Kaggle Home Credit Default Risk Competition

https://www.kaggle.com/c/home-credit-default-risk (https://www.kaggle.com/c/home-credit-default-risk)

This competition is sponsored by Home Credit (http://www.homecredit.net), whose mission is to provide a positive and safe borrowing experience to groups of people that traditional, mainstream banks and financial institutions typically refuse to serve.

In order to make lending decisions on applicants from this demographic, Home Credit needs an algorithm that will take as inputs various financial and personal information originally taken from a loan applicant's profile, and then compute a probability that the applicant will have trouble paying back the loan. This probability will be in the range [0.0, 1.0], where 1.0 represents a 100% certainty that the applicant will have repayment difficulties and 0.0 indicates that there is zero chance that the applicant will ever miss any payments. The algorithm will be tested and ranked on Kaggle based on a set predictions it makes for 48,744 individuals who previously borrowed from Home Credit.

Solution algorithms will be trained on a set of datapoints from 307,511 previous Home Credit borrowers. It is imperative that some portion, say 20%, of the training set is set aside to serve as a validation set. Alternatively, an algorithm such as K-Fold Cross Validation could be used.

To submit a solution on Kaggle, a CSV file must be produced that contains one header row, and 48,744 prediction rows, where each prediction row contains both a user ID, the `SKI_ID_CURR` column, and the probability, the `TARGET` column, of that user having repayment defficulties. The file must be formatted as follows:

```
SK_ID_CURR,TARGET
100001,0.1
100005,0.9
100013,0.2
etc.
```

Home Credit knows which borrowers in the test set were delinquent, and which ones never made a late payment. A good algorithm will need to predict a high probability of delinquent repayment for the majority of borrowers who did in fact make late payments (those whose `TARGET` value is 1 in the main table in the dataset). This algorithm will also need to predict a low probability of delinquent repayment for the majority of borrowers who never made a late payment (those whose `TARGET` value is 0 in the main table in the dataset).

Table of Contents

I. Data Exploration

In [47]:
```
# Import libraries necessary for this project.
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns

# Import supplementary visualizations code visuals.py
import visuals as vs

# Display matplotlib plots inline in this notebook.
%matplotlib inline
# Make plots display well on retina displays
%config InlineBackend.figure_format = 'retina'
# Set dpi of plots displayed inline
mpl.rcParams['figure.dpi'] = 300
# Configure style of plots
plt.style.use('fivethirtyeight')
# Make plots smaller
sns.set_context('paper')

# Allows the use of display() for dataframes.
from IPython.display import display
# Have all columns appear when dataframes are displayed.
pd.set_option('display.max_columns', None)
# Have 100 rows appear when a dataframe is displayed
pd.set_option('display.max_rows', 500)
# Display dimensions whenever a dataframe is printed out.
pd.set_option('display.show_dimensions', True)
```

Data Description

From https://www.kaggle.com/c/home-credit-default-risk/data (https://www.kaggle.com/c/home-credit-default-risk/data):

1. **application_{train|test}.csv**
    - This is the main table, broken into two files for Train (with TARGET) and Test (without TARGET).
    - Static data for all applications. One row represents one loan in our data sample.
2. **bureau.csv**
    - All client's previous credits provided by other financial institutions that were reported to Credit Bureau (for clients who have a loan in our sample).
    - For every loan in our sample, there are as many rows as number of credits the client had in Credit Bureau before the application date.
3. **bureau_balance.csv**

- Monthly balances of previous credits in Credit Bureau.
- This table has one row for each month of history of every previous credit reported to Credit Bureau – i.e the table has (#loans in sample # *of relative previous credits* # of months where we have some history observable for the previous credits) rows.

4. **previous_application.csv**

    - All previous applications for Home Credit loans of clients who have loans in our sample.
    - There is one row for each previous application related to loans in our data sample.

5. **POS_CASH_balance.csv**

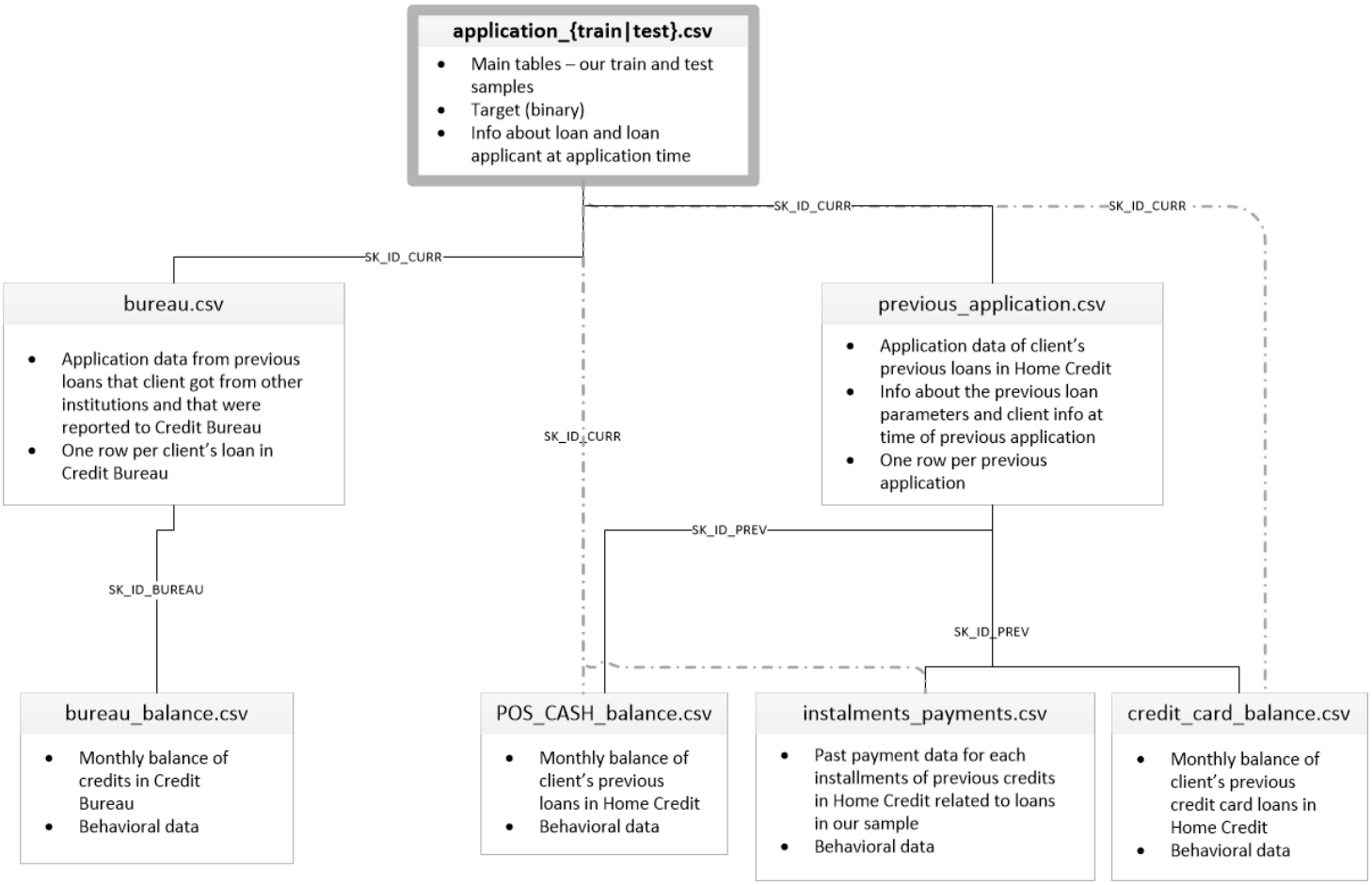
    - Monthly balance snapshots of previous POS (point of sales) and cash loans that the applicant had with Home Credit.
    - This table has one row for each month of history of every previous credit in Home Credit (consumer credit and cash loans) related to loans in our sample – i.e. the table has (#loans in sample # *of relative previous credits* # of months in which we have some history observable for the previous credits) rows.

6. **installments_payments.csv**

    - Repayment history for the previously disbursed credits in Home Credit related to the loans in our sample.
    - There is a) one row for every payment that was made plus b) one row each for missed payment.
    - One row is equivalent to one payment of one installment OR one installment corresponding to one payment of one previous Home Credit credit related to loans in our sample.

7. **credit_card_balance.csv**

    - Monthly balance snapshots of previous credit cards that the applicant has with Home Credit.
    - This table has one row for each month of history of every previous credit in Home Credit (consumer credit and cash loans) related to loans in our sample – i.e. the table has (#loans in sample # *of relative previous credit cards* # of months where we have some history observable for the previous credit card) rows.



In [2]:
```
# Load the main data tables
application_train_data = pd.read_csv("data/application_train.csv")
application_test_data = pd.read_csv("data/application_test.csv")
```

In [71]:
```
# Load the Bureau data table
bureau_data = pd.read_csv("data/bureau.csv")
```

In [5]:
```
# Load all other data tables
bureau_balance_data = pd.read_csv("data/bureau_balance.csv")
previous_application_data = pd.read_csv("data/previous_application.csv
")
POS_CASH_balance_data = pd.read_csv("data/POS_CASH_balance.csv")
installments_payments_data = pd.read_csv("data/installments_payments.c
sv")
credit_card_balance_data = pd.read_csv("data/credit_card_balance.csv")
```

1. Main Data Table (application_{train|test}.csv)

In [3]:
```
# Total number of entries in training group
print("Total number of entries in training group: {}".format(applicati
on_train_data.shape[0]))
```

```
Total number of entries in training group: 307511
```

In [4]:
```
# Total number of entries in test group
print("Total number of entries in test group: {}".format(application_t
est_data.shape[0]))
```

```
Total number of entries in test group: 48744
```

In [5]:
```
# Total number of features in the main (application) data table
print("Total number of features in main (application) data table: {}".
format(application_train_data.shape[1]))
```

```
Total number of features in main (application) data table: 122
```

The first two features in the main data table training group, `SK_ID_CURR` and `TARGET`, represent the borrower's ID number and target data (whether or not they made at least one late payment), respectively.

There are therefore 120 features in the main data table that can be used to predict a borrowers' targets.

In [183]:
```
# Display the first 500 records
display(application_train_data.head(n=500))
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_C
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	100008	0	Cash loans	M	N
6	100009	0	Cash loans	F	Y
7	100010	0	Cash loans	M	Y
8	100011	0	Cash loans	F	N

9	100012	0	Revolving loans	M	N
10	100014	0	Cash loans	F	N
11	100015	0	Cash loans	F	N
12	100016	0	Cash loans	F	N
13	100017	0	Cash loans	M	Y
14	100018	0	Cash loans	F	N
15	100019	0	Cash loans	M	Y
16	100020	0	Cash loans	M	N
17	100021	0	Revolving loans	F	N
18	100022	0	Revolving loans	F	N
19	100023	0	Cash loans	F	N
20	100024	0	Revolving loans	M	Y
21	100025	0	Cash loans	F	Y
22	100026	0	Cash loans	F	N
23	100027	0	Cash loans	F	N
24	100029	0	Cash loans	M	Y
25	100030	0	Cash loans	F	N
26	100031	1	Cash loans	F	N
27	100032	0	Cash loans	M	N
28	100033	0	Cash loans	M	Y

29	100034	0	Revolving loans	M	N
30	100035	0	Cash loans	F	N
31	100036	0	Cash loans	F	N
32	100037	0	Cash loans	F	N
33	100039	0	Cash loans	M	Y
34	100040	0	Cash loans	F	N
35	100041	0	Cash loans	F	N
36	100043	0	Cash loans	F	N
37	100044	0	Cash loans	M	N
38	100045	0	Cash loans	F	N
39	100046	0	Revolving loans	M	Y
40	100047	1	Cash loans	M	N
41	100048	0	Cash loans	F	N
42	100049	1	Cash loans	F	N
43	100050	0	Cash loans	F	N
44	100051	0	Cash loans	M	N
45	100052	0	Revolving loans	F	N
46	100053	0	Cash loans	F	N
47	100054	0	Cash loans	F	N
48	100055	0	Cash loans	F	N

49	100056	0	Cash loans	M	Y
50	100058	0	Revolving loans	F	N
51	100059	0	Cash loans	M	Y
52	100060	0	Cash loans	M	Y
53	100061	0	Cash loans	F	N
54	100062	0	Cash loans	M	Y
55	100063	0	Cash loans	F	N
56	100064	0	Cash loans	F	N
57	100068	0	Revolving loans	M	N
58	100069	0	Cash loans	M	Y
59	100070	0	Cash loans	M	Y
60	100071	0	Cash loans	F	N
61	100072	0	Cash loans	M	N
62	100073	0	Cash loans	M	Y
63	100075	0	Cash loans	F	N
64	100076	0	Cash loans	M	Y
65	100077	0	Cash loans	M	N
66	100078	0	Cash loans	F	Y
67	100079	0	Revolving loans	M	N
68	100080	0	Revolving loans	F	N
69	100081	0	Cash loans	M	Y
70	100082	0	Cash loans	M	N
71	100083	0	Cash loans	M	Y

72	100084	0	Cash loans	M	N
73	100085	0	Cash loans	M	N
74	100086	0	Cash loans	F	N
75	100087	0	Cash loans	F	N
76	100088	0	Revolving loans	F	N
77	100089	0	Cash loans	M	N
78	100093	0	Cash loans	F	N
79	100094	0	Cash loans	F	N
80	100095	0	Revolving loans	F	N
81	100096	1	Cash loans	F	N
82	100097	0	Cash loans	M	Y
83	100098	0	Revolving loans	M	Y
84	100099	0	Cash loans	F	N
85	100100	0	Cash loans	M	Y
86	100101	0	Cash loans	F	Y
87	100102	0	Cash loans	F	N
88	100103	0	Cash loans	F	N
89	100104	0	Cash loans	F	N
90	100105	0	Cash loans	F	N
91	100108	0	Cash loans	F	N

92	100110	0	Cash loans	M	Y
93	100111	0	Cash loans	F	Y
94	100112	1	Cash loans	M	Y
95	100113	0	Cash loans	M	Y
96	100114	0	Cash loans	F	N
97	100115	0	Cash loans	F	N
98	100116	0	Cash loans	F	N
99	100118	0	Cash loans	F	N
100	100119	0	Revolving loans	F	N
101	100120	0	Cash loans	F	N
102	100121	0	Cash loans	M	N
103	100122	0	Cash loans	F	N
104	100123	0	Cash loans	F	N
105	100124	0	Cash loans	M	N
106	100125	0	Cash loans	F	N
107	100126	0	Revolving loans	M	N
108	100127	0	Cash loans	M	N
109	100129	0	Revolving loans	F	Y
110	100130	1	Cash loans	F	N
111	100131	0	Cash loans	F	N
112	100132	0	Cash loans	F	N

113	100133	0	Cash loans	F	N
114	100134	0	Revolving loans	F	N
115	100135	0	Cash loans	M	N
116	100136	0	Cash loans	M	Y
117	100137	0	Cash loans	F	Y
118	100138	0	Cash loans	M	Y
119	100139	0	Cash loans	F	N
120	100140	0	Revolving loans	F	N
121	100142	0	Cash loans	M	Y
122	100143	0	Revolving loans	M	N
123	100144	0	Cash loans	F	N
124	100145	0	Cash loans	F	Y
125	100146	0	Cash loans	M	Y
126	100147	0	Cash loans	F	N
127	100148	0	Cash loans	F	N
128	100149	0	Cash loans	M	Y
129	100151	0	Cash loans	M	N
130	100152	0	Cash loans	F	N
131	100153	0	Cash loans	M	Y
132	100154	0	Revolving loans	M	N
133	100155	0	Cash loans	M	N

134	100156	0	Cash loans	F	Y
135	100157	0	Cash loans	M	N
136	100158	0	Cash loans	M	N
137	100159	0	Cash loans	F	N
138	100160	1	Cash loans	M	N
139	100161	0	Cash loans	F	N
140	100162	0	Cash loans	F	N
141	100163	0	Cash loans	F	N
142	100164	0	Cash loans	F	Y
143	100165	0	Cash loans	F	Y
144	100166	0	Cash loans	F	N
145	100167	0	Cash loans	F	N
146	100173	0	Cash loans	F	N
147	100174	0	Revolving loans	F	N
148	100175	0	Cash loans	F	N
149	100176	0	Cash loans	F	N
150	100177	0	Cash loans	M	N
151	100178	0	Cash loans	F	N
152	100179	0	Cash loans	F	Y
153	100180	0	Cash loans	F	N
154	100181	1	Cash loans	F	N
155	100182	0	Revolving loans	F	N

156	100183	0	Cash loans	F	N
157	100185	0	Cash loans	M	N
158	100186	0	Cash loans	F	N
159	100188	0	Cash loans	F	N
160	100189	0	Cash loans	F	N
161	100190	0	Cash loans	M	Y
162	100191	0	Cash loans	F	N
163	100192	1	Cash loans	F	N
164	100193	0	Cash loans	F	Y
165	100194	0	Revolving loans	F	N
166	100195	0	Cash loans	M	Y
167	100196	0	Cash loans	F	Y
168	100197	0	Cash loans	F	N
169	100198	0	Cash loans	M	Y
170	100199	0	Cash loans	F	N
171	100200	0	Cash loans	F	N
172	100201	0	Cash loans	F	N
173	100202	0	Cash loans	F	N
174	100203	0	Cash loans	F	N
175	100204	0	Cash loans	F	N
176	100205	0	Cash loans	F	N
177	100206	0	Cash loans	M	Y

178	100207	0	Cash loans	F	N
179	100208	0	Cash loans	M	N
180	100209	1	Revolving loans	M	N
181	100210	0	Cash loans	F	N
182	100211	0	Cash loans	M	N
183	100213	0	Cash loans	F	N
184	100214	1	Cash loans	F	N
185	100215	0	Cash loans	F	Y
186	100216	0	Cash loans	F	N
187	100217	0	Cash loans	F	N
188	100218	0	Cash loans	M	Y
189	100219	0	Cash loans	M	N
190	100220	0	Cash loans	F	N
191	100221	0	Cash loans	F	N
192	100224	0	Cash loans	F	Y
193	100225	0	Cash loans	M	Y
194	100226	0	Revolving loans	F	N
195	100227	0	Cash loans	M	N
196	100228	0	Cash loans	F	N
197	100229	0	Cash loans	F	N
198	100230	0	Revolving loans	F	N

199	100231	0	Cash loans	M	Y
200	100233	0	Cash loans	F	N
201	100234	0	Revolving loans	F	N
202	100235	0	Cash loans	F	N
203	100236	0	Cash loans	M	Y
204	100237	0	Cash loans	F	N
205	100238	0	Cash loans	M	Y
206	100239	0	Cash loans	F	N
207	100240	0	Cash loans	M	N
208	100242	0	Cash loans	M	Y
209	100243	0	Cash loans	M	N
210	100244	0	Cash loans	M	N
211	100246	1	Cash loans	F	N
212	100247	0	Cash loans	F	N
213	100248	0	Cash loans	F	N
214	100249	0	Cash loans	F	N
215	100250	0	Cash loans	F	Y
216	100251	0	Cash loans	M	Y
217	100252	0	Cash loans	F	N

218	100254	0	Cash loans	M	Y
219	100255	0	Cash loans	M	Y
220	100257	0	Cash loans	F	N
221	100258	0	Cash loans	F	N
222	100259	0	Cash loans	M	Y
223	100260	0	Cash loans	M	N
224	100261	0	Cash loans	F	N
225	100262	0	Cash loans	M	Y
226	100263	0	Cash loans	F	N
227	100264	0	Cash loans	F	N
228	100265	0	Cash loans	F	N
229	100266	0	Cash loans	F	N
230	100267	0	Cash loans	M	N
231	100268	0	Cash loans	F	N
232	100269	0	Cash loans	M	N
233	100270	0	Revolving loans	F	N
234	100272	0	Revolving loans	F	N
235	100273	1	Cash loans	F	N
236	100274	0	Cash loans	M	Y
237	100275	0	Cash loans	F	Y
238	100276	0	Cash loans	F	Y

239	100277	0	Cash loans	M	Y
240	100279	0	Cash loans	M	N
241	100281	0	Cash loans	F	N
242	100282	1	Revolving loans	F	Y
243	100283	0	Cash loans	F	N
244	100284	0	Cash loans	M	Y
245	100285	0	Revolving loans	F	N
246	100286	1	Cash loans	M	Y
247	100287	0	Cash loans	F	N
248	100288	0	Cash loans	F	N
249	100289	0	Cash loans	M	Y
250	100290	0	Cash loans	M	N
251	100291	0	Cash loans	F	N
252	100292	0	Cash loans	F	N
253	100293	0	Cash loans	F	N
254	100294	0	Cash loans	F	N
255	100295	1	Cash loans	M	Y
256	100296	0	Cash loans	M	N
257	100297	0	Cash loans	M	Y
258	100298	0	Cash loans	F	N
259	100299	0	Revolving loans	F	N

260	100300	1	Cash loans	M	N
261	100301	1	Cash loans	M	N
262	100302	0	Cash loans	F	N
263	100303	0	Cash loans	F	N
264	100304	0	Revolving loans	F	N
265	100305	0	Cash loans	M	Y
266	100307	0	Cash loans	M	N
267	100308	0	Cash loans	M	N
268	100309	0	Cash loans	F	N
269	100310	0	Cash loans	M	N
270	100313	0	Cash loans	F	N
271	100314	0	Cash loans	M	N
272	100315	0	Cash loans	F	Y
273	100316	0	Cash loans	F	N
274	100317	0	Cash loans	F	N
275	100318	0	Cash loans	F	N
276	100319	0	Cash loans	F	N
277	100320	0	Cash loans	M	Y
278	100321	0	Cash loans	M	Y
279	100322	0	Cash loans	M	Y
280	100323	0	Cash loans	M	Y
281	100324	0	Cash loans	F	Y

282	100325	0	Cash loans	F	N
283	100326	1	Cash loans	M	Y
284	100327	0	Cash loans	M	Y
285	100328	0	Cash loans	F	N
286	100329	0	Cash loans	F	Y
287	100332	0	Cash loans	F	N
288	100333	0	Cash loans	F	N
289	100334	0	Cash loans	M	N
290	100335	0	Cash loans	F	N
291	100336	1	Cash loans	F	Y
292	100337	0	Cash loans	F	N
293	100338	0	Cash loans	F	N
294	100339	0	Revolving loans	M	N
295	100340	0	Cash loans	F	N
296	100341	0	Cash loans	M	Y
297	100342	0	Cash loans	M	N
298	100343	0	Cash loans	M	Y
299	100344	0	Cash loans	M	N
300	100345	0	Cash loans	F	N
301	100346	0	Cash loans	F	N

302	100347	0	Cash loans	F	N
303	100348	0	Cash loans	F	N
304	100349	0	Cash loans	F	N
305	100350	0	Cash loans	F	N
306	100351	0	Cash loans	F	N
307	100353	0	Cash loans	F	N
308	100354	0	Revolving loans	F	N
309	100355	0	Cash loans	M	N
310	100356	0	Cash loans	F	N
311	100357	0	Revolving loans	F	N
312	100359	0	Cash loans	M	Y
313	100360	0	Revolving loans	F	N
314	100361	0	Cash loans	M	N
315	100362	0	Cash loans	F	N
316	100363	0	Cash loans	F	Y
317	100364	0	Cash loans	F	N
318	100365	0	Cash loans	F	N
319	100366	0	Revolving loans	M	Y
320	100368	0	Cash loans	F	N
321	100369	0	Cash loans	F	N

322	100370	0	Cash loans	F	N
323	100371	0	Cash loans	F	Y
324	100372	0	Cash loans	F	N
325	100373	0	Cash loans	F	N
326	100374	0	Cash loans	F	Y
327	100375	0	Cash loans	M	N
328	100376	0	Cash loans	M	Y
329	100377	0	Cash loans	M	N
330	100378	0	Cash loans	F	Y
331	100379	0	Cash loans	M	Y
332	100380	0	Cash loans	F	N
333	100381	0	Cash loans	F	N
334	100383	0	Cash loans	M	Y
335	100387	0	Cash loans	M	Y
336	100388	0	Cash loans	F	N
337	100389	0	Cash loans	M	Y
338	100390	0	Cash loans	F	N
339	100391	0	Cash loans	F	N
340	100392	0	Revolving loans	M	Y
341	100393	0	Cash loans	M	Y
342	100394	0	Cash loans	M	N
343	100395	0	Cash loans	M	Y

344	100396	1	Cash loans	M	N
345	100397	0	Cash loans	F	N
346	100398	0	Cash loans	F	N
347	100400	0	Cash loans	F	N
348	100401	1	Cash loans	F	N
349	100402	0	Cash loans	F	N
350	100403	0	Cash loans	M	Y
351	100405	0	Cash loans	M	Y
352	100406	0	Cash loans	F	N
353	100407	0	Cash loans	F	N
354	100408	0	Cash loans	F	N
355	100409	0	Cash loans	M	N
356	100410	0	Cash loans	F	Y
357	100411	0	Cash loans	M	Y
358	100412	0	Cash loans	F	Y
359	100413	0	Cash loans	M	N
360	100414	0	Cash loans	F	N
361	100415	0	Cash loans	F	N
362	100417	0	Cash loans	F	N
363	100418	0	Cash loans	F	N

364	100419	0	Cash loans	F	N
365	100420	0	Cash loans	F	N
366	100421	0	Cash loans	M	Y
367	100423	0	Cash loans	F	Y
368	100424	1	Cash loans	M	N
369	100425	0	Cash loans	F	Y
370	100427	0	Cash loans	F	N
371	100428	0	Cash loans	F	Y
372	100429	0	Cash loans	M	N
373	100430	0	Cash loans	F	N
374	100431	0	Cash loans	F	N
375	100432	0	Cash loans	F	N
376	100433	0	Cash loans	F	N
377	100434	0	Cash loans	F	N
378	100435	0	Cash loans	F	N
379	100436	0	Revolving loans	F	N
380	100437	0	Cash loans	F	N
381	100439	1	Cash loans	M	N
382	100440	0	Cash loans	F	N
383	100441	0	Cash loans	F	N
384	100442	0	Cash loans	F	N

385	100443	0	Cash loans	F	N
386	100448	0	Cash loans	F	N
387	100449	0	Cash loans	F	N
388	100451	0	Cash loans	F	N
389	100452	1	Cash loans	M	N
390	100453	0	Cash loans	M	Y
391	100454	0	Revolving loans	M	Y
392	100455	0	Cash loans	F	Y
393	100456	0	Cash loans	M	Y
394	100457	0	Revolving loans	F	Y
395	100458	0	Cash loans	F	N
396	100459	0	Revolving loans	M	Y
397	100460	0	Revolving loans	F	N
398	100461	0	Cash loans	F	Y
399	100462	0	Cash loans	M	N
400	100463	0	Cash loans	F	N
401	100464	0	Cash loans	M	N
402	100465	0	Cash loans	M	N
403	100467	0	Cash loans	F	N
404	100468	0	Cash loans	F	Y
405	100469	0	Revolving loans	M	Y

406	100470	0	Cash loans	F	Y
407	100471	0	Cash loans	F	Y
408	100472	1	Cash loans	M	Y
409	100473	0	Cash loans	M	Y
410	100474	0	Revolving loans	M	Y
411	100475	0	Revolving loans	F	Y
412	100476	0	Cash loans	M	Y
413	100477	1	Cash loans	M	Y
414	100478	0	Cash loans	F	N
415	100479	0	Revolving loans	M	Y
416	100480	0	Cash loans	M	N
417	100481	0	Revolving loans	M	Y
418	100482	0	Cash loans	F	Y
419	100485	1	Revolving loans	M	Y
420	100486	0	Cash loans	F	N
421	100487	0	Cash loans	M	Y
422	100488	0	Cash loans	F	Y
423	100489	0	Cash loans	M	N
424	100490	1	Cash loans	M	N
425	100491	0	Cash loans	M	Y
426	100492	0	Cash loans	F	N

427	100493	0	Cash loans	F	Y
428	100495	0	Cash loans	F	Y
429	100496	0	Cash loans	F	N
430	100497	0	Revolving loans	M	N
431	100498	0	Cash loans	M	N
432	100499	0	Cash loans	F	Y
433	100500	0	Cash loans	F	N
434	100501	0	Cash loans	M	N
435	100502	0	Cash loans	F	Y
436	100503	0	Cash loans	F	N
437	100504	0	Cash loans	F	N
438	100505	0	Cash loans	M	Y
439	100506	0	Cash loans	M	Y
440	100507	0	Cash loans	M	N
441	100508	0	Cash loans	F	Y
442	100509	0	Cash loans	F	N
443	100511	0	Cash loans	F	N
444	100513	0	Cash loans	F	N
445	100514	0	Cash loans	M	N
446	100515	0	Cash loans	F	N

447	100516	0	Cash loans	F	N
448	100518	0	Revolving loans	F	N
449	100519	0	Cash loans	F	N
450	100520	0	Revolving loans	F	N
451	100521	0	Revolving loans	F	N
452	100522	0	Cash loans	F	Y
453	100523	0	Cash loans	M	Y
454	100524	0	Cash loans	M	N
455	100525	0	Cash loans	M	N
456	100526	0	Cash loans	M	Y
457	100527	0	Revolving loans	F	N
458	100528	0	Cash loans	F	N
459	100529	0	Cash loans	F	N
460	100530	0	Cash loans	F	Y
461	100531	0	Cash loans	F	Y
462	100532	0	Cash loans	M	Y
463	100533	0	Revolving loans	M	N
464	100534	0	Cash loans	F	N
465	100535	0	Cash loans	M	Y
466	100536	0	Cash loans	F	Y

467	100537	0	Cash loans	F	N
468	100538	0	Cash loans	M	Y
469	100539	0	Cash loans	M	N
470	100540	1	Cash loans	F	N
471	100541	1	Cash loans	F	Y
472	100542	0	Cash loans	F	N
473	100543	0	Cash loans	F	N
474	100544	0	Cash loans	F	N
475	100545	0	Cash loans	F	N
476	100546	0	Cash loans	M	N
477	100547	1	Cash loans	M	Y
478	100549	0	Cash loans	F	N
479	100550	0	Cash loans	F	Y
480	100554	0	Cash loans	F	Y
481	100555	0	Cash loans	F	N
482	100556	0	Cash loans	F	N
483	100557	0	Cash loans	F	Y
484	100558	0	Revolving loans	M	N
485	100559	0	Cash loans	F	Y
486	100560	0	Cash loans	F	N
487	100562	0	Revolving loans	M	N

488	100563	0	Cash loans	F	N
489	100564	0	Cash loans	M	N
490	100565	0	Cash loans	F	N
491	100566	0	Cash loans	F	N
492	100567	1	Revolving loans	M	Y
493	100570	0	Cash loans	F	Y
494	100571	0	Cash loans	M	Y
495	100572	0	Cash loans	F	N
496	100573	0	Cash loans	F	Y
497	100574	0	Cash loans	M	Y
498	100575	0	Cash loans	M	Y
499	100576	0	Cash loans	F	N

500 rows × 122 columns

In [191]:
```
# Display the above data sample table, but with axes transposed and li
mited to 5 records, so that the table can
# be included in the project's writeup.
display(application_train_data.head(n=5).transpose())
```

	0	1	2	
SK_ID_CURR	100002	100003	100004	1(
TARGET	1	0	0	0
NAME_CONTRACT_TYPE	Cash loans	Cash loans	Revolving loans	C
CODE_GENDER	M	F	M	F
FLAG_OWN_CAR	N	N	Y	N
FLAG_OWN_REALTY	Y	N	Y	Y
CNT_CHILDREN	0	0	0	0

AMT_INCOME_TOTAL	202500	270000	67500	13
AMT_CREDIT	406598	1.2935e+06	135000	31
AMT_ANNUITY	24700.5	35698.5	6750	29
AMT_GOODS_PRICE	351000	1.1295e+06	135000	29
NAME_TYPE_SUITE	Unaccompanied	Family	Unaccompanied	Un
NAME_INCOME_TYPE	Working	State servant	Working	W
NAME_EDUCATION_TYPE	Secondary / secondary special	Higher education	Secondary / secondary special	Se se sp
NAME_FAMILY_STATUS	Single / not married	Married	Single / not married	Ci
NAME_HOUSING_TYPE	House / apartment	House / apartment	House / apartment	Ho ap
REGION_POPULATION_RELATIVE	0.018801	0.003541	0.010032	0.
DAYS_BIRTH	-9461	-16765	-19046	-1
DAYS_EMPLOYED	-637	-1188	-225	-3
DAYS_REGISTRATION	-3648	-1186	-4260	-9
DAYS_ID_PUBLISH	-2120	-291	-2531	-2
OWN_CAR_AGE	NaN	NaN	26	Na
FLAG_MOBIL	1	1	1	1
FLAG_EMP_PHONE	1	1	1	1
FLAG_WORK_PHONE	0	0	1	0
FLAG_CONT_MOBILE	1	1	1	1
FLAG_PHONE	1	1	1	0
FLAG_EMAIL	0	0	0	0
OCCUPATION_TYPE	Laborers	Core staff	Laborers	La
CNT_FAM_MEMBERS	1	2	1	2
REGION_RATING_CLIENT	2	1	2	2
REGION_RATING_CLIENT_W_CITY	2	1	2	2
WEEKDAY_APPR_PROCESS_START	WEDNESDAY	MONDAY	MONDAY	W
HOUR_APPR_PROCESS_START	10	11	9	17
REG_REGION_NOT_LIVE_REGION	0	0	0	0

REG_REGION_NOT_WORK_REGION	0	0	0	0
LIVE_REGION_NOT_WORK_REGION	0	0	0	0
REG_CITY_NOT_LIVE_CITY	0	0	0	0
REG_CITY_NOT_WORK_CITY	0	0	0	0
LIVE_CITY_NOT_WORK_CITY	0	0	0	0
ORGANIZATION_TYPE	Business Entity Type 3	School	Government	Bu Ty
EXT_SOURCE_1	0.083037	0.311267	NaN	Na
EXT_SOURCE_2	0.262949	0.622246	0.555912	0.
EXT_SOURCE_3	0.139376	NaN	0.729567	Na
APARTMENTS_AVG	0.0247	0.0959	NaN	Na
BASEMENTAREA_AVG	0.0369	0.0529	NaN	Na
YEARS_BEGINEXPLUATATION_AVG	0.9722	0.9851	NaN	Na
YEARS_BUILD_AVG	0.6192	0.796	NaN	Na
COMMONAREA_AVG	0.0143	0.0605	NaN	Na
ELEVATORS_AVG	0	0.08	NaN	Na
ENTRANCES_AVG	0.069	0.0345	NaN	Na
FLOORSMAX_AVG	0.0833	0.2917	NaN	Na
FLOORSMIN_AVG	0.125	0.3333	NaN	Na
LANDAREA_AVG	0.0369	0.013	NaN	Na
LIVINGAPARTMENTS_AVG	0.0202	0.0773	NaN	Na
LIVINGAREA_AVG	0.019	0.0549	NaN	Na
NONLIVINGAPARTMENTS_AVG	0	0.0039	NaN	Na
NONLIVINGAREA_AVG	0	0.0098	NaN	Na
APARTMENTS_MODE	0.0252	0.0924	NaN	Na
BASEMENTAREA_MODE	0.0383	0.0538	NaN	Na
YEARS_BEGINEXPLUATATION_MODE	0.9722	0.9851	NaN	Na
YEARS_BUILD_MODE	0.6341	0.804	NaN	Na
COMMONAREA_MODE	0.0144	0.0497	NaN	Na
ELEVATORS_MODE	0	0.0806	NaN	Na
ENTRANCES_MODE	0.069	0.0345	NaN	Na
FLOORSMAX_MODE	0.0833	0.2917	NaN	Na

FLOORSMIN_MODE	0.125	0.3333	NaN	Na
LANDAREA_MODE	0.0377	0.0128	NaN	Na
LIVINGAPARTMENTS_MODE	0.022	0.079	NaN	Na
LIVINGAREA_MODE	0.0198	0.0554	NaN	Na
NONLIVINGAPARTMENTS_MODE	0	0	NaN	Na
NONLIVINGAREA_MODE	0	0	NaN	Na
APARTMENTS_MEDI	0.025	0.0968	NaN	Na
BASEMENTAREA_MEDI	0.0369	0.0529	NaN	Na
YEARS_BEGINEXPLUATATION_MEDI	0.9722	0.9851	NaN	Na
YEARS_BUILD_MEDI	0.6243	0.7987	NaN	Na
COMMONAREA_MEDI	0.0144	0.0608	NaN	Na
ELEVATORS_MEDI	0	0.08	NaN	Na
ENTRANCES_MEDI	0.069	0.0345	NaN	Na
FLOORSMAX_MEDI	0.0833	0.2917	NaN	Na
FLOORSMIN_MEDI	0.125	0.3333	NaN	Na
LANDAREA_MEDI	0.0375	0.0132	NaN	Na
LIVINGAPARTMENTS_MEDI	0.0205	0.0787	NaN	Na
LIVINGAREA_MEDI	0.0193	0.0558	NaN	Na
NONLIVINGAPARTMENTS_MEDI	0	0.0039	NaN	Na
NONLIVINGAREA_MEDI	0	0.01	NaN	Na
FONDKAPREMONT_MODE	reg oper account	reg oper account	NaN	Na
HOUSETYPE_MODE	block of flats	block of flats	NaN	Na
TOTALAREA_MODE	0.0149	0.0714	NaN	Na
WALLSMATERIAL_MODE	Stone, brick	Block	NaN	Na
EMERGENCYSTATE_MODE	No	No	NaN	Na
OBS_30_CNT_SOCIAL_CIRCLE	2	1	0	2
DEF_30_CNT_SOCIAL_CIRCLE	2	0	0	0
OBS_60_CNT_SOCIAL_CIRCLE	2	1	0	2
DEF_60_CNT_SOCIAL_CIRCLE	2	0	0	0
DAYS_LAST_PHONE_CHANGE	-1134	-828	-815	-6

FLAG_DOCUMENT_2	0	0	0	0
FLAG_DOCUMENT_3	1	1	0	1
FLAG_DOCUMENT_4	0	0	0	0
FLAG_DOCUMENT_5	0	0	0	0
FLAG_DOCUMENT_6	0	0	0	0
FLAG_DOCUMENT_7	0	0	0	0
FLAG_DOCUMENT_8	0	0	0	0
FLAG_DOCUMENT_9	0	0	0	0
FLAG_DOCUMENT_10	0	0	0	0
FLAG_DOCUMENT_11	0	0	0	0
FLAG_DOCUMENT_12	0	0	0	0
FLAG_DOCUMENT_13	0	0	0	0
FLAG_DOCUMENT_14	0	0	0	0
FLAG_DOCUMENT_15	0	0	0	0
FLAG_DOCUMENT_16	0	0	0	0
FLAG_DOCUMENT_17	0	0	0	0
FLAG_DOCUMENT_18	0	0	0	0
FLAG_DOCUMENT_19	0	0	0	0
FLAG_DOCUMENT_20	0	0	0	0
FLAG_DOCUMENT_21	0	0	0	0
AMT_REQ_CREDIT_BUREAU_HOUR	0	0	0	Na
AMT_REQ_CREDIT_BUREAU_DAY	0	0	0	Na
AMT_REQ_CREDIT_BUREAU_WEEK	0	0	0	Na
AMT_REQ_CREDIT_BUREAU_MON	0	0	0	Na
AMT_REQ_CREDIT_BUREAU_QRT	0	0	0	Na
AMT_REQ_CREDIT_BUREAU_YEAR	1	0	0	Na

122 rows × 5 columns

In [7]:
```
# Display a statistical description of the numerical features, along w
ith all features that
# have alredy been one-hot encoded, in the main (application) data tab
le.
display(application_train_data.describe())
```

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_C
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.0751
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.99026
std	102790.175348	0.272419	0.722121	2.371231e+05	4.0249(
min	100002.000000	0.000000	0.000000	2.565000e+04	4.5000(
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.7000(
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.1353
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.0865(
max	456255.000000	1.000000	19.000000	1.170000e+08	4.0500(

8 rows × 106 columns

In [187]:
```
# Display the above statistical description table, but with axes inver
ted, so that the table can
# be included in the project's writeup.
display(application_train_data.describe().transpose())
```

	count	mean	std	
SK_ID_CURR	307511.0	278180.518577	102790.175348	1.000(
TARGET	307511.0	0.080729	0.272419	0.000(
CNT_CHILDREN	307511.0	0.417052	0.722121	0.000(
AMT_INCOME_TOTAL	307511.0	168797.919297	237123.146279	2.565(
AMT_CREDIT	307511.0	599025.999706	402490.776996	4.500(
AMT_ANNUITY	307499.0	27108.573909	14493.737315	1.615
AMT_GOODS_PRICE	307233.0	538396.207429	369446.460540	4.050(
REGION_POPULATION_RELATIVE	307511.0	0.020868	0.013831	2.900(
DAYS_BIRTH	307511.0	-16036.995067	4363.988632	-2.522
DAYS_EMPLOYED	307511.0	63815.045904	141275.766519	-1.791
DAYS_REGISTRATION	307511.0	-4986.120328	3522.886321	-2.467
DAYS_ID_PUBLISH	307511.0	-2994.202373	1509.450419	-7.197

OWN_CAR_AGE	104582.0	12.061091	11.944812	0.000(
FLAG_MOBIL	307511.0	0.999997	0.001803	0.000(
FLAG_EMP_PHONE	307511.0	0.819889	0.384280	0.000(
FLAG_WORK_PHONE	307511.0	0.199368	0.399526	0.000(
FLAG_CONT_MOBILE	307511.0	0.998133	0.043164	0.000(
FLAG_PHONE	307511.0	0.281066	0.449521	0.000(
FLAG_EMAIL	307511.0	0.056720	0.231307	0.000(
CNT_FAM_MEMBERS	307509.0	2.152665	0.910682	1.000(
REGION_RATING_CLIENT	307511.0	2.052463	0.509034	1.000(
REGION_RATING_CLIENT_W_CITY	307511.0	2.031521	0.502737	1.000(
HOUR_APPR_PROCESS_START	307511.0	12.063419	3.265832	0.000(
REG_REGION_NOT_LIVE_REGION	307511.0	0.015144	0.122126	0.000(
REG_REGION_NOT_WORK_REGION	307511.0	0.050769	0.219526	0.000(
LIVE_REGION_NOT_WORK_REGION	307511.0	0.040659	0.197499	0.000(
REG_CITY_NOT_LIVE_CITY	307511.0	0.078173	0.268444	0.000(
REG_CITY_NOT_WORK_CITY	307511.0	0.230454	0.421124	0.000(
LIVE_CITY_NOT_WORK_CITY	307511.0	0.179555	0.383817	0.000(
EXT_SOURCE_1	134133.0	0.502130	0.211062	1.456(
EXT_SOURCE_2	306851.0	0.514393	0.191060	8.173(
EXT_SOURCE_3	246546.0	0.510853	0.194844	5.272(
APARTMENTS_AVG	151450.0	0.117440	0.108240	0.000(
BASEMENTAREA_AVG	127568.0	0.088442	0.082438	0.000(
YEARS_BEGINEXPLUATATION_AVG	157504.0	0.977735	0.059223	0.000(
YEARS_BUILD_AVG	103023.0	0.752471	0.113280	0.000(
COMMONAREA_AVG	92646.0	0.044621	0.076036	0.000(
ELEVATORS_AVG	143620.0	0.078942	0.134576	0.000(
ENTRANCES_AVG	152683.0	0.149725	0.100049	0.000(
FLOORSMAX_AVG	154491.0	0.226282	0.144641	0.000(
FLOORSMIN_AVG	98869.0	0.231894	0.161380	0.000(
LANDAREA_AVG	124921.0	0.066333	0.081184	0.000(
LIVINGAPARTMENTS_AVG	97312.0	0.100775	0.092576	0.000(

LIVINGAREA_AVG	153161.0	0.107399	0.110565	0.0000
NONLIVINGAPARTMENTS_AVG	93997.0	0.008809	0.047732	0.0000
NONLIVINGAREA_AVG	137829.0	0.028358	0.069523	0.0000
APARTMENTS_MODE	151450.0	0.114231	0.107936	0.0000
BASEMENTAREA_MODE	127568.0	0.087543	0.084307	0.0000
YEARS_BEGINEXPLUATATION_MODE	157504.0	0.977065	0.064575	0.0000
YEARS_BUILD_MODE	103023.0	0.759637	0.110111	0.0000
COMMONAREA_MODE	92646.0	0.042553	0.074445	0.0000
ELEVATORS_MODE	143620.0	0.074490	0.132256	0.0000
ENTRANCES_MODE	152683.0	0.145193	0.100977	0.0000
FLOORSMAX_MODE	154491.0	0.222315	0.143709	0.0000
FLOORSMIN_MODE	98869.0	0.228058	0.161160	0.0000
LANDAREA_MODE	124921.0	0.064958	0.081750	0.0000
LIVINGAPARTMENTS_MODE	97312.0	0.105645	0.097880	0.0000
LIVINGAREA_MODE	153161.0	0.105975	0.111845	0.0000
NONLIVINGAPARTMENTS_MODE	93997.0	0.008076	0.046276	0.0000
NONLIVINGAREA_MODE	137829.0	0.027022	0.070254	0.0000
APARTMENTS_MEDI	151450.0	0.117850	0.109076	0.0000
BASEMENTAREA_MEDI	127568.0	0.087955	0.082179	0.0000
YEARS_BEGINEXPLUATATION_MEDI	157504.0	0.977752	0.059897	0.0000
YEARS_BUILD_MEDI	103023.0	0.755746	0.112066	0.0000
COMMONAREA_MEDI	92646.0	0.044595	0.076144	0.0000
ELEVATORS_MEDI	143620.0	0.078078	0.134467	0.0000
ENTRANCES_MEDI	152683.0	0.149213	0.100368	0.0000
FLOORSMAX_MEDI	154491.0	0.225897	0.145067	0.0000
FLOORSMIN_MEDI	98869.0	0.231625	0.161934	0.0000
LANDAREA_MEDI	124921.0	0.067169	0.082167	0.0000
LIVINGAPARTMENTS_MEDI	97312.0	0.101954	0.093642	0.0000
LIVINGAREA_MEDI	153161.0	0.108607	0.112260	0.0000
NONLIVINGAPARTMENTS_MEDI	93997.0	0.008651	0.047415	0.0000
NONLIVINGAREA_MEDI	137829.0	0.028236	0.070166	0.0000

TOTALAREA_MODE	159080.0	0.102547	0.107462	0.0000
OBS_30_CNT_SOCIAL_CIRCLE	306490.0	1.422245	2.400989	0.0000
DEF_30_CNT_SOCIAL_CIRCLE	306490.0	0.143421	0.446698	0.0000
OBS_60_CNT_SOCIAL_CIRCLE	306490.0	1.405292	2.379803	0.0000
DEF_60_CNT_SOCIAL_CIRCLE	306490.0	0.100049	0.362291	0.0000
DAYS_LAST_PHONE_CHANGE	307510.0	-962.858788	826.808487	-4.292
FLAG_DOCUMENT_2	307511.0	0.000042	0.006502	0.0000
FLAG_DOCUMENT_3	307511.0	0.710023	0.453752	0.0000
FLAG_DOCUMENT_4	307511.0	0.000081	0.009016	0.0000
FLAG_DOCUMENT_5	307511.0	0.015115	0.122010	0.0000
FLAG_DOCUMENT_6	307511.0	0.088055	0.283376	0.0000
FLAG_DOCUMENT_7	307511.0	0.000192	0.013850	0.0000
FLAG_DOCUMENT_8	307511.0	0.081376	0.273412	0.0000
FLAG_DOCUMENT_9	307511.0	0.003896	0.062295	0.0000
FLAG_DOCUMENT_10	307511.0	0.000023	0.004771	0.0000
FLAG_DOCUMENT_11	307511.0	0.003912	0.062424	0.0000
FLAG_DOCUMENT_12	307511.0	0.000007	0.002550	0.0000
FLAG_DOCUMENT_13	307511.0	0.003525	0.059268	0.0000
FLAG_DOCUMENT_14	307511.0	0.002936	0.054110	0.0000
FLAG_DOCUMENT_15	307511.0	0.001210	0.034760	0.0000
FLAG_DOCUMENT_16	307511.0	0.009928	0.099144	0.0000
FLAG_DOCUMENT_17	307511.0	0.000267	0.016327	0.0000
FLAG_DOCUMENT_18	307511.0	0.008130	0.089798	0.0000
FLAG_DOCUMENT_19	307511.0	0.000595	0.024387	0.0000
FLAG_DOCUMENT_20	307511.0	0.000507	0.022518	0.0000
FLAG_DOCUMENT_21	307511.0	0.000335	0.018299	0.0000
AMT_REQ_CREDIT_BUREAU_HOUR	265992.0	0.006402	0.083849	0.0000
AMT_REQ_CREDIT_BUREAU_DAY	265992.0	0.007000	0.110757	0.0000
AMT_REQ_CREDIT_BUREAU_WEEK	265992.0	0.034362	0.204685	0.0000
AMT_REQ_CREDIT_BUREAU_MON	265992.0	0.267395	0.916002	0.0000
AMT_REQ_CREDIT_BUREAU_QRT	265992.0	0.265474	0.794056	0.0000

AMT_REQ_CREDIT_BUREAU_YEAR	265992.0	1.899974	1.869295	0.000(

106 rows × 8 columns

In [9]:
```
# Count number of delinquent repayers ('TARGET' value of 1) and non-de
linquent repayers
# ('TARGET' value of 0) in the training set of the main data table.

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

Out[9]:
```
0    282686
1     24825
Name: TARGET, Length: 2, dtype: int64
```

In [10]:
```
# Fraction of applicants in training set who were delinquent repayers.
If you took a random
# sample, this is the probability you would select a delinquent repaye
r by chance:
repayers_fraction = round(24825/(24825+282686), 4)
print('Fraction of training set who were delinquent payers: {}'.format
(repayers_fraction))
```

```
Fraction of training set who were delinquent payers: 0.0807
```

Main Data Table Features with 'NaN' Entries

Do any features have mostly 'NaN' for their entries -- are any features too sparse to be of use?

In [24]:
```
# Get numerical counts of number of NaN entries in each column (featur
e) in the main data table.
features_sorted_by_NaN_count = application_train_data.isnull().sum().s
ort_values(ascending=False)

# Display only features with NaN counts greater than 0.
feature_NaN_counts = features_sorted_by_NaN_count[features_sorted_by_N
aN_count > 0]

# Create a dataframe to summarize 'NaN' entries of features in the mai
n data table (application_train_data)
feature_NaN_summary = pd.DataFrame(index=feature_NaN_counts.index, col
umns=['#_NaN_Entries','Fraction_of_entries_that_are_NaN','#_NaN_entrie
s_who_are_Delinquent','Fraction_of_NaN_entries_who_are_Delinquent','#n
on_NaN_Entries','#non_NaN_entries_who_are_Delinquent','Fraction_of_non
_NaN_entries_who_are_Delinquent'])

# Fill each row in the NaN summary dataframe
for feature_name in NaN_summary.index:
    # Get the amount and fraction of delinquents among all borrowers w
ho
```

```
    # have an 'NaN' entry for a particular feature. Do this for each f
eature that has at least
    # one 'NaN' entry.
    number_of_NaN = feature_NaN_counts.loc[feature_name]
    feature_NaN_summary['#_NaN_Entries'][feature_name] = number_of_NaN
    number_delinquents_who_are_NaN = application_train_data[(applicati
on_train_data[feature_name].isnull()) & (application_train_data['TARGE
T'] == 1)].shape[0]
    feature_NaN_summary['#_NaN_entries_who_are_Delinquent'][feature_na
me] = number_delinquents_who_are_NaN
    fraction_of_NaN_entries_who_are_delinquents = round(number_delinqu
ents_who_are_NaN/number_of_NaN,4)
    feature_NaN_summary['Fraction_of_NaN_entries_who_are_Delinquent'][
feature_name] = fraction_of_NaN_entries_who_are_delinquents

    # Get the amount of non-'NaN' entries in each feature that
    # has at least one 'NaN' entry.
    number_of_records = application_train_data[feature_name].shape[0]
    number_of_non_NaN = number_of_records - number_of_NaN
    feature_NaN_summary['#non_NaN_Entries'][feature_name] = number_of_
non_NaN

    # Get the fraction of the total entries for a feature that are 'Na
N'
    fraction_of_feature_entries_that_are_NaN = round(number_of_NaN/(nu
mber_of_NaN+number_of_non_NaN),4)
    feature_NaN_summary['Fraction_of_entries_that_are_NaN'][feature_na
me] = fraction_of_feature_entries_that_are_NaN

    # Get the amount and fraction of delinquents among all borrowers w
ho
    # have a non-'NaN' entry for a particular feature. Do this for eac
h feature that has at least
    # one 'NaN' entry.
    number_delinquents_who_are_not_NaN = application_train_data[(appli
cation_train_data[feature_name].notnull()) & (application_train_data['
TARGET'] == 1)].shape[0]
    feature_NaN_summary['#non_NaN_entries_who_are_Delinquent'][feature
_name] = number_delinquents_who_are_not_NaN
    fraction_of_non_NaN_entries_who_are_delinquents = round(number_del
inquents_who_are_not_NaN/number_of_non_NaN,4)
    feature_NaN_summary['Fraction_of_non_NaN_entries_who_are_Delinquen
t'][feature_name] = fraction_of_NaN_entries_who_are_delinquents

# Display the NaN summary dataframe below
display(feature_NaN_summary, 'display.max_columns')
```

	#_NaN_Entries	Fraction_of_entries_that_are_Na
COMMONAREA_MEDI	214865	0.6987
COMMONAREA_AVG	214865	0.6987

COMMONAREA_MODE	214865	0.6987
NONLIVINGAPARTMENTS_MODE	213514	0.6943
NONLIVINGAPARTMENTS_MEDI	213514	0.6943
NONLIVINGAPARTMENTS_AVG	213514	0.6943
FONDKAPREMONT_MODE	210295	0.6839
LIVINGAPARTMENTS_MEDI	210199	0.6835
LIVINGAPARTMENTS_MODE	210199	0.6835
LIVINGAPARTMENTS_AVG	210199	0.6835
FLOORSMIN_MEDI	208642	0.6785
FLOORSMIN_MODE	208642	0.6785
FLOORSMIN_AVG	208642	0.6785
YEARS_BUILD_MEDI	204488	0.665
YEARS_BUILD_AVG	204488	0.665
YEARS_BUILD_MODE	204488	0.665
OWN_CAR_AGE	202929	0.6599
LANDAREA_MODE	182590	0.5938
LANDAREA_AVG	182590	0.5938
LANDAREA_MEDI	182590	0.5938
BASEMENTAREA_MEDI	179943	0.5852
BASEMENTAREA_AVG	179943	0.5852
BASEMENTAREA_MODE	179943	0.5852
EXT_SOURCE_1	173378	0.5638
NONLIVINGAREA_MEDI	169682	0.5518
NONLIVINGAREA_AVG	169682	0.5518
NONLIVINGAREA_MODE	169682	0.5518
ELEVATORS_MODE	163891	0.533
ELEVATORS_AVG	163891	0.533
ELEVATORS_MEDI	163891	0.533
WALLSMATERIAL_MODE	156341	0.5084
APARTMENTS_MODE	156061	0.5075
APARTMENTS_AVG	156061	0.5075

APARTMENTS_MEDI	156061	0.5075
ENTRANCES_MEDI	154828	0.5035
ENTRANCES_MODE	154828	0.5035
ENTRANCES_AVG	154828	0.5035
LIVINGAREA_MEDI	154350	0.5019
LIVINGAREA_MODE	154350	0.5019
LIVINGAREA_AVG	154350	0.5019
HOUSETYPE_MODE	154297	0.5018
FLOORSMAX_MODE	153020	0.4976
FLOORSMAX_MEDI	153020	0.4976
FLOORSMAX_AVG	153020	0.4976
YEARS_BEGINEXPLUATATION_MEDI	150007	0.4878
YEARS_BEGINEXPLUATATION_AVG	150007	0.4878
YEARS_BEGINEXPLUATATION_MODE	150007	0.4878
TOTALAREA_MODE	148431	0.4827
EMERGENCYSTATE_MODE	145755	0.474
OCCUPATION_TYPE	96391	0.3135
EXT_SOURCE_3	60965	0.1983
AMT_REQ_CREDIT_BUREAU_QRT	41519	0.135
AMT_REQ_CREDIT_BUREAU_YEAR	41519	0.135
AMT_REQ_CREDIT_BUREAU_WEEK	41519	0.135
AMT_REQ_CREDIT_BUREAU_MON	41519	0.135
AMT_REQ_CREDIT_BUREAU_DAY	41519	0.135
AMT_REQ_CREDIT_BUREAU_HOUR	41519	0.135
NAME_TYPE_SUITE	1292	0.0042
OBS_30_CNT_SOCIAL_CIRCLE	1021	0.0033
OBS_60_CNT_SOCIAL_CIRCLE	1021	0.0033
DEF_60_CNT_SOCIAL_CIRCLE	1021	0.0033
DEF_30_CNT_SOCIAL_CIRCLE	1021	0.0033
EXT_SOURCE_2	660	0.0021
AMT_GOODS_PRICE	278	0.0009

AMT_ANNUITY	12	0
CNT_FAM_MEMBERS	2	0
DAYS_LAST_PHONE_CHANGE	1	0

67 rows × 7 columns

`'display.max_columns'`

The goal of creating the above summary of all 'NaN' entries in the main data table was to investigate three specific questions concerning these 67 features that each have at least one 'NaN' entry:

1. Whether simply having 'NaN' for a particular feature could help predict whether a borrower was delinquent.
2. Whether certain features with numerous 'NaN' entries would even be useful in predicting whether a borrower was delinquent. I am suspicious that a few of these features, such as `COMMONAREA_MEDI`, may have too few non-'NaN' entries that belong to delinquent payers. If too small a portion of the training set's target segment is included in these features, it's hard to see how they will be useful in making predictions that generalize to unseen datapoints.
3. Do 'NaN' values primarily belong to numerical features, categorical features, or both?

In the case of question `1.`, I can confirm that for all 67 features, a borrower simply having an 'NaN' entry for a feature does not in any way shape or form predict whether the borrower will be more or less likely to be delinquent. I verified this for each feature by looking at the fractions of both its 'NaN' and non-'NaN' cohorts that were delinquents. For each feature, I found that the fraction of 'NaN' borrowers who were delinquent was identical to the fraction of non-'NaN' borrowers who were delinquent. If being 'NaN' for a particular feature were to have any chance of being a meaningful predictor of delinquency, I would have expected these two proportions to have had a stastically significant difference for that feature.

For question `2.`, it's helpful to remember that, as confirmed above, there are 24,825 borrowers in the training dataset who were delinquent repayers (who had a `TARGET` value of 1). Any feature that I retain in spite of its 'NaN' entries needs to still have a large enough amount of non-'NaN' entries that belong to delinquents. If the entire training set's delinquent population is not adequately represented among a particular feature's valid data points, it's unlikely that any predictions meaningfully informed by this feature would generalize well to unseen datapoints. The effective training set size for this feature would be just too small, and my model would be at risk of underfitting.

However, the big question is: what should the cutoff line be? What fraction of the 24,825 delinquent borrowers in the training set need to be captured by a feature's valid data points in order for the feature to have a chance of being useful in making predictions? There is no general rule of thumb that I can use to answer this for each of these features that has 'NaN' values. The short answer is, it depends -- on many factors such as distribution of the feature's valid data, as well as the type of classifier algorithm I'm using.

This is why for the time being I won't remove *any* of these features -- even features like `COMMONAREA_MEDI` that contain 'NaN' in over two-thirds of their entries. Instead of running detailed statistical analyses on these features' distributions, I will instead experiment with dimensionality reduction algorithms such as PCA and/or feature selection algorithms such as SelectKBest.

If a feature had had, say, 95% of its entries as 'NaN', I probably would have removed it from the dataset at this point. However, since the sparsest features in the main data table still have valid data in just over 30% of their entries, I don't want to prematurely remove a feature that may have a chance, however remote, of possibly contributing to useful predictions.

Finally, to answer question `3.`, I found that all but one of the 46 normalized numerical features contain 'NaN' values. About half of the 21 non-normalized numerical features contain 'NaN' entries. Only 3 out of the 15 categorical features that will need to be one-hot encoded contain 'NaN' entries. None of the already one-hot encoded categorical features contain 'NaN' entries.

Main Data Table Borrowers with 'NaN' Entries

Do any borrowers have mostly 'NaN' entries for their feature data -- could any datapoints representing borrowers be classified as outliers and removed because their feature data is too sparse?

In [43]:
```
# Get numerical counts of number of NaN entries in each row (borrower)
in the main data table.
borrowers_sorted_by_NaN_count = application_train_data.isnull().sum(ax
is=1).sort_values(ascending=False)

# Display the number of 'NaN' entries for each borrower (left column
# is borrower ID, right column is number of 'NaN' entries).
display(borrowers_sorted_by_NaN_count)
```

```
133770    61
244833    61
150206    61
69707     61
269786    61
269492    61
116937    61
185713    61
197736    61
87399     60
7077      60
115346    60
201306    60
146561    60
68013     60
53941     60
48839     60
36899     60
54457     60
241006    60
290495    60
127942    60
260125    60
109581    60
```

266184	60
172595	60
271341	60
258474	60
12087	60
193598	59
177876	59
249616	59
219410	59
74199	59
247473	59
18244	59
261487	59
148096	59
80085	59
53550	59
168835	59
198111	59
3498	59
236260	59
28769	59
286217	59
63325	59
213503	59
39841	59
45043	59
109165	59
215458	59
154667	59
62176	59
299853	59
303509	59
193060	58
50126	58
131669	58
277217	58
113159	58
254163	58
214296	58
49241	58
261513	58
256958	58
143851	58
224619	58
226725	58
26398	58
226384	58
116520	58
3718	58
277969	58
43073	58
3525	58
7730	58

254388	58
193578	58
187145	58
187120	58
41982	58
267335	58
114010	57
80680	57
178880	57
75436	57
204013	57
282990	57
46822	57
75397	57
112641	57
290491	57
107081	57
270250	57
259914	57
251174	57
223106	57
58989	57
44953	57
212556	57
193059	57
107773	57
170830	57
145534	57
267537	57
27137	57
125828	57
47391	57
193117	57
22753	57
304867	57
95501	57
155988	57
108760	57
245893	57
7287	57
104716	57
2036	57
108763	57
245532	57
63171	57
193875	57
241925	57
63037	57
222063	57
63053	57
295068	57
42186	57
164737	57

251189	57
238617	57
78542	57
24378	57
63039	57
162648	57
193283	57
303110	57
201301	57
63032	57
42179	57
102350	57
269931	57
212129	57
120684	57
222937	57
82420	57
280019	57
222933	57
193246	57
156094	57
108746	57
47333	57
234418	57
11886	57
47350	57
295043	57
278331	57
203405	57
267554	57
238628	57
6958	57
135812	57
262887	57
37825	57
47320	57
203957	57
120707	57
125859	57
75975	57
95395	57
56169	57
135800	57
178861	57
178848	57
234452	57
238621	57
142982	57
117745	57
234373	57
89954	57
19546	57
198215	57

198207	57
127559	57
14449	57
179087	57
95689	57
18568	57
238704	57
155804	57
160076	57
282912	57
217582	57
179055	57
164971	57
192869	57
37501	57
37513	57
140843	57
234305	57
112551	57
271189	57
298610	57
298609	57
298605	57
234284	57
95702	57
204031	57
304975	57
256507	57
204163	57
179160	57
128618	57
276469	57
296727	57
1333	57
192772	57
78638	57
18417	57
10430	57
175075	57
120853	57
125727	57
271354	57
44919	57
217563	57
301108	57
117720	57
155777	57
108819	57
127561	57
287816	57
290569	57
125729	57
21664	57

192901	57
15892	57
162580	57
234340	57
53885	57
223341	57
127514	57
198175	57
252525	57
80668	57
107758	57
223330	57
256563	57
214826	57
	..
77356	0
241577	0
241614	0
77545	0
181695	0
181681	0
121620	0
241723	0
181674	0
14060	0
121635	0
77419	0
241799	0
20702	0
121643	0
121653	0
241139	0
181910	0
244113	0
240530	0
7546	0
121084	0
78411	0
25420	0
25421	0
121095	0
240512	0
121104	0
78233	0
182310	0
240595	0
182309	0
240598	0
78273	0
182308	0
121133	0
240451	0
121079	0

240439	0
240436	0
182464	0
78592	0
78576	0
121032	0
182449	0
182434	0
240281	0
78511	0
240330	0
121054	0
78497	0
240398	0
78472	0
240417	0
240423	0
25433	0
240679	0
181912	0
121301	0
182211	0
240948	0
240952	0
78011	0
20736	0
77991	0
182026	0
181969	0
78224	0
20733	0
25466	0
181952	0
77906	0
241065	0
241096	0
241100	0
182228	0
78074	0
78083	0
240866	0
240682	0
78189	0
7517	0
240731	0
78168	0
240757	0
78165	0
240771	0
240796	0
78139	0
182245	0
121224	0

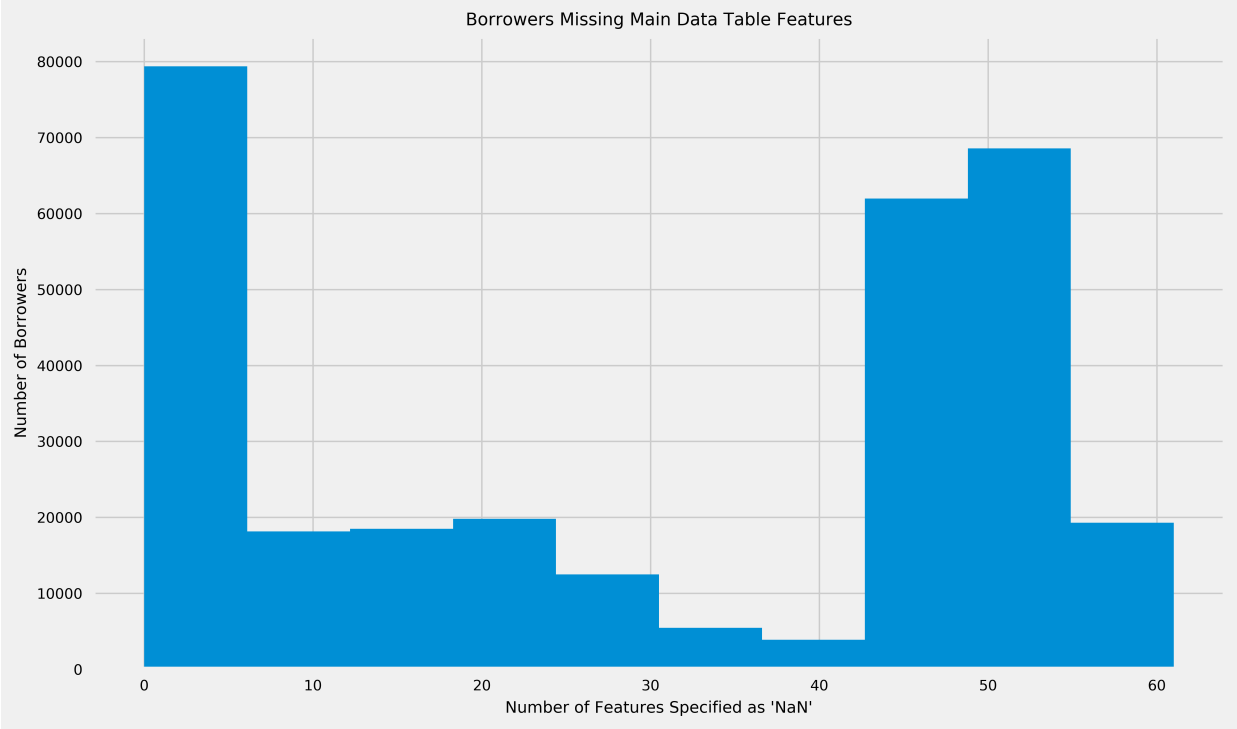
121245	0
240848	0
7501	0
121712	0
121714	0
242031	0
243339	0
243242	0
180917	0
122106	0
243273	0
76182	0
25657	0
180896	0
180867	0
180815	0
76132	0
76122	0
122161	0
243503	0
243513	0
243528	0
243539	0
180927	0
243199	0
180940	0
243192	0
242967	0
76318	0
243020	0
243023	0
180993	0
243050	0
122073	0
20633	0
243103	0
14155	0
243125	0
76263	0
76242	0
243161	0
122091	0
243550	0
122250	0
77138	0
180572	0
243916	0
180609	0
75869	0
243964	0
75864	0
122329	0
122356	0

243999	0
243623	0
75803	0
180521	0
180498	0
75785	0
244068	0
20592	0
75736	0
75893	0
75912	0
180657	0
243885	0
243642	0
180790	0
122273	0
76026	0
243722	0
75976	0
180756	0
122300	0
243789	0
180748	0
75948	0
75934	0
243849	0
180682	0
243884	0
242965	0
181001	0
76341	0
181311	0
242355	0
242362	0
242371	0
242396	0
121861	0
76835	0
121874	0
242452	0
76356	0
181308	0
121892	0
76775	0
242482	0
76759	0
181290	0
242496	0
121847	0
76882	0
242318	0
76889	0
242066	0

77095	0
181445	0
77027	0
181429	0
181428	0
76985	0
20679	0
242161	0
242199	0
121800	0
121806	0
242233	0
181385	0
242294	0
242515	0
14116	0
121921	0
181113	0
242775	0
242785	0
122007	0
181088	0
76459	0
242838	0
181057	0
242874	0
242877	0
7261	0
76401	0
242905	0
20639	0
122033	0
122038	0
181112	0
76498	0
76728	0
76501	0
20654	0
76717	0
121946	0
181210	0
242560	0
181194	0
121951	0
181145	0
76621	0
242659	0
242676	0
76611	0
76606	0
181134	0
76597	0
153755	0

```
Length: 307511, dtype: int64
```

In [53]:
```
# Plot a histogram showing the number of borrowers that have a
# certain amount of their feature data as 'NaN' entries.
plt.figure(figsize = (10,6))
plt.hist(borrowers_sorted_by_NaN_count)
plt.title('Borrowers Missing Main Data Table Features')
plt.xlabel('Number of Features Specified as \'NaN\'')
plt.ylabel('Number of Borrowers')
plt.savefig('borrowersnandata.png')
plt.show()
```



The above plot confirms that no borrowers are missing too many features such that they'd be considered outliers and would need to be removed from the trainingset.

At most, a borrower may be missing roughly only half of the main data table's 120 features. Just under 20,000 of the training dataset's 307,511 borrower records face this "worst-case" scenario. As it is, having 61 missing features falls well below the threshold at which I would decide to remove a borrower from the training dataset for having feature data that is too sparse. A borrower would have to be missing over 100 features, at least 5/6 of the featureset, for me to take the time to explore more deeply whether they may be an outlier.

Main Data Table Numerical and Categorical Feature Investigation: Unexpected Values

I explored samples from all features, as well as statistical descriptions of each numerical feature in the main data table (count, mean, standard deviation, minimum value, 25th percentile, 50th percentile, 75th

data table (count, mean, standard deviation, minimum value, 25th percentile, 50th percentile, 75th percentile, and maximum value) to ensure that no features contained unexpected values that fall outside the range one would expect based on the feature's definition.

Examples of unexpected values include entries that are impossibly small/large, or values that are negative when only positive values would be expected.

I came across the following five anomalies:

1. The following five numerical features indicate the number of days prior to the loan application's submission that a particular event took place:

   - `DAYS_BIRTH`
   - `DAYS_EMPLOYED`
   - `DAYS_REGISTRATION`
   - `DAYS_ID_PUBLISH`
   - `DAYS_LAST_PHONE_CHANGE`

   For example, `DAYS_LAST_PHONE_CHANGE` is defined by Home Credit as: "*How many days before application did client change phone?*"

   Values for the above five features are negative, which is expected since each value represents a point in time prior to the time of the loan application's submission, which Home Credit defines as time 0, which is the maximum value for a few of these features.

   What's unexpected is that the `DAYS_EMPLOYED` feature (the number of days the applicant has had a job prior to the day they submitted their loan application) has a maximum value that is both a *positive* number as well as unbelievably large. The maximum value for `DAYS_EMPLOYED` is 365,243 days, or just over 1,000 years.

   No human being lives for 1,000 years, let alone sustains a job for that long, so this entry clearly indicates that some sort of mistake was made. What's not yet clear to me is whether `DAYS_EMPLOYED` contains only one, a handful, or possibly several such points. Clearly this particular data point and any similar to it are outliers that should be removed, especially if the `DAYS_EMPLOYED` feature turns out to be otherwise useful for predicting target values. Indeed, based on my intuition, `DAYS_EMPLOYED` is one of the first features that I would guess would be relevant in predicting whether a borrower would eventually make a late loan payment.

   Creating and inspecting a histogram of this feature's data should help me to guage whether or not there are any other outlier data points that would need to be removed.
2. I was undecided whether the features `OWN_CAR_AGE`, the age of the applicant's car, `HOUR_APPR_PROCESS_START`, the hour the loan application was submitted, `CNT_CHILDREN`, the number of children that the applicant has, and `CNT_FAM_MEMBERS`, the size of the applicant's family, should be thought of as categorical or numerical.

   The first reason for this uncertainty was because the entries in each feature were rounded to whole numbers. The second reason was that the range of whole number entries for each feature was quite limited -- [0,23] for `HOUR_APPR_PROCESS_START`, [0.0,91.0] for `OWN_CAR_AGE`, [0.0,19.0] for `CNT_CHILDREN`, and [1.0,20.0] for `CNT_FAM_MEMBERS`. Although the range of values in

OWN_CAR_AGE is nearly four times that of HOUR_APPR_PROCESS_START, upon exploring the individual entries, I found that the majority of values appeared to be in the range [0.0,20.0]. This makes sense, considering that most cars don't last longer than twenty years. The efective range of HOUR_APPR_PROCESS_START values appeared to be even more narrow, with most entries concentrated inside the range [9,17]. This also makes sense, as regular business hours typically run from 9AM to 5PM.

I ultimately decided that even though the effective ranges of both OWN_CAR_AGE and HOUR_APPR_PROCESS_START, are far more narrow than other numerical features in the main data table, the nature of each feature's data requires that I treat both as numerical features.

The other categorical features in the main data table, such as whether the applicant owns a car, or the applicant's housing type, each have distinct entries that can encapsulate wildly different meanings and implications. The condition of owning a car is very different from the condition of not owning a car, and the lifestyle, financial and otherwise, of someone living in an apartment is likely very different from that of someone living in a stand-alone house.

After thinking along these lines, it was easy for me to see that neither the entries in OWN_CAR_AGE nor those HOUR_APPR_PROCESS_START are necessarily always that different from one another in meaning or implication. For example, is submitting a loan application at 3PM really that different from submitting at 4PM? What about having a car that's eight years old versus having a car that's nine years old? For data like this, it is far more likely that there are meaningful sub-ranges, such as the afternoon hours of 1PM to 5PM, that may be helpful in predicting a borrower's target value. The only way I will be able to discover these sub-ranges is if I treat these features as numerical, not categorical.

Things are different for the CNT_CHILDREN and CNT_FAM_MEMBERS features. Although the most children any loan applicant had was 19, at least 75% of all applicants had either zero children or just one child. I decided that it makes most sense to treat CNT_CHILDREN as a categorical feature, and re-engineer it to segment the borrower population into the following two categories: having no children, and having one or more children. Although I expect there will groups of borrowers of diminishing size that have 2,3,4,...,19 children, my hypothesis is that having no children versus having at least one child will be the information most useful to predict target values for the overall population of borrowers. Even if I were to spend time investigating the effects of having 2 vs. 3 vs. 4 vs. ... vs. 19 children, I know that my findings would apply to less than 25% of the overall population and any predictions informed by these effects likely wouldn't generalize well to unseen datapoints.

I will transform the CNT_CHILDREN feature into a binary categorical feature called HAS_CHILDREN. If the value of CNT_CHILDREN is greater than 0, the value of HAS_CHILDREN will be 1. If the value of CNT_CHILDREN is 0, the value of HAS_CHILDREN will be 0.

For CNT_FAM_MEMBERS, the situation is somewhat similar. 25% of borrowers in the training set have a family size of just one, 50% have a family of two or less, and 75% of borrowers have families of 3 people or less. I plan to re-engineer this feature to categorically segment borrowers into the following three groups: having a family size of one, a family size of two, and having a family that's three people or larger.

I will transform the CNT_FAM_MEMBERS feature into a categorical feature called NUMBER_FAMILY_MEMBERS. If CNT_FAM_MEMBERS is 1.0, then the value of NUMBER_FAMILY_MEMBERS will be 'one'. If CNT_FAM_MEMBERS is 2.0, then NUMBER_FAMILY_MEMBERS will be 'two'. If CNT_FAM_MEMBERS is 3.0 or greater, then NUMBER_FAMILY_MEMBERS will be 'three_plus'. The new categorical feature NUMBER_FAMILY_MEMBERS will eventually be one-hot encoded.

3. While most features reported as being normalized have a max value of 1.0 and a min value of 0.0. The following three features all have values within the range (0.0, 1.0), none of them have max values as 1.0, nor min values of 0.0. Even though Home Credit states that each of these three features have been normalized, because the max and min values of their range supposedly normalized values are different than what I've observed for all other features reported as normalized, I will need to pay special attention to the graphs of the distributions of these three features when I conduct my exploratory data visualization, in order to verify that these features' data is indeed distributed normally:

    - EXT_SOURCE_1
    - EXT_SOURCE_2
    - EXT_SOURCE_3
4. The feature REGION_POPULATION_RELATIVE appears to be a unique case in that while it has also supposedly been normalized, its values merely fall into the range [0.000290, 0.072508]. All other features that Home Credit claims have been normalized more or less fall into the range [0.0,1.0]. It's therefore a given that I'll need to min-max scale this feature during data preprocessing.
5. The following four features were defined by Home Credit as being "normalized":

    - FONDKAPREMONT_MODE
    - HOUSETYPE_MODE
    - WALLSMATERIAL_MODE
    - EMERGENCYSTATE_MODE

    Specifically, the definition contained in the HomeCredit_columns_description.csv (https://www.kaggle.com/c/9120/download/HomeCredit_columns_description.csv) file provided by Home Credit stated that these features were: "*Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor*"

    However, upon further investigation I found that each of these four features was in fact categorical, and would need to be one-hot encoded. WALLSMATERIAL_MODE, for example, contains entries such as "Stone, brick", "panel", and "block" that indicate the material(s) of which the walls in the borrower's house are built.

Main Data Table Numerical and Categorical Feature Summary

I divided the main data table's features into groups based both on whether the feature is categorical or numerical, as well as how the feature would need to be preprocessed. ie. has the feature already been normalized, has it already been one-hot encoded, etc. Features that contain at least one 'NaN' value are highlighted in red.

Categorical features needing one-hot encoding:

1. NAME_CONTRACT_TYPE
2. CODE_GENDER
3. FLAG_OWN_CAR
4. FLAG_OWN_REALTY
5. NAME_INCOME_TYPE
6. NAME_EDUCATION_TYPE
7. NAME_FAMILY_STATUS
8. NAME_HOUSING_TYPE
9. REGION_RATING_CLIENT
10. REGION_RATING_CLIENT_W_CITY
11. WEEKDAY_APPR_PROCESS_START
12. ORGANIZATION_TYPE
13. NAME_TYPE_SUITE (1,292 'NaN' entries)
14. OCCUPATION_TYPE (96,391 'NaN' entries)

Categorical features originally mis-identified by Home Credit as normalized (they also need to be one-hot encoded):

1. EMERGENCYSTATE_MODE (145,755 'NaN' entries)
2. HOUSETYPE_MODE (154,297 'NaN' entries)
3. WALLSMATERIAL_MODE (156,341 'NaN' entries)
4. FONDKAPREMONT_MODE (210,295 'NaN' entries)

Categorical features needing re-engineering:

1. CNT_CHILDREN: Will be transformed into a binary categorical feature called `HAS_CHILDREN`, with value of 1 if borrower has 0 children vs. a value of 1 if borrower has at least one child.
2. CNT_FAM_MEMBERS (2 'NaN' entries): Will be transformed into categorical feature called `NUMBER_FAMILY_MEMBERS`, with values of 'one', 'two', or 'three_plus' depending on whether the borrower's value for `CNT_FAM_MEMBERS` was 1, 2, or 3 or more. `NUMBER_FAMILY_MEMBERS` will eventually be one-hot encoded.

Binary Categorical features already one-hot encoded:

1. FLAG_MOBIL
2. FLAG_EMP_PHONE
3. FLAG_WORK_PHONE
4. FLAG_CONT_MOBILE
5. FLAG_PHONE
6. FLAG_EMAIL
7. REG_REGION_NOT_LIVE_REGION
8. REG_REGION_NOT_WORK_REGION
9. LIVE_REGION_NOT_WORK_REGION
10. REG_CITY_NOT_LIVE_CITY
11. REG_CITY_NOT_WORK_CITY
12. LIVE_CITY_NOT_WORK_CITY
13. FLAG_DOCUMENT_2
14. FLAG_DOCUMENT_3
15. FLAG_DOCUMENT_4
16. FLAG_DOCUMENT_5
17. FLAG_DOCUMENT_6
18. FLAG_DOCUMENT_7
19. FLAG_DOCUMENT_8
20. FLAG_DOCUMENT_9
21. FLAG_DOCUMENT_10
22. FLAG_DOCUMENT_11
23. FLAG_DOCUMENT_12
24. FLAG_DOCUMENT_13
25. FLAG_DOCUMENT_14
26. FLAG_DOCUMENT_15
27. FLAG_DOCUMENT_16
28. FLAG_DOCUMENT_17
29. FLAG_DOCUMENT_18
30. FLAG_DOCUMENT_19
31. FLAG_DOCUMENT_20
32. FLAG_DOCUMENT_21

Numerical features not identified as normalized, that have roughly normal distributions:

1. DAYS_BIRTH
2. DAYS_ID_PUBLISH
3. HOUR_APPR_PROCESS_START

Numerical features not identified as normalized, that have skewed distributions:

1. AMT_INCOME_TOTAL
2. AMT_CREDIT
3. AMT_ANNUITY (12 'NaN' entries)
4. AMT_GOODS_PRICE (278 'NaN' entries)
5. OBS_30_CNT_SOCIAL_CIRCLE (1,021 'NaN' entries)
6. DEF_30_CNT_SOCIAL_CIRCLE (1,021 'NaN' entries)
7. OBS_60_CNT_SOCIAL_CIRCLE (1,021 'NaN' entries)
8. DEF_60_CNT_SOCIAL_CIRCLE (1,021 'NaN' entries)
9. AMT_REQ_CREDIT_BUREAU_HOUR (41,519 'NaN' entries)
10. AMT_REQ_CREDIT_BUREAU_DAY (41,519 'NaN' entries)
11. AMT_REQ_CREDIT_BUREAU_WEEK (41,519 'NaN' entries)
12. AMT_REQ_CREDIT_BUREAU_MON (41,519 'NaN' entries)
13. AMT_REQ_CREDIT_BUREAU_QRT (41,519 'NaN' entries)
14. AMT_REQ_CREDIT_BUREAU_YEAR (41,519 'NaN' entries)
15. OWN_CAR_AGE (202,929 'NaN' entries)

Numerical features not identified as normalized, that have skewed distributions and negative values:

1. DAYS_EMPLOYED: Will be transformed into a binary categorical feature called `HAS_JOB`, with value of 1 if borrower has a value of 0 or less for `DAYS_EMPLOYED`. `HAS_JOB` will have a value of 0 if the borrower is one of the 55,374 folks who has a value of 365243 for `DAYS_EMPLOYED`.
2. DAYS_REGISTRATION
3. DAYS_LAST_PHONE_CHANGE (1 'NaN' entry)

Numerical features identified as normalized, which are scaled to range [0,1]:

1. EXT_SOURCE_2 (660 'NaN' entries)
2. EXT_SOURCE_3 (60,965 'NaN' entries)
3. TOTALAREA_MODE (148,431 'NaN' entries)
4. YEARS_BEGINEXPLUATATION_AVG (150,007 'NaN' entries)
5. YEARS_BEGINEXPLUATATION_MODE (150,007 'NaN' entries)
6. YEARS_BEGINEXPLUATATION_MEDI (150,007 'NaN' entries)
7. FLOORSMAX_AVG (153,020 'NaN' entries)
8. FLOORSMAX_MODE (153,020 'NaN' entries)
9. FLOORSMAX_MEDI (153,020 'NaN' entries)
10. LIVINGAREA_AVG (154,350 'NaN' entries)
11. LIVINGAREA_MODE (154,350 'NaN' entries)
12. LIVINGAREA_MEDI (154,350 'NaN' entries)
13. ENTRANCES_AVG (154,828 'NaN' entries)
14. ENTRANCES_MODE (154,828 'NaN' entries)
15. ENTRANCES_MEDI (154,828 'NaN' entries)

16. APARTMENTS_AVG (156,061 'NaN' entries)
17. APARTMENTS_MODE (156,061 'NaN' entries)
18. APARTMENTS_MEDI (156,061 'NaN' entries)
19. ELEVATORS_AVG (163,891 'NaN' entries)
20. ELEVATORS_MODE (163,891 'NaN' entries)
21. ELEVATORS_MEDI (163,891 'NaN' entries)
22. NONLIVINGAREA_AVG (169,682 'NaN' entries)
23. NONLIVINGAREA_MODE (169,682 'NaN' entries)
24. NONLIVINGAREA_MEDI (169,682 'NaN' entries)
25. EXT_SOURCE_1 (173,378 'NaN' entries)
26. BASEMENTAREA_AVG (179,943 'NaN' entries)
27. BASEMENTAREA_MODE (179,943 'NaN' entries)
28. BASEMENTAREA_MEDI (179,943 'NaN' entries)
29. LANDAREA_AVG (182,590 'NaN' entries)
30. LANDAREA_MODE (182,590 'NaN' entries)
31. LANDAREA_MEDI (182,590 'NaN' entries)
32. YEARS_BUILD_AVG (204,488 'NaN' entries)
33. YEARS_BUILD_MODE (204,488 'NaN' entries)
34. YEARS_BUILD_MEDI (204,488 'NaN' entries)
35. FLOORSMIN_AVG (208,642 'NaN' entries)
36. FLOORSMIN_MODE (208,642 'NaN' entries)
37. FLOORSMIN_MEDI (208,642 'NaN' entries)
38. LIVINGAPARTMENTS_AVG (210,199 'NaN' entries)
39. LIVINGAPARTMENTS_MODE (210,199 'NaN' entries)
40. LIVINGAPARTMENTS_MEDI (210,199 'NaN' entries)
41. NONLIVINGAPARTMENTS_AVG (213,514 'NaN' entries)
42. NONLIVINGAPARTMENTS_MODE (213,514 'NaN' entries)
43. NONLIVINGAPARTMENTS_MEDI (213,514 'NaN' entries)
44. COMMONAREA_AVG (214,865 'NaN' entries)
45. COMMONAREA_MODE (214,865 'NaN' entries)
46. COMMONAREA_MEDI (214,865 'NaN' entries)

Numerical features identified as normalized, which are *not* scaled to range [0,1]:

1. REGION_POPULATION_RELATIVE

Summing the above lists gives 14+4+2+32+3+15+3+46+1 = 120 features, which are all of the features in the main data table.

Main Data Table Featureset Definitions

1. **SK_ID_CURR**: ID of loan in our sample
2. **TARGET**: Target variable (1 - client with payment difficulties: he/she had late payment more than X

days on at least one of the first Y installments of the loan in our sample, 0 - all other cases)

3. **NAME_CONTRACT_TYPE**: Identification if loan is cash or revolving
4. **CODE_GENDER**: Gender of the client
5. **FLAG_OWN_CAR**: Flag if the client owns a car
6. **FLAG_OWN_REALTY**: Flag if client owns a house or flat
7. **CNT_CHILDREN**: Number of children the client has
8. **AMT_INCOME_TOTAL**: Income of the client
9. **AMT_CREDIT**: Credit amount of the loan
10. **AMT_ANNUITY**: Loan annuity
11. **AMT_GOODS_PRICE**: For consumer loans it is the price of the goods for which the loan is given
12. **NAME_TYPE_SUITE**: Who was accompanying client when he was applying for the loan
13. **NAME_INCOME_TYPE**: Clients income type (businessman, working, maternity leave,Ö)
14. **NAME_EDUCATION_TYPE**: Level of highest education the client achieved
15. **NAME_FAMILY_STATUS**: Family status of the client
16. **NAME_HOUSING_TYPE**: What is the housing situation of the client (renting, living with parents, ...)
17. **REGION_POPULATION_RELATIVE**: Normalized population of region where client lives (higher number means the client lives in more populated region) -- normalized
18. **DAYS_BIRTH**: Client's age in days at the time of application -- time only relative to the application
19. **DAYS_EMPLOYED**: How many days before the application the person started current employment -- time only relative to the application
20. **DAYS_REGISTRATION**: How many days before the application did client change his registration -- time only relative to the application
21. **DAYS_ID_PUBLISH**: How many days before the application did client change the identity document with which he applied for the loan -- time only relative to the application
22. **OWN_CAR_AGE**: Age of client's car
23. **FLAG_MOBIL**: Did client provide mobile phone (1=YES, 0=NO)
24. **FLAG_EMP_PHONE**: Did client provide work phone (1=YES, 0=NO)
25. **FLAG_WORK_PHONE**: Did client provide home phone (1=YES, 0=NO)
26. **FLAG_CONT_MOBILE**: Was mobile phone reachable (1=YES, 0=NO)
27. **FLAG_PHONE**: Did client provide home phone (1=YES, 0=NO)
28. **FLAG_EMAIL**: Did client provide email (1=YES, 0=NO)
29. **OCCUPATION_TYPE**: What kind of occupation does the client have
30. **CNT_FAM_MEMBERS**: How many family members does client have
31. **REGION_RATING_CLIENT**: Our rating of the region where client lives (1,2,3)
32. **REGION_RATING_CLIENT_W_CITY**: Our rating of the region where client lives with taking city into account (1,2,3)
33. **WEEKDAY_APPR_PROCESS_START**: On which day of the week did the client apply for the loan
34. **HOUR_APPR_PROCESS_START**: Approximately at what hour did the client apply for the loan rounded
35. **REG_REGION_NOT_LIVE_REGION**: Flag if client's permanent address does not match contact address (1=different, 0=same, at region level)
36. **REG_REGION_NOT_WORK_REGION**: Flag if client's permanent address does not match work address (1=different, 0=same, at region level)
37. **LIVE_REGION_NOT_WORK_REGION**: Flag if client's contact address does not match work address (1=different, 0=same, at region level)

38. **REG_CITY_NOT_LIVE_CITY**: Flag if client's permanent address does not match contact address (1=different, 0=same, at city level)
39. **REG_CITY_NOT_WORK_CITY**: Flag if client's permanent address does not match work address (1=different, 0=same, at city level)
40. **LIVE_CITY_NOT_WORK_CITY**: Flag if client's contact address does not match work address (1=different, 0=same, at city level)
41. **ORGANIZATION_TYPE**: Type of organization where client works
42. **EXT_SOURCE_1**: Normalized score from external data source -- normalized
43. **EXT_SOURCE_2**: Normalized score from external data source -- normalized
44. **EXT_SOURCE_3**: Normalized score from external data source -- normalized
45. **APARTMENTS_AVG**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
46. **BASEMENTAREA_AVG**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
47. **YEARS_BEGINEXPLUATATION_AVG**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
48. **YEARS_BUILD_AVG**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
49. **COMMONAREA_AVG**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
50. **ELEVATORS_AVG**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
51. **ENTRANCES_AVG**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
52. **FLOORSMAX_AVG**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
53. **FLOORSMIN_AVG**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of

floor -- normalized

54. **LANDAREA_AVG**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
55. **LIVINGAPARTMENTS_AVG**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
56. **LIVINGAREA_AVG**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
57. **NONLIVINGAPARTMENTS_AVG**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
58. **NONLIVINGAREA_AVG**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
59. **APARTMENTS_MODE**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
60. **BASEMENTAREA_MODE**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
61. **YEARS_BEGINEXPLUATATION_MODE**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
62. **YEARS_BUILD_MODE**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
63. **COMMONAREA_MODE**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
64. **ELEVATORS_MODE**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized

65. **ENTRANCES_MODE**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
66. **FLOORSMAX_MODE**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
67. **FLOORSMIN_MODE**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
68. **LANDAREA_MODE**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
69. **LIVINGAPARTMENTS_MODE**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
70. **LIVINGAREA_MODE**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
71. **NONLIVINGAPARTMENTS_MODE**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
72. **NONLIVINGAREA_MODE**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
73. **APARTMENTS_MEDI**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
74. **BASEMENTAREA_MEDI**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
75. **YEARS_BEGINEXPLUATATION_MEDI**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
76. **YEARS_BUILD_MEDI**: Normalized information about building where the client lives, What is

average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized

77. **COMMONAREA_MEDI**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
78. **ELEVATORS_MEDI**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
79. **ENTRANCES_MEDI**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
80. **FLOORSMAX_MEDI**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
81. **FLOORSMIN_MEDI**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
82. **LANDAREA_MEDI**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
83. **LIVINGAPARTMENTS_MEDI**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
84. **LIVINGAREA_MEDI**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
85. **NONLIVINGAPARTMENTS_MEDI**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
86. **NONLIVINGAREA_MEDI**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
87. **FONDKAPREMONT_MODE**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area,

living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized

88. **HOUSETYPE_MODE**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
89. **TOTALAREA_MODE**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
90. **WALLSMATERIAL_MODE**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
91. **EMERGENCYSTATE_MODE**: Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor -- normalized
92. **OBS_30_CNT_SOCIAL_CIRCLE**: How many observation of client's social surroundings with observable 30 DPD (days past due) default
93. **DEF_30_CNT_SOCIAL_CIRCLE**: How many observation of client's social surroundings defaulted on 30 DPD (days past due)
94. **OBS_60_CNT_SOCIAL_CIRCLE**: How many observation of client's social surroundings with observable 60 DPD (days past due) default
95. **DEF_60_CNT_SOCIAL_CIRCLE**: How many observation of client's social surroundings defaulted on 60 (days past due) DPD
96. **DAYS_LAST_PHONE_CHANGE**: How many days before application did client change phone
97. **FLAG_DOCUMENT_2**: Did client provide document 2
98. **FLAG_DOCUMENT_3**: Did client provide document 3
99. **FLAG_DOCUMENT_4**: Did client provide document 4
100. **FLAG_DOCUMENT_5**: Did client provide document 5
101. **FLAG_DOCUMENT_6**: Did client provide document 6
102. **FLAG_DOCUMENT_7**: Did client provide document 7
103. **FLAG_DOCUMENT_8**: Did client provide document 8
104. **FLAG_DOCUMENT_9**: Did client provide document 9
105. **FLAG_DOCUMENT_10**: Did client provide document 10
106. **FLAG_DOCUMENT_11**: Did client provide document 11
107. **FLAG_DOCUMENT_12**: Did client provide document 12
108. **FLAG_DOCUMENT_13**: Did client provide document 13
109. **FLAG_DOCUMENT_14**: Did client provide document 14
110. **FLAG_DOCUMENT_15**: Did client provide document 15
111. **FLAG_DOCUMENT_16**: Did client provide document 16
112. **FLAG_DOCUMENT_17**: Did client provide document 17
113. **FLAG_DOCUMENT_18**: Did client provide document 18
114. **FLAG_DOCUMENT_19**: Did client provide document 19

115. **FLAG_DOCUMENT_20**: Did client provide document 20
116. **FLAG_DOCUMENT_21**: Did client provide document 21
117. **AMT_REQ_CREDIT_BUREAU_HOUR**: Number of enquiries to Credit Bureau about the client one hour before application
118. **AMT_REQ_CREDIT_BUREAU_DAY**: Number of enquiries to Credit Bureau about the client one day before application (excluding one hour before application)
119. **AMT_REQ_CREDIT_BUREAU_WEEK**: Number of enquiries to Credit Bureau about the client one week before application (excluding one day before application)
120. **AMT_REQ_CREDIT_BUREAU_MON**: Number of enquiries to Credit Bureau about the client one month before application (excluding one week before application)
121. **AMT_REQ_CREDIT_BUREAU_QRT**: Number of enquiries to Credit Bureau about the client 3 month before application (excluding one month before application)
122. **AMT_REQ_CREDIT_BUREAU_YEAR**: Number of enquiries to Credit Bureau about the client one day year (excluding last 3 months before application)

2. Bureau Data Table (bureau.csv)

In [188]:
```
# Display the first five records
display(bureau_data.head(n=5))
```

	SK_ID_CURR	SK_ID_BUREAU	CREDIT_ACTIVE	CREDIT_CURRENCY	DAYS_CRED
0	215354	5714462	Closed	currency 1	-497
1	215354	5714463	Active	currency 1	-208
2	215354	5714464	Active	currency 1	-203
3	215354	5714465	Active	currency 1	-203
4	215354	5714466	Active	currency 1	-629

5 rows × 17 columns

In [190]:
```
# Display the above data sample table, but with axes transposed, so th
at the table can
# be included in the project's writeup.
display(bureau_data.head(n=5).transpose())
```

	0	1	2	3	4
SK_ID_CURR	215354	215354	215354	215354	215354
SK_ID_BUREAU	5714462	5714463	5714464	5714465	5714466
CREDIT_ACTIVE	Closed	Active	Active	Active	Active
CREDIT_CURRENCY	currency 1	currency 1	currency 1	currency 1	currency 1
DAYS_CREDIT	-497	-208	-203	-203	-629
CREDIT_DAY_OVERDUE	0	0	0	0	0
DAYS_CREDIT_ENDDATE	-153	1075	528	NaN	1197
DAYS_ENDDATE_FACT	-153	NaN	NaN	NaN	NaN
AMT_CREDIT_MAX_OVERDUE	NaN	NaN	NaN	NaN	77674.5
CNT_CREDIT_PROLONG	0	0	0	0	0
AMT_CREDIT_SUM	91323	225000	464324	90000	2.7e+06
AMT_CREDIT_SUM_DEBT	0	171342	NaN	NaN	NaN
AMT_CREDIT_SUM_LIMIT	NaN	NaN	NaN	NaN	NaN
AMT_CREDIT_SUM_OVERDUE	0	0	0	0	0
CREDIT_TYPE	Consumer credit	Credit card	Consumer credit	Credit card	Consumer credit
DAYS_CREDIT_UPDATE	-131	-20	-16	-16	-21
AMT_ANNUITY	NaN	NaN	NaN	NaN	NaN

17 rows × 5 columns

In [198]:
```
# Display a statistical description of the numerical features in the b
ureau data table.
display(bureau_data.describe())
```

	SK_ID_CURR	SK_ID_BUREAU	DAYS_CREDIT	CREDIT_DAY_OVERDUE	DAYS_
count	1.716428e+06	1.716428e+06	1.716428e+06	1.716428e+06	1.6108
mean	2.782149e+05	5.924434e+06	-1.142108e+03	8.181666e-01	5.1051
std	1.029386e+05	5.322657e+05	7.951649e+02	3.654443e+01	4.9942
min	1.000010e+05	5.000000e+06	-2.922000e+03	0.000000e+00	-4.206
25%	1.888668e+05	5.463954e+06	-1.666000e+03	0.000000e+00	-1.138
50%	2.780550e+05	5.926304e+06	-9.870000e+02	0.000000e+00	-3.300
75%	3.674260e+05	6.385681e+06	-4.740000e+02	0.000000e+00	4.7400
max	4.562550e+05	6.843457e+06	0.000000e+00	2.792000e+03	3.1199

8 rows × 14 columns

In [199]:
```
# Display the above statistical description table, but with axes inver
ted, so that the table can
# be included in the project's writeup.
display(bureau_data.describe().transpose())
```

	count	mean	std	min
SK_ID_CURR	1716428.0	2.782149e+05	1.029386e+05	100001.000
SK_ID_BUREAU	1716428.0	5.924434e+06	5.322657e+05	5000000.000
DAYS_CREDIT	1716428.0	-1.142108e+03	7.951649e+02	-2922.000
CREDIT_DAY_OVERDUE	1716428.0	8.181666e-01	3.654443e+01	0.000
DAYS_CREDIT_ENDDATE	1610875.0	5.105174e+02	4.994220e+03	-42060.000
DAYS_ENDDATE_FACT	1082775.0	-1.017437e+03	7.140106e+02	-42023.000
AMT_CREDIT_MAX_OVERDUE	591940.0	3.825418e+03	2.060316e+05	0.000
CNT_CREDIT_PROLONG	1716428.0	6.410406e-03	9.622391e-02	0.000
AMT_CREDIT_SUM	1716415.0	3.549946e+05	1.149811e+06	0.000
AMT_CREDIT_SUM_DEBT	1458759.0	1.370851e+05	6.774011e+05	-4705600.320
AMT_CREDIT_SUM_LIMIT	1124648.0	6.229515e+03	4.503203e+04	-586406.115
AMT_CREDIT_SUM_OVERDUE	1716428.0	3.791276e+01	5.937650e+03	0.000
DAYS_CREDIT_UPDATE	1716428.0	-5.937483e+02	7.207473e+02	-41947.000
AMT_ANNUITY	489637.0	1.571276e+04	3.258269e+05	0.000

14 rows × 8 columns

Bureau Data Table Featureset Definitions

1. **SK_ID_CURR**: ID of loan in our sample - one loan in our sample can have 0,1,2 or more related previous credits in credit bureau -- hashed
2. **SK_ID_BUREAU**: Recoded ID of previous Credit Bureau credit related to our loan (unique coding for each loan application) -- hashed
3. **CREDIT_ACTIVE**: Status of the Credit Bureau (CB) reported credits
4. **CREDIT_CURRENCY**: Recoded currency of the Credit Bureau credit -- recoded
5. **DAYS_CREDIT**: How many days before current application did client apply for Credit Bureau credit -- time only relative to the application
6. **CREDIT_DAY_OVERDUE**: Number of days past due on CB credit at the time of application for related loan in our sample
7. **DAYS_CREDIT_ENDDATE**: Remaining duration of CB credit (in days) at the time of application in Home Credit -- time only relative to the application
8. **DAYS_ENDDATE_FACT**: Days since CB credit ended at the time of application in Home Credit (only for closed credit) -- time only relative to the application
9. **AMT_CREDIT_MAX_OVERDUE**: Maximal amount overdue on the Credit Bureau credit so far (at application date of loan in our sample)
10. **CNT_CREDIT_PROLONG**: How many times was the Credit Bureau credit prolonged
11. **AMT_CREDIT_SUM**: Current credit amount for the Credit Bureau credit
12. **AMT_CREDIT_SUM_DEBT**: Current debt on Credit Bureau credit
13. **AMT_CREDIT_SUM_LIMIT**: Current credit limit of credit card reported in Credit Bureau
14. **AMT_CREDIT_SUM_OVERDUE**: Current amount overdue on Credit Bureau credit
15. **CREDIT_TYPE**: Type of Credit Bureau credit (Car, cash,...)
16. **DAYS_CREDIT_UPDATE**: How many days before loan application did last information about the Credit Bureau credit come -- time only relative to the application
17. **AMT_ANNUITY**: Annuity of the Credit Bureau credit

3. Bureau Balance Data Table (bureau_balance.csv)

In [24]:
```
# Display the first five records
display(bureau_balance_data.head(n=5))
```

	SK_ID_BUREAU	MONTHS_BALANCE	STATUS
0	5715448	0	C
1	5715448	-1	C
2	5715448	-2	C
3	5715448	-3	C
4	5715448	-4	C

Bureau Balance Data Table Featureset Definitions

1. **SK_ID_BUREAU**: Recoded ID of Credit Bureau credit (unique coding for each application) - use this to join to CREDIT_BUREAU table -- hashed
2. **MONTHS_BALANCE**: Month of balance relative to application date (-1 means the freshest balance date) -- time only relative to the application
3. **STATUS**: Status of Credit Bureau loan during the month (active, closed, DPD0-30,Ö [C means closed, X means status unknown, 0 means no DPD, 1 means maximal did during month between 1-30, 2 means DPD 31-60,Ö 5 means DPD 120+ or sold or written off])

4. Previous Application Data Table (previous_application.csv)

In [26]:
```
# Display the first five records
display(previous_application_data.head(n=5))
```

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLI
0	2030495	271877	Consumer loans	1730.430	17145.0
1	2802425	108129	Cash loans	25188.615	607500.0
2	2523466	122040	Cash loans	15060.735	112500.0
3	2819243	176158	Cash loans	47041.335	450000.0
4	1784265	202054	Cash loans	31924.395	337500.0

5 rows × 37 columns

Previous Application Data Table Featureset Definitions

1. **SK_ID_PREV**: ID of previous credit in Home credit related to loan in our sample. (One loan in our sample can have 0,1,2 or more previous loan applications in Home Credit, previous application could, but not necessarily have to lead to credit) -- hashed
2. **SK_ID_CURR**: ID of loan in our sample -- hashed
3. **NAME_CONTRACT_TYPE**: Contract product type (Cash loan, consumer loan [POS] ,...) of the previous application
4. **AMT_ANNUITY**: Annuity of previous application
5. **AMT_APPLICATION**: For how much credit did client ask on the previous application
6. **AMT_CREDIT**: Final credit amount on the previous application. This differs from AMT_APPLICATION in a way that the AMT_APPLICATION is the amount for which the client initially applied for, but during our approval process he could have received different amount - AMT_CREDIT
7. **AMT_DOWN_PAYMENT**: Down payment on the previous application
8. **AMT_GOODS_PRICE**: Goods price of good that client asked for (if applicable) on the previous

application

9. **WEEKDAY_APPR_PROCESS_START**: On which day of the week did the client apply for previous application
10. **HOUR_APPR_PROCESS_START**: Approximately at what day hour did the client apply for the previous application -- rounded
11. **FLAG_LAST_APPL_PER_CONTRACT**: Flag if it was last application for the previous contract. Sometimes by mistake of client or our clerk there could be more applications for one single contract
12. **NFLAG_LAST_APPL_IN_DAY**: Flag if the application was the last application per day of the client. Sometimes clients apply for more applications a day. Rarely it could also be error in our system that one application is in the database twice
13. **RATE_DOWN_PAYMENT**: Down payment rate normalized on previous credit -- normalized
14. **RATE_INTEREST_PRIMARY**: Interest rate normalized on previous credit -- normalized
15. **RATE_INTEREST_PRIVILEGED**: Interest rate normalized on previous credit -- normalized
16. **NAME_CASH_LOAN_PURPOSE**: Purpose of the cash loan
17. **NAME_CONTRACT_STATUS**: Contract status (approved, cancelled, ...) of previous application
18. **DAYS_DECISION**: Relative to current application when was the decision about previous application made time only relative to the application
19. **NAME_PAYMENT_TYPE**: Payment method that client chose to pay for the previous application
20. **CODE_REJECT_REASON**: Why was the previous application rejected
21. **NAME_TYPE_SUITE**: Who accompanied client when applying for the previous application
22. **NAME_CLIENT_TYPE**: Was the client old or new client when applying for the previous application
23. **NAME_GOODS_CATEGORY**: What kind of goods did the client apply for in the previous application
24. **NAME_PORTFOLIO**: Was the previous application for CASH, POS, CAR, Ö
25. **NAME_PRODUCT_TYPE**: Was the previous application x-sell o walk-in
26. **CHANNEL_TYPE**: Through which channel we acquired the client on the previous application
27. **SELLERPLACE_AREA**: Selling area of seller place of the previous application
28. **NAME_SELLER_INDUSTRY**: The industry of the seller
29. **CNT_PAYMENT**: Term of previous credit at application of the previous application
30. **NAME_YIELD_GROUP**: Grouped interest rate into small medium and high of the previous application -- grouped
31. **PRODUCT_COMBINATION**: Detailed product combination of the previous application
32. **DAYS_FIRST_DRAWING**: Relative to application date of current application when was the first disbursement of the previous application -- time only relative to the application
33. **DAYS_FIRST_DUE**: Relative to application date of current application when was the first due supposed to be of the previous application -- time only relative to the application
34. **DAYS_LAST_DUE_1ST_VERSION**: Relative to application date of current application when was the first due of the previous application -- time only relative to the application
35. **DAYS_LAST_DUE**: Relative to application date of current application when was the last due date of the previous application -- time only relative to the application
36. **DAYS_TERMINATION**: Relative to application date of current application when was the expected termination of the previous application -- time only relative to the application
37. **NFLAG_INSURED_ON_APPROVAL**: Did the client requested insurance during the previous application

5. POS CASH Balance Data Table (POS_CASH_balance.csv)

In [20]:
```
# Display the first five records
display(POS_CASH_balance_data.head(n=5))
```

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	CNT_INSTALMENT	CNT_INSTALM
0	1803195	182943	-31	48.0	45.0
1	1715348	367990	-33	36.0	35.0
2	1784872	397406	-32	12.0	9.0
3	1903291	269225	-35	48.0	42.0
4	2341044	334279	-35	36.0	35.0

POS CASH Balance Data Table Featureset Definitions

1. **SK_ID_PREV**: ID of previous credit in Home Credit related to loan in our sample. (One loan in our sample can have 0,1,2 or more previous loans in Home Credit)
2. **SK_ID_CURR**: ID of loan in our sample
3. **MONTHS_BALANCE**: Month of balance relative to application date (-1 means the information to the freshest monthly snapshot, 0 means the information at application - often it will be the same as -1 as many banks are not updating the information to Credit Bureau regularly) -- time only relative to the application
4. **CNT_INSTALMENT**: Term of previous credit (can change over time)
5. **CNT_INSTALMENT_FUTURE**: Installments left to pay on the previous credit
6. **NAME_CONTRACT_STATUS**: Contract status during the month
7. **SK_DPD**: DPD (days past due) during the month of previous credit
8. **SK_DPD_DEF**: DPD during the month with tolerance (debts with low loan amounts are ignored) of the previous credit

6. Installments Payments Data Table (installments_payments.csv)

In [21]:
```
# Display the first five records
display(installments_payments_data.head(n=5))
```

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUM
0	1054186	161674	1.0	6
1	1330831	151639	0.0	34
2	2085231	193053	2.0	1
3	2452527	199697	1.0	3
4	2714724	167756	1.0	2

Installments Payments Data Table Featureset Definitions

1. **SK_ID_PREV**: ID of previous credit in Home credit related to loan in our sample. (One loan in our sample can have 0,1,2 or more previous loans in Home Credit) -- hashed
2. **SK_ID_CURR**: ID of loan in our sample -- hashed
3. **NUM_INSTALMENT_VERSION**: Version of installment calendar (0 is for credit card) of previous credit. Change of installment version from month to month signifies that some parameter of payment calendar has changed
4. **NUM_INSTALMENT_NUMBER**: On which installment we observe payment
5. **DAYS_INSTALMENT**: When the installment of previous credit was supposed to be paid (relative to application date of current loan) -- time only relative to the application
6. **DAYS_ENTRY_PAYMENT**: When was the installments of previous credit paid actually (relative to application date of current loan) -- time only relative to the application
7. **AMT_INSTALMENT**: What was the prescribed installment amount of previous credit on this installment
8. **AMT_PAYMENT**: What the client actually paid on previous credit on this installment

7. Credit Card Balance Data Table (credit_card_balance.csv)

In [22]:
```
# Display the first five records
display(credit_card_balance_data.head(n=5))
```

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	AMT_CREDIT_LI
0	2562384	378907	-6	56.970	135000
1	2582071	363914	-1	63975.555	45000
2	1740877	371185	-7	31815.225	450000
3	1389973	337855	-4	236572.110	225000
4	1891521	126868	-1	453919.455	450000

5 rows × 23 columns

Credit Card Balance Data Table Featureset Definitions

1. **SK_ID_PREV**: ID of previous credit in Home credit related to loan in our sample. (One loan in our sample can have 0,1,2 or more previous loans in Home Credit) -- hashed
2. **SK_ID_CURR**: ID of loan in our sample -- hashed
3. **MONTHS_BALANCE**: Month of balance relative to application date (-1 means the freshest balance date) -- time only relative to the application
4. **AMT_BALANCE**: Balance during the month of previous credit
5. **AMT_CREDIT_LIMIT_ACTUAL**: Credit card limit during the month of the previous credit
6. **AMT_DRAWINGS_ATM_CURRENT**: Amount drawing at ATM during the month of the previous credit
7. **AMT_DRAWINGS_CURRENT**: Amount drawing during the month of the previous credit
8. **AMT_DRAWINGS_OTHER_CURRENT**: Amount of other drawings during the month of the previous credit
9. **AMT_DRAWINGS_POS_CURRENT**: Amount drawing or buying goods during the month of the previous credit
10. **AMT_INST_MIN_REGULARITY**: Minimal installment for this month of the previous credit
11. **AMT_PAYMENT_CURRENT**: How much did the client pay during the month on the previous credit
12. **AMT_PAYMENT_TOTAL_CURRENT**: How much did the client pay during the month in total on the previous credit
13. **AMT_RECEIVABLE_PRINCIPAL**: Amount receivable for principal on the previous credit
14. **AMT_RECIVABLE**: Amount receivable on the previous credit
15. **AMT_TOTAL_RECEIVABLE**: Total amount receivable on the previous credit
16. **CNT_DRAWINGS_ATM_CURRENT**: Number of drawings at ATM during this month on the previous credit
17. **CNT_DRAWINGS_CURRENT**: Number of drawings during this month on the previous credit
18. **CNT_DRAWINGS_OTHER_CURRENT**: Number of other drawings during this month on the previous credit
19. **CNT_DRAWINGS_POS_CURRENT**: Number of drawings for goods during this month on the previous credit
20. **CNT_INSTALMENT_MATURE_CUM**: Number of paid installments on the previous credit
21. **NAME_CONTRACT_STATUS**: Contract status (active signed,...) on the previous credit
22. **SK_DPD**: DPD (Days past due) during the month on the previous credit
23. **SK_DPD_DEF**: DPD (Days past due) during the month with tolerance (debts with low loan amounts are ignored) of the previous credit

Feature Engineering

What is one feature that could be engineered from the data contained in the six tables that are supplementary to the main data table?

Of the six other tables in the dataset outside of the main data table, four tables (`previous_application.csv`, `POS_CASH_balance.csv`, `installments_payments.csv`, and `credit_card_balance.csv`) contain information pertaining to previous loan applications, or payback histories on prior loans, that an applicant has had with Home Credit. To engineer a new feature, I instead intend to focus on the two data tables that describe applicants' payback performance with lenders *other than* Home Credit:

- `bureau.csv`
- `bureau_balance.csv`

My hypothesis is that out of the six supplementary data tables, the above two tables will be the greatest source of supplementary insight.

`bureau.csv` contains summary information of applicants' loans from other lenders, such as the amount and type of loan, and the total amount, if any, of the repayment balance that's overdue. `bureau_balance.csv` contains the month by month statuses (whether a particular month's balance payment was received and processed, or the extent to which payment is overdue) for each loan described in `bureau.csv`.

Although the month-by-month payment statuses in `bureau_balance.csv` may prove useful with the right kind of time series analysis, for the purposes of this project I will attempt to engineer a feature that is based on the data contained in the features in `bureau.csv`. In particular, I will focus on the features in `bureau.csv` that indicate whether an individual has had difficulty repaying previous loans, the extent of that difficulty, and how recently that difficulty has occurred. Some potentially useful features include:

- `DAYS_CREDIT`: Number of days since individual had applied for a loan.
- `DAYS_CREDIT_UPDATE`: Number of days since individual's information on the credit bureau was updated.
- `CREDIT_DAY_OVERDUE`: Number of days that individual's loan payments have been overdue.
- `AMT_CREDIT_MAX_OVERDUE`: Maximum amount individual has ever been overdue on their loan payments.
- `AMT_CREDIT_SUM_OVERDUE`: Amount individual is currently overdue on their loan payments.

Since I'm interested in knowing whether a Home Credit loan applicant has had *recent* difficulty paying back loans they've received from other creditors, I will build a feature based solely on the `CREDIT_DAY_OVERDUE` feature. For simplicity's sake, my engineered feature will merely indicate whether or not a Home Credit applicant currently has overdue loan payments from other creditors.

My new feature will be titled `HAS_CREDIT_BUREAU_LOANS_OVERDUE`. If a Home Credit applicant has at least one loan in `bureau.csv` for which `CREDIT_DAY_OVERDUE` has a value greater than 0, the value of my new feature in the row belonging to that applicant's Home Credit borrower ID will be 1. Otherwise, the value will be 0. I will engineer this feature during the data preprocessing phase, and once it has been created I will append it to the main data table.

II. Exploratory Visualization

1. Plot Distributions of Numerical Main Data Table Features Identified as Normalized

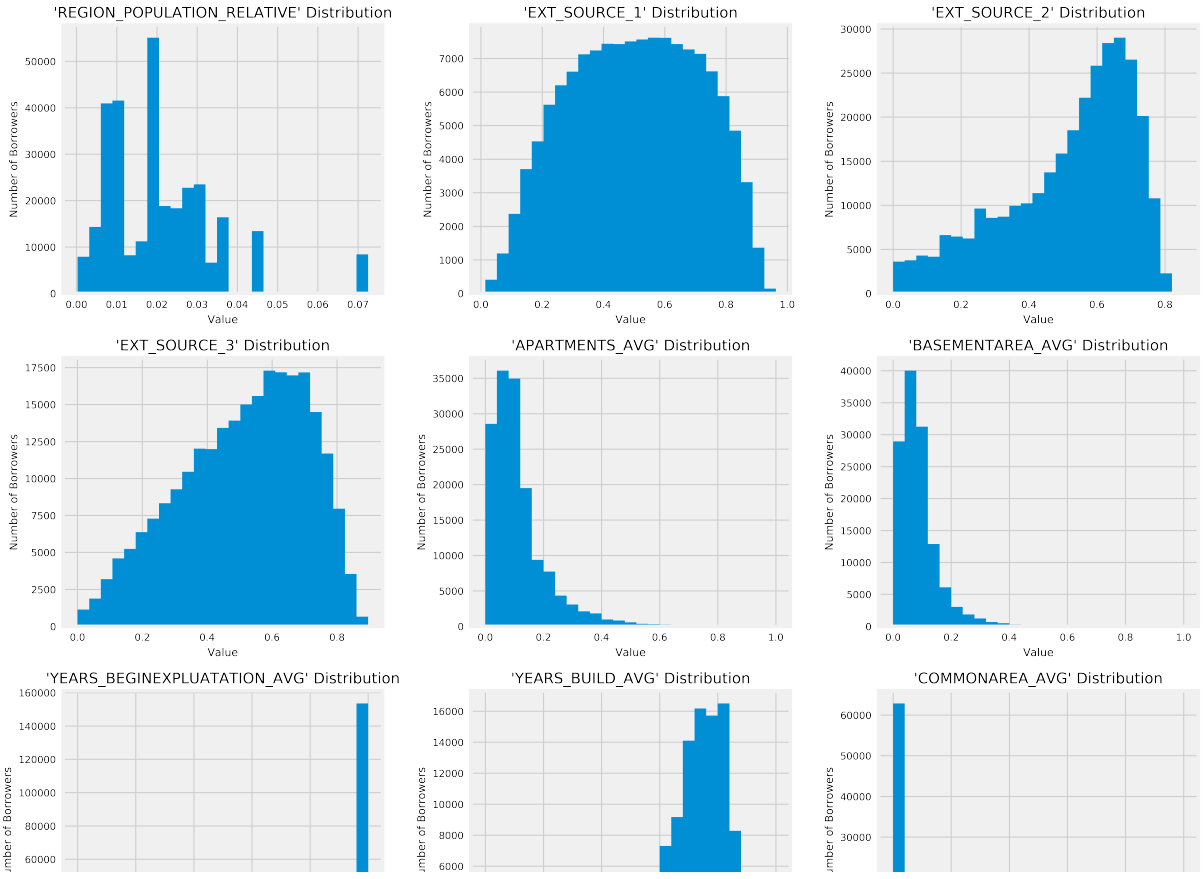
Home Credit had identified the following 47 numerical features as being normalized. I plotted histograms of each of these features in order to confirm that this is indeed that case, as well as to identify any outliers that might need to be removed.

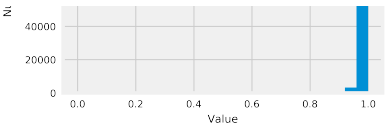
```
'REGION_POPULATION_RELATIVE', 'EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_
3', 'APARTMENTS_AVG', 'BASEMENTAREA_AVG', 'YEARS_BEGINEXPLUATATION_AVG', '
YEARS_BUILD_AVG', 'COMMONAREA_AVG', 'ELEVATORS_AVG', 'ENTRANCES_AVG', 'FLO
ORSMAX_AVG', 'FLOORSMIN_AVG', 'LANDAREA_AVG', 'LIVINGAPARTMENTS_AVG', 'LIV
INGAREA_AVG', 'NONLIVINGAPARTMENTS_AVG', 'NONLIVINGAREA_AVG', 'APARTMENTS_
MODE', 'BASEMENTAREA_MODE', 'YEARS_BEGINEXPLUATATION_MODE', 'YEARS_BUILD_M
ODE', 'COMMONAREA_MODE', 'ELEVATORS_MODE', 'ENTRANCES_MODE', 'FLOORSMAX_MO
DE', 'FLOORSMIN_MODE', 'LANDAREA_MODE', 'LIVINGAPARTMENTS_MODE', 'LIVINGAR
EA_MODE', 'NONLIVINGAPARTMENTS_MODE', 'NONLIVINGAREA_MODE', 'APARTMENTS_ME
DI', 'BASEMENTAREA_MEDI', 'YEARS_BEGINEXPLUATATION_MEDI', 'YEARS_BUILD_MED
I', 'COMMONAREA_MEDI', 'ELEVATORS_MEDI', 'ENTRANCES_MEDI', 'FLOORSMAX_MEDI
', 'FLOORSMIN_MEDI', 'LANDAREA_MEDI', 'LIVINGAPARTMENTS_MEDI', 'LIVINGAREA
_MEDI', 'NONLIVINGAPARTMENTS_MEDI', 'NONLIVINGAREA_MEDI', 'TOTALAREA_MODE'
```

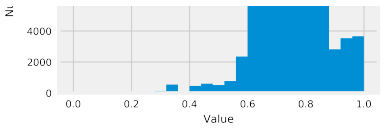
The plotted histograms ignored all 'NaN' entries in each of the above features. Plots for all 47 features can be viewed in the Appendix.

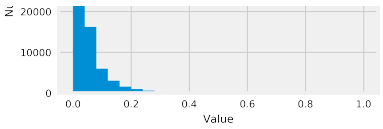
In [34]:
```
# List of normalized features
normalized_numerical_features = ['REGION_POPULATION_RELATIVE', 'EXT_SO
URCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3', 'APARTMENTS_AVG', 'BASEMENTAR
EA_AVG', 'YEARS_BEGINEXPLUATATION_AVG', 'YEARS_BUILD_AVG', 'COMMONAREA
_AVG', 'ELEVATORS_AVG', 'ENTRANCES_AVG', 'FLOORSMAX_AVG', 'FLOORSMIN_A
VG', 'LANDAREA_AVG', 'LIVINGAPARTMENTS_AVG', 'LIVINGAREA_AVG', 'NONLIV
INGAPARTMENTS_AVG', 'NONLIVINGAREA_AVG', 'APARTMENTS_MODE', 'BASEMENTA
REA_MODE', 'YEARS_BEGINEXPLUATATION_MODE', 'YEARS_BUILD_MODE', 'COMMON
AREA_MODE', 'ELEVATORS_MODE', 'ENTRANCES_MODE', 'FLOORSMAX_MODE', 'FLO
ORSMIN_MODE', 'LANDAREA_MODE', 'LIVINGAPARTMENTS_MODE', 'LIVINGAREA_MO
DE', 'NONLIVINGAPARTMENTS_MODE', 'NONLIVINGAREA_MODE', 'APARTMENTS_MED
I', 'BASEMENTAREA_MEDI', 'YEARS_BEGINEXPLUATATION_MEDI', 'YEARS_BUILD_
MEDI', 'COMMONAREA_MEDI', 'ELEVATORS_MEDI', 'ENTRANCES_MEDI', 'FLOORSM
AX_MEDI', 'FLOORSMIN_MEDI', 'LANDAREA_MEDI', 'LIVINGAPARTMENTS_MEDI',
'LIVINGAREA_MEDI', 'NONLIVINGAPARTMENTS_MEDI', 'NONLIVINGAREA_MEDI', '
TOTALAREA_MODE']

# Plot histogram of each normalized feature, omitting any rows (borrow
ers) that have a value
# of 'NaN' for the particular feature
vs.plot_feature_distributions(application_train_data[normalized_numeri
cal_features], title='Distributions of Main Data Table\'s Normalized F
eatures', figsize=(14,60), num_cols=3)
```

Distributions of Main Data Table's Normalized Features

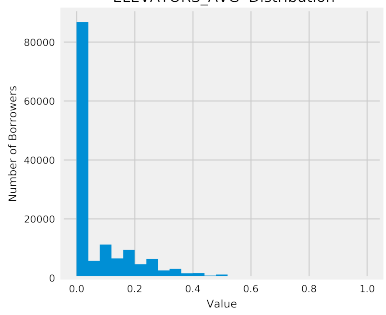




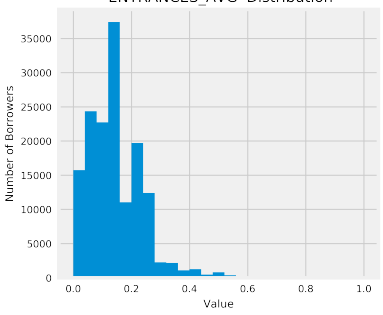




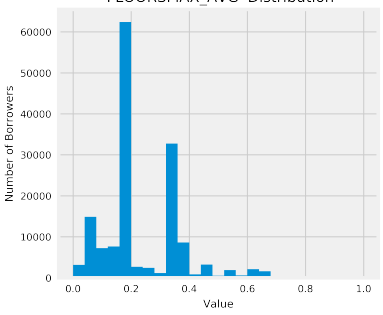
'ELEVATORS_AVG' Distribution



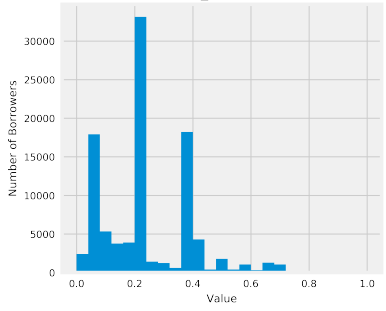
'ENTRANCES_AVG' Distribution



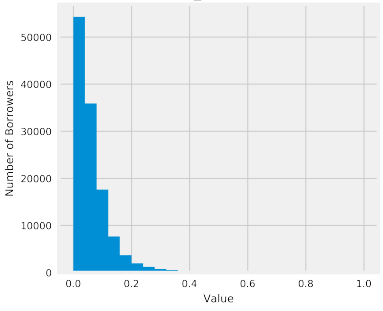
'FLOORSMAX_AVG' Distribution



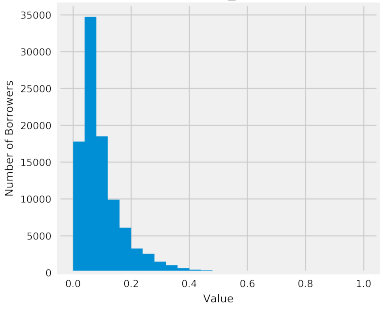
'FLOORSMIN_AVG' Distribution



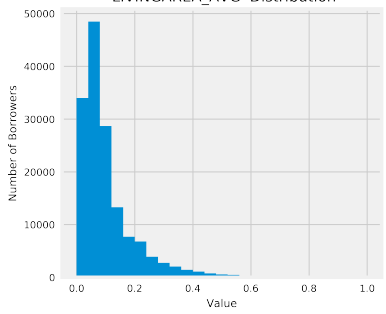
'LANDAREA_AVG' Distribution



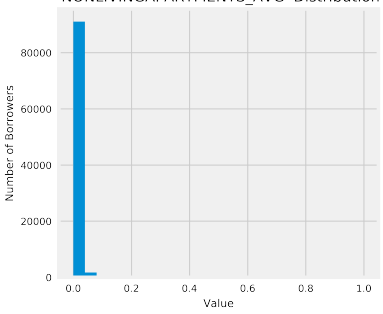
'LIVINGAPARTMENTS_AVG' Distribution



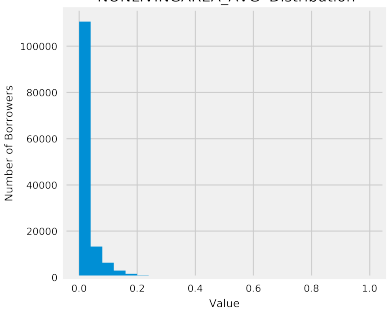
'LIVINGAREA_AVG' Distribution



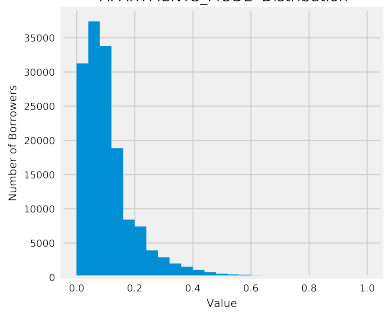
'NONLIVINGAPARTMENTS_AVG' Distribution



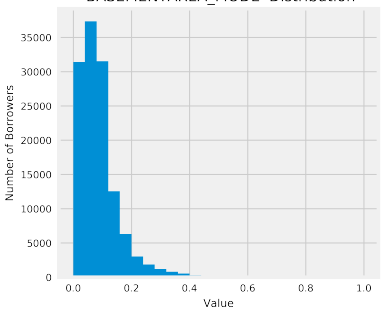
'NONLIVINGAREA_AVG' Distribution



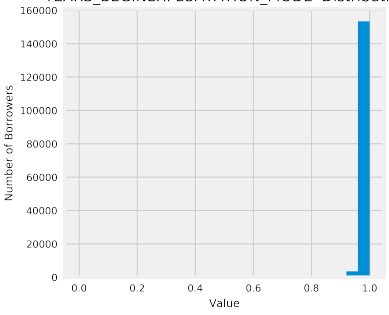
'APARTMENTS_MODE' Distribution



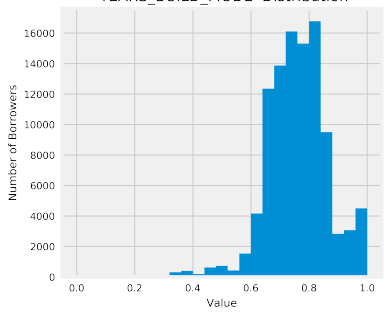
'BASEMENTAREA_MODE' Distribution



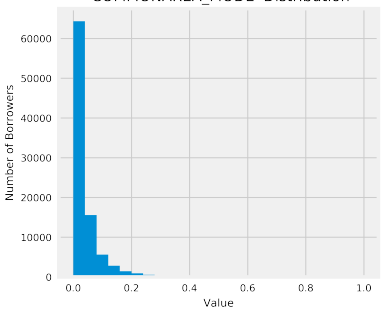
'YEARS_BEGINEXPLUATATION_MODE' Distribution



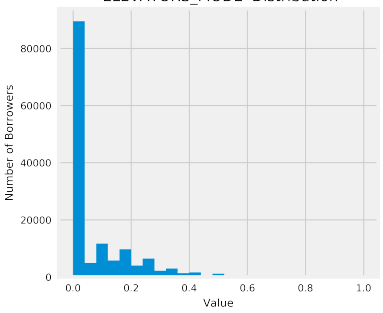
'YEARS_BUILD_MODE' Distribution



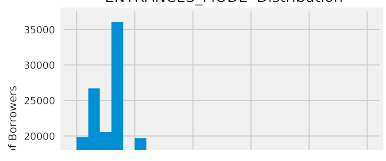
'COMMONAREA_MODE' Distribution



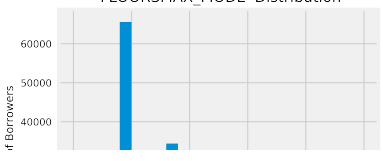
'ELEVATORS_MODE' Distribution



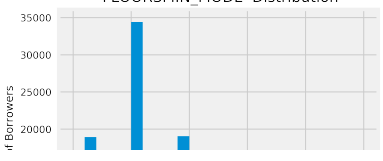
'ENTRANCES_MODE' Distribution

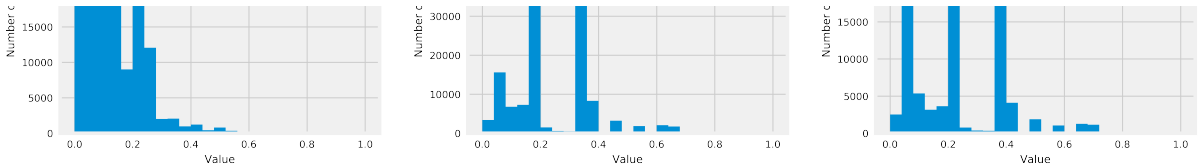


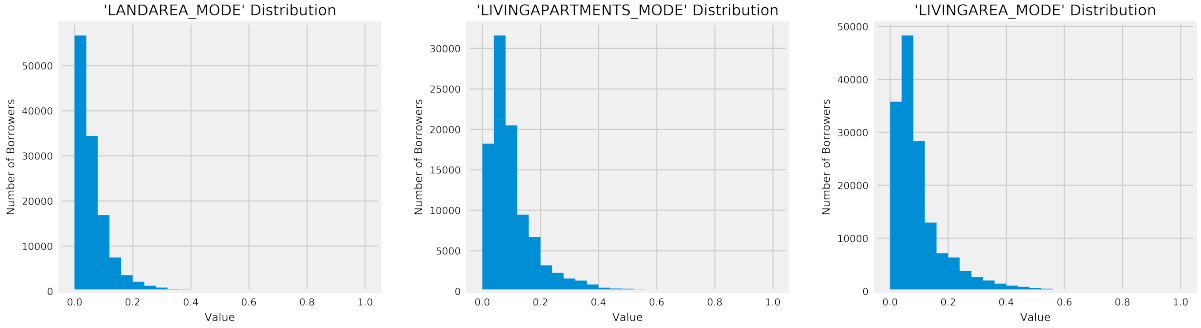
'FLOORSMAX_MODE' Distribution

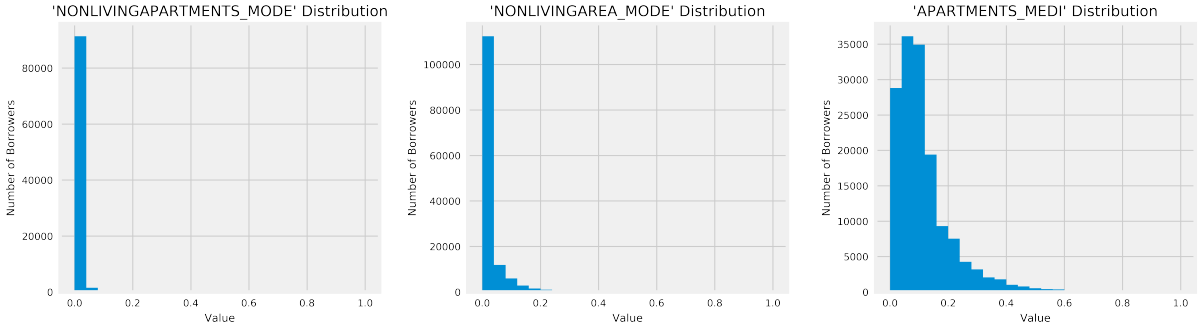


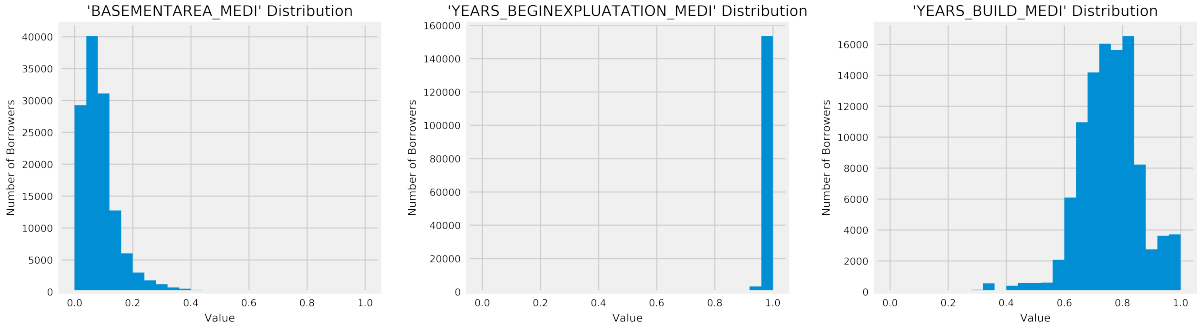
'FLOORSMIN_MODE' Distribution

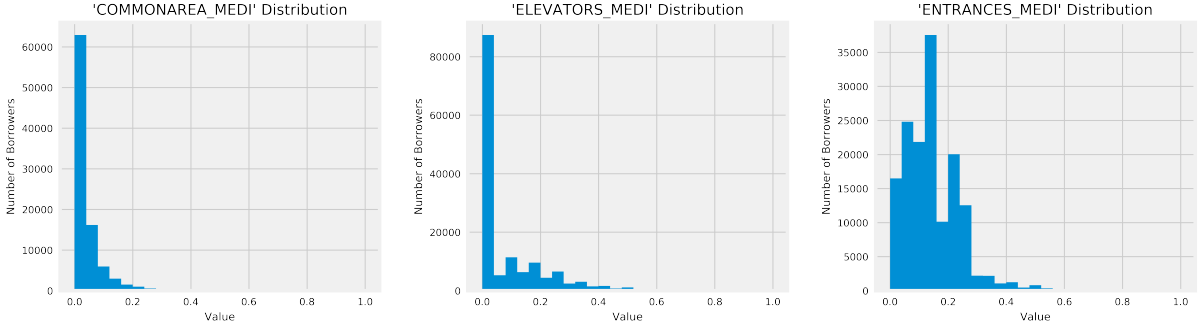


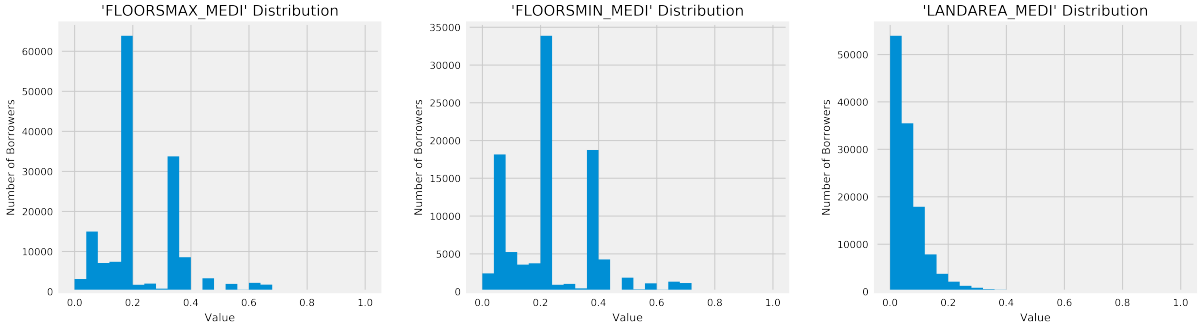


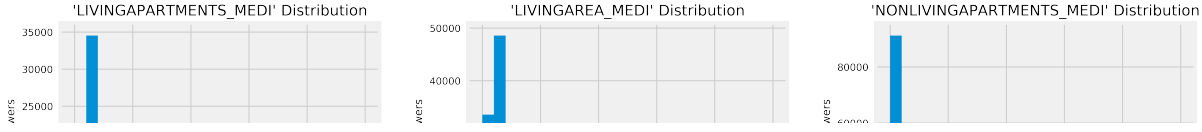

'LANDAREA_MODE' Distribution
'LIVINGAPARTMENTS_MODE' Distribution
'LIVINGAREA_MODE' Distribution
Number of Borrowers
Value

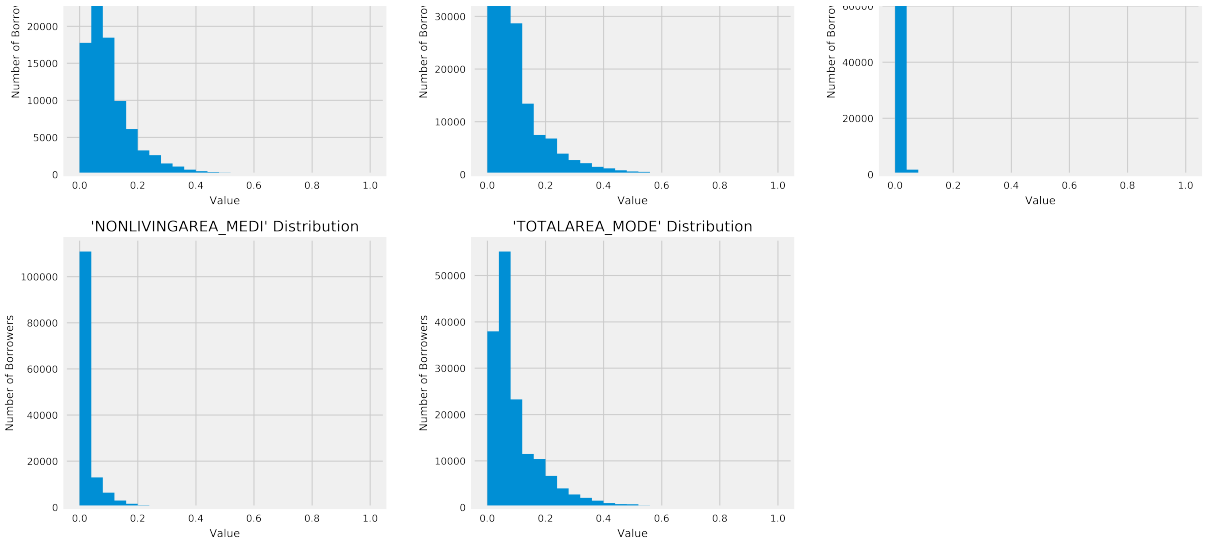

'NONLIVINGAPARTMENTS_MODE' Distribution
'NONLIVINGAREA_MODE' Distribution
'APARTMENTS_MEDI' Distribution
Number of Borrowers
Value


'BASEMENTAREA_MEDI' Distribution
'YEARS_BEGINEXPLUATATION_MEDI' Distribution
'YEARS_BUILD_MEDI' Distribution
Number of Borrowers
Value


'COMMONAREA_MEDI' Distribution
'ELEVATORS_MEDI' Distribution
'ENTRANCES_MEDI' Distribution
Number of Borrowers
Value


'FLOORSMAX_MEDI' Distribution
'FLOORSMIN_MEDI' Distribution
'LANDAREA_MEDI' Distribution
Number of Borrowers
Value

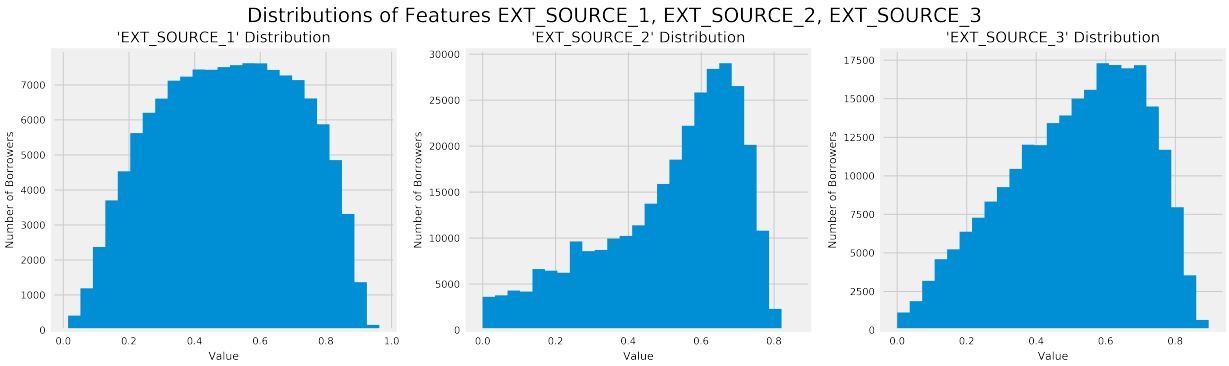

'LIVINGAPARTMENTS_MEDI' Distribution
'LIVINGAREA_MEDI' Distribution
'NONLIVINGAPARTMENTS_MEDI' Distribution



I paid special attention to the distributions of the four features I had found anomalous while initially exploring the dataset, `REGION_POPULATION_RELATIVE`, `EXT_SOURCE_1`, `EXT_SOURCE_2`, and `EXT_SOURCE_3`.

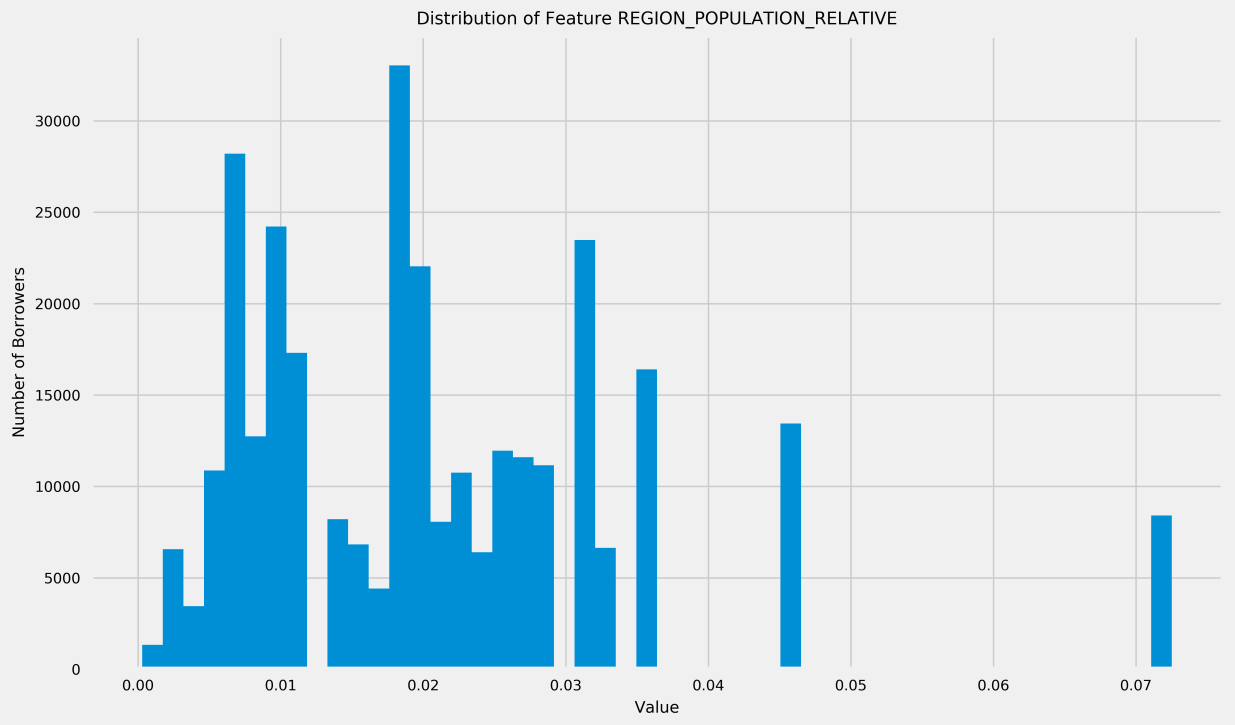
It turned out that `EXT_SOURCE_1`, `EXT_SOURCE_2`, and `EXT_SOURCE_3` were the three features I should have been least concerned about -- the shapes of these features' distributions more closely resembled that of the normal bell curve than the shapes of all the other normalized numerical features. And upon closer review, perhaps this shouldn't be so surprising. According to Home Credit's definitions, these three features represent normalized scores that come from an "external data source," and I can only surmise that whatever methodology the external source used to devise and assign these scores may be what causes their values to be more normally distributed across the dataset.

In [4]:
```
# Plot histogram of ['EXT_SOURCE_1','EXT_SOURCE_2','EXT_SOURCE_3'], om
itting any rows (borrowers) that have a value
# of 'NaN' for the particular feature
vs.plot_feature_distributions(application_train_data[['EXT_SOURCE_1',
'EXT_SOURCE_2', 'EXT_SOURCE_3']], title='Distributions of Features EXT
_SOURCE_1, EXT_SOURCE_2, EXT_SOURCE_3', figsize=(14,4), num_cols=3)
```



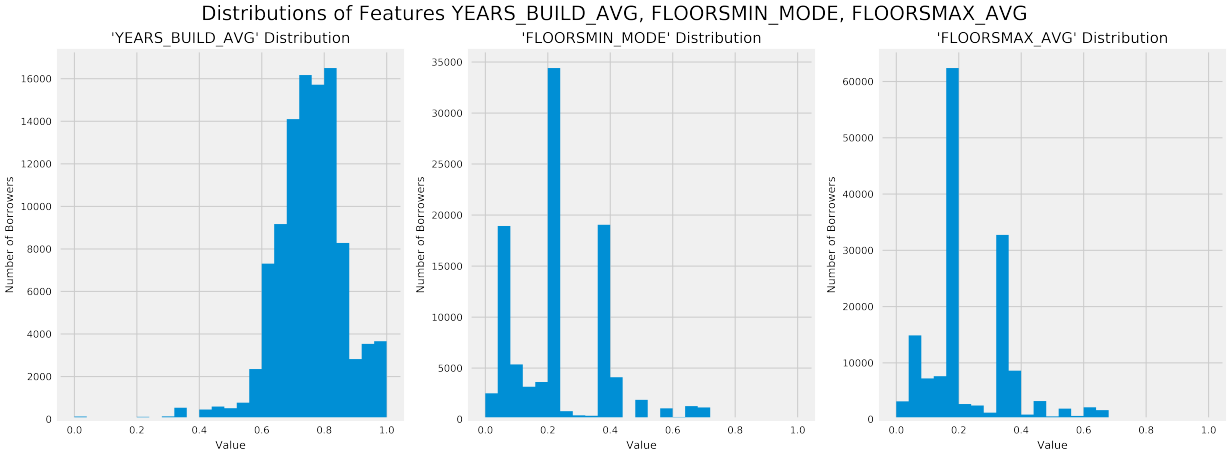
As I had discovered while exploring the dataset, the feature `REGION_POPULATION_RELATIVE` does indeed have all its values within an approximate range of [0.00,0.07]. There is nothing in the feature's definition that suggests why, out of all the other normalized features, that this would be the only feature not scaled to the range [0.0,1.0]. Thankfully, this feature also exhibits minimal positive/negative skewness. The only adjustment necessary would be to min-max scale this feature to the range [0.0,1.0] when preprocessing the data.

In [52]:
```
# Plot histogram of ['REGION_POPULATION_RELATIVE'], omitting any rows
(borrowers) that have a value
# of 'NaN' for the particular feature
region_population_relative_data = application_train_data['REGION_POPUL
ATION_RELATIVE']
filtered_region_population_relative_data = region_population_relative_
data[~np.isnan(region_population_relative_data)]
plt.figure(figsize = (10,6))
plt.hist(filtered_region_population_relative_data, bins=50)
plt.title('Distribution of Feature REGION_POPULATION_RELATIVE')
plt.xlabel('Value')
plt.ylabel('Number of Borrowers')
plt.savefig('distribREGIONPOPULATIONRELATIVE.png')
plt.show()
```



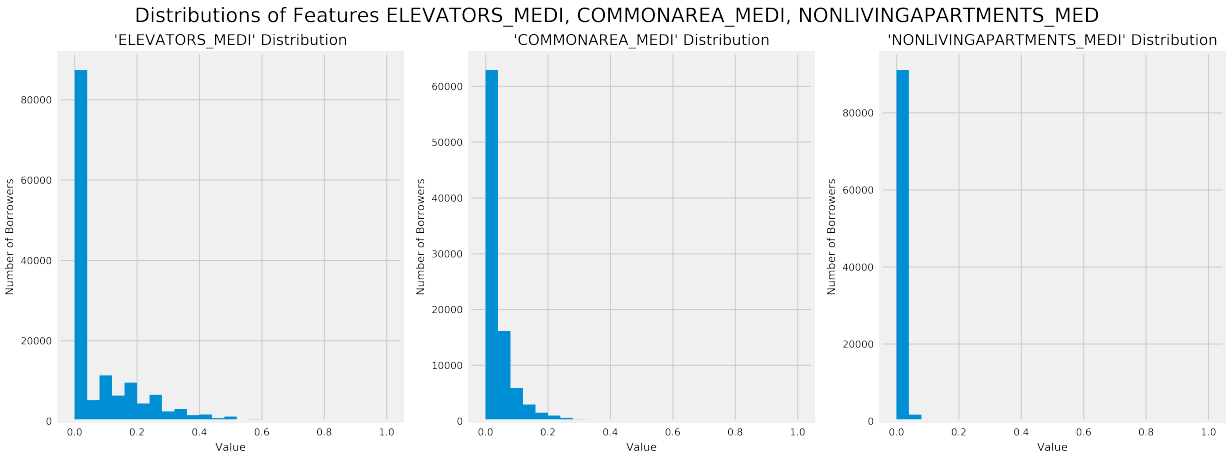
Other normalized features, particularly those concerning general characteristics of a borrower's residence, such as `YEARS_BUILD_AVG`, `FLOORSMIN_MODE`, and `FLOORSMAX_AVG` also had the appearance of a normal distribution without much positive or negative skewness. This intuitively makes sense as these general features about number of floors and the year the residence was built will pertain to all individuals' residences, regardless of the dwelling type.

In [28]:
```
# Plot histogram of ['YEARS_BUILD_AVG', 'FLOORSMIN_MODE, 'FLOORSMAX_AVG'], omitting any rows (borrowers) that have a value
# of 'NaN' for the particular feature
vs.plot_feature_distributions(application_train_data[['YEARS_BUILD_AVG', 'FLOORSMIN_MODE', 'FLOORSMAX_AVG']], title='Distributions of Features YEARS_BUILD_AVG, FLOORSMIN_MODE, FLOORSMAX_AVG', figsize=(14,5), num_cols=3)
```



On the other hand, the rest of the normalized features, such as `ELEVATORS_MEDI`, `COMMONAREA_MEDI`, `NONLIVINGAPARTMENTS_MEDI`, describe more niche characteristics of a residence building that may not apply to many of the borrowers. For example, individuals who live in a house wouldn't be expected to have any elevators, a public common area, or non-living apartments in their residence. And as such, these features tend to be noticeably positively skewed.

In [33]:
```
# Plot histogram of ['EXT_SOURCE_1','EXT_SOURCE_2','EXT_SOURCE_3'], omitting any rows (borrowers) that have a value
# of 'NaN' for the particular feature
vs.plot_feature_distributions(application_train_data[['ELEVATORS_MEDI', 'COMMONAREA_MEDI', 'NONLIVINGAPARTMENTS_MEDI']], title='Distributions of Features ELEVATORS_MEDI, COMMONAREA_MEDI, NONLIVINGAPARTMENTS_MED', figsize=(14,5), num_cols=3)
```

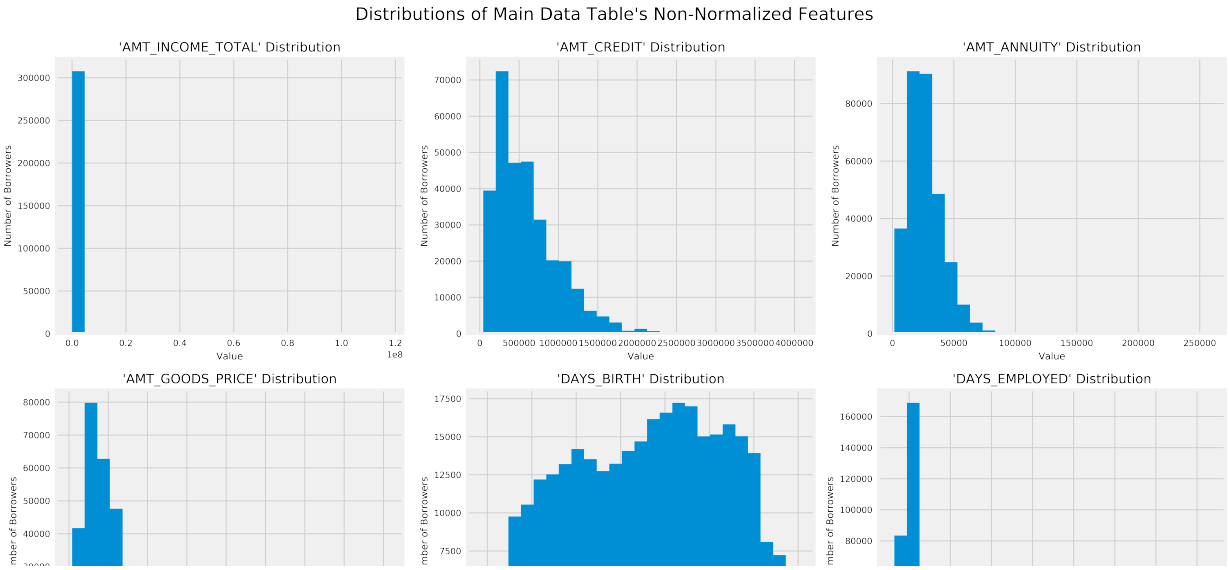


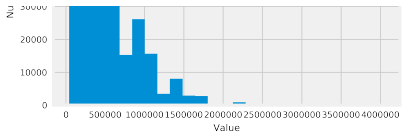
2. Plot Distributions of Non-Normalized Main Data Table Numerical Features

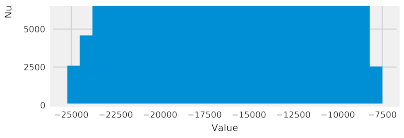
The following 21 numerical features were not identified by Home Credit as being normalized. I plotted histograms of each of these features in order to observe their skewness and to discover which ones would be candidates for log-normalization:

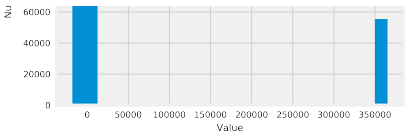
```
'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE', 'DAYS_
BIRTH', 'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'DAYS_ID_PUBLISH', 'OWN_CAR_
AGE', 'HOUR_APPR_PROCESS_START', 'OBS_30_CNT_SOCIAL_CIRCLE', 'DEF_30_CNT_S
OCIAL_CIRCLE', 'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIRCLE', 'DA
YS_LAST_PHONE_CHANGE', 'AMT_REQ_CREDIT_BUREAU_HOUR', 'AMT_REQ_CREDIT_BUREA
U_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK', 'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_RE
Q_CREDIT_BUREAU_QRT', 'AMT_REQ_CREDIT_BUREAU_YEAR'
```

In [4]:
```
# List of non-normalized features
non_normalized_numerical_features = ['AMT_INCOME_TOTAL', 'AMT_CREDIT',
'AMT_ANNUITY', 'AMT_GOODS_PRICE', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'DAYS
_REGISTRATION', 'DAYS_ID_PUBLISH', 'OWN_CAR_AGE', 'HOUR_APPR_PROCESS_S
TART', 'OBS_30_CNT_SOCIAL_CIRCLE', 'DEF_30_CNT_SOCIAL_CIRCLE', 'OBS_60
_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIRCLE', 'DAYS_LAST_PHONE_CHAN
GE', 'AMT_REQ_CREDIT_BUREAU_HOUR', 'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_R
EQ_CREDIT_BUREAU_WEEK', 'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_B
UREAU_QRT', 'AMT_REQ_CREDIT_BUREAU_YEAR']

# Plot histogram of each non-normalized feature, omitting any rows (bo
rrowers) that have a value
# of 'NaN' for the particular feature
vs.plot_feature_distributions(application_train_data[non_normalized_nu
merical_features], title='Distributions of Main Data Table\'s Non-Norm
alized Features', figsize=(16,30), num_cols=3)
```

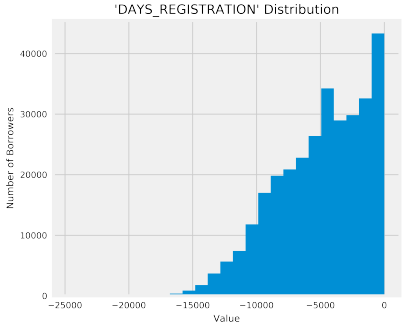




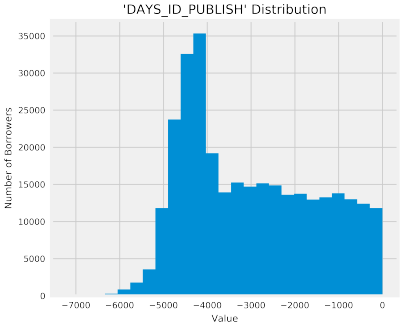




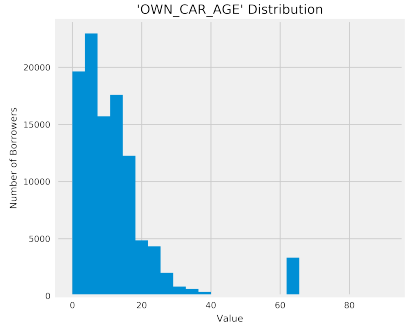
'DAYS_REGISTRATION' Distribution



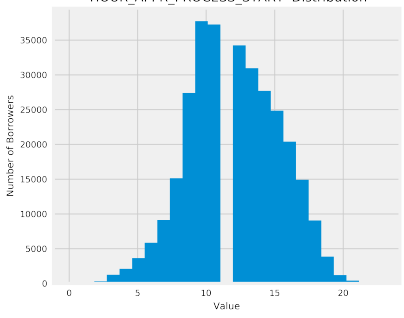
'DAYS_ID_PUBLISH' Distribution



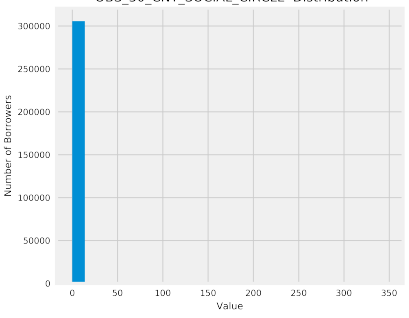
'OWN_CAR_AGE' Distribution



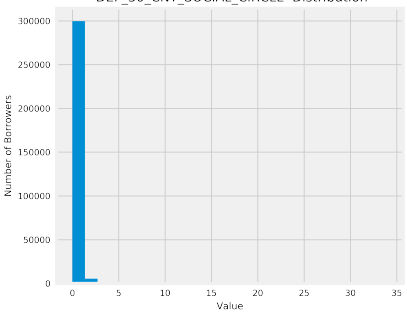
'HOUR_APPR_PROCESS_START' Distribution



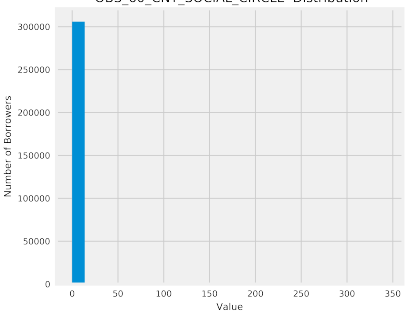
'OBS_30_CNT_SOCIAL_CIRCLE' Distribution



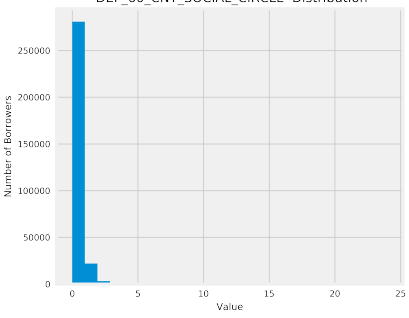
'DEF_30_CNT_SOCIAL_CIRCLE' Distribution



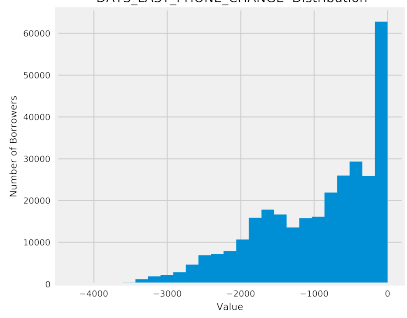
'OBS_60_CNT_SOCIAL_CIRCLE' Distribution



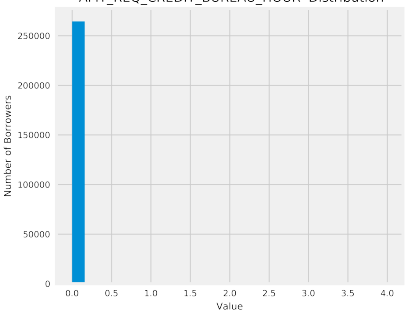
'DEF_60_CNT_SOCIAL_CIRCLE' Distribution



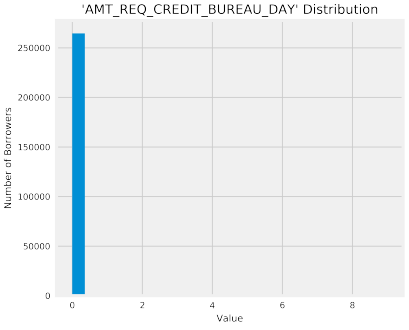
'DAYS_LAST_PHONE_CHANGE' Distribution



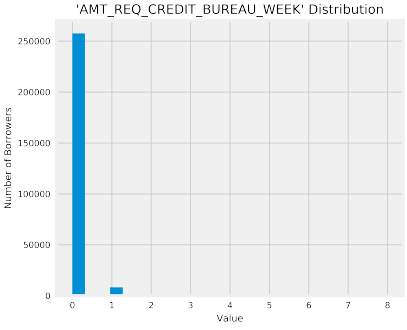
'AMT_REQ_CREDIT_BUREAU_HOUR' Distribution



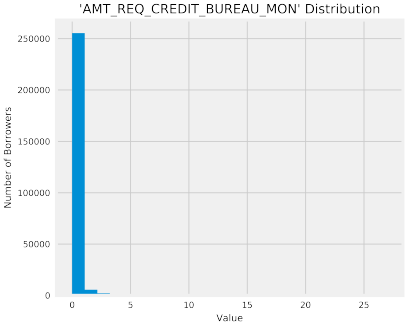
'AMT_REQ_CREDIT_BUREAU_DAY' Distribution



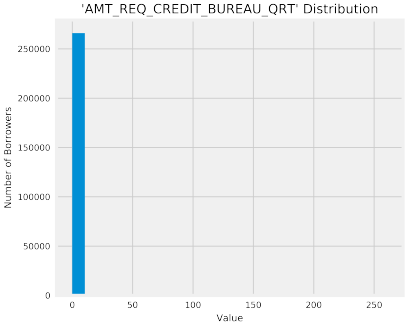
'AMT_REQ_CREDIT_BUREAU_WEEK' Distribution



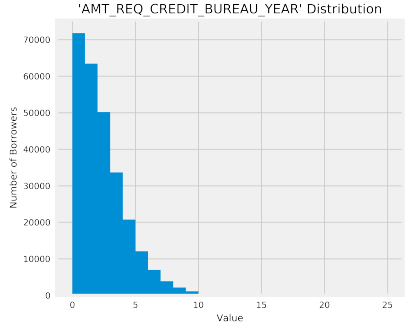
'AMT_REQ_CREDIT_BUREAU_MON' Distribution



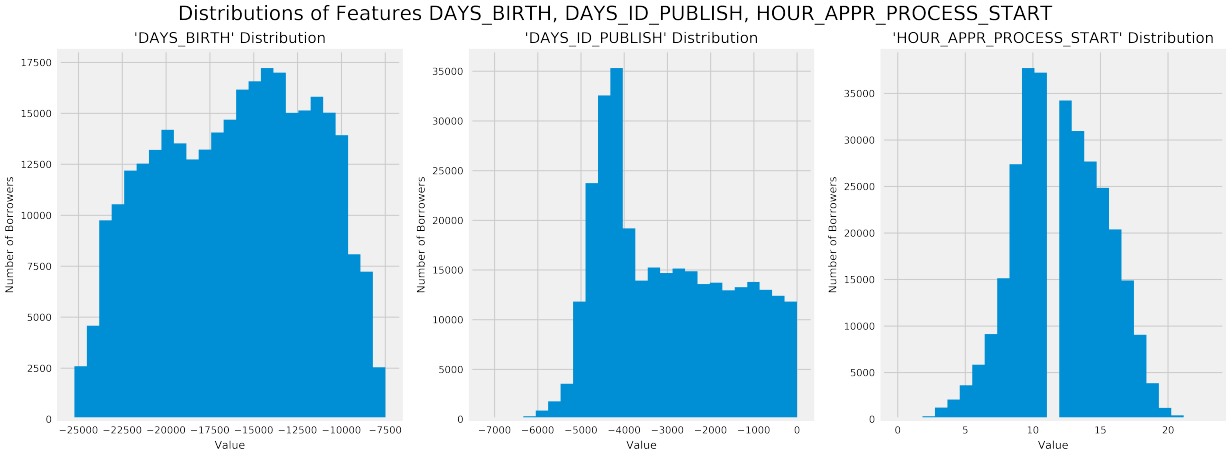
'AMT_REQ_CREDIT_BUREAU_QRT' Distribution



'AMT_REQ_CREDIT_BUREAU_YEAR' Distribution



In [41]:
```
# Plot histogram of ['DAYS_BIRTH','DAYS_ID_PUBLISH','HOUR_APPR_PROCESS
_START'], omitting any rows (borrowers) that have a value
# of 'NaN' for the particular feature
vs.plot_feature_distributions(application_train_data[['DAYS_BIRTH','DA
YS_ID_PUBLISH','HOUR_APPR_PROCESS_START']], title='Distributions of Fe
atures DAYS_BIRTH, DAYS_ID_PUBLISH, HOUR_APPR_PROCESS_START', figsize=
(14,5), num_cols=3)
```



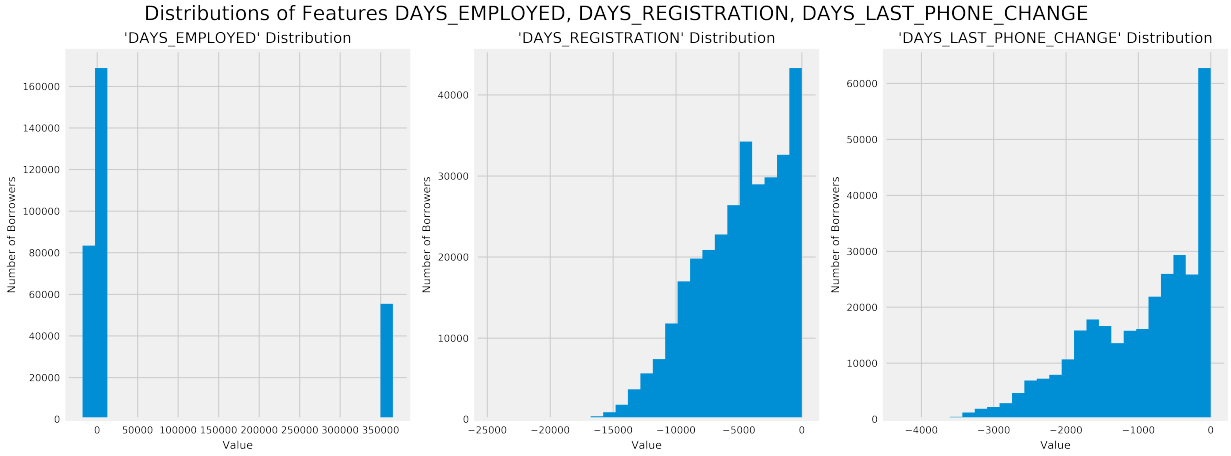
While all of these 21 features will need to be min-max scaled to a range of [0.0,1.0], the following three features already exhibit non-skewed, normal shaped distributions. It will not be necessary for them to be log-normalized:

- `DAYS_BIRTH`
- `DAYS_ID_PUBLISH`
- `HOUR_APPR_PROCESS_START`

The rest of the features, especially those with majority of their values concentrated close to zero yet also having a smattering of large-valued data points, are good candidates for log-normalization. Doing this may prevent these features' very large and very small values from negatively affecting the performance of a learning algorithm.

There are three non-normalized features `DAYS_EMPLOYED`, `DAYS_REGISTRATION`, and `DAYS_LAST_PHONE_CHANGE` that both have skewed distributions, as well as values that fall inside a range of negative numbers. Because log-transformation cannot be run on negative values, these features' distributions would first need to be translated positively to the right, such that all their values are greater than or equal to zero.

In [35]:
```
# Plot histogram of ['DAYS_EMPLOYED','DAYS_REGISTRATION','DAYS_LAST_PHONE_CHANGE'], omitting any rows (borrowers) that have skewed
# distributions over a range of negative values.
vs.plot_feature_distributions(application_train_data[['DAYS_EMPLOYED', 'DAYS_REGISTRATION','DAYS_LAST_PHONE_CHANGE']], title='Distributions of Features DAYS_EMPLOYED, DAYS_REGISTRATION, DAYS_LAST_PHONE_CHANGE', figsize=(14,5), num_cols=3)
```

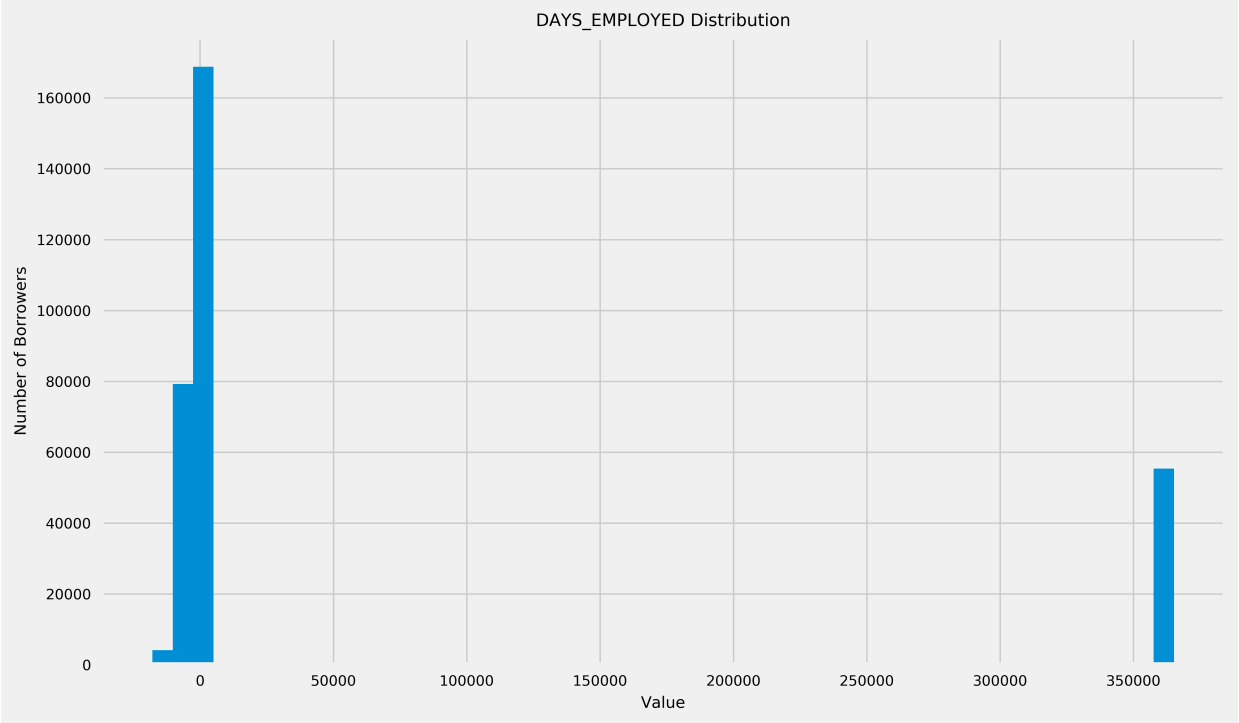


Unfortunately, `DAYS_EMPLOYED` and `OWN_CAR_AGE` contain an alarming amount of high-valued outliers that likely need to be addressed. I will further explore each of these features below.

3. Deeper Dive into the `DAYS_EMPLOYED` Feature

I paid extra close attention to the histogram of `DAYS_EMPLOYED`, in order to observe whether it has other impossibly large, positive entries similar to the value of 365,243 days, or just over 1,000 years, that I observed above.

In [54]:
```
# Draw a larger plot of the DAYS_EMPLOYED feature's histogram
plt.figure(figsize = (10,6))
plt.hist(application_train_data['DAYS_EMPLOYED'], bins=50)
plt.title('DAYS_EMPLOYED Distribution')
plt.xlabel('Value')
plt.ylabel('Number of Borrowers')
plt.savefig('distribDAYSEMPLOYED.png')
plt.show()
```



In [66]:
```
application_train_data['DAYS_EMPLOYED'].value_counts().sort_index(ascending=False)
```

Out[66]:

365243	55374
0	2
-1	1
-2	2
-3	3
-4	4
-5	1
-6	2
-7	1
-8	2
-9	3
-10	1
-11	2
-12	6
-13	3
-14	3
-15	6
-16	7
-17	5
-18	3

-19	3
-20	3
-21	2
-22	2
-23	1
-24	3
-25	3
-26	7
-27	2
-28	4
-29	5
-30	4
-31	8
-32	2
-33	2
-34	7
-35	3
-36	3
-37	5
-38	10
-39	2
-40	3
-41	12
-42	10
-43	13
-44	7
-45	6
-46	10
-47	4
-48	6
-49	9
-50	3
-51	10
-52	6
-53	6
-54	11
-55	10
-56	11
-57	11
-58	9
-59	12
-60	6
-61	6
-62	11
-63	11
-64	11
-65	12
-66	14
-67	15
-68	16
-69	15
-70	9
-71	8

−72	11
−73	16
−74	22
−75	15
−76	23
−77	18
−78	24
−79	24
−80	17
−81	35
−82	28
−83	17
−84	22
−85	30
−86	28
−87	27
−88	29
−89	35
−90	41
−91	37
−92	62
−93	52
−94	51
−95	62
−96	81
−97	69
−98	81
−99	78
−100	63
−101	81
−102	103
−103	75
−104	95
−105	105
−106	114
−107	104
−108	99
−109	115
−110	81
−111	96
−112	111
−113	111
−114	77
−115	117
−116	136
−117	102
−118	119
−119	116
−120	99
−121	98
−122	99
−123	104
−124	107

−125	112
−126	106
−127	111
−128	98
−129	116
−130	109
−131	86
−132	117
−133	117
−134	100
−135	112
−136	105
−137	123
−138	98
−139	122
−140	105
−141	109
−142	94
−143	108
−144	94
−145	77
−146	108
−147	100
−148	90
−149	82
−150	90
−151	70
−152	81
−153	96
−154	87
−155	91
−156	90
−157	104
−158	83
−159	87
−160	85
−161	99
−162	100
−163	114
−164	109
−165	99
−166	76
−167	105
−168	92
−169	94
−170	106
−171	93
−172	95
−173	95
−174	96
−175	82
−176	91
−177	108

−178	98
−179	108
−180	100
−181	118
−182	108
−183	101
−184	126
−185	128
−186	110
−187	106
−188	137
−189	102
−190	97
−191	120
−192	114
−193	134
−194	127
−195	133
−196	136
−197	114
−198	122
−199	151
−200	156
−201	130
−202	110
−203	112
−204	116
−205	120
−206	115
−207	138
−208	115
−209	125
−210	112
−211	104
−212	150
−213	131
−214	132
−215	138
−216	137
−217	111
−218	120
−219	106
−220	110
−221	106
−222	135
−223	119
−224	152
−225	119
−226	107
−227	118
−228	117
−229	143
−230	151

-231	140
-232	105
-233	128
-234	126
-235	100
-236	115
-237	127
-238	122
-239	93
-240	120
-241	124
-242	103
-243	90
-244	118
-245	79
-246	107
-247	93
-248	106
...	
-14441	2
-14445	1
-14452	1
-14453	1
-14455	1
-14456	2
-14460	1
-14468	2
-14473	3
-14474	1
-14478	1
-14482	1
-14487	1
-14507	2
-14509	1
-14513	1
-14522	3
-14536	1
-14540	2
-14541	1
-14543	1
-14553	1
-14556	1
-14559	2
-14571	1
-14574	1
-14584	1
-14589	1
-14590	3
-14597	1
-14604	1
-14607	1
-14619	1
-14620	1

-14627	1
-14628	1
-14637	1
-14640	1
-14643	1
-14644	1
-14648	1
-14651	1
-14660	1
-14666	1
-14679	1
-14688	1
-14689	1
-14691	1
-14694	1
-14701	1
-14719	1
-14722	2
-14723	1
-14726	1
-14743	1
-14747	1
-14749	1
-14756	1
-14775	1
-14778	2
-14786	1
-14797	1
-14801	1
-14802	1
-14805	1
-14810	1
-14819	1
-14829	1
-14832	1
-14833	1
-14848	1
-14849	1
-14850	1
-14854	1
-14860	2
-14872	1
-14887	1
-14894	1
-14910	2
-14915	1
-14928	1
-14933	1
-14949	1
-14955	1
-14957	1
-14966	1
-14968	1

-14974	1
-14977	1
-14978	1
-14981	1
-14985	2
-14988	1
-14990	1
-14991	1
-15019	1
-15021	1
-15027	1
-15034	1
-15038	1
-15043	1
-15048	1
-15051	2
-15052	1
-15057	1
-15059	1
-15060	1
-15066	2
-15072	1
-15080	1
-15084	1
-15087	1
-15094	1
-15095	1
-15107	2
-15116	1
-15128	1
-15137	1
-15147	2
-15153	1
-15154	1
-15155	1
-15180	1
-15181	1
-15183	1
-15202	1
-15203	1
-15210	1
-15226	1
-15227	1
-15229	1
-15238	1
-15277	1
-15285	1
-15290	1
-15300	1
-15303	1
-15323	1
-15327	1
-15334	1

-15338	1
-15342	1
-15348	1
-15360	1
-15368	1
-15369	1
-15371	1
-15382	1
-15396	1
-15412	1
-15422	1
-15427	1
-15431	1
-15439	1
-15473	1
-15474	2
-15476	1
-15477	1
-15488	1
-15499	1
-15509	2
-15516	1
-15524	1
-15530	2
-15542	1
-15543	1
-15568	1
-15569	1
-15578	1
-15625	1
-15629	1
-15632	1
-15661	1
-15676	1
-15687	1
-15688	1
-15689	1
-15691	1
-15699	1
-15713	1
-15726	1
-15727	1
-15729	1
-15783	1
-15791	1
-15834	1
-15837	1
-15845	2
-15860	1
-15871	1
-15882	1
-15890	1
-15911	1

-15943	1
-16032	1
-16037	1
-16061	1
-16069	1
-16093	1
-16103	1
-16113	1
-16121	1
-16133	1
-16135	1
-16142	1
-16160	1
-16169	1
-16220	1
-16221	1
-16236	1
-16260	1
-16263	1
-16265	1
-16266	1
-16304	1
-16308	1
-16310	1
-16314	1
-16343	1
-16348	1
-16352	1
-16358	1
-16360	1
-16364	1
-16365	1
-16375	1
-16424	1
-16429	1
-16452	1
-16481	1
-16492	1
-16495	2
-16499	1
-16538	1
-16554	1
-16607	1
-16632	1
-16651	1
-16678	1
-16767	1
-16836	1
-16849	1
-16852	1
-17139	1
-17170	1
-17522	1

```
-17531          1
-17546          1
-17583          1
-17912          1
Name: DAYS_EMPLOYED, Length: 12574, dtype: int64
```

Based on the definition of `DAYS_EMPLOYED`, valid values should be in the range (-inf,0]. Unfortunately, 55,374 entries, or nearly one-sixth of the entire training dataset, has an entry of `365243` for this feature. Thankfully, `DAYS_EMPLOYED` has no 'NaN' entries. Were this value to be interpreted literally, it would indicate that one-sixth of the dataset got a job just over 1,000 years after submitting their loan applications to Home Credit.

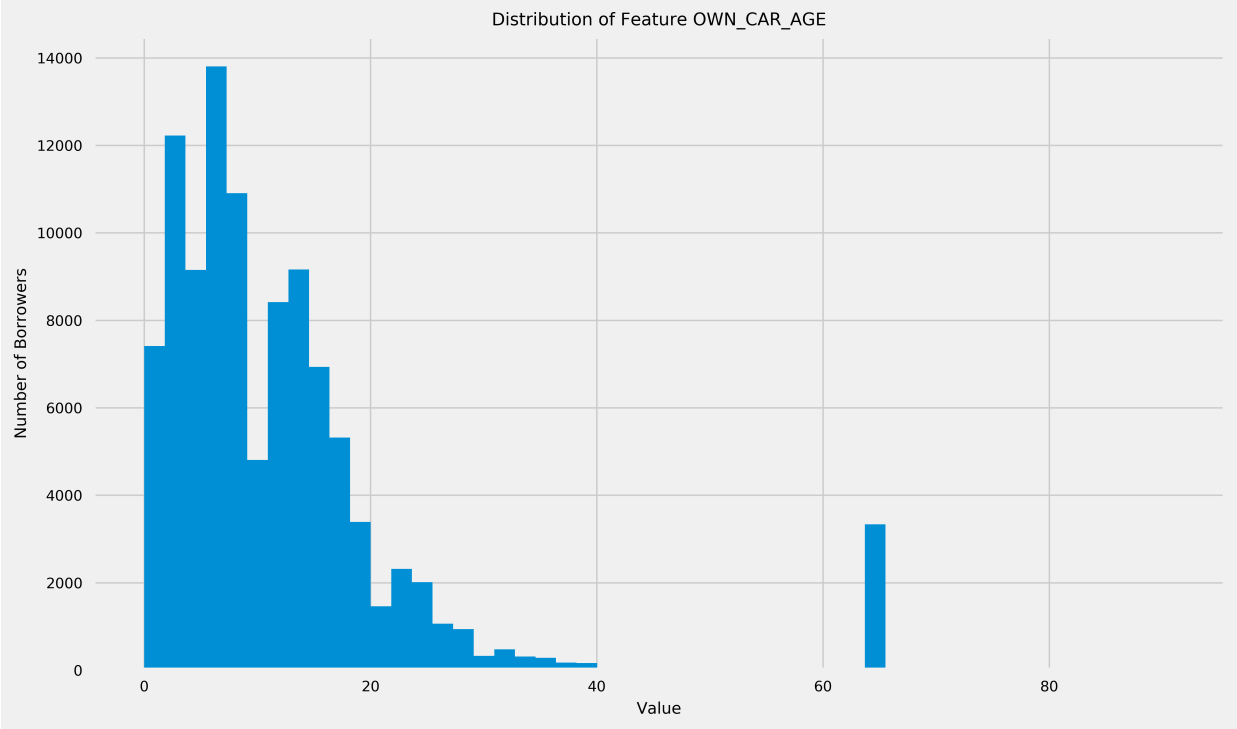
This meaning is obviously absurd and there has to be another reason that so many borrowers had the value of `365243` entered for this feature. Since there were no instances of borrowers having a different non-zero positive value for this feature, my best guess is that `365243` was not meant to indicate a numerical value. I believe that this value was entered for applicants who did not have a job when they submitted their loan application to Home Credit. Since any negative integer or 0 would be a valid entry for this feature, and perhaps due to a data entry system's inability to accept any value besides an integer, I hypothesize that the original data enterers simply entered the largest positive integer that the system would accept in order to indicate that the applicant didn't have a job.

Intuitively, I can see that `DAYS_EMPLOYED` may well be a good predictor of target segments -- after all, if someone doesn't have a job, it stands to reason that there is a greater chance they won't have enough money to make loan payments on time. Unfortunately, I am not confident that the feature, as currently structured, would be able to adequately convey this information to a learning algorithm.

I propose to replace the `DAYS_EMPLOYED` feature with a new categorical feature called `HAS_JOB`. All individuals who have a value of `365243` for `DAYS_EMPLOYED` will be assigned a value of `0` for `HAS_JOB`. All individuals who have a value of 0 or less for `DAYS_EMPLOYED` will be assigned a value of `1` for `HAS_JOB`.

4. Deeper Dive into the `OWN_CAR_AGE` Feature

In [56]:
```
# Plot histogram of ['OWN_CAR_AGE'], omitting any rows (borrowers) tha
t have a value
# of 'NaN' for the particular feature
own_car_age_data = application_train_data['OWN_CAR_AGE']
filtered_own_car_age_data = own_car_age_data[~np.isnan(own_car_age_dat
a)]
plt.figure(figsize = (10,6))
plt.hist(filtered_own_car_age_data, bins=50)
plt.title('Distribution of Feature OWN_CAR_AGE')
plt.xlabel('Value')
plt.ylabel('Number of Borrowers')
plt.savefig('distribOWNCARAGE.png')
plt.show()
```



There appear to be just under 4,000 borrowers who have a value for `OWN_CAR_AGE` that's between 60 and 70. This is far too many people to indicate anything but some sort of anomaly. The only kinds of cars that are still functional after 60+ years are collectible classic cars, and folks who apply for loans from Home Credit come from a far less well-off demographic than that which is associated with classic car collection.

In [40]:
```
application_train_data['OWN_CAR_AGE'].value_counts().sort_index(ascend
ing=False)
```

Out[40]:
```
91.0       2
69.0       1
65.0     891
64.0    2443
63.0       2
57.0       1
56.0       1
55.0       4
```

54.0	12
52.0	1
51.0	3
50.0	1
49.0	6
48.0	1
47.0	1
46.0	5
45.0	11
44.0	21
43.0	19
42.0	42
41.0	58
40.0	85
39.0	78
38.0	97
37.0	75
36.0	124
35.0	157
34.0	183
33.0	132
32.0	208
31.0	267
30.0	326
29.0	397
28.0	542
27.0	483
26.0	580
25.0	865
24.0	1150
23.0	1067
22.0	1250
21.0	1462
20.0	1527
19.0	1864
18.0	2418
17.0	2899
16.0	3355
15.0	3580
14.0	4594
13.0	4566
12.0	4257
11.0	4161
10.0	4806
9.0	5020
8.0	5887
7.0	7424
6.0	6382
5.0	3595
4.0	5557
3.0	6370
2.0	5852
1.0	5280

```
0.0     2134
Name: OWN_CAR_AGE, Length: 62, dtype: int64
```

Nonetheless, based on the unique value counts above, I can see right away that the distribution of `OWN_CAR_AGE` is less problematic than was the case for `DAYS_EMPLOYED`. There is at least a smooth decrease in numbers of users having older cars, right up until the anomalous spike.

However, the distribution's pattern would indicate that I should expect only one or two borrowers each to have cars that are 64 and 65 years old. I can't hazard a reasonable guess that could explain why a total of 3,334 individuals have cars aged 64 or 65 years, and I can't formulate a compelling justification for removing these entries from the `OWN_CAR_AGE` feature. The good news is that this is probably ok. Since 3,334 borrowers is only just over 3% of the `OWN_CAR_AGE` feature's 104,582 valid non-'NaN' entries, log-normalizing this feature should be enough to counteract any negative effects that the anomalous spike may have on a learning algorithm.

III. Algorithms and Techniques

In this section I lay out a roadmap for devising a learning algorithm that predicts which borrowers will make at least one late loan payment (which borrowers have a `TARGET` value of 1). My approach takes into account everything I learned while exploring and visualizing the main data table.

Data Preprocessing:

1. Create 7 lists that categorize the features into following different groups based on how each feature will need to be preprocessed: categorical features needing one-hot encoding, binary categorical features, non-normalized numerical features with skewed distributions and negative values, non-normalized numerical features with skewed distributions and only positive values, numerical features wit normal distributions but not scaled to the range [0,1], numerical features with normal distributions and scaled to the range [0,1], and finally, features that will be re-engineered and transformed into different features.
2. Separate targets column from training dataset.
3. Use train_test_split from sklearn.cross_validation to create a test validation set that is 20% of the size of the total training set: Will allow me to compare performance of various learning algorithms without overfitting to the training data.
4. Use the numerical `CNT_CHILDREN` feature to engineer a binary categorical feature called `HAS_CHILDREN`.
5. Drop `CNT_CHILDREN` from the main dataframe.
6. Use the `CNT_FAM_MEMBERS` feature to engineer a categorical feature called `FAMILY_SIZE`.
7. Drop `CNT_FAM_MEMBERS` from the main dataframe.
8. Use the `CREDIT_DAY_OVERDUE` feature in `bureau.csv` to engineer the binary categorical `HAS_CREDIT_BUREAU_LOANS_OVERDUE` feature.
9. Use the `DAYS_EMPLOYED` feature to engineer a binary categorical feature called `HAS_JOB`.

10. Drop the DAYS_EMPLOYED feature from the main dataframe.
11. Translate to positive ranges all values of the 2 non-normalized numerical features that have skewed distributions and negative values: DAYS_REGISTRATION, and DAYS_LAST_PHONE_CHANGE.
12. Log-transform all 17 non-normalized numerical features that have skewed distributions. These 17 features include the 2 that were translated to positive ranges in Step 11.
13. Replace 'NaN' values for all numerical features with each feature's mean using Sklearn's Imputer class.
14. Remove the borrower ID column, SK_ID_CURR, from the main dataframe.
15. One-hot encode all 19 non-binary categorical features.
16. Replace all 'NaN' values in all binary categorical features with 0.
17. Fit a min-max scaler to each of the 17 log-transformed numerical features, as well as to the features DAYS_BIRTH, DAYS_ID_PUBLISH, HOUR_APPR_PROCESS_START, and the normalized feature REGION_POPULATION_RELATIVE. Each feature will be scaled to a range [0.0, 1.0].
18. Build a data preprocessing pipeline to used for all testing sets. This pipeline will recreate all features that were engineered in the training set during the original data preprocessing phase. It will impute missing numerical values with each column's mean, and replace missing 'NaN' values of binary categorical features with 0. The pipeline will also apply the min-max scaling transforms originally fit on features in the training set to all data points in a testing set. Finally, any columns representing binary one-hot encoded features that are in the training set but absent from a test set will be added to the test set and filled with all 0s. Similarly, any columns in a test dataframe that represent one-hot encoded features that aren't present in the training set will be removed from the test dataframe.
19. Run this pipeline to preprocess the test validation set.

Implementation:

1. Create an ROC area-under-curve scorer method.
2. Use the Gaussian Naive Bayes classifier to make predictions on the test validation set. Calculate the area under ROC curve score of these predictions.
3. Perform GridSearchCV on an AdaBoost classifier learning algorithm to discover the highest scoring hyperparameter combination by exploring the trade-off between the learning_rate and n_estimators parameters.
4. Make predictions on the test validation set using both the AdaBoost classifier with the hyperparameters that had been chosen using GridSearchCV. Calculate the area under ROC curve score of these predictions.
5. Use a Logistic Regression classifier to make predictions, and calculate its ROC AUC score.
6. Use a Multi Layer Perceptron classifier to make predictions, and calculate its ROC AUC score.
7. Use a LightGBM classifier to make predictions, and calculate its ROC AUC score.

Refinement:

1. Use PCA to reduce the dimensionality of the 67 numerical features that are being used.
    A. Fit PCA on all 67 numerical features, trying different values for the n_components parameter, observing how many principle components are needed to explain roughly 90% of the variance in the numerical features.
    B. Use PCA to reduce the dimension space of the numerical features to the optimal number of principle components discovered in Step 1. Create a dataframe to contain these reduced features.

    C. Drop all 67 numerical features from the original preprocessed dataframe, so that it only contains the 184 binary categorical features. Append the dataframe containing the reduced numerical features back to this original dataframe.
2. Use SelectKBest with different values of k to see if there is a group of k features that causes the learning algorithms to perform better when they are trained on just these k features.
3. Select the featurespace (All, PCA-reduced, or SelectKBest) and learning algorithm that has had the highest ROC AUC score on the test validation set thus far. Further tune this algorithm's hyperparameters, using GridSearchCV to gain further intuition while tuning.
4. Create a prediction pipeline that uses the best performing learning algorithm to make predictions on a the actual test datatable, `application_test.csv`, returns the area under the ROC curve as a score, and outputs a CSV file containing the posterior probabilities of the classifier's predictions for a testing data set.
5. Train the learning algorithm on the entire training data table in `application_train.csv`.
6. Preprocess the data in `application_test.csv`.
7. Make predictions on the test data set in application_test.csv using the best learning algorithm. Submit the CSV file containing prediction probabilities to Kaggle and observe the score received from Kaggle.

IV. Data Preprocessing

In [578]:
```
# Some imports are redundant with imports made in the early code block
s
# of this notebook. Repeated here for convenience, so that code blocks
# from much higher up don't have to be re-executed when re-initiating
# this notebook.

# Import necessary libraries.
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns

# Import supplementary visualizations code visuals.py
import visuals as vs

# Display matplotlib plots inline in this notebook.
%matplotlib inline
# Make plots display well on retina displays
%config InlineBackend.figure_format = 'retina'
# Set dpi of plots displayed inline
mpl.rcParams['figure.dpi'] = 300
# Configure style of plots
plt.style.use('fivethirtyeight')
# Make plots smaller
sns.set_context('paper')

# Allows the use of display() for dataframes.
```

```
from IPython.display import display
# Have all columns appear when dataframes are displayed.
pd.set_option('display.max_columns', None)
# Have 100 rows appear when a dataframe is displayed
pd.set_option('display.max_rows', 500)
# Display dimensions whenever a dataframe is printed out.
pd.set_option('display.show_dimensions', True)

# Import data preprocessing libraries
from sklearn.preprocessing import Imputer
from sklearn.preprocessing import MinMaxScaler

# Import feature selection/dimensionality reduction libraries
from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest, f_classif, chi2

# Import learning algorithms
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import AdaBoostClassifier

# Import ROC area-under-curve score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import make_scorer

# Import train-test split, ShuffleSplit, GridSearchCV, and K-fold cros
s validation
from sklearn.model_selection import train_test_split
from sklearn.model_selection import PredefinedSplit
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import GridSearchCV, ParameterGrid
from sklearn.model_selection import StratifiedKFold

# Import a Logistic Regression classifier
from sklearn.linear_model import LogisticRegression

# Import a Multi-layer Perceptron classifier
from sklearn.neural_network import MLPClassifier

# Import a LightGBM classifier
import lightgbm as lgb

# In order to create CSV files
import csv
```

In [579]:
```
# Load the main data tables
application_train_data = pd.read_csv("data/application_train.csv")
application_test_data = pd.read_csv("data/application_test.csv")

# Load the Bureau data table
bureau_data = pd.read_csv("data/bureau.csv")
```

In [580]:
```
# Step 1: Create lists of different feature types in the main data
```

```
# frame, based on how each type will need to be preprocessed.

# 1. All 18 categorical features needing one-hot encoding.
#    Includes the 4 categorical features originally
#    mis-identified as having been normalized:
#    EMERGENCYSTATE_MODE, HOUSETYPE_MODE, WALLSMATERIAL_MODE,
#    FONDKAPREMONT_MODE
cat_feat_need_one_hot = [
    'NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR',
    'FLAG_OWN_REALTY', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE',
    'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'REGION_RATING_CLIENT',
    'REGION_RATING_CLIENT_W_CITY', 'WEEKDAY_APPR_PROCESS_START', 'ORGA
NIZATION_TYPE',
    'NAME_TYPE_SUITE', 'OCCUPATION_TYPE', 'EMERGENCYSTATE_MODE',
    'HOUSETYPE_MODE', 'WALLSMATERIAL_MODE', 'FONDKAPREMONT_MODE'
]

# 2. All 32 binary categorical features already one-hot encoded.
bin_cat_feat = [
    'FLAG_MOBIL', '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'
]

# 3. All 2 non-normalized numerical features with skewed distributions
#    and negative values. These features will need to have their
#    distributions translated to positive ranges before being
#    log-transformed, and then later scaled to the range [0,1].
non_norm_feat_neg_values_skewed = [
    'DAYS_REGISTRATION', 'DAYS_LAST_PHONE_CHANGE'
]


# 4. All 15 non-normalized numerical features with skewed distribution
s,
#    and only positive values. These features will need to be
#    log-transformed, and eventually scaled to the range [0,1].
non_norm_feat_pos_values_skewed = [
    'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY',
    'AMT_GOODS_PRICE', 'OBS_30_CNT_SOCIAL_CIRCLE', 'DEF_30_CNT_SOCIAL_
CIRCLE',
    'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIRCLE', 'AMT_REQ_C
REDIT_BUREAU_HOUR',
```

```
    'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK', 'AMT_RE
Q_CREDIT_BUREAU_MON',
    'AMT_REQ_CREDIT_BUREAU_QRT', 'AMT_REQ_CREDIT_BUREAU_YEAR', 'OWN_CA
R_AGE'
]

# 5. All 4 numerical features with normal shapes but needing to be sca
led
#    to the range [0,1].
norm_feat_need_scaling = [
    'DAYS_BIRTH', 'DAYS_ID_PUBLISH', 'HOUR_APPR_PROCESS_START',
    'REGION_POPULATION_RELATIVE'
]

# 6. All 46 numerical features that have been normalized to the range
#   [0,1]. These features will need neither log-transformation, nor
#   any further scaling.
norm_feat_not_need_scaling = [
    'EXT_SOURCE_2', 'EXT_SOURCE_3', 'YEARS_BEGINEXPLUATATION_AVG',
    'YEARS_BEGINEXPLUATATION_MODE', 'YEARS_BEGINEXPLUATATION_MEDI', 'F
LOORSMAX_AVG',
    'FLOORSMAX_MODE', 'FLOORSMAX_MEDI', 'LIVINGAREA_AVG',
    'LIVINGAREA_MODE', 'LIVINGAREA_MEDI', 'ENTRANCES_AVG',
    'ENTRANCES_MODE', 'ENTRANCES_MEDI', 'APARTMENTS_AVG',
    'APARTMENTS_MODE', 'APARTMENTS_MEDI', 'ELEVATORS_AVG',
    'ELEVATORS_MODE', 'ELEVATORS_MEDI', 'NONLIVINGAREA_AVG',
    'NONLIVINGAREA_MODE', 'NONLIVINGAREA_MEDI', 'EXT_SOURCE_1',
    'BASEMENTAREA_AVG', 'BASEMENTAREA_MODE', 'BASEMENTAREA_MEDI',
    'LANDAREA_AVG', 'LANDAREA_MODE', 'LANDAREA_MEDI',
    'YEARS_BUILD_AVG', 'YEARS_BUILD_MODE', 'YEARS_BUILD_MEDI',
    'FLOORSMIN_AVG', 'FLOORSMIN_MODE', 'FLOORSMIN_MEDI',
    'LIVINGAPARTMENTS_AVG', 'LIVINGAPARTMENTS_MODE', 'LIVINGAPARTMENTS
_MEDI',
    'NONLIVINGAPARTMENTS_AVG', 'NONLIVINGAPARTMENTS_MODE', 'NONLIVINGA
PARTMENTS_MEDI',
    'COMMONAREA_AVG', 'COMMONAREA_MODE', 'COMMONAREA_MEDI',
    'TOTALAREA_MODE'
]

# 7. The remaining 3 features in the main data frame that will be
#    re-engineered and transformed into different features
feat_to_be_reengineered = [
    'CNT_CHILDREN', 'CNT_FAM_MEMBERS', 'DAYS_EMPLOYED'
]
```

In [581]:
```
# Verify that all 120 features in the main data frame have been catego
rized
# according to how they will be preprocessed.
count_of_categorized_features = len(cat_feat_need_one_hot) + len(bin_c
at_feat) + len(non_norm_feat_neg_values_skewed)\
+ len(non_norm_feat_pos_values_skewed) + len(norm_feat_need_scaling) +
len(norm_feat_not_need_scaling) + len(feat_to_be_reengineered)
print('Number of features in main data frame that have been categorize
d: {}. Expected: 120.'.format(count_of_categorized_features))
```

```
Number of features in main data frame that have been categorized: 12
0. Expected: 120.
```

In [582]:
```
#Step 2: Separate target data from training dataset.
targets = application_train_data['TARGET']
features_raw = application_train_data.drop('TARGET', axis = 1)
```

In [583]:
```
# Step 3: Use train_test_split from sklearn.cross_validation to
# create a test validation set that is 20% of the size of the total tr
aining set:
# Will allow me to compare performance of various learning algorithms
without
# overfitting to the training data.
X_train_raw, X_test_raw, y_train, y_test = train_test_split(features_r
aw,
                                                            targets,
                                                            test_size = 0.2,
                                                            random_state = 42)
```

In [584]:
```
# Step 4: Use the CNT_CHILDREN feature to engineer a binary
# categorical feature called HAS_CHILDREN. If value of CNT_CHILDREN is
# greater than 0, the value of HAS_CHILDREN will be 1. If value of CNT
_CHILDREN is
# 0, value of HAS_CHILDREN will be 0.
CNT_CHILDREN_train = X_train_raw['CNT_CHILDREN']
HAS_CHILDREN = CNT_CHILDREN_train.map(lambda x: 1 if x > 0 else 0)

# Append the newly engineered HAS_CHILDREN feature to the main datafra
me.
X_train_raw = X_train_raw.assign(HAS_CHILDREN=HAS_CHILDREN.values)
```

In [585]:
```
# Step 5: Drop the CNT_CHILDREN column from the main dataframe
X_train_raw = X_train_raw.drop('CNT_CHILDREN', axis=1)

# Add the new HAS_CHILDREN feature to the list of binary categorical
# features that are already one-hot encoded. There are now 33 such fea
tures.
bin_cat_feat = bin_cat_feat + ['HAS_CHILDREN']
```

In [586]:
```
# Step 6. Use the CNT_FAM_MEMBERS feature to engineer a categorical fe
ature called NUMBER_FAMILY_MEMBERS.
# If CNT_FAM_MEMBERS is 1.0, then the value of NUMBER_FAMILY_MEMBERS w
ill be 'one'. If CNT_FAM_MEMBERS is 2.0,
# then NUMBER_FAMILY_MEMBERS will be 'two'. If CNT_FAM_MEMBERS is 3.0
or greater, then NUMBER_FAMILY_MEMBERS will
# be 'three_plus'.
CNT_FAM_MEMBERS_train = X_train_raw['CNT_FAM_MEMBERS']
NUMBER_FAMILY_MEMBERS = CNT_FAM_MEMBERS_train.map(lambda x: 'one' if x
== 1 else ('two' if x == 2 else 'three_plus'))

# Append the newly engineered FAMILY_SIZE feature to the main datafram
e.
X_train_raw = X_train_raw.assign(NUMBER_FAMILY_MEMBERS=NUMBER_FAMILY_M
EMBERS.values)
```

In [587]:
```
# Step 7. Drop the CNT_FAM_MEMBERS feature from the main dataframe
X_train_raw = X_train_raw.drop('CNT_FAM_MEMBERS', axis=1)

# Add the new NUMBER_FAMILY_MEMBERS feature to the list of categorical
# features that will need to be one-hot encoded. There are now 19 of t
hese features.
cat_feat_need_one_hot = cat_feat_need_one_hot + ['NUMBER_FAMILY_MEMBER
S']
```

In [588]:
```
# Step 8. Use the CREDIT_DAY_OVERDUE feature in bureau.csv to engineer
the binary
# categorical HAS_CREDIT_BUREAU_LOANS_OVERDUE feature. If CREDIT_DAY_O
VERDUE for a
# particular borrower ID (SK_ID_CURR) is greater than 0, then the valu
e of
# HAS_CREDIT_BUREAU_LOANS_OVERDUE will be 1. If CREDIT_DAY_OVERDUE for
a particular
# borrower ID is 0, then the value of HAS_CREDIT_BUREAU_LOANS_OVERDUE
will be 0.

# Filter the bureau data table for loans which are overdue (have a val
ue
# for CREDIT_DAY_OVERDUE that's greater than 0)
bureau_data_filtered_for_overdue = bureau_data[bureau_data['CREDIT_DAY
_OVERDUE'] > 0]

def build_feature_HAS_CREDIT_BUREAU_LOANS_OVERDUE(dataframe):
    """
    Use the CREDIT_DAY_OVERDUE feature in bureau.csv to engineer the b
inary
    categorical HAS_CREDIT_BUREAU_LOANS_OVERDUE feature. If CREDIT_DAY
_OVERDUE for a
    particular borrower ID (SK_ID_CURR) is greater than 0, then the va
lue of
    HAS_CREDIT_BUREAU_LOANS_OVERDUE will be 1. If CREDIT_DAY_OVERDUE f
or a particular
```

```
    borrower ID is 0, then the value of HAS_CREDIT_BUREAU_LOANS_OVERDU
E will be 0.

    Parameters:
        dataframe: Pandas dataframe containing a training or testing d
ataset

    Returns: The dataframe with HAS_CREDIT_BUREAU_LOANS_OVERDUE featur
e appended to it.
    """
    # Create a series called HAS_CREDIT_BUREAU_LOANS_OVERDUE and fill
it with zeros.
    # Its index is identical to that of the main dataframe. It will ev
entually be appended
    # to the main data frame as a column.
    HAS_CREDIT_BUREAU_LOANS_OVERDUE = pd.Series(data=0, index = datafr
ame['SK_ID_CURR'].index)

    # A list of all the borrowers IDs in the main dataframe
    main_data_table_borrower_IDs = dataframe['SK_ID_CURR'].values

    # For each loan in the bureau data table that is overdue
    # (has a value for CREDIT_DAY_OVERDUE that's greater than 0)
    for index, row in bureau_data_filtered_for_overdue.iterrows():
        # The borrower ID (SK_ID_CURR) that owns the overdue loan
        borrower_ID = row['SK_ID_CURR']
        # If the borrower ID owning the overdue loan is also
        # in the main data frame, then enter a value of 1 in
        # the series HAS_CREDIT_BUREAU_LOANS_OVERDUE at an index
        # that is identical to the index of the borrower ID
        # in the main data frame.
        if borrower_ID in main_data_table_borrower_IDs:
            # The index of the borrower's row in the main data table.
            borrower_index_main_data_table = dataframe.index[dataframe
['SK_ID_CURR'] == borrower_ID].tolist()[0]
            # Place a value of 1 at the index of the series HAS_CREDIT
_BUREAU_LOANS_OVERDUE
            # which corresponds to the index of the borrower's ID in t
he main data table.
            HAS_CREDIT_BUREAU_LOANS_OVERDUE.loc[borrower_index_main_da
ta_table] = 1
    # Append the newly engineered HAS_CREDIT_BUREAU_LOANS_OVERDUE feat
ure to the main dataframe.
    dataframe = dataframe.assign(HAS_CREDIT_BUREAU_LOANS_OVERDUE=HAS_C
REDIT_BUREAU_LOANS_OVERDUE.values)
    return dataframe

# Build the HAS_CREDIT_BUREAU_LOANS_OVERDUE feature
X_train_raw = build_feature_HAS_CREDIT_BUREAU_LOANS_OVERDUE(X_train_ra
w)

# Add the new HAS_CREDIT_BUREAU_LOANS_OVERDUE feature to the list of b
inary categorical
```

```
# features. There are now 34 of these features.
bin_cat_feat = bin_cat_feat + ['HAS_CREDIT_BUREAU_LOANS_OVERDUE']
```

In [589]:
```
# Find out what fraction of borrowers in the dataframe have overdue
# loans from other lenders besides Home Credit.
num_borrowers_maindataframe = X_train_raw.shape[0]
num_borrowers_maindataframe_with_other_overdue_loans = X_train_raw[X_t
rain_raw['HAS_CREDIT_BUREAU_LOANS_OVERDUE'] == 1].shape[0]
percent_borrowers_with_other_overdue_loans = round(num_borrowers_maind
ataframe_with_other_overdue_loans*100./num_borrowers_maindataframe, 2)
print('{} borrowers, or {}% of the training segment\'s {} borrowers ha
ve overdue loans from other lenders.'.format(num_borrowers_maindatafra
me_with_other_overdue_loans, percent_borrowers_with_other_overdue_loan
s, num_borrowers_maindataframe))
```

```
2693 borrowers, or 1.09% of the training segment's 246008 borrowers
have overdue loans from other lenders.
```

In [590]:
```
# Step 9. Use the DAYS_EMPLOYED feature to engineer a binary categoric
al feature called HAS_JOB.
# If the value of DAYS_EMPLOYED is 0 or less, then HAS_JOB will be 1.
Otherwise, HAS_JOB will
# be 0. This condition will apply to all borrowers who had a value of
365243 for DAYS_EMPLOYED,
# which I hypothesized can be best interpreted as meaning that a borro
wer does not have a job.
DAYS_EMPLOYED_train = X_train_raw['DAYS_EMPLOYED']
HAS_JOB = DAYS_EMPLOYED_train.map(lambda x: 1 if x <= 0 else 0)

# Append the newly engineered FAMILY_SIZE feature to the main datafram
e.
X_train_raw = X_train_raw.assign(HAS_JOB=HAS_JOB.values)
```

In [591]:
```
# Step 10. Drop the DAYS_EMPLOYED feature from the main dataframe
X_train_raw = X_train_raw.drop('DAYS_EMPLOYED', axis=1)

# Add the new HAS_JOB feature to the list of binary categorical featur
es.
# There are now 35 of these features.
bin_cat_feat = bin_cat_feat + ['HAS_JOB']
```

In [592]:
```
# Step 11. Translate the 2 non-normalized numerical features that have
skewed distributions
# and negative values: DAYS_REGISTRATION, and DAYS_LAST_PHONE_CHANGE

def translate_negative_valued_features(dataframe, feature_name_list):
    """
    Translate a dataset's continuous features containing several negat
ive
    values. The dataframe is modified such that all values of each fea
ture
    listed in the feature_name_list parameter become positive.

    Parameters:
        dataframe: Pandas dataframe containing the features
        feature_name_list: List of strings, containing the names
                           of each feature whose values will be
                           translated
    """
    for feature in feature_name_list:
        # The minimum, most-negative, value of the feature
        feature_min_value = dataframe[feature].min()
        # Translate each value of the feature in a positive direction,
        # of magnitude that's equal to the feature's most negative val
ue.
        dataframe[feature] = dataframe[feature].apply(lambda x: x - fe
ature_min_value)

# Translate the above two negatively-valued features to positive value
s
translate_negative_valued_features(X_train_raw, non_norm_feat_neg_valu
es_skewed)
```

In [593]:
```
# Step 12. Log-transform all 17 non-normalized numerical features that
have skewed distributions.
# These 17 features include the 2 that were translated to positive ran
ges in Step 11.

# Add the 2 features translated to positive ranges above in Step 11 to
# the list of non-normalized skewed features with positive values. Thi
s is
# the set of features that will be log-transformed
log_transform_feats = non_norm_feat_pos_values_skewed + non_norm_feat_
neg_values_skewed

X_train_raw[log_transform_feats] = X_train_raw[log_transform_feats].ap
ply(lambda x: np.log(x + 1))
```

In [594]:
```
# Step 13. Replace 'NaN' values for all numerical features with each f
eature's mean. Fit an imputer
# to each numerical feature containing at least one 'NaN' entry.

# Create a list of all the 67 numerical features in the main dataframe
. These include all
# 17 features that were log-transformed in Step 12, as well as the 4 n
ormal features that
# still need to be scaled, as well as the 46 normal features that don'
t need scaling.
numerical_features = log_transform_feats + norm_feat_need_scaling + no
rm_feat_not_need_scaling

# Create a list of all numerical features in the training set that hav
e at least one 'NaN' entry
numerical_features_with_nan = X_train_raw[numerical_features].columns[
X_train_raw[numerical_features].isna().any()].tolist()

# Create an imputer
imputer = Imputer()
# Fit the imputer to each numerical feature in the training set that h
as 'NaN' values,
# and replace each 'NaN' entry of each feature with that feature's mea
n.
X_train_raw[numerical_features_with_nan] = imputer.fit_transform(X_tra
in_raw[numerical_features_with_nan])
```

In [595]:
```
# Step 14. Remove the borrower ID column, SK_ID_CURR, from the main da
taframe
X_train_raw = X_train_raw.drop('SK_ID_CURR', axis=1)
```

In [596]:
```
# Verify that the main training dataframe has the expected number of c
olumns.
# Dataframe initially had 122 columns.
# 4 features have been added (HAS_CHILDREN, NUMBER_FAMILY_MEMBERS, HAS
_CREDIT_BUREAU_LOANS_OVERDUE, HAS_JOB).
# 5 columns have been removed (TARGET, SK_ID_CURR, DAYS_EMPLOYED, CNT_
CHILDREN, CNT_FAM_MEMBERS).
# Expected number of columns is thus 121.
print('The main training dataframe now has {} columns. Expected: 121.'
.format(X_train_raw.shape[1]))
```

```
The main training dataframe now has 121 columns. Expected: 121.
```

In [597]:
```
# Step 15. One-hot encode all 19 non-binary categorical features.
X_train_raw = pd.get_dummies(X_train_raw, columns=cat_feat_need_one_ho
t)

# Create a list that includes only the newly one-hot encoded features
# as well as all the categorical features that were already binary.
all_bin_cat_feat = X_train_raw.columns.tolist()
for column_name in X_train_raw[numerical_features].columns.tolist():
    all_bin_cat_feat.remove(column_name)
```

In [598]:
```
# Observe how many binary features now exist in the dataframe after on
e-hot encoding the
# 19 non-binary categorical features.
print('After one-hot encoding, there are now {} binary features in the
main training dataframe.'.format(len(all_bin_cat_feat)))
```

```
After one-hot encoding, there are now 184 binary features in the mai
n training dataframe.
```

In [599]:
```
# Observe how many total columns now exist in the dataframe after one-
hot encoding.
# It is expected there are 184 binary categorical features, and 67 sca
led numerical features
# for a total of 251 features.
print('After one-hot encoding, there are now {} columns in the main tr
aining dataframe. Expected: 251.'.format(X_train_raw.shape[1]))
```

```
After one-hot encoding, there are now 251 columns in the main traini
ng dataframe. Expected: 251.
```

In [600]:
```
# Step 16. Replace all 'NaN' values in all binary categorical features
with 0.

# Create a list of binary categorical features with at least one 'NaN'
entry
bin_cat_feat_with_nan = X_train_raw[all_bin_cat_feat].columns[X_train_
raw[all_bin_cat_feat].isna().any()].tolist()

# Replace each 'NaN' value in each of these binary features with 0
X_train_raw[bin_cat_feat_with_nan] = X_train_raw[bin_cat_feat_with_nan
].fillna(value=0)
```

In [601]:
```
# Step 17. Fit a min-max scaler to each of the 17 log-transformed numerical features, as well
# as to the 4 features DAYS_BIRTH, DAYS_ID_PUBLISH, HOUR_APPR_PROCESS_START, and the normalized
# feature REGION_POPULATION_RELATIVE. Each feature will be scaled to a range [0.0, 1.0].

# Build a list of all 21 features needing scaling. Add the list of features that
# were log-normalized above in Step 12 to the list of normally shaped features
# that need to be scaled to the range [0,1].
feats_to_scale = norm_feat_need_scaling + log_transform_feats

# Initialize a scaler with the default range of [0,1]
scaler = MinMaxScaler()

# Fit the scaler to each of the features of the train set that need to be scaled,
# then transform each of these features' values to the new scale.
X_train_raw[feats_to_scale] = scaler.fit_transform(X_train_raw[feats_to_scale])

# Rename the dataframe to indicate that its columns have been fully preprocessed.
X_train_final = X_train_raw
```

In [602]:
```
# Indicate that training set preprocessing is done.
# Verify that the training dataframe has the expected number of columns.
# It is expected there are 184 binary categorical features,
# and 67 numerical features for a total of 251 features.
print('Training set preprocessing complete. The final training dataframe now has {} columns. Expected: 251.'.format(X_train_final.shape[1]))
```

```
Training set preprocessing complete. The final training dataframe now has 251 columns. Expected: 251.
```

In [603]:
```
# Step 18. Build a data preprocessing pipeline to used for all testing sets.
# This pipeline will recreate all features that were engineered in the
# training set during the original data preprocessing phase.
# The pipeline will also apply the min-max scaling transforms
# originally fit on features in the training set to all datapoints in a
# testing set.

def adjust_columns_application_test_csv_table(testing_dataframe):
    """
    After it is one-hot encoded, application_test.csv data table will have one
    extra column, 'REGION_RATING_CLIENT_W_CITY_-1', that is not presen
```

```
t in the
    training dataframe. This column will be removed from the testing d
atatable
    in this case. Only 1 of the 48,744 rows in application_test.csv wi
ll have a
    value of 1 for this feature following one-hot encoding. I am not w
orried
    about this column's elimination from the testing dataframe affecti
ng predictions.

    Additionally, unlike the test validation set, which originally com
prised 20% of
    application_train.csv, application_test.csv will be missing the fo
llowing columns
    after it is one-hot encoded:

    'CODE_GENDER_XNA', 'NAME_INCOME_TYPE_Maternity leave', 'NAME_FAMIL
Y_STATUS_Unknown'

    In this case, we need to insert these columns into the testing dat
aframe, at
    the exact same indices they are located at in the fully preprocess
ed training
    dataframe.Each inserted column will be filled with all zeros. (If
each of these
    binary features are missing from the application_test.csv data tab
le, we can infer
    that each borrower in thatdata table obviously would have a 0 for
each feature were
    it present.)

    Parameters:
        testing_dataframe: Pandas dataframe containing the testing dat
aset
                           contained in the file application_test.csv

    Returns: a testing dataframe containing the exact same columns and
             column order as found in the training dataframe
    """

    # Identify any columns in the one-hot encoded testing_dataframe th
at
    # are not in X_train_raw. These columns will need to be removed fr
om the
    # testing_dataframe. (Expected that there will only be one such
    # column: 'REGION_RATING_CLIENT_W_CITY_-1')
    X_train_columns_list = X_train_raw.columns.tolist()
    testing_dataframe_columns_list = testing_dataframe.columns.tolist(
)
    for column_name in X_train_columns_list:
        if column_name in testing_dataframe_columns_list:
            testing_dataframe_columns_list.remove(column_name)
            columns_not_in_X_train_raw = testing_dataframe_columns_lis
```

```
t

    # Drop any column from the testing_dataframe that is not in the
    # training dataframe. Expected to only be the one column 'REGION_R
ATING_CLIENT_W_CITY_-1'
    for column in columns_not_in_X_train_raw:
        testing_dataframe = testing_dataframe.drop(column, axis=1)

    # Get the column indices of each of the features 'CODE_GENDER_XNA'
,
    #'NAME_INCOME_TYPE_Maternity leave', 'NAME_FAMILY_STATUS_Unknown'
from
    # the raw training dataframe (X_train_raw) prior to having having
PCA run on it.
    loc_code_gender_training_frame = X_train_raw.columns.get_loc('CODE
_GENDER_XNA')
    loc_name_income_type_maternity_leave_training_frame = X_train_raw.
columns.get_loc('NAME_INCOME_TYPE_Maternity leave')
    loc_name_family_status_unknown_training_frame = X_train_raw.column
s.get_loc('NAME_FAMILY_STATUS_Unknown')

    # Insert each column into the testing dataframe at the same index
it had
    # in the X_train_raw dataframe before PCA was run. Fill each colum
n with all 0s.
    # Order is important. 'CODE_GENDER_XNA' should be inserted first,
followed by
    # 'NAME_INCOME_TYPE_Maternity leave', and then finally 'NAME_FAMIL
Y_STATUS_Unknown'.
    testing_dataframe.insert(loc=loc_code_gender_training_frame, colum
n='CODE_GENDER_XNA', value=0)
    testing_dataframe.insert(loc=loc_name_income_type_maternity_leave_
training_frame, column='NAME_INCOME_TYPE_Maternity leave', value=0)
    testing_dataframe.insert(loc=loc_name_family_status_unknown_traini
ng_frame, column='NAME_FAMILY_STATUS_Unknown', value=0)
    return testing_dataframe

def test_set_preprocessing_pipeline(testing_dataframe):
    """
    Recreate all features that were engineered in the training set dur
ing
    the original data preprocessing phase. The pipeline will also appl
y
    an imputer to the test data table fill 'NaN' values. Binary featur
e's 'Nan'
    values will be filled with 0. The min-max scaler fit on features
    in the training set will be applied to the numerical features in t
he testing set.

    Parameters:
        testing_dataframe: Pandas dataframe containing a testing datas
et
```

```
    Returns: a fully preprocessed testing dataframe
    """

    # Create the HAS_CHILDREN feature.
    CNT_CHILDREN_test = testing_dataframe['CNT_CHILDREN']
    HAS_CHILDREN = CNT_CHILDREN_test.map(lambda x: 1 if x > 0 else 0)

    # Append the newly engineered HAS_CHILDREN feature to the main dat
aframe.
    testing_dataframe = testing_dataframe.assign(HAS_CHILDREN=HAS_CHIL
DREN.values)

    # Drop the CNT_CHILDREN column from the main dataframe
    testing_dataframe = testing_dataframe.drop('CNT_CHILDREN', axis=1)

    # Create the NUMBER_FAMILY_MEMBERS feature.
    CNT_FAM_MEMBERS_test = testing_dataframe['CNT_FAM_MEMBERS']
    NUMBER_FAMILY_MEMBERS = CNT_FAM_MEMBERS_test.map(lambda x: 'one' i
f x == 1 else ('two' if x == 2 else 'three_plus'))

    # Append the newly engineered FAMILY_SIZE feature to the main data
frame.
    testing_dataframe = testing_dataframe.assign(NUMBER_FAMILY_MEMBERS
=NUMBER_FAMILY_MEMBERS.values)

    # Drop the CNT_FAM_MEMBERS feature from the main dataframe
    testing_dataframe = testing_dataframe.drop('CNT_FAM_MEMBERS', axis
=1)

    # Build the HAS_CREDIT_BUREAU_LOANS_OVERDUE feature
    testing_dataframe = build_feature_HAS_CREDIT_BUREAU_LOANS_OVERDUE(
testing_dataframe)

    # Create the HAS_JOB feature
    DAYS_EMPLOYED_test = testing_dataframe['DAYS_EMPLOYED']
    HAS_JOB = DAYS_EMPLOYED_test.map(lambda x: 1 if x <= 0 else 0)

    # Append the newly engineered FAMILY_SIZE feature to the main data
frame.
    testing_dataframe = testing_dataframe.assign(HAS_JOB=HAS_JOB.value
s)

    # Drop the DAYS_EMPLOYED feature from the main dataframe
    testing_dataframe = testing_dataframe.drop('DAYS_EMPLOYED', axis=1
)

    # Translate the two negatively-valued features DAYS_REGISTRATION,
and
    # DAYS_LAST_PHONE_CHANGE to positive values
    translate_negative_valued_features(testing_dataframe, non_norm_fea
t_neg_values_skewed)

    # Log-transform all 17 non-normalized numerical features that have
```

```
    skewed distributions.
    testing_dataframe[log_transform_feats] = testing_dataframe[log_tra
nsform_feats].apply(lambda x: np.log(x + 1))

    # Create a list of all numerical features in the testing dataframe
that have at least one 'NaN' entry
    numerical_features_with_nan_testing = testing_dataframe[numerical_
features].columns[testing_dataframe[numerical_features].isna().any()].
tolist()

    # Use an imputer to replace 'NaN' values for all numerical feature
s with each feature's mean.
    testing_dataframe[numerical_features_with_nan_testing] = imputer.f
it_transform(testing_dataframe[numerical_features_with_nan_testing])

    # Remove the borrower ID column, SK_ID_CURR, from the main datafra
me
    testing_dataframe = testing_dataframe.drop('SK_ID_CURR', axis=1)

    # One-hot encode all 19 non-binary categorical features.
    testing_dataframe = pd.get_dummies(testing_dataframe, columns=cat_
feat_need_one_hot)

    # After one-hot encoding, the testing dataframe from application_t
est.csv will be
    # missing 2 columns that are in the training dataframe. It will al
so have an extra
    # column that was not in the training dataframe, giving it 249 tot
al columns.
    # If this is the case, we need to modify this testing dataframe so
that its columns
    # and column order is consistent with the training dataframe.
    if testing_dataframe.shape[1] == 249:
        testing_dataframe = adjust_columns_application_test_csv_table(
testing_dataframe)

    # Create a list of the binary categorical features with at least o
ne 'NaN' entry
    bin_cat_feat_with_nan_testing = testing_dataframe[all_bin_cat_feat
].columns[testing_dataframe[all_bin_cat_feat].isna().any()].tolist()

    # Replace each 'NaN' value in each of these binary features with 0
    testing_dataframe[bin_cat_feat_with_nan_testing] = testing_datafra
me[bin_cat_feat_with_nan_testing].fillna(value=0)

    # Transform each of the 21 features that need to be scaled to the
range [0,1] using
    # the min-max scaler fit on the training set.
    testing_dataframe[feats_to_scale] = scaler.transform(testing_dataf
rame[feats_to_scale])

    return testing_dataframe
```

In [604]:
```
# Step 19. Preprocess the test validation set.
X_test_final = test_set_preprocessing_pipeline(X_test_raw)
```

In [605]:
```
# Verify that the test validation dataframe has the expected number of
columns after
# preprocessing its data. It is expected there are 184 binary categori
cal features,
# and 67 numerical features for a total of 251 features.
print('Test validation set preprocessing complete. The final test vali
dation dataframe now has {} columns. Expected: 251.'.format(X_test_fin
al.shape[1]))
```

```
Test validation set preprocessing complete. The final test validatio
n dataframe now has 251 columns. Expected: 251.
```

V. Implementation

In [606]:
```
# Lists of of probability predictions and classifier names.
# To be used to plot ROC curves of each classifier's prediction
# probabilities.

y_score_list = []
clf_label_list = []
```

In [607]:
```
# Step 1. Create an ROC area-under-curve scorer.
def roc_auc_scorer(y_targets, y_score):
    """
    Calculates and returns the area under the ROC curve between
    the true target values and the probability estimates of the
    predicted values.
    """
    # Calculate the performance score between 'y_true' and 'y_predict'
    score = roc_auc_score(y_targets, y_score)

    # Return the score
    return score
```

In [608]:
```
# Step 2. Use the Gaussian Naive Bayes classifier to make predictions
on
# the test validation set. Calculate the area under ROC curve score of
# these predictions.

# Fit a Gaussian Naive Bayes classifier to the training dataframe.
clf_naive_bayes = GaussianNB()
clf_naive_bayes.fit(X_train_final, y_train)

# The Naive Bayes estimates of probability of the positive class (TARG
ET=1):
# the probability estimate of each borrower making at least one late l
oan payment.
naive_bayes_y_score = clf_naive_bayes.predict_proba(X_test_final)[:, 1
]

# The area under the ROC curve between the true target values and the
# probability estimates of the predicted values.
naive_bayes_roc_auc_score = roc_auc_scorer(y_test, naive_bayes_y_score
)

# Add the Naive Bayes classifier's scores to the results list.
y_score_list.append(naive_bayes_y_score)
clf_label_list.append('Naive Bayes All Features')

print('Naive Bayes (All Features) test validation set predictions\'
ROC AUC score: {}'.format(naive_bayes_roc_auc_score))
```

```
Naive Bayes (All Features) test validation set predictions' ROC AUC
score: 0.546645662333944
```

In [610]:
```
# Step 3. Create a method that performs GridSearchCV on a
# AdaBoost classifier learning algorithm to discover the highest
# scoring hyperparameter combination.
def find_best_hyperameters_adaboost(X_train, y_train):
    """
    Performs grid search over the 'n_estimators' parameter of an AdaBo
ost
    classifier trained on the input data [X_train, y_train].
    """

    # Create an AdaBoost classifier object
    clf = AdaBoostClassifier()

    # Create a dictionary for the parameter 'n_estimators' with differ
ent values
    # that will be attempted.
    params = {
        'learning_rate': [0.01, 0.1, 1.0],
        'n_estimators': [200, 250, 500,1000],
        'random_state': [42]
    }

    # Transform 'roc_auc_scorer' into a scoring function using 'make_s
corer'
    scoring_fnc = make_scorer(roc_auc_scorer)

    # Create a GridSearchCV object.
    grid = GridSearchCV(clf, params, scoring_fnc, cv=3)

    # Fit the grid search object to the data to compute the optimal mo
del
    grid = grid.fit(X_train, y_train)

    # Return the optimal model after fitting the data
    return grid.best_estimator_

# The AdaBoost classifier with hyperparameter values that scored the b
est
# in GridSearchCV.
clf_AdaBoost = find_best_hyperameters_adaboost(X_train_final, y_train)

print('Highest scoring AdaBoost classifier after running GridSearchCV:
{}'.format(clf_AdaBoost))
```

```
Highest scoring AdaBoost classifier after running GridSearchCV: AdaB
oostClassifier(algorithm='SAMME.R', base_estimator=None,
          learning_rate=1.0, n_estimators=1000, random_state=42)
```

In [611]:
```
# Step 4. Use the AdaBoost classifier to make predictions on
# the test validation set. Calculate the area under ROC curve score of
# these predictions.

# The AdaBoost classifier's estimates of probability of the positive c
lass (TARGET=1):
# the probability estimate of each borrower making at least one late l
oan payment.
adaBoost_y_score = clf_AdaBoost.predict_proba(X_test_final)[:, 1]

# The area under the ROC curve between the true target values and the
# probability estimates of the predicted values.
adaBoost_roc_auc_score = roc_auc_scorer(y_test, adaBoost_y_score)

# Add the AdaBoost classifier's scores to the results list.
y_score_list.append(adaBoost_y_score)
clf_label_list.append('AdaBoost All Features')

print('AdaBoost (All Features) test validation set predictions\' ROC A
UC score: {}'.format(adaBoost_roc_auc_score))
```

```
AdaBoost (All Features) test validation set predictions' ROC AUC sco
re: 0.7462758964509755
```

In [612]:
```
# Determine and display feature importances as determined by the AdaBo
ost classifier after
# it was fit on the full featureset.

feature_list = X_train_final.columns.values
feature_importance_list = clf_AdaBoost.feature_importances_
rows_list = []
for i in range(len(feature_importance_list)):
    if feature_importance_list[i] > 0:
        dictionary = {}
        dictionary['Feature Name'] = feature_list[i]
        dictionary['Importance'] = feature_importance_list[i]
        rows_list.append(dictionary)

adaBoost_feature_importances = pd.DataFrame(rows_list, columns=['Featu
re Name', 'Importance'])
adaBoost_feature_importances = adaBoost_feature_importances.sort_value
s('Importance',ascending=False)

display(adaBoost_feature_importances)
```

	Feature Name	Importance
14	EXT_SOURCE_1	0.075
15	EXT_SOURCE_2	0.073
1	AMT_CREDIT	0.066

2	AMT_ANNUITY	0.063
5	DAYS_BIRTH	0.061
16	EXT_SOURCE_3	0.052
3	AMT_GOODS_PRICE	0.046
7	DAYS_ID_PUBLISH	0.034
61	DAYS_LAST_PHONE_CHANGE	0.032
56	TOTALAREA_MODE	0.027
6	DAYS_REGISTRATION	0.027
0	AMT_INCOME_TOTAL	0.022
31	APARTMENTS_MODE	0.017
35	COMMONAREA_MODE	0.015
28	LIVINGAREA_AVG	0.013
40	LANDAREA_MODE	0.012
30	NONLIVINGAREA_AVG	0.012
32	BASEMENTAREA_MODE	0.011
48	COMMONAREA_MEDI	0.011
4	REGION_POPULATION_RELATIVE	0.011
42	LIVINGAREA_MODE	0.010
21	COMMONAREA_AVG	0.010
26	LANDAREA_AVG	0.009
8	OWN_CAR_AGE	0.009
27	LIVINGAPARTMENTS_AVG	0.009
54	LIVINGAREA_MEDI	0.009
11	HOUR_APPR_PROCESS_START	0.009
44	NONLIVINGAREA_MODE	0.008
45	APARTMENTS_MEDI	0.007
33	YEARS_BEGINEXPLUATATION_MODE	0.007
77	AMT_REQ_CREDIT_BUREAU_MON	0.007
18	BASEMENTAREA_AVG	0.006
25	FLOORSMIN_AVG	0.005
78	AMT_REQ_CREDIT_BUREAU_QRT	0.005

47	YEARS_BEGINEXPLUATATION_MEDI	0.005
46	BASEMENTAREA_MEDI	0.005
17	APARTMENTS_AVG	0.005
34	YEARS_BUILD_MODE	0.005
53	LIVINGAPARTMENTS_MEDI	0.005
20	YEARS_BUILD_AVG	0.005
29	NONLIVINGAPARTMENTS_AVG	0.005
38	FLOORSMAX_MODE	0.004
24	FLOORSMAX_AVG	0.004
41	LIVINGAPARTMENTS_MODE	0.004
75	AMT_REQ_CREDIT_BUREAU_DAY	0.004
19	YEARS_BEGINEXPLUATATION_AVG	0.004
79	AMT_REQ_CREDIT_BUREAU_YEAR	0.004
59	OBS_60_CNT_SOCIAL_CIRCLE	0.004
156	WALLSMATERIAL_MODE_Panel	0.003
83	CODE_GENDER_M	0.003
58	DEF_30_CNT_SOCIAL_CIRCLE	0.003
51	FLOORSMIN_MEDI	0.003
76	AMT_REQ_CREDIT_BUREAU_WEEK	0.003
52	LANDAREA_MEDI	0.003
88	NAME_INCOME_TYPE_State servant	0.003
50	ENTRANCES_MEDI	0.003
39	FLOORSMIN_MODE	0.003
37	ENTRANCES_MODE	0.003
92	NAME_EDUCATION_TYPE_Higher education	0.003
36	ELEVATORS_MODE	0.003
23	ENTRANCES_AVG	0.002
144	OCCUPATION_TYPE_Core staff	0.002
71	FLAG_DOCUMENT_15	0.002
9	FLAG_WORK_PHONE	0.002
22	ELEVATORS_AVG	0.002

95	NAME_FAMILY_STATUS_Married	0.002
130	ORGANIZATION_TYPE_Self-employed	0.002
60	DEF_60_CNT_SOCIAL_CIRCLE	0.002
49	ELEVATORS_MEDI	0.002
108	ORGANIZATION_TYPE_Construction	0.002
43	NONLIVINGAPARTMENTS_MODE	0.002
129	ORGANIZATION_TYPE_Security Ministries	0.001
131	ORGANIZATION_TYPE_Trade: type 2	0.001
132	ORGANIZATION_TYPE_Trade: type 3	0.001
157	FONDKAPREMONT_MODE_org spec account	0.001
128	ORGANIZATION_TYPE_School	0.001
127	ORGANIZATION_TYPE_Restaurant	0.001
126	ORGANIZATION_TYPE_Realtor	0.001
125	ORGANIZATION_TYPE_Postal	0.001
124	ORGANIZATION_TYPE_Police	0.001
123	ORGANIZATION_TYPE_Mobile	0.001
122	ORGANIZATION_TYPE_Military	0.001
121	ORGANIZATION_TYPE_Legal Services	0.001
120	ORGANIZATION_TYPE_Insurance	0.001
119	ORGANIZATION_TYPE_Industry: type 9	0.001
118	ORGANIZATION_TYPE_Industry: type 5	0.001
117	ORGANIZATION_TYPE_Industry: type 3	0.001
134	ORGANIZATION_TYPE_Trade: type 6	0.001
116	ORGANIZATION_TYPE_Industry: type 12	0.001
133	ORGANIZATION_TYPE_Trade: type 5	0.001
153	WALLSMATERIAL_MODE_Block	0.001
135	ORGANIZATION_TYPE_Trade: type 7	0.001
136	ORGANIZATION_TYPE_Transport: type 2	0.001
158	FONDKAPREMONT_MODE_reg oper spec account	0.001
155	WALLSMATERIAL_MODE_Others	0.001
114	ORGANIZATION_TYPE_Industry: type 1	0.001

154	WALLSMATERIAL_MODE_Monolithic	0.001
152	HOUSETYPE_MODE_terraced house	0.001
151	OCCUPATION_TYPE_Security staff	0.001
150	OCCUPATION_TYPE_Sales staff	0.001
149	OCCUPATION_TYPE_Medicine staff	0.001
148	OCCUPATION_TYPE_Low-skill Laborers	0.001
147	OCCUPATION_TYPE_Laborers	0.001
146	OCCUPATION_TYPE_High skill tech staff	0.001
145	OCCUPATION_TYPE_Drivers	0.001
143	OCCUPATION_TYPE_Cooking staff	0.001
142	OCCUPATION_TYPE_Cleaning staff	0.001
141	OCCUPATION_TYPE_Accountants	0.001
140	NAME_TYPE_SUITE_Spouse, partner	0.001
139	NAME_TYPE_SUITE_Children	0.001
138	ORGANIZATION_TYPE_Transport: type 4	0.001
137	ORGANIZATION_TYPE_Transport: type 3	0.001
115	ORGANIZATION_TYPE_Industry: type 10	0.001
80	HAS_CREDIT_BUREAU_LOANS_OVERDUE	0.001
113	ORGANIZATION_TYPE_Housing	0.001
67	FLAG_DOCUMENT_8	0.001
84	FLAG_OWN_CAR_N	0.001
82	NAME_CONTRACT_TYPE_Revolving loans	0.001
81	NAME_CONTRACT_TYPE_Cash loans	0.001
74	FLAG_DOCUMENT_18	0.001
73	FLAG_DOCUMENT_17	0.001
72	FLAG_DOCUMENT_16	0.001
70	FLAG_DOCUMENT_14	0.001
69	FLAG_DOCUMENT_13	0.001
68	FLAG_DOCUMENT_11	0.001
66	FLAG_DOCUMENT_6	0.001
112	ORGANIZATION_TYPE_Hotel	0.001

65	FLAG_DOCUMENT_5	0.001
64	FLAG_DOCUMENT_4	0.001
63	FLAG_DOCUMENT_3	0.001
62	FLAG_DOCUMENT_2	0.001
57	OBS_30_CNT_SOCIAL_CIRCLE	0.001
55	NONLIVINGAREA_MEDI	0.001
13	REG_CITY_NOT_LIVE_CITY	0.001
12	REG_REGION_NOT_LIVE_REGION	0.001
10	FLAG_PHONE	0.001
85	FLAG_OWN_REALTY_N	0.001
86	NAME_INCOME_TYPE_Commercial associate	0.001
87	NAME_INCOME_TYPE_Maternity leave	0.001
89	NAME_INCOME_TYPE_Student	0.001
111	ORGANIZATION_TYPE_Government	0.001
110	ORGANIZATION_TYPE_Emergency	0.001
109	ORGANIZATION_TYPE_Electricity	0.001
107	ORGANIZATION_TYPE_Business Entity Type 3	0.001
106	ORGANIZATION_TYPE_Bank	0.001
105	WEEKDAY_APPR_PROCESS_START_WEDNESDAY	0.001
104	WEEKDAY_APPR_PROCESS_START_SUNDAY	0.001
103	WEEKDAY_APPR_PROCESS_START_SATURDAY	0.001
102	WEEKDAY_APPR_PROCESS_START_MONDAY	0.001
101	REGION_RATING_CLIENT_W_CITY_3	0.001
100	REGION_RATING_CLIENT_W_CITY_1	0.001
99	NAME_HOUSING_TYPE_Rented apartment	0.001
98	NAME_HOUSING_TYPE_Office apartment	0.001
97	NAME_HOUSING_TYPE_Municipal apartment	0.001
96	NAME_FAMILY_STATUS_Widow	0.001
94	NAME_EDUCATION_TYPE_Secondary / secondary special	0.001
93	NAME_EDUCATION_TYPE_Incomplete higher	0.001
91	NAME_EDUCATION_TYPE_Academic degree	0.001

90	NAME_INCOME_TYPE_Unemployed	0.001
159	NUMBER_FAMILY_MEMBERS_two	0.001

160 rows × 2 columns

In [613]:
```
# Step 5. Try using a Logistic Regression classifier to make predictio
ns.

# Fit the classifier to the training data.
clf_logistic_regression = LogisticRegression(penalty='l2', random_stat
e=42, solver='liblinear')
clf_logistic_regression.fit(X_train_final, y_train)

# The logistical regression classifier's estimates of probability of t
he positive
# class (TARGET=1): the probability estimate of each borrower making a
t least one
# late loan payment.
logistic_regression_y_score = clf_logistic_regression.predict_proba(X_
test_final)[:, 1]

# The area under the ROC curve between the true target values and the
# probability estimates of the predicted values.
logistic_regression_roc_auc_score = roc_auc_scorer(y_test, logistic_re
gression_y_score)

# Add the Logistic Regression classifier's scores to the results list.
y_score_list.append(logistic_regression_y_score)
clf_label_list.append('Logistic Regression All Features')

print('Logistic Regression (All Features) test validation set predicti
ons\' ROC AUC score: {}'.format(logistic_regression_roc_auc_score))
```

```
Logistic Regression (All Features) test validation set predictions'
ROC AUC score: 0.7471756350178691
```

In [614]:
```
# Step 6. Try using a Multi-layer Perceptron classifier to make predic
tions.

# Fit the classifier to the training data.
clf_mlp = MLPClassifier(
    hidden_layer_sizes=100, activation='identity', solver='adam', alph
a=0.001, batch_size=200,
    learning_rate_init=0.001, random_state=42
)
clf_mlp.fit(X_train_final, y_train)

# The multi-layer perceptron classifier's estimates of probability of
the positive
# class (TARGET=1): the probability estimate of each borrower making a
t least one
# late loan payment.
mlp_y_score = clf_mlp.predict_proba(X_test_final)[:, 1]

# The area under the ROC curve between the true target values and the
# probability estimates of the predicted values.
mlp_roc_auc_score = roc_auc_scorer(y_test, mlp_y_score)

# Add the Multi-Layer Perceptron classifier's scores to the results li
st.
y_score_list.append(mlp_y_score)
clf_label_list.append('Multi-Layer Perceptron All Features')

print('Multi-layer Perceptron (All Features) test validation set predi
ctions\' ROC AUC score: {}'.format(mlp_roc_auc_score))
```

```
Multi-layer Perceptron (All Features) test validation set prediction
s' ROC AUC score: 0.7429017839300756
```

In [615]:
```
# Step 7. Try using a LightGBM classifier.

# Convert preprocessed training dataset into LightGBM dataset format
lightgbm_training = lgb.Dataset(X_train_final, label=y_train)

# Specify parameters
params = {}
params['learning_rate'] = 0.01
params['boosting_type'] = 'gbdt'
params['objective'] = 'binary'
params['metric'] = 'auc'
params['sub_feature'] = 0.3
params['num_leaves'] = 100
params['min_data_in_leaf'] = 500
params['max_depth'] = 10
params['max_bin'] = 64
#params['min_data_in_bin'] = 3
#params['lambda_l1'] = 0.01
params['lambda_l2'] = 0.01
#params['min_gain_to_split'] = 0.01
params['bagging_freq'] = 100
params['bagging_fraction'] = 0.9
#params['feature_fraction'] = 0.5

# Fit the LightGBM classifier to the training data
clf_lgb = lgb.train(params, lightgbm_training, 1500)

# Classifier's estimates of probability of the positive class (TARGET=
1): the
# probability estimate of each borrower making at least one late loan
payment.
lgb_y_score = clf_lgb.predict(X_test_final)

# The area under the ROC curve between the true target values and the
# probability estimates of the predicted values.
lgb_roc_auc_score = roc_auc_scorer(y_test, lgb_y_score)

# Add the LightGBM classifier's scores to the results list.
y_score_list.append(lgb_y_score)
clf_label_list.append('LightGBM All Features')

print('LightGBM (All Features) test validation set predictions\' ROC A
UC score: {}'.format(lgb_roc_auc_score))
```

```
LightGBM (All Features) test validation set predictions' ROC AUC sco
re: 0.7592132612569703
```

In [243]:
```
# Step 7. Build a prediction pipeline for the testing data table (appl
ication_test.csv) that
# saves prediction probabilities to a CSV file, which will then be sub
mitted on Kaggle.

# def testing_data_table_predictions_to_csv(clf, testing_data_table, i
```

```
sLightGBM):
#     """
#     A prediction pipeline that:
#     1. Preprocesses the 48,744 row testing data table
#     2. Uses a classifier to compute estimates of the probability of
the positive
#        class (TARGET=1) for each borrower: the probability estimate
of each borrower
#        making at least one late loan payment.
#     3. Saves a CSV file that contains probabilities of target labels
for each
#        borrower (SK_ID_CURR) in the testing data table.
#     4. isLightGBM: Boolean, a flag that indicates whether or not the
classifier is
#                    LightGBM. If True,

#     Parameters:
#         clf: A machine learning classifier object that has already b
een fit to
#                     the training data.
#         testing_data_table: Pandas dataframe containing the testing
dataset.
#     """

#     # Get a list of the borrower IDs (SK_ID_CURR column). The borrow
er ID must be
#     # placed in each row of CSV file that will be created.
#     borrower_IDs = testing_data_table['SK_ID_CURR']

#     # Preprocess the testing data table so that predictions can be m
ade on it.
#     X_test_final = test_set_preprocessing_pipeline(testing_data_tabl
e)
#     #print('application_test.csv testing set processing complete. Th
e processed dataframe now has {} columns. Expected: 251.'.format(X_tes
t_final.shape[1]))

#     # Classifier's estimates of probability of the positive class (T
ARGET=1): the
#     # probability estimate of each borrower making at least one late
loan payment.
#     # If classifier is LightGBM, the method for making predictions i
s merely 'predict'
#     # and the arrray containing these probabilities has slightly dif
ferent shape than
#     # those produced by the other classifiers.
#     if isLightGBM:
#         clf_y_score = clf.predict(X_test_final)
#     else:
#         clf_y_score = clf.predict_proba(X_test_final)[:, 1]

#     # Create the CSV file that will be saved
#     file_output = 'dellinger_kaggle_home_credit_submission2.csv'
```

```
#     # Write to the CSV file
#     with open(file_output, 'w') as csvfile:
#         writer = csv.writer(csvfile)
#         # Write the header row
#         writer.writerow(['SK_ID_CURR','TARGET'])
#         # Write a row for each borrower that contains the
#         # prediction probability of their label.
#         for index, value in borrower_IDs.iteritems():
#             writer.writerow([value, clf_y_score[index]])

# # To submit to Kaggle: the LightGBM Classifier's predictions on full
featureset.
# # Create predictions on the data in the testing data table (applicat
ion_test.csv)
# # using the LightGBM classifier fit above in Step 5. Also create a C
SV
# # file containing the prediction probabilities for each borrower ID
(SK_ID_CURR)
# # in the testing data table.
# testing_data_table_predictions_to_csv(clf_lgb, application_test_data
, True)
```

```
application_test.csv testing set processing complete. The processed
dataframe now has 251 columns. Expected: 251.
```

VI. Refinement

PCA Reduce Numeric Features

In [616]:
```
# Step 1. Now try training all classifiers on a featureset where PCA i
s used to compress the
# dimensions of the 67 numerical features.

# Load the main data tables
application_train_data = pd.read_csv("data/application_train.csv")
application_test_data = pd.read_csv("data/application_test.csv")

# Load the Bureau data table
bureau_data = pd.read_csv("data/bureau.csv")

# 1: Create lists of different feature types in the main data
# frame, based on how each type will need to be preprocessed.

# i. All 18 categorical features needing one-hot encoding.
#    Includes the 4 categorical features originally
#    mis-identified as having been normalized:
#    EMERGENCYSTATE_MODE, HOUSETYPE_MODE, WALLSMATERIAL_MODE,
#    FONDKAPREMONT_MODE
cat_feat_need_one_hot = [
```

```
    'NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR',
    'FLAG_OWN_REALTY', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE',
    'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'REGION_RATING_CLIENT',
    'REGION_RATING_CLIENT_W_CITY', 'WEEKDAY_APPR_PROCESS_START', 'ORGA
NIZATION_TYPE',
    'NAME_TYPE_SUITE', 'OCCUPATION_TYPE', 'EMERGENCYSTATE_MODE',
    'HOUSETYPE_MODE', 'WALLSMATERIAL_MODE', 'FONDKAPREMONT_MODE'
]

# ii. All 32 binary categorical features already one-hot encoded.
bin_cat_feat = [
    'FLAG_MOBIL', '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'
]

# iii. All 2 non-normalized numerical features with skewed distributio
ns
#    and negative values. These features will need to have their
#    distributions translated to positive ranges before being
#    log-transformed, and then later scaled to the range [0,1].
non_norm_feat_neg_values_skewed = [
    'DAYS_REGISTRATION', 'DAYS_LAST_PHONE_CHANGE'
]


# iv. All 15 non-normalized numerical features with skewed distributio
ns,
#    and only positive values. These features will need to be
#    log-transformed, and eventually scaled to the range [0,1].
non_norm_feat_pos_values_skewed = [
    'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY',
    'AMT_GOODS_PRICE', 'OBS_30_CNT_SOCIAL_CIRCLE', 'DEF_30_CNT_SOCIAL_
CIRCLE',
    'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIRCLE', 'AMT_REQ_C
REDIT_BUREAU_HOUR',
    'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK', 'AMT_RE
Q_CREDIT_BUREAU_MON',
    'AMT_REQ_CREDIT_BUREAU_QRT', 'AMT_REQ_CREDIT_BUREAU_YEAR', 'OWN_CA
R_AGE'
]

# v. All 4 numerical features with normal shapes but needing to be sca
```

```
led
#    to the range [0,1].
norm_feat_need_scaling = [
    'DAYS_BIRTH', 'DAYS_ID_PUBLISH', 'HOUR_APPR_PROCESS_START',
    'REGION_POPULATION_RELATIVE'
]

# vi. All 46 numerical features that have been normalized to the range
#   [0,1]. These features will need neither log-transformation, nor
#   any further scaling.
norm_feat_not_need_scaling = [
    'EXT_SOURCE_2', 'EXT_SOURCE_3', 'YEARS_BEGINEXPLUATATION_AVG',
    'YEARS_BEGINEXPLUATATION_MODE', 'YEARS_BEGINEXPLUATATION_MEDI', 'F
LOORSMAX_AVG',
    'FLOORSMAX_MODE', 'FLOORSMAX_MEDI', 'LIVINGAREA_AVG',
    'LIVINGAREA_MODE', 'LIVINGAREA_MEDI', 'ENTRANCES_AVG',
    'ENTRANCES_MODE', 'ENTRANCES_MEDI', 'APARTMENTS_AVG',
    'APARTMENTS_MODE', 'APARTMENTS_MEDI', 'ELEVATORS_AVG',
    'ELEVATORS_MODE', 'ELEVATORS_MEDI', 'NONLIVINGAREA_AVG',
    'NONLIVINGAREA_MODE', 'NONLIVINGAREA_MEDI', 'EXT_SOURCE_1',
    'BASEMENTAREA_AVG', 'BASEMENTAREA_MODE', 'BASEMENTAREA_MEDI',
    'LANDAREA_AVG', 'LANDAREA_MODE', 'LANDAREA_MEDI',
    'YEARS_BUILD_AVG', 'YEARS_BUILD_MODE', 'YEARS_BUILD_MEDI',
    'FLOORSMIN_AVG', 'FLOORSMIN_MODE', 'FLOORSMIN_MEDI',
    'LIVINGAPARTMENTS_AVG', 'LIVINGAPARTMENTS_MODE', 'LIVINGAPARTMENTS
_MEDI',
    'NONLIVINGAPARTMENTS_AVG', 'NONLIVINGAPARTMENTS_MODE', 'NONLIVINGA
PARTMENTS_MEDI',
    'COMMONAREA_AVG', 'COMMONAREA_MODE', 'COMMONAREA_MEDI',
    'TOTALAREA_MODE'
]

# vii. The remaining 3 features in the main data frame that will be
#    re-engineered and transformed into different features
feat_to_be_reengineered = [
    'CNT_CHILDREN', 'CNT_FAM_MEMBERS', 'DAYS_EMPLOYED'
]

# 2: Separate target data from training dataset.
targets = application_train_data['TARGET']
features_raw = application_train_data.drop('TARGET', axis = 1)

# 3: Use train_test_split from sklearn.cross_validation to
# create a test validation set that is 20% of the size of the total tr
aining set:
# Will allow me to compare performance of various learning algorithms
without
# overfitting to the training data.
X_train_raw, X_test_raw, y_train, y_test = train_test_split(features_r
aw,
                                                            targets,
                                                            test_size = 0.2,
                                                            random_state = 42)
```

```
# 4: Use the CNT_CHILDREN feature to engineer a binary
# categorical feature called HAS_CHILDREN. If value of CNT_CHILDREN is
# greater than 0, the value of HAS_CHILDREN will be 1. If value of CNT
_CHILDREN is
# 0, value of HAS_CHILDREN will be 0.
CNT_CHILDREN_train = X_train_raw['CNT_CHILDREN']
HAS_CHILDREN = CNT_CHILDREN_train.map(lambda x: 1 if x > 0 else 0)

# Append the newly engineered HAS_CHILDREN feature to the main datafra
me.
X_train_raw = X_train_raw.assign(HAS_CHILDREN=HAS_CHILDREN.values)

# 5: Drop the CNT_CHILDREN column from the main dataframe
X_train_raw = X_train_raw.drop('CNT_CHILDREN', axis=1)

# Add the new HAS_CHILDREN feature to the list of binary categorical
# features that are already one-hot encoded. There are now 33 such fea
tures.
bin_cat_feat = bin_cat_feat + ['HAS_CHILDREN']

# 6. Use the CNT_FAM_MEMBERS feature to engineer a categorical feature
called NUMBER_FAMILY_MEMBERS.
# If CNT_FAM_MEMBERS is 1.0, then the value of NUMBER_FAMILY_MEMBERS w
ill be 'one'. If CNT_FAM_MEMBERS is 2.0,
# then NUMBER_FAMILY_MEMBERS will be 'two'. If CNT_FAM_MEMBERS is 3.0
or greater, then NUMBER_FAMILY_MEMBERS will
# be 'three_plus'.
CNT_FAM_MEMBERS_train = X_train_raw['CNT_FAM_MEMBERS']
NUMBER_FAMILY_MEMBERS = CNT_FAM_MEMBERS_train.map(lambda x: 'one' if x
== 1 else ('two' if x == 2 else 'three_plus'))

# Append the newly engineered FAMILY_SIZE feature to the main datafram
e.
X_train_raw = X_train_raw.assign(NUMBER_FAMILY_MEMBERS=NUMBER_FAMILY_M
EMBERS.values)

# 7. Drop the CNT_FAM_MEMBERS feature from the main dataframe
X_train_raw = X_train_raw.drop('CNT_FAM_MEMBERS', axis=1)

# Add the new NUMBER_FAMILY_MEMBERS feature to the list of categorical
# features that will need to be one-hot encoded. There are now 19 of t
hese features.
cat_feat_need_one_hot = cat_feat_need_one_hot + ['NUMBER_FAMILY_MEMBER
S']

# 8. Use the CREDIT_DAY_OVERDUE feature in bureau.csv to engineer the
binary
# categorical HAS_CREDIT_BUREAU_LOANS_OVERDUE feature. If CREDIT_DAY_O
VERDUE for a
# particular borrower ID (SK_ID_CURR) is greater than 0, then the valu
e of
# HAS_CREDIT_BUREAU_LOANS_OVERDUE will be 1. If CREDIT_DAY_OVERDUE for
```

```
a particular
# borrower ID is 0, then the value of HAS_CREDIT_BUREAU_LOANS_OVERDUE
will be 0.

# Filter the bureau data table for loans which are overdue (have a val
ue
# for CREDIT_DAY_OVERDUE that's greater than 0)
bureau_data_filtered_for_overdue = bureau_data[bureau_data['CREDIT_DAY
_OVERDUE'] > 0]

def build_feature_HAS_CREDIT_BUREAU_LOANS_OVERDUE(dataframe):
    """
    Use the CREDIT_DAY_OVERDUE feature in bureau.csv to engineer the b
inary
    categorical HAS_CREDIT_BUREAU_LOANS_OVERDUE feature. If CREDIT_DAY
_OVERDUE for a
    particular borrower ID (SK_ID_CURR) is greater than 0, then the va
lue of
    HAS_CREDIT_BUREAU_LOANS_OVERDUE will be 1. If CREDIT_DAY_OVERDUE f
or a particular
    borrower ID is 0, then the value of HAS_CREDIT_BUREAU_LOANS_OVERDU
E will be 0.

    Parameters:
        dataframe: Pandas dataframe containing a training or testing d
ataset

    Returns: The dataframe with HAS_CREDIT_BUREAU_LOANS_OVERDUE featur
e appended to it.
    """
    # Create a series called HAS_CREDIT_BUREAU_LOANS_OVERDUE and fill
it with zeros.
    # Its index is identical to that of the main dataframe. It will ev
entually be appended
    # to the main data frame as a column.
    HAS_CREDIT_BUREAU_LOANS_OVERDUE = pd.Series(data=0, index = datafr
ame['SK_ID_CURR'].index)

    # A list of all the borrowers IDs in the main dataframe
    main_data_table_borrower_IDs = dataframe['SK_ID_CURR'].values

    # For each loan in the bureau data table that is overdue
    # (has a value for CREDIT_DAY_OVERDUE that's greater than 0)
    for index, row in bureau_data_filtered_for_overdue.iterrows():
        # The borrower ID (SK_ID_CURR) that owns the overdue loan
        borrower_ID = row['SK_ID_CURR']
        # If the borrower ID owning the overdue loan is also
        # in the main data frame, then enter a value of 1 in
        # the series HAS_CREDIT_BUREAU_LOANS_OVERDUE at an index
        # that is identical to the index of the borrower ID
        # in the main data frame.
        if borrower_ID in main_data_table_borrower_IDs:
            # The index of the borrower's row in the main data table.
```

```
            borrower_index_main_data_table = dataframe.index[dataframe
['SK_ID_CURR'] == borrower_ID].tolist()[0]
            # Place a value of 1 at the index of the series HAS_CREDIT
_BUREAU_LOANS_OVERDUE
            # which corresponds to the index of the borrower's ID in t
he main data table.
            HAS_CREDIT_BUREAU_LOANS_OVERDUE.loc[borrower_index_main_da
ta_table] = 1
    # Append the newly engineered HAS_CREDIT_BUREAU_LOANS_OVERDUE feat
ure to the main dataframe.
    dataframe = dataframe.assign(HAS_CREDIT_BUREAU_LOANS_OVERDUE=HAS_C
REDIT_BUREAU_LOANS_OVERDUE.values)
    return dataframe

# Build the HAS_CREDIT_BUREAU_LOANS_OVERDUE feature
X_train_raw = build_feature_HAS_CREDIT_BUREAU_LOANS_OVERDUE(X_train_ra
w)

# Add the new HAS_CREDIT_BUREAU_LOANS_OVERDUE feature to the list of b
inary categorical
# features. There are now 34 of these features.
bin_cat_feat = bin_cat_feat + ['HAS_CREDIT_BUREAU_LOANS_OVERDUE']

# 9. Use the DAYS_EMPLOYED feature to engineer a binary categorical fe
ature called HAS_JOB.
# If the value of DAYS_EMPLOYED is 0 or less, then HAS_JOB will be 1.
Otherwise, HAS_JOB will
# be 0. This condition will apply to all borrowers who had a value of
365243 for DAYS_EMPLOYED,
# which I hypothesized can be best interpreted as meaning that a borro
wer does not have a job.
DAYS_EMPLOYED_train = X_train_raw['DAYS_EMPLOYED']
HAS_JOB = DAYS_EMPLOYED_train.map(lambda x: 1 if x <= 0 else 0)

# Append the newly engineered FAMILY_SIZE feature to the main datafram
e.
X_train_raw = X_train_raw.assign(HAS_JOB=HAS_JOB.values)

# 10. Drop the DAYS_EMPLOYED feature from the main dataframe
X_train_raw = X_train_raw.drop('DAYS_EMPLOYED', axis=1)

# Add the new HAS_JOB feature to the list of binary categorical featur
es.
# There are now 35 of these features.
bin_cat_feat = bin_cat_feat + ['HAS_JOB']

# 11. Translate the 2 non-normalized numerical features that have skew
ed distributions
# and negative values: DAYS_REGISTRATION, and DAYS_LAST_PHONE_CHANGE

def translate_negative_valued_features(dataframe, feature_name_list):
    """
    Translate a dataset's continuous features containing several negat
```

```
ive
    values. The dataframe is modified such that all values of each fea
ture
    listed in the feature_name_list parameter become positive.

    Parameters:
        dataframe: Pandas dataframe containing the features
        feature_name_list: List of strings, containing the names
                           of each feature whose values will be
                           translated
    """
    for feature in feature_name_list:
        # The minimum, most-negative, value of the feature
        feature_min_value = dataframe[feature].min()
        # Translate each value of the feature in a positive direction,
        # of magnitude that's equal to the feature's most negative val
ue.
        dataframe[feature] = dataframe[feature].apply(lambda x: x - fe
ature_min_value)

# Translate the above two negatively-valued features to positive value
s
translate_negative_valued_features(X_train_raw, non_norm_feat_neg_valu
es_skewed)

# 12. Log-transform all 17 non-normalized numerical features that have
skewed distributions.
# These 17 features include the 2 that were translated to positive ran
ges in Step 11.

# Add the 2 features translated to positive ranges above in Step 11 to
# the list of non-normalized skewed features with positive values. Thi
s is
# the set of features that will be log-transformed
log_transform_feats = non_norm_feat_pos_values_skewed + non_norm_feat_
neg_values_skewed

X_train_raw[log_transform_feats] = X_train_raw[log_transform_feats].ap
ply(lambda x: np.log(x + 1))

# 13. Replace 'NaN' values for all numerical features with each featur
e's mean. Fit an imputer
# to each numerical feature containing at least one 'NaN' entry.

# Create a list of all the 67 numerical features in the main dataframe
. These include all
# 17 features that were log-transformed in Step 12, as well as the 4 n
ormal features that
# still need to be scaled, as well as the 46 normal features that don'
t need scaling.
numerical_features = log_transform_feats + norm_feat_need_scaling + no
rm_feat_not_need_scaling
```

```
# Create a list of all numerical features in the training set that hav
e at least one 'NaN' entry
numerical_features_with_nan = X_train_raw[numerical_features].columns[
X_train_raw[numerical_features].isna().any()].tolist()

# Create an imputer
imputer = Imputer()
# Fit the imputer to each numerical feature in the training set that h
as 'NaN' values,
# and replace each 'NaN' entry of each feature with that feature's mea
n.
X_train_raw[numerical_features_with_nan] = imputer.fit_transform(X_tra
in_raw[numerical_features_with_nan])

# 14. Remove the borrower ID column, SK_ID_CURR, from the main datafra
me
X_train_raw = X_train_raw.drop('SK_ID_CURR', axis=1)

# 15. One-hot encode all 19 non-binary categorical features.
X_train_raw = pd.get_dummies(X_train_raw, columns=cat_feat_need_one_ho
t)

# Create a list that includes only the newly one-hot encoded features
# as well as all the categorical features that were already binary.
all_bin_cat_feat = X_train_raw.columns.tolist()
for column_name in X_train_raw[numerical_features].columns.tolist():
    all_bin_cat_feat.remove(column_name)

# 16. Replace all 'NaN' values in all binary categorical features with
0.

# Create a list of binary categorical features with at least one 'NaN'
entry
bin_cat_feat_with_nan = X_train_raw[all_bin_cat_feat].columns[X_train_
raw[all_bin_cat_feat].isna().any()].tolist()

# Replace each 'NaN' value in each of these binary features with 0
X_train_raw[bin_cat_feat_with_nan] = X_train_raw[bin_cat_feat_with_nan
].fillna(value=0)

# 17. Fit a min-max scaler to each of the 17 log-transformed numerical
features, as well
# as to the 4 features DAYS_BIRTH, DAYS_ID_PUBLISH, HOUR_APPR_PROCESS_
START, and the normalized
# feature REGION_POPULATION_RELATIVE. Each feature will be scaled to a
range [0.0, 1.0].

# Build a list of all 21 features needing scaling. Add the list of fea
tures that
# were log-normalized above in Step 12 to the list of normally shaped
features
# that need to be scaled to the range [0,1].
feats_to_scale = norm_feat_need_scaling + log_transform_feats
```

```
# Initialize a scaler with the default range of [0,1]
scaler = MinMaxScaler()

# Fit the scaler to each of the features of the train set that need to
be scaled,
# then transform each of these features' values to the new scale.
X_train_raw[feats_to_scale] = scaler.fit_transform(X_train_raw[feats_t
o_scale])

# Rename the dataframe to indicate that its columns have been fully pr
eprocessed.
X_train_processed = X_train_raw
```

In [617]:
```
# 18. Fit PCA on all numerical features and observe how many
# components explain approximately 90% of the variance in the data.
pca = PCA(n_components = 17)
pca.fit(X_train_processed[numerical_features])

# Number of components used for pca
n_components = len(pca.explained_variance_ratio_)

# The total percent explained variance of all components used in PCA
percent_explained_var_all_n_components = round(sum(pca.explained_varia
nce_ratio_)*100, 2)

print('Explained variance ratios for each component:')
print(pca.explained_variance_ratio_)
print('\r')
print('{}% of variance of numerical features explained by {}
components.'.format(percent_explained_var_all_n_components, n_componen
ts))
```

```
Explained variance ratios for each component:
[ 0.1712049   0.12231898  0.09857175  0.06724392  0.05962164  0.0568
2911
  0.05083881  0.04976384  0.04510004  0.04230571  0.02981382  0.0251
2118
  0.01950228  0.01827228  0.01581516  0.0155643   0.01255201]

90.04% of variance of numerical features explained by 17 components.
```

In [618]:
```
# Display the head of the dataframe created to store the values of the
reduced numerical
# feature dimensions output by PCA.
display(X_train_reduced_numerical_features.head())
```

	PCA Dimension 1	PCA Dimension 2	PCA Dimension 3	PCA Dimension 4	PCA Dimension 5	PCA Dimension 6	I Dimen
123473	0.038490	-0.418327	0.132259	0.032978	0.035820	-0.006637	-0.2471
10118	-0.334238	0.278945	0.174923	0.010723	-0.197557	-0.111055	0.0533
64716	0.096831	-0.553266	-0.020605	-0.092755	-0.095095	0.246250	0.1128
234940	0.139965	0.268606	-0.222063	0.105934	-0.107170	0.132780	-0.1098
236051	0.016087	-0.177554	0.032054	0.062543	-0.231354	0.287856	0.2189

5 rows × 17 columns

In [619]:
```
# 19. Use PCA to reduce the dimension space of the numerical features
# to the optimal number of principle components discovered above.

reduced_numerical_data = pca.transform(X_train_processed[numerical_fea
tures])

# Create a DataFrame for the reduced data
X_train_reduced_numerical_features = pd.DataFrame(reduced_numerical_da
ta, index=X_train_processed.index, columns = [
    'PCA Dimension 1', 'PCA Dimension 2', 'PCA Dimension 3',
    'PCA Dimension 4', 'PCA Dimension 5', 'PCA Dimension 6',
    'PCA Dimension 7', 'PCA Dimension 8', 'PCA Dimension 9',
    'PCA Dimension 10', 'PCA Dimension 11', 'PCA Dimension 12',
    'PCA Dimension 13', 'PCA Dimension 14', 'PCA Dimension 15',
    'PCA Dimension 16', 'PCA Dimension 17'
])

# 20. Drop all 67 numerical features from the original preprocessed
# dataframe, so that it only contains the 184 binary categorical featu
res.
# Append the dataframe containing the reduced numerical features back
to
# this original dataframe.

# Drop the 67 original numerical features from the dataframe
X_train_processed = X_train_processed.drop(numerical_features, axis=1)

# Merge the dataframe with the dataframe containing the  17 reduced
# numerical features
X_train_final = pd.merge(left=X_train_processed, right=X_train_reduced
_numerical_features, left_index=True, right_index=True)
```

```
# 21. Build a data preprocessing pipeline to used for all testing sets
.
# This pipeline will recreate all features that were engineered in the
# training set during the original data preprocessing phase.
# The pipeline will also apply the imputer, min-max, and PCA transform
s
# originally fit on features in the training set to all datapoints in
a
# testing set.

def adjust_columns_application_test_csv_table(testing_dataframe):
    """
    After it is one-hot encoded, application_test.csv data table will
have one
    extra column, 'REGION_RATING_CLIENT_W_CITY_-1', that is not presen
t in the
    training dataframe. This column will be removed from the testing d
atatable
    in this case. Only 1 of the 48,744 rows in application_test.csv wi
ll have a
    value of 1 for this feature following one-hot encoding. I am not w
orried
    about this column's elimination from the testing dataframe affecti
ng predictions.

    Additionally, unlike the test validation set, which originally com
prised 20% of
    application_train.csv, application_test.csv will be missing the fo
llowing columns
    after it is one-hot encoded:

    'CODE_GENDER_XNA', 'NAME_INCOME_TYPE_Maternity leave', 'NAME_FAMIL
Y_STATUS_Unknown'

    In this case, we need to insert these columns into the testing dat
aframe, at
    the exact same indices they are located at in the fully preprocess
ed training
    dataframe.Each inserted column will be filled with all zeros. (If
each of these
    binary features are missing from the application_test.csv data tab
le, we can infer
    that each borrower in thatdata table obviously would have a 0 for
each feature were
    it present.)

    Parameters:
        testing_dataframe: Pandas dataframe containing the testing dat
aset
                           contained in the file application_test.csv

    Returns: a testing dataframe containing the exact same columns and
```

```
            column order as found in the training dataframe
    """

    # Identify any columns in the one-hot encoded testing_dataframe th
at
    # are not in X_train_raw. These columns will need to be removed fr
om the
    # testing_dataframe. (Expected that there will only be one such
    # column: 'REGION_RATING_CLIENT_W_CITY_-1')
    X_train_columns_list = X_train_raw.columns.tolist()
    testing_dataframe_columns_list = testing_dataframe.columns.tolist(
)
    for column_name in X_train_columns_list:
        if column_name in testing_dataframe_columns_list:
            testing_dataframe_columns_list.remove(column_name)
            columns_not_in_X_train_raw = testing_dataframe_columns_lis
t

    # Drop any column from the testing_dataframe that is not in the
    # training dataframe. Expected to only be the one column 'REGION_R
ATING_CLIENT_W_CITY_-1'
    for column in columns_not_in_X_train_raw:
        testing_dataframe = testing_dataframe.drop(column, axis=1)

    # Get the column indices of each of the features 'CODE_GENDER_XNA'
,
    #'NAME_INCOME_TYPE_Maternity leave', 'NAME_FAMILY_STATUS_Unknown'
from
    # the raw training dataframe (X_train_raw) prior to having having
PCA run on it.
    loc_code_gender_training_frame = X_train_raw.columns.get_loc('CODE
_GENDER_XNA')
    loc_name_income_type_maternity_leave_training_frame = X_train_raw.
columns.get_loc('NAME_INCOME_TYPE_Maternity leave')
    loc_name_family_status_unknown_training_frame = X_train_raw.column
s.get_loc('NAME_FAMILY_STATUS_Unknown')

    # Insert each column into the testing dataframe at the same index
it had
    # in the X_train_raw dataframe before PCA was run. Fill each colum
n with all 0s.
    # Order is important. 'CODE_GENDER_XNA' should be inserted first,
followed by
    # 'NAME_INCOME_TYPE_Maternity leave', and then finally 'NAME_FAMIL
Y_STATUS_Unknown'.
    testing_dataframe.insert(loc=loc_code_gender_training_frame, colum
n='CODE_GENDER_XNA', value=0)
    testing_dataframe.insert(loc=loc_name_income_type_maternity_leave_
training_frame, column='NAME_INCOME_TYPE_Maternity leave', value=0)
    testing_dataframe.insert(loc=loc_name_family_status_unknown_traini
ng_frame, column='NAME_FAMILY_STATUS_Unknown', value=0)
    return testing_dataframe
```

```
def test_set_preprocessing_pipeline(testing_dataframe):
    """
    Recreate all features that were engineered in the training set dur
ing
    the original data preprocessing phase. Missing numerical 'NaN' val
ues
    will be filled with an imputer. Missing binary categorical feature
'NaN'
    values will be replaced with 0. The pipeline will also apply
    the min-max and PCA transforms originally fit on features
    in the training set to numerical features in the testing set.

    Parameters:
        testing_dataframe: Pandas dataframe containing a testing datas
et

    Returns: a fully preprocessed testing dataframe
    """

    # Create the HAS_CHILDREN feature.
    CNT_CHILDREN_test = testing_dataframe['CNT_CHILDREN']
    HAS_CHILDREN = CNT_CHILDREN_test.map(lambda x: 1 if x > 0 else 0)

    # Append the newly engineered HAS_CHILDREN feature to the main dat
aframe.
    testing_dataframe = testing_dataframe.assign(HAS_CHILDREN=HAS_CHIL
DREN.values)

    # Drop the CNT_CHILDREN column from the main dataframe
    testing_dataframe = testing_dataframe.drop('CNT_CHILDREN', axis=1)

    # Create the NUMBER_FAMILY_MEMBERS feature.
    CNT_FAM_MEMBERS_test = testing_dataframe['CNT_FAM_MEMBERS']
    NUMBER_FAMILY_MEMBERS = CNT_FAM_MEMBERS_test.map(lambda x: 'one' i
f x == 1 else ('two' if x == 2 else 'three_plus'))

    # Append the newly engineered FAMILY_SIZE feature to the main data
frame.
    testing_dataframe = testing_dataframe.assign(NUMBER_FAMILY_MEMBERS
=NUMBER_FAMILY_MEMBERS.values)

    # Drop the CNT_FAM_MEMBERS feature from the main dataframe
    testing_dataframe = testing_dataframe.drop('CNT_FAM_MEMBERS', axis
=1)

    # Build the HAS_CREDIT_BUREAU_LOANS_OVERDUE feature
    testing_dataframe = build_feature_HAS_CREDIT_BUREAU_LOANS_OVERDUE(
testing_dataframe)

    # Create the HAS_JOB feature
    DAYS_EMPLOYED_test = testing_dataframe['DAYS_EMPLOYED']
    HAS_JOB = DAYS_EMPLOYED_test.map(lambda x: 1 if x <= 0 else 0)
```

```
    # Append the newly engineered FAMILY_SIZE feature to the main data
frame.
    testing_dataframe = testing_dataframe.assign(HAS_JOB=HAS_JOB.value
s)

    # Drop the DAYS_EMPLOYED feature from the main dataframe
    testing_dataframe = testing_dataframe.drop('DAYS_EMPLOYED', axis=1
)

    # Translate the two negatively-valued features DAYS_REGISTRATION,
and
    # DAYS_LAST_PHONE_CHANGE to positive values
    translate_negative_valued_features(testing_dataframe, non_norm_fea
t_neg_values_skewed)

    # Log-transform all 17 non-normalized numerical features that have
skewed distributions.
    testing_dataframe[log_transform_feats] = testing_dataframe[log_tra
nsform_feats].apply(lambda x: np.log(x + 1))

    # Create a list of all numerical features in the testing dataframe
that have at least one 'NaN' entry
    numerical_features_with_nan_testing = testing_dataframe[numerical_
features].columns[testing_dataframe[numerical_features].isna().any()].
tolist()

    # Use an imputer to replace 'NaN' values for all numerical feature
s with each feature's mean.
    testing_dataframe[numerical_features_with_nan_testing] = imputer.f
it_transform(testing_dataframe[numerical_features_with_nan_testing])

    # Remove the borrower ID column, SK_ID_CURR, from the main datafra
me
    testing_dataframe = testing_dataframe.drop('SK_ID_CURR', axis=1)

    # One-hot encode all 19 non-binary categorical features.
    testing_dataframe = pd.get_dummies(testing_dataframe, columns=cat_
feat_need_one_hot)

    # After one-hot encoding, the testing dataframe from application_t
est.csv will be
    # missing 2 columns that are in the training dataframe. It will al
so have an extra
    # column that was not in the training dataframe, giving it 249 tot
al columns.
    # If this is the case, we need to modify this testing dataframe so
that its columns
    # and column order is consistent with the training dataframe.
    if testing_dataframe.shape[1] == 249:
        testing_dataframe = adjust_columns_application_test_csv_table(
testing_dataframe)

    # Create a list of the binary categorical features with at least o
```

```
ne 'NaN' entry
    bin_cat_feat_with_nan_testing = testing_dataframe[all_bin_cat_feat
].columns[testing_dataframe[all_bin_cat_feat].isna().any()].tolist()

    # Replace each 'NaN' value in each of these binary features with 0
    testing_dataframe[bin_cat_feat_with_nan_testing] = testing_datafra
me[bin_cat_feat_with_nan_testing].fillna(value=0)

    # Transform each of the 21 features that need to be scaled to the
range [0,1] using
    # the min-max scaler fit on the training set.
    testing_dataframe[feats_to_scale] = scaler.transform(testing_dataf
rame[feats_to_scale])

    # Use the PCA algorithm fit on the training set to reduce the dime
nsion space of
    # the numerical features in the testing set.
    reduced_numerical_data_testing = pca.transform(testing_dataframe[n
umerical_features])

    # Create a DataFrame for the reduced data
    testing_dataframe_reduced_numerical_features = pd.DataFrame(reduce
d_numerical_data_testing, index=testing_dataframe.index, columns = [
        'PCA Dimension 1', 'PCA Dimension 2', 'PCA Dimension 3',
        'PCA Dimension 4', 'PCA Dimension 5', 'PCA Dimension 6',
        'PCA Dimension 7', 'PCA Dimension 8', 'PCA Dimension 9',
        'PCA Dimension 10', 'PCA Dimension 11', 'PCA Dimension 12',
        'PCA Dimension 13', 'PCA Dimension 14', 'PCA Dimension 15',
        'PCA Dimension 16', 'PCA Dimension 17'
    ])

    # Drop the 67 original numerical features from the dataframe
    testing_dataframe = testing_dataframe.drop(numerical_features, axi
s=1)

    # Merge the dataframe with the dataframe containing the 17 reduced
    # numerical features.
    testing_dataframe = pd.merge(left=testing_dataframe, right=testing
_dataframe_reduced_numerical_features, left_index=True, right_index=Tr
ue)
    # Return the fully preprocessed testing dataframe
    return testing_dataframe

# 22. Preprocess the test validation set.
X_test_final = test_set_preprocessing_pipeline(X_test_raw)

# Verify that both the training and test validation dataframes have th
e expected number of columns after
# preprocessing its data and transforming their features using the PCA
algorithm that was fit on the training
# data's numerical features. It is expected there are 184 binary categ
orical features, and 17 reduced numerical
# features for a total of 201 features.
```

```
print('Training set preprocessing complete. The final training datafra
me now has {} columns. Expected: 201.'.format(X_train_final.shape[1]))
print('Test validation set preprocessing complete. The final test vali
dation dataframe now has {} columns. Expected: 201.'.format(X_test_fin
al.shape[1]))
```

```
Training set preprocessing complete. The final training dataframe no
w has 201 columns. Expected: 201.
Test validation set preprocessing complete. The final test validatio
n dataframe now has 201 columns. Expected: 201.
```

In [620]:
```
# Train the classifiers and compute prediction probabilities:

# 1. Use a Gaussian Naive Bayes classifier to make predictions on
# the test validation set. Calculate the area under ROC curve score of
# these predictions.

# Fit a Gaussian Naive Bayes classifier to the training dataframe.
clf_naive_bayes = GaussianNB()
clf_naive_bayes.fit(X_train_final, y_train)

# The Naive Bayes estimates of probability of the positive class (TARG
ET=1):
# the probability estimate of each borrower making at least one late l
oan payment.
naive_bayes_PCA_y_score = clf_naive_bayes.predict_proba(X_test_final)[
:, 1]

# The area under the ROC curve between the true target values and the
# probability estimates of the predicted values.
naive_bayes_PCA_roc_auc_score = roc_auc_scorer(y_test, naive_bayes_PCA
_y_score)

# Add the Naive Bayes classifier's scores to the results list.
y_score_list.append(naive_bayes_PCA_y_score)
clf_label_list.append('Naive Bayes PCA')

print('Naive Bayes (PCA) test validation set predictions\' ROC AUC sco
re: {}'.format(naive_bayes_PCA_roc_auc_score))

# 2. Use an AdaBoost classifier to make predictions on the test valida
tion set.
# Calculate the area under ROC curve score of these predictions.

# Fit the AdaBoost classifier, using the parameter for 'n_estimators'
discovered
# when running GridSearchCV on AdaBoost above for the full featureset.
clf_AdaBoost = AdaBoostClassifier(learning_rate=1.0, n_estimators=1000
, random_state=42)
clf_AdaBoost.fit(X_train_final, y_train)

# The AdaBoost classifier's estimates of probability of the positive c
lass (TARGET=1):
```

```
# the probability estimate of each borrower making at least one late l
oan payment.
adaBoost_PCA_y_score = clf_AdaBoost.predict_proba(X_test_final)[:, 1]

# The area under the ROC curve between the true target values and the
# probability estimates of the predicted values.
adaBoost_PCA_roc_auc_score = roc_auc_scorer(y_test, adaBoost_PCA_y_sco
re)

# Add the AdaBoost classifier's scores to the results list.
y_score_list.append(adaBoost_PCA_y_score)
clf_label_list.append('AdaBoost PCA')

print('AdaBoost (PCA) test validation set predictions\' ROC AUC score:
{}'.format(adaBoost_PCA_roc_auc_score))

# 3. Try using a Logistic Regression classifier to make predictions.

# Fit the classifier to the training data.
clf_logistic_regression = LogisticRegression(penalty='l1', random_stat
e=42, solver='liblinear', max_iter=100)
clf_logistic_regression.fit(X_train_final, y_train)

# The logistical regression classifier's estimates of probability of t
he positive
# class (TARGET=1): the probability estimate of each borrower making a
t least one
# late loan payment.
logistic_regression_PCA_y_score = clf_logistic_regression.predict_prob
a(X_test_final)[:, 1]

# The area under the ROC curve between the true target values and the
# probability estimates of the predicted values.
logistic_regression_PCA_roc_auc_score = roc_auc_scorer(y_test, logisti
c_regression_PCA_y_score)

# Add the Logistic Regression classifier's scores to the results list.
y_score_list.append(logistic_regression_PCA_y_score)
clf_label_list.append('Logistic Regression PCA')

print('Logistic Regression (PCA) test validation set predictions\' ROC
AUC score: {}'.format(logistic_regression_PCA_roc_auc_score))

# 4. Try using a Multi-layer Perceptron classifier to make predictions
.

# Fit the classifier to the training data.
clf_mlp = MLPClassifier(
    hidden_layer_sizes=100, activation='identity', solver='adam', alph
a=0.001, batch_size=200,
    learning_rate_init=0.001, random_state=42, warm_start=False
)
clf_mlp.fit(X_train_final, y_train)
```

```
# The multi-layer perceptron classifier's estimates of probability of
the positive
# class (TARGET=1): the probability estimate of each borrower making a
t least one
# late loan payment.
mlp_PCA_y_score = clf_mlp.predict_proba(X_test_final)[:, 1]

# The area under the ROC curve between the true target values and the
# probability estimates of the predicted values.
mlp_PCA_roc_auc_score = roc_auc_scorer(y_test, mlp_PCA_y_score)

# Add the Multi-Layer Perceptron classifier's scores to the results li
st.
y_score_list.append(mlp_PCA_y_score)
clf_label_list.append('Multi-Layer Perceptron PCA')

print('Multi-layer Perceptron (PCA) test validation set predictions\'
ROC AUC score: {}'.format(mlp_PCA_roc_auc_score))

# 5. Try using a LightGBM classifier.

# Convert preprocessed training dataset into LightGBM dataset format
lightgbm_training = lgb.Dataset(X_train_final, label=y_train)

# Specify parameters
params = {}
params['learning_rate'] = 0.01
params['boosting_type'] = 'gbdt'
params['objective'] = 'binary'
params['metric'] = 'auc'
params['sub_feature'] = 0.3
params['num_leaves'] = 100
params['min_data_in_leaf'] = 500
params['max_depth'] = 10
params['max_bin'] = 64
#params['min_data_in_bin'] = 3
#params['lambda_l1'] = 0.01
params['lambda_l2'] = 0.01
#params['min_gain_to_split'] = 0.01
params['bagging_freq'] = 100
params['bagging_fraction'] = 0.9
#params['feature_fraction'] = 0.5

# Fit the LightGBM classifier to the training data
clf_lgb = lgb.train(params, lightgbm_training, 1500)

# Classifier's estimates of probability of the positive class (TARGET=
1): the
# probability estimate of each borrower making at least one late loan
payment.
lgb_PCA_y_score = clf_lgb.predict(X_test_final)
```

```
# The area under the ROC curve between the true target values and the
# probability estimates of the predicted values.
lgb_PCA_roc_auc_score = roc_auc_scorer(y_test, lgb_PCA_y_score)

# Add the LightGBM classifier's scores to the results list.
y_score_list.append(lgb_PCA_y_score)
clf_label_list.append('LightGBM PCA')

print('LightGBM (PCA) test validation set predictions\' ROC AUC score:
{}'.format(lgb_PCA_roc_auc_score))
```

```
Naive Bayes (PCA) test validation set predictions' ROC AUC score: 0.
5452255614331999
AdaBoost (PCA) test validation set predictions' ROC AUC score: 0.741
5669749755673
Logistic Regression (PCA) test validation set predictions' ROC AUC s
core: 0.743963963781135
Multi-layer Perceptron (PCA) test validation set predictions' ROC AU
C score: 0.7439527449175637
LightGBM (PCA) test validation set predictions' ROC AUC score: 0.748
3887050110797
```

SelectKBest Feature Selection

In [633]:
```
# Step 2. Try training all classifiers on a featureset where SelectKBe
st feature selection
# has been used to narrow down the full featureset's 251 features to t
he best performing features.

# Preprocess the dataset similar to how it was done above. However, th
is time use SelectKBest
# to only use a portion of the 251 features that exist after one-hot e
ncoding.

# Load the main data tables
application_train_data = pd.read_csv("data/application_train.csv")
application_test_data = pd.read_csv("data/application_test.csv")

# Load the Bureau data table
bureau_data = pd.read_csv("data/bureau.csv")

# 1: Create lists of different feature types in the main data
# frame, based on how each type will need to be preprocessed.

# i. All 18 categorical features needing one-hot encoding.
#    Includes the 4 categorical features originally
#    mis-identified as having been normalized:
#    EMERGENCYSTATE_MODE, HOUSETYPE_MODE, WALLSMATERIAL_MODE,
#    FONDKAPREMONT_MODE
cat_feat_need_one_hot = [
    'NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR',
```

```
    'FLAG_OWN_REALTY', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE',
    'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'REGION_RATING_CLIENT',
    'REGION_RATING_CLIENT_W_CITY', 'WEEKDAY_APPR_PROCESS_START', 'ORGA
NIZATION_TYPE',
    'NAME_TYPE_SUITE', 'OCCUPATION_TYPE', 'EMERGENCYSTATE_MODE',
    'HOUSETYPE_MODE', 'WALLSMATERIAL_MODE', 'FONDKAPREMONT_MODE'
]

# ii. All 32 binary categorical features already one-hot encoded.
bin_cat_feat = [
    'FLAG_MOBIL', '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'
]

# iii. All 2 non-normalized numerical features with skewed distributio
ns
#    and negative values. These features will need to have their
#    distributions translated to positive ranges before being
#    log-transformed, and then later scaled to the range [0,1].
non_norm_feat_neg_values_skewed = [
    'DAYS_REGISTRATION', 'DAYS_LAST_PHONE_CHANGE'
]


# iv. All 15 non-normalized numerical features with skewed distributio
ns,
#    and only positive values. These features will need to be
#    log-transformed, and eventually scaled to the range [0,1].
non_norm_feat_pos_values_skewed = [
    'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY',
    'AMT_GOODS_PRICE', 'OBS_30_CNT_SOCIAL_CIRCLE', 'DEF_30_CNT_SOCIAL_
CIRCLE',
    'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIRCLE', 'AMT_REQ_C
REDIT_BUREAU_HOUR',
    'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK', 'AMT_RE
Q_CREDIT_BUREAU_MON',
    'AMT_REQ_CREDIT_BUREAU_QRT', 'AMT_REQ_CREDIT_BUREAU_YEAR', 'OWN_CA
R_AGE'
]

# v. All 4 numerical features with normal shapes but needing to be sca
led
```

```
#    to the range [0,1].
norm_feat_need_scaling = [
    'DAYS_BIRTH', 'DAYS_ID_PUBLISH', 'HOUR_APPR_PROCESS_START',
    'REGION_POPULATION_RELATIVE'
]

# vi. All 46 numerical features that have been normalized to the range
#   [0,1]. These features will need neither log-transformation, nor
#   any further scaling.
norm_feat_not_need_scaling = [
    'EXT_SOURCE_2', 'EXT_SOURCE_3', 'YEARS_BEGINEXPLUATATION_AVG',
    'YEARS_BEGINEXPLUATATION_MODE', 'YEARS_BEGINEXPLUATATION_MEDI', 'F
LOORSMAX_AVG',
    'FLOORSMAX_MODE', 'FLOORSMAX_MEDI', 'LIVINGAREA_AVG',
    'LIVINGAREA_MODE', 'LIVINGAREA_MEDI', 'ENTRANCES_AVG',
    'ENTRANCES_MODE', 'ENTRANCES_MEDI', 'APARTMENTS_AVG',
    'APARTMENTS_MODE', 'APARTMENTS_MEDI', 'ELEVATORS_AVG',
    'ELEVATORS_MODE', 'ELEVATORS_MEDI', 'NONLIVINGAREA_AVG',
    'NONLIVINGAREA_MODE', 'NONLIVINGAREA_MEDI', 'EXT_SOURCE_1',
    'BASEMENTAREA_AVG', 'BASEMENTAREA_MODE', 'BASEMENTAREA_MEDI',
    'LANDAREA_AVG', 'LANDAREA_MODE', 'LANDAREA_MEDI',
    'YEARS_BUILD_AVG', 'YEARS_BUILD_MODE', 'YEARS_BUILD_MEDI',
    'FLOORSMIN_AVG', 'FLOORSMIN_MODE', 'FLOORSMIN_MEDI',
    'LIVINGAPARTMENTS_AVG', 'LIVINGAPARTMENTS_MODE', 'LIVINGAPARTMENTS
_MEDI',
    'NONLIVINGAPARTMENTS_AVG', 'NONLIVINGAPARTMENTS_MODE', 'NONLIVINGA
PARTMENTS_MEDI',
    'COMMONAREA_AVG', 'COMMONAREA_MODE', 'COMMONAREA_MEDI',
    'TOTALAREA_MODE'
]

# vii. The remaining 3 features in the main data frame that will be
#    re-engineered and transformed into different features
feat_to_be_reengineered = [
    'CNT_CHILDREN', 'CNT_FAM_MEMBERS', 'DAYS_EMPLOYED'
]

# 2: Separate target data from training dataset.
targets = application_train_data['TARGET']
features_raw = application_train_data.drop('TARGET', axis = 1)

# 3: Use train_test_split from sklearn.cross_validation to
# create a test validation set that is 20% of the size of the total tr
aining set:
# Will allow me to compare performance of various learning algorithms
without
# overfitting to the training data.
X_train_raw, X_test_raw, y_train, y_test = train_test_split(features_r
aw,
                                                            targets,
                                                            test_size = 0.2,
                                                            random_state = 42)
```

```
# 4: Use the CNT_CHILDREN feature to engineer a binary
# categorical feature called HAS_CHILDREN. If value of CNT_CHILDREN is
# greater than 0, the value of HAS_CHILDREN will be 1. If value of CNT
_CHILDREN is
# 0, value of HAS_CHILDREN will be 0.
CNT_CHILDREN_train = X_train_raw['CNT_CHILDREN']
HAS_CHILDREN = CNT_CHILDREN_train.map(lambda x: 1 if x > 0 else 0)

# Append the newly engineered HAS_CHILDREN feature to the main datafra
me.
X_train_raw = X_train_raw.assign(HAS_CHILDREN=HAS_CHILDREN.values)

# 5: Drop the CNT_CHILDREN column from the main dataframe
X_train_raw = X_train_raw.drop('CNT_CHILDREN', axis=1)

# Add the new HAS_CHILDREN feature to the list of binary categorical
# features that are already one-hot encoded. There are now 33 such fea
tures.
bin_cat_feat = bin_cat_feat + ['HAS_CHILDREN']

# 6. Use the CNT_FAM_MEMBERS feature to engineer a categorical feature
called NUMBER_FAMILY_MEMBERS.
# If CNT_FAM_MEMBERS is 1.0, then the value of NUMBER_FAMILY_MEMBERS w
ill be 'one'. If CNT_FAM_MEMBERS is 2.0,
# then NUMBER_FAMILY_MEMBERS will be 'two'. If CNT_FAM_MEMBERS is 3.0
or greater, then NUMBER_FAMILY_MEMBERS will
# be 'three_plus'.
CNT_FAM_MEMBERS_train = X_train_raw['CNT_FAM_MEMBERS']
NUMBER_FAMILY_MEMBERS = CNT_FAM_MEMBERS_train.map(lambda x: 'one' if x
== 1 else ('two' if x == 2 else 'three_plus'))

# Append the newly engineered FAMILY_SIZE feature to the main datafram
e.
X_train_raw = X_train_raw.assign(NUMBER_FAMILY_MEMBERS=NUMBER_FAMILY_M
EMBERS.values)

# 7. Drop the CNT_FAM_MEMBERS feature from the main dataframe
X_train_raw = X_train_raw.drop('CNT_FAM_MEMBERS', axis=1)

# Add the new NUMBER_FAMILY_MEMBERS feature to the list of categorical
# features that will need to be one-hot encoded. There are now 19 of t
hese features.
cat_feat_need_one_hot = cat_feat_need_one_hot + ['NUMBER_FAMILY_MEMBER
S']

# 8. Use the CREDIT_DAY_OVERDUE feature in bureau.csv to engineer the
binary
# categorical HAS_CREDIT_BUREAU_LOANS_OVERDUE feature. If CREDIT_DAY_O
VERDUE for a
# particular borrower ID (SK_ID_CURR) is greater than 0, then the valu
e of
# HAS_CREDIT_BUREAU_LOANS_OVERDUE will be 1. If CREDIT_DAY_OVERDUE for
a particular
```

```
# borrower ID is 0, then the value of HAS_CREDIT_BUREAU_LOANS_OVERDUE
will be 0.

# Filter the bureau data table for loans which are overdue (have a val
ue
# for CREDIT_DAY_OVERDUE that's greater than 0)
bureau_data_filtered_for_overdue = bureau_data[bureau_data['CREDIT_DAY
_OVERDUE'] > 0]

def build_feature_HAS_CREDIT_BUREAU_LOANS_OVERDUE(dataframe):
    """
    Use the CREDIT_DAY_OVERDUE feature in bureau.csv to engineer the b
inary
    categorical HAS_CREDIT_BUREAU_LOANS_OVERDUE feature. If CREDIT_DAY
_OVERDUE for a
    particular borrower ID (SK_ID_CURR) is greater than 0, then the va
lue of
    HAS_CREDIT_BUREAU_LOANS_OVERDUE will be 1. If CREDIT_DAY_OVERDUE f
or a particular
    borrower ID is 0, then the value of HAS_CREDIT_BUREAU_LOANS_OVERDU
E will be 0.

    Parameters:
        dataframe: Pandas dataframe containing a training or testing d
ataset

    Returns: The dataframe with HAS_CREDIT_BUREAU_LOANS_OVERDUE featur
e appended to it.
    """
    # Create a series called HAS_CREDIT_BUREAU_LOANS_OVERDUE and fill
it with zeros.
    # Its index is identical to that of the main dataframe. It will ev
entually be appended
    # to the main data frame as a column.
    HAS_CREDIT_BUREAU_LOANS_OVERDUE = pd.Series(data=0, index = datafr
ame['SK_ID_CURR'].index)

    # A list of all the borrowers IDs in the main dataframe
    main_data_table_borrower_IDs = dataframe['SK_ID_CURR'].values

    # For each loan in the bureau data table that is overdue
    # (has a value for CREDIT_DAY_OVERDUE that's greater than 0)
    for index, row in bureau_data_filtered_for_overdue.iterrows():
        # The borrower ID (SK_ID_CURR) that owns the overdue loan
        borrower_ID = row['SK_ID_CURR']
        # If the borrower ID owning the overdue loan is also
        # in the main data frame, then enter a value of 1 in
        # the series HAS_CREDIT_BUREAU_LOANS_OVERDUE at an index
        # that is identical to the index of the borrower ID
        # in the main data frame.
        if borrower_ID in main_data_table_borrower_IDs:
            # The index of the borrower's row in the main data table.
            borrower_index_main_data_table = dataframe.index[dataframe
```

```
['SK_ID_CURR'] == borrower_ID].tolist()[0]
            # Place a value of 1 at the index of the series HAS_CREDIT
_BUREAU_LOANS_OVERDUE
            # which corresponds to the index of the borrower's ID in t
he main data table.
            HAS_CREDIT_BUREAU_LOANS_OVERDUE.loc[borrower_index_main_da
ta_table] = 1
    # Append the newly engineered HAS_CREDIT_BUREAU_LOANS_OVERDUE feat
ure to the main dataframe.
    dataframe = dataframe.assign(HAS_CREDIT_BUREAU_LOANS_OVERDUE=HAS_C
REDIT_BUREAU_LOANS_OVERDUE.values)
    return dataframe

# Build the HAS_CREDIT_BUREAU_LOANS_OVERDUE feature
X_train_raw = build_feature_HAS_CREDIT_BUREAU_LOANS_OVERDUE(X_train_ra
w)

# Add the new HAS_CREDIT_BUREAU_LOANS_OVERDUE feature to the list of b
inary categorical
# features. There are now 34 of these features.
bin_cat_feat = bin_cat_feat + ['HAS_CREDIT_BUREAU_LOANS_OVERDUE']

# 9. Use the DAYS_EMPLOYED feature to engineer a binary categorical fe
ature called HAS_JOB.
# If the value of DAYS_EMPLOYED is 0 or less, then HAS_JOB will be 1.
Otherwise, HAS_JOB will
# be 0. This condition will apply to all borrowers who had a value of
365243 for DAYS_EMPLOYED,
# which I hypothesized can be best interpreted as meaning that a borro
wer does not have a job.
DAYS_EMPLOYED_train = X_train_raw['DAYS_EMPLOYED']
HAS_JOB = DAYS_EMPLOYED_train.map(lambda x: 1 if x <= 0 else 0)

# Append the newly engineered FAMILY_SIZE feature to the main datafram
e.
X_train_raw = X_train_raw.assign(HAS_JOB=HAS_JOB.values)

# 10. Drop the DAYS_EMPLOYED feature from the main dataframe
X_train_raw = X_train_raw.drop('DAYS_EMPLOYED', axis=1)

# Add the new HAS_JOB feature to the list of binary categorical featur
es.
# There are now 35 of these features.
bin_cat_feat = bin_cat_feat + ['HAS_JOB']

# 11. Translate the 2 non-normalized numerical features that have skew
ed distributions
# and negative values: DAYS_REGISTRATION, and DAYS_LAST_PHONE_CHANGE

def translate_negative_valued_features(dataframe, feature_name_list):
    """
    Translate a dataset's continuous features containing several negat
ive
```

```
    values. The dataframe is modified such that all values of each fea
ture
    listed in the feature_name_list parameter become positive.

    Parameters:
        dataframe: Pandas dataframe containing the features
        feature_name_list: List of strings, containing the names
                           of each feature whose values will be
                           translated
    """
    for feature in feature_name_list:
        # The minimum, most-negative, value of the feature
        feature_min_value = dataframe[feature].min()
        # Translate each value of the feature in a positive direction,
        # of magnitude that's equal to the feature's most negative val
ue.
        dataframe[feature] = dataframe[feature].apply(lambda x: x - fe
ature_min_value)

# Translate the above two negatively-valued features to positive value
s
translate_negative_valued_features(X_train_raw, non_norm_feat_neg_valu
es_skewed)

# 12. Log-transform all 17 non-normalized numerical features that have
skewed distributions.
# These 17 features include the 2 that were translated to positive ran
ges in Step 11.

# Add the 2 features translated to positive ranges above in Step 11 to
# the list of non-normalized skewed features with positive values. Thi
s is
# the set of features that will be log-transformed
log_transform_feats = non_norm_feat_pos_values_skewed + non_norm_feat_
neg_values_skewed

X_train_raw[log_transform_feats] = X_train_raw[log_transform_feats].ap
ply(lambda x: np.log(x + 1))

# 13. Replace 'NaN' values for all numerical features with each featur
e's mean. Fit an imputer
# to each numerical feature containing at least one 'NaN' entry.

# Create a list of all the 67 numerical features in the main dataframe
. These include all
# 17 features that were log-transformed in Step 12, as well as the 4 n
ormal features that
# still need to be scaled, as well as the 46 normal features that don'
t need scaling.
numerical_features = log_transform_feats + norm_feat_need_scaling + no
rm_feat_not_need_scaling

# Create a list of all numerical features in the training set that hav
```

```
e at least one 'NaN' entry
numerical_features_with_nan = X_train_raw[numerical_features].columns[
X_train_raw[numerical_features].isna().any()].tolist()

# Create an imputer
imputer = Imputer()
# Fit the imputer to each numerical feature in the training set that h
as 'NaN' values,
# and replace each 'NaN' entry of each feature with that feature's mea
n.
X_train_raw[numerical_features_with_nan] = imputer.fit_transform(X_tra
in_raw[numerical_features_with_nan])

# 14. Remove the borrower ID column, SK_ID_CURR, from the main datafra
me
X_train_raw = X_train_raw.drop('SK_ID_CURR', axis=1)

# 15. One-hot encode all 19 non-binary categorical features.
X_train_raw = pd.get_dummies(X_train_raw, columns=cat_feat_need_one_ho
t)

# Create a list that includes only the newly one-hot encoded features
# as well as all the categorical features that were already binary.
all_bin_cat_feat = X_train_raw.columns.tolist()
for column_name in X_train_raw[numerical_features].columns.tolist():
    all_bin_cat_feat.remove(column_name)

# 16. Replace all 'NaN' values in all binary categorical features with
0.

# Create a list of binary categorical features with at least one 'NaN'
entry
bin_cat_feat_with_nan = X_train_raw[all_bin_cat_feat].columns[X_train_
raw[all_bin_cat_feat].isna().any()].tolist()

# Replace each 'NaN' value in each of these binary features with 0
X_train_raw[bin_cat_feat_with_nan] = X_train_raw[bin_cat_feat_with_nan
].fillna(value=0)

# 17. Fit a min-max scaler to each of the 17 log-transformed numerical
features, as well
# as to the 4 features DAYS_BIRTH, DAYS_ID_PUBLISH, HOUR_APPR_PROCESS_
START, and the normalized
# feature REGION_POPULATION_RELATIVE. Each feature will be scaled to a
range [0.0, 1.0].

# Build a list of all 21 features needing scaling. Add the list of fea
tures that
# were log-normalized above in Step 12 to the list of normally shaped
features
# that need to be scaled to the range [0,1].
feats_to_scale = norm_feat_need_scaling + log_transform_feats
```

```
# Initialize a scaler with the default range of [0,1]
scaler = MinMaxScaler()

# Fit the scaler to each of the features of the train set that need to
be scaled,
# then transform each of these features' values to the new scale.
X_train_raw[feats_to_scale] = scaler.fit_transform(X_train_raw[feats_t
o_scale])

# Rename the dataframe to indicate that its columns have been fully pr
eprocessed.
X_train_processed = X_train_raw
```

In [634]:
```
# 18. Fit selectKBest to the fully processed full feature set.

selectK = SelectKBest(score_func=f_classif, k=10)
selectK.fit(X_train_processed, y_train)

# Rank each feature by its score in SelectKBest and display
feature_list = X_train_processed.columns.values
feature_importance_list = selectK.scores_
rows_list = []
for i in range(len(feature_importance_list)):
    dictionary = {}
    dictionary['Feature Name'] = feature_list[i]
    dictionary['Score'] = feature_importance_list[i]
    rows_list.append(dictionary)

selectKBest_feature_scores = pd.DataFrame(rows_list, columns=['Feature
Name', 'Score'])
selectKBest_feature_scores_ranked = selectKBest_feature_scores.sort_va
lues('Score',ascending=False)

# Ranked scores of each feature using SelectKBest with a
# f_classif scorer.
display(selectKBest_feature_scores_ranked)
```

	Feature Name	Score
23	EXT_SOURCE_2	6351.644583
24	EXT_SOURCE_3	6328.437711
22	EXT_SOURCE_1	2418.609627
5	DAYS_BIRTH	1491.612647
120	NAME_EDUCATION_TYPE_Higher education	770.750915
118	NAME_INCOME_TYPE_Working	762.909544
105	CODE_GENDER_M	741.230537
104	CODE_GENDER_F	741.011078

7	DAYS_ID_PUBLISH	672.617697
72	DAYS_LAST_PHONE_CHANGE	664.547926
20	REG_CITY_NOT_WORK_CITY	641.963446
141	REGION_RATING_CLIENT_W_CITY_3	617.904943
123	NAME_EDUCATION_TYPE_Secondary / secondary special	587.421576
138	REGION_RATING_CLIENT_3	559.755872
114	NAME_INCOME_TYPE_Pensioner	519.156107
101	HAS_JOB	514.506126
206	ORGANIZATION_TYPE_XNA	514.506126
10	FLAG_EMP_PHONE	514.247189
74	FLAG_DOCUMENT_3	489.176233
19	REG_CITY_NOT_LIVE_CITY	487.748885
222	OCCUPATION_TYPE_Laborers	451.649992
232	EMERGENCYSTATE_MODE_No	430.986786
6	DAYS_REGISTRATION	426.892776
139	REGION_RATING_CLIENT_W_CITY_1	419.102104
136	REGION_RATING_CLIENT_1	391.975347
234	HOUSETYPE_MODE_block of flats	391.084471
4	REGION_POPULATION_RELATIVE	335.592000
69	DEF_30_CNT_SOCIAL_CIRCLE	276.154409
21	LIVE_CITY_NOT_WORK_CITY	271.268415
71	DEF_60_CNT_SOCIAL_CIRCLE	262.072203
241	WALLSMATERIAL_MODE_Panel	261.178193
103	NAME_CONTRACT_TYPE_Revolving loans	247.628491
102	NAME_CONTRACT_TYPE_Cash loans	247.628491
32	FLOORSMAX_AVG	211.969217
135	NAME_HOUSING_TYPE_With parents	210.523971
191	ORGANIZATION_TYPE_Self-employed	209.063468
11	FLAG_WORK_PHONE	208.123695
60	FLOORSMAX_MEDI	207.749142
46	FLOORSMAX_MODE	203.088318

218	OCCUPATION_TYPE_Drivers	200.625618
77	FLAG_DOCUMENT_6	197.942808
100	HAS_CREDIT_BUREAU_LOANS_OVERDUE	197.189819
223	OCCUPATION_TYPE_Low-skill Laborers	186.988198
131	NAME_HOUSING_TYPE_House / apartment	180.158575
127	NAME_FAMILY_STATUS_Single / not married	170.537185
125	NAME_FAMILY_STATUS_Married	151.100606
154	ORGANIZATION_TYPE_Business Entity Type 3	149.376673
8	OWN_CAR_AGE	147.047893
13	FLAG_PHONE	136.936022
3	AMT_GOODS_PRICE	132.906700
108	FLAG_OWN_CAR_Y	132.261176
107	FLAG_OWN_CAR_N	132.261176
115	NAME_INCOME_TYPE_State servant	127.037951
15	HOUR_APPR_PROCESS_START	126.874585
246	FONDKAPREMONT_MODE_reg oper account	125.770412
124	NAME_FAMILY_STATUS_Civil marriage	124.294047
36	LIVINGAREA_AVG	119.925762
64	LIVINGAREA_MEDI	119.437499
67	TOTALAREA_MODE	119.138985
30	ELEVATORS_AVG	117.493051
58	ELEVATORS_MEDI	114.950479
214	OCCUPATION_TYPE_Accountants	110.879705
99	HAS_CHILDREN	106.044789
50	LIVINGAREA_MODE	105.500373
44	ELEVATORS_MODE	102.355106
156	ORGANIZATION_TYPE_Construction	99.328685
129	NAME_FAMILY_STATUS_Widow	98.631146
217	OCCUPATION_TYPE_Core staff	91.826127
228	OCCUPATION_TYPE_Sales staff	91.511697
25	APARTMENTS_AVG	88.609484

53	APARTMENTS_MEDI	87.332153
134	NAME_HOUSING_TYPE_Rented apartment	85.727777
250	NUMBER_FAMILY_MEMBERS_two	83.000603
0	AMT_INCOME_TOTAL	79.697778
224	OCCUPATION_TYPE_Managers	76.848477
39	APARTMENTS_MODE	76.238322
33	FLOORSMIN_AVG	75.487045
203	ORGANIZATION_TYPE_Transport: type 3	73.694931
61	FLOORSMIN_MEDI	73.091264
98	AMT_REQ_CREDIT_BUREAU_YEAR	70.392198
47	FLOORSMIN_MODE	70.376105
249	NUMBER_FAMILY_MEMBERS_three_plus	63.195011
220	OCCUPATION_TYPE_High skill tech staff	50.862403
230	OCCUPATION_TYPE_Security staff	50.014630
26	BASEMENTAREA_AVG	48.016617
188	ORGANIZATION_TYPE_School	46.025391
54	BASEMENTAREA_MEDI	45.458173
242	WALLSMATERIAL_MODE_Stone, brick	44.926757
1	AMT_CREDIT	41.884275
216	OCCUPATION_TYPE_Cooking staff	40.940439
31	ENTRANCES_AVG	39.429477
59	ENTRANCES_MEDI	39.400943
35	LIVINGAPARTMENTS_AVG	37.988358
63	LIVINGAPARTMENTS_MEDI	36.611648
87	FLAG_DOCUMENT_16	36.435649
40	BASEMENTAREA_MODE	35.665519
56	YEARS_BUILD_MEDI	35.358525
28	YEARS_BUILD_AVG	35.246605
122	NAME_EDUCATION_TYPE_Lower secondary	34.820010
42	YEARS_BUILD_MODE	34.147157
137	REGION_RATING_CLIENT_2	33.593134

187	ORGANIZATION_TYPE_Restaurant	33.449146
49	LIVINGAPARTMENTS_MODE	33.001197
45	ENTRANCES_MODE	32.858163
247	FONDKAPREMONT_MODE_reg oper spec account	30.999298
84	FLAG_DOCUMENT_13	30.248832
245	FONDKAPREMONT_MODE_org spec account	29.872789
57	COMMONAREA_MEDI	27.419156
29	COMMONAREA_AVG	27.195589
140	REGION_RATING_CLIENT_W_CITY_2	26.920902
112	NAME_INCOME_TYPE_Commercial associate	26.407128
179	ORGANIZATION_TYPE_Medicine	25.878244
96	AMT_REQ_CREDIT_BUREAU_MON	24.883604
180	ORGANIZATION_TYPE_Military	24.860737
160	ORGANIZATION_TYPE_Government	23.012125
183	ORGANIZATION_TYPE_Police	22.996228
190	ORGANIZATION_TYPE_Security Ministries	22.325024
85	FLAG_DOCUMENT_14	22.270497
151	ORGANIZATION_TYPE_Bank	21.582545
68	OBS_30_CNT_SOCIAL_CIRCLE	21.539336
70	OBS_60_CNT_SOCIAL_CIRCLE	20.763439
239	WALLSMATERIAL_MODE_Monolithic	20.625953
243	WALLSMATERIAL_MODE_Wooden	20.565558
43	COMMONAREA_MODE	20.548122
169	ORGANIZATION_TYPE_Industry: type 3	20.308288
73	FLAG_DOCUMENT_2	18.931986
225	OCCUPATION_TYPE_Medicine staff	17.111182
89	FLAG_DOCUMENT_18	16.691907
38	NONLIVINGAREA_AVG	15.939986
200	ORGANIZATION_TYPE_Trade: type 7	15.807700
66	NONLIVINGAREA_MEDI	15.601615
196	ORGANIZATION_TYPE_Trade: type 3	15.230959

150	ORGANIZATION_TYPE_Agriculture	15.174833
163	ORGANIZATION_TYPE_Industry: type 1	15.080143
231	OCCUPATION_TYPE_Waiters/barmen staff	15.003952
55	YEARS_BEGINEXPLUATATION_MEDI	14.919927
205	ORGANIZATION_TYPE_University	14.719397
52	NONLIVINGAREA_MODE	14.526821
79	FLAG_DOCUMENT_8	14.285048
117	NAME_INCOME_TYPE_Unemployed	14.129753
27	YEARS_BEGINEXPLUATATION_AVG	13.881824
215	OCCUPATION_TYPE_Cleaning staff	12.054264
109	FLAG_OWN_REALTY_N	11.936645
110	FLAG_OWN_REALTY_Y	11.936645
208	NAME_TYPE_SUITE_Family	11.908629
41	YEARS_BEGINEXPLUATATION_MODE	11.846402
213	NAME_TYPE_SUITE_Unaccompanied	11.159846
237	WALLSMATERIAL_MODE_Block	10.779570
166	ORGANIZATION_TYPE_Industry: type 12	10.587189
189	ORGANIZATION_TYPE_Security	10.553042
204	ORGANIZATION_TYPE_Transport: type 4	10.501781
233	EMERGENCYSTATE_MODE_Yes	10.331828
17	REG_REGION_NOT_WORK_REGION	9.773022
62	LANDAREA_MEDI	9.456271
177	ORGANIZATION_TYPE_Kindergarten	8.910753
16	REG_REGION_NOT_LIVE_REGION	8.888041
34	LANDAREA_AVG	8.487783
86	FLAG_DOCUMENT_15	8.097889
48	LANDAREA_MODE	7.846998
199	ORGANIZATION_TYPE_Trade: type 6	7.657575
97	AMT_REQ_CREDIT_BUREAU_QRT	6.639310
248	NUMBER_FAMILY_MEMBERS_one	6.402200
133	NAME_HOUSING_TYPE_Office apartment	5.850871

119	NAME_EDUCATION_TYPE_Academic degree	5.834419
235	HOUSETYPE_MODE_specific housing	5.799445
182	ORGANIZATION_TYPE_Other	5.775608
170	ORGANIZATION_TYPE_Industry: type 4	5.669210
175	ORGANIZATION_TYPE_Industry: type 9	5.574003
211	NAME_TYPE_SUITE_Other_B	5.037998
94	AMT_REQ_CREDIT_BUREAU_DAY	4.873239
113	NAME_INCOME_TYPE_Maternity leave	4.732608
153	ORGANIZATION_TYPE_Business Entity Type 2	4.640050
143	WEEKDAY_APPR_PROCESS_START_MONDAY	4.344612
158	ORGANIZATION_TYPE_Electricity	4.222862
147	WEEKDAY_APPR_PROCESS_START_TUESDAY	4.214625
195	ORGANIZATION_TYPE_Trade: type 2	3.997287
226	OCCUPATION_TYPE_Private service staff	3.974990
229	OCCUPATION_TYPE_Secretaries	3.938090
82	FLAG_DOCUMENT_11	3.862618
157	ORGANIZATION_TYPE_Culture	3.822379
244	FONDKAPREMONT_MODE_not specified	3.709716
185	ORGANIZATION_TYPE_Realtor	3.672893
80	FLAG_DOCUMENT_9	3.538684
161	ORGANIZATION_TYPE_Hotel	3.293603
121	NAME_EDUCATION_TYPE_Incomplete higher	3.078681
144	WEEKDAY_APPR_PROCESS_START_SATURDAY	2.607667
219	OCCUPATION_TYPE_HR staff	2.507271
197	ORGANIZATION_TYPE_Trade: type 4	2.406371
181	ORGANIZATION_TYPE_Mobile	2.100879
184	ORGANIZATION_TYPE_Postal	2.033214
155	ORGANIZATION_TYPE_Cleaning	1.943421
88	FLAG_DOCUMENT_17	1.820263
198	ORGANIZATION_TYPE_Trade: type 5	1.756634
159	ORGANIZATION_TYPE_Emergency	1.615530

75	FLAG_DOCUMENT_4	1.582233
201	ORGANIZATION_TYPE_Transport: type 1	1.524013
192	ORGANIZATION_TYPE_Services	1.479096
92	FLAG_DOCUMENT_21	1.449279
165	ORGANIZATION_TYPE_Industry: type 11	1.438828
171	ORGANIZATION_TYPE_Industry: type 5	1.425255
162	ORGANIZATION_TYPE_Housing	1.415296
18	LIVE_REGION_NOT_WORK_REGION	1.376643
194	ORGANIZATION_TYPE_Trade: type 1	1.342503
210	NAME_TYPE_SUITE_Other_A	1.334424
116	NAME_INCOME_TYPE_Student	1.318510
142	WEEKDAY_APPR_PROCESS_START_FRIDAY	1.272125
202	ORGANIZATION_TYPE_Transport: type 2	1.205310
164	ORGANIZATION_TYPE_Industry: type 10	1.188234
176	ORGANIZATION_TYPE_Insurance	1.155082
236	HOUSETYPE_MODE_terraced house	1.085247
132	NAME_HOUSING_TYPE_Municipal apartment	1.038929
78	FLAG_DOCUMENT_7	0.989680
145	WEEKDAY_APPR_PROCESS_START_SUNDAY	0.919733
240	WALLSMATERIAL_MODE_Others	0.869154
111	NAME_INCOME_TYPE_Businessman	0.791085
37	NONLIVINGAPARTMENTS_AVG	0.563170
126	NAME_FAMILY_STATUS_Separated	0.526550
207	NAME_TYPE_SUITE_Children	0.494781
212	NAME_TYPE_SUITE_Spouse, partner	0.441211
65	NONLIVINGAPARTMENTS_MEDI	0.427392
238	WALLSMATERIAL_MODE_Mixed	0.417794
2	AMT_ANNUITY	0.413282
93	AMT_REQ_CREDIT_BUREAU_HOUR	0.394736
193	ORGANIZATION_TYPE_Telecom	0.385876
178	ORGANIZATION_TYPE_Legal Services	0.384521

91	FLAG_DOCUMENT_20	0.380741
81	FLAG_DOCUMENT_10	0.351586
221	OCCUPATION_TYPE_IT staff	0.337380
227	OCCUPATION_TYPE_Realty agents	0.335111
76	FLAG_DOCUMENT_5	0.310443
209	NAME_TYPE_SUITE_Group of people	0.303351
106	CODE_GENDER_XNA	0.263688
148	WEEKDAY_APPR_PROCESS_START_WEDNESDAY	0.228726
51	NONLIVINGAPARTMENTS_MODE	0.152578
152	ORGANIZATION_TYPE_Business Entity Type 1	0.101873
146	WEEKDAY_APPR_PROCESS_START_THURSDAY	0.097273
128	NAME_FAMILY_STATUS_Unknown	0.087895
83	FLAG_DOCUMENT_12	0.087895
9	FLAG_MOBIL	0.087893
172	ORGANIZATION_TYPE_Industry: type 6	0.080245
173	ORGANIZATION_TYPE_Industry: type 7	0.064579
12	FLAG_CONT_MOBILE	0.064415
167	ORGANIZATION_TYPE_Industry: type 13	0.054384
174	ORGANIZATION_TYPE_Industry: type 8	0.030311
90	FLAG_DOCUMENT_19	0.028870
149	ORGANIZATION_TYPE_Advertising	0.014839
130	NAME_HOUSING_TYPE_Co-op apartment	0.007719
186	ORGANIZATION_TYPE_Religion	0.006141
95	AMT_REQ_CREDIT_BUREAU_WEEK	0.002720
168	ORGANIZATION_TYPE_Industry: type 2	0.000671
14	FLAG_EMAIL	0.000029

251 rows × 2 columns

In [635]:
```
# Select the top k=30 features according to their f_classif scores.
selectK_top_features = selectKBest_feature_scores_ranked['Feature Name
'].values[:30]
print(selectK_top_features)
```

```
['EXT_SOURCE_2' 'EXT_SOURCE_3' 'EXT_SOURCE_1' 'DAYS_BIRTH'
 'NAME_EDUCATION_TYPE_Higher education' 'NAME_INCOME_TYPE_Working'
 'CODE_GENDER_M' 'CODE_GENDER_F' 'DAYS_ID_PUBLISH' 'DAYS_LAST_PHONE_
CHANGE'
 'REG_CITY_NOT_WORK_CITY' 'REGION_RATING_CLIENT_W_CITY_3'
 'NAME_EDUCATION_TYPE_Secondary / secondary special'
 'REGION_RATING_CLIENT_3' 'NAME_INCOME_TYPE_Pensioner' 'HAS_JOB'
 'ORGANIZATION_TYPE_XNA' 'FLAG_EMP_PHONE' 'FLAG_DOCUMENT_3'
 'REG_CITY_NOT_LIVE_CITY' 'OCCUPATION_TYPE_Laborers'
 'EMERGENCYSTATE_MODE_No' 'DAYS_REGISTRATION'
 'REGION_RATING_CLIENT_W_CITY_1' 'REGION_RATING_CLIENT_1'
 'HOUSETYPE_MODE_block of flats' 'REGION_POPULATION_RELATIVE'
 'DEF_30_CNT_SOCIAL_CIRCLE' 'LIVE_CITY_NOT_WORK_CITY'
 'DEF_60_CNT_SOCIAL_CIRCLE']
```

In [636]:
```
# Determine what number of features have an aggregate f_classif score
that
# comprises 90% of the aggregate f_classif of all features.

aggregate_f_classif_score_all_features = selectKBest_feature_scores_ra
nked['Score'].sum()
aggregate_f_classif_score_top_30_features = selectKBest_feature_scores
_ranked[:30]['Score'].sum()
print('Aggregate f_classif score of all 251 features: {}'.format(aggre
gate_f_classif_score_all_features))
print('Aggregate f_classif score of top 30 features: {}'.format(aggreg
ate_f_classif_score_all_features))
print('Top 30 features\' total score is {}% of the total score of all
251 features.'.format(round(aggregate_f_classif_score_top_30_features*
100./aggregate_f_classif_score_all_features,2)))
```

```
Aggregate f_classif score of all 251 features: 39060.04565296721
Aggregate f_classif score of top 30 features: 39060.04565296721
Top 30 features' total score is 76.92% of the total score of all 251
features.
```

In [637]:
```
# Reduce the training dataset to the top 30 features:
X_train_final = X_train_processed[selectK_top_features]
```

In [638]:
```
display(X_train_final)
```

	EXT_SOURCE_2	EXT_SOURCE_3	EXT_SOURCE_1	DAYS_BIRTH	NAME_EDU
123473	0.358568	0.563835	0.524685	0.105975	0

10118	0.490305	0.595456	0.244926	0.806539	0
64716	0.643404	0.706205	0.502462	0.113641	0
234940	0.426431	0.506484	0.288642	0.818828	0
236051	0.445701	0.528093	0.790210	0.531623	0
30611	0.716030	0.546023	0.502462	0.479594	1
871	0.683097	0.511034	0.386333	0.851635	1
153082	0.181508	0.746300	0.502462	0.425761	0
188110	0.501046	0.725276	0.502462	0.488219	1
278046	0.288642	0.636376	0.502462	0.331116	0
26644	0.743533	0.081186	0.502462	0.787655	0
247686	0.505397	0.554947	0.502462	0.792334	0
115392	0.105422	0.691021	0.502462	0.839684	1
3950	0.486961	0.493863	0.502462	0.556595	0
54612	0.653632	0.662638	0.502462	0.531736	0
17380	0.762581	0.633032	0.502462	0.352818	1
146196	0.606717	0.382502	0.502462	0.416347	0
24352	0.568313	0.610991	0.354043	0.682751	0
188080	0.478874	0.511034	0.611665	0.777170	0
102436	0.511562	0.542445	0.502462	0.526268	1
158911	0.705507	0.397946	0.924966	0.507948	1
123277	0.563042	0.463275	0.371116	0.510879	0
114608	0.515009	0.511034	0.502462	0.403157	0
17870	0.511169	0.742182	0.411021	0.737148	0
197941	0.747103	0.716570	0.502462	0.568884	1
53438	0.465103	0.511034	0.502462	0.170688	0
64507	0.624017	0.709189	0.502462	0.443630	0
255439	0.570672	0.588488	0.388705	0.552537	0
218916	0.208960	0.511034	0.487548	0.612232	1
295147	0.456058	0.646330	0.502462	0.820462	0
182050	0.441130	0.598926	0.502462	0.081003	0
87096	0.199041	0.477649	0.502462	0.560992	0

172732	0.460710	0.513694	0.502462	0.113191	0
98310	0.670774	0.326475	0.502462	0.898647	1
288663	0.288822	0.353988	0.502462	0.895321	0
214467	0.116698	0.511034	0.498825	0.781454	0
247130	0.537984	0.549597	0.458548	0.761556	1
109604	0.759539	0.166406	0.502462	0.864713	0
73005	0.390561	0.511034	0.502462	0.270970	1
128669	0.186399	0.631355	0.480488	0.052255	0
296977	0.490977	0.810618	0.502462	0.378523	0
190097	0.488168	0.538863	0.430888	0.907723	1
130675	0.707534	0.511034	0.840726	0.216685	0
111766	0.446620	0.511034	0.774983	0.315220	0
28852	0.449783	0.511034	0.502462	0.914600	0
107963	0.752819	0.639708	0.502462	0.543968	0
89263	0.677184	0.413597	0.502462	0.245434	1
159031	0.243943	0.754406	0.502462	0.553664	1
156618	0.604209	0.408359	0.502462	0.608794	0
274042	0.124147	0.511034	0.502462	0.269222	0
9154	0.390332	0.678568	0.502462	0.564994	0
269933	0.412904	0.531686	0.750257	0.179425	0
19801	0.001707	0.511034	0.575900	0.423619	0
164848	0.714328	0.106156	0.332448	0.712401	1
61860	0.172073	0.675413	0.502462	0.211838	0
42512	0.733945	0.619528	0.502462	0.554059	1
177530	0.276948	0.511034	0.502462	0.747520	0
50358	0.377455	0.629674	0.076492	0.790079	0
134464	0.628950	0.404878	0.502462	0.953044	0
115885	0.582283	0.431192	0.373082	0.671815	0
229436	0.571068	0.656158	0.502462	0.786415	0
239717	0.624171	0.511034	0.502462	0.736528	0
129475	0.407297	0.232725	0.502462	0.362063	0

284087	0.237517	0.511034	0.502462	0.272492	0
35901	0.470292	0.661024	0.502462	0.364769	0
291686	0.711060	0.710674	0.747863	0.381229	0
200245	0.408411	0.659406	0.756371	0.244814	0
215522	0.653548	0.614414	0.502462	0.893236	0
272400	0.406057	0.746300	0.571282	0.428749	0
235942	0.433349	0.213967	0.502462	0.224126	0
212595	0.571732	0.528093	0.167642	0.787148	0
243900	0.632993	0.424130	0.502462	0.401240	0
60106	0.601183	0.636376	0.338494	0.666516	0
98450	0.230590	0.368969	0.502462	0.089121	1
253524	0.685619	0.122180	0.801598	0.311330	0
34303	0.648336	0.689479	0.502462	0.424803	0
33986	0.690678	0.629674	0.502462	0.221759	0
307328	0.581443	0.511034	0.502462	0.654848	0
127384	0.596765	0.695622	0.502462	0.062345	0
271288	0.320505	0.695622	0.502462	0.416798	0
295490	0.677771	0.511034	0.502462	0.191995	0
278189	0.652904	0.323311	0.494823	0.799718	0
34614	0.618868	0.511034	0.502462	0.943517	0
149615	0.478214	0.673830	0.502462	0.192277	0
77443	0.688436	0.798137	0.628136	0.812740	0
179564	0.542177	0.375711	0.502462	0.160090	0
150275	0.409611	0.520898	0.502462	0.771477	0
148777	0.255051	0.511034	0.502462	0.640586	0
101942	0.720975	0.511034	0.502462	0.283258	0
272225	0.341015	0.755740	0.502462	0.447069	1
66673	0.498837	0.569149	0.502462	0.640192	0
147090	0.703253	0.584990	0.502462	0.700564	1
77204	0.425666	0.747663	0.568502	0.842559	0
233326	0.220650	0.353988	0.688064	0.171364	0

205036	0.437317	0.251239	0.647650	0.201917	0
178930	0.641827	0.694093	0.502462	0.673055	1
56363	0.584197	0.511034	0.502462	0.287711	1
53669	0.106549	0.511034	0.502462	0.962401	0
302784	0.005740	0.846378	0.509937	0.538331	0
132760	0.192316	0.511892	0.502462	0.301522	0
288521	0.677590	0.511034	0.502462	0.688444	0
261341	0.738780	0.687933	0.369809	0.861950	0
243593	0.349582	0.511034	0.502462	0.383484	0
266528	0.193409	0.283712	0.556335	0.441883	0
90544	0.438797	0.722393	0.606939	0.603777	1
34136	0.303049	0.392774	0.502462	0.250620	0
42832	0.657537	0.511892	0.413462	0.820744	1
144727	0.267025	0.609276	0.282714	0.813923	1
240325	0.723055	0.397946	0.599863	0.229312	0
119732	0.698771	0.272134	0.436843	0.545265	0
24487	0.627132	0.511034	0.502462	0.186077	0
48665	0.579755	0.392774	0.502462	0.529763	1
150432	0.651986	0.468660	0.502462	0.785287	0
267262	0.423475	0.340906	0.502462	0.599605	0
155284	0.657665	0.581484	0.502462	0.254340	0
206231	0.295501	0.511034	0.827174	0.012120	0
79659	0.250618	0.191822	0.502462	0.093799	0
105861	0.596118	0.540654	0.502462	0.076156	0
60905	0.183960	0.816092	0.662243	0.479425	1
97973	0.604419	0.511034	0.648761	0.321364	1
77723	0.722012	0.294083	0.502462	0.891488	0
21849	0.700246	0.511034	0.570431	0.359865	0
152601	0.515752	0.081726	0.502462	0.803551	0
58344	0.564591	0.474051	0.502462	0.473281	0
304943	0.481352	0.286652	0.502462	0.525536	0

117530	0.569203	0.418854	0.502462	0.373112	0
208339	0.277535	0.511034	0.502462	0.279481	0
305626	0.686251	0.733815	0.571351	0.517700	0
9155	0.535820	0.777659	0.142477	0.752255	1
207930	0.677619	0.535276	0.502462	0.100676	0
93512	0.293090	0.704706	0.502462	0.572322	0
273246	0.355947	0.511034	0.502462	0.238501	0
140715	0.490436	0.363945	0.490305	0.157554	0
132778	0.040029	0.396220	0.369844	0.606313	1
198173	0.696931	0.511034	0.502462	0.313191	0
292505	0.630164	0.643026	0.502462	0.285118	0
255106	0.243182	0.272134	0.502462	0.866516	0
138610	0.170726	0.511034	0.502462	0.881793	1
306173	0.754151	0.436506	0.502462	0.340474	0
220265	0.561621	0.529890	0.569087	0.419673	0
51256	0.414113	0.511034	0.567654	0.386753	0
201732	0.656670	0.511034	0.502462	0.684386	1
288536	0.683996	0.267869	0.502462	0.107610	0
112307	0.150884	0.304672	0.502462	0.594589	0
77721	0.313437	0.547810	0.764776	0.093292	0
172105	0.566617	0.641368	0.502462	0.377452	0
116818	0.267183	0.102119	0.502462	0.160710	0
180936	0.127223	0.621226	0.454342	0.415163	0
120658	0.431621	0.574447	0.502462	0.874915	1
118024	0.271646	0.340906	0.502462	0.885344	1
112504	0.606033	0.766234	0.770241	0.397407	0
67980	0.663290	0.461482	0.741147	0.113191	0
112916	0.712285	0.609276	0.244460	0.780327	0
101533	0.658143	0.511034	0.502462	0.542334	0
104147	0.300152	0.511034	0.345685	0.145998	0
76686	0.403907	0.180888	0.502462	0.436359	0

207436	0.479747	0.511034	0.366611	0.689515	0
172169	0.562424	0.349055	0.502462	0.274183	0
143284	0.114053	0.761026	0.502462	0.545998	0
210003	0.634937	0.753067	0.502462	0.426832	0
195107	0.736444	0.511034	0.439958	0.380665	0
289874	0.490988	0.349055	0.502462	0.720237	0
152680	0.141585	0.108924	0.405245	0.583878	0
120326	0.346877	0.751724	0.803791	0.636809	1
269356	0.090962	0.454321	0.395232	0.739402	0
287558	0.628955	0.483050	0.433595	0.639177	0
250086	0.425174	0.511034	0.149770	0.489966	0
26857	0.531036	0.258084	0.502462	0.332413	0
84845	0.029389	0.511034	0.502462	0.473393	0
52157	0.406336	0.607557	0.502462	0.908117	0
211319	0.626451	0.372334	0.502462	0.164600	0
137097	0.456166	0.542445	0.502462	0.078523	0
209872	0.650983	0.324891	0.502462	0.283766	1
301770	0.393355	0.501075	0.502462	0.354453	0
288760	0.689851	0.697147	0.557489	0.854171	1
48988	0.510644	0.659406	0.502462	0.360316	1
289763	0.653716	0.612704	0.698067	0.176494	0
27937	0.665620	0.269286	0.746170	0.080834	0
166121	0.771200	0.712155	0.648348	0.073844	0
232137	0.710250	0.709189	0.427683	0.701466	0
157941	0.564338	0.654529	0.652502	0.455186	0
99266	0.560027	0.306202	0.377689	0.563247	0
110569	0.595702	0.312365	0.502462	0.659639	1
82796	0.405550	0.722393	0.502462	0.167136	0
198461	0.130079	0.812823	0.606110	0.686077	1
137600	0.316656	0.511034	0.502462	0.763811	0
135401	0.565693	0.583238	0.308632	0.795998	1

32355	0.648356	0.380800	0.502462	0.201409	0
230153	0.627806	0.205598	0.497917	0.628579	0
96807	0.268663	0.759712	0.363247	0.452988	0
171943	0.443314	0.816092	0.366408	0.623675	0
196850	0.574541	0.565608	0.585259	0.541037	1
81644	0.592992	0.673830	0.488315	0.172097	0
71169	0.609281	0.700184	0.502462	0.670180	0
92030	0.267316	0.595456	0.688336	0.125085	0
163623	0.348091	0.511034	0.228615	0.820857	0
80188	0.636896	0.511034	0.502462	0.700113	0
253701	0.588128	0.486653	0.520878	0.894138	1
49531	0.317764	0.511034	0.502462	0.260372	0
95855	0.682002	0.353988	0.502462	0.274464	0
29159	0.163668	0.368969	0.502462	0.788895	1
306498	0.600978	0.647977	0.882463	0.223619	1
168319	0.221422	0.544235	0.398556	0.528298	0
204908	0.554859	0.511034	0.284806	0.217193	0
129134	0.378334	0.265049	0.631989	0.562514	0
247729	0.539119	0.793449	0.502462	0.296449	0
44818	0.527331	0.441836	0.502462	0.252368	0
80098	0.409061	0.730987	0.641819	0.333033	1
63516	0.490578	0.362277	0.381186	0.832244	1
212654	0.499685	0.454321	0.502462	0.557948	0
202974	0.752411	0.222581	0.856315	0.472266	0
205318	0.705366	0.712155	0.502462	0.197520	0
163661	0.643846	0.511034	0.502462	0.513923	0
176811	0.385462	0.355639	0.502462	0.126550	1
219430	0.720640	0.511034	0.342692	0.794081	0
1368	0.260856	0.706205	0.502462	0.496843	0
259107	0.683622	0.673830	0.502462	0.659357	1
224518	0.615360	0.562060	0.502462	0.203326	0

106822	0.714377	0.236611	0.502462	0.636359	0
224548	0.303456	0.662638	0.502462	0.190304	0
175438	0.779311	0.422370	0.332291	0.705693	0
302357	0.715002	0.499272	0.558847	0.726494	1
287159	0.508311	0.394495	0.320763	0.801015	1
63740	0.684246	0.740799	0.502462	0.184611	0
136381	0.402749	0.190706	0.502462	0.905299	0
186925	0.565661	0.470456	0.502462	0.387542	0
224592	0.658128	0.670652	0.502462	0.073224	0
190389	0.683712	0.622922	0.502462	0.198422	0
214556	0.512573	0.706205	0.502462	0.179820	0
254298	0.553098	0.581484	0.502462	0.240981	1
79111	0.596197	0.694093	0.724405	0.204228	0
21596	0.741115	0.342529	0.608261	0.590079	1
210443	0.726657	0.766234	0.502462	0.280101	0
6660	0.429391	0.629674	0.796800	0.331060	0
259398	0.116365	0.511034	0.534066	0.545941	0
96567	0.702764	0.683269	0.627016	0.447745	0
253843	0.665314	0.583238	0.502462	0.212740	0
289555	0.418343	0.511034	0.107190	0.861387	0
271248	0.223191	0.486653	0.108520	0.884611	0
232677	0.367178	0.253963	0.502462	0.542728	0
253321	0.733410	0.241861	0.502462	0.318489	0
95151	0.701566	0.709189	0.502462	0.501804	0
96812	0.163803	0.777659	0.502462	0.017587	0
168398	0.174921	0.710674	0.502462	0.326888	0
295235	0.653642	0.511034	0.502462	0.160823	1
118099	0.768710	0.588488	0.502462	0.345490	1
10180	0.642993	0.510090	0.502462	0.662965	0
216298	0.437591	0.384207	0.502462	0.390699	0
290928	0.485049	0.865896	0.502462	0.767136	0

77472	0.234437	0.403142	0.673917	0.629594	1
...	...	...	...	...	...
243548	0.589049	0.511034	0.502462	0.452593	0
133272	0.771258	0.511034	0.502462	0.052706	0
213090	0.245350	0.424130	0.269910	0.685682	0
300504	0.671271	0.631355	0.502462	0.076325	0
133629	0.588520	0.554947	0.502462	0.111725	0
179426	0.455125	0.553165	0.502462	0.058568	0
184423	0.585190	0.511034	0.502462	0.684949	0
267683	0.448247	0.771362	0.540896	0.740135	1
32711	0.720746	0.704706	0.353358	0.751973	1
143946	0.755797	0.659406	0.502462	0.852762	0
274327	0.606508	0.687933	0.502462	0.325648	1
6295	0.733201	0.593718	0.502462	0.246843	1
4499	0.667380	0.665855	0.502462	0.634498	1
66690	0.637841	0.321735	0.502462	0.107272	1
15151	0.599761	0.511034	0.508859	0.596731	0
173714	0.014522	0.207964	0.131686	0.880383	0
269544	0.344191	0.595456	0.502462	0.677847	0
174088	0.650785	0.119119	0.502462	0.287937	0
192506	0.469796	0.619528	0.729559	0.199887	0
291999	0.687741	0.424130	0.190005	0.735006	0
117796	0.705452	0.602386	0.556787	0.328918	0
283076	0.589552	0.105473	0.502462	0.703608	1
162688	0.357282	0.524496	0.502462	0.353044	0
133121	0.341123	0.083917	0.502462	0.361781	0
147443	0.160405	0.275000	0.379340	0.591263	0
107512	0.775181	0.586740	0.502462	0.089628	0
120975	0.564064	0.656158	0.610182	0.249831	0
147718	0.371186	0.633032	0.777601	0.534273	1
30306	0.491196	0.510090	0.502462	0.477452	0

85981	0.703568	0.612704	0.502462	0.222492	0
284062	0.577937	0.331251	0.400087	0.725141	1
47254	0.328144	0.612704	0.653159	0.243292	0
301528	0.702097	0.431192	0.502462	0.243517	1
8155	0.749947	0.638044	0.502462	0.149380	1
132874	0.272695	0.120641	0.502462	0.942052	0
299648	0.241355	0.105473	0.408043	0.218489	0
273255	0.727733	0.445396	0.553543	0.627565	1
219686	0.697987	0.329655	0.832507	0.418433	1
71295	0.533643	0.196334	0.502462	0.282469	1
84896	0.571866	0.607557	0.901461	0.208230	0
214835	0.726118	0.511034	0.912721	0.191770	0
133983	0.538869	0.540654	0.502462	0.642108	0
142483	0.756684	0.730987	0.732768	0.802480	1
237714	0.732936	0.294083	0.502462	0.898478	0
226814	0.670760	0.773896	0.467801	0.605975	1
34754	0.557375	0.477649	0.502462	0.714374	0
112547	0.539038	0.404878	0.502462	0.433822	1
133883	0.508316	0.336062	0.247742	0.696900	0
15708	0.549841	0.511034	0.300443	0.554566	1
51663	0.644662	0.673830	0.502462	0.822717	0
86202	0.503277	0.657784	0.502462	0.297802	0
22671	0.160288	0.389339	0.502462	0.332694	0
110078	0.057349	0.221335	0.772319	0.548591	0
34698	0.177272	0.528093	0.502462	0.725592	0
89045	0.528721	0.329655	0.494142	0.317926	0
106081	0.314709	0.652897	0.502462	0.114431	0
168229	0.632307	0.511034	0.589663	0.522661	0
287486	0.746126	0.629674	0.333642	0.667644	0
109751	0.029899	0.504681	0.502462	0.521646	1
98506	0.274185	0.511034	0.502462	0.195265	0

183734	0.171181	0.622922	0.502462	0.117080	1
70467	0.771659	0.772631	0.798818	0.421533	1
194806	0.288153	0.562060	0.626181	0.149605	0
157381	0.098745	0.806149	0.502462	0.146167	0
235167	0.637159	0.511034	0.447376	0.224126	0
125657	0.302486	0.511034	0.502462	0.806370	1
218164	0.458381	0.077499	0.321505	0.816234	0
177789	0.491010	0.754406	0.502462	0.381849	0
257426	0.468805	0.700184	0.502462	0.713303	0
269536	0.479185	0.506484	0.502462	0.140361	0
112859	0.671961	0.090815	0.502462	0.510316	0
225281	0.568849	0.511034	0.343132	0.806708	1
288880	0.621264	0.746300	0.502462	0.333596	0
246458	0.456562	0.746300	0.502462	0.116347	0
93264	0.660353	0.820383	0.688413	0.216404	0
256951	0.117928	0.554947	0.507374	0.657328	0
216688	0.273267	0.440058	0.502462	0.195434	0
67172	0.673992	0.738020	0.371270	0.744701	0
241025	0.540776	0.420611	0.659305	0.430778	0
65318	0.586368	0.706205	0.502462	0.160147	0
214283	0.560744	0.377404	0.502462	0.518884	0
193075	0.689921	0.511034	0.118002	0.948309	0
39353	0.388762	0.785052	0.502462	0.648140	0
118451	0.469425	0.511034	0.502462	0.124690	1
68840	0.606132	0.546023	0.502462	0.186979	0
61087	0.055869	0.595456	0.502462	0.437317	0
165838	0.504621	0.292588	0.502462	0.559752	0
13545	0.698927	0.634706	0.699513	0.335344	0
228576	0.514385	0.497469	0.502462	0.491094	1
265892	0.634282	0.265049	0.471588	0.789346	0
225928	0.474761	0.511034	0.502462	0.094983	0

136602	0.103586	0.445396	0.502462	0.280496	0
21959	0.635504	0.683269	0.892529	0.177170	0
55016	0.433172	0.408359	0.762635	0.282018	0
127948	0.109211	0.511034	0.502462	0.083145	0
92787	0.400122	0.511034	0.502462	0.191375	0
137848	0.729213	0.522697	0.502462	0.116122	0
128376	0.777149	0.511034	0.502462	0.245096	0
161152	0.139769	0.408359	0.502462	0.346110	0
285920	0.676256	0.481249	0.502462	0.359132	1
44262	0.415143	0.567379	0.245165	0.702255	0
75766	0.444383	0.744932	0.502462	0.839741	0
288249	0.181751	0.265049	0.284552	0.951127	0
18047	0.489108	0.579727	0.502462	0.216855	0
152906	0.769040	0.511034	0.777013	0.169617	0
25939	0.341413	0.384207	0.502462	0.653608	0
200235	0.514385	0.723837	0.502462	0.516460	0
243307	0.179895	0.703203	0.502462	0.647520	0
2693	0.565478	0.746300	0.839738	0.092672	0
109556	0.012572	0.511034	0.502462	0.914262	0
64044	0.603079	0.382502	0.583951	0.189290	0
43585	0.711343	0.683269	0.502462	0.264036	1
164899	0.513797	0.712155	0.502462	0.668771	0
250396	0.680159	0.662638	0.502462	0.139741	0
249002	0.645834	0.384207	0.502462	0.539008	0
198286	0.668607	0.417100	0.502462	0.649718	0
202982	0.759248	0.770087	0.731817	0.504228	1
105878	0.615184	0.687933	0.584956	0.677734	0
211428	0.585944	0.622922	0.493064	0.543574	1
110687	0.401661	0.511034	0.241867	0.939515	0
185340	0.574626	0.071055	0.502462	0.664769	0
107059	0.197120	0.657784	0.502462	0.323168	0

122096	0.492994	0.085595	0.175511	0.914092	0
248550	0.598848	0.344155	0.502462	0.678805	1
50015	0.486267	0.689479	0.715775	0.526719	1
245310	0.381615	0.574447	0.502462	0.369504	0
253618	0.556668	0.598926	0.454929	0.743461	1
23419	0.342830	0.511034	0.502462	0.865220	1
122409	0.698863	0.461482	0.335649	0.702029	1
218969	0.582709	0.736623	0.502462	0.158061	0
218126	0.527462	0.610991	0.349502	0.617362	0
119176	0.014315	0.475850	0.603088	0.378072	0
121626	0.574717	0.733815	0.502462	0.551240	0
90982	0.714507	0.636376	0.793094	0.304340	0
98806	0.033521	0.511034	0.502462	0.205919	0
214020	0.615986	0.429424	0.502462	0.678072	1
184064	0.658335	0.511034	0.502462	0.151578	0
158338	0.471986	0.669057	0.733521	0.434555	1
139182	0.212466	0.681706	0.502462	0.311218	0
201664	0.709494	0.095070	0.250386	0.833822	0
24538	0.673093	0.755740	0.466927	0.532525	0
273109	0.288122	0.761026	0.502462	0.489966	0
123684	0.364221	0.483050	0.387094	0.566460	0
77373	0.726827	0.372334	0.502462	0.398027	0
297366	0.499029	0.510090	0.502462	0.868320	0
30535	0.358840	0.617826	0.708992	0.374126	0
270536	0.208819	0.511034	0.502462	0.543968	0
284806	0.651902	0.570917	0.502462	0.201409	1
183323	0.565757	0.211551	0.266392	0.894645	0
104488	0.600931	0.511034	0.502462	0.743236	0
288998	0.593019	0.275000	0.502462	0.225197	0
134633	0.606148	0.807274	0.502462	0.113134	1
300804	0.633206	0.457900	0.869146	0.630665	1

12666	0.521583	0.349055	0.502462	0.232356	1
129312	0.383113	0.511034	0.502462	0.508568	0
61858	0.578225	0.387625	0.809070	0.485851	0
13986	0.627382	0.782608	0.502462	0.653100	0
164231	0.545714	0.438281	0.662712	0.803326	0
301648	0.397228	0.251239	0.502462	0.250338	0
136330	0.384364	0.519097	0.287243	0.595039	0
179262	0.683035	0.515495	0.424607	0.553946	0
124375	0.031548	0.689479	0.502462	0.207272	0
133767	0.449626	0.294083	0.287380	0.883315	0
120151	0.507250	0.492060	0.502462	0.297238	0
264712	0.744960	0.291097	0.118813	0.865558	0
302918	0.401855	0.256706	0.721345	0.824803	0
48984	0.668050	0.506484	0.502462	0.613416	0
38044	0.753209	0.746300	0.502462	0.576888	0
236584	0.687990	0.614414	0.502462	0.154904	0
217851	0.716208	0.788681	0.502462	0.613923	0
194776	0.624597	0.362277	0.598003	0.309921	1
274329	0.609515	0.511034	0.593740	0.628016	0
9268	0.655145	0.459690	0.502462	0.265276	0
82798	0.631076	0.424130	0.502462	0.348027	0
24300	0.517101	0.252599	0.210627	0.593856	0
23247	0.601026	0.511034	0.502462	0.753382	1
220884	0.348146	0.413597	0.502462	0.563754	0
55591	0.668040	0.511034	0.397582	0.723788	0
263160	0.561508	0.829750	0.502462	0.399887	0
257750	0.636248	0.590233	0.585619	0.719391	1
40397	0.529522	0.511034	0.450636	0.871871	1
11534	0.557720	0.511034	0.561073	0.388839	0
99299	0.458348	0.399676	0.502462	0.403551	0
223165	0.183987	0.499272	0.502462	0.551804	0

37065	0.455537	0.636376	0.502462	0.132694	0
196769	0.727048	0.629674	0.668181	0.616742	0
202283	0.556776	0.511034	0.364565	0.851297	1
271836	0.268521	0.511034	0.603256	0.280496	0
171829	0.352765	0.558507	0.250749	0.571139	0
240181	0.423347	0.511034	0.564334	0.776099	0
208261	0.145880	0.511034	0.502462	0.617756	0
281601	0.627306	0.511034	0.502462	0.736753	0
200551	0.640554	0.621226	0.884205	0.271082	1
298064	0.762391	0.372334	0.710067	0.765276	0
80077	0.650904	0.789880	0.502462	0.311894	0
106530	0.632297	0.627991	0.502462	0.429030	1
279303	0.018314	0.511034	0.558322	0.113980	0
256508	0.694315	0.591977	0.502462	0.154453	0
48555	0.697642	0.698668	0.502462	0.325310	1
154555	0.669241	0.497469	0.779832	0.393743	0
251995	0.490409	0.332851	0.502462	0.421139	0
124243	0.597586	0.511034	0.502462	0.084273	1
252801	0.516795	0.493863	0.624748	0.177508	1
68148	0.615194	0.614414	0.502462	0.270688	0
207624	0.621809	0.593718	0.521853	0.311218	0
174073	0.700402	0.385915	0.502462	0.180214	0
205041	0.170701	0.401407	0.502462	0.270011	0
239629	0.667729	0.304672	0.625365	0.249887	0
239931	0.723261	0.798137	0.502462	0.503720	1
270936	0.679645	0.190706	0.502462	0.377903	0
141699	0.388689	0.843544	0.477087	0.556032	0
256840	0.718818	0.880268	0.679845	0.516460	0
41606	0.526181	0.846378	0.502462	0.131454	0
3890	0.590446	0.643026	0.502462	0.133709	0
273538	0.614464	0.569149	0.815896	0.317982	0

31551	0.472188	0.511034	0.578797	0.637768	0
67435	0.637553	0.723837	0.502462	0.155468	0
84654	0.730975	0.511034	0.502462	0.160034	0
129981	0.598779	0.452534	0.374224	0.584047	0
65725	0.608917	0.188490	0.447677	0.437655	0
123855	0.558405	0.803885	0.502462	0.476832	1
2747	0.718509	0.511034	0.502462	0.124239	0
130523	0.720829	0.502878	0.502462	0.410485	0
149503	0.590879	0.303146	0.700113	0.630609	0
156730	0.676758	0.511034	0.502462	0.199380	0
258795	0.638508	0.713631	0.703340	0.491939	0
184779	0.630077	0.267869	0.502462	0.686133	0
214176	0.688060	0.432962	0.502462	0.602311	1
235796	0.703955	0.511034	0.502462	0.404904	1
103355	0.089042	0.165407	0.502462	0.642165	0
267455	0.679955	0.718033	0.502462	0.626437	1
199041	0.644206	0.670652	0.502462	0.193010	0
252709	0.540532	0.535276	0.502462	0.226043	0
194027	0.136195	0.520898	0.502462	0.484611	0
262913	0.518139	0.349055	0.502462	0.936753	1
64820	0.738505	0.768808	0.596087	0.544701	1
41090	0.738370	0.352340	0.737286	0.309865	0
278167	0.699755	0.716570	0.460309	0.556933	0
191335	0.663158	0.173527	0.502462	0.217531	1
175203	0.265018	0.466864	0.724350	0.626494	0
87498	0.317323	0.738020	0.502462	0.363585	0
137337	0.627444	0.385915	0.470037	0.841319	1
54886	0.628664	0.546023	0.502462	0.216234	0
207892	0.446620	0.511034	0.502462	0.270913	0
110268	0.767108	0.372334	0.502462	0.519673	0
119879	0.617180	0.511034	0.502462	0.605862	0

259178	0.722398	0.370650	0.455170	0.548873	0
131932	0.608427	0.554947	0.305919	0.615558	1
146867	0.597959	0.259468	0.611373	0.705919	0
121958	0.408400	0.275000	0.351724	0.679425	1

246008 rows × 30 columns

In [639]:
```
# 21. Build a data preprocessing pipeline to used for all testing sets
.
# This pipeline will recreate all features that were engineered in the
# training set during the original data preprocessing phase.
# The pipeline will also apply the imputer, min-max, and PCA transform
s
# originally fit on features in the training set to all datapoints in
a
# testing set.

def adjust_columns_application_test_csv_table(testing_dataframe):
    """
    After it is one-hot encoded, application_test.csv data table will
have one
    extra column, 'REGION_RATING_CLIENT_W_CITY_-1', that is not presen
t in the
    training dataframe. This column will be removed from the testing d
atatable
    in this case. Only 1 of the 48,744 rows in application_test.csv wi
ll have a
    value of 1 for this feature following one-hot encoding. I am not w
orried
    about this column's elimination from the testing dataframe affecti
ng predictions.

    Additionally, unlike the test validation set, which originally com
prised 20% of
    application_train.csv, application_test.csv will be missing the fo
llowing columns
    after it is one-hot encoded:

    'CODE_GENDER_XNA', 'NAME_INCOME_TYPE_Maternity leave', 'NAME_FAMIL
Y_STATUS_Unknown'

    In this case, we need to insert these columns into the testing dat
aframe, at
    the exact same indices they are located at in the fully preprocess
ed training
    dataframe.Each inserted column will be filled with all zeros. (If
each of these
    binary features are missing from the application_test.csv data tab
le, we can infer
    that each borrower in thatdata table obviously would have a 0 for
```

```
each feature were
    it present.)

    Parameters:
        testing_dataframe: Pandas dataframe containing the testing dat
aset
                           contained in the file application_test.csv

    Returns: a testing dataframe containing the exact same columns and
             column order as found in the training dataframe
    """

    # Identify any columns in the one-hot encoded testing_dataframe th
at
    # are not in X_train_raw. These columns will need to be removed fr
om the
    # testing_dataframe. (Expected that there will only be one such
    # column: 'REGION_RATING_CLIENT_W_CITY_-1')
    X_train_columns_list = X_train_raw.columns.tolist()
    testing_dataframe_columns_list = testing_dataframe.columns.tolist(
)
    for column_name in X_train_columns_list:
        if column_name in testing_dataframe_columns_list:
            testing_dataframe_columns_list.remove(column_name)
            columns_not_in_X_train_raw = testing_dataframe_columns_lis
t

    # Drop any column from the testing_dataframe that is not in the
    # training dataframe. Expected to only be the one column 'REGION_R
ATING_CLIENT_W_CITY_-1'
    for column in columns_not_in_X_train_raw:
        testing_dataframe = testing_dataframe.drop(column, axis=1)

    # Get the column indices of each of the features 'CODE_GENDER_XNA'
,
    #'NAME_INCOME_TYPE_Maternity leave', 'NAME_FAMILY_STATUS_Unknown'
from
    # the raw training dataframe (X_train_raw) prior to having having
PCA run on it.
    loc_code_gender_training_frame = X_train_raw.columns.get_loc('CODE
_GENDER_XNA')
    loc_name_income_type_maternity_leave_training_frame = X_train_raw.
columns.get_loc('NAME_INCOME_TYPE_Maternity leave')
    loc_name_family_status_unknown_training_frame = X_train_raw.column
s.get_loc('NAME_FAMILY_STATUS_Unknown')

    # Insert each column into the testing dataframe at the same index
it had
    # in the X_train_raw dataframe before PCA was run. Fill each colum
n with all 0s.
    # Order is important. 'CODE_GENDER_XNA' should be inserted first,
followed by
    # 'NAME_INCOME_TYPE_Maternity leave', and then finally 'NAME_FAMIL
```

```
Y_STATUS_Unknown'.
    testing_dataframe.insert(loc=loc_code_gender_training_frame, colum
n='CODE_GENDER_XNA', value=0)
    testing_dataframe.insert(loc=loc_name_income_type_maternity_leave_
training_frame, column='NAME_INCOME_TYPE_Maternity leave', value=0)
    testing_dataframe.insert(loc=loc_name_family_status_unknown_traini
ng_frame, column='NAME_FAMILY_STATUS_Unknown', value=0)
    return testing_dataframe

def test_set_preprocessing_pipeline(testing_dataframe):
    """
    Recreate all features that were engineered in the training set dur
ing
    the original data preprocessing phase. Missing numerical 'NaN' val
ues
    will be filled with an imputer. Missing binary categorical feature
'NaN'
    values will be replaced with 0. The pipeline will also apply
    the min-max and PCA transforms originally fit on features
    in the training set to numerical features in the testing set.

    Parameters:
        testing_dataframe: Pandas dataframe containing a testing datas
et

    Returns: a fully preprocessed testing dataframe
    """

    # Create the HAS_CHILDREN feature.
    CNT_CHILDREN_test = testing_dataframe['CNT_CHILDREN']
    HAS_CHILDREN = CNT_CHILDREN_test.map(lambda x: 1 if x > 0 else 0)

    # Append the newly engineered HAS_CHILDREN feature to the main dat
aframe.
    testing_dataframe = testing_dataframe.assign(HAS_CHILDREN=HAS_CHIL
DREN.values)

    # Drop the CNT_CHILDREN column from the main dataframe
    testing_dataframe = testing_dataframe.drop('CNT_CHILDREN', axis=1)

    # Create the NUMBER_FAMILY_MEMBERS feature.
    CNT_FAM_MEMBERS_test = testing_dataframe['CNT_FAM_MEMBERS']
    NUMBER_FAMILY_MEMBERS = CNT_FAM_MEMBERS_test.map(lambda x: 'one' i
f x == 1 else ('two' if x == 2 else 'three_plus'))

    # Append the newly engineered FAMILY_SIZE feature to the main data
frame.
    testing_dataframe = testing_dataframe.assign(NUMBER_FAMILY_MEMBERS
=NUMBER_FAMILY_MEMBERS.values)

    # Drop the CNT_FAM_MEMBERS feature from the main dataframe
    testing_dataframe = testing_dataframe.drop('CNT_FAM_MEMBERS', axis
=1)
```

```
    # Build the HAS_CREDIT_BUREAU_LOANS_OVERDUE feature
    testing_dataframe = build_feature_HAS_CREDIT_BUREAU_LOANS_OVERDUE(
testing_dataframe)

    # Create the HAS_JOB feature
    DAYS_EMPLOYED_test = testing_dataframe['DAYS_EMPLOYED']
    HAS_JOB = DAYS_EMPLOYED_test.map(lambda x: 1 if x <= 0 else 0)

    # Append the newly engineered FAMILY_SIZE feature to the main data
frame.
    testing_dataframe = testing_dataframe.assign(HAS_JOB=HAS_JOB.value
s)

    # Drop the DAYS_EMPLOYED feature from the main dataframe
    testing_dataframe = testing_dataframe.drop('DAYS_EMPLOYED', axis=1
)

    # Translate the two negatively-valued features DAYS_REGISTRATION,
and
    # DAYS_LAST_PHONE_CHANGE to positive values
    translate_negative_valued_features(testing_dataframe, non_norm_fea
t_neg_values_skewed)

    # Log-transform all 17 non-normalized numerical features that have
skewed distributions.
    testing_dataframe[log_transform_feats] = testing_dataframe[log_tra
nsform_feats].apply(lambda x: np.log(x + 1))

    # Create a list of all numerical features in the testing dataframe
that have at least one 'NaN' entry
    numerical_features_with_nan_testing = testing_dataframe[numerical_
features].columns[testing_dataframe[numerical_features].isna().any()].
tolist()

    # Use an imputer to replace 'NaN' values for all numerical feature
s with each feature's mean.
    testing_dataframe[numerical_features_with_nan_testing] = imputer.f
it_transform(testing_dataframe[numerical_features_with_nan_testing])

    # Remove the borrower ID column, SK_ID_CURR, from the main datafra
me
    testing_dataframe = testing_dataframe.drop('SK_ID_CURR', axis=1)

    # One-hot encode all 19 non-binary categorical features.
    testing_dataframe = pd.get_dummies(testing_dataframe, columns=cat_
feat_need_one_hot)

    # After one-hot encoding, the testing dataframe from application_t
est.csv will be
    # missing 2 columns that are in the training dataframe. It will al
so have an extra
    # column that was not in the training dataframe, giving it 249 tot
```

```
al columns.
    # If this is the case, we need to modify this testing dataframe so
that its columns
    # and column order is consistent with the training dataframe.
    if testing_dataframe.shape[1] == 249:
        testing_dataframe = adjust_columns_application_test_csv_table(
testing_dataframe)

    # Create a list of the binary categorical features with at least o
ne 'NaN' entry
    bin_cat_feat_with_nan_testing = testing_dataframe[all_bin_cat_feat
].columns[testing_dataframe[all_bin_cat_feat].isna().any()].tolist()

    # Replace each 'NaN' value in each of these binary features with 0
    testing_dataframe[bin_cat_feat_with_nan_testing] = testing_datafra
me[bin_cat_feat_with_nan_testing].fillna(value=0)

    # Transform each of the 21 features that need to be scaled to the
range [0,1] using
    # the min-max scaler fit on the training set.
    testing_dataframe[feats_to_scale] = scaler.transform(testing_dataf
rame[feats_to_scale])

    # Reduce the testing dataframe to the top selectKbest features:
    testing_dataframe = testing_dataframe[selectK_top_features]

    return testing_dataframe

# 22. Preprocess the test validation set.
X_test_final = test_set_preprocessing_pipeline(X_test_raw)

# Verify that both the training and test validation dataframes have th
e expected number of columns after
# preprocessing its data and reducing their featurespace to the top 30
features returned by SelectKBest.
print('Training set preprocessing complete. The final training datafra
me now has {} columns. Expected: 30.'.format(X_train_final.shape[1]))
print('Test validation set preprocessing complete. The final test vali
dation dataframe now has {} columns. Expected: 30.'.format(X_test_fina
l.shape[1]))
```

```
Training set preprocessing complete. The final training dataframe no
w has 30 columns. Expected: 30.
Test validation set preprocessing complete. The final test validatio
n dataframe now has 30 columns. Expected: 30.
```

In [640]:
```
# Train the classifiers and compute prediction probabilities.

# 1. Use a Gaussian Naive Bayes classifier to make predictions on
# the test validation set. Calculate the area under ROC curve score of
# these predictions.

# Fit a Gaussian Naive Bayes classifier to the training dataframe.
```

```
clf_naive_bayes = GaussianNB()
clf_naive_bayes.fit(X_train_final, y_train)

# The Naive Bayes estimates of probability of the positive class (TARG
ET=1):
# the probability estimate of each borrower making at least one late l
oan payment.
naive_bayes_selectKbest_y_score = clf_naive_bayes.predict_proba(X_test
_final)[:, 1]

# The area under the ROC curve between the true target values and the
# probability estimates of the predicted values.
naive_bayes_selectKbest_roc_auc_score = roc_auc_scorer(y_test, naive_b
ayes_selectKbest_y_score)

# Add the Naive Bayes classifier's scores to the results list.
y_score_list.append(naive_bayes_selectKbest_y_score)
clf_label_list.append('Naive Bayes SelectKBest, K=30')

print('Naive Bayes (SelectKBest) test validation set predictions\' ROC
AUC score: {}'.format(naive_bayes_selectKbest_roc_auc_score))

# 2. Use an AdaBoost classifier to make predictions on the test valida
tion set.
# Calculate the area under ROC curve score of these predictions.

# Fit the AdaBoost classifier, using the parameter for 'n_estimators'
discovered
# when running GridSearchCV on AdaBoost above for the full featureset.
clf_AdaBoost = AdaBoostClassifier(learning_rate=1.0, n_estimators=1000
, random_state=42)
clf_AdaBoost.fit(X_train_final, y_train)

# The AdaBoost classifier's estimates of probability of the positive c
lass (TARGET=1):
# the probability estimate of each borrower making at least one late l
oan payment.
adaBoost_selectKbest_y_score = clf_AdaBoost.predict_proba(X_test_final
)[:, 1]

# The area under the ROC curve between the true target values and the
# probability estimates of the predicted values.
adaBoost_selectKbest_roc_auc_score = roc_auc_scorer(y_test, adaBoost_s
electKbest_y_score)

# Add the AdaBoost classifier's scores to the results list.
y_score_list.append(adaBoost_selectKbest_y_score)
clf_label_list.append('AdaBoost SelectKBest, K=30')

print('AdaBoost (SelectKBest) test validation set predictions\' ROC AU
C score: {}'.format(adaBoost_selectKbest_roc_auc_score))

# 3. Try using a Logistic Regression classifier to make predictions.
```

```
# Fit the classifier to the training data.
clf_logistic_regression = LogisticRegression(penalty='l2', random_stat
e=42, solver='liblinear', max_iter=100)
clf_logistic_regression.fit(X_train_final, y_train)

# The logistical regression classifier's estimates of probability of t
he positive
# class (TARGET=1): the probability estimate of each borrower making a
t least one
# late loan payment.
logistic_regression_selectKbest_y_score = clf_logistic_regression.pred
ict_proba(X_test_final)[:, 1]

# The area under the ROC curve between the true target values and the
# probability estimates of the predicted values.
logistic_regression_selectKbest_roc_auc_score = roc_auc_scorer(y_test,
logistic_regression_selectKbest_y_score)

# Add the Logistic Regression classifier's scores to the results list.
y_score_list.append(logistic_regression_selectKbest_y_score)
clf_label_list.append('Logistic Regression SelectKBest, K=30')

print('Logistic Regression (SelectKBest) test validation set predictio
ns\' ROC AUC score: {}'.format(logistic_regression_selectKbest_roc_auc
_score))

# 4. Try using a Multi-layer Perceptron classifier to make predictions
.

# Fit the classifier to the training data.
clf_mlp = MLPClassifier(
    hidden_layer_sizes=100, activation='identity', solver='adam', alph
a=0.001, batch_size=200,
    learning_rate_init=0.001, random_state=42
)
clf_mlp.fit(X_train_final, y_train)

# The multi-layer perceptron classifier's estimates of probability of
the positive
# class (TARGET=1): the probability estimate of each borrower making a
t least one
# late loan payment.
mlp_selectKbest_y_score = clf_mlp.predict_proba(X_test_final)[:, 1]

# The area under the ROC curve between the true target values and the
# probability estimates of the predicted values.
mlp_selectKbest_roc_auc_score = roc_auc_scorer(y_test, mlp_selectKbest
_y_score)

# Add the Multi-Layer Perceptron classifier's scores to the results li
st.
y_score_list.append(mlp_selectKbest_y_score)
```

```
clf_label_list.append('Multi-Layer Perceptron SelectKBest, K=30')

print('Multi-layer Perceptron (SelectKBest) test validation set predic
tions\' ROC AUC score: {}'.format(mlp_selectKbest_roc_auc_score))

# 5. Try using a LightGBM classifier.

# Convert preprocessed training dataset into LightGBM dataset format
lightgbm_training = lgb.Dataset(X_train_final, label=y_train)

# Specify parameters
params = {}
params['learning_rate'] = 0.01
params['boosting_type'] = 'gbdt'
params['objective'] = 'binary'
params['metric'] = 'auc'
params['sub_feature'] = 0.3
params['num_leaves'] = 100
params['min_data_in_leaf'] = 500
params['max_depth'] = 10
params['max_bin'] = 64
#params['min_data_in_bin'] = 3
#params['lambda_l1'] = 0.01
params['lambda_l2'] = 0.01
#params['min_gain_to_split'] = 0.01
params['bagging_freq'] = 100
params['bagging_fraction'] = 0.9
#params['feature_fraction'] = 0.5

# Fit the LightGBM classifier to the training data
clf_lgb = lgb.train(params, lightgbm_training, 1500)

# Classifier's estimates of probability of the positive class (TARGET=
1): the
# probability estimate of each borrower making at least one late loan
payment.
lgb_selectKbest_y_score = clf_lgb.predict(X_test_final)

# The area under the ROC curve between the true target values and the
# probability estimates of the predicted values.
lgb_selectKbest_roc_auc_score = roc_auc_scorer(y_test, lgb_selectKbest
_y_score)

# Add the LightGBM classifier's scores to the results list.
y_score_list.append(lgb_selectKbest_y_score)
clf_label_list.append('LightGBM SelectKBest, K=30')

print('LightGBM (SelectKBest) test validation set predictions\' ROC AU
C score: {}'.format(lgb_selectKbest_roc_auc_score))
```

```
Naive Bayes (SelectKBest) test validation set predictions' ROC AUC s
core: 0.6748662184461512
AdaBoost (SelectKBest) test validation set predictions' ROC AUC scor
e: 0.7332167998294561
Logistic Regression (SelectKBest) test validation set predictions' R
OC AUC score: 0.7367170423891469
Multi-layer Perceptron (SelectKBest) test validation set predictions
' ROC AUC score: 0.7358901049573279
LightGBM (SelectKBest) test validation set predictions' ROC AUC scor
e: 0.7394787228642934
```

GridSearchCV Tuning of LightGBM classifier trained on full featureset

In [654]:
```
# Step 3. Train a LightGBM classifier on the full featureset, and use
GridSearchCV
# to develop further intuition on LightGBM parameter tuning.

# Load the main data tables
application_train_data = pd.read_csv("data/application_train.csv")
application_test_data = pd.read_csv("data/application_test.csv")

# Load the Bureau data table
bureau_data = pd.read_csv("data/bureau.csv")

# 1: Create lists of different feature types in the main data
# frame, based on how each type will need to be preprocessed.

# i. All 18 categorical features needing one-hot encoding.
#    Includes the 4 categorical features originally
#    mis-identified as having been normalized:
#    EMERGENCYSTATE_MODE, HOUSETYPE_MODE, WALLSMATERIAL_MODE,
#    FONDKAPREMONT_MODE
cat_feat_need_one_hot = [
    'NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR',
    'FLAG_OWN_REALTY', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE',
    'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'REGION_RATING_CLIENT',
    'REGION_RATING_CLIENT_W_CITY', 'WEEKDAY_APPR_PROCESS_START', 'ORGA
NIZATION_TYPE',
    'NAME_TYPE_SUITE', 'OCCUPATION_TYPE', 'EMERGENCYSTATE_MODE',
    'HOUSETYPE_MODE', 'WALLSMATERIAL_MODE', 'FONDKAPREMONT_MODE'
]

# ii. All 32 binary categorical features already one-hot encoded.
bin_cat_feat = [
    'FLAG_MOBIL', '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'
]

# iii. All 2 non-normalized numerical features with skewed distributio
ns
#    and negative values. These features will need to have their
#    distributions translated to positive ranges before being
#    log-transformed, and then later scaled to the range [0,1].
non_norm_feat_neg_values_skewed = [
    'DAYS_REGISTRATION', 'DAYS_LAST_PHONE_CHANGE'
]


# iv. All 15 non-normalized numerical features with skewed distributio
ns,
#    and only positive values. These features will need to be
#    log-transformed, and eventually scaled to the range [0,1].
non_norm_feat_pos_values_skewed = [
    'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY',
    'AMT_GOODS_PRICE', 'OBS_30_CNT_SOCIAL_CIRCLE', 'DEF_30_CNT_SOCIAL_
CIRCLE',
    'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIRCLE', 'AMT_REQ_C
REDIT_BUREAU_HOUR',
    'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK', 'AMT_RE
Q_CREDIT_BUREAU_MON',
    'AMT_REQ_CREDIT_BUREAU_QRT', 'AMT_REQ_CREDIT_BUREAU_YEAR', 'OWN_CA
R_AGE'
]

# v. All 4 numerical features with normal shapes but needing to be sca
led
#    to the range [0,1].
norm_feat_need_scaling = [
    'DAYS_BIRTH', 'DAYS_ID_PUBLISH', 'HOUR_APPR_PROCESS_START',
    'REGION_POPULATION_RELATIVE'
]

# vi. All 46 numerical features that have been normalized to the range
#   [0,1]. These features will need neither log-transformation, nor
#   any further scaling.
norm_feat_not_need_scaling = [
    'EXT_SOURCE_2', 'EXT_SOURCE_3', 'YEARS_BEGINEXPLUATATION_AVG',
    'YEARS_BEGINEXPLUATATION_MODE', 'YEARS_BEGINEXPLUATATION_MEDI', 'F
LOORSMAX_AVG',
    'FLOORSMAX_MODE', 'FLOORSMAX_MEDI', 'LIVINGAREA_AVG',
    'LIVINGAREA_MODE', 'LIVINGAREA_MEDI', 'ENTRANCES_AVG',
    'ENTRANCES_MODE', 'ENTRANCES_MEDI', 'APARTMENTS_AVG',
    'APARTMENTS_MODE', 'APARTMENTS_MEDI', 'ELEVATORS_AVG',
```

```
    'ELEVATORS_MODE', 'ELEVATORS_MEDI', 'NONLIVINGAREA_AVG',
    'NONLIVINGAREA_MODE', 'NONLIVINGAREA_MEDI', 'EXT_SOURCE_1',
    'BASEMENTAREA_AVG', 'BASEMENTAREA_MODE', 'BASEMENTAREA_MEDI',
    'LANDAREA_AVG', 'LANDAREA_MODE', 'LANDAREA_MEDI',
    'YEARS_BUILD_AVG', 'YEARS_BUILD_MODE', 'YEARS_BUILD_MEDI',
    'FLOORSMIN_AVG', 'FLOORSMIN_MODE', 'FLOORSMIN_MEDI',
    'LIVINGAPARTMENTS_AVG', 'LIVINGAPARTMENTS_MODE', 'LIVINGAPARTMENTS
_MEDI',
    'NONLIVINGAPARTMENTS_AVG', 'NONLIVINGAPARTMENTS_MODE', 'NONLIVINGA
PARTMENTS_MEDI',
    'COMMONAREA_AVG', 'COMMONAREA_MODE', 'COMMONAREA_MEDI',
    'TOTALAREA_MODE'
]

# vii. The remaining 3 features in the main data frame that will be
#    re-engineered and transformed into different features
feat_to_be_reengineered = [
    'CNT_CHILDREN', 'CNT_FAM_MEMBERS', 'DAYS_EMPLOYED'
]

# 2: Separate target data from training dataset.
targets = application_train_data['TARGET']
features_raw = application_train_data.drop('TARGET', axis = 1)

# 3: Use train_test_split from sklearn.cross_validation to
# create a test validation set that is 20% of the size of the total tr
aining set:
# Will allow me to compare performance of various learning algorithms
without
# overfitting to the training data.
X_train_raw, X_test_raw, y_train, y_test = train_test_split(features_r
aw,
                                                            targets,
                                                            test_size = 0.2,
                                                            random_state = 42)

# 4: Use the CNT_CHILDREN feature to engineer a binary
# categorical feature called HAS_CHILDREN. If value of CNT_CHILDREN is
# greater than 0, the value of HAS_CHILDREN will be 1. If value of CNT
_CHILDREN is
# 0, value of HAS_CHILDREN will be 0.
CNT_CHILDREN_train = X_train_raw['CNT_CHILDREN']
HAS_CHILDREN = CNT_CHILDREN_train.map(lambda x: 1 if x > 0 else 0)

# Append the newly engineered HAS_CHILDREN feature to the main datafra
me.
X_train_raw = X_train_raw.assign(HAS_CHILDREN=HAS_CHILDREN.values)

# 5: Drop the CNT_CHILDREN column from the main dataframe
X_train_raw = X_train_raw.drop('CNT_CHILDREN', axis=1)

# Add the new HAS_CHILDREN feature to the list of binary categorical
# features that are already one-hot encoded. There are now 33 such fea
```

```
tures.
bin_cat_feat = bin_cat_feat + ['HAS_CHILDREN']

# 6. Use the CNT_FAM_MEMBERS feature to engineer a categorical feature
called NUMBER_FAMILY_MEMBERS.
# If CNT_FAM_MEMBERS is 1.0, then the value of NUMBER_FAMILY_MEMBERS w
ill be 'one'. If CNT_FAM_MEMBERS is 2.0,
# then NUMBER_FAMILY_MEMBERS will be 'two'. If CNT_FAM_MEMBERS is 3.0
or greater, then NUMBER_FAMILY_MEMBERS will
# be 'three_plus'.
CNT_FAM_MEMBERS_train = X_train_raw['CNT_FAM_MEMBERS']
NUMBER_FAMILY_MEMBERS = CNT_FAM_MEMBERS_train.map(lambda x: 'one' if x
== 1 else ('two' if x == 2 else 'three_plus'))

# Append the newly engineered FAMILY_SIZE feature to the main datafram
e.
X_train_raw = X_train_raw.assign(NUMBER_FAMILY_MEMBERS=NUMBER_FAMILY_M
EMBERS.values)

# 7. Drop the CNT_FAM_MEMBERS feature from the main dataframe
X_train_raw = X_train_raw.drop('CNT_FAM_MEMBERS', axis=1)

# Add the new NUMBER_FAMILY_MEMBERS feature to the list of categorical
# features that will need to be one-hot encoded. There are now 19 of t
hese features.
cat_feat_need_one_hot = cat_feat_need_one_hot + ['NUMBER_FAMILY_MEMBER
S']

# 8. Use the CREDIT_DAY_OVERDUE feature in bureau.csv to engineer the
binary
# categorical HAS_CREDIT_BUREAU_LOANS_OVERDUE feature. If CREDIT_DAY_O
VERDUE for a
# particular borrower ID (SK_ID_CURR) is greater than 0, then the valu
e of
# HAS_CREDIT_BUREAU_LOANS_OVERDUE will be 1. If CREDIT_DAY_OVERDUE for
a particular
# borrower ID is 0, then the value of HAS_CREDIT_BUREAU_LOANS_OVERDUE
will be 0.

# Filter the bureau data table for loans which are overdue (have a val
ue
# for CREDIT_DAY_OVERDUE that's greater than 0)
bureau_data_filtered_for_overdue = bureau_data[bureau_data['CREDIT_DAY
_OVERDUE'] > 0]

def build_feature_HAS_CREDIT_BUREAU_LOANS_OVERDUE(dataframe):
    """
    Use the CREDIT_DAY_OVERDUE feature in bureau.csv to engineer the b
inary
    categorical HAS_CREDIT_BUREAU_LOANS_OVERDUE feature. If CREDIT_DAY
_OVERDUE for a
    particular borrower ID (SK_ID_CURR) is greater than 0, then the va
lue of
```

```
    HAS_CREDIT_BUREAU_LOANS_OVERDUE will be 1. If CREDIT_DAY_OVERDUE f
or a particular
    borrower ID is 0, then the value of HAS_CREDIT_BUREAU_LOANS_OVERDU
E will be 0.

    Parameters:
        dataframe: Pandas dataframe containing a training or testing d
ataset

    Returns: The dataframe with HAS_CREDIT_BUREAU_LOANS_OVERDUE featur
e appended to it.
    """
    # Create a series called HAS_CREDIT_BUREAU_LOANS_OVERDUE and fill
it with zeros.
    # Its index is identical to that of the main dataframe. It will ev
entually be appended
    # to the main data frame as a column.
    HAS_CREDIT_BUREAU_LOANS_OVERDUE = pd.Series(data=0, index = datafr
ame['SK_ID_CURR'].index)

    # A list of all the borrowers IDs in the main dataframe
    main_data_table_borrower_IDs = dataframe['SK_ID_CURR'].values

    # For each loan in the bureau data table that is overdue
    # (has a value for CREDIT_DAY_OVERDUE that's greater than 0)
    for index, row in bureau_data_filtered_for_overdue.iterrows():
        # The borrower ID (SK_ID_CURR) that owns the overdue loan
        borrower_ID = row['SK_ID_CURR']
        # If the borrower ID owning the overdue loan is also
        # in the main data frame, then enter a value of 1 in
        # the series HAS_CREDIT_BUREAU_LOANS_OVERDUE at an index
        # that is identical to the index of the borrower ID
        # in the main data frame.
        if borrower_ID in main_data_table_borrower_IDs:
            # The index of the borrower's row in the main data table.
            borrower_index_main_data_table = dataframe.index[dataframe
['SK_ID_CURR'] == borrower_ID].tolist()[0]
            # Place a value of 1 at the index of the series HAS_CREDIT
_BUREAU_LOANS_OVERDUE
            # which corresponds to the index of the borrower's ID in t
he main data table.
            HAS_CREDIT_BUREAU_LOANS_OVERDUE.loc[borrower_index_main_da
ta_table] = 1
    # Append the newly engineered HAS_CREDIT_BUREAU_LOANS_OVERDUE feat
ure to the main dataframe.
    dataframe = dataframe.assign(HAS_CREDIT_BUREAU_LOANS_OVERDUE=HAS_C
REDIT_BUREAU_LOANS_OVERDUE.values)
    return dataframe

# Build the HAS_CREDIT_BUREAU_LOANS_OVERDUE feature
X_train_raw = build_feature_HAS_CREDIT_BUREAU_LOANS_OVERDUE(X_train_ra
w)
```

```
# Add the new HAS_CREDIT_BUREAU_LOANS_OVERDUE feature to the list of b
inary categorical
# features. There are now 34 of these features.
bin_cat_feat = bin_cat_feat + ['HAS_CREDIT_BUREAU_LOANS_OVERDUE']

# 9. Use the DAYS_EMPLOYED feature to engineer a binary categorical fe
ature called HAS_JOB.
# If the value of DAYS_EMPLOYED is 0 or less, then HAS_JOB will be 1.
Otherwise, HAS_JOB will
# be 0. This condition will apply to all borrowers who had a value of
365243 for DAYS_EMPLOYED,
# which I hypothesized can be best interpreted as meaning that a borro
wer does not have a job.
DAYS_EMPLOYED_train = X_train_raw['DAYS_EMPLOYED']
HAS_JOB = DAYS_EMPLOYED_train.map(lambda x: 1 if x <= 0 else 0)

# Append the newly engineered FAMILY_SIZE feature to the main datafram
e.
X_train_raw = X_train_raw.assign(HAS_JOB=HAS_JOB.values)

# 10. Drop the DAYS_EMPLOYED feature from the main dataframe
X_train_raw = X_train_raw.drop('DAYS_EMPLOYED', axis=1)

# Add the new HAS_JOB feature to the list of binary categorical featur
es.
# There are now 35 of these features.
bin_cat_feat = bin_cat_feat + ['HAS_JOB']

# 11. Translate the 2 non-normalized numerical features that have skew
ed distributions
# and negative values: DAYS_REGISTRATION, and DAYS_LAST_PHONE_CHANGE

def translate_negative_valued_features(dataframe, feature_name_list):
    """
    Translate a dataset's continuous features containing several negat
ive
    values. The dataframe is modified such that all values of each fea
ture
    listed in the feature_name_list parameter become positive.

    Parameters:
        dataframe: Pandas dataframe containing the features
        feature_name_list: List of strings, containing the names
                           of each feature whose values will be
                           translated
    """
    for feature in feature_name_list:
        # The minimum, most-negative, value of the feature
        feature_min_value = dataframe[feature].min()
        # Translate each value of the feature in a positive direction,
        # of magnitude that's equal to the feature's most negative val
ue.
        dataframe[feature] = dataframe[feature].apply(lambda x: x - fe
```

```
ature_min_value)

# Translate the above two negatively-valued features to positive value
s
translate_negative_valued_features(X_train_raw, non_norm_feat_neg_valu
es_skewed)

# 12. Log-transform all 17 non-normalized numerical features that have
skewed distributions.
# These 17 features include the 2 that were translated to positive ran
ges in Step 11.

# Add the 2 features translated to positive ranges above in Step 11 to
# the list of non-normalized skewed features with positive values. Thi
s is
# the set of features that will be log-transformed
log_transform_feats = non_norm_feat_pos_values_skewed + non_norm_feat_
neg_values_skewed

X_train_raw[log_transform_feats] = X_train_raw[log_transform_feats].ap
ply(lambda x: np.log(x + 1))

# 13. Replace 'NaN' values for all numerical features with each featur
e's mean. Fit an imputer
# to each numerical feature containing at least one 'NaN' entry.

# Create a list of all the 67 numerical features in the main dataframe
. These include all
# 17 features that were log-transformed in Step 12, as well as the 4 n
ormal features that
# still need to be scaled, as well as the 46 normal features that don'
t need scaling.
numerical_features = log_transform_feats + norm_feat_need_scaling + no
rm_feat_not_need_scaling

# Create a list of all numerical features in the training set that hav
e at least one 'NaN' entry
numerical_features_with_nan = X_train_raw[numerical_features].columns[
X_train_raw[numerical_features].isna().any()].tolist()

# Create an imputer
imputer = Imputer()
# Fit the imputer to each numerical feature in the training set that h
as 'NaN' values,
# and replace each 'NaN' entry of each feature with that feature's mea
n.
X_train_raw[numerical_features_with_nan] = imputer.fit_transform(X_tra
in_raw[numerical_features_with_nan])

# 14. Remove the borrower ID column, SK_ID_CURR, from the main datafra
me
X_train_raw = X_train_raw.drop('SK_ID_CURR', axis=1)
```

```
# 15. One-hot encode all 19 non-binary categorical features.
X_train_raw = pd.get_dummies(X_train_raw, columns=cat_feat_need_one_ho
t)

# Create a list that includes only the newly one-hot encoded features
# as well as all the categorical features that were already binary.
all_bin_cat_feat = X_train_raw.columns.tolist()
for column_name in X_train_raw[numerical_features].columns.tolist():
    all_bin_cat_feat.remove(column_name)

# 16. Replace all 'NaN' values in all binary categorical features with
0.

# Create a list of binary categorical features with at least one 'NaN'
entry
bin_cat_feat_with_nan = X_train_raw[all_bin_cat_feat].columns[X_train_
raw[all_bin_cat_feat].isna().any()].tolist()

# Replace each 'NaN' value in each of these binary features with 0
X_train_raw[bin_cat_feat_with_nan] = X_train_raw[bin_cat_feat_with_nan
].fillna(value=0)

# 17. Fit a min-max scaler to each of the 17 log-transformed numerical
features, as well
# as to the 4 features DAYS_BIRTH, DAYS_ID_PUBLISH, HOUR_APPR_PROCESS_
START, and the normalized
# feature REGION_POPULATION_RELATIVE. Each feature will be scaled to a
range [0.0, 1.0].

# Build a list of all 21 features needing scaling. Add the list of fea
tures that
# were log-normalized above in Step 12 to the list of normally shaped
features
# that need to be scaled to the range [0,1].
feats_to_scale = norm_feat_need_scaling + log_transform_feats

# Initialize a scaler with the default range of [0,1]
scaler = MinMaxScaler()

# Fit the scaler to each of the features of the train set that need to
be scaled,
# then transform each of these features' values to the new scale.
X_train_raw[feats_to_scale] = scaler.fit_transform(X_train_raw[feats_t
o_scale])

# Rename the dataframe to indicate that its columns have been fully pr
eprocessed.
X_train_final = X_train_raw
```

In [648]:
```
# 18. Build a data preprocessing pipeline to used for all testing sets
.
# This pipeline will recreate all features that were engineered in the
# training set during the original data preprocessing phase.
```

```
# The pipeline will also apply the imputer, min-max, and PCA transforms
# originally fit on features in the training set to all datapoints in a
# testing set.

def adjust_columns_application_test_csv_table(testing_dataframe):
    """
    After it is one-hot encoded, application_test.csv data table will have one
    extra column, 'REGION_RATING_CLIENT_W_CITY_-1', that is not present in the
    training dataframe. This column will be removed from the testing datatable
    in this case. Only 1 of the 48,744 rows in application_test.csv will have a
    value of 1 for this feature following one-hot encoding. I am not worried
    about this column's elimination from the testing dataframe affecting predictions.

    Additionally, unlike the test validation set, which originally comprised 20% of
    application_train.csv, application_test.csv will be missing the following columns
    after it is one-hot encoded:

    'CODE_GENDER_XNA', 'NAME_INCOME_TYPE_Maternity leave', 'NAME_FAMILY_STATUS_Unknown'

    In this case, we need to insert these columns into the testing dataframe, at
    the exact same indices they are located at in the fully preprocessed training
    dataframe.Each inserted column will be filled with all zeros. (If each of these
    binary features are missing from the application_test.csv data table, we can infer
    that each borrower in thatdata table obviously would have a 0 for each feature were
    it present.)

    Parameters:
        testing_dataframe: Pandas dataframe containing the testing dataset
                           contained in the file application_test.csv

    Returns: a testing dataframe containing the exact same columns and
             column order as found in the training dataframe
    """

    # Identify any columns in the one-hot encoded testing_dataframe that
```

```
    # are not in X_train_raw. These columns will need to be removed fr
om the
    # testing_dataframe. (Expected that there will only be one such
    # column: 'REGION_RATING_CLIENT_W_CITY_-1')
    X_train_columns_list = X_train_raw.columns.tolist()
    testing_dataframe_columns_list = testing_dataframe.columns.tolist(
)
    for column_name in X_train_columns_list:
        if column_name in testing_dataframe_columns_list:
            testing_dataframe_columns_list.remove(column_name)
            columns_not_in_X_train_raw = testing_dataframe_columns_lis
t

    # Drop any column from the testing_dataframe that is not in the
    # training dataframe. Expected to only be the one column 'REGION_R
ATING_CLIENT_W_CITY_-1'
    for column in columns_not_in_X_train_raw:
        testing_dataframe = testing_dataframe.drop(column, axis=1)

    # Get the column indices of each of the features 'CODE_GENDER_XNA'
,
    #'NAME_INCOME_TYPE_Maternity leave', 'NAME_FAMILY_STATUS_Unknown'
from
    # the raw training dataframe (X_train_raw) prior to having having
PCA run on it.
    loc_code_gender_training_frame = X_train_raw.columns.get_loc('CODE
_GENDER_XNA')
    loc_name_income_type_maternity_leave_training_frame = X_train_raw.
columns.get_loc('NAME_INCOME_TYPE_Maternity leave')
    loc_name_family_status_unknown_training_frame = X_train_raw.column
s.get_loc('NAME_FAMILY_STATUS_Unknown')

    # Insert each column into the testing dataframe at the same index
it had
    # in the X_train_raw dataframe before PCA was run. Fill each colum
n with all 0s.
    # Order is important. 'CODE_GENDER_XNA' should be inserted first,
followed by
    # 'NAME_INCOME_TYPE_Maternity leave', and then finally 'NAME_FAMIL
Y_STATUS_Unknown'.
    testing_dataframe.insert(loc=loc_code_gender_training_frame, colum
n='CODE_GENDER_XNA', value=0)
    testing_dataframe.insert(loc=loc_name_income_type_maternity_leave_
training_frame, column='NAME_INCOME_TYPE_Maternity leave', value=0)
    testing_dataframe.insert(loc=loc_name_family_status_unknown_traini
ng_frame, column='NAME_FAMILY_STATUS_Unknown', value=0)
    return testing_dataframe

def test_set_preprocessing_pipeline(testing_dataframe):
    """
    Recreate all features that were engineered in the training set dur
ing
    the original data preprocessing phase. The pipeline will also appl
```

```
y
    an imputer to the test data table fill 'NaN' values. Binary featur
e's 'Nan'
    values will be filled with 0. The min-max scaler fit on features
    in the training set will be applied to the numerical features in t
he testing set.

    Parameters:
        testing_dataframe: Pandas dataframe containing a testing datas
et

    Returns: a fully preprocessed testing dataframe
    """

    # Create the HAS_CHILDREN feature.
    CNT_CHILDREN_test = testing_dataframe['CNT_CHILDREN']
    HAS_CHILDREN = CNT_CHILDREN_test.map(lambda x: 1 if x > 0 else 0)

    # Append the newly engineered HAS_CHILDREN feature to the main dat
aframe.
    testing_dataframe = testing_dataframe.assign(HAS_CHILDREN=HAS_CHIL
DREN.values)

    # Drop the CNT_CHILDREN column from the main dataframe
    testing_dataframe = testing_dataframe.drop('CNT_CHILDREN', axis=1)

    # Create the NUMBER_FAMILY_MEMBERS feature.
    CNT_FAM_MEMBERS_test = testing_dataframe['CNT_FAM_MEMBERS']
    NUMBER_FAMILY_MEMBERS = CNT_FAM_MEMBERS_test.map(lambda x: 'one' i
f x == 1 else ('two' if x == 2 else 'three_plus'))

    # Append the newly engineered FAMILY_SIZE feature to the main data
frame.
    testing_dataframe = testing_dataframe.assign(NUMBER_FAMILY_MEMBERS
=NUMBER_FAMILY_MEMBERS.values)

    # Drop the CNT_FAM_MEMBERS feature from the main dataframe
    testing_dataframe = testing_dataframe.drop('CNT_FAM_MEMBERS', axis
=1)

    # Build the HAS_CREDIT_BUREAU_LOANS_OVERDUE feature
    testing_dataframe = build_feature_HAS_CREDIT_BUREAU_LOANS_OVERDUE(
testing_dataframe)

    # Create the HAS_JOB feature
    DAYS_EMPLOYED_test = testing_dataframe['DAYS_EMPLOYED']
    HAS_JOB = DAYS_EMPLOYED_test.map(lambda x: 1 if x <= 0 else 0)

    # Append the newly engineered FAMILY_SIZE feature to the main data
frame.
    testing_dataframe = testing_dataframe.assign(HAS_JOB=HAS_JOB.value
s)
```

```
    # Drop the DAYS_EMPLOYED feature from the main dataframe
    testing_dataframe = testing_dataframe.drop('DAYS_EMPLOYED', axis=1
)

    # Translate the two negatively-valued features DAYS_REGISTRATION,
and
    # DAYS_LAST_PHONE_CHANGE to positive values
    translate_negative_valued_features(testing_dataframe, non_norm_fea
t_neg_values_skewed)

    # Log-transform all 17 non-normalized numerical features that have
skewed distributions.
    testing_dataframe[log_transform_feats] = testing_dataframe[log_tra
nsform_feats].apply(lambda x: np.log(x + 1))

    # Create a list of all numerical features in the testing dataframe
that have at least one 'NaN' entry
    numerical_features_with_nan_testing = testing_dataframe[numerical_
features].columns[testing_dataframe[numerical_features].isna().any()].
tolist()

    # Use an imputer to replace 'NaN' values for all numerical feature
s with each feature's mean.
    testing_dataframe[numerical_features_with_nan_testing] = imputer.f
it_transform(testing_dataframe[numerical_features_with_nan_testing])

    # Remove the borrower ID column, SK_ID_CURR, from the main datafra
me
    testing_dataframe = testing_dataframe.drop('SK_ID_CURR', axis=1)

    # One-hot encode all 19 non-binary categorical features.
    testing_dataframe = pd.get_dummies(testing_dataframe, columns=cat_
feat_need_one_hot)

    # After one-hot encoding, the testing dataframe from application_t
est.csv will be
    # missing 2 columns that are in the training dataframe. It will al
so have an extra
    # column that was not in the training dataframe, giving it 249 tot
al columns.
    # If this is the case, we need to modify this testing dataframe so
that its columns
    # and column order is consistent with the training dataframe.
    if testing_dataframe.shape[1] == 249:
        testing_dataframe = adjust_columns_application_test_csv_table(
testing_dataframe)

    # Create a list of the binary categorical features with at least o
ne 'NaN' entry
    bin_cat_feat_with_nan_testing = testing_dataframe[all_bin_cat_feat
].columns[testing_dataframe[all_bin_cat_feat].isna().any()].tolist()

    # Replace each 'NaN' value in each of these binary features with 0
```

```
    testing_dataframe[bin_cat_feat_with_nan_testing] = testing_datafra
me[bin_cat_feat_with_nan_testing].fillna(value=0)

    # Transform each of the 21 features that need to be scaled to the
range [0,1] using
    # the min-max scaler fit on the training set.
    testing_dataframe[feats_to_scale] = scaler.transform(testing_dataf
rame[feats_to_scale])

    return testing_dataframe

# 19. Preprocess the test validation set.
X_test_final = test_set_preprocessing_pipeline(X_test_raw)

# Verify that both the training and test validation dataframes have th
e expected number of columns after
# preprocessing its data and reducing their featurespace to the top 30
features returned by SelectKBest.
print('Training set preprocessing complete. The final training datafra
me now has {} columns. Expected: 251.'.format(X_train_final.shape[1]))
print('Test validation set preprocessing complete. The final test vali
dation dataframe now has {} columns. Expected: 251.'.format(X_test_fin
al.shape[1]))
```

```
Training set preprocessing complete. The final training dataframe no
w has 251 columns. Expected: 251.
Test validation set preprocessing complete. The final test validatio
n dataframe now has 251 columns. Expected: 251.
```

In [433]:
```
# Performs GridSearchCV on a LightGBM classifier learning
# algorithm to gain further intuition to aid in hyperparameter
# tuning.

# # Create cross-validation sets from the training data
# cv_sets = StratifiedKFold(n_splits = 5, random_state = 42)

# Transform 'roc_auc_scorer' into a scoring function using 'make_score
r'
scoring_fnc = make_scorer(roc_auc_scorer)

# Create an LightGBM classifier object
clf = lgb.LGBMClassifier(learning_rate = 0.1,
                         boosting_type = 'gbdt',
                         objective = 'binary',
                         metric = 'auc',
                         sub_feature = 0.3,
                         num_leaves = 50,
                         min_data_in_leaf = 500,
                         max_depth = -1,
                         max_bin = 100,
                         lambda_l2 = 0.1,
                         bagging_freq = 3,
                         bagging_fraction = 0.9,
                         random_state = 42,
                        )

# The parameters to search
grid_params = {
    'learning_rate': [0.001, 0.01, 0.1],
    'sub_feature': [0.3],
    'num_leaves': [200],
    'lambda_l2': [0.1],
    'min_data_in_leaf': [40],
    'max_depth': [-1]
}

# Create a GridSearchCV object.
grid = GridSearchCV(clf, grid_params, scoring_fnc, cv=3)

# Fit the grid search object to the data to compute the optimal model
grid.fit(X_train_final, y_train)

# Print the best parameters found
print('Best hyperparameter combo:\r')
print(grid.best_params_)
print('ROC AUC score of best hyperparameter combo:\r')
print(roc_auc_score(y_test, grid.predict_proba(X_test_final)[:,1]))
```

```
/anaconda3/envs/home_credit_default_risk_competition/lib/python3.5/s
ite-packages/lightgbm/engine.py:99: UserWarning: Found `num_iteratio
ns` in params. Will use it instead of argument
```

```
  warnings.warn("Found `{}` in params. Will use it instead of argume
nt".format(alias))
/anaconda3/envs/home_credit_default_risk_competition/lib/python3.5/s
ite-packages/lightgbm/engine.py:99: UserWarning: Found `num_iteratio
ns` in params. Will use it instead of argument
  warnings.warn("Found `{}` in params. Will use it instead of argume
nt".format(alias))
/anaconda3/envs/home_credit_default_risk_competition/lib/python3.5/s
ite-packages/lightgbm/engine.py:99: UserWarning: Found `num_iteratio
ns` in params. Will use it instead of argument
  warnings.warn("Found `{}` in params. Will use it instead of argume
nt".format(alias))
/anaconda3/envs/home_credit_default_risk_competition/lib/python3.5/s
ite-packages/lightgbm/engine.py:99: UserWarning: Found `num_iteratio
ns` in params. Will use it instead of argument
  warnings.warn("Found `{}` in params. Will use it instead of argume
nt".format(alias))
/anaconda3/envs/home_credit_default_risk_competition/lib/python3.5/s
ite-packages/lightgbm/engine.py:99: UserWarning: Found `num_iteratio
ns` in params. Will use it instead of argument
  warnings.warn("Found `{}` in params. Will use it instead of argume
nt".format(alias))
/anaconda3/envs/home_credit_default_risk_competition/lib/python3.5/s
ite-packages/lightgbm/engine.py:99: UserWarning: Found `num_iteratio
ns` in params. Will use it instead of argument
  warnings.warn("Found `{}` in params. Will use it instead of argume
nt".format(alias))
/anaconda3/envs/home_credit_default_risk_competition/lib/python3.5/s
ite-packages/lightgbm/engine.py:99: UserWarning: Found `num_iteratio
ns` in params. Will use it instead of argument
  warnings.warn("Found `{}` in params. Will use it instead of argume
nt".format(alias))
/anaconda3/envs/home_credit_default_risk_competition/lib/python3.5/s
ite-packages/lightgbm/engine.py:99: UserWarning: Found `num_iteratio
ns` in params. Will use it instead of argument
  warnings.warn("Found `{}` in params. Will use it instead of argume
nt".format(alias))
/anaconda3/envs/home_credit_default_risk_competition/lib/python3.5/s
ite-packages/lightgbm/engine.py:99: UserWarning: Found `num_iteratio
ns` in params. Will use it instead of argument
  warnings.warn("Found `{}` in params. Will use it instead of argume
nt".format(alias))
/anaconda3/envs/home_credit_default_risk_competition/lib/python3.5/s
ite-packages/lightgbm/engine.py:99: UserWarning: Found `num_iteratio
ns` in params. Will use it instead of argument
  warnings.warn("Found `{}` in params. Will use it instead of argume
nt".format(alias))
/anaconda3/envs/home_credit_default_risk_competition/lib/python3.5/s
ite-packages/lightgbm/engine.py:99: UserWarning: Found `num_iteratio
ns` in params. Will use it instead of argument
  warnings.warn("Found `{}` in params. Will use it instead of argume
nt".format(alias))
/anaconda3/envs/home_credit_default_risk_competition/lib/python3.5/s
```

```
ite-packages/lightgbm/engine.py:99: UserWarning: Found `num_iteratio
ns` in params. Will use it instead of argument
  warnings.warn("Found `{}` in params. Will use it instead of argume
nt".format(alias))
/anaconda3/envs/home_credit_default_risk_competition/lib/python3.5/s
ite-packages/lightgbm/engine.py:99: UserWarning: Found `num_iteratio
ns` in params. Will use it instead of argument
  warnings.warn("Found `{}` in params. Will use it instead of argume
nt".format(alias))
/anaconda3/envs/home_credit_default_risk_competition/lib/python3.5/s
ite-packages/lightgbm/engine.py:99: UserWarning: Found `num_iteratio
ns` in params. Will use it instead of argument
  warnings.warn("Found `{}` in params. Will use it instead of argume
nt".format(alias))
/anaconda3/envs/home_credit_default_risk_competition/lib/python3.5/s
ite-packages/lightgbm/engine.py:99: UserWarning: Found `num_iteratio
ns` in params. Will use it instead of argument
  warnings.warn("Found `{}` in params. Will use it instead of argume
nt".format(alias))
/anaconda3/envs/home_credit_default_risk_competition/lib/python3.5/s
ite-packages/lightgbm/engine.py:99: UserWarning: Found `num_iteratio
ns` in params. Will use it instead of argument
  warnings.warn("Found `{}` in params. Will use it instead of argume
nt".format(alias))
/anaconda3/envs/home_credit_default_risk_competition/lib/python3.5/s
ite-packages/lightgbm/engine.py:99: UserWarning: Found `num_iteratio
ns` in params. Will use it instead of argument
  warnings.warn("Found `{}` in params. Will use it instead of argume
nt".format(alias))
/anaconda3/envs/home_credit_default_risk_competition/lib/python3.5/s
ite-packages/lightgbm/engine.py:99: UserWarning: Found `num_iteratio
ns` in params. Will use it instead of argument
  warnings.warn("Found `{}` in params. Will use it instead of argume
nt".format(alias))
/anaconda3/envs/home_credit_default_risk_competition/lib/python3.5/s
ite-packages/lightgbm/engine.py:99: UserWarning: Found `num_iteratio
ns` in params. Will use it instead of argument
  warnings.warn("Found `{}` in params. Will use it instead of argume
nt".format(alias))
/anaconda3/envs/home_credit_default_risk_competition/lib/python3.5/s
ite-packages/lightgbm/engine.py:99: UserWarning: Found `num_iteratio
ns` in params. Will use it instead of argument
  warnings.warn("Found `{}` in params. Will use it instead of argume
nt".format(alias))
/anaconda3/envs/home_credit_default_risk_competition/lib/python3.5/s
ite-packages/lightgbm/engine.py:99: UserWarning: Found `num_iteratio
ns` in params. Will use it instead of argument
  warnings.warn("Found `{}` in params. Will use it instead of argume
nt".format(alias))
/anaconda3/envs/home_credit_default_risk_competition/lib/python3.5/s
ite-packages/lightgbm/engine.py:99: UserWarning: Found `num_iteratio
ns` in params. Will use it instead of argument
  warnings.warn("Found `{}` in params. Will use it instead of argume
```

```
nt".format(alias))
/anaconda3/envs/home_credit_default_risk_competition/lib/python3.5/s
ite-packages/lightgbm/engine.py:99: UserWarning: Found `num_iteratio
ns` in params. Will use it instead of argument
  warnings.warn("Found `{}` in params. Will use it instead of argume
nt".format(alias))
/anaconda3/envs/home_credit_default_risk_competition/lib/python3.5/s
ite-packages/lightgbm/engine.py:99: UserWarning: Found `num_iteratio
ns` in params. Will use it instead of argument
  warnings.warn("Found `{}` in params. Will use it instead of argume
nt".format(alias))
/anaconda3/envs/home_credit_default_risk_competition/lib/python3.5/s
ite-packages/lightgbm/engine.py:99: UserWarning: Found `num_iteratio
ns` in params. Will use it instead of argument
  warnings.warn("Found `{}` in params. Will use it instead of argume
nt".format(alias))
/anaconda3/envs/home_credit_default_risk_competition/lib/python3.5/s
ite-packages/lightgbm/engine.py:99: UserWarning: Found `num_iteratio
ns` in params. Will use it instead of argument
  warnings.warn("Found `{}` in params. Will use it instead of argume
nt".format(alias))
/anaconda3/envs/home_credit_default_risk_competition/lib/python3.5/s
ite-packages/lightgbm/engine.py:99: UserWarning: Found `num_iteratio
ns` in params. Will use it instead of argument
  warnings.warn("Found `{}` in params. Will use it instead of argume
nt".format(alias))
/anaconda3/envs/home_credit_default_risk_competition/lib/python3.5/s
ite-packages/lightgbm/engine.py:99: UserWarning: Found `num_iteratio
ns` in params. Will use it instead of argument
  warnings.warn("Found `{}` in params. Will use it instead of argume
nt".format(alias))

Best hyperparameter combo:
{'num_leaves': 200, 'num_iterations': 10000, 'learning_rate': 0.1, '
sub_feature': 0.3, 'lambda_l2': 0.1, 'max_depth': -1, 'min_data_in_l
eaf': 40}
ROC AUC score of best hyperparameter combo:
0.723550734163
```

In [658]:

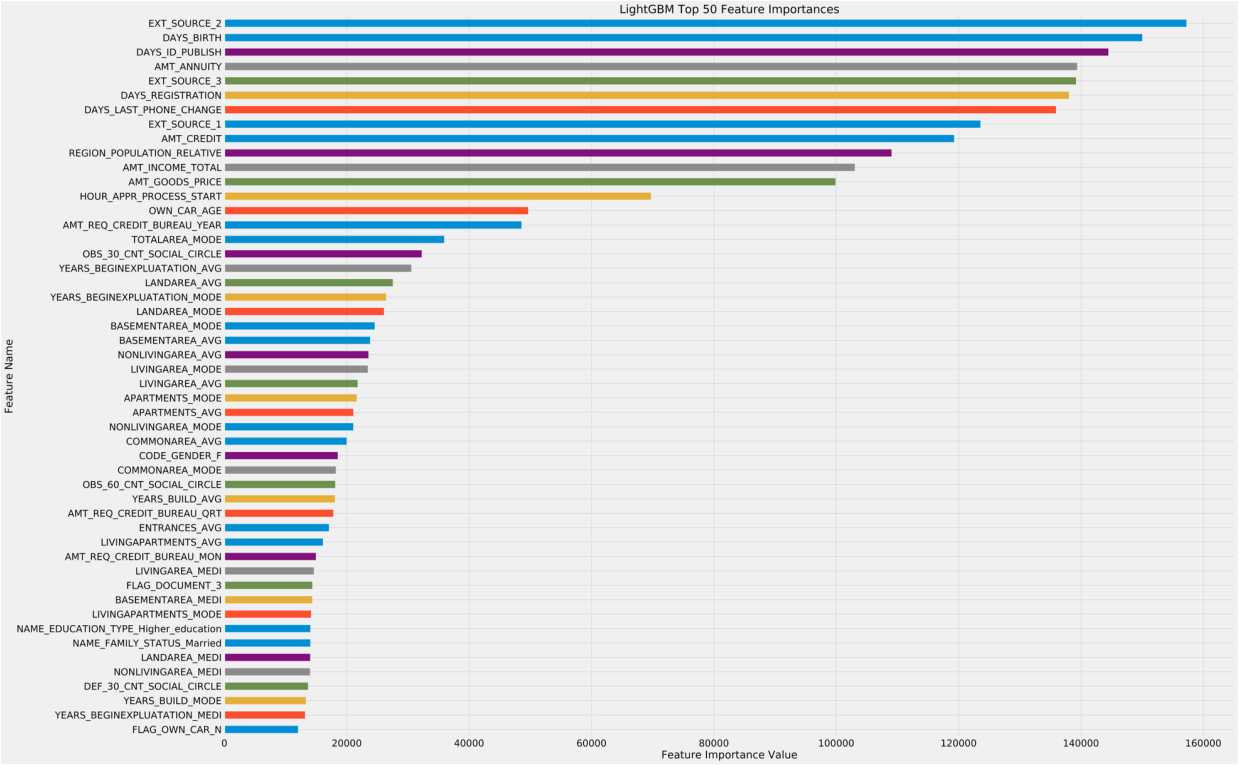
```
print(grid.predict_proba(X_test_final)[:,1])
```

```
[  8.54037372e-03   2.34626595e-06   1.15907921e-05 ...,   1.4477113
5e-06
   2.02137025e-05   4.02830947e-05]
```

In [655]:
```
# Use the LightGBM classifier with parameters discovered in GridSearch
CV to
# make predictions on the test validation set. Calculate the area unde
r ROC
# curve score of these predictions.

# After running GridSearchCV above, I observed that:
# 1.

# Convert dataframes to LGB format
lgb_training = lgb.Dataset(X_train_final, y_train)

# Final parameters for LightGBM training
params = {}
params['learning_rate'] = 0.001
params['boosting_type'] = 'gbdt'
params['objective'] = 'binary'
params['metric'] = 'auc'
params['num_leaves'] = 200
params['max_depth'] = 20
params['max_bin'] = 110
params['lambda_l2'] = 0.1
params['bagging_freq'] = 1
params['bagging_fraction'] = 0.95
params['bagging_seed'] = 1
params['feature_fraction'] = 0.9
params['feature_fraction_seed'] = 1
params['random_state'] = 42

# Fit the LightGBM classifier to the training data
clf_lgb = lgb.train(params, lgb_training, 15000)

# Classifier's estimates of probability of the positive class (TARGET=
1): the
# probability estimate of each borrower making at least one late loan
payment.
lgb_tuned_y_score = clf_lgb.predict(X_test_final)

# The area under the ROC curve between the true target values and the
# probability estimates of the predicted values.
lgb_tuned_roc_auc_score = roc_auc_scorer(y_test, lgb_tuned_y_score)

# Add the LightGBM classifier's scores to the results list.
#y_score_list.append(lgb_tuned_y_score)
#clf_label_list.append('LightGBM All Features, Further Tuning')

print('LightGBM (All Features, Further Tuning) test validation set pre
dictions\' ROC AUC score: {}'.format(lgb_tuned_roc_auc_score))
```

```
LightGBM (All Features, Further Tuning) test validation set predicti
ons' ROC AUC score: 0.7609160310721934
```

In [698]:
```
# Plot the 50 largest LightGBM feature importances:
plt.figure(figsize = (34,26), dpi=300)
feat_importances = pd.Series(clf_lgb.feature_importance(importance_type='split', iteration=-1), clf_lgb.feature_name())
feat_importances = feat_importances.nlargest(50).sort_values(axis='index', ascending = True)
feat_importances.plot(kind='barh')
plt.title('LightGBM Top 50 Feature Importances', fontsize=24)
plt.xlabel('Feature Importance Value', fontsize=22)
plt.ylabel('Feature Name', fontsize=22)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.savefig('lightGBMFeatureImportances.png')
plt.show()
```



VII. Results

ROC AUC scores of various classifiers

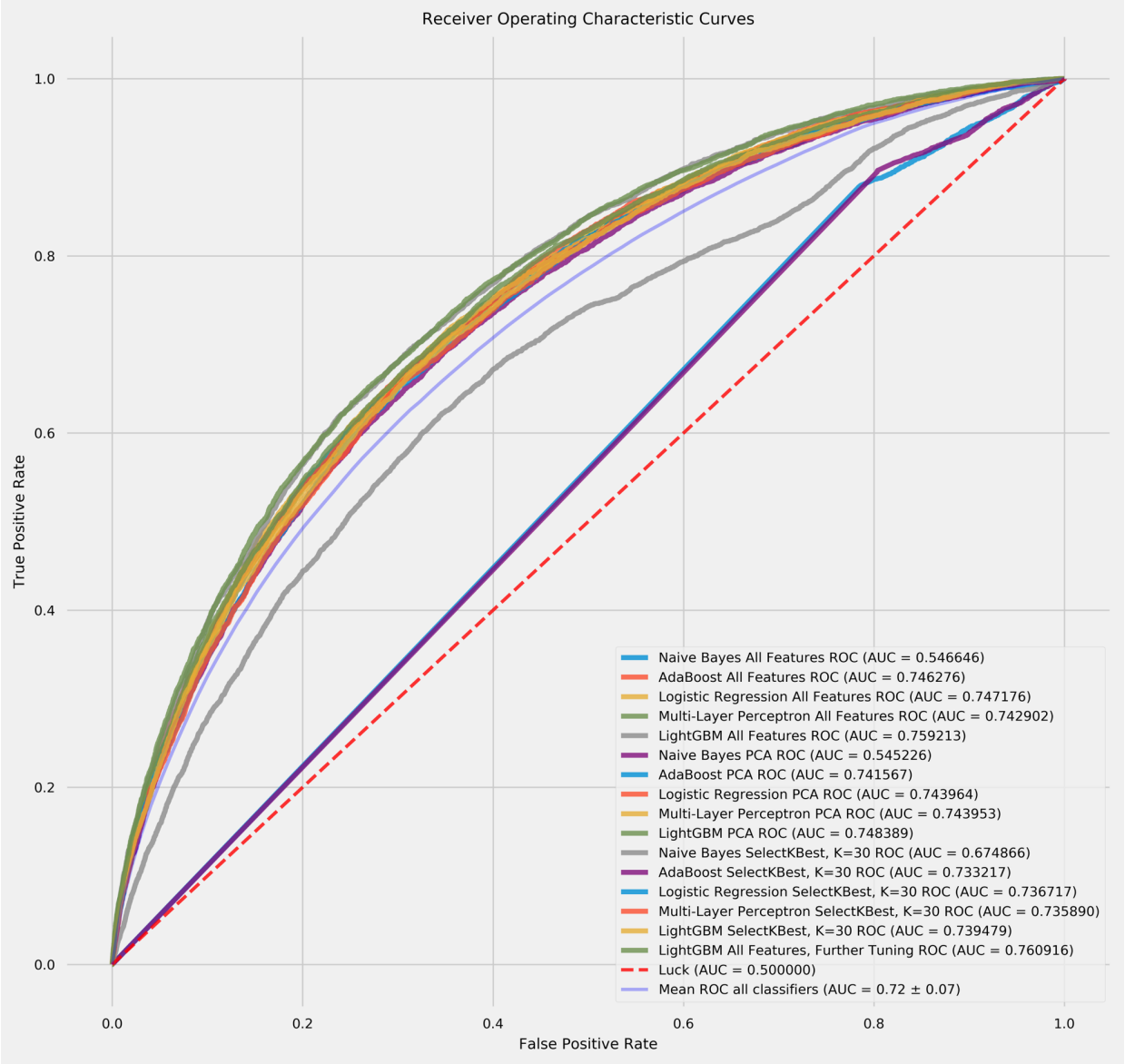
Unless specifically noted otherwise, each classifier was trained on the 184 binary features and 17 PCA reduced numerical features that comprised all 201 features from the original preprocessed training set.

Certain LightGBM classifiers were trained on various different feature subsets/supersets of this original training set. These classifiers are indicated in the table and plot that follow.

ROC AUC scores indicate performance of classifier probability predictions made for the labels of the test validation set, which is 20% of the size (rows) of the training set contained in the application_train.csv data table:

Classifier Name	**ROC AUC Score**
Naive Bayes (All Features)	0.546645662333944
AdaBoost (All Features)	0.7462758964509755
Logistic Regression (All Features)	0.7471756350178691
Multi-Layer Perceptron (All Features)	0.7429017839300756
LightGBM (All Features)	0.7592132612569703
Naive Bayes (PCA)	0.5452255614331999
AdaBoost (PCA)	0.7415669749755673
Logistic Regression (PCA)	0.743963963781135
Multi-Layer Perceptron (PCA)	0.7439527449175637
LightGBM (PCA)	0.7483887050110797
Naive Bayes (SelectKBest Features, K=30)	0.6748662184461512
AdaBoost (SelectKBest Features, K=30)	0.7330739254581403
Logistic Regression (SelectKBest Features, K=30)	0.7367180213600446
Multi-Layer Perceptron (SelectKBest Features, K=30)	0.7358901049573279
LightGBM (SelectKBest Features, K=30)	0.7394787228642934
LightGBM (All Features, Further Tuning)	0.7609160310721934

In [650]:
```
# Display ROC curves of the Naive Bayes, AdaBoost, Logistic Regression
, and
# Multi-Layer Perceptron classifiers' probability predictions.
vs.plot_roc_curves(y_test, y_score_list, clf_label_list, title='Receiv
er Operating Characteristic Curves');
```



VIII. Submitting to Kaggle

In [651]:
```
# Train the final LightGBM classifier on the entire training set

# Load the main data tables
application_train_data = pd.read_csv("data/application_train.csv")
application_test_data = pd.read_csv("data/application_test.csv")

# Load the Bureau data table
bureau_data = pd.read_csv("data/bureau.csv")
```

```
# 1. Create lists of different feature types in the main data
# frame, based on how each type will need to be preprocessed.

# i. All 18 categorical features needing one-hot encoding.
#    Includes the 4 categorical features originally
#    mis-identified as having been normalized:
#    EMERGENCYSTATE_MODE, HOUSETYPE_MODE, WALLSMATERIAL_MODE,
#    FONDKAPREMONT_MODE
cat_feat_need_one_hot = [
    'NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR',
    'FLAG_OWN_REALTY', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE',
    'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'REGION_RATING_CLIENT',
    'REGION_RATING_CLIENT_W_CITY', 'WEEKDAY_APPR_PROCESS_START', 'ORGA
NIZATION_TYPE',
    'NAME_TYPE_SUITE', 'OCCUPATION_TYPE', 'EMERGENCYSTATE_MODE',
    'HOUSETYPE_MODE', 'WALLSMATERIAL_MODE', 'FONDKAPREMONT_MODE'
]

# ii. All 32 binary categorical features already one-hot encoded.
bin_cat_feat = [
    'FLAG_MOBIL', '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'
]

# iii. All 2 non-normalized numerical features with skewed distributio
ns
#    and negative values. These features will need to have their
#    distributions translated to positive ranges before being
#    log-transformed, and then later scaled to the range [0,1].
non_norm_feat_neg_values_skewed = [
    'DAYS_REGISTRATION', 'DAYS_LAST_PHONE_CHANGE'
]


# iv. All 15 non-normalized numerical features with skewed distributio
ns,
#    and only positive values. These features will need to be
#    log-transformed, and eventually scaled to the range [0,1].
non_norm_feat_pos_values_skewed = [
    'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY',
    'AMT_GOODS_PRICE', 'OBS_30_CNT_SOCIAL_CIRCLE', 'DEF_30_CNT_SOCIAL_
CIRCLE',
```

```
    'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIRCLE', 'AMT_REQ_C
REDIT_BUREAU_HOUR',
    'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK', 'AMT_RE
Q_CREDIT_BUREAU_MON',
    'AMT_REQ_CREDIT_BUREAU_QRT', 'AMT_REQ_CREDIT_BUREAU_YEAR', 'OWN_CA
R_AGE'
]

# v. All 4 numerical features with normal shapes but needing to be sca
led
#    to the range [0,1].
norm_feat_need_scaling = [
    'DAYS_BIRTH', 'DAYS_ID_PUBLISH', 'HOUR_APPR_PROCESS_START',
    'REGION_POPULATION_RELATIVE'
]

# vi. All 46 numerical features that have been normalized to the range
#   [0,1]. These features will need neither log-transformation, nor
#   any further scaling.
norm_feat_not_need_scaling = [
    'EXT_SOURCE_2', 'EXT_SOURCE_3', 'YEARS_BEGINEXPLUATATION_AVG',
    'YEARS_BEGINEXPLUATATION_MODE', 'YEARS_BEGINEXPLUATATION_MEDI', 'F
LOORSMAX_AVG',
    'FLOORSMAX_MODE', 'FLOORSMAX_MEDI', 'LIVINGAREA_AVG',
    'LIVINGAREA_MODE', 'LIVINGAREA_MEDI', 'ENTRANCES_AVG',
    'ENTRANCES_MODE', 'ENTRANCES_MEDI', 'APARTMENTS_AVG',
    'APARTMENTS_MODE', 'APARTMENTS_MEDI', 'ELEVATORS_AVG',
    'ELEVATORS_MODE', 'ELEVATORS_MEDI', 'NONLIVINGAREA_AVG',
    'NONLIVINGAREA_MODE', 'NONLIVINGAREA_MEDI', 'EXT_SOURCE_1',
    'BASEMENTAREA_AVG', 'BASEMENTAREA_MODE', 'BASEMENTAREA_MEDI',
    'LANDAREA_AVG', 'LANDAREA_MODE', 'LANDAREA_MEDI',
    'YEARS_BUILD_AVG', 'YEARS_BUILD_MODE', 'YEARS_BUILD_MEDI',
    'FLOORSMIN_AVG', 'FLOORSMIN_MODE', 'FLOORSMIN_MEDI',
    'LIVINGAPARTMENTS_AVG', 'LIVINGAPARTMENTS_MODE', 'LIVINGAPARTMENTS
_MEDI',
    'NONLIVINGAPARTMENTS_AVG', 'NONLIVINGAPARTMENTS_MODE', 'NONLIVINGA
PARTMENTS_MEDI',
    'COMMONAREA_AVG', 'COMMONAREA_MODE', 'COMMONAREA_MEDI',
    'TOTALAREA_MODE'
]

# vii. The remaining 3 features in the main data frame that will be
#    re-engineered and transformed into different features
feat_to_be_reengineered = [
    'CNT_CHILDREN', 'CNT_FAM_MEMBERS', 'DAYS_EMPLOYED'
]

# 2. Separate target data from training dataset.
targets = application_train_data['TARGET']
features_raw = application_train_data.drop('TARGET', axis = 1)

# 3. Because the entire training set from the file application_train.c
sv
```

```
# is being used for training, there is no need at this point to do a
# train test validation split.
y_train = targets
X_train_raw = features_raw

# 4. Use the CNT_CHILDREN feature to engineer a binary
# categorical feature called HAS_CHILDREN. If value of CNT_CHILDREN is
# greater than 0, the value of HAS_CHILDREN will be 1. If value of CNT
_CHILDREN is
# 0, value of HAS_CHILDREN will be 0.
CNT_CHILDREN_train = X_train_raw['CNT_CHILDREN']
HAS_CHILDREN = CNT_CHILDREN_train.map(lambda x: 1 if x > 0 else 0)

# Append the newly engineered HAS_CHILDREN feature to the main datafra
me.
X_train_raw = X_train_raw.assign(HAS_CHILDREN=HAS_CHILDREN.values)

# 5. Drop the CNT_CHILDREN column from the main dataframe
X_train_raw = X_train_raw.drop('CNT_CHILDREN', axis=1)

# Add the new HAS_CHILDREN feature to the list of binary categorical
# features that are already one-hot encoded. There are now 33 such fea
tures.
bin_cat_feat = bin_cat_feat + ['HAS_CHILDREN']

# 6. Use the CNT_FAM_MEMBERS feature to engineer a categorical feature
called NUMBER_FAMILY_MEMBERS.
# If CNT_FAM_MEMBERS is 1.0, then the value of NUMBER_FAMILY_MEMBERS w
ill be 'one'. If CNT_FAM_MEMBERS is 2.0,
# then NUMBER_FAMILY_MEMBERS will be 'two'. If CNT_FAM_MEMBERS is 3.0
or greater, then NUMBER_FAMILY_MEMBERS will
# be 'three_plus'.
CNT_FAM_MEMBERS_train = X_train_raw['CNT_FAM_MEMBERS']
NUMBER_FAMILY_MEMBERS = CNT_FAM_MEMBERS_train.map(lambda x: 'one' if x
== 1 else ('two' if x == 2 else 'three_plus'))

# Append the newly engineered FAMILY_SIZE feature to the main datafram
e.
X_train_raw = X_train_raw.assign(NUMBER_FAMILY_MEMBERS=NUMBER_FAMILY_M
EMBERS.values)

# 7. Drop the CNT_FAM_MEMBERS feature from the main dataframe
X_train_raw = X_train_raw.drop('CNT_FAM_MEMBERS', axis=1)

# Add the new NUMBER_FAMILY_MEMBERS feature to the list of categorical
# features that will need to be one-hot encoded. There are now 19 of t
hese features.
cat_feat_need_one_hot = cat_feat_need_one_hot + ['NUMBER_FAMILY_MEMBER
S']

# 8. Use the CREDIT_DAY_OVERDUE feature in bureau.csv to engineer the
binary
# categorical HAS_CREDIT_BUREAU_LOANS_OVERDUE feature. If CREDIT_DAY_O
```

```
VERDUE for a
# particular borrower ID (SK_ID_CURR) is greater than 0, then the valu
e of
# HAS_CREDIT_BUREAU_LOANS_OVERDUE will be 1. If CREDIT_DAY_OVERDUE for
a particular
# borrower ID is 0, then the value of HAS_CREDIT_BUREAU_LOANS_OVERDUE
will be 0.

# Filter the bureau data table for loans which are overdue (have a val
ue
# for CREDIT_DAY_OVERDUE that's greater than 0)
bureau_data_filtered_for_overdue = bureau_data[bureau_data['CREDIT_DAY
_OVERDUE'] > 0]

def build_feature_HAS_CREDIT_BUREAU_LOANS_OVERDUE(dataframe):
    """
    Use the CREDIT_DAY_OVERDUE feature in bureau.csv to engineer the b
inary
    categorical HAS_CREDIT_BUREAU_LOANS_OVERDUE feature. If CREDIT_DAY
_OVERDUE for a
    particular borrower ID (SK_ID_CURR) is greater than 0, then the va
lue of
    HAS_CREDIT_BUREAU_LOANS_OVERDUE will be 1. If CREDIT_DAY_OVERDUE f
or a particular
    borrower ID is 0, then the value of HAS_CREDIT_BUREAU_LOANS_OVERDU
E will be 0.

    Parameters:
        dataframe: Pandas dataframe containing a training or testing d
ataset

    Returns: The dataframe with HAS_CREDIT_BUREAU_LOANS_OVERDUE featur
e appended to it.
    """
    # Create a series called HAS_CREDIT_BUREAU_LOANS_OVERDUE and fill
it with zeros.
    # Its index is identical to that of the main dataframe. It will ev
entually be appended
    # to the main data frame as a column.
    HAS_CREDIT_BUREAU_LOANS_OVERDUE = pd.Series(data=0, index = datafr
ame['SK_ID_CURR'].index)

    # A list of all the borrowers IDs in the main dataframe
    main_data_table_borrower_IDs = dataframe['SK_ID_CURR'].values

    # For each loan in the bureau data table that is overdue
    # (has a value for CREDIT_DAY_OVERDUE that's greater than 0)
    for index, row in bureau_data_filtered_for_overdue.iterrows():
        # The borrower ID (SK_ID_CURR) that owns the overdue loan
        borrower_ID = row['SK_ID_CURR']
        # If the borrower ID owning the overdue loan is also
        # in the main data frame, then enter a value of 1 in
        # the series HAS_CREDIT_BUREAU_LOANS_OVERDUE at an index
```

```
        # that is identical to the index of the borrower ID
        # in the main data frame.
        if borrower_ID in main_data_table_borrower_IDs:
            # The index of the borrower's row in the main data table.
            borrower_index_main_data_table = dataframe.index[dataframe
['SK_ID_CURR'] == borrower_ID].tolist()[0]
            # Place a value of 1 at the index of the series HAS_CREDIT
_BUREAU_LOANS_OVERDUE
            # which corresponds to the index of the borrower's ID in t
he main data table.
            HAS_CREDIT_BUREAU_LOANS_OVERDUE.loc[borrower_index_main_da
ta_table] = 1
    # Append the newly engineered HAS_CREDIT_BUREAU_LOANS_OVERDUE feat
ure to the main dataframe.
    dataframe = dataframe.assign(HAS_CREDIT_BUREAU_LOANS_OVERDUE=HAS_C
REDIT_BUREAU_LOANS_OVERDUE.values)
    return dataframe

# Build the HAS_CREDIT_BUREAU_LOANS_OVERDUE feature
X_train_raw = build_feature_HAS_CREDIT_BUREAU_LOANS_OVERDUE(X_train_ra
w)

# Add the new HAS_CREDIT_BUREAU_LOANS_OVERDUE feature to the list of b
inary categorical
# features. There are now 34 of these features.
bin_cat_feat = bin_cat_feat + ['HAS_CREDIT_BUREAU_LOANS_OVERDUE']

# 9. Use the DAYS_EMPLOYED feature to engineer a binary categorical fe
ature called HAS_JOB.
# If the value of DAYS_EMPLOYED is 0 or less, then HAS_JOB will be 1.
Otherwise, HAS_JOB will
# be 0. This condition will apply to all borrowers who had a value of
365243 for DAYS_EMPLOYED,
# which I hypothesized can be best interpreted as meaning that a borro
wer does not have a job.
DAYS_EMPLOYED_train = X_train_raw['DAYS_EMPLOYED']
HAS_JOB = DAYS_EMPLOYED_train.map(lambda x: 1 if x <= 0 else 0)

# Append the newly engineered FAMILY_SIZE feature to the main datafram
e.
X_train_raw = X_train_raw.assign(HAS_JOB=HAS_JOB.values)

# 10. Drop the DAYS_EMPLOYED feature from the main dataframe
X_train_raw = X_train_raw.drop('DAYS_EMPLOYED', axis=1)

# Add the new HAS_JOB feature to the list of binary categorical featur
es.
# There are now 35 of these features.
bin_cat_feat = bin_cat_feat + ['HAS_JOB']

# 11. Translate the 2 non-normalized numerical features that have skew
ed distributions
# and negative values: DAYS_REGISTRATION, and DAYS_LAST_PHONE_CHANGE
```

```
def translate_negative_valued_features(dataframe, feature_name_list):
    """
    Translate a dataset's continuous features containing several negat
ive
    values. The dataframe is modified such that all values of each fea
ture
    listed in the feature_name_list parameter become positive.

    Parameters:
        dataframe: Pandas dataframe containing the features
        feature_name_list: List of strings, containing the names
                           of each feature whose values will be
                           translated
    """
    for feature in feature_name_list:
        # The minimum, most-negative, value of the feature
        feature_min_value = dataframe[feature].min()
        # Translate each value of the feature in a positive direction,
        # of magnitude that's equal to the feature's most negative val
ue.
        dataframe[feature] = dataframe[feature].apply(lambda x: x - fe
ature_min_value)

# Translate the above two negatively-valued features to positive value
s
translate_negative_valued_features(X_train_raw, non_norm_feat_neg_valu
es_skewed)

# 12. Log-transform all 17 non-normalized numerical features that have
skewed distributions.
# These 17 features include the 2 that were translated to positive ran
ges in Step 11.

# Add the 2 features translated to positive ranges above in Step 11 to
# the list of non-normalized skewed features with positive values. Thi
s is
# the set of features that will be log-transformed
log_transform_feats = non_norm_feat_pos_values_skewed + non_norm_feat_
neg_values_skewed

X_train_raw[log_transform_feats] = X_train_raw[log_transform_feats].ap
ply(lambda x: np.log(x + 1))

# 13. Replace 'NaN' values for all numerical features with each featur
e's mean. Fit an imputer
# to each numerical feature containing at least one 'NaN' entry.

# Create a list of all the 67 numerical features in the main dataframe
. These include all
# 17 features that were log-transformed in Step 12, as well as the 4 n
ormal features that
# still need to be scaled, as well as the 46 normal features that don'
```

```
t need scaling.
numerical_features = log_transform_feats + norm_feat_need_scaling + no
rm_feat_not_need_scaling

# Create a list of all numerical features in the training set that hav
e at least one 'NaN' entry
numerical_features_with_nan = X_train_raw[numerical_features].columns[
X_train_raw[numerical_features].isna().any()].tolist()

# Create an imputer
imputer = Imputer()
# Fit the imputer to each numerical feature in the training set that h
as 'NaN' values,
# and replace each 'NaN' entry of each feature with that feature's mea
n.
X_train_raw[numerical_features_with_nan] = imputer.fit_transform(X_tra
in_raw[numerical_features_with_nan])

# 14. Remove the borrower ID column, SK_ID_CURR, from the main datafra
me
X_train_raw = X_train_raw.drop('SK_ID_CURR', axis=1)

# 15. One-hot encode all 19 non-binary categorical features.
X_train_raw = pd.get_dummies(X_train_raw, columns=cat_feat_need_one_ho
t)

# Create a list that includes only the newly one-hot encoded features
# as well as all the categorical features that were already binary.
all_bin_cat_feat = X_train_raw.columns.tolist()
for column_name in X_train_raw[numerical_features].columns.tolist():
    all_bin_cat_feat.remove(column_name)

# 16. Replace all 'NaN' values in all binary categorical features with
0.

# Create a list of binary categorical features with at least one 'NaN'
entry
bin_cat_feat_with_nan = X_train_raw[all_bin_cat_feat].columns[X_train_
raw[all_bin_cat_feat].isna().any()].tolist()

# Replace each 'NaN' value in each of these binary features with 0
X_train_raw[bin_cat_feat_with_nan] = X_train_raw[bin_cat_feat_with_nan
].fillna(value=0)

# 17. Fit a min-max scaler to each of the 17 log-transformed numerical
features, as well
# as to the 4 features DAYS_BIRTH, DAYS_ID_PUBLISH, HOUR_APPR_PROCESS_
START, and the normalized
# feature REGION_POPULATION_RELATIVE. Each feature will be scaled to a
range [0.0, 1.0].

# Build a list of all 21 features needing scaling. Add the list of fea
tures that
```

```
# were log-normalized above in Step 12 to the list of normally shaped features
# that need to be scaled to the range [0,1].
feats_to_scale = norm_feat_need_scaling + log_transform_feats

# Initialize a scaler with the default range of [0,1]
scaler = MinMaxScaler()

# Fit the scaler to each of the features of the train set that need to be scaled,
# then transform each of these features' values to the new scale.
X_train_raw[feats_to_scale] = scaler.fit_transform(X_train_raw[feats_to_scale])

# Rename the dataframe to indicate that its columns have been fully preprocessed.
X_train_final = X_train_raw

print('Entire training dataset preprocessing complete.')
print('Number of columns: {}. Expected: 251.'.format(X_train_final.shape[1]))
print('Number of rows: {}. Expected: 307511.'.format(X_train_final.shape[0]))
print('Number of labels: {}. Expected: 307511.'.format(y_train.shape[0]))
```

```
Entire training dataset preprocessing complete.
Number of columns: 251. Expected: 251.
Number of rows: 307511. Expected: 307511.
Number of labels: 307511. Expected: 307511.
```

In [652]:
```
# Fit a LightGBM classifier to the entire training set using the param
eters
# that were tuned in the final refinement step above.

# Convert the entire training dataframe to LGB format
lgb_training = lgb.Dataset(X_train_final, y_train)

# Final parameters for LightGBM training
params = {}
params['learning_rate'] = 0.001
params['boosting_type'] = 'gbdt'
params['objective'] = 'binary'
params['metric'] = 'auc'
params['num_leaves'] = 200
params['max_depth'] = 20
params['max_bin'] = 110
params['lambda_l2'] = 0.1
params['bagging_freq'] = 1
params['bagging_fraction'] = 0.95
params['bagging_seed'] = 1
params['feature_fraction'] = 0.9
params['feature_fraction_seed'] = 1
params['random_state'] = 42

# Fit the LightGBM classifier to the training data
clf_lgb = lgb.train(params, lgb_training, 15000)
```

In [653]:
```
# Build a prediction pipeline for the testing data table (application_
test.csv) that
# saves prediction probabilities to a CSV file, which will then be sub
mitted on Kaggle.

def testing_data_table_predictions_to_csv(clf, testing_data_table, isL
ightGBM):
    """
    A prediction pipeline that:
    1. Preprocesses the 48,744 row testing data table
    2. Uses a classifier to compute estimates of the probability of th
e positive
       class (TARGET=1) for each borrower: the probability estimate of
each borrower
       making at least one late loan payment.
    3. Saves a CSV file that contains probabilities of target labels f
or each
       borrower (SK_ID_CURR) in the testing data table.
    4. isLightGBM: Boolean, a flag that indicates whether or not the c
lassifier is
                   LightGBM. If True,

    Parameters:
        clf: A machine learning classifier object that has already bee
n fit to
```

```
                    the training data.
        testing_data_table: Pandas dataframe containing the testing da
taset.
    """

    # Get a list of the borrower IDs (SK_ID_CURR column). The borrower
ID must be
    # placed in each row of CSV file that will be created.
    borrower_IDs = testing_data_table['SK_ID_CURR']

    # Preprocess the testing data table so that predictions can be mad
e on it.
    X_test_final = test_set_preprocessing_pipeline(testing_data_table)
    #print('application_test.csv testing set processing complete. The
processed dataframe now has {} columns. Expected: 251.'.format(X_test_
final.shape[1]))

    # Classifier's estimates of probability of the positive class (TAR
GET=1): the
    # probability estimate of each borrower making at least one late l
oan payment.
    # If classifier is LightGBM, the method for making predictions is
merely 'predict'
    # and the arrray containing these probabilities has slightly diffe
rent shape than
    # those produced by the other classifiers.
    if isLightGBM:
        clf_y_score = clf.predict(X_test_final)
    else:
        clf_y_score = clf.predict_proba(X_test_final)[:, 1]

    # Create the CSV file that will be saved
    file_output = 'dellinger_kaggle_home_credit_submission5.csv'
    # Write to the CSV file
    with open(file_output, 'w') as csvfile:
        writer = csv.writer(csvfile)
        # Write the header row
        writer.writerow(['SK_ID_CURR','TARGET'])
        # Write a row for each borrower that contains the
        # prediction probability of their label.
        for index, value in borrower_IDs.iteritems():
            writer.writerow([value, clf_y_score[index]])

# To submit to Kaggle: the LightGBM Classifier's predictions on full f
eatureset.
# Create predictions on the data in the testing data table (applicatio
n_test.csv)
# using the LightGBM classifier fit above. Also create a CSV
# file containing the prediction probabilities for each borrower ID (S
K_ID_CURR)
# in the testing data table.
testing_data_table_predictions_to_csv(clf_lgb, application_test_data,
True)
```