ASSET ESTIMATION OF THE BANK CUSTOMERS VIA MACHINE LEARNING APPROACH

Bachelor's Thesis

Advisor: Assoc. Prof. Nezir AYDIN
Prepared by: Belce KAYA
Student Number: 15061049

ABSTRACT

This thesis is about the asset amount prediction of the bank customer with the available data (Bank Data Warehouse, KKB, Neighborhood data). In that way, it is aimed to make the right campaign for the right customer. In order to establish this predictive model, the main goal is to use machine learning algorithms and to make the model in a software program that supports these algorithms. As mentioned in the literature review, there are various algorithms suitable for the predictive model. The most accurate algorithm which gives the most optimal result is desired to choose.

INTRODUCTION
LITERATURE REVIEW

Predictive Modeling

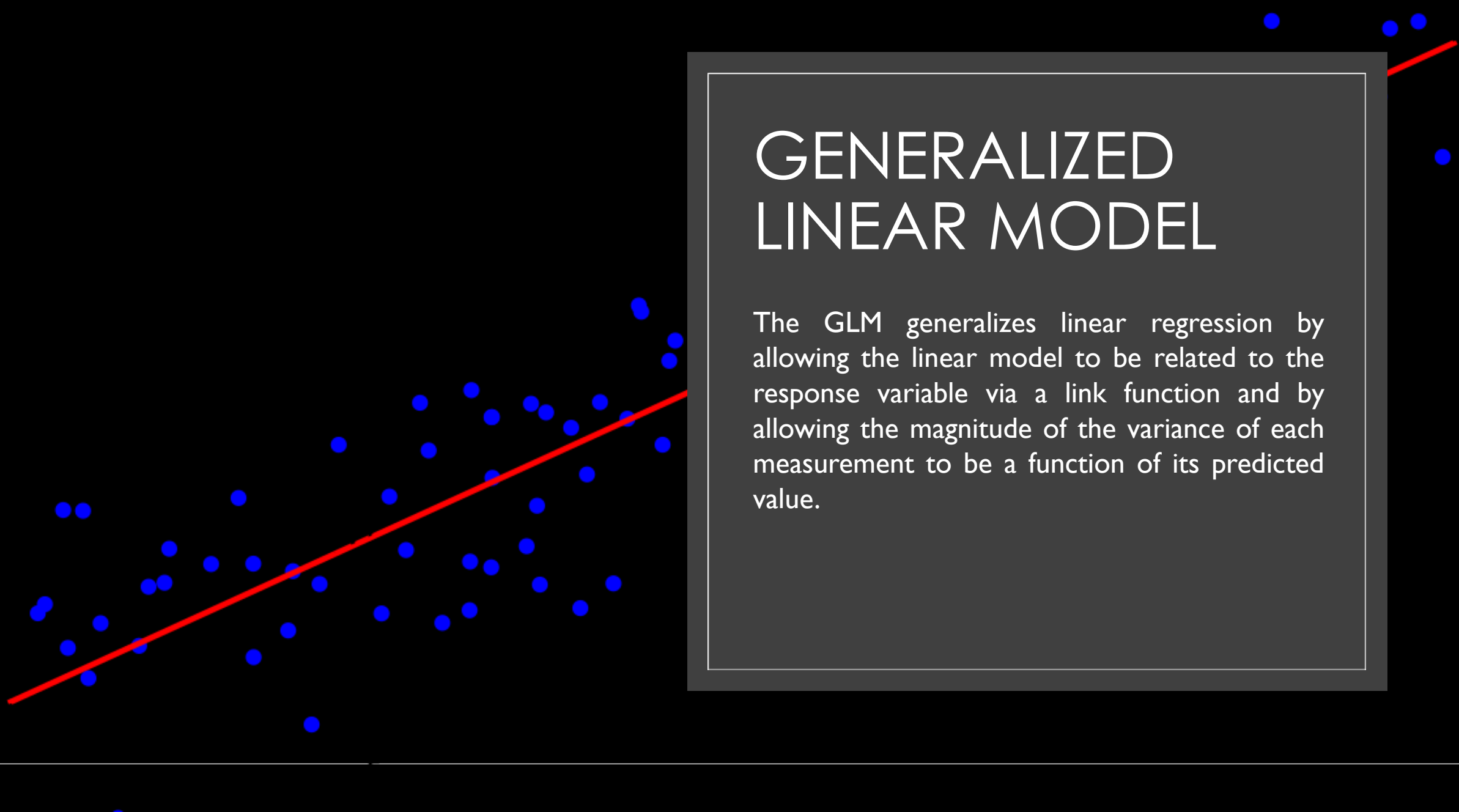
Generally used to feed predictive models:

- Transaction data
- CRM data
- Customer service data
- Survey or polling data
- Digital marketing and advertising data
- Economic data
- Demographic data
- Machine-generated data (for example, telemetric data or data from sensors)
- Geographical data
- Web traffic data

INTRODUCTION
LITERATURE REVIEW

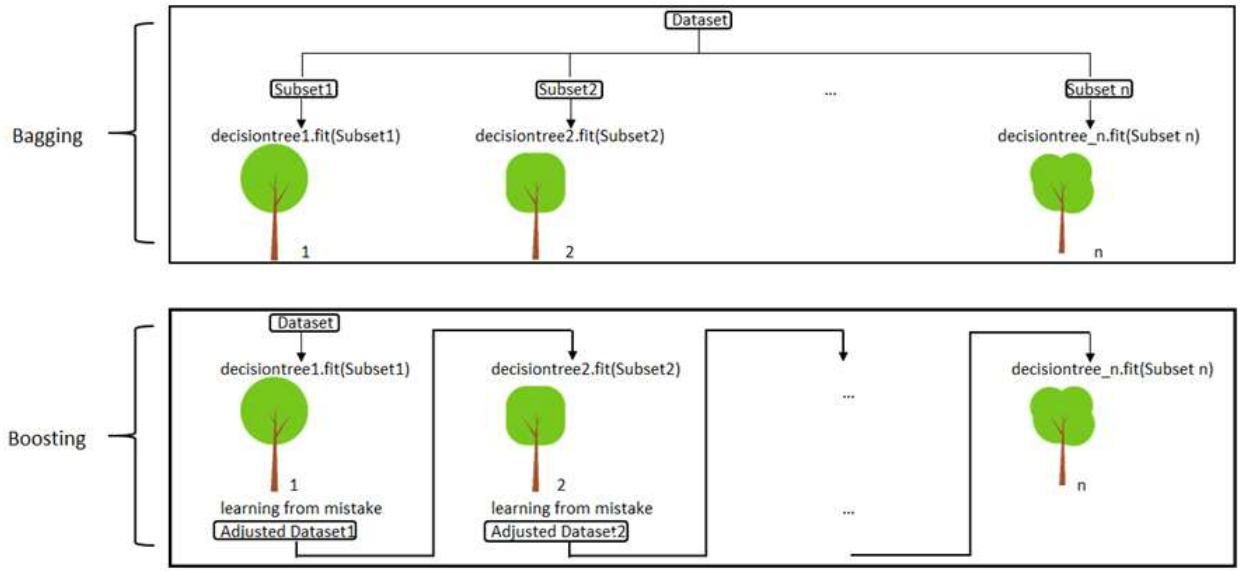
Types of Predictive Models

- Generalized Linear Models (GLM)
- Random Forests
- Decision Trees
- Neural Networks
- Gradient Boosting Models (GBM)
- Extended Gradient Boosting Models (XGBoost)



GENERALIZED LINEAR MODEL

The GLM generalizes linear regression by allowing the linear model to be related to the response variable via a link function and by allowing the magnitude of the variance of each measurement to be a function of its predicted value.

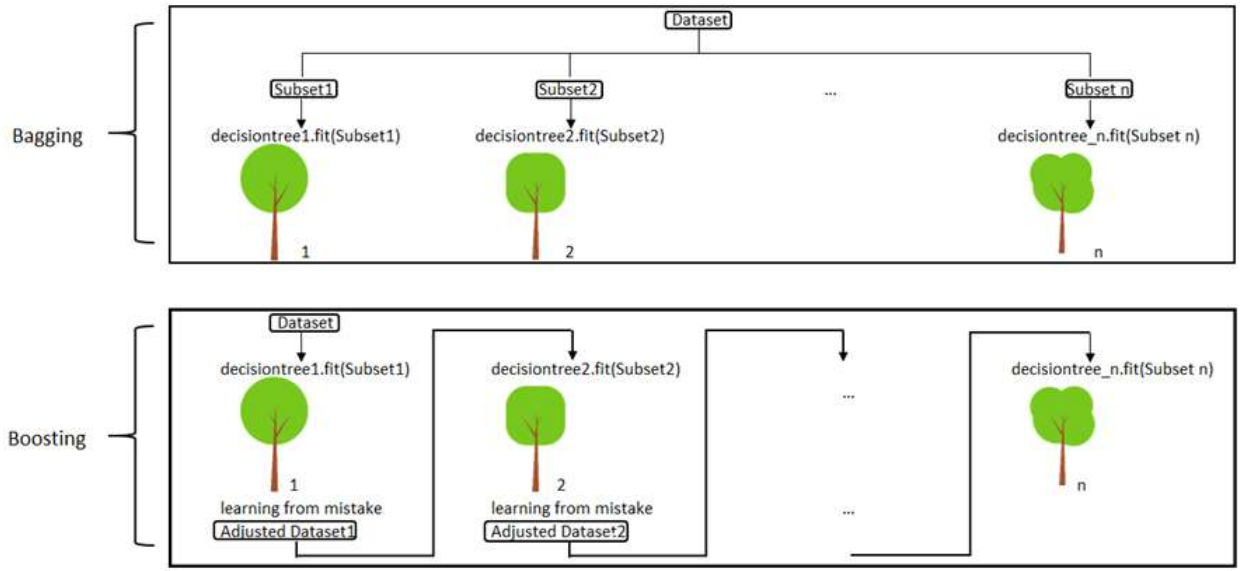


BOOSTING AND BAGGING

BAGGING (BOOTSTRAP):

- trains a bunch of individual models in a parallel way
- uses the same training algorithms for every predictor and trains them on different random subsets

Decision Trees, Random Forest



BOOSTING AND BAGGING

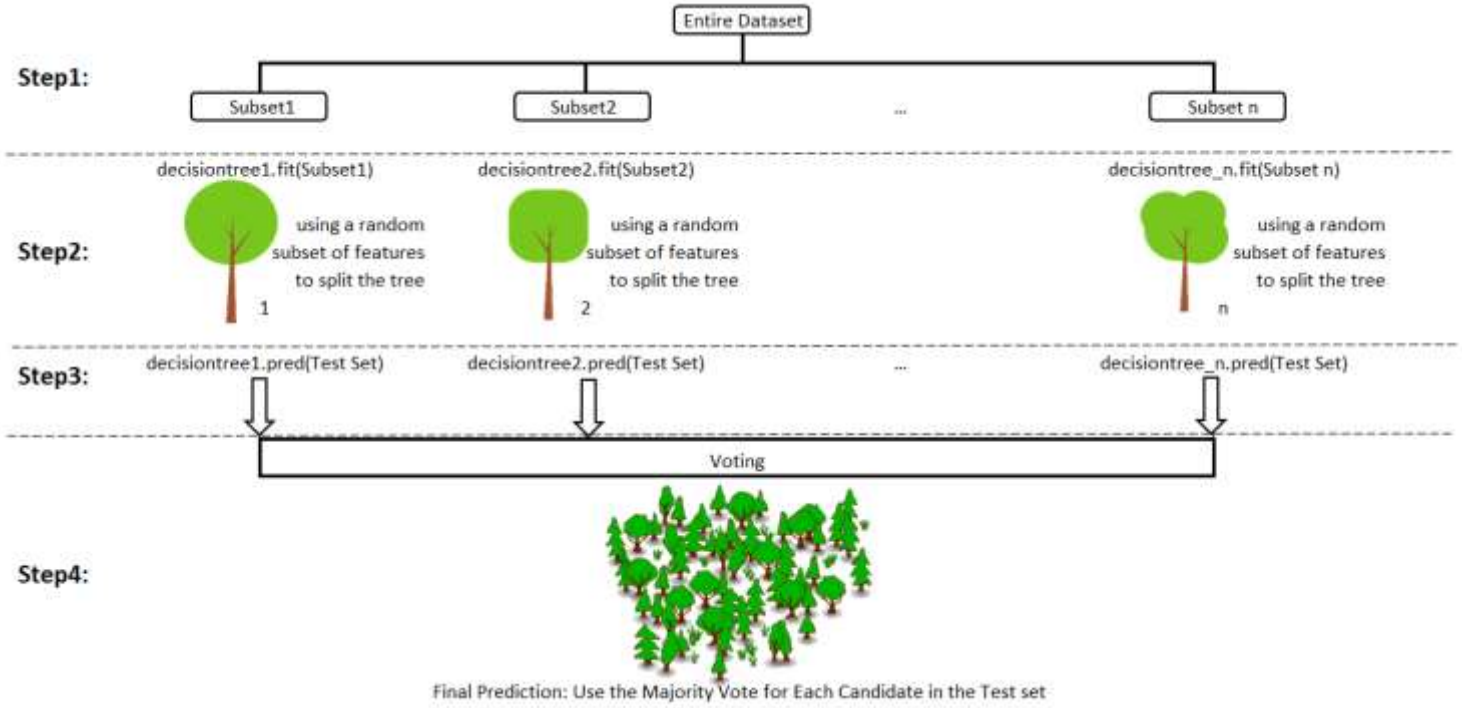
BOOSTING:

- Ensemble method that can combine several weak learns into a strong learner.
- trains predictors sequentially
- adds iterations of the model sequentially
- adjusts the weights of the weak learners along the way.

AdaBoost, Gradient Tree Boosting, XGBoost

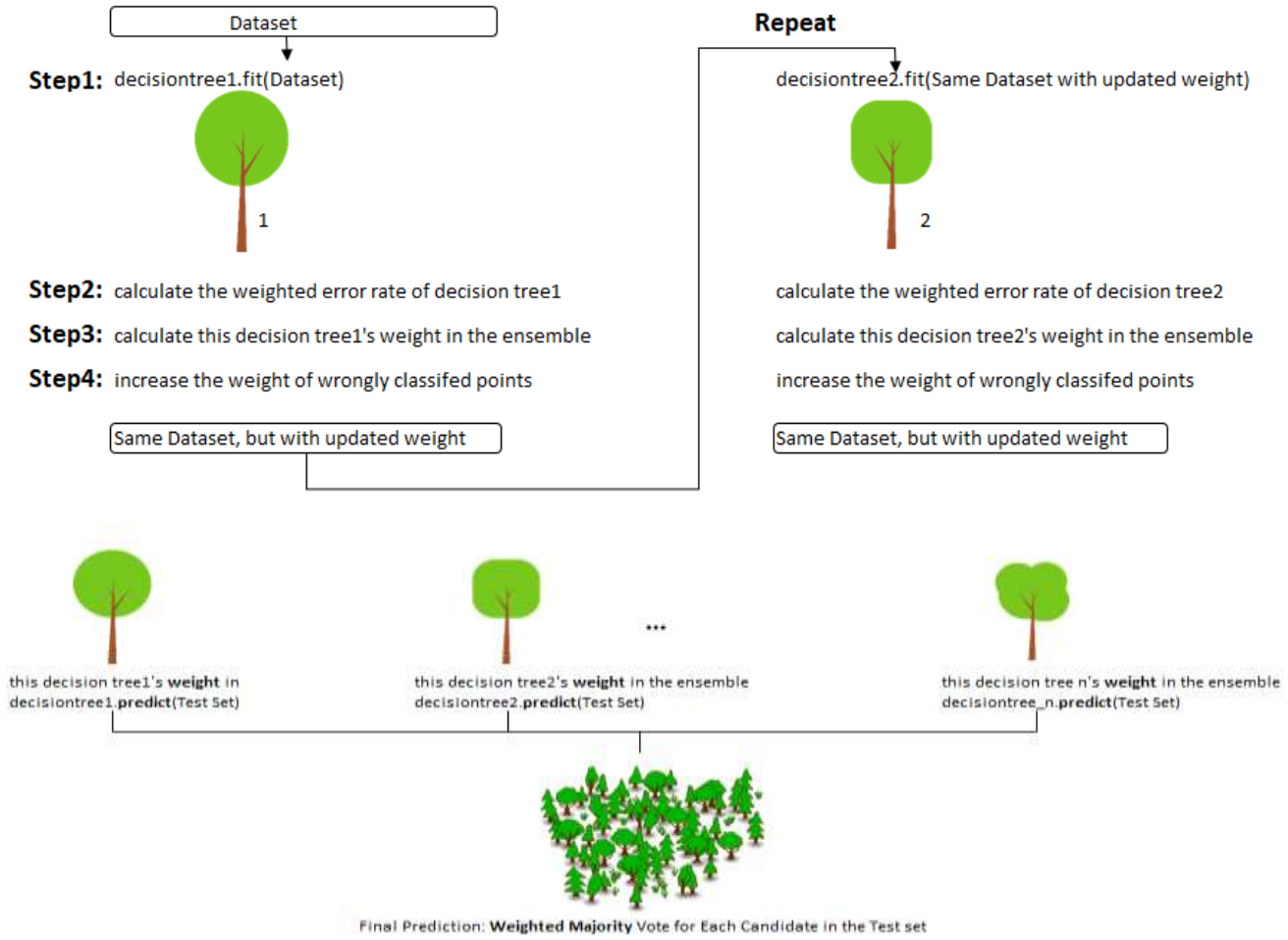
Random forest is an ensemble model of Decision Trees, trained via the bagging method

ENSEMBLE LEARNING ALGORITHMS – RANDOM FOREST



- AdaBoost is a boosting ensemble model and works especially well with the decision tree. Boosting model's key is learning from the previous mistakes,
- AdaBoost learns from the mistakes by increasing the weight of misclassified data points.

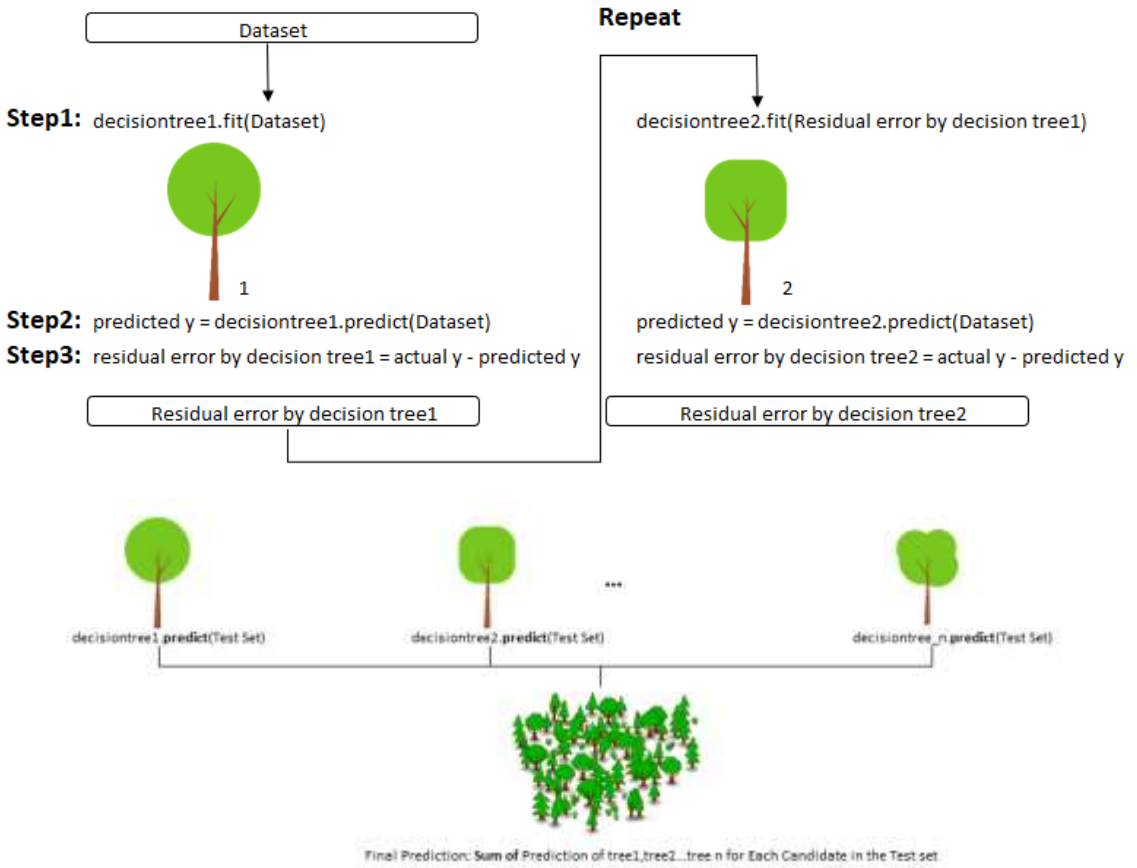
ENSEMBLE LEARNING ALGORITHMS – ADABOOST



The weight of this tree = learning rate * log ((1 — e) / e)

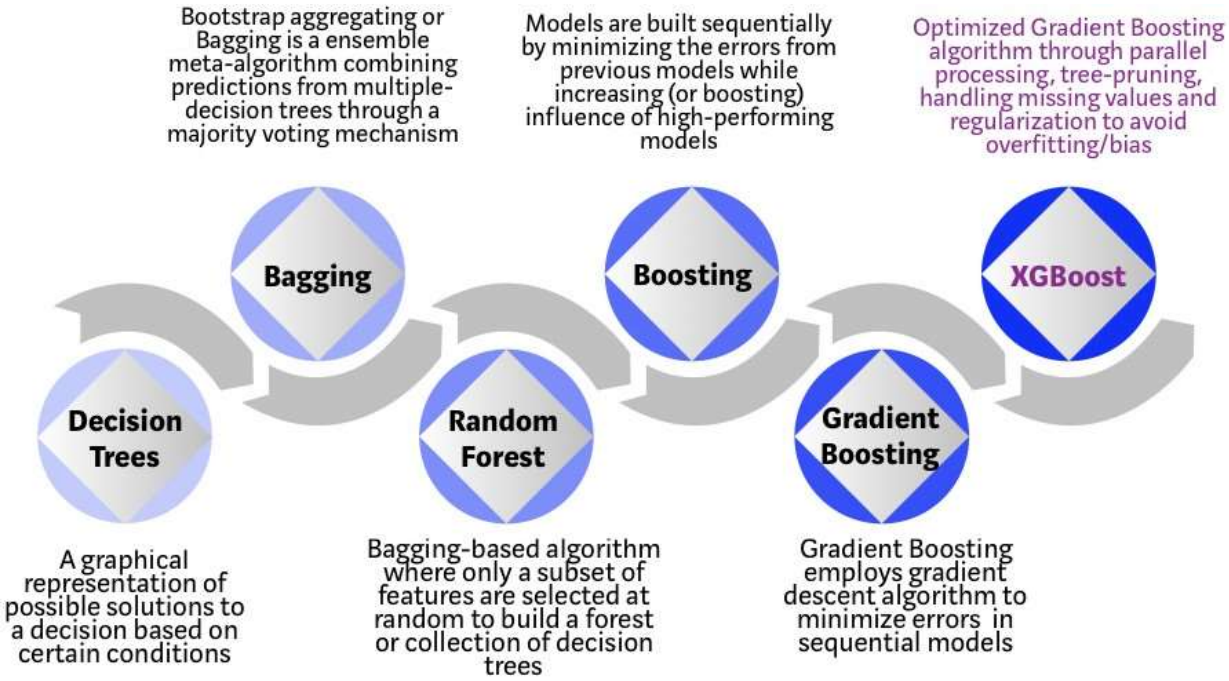
- Gradient boosting is another boosting model. As one may remember, boosting model's key is learning from the previous mistakes.
- Gradient Boosting learns from the residual error directly, rather than update the weights of data points.

ENSEMBLE LEARNING ALGORITHMS – GRADIENT BOOSTING

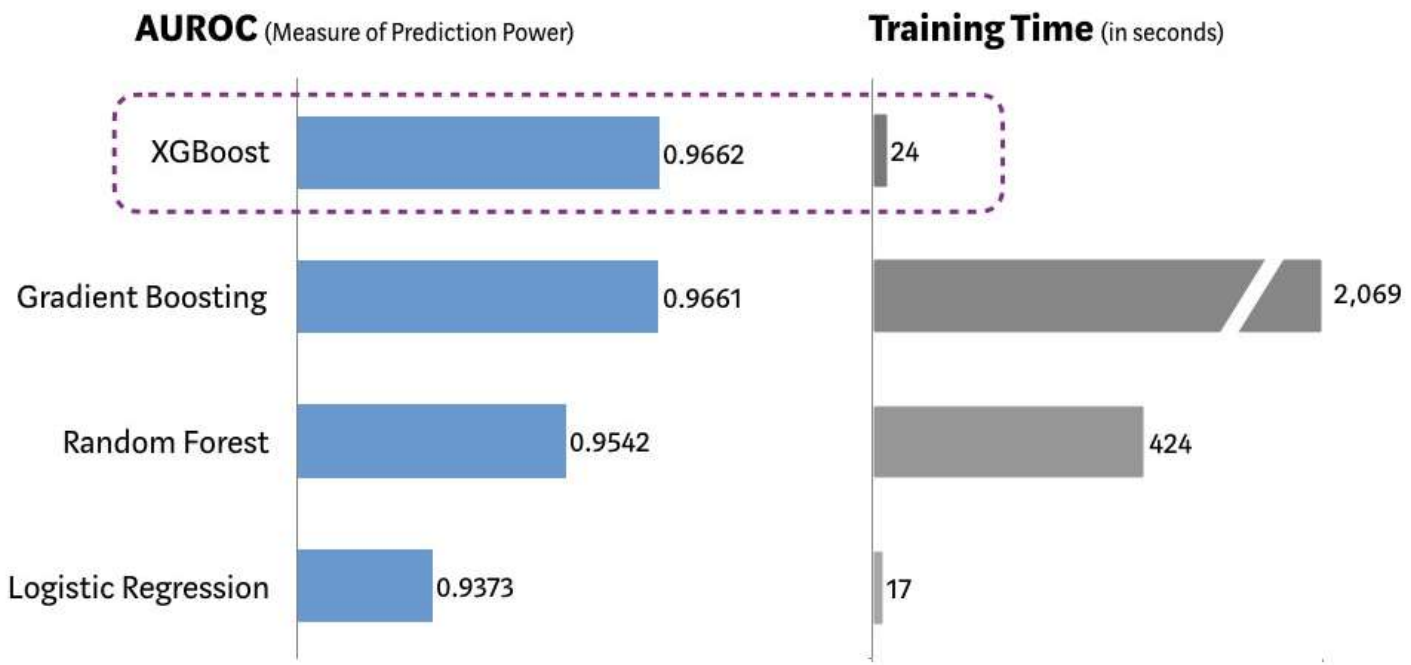


- XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework.
- In prediction problems involving unstructured data (images, text, etc.) artificial neural networks tend to outperform all other algorithms or frameworks. However, when it comes to small-to-medium structured /tabular data, decision tree-based algorithms are considered best-in-class right now.

XGBOOST



EVOLUTION OF TREE-BASED ALGOS

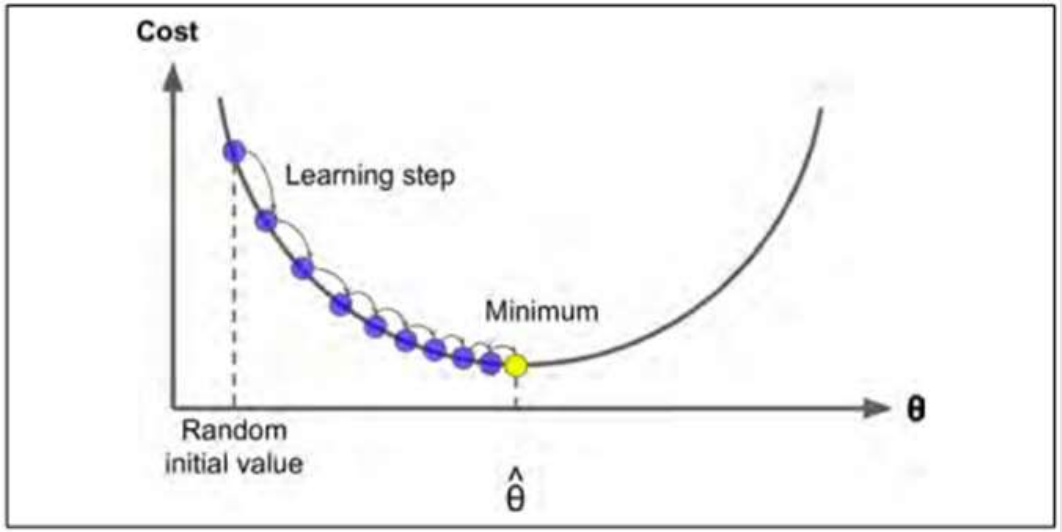


COMPARISON OF ALGORITHMS

XGBoost Hyper-parameter Tuning

Gradient Descent (Learning Rate - eta)

- optimization algorithm capable of finding optimal solutions to a wide range of problems
- tweaks parameters iteratively in order to minimize a cost function
- after each boosting step, the weights of new features can be gotten. eta shrinks the feature weights to make the boosting process more conservative and to reach the best optimum.
- **Once the gradient is zero, minimum point has been reached!**

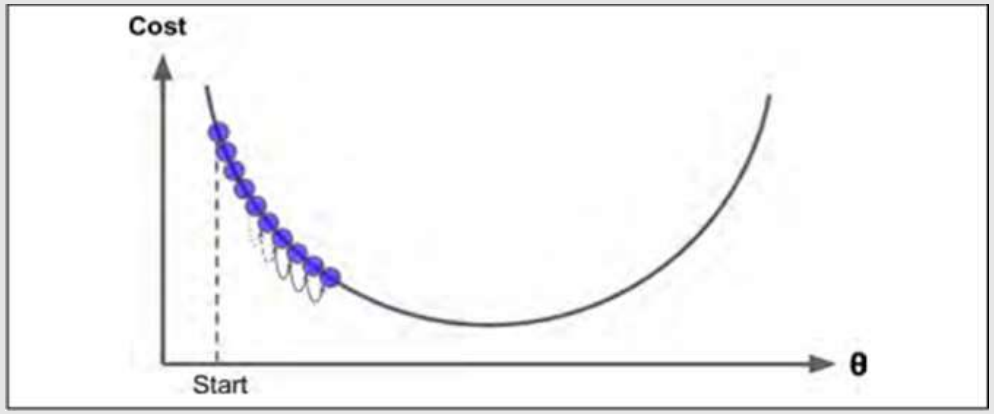


XGBoost Hyper-parameter Tuning

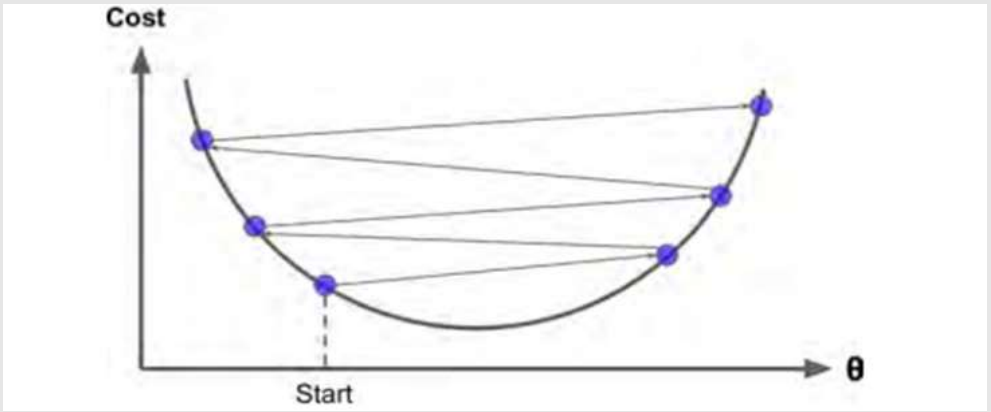
Gradient Descent (Learning Rate - eta)

- An important parameter in Gradient Descent is the size of the steps, determined by the learning rate hyperparameter.

Takes a long time! Many iterations…	Takes a short time but fails to find a good solution!



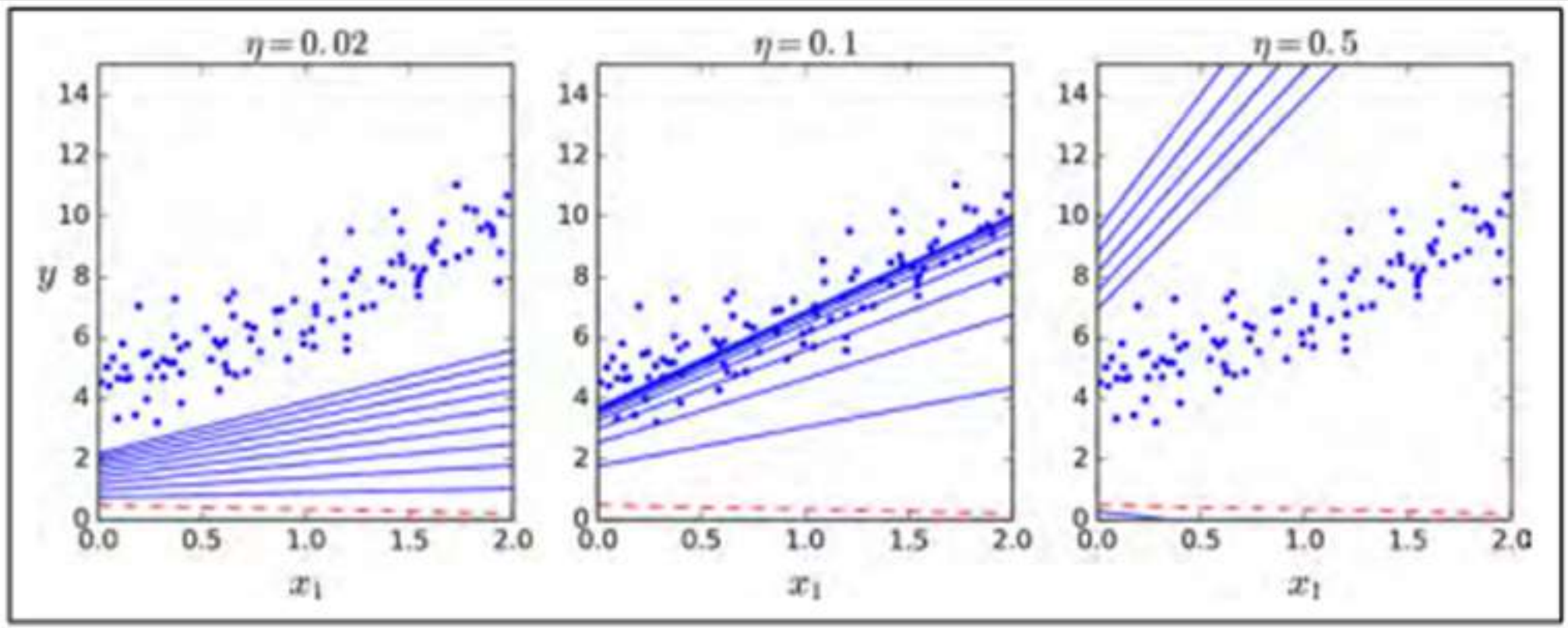
Too Small Learning Rate



Too large Learning Rate

XGBoost Hyper-parameter Tuning

Gradient Descent (Learning Rate - eta)



Gradient Descent with Various Learning Rate

XGBoost Hyper-parameter Tuning

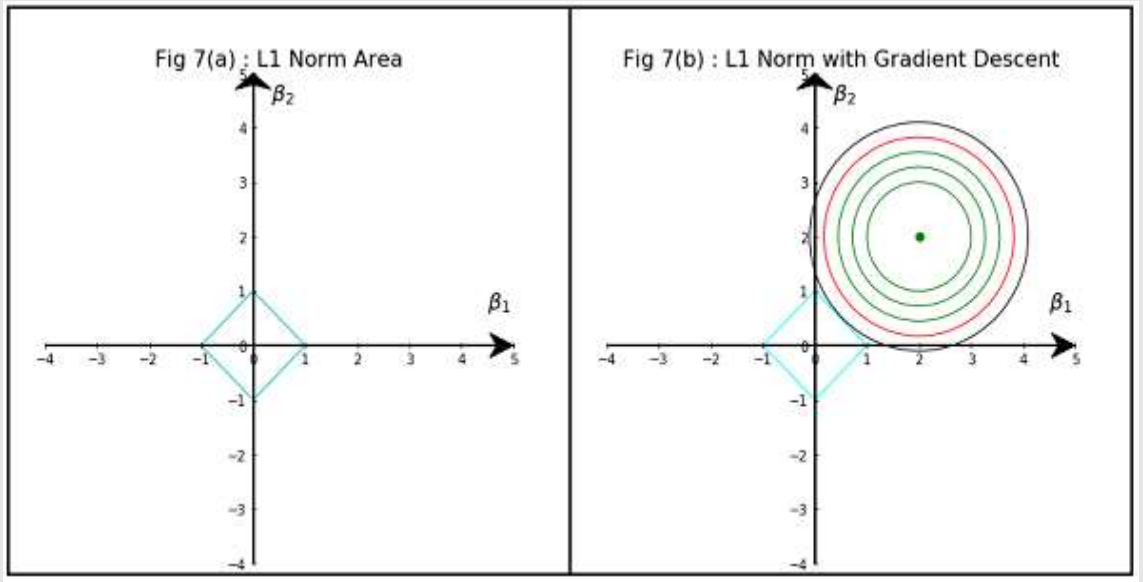
Gradient Descent (L1/L2 Regularization Constants-Lambda)

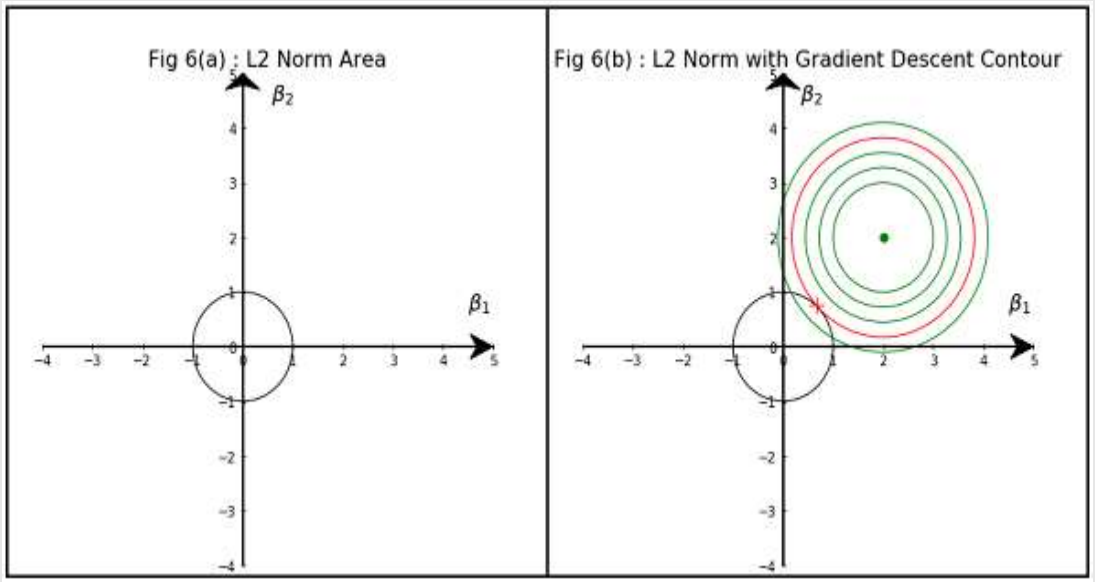
$$\hat{\beta}^{\text{lasso}} = \underset{\beta}{\operatorname{argmin}} \left\{ \frac{1}{2} \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij} \beta_j \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\}$$

L1 Norm or Lasso Regression

$$\hat{\beta}^{\text{ridge}} = \underset{\beta}{\operatorname{argmin}} \left\{ \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij} \beta_j \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\}$$

L2 Norm or Ridge Regression

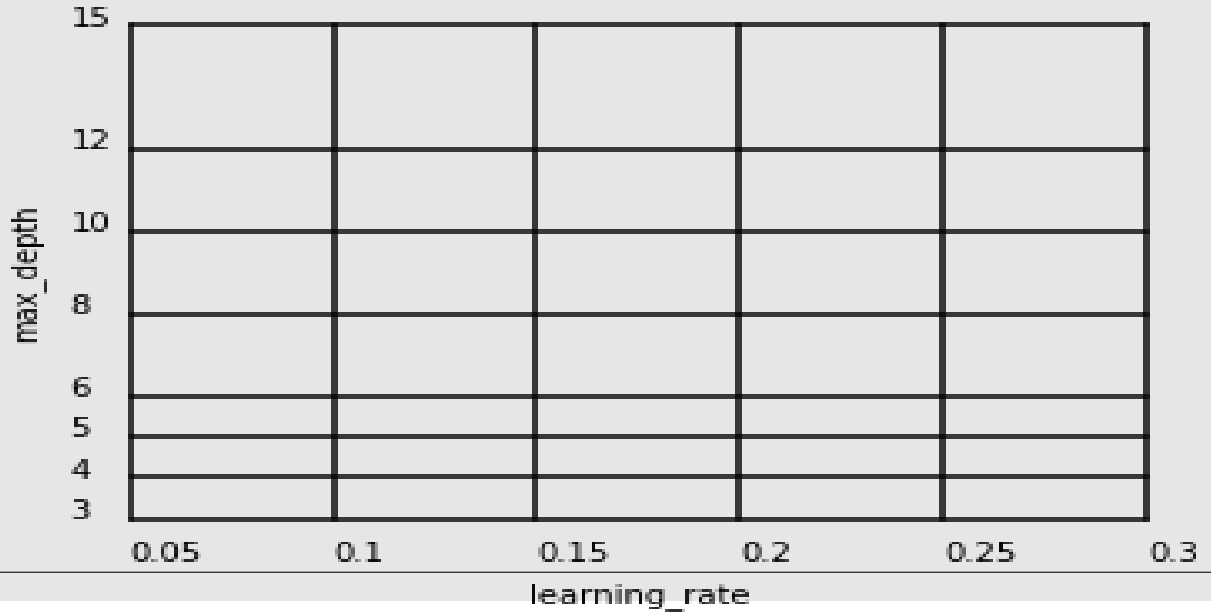




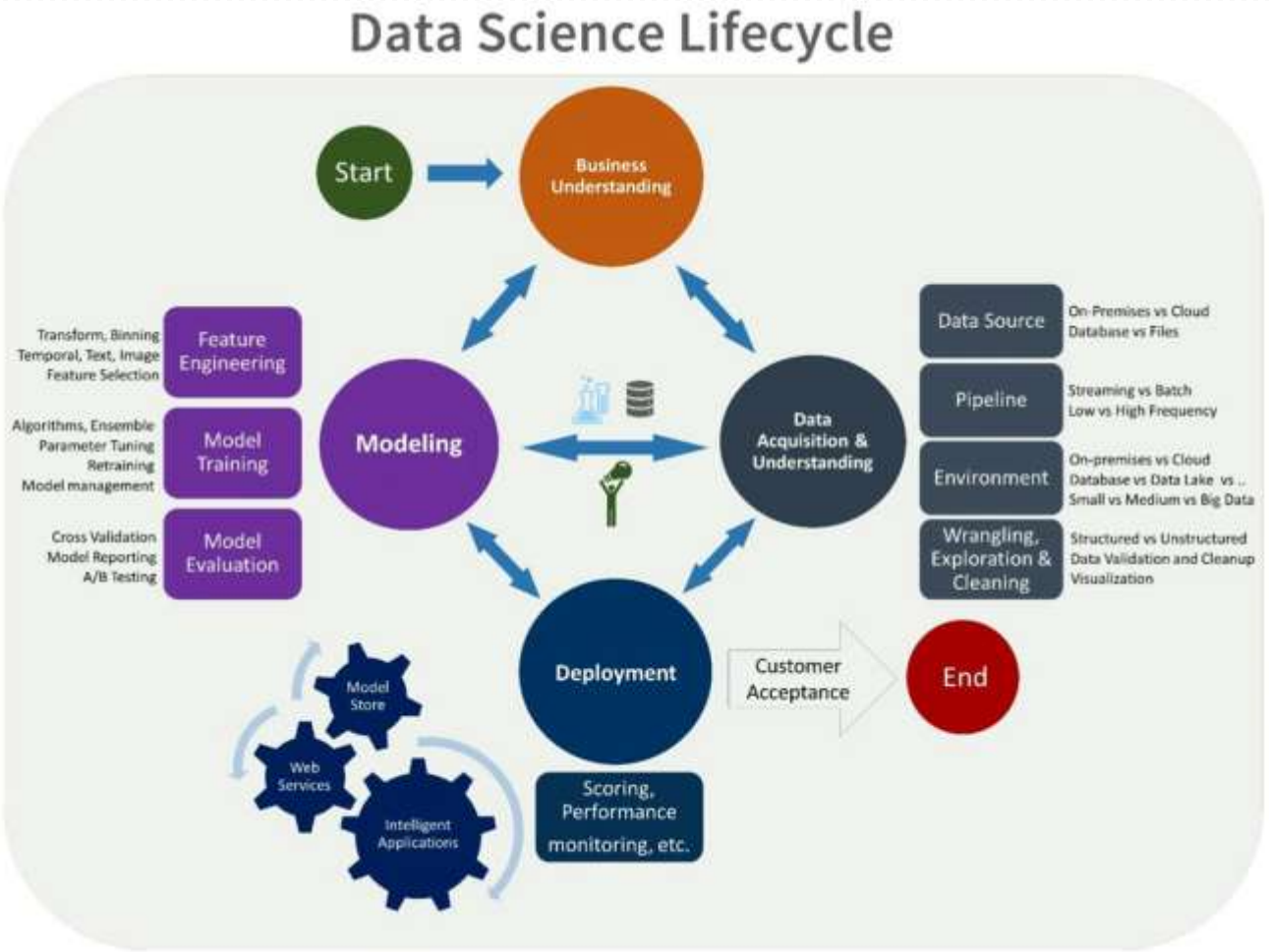
XGBoost Hyper-parameter Tuning

Grid Search (GS)

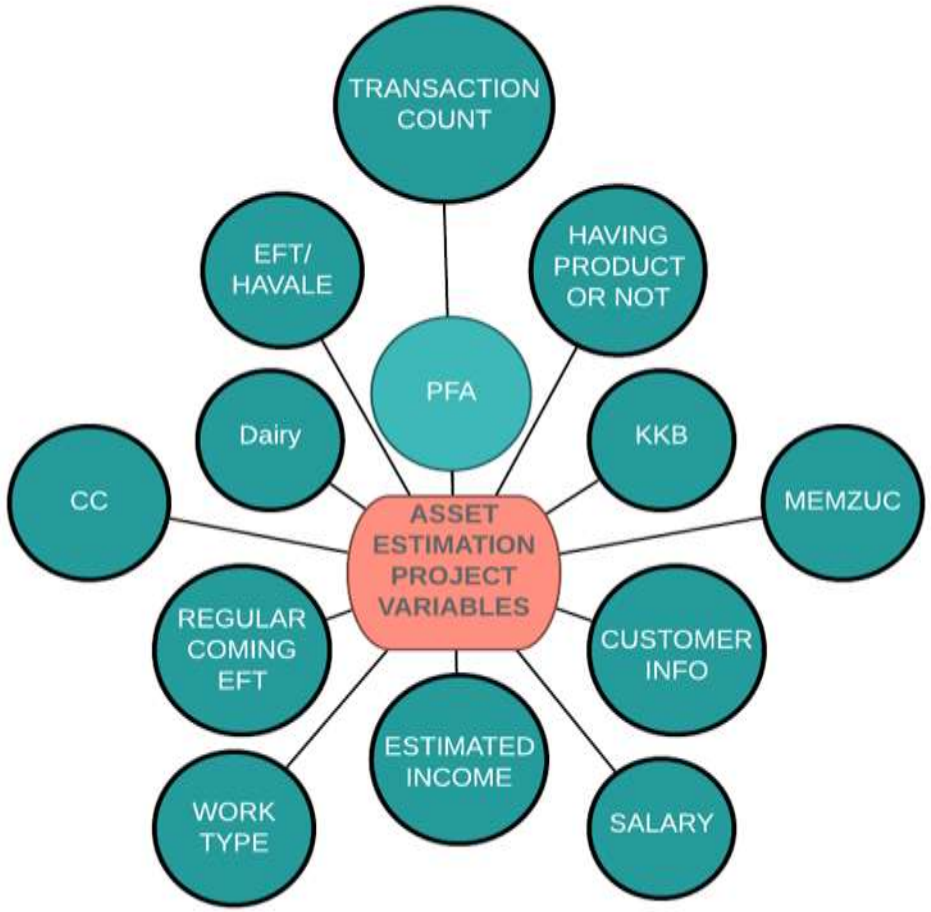
```
{"learning_rate"      : [0.05, 0.10, 0.15, 0.20, 0.25, 0.30 ] ,
 "max_depth"          : [ 3,  4,  5,  6,  8,  10,  12,  15],
 "min_child_weight"   :  [  1,  3,  5,  7  ],
 "gamma"              : [ 0.0,  0.1,  0.2  ,  0.3,  0.4  ],
 "colsample_bytree"   : [ 0.3, 0.4, 0.5 , 0.7 ] }
```



METHODOLOGY



DATA ACQUISITION & UNDERSTANDING



DATA ACQUISITION & UNDERSTANDING

KKB

QUERY_ID	CREDIT TYPE	BANK
2	02	YELLOW BANK
2	23(CC)	YELLOW BANK
2	26	YELLOW BANK
2	23(CC)	X(INVISIBLE)

DATA ACQUISITION & UNDERSTANDING

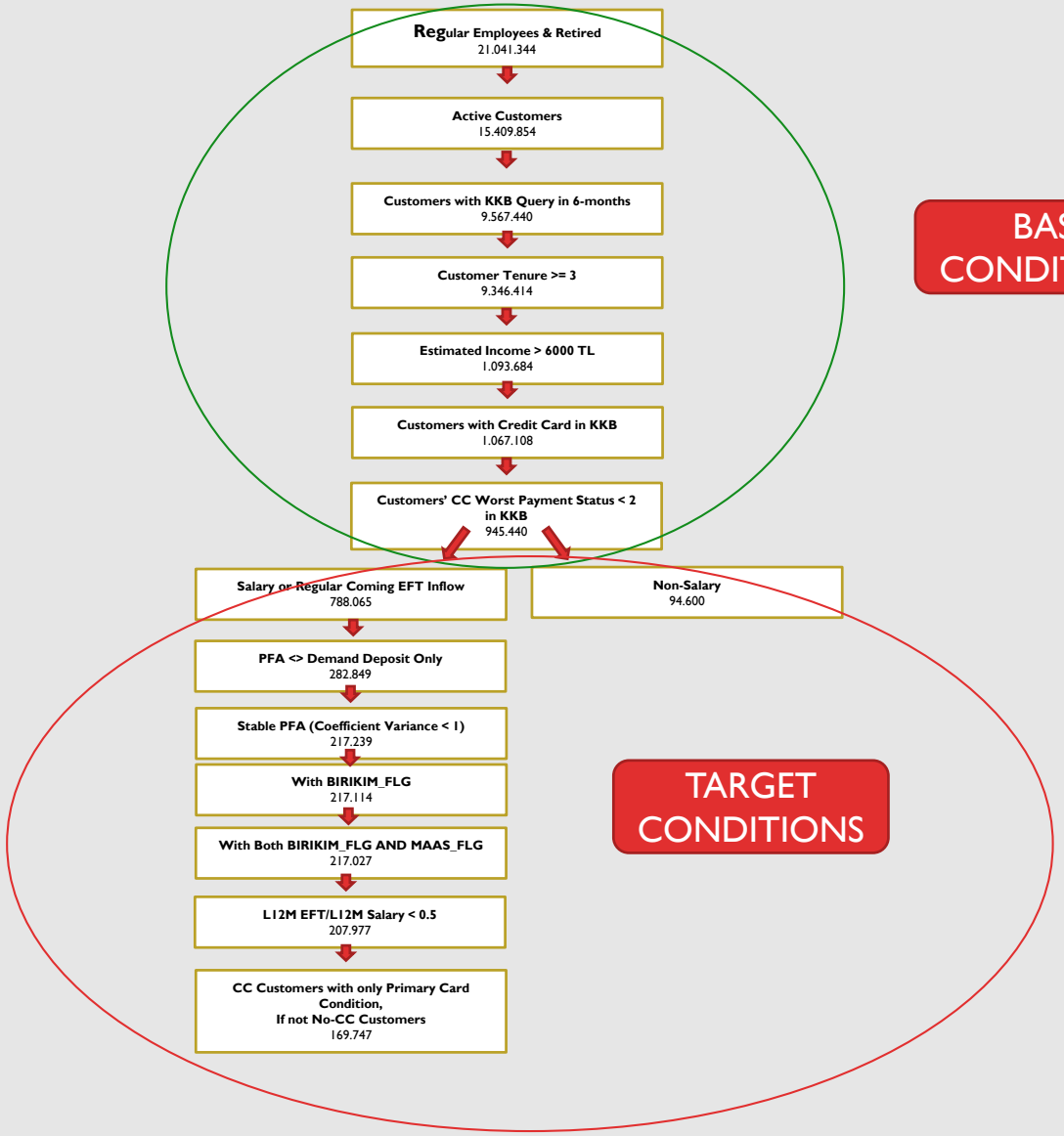
CONSTRAINTS

ASSET: In this case, demand deposit, foreign currency (FX), time deposit, Investment, Bond, repo etc. **(+)**

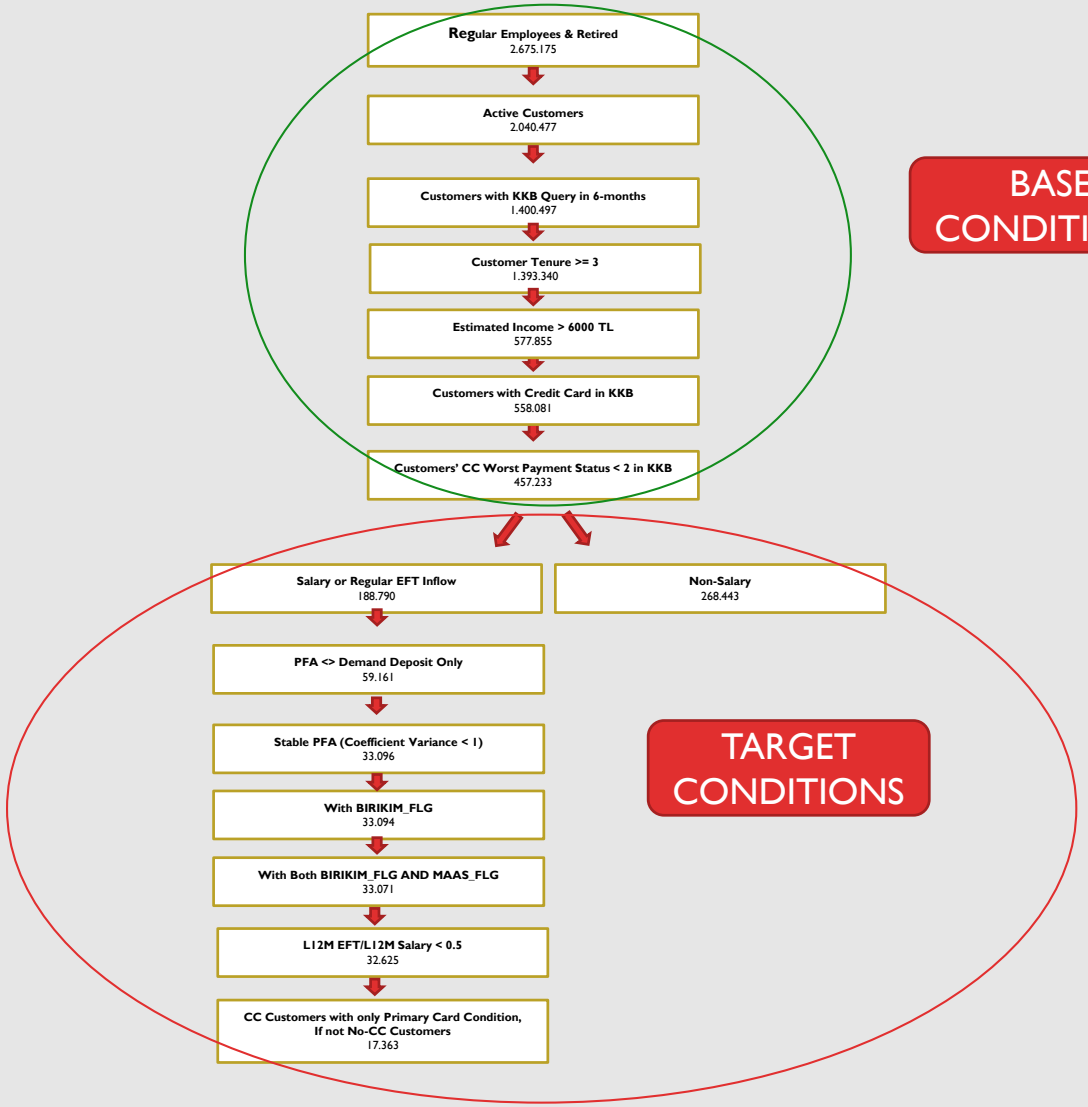
DEBT: In this case, Credit card and Credits. **(-)**

- Review periods: 2017-12, 2018-06, 2018-11
- Definition of target customers that they are known, loyal customers
- At least one product is necessary to obtain whether the customer is active or not for our bank
- Two segments: Employed Retired and Self Employed
- 12-monthly review was made based on each period
- removing some customers that they have only demand deposit source, but if a customer has more than or equal to 1 million TRY, keeping this customer.
- customers with 6 distinct month payment in last 12 months
- customers with at least 1-month payment within last 2 months

EMPLOYED AND RETIRED SEGMENT



SELF-EMPLOYED SEGMENT



MAAS_AY_CNT>10 with maas_flg

Sum of TARGET_FLAG	Estimated Income									
Customer Age	**6K-7K**	**7K-8K**	**8K-9K**	**9K-10K**	**10K-12.5K**	**12.5K-15K**	**15K-17.5K**	**17.5K-20K**	**20K+**	**Grand Total**
18_25	472	258	126	57	66	18	13	7	6	1023
26_35	6844	4912	3611	2854	4863	2015	768	410	1354	27631
36_50	7723	5896	4829	3921	7398	5289	3340	1951	4428	44775
50P	1634	1422	1144	971	1943	1318	883	641	2144	12100
Grand Total	**16673**	**12488**	**9710**	**7803**	**14270**	**8640**	**5004**	**3009**	**7932**	**85529**

MAAS_AY_CNT>10 without maas_flg

Sum of TARGET_FLAG	Estimated Income									
Customer Age	**6K-7K**	**7K-8K**	**8K-9K**	**9K-10K**	**10K-12.5K**	**12.5K-15K**	**15K-17.5K**	**17.5K-20K**	**20K+**	**Grand Total**
18_25	472	258	126	57	66	18	13	7	6	1023
26_35	6851	4917	3612	2856	4863	2017	768	413	1354	27651
36_50	7731	5902	4833	3926	7410	5293	3342	1954	4429	44820
50P	1638	1422	1145	972	1945	1319	883	641	2147	12112
Grand Total	**16692**	**12499**	**9716**	**7811**	**14284**	**8647**	**5006**	**3015**	**7936**	**85606**

MAAS_AY_CNT>6 with maas_flg

Sum of TARGET_FLAG	Column Labels									
Row Labels	**4.6K-7K**	**5.7K-8K**	**6.8K-9K**	**7.9K-10K**	**8.10K-12.5K**	**90.12.5K-15K**	**91.15K-17.5K**	**92.17.5K-20K**	**93.20K+**	**Grand Total**
0.18_25	550	284	138	63	72	18	13	8	10	1156
1.26_35	7766	5526	3977	3131	5207	2155	827	441	1411	30441
2.36_50	8636	6576	5310	4318	8141	5793	3629	2101	4742	49246
50P	1817	1544	1262	1065	2115	1451	974	697	2302	13227
Grand Total	**18769**	**13930**	**10687**	**8577**	**15535**	**9417**	**5443**	**3247**	**8465**	**94070**

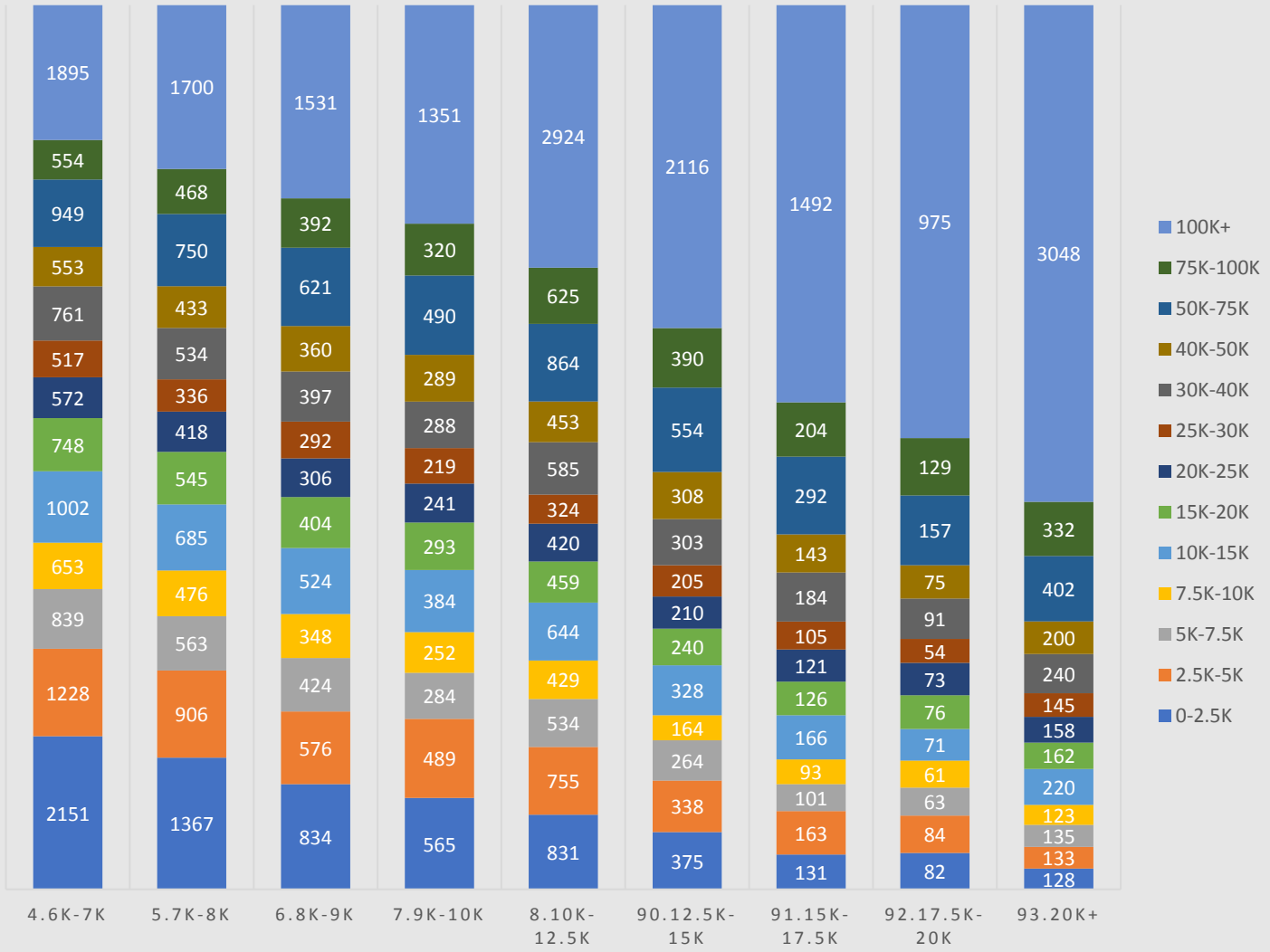
MAAS_AY_CNT>6 without maas_flg

Sum of TARGET_FLAG	Column Labels									
Row Labels	**6K-7K**	**7K-8K**	**8K-9K**	**9K-10K**	**10K-12.5K**	**12.5K-15K**	**15K-17.5K**	**17.5K-20K**	**20K+**	**Grand Total**
18_25	550	284	138	63	72	18	13	8	10	1156
26_35	7776	5533	3978	3133	5209	2158	827	444	1411	30469
36_50	8646	6584	5314	4324	8155	5802	3633	2104	4743	49305
50P	1821	1545	1264	1066	2117	1452	974	697	2307	13243
Grand Total	**18793**	**13946**	**10694**	**8586**	**15553**	**9430**	**5447**	**3253**	**8471**	**94173**

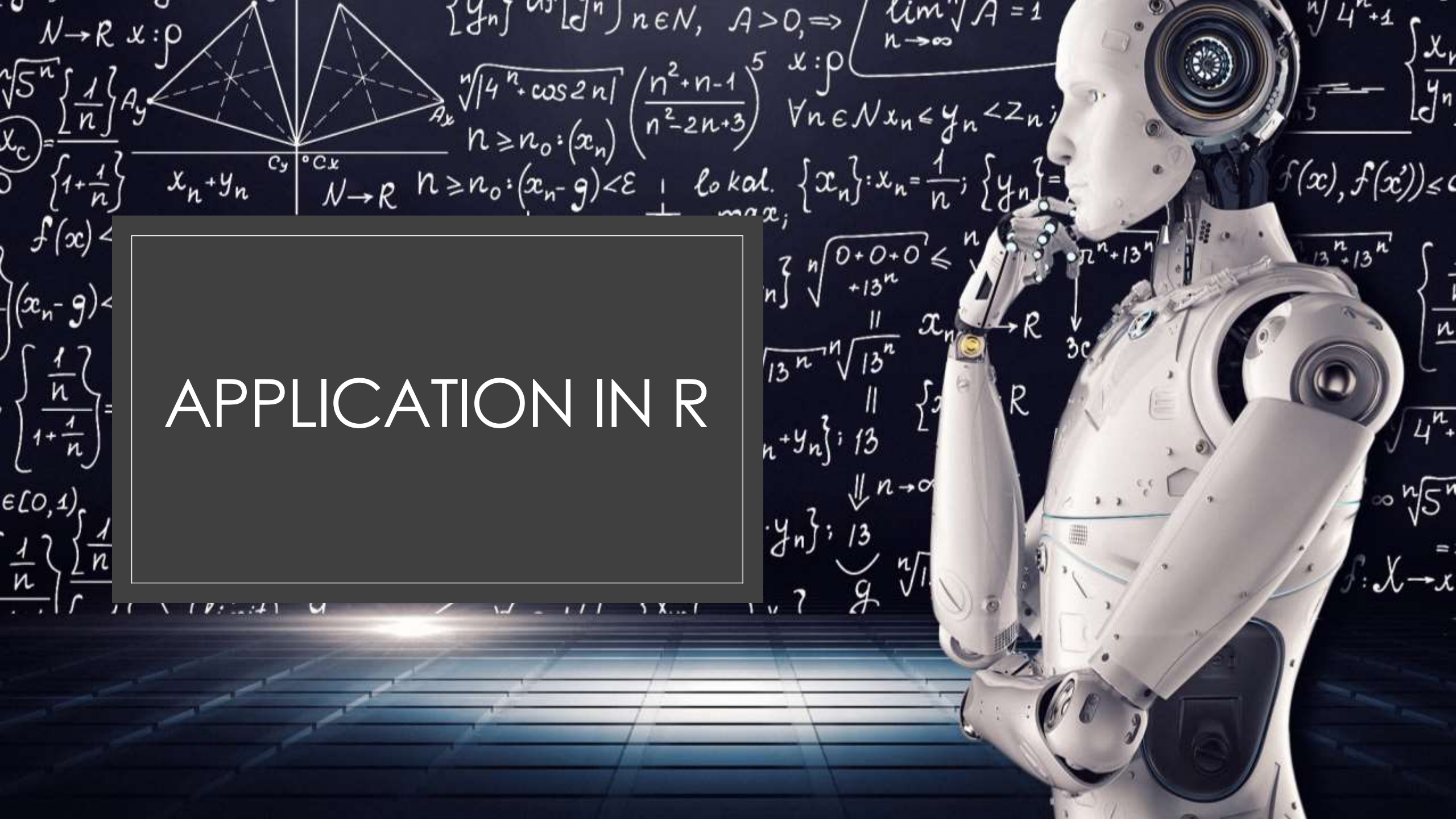
Sum of TARGET_FLAG	ESTIMATED INCOME								
CUST_AGE	6K-7K	7K-8K	8K-9K	9K-10K	10K-12.5K	12.5K-15K	15K-17.5K	17.5K-20K	20K+
18_25	47.58%	24.57%	11.94%	5.45%	6.23%	1.56%	1.12%	0.69%	0.87%
26_35	25.51%	18.15%	13.06%	10.29%	17.11%	7.08%	2.72%	1.45%	4.64%
36_50	17.54%	13.35%	10.78%	8.77%	16.53%	11.76%	7.37%	4.27%	9.63%
50P	13.74%	11.67%	9.54%	8.05%	15.99%	10.97%	7.36%	5.27%	17.40%
Grand Total	**19.95%**	**14.81%**	**11.36%**	**9.12%**	**16.51%**	**10.01%**	**5.79%**	**3.45%**	**9.00%**

TARGET_FLAG1	PFA Bands												
Estimated Income	0-2.5K (%)	2.5K-5K (%)	5K-7.5K (%)	7.5K-10K (%)	10K-15K (%)	15K-20K (%)	20K-25K (%)	25K-30K (%)	30K-40K (%)	40K-50K (%)	50K-75K (%)	75K-100K (%)	100K+ (%)
6K-7K	17.32	9.89	6.75	5.26	8.07	6.02	4.60	4.16	6.13	4.45	7.64	4.46	15.26
7K-8K	14.89	9.87	6.13	5.18	7.46	5.94	4.55	3.66	5.82	4.72	8.17	5.10	18.52
8K-9K	11.90	8.22	6.05	4.97	7.48	5.76	4.37	4.17	5.66	5.14	8.86	5.59	21.84
9K-10K	10.34	8.95	5.20	4.61	7.03	5.36	4.41	4.01	5.27	5.29	8.97	5.86	24.72
10K-12.5K	8.44	7.67	5.42	4.36	6.54	4.66	4.27	3.29	5.94	4.60	8.77	6.35	29.69
12.5K-15K	6.47	5.83	4.56	2.83	5.66	4.14	3.62	3.54	5.23	5.31	9.56	6.73	36.51
15K-17.5K	3.94	4.91	3.04	2.80	5.00	3.79	3.64	3.16	5.54	4.31	8.79	6.14	44.93
17.5K-20K	4.12	4.22	3.16	3.06	3.57	3.82	3.67	2.71	4.57	3.77	7.89	6.48	48.97
20K+	2.36	2.45	2.49	2.27	4.05	2.99	2.91	2.67	4.42	3.69	7.41	6.12	56.17
Grand Total	**10.69**	**7.73**	**5.30**	**4.30**	**6.66**	**5.05**	**4.17**	**3.63**	**5.60**	**4.65**	**8.40**	**5.65**	**28.17**

INCOME-PFA BASED ON ASSET BASE TABLE



APPLICATION IN R



Packages

data.table

xgboost

RJDBC

sqldf

dplyr

openxlsx

h2o

ggplot2

matrix

corrplot

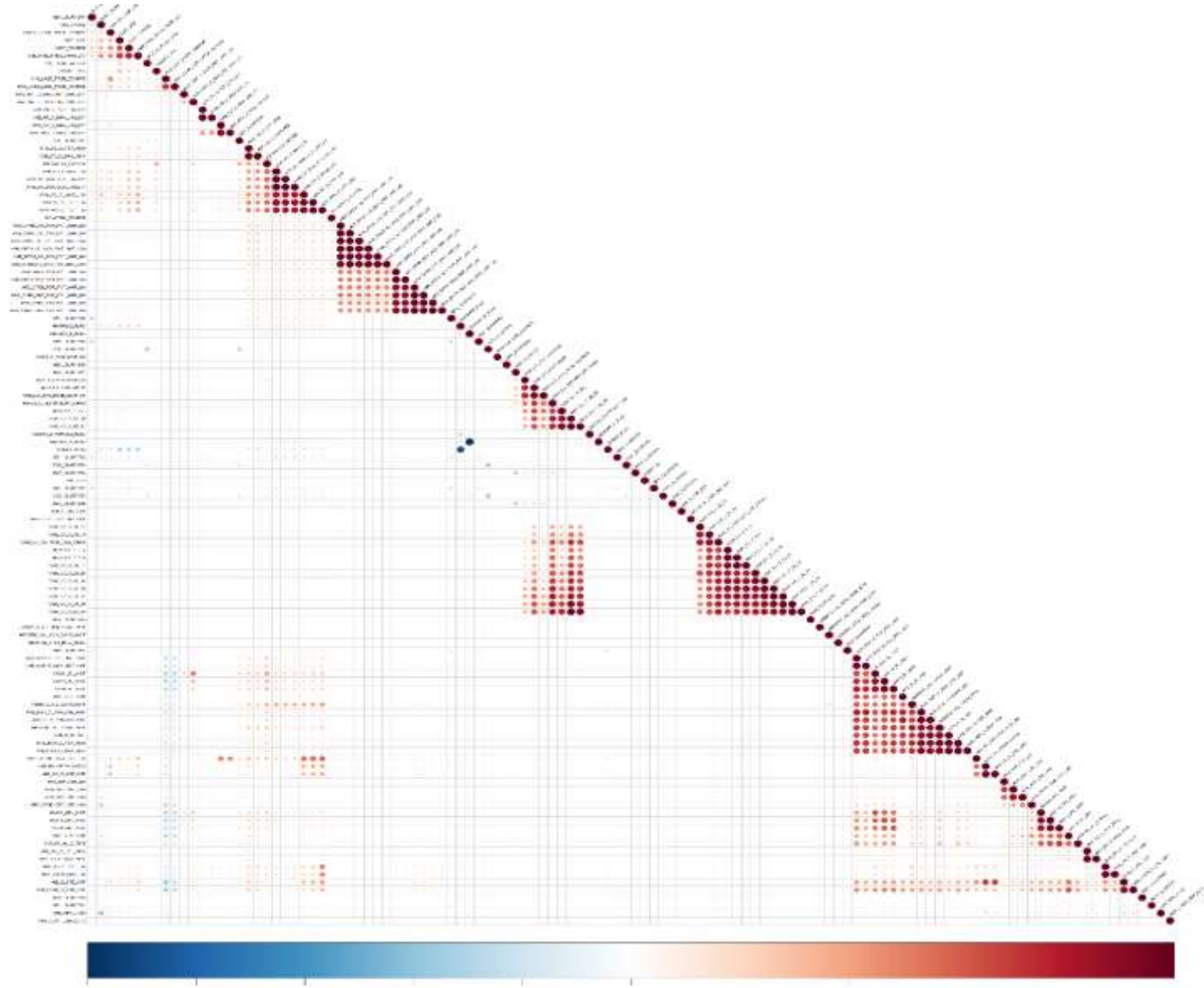
CORRELATION ANALYSIS

Variables that lead to zero standard deviation
"TARGET_FLAG" ,
"TARGET_FLAG2" ,
"KKB_OPEN_NREV_MAX_PMT_AMT_LM",
"KKB_OPEN_NREV_TOT_PMT_AMT_LM",
"GENDER_NULL_FLAG" ,
"KKB_C_CNT_DEF_DL36" ,
"KKB_OPEN_NREV_AVG_PMT_AMT_LM",
"KKB_O_PL_LAST_INS_AMT",
"BEH_CLUSTER0",
"MAH_CLUSTER0",
"MAH_CLUSTER2"

$$r = r_{xy} = \frac{cov(x, y)}{S_x * S_y}$$

Pearson's Coefficient

Size of Correlation	Interpretation
.90 to 1.00 (−.90 to −1.00)	Very high positive (negative) correlation
.70 to .90 (−.70 to −.90)	High positive (negative) correlation
.50 to .70 (−.50 to −.70)	Moderate positive (negative) correlation
.30 to .50 (−.30 to −.50)	Low positive (negative) correlation
.00 to .30 (.00 to −.30)	negligible correlation



ONE HOT ENCODING AND GRID SEARCH ALGORITHM

```
#####################################################################
####   Create the (sparse) model matrices for all samples    #####
#####################################################################

#strain <- sparse.model.matrix(TARGET_PFA ~ .- 1  , data = modelDevSample)

model_formula <- as.formula(TARGET_PFA ~ . - 1) #TARGET'I ÇIKARMAK İÇİN

sparse_train <- sparse.model.matrix(model_formula, data = modelDevSample)
label_train <- modelDevSample$TARGET_PFA
dense_train <- xgb.DMatrix(data = sparse_train,  label = label_train)

sparse_test <- sparse.model.matrix(model_formula, data = modelTestSample)
label_test <- modelTestSample$TARGET_PFA
dense_test <- xgb.DMatrix(data = sparse_test, label = label_test)


#bst <- xgboost(data = dense_train, max_depth = 2, nthread =2, nrounds = 2, verbose = 0) ->> verbose

class(sparse_train) #dgCMatrix
dim(sparse_train) #dimension
head(sparse_train)
# ilk önce sparce matrix yapısına göre değişkenler düzenlenir, daha sonrasında
# bu düzenlenen değişkenlerin label'i belirlenir yani target değişkeni..
# daha sonrasında Dmatrix de bu matris biçimi model biçimi olarak birleştirilir.

searchGridSubCol <- expand.grid(gamma = c(0,1), #default value set to 0.
                                max_depth = c(6,5), #!!!! the default value is set to 6.
                                lambda = c(0.2,0.4, 0.5),
                                eta = c(0.1,0.2,0.3) #default value is set to 0.3
)

ntrees <- 200
```

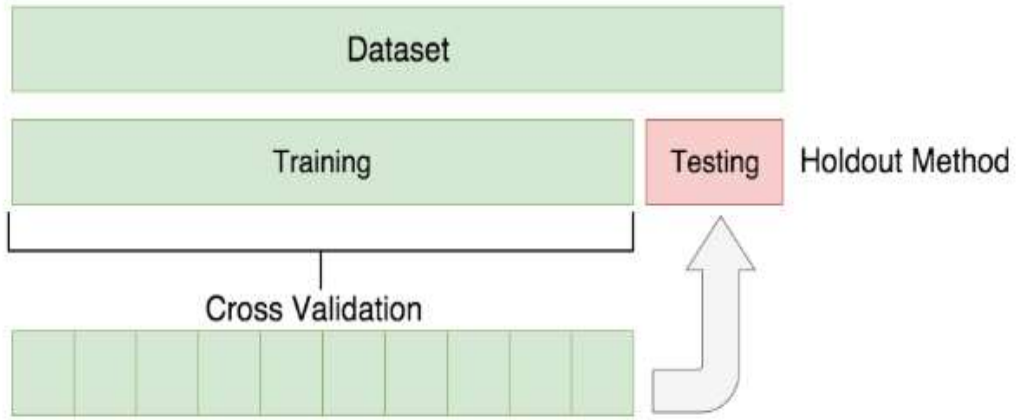
GRID SEARCH ALGORITHM FUNCTION

	gamma	max_depth	lambda	eta
1	0	6	0.2	0.1
2	1	6	0.2	0.1
3	0	5	0.2	0.1
4	1	5	0.2	0.1
5	0	6	0.4	0.1
6	1	6	0.4	0.1
7	0	5	0.4	0.1
8	1	5	0.4	0.1
9	0	6	0.5	0.1
10	1	6	0.5	0.1
11	0	5	0.5	0.1
12	1	5	0.5	0.1
13	0	6	0.2	0.2
14	1	6	0.2	0.2
15	0	5	0.2	0.2
16	1	5	0.2	0.2
17	0	6	0.4	0.2
18	1	6	0.4	0.2
19	0	5	0.4	0.2
20	1	5	0.4	0.2
21	0	6	0.5	0.2
22	1	6	0.5	0.2
23	0	5	0.5	0.2
24	1	5	0.5	0.2
25	0	6	0.2	0.3
26	1	6	0.2	0.3
27	0	5	0.2	0.3
28	1	5	0.2	0.3
29	0	6	0.4	0.3
30	1	6	0.4	0.3
31	0	5	0.4	0.3
32	1	5	0.4	0.3
33	0	6	0.5	0.3

Test RMSE	Train RMSE	currentGamma	currentLambda	currentDepth	currentEta
191862.7469	134563.6375	1	0.4	6	0.1
191909.0281	135150.0938	1	0.5	6	0.1
192954.6	111235.4812	1	0.5	6	0.2
192996.6188	134941.3094	1	0.2	6	0.1
193331.7219	134308.2469	0	0.4	6	0.1
193371.8406	110986.325	0	0.5	6	0.2
193418.7406	151562.5781	1	0.2	5	0.1
193508.275	152480.4031	0	0.4	5	0.1
193531.3906	133688.1344	0	0.2	6	0.1
193748.5906	135554.175	0	0.5	6	0.1
193754.1719	110486.9344	1	0.4	6	0.2
193795.0563	152847.125	0	0.5	5	0.1
194207.3688	152523.1156	1	0.4	5	0.1
194522.9313	110061.1375	1	0.2	6	0.2
194690.8063	133694.9531	0	0.5	5	0.2
194732.5031	151728.7188	0	0.2	5	0.1
194776.6531	133481.1781	1	0.4	5	0.2
194864.975	110233.4688	0	0.2	6	0.2
194952.025	133268.2063	0	0.4	5	0.2
195029.8719	153308.2813	1	0.5	5	0.1
195163.4438	110599.0719	0	0.4	6	0.2
196246.9813	133425.5438	1	0.5	5	0.2
196247.0938	132284.5281	0	0.2	5	0.2
196275.7281	131924.4031	1	0.2	5	0.2
196612.6594	118691.8219	1	0.5	5	0.3
196803.7344	120124.6094	0	0.5	5	0.3
197662.7375	118977.6984	0	0.4	5	0.3
197771.65	92786.7125	1	0.2	6	0.3
197774.5281	94370.97031	0	0.4	6	0.3
198032.4	93143.16875	0	0.2	6	0.3
198100.2063	93332.60156	1	0.4	6	0.3
198249.2719	119066.3359	1	0.4	5	0.3
198543.8094	94194.02344	1	0.5	6	0.3

GRID SEARCH OUTPUT OF COMBINATIONS

```r
rmseErrorsHyperparameters <- apply(searchGridSubCol, 1, function(parameterList){
  #Extract Parameters to test
  currentGamma <- parameterList[["gamma"]]
  currentDepth <- parameterList[["max_depth"]]
  currentLambda <- parameterList[["lambda"]]
  currentEta <- parameterList[["eta"]]
  xgboostModelCV <- xgb.cv(data =  dense_train,
                           nrounds = ntrees,
                           nfold = 5,
                           showsd = F,
                           print_every_n = 10, #her 10 tanede bir öğren, her 10 dallanmada öğren.
                           "eval_metric" = "rmse",
                           "objective"   = "reg:linear",
                           "booster" = "dart",
                           "gamma" = currentGamma,
                           "eta" = currentEta,
                           "lambda" = currentLambda,
                           "max.depth" = currentDepth,
                           "seed" = 123456,
                           maximize = TRUE)

  xvalidationScores <- as.data.table(xgboostModelCV$evaluation_log)
  rmse <- tail(xvalidationScores$test_rmse_mean, 1)
  trmse <- tail(xvalidationScores$train_rmse_mean,1)
  output <- c(rmse, trmse, currentGamma,currentLambda , currentDepth, currentEta)})
```




```
write.xlsx(output, file = "GridSearchAlgorithm.xlsx") #testin min olmadı

xgboost_params <- list(
  objective   = "reg:linear", #lineer regresyon
  booster="dart",
  eval_metric = "rmse"
)

mod_xgb_dart <- xgb.cv(
  params = xgboost_params,
  nrounds = 200,
  prediction = TRUE,
  data = dense_train,
  nfold = 10,
  showsd = FALSE,
  maximize = FALSE,
  early_stopping_rounds = 30,
  max.depth=6,
  gamma = 1,
  eta = 0.1,
  lambda = 0.4,
  print_every_n = 1
)
```

CROSS VALIDATION

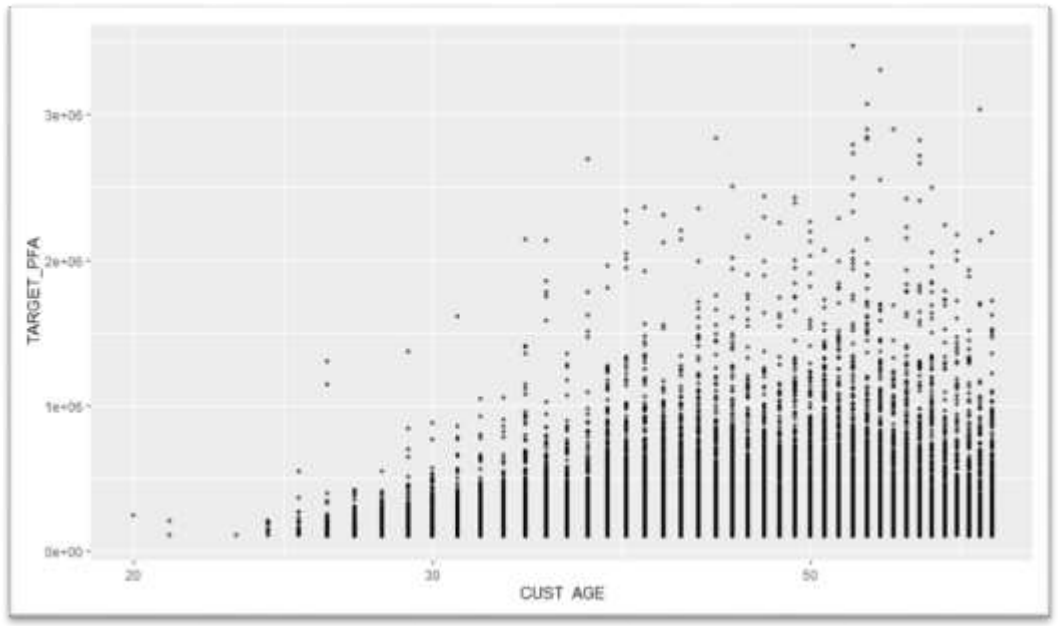
CROSS VALIDATION

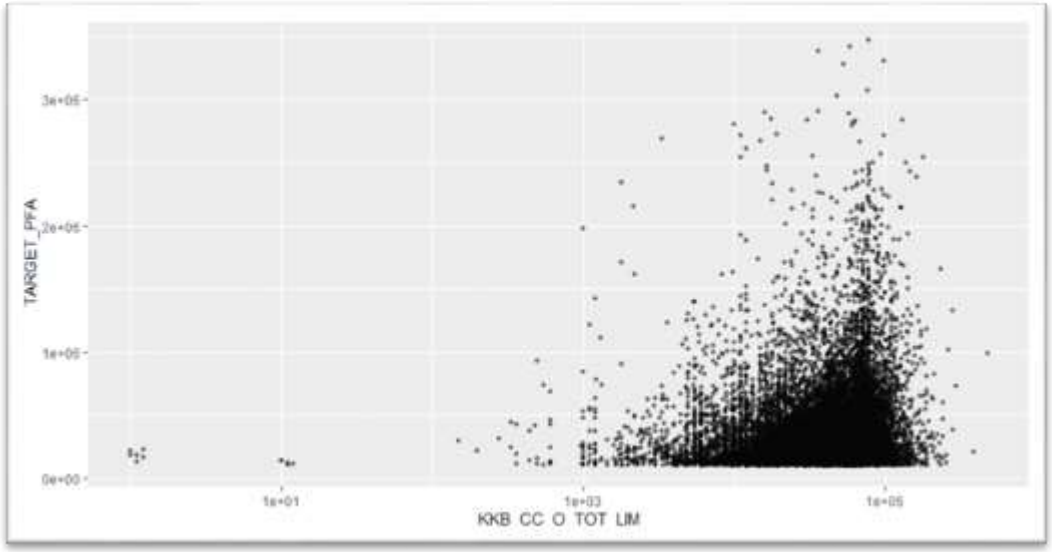
```r
best_iteration = mod_xgb_dart$best_iteration


mod_xgb_dart_best <- xgboost(
  params = xgboost_params,
  data = dense_train,
  nrounds = best_iteration,
  showsd = FALSE,
  maximize = FALSE,
  max.depth=6,
  gamma = 1,
  eta = 0.1,
  lambda = 0.4,
  print_every_n = 50
)

#importance_XGB  <- xgb.importance(feature_names = dimnames(sparse_train)[[2]],model = mod_xgb_dart_best)

importance <- xgb.importance(feature_names = sparse_train@Dimnames[[2]], model = mod_xgb_dart_best)
```

Best iteration is found after cross validation.

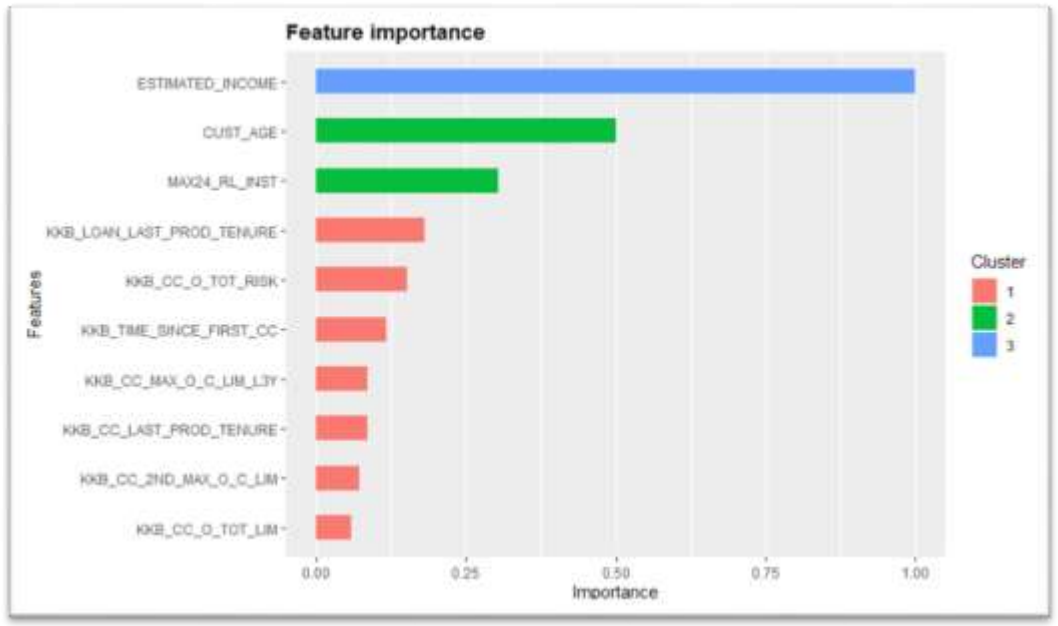
Again with GS result params, the code is run in every 50 rounds.

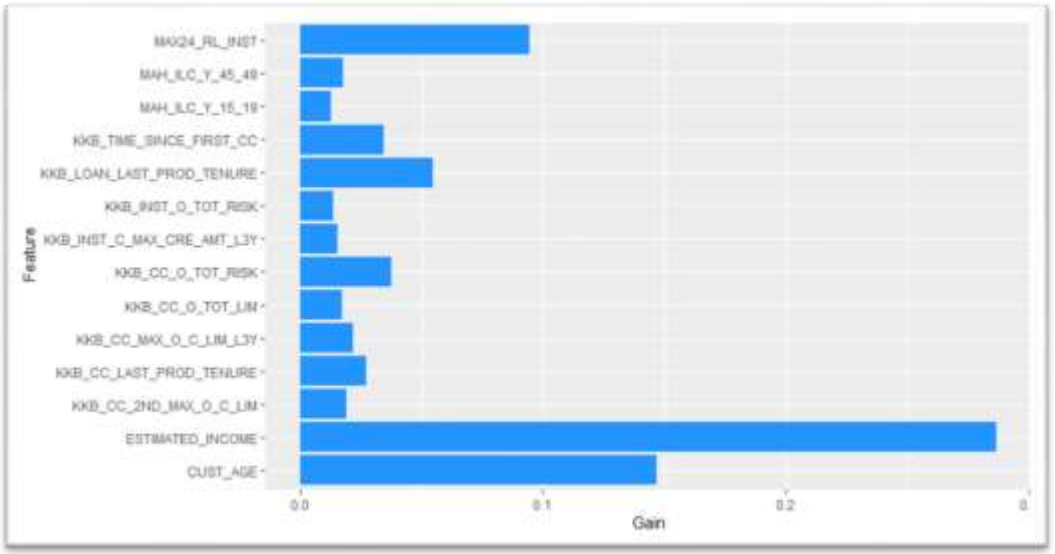
As a result, the best model is generated and importance features of the model is found with below code.





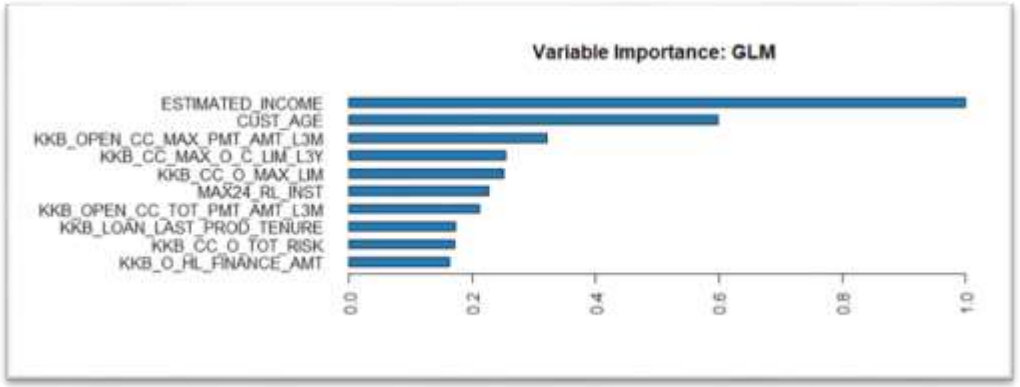
XGBOOST PREDICTION / FEATURES ANALYSIS

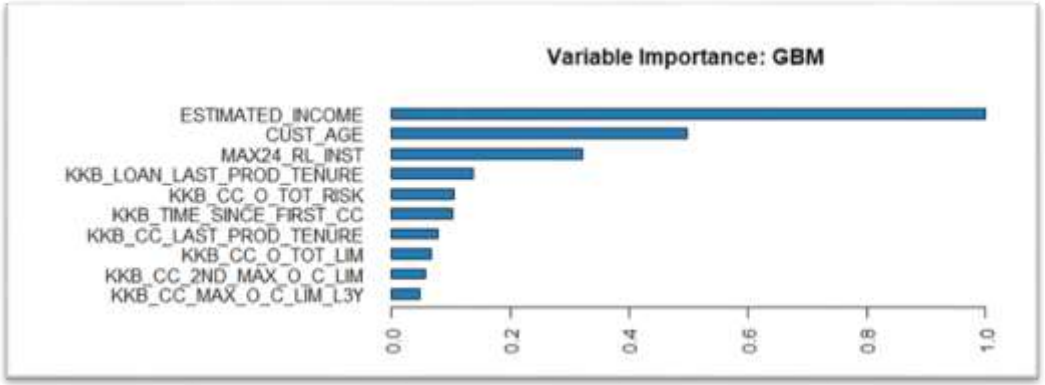


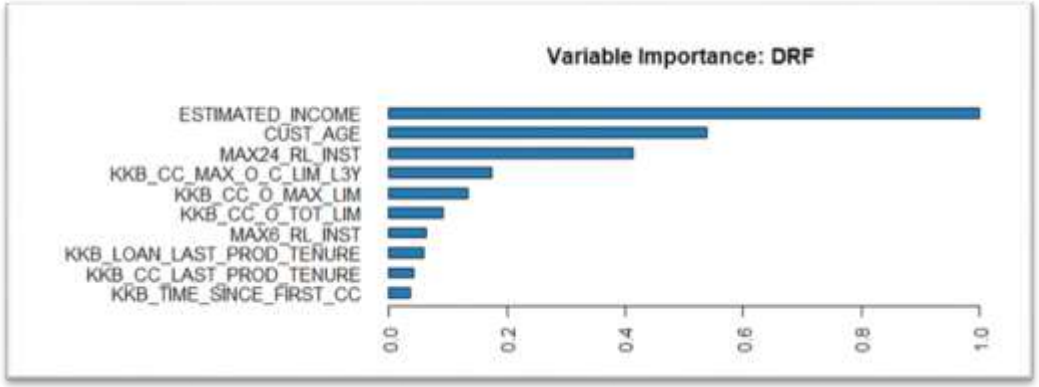


FEATURE IMPORTANCE FOR XGBOOST

FEATURE IMPORTANCE FOR GLM, GBM, RF







TARGET_PFA > 50,000

performance_fulllist_glm	Sample	Target_vs_Estimated_MAE	Target_vs_Estimated_MAPE
1	Train	146.974	56%
2	Test	153.162	58%

TARGET_PFA > 50,000

performance_fulllist_xgb	Sample	Target_vs_Estimated_MAE	Target_vs_Estimated_MAPE
1	Train	69.895	12%
2	Test	98.642	17%

TARGET_PFA > 50,000

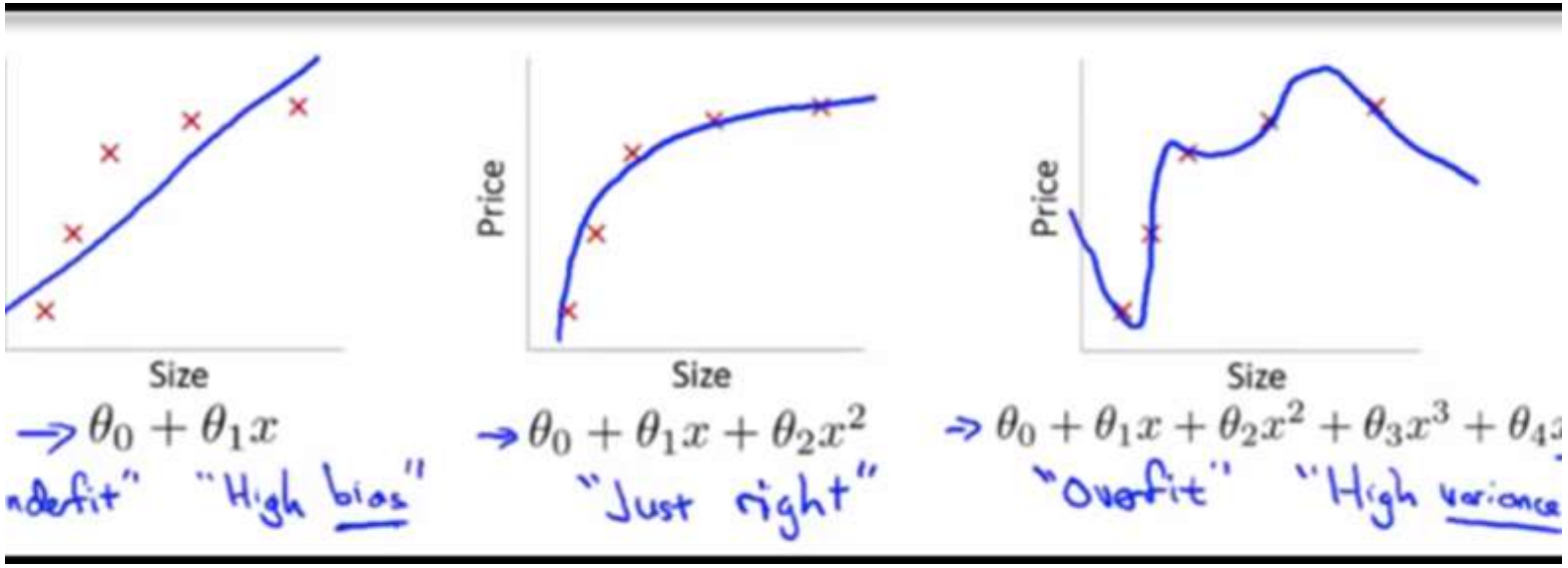
performance_fulllist_gbm	Sample	Target_vs_Estimated_MAE	Target_vs_Estimated_MAPE
1	Train	82.975	15%
2	Test	106.243	18%

TARGET_PFA > 50,000

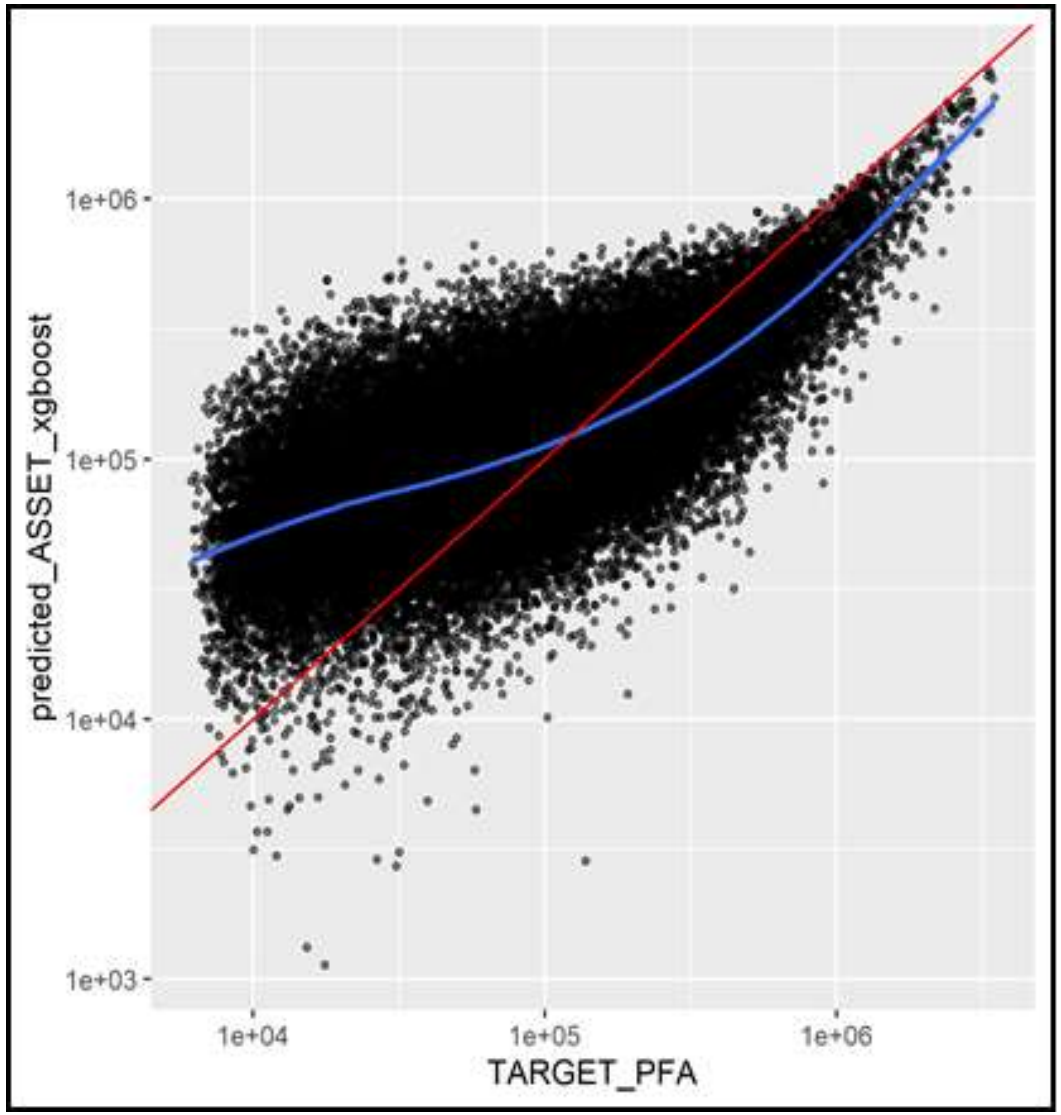
performance_fulllist_rf	Sample	Target_vs_Estimated_MAE	Target_vs_Estimated_MAPE
1	Train	137.478	66%
2	Test	141.874	69%

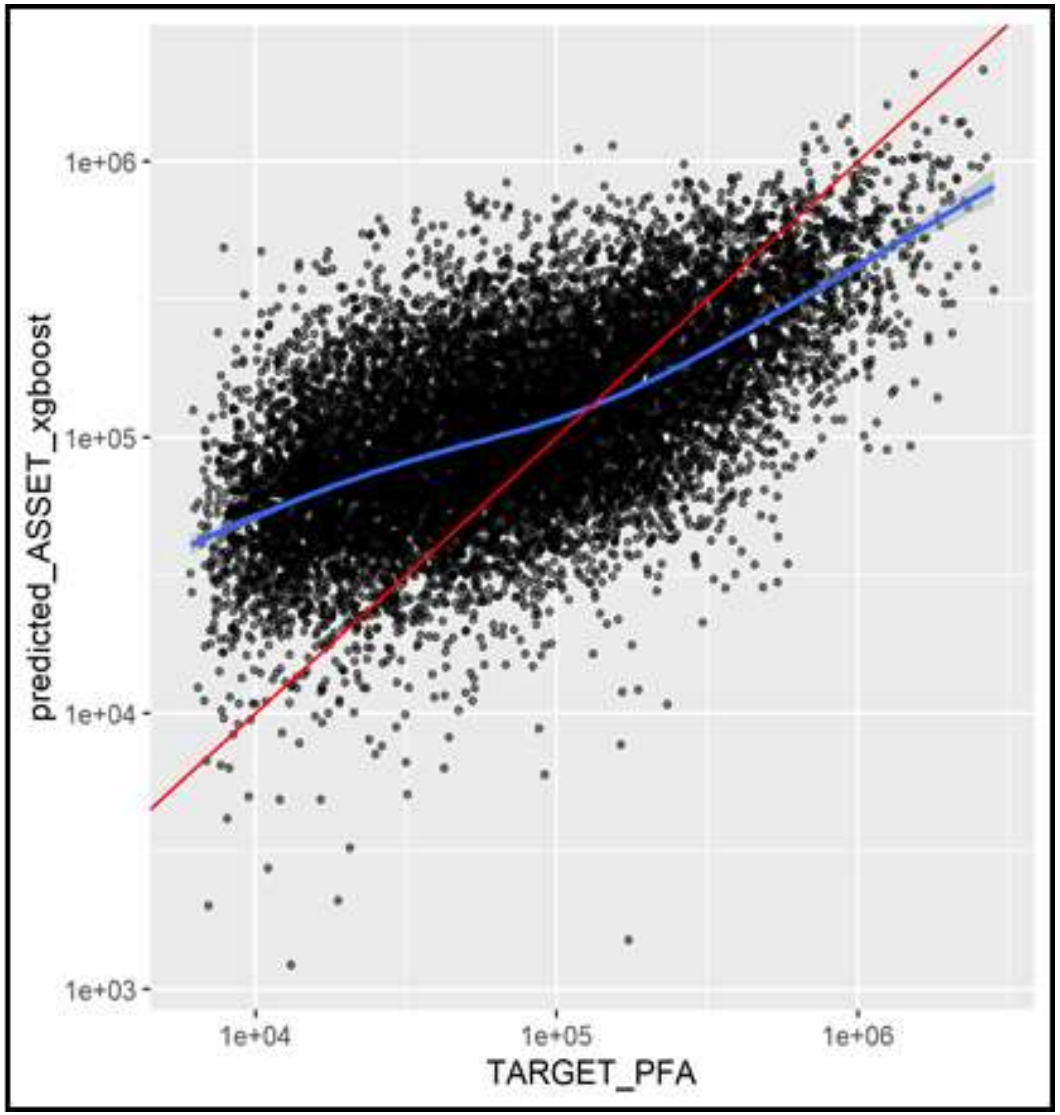
MODEL STATISTICS

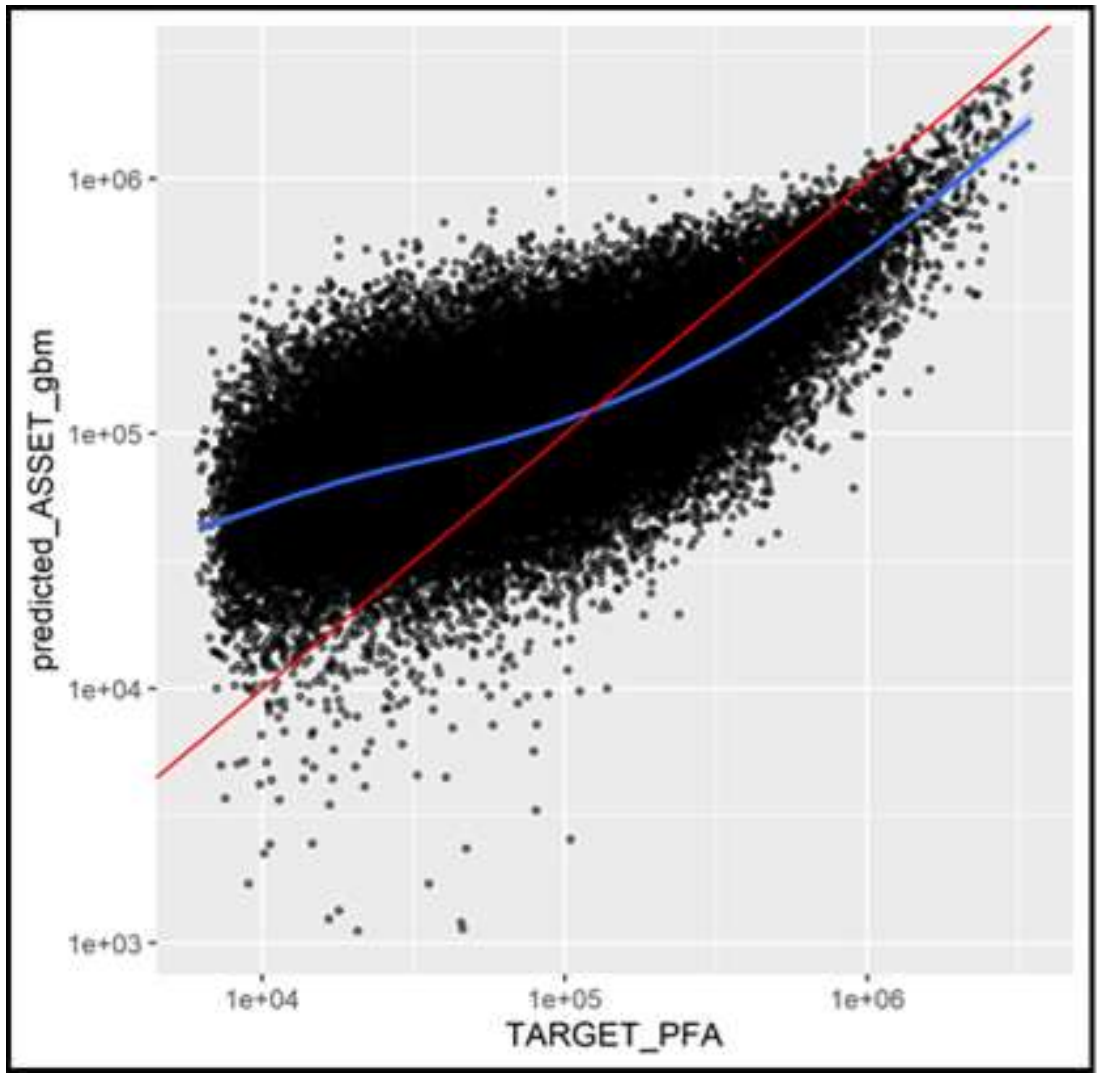
- TARGET_PFA amount was determined more than 50,000 quantities because the customers that has huge PFA amount is target group of the project. Quite a lot of products can be sold, and the right campaign is made to these target customers. Thanks to this, company wins. For this reason, the error rates (MAE and MAPE) between the real value and the estimated value of the customer groups with high PFA ratio were examined.

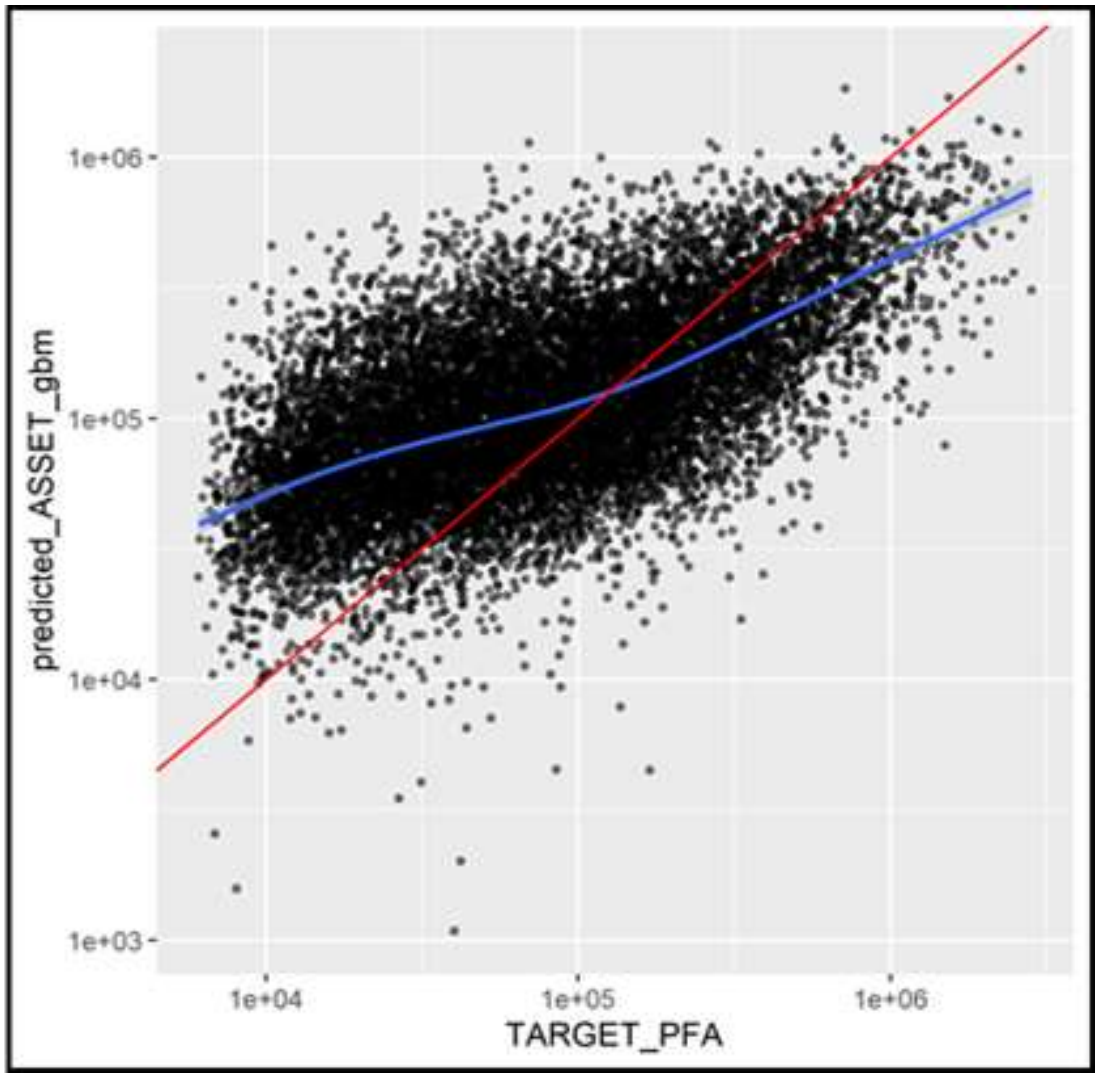


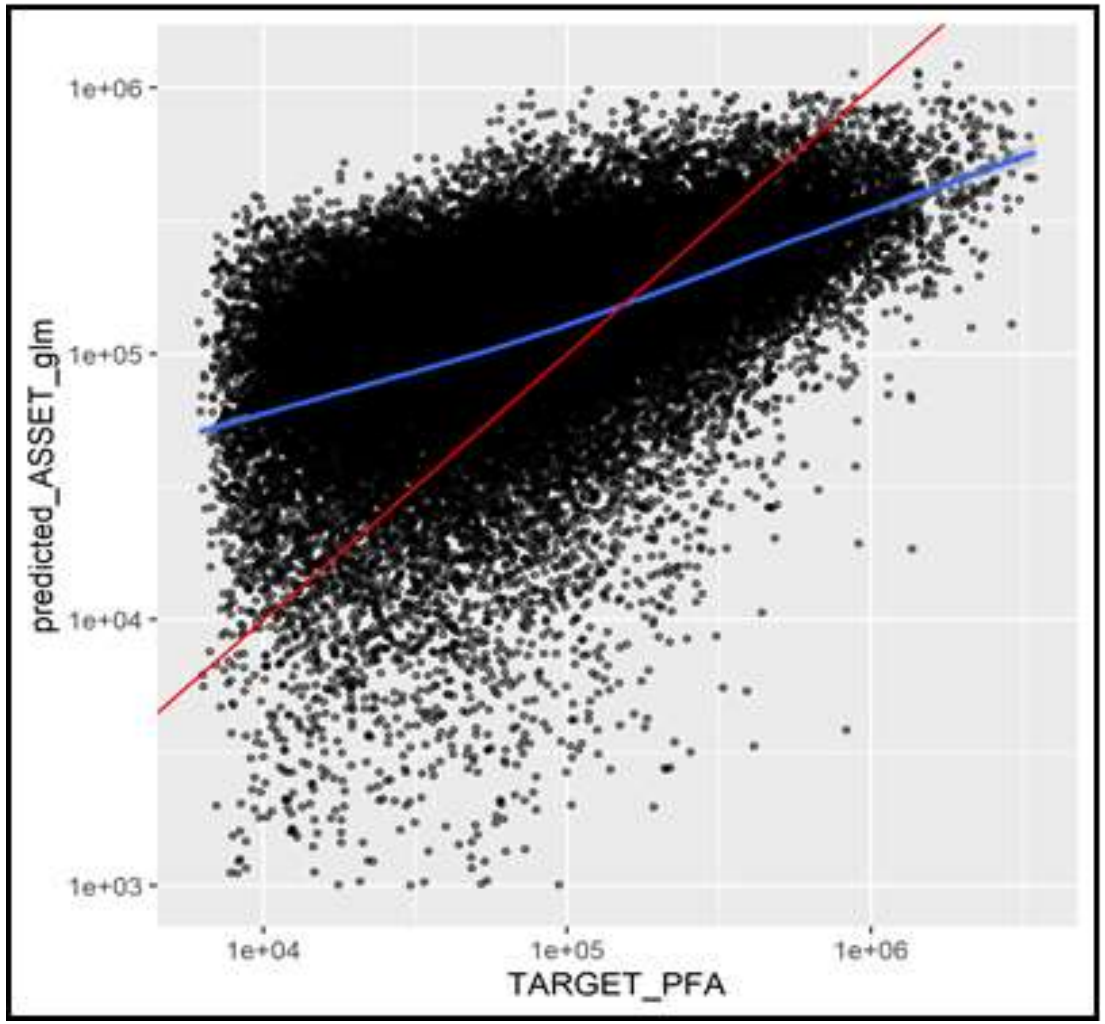
MODEL PERFORMANCE GRAPHS

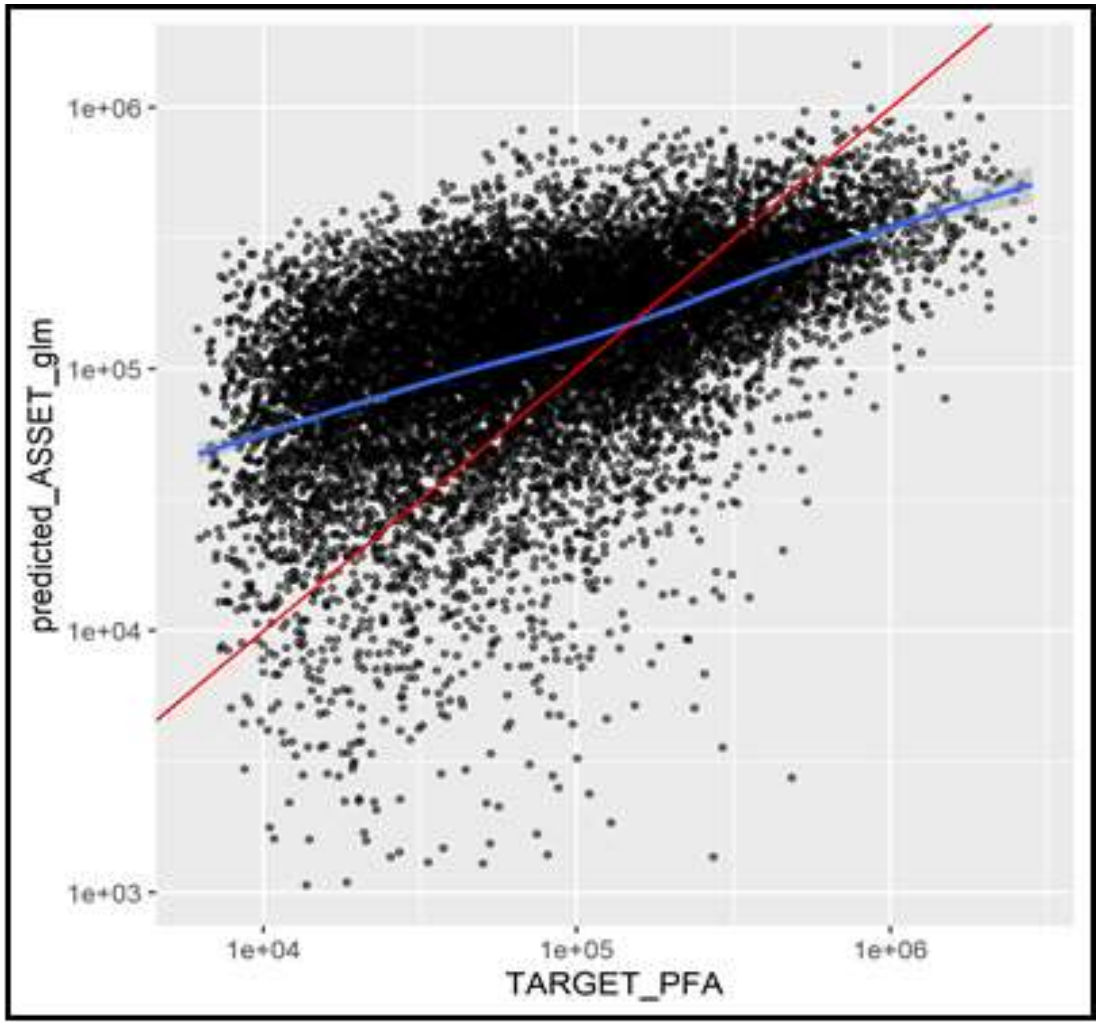

predicted_ASSET_xgboost
1e+06
1e+05
1e+04
1e+03
1e+04
1e+05
1e+06
TARGET_PFA

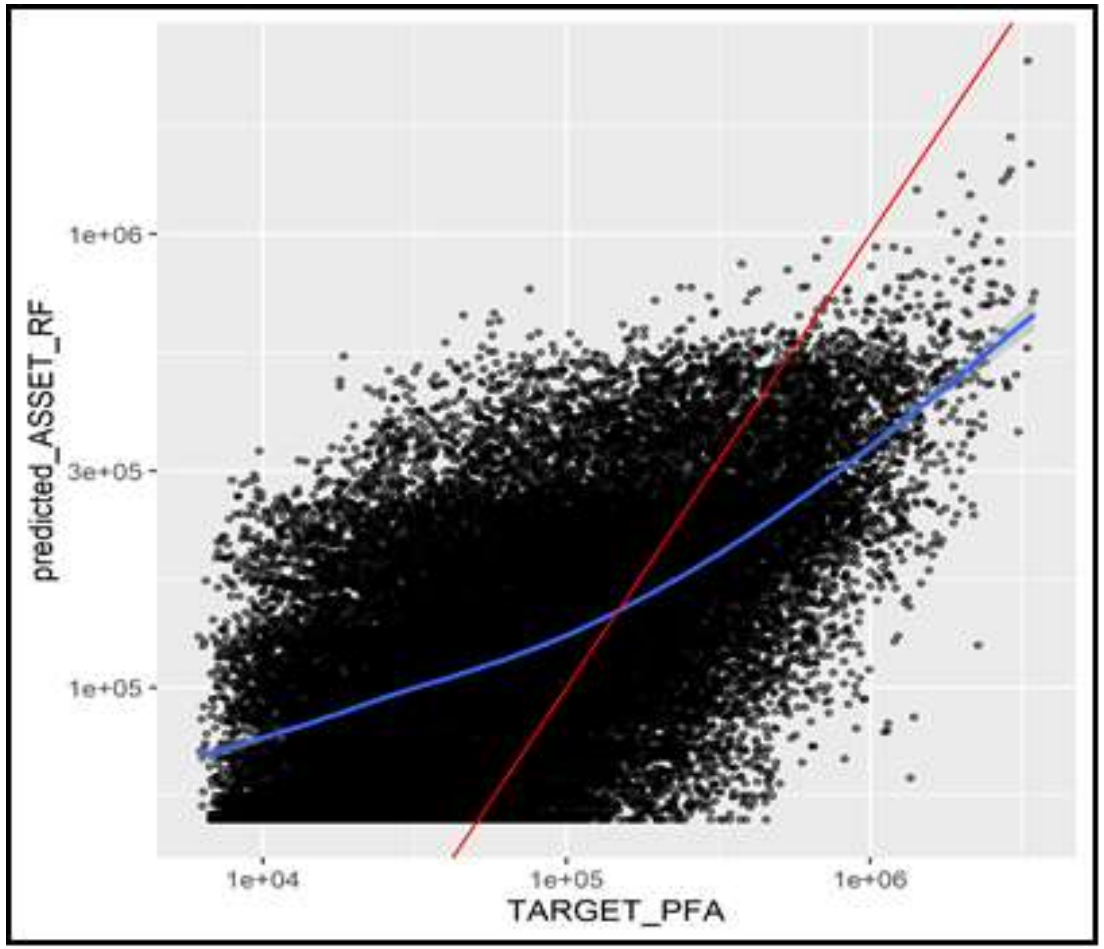

predicted_ASSET_xgboost
1e+06
1e+05
1e+04
1e+03
1e+04
1e+05
1e+06
TARGET_PFA

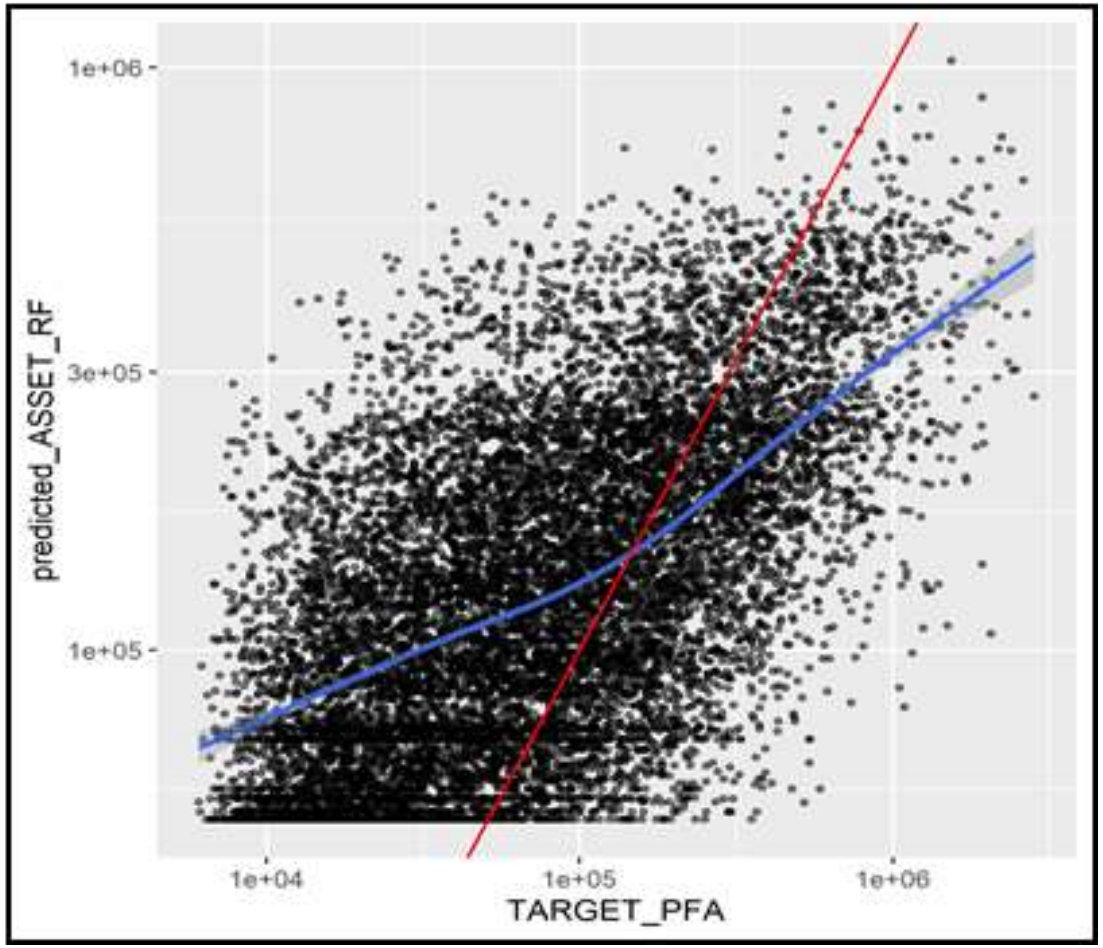

predicted_ASSET_gbm
1e+06
1e+05
1e+04
1e+03
1e+04
1e+05
1e+06
TARGET_PFA


predicted_ASSET_gbm
1e+06
1e+05
1e+04
1e+03
1e+04
1e+05
1e+06
TARGET_PFA


predicted_ASSET_glm
1e+06
1e+05
1e+04
1e+03
1e+04
1e+05
1e+06
TARGET_PFA


predicted_ASSET_glm
1e+06
1e+05
1e+04
1e+03
1e+04
1e+05
1e+06
TARGET_PFA


predicted_ASSET_RF
1e+06
3e+05
1e+05
1e+04
1e+05
1e+06
TARGET_PFA


predicted_ASSET_RF
1e+06
3e+05
1e+05
1e+04
1e+05
1e+06
TARGET_PFA

CONCLUSION

- In this thesis, four machine learning algorithms that were examined in the literature review header used to predict the asset of bank's customers according to many conditions created in Oracle SQL software. Model performance was measured with MAE and MAPE KPI forecast error types. After examination of MAE and MAPE values for test samples and also model performance graphs, the XGBoost algorithm was chosen as an optimal model for this case. Test MAE and MAPE values were found 98.642, 17% respectively.
- As a result, not all models but for wealthy customers, the XGBoost model predicted well. GBM model followed it with almost no difference. In other words, the aim of the project could be reached.