0
PWBUSEQ	663578.26	108438.88	-1350000.0	9000000.0	9.00	92.48	0
PWDSTOMR	117809.40	22604.03	0.0	1600000.0	8.49	88.59	0
PWDSLOAN	9494.05	2062.12	0.0	165000.0	8.39	97.89	0
PWDSTCRD	7731.29	2568.98	0.0	150000.0	7.88	98.24	0
PATTCRLM	31788.76	23170.63	0.0	725000.0	7.35	110.55	0
PEFATINC	80557.09	86595.35	-479300.0	2159100.0	6.88	113.09	0
PWAPRVAL	514434.83	368323.67	0.0	11500000.0	6.30	96.43	0
PWNETWPT	1734456.15	1023933.30	-910500.0	30055000.0	5.73	54.37	0
PWATOTPT	1781139.29	1148902.62	-228730.0	31004175.0	5.57	52.29	0
PATTLMLC	103691.18	42590.00	0.0	1650000.0	5.52	46.23	0
PASRFNMG	0.29	0.06	0.0	2.0	4.90	25.23	0
PWARRSPL	239245.33	98845.74	0.0	2800000.0	4.89	33.52	0
PWASTVHE	37922.15	26865.90	0.0	575000.0	4.80	43.94	0
PWATFS	42081.10	20118.75	0.0	700000.0	4.20	36.45	0
IMP_PASRCST	291748.63	193677.30	0.0	3800000.0	4.08	30.22	0
PWDTOTAL	222942.49	124969.31	0.0	2807500.0	3.83	22.82	0
PWDPRMOR	151877.84	77315.65	0.0	1900000.0	3.40	20.41	0
PASRCNMG	0.63	0.25	0.0	4.0	3.21	11.96	0
PWDSTVHN	16591.83	7757.96	0.0	155000.0	3.10	12.46	0
PWARPPT	398283.88	214612.45	0.0	3600000.0	2.95	11.31	0
PASRDWNG	32.25	22.73	0.0	100.0	1.54	1.02	0
PASRBUYG	12.63	10.25	0.0	59.0	1.44	1.62	0
PEFGTR	12439.28	11877.01	0.0	145000.0	1.36	4.85	0
PASRCURG	2.26	1.49	0.0	10.0	1.20	0.43	0
PASRMRYG	1.97	1.32	0.0	6.0	1.16	-0.13	0
PEXMG1F	0.92	0.62	0.0	3.0	1.15	-0.07	0
PASRMPFG	10.19	6.83	0.0	30.0	1.07	-0.50	0
PASRINTG	1.73	1.24	0.0	6.0	0.87	-0.75	0
PASRMOAG	14.43	10.46	0.0	40.0	0.76	-1.17	0
PFCRN	1.57	2.43	0.0	5.5	0.57	-0.54	0
PFSZ	1.07	2.25	1.0	4.0	0.46	-1.01	0
PAGEMIEG	16.61	55.22	19.0	84.0	-0.09	-0.96	0

Figure 45. Descriptive Statistics Code – Categorical Data

```
# 1. Initial descriptive statistics for categorical data
pre_eda_cat_data=pd.DataFrame({
'median':(data[cat_data].median(axis=0)).round(2),
'mode':data[cat_data].value_counts().idxmax(),
'min':(data[cat_data].min(axis=0)).round(),
'max':data[cat_data].max(axis=0),
'null values':data[cat_data].isnull().sum()},index=cat_data)
# Creating the 'Percentage Mode','Percentage least common value', and 'Count'
cols=['% - mode','% - least common value','number of classes']
lst=[]
for i in data[cat_data]:
    percentage=(data[i].value_counts(normalize=True)).max()*100
    percentage2=(data[i].value_counts(normalize=True)).min()*100
    count=(data[i].value_counts()).count()
    lst.append([percentage,percentage2,count])
class_dist=pd.DataFrame(lst, columns=cols, index=cat_data)
# Joining the two dataframe and sorting values
pre_eda_cat_data_stats=pd.concat([pre_eda_cat_data,class_dist], axis=1)
pre_eda_cat_data_stats['% - mode']=(pre_eda_cat_data_stats['% - mode']).round(2)
pre_eda_cat_data_stats['% - least common value']=(pre_eda_cat_data_stats['% - least common value']).round(2)
pre_eda_cat_data_stats.sort_values(by=['% - mode'], ascending=False).head(50)
```

Data Quality – Integer Data: Correlated Variables

The code in Figure 47 was run to 1) find variables with a Person correlation index greater than 0.7, 2) include variables' skewness and correlation to the target variable to decide which correlated variables are provide the highest quality information. The rest were be discarded.

Figure 47. Data Quality Code – Integer Data

```
# Finding Integer Correlated Variables ~ @ > 0.7 perason corr
correlation_matrix_int_data=data[int_data].corr(method='pearson')
lst=[]
cond=correlation_matrix_int_data>=0.7
for col in int_data:
    rows = list(cond[col][cond[col]==True].index)
    for row in rows:
        lst.append((row,col))
correlated=pd.DataFrame(lst, columns=['Variable 1','Variable 2'])
correlated2=correlated[correlated['Variable 1']!=correlated['Variable 2']]
correlated_unique=correlated2[~correlated2[['Variable 1','Variable 2']].apply(frozenset, axis=1).duplicated()]
correlated_unique.reset_index(drop=True, inplace=True)
correlated_unique.index+=1
correlated_unique.sort_values(by='Variable 1').inplace=True
# Pulling the skewness for each variable from pre_eda_int_data_stats & Calculating variables' correlation to the target variable
lstv1=[]
lstv2=[]
lst1=correlated_unique['Variable 1'].tolist()
lst2=correlated_unique['Variable 2'].tolist()
for i in lst1:
    cor1=data[i].corr(data['PATTSKP'])
    skew1=pre_eda_int_data_stats.loc[i,'skew']
    lstv1.append([cor1,skew1])
var1cor=pd.DataFrame(lstv1, columns=['Var1-Target Corr','Skew Var 1'])
for j in lst2:
    cor2=data[j].corr(data['PATTSKP'])
    skew2=pre_eda_int_data_stats.loc[j,'skew']
    lstv2.append([cor2,skew2])
var2cor=pd.DataFrame(lstv2, columns=['Var2-Target Corr','Skew Var 2'])
varcor=pd.concat([var1cor,var2cor], axis=1)
varcor.set_index(correlated_unique.index,inplace=True)
integer_var_cor=pd.concat([correlated_unique,varcor],axis=1)
```

Figure 46. Descriptive Statistics Data Frame – Categorical Data

8]

	median	mode	min	max	null values	% - mode	% - least common value	number of classes
D_PWASTONF	1.0	1	1	1	0	100.00	100.00	1
D_PAS1MRAG	0.0	0	0	1	0	97.74	2.26	2
PATTPAYD	2.0	2	1	2	0	97.62	2.38	2
D_PWASTBND	0.0	0	0	1	0	97.14	2.86	2
D1_PWNETWPT	0.0	0	0	1	0	95.38	4.62	2
D_PASRFNMG	0.0	0	0	1	0	94.77	5.23	2
D_PWASTDEP	1.0	1	0	1	0	94.67	5.33	2
D_PWAOTPEN	0.0	0	0	1	0	93.48	6.52	2
PATTDIF	6.0	6	1	6	0	93.11	2.25	3
D_PASR1MFA	0.0	0	0	1	0	91.75	8.25	2
D_PWDSTODB	0.0	0	0	1	0	91.25	8.75	2
PATTCRR	6.0	6	1	6	0	90.92	1.78	3
PATTCRC	1.0	1	1	2	0	90.92	9.08	2
D_PEFATINC	0.0	0	0	1	0	89.99	10.01	2
D_PWATOTPT	0.0	0	0	1	0	89.99	10.01	2
PASRCON	2.0	2	1	2	0	89.97	10.03	2
D2_PWNETWPT	0.0	0	0	1	0	89.97	10.03	2
D_PEFMTINC	0.0	0	0	1	0	89.91	10.09	2
D1_PEFGTR	1.0	1	0	1	0	89.62	10.38	2
D2_PEFGTR	0.0	0	0	1	0	89.41	10.59	2
D_PWDSLOAN	0.0	0	0	1	0	89.25	10.75	2
D_PWASTSTK	0.0	0	0	1	0	88.51	11.49	2
D_PWARRIF	0.0	1	0	1	0	87.45	12.55	2
D_PWASTMUI	0.0	0	0	1	0	86.73	13.27	2
D_PWASTVHE	1.0	1	0	1	0	85.28	14.72	2
D_PASRCNMG	0.0	0	0	1	0	82.48	17.52	2
PBUSIND	2.0	2	1	2	0	80.69	19.31	2
PFSZ1824	2.0	2	1	9	0	78.45	8.79	3
D_PWASTOIN	0.0	0	0	1	0	77.26	22.74	2
PATTRSP	1.0	1	1	2	0	71.76	28.24	2
D_PASRBUYG	1.0	1	0	1	0	71.01	28.99	2
PATTRSL	2.0	2	1	6	0	70.40	1.36	3
D_PWDTOTAL	1.0	0	0	1	0	70.26	29.74	2
D_PATTCRU	0.0	0	0	1	0	69.28	30.72	2
PFSZ0017	2.0	2	1	9	0	68.84	8.79	3
D_PWDSTVHN	0.0	0	0	1	0	68.67	31.33	2
PASRRNTG	2.0	2	1	6	0	67.90	2.79	3
D_PINHERT	0.0	0	0	1	0	67.02	32.98	2
D_PFTENUR	0.0	1	0	1	0	64.70	35.30	2
PASRSKP	6.0	6	1	6	0	64.26	1.30	3
PASRINT	6.0	6	1	6	0	64.26	1.58	4
D_PASR1MR	0.0	0	0	1	0	64.26	35.74	2
PASR1MR	6.0	6	1	6	0	64.26	2.26	3
PDWTYP	1.0	1	1	9	0	63.38	3.25	4
PLFFPTME	1.0	3	1	6	0	61.14	2.67	4
PFSZ65UP	2.0	1	1	9	0	61.09	8.79	3
PEFMJSIF	2.0	6	1	7	0	60.78	0.23	7
PGDRMIE	1.0	2	1	2	0	59.94	40.06	2
PASRDPO4	2.0	2	1	6	0	59.93	5.11	3
PASRDPO5	2.0	2	1	6	0	59.36	5.68	3

Figure 48 shows all the unique pairs of variables that have a correlation higher than 0.7. The variables with the lowest skewness and higher correlation to the target variables were considered superior and maintained in the data. A quick scan of the table that IMP_PEFMTINC and PEFATINC are to each other. However, IMP_PEFMTINC is a lower quality variable because it is more highly skewed and its correlation to the target variable is lower than PEFATINC. For a visualization of the correlation coefficients, refer to Figure 49, a heat map of the correlation coefficients before lower-quality variables are dropped.

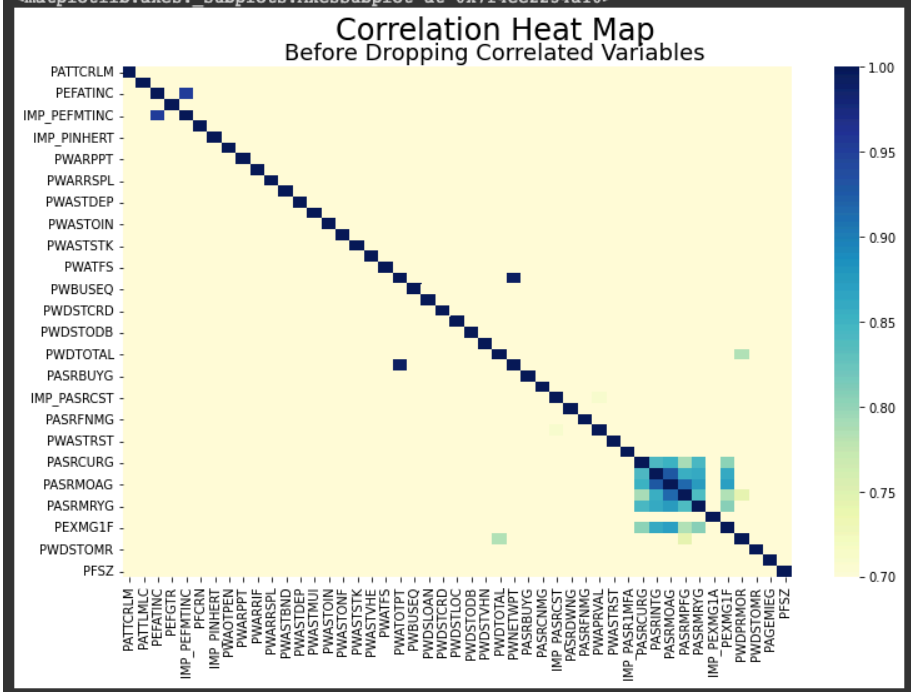
Figure 48. Data Quality Statistics – Integer Data

	Variable 1	Variable 2	Var1-Target Corr	Skew Var 1	Var2-Target Corr	Skew Var 2
1	IMP_PEFMTINC	PEFATINC	0.053063	9.18	0.060128	6.88
2	PWNETWPT	PWATOTPT	0.086922	5.73	0.084051	5.57
3	PWDPRMOR	PWDTOTAL	-0.005281	3.40	-0.004738	3.83
4	PWAPRVAL	IMP_PASRCST	0.086758	6.30	0.064792	4.08
5	PASRINTG	PASRCURG	-0.042689	0.87	-0.030667	1.20
6	PASRMOAG	PASRCURG	-0.034111	0.76	-0.030667	1.20
7	PASRMPFG	PASRCURG	-0.043524	1.07	-0.030667	1.20
8	PASRMRYG	PASRCURG	-0.024265	1.16	-0.030667	1.20
9	PEXMG1F	PASRCURG	-0.016794	1.15	-0.030667	1.20
10	PASRMOAG	PASRINTG	-0.034111	0.76	-0.042689	0.87
11	PASRMPFG	PASRINTG	-0.043524	1.07	-0.042689	0.87
12	PASRMRYG	PASRINTG	-0.024265	1.16	-0.042689	0.87
13	PEXMG1F	PASRINTG	-0.016794	1.15	-0.042689	0.87
14	PASRMPFG	PASRMOAG	-0.043524	1.07	-0.034111	0.76
15	PASRMRYG	PASRMOAG	-0.024265	1.16	-0.034111	0.76
16	PEXMG1F	PASRMOAG	-0.016794	1.15	-0.034111	0.76
17	PASRMRYG	PASRMPFG	-0.024265	1.16	-0.043524	1.07
18	PEXMG1F	PASRMPFG	-0.016794	1.15	-0.043524	1.07
19	PWDPRMOR	PASRMPFG	-0.005281	3.40	-0.043524	1.07
20	PEXMG1F	PASRMRYG	-0.016794	1.15	-0.024265	1.16

Figure 49. Correlation Coefficients Heat Map

```
# Correlation Heat Map
fig=plt.figure(figsize = (12,7.5))
plt.text(x=0.47, y=0.92, s="Correlation Heat Map", fontsize=24, ha="center", transform=fig.transFigure)
plt.text(x=0.47, y=0.89, s= "Before Dropping Correlated Variables", fontsize=18, ha="center", transform=fig.transFigure)
sns.heatmap(correlation_matrix_int_data, vmin=0.7, cmap='YlGnBu')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4ee2234a10>
```



A second correlation test was run after dropping the lower-quality variables (PWATOTPT, IMP_PASRCST, PASRINTG, PASRMOAG, PASRMPFG, IMP_PEFMTINC, PASRMRYG, PWDPRMOR, PEXMG1F). No additional correlation was found. Refer to Figure 50 for a visualization of the correlation heat map after dropping the correlated variables.

As the last step in this section, a simple decision tree classifier was run to identify any potential problems with the data. As it was found, the variable PATTDIF – skipped card payment due to financial difficulties, had a 99.46% correlation to the Target Variable. Looking to avoid further complications, the variable PASRSKP – have skipped or delayed a mortgage payment, was also dropped. Refer to Figure 51 and Figure 52 for visualizations of the two decision tree classifiers created during this section of the analysis.

Figure 50. Correlation Coefficients Heat Map – After Dropping Correlated Variables

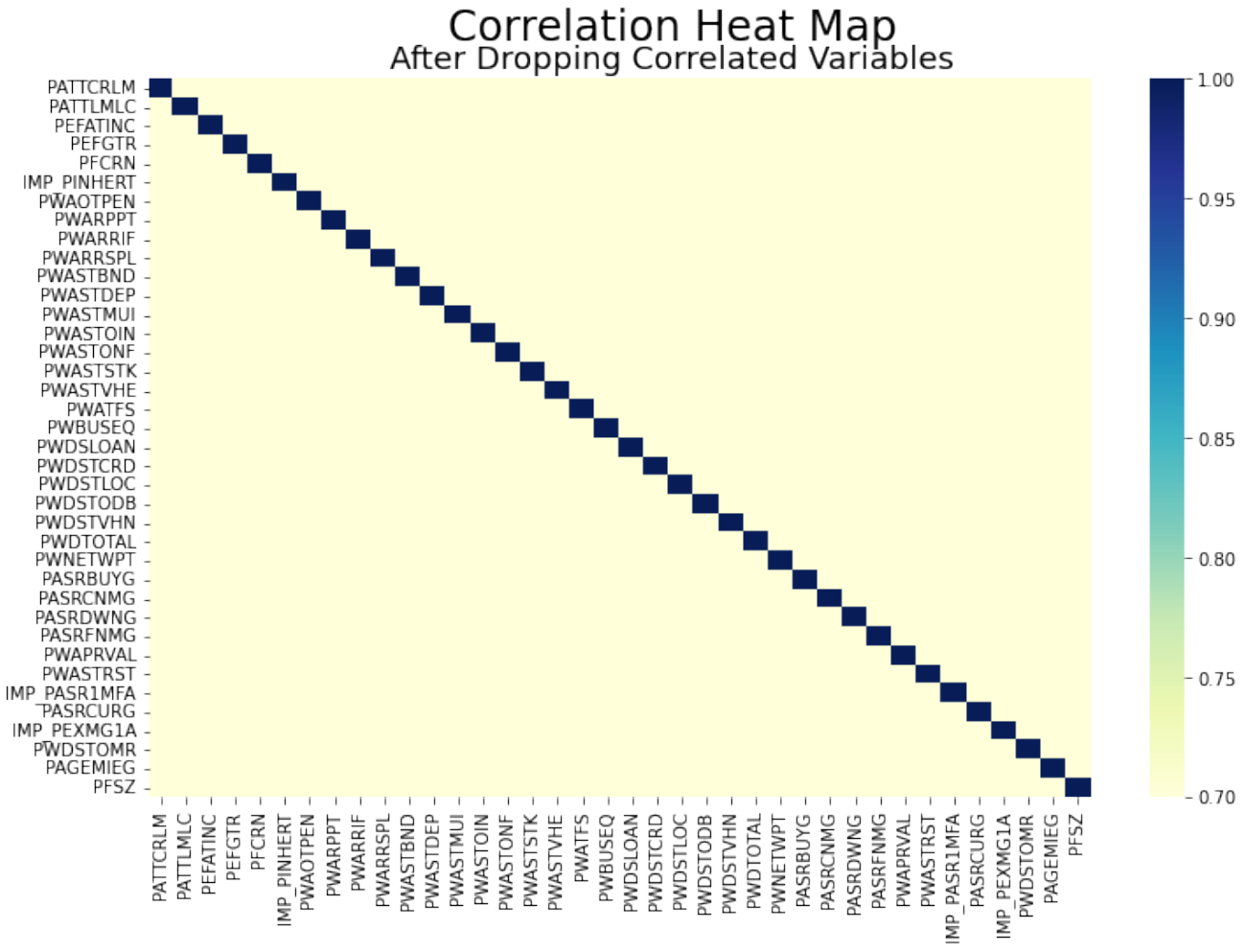
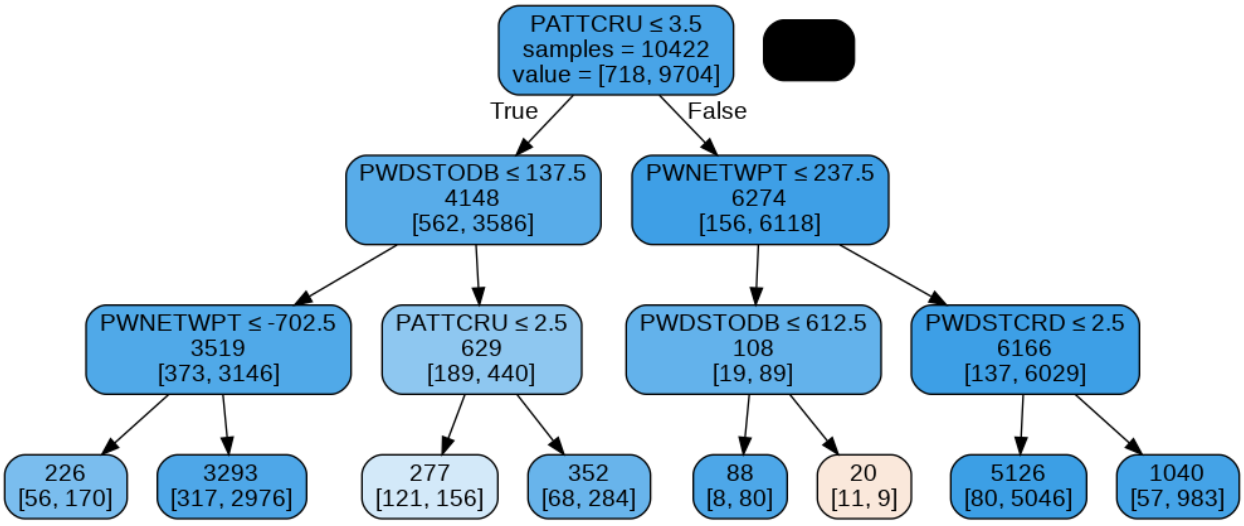


Figure 51. Decision Tree Classifier – First Run

Figure 52. Decision Tree Classifier – After Dropping Variables (Max Depth=3)



Data Quality – Categorical Data: Correlated Variables

The code in Figure 53 was run to 1) create a data frame with the variables' Chi Square test of independence values and their respective p-values 2) find the unique pairs of correlated variables with a p-value greater than 0.05 3) append each variable's correlation coefficient with the target variable. After performing the test of independence, it was found that there are 183 unique pairs of non-independent variables (p-value greater than 0.05). A sample of 10 pairs of variables was printed along with their correlation coefficients with the target variable, PATTSKP. Refer to Figure 54 of a visualization of the code and results.

Figure 54. Decision Tree Classifier – After Dropping Variables (Max Depth=3)

```
# Printing pairs of non-independent categorical variables
if cat_var_cor['Var1'].empty:
    print('There are no additional non-independent categorical variables')
else:
    print(f'There are {len(cat_var_cor)} pairs of non-independent variables')
    print("Sample of (at least) 10 pairs and their respective correlation coefficient with the target variable")
    print(cat_var_cor.head(10))
```

```
There are 183 pairs of non-independent variables
Sample of (at least) 10 pairs and their respective correlation coefficient with the target variable
        Var1       Var2      Var1      Var2
0   D2_PEFGTR    PATTCRC  0.016074 -0.077577
1  D_PWDSLOAN    PATTCRC -0.120907 -0.077577
2     PFSZ65UP    PATTCRC -0.006336 -0.077577
3   D2_PEFGTR    PATTCRR  0.016074  0.086460
4     PFSZ1824    PATTCRR  0.023877  0.086460
5      PBUSIND  D_PATTCRU -0.001613 -0.178527
6  D_PEFATINC   D_PATTCRU -0.056990 -0.178527
7   D1_PEFGTR   D_PATTCRU -0.003157 -0.178527
8  D_PWAOTPEN   D_PATTCRU  0.027378 -0.178527
9   D_PWARPPT   D_PATTCRU  0.050004 -0.178527
```

Figure 53. Data Quality Code – Integer Data

```
# First Test of Independence for Categorical Variables (ChiSqure)
# Creating a ChiSquare dataframe to determine if categorical variables are correlated p-value higher>0.05
cor_cat_data=data[new_cat_data]
factors_paired = [(i,j) for i in cor_cat_data.columns for j in cor_cat_data.columns]
chi2, p_values =[], []
for f in factors_paired:
    if f[0] != f[1]:
        chitest = chi2_contingency(pd.crosstab(cor_cat_data[f[0]], cor_cat_data[f[1]]))
        chi2.append(chitest[0])
        p_values.append(chitest[1])
    else:
        chi2.append(0)
        p_values.append(0)
chi2=np.array(chi2).reshape((cor_cat_data.shape[1],cor_cat_data.shape[1]))
chi2=pd.DataFrame(chi2, index=cor_cat_data.columns, columns=cor_cat_data.columns)
p_values=np.array(p_values).reshape((cor_cat_data.shape[1],cor_cat_data.shape[1]))
p_values=pd.DataFrame(p_values, index=cor_cat_data.columns, columns=cor_cat_data.columns)
# Finding the Unique Pairs of Categorical Correlated Variables
correlation_matrix_cat_data=p_values
lst=[]
cond=correlation_matrix_cat_data>=0.05
for col in new_cat_data:
    rows = list(cond[col][cond[col]==True].index)
    for row in rows:
        lst.append((row,col))
correlated=pd.DataFrame(lst, columns=['Var1','Var2'])
correlated2=correlated[correlated['Var1']!=correlated['Var2']]
cat_correlated_unique=correlated2[~correlated2[['Var1','Var2']].apply(frozenset, axis=1).duplicated()]
correlated_unique.reset_index(inplace=True, drop=True)
cat_correlated_unique.sort_values(by='Var1').inplace=True
cat_correlated_unique
# Finding the correlation variables to the Target Variable
lstv1, lstv2 =[], []
lst1=cat_correlated_unique.Var1.tolist()
lst2=cat_correlated_unique.Var2.tolist()
for i in lst1:
    cor1=data[i].corr(data['PATTSKP'])
    lstv1.append([cor1])
var1cor=pd.DataFrame(lstv1, columns=['Var1'])
for j in lst2:
    cor2=data[j].corr(data['PATTSKP'])
    lstv2.append([cor2])
var2cor=pd.DataFrame(lstv2, columns=['Var2'])
varcor=pd.concat([var1cor,var2cor], axis=1)
varcor.set_index(cat_correlated_unique.index,inplace=True)
cat_var_cor=pd.concat([cat_correlated_unique,varcor],axis=1)
cat_var_cor.reset_index(inplace=True, drop=True)
```

After the first test of independence, 30 variables were dropped. A second test of independence running the same code was performed. 16 additional pairs of correlated variables were found. 8 additional variables were dropped after the second test of independence, refer to Figure 55 –the variables marked with a star sign were dropped after the second test. A third test of independence was performed. No additional non-independent variables at a 95% confidence level were found. Refer to Appendix II for a visualization of the descriptive statistics of the remaining categorical variables.

Figure 55. Data Quality Code – Integer Data

D1_PEFGTR	D_PWASTOIN	PEDUCMIE	D_PEFATINC*
PBUSIND	PGDRMIE	D_PASR1MR	D_PWATFS*
D2_PEFGTR	D_PEFMTINC	PDWTYP	D_PWARPPT*
D_PASRFNMG	D_PAS1MRAG	PPVRES	PATTCRC*
D_PWASTBND	PNBEARG	PASRINT	D_PWDSTVHN*
PASRCON	D_PINHERT	PASR1MR	PATTLCR*
D_PASR1MFA	PFSZ1824	PFSZ0017	D_PWASTDEP*
D_PWAOTPEN	PFSZ4564	PREGION	D_PWDTOTAL*
D_PWASTSTK	PFSZ65UP	PFSZ2544	
D_PASRBUYG	D_PWASTVHE	D_PASRCNMG	

Data Quality – Integer Data: Skewness

Out of the 38 remaining integer variables, 12 had a skew greater than 10 and only three had a skew lower than 1. Refer to Figure 56.

Figure 56.

	std dev	mean	min	max	skew	kurtosis	null values
PWASTSTK	429036.85	39434.71	1.0	16250000.0	25.93	842.01	0
PWASTBND	34303.47	2049.53	0.0	1100000.0	24.82	687.50	0
PWASTOIN	124658.80	13665.23	0.0	4600000.0	24.51	759.31	0
PWASTMUI	442391.02	49345.34	0.0	15750000.0	22.86	687.79	0
PWARRIF	197212.37	29524.19	0.0	7250000.0	21.44	657.24	0
PWDSTODB	16624.55	1806.94	0.0	500000.0	20.04	490.55	0
PWAOTPEN	42875.86	4337.22	0.0	1300000.0	18.11	405.17	0
PWASTRST	529246.43	108198.70	0.0	17250000.0	17.12	452.12	0
IMP_PASR1MFA	6400.61	1006.60	0.0	140000.0	11.47	172.30	0
IMP_PEXMG1A	940.41	367.61	0.0	24000.0	10.74	199.06	0
IMP_PINHERT	197833.18	49067.26	0.0	3900000.0	10.74	158.64	0
PWDSTLOC	50561.14	10853.64	0.0	1100000.0	10.40	150.46	0

The first strategy implemented to solve this problem was to censor records. As it was discussed in an earlier section of this report, there are strong theoretical and practical reasons to

believe that a considerably small group of ultra-wealthy and top-earning households behave differently from the rest of the population. From a statistical perspective, censoring these records is also very helpful because it reduces the severe skewness found in the majority of the data.

Looking to target only the households that have a disproportionately higher wealth that spans multiple dimensions of financial success – Net worth, Assets, Real Estate & Mortgage, Income, Investments Funds, Lines of Credit, Business Equity, and Debt – the censored records must have a wealth that is either 1.5 or 3.0 standard deviations to the right of the mean for *each* of these variables. The variables that were found to have the highest skew were set at 1.5 standard deviations to enforce a more aggressive censoring. 15.18% of the records were dropped. Refer to Figure 57 for a visualization of the # of Standard deviations. Go to Appendix III to find the code that was used to implement the record exclusions.

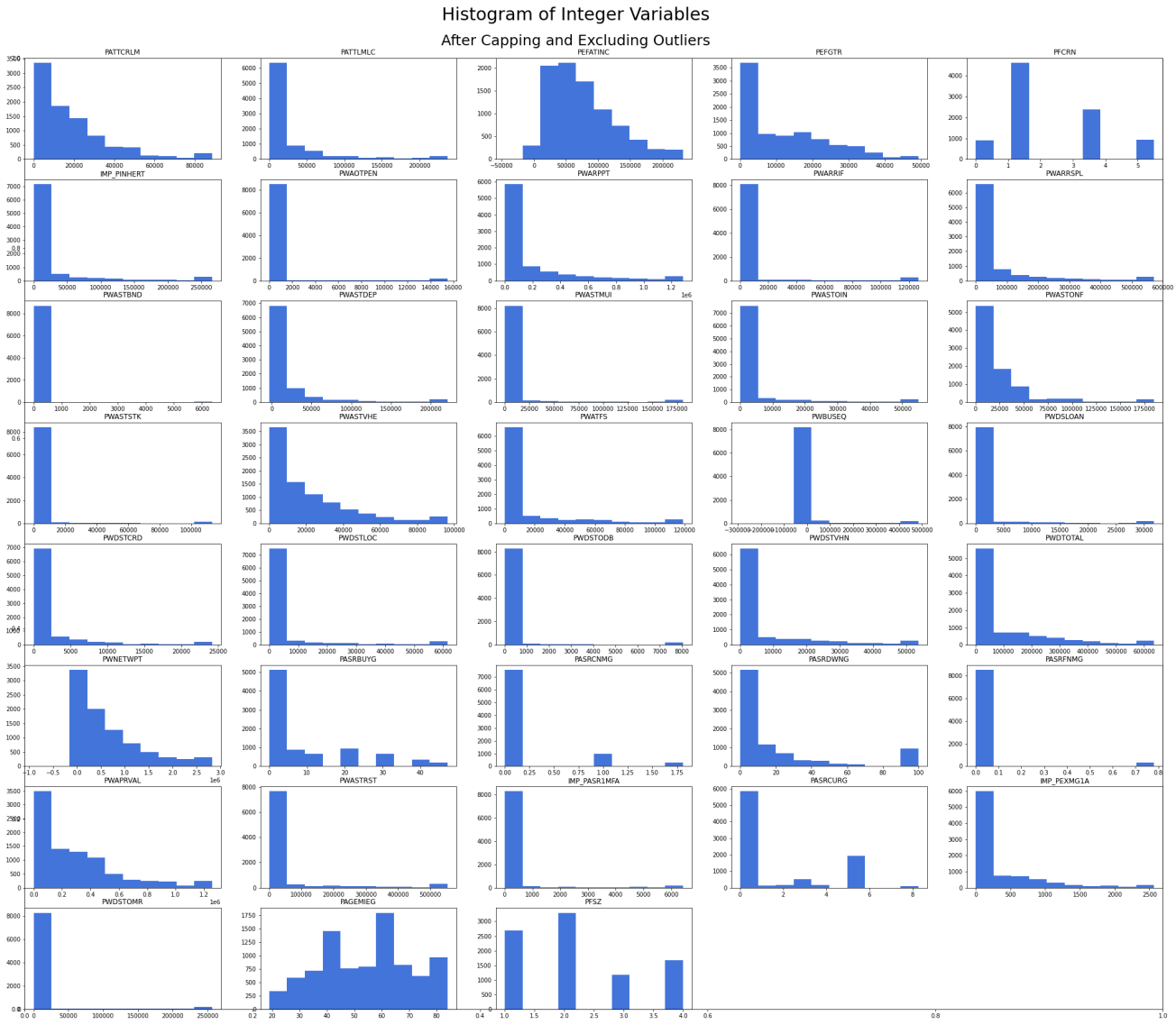
Figure 57.

Variable	Description	# of Standard Deviations
PWNETWPT	Net worth of the family unit (Termination basis)	1.5
PWASTVHE	Vehicles	3.0
PWASTRST	Real estate other than principal residence	3.0
PWAPRVAL	Value of the principal residence	3.0
IMP_PASR1MFA	Amount of additional mortgage payments	3.0
IMP_PINHERT	Total value of inheritances received in 2016 constant dollars	3.0
PEFATINC	After-tax income	3.0
PWASTSTK	Stock and shares, non registered	1.5
PWASTBND	Bonds, non-registered	1.5
PWASTOIN	Other investments or financial assets	1.5
PWASTMUI	Mutual funds, other investment, Income trusts, non-registered	1.5
PWARRIF	Registered retirement income funds (RRIFs)	1.5
PWAOTPEN	Other retirement funds	1.5
PWDSTLOC	Line-of-credit debt (home and other line of credit)	3.0
PWBUSEQ	Equity value of businesses operated by the family unit	3.0
PWDSTODB	Other debt (other loans and other money owed)	1.5

After censoring of the ultra-wealthy household records, a cap and floor at plus/minus 3 standard deviations from the mean was implemented. These strategies proved to be efficient, as

the maximum skew dropped from 25.93 to 9.41. Five variables had a skew lower than 1. Refer to Figure 58 for a visualization of a histogram for each of the integer variables.

Figure 58.



Following the censoring of records, and cap and floor, a logarithmic base 2 transformation and a binning were applied to limit the skew of the data. All the variables with non-negative records and a skew greater than 1 were applied the log base 2 transformation. The rest of the records were binned. Refer to Figure 59 for a visualization of the code utilized to perform the transformations. 17 new variables had an acceptable skew that fell under plus/minus

1. Refer to Appendix IV for a visualization of the variables' descriptive statistics after performing the aforementioned transformations.

Figure 59.

```
#First transformation: LOG base 2 for non-negative variables/Binning for negative variables
negatives=eda_capped_data[data[eda_capped_data.index].min()<0].index.tolist()
lst_log_added_1=[]
lst_bin_added_1=[]
for i in eda_capped_data.index:
    if i not in negatives:
        if eda_capped_data.loc[i,'skew']>1.0:
            data['LOG_1_'+str(i)]=np.log2(data.loc[:,i]+1)
            lst_log_added_1.append(i)
    elif i in negatives:
        data['BIN_1_'+str(i)]=pd.qcut(data.loc[:,i], q=100, duplicates='drop', labels=False)
        lst_bin_added_1.append(i)

added_log=['LOG_1_'+i for i in lst_log_added_1]
added_bin=['BIN_1_'+i for i in lst_bin_added_1]
transformed_data_1=added_log+added_bin
```

A binning was applied to the 17 variables for which the logarithmic base 2 transformation did not bring their skewness to an adequate level. This was an alternative method; the binning was not applied over the log-transformed variables, it was applied to the base variable. Refer to Figure 60 for a visualization of the code used to perform the second transformations. The second binning worked appropriately on two additional variables. Refer to Appendix V for a visualization of the descriptive statistics of the successfully transformed variables.

Figure 60.

```
#Second transformation: Binning for Unsuccessful Transformations 1
negatives=eda_transformed_data_1[data[eda_transformed_data_1.index].min()<0].index.tolist()
var_unsuccessful_transformation_1=[s.replace('LOG_1_','').replace('BIN_1_','') for s in unsuccessful_transformation_1]

lst_bin_added_2=[]
for i in var_unsuccessful_transformation_1:
    if i not in negatives:
        data['BIN_2_'+str(i)]=pd.qcut(data.loc[:,i], q=100, duplicates='drop', labels=False)
        lst_bin_added_2.append(i)
    else:
        pass

lst_bin_added_2=['BIN_2_'+i for i in lst_bin_added_2]
transformed_data_2=lst_bin_added_2
```

The Target Variable: PATTSKP

The target variable for this model is PATTSKP - Have skipped or delayed a non-mortgage payment. This is a binary category, the negative responses were recoded as 0, positives remained as 1. Out of the 8840 remaining records, 646 or 7.30% are positives; 8194 or 92.70% are negatives. The negative responses for all the variables in the dataset were recoded as zeroes.

Model Exploration

Background

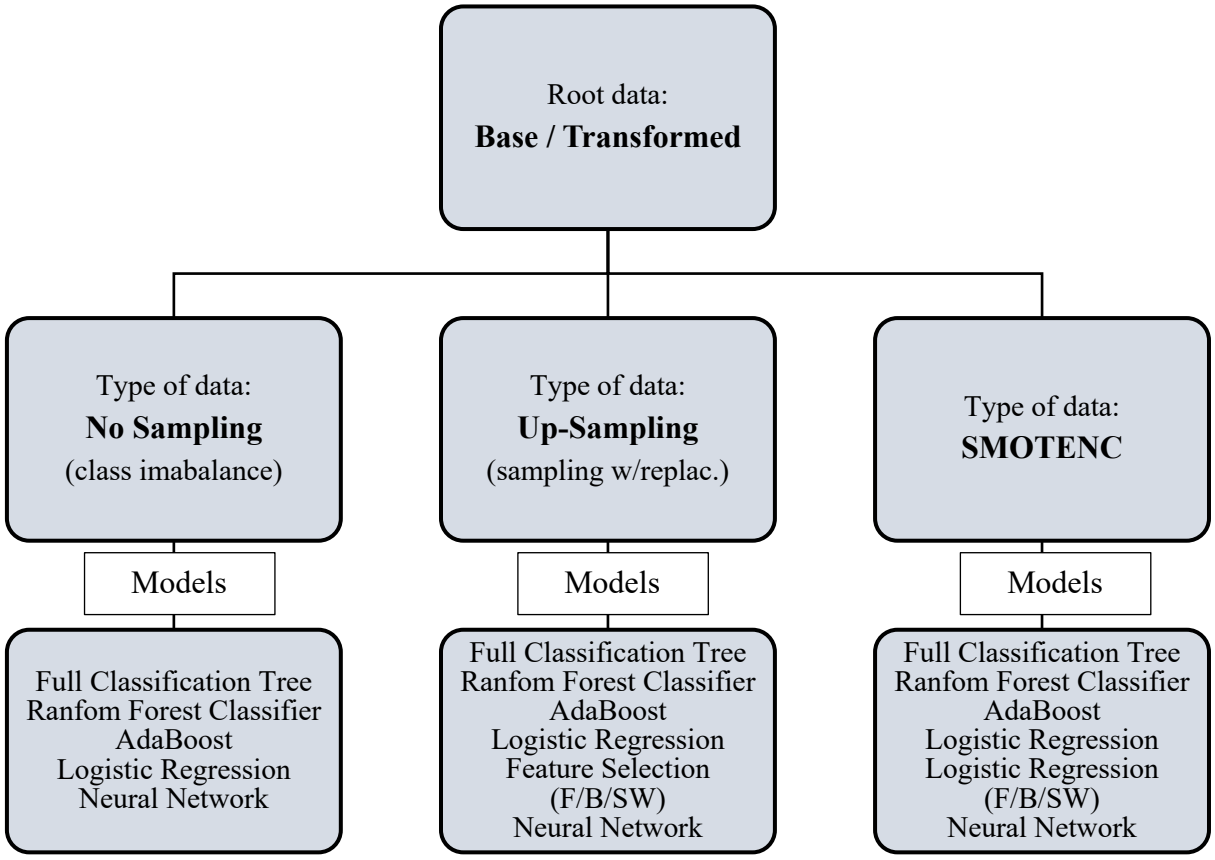
Once the data reconfiguration, data cleaning, and feature engineering processes of this project were completed, there were log-transformed and binned data variables available for modeling. Transformed variables' original data was not discarded, therefore, *base* variables were also available for modeling. The data was split into two main groups: base variables, i.e., data to which no transformations were applied, and transformed variables i.e., data to which binning or log base 2 transformations were applied.

Shortly after running one of the first models, a base data Random Forest Classifier, it was detected that the model was doing a very good job at predicting the majority – negative – class, but was completely overlooking the minority – positive – class. This was a clear indication that there was a serious class imbalance problem that had to be resolved.

Two different methods were implemented to solve the class imbalance: up-sampling (random sampling with replacement) and SMOTENC. This way, class-imbalance (no sampling), random sampling with replacement (up-sampling), and SMOTENC sets of data were now available for the base data. Three additional sets of data – that included transformed variables instead of the base variables– were available as well, making a total of 6 different datasets; 3 for the base data and 3 for the transformed data. Refer to Figure 61 for a visualization of the

datasets. Figure 61 also lists the models run using each set of data (Decision Trees, Logistic Regressions, and Neural Networks).

Figure 61.



All the models in this project used the same code, only slightly modified to make reference to the appropriate datasets. For example, the code used to run a decision tree using base data was also used to run the same tree using the transformed data. Therefore, only the names of the datasets, models, and new tables or figures were changed out of precaution. The *base–no sampling data Full Classification tree* makes reference to the *base_no_samp_train_X* and *base_no_samp_train_y* datasets created during the training and validation data partitioning step. The *trans–no sampling data Full Classification tree* makes reference to its corresponding training X and y datasets, but instead of the prefix *base*, the prefix *trans* was used during the

training and validation data partitioning of the transformed data, which means those datasets were named *trans_no_samp_train_X* and *trans_no_samp_train_y*. Refer to Figure 62 for a visualization of the decision tree classifier example code. Also, go to Appendix VI to consult the name of all the different dataset names used for modeling. 42 different models were built, 21 for the base and transformed data, each.

Figure 62.

```
# Full Classification Tree - BASE-NO_SAMP
fullClassTree_b_ns = DecisionTreeClassifier(random_state=1).fit(base_no_samp_train_X,base_no_samp_train_y)
acc_fct_b_ns=classificationSummary(base_no_samp_valid_y,fullClassTree_b_ns.predict(base_no_samp_valid_X))
# Full Classification Tree - TRANS-NO_SAMP
fullClassTree_t_ns = DecisionTreeClassifier(random_state=1).fit(trans_no_samp_train_X,trans_no_samp_train_y)
acc_fct_t_ns=classificationSummary(trans_no_samp_valid_y,fullClassTree_t_ns.predict(trans_no_samp_valid_X))
```

Having established that the base and transformed data models were run using the same parameters – and therefore the same code – only the transformed data models will be illustrated and discussed during this section of the present report. Illustrations of the code for the base data models will be included in the Appendix section.

Class Imbalance – No Sampled Data

The class imbalance – no sampled data – consists of 72 independent variables with a total of 8840 records. All the models created discussed during this section were built using imbalanced data for the target variable; 646 records missed a payment while 8194 records did not. Refer to Figure 63 for a visualization of the code used to create the train and validation datasets for this section. A 50-50 split was made for the training and validation data.

Full Classification Tree. A Full Classification Tree was built as baseline model. A Random State was set to ensure reproducibility of results. A confusion matrix was printed in addition to the validation AUCROC and training accuracy. Refer to Figure 64.

Figure 63.

```
# Data Partitioning and Variable Declaration
trans_no_samp_X = trans_data.drop(columns=['PATTSKP'])
trans_no_samp_y = trans_data['PATTSKP']
trans_no_samp_train_X, trans_no_samp_valid_X, trans_no_samp_train_y, trans_no_samp_valid_y = train_test_split(trans_no_samp_X,trans_no_samp_y,
                                                                                                              test_size=0.5, random_state=1)
print('PATTSKP - Value Counts\n    Yes: {}\n    No: {}'.format(data.PATTSKP.value_counts()[1],data.PATTSKP.value_counts()[0]))
```

```
PATTSKP - Value Counts
    Yes: 646
    No: 8194
```

```
trans_no_samp_X.shape
```

```
(8840, 72)
```

```
trans_no_samp_y.shape
```

```
(8840,)
```

Figure 64.

```
# Full Classification Tree - TRANS-NO_SAMP
fullClassTree_t_ns = DecisionTreeClassifier(random_state=1).fit(trans_no_samp_train_X,trans_no_samp_train_y)
acc_fct_t_ns=classificationSummary(trans_no_samp_valid_y,fullClassTree_t_ns.predict(trans_no_samp_valid_X))
# AUCROC
prob_y = fullClassTree_t_ns.predict_proba(trans_no_samp_valid_X)
prob_y = [p[1] for p in prob_y]
roc_fct_t_ns=roc_auc_score(trans_no_samp_valid_y,prob_y)
fullClassTree_t_ns_acc_train=fullClassTree_t_ns.score(trans_no_samp_train_X, trans_no_samp_train_y)
print("\nAUCROC: {:.4f}\nTraining Accuracy: {:.2f}".format(roc_fct_t_ns,fullClassTree_t_ns_acc_train))
```

Random Forest Classifier Tree. A Random Forest Classifier with Random State was run. As it was found, increasing the number of estimators did improve accuracy; therefore, the model was set to 750 number of estimators (n_estimators). The model was limited to a maximal depth (max_depth) of 3 to prevent overfitting. A confusion matrix was printed in addition to the validation AUCROC and training accuracy. A Feature Importance plot was also printed. Refer to Figure 65.

Figure 65.

```
# Random Forest Classifier - TRANS-NO_SAMP
rfc_t_ns = RandomForestClassifier(random_state=123, n_estimators=750, max_depth=3).fit(trans_no_samp_train_X, trans_no_samp_train_y)
acc_rfc_t_ns=classificationSummary(trans_no_samp_valid_y, rfc_t_ns.predict(trans_no_samp_valid_X))
# AUROC
prob_y = rfc_t_ns.predict_proba(trans_no_samp_valid_X)
prob_y = [p[1] for p in prob_y]
roc_rfc_t_ns=roc_auc_score(trans_no_samp_valid_y,prob_y)
rfc_t_ns_acc_train=rfc_t_ns.score(trans_no_samp_train_X, trans_no_samp_train_y)
print("\nRandom Forest Classifier (Trans-No_Samp) AUCROC: {:.4f}\nTraining Accuracy: {:.4f}\n".format(roc_rfc_t_ns,rfc_t_ns_acc_train))
# Feature Importance
importance= rfc_t_ns.feature_importances_
df= pd.DataFrame({'feature': trans_no_samp_train_X.columns, 'importance': np.round(importance,3)})
# Feature Importance Plot
selected_imp=df.sort_values('importance', ascending=False)[:12]
dist, ax=plt.subplots(figsize=(8,6))
ax1=sns.barplot(x = selected_imp['importance'], y = selected_imp['feature'])
ax1=plt.ylabel('Features')
dist.suptitle("Feature Importance")
ax1=plt.title("Random Forest Classifier - TRANS-No_Samp")
```

AdaBoost Decision Tree Classifier. An AdaBoost – Decision Tree Classifier was run. The Decision Tree Classifier was given a random state and was limited to a maximal depth of 3 to prevent overfitting. Looking to maximize its accuracy the AdaBoost was set to 750 number of estimators (n_estimators) and was also given a random state to ensure reproducibility. A confusion matrix was printed in addition to the validation AUCROC and training accuracy. Additionally, a Feature Importance plot was printed. Refer to Figure 66.

Figure 66.

```
# AdaBoost Decision Tree Classifier - TRANS-NO_SAMP
ada_t_ns=AdaBoostClassifier(DecisionTreeClassifier(random_state=123, max_depth=3),n_estimators=750, random_state=1).fit(trans_no_samp_train_X, trans_no_samp_train_y)
acc_ada_t_ns=classificationSummary(trans_no_samp_valid_y, ada_t_ns.predict(trans_no_samp_valid_X))
# AUROC
prob_y = ada_t_ns.predict_proba(trans_no_samp_valid_X)
prob_y = [p[1] for p in prob_y]
roc_ada_t_ns=roc_auc_score(trans_no_samp_valid_y, prob_y)
ada_t_ns_acc_train=ada_t_ns.score(trans_no_samp_train_X, trans_no_samp_train_y)
print("\nAdaBoost Decision Tree Classifier (Trans-No_Samp) AUCROC: {:.4f}\nTraining Accuracy: {:.4f}\n".format(roc_ada_t_ns,ada_t_ns_acc_train))
# Feature Importance
importance= ada_t_ns.feature_importances_
df= pd.DataFrame({'feature': trans_no_samp_train_X.columns, 'importance': np.round(importance,3)})
selected_imp=df.sort_values('importance', ascending=False)[:10]
# Feature Importance Plot
dist, ax=plt.subplots(figsize=(8,6))
ax1=sns.barplot(x = selected_imp['importance'], y = selected_imp['feature'])
ax1=plt.ylabel('Features')
dist.suptitle("Feature Importance")
ax1=plt.title("AdaBoost Decision Tree Classifier - Trans-No_Samp")
```

Logistic Regression. Looking to maximize its accuracy, the Logistic Regression was set to 10,000 maximal iterations (max_iter), it was given a 'balanced' class weight (class_weight) to better account for the class imbalance in the data. The 'liblinear' solver generated the best results. A confusion matrix was printed in addition to the validation AUCROC and training accuracy. Additionally, a Feature Importance plot was printed. Refer to Figure 67.

Figure 67.

```
# Logistic Regression - TRANS-No_Samp
LogReg_t_ns = LogisticRegression(max_iter=10000, class_weight='balanced', solver='liblinear').fit(trans_no_samp_train_X, trans_no_samp_train_y)
acc_log_t_ns=classificationSummary(trans_no_samp_valid_y, LogReg_t_ns.predict(trans_no_samp_valid_X))
# AUROC
prob_y = LogReg_t_ns.predict_proba(trans_no_samp_valid_X)
prob_y = [p[1] for p in prob_y]
ROC_LOG_T_NS=roc_auc_score(trans_no_samp_valid_y, prob_y)
LogRegACC_train=LogReg_t_ns.score(trans_no_samp_train_X, trans_no_samp_train_y)
print("\nLogistic Regression (Trans-No_Samp) AUCROC: {:.4f}\nTraining Accuracy: {:.4f}\n".format(ROC_LOG_T_NS,LogRegACC_train))
#Shap
explainer = shap.Explainer(LogReg_t_ns, trans_no_samp_train_X, feature_names=trans_no_samp_train_X.columns.tolist())
shap_values = explainer(trans_no_samp_train_X)
shap.plots.beeswarm(shap_values)
```

Neural Network. As a first step, the data was normalized using the sklearn StandardScaler and clipped at -5, 5 to eliminate any remaining outliers. Taking into account that the target variable of this model has class imbalance, a weight initialization using the target data bias on a factor of the log of the positive instances divided by the negative instances was set, refer to Figure 68.

Figure 68.

```
# Data Normalization for NN
# Splitting data with no previous transformations into train-test-validation
train_df_t_ns, val_df_t_ns = train_test_split(trans_data, test_size=0.5)
# Create np arrays of labels and features.
train_labels_t_ns = np.array(train_df_t_ns.pop('PATTSKP'))
bool_train_labels_t_ns = train_labels_t_ns != 0
val_labels_t_ns = np.array(val_df_t_ns.pop('PATTSKP'))
train_features_t_ns = np.array(train_df_t_ns)
val_features_t_ns = np.array(val_df_t_ns)
# Normalizing data
scaler = StandardScaler()
train_features_t_ns = scaler.fit_transform(train_features_t_ns)
val_features_t_ns = scaler.transform(val_features_t_ns)
# Eliminating outliers
train_features_t_ns = np.clip(train_features_t_ns, -5, 5)
val_features_t_ns = np.clip(val_features_t_ns, -5, 5)
# Defining positive and negative target responses - will be used to set the weights and bias
pos_df_t_ns = pd.DataFrame(train_features_t_ns[ bool_train_labels_t_ns], columns=train_df_t_ns.columns)
neg_df_t_ns = pd.DataFrame(train_features_t_ns[~bool_train_labels_t_ns], columns=train_df_t_ns.columns)
neg_t_ns, pos_t_ns = np.bincount(data['PATTSKP'])
total_t_ns = neg_t_ns + pos_t_ns
# Defining the initial bias of the model - avoiding the model only 'learns' the bias during first epochs there's class
initial_bias_t_ns = np.log([pos_t_ns/neg_t_ns])
print('Initial bias:', round(initial_bias_t_ns[0],2))
# Scaling by total/2 helps keep the loss to a similar magnitude.
weight_for_0_t_ns = (1/neg_t_ns)*(total_t_ns/2.0)
weight_for_1_t_ns = (1/pos_t_ns)*(total_t_ns/2.0)
class_weight_t_ns = {0: weight_for_0_t_ns, 1: weight_for_1_t_ns}
print('Weight for class 0: {:.2f}'.format(weight_for_0_t_ns))
print('Weight for class 1: {:.2f}\n'.format(weight_for_1_t_ns))
```

One densely connected layer with 12 neurons,'relu' activation, and dropout of 0.5 – to prevent overfitting – were added. Since this is a classification problem, a 1 neuron 'sigmoid' activation function was added too. The model was compiled; the 'Adam' optimizer with a learning rate of 1e-3 and Binary Cross Entropy were set. The model was given 500 epochs to train on a batch size of 2000 to allow it to find enough positive instances from which it could learn on each iteration. An early stopping was placed; it was set to monitor the validation precision with a patience of 15. Best weights were restored with verbose, refer to Figure 69.

Figure 69.

```
                                                    # Function: Model
# Construct the Model
nn_t_ns = keras.Sequential([
    keras.layers.Dense(12, activation='relu',input_shape=(train_features_t_ns.shape[-1],)),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(1, activation='sigmoid')])
# Compile Model
nn_t_ns.compile(optimizer=keras.optimizers.Adam(learning_rate=1e-3),loss=keras.losses.BinaryCrossentropy(),metrics=METRICS)
# Defining Additional Parameters
EPOCHS = 500
BATCH_SIZE = 2000
early_stopping_t_ns = tf.keras.callbacks.EarlyStopping(monitor='val_prc',verbose=1,patience=15,mode='max',restore_best_weights=True)
# Fit the model
nn_t_ns.load_weights(initial_weights_t_ns)
nn_t_ns_history = nn_t_ns.fit(
    train_features_t_ns,
    train_labels_t_ns,
    batch_size=BATCH_SIZE,
    epochs=EPOCHS,
    verbose=0,
    callbacks=[early_stopping_t_ns],
    validation_data=(val_features_t_ns, val_labels_t_ns),
    class_weight=class_weight_t_ns)
# Evaluate the model
train_predictions_weighted_t_ns = nn_t_ns.predict(train_features_t_ns, batch_size=BATCH_SIZE)
valid_predictions_weighted_t_ns = nn_t_ns.predict(val_features_t_ns, batch_size=BATCH_SIZE)
nn_t_ns_results = nn_t_ns.evaluate(val_features_t_ns, val_labels_t_ns, batch_size=BATCH_SIZE, verbose=0)
print('\n\nResults:\n    TN - FN: {:.0f} / {:.0f}\n    FP - TP: {:.0f} / {:.0f}\n\n    Validation Accuracy: {:.4}\n    Validation AUC
```

The model was evaluated with training and validation data to look for signs of overfitting. Accuracy metrics such as accuracy, AUC_ROC, and Recall were printed using the validation data. A simple confusion matrix was also built. Lastly, visuals of the Loss Improvement – Accounting vs Not Accounting for Bias, and training and validation Accuracy, AUC, Recall and Loss were printed, refer to Figure 70.

Figure 70.

```
# Visuals
metrics = ['accuracy', 'auc','recall','loss']
fig, ax=plt.subplots(2,2, figsize=(12,10))
fig.text(x=0.5, y=0.90, s='Neural Network Metrics', fontsize=20, ha='center')
for n, metric in enumerate(metrics):
  name = metric.replace("_"," ").capitalize()
  plt.subplot(2,2,n+1)
  plt.plot(nn_t_ns_history.epoch, nn_t_ns_history.history[metric], color="blue", label='Train')
  plt.plot(nn_t_ns_history.epoch, nn_t_ns_history.history['val_'+metric],color="red", linestyle="--", label='Val')
  plt.xlabel('Epoch')
  plt.ylabel(name)
  if metric == 'loss':
    plt.ylim([0, plt.ylim()[1]])
  elif metric == 'auc':
    plt.ylim([0.5,1])
  elif metric == 'accuracy':
    plt.ylim([0.5,1])
  else:
    plt.ylim([0,1])
  plt.legend();
```

Random Sampling with Replacement – Up-Sampled Data

The next group of models will be run on random sampling with replacement (up-sampled) data, refer to Figure 71. The following steps were followed to complete the sample with replacement:

1. The trans_data (data with transformed variables) was divided into training and validation to prevent any data leakage.

   Note: if the up-sampling was performed before the data partitioning, there is an extremely high probability of severe data leakage because the positive-instance records found in the training and validation datasets would be repetitions of the very few original positive instances. By partitioning the data after the up-sampling, it is very difficult to prevent repetitions of the same positive instances from going into the train and validation datasets. This situation would undermine the effectiveness and reliability of the models as they would be severely overfitting and their recall estimates would be extremely inflated.

Figure 71.

```
# Train-test split data
trans_resamp_train, trans_resamp_valid = train_test_split(trans_data, test_size=0.5, random_state=1)
# Identify positives and negatives
data_major_train = trans_resamp_train[data.PATTSKP==0]
data_minor_train = trans_resamp_train[data.PATTSKP==1]
data_major_valid = trans_resamp_valid[data.PATTSKP==0]
data_minor_valid = trans_resamp_valid[data.PATTSKP==1]
# Upsampling the minor class (positives)
data_minor_train_resampled = resample(data_minor_train, replace=True, n_samples=4119, random_state=1)
data_minor_valid_resampled = resample(data_minor_valid, replace=True, n_samples=4075, random_state=1)
# Merging the minor and major class datasets
data_resampled_train = pd.concat([data_major_train, data_minor_train_resampled])
data_resampled_valid = pd.concat([data_major_valid, data_minor_valid_resampled])
# Value counts
print('Train - Value Counts PATTSKP:\n    Negatives: {}\n    Positives: {}\n'.format(data_resampled_valid.PATTSKP.value_counts()[0],
                                                                                      data_resampled_valid.PATTSKP.value_counts()[1]))
print('Valid - Value Counts PATTSKP:\n    Negatives: {}\n    Positives: {}\n'.format(data_resampled_train.PATTSKP.value_counts()[0],
                                                                                      data_resampled_train.PATTSKP.value_counts()[1]))

# Defining labels and features
trans_resamp_valid_y = data_resampled_valid.PATTSKP
trans_resamp_valid_X = data_resampled_valid.drop(columns=['PATTSKP'], axis=1)
trans_resamp_train_y = data_resampled_train.PATTSKP
trans_resamp_train_X = data_resampled_train.drop(columns=['PATTSKP'], axis=1)
```

```
Train - Value Counts PATTSKP:
    Negatives: 4075
    Positives: 4075

Valid - Value Counts PATTSKP:
    Negatives: 4119
    Positives: 4119
```

2. The positive and negative instances for the training and validation datasets were separated and counted.
3. The minority class (positives) for the training and validation were up-sampled by random sampling with replacement to have the same count as negatives in each dataset.
4. The majority class and up-sampled minority class datasets were concatenated.
5. The data was divided into features and labels – there is no need to perform additional data partitioning because this step was already performed. Actually, partitioning a second time would create data leakage.

Full Classification Tree. A Full Classification Tree was run using the same parameters used for the class imbalance data. Refer to Figure 72 for a visualization of the code.

Figure 72.

```
# Full Classification Tree - TRANS-RESAMP
fullClassTree_t_rs = DecisionTreeClassifier(random_state=1).fit(trans_resamp_train_X,trans_resamp_train_y)
acc_fct_t_rs=classificationSummary(trans_resamp_valid_y,fullClassTree_t_rs.predict(trans_resamp_valid_X))
# AUCROC
prob_y = fullClassTree_t_rs.predict_proba(trans_resamp_valid_X)
prob_y = [p[1] for p in prob_y]
roc_fct_t_rs=roc_auc_score(trans_resamp_valid_y,prob_y)
fullClassTree_t_rs_acc_train=fullClassTree_t_rs.score(trans_resamp_train_X, trans_resamp_train_y)
print("\nAUCROC: {:.4f}\nFull Classification Tree Training Accuracy: {:.2f}".format(roc_fct_t_rs,fullClassTree_t_rs_acc_train))
```

Random Forest Classifier Tree. A Random Forest Classifier Tree was run using the same parameters used for the class imbalance data. Refer to Figure 73 for a visualization.

Figure 73.

```
# Random Forest Classifier - TRANS-RESAMP
rfc_t_rs = RandomForestClassifier(random_state=123, n_estimators=750, max_depth=3).fit(trans_resamp_train_X, trans_resamp_train_y)
acc_rfc_t_rs=classificationSummary(trans_resamp_valid_y, rfc_t_rs.predict(trans_resamp_valid_X))
# AUROC
prob_y = rfc_t_rs.predict_proba(trans_resamp_valid_X)
prob_y = [p[1] for p in prob_y]
roc_rfc_t_rs=roc_auc_score(trans_resamp_valid_y,prob_y)
rfc_t_rs_acc_train=rfc_t_rs.score(trans_resamp_train_X, trans_resamp_train_y)
print("\nRandom Forest Classifier (Trans-Resamp) AUCROC: {:.4f}\nTraining Accuracy: {:.4f}\n".format(roc_rfc_t_rs,rfc_t_rs_acc_train))
# Feature Importance
importance= rfc_t_rs.feature_importances_
df= pd.DataFrame({'feature': trans_resamp_train_X.columns, 'importance': np.round(importance,3)})
# Feature Importance Plot
selected_imp=df.sort_values('importance', ascending=False)[:12]
dist, ax=plt.subplots(figsize=(8,6))
ax1=sns.barplot(x = selected_imp['importance'], y = selected_imp['feature'])
ax1=plt.ylabel('Features')
dist.suptitle("Feature Importance")
ax1=plt.title("Random Forest Classifier - Trans-Resamp")
```

AdaBoost Decision Tree Classifier. An AdaBoost – Decision Tree Classifier was run using the same parameters used for the class imbalance data. Refer to Figure 74 for a visualization.

Figure 74.

```
# AdaBoost Decision Tree Classifier - TRANS-RESAMP
ada_t_rs=AdaBoostClassifier(DecisionTreeClassifier(random_state=123, max_depth=3),n_estimators=750, random_state=1).fit(trans_resamp_train_X, trans_resamp_train_y)
acc_ada_t_rs=classificationSummary(trans_resamp_valid_y, ada_t_rs.predict(trans_resamp_valid_X))
# AUROC
prob_y = ada_t_rs.predict_proba(trans_resamp_valid_X)
prob_y = [p[1] for p in prob_y]
roc_ada_t_rs=roc_auc_score(trans_resamp_valid_y, prob_y)
ada_t_rs_acc_train=ada_t_rs.score(trans_resamp_train_X, trans_resamp_train_y)
print("\nAdaBoost Decision Tree Classifier (Trans-Resamp) AUCROC: {:.4f}\nTraining Accuracy: {:.4f}\n".format(roc_ada_t_rs,ada_t_rs_acc_train))
# Feature Importance
importance= ada_t_rs.feature_importances_
df= pd.DataFrame({'feature': trans_resamp_train_X.columns, 'importance': np.round(importance,3)})
selected_imp=df.sort_values('importance', ascending=False)[:10]
# Feature Importance Plot
dist, ax=plt.subplots(figsize=(8,6))
ax1=sns.barplot(x = selected_imp['importance'], y = selected_imp['feature'])
ax1=plt.ylabel('Features')
dist.suptitle("Feature Importance")
ax1=plt.title("AdaBoost Decision Tree Classifier - Trans-Resamp")
```

Logistic Regression. A Logistic Regression was run using the same parameters used for the class imbalance data. Refer to Figure 75 for a visualization.

Figure 75.

```
# Logistic Regression - TRANS-Resamp
LogReg_t_rs = LogisticRegression(max_iter=10000, solver='liblinear', class_weight='balanced').fit(trans_resamp_train_X, trans_resamp_train_y)
acc_log_t_rs=classificationSummary(trans_resamp_valid_y, LogReg_t_rs.predict(trans_resamp_valid_X))
# AUROC
prob_y = LogReg_t_rs.predict_proba(trans_resamp_valid_X)
prob_y = [p[1] for p in prob_y]
roc_log_t_rs=roc_auc_score(trans_resamp_valid_y, prob_y)
LogReg_t_rs_acc_train=LogReg_t_rs.score(trans_resamp_train_X, trans_resamp_train_y)
print("\nLogistic Regression (Trans-Resamp) AUCROC: {:.4f}\nTraining Accuracy: {:.4f}\n".format(roc_log_t_rs,LogReg_t_rs_acc_train))
# SHAP Plot
explainer = shap.Explainer(LogReg_t_rs, trans_resamp_train_X, feature_names=trans_resamp_train_X.columns.tolist())
shap_values = explainer(trans_resamp_train_X)
shap.plots.beeswarm(shap_values)
```

Logistic Regression – Forward Inclusion Feature Selection. A Sequential Feature Selection (SFS) – Logistic Regression model from mlx.tend.feature_selection was run. The parameters for the Logistic Regression were: maximal iterations (max_iter) 1000 – to allow the model to converge, solver ‘liblinear’ – performed better, class weight (class_weight) ‘balanced’. Parameters for the Sequential Feature Selection include: forward ‘true’, floating ‘false’, verbose ‘0’, k_features ‘15’, and scoring ‘roc_auc’.

A Logistic Regression using the same parameters from the previous model was run on the 15 best features returned by the SFS. A confusion matrix was printed in addition to the validation AUCROC and training accuracy. Additionally, a Feature Importance plot was printed. Refer to Figure 76 for a visualization.

Figure 76.

```
# Sequential Feature Selection: Forward
sfs_fwd_trans_resamp = SFS(LogisticRegression(max_iter=1000, solver='liblinear',
                                              class_weight='balanced'),
                           forward=True, floating=False, verbose=0, k_features=15, scoring='roc_auc', n_jobs=-1).fit(trans_resamp_train_X, trans_resamp_train_y)
# Creating a Results DataFrame
fwd_sel_trans_resamp=pd.DataFrame.from_dict(sfs_fwd_trans_resamp.get_metric_dict())
# Getting a list of the best 15 features
fwd_sel_var_trans_resamp=list(fwd_sel_trans_resamp.loc['feature_names',15])
LogReg_t_rs = LogisticRegression(max_iter=1000000, class_weight='balanced', solver='liblinear').fit(trans_resamp_train_X[fwd_sel_var_trans_resamp], trans_resamp_train_y)
acc_log_t_rs=classificationSummary(trans_resamp_valid_y, LogReg_t_rs.predict(trans_resamp_valid_X[fwd_sel_var_trans_resamp]))
# AUROC
prob_y = LogReg_t_rs.predict_proba(trans_resamp_valid_X[fwd_sel_var_trans_resamp])
prob_y = [p[1] for p in prob_y]
roc_log_t_rs=roc_auc_score(trans_resamp_valid_y, prob_y)
# Valid and Training Accuracy
LogReg_t_rs_acc_valid=LogReg_t_rs.score(trans_resamp_valid_X[fwd_sel_var_trans_resamp], trans_resamp_valid_y)
LogReg_t_rs_acc_train=LogReg_t_rs.score(trans_resamp_train_X[fwd_sel_var_trans_resamp], trans_resamp_train_y)
print("\nEvaluation of the 15 best features:\n    Logistic Regression (Trans-Resamp) VALID AUCROC: {:.4f}\n    Training Accuracy: {:.4f}\n    Validation Accuracy: {:.4f}\n"
# SHAP importances
explainer = shap.Explainer(LogReg_t_rs, trans_resamp_train_X[fwd_sel_var_trans_resamp], feature_names=trans_resamp_train_X[fwd_sel_var_trans_resamp].columns.tolist())
shap_values = explainer(trans_resamp_train_X[fwd_sel_var_trans_resamp])
shap.plots.beeswarm(shap_values)
```

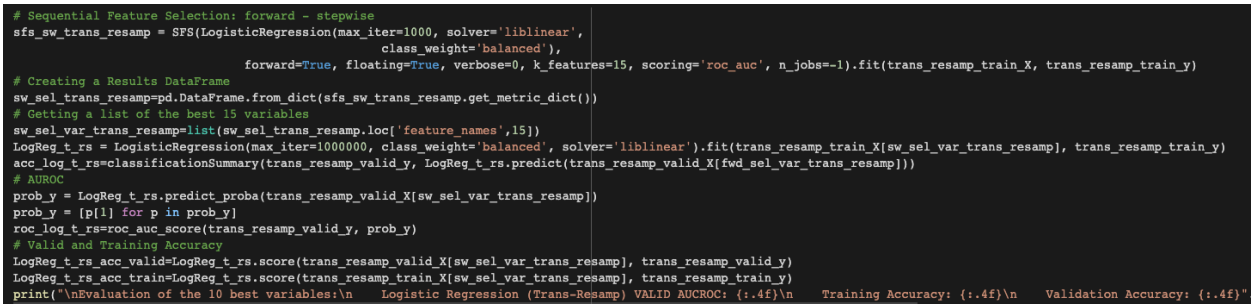
Logistic Regression – Backward Exclusion Feature Selection. A SFS model was run using the same parameters used in the Forward Exclusion Model, with the only difference being that a backward selection was made (Forward= ‘False’); however, the model proved to be considerably much more computationally expensive and was decided that it would be discarded. Refer to Figure 77 for a visualization.

Figure 77.

```
# Sequential Feature Selection: Backward
#sfs_fwd_trans_resamp = SFS(LogisticRegression(max_iter=1000, solver='liblinear',
#                                              class_weight='balanced'),
#                           forward=False, verbose=2, k_features=15, scoring='roc_auc', n_jobs=-1).fit(trans_resamp_train_X, trans_resamp_train_y)
# Creating a Results DataFrame
#bwd_sel_trans_resamp=pd.DataFrame.from_dict(sfs_fwd_trans_resamp.get_metric_dict())
# Getting a list of the best 15 variables
#bwd_sel_var_trans_resamp=list(bwd_sel_trans_resamp.loc['feature_names',15])
#LogReg_t_rs = LogisticRegression(max_iter=1000000, class_weight='balanced', solver='liblinear').fit(trans_resamp_train_X[bwd_sel_var_trans_resamp], trans_resamp_train_y)
# AUROC
#prob_y = LogReg_t_rs.predict_proba(trans_resamp_valid_X[bwd_sel_var_trans_resamp])
#prob_y = [p[1] for p in prob_y]
#roc_log_t_rs=roc_auc_score(trans_resamp_valid_y, prob_y)
# Valid and Training Accuracy
#LogReg_t_rs_acc_valid=LogReg_t_rs.score(trans_resamp_valid_X[bwd_sel_var_trans_resamp], trans_resamp_valid_y)
#LogReg_t_rs_acc_train=LogReg_t_rs.score(trans_resamp_train_X[bwd_sel_var_trans_resamp], trans_resamp_train_y)
#print("\nEvaluation of the 15 best variables:\n    Logistic Regression (Trans-Resamp) VALID AUCROC: {:.4f}\n    Training Accuracy: {:.4f}\n    Validation Accuracy: {:.4f}"
```

Logistic Regression – Stepwise/Forward Feature Selection. A SFS model was run using the same parameters used in the Forward Exclusion Model, with the only difference being that this model was stepwise (Floating= ‘True’). Refer to Figure 78 for a visualization.

Figure 78.



```
# Sequential Feature Selection: forward - stepwise
sfs_sw_trans_resamp = SFS(LogisticRegression(max_iter=1000, solver='liblinear',
                                             class_weight='balanced'),
                          forward=True, floating=True, verbose=0, k_features=15, scoring='roc_auc', n_jobs=-1).fit(trans_resamp_train_X, trans_resamp_train_y)
# Creating a Results DataFrame
sw_sel_trans_resamp=pd.DataFrame.from_dict(sfs_sw_trans_resamp.get_metric_dict())
# Getting a list of the best 15 variables
sw_sel_var_trans_resamp=list(sw_sel_trans_resamp.loc['feature_names',15])
LogReg_t_rs = LogisticRegression(max_iter=1000000, class_weight='balanced', solver='liblinear').fit(trans_resamp_train_X[sw_sel_var_trans_resamp], trans_resamp_train_y)
acc_log_t_rs=classificationSummary(trans_resamp_valid_y, LogReg_t_rs.predict(trans_resamp_valid_X[fwd_sel_var_trans_resamp]))
# AUROC
prob_y = LogReg_t_rs.predict_proba(trans_resamp_valid_X[sw_sel_var_trans_resamp])
prob_y = [p[1] for p in prob_y]
roc_log_t_rs=roc_auc_score(trans_resamp_valid_y, prob_y)
# Valid and Training Accuracy
LogReg_t_rs_acc_valid=LogReg_t_rs.score(trans_resamp_valid_X[sw_sel_var_trans_resamp], trans_resamp_valid_y)
LogReg_t_rs_acc_train=LogReg_t_rs.score(trans_resamp_train_X[sw_sel_var_trans_resamp], trans_resamp_train_y)
print("\nEvaluation of the 10 best variables:\n    Logistic Regression (Trans-Resamp) VALID AUCROC: {:.4f}\n    Training Accuracy: {:.4f}\n    Validation Accuracy: {:.4f}"
```

Neural Network. A Neural Network was run using the same parameters used in for the class imbalance data. As it pertains to the bias, since the up-sampled data has equal weights for both classes (1), it will automatically set to 0. The model was still pre-trained to learn the initial weights.

SMOTENC – Up-Sampled Data

The SMOTENC was used as a second method to tackle the class imbalance problem. Instead of taking a resample approach to increase the number of instances for the minority class, new 'synthetic' instances were created. In theory, this method should prove to be more efficient as 'new' variance is added to the model. The execution of the SMOTENC code was the same as the random sampling with replacement.

1. Divide the data into training and validation to prevent any data leakage.
2. Define labels and targets (necessary for SMOTENC).
3. Run the SMOTENC code to up-sample the minority class for the training data and validation data separately.

SMOTENC Data: Models. All the models that were run for the random sampling with replacement data were also run for the SMOTENC data. The same parameters were implemented on the models as exactly the same logic applies. For simplicity and efficiency, those models will not be further discussed in this section.

Model Recommendation

Overview of the Models

Even though all the models built for this project essentially used the same data, their performance varied by a high margin. The models built using the non-up-sampled data performed consistently poorer when compared to the rest of the models. Comparing the random-sampling-with-replacement data models to the SMOTENC's, the synthetic data models performed better across the board. Lastly, looking at the SMOTENC base versus transformed data models, the transformed data models performed better, refer to Figure 79 for a full list of models' accuracy scores. The three best performing models are the SMOTENC-transformed data Random Forest Classifier, AdaBoost Tree, and Neural Network.

Figure 79.

		Base			Trans		
		AUCROC	Recall	Accuracy	AUCROC	Recall	Accuracy
NO-SAMP	Full Tree	55.72%	18.55%	87.08%	55.79%	18.55%	87.72%
	RFC	79.60%	0.00%	92.19%	79.63%	0.00%	92.19%
	AdaBoost	68.53%	8.69%	91.52%	70.01%	7.25%	91.36%
	Logistic	76.12%	70.72%	67.17%	80.66%	72.67%	73.39%
	NN	80.22%	64.26%	77.65%	80.62%	65.05%	78.73%
UP-SAMP	Full Tree	54.63%	15.58%	54.63%	54.38%	14.80%	54.38%
	RFC	78.80%	75.80%	72.09%	78.71%	75.88%	72.11%
	AdaBoost	70.85%	9.99%	70.85%	68.09%	8.27%	53.03%
	Logistic	75.05%	68.25%	68.92%	79.61%	73.84%	72.86%
	SW	78.79%	72.49%	72.02%	80.89%	74.09%	73.75%
	BWD	--	--	--	--	--	--
	FW	78.53%	72.61%	72.04%	80.89%	74.09%	75.76%
	NN	79.74%	67.39%	71.99%	79.33%	72.75%	71.23%
SYN	Full Tree	78.99%	68.49%	78.99%	83.36%	75.95%	83.36%
	RFC	89.30%	85.42%	80.61%	91.85%	91.60%	83.45%
	AdaBoost	95.50%	78.94%	88.34%	96.15%	84.49%	91.28%
	Logistic	75.80%	68.79%	69.01%	89.44%	79.31%	80.83%
	SW	84.35%	78.50%	77.15%	89.19%	80.39%	83.89%
	BWD	--	--	--	--	--	--
	FW	84.04%	80.66%	77.80%	89.19%	80.39%	83.89%
	NN	89.61%	72.39%	80.07%	95.54%	84.27%	88.69%

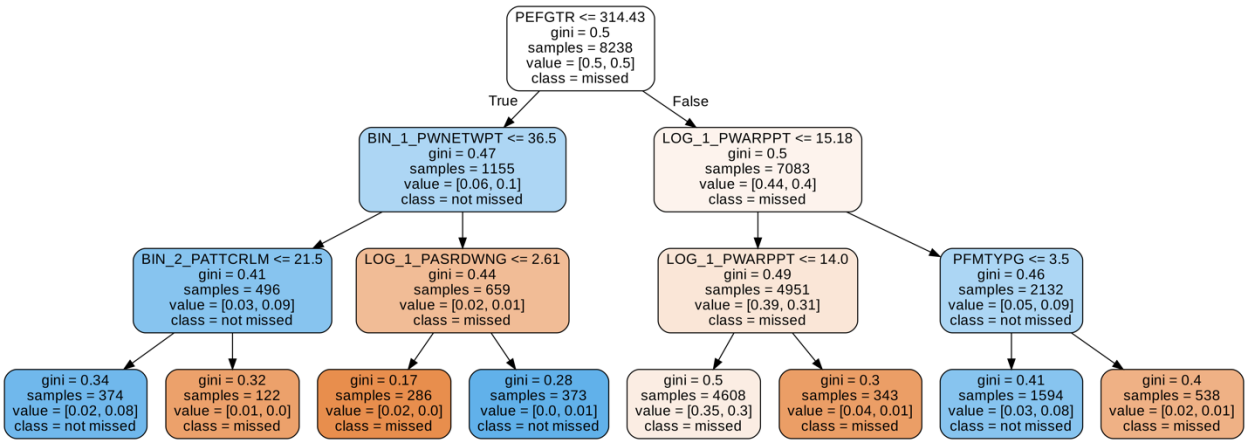
Note: the accuracy of these models is limited to predicting the probability of skipping or delaying a non-mortgage payment, excluding (approximately) 15% of the wealthiest and higher-earning Canadian households.

AdaBoost Classifier: SMOTENC – Transformed Data

This AdaBoost Classifier was found to be the best performing model. As it has been discussed in a previous section of this report, this is a classification project, and due to the specific business case at hand, correctly classifying positive instances has been prioritized. Therefore, the highest-ranking measures of success for this project are the validation's AUC-ROC and Recall as both of these metrics quantify the probability of correctly classifying positive instances of the target variable.

Model Statistics and Theory. The AdaBoost Classifier has been considered the best model due to its 96.15% AUC-ROC, the highest among all the models built for this project. Figure 80 shows a plot of the tree. Refer to Figure 80 for a visualization of the model's main accuracy statistics and its 10-highest importance features.

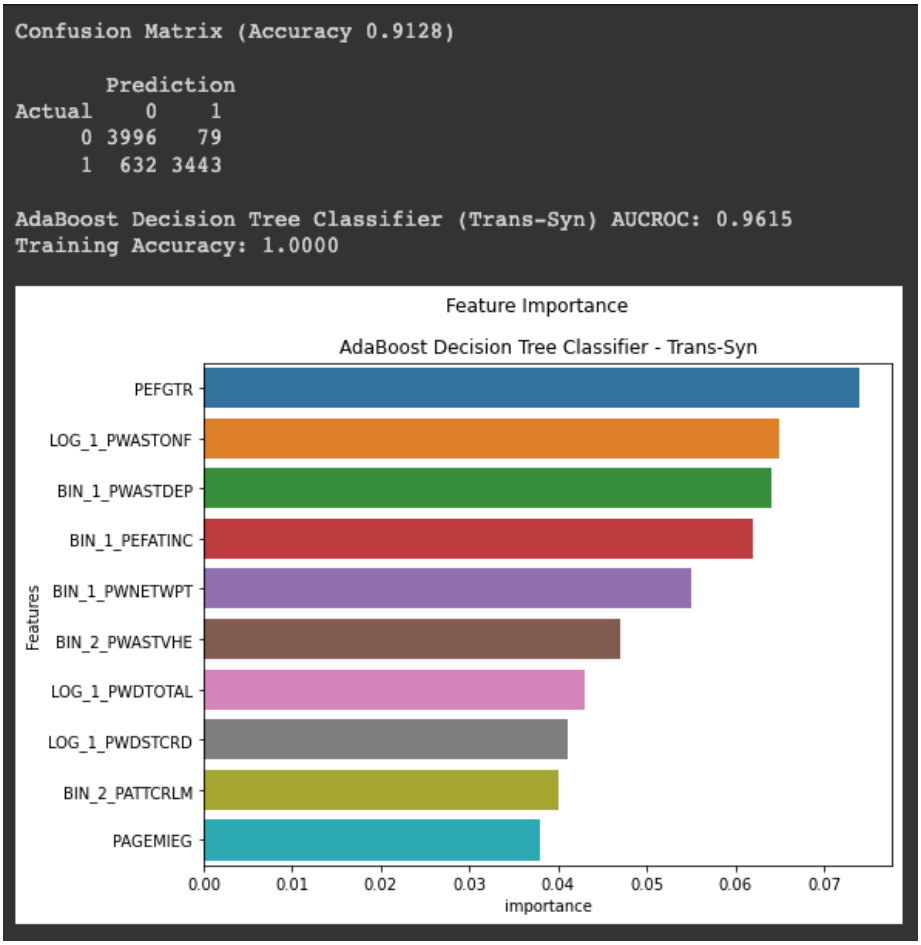
Figure 80.



As it has been found, many of the models built during this project consistently struggled to predict the positive instance of the target variable, which coincidently, carries the highest

practical importance to the business case at hand. It is not unexpected that the AdaBoost was the best performing model given that this is a classifier that focuses on learning the most *difficult* cases, which means, correctly classifying the positive instances of the target variable. Moreover, this model performed considerably better as the class imbalance issue was addressed by synthesizing new instances of the minority class, which meant that the model had more information to learn to predict positive responses to the target variable. Refer to Figure 81.

Figure 81.



In theory, decision trees are not majorly affected by skewed data due to the way they work – decision trees find the best split point based on a given metric of entropy even if there still is data that extends long to the tails of the distribution. However, it is interesting that the

model assigned to the root node a variable that was not transformed and presented a manageable skew immediately after the records corresponding to the ultra-rich households were discarded from the data.

Model Drivers. Other than finding a highly accurate model, the interpretability and insights derived from such a model were categorized as very important factors to the overall success of this project, even before it was started. Although the base form of PEFGTR – Government transfers, federal & provincial, is found to be highly important (it is found in the root node of our best model), painting a complete picture of the households that are the most and least likely to skip or delay a non-mortgage payment is still a somewhat difficult task to accomplish.

Looking to fully address the goals set at beginning of this project, the linear form of the 10-highest importance features, refer to Figure 79, were selected to profile households with the highest and lowest probabilities to skip or delay a payment. Refer to Figure 82 for a visualization of the records with the highest and lowest probabilities to skip or delay a payment.

Figure 82.

HIGH AND LOW PROBABILITY OF MISSING OR SKIPPING A PAYMENT

	Concept	Hi-Inc Hi-Proba	Lo-Inc Hi-Proba	Hi-Inc Lo-Proba	Lo-Inc Lo-Proba	Max	Min	Mean
PEFGTR	Government Transfer	1435.0	18957.0	500.0	325.0	49326.0	0.0	10843.0
PEFATINC	After-Tax Income	152632.0	18957.0	176975.0	46075.0	232120.0	-26000.0	71199.0
PWASTONF	Other Non-Financial Assets	24389.0	365.0	9500.0	5000.0	184793.0	175.0	17702.0
PWASTDEP	Total Money in Banks	1666.0	55.0	4750.0	20000.0	221297.0	-3200.0	12520.0
PATTCRLM	Total Credit Limit (CC)	17806.0	0.0	15000.0	0.0	88534.0	0.0	15228.0
PWASTVHE	Total Debt on Vehicles	39639.0	464.0	82500.0	15500.0	96432.0	0.0	18566.0
PWNETWPT	Net Worth of Family	1031923.0	1812.0	774400.0	500250.0	2831148.0	-910500.0	485644.0
PWDTOTAL	Total Debts of Family	270398.0	0.0	389100.0	5250.0	636230.0	0.0	107091.0
PAGEMIEG	Age of Major Income Earner	49.0	62.0	39.0	69.0	84.0	19.0	50.0
PWDSTCRD	Total CC and Installment Debt	5744.0	0.0	4100.0	0.0	24162.0	0.0	2919.0

Note: it should be clearly stated that the model specification of this AdaBoost Classifier includes non-linear variables. Equal models performed better using the non-linear form of the

same variables; which clearly indicates that the relationships between the features and the target are non-linear. However, a profiling exercise using the linear form of the 10 most important variables was conducted to derive general insights and add interpretability to this project under the understanding that the true relationships are non-linear and the full-specification model should always be implemented to return the best predictions.

Insights. There are various valuable insights that can be derived from creating household profiles based on their probabilities of skipping or delaying a payment. Among the most remarkable elements, it was found that:

1. Preventing bias. The model identifies high-earning and high net-worth families with high probabilities of missing a payment. Also, this model identifies lower-earning households with low-score probabilities of missing a payment.
2. Finding diversity. An extension to point 1 that should not go unsaid, it is encouraging that the model was able to find diverse households at both ends of the spectrum. It should be noted that this model is working with Statistics Canada data that encompasses the totality of the population. And while the ultra-wealthy household records were removed, this model – I believe – effectively captures the diversity of the Canadian population: people with different probabilities have different incomes, assets, ages, number of family members, etc.
3. Understanding relationships between financial metrics. The model seemed to learn, understand, and give priority to the relationship between wealth and debts, and most importantly, it avoided over-prioritizing net worth.

   High net-worth households that are highly leveraged (from a financial perspective, leverage represents the relationship between assets and liabilities) seem to be of interest

for the model. For example, the High Income – High Probability record in Figure 82 has a high Net Worth and After-Tax Income as compared to the maximum and mean values of the population. However, as it was found, the Total Money in Banks for the family is relatively low compared to the various liabilities of the family (Total Debts of Family, Total CC and Installment Debt, and Total Debt on Vehicles). While it is a possibility that the family may be able to easily gain liquidity by selling financial assets, the model may be identifying the relationship of these factors as a potential source of risk.

4. Collateral. It is possible that the model identifies different metrics of wealth to assess the probability of non-payment. For example, there are Low Income - Low Probability households, refer to Figure 82; however, these households have a relatively high bank account balance compared to their liabilities. Low Probability records that don't have high bank balances do have high incomes that offset their liabilities.

Random Forest Classifier: SMOTENC – Transformed Data

The Random Forest Classifier was the second-best performing model. At a 91.85% validation AUC-ROC and 91.60% Recall, this was the best model for purely identifying positive instances. Refer to Figure 83 for a visualization of the tree and to Figure 84 for a visualization of the model's main accuracy statistics and a plot of its 10-highest importance features.

A finding that should not be overlooked is that this model, while also highly accurate, utilized different features to assess the probability of getting a positive response to the target variable. The insights derived from the best performing model could be expanded by analyzing the most important features of this model. An insights table can be found in Figure 85.

Figure 83.

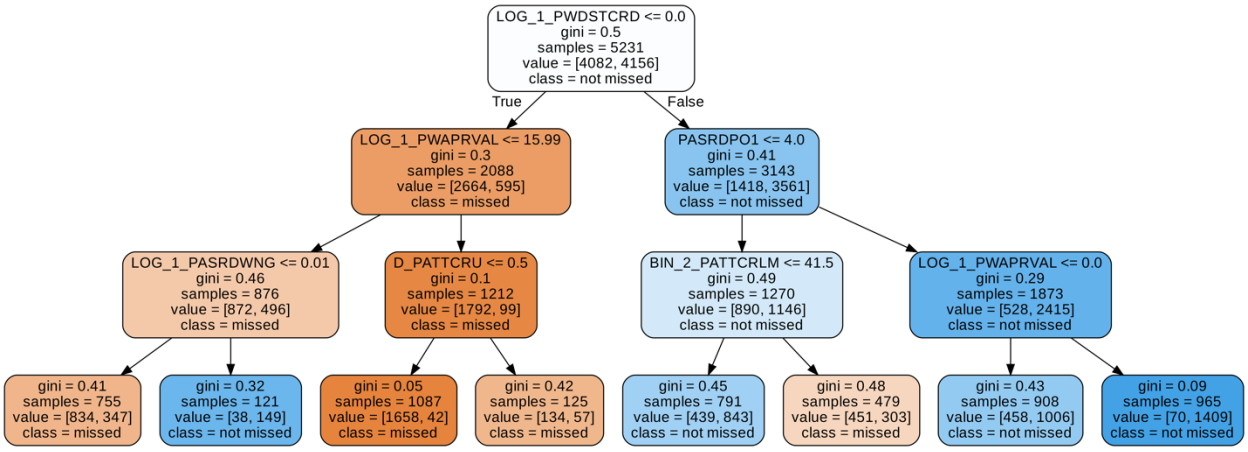


Figure 84.

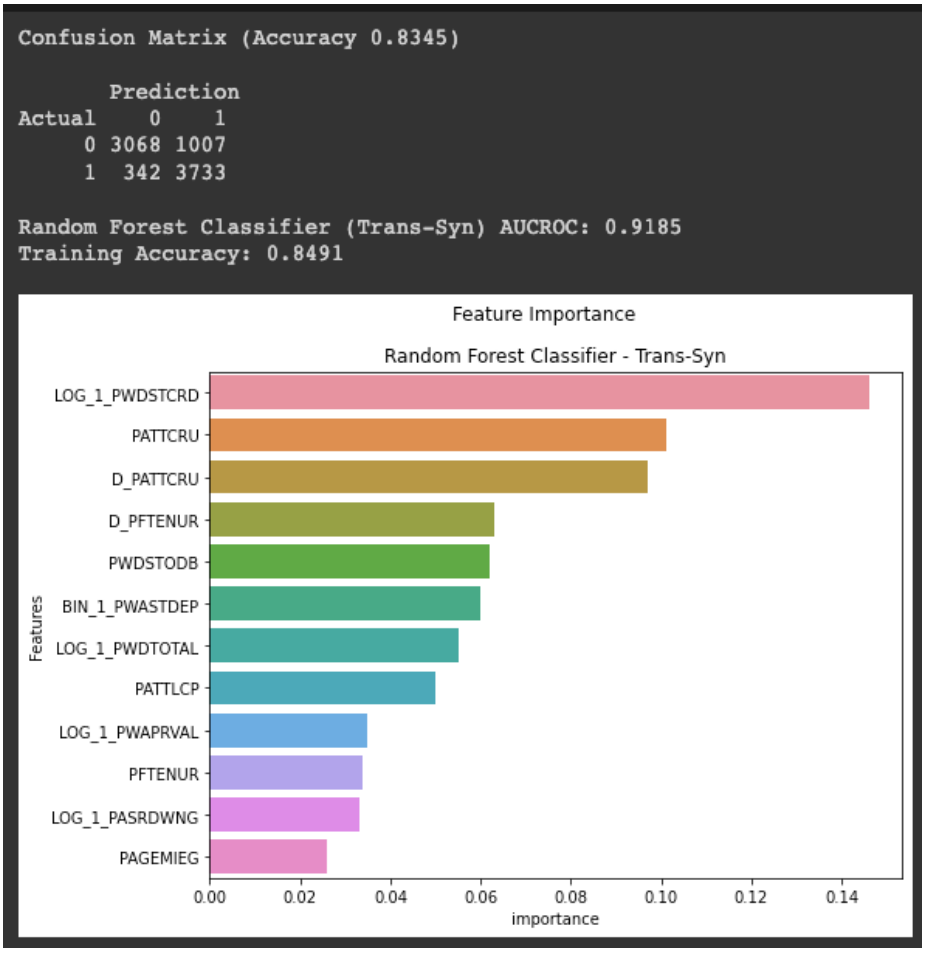


Figure 85.

HIGH AND LOW PROBABILITY OF MISSING OR SKIPPING A PAYMENT

	Concept	Hi-Inc Hi-Proba-	Hi-Inc Hi-Proba	Hi-Inc Lo-Proba	Lo-Inc Lo-Proba	Max	Min	Mean
PEFATINC	After Tax Income	109299.0	65413.0	140375.0	33125.0	232120.0	-26000.0	71199.0
PWDSTCRD	Total CC and Installment Debt	1982.0	1641.0	0.0	0.0	24162.0	0.0	2919.0
PATTCRU	CC Balance Paid on Time?	3.0	3.0	4.0	4.0	NaN	NaN	NaN
D_PATTCRU	Paid the Minimum or Less?	1.0	1.0	0.0	0.0	NaN	NaN	NaN
D_PFTENUR	Owns a Home with No Mortgage?	0.0	0.0	1.0	1.0	NaN	NaN	NaN
PWASTDEP	Total Money in Banks	1511.0	-289.0	45000.0	10000.0	221297.0	-3200.0	12520.0
PWDSTODB	Total of other Debt	0.0	782.0	0.0	0.0	8050.0	0.0	674.0
PWDTOTAL	Total Debts of Family	209731.0	125979.0	0.0	9500.0	636230.0	0.0	107091.0
PATTLCP	Line of Credit Balance Paid on Time?	2.0	0.0	4.0	0.0	NaN	NaN	NaN
PWAPRVAL	Value of Principal Residence	551464.0	107816.0	950000.0	95000.0	1257136.0	0.0	237959.0
PAGEMIEG	Age of Major Income Earner	54.0	37.0	69.0	59.0	84.0	19.0	50.0

Neural Network: SMOTENC – Transformed Data

The SMOTENC-Transformed data's Neural Network was among the top three performing models. Building a neural network that returns consistently solid accuracy metrics on class-imbalanced data was a very difficult task, and even when the overall accuracy of the neural networks built was vastly improved after the class imbalance problem was tackled with SMOTENC; compared to the two best performing models of this project, Recall remained low. Refer to Figure 86 and Figure 87 for visualizations of the main accuracy statistics of this model.

Figure 86.

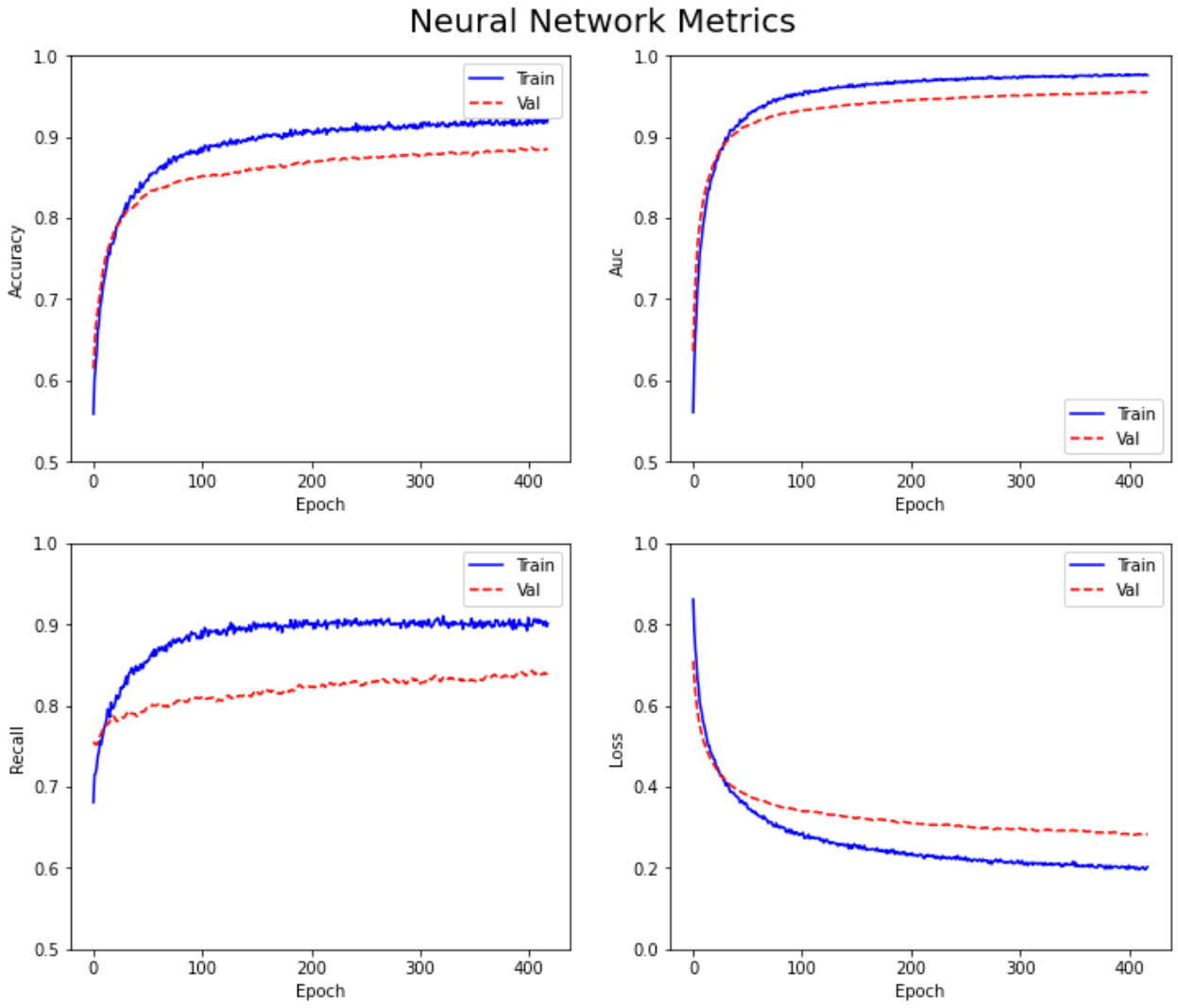
```
Initial bias: 0.0
Weight for class 0: 1.00
Weight for class 1: 1.00

Restoring model weights from the end of the best epoch: 403.
Epoch 418: early stopping

Results:
    TN - FN: 3794 / 641
    FP - TP: 281 / 3434

    Validation Accuracy: 0.8869
    Validation AUC_ROC: 0.9554
    Validation Recall: 0.8427
    Precision: 0.9244
```

Figure 87.



Validation and Governance

Risk of Data Drift and Model Specification

The models in this project were built using the Survey of Financial Security 2019 (SFS-2019) created by Statistics Canada. This data portrays the Canadian population as it existed prior to a ground-breaking event: the COVID-19 pandemic. Since the long-term objective of this project is to understand how the economic shock caused by the COVID-19 pandemic impacted Canadian households' finances – as measured by their ability to make non-mortgage payments –

by the project's very nature, it is expected that the data collected at different points in time i.e., before and after the pandemic, will be different.

Different approaches to capitalize on the unique research opportunity created by the COVID-19 pandemic were explored:

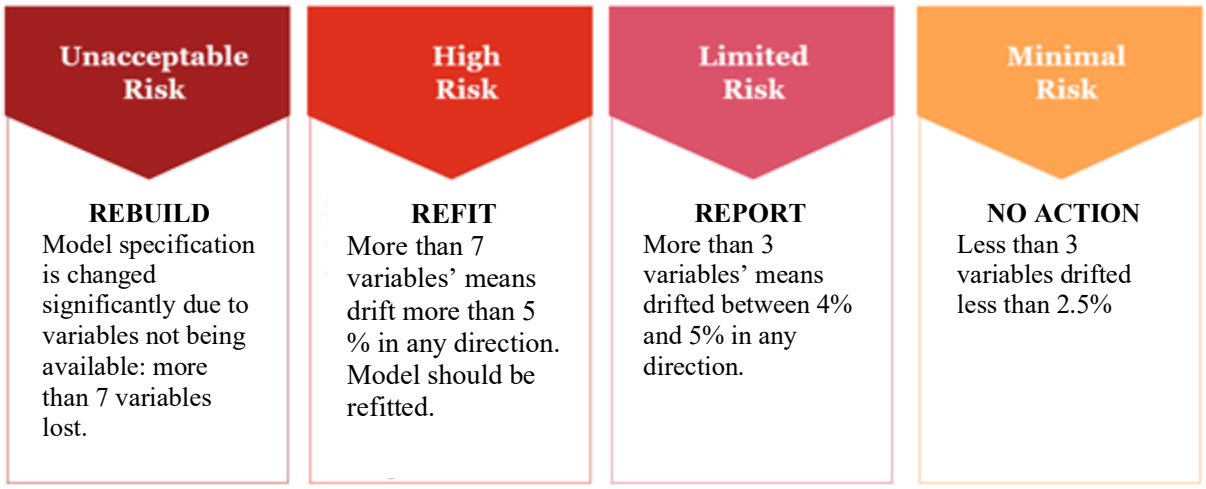
1) build a traditional cross-sectional data model to predict the same variable. A baseline model built with SFS-2019 data could be used to create new insights by utilizing data collected in the future.
2) build panel-data models that look at the same population at two different points in time.
3) building time-series data models that measure the variance of variables over time.

Due to obvious data limitations – no post-COVID-19 data has been collected or released yet – and the complexities of performing panel-data and time-series data analyses, it was decided that cross-sectional model(s) would be built, taking into consideration the possibility of drift. And although we are currently going through a period of high volatility, the potential drivers of change in the SFS-2019's variables are not tracked and do not directly appear in the dataset. Therefore, as long as there is data stationarity, the models built for this project should be able to translate well to different points in time.

An additional point that should be raised is that working with Statistics Canada data provides assurances in regards to the quality standards that will be enforced for data collection over the years. The variables and data treatment of the SFS rarely changes, and when changes do occur, how and why these changes were made is announced to the public. Moreover, it is virtually impossible that missing values would be found because Statistics Canada has established its own imputation methods; if data was missing, it would be because a variable was

not tracked any longer, which again, is highly unlikely. For a visualization of the risk tiering, refer to Figure 88.

Figure 88.



Data Statistics for 10 Most Important Features

The best model for this project was an AdaBoost Classifier built on logarithmic base 2 transformed data and other binned data, up-sampled using SMOTENC to solve class imbalance. A point of relative concern for the model stability over time is that the full model specification includes 73 variables, including the target. Such an extensive model increases the possibility of finding complications as data may become unavailable or drift. Refer back to Figure 85 to consult the statistics for the model's 10 most important variables. A +/- 5% drift around the mean for each variable would be considered acceptable for this project due to the high volatility that is currently found in the market. Variables should be monitored for drift every 6 months.

Missing Data & Caps and Floors

As was explained before, the probability of finding missing data in Statistics Canada data is considered to be quite low. However, if the situation did occur, the most likely approach would be dropping the variable – as the most likely reason to have missing data would be a given

variable was not tracked anymore. If missing values did appear, imputations according to Statistics Canada procedures would ensue in order to maintain consistency across the data.

Figure 89.

Descriptive Statistics - Important Model Components

	Mean	Mean+5%	Mean-5%	Std Dev	Min	Max	1st Qrt	3st Qrt
PEFGTR	10843.0	11385.0	10301.0	10867.0	0.0	49326.0	1350.0	17529.0
PEFATINC	71199.0	74759.0	67639.0	48661.0	-26000.0	232120.0	33986.0	96619.0
PWASTONF	17702.0	18587.0	16817.0	25878.0	175.0	184793.0	3113.0	21000.0
PWASTDEP	12520.0	13146.0	11894.0	31498.0	-3200.0	221297.0	550.0	9157.0
PATTCRLM	15228.0	15990.0	14467.0	17668.0	0.0	88534.0	2512.0	21000.0
PWASTVHE	18566.0	19494.0	17637.0	21393.0	0.0	96432.0	2462.0	27000.0
PWNETWPT	485644.0	509926.0	461361.0	646076.0	-910500.0	2831148.0	19874.0	753463.0
PWDTOTAL	107091.0	112446.0	101736.0	152113.0	0.0	636230.0	1552.0	167000.0
PAGEMIEG	50.0	52.0	47.0	16.0	19.0	84.0	38.0	61.0
PWDSTCRD	2919.0	3065.0	2773.0	4893.0	0.0	24162.0	0.0	3900.0

Cap and Floors at +-3 standard deviations are to be implemented for all data. Data censoring is also required to maintain comparability in results. Refer to Figure 90 for a complete of the parameters used to censor records.

Model Health & Stability

It is important that this model continues to provide accurate results over time as the implications of inaccurate estimations could result in dire consequences for consumers, companies, and other institutions. The potential of loss income by consumers and companies, due to incorrect assessments should not be taken lightly; even potential penalties may stem from subpar model performance.

Initial Model Fit Statistics. This model's initial AUC-ROC score stands at 96.15%; its Recall stands at 84.49%. A maximum of 5% reduction for both accuracy metrics would be accepted. AUC-ROC: 91.34, Recall: 80.27. Refer to Figure 91.

Figure 90.

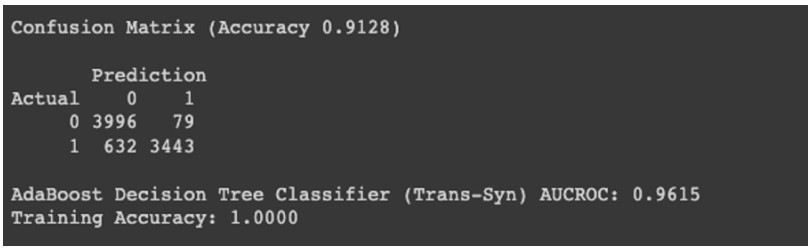


Figure 91.

Variable	# of Standard Deviations
PWNETWPT	1.5
PWASTVHE	3.0
PWASTRST	3.0
PWAPRVAL	3.0
IMP_PASR1MFA	3.0
IMP_PINHERT	3.0
PEFATINC	3.0
PWASTSTK	1.5
PWASTBND	1.5
PWASTOIN	1.5
PWASTMUI	1.5
PWARRIF	1.5
PWAOTPEN	1.5
PWDSTLOC	3.0
PWBUSEQ	3.0
PWDSTODB	1.5

Recommendation: AdaBoost Classifier

Multi-Directional Relationships of Income, Liabilities, and Net Worth

Due to the nature of the variables' multi-directional relationships found by the best model – an AdaBoost Classifier built on logarithmic base 2 transformed and other binned data; up-sampled using SMOTENC to solve class imbalance – it is difficult to provide a clear-cut threshold to determine if a household will skip or delay a non-mortgage payment. Moreover,

providing this type of insight would lead to inaccurate conclusions, businesses' loss of money, and poor customer experience.

More than providing a one-size-fits-all recommendation based on selected variables' threshold, the best recommendation based on the best model's findings is to:

"Assess households' financials based on a combination of factors, always avoiding reaching a conclusion that only considers stand-alone measures of income, wealth, or demographics. Financial measures, such as liquidity and leverage, should play a big role in the assessment of credit worthiness. Being a homeowner with no mortgage payments, as well as payment patterns of other liabilities, like lines of credit and credit cards, are additional powerful indicators."

The Results Limitations: The Ultra Wealthy

The 15% wealthiest and higher-earning Canadians were excluded from the analysis performed in this project. Therefore, the generalization of the findings presented to the excluded portion of the population should be exercised with caution. No additional exclusions were made.

There are certain factors that limit the reach of the findings presented in this project. For example, there is no way to set control measures to resurvey only the population represented in the current data. Since the Canadian population and market conditions will be different in the future, new survey data may be non-stationary. The high-inflation and high interest rate environment currently found around the world will present challenges if comparisons to new models would be drawn in the future.

The Next Steps: Clustering Models and Running Models on New Data

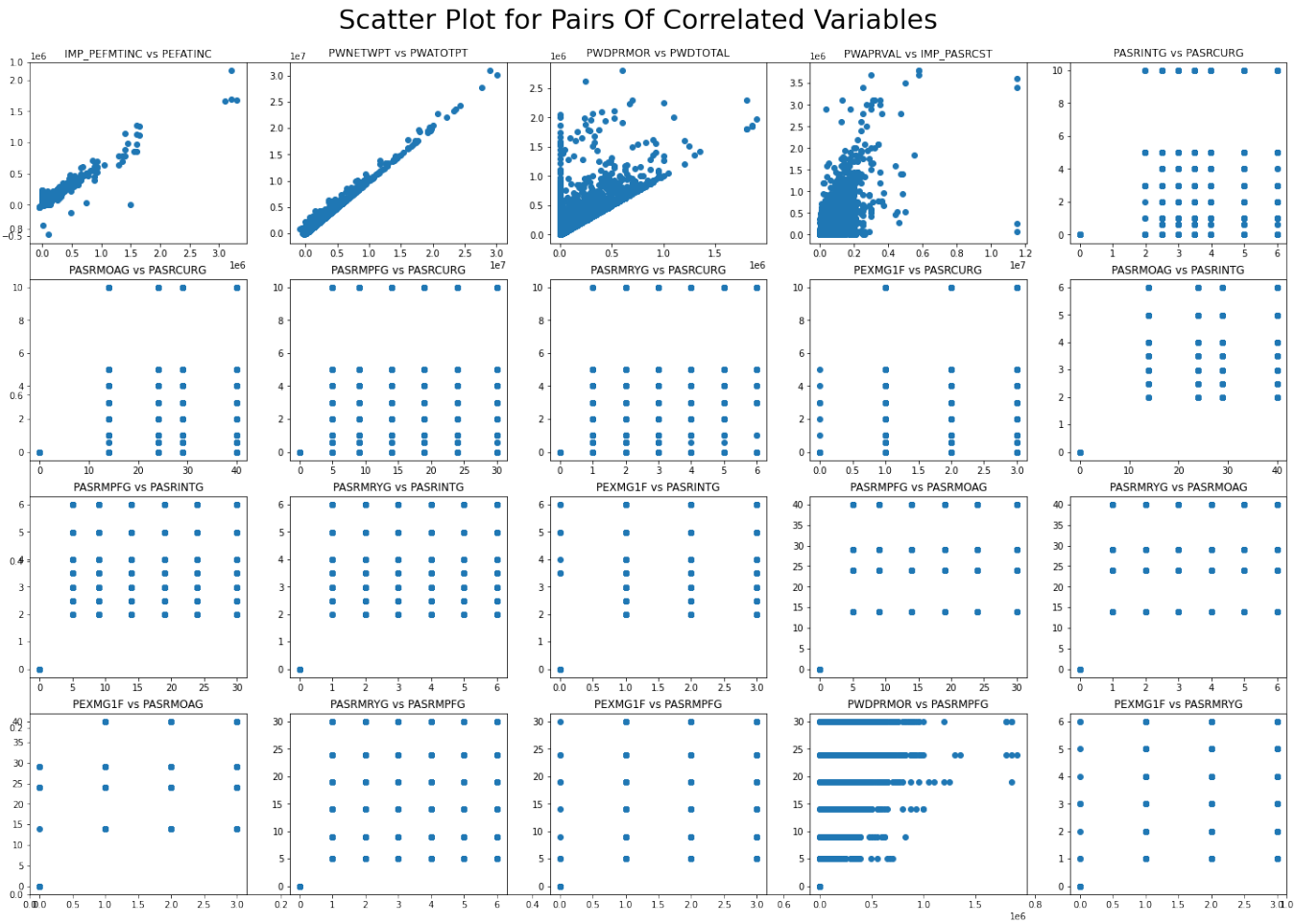
Even with the limitations presented by a rapidly changing Canadian population and high market volatility, the larger intent of this project is to understand how the economic shock caused

by COVID-19 pandemic affected Canadians' financial wellbeing and their ability to meet their financial liabilities.

The findings of this project only represent the first step in the process of understanding the COVID-19 pandemic's impact on Canadians' finances. The next step likely is to perform clustering techniques to identify the characteristics that define specific groups of Canadians – prior to the pandemic. Later these groups would be compared to similar groups of people, but under the new market conditions found a post-pandemic world.

Appendix

Appendix I: Scatterplot of correlated variables



Appendix II: Descriptive Statistics or Remaining – Non-Dependent – Categorical Variables

	median	mode	min	max	null values	% - mode	% - least common value	number of classes
PATTPAYD	2.0	2	1	2	0	97.62	2.38	2
D1_PWNETWPT	0.0	0	0	1	0	95.38	4.62	2
D_PWDSTODB	0.0	0	0	1	0	91.25	8.75	2
PATTCRR	6.0	2	1	6	0	90.92	1.78	3
D_PWATOTPT	0.0	1	0	1	0	89.99	10.01	2
D2_PWNETWPT	0.0	1	0	1	0	89.97	10.03	2
D_PWDSLOAN	0.0	0	0	1	0	89.25	10.75	2
D_PWARRIF	0.0	0	0	1	0	87.45	12.55	2
D_PWASTMUI	0.0	0	0	1	0	86.73	13.27	2
PATTRSP	1.0	2	1	2	0	71.76	28.24	2
PATTRSL	2.0	6	1	6	0	70.40	1.36	3
D_PATTCRU	0.0	0	0	1	0	69.28	30.72	2
PASRRNTG	2.0	6	1	6	0	67.90	2.79	3
D_PFTENUR	0.0	0	0	1	0	64.70	35.30	2
PLFFPTME	1.0	3	1	6	0	61.14	2.67	4
PEFMJSIF	2.0	4	1	7	0	60.78	0.23	7
PASRDPO4	2.0	6	1	6	0	59.93	5.11	3
PASRDPO5	2.0	6	1	6	0	59.36	5.68	3
PLFCHRME	1.0	6	1	6	0	58.99	6.60	3
PATTCRU	4.0	0	0	5	0	57.95	1.29	6
PATTRSH	2.0	6	1	6	0	57.89	13.87	3
PASRDPO3	2.0	6	1	6	0	55.87	9.17	3
PATTRSA	2.0	6	1	6	0	55.18	16.58	3
PLFPDMEG	1.0	6	1	6	0	55.18	10.41	3
D_PWARRSPL	1.0	0	0	1	0	54.98	45.02	2
PATTRSR	2.0	6	1	6	0	53.26	18.50	3
PATTSTIN	2.0	3	1	3	0	49.03	12.74	3
PATTLCP	2.0	0	0	5	0	44.77	0.76	6
PFRSPST	2.0	3	1	3	0	40.59	28.24	3
PRETIRME	2.0	1	1	9	0	37.33	0.65	4
PASRDPO2	2.0	6	1	6	0	36.34	28.71	3
PASRDPO1	2.0	6	1	6	0	36.14	28.90	3
PFTENUR	2.0	3	1	3	0	35.71	28.99	3
PFMTYPG	2.0	1	1	9	0	28.32	9.48	5

Appendix III. Record Censoring

```
#Exclusion of households with the highest wealth (across multiple dimensions)
data=data[
#Net worth
(data['PWNETWPT']<=(data['PWNETWPT'].mean()+((data['PWNETWPT'].std())*1.5))) &
#Assets - Tangible-non-realestate
(data['PWASTVHE']<=(data['PWASTVHE'].mean()+((data['PWASTVHE'].std())*3))) &
#Real estate & mortgage
(data['PWASTRST']<=(data['PWASTRST'].mean()+((data['PWASTRST'].std())*3))) &
(data['PWAPRVAL']<=(data['PWAPRVAL'].mean()+((data['PWAPRVAL'].std())*3))) &
(data['IMP_PASR1MFA']<=(data['IMP_PASR1MFA'].mean()+((data['IMP_PASR1MFA'].std())*3))) &
#Income
(data['IMP_PINHERT']<=(data['IMP_PINHERT'].mean()+((data['IMP_PINHERT'].std())*3)))&
(data['PEFATINC']<=(data['PEFATINC'].mean()+((data['PEFATINC'].std())*3))) &
#Funds
(data['PWASTSTK']<=(data['PWASTSTK'].mean()+((data['PWASTSTK'].std())*1.5)))&
(data['PWASTBND']<=(data['PWASTBND'].mean()+((data['PWASTBND'].std())*1.5)))&
(data['PWASTOIN']<=(data['PWASTOIN'].mean()+((data['PWASTOIN'].std())*1.5))) &
(data['PWASTMUI']<=(data['PWASTMUI'].mean()+((data['PWASTMUI'].std())*1.5))) &
(data['PWARRIF']<=(data['PWARRIF'].mean()+((data['PWARRIF'].std())*1.5))) &
(data['PWAOTPEN']<=(data['PWAOTPEN'].mean()+((data['PWAOTPEN'].std())*1.5))) &
#Lines of credit
(data['PWDSTLOC']<=(data['PWDSTLOC'].mean()+((data['PWDSTLOC'].std())*3))) &
#Business Equity
(data['PWBUSEQ']<=(data['PWBUSEQ'].mean()+((data['PWBUSEQ'].std())*3))) &
#Debt
(data['PWDSTODB']<=(data['PWDSTODB'].mean()+((data['PWDSTODB'].std())*1.5)))]
```

Appendix IV. Data Statistics – Transformed Ata

	std dev	mean	min	max	skew	kurtosis	null values
LOG_1_PWASTBND	1.58	0.23	0.0	12.635826	7.03	48.32	0
LOG_1_PASRFNMG	0.16	0.03	0.0	0.832868	4.74	20.46	0
LOG_1_PWAOTPEN	2.76	0.63	0.0	13.918646	4.21	16.05	0
LOG_1_IMP_PASR1MFA	2.91	0.79	0.0	12.650195	3.47	10.23	0
LOG_1_PWDSTOMR	4.26	1.15	0.0	17.970332	3.47	10.09	0
LOG_1_PWDSTODB	3.10	0.90	0.0	12.975025	3.24	8.76	0
LOG_1_PWASTSTK	3.63	2.07	1.0	16.791192	3.23	8.77	0
BIN_1_PWBUSEQ	2.37	0.79	0.0	11.000000	3.20	9.23	0
LOG_1_PWASTMUI	4.57	1.50	0.0	17.466760	2.79	5.99	0
LOG_1_PWARRIF	4.71	1.58	0.0	16.951576	2.69	5.36	0
LOG_1_PWDSLOAN	4.30	1.55	0.0	14.955854	2.45	4.11	0
LOG_1_PASRCNMG	0.40	0.16	0.0	1.491163	2.20	3.24	0
LOG_1_PWASTRST	6.58	3.07	0.0	19.077959	1.72	1.03	0
LOG_1_PWDSTLOC	5.54	2.73	0.0	15.909203	1.58	0.59	0
LOG_1_PWASTOIN	5.43	2.74	0.0	15.746351	1.54	0.49	0
LOG_1_IMP_PINHERT	7.05	4.58	0.0	18.022925	0.95	-1.00	0
LOG_1_PWDSTVHN	6.55	4.46	0.0	15.714750	0.81	-1.29	0
LOG_1_PASRCURG	1.15	0.81	0.0	3.214518	0.80	-1.25	0
LOG_1_PWDSTCRD	5.70	4.33	0.0	14.560529	0.64	-1.45	0
LOG_1_IMP_PEXMG1A	4.56	3.40	0.0	11.324999	0.64	-1.52	0
LOG_1_PWATFS	6.93	6.14	0.0	16.857609	0.35	-1.71	0
LOG_1_PASRDWNG	2.41	2.67	0.0	6.658211	0.19	-1.38	0
LOG_1_PASRBUYG	1.93	2.26	0.0	5.592172	0.13	-1.46	0
BIN_1_PWASTDEP	24.11	38.40	0.0	82.000000	0.12	-1.10	0
LOG_1_PWARRSPL	7.89	8.12	0.0	19.126209	0.03	-1.85	0
BIN_1_PEFATINC	28.87	49.47	0.0	98.000000	-0.00	-1.20	0
BIN_1_PWNETWPT	28.86	49.49	0.0	98.000000	-0.00	-1.20	0
LOG_1_PATTLMLC	7.45	7.73	0.0	17.856549	-0.02	-1.90	0
LOG_1_PWARPPT	8.72	9.40	0.0	20.290328	-0.08	-1.88	0
LOG_1_PWASTONF	2.24	13.09	7.0	17.495558	-0.45	-0.33	0
LOG_1_PWDTOTAL	7.47	11.03	0.0	19.279192	-0.64	-1.33	0
LOG_1_PWAPRVAL	8.63	12.34	0.0	20.261710	-0.71	-1.45	0
LOG_1_PWASTVHE	5.32	11.67	0.0	16.557240	-1.53	0.79	0
LOG_1_PATTCRLM	4.44	12.22	0.0	16.433958	-2.05	3.04	0

Appendix V. Descriptive Statistics – Successfully Transformed Variables

	std dev	mean	min	max	skew	kurtosis	null values
BIN_2_PWASTBND	0.22	0.03	0	2	7.90	63.74	0
BIN_2_PASRCNMG	0.18	0.03	0	1	5.25	25.53	0
BIN_2_PWAOTPEN	0.66	0.14	0	4	5.00	24.33	0
BIN_2_PWDSTOMR	0.87	0.20	0	5	4.59	20.36	0
BIN_2_PWDSTODB	0.99	0.25	0	5	4.10	15.63	0
BIN_2_IMP_PASR1MFA	0.99	0.25	0	5	4.09	15.56	0
BIN_2_PWASTSTK	1.33	0.35	0	7	4.00	15.18	0
BIN_2_PWASTMUI	1.47	0.42	0	7	3.65	12.20	0
BIN_2_PWARRIF	1.74	0.52	0	8	3.45	10.70	0
BIN_2_PWDSLOAN	2.10	0.65	0	10	3.37	10.33	0
BIN_2_PWBUSEQ	2.37	0.79	0	11	3.20	9.23	0
BIN_2_PWASTRST	4.13	1.67	0	16	2.46	4.70	0
BIN_2_PWDSTLOC	4.46	1.85	0	17	2.37	4.29	0
BIN_2_PWASTOIN	4.79	2.05	0	18	2.28	3.86	0
BIN_2_PASRFNMG	0.00	0.00	0	0	0.00	0.00	0
BIN_2_PATTCRLM	20.18	30.82	0	66	-0.04	-1.20	0
BIN_2_PWASTVHE	24.19	32.13	0	76	0.17	-1.22	0

References

Business cycle dating committee announcement July 19, 2021. NBER. (2021). Retrieved August 12, 2022, from https://www.nber.org/news/business-cycle-dating-committee-announcement-july-19-2021

How global labor markets are recovering from covid-19. Deloitte Insights. (2022). Retrieved August 12, 2022, from https://www2.deloitte.com/us/en/insights/economy/issues-by-the-numbers/impact-covid-19-labor-market-globally.html

News release. Gross Domestic Product, Third Quarter 2020 (Advance Estimate) | U.S. Bureau of Economic Analysis (BEA). (2020). Retrieved August 12, 2022, from https://www.bea.gov/news/2020/gross-domestic-product-third-quarter-2020-advance-estimate

Remes, J., Manyika, J., Smit, S., Kohli, S., Fabius, V., Dixon-Fyle, S., & Nakaliuzhnyi, A. (2021, December 15). *The consumer demand recovery and lasting effects of COVID-19*. McKinsey & Company. Retrieved August 12, 2022, from https://www.mckinsey.com/industries/consumer-packaged-goods/our-insights/the-consumer-demand-recovery-and-lasting-effects-of-covid-19

Rockefeller, B. (2022, May 20). *Global Supply Chain Pressure index: May 2022 update*. Liberty Street Economics. Retrieved August 12, 2022, from https://libertystreeteconomics.newyorkfed.org/2022/05/global-supply-chain-pressure-index-may-2022-update/

Statistics Canada. (2020, December 22). *Survey of Financial Security*. Retrieved from https://www23.statcan.gc.ca/imdb/p2SV.pl?Function=getSurvey&Id=1252634