

# application\_train\_test\_quick\_eda

September 15, 2019

```
[107]: # Load libraries

%matplotlib inline
import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

from scipy import stats

[2]: def quick_overview_df( dataset_df ):
    display( 'shape:', dataset_df.shape )
    display( 'sample:', dataset_df.sample() )
    display( 'NaN values:', dataset_df.isnull().sum() )
    display( 'duplicates:', dataset_df.duplicated().sum() )
    dataset_df.info()

[68]: # Overview 'Application' table (train and test)

APPLICATION_TRAIN_FILEPATH = 'data/application_train.csv'
application_train = pd.read_csv( APPLICATION_TRAIN_FILEPATH, header=0 )

APPLICATION_TEST_FILEPATH = 'data/application_test.csv'
application_test = pd.read_csv( APPLICATION_TEST_FILEPATH, header=0 )

[4]: # quick_overview_df( application_train )

[5]: # quick_overview_df( application_test )

[53]: def display_freq_plot( dataset_df, col_name, ax, \
                             title_add='', sort_class=False, **kwargs):
    data_to_plot = dataset_df[col_name].value_counts()
    if sort_class:
        data_to_plot = data_to_plot.sort_index()
    ax.bar(
        data_to_plot.index.values, data_to_plot.values, **kwargs
    )
    col_name_ncount = len( dataset_df[col_name] )
```

```

for patch in ax.patches:
    x = patch.get_x()
    y = patch.get_height()
    y_pct = y * 100.0 / col_name_ncount
    ax.annotate(
        '{0}, {1:.2f}%'.format( y, y_pct ),
        (x + 0.4, y),
        ha='center', va='bottom'
    )
for tick in ax.get_xticklabels():
    tick.set_rotation(90)
ax.set_xticks( data_to_plot.index.values )
ax.set_title( '{0} Frequency Plot {1}'.format(col_name, title_add) )
ax.set_xlabel( '{0} Classes'.format(col_name) )
ax.set_ylabel( 'Frequency [count]' )

def display_stacked_freq_plot_by_target( target_0_df, target_1_df, col_name, ax):
    target_0_toplot_data = target_0_df[col_name].value_counts()
    target_1_toplot_data = target_1_df[col_name].value_counts()
    ax.bar(
        target_0_toplot_data.index.values, target_0_toplot_data.values,
        label='target=0'
    )
    ax.bar(
        target_1_toplot_data.index.values, target_1_toplot_data.values,
        label='target=1'
    )
    zipped_target0_target1_values = list( zip(target_0_toplot_data,
→target_1_toplot_data) )
    for idx, patch in enumerate(ax.patches):
        if idx >= len(zipped_target0_target1_values):
            idx -= len(zipped_target0_target1_values)
        x = patch.get_x()
        y = patch.get_height()
        y_pct = y * 100.0 / sum(zipped_target0_target1_values[idx])
        ax.annotate(
            '{0:.2f}%'.format( y_pct ),
            (x + 0.4, y + 50),
            ha='center', va='bottom'
        )
    for tick in ax.get_xticklabels():
        tick.set_rotation(90)
    ax.set_title( '' )
    ax.legend()

```

```

def display_freq_plot_by_target( target_0_df, target_1_df, col_name ):
    fig, [ax_0, ax_1, ax_2] = plt.subplots( 1, 3, figsize=(17, 5) )
    # target={0,1} - stacked bar chart
    display_stacked_freq_plot_by_target( target_0_df, target_1_df, col_name,
    →ax=ax_0 )
    ax_0.set_title('{0} (TARGET=[0;1])'.format(col_name))
    # target=0
    display_freq_plot( target_0_df, col_name, ax=ax_1 )
    ax_1.set_title('{0} (TARGET=0)'.format(col_name))
    # target=1
    display_freq_plot( target_1_df, col_name, ax=ax_2 )
    ax_2.set_title('{0} (TARGET=1)'.format(col_name))

def display_freq_plot_traintest( train_df, test_df, col_name ):
    fig, [ax_0, ax_1] = plt.subplots( 1, 2, figsize=(15, 5) )
    display_freq_plot( train_df, col_name, ax=ax_0, title_add='TRAIN' )
    display_freq_plot( test_df, col_name, ax=ax_1, title_add='TEST' )

```

```

[69]: # Type 1 overview: basic frequency overview

fig, ax_0 = plt.subplots()
display_freq_plot( application_train, 'TARGET', ax=ax_0 )
plt.show()

cols_to_overview = [
    'NAME_CONTRACT_TYPE', 'CODE_GENDER', 'NAME_FAMILY_STATUS',
    'FLAG_OWN_CAR', 'FLAG_OWN_REALTY',
    'CNT_CHILDREN', 'CNT_FAM_MEMBERS',
    'NAME_INCOME_TYPE', 'OCCUPATION_TYPE', 'NAME_EDUCATION_TYPE',
    →'NAME_HOUSING_TYPE']

for col_name in cols_to_overview:
    display_freq_plot_traintest( application_train, application_test, col_name)

# Overall: Train and Test have almost the same proportion of people in all
→classes

# TARGET:
# Unbalanced feature: 92%=0, 8%=1.
# Most of borrowers didn't have problems with repaying their loans

# NAME_CONTRACT_TYPE : Identification if loan is cash or revolving
# Cash loans are 10x more popular than revolving loans
# Extremely unbalanced in test: 99%/1% classes proportion in test set.

# CODE_GENDER : Gender of the client

```

```

# women applied twice as often as men

# NAME_FAMILY_STATUS : Family status of the client
# Percentage proportion in train / in test - almost the same
# Overwhelming category - married
# Main category: marries (train - 63%, test - 66%)

# FLAG_OWN_CAR
# Most of borrowers don't own a car (2/3)

# FLAG_OWN_REALTY
# Most of borrowers own a property (2/3)

# CNT_CHILDREN
# Most of borrowers (70% train, 71% test) don't have any children
# 0 / 1 / 2 children = 3.5 / 2 / 1 proportion

# CNT_FAM_MEMBERS
# Most borrowers have 2 family members
# >=5 members - <1.5% of total borrowers
# Might have correlation with cnt_children - 1 and 2 fam members - no children
→ (22%+51% in train, 21%+53% in test )

# NAME_INCOME_TYPE
# 50% of total borrowers - Working
# Almost no unemployed, students or businessmen
# Working / Commercial associate = 2.5 / 1
# Pensioners - almost 20% - interesting class

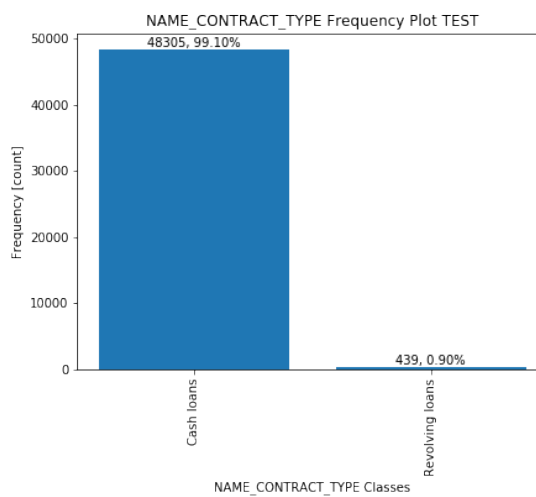
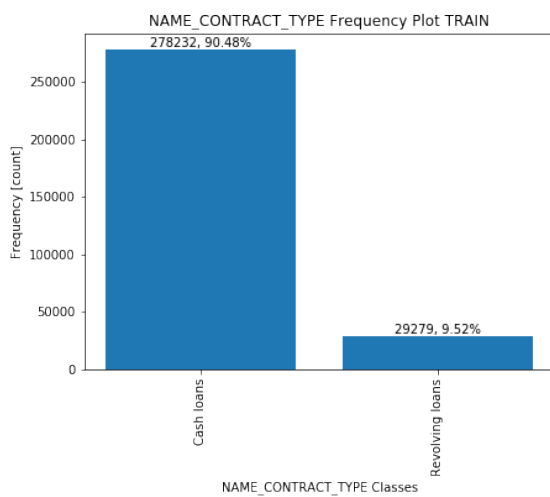
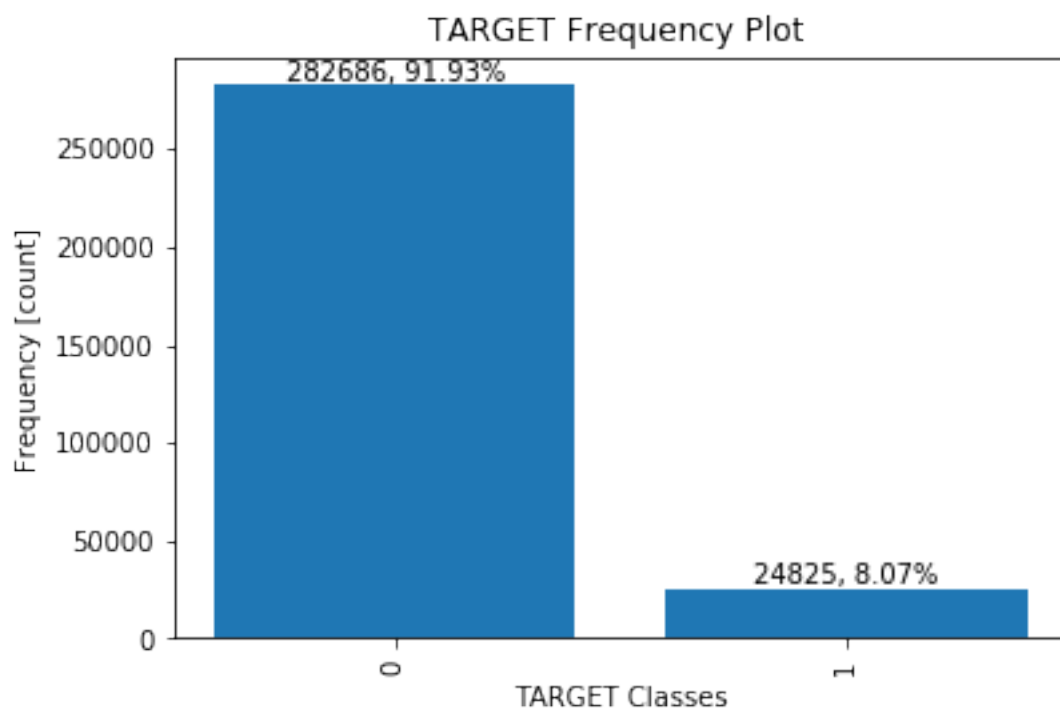
# OCCUPATION_TYPE
# Most of the borrowers are laborers ("others" feature?), work in sales, core
→ staff ("others feature?"), 'managers', 'drivers'
# 'laborers' and 'core staff' might mean any kind of occupation
# IT staff - least popular class of borrowers
# Interesting - 'high skill tech staff' - does high skill means education?

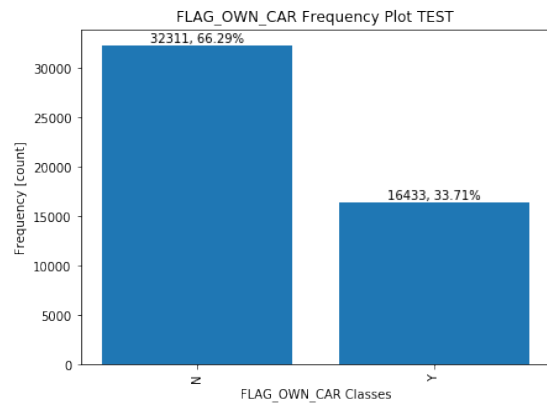
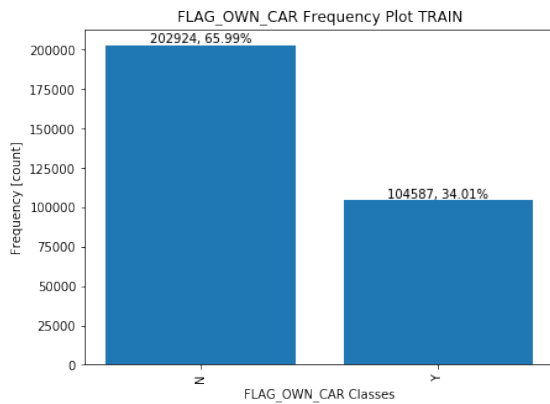
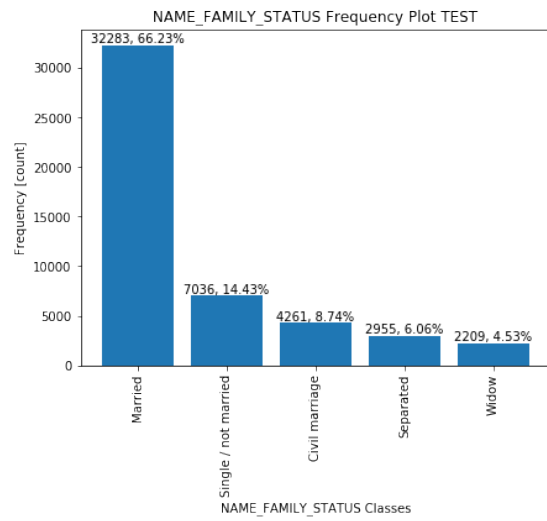
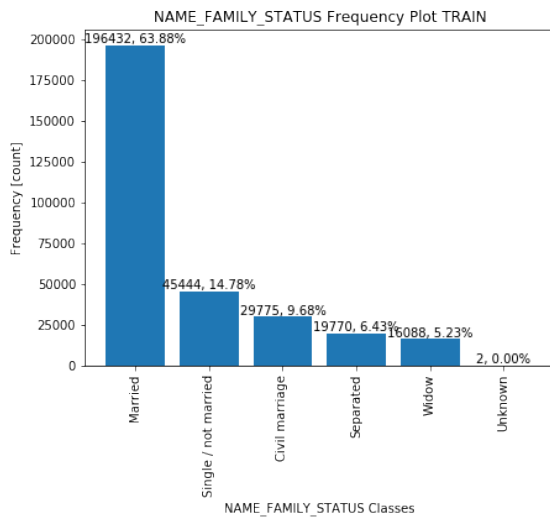
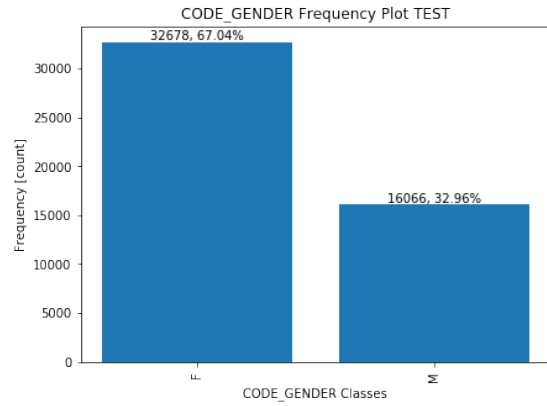
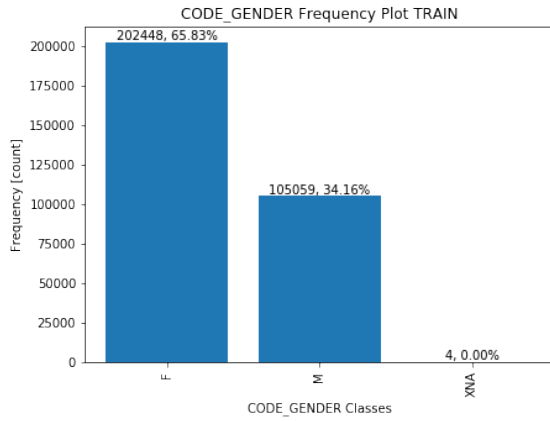
# NAME_EDUCATION_TYPE
# Top class: secondary/secondary special () degree - 70%
# Higher education: 25%
# Others: sum < 5%

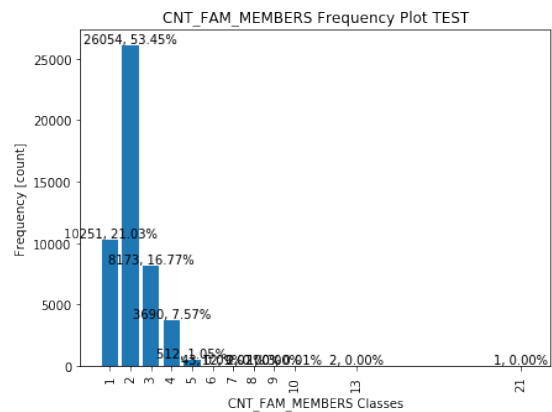
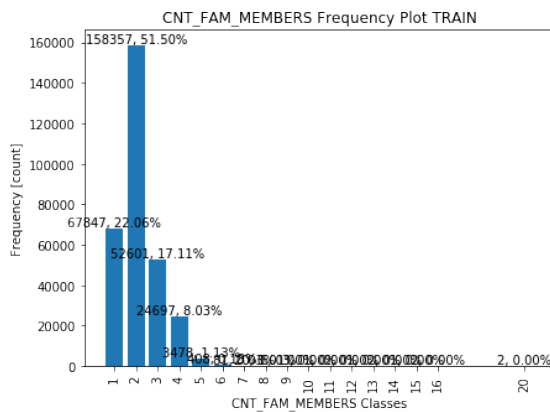
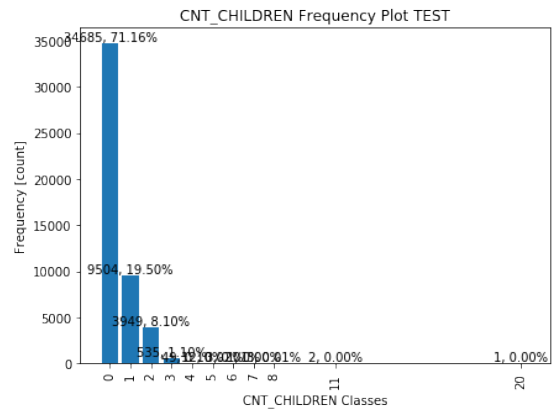
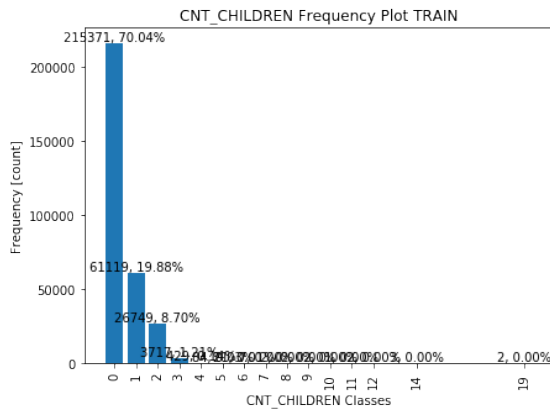
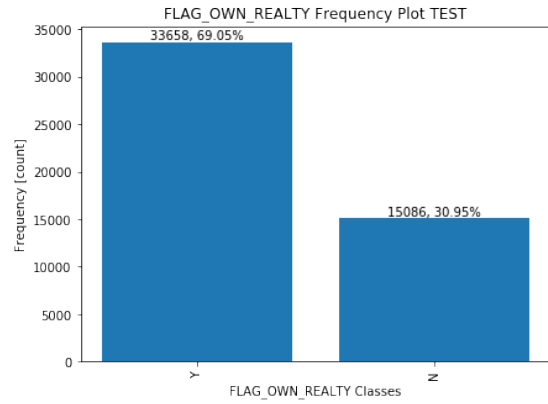
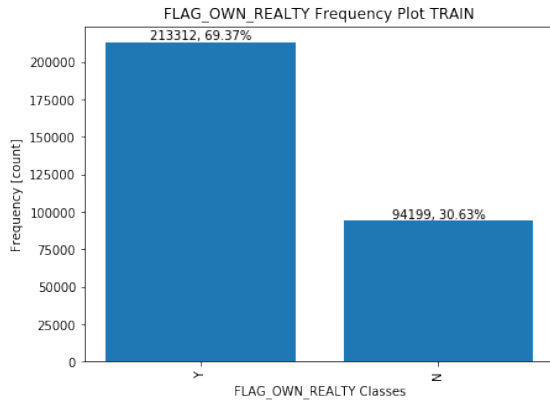
# NAME_HOUSING_TYPE
# This might NOT be a good feature because it represents average state of how
→ people live in Russia
# Top class: home/apartment
# Top class holds 90% of all borrowers

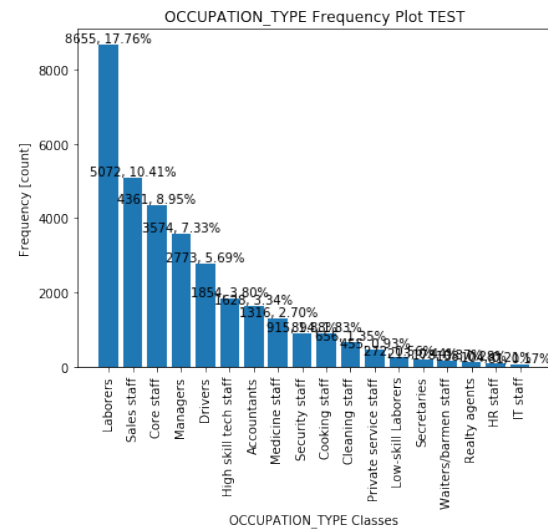
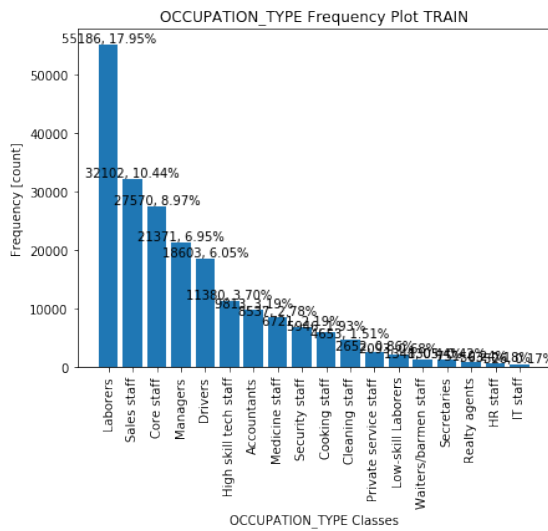
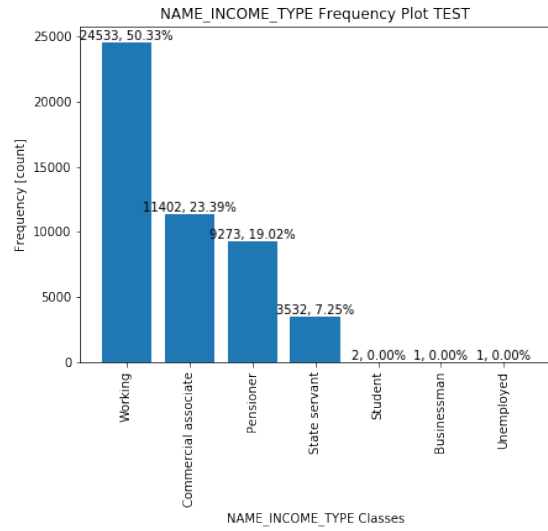
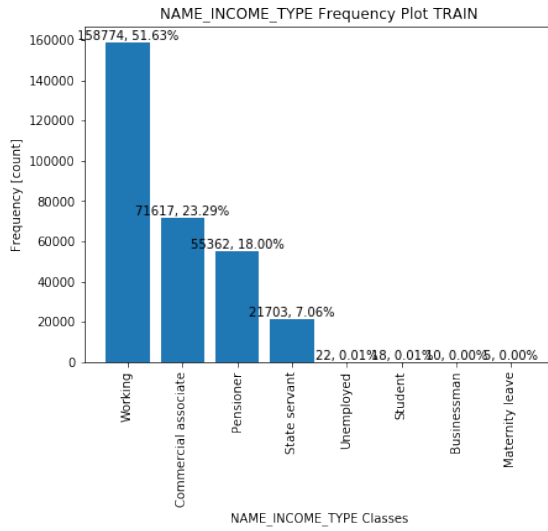
```

#

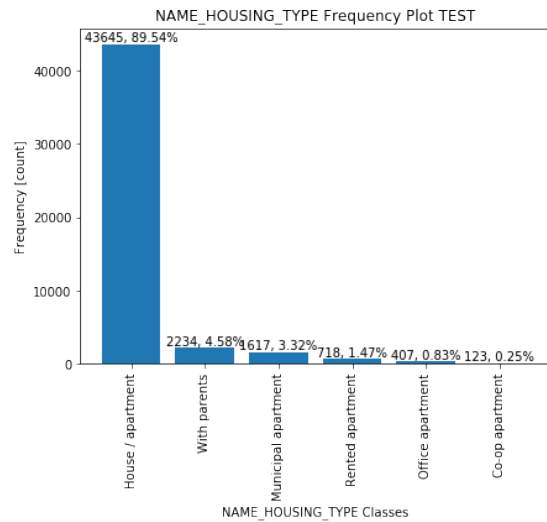
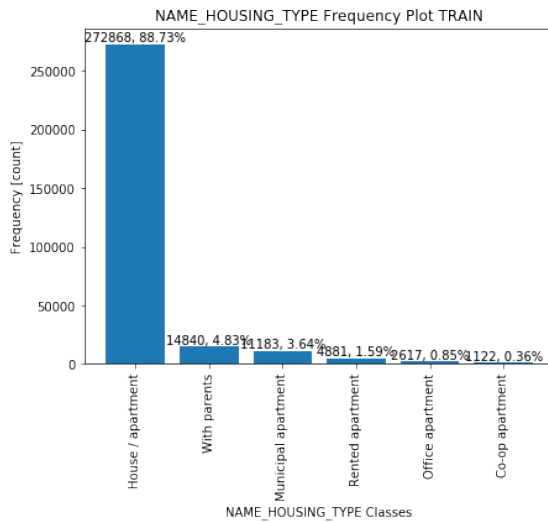
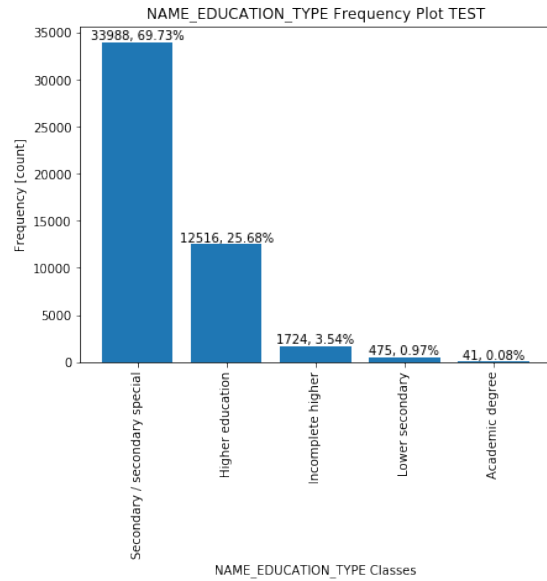
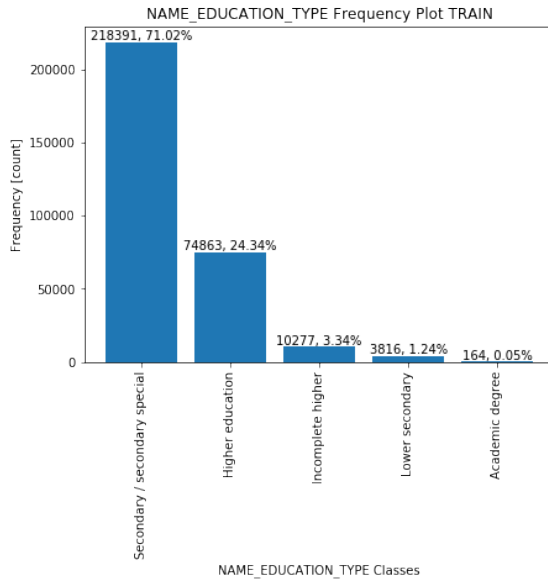












[84]: *# ugly: Manually remove outliers*

```
application_train = application_train[ application_train['NAME_FAMILY_STATUS'] !=
    ↳ 'Unknown' ]
application_train = application_train[ application_train['CODE_GENDER'] !=
    ↳ 'XNA' ]
application_train = application_train[ application_train['CNT_CHILDREN'] < 4 ]
application_train = application_train[ application_train['NAME_INCOME_TYPE'] !=
    ↳ 'Businessman' ]
application_train = application_train[ application_train['NAME_INCOME_TYPE'] !=
    ↳ 'Student' ]
```

```
application_train = application_train[ application_train['NAME_INCOME_TYPE'] != 'Unemployed' ]
```

```
[82]: application_train['NAME_INCOME_TYPE'].value_counts()
```

```
[82]: Working                158390
      Commercial associate    71514
      Pensioner              55341
      State servant          21650
      Unemployed              22
      Student                 18
      Businessman             10
      Name: NAME_INCOME_TYPE, dtype: int64
```

```
[83]: application_test['NAME_INCOME_TYPE'].value_counts()
```

```
[83]: Working                24533
      Commercial associate    11402
      Pensioner              9273
      State servant          3532
      Student                 2
      Businessman             1
      Unemployed              1
      Name: NAME_INCOME_TYPE, dtype: int64
```

```
[85]: # target=0 and target=1 DFs
```

```
train_target_0_rows = application_train[application_train['TARGET'] == 0]
train_target_1_rows = application_train[application_train['TARGET'] == 1]
```

```
[86]: # Type 2 overview: freq plot by target={0, 1}
```

```
cols_to_overview = [
    'NAME_CONTRACT_TYPE', 'CODE_GENDER', 'NAME_FAMILY_STATUS',
    'FLAG_OWN_CAR', 'FLAG_OWN_REALTY',
    'CNT_CHILDREN', 'CNT_FAM_MEMBERS',
    'NAME_INCOME_TYPE', 'OCCUPATION_TYPE', 'NAME_EDUCATION_TYPE',
    'NAME_HOUSING_TYPE']

for col_name in cols_to_overview:
    display_freq_plot_by_target( train_target_0_rows, train_target_1_rows,
    col_name )

# NAME_CONTRACT_TYPE
# Cash loans are slightly more prone to become problematic (late payment)
# target=1 borrowers distributed almost equally between 2 classes: 8% in cash
# loans, 5% in revolving loans.

# CODE_GENDER
```

```

# Even though females borrow more often, MEN make more late payments (f:7%, m:
→10%)

# NAME_FAMILY_STATUS
# All classes have almost equal percentage of bad borrowers: min=5.82%, max=9.
→94%, mean=8.264%
# Top bad borrower class: civil marriage (9.94%), single/not married (9.81%)
# Top bad borrower classes are not the major borrowers: only 9.49% and 14.5%
# Top class by count - Married - are the second best class - only 7.56%
# Best class - widows; onlt 5.48 didn't repay their loans

# FLAG_OWN_CAR
# Equal distribution: those who don't have a car have slightly more chances (1.
→25%) to not repay back

# FLAG_OWN_REALTY
# Equal distribution: borrowers without realty have slightly more chances (0.
→4%) to not repay back

# CNT_CHILDREN
# 0, 1, 2 children - almost the same chance to NOT repay back
# 3 children - 3% - 1/3 of prev prob

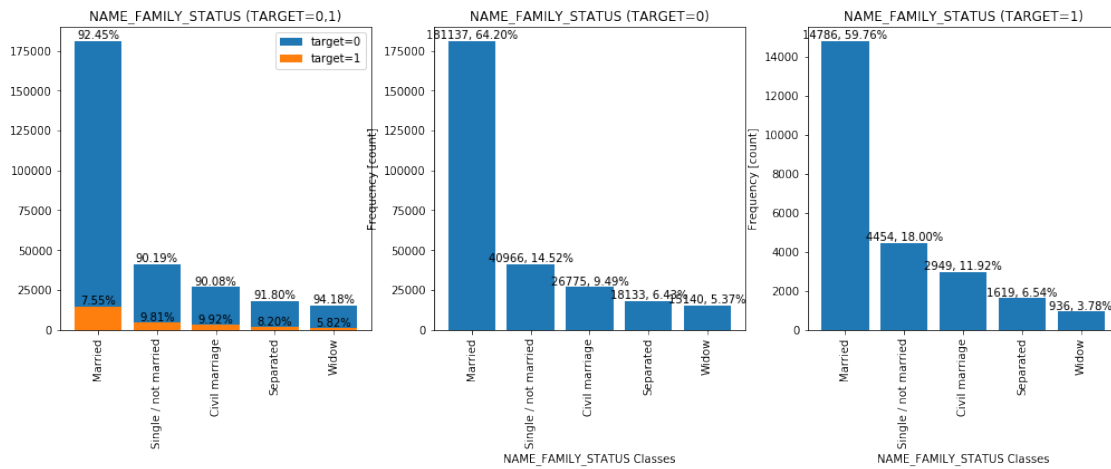
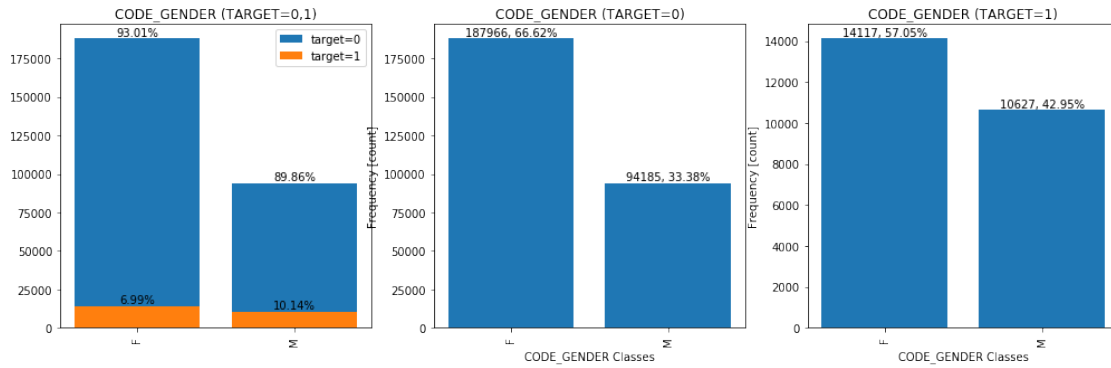
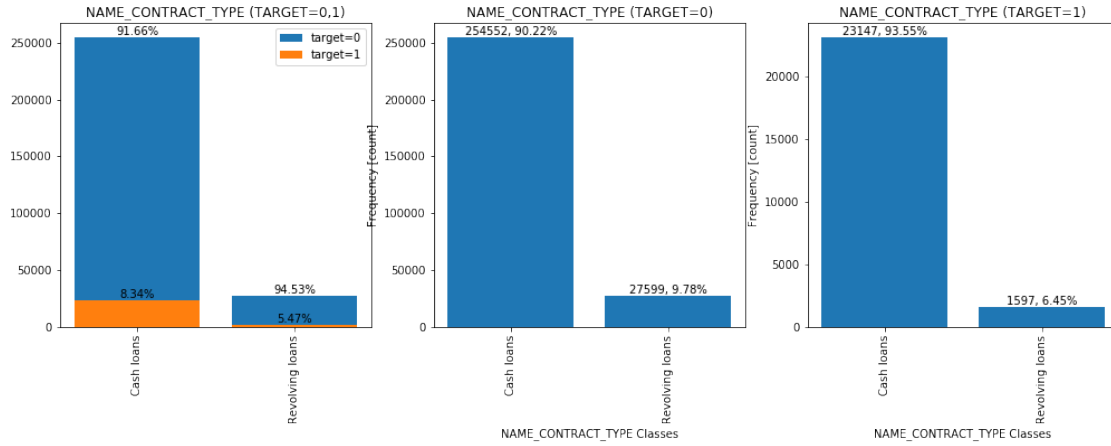
# CNT_FAM_MEMBERS
# 5 members (3+ children) - highest probability to not repay back (9.5%)
# 1, 3, 4 members - higher prob to NOT repay back than 2 members

# OCCUPATION_TYPE
# Drivers - bad borrowers in test set
# Security staff - same as Drivers - bad in test set
# Top : laborers and sales staff = 10.5% and 9.6% didn't pay back their loans

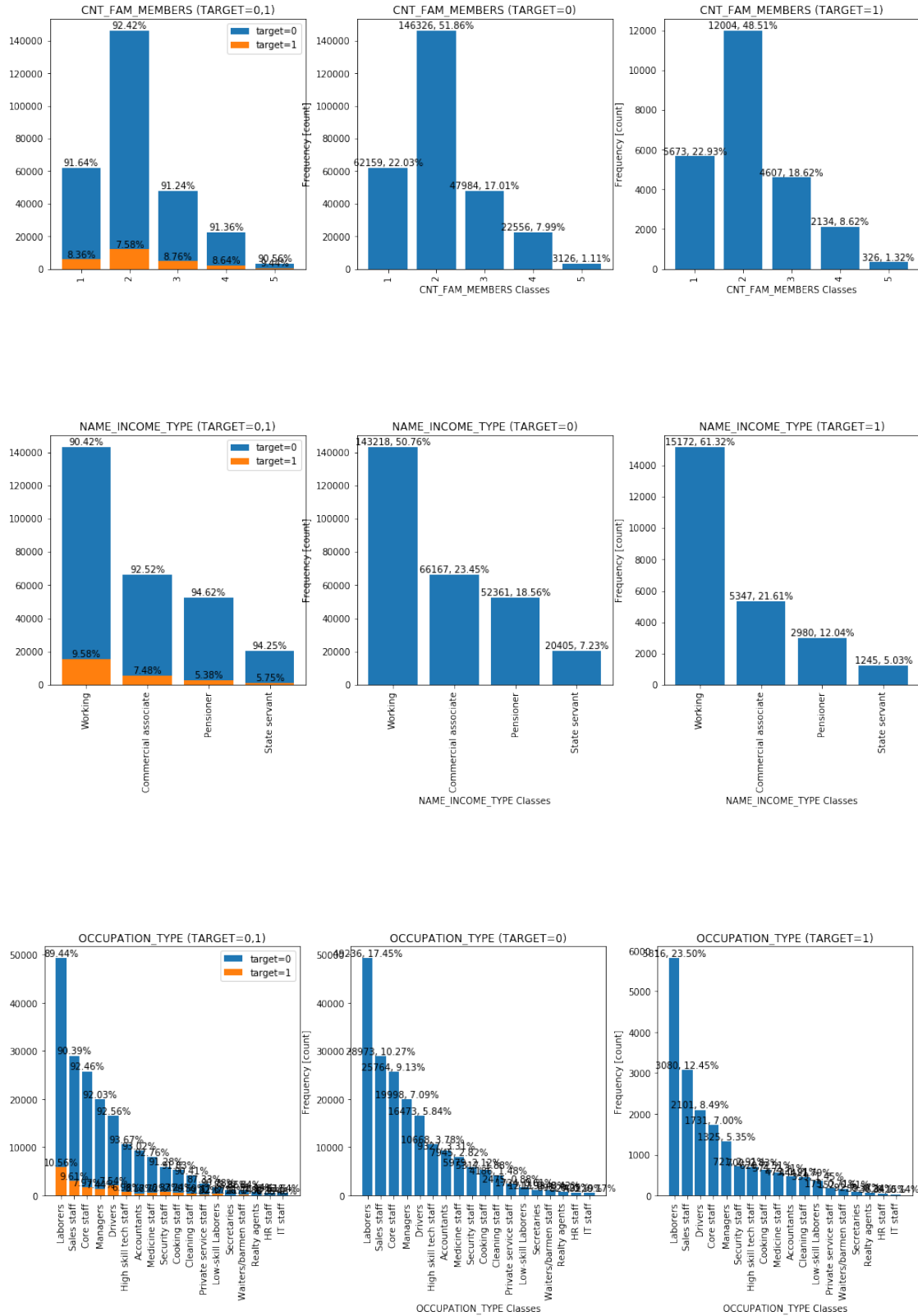
# NAME_EDUCATION_TYPE
# Lower secondary - critical class - really bad borrowers: 18% didn't repay
→back
# Higher education and academic degree - best ones
# Secondary and incomplete - same proportion of bad borrowers

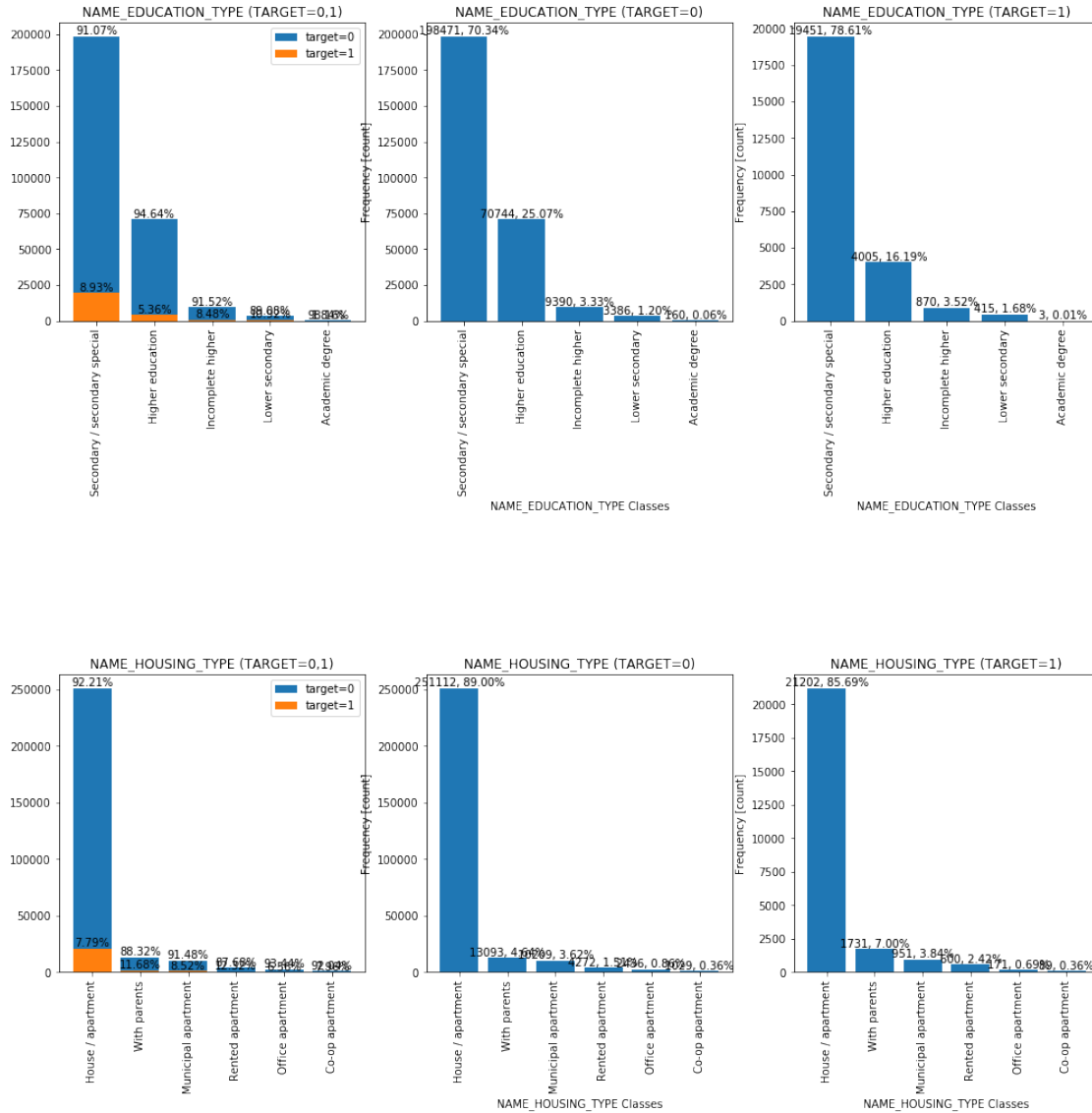
# NAME_HOUSING_TYPE
# Rented apartment - worst category of borrowers - 12.3%
# With parents - second worst category of borrowers - 11.7%
# Others: 7-9%

```









```
[142]: def display_dist( dataset_df, col_name, ax, \
                    n_bins=100, display_kde=True, title_add_text='', \
                    log_transform=False ):
    data_to_display = dataset_df[col_name]
    if log_transform:
        data_to_display = np.log( data_to_display )
    ax.hist(
        data_to_display,
        bins=n_bins,
        density=True,
        label='{0} distribution'.format(col_name),
        color='black'
    )
```

```

if display_kde:
    data_to_display.plot(kind='density', color='red', ax=ax)
ax.set_title('{0} density {1}'.format(col_name, title_add_text))
ax.legend()

def display_distr_traintest( train_df, test_df, col_name, display_kde=False,
    →log_transform=False ):
    fig, [ax_0, ax_1] = plt.subplots( 1, 2, figsize=(15, 7) )
    display_distr( train_df, col_name, ax=ax_0,
        title_add_text='TRAIN', display_kde=display_kde,
    →log_transform=log_transform )
    display_distr( test_df, col_name, ax=ax_1,
        title_add_text='TEST', display_kde=display_kde,
    →log_transform=log_transform )

def display_full_distr_overview( train_df, test_df, col_name ):
    display( train_df[col_name].describe(include='all') )
    display( test_df[col_name].describe(include='all') )
    display_distr_traintest( train_df, test_df, col_name, display_kde=True )
    display_distr_traintest(
        train_df, test_df, col_name,
        log_transform=True, display_kde=True
    )

```

[143]: # Type 3 overview: distribution of continuous features

```

# AMT_INCOME_TOTAL
# train: lots of outliers; range: [0.02565 mln; 117 mln]; mean = 0.0168 mln =
    →168k
# test: range: [0.0269415 mln; 4.41 mln]; mean = 0.0178 mln = 178k
display( 'AMT_INCOME_TOTAL' )
display_full_distr_overview( application_train, application_test,
    →'AMT_INCOME_TOTAL' )
display_distr_traintest( # borrowers withh total income < 10 mln
    application_train[ application_train['AMT_INCOME_TOTAL'] < 10**6 ],
    application_test[ application_test['AMT_INCOME_TOTAL'] < 10**6 ],
    'AMT_INCOME_TOTAL',
    display_kde=True
)

```

'AMT\_INCOME\_TOTAL'

```

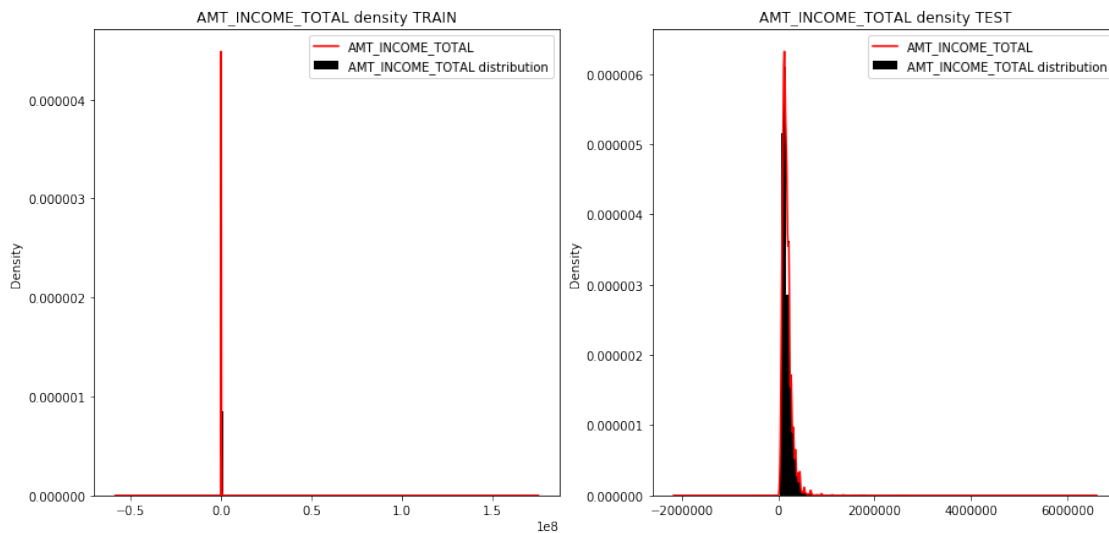
count    3.068950e+05
mean     1.687841e+05
std      2.372771e+05
min      2.565000e+04

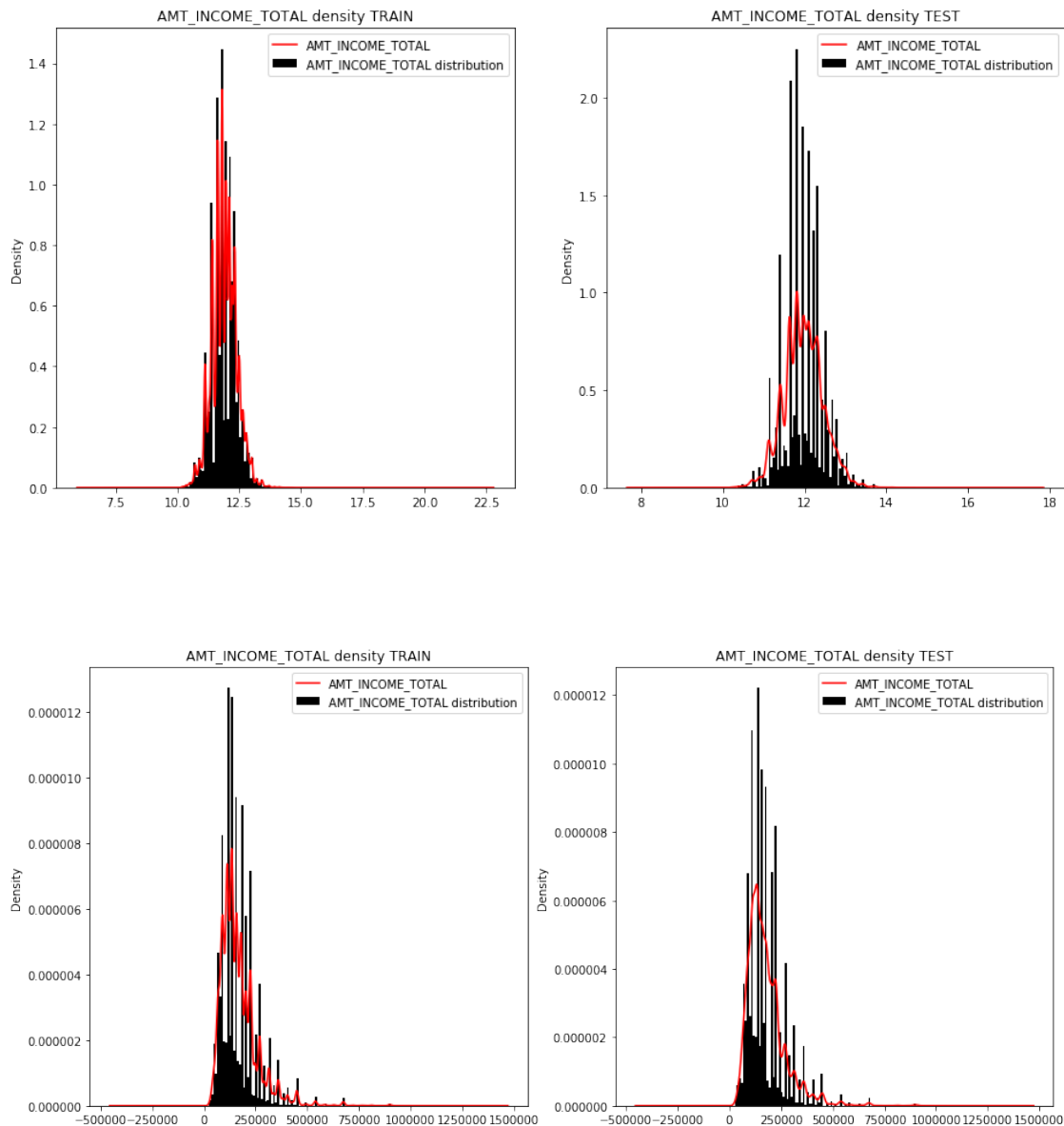
```



```
25%      1.125000e+05
50%      1.476000e+05
75%      2.025000e+05
max       1.170000e+08
Name: AMT_INCOME_TOTAL, dtype: float64
```

```
count      4.874400e+04
mean       1.784318e+05
std        1.015226e+05
min        2.694150e+04
25%        1.125000e+05
50%        1.575000e+05
75%        2.250000e+05
max        4.410000e+06
Name: AMT_INCOME_TOTAL, dtype: float64
```





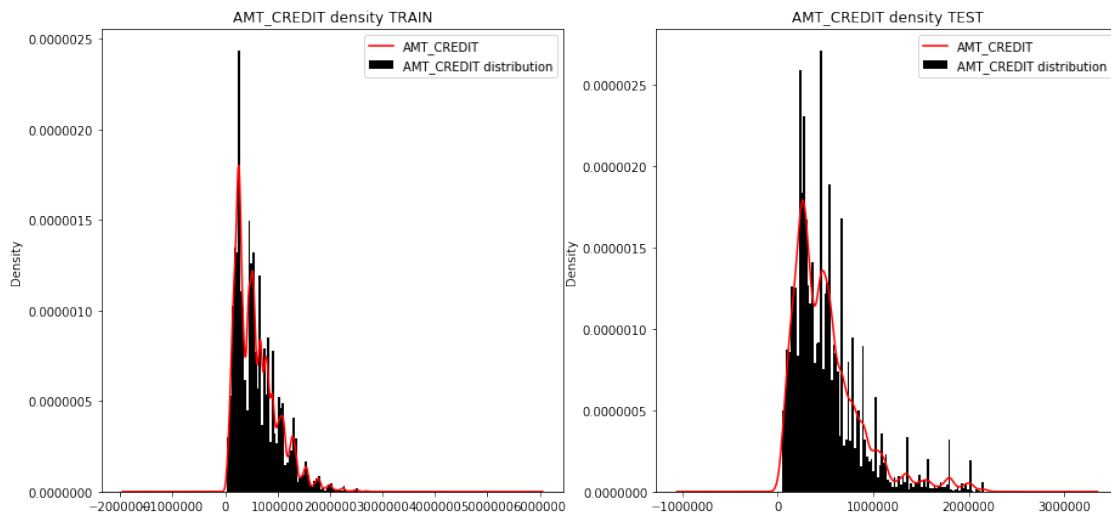
```
[144]: # AMT_CREDIT
display( 'AMT_CREDIT' )
display_full_distr_overview( application_train, application_test, 'AMT_CREDIT' )
```

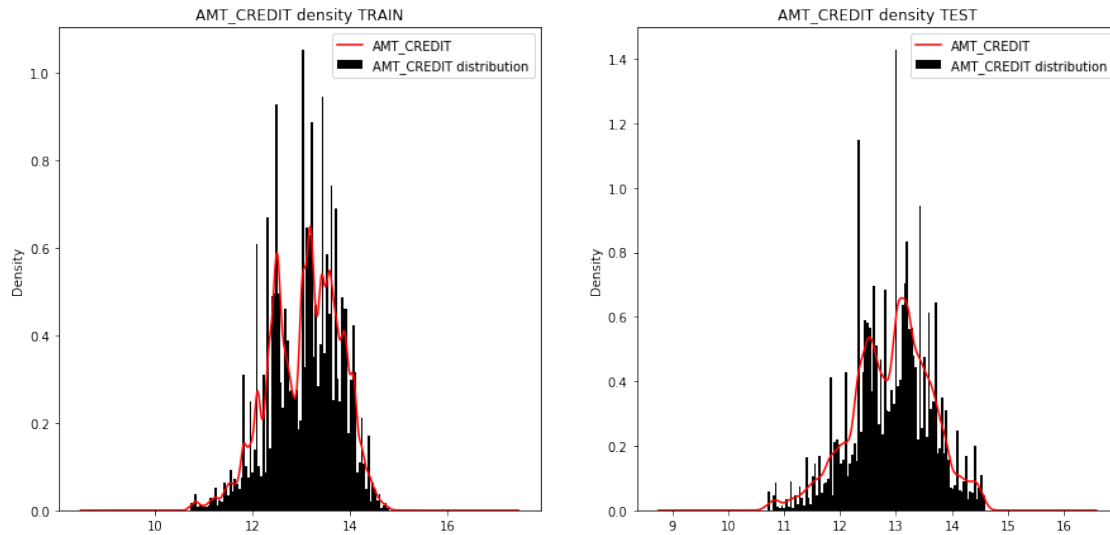
'AMT\_CREDIT'

count	3.068950e+05
mean	5.989916e+05
std	4.024480e+05
min	4.500000e+04
25%	2.700000e+05

50% 5.135310e+05  
75% 8.086500e+05  
max 4.050000e+06  
Name: AMT\_CREDIT, dtype: float64

count 4.874400e+04  
mean 5.167404e+05  
std 3.653970e+05  
min 4.500000e+04  
25% 2.606400e+05  
50% 4.500000e+05  
75% 6.750000e+05  
max 2.245500e+06  
Name: AMT\_CREDIT, dtype: float64





```
[145]: # AMT_ANNUITY
# Most of borrowers should have pay <100k per year - weird why such a small
→amount
display( 'AMT_ANNUITY' )
display_full_distr_overview( application_train, application_test, 'AMT_ANNUITY'
→)
```

'AMT\_ANNUITY'

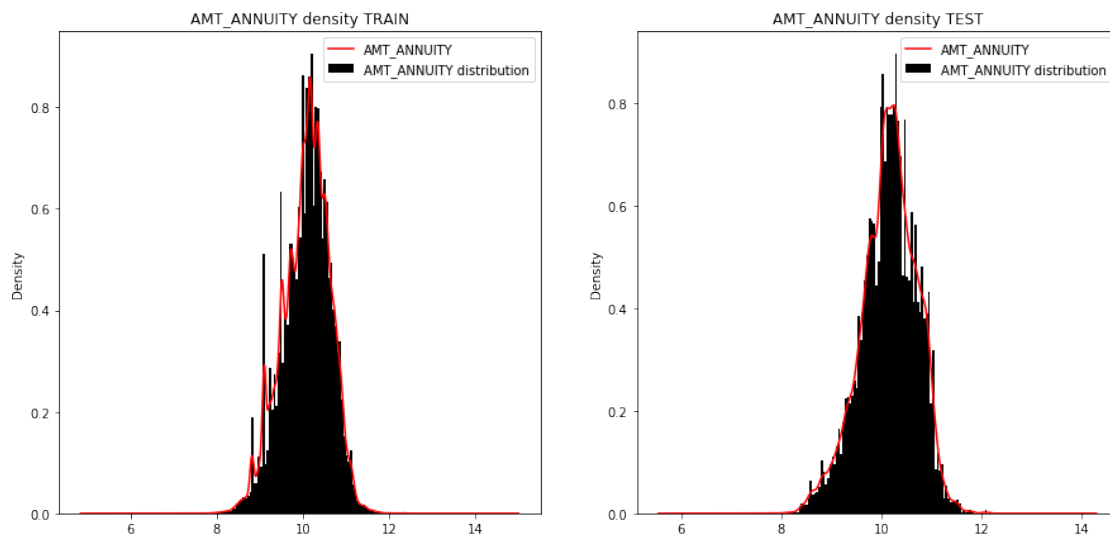
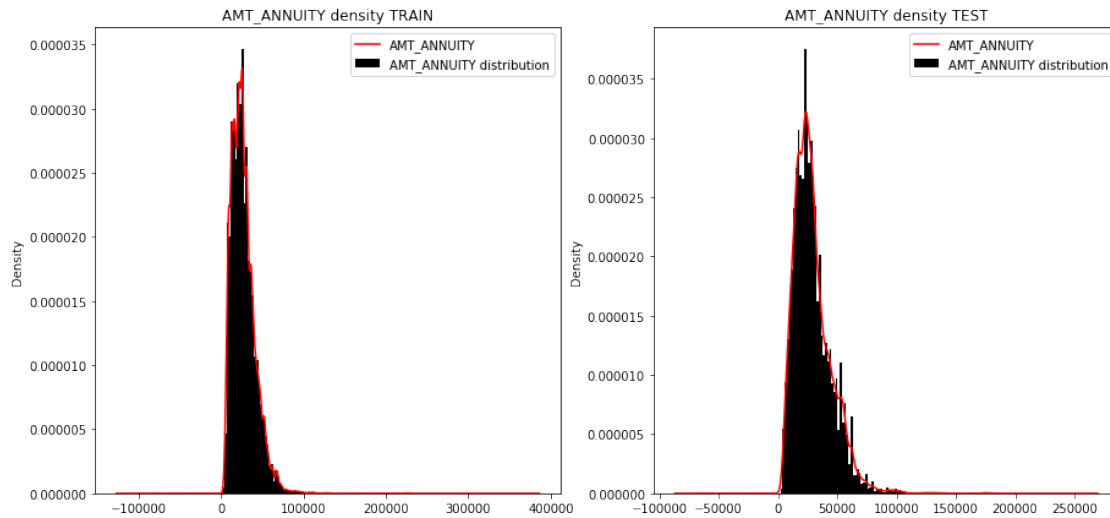
```
count    306883.000000
mean      27105.274497
std       14483.051034
min       1615.500000
25%       16524.000000
50%       24903.000000
75%       34596.000000
max       258025.500000
Name: AMT_ANNUITY, dtype: float64
```

```
count    48720.000000
mean      29426.240209
std       16016.368315
min       2295.000000
25%       17973.000000
50%       26199.000000
75%       37390.500000
max       180576.000000
Name: AMT_ANNUITY, dtype: float64
```

```

/home/max/.local/lib/python3.7/site-packages/numpy/lib/histograms.py:824:
RuntimeWarning: invalid value encountered in greater_equal
    keep = (tmp_a >= first_edge)
/home/max/.local/lib/python3.7/site-packages/numpy/lib/histograms.py:825:
RuntimeWarning: invalid value encountered in less_equal
    keep &= (tmp_a <= last_edge)

```



```

[146]: # AMT_GOODS_PRICE
# Peaks mean '25k', '50k', '75k', '100k', '150k', '200k' - approximate costs_
→told by borrowers
display( 'AMT_GOODS_PRICE' )

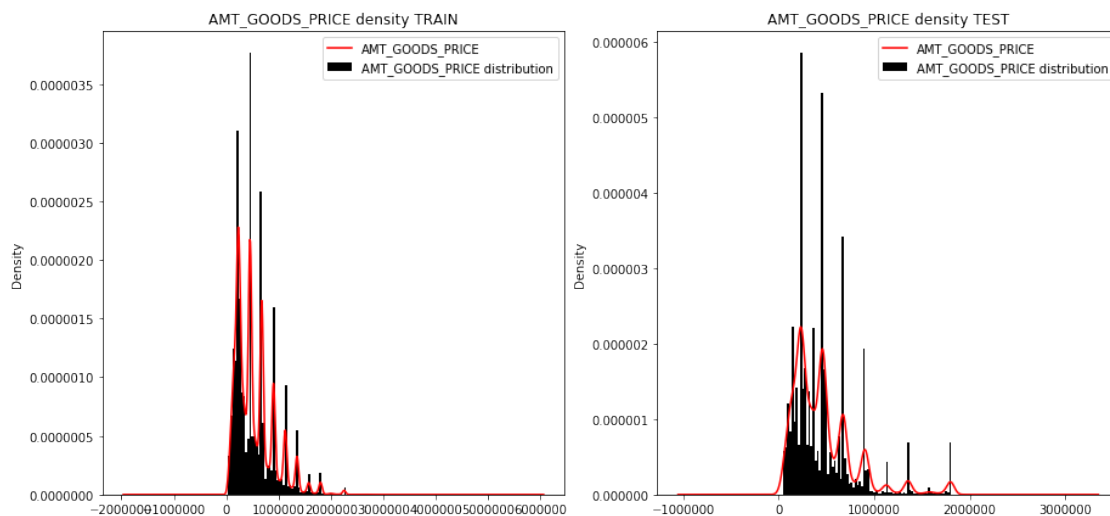
```

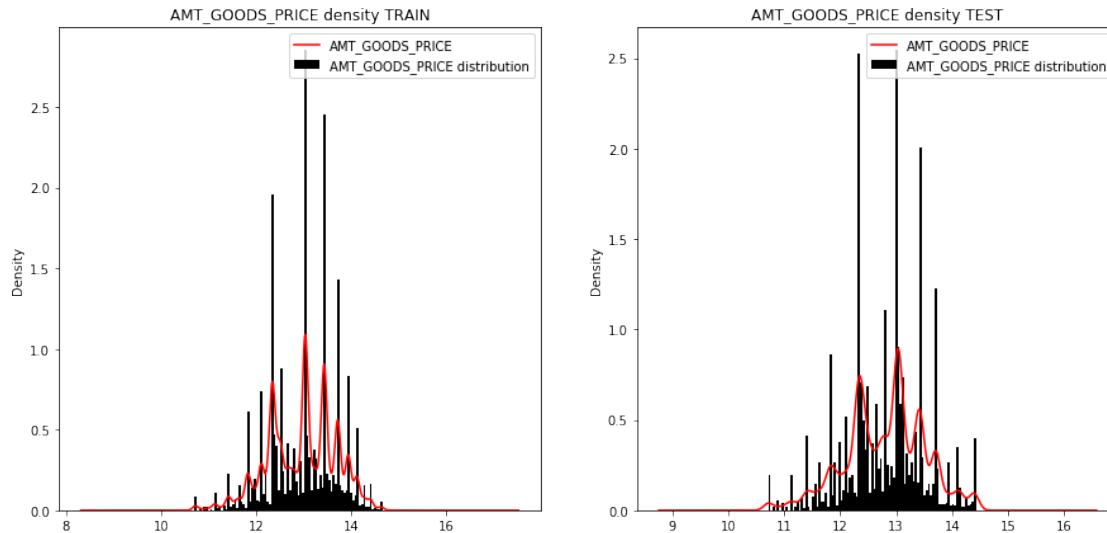
```
display_full_distr_overview( application_train, application_test,
→ 'AMT_GOODS_PRICE' )
```

'AMT\_GOODS\_PRICE'

```
count      3.066210e+05
mean       5.383644e+05
std        3.693834e+05
min        4.050000e+04
25%        2.385000e+05
50%        4.500000e+05
75%        6.795000e+05
max        4.050000e+06
Name: AMT_GOODS_PRICE, dtype: float64
```

```
count      4.874400e+04
mean       4.626188e+05
std        3.367102e+05
min        4.500000e+04
25%        2.250000e+05
50%        3.960000e+05
75%        6.300000e+05
max        2.245500e+06
Name: AMT_GOODS_PRICE, dtype: float64
```





```
[169]: # DAYS_BIRTH
# [20, 69] (so pensioners are not 80yo, but max=69y)
# mean: 43y train, 44y test
# 40->50 plunge - reason ?
# 50->55 rise - retirement?

display( 'DAYS_BIRTH transformed to YEARS_BIRTH' )

DAYS_IN_YEAR = 365
train_years_birth = pd.DataFrame( application_train['DAYS_BIRTH'] /_
    ↳DAYS_IN_YEAR * -1.0 )
test_years_birth = pd.DataFrame( application_test['DAYS_BIRTH'] / DAYS_IN_YEAR_
    ↳* -1.0 )

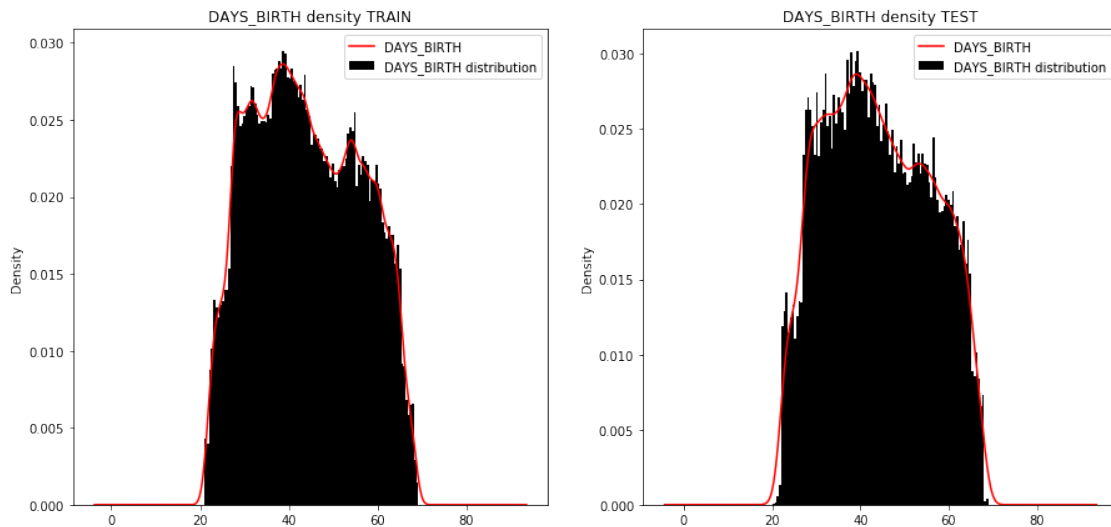
display( train_years_birth.describe(include='all').T )
display( test_years_birth.describe(include='all').T )

display_distr_taintest( # borrowers withh total income < 10 mln
    train_years_birth, test_years_birth, 'DAYS_BIRTH',
    display_kde=True, log_transform=False
)
```

'DAYS\_BIRTH transformed to YEARS\_BIRTH'

	count	mean	std	min	25%	50% \
DAYS_BIRTH	306895.0	43.946676	11.961695	20.517808	34.005479	43.172603
		75%	max			
DAYS_BIRTH	53.939726	69.120548				

	count	mean	std	min	25%	50%	\
DAYS_BIRTH	48744.0	44.02215	11.851782	20.10411	34.235616	43.246575	
		75%	max				
DAYS_BIRTH	53.8	69.027397					



```
[173]: # DAYS_EMPLOYED
# anomaly at years=-1000 - 'unemployed' ?
# note: because data source comes from Russia, many of borrowers could lie
# → about job OR job could not register them officially (gray/dark salary)
# 90s - did Home Credit count job experience from USSR ?

# Most of borrowers are 'young' (!) professionals - 50pentl is 4.5y in train,
# → 4.8 in test
# Almost no borrowers with 40+ years job experience

display( 'DAYS_EMPLOYED transformed to YEARS_EMPLOYED' )

DAYS_IN_YEAR = 365
train_years_emp = pd.DataFrame( application_train['DAYS_EMPLOYED'] /
    → DAYS_IN_YEAR * -1.0 )
test_years_emp = pd.DataFrame( application_test['DAYS_EMPLOYED'] / DAYS_IN_YEAR,
    → * -1.0 )

display( train_years_emp[train_years_emp['DAYS_EMPLOYED'] > 0].
    → describe(include='all').T )
```



```
display( test_years_emp[test_years_emp['DAYS_EMPLOYED'] > 0].
→describe(include='all').T )

display_distr_taintest( # borrowers withh total income < 10 mln
    train_years_emp[train_years_emp['DAYS_EMPLOYED'] > 0],
    test_years_emp[test_years_emp['DAYS_EMPLOYED'] > 0],
    'DAYS_EMPLOYED',
    display_kde=True, log_transform=False
)
```

'DAYS\_EMPLOYED transformed to YEARS\_EMPLOYED'

	count	mean	std	min	25%	50% \
DAYS_EMPLOYED	251562.0	6.532011	6.409052	0.00274	2.10137	4.512329
	75%	max				
DAYS_EMPLOYED	8.69863	49.073973				

	count	mean	std	min	25%	50% \
DAYS_EMPLOYED	39470.0	6.785586	6.323189	0.00274	2.358904	4.835616
	75%	max				
DAYS_EMPLOYED	9.119863	47.843836				

