France Policy Change Proposal
PPLE Bank Credit Risk Committee- April 2025

Justo Ruiz Cano, Victoria Robert, Javier Alonso, Jeppe Falch, Dan Hasson

THE BACKGROUND

Situation: Revolut is not giving enough loans after policy change.

Implemented Policy July 2024:

- Age to Use: 25 or more (previously 18 or more)
- Days as customer: 90 or more (previously 30 or more)
- Minimum balance 30 days: 10 or more (previously 0 or more)
- Gambling transactions: Always 0 (20 or less before)
- ATM withdrawals: Less than 100 (less than 1,000 before)
- Other credit lines: Always FALSE (TRUE or FALSE)

Proposal: Investigate a new way of predicting the defaults through non-linear models as there may be a deeper, more complex interplay between default and variables.

Next Steps

Goal: Create models which can successfully classify defaults and then see how we can "tweak" them to ensure our main objective: more loans given out.

Models

- Decision Trees
- XGBoost
- Logistic Regression

Methodology

Data Preparation

Model Creation

Model Evaluation

Final Decision

Data Preparation

Data Preparation

Step 1: Cleaning and categorizing the data

- Original dataset had SP and FR Loans —> We're only interested in France
- We only want to use **loans which have had the chance to default**
    - We filter by **MOB >= 3** (ensures they've all had a chance to Default after 30 days 3 months on book)
- Multiple entries of the same loan and how it evolves in time! —> Groupby id and select.
    - Loss of information

Data Preparation

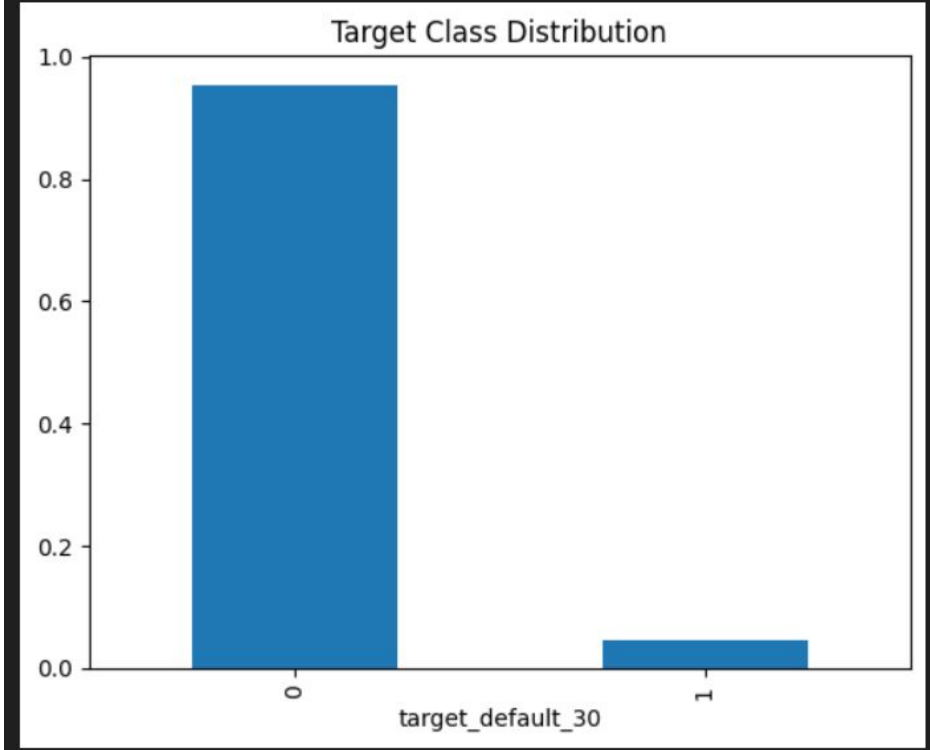
Step 2: Basic EDA

0 = Non-Default

1 = Default

- As you can see, huge class imbalance.
- This is a problem, a big one
    - Leads to overfitting
    - Evaluation metrics are not as straightforward to understand

```
target_default_30
0    1369
1      66
Name: count, dtype: int64
```



Dealing with unbalanced Data

Depending on your use case, there are a variety of ways to deal with unbalanced data.

- Oversampling
- Undersampling
- SMOTE: Synthetic Minority Oversampling Technique

However, remember use case: Loan defaults —> this distribution is natural and we want our model to be trained in similar conditions it will find in production.

Data Preparation

Step 3: Split into train and test split and encode categorical variables.

- Stratify
- Models are not, usually, able to take raw text, such as: **Declared_accommodation_type**
    - Since it is **categorical** we can use **one-hot encoding**

Declared_accommodation_type
Renting
Homeowner with Mortgage
Living with Parents
Homeowner without Mortgage

Renting	Homeowner with Mortgage	Living with Parents	Homeowner without Mortgage
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

Model Building

Yay :D

Model Building: Decision Trees

Context

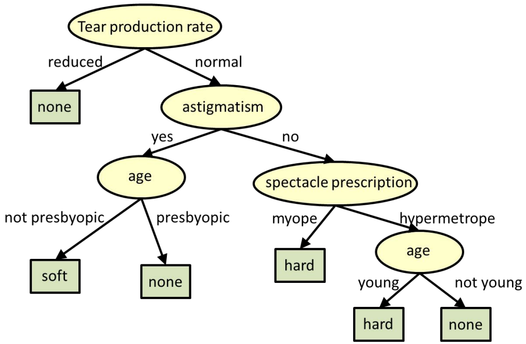
Decision tree is an algorithm which splits data based on what **best divides the data into 2**

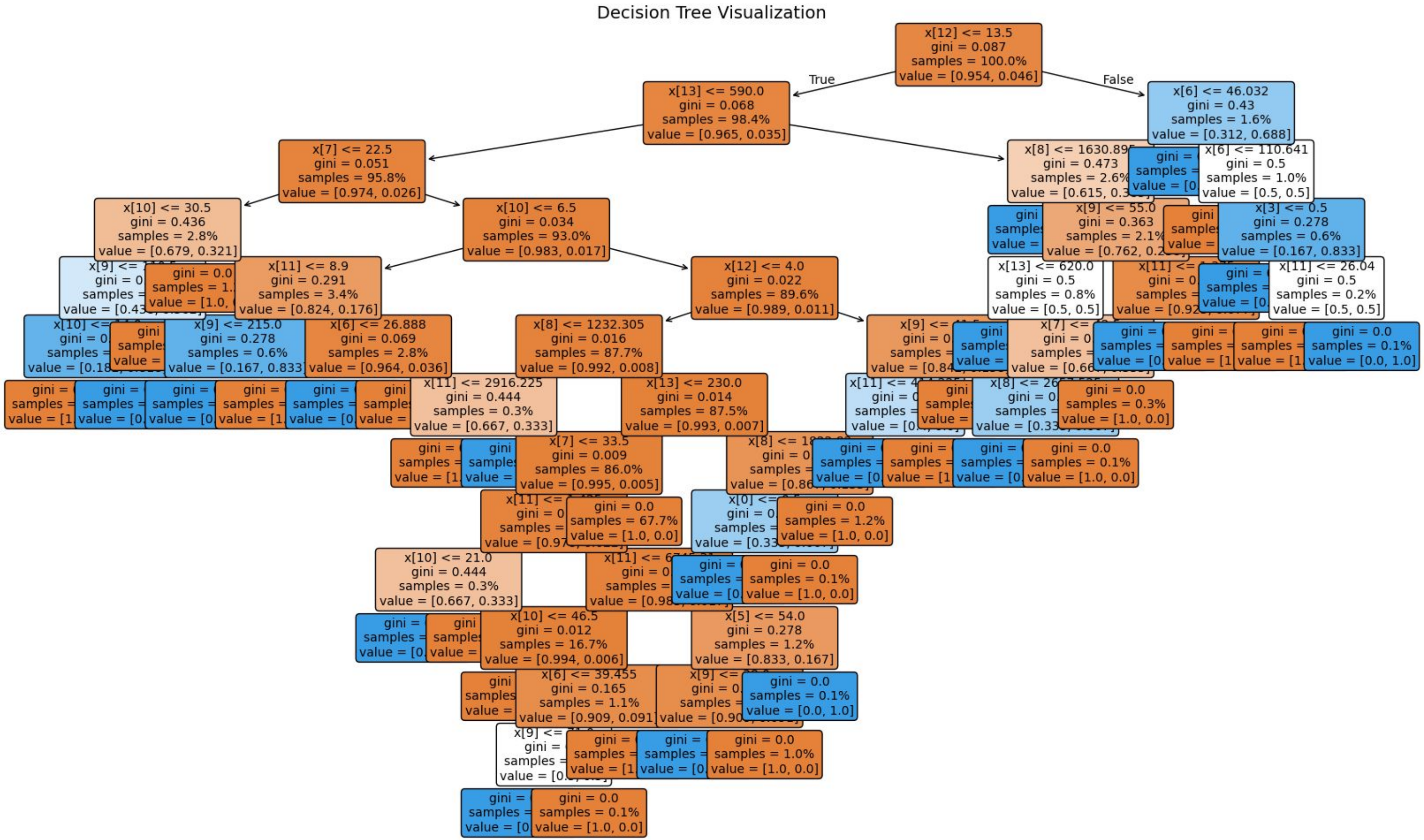
- For classification, this is the **GINI index**

$$Gini = 1 - \sum_{i=1}^{k} p_i^2$$

P = probability of choosing an element from class i in that node.
K = number of classes

Gini aims to create the "purest" nodes = feature that produces the lowest gini is chosen




Decision Tree Visualization
x[12] <= 13.5
gini = 0.087
samples = 100.0%
value = [0.954, 0.046]
True
False
x[13] <= 590.0
gini = 0.068
samples = 98.4%
value = [0.965, 0.035]
x[6] <= 46.032
gini = 0.43
samples = 1.6%
value = [0.312, 0.688]
x[7] <= 22.5
gini = 0.051
samples = 95.8%
value = [0.974, 0.026]
x[6] <= 110.641
gini = 0.5
samples = 1.0%
value = [0.5, 0.5]
x[10] <= 30.5
gini = 0.436
samples = 2.8%
value = [0.679, 0.321]
x[10] <= 6.5
gini = 0.034
samples = 93.0%
value = [0.983, 0.017]
x[3] <= 0.5
gini = 0.278
samples = 0.6%
value = [0.167, 0.833]
x[11] <= 8.9
gini = 0.291
samples = 3.4%
value = [0.824, 0.176]
x[12] <= 4.0
gini = 0.022
samples = 89.6%
value = [0.989, 0.011]
x[13] <= 620.0
gini = 0.5
samples = 0.8%
value = [0.5, 0.5]
x[11] <= 26.04
gini = 0.5
samples = 0.2%
value = [0.5, 0.5]
x[9] <= 215.0
gini = 0.278
samples = 0.6%
value = [0.167, 0.833]
x[6] <= 26.888
gini = 0.069
samples = 2.8%
value = [0.964, 0.036]
x[8] <= 1232.305
gini = 0.016
samples = 87.7%
value = [0.992, 0.008]
gini = 0.0
samples = 0.1%
value = [0.0, 1.0]
x[11] <= 2916.225
gini = 0.444
samples = 0.3%
value = [0.667, 0.333]
x[13] <= 230.0
gini = 0.014
samples = 87.5%
value = [0.993, 0.007]
gini = 0.0
samples = 0.3%
value = [1.0, 0.0]
x[7] <= 33.5
gini = 0.009
samples = 86.0%
value = [0.995, 0.005]
gini = 0.0
samples = 0.1%
value = [1.0, 0.0]
gini = 0.0
samples = 67.7%
value = [1.0, 0.0]
gini = 0.0
samples = 1.2%
value = [1.0, 0.0]
x[10] <= 21.0
gini = 0.444
samples = 0.3%
value = [0.667, 0.333]
gini = 0.0
samples = 0.1%
value = [1.0, 0.0]
x[10] <= 46.5
gini = 0.012
samples = 16.7%
value = [0.994, 0.006]
x[5] <= 54.0
gini = 0.278
samples = 1.2%
value = [0.833, 0.167]
x[6] <= 39.455
gini = 0.165
samples = 1.1%
value = [0.909, 0.091]
gini = 0.0
samples = 0.1%
value = [0.0, 1.0]
gini = 0.0
samples = 1.0%
value = [1.0, 0.0]
gini = 0.0
samples = 0.1%
value = [1.0, 0.0]

Model Building: Decision Trees

As you can tell, Decision trees are **prone to overfitting.**

So, there are various **hyperparameters** we can tune to ensure generalizability.

The following are the most commonly used ones (and the ones I used for the model):

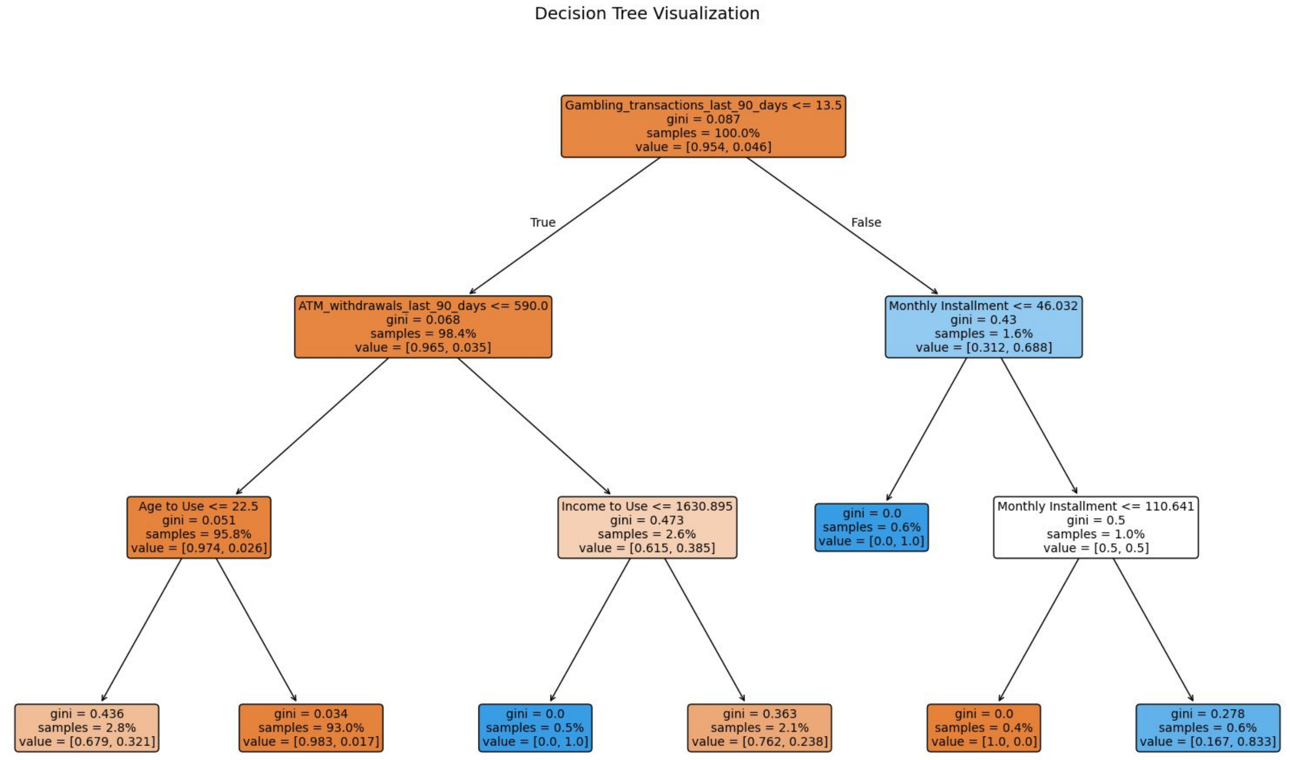
- Max_depth: Limits the depth of the tree
- Min_samples_split: minimum number of samples **in the node** to produce a split
- Min_samples_leaf: minimum number of samples to be a **leaf node**
- Max_features: amount of features to consider
- Criterion: Gini or entropy

Additionally, we can train the model to focus on **precision** instead of **accuracy**

Model Building: Decision Trees

```python
param_grid = {
    'max_depth': [3, 5, 7, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2', None],
    'criterion': ['gini']
}
```

Model Building: Decision Trees



Decision Trees: Metrics and further analysis

Classification Report:

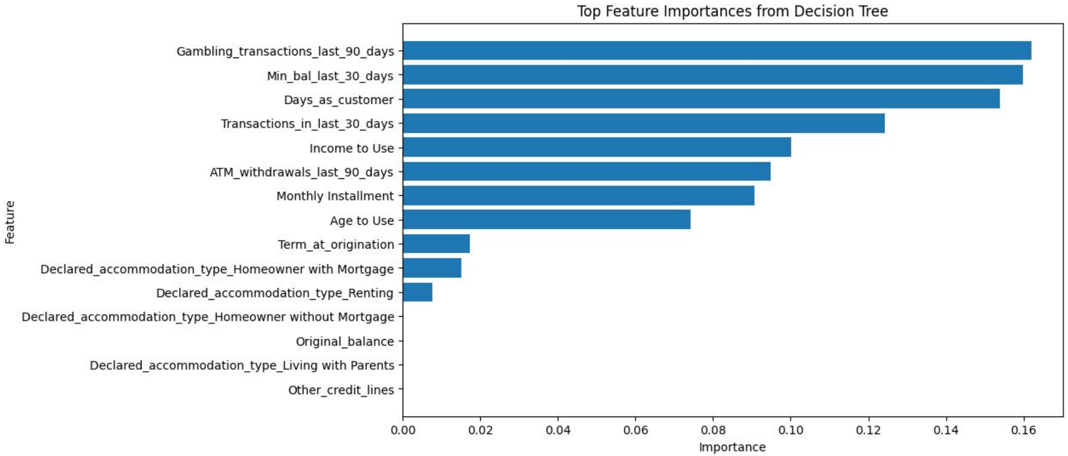
	precision	recall	f1-score	support
0	0.96	1.00	0.97	411
1	0.33	0.05	0.09	20
accuracy			0.95	431
macro avg	0.64	0.52	0.53	431
weighted avg	0.93	0.95	0.93	431

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

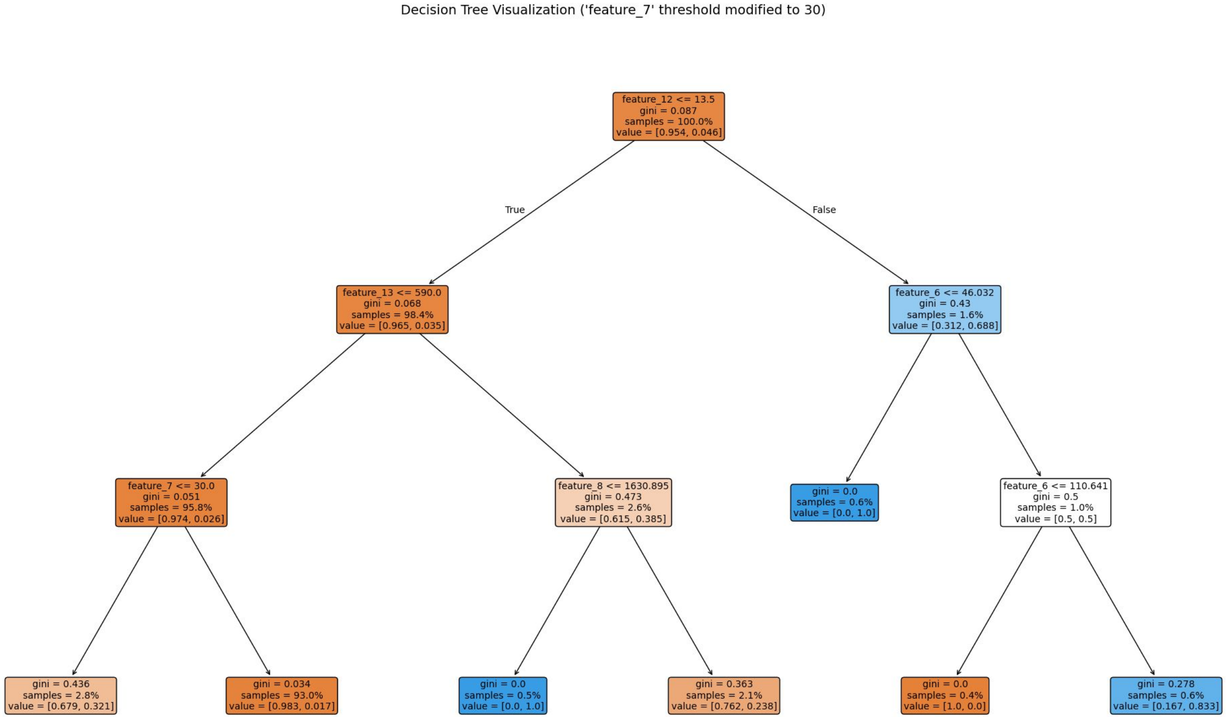
$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

Decision Trees: Metrics and further analysis

	Feature	Importance
12	Gambling_transactions_last_90_days	0.162055
11	Min_bal_last_30_days	0.159902
9	Days_as_customer	0.153929
10	Transactions_in_last_30_days	0.124246
8	Income to Use	0.100162
13	ATM_withdrawals_last_90_days	0.094933
6	Monthly Installment	0.090620
7	Age to Use	0.074109
5	Term_at_origination	0.017260
0	Declared_accommodation_type_Homeowner with Mor...	0.015189



Decision Trees: Metrics and further analysis



Model Building: XGBoost

Context

Insanely strong algorithm.

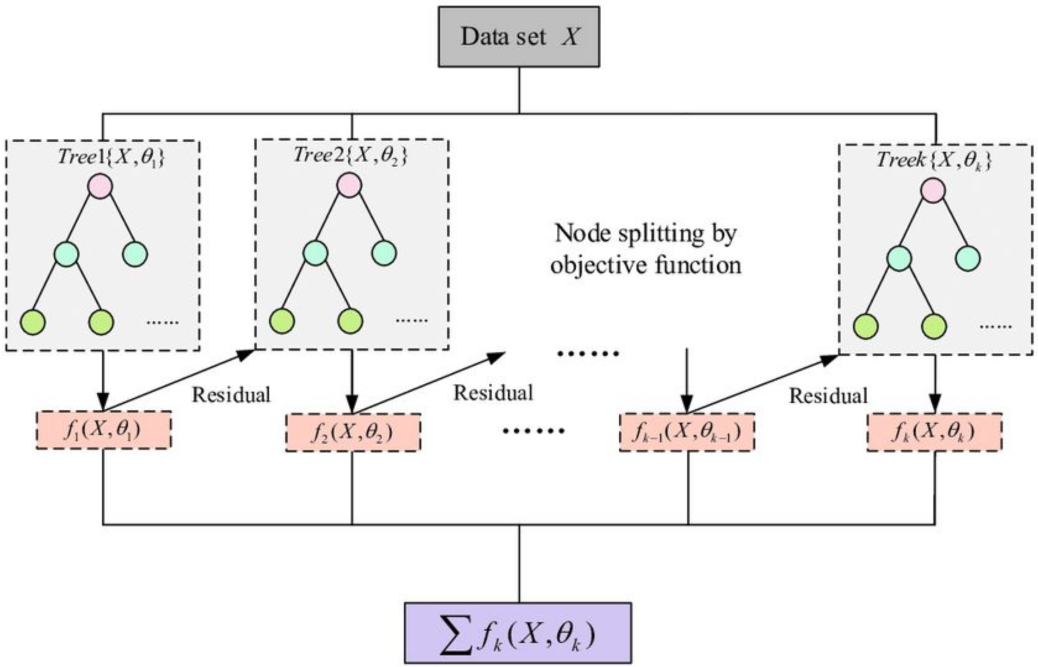
More computationally intensive.

Builds Tree

Computes residuals

Trees are improved and joined in additive manner

Uses Gradient descent ~ learning



Model Building: XGBoost

This is the model we focused most on.

We did our fine tuning in 2 stages:

1. Hyperparameter tuning
2. Threshold tuning

```
Evaluation at threshold = 0.15
Confusion Matrix:
 [[236  38]
 [  0  13]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.86      0.93       274
           1       0.25      1.00      0.41        13

    accuracy                           0.87       287
   macro avg       0.63      0.93      0.67       287
weighted avg       0.97      0.87      0.90       287
```

Model Building: XGBoost

<u>Hyperparameters</u>: XGBoost has a lot of important hyperparameters. Most notable ones you should know:

- **Objective**: tells the model we are classifying into binary and to output probabilities.
- **Gamma**: minimum reduction in loss function to create new split in tree
- **Leaning rate**: impact of each tree to the final prediction. Lower = more generalizable, but more intensive.
- **Reg lambda**: Ridge regularization. Creates penalties for larger weights.
- **Reg alpha**: Lasso regularization. Drives coefficients to 0 (feature selection)

```
classifier = XGBClassifier(
    scale_pos_weight=weight * 3,
    objective='binary:logistic',
    eval_metric='logloss',
    max_depth=5,
    min_child_weight=1,
    gamma=0.1,
    learning_rate=0.1,
    subsample=0.8,
    colsample_bytree=0.8,
    max_delta_step=5,
    reg_lambda=10.0,
    reg_alpha=10,
    random_state=42,
    use_label_encoder=False

)
```

Model Building: XGBoost

<u>Threshold</u>:

This model is creating a probability score.

- If threshold is = 0.5
  - X > 0.5 → Default
  - X < 0.5 → Non default

```
cost_fp = 500   # False positive: wrongly flagged as defaulter
cost_fn = 5000  # False negative: missed actual defaulter
```

```
✅ Best Threshold: 0.15 with Minimum Cost: $19,000
```

```
Evaluation at threshold = 0.15
Confusion Matrix:
 [[236  38]
 [  0  13]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.86      0.93       274
           1       0.25      1.00      0.41        13

    accuracy                           0.87       287
   macro avg       0.63      0.93      0.67       287
weighted avg       0.97      0.87      0.90       287
```

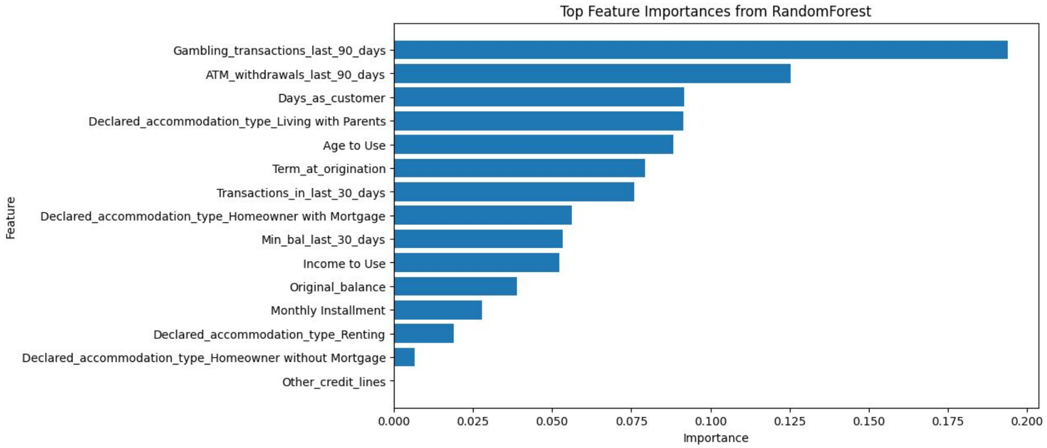
XGBoost: Metrics and further analysis

```
Evaluation at threshold = 0.15
Confusion Matrix:
 [[236  38]
 [  0  13]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.86      0.93       274
           1       0.25      1.00      0.41        13

    accuracy                           0.87       287
   macro avg       0.63      0.93      0.67       287
weighted avg       0.97      0.87      0.90       287
```

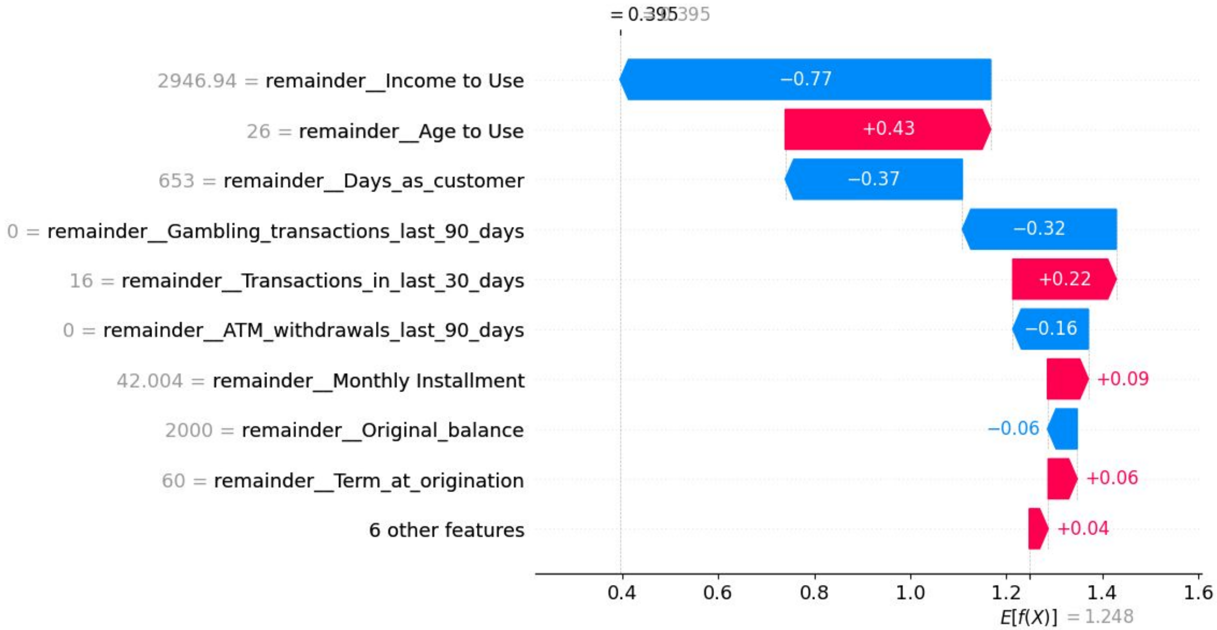
```
Evaluation at threshold = 0.8
Confusion Matrix:
 [[268   6]
 [  8   5]]

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.98      0.97       274
           1       0.45      0.38      0.42        13

    accuracy                           0.95       287
   macro avg       0.71      0.68      0.70       287
weighted avg       0.95      0.95      0.95       287
```

XGBoost: Metrics and further analysis

	Feature	Importance
12	Gambling_transactions_last_90_days	0.193953
13	ATM_withdrawals_last_90_days	0.125225
9	Days_as_customer	0.091703
2	Declared_accommodation_type_Living with Parents	0.091491
7	Age to Use	0.088373
5	Term_at_origination	0.079260
10	Transactions_in_last_30_days	0.076020
0	Declared_accommodation_type_Homeowner with Mor...	0.056344
11	Min_bal_last_30_days	0.053256
8	Income to Use	0.052293



XGBoost SHAP Visualisations



Model Building: Logistic Regression

Context

Logistic Regression is used for **binary classification**

- It predicts the **probability** that an instance is part of one group or not.

Calculates the **weighted sum** of independent features.

$$z = w_1x_1 + w_2x_2 + \ldots + w_nx_n + b$$

Output is passed through a **sigmoid function** to make the result between 0 - 1 where we then use the threshold to determine what group it belongs to.

The weights show the importance of each variable to the final prediction.

Logistic Regression: Metrics and further analysis

```
Confusion Matrix:
 [[333  78]
 [  0  20]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.81      0.90       411
           1       0.20      1.00      0.34        20

    accuracy                           0.82       431
   macro avg       0.60      0.91      0.62       431
weighted avg       0.96      0.82      0.87       431

ROC AUC: 0.9509732360097324
```

	Feature	Coefficient	Odds Ratio
12	num__Gambling_transactions_last_90_days	1.436692	4.206758
2	cat__Declared_accommodation_type_Living with P...	1.435664	4.202434
13	num__ATM_withdrawals_last_90_days	1.336708	3.806494
4	num__Original_balance	0.267234	1.306347
15	num__Reported_in_credit_bureau	0.000000	1.000000
11	num__Min_bal_last_30_days	-0.117192	0.889414
6	num__Monthly Installment	-0.211453	0.809408
5	num__Term_at_origination	-0.212265	0.808751
14	num__Other_credit_lines	-0.268680	0.764388
3	cat__Declared_accommodation_type_Renting	-0.283526	0.753124
8	num__Income to Use	-0.386160	0.679662
1	cat__Declared_accommodation_type_Homeowner wit...	-0.497660	0.607952
9	num__Days_as_customer	-0.507062	0.602263
7	num__Age to Use	-0.555621	0.573716
10	num__Transactions_in_last_30_days	-0.601978	0.547727
0	cat__Declared_accommodation_type_Homeowner wit...	-0.633948	0.530493

Logistic Regression: Metrics and further analysis

Slight overfitting.

TRAINING PERFORMANCE:

	precision	recall	f1-score	support
0	0.91	0.91	0.91	46
1	0.91	0.91	0.91	46
accuracy			0.91	92
macro avg	0.91	0.91	0.91	92
weighted avg	0.91	0.91	0.91	92

TEST PERFORMANCE:

	precision	recall	f1-score	support
0	1.00	0.81	0.90	411
1	0.20	1.00	0.34	20
accuracy			0.82	431
macro avg	0.60	0.91	0.62	431
weighted avg	0.96	0.82	0.87	431

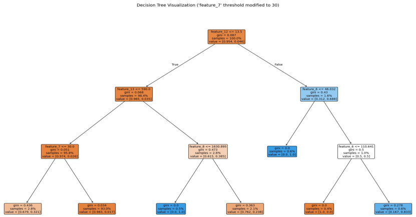
Train ROC AUC: 0.9546313799621928
Test ROC AUC: 0.9509732360097324

Okay, so what?

Decision Trees

XGBoost

Logistic Regression




```
classifier = XGBClassifier(
    scale_pos_weight=weight * 3,
    objective='binary:logistic',
    eval_metric='logloss',
    max_depth=5,
    min_child_weight=1,
    gamma=0.1,
    learning_rate=0.1,
    subsample=0.8,
    colsample_bytree=0.8,
    max_delta_step=5,
    reg_lambda=10.0,
    reg_alpha=10,
    random_state=42,
    use_label_encoder=False

)
```

	Feature	Coefficient	Odds Ratio
12	num__Gambling_transactions_last_90_days	1.436692	4.206758
2	cat__Declared_accommodation_type_Living with P...	1.435664	4.202434
13	num__ATM_withdrawals_last_90_days	1.336708	3.806494
4	num__Original_balance	0.267234	1.306347
15	num__Reported_in_credit_bureau	0.000000	1.000000
11	num__Min_bal_last_30_days	-0.117192	0.889414
6	num__Monthly Installment	-0.211453	0.809408
5	num__Term_at_origination	-0.212265	0.808751
14	num__Other_credit_lines	-0.268680	0.764388
3	cat__Declared_accommodation_type_Renting	-0.283526	0.753124
8	num__Income to Use	-0.386160	0.679662
1	cat__Declared_accommodation_type_Homeowner wit...	-0.497660	0.607952
9	num__Days_as_customer	-0.507062	0.602263
7	num__Age to Use	-0.555621	0.573716
10	num__Transactions_in_last_30_days	-0.601978	0.547727
0	cat__Declared_accommodation_type_Homeowner wit...	-0.633948	0.530493

The ask

We believe that using XGBoost as the core classifier will help balance risk and loans created.

This is because we can very quickly change the model's threshold to ensure we can correct the trajectory of our loan policies.

This model, whilst not fully interpretable like the logistic regression, does have very strong metrics and can be a) further fine tuned and b) has a risk/cost implementation within it which is very easy to change.

Additionally, they can capture non-linear interactions between the variables making it very robust.

XGBoost

```
classifier = XGBClassifier(
    scale_pos_weight=weight * 3,
    objective='binary:logistic',
    eval_metric='logloss',
    max_depth=5,
    min_child_weight=1,
    gamma=0.1,
    learning_rate=0.1,
    subsample=0.8,
    colsample_bytree=0.8,
    max_delta_step=5,
    reg_lambda=10.0,
    reg_alpha=10,
    random_state=42,
    use_label_encoder=False

)
```

XGBoost in production

As mentioned before, we can change our threshold to create new loans.

Using multiple thresholds, you can see which one suits your business needs.

Threshold Score	Predicted Defaults	Default Rate	Rate Increase (vs. Previous)	% Increase in Default Rate
0.9	26	0.130	0.000	0.000
0.8	35	0.175	0.045	34.615
0.7	46	0.230	0.055	31.429
0.6	51	0.255	0.025	10.869
0.5	61	0.305	0.050	19.608
0.4	66	0.330	0.025	8.333
0.3	73	0.365	0.035	10.606
0.2	82	0.410	0.045	12.328
0.1	98	0.490	0.080	19.512

Key Assumptions and Possible Changes

Population is similar to the sample.

Pre-Policy data and Post-policy data mixed → could lead to some problems.

Models may be slightly overfitting.

Costs associated with defaults.

Thank you