2023 Business Analytics Competition Report

Alejandro Mata & Drew Bruggman

This report provides detailed explanations regarding our approach to feature selection, the model utilized, and the scoring methodology employed in our submission.
Note: While efforts were made to fix random seeds, the figures presented in this report may exhibit minor discrepancies (<0.2%) from those observed in the respective notebooks.

Dependencies:

	URL	**Version**
Python	*https://www.python.org*	*3.11.5*
Pandas	*https://pandas.pydata.org*	*2.1.3*
Numpy	*https://numpy.org/devdocs/index.html*	*1.25.2*
Matplotlib	*https://matplotlib.org*	*3.7.3*
Seaborn	*https://seaborn.pydata.org*	*0.12.2*
Scikit-Learn	*https://scikit-learn.org/stable/*	*1.3.2*
Ydata-Profiling	*https://docs.profiling.ydata.ai/latest/*	*4.6.3*
XGBoost	*https://xgboost.readthedocs.io/en/stable/index.html*	*2.0.2*
TensorFlow	*https://www.tensorflow.org*	*2.15.0*

0. Results summary

Our research showed that tree-based models were able to *learn* the complex non-linear relationships between our features and target very accurately. Form among them, *XGBoost* – the industry standard for tree-based models - offered outstanding results.

Best Model	Boosting (from XGBoost)
Best Obtained Profit	**$7,209,306,030.15**
Profit captured	92.10%
Cutoff for future applicants	0.525

1. Data Exploration and Preprocessing

1.1. Data Exploration and Cleaning

Related Notebook: "preprocessing_code.ipynb"

The first stage of our problem was exploring the distribution of features and their relationship with out target value, 'MIS_Status'. To do so, we crafted basic *Tableau* graphs (see "*tableau graphs.twb")* and examined the report generated by *ydata-profiling* (see "*raw_report.html"*).

After checking the stats, we selected the features to keep and discarded the rest. Additionally, we engineered the Interest rate feature by cross-referencing the date of dispersal with an external reference to a table of Federal interest rates by month. The decisions we made on a per-feature basis are listed below:

1. *LoanNr_ChkDgt:* Primary key of the database, offers no predictive value and gets dropped.
2. *Name:* Name of the borrower, it gets dropped. Further analysis could future engineer it into the number of loans per business.
3. *City:* As seen in our *Tableau* graphs and report, a correlation exists between the status of a loan and the location of the business. Therefore, it will be kept for feature engineering.
4. *State:* Just like *City*, we chose to keep it for future feature engineering.
5. *ZIP:* It offers more granularity on the location of the loan than needed. We considered *City* and *State* to be adequate, so therefore we dropped it.
6. *Bank:* Our analysis showed a correlation, so gets kept for feature engineering.
7. *Bank State:* Doesn't add any relevant information, so it was dropped.
8. *NAICS:* We examined its connection to our target and opted to retain only the initial two digits, representing a broader industry scope. To delve deeper, refer to Census, 2002 for additional field insights. Observing its histogram, these broader industries appear as grouped points, depicted by bars.

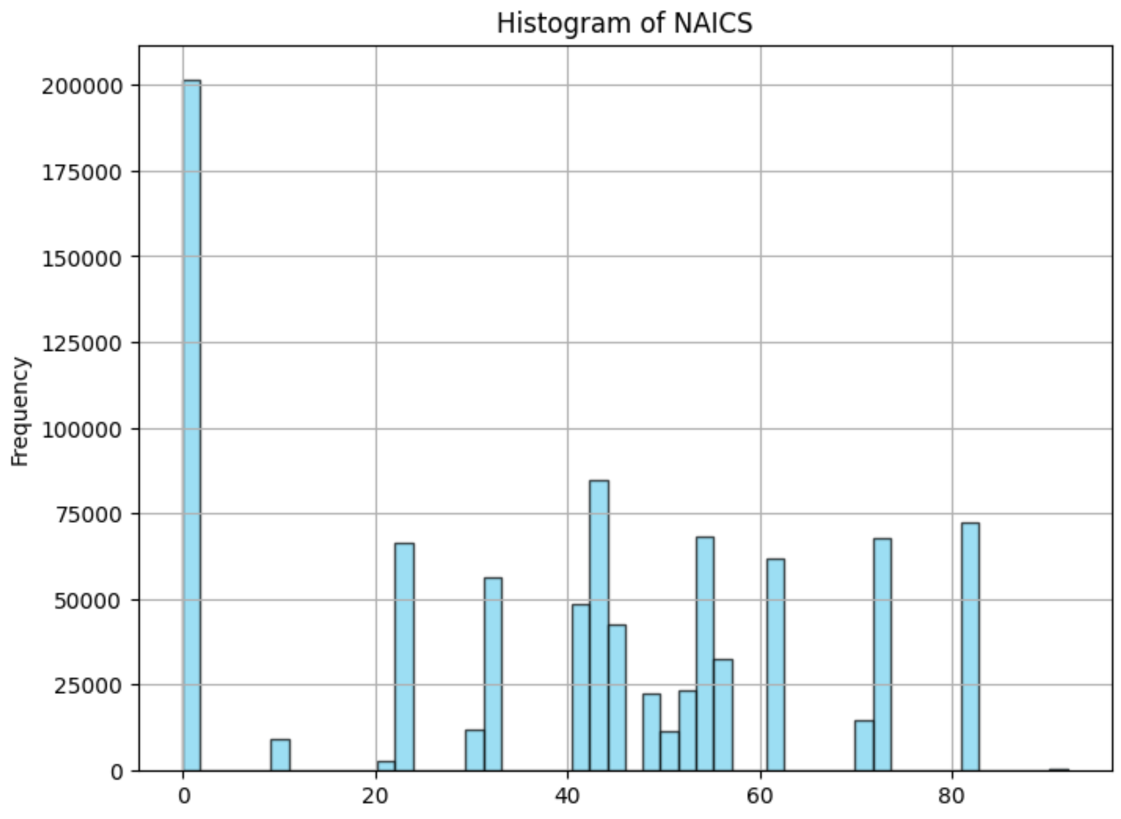


Figure 1. NAICS histogram

9. *ApprovalDate:* The primary function of this feature is as a reference to associate a loan with a federal interest rate.
10. *ApprovalFY:* Being included inside of *ApprovalDate,* we consider it to not carry additional value and gets dropped.
11. *Term:* We noticed a strong connection between a loan's term length and its likelihood to default. As a result, it's marked as a numerical feature in our selection.
12. *NoEmp:* We consider the number of employees to be a valuable metric for the health of a business. Gets selected.
13. *CreateJob:* Similarly to *NoEmp*, seems to carry valuable information on the status of a business.
14. *RetainedJob:* Like the two previous ones, gets selected.
15. *FranchiseCode:* We identified some of the most common franchises. There is a correlation, so it gets kept.
16. *UrbanRural:* We understand the value 0 (undefined) as those business which are not urban nor rural (i.e., Suburban). As there is a correlation with out target value, we keep it as a feature.
17. *RevLineCr:* Gets kept as a binary feature, as it shows correlation with our target.
18. *LowDoc:* Similar to *RevLineCr.*
19. *ChgOffDate:* This feature contains information about our target (a loan has only a charge off date if it defaults). Therefore, gets dropped.
20. *DisbursementDate:* As we are already keeping *ApprovalDate*, we don't consider this feature to add value.
21. *DisbursementGross:* There is a correlation between the size of the loan and its default probability, so it is kept.
22. *BalanceGross:* It is 0 for most of our loans (only 14 loans have a non-zero value) so it gets dropped.
23. *MIS_Status:* Our target, representing the loan status. Gets kept as *Default* (being 1 if the loan defaulted and 0 otherwise).
24. *ChgOffPrinGr:* Similarly to *ChgOffDate*, this feature supposes target leaking into our data. Gets dropped.
25. *GrAppv:* Another representation of the size of the loan, we decided to keep it.
26. *SBA_Appv:* Not only it can be yet another representation of the loan's size, but it can also be engineered into the ratio of insurance of a loan by the SBA.

After identifying the features that we considered to carry predictive value, we cleaned the data by dropping *NaNs*, *nulls*, missing values, and other inconsistencies.
The cleaned data included 870,514 observations, as 28,651 had been dropped during the cleaning.

Then, we added our engineered features, which we considered to add predictive value to our modelling.

1. *SBARatio:* The SBA insurance ratio, defined as:

$$SBARatio = \frac{SBA\_Appv}{GrAppv}$$

We see the ratio as a potent predictor, reflecting the SBA's confidence in a particular loan. When we plotted a histogram of the insurance ratio, it revealed typical ratios, indicating their use of a model to assess a loan's risk.

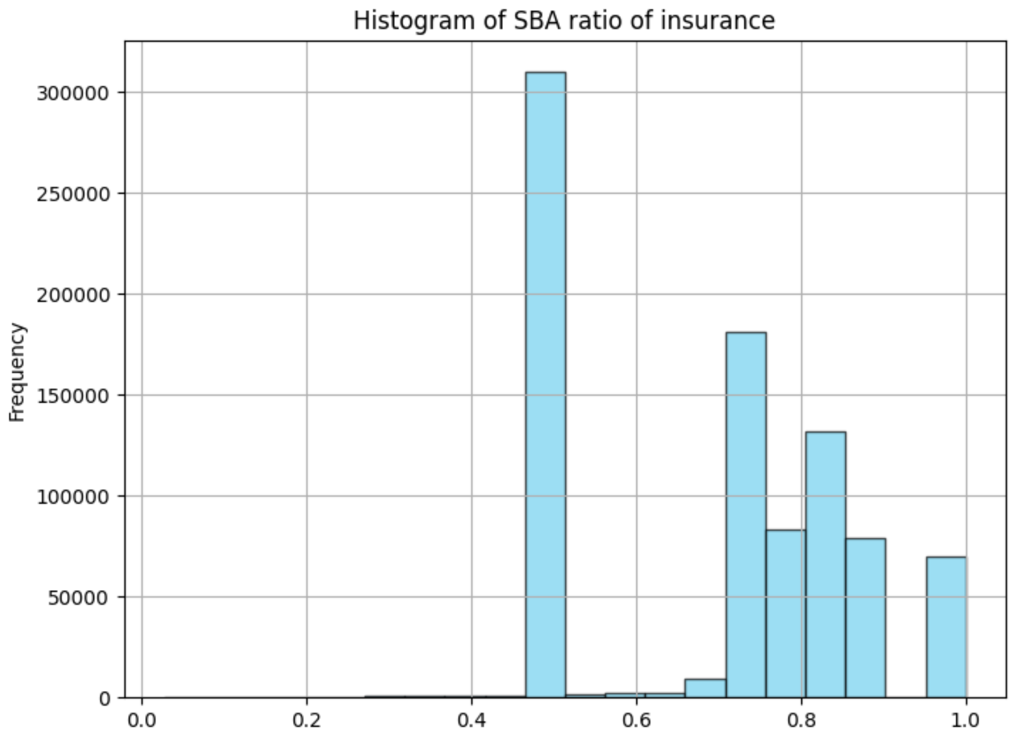


Figure 2. SBA Ratio histogram

2. *InterestRate:* It represents the federal funds rate at the time a loan was approved. This helps estimate the loan's interest rate, as it's closely tied to the federal rate. To derive this, we fetched the historic Fed funds interest rates as a CSV file and employed *pandas'* vectorized operations to integrate each loan's interest rate.
3. *NAICS_i:* The two first digits of NAICS, encoding the broader industry instead of the specific one.
4. *CityLoans, BankLoans, StateLoans:* As proxies, the sizes of a city, bank, and state. Each one indicates the count of loans associated with them. See *Why Does Frequency Encoding Work?*
5. *CityDefault, BankDefault, StateDefault, FranchiseDefault:* They contain the default ratio within each of these categories, utilizing target encoding. This strategy is a well-established and commonly employed technique within the industry. See Trevisan (2022).

1.2. Correlation analysis.

Armed with this array of features, we generated a correlation matrix using *pandas* and *seaborn*, examining their relationships with our target variable. See Chip (2023) for details.

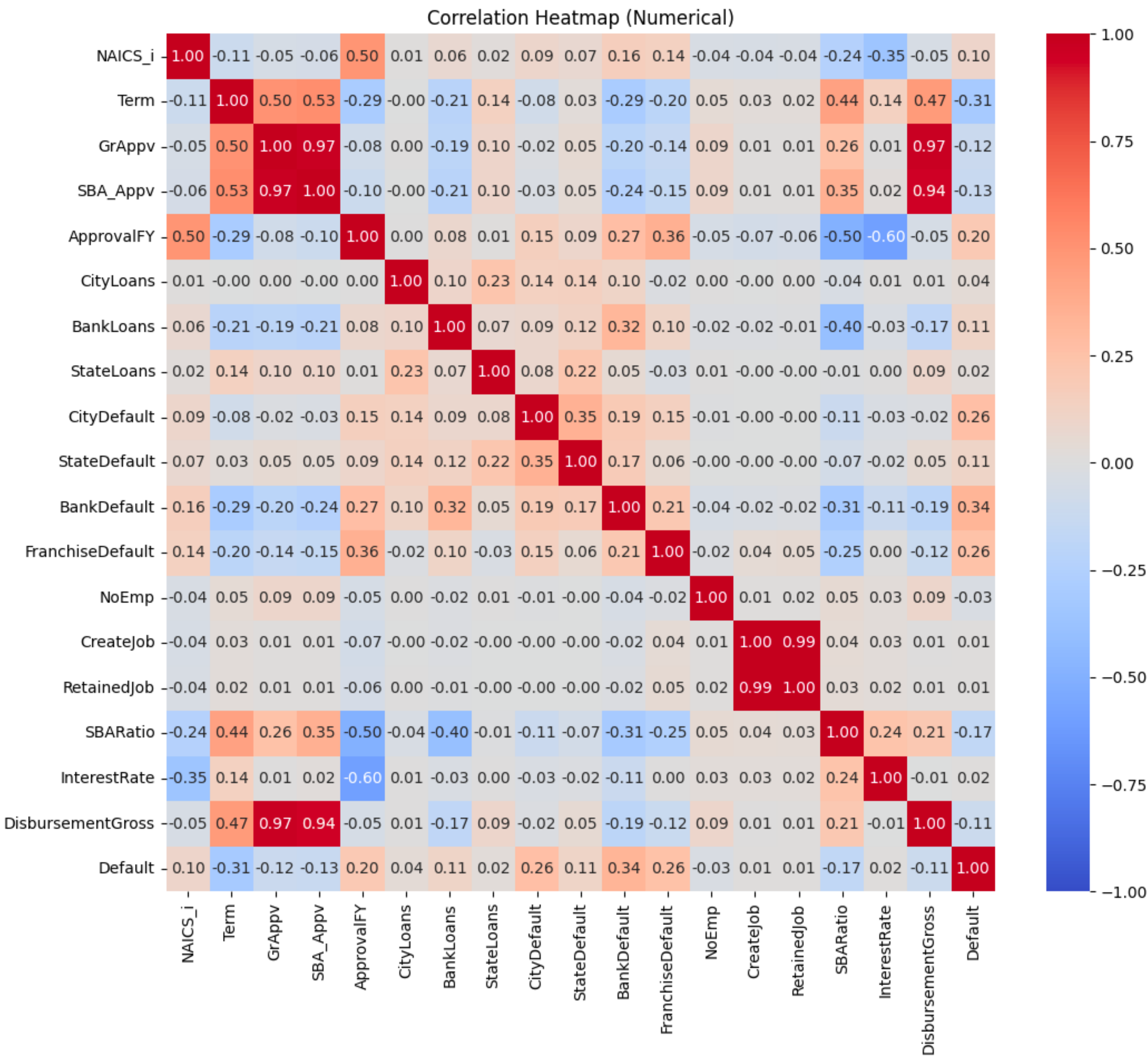


Figure 3. Correlation heatmap

This gave us the first insights on how the features were distributed, and which ones were more promising as predictors. Then, we plotted density graphs for each of the numerical features, obtaining interesting information on how our target was distributed.
For more details on how to interpret a density plot, see Mulani (2022).

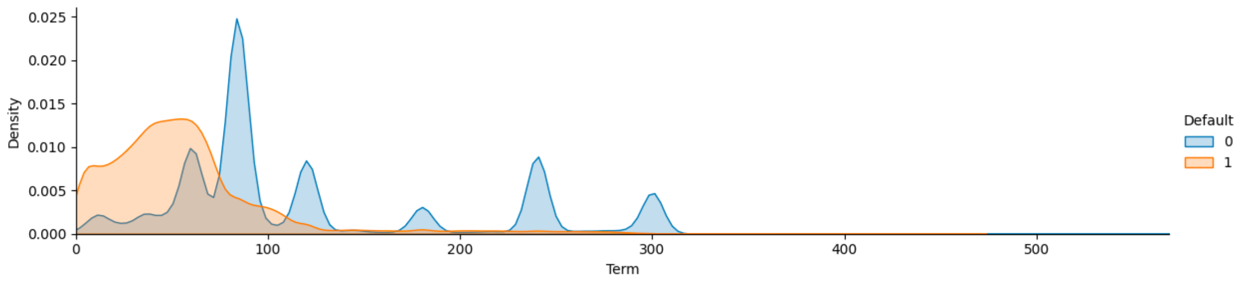


Figure 4. Term density plot

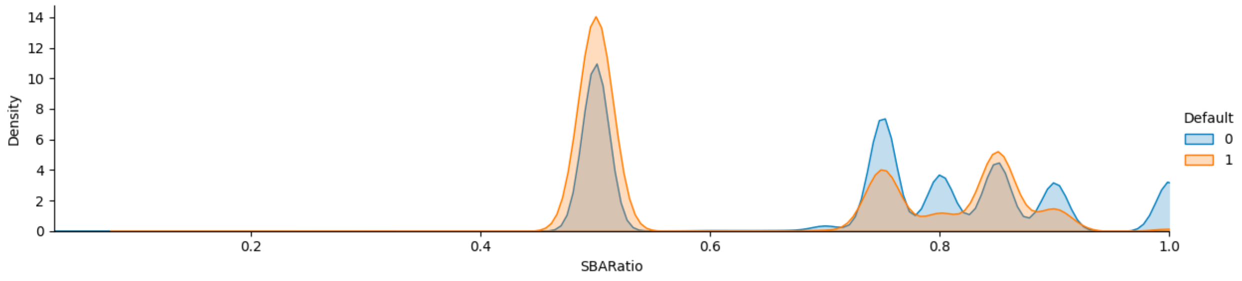


Figure 5. SBA Ratio density plot

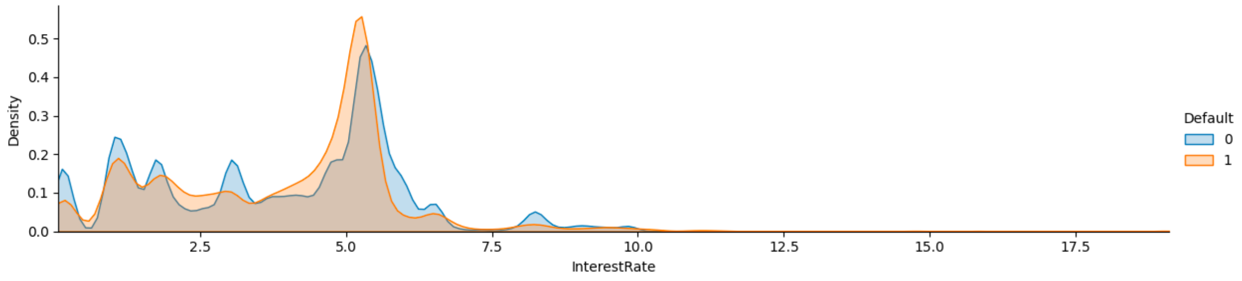


Figure 6. Fed Funds density plot

The most interesting plots correspond to a loan's term, SBA Ratio, and Federal interest rate. These graphs highlight that loans with extended terms, lower interest rates, and higher SBA insurance are less prone to default, as depicted above.

1.3. Importance analysis.

Related Notebook: "importance_analysis.ipynb"

Our dataset is mainly tabular and showcases a diverse range of features, including categorical, binary, discrete, and continuous ones, highlighting its significant heterogeneity. Research suggesting the superiority of tree-based models over neural networks in such scenarios (Grinsztajn, 2022) led us to favor these models. Thus, our subsequent step focused on conducting an importance analysis within a single decision tree framework. This served as a baseline for performance and provided insights into how our predictors influence the target variable.

Furthermore, it would reveal whether the *frequency encoded* and *target encoded* engineered features would benefit the model.
We decided to run five iterations of the tree:

I. **Baseline**: single tree, employing features *Term, InterestRate, UrbanRural* (one-hot encoded), *SBARatio, DisbursementGross, GrAppv, RetainedJob, SBA_Appv, RevLineCr, NoEmp, isNewBusiness, LowDoc, CreateJob, isFranchise.*
II. **Feature elimination**: single tree, employing the most important features. See *Decision Tree Vs Logistic Regression Feature Importances*.
III. **Count encoding:** *CityLoans, BankLoans, StateLoans* and their combinations get added.
IV. **Target encoding:** *CityDefault, BankDefault, StateDefault, FranchiseDefault, NAICSDefault* and their combinations get added.
V. **One-hot encoding:** NAICS_i gets one-hot encoded to all the possible broader industries,

1.3.1. Baseline performance.

It's a solid practice to establish a baseline performance before model training. This allows for comparison with future results, determining whether increased complexity leads to performance improvement (Ameisen, 2018). Given our data's 17% default ratio, a dummy model consistently predicting no defaults would achieve an 83% accuracy! Hence, accuracy is dismissed as a metric in favor of more informative ones such as precision, recall, ROC AUC, and F1 (Brownlee, 2021).

1.3.2. Feature importance and elimination.

In constructing our single decision tree, we opted for manual tuning of the hyperparameters *max_depth* and *min_samples_split*, considering their effectiveness in combating overfitting (refer to the *scikit-learn* documentation). After selecting a range of hyperparameters for experimentation, our model demonstrated optimal performance within a maximum depth of 12-16 and a minimum sample split of 100-200.
For a visualization on how this parameter affect the bias/variance tradeoff, we graphed the resulting testing and *cross-validation* F1 scores.

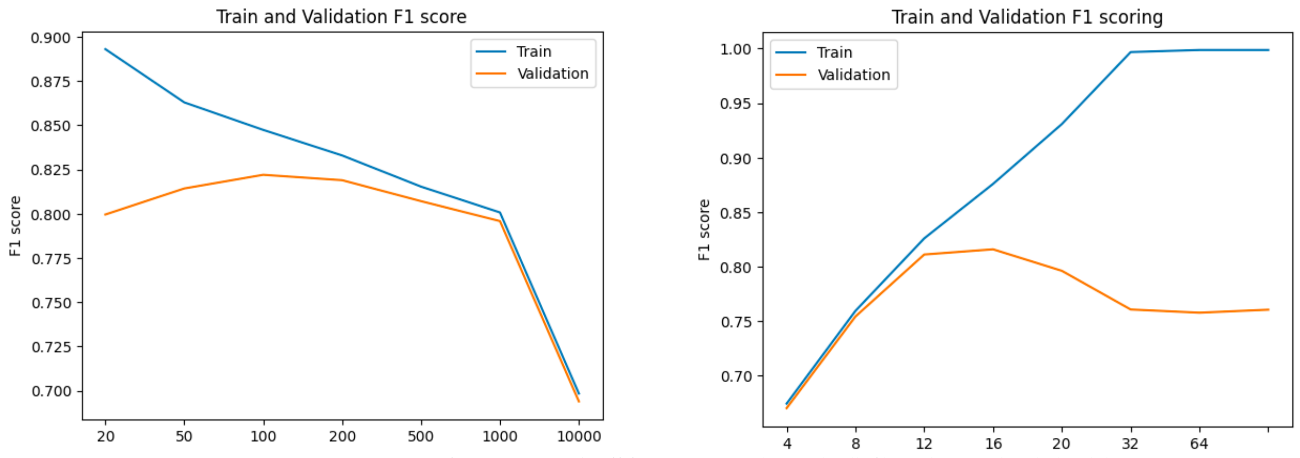


Figure 7. Bias/Variance tradeoff for min_samples_split (left) and max_depth (right)

It is a good representation on how very high(low) min_samples_split or very low(high) max_depth under(over)fit. From that tree, we obtained the importance ranking of our predictors:

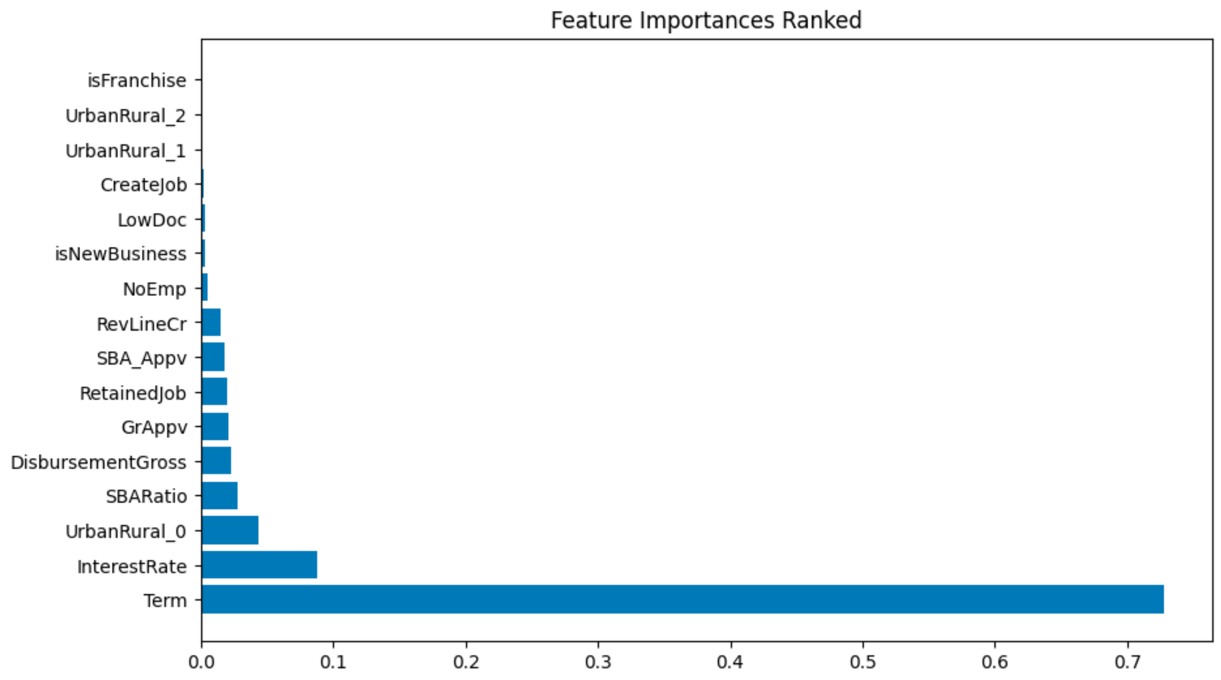


Figure 8. Feature importance on a single tree.

With the importance graph in hand, we selected the top 9 most significant features to train the tree and then compared it to our previous iteration. The lost in all metrics was minimal (see table at the end of this section), so we decided to limit ourselves to those features.

1.3.3. Alternative Encodings

In successive iterations, we performed population encoding, target encoding and one-hot encoding in the features mentioned before. The scores for each iteration, when predicting both a test subset and the whole *dataframe*, were as follows:

	Recall	F1
Baseline Test	0.81246	0.82343
Baseline Total	0.825399	0.837152
RFE Test	0.812353	0.823143
RFE Total	0.82507	0.836228
Count Encoding Test	**0.818367**	**0.83427**
Count Encoding Total	**0.828996**	**0.84713**
Target Encoding Test	0.817541	0.837133
Target Encoding Total	0.830323	0.85033
One-Hot Encoding Test	0.815511	0.833899
One-Hot Encoding Total	0.829117	0.845964

Hence, we opted for the reduced feature set, incorporating count encoded features (highlighted in bold in the table above), as the increased complexity from more advanced encodings did not improve the model's performance.

2. Model Training

Related Notebook: "trees.ipynb"

As stated before, and following the conclusion from Grinsztajn (2022), we decided to focus first on tree-based models, in crescent order of complexity. For the dataset, we took the clean *dataframe* generated after preprocessing, including the engineered columns of *SBARatio* and *InterestRate.*
Once imported, we one-hot encoded *UrbanRural* (resulting in three columns that could be interpreted as *is suburban, is urban, is rural*) and added the population encoded *CityLoans, StateLoans* and *BankLoans.*
We partitioned the data into arbitrary segments, allocating 85% for training and 15% for testing. Selecting the 9 most important features, we proceeded to fit our models.

2.1. Single Tree

Our initial focus centered on optimizing a single tree, recognizing that these findings would prove valuable in constructing bagging and random forest models (as outlined in the *scikit-learn* documentation).
We commenced hyperparameter tuning, starting with the weight of our features. Considering the significant impact of our target's imbalance on profitability, we prioritized optimizing weight for profit, aiming to identify the weight that yielded the highest profit. Then, we did a *bias/variance* check by doing predictions on the total dataset and an unseen test partition.
Finally, knowing that the model was not overfitting, in a process we replicated for all subsequent models, we iterated through various probabilities to determine the best cut-off point. Our analysis revealed 0.575 as the most profitable cut-off for a single tree. The score was as follows:

	Recall	Precision	ROC	F1	**Profit, $**	**Cutoff**
Single Tree	0.907269	0.719411	0.91702	0.802492	**6,810,022,816.65**	**0.575**

2.2.Bagging
2.3.

For the bagging estimator (refer to the *scikit-learn* documentation), we used the Single Tree that we had already optimized. Then we optimized for profitability the number of estimators, and the maximum number of features and samples to train on. After doing the *bias/variance* check , we obtained the best cut-off probability. The scores were the following:

	Recall	Precision	ROC	F1	**Profit, $**	**Cutoff**
Bagging	0.939059	0.725288	0.932726	0.818445	**7,007,832,782.40**	**0.45**

A noticeable increase with respect to a single tree can be seen.

2.3. Random Forest

Our final model within the ***scikit-learn*** tree-based models was a *Random Forest Classifier.* As for this model we needed a broader search of hyperparameters, we decided to use *HalvingGridSearchCV* (refer to *scikit-learn* documentation).
After various computing power – and time – intensive iterations, we converged on a set of *hyperparameters* that we hardcoded in the model, so it doesn't need to run again. After doing the *bias/variance* test, we proceeded with the cut-off selection, and obtained the following results:

	Recall	Precision	ROC	F1	**Profit, $**	**Cutoff**
Random Forest	0.924045	0.793664	0.937165	0.853907	**7,097,291,355.55**	**0.55**

Being a more advanced model, it brought another noticeable improvement in our scorings.

2.4. Boosting

Finally, we delved into what has emerged as the industry standard for tree-based models (Torres, 2023): *XGBoost*. Given its complexity, we opted for manual hyperparameter tuning, mirroring our approach with a single tree: systematically iterating through various values for individual hyperparameters and evaluating train and *Cross-Validation F1* scores to determine the optimal settings.
We iterated over *eta* (also known as the learning rate)*, gamma,* number of estimators, maximum depth for each estimator, subsampling ratio, subset of features per tree, and *lambda* and *alpha* regularizations. Upon finding the optimal hyperparameters, we assessed bias/variance and confirmed that the model avoided overfitting. Following this, we focused on optimizing the scaling for the positive target to maximize profit, aiming to address the dataset's imbalance.
Finally, we optimized the cutoff for profit, arriving to excellent results. The whole scoring matrix for all four tree-based models, along with a theoretical perfect estimator is as follows:

Model	Recall	Precision	ROC	F1	Profit, $	Cutoff
Single Tree	0.907269	0.719411	0.917020	0.802492	6,810,022,816.65	0.575
Bagging	0.939059	0.725288	0.932726	0.818445	7,007,832,782.40	0.45
Random Forest	0.924045	0.793664	0.937165	0.853907	7,097,291,355.55	0.55
XGBoost	**0.928849**	**0.823537**	**0.943831**	**0.873029**	**7,193,562,322.05**	**0.5**
Perfect Estimator	1.000000	1.000000	1.000000	1.000000	7,835,152,496.90	-

We then exported the *XGBoost* model to *JSON* as the best tree-based model, along with the data in the exact shape that it was used for its training. The confusion matrix and relevant metrics are shown at the end of this document.

2.5. kNN

The *k-Nearest-Neighbors* algorithm that drives the model involves a simple 2-step-process:

1. Get the distance from any one data point to all other datapoints as a list, ranked shortest to longest.
2. Sample the top K neighbors from that generated list and use that sample to decide what cluster the data point most-resembles.

These are the K neighbors to the datapoint with the shortest vector-distance from it, hence: K-Nearest Neighbors. The model suffers from a number of potential pitfalls on a conceptual level when applied to our situation:

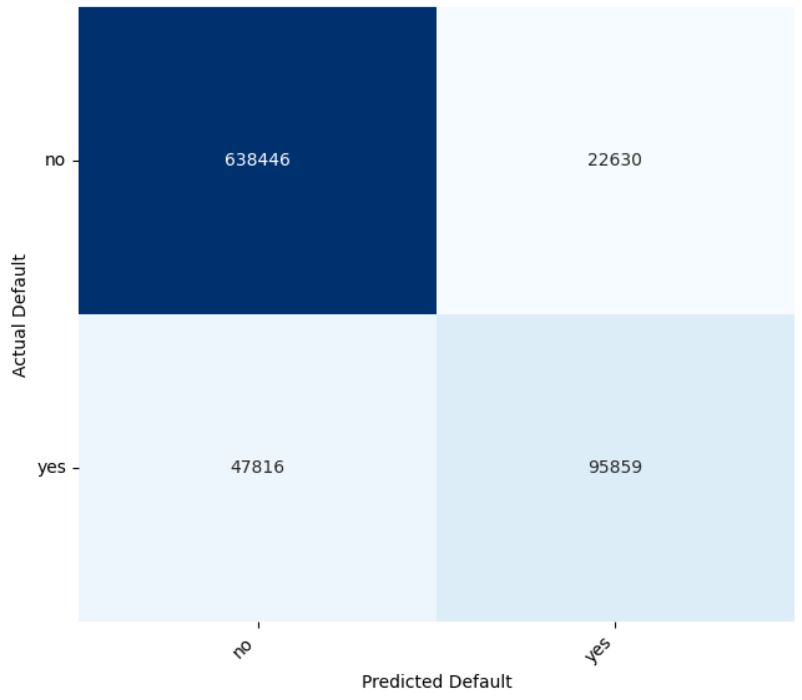
- Its calculations are distance-based, and as such suffers from feature sets where different features have wildly different potential value-ranges. In this instance, the larger-valued features have much more pull on the outcome than categorical features which at-most have a distance value of 1. This is why scaling the relevant data down is imperative.

- Even after scaling, however, there is still an issue with evaluating the weight any one categorical feature should have. They are non-continuous and as such are very all-or-nothing when evaluating their contributions to an output; finding nuance and identifying the border between belonging to one cluster vs another can be fickle, and borders can become artificially (and inaccurately) jagged as a result.

- Finally, KNN is also very sensitive to skewed data. The algorithm relies on an underlying assumption that the numeric data it is working with is evenly centered, and movement from that center is of equivalent significance in either direction. There is no directionality to KNN's measurements, just raw distance, and so if there is some implied significance to the direction of a values deviation from center, it is lost in the KNN model.

The ideal underlying data that drives a KNN model is a set of normally distributed numeric attributes that occupy a common range of values. They are ideally Z-scores of continuous data values. Unfortunately, this means that the data being analyzed in this project is a poor candidate for KNN, as it contains a substantial amount of categorical data, and the continuous data that is present tends to be skewed to the right.
Therefore, it results on the following scores and confusion matrix:

Model	Recall	Precision	ROC	F1	**Profit, $**	**Cutoff**
kNN	0.667193	0.80901	0.917020	0.816480	**3,549,621,485.55**	**0.5**



2.6. Discriminant Analysis and Logit Regression

Linear discriminate analysis works best with values that are normally distributed, and while it can play well with binary values, it requires a decently balanced hybrid of binary and continuous values in order to not undermine the underlying assumptions that allow it to function in a meaningful way.
Unfortunately, a lot of the data that we had at our disposal had a substantial amount of multicollinearity and as such we needed to remove a lot of these potential features in order to not pollute the model. What was left when we were done pruning was a sparse selection of features that didn't contribute much to predictions (setting aside the 'Term' feature, which in general was an MVP for our feature list across the board). The resulting model performed poorly and because of our understanding of LDA's needs it was decided that successful optimization of the model would most likely would lead to negligible improvements at best.
Logistic regression faced similar challenges, and we therefore assumed that it would not be efficient in classifying our data.

2.8. Neural Network

Neural Networks stand as versatile and robust models in machine learning, capable of extracting valuable insights from diverse datasets when appropriately engineered. However, their computational demands and time-intensive nature led us to prioritize other approaches. Given these constraints, we opted for a tree-based solution, which proved reliable and significantly less computationally taxing. While a Neural Network solution might offer superior performance with ample resources and time, our current circumstances as busy students limited our capacity to delve into this more time-consuming task.

3. Results

Related Notebook: "result.ipynb"

The XGBoost model we selected yielded an ROC curve (*Figure 10* that closely resembled that of a perfect classifier, indicating a high level of accuracy (Narkhede, 2022).

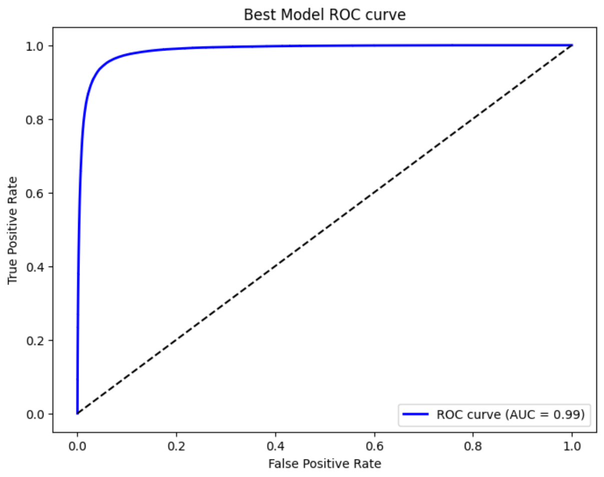


Figure 9. XGBoost model ROC Curve

The confusion matrix (*Figure 11)* for the validation set showed that we correctly identified **68,957** unseen paid in full loans and **13,984** defaulted loans.
Knowing our optimum cut-off probability for future applicant to be **0.5**, we iterated over our dataset to find out that we would optimally accept **80.38%** of the loans when ranked from least to most risky (see *Figure 12)*.

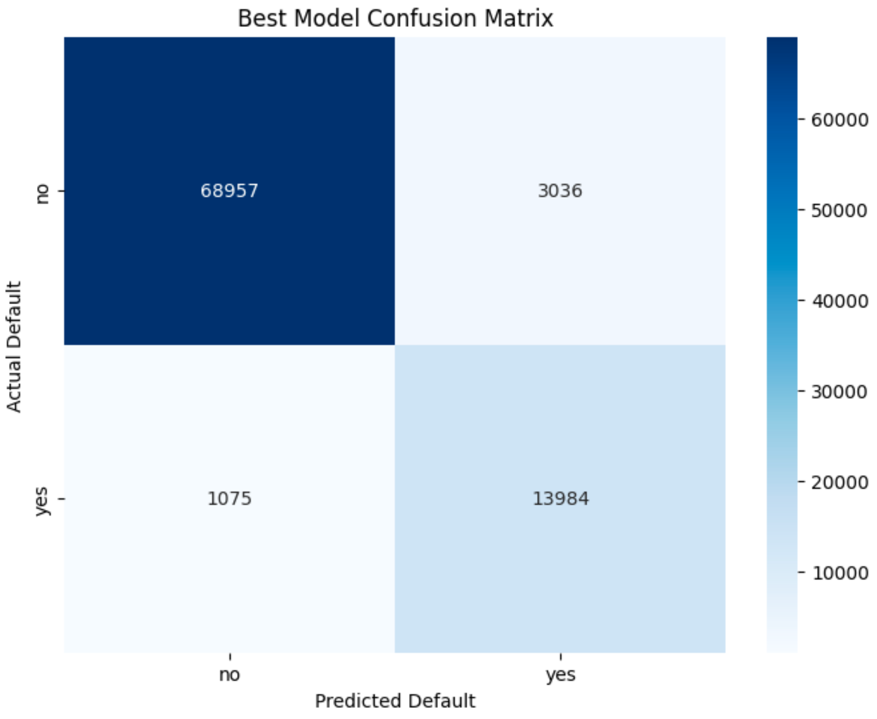
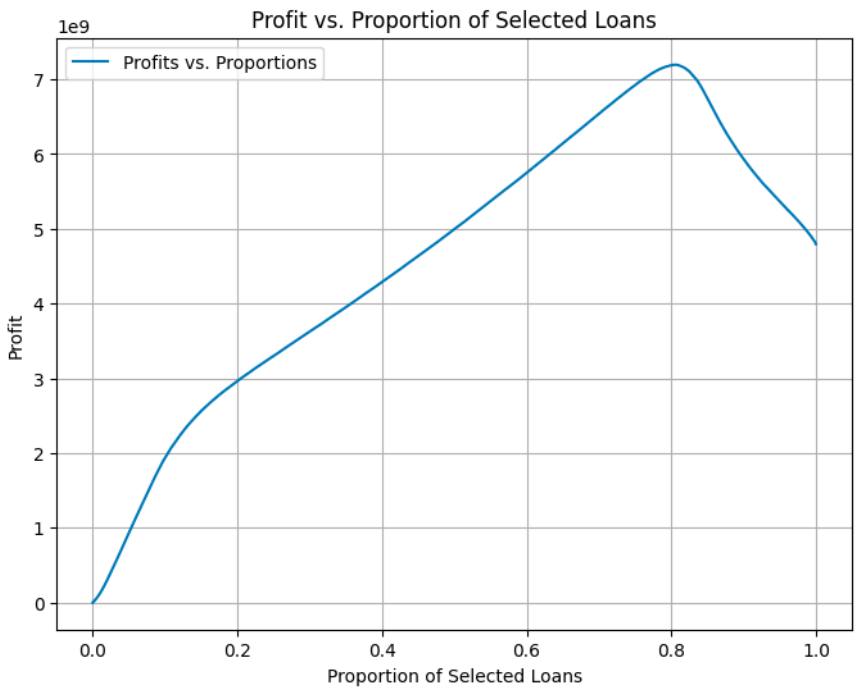


Figure 10. XGBoost model confusion matrix



4. Conclusion

The strengths of a tree-based model shine above other models in situations where there are many binary values from which to define decisions by. While there was a very influential numeric value in the 'Term' feature, the rest of our features were largely categoric and therefore led to the outcome seen above. While we were able to assemble many other models, they all tended to fall flat for one reason or another: Data was skewed, wasn't normally distributed, led to multicollinearity, etc. Navigating the process of optimizing other models was daunting but also drove home the different strengths and weaknesses of the many machine learning models we worked with.
Overall, it has been a very good learning experience.

Bibliography:

North American Industry Classification System (NAICS) U.S. Census Bureau. (n.d.). https://www.census.gov/naics/

Federal funds effective rate. (2023, December 1). https://fred.stlouisfed.org/series/FEDFUNDS

Why does frequency encoding work? (n.d.). Data Science Stack Exchange. https://datascience.stackexchange.com/questions/63749/why-does-frequency-encoding-work

Trevisan, V. (2022, March 24). Target-encoding categorical variables - towards data science. *Medium*. https://towardsdatascience.com/dealing-with-categorical-variables-by-using-target-encoder-a0f1733a4c69

Chip. (2023, December 4). *How to read a correlation heatmap?* QuantHub. https://www.quanthub.com/how-to-read-a-correlation-heatmap/#:~:text=A%20correlation%20heatmap%20is%20a,as%20a%20color%2Dcoded%20matrix.

Mulani, S. (2022, August 3). *Seaborn Kdeplot - a comprehensive guide*. DigitalOcean. https://www.digitalocean.com/community/tutorials/seaborn-kdeplot

Grinsztajn, L. (2022, June 29). *Why do tree-based models still outperform deep learning on typical tabular data?* OpenReview. https://openreview.net/forum?id=Fp7__phQszn

Ameisen, E. (2018, June 20). Always start with a stupid model, no exceptions. - Insight. *Medium*. https://blog.insightdatascience.com/always-start-with-a-stupid-model-no-exceptions-3a22314b9aaa

Brownlee, J. (2021, April 30). *Tour of Evaluation Metrics for Imbalanced Classification*. MachineLearningMastery.com. https://machinelearningmastery.com/tour-of-evaluation-metrics-for-imbalanced-classification/

Decision tree vs logistic regression feature importances. (n.d.). Data Science Stack Exchange. https://datascience.stackexchange.com/questions/116549/decision-tree-vs-logistic-regression-feature-importances#:~:text=In%20decision%20trees%2C%20feature%20importance,splitting%20on%20a%20particular%20feature.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, *24*(2), 123–140. https://doi.org/10.1007/bf00058655

Torres, L. F. (2023, December 5). XGBoost: The King of Machine Learning Algorithms - LatinXinAI - Medium. *Medium*. https://medium.com/latinxinai/xgboost-the-king-of-machine-learning-algorithms-6b5c0d4acd87

Chen, T., & Guestrin, C. (2016). XGBoost. *KDD '16*. https://doi.org/10.1145/2939672.2939785
Narkhede, S. (2022, March 5). Understanding AUC - ROC Curve - towards data science. *Medium*. https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

Saji, B. (2022, July 22). *A quick introduction to K – Nearest neighbor (KNN) classification using Python*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/01/a-quick-introduction-to-k-nearest-neighbor-knn-classification-using-python/