CIS 8392
Topics in Big Data Analytics

#Data Wrangling for Machine Learning

Yu-Kai Lin

Agenda

- What makes a "high performance" ML pipeline?
- R and H2O

[**Acknowledgments**] The materials in the following slides are based on the source(s) below:

- An Introduction to Machine Learning with R by Laurent Gatto
- H2O Tutorials
- Kaggle Competition In 30 Minutes: Predict Home Credit Default Risk With R by Matt Dancho

Prerequisites

```
install.packages(c("skimr", "recipes"))
library(skimr)
library(recipes)
library(stringr)
library(tidyverse)
```

Scope of this lecture

I assume that you understand:

- the intuitions behind some popular ML algorithms
- how to use some of these ML algorithms (either in R or in Python)
- the roles of training and testing data
- metrics for evaluating ML models (reference)

So, these won't be my focus in this lecture. Instead, I want to discuss:

- what are the criteria in high performance ML
- how to seamlessly put together a high performance ML pipeline
- H2O, an open source ML platform
- implementing deep learning and AutoML in H2O

= QUESTION =

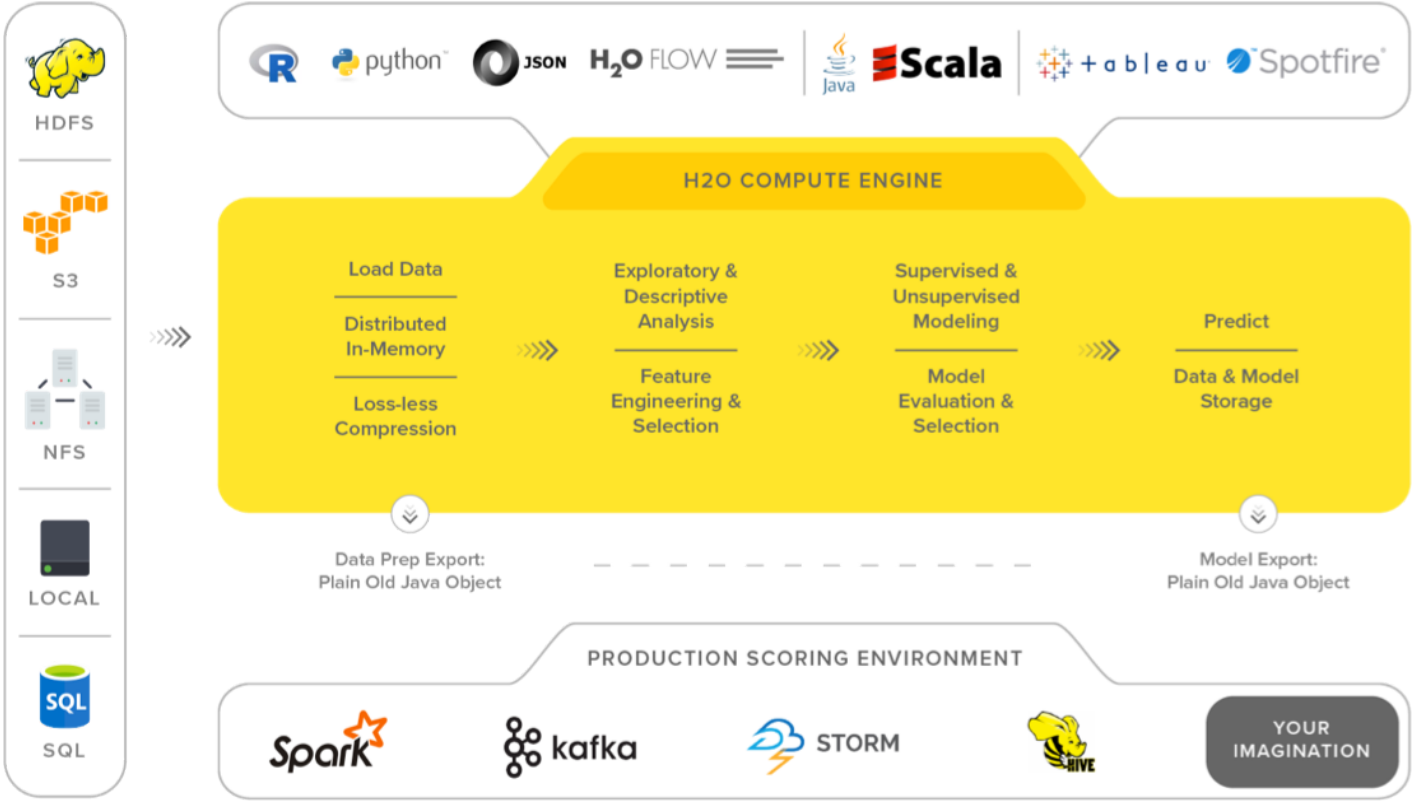
What are the criteria in high performance ML?

Criteria for high performance ML

- **Parallelization**: capable of using multiple cores on a local machine
- **Memory management**: efficiently store data in memory with loss-less compression
- **Salable**: run on local machine or a multi-node Spark/Hadoop cluster
- **Efficient implementation**: efficiently implement ML algorithms
- **Grid parameter search**: easily search and tune the (hyper-)parameters in a ML algorithm
- **Model selection**: easily construct and identify the best ML algorithm

H2O

H2O is the leading open source platform that makes it easy for enterprise to deploy AI and deep learning to solve complex problems.



H2O

H2O is popular:

- More than 9,000 organizations and 80,000+ data scientists depend on H2O for critical applications like predictive maintenance and operational intelligence.
- Used by 169 Fortune 500 enterprises, including 8 of the world's 10 largest banks, 7 of the 10 largest insurance companies, and 4 of the top 10 healthcare companies. See here for stories of how companies are using H2O.
- Notable customers include Capital One, Progressive Insurance, Transamerica, Comcast, Nielsen Catalina Solutions, Macy's, Walgreens, and Kaiser Permanente.

H2O

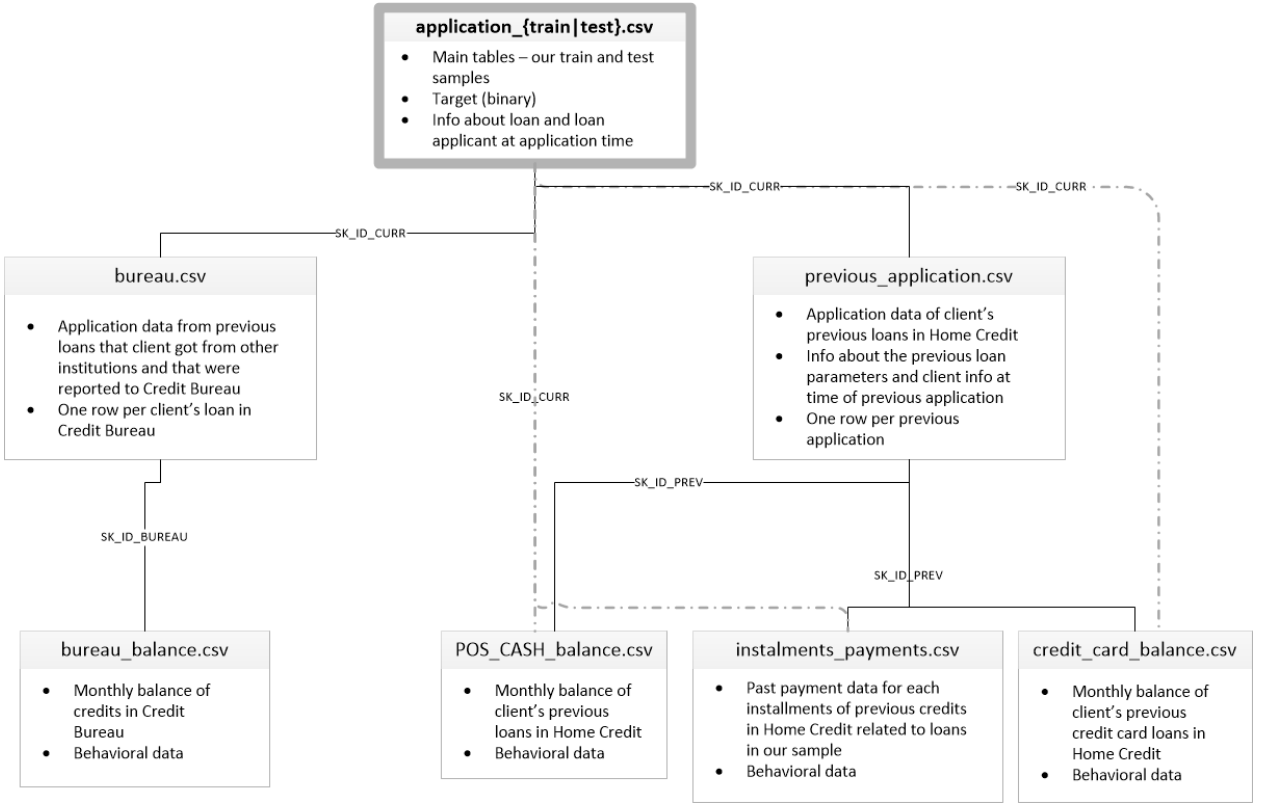
H2O is capable:

- Interfaces for R, Python, Scala, Java, JSON, and JavaScript
- Designed to run in standalone mode, on Hadoop, or within a Spark Cluster, and typically deploys within minutes.
- Supports the most widely used machine learning algorithms including random forest, gradient boosted machines, generalized linear models, deep learning and more.

Home Credit Default Risk dataset

https://www.kaggle.com/c/home-credit-default-risk/data

In this Kaggle Competition, the goal is to predict personal default risk. The diagram below shows the interrelationships between the data files provided.



What Is Credit Default Risk?

In the world of finance, Credit or Default Risk is the probability that companies or individuals will be unable to make the required payments on their debt obligations. Lenders are exposed to default risk on every extension of credit. Think of financial institutions like **Citi**, which lend to millions of people worldwide, or **JP Morgan Chase & Co**, which lend to hundreds of thousands of companies worldwide.

Every time Citi or JP Morgan extends a loan to a person or a corporation it assesses the level of risk involved in the transaction. This is because every once in a while people or companies won't be able to make payments.

For this lecture, we will only use the two main files, `application_train.csv` and `application_test.csv`, since they contain the most important information. You can download them from links below:

- https://www.dropbox.com/s/5yyni251q7ea3qy/application_test.csv?dl=0
- https://www.dropbox.com/s/ri0kpkjqy40nwz7/application_train.csv?dl=0

However, you can easily include other files in your analysis by using **dplyr**'s join functions.

Now, assume that our RStudio working directory is **C:/CIS8392/** and we put the csv files within the **C:/CIS8392/data/loan** folder:

```
library(tidyverse)
setwd("C:/CIS8392/")
application_train <- read_csv("data/loan/application_train.csv",
                              na = c("", NA, "-1"))
application_test <- read_csv("data/loan/application_test.csv",
                              na = c("", NA, "-1"))

View(application_train)
View(application_test)
```

```
View(application_train) # see the training data
```

SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN
100002	1	Cash loans	M	N	Y	0
100003	0	Cash loans	F	N	N	0
100004	0	Revolving loans	M	Y	Y	0
100006	0	Cash loans	F	N	Y	0
100007	0	Cash loans	M	N	Y	0
100008	0	Cash loans	M	N	Y	0
100009	0	Cash loans	F	Y	Y	1
100010	0	Cash loans	M	Y	Y	0
100011	0	Cash loans	F	N	Y	0
100012	0	Revolving loans	M	N	Y	0

```
View(application_test) # see the test data
```

SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INC
100001	Cash loans	F	N	Y	0	
100005	Cash loans	M	N	Y	0	
100013	Cash loans	M	Y	Y	0	
100028	Cash loans	F	N	Y	2	
100038	Cash loans	M	Y	N	1	
100042	Cash loans	F	Y	Y	0	
100057	Cash loans	M	Y	Y	2	
100065	Cash loans	M	N	Y	0	
100066	Cash loans	F	N	Y	0	
100067	Cash loans	F	Y	Y	1	

Summary of observations

1. The training data contains 307K observations, which are people applied for and received loans
2. The `TARGET` column identifies whether or not the person defaulted
3. The remaining 121 features describe various attributes about the person, loan, and application
4. The testing data (~48K observations) does not have the `TARGET` column
5. There are many missing values in there, shown as `NA`
6. Many columns have character values, but they look like categorical values (**factor**)
7. Many columns have very few distinct numeric values, which also suggest that they are likely categorical values (**factor**)

Better organize the data

In supervised ML, it is customary to name the outcome variable `y` and the predictor variables `x`. Always name and organize your data in the following way so that it is easy to understand and reusable in different ML projects:

```
# Training data: Separate into x and y tibbles
x_train_tbl <- application_train %>% select(-TARGET)
y_train_tbl <- application_train %>% select(TARGET)

# Testing data
x_test_tbl  <- application_test

# Remove the original data to save memory
rm(application_train)
rm(application_test)
```

Further data inspection

```
#install.packages("skimr")
library(skimr)
x_train_tbl_skim = partition(skim(x_train_tbl))
names(x_train_tbl_skim)
```

```
## [1] "character" "numeric"
```

```
x_train_tbl_skim$character
```

```
## # A tibble: 16 x 8
##    skim_variable   n_missing complete_rate   min   max empty n_unique whitespace
##    <chr>               <int>         <dbl> <int> <int> <int>    <int>      <int>
##  1 NAME_CONTRACT_~         0         1        10    15     0        2          0
##  2 CODE_GENDER             0         1         1     3     0        3          0
##  3 FLAG_OWN_CAR            0         1         1     1     0        2          0
##  4 FLAG_OWN_REALTY         0         1         1     1     0        2          0
##  5 NAME_TYPE_SUITE      2203         0.994     6    15     0        7          0
##  6 NAME_INCOME_TY~         0         1         7    20     0        8          0
##  7 NAME_EDUCATION~         0         1        15    29     0        5          0
##  8 NAME_FAMILY_ST~         0         1         5    20     0        6          0
##  9 NAME_HOUSING_T~         0         1        12    19     0        6          0
## 10 OCCUPATION_TYPE    111996         0.686     7    21     0       18          0
## 11 WEEKDAY_APPR_P~         0         1         6     9     0        7          0
## 12 ORGANIZATION_T~         0         1         3    22     0       58          0
## 13 FONDKAPREMONT_~    243092         0.318    13    21     0        4          0
## 14 HOUSETYPE_MODE     177916         0.501    14    16     0        3          0
## 15 WALLSMATERIAL_~    180234         0.494     5    12     0        7          0
## 16 EMERGENCYSTATE~    167964         0.529     2     3     0        2          0
```

```
x_train_tbl_skim$numeric %>% tibble()
```

```
## # A tibble: 105 x 11
##    skim_variable    n_missing complete_rate      mean       sd       p0      p25
##    <chr>                <int>         <dbl>     <dbl>    <dbl>    <dbl>    <dbl>
##  1 SK_ID_CURR               0         1       2.78e+5  1.03e+5  1.00e+5  1.89e+5
##  2 CNT_CHILDREN             0         1       4.14e-1  7.20e-1  0        0
##  3 AMT_INCOME_TOTAL         0         1       1.70e+5  2.24e+5  2.56e+4  1.12e+5
##  4 AMT_CREDIT               0         1       5.88e+5  3.99e+5  4.5 e+4  2.7 e+5
##  5 AMT_ANNUITY             36         1.00    2.74e+4  1.47e+4  1.62e+3  1.67e+4
##  6 AMT_GOODS_PRICE        278         0.999   5.28e+5  3.66e+5  4.05e+4  2.34e+5
##  7 REGION_POPULATI~         0         1       2.09e-2  1.39e-2  2.53e-4  1.00e-2
##  8 DAYS_BIRTH               0         1      -1.60e+4  4.36e+3 -2.52e+4 -1.97e+4
##  9 DAYS_EMPLOYED            0         1       6.43e+4  1.42e+5 -1.79e+4 -2.78e+3
## 10 DAYS_REGISTRATI~         0         1      -4.98e+3  3.53e+3 -2.47e+4 -7.48e+3
## # ... with 95 more rows, and 4 more variables: p50 <dbl>, p75 <dbl>,
## #   p100 <dbl>, hist <chr>
```

Data wrangling for ML

We see that there are 2 data type formats: numeric and character. For H2O we need **numeric** and **factor**.

The main things to focus on are:

1. **Character data that needs to be factored**: H2O requires any character data to be converted to a factor.

2. **Numeric data that has a low number of unique counts**: This is common in business data where there are numeric features that have low number of unique values. These are typically categorical data.

3. **Missing values**: Missing values need to be either imputed or to have the columns or rows dropped.

Character data

First, we collected the names of all character data. These columns will be converted into factors.

```
string_2_factor_names <- x_train_tbl_skim$character$skim_variable

string_2_factor_names
```

```
##  [1] "NAME_CONTRACT_TYPE"          "CODE_GENDER"
##  [3] "FLAG_OWN_CAR"                "FLAG_OWN_REALTY"
##  [5] "NAME_TYPE_SUITE"             "NAME_INCOME_TYPE"
##  [7] "NAME_EDUCATION_TYPE"         "NAME_FAMILY_STATUS"
##  [9] "NAME_HOUSING_TYPE"           "OCCUPATION_TYPE"
## [11] "WEEKDAY_APPR_PROCESS_START" "ORGANIZATION_TYPE"
## [13] "FONDKAPREMONT_MODE"          "HOUSETYPE_MODE"
## [15] "WALLSMATERIAL_MODE"          "EMERGENCYSTATE_MODE"
```

Numeric factor data

Next, let's look at which numeric data should be factored (categorical). These typically have a low number of unique levels.

```
unique_numeric_values_tbl <- x_train_tbl %>%
  select(x_train_tbl_skim$numeric$skim_variable) %>%
  map_df(~ unique(.) %>% length()) %>%
  gather() %>%
  arrange(value) %>%
  mutate(key = as_factor(key))

unique_numeric_values_tbl
```

```
## # A tibble: 105 x 2
##    key                          value
##    <fct>                        <int>
##  1 FLAG_MOBIL                       2
##  2 FLAG_EMP_PHONE                   2
##  3 FLAG_WORK_PHONE                  2
##  4 FLAG_CONT_MOBILE                 2
##  5 FLAG_PHONE                       2
##  6 FLAG_EMAIL                       2
##  7 REG_REGION_NOT_LIVE_REGION       2
##  8 REG_REGION_NOT_WORK_REGION       2
##  9 LIVE_REGION_NOT_WORK_REGION      2
## 10 REG_CITY_NOT_LIVE_CITY           2
```

```
factor_limit <- 7 # if the numeric column has less than 7 distinct values
                  # we consider it as a factor

num_2_factor_names <- unique_numeric_values_tbl %>%
  filter(value < factor_limit) %>%
  arrange(desc(value)) %>%
  pull(key) %>% # pull out a single variable as a vector
  as.character()

num_2_factor_names
```

```
##  [1] "AMT_REQ_CREDIT_BUREAU_HOUR"  "REGION_RATING_CLIENT"
##  [3] "REGION_RATING_CLIENT_W_CITY" "FLAG_MOBIL"
##  [5] "FLAG_EMP_PHONE"              "FLAG_WORK_PHONE"
##  [7] "FLAG_CONT_MOBILE"            "FLAG_PHONE"
##  [9] "FLAG_EMAIL"                  "REG_REGION_NOT_LIVE_REGION"
## [11] "REG_REGION_NOT_WORK_REGION"  "LIVE_REGION_NOT_WORK_REGION"
## [13] "REG_CITY_NOT_LIVE_CITY"      "REG_CITY_NOT_WORK_CITY"
## [15] "LIVE_CITY_NOT_WORK_CITY"     "FLAG_DOCUMENT_2"
## [17] "FLAG_DOCUMENT_3"             "FLAG_DOCUMENT_4"
## [19] "FLAG_DOCUMENT_5"             "FLAG_DOCUMENT_6"
## [21] "FLAG_DOCUMENT_7"             "FLAG_DOCUMENT_8"
## [23] "FLAG_DOCUMENT_9"             "FLAG_DOCUMENT_10"
## [25] "FLAG_DOCUMENT_11"            "FLAG_DOCUMENT_12"
## [27] "FLAG_DOCUMENT_13"            "FLAG_DOCUMENT_14"
## [29] "FLAG_DOCUMENT_15"            "FLAG_DOCUMENT_16"
## [31] "FLAG_DOCUMENT_17"            "FLAG_DOCUMENT_18"
## [33] "FLAG_DOCUMENT_19"            "FLAG_DOCUMENT_20"
```

`recipes` for ML data transformation

Once we know the issues in our training (and testing) data, how can we efficiently and safely transform the data so that they are appropriate for ML classifiers?

```
#install.packages("recipes")
library(recipes)
rec_obj <- recipe(~ ., data = x_train_tbl) %>%
  step_string2factor(all_of(string_2_factor_names)) %>%
  step_num2factor(all_of(num_2_factor_names),
                  levels=str_c(0:7),
                  transform = function(x) x + 1) %>%
  step_impute_mean(all_numeric()) %>% # missing values in numeric columns
  step_impute_mode(all_nominal()) %>% # missing values in factor columns
  prep(training = x_train_tbl)
```

```
rec_obj
```

```
## Data Recipe
##
## Inputs:
##
##       role #variables
##  predictor        121
##
## Training data contained 307511 data points and 298910 incomplete rows.
##
## Operations:
##
## Factor variables from NAME_CONTRACT_TYPE, CODE_GENDER, ... [trained]
## Factor variables from 35 items [trained]
## Mean Imputation for SK_ID_CURR, CNT_CHILDREN, ... [trained]
## Mode Imputation for NAME_CONTRACT_TYPE, CODE_GENDER, ... [trained]
```

With that recipe, we can start baking!

```
x_train_processed_tbl <- bake(rec_obj, x_train_tbl)
x_test_processed_tbl  <- bake(rec_obj, x_test_tbl)
```

This ensures that we **bake** our training data **and** testing data in the same way!

```
x_train_tbl %>%  # original data before baking
  select(1:30) %>%
  glimpse()
```

```
## Rows: 307,511
## Columns: 30
## $ SK_ID_CURR                  <dbl> 100002, 100003, 100004, 100006, 100007, 100~
## $ NAME_CONTRACT_TYPE          <chr> "Cash loans", "Cash loans", "Revolving loan~
## $ CODE_GENDER                 <chr> "M", "F", "M", "F", "M", "M", "F", "M", "F"~
## $ FLAG_OWN_CAR                <chr> "N", "N", "Y", "N", "N", "N", "Y", "Y", "N"~
## $ FLAG_OWN_REALTY             <chr> "Y", "N", "Y", "Y", "Y", "Y", "Y", "Y", "Y"~
## $ CNT_CHILDREN                <dbl> 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0~
## $ AMT_INCOME_TOTAL            <dbl> 202500.00, 270000.00, 67500.00, 135000.00, ~
## $ AMT_CREDIT                  <dbl> 406597.5, 1293502.5, 135000.0, 312682.5, 51~
## $ AMT_ANNUITY                 <dbl> 24700.5, 35698.5, 6750.0, 29686.5, 21865.5,~
## $ AMT_GOODS_PRICE             <dbl> 351000, 1129500, 135000, 297000, 513000, 45~
## $ NAME_TYPE_SUITE             <chr> "Unaccompanied", "Family", "Unaccompanied",~
## $ NAME_INCOME_TYPE            <chr> "Working", "State servant", "Working", "Wor~
## $ NAME_EDUCATION_TYPE         <chr> "Secondary / secondary special", "Higher ed~
## $ NAME_FAMILY_STATUS          <chr> "Single / not married", "Married", "Single ~
## $ NAME_HOUSING_TYPE           <chr> "House / apartment", "House / apartment", "~
## $ REGION_POPULATION_RELATIVE  <dbl> 0.018801, 0.003541, 0.010032, 0.008019, 0.0~
## $ DAYS_BIRTH                  <dbl> -9461, -16765, -19046, -19005, -19932, -169~
## $ DAYS_EMPLOYED               <dbl> -637, -1188, -225, -3039, -3038, -1588, -31~
## $ DAYS_REGISTRATION           <dbl> -3648, -1186, -4260, -9833, -4311, -4970, -~
## $ DAYS_ID_PUBLISH             <dbl> -2120, -291, -2531, -2437, -3458, -477, -61~
## $ OWN_CAR_AGE                 <dbl> NA, NA, 26, NA, NA, NA, 17, 8, NA, NA, NA, ~
## $ FLAG_MOBIL                  <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
```

```
x_train_processed_tbl %>%  # processed data after baking
  select(1:30) %>%
  glimpse()
```

```
## Rows: 307,511
## Columns: 30
## $ SK_ID_CURR                 <dbl> 100002, 100003, 100004, 100006, 100007, 100~
## $ NAME_CONTRACT_TYPE         <fct> Cash loans, Cash loans, Revolving loans, Ca~
## $ CODE_GENDER                <fct> M, F, M, F, M, M, F, M, F, M, F, F, F, M, F~
## $ FLAG_OWN_CAR               <fct> N, N, Y, N, N, N, Y, Y, N, N, N, N, N, Y, N~
## $ FLAG_OWN_REALTY            <fct> Y, N, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, N, Y~
## $ CNT_CHILDREN               <dbl> 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0~
## $ AMT_INCOME_TOTAL           <dbl> 202500.00, 270000.00, 67500.00, 135000.00, ~
## $ AMT_CREDIT                 <dbl> 406597.5, 1293502.5, 135000.0, 312682.5, 51~
## $ AMT_ANNUITY                <dbl> 24700.5, 35698.5, 6750.0, 29686.5, 21865.5,~
## $ AMT_GOODS_PRICE            <dbl> 351000, 1129500, 135000, 297000, 513000, 45~
## $ NAME_TYPE_SUITE            <fct> "Unaccompanied", "Family", "Unaccompanied",~
## $ NAME_INCOME_TYPE           <fct> Working, State servant, Working, Working, W~
## $ NAME_EDUCATION_TYPE        <fct> Secondary / secondary special, Higher educa~
## $ NAME_FAMILY_STATUS         <fct> Single / not married, Married, Single / not~
## $ NAME_HOUSING_TYPE          <fct> House / apartment, House / apartment, House~
## $ REGION_POPULATION_RELATIVE <dbl> 0.018801, 0.003541, 0.010032, 0.008019, 0.0~
## $ DAYS_BIRTH                 <dbl> -9461, -16765, -19046, -19005, -19932, -169~
## $ DAYS_EMPLOYED              <dbl> -637, -1188, -225, -3039, -3038, -1588, -31~
## $ DAYS_REGISTRATION          <dbl> -3648, -1186, -4260, -9833, -4311, -4970, -~
## $ DAYS_ID_PUBLISH            <dbl> -2120, -291, -2531, -2437, -3458, -477, -61~
## $ OWN_CAR_AGE                <dbl> 12.06109, 12.06109, 26.00000, 12.06109, 12,~
## $ FLAG_MOBIL                 <fct> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
```

The outcome (y) also needs to be transformed:

```
rec_obj_for_y <- recipe(~ ., data = y_train_tbl) %>%
  step_num2factor("TARGET", levels = c("0","1"),
                  transform = function(x) x + 1) %>%
  prep(stringsAsFactors = FALSE)
y_train_processed_tbl  <- bake(rec_obj_for_y, y_train_tbl)
```

```
y_train_tbl
```

```
## # A tibble: 307,511 x 1
##    TARGET
##     <dbl>
##  1      1
##  2      0
##  3      0
##  4      0
##  5      0
##  6      0
##  7      0
##  8      0
##  9      0
## 10      0
## # ... with 307,501 more rows
```

```
y_train_processed_tbl
```

```
## # A tibble: 307,511 x 1
##    TARGET
##    <fct>
##  1 1
##  2 0
##  3 0
##  4 0
##  5 0
##  6 0
##  7 0
##  8 0
##  9 0
## 10 0
## # ... with 307,501 more rows
```

Before we take a break, let's save the processed data to your working directory. We will need it later.

```
write_rds(x_train_processed_tbl, "x_train_processed_tbl.rds")
write_rds(x_test_processed_tbl, "x_test_processed_tbl.rds")
write_rds(y_train_processed_tbl, "y_train_processed_tbl.rds")
```

Mid-Semester Course Feedback

Providing you and other students with high-quality learning experience is important to me as your lecturer.

Please take a minute during the break to provide some feedback. The information you provide will help improve this course. Your responses are anonymous.

https://tinyurl.com/yxdfeb6q