In [60]:
```
# Let's start with importing numpy and pandas to manipulate our dataset
import numpy as np
import pandas as pd

# The same with sklearn processing in order to deal with with categorical vari
ables
from sklearn.preprocessing import LabelEncoder

# File system manangement
import os

# Exceeding warnings suppressed
import warnings
warnings.filterwarnings('ignore')

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

In [61]:
```
# Listing the files available
print(os.listdir("C:/Users/Dejvid/Desktop/Lab/home-credit-default-risk/"))
```

```
['application_test.csv', 'application_train.csv', 'bureau.csv', 'bureau_balan
ce.csv', 'credit_card_balance.csv', 'HomeCredit_columns_description.csv', 'in
stallments_payments.csv', 'POS_CASH_balance.csv', 'previous_application.csv',
'results.csv', 'sample_submission.csv']
```

In [62]:
```
# Approaching the training dataset shape
app_train = pd.read_csv('C:/Users/Dejvid/Desktop/Lab/home-credit-default-risk/
application_train.csv')
print('Training data shape: ', app_train.shape)
app_train.head()
```

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

Out[62]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_O
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	

5 rows × 122 columns

In [63]:
```
# Testing the data features
app_test = pd.read_csv('C:/Users/Dejvid/Desktop/Lab/home-credit-default-risk/application_test.csv')
print('Testing data shape: ', app_test.shape)
app_test.head()
```

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

Out[63]:

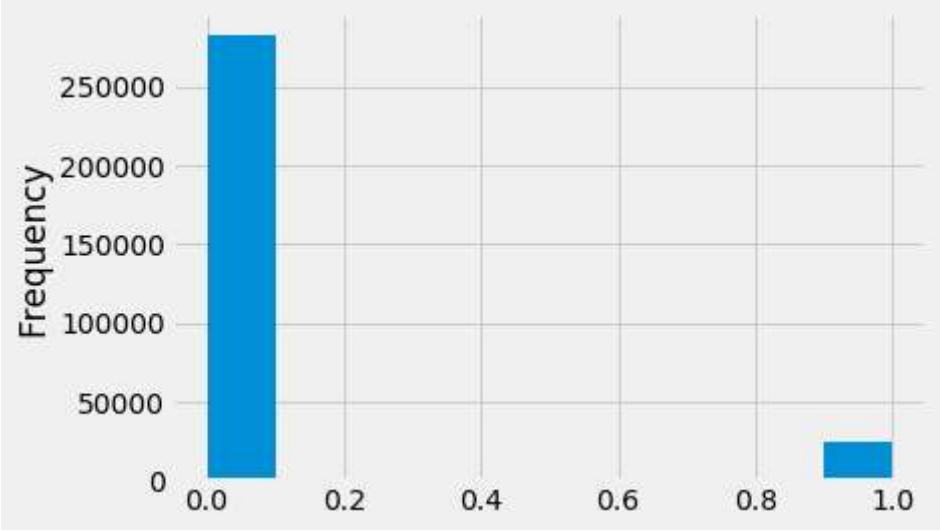
	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALT
0	100001	Cash loans	F	N	
1	100005	Cash loans	M	N	
2	100013	Cash loans	M	Y	
3	100028	Cash loans	F	N	
4	100038	Cash loans	M	Y	

5 rows × 121 columns

In [64]:
```
app_train['TARGET'].value_counts()
```

Out[64]:
```
0    282686
1     24825
Name: TARGET, dtype: int64
```

In [65]:
```
app_train['TARGET'].astype(int).plot.hist();
```



In [66]:
```
# Creting the function to calculate the missing values in the columns
def missing_values_table(df):
        # To find the total missing values
        mis_val = df.isnull().sum()

        # Convert it in a percentage
        mis_val_percent = 100 * df.isnull().sum() / len(df)

        # Make a table with the results obtained
        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 obtained according to the percentage of missing desce
nding
        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)

        # Get summary information of the results with a print
        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.")

        # Finally return the dataframe coded with missing information
        return mis_val_table_ren_columns
```

In [67]:
```
# Knit a visual of the most missing values statisticsin order to decide how to continue
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.
```

Out[67]:

	Missing Values	% of Total Values
COMMONAREA_MEDI	214865	69.9
COMMONAREA_AVG	214865	69.9
COMMONAREA_MODE	214865	69.9
NONLIVINGAPARTMENTS_MEDI	213514	69.4
NONLIVINGAPARTMENTS_MODE	213514	69.4
NONLIVINGAPARTMENTS_AVG	213514	69.4
FONDKAPREMONT_MODE	210295	68.4
LIVINGAPARTMENTS_MODE	210199	68.4
LIVINGAPARTMENTS_MEDI	210199	68.4
LIVINGAPARTMENTS_AVG	210199	68.4
FLOORSMIN_MODE	208642	67.8
FLOORSMIN_MEDI	208642	67.8
FLOORSMIN_AVG	208642	67.8
YEARS_BUILD_MODE	204488	66.5
YEARS_BUILD_MEDI	204488	66.5
YEARS_BUILD_AVG	204488	66.5
OWN_CAR_AGE	202929	66.0
LANDAREA_AVG	182590	59.4
LANDAREA_MEDI	182590	59.4
LANDAREA_MODE	182590	59.4

In [68]:
```
# Analysing the number of each type of column (floats, int, ect)
app_train.dtypes.value_counts()
```

Out[68]:
```
float64    65
int64      41
object     16
dtype: int64
```

In [69]:
```
# Spotlight the number of unique classes in each object column
app_train.select_dtypes('object').apply(pd.Series.nunique, axis = 0)
```

Out[69]:
```
NAME_CONTRACT_TYPE              2
CODE_GENDER                     3
FLAG_OWN_CAR                    2
FLAG_OWN_REALTY                 2
NAME_TYPE_SUITE                 7
NAME_INCOME_TYPE                8
NAME_EDUCATION_TYPE             5
NAME_FAMILY_STATUS              6
NAME_HOUSING_TYPE               6
OCCUPATION_TYPE                18
WEEKDAY_APPR_PROCESS_START      7
ORGANIZATION_TYPE              58
FONDKAPREMONT_MODE              4
HOUSETYPE_MODE                  3
WALLSMATERIAL_MODE              7
EMERGENCYSTATE_MODE             2
dtype: int64
```

In [70]:
```
# Create a label encoder object and counter
le = LabelEncoder()
le_count = 0

# define a loop to iterate through the columns
for col in app_train:
    if app_train[col].dtype == 'object':
        # The condition stands for "If 2 or fewer unique categories"
        if len(list(app_train[col].unique())) <= 2:
            # Make it approach 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])

            # Just to count how many columns were label encoded
            le_count += 1

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

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

In [71]:
```
# Set the one-hot encoding of categorical variables in order to facilitate the
processing of the prediction
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:  (307511, 243)
Testing Features shape:  (48744, 239)
```

In [72]:
```
train_labels = app_train['TARGET']

# Align the training and testing data together, but 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:  (307511, 240)
Testing Features shape:  (48744, 239)
```

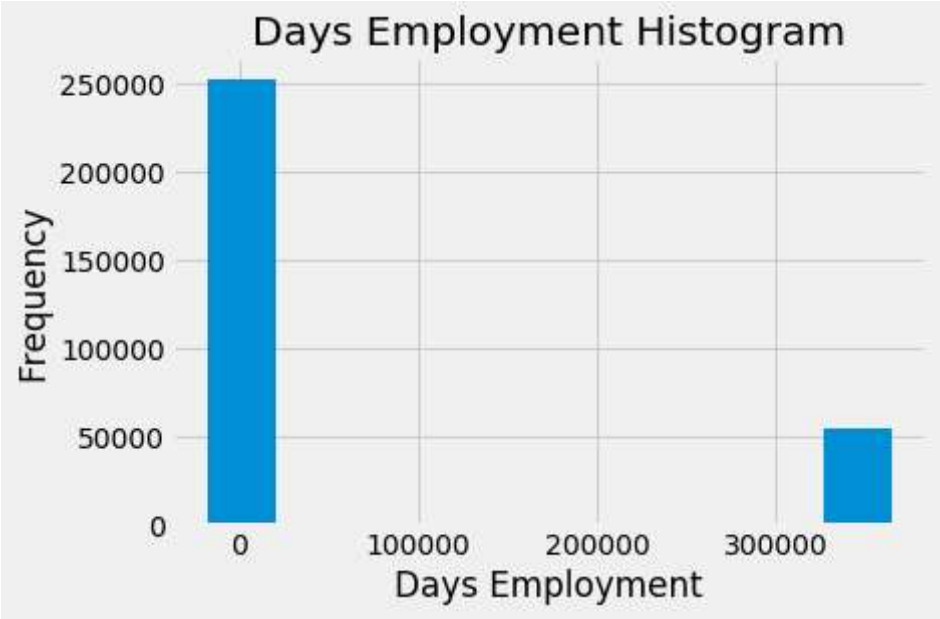
In [73]:
```
(app_train['DAYS_BIRTH'] / -365).describe()
```

Out[73]:
```
count    307511.000000
mean         43.936973
std          11.956133
min          20.517808
25%          34.008219
50%          43.150685
75%          53.923288
max          69.120548
Name: DAYS_BIRTH, dtype: float64
```

In [74]:
```
app_train['DAYS_EMPLOYED'].describe()
```

Out[74]:
```
count    307511.000000
mean      63815.045904
std      141275.766519
min      -17912.000000
25%       -2760.000000
50%       -1213.000000
75%        -289.000000
max      365243.000000
Name: DAYS_EMPLOYED, dtype: float64
```

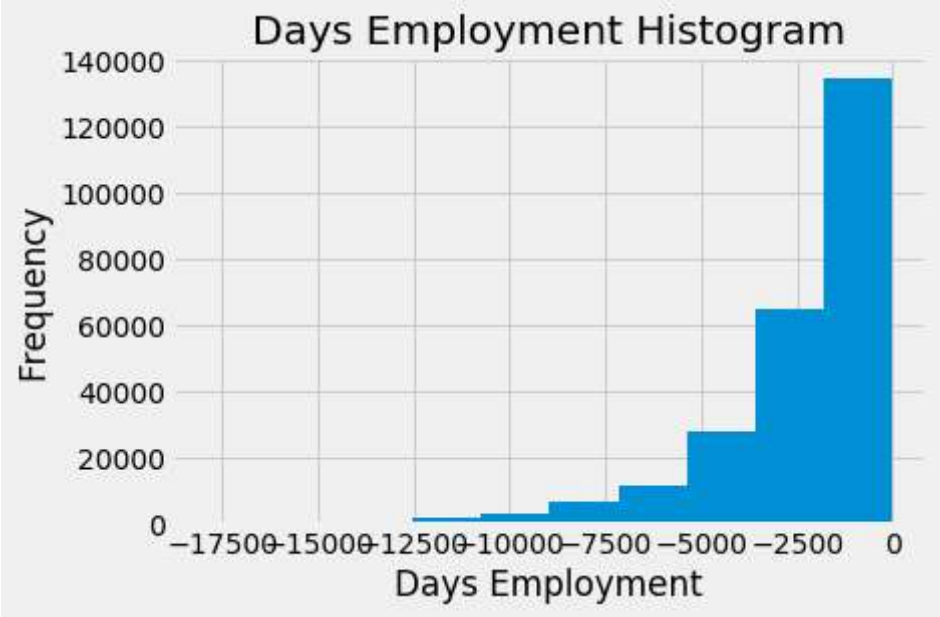
In [75]:
```
app_train['DAYS_EMPLOYED'].plot.hist(title = 'Days Employment Histogram');
plt.xlabel('Days Employment');
```



In [76]:
```
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['TARGE
T'].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 8.66% of loans
The anomalies default on 5.40% of loans
There are 55374 anomalous days of employment
```

In [77]:
```
# In order to substitute the anomalous values,create an anomalous flag column
app_train['DAYS_EMPLOYED_ANOM'] = app_train["DAYS_EMPLOYED"] == 365243

# Then 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');
```



In [78]:
```
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
```

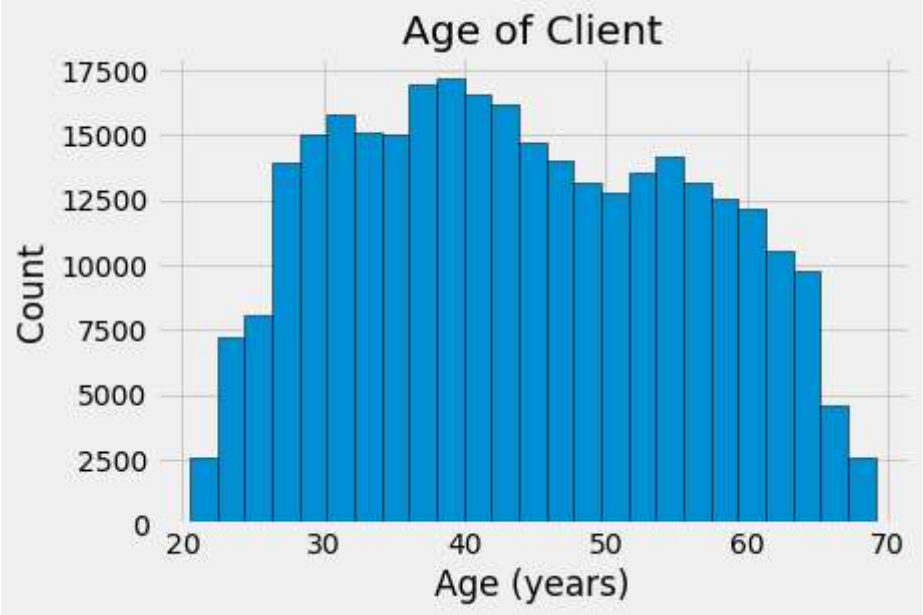
In [80]:
```
#Here we use .corr to find the correlation with the target and we sorted them
correlations = app_train.corr()['TARGET'].sort_values()

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

```
Most Positive Correlations:
 OCCUPATION_TYPE_Laborers                              0.043019
FLAG_DOCUMENT_3                                        0.044346
REG_CITY_NOT_LIVE_CITY                                 0.044395
FLAG_EMP_PHONE                                         0.045982
NAME_EDUCATION_TYPE_Secondary / secondary special      0.049824
REG_CITY_NOT_WORK_CITY                                 0.050994
DAYS_ID_PUBLISH                                        0.051457
CODE_GENDER_M                                          0.054713
DAYS_LAST_PHONE_CHANGE                                 0.055218
NAME_INCOME_TYPE_Working                               0.057481
REGION_RATING_CLIENT                                   0.058899
REGION_RATING_CLIENT_W_CITY                            0.060893
DAYS_EMPLOYED                                          0.074958
DAYS_BIRTH                                             0.078239
TARGET                                                 1.000000
Name: TARGET, dtype: float64

Most Negative Correlations:
 EXT_SOURCE_3                            -0.178919
EXT_SOURCE_2                            -0.160472
EXT_SOURCE_1                            -0.155317
NAME_EDUCATION_TYPE_Higher education    -0.056593
CODE_GENDER_F                           -0.054704
NAME_INCOME_TYPE_Pensioner              -0.046209
DAYS_EMPLOYED_ANOM                      -0.045987
ORGANIZATION_TYPE_XNA                   -0.045987
FLOORSMAX_AVG                           -0.044003
FLOORSMAX_MEDI                          -0.043768
FLOORSMAX_MODE                          -0.043226
EMERGENCYSTATE_MODE_No                  -0.042201
HOUSETYPE_MODE_block of flats           -0.040594
AMT_GOODS_PRICE                         -0.039645
REGION_POPULATION_RELATIVE              -0.037227
Name: TARGET, dtype: float64
```

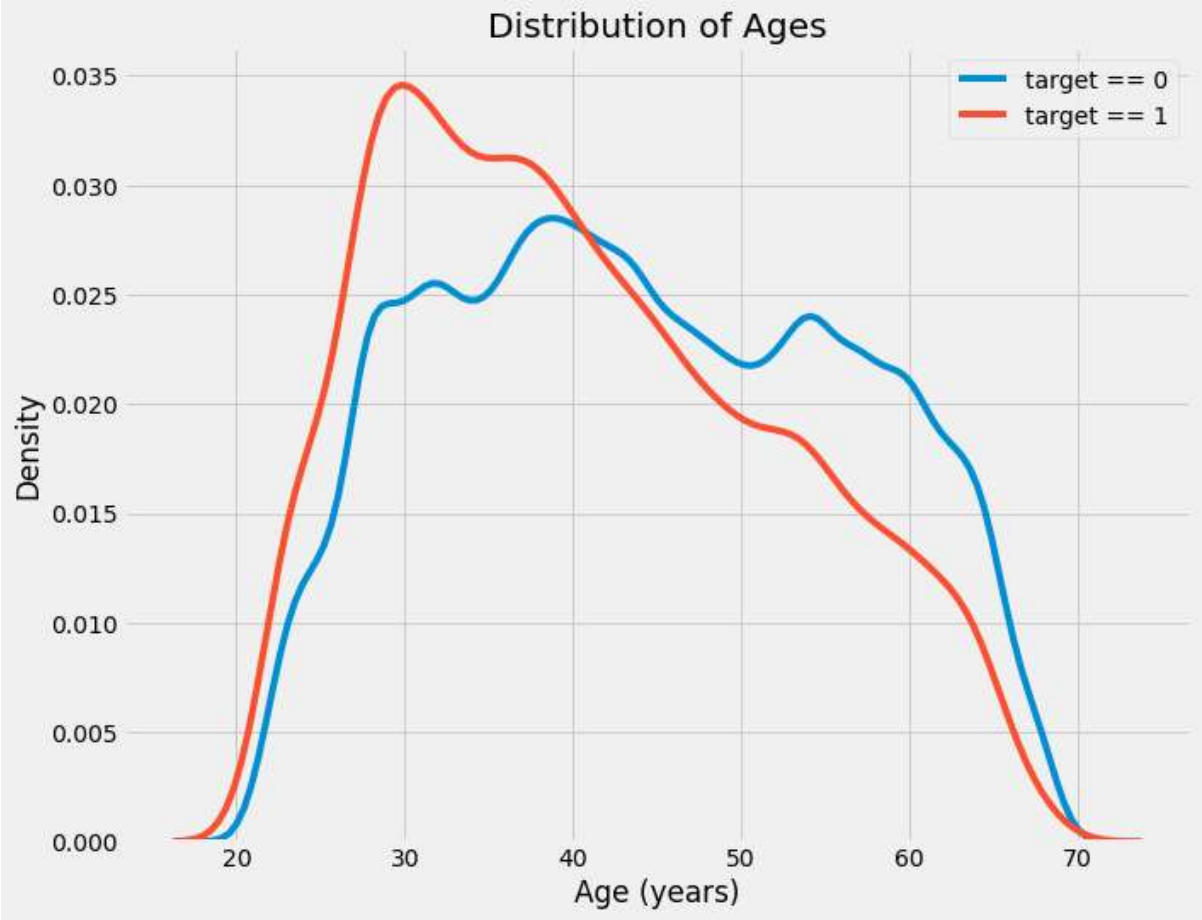
In [81]:
```
#we find the correlation of the positive days since birth and target
app_train['DAYS_BIRTH'] = abs(app_train['DAYS_BIRTH'])
app_train['DAYS_BIRTH'].corr(app_train['TARGET'])
```

Out[81]:
```
-0.07823930830982712
```

In [82]:
```
#Here we want to do a plot to see the distribution of ages in year
#we set the style of the plot
plt.style.use('fivethirtyeight')

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



In [83]:
```
#Now we make a more clear plot with KDE

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 A
ges');
```



In [84]:
```
#Here we put Age information into a separate dataframe to bin the age data
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)
```

Out[84]:

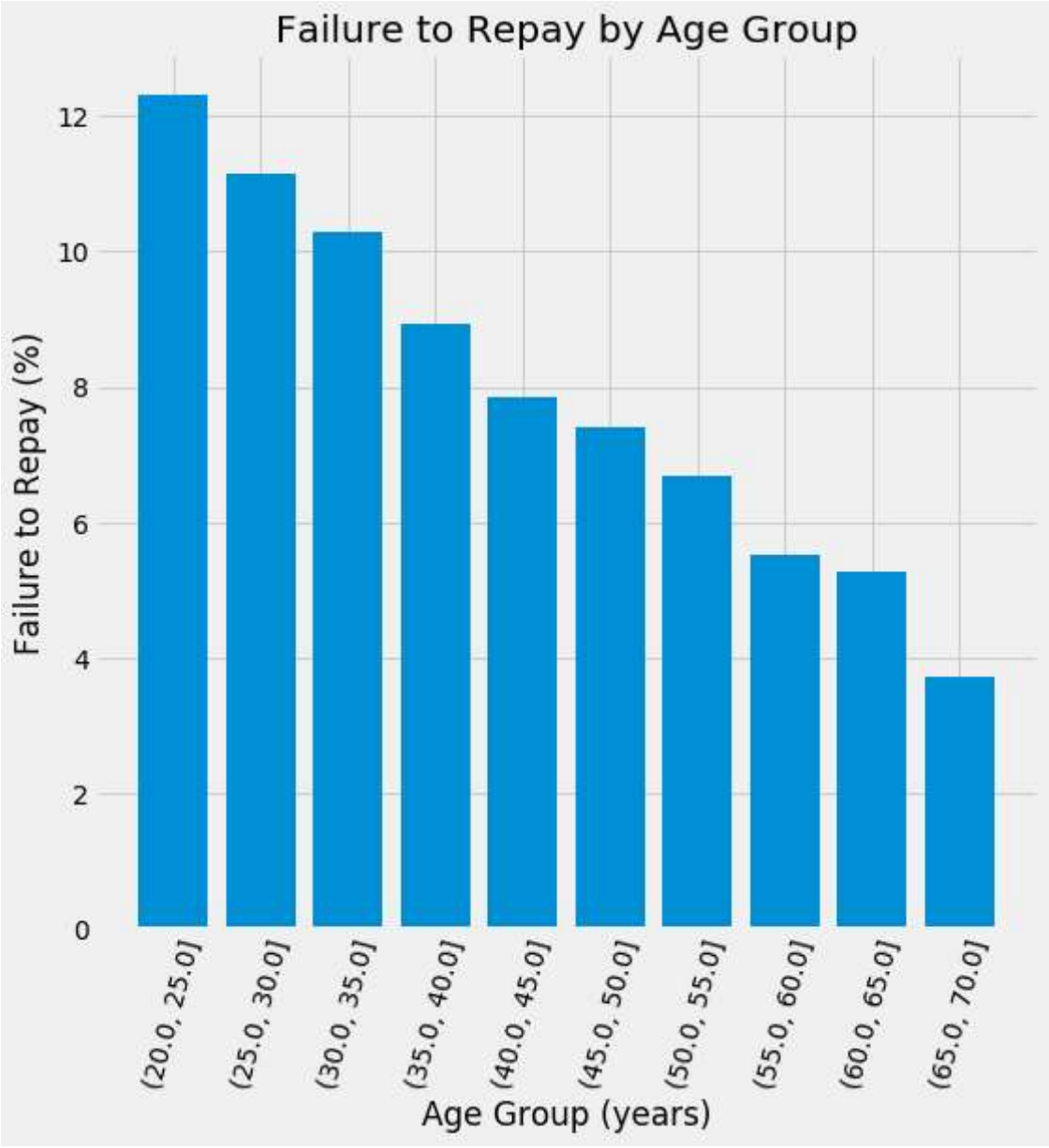
	TARGET	DAYS_BIRTH	YEARS_BIRTH	YEARS_BINNED
0	1	9461	25.920548	(25.0, 30.0]
1	0	16765	45.931507	(45.0, 50.0]
2	0	19046	52.180822	(50.0, 55.0]
3	0	19005	52.068493	(50.0, 55.0]
4	0	19932	54.608219	(50.0, 55.0]
5	0	16941	46.413699	(45.0, 50.0]
6	0	13778	37.747945	(35.0, 40.0]
7	0	18850	51.643836	(50.0, 55.0]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

In [85]:
```
#Now we group by the bin and calculate averages
age_groups  = age_data.groupby('YEARS_BINNED').mean()
age_groups
```

Out[85]:

YEARS_BINNED	TARGET	DAYS_BIRTH	YEARS_BIRTH
(20.0, 25.0]	0.123036	8532.795625	23.377522
(25.0, 30.0]	0.111436	10155.219250	27.822518
(30.0, 35.0]	0.102814	11854.848377	32.479037
(35.0, 40.0]	0.089414	13707.908253	37.555913
(40.0, 45.0]	0.078491	15497.661233	42.459346
(45.0, 50.0]	0.074171	17323.900441	47.462741
(50.0, 55.0]	0.066968	19196.494791	52.593136
(55.0, 60.0]	0.055314	20984.262742	57.491131
(60.0, 65.0]	0.052737	22780.547460	62.412459
(65.0, 70.0]	0.037270	24292.614340	66.555108

In [86]:
```
#And we graph the age bins and the average of the target as a bar plot
plt.figure(figsize = (8, 8))
plt.bar(age_groups.index.astype(str), 100 * age_groups['TARGET'])

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

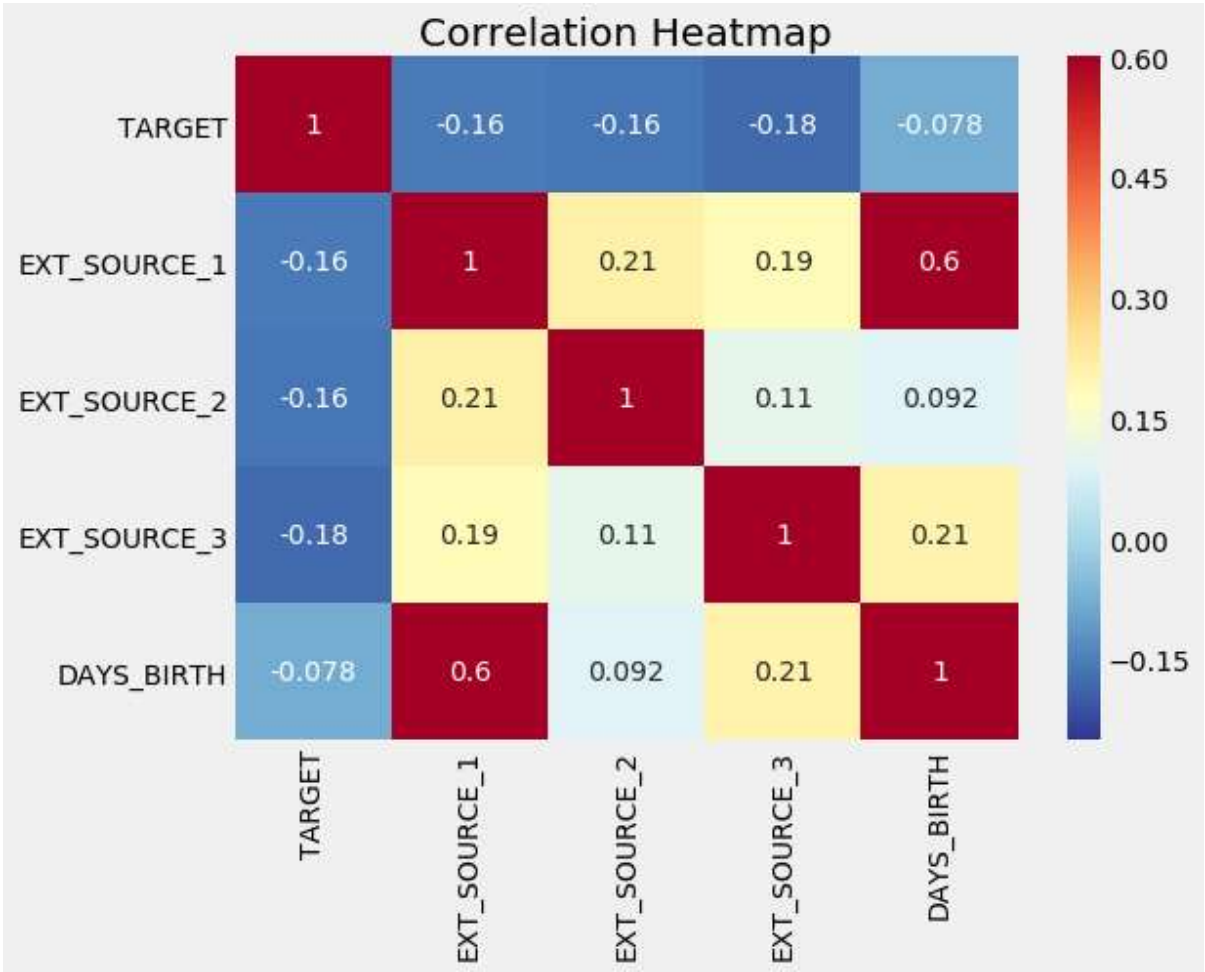


In [87]:
```
#Now we focus on the EXT_SOURCE variables we extract them and show correlation
s
ext_data = app_train[['TARGET', 'EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3'
, 'DAYS_BIRTH']]
ext_data_corrs = ext_data.corr()
ext_data_corrs
```

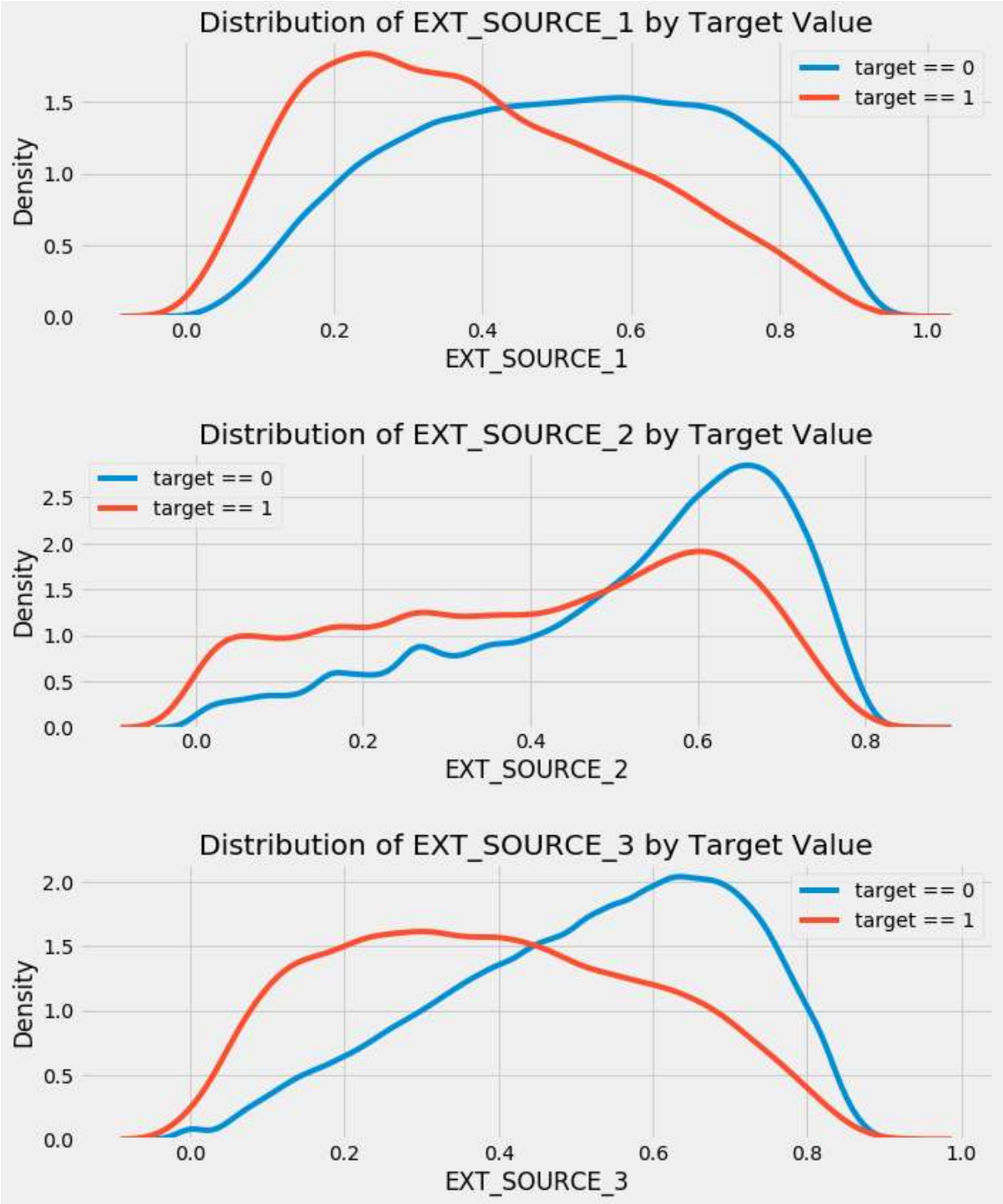
Out[87]:

	TARGET	EXT_SOURCE_1	EXT_SOURCE_2	EXT_SOURCE_3	DAYS_BIRTH
TARGET	1.000000	-0.155317	-0.160472	-0.178919	-0.078239
EXT_SOURCE_1	-0.155317	1.000000	0.213982	0.186846	0.600610
EXT_SOURCE_2	-0.160472	0.213982	1.000000	0.109167	0.091996
EXT_SOURCE_3	-0.178919	0.186846	0.109167	1.000000	0.205478
DAYS_BIRTH	-0.078239	0.600610	0.091996	0.205478	1.000000

In [88]:
```
#Heatmap of correlations
plt.figure(figsize = (8, 6))
sns.heatmap(ext_data_corrs, cmap = plt.cm.RdYlBu_r, vmin = -0.25, annot = True
, vmax = 0.6)
plt.title('Correlation Heatmap');
```



In [89]:
```
#And we make also a plot to see better the distribution of each EXT_SOURCE by
 Target Value
plt.figure(figsize = (10, 12))

#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 = 'targ
et == 0')
    #plot loans that were not repaid
    sns.kdeplot(app_train.loc[app_train['TARGET'] == 1, source], label = 'targ
et == 1')

    #Label the plots
    plt.title('Distribution of %s by Target Value' % source)
    plt.xlabel('%s' % source); plt.ylabel('Density');

plt.tight_layout(h_pad = 2.5)
```


Distribution of EXT_SOURCE_1 by Target Value
target == 0
target == 1
Density
EXT_SOURCE_1
Distribution of EXT_SOURCE_2 by Target Value
target == 0
target == 1
Density
EXT_SOURCE_2
Distribution of EXT_SOURCE_3 by Target Value
target == 0
target == 1
Density
EXT_SOURCE_3

In [90]:
```
#we make a new dataframe for polynomial features
poly_features = app_train[['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3', 'DA
YS_BIRTH', 'TARGET']]
poly_features_test = app_test[['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3',
'DAYS_BIRTH']]

#we use imputer for handling missing values
from sklearn.preprocessing import Imputer
imputer = Imputer(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)
```

In [91]:
```
#Than we train the polynomial features
poly_transformer.fit(poly_features)

#and 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:  (307511, 35)
```

In [92]:
```
poly_transformer.get_feature_names(input_features = ['EXT_SOURCE_1', 'EXT_SOUR
CE_2', 'EXT_SOURCE_3', 'DAYS_BIRTH'])[:15]
```

Out[92]:
```
['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']
```

In [93]:
```
#Create a dataframe of the features
poly_features = pd.DataFrame(poly_features,
                             columns = poly_transformer.get_feature_names(['EX
T_SOURCE_1', 'EXT_SOURCE_2',
                                                                           'EX
T_SOURCE_3', 'DAYS_BIRTH']))

#than we add in the Target
poly_features['TARGET'] = poly_target

#to 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                -0.193939
EXT_SOURCE_1 EXT_SOURCE_2 EXT_SOURCE_3   -0.189605
EXT_SOURCE_2 EXT_SOURCE_3 DAYS_BIRTH     -0.181283
EXT_SOURCE_2^2 EXT_SOURCE_3              -0.176428
EXT_SOURCE_2 EXT_SOURCE_3^2              -0.172282
EXT_SOURCE_1 EXT_SOURCE_2                -0.166625
EXT_SOURCE_1 EXT_SOURCE_3                -0.164065
EXT_SOURCE_2                             -0.160295
EXT_SOURCE_2 DAYS_BIRTH                  -0.156873
EXT_SOURCE_1 EXT_SOURCE_2^2              -0.156867
Name: TARGET, dtype: float64
DAYS_BIRTH     -0.078239
DAYS_BIRTH^2   -0.076672
DAYS_BIRTH^3   -0.074273
TARGET          1.000000
1                    NaN
Name: TARGET, dtype: float64
```

In [94]:
```
#Than we put test features into dataframe
poly_features_test = pd.DataFrame(poly_features_test,
                                  columns = poly_transformer.get_feature_names
(['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 = 'lef
t')

#Merge polinomial 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 = 'l
eft')

#Align the dataframes
app_train_poly, app_test_poly = app_train_poly.align(app_test_poly, join = 'in
ner', 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:  (307511, 275)
Testing data with polynomial features shape:   (48744, 275)
```

In [95]:
```
#we do the same with the polynomual features on less correlated features
#Make a new dataframe for polynomial features
poly_features1 = app_train[['TARGET','REGION_POPULATION_RELATIVE', 'AMT_GOODS_
PRICE', 'HOUSETYPE_MODE_block of flats', 'OCCUPATION_TYPE_Laborers', 'REG_CITY
_NOT_LIVE_CITY']]
poly_features_test1 = app_test[['REGION_POPULATION_RELATIVE', 'AMT_GOODS_PRIC
E', 'HOUSETYPE_MODE_block of flats', 'OCCUPATION_TYPE_Laborers', 'REG_CITY_NOT
_LIVE_CITY']]

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

poly_target1 = poly_features1['TARGET']

poly_features1.drop(['TARGET'],axis=1,inplace=True)

#Need to impute missing values
poly_features1 = imputer.fit_transform(poly_features1)
poly_features_test1 = imputer.transform(poly_features_test1)

from sklearn.preprocessing import PolynomialFeatures

#Create the polynomial object with specified degree
poly_transformer1 = PolynomialFeatures(degree = 3)
```

In [96]:
```
#Train the polynomial features
poly_transformer1.fit(poly_features1)

#Transform the features
poly_features1 = poly_transformer1.transform(poly_features1)
poly_features_test1 = poly_transformer1.transform(poly_features_test1)
print('Polynomial Features shape: ', poly_features1.shape)
```

```
Polynomial Features shape:  (307511, 56)
```

In [97]:
```
poly_transformer1.get_feature_names(input_features = ['REGION_POPULATION_RELATIVE', 'AMT_GOODS_PRICE', 'HOUSETYPE_MODE_block of flats', 'OCCUPATION_TYPE_Laborers', 'REG_CITY_NOT_LIVE_CITY'])[:15]
```

Out[97]:
```
['1',
 'REGION_POPULATION_RELATIVE',
 'AMT_GOODS_PRICE',
 'HOUSETYPE_MODE_block of flats',
 'OCCUPATION_TYPE_Laborers',
 'REG_CITY_NOT_LIVE_CITY',
 'REGION_POPULATION_RELATIVE^2',
 'REGION_POPULATION_RELATIVE AMT_GOODS_PRICE',
 'REGION_POPULATION_RELATIVE HOUSETYPE_MODE_block of flats',
 'REGION_POPULATION_RELATIVE OCCUPATION_TYPE_Laborers',
 'REGION_POPULATION_RELATIVE REG_CITY_NOT_LIVE_CITY',
 'AMT_GOODS_PRICE^2',
 'AMT_GOODS_PRICE HOUSETYPE_MODE_block of flats',
 'AMT_GOODS_PRICE OCCUPATION_TYPE_Laborers',
 'AMT_GOODS_PRICE REG_CITY_NOT_LIVE_CITY']
```

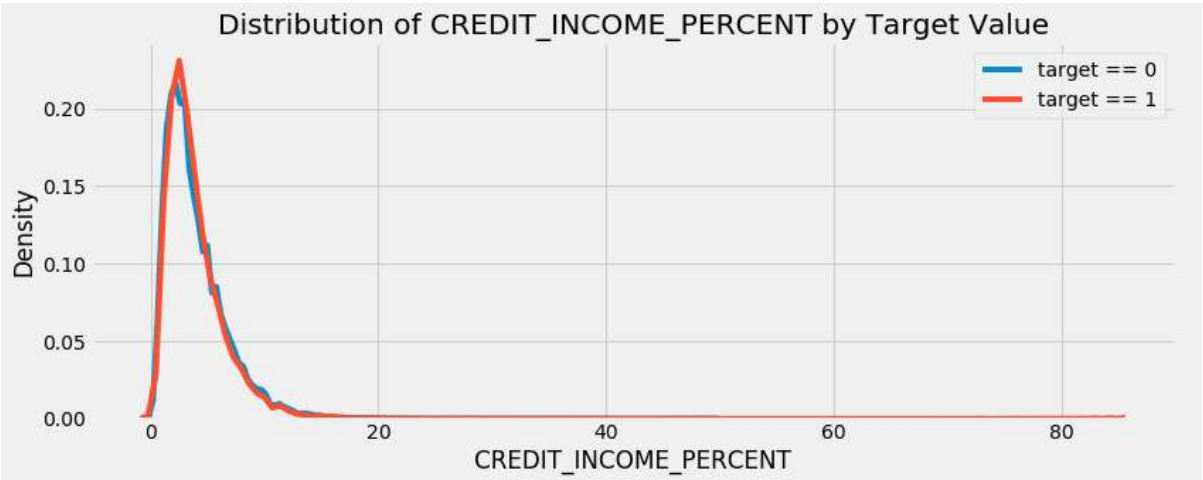
In [98]:
```
#Create a dataframe of the features
poly_features1 = pd.DataFrame(poly_features1,
                              columns = poly_transformer1.get_feature_names(['R
EGION_POPULATION_RELATIVE', 'AMT_GOODS_PRICE', 'HOUSETYPE_MODE_block of flats'
, 'OCCUPATION_TYPE_Laborers', 'REG_CITY_NOT_LIVE_CITY']))

#Add in the target
poly_features1['TARGET'] = poly_target1

#Find the correlations with the target
poly_corrs1 = poly_features1.corr()['TARGET'].sort_values()

#Display most negative and most positive
print(poly_corrs1.head(10))
print(poly_corrs1.tail(5))
```

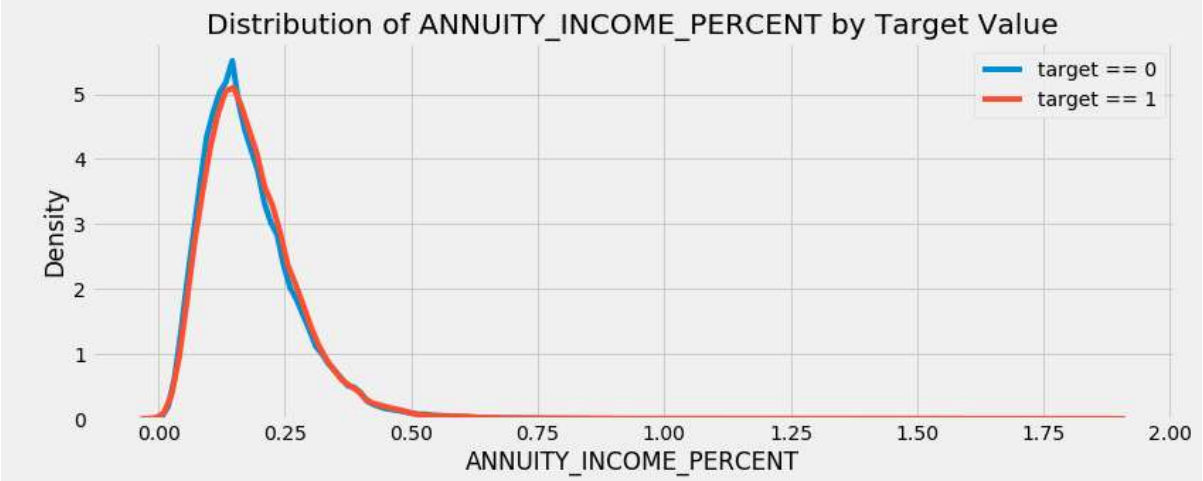
```
AMT_GOODS_PRICE HOUSETYPE_MODE_block of flats^2                          -
0.045723
AMT_GOODS_PRICE HOUSETYPE_MODE_block of flats                            -
0.045723
REGION_POPULATION_RELATIVE AMT_GOODS_PRICE                               -
0.045124
REGION_POPULATION_RELATIVE AMT_GOODS_PRICE HOUSETYPE_MODE_block of flats  -
0.044130
REGION_POPULATION_RELATIVE HOUSETYPE_MODE_block of flats^2               -
0.043955
REGION_POPULATION_RELATIVE HOUSETYPE_MODE_block of flats                 -
0.043955
AMT_GOODS_PRICE^2                                                        -
0.042902
REGION_POPULATION_RELATIVE AMT_GOODS_PRICE^2                             -
0.041442
AMT_GOODS_PRICE^2 HOUSETYPE_MODE_block of flats                          -
0.040789
HOUSETYPE_MODE_block of flats^2                                          -
0.040594
Name: TARGET, dtype: float64
REG_CITY_NOT_LIVE_CITY^3    0.044395
REG_CITY_NOT_LIVE_CITY^2    0.044395
REG_CITY_NOT_LIVE_CITY      0.044395
TARGET                      1.000000
1                                NaN
Name: TARGET, dtype: float64
```

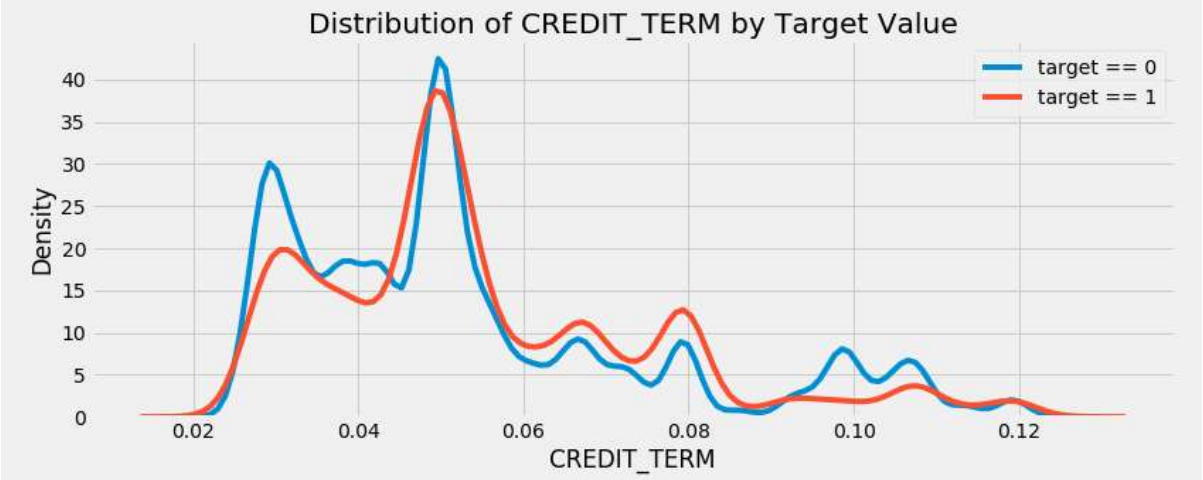
In [100]:
```
#Creating the new Features in both the train and test dataframes.

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

app_train_domain['CREDIT_INCOME_PERCENT'] = app_train_domain['AMT_CREDIT'] / a
pp_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']
```

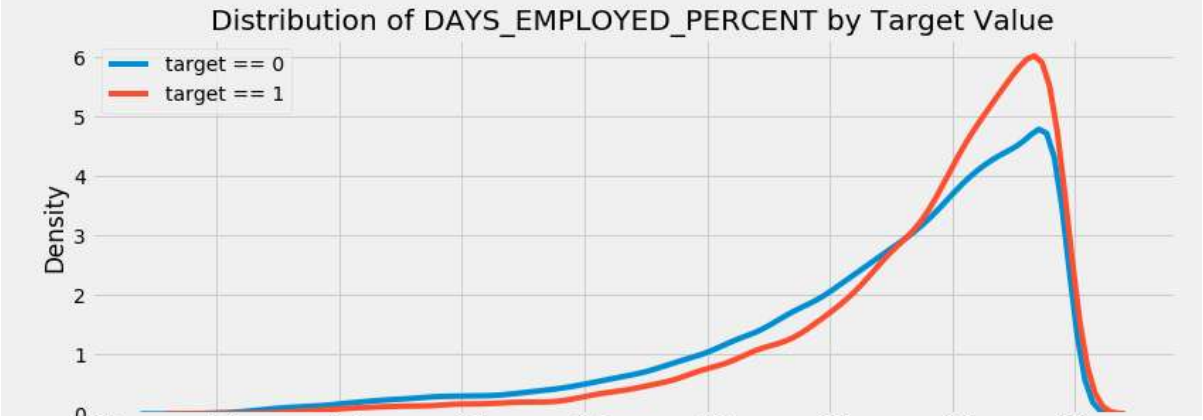
In [101]:
```
app_test_domain['CREDIT_INCOME_PERCENT'] = app_test_domain['AMT_CREDIT'] / app
_test_domain['AMT_INCOME_TOTAL']
app_test_domain['ANNUITY_INCOME_PERCENT'] = app_test_domain['AMT_ANNUITY'] / a
pp_test_domain['AMT_INCOME_TOTAL']
app_test_domain['CREDIT_TERM'] = app_test_domain['AMT_ANNUITY'] / app_test_dom
ain['AMT_CREDIT']
app_test_domain['DAYS_EMPLOYED_PERCENT'] = app_test_domain['DAYS_EMPLOYED'] /
app_test_domain['DAYS_BIRTH']
```

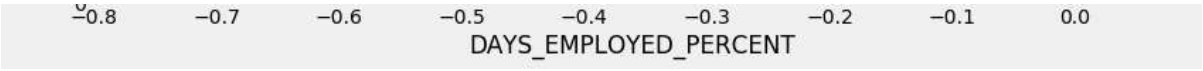
In [102]:
```
#Plotting the new features

plt.figure(figsize = (12, 20))
# iterating over the new features
for i, feature in enumerate(['CREDIT_INCOME_PERCENT', 'ANNUITY_INCOME_PERCENT'
, 'CREDIT_TERM', 'DAYS_EMPLOYED_PERCENT']):

    # creating a subplot for each source
    plt.subplot(4, 1, i + 1)
    # plotting the loans that have been repaid
    sns.kdeplot(app_train_domain.loc[app_train_domain['TARGET'] == 0, feature
], label = 'target == 0')
    #plotting the loans that have NOT been repaid
    sns.kdeplot(app_train_domain.loc[app_train_domain['TARGET'] == 1, feature
], label = 'target == 1')

    # Labelling the graphs
    plt.title('Distribution of %s by Target Value' % feature)
    plt.xlabel('%s' % feature); plt.ylabel('Density');

plt.tight_layout(h_pad = 2.5)
```


Distribution of CREDIT_INCOME_PERCENT by Target Value
target == 0
target == 1
Density
0.00
0.05
0.10
0.15
0.20
0
20
40
60
80
CREDIT_INCOME_PERCENT


Distribution of ANNUITY_INCOME_PERCENT by Target Value
target == 0
target == 1
Density
0
1
2
3
4
5
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
ANNUITY_INCOME_PERCENT


Distribution of CREDIT_TERM by Target Value
target == 0
target == 1
Density
0
5
10
15
20
25
30
35
40
0.02
0.04
0.06
0.08
0.10
0.12
CREDIT_TERM


Distribution of DAYS_EMPLOYED_PERCENT by Target Value
target == 0
target == 1
Density
0
1
2
3
4
5
6



In [103]:
```
from sklearn.preprocessing import MinMaxScaler, Imputer

# Dropping  target feature from training dataset
if 'TARGET' in app_train:
    train = app_train.drop(columns = ['TARGET'])
else:
    train = app_train.copy()

# Creating a list of feature names
features = list(train.columns)

# Creating a copy oftesting data
test = app_test.copy()

# Replacing missing values with the median
imputer = Imputer(strategy = 'median')

# Scaling all features to a range from 0 to 1
scaler = MinMaxScaler(feature_range = (0, 1))

# Fitting on training dataset
imputer.fit(train)

# Transforming the training and the testing datasets
train = imputer.transform(train)
test = imputer.transform(app_test)

# Repeating  using 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:  (307511, 240)
Testing data shape:  (48744, 240)
```

In [104]:
```
from sklearn.linear_model import LogisticRegression

# Making the Logistic Regression model with the specified regularization param
eter to avoid overfitting
log_reg = LogisticRegression(C = 0.0001)

# Training with the training dataset
log_reg.fit(train, train_labels)
```

Out[104]:
```
LogisticRegression(C=0.0001, class_weight=None, dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='warn', n_jobs=None, penalty='l2', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False)
```

In [105]:
```
# Doing predictions using only the second column (the one of not repaid loans)
log_reg_pred = log_reg.predict_proba(test)[:, 1]
```

In [106]:
```
# Submitting the dataframe
submit = app_test[['SK_ID_CURR']]
submit['TARGET'] = log_reg_pred

submit.head()
```

Out[106]:

	SK_ID_CURR	TARGET
0	100001	0.087750
1	100005	0.163957
2	100013	0.110238
3	100028	0.076575
4	100038	0.154924

In [107]:
```
# Saving the dataframe to a csv file
submit.to_csv('log_reg_baseline.csv', index = False)
```

In [108]:
```
from sklearn.ensemble import RandomForestClassifier

# Random forest classifier
random_forest = RandomForestClassifier(n_estimators = 100, random_state = 50,
verbose = 1, n_jobs = -1)
```

In [109]:
```
# Training on the training dataset
random_forest.fit(train, train_labels)

# Extrapolating each feature importance
feature_importance_values = random_forest.feature_importances_
feature_importances = pd.DataFrame({'feature': features, 'importance': feature
_importance_values})

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

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent worke
rs.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   29.8s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:  1.3min finished
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent worker
s.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.2s
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:    0.7s finished
```

In [110]:
```
# Submitting dataframe
submit = app_test[['SK_ID_CURR']]
submit['TARGET'] = predictions

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

In [111]:
```
#Trying to predict the new features

poly_features_names = list(app_train_poly.columns)

# Imputing polynomial features
imputer = Imputer(strategy = 'median')

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

# Scaling 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)
```

In [112]:
```
# Training on the training dataset
random_forest_poly.fit(poly_features, train_labels)

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

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent worke
rs.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   36.9s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:  1.6min finished
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent worker
s.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.0s
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:    0.2s finished
```

In [113]:
```
# Submitting the dataframe
submit = app_test[['SK_ID_CURR']]
submit['TARGET'] = predictions

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

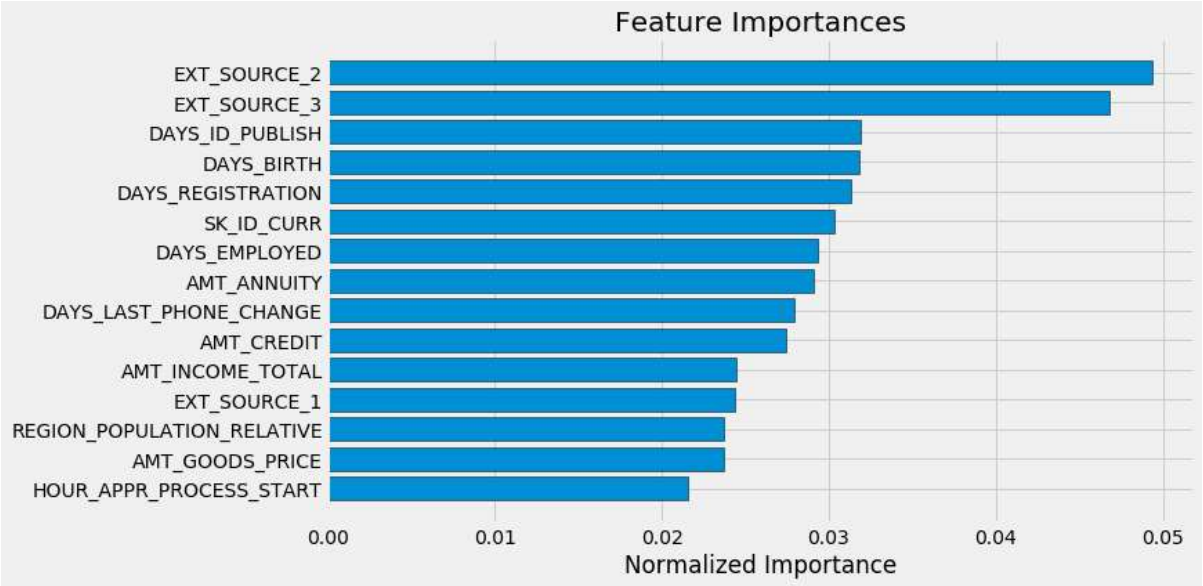
In [114]:
```
#Testing the domain features we created

app_train_domain = app_train_domain.drop(columns = 'TARGET')

domain_features_names = list(app_train_domain.columns)

# Imputing the new features
imputer = Imputer(strategy = 'median')

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

# Scaling the 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)

# Training on the training dataframe
random_forest_domain.fit(domain_features, train_labels)

# Extrapolating the importance of features
feature_importance_values_domain = random_forest_domain.feature_importances_
feature_importances_domain = pd.DataFrame({'feature': domain_features_names,
'importance': feature_importance_values_domain})

# Making predictions on the testing dataframe
predictions = random_forest_domain.predict_proba(domain_features_test)[:, 1]
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent worke
rs.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   29.4s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:  1.3min finished
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent worker
s.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.1s
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:    0.6s finished
```

In [115]:
```
# Submitting the dataframe
submit = app_test[['SK_ID_CURR']]
submit['TARGET'] = predictions

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

In [116]:
```
#Creating a function to plot the features

def plot_feature_importances(df):

    # Sorting features
    df = df.sort_values('importance', ascending = False).reset_index()

    # Normalization of feature importances to add up to one
    df['importance_normalized'] = df['importance'] / df['importance'].sum()

    # Creating a horizontal barchart to compare feature importances
    plt.figure(figsize = (10, 6))
    ax = plt.subplot()

    # Reversing the index to show the most meaningful features on the top of t
he barchart
    ax.barh(list(reversed(list(df.index[:15]))),
            df['importance_normalized'].head(15),
            align = 'center', edgecolor = 'k')

    ax.set_yticks(list(reversed(list(df.index[:15]))))
    ax.set_yticklabels(df['feature'].head(15))

    # Labeling the graphic
    plt.xlabel('Normalized Importance'); plt.title('Feature Importances')
    plt.show()

    return df
```

In [117]:
```
# Displaying features that have the highest impact in the model
feature_importances_sorted = plot_feature_importances(feature_importances)
```



In [118]:
```
feature_importances_domain_sorted = plot_feature_importances(feature_importanc
es_domain)
```