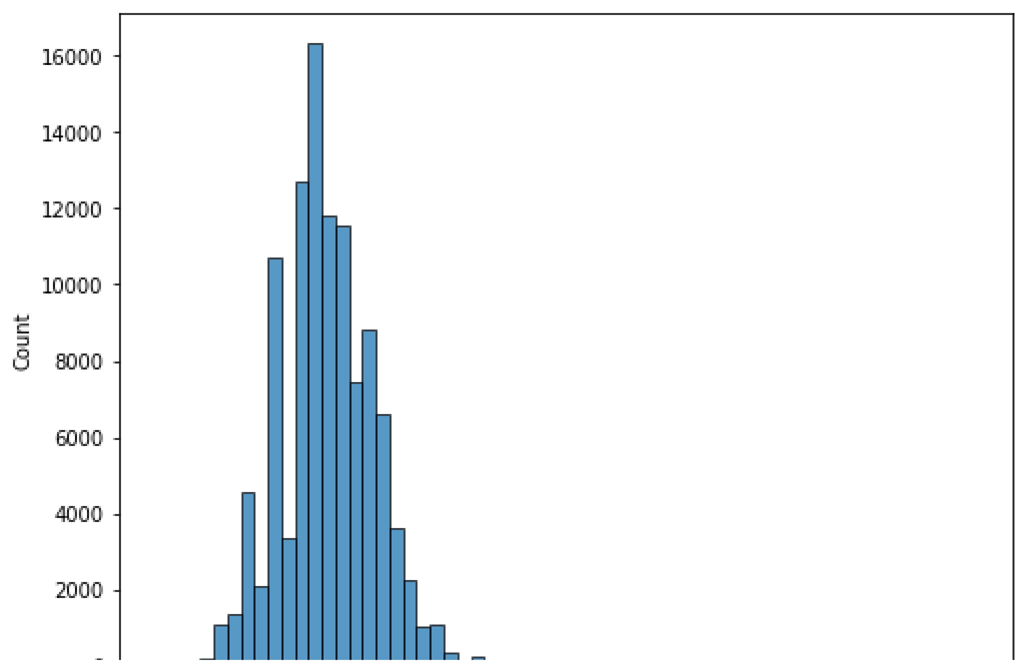
In [1]:
```python
import pandas as pd
import numpy as np
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
train_data = pd.read_csv('training.csv')
test_data = pd.read_csv('test.csv')
```

# Distribution of AMT_INCOME_TOTAL

In [38]:
```python
import matplotlib.pylab as plt
x = np.log(train_data.AMT_INCOME_TOTAL)
fig, ax = plt.subplots(figsize = (8,6))
ax = sns.histplot(x,bins = 60)
plt.show()
```




The distribution of AMT_INCOME_TOTAL after log looks like normal distribution.

# Missing Values

In [35]:
```
def missing_values(df):
        # Total missing values
        mis_sum = df.isnull().sum()
        # Percentage of missing values
        mis_val_percent = 100 * df.isnull().sum() / len(df)
        mis_val_table = pd.concat([mis_sum, mis_val_percent], axis=1)
        # Rename the columns
        new_mis_val_table = mis_val_table.rename(
        columns = {0 : 'Total missing', 1 : 'Percentage'})
        # Sort the table by percentage of missing descending
        new_mis_val_table = new_mis_val_table[
            new_mis_val_table.iloc[:,1] != 0].sort_values(
        'Percentage', ascending=False).round(1)
        #summary information
        print (str(new_mis_val_table.shape[0]) +
              " columns have missing values.")
        # Return the dataframe with missing information
        return new_mis_val_table
missing_values = missing_values(train_data)
print(missing_values.head(15))
```

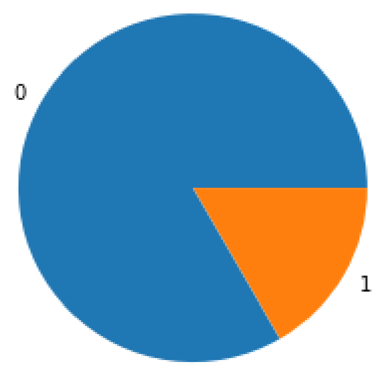
```
60 columns have missing values.
                          Total missing  Percentage
COMMONAREA_MEDI                   76069        70.4
COMMONAREA_AVG                    76069        70.4
COMMONAREA_MODE                   76069        70.4
NONLIVINGAPARTMENTS_MODE          75554        70.0
NONLIVINGAPARTMENTS_AVG           75554        70.0
NONLIVINGAPARTMENTS_MEDI          75554        70.0
LIVINGAPARTMENTS_AVG              74389        68.9
LIVINGAPARTMENTS_MODE             74389        68.9
LIVINGAPARTMENTS_MEDI             74389        68.9
FLOORSMIN_MEDI                    73866        68.4
FLOORSMIN_MODE                    73866        68.4
FLOORSMIN_AVG                     73866        68.4
YEARS_BUILD_AVG                   72466        67.1
YEARS_BUILD_MODE                  72466        67.1
YEARS_BUILD_MEDI                  72466        67.1
```

It can be seen from the table that some columns have over 70% missing values.Since it is impossible to know whether these columns will be useful to our model or not. All columsn will be kept for now.

## Distribution of Target

In [4]:
```
train_data['TARGET'].value_counts()
plt.pie(train_data['TARGET'].value_counts(),labels = [0,1])
```

Out[4]:
```
([<matplotlib.patches.Wedge at 0x7fc44a3747b8>,
  <matplotlib.patches.Wedge at 0x7fc44a374cf8>],
 [Text(-0.9526279098330699, 0.5500000594609756, '0'),
  Text(0.9526278840857079, -0.5500001040567061, '1')])
```




There are far more loans that were repaid on time than loans that were not repaid. So this is an imbalanced class.

# Feature Types

In [5]:
```
train_data.select_dtypes("object")
bool_features = train_data[train_data[train_data.columns[train_data.apply(1
numerical_features = np.setdiff1d(train_data.select_dtypes(include=["int64"
```

In [34]:
```
bool_features
```

Out[34]:
```
array(['TARGET', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'FLAG_EMP_PHONE',
       'FLAG_WORK_PHONE', 'FLAG_CONT_MOBILE', 'FLAG_PHONE', 'FLAG_EMAIL',
       'REG_REGION_NOT_LIVE_REGION', 'REG_REGION_NOT_WORK_REGION',
       'LIVE_REGION_NOT_WORK_REGION', 'REG_CITY_NOT_LIVE_CITY',
       'REG_CITY_NOT_WORK_CITY', 'LIVE_CITY_NOT_WORK_CITY',
       'FLAG_DOCUMENT_2', 'FLAG_DOCUMENT_3', 'FLAG_DOCUMENT_4',
       'FLAG_DOCUMENT_5', 'FLAG_DOCUMENT_6', 'FLAG_DOCUMENT_7',
       'FLAG_DOCUMENT_8', 'FLAG_DOCUMENT_9', 'FLAG_DOCUMENT_10',
       'FLAG_DOCUMENT_11', 'FLAG_DOCUMENT_12', 'FLAG_DOCUMENT_13',
       'FLAG_DOCUMENT_14', 'FLAG_DOCUMENT_15', 'FLAG_DOCUMENT_16',
       'FLAG_DOCUMENT_17', 'FLAG_DOCUMENT_18', 'FLAG_DOCUMENT_19',
       'FLAG_DOCUMENT_20', 'FLAG_DOCUMENT_21'], dtype=object)
```

In [6]:
```
categorical_features = np.setdiff1d(train_data.columns.values,numerical_fea
categorical_features = np.setdiff1d(categorical_features,bool_features)
categorical_features
```

Out[6]:
```
array(['CODE_GENDER', 'EMERGENCYSTATE_MODE', 'FONDKAPREMONT_MODE',
       'HOUSETYPE_MODE', 'NAME_CONTRACT_TYPE', 'NAME_EDUCATION_TYPE',
       'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'NAME_INCOME_TYPE',
       'NAME_TYPE_SUITE', 'OCCUPATION_TYPE', 'ORGANIZATION_TYPE',
       'WALLSMATERIAL_MODE', 'WEEKDAY_APPR_PROCESS_START'], dtype=object)
```

# Encoding categorical columns

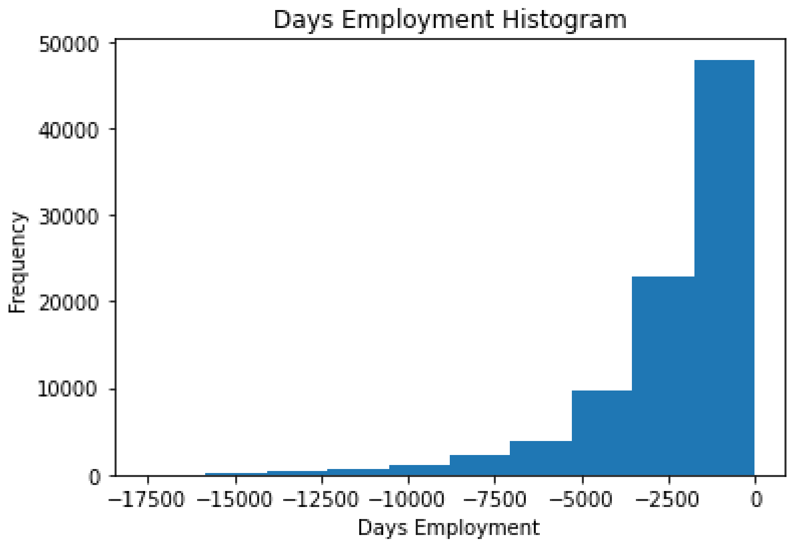
In [7]:
```
le = LabelEncoder()
for col in train_data:
    if train_data[col].dtype == 'object':
        # Train on the training data
        le.fit(train_data[col])
        # Transform both training and testing data
        train_data[col] = le.transform(train_data[col])
        test_data[col] = le.transform(test_data[col])
```

In [9]:
```
train_data['DAYS_EMPLOYED'].describe()
```

Out[9]:
```
count    108000.000000
mean      62031.638657
std      139705.501529
min      -17583.000000
25%       -2694.000000
50%       -1186.500000
75%        -298.000000
max      365243.000000
Name: DAYS_EMPLOYED, dtype: float64
```

365243 days is not reasonable since its 1000 years

In [10]:
```
train_data['DAYS_EMPLOYED'].replace({365243: np.nan}, inplace = True)
#check days employment now
train_data['DAYS_EMPLOYED'].plot.hist(title = 'Days Employment Histogram');
plt.xlabel('Days Employment');
```




In [11]:
```
test_data["DAYS_EMPLOYED"].replace({365243: np.nan}, inplace = True)
```

# Correlation between columns and Income

In [12]:
```python
# Find correlations with the target and sort
correlations = train_data.corr()['AMT_INCOME_TOTAL'].sort_values()

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

```
Most Positive Correlations:
 NONLIVINGAREA_AVG        0.076074
COMMONAREA_MODE          0.082129
LIVINGAPARTMENTS_MODE    0.090880
COMMONAREA_MEDI          0.095339
AMT_CREDIT               0.097151
AMT_GOODS_PRICE          0.098349
COMMONAREA_AVG           0.098351
LIVINGAPARTMENTS_MEDI    0.104011
LIVINGAPARTMENTS_AVG     0.106068
AMT_ANNUITY              0.119103
FLOORSMIN_MODE           0.139135
FLOORSMIN_MEDI           0.146638
FLOORSMIN_AVG            0.148520
AMT_INCOME_TOTAL         1.000000
FLAG_MOBIL                    NaN
Name: AMT_INCOME_TOTAL, dtype: float64

Most Negative Correlations:
 OWN_CAR_AGE                  -0.103104
REGION_RATING_CLIENT_W_CITY   -0.057794
NAME_EDUCATION_TYPE           -0.057024
REGION_RATING_CLIENT          -0.054221
ORGANIZATION_TYPE             -0.038228
HOUSETYPE_MODE                -0.033072
EMERGENCYSTATE_MODE           -0.033036
WALLSMATERIAL_MODE            -0.029727
FLAG_DOCUMENT_6               -0.028973
NAME_INCOME_TYPE              -0.026865
OCCUPATION_TYPE               -0.025384
FONDKAPREMONT_MODE            -0.021459
EXT_SOURCE_3                  -0.021119
FLAG_WORK_PHONE               -0.012375
NAME_FAMILY_STATUS            -0.011499
Name: AMT_INCOME_TOTAL, dtype: float64
```

In [31]:
```python
correlations = train_data.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:
 EMERGENCYSTATE_MODE            0.060247
FLAG_DOCUMENT_3                0.060367
FLAG_EMP_PHONE                 0.063485
NAME_INCOME_TYPE               0.066247
REG_CITY_NOT_WORK_CITY         0.069068
DAYS_ID_PUBLISH                0.069901
CODE_GENDER                    0.074942
DAYS_LAST_PHONE_CHANGE         0.076270
NAME_EDUCATION_TYPE            0.076871
REGION_RATING_CLIENT           0.081153
REGION_RATING_CLIENT_W_CITY    0.083135
DAYS_EMPLOYED                  0.102496
DAYS_BIRTH                     0.107622
TARGET                         1.000000
FLAG_MOBIL                          NaN
Name: TARGET, dtype: float64

Most Negative Correlations:
 EXT_SOURCE_3                  -0.237890
EXT_SOURCE_2                  -0.214573
EXT_SOURCE_1                  -0.210953
FLOORSMAX_AVG                 -0.060002
FLOORSMAX_MEDI                -0.059595
FLOORSMAX_MODE                -0.058879
AMT_GOODS_PRICE               -0.053854
REGION_POPULATION_RELATIVE    -0.053670
ELEVATORS_AVG                 -0.046322
ELEVATORS_MEDI                -0.045815
NAME_CONTRACT_TYPE            -0.045216
LIVINGAREA_AVG                -0.044108
FLOORSMIN_AVG                 -0.044047
TOTALAREA_MODE                -0.044000
FLOORSMIN_MEDI                -0.043995
Name: TARGET, dtype: float64
```

# Ext sources

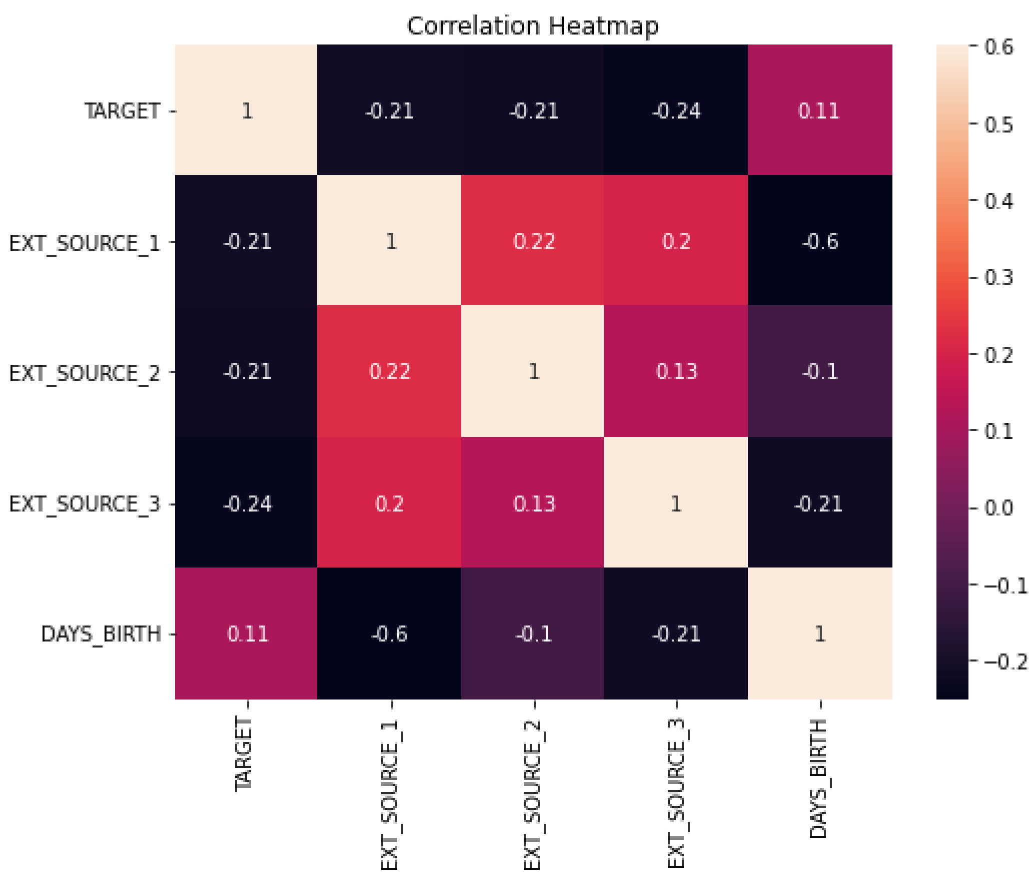
Since all three ext sources have negative correlation with Target, a visualized heatmap is plotted between ext_sources and days birth since days_birth has the largest value of correlation with target.

In [15]:
```
# Extract the EXT_SOURCE variables and show correlations
ext_data = train_data[['TARGET', 'EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURC
ext_data_corrs = ext_data.corr()
ext_data_corrs
```

Out[15]:

| | TARGET | EXT_SOURCE_1 | EXT_SOURCE_2 | EXT_SOURCE_3 | DAYS_BIRTH |
|---|---|---|---|---|---|
| **TARGET** | 1.000000 | -0.210953 | -0.214573 | -0.237890 | 0.107622 |
| **EXT_SOURCE_1** | -0.210953 | 1.000000 | 0.222042 | 0.200372 | -0.597115 |
| **EXT_SOURCE_2** | -0.214573 | 0.222042 | 1.000000 | 0.126572 | -0.101087 |
| **EXT_SOURCE_3** | -0.237890 | 0.200372 | 0.126572 | 1.000000 | -0.213923 |
| **DAYS_BIRTH** | 0.107622 | -0.597115 | -0.101087 | -0.213923 | 1.000000 |

In [16]:
```
plt.figure(figsize = (8, 6))
# Heatmap of correlations
sns.heatmap(ext_data_corrs, vmin = -0.25, annot = True, vmax = 0.6)
plt.title('Correlation Heatmap');
```




From row 1 we can see that EXT_SOURCE have negative correlations with target, when the value of the EXT_SOURCE increases, the client is more likely to repay the loan.

# Polynomial Features

In [17]:
```
poly = train_data[['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3', 'DAYS_BI
poly_test = test_data[['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3', 'DAY

#imputer for missing value
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy = 'median')
poly_target = poly['TARGET']

poly = poly.drop(columns = ['TARGET'])
# Need to impute missing values
poly = imputer.fit_transform(poly)
poly_test = imputer.transform(poly_test)

from sklearn.preprocessing import PolynomialFeatures

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

In [18]:
```
# Train the polynomial features
poly_transformer.fit(poly)

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

```
Polynomial Features shape:  (108000, 35)
```

All polynomial features names

In [19]:
```
poly_columns = poly_transformer.get_feature_names(input_features = ['EXT_SO
```

In [20]:
```python
# Create a dataframe of the features
poly_features = pd.DataFrame(poly, columns = poly_columns)
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                -0.261721
EXT_SOURCE_1 EXT_SOURCE_2 EXT_SOURCE_3   -0.258110
EXT_SOURCE_2^2 EXT_SOURCE_3              -0.240964
EXT_SOURCE_2 EXT_SOURCE_3^2              -0.234978
EXT_SOURCE_1 EXT_SOURCE_2                -0.225554
EXT_SOURCE_1 EXT_SOURCE_3                -0.222443
EXT_SOURCE_2                             -0.214422
EXT_SOURCE_1 EXT_SOURCE_2^2              -0.214253
EXT_SOURCE_3                             -0.207714
EXT_SOURCE_1 EXT_SOURCE_3^2              -0.206252
Name: TARGET, dtype: float64
EXT_SOURCE_2 DAYS_BIRTH                 0.213572
EXT_SOURCE_1 EXT_SOURCE_2 DAYS_BIRTH    0.213581
EXT_SOURCE_2 EXT_SOURCE_3 DAYS_BIRTH    0.247759
TARGET                                  1.000000
1                                            NaN
Name: TARGET, dtype: float64
```

By comparing with previous correlations with single feature, polynomial columns have greater absolute correlations with Target, which could use in feature selection later

# Add domain features

In [21]:
```
train_domain = train_data.copy()
test_domain = test_data.copy()

train_domain['AMT_INCOME_TOTAL'].replace({np.nan: 0}, inplace = True)
train_domain['AMT_CREDIT'].replace({np.nan: 0}, inplace = True)
train_domain['AMT_ANNUITY'].replace({np.nan: 0}, inplace = True)
train_domain['DAYS_EMPLOYED'].replace({np.nan: 0}, inplace = True)

train_domain['CREDIT_INCOME_PERCENT'] = train_domain['AMT_CREDIT']/train_do
train_domain['ANNUITY_INCOME_PERCENT'] = train_domain['AMT_ANNUITY']/train_
train_domain['CREDIT_TERM'] = train_domain['AMT_ANNUITY']/train_domain['AMT
train_domain['DAYS_EMPLOYED_PERCENT'] = train_domain['DAYS_EMPLOYED']/train

test_domain['CREDIT_INCOME_PERCENT'] = test_domain['AMT_CREDIT']/test_domai
test_domain['ANNUITY_INCOME_PERCENT'] = test_domain['AMT_ANNUITY'] / test_d
test_domain['CREDIT_TERM'] = test_domain['AMT_ANNUITY'] / test_domain['AMT_
test_domain['DAYS_EMPLOYED_PERCENT'] = test_domain['DAYS_EMPLOYED'] / test_
```

In [22]:
```
train_labels = train_data['TARGET']
```

In [24]:
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report,accuracy_score
```

Fit a random forest classifier and get an accuracy

In [25]:
```python
if 'TARGET' in train_domain:
    train_domain = train_domain.drop(['TARGET'],axis = 1)
else:
    train_domain = train_domain.copy()
if 'TARGET' in test_domain:
    y_test = test_domain.TARGET
    test_domain = test_domain.drop(['TARGET'],axis = 1)
else:
    test_domain = test_domain.copy()
domain_features_names = list(train_domain.columns)

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

train_domain = imputer.fit_transform(train_domain)
test_domain = imputer.transform(test_domain)

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

train_domain = scaler.fit_transform(train_domain)
test_domain = scaler.transform(test_domain)

random_forest_domain = RandomForestClassifier(n_estimators = 100, random_st

random_forest_domain.fit(train_domain, train_labels)

# Extract feature importances
feature_importance_values_domain = random_forest_domain.feature_importances
feature_importances_domain = pd.DataFrame({'feature': domain_features_names

# Make predictions on the test data
y_pred = random_forest_domain.predict(test_domain)
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 8 concurrent w
orkers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    3.1s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    7.7s finished
[Parallel(n_jobs=8)]: Using backend ThreadingBackend with 8 concurrent wo
rkers.
[Parallel(n_jobs=8)]: Done  34 tasks      | elapsed:    0.0s
[Parallel(n_jobs=8)]: Done 100 out of 100 | elapsed:    0.1s finished
```

In [26]:
```
from sklearn.metrics import classification_report,accuracy_score
print("Accuracy: "+str(accuracy_score(y_test, y_pred)))
print(classification_report(y_test,y_pred))
```

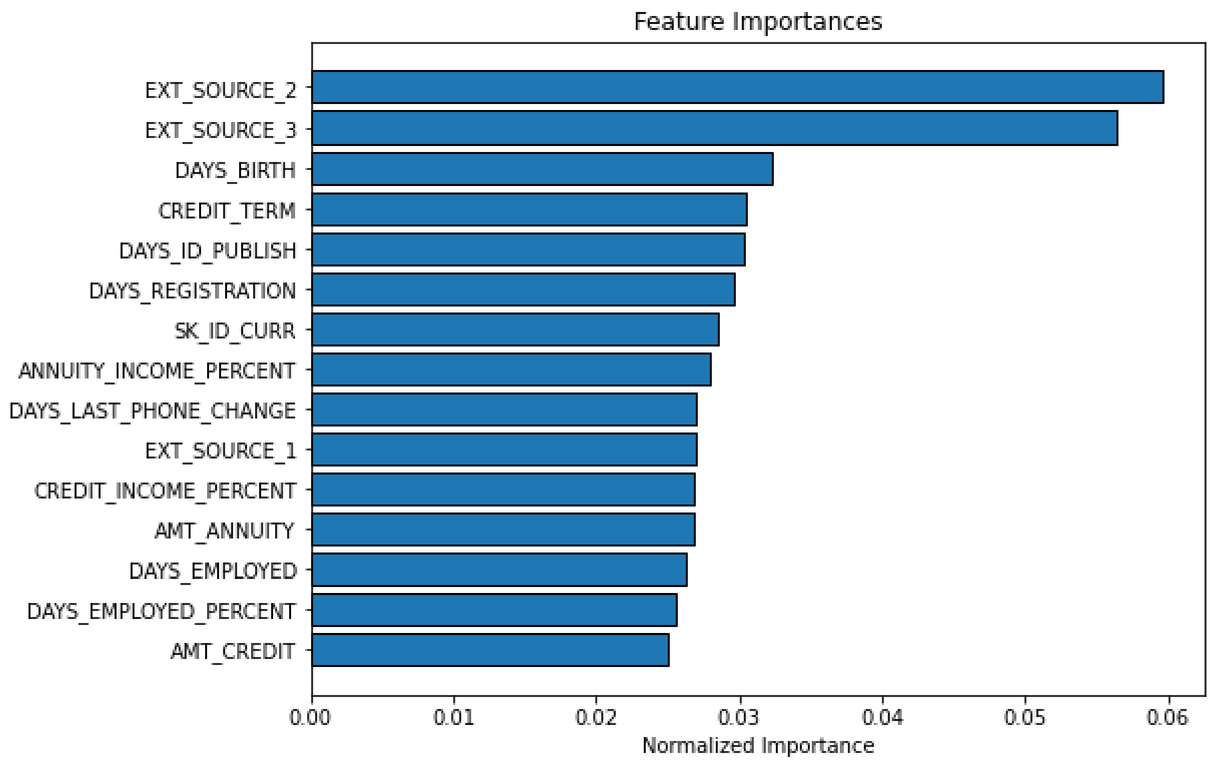
```
Accuracy: 0.837
              precision    recall  f1-score   support

           0       0.84      0.99      0.91     10000
           1       0.64      0.05      0.09      2000

    accuracy                           0.84     12000
   macro avg       0.74      0.52      0.50     12000
weighted avg       0.81      0.84      0.77     12000
```

In [27]:
```
def feature_importance(data):
    # Sort features according to importance
    data = data.sort_values('importance', ascending = False).reset_index()
    # Normalize the feature importances to add up to one
    data['new_importance'] = data['importance'] / data['importance'].sum()
    # Make a horizontal bar chart of feature importances
    plt.figure(figsize = (8, 6))
    ax = plt.subplot()
    # reverse the index
    ax.barh(list(reversed(list(data.index[:15]))),
            data['new_importance'].head(15),
            align = 'center', edgecolor = 'k')
    # Set the yticks and labels
    ax.set_yticks(list(reversed(list(data.index[:15]))))
    ax.set_yticklabels(data['feature'].head(15))
    # Plot labeling
    plt.xlabel('Normalized Importance'); plt.title('Feature Importances')
    plt.show()
    return data
```

In [28]:
```
feature_importances_domain_sorted = feature_importance(feature_importances_
```




It can be seen in the plot that some of the domain features such as annuity_income_percent has a greater correlation with target. So that the domain features is relatively useful. In my code I use select Kbest method to choose 30 best features to train.