GitHub Link: https://github.com/Sabag2127/ADS_508_Team_7_Project

Video Link: https://youtu.be/HzK233pWEAM

A Predictive Analysis of Loan Approvals through Classification Modeling and Cloud Computing

Saba Alemayehu, Dennis Myasnyankin, and Anusia Edward

Shiley-Marcos School of Engineering, University of San Diego

Abstract

The SDA Bank has recently experienced financial loss due to a surge in defaults caused by unreliable borrowers. In order to help mitigate inattentive loan approvals in the future, available financial information was utilized to develop and assess five predictive models. The classification models explored include: random forest, naive bayes, k-nearest neighbor, logistic regression, and XGBoost logistic regression. The predictive power of each model was evaluated based on the following performance metrics: accuracy, precision, recall, and f-1 score. A final model for analyzing the creditworthiness of SDA's clientele was produced, capable of generating sufficient predictions for loan approvals.

Keywords: loans, classification models, random forest, naive bayes, k-nearest neighbor, logistic regression, XGBoost, cloud computing, accuracy, precision, recall, f-1 score

Table of Contents

Background

SDA Bank is a financial institution that offers its services to individuals, small and middle market businesses, as well as large corporations. SDA Bank provides its clientele with numerous monetary resources, ranging from banking and investment management programs to financial assistance and risk mitigation services. Armed with a capable staff of two-thousand employees, the bank is committed to providing thorough, comprehensive support to all of its members. SDA's current consumer-base is comprised of nearly five hundred thousand individuals, steadily growing on a daily basis. In order to uphold the company's commitment to conveying the best financial advice, a Data Team has been established to analyze all of the records collected over the years, and derive actionable insights to further benefit its customers.

Problem Statement

The SDA Bank has recently observed an uptick in defaults caused by borrowers failing to repay loans on time. As a result, SDA has experienced a substantial decrease in funds due to these unsuitable loan candidates. Loans provide banks with a separate stream of income through the interest accrued during their repayment. Addressing this problem will not only help mitigate the present deficit concerns, but enable the bank to successfully provide loans for other potential borrowers through an increased cash flow. In order to resolve this issue in the future, the bank has tasked its Data Team with identifying suitable loan candidates based on existing client records. The Data Team has decided to analyze a variety of potential predictors that could be used to develop classification models. These models will utilize available data to determine whether individuals applying for loans are worth SDA's financial investment.

Goals

Three main goals were set for the purpose of this project. The three goals are as follows: (1) create a classification-based machine learning algorithm to determine whether clients are good investments, (2) identify specific attributes associated with creditworthy clientele, and (3) decrease investments in borrowers at risk of defaulting on loans. Accomplishing these objectives will help reduce the bank's financial losses and increase its capital reserves from the profits generated by accumulated interest on reliable loans.

Non-Goals

In order to ensure that this project would remain focused and thorough throughout the course of analysis, three non-goals were determined as well. The three non-goals are as follows: (1) the models developed in this project will not be used to identify potential targets for loan marketing opportunities, (2) the analysis performed in this project will not be looking into factors associated with bad loan candidates, and (3) this project will not be used as a way to blacklist clients who are currently classified as bad loan candidates.

Data Sources

The data was sourced from CTU Prague Relational Learning Repository. From the relational database, titled "Finance", the following four datasets were chosen for this project: trans.csv, trans_2.csv, account.csv, and loan.csv. The trans.csv has 1,056,320 rows, the trans_2.csv has 137,327 rows, the account.csv file has 4,501 rows, and the loan.csv file has 683 rows. The variables within these datasets consist of binary, categorical, ordinal, and numerical variables. It should also be noted that the following data risks were identified within the datasets: missing values, class imbalance of the target variable, duplicates, structural errors, and unwarranted or irrelevant data. The data was processed and analyzed through Python in *SageMaker*. The code for this process is documented and stored in GitHub. The individual datasets are also stored in S3, which can be accessed through *SageMaker*.

Data Ingestion

The data for this project is stored in S3 buckets. This data was manually added to the following S3 folder: “s3://ads508loanapproval/datasets/”, which holds 4 buckets containing the data from the CSV files chosen for this project: (1) trans.csv, (2) trans_2.csv, (3) loan.csv, and (4) account.csv. The path to access the datasets from AWS S3 is as follows: 's3://ads508loanapproval/datasets/{data#}/{filename}.csv’. For example, in order to access the first dataset trans.csv, the following path would be used: ‘s3://ads508loanapproval/datasets/data1/trans.csv’.

The tools utilized for data ingestion and exploration include the following: *boto3, Matplotlib, Plotly, Seaborn, Pandas, pyAthena,* and *SageMaker Data Wrangler*. The *boto3* package allows one to create, update, and delete AWS resources from the Python scripts. For the purpose of this project, this package aided with the ingestion of the data from the previously specified AWS S3 bucket into the AWS SageMaker Python script. Additionally, this package was later used within the project to manipulate the data. *pyAthena* was also later used, in conjunction with *boto3* and *SageMaker*, in order to ingest the data into an *Athena* database. This was done to aid with combining datasets. For the data exploration part of the project, the *Matplotlib*, *Plotly*, and *Seaborn* packages were leveraged to create visualizations for features of interest. *AWS SageMaker Data Wrangler* was used as a method for interacting with the data. More specifically, this package was used to read in the data and to perform data quality checks. The *Pandas* package was used for general analysis transformations and visualizing data.

Data Exploration

The preliminary data exploration investigated the sourced datasets on an individual basis. The following was examined in each dataset: the total number of rows and columns, data types associated with the features, the amount of different data types present, the number of missing values, overall memory usage, and statistical summaries of numeric features. Histograms were created to visualize the distribution of numeric variables and determine skewness. Bar plots were constructed to illustrate value breakdowns in categorical variables and check whether they were disproportionate. Class imbalance of the target variable, "status", was observed in the *loan* dataset; this issue needed to be addressed prior to model training. The "amount" and "balance" features in the *trans* and *trans_2* datasets, along with "amounts" and "payments" in the *loan* dataset, were right-skewed. The positive skews depicted by the distributions suggested normalization of the features during the preprocessing phase. Finally, individual correlation heat maps were created for each dataset. None of them raised concerns regarding multicollinearity among features.

After combining the four datasets into a single source of truth, the merged dataset was used to conduct a second exploratory analysis. Information on dataset shape, feature data types, null counts, memory usage, and summary statistics of numerical values were extracted from this unified dataset. Another heatmap was created to double check potential multicollinearity. With correlation values below 0.7, it was concluded that no multicollinearity existed among the variables in the newly formed dataset. Supplementary visualizations were produced for more in-depth analysis of relationships between consolidated features and the target variable.

First, a scatterplot was produced for the 'avg_pay_amt' and 'pay_amt' features displayed in Figure 1. The 'avg_pay_amt' refers to the average payment amount submitted by a client based on their previous transactions, while ‘pay_amt’ refers to the total amount a client is required to pay toward their loan. Looking at Figure 1, two interesting observations can be made. They are as follows: (1) the average repayment amounts spanning forty to fifty thousand dollars were processed using credit cards not affiliated with SDA bank, and (2) the majority of loan repayments ranging from fifty to sixty thousand dollars were submitted via bank-to-bank transfers. The horizontal bar chart displayed in Figure 2 was produced to gauge creditworthiness of customers based on the duration of loan contracts. The bars labeled “A” and “C” indicate stability, while the labels “B” and “D” indicate volatility in regards to loan repayments. This suggests that borrowers with longer contracts are preferred to those with shorter term agreements.

Figure 1.

Scatterplot of client payments

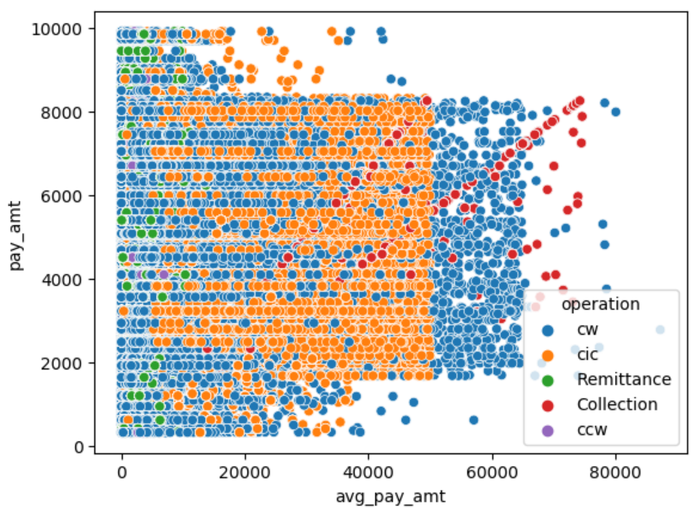
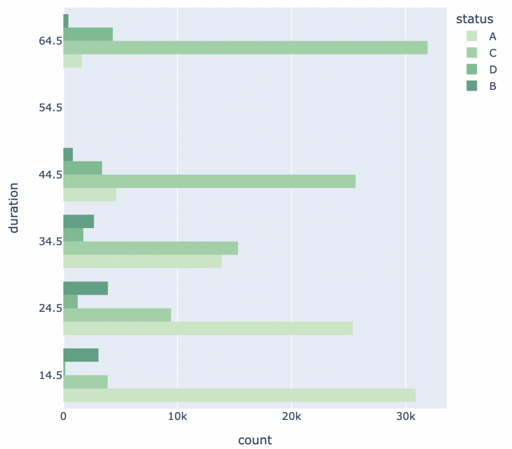


Figure 2.

Status based on Loan Duration



The payment frequency of clients required to return distinct amounts toward their loan balance is illustrated in Figure 3. Interpreting the plot below, it is evident that regardless of the amount owed, most clients are expected to submit monthly payments. More specifically, it can be noted that clients required to pay more do not necessarily have to pay more frequently than clients presenting smaller balances. In order to observe recommendable clientele based on payment frequency, the visualization displayed in Figure 4 was constructed. This bar plot reveals that customers required to send bimonthly payments exhibit a higher probability of defaulting on loans. This may signify that contracts linked to bimonthly payments are of higher risk.

Figure 3.

Frequency of payment based on requirements and balance

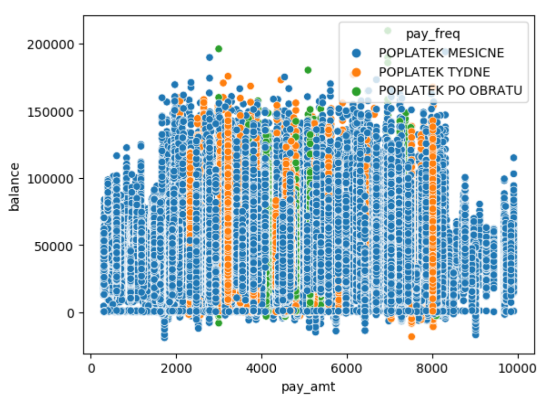
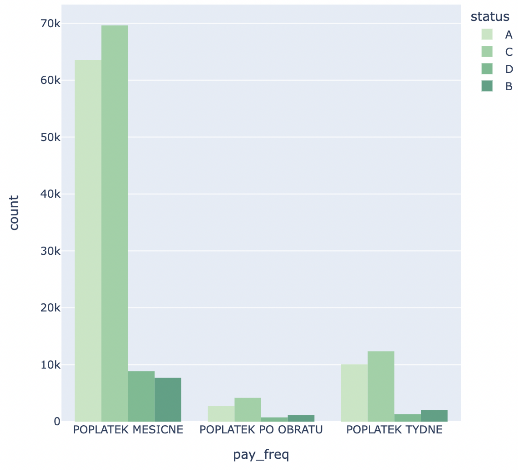


Figure 4.

Status based on payment frequency



Data Preparation

The general data cleansing techniques and steps that were taken for this project include combining datasets, multicollinearity check and corrections, recoding categorical data, dropping unnecessary or irrelevant columns, correcting formatting errors, renaming columns for clarity, class imbalance bias detection and mitigation, data splitting, and normalizing numeric data.

In order to carry out this project, the following four datasets were chosen: trans.csv, trans_2.csv, loan.csv, and account.csv. The *trans* dataset consists of 1,056,320 records and 11 columns. The columns within this dataset are as follows: ‘index’, ‘trans_id’, ‘account_id’, ‘date’, ‘type’, ‘operation’, ‘amount’, ‘balance’, ‘k_symbol’, ‘bank’, and ‘account’. From this dataset, the variables that were kept include ‘trans_id’, ‘account_id’, ‘amount’, and ‘balance’. The ‘trans_id’ variable indicates the unique identifier of the payment towards one’s loan. The ‘account_id’ variable indicates which client’s account is being referenced. The ‘amount’ column indicates the average amount that was paid towards the client’s loan over the duration of the loan. The ‘balance’ column indicates the amount of money that still needs to be paid towards the loan. The ‘index’ variable was dropped as it does not provide any new information. The ‘date’ column was also dropped because the information this variable gives is also present in a more concise manner under the ‘duration’ column. The ‘type’ and ‘operation’ columns were dropped from this dataset. Since the values of these two columns were recorded in Czech, the English translated versions of these variables, found in the ‘trans_2’ dataset, were kept instead. The ‘k_symbol’ and ‘bank’ columns reflect which bank the information is recorded at. This information is not necessary for the project at hand as this project is only considering data in regards to the SDA bank.

The ‘trans_2’ dataset has a total of 137,327 records and 6 columns. The columns within this dataset are as follows: ‘index’, ‘transaction_id’, ‘type’, ‘operation’, ‘amount2’, and ‘balance’. From this dataset, the variables that were kept include ‘transaction_id’, ‘type’, and ‘operation’. The ‘transaction_id’ variable indicates the unique identifier of the payment towards one’s loan. The ‘type’ variable indicates the type of transaction (credit or withdrawal), while the ‘operation’ variable indicates the mode of the payment (credit card, cash, bank to bank transfer). The ‘index’ variable was not kept as it does not provide any new or relevant information. The information from the ‘amount2’ and ‘balance’ columns were already retrieved from the previous dataset and therefore unnecessary to retain.

The ‘loan’ dataset has a total of 683 records and 8 columns. The columns within this dataset are as follows: ‘index’, ‘loan_id’, ‘account_id’, ‘date’, ‘amount’, ‘duration’, ‘payments’, and ‘status’. From this dataset, the variables that were kept include ‘loan_id’, ‘amount’, ‘duration’, ‘payments’, and ‘status’. The ‘loan_id’ column indicates the unique identity of the corresponding loan contract. The ‘amount’ column refers to the amount of money that was borrowed from the bank as a loan. The ‘duration’ column indicates the amount of time, specified within the contract, in which the client has to pay back the loan. The 'payments' column indicates the amount of money that is expected to be paid each week, month, or twice a month. The 'status' column indicates whether or not the loan borrowing transaction, between the client and the bank, was a success. For the purpose of this project, a successful loan transaction is defined as a borrower being on track to pay back the loan within the given time, or the borrower being able to fully pay off the loan within the given time. The other variables were not included as their information was reflected in the previous datasets mentioned.

The 'account' dataset has a total of 4,501 records and 5 columns. The columns within this dataset are as follows: 'index', 'account_id', 'district_id', 'frequency', and 'date'. From this dataset, the 'frequency' variable was kept. This variable indicates when each of the payments are expected to be paid (weekly, monthly, or bimonthly) according to the contract. Overall, it can be noted that the following variables from all four datasets were kept: 'trans_id', 'account_id', 'amount', 'balance', 'type', 'operation', 'loan_id', 'amount', 'duration', 'payments', 'status' and 'frequency'. All four datasets were combined using *pyAthena*, *pandas*, *boto3*, and *SageMaker*. The four datasets were ingested from S3 into the Athena database, which was named as "SDAloans". The *pd.read_sql* function was used to create tables with the specified variables mentioned above. The four tables were then combined using the *left join* function. While combining the four datasets through AWS *pyAthena* in *SageMaker*, redundant columns were excluded. Based on this, no fields were needed to be combined for this project.

The two key transformations that were applied to the data include the transformation of categorical variables and the transformation of numeric variables. In terms of transforming categorical data, this was accomplished by dummy coding the target variable 'status'. Initially, the 'status' attribute was composed of the following values: A, B, C, and D. The 'A' indicates that a client's contract had ended and their loan had been completely paid. The ‘B’ indicates that the client's contract ended; however, the loan remains unpaid. The ‘C’ indicates that the loan contract is still in progress and that the client is on track to pay off the loan. The ‘D’ indicates that the client’s contract is in progress and that the client is in debt as they are behind on paying back their loans. Based on the definitions of each of the four values under ‘status’, it is evident that ‘A’ and ‘C’ indicate a good loan candidate while ‘B’ and ‘D’ indicate a poor loan candidate. In order to reflect this, the ‘status’ column was re-coded using the *.map* function so that ‘A’ and ‘C’ are indicated by 1 (good loan client) and ‘B’ and ‘D’ are indicated by 0 (poor loan client). The other two categorical variables, ‘operation’ and ‘pay_freq’, were handled using the *one-hot-encoding* function. Essentially, what this resulted in is binary columns for each of the unique values under the ‘operation’ and ‘pay_freq’ columns. Based on this, a total of eight additional columns were produced: ‘pay_mo’ (pay monthly), ‘pay_wk’ (pay weekly), ‘pay_bimo’ (pay bimonthly), ‘op_ccp’ (credit card payment), ‘op_cp’ (cash payment), ‘op_ccb’ (credit card payment through another bank), ‘op_cb’ (cash payment through another bank), and ‘op_bb’ (bank-to-bank transfer).

The second transformation that was applied to the dataset was the normalization of the numeric data. The numeric data for this dataset includes the following columns: ‘loan_amt’, ‘duration’, ‘pay_amt’, ‘avg_pay_amt’, and ‘balance’. It should be noted that this step was taken after the splitting of the data in order to ensure that there were no concerns of data leakage. For the normalization of the data, the first step taken was to use sk.learn’s *preprocessing.StandardScaler()* function and *scaler.fit_transform* function on the train data. This was done in order to ensure that the test and validation sets remain as unseen data. After the normalization of the numeric data was done based on the train set, the normalized columns which were indicated by the prefix of ‘z’ were joined to the original data frame using the *pd.concat* function. Then, the normalized values were set to the train, test, and validation sets using the *.index* function. Finally, the dimensions of each of the sets (train, test, and validation) were checked to ensure that there were no mishaps while performing the data transformation.

For the purpose of this project, the way in which class imbalance was handled was through the oversampling of the minority class. Oversampling can aid with creating a more balanced dataset so that the results of the model predictions are more accurate. Essentially, oversampling was chosen over undersampling, as oversampling will ensure that no information is lost. The decision of using the *RandomOverSampler* function over other oversampling techniques such as *SMOTE* oversampling is because it is considered more robust in terms of model results than *SMOTE* oversampling (Chadha, 2022). Initially, the dataset had an imbalance of twelve non-recommendable to eighty-eight recommendable loan candidates. After balancing the dataset, the ratio of non-recommendable loan candidates to recommendable loan candidates is forty-two to fifty-eight. This change indicates the proficiency of the oversampling technique. The data was split using *fast_ml.model_development’s train_valid_test_split* function. Using this function, the data was split into 70% training, 15% testing, and 15% validation sets. Based on the data splits, the following dimensions for each of the sets were obtained: x_train (47497, 13), y_train (47497, 1), x_valid (10178, 13), y_valid (10178, 1), x_test (10179, 13), and y_test: (10179, 1).

Model Training

In order to train the model, the following tools were utilized: “bring your own script”, “bring your own container”, and “built-in-algorithms”. The tool “bring your own script” was used to write code for creating and training the following classification models: random forest, naive bayes, k-nearest neighbor, and logistic regression. The “bring your own container” tool was used alongside the “built-in-algorithms” tool to create a container and run the XGBoost model. This model was implemented in order to optimize the initial logistic regression model. Each of these tools were utilized to provide flexibility in creating and training the various models.

The machine learning algorithms that were initially employed for this analysis include the following four models: random forest, naive bayes, k-nearest neighbor, and logistic regression. The random forest model can account for large dataset sizes. It can also mitigate any overfitting concerns, and it is time efficient (Radha, 2023). The naive bayes model was chosen as one of the models based on its overall efficiency and robustness in its prediction results (Yildirim, 2021). The k-nearest neighbor model is a simple model that therefore has easily interpretable predictions, and the model is recognized for having accurate predictions (Vatsal, 2022). The logistic regression model was also employed as one of the models, as this model is efficient and is known to provide results that are easily interpretable (*Logistic Regression Model Explained - AWS*, n.d.). Additionally, through Amazon Sagemaker, the XGBoost model was employed to optimize the logistic regression model. The XGBoost model can efficiently handle large datasets, and its range of hyperparameters can be tuned to optimize the model (*How XGBoost Works - Amazon SageMaker*, n.d.). Furthermore, the XGBoost model provides a faster alternative to optimization than other commonly used optimizers such as GridSearchcv (*How XGBoost Works - Amazon SageMaker*, n.d.). For the purposes of this project, these five models were employed to account for the size of the data and for the prediction accuracy when determining recommendable over non-recommendable loan candidates.

The model that was optimized through hyperparameter tuning was the XGBoost model. For this model, the following tuning parameter was passed on the train data: “sagemaker.inputs.TrainingInput”. This parameter was passed onto the training data in the S3 container. This S3 container was named as “train” in a ‘csv’ format. Additionally, the “sagemaker.estimator.Estimator” parameter, along with the “xgb.set_hyperparameters” parameters, were passed. The specific parameters of the XGBoost are as follows: max_depth=5, eta=0.2, gamma=4, min_child_weight=6, subsample=0.8, silent=0, objective='binary:logistic', num_round=100. Since cross validation is used in conjunction with max_depth, the max_depth parameter was set to 5. This parameter was used to help with concerns of overfitting. To further address overfitting, the min_child_weight was set to a value of 6. In order to increase the robustness of the model, the eta was set to a value of 0.2. To ensure that the model was conservative, the gamma parameter was set to a value of 4. To correct for any concerns of underfitting, the subsample parameter was set to 0.8. The silent parameter was set to zero to further understand how the model was trained. Lastly, the objective was set to ‘binary:logistic’ to account for the model being a binary classification model.

The instance that was needed in order to process the data was “m1.m5.large”. Initially, the instance was smaller; however, this caused the kernel to shut down each time the XGBoost model was run. In order to eliminate this issue, the instance was set to a larger size. The size of the data that was processed was 47,497 records.

The model was evaluated using the following metrics: accuracy, precision, recall, and F1. Additionally, an ROC plot of all the model results were plotted in order to determine the best model. These evaluation metrics aided in determining the best model to address the business goal of creating a model that will help predict which bank clients would be recommendable for a loan.

Measuring Impact

This project aims to create a classification model capable of correctly determining which clients are potentially good loan investments, based on a variety of financial qualities. To ensure the model serves its purpose, producing recommendations that will benefit the bank financially, two important metrics that were investigated are sensitivity and AUC scores for ROC graphs. Sensitivity evaluates the model's ability to predict true positives. For the purpose of this project, this metric helps determine recommendable clients. The AUC score, which stands for "Area under the ROC curve," helps determine how well the models can differentiate between good and bad loan investments.

Security, Privacy and Other Risks

In terms of security concerns, the data that is being processed and analyzed does not include protected health information (PHI). This is the case as the data reflects financial information, not health. Furthermore, it should be noted that this data does include Personally Identifiable Information (PII), as the information has been coded down to various IDs such as bank account IDs. This identification could possibly be used indirectly to infer the identity of the individual. Additionally, there is information regarding the individuals financial information in regards to their bank account balance and transaction balances. This can also be regarded as PII. The user behavior, in terms of financial transactions, is stored within the data. Additionally, credit card data is stored as individual transactions within the dataset.

Bias and Ethical Considerations

The data bias that is considered for this project is the class imbalance bias that is present within the target variable 'status' within the loan.csv dataset. Through the exploratory data analysis, it is evident that there is a strong class imbalance between the number of loans that were approved versus the loans that were not approved. Class imbalances lead to the machine learning classifier having a tendency to be more biased towards the majority class, which in this case would be individuals whose loan applications were approved. This in turn will cause less accurate classification of the minority class, which in this case would be individuals whose loan applications were rejected.

An ethical consideration is the possible way in which the results of the algorithm will be employed. The dataset contains a unique customer identifier, which should solely be used for the purpose of identifying appropriate clients to be considered as loan borrowers for this particular bank. This project is not created to blacklist clients from becoming eligible for borrowing loans in the future or from being considered by other banks for loans or property investments. Additionally, another ethical concern that may be considered is the safety and security of the client records. The data contains sensitive information regarding an individual's financial information such as credit card information, transaction balances, and bank account balances that should be protected. Based on this, the security of the data should be a priority to ensure that the clients personal and financial information is not breached.

Future Enhancements

With extended time and additional resources, the following enhancements can be made to this project: model optimizations, the inclusion of additional features, and the automation of the current model pipeline. The final classification model was developed using Amazon Sagemaker’s Built-In XGBoost hyperparameter tuning tools. Incorporating this algorithm as opposed to the other potential models initially examined, stemmed from its availability and ease of use thanks to Amazon’s Sagemaker platform. Other classification models also showed great potential in determining loan approvals, when evaluating for the accuracy, precision, recall, and f-1 score performance metrics. If more time was available for project completion, the K-Nearest Neighbors and Random Forest model hyper-parameter tuning could show additional improvements in the metrics examined during analysis.

The second enhancement that should be considered is the inclusion of additional features. Additional loan approval data could be utilized in the future. From the client data set examined during exploratory data analysis, the district_id could have been incorporated in model training if more information was available on the districts referenced. If useful insights can be derived from district information, such as which districts are more likely to be associated with loan-worthy clientele, that data can be included in future training opportunities. The database from which our initial data was acquired also had data sets on card information. Merging credit card information that does not lead to PII issues, such as card status/level (Platinum, Gold, Silver, etc.) may provide useful insights too. Including additional features could enable developing even more accurate models in the future.

Finally, the last enhancement that should be considered is the automation of the current model pipeline. The loan approval pipeline in its present state is relatively static, meaning if more data becomes available down the line, there are no mechanisms in place to incorporate it. Automating the pipeline to handle the cleaning and preparation of data fed into it would be something worth consideration. The automation of these processes goes hand in hand with scalability of the model. Although the model is capable of handling thousands of records, whether those capabilities extend to millions of records remains unknown. Adjusting the pipeline's infrastructure to automate data cleaning and preparation tasks would allow us to better gauge scalability concerns in the future.

References

Chadha, A. S. (2022, January 6). *Handling Imbalanced Datasets With Oversampling Techniques. It's Pros & Cons.* Medium. https://medium.com/analytics-vidhya/handling-imbalanced-datasets-with-oversampling-techniques-its-pros-cons-ba9f36ac5b71

Financial Dataset. (n.d.). CTU Prague Relational Learning Repository. https://relational.fit.cvut.cz/dataset/Financial

How XGBoost Works - Amazon SageMaker. (n.d.). https://docs.aws.amazon.com/sagemaker/latest/dg/xgboost-HowItWorks.html

Logistic Regression Model Explained - AWS. (n.d.). Amazon Web Services, Inc. https://aws.amazon.com/what-is/logistic-regression/#:~:text=Logistic%20regression%20is%20a%20data,outcomes%2C%20like%20yes%20or%20no.

Radha, S. E. (2023, March 24). Understand Random Forest Algorithms With Examples (Updated 2023). Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/

Vatsal, V. (2022, May 20). K Nearest Neighbours Explained - Towards Data Science. Medium. https://towardsdatascience.com/k-nearest-neighbours-explained-7c49853633b6

Yıldırım, S. (2021, December 13). Naive Bayes Classifier — Explained - Towards Data Science. Medium. https://towardsdatascience.com/naive-bayes-classifier-explained-50f9723571ed

A Predictive Analysis of Loan Approvals through Classification Modeling and Cloud Computing

Saba Alemayehu, Dennis Myasnyankin, and Anusia Edward

In [54]:

```
# Necessary pips
#! pip install pyathena
#! pip install awswrangler
#! pip install fast_ml
#! pip install smclarify
#! pip install -U seaborn
#! pip install -U kaleido
#! pip install imblearn
#! pip install xgboost
```

In [3]:

```
# Necessary Imports
import boto3, os, sagemaker
from sagemaker import clarify
import io
import pandas as pd
from pandas.core.internals import concat
import seaborn as sns
import numpy as np
import fast_ml
from fast_ml.model_development import train_valid_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn import preprocessing
from sklearn.utils import resample
import imblearn
from imblearn.over_sampling import RandomOverSampler
from smclarify.bias import report
import matplotlib.pyplot as plt
from plotly.subplots import make_subplots
import plotly.express as px
import plotly.graph_objects as go
import plotly.io as pio
pio.renderers.default = "svg" #"/svg"
import pyathena as pa
from pyathena import connect
from pyathena.pandas.cursor import PandasCursor
import awswrangler as wr
from sklearn.model_selection import train_test_split, \
RepeatedStratifiedKFold, RandomizedSearchCV
from sklearn.metrics import roc_curve, auc, mean_squared_error,\
precision_score, recall_score, f1_score, accuracy_score,\
confusion_matrix, plot_confusion_matrix, classification_report
from sagemaker.tuner import HyperparameterTuner
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
```

```python
from sklearn import metrics
from scipy.stats import loguniform
from sagemaker.serializers import CSVSerializer
import xgboost
import warnings
warnings.filterwarnings("ignore")
```

Importing Data from S3 Bucket

In [4]:
```python
# Data #1: trans.csv from s3 bucket
trans_df = wr.s3.read_csv(path="s3://s3://ads508loanapproval/\
datasets/data1/trans.csv")
trans_df.head(3)
```

Out[4]:

	index	trans_id	account_id	date	type	operation	amount	balance	k_symbol	bank
0	0	1	1	3/24/95	PRIJEM	VKLAD	1000	1000	NaN	NaN
1	1	5	1	4/13/95	PRIJEM	PREVOD Z UCTU	3679	4679	NaN	AB
2	2	6	1	5/13/95	PRIJEM	PREVOD Z UCTU	3679	20977	NaN	AB

In [5]:
```python
# Data #2: trans_2.csv from s3 bucket
trans2_df = wr.s3.read_csv(path="s3://s3://ads508loanapproval/\
datasets/data2/trans_2.csv")
trans2_df.head(3)
```

Out[5]:

	index	transaction_id	type	operation	amount2	balance
0	0	289	Credit	Collection	0	0
1	1	290	Credit	Collection	0	0
2	2	291	Credit	Collection	0	0

In [6]:
```python
# Data #3: loan.csv from s3 bucket
loan_df = wr.s3.read_csv(path="s3://s3://ads508loanapproval/\
datasets/data3/loan.csv")
loan_df.head(3)
```

Out[6]:

	index	loan_id	account_id	date	amount	duration	payments	status
0	0	4959	2	1/5/94	80952	24	3373	A
1	1	4961	19	4/29/96	30276	12	2523	B
2	2	4962	25	12/8/97	30276	12	2523	A

In [7]:
```python
# Data #4: account.csv from s3 bucket
account_df = wr.s3.read_csv(path="s3://s3://ads508loanapproval/\
datasets/data4/account.csv")
account_df.head(3)
```

Out[7]:

	index	account_id	district_id	frequency	date
0	0	1	18	POPLATEK MESICNE	3/24/95
1	1	2	1	POPLATEK MESICNE	2/26/93
2	2	3	5	POPLATEK MESICNE	7/7/97

Preliminary Exploratory Data Analysis (EDA)

Data 1: trans

In [9]:
```
# number of rows and columns
trans_df.shape
```

Out[9]:
```
(1048575, 11)
```

The .shape() function was used in order to determine the number of rows and columns within this dataset. There are a total of 1,056,320 records and 11 columns.

In [10]:
```
# df info
trans_df.info()
```

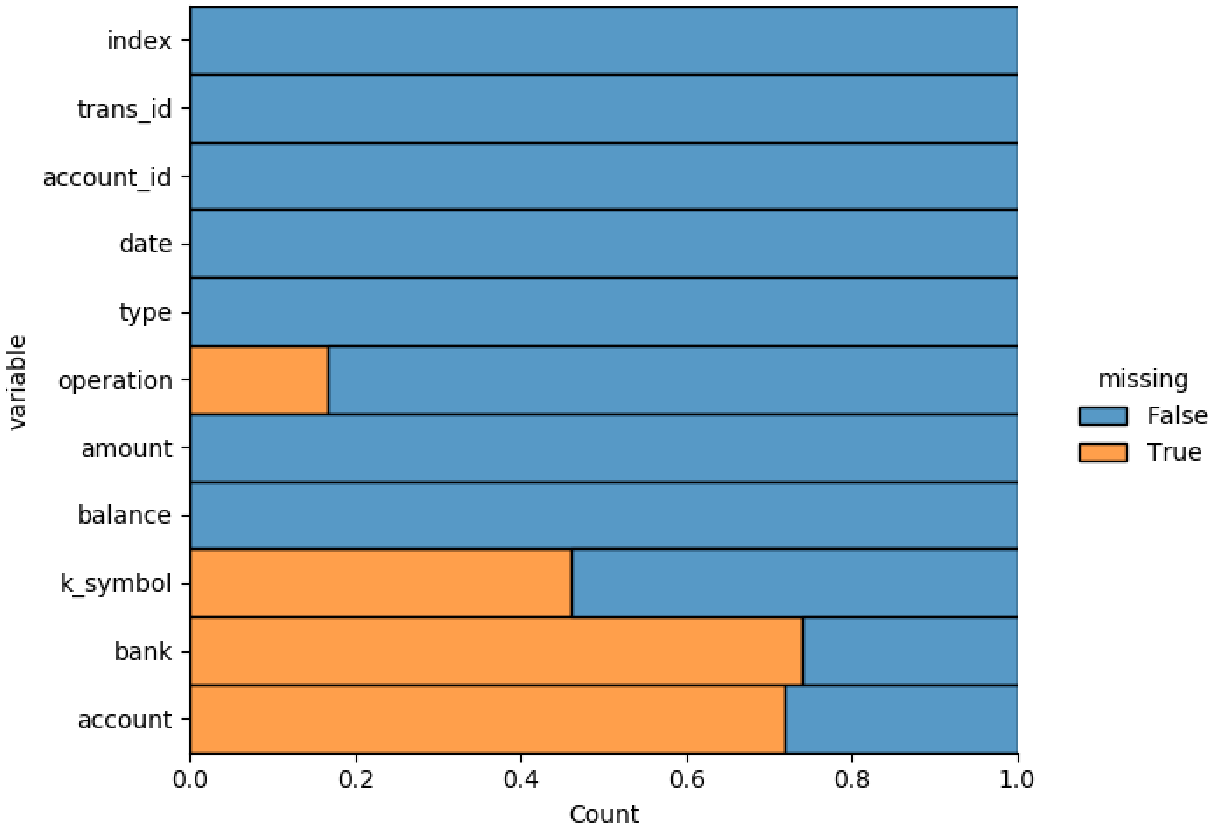
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 11 columns):
 #   Column      Non-Null Count    Dtype
---  ------      --------------    -----
 0   index       1048575 non-null  int64
 1   trans_id    1048575 non-null  int64
 2   account_id  1048575 non-null  int64
 3   date        1048575 non-null  object
 4   type        1048575 non-null  object
 5   operation   873059 non-null   object
 6   amount      1048575 non-null  int64
 7   balance     1048575 non-null  int64
 8   k_symbol    566694 non-null   object
 9   bank        273508 non-null   object
 10  account     295389 non-null   float64
dtypes: float64(1), int64(5), object(5)
memory usage: 88.0+ MB
```

The .info() function outputs an overview of the dataset. From this overview the following can be noted: names of columns, number of non-null present within the dataset, types of data, number of the varying dtypes, and memory usage.

In [11]:
```
# heat map for missing values
plt.figure(figsize=(6,6))
sns.displot(
    data=trans_df.isna().melt(value_name="missing"),
    y="variable",
    hue="missing",
    multiple="fill",
    aspect=1.25
```

```
)
plt.show()
```

<Figure size 600x600 with 0 Axes>



A heat map visualization of the missing values was produced. From this visualization, it is evident that the following rows will need to handled for missing values: operation, k_symbol, bank, account.

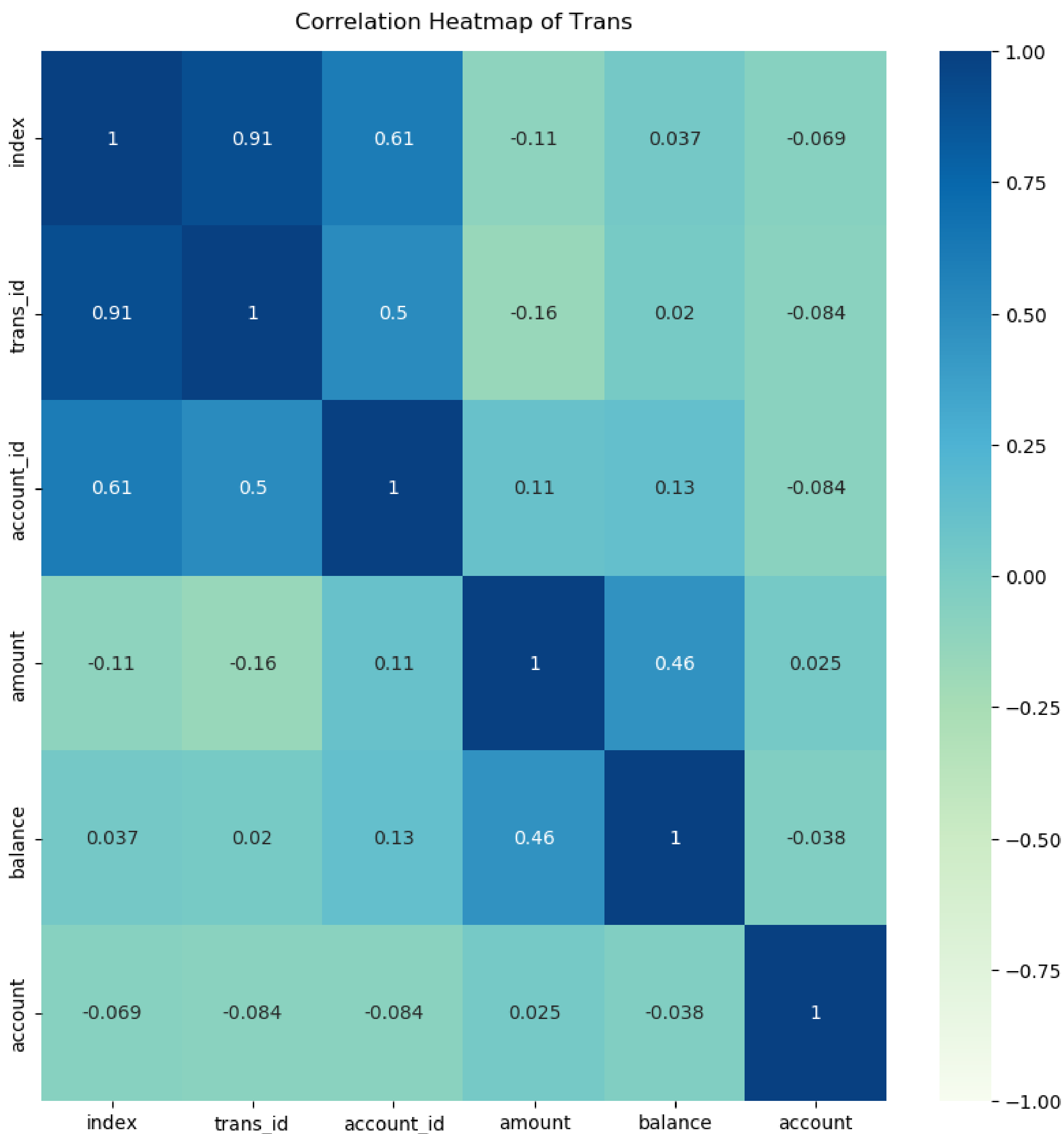
In [12]:
```
# data description
trans_df.describe()
```

Out[12]:

	index	trans_id	account_id	amount	balance	accoun
count	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	2.953890e+0
mean	5.242870e+05	1.318043e+06	2.917406e+03	5.966699e+03	3.850006e+04	4.567092e+0
std	3.026977e+05	1.215395e+06	2.472923e+03	9.544910e+03	2.211196e+04	3.066340e+0
min	0.000000e+00	1.000000e+00	1.000000e+00	0.000000e+00	-4.112600e+04	0.000000e+0
25%	2.621435e+05	4.266925e+05	1.199000e+03	1.370000e+02	2.238900e+04	1.782858e+0
50%	5.242870e+05	8.525580e+05	2.423000e+03	2.126000e+03	3.311300e+04	4.575095e+0
75%	7.864305e+05	1.935710e+06	3.636000e+03	6.851000e+03	4.957300e+04	7.201341e+0
max	1.048574e+06	3.664355e+06	1.138200e+04	8.740000e+04	2.096370e+05	9.999420e+0

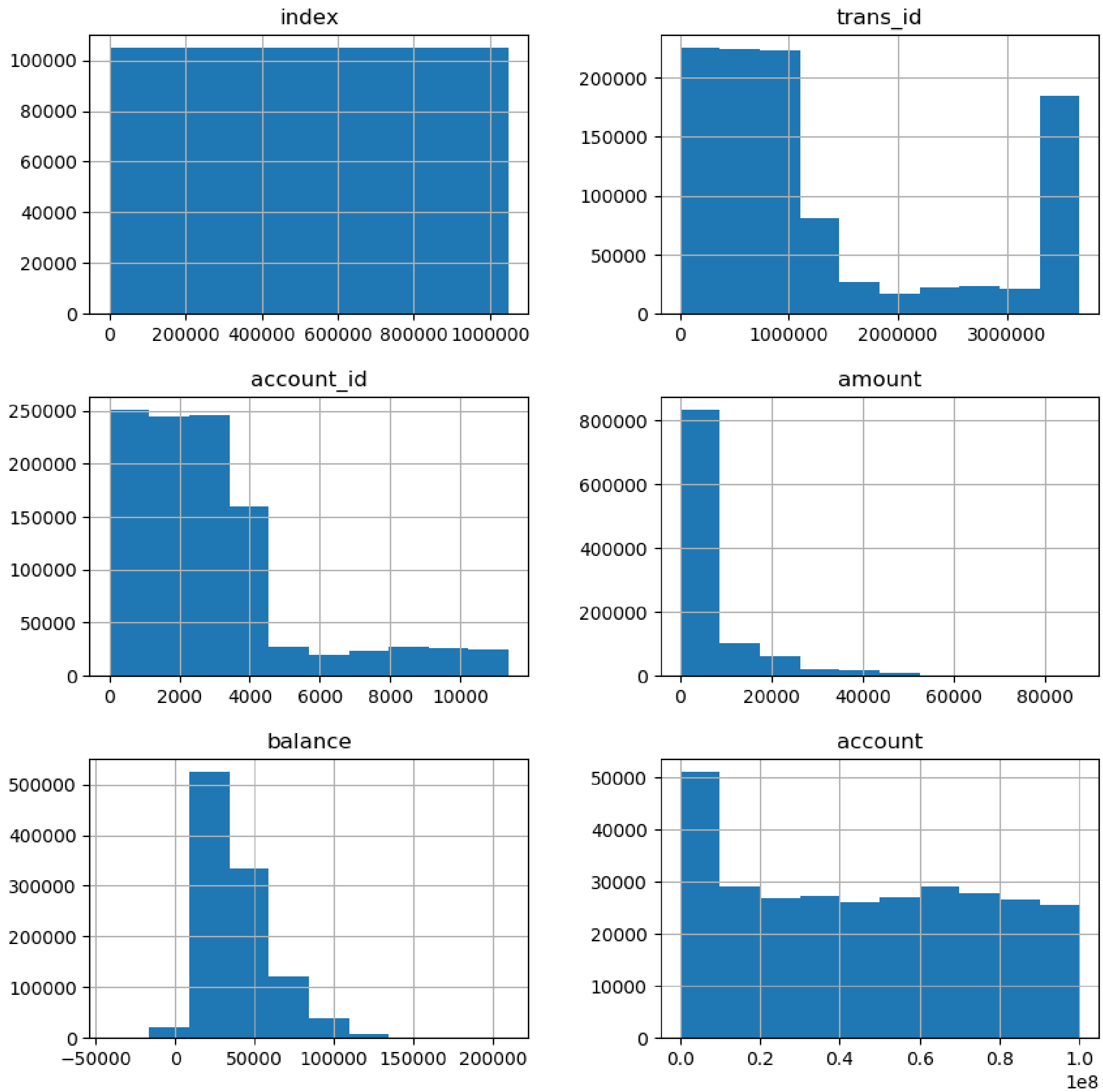
The .describe() function was used to generate a statistical summary for the numerical columns present within the data. Statistical measurements such as percentiles, mean, and standard deviation were included.

In [13]:
```
# correlation heatmap
plt.figure(figsize=(10, 10))
heatmap = sns.heatmap(trans_df.corr(), vmin=-1, vmax=1, annot=True, cmap="GnBu'
heatmap.set_title('Correlation Heatmap of Trans',
                  fontdict={'fontsize':12}, pad=12);
plt.show()
```



A correlation heatmap was produced in order to determine if there were any multicollinearity concerns. Looking at the correlation heat map, it is evident that there are some concerns of multicollinearity that will need to be addressed while pre-processing the data, as some of the correlation values are greater than 0.7. For example, index and trans_id are highly correlated with a pearson correlation value of 0.91.

In [14]:
```
# histogram of numeric variables
trans_df.hist(figsize=[10, 10])
plt.show()
```



Looking at the histograms above it is evident the variables account_id, amount, balance, and account are right skewed. This indicates that the mean values are greater than the median values resulting in the data being positively skewed. The index column is normally distributed and the trans_id column has a u-shaped distribution. Each of these distributions indicate that during the pre-processing normalization of the data through scaling and centering of the data should be considered.

In [15]:
```
# bar graph of categorical variables
fig = make_subplots(
    rows=2, cols=2,
    subplot_titles=("Transaction Type", "Operation Type", "K_Symbol",
```

```
                    "Bank"))

fig.add_trace(
    go.Histogram(x=trans_df['type']),
    row=1, col=1
)

fig.add_trace(
    go.Histogram(x=trans_df['operation']),
    row=1, col=2
)

fig.add_trace(
    go.Histogram(x=trans_df['k_symbol']),
    row=2, col=1
)

fig.add_trace(
    go.Histogram(x=trans_df['bank']),
    row=2, col=2
)

fig.update_layout(height=600, width=800,
                  title_text="Histograms of Categorical Variables",
                  bargap=0.2)

fig.show()
```

Looking at the plot above it is evident that transaction type and operation type is slightly left skewed while bank is normally distributed. The variable k_symbol has a u-shaped distribution. The variables' categories are in Czech and roughly translate to transaction types of credit card versus cash, and then under operation type it specifies further.

Data 2: trans_2

In [16]:
```
# number of rows and columns
trans2_df.shape
```

Out[16]:
```
(137326, 6)
```

The .shape() function was used in order to determine the number of rows and columns within this dataset. There are a total of 137,326 records and 6 columns.

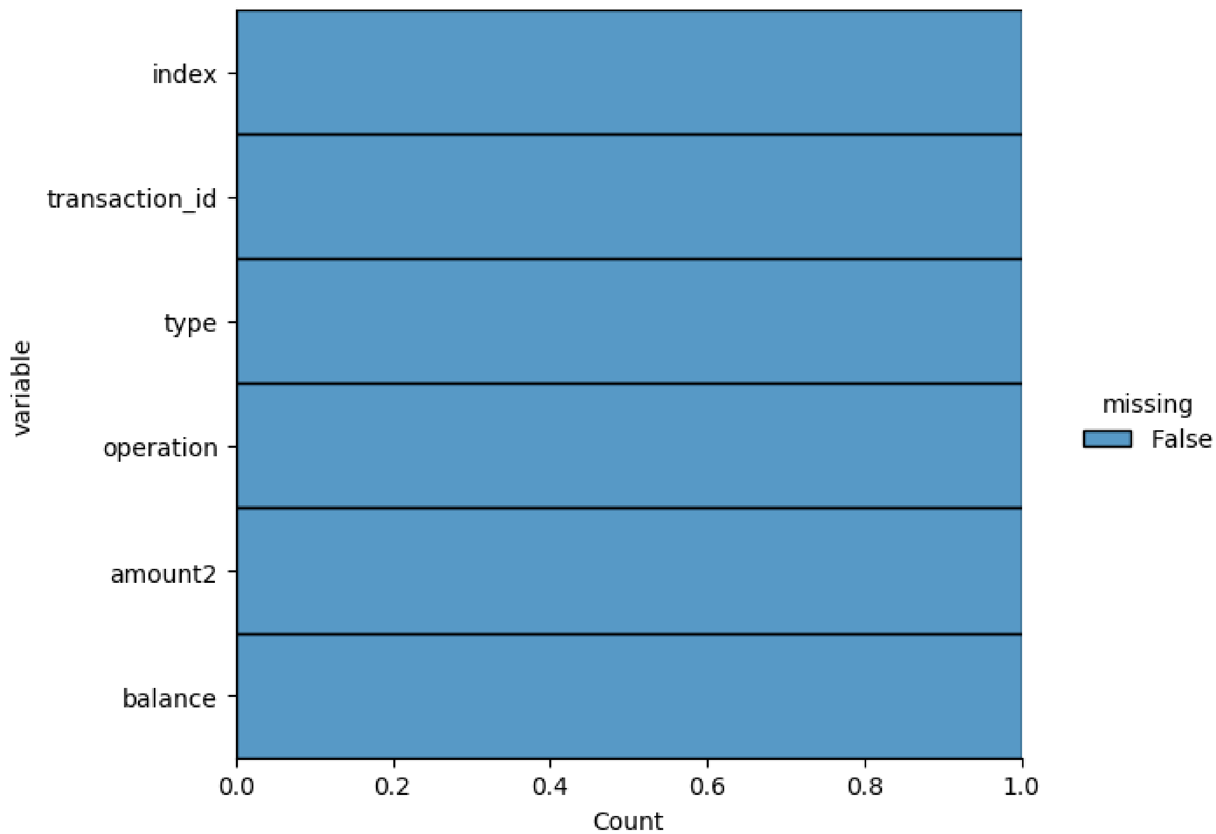
In [17]:
```
# df info
trans2_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 137326 entries, 0 to 137325
Data columns (total 6 columns):
 #   Column          Non-Null Count   Dtype 
---  ------          --------------   ----- 
 0   index           137326 non-null  int64 
 1   transaction_id  137326 non-null  int64 
 2   type            137326 non-null  object
 3   operation       137326 non-null  object
 4   amount2         137326 non-null  int64 
 5   balance         137326 non-null  int64 
dtypes: int64(4), object(2)
memory usage: 6.3+ MB
```

The .info() function outputs an overview of the dataset. From this overview the following can be noted: column names, number of non-null present within the dataset, data type, number of the varying dtypes, and memory usage.

In [18]:
```
# heat map for missing values
plt.figure(figsize=(6,6))
sns.displot(
    data=trans2_df.isna().melt(value_name="missing"),
    y="variable",
    hue="missing",
    multiple="fill",
    aspect=1.25
)
plt.show()
```

```
<Figure size 600x600 with 0 Axes>
```



A heat map visualization of the missing values was produced. From this visualization, it is evident that there are no missing values that need to be addressed while preprocessing the data.

In [19]:
```
# data description
trans2_df.describe()
```

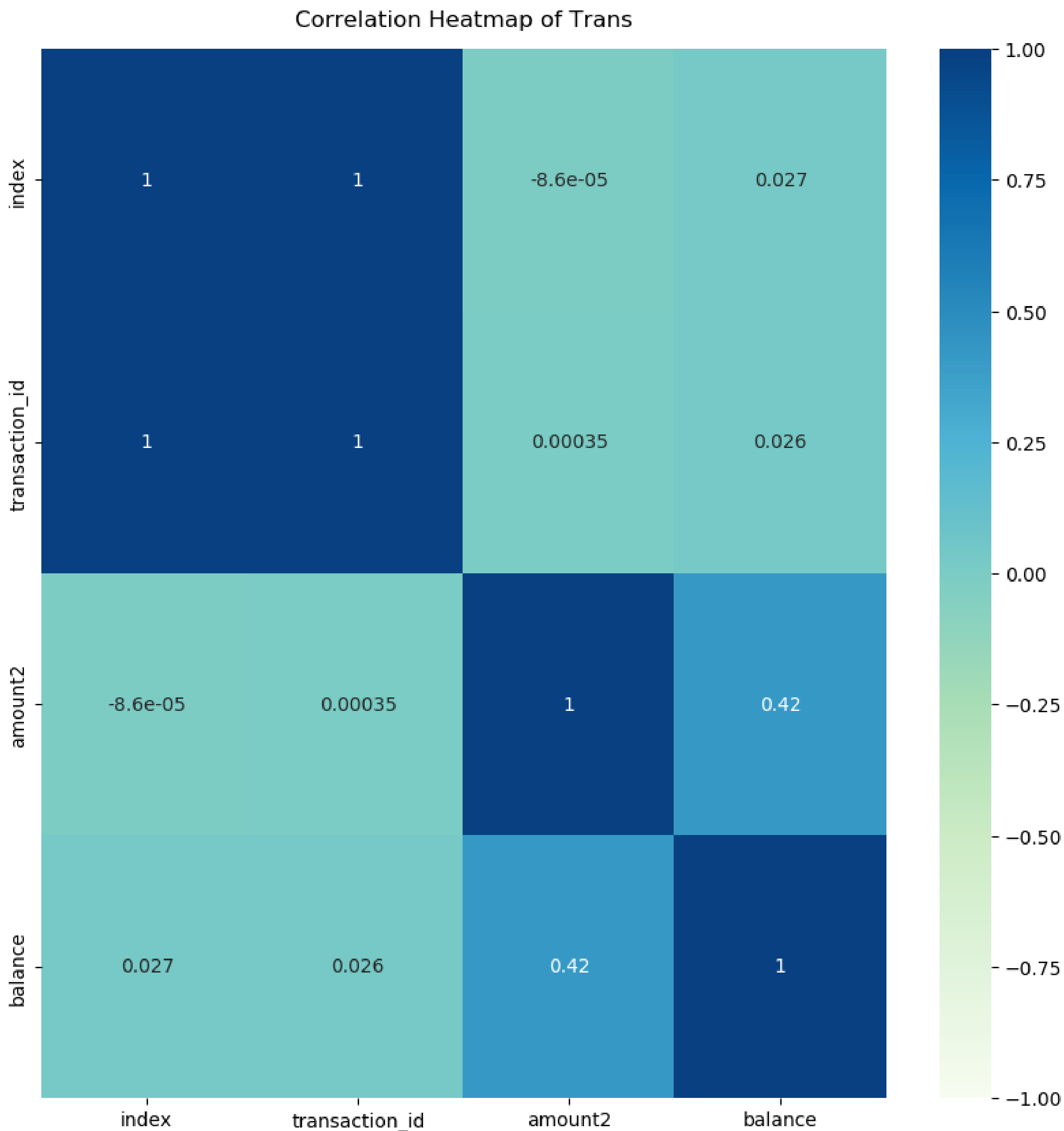
Out[19]:

	index	transaction_id	amount2	balance
count	137326.000000	1.373260e+05	137326.000000	137326.000000
mean	68662.500000	1.787270e+06	0.168431	0.369260
std	39642.745871	9.920609e+05	0.448134	0.536253
min	0.000000	2.890000e+02	0.000000	0.000000
25%	34331.250000	8.992222e+05	0.000000	0.000000
50%	68662.500000	1.779301e+06	0.000000	0.000000
75%	102993.750000	2.644035e+06	0.000000	1.000000
max	137325.000000	3.682967e+06	3.000000	3.000000

The .describe() function was used to generate a statistical summary for the numerical columns present within the data. Statistical measurements such as percentiles, mean, and standard deviation were included.

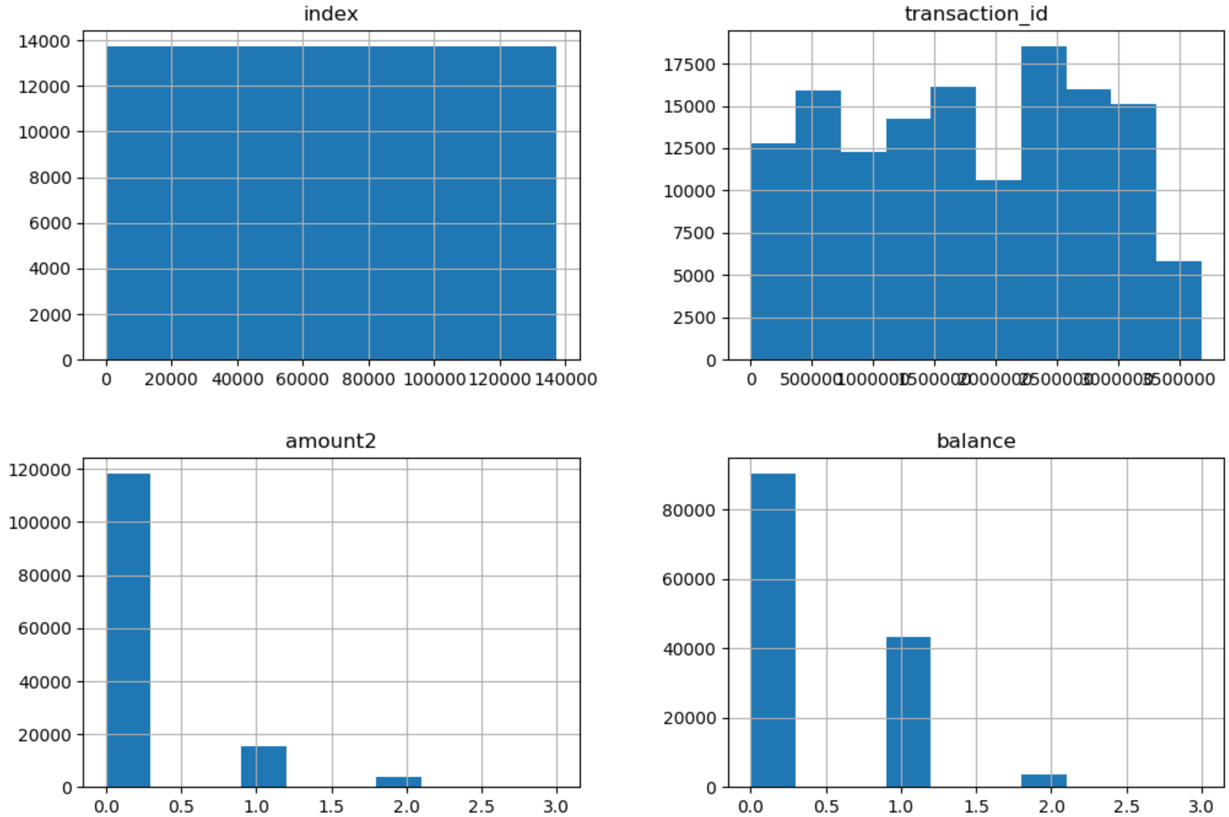
In [20]:
```
# correlation heatmap
plt.figure(figsize=(10, 10))
heatmap = sns.heatmap(trans2_df.corr(), vmin=-1, vmax=1, annot=True,
```

```
                      cmap="GnBu")
heatmap.set_title('Correlation Heatmap of Trans',
                  fontdict={'fontsize':12}, pad=12);
plt.show()
```



A correlation heatmap was produced in order to determine if there were any multicollinearity concerns. Looking at the correlation heat map, it is evident that there are some concerns of multicollinearity that will need to be addressed while pre-processing the data, as some of the correlation values are greater than 0.7. For example, index and transaction_id are highly correlated with a pearson corrleation value of 1.0.

In [21]:
```
# histogram of numeric variables
trans2_df.hist(figsize=[12, 8])
plt.show()
```



Looking at the histograms above, it is evident the variables amount2 and balance are right skewed. This indicates that the mean values are greater than the median values resulting in the data being positively skewed. The index column is normally distributed and the trans_id column has a u-shaped distribution. Each of these distributions indicate that during the pre-processing, normalization of the data through scaling and centering of the data should be considered.

In [22]:

```
# bar graph of categorical variables
fig = make_subplots(
    rows=1, cols=2,
    subplot_titles=("Transaction Type", "Operation Type"))

fig.add_trace(
    go.Histogram(x=trans2_df['type']),
    row=1, col=1
)

fig.add_trace(
    go.Histogram(x=trans2_df['operation']),
    row=1, col=2
)
fig.update_layout(height=600, width=800,
                  title_text="Histograms of Categorical Variables",
                  bargap=0.2)

fig.show()
```

Data 3: loan

In [23]:
```
# number of rows and columns
loan_df.shape
```

Out[23]:
```
(682, 8)
```

The .shape() function was used in order to determine the number of rows and columns within this dataset. There are a total of 682 records and 8 columns.

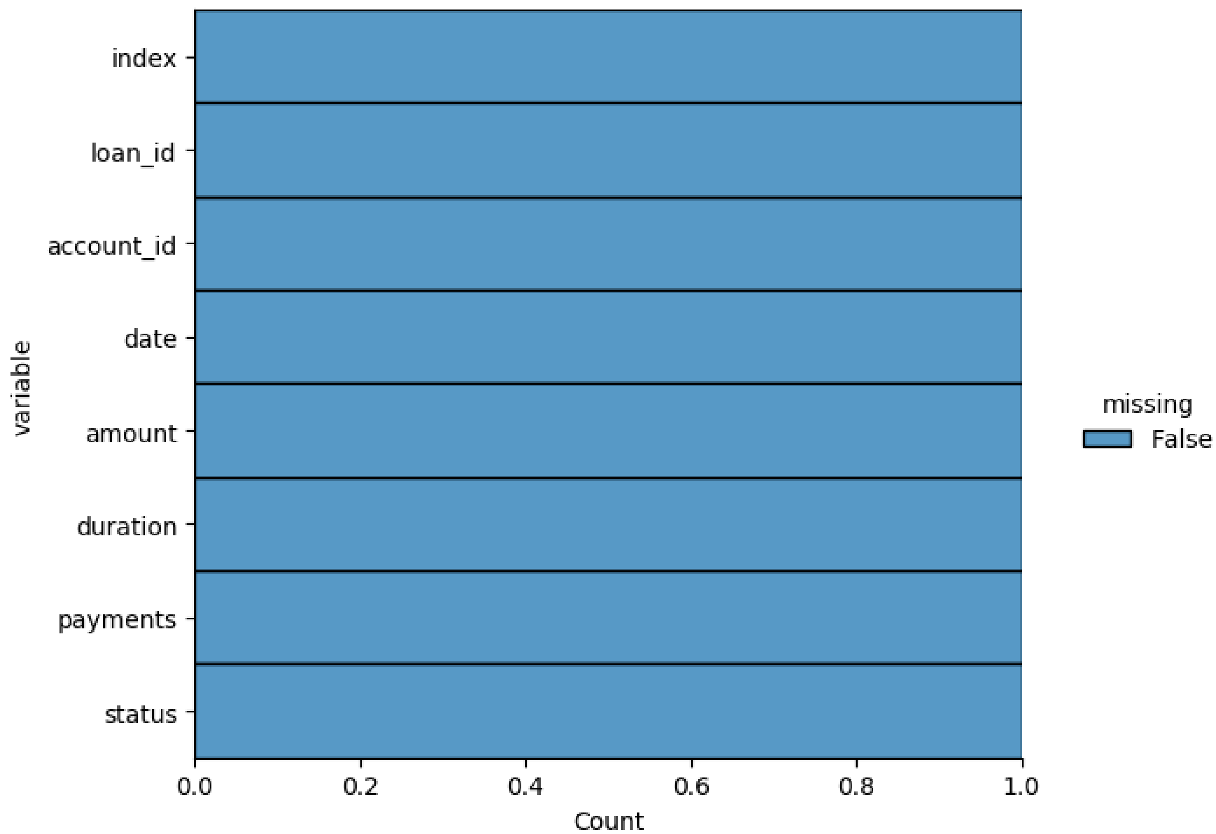
In [24]:
```
# df info
loan_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 682 entries, 0 to 681
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   index       682 non-null    int64 
 1   loan_id     682 non-null    int64 
 2   account_id  682 non-null    int64 
 3   date        682 non-null    object
 4   amount      682 non-null    int64 
 5   duration    682 non-null    int64 
 6   payments    682 non-null    int64 
 7   status      682 non-null    object
dtypes: int64(6), object(2)
memory usage: 42.8+ KB
```

The .info() function outputs an overview of the dataset. From this overview, the following can be noted: column names, number of non-null present within the dataset, data type, number of the varying dtypes, and memory usage.

In [25]:
```
# heat map for missing values
plt.figure(figsize=(6,6))
sns.displot(
    data=loan_df.isna().melt(value_name="missing"),
    y="variable",
    hue="missing",
    multiple="fill",
    aspect=1.25
)
plt.show()
```

```
<Figure size 600x600 with 0 Axes>
```



A heat map visualization of the missing values was produced. From this visualization, it is evident that there are no missing values that need to be addressed while preprocessing the data.

In [26]:
```
# data description
loan_df.describe()
```

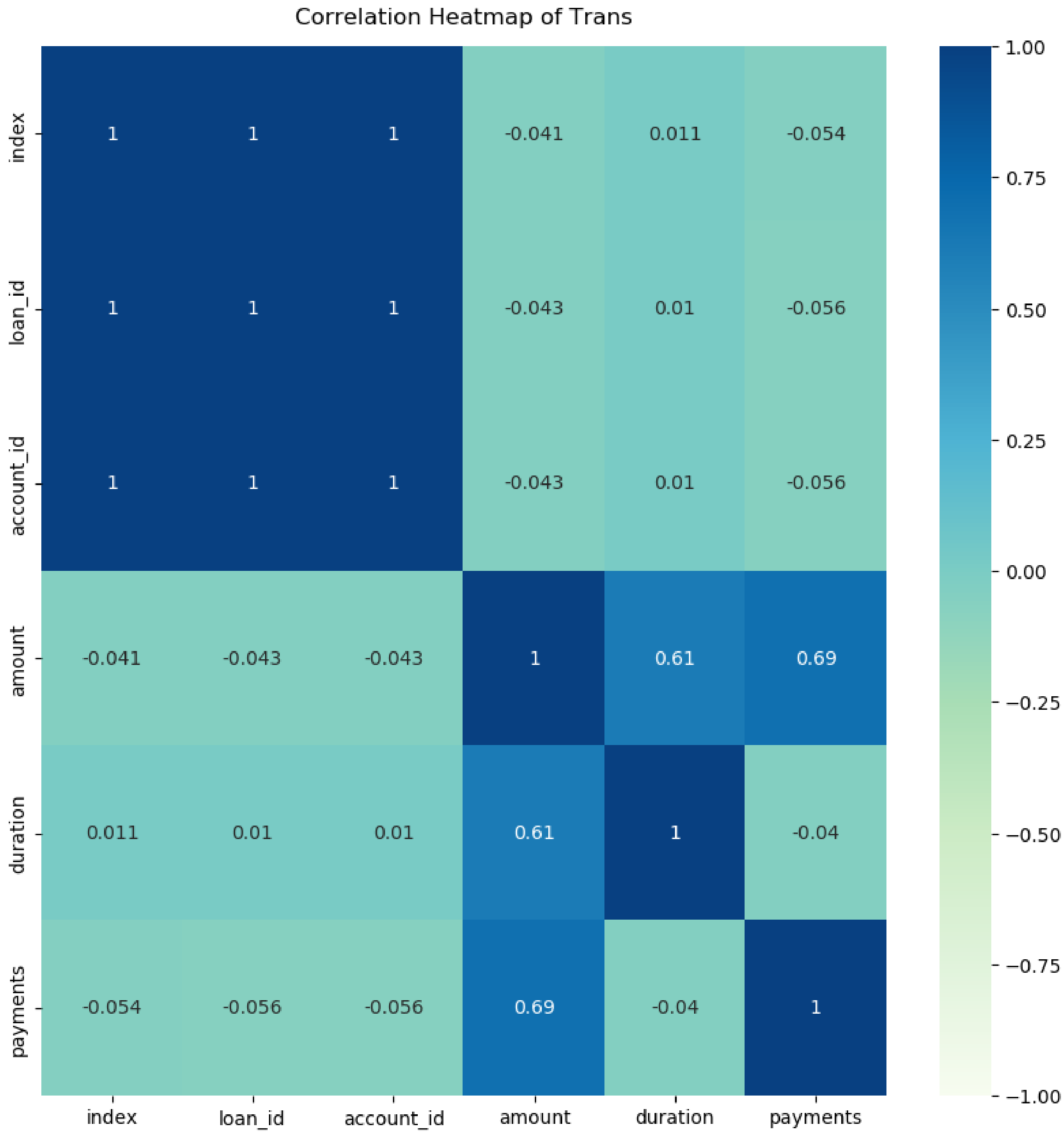
Out [26]:

	index	loan_id	account_id	amount	duration	payments
count	682.000000	682.000000	682.000000	682.000000	682.000000	682.000000
mean	340.500000	6172.466276	5824.162757	151410.175953	36.492669	4190.664223
std	197.020726	682.579279	3283.512681	113372.406310	17.075219	2215.830344
min	0.000000	4959.000000	2.000000	4980.000000	12.000000	304.000000
25%	170.250000	5577.500000	2967.000000	66732.000000	24.000000	2477.000000
50%	340.500000	6176.500000	5738.500000	116928.000000	36.000000	3934.000000
75%	510.750000	6752.500000	8686.000000	210654.000000	48.000000	5813.500000
max	681.000000	7308.000000	11362.000000	590820.000000	60.000000	9910.000000

The .describe() function was used to generate a statistical summary for the numerical columns present within the data. Statistical measurements such as percentiles, mean, and standard deviation were included.

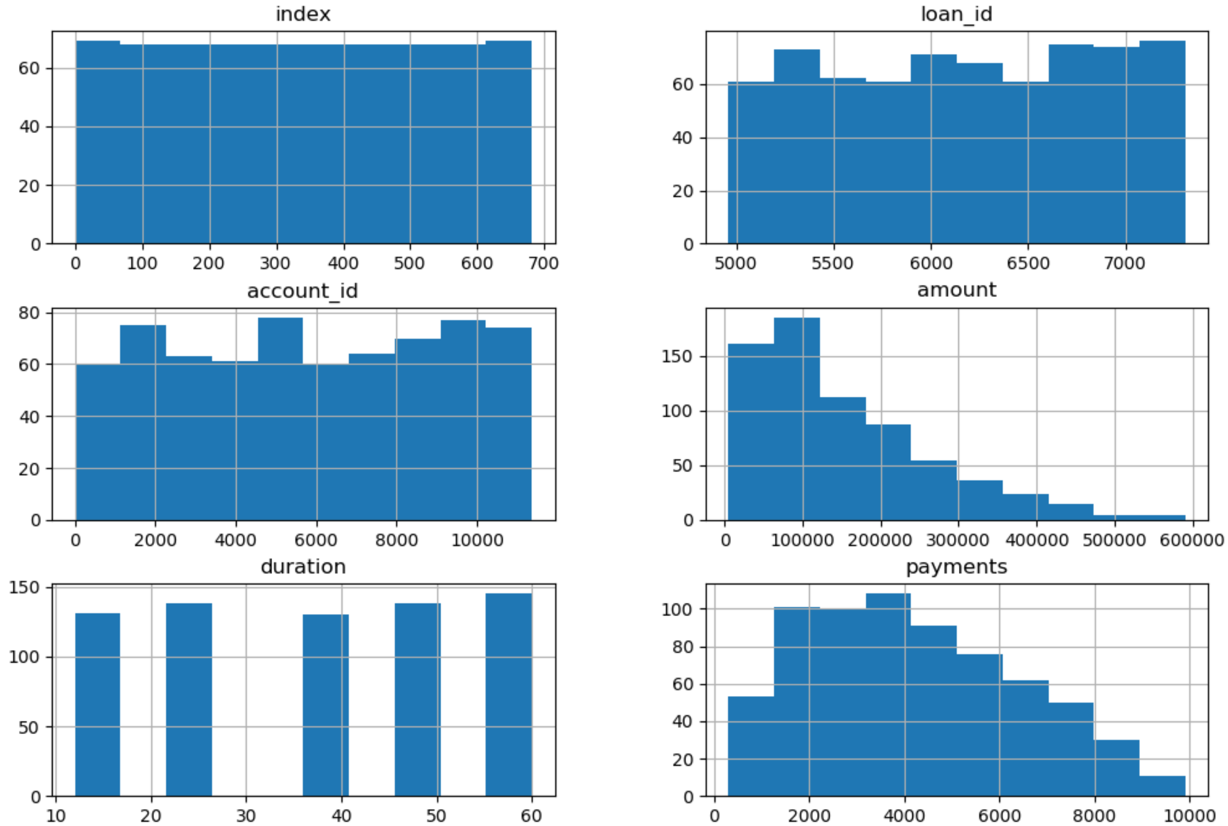
In [27]:

```
# correlation heatmap
plt.figure(figsize=(10, 10))
heatmap = sns.heatmap(loan_df.corr(), vmin=-1, vmax=1, annot=True, cmap="GnBu")
heatmap.set_title('Correlation Heatmap of Trans',
                  fontdict={'fontsize':12}, pad=12);
plt.show()
```



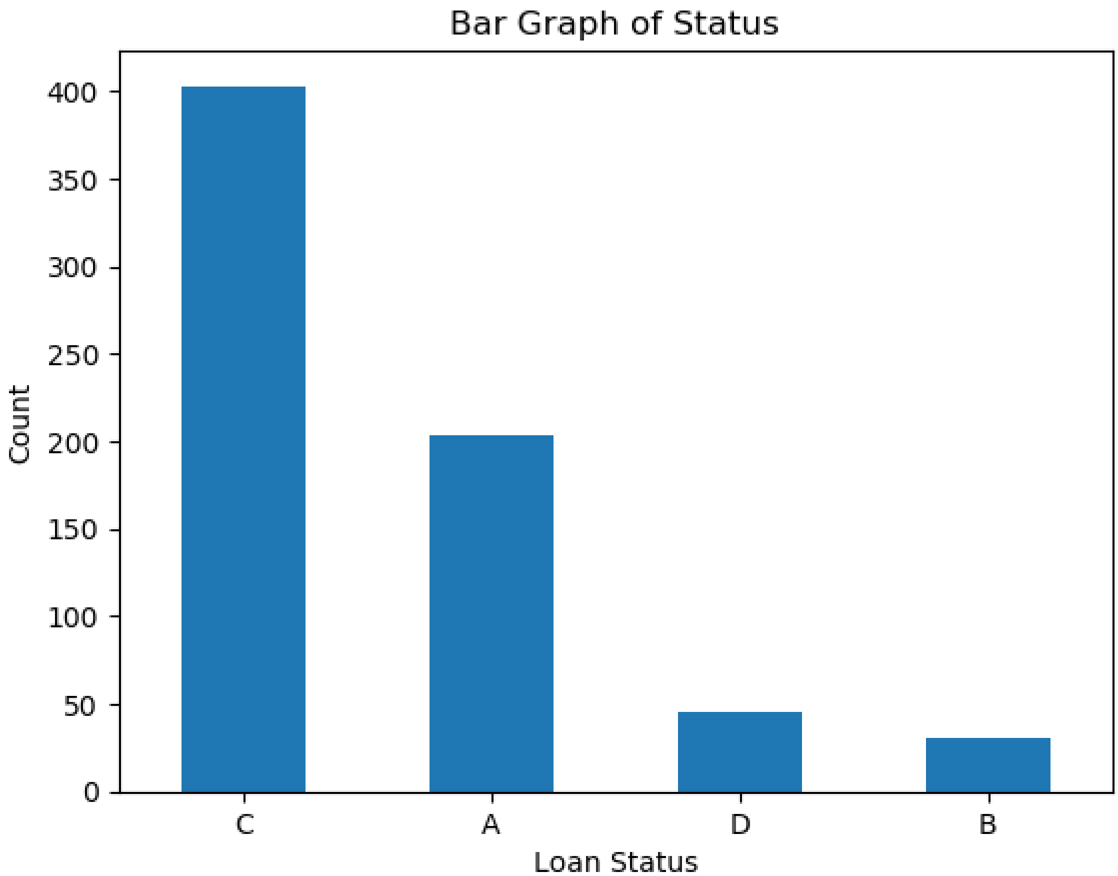
A correlation heatmap was produced in order to determine if there were any multicollinearity concerns. Looking at the correlation heat map, it is evident that there are some concerns of multicollinearity that will need to be addressed while pre-processing the data, as some of the correlation values are greater than 0.7. For example, index and loan_id are highly correlated with a pearson corrleation value of 1.

In [28]:
```
# histogram of numeric variables
loan_df.hist(figsize=[12, 8])
plt.show()
```



Looking at the histograms above, it is evident the variable amounts and payments are right skewed. This indicates that the mean values are greater than the median values resulting in the data being positively skewed. The variables index, duration, account_id, and loan_id are normally distributed. Each of these distributions indicate that during the pre-processing, normalization of the data through scaling and centering of the data should be considered.

In [29]:
```
# bar graph of categorical variable
loan_df.status.value_counts().plot(kind="bar")
plt.title("Bar Graph of Status")
plt.xlabel("Loan Status")
plt.xticks(rotation=0)
plt.ylabel("Count")
plt.show()
```



The loan_status variable is our target variable. The status types C and A indicate a successful loan borrowing interaction, and D and B indicate an unsuccessful loan borrowing interaction. Looking at the bar plot above, it is evident that there is a class imbalance between the successful and unsuccessful loan borrowers. Based on this class imbalance, the data should most likely be split using a stratified approach along with the use of the resample function to mitigate the class imbalance.

Data 4: account

In [30]:
```
# number of rows and columns
account_df.shape
```

Out[30]:
```
(4500, 5)
```

The .shape() function was used in order to determine the number of rows and columns within this dataset. There are a total of 4500 records and 5 columns.

In [31]:
```
# df info
account_df.info()
```

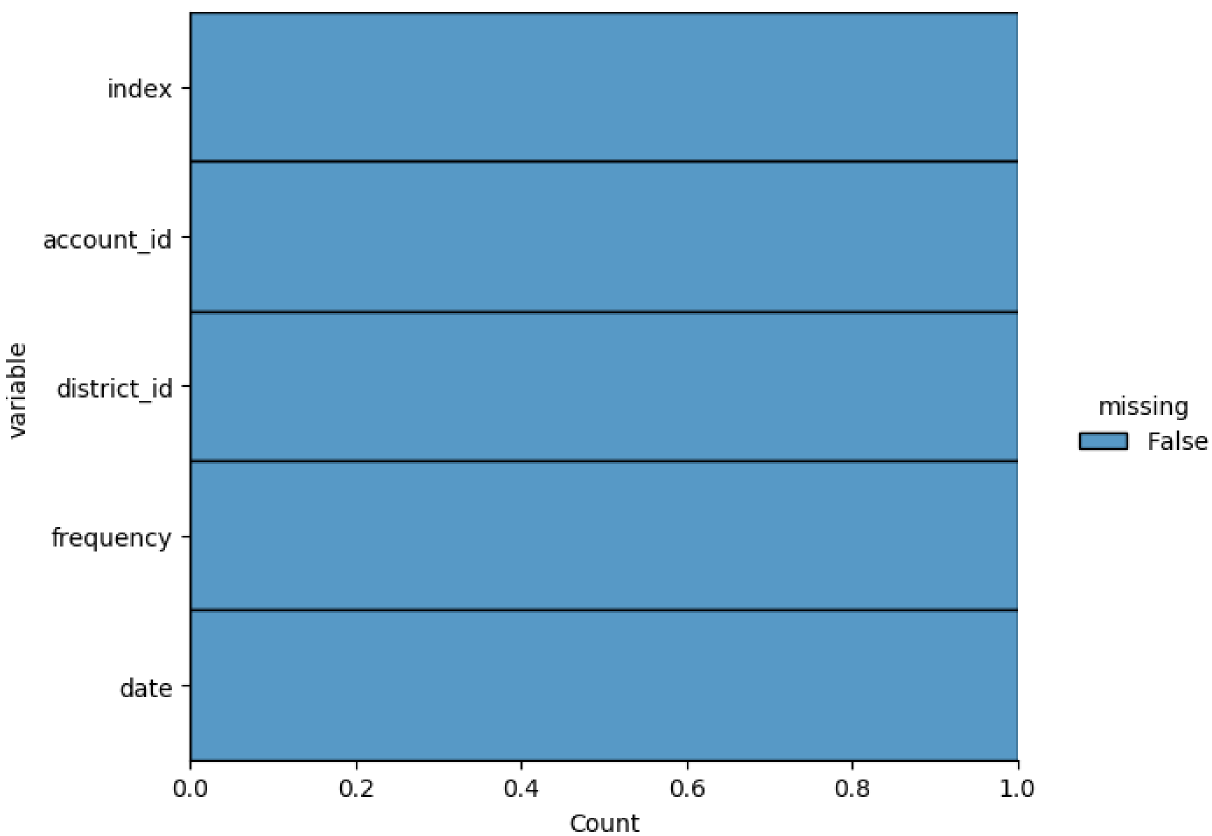
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4500 entries, 0 to 4499
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   index        4500 non-null   int64 
 1   account_id   4500 non-null   int64 
 2   district_id  4500 non-null   int64 
 3   frequency    4500 non-null   object
 4   date         4500 non-null   object
dtypes: int64(3), object(2)
memory usage: 175.9+ KB
```

The .info() function outputs an overview of the dataset. From this overview, the following can be noted: column names, number of non-null present within the dataset, data type, number of the varying dtypes, and memory usage.

In [32]:

```python
# heat map for missing values
plt.figure(figsize=(6,6))
sns.displot(
    data=account_df.isna().melt(value_name="missing"),
    y="variable",
    hue="missing",
    multiple="fill",
    aspect=1.25
)
plt.show()
```

```
<Figure size 600x600 with 0 Axes>
```



A heat map visualization of the missing values was produced. From this visualization, it is evident that there are no missing values that need to be addressed while preprocessing the data.

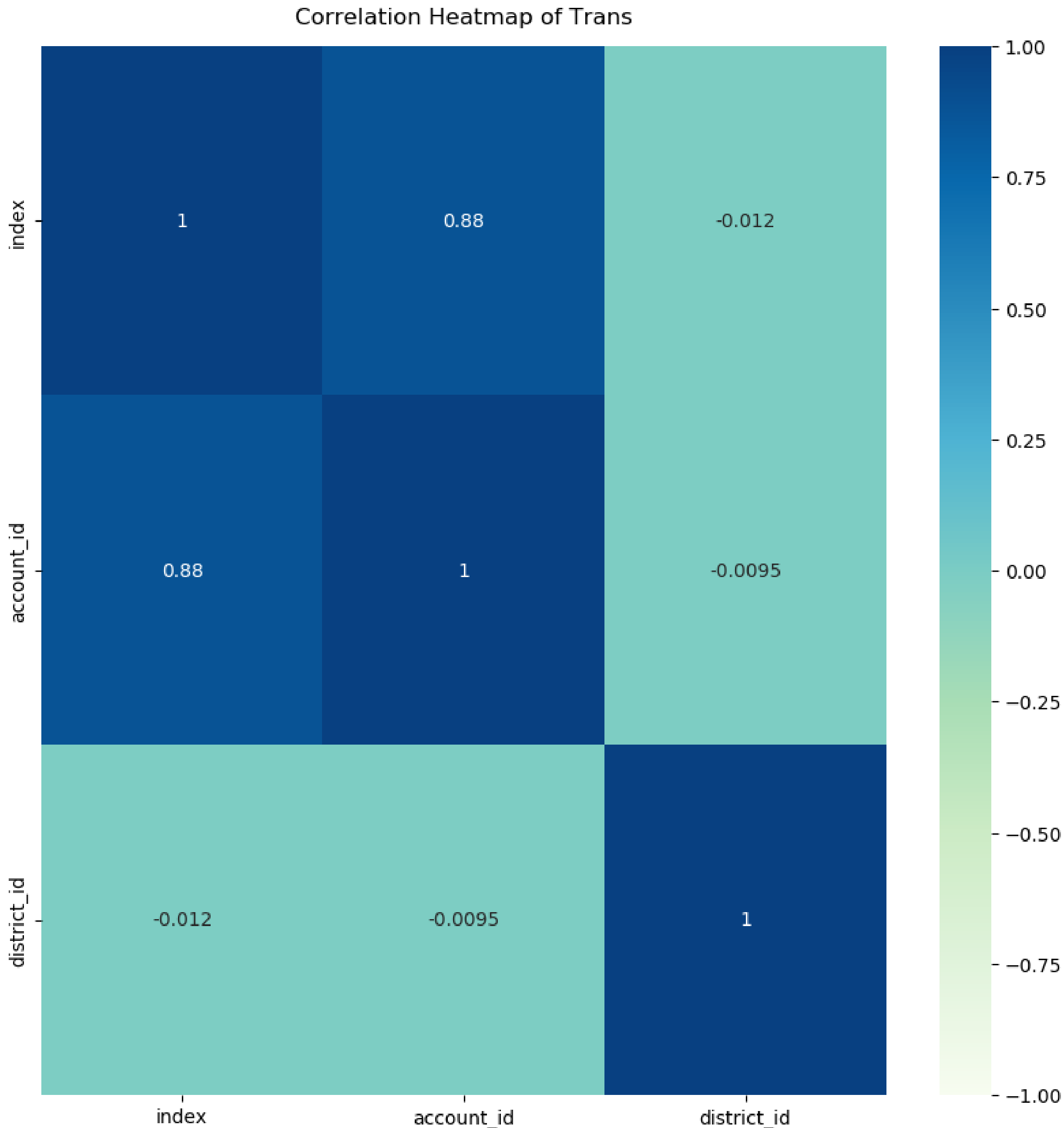
In [33]:
```
# data description
account_df.describe()
```

Out[33]:

	index	account_id	district_id
count	4500.000000	4500.000000	4500.000000
mean	2249.500000	2786.067556	37.310444
std	1299.182435	2313.811984	25.177217
min	0.000000	1.000000	1.000000
25%	1124.750000	1182.750000	13.000000
50%	2249.500000	2368.000000	38.000000
75%	3374.250000	3552.250000	60.000000
max	4499.000000	11382.000000	77.000000

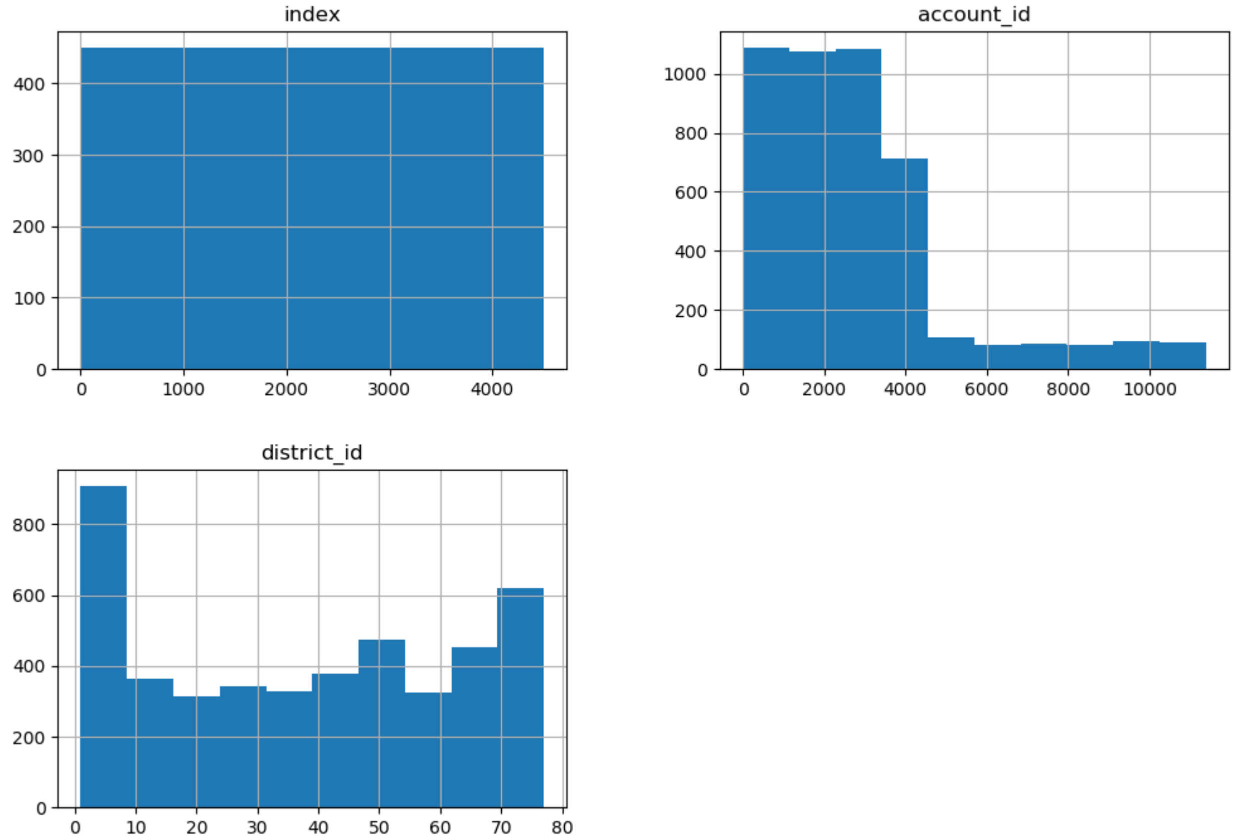
The .describe() function was used to generate a statistical summary for the numerical columns present within the data. Statistical measurements such as percentiles, mean, and standard deviation were included.

In [34]:
```
# correlation heatmap
plt.figure(figsize=(10, 10))
heatmap = sns.heatmap(account_df.corr(), vmin=-1, vmax=1, annot=True, cmap="GnB
heatmap.set_title('Correlation Heatmap of Trans',
                  fontdict={'fontsize':12}, pad=12);
plt.show()
```



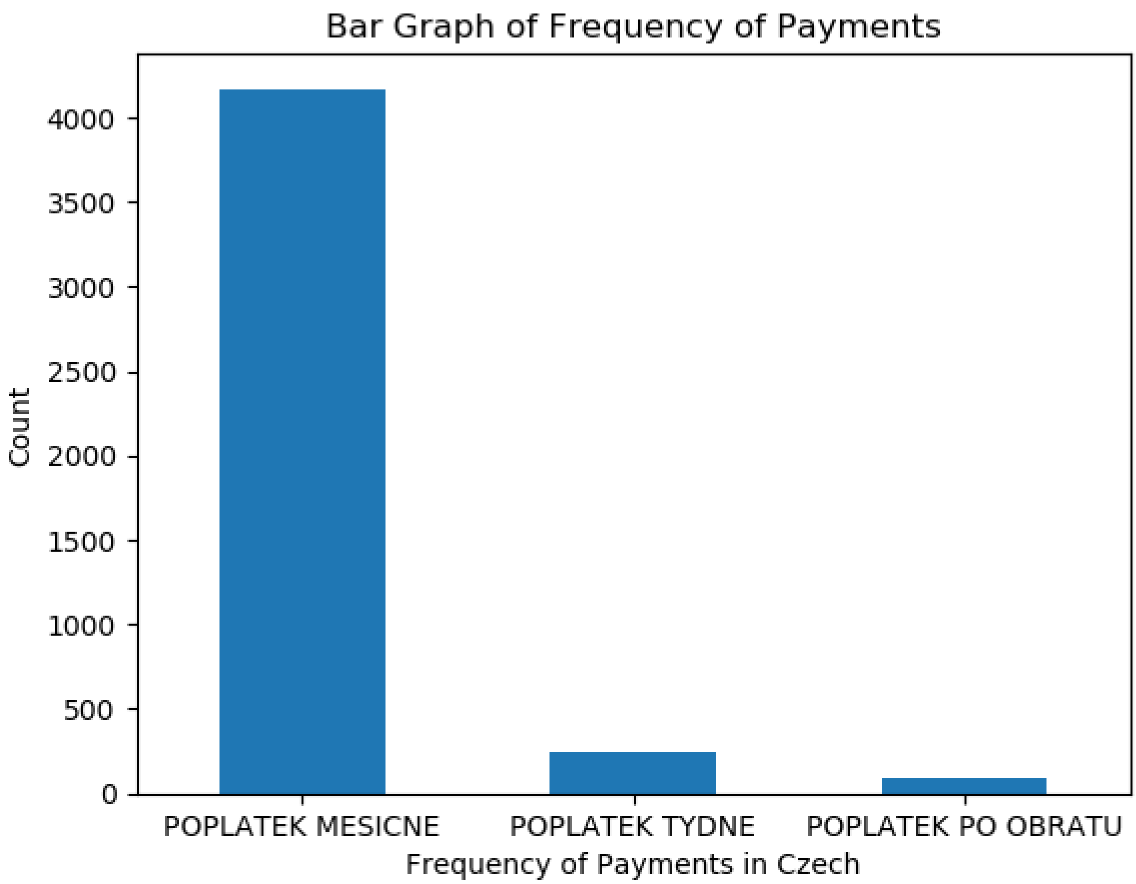
A correlation heatmap was produced in order to determine if there were any multicollinearity concerns. Looking at the correlation heat map, it is evident that there are some concerns of multicollinearity that will need to be addressed while pre-processing the data, as some of the correlation values are greater than 0.7. For example, index and account_id are highly correlated with a pearson corrleation value of 0.88.

In [35]:
```
# histogram of numeric variables
account_df.hist(figsize=[12, 8])
plt.show()
```



Looking at the histograms above, it is evident the variable account_id is right skewed. This indicates that the mean values are greater than the median values resulting in the data being positively skewed. The index column, and district_id are normally distributed. Each of these distributions indicate that during the pre-processing, normalization of the data through scaling and centering of the data should be considered.

In [36]:
```
# bar graph of categorical variable
account_df.frequency.value_counts().plot(kind="bar")
plt.title("Bar Graph of Frequency of Payments")
plt.xlabel("Frequency of Payments in Czech")
plt.xticks(rotation=0)
plt.ylabel("Count")
plt.show()
```



This graph is heavily right skewed. Additionally, it should be noted that the frequency of payments are in Czech. The frequency of payments roughtly translates to monthly payments, weekly payments, and bimonthly payments.

Data Wrangling

In [4]:
```
# database (db) set-up using athena
sess = sagemaker.Session()
bucket = sess.default_bucket()
role = sagemaker.get_execution_role()
region = boto3.Session().region_name
ingest_create_athena_db_passed = False
```

In [5]:
```
# database (db) name set-up
db_name = "SDAloans"
# s3 staging directory
s3_sg_dir = "s3://{0}/athena/staging".format(bucket)
# connection via directory for querying
conn = connect(region_name=region, s3_staging_dir=s3_sg_dir)
```

In [6]:
```
# creating the database (db = SDAloans)
statement = "CREATE DATABASE IF NOT EXISTS {}".format(db_name)
print(statement)
pd.read_sql(statement, conn)
```

```
CREATE DATABASE IF NOT EXISTS SDAloans
```

Out[6]: —

In [7]:
```
# verification of db creation
statement = "SHOW DATABASES"
df_show = pd.read_sql(statement, conn)
df_show.head(3)
```

Out[7]:

	database_name
0	default
1	sdaloans

In [8]:
```
# setting directory to s3 bucket with files
# SDAloans_dir = 's3://ads508loanapproval/datasets/data1'
```

In [9]:
```
# SQL: reading in the 1st dataset=trans.csv as a table (tb) into directory
tb1_name ='trans'
pd.read_sql(f'DROP TABLE IF EXISTS {db_name}.{tb1_name}', conn)

create_table = f"""
CREATE EXTERNAL TABLE IF NOT EXISTS {db_name}.{tb1_name}(
                index int,
                trans_id int,
                account_id int,
                date date,
                type string,
                operation string,
                amount int,
                balance int,
                k_symbol string,
                bank string,
                account int
                )

                ROW FORMAT DELIMITED
                FIELDS TERMINATED BY ','
                LOCATION 's3://ads508loanapproval/datasets/data1/'
                TBLPROPERTIES ('skip.header.line.count'='1')
"""
pd.read_sql(create_table, conn)
pd.read_sql(f'SELECT * FROM {db_name}.{tb1_name} LIMIT 3', conn)
```

Out[9]:

	index	trans_id	account_id	date	type	operation	amount	balance	k_symbol	bank	a
0	515583	839750	2859	None	VYDAJ	VYBER	3600	64639			
1	515584	839751	2859	None	VYDAJ	VYBER	3900	26081			
2	515585	839754	2859	None	VYDAJ	VYBER	6000	29981			

In [10]:
```
# SQL: reading in the 2nd dataset=trans_2.csv as a table (tb) into directory
tb2_name ='trans_2'
pd.read_sql(f'DROP TABLE IF EXISTS {db_name}.{tb2_name}', conn)

create_table = f"""
CREATE EXTERNAL TABLE IF NOT EXISTS {db_name}.{tb2_name}(
                index int,
```

```
                 trans_id int,
                 type string,
                 operation string,
                 amount2 int,
                 balance int
                 )

                 ROW FORMAT DELIMITED
                 FIELDS TERMINATED BY ','
                 LOCATION 's3://ads508loanapproval/datasets/data2/'
                 TBLPROPERTIES ('skip.header.line.count'='1')
"""
pd.read_sql(create_table, conn)
pd.read_sql(f'SELECT * FROM {db_name}.{tb2_name} LIMIT 3', conn)
```

Out[10]:

	index	trans_id	type	operation	amount2	balance
0	0	289	Credit	Collection	0	0
1	1	290	Credit	Collection	0	0
2	2	291	Credit	Collection	0	0

In [11]:
```
# SQL: reading in the 3rd dataset=loan.csv as a table (tb) into directory
tb3_name ='loan'
pd.read_sql(f'DROP TABLE IF EXISTS {db_name}.{tb3_name}', conn)

create_table = f"""
CREATE EXTERNAL TABLE IF NOT EXISTS {db_name}.{tb3_name}(
                 index int,
                 loan_id int,
                 account_id int,
                 date date,
                 amount int,
                 duration int,
                 payments float,
                 status string
                 )

                 ROW FORMAT DELIMITED
                 FIELDS TERMINATED BY ','
                 LOCATION 's3://ads508loanapproval/datasets/data3/'
                 TBLPROPERTIES ('skip.header.line.count'='1')
"""
pd.read_sql(create_table, conn)
pd.read_sql(f'SELECT * FROM {db_name}.{tb3_name} LIMIT 3', conn)
```

Out[11]:

	index	loan_id	account_id	date	amount	duration	payments	status
0	0	4959	2	None	80952	24	3373.0	A
1	1	4961	19	None	30276	12	2523.0	B
2	2	4962	25	None	30276	12	2523.0	A

In [12]:
```
# SQL: reading in the 4th dataset=account.csv as a table (tb) into directory
tb4_name ='account'
pd.read_sql(f'DROP TABLE IF EXISTS {db_name}.{tb4_name}', conn)

create_table = f"""
```

```
CREATE EXTERNAL TABLE IF NOT EXISTS {db_name}.{tb4_name}(
                index int,
                account_id int,
                district_id int,
                frequency string,
                date date
                )

                ROW FORMAT DELIMITED
                FIELDS TERMINATED BY ','
                LOCATION 's3://ads508loanapproval/datasets/data4/'
                TBLPROPERTIES ('skip.header.line.count'='1')
"""
pd.read_sql(create_table, conn)
pd.read_sql(f'SELECT * FROM {db_name}.{tb4_name} LIMIT 3', conn)
```

Out[12]:

	index	account_id	district_id	frequency	date
0	0	1	18	POPLATEK MESICNE	None
1	1	2	1	POPLATEK MESICNE	None
2	2	3	5	POPLATEK MESICNE	None

In [13]:

```
# verification of db creation + storing
statement = "SHOW DATABASES"
df_show = pd.read_sql(statement, conn)
df_show.head(3)
```

Out[13]:

	database_name
0	default
1	sdaloans

In [14]:

```
if db_name in df_show.values:
    ingest_create_athena_db_passed = True
```

In [15]:

```
%store ingest_create_athena_db_passed
```

```
Stored 'ingest_create_athena_db_passed' (bool)
```

In [16]:

```
# SQL: merging tb 1-4 + saving as a df + renaming + removal of unnecessary attr
df_i=pd.read_sql(f'SELECT * FROM (SELECT l.amount as loan_amt, l.duration,\
l.payments as pay_amt, l.status, a.frequency as pay_freq,\
t.amount as avg_pay_amt, t.balance, tr.operation FROM {db_name}.{tb3_name} as l
 LEFT JOIN {db_name}.{tb4_name} as a\
 on l.account_id = a.account_id\
 LEFT JOIN {db_name}.{tb1_name} as t\
 on l.account_id = t.account_id\
 LEFT JOIN {db_name}.{tb2_name} as tr\
 on t.trans_id = tr.trans_id)', conn)
```

In [17]:

```
# check
df_i.head(3)
```

Out[17]:

	loan_amt	duration	pay_amt	status	pay_freq	avg_pay_amt	balance	operation
0	30276	12	2523.0	B	POPLATEK MESICNE	36	10840	None
1	30276	12	2523.0	B	POPLATEK MESICNE	5	8211	cw
2	30276	12	2523.0	B	POPLATEK MESICNE	35	8188	None

Secondary Exploratory Data Analysis (EDA)

In [18]:

```
# df copy
df = df_i.copy()
```

Above is the creation of a copy of the original merged dataframe. This was done in order to ensure that while working an uneffected dataframe copy can be referred back to.

In [52]:

```
# number of rows and columns
df.shape
```

Out[52]:

```
(184356, 8)
```

In [53]:

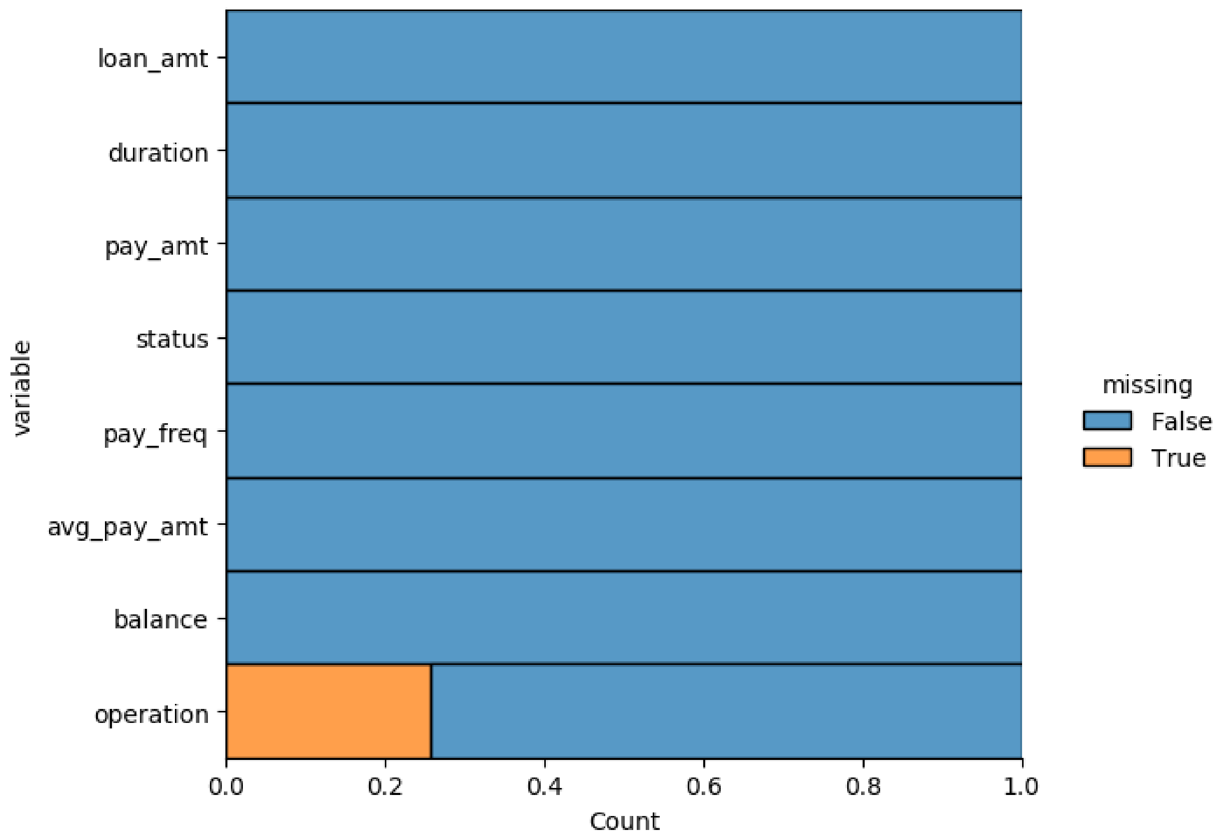
```
# df info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 184356 entries, 0 to 184355
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype  
---  ------       --------------   -----  
 0   loan_amt     184356 non-null  int64  
 1   duration     184356 non-null  int64  
 2   pay_amt      184356 non-null  float64
 3   status       184356 non-null  object 
 4   pay_freq     184356 non-null  object 
 5   avg_pay_amt  184356 non-null  int64  
 6   balance      184356 non-null  int64  
 7   operation    137187 non-null  object 
dtypes: float64(1), int64(4), object(3)
memory usage: 11.3+ MB
```

In [54]:

```
# heat map for missing values
plt.figure(figsize=(3,3))
sns.displot(
    data=df.isna().melt(value_name="missing"),
    y="variable",
    hue="missing",
    multiple="fill",
    aspect=1.25
)
plt.show()
```

```
<Figure size 300x300 with 0 Axes>
```

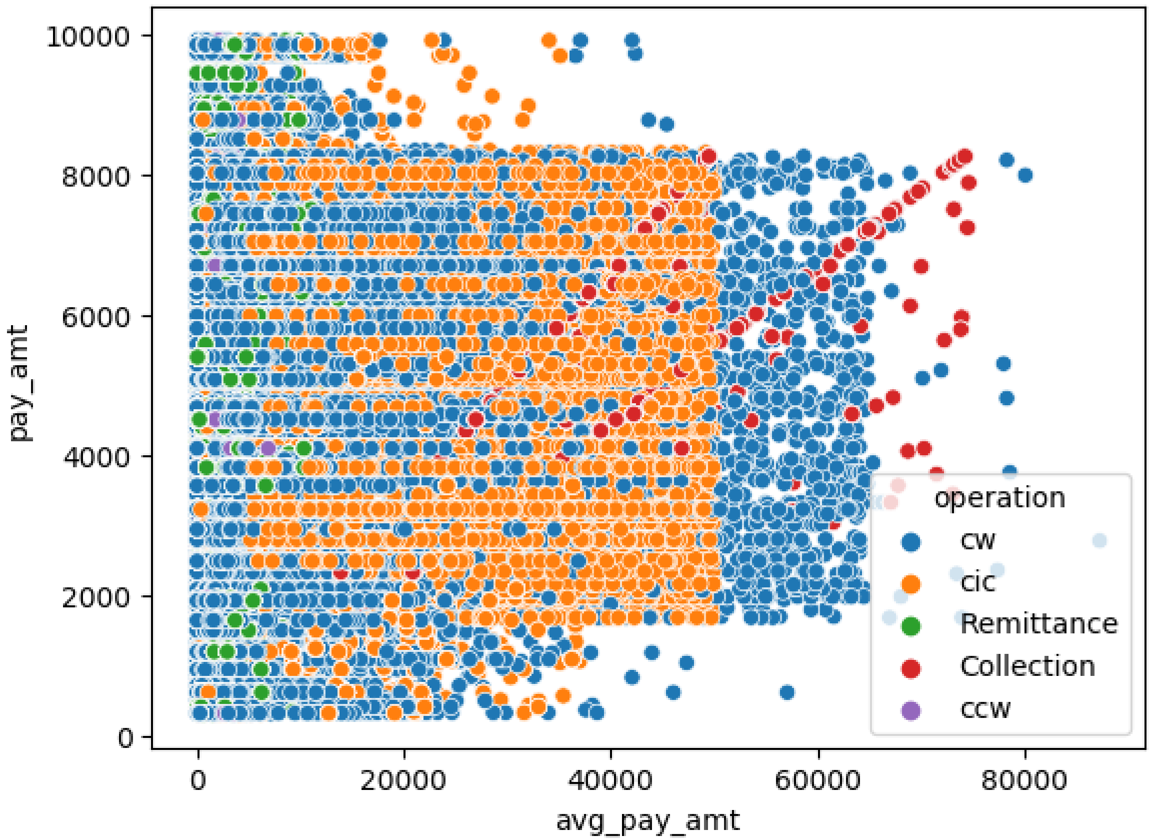


In [55]:
```
# data description
df.describe()
```

Out[55]:

	loan_amt	duration	pay_amt	avg_pay_amt	balance
count	184356.000000	184356.000000	184356.000000	184356.000000	184356.000000
mean	146592.543318	35.685804	4178.310459	8593.248389	45641.627335
std	110100.818539	17.216874	2207.773430	11912.346644	24980.558236
min	4980.000000	12.000000	304.000000	0.000000	-19310.000000
25%	65184.000000	24.000000	2477.000000	233.000000	27475.750000
50%	111384.000000	36.000000	3900.000000	3900.000000	41072.000000
75%	202848.000000	48.000000	5900.000000	11200.000000	59649.250000
max	590820.000000	60.000000	9910.000000	87300.000000	209637.000000

In [56]:
```
sns.scatterplot(data=df, x="avg_pay_amt", y="pay_amt", hue="operation")
plt.show()
```

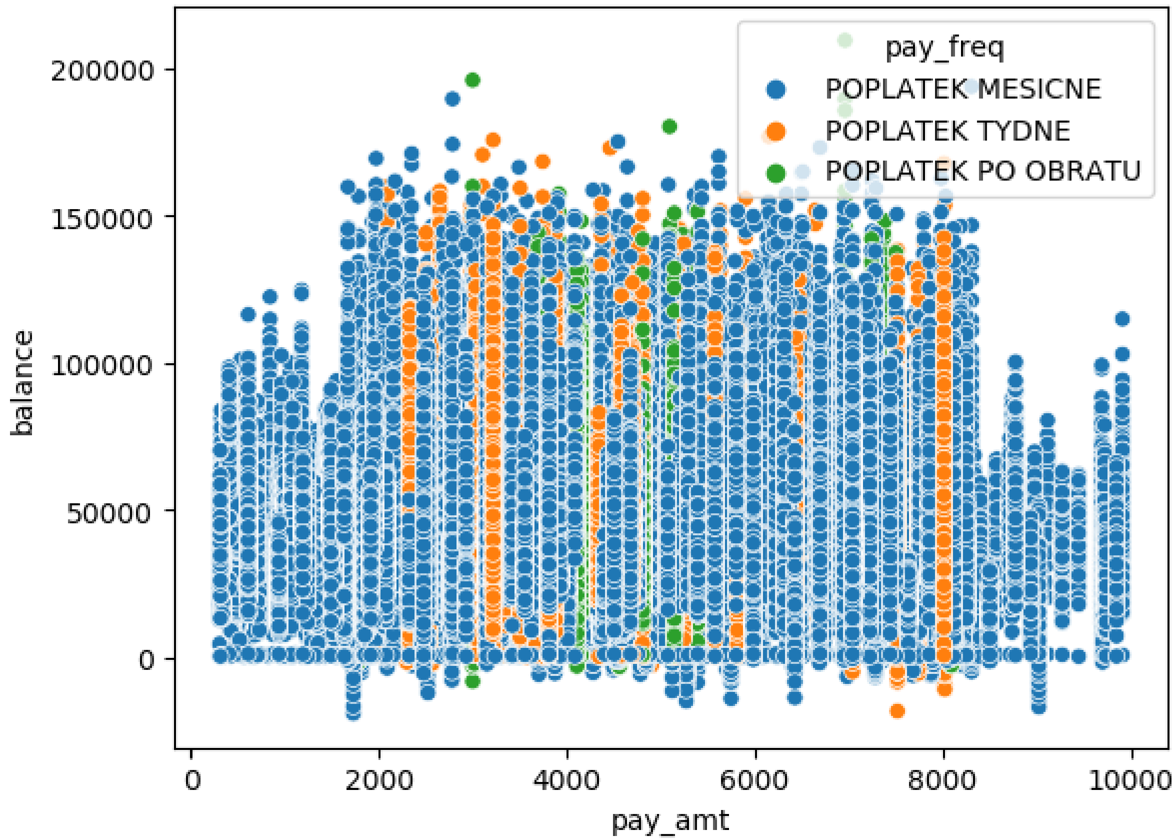


From the plot above, it is evident that clients that that have paid, on average, 40000 dollars to 50000 dollars have made payments using credit card payments. The credit cards used to pay back the SDA bank are associated with another bank. It can also be noted that clientts that have paid, on average, 50000 dollars to 65000 dollars towards their loans have opted to use bank-to-bank transfers.

In [57]:
```
px.histogram(df, y='duration', color='status',
             barmode = 'group',
              color_discrete_sequence=px.colors.sequential.Blugrn,
             width=600, height =600)
```

Looking at the plot above, it is evident that the most amount of non recommendable clients are in the loan contracts that are shorter. For example, loans that are to be paid in approximately fifteen months have a higher level of non recommendable clients than loans that can be paid in sixty-four months. This may indicate that shorter loan duration contracts are of higher risk. Another thing to note is that regardless of the duration of the loan, most of the clients are in good standing with the SDA bank.

In [58]:
```
sns.scatterplot(data=df, x="pay_amt", y="balance", hue="pay_freq")
plt.show()
```



From the plot above, it is evident that the majority of clients, regardless of the amount of money that they borrowed, are required to make monthly payments towards their loans.

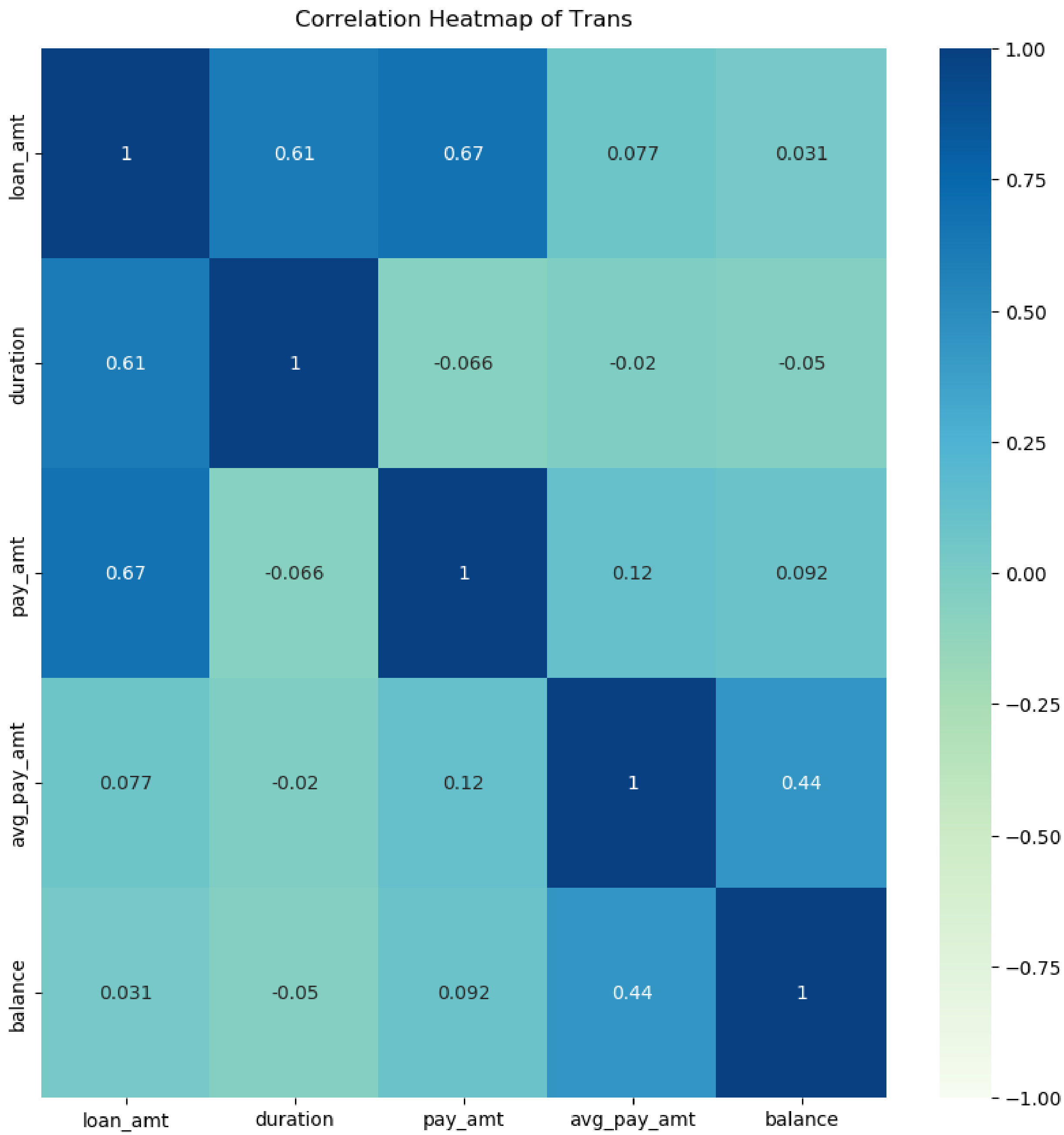
In [59]:
```
px.histogram(df, x='pay_freq', color='status',
            barmode = 'group',
             color_discrete_sequence=px.colors.sequential.Blugrn,
            width=600, height =600)
```

Looking at the plot above, it is evident that most of the clients, regardless of required payment frequency, are in good standing with the bank. It is evident, based on proportions, that those that are required to pay twice a month have the highest rate of non recommendable clients. This may indicate that payment contracts that reflect bimonthly payments are of high risk. Further analysis should be conducted in order to verify this conjecture.

Data Preparation

Multicollinearity Checks + Corrections

In [60]:
```
# correlation heat map to check for multicollinearity
plt.figure(figsize=(10, 10))
heatmap = sns.heatmap(df.corr(), vmin=-1, vmax=1, annot=True, cmap="GnBu")
heatmap.set_title('Correlation Heatmap of Trans',
                  fontdict={'fontsize':12}, pad=12);
plt.show()
```



No concerns of multicollinearity that need to be corrected for as none of the coefficients exceed 0.7.

Recoding of Categorical Data

In [28]:
```
# checking the values w/in the target var = status
df = df_i.copy()
df.status.unique()
```

Out[28]:
```
array(['B', 'A', 'D', 'C'], dtype=object)
```

In [29]:
```
# dummy coding the target variable "status" manually
df['status'] = df['status'].map({'A': 1, 'B': 0,'C': 1,'D': 0})
```

```
# check
df.status.unique()
```

Out[29]:
```
array([0, 1])
```

`status` : status of loan payment

- 'A': contract ended; loan paid
- 'B': contract ended; loan unpaid
- 'C': contract in progress; client on track to pay off loan
- 'D': contract in progress; client in debt

Based on these indicators of A, B, C, and D, it is evident that A and C indicate good loan candidates while B and D indicate poor loan candidates. The `status` column was therefore recoded to reflect this by indicating A and C as 1 and B and D as 0.

In [38]:
```
# handling missing values
df = df.dropna()
```

In [39]:
```
# dummy coding a predictor variable "operation" using one-hot-encoding
df_cat = pd.concat([df, pd.get_dummies(df['operation'], prefix='op')],axis=1)
```

In [40]:
```
# dummy coding a predictor variable "pay_freq" using one-hot-encoding
df_cat = pd.concat([df_cat, pd.get_dummies(df_cat['pay_freq'], prefix='pay')],a
# check
df_cat.head(2)
```

Out[40]:

	loan_amt	duration	pay_amt	status	pay_freq	avg_pay_amt	balance	operation	op_Colle
1	30276	12	2523.0	0	POPLATEK MESICNE	5	8211	cw	
4	30276	12	2523.0	0	POPLATEK MESICNE	7	7507	cw	

In [41]:
```
# dropping "operation" and "payment_freq"
df_cat.drop(['operation', 'pay_freq'], axis=1, inplace = True)

# drop check
print(df_cat.columns)
```

```
Index(['loan_amt', 'duration', 'pay_amt', 'status', 'avg_pay_amt', 'balance',
       'op_Collection', 'op_Remittance', 'op_ccw', 'op_cic', 'op_cw',
       'pay_POPLATEK MESICNE', 'pay_POPLATEK PO OBRATU', 'pay_POPLATEK TYDN
E'],
      dtype='object')
```

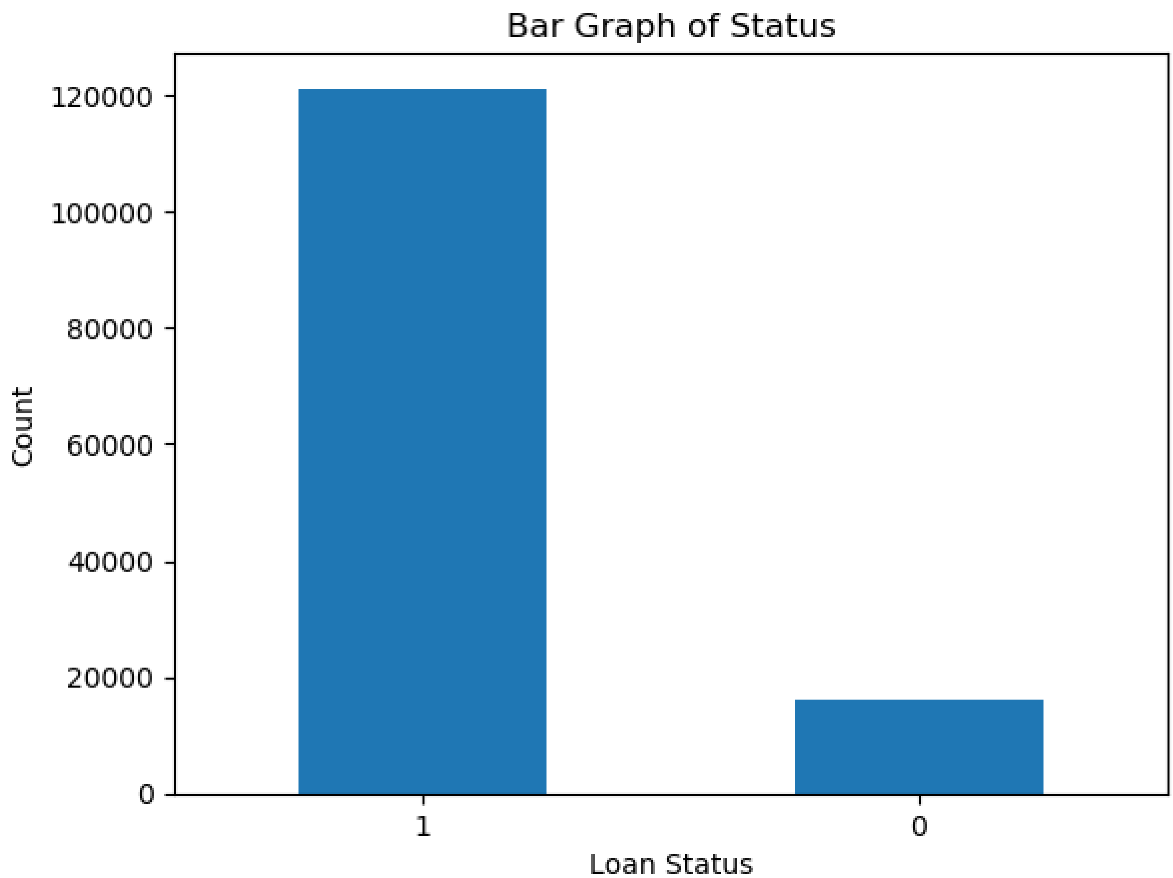
In [42]:
```
# correcting column names
df_cat.rename(columns = {'op_Collection':'op_cb',
                          'op_Remittance':'op_cp',
                          'op_ccw':'op_ccp',
                         'op_cic':'op_ccb',
                         'op_cw':'op_bb',
                         'pay_POPLATEK MESICNE':'pay_mo',
                         'pay_POPLATEK PO OBRATU':'pay_bimo',
```

```
                              'pay_POPLATEK TYDNE':'pay_wk'}, inplace = True)
# renaming check
print(df_cat.columns)
```

```
Index(['loan_amt', 'duration', 'pay_amt', 'status', 'avg_pay_amt', 'balance',
       'op_cb', 'op_cp', 'op_ccp', 'op_ccb', 'op_bb', 'pay_mo', 'pay_bimo',
       'pay_wk'],
      dtype='object')
```

Bias Detection

In [43]:
```
# visualization of target variable status
df_cat.status.value_counts().plot(kind="bar")
plt.title("Bar Graph of Status")
plt.xlabel("Loan Status")
plt.xticks(rotation=0)
plt.ylabel("Count")
plt.show()
```



Looking at the bar chart above, it is evident that there is the bias of class imbalnce present within this dataset. This is the case as the two outcomes are clearly unbalanced. The way in which this could be handled is through oversampling the minority group or undersampling the majoirity group.

Mitigation of Class Imbalance

For the purpose of this project, the way in which class imbalance was chosen to be handeled was through oversampling the minority class. Oversampling can aid with creating a more balanced dataset so that the results of the model predictions are more accurate. Essentially, oversampling was chosen over undersampling as oversampling will ensure that no information is lost. The technique of using the RandomOverSampler function over other oversampling techniques such as SMOTE oversmapling is because it is considered more roboust in terms of model results than SMOTE oversampling (Chadha, 2022).

In [44]:
```
# oversampling miniority class to handle the imbalance
oversample = RandomOverSampler(sampling_strategy='minority')
col = "status"
x = df_cat.loc[:, df_cat.columns !=col]
y = df_cat["status"]
x_over, y_over = oversample.fit_resample(x,y)
df_cat2 = pd.concat([pd.DataFrame(x_over),
                 pd.DataFrame(y_over)], axis=1)
df_cat2.columns = df_cat.columns
```

In [45]:
```
# oversampling + rebalancing check
z_ct = df_cat2['status'].value_counts()[0]
o_ct = df_cat2['status'].value_counts()[1]
z_o = z_ct + o_ct
z_ct1 = df_cat['status'].value_counts()[0]
o_ct1 = df_cat['status'].value_counts()[1]
z_o1 = z_ct1 + o_ct1
print('Nonrecommendable Loan Candidates(%) before Oversampling:', round(z_ct1/z
print('Recommendable Loan Candidates(%) before Oversampling:', round(o_ct1/z_o1
print('Nonrecommendable Loan Candidates(%) after Oversampling:', round(z_ct/z_c
print('Recommendable Loan Candidates(%) after Oversampling:', round(o_ct/z_o,2)
```

```
Nonrecommendable Loan Candidates(%) before Oversampling: 0.12
Recommendable Loan Candidates(%) before Oversampling: 0.88
Nonrecommendable Loan Candidates(%) after Oversampling: 0.42
Recommendable Loan Candidates(%) after Oversampling: 0.58
```

Looking at the results above based on the RandomOverSampler, oversampling technique, it is evident that the oversampling technique really helped balance out the dataset. Initially, the data was drastically imbalanced with 12:88 percent non recommendable to recommendable loan candidates distribution within the dataset. Now, after oversampling, there is a 42:58 percent non recommendable to recommendable loan candidate distribution within the dataset.

Columns:

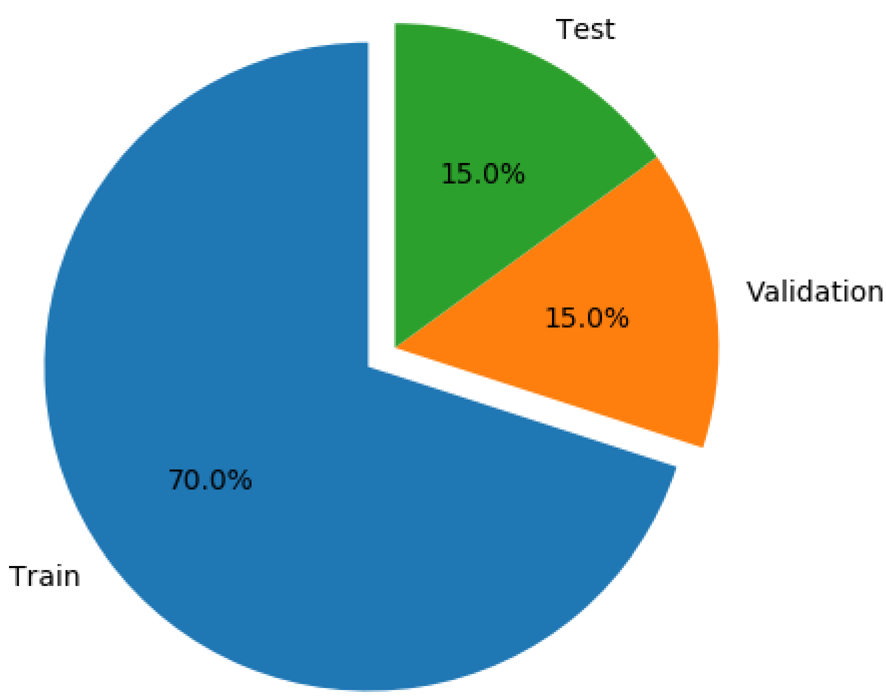
- `loan_amt:` amount of money borrowed from the bank
- `duration:` contract-based time allotment in which the loan is expected to be paid
- `pay_amt:` amount to be paid to the bank each month, week, or twice a month
- `status:` loan status (completed: paid or debt, in-progress: paid or debt)
- `pay_mo:` indicates if the client if required to pay once a month
- `pay_wk:` indicates if the client is required to pay once a week
- `pay_bimo:` indicates if the client is required to pay twice a month

- `avg_pay_amt:` average amount of money paid by client
- `balance:` loan balance
- `op_ccp:` credit card payment
- `op_cp:` cash payment
- `op_ccb:` credit card payment through another bank
- `op_cb:` cash payment through another bank
- `op_bb:` bank to bank transfer

Data Splitting

In [48]:
```
# splitting data into train, test, and validation sets
df3, df4 = train_test_split(df_cat, train_size = 67854, random_state = 902)
x_train, y_train, x_valid, y_valid, x_test, y_test = train_valid_test_split(
    df3, target = 'status', train_size=0.7, valid_size=0.15, test_size=0.15
)
```

In [49]:
```
# pie visualization of splits
labels = ["Train", "Validation", "Test"]
sizes = [len(x_train.index), len(x_valid.index), len(x_test.index)]
exp = (0.1, 0, 0)
fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=exp, labels=labels, autopct="%1.1f%%", startangle=90)
ax1.axis("equal")
plt.show()
```



In [50]:
```
# checking dimensions
print('x_train:', x_train.shape, 'y_train:', y_train.shape)
print('x_valid:', x_valid.shape, 'y_valid:', y_valid.shape)
print('x_test:', x_test.shape, 'y_test:', y_test.shape)
```

```
x_train: (47497, 13) y_train: (47497,)
x_valid: (10178, 13) y_valid: (10178,)
x_test: (10179, 13) y_test: (10179,)
```

In [51]:

```
# normalizing numeric data based on the train
scaler = preprocessing.StandardScaler()
scaler.fit_transform(df_cat2[['loan_amt', 'duration', 'pay_amt', 'avg_pay_amt',

# combining the normalized values to the origional dataframe
tNorm = pd.concat([pd.DataFrame(scaler.fit_transform(df_cat2[['loan_amt','durat
                                columns=['z_loan_amt','z_duration', 'z_pay_amt'
                   df_cat2[['pay_mo', 'pay_wk', 'pay_bimo', 'op_ccp', 'op_cp',
                          'op_cb', 'op_bb']]], axis = 1)

# setting the normalized values to the train, valid and test sets
trainNorm = tNorm.iloc[x_train.index]
validNorm = tNorm.iloc[x_valid.index]
testNorm = tNorm.iloc[x_test.index]
```

In [52]:

```
# double checking dimensions
print('x_train:', x_train.shape, 'y_train:', y_train.shape)
print('x_valid:', x_valid.shape, 'y_valid:', y_valid.shape)
print('x_test:', x_test.shape, 'y_test:', y_test.shape)
```

```
x_train: (47497, 13) y_train: (47497,)
x_valid: (10178, 13) y_valid: (10178,)
x_test: (10179, 13) y_test: (10179,)
```

Modeling

Models

In [44]:

```
# Random Forest Model
rf = RandomForestClassifier(max_depth=2, random_state=902)
rf.fit(x_train, y_train)
```

Out[44]:

```
RandomForestClassifier(max_depth=2, random_state=902)
```

In [45]:

```
# Random Forest Model Prediction on Validation Set
rf_pred = rf.predict(x_valid)
```

In [47]:

```
# Naive Bayes Model
nb = GaussianNB()
nb.fit(x_train, y_train)
```

Out[47]:

```
GaussianNB()
```

In [48]:

```
# Naive Bayes Model Prediction on Validation Set
nb_pred = nb.predict(x_valid)
```

In [50]:

```
# K-Nearest Neighbor (KNN)  Model
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(x_train, y_train)
```

Out[50]:

```
KNeighborsClassifier()
```

In [51]:
```
# KNN Model Prediction on Validation Set
knn_pred = knn.predict(x_valid)
```

In [54]:
```
# Logistic Regression Model
log = LogisticRegression(random_state=902)
log.fit(x_train, y_train)
```

Out[54]:
```
LogisticRegression(random_state=902)
```

In [55]:
```
# Logistic Regression Model Prediction on Validation Set
log_pred = log.predict(x_valid)
```

Model Evaluations & Comparisons

In [56]:
```
# Random Forest Model Evals
print(accuracy_score(y_valid, rf_pred))
print('Classification Report \n',
       classification_report(y_valid, rf_pred))
```

```
0.8848496757712714
Classification Report
               precision    recall  f1-score   support

           0       1.00      0.01      0.01      1180
           1       0.88      1.00      0.94      8998

    accuracy                           0.88     10178
   macro avg       0.94      0.50      0.48     10178
weighted avg       0.90      0.88      0.83     10178
```

In [57]:
```
# Naive Bayes Model Evals - Confusion Matrix
print(accuracy_score(y_valid, nb_pred))
print('Classification Report \n',
       classification_report(y_valid, nb_pred))
```

```
0.8665749656121046
Classification Report
               precision    recall  f1-score   support

           0       0.33      0.14      0.20      1180
           1       0.90      0.96      0.93      8998

    accuracy                           0.87     10178
   macro avg       0.61      0.55      0.56     10178
weighted avg       0.83      0.87      0.84     10178
```

In [64]:
```
# KNN Model Evals - Confusion Matrix
print(accuracy_score(y_valid, knn_pred))
print('Classification Report \n',
       classification_report(y_valid, knn_pred))
```

```
0.9191393201021811
Classification Report
               precision    recall  f1-score   support

           0       0.72      0.49      0.58      1180
           1       0.94      0.98      0.96      8998

    accuracy                           0.92     10178
   macro avg       0.83      0.73      0.77     10178
weighted avg       0.91      0.92      0.91     10178
```

In [59]:
```python
# Logistic Regression Model Evals - Confusion Matrix
print(accuracy_score(y_valid, log_pred))
print('Classification Report \n',
       classification_report(y_valid, log_pred))
```

```
0.8852426802908233
Classification Report
               precision    recall  f1-score   support

           0       0.54      0.07      0.12      1180
           1       0.89      0.99      0.94      8998

    accuracy                           0.89     10178
   macro avg       0.72      0.53      0.53     10178
weighted avg       0.85      0.89      0.84     10178
```

In [60]:
```python
# Random Forest, Naive Bayes, and KNN Model Comparisons
fig, (ax1, ax2) = plt.subplots(1, 2) # setting up the two plots
fig.suptitle('Model Comparison', fontsize=16, fontweight='bold')
fig.set_figheight(7)
fig.set_figwidth(14)
fig.set_facecolor('white')

barWidth = 0.2 # plot 1 = metric comparison
rf_score = [accuracy_score(y_valid,rf_pred), precision_score(y_valid, rf_pred),

nb_score = [accuracy_score(y_valid,nb_pred), precision_score(y_valid, nb_pred),

knn_score = [accuracy_score(y_valid,knn_pred), precision_score(y_valid, knn_pre

log_score = [accuracy_score(y_valid,log_pred), precision_score(y_valid, log_pre


r1 = np.arange(len(rf_score)) # bar set-up on x-axis
r2 = [x + barWidth for x in r1]
r3 = [x + barWidth for x in r2]
r4 = [x + barWidth for x in r3]

ax1.bar(r1, rf_score, width=barWidth, edgecolor='white', label='Random Forest')
ax1.bar(r2, nb_score, width=barWidth, edgecolor='white', label='Naive Bayes')
ax1.bar(r3, knn_score, width=barWidth, edgecolor='white', label='K-Nearest Neig
ax1.bar(r4, log_score, width=barWidth, edgecolor='white', label='Logistic Regre

ax1.set_xlabel('Metrics', fontweight='bold') # x/y-axis set-up
labels = ['Accuracy', 'Precision', 'Recall', 'F1']
ax1.set_xticks([r + (barWidth * 1.5) for r in range(len(rf_score))], )
ax1.set_xticklabels(labels)
```

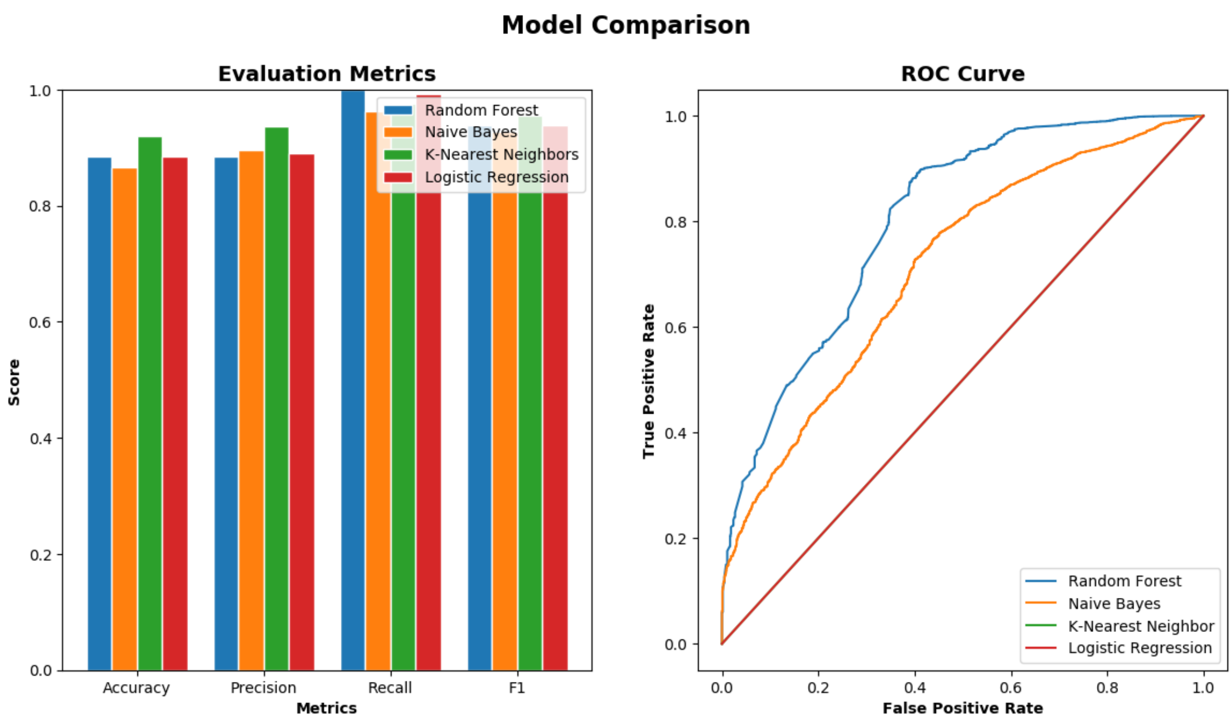
```
ax1.set_ylabel('Score', fontweight='bold')
ax1.set_ylim(0, 1)

ax1.set_title('Evaluation Metrics', fontsize=14, fontweight='bold') # title/leg
ax1.legend()

y_pred_proba = rf.predict_proba(x_valid)[::,1] # random forest
fpr, tpr, _ = metrics.roc_curve(y_valid,  y_pred_proba)
ax2.plot(fpr, tpr, label='Random Forest')
y_pred_proba1 = nb.predict_proba(x_valid)[::,1] # naive bayes
fpr1, tpr1, _ = metrics.roc_curve(y_valid,  y_pred_proba1)
ax2.plot(fpr1, tpr1, label='Naive Bayes')
y_pred_proba2 = knn.predict_proba(x_valid)[::,1] # knn
fpr2, tpr2, _ = metrics.roc_curve(y_valid,  y_pred_proba2)
ax2.plot(fpr2, fpr2, label='K-Nearest Neighbor')
y_pred_proba5 = log.predict_proba(x_valid)[::,1] # logistic regression
fpr5, tpr5, _ = metrics.roc_curve(y_valid,  y_pred_proba5)
ax2.plot(fpr5, fpr5, label='Logistic Regression')

ax2.set_xlabel('False Positive Rate', fontweight='bold')
ax2.set_ylabel('True Positive Rate', fontweight='bold')

ax2.set_title('ROC Curve', fontsize=14, fontweight='bold')
ax2.legend(loc=4)
plt.show()
```



Amazon SageMaker: Built-In XGBoost Model

In [115…

```
role = sagemaker.get_execution_role()
my_region = boto3.session.Session().region_name # instance's region
xgboost_container = sagemaker.image_uris.retrieve("xgboost", # XGBoost Containe
                                                  my_region,
                                                  "latest")
```

```
INFO:sagemaker.image_uris:Ignoring unnecessary instance type: None.
```

In [157…
```
train, test = np.split(df3.sample(frac=1, random_state=902),
                                  [int(0.7 * len(df3))])
print(train.shape, test.shape)
```

```
(47497, 14) (20357, 14)
```

In [158…
```
train["status"] = pd.to_numeric(train["status"])
```

In [159…
```
train = train.apply(lambda x: pd.factorize(x)[0])
```

In [160…
```
train.to_csv("train.csv", header = False, index=False)
```

In [161…
```
train.dtypes
```

Out[161]:
```
loan_amt       int64
duration       int64
pay_amt        int64
status         int64
avg_pay_amt    int64
balance        int64
op_cb          int64
op_cp          int64
op_ccp         int64
op_ccb         int64
op_bb          int64
pay_mo         int64
pay_bimo       int64
pay_wk         int64
dtype: object
```

In [162…
```
# rearranging columns w/status first b/c otherwise issues arise for the XGBoost
train = train[['status','loan_amt', 'duration', 'pay_amt', 'avg_pay_amt', 'bala
       'op_cb', 'op_cp', 'op_ccp', 'op_ccb', 'op_bb', 'pay_mo', 'pay_bimo',
       'pay_wk']]
```

In [165…
```
# bucketing training data to S3 bucket
#s3 = boto3.resource('s3')
#s3.meta.client.upload_file("train.csv", 'ads508loanapproval', 'train')
```

In [175…
```
# bucketing training data to s3
s3_client = boto3.client("s3")
BUCKET='ads508loanapproval'
KEY='train'
response = s3_client.get_object(Bucket=BUCKET, Key=KEY)

with io.StringIO() as csv_buffer:
    train.to_csv(csv_buffer, index=False, header=False)

    response = s3_client.put_object(
        Bucket=BUCKET, Key=KEY, Body=csv_buffer.getvalue()
    )
```

In [166…
```
# adding training params
s3_input_train = sagemaker.inputs.TrainingInput(s3_data=\
        's3://{}/train'.format(BUCKET), content_type='csv')
```

In [176…

```
# sagemaker sess + instance setting
sess = sagemaker.Session()
xgb = sagemaker.estimator.Estimator(xgboost_container,role,
                                    instance_count=1,
                                    instance_type='ml.m5.large',
                                    output_path='s3://{}/output'.format(BUCKET)
                                    sagemaker_session=sess)
# hyperparameter settings
xgb.set_hyperparameters(max_depth=5,eta=0.2,gamma=4,min_child_weight=6,
                        subsample=0.8,silent=0,
                        objective='binary:logistic',num_round=100)
```

In [177…

```
# model training
xgb.fit({'train': s3_input_train})
```

```
INFO:sagemaker:Creating training-job with name: xgboost-2023-04-02-18-57-51-61
3
```

```
2023-04-02 18:57:52 Starting - Starting the training job...
2023-04-02 18:58:07 Starting - Preparing the instances for training...
2023-04-02 18:58:57 Downloading - Downloading input data......
2023-04-02 18:59:37 Training - Downloading the training image...
2023-04-02 19:00:23 Uploading - Uploading generated training modelArguments: t
rain
[2023-04-02:19:00:18:INFO] Running standalone xgboost training.
[2023-04-02:19:00:18:INFO] Path /opt/ml/input/data/validation does not exist!
[2023-04-02:19:00:18:INFO] File size need to be processed in the node: 1.69mb.
Available memory size in the node: 306.05mb
[2023-04-02:19:00:18:INFO] Determined delimiter of CSV input is ','
[19:00:18] S3DistributionType set as FullyReplicated
[19:00:18] 47497x13 matrix with 617461 entries loaded from /opt/ml/input/data/
train?format=csv&label_column=0&delimiter=,
[19:00:18] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 28 extra n
odes, 0 pruned nodes, max_depth=5
[0]#011train-error:0.100764
[19:00:18] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 48 extra n
odes, 0 pruned nodes, max_depth=5
[1]#011train-error:0.091943
[19:00:18] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 28 extra n
odes, 0 pruned nodes, max_depth=5
[2]#011train-error:0.100006
[19:00:18] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 34 extra n
odes, 0 pruned nodes, max_depth=5
[3]#011train-error:0.100238
[19:00:18] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 34 extra n
odes, 0 pruned nodes, max_depth=5
[4]#011train-error:0.101185
[19:00:18] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 34 extra n
odes, 0 pruned nodes, max_depth=5
[5]#011train-error:0.099354
[19:00:18] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 34 extra n
odes, 0 pruned nodes, max_depth=5
[6]#011train-error:0.10068
[19:00:18] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 40 extra n
odes, 0 pruned nodes, max_depth=5
[7]#011train-error:0.102512
[19:00:18] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 50 extra n
odes, 2 pruned nodes, max_depth=5
[8]#011train-error:0.09468
[19:00:18] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 36 extra n
odes, 0 pruned nodes, max_depth=5
[9]#011train-error:0.092911
[19:00:18] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 32 extra n
odes, 2 pruned nodes, max_depth=5
[10]#011train-error:0.089985
[19:00:18] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 46 extra n
odes, 6 pruned nodes, max_depth=5
[11]#011train-error:0.090258
[19:00:18] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 30 extra n
odes, 0 pruned nodes, max_depth=5
[12]#011train-error:0.094785
[19:00:18] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 16 extra n
odes, 0 pruned nodes, max_depth=5
[13]#011train-error:0.094785
[19:00:18] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 30 extra n
odes, 2 pruned nodes, max_depth=5
[14]#011train-error:0.093627
[19:00:18] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 36 extra n
```

```
odes, 0 pruned nodes, max_depth=5
[15]#011train-error:0.0847
[19:00:18] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 34 extra n
odes, 0 pruned nodes, max_depth=5
[16]#011train-error:0.0847
[19:00:18] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 14 extra n
odes, 2 pruned nodes, max_depth=5
[17]#011train-error:0.08569
[19:00:18] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 34 extra n
odes, 0 pruned nodes, max_depth=5
[18]#011train-error:0.086048
[19:00:18] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 14 extra n
odes, 0 pruned nodes, max_depth=5
[19]#011train-error:0.084132
[19:00:18] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 54 extra n
odes, 0 pruned nodes, max_depth=5
[20]#011train-error:0.076384
[19:00:18] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 16 extra n
odes, 0 pruned nodes, max_depth=5
[21]#011train-error:0.076363
[19:00:18] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 46 extra n
odes, 0 pruned nodes, max_depth=5
[22]#011train-error:0.073478
[19:00:18] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 18 extra n
odes, 0 pruned nodes, max_depth=5
[23]#011train-error:0.07331
[19:00:18] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 14 extra n
odes, 0 pruned nodes, max_depth=5
[24]#011train-error:0.07251
[19:00:19] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 24 extra n
odes, 0 pruned nodes, max_depth=5
[25]#011train-error:0.066531
[19:00:19] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 38 extra n
odes, 0 pruned nodes, max_depth=5
[26]#011train-error:0.064846
[19:00:19] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 20 extra n
odes, 0 pruned nodes, max_depth=5
[27]#011train-error:0.063815
[19:00:19] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 50 extra n
odes, 2 pruned nodes, max_depth=5
[28]#011train-error:0.06032
[19:00:19] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 44 extra n
odes, 4 pruned nodes, max_depth=5
[29]#011train-error:0.059014
[19:00:19] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 20 extra n
odes, 0 pruned nodes, max_depth=5
[30]#011train-error:0.058319
[19:00:19] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 16 extra n
odes, 0 pruned nodes, max_depth=5
[31]#011train-error:0.058319
[19:00:19] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 26 extra n
odes, 2 pruned nodes, max_depth=5
[32]#011train-error:0.055751
[19:00:19] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 34 extra n
odes, 4 pruned nodes, max_depth=5
[33]#011train-error:0.054361
[19:00:19] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 16 extra n
odes, 0 pruned nodes, max_depth=5
[34]#011train-error:0.054193
[19:00:19] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 26 extra n
```

```
odes, 0 pruned nodes, max_depth=5
[35]#011train-error:0.050003
[19:00:19] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 14 extra n
odes, 0 pruned nodes, max_depth=5
[36]#011train-error:0.048635
[19:00:19] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 10 extra n
odes, 2 pruned nodes, max_depth=5
[37]#011train-error:0.048803
[19:00:19] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 42 extra n
odes, 2 pruned nodes, max_depth=5
[38]#011train-error:0.042318
[19:00:19] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 32 extra n
odes, 4 pruned nodes, max_depth=5
[39]#011train-error:0.041582
[19:00:19] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 34 extra n
odes, 0 pruned nodes, max_depth=5
[40]#011train-error:0.040339
[19:00:19] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 12 extra n
odes, 0 pruned nodes, max_depth=5
[41]#011train-error:0.040339
[19:00:19] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 22 extra n
odes, 2 pruned nodes, max_depth=5
[42]#011train-error:0.040339
[19:00:19] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 46 extra n
odes, 2 pruned nodes, max_depth=5
[43]#011train-error:0.039139
[19:00:19] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 18 extra n
odes, 0 pruned nodes, max_depth=5
[44]#011train-error:0.039139
[19:00:19] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 10 extra n
odes, 2 pruned nodes, max_depth=5
[45]#011train-error:0.039139
[19:00:19] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 20 extra n
odes, 0 pruned nodes, max_depth=5
[46]#011train-error:0.038571
[19:00:19] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 22 extra n
odes, 4 pruned nodes, max_depth=5
[47]#011train-error:0.038402
[19:00:19] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 32 extra n
odes, 0 pruned nodes, max_depth=5
[48]#011train-error:0.037581
[19:00:19] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 28 extra n
odes, 0 pruned nodes, max_depth=5
[49]#011train-error:0.037139
[19:00:19] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 30 extra n
odes, 0 pruned nodes, max_depth=5
[50]#011train-error:0.034213
[19:00:20] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 24 extra n
odes, 0 pruned nodes, max_depth=5
[51]#011train-error:0.033434
[19:00:20] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 22 extra n
odes, 4 pruned nodes, max_depth=5
[52]#011train-error:0.028739
[19:00:20] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 10 extra n
odes, 2 pruned nodes, max_depth=5
[53]#011train-error:0.028739
[19:00:20] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 36 extra n
odes, 2 pruned nodes, max_depth=5
[54]#011train-error:0.027939
[19:00:20] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 34 extra n
```

```
odes, 2 pruned nodes, max_depth=5
[55]#011train-error:0.02737
[19:00:20] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 20 extra n
odes, 0 pruned nodes, max_depth=5
[56]#011train-error:0.026528
[19:00:20] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 20 extra n
odes, 0 pruned nodes, max_depth=5
[57]#011train-error:0.026528
[19:00:20] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 26 extra n
odes, 2 pruned nodes, max_depth=5
[58]#011train-error:0.024907
[19:00:20] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 36 extra n
odes, 2 pruned nodes, max_depth=5
[59]#011train-error:0.023265
[19:00:20] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 30 extra n
odes, 2 pruned nodes, max_depth=5
[60]#011train-error:0.022907
[19:00:20] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 40 extra n
odes, 4 pruned nodes, max_depth=5
[61]#011train-error:0.022212
[19:00:20] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 38 extra n
odes, 4 pruned nodes, max_depth=5
[62]#011train-error:0.02158
[19:00:20] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 20 extra n
odes, 0 pruned nodes, max_depth=5
[63]#011train-error:0.020907
[19:00:20] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 26 extra n
odes, 4 pruned nodes, max_depth=5
[64]#011train-error:0.019264
[19:00:20] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 26 extra n
odes, 12 pruned nodes, max_depth=5
[65]#011train-error:0.018633
[19:00:20] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 22 extra n
odes, 2 pruned nodes, max_depth=5
[66]#011train-error:0.015411
[19:00:20] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 14 extra n
odes, 0 pruned nodes, max_depth=5
[67]#011train-error:0.015411
[19:00:20] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 18 extra n
odes, 0 pruned nodes, max_depth=5
[68]#011train-error:0.014422
[19:00:20] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 18 extra n
odes, 2 pruned nodes, max_depth=5
[69]#011train-error:0.01438
[19:00:20] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 26 extra n
odes, 2 pruned nodes, max_depth=5
[70]#011train-error:0.014148
[19:00:20] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 26 extra n
odes, 8 pruned nodes, max_depth=5
[71]#011train-error:0.013748
[19:00:20] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 20 extra n
odes, 0 pruned nodes, max_depth=5
[72]#011train-error:0.01398
[19:00:20] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 18 extra n
odes, 0 pruned nodes, max_depth=5
[73]#011train-error:0.013938
[19:00:21] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 34 extra n
odes, 2 pruned nodes, max_depth=5
[74]#011train-error:0.013664
[19:00:21] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 20 extra n
```

```
odes, 0 pruned nodes, max_depth=5
[75]#011train-error:0.013622
[19:00:21] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 22 extra n
odes, 2 pruned nodes, max_depth=5
[76]#011train-error:0.013011
[19:00:21] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 24 extra n
odes, 6 pruned nodes, max_depth=5
[77]#011train-error:0.012674
[19:00:21] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 28 extra n
odes, 4 pruned nodes, max_depth=5
[78]#011train-error:0.01299
[19:00:21] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 26 extra n
odes, 4 pruned nodes, max_depth=5
[79]#011train-error:0.01299
[19:00:21] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 22 extra n
odes, 0 pruned nodes, max_depth=5
[80]#011train-error:0.012801
[19:00:21] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 20 extra n
odes, 0 pruned nodes, max_depth=5
[81]#011train-error:0.01238
[19:00:21] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 22 extra n
odes, 0 pruned nodes, max_depth=5
[82]#011train-error:0.01238
[19:00:21] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 22 extra n
odes, 6 pruned nodes, max_depth=5
[83]#011train-error:0.01238
[19:00:21] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 18 extra n
odes, 0 pruned nodes, max_depth=5
[84]#011train-error:0.012211
[19:00:21] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 24 extra n
odes, 6 pruned nodes, max_depth=5
[85]#011train-error:0.012211
[19:00:21] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 22 extra n
odes, 2 pruned nodes, max_depth=5
[86]#011train-error:0.012253
[19:00:21] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 40 extra n
odes, 4 pruned nodes, max_depth=5
[87]#011train-error:0.01219
[19:00:21] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 28 extra n
odes, 6 pruned nodes, max_depth=5
[88]#011train-error:0.011853
[19:00:21] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 20 extra n
odes, 4 pruned nodes, max_depth=5
[89]#011train-error:0.011832
[19:00:21] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 18 extra n
odes, 2 pruned nodes, max_depth=5
[90]#011train-error:0.011832
[19:00:21] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 20 extra n
odes, 2 pruned nodes, max_depth=5
[91]#011train-error:0.011832
[19:00:21] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 28 extra n
odes, 2 pruned nodes, max_depth=5
[92]#011train-error:0.011095
[19:00:21] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 24 extra n
odes, 6 pruned nodes, max_depth=5
[93]#011train-error:0.011032
[19:00:21] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 28 extra n
odes, 4 pruned nodes, max_depth=5
[94]#011train-error:0.009811
[19:00:21] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 22 extra n
```

```
odes, 2 pruned nodes, max_depth=5
[95]#011train-error:0.009811
[19:00:21] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 36 extra n
odes, 2 pruned nodes, max_depth=5
[96]#011train-error:0.00918
[19:00:21] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 16 extra n
odes, 0 pruned nodes, max_depth=5
[97]#011train-error:0.00918
[19:00:21] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 14 extra n
odes, 2 pruned nodes, max_depth=4
[98]#011train-error:0.00918
[19:00:21] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 26 extra n
odes, 0 pruned nodes, max_depth=5
[99]#011train-error:0.00918

2023-04-02 19:00:34 Completed - Training job completed
Training seconds: 97
Billable seconds: 97
```

In [179…

```python
# setting the predictions
xgb_predictor = xgb.deploy(initial_instance_count=1,
                           instance_type='ml.m5.large')
```

```
INFO:sagemaker:Creating model with name: xgboost-2023-04-02-19-03-35-209
INFO:sagemaker:Creating endpoint-config with name xgboost-2023-04-02-19-03-35-
209
INFO:sagemaker:Creating endpoint with name xgboost-2023-04-02-19-03-35-209
----!
```

In [180…

```python
# running the set predictions
test_array = test.drop(['status'], axis=1).values

# setting serializer type
xgb_predictor.serializer = CSVSerializer()
predi = xgb_predictor.predict(test_array).decode('utf-8')

# setting data as array
predi_array = np.fromstring(predi[1:], sep=',')
print(predi_array.shape)
```

```
(20357,)
```

In [192…

```python
test.shape
```

Out[192]:

```
(20357, 14)
```

In [224…

```python
# XGBoost Model Eval
cm = pd.crosstab(index=test["status"],
                 columns=np.round(predi_array),
                 rownames=['Observed'],
                 colnames=['Predicted'])
tn = cm.iloc[0,0]
fn = cm.iloc[1,0]
fp = cm.iloc[0,0]
tp = cm.iloc[1,0]


print("Precision", tp/(tp+fp)*100)
print("Sensitivity", tp/(tp+fn)*100)
```

```python
print("Recall", tn/(tn+fp)*100)
print("Accuracy", (tp+tn)/(tp+fn+tn+fp)*100)
```

```
Precision 88.24974210345336
Sensitivity 50.0
Recall 50.0
Accuracy 50.0
```

In [225…

```python
# deleteting endpoint
xgb_predictor.delete_endpoint(delete_endpoint_config=True)
```

```
INFO:sagemaker:Deleting endpoint configuration with name: xgboost-2023-04-02-1
9-03-35-209
INFO:sagemaker:Deleting endpoint with name: xgboost-2023-04-02-19-03-35-209
```

Resources

Chadha, A. S. (2022, January 6). Handling Imbalanced Datasets With Oversampling Techniques. It's Pros & Cons. Medium. https://medium.com/analytics-vidhya/handling-imbalanced-datasets-with-oversampling-techniques-its-pros-cons-ba9f36ac5b71

Financial Dataset. (n.d.). CTU Prague Relational Learning Repository. https://relational.fit.cvut.cz/dataset/Financial

How XGBoost Works - Amazon SageMaker. (n.d.). https://docs.aws.amazon.com/sagemaker/latest/dg/xgboost-HowItWorks.html

Logistic Regression Model Explained - AWS. (n.d.). Amazon Web Services, Inc. https://aws.amazon.com/what-is/logistic-regression/#:~:text=Logistic%20regression%20is%20a%20data,outcomes%2C%20like%20yes

Radha, S. E. (2023, March 24). Understand Random Forest Algorithms With Examples (Updated 2023). Analytics Vidhya.https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/

Vatsal, V. (2022, May 20). K Nearest Neighbours Explained - Towards Data Science. Medium. https://towardsdatascience.com/k-nearest-neighbours-explained-7c49853633b6

Yıldırım, S. (2021, December 13). Naive Bayes Classifier — Explained - Towards Data Science. Medium. https://towardsdatascience.com/naive-bayes-classifier-explained-50f9723571ed