risk_project

April 24, 2023

[4]:
```
import pandas as pd
import numpy as np
# matplotlib and seaborn for plotting
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
```

[5]:
```
# Suppress warnings
import warnings
warnings.filterwarnings('ignore')
```

[6]:
```
app_train= pd.read_csv('application_train.csv',nrows=150000)
print('Training data shape: ', app_train.shape)
app_train.head()
```

```
Training data shape:  (150000, 122)
```

[6]:
```
   SK_ID_CURR  TARGET NAME_CONTRACT_TYPE CODE_GENDER FLAG_OWN_CAR  \
0      100002       1         Cash loans           M            N
1      100003       0         Cash loans           F            N
2      100004       0    Revolving loans           M            Y
3      100006       0         Cash loans           F            N
4      100007       0         Cash loans           M            N

  FLAG_OWN_REALTY  CNT_CHILDREN  AMT_INCOME_TOTAL  AMT_CREDIT  AMT_ANNUITY  \
0               Y             0          202500.0    406597.5      24700.5
1               N             0          270000.0   1293502.5      35698.5
2               Y             0           67500.0    135000.0       6750.0
3               Y             0          135000.0    312682.5      29686.5
4               Y             0          121500.0    513000.0      21865.5

   …  FLAG_DOCUMENT_18 FLAG_DOCUMENT_19 FLAG_DOCUMENT_20 FLAG_DOCUMENT_21  \
0  …                 0                0                0                0
1  …                 0                0                0                0
2  …                 0                0                0                0
3  …                 0                0                0                0
4  …                 0                0                0                0
```

```
   AMT_REQ_CREDIT_BUREAU_HOUR AMT_REQ_CREDIT_BUREAU_DAY  \
0                        0.0                       0.0
1                        0.0                       0.0
2                        0.0                       0.0
3                        NaN                       NaN
4                        0.0                       0.0

   AMT_REQ_CREDIT_BUREAU_WEEK  AMT_REQ_CREDIT_BUREAU_MON  \
0                         0.0                        0.0
1                         0.0                        0.0
2                         0.0                        0.0
3                         NaN                        NaN
4                         0.0                        0.0

   AMT_REQ_CREDIT_BUREAU_QRT  AMT_REQ_CREDIT_BUREAU_YEAR
0                        0.0                         1.0
1                        0.0                         0.0
2                        0.0                         0.0
3                        NaN                         NaN
4                        0.0                         0.0

[5 rows x 122 columns]
```

[7]:
```python
def test_na_num(df):
    for column in df:
        print(sum(df[column].isna()), column)
```

[8]:
```python
test_na_num(app_train)
```

```
0 SK_ID_CURR
0 TARGET
0 NAME_CONTRACT_TYPE
0 CODE_GENDER
0 FLAG_OWN_CAR
0 FLAG_OWN_REALTY
0 CNT_CHILDREN
0 AMT_INCOME_TOTAL
0 AMT_CREDIT
8 AMT_ANNUITY
126 AMT_GOODS_PRICE
614 NAME_TYPE_SUITE
0 NAME_INCOME_TYPE
0 NAME_EDUCATION_TYPE
0 NAME_FAMILY_STATUS
0 NAME_HOUSING_TYPE
0 REGION_POPULATION_RELATIVE
0 DAYS_BIRTH
```

```
0 DAYS_EMPLOYED
0 DAYS_REGISTRATION
0 DAYS_ID_PUBLISH
99076 OWN_CAR_AGE
0 FLAG_MOBIL
0 FLAG_EMP_PHONE
0 FLAG_WORK_PHONE
0 FLAG_CONT_MOBILE
0 FLAG_PHONE
0 FLAG_EMAIL
47081 OCCUPATION_TYPE
1 CNT_FAM_MEMBERS
0 REGION_RATING_CLIENT
0 REGION_RATING_CLIENT_W_CITY
0 WEEKDAY_APPR_PROCESS_START
0 HOUR_APPR_PROCESS_START
0 REG_REGION_NOT_LIVE_REGION
0 REG_REGION_NOT_WORK_REGION
0 LIVE_REGION_NOT_WORK_REGION
0 REG_CITY_NOT_LIVE_CITY
0 REG_CITY_NOT_WORK_CITY
0 LIVE_CITY_NOT_WORK_CITY
0 ORGANIZATION_TYPE
84480 EXT_SOURCE_1
328 EXT_SOURCE_2
29675 EXT_SOURCE_3
76287 APARTMENTS_AVG
87878 BASEMENTAREA_AVG
73290 YEARS_BEGINEXPLUATATION_AVG
99733 YEARS_BUILD_AVG
104837 COMMONAREA_AVG
80103 ELEVATORS_AVG
75651 ENTRANCES_AVG
74767 FLOORSMAX_AVG
101773 FLOORSMIN_AVG
89126 LANDAREA_AVG
102590 LIVINGAPARTMENTS_AVG
75441 LIVINGAREA_AVG
104147 NONLIVINGAPARTMENTS_AVG
82904 NONLIVINGAREA_AVG
76287 APARTMENTS_MODE
87878 BASEMENTAREA_MODE
73290 YEARS_BEGINEXPLUATATION_MODE
99733 YEARS_BUILD_MODE
104837 COMMONAREA_MODE
80103 ELEVATORS_MODE
75651 ENTRANCES_MODE
74767 FLOORSMAX_MODE
```

```
101773 FLOORSMIN_MODE
89126 LANDAREA_MODE
102590 LIVINGAPARTMENTS_MODE
75441 LIVINGAREA_MODE
104147 NONLIVINGAPARTMENTS_MODE
82904 NONLIVINGAREA_MODE
76287 APARTMENTS_MEDI
87878 BASEMENTAREA_MEDI
73290 YEARS_BEGINEXPLUATATION_MEDI
99733 YEARS_BUILD_MEDI
104837 COMMONAREA_MEDI
80103 ELEVATORS_MEDI
75651 ENTRANCES_MEDI
74767 FLOORSMAX_MEDI
101773 FLOORSMIN_MEDI
89126 LANDAREA_MEDI
102590 LIVINGAPARTMENTS_MEDI
75441 LIVINGAREA_MEDI
104147 NONLIVINGAPARTMENTS_MEDI
82904 NONLIVINGAREA_MEDI
102561 FONDKAPREMONT_MODE
75421 HOUSETYPE_MODE
72558 TOTALAREA_MODE
76384 WALLSMATERIAL_MODE
71243 EMERGENCYSTATE_MODE
503 OBS_30_CNT_SOCIAL_CIRCLE
503 DEF_30_CNT_SOCIAL_CIRCLE
503 OBS_60_CNT_SOCIAL_CIRCLE
503 DEF_60_CNT_SOCIAL_CIRCLE
1 DAYS_LAST_PHONE_CHANGE
0 FLAG_DOCUMENT_2
0 FLAG_DOCUMENT_3
0 FLAG_DOCUMENT_4
0 FLAG_DOCUMENT_5
0 FLAG_DOCUMENT_6
0 FLAG_DOCUMENT_7
0 FLAG_DOCUMENT_8
0 FLAG_DOCUMENT_9
0 FLAG_DOCUMENT_10
0 FLAG_DOCUMENT_11
0 FLAG_DOCUMENT_12
0 FLAG_DOCUMENT_13
0 FLAG_DOCUMENT_14
0 FLAG_DOCUMENT_15
0 FLAG_DOCUMENT_16
0 FLAG_DOCUMENT_17
0 FLAG_DOCUMENT_18
0 FLAG_DOCUMENT_19
```

```
0 FLAG_DOCUMENT_20
0 FLAG_DOCUMENT_21
20117 AMT_REQ_CREDIT_BUREAU_HOUR
20117 AMT_REQ_CREDIT_BUREAU_DAY
20117 AMT_REQ_CREDIT_BUREAU_WEEK
20117 AMT_REQ_CREDIT_BUREAU_MON
20117 AMT_REQ_CREDIT_BUREAU_QRT
20117 AMT_REQ_CREDIT_BUREAU_YEAR
```

[9]:

```python
# Testing data features
app_test = pd.read_csv('application_test.csv')
print('Testing data shape: ', app_test.shape)
app_test.head()
```

```
Testing data shape:  (48744, 121)
```

[9]:

```
   SK_ID_CURR NAME_CONTRACT_TYPE CODE_GENDER FLAG_OWN_CAR FLAG_OWN_REALTY  \
0      100001         Cash loans           F            N               Y
1      100005         Cash loans           M            N               Y
2      100013         Cash loans           M            Y               Y
3      100028         Cash loans           F            N               Y
4      100038         Cash loans           M            Y               N

   CNT_CHILDREN  AMT_INCOME_TOTAL  AMT_CREDIT  AMT_ANNUITY  AMT_GOODS_PRICE  \
0             0          135000.0    568800.0      20560.5         450000.0
1             0           99000.0    222768.0      17370.0         180000.0
2             0          202500.0    663264.0      69777.0         630000.0
3             2          315000.0   1575000.0      49018.5        1575000.0
4             1          180000.0    625500.0      32067.0         625500.0

   … FLAG_DOCUMENT_18 FLAG_DOCUMENT_19 FLAG_DOCUMENT_20 FLAG_DOCUMENT_21  \
0  …                0                0                0                0
1  …                0                0                0                0
2  …                0                0                0                0
3  …                0                0                0                0
4  …                0                0                0                0

  AMT_REQ_CREDIT_BUREAU_HOUR  AMT_REQ_CREDIT_BUREAU_DAY  \
0                        0.0                        0.0
1                        0.0                        0.0
2                        0.0                        0.0
3                        0.0                        0.0
4                        NaN                        NaN

   AMT_REQ_CREDIT_BUREAU_WEEK  AMT_REQ_CREDIT_BUREAU_MON  \
0                         0.0                        0.0
1                         0.0                        0.0
```

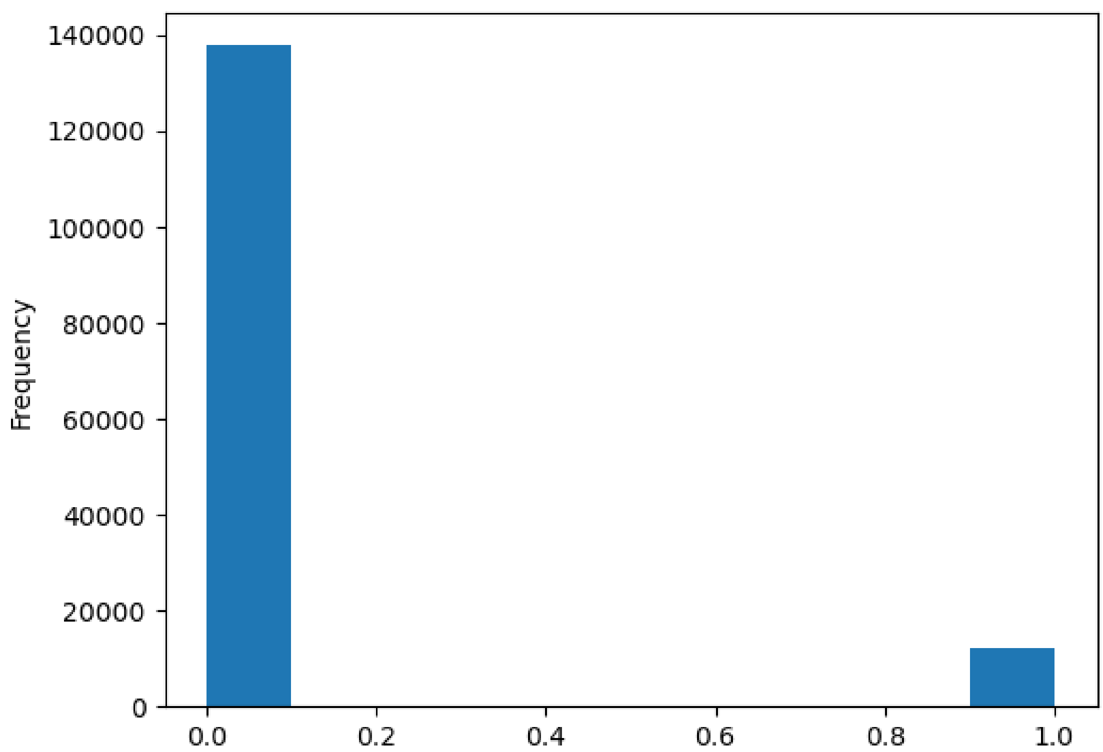
```
2                          0.0                          0.0
3                          0.0                          0.0
4                          NaN                          NaN

   AMT_REQ_CREDIT_BUREAU_QRT  AMT_REQ_CREDIT_BUREAU_YEAR
0                        0.0                         0.0
1                        0.0                         3.0
2                        1.0                         4.0
3                        0.0                         3.0
4                        NaN                         NaN

[5 rows x 121 columns]
```

[10]:
```
#test_na_num(app_test)
```

[11]:
```
app_train['TARGET'].value_counts()
```

[11]:
```
0    137747
1     12253
Name: TARGET, dtype: int64
```

[12]:
```
app_train['TARGET'].astype(int).plot.hist();
```



[13]:
```
#Examine missing value
```

[14]:
```
# Function to calculate missing values by column# Funct
def missing_values_table(df):
        # Total missing values
        mis_val = df.isnull().sum()

        # Percentage of missing values
        mis_val_percent = 100 * df.isnull().sum() / len(df)

        # Make a table with the results
        mis_val_table = pd.concat([mis_val, mis_val_percent], axis=1)

        # Rename the columns
        mis_val_table_ren_columns = mis_val_table.rename(
        columns = {0 : 'Missing Values', 1 : '% of Total Values'})

        # Sort the table by percentage of missing descending
        mis_val_table_ren_columns = mis_val_table_ren_columns[
            mis_val_table_ren_columns.iloc[:,1] != 0].sort_values(
        '% of Total Values', ascending=False).round(1)

        # Print some summary information
        print ("Your selected dataframe has " + str(df.shape[1]) + " columns.\n"
            "There are " + str(mis_val_table_ren_columns.shape[0]) +
              " columns that have missing values.")

        # Return the dataframe with missing information
        return mis_val_table_ren_columns
```

[15]:
```
missing_values = missing_values_table(app_train)
missing_values.head(20)
```

```
Your selected dataframe has 122 columns.
There are 67 columns that have missing values.
```

[15]:

	Missing Values	% of Total Values
COMMONAREA_MEDI	104837	69.9
COMMONAREA_AVG	104837	69.9
COMMONAREA_MODE	104837	69.9
NONLIVINGAPARTMENTS_MEDI	104147	69.4
NONLIVINGAPARTMENTS_MODE	104147	69.4
NONLIVINGAPARTMENTS_AVG	104147	69.4
LIVINGAPARTMENTS_MODE	102590	68.4
LIVINGAPARTMENTS_MEDI	102590	68.4
LIVINGAPARTMENTS_AVG	102590	68.4
FONDKAPREMONT_MODE	102561	68.4

```
FLOORSMIN_MODE                       101773              67.8
FLOORSMIN_MEDI                       101773              67.8
FLOORSMIN_AVG                        101773              67.8
YEARS_BUILD_MODE                      99733              66.5
YEARS_BUILD_MEDI                      99733              66.5
YEARS_BUILD_AVG                       99733              66.5
OWN_CAR_AGE                           99076              66.1
LANDAREA_AVG                          89126              59.4
LANDAREA_MEDI                         89126              59.4
LANDAREA_MODE                         89126              59.4
```

[16]:
```
app_train.dtypes.value_counts()
```

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

[17]:
```
# Create a label encoder object
le = LabelEncoder()
le_count = 0

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

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

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

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

[18]:
```
# one-hot encoding of categorical variables
app_train = pd.get_dummies(app_train)
app_test = pd.get_dummies(app_test)

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

```
Training Features shape:  (150000, 243)
```

```
Testing Features shape:  (48744, 239)
```

[19]:
```
train_labels = app_train['TARGET']

# Align the training and testing data, keep only columns present in both␣
↪dataframes
app_train, app_test = app_train.align(app_test, join = 'inner', axis = 1)

# Add the target back in
app_train['TARGET'] = train_labels

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

```
Training Features shape:  (150000, 240)
Testing Features shape:  (48744, 239)
```

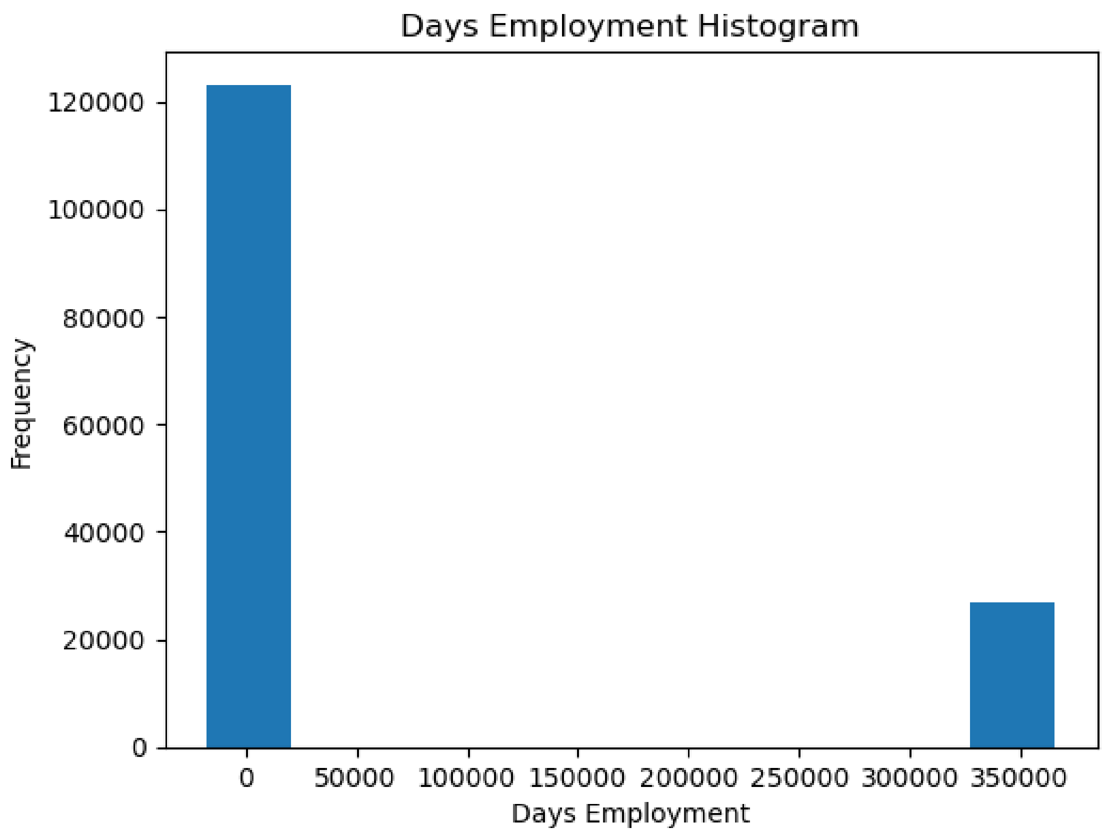
[20]:
```
(app_train['DAYS_BIRTH'] / -365).describe()
```

[20]:
```
count    150000.000000
mean         43.897489
std          11.969258
min          21.030137
25%          33.923288
50%          43.091781
75%          53.890411
max          69.043836
Name: DAYS_BIRTH, dtype: float64
```

[21]:
```
app_train['DAYS_EMPLOYED'].describe()
```

[21]:
```
count    150000.000000
mean      63600.189720
std      141090.185226
min      -17531.000000
25%       -2752.000000
50%       -1214.000000
75%        -289.000000
max      365243.000000
Name: DAYS_EMPLOYED, dtype: float64
```

[22]:
```
app_train['DAYS_EMPLOYED'].plot.hist(title = 'Days Employment Histogram');
plt.xlabel('Days Employment');
```



[23]:
```
sum(app_train['DAYS_EMPLOYED']==365243)
```

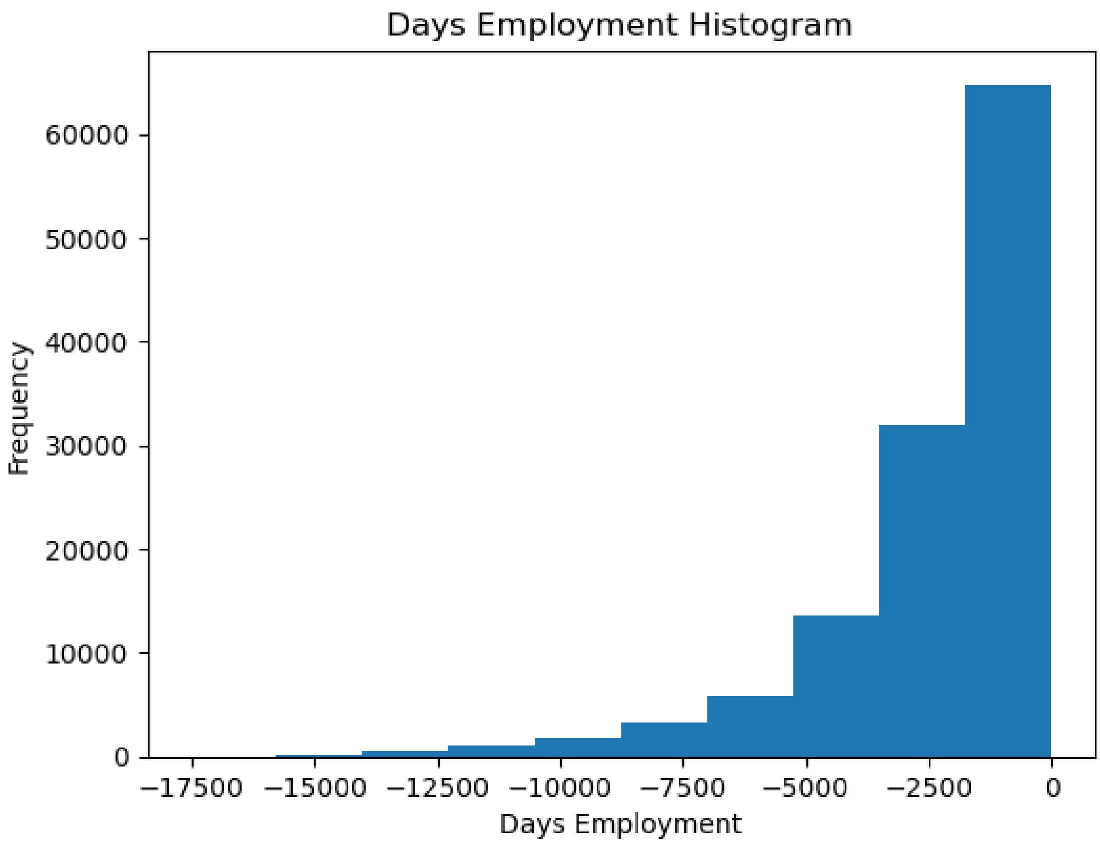
[23]: 26921

[24]:
```
anom = app_train[app_train['DAYS_EMPLOYED'] == 365243]
non_anom = app_train[app_train['DAYS_EMPLOYED'] != 365243]
print('The non-anomalies default on %0.2f%% of loans' % (100 *␣
  ↪non_anom['TARGET'].mean()))
print('The anomalies default on %0.2f%% of loans' % (100 * anom['TARGET'].
  ↪mean()))
print('There are %d anomalous days of employment' % len(anom))
```

```
The non-anomalies default on 8.75% of loans
The anomalies default on 5.50% of loans
There are 26921 anomalous days of employment
```

[25]:
```
# Create an anomalous flag column
app_train['DAYS_EMPLOYED_ANOM'] = app_train["DAYS_EMPLOYED"] == 365243

# Replace the anomalous values with nan
```

```
app_train['DAYS_EMPLOYED'].replace({365243: np.nan}, inplace = True)

app_train['DAYS_EMPLOYED'].plot.hist(title = 'Days Employment Histogram');
plt.xlabel('Days Employment');
```



[26]:
```
app_train['DAYS_EMPLOYED_ANOM']
```

[26]:
```
0          False
1          False
2          False
3          False
4          False
           …
149995      True
149996     False
149997     False
149998     False
149999     False
Name: DAYS_EMPLOYED_ANOM, Length: 150000, dtype: bool
```

[27]:
```
app_test['DAYS_EMPLOYED_ANOM'] = app_test["DAYS_EMPLOYED"] == 365243
app_test["DAYS_EMPLOYED"].replace({365243: np.nan}, inplace = True)

print('There are %d anomalies in the test data out of %d entries' %␣
↪(app_test["DAYS_EMPLOYED_ANOM"].sum(), len(app_test)))
```

```
There are 9274 anomalies in the test data out of 48744 entries
```

[28]:
```
# Find correlations with the target and sort
correlations = app_train.corr()['TARGET'].sort_values()

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

```
Most Positive Correlations:
 OWN_CAR_AGE                                           0.041785
FLAG_DOCUMENT_3                                      0.044801
FLAG_EMP_PHONE                                       0.045584
OCCUPATION_TYPE_Laborers                             0.045945
NAME_EDUCATION_TYPE_Secondary / secondary special    0.050222
REG_CITY_NOT_WORK_CITY                               0.051757
DAYS_ID_PUBLISH                                      0.052632
DAYS_LAST_PHONE_CHANGE                               0.054896
CODE_GENDER_M                                        0.055870
NAME_INCOME_TYPE_Working                             0.058115
REGION_RATING_CLIENT                                 0.060561
REGION_RATING_CLIENT_W_CITY                          0.061751
DAYS_EMPLOYED                                        0.073448
DAYS_BIRTH                                           0.079078
TARGET                                               1.000000
Name: TARGET, dtype: float64

Most Negative Correlations:
 EXT_SOURCE_3                            -0.176890
EXT_SOURCE_2                           -0.162884
EXT_SOURCE_1                           -0.155094
NAME_EDUCATION_TYPE_Higher education   -0.056386
CODE_GENDER_F                          -0.055856
FLOORSMAX_AVG                          -0.047586
FLOORSMAX_MEDI                         -0.047466
FLOORSMAX_MODE                         -0.046419
NAME_INCOME_TYPE_Pensioner             -0.045842
DAYS_EMPLOYED_ANOM                     -0.045611
ORGANIZATION_TYPE_XNA                  -0.045611
EMERGENCYSTATE_MODE_No                 -0.043953
HOUSETYPE_MODE_block of flats          -0.042745
FLOORSMIN_AVG                          -0.040722
```

```
FLOORSMIN_MEDI                              -0.040663
Name: TARGET, dtype: float64
```

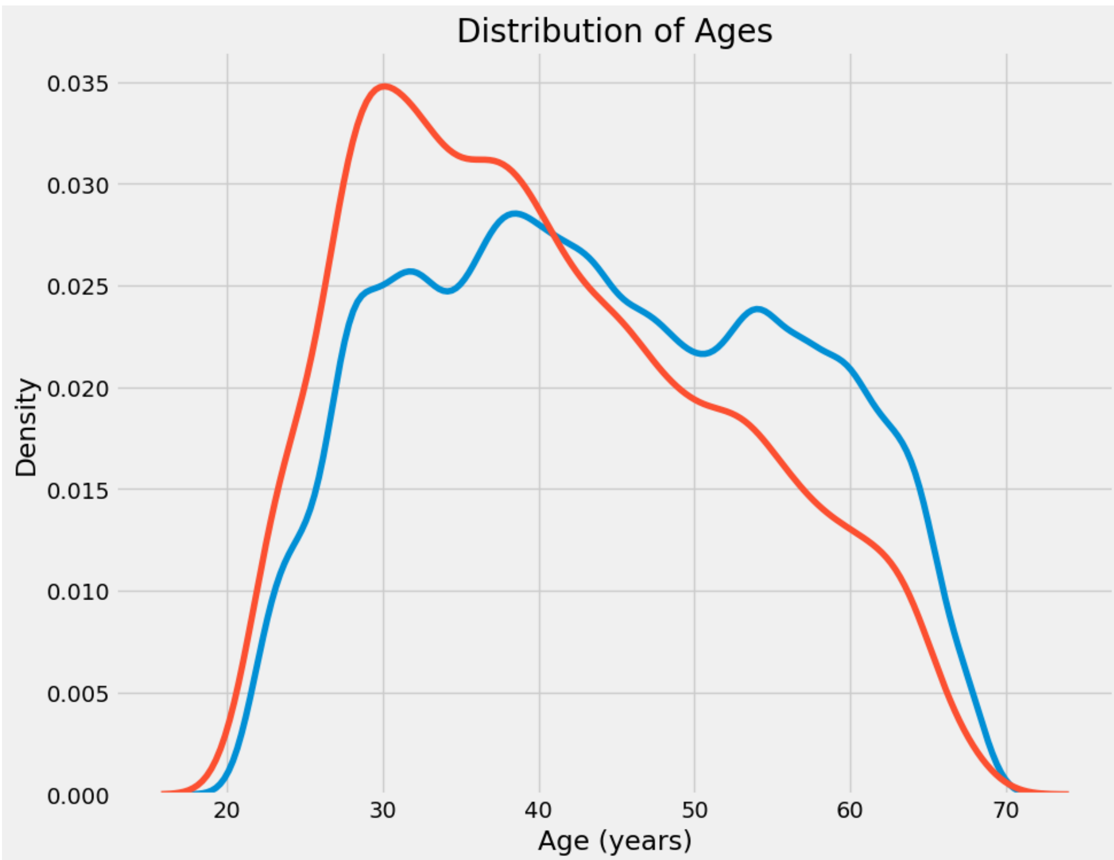
[29]:
```
# Find the correlation of the positive days since birth and target
app_train['DAYS_BIRTH'] = abs(app_train['DAYS_BIRTH'])
app_train['DAYS_BIRTH'].corr(app_train['TARGET'])
#this means when clients get older, they are less likely to default.
```

[29]:
```
-0.07907768556462803
```

[30]:
```
# Set the style of plots
plt.style.use('fivethirtyeight')

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

[31]:
```
plt.figure(figsize = (10, 8))

# KDE plot of loans that were repaid on time
```

```
sns.kdeplot(app_train.loc[app_train['TARGET'] == 0, 'DAYS_BIRTH'] / 365, label␣
 ↪= 'target == 0')

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

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



[32]:
```
# Age information into a separate dataframe
age_data = app_train[['TARGET', 'DAYS_BIRTH']]
age_data['YEARS_BIRTH'] = age_data['DAYS_BIRTH'] / 365

# Bin the age data
age_data['YEARS_BINNED'] = pd.cut(age_data['YEARS_BIRTH'], bins = np.
 ↪linspace(20, 70, num = 11))
age_data.head(10)
```

[32]:

```
   TARGET  DAYS_BIRTH  YEARS_BIRTH  YEARS_BINNED
0       1        9461    25.920548  (25.0, 30.0]
1       0       16765    45.931507  (45.0, 50.0]
2       0       19046    52.180822  (50.0, 55.0]
3       0       19005    52.068493  (50.0, 55.0]
4       0       19932    54.608219  (50.0, 55.0]
5       0       16941    46.413699  (45.0, 50.0]
6       0       13778    37.747945  (35.0, 40.0]
7       0       18850    51.643836  (50.0, 55.0]
8       0       20099    55.065753  (55.0, 60.0]
9       0       14469    39.641096  (35.0, 40.0]
```

[33]:

```
# Group by the bin and calculate averages
age_groups  = age_data.groupby('YEARS_BINNED').mean()
age_groups
```

[33]:

```
                TARGET    DAYS_BIRTH  YEARS_BIRTH
YEARS_BINNED
(20.0, 25.0]  0.121172   8526.558996    23.360436
(25.0, 30.0]  0.113172  10155.592604    27.823541
(30.0, 35.0]  0.105326  11849.435405    32.464207
(35.0, 40.0]  0.090467  13707.467404    37.554705
(40.0, 45.0]  0.078595  15497.632157    42.459266
(45.0, 50.0]  0.075434  17322.595896    47.459167
(50.0, 55.0]  0.067795  19194.894358    52.588752
(55.0, 60.0]  0.054991  20983.522133    57.489102
(60.0, 65.0]  0.053577  22782.215205    62.417028
(65.0, 70.0]  0.037674  24295.421239    66.562798
```

[34]:

```
'''

plt.figure(figsize = (8, 8))

# Graph the age bins and the average of the target as a bar plot
plt.bar(age_groups.index.astype(str), 100 * age_groups['TARGET'])

# Plot labeling
plt.xticks(rotation = 75); plt.xlabel('Age Group (years)'); plt.ylabel('Failure␣
↪to Repay (%)')
plt.title('Failure to Repay by Age Group');

'''
```

[34]:

```
"\n\nplt.figure(figsize = (8, 8))\n\n# Graph the age bins and the average of the
target as a bar plot\nplt.bar(age_groups.index.astype(str), 100 *
age_groups['TARGET'])\n\n# Plot labeling\nplt.xticks(rotation = 75);
plt.xlabel('Age Group (years)'); plt.ylabel('Failure to Repay
```

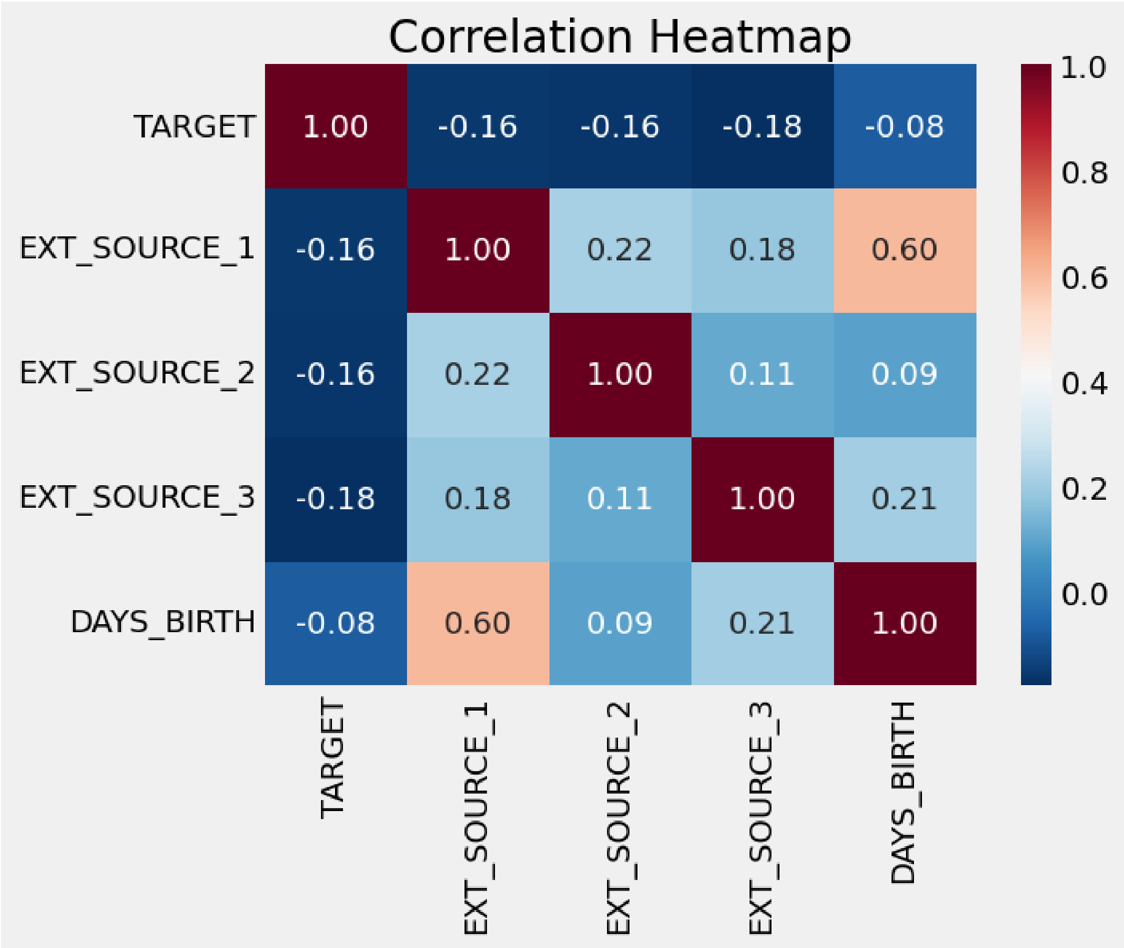
```
(%)')\nplt.title('Failure to Repay by Age Group');\n\n"
```

[35]:
```
# Extract the EXT_SOURCE variables and show correlations
ext_data = app_train[['TARGET', 'EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3',␣
↪'DAYS_BIRTH']]
ext_data_corrs = ext_data.corr()
ext_data_corrs
```

[35]:

	TARGET	EXT_SOURCE_1	EXT_SOURCE_2	EXT_SOURCE_3	DAYS_BIRTH
TARGET	1.000000	-0.155094	-0.162884	-0.176890	-0.079078
EXT_SOURCE_1	-0.155094	1.000000	0.215711	0.183186	0.602974
EXT_SOURCE_2	-0.162884	0.215711	1.000000	0.109469	0.092276
EXT_SOURCE_3	-0.176890	0.183186	0.109469	1.000000	0.209276
DAYS_BIRTH	-0.079078	0.602974	0.092276	0.209276	1.000000

[36]:
```
corr_matrix = ext_data_corrs
fig, ax = plt.subplots()
sns.heatmap(corr_matrix, annot=True, cmap='RdBu_r', ax=ax, fmt = "0.2f",)
ax.set_title('Correlation Heatmap')

plt.show()
```



[37]:
```
'''

plt.figure(figsize = (15, 17))

# iterate through the sources
for i, source in enumerate(['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3']):

    # create a new subplot for each source
    plt.subplot(3, 1, i + 1)
    # plot repaid loans
    sns.kdeplot(app_train.loc[app_train['TARGET'] == 0, source], label =␣
↪'target == 0')
    # plot loans that were not repaid
    sns.kdeplot(app_train.loc[app_train['TARGET'] == 1, source], label =␣
↪'target == 1')

    # Label the plots
    plt.title('Distribution of %s by Target Value' % source)
```

```
    plt.xlabel('%s' % source); plt.ylabel('Density');
    plt.legend()
    plt.show()

plt.tight_layout(h_pad = 2.5)

'''
```

[37]: "\n\nplt.figure(figsize = (15, 17))\n\n# iterate through the sources\nfor i, source in enumerate(['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3']):\n   \n # create a new subplot for each source\n   plt.subplot(3, 1, i + 1)\n   # plot repaid loans\n   sns.kdeplot(app_train.loc[app_train['TARGET'] == 0, source], label = 'target == 0')\n   # plot loans that were not repaid\n sns.kdeplot(app_train.loc[app_train['TARGET'] == 1, source], label = 'target == 1')\n   \n   # Label the plots\n   plt.title('Distribution of %s by Target Value' % source)\n   plt.xlabel('%s' % source); plt.ylabel('Density');\n plt.legend()\n   plt.show()\n\nplt.tight_layout(h_pad = 2.5)\n\n"

[38]:
```
ext_data.head(10)
```

[38]:

	TARGET	EXT_SOURCE_1	EXT_SOURCE_2	EXT_SOURCE_3	DAYS_BIRTH
0	1	0.083037	0.262949	0.139376	9461
1	0	0.311267	0.622246	NaN	16765
2	0	NaN	0.555912	0.729567	19046
3	0	NaN	0.650442	NaN	19005
4	0	NaN	0.322738	NaN	19932
5	0	NaN	0.354225	0.621226	16941
6	0	0.774761	0.724000	0.492060	13778
7	0	NaN	0.714279	0.540654	18850
8	0	0.587334	0.205747	0.751724	20099
9	0	NaN	0.746644	NaN	14469

[39]:
```
'''
# Copy the data for plotting
plot_data = ext_data.drop(columns = ['DAYS_BIRTH']).copy()

# Add in the age of the client in years
plot_data['YEARS_BIRTH'] = age_data['YEARS_BIRTH']

# Drop na values and limit to first 100000 rows
plot_data = plot_data.dropna().loc[:100000, :]

# Function to calculate correlation coefficient between two columns
def corr_func(x, y, **kwargs):
    r = np.corrcoef(x, y)[0][1]
    ax = plt.gca()
    ax.annotate("r = {:.2f}".format(r),
```

```
                 xy=(.2, .8), xycoords=ax.transAxes,
                 size = 20)

# Create the pairgrid object
grid = sns.PairGrid(data = plot_data, diag_sharey=False,
                    hue = 'TARGET',
                    vars = [x for x in list(plot_data.columns) if x !=␣
 ↪'TARGET'])

# Upper is a scatter plot
grid.map_upper(plt.scatter, alpha = 0.2)

# Diagonal is a histogram
grid.map_diag(sns.kdeplot)

#grid.map_diag(sns.kdeplot, data=app_train.loc[app_train['TARGET'] == 1,␣
 ↪source])

    # plot loans that were not repaid
# Bottom is density plot
grid.map_lower(sns.kdeplot, cmap = plt.cm.OrRd_r);

plt.suptitle('Ext Source and Age Features Pairs Plot', size = 32, y = 1.05);

'''
```

[39]:

```
'\n# Copy the data for plotting\nplot_data = ext_data.drop(columns =
[\'DAYS_BIRTH\']).copy()\n\n# Add in the age of the client in
years\nplot_data[\'YEARS_BIRTH\'] = age_data[\'YEARS_BIRTH\']\n\n# Drop na
values and limit to first 100000 rows\nplot_data =
plot_data.dropna().loc[:100000, :]\n\n# Function to calculate correlation
coefficient between two columns\ndef corr_func(x, y, **kwargs):\n    r =
np.corrcoef(x, y)[0][1]\n    ax = plt.gca()\n    ax.annotate("r =
{:.2f}".format(r),\n                xy=(.2, .8), xycoords=ax.transAxes,\n
size = 20)\n\n# Create the pairgrid object\ngrid = sns.PairGrid(data =
plot_data, diag_sharey=False,\n                    hue = \'TARGET\', \n
vars = [x for x in list(plot_data.columns) if x != \'TARGET\'])\n\n# Upper is a
scatter plot\ngrid.map_upper(plt.scatter, alpha = 0.2)\n\n# Diagonal is a
histogram\ngrid.map_diag(sns.kdeplot)\n\n#grid.map_diag(sns.kdeplot,
data=app_train.loc[app_train[\'TARGET\'] == 1, source])\n\n    # plot loans that
were not repaid\n# Bottom is density plot\ngrid.map_lower(sns.kdeplot, cmap =
plt.cm.OrRd_r);\n\nplt.suptitle(\'Ext Source and Age Features Pairs Plot\', size
= 32, y = 1.05);\n\n'
```

[40]:

```
#plot_data
```

[41]:
```python
# Make a new dataframe for polynomial features
poly_features = app_train[['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3',␣
↪'DAYS_BIRTH', 'TARGET']]
poly_features_test = app_test[['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3',␣
↪'DAYS_BIRTH']]

# imputer for handling missing values
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy = 'median')

poly_target = poly_features['TARGET']

poly_features = poly_features.drop(columns = ['TARGET'])

# Need to impute missing values
poly_features = imputer.fit_transform(poly_features)
poly_features_test = imputer.transform(poly_features_test)

from sklearn.preprocessing import PolynomialFeatures

# Create the polynomial object with specified degree
poly_transformer = PolynomialFeatures(degree = 3)
```

[42]:
```python
# Train the polynomial features
poly_transformer.fit(poly_features)

# Transform the features
poly_features = poly_transformer.transform(poly_features)
poly_features_test = poly_transformer.transform(poly_features_test)
print('Polynomial Features shape: ', poly_features.shape)
```

```
Polynomial Features shape:  (150000, 35)
```

[43]:
```python
poly_transformer.get_feature_names_out(input_features = ['EXT_SOURCE_1',␣
↪'EXT_SOURCE_2', 'EXT_SOURCE_3', 'DAYS_BIRTH'])[:15]
```

[43]:
```
array(['1', 'EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3', 'DAYS_BIRTH',
       'EXT_SOURCE_1^2', 'EXT_SOURCE_1 EXT_SOURCE_2',
       'EXT_SOURCE_1 EXT_SOURCE_3', 'EXT_SOURCE_1 DAYS_BIRTH',
       'EXT_SOURCE_2^2', 'EXT_SOURCE_2 EXT_SOURCE_3',
       'EXT_SOURCE_2 DAYS_BIRTH', 'EXT_SOURCE_3^2',
       'EXT_SOURCE_3 DAYS_BIRTH', 'DAYS_BIRTH^2'], dtype=object)
```

[44]:
```python
#There are 35 features with individual features raised to powers up to degree 3␣
↪and interaction terms. Now, we can see whether any of these new features are␣
↪correlated with the target.
```

[45]:
```
# Create a dataframe of the features
poly_features = pd.DataFrame(poly_features,
                             columns = poly_transformer.
 ↪get_feature_names_out(['EXT_SOURCE_1', 'EXT_SOURCE_2',
                                                                                   ␣
 ↪'EXT_SOURCE_3', 'DAYS_BIRTH']))

# Add in the target
poly_features['TARGET'] = poly_target

# Find the correlations with the target
poly_corrs = poly_features.corr()['TARGET'].sort_values()

# Display most negative and most positive
print(poly_corrs.head(10))
print(poly_corrs.tail(5))
```

```
EXT_SOURCE_2 EXT_SOURCE_3                -0.194612
EXT_SOURCE_1 EXT_SOURCE_2 EXT_SOURCE_3   -0.189549
EXT_SOURCE_2 EXT_SOURCE_3 DAYS_BIRTH     -0.181839
EXT_SOURCE_2^2 EXT_SOURCE_3              -0.176669
EXT_SOURCE_2 EXT_SOURCE_3^2              -0.172417
EXT_SOURCE_1 EXT_SOURCE_2                -0.167382
EXT_SOURCE_2                             -0.162728
EXT_SOURCE_1 EXT_SOURCE_3                -0.162481
EXT_SOURCE_2 DAYS_BIRTH                  -0.158770
EXT_SOURCE_1 EXT_SOURCE_2^2              -0.156893
Name: TARGET, dtype: float64
DAYS_BIRTH     -0.079078
DAYS_BIRTH^2   -0.077616
DAYS_BIRTH^3   -0.075257
TARGET          1.000000
1                    NaN
Name: TARGET, dtype: float64
```

[46]:
```
# Put test features into dataframe
poly_features_test = pd.DataFrame(poly_features_test,
                                  columns = poly_transformer.
 ↪get_feature_names_out(['EXT_SOURCE_1', 'EXT_SOURCE_2',
                                                                                        ␣
 ↪'EXT_SOURCE_3', 'DAYS_BIRTH']))

# Merge polynomial features into training dataframe
poly_features['SK_ID_CURR'] = app_train['SK_ID_CURR']
app_train_poly = app_train.merge(poly_features, on = 'SK_ID_CURR', how = 'left')

# Merge polnomial features into testing dataframe
```

```
poly_features_test['SK_ID_CURR'] = app_test['SK_ID_CURR']
app_test_poly = app_test.merge(poly_features_test, on = 'SK_ID_CURR', how =␣
 ↪'left')

# Align the dataframes
app_train_poly, app_test_poly = app_train_poly.align(app_test_poly, join =␣
 ↪'inner', axis = 1)

# Print out the new shapes
print('Training data with polynomial features shape: ', app_train_poly.shape)
print('Testing data with polynomial features shape:  ', app_test_poly.shape)
```

```
Training data with polynomial features shape:  (150000, 275)
Testing data with polynomial features shape:   (48744, 275)
```

[47]:

```
#Create these features
#CREDIT_INCOME_PERCENT: the percentage of the credit amount relative to a␣
 ↪client's income
#ANNUITY_INCOME_PERCENT: the percentage of the loan annuity relative to a␣
 ↪client's income
#CREDIT_TERM: the length of the payment in months (since the annuity is the␣
 ↪monthly amount due
#DAYS_EMPLOYED_PERCENT: the percentage of the days employed relative to the␣
 ↪client's age
```

[48]:

```
app_train_domain = app_train.copy()
app_test_domain = app_test.copy()

app_train_domain['CREDIT_INCOME_PERCENT'] = app_train_domain['AMT_CREDIT'] /␣
 ↪app_train_domain['AMT_INCOME_TOTAL']
app_train_domain['ANNUITY_INCOME_PERCENT'] = app_train_domain['AMT_ANNUITY'] /␣
 ↪app_train_domain['AMT_INCOME_TOTAL']
app_train_domain['CREDIT_TERM'] = app_train_domain['AMT_ANNUITY'] /␣
 ↪app_train_domain['AMT_CREDIT']
app_train_domain['DAYS_EMPLOYED_PERCENT'] = app_train_domain['DAYS_EMPLOYED'] /␣
 ↪app_train_domain['DAYS_BIRTH']
```

[49]:

```
app_test_domain['CREDIT_INCOME_PERCENT'] = app_test_domain['AMT_CREDIT'] /␣
 ↪app_test_domain['AMT_INCOME_TOTAL']
app_test_domain['ANNUITY_INCOME_PERCENT'] = app_test_domain['AMT_ANNUITY'] /␣
 ↪app_test_domain['AMT_INCOME_TOTAL']
app_test_domain['CREDIT_TERM'] = app_test_domain['AMT_ANNUITY'] /␣
 ↪app_test_domain['AMT_CREDIT']
app_test_domain['DAYS_EMPLOYED_PERCENT'] = app_test_domain['DAYS_EMPLOYED'] /␣
 ↪app_test_domain['DAYS_BIRTH']
```

[50]:
```
'''
#Visualize New Variables¶
plt.figure(figsize = (12, 20))
# iterate through the new features
for i, feature in enumerate(['CREDIT_INCOME_PERCENT', 'ANNUITY_INCOME_PERCENT',␣
↪'CREDIT_TERM', 'DAYS_EMPLOYED_PERCENT']):

    # create a new subplot for each source
    plt.subplot(4, 1, i + 1)
    # plot repaid loans
    sns.kdeplot(app_train_domain.loc[app_train_domain['TARGET'] == 0, feature],␣
↪label = 'target == 0')
    # plot loans that were not repaid
    sns.kdeplot(app_train_domain.loc[app_train_domain['TARGET'] == 1, feature],␣
↪label = 'target == 1')

    # Label the plots
    plt.title('Distribution of %s by Target Value' % feature)
    plt.xlabel('%s' % feature); plt.ylabel('Density');
    plt.legend()
plt.tight_layout(h_pad = 2.5)

'''
```

[50]:
```
"\n#Visualize New Variables¶\nplt.figure(figsize = (12, 20))\n# iterate through
the new features\nfor i, feature in enumerate(['CREDIT_INCOME_PERCENT',
'ANNUITY_INCOME_PERCENT', 'CREDIT_TERM', 'DAYS_EMPLOYED_PERCENT']):\n    \n    #
create a new subplot for each source\n    plt.subplot(4, 1, i + 1)\n    # plot
repaid loans\n    sns.kdeplot(app_train_domain.loc[app_train_domain['TARGET'] ==
0, feature], label = 'target == 0')\n    # plot loans that were not repaid\n
sns.kdeplot(app_train_domain.loc[app_train_domain['TARGET'] == 1, feature],
label = 'target == 1')\n    \n    # Label the plots\n    plt.title('Distribution
of %s by Target Value' % feature)\n    plt.xlabel('%s' % feature);
plt.ylabel('Density');\n    plt.legend()\nplt.tight_layout(h_pad = 2.5)\n\n"
```

[51]:
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.impute import SimpleImputer

# Drop the target from the training data
if 'TARGET' in app_train:
    train = app_train.drop(columns = ['TARGET'])
else:
    train = app_train.copy()

# Feature names
features = list(train.columns)
```

```
# Copy of the testing data
test = app_test.copy()

# Median imputation of missing values
imputer = SimpleImputer(strategy = 'median')

# Scale each feature to 0-1
scaler = MinMaxScaler(feature_range = (0, 1))

# Fit on the training data
imputer.fit(train)

# Transform both training and testing data
train = imputer.transform(train)
test = imputer.transform(app_test)

# Repeat with the scaler
scaler.fit(train)
train = scaler.transform(train)
test = scaler.transform(test)

print('Training data shape: ', train.shape)
print('Testing data shape: ', test.shape)
```

```
Training data shape:  (150000, 240)
Testing data shape:  (48744, 240)
```

[52]:
```
from sklearn.linear_model import LogisticRegression

# Make the model with the specified regularization parameter
log_reg = LogisticRegression(C = 0.0001)

# Train on the training data
log_reg.fit(train, train_labels)
```

[52]:
```
LogisticRegression(C=0.0001)
```

[58]:
```
# Make predictions
# Make sure to select the second column only
log_reg_pred = log_reg.predict_proba(test)[:, 1]
```

[60]:
```
# Submission dataframe
submit = app_test[['SK_ID_CURR']]
submit['TARGET'] = log_reg_pred

submit.head()
```

[60]:
	SK_ID_CURR	TARGET
0	100001	0.078397
1	100005	0.115970
2	100013	0.085067
3	100028	0.080295
4	100038	0.116831

[61]:
```
# Save the submission to a csv file
submit.to_csv('log_reg_baseline.csv', index = False)
#this model has a 0.67 accuracy score.
```

[62]:
```
#using random forest to predict
```

[63]:
```
from sklearn.ensemble import RandomForestClassifier

# Make the random forest classifier
random_forest = RandomForestClassifier(n_estimators = 100, random_state = 50,␣
 ↪verbose = 1, n_jobs = -1)
```

[64]:
```
# Train on the training data
random_forest.fit(train, train_labels)

# Extract feature importances
feature_importance_values = random_forest.feature_importances_
feature_importances = pd.DataFrame({'feature': features, 'importance':␣
 ↪feature_importance_values})

# Make predictions on the test data
predictions = random_forest.predict_proba(test)[:, 1]
```

```
[Parallel(n_jobs=-1)]: Using backend SequentialBackend with 1 concurrent
workers.
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:  1.1min finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    1.6s finished
```

[67]:
```
# Make a submission dataframe
submit = app_test[['SK_ID_CURR']]
submit['TARGET'] = predictions

# Save the submission dataframe
submit.to_csv('random_forest_baseline.csv', index = False)
```

[68]:
```
#Make Predictions using Engineered Features
```

[70]:
```
poly_features_names = list(app_train_poly.columns)
```

```python
# Impute the polynomial features
imputer = SimpleImputer(strategy = 'median')

poly_features = imputer.fit_transform(app_train_poly)
poly_features_test = imputer.transform(app_test_poly)

# Scale the polynomial features
scaler = MinMaxScaler(feature_range = (0, 1))

poly_features = scaler.fit_transform(poly_features)
poly_features_test = scaler.transform(poly_features_test)

random_forest_poly = RandomForestClassifier(n_estimators = 100, random_state =␣
↪50, verbose = 1, n_jobs = -1)
```

[71]:
```python
# Train on the training data
random_forest_poly.fit(poly_features, train_labels)

# Make predictions on the test data
predictions = random_forest_poly.predict_proba(poly_features_test)[:, 1]
```

```
[Parallel(n_jobs=-1)]: Using backend SequentialBackend with 1 concurrent
workers.
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:  1.9min finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.9s finished
```

[72]:
```python
# Make a submission dataframe
submit = app_test[['SK_ID_CURR']]
submit['TARGET'] = predictions

# Save the submission dataframe
submit.to_csv('random_forest_baseline_engineered.csv', index = False)
```

[73]:
```python
#This model scored 0.678 when submitted to the competition, exactly the same as␣
↪that without the engineered features. Given these results, it does not␣
↪appear that our feature construction helped in this case.
```

[75]:
```python
#Testing Domain Features¶
#Now we can test the domain features we made by hand.
```

[78]:
```python
if 'TARGET' in app_train_domain:
    app_train_domain = app_train_domain.drop(columns = 'TARGET')
else:
    app_train_domain = app_train_domain.copy()
```

```
domain_features_names = list(app_train_domain.columns)

# Impute the domainnomial features
imputer = SimpleImputer(strategy = 'median')

domain_features = imputer.fit_transform(app_train_domain)
domain_features_test = imputer.transform(app_test_domain)

# Scale the domainnomial features
scaler = MinMaxScaler(feature_range = (0, 1))

domain_features = scaler.fit_transform(domain_features)
domain_features_test = scaler.transform(domain_features_test)

random_forest_domain = RandomForestClassifier(n_estimators = 100, random_state␣
↪= 50, verbose = 1, n_jobs = -1)

# Train on the training data
random_forest_domain.fit(domain_features, train_labels)

# Extract feature importances
feature_importance_values_domain = random_forest_domain.feature_importances_
feature_importances_domain = pd.DataFrame({'feature': domain_features_names,␣
↪'importance': feature_importance_values_domain})

# Make predictions on the test data
predictions = random_forest_domain.predict_proba(domain_features_test)[:, 1]
```

```
[Parallel(n_jobs=-1)]: Using backend SequentialBackend with 1 concurrent
workers.
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:  1.2min finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    1.5s finished
```

[79]:
```
# Make a submission dataframe
submit = app_test[['SK_ID_CURR']]
submit['TARGET'] = predictions

# Save the submission dataframe
submit.to_csv('random_forest_baseline_domain.csv', index = False)
```