Lapse Drivers

MODEL Design document

Janco Van Niekerk

2024

Contents

[1. Introduction 2](#_Toc177490593)

[2. Process Overview 2](#_Toc177490594)

[2.1. Data 3](#_Toc177490595)

[2.2. Packages 4](#_Toc177490596)

[3. Training and Inference Pipelines 4](#_Toc177490597)

[3.1. Validation Strategy 5](#_Toc177490598)

[3.2. Training Pipeline 7](#_Toc177490599)

[3.3. Production Pipeline 8](#_Toc177490600)

[4. Machine Learning Pipelines 9](#_Toc177490601)

[4.1. Preprocessing in ML-Pipelines 10](#_Toc177490602)

[4.2. Model-Pipeline 1 11](#_Toc177490603)

[4.3. Model-Pipeline 2 11](#_Toc177490604)

[4.4. Model-Pipeline 3 12](#_Toc177490605)

[5. Feature Effect and Importance Investigation 13](#_Toc177490606)

[6. Causal Analysis 14](#_Toc177490607)

[6.1. Net Risk Reduction 15](#_Toc177490608)

[6.2. Scenario 1: Observed confounding 16](#_Toc177490609)

[6.3. Scenario 2: Non-confounding redundancies 16](#_Toc177490610)

[6.4. Other Scenarios 17](#_Toc177490611)

[7. Conclusion 17](#_Toc177490612)

List of Figures:

[Figure 1 - Entity Relationship Diagram 3](#_Toc177490535)

[Figure 2 - Validation Strategy 6](#_Toc177490536)

[Figure 3 - Development Training Pipeline 7](#_Toc177490537)

[Figure 4 - Production Training Pipeline 8](#_Toc177490538)

[Figure 5 - Random Forest Classifier 11](#_Toc177490539)

[Figure 6 - Regularized Logistic Regression 12](#_Toc177490540)

[Figure 7 - Gradient Boosting Classifier 12](#_Toc177490541)

[Figure 8 – Example Mean Absolute Shap Values 13](#_Toc177490542)

[Figure 9 - Example Beeswarm Plot 13](#_Toc177490543)

[Figure 10 - Example Shap Scatter plot 14](#_Toc177490544)

[Figure 11 - Control for Confounders 16](#_Toc177490545)

[Figure 12 - Handling Non-confounding Redundancies 17](#_Toc177490546)

# Introduction

This project aims to develop a predictive model and data pipeline to identify the drivers of customer lapses, which will inform retention campaigns. The model and pipeline are designed to handle unspecified data fields and sparse survey data.

This document outlines the process for training, validating, and deploying the required models and outputs. The primary objective is to uncover and analyse the drivers of customer lapses. To achieve this, we will use interpretability tools, such as SHAP, to reveal key relationships between input features and predicted outcomes. However, it's important to note that these tools highlight correlations rather than causations. As discussed in the Microsoft notebook [here](https://github.com/shap/shap/blob/master/notebooks/overviews/Be%20careful%20when%20interpreting%20predictive%20models%20in%20search%20of%20causal%20insights.ipynb), correlations identified by these techniques do not imply causation.

To accurately assess the impact of business changes on specific variables, we must also develop causal models. This requires making certain assumptions and applying tools from causal analysis. Therefore, our approach will involve designing and implementing multiple models, each of which will be detailed in this document.

# Process Overview

To thoroughly investigate the drivers of customer lapses and meet the project requirements, we will focus on producing the following outputs:

* **Pipeline for Model Training:** An approach to training our predictive models.
* **Pipeline for Model Deployment:** A streamlined process for deploying the models into a production environment.
* **Shapley Values:** To understand the effects and importance of features from a correlation perspective.
* **Causal Analysis:** Evaluation of how changes in feature values impact lapse probabilities, which will require the development of causal models.

## Data

To achieve these outputs, we need to process and transform the available data. The data we will use includes:

• Customer survey data with sparse entries, saved to customer\_survey.csv. Survey data fields include:

* POL\_NUMBER (string)
* MONTH \_KEY (datetime, simplified to the first of the month)
* HOW\_LIKELY\_ARE\_YOU\_TO\_RECOMMEND\_THE\_PRODUCT (int64, range:1-5)
* GENERAL\_FEEDBACK (string)

• Lapse data, saved to lapse.csv. Lapse data fields include:

* POL\_NUMBER (string)
* MONTH \_KEY (datetime, simplified to the first of the month, ranging from 01 January 2000 to 01 August 2024)
* AGE (int64)
* DURATION (int64, number of months that the policy has been in-force, range 1-563)
* GENDER (“M” or “F”)
* LAPSE\_IN\_12M (bool, 1 indicates that the policy has lapsed within 12 months from the MONTH\_KEY)
* 10-100 unspecified fields, including numeric and categorical columns

We assume that there are no features contributing to leakage (i.e., no feature values that would not be available at prediction time). We also assume a one-to-one relationship between customer survey data and lapse data, linked by POL\_NUMBER, and that the MONTH\_KEY is consistent across both datasets, representing the month and year when the survey was completed (Figure 1).

A diagram of a survey

Description automatically generated

Figure - Entity Relationship Diagram

In cases where multiple surveys exist per lapse entry, we could engineer time-dependent features. For instance:

* **Average Response:** The average response to the question "HOW LIKELY ARE YOU TO RECOMMEND THIS PRODUCT" over the past two years for each lapse entry.

These advanced techniques are not covered in this example but may be considered in future iterations if it is established that there is a many-to-one relationship between the tables.

## Packages

The following python packages will be required:

* numpy
* pandas
* shap
* matplotlib
* econml
* sklearn
* joblib
* dateutil

# Training and Inference Pipelines

The steps described below should be catered to the training and deployment pipeline respectively. The differences in these two pipelines are discussed under heading 3.2 and 3.3.

In this document, the term ‘model-pipeline’ refers to the specific combination of preprocessing techniques (e.g., one-hot encoding) and machine learning algorithms (e.g., Logistic Regression). Preprocessing steps must be applied in accordance with the validation strategy to prevent leakage. Therefore, these steps should be performed during the training and hyperparameter selection phases, rather than being applied in advance to the entire dataset. To facilitate this process, the combination of preprocessing techniques and machine learning algorithms is organized into an ‘ml-pipeline,’ streamlining the training workflow.

This project is divided into several key steps to ensure a robust and effective model training, validation, selection, and deployment process:

1. **Merge Datasets and Preprocessing:**
   * **Step 1 (Pipeline\_1):** Merge the customer survey and lapse datasets and remove non-informative ID columns. To investigate all potential drivers, we will not eliminate other non-informative features in the data or address multicollinearity by means of feature reduction.
   * **Step 2:** Auto-detect column types (and manually adjust the assigned types if required) to ensure accurate data representation for downstream processing.
   * **Step 3 (Pipeline\_2):** Perform preprocessing based on column types. This step is done to avoid reapplication of transformations when algorithms are compared. For this reason, transformations which can introduce leakage should not be included in this pipeline. A suitable transformation could include creating additional columns like Date\_Year and Date\_Month.
2. **Training, Model Selection and Validation:**
   * **Step 4:** Apply preprocessing techniques specific to the algorithm being used (e.g., scaling, imputation, one-hot encoding) and train the ML-algorithm (forming the model pipeline). This step ensures preprocessing is applied only to training data to avoid leakage. Optimize hyperparameters using cross-validation on sliding windows (Figure 2). The sliding window technique is employed to address the dynamic nature of the business environment. And therefore, ensures that our models are optimized to remain relevant and up-to-date despite changes over time.
   * **Step 8:** Train a new model pipeline on the training data of the validation partition and evaluate it on the respective validation set for this partition. Compare model pipelines based on their performance on this set.
   * **Step 9:** For final unbiased performance evaluation, train the best model pipeline on the holdout partition’s training data and assess it on the holdout partition’s validation set. Note that we cannot use the validation set from the previous step to obtain the final unbiased estimate of the model’s performance, as it was used to select the best-performing model pipeline. This previous selection process itself is an optimization step that needs to be independently evaluated.
3. **Model Updates:**
   * **Step 10:** Once it has been confirmed that the model pipeline performs well on the holdout set, we can choose the best-performing model, retrain it using the latest six months of data to incorporate recent changes in the business environment.

## Validation Strategy

We will evaluate various possibilities, including:

* Algorithm type and preprocessing steps
* Hyperparameter settings

A robust validation strategy is crucial to avoid overfitting, especially with many features, model-pipelines and hyperparameter variations.

Given the dynamic nature of business environments, relationships between variables may change over time. Thus, our approach will account for this effect, recognizing that a model trained on data spanning 10 years may not always be beneficial. Thus, our model training, selection and validation will focus on the most recent available data.

To address this time-dependence, we will use a sliding window technique for hyperparameter tuning and algorithm validation. This method divides the data into several partitions, each with its own training and validation sets. The algorithm is trained on the training set of each partition and evaluated on the corresponding validation set (see Figure 2).

A close up of a document

Description automatically generated with medium confidence

Figure - Validation Strategy

Below is a list of the partitions and their respective roles.:

* 3 Cross-Validation Partitions:
  + Used for selecting the best hyperparameters.
* Validation Partition:
  + Used to compare the performance of various algorithms. This partition ensures a fair comparison, as the mean score from the cross-validation partitions alone isn't sufficient since each algorithm has already ‘seen’ the validation sets within those partitions.
* Holdout Partition:
  + Provides a final unbiased evaluation of algorithm performance. The validation partition cannot be used for this purpose, as the 'best' algorithm has been selected based on its performance on the validation partition itself.

In practice, the validation set represents the duration a model goes without training on the latest data. Our model will train on a dataset and then predict on a validation set, simulating the scenario of retraining the model once every 6 months (as advised by business domain experts).

To align with business changes, such as customer behaviour changing every six months, each validation set should cover six months. To determine the training set size, which should be approximately 70% of the data compared to the validation set, we divide 6 months by 0.3.

The metric used to choose the best hyperparameters will be log loss. Using this metric, we can estimate how well the model predicts **probabilities**, which will be essential for investigating lapse drivers. We're thus not concerned with classification accuracy, which is the default metric for many classifiers in scikit-learn.

We will validate each model-pipeline, using the optimal hyperparameters identified during cross-validation, by calculating log loss on the validation set of the validation partition.

This step will allow us to proceed to the next optimization step which is choosing the best performing algorithm.

## Training Pipeline

One key difference between the data transformations in the training and production pipelines is that the training pipeline must create various partitions for the validation strategy to identify the best model pipelines and hyperparameter settings. Sorting the data by date is crucial for applying the sliding window technique, which is an essential component of the validation strategy. Figure 3 illustrates the data flow and model training steps involved in training and selecting the best model pipeline.

A diagram of a process

Description automatically generated

Figure - Development Training Pipeline

## Production Pipeline

The production pipeline is used for inference and employs the best-performing model-pipeline identified during the training, selection, and validation phases. It can also be utilized to retrain this model pipeline on the latest data as needed. According to business stakeholders, customer behaviour changes significantly every six months, so the model should be retrained at least once every six months.

Additionally, the production pipeline allows developers to adjust components, such as column types or ID column specifications, in response to changes in the database. The retraining process must filter the source data accurately to ensure the model is trained on the correct training window length (which is the six-month validation window divided by 0.3 to determine the appropriate training window length).

Date sorting is no longer necessary—there is only one partition, and time-based splitting for validation is not required. Additionally, we need predictions returned in the same order as the input data to correctly match predictions to entries. Sorting by date, as in the original pipeline\_1, could misalign the rows and predictions. For this reason, ‘pipeline\_1’ is modified for production use, as illustrated in Figure 4.

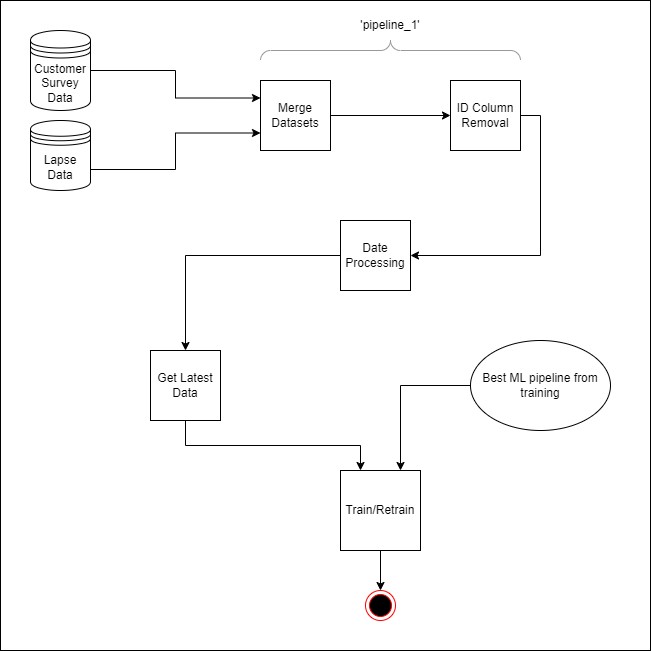


Figure - Production Training Pipeline

# Machine Learning Pipelines

As mentioned, in this document, the term ‘model-pipeline’ refers to the specific combination of preprocessing techniques (e.g., one-hot encoding) and machine learning algorithms (e.g., Logistic Regression).

Various machine-learning algorithms were tested along with common hyperparameter settings for each:

**Regularized Logistic Regression:**

Chosen for its simplicity and efficiency in handling linear relationships between features and the target variable. Regularization helps prevent overfitting and handles high-dimensional data well.

Hyperparameters Evaluated:

* Regularization Strength: [0.1, 1.0, 10.0]

**Random Forest Classifier:**

Selected for its robustness against overfitting and ability to capture complex interactions between features through ensemble learning.

Hyperparameters Evaluated:

* n\_estimators': [400, 300, 100]
* min\_samples\_leaf': [1, 2, 4]

**Gradient Boosting Classifier:**

Used for its high predictive accuracy and flexibility in modelling complex relationships.

Hyperparameters Evaluated:

* n\_estimators': [100, 300, 400]
* min\_samples\_leaf': [1, 2, 4]

## Preprocessing in ML-Pipelines

These machine learning algorithms are evaluated in conjunction with their respective preprocessing techniques according to the validation strategy discussed.

**NOTES ON TEXT-ENCODING**

Text features will be encoded using an n-gram technique, including both uni-grams and bi-grams.

**NOTES ON FEATURE SCALING:**

One of the chosen algorithms will be regularized logistic regression. Due to the potentially large number of features, we will also add L2 regularization to this algorithm.

Tree models are not sensitive to feature scaling and therefore we won’t be applying feature scaling to the numeric values of any of the tree-based algorithms. However, for our regularized logistic regression algorithm we will apply feature scaling due to it being sensitive to very large feature values.

Because we are unaware of the precise feature distribution, we will not be using z-score normalization but rather use maximum absolute feature scaling which has a range of [-1, 1]. This works well with our approach for some of the preprocessing steps applied (such as one-hot encoding) due to them already having a value of either 0 or 1. We will thus also scale the values received from our n-gram technique for the regularized logistic regression algorithm.

**NOTES ON CATEGORICAL ENCODING AND IMPUTATION:**

All one-hot encoders will have a max\_categories of 20 to prevent an excessive number of new features.

Missing categorical values will be assigned a 'Missing' class, creating a new category for missing values.

We will use handle\_unknown='infrequent\_if\_exist' to create an 'infrequent' category for values outside the 20 most common categories. This also handles unknown categories at prediction time by ensuring that the corresponding one-hot encoded columns will be all zeros.

**NOTES ON NUMERICAL IMPUTATION**

Numerical missing values will be imputed with the mean. Additionally, a missing indicator for each column will be added using SimpleImputer(add\_indicator=True), where a value of 1 indicates missing data. This will allow us to examine the relationship between missing values and lapse probabilities.

## Model-Pipeline 1

The first machine learning pipeline (Figure 5) can be summarized as follows:  
  
**Machine learning Algorithm**:

* Random Forest Classifier

**Preprocessing Techniques**:

* n-gram encoding for text features
* 'Constant' value imputation for missing categorical features
* One-hot encoding for categorical features
* Mean imputation for numerical features with a missing indicator

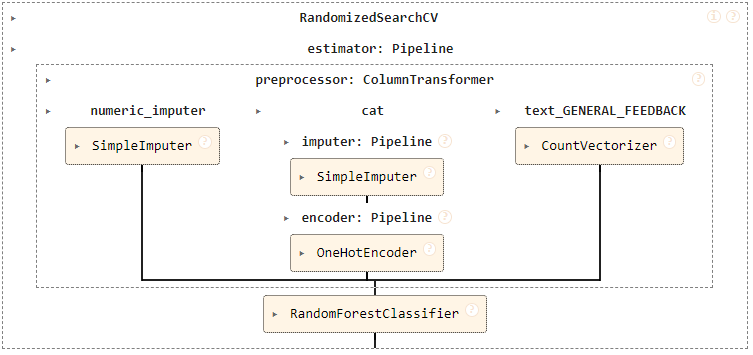


Figure - Random Forest Classifier

## Model-Pipeline 2

The second machine learning pipeline (Figure 6) can be summarized as follows:

**Machine learning Algorithm**:

* Regularized Logistic Regression

**Preprocessing Techniques**:

* n-gram encoding for text features
* MaxAbs scaling for numerical features (also scales n-gram text encodings)
* One-hot encoding for categorical features
* Mean imputation for numerical features with a missing indicator

A screenshot of a computer

Description automatically generated

Figure - Regularized Logistic Regression

## Model-Pipeline 3

The third machine learning pipeline (Figure 7) can be summarized as follows:

**Machine learning Algorithm**:

* Gradient Boosting Classifier

**Preprocessing Techniques**:

* n-gram encoding for text features
* 'Constant' value imputation for missing categorical features
* One-hot encoding for categorical features
* Mean imputation for numerical features with a missing indicator

A screenshot of a computer

Description automatically generated

Figure - Gradient Boosting Classifier

# Feature Effect and Importance Investigation

With our best model-pipeline trained on the latest data, we can calculate Shapley values. These values can be utilized in various ways, including:

* **Assessing Feature Importance:** By examining the mean absolute SHAP values of each feature. (Example Figure 8)
* **Creating Beeswarm Plots:** To visualize the effect (correlation) of each feature on lapse probabilities for both high and low feature values. (Figure 9)
* **Generating Scatter Plots:** To explore how different values of a feature correlate with lapse probability. (Figure 10)

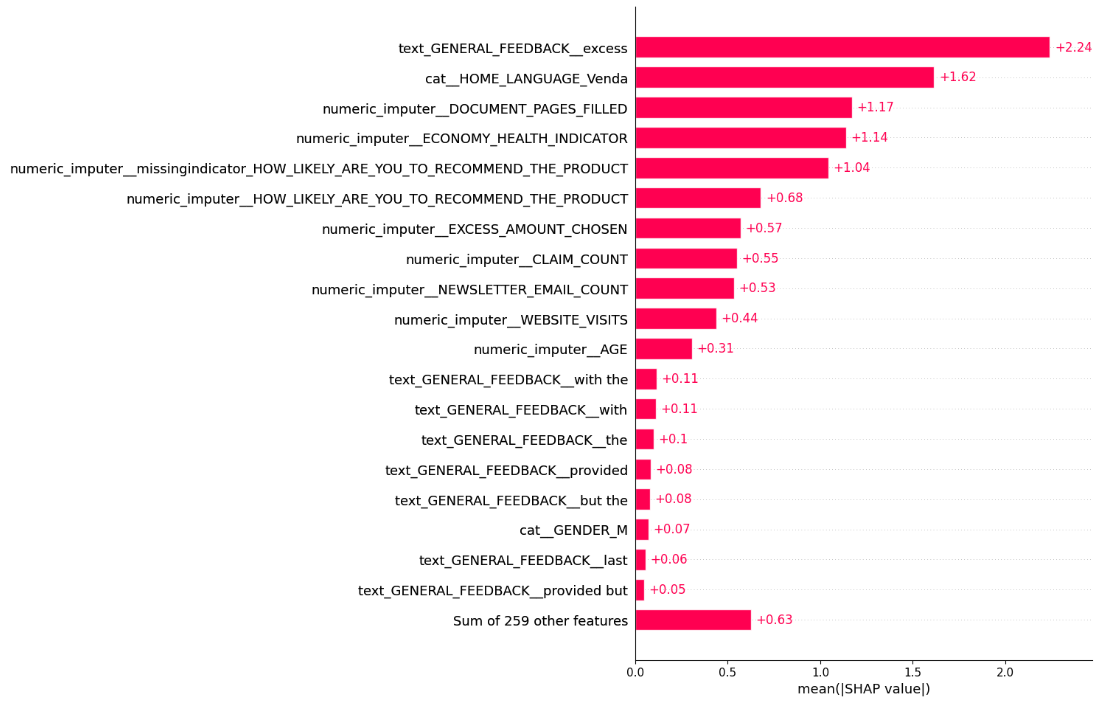


Figure – Example Mean Absolute Shap Values

A graph with red and blue lines

Description automatically generated

Figure - Example Beeswarm Plot

A graph with blue dots

Description automatically generated

Figure - Example Shap Scatter plot

It's crucial to consider the presence of missing values when analysing feature importances, and stakeholders should be informed of this. Due to mean imputation, some SHAP values may not perfectly represent the correlation with lapse probability. Ideally, non-imputed values should be analysed separately, but this will be beyond the scope of this exercise.

Additionally, some processed feature names may not be user-friendly for business stakeholders, as they include technical preprocessing terms. We will export the SHAP values for later use, allowing them to be imported into a report or application where feature names can be adjusted to more familiar terms.

# Causal Analysis

In real-world datasets, features are often not independent and unconfounded, so standard predictive models may not capture true causal effects. Consequently, SHAP explanations may not reveal causality. However, observational causal inference tools can sometimes address or minimize this issue.

At this point, it is crucial to engage with business stakeholders to:

* Draw a causal graph using SHAP value plots as a starting point to guide discussions.
* Determine with stakeholders which variables should be analysed to inform a potential retention campaign strategy and explore their causal impact on lapse probability.

When exploring the true causal impact of changing a variable we account for the following cases:

* Variables with observed confounders
* Variables with non-confounding redundancies

Since we ideally need experimental data to test our causal models, we will use the best-performing model pipeline and assume it will also work well for building causal models.

## Net Risk Reduction

When using causal inference techniques (such as the techniques discussed in the next section), simply stating the change in lapse probability may not give stakeholders full understanding of the business outcomes associated with changing the treatment value. It is for this reason that we need to investigate a more informative approach.

Changing the value of the treatment may have a cost attached to it. Therefore, it will be useful to know what the gain of changing the treatment value is so that we can compare it with the cost and make the business decision on whether it is worth a change (i.e. changing the treatment value) being implemented.

The change in risk for a single customer subjected to a different treatment can be defined as follows:

Risk\_Change = Lapse\_impact\*△Lapse\_probability

Where:

* Lapse\_impact: the cost associated with losing the customer i.e. the revenue which you would not receive from them for the next 12 months.
* △Lapse\_probability: Change in the probability of lapse if subjected to the new treatment

Therefore, using the △Lapse\_probability of each customer, the total risk change for all customers in the sample can be calculated as:

Total\_risk\_change = (Lapse\_impact \* △Lapse\_probability\_1) + (Lapse\_impact \* △Lapse\_probability\_2) + ..... + (Lapse\_impact\*△Lapse\_probability\_n)

If we assume that the lapse impact is the same for all customers, we get:

Total\_risk\_change = Lapse\_impact\*(Sum\_of\_all\_△Lapse\_probabilities)

To get the total reduction in risk, we do the following:

Total\_risk\_reduction = (-1\*Total\_risk\_change)

As mentioned, it should be determined if the cost associated with implementing a change is worth it. This can be determined by calculating the 'Net of risk reduction':

Net\_risk\_reduction = Total\_risk\_reduction - Implementation\_Cost

We will calculate the causal effect of changing certain values on net risk reduction by using the techniques discussed in the following sections.

Depending on business needs, we will evaluate the effect of changing variables in two main ways:

1. Change all values of the treatment to the **same uniform** value: As an example, lets propose that certain stakeholders mentioned that they think it is possible to change the paperwork process so that each of their customers only ever have to fill out documentation once and that they believe these documents can be the same amount of pages for everyone.
2. Increase/decrease the current treatment values by a specific amount: Other stakeholders believe that it will not be possible to create a single and similar paperwork process for all their customers. They argue that certain customers are unique and therefore require additional document pages to be filled. They are convinced however, that they can reduce everyone’s document pages by 3 pages ex. if a customer usually fills 10 pages, they will now only fill 7.

## Scenario 1: Observed confounding

If we want to investigate the effect of changing a variable with observed confounders, we will use a technique called 'double machine-learning'. It uses machine learning to deconfound the treatment variable and then estimates how much the outcome variable will change when we change the treatment variable by a certain amount.

It is important that we control for all possible confounders when using double machine learning. (Figure 11 serves as an example).

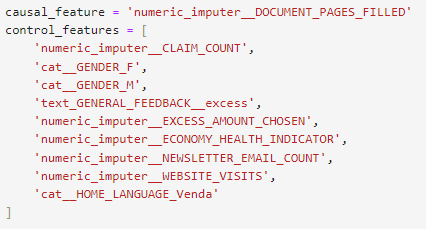


Figure - Control for Confounders

## Scenario 2: Non-confounding redundancies

In some cases, we may be interested in the effect of changing a variable which is not confounded by any other variables. However, if it is known (from the causal graph) that there is downstream effect of another variable on lapse probability we deal with a case where there is an indirect effect of this downstream variable on lapse probability.

In such cases, there’s a 'non-confounding redundancy,' and double-ML will only provide the direct effect of changing the treatment variable, excluding any indirect effects. To capture the 'full effect,' we can remove the downstream variable from our model and train it using only the treatment variable.

An example of this can be seen in Figure 12. In this example newsletter email count was assumed not to be confounded by other variables according to the causal graph. However, it was hypothesized to influence lapse probability as well as website visits. Website visits, in turn, were thought to affect lapse probability, creating an 'indirect effect.' Therefore, a model was built on training data that only includes newsletter email count as a feature.

A screenshot of a computer

Description automatically generated

Figure - Handling Non-confounding Redundancies

## Other Scenarios

The project will address other scenarios related to causality. However, since these scenarios do not involve building models, they are not covered in this document.

# Conclusion

This document outlines the design and process for developing a predictive model and pipeline to understand customer lapses and inform retention strategies. It covers the steps involved in data preparation, model training, and evaluation, including handling data transformations, choosing machine learning algorithms, and applying validation strategies.

By leveraging Shapley values, we aim to gain insights into feature importance and the impact of various factors on lapse probability. We have covered how to build ML models which can be used for causal inference in a variety of scenarios.