Libraries with fixed versions and data loading

In [ ]:

In [ ]:
```
# %pip install xgboost==1.6.2
# %pip install shap==0.41.0
# %pip install pandas==1.3.5
# %pip install plotly==5.10.0
# %pip install scikit-learn==1.0.2
# %pip install seaborn
# %pip install graphviz
# %pip install wheel
# %pip install sklearn
# %pip install glmnet
```

In [ ]:
```
import xgboost as xgb
import pandas as pd
import numpy as np
import shap
import plotly.express as px

from sklearn.model_selection import train_test_split
from sklearn import metrics
```

In [ ]:
```
# # Enable export of plotly interactivity in html export
# from plotly.offline import iplot, init_notebook_mode
# init_notebook_mode(connected = True)
```

To load the data, you can simply put it on Google Drive and read it in a few lines of code (there are multiple other ways to do it).

For speed, we can load only the provided 26 columns (1 target, 1 ID and 24 features).

In [ ]:
```
#from google.colab import drive
#drive.mount('/content/drive')
```

In [ ]:
```
ap_train = pd.read_csv('Data/CSV/application_train.csv',
                       usecols =  ['TARGET', 'SK_ID_CURR', 'NAME_CONTRACT_TYPE', 'CODE
                                   'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TOTA
                                   'AMT_GOODS_PRICE', 'NAME_TYPE_SUITE', 'NAME_INCOME_
                                   'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'REGION_
                                   'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'DAYS_ID_PUBL
                                   'EXT_SOURCE_2', 'EXT_SOURCE_3'])
print(f"Application train dataset shape: {ap_train.shape}")
```

```
Application train dataset shape: (307511, 26)
```

For simplicity, we'll create a AGE (in years) variable from DAYS_BIRTH (in days).

In [ ]:
```
ap_train['AGE'] = -round(ap_train['DAYS_BIRTH']/365)
ap_train.drop(columns='DAYS_BIRTH', inplace=True)
```

[#1] Interesting variables

In [ ]:
```
int_cols = ["CODE_GENDER", "CNT_CHILDREN", "AMT_INCOME_TOTAL", "NAME_INCOME_TYPE",
```

[#2] Interesting variables with respect to target

In [ ]:
```
ap_train.dtypes
```

Out[ ]:
```
SK_ID_CURR                    int64
TARGET                        int64
NAME_CONTRACT_TYPE           object
CODE_GENDER                  object
FLAG_OWN_CAR                 object
FLAG_OWN_REALTY              object
CNT_CHILDREN                  int64
AMT_INCOME_TOTAL            float64
AMT_CREDIT                  float64
AMT_ANNUITY                 float64
AMT_GOODS_PRICE             float64
NAME_TYPE_SUITE              object
NAME_INCOME_TYPE             object
NAME_EDUCATION_TYPE          object
NAME_FAMILY_STATUS           object
NAME_HOUSING_TYPE            object
REGION_POPULATION_RELATIVE  float64
DAYS_EMPLOYED                 int64
DAYS_REGISTRATION           float64
DAYS_ID_PUBLISH               int64
OWN_CAR_AGE                 float64
OCCUPATION_TYPE              object
EXT_SOURCE_1                float64
EXT_SOURCE_2                float64
EXT_SOURCE_3                float64
AGE                         float64
dtype: object
```

In [ ]:
```
ap_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Data columns (total 26 columns):
 #   Column                       Non-Null Count   Dtype  
---  ------                       --------------   -----  
 0   SK_ID_CURR                   307511 non-null  int64  
 1   TARGET                       307511 non-null  int64  
 2   NAME_CONTRACT_TYPE           307511 non-null  object 
 3   CODE_GENDER                  307511 non-null  object 
 4   FLAG_OWN_CAR                 307511 non-null  object 
 5   FLAG_OWN_REALTY              307511 non-null  object 
 6   CNT_CHILDREN                 307511 non-null  int64  
 7   AMT_INCOME_TOTAL             307511 non-null  float64
 8   AMT_CREDIT                   307511 non-null  float64
 9   AMT_ANNUITY                  307499 non-null  float64
 10  AMT_GOODS_PRICE              307233 non-null  float64
 11  NAME_TYPE_SUITE              306219 non-null  object 
 12  NAME_INCOME_TYPE             307511 non-null  object 
 13  NAME_EDUCATION_TYPE          307511 non-null  object 
 14  NAME_FAMILY_STATUS           307511 non-null  object 
 15  NAME_HOUSING_TYPE            307511 non-null  object 
 16  REGION_POPULATION_RELATIVE   307511 non-null  float64
 17  DAYS_EMPLOYED                307511 non-null  int64  
 18  DAYS_REGISTRATION            307511 non-null  float64
 19  DAYS_ID_PUBLISH              307511 non-null  int64  
 20  OWN_CAR_AGE                  104582 non-null  float64
 21  OCCUPATION_TYPE              211120 non-null  object 
 22  EXT_SOURCE_1                 134133 non-null  float64
 23  EXT_SOURCE_2                 306851 non-null  float64
 24  EXT_SOURCE_3                 246546 non-null  float64
 25  AGE                          307511 non-null  float64
dtypes: float64(11), int64(5), object(10)
memory usage: 61.0+ MB
```

In [ ]:
```
ap_train.describe()
```

Out[ ]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_A
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	30749
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	2710
std	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	1449
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	161
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	1652
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	2490
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	3459
max	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	25802

In [ ]:
```
int_cols_con = []
int_cols_cat = []

for col in int_cols:
    if ap_train[col].dtype == object:
        int_cols_cat.append(col)
```

```
    else:
        int_cols_con.append(col)
```

In [ ]:
```
int_cols_con
```

Out[ ]:
```
['CNT_CHILDREN', 'AMT_INCOME_TOTAL', 'AGE']
```

In [ ]:
```
int_cols_cat
```

Out[ ]:
```
['CODE_GENDER',
 'NAME_INCOME_TYPE',
 'NAME_EDUCATION_TYPE',
 'NAME_FAMILY_STATUS',
 'NAME_HOUSING_TYPE']
```

In [ ]:
```
df_binned = pd.DataFrame()
df_binned["TARGET"] = ap_train["TARGET"]
for col in int_cols:
    df_binned[col] = ap_train[col]

df_binned = df_binned[df_binned.CODE_GENDER != "XNA"]
```

In [ ]:
```
print(df_binned.shape)
```

```
(307507, 9)
```

In [ ]:
```
def cap_data(dati, col_list):
    for col in col_list:
        print("capping the ",col)
        if (((dati[col].dtype)=='float64') | ((dati[col].dtype)=='int64')):
            percentiles = dati[col].quantile([0.01,0.99]).values
            dati[col][dati[col] <= percentiles[0]] = percentiles[0]
            dati[col][dati[col] >= percentiles[1]] = percentiles[1]
        else:
            dati[col]=dati[col]
    return dati

df_binned=cap_data(df_binned, int_cols_con)
```

```
capping the  CNT_CHILDREN
capping the  AMT_INCOME_TOTAL
capping the  AGE
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
```

In [ ]:
```
df_binned.shape
```

Out[ ]:
```
(307507, 9)
```

In [ ]:
```
for col in int_cols_con:
    df_binned[col] = pd.cut(df_binned[col], 6, precision=0)
```

In [ ]:
```
df_binned["CNT_CHILDREN"].describe()
```

Out[ ]:
```
count            307507
unique                4
top       (-0.003, 0.5]
freq             215369
Name: CNT_CHILDREN, dtype: object
```

In [ ]:
```
df_binned["AMT_INCOME_TOTAL"]
```

Out[ ]:
```
0         (187500.0, 258750.0]
1         (258750.0, 330000.0]
2          (44572.0, 116250.0]
3         (116250.0, 187500.0]
4         (116250.0, 187500.0]
                  ...
307506    (116250.0, 187500.0]
307507     (44572.0, 116250.0]
307508    (116250.0, 187500.0]
307509    (116250.0, 187500.0]
307510    (116250.0, 187500.0]
Name: AMT_INCOME_TOTAL, Length: 307507, dtype: category
Categories (6, interval[float64, right]): [(44572.0, 116250.0] < (116250.0, 18750
0.0] < (187500.0, 258750.0] < (258750.0, 330000.0] < (330000.0, 401250.0] < (40125
0.0, 472500.0]]
```

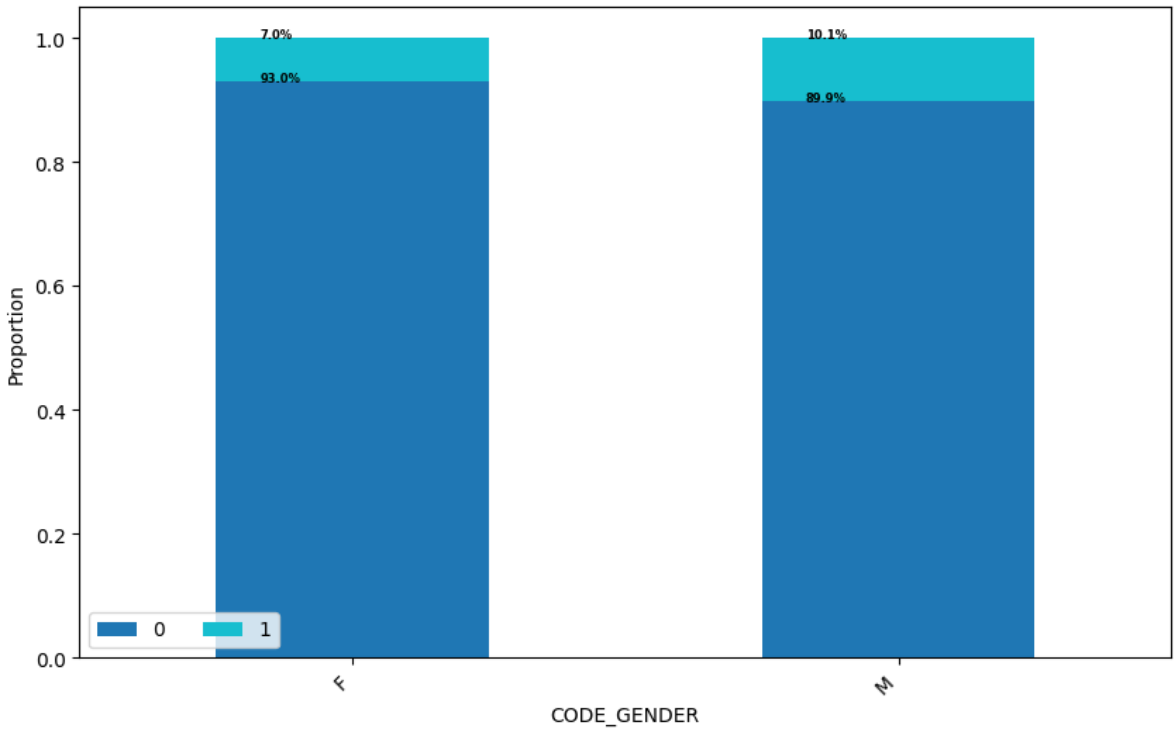
In [ ]:
```
df_binned["TARGET"].describe()
```

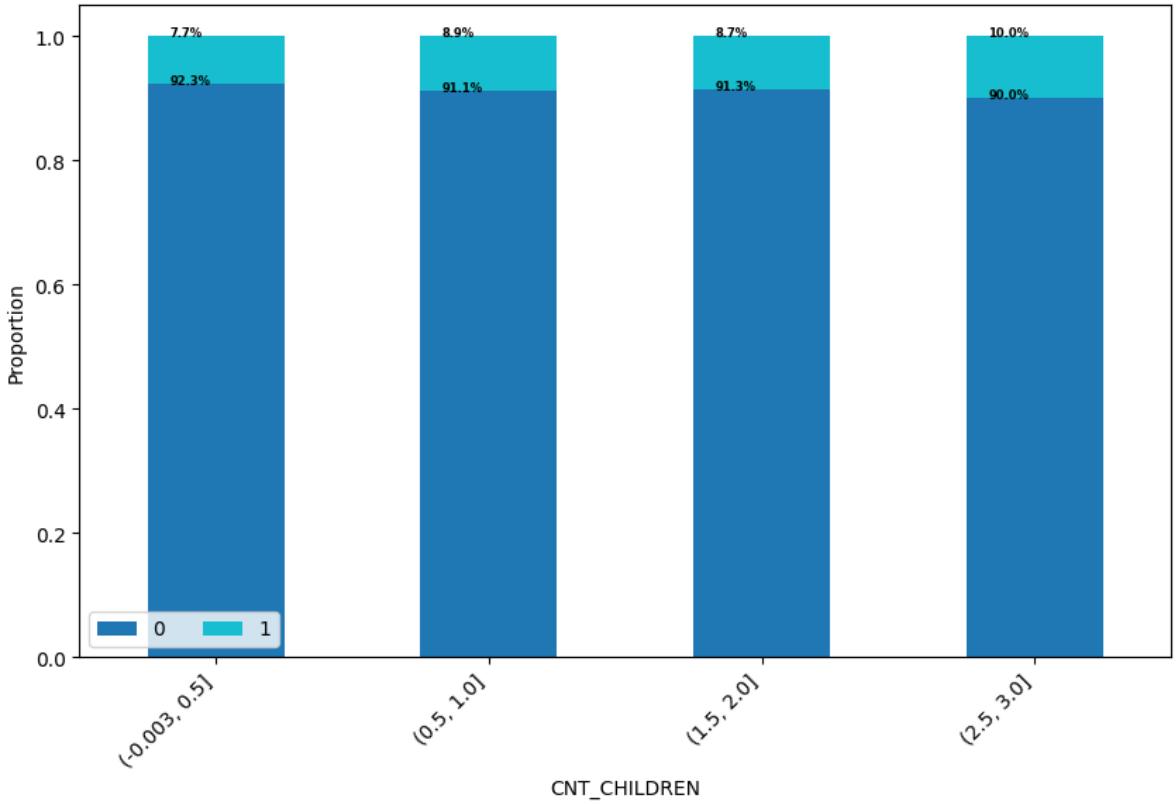
Out[ ]:
```
count    307507.00000
mean          0.08073
std           0.27242
min           0.00000
25%           0.00000
50%           0.00000
75%           0.00000
max           1.00000
Name: TARGET, dtype: float64
```

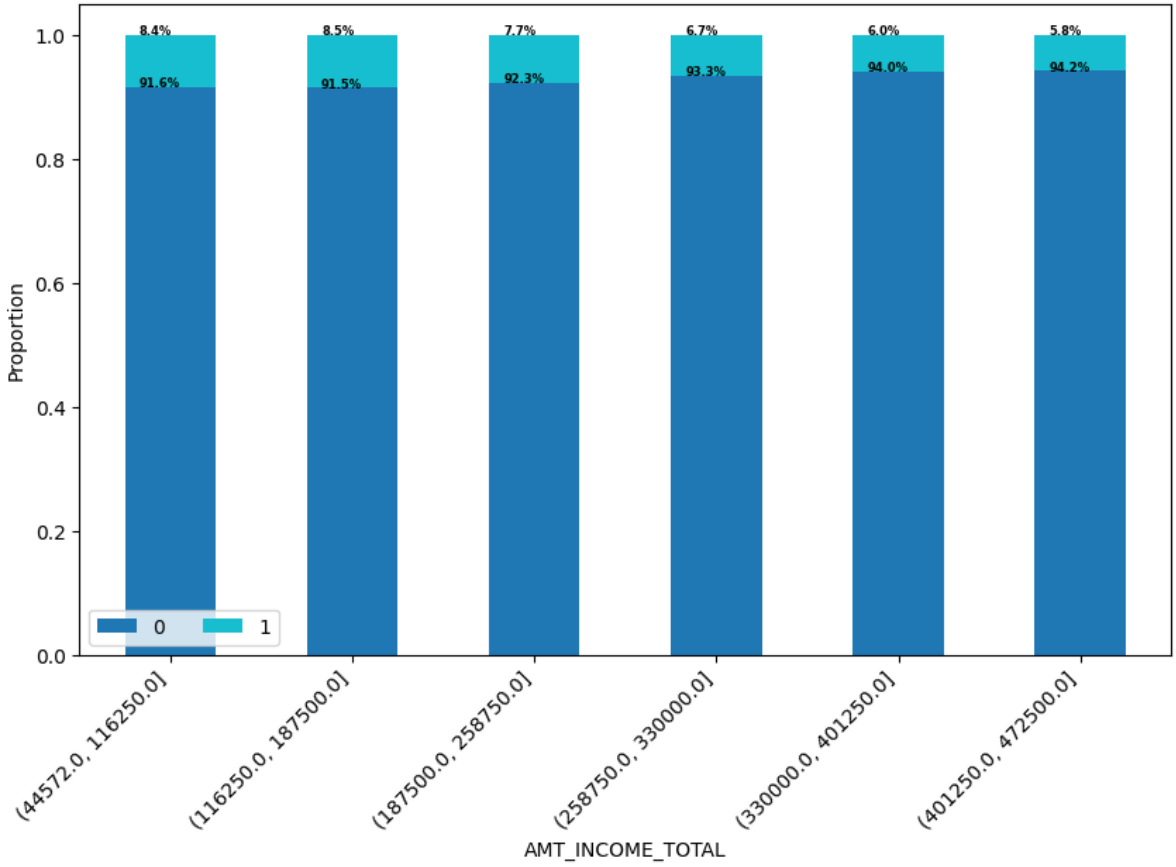
In [ ]:
```
import os

outpath = 'Plots'

# Check whether the specified path exists or not
isExist = os.path.exists(outpath)

if not isExist:
  # Create a new directory because it does not exist
  os.makedirs(outpath)
  print("The new directory is created!")

os.chdir(outpath)
```

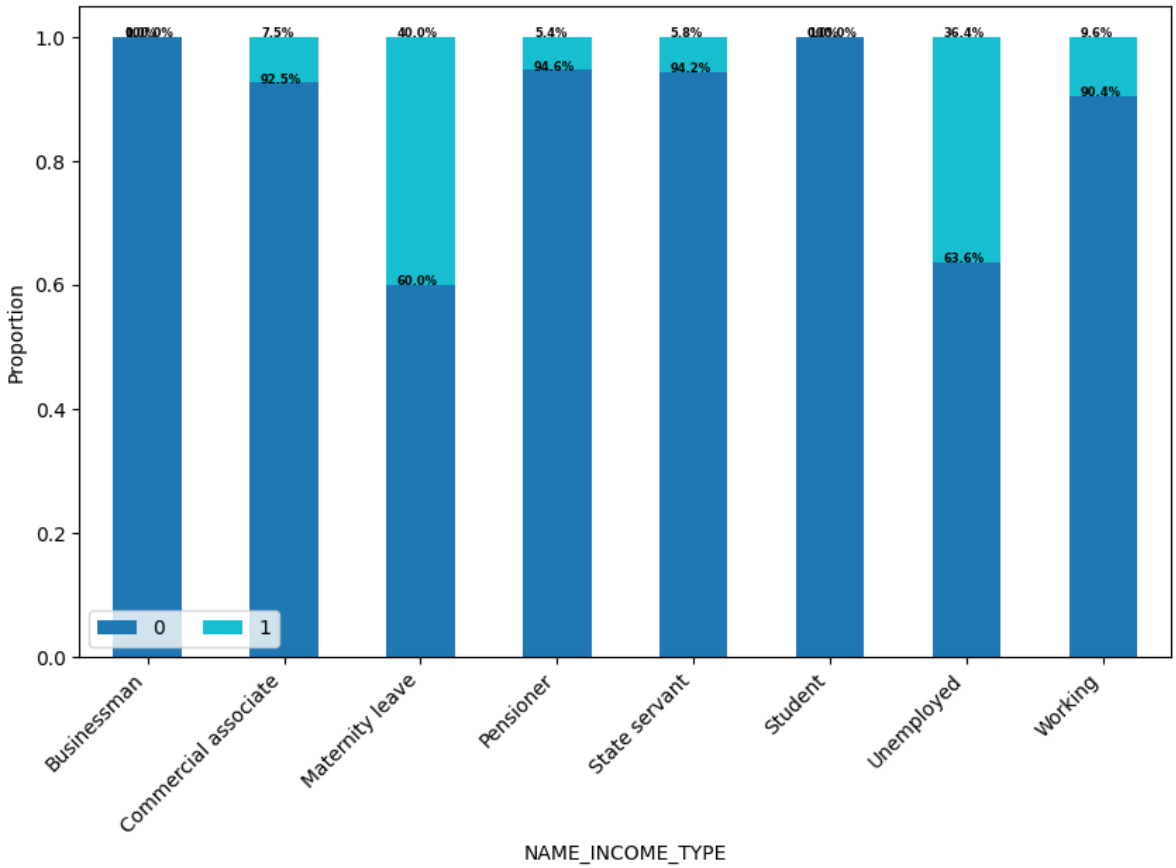
In [ ]:
```
import matplotlib.pyplot as plt



for col in int_cols:
    cross_tab_prop = pd.crosstab(index=df_binned[col],
                                 columns=df_binned["TARGET"],
                                     normalize="index")
    cross_tab_prop

    cross_tab = pd.crosstab(index=df_binned[col],
```

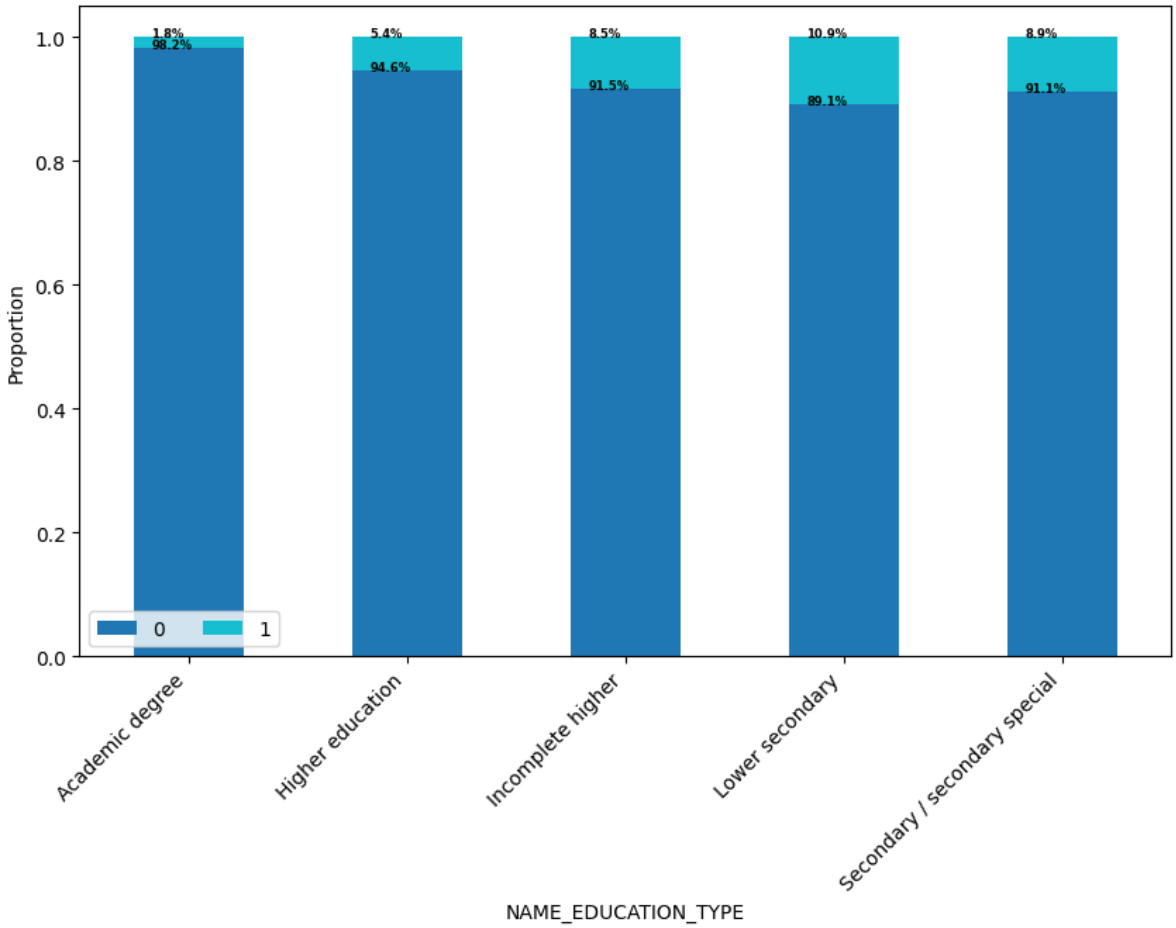
```
                            columns=df_binned["TARGET"])
cross_tab

cross_tab_prop.plot(kind='bar',
                    stacked=True,
                    colormap='tab10',
                    figsize=(10, 6))

plt.legend(loc="lower left", ncol=2)
plt.xlabel(str(col))
plt.ylabel("Proportion")
plt.xticks(rotation=45, ha='right')


for n, x in enumerate([*cross_tab.index.values]):
    for (proportion, y_loc) in zip(cross_tab_prop.loc[x],
                                   cross_tab_prop.loc[x].cumsum()):

        plt.text(x=n - 0.17,
                 y=y_loc,
                 s=f'{np.round(proportion * 100, 1)}%',
                 color="black",
                 fontsize=6,
                 fontweight="bold")

plt.savefig("ORIGINAL_" + str(col), bbox_inches = "tight")
plt.show()
```

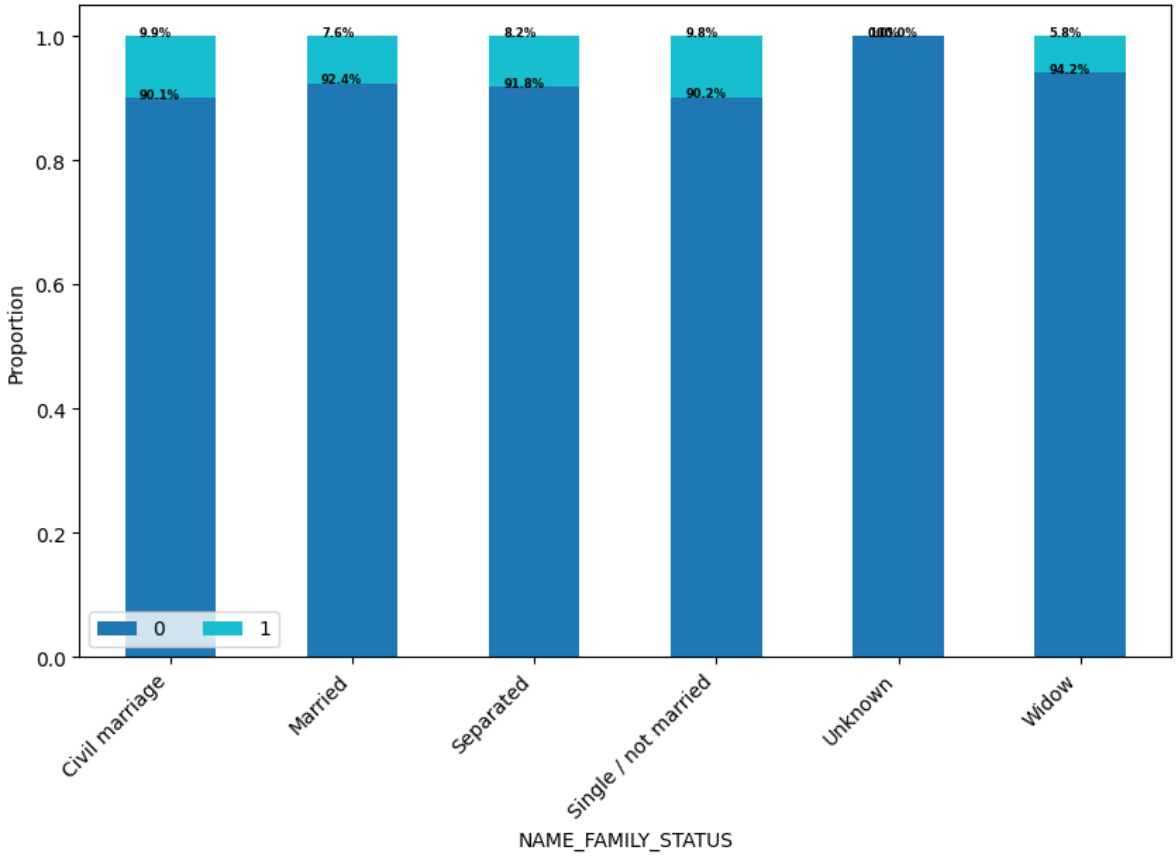


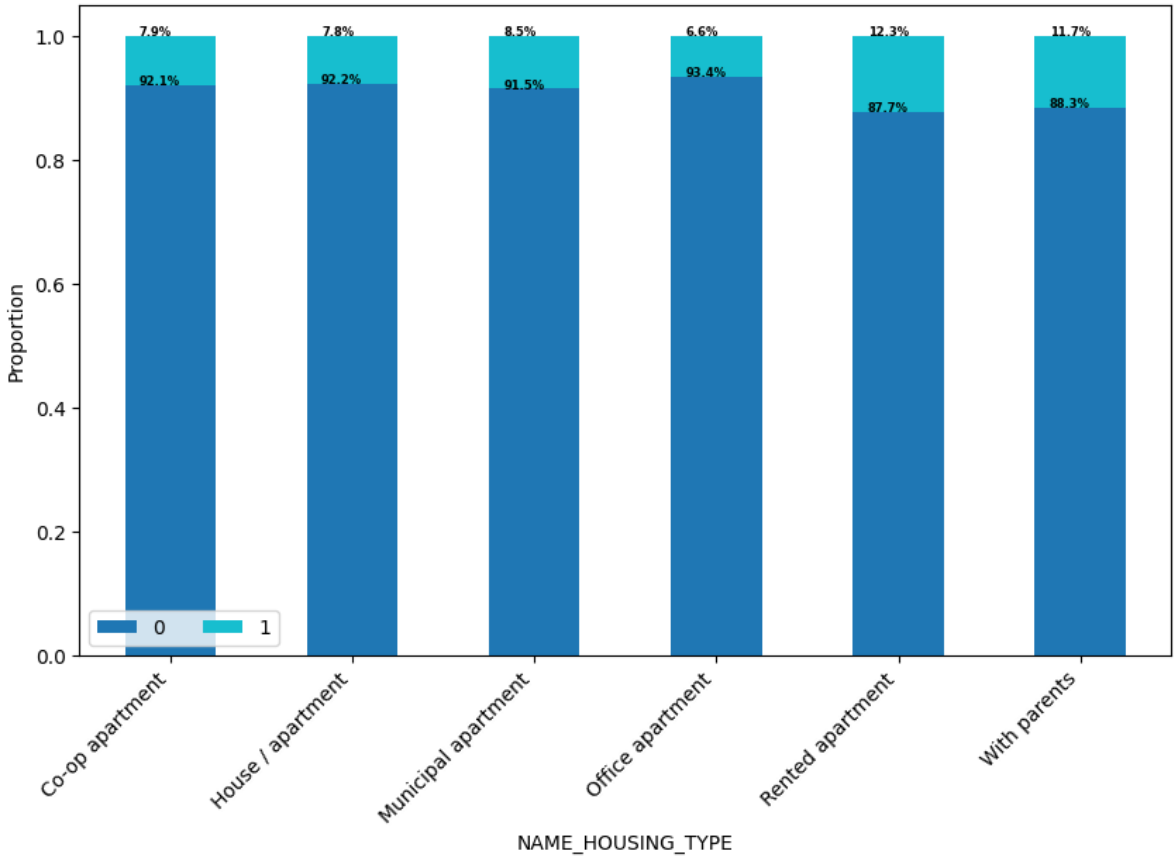

7.7%
8.9%
8.7%
10.0%
92.3%
91.1%
91.3%
90.0%
Proportion
0.0
0.2
0.4
0.6
0.8
1.0
0
1
(-0.003, 0.5]
(0.5, 1.0]
(1.5, 2.0]
(2.5, 3.0]
CNT_CHILDREN

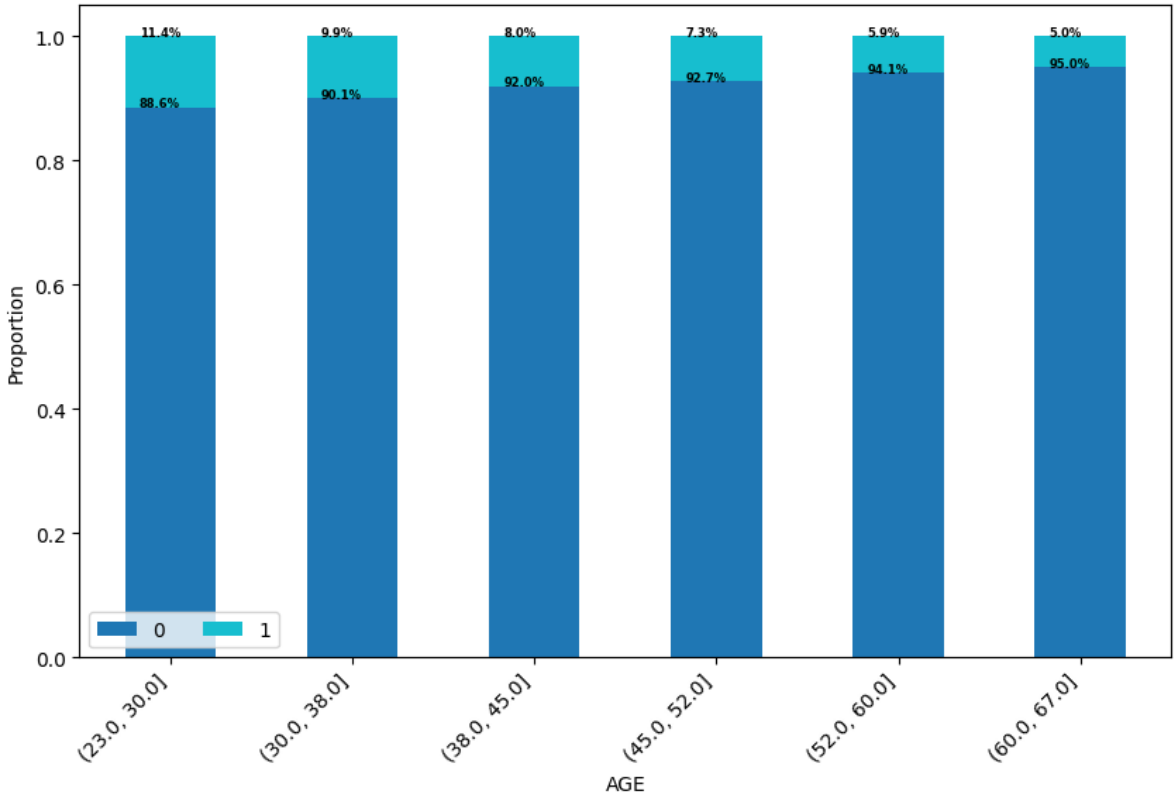

8.4%
8.5%
7.7%
6.7%
6.0%
5.8%
91.6%
91.5%
92.3%
93.3%
94.0%
94.2%
Proportion
0.0
0.2
0.4
0.6
0.8
1.0
0
1
(44572.0, 116250.0]
(116250.0, 187500.0]
(187500.0, 258750.0]
(258750.0, 330000.0]
(330000.0, 401250.0]
(401250.0, 472500.0]
AMT_INCOME_TOTAL


Proportion
1.0
0.8
0.6
0.4
0.2
0.0
7.5%
92.5%
40.0%
60.0%
5.4%
94.6%
5.8%
94.2%
36.4%
63.6%
9.6%
90.4%
0
1
Businessman
Commercial associate
Maternity leave
Pensioner
State servant
Student
Unemployed
Working
NAME_INCOME_TYPE


Proportion
1.0
0.8
0.6
0.4
0.2
0.0
1.8%
98.2%
5.4%
94.6%
8.5%
91.5%
10.9%
89.1%
8.9%
91.1%
0
1
Academic degree
Higher education
Incomplete higher
Lower secondary
Secondary / secondary special
NAME_EDUCATION_TYPE


9.9%
7.6%
8.2%
9.8%
5.8%
90.1%
92.4%
91.8%
90.2%
94.2%
Proportion
1.0
0.8
0.6
0.4
0.2
0.0
0
1
Civil marriage
Married
Separated
Single / not married
Unknown
Widow
NAME_FAMILY_STATUS


7.9%
7.8%
8.5%
6.6%
12.3%
11.7%
92.1%
92.2%
91.5%
93.4%
87.7%
88.3%
Proportion
1.0
0.8
0.6
0.4
0.2
0.0
0
1
Co-op apartment
House / apartment
Municipal apartment
Office apartment
Rented apartment
With parents
NAME_HOUSING_TYPE



Preparing the data

Let's prepare the data for a simple ML model.

First of all, feature encoding.

In [ ]:
```
ap_train_unbiased = ap_train.copy(deep = True)
```

In [ ]:
```
ap_train['CODE_GENDER_M'] = np.select([ap_train['CODE_GENDER'] == 'M', ap_train['CO
ap_train['FLAG_OWN_CAR'] = np.where(ap_train['FLAG_OWN_CAR'] == 'Y', 1, 0)
ap_train['FLAG_OWN_REALTY'] = np.where(ap_train['FLAG_OWN_REALTY'] == 'Y', 1, 0)
ap_train.drop(columns='CODE_GENDER', inplace=True)
```

In [ ]:
```
ap_objects = list(ap_train.select_dtypes(include=['object']).columns)
ap_train[ap_objects] = ap_train[ap_objects].astype('category')
```

Let's separate the target from the rest of the data

In [ ]:
```
ap_train_target = ap_train.pop('TARGET')
print(f"Target dataset shape: {ap_train_target.shape}")
```

```
Target dataset shape: (307511,)
```

Let's split the original dataset in two:

- 80% for the train dataset
- 20% for the test one

In [ ]:
```
df_train, df_test, df_target_train, df_target_test = train_test_split(
    ap_train, ap_train_target, test_size=0.2, stratify=ap_train_target, random_stat
```

```
print(f"Train dataset shape: {df_train.shape}")
print(f"Test dataset shape: {df_test.shape}")
```

```
Train dataset shape: (246008, 25)
Test dataset shape: (61503, 25)
```

In [ ]:
```
df_train.dtypes
```

Out[ ]:
```
SK_ID_CURR                      int64
NAME_CONTRACT_TYPE           category
FLAG_OWN_CAR                    int32
FLAG_OWN_REALTY                 int32
CNT_CHILDREN                    int64
AMT_INCOME_TOTAL              float64
AMT_CREDIT                    float64
AMT_ANNUITY                   float64
AMT_GOODS_PRICE               float64
NAME_TYPE_SUITE              category
NAME_INCOME_TYPE             category
NAME_EDUCATION_TYPE          category
NAME_FAMILY_STATUS           category
NAME_HOUSING_TYPE            category
REGION_POPULATION_RELATIVE    float64
DAYS_EMPLOYED                   int64
DAYS_REGISTRATION             float64
DAYS_ID_PUBLISH                 int64
OWN_CAR_AGE                   float64
OCCUPATION_TYPE              category
EXT_SOURCE_1                  float64
EXT_SOURCE_2                  float64
EXT_SOURCE_3                  float64
AGE                           float64
CODE_GENDER_M                 float64
dtype: object
```

In [ ]:
```
cat_cols = []
for col in df_train.columns:
    if df_train[col].dtype == "category":
        cat_cols.append(col)
cat_cols
```

Out[ ]:
```
['NAME_CONTRACT_TYPE',
 'NAME_TYPE_SUITE',
 'NAME_INCOME_TYPE',
 'NAME_EDUCATION_TYPE',
 'NAME_FAMILY_STATUS',
 'NAME_HOUSING_TYPE',
 'OCCUPATION_TYPE']
```

In [ ]:
```
from sklearn import preprocessing

lbl = preprocessing.LabelEncoder()

for col in cat_cols:
    df_train[col] = lbl.fit_transform(df_train[col].astype(str))
```

Create a basic ML model and scoring on the test set

I'll train a simple XGBoost model (parameters previously chosen with cross-validation).

In [ ]:
```
df_train_dmatrix = xgb.DMatrix(df_train.drop(columns='SK_ID_CURR'), label = df_targ

param = {'max_depth':6,
         'eta':.2,
         'subsample':.9,
         'colsample_bytree':.9,
         'scale_pos_weight':10,
         'objective':'binary:logistic',
         'tree_method':'exact'}
xgb_base_model = xgb.train(param, df_train_dmatrix, num_boost_round=50)
```

Let's score the test set.

In [ ]:
```
df_test_dmatrix = xgb.DMatrix(df_test.drop(columns='SK_ID_CURR'), enable_categorica
xgb_base_test_results = xgb_base_model.predict(df_test_dmatrix)
```

What are the risk scores (from 0 to 1) of the first 5 customers in the test set? And what's the overall AUC on the test set?

In [ ]:
```
xgb_base_test_results[:5]
```

Out[ ]:
```
array([0.412908  , 0.3104497 , 0.7822344 , 0.27786046, 0.47344765],
      dtype=float32)
```

In [ ]:
```
fpr, tpr, thresholds = metrics.roc_curve(df_target_test, xgb_base_test_results, pos
metrics.auc(fpr, tpr)
```

Out[ ]:
```
0.7533808950317153
```

The third customer has a higher risk score (0.796) than the others in the first 5 records.

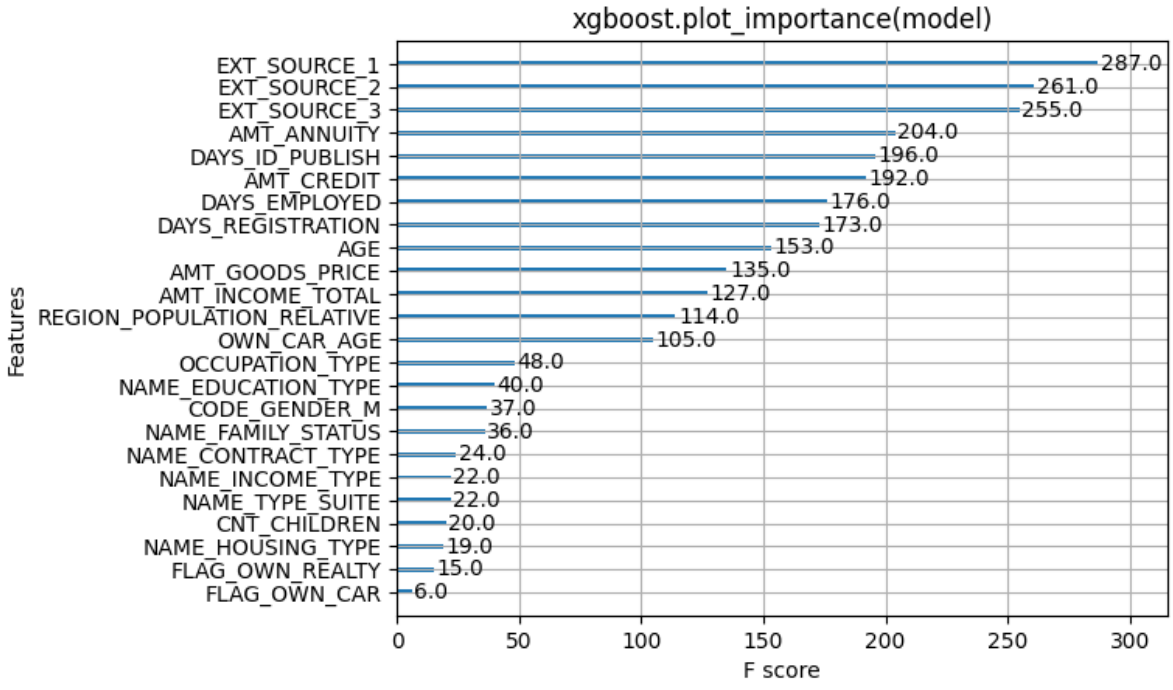
This simple model has an AUC on the test set around **0.754**, a decent baseline performance.

[#3] Evaluating feature importance

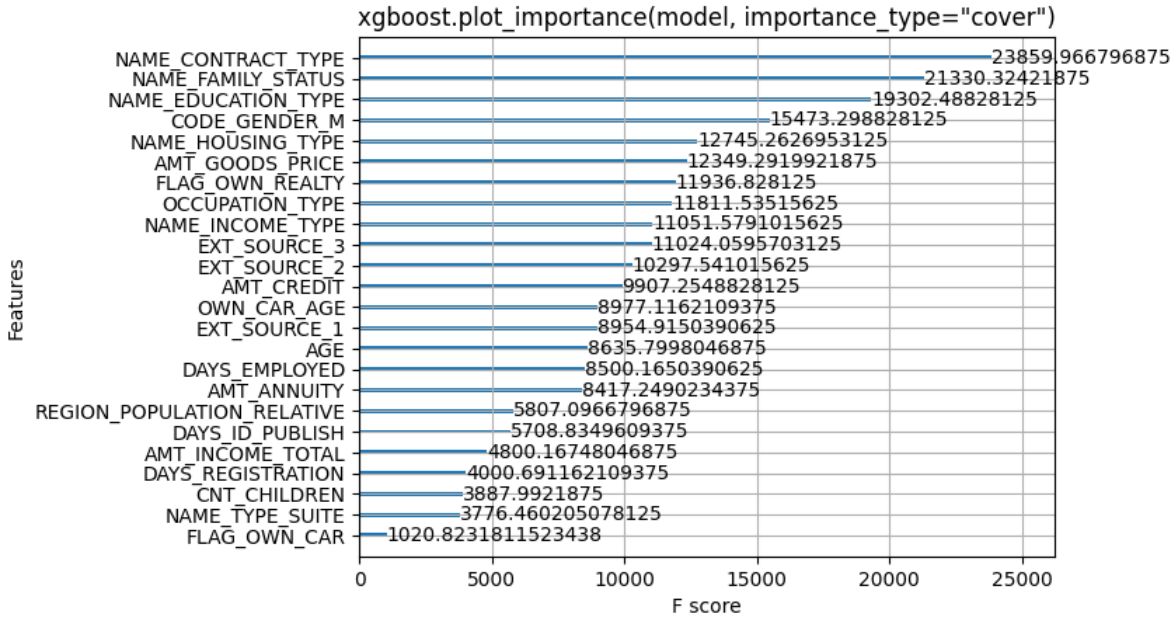
In [ ]:
```
xgb_base_model.get_fscore()
```

Out[ ]:
```
{'NAME_CONTRACT_TYPE': 24.0,
 'FLAG_OWN_CAR': 6.0,
 'FLAG_OWN_REALTY': 15.0,
 'CNT_CHILDREN': 20.0,
 'AMT_INCOME_TOTAL': 127.0,
 'AMT_CREDIT': 192.0,
 'AMT_ANNUITY': 204.0,
 'AMT_GOODS_PRICE': 135.0,
 'NAME_TYPE_SUITE': 22.0,
 'NAME_INCOME_TYPE': 22.0,
 'NAME_EDUCATION_TYPE': 40.0,
 'NAME_FAMILY_STATUS': 36.0,
 'NAME_HOUSING_TYPE': 19.0,
 'REGION_POPULATION_RELATIVE': 114.0,
 'DAYS_EMPLOYED': 176.0,
 'DAYS_REGISTRATION': 173.0,
 'DAYS_ID_PUBLISH': 196.0,
 'OWN_CAR_AGE': 105.0,
 'OCCUPATION_TYPE': 48.0,
 'EXT_SOURCE_1': 287.0,
 'EXT_SOURCE_2': 261.0,
 'EXT_SOURCE_3': 255.0,
 'AGE': 153.0,
 'CODE_GENDER_M': 37.0}
```

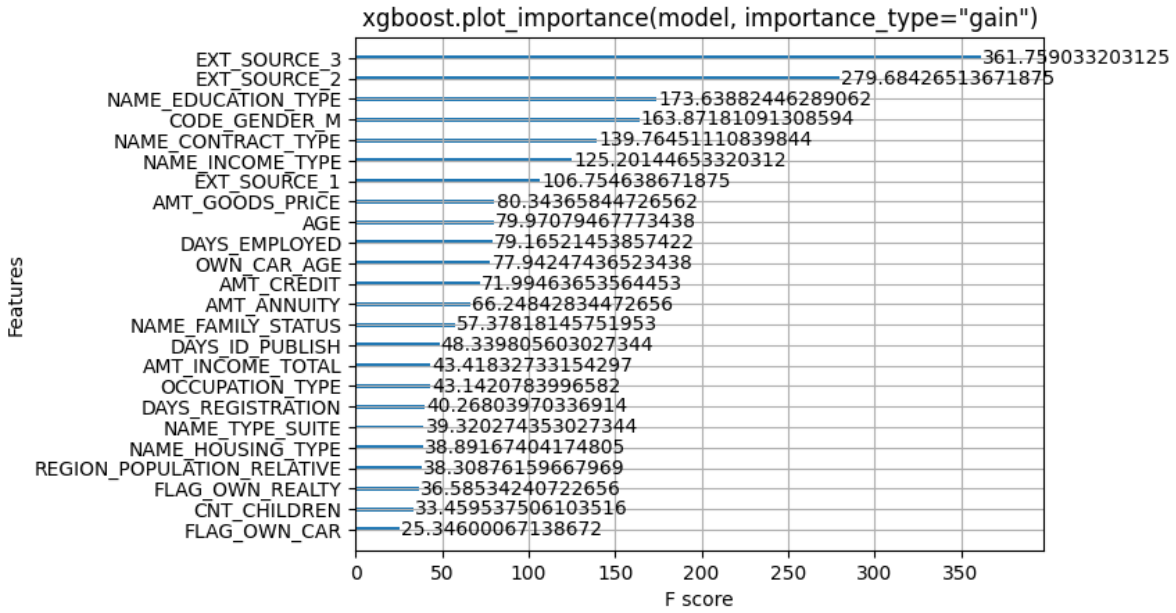
In [ ]:
```
xgb.plot_importance(xgb_base_model)
plt.title("xgboost.plot_importance(model)")
plt.show()
```



In [ ]:
```
xgb.plot_importance(xgb_base_model, importance_type="cover")
plt.title('xgboost.plot_importance(model, importance_type="cover")')
plt.show()
```

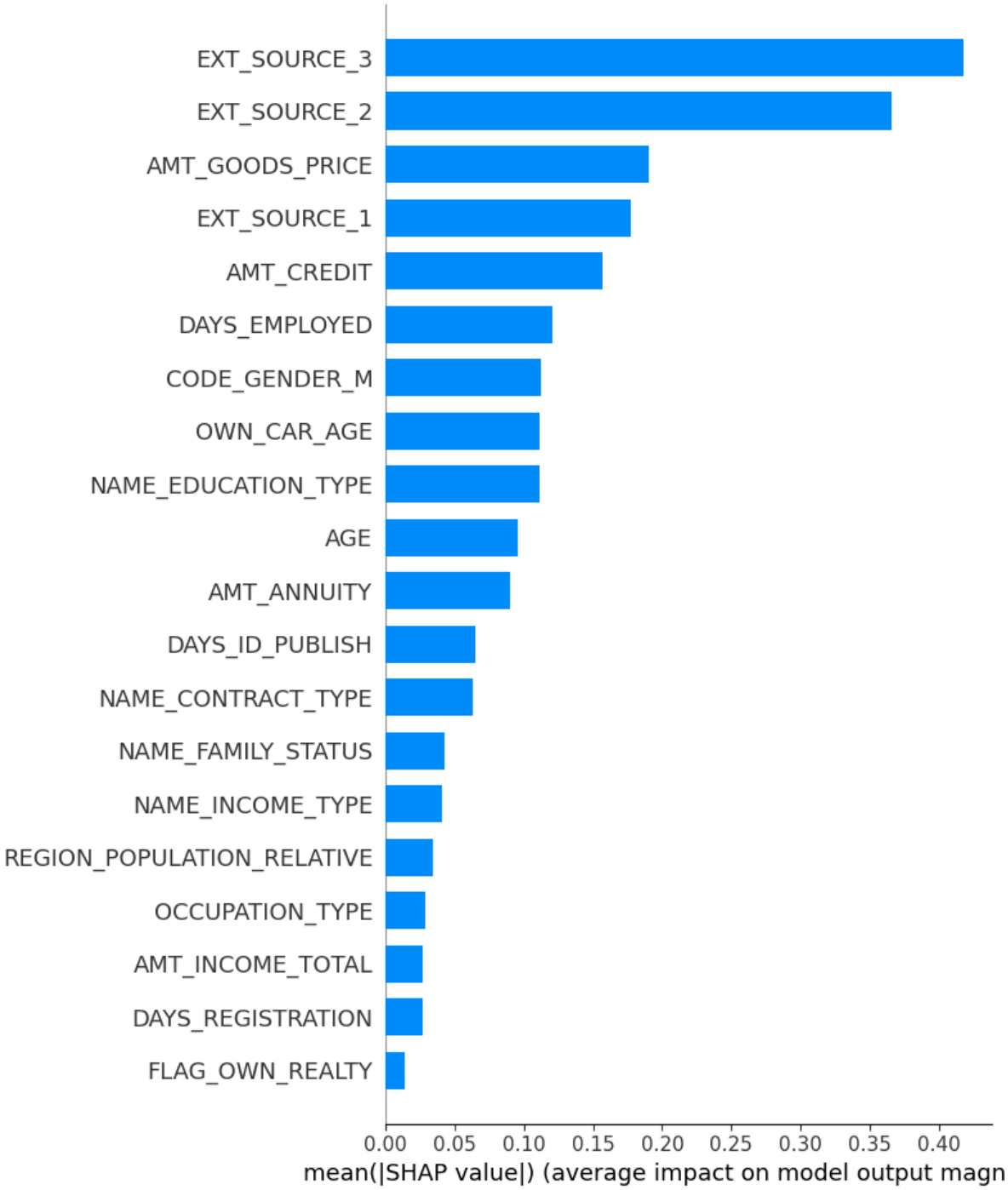


In [ ]:
```
xgb.plot_importance(xgb_base_model, importance_type="gain")
plt.title('xgboost.plot_importance(model, importance_type="gain")')
plt.show()
```



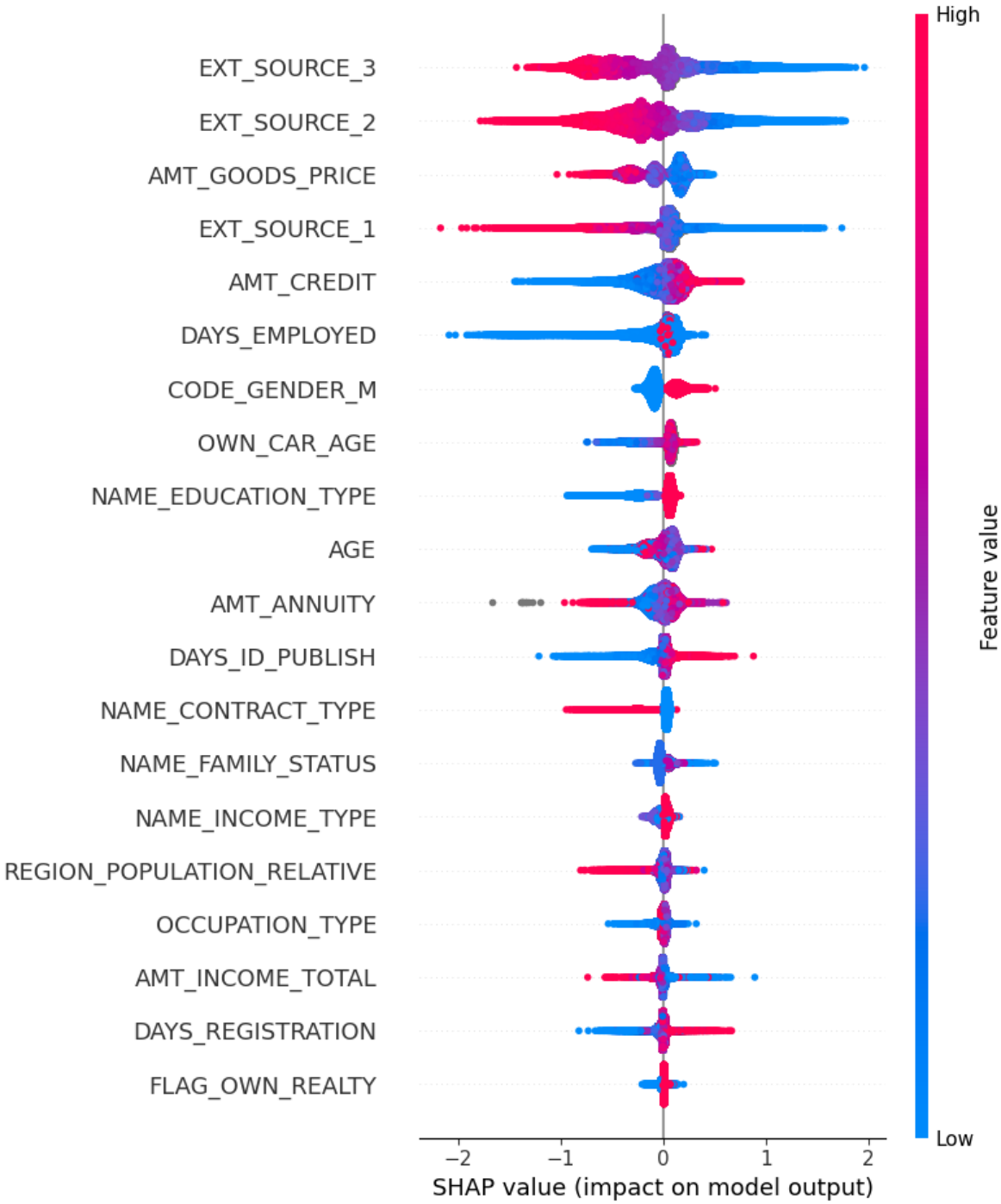
In [ ]:
```
explainer = shap.Explainer(xgb_base_model)
shap_values = explainer.shap_values(df_train.drop(columns='SK_ID_CURR'))
```

In [ ]:
```
shap.summary_plot(shap_values, ap_train.drop(columns='SK_ID_CURR'), plot_type="bar"
```



In [ ]:
```
shap.summary_plot(shap_values, df_train.drop(columns='SK_ID_CURR'))
```

```
No data for colormapping provided via 'c'. Parameters 'vmin', 'vmax' will be ignor
ed
```



Feature importance: Variables are ranked in descending order. Impact: The horizontal location shows whether the effect of that value is associated with a higher or lower prediction. Original value: Color shows whether that variable is high (in red) or low (in blue) for that observation. Correlation: A high level of "EXT_SOURCE" has a high and negative impact on the target. The “high” comes from the red color, and the "negative” impact is shown on the X-axis.

[#4] Comparing the model predictions with respect to the original training data

In [ ]:
```
xgb_base_test_results.shape
```

Out[ ]:
```
(61503,)
```

In [ ]:
```
df_test.drop(columns='SK_ID_CURR').shape
```

Out[ ]:
```
(61503, 24)
```

In [ ]:
```
df_test.drop(columns='SK_ID_CURR').columns
```

Out[ ]:
```
Index(['NAME_CONTRACT_TYPE', '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_EMPLOYED', 'DAYS_REGISTRATION', 'DAYS_ID_PUBLISH', 'OWN_CAR_AGE',
       'OCCUPATION_TYPE', 'EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3',
       'AGE', 'CODE_GENDER_M'],
      dtype='object')
```

In [ ]:
```
int_cols_cat
```

Out[ ]:
```
['CODE_GENDER',
 'NAME_INCOME_TYPE',
 'NAME_EDUCATION_TYPE',
 'NAME_FAMILY_STATUS',
 'NAME_HOUSING_TYPE']
```

In [ ]:
```
int_cols_cat_test = []
for col in int_cols_cat:
    check = col
    res = [i for i in df_test.drop(columns='SK_ID_CURR').columns if check in i]
    int_cols_cat_test += res
```

In [ ]:
```
df_test_binned=cap_data(df_test.drop(columns='SK_ID_CURR'), int_cols_con)
```

```
capping the  CNT_CHILDREN
capping the  AMT_INCOME_TOTAL
capping the  AGE
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

In [ ]:
```
for col in int_cols_con:
    df_test_binned[col] = pd.cut(df_test_binned[col], 6, precision=0)
```

In [ ]:
```
int_cols_test = int_cols_con + int_cols_cat_test
```

In [ ]:
```
int_cols_test
```

Out[ ]:
```
['CNT_CHILDREN',
 'AMT_INCOME_TOTAL',
 'AGE',
 'CODE_GENDER_M',
 'NAME_INCOME_TYPE',
 'NAME_EDUCATION_TYPE',
 'NAME_FAMILY_STATUS',
 'NAME_HOUSING_TYPE']
```

In [ ]:
```
from sklearn.metrics import accuracy_score
predictions = [round(value) for value in xgb_base_test_results]
accuracy = accuracy_score(df_target_test, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100))
```

```
Accuracy: 74.70%
```

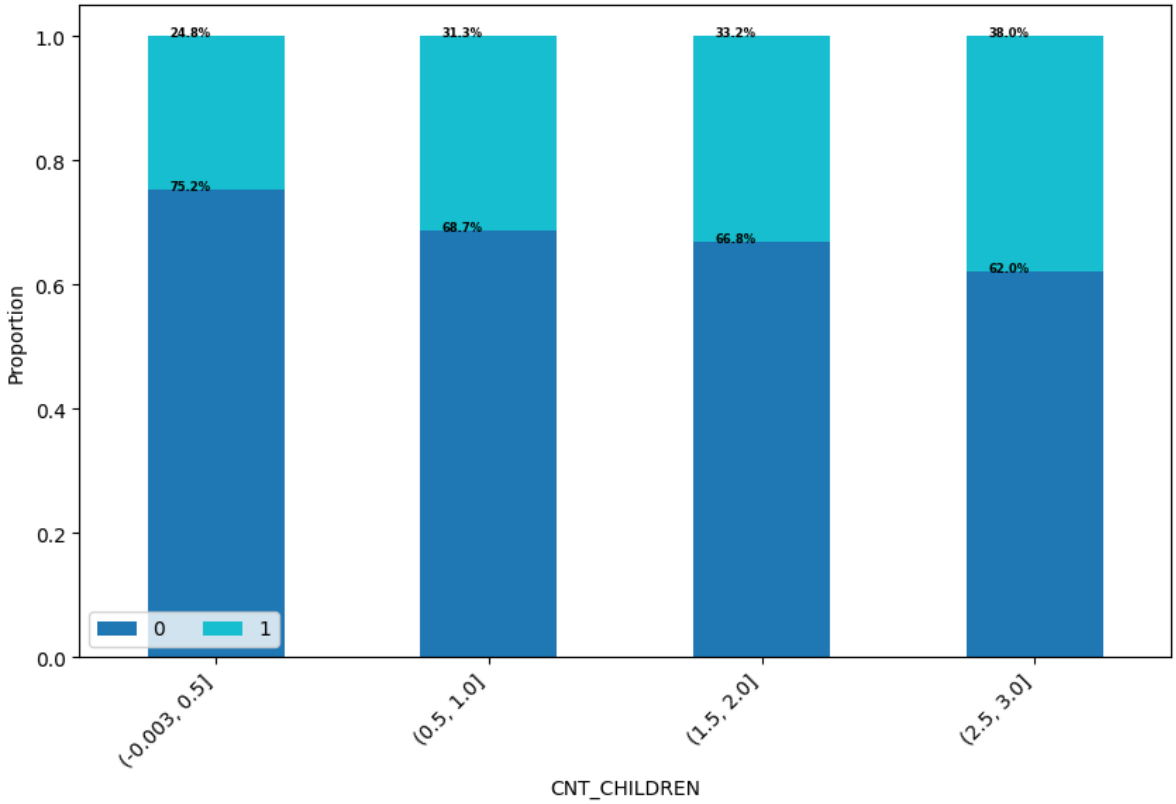
In [ ]:
```
df_test_binned["TARGET"] = predictions
```

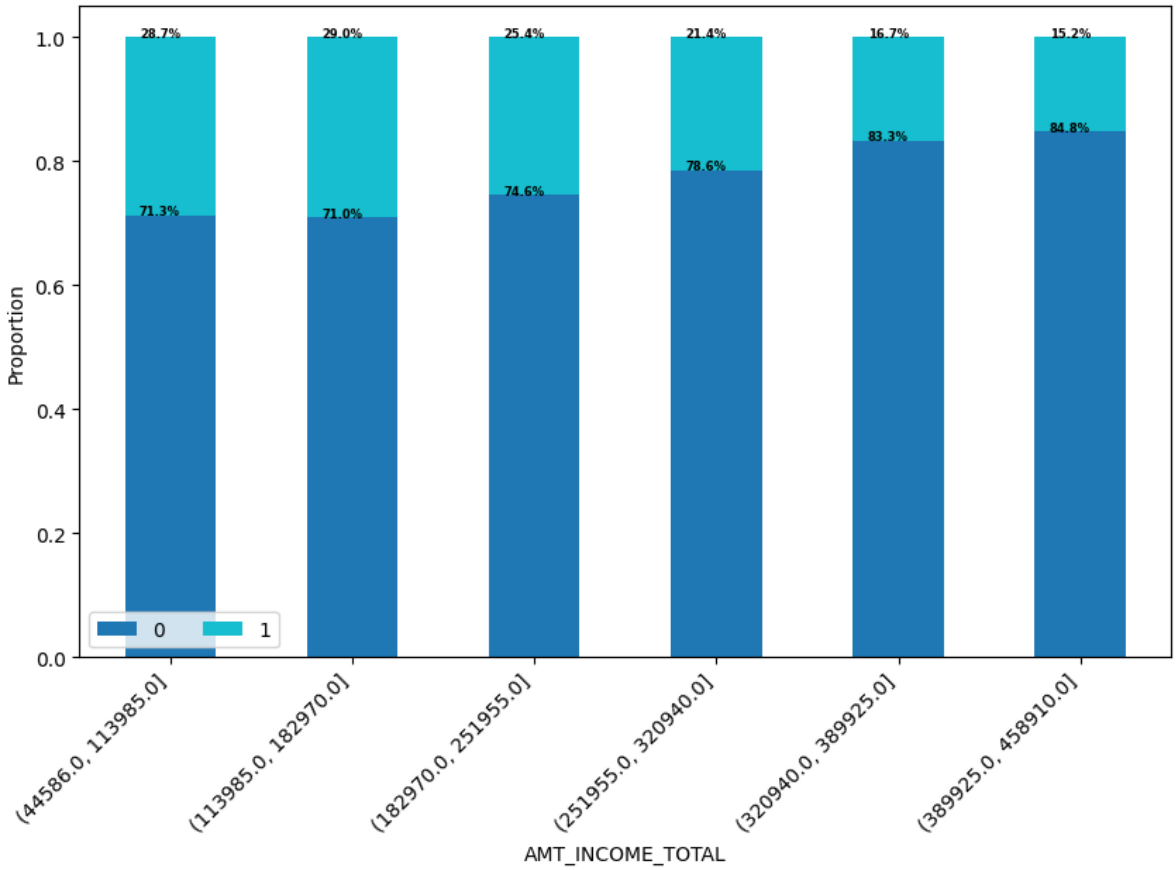
In [ ]:
```
df_test_binned.shape
```

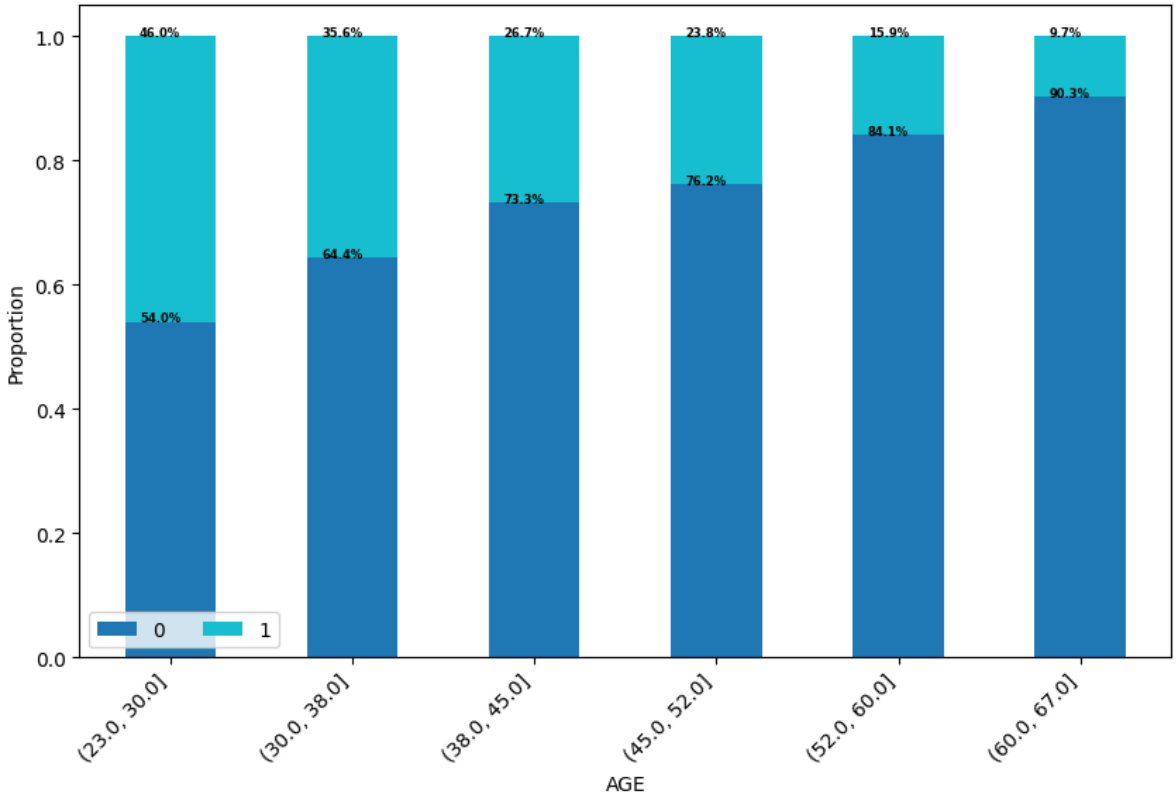
Out[ ]:
```
(61503, 25)
```

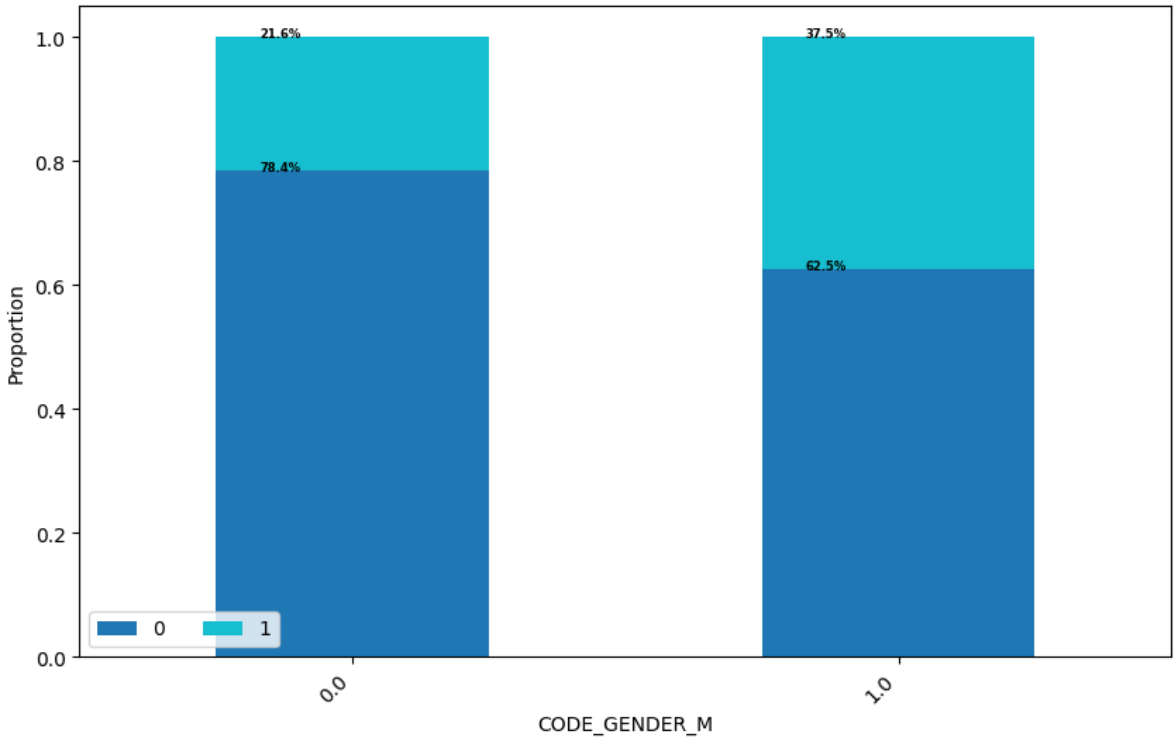
In [ ]:
```

```

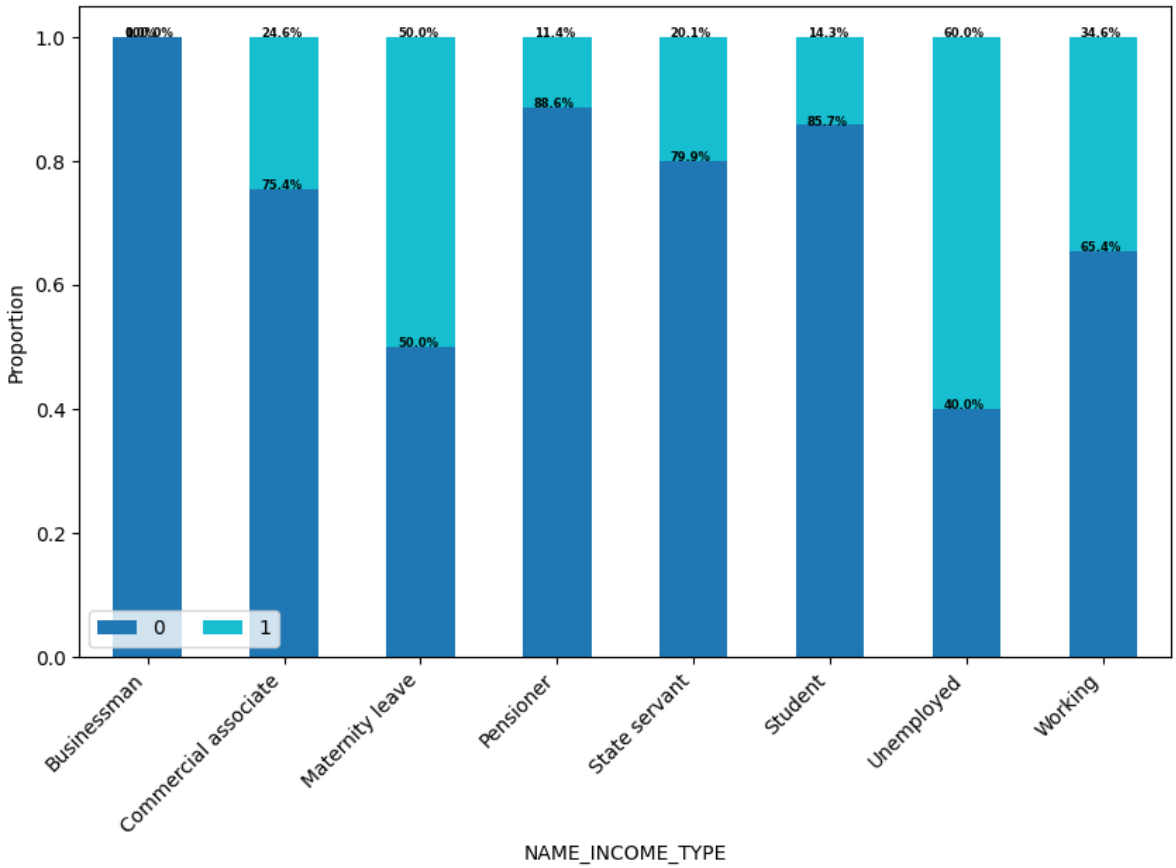
In [ ]:
```
for col in int_cols_test:
    cross_tab_prop = pd.crosstab(index=df_test_binned[col],
                                 columns=df_test_binned["TARGET"],
                                     normalize="index")
    cross_tab_prop

    cross_tab = pd.crosstab(index=df_test_binned[col],
                                 columns=df_test_binned["TARGET"])
    cross_tab

    cross_tab_prop.plot(kind='bar',
                        stacked=True,
                        colormap='tab10',
                        figsize=(10, 6))

    plt.legend(loc="lower left", ncol=2)
    plt.xlabel(str(col))
    plt.ylabel("Proportion")
    plt.xticks(rotation=45, ha='right')


    for n, x in enumerate([*cross_tab.index.values]):
        for (proportion, y_loc) in zip(cross_tab_prop.loc[x],
                                       cross_tab_prop.loc[x].cumsum()):

            plt.text(x=n - 0.17,
                    y=y_loc,
                    s=f'{np.round(proportion * 100, 1)}%',
                    color="black",
                    fontsize=6,
                    fontweight="bold")

    plt.savefig("PREDICTED_" + str(col), bbox_inches = "tight")
    plt.show()
```

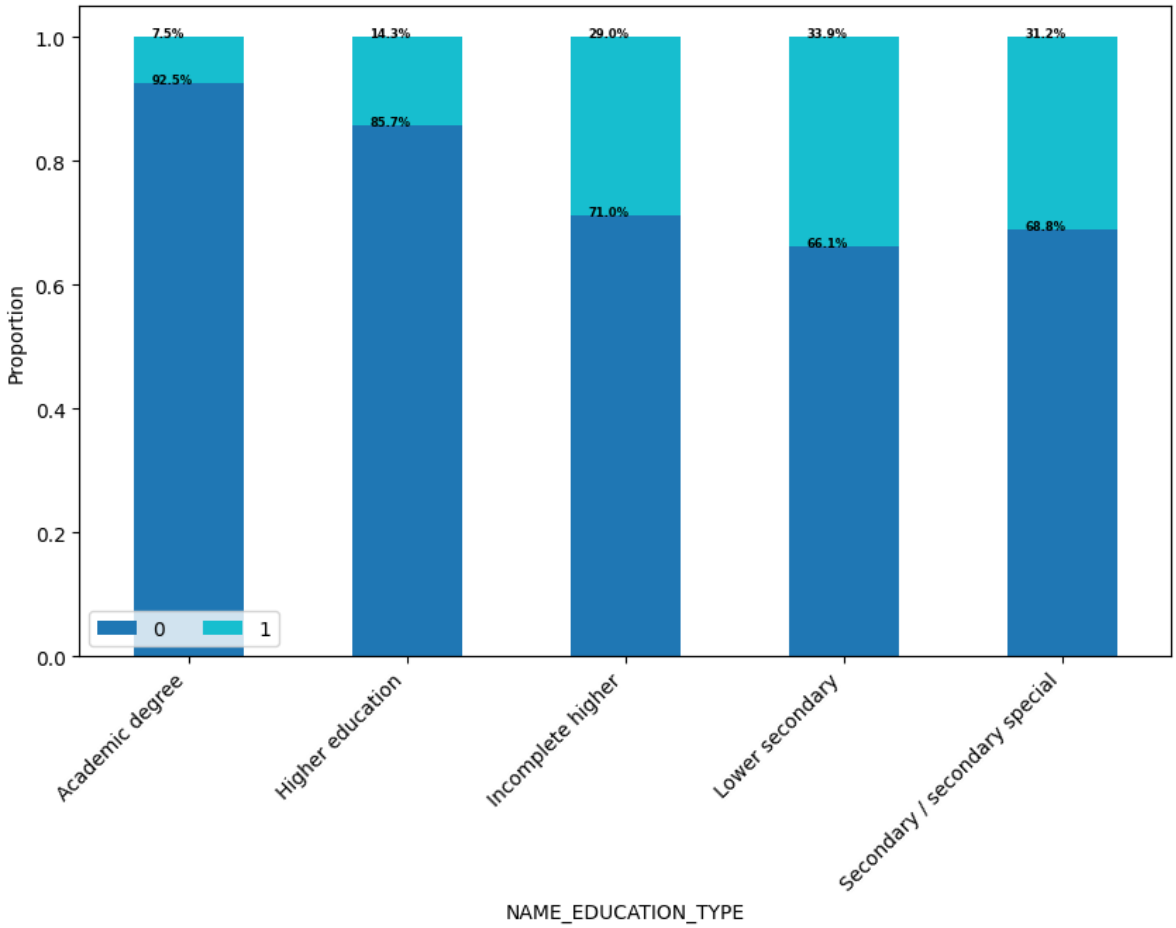

24.8%
31.3%
33.2%
38.0%
75.2%
68.7%
66.8%
62.0%
1.0
0.8
0.6
0.4
0.2
0.0
Proportion
0
1
(-0.003, 0.5]
(0.5, 1.0]
(1.5, 2.0]
(2.5, 3.0]
CNT_CHILDREN

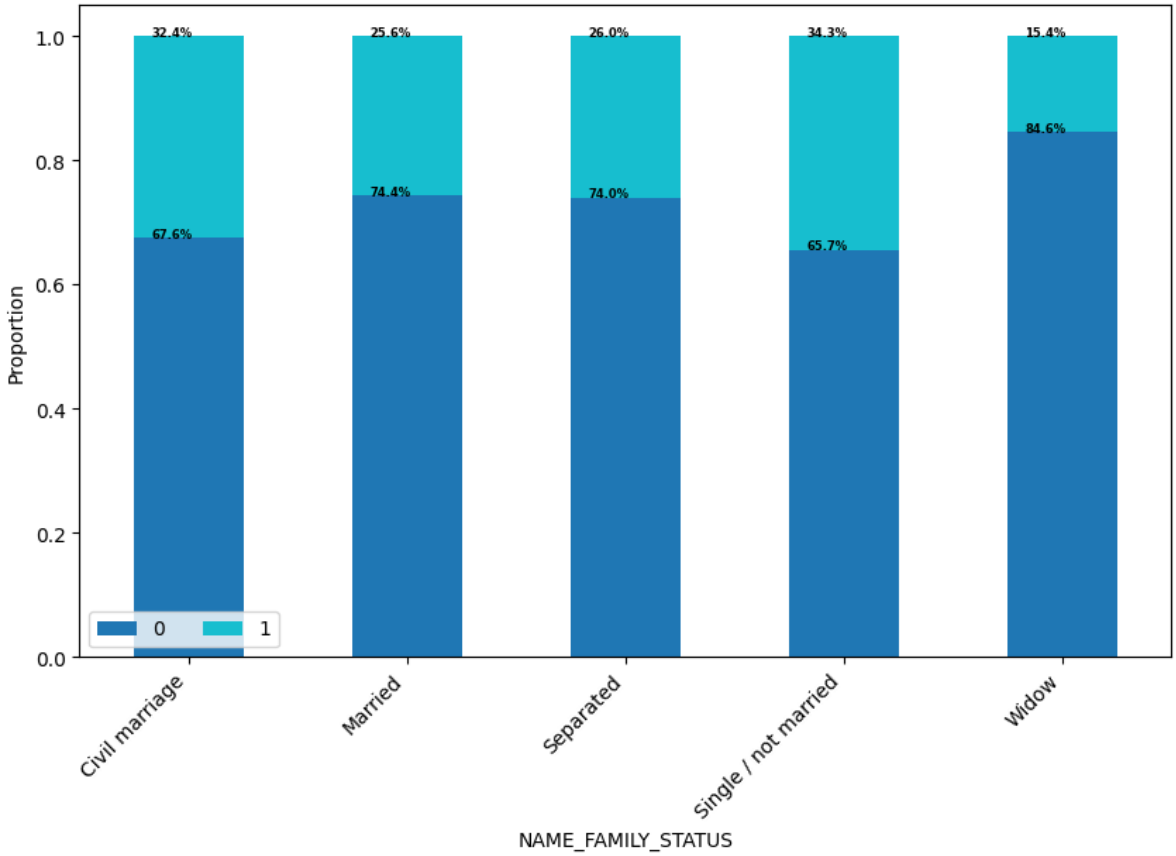

28.7%
29.0%
25.4%
21.4%
16.7%
15.2%
71.3%
71.0%
74.6%
78.6%
83.3%
84.8%
1.0
0.8
0.6
0.4
0.2
0.0
Proportion
0
1
(44586.0, 113985.0]
(113985.0, 182970.0]
(182970.0, 251955.0]
(251955.0, 320940.0]
(320940.0, 389925.0]
(389925.0, 458910.0]
AMT_INCOME_TOTAL

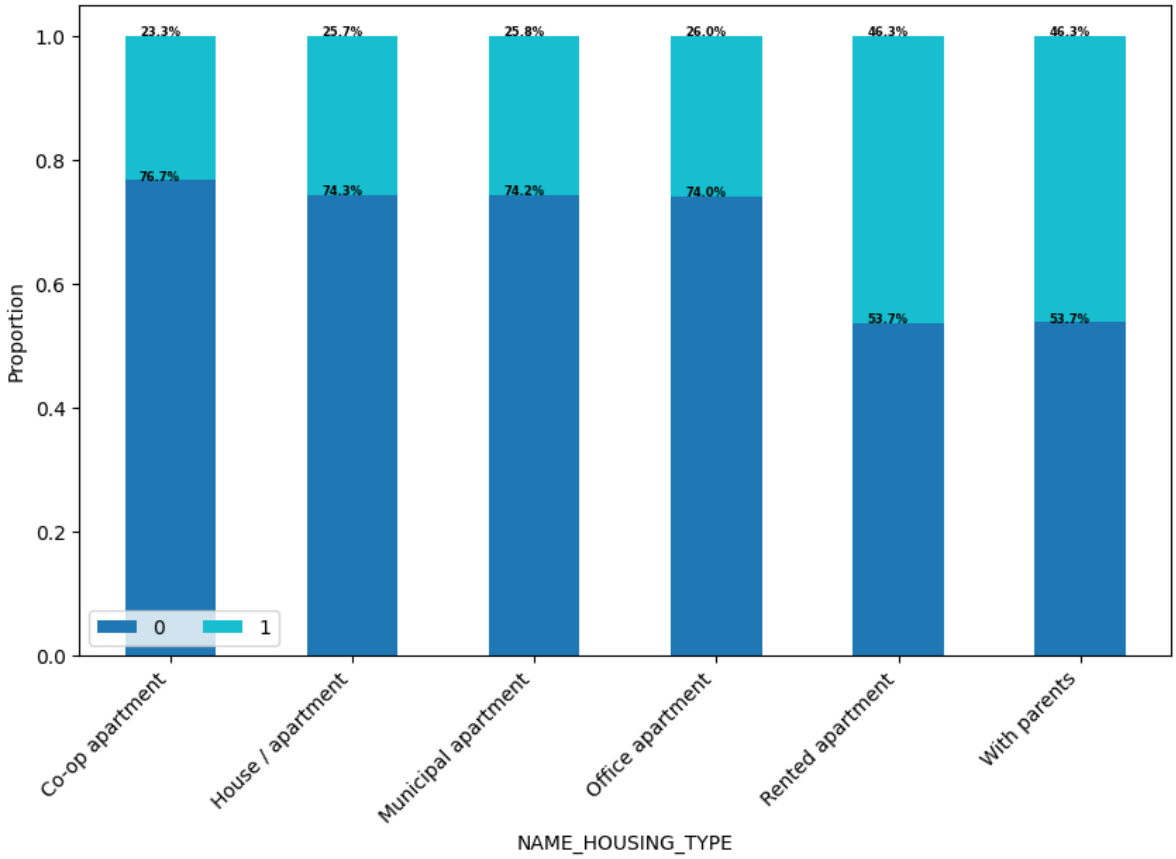

46.0%
35.6%
26.7%
23.8%
15.9%
9.7%
54.0%
64.4%
73.3%
76.2%
84.1%
90.3%
Proportion
0
1
(23.0, 30.0]
(30.0, 38.0]
(38.0, 45.0]
(45.0, 52.0]
(52.0, 60.0]
(60.0, 67.0]
AGE


21.6%
37.5%
78.4%
62.5%
Proportion
0
1
0.0
1.0
CODE_GENDER_M


Proportion
1.0
0.8
0.6
0.4
0.2
0.0
24.6%
50.0%
11.4%
20.1%
14.3%
60.0%
34.6%
75.4%
50.0%
88.6%
79.9%
85.7%
40.0%
65.4%
0
1
Businessman
Commercial associate
Maternity leave
Pensioner
State servant
Student
Unemployed
Working
NAME_INCOME_TYPE


Proportion
1.0
0.8
0.6
0.4
0.2
0.0
7.5%
14.3%
29.0%
33.9%
31.2%
92.5%
85.7%
71.0%
66.1%
68.8%
0
1
Academic degree
Higher education
Incomplete higher
Lower secondary
Secondary / secondary special
NAME_EDUCATION_TYPE





[#5] Comparison of a few cases, changing just 1 or 2 features

In [ ]:
```
sample_size = 100
```

In [ ]:
```
df_test_dmatrix_subset_original = xgb.DMatrix(df_test[:sample_size].drop(columns='S
xgb_base_test_results_subset_original = xgb_base_model.predict(df_test_dmatrix_subs
predictions_subset_original = [round(value) for value in xgb_base_test_results_subs
predictions_subset_original
```

Out[ ]:
```
[0,
 0,
 1,
 0,
 0,
 1,
 0,
 0,
 1,
 0,
 0,
 1,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 1,
 0,
 1,
 0,
 0,
 0,
 1,
 0,
 1,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 1,
 0,
 0,
 0,
 0,
 0,
 0,
```

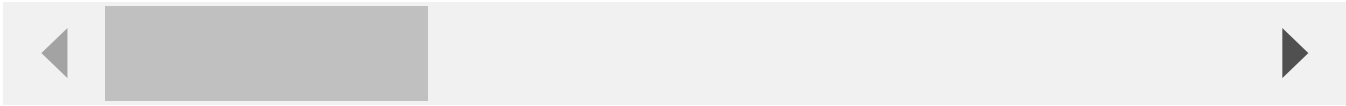
```
0,
0,
1,
0,
1,
1,
0,
0,
0,
1,
0,
0,
1,
0,
1,
1,
0,
1,
1,
1,
1,
0,
1,
1,
1,
0,
0,
0,
0,
0,
0,
1,
0,
0,
1,
1,
0,
1,
1]
```

In [ ]:
```
df_test_subset_altered = df_test[:sample_size].drop(columns='SK_ID_CURR').copy(deep
df_test_subset_altered
```

Out[ ]:

	NAME_CONTRACT_TYPE	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INC
256571	Cash loans	1	1	1	
191493	Cash loans	0	0	0	
103497	Cash loans	0	1	0	
130646	Cash loans	0	0	0	
211898	Cash loans	0	1	0	
...	...	...	...	...	
130397	Cash loans	0	1	0	
157812	Cash loans	0	1	1	
47796	Cash loans	1	1	1	
238921	Cash loans	1	0	1	
206875	Cash loans	0	1	1	

100 rows × 24 columns



In [ ]:
```
# import random
# random.seed(22)
# for i in range(len(df_test_subset_altered)):
#     for col in int_cols_test:
#         rng = random.randint(0, len(df_test_subset_altered) - 1)
#         # print(rng)
#         df_test_subset_altered.at[df_test_subset_altered.index[i], col] = df_test
# df_test_subset_altered
```

In [ ]:
```
df_test_subset_altered.dtypes
```

Out[ ]:
```
NAME_CONTRACT_TYPE           category
FLAG_OWN_CAR                    int32
FLAG_OWN_REALTY                 int32
CNT_CHILDREN                    int64
AMT_INCOME_TOTAL              float64
AMT_CREDIT                    float64
AMT_ANNUITY                   float64
AMT_GOODS_PRICE               float64
NAME_TYPE_SUITE              category
NAME_INCOME_TYPE             category
NAME_EDUCATION_TYPE          category
NAME_FAMILY_STATUS           category
NAME_HOUSING_TYPE            category
REGION_POPULATION_RELATIVE    float64
DAYS_EMPLOYED                   int64
DAYS_REGISTRATION             float64
DAYS_ID_PUBLISH                 int64
OWN_CAR_AGE                   float64
OCCUPATION_TYPE              category
EXT_SOURCE_1                  float64
EXT_SOURCE_2                  float64
EXT_SOURCE_3                  float64
AGE                           float64
CODE_GENDER_M                 float64
dtype: object
```

In [ ]:
```
df_test_subset_altered["NAME_FAMILY_STATUS"].describe
```

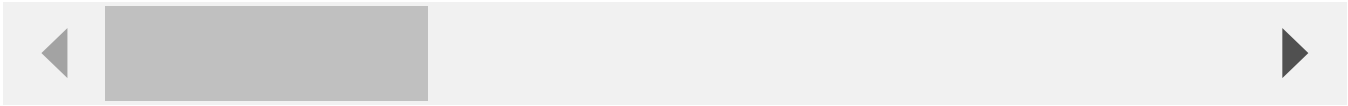
Out[ ]:
```
<bound method NDFrame.describe of 256571               Married
191493                 Married
103497           Civil marriage
130646    Single / not married
211898           Civil marriage
                  ...
130397                 Married
157812                 Married
47796                  Married
238921                 Married
206875                 Married
Name: NAME_FAMILY_STATUS, Length: 100, dtype: category
Categories (6, object): ['Civil marriage', 'Married', 'Separated', 'Single / not m
arried', 'Unknown', 'Widow']>
```

In [ ]:
```
for i in range(len(df_test_subset_altered)):
    df_test_subset_altered.at[df_test_subset_altered.index[i], "NAME_EDUCATION_TYPE
    df_test_subset_altered.at[df_test_subset_altered.index[i], "NAME_FAMILY_STATUS"
    if i % 2 == 0:
        df_test_subset_altered.at[df_test_subset_altered.index[i], "CODE_GENDER_M"]
        df_test_subset_altered.at[df_test_subset_altered.index[i], "NAME_INCOME_TYP
    else:
        df_test_subset_altered.at[df_test_subset_altered.index[i], "CODE_GENDER_M"]
        df_test_subset_altered.at[df_test_subset_altered.index[i], "NAME_INCOME_TYP

df_test_subset_altered
```

Out[ ]:

	NAME_CONTRACT_TYPE	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INC
256571	Cash loans	1	1	1	
191493	Cash loans	0	0	0	
103497	Cash loans	0	1	0	
130646	Cash loans	0	0	0	
211898	Cash loans	0	1	0	
...	...	...	...	...	
130397	Cash loans	0	1	0	
157812	Cash loans	0	1	1	
47796	Cash loans	1	1	1	
238921	Cash loans	1	0	1	
206875	Cash loans	0	1	1	

100 rows × 24 columns



In [ ]:

```
df_test_dmatrix_subset_altered = xgb.DMatrix(df_test_subset_altered, enable_categor
xgb_base_test_results_subset_altered = xgb_base_model.predict(df_test_dmatrix_subse
predictions_subset_altered = [round(value) for value in xgb_base_test_results_subse
predictions_subset_altered
```

Out[ ]:
```
[1,
 0,
 1,
 0,
 1,
 1,
 0,
 0,
 1,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 1,
 0,
 1,
 0,
 1,
 0,
 1,
 0,
 1,
 0,
 0,
 0,
 1,
 0,
 1,
 0,
 1,
 0,
 0,
 0,
 1,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 1,
 0,
 1,
 0,
 0,
 0,
 0,
 0,
 1,
```

```
0,
0,
1,
0,
1,
1,
0,
0,
0,
1,
0,
1,
1,
0,
1,
1,
0,
1,
1,
1,
1,
1,
1,
1,
1,
1,
0,
0,
0,
0,
0,
0,
1,
0,
0,
1,
1,
0,
1,
1]
```

In [ ]:
```
print(predictions_subset_original, predictions_subset_altered, sep = "\n")
```

```
[0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1,
1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1]
[1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1,
0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1,
1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1]
```

In [ ]:
```
# diff = 0
# for i in range(len(predictions_subset_altered)):
#     diff += abs(int(predictions_subset_altered[i]) - int(predictions_subset_origi


# print("Difference: %.2f%%" % (diff / sample_size * 100))
```

In [ ]:
```
diff = sum(predictions_subset_altered) - sum(predictions_subset_original)

print("Difference: %.2f%%" % (diff / sample_size * 100))
```

```
Difference: 9.00%
```

[#6] Wrap up: characteristics of the training dataset and how a ML model learns them

[#7] Dropping all our concerns

In [ ]:
```
ap_train_unbiased.drop(columns=int_cols, inplace = True)
```

In [ ]:
```
ap_train_unbiased['FLAG_OWN_CAR'] = np.where(ap_train_unbiased['FLAG_OWN_CAR'] == '
ap_train_unbiased['FLAG_OWN_REALTY'] = np.where(ap_train_unbiased['FLAG_OWN_REALTY'
```

In [ ]:
```
ap_objects_unbiased = list(ap_train_unbiased.select_dtypes(include=['object']).colu
ap_train_unbiased[ap_objects_unbiased] = ap_train_unbiased[ap_objects_unbiased].ast
```

Let's separate the target from the rest of the data

In [ ]:
```
ap_train_target_unbiased = ap_train_unbiased.pop('TARGET')
print(f"Target unbiased dataset shape: {ap_train_target_unbiased.shape}")
```

```
Target unbiased dataset shape: (307511,)
```

Let's split the original dataset in two:

- 80% for the train dataset
- 20% for the test one

In [ ]:
```
df_train_unbiased, df_test_unbiased, df_target_train_unbiased, df_target_test_unbia
    ap_train_unbiased, ap_train_target_unbiased, test_size=0.2, stratify=ap_train_t

print(f"Train unbiased dataset shape: {df_train_unbiased.shape}")
print(f"Test unbiased dataset shape: {df_test_unbiased.shape}")
```

```
Train unbiased dataset shape: (246008, 17)
Test unbiased dataset shape: (61503, 17)
```

In [ ]:
```
df_train_unbiased.dtypes
```

Out[ ]:
```
SK_ID_CURR                    int64
NAME_CONTRACT_TYPE         category
FLAG_OWN_CAR                  int32
FLAG_OWN_REALTY               int32
AMT_CREDIT                  float64
AMT_ANNUITY                 float64
AMT_GOODS_PRICE             float64
NAME_TYPE_SUITE            category
REGION_POPULATION_RELATIVE  float64
DAYS_EMPLOYED                 int64
DAYS_REGISTRATION           float64
DAYS_ID_PUBLISH               int64
OWN_CAR_AGE                 float64
OCCUPATION_TYPE            category
EXT_SOURCE_1                float64
EXT_SOURCE_2                float64
EXT_SOURCE_3                float64
dtype: object
```

In [ ]:
```
cat_cols = []
for col in df_train_unbiased.columns:
    if df_train_unbiased[col].dtype == "category":
        cat_cols.append(col)
cat_cols
```

Out[ ]:
```
['NAME_CONTRACT_TYPE', 'NAME_TYPE_SUITE', 'OCCUPATION_TYPE']
```

In [ ]:
```
from sklearn import preprocessing

lbl = preprocessing.LabelEncoder()

for col in cat_cols:
    df_train_unbiased[col] = lbl.fit_transform(df_train_unbiased[col].astype(str))
```

Create a basic ML model and scoring on the test set

I'll train a simple XGBoost model (parameters previously chosen with cross-validation).

In [ ]:
```
df_train_dmatrix_unbiased = xgb.DMatrix(df_train_unbiased.drop(columns='SK_ID_CURR'

params = {'max_depth':6,
          'eta':.2,
          'subsample':.9,
          'colsample_bytree':.9,
          'scale_pos_weight':10,
          'objective':'binary:logistic',
          'tree_method':'exact'}
xgb_base_model_unbiased = xgb.train(params, df_train_dmatrix_unbiased, num_boost_rc
```

Let's score the test set.

In [ ]:
```
df_test_dmatrix_unbiased = xgb.DMatrix(df_test_unbiased.drop(columns='SK_ID_CURR'),
xgb_base_test_results_unbiased = xgb_base_model_unbiased.predict(df_test_dmatrix_ur
```

What are the risk scores (from 0 to 1) of the first 5 customers in the test set? And what's the overall AUC on the test set?

In [ ]:
```
xgb_base_test_results_unbiased[:5]
```

Out[ ]:
```
array([0.41405448, 0.2634759 , 0.7225189 , 0.3854774 , 0.49003527],
      dtype=float32)
```

In [ ]:
```
fpr, tpr, thresholds = metrics.roc_curve(df_target_test_unbiased, xgb_base_test_res
metrics.auc(fpr, tpr)
```

Out[ ]:
```
0.7373679964356246
```

The third customer has a higher risk score (0.722) than the others in the first 5 records.

This simple model has an AUC on the test set around **0.737**, a decent baseline performance.

[#8] A closer look at the performance without features of ethical concern

[#9] Wrapping up again: what happens by just removing the features of concern

[#10] Hints at a possible (but questionable!) solution