Data Analysis 3 - Assignment 3: Technical Report

Apo Duman - Yahya Kocakale

Prediction of Defaulted Companies: Comparison of ML Models

1- Introduction

In this project, it is aimed to create the best possible model to predict defaulted companies in the 'Manufacture of computer, electronic and optical products' sector for 2015. Default is specifically defined as companies that were in operation in 2014 but ceased to exist in 2015. Small or medium-sized enterprises (SMEs) with annual sales ranging from €1000 to €10 million will be used in model selection.

Using panel data spanning 2005 to 2016, we aim to identify key factors contributing to firm default in this industry segment. Leveraging advanced modeling techniques, our goal is to develop a predictive model that allows stakeholders to proactively manage financial risks and increase industry stability.

Through rigorous data analysis and model building processes, we aim to provide valuable insight into firm default dynamics, facilitating informed decision-making for risk mitigation and sustainable business practices in a given industry.

We will undertake rigorous data pre-processing, variable selection and model building processes to create the best predictive model possible. Using advanced techniques such as logistic regression, ensemble methods and cross-validation, we strive to achieve high accuracy and reliability in predicting firm defaults, thus facilitating proactive risk management practices.

2- EDA and Label Engineering

In this project, raw data was taken from this **link.** In the data cleaning section, "COGS", "finished_prod", "net_dom_sales", "net_exp_sales", "wages" columns containing too many NAN values were dropped and the data set was filtered so that it does not contain 2016 data. Dummy variables have been created to see whether a company is a default or not based on its sales amount a year ago.

```
# drop variables with many NAs
data = data.drop(
    columns=["COGS", "finished_prod", "net_dom_sales", "net_exp_sales", "wages"]
)
data = data.query("year !=2016")
```

3- Sample Design and Feature Engineering

This section starts with creating industry (Manufacture of computer, electronic and optical products), year, annual sales (1000 Euros to 10 million Euros) filters. Then, dummy variables are created for firm characteristics and financial variables. After that we focus on two key aspects: normalizing profit and loss (P&L) items by sales and normalizing balance sheet (BS) items by total assets.

```python
# generate total assets
data["total_assets_bs"] = (
    data["intang_assets"] + data["curr_assets"] + data["fixed_assets"]
)
data["total_assets_bs"].describe()
```

We define two lists, pl_names and bs_names, which contain the names of profit/loss and balance sheet variables, respectively. These variables will be used for subsequent calculations.

```python
pl_names = [
    "extra_exp",
    "extra_inc",
    "extra_profit_loss",
    "inc_bef_tax",
    "inventories",
    "material_exp",
    "profit_loss_year",
    "personnel_exp",
]
bs_names = [
    "intang_assets",
    "curr_liab",
    "fixed_assets",
    "liq_assets",
    "curr_assets",
    "share_eq",
    "subscribed_cap",
    "tang_assets",
]
```

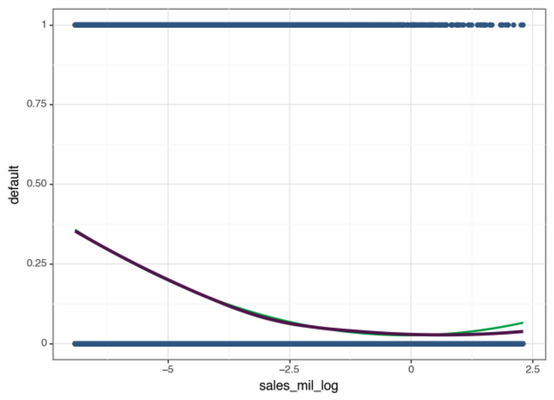
For each profit/loss item in the pl_names list, we divide the corresponding column values by the “sales” column. This normalization facilitates comparisons of these financial metrics relative to the company’s revenue.

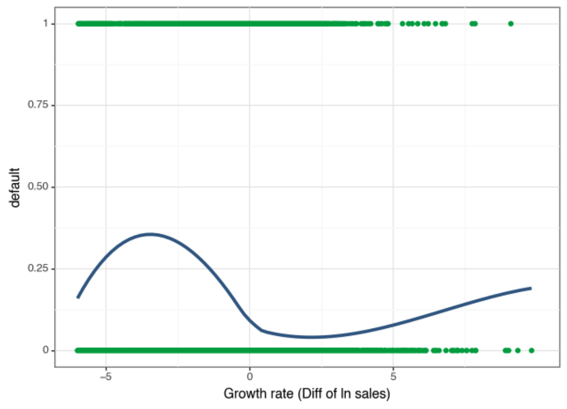
```python
# divide all pl_names elements by sales and create new column for it
data[[col + "_pl" for col in pl_names]] = data[pl_names].div(
    data["sales"], axis="index"
)
```

Similarly, for each balance sheet item in the bs_names list, we divide the corresponding column values by the “total_assets_bs” column. This normalization enables us to analyze the composition of the balance sheet relative to the total assets of the company. In cases where the “total_assets_bs” column contains NaN values, indicating missing or undefined total assets, we set the normalized balance sheet item values to NaN as well. This ensures consistency in the data treatment.

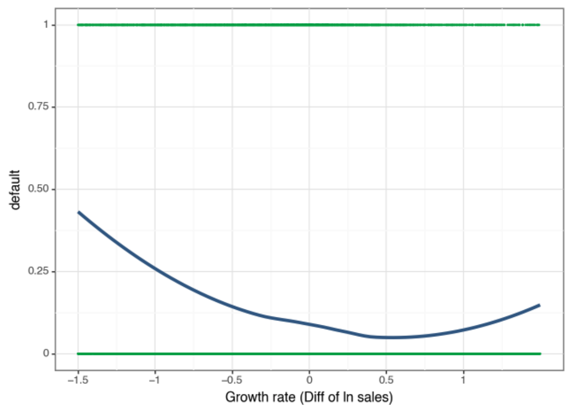
```python
# divide all bs_names elements by total_assets_bs and create new column for it
data[[col + "_bs" for col in bs_names]] = (
    data[bs_names]
    .div(data["total_assets_bs"], axis="index")
    .replace((np.inf, -np.inf, np.nan), (0, 0, 0))
)
# get Nan values where total_assets_bs is NaN
for col in bs_names:
    data[col + "_bs"] = np.where(
        data["total_assets_bs"].isna(), np.nan, data[col + "_bs"]
    )
```

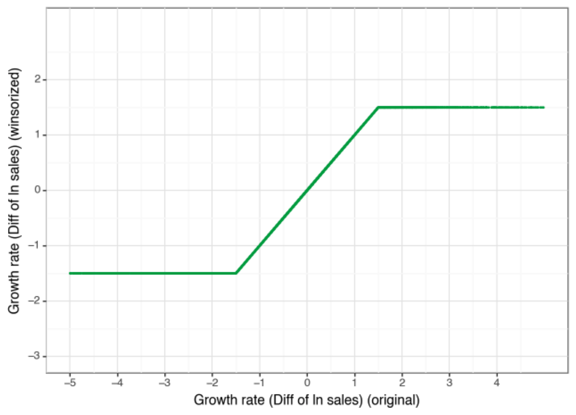
Additionally, we create flags to identify specific characteristics of financial variables. For instance, we classify variables as “flag_high” if they exceed a certain threshold, “flag_low” if they fall below a threshold, and “flag_error” if they are negative. These flags aid in identifying and categorizing anomalies or outliers in the data. After that, we perform some additional data processing steps, including CEO age calculation, handling of labor-related variables, and conversion of categorical variables. The relationship between the logarithm of sales (sales_mil_log) and the default status (default) are shown below by using a scatter plot, overlaid with two smoothed trend lines: one fitted with a linear regression model and another using locally weighted scatterplot smoothing (LOESS). A similar visualization is showed below for the growth rate of sales (d1_sales_mil_log) against the default status





Then, several new variables are generated based on the growth rate of sales (d1_sales_mil_log). Also, flags are created to identify instances where the growth rate exceeds certain thresholds. Additionally, the growth rate variable is modified to limit extreme values, a process known as “winsorization”. To depict the relationship between the original and winsorized growth rates of sales, additional visualizations are created and can be seen below. These figures provide insights into the impact of winsorization on the data distribution.





Matplotlib, patsy, scikit-learn, and statsmodels libraries are imported because these models are essential for predicting default status based on the dataset’s features and evaluating their performance using metrics like mean squared error, ROC-AUC score, and confusion matrix. Also, helper functions are defined to facilitate regression analysis and model evaluation. These functions calculate regression metrics and coefficients and summarize cross-validated model performance. Several Python libraries are imported, including matplotlib, patsy, scikit-learn, and statsmodels. These models are essential for predicting default status based on the dataset's features and evaluating their performance using metrics like mean squared error, ROC-AUC score, and confusion matrix. Also helper functions are defined to facilitate regression analysis and model evaluation. These functions calculate regression metrics and coefficients and summarize cross-validated model performance.

4- Creating Models

In this section, after defining the sets of variables like main firm variables, further financial variables, flag variables, growth variable, human capital related variables, firms history related variables and interacitons for logit and LASSO, these models were created by using this train and holdout data:

```
data_train = data.query("year == 2014 & ind2 != 26")
data_holdout = data.query("year == 2014 & ind2 == 26")
```

4.1- Simple Logit Models(M1-M5):

These setups define specific combinations of variables for different model specifications, ranging from simple logistic regression models to more complex setups incorporating various sets of predictors and interactions.

4.2- Logit + Lasso:

This set includes variables intended for logistic regression models with LASSO regularization, which helps in feature selection and model regularization.

4.3- Random Forest

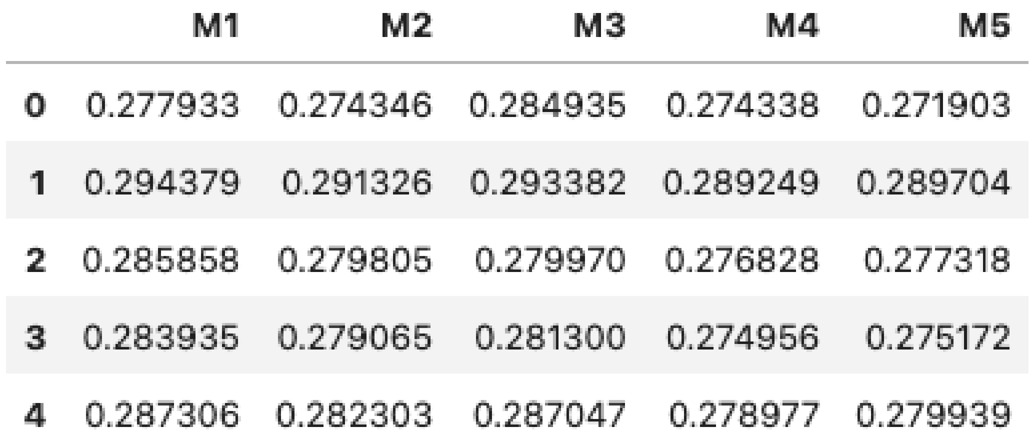
Additionally, the data is split into separate training and holdout datasets (data_train and data_holdout) based on the year and industry code, facilitating model training and evaluation on independent datasets.

By defining these variable sets and datasets, we establish the foundation for building and evaluating predictive models for default risk assessment using a variety of machine learning techniques.

5- Predict probabilities with logit and Lasso with CV

To predict probabilities with logistic regression and LASSO using 5-fold cross-validation, we follow these steps:

- Specify a 5-fold cross-validation method (k) to evaluate the models' performance.
- Define a list of variable sets (logit_model_vars) for different logistic models (M1 to M5).
- For each model specification, fit logistic regression models using cross-validation (LogisticRegressionCV) with the specified parameters.
- Calculate the root mean squared error (RMSE) on the test set for each fold of cross-validation.



	M1	M2	M3	M4	M5
0	0.277933	0.274346	0.284935	0.274338	0.271903
1	0.294379	0.291326	0.293382	0.289249	0.289704
2	0.285858	0.279805	0.279970	0.276828	0.277318
3	0.283935	0.279065	0.281300	0.274956	0.275172
4	0.287306	0.282303	0.287047	0.278977	0.279939

Figure 1: 5-Fold Cross-Validation

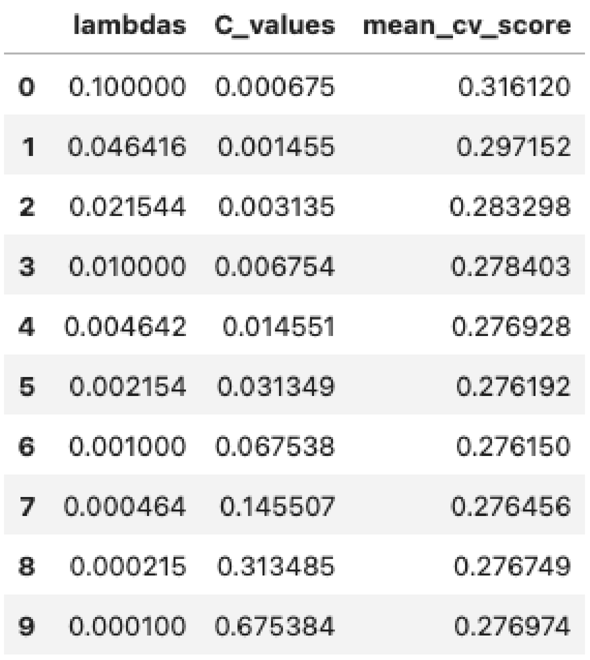
5.1- Cross Validate Logit Models

5.2 Logit + LASSO:

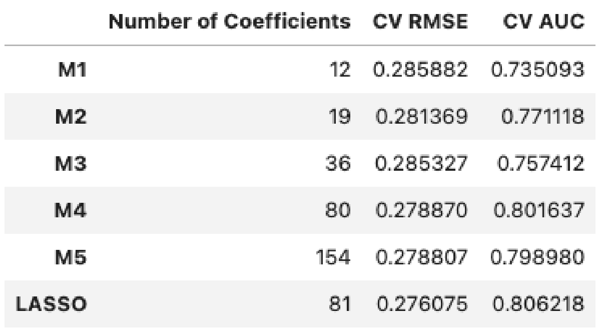
- Specify the model equation using all the variables intended for logistic regression with LASSO regularization (logit_lasso_vars).
- Normalize the input variables and set up regularization parameters for LASSO.
- Initialize and fit logistic regression with LASSO using cross-validation (LogisticRegressionCV) with the specified parameters.
- Calculate the mean cross-validated RMSE (after converting negative Brier scores) and select the best lambda for LASSO regularization.
- Extract the cross-validated test RMSE for the LASSO model with the best lambda.

5.3 AUC, Calibration Curve, Confusion Matrix, ROC

- Calculate the area under the ROC curve (AUC) for both logistic regression and LASSO models.
- For logistic regression, use the same procedure as for RMSE calculation, but with the scoring parameter set to "roc_auc".
- For LASSO, fit a logistic regression model with LASSO using cross-validation and calculate the AUC for the best lambda.
- Compile a summary frame containing the number of coefficients, cross-validated RMSE, and AUC for each model.



	lambdas	C_values	mean_cv_score
0	0.100000	0.000675	0.316120
1	0.046416	0.001455	0.297152
2	0.021544	0.003135	0.283298
3	0.010000	0.006754	0.278403
4	0.004642	0.014551	0.276928
5	0.002154	0.031349	0.276192
6	0.001000	0.067538	0.276150
7	0.000464	0.145507	0.276456
8	0.000215	0.313485	0.276749
9	0.000100	0.675384	0.276974



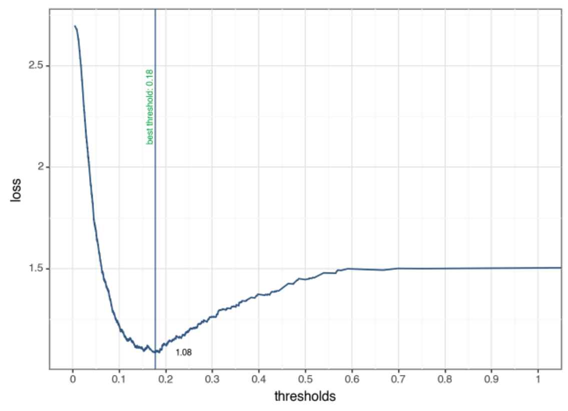
	Number of Coefficients	CV RMSE	CV AUC
M1	12	0.285882	0.735093
M2	19	0.281369	0.771118
M3	36	0.285327	0.757412
M4	80	0.278870	0.801637
M5	154	0.278807	0.798980
LASSO	81	0.276075	0.806218

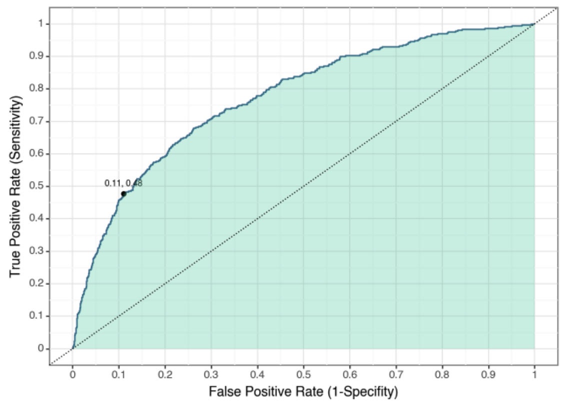
5.4 Minimization of the loss function

To find the optimal threshold that minimizes the loss function, we can iterate through different models and folds, calculating the cost associated with false negative and false positive classifications for each threshold. Then, select the threshold with the lowest overall cost.

```
FP = 3
FN = 15
cost = FN / FP
```

```
summary_with_lossfnc = pd.DataFrame(
    best_thresholds_cv.items(), columns=["Model", "Avg of optimal thresholds"]
)
summary_with_lossfnc["Threshold for Fold5"] = fold5_threshold.values()
summary_with_lossfnc["Avg expected loss"] = expected_loss_cv.values()
summary_with_lossfnc["Expected loss for Fold5"] = fold5_expected_loss.values()
```



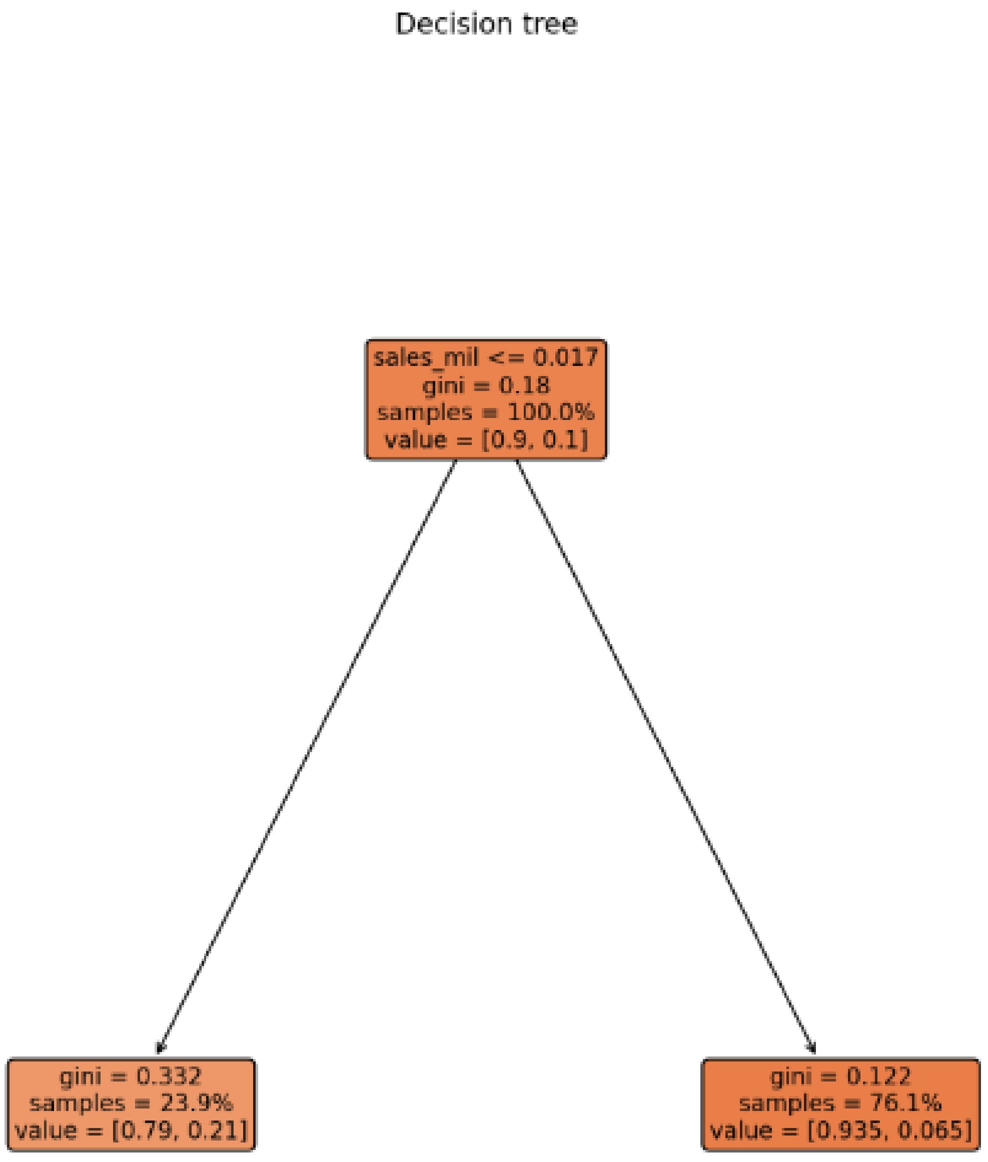


6- Prediction with Random Forest

Now, as a final step, we employ a random forest classification to predict defaults. We will compare this model with previous Logit and Lasso models

6.1- Graph example for decision tree

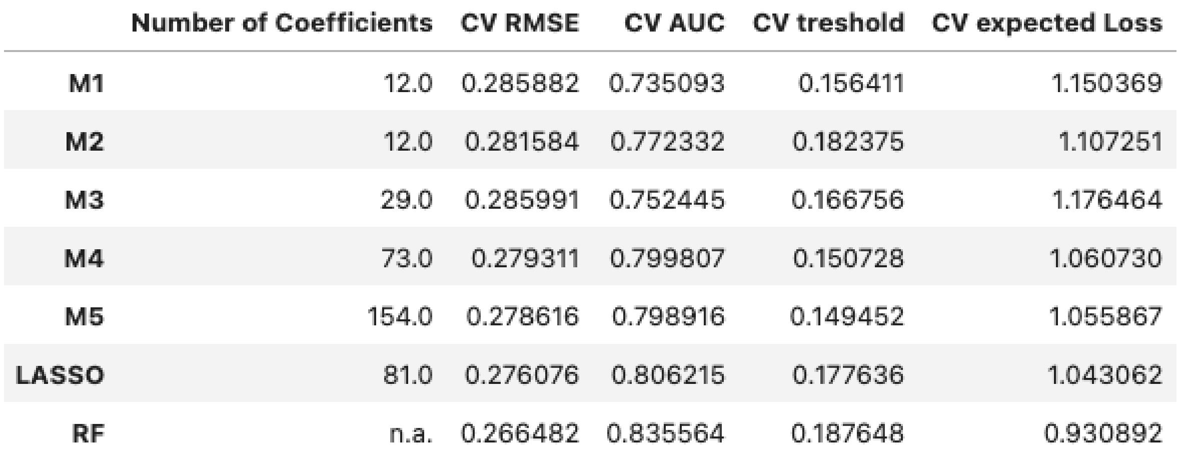
We start by visualizing a decision tree as an example. Using the features “sales_mil”, “profit_loss_year”, and “foreign_management”, we build a decision tree classifier and plot it to understand the decision-making pro-



cess.

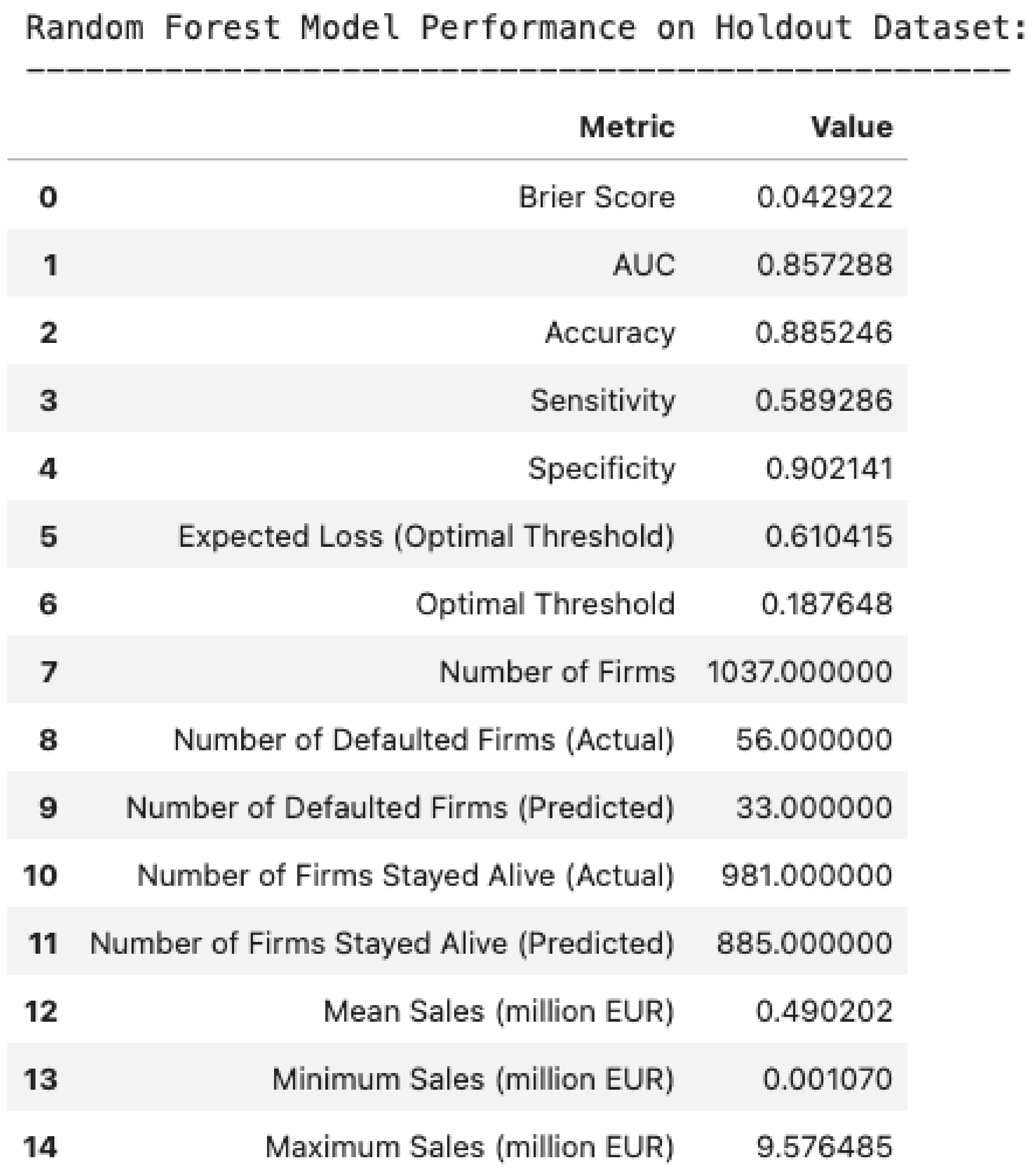
6.2- Random Forest

As a final step, we employ a random forest classification to predict defaults. We will compare this model with previous Logit and Lasso models. We created a random forest classifier to predict probabilities. We perform hyperparameter tuning using grid search with cross-validation and evaluate the model's performance using metrics such as ROC AUC and RMSE.



	Number of Coefficients	CV RMSE	CV AUC	CV treshold	CV expected Loss
M1	12.0	0.285882	0.735093	0.156411	1.150369
M2	12.0	0.281584	0.772332	0.182375	1.107251
M3	29.0	0.285991	0.752445	0.166756	1.176464
M4	73.0	0.279311	0.799807	0.150728	1.060730
M5	154.0	0.278616	0.798916	0.149452	1.055867
LASSO	81.0	0.276076	0.806215	0.177636	1.043062
RF	n.a.	0.266482	0.835564	0.187648	0.930892

Figure 2: Performance of Models



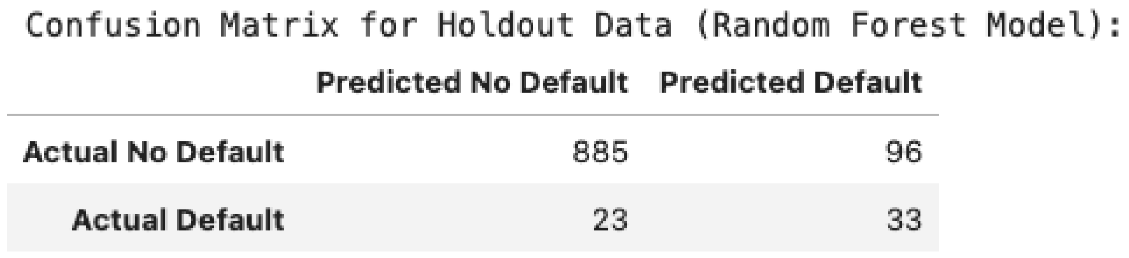
```
Random Forest Model Performance on Holdout Dataset:
--------------------------------------------------
```

	Metric	Value
0	Brier Score	0.042922
1	AUC	0.857288
2	Accuracy	0.885246
3	Sensitivity	0.589286
4	Specificity	0.902141
5	Expected Loss (Optimal Threshold)	0.610415
6	Optimal Threshold	0.187648
7	Number of Firms	1037.000000
8	Number of Defaulted Firms (Actual)	56.000000
9	Number of Defaulted Firms (Predicted)	33.000000
10	Number of Firms Stayed Alive (Actual)	981.000000
11	Number of Firms Stayed Alive (Predicted)	885.000000
12	Mean Sales (million EUR)	0.490202
13	Minimum Sales (million EUR)	0.001070
14	Maximum Sales (million EUR)	9.576485

```python
best_logit_optimal_treshold = best_thresholds_cv["M2"]

# Get expected loss on holdout
holdout_treshold = np.where(
    data_holdout["best_logit_pred_proba"] < best_logit_optimal_treshold, 0, 1
)
tn, fp, fn, tp = confusion_matrix(
    data_holdout["default"], holdout_treshold, labels=[0, 1]
).ravel()
expected_loss_holdout = (fp * FP + fn * FN) / len(data_holdout["default"])
round(expected_loss_holdout, 3)
```

- Compared to logit and lass models, random forest gives better results both on training and on holdout data. While M2 model gives an expected loss of 0.686 (least among logit models) on holdout data, Random Forest gives is 0.61.


Confusion Matrix for Holdout Data (Random Forest Model):

	Predicted No Default	Predicted Default
Actual No Default	885	96
Actual Default	23	33

The summary table for our selected model shows that we predict 33 defaults out of 56 actual defaults, which is not a bad success rate.

7- Conclusion

In this analysis, we explored the use of random forest for predicting defaulted firms in the 'Manufacture of computer, electronic and optical products' industry for the year 2015. Through the development and evaluation of decision trees and random forests, we identified optimal parameters and evaluated model performance using cross-validation. The selected Random Forest model demonstrated promising results in terms of RMSE, AUC, and expected loss. This predictive model can serve as a valuable tool for stakeholders in proactively managing financial risks and facilitating informed decision-making within the specified industry segment.