> **Algoritmos para Big Data**

> **2023/24**

Binary classification

This lecture is about binary classification in a discrete space. We will setup a ML processing pipeline to achieve our goals. The dataset relates to the domain of banking industry, specifically about credit risk.

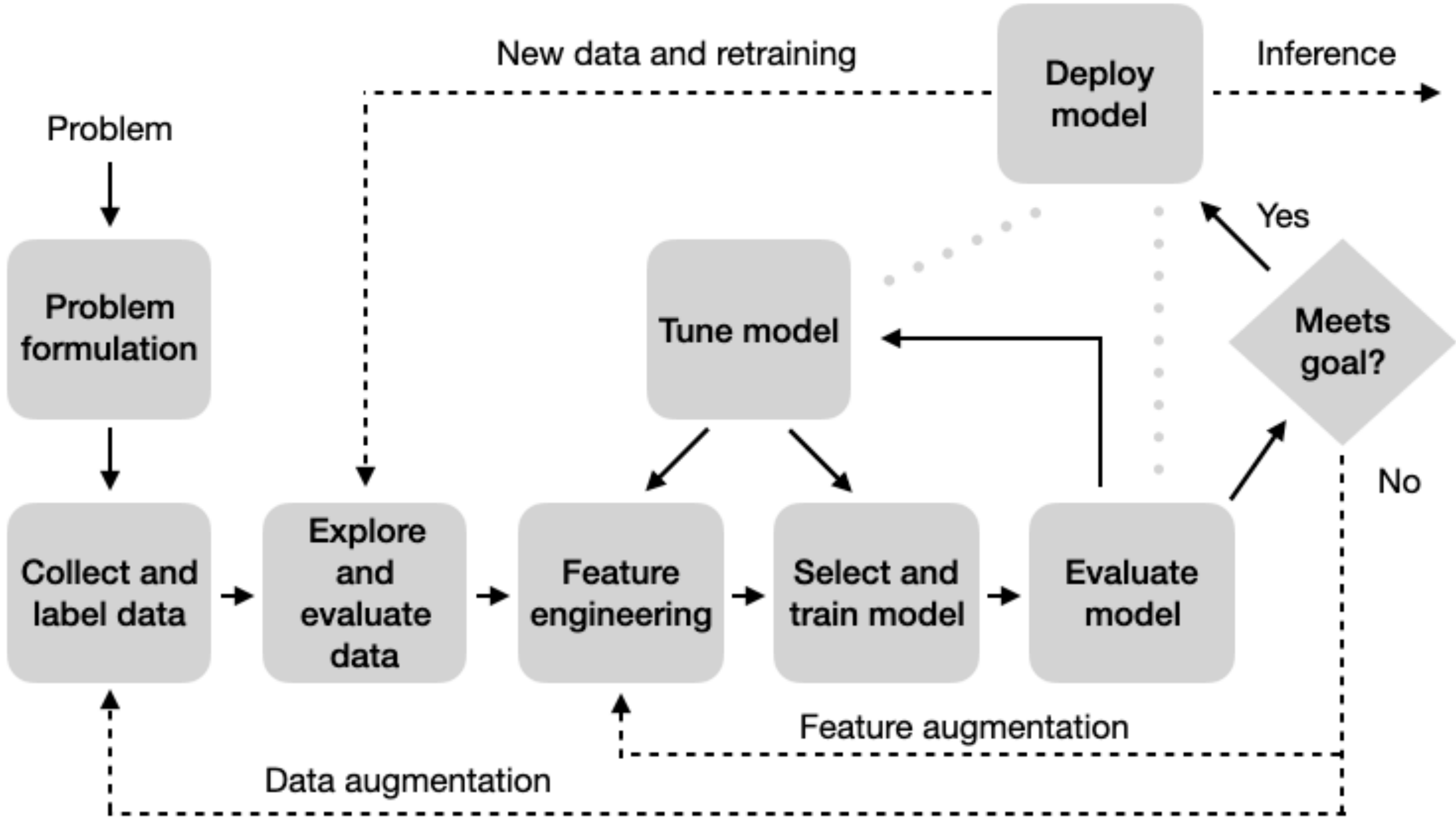
ML pipelines

As stated in the Spark's programming guide, **"ML Pipelines provide a uniform set of high-level APIs built on top of DataFrames that help users create and tune practical machine learning pipelines."**

Hence, it is possible to combine multiple algorithms into a single pipeline, or workflow. Besides DataFrames, it involves the following:

1. Transformer: an algorithm which can transform one DataFrame into another DataFrame. For example, an ML model is a Transformer which transforms a DataFrame with features into a DataFrame with predictions.
2. Estimator: an algorithm which can be fit on a DataFrame to produce a Transformer. For example, a learning algorithm is an Estimator which trains on a DataFrame and produces a model.
3. Pipeline: the way to chain multiple Transformers and Estimators together to specify an ML workflow. 4. Parameter: all Transformers and Estimators share a common API for specifying parameters. Further details can be found in http://spark.apache.org/docs/latest/ml-pipeline.html

Recall that, in general, a typical ML workflow is designed to work as depicted below:



Problem formulation

This exercise is about home credit default risk. Our case-study is based on a Kaggle dataset that has been used in a competition. Details can be found in

> https://www.kaggle.com/competitions/home-credit-default-risk/ .

The goal is predict if a particular credit application might face payment difficulties or not. This is shown by a feature called `TARGET`. In the end, it is a binary classification problem.

Basically, the functional requirements for the Spark program we are about to create are as follows:

1. To load the datasets under analysis and making sure it can be further processed by a ML classifier.
2. To create a classification model supported by a SVM algorithm that is fit for the purpose.
3. To evaluate the quality of the classifier that has been built.

As for data availability, you can find the archive **home-credit-default.zip** in the location

> `https://bigdata.iscte.me/abd/home-credit-default.zip` .

To solve the problem, we focus on three data files from the zip archive:

- HomeCredit_columns_description.csv
- application_train.csv
- application_test.csv

The application_train.csv and similar application_test.csv contain the most important features.

Later on, as an additional exercise, you may use all the given data to enhance the predictive power of the model.

Information collected from the site of the competition

application_{train|test}.csv

This is the main table, broken into two files for Train (with TARGET) and Test (without TARGET). Static data for all applications. One row represents one loan in our data sample.

bureau.csv

All client's previous credits provided by other financial institutions that were reported to Credit Bureau (for clients who have a loan in our sample). For every loan in our sample, there are as many rows as number of credits the client had in Credit Bureau before the application date.

bureau_balance.csv

Monthly balances of previous credits in Credit Bureau. This table has one row for each month of history of every previous credit reported to Credit Bureau – i.e the table has (#loans in sample * # of relative previous credits * # of months where we have some history observable for the previous credits) rows.

POS_CASH_balance.csv

Monthly balance snapshots of previous POS (point of sales) and cash loans that the applicant had with Home Credit. This table has one row for each month of history of every previous credit in Home Credit (consumer credit and cash loans) related to loans in our sample – i.e. the table has (#loans in sample * # of relative previous credits * # of months in which we have some history observable for the previous credits) rows.

credit_card_balance.csv

Monthly balance snapshots of previous credit cards that the applicant has with Home Credit. This table has one row for each month of history of every previous credit in Home Credit (consumer credit and cash loans) related to loans in our sample – i.e. the table has (#loans in sample * # of relative previous credit cards * # of months where we have some history observable for the previous credit card) rows.

previous_application.csv

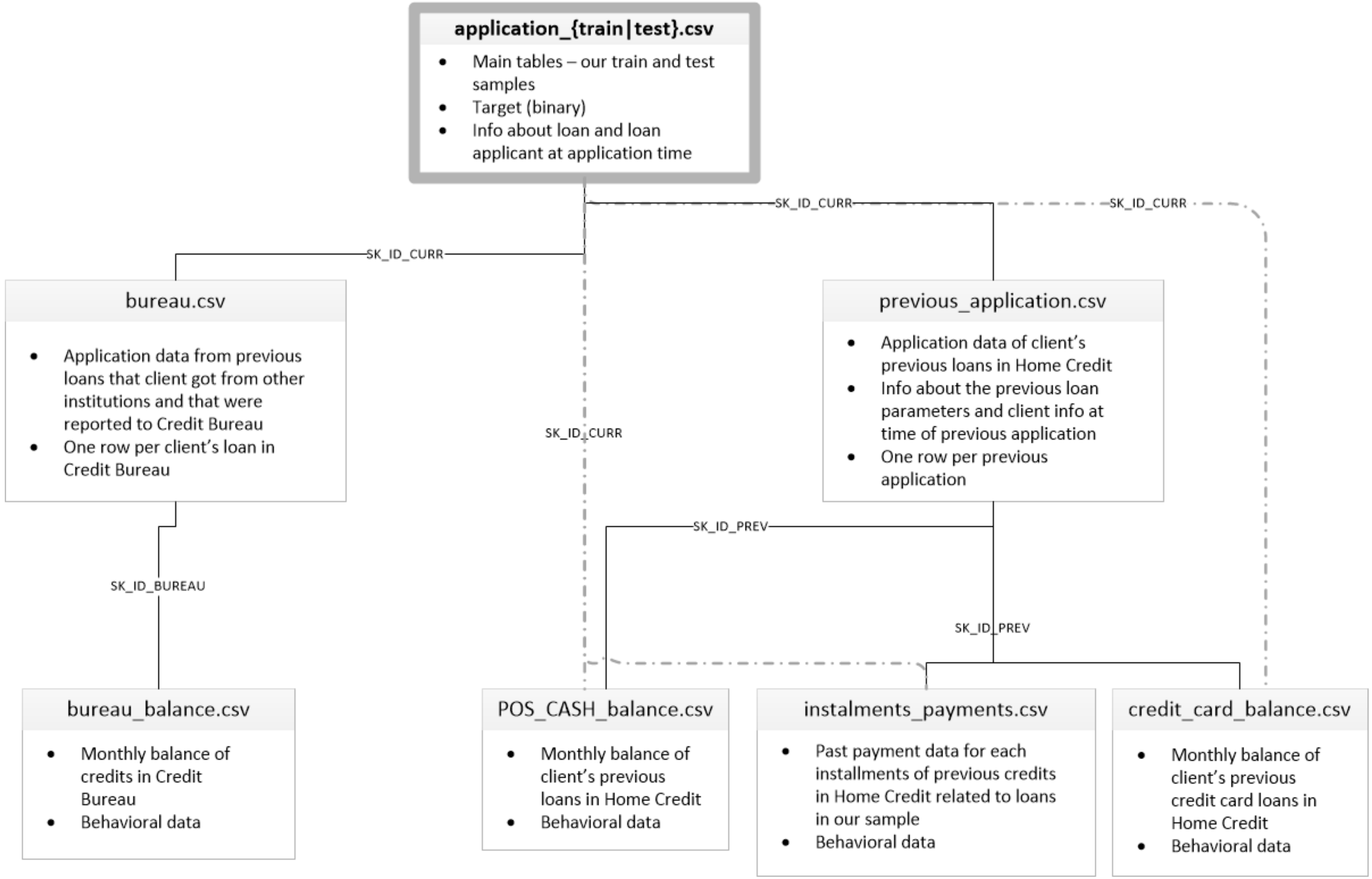
All previous applications for Home Credit loans of clients who have loans in our sample. There is one row for each previous application related to loans in our data sample.

installments_payments.csv

Repayment history for the previously disbursed credits in Home Credit related to the loans in our sample. There is a) one row for every payment that was made plus b) one row each for missed payment. One row is equivalent to one payment of one installment OR one installment corresponding to one payment of one previous Home Credit credit related to loans in our sample.

HomeCredit_columns_description.csv

This file contains descriptions for the columns in the various data files.



Initial settings

Prior to any computation, let us deal with required imports and create a Spark session, as well as defining useful functions.

Additional packages

If we need, we can install more packages, e.g. matplotlib. We suggest to execute the commands in a Terminal.

Furthermore, it is worth checking commands to deal with installing packages in the environment. For example:

1. List all packages in the current environment: `conda list`
2. List all packages installed into the environment `pyspark_env` : `conda list -n pyspark_env`
3. Save packages for future use: `conda list –export > package-list.txt`
4. Reinstall packages from an export file: `conda create -n pyspark_env –file package-list.txt`

In [1]:
```
import findspark, pyspark

from pyspark.sql import SparkSession
from pyspark.sql.types import *
import pyspark.sql.functions as F
```

In [2]:
```
import os

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

In [3]:
```
# Create the Spark session

findspark.init()
findspark.find()

spark = SparkSession\
        .builder\
        .appName("HomeCreditDefaultRisk")\
        .config("spark.sql.shuffle.partitions",6)\
        .config("spark.sql.repl.eagereval.enabled",True)\
```

```
        .getOrCreate()
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/03/13 23:26:16 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-j
```

In [4]:

```

```

Out[4]: **SparkSession - in-memory**

SparkContext

Spark UI

Version	`v3.5.0`
Master	`local[*]`
AppName	`HomeCreditDefaultRisk`

In [5]:

```
import sys

from pyspark.ml import Pipeline
from pyspark.ml.stat import Correlation
from pyspark.ml.feature import VectorAssembler, StringIndexer, OneHotEncoder
from pyspark.ml.classification import LinearSVC
from pyspark.ml.evaluation import BinaryClassificationEvaluator
```

In [6]:

```
from IPython.core.display import HTML
display(HTML("<style>pre { white-space: pre !important; }</style>"))
```

Useful functions

The visualization functions below rely on Seaborn to plot data but as Python data frame.

See https://seaborn.pydata.org/index.html

We encourage you to use your own plotting functions. Remember: *"A picture is worth a thousand words"*

In [7]:
```
def plotHorizBar(df, xcol, ycol, colour):
    return sns.barplot(data=df, x=xcol, y=ycol, color=colour)
```

In [8]:
```
def plotLine(df, xcol, ycol):
    return sns.lineplot(data=df, x=xcol, y=ycol)
```

In [9]:
```
def plotBar(df, xcol, ycol, huecol=None):
    return sns.barplot(data=df, x=xcol, y=ycol, hue=huecol)
```

In [10]:
```
def plotScatter(df, xcol, ycol, huecol=None):
    return sns.scatterplot(data=df, x=xcol, y=ycol, hue=huecol)
```

In [11]:
```
def plotScatterMatrix(df, huecol=None):
    return sns.pairplot(data=df, hue=huecol)
```

In [12]:
```
def plotCorrelationMatrix(df, annot=False):
    # compute the correlation matrix
    corr = df.corr()

    # generate a mask for the upper triangle
    mask = np.triu(np.ones_like(corr, dtype=bool))

    # set up the matplotlib figure
    f, ax = plt.subplots(figsize=(11, 9))

    # generate a custom colormap
    #cmap = sns.divergent_palette(230, 20, as_cmap=True)

    cmap='coolwarm'

    # draw the heatmap with the mask and correct aspect ratio
    return sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0, annot=annot,
                square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

In [13]:
```
def plotBox(df, xcol, ycol, huecol=None, kind='box'):
    return sns.catplot(data=df, x=xcol, y=ycol, hue=huecol, kind=kind)
```

In [14]:
```
# Function to get columns of numeric type in a DataFrame

def numeric_columns(df):
    cls_numeric = []
    for x, t in df.dtypes:
        if t in ['int', 'double']:
            cls_numeric.append(x)
    return cls_numeric
```

In [15]:
```
# Function to figure out the profile of nulls and uniques for ecah column in a DataFrame

def compute_nulls_and_uniques(df, cols):
    total = df.count()
    results = []
    for cl in cols:
        knulls = df.select(cl).filter(F.col(cl).isNull()).count()
        knullsperc = knulls / total
        knans = df.select(cl).filter(F.isnan(cl)).count()
        knansperc = knans / total
        kuniques = df.select(cl).distinct().count()
        kuniquesperc = kuniques / total
        results.append(Row(feature = cl, count_nulls = knulls, percentage_nulls = knullsperc,
                           count_nans = knans, percentage_nans = knansperc,
                           count_uniques = kuniques, percentage_uniques = kuniquesperc))

    return spark.createDataFrame(results)
```

Collect and label data

Data ingestion

Checking working diretory and data files

In [ ]:
```
pwd
```

In [17]:
```
data_dir =
```

In [ ]:
```
ls -la $data_dir
```

In [ ]:
```
# Alternative command to list data files

print(os.listdir(data_dir))
```

In [20]:
```
! head -n 2
```

```
Id,Table,Row,Description,Special
1,application_{train|test}.csv,SK_ID_CURR,ID of loan in our sample,
```

In [21]:
```
! head -n 2
```

```
SK_ID_CURR,TARGET,NAME_CONTRACT_TYPE,CODE_GENDER,FLAG_OWN_CAR,FLAG_OWN_REALTY,CNT_CHILDREN,AMT_INCOME_TOTAL,AMT_CR
100002,1,Cash loans,M,N,Y,0,202500.0,406597.5,24700.5,351000.0,Unaccompanied,Working,Secondary / secondary special
```

In [22]:
```
! head -n 2
```

```
SK_ID_CURR,NAME_CONTRACT_TYPE,CODE_GENDER,FLAG_OWN_CAR,FLAG_OWN_REALTY,CNT_CHILDREN,AMT_INCOME_TOTAL,AMT_CREDIT,AM
100001,Cash loans,F,N,Y,0,135000.0,568800.0,20560.5,450000.0,Unaccompanied,Working,Higher education,Married,House
```

Reading the datasets

In [23]:
```
filename = data_dir + "HomeCredit_columns_description.csv"
df_HomeCredit_columns_description = spark.read.csv(filename, header="true", inferSchema="true", sep=',')
```

In [24]:
```
filename = data_dir + "application_test.csv"
df_application_test =
```

In [25]:

```
filename = data_dir + "application_train.csv"
df_application_train =
```

Checking data

Schema, show and count

In [26]:

```
```

```
root
 |-- Id: integer (nullable = true)
 |-- Table: string (nullable = true)
 |-- Row: string (nullable = true)
 |-- Description: string (nullable = true)
 |-- Special: string (nullable = true)

+---+-----------------------------+---------------------------+----------------------------------------------------------
|Id |Table                        |Row                        |Description
+---+-----------------------------+---------------------------+----------------------------------------------------------
|1  |application_{train|test}.csv|SK_ID_CURR                 |ID of loan in our sample
|2  |application_{train|test}.csv|TARGET                     |Target variable (1 - client with payment difficulties
|5  |application_{train|test}.csv|NAME_CONTRACT_TYPE         |Identification if loan is cash or revolving
|6  |application_{train|test}.csv|CODE_GENDER                |Gender of the client
|7  |application_{train|test}.csv|FLAG_OWN_CAR               |Flag if the client owns a car
|8  |application_{train|test}.csv|FLAG_OWN_REALTY            |Flag if client owns a house or flat
|9  |application_{train|test}.csv|CNT_CHILDREN               |Number of children the client has
|10 |application_{train|test}.csv|AMT_INCOME_TOTAL           |Income of the client
|11 |application_{train|test}.csv|AMT_CREDIT                 |Credit amount of the loan
|12 |application_{train|test}.csv|AMT_ANNUITY                |Loan annuity
|13 |application_{train|test}.csv|AMT_GOODS_PRICE            |For consumer loans it is the price of the goods for w
|14 |application_{train|test}.csv|NAME_TYPE_SUITE            |Who was accompanying client when he was applying for
|15 |application_{train|test}.csv|NAME_INCOME_TYPE           |Clients income type (businessman, working, maternity
|16 |application_{train|test}.csv|NAME_EDUCATION_TYPE        |Level of highest education the client achieved
|17 |application_{train|test}.csv|NAME_FAMILY_STATUS         |Family status of the client
|18 |application_{train|test}.csv|NAME_HOUSING_TYPE          |What is the housing situation of the client (renting,
|19 |application_{train|test}.csv|REGION_POPULATION_RELATIVE|Normalized population of region where client lives (h
|20 |application_{train|test}.csv|DAYS_BIRTH                 |Client's age in days at the time of application
|21 |application_{train|test}.csv|DAYS_EMPLOYED              |How many days before the application the person start
|22 |application_{train|test}.csv|DAYS_REGISTRATION          |How many days before the application did client chang
+---+-----------------------------+---------------------------+----------------------------------------------------------
only showing top 20 rows
```

Out[26]: 219

In [27]:

```
root
```

```
 |-- SK_ID_CURR: integer (nullable = true)
 |-- TARGET: integer (nullable = true)
 |-- NAME_CONTRACT_TYPE: string (nullable = true)
 |-- CODE_GENDER: string (nullable = true)
 |-- FLAG_OWN_CAR: string (nullable = true)
 |-- FLAG_OWN_REALTY: string (nullable = true)
 |-- CNT_CHILDREN: integer (nullable = true)
 |-- AMT_INCOME_TOTAL: double (nullable = true)
 |-- AMT_CREDIT: double (nullable = true)
 |-- AMT_ANNUITY: double (nullable = true)
 |-- AMT_GOODS_PRICE: double (nullable = true)
 |-- NAME_TYPE_SUITE: string (nullable = true)
 |-- NAME_INCOME_TYPE: string (nullable = true)
 |-- NAME_EDUCATION_TYPE: string (nullable = true)
 |-- NAME_FAMILY_STATUS: string (nullable = true)
 |-- NAME_HOUSING_TYPE: string (nullable = true)
 |-- REGION_POPULATION_RELATIVE: double (nullable = true)
 |-- DAYS_BIRTH: integer (nullable = true)
 |-- DAYS_EMPLOYED: integer (nullable = true)
 |-- DAYS_REGISTRATION: double (nullable = true)
 |-- DAYS_ID_PUBLISH: integer (nullable = true)
 |-- OWN_CAR_AGE: double (nullable = true)
 |-- FLAG_MOBIL: integer (nullable = true)
 |-- FLAG_EMP_PHONE: integer (nullable = true)
 |-- FLAG_WORK_PHONE: integer (nullable = true)
 |-- FLAG_CONT_MOBILE: integer (nullable = true)
 |-- FLAG_PHONE: integer (nullable = true)
 |-- FLAG_EMAIL: integer (nullable = true)
 |-- OCCUPATION_TYPE: string (nullable = true)
 |-- CNT_FAM_MEMBERS: double (nullable = true)
 |-- REGION_RATING_CLIENT: integer (nullable = true)
 |-- REGION_RATING_CLIENT_W_CITY: integer (nullable = true)
 |-- WEEKDAY_APPR_PROCESS_START: string (nullable = true)
 |-- HOUR_APPR_PROCESS_START: integer (nullable = true)
 |-- REG_REGION_NOT_LIVE_REGION: integer (nullable = true)
 |-- REG_REGION_NOT_WORK_REGION: integer (nullable = true)
 |-- LIVE_REGION_NOT_WORK_REGION: integer (nullable = true)
 |-- REG_CITY_NOT_LIVE_CITY: integer (nullable = true)
 |-- REG_CITY_NOT_WORK_CITY: integer (nullable = true)
```

```
 |-- LIVE_CITY_NOT_WORK_CITY: integer (nullable = true)
 |-- ORGANIZATION_TYPE: string (nullable = true)
 |-- EXT_SOURCE_1: double (nullable = true)
 |-- EXT_SOURCE_2: double (nullable = true)
 |-- EXT_SOURCE_3: double (nullable = true)
 |-- APARTMENTS_AVG: double (nullable = true)
 |-- BASEMENTAREA_AVG: double (nullable = true)
 |-- YEARS_BEGINEXPLUATATION_AVG: double (nullable = true)
 |-- YEARS_BUILD_AVG: double (nullable = true)
 |-- COMMONAREA_AVG: double (nullable = true)
 |-- ELEVATORS_AVG: double (nullable = true)
 |-- ENTRANCES_AVG: double (nullable = true)
 |-- FLOORSMAX_AVG: double (nullable = true)
 |-- FLOORSMIN_AVG: double (nullable = true)
 |-- LANDAREA_AVG: double (nullable = true)
 |-- LIVINGAPARTMENTS_AVG: double (nullable = true)
 |-- LIVINGAREA_AVG: double (nullable = true)
 |-- NONLIVINGAPARTMENTS_AVG: double (nullable = true)
 |-- NONLIVINGAREA_AVG: double (nullable = true)
 |-- APARTMENTS_MODE: double (nullable = true)
 |-- BASEMENTAREA_MODE: double (nullable = true)
 |-- YEARS_BEGINEXPLUATATION_MODE: double (nullable = true)
 |-- YEARS_BUILD_MODE: double (nullable = true)
 |-- COMMONAREA_MODE: double (nullable = true)
 |-- ELEVATORS_MODE: double (nullable = true)
 |-- ENTRANCES_MODE: double (nullable = true)
 |-- FLOORSMAX_MODE: double (nullable = true)
 |-- FLOORSMIN_MODE: double (nullable = true)
 |-- LANDAREA_MODE: double (nullable = true)
 |-- LIVINGAPARTMENTS_MODE: double (nullable = true)
 |-- LIVINGAREA_MODE: double (nullable = true)
 |-- NONLIVINGAPARTMENTS_MODE: double (nullable = true)
 |-- NONLIVINGAREA_MODE: double (nullable = true)
 |-- APARTMENTS_MEDI: double (nullable = true)
 |-- BASEMENTAREA_MEDI: double (nullable = true)
 |-- YEARS_BEGINEXPLUATATION_MEDI: double (nullable = true)
 |-- YEARS_BUILD_MEDI: double (nullable = true)
 |-- COMMONAREA_MEDI: double (nullable = true)
 |-- ELEVATORS_MEDI: double (nullable = true)
```

```
 |-- ENTRANCES_MEDI: double (nullable = true)
 |-- FLOORSMAX_MEDI: double (nullable = true)
 |-- FLOORSMIN_MEDI: double (nullable = true)
 |-- LANDAREA_MEDI: double (nullable = true)
 |-- LIVINGAPARTMENTS_MEDI: double (nullable = true)
 |-- LIVINGAREA_MEDI: double (nullable = true)
 |-- NONLIVINGAPARTMENTS_MEDI: double (nullable = true)
 |-- NONLIVINGAREA_MEDI: double (nullable = true)
 |-- FONDKAPREMONT_MODE: string (nullable = true)
 |-- HOUSETYPE_MODE: string (nullable = true)
 |-- TOTALAREA_MODE: double (nullable = true)
 |-- WALLSMATERIAL_MODE: string (nullable = true)
 |-- EMERGENCYSTATE_MODE: string (nullable = true)
 |-- OBS_30_CNT_SOCIAL_CIRCLE: double (nullable = true)
 |-- DEF_30_CNT_SOCIAL_CIRCLE: double (nullable = true)
 |-- OBS_60_CNT_SOCIAL_CIRCLE: double (nullable = true)
 |-- DEF_60_CNT_SOCIAL_CIRCLE: double (nullable = true)
 |-- DAYS_LAST_PHONE_CHANGE: double (nullable = true)
 |-- FLAG_DOCUMENT_2: integer (nullable = true)
 |-- FLAG_DOCUMENT_3: integer (nullable = true)
 |-- FLAG_DOCUMENT_4: integer (nullable = true)
 |-- FLAG_DOCUMENT_5: integer (nullable = true)
 |-- FLAG_DOCUMENT_6: integer (nullable = true)
 |-- FLAG_DOCUMENT_7: integer (nullable = true)
 |-- FLAG_DOCUMENT_8: integer (nullable = true)
 |-- FLAG_DOCUMENT_9: integer (nullable = true)
 |-- FLAG_DOCUMENT_10: integer (nullable = true)
 |-- FLAG_DOCUMENT_11: integer (nullable = true)
 |-- FLAG_DOCUMENT_12: integer (nullable = true)
 |-- FLAG_DOCUMENT_13: integer (nullable = true)
 |-- FLAG_DOCUMENT_14: integer (nullable = true)
 |-- FLAG_DOCUMENT_15: integer (nullable = true)
 |-- FLAG_DOCUMENT_16: integer (nullable = true)
 |-- FLAG_DOCUMENT_17: integer (nullable = true)
 |-- FLAG_DOCUMENT_18: integer (nullable = true)
 |-- FLAG_DOCUMENT_19: integer (nullable = true)
 |-- FLAG_DOCUMENT_20: integer (nullable = true)
 |-- FLAG_DOCUMENT_21: integer (nullable = true)
 |-- AMT_REQ_CREDIT_BUREAU_HOUR: double (nullable = true)
```

```
 |-- AMT_REQ_CREDIT_BUREAU_DAY: double (nullable = true)
 |-- AMT_REQ_CREDIT_BUREAU_WEEK: double (nullable = true)
 |-- AMT_REQ_CREDIT_BUREAU_MON: double (nullable = true)
 |-- AMT_REQ_CREDIT_BUREAU_QRT: double (nullable = true)
 |-- AMT_REQ_CREDIT_BUREAU_YEAR: double (nullable = true)
```

```
24/03/13 23:26:23 WARN SparkStringUtils: Truncated the string representation of a plan since it was too large. Thi
```

```
+----------+------+------------------+-----------+------------+---------------+------------+----------------+-----
|SK_ID_CURR|TARGET|NAME_CONTRACT_TYPE|CODE_GENDER|FLAG_OWN_CAR|FLAG_OWN_REALTY|CNT_CHILDREN|AMT_INCOME_TOTAL|AMT_C
+----------+------+------------------+-----------+------------+---------------+------------+----------------+-----
|    100002|     1|        Cash loans|          M|           N|              Y|           0|        202500.0|  406
|    100003|     0|        Cash loans|          F|           N|              N|           0|        270000.0| 1293
|    100004|     0|   Revolving loans|          M|           Y|              Y|           0|         67500.0|  135
|    100006|     0|        Cash loans|          F|           N|              Y|           0|        135000.0|  312
|    100007|     0|        Cash loans|          M|           N|              Y|           0|        121500.0|  513
|    100008|     0|        Cash loans|          M|           N|              Y|           0|         99000.0|  490
|    100009|     0|        Cash loans|          F|           Y|              Y|           1|        171000.0| 1560
|    100010|     0|        Cash loans|          M|           Y|              Y|           0|        360000.0| 1530
|    100011|     0|        Cash loans|          F|           N|              Y|           0|        112500.0| 1019
|    100012|     0|   Revolving loans|          M|           N|              Y|           0|        135000.0|  405
|    100014|     0|        Cash loans|          F|           N|              Y|           1|        112500.0|  652
|    100015|     0|        Cash loans|          F|           N|              Y|           0|       38419.155|  148
|    100016|     0|        Cash loans|          F|           N|              Y|           0|         67500.0|   80
|    100017|     0|        Cash loans|          M|           Y|              N|           1|        225000.0|  918
|    100018|     0|        Cash loans|          F|           N|              Y|           0|        189000.0|  773
|    100019|     0|        Cash loans|          M|           Y|              Y|           0|        157500.0|  299
|    100020|     0|        Cash loans|          M|           N|              N|           0|        108000.0|  509
|    100021|     0|   Revolving loans|          F|           N|              Y|           1|         81000.0|  270
|    100022|     0|   Revolving loans|          F|           N|              Y|           0|        112500.0|  157
|    100023|     0|        Cash loans|          F|           N|              Y|           1|         90000.0|  544
+----------+------+------------------+-----------+------------+---------------+------------+----------------+-----
only showing top 20 rows
```

Out[27]: 307511

In [28]:

```
root
 |-- SK_ID_CURR: integer (nullable = true)
 |-- NAME_CONTRACT_TYPE: string (nullable = true)
 |-- CODE_GENDER: string (nullable = true)
 |-- FLAG_OWN_CAR: string (nullable = true)
 |-- FLAG_OWN_REALTY: string (nullable = true)
 |-- CNT_CHILDREN: integer (nullable = true)
 |-- AMT_INCOME_TOTAL: double (nullable = true)
 |-- AMT_CREDIT: double (nullable = true)
 |-- AMT_ANNUITY: double (nullable = true)
 |-- AMT_GOODS_PRICE: double (nullable = true)
 |-- NAME_TYPE_SUITE: string (nullable = true)
 |-- NAME_INCOME_TYPE: string (nullable = true)
 |-- NAME_EDUCATION_TYPE: string (nullable = true)
 |-- NAME_FAMILY_STATUS: string (nullable = true)
 |-- NAME_HOUSING_TYPE: string (nullable = true)
 |-- REGION_POPULATION_RELATIVE: double (nullable = true)
 |-- DAYS_BIRTH: integer (nullable = true)
 |-- DAYS_EMPLOYED: integer (nullable = true)
 |-- DAYS_REGISTRATION: double (nullable = true)
 |-- DAYS_ID_PUBLISH: integer (nullable = true)
 |-- OWN_CAR_AGE: double (nullable = true)
 |-- FLAG_MOBIL: integer (nullable = true)
 |-- FLAG_EMP_PHONE: integer (nullable = true)
 |-- FLAG_WORK_PHONE: integer (nullable = true)
 |-- FLAG_CONT_MOBILE: integer (nullable = true)
 |-- FLAG_PHONE: integer (nullable = true)
 |-- FLAG_EMAIL: integer (nullable = true)
 |-- OCCUPATION_TYPE: string (nullable = true)
 |-- CNT_FAM_MEMBERS: double (nullable = true)
 |-- REGION_RATING_CLIENT: integer (nullable = true)
 |-- REGION_RATING_CLIENT_W_CITY: integer (nullable = true)
 |-- WEEKDAY_APPR_PROCESS_START: string (nullable = true)
 |-- HOUR_APPR_PROCESS_START: integer (nullable = true)
 |-- REG_REGION_NOT_LIVE_REGION: integer (nullable = true)
 |-- REG_REGION_NOT_WORK_REGION: integer (nullable = true)
 |-- LIVE_REGION_NOT_WORK_REGION: integer (nullable = true)
 |-- REG_CITY_NOT_LIVE_CITY: integer (nullable = true)
 |-- REG_CITY_NOT_WORK_CITY: integer (nullable = true)
```

```
 |-- LIVE_CITY_NOT_WORK_CITY: integer (nullable = true)
 |-- ORGANIZATION_TYPE: string (nullable = true)
 |-- EXT_SOURCE_1: double (nullable = true)
 |-- EXT_SOURCE_2: double (nullable = true)
 |-- EXT_SOURCE_3: double (nullable = true)
 |-- APARTMENTS_AVG: double (nullable = true)
 |-- BASEMENTAREA_AVG: double (nullable = true)
 |-- YEARS_BEGINEXPLUATATION_AVG: double (nullable = true)
 |-- YEARS_BUILD_AVG: double (nullable = true)
 |-- COMMONAREA_AVG: double (nullable = true)
 |-- ELEVATORS_AVG: double (nullable = true)
 |-- ENTRANCES_AVG: double (nullable = true)
 |-- FLOORSMAX_AVG: double (nullable = true)
 |-- FLOORSMIN_AVG: double (nullable = true)
 |-- LANDAREA_AVG: double (nullable = true)
 |-- LIVINGAPARTMENTS_AVG: double (nullable = true)
 |-- LIVINGAREA_AVG: double (nullable = true)
 |-- NONLIVINGAPARTMENTS_AVG: double (nullable = true)
 |-- NONLIVINGAREA_AVG: double (nullable = true)
 |-- APARTMENTS_MODE: double (nullable = true)
 |-- BASEMENTAREA_MODE: double (nullable = true)
 |-- YEARS_BEGINEXPLUATATION_MODE: double (nullable = true)
 |-- YEARS_BUILD_MODE: double (nullable = true)
 |-- COMMONAREA_MODE: double (nullable = true)
 |-- ELEVATORS_MODE: double (nullable = true)
 |-- ENTRANCES_MODE: double (nullable = true)
 |-- FLOORSMAX_MODE: double (nullable = true)
 |-- FLOORSMIN_MODE: double (nullable = true)
 |-- LANDAREA_MODE: double (nullable = true)
 |-- LIVINGAPARTMENTS_MODE: double (nullable = true)
 |-- LIVINGAREA_MODE: double (nullable = true)
 |-- NONLIVINGAPARTMENTS_MODE: double (nullable = true)
 |-- NONLIVINGAREA_MODE: double (nullable = true)
 |-- APARTMENTS_MEDI: double (nullable = true)
 |-- BASEMENTAREA_MEDI: double (nullable = true)
 |-- YEARS_BEGINEXPLUATATION_MEDI: double (nullable = true)
 |-- YEARS_BUILD_MEDI: double (nullable = true)
 |-- COMMONAREA_MEDI: double (nullable = true)
 |-- ELEVATORS_MEDI: double (nullable = true)
```

```
|-- ENTRANCES_MEDI: double (nullable = true)
|-- FLOORSMAX_MEDI: double (nullable = true)
|-- FLOORSMIN_MEDI: double (nullable = true)
|-- LANDAREA_MEDI: double (nullable = true)
|-- LIVINGAPARTMENTS_MEDI: double (nullable = true)
|-- LIVINGAREA_MEDI: double (nullable = true)
|-- NONLIVINGAPARTMENTS_MEDI: double (nullable = true)
|-- NONLIVINGAREA_MEDI: double (nullable = true)
|-- FONDKAPREMONT_MODE: string (nullable = true)
|-- HOUSETYPE_MODE: string (nullable = true)
|-- TOTALAREA_MODE: double (nullable = true)
|-- WALLSMATERIAL_MODE: string (nullable = true)
|-- EMERGENCYSTATE_MODE: string (nullable = true)
|-- OBS_30_CNT_SOCIAL_CIRCLE: double (nullable = true)
|-- DEF_30_CNT_SOCIAL_CIRCLE: double (nullable = true)
|-- OBS_60_CNT_SOCIAL_CIRCLE: double (nullable = true)
|-- DEF_60_CNT_SOCIAL_CIRCLE: double (nullable = true)
|-- DAYS_LAST_PHONE_CHANGE: double (nullable = true)
|-- FLAG_DOCUMENT_2: integer (nullable = true)
|-- FLAG_DOCUMENT_3: integer (nullable = true)
|-- FLAG_DOCUMENT_4: integer (nullable = true)
|-- FLAG_DOCUMENT_5: integer (nullable = true)
|-- FLAG_DOCUMENT_6: integer (nullable = true)
|-- FLAG_DOCUMENT_7: integer (nullable = true)
|-- FLAG_DOCUMENT_8: integer (nullable = true)
|-- FLAG_DOCUMENT_9: integer (nullable = true)
|-- FLAG_DOCUMENT_10: integer (nullable = true)
|-- FLAG_DOCUMENT_11: integer (nullable = true)
|-- FLAG_DOCUMENT_12: integer (nullable = true)
|-- FLAG_DOCUMENT_13: integer (nullable = true)
|-- FLAG_DOCUMENT_14: integer (nullable = true)
|-- FLAG_DOCUMENT_15: integer (nullable = true)
|-- FLAG_DOCUMENT_16: integer (nullable = true)
|-- FLAG_DOCUMENT_17: integer (nullable = true)
|-- FLAG_DOCUMENT_18: integer (nullable = true)
|-- FLAG_DOCUMENT_19: integer (nullable = true)
|-- FLAG_DOCUMENT_20: integer (nullable = true)
|-- FLAG_DOCUMENT_21: integer (nullable = true)
|-- AMT_REQ_CREDIT_BUREAU_HOUR: double (nullable = true)
```

```
 |-- AMT_REQ_CREDIT_BUREAU_DAY: double (nullable = true)
 |-- AMT_REQ_CREDIT_BUREAU_WEEK: double (nullable = true)
 |-- AMT_REQ_CREDIT_BUREAU_MON: double (nullable = true)
 |-- AMT_REQ_CREDIT_BUREAU_QRT: double (nullable = true)
 |-- AMT_REQ_CREDIT_BUREAU_YEAR: double (nullable = true)

+----------+------------------+-----------+------------+---------------+------------+----------------+----------+-
|SK_ID_CURR|NAME_CONTRACT_TYPE|CODE_GENDER|FLAG_OWN_CAR|FLAG_OWN_REALTY|CNT_CHILDREN|AMT_INCOME_TOTAL|AMT_CREDIT|A
+----------+------------------+-----------+------------+---------------+------------+----------------+----------+-
|    100001|        Cash loans|          F|           N|              Y|           0|        135000.0|  568800.0|
|    100005|        Cash loans|          M|           N|              Y|           0|         99000.0|  222768.0|
|    100013|        Cash loans|          M|           Y|              Y|           0|        202500.0|  663264.0|
|    100028|        Cash loans|          F|           N|              Y|           2|        315000.0| 1575000.0|
|    100038|        Cash loans|          M|           Y|              N|           1|        180000.0|  625500.0|
|    100042|        Cash loans|          F|           Y|              Y|           0|        270000.0|  959688.0|
|    100057|        Cash loans|          M|           Y|              Y|           2|        180000.0|  499221.0|
|    100065|        Cash loans|          M|           N|              Y|           0|        166500.0|  180000.0|
|    100066|        Cash loans|          F|           N|              Y|           0|        315000.0|  364896.0|
|    100067|        Cash loans|          F|           Y|              Y|           1|        162000.0|   45000.0|
|    100074|        Cash loans|          F|           N|              Y|           0|         67500.0|  675000.0|
|    100090|        Cash loans|          F|           N|              Y|           0|        135000.0|  261621.0|
|    100091|        Cash loans|          F|           N|              Y|           0|        247500.0|  296280.0|
|    100092|        Cash loans|          F|           Y|              Y|           0|         90000.0|  360000.0|
|    100106|   Revolving loans|          M|           N|              Y|           0|        180000.0|  157500.0|
|    100107|        Cash loans|          M|           Y|              Y|           0|        180000.0|  296280.0|
|    100109|        Cash loans|          F|           Y|              Y|           0|        202500.0|  407520.0|
|    100117|        Cash loans|          M|           Y|              Y|           0|         90000.0|  499221.0|
|    100128|        Cash loans|          F|           Y|              Y|           1|        225000.0|  431280.0|
|    100141|        Cash loans|          F|           Y|              Y|           0|        175500.0|  478498.5|
+----------+------------------+-----------+------------+---------------+------------+----------------+----------+-
only showing top 20 rows
```

Out[28]: 48744

In [ ]:
```python
df_HomeCredit_columns_description.select("Table").distinct().show(truncate=False)
```

In [30]:
```python
df_HomeCredit_columns_description.filter
```

```
+---+----------------------------+----------------------------+---------------------------------------------------
|Id |Table                       |Row                         |Description
+---+----------------------------+----------------------------+---------------------------------------------------
|1  |application_{train|test}.csv|SK_ID_CURR                  |ID of loan in our sample
|2  |application_{train|test}.csv|TARGET                      |Target variable (1 - client with payment difficulti
|5  |application_{train|test}.csv|NAME_CONTRACT_TYPE          |Identification if loan is cash or revolving
|6  |application_{train|test}.csv|CODE_GENDER                 |Gender of the client
|7  |application_{train|test}.csv|FLAG_OWN_CAR                |Flag if the client owns a car
|8  |application_{train|test}.csv|FLAG_OWN_REALTY             |Flag if client owns a house or flat
|9  |application_{train|test}.csv|CNT_CHILDREN                |Number of children the client has
|10 |application_{train|test}.csv|AMT_INCOME_TOTAL            |Income of the client
|11 |application_{train|test}.csv|AMT_CREDIT                  |Credit amount of the loan
|12 |application_{train|test}.csv|AMT_ANNUITY                 |Loan annuity
|13 |application_{train|test}.csv|AMT_GOODS_PRICE             |For consumer loans it is the price of the goods for
|14 |application_{train|test}.csv|NAME_TYPE_SUITE             |Who was accompanying client when he was applying fo
|15 |application_{train|test}.csv|NAME_INCOME_TYPE            |Clients income type (businessman, working, maternit
|16 |application_{train|test}.csv|NAME_EDUCATION_TYPE         |Level of highest education the client achieved
|17 |application_{train|test}.csv|NAME_FAMILY_STATUS          |Family status of the client
|18 |application_{train|test}.csv|NAME_HOUSING_TYPE           |What is the housing situation of the client (rentir
|19 |application_{train|test}.csv|REGION_POPULATION_RELATIVE  |Normalized population of region where client lives
|20 |application_{train|test}.csv|DAYS_BIRTH                  |Client's age in days at the time of application
|21 |application_{train|test}.csv|DAYS_EMPLOYED               |How many days before the application the person sta
|22 |application_{train|test}.csv|DAYS_REGISTRATION           |How many days before the application did client cha
|23 |application_{train|test}.csv|DAYS_ID_PUBLISH             |How many days before the application did client cha
|24 |application_{train|test}.csv|OWN_CAR_AGE                 |Age of client's car
|25 |application_{train|test}.csv|FLAG_MOBIL                  |Did client provide mobile phone (1=YES, 0=NO)
|26 |application_{train|test}.csv|FLAG_EMP_PHONE              |Did client provide work phone (1=YES, 0=NO)
|27 |application_{train|test}.csv|FLAG_WORK_PHONE             |Did client provide home phone (1=YES, 0=NO)
|28 |application_{train|test}.csv|FLAG_CONT_MOBILE            |Was mobile phone reachable (1=YES, 0=NO)
|29 |application_{train|test}.csv|FLAG_PHONE                  |Did client provide home phone (1=YES, 0=NO)
|30 |application_{train|test}.csv|FLAG_EMAIL                  |Did client provide email (1=YES, 0=NO)
|31 |application_{train|test}.csv|OCCUPATION_TYPE             |What kind of occupation does the client have
|32 |application_{train|test}.csv|CNT_FAM_MEMBERS             |How many family members does client have
|33 |application_{train|test}.csv|REGION_RATING_CLIENT        |Our rating of the region where client lives (1,2,3)
|34 |application_{train|test}.csv|REGION_RATING_CLIENT_W_CITY |Our rating of the region where client lives with ta
|35 |application_{train|test}.csv|WEEKDAY_APPR_PROCESS_START  |On which day of the week did the client apply for t
|36 |application_{train|test}.csv|HOUR_APPR_PROCESS_START     |Approximately at what hour did the client apply for
|37 |application_{train|test}.csv|REG_REGION_NOT_LIVE_REGION  |Flag if client's permanent address does not match c
|38 |application_{train|test}.csv|REG_REGION_NOT_WORK_REGION  |Flag if client's permanent address does not match w
```

```
|39 |application_{train|test}.csv|LIVE_REGION_NOT_WORK_REGION |Flag if client's contact address does not match wor
|40 |application_{train|test}.csv|REG_CITY_NOT_LIVE_CITY      |Flag if client's permanent address does not match c
|41 |application_{train|test}.csv|REG_CITY_NOT_WORK_CITY      |Flag if client's permanent address does not match w
|42 |application_{train|test}.csv|LIVE_CITY_NOT_WORK_CITY     |Flag if client's contact address does not match wor
|43 |application_{train|test}.csv|ORGANIZATION_TYPE           |Type of organization where client works
|44 |application_{train|test}.csv|EXT_SOURCE_1                |Normalized score from external data source
|45 |application_{train|test}.csv|EXT_SOURCE_2                |Normalized score from external data source
|46 |application_{train|test}.csv|EXT_SOURCE_3                |Normalized score from external data source
|47 |application_{train|test}.csv|APARTMENTS_AVG              |Normalized information about building where the cli
|48 |application_{train|test}.csv|BASEMENTAREA_AVG            |Normalized information about building where the cli
|49 |application_{train|test}.csv|YEARS_BEGINEXPLUATATION_AVG |Normalized information about building where the cli
|50 |application_{train|test}.csv|YEARS_BUILD_AVG             |Normalized information about building where the cli
|51 |application_{train|test}.csv|COMMONAREA_AVG              |Normalized information about building where the cli
|52 |application_{train|test}.csv|ELEVATORS_AVG               |Normalized information about building where the cli
|53 |application_{train|test}.csv|ENTRANCES_AVG               |Normalized information about building where the cli
|54 |application_{train|test}.csv|FLOORSMAX_AVG               |Normalized information about building where the cli
|55 |application_{train|test}.csv|FLOORSMIN_AVG               |Normalized information about building where the cli
|56 |application_{train|test}.csv|LANDAREA_AVG                |Normalized information about building where the cli
|57 |application_{train|test}.csv|LIVINGAPARTMENTS_AVG        |Normalized information about building where the cli
|58 |application_{train|test}.csv|LIVINGAREA_AVG              |Normalized information about building where the cli
|59 |application_{train|test}.csv|NONLIVINGAPARTMENTS_AVG     |Normalized information about building where the cli
|60 |application_{train|test}.csv|NONLIVINGAREA_AVG           |Normalized information about building where the cli
|61 |application_{train|test}.csv|APARTMENTS_MODE             |Normalized information about building where the cli
|62 |application_{train|test}.csv|BASEMENTAREA_MODE           |Normalized information about building where the cli
|63 |application_{train|test}.csv|YEARS_BEGINEXPLUATATION_MODE|Normalized information about building where the cli
|64 |application_{train|test}.csv|YEARS_BUILD_MODE            |Normalized information about building where the cli
|65 |application_{train|test}.csv|COMMONAREA_MODE             |Normalized information about building where the cli
|66 |application_{train|test}.csv|ELEVATORS_MODE              |Normalized information about building where the cli
|67 |application_{train|test}.csv|ENTRANCES_MODE              |Normalized information about building where the cli
|68 |application_{train|test}.csv|FLOORSMAX_MODE              |Normalized information about building where the cli
|69 |application_{train|test}.csv|FLOORSMIN_MODE              |Normalized information about building where the cli
|70 |application_{train|test}.csv|LANDAREA_MODE               |Normalized information about building where the cli
|71 |application_{train|test}.csv|LIVINGAPARTMENTS_MODE       |Normalized information about building where the cli
|72 |application_{train|test}.csv|LIVINGAREA_MODE             |Normalized information about building where the cli
|73 |application_{train|test}.csv|NONLIVINGAPARTMENTS_MODE    |Normalized information about building where the cli
|74 |application_{train|test}.csv|NONLIVINGAREA_MODE          |Normalized information about building where the cli
|75 |application_{train|test}.csv|APARTMENTS_MEDI             |Normalized information about building where the cli
|76 |application_{train|test}.csv|BASEMENTAREA_MEDI           |Normalized information about building where the cli
|77 |application_{train|test}.csv|YEARS_BEGINEXPLUATATION_MEDI|Normalized information about building where the cli
```

78	application_{train|test}.csv	YEARS_BUILD_MEDI	Normalized information about building where the cli
79	application_{train|test}.csv	COMMONAREA_MEDI	Normalized information about building where the cli
80	application_{train|test}.csv	ELEVATORS_MEDI	Normalized information about building where the cli
81	application_{train|test}.csv	ENTRANCES_MEDI	Normalized information about building where the cli
82	application_{train|test}.csv	FLOORSMAX_MEDI	Normalized information about building where the cli
83	application_{train|test}.csv	FLOORSMIN_MEDI	Normalized information about building where the cli
84	application_{train|test}.csv	LANDAREA_MEDI	Normalized information about building where the cli
85	application_{train|test}.csv	LIVINGAPARTMENTS_MEDI	Normalized information about building where the cli
86	application_{train|test}.csv	LIVINGAREA_MEDI	Normalized information about building where the cli
87	application_{train|test}.csv	NONLIVINGAPARTMENTS_MEDI	Normalized information about building where the cli
88	application_{train|test}.csv	NONLIVINGAREA_MEDI	Normalized information about building where the cli
89	application_{train|test}.csv	FONDKAPREMONT_MODE	Normalized information about building where the cli
90	application_{train|test}.csv	HOUSETYPE_MODE	Normalized information about building where the cli
91	application_{train|test}.csv	TOTALAREA_MODE	Normalized information about building where the cli
92	application_{train|test}.csv	WALLSMATERIAL_MODE	Normalized information about building where the cli
93	application_{train|test}.csv	EMERGENCYSTATE_MODE	Normalized information about building where the cli
94	application_{train|test}.csv	OBS_30_CNT_SOCIAL_CIRCLE	How many observation of client's social surrounding
95	application_{train|test}.csv	DEF_30_CNT_SOCIAL_CIRCLE	How many observation of client's social surrounding
96	application_{train|test}.csv	OBS_60_CNT_SOCIAL_CIRCLE	How many observation of client's social surrounding
97	application_{train|test}.csv	DEF_60_CNT_SOCIAL_CIRCLE	How many observation of client's social surrounding
98	application_{train|test}.csv	DAYS_LAST_PHONE_CHANGE	How many days before application did client change
99	application_{train|test}.csv	FLAG_DOCUMENT_2	Did client provide document 2
100	application_{train|test}.csv	FLAG_DOCUMENT_3	Did client provide document 3
101	application_{train|test}.csv	FLAG_DOCUMENT_4	Did client provide document 4
102	application_{train|test}.csv	FLAG_DOCUMENT_5	Did client provide document 5
103	application_{train|test}.csv	FLAG_DOCUMENT_6	Did client provide document 6
104	application_{train|test}.csv	FLAG_DOCUMENT_7	Did client provide document 7
105	application_{train|test}.csv	FLAG_DOCUMENT_8	Did client provide document 8
106	application_{train|test}.csv	FLAG_DOCUMENT_9	Did client provide document 9
107	application_{train|test}.csv	FLAG_DOCUMENT_10	Did client provide document 10
108	application_{train|test}.csv	FLAG_DOCUMENT_11	Did client provide document 11
109	application_{train|test}.csv	FLAG_DOCUMENT_12	Did client provide document 12
110	application_{train|test}.csv	FLAG_DOCUMENT_13	Did client provide document 13
111	application_{train|test}.csv	FLAG_DOCUMENT_14	Did client provide document 14
112	application_{train|test}.csv	FLAG_DOCUMENT_15	Did client provide document 15
113	application_{train|test}.csv	FLAG_DOCUMENT_16	Did client provide document 16
114	application_{train|test}.csv	FLAG_DOCUMENT_17	Did client provide document 17
115	application_{train|test}.csv	FLAG_DOCUMENT_18	Did client provide document 18
116	application_{train|test}.csv	FLAG_DOCUMENT_19	Did client provide document 19

```
|117|application_{train|test}.csv|FLAG_DOCUMENT_20            |Did client provide document 20
|118|application_{train|test}.csv|FLAG_DOCUMENT_21            |Did client provide document 21
|119|application_{train|test}.csv|AMT_REQ_CREDIT_BUREAU_HOUR  |Number of enquiries to Credit Bureau about the clie
|120|application_{train|test}.csv|AMT_REQ_CREDIT_BUREAU_DAY   |Number of enquiries to Credit Bureau about the clie
|121|application_{train|test}.csv|AMT_REQ_CREDIT_BUREAU_WEEK  |Number of enquiries to Credit Bureau about the clie
|122|application_{train|test}.csv|AMT_REQ_CREDIT_BUREAU_MON   |Number of enquiries to Credit Bureau about the clie
|123|application_{train|test}.csv|AMT_REQ_CREDIT_BUREAU_QRT   |Number of enquiries to Credit Bureau about the clie
|124|application_{train|test}.csv|AMT_REQ_CREDIT_BUREAU_YEAR  |Number of enquiries to Credit Bureau about the clie
+---+----------------------------+----------------------------+--------------------------------------------------
```

Explore and evaluate data

Let us get some data insight, with some **exploratory data analysis** based on descriptive statistics and visualizations.

The dataframes of concern are the ones from the {train/test} data files.

Note on data quality profiling and exploratory data analysis

There is an interesting tool to profile dataframes but unfortunately as it stands the Python version we are using now is higher than the ones accepted by the tool.

> See https://docs.profiling.ydata.ai/latest/

Datatypes

Is there a need to make adjustments/adding new fields to the data types specified in the dataframes? See the corresponding schema.

Nulls, NaN and uniques

Identify number of nulls or NaN in columns as well uniques. This is helpful to further investigating features of concern.

In [31]:
```
df_application_train.
```

Out[31]:
```
['SK_ID_CURR',
 'TARGET',
 'NAME_CONTRACT_TYPE',
 'CODE_GENDER',
 'FLAG_OWN_CAR',
 'FLAG_OWN_REALTY',
 'CNT_CHILDREN',
 'AMT_INCOME_TOTAL',
 'AMT_CREDIT',
 'AMT_ANNUITY',
 'AMT_GOODS_PRICE',
 'NAME_TYPE_SUITE',
 'NAME_INCOME_TYPE',
 'NAME_EDUCATION_TYPE',
 'NAME_FAMILY_STATUS',
 'NAME_HOUSING_TYPE',
 'REGION_POPULATION_RELATIVE',
 'DAYS_BIRTH',
 'DAYS_EMPLOYED',
 'DAYS_REGISTRATION',
 'DAYS_ID_PUBLISH',
 'OWN_CAR_AGE',
 'FLAG_MOBIL',
 'FLAG_EMP_PHONE',
 'FLAG_WORK_PHONE',
 'FLAG_CONT_MOBILE',
 'FLAG_PHONE',
 'FLAG_EMAIL',
 'OCCUPATION_TYPE',
 'CNT_FAM_MEMBERS',
 'REGION_RATING_CLIENT',
 'REGION_RATING_CLIENT_W_CITY',
 'WEEKDAY_APPR_PROCESS_START',
 'HOUR_APPR_PROCESS_START',
 'REG_REGION_NOT_LIVE_REGION',
 'REG_REGION_NOT_WORK_REGION',
 'LIVE_REGION_NOT_WORK_REGION',
```

```
'REG_CITY_NOT_LIVE_CITY',
'REG_CITY_NOT_WORK_CITY',
'LIVE_CITY_NOT_WORK_CITY',
'ORGANIZATION_TYPE',
'EXT_SOURCE_1',
'EXT_SOURCE_2',
'EXT_SOURCE_3',
'APARTMENTS_AVG',
'BASEMENTAREA_AVG',
'YEARS_BEGINEXPLUATATION_AVG',
'YEARS_BUILD_AVG',
'COMMONAREA_AVG',
'ELEVATORS_AVG',
'ENTRANCES_AVG',
'FLOORSMAX_AVG',
'FLOORSMIN_AVG',
'LANDAREA_AVG',
'LIVINGAPARTMENTS_AVG',
'LIVINGAREA_AVG',
'NONLIVINGAPARTMENTS_AVG',
'NONLIVINGAREA_AVG',
'APARTMENTS_MODE',
'BASEMENTAREA_MODE',
'YEARS_BEGINEXPLUATATION_MODE',
'YEARS_BUILD_MODE',
'COMMONAREA_MODE',
'ELEVATORS_MODE',
'ENTRANCES_MODE',
'FLOORSMAX_MODE',
'FLOORSMIN_MODE',
'LANDAREA_MODE',
'LIVINGAPARTMENTS_MODE',
'LIVINGAREA_MODE',
'NONLIVINGAPARTMENTS_MODE',
'NONLIVINGAREA_MODE',
'APARTMENTS_MEDI',
'BASEMENTAREA_MEDI',
'YEARS_BEGINEXPLUATATION_MEDI',
'YEARS_BUILD_MEDI',
```

```
'COMMONAREA_MEDI',
'ELEVATORS_MEDI',
'ENTRANCES_MEDI',
'FLOORSMAX_MEDI',
'FLOORSMIN_MEDI',
'LANDAREA_MEDI',
'LIVINGAPARTMENTS_MEDI',
'LIVINGAREA_MEDI',
'NONLIVINGAPARTMENTS_MEDI',
'NONLIVINGAREA_MEDI',
'FONDKAPREMONT_MODE',
'HOUSETYPE_MODE',
'TOTALAREA_MODE',
'WALLSMATERIAL_MODE',
'EMERGENCYSTATE_MODE',
'OBS_30_CNT_SOCIAL_CIRCLE',
'DEF_30_CNT_SOCIAL_CIRCLE',
'OBS_60_CNT_SOCIAL_CIRCLE',
'DEF_60_CNT_SOCIAL_CIRCLE',
'DAYS_LAST_PHONE_CHANGE',
'FLAG_DOCUMENT_2',
'FLAG_DOCUMENT_3',
'FLAG_DOCUMENT_4',
'FLAG_DOCUMENT_5',
'FLAG_DOCUMENT_6',
'FLAG_DOCUMENT_7',
'FLAG_DOCUMENT_8',
'FLAG_DOCUMENT_9',
'FLAG_DOCUMENT_10',
'FLAG_DOCUMENT_11',
'FLAG_DOCUMENT_12',
'FLAG_DOCUMENT_13',
'FLAG_DOCUMENT_14',
'FLAG_DOCUMENT_15',
'FLAG_DOCUMENT_16',
'FLAG_DOCUMENT_17',
'FLAG_DOCUMENT_18',
'FLAG_DOCUMENT_19',
'FLAG_DOCUMENT_20',
```

```
'FLAG_DOCUMENT_21',
'AMT_REQ_CREDIT_BUREAU_HOUR',
'AMT_REQ_CREDIT_BUREAU_DAY',
'AMT_REQ_CREDIT_BUREAU_WEEK',
'AMT_REQ_CREDIT_BUREAU_MON',
'AMT_REQ_CREDIT_BUREAU_QRT',
'AMT_REQ_CREDIT_BUREAU_YEAR']
```

In [32]:

```python
cols_to_check = ['SK_ID_CURR',
                 'TARGET',
 'NAME_CONTRACT_TYPE',
 'CODE_GENDER',
 'FLAG_OWN_CAR',
 'FLAG_OWN_REALTY',
 'CNT_CHILDREN',
 'AMT_INCOME_TOTAL',
 'AMT_CREDIT',
 'AMT_ANNUITY',
 'AMT_GOODS_PRICE',
 'NAME_TYPE_SUITE',
 'NAME_INCOME_TYPE',
 'NAME_EDUCATION_TYPE',
 'NAME_FAMILY_STATUS',
 'NAME_HOUSING_TYPE',
 'REGION_POPULATION_RELATIVE',
 'DAYS_BIRTH',
 'DAYS_EMPLOYED',
 'DAYS_REGISTRATION',
 'DAYS_ID_PUBLISH',
 'OWN_CAR_AGE',
 'FLAG_MOBIL',
 'FLAG_EMP_PHONE',
 'FLAG_WORK_PHONE',
 'FLAG_CONT_MOBILE',
 'FLAG_PHONE',
 'FLAG_EMAIL',
 'OCCUPATION_TYPE',
 'CNT_FAM_MEMBERS',
 'REGION_RATING_CLIENT',
```

```
'REGION_RATING_CLIENT_W_CITY',
'WEEKDAY_APPR_PROCESS_START',
'HOUR_APPR_PROCESS_START',
'REG_REGION_NOT_LIVE_REGION',
'REG_REGION_NOT_WORK_REGION',
'LIVE_REGION_NOT_WORK_REGION',
'REG_CITY_NOT_LIVE_CITY',
'REG_CITY_NOT_WORK_CITY',
'LIVE_CITY_NOT_WORK_CITY',
'ORGANIZATION_TYPE',
'EXT_SOURCE_1',
'EXT_SOURCE_2',
'EXT_SOURCE_3',
]
```

In [33]:
```
df_train_nulls_uniques = compute_nulls_and_uniques(df_application_train, cols_to_check)
```

In [34]:
```
df_train_nulls_uniques.
```

```
+---------------------------+-----------+---------------------+----------+---------------+-------------+----------
|feature                    |count_nulls|percentage_nulls     |count_nans|percentage_nans|count_uniques|percentage
+---------------------------+-----------+---------------------+----------+---------------+-------------+----------
|SK_ID_CURR                 |0          |0.0                  |0         |0.0            |307511       |1.0
|TARGET                     |0          |0.0                  |0         |0.0            |2            |6.50383238
|NAME_CONTRACT_TYPE         |0          |0.0                  |0         |0.0            |2            |6.50383238
|CODE_GENDER                |0          |0.0                  |0         |0.0            |3            |9.75574857
|FLAG_OWN_CAR               |0          |0.0                  |0         |0.0            |2            |6.50383238
|FLAG_OWN_REALTY            |0          |0.0                  |0         |0.0            |2            |6.50383238
|CNT_CHILDREN               |0          |0.0                  |0         |0.0            |15           |4.87787428
|AMT_INCOME_TOTAL           |0          |0.0                  |0         |0.0            |2548         |0.00828588
|AMT_CREDIT                 |0          |0.0                  |0         |0.0            |5603         |0.01822048
|AMT_ANNUITY                |12         |3.9022994299390916E-5|0         |0.0            |13673        |0.04446345
|AMT_GOODS_PRICE            |278        |9.040327012692228E-4 |0         |0.0            |1003         |0.00326167
|NAME_TYPE_SUITE            |1292       |0.004201475719567756 |0         |0.0            |8            |2.60153295
|NAME_INCOME_TYPE           |0          |0.0                  |0         |0.0            |8            |2.60153295
|NAME_EDUCATION_TYPE        |0          |0.0                  |0         |0.0            |5            |1.62595809
|NAME_FAMILY_STATUS         |0          |0.0                  |0         |0.0            |6            |1.95114971
|NAME_HOUSING_TYPE          |0          |0.0                  |0         |0.0            |6            |1.95114971
|REGION_POPULATION_RELATIVE |0          |0.0                  |0         |0.0            |81           |2.63405211
```

```
|DAYS_BIRTH                 |0          |0.0                   |0         |0.0            |17460        |0.05677845
|DAYS_EMPLOYED              |0          |0.0                   |0         |0.0            |12574        |0.04088959
|DAYS_REGISTRATION          |0          |0.0                   |0         |0.0            |15688        |0.05101606
|DAYS_ID_PUBLISH            |0          |0.0                   |0         |0.0            |6168         |0.02005781
|OWN_CAR_AGE                |202929     |0.6599081008484249    |0         |0.0            |63           |2.04870720
|FLAG_MOBIL                 |0          |0.0                   |0         |0.0            |2            |6.50383238
|FLAG_EMP_PHONE             |0          |0.0                   |0         |0.0            |2            |6.50383238
|FLAG_WORK_PHONE            |0          |0.0                   |0         |0.0            |2            |6.50383238
|FLAG_CONT_MOBILE           |0          |0.0                   |0         |0.0            |2            |6.50383238
|FLAG_PHONE                 |0          |0.0                   |0         |0.0            |2            |6.50383238
|FLAG_EMAIL                 |0          |0.0                   |0         |0.0            |2            |6.50383238
|OCCUPATION_TYPE            |96391      |0.31345545362604915   |0         |0.0            |19           |6.17864076
|CNT_FAM_MEMBERS            |2          |6.503832383231819E-6  |0         |0.0            |18           |5.85344914
|REGION_RATING_CLIENT       |0          |0.0                   |0         |0.0            |3            |9.75574857
|REGION_RATING_CLIENT_W_CITY|0          |0.0                   |0         |0.0            |3            |9.75574857
|WEEKDAY_APPR_PROCESS_START |0          |0.0                   |0         |0.0            |7            |2.27634133
|HOUR_APPR_PROCESS_START    |0          |0.0                   |0         |0.0            |24           |7.80459885
|REG_REGION_NOT_LIVE_REGION |0          |0.0                   |0         |0.0            |2            |6.50383238
|REG_REGION_NOT_WORK_REGION |0          |0.0                   |0         |0.0            |2            |6.50383238
|LIVE_REGION_NOT_WORK_REGION|0          |0.0                   |0         |0.0            |2            |6.50383238
|REG_CITY_NOT_LIVE_CITY     |0          |0.0                   |0         |0.0            |2            |6.50383238
|REG_CITY_NOT_WORK_CITY     |0          |0.0                   |0         |0.0            |2            |6.50383238
|LIVE_CITY_NOT_WORK_CITY    |0          |0.0                   |0         |0.0            |2            |6.50383238
|ORGANIZATION_TYPE          |0          |0.0                   |0         |0.0            |58           |1.88611139
|EXT_SOURCE_1               |173378     |0.5638107254699832    |0         |0.0            |114585       |0.37262081
|EXT_SOURCE_2               |660        |0.0021462646864665006 |0         |0.0            |119832       |0.38968362
|EXT_SOURCE_3               |60965      |0.19825307062186392   |0         |0.0            |815          |0.00265031
+---------------------------+-----------+----------------------+----------+---------------+-------------+----------
```

In [ ]:

```python
# Taking the decision to forget some columns.
#
# Let us simply forget those columns with nulls,
# so leaving by now considerations like imputing data or dropping records

cols_to_forget = ['AMT_ANNUITY', 'AMT_GOODS_PRICE', 'NAME_TYPE_SUITE', 'OWN_CAR_AGE',
                  'OCCUPATION_TYPE', 'CNT_FAM_MEMBERS', 'EXT_SOURCE_1','EXT_SOURCE_2','EXT_SOURCE_3']
cols_interest = [c for c in cols_to_check if c not in cols_to_forget]
```

```
cols_interest
```

Summary to figure out outliers

Summary of values for columns of interest. Use of describe() or summary()

In [36]:
```
df_application_train.
```

```
[Stage 556:=======>                                                 (1 + 7) / 8]
+-------+------------------+-------------------+------------------+-----------+------------+--------------+------
|summary|        SK_ID_CURR|             TARGET|NAME_CONTRACT_TYPE|CODE_GENDER|FLAG_OWN_CAR|FLAG_OWN_REALTY|
+-------+------------------+-------------------+------------------+-----------+------------+--------------+------
|  count|            307511|             307511|            307511|     307511|      307511|        307511|
|   mean|278180.51857657125|0.08072881945686496|              NULL|       NULL|        NULL|          NULL|0.417(
| stddev|102790.17534842453| 0.2724186456483939|              NULL|       NULL|        NULL|          NULL|0.7221
|    min|            100002|                  0|        Cash loans|          F|           N|             N|
|    max|            456255|                  1|   Revolving loans|        XNA|           Y|             Y|
+-------+------------------+-------------------+------------------+-----------+------------+--------------+------
```

Feature Engineering

Now we have to prepare data in a way that it can be properly used by ML algorithms, which includes selection and extraction of features, as well as dealing with poor data quality if that is the case.

Saving clean data

Saving data for further use if needed.

In [ ]:
```
cols_interest_test = [c for c in cols_interest if c != 'TARGET']
```

```
cols_interest_test
```

In [38]:
```
# Drop columns that will not be used anymore
# ... or select just the ones of interest
df_application_train = df_application_train.select(cols_interest)
df_application_test = df_application_test.select(cols_interest_test)
```

In [39]:
```
# We should also have a smaller dataset to set up the model,
# just for the purpose of working locally

# and we should understand the implications of sampling

seed = 5
with_replacement = False
fraction = 0.3          # reduce to 30%
df_application_train_small = df_application_train.sample(withReplacement=with_replacement,
                                                         fraction=fraction, seed=seed)
```

In [40]:
```
[df_application_train.count(), df_application_train_small.count(), df_application_test.count()]
```

Out[40]:
```
[307511, 92075, 48744]
```

In [41]:
```
# Save the data frames to files in parquet for future use in case of need

df_application_train.write.mode("overwrite").parquet("application_for_model")
df_application_train_small.
df_application_test.

# and later on, we can use spark.read.parquet() to load files
```

```
24/03/13 23:27:01 WARN MemoryManager: Total allocation exceeds 95.00% (1,020,054,720 bytes) of heap memory
Scaling row group sizes to 95.00% for 8 writers
24/03/13 23:27:02 WARN MemoryManager: Total allocation exceeds 95.00% (1,020,054,720 bytes) of heap memory
Scaling row group sizes to 95.00% for 8 writers
```

Check in the running directory if that was accomplished, incluing the files in parquet

In [ ]:
```
ls -la application_for_model
```

In [ ]:
```
ls -la application_for_model_small
```

In [ ]:
```
ls -la application_for_test
```

Data for the model hereafter

In [45]:
```
df_for_model = df_application_train
# df_for_model = df_application_train_small
```

In [46]:
```
# Delete memory consuming variables that are no longer needed

# del
```

Final overview regarding data to be used in the model

After establishing the clean data to be used, let us get an overview about what we have achieved, with some statistics and visualizations. Now we may look at specific columns in more detail.

Descriptive statistics

See also results above regarding the data that has been checked (nulls, NaN, uniqueness and describe/summary)

In [47]:
```
cols_numeric = numeric_columns(df_for_model)
```

In [48]:
```
# Just to check that there are no nulls or NaN

for cl in df_for_model.columns:
    knulls_nans = df_for_model.select(cl).filter(F.col(cl).isNull() | F.isnan(cl)).count()
    if knulls_nans > 0:
        print(f"Warning: Feature {cl} has got {knulls_nans} rows with nulls or NaN in the data for building the r
```

In [49]:
```
# Features with not many distinct values (see above)
```

```python
cls_1 = [
 'NAME_CONTRACT_TYPE',
 'CODE_GENDER',
 'FLAG_OWN_CAR',
 'FLAG_OWN_REALTY',
 'CNT_CHILDREN',
 'NAME_INCOME_TYPE',
 'NAME_EDUCATION_TYPE',
 'NAME_FAMILY_STATUS',
 'NAME_HOUSING_TYPE',
 'FLAG_MOBIL',
 'FLAG_EMP_PHONE',
 'FLAG_WORK_PHONE',
 'FLAG_CONT_MOBILE',
 'FLAG_PHONE',
 'FLAG_EMAIL',
 'REGION_RATING_CLIENT',
 'REGION_RATING_CLIENT_W_CITY',
 'WEEKDAY_APPR_PROCESS_START',
 'REG_REGION_NOT_LIVE_REGION',
 'REG_REGION_NOT_WORK_REGION',
 'LIVE_REGION_NOT_WORK_REGION',
 'REG_CITY_NOT_LIVE_CITY',
 'REG_CITY_NOT_WORK_CITY',
 'LIVE_CITY_NOT_WORK_CITY'
]
```

In [50]:

```python
print('\nShowing a few uniques:')
for cl in cls_1:
    df_for_model.
```

```
Showing a few uniques:
+------------------+
|NAME_CONTRACT_TYPE|
+------------------+
|Cash loans        |
|Revolving loans   |
+------------------+
```

```
+-----------+
|CODE_GENDER|
+-----------+
|F          |
|M          |
|XNA        |
+-----------+

+------------+
|FLAG_OWN_CAR|
+------------+
|N           |
|Y           |
+------------+

+---------------+
|FLAG_OWN_REALTY|
+---------------+
|N              |
|Y              |
+---------------+

+------------+
|CNT_CHILDREN|
+------------+
|0           |
|1           |
|2           |
|3           |
|4           |
|5           |
|6           |
|7           |
|8           |
|9           |
|10          |
|11          |
|12          |
```

```
|14          |
|19          |
+------------+

+--------------------+
|NAME_INCOME_TYPE    |
+--------------------+
|Businessman         |
|Commercial associate|
|Maternity leave     |
|Pensioner           |
|State servant       |
|Student             |
|Unemployed          |
|Working             |
+--------------------+

+-----------------------------+
|NAME_EDUCATION_TYPE          |
+-----------------------------+
|Academic degree              |
|Higher education             |
|Incomplete higher            |
|Lower secondary              |
|Secondary / secondary special|
+-----------------------------+

+--------------------+
|NAME_FAMILY_STATUS  |
+--------------------+
|Civil marriage      |
|Married             |
|Separated           |
|Single / not married|
|Unknown             |
|Widow               |
+--------------------+

+-------------------+
```

```
|NAME_HOUSING_TYPE  |
+-------------------+
|Co-op apartment    |
|House / apartment  |
|Municipal apartment|
|Office apartment   |
|Rented apartment   |
|With parents       |
+-------------------+

+----------+
|FLAG_MOBIL|
+----------+
|0         |
|1         |
+----------+

+--------------+
|FLAG_EMP_PHONE|
+--------------+
|0             |
|1             |
+--------------+

+---------------+
|FLAG_WORK_PHONE|
+---------------+
|0              |
|1              |
+---------------+

+----------------+
|FLAG_CONT_MOBILE|
+----------------+
|0               |
|1               |
+----------------+

+----------+
```

```
|FLAG_PHONE|
+----------+
|0         |
|1         |
+----------+

+----------+
|FLAG_EMAIL|
+----------+
|0         |
|1         |
+----------+

+--------------------+
|REGION_RATING_CLIENT|
+--------------------+
|1                   |
|2                   |
|3                   |
+--------------------+

+---------------------------+
|REGION_RATING_CLIENT_W_CITY|
+---------------------------+
|1                          |
|2                          |
|3                          |
+---------------------------+

+--------------------------+
|WEEKDAY_APPR_PROCESS_START|
+--------------------------+
|FRIDAY                    |
|MONDAY                    |
|SATURDAY                  |
|SUNDAY                    |
|THURSDAY                  |
|TUESDAY                   |
|WEDNESDAY                 |
```

```
+--------------------------+

+--------------------------+
|REG_REGION_NOT_LIVE_REGION|
+--------------------------+
|0                         |
|1                         |
+--------------------------+

+--------------------------+
|REG_REGION_NOT_WORK_REGION|
+--------------------------+
|0                         |
|1                         |
+--------------------------+

+---------------------------+
|LIVE_REGION_NOT_WORK_REGION|
+---------------------------+
|0                          |
|1                          |
+---------------------------+

+----------------------+
|REG_CITY_NOT_LIVE_CITY|
+----------------------+
|0                     |
|1                     |
+----------------------+

+----------------------+
|REG_CITY_NOT_WORK_CITY|
+----------------------+
|0                     |
|1                     |
+----------------------+

+-----------------------+
|LIVE_CITY_NOT_WORK_CITY|
```

```
+-----------------------+
|0                      |
|1                      |
+-----------------------+
```

Correlations

In [51]:
```
# Checking correlations among some columns
#
# Correlation needs vectors so we convert to vector column first
# See VectorAssembler in the Spark's documentation

# The columns to compute correlations - numeric types but no nulls
cols_corr = cols_numeric

# Assemble columns
vector_col = "corr_features"
assembler = VectorAssembler(inputCols=cols_corr, outputCol=vector_col, handleInvalid = "skip")  # "keep"
df_vector = assembler.transform(df_for_model).select(vector_col)

# Get correlation matrix - it can be Pearson’s (default) or Spearman’s correlation

# corr = Correlation.corr(df_vector, vector_col).head()
# print("Pearson correlation matrix:\n" + str(corr[0]))

# corr = Correlation.corr(df_vector, vector_col, "spearman").head()
# print("Spearman correlation matrix:\n" + str(corr[0]))

corr_matrix = Correlation.corr(df_vector, vector_col).collect()[0][0].toArray().tolist()
# corr_matrix
```

```
24/03/13 23:27:22 WARN InstanceBuilder: Failed to load implementation from:dev.ludovic.netlib.blas.JNIBLAS
```

In order to visualize, first we convert to Pandas dataframe and then plot it

In [ ]:
```
# Plot computed correlation
```

```
df_plot = pd.DataFrame(data = corr_matrix, index=cols_corr, columns=cols_corr)
plotCorrelationMatrix(df_plot, annot=True)
plt.title('Correlations among numerical features')
plt.show()
```

Overall picture

Besides the correlation matrix above, we are going to view and/or visualize data to learn more about the data.

Feel free to add and/or remove visualizations.

In [53]:
```
# Counting of the dependent variable TARGET

df_for_model.
```

```
+------+------+
|TARGET| count|
+------+------+
|     1| 24825|
|     0|282686|
+------+------+
```

The counting above shows a clear imbalance in the distribution of the dependent variable TARGET: The critical class 1 is significantly less frequently than class 0.

Recalling the description of the target variable:

- 1 - client with payment difficulties: he/she had late payment more than X days on at least one of the first Y installments of the loan in our sample
- 0 - all other cases

In [ ]:
```
#  Type of contract

df_plot = df_for_model.groupBy('NAME_CONTRACT_TYPE').count().toPandas()
plotBar(df_plot, 'NAME_CONTRACT_TYPE', 'count')
```

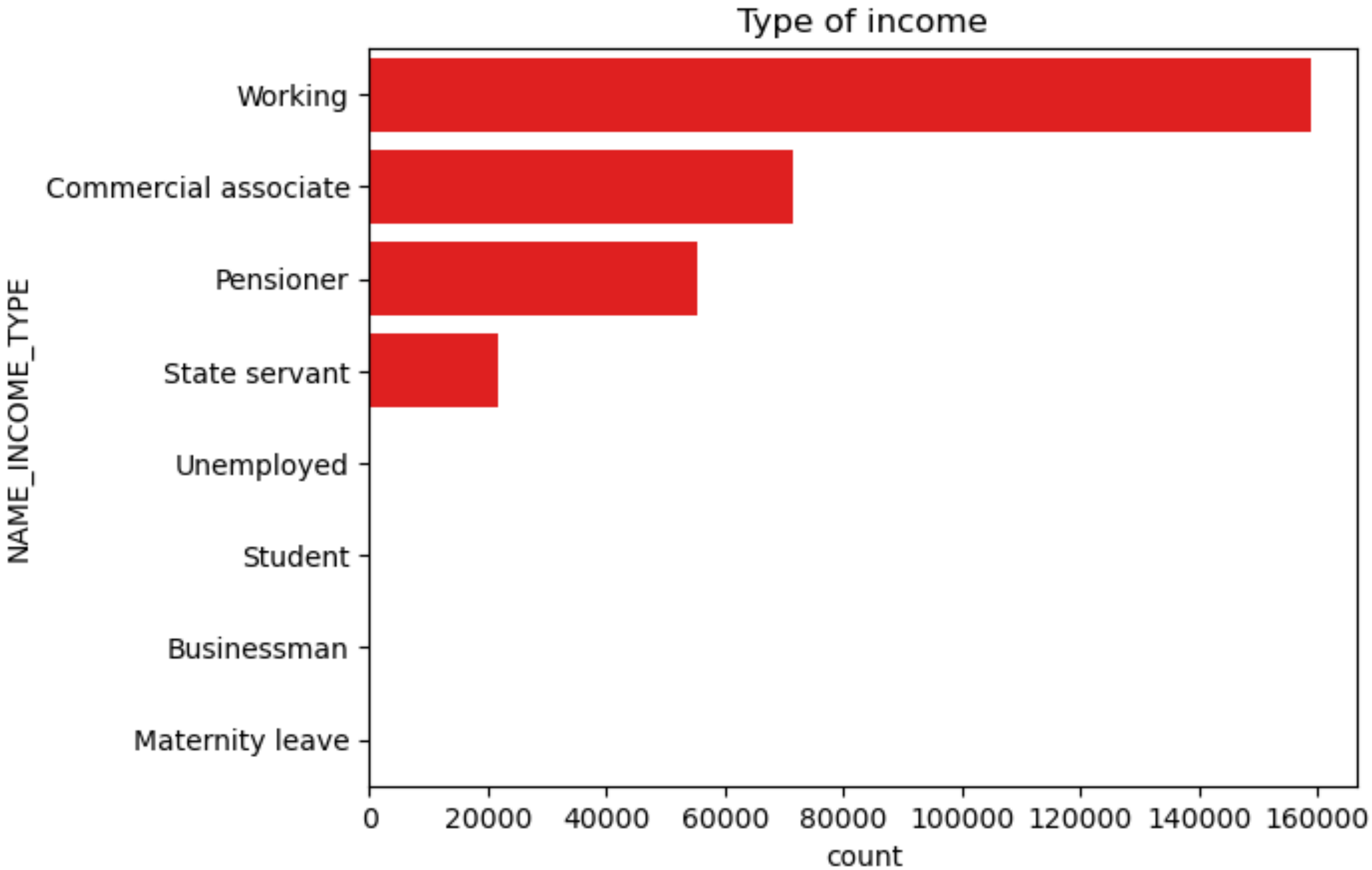
```
plt.title('Type of contract')
plt.show()
```

In [55]:
```
df_plot.head()
```

Out[55]:

	NAME_CONTRACT_TYPE	count
0	Cash loans	278232
1	Revolving loans	29279

In [56]:
```
# Type of income

df_plot = (


           )
plotHorizBar(df_plot, 'count', 'NAME_INCOME_TYPE', 'red')
plt.title('Type of income')
plt.show()
```



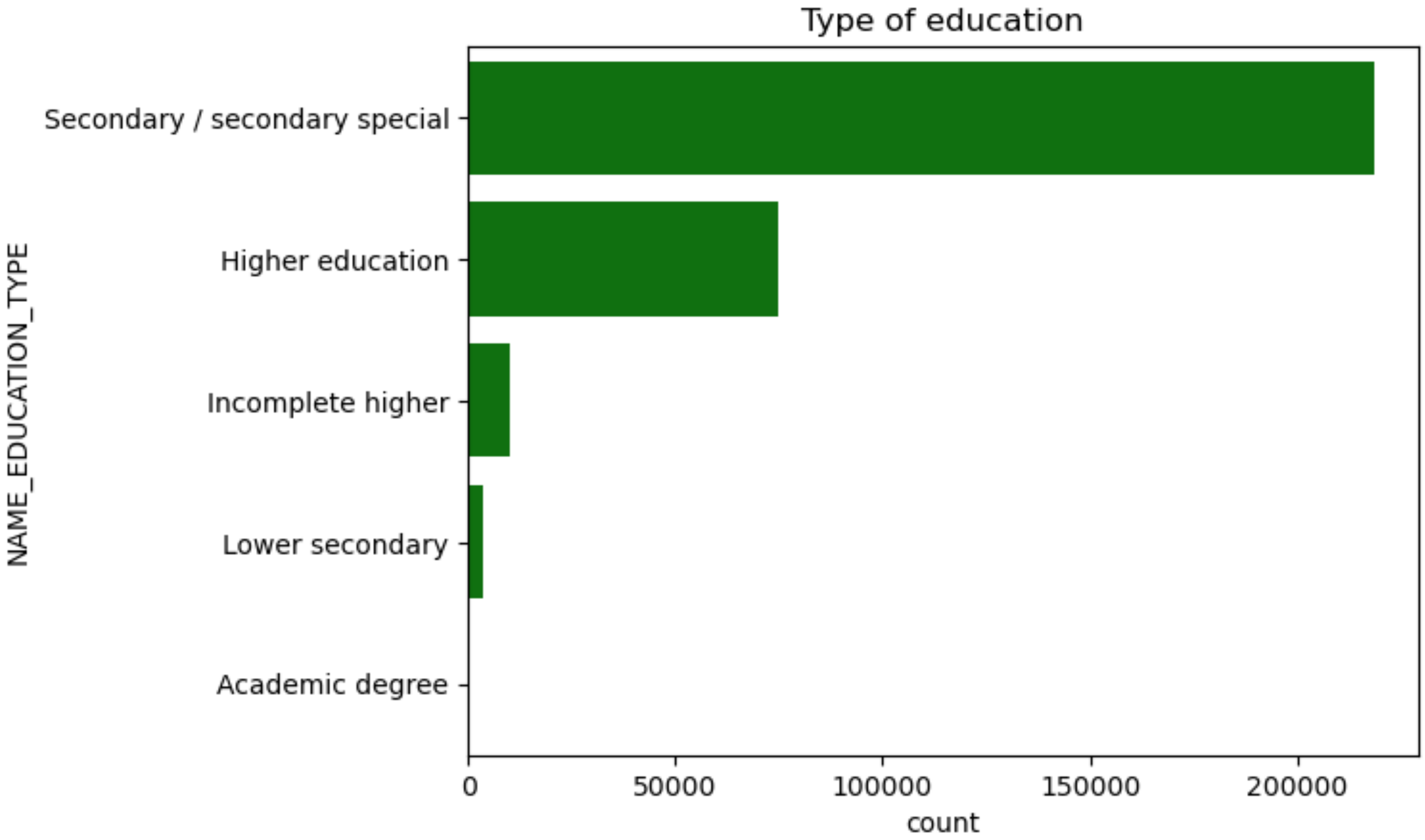
In [57]:
```
df_plot.head()
```

Out[57]:

	NAME_INCOME_TYPE	count
0	Working	158774
1	Commercial associate	71617
2	Pensioner	55362
3	State servant	21703
4	Unemployed	22

In [58]:

```
# Type of education

df_plot = (



          )
plotHorizBar(df_plot, 'count', 'NAME_EDUCATION_TYPE', 'green')
plt.title('Type of education')
plt.show()
```



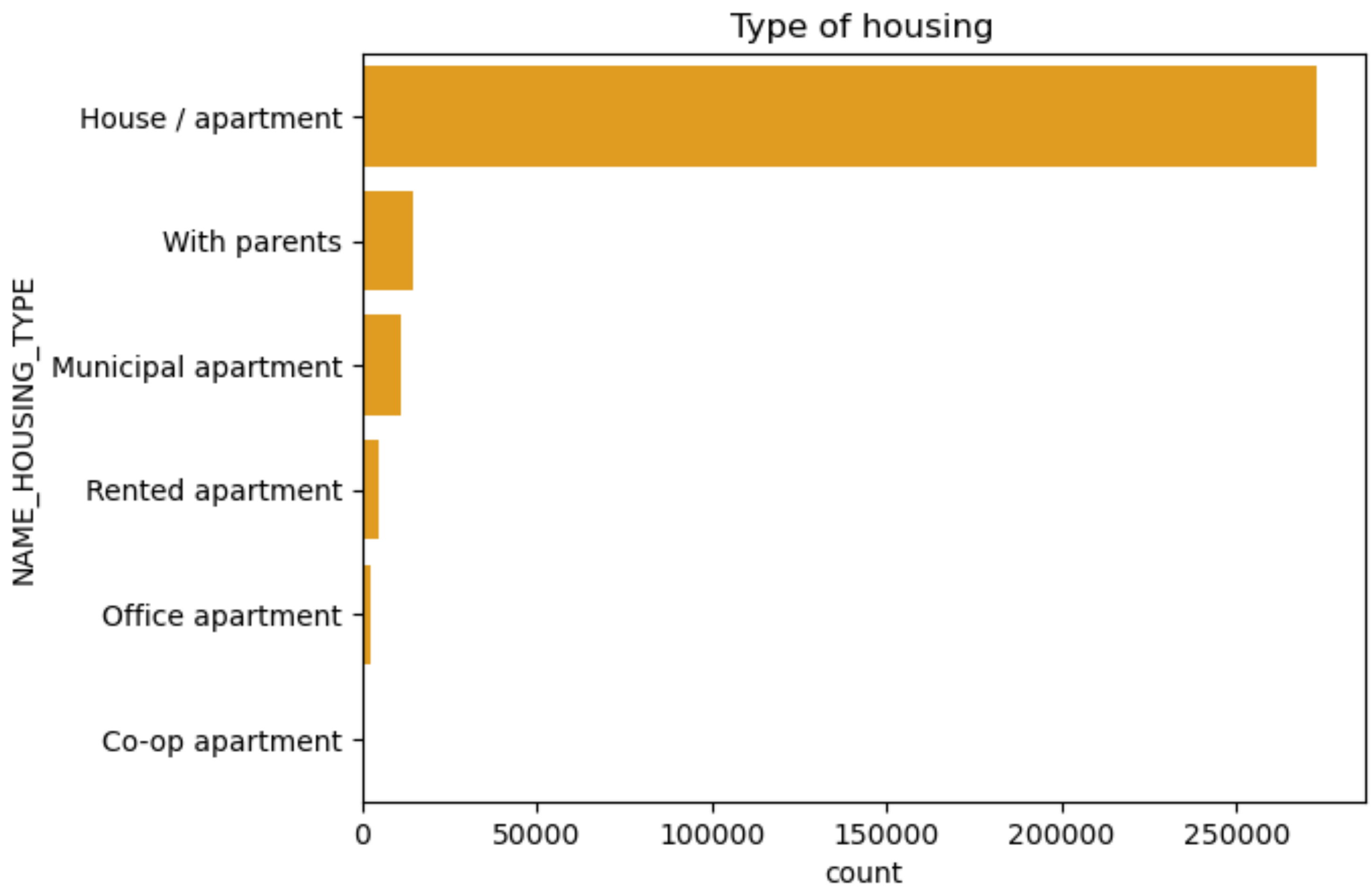
In [59]:
```
df_plot.head()
```

Out[59]:

	NAME_EDUCATION_TYPE	count
0	Secondary / secondary special	218391
1	Higher education	74863
2	Incomplete higher	10277
3	Lower secondary	3816
4	Academic degree	164

In [60]:

```
# Type of housing

df_plot = (



          )
plotHorizBar(df_plot, 'count', 'NAME_HOUSING_TYPE', 'orange')
plt.title('Type of housing')
plt.show()
```



In [61]:
```
df_plot.head()
```

Out[61]:

	NAME_HOUSING_TYPE	count
0	House / apartment	272868
1	With parents	14840
2	Municipal apartment	11183
3	Rented apartment	4881
4	Office apartment	2617

Columns selection, encoding and vector assembling

It is time to start thinking about which features/columns to use in the model, whether existing or new derived ones. To do so, the best we understand what the business is all about the better, including in relation to the characteristics of the data we are given. Statistics that we have made, and more we might do, would help to figure out patterns of interest.

Once the columns of interest for the classifer are set out, we have to enter into the specifics of the algorithms.

We are going to use `StringIndexer` and `OneHotEncoder`, as the ML algorithms we are about to use do require processing numbers not text. And because of those algorithms also requiring that all input features are contained within a single vector, we need a transformation. So we use the `VectorAsAssembler` transformer, already used above.

In order to select the features to use, one should also take into account the correlations among them. But as a starting experiment, let us use as many as possible.

Also, we have to make sure that the column target (binary label) is of numeric type. That is the case: `TARGET`.

Notes:

- MLlib provides a set of tools to help tackling this issue of features. See http://spark.apache.org/docs/latest/ml-features.html .
- Another useful Spark's functionality is Imputer, which completes missing values in a dataset, using the mean, median or mode of the columns in which the missing values are located. The input columns have to be of numeric type. At this moment, there is not

need given that we have excluded (wrongly) problematic columns. More on that in the section below regarding additional exercises.

In [62]:
```
cols_numeric
```

Out[62]:
```
['SK_ID_CURR',
 'TARGET',
 'CNT_CHILDREN',
 'AMT_INCOME_TOTAL',
 'AMT_CREDIT',
 'REGION_POPULATION_RELATIVE',
 'DAYS_BIRTH',
 'DAYS_EMPLOYED',
 'DAYS_REGISTRATION',
 'DAYS_ID_PUBLISH',
 'FLAG_MOBIL',
 'FLAG_EMP_PHONE',
 'FLAG_WORK_PHONE',
 'FLAG_CONT_MOBILE',
 'FLAG_PHONE',
 'FLAG_EMAIL',
 'REGION_RATING_CLIENT',
 'REGION_RATING_CLIENT_W_CITY',
 'HOUR_APPR_PROCESS_START',
 'REG_REGION_NOT_LIVE_REGION',
 'REG_REGION_NOT_WORK_REGION',
 'LIVE_REGION_NOT_WORK_REGION',
 'REG_CITY_NOT_LIVE_CITY',
 'REG_CITY_NOT_WORK_CITY',
 'LIVE_CITY_NOT_WORK_CITY']
```

In [63]:
```
cols_not_features = ['SK_ID_CURR', 'TARGET']

cols_non_numeric = [c for c in df_for_model.columns if c not in cols_numeric]
cols_non_numeric
```

```
Out[63]: ['NAME_CONTRACT_TYPE',
 'CODE_GENDER',
 'FLAG_OWN_CAR',
 'FLAG_OWN_REALTY',
 'NAME_INCOME_TYPE',
 'NAME_EDUCATION_TYPE',
 'NAME_FAMILY_STATUS',
 'NAME_HOUSING_TYPE',
 'WEEKDAY_APPR_PROCESS_START',
 'ORGANIZATION_TYPE']
```

```
In [64]: # Encoding columns and vector assembling them
# See Chapter 10 of the book "Learning Spark - Lightning-Fast Data Analytics"

categorical_cols = [i for i in cols_non_numeric if i not in cols_not_features]
non_categorical_cols = [i for i in cols_numeric if i not in cols_not_features]

index_output_cols = [x + ' Index' for x in categorical_cols]
ohe_output_cols = [x + ' OHE' for x in categorical_cols]

string_indexer = StringIndexer(inputCols=categorical_cols, outputCols=index_output_cols, handleInvalid="skip")

ohe_encoder = OneHotEncoder(inputCols=index_output_cols, outputCols=ohe_output_cols)

# Put all input features into a single vector, by using a transformer

assembler_inputs = ohe_output_cols + non_categorical_cols
vec_assembler = VectorAssembler(inputCols=assembler_inputs, outputCol="features")
assembler_inputs
```

Out[64]:
```
['NAME_CONTRACT_TYPE OHE',
 'CODE_GENDER OHE',
 'FLAG_OWN_CAR OHE',
 'FLAG_OWN_REALTY OHE',
 'NAME_INCOME_TYPE OHE',
 'NAME_EDUCATION_TYPE OHE',
 'NAME_FAMILY_STATUS OHE',
 'NAME_HOUSING_TYPE OHE',
 'WEEKDAY_APPR_PROCESS_START OHE',
 'ORGANIZATION_TYPE OHE',
 'CNT_CHILDREN',
 'AMT_INCOME_TOTAL',
 'AMT_CREDIT',
 'REGION_POPULATION_RELATIVE',
 'DAYS_BIRTH',
 'DAYS_EMPLOYED',
 'DAYS_REGISTRATION',
 'DAYS_ID_PUBLISH',
 'FLAG_MOBIL',
 'FLAG_EMP_PHONE',
 'FLAG_WORK_PHONE',
 'FLAG_CONT_MOBILE',
 'FLAG_PHONE',
 'FLAG_EMAIL',
 'REGION_RATING_CLIENT',
 'REGION_RATING_CLIENT_W_CITY',
 'HOUR_APPR_PROCESS_START',
 'REG_REGION_NOT_LIVE_REGION',
 'REG_REGION_NOT_WORK_REGION',
 'LIVE_REGION_NOT_WORK_REGION',
 'REG_CITY_NOT_LIVE_CITY',
 'REG_CITY_NOT_WORK_CITY',
 'LIVE_CITY_NOT_WORK_CITY']
```

In [65]:
```
categorical_cols
```

Out[65]:
```
['NAME_CONTRACT_TYPE',
 'CODE_GENDER',
 'FLAG_OWN_CAR',
 'FLAG_OWN_REALTY',
 'NAME_INCOME_TYPE',
 'NAME_EDUCATION_TYPE',
 'NAME_FAMILY_STATUS',
 'NAME_HOUSING_TYPE',
 'WEEKDAY_APPR_PROCESS_START',
 'ORGANIZATION_TYPE']
```

In [66]:
```
non_categorical_cols
```

Out[66]:
```
['CNT_CHILDREN',
 'AMT_INCOME_TOTAL',
 'AMT_CREDIT',
 'REGION_POPULATION_RELATIVE',
 'DAYS_BIRTH',
 'DAYS_EMPLOYED',
 'DAYS_REGISTRATION',
 'DAYS_ID_PUBLISH',
 'FLAG_MOBIL',
 'FLAG_EMP_PHONE',
 'FLAG_WORK_PHONE',
 'FLAG_CONT_MOBILE',
 'FLAG_PHONE',
 'FLAG_EMAIL',
 'REGION_RATING_CLIENT',
 'REGION_RATING_CLIENT_W_CITY',
 'HOUR_APPR_PROCESS_START',
 'REG_REGION_NOT_LIVE_REGION',
 'REG_REGION_NOT_WORK_REGION',
 'LIVE_REGION_NOT_WORK_REGION',
 'REG_CITY_NOT_LIVE_CITY',
 'REG_CITY_NOT_WORK_CITY',
 'LIVE_CITY_NOT_WORK_CITY']
```

Select and train model

Now it is time to train and test a model to be used for binary classification, that is, to decide whether there is a fraud or not.

We are going to use a Linear Support Vector Machine algorithm, as presented in

> http://spark.apache.org/docs/latest/ml-classification-regression.html#linear-support-vector-machine .

But at this point in time, probably it is worth considering to look at both the supervised learning and the ML pipeline slides from the lectures.

Partitioning of data

The step of creating a ML model means we should keep some part of the data in the dark. Basic standard split is 80/10/10 (or 70/15/15 if dataset is large), assuming a train/validation/test split.

Recall that if the validation part is relatively too small, then the model will memorize the data so it will reach an overfit situation. That would be bad as it no longer have data to evaluate how well it will generalize to unseen data. So, model performance is usually measured against a held-out test set consisting of examples that have never been seen before.

Also, notice that data highlighting dificluties in payments is less and imbalanced. Ideally, we should carry out better tuning for the data split, as it affects the performance of the model.

Hence, we will consider the following:

- Training dataset: 80% of examples used for model training
- Validation dataset: 20% to validate our models after the training and possibly decide on changes
- Test dataset: the dataset provided for test

In [67]:
```
# train/validation split

df_train, df_validation = df_for_model.randomSplit([0.8, 0.2], 42)
```

```
# Caching data ... just the training part as it is accessed many times by the algorithm
# But, it might not be a good idea if we are using a local computer and large dataset!
# df_train.cache()

# Print the number of rows in each part
print(f"There are {df_train.count()} rows in the training set and {df_validation.count()} in the validation set.'
```

```
There are 246240 rows in the training set and 61271 in the validation set.
```

Notice:

As we did before, we may consider storing the data split into files, should we want to use it elsewhere.

This relates to the need of guaranteeing unicity in a different environment. We leave it as it is now.

In [68]:
```
# Linear SVC algorithm
# default: featuresCol='features', labelCol='label', predictionCol='prediction'

lsvc = LinearSVC(maxIter=10, regParam=0.1, labelCol='TARGET')
```

ML pipeline configuration

In [69]:
```
# The pipeline holds four stages as set above:
#  1. string_indexer
#  2. ohe_encoder
#  3. vec_assembler (related to assembling features into vector)
#  4. lsvc (related to ML estimator)

pipeline = Pipeline(stages=[string_indexer, ohe_encoder, vec_assembler, lsvc])
```

Model fitting

Get the model (as transformer) by fitting the pipeline to the training data.

In [70]:
```python
pipeline_model = pipeline.fit(df_train)
```

Evaluate model

Let us evaluate the Linear SVM model that has been built.

Validating the model

It is time to apply the model built to validation data. Again, we will use the pipeline set above, meaning the stages already specified will be reused. Notice that, since the pipeline model is a transformer, we can easily apply it to validation data.

In [ ]:
```python
# Make predictions on validation data and show values of columns of interest

df_prediction = pipeline_model.transform(df_validation)

# Check its schema

df_prediction.printSchema()
```

In [ ]:
```python
# Columns to be focus on

df_prediction.select('features', 'rawPrediction', 'prediction', 'TARGET').show(truncate=False)
```

Evaluation metrics

How right is the model? Let us start to figure out by using:

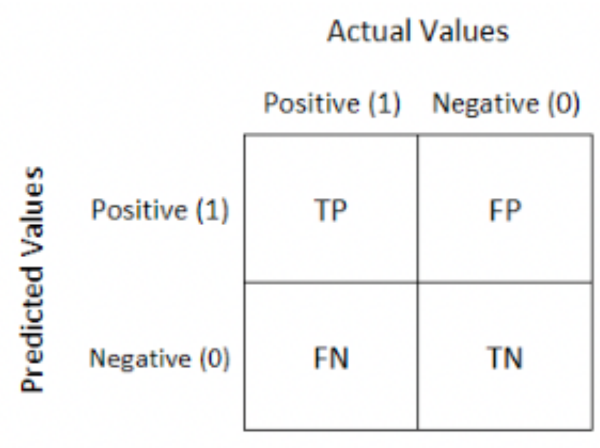
1. Specific evaluator
2. Confusion matrix

In [73]:
```
# Compute evaluation metrics on test data

prediction_label = df_prediction.select('rawPrediction', 'prediction', 'TARGET')

# supports metricName="areaUnderROC" (default) and "areaUnderPR"
# it relates to sensitivity (TP rate) and specificity (FP rate)

evaluator = BinaryClassificationEvaluator(labelCol='TARGET')

print("areaUnderROC = " + str(evaluator.evaluate(prediction_label)))
# print("areaUnderPR = " + str(evaluator.evaluate(prediction_label, {evaluator.metricName: 'areaUnderPR'})))
```

```
areaUnderROC = 0.600860425038266
```

Recalling the confusion matrix:

- True Positive: the prediction was positive and it is true.
- True Negative: the prediction was negative and it is true.
- False Positive: the prediction was positive and it is false.
- False Negative: the prediction was negative and it is false.



It follows the TP, TN, FP and FN computations.

In [74]:
```
# Counting rows for each case TP, TN, FP and FN respectively

n = df_prediction.count()
tp = df_prediction.filter(F.expr('prediction > 0') & F.expr('TARGET == prediction')).count()
tn = df_prediction.filter(                    ).count()
fp = df_prediction.filter(                    ).count()
fn = n - tp - tn - fp
[tp, tn, fp, fn, n]
```

Out[74]:
```
[1, 56209, 0, 5061, 61271]
```

Accuracy = (TP + TN) / (TP + TN + FP + FN)

How often the classifier is correct? (score)

Metric widely used but not so useful when there are many TN cases.

In [75]:
```
accuracy = (tp + tn) / (tp + tn + fp + fn)
```

Precision = TP / (TP + FP)

Positive predictive value - proportion of positive results that were correctly identified.

It removes NP and FN from consideration.

In [76]:
```
precision = tp / (tp + fp)
```

Recall = TP / (TP + FN)

True positive rate. (hit rate, sensitivity)

In [77]:
```
recall = tp / (tp + fn)
```

Specifity = TN / (TN + FP)

True negative rate. (selectivity)

In [78]:
```
specificity = tn / (tn + fp)
```

F1 score = 2 * Recall * Precision / (Recall + Precision)

Useful metric because it is difficult to compare two models with low precision and high recall or vice versa. Indeed, by combining recall and precision it helps to measure them at once.

In [79]:
```
f1_score = 2 * recall * precision / (recall + precision)
```

In [80]:
```
# Confusion matrix conclusions

print("TP = {}, TN = {}, FP = {}, FN = {}, Total = {}".format(tp, tn, fp, fn, n))
print("Accuracy = {}".format(accuracy))
print("Precison = {}".format(precision))
print("Recall = {}".format(recall))
print("Specificity = {}".format(specificity))
print("F1 score = {}".format(f1_score))
```

```
TP = 1, TN = 56209, FP = 0, FN = 5061, Total = 61271
Accuracy = 0.9173997486576031
Precison = 1.0
Recall = 0.00019755037534571315
Specificity = 1.0
F1 score = 0.0003950227138060438
```

Considerations:

1. *Which of the above metrics are most relevant for performance analysis in this particular study, considering the dataset that has been used?*

Classification of rare events is challenging in imbalanced datasets, as the TARGET variable highlights. In such scenarios, it is better to use the metric *area under the receiver operating characteristic curve (AUC)*, which evaluates ranking ability and it is particularly relevant in distinguishing between classes in imbalanced contexts. AUC measures the trade-off between the true positive rate and the false positive rate, so providing a more nuanced understanding of a model's capacity to identify rare events.

Notice that the metric *accuracy* for example reflects overall prediction correctness but it can be misleading by not accounting adequately for the performance on minority classes.

2. *Precision is very low due to the fact that the dataset is imbalanced, with true negatives having a much larger number of instances than true positives. In order to further improve results, it would be nice to train the model on more balanced datasets.*

Visual analysis

Plotting `prediction` obtained above versus `TARGET` .

In [81]:
```
# Plots

# We leave it as exercise
```

Saving the pipeline

In [82]:
```
# We can save the pipeline for further use should it be required

pipeline.save("pipeline-LinearSVM")

# later on, it can be loaded anywhere
```

In [83]:
```
# Furthermore, we can save the pipeline after fit, that is, the model

pipeline_model.
```

In [ ]:
```
ls -la
```

In [ ]:
```
ls -la pipeline-LinearSVM
```

Tune model

We should improve the model. For example, we can think about:

- How can we interpret the scores above?
- How to handle class imbalance, so clear in this dataset?
- Could a model with different set of features and/or target engineering would perform better?
- And what about using real-time data, that is, not training nor validation data? At least we must apply now the model to the test dataset considered earlier on.

Additional exercises

Test validation

To follow up the code above, now we can figure out how the model that has been built will perform against the given test dataset.

To do so, you may continue working in this notebook or create a new one for the purpose. Recall that

- the model is already stored in file;
- the test dataset is also stored in file, and accordingly to the train dataset that has been used for building the model.

ML model improvement

Create one or more notebooks with similar implementation as this one but using the following classifiers instead:

1. Logistic Regression
2. Decision Tree

Also, try to improve the process of feature/target engineering, given the framework set out. By the way, you should take into account correlations among features. Also, you may include in the model some features that were not used so far, like EXT_SOURCE_1, EXT_SOURCE_2, and EXT_SOURCE_3.

See related information in:

http://spark.apache.org/docs/latest/ml-classification-regression.html#logistic-regression (http://spark.apache.org/docs/latest/ml-classification-regression.html#logistic-regression)

http://spark.apache.org/docs/latest/ml-classification-regression.html#decision-tree-classifier (http://spark.apache.org/docs/latest/ml-classification-regression.html#decision-tree-classifier)

Profile of applicants of housing credits

Using also other datasets you find in the given zip archive, create a new notebook to profile applicants of housing credits. Make sure you have a careful working plan in order to reach the goals.

References

- Learning Spark - Lightning-Fast Data Analytics, 2nd Ed. J. Damji, B. Wenig, T. Das, and D. Lee. O'Reilly, 2020
- http://spark.apache.org/docs/latest/ml-guide.html
- https://docs.python.org/3/
- https://www.kaggle.com/competitions/home-credit-default-risk/

In [ ]:

```

```