Predicting Customer Churn for Improved Retention Strategies: A Machine Learning Approach in the EV Charging Sector

Clever

***Submission date**:*
24th May
***Number of characters**:*
xxxx
***Number of physical pages**:*
xxxx

BSc in Business Administration and Digital Management

Course
Business Data Analytics, Quantitative Methods and Visualisation

Authors
Frederik Ambye (xxxxxxx)
Storm Bredahl (S169512)

Table of Contents

Introduction & Business Relevance

With increasing environmental concerns and governmental incentives promoting green transportation, the adoption of electric vehicles (EVs) is on the rise. Clever, a market leader in Denmark's EV charging sector, plays a crucial role in developing the country's public charging infrastructure, alongside providing reliable home charging solutions. Clever’s vision is to make the transition from traditional gasoline-powered cars to EVs as seamless and comfortable as possible. This is done by utilizing a subscription business model, offering their customers a simple “All-in-One” subscription that includes both public and home EV charging.

In this market, where competitors are constantly emerging with new offers and technologies, maintaining a loyal customer base is essential. It is well-established that retaining existing customers is more cost-effective and efficient than acquiring new ones (Gallo, 2014). To aid in maintaining customer loyalty, this research will utilize data visualization techniques and machine learning to help identify potential churners before they leave.

Our goal with this paper is to deploy a series of machine learning estimators to predict customer churn using a dataset obtained from Clever. Additionally, we will employ data visualization techniques to uncover patterns and relationships within the data. Throughout the paper, it will be discussed how the research, and its results, can provide business value for Clever, such as deploying our models for customer retention campaigns.

Initial Data Understanding

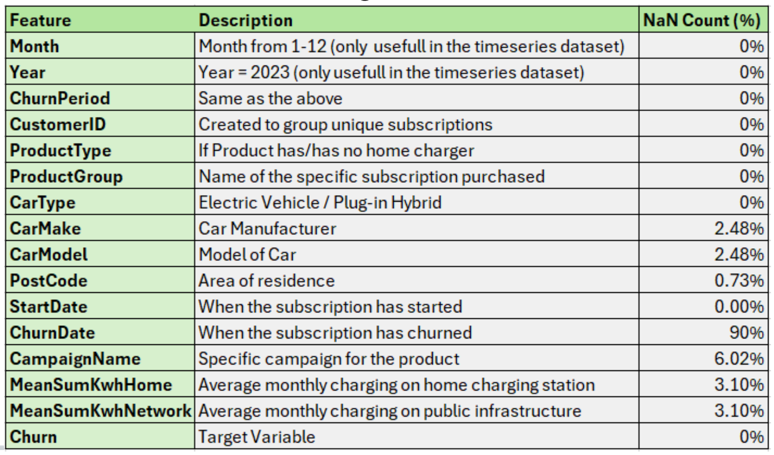
The dataset we were able to obtain from Clever, is a dataset containing all subscriptions that have been active from 2023-01-01 to 2023-12-31. However, it is also registered if these subscriptions have churned up until 2024-02-29. The dataset is temporal, where each row of the dataset monitors a single month where a subscription has been active. The only data that changes within each unique subscription during this period is KwH-usage (Charging usage), which represents the total Kwh-usage a subscription has had during the specific month monitored. The Dataset has 860059 rows by 15 columns. For the scope of this research, we have decided to simplify the dataset by isolating the last row of each unique subscription, which is the row where it is registered if a subscription has churned or not. Since the only data that changes throughout each unique subscription is Kwh-usage, we opted for calculating the mean of all monthly usage a unique subscription has had, and including it in the isolated row. Future analysis could use the temporal data to identify a correlation between churn and increasing/declining Kwh-usage.

The initial dataset contained no unique subscription identifier, so we created one and used it to isolate the last row of each unique subscription. We recommend looking at Appendix 1, which showcases the code used for isolating each unique subscription, calculating mean Kwh-usage, and selecting the last row. The grouping of unique subscriptions has been cross-checked in multiple ways. However, the results of this entire project depend on the grouping being correct. The updated dataset is then used for the remainder of the analysis.

Data Cleaning and Feature Engineering

After converting our dataset, it now contains 107888 rows by 16 columns, with each row now representing each unique subscription that has been active during 2023. While instances of the dataset are unique subscriptions, it is assumed throughout this paper that each subscription represents a unique customer.

Before exploring the dataset further, we wanted to address potential issues with the data based on our domain knowledge.



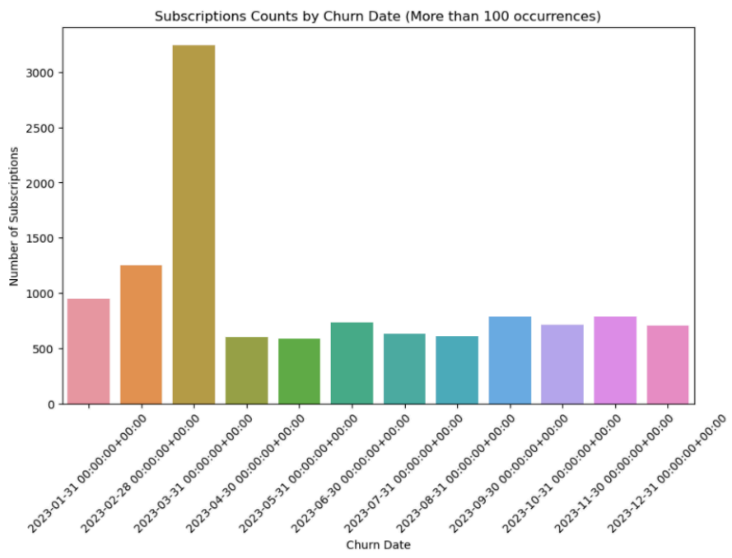
Feature	Description	NaN Count (%)
Month	Month from 1-12 (only usefull in the timeseries dataset)	0%
Year	Year = 2023 (only usefull in the timeseries dataset)	0%
ChurnPeriod	Same as the above	0%
CustomerID	Created to group unique subscriptions	0%
ProductType	If Product has/has no home charger	0%
ProductGroup	Name of the specific subscription purchased	0%
CarType	Electric Vehicle / Plug-in Hybrid	0%
CarMake	Car Manufacturer	2.48%
CarModel	Model of Car	2.48%
PostCode	Area of residence	0.73%
StartDate	When the subscription has started	0.00%
ChurnDate	When the subscription has churned	90%
CampaignName	Specific campaign for the product	6.02%
MeanSumKwhHome	Average monthly charging on home charging station	3.10%
MeanSumKwhNetwork	Average monthly charging on public infrastructure	3.10%
Churn	Target Variable	0%

Removing Converted Subscriptions

Firstly, we removed all instances where 'ChurnDate' is set, but the customers haven't churned. This is because these instances are not churns, they are instead "End-Dates" for a subscription, where the subscription has been converted into a new one. This means that there are duplicate entries of unique customers in the dataset if their subscription has been converted throughout 2023. With the data we have, we have no way of identifying which subscription has been converted into the other. To obtain a clearer overview of Clever's present churn percent, and to avoid our models learning from these 'outdated' instances, we have decided to remove them. This decision, however, causes its own limitations, as certain features such as tenure and usage may be incorrectly perceived in the latest active subscription. Nevertheless, we then have a dataset containing only the latest active subscription from each customer. This process can be seen in Appendix 3.

Removing Churns from 31-03-23

Secondly, we also know from domain knowledge, that Clever had a massive product change in March 2023. This change forced customers to churn if they didn't accept new terms and conditions. The spike can be seen in Graph 1. To avoid our models learning from these "Automatic Churns", we decided to remove all churns which happened on 31-03-23. This will likely also remove some true churns on this date. However, with our obtained data we have no way of differentiating the 'Automatic' churns from true churns. We now have a dataset containing only the latest active subscription from each customer and with excluded churns on 31-03-31.

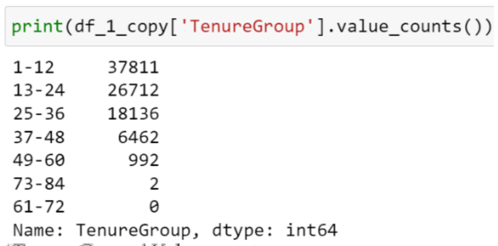


Before removing converted subscriptions and excluding churns on 31-03-31, the churn percentage was 11.52%. After the removals, it is 9.92%. We believe this updated data frame will be more representative of Clever's churn, than if we didn't address these issues at all. A side-by-side comparison of the initial and the revised dataset can be seen in Appendix 4.

The next step in our EDA revolves around cleaning NaN values and engineering new features that we would like to use for our visualizations and models. We have gone through each feature individually and addressed NaN values, alongside handling outliers and deriving new features.

Addressing 'StartDate' and 'ChurnDate'.

Naturally, there are several missing values in 'ChurnDate', as this feature now only contains data if a customer has churned. We wanted to check the correlation between how long a customer has been at Clever and churn. To do this, we created a new feature called 'TenureGroup' from 'StartDate' and 'ChurnDate'. We used 29-02-2024 as the cutoff date for non-churners, as it's the last date where the dataset has registered if a customer has churned. We only had 6 outliers which had more tenure than possible, so we decided to remove these few instances. From 'tenure' we grouped the data into months, and then into



```
print(df_1_copy['TenureGroup'].value_counts())

1-12     37811
13-24    26712
25-36    18136
37-48     6462
49-60      992
73-84        2
61-72        0
Name: TenureGroup, dtype: int64
```

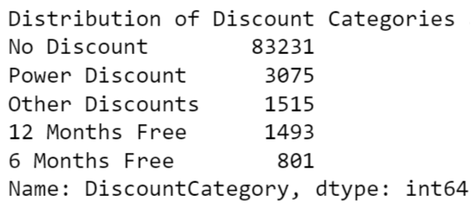
'TenureGroup' Valuecounts

‘TenureGroup’ which represents the group to which a customer belongs from respectively 1-12months, 13-24months, etc. This grouping, however, is limited by the fact that we removed ‘Converted’ Subscriptions. There will be several instances in the 1-12 category, who have been customers for longer but are still captured by the 1-12 category, because their subscription was converted, and thereby started anew. This must be considered when determining the feature’s importance.

Addressing CampaignName

The CampaignName feature represents which campaign a customer has received when purchasing their subscription. We are interested in the cases where a customer has received a discount when purchasing their subscription. Firstly, we want to address the NaN values of this feature. From our domain knowledge, we suspect, that when a customer’s subscription is on hold (‘Bero’ in ‘ProductType’), then they won’t have a campaign. We checked and 94% of missing values in CampaignName, were instances with ‘Bero’ in ‘ProductType’. We replaced these missing values with ‘Bero Campaign’, and the remaining 0.23% of NaN values in the feature with ‘Unknown Campaign’.

The next step in addressing CampaignName is the feature’s high cardinality. There were 688 unique campaigns. From inspecting these campaigns, we could conclude that almost all the campaigns had no meaning to them. They were simply a combination of which product was purchased and the car model. We decided to engineer a new feature called “DiscountCategory” based on CampaignName, which identifies the instances where a customer has received a discount.


```
Distribution of Discount Categories
No Discount        83231
Power Discount      3075
Other Discounts     1515
12 Months Free      1493
6 Months Free        801
Name: DiscountCategory, dtype: int64
```
‘Discount Categories’ Valuecounts

Addressing MeanSumKwhhome and MeanSumKwhNetwork

SumKwhHome & Network contain 3.10% null values. We know from domain knowledge, that these NaN values likely represent when Clever has trouble registering their customer’s usage. We initially experimented with dropping these rows, which were 2872 rows. However, out of these rows, 39.8% of them were churns. We decided to preserve this information by engineering a feature called ‘ChargingIssueFlag’, which captures with ‘1’ if either of MeanSumKwhHome or Network has had a NaN value. We then imputed the mean of the entire columns to replace the NaN values (Chandrikasai, 2023). This way we could have a separate feature that shows if Clever has had problems with registering KwH usage for the customer.

Addressing CarMake, CarModel, and PostCode Missing values

For these remaining three features with missing values, we debated whether the instances were worth keeping or not since they only represent 2.66% of the total data. The 'CarMake' and 'Carmodel' features are dependent on each other, meaning that if there is a missing value in 'CarMake' the car model will also be missing.

Of the 2460 rows which would be dropped, 26.6% of them would be churners. This indicates that the rows with NaN values are likely NMAR (not missing at random), just like with MeanSumKwhHome & Network. Removing the instances will impact model training and predictions, as it could potentially bias our models since they will be trained on data that inherently has a slightly lower churn rate than the true population. This could affect our model's generalization to new data, if new data would include similar patterns of missingness. However, we do not have the knowledge to identify why these rows have missing values. If we assume future data won't have similar patterns of missingness, dropping these rows will be beneficial to our models. In this case, we would not want our models to become biased for 'Unknown' replacement values.

We will assume that future data won't have the same patterns of missingness, and we will therefore continue with an updated data frame, that has dropped all remaining rows with NaN values, which constitutes 2.66% of the total data.

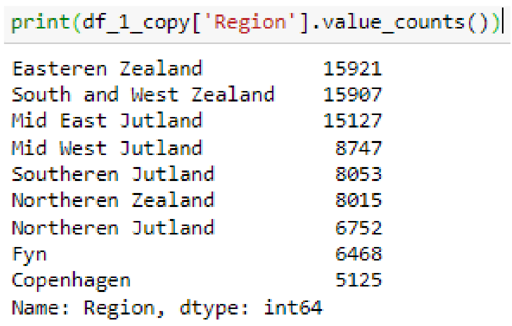
Addressing CarMake & Carmodel's many unique values and outliers

The problem with leaving a feature like "CarModel" with many unique values in the data frame for the models, is that it contains many unique car brands, which could increase the dimensionality of our models significantly when we use one-hot encoding.

High dimensionality can also lead to overfitting, where our models learn noise and specific details from the training data that do not generalize to unseen data. When there are very few data points for a specific "CarModel", then our models might predict 100% churn for that specific CarModel, if all the instances have churned. We will therefore redefine the 'CarModel' feature so it only includes models with over 200 instances of data, the rest are assigned to an 'Other' category. The same is done for the 'CarMake' feature, including only CarMakes with over 100 instances of data, and the rest are assigned to 'Other'. Finally, we corrected mistakes that assigned the same car to two different categories, such as 'Tesla 3' = 'Tesla Model 3'.

Addressing Postcode

For the ‘Postcode’ feature, we decided to engineer a new feature called ‘Region’. We wanted to group ‘Postcode’ into another feature that represents which area of Denmark a respective postcode belongs to. Firstly, we identified that the feature was wrongly assigned as an ‘object’. We inspected which instances were not integers, as all postcodes should be, and found a single instance with a ‘space’ in the middle of the postcode. After correcting, we could convert the feature into an integer. We then defined a function that assigned each postcode to its region-specific category (Appendix 6).



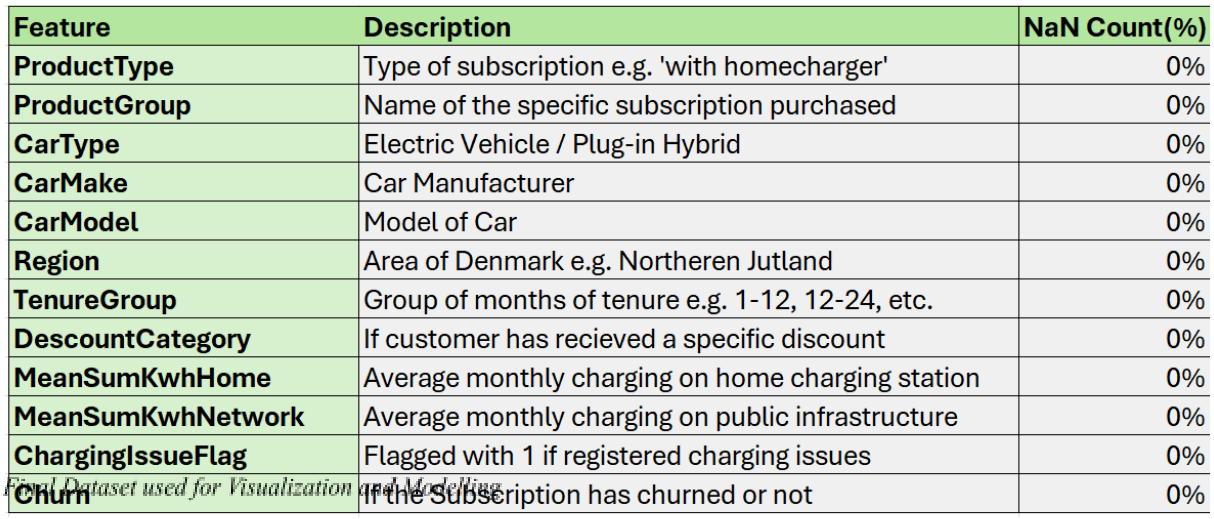
```
print(df_1_copy['Region'].value_counts())
```

```
Easteren Zealand          15921
South and West Zealand    15907
Mid East Jutland          15127
Mid West Jutland           8747
Southeren Jutland          8053
Northeren Zealand          8015
Northeren Jutland          6752
Fyn                        6468
Copenhagen                 5125
Name: Region, dtype: int64
```

'Region' Valuecounts

Final removal of unused columns

Finally, we removed four features that do not provide any value for our further analysis and models. The first feature is ‘CustomerID’, which was used to identify each unique subscription when converting from the time-series dataset. We then removed ‘ChurnPeriod’, ‘Year’, and ‘Month’, as these features were only useful for the initial time-series dataset. Below is the full representation of the data frame after cleaning and engineering, which will be used for visualizations and our models (Unique values and churn % can be seen in Appendix 5).



Feature	**Description**	**NaN Count(%)**
ProductType	Type of subscription e.g. 'with homecharger'	0%
ProductGroup	Name of the specific subscription purchased	0%
CarType	Electric Vehicle / Plug-in Hybrid	0%
CarMake	Car Manufacturer	0%
CarModel	Model of Car	0%
Region	Area of Denmark e.g. Northeren Jutland	0%
TenureGroup	Group of months of tenure e.g. 1-12, 12-24, etc.	0%
DescountCategory	If customer has recieved a specific discount	0%
MeanSumKwhHome	Average monthly charging on home charging station	0%
MeanSumKwhNetwork	Average monthly charging on public infrastructure	0%
ChargingIssueFlag	Flagged with 1 if registered charging issues	0%
Churn	If the Subscription has churned or not	0%

Final Dataset used for Visualization and Modelling

Data Visualization

The visualization of data plays an important role in both understanding and communicating patterns within datasets. By representing Clever's customer data graphically, we can find relationships as well as gain insig hts into what would otherwise remain hidden. Our goal in this section is to visualize and analyze churn among the features in the dataset, using various visualization techniques, aided by Tableau's graphing tools - thereby offering a comprehensive perspective on churn patterns, and the influencing factors behind them.

Our initial objective was to illustrate the distribution of churned and non-churned customers (Figure 1) - using a pie chart, we were able to visualize these proportions clearly and effectively. The light grey segment shows 8,523 customers who have churned, while the dark segment depicts 81,592 customers who have not. We chose to use a pie chart as it can offer a quick and concise overview of the ratio between these two groups.

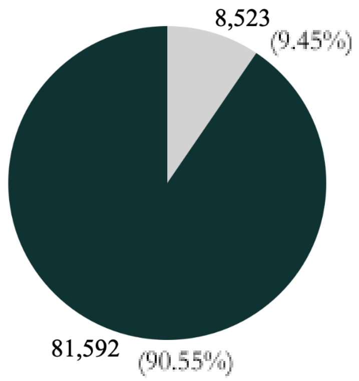


Figure 1 Pie chart displaying churned customers in grey, and customers who haven't churned in green.

Churn by Region

To analyze churn rates regionally, we decided to use a choropleth map for visualization (Figure 2), therefore offering a regional perspective that allows for quick identification of areas with higher churn. The shades of green/grey represent the variation in churn rates across Danish regions, with darker green depicting higher churn rates, and grey indicating lower churn rates; as stated by Heer et al. (2010), we've further ensured that the map uses normalized values (percentages) to avoid misleading interpretations. Displaying churn rates as percentages therefore ensures accurate comparisons between regions with different customer bases (p. 10).

When we consider the additional choropleth map (see Appendix 11) showing the total number of customers per region, further insights emerge:

High Customer Base in Eastern Zealand: The Eastern Zealand region surrounding Copenhagen has the highest concentration of customers, with 15,921 users. Despite this high overall density, the Copenhagen area itself holds only 5,125 customers but still has the highest churn rate at 0.13737. Through our domain knowledge, this discrepancy can be explained by the lack of charging infrastructure and the inability of many users to have *home* charging stations in highly dense living spaces like Copenhagen. Yet, those living in Eastern

Zealand are more likely to have home garages that can implement such home charging stations.

Western Jutland with High User Counts and Moderate Churn: Western Jutland holds a significant customer base, with the largest group located in the Midwest Jutland region (15,127 users). Despite this, they maintain a relatively moderate churn rate of 0.094, pointing to successful customer retention strategies or a lack of competitive alternatives.

Correlation Between Customer Density and Churn: The maps reveal a correlation between customer density and churn rates. While some regions with low customer densities, such as Copenhagen, show higher churn rates, other areas, like Midwest Jutland, maintain lower churn.

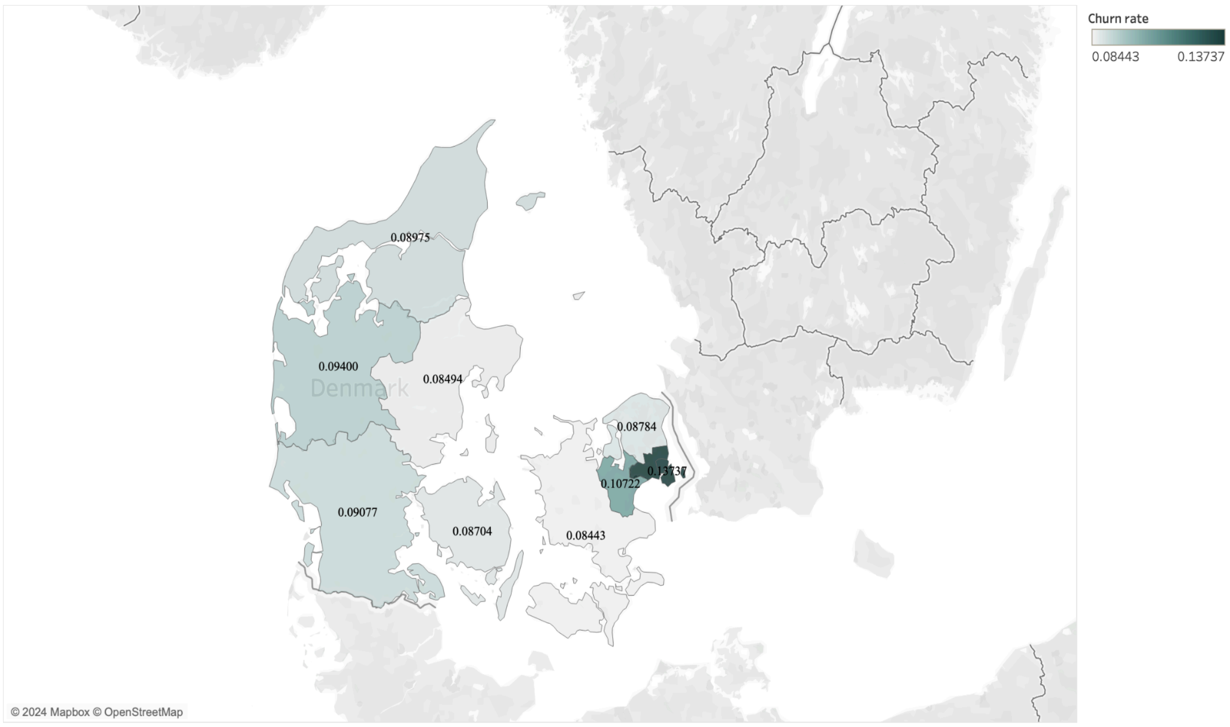


Figure 2. Choropleth map displaying areas with high rates of churn in shades of green, and customers with low rates of churn in dark grey.

Churn by Product Type

The chosen visualization for presenting churn rate by product types (figure 3) uses a dual-axis chart that combines a bar chart and an area chart. This visualization method is selected to effectively display both quantitative counts and rate metrics within the same graphical space, providing an overview of churn across different product types. Each green bar represents the number of customers using a specific product type, while the grey area indicates the churn rate. The bar and area chart displaying churn rates and the number of customers per product type employs key visualization principles as highlighted by Heer et al. (2010) through comparative analysis. Combining bars and area, not only maximizes data density on a single chart but also facilitates a multifaceted understanding of the data by allowing simultaneous visualization of two different metrics, aligning with Heer et al.'s principles of effective visual encoding (Heer et al., 2010, p.26). Albeit traditionally used for temporal data, we believe area charts remain appropriate here in a dual-axis chart due to their ability to both condense data onto a single chart and provide a visual understanding of the relationship between the churn rate and the number of customers per product type.

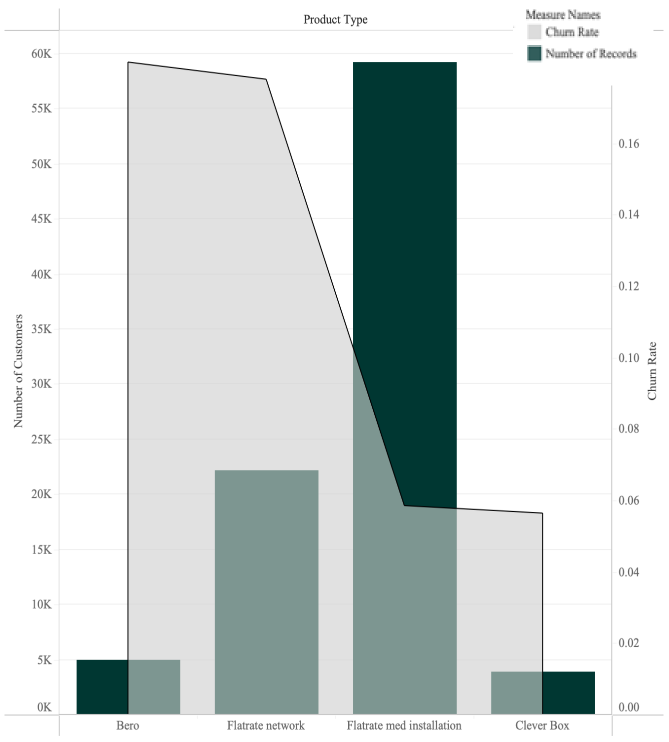


Figure 3. Dual-axis bar and area chart of 'ProductType'

Bero: Subscriptions within the Bero product type are those who have paused subscriptions. These customers appear to have the smallest customer base but also show the highest churn rate, therefore indicating that users are not returning after pausing their subscription, suggesting that further engagement efforts may be necessary to encourage reactivation.

Flatrate Network: This category, involving users who only use public charging stations, has a significant customer base but also a high churn rate. A higher churn rate in 'Flatrate Network' suggests that customers are not as committed to the company, as when they have a home-charging station.

Flatrate med Installation: Customers with a home charging station exhibit a lower churn rate, despite a large customer base. This suggests that users who invest in home charging are more likely to stay committed. This makes sense, as installing a home-charger on average costs 7.999DKK.

Clever Box: The new product launch, Clever Box, shows a relatively low churn rate. This promising trend implies that this new offering aligns well with customer needs, though the current customer base remains small.

Churn by KwH usage on Home-charging station

As with previous charts, we selected the dual-axis bar and area chart to visually represent two key variables: customer count (in bar form) and churn rate (in area form) (figure 4) for varying levels of home EV charging power consumption (Kwh). The choice of this chart aims to simultaneously depict the distribution of customer power usage and the churn rate corresponding to each usage bracket. This visualization provides an effective means to understand the correlation between power consumption and customer churn.

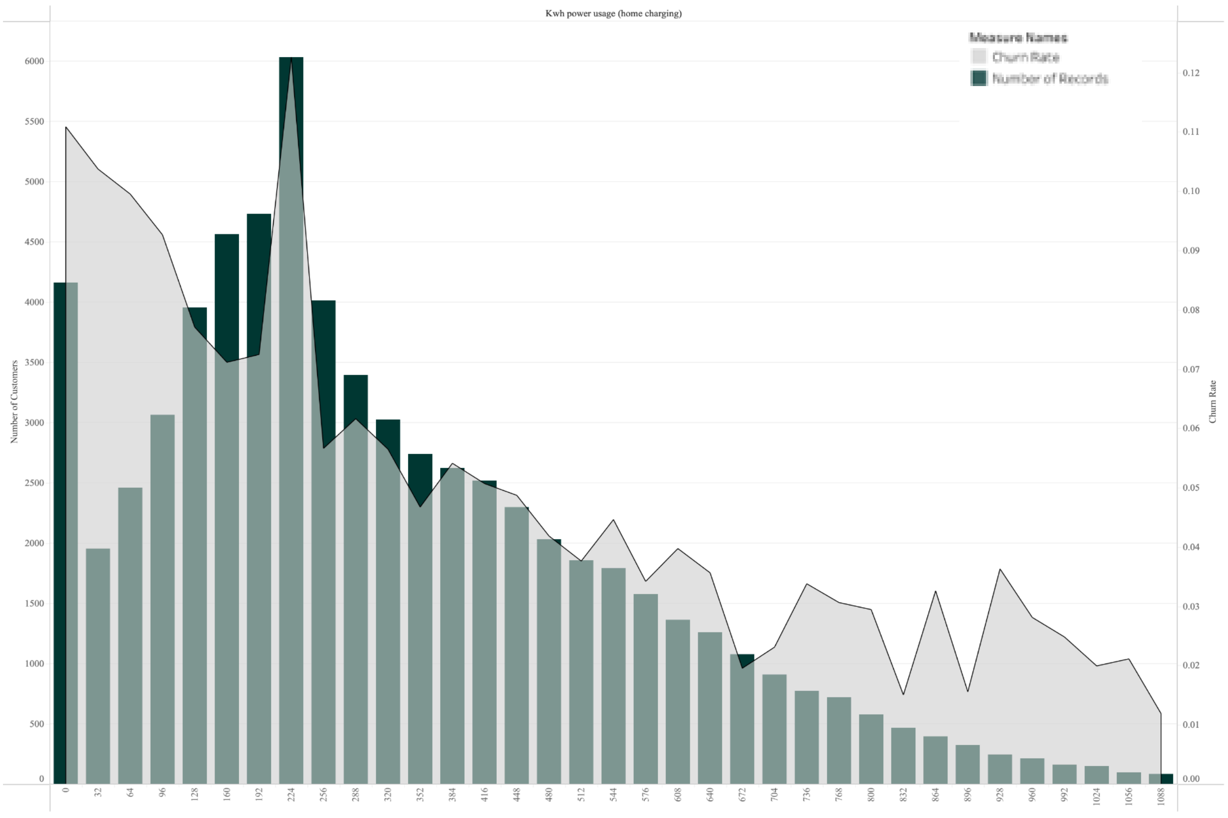


Figure 4. Dual-axis bar and area chart of 'KwhHome'

The churn rate (displayed by the grey area) is highest at the lower and middle levels of power consumption, indicating that customers in this bracket are more likely to leave, yet also have the highest number of customers utilizing that amount of power. As power usage increases past the middle ranges, the churn rate decreases steadily. Low churn rates at higher consumption levels suggest stronger customer retention.

Visualization Conclusions

The visualizations used, following Heer et al. (2010)'s principles, helped us enhance our understanding of Clever's data pre-modelling. The pie chart provided a clear overview of churn rates, the choropleth map shows regional churn rates accurately, and the dual-axis bar and area charts effectively compared customer counts and churn rates across categories. We ensured the use of consistent color schemes and simple designs to maximize clarity and engagement. These visualizations improve data interpretation and presentation quality.

Final Preprocessing

With our cleaned dataset, our final step is encoding to make the dataset ready for our models. We will be creating three separate estimators: a Logistic Regression classifier, a Decision Tree classifier, and a Random Forest classifier. Since most features in our dataset are objects, we must convert them into numerical features. This will be done with one-hot encoding. We have utilized dummy variables from Pandas to perform this step. Our dataset now contains 90115 rows x 166 columns, from which we will perform an 80/20 train/test split, with random state set to 42, and with stratification. Stratification ensures that the class proportions are maintained in both the training and test sets.

Modeling and Results

Determining evaluation metrics

Our dataset is highly imbalanced, with our minority class 'Churn' only representing 9.46% of the data. Because of this, accuracy will be an insufficient scoring metric, since it represents how good our model is at both predicting the positive 'Churn' class and the negative class 'No Churn'. Accuracy is the number of true positives and true negatives divided by all samples (Müller & Guido, 2017, p. 282). We are primarily interested in how well our models can predict the minority class 'Churn'. It is thereby essential to evaluate the models using precision, recall, and f1-scoring.

From a business value perspective, it can be discussed which of the three scoring methods would be most beneficial. Precision measures how many samples predicted as positive are actually positive (Ibid). This measure is used when the goal is to limit the number of false positives (Ibid). However, while precision is important, optimizing for precision may mean that our models might miss out on a high number of true positives, to keep the number of false positives low. When predicting churn, this might not be preferable, because it can be argued that each missed churn prediction would lead to a higher financial loss than a few incorrect churn predictions. If Clever were to, for example, send out a retention campaign to all predicted churners, then mistakenly sending out the campaign to some non-churns would not be as bad as missing the majority of actual churners. This is true if the cost of having customers churn outweighs the cost of offering campaigns to non-churners. Recall, on the other hand, may then be a more relevant evaluation metric.

Recall measures how many of the positive samples (Churn) are captured by the positive predictions (Müller & Guido, 2017, p. 283). From a business perspective, evaluating and tuning with this metric would ensure that most potential churn cases are identified, even at the risk of flagging some non-churn cases incorrectly.

F1-score is the harmonic mean of Precision and Recall, and while it is a better evaluation metric than accuracy for our imbalanced datasheet, we still care more about not missing a churn prediction, than having false positives (incorrectly predicting churn where there are none). We will therefore have Recall in mind when evaluating and tuning our models.

Setting a baseline to compare our models against

In our dataset, the majority class "No Churn" represents 90.54% of the data, which means a naive model that always predicts "No Churn" will achieve an accuracy of 90.54%. In the same sense, if we have a naive model which always predicts 'Churn', it will have an accuracy of 9.46%. This naive model would have perfect recall, but very low precision and accuracy. However, we can also calculate the F1 score to have a baseline to compare our models against. In our case, a naive model that always predicts churn would have a precision of 0.09, a recall of 1, and thereby an f1 score of 0.17.

Addressing the imbalanced datasheet – SMOTE and RandomUndersampler

Since our dataset is heavily imbalanced, we will also train our models on oversampled and undersampled training data and compare the results to the initial models. We will use Imbalanced Learn's RandomUndersampler (RUS) to reduce our majority class (non-churners). We will, respectively, also use their Synthetic Minority Over-Sampling Technique (SMOTE) to increase our minority class (churners). Both techniques are attempts to balance our training data, creating an incentive for our models to predict the minority class.

Objective of our models

For each of our three separate estimators, we will create four tests and compare them with each other. We will create an initial model, without tuning and data balancing techniques. We will cross-validate the model using GridSearchCV, and determine the optimal hyperparameters for the model. We will also train the models on training data with, respectively, SMOTE and RUS balancing. The models will be compared to each other, and to our naive model which always predicts churn. Finally, we will look at coefficients from a logistic regression model to identify feature importance. A full visualization of model results can be in the model conclusion section.

Logistic Regression Classifier

For the logistic regression model, we created a pipeline that standard scales our two numeric features 'MeanSumKwhHome' and 'MeanSumKwhNetwork'. This is to avoid the coefficients being affected by the different scales of the two numeric features and the one-hot-encoded features. The initial model got a train accuracy score of 90.5%, and a test accuracy score of 90.6%, meaning that the model is neither overfitting nor underfitting. The model is performing very poorly, however, with a precision score of 0.48, but only 0.04 in recall, determining a 0.07 F1-score. Our model is only identifying 4% of actual churn cases, and 50% of these instances predicted as churn were correctly identified. The low recall is expected for a logistic regression model trained on an imbalanced datasheet. This is because LR models without any adjustments for class balance tend to favor the majority class. However, we will now address the class imbalance with SMOTE and RUS techniques.

Logistic regression with class imbalance adjustments

To address our dataset class imbalance, we resampled our training data with SMOTE and RUS respectively. We then fitted our logistic regression pipeline to the resampled training data and tested it on the unmodified test data. Our model using the synthetic samples generated by SMOTE did not improve the model significantly, meaning that the samples created are not representative of any churn patterns.

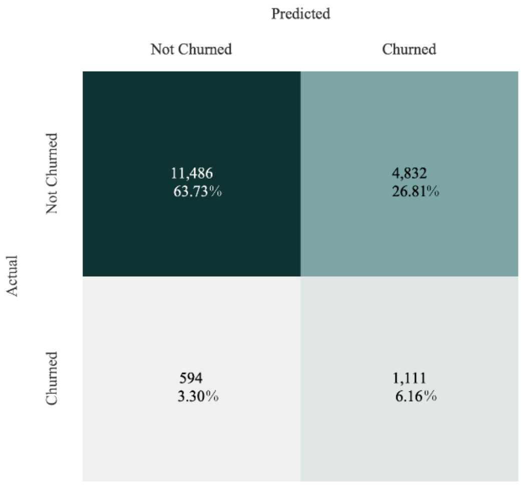


Figure 5. Logistic Regression with RUS confusion matrix.

Balancing the training data with RUS, however, has significantly influenced the model's scoring. Reducing the size of our majority class has led to the model scoring 0.19 in Precision, 0.65 in Recall, with an f1-score of 0.29. With this balancing technique, our model can now identify 65% of all actual churn cases. This has come with the tradeoff of lower precision. From the confusion matrix (Figure 5), we can see that the model now correctly predicts the true negatives 11486 times (non-churners). The model only misses identifying 594 customers who churn, but correctly predicts 1111 churners. The tradeoff between recall and precision can be seen, as the model mistakes 4832 customers for churners, when they were actually non-churners. The train and test accuracy has fallen to 69.8% and 69.5% respectively, indicating that the model is likely underfitting. Either the model is too simple, or our features do not have enough importance.

We then performed cross-validation to validate the consistency of the LogReg model among 5 folds. This ensures that our initial test results were not due to any particular split of the data. We created a new pipeline, using IMBlearn's pipeline, to include RUS directly in the cross-validation process (Muralidhar, 2021). The process was then conducted on the original training data, incorporating RUS as part of each fold's training process in the cross-validation loop. The results we got were 0.30 in F1, 0.67 in recall, and 0.19 in precision, which almost doesn't differ from our model's results on the test data, which helps confirm our model's generalizability. See Appendix 8 for the process.

Decision Tree Classifier

Our initial decision tree obtained a train accuracy of 99.8%, but only a test accuracy of 85.5%. This indicates that the model is overfitting, which is common for decision trees, as they will keep attempting to guess until it is confident in predicting the right class in the training data, hence the high accuracy of the training data. The model performed worse than our logistic regression model, with and without RUS/SMOTE.

Random Forest Classifier

Since the decision tree is overfitting, we can try to address this problem using the Random Forest Classifier (Müller & Guido, 2017, p. 83). The Random Forest estimator attempts to improve prediction accuracy by combining multiple decision trees. The initial model does have higher test accuracy than our decision tree, the initial model also performs better than our initial logistic regression model. Notably, the Random Forest model with SMOTE balancing performs the best out of all our models, when evaluated with F1-scoring, at 0.31. When deploying the model on RUS training data, it is however just below its logistic regression counterpart in terms of recall and F1-score. A confusion matrix can be seen in Appendix 9, comparing it to the LR counterpart.

Tuning the models & Cross-validation of Random Forest and Decision Tree Classifier

To optimize our results, we also attempted to conduct a GridSearchCV to cross-search different combinations of hyperparameters for our separate models with scoring set to 'recall'. The Gridsearch was done over 5 folds. The results are shown in the model-scorings visualization (Figure 6), and an example of our Gridsearch for the random forest classifier can be seen in Appendix 8. By tuning the parameters of all three of our models, they barely improved. The tuned decision tree was the only model that significantly benefited from the tuning, with an increase of 0.042 in test accuracy. We may have too many features that are not contributing to the model's predictive power, which would hinder our parameter tuning from having a significant impact. Additionally, we often ran into computational issues when trying to search for a broader range of parameters.

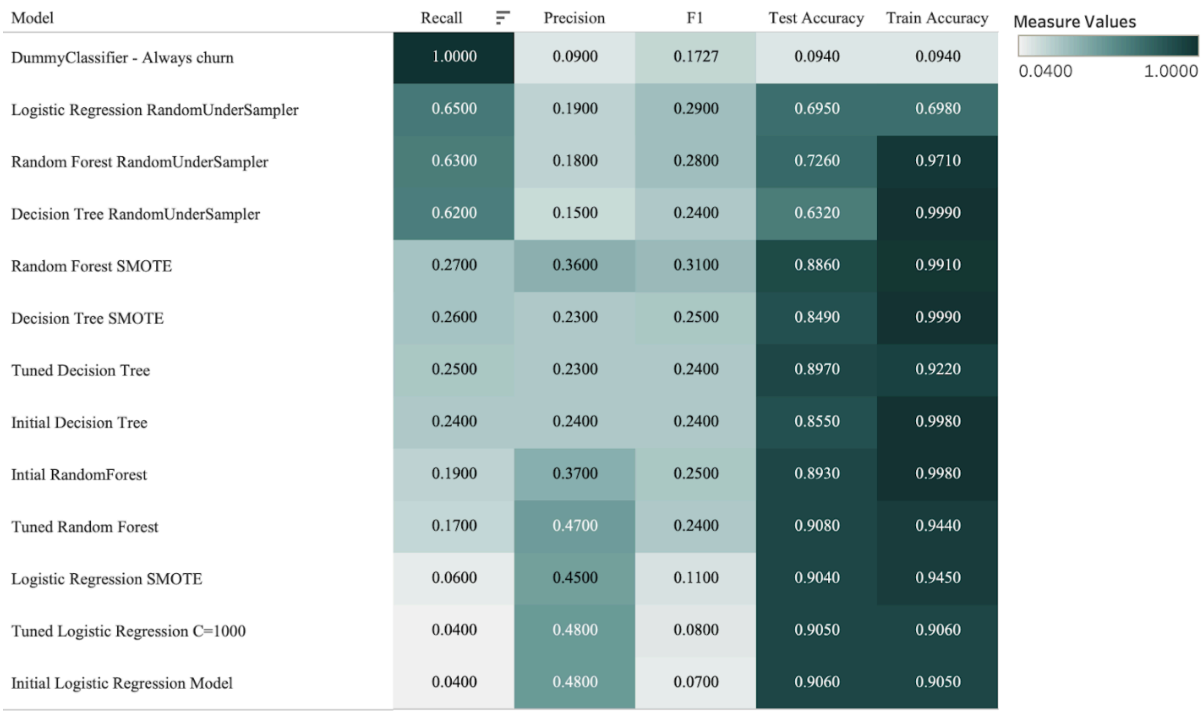
Logistic Regression Coefficient Report

To evaluate the importance of our features, the coefficients from our logistic regression model can provide insights. These coefficients tell us how each feature is associated with churn. A positive coefficient shows that if the feature value is high, so is churn. Contrary to this, if the coefficient is negative, then there is less probability of the model predicting churn. The coefficient report shows that we have some features with a very high impact on churn prediction. Our created 'ChargingIssueFlag' has the highest coefficient at 1.65, suggesting that if Clever has problems registering charging with a customer, then the probability of churn increases. Likewise, the '6 months free discount', and a couple of different Car models, also have over 1 in their coefficient, suggesting a correlation between the features and churn. On the other hand, the Power discount category has a negative correlation of -3.05, suggesting that the feature contributes to fewer churn instances. Another interesting conclusion from the report shows that as 'TenureGroup' increases, the respective coefficient decreases. This indicates that the longer a customer has been at Clever, the less likely they are to churn. The vast majority of all our features hover around 0.5 to -0.5, indicating that their importance is not crucial to predicting churn, which is a valuable insight in itself. An overview of insights from the coefficient report can be seen in Appendix 10.

Model Conclusions

Our success with RUS compared to SMOTE suggests that reducing the amount of data we have performs better than increasing noise which may be generated by SMOTE's synthetic data generation. This likely correlates with our conclusions from the coefficient report, which suggests that many of our features are not important for identifying churn. There are simply too many features in our dataset with no correlation with our target variable 'churn', resulting in our models having poor predictive power for the minority class.

Nevertheless, we can conclude that if Clever were to send out a campaign to customers based on our best-performing model: the RUS Logistic Regression model's test results, then the campaign would correctly reach 1111 churners, and only miss 594 churners. It would however also falsely reach 4832 customers who were not considered churners. It is here



Model	Recall	Precision	F1	Test Accuracy	Train Accuracy
DummyClassifier - Always churn	1.0000	0.0900	0.1727	0.0940	0.0940
Logistic Regression RandomUnderSampler	0.6500	0.1900	0.2900	0.6950	0.6980
Random Forest RandomUnderSampler	0.6300	0.1800	0.2800	0.7260	0.9710
Decision Tree RandomUnderSampler	0.6200	0.1500	0.2400	0.6320	0.9990
Random Forest SMOTE	0.2700	0.3600	0.3100	0.8860	0.9910
Decision Tree SMOTE	0.2600	0.2300	0.2500	0.8490	0.9990
Tuned Decision Tree	0.2500	0.2300	0.2400	0.8970	0.9220
Initial Decision Tree	0.2400	0.2400	0.2400	0.8550	0.9980
Intial RandomForest	0.1900	0.3700	0.2500	0.8930	0.9980
Tuned Random Forest	0.1700	0.4700	0.2400	0.9080	0.9440
Logistic Regression SMOTE	0.0600	0.4500	0.1100	0.9040	0.9450
Tuned Logistic Regression C=1000	0.0400	0.4800	0.0800	0.9050	0.9060
Initial Logistic Regression Model	0.0400	0.4800	0.0700	0.9060	0.9050

Figure 6. Visualization of model scorings.

where Clever could evaluate if such a campaign would be profitable. If the cost of potentially having the 1111 customers churn outweighs the costs of sending discounts to the 4832 non-churners, then our model could be deployed for strategic decision-making. Figure 6 shows a full overview of each model's scoring, sorted by Recall.

Reflections

Receiving a dataset of this magnitude without additional domain knowledge about the company has presented a challenge. Having more knowledge about the data would certainly influence the research. Our finding that instances with NaN values highly correlate with churn could possibly be explained through domain knowledge, and thereby other strategies to handle these missing values could be developed, allowing the models to be improved. Additionally, knowledge about how to handle 'converted' subscriptions, and 'Automatic Churns' from 31-03-23, would also have altered the results of this research. Finally, revisiting feature analysis with the knowledge gained from the research to be more thorough with feature selection could lead to further improved models. In a larger project, exploring additional features and further interactions between existing features would benefit model performances.

Conclusion

The goal of our research has been to identify underlying indicators of churning customers at Clever, alongside building an estimator which can successfully predict churn. We have cleaned and pre-processed our data while engineering new features to identify churn. The dataset has been visualized, showing patterns between churn and several features. We have argued that 'Recall' is the best scoring method when evaluating our models for this specific churn case. From our modeling, we have created three separate estimators, from which the best was our Logistic Regression model on Random Under-sampled training data, scoring 65% in recall and 19% in precision. This model could be used to send out targeted campaigns for retention strategies. The Logistic Regression model's coefficient report shows us which features are important to predict churn. It reveals that churn is likely when there are issues with Clever registering charging, and with the 6-month free campaign. Furthermore, it indicates that customers are less likely to churn the longer they have been customers. Overall, the results from our research can potentially be profitable for Clever if it were to be used for customer retention strategies.

References

Chandrikasai. (2023, April 11). When to use mean, median, and mode for imputing missing values. *Medium.com*. https://medium.com/@chandrikasai9997/imputing-missing-values-is-another-technique-used-to-handle-missing-data-in-a-dataset-824957ce71b4#:~:text=Mean%3A%20The%20mean%20is%20a,not%20depend%20on%20unobserved%20variables.

Gallo, A. (2014, October 29). The value of keeping the right customers. *Harvard Business Review*. https://hbr.org/2014/10/the-value-of-keeping-the-right-customers

Heer et al. (2010). *A Tour through the Visualization Zoo*. Communications of the ACM.

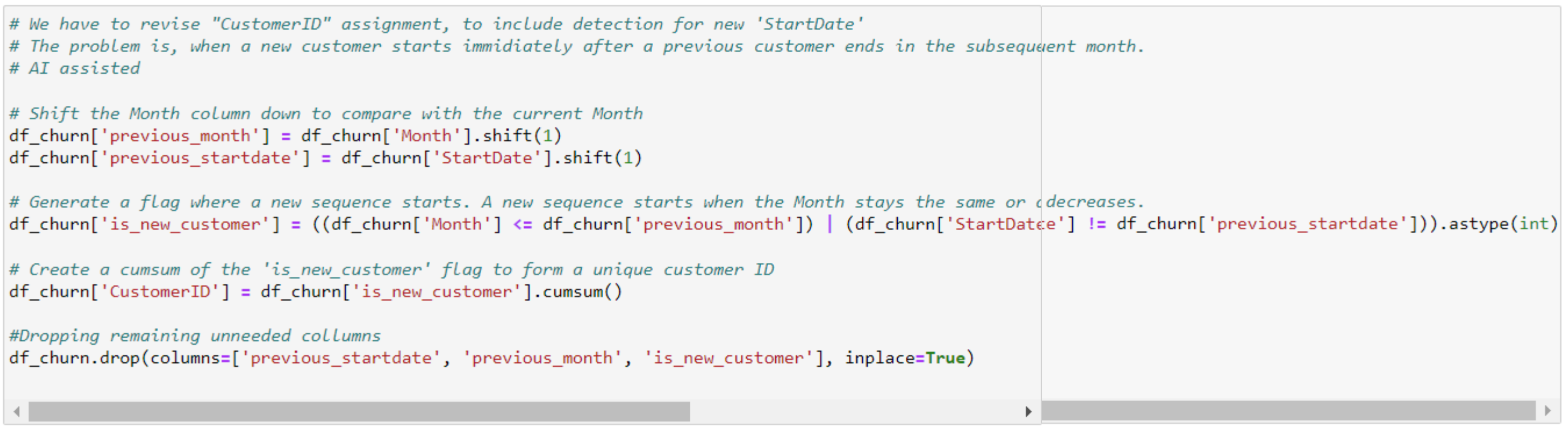
Muralidhar, K. (2021, March 29). The right way of using SMOTE with cross-validation. *Towards Data Science.* https://towardsdatascience.com/the-right-way-of-using-smote-with-cross-validation-92a8d09d00c7

Müller, A. C., & Guido, S. (2017). *Introduction to Machine Learning with Python*. O'Reilly Media, Inc.

Appendixes

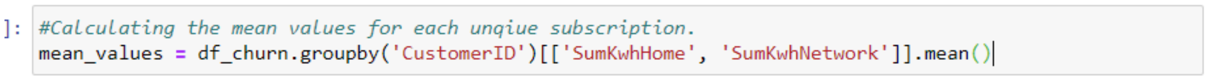
Appendix 1 – Time-series dataset conversion

Final grouping code after tweaking:

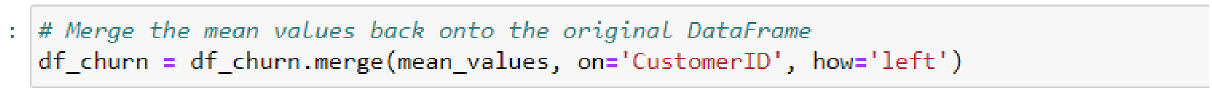


```
# We have to revise "CustomerID" assignment, to include detection for new 'StartDate'
# The problem is, when a new customer starts immidiately after a previous customer ends in the subsequent month.
# AI assisted

# Shift the Month column down to compare with the current Month
df_churn['previous_month'] = df_churn['Month'].shift(1)
df_churn['previous_startdate'] = df_churn['StartDate'].shift(1)

# Generate a flag where a new sequence starts. A new sequence starts when the Month stays the same or decreases.
df_churn['is_new_customer'] = ((df_churn['Month'] <= df_churn['previous_month']) | (df_churn['StartDate'] != df_churn['previous_startdate'])).astype(int)

# Create a cumsum of the 'is_new_customer' flag to form a unique customer ID
df_churn['CustomerID'] = df_churn['is_new_customer'].cumsum()

#Dropping remaining unneeded collumns
df_churn.drop(columns=['previous_startdate', 'previous_month', 'is_new_customer'], inplace=True)
```

Collecting the mean of 'SumKwhHome & Network'.

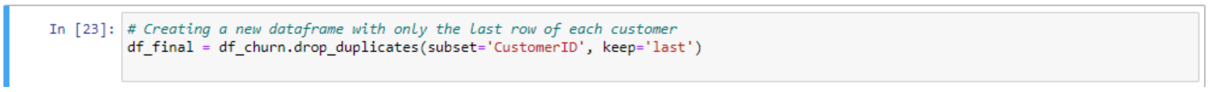


```
]: #Calculating the mean values for each unqiue subscription.
   mean_values = df_churn.groupby('CustomerID')[['SumKwhHome', 'SumKwhNetwork']].mean()
```



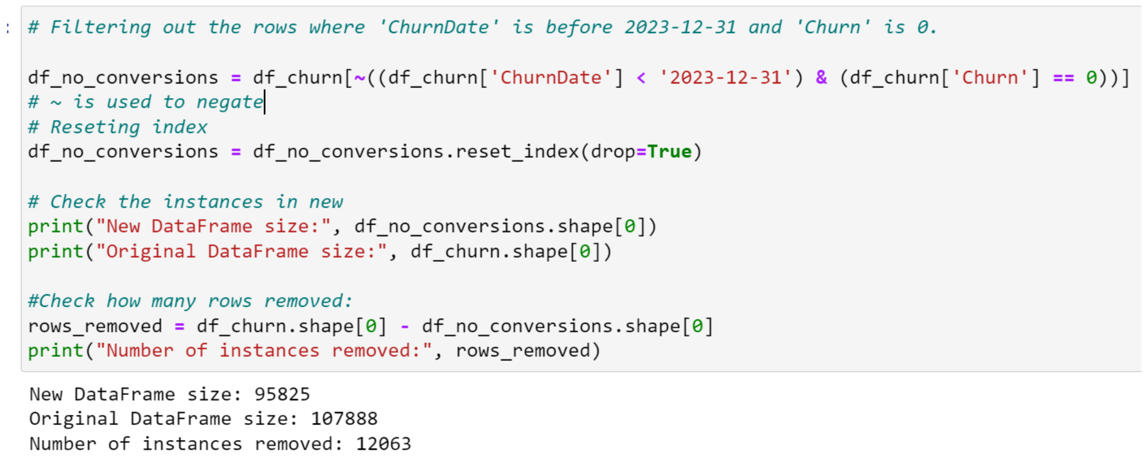
```
: # Merge the mean values back onto the original DataFrame
  df_churn = df_churn.merge(mean_values, on='CustomerID', how='left')
```

Including only the last row of each unique Customer.



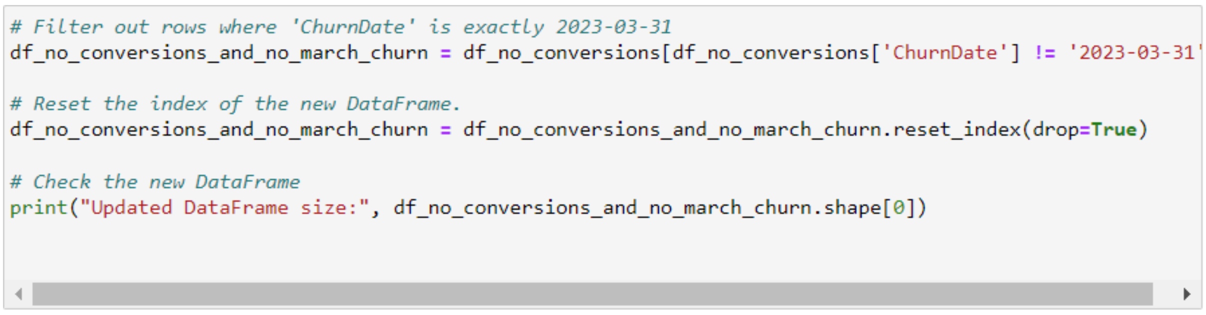
```
In [23]: # Creating a new dataframe with only the last row of each customer
         df_final = df_churn.drop_duplicates(subset='CustomerID', keep='last')
```

Appendix 2 – Removing Converted Subscriptions



```
: # Filtering out the rows where 'ChurnDate' is before 2023-12-31 and 'Churn' is 0.

  df_no_conversions = df_churn[~((df_churn['ChurnDate'] < '2023-12-31') & (df_churn['Churn'] == 0))]
  # ~ is used to negate
  # Reseting index
  df_no_conversions = df_no_conversions.reset_index(drop=True)

  # Check the instances in new
  print("New DataFrame size:", df_no_conversions.shape[0])
  print("Original DataFrame size:", df_churn.shape[0])

  #Check how many rows removed:
  rows_removed = df_churn.shape[0] - df_no_conversions.shape[0]
  print("Number of instances removed:", rows_removed)

  New DataFrame size: 95825
  Original DataFrame size: 107888
  Number of instances removed: 12063
```

Appendix 3 – Removing Churns from 31-03-23.

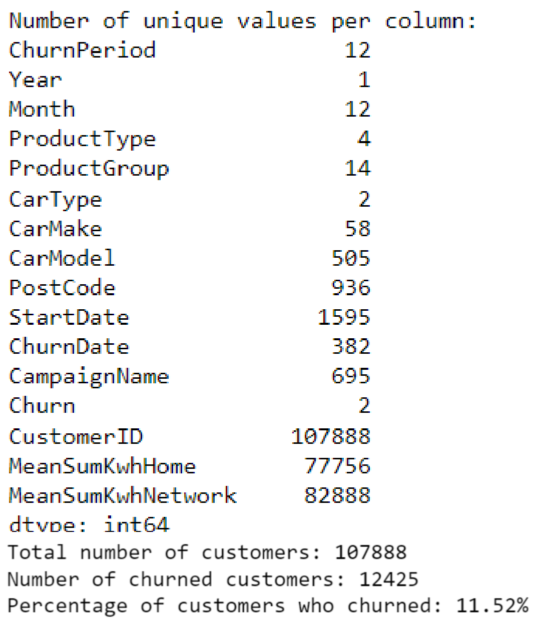


```python
# Filter out rows where 'ChurnDate' is exactly 2023-03-31
df_no_conversions_and_no_march_churn = df_no_conversions[df_no_conversions['ChurnDate'] != '2023-03-31'

# Reset the index of the new DataFrame.
df_no_conversions_and_no_march_churn = df_no_conversions_and_no_march_churn.reset_index(drop=True)

# Check the new DataFrame
print("Updated DataFrame size:", df_no_conversions_and_no_march_churn.shape[0])
```

```
Updated DataFrame size: 92581
```

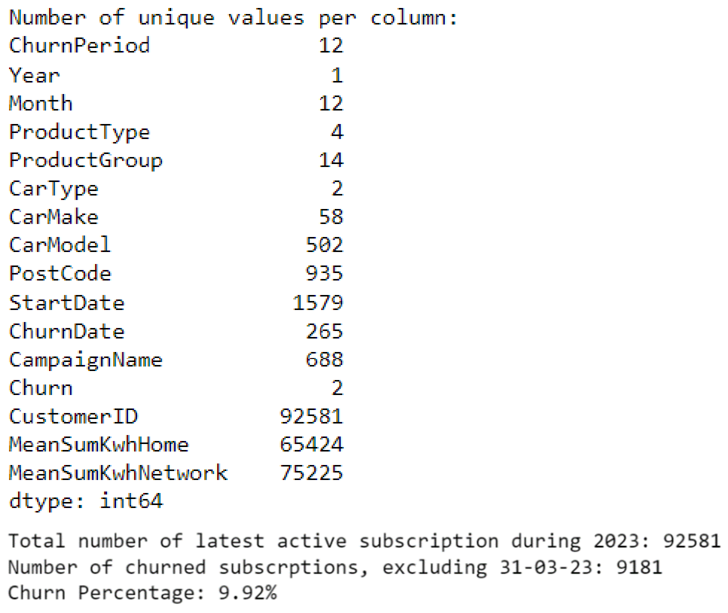
Appendix 4– Side by side between initial dataset and dataset after removing converted subscriptions and churns on the 31-03-23.

Initial Dataset 1



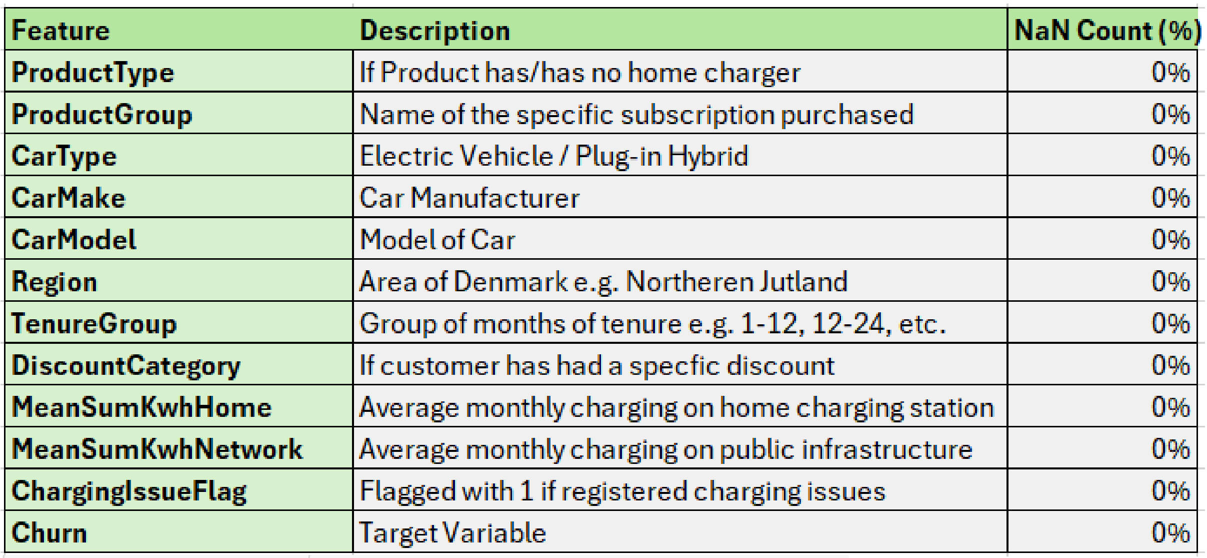
```
Number of unique values per column:
ChurnPeriod              12
Year                      1
Month                    12
ProductType               4
ProductGroup             14
CarType                   2
CarMake                  58
CarModel                505
PostCode                936
StartDate              1595
ChurnDate               382
CampaignName            695
Churn                     2
CustomerID           107888
MeanSumKwhHome        77756
MeanSumKwhNetwork     82888
dtype: int64
Total number of customers: 107888
Number of churned customers: 12425
Percentage of customers who churned: 11.52%
```

Updated Dataset 2

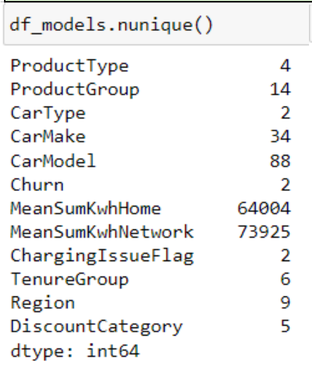


```
Number of unique values per column:
ChurnPeriod              12
Year                      1
Month                    12
ProductType               4
ProductGroup             14
CarType                   2
CarMake                  58
CarModel                502
PostCode                935
StartDate              1579
ChurnDate               265
CampaignName            688
Churn                     2
CustomerID            92581
MeanSumKwhHome        65424
MeanSumKwhNetwork     75225
dtype: int64

Total number of latest active subscription during 2023: 92581
Number of churned subscrptions, excluding 31-03-23: 9181
Churn Percentage: 9.92%
```

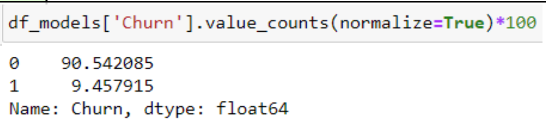
Appendix 5 – Final Cleaned dataset used for visualization and modeling.



Feature	Description	NaN Count (%)
ProductType	If Product has/has no home charger	0%
ProductGroup	Name of the specific subscription purchased	0%
CarType	Electric Vehicle / Plug-in Hybrid	0%
CarMake	Car Manufacturer	0%
CarModel	Model of Car	0%
Region	Area of Denmark e.g. Northeren Jutland	0%
TenureGroup	Group of months of tenure e.g. 1-12, 12-24, etc.	0%
DiscountCategory	If customer has had a specfic discount	0%
MeanSumKwhHome	Average monthly charging on home charging station	0%
MeanSumKwhNetwork	Average monthly charging on public infrastructure	0%
ChargingIssueFlag	Flagged with 1 if registered charging issues	0%
Churn	Target Variable	0%

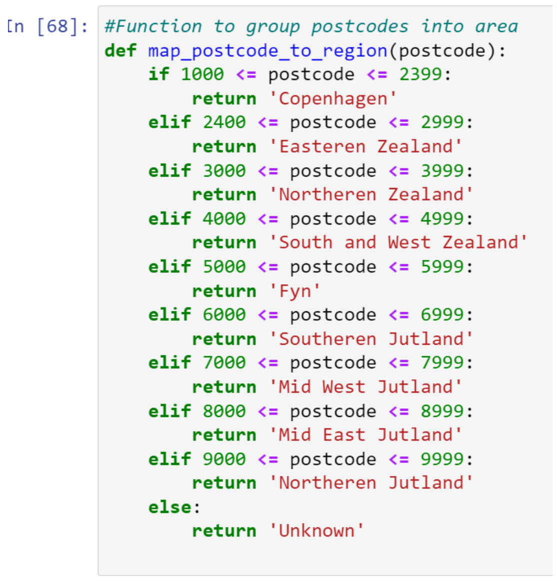


```
df_models.nunique()

ProductType              4
ProductGroup            14
CarType                  2
CarMake                 34
CarModel                88
Churn                    2
MeanSumKwhHome       64004
MeanSumKwhNetwork    73925
ChargingIssueFlag        2
TenureGroup              6
Region                   9
DiscountCategory         5
dtype: int64
```



```
df_models['Churn'].value_counts(normalize=True)*100

0    90.542085
1     9.457915
Name: Churn, dtype: float64
```

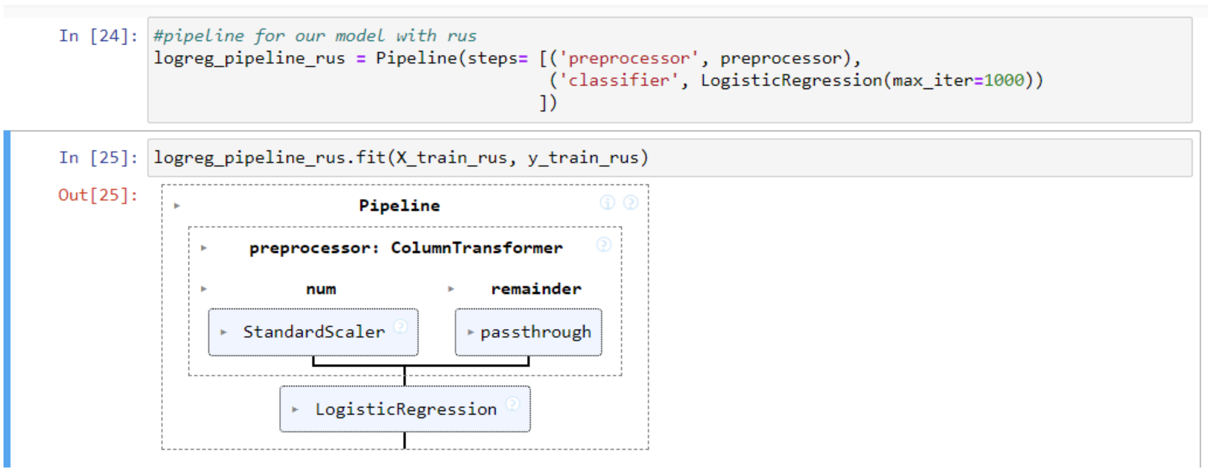
Appendix 6 – Postcode Grouping Function



In [68]:

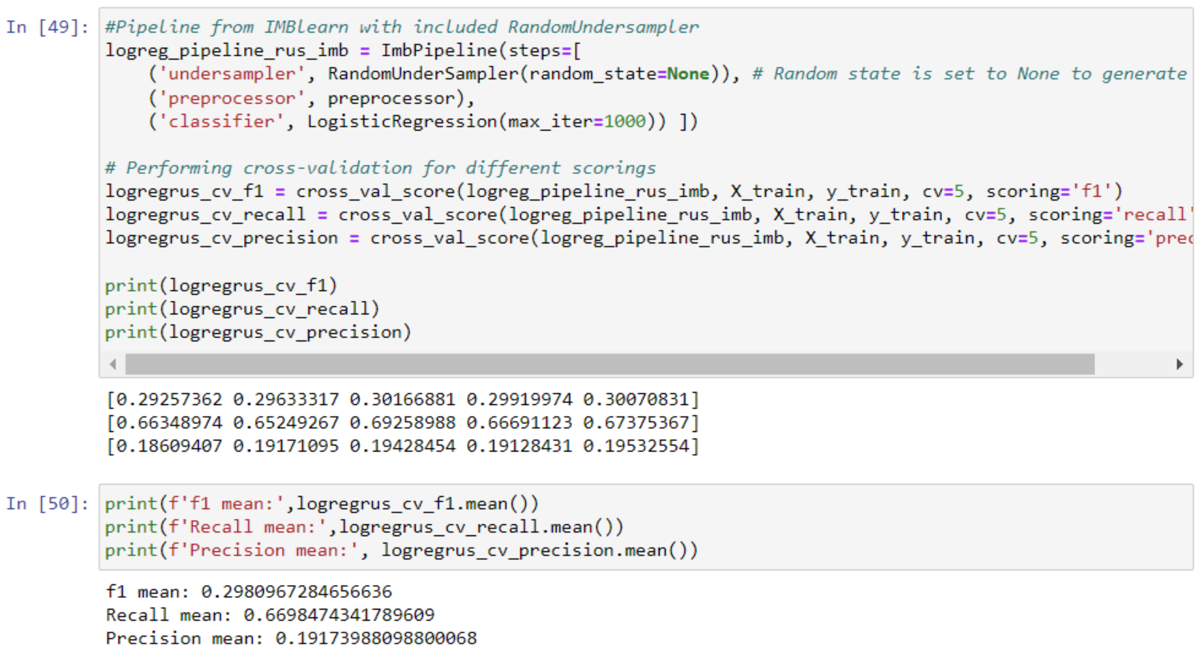
```python
#Function to group postcodes into area
def map_postcode_to_region(postcode):
    if 1000 <= postcode <= 2399:
        return 'Copenhagen'
    elif 2400 <= postcode <= 2999:
        return 'Easteren Zealand'
    elif 3000 <= postcode <= 3999:
        return 'Northeren Zealand'
    elif 4000 <= postcode <= 4999:
        return 'South and West Zealand'
    elif 5000 <= postcode <= 5999:
        return 'Fyn'
    elif 6000 <= postcode <= 6999:
        return 'Southeren Jutland'
    elif 7000 <= postcode <= 7999:
        return 'Mid West Jutland'
    elif 8000 <= postcode <= 8999:
        return 'Mid East Jutland'
    elif 9000 <= postcode <= 9999:
        return 'Northeren Jutland'
    else:
        return 'Unknown'
```

Appendix 7 - LogReg Model Random Undersampler


```
In [24]: #pipeline for our model with rus
         logreg_pipeline_rus = Pipeline(steps= [('preprocessor', preprocessor),
                                                 ('classifier', LogisticRegression(max_iter=1000))
                                                ])

In [25]: logreg_pipeline_rus.fit(X_train_rus, y_train_rus)

Out[25]:
```

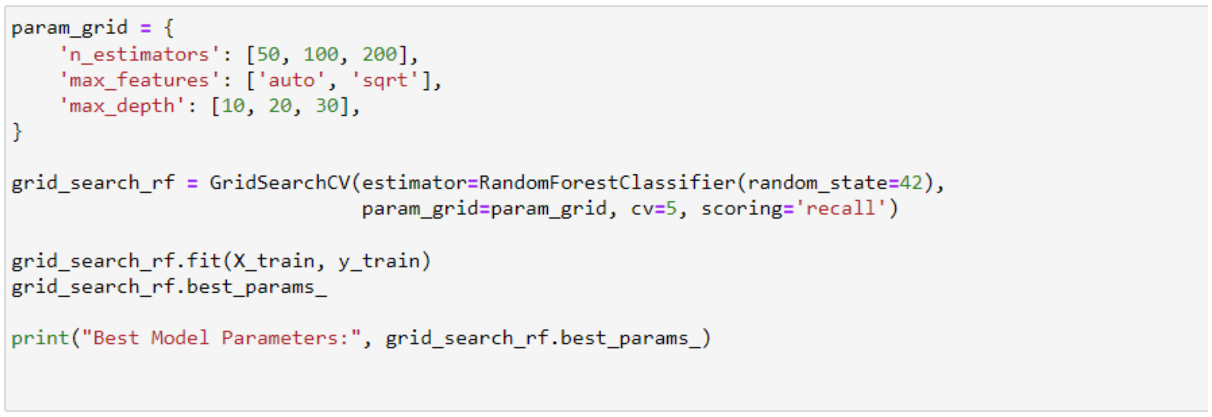
```
Training Accuracy: 0.6956585508946905
Test Accuracy: 0.6989402430228042
Classification Report (Test):
              precision    recall  f1-score   support

           0       0.95      0.70      0.81     16318
           1       0.19      0.65      0.29      1705

    accuracy                           0.70     18023
   macro avg       0.57      0.68      0.55     18023
weighted avg       0.88      0.70      0.76     18023

Confusion Matrix:
 [[11486  4832]
 [  594  1111]]
```

Appendix 8 - Cross-validation of Logistic Regression Model



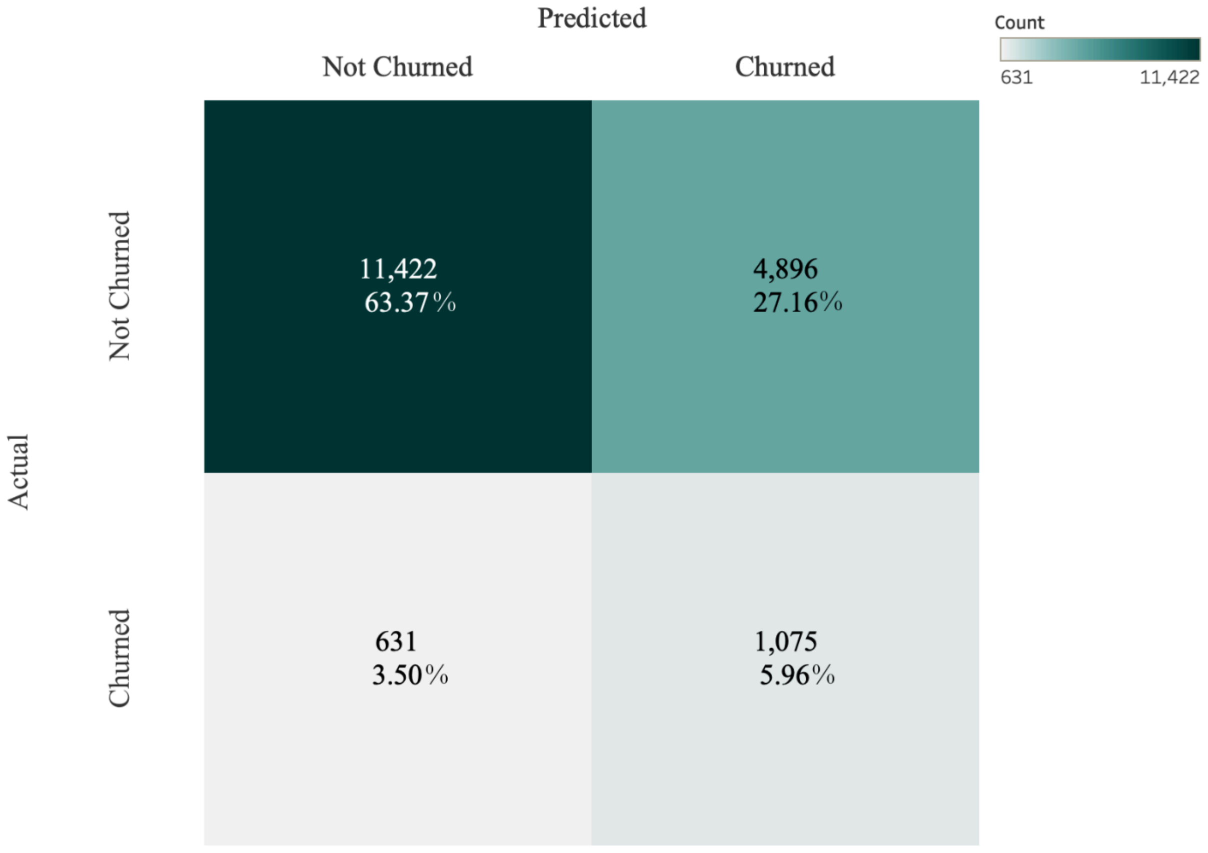
```
In [49]: #Pipeline from IMBlearn with included RandomUndersampler
         logreg_pipeline_rus_imb = ImbPipeline(steps=[
             ('undersampler', RandomUnderSampler(random_state=None)), # Random state is set to None to generate
             ('preprocessor', preprocessor),
             ('classifier', LogisticRegression(max_iter=1000)) ])

         # Performing cross-validation for different scorings
         logregrus_cv_f1 = cross_val_score(logreg_pipeline_rus_imb, X_train, y_train, cv=5, scoring='f1')
         logregrus_cv_recall = cross_val_score(logreg_pipeline_rus_imb, X_train, y_train, cv=5, scoring='recall'
         logregrus_cv_precision = cross_val_score(logreg_pipeline_rus_imb, X_train, y_train, cv=5, scoring='prec

         print(logregrus_cv_f1)
         print(logregrus_cv_recall)
         print(logregrus_cv_precision)

         [0.29257362 0.29633317 0.30166881 0.29919974 0.30070831]
         [0.66348974 0.65249267 0.69258988 0.66691123 0.67375367]
         [0.18609407 0.19171095 0.19428454 0.19128431 0.19532554]

In [50]: print(f'f1 mean:',logregrus_cv_f1.mean())
         print(f'Recall mean:',logregrus_cv_recall.mean())
         print(f'Precision mean:', logregrus_cv_precision.mean())

         f1 mean: 0.2980967284656636
         Recall mean: 0.6698474341789609
         Precision mean: 0.19173988098800068
```

Appendix 8 - Random Forest Tuning Example



```
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_features': ['auto', 'sqrt'],
    'max_depth': [10, 20, 30],
}

grid_search_rf = GridSearchCV(estimator=RandomForestClassifier(random_state=42),
                              param_grid=param_grid, cv=5, scoring='recall')

grid_search_rf.fit(X_train, y_train)
grid_search_rf.best_params_

print("Best Model Parameters:", grid_search_rf.best_params_)
```

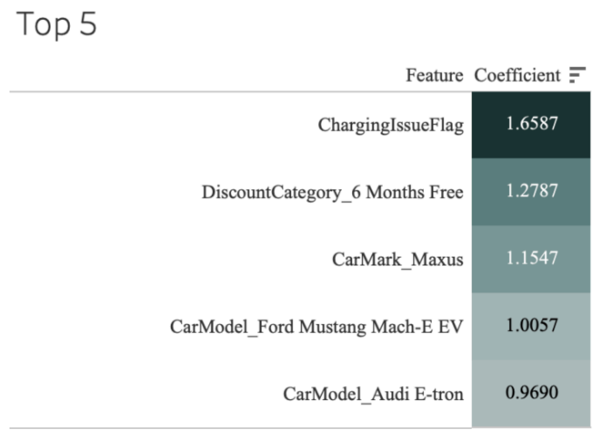
Appendix 9 – Random Forest RUS Confusion Matrix



		Predicted: Not Churned	Predicted: Churned
Actual	Not Churned	11,422 63.37%	4,896 27.16%
	Churned	631 3.50%	1,075 5.96%

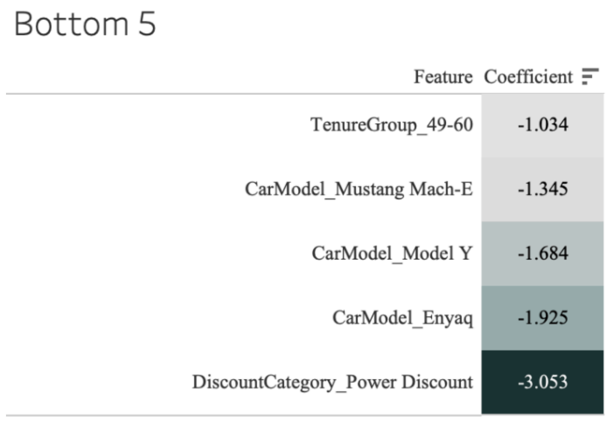
Count
631
11,422

Appendix 10 – Logistic Regression Coefficient Report



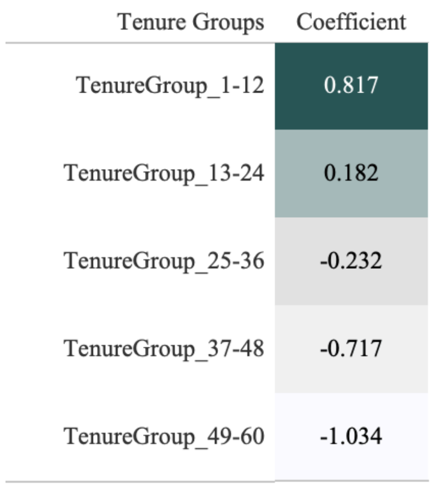
Top 5

Feature	Coefficient
ChargingIssueFlag	1.6587
DiscountCategory_6 Months Free	1.2787
CarMark_Maxus	1.1547
CarModel_Ford Mustang Mach-E EV	1.0057
CarModel_Audi E-tron	0.9690



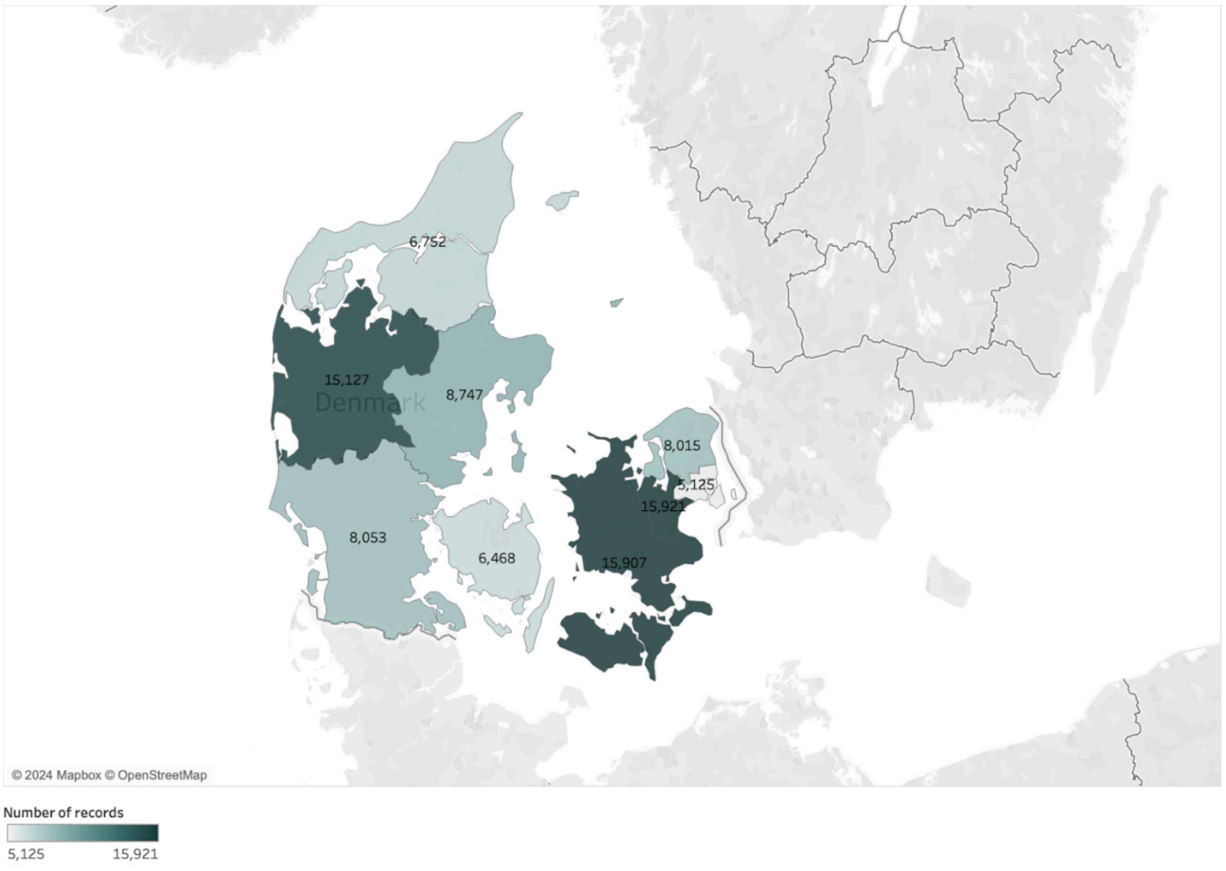
Bottom 5

Feature	Coefficient
TenureGroup_49-60	-1.034
CarModel_Mustang Mach-E	-1.345
CarModel_Model Y	-1.684
CarModel_Enyaq	-1.925
DiscountCategory_Power Discount	-3.053



Tenure Groups	Coefficient
TenureGroup_1-12	0.817
TenureGroup_13-24	0.182
TenureGroup_25-36	-0.232
TenureGroup_37-48	-0.717
TenureGroup_49-60	-1.034

Appendix 11 – Total Subscriptions per region


6,752
15,127
8,747
Denmark
8,015
5,125
15,921
8,053
6,468
15,907
© 2024 Mapbox © OpenStreetMap
Number of records
5,125
15,921