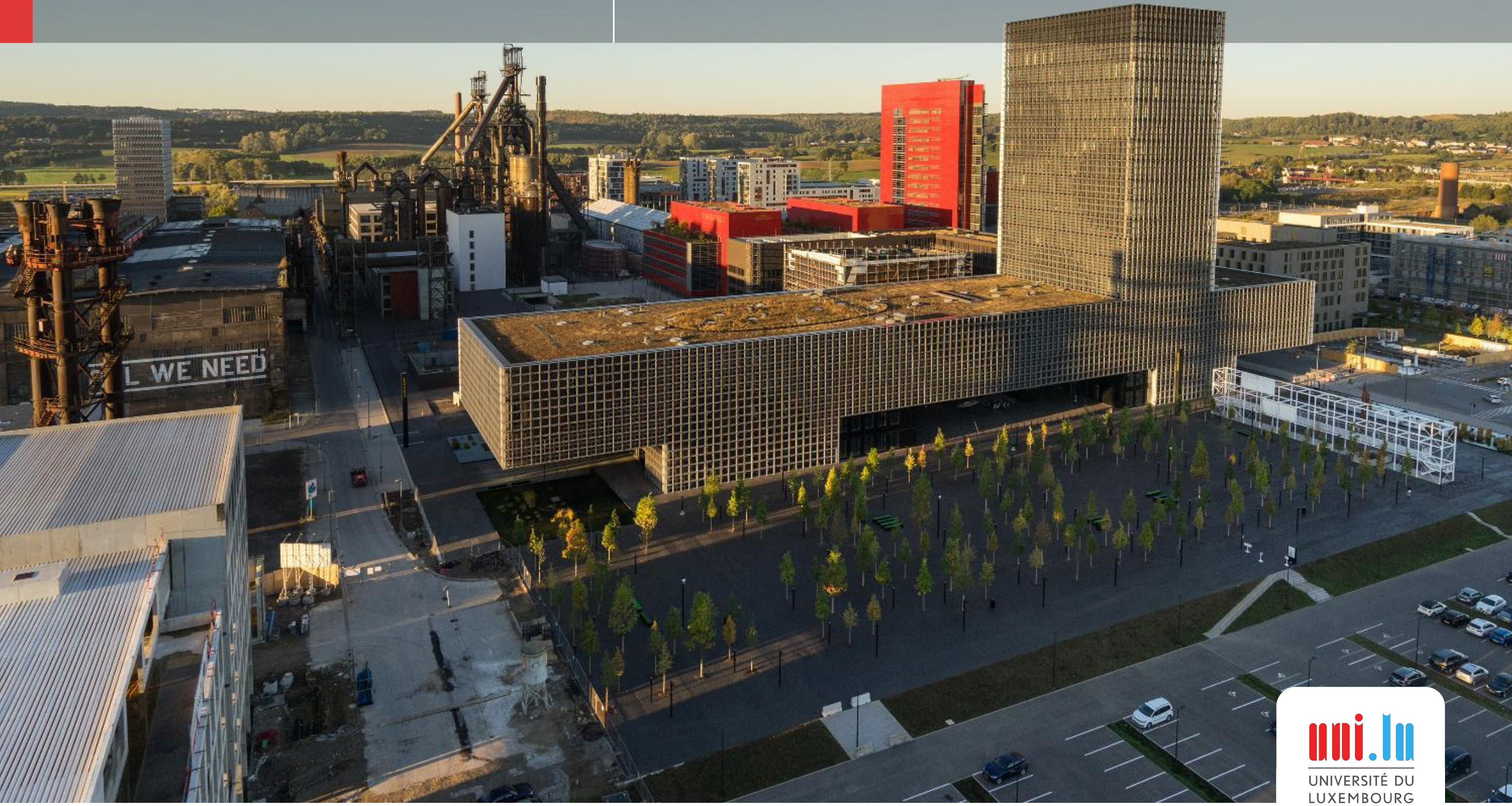
University of Luxembourg

Multilingual. Personalised. Connected.

Applied ML – Project Part 1

Fabien BERNIER & Yann HOFFMANN - October 2022



UNIVERSITÉ DU LUXEMBOURG

Previously: Machine Learning Project Lifecycle

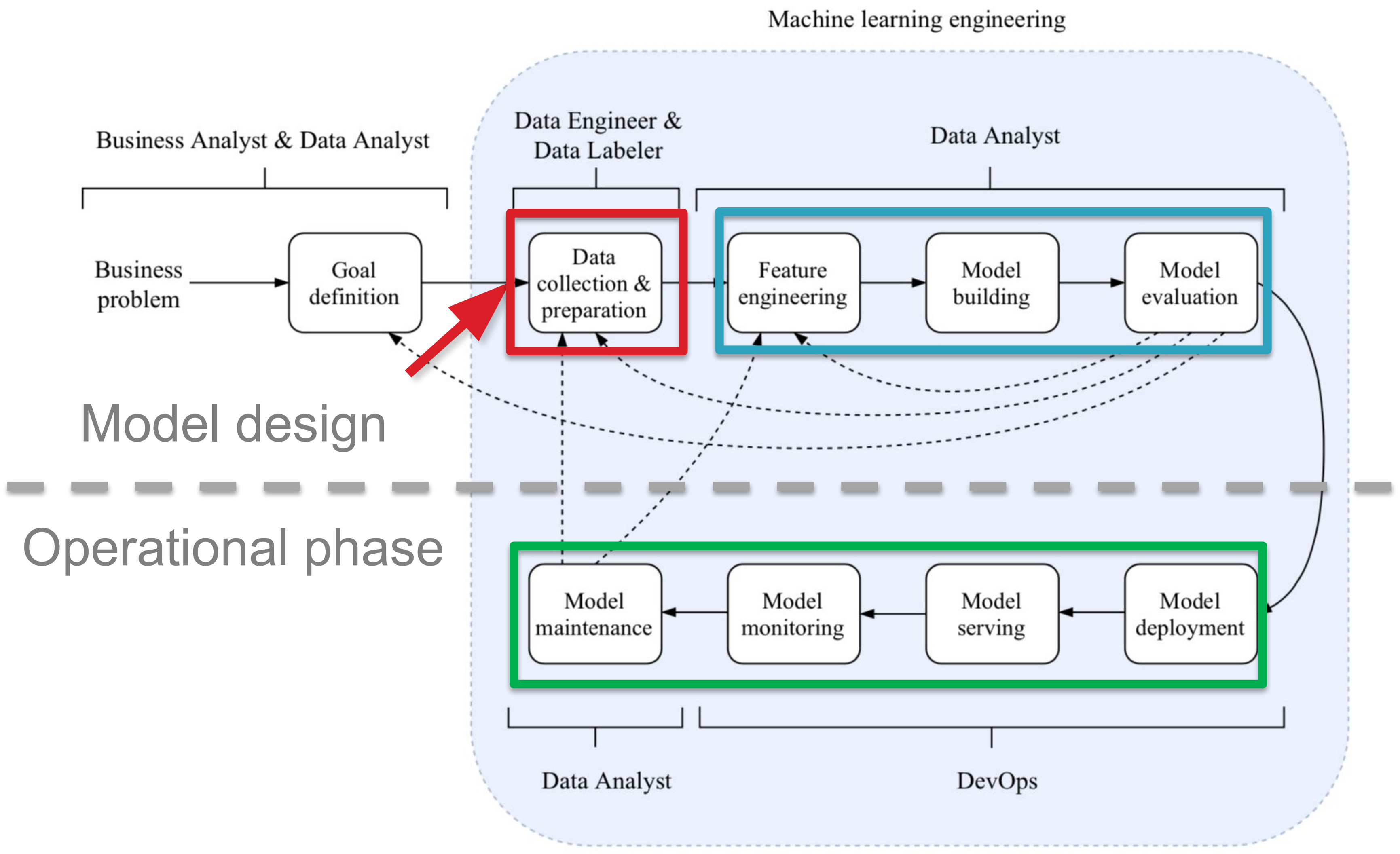


Figure 3: Machine learning project life cycle.

Use case

HOME CREDIT

UNIVERSITÉ DU LUXEMBOURG

securityandtrust.lu

What we will see

- Actual practice of Machine Learning (ML) & Data Science (DS) in Software Engineering

- Machine Learning pipeline
  - Data collection
  - Data exploration
  - Feature Engineering
  - Machine Learning model
  - Deployment

- Using Python3 DS & ML frameworks to build a real-case software: Numpy, Pandas and Scikit Learn

- This course is accompanied by a Python notebook

Data and variable dictionary

Data available to download here:

- https://drive.google.com/drive/folders/1cSq-3a3KusGU05Qb9UQMkKXqH9posbeT?usp=sharing

- Dictionary of variables are available on the Homecredit_columns_description.csv

The notebook containing the code presented here available here:

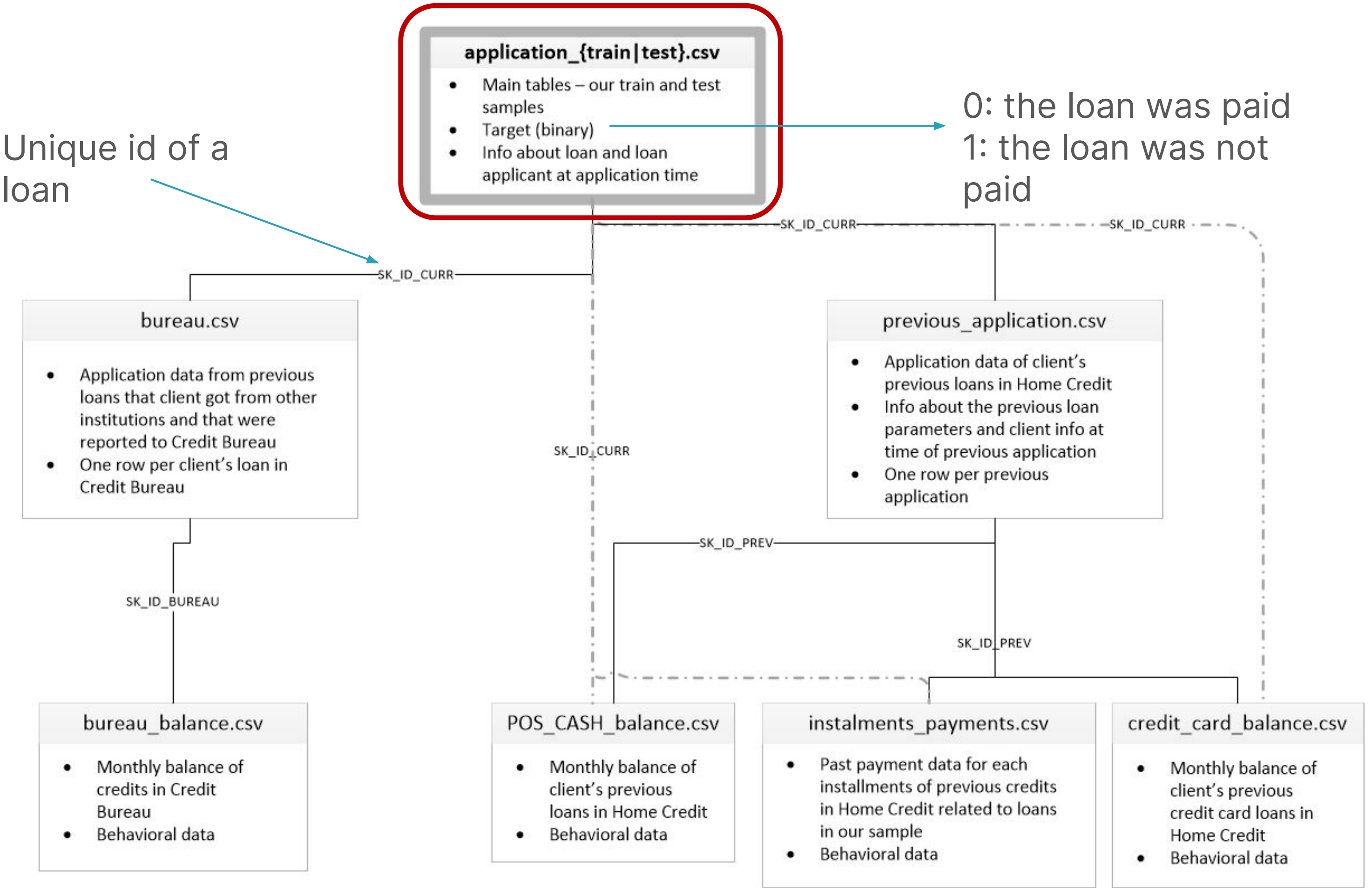
- https://colab.research.google.com/drive/148fNMkCB0RtBKMHXjOC0uCgbWkE_PERJ?usp=sharing

You must complete another notebook available on Moodle for the project.

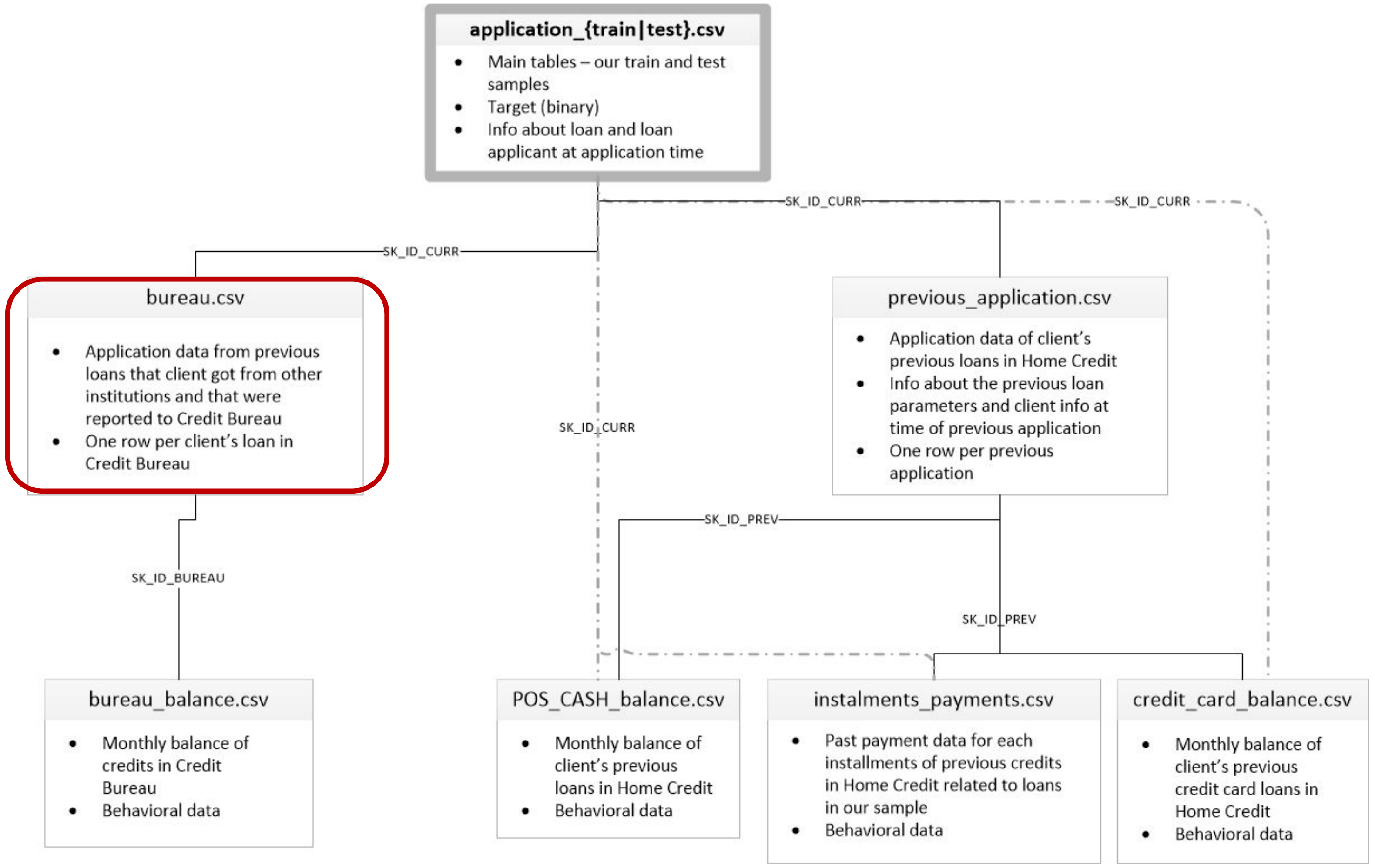
Data Collection > The dataset

Unique id of a loan

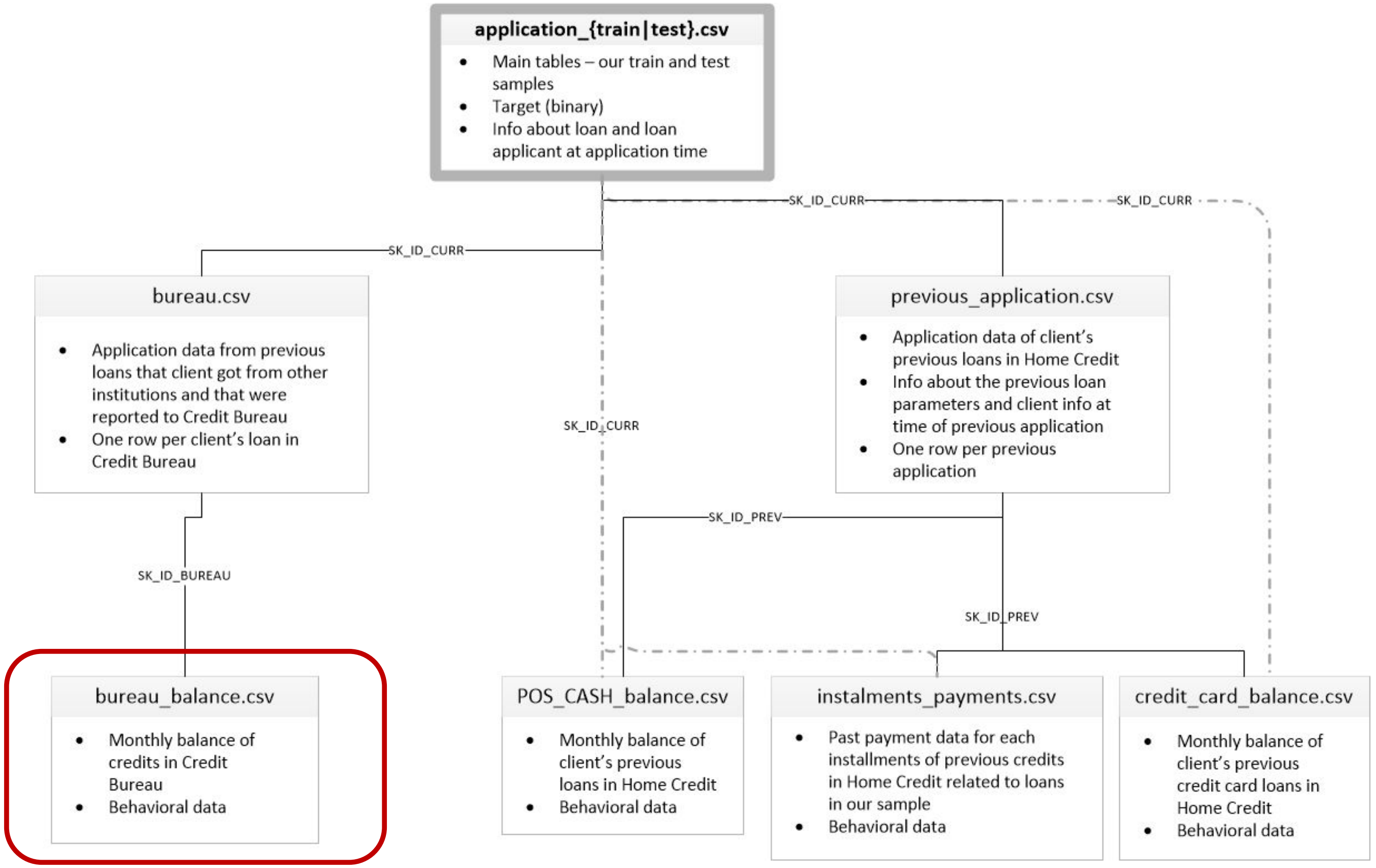
0: the loan was paid
1: the loan was not paid


application_{train|test}.csv
Main tables – our train and test samples
Target (binary)
Info about loan and loan applicant at application time
SK_ID_CURR
SK_ID_CURR
SK_ID_CURR
SK_ID_CURR
bureau.csv
Application data from previous loans that client got from other institutions and that were reported to Credit Bureau
One row per client's loan in Credit Bureau
previous_application.csv
Application data of client's previous loans in Home Credit
Info about the previous loan parameters and client info at time of previous application
One row per previous application
SK_ID_BUREAU
SK_ID_PREV
SK_ID_PREV
bureau_balance.csv
Monthly balance of credits in Credit Bureau
Behavioral data
POS_CASH_balance.csv
Monthly balance of client's previous loans in Home Credit
Behavioral data
instalments_payments.csv
Past payment data for each installments of previous credits in Home Credit related to loans in our sample
Behavioral data
credit_card_balance.csv
Monthly balance of client's previous credit card loans in Home Credit
Behavioral data

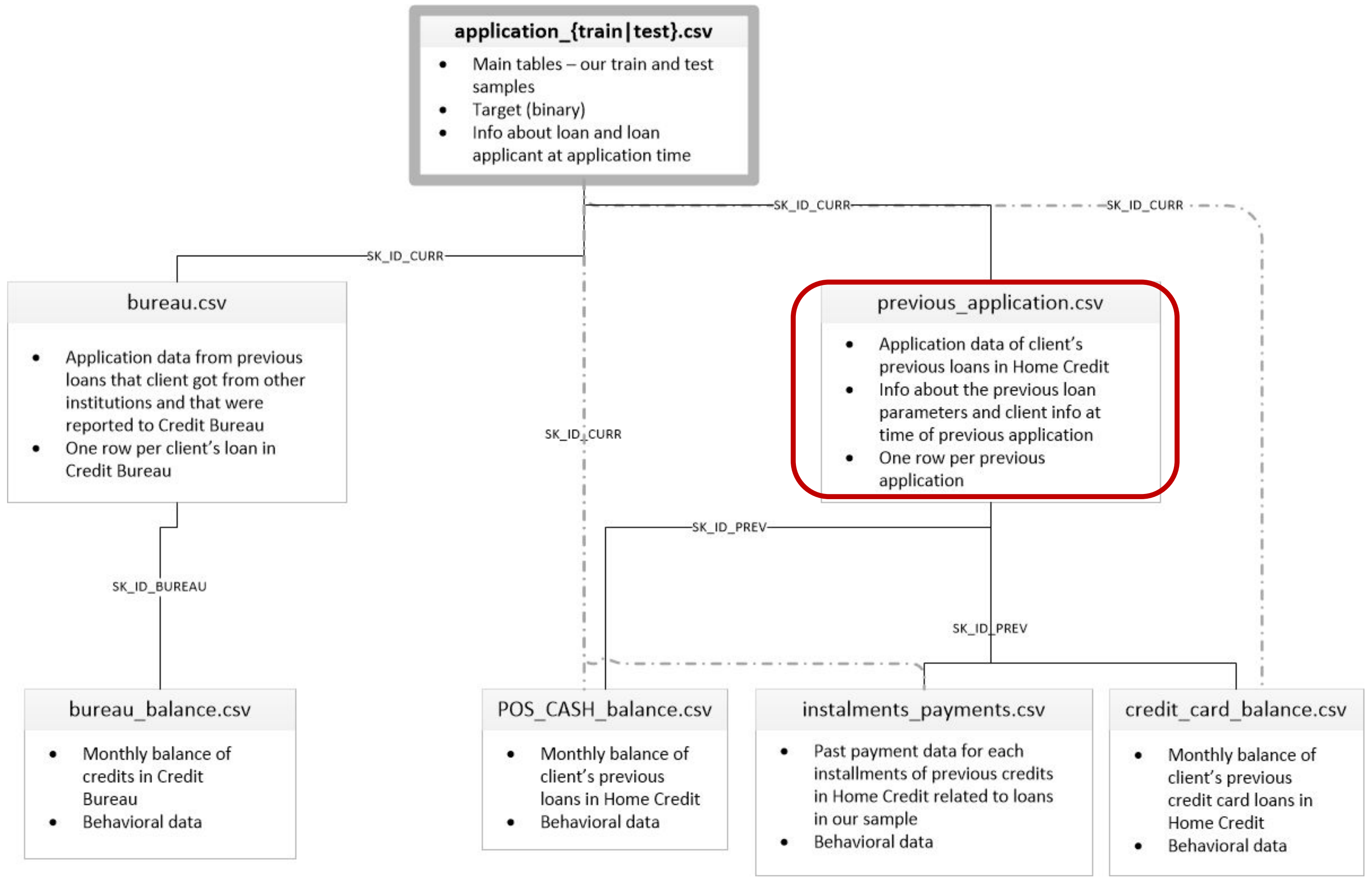
Data Collection > The dataset


application_{train|test}.csv
Main tables – our train and test samples
Target (binary)
Info about loan and loan applicant at application time
SK_ID_CURR
SK_ID_CURR
SK_ID_CURR
SK_ID_CURR
bureau.csv
Application data from previous loans that client got from other institutions and that were reported to Credit Bureau
One row per client's loan in Credit Bureau
previous_application.csv
Application data of client's previous loans in Home Credit
Info about the previous loan parameters and client info at time of previous application
One row per previous application
SK_ID_BUREAU
SK_ID_PREV
SK_ID_PREV
bureau_balance.csv
Monthly balance of credits in Credit Bureau
Behavioral data
POS_CASH_balance.csv
Monthly balance of client's previous loans in Home Credit
Behavioral data
instalments_payments.csv
Past payment data for each installments of previous credits in Home Credit related to loans in our sample
Behavioral data
credit_card_balance.csv
Monthly balance of client's previous credit card loans in Home Credit
Behavioral data

Data Collection > The dataset

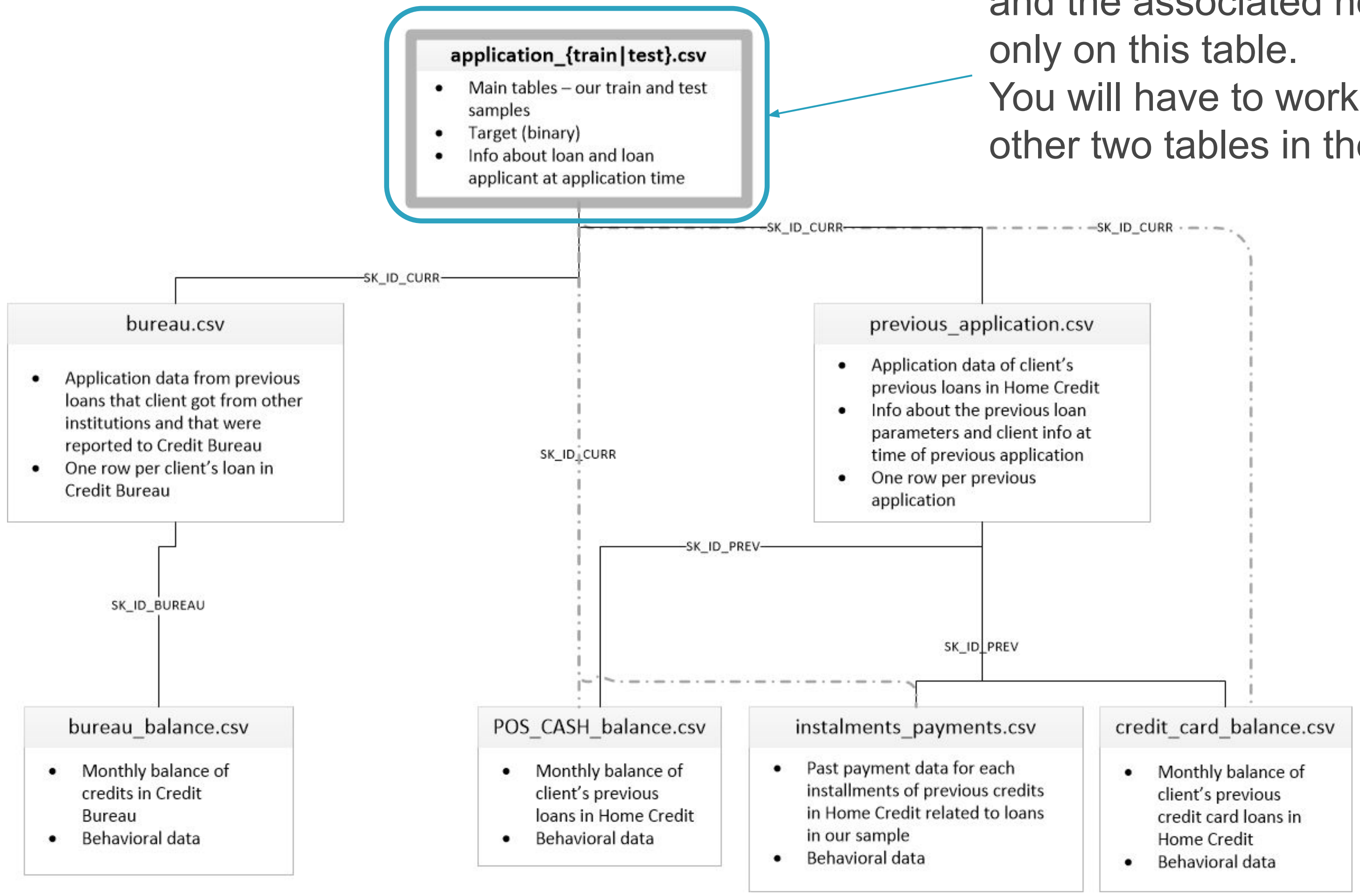

application_{train|test}.csv
Main tables – our train and test samples
Target (binary)
Info about loan and loan applicant at application time
SK_ID_CURR
SK_ID_CURR
SK_ID_CURR
SK_ID_CURR
bureau.csv
Application data from previous loans that client got from other institutions and that were reported to Credit Bureau
One row per client's loan in Credit Bureau
previous_application.csv
Application data of client's previous loans in Home Credit
Info about the previous loan parameters and client info at time of previous application
One row per previous application
SK_ID_BUREAU
SK_ID_PREV
SK_ID_PREV
bureau_balance.csv
Monthly balance of credits in Credit Bureau
Behavioral data
POS_CASH_balance.csv
Monthly balance of client's previous loans in Home Credit
Behavioral data
instalments_payments.csv
Past payment data for each installments of previous credits in Home Credit related to loans in our sample
Behavioral data
credit_card_balance.csv
Monthly balance of client's previous credit card loans in Home Credit
Behavioral data

Data Collection > The dataset



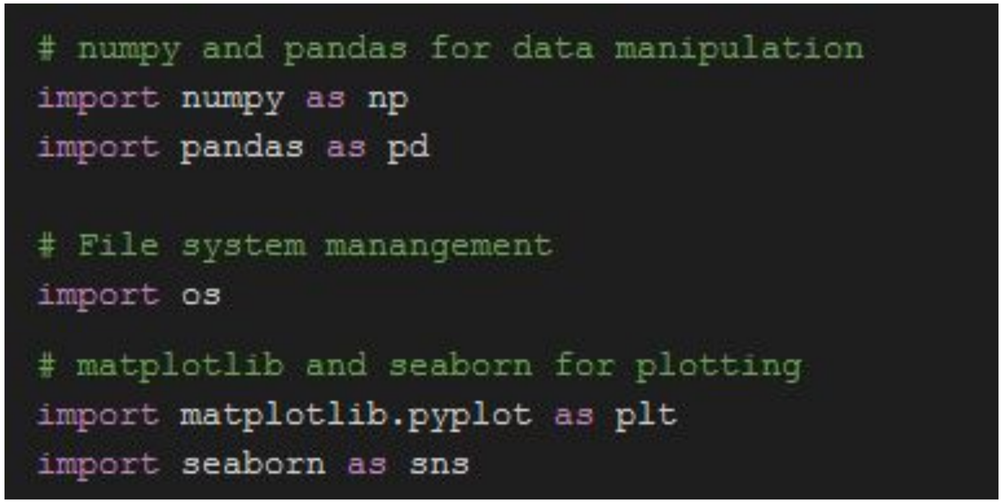
application_{train|test}.csv
Main tables – our train and test samples
Target (binary)
Info about loan and loan applicant at application time
SK_ID_CURR
SK_ID_CURR
SK_ID_CURR
SK_ID_CURR
bureau.csv
Application data from previous loans that client got from other institutions and that were reported to Credit Bureau
One row per client's loan in Credit Bureau
previous_application.csv
Application data of client's previous loans in Home Credit
Info about the previous loan parameters and client info at time of previous application
One row per previous application
SK_ID_BUREAU
SK_ID_PREV
SK_ID_PREV
bureau_balance.csv
Monthly balance of credits in Credit Bureau
Behavioral data
POS_CASH_balance.csv
Monthly balance of client's previous loans in Home Credit
Behavioral data
instalments_payments.csv
Past payment data for each installments of previous credits in Home Credit related to loans in our sample
Behavioral data
credit_card_balance.csv
Monthly balance of client's previous credit card loans in Home Credit
Behavioral data

Data Collection > The dataset


application_{train|test}.csv
Main tables – our train and test samples
Target (binary)
Info about loan and loan applicant at application time
SK_ID_CURR
SK_ID_CURR
SK_ID_CURR
SK_ID_CURR
bureau.csv
Application data from previous loans that client got from other institutions and that were reported to Credit Bureau
One row per client's loan in Credit Bureau
previous_application.csv
Application data of client's previous loans in Home Credit
Info about the previous loan parameters and client info at time of previous application
One row per previous application
SK_ID_BUREAU
SK_ID_PREV
SK_ID_PREV
bureau_balance.csv
Monthly balance of credits in Credit Bureau
Behavioral data
POS_CASH_balance.csv
Monthly balance of client's previous loans in Home Credit
Behavioral data
instalments_payments.csv
Past payment data for each installments of previous credits in Home Credit related to loans in our sample
Behavioral data
credit_card_balance.csv
Monthly balance of client's previous credit card loans in Home Credit
Behavioral data

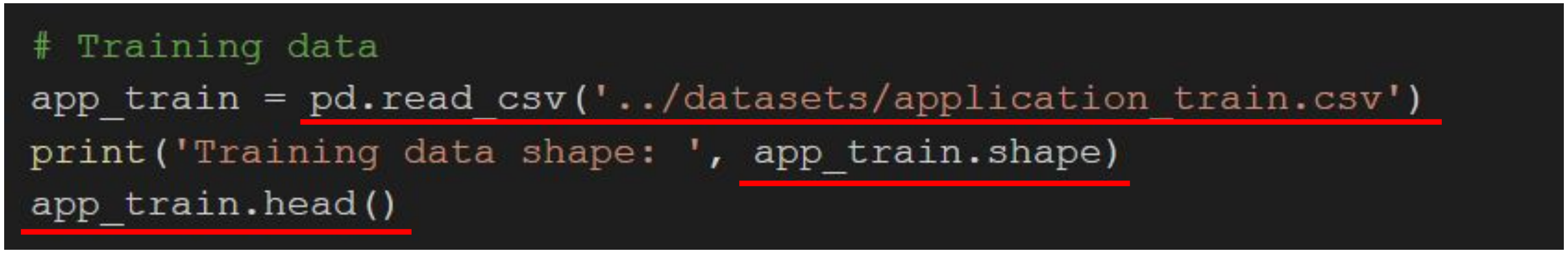
We focus this course and the associated notebook only on this table.
You will have to work on the other two tables in the project.

Data Collection > Loading libraries

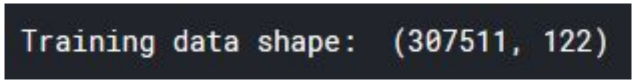


```
# numpy and pandas for data manipulation
import numpy as np
import pandas as pd

# File system manangement
import os

# matplotlib and seaborn for plotting
import matplotlib.pyplot as plt
import seaborn as sns
```

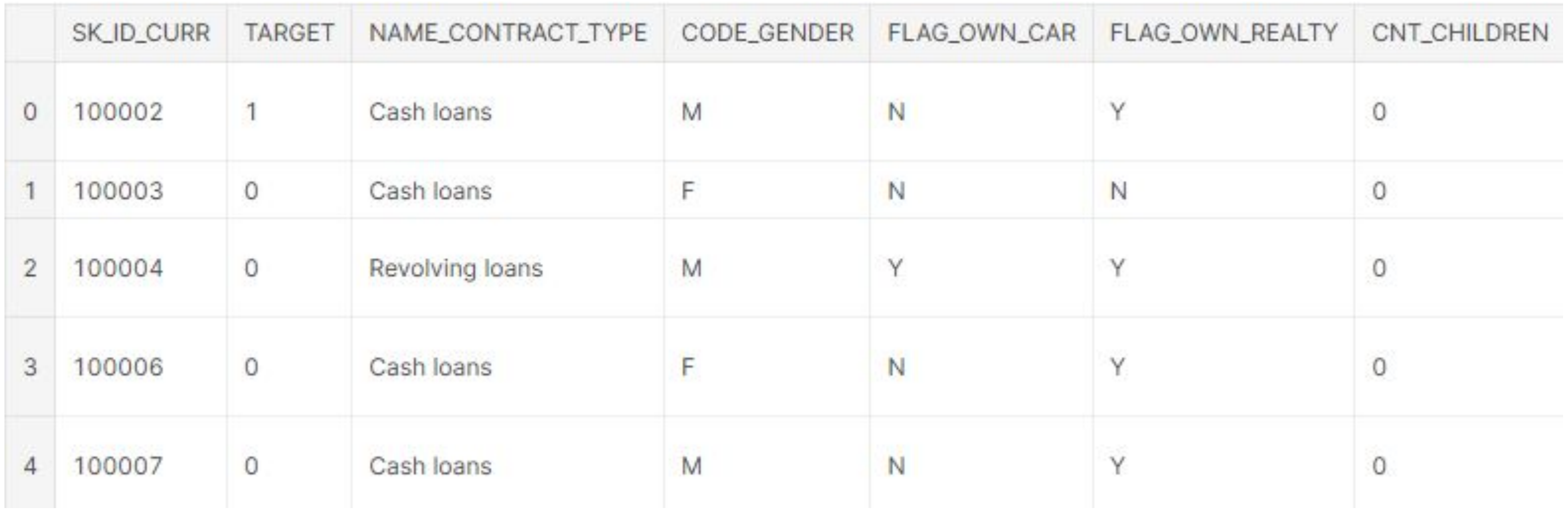
Data Collection > Loading datasets



```
# Training data
app_train = pd.read_csv('../datasets/application_train.csv')
print('Training data shape: ', app_train.shape)
app_train.head()
```



```
Training data shape:  (307511, 122)
```



	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN
0	100002	1	Cash loans	M	N	Y	0
1	100003	0	Cash loans	F	N	N	0
2	100004	0	Revolving loans	M	Y	Y	0
3	100006	0	Cash loans	F	N	Y	0
4	100007	0	Cash loans	M	N	Y	0

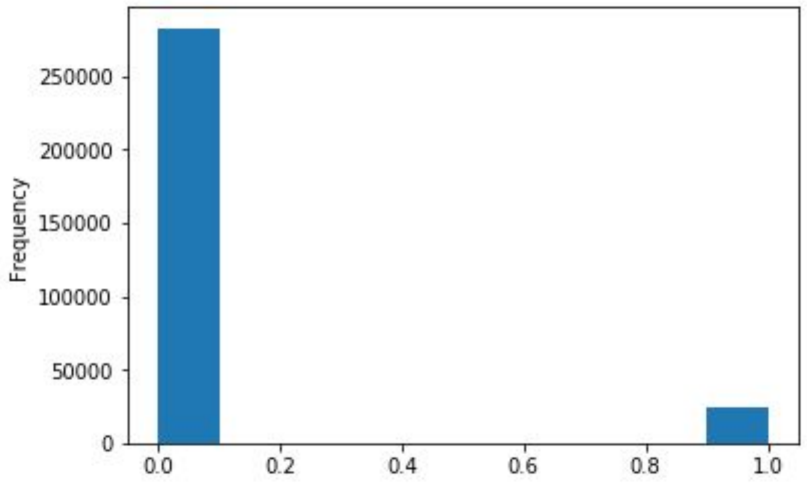
Data exploration > Target distribution

```
app_train['TARGET'].value_counts()
```

```
app_train['TARGET'].astype(int).plot.hist();
```

Class imbalance

```
0    282686
1     24825
Name: TARGET, dtype: int64
```



Sampling
Data augmentation

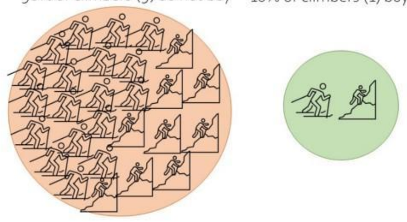
Data exploration > Target distribution > Class imbalance

Techniques to deal with class imbalance

- **Undersampling of the majority class**
  - Randomly discard examples of the class that is present too often until there is the same number of examples for both classes
  - Pro: easy
  - Con: many valuable data lost
- **Oversampling of the minority class**
  - Randomly duplicate examples of the class that is present the least until there is the same number of examples for both classes
  - Pro: no data lost
  - Con: many duplicates, learn on “false” information
- **Data Augmentation**
  - Similar to oversampling but add a small perturbations to the duplicated examples. See the SMOTE algorithm.
  - Pro: no duplicated data
  - Con: more complex

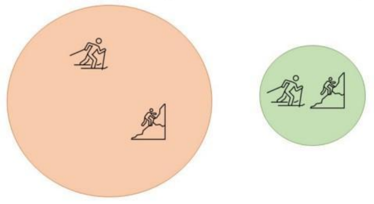
95% of skiers (19) do not buy
90% of climbers (9) do not buy

5% of skiers (1) buy
10% of climbers (1) buy

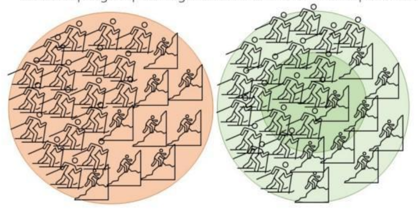


In total, 28 visitors don’t buy and 2 buy.
A model that predicts that **nobody ever buys** is correct in 28 out of 30 cases.
Accuracy = 93%, **but the model is useless.**

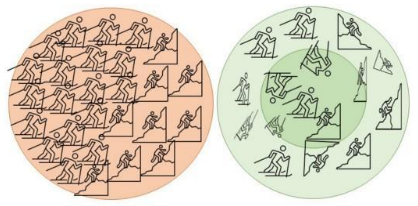
Undersampling: deleting occurrences of the more frequent class



Oversampling: duplicating occurrences of the less frequent class

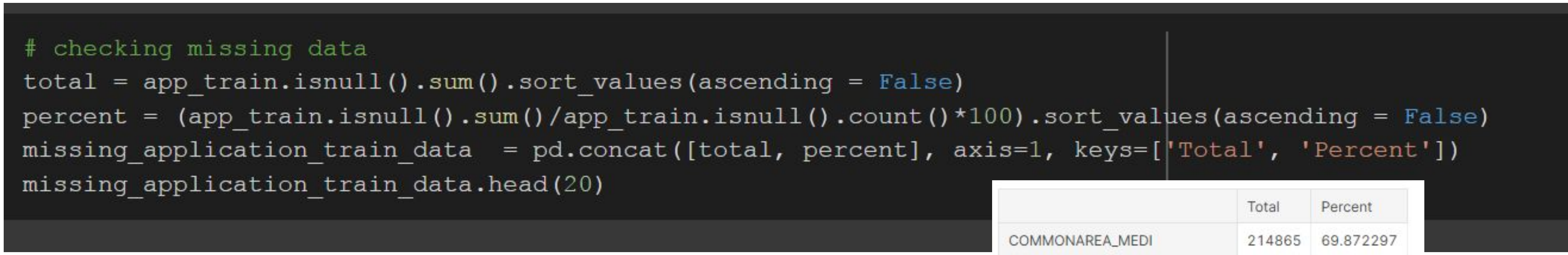


Data Augmentation: duplicating and perturbing occurrences of the less frequent class

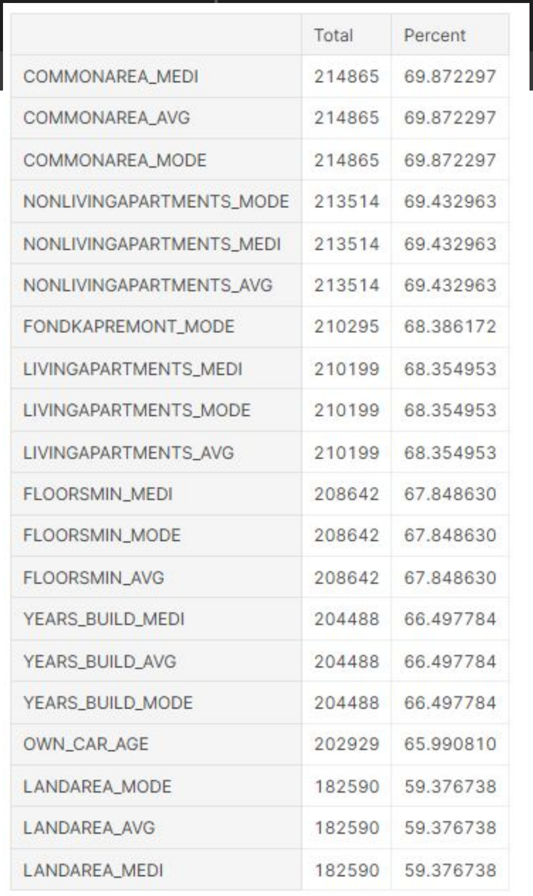


Images from https://towardsdatascience.com/smote-fdce2f605729

Data exploration > Missing values



```
# checking missing data
total = app_train.isnull().sum().sort_values(ascending = False)
percent = (app_train.isnull().sum()/app_train.isnull().count()*100).sort_values(ascending = False)
missing_application_train_data  = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing_application_train_data.head(20)
```



	Total	Percent
COMMONAREA_MEDI	214865	69.872297
COMMONAREA_AVG	214865	69.872297
COMMONAREA_MODE	214865	69.872297
NONLIVINGAPARTMENTS_MODE	213514	69.432963
NONLIVINGAPARTMENTS_MEDI	213514	69.432963
NONLIVINGAPARTMENTS_AVG	213514	69.432963
FONDKAPREMONT_MODE	210295	68.386172
LIVINGAPARTMENTS_MEDI	210199	68.354953
LIVINGAPARTMENTS_MODE	210199	68.354953
LIVINGAPARTMENTS_AVG	210199	68.354953
FLOORSMIN_MEDI	208642	67.848630
FLOORSMIN_MODE	208642	67.848630
FLOORSMIN_AVG	208642	67.848630
YEARS_BUILD_MEDI	204488	66.497784
YEARS_BUILD_AVG	204488	66.497784
YEARS_BUILD_MODE	204488	66.497784
OWN_CAR_AGE	202929	65.990810
LANDAREA_MODE	182590	59.376738
LANDAREA_AVG	182590	59.376738
LANDAREA_MEDI	182590	59.376738

There are 67 columns that have missing values

Techniques to deal with missing data

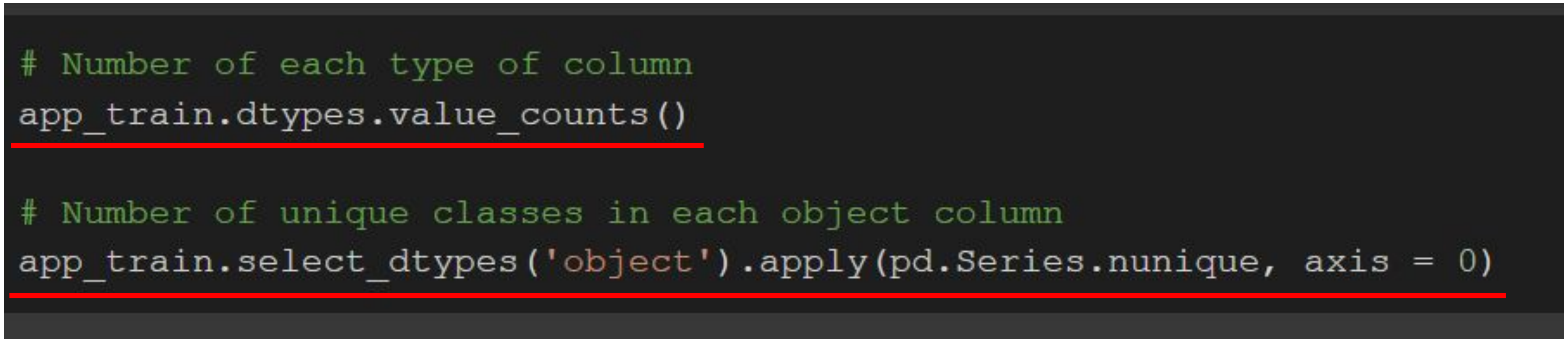
- **Discard** columns and/or rows with many missing values
  - **not** recommended, because it might be an important variable, or an observation that you need to predict. Recommended if all values are missing.
- **Categorization**: for categorical variables, create a “missing value” category
  - recommended
- **Imputation**: fill values
  - Simplest: impute a constant value (most frequent, or mean/median)
  - Nearest neighbors imputation: impute the value of the observation closest according to a metric that supports missing values
  - Modeling imputation: recursively model one column based on the other columns
- Use a model that support missing values
  - e.g., gradient boosting

□ You will have to choose one or several techniques and implement it/them

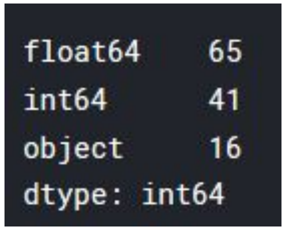
Data exploration > Missing values > Filling

Feature Engineering

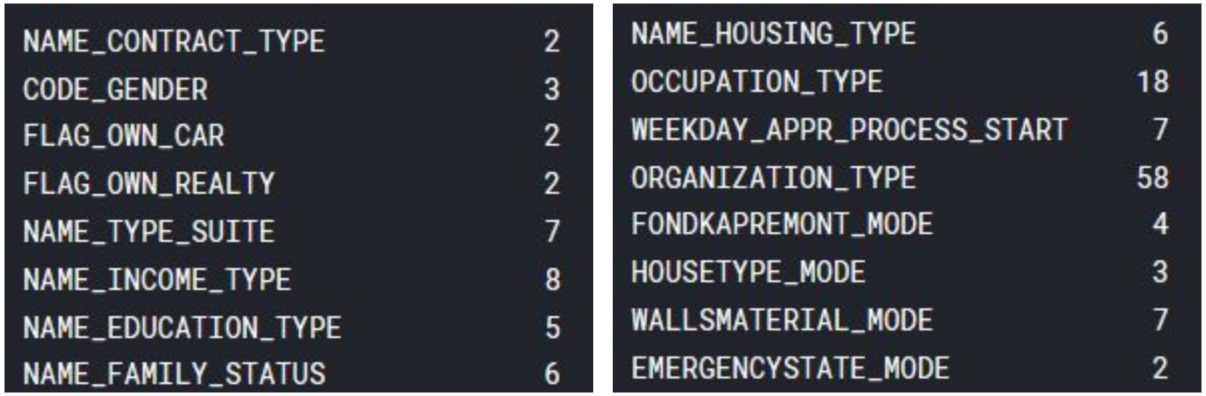
Data exploration > Categorical encoding



```python
# Number of each type of column
app_train.dtypes.value_counts()

# Number of unique classes in each object column
app_train.select_dtypes('object').apply(pd.Series.nunique, axis = 0)
```



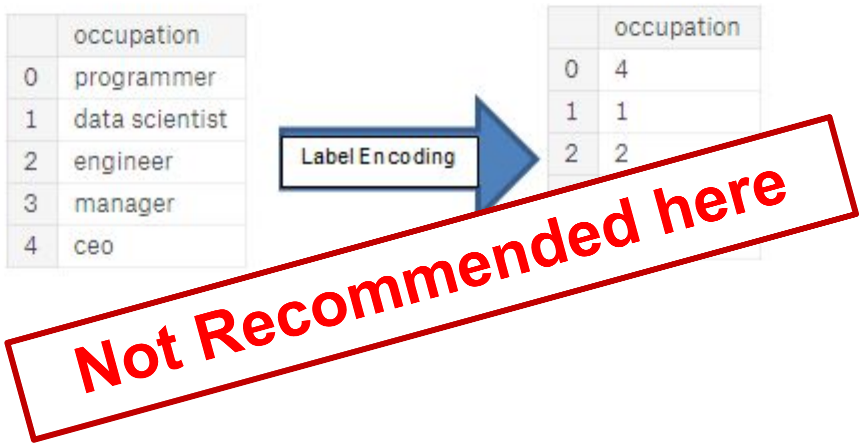
```
float64    65
int64      41
object     16
dtype: int64
```



```
NAME_CONTRACT_TYPE             2
CODE_GENDER                    3
FLAG_OWN_CAR                   2
FLAG_OWN_REALTY                2
NAME_TYPE_SUITE                7
NAME_INCOME_TYPE               8
NAME_EDUCATION_TYPE            5
NAME_FAMILY_STATUS             6
NAME_HOUSING_TYPE              6
OCCUPATION_TYPE               18
WEEKDAY_APPR_PROCESS_START     7
ORGANIZATION_TYPE             58
FONDKAPREMONT_MODE             4
HOUSETYPE_MODE                 3
WALLSMATERIAL_MODE             7
EMERGENCYSTATE_MODE            2
```

Data exploration > Categorical encoding > Label encoding

We assign each unique category in a categorical variable with an integer.



	occupation
0	programmer
1	data scientist
2	engineer
3	manager
4	ceo

	occupation
0	4
1	1
2	2

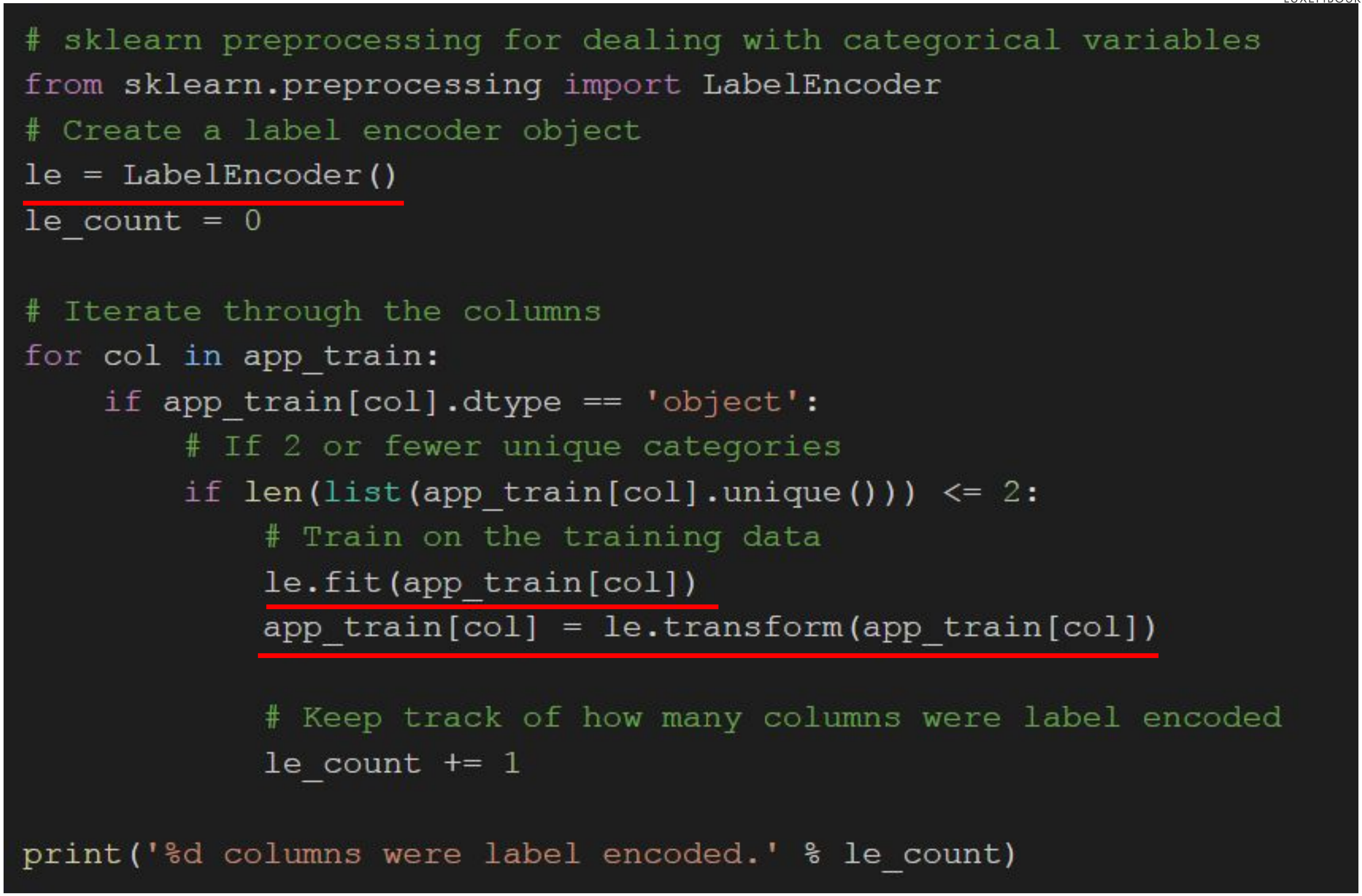
PROs:

- No new columns created
- Ideal for binary categories or ordinal data (e.g., categories of ages)

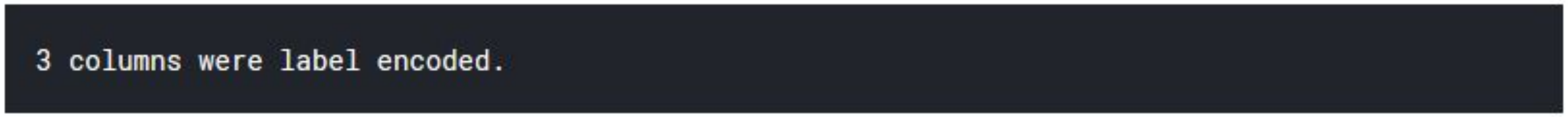
CONs:

- Imply closeness of label (0□1 “=” 3□4)
- Arbitrary ordering

Data exploration > Categorical encoding > Label encoding



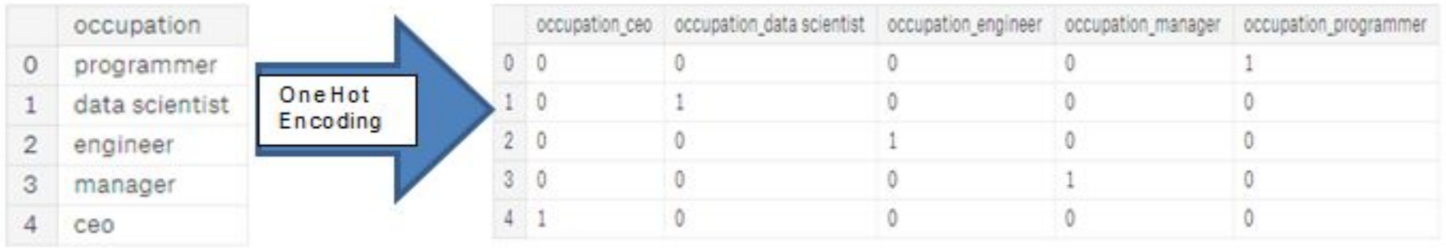
```
# sklearn preprocessing for dealing with categorical variables
from sklearn.preprocessing import LabelEncoder
# Create a label encoder object
le = LabelEncoder()
le_count = 0

# Iterate through the columns
for col in app_train:
    if app_train[col].dtype == 'object':
        # If 2 or fewer unique categories
        if len(list(app_train[col].unique())) <= 2:
            # Train on the training data
            le.fit(app_train[col])
            app_train[col] = le.transform(app_train[col])

            # Keep track of how many columns were label encoded
            le_count += 1

print('%d columns were label encoded.' % le_count)
```



```
3 columns were label encoded.
```

Data exploration > Categorical encoding > One-hot encoding

We create a new column for each unique category in a categorical variable. Each observation receives a 1 in the column for its corresponding category and a 0 in all other new columns.



	occupation
0	programmer
1	data scientist
2	engineer
3	manager
4	ceo

	occupation_ceo	occupation_data scientist	occupation_engineer	occupation_manager	occupation_programmer
0	0	0	0	0	1
1	0	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	0
4	1	0	0	0	0

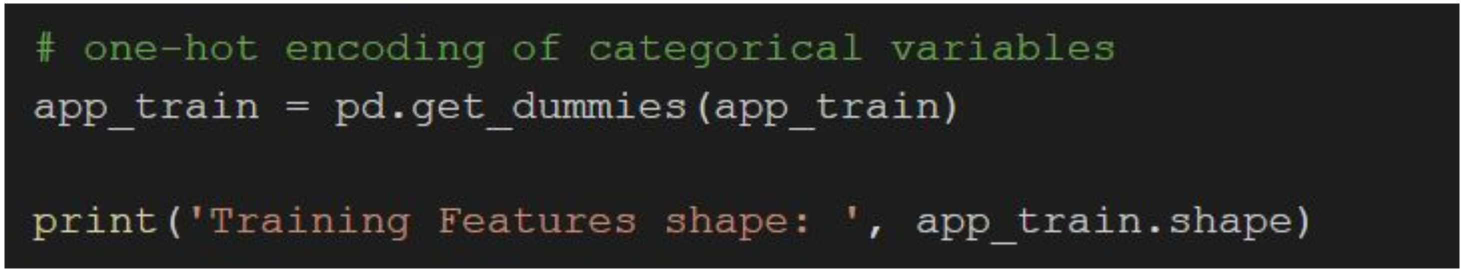
PROs:

- No relative values assumptions & closeness
- Handles "AND" cases

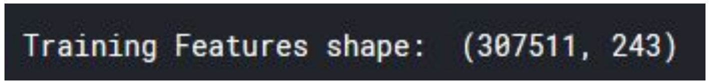
CONs:

- Number of features explodes with many categories to each variable

Data exploration > Categorical encoding > One-hot encoding



```
# one-hot encoding of categorical variables
app_train = pd.get_dummies(app_train)

print('Training Features shape: ', app_train.shape)
```



```
Training Features shape:  (307511, 243)
```

Data exploration > Anomalies

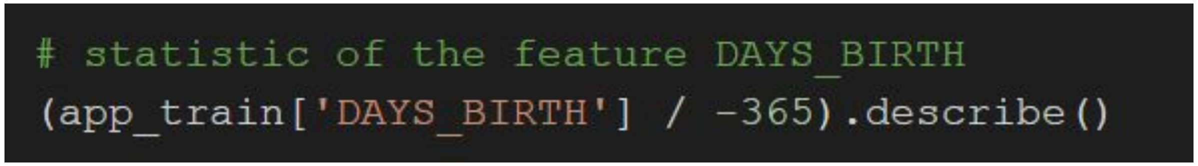
Mistyped values

Noised measures

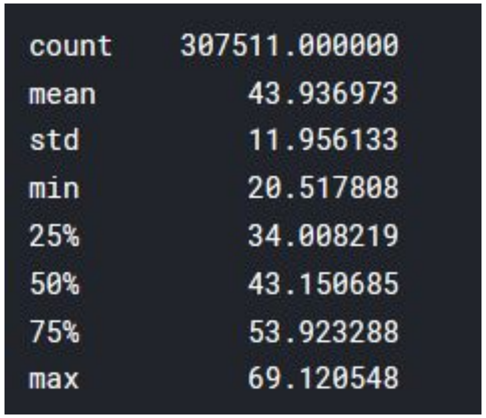
Data collection errors

Outlier values

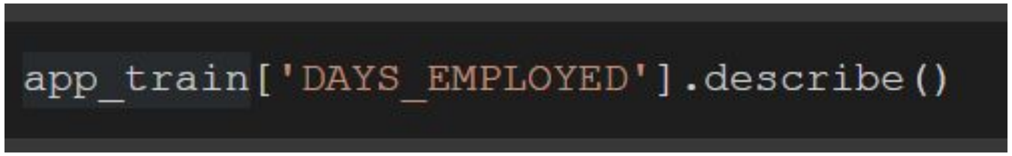
Data exploration > Anomalies



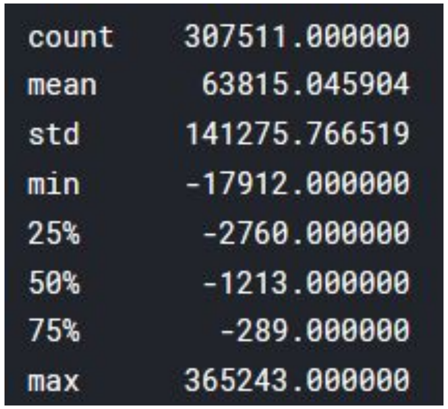
```
# statistic of the feature DAYS_BIRTH
(app_train['DAYS_BIRTH'] / -365).describe()
```



```
count    307511.000000
mean         43.936973
std          11.956133
min          20.517808
25%          34.008219
50%          43.150685
75%          53.923288
max          69.120548
```

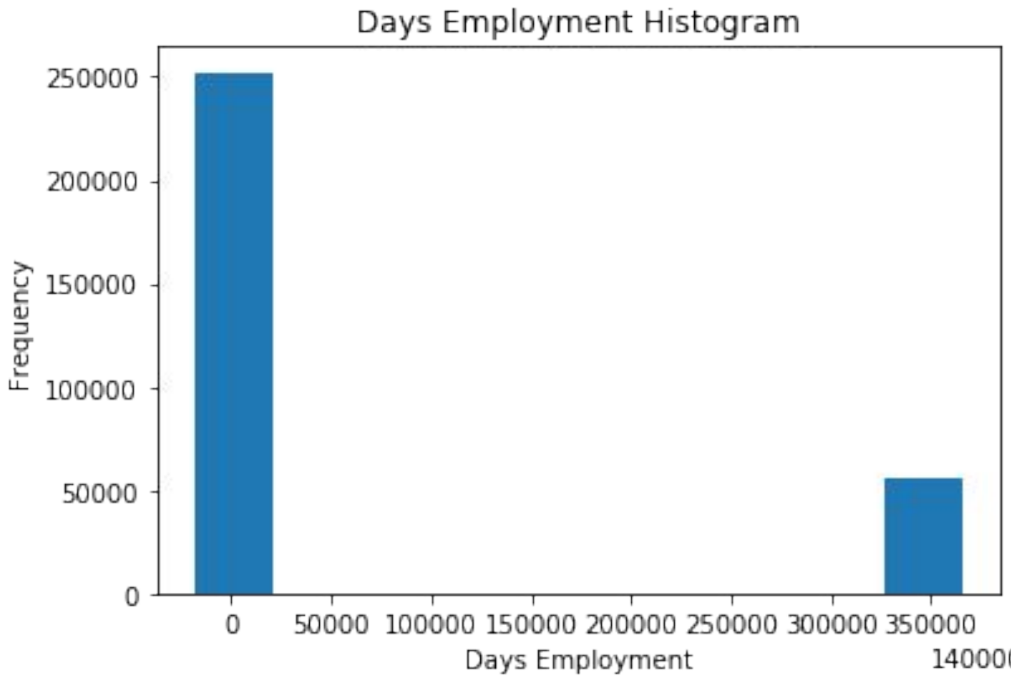


```
app_train['DAYS_EMPLOYED'].describe()
```

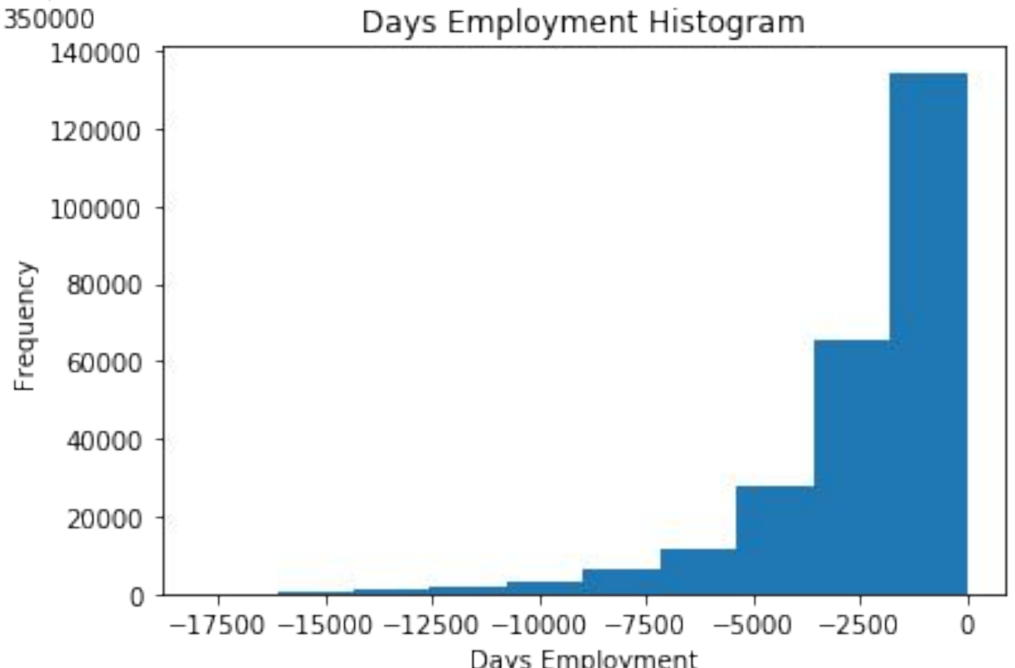


```
count    307511.000000
mean      63815.045904
std      141275.766519
min      -17912.000000
25%       -2760.000000
50%       -1213.000000
75%        -289.000000
max      365243.000000
```

Data exploration > Anomalies



With abnormal values

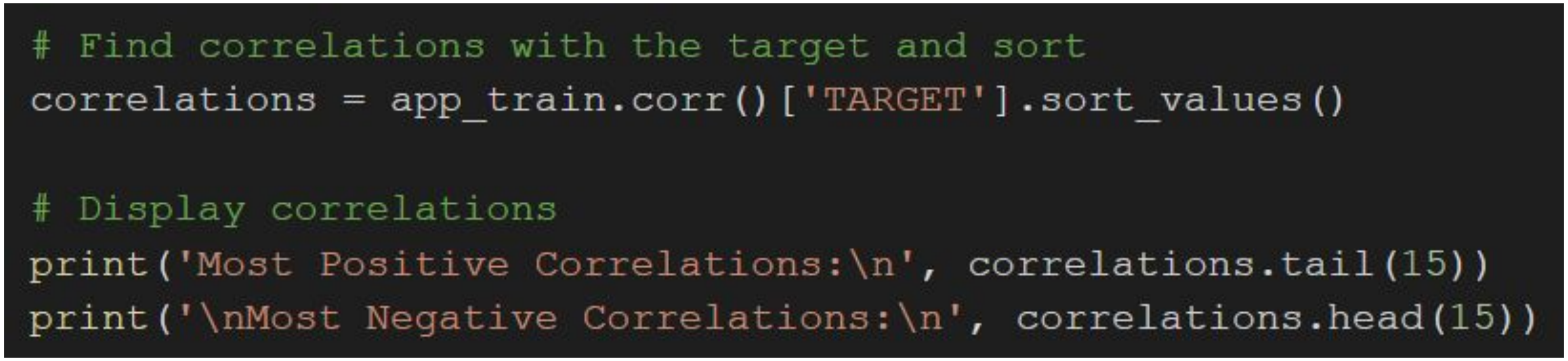


Abnormal values removed

Data exploration > Feature effects > Correlations

Pearson Correlation

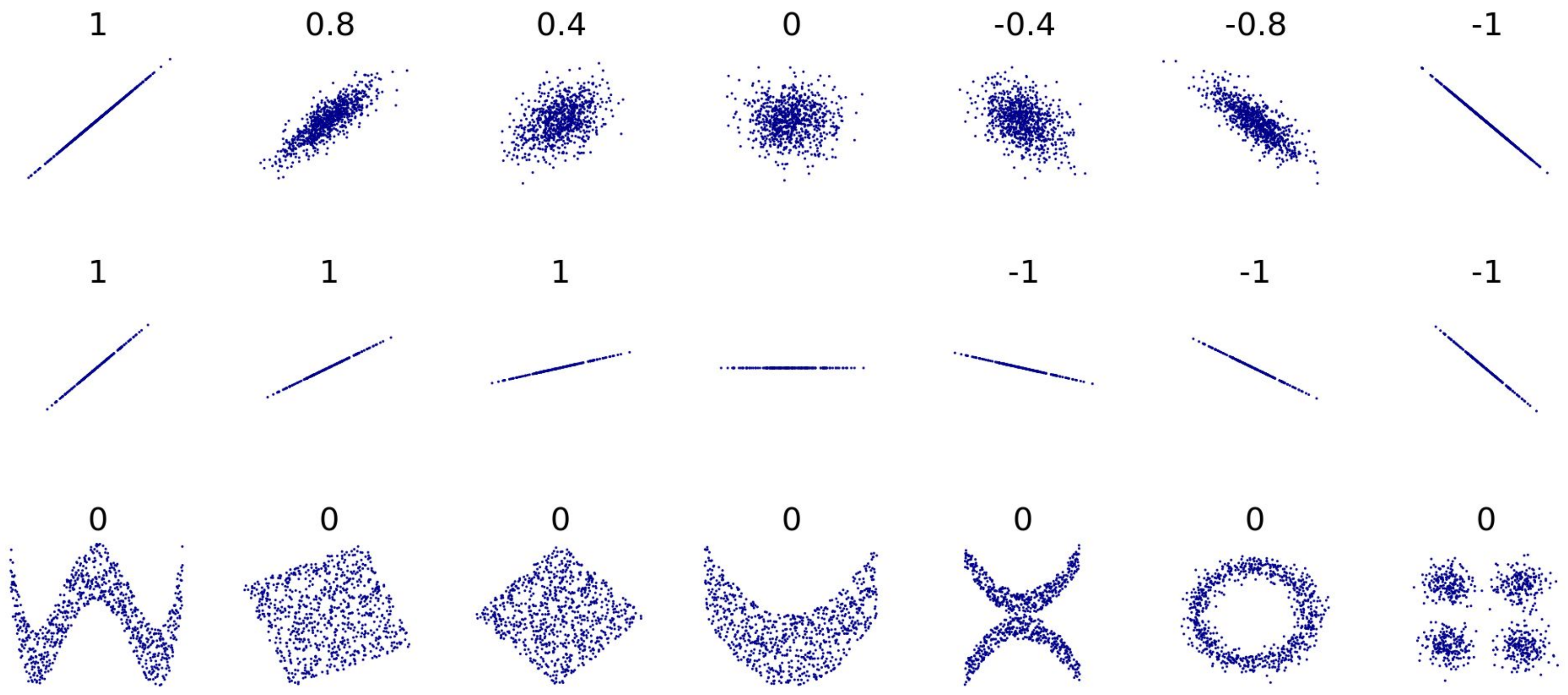
$$\rho_{X,Y} = \mathrm{corr}(X,Y) = \frac{\mathrm{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{\mathrm{E}[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y}$$



```
# Find correlations with the target and sort
correlations = app_train.corr()['TARGET'].sort_values()

# Display correlations
print('Most Positive Correlations:\n', correlations.tail(15))
print('\nMost Negative Correlations:\n', correlations.head(15))
```

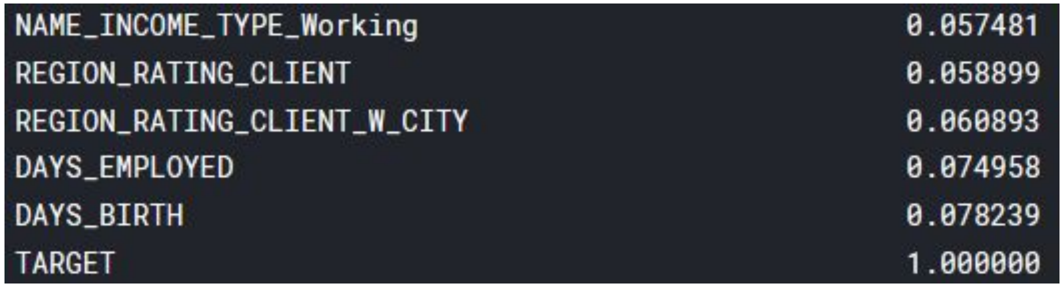
Data exploration > Feature effects > Correlations

In statistics, correlation or dependence is any statistical relationship, whether causal or not, between two random variables or bivariate data. In the broadest sense correlation is any statistical association, though it commonly refers to the degree to which a pair of variables are **linearly** related. [Wikipedia]

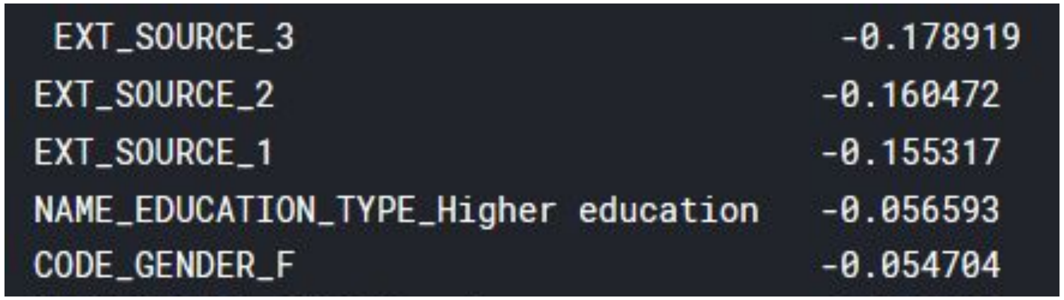


Data exploration > Feature effects > Correlations

- .00-.19 “very weak”
- .20-.39 “weak”
- .40-.59 “moderate”
- .60-.79 “strong”
- .80-1.0 “very strong”



```
NAME_INCOME_TYPE_Working                0.057481
REGION_RATING_CLIENT                    0.058899
REGION_RATING_CLIENT_W_CITY             0.060893
DAYS_EMPLOYED                           0.074958
DAYS_BIRTH                              0.078239
TARGET                                  1.000000
```



```
 EXT_SOURCE_3                          -0.178919
EXT_SOURCE_2                          -0.160472
EXT_SOURCE_1                          -0.155317
NAME_EDUCATION_TYPE_Higher education  -0.056593
CODE_GENDER_F                         -0.054704
```

Data exploration > Feature effects > Collinearity

Collinear variables are those which are highly correlated with one another (strong linear relationship).

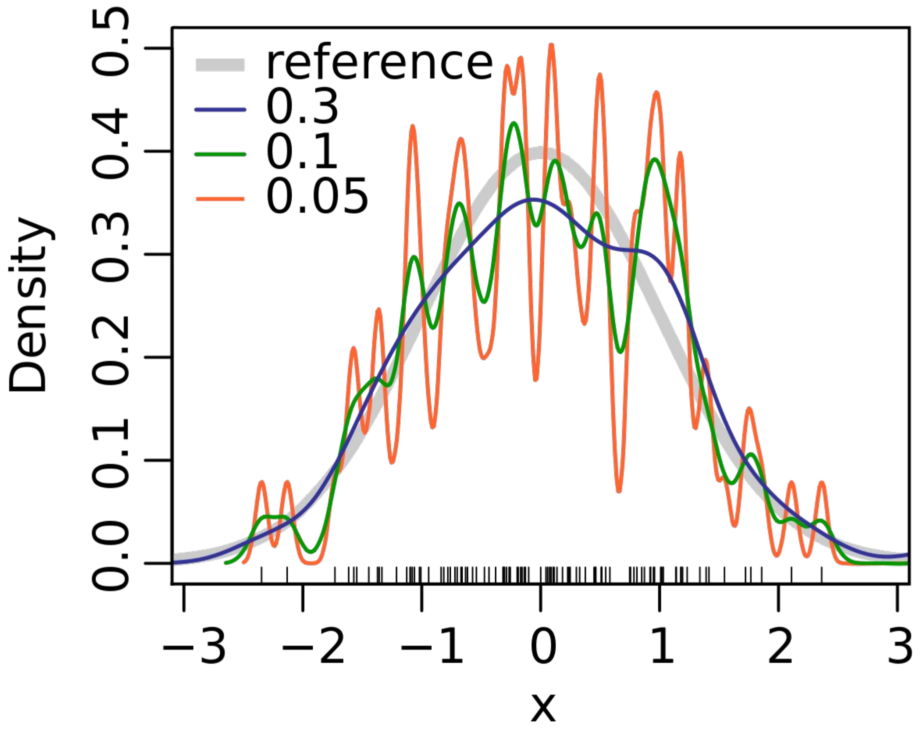
Two variables X_1 and X_2 are said to be perfectly collinear if:

$$X_1 = X_2 + \text{constant}$$

These can decrease the model's availablility to learn, decrease model interpretability, and decrease generalization performance on the test set.

Data exploration > Feature effects > Kernel Density Estimate

In statistics, kernel density estimation (KDE) is a non-parametric way to estimate the probability density function of a random variable. Kernel density estimation is a fundamental data smoothing problem where inferences about the population are made, based on a finite data sample. [Wikipedia]

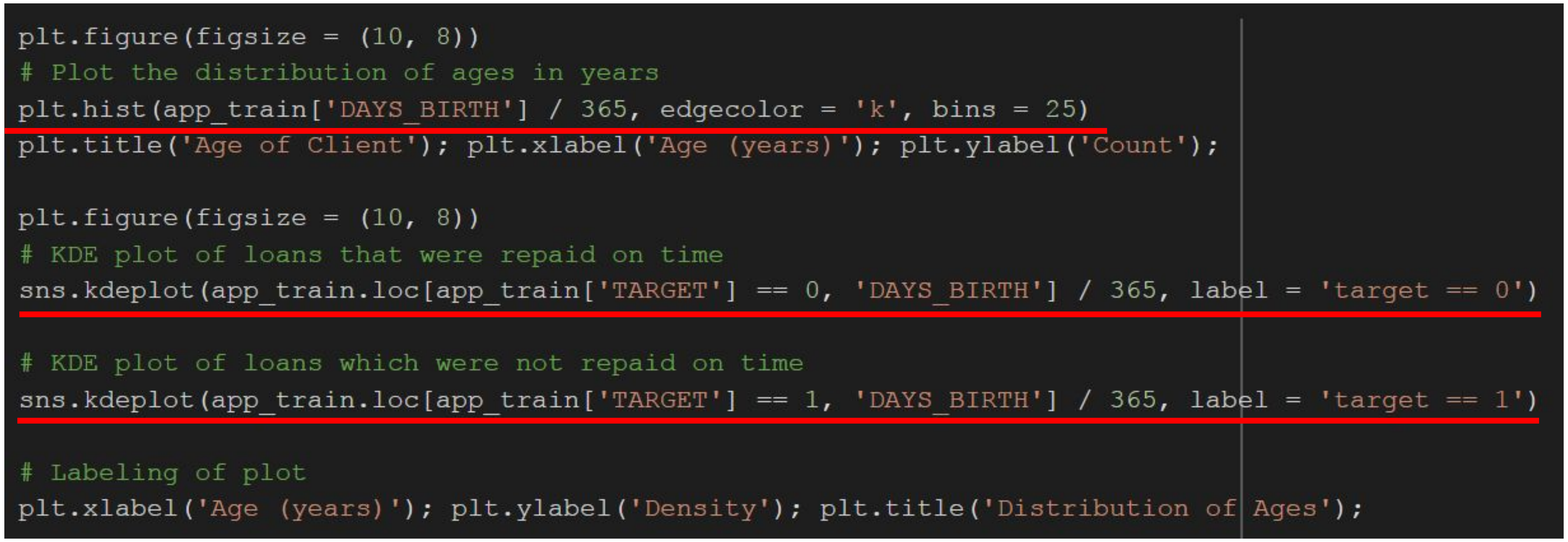


$$\widehat{f}_h(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x - x_i) = \frac{1}{nh}\sum_{i=1}^{n} K\Big(\frac{x - x_i}{h}\Big),$$

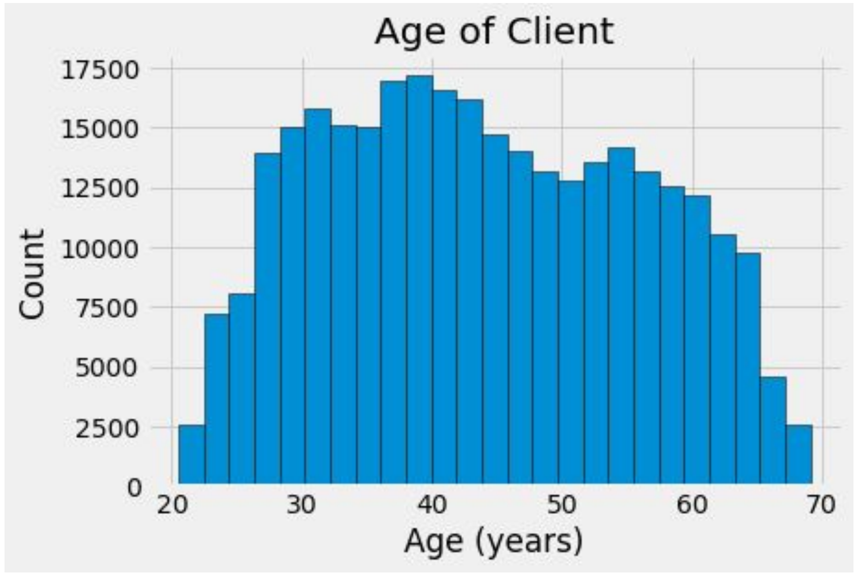
where K is the kernel — a non-negative function

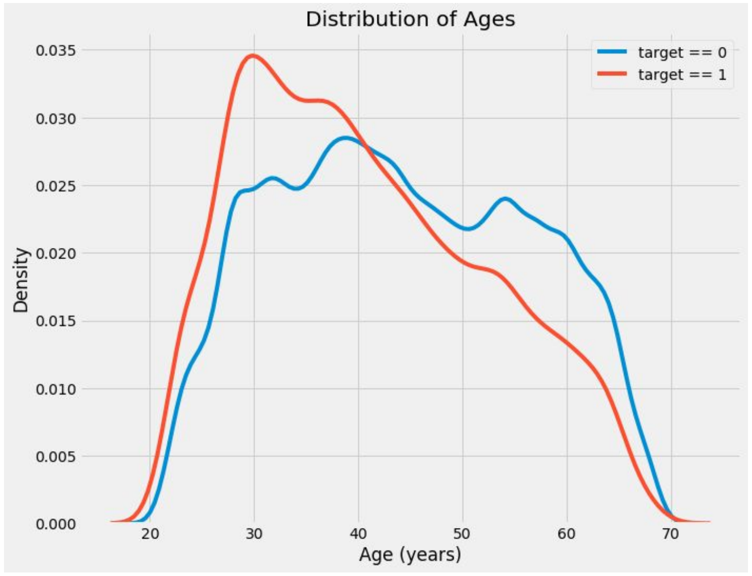
and h > 0 is a smoothing parameter called the bandwidth.

Data exploration > Feature effects > Kernel Density Estimate



```
plt.figure(figsize = (10, 8))
# Plot the distribution of ages in years
plt.hist(app_train['DAYS_BIRTH'] / 365, edgecolor = 'k', bins = 25)
plt.title('Age of Client'); plt.xlabel('Age (years)'); plt.ylabel('Count');

plt.figure(figsize = (10, 8))
# KDE plot of loans that were repaid on time
sns.kdeplot(app_train.loc[app_train['TARGET'] == 0, 'DAYS_BIRTH'] / 365, label = 'target == 0')

# KDE plot of loans which were not repaid on time
sns.kdeplot(app_train.loc[app_train['TARGET'] == 1, 'DAYS_BIRTH'] / 365, label = 'target == 1')

# Labeling of plot
plt.xlabel('Age (years)'); plt.ylabel('Density'); plt.title('Distribution of Ages');
```

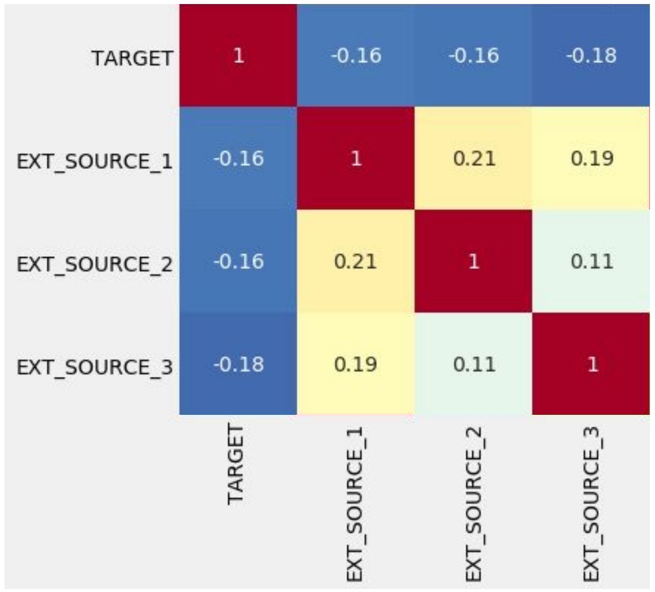




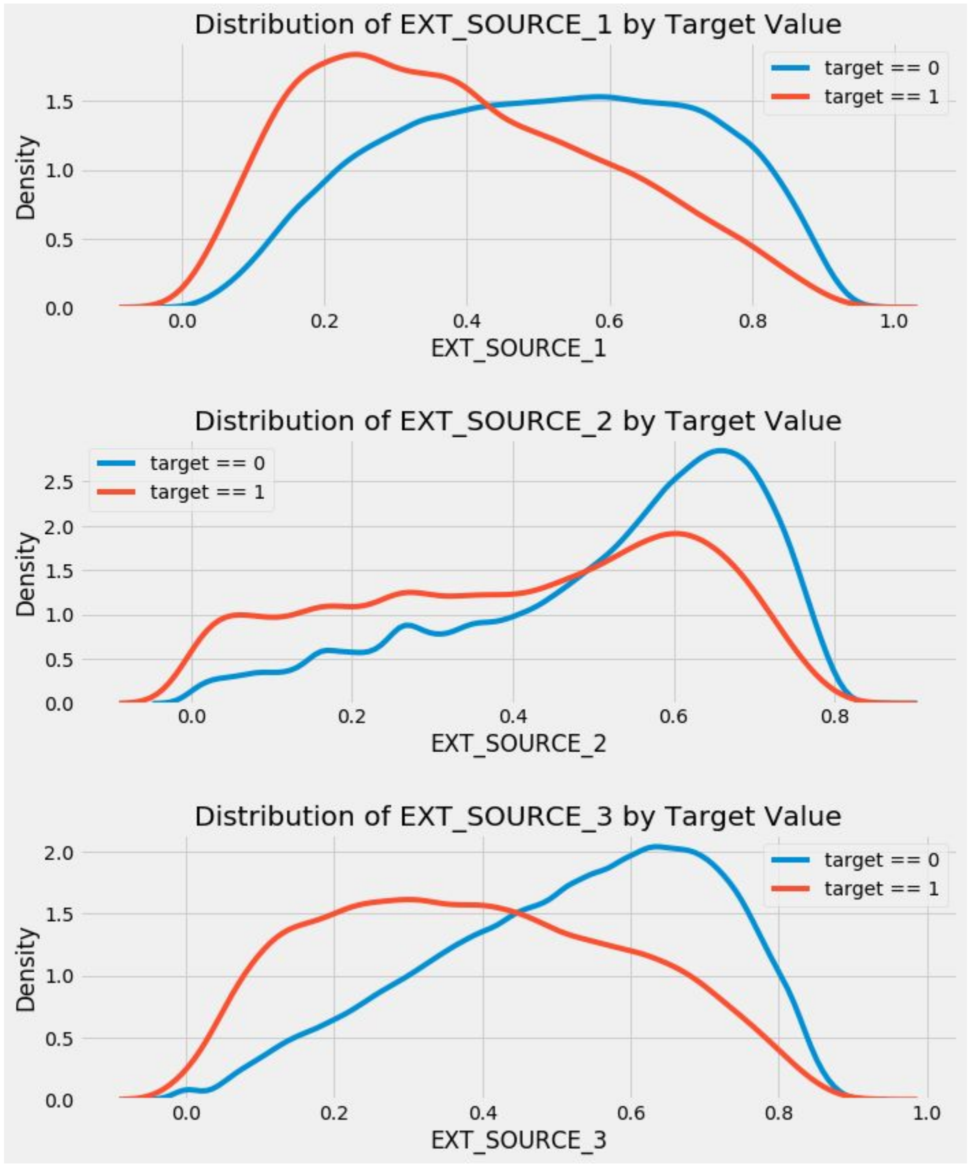
Data exploration > Feature effects > Kernel Density Estimate

We do the same with the features

'EXT_SOURCE_1',
'EXT_SOURCE_2',
'EXT_SOURCE_3'



	TARGET	EXT_SOURCE_1	EXT_SOURCE_2	EXT_SOURCE_3
TARGET	1	-0.16	-0.16	-0.18
EXT_SOURCE_1	-0.16	1	0.21	0.19
EXT_SOURCE_2	-0.16	0.21	1	0.11
EXT_SOURCE_3	-0.18	0.19	0.11	1



Next lecture

Feature Engineering

Machine Learning Model

Deployment

Practice!

https://colab.research.google.com/drive/1FY9OuKp2hCtWdW1OFUv7S0i78pCLId-E

https://drive.google.com/drive/folders/1cSq-3a3KusGU05Qb9UQMkKXqH9posbeT?usp=sharing

Project

Part 1 - Data preparation

Preparation

To read on your own

- **Mandatory:** read the “10 minutes to pandas” tutorial
  □ https://pandas.pydata.org/pandas-docs/stable/user_guide/10min.html

- Recommended for the project but optional: browse through the full pandas user guide
  □ https://pandas.pydata.org/pandas-docs/stable/user_guide/index.html

To do on your own

1. Set up a working Python Notebook
   - Using Jupyter Notebooks: https://www.dataquest.io/blog/jupyter-notebook-tutorial/ or www.datacamp.com/community/tutorials/tutorial-jupyter-notebook
   - OR Using Google Colab http://colab.research.google.com

2. Download notebooks and ensure that the libraries presented in the lecture are installed
   - The project notebook will be on Moodle
   - You can use the course notebook to help you (see URL in a previous slide)
   - Import numpy, pandas, and sklearn

3. Fill the cell with code or text answering the questions

Feature Engineering > Polynomial Features

Feature Engineering

Feature Engineering > Aggregation

Feature Engineering

Feature Engineering > Smoothing

Feature Engineering

Feature Engineering > Sampling

Feature Engineering