Data에 관해

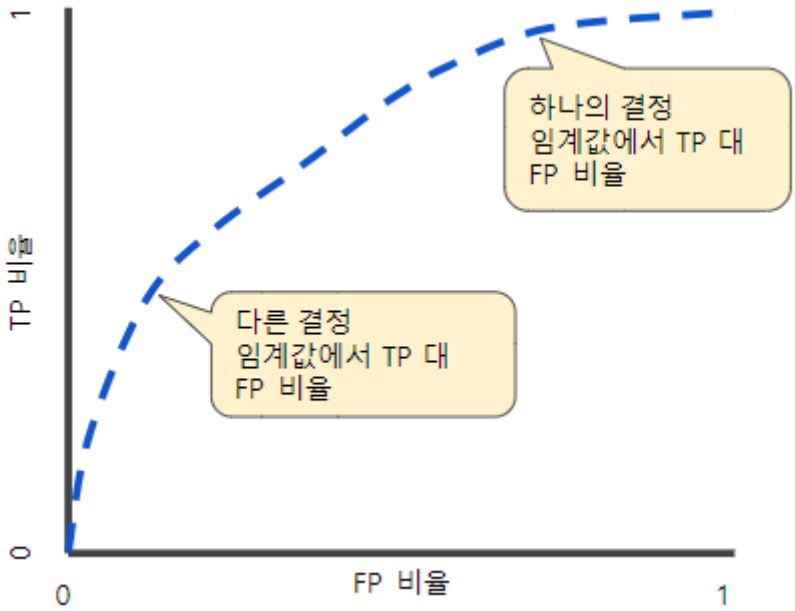
- 소비자가 대출을 다시 갚을지 안 갚을지를 판별하는 것.

Data 종류

- application_train/application_test : train, test data (TARGET 0 ==> 대출갚음, TARGET 1 ==> 대출 안 갚음)
- bureau(사무국?단체?) : 다른 금융 기관에서의 고객의 이전 신용정보. 각각의 previous credit은 고유의 bureau를 갖고 있고 하나의 loan은 여러 개의 previous credits을 가질 수 있다.
- bureau_balance : bureau의 previous credits에 대한 월별 data. 각 행은 한달치 previous credit이고 하나의 previous credit은 여러 개의 행을 가질 수 있다. 이 때 하나의 행은 각 달의 credit length만큼(?)
- previous_application : Home Credit에서 받은 이전 대출 신청서(application)들.
- POS_CASH_BALANCE : 이전 point of sale 혹은 현금 대출에 대한 월별 data
- credit_card_balance : Home Credit에서 받은 고객들의 이전 신용카드 정보에 대한 월별 데이터
- installments_payment : Home Credit 대출의 이전 payment 기록

Metric : ROC AUC

- ROC가 사용하는 2가지 지표 TPR(참 양성 비율), FPR(허위 양성 비율)
- AUC는 ROC 곡선 아래의 적분값을 의미함
- ROC AUC metric을 사용하게 되면 0 또는 1의 prediction으로 나오는 것이 아니라 0과 1사이의 어떤 확률적인 값으로 나오게 된다고 한다.
- 이러한 특성으로 인해 Class 불균형이 심할 때 이와 같은 방식을 사용함.
- 예를들어 어떤 사람이 테러리스트인지 아닌지 맞추는 문제를 99.9999%의 정확도로 맞추고 싶다면 단순히 테러리스트가 아니라고 분류하면 된다. 이러한 문제를 대비해서 사용하는 더 정확도 높은 지표가 ROC AUC 이다.



In [1]:

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import os
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:

```
print(os.listdir("../"))
```

```
['Untitled.ipynb', '.ipynb_checkpoints', 'bit_trading', '2021_March_IMC', '(Kaggle)P
orto_Seguro_Driver_Safety', '(Kaggle)Home_Credit_Default_Risk']
```

In [3]:

```
app_train = pd.read_csv('./input/application_train.csv')
print('Training data shape: ', app_train.shape)
app_train.head()
```

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

Out[3]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_(
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

Train set에서는 총 307511의 데이터들이 있고 각 데이터에 대해서는 122개의 feature들이 있다. 그 중 TARGET 값도 포함이 되어있다.

In [4]:

```
app_test = pd.read_csv('./input/application_test.csv')
print('Testing data shape: ', app_test.shape)
app_test.head()
```

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

Out[4]:

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REAL
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

Test set에서는 48744개의 데이터들이 있고 TARGET값을 제외한 121개의 정보들이 있다.

In [5]:

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

Out[5]:

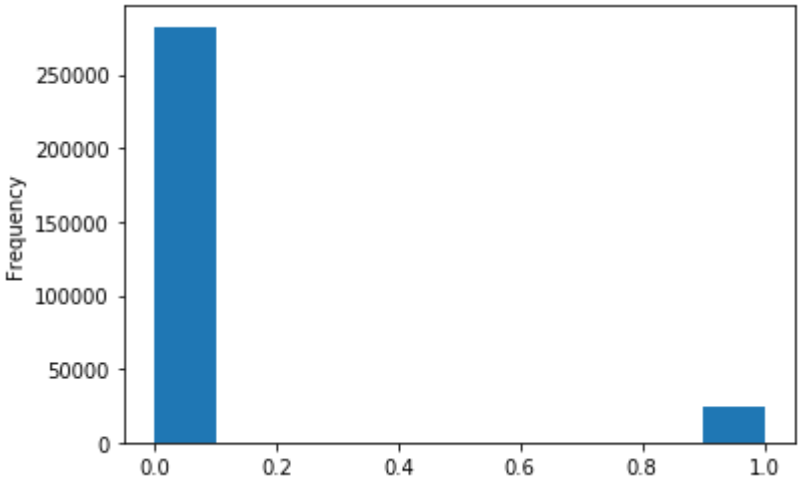
```
0    282686
1     24825
Name: TARGET, dtype: int64
```

In [6]:

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

Out[6]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3a4aa18940>
```



Data 불균형 문제가 눈에 보인다. Repaid 된 경우(0.0)가 Repaid 되지 않은 경우(1.0)보다 훨씬 많다.

In [7]:

```
def missing_values_table(df):
    mis_val = df.isnull().sum()
    mis_val_percent = 100*df.isnull().sum()/len(df)
    mis_val_table =pd.concat([mis_val, mis_val_percent], axis=1)
    mis_val_table_ren_columns = mis_val_table.rename(columns = {0:'Missing Values', 1:'% of Total Va
    mis_val_table_ren_columns = mis_val_table_ren_columns[mis_val_table_ren_columns.iloc[:,1] !=0].s
    '% of Total Values', ascending=False).round(1)
    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 mis_val_table_ren_columns
```

In [8]:

```
missing_values = missing_values_table(app_train)
missing_values.head(20)
```

```
Your selected dataframe has122columns.
There are 67columns that have missing values.
```

Out[8]:

	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

In [9]:

```
app_train.dtypes.value_counts()
```

Out[9]:

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

In [10]:

```
app_train.select_dtypes('object').apply(pd.Series.nunique, axis=0)
```

Out[10]:

```
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
```

- Categorical variable 중 2 unique categories를 가지는 경우 label encoding을 하고

- 2개보다 더 많이 갖고 있는 경우 one-hot encoding을 함

In [11]:

```
le = LabelEncoder()
le_count = 0

# label encoding
for col in app_train:
    if app_train[col].dtype == 'object':
        if len(list(app_train[col].unique()))<=2:
            le.fit(app_train[col])
            app_train[col] = le.transform(app_train[col])
            app_test[col] = le.transform(app_test[col])
            le_count+=1
print('%d columns were label encoded.' % le_count)
```

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

In [12]:

```
# one-hot encoding
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)
```

One hot encoding으로 인해 training data에 column들이 더 생기게 된다. 왜냐하면 몇몇 categorical value들은 testing data에 없기 때문. 그래서 testing data에 없는 column들을 training data에서 지워주기 위해 dataframe align을 진행해줘야 한다.

진행할 때는 training data에서 target value를 빼준다. 얘는 test data에 없긴하지만 train data에 필요한 정보이다. align을 할 때는 axis=1로 둬서 row가 아닌 column에 대해 진행할 수 있도록 설정한다.

test data에 없는 column들을 왜 제거하지?
column 제거를 하는데 왜 align(정렬)을 수행하지?

In [13]:

```
train_labels = app_train['TARGET']
app_train, app_test = app_train.align(app_test, join = 'inner',axis=1)
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)
```

이렇게 되면 이제 feature가 제대로 원하는대로 나오게 된다. Train, test feature의 column 개수가 1개만 차이가 난다.

Anomalies를 처리하기 위한 과정. Anomalies를 정량적으로 분석하기 위해서는 describe를 통해 column의 statistics를 describe()로 살펴본다. 이 때 DAYS_BIRTH 는 current loan application에 상대적으로(?) 기록되었기 때문에 음수로 작성되어있다. 그래서 -365로 나누어준다.

In [14]:

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

Out[14]:

```
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 [15]:

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

Out[15]:

```
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
```

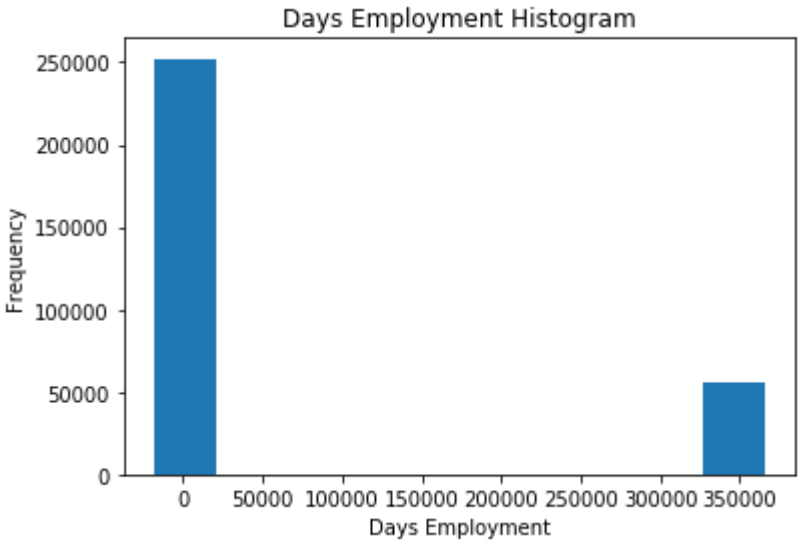
Max값이 1000년? 이상하다

In [16]:

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

Out[16]:

```
Text(0.5, 0, 'Days Employment')
```



In [17]:

```
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 8.66% of loans
The anomalies default on 5.40% of loans
There are 55374 anomalous days of employment
```

보통은 anomalies 값들은 missing value로 놓고 Imputation(결측값 대체)을 이용해 값을 채워넣는 방안이 일반적이다. 그런데 이 경우 anomalies 값도 어느 정도 의미를 갖고 있는 듯 하여 값을 np.nan (not a number) 으로 채워넣고 boolean column을 새로 추가해서 anomaly한 값인지 아닌지를 판별할 수 있도록 할 예정

In [18]:

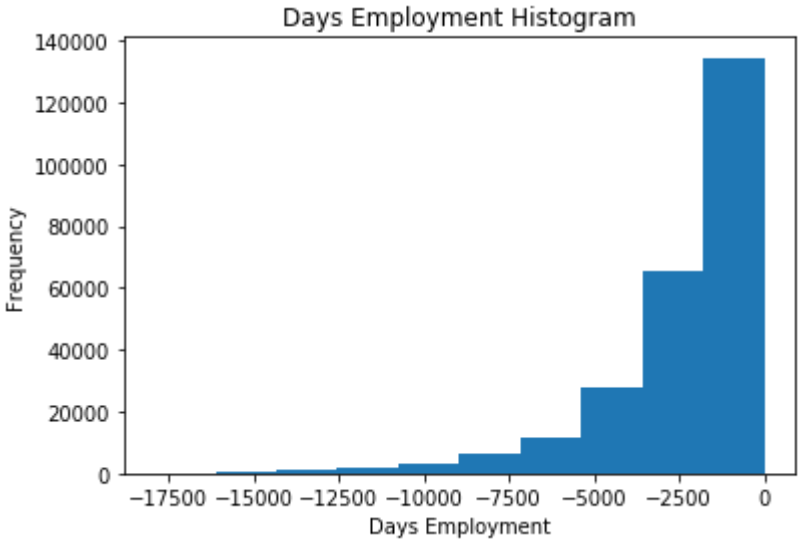
```
app_train['DAYS_EMPLOYED_ANOM'] = app_train["DAYS_EMPLOYED"]==365243

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

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

Out[18]:

```
Text(0.5, 0, 'Days Employment')
```



추후에 anomalous한 값들은 다른 값으로 대체할 예정 (아마 median 값으로). 우리가 training data에 대해서 진행한 것은 test data에 대해서도 진행해줘야 한다.

In [19]:

```
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'].
```

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

Correlations

각각의 feature 와 target과의 관계에 대해 알아보자. 각 변수와 target에 대한 Pearson correlation coefficient를 `.corr` 이라는 dataframe method를 이용해 구해볼 수 있다.

- .00 - .19 : "매우 약함"
- .20 - .39 : "약함"
- .40 - .59 : "보통"
- .60 - .79 : "강함"
- .80 - 1.0 : "매우 강함"

In [20]:

```
correlations = app_train.corr()['TARGET'].sort_values()
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
```

보면 가장 관련이 높은 feature는 DAYS_BIRTH이다.(TARGET을 제외하면). DAYS_BIRTH는 대출 시점에서의 고객의 age를 의미한다. 한 가지 헷갈리는거는 고객이 나이가 들수록 대출에 대한 채무 불이행 가능성(TARGET==0)이 줄어든다(?)는 것이다. 그래서 우리는 값의 절댓값을 이용하여 관계를 negative하게 만들어줄 것이다.

Repayment에 나이가 미치는 영향

In [21]:

```
app_train['DAYS_BIRTH'] = abs(app_train['DAYS_BIRTH'])
app_train['DAYS_BIRTH'].corr(app_train['TARGET'])
```

Out[21]:

-0.07823930830982694

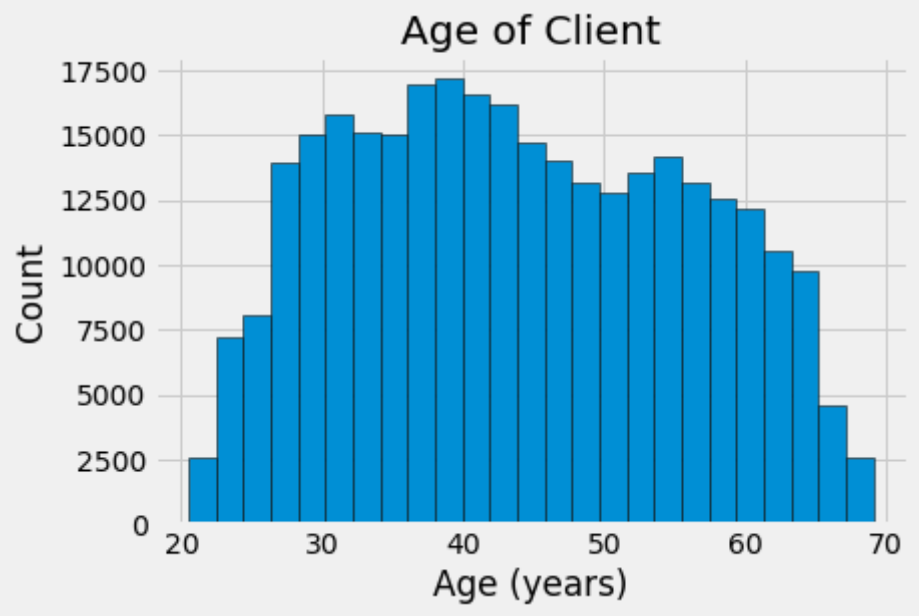
나이가 들수록 target과 음의 선형 관계(negative linear relationship)가 성립된다. 즉, 고객이 나이가 들수록 대출을 더 제때 잘 값는다는 것을 알 수 있다.
음의 관계가 나오는 것은 나이가 들 수록 TARGET값이 0에 가까워질 확률이 올라가기 때문이라고 보면 될까(?)

In [22]:

```
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')
```

Out[22]:

```
Text(0, 0.5, 'Count')
```



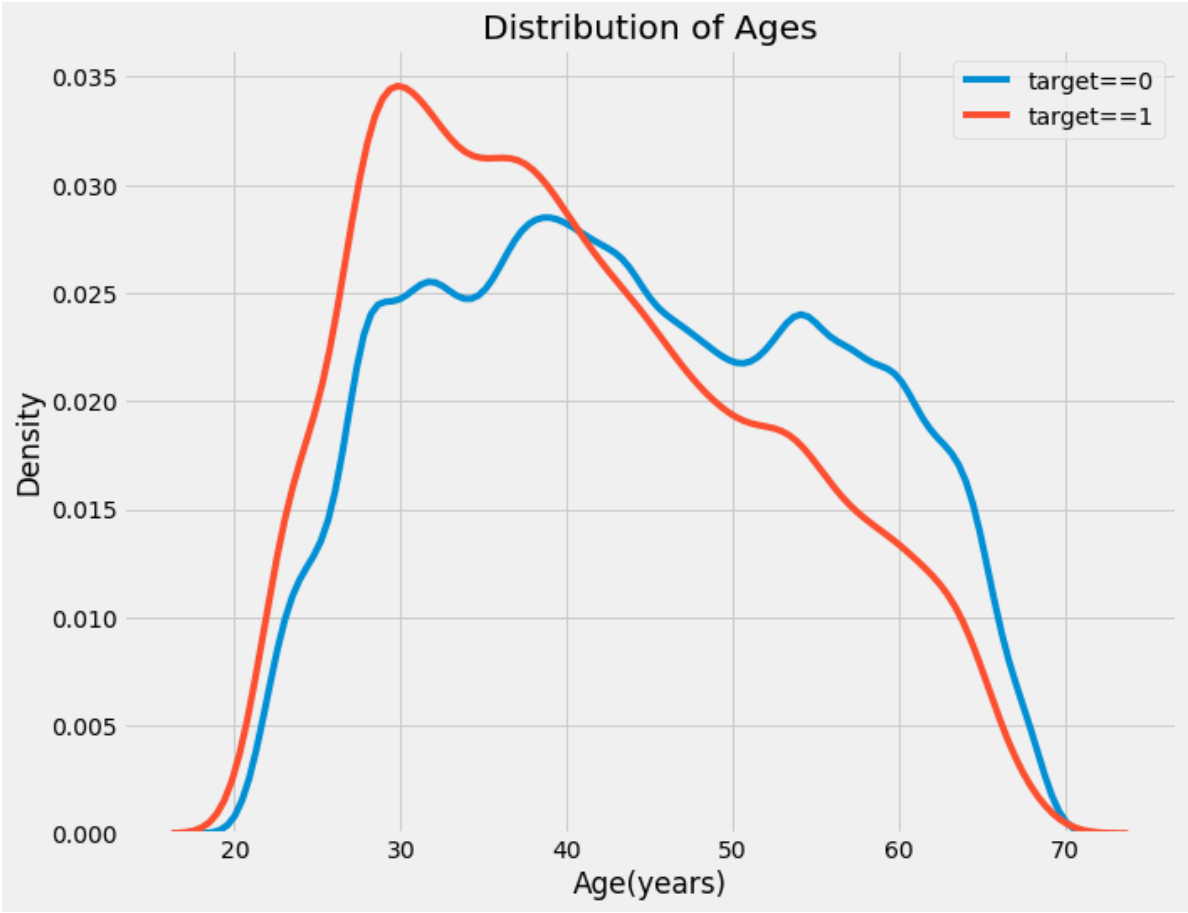
나이가 Target에 미치는 영향을 시각화하기 위해 KDE(Kernel Density Estimation)plot을 이용해보자. (Smoothed histogram이라고 생각하면 된다.)

In [23]:

```
plt.figure(figsize = (10, 8))
sns.kdeplot(app_train.loc[app_train['TARGET']==0, 'DAYS_BIRTH']/365, label = 'target==0')
sns.kdeplot(app_train.loc[app_train['TARGET']==1, 'DAYS_BIRTH']/365, label = 'target==1')
plt.xlabel('Age(years)')
plt.ylabel('Density')
plt.title('Distribution of Ages')
```

Out[23]:

```
Text(0.5, 1.0, 'Distribution of Ages')
```



Target == 1인 값이 나이가 적은 쪽에 치우쳐져 있다.
다른 방법으로도 관계성을 파악해보자. 나이를 5년 씩 잘라본다. 그리고 각각의 카테고리에 대해서 평균 target값을 분석하여 repaid 하지 않은 경우의 비율을 알아본다.

In [24]:

```
age_data = app_train[['TARGET', 'DAYS_BIRTH']]
age_data['YEARS_BIRTH'] = age_data['DAYS_BIRTH']/365
age_data['YEARS_BINNED'] = pd.cut(age_data['YEARS_BIRTH'], bins = np.linspace(20, 70, num=11))
age_data.head(10)
```

Out[24]:

	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 [25]:

```
age_groups = age_data.groupby('YEARS_BINNED').mean()
age_groups
```

Out[25]:

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

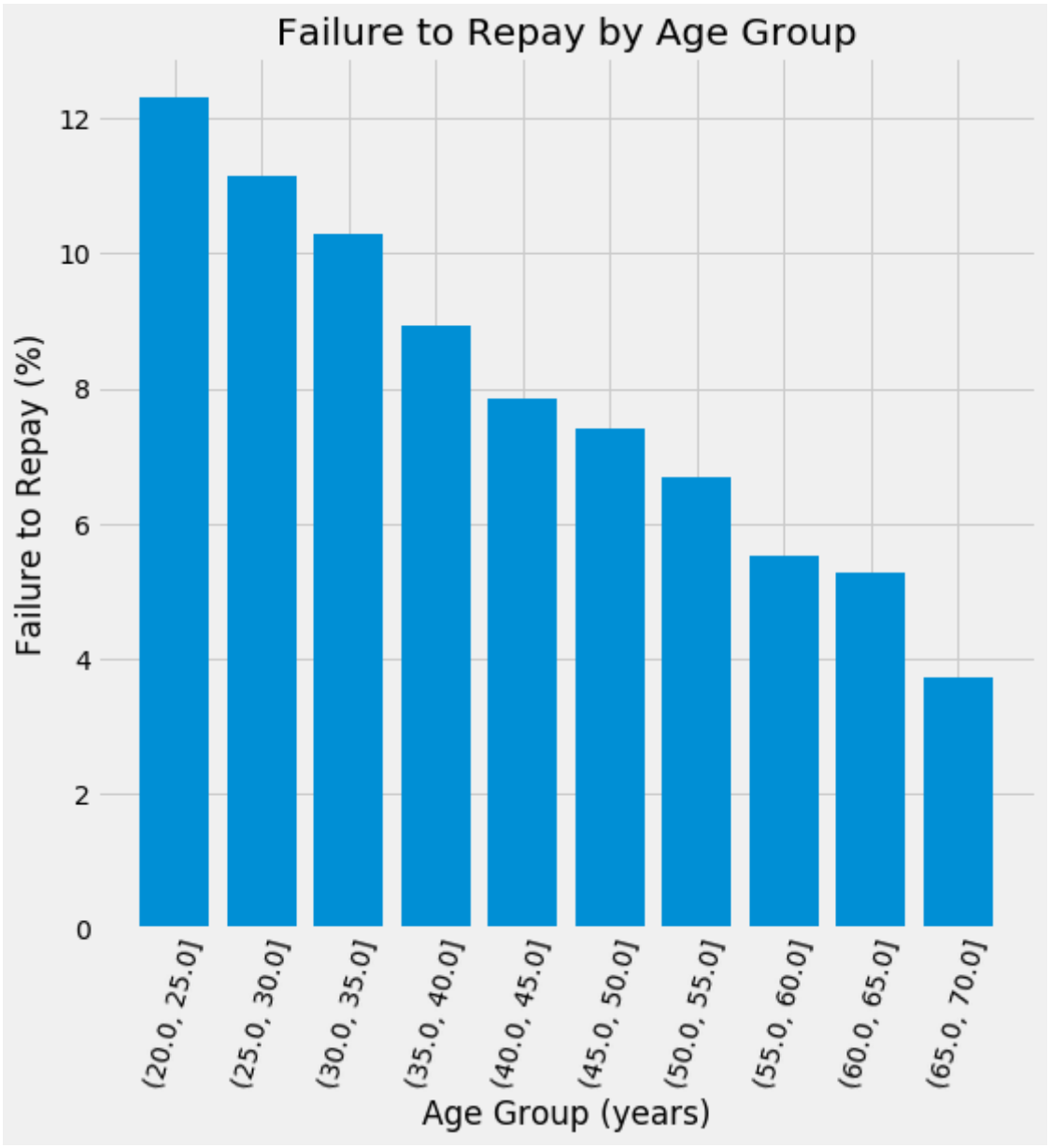
In [26]:

```
plt.figure(figsize = (8, 8))
plt.bar(age_groups.index.astype(str), 100*age_groups['TARGET'])

plt.xticks(rotation = 75)
plt.xlabel('Age Group (years)')
plt.ylabel('Failure to Repay (%)')
plt.title('Failure to Repay by Age Group')
```

Out[26]:

```
Text(0.5, 1.0, 'Failure to Repay by Age Group')
```



이 표를 통해 나이가 적을 수록 repay하지 않을 확률이 높다는 것을 알아챌 수 있따.

Exterior Sources

target과 가장 강한 negative correlation을 갖고 있는 3가지 feature들은 `EXT_SOURCE_1` , `EXT_SOURCE_2` , `EXT_SOURCE_3` 이다. 뒤에 뭔 말인지는 모르겠는데 대충 누적 신용 등급의 종류일 수도 있다라고 한다.

In [27]:

```
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[27]:

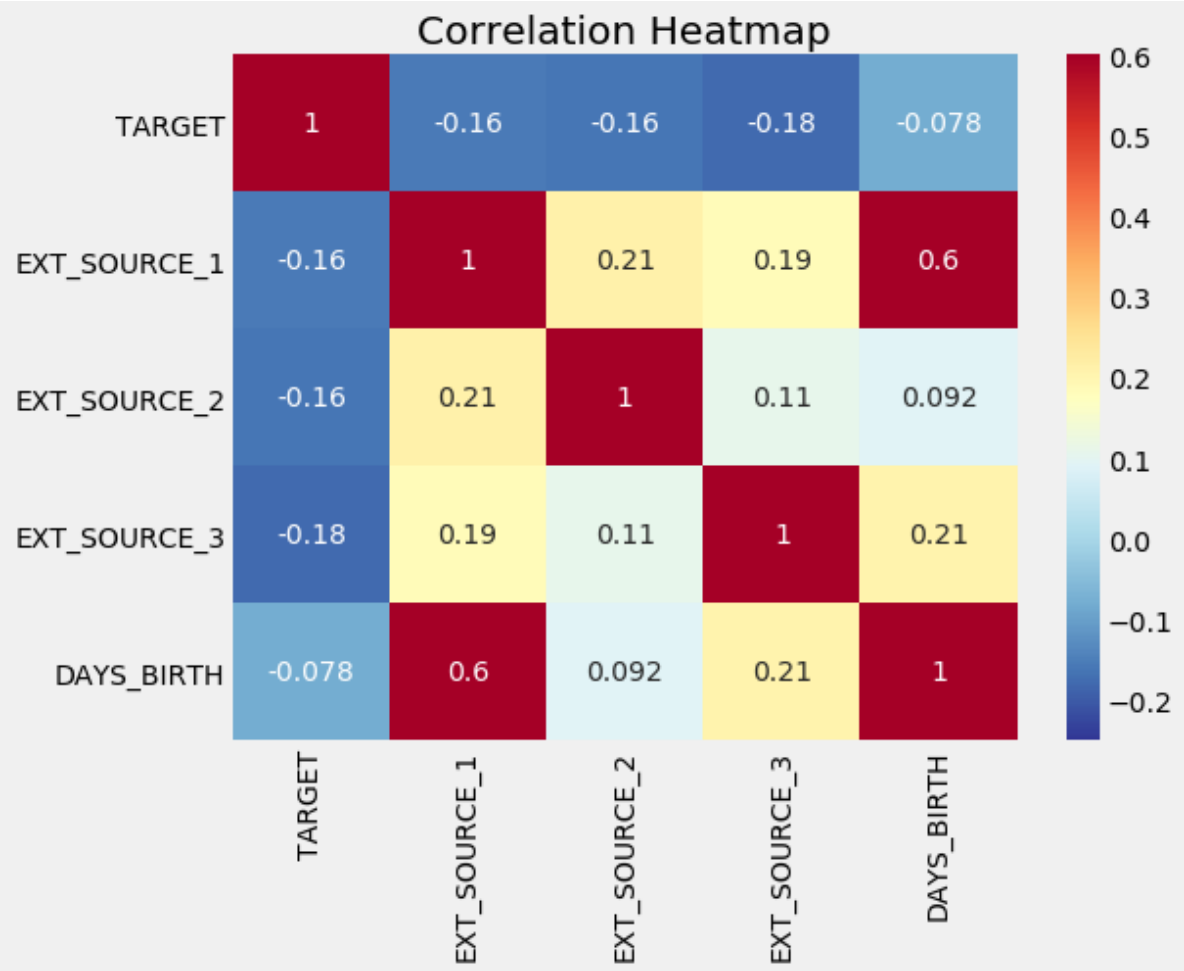
	TARGET	EXT_SOURCE_1	EXT_SOURCE_2	EXT_SOURCE_3	DAYS_BIRTH
TARGET	1.000000	-0.155317	-0.160472	-0.178919	-0.078239
EXT_SOURCE_1	-0.155317	1.000000	0.213982	0.186846	0.600610
EXT_SOURCE_2	-0.160472	0.213982	1.000000	0.109167	0.091996
EXT_SOURCE_3	-0.178919	0.186846	0.109167	1.000000	0.205478
DAYS_BIRTH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

In [28]:

```
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')
```

Out[28]:

Text(0.5, 1, 'Correlation Heatmap')

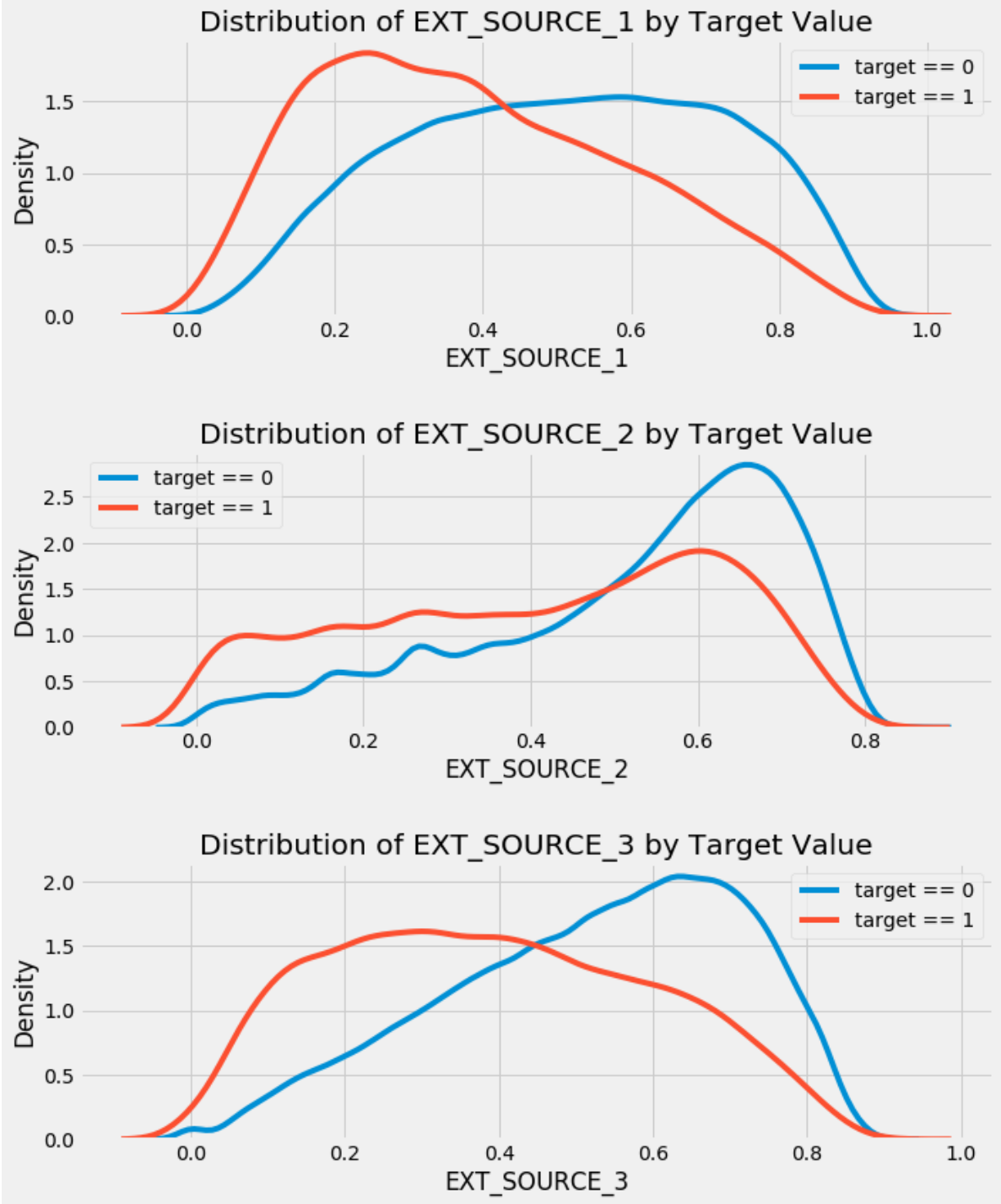


EXT_SOURCE는 모두 target과 negative한 관계를 갖고 있으며 EXT_SOURCE의 값이 올라갈 수록 repay의 비율이 높아진다는 것을 확인할 수 있다.
다음으로는 각 feature의 distribution에 대해 알아보자.

In [29]:

```
plt.figure(figsize = (10, 12))
for i, source in enumerate(['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3']):
    plt.subplot(3, 1, i+1)
    sns.kdeplot(app_train.loc[app_train['TARGET'] ==0, source], label = 'target == 0')
    sns.kdeplot(app_train.loc[app_train['TARGET'] ==1, source], label = 'target == 1')

    plt.title('Distribution of %s by Target Value' % source)
    plt.xlabel('%s' % source)
    plt.ylabel('Density')
plt.tight_layout(h_pad = 2.5)
```



EXT_SOURCE_3에서 0과 1의 분포 차이가 가장 크게 나타난다. 사실 EXT_SOURCE의 경우 모두 관계성이 매우 약함 으로 나타났지만 그래도 여전히 유용한 정보로 사용될 수 있을 것이라고 본다.

Pairs Plot

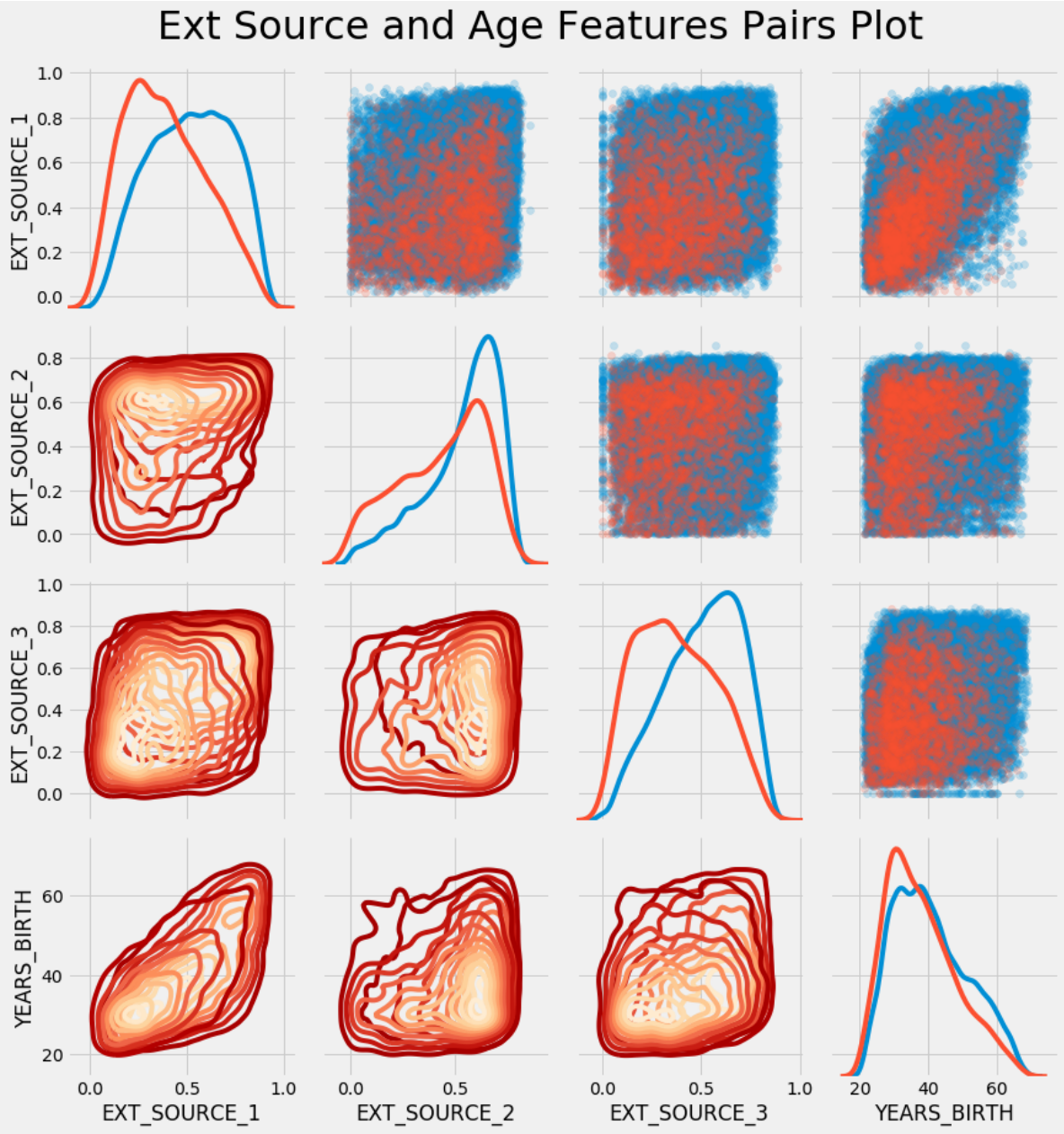
EXT_SOURCE와 DAYS_BIRTH에 대한 pairs plot을 그려보자.

In [30]:

```
plot_data = ext_data.drop(columns = ['DAYS_BIRTH']).copy()
plot_data['YEARS_BIRTH'] = age_data['YEARS_BIRTH']
plot_data = plot_data.dropna().loc[:100000, :]
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)
grid = sns.PairGrid(data = plot_data, size = 3, diag_sharey = False, hue = 'TARGET', vars = [x for
grid.map_upper(plt.scatter, alpha = 0.2)
grid.map_diag(sns.kdeplot)
grid.map_lower(sns.kdeplot, cmap = plt.cm.OrRd_r)
plt.suptitle('Ext Source and Age Features Pairs Plot', size = 32, y = 1.05)
```

Out[30]:

```
Text(0.5, 1.05, 'Ext Source and Age Features Pairs Plot')
```



빨간색이 채무이행 되지 않은 대출이고 파란색이 채무이행이 잘 된 대출을 뜻한다. EXT_SOURCE_1과 DAYS_BIRTH사이의 별다른 positive linear relationship은 보이지 않는다.

Feature Engineering

- Polynomial features
- Domain knowledge features

Polynomial Features

EXT_SOURCE_1^2, EXT_SOURCE_2^2, EXT_SOURCE_1 X EXT_SOURCE_2, EXT_SOURCE_1 X EXT_SOURCE_2^2, EXT_SOURCE_1^2 X EXT_SOURCE_2^2 와 같은 feature들을 만들어 줄 수 있다. 각각의 데이터 자체는 target value와 연관이 약할 수 있지만 저런 식으로 combination을 해서 생각해보면 target에 strong relationship을 가질 수 있을 것이다. [polynomial data engineering (https://jakevdp.github.io/PythonDataScienceHandbook/05.04-feature-engineering.html)](https://jakevdp.github.io/PythonDataScienceHandbook/05.04-feature-engineering.html) 을 참고해볼 수 있다. ScikitLearn의 `PolynomialFeatures` 라는 유용한 class를 이용할 예정. Degree of 3(3차원)을 이용할 예정. 너무 높은 차수를 이용하게 되면 feature들이 exponentially scale될 수 있고 overfitting 문제가 발생할 수 있기 때문.

In [31]:

```
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']]

# from sklearn.preprocessing import Imputer
from sklearn.impute import SimpleImputer as Imputer
imputer = Imputer(strategy = 'median')
poly_target = poly_features['TARGET']
poly_features = poly_features.drop(columns = ['TARGET'])

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

from sklearn.preprocessing import PolynomialFeatures
poly_transformer = PolynomialFeatures(degree=3) # degree specify
```

In [32]:

```
poly_transformer.fit(poly_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 [33]:

```
poly_transformer.get_feature_names(input_features = ['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3',
```

Out[33]:

```
['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']
```

35개의 feature들이 생성되었다. 이제 저 feature들이 target과 연관성이 있는지 확인해보자.

In [34]:

```
poly_features = pd.DataFrame(poly_features, columns = poly_transformer.get_feature_names(['EXT_SOUR
                                                                                          'EXT_SOURCE_3', 'DAYS_BIF
poly_features['TARGET'] = poly_target
poly_corrs = poly_features.corr()['TARGET'].sort_values()
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 [35]:

```
poly_features_test = pd.DataFrame(poly_features_test, columns = poly_transformer.get_feature_names(

# 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 polynomial 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 dataframes
app_train_poly, app_test_poly = app_train_poly.align(app_test_poly, join='inner', axis=1)

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)
```

Domain Knowledge Features

여기 (https://www.kaggle.com/jsaguiar/lightgbm-with-simple-features?scriptVersionId=6025993)를 참고해서 직접 만드신 금융 데이터들. 고객이 융자를 연체할 것인지 안 할 것인지를 판별할 수 있는 지표

CREDIT_INCOME_PERCENT : 고객 수입 대비 credit 양을 퍼센티지로 나타낸 것
ANNUITY_INCOME_PERCENT : 고객 수입 대비 대출연금 퍼센티지
CREDIT_ITEM : 지급기간 (월) (연금이 만기월액이기 때문(?))
DAYS_EMPLOYED_PERCENT : 고객 나이 대비 고용 기간을 퍼센티지로 나타낸 것

In [36]:

```
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_
app_train_domain['ANNUITY_INCOME_PERCENT'] = app_train_domain['AMT_ANNUITY'] / app_train_domain['AM
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['DA
```

In [37]:

```
app_test_domain['CREDIT_INCOME_PERCENT'] = app_test_domain['AMT_CREDIT'] / app_test_domain['AMT_INC
app_test_domain['ANNUITY_INCOME_PERCENT'] = app_test_domain['AMT_ANNUITY'] / app_test_domain['AMT_I
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_
```

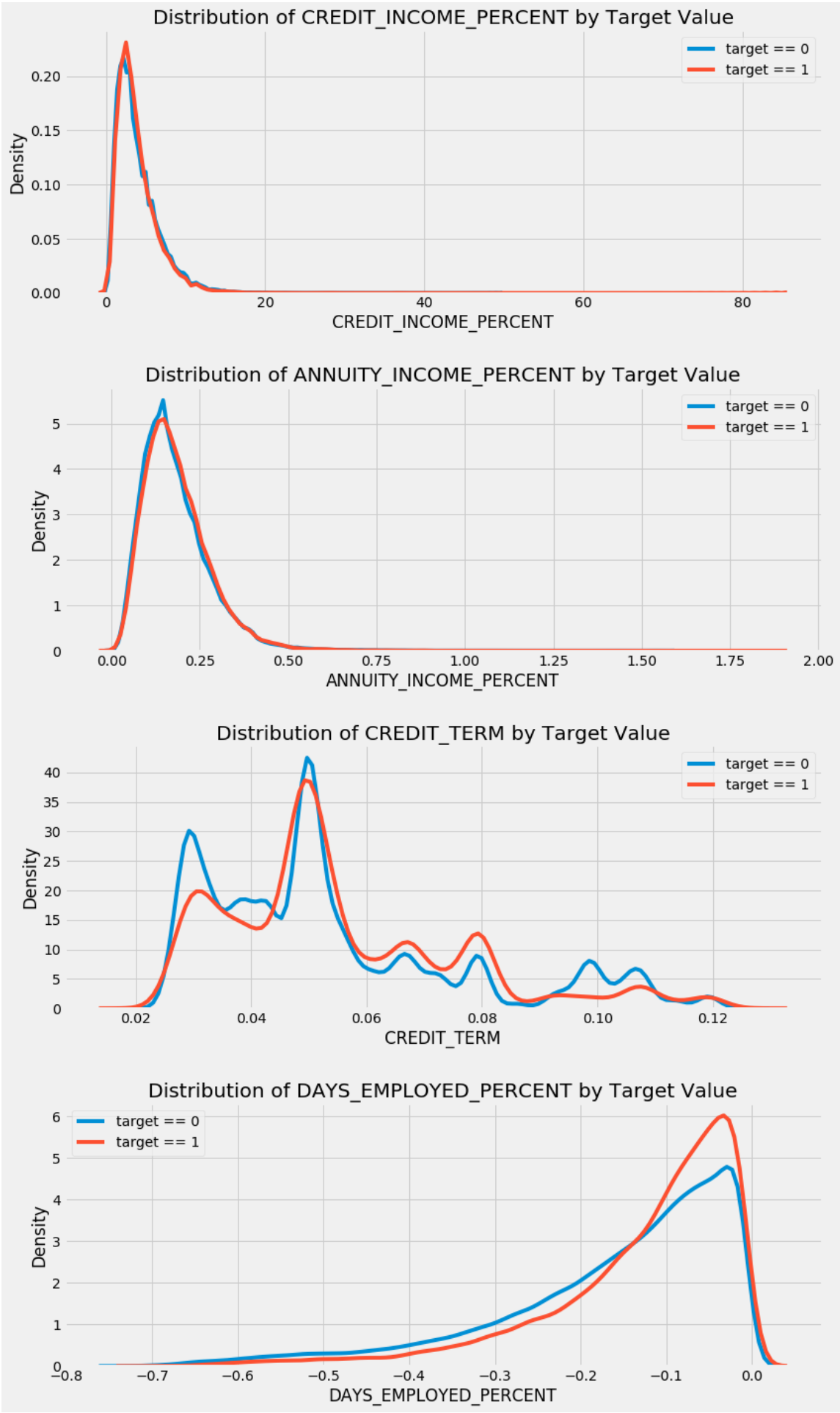
Visualize New Variables

KDE Plot으로 새로 만든 변수들을 시각화해봅시다

In [39]:

```
plt.figure(figsize = (12, 20))
for i, feature in enumerate(['CREDIT_INCOME_PERCENT', 'ANNUITY_INCOME_PERCENT', 'CREDIT_TERM', 'DAY
    plt.subplot(4, 1, i+1)
    sns.kdeplot(app_train_domain.loc[app_train_domain['TARGET']==0, feature], label = 'target == 0'
    sns.kdeplot(app_train_domain.loc[app_train_domain['TARGET'] == 1, feature], label = 'target ==

    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
0.20
0.15
0.10
0.05
0.00
0
20
40
60
80
CREDIT_INCOME_PERCENT
Distribution of ANNUITY_INCOME_PERCENT by Target Value
target == 0
target == 1
Density
5
4
3
2
1
0
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
40
35
30
25
20
15
10
5
0
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
6
5
4
3
2
1
0
−0.8
−0.7
−0.6
−0.5
−0.4
−0.3
−0.2
−0.1
0.0
DAYS_EMPLOYED_PERCENT

In [42]:

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.impute import SimpleImputer
if 'TARGET' in app_train:
    train = app_train.drop(columns = ['TARGET'])
else :
    train = app_train.copy()

features = list(train.columns)
test = app_test.copy()
imputer = Imputer(strategy = 'median')
scaler = MinMaxScaler(feature_range=(0,1))
imputer.fit(train)

train = imputer.transform(train)
test = imputer.transform(app_test)

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 [43]:

```
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression(C=0.0001)
log_reg.fit(train, train_labels)
```

Out[43]:

```
LogisticRegression(C=0.0001)
```

In [44]:

```
log_reg_pred = log_reg.predict_proba(test)[:, 1]
```

sample_submission.csv 에는 SK_ID_CURR과 TARGET 의 형태로 들어갈 예정.

In [45]:

```
submit = app_test[['SK_ID_CURR']]
submit['TARGET'] = log_reg_pred
submit.head()
```

Out[45]:

	SK_ID_CURR	TARGET
0	100001	0.078515
1	100005	0.137926
2	100013	0.082194
3	100028	0.080921
4	100038	0.132618

0과 1사이의 예측값을 이용하여 applicant를 분류할 때 대출이 위험하다는 것을 확인할 수 있는 임계값을 설정할 수 있다.

In [46]:

```
submit.to_csv('log_reg_baseline.csv', index = False)
```

Logistic Regression 써서 한거는 0.671 정도의 점수가 나오게 된다.

Improved Model : Random Forest

100개의 tree들을 random forest에 사용해보자.

In [47]:

```
klearn.ensemble import RandomForestClassifier
forest = RandomForestClassifier(n_estimators = 100, random_state = 50, verbose = 1, n_jobs = -1)
```

In [48]:

```
random_forest.fit(train, train_labels)
feature_importance_values = random_forest.feature_importances_
feature_importances = pd.DataFrame({'feature' : features, 'importance' : feature_importance_values})
predictions = random_forest.predict_proba(test)[:, 1]
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done  18 tasks      | elapsed:    4.3s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   14.3s finished
[Parallel(n_jobs=16)]: Using backend ThreadingBackend with 16 concurrent workers.
[Parallel(n_jobs=16)]: Done  18 tasks      | elapsed:    0.1s
[Parallel(n_jobs=16)]: Done 100 out of 100 | elapsed:    0.3s finished
```

In [49]:

```
submit = app_test[['SK_ID_CURR']]
submit['TARGET'] = predictions
submit.to_csv('random_forest_baseline_csv', index = False)
```

얘도 하면 0.678 정도의 점수가 나온다.

Engineered된 Feature들로 예측하기

Polynomial Features와 Domain Knowledge가 model을 개선시켰는지 확인해보자.

In [50]:

```
poly_features_names = list(app_train_poly.columns)
imputer = Imputer(strategy = 'median')
poly_features = imputer.fit_transform(app_train_poly)
poly_features_test = imputer.transform(app_test_poly)

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_j
```

In [51]:

```
random_forest_poly.fit(poly_features, train_labels)
predictions = random_forest_poly.predict_proba(poly_features_test)[:, 1]
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done  18 tasks      | elapsed:    6.9s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   23.0s finished
[Parallel(n_jobs=16)]: Using backend ThreadingBackend with 16 concurrent workers.
[Parallel(n_jobs=16)]: Done  18 tasks      | elapsed:    0.1s
[Parallel(n_jobs=16)]: Done 100 out of 100 | elapsed:    0.2s finished
```

In [52]:

```
submit = app_test[['SK_ID_CURR']]
submit['TARGET'] = predictions

submit.to_csv('random_forest)baseline)engineered.csv', index = False)
```

얘도 점수가 0.678 정도 나온다고 한다.

Domain Features 테스트

In [53]:

```
app_train_domain = app_train_domain.drop(columns = 'TARGET')
domain_features_names = list(app_train_domain.columns)
imputer = Imputer(strategy = 'median')
domain_features = imputer.fit_transform(app_train_domain)
domain_features_test = imputer.transform(app_test_domain)

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
random_forest_domain.fit(domain_features, train_labels)
feature_importance_values_domain = random_forest_domain.feature_importances_
feature_importances_domain = pd.DataFrame({'feature': domain_features_names, 'importance': feature_i
predictions = random_forest_domain.predict_proba(domain_features_test)[:, 1]
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done  18 tasks      | elapsed:    4.4s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   15.2s finished
[Parallel(n_jobs=16)]: Using backend ThreadingBackend with 16 concurrent workers.
[Parallel(n_jobs=16)]: Done  18 tasks      | elapsed:    0.1s
[Parallel(n_jobs=16)]: Done 100 out of 100 | elapsed:    0.3s finished
```

In [54]:

```
submit = app_test[['SK_ID_CURR']]
submit['TARGET'] = predictions
submit.to_csv('random_forest_baseline_domain.csv', index = False)
```

In [55]:

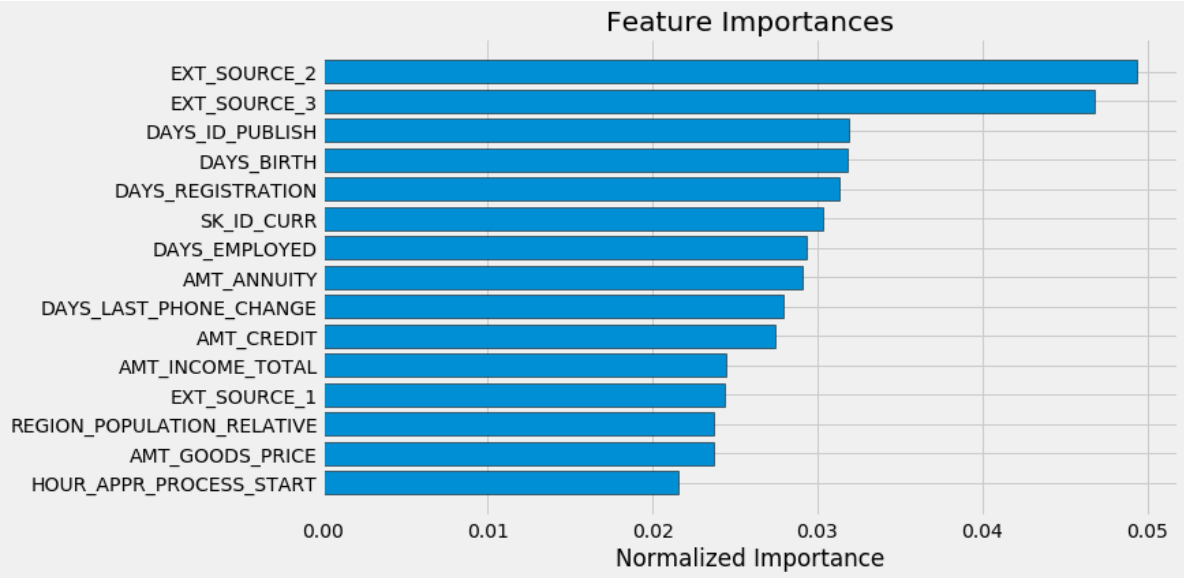
```
def plot_feature_importances(df):
    df = df.sort_values('importance', ascending = False).reset_index()
    df['importance_normalized'] = df['importance'] / df['importance'].sum()

    plt.figure(figsize = (10, 6))
    ax = plt.subplot()
    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))
    plt.xlabel('Normalized Importance'); plt.title('Feature Importances')
    plt.show()

    return df
```

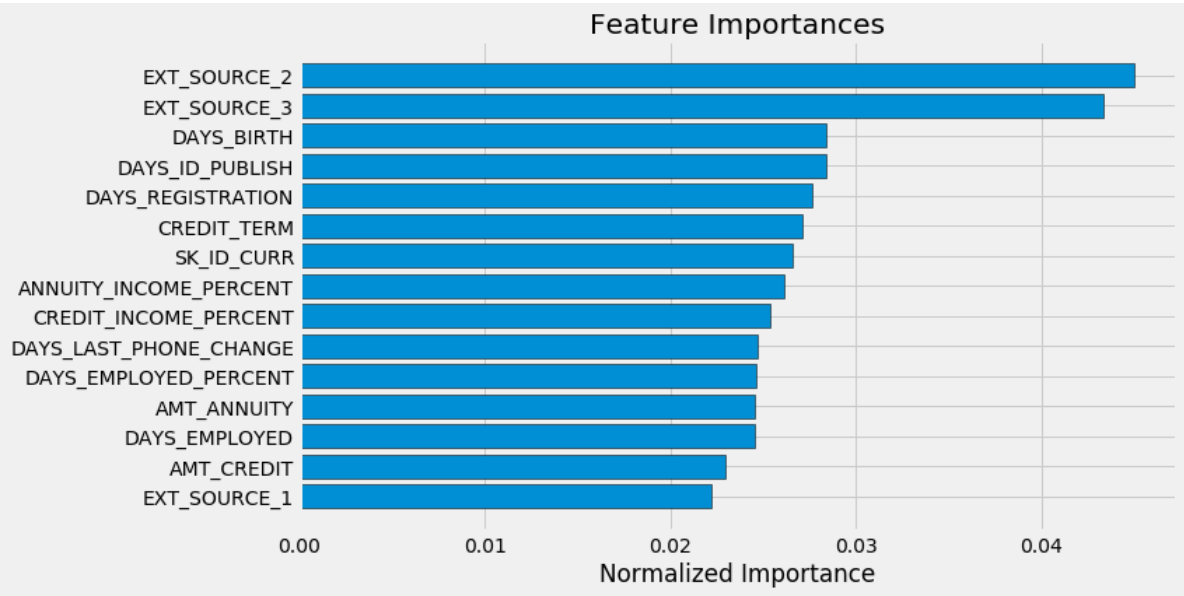
In [56]:

```
feature_importances_sorted = plot_feature_importances(feature_importances)
```



In [57]:

```
feature_importances_domain_sorted = plot_feature_importances(feature_importances_domain)
```



Domain 지식 기반으로 만든 변수들 모두 importances의 상위권에 올랐다.

Light Gradient Boosting Machine

In [59]:

```
from sklearn.model_selection import KFold
from sklearn.metrics import roc_auc_score
import lightgbm as lgb
import gc

def model(features, test_features, encoding = 'ohe', n_folds = 5):
    train_ids = features['SK_ID_CURR']
    test_ids = test_features['SK_ID_CURR']

    # Extract the labels for training
    labels = features['TARGET']

    # Remove the ids and target
    features = features.drop(columns = ['SK_ID_CURR', 'TARGET'])
    test_features = test_features.drop(columns = ['SK_ID_CURR'])


    # One Hot Encoding
    if encoding == 'ohe':
        features = pd.get_dummies(features)
        test_features = pd.get_dummies(test_features)

        # Align the dataframes by the columns
        features, test_features = features.align(test_features, join = 'inner', axis = 1)

        # No categorical indices to record
        cat_indices = 'auto'

    # Integer label encoding
    elif encoding == 'le':

        # Create a label encoder
        label_encoder = LabelEncoder()

        # List for storing categorical indices
        cat_indices = []

        # Iterate through each column
        for i, col in enumerate(features):
            if features[col].dtype == 'object':
                # Map the categorical features to integers
                features[col] = label_encoder.fit_transform(np.array(features[col].astype(str)).resh
                test_features[col] = label_encoder.transform(np.array(test_features[col].astype(str)

                # Record the categorical indices
                cat_indices.append(i)

    # Catch error if label encoding scheme is not valid
    else:
        raise ValueError("Encoding must be either 'ohe' or 'le'")

    print('Training Data Shape: ', features.shape)
    print('Testing Data Shape: ', test_features.shape)

    # Extract feature names
    feature_names = list(features.columns)

    # Convert to np arrays
    features = np.array(features)
```

```
test_features = np.array(test_features)

# Create the kfold object
k_fold = KFold(n_splits = n_folds, shuffle = True, random_state = 50)

# Empty array for feature importances
feature_importance_values = np.zeros(len(feature_names))

# Empty array for test predictions
test_predictions = np.zeros(test_features.shape[0])

# Empty array for out of fold validation predictions
out_of_fold = np.zeros(features.shape[0])

# Lists for recording validation and training scores
valid_scores = []
train_scores = []

# Iterate through each fold
for train_indices, valid_indices in k_fold.split(features):

    # Training data for the fold
    train_features, train_labels = features[train_indices], labels[train_indices]
    # Validation data for the fold
    valid_features, valid_labels = features[valid_indices], labels[valid_indices]

    # Create the model
    model = lgb.LGBMClassifier(n_estimators=10000, objective = 'binary',
                               class_weight = 'balanced', learning_rate = 0.05,
                               reg_alpha = 0.1, reg_lambda = 0.1,
                               subsample = 0.8, n_jobs = -1, random_state = 50)

    # Train the model
    model.fit(train_features, train_labels, eval_metric = 'auc',
              eval_set = [(valid_features, valid_labels), (train_features, train_labels)],
              eval_names = ['valid', 'train'], categorical_feature = cat_indices,
              early_stopping_rounds = 100, verbose = 200)

    # Record the best iteration
    best_iteration = model.best_iteration_

    # Record the feature importances
    feature_importance_values += model.feature_importances_ / k_fold.n_splits

    # Make predictions
    test_predictions += model.predict_proba(test_features, num_iteration = best_iteration)[:, 1

    # Record the out of fold predictions
    out_of_fold[valid_indices] = model.predict_proba(valid_features, num_iteration = best_iterat

    # Record the best score
    valid_score = model.best_score_['valid']['auc']
    train_score = model.best_score_['train']['auc']

    valid_scores.append(valid_score)
    train_scores.append(train_score)

    # Clean up memory
    gc.enable()
    del model, train_features, valid_features
    gc.collect()
```

```
    # Make the submission dataframe
    submission = pd.DataFrame({'SK_ID_CURR': test_ids, 'TARGET': test_predictions})

    # Make the feature importance dataframe
    feature_importances = pd.DataFrame({'feature': feature_names, 'importance': feature_importance_v

    # Overall validation score
    valid_auc = roc_auc_score(labels, out_of_fold)

    # Add the overall scores to the metrics
    valid_scores.append(valid_auc)
    train_scores.append(np.mean(train_scores))

    # Needed for creating dataframe of validation scores
    fold_names = list(range(n_folds))
    fold_names.append('overall')

    # Dataframe of validation scores
    metrics = pd.DataFrame({'fold': fold_names,
                            'train': train_scores,
                            'valid': valid_scores})

    return submission, feature_importances, metrics
```

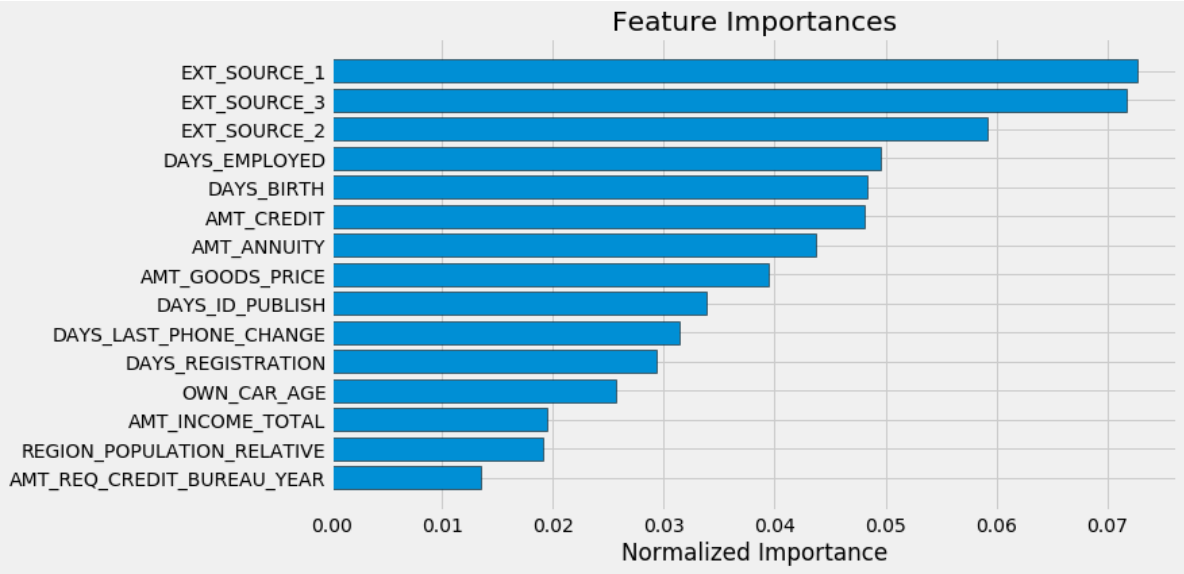
In [60]:

```
submission, fi, metrics = model(app_train, app_test)
print('Baseline metrics')
print(metrics)
```

```
Training Data Shape:  (307511, 239)
Testing Data Shape:  (48744, 239)
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.798723   train's binary_logloss: 0.547797       valid's auc:
0.755039        valid's binary_logloss: 0.563266
[400]   train's auc: 0.82838    train's binary_logloss: 0.518334       valid's auc:
0.755107        valid's binary_logloss: 0.545575
Early stopping, best iteration is:
[315]   train's auc: 0.816657   train's binary_logloss: 0.530116       valid's auc:
0.755215        valid's binary_logloss: 0.552627
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.798409   train's binary_logloss: 0.548179       valid's auc:
0.758332        valid's binary_logloss: 0.563587
[400]   train's auc: 0.828244   train's binary_logloss: 0.518308       valid's auc:
0.758563        valid's binary_logloss: 0.545588
Early stopping, best iteration is:
[317]   train's auc: 0.8169     train's binary_logloss: 0.529878       valid's auc:
0.758754        valid's binary_logloss: 0.552413
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.797648   train's binary_logloss: 0.549331       valid's auc:
0.763246        valid's binary_logloss: 0.564236
Early stopping, best iteration is:
[264]   train's auc: 0.808111   train's binary_logloss: 0.539063       valid's auc:
0.76363 valid's binary_logloss: 0.557905
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.798855   train's binary_logloss: 0.547952       valid's auc:
0.757131        valid's binary_logloss: 0.562234
Early stopping, best iteration is:
[280]   train's auc: 0.811887   train's binary_logloss: 0.535139       valid's auc:
0.757583        valid's binary_logloss: 0.554287
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.797918   train's binary_logloss: 0.548584       valid's auc:
0.758065        valid's binary_logloss: 0.564721
Early stopping, best iteration is:
[287]   train's auc: 0.811617   train's binary_logloss: 0.535146       valid's auc:
0.758344        valid's binary_logloss: 0.556636
Baseline metrics
      fold     train     valid
0        0  0.816657  0.755215
1        1  0.816900  0.758754
2        2  0.808111  0.763630
3        3  0.811887  0.757583
4        4  0.811617  0.758344
5  overall  0.813034  0.758705
```

In [61]:

```
fi_sorted = plot_feature_importances(fi)
```



In [62]:

```
submission.to_csv('baseline_lgb.csv', index = False)
```

Light GBM 썼을 때는 0.735 정도의 score이 나왔다고 한다.

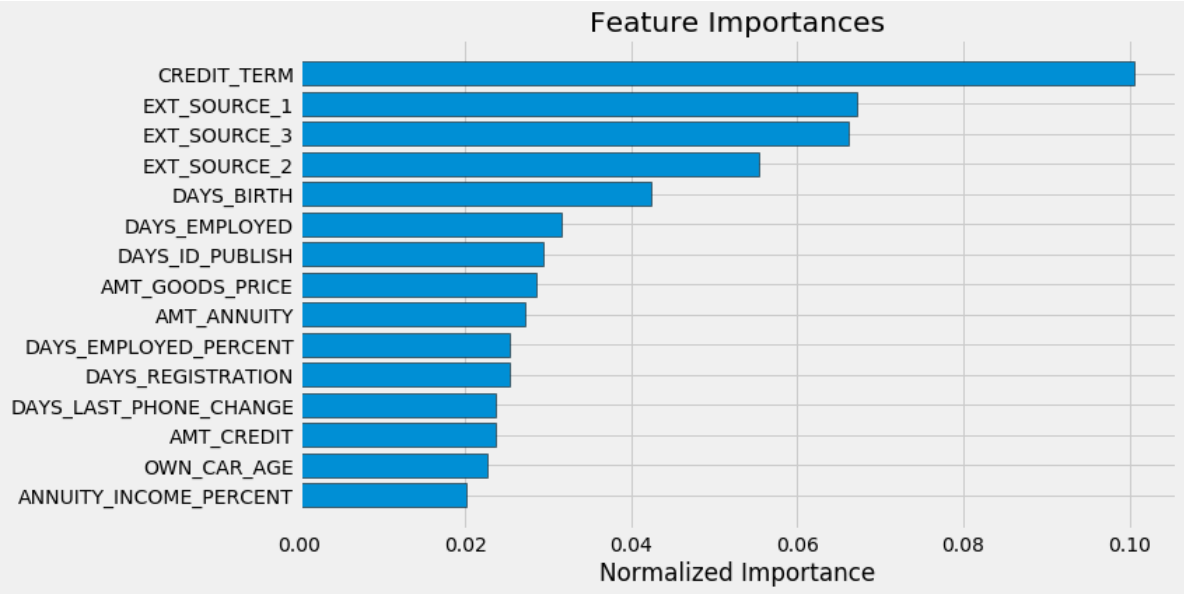
In [63]:

```
app_train_domain['TARGET'] = train_labels
submission_domain, fi_domain, metrics_domain = model(app_train_domain, app_test_domain)
print('Baseline with domain knowledge features metrics')
print(metrics_domain)
```

```
Training Data Shape:  (307511, 243)
Testing Data Shape:  (48744, 243)
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.804779   train's binary_logloss: 0.541283       valid's auc:
0.762511        valid's binary_logloss: 0.557227
Early stopping, best iteration is:
[268]   train's auc: 0.815523   train's binary_logloss: 0.530413       valid's auc:
0.763069        valid's binary_logloss: 0.550276
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.804016   train's binary_logloss: 0.542318       valid's auc:
0.765768        valid's binary_logloss: 0.557819
Early stopping, best iteration is:
[218]   train's auc: 0.807075   train's binary_logloss: 0.539112       valid's auc:
0.766062        valid's binary_logloss: 0.555952
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.8038     train's binary_logloss: 0.542856       valid's auc:
0.7703  valid's binary_logloss: 0.557925
[400]   train's auc: 0.834559   train's binary_logloss: 0.511454       valid's auc:
0.770511        valid's binary_logloss: 0.538558
Early stopping, best iteration is:
[383]   train's auc: 0.832138   train's binary_logloss: 0.514008       valid's auc:
0.77073 valid's binary_logloss: 0.54009
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.804603   train's binary_logloss: 0.541718       valid's auc:
0.765497        valid's binary_logloss: 0.556274
Early stopping, best iteration is:
[238]   train's auc: 0.8111     train's binary_logloss: 0.535149       valid's auc:
0.765884        valid's binary_logloss: 0.552351
Training until validation scores don't improve for 100 rounds
[200]   train's auc: 0.804782   train's binary_logloss: 0.541397       valid's auc:
0.765076        valid's binary_logloss: 0.558641
Early stopping, best iteration is:
[290]   train's auc: 0.819404   train's binary_logloss: 0.526653       valid's auc:
0.765249        valid's binary_logloss: 0.549657
Baseline with domain knowledge features metrics
      fold     train     valid
0        0  0.815523  0.763069
1        1  0.807075  0.766062
2        2  0.832138  0.770730
3        3  0.811100  0.765884
4        4  0.819404  0.765249
5  overall  0.817048  0.766186
```

In [64]:

```
fi_sorted = plot_feature_importances(fi_domain)
```



In [65]:

```
submission_domain.to_csv('baseline_lgb_domain_features.csv', index =  False)
```

이 경우 0.754의 점수를 얻었다고 한다. Feature engineering이 이번 대회의 가장 중요한 부분이었음을 알 수 있다.