In [1]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os

from sklearn import svm, tree, linear_model, neighbors, naive_bayes, ensemble, discriminant_analysis, g
aussian_process
import tensorflow as tf
from tensorflow import keras

from sklearn import metrics
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import ShuffleSplit
from sklearn import model_selection

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
```

In [3]:
```python
os.getcwd()
```

Out[3]:
```
'C:\\Users\\robtu'
```

In [2]:
```python
os.chdir(r"C:\Users\robtu\Kaggle Competitions")
```

In [3]:
```python
train_df = pd.read_csv("train.csv")
test_df = pd.read_csv("test.csv")
train_df.head()
```

Out[3]:

	id	cat0	cat1	cat2	cat3	cat4	cat5	cat6	cat7	cat8	...	cont2	cont3	cont4	cont5	cont6	cont7	cont8
0	0	A	I	A	B	B	BI	A	S	Q	...	0.759439	0.795549	0.681917	0.621672	0.592184	0.791921	0.815254
1	1	A	I	A	A	E	BI	K	W	AD	...	0.386385	0.541366	0.388982	0.357778	0.600044	0.408701	0.399353
2	2	A	K	A	A	E	BI	A	E	BM	...	0.343255	0.616352	0.793687	0.552877	0.352113	0.388835	0.412303
3	3	A	K	A	C	E	BI	A	Y	AD	...	0.831147	0.807807	0.800032	0.619147	0.221789	0.897617	0.633669
4	4	A	I	G	B	E	BI	C	G	Q	...	0.338818	0.277308	0.610578	0.128291	0.578764	0.279167	0.351103

5 rows × 32 columns

In [4]:
```python
df = pd.concat([train_df.drop(columns=['target']), test_df], ignore_index = True)
y_train = train_df['target']
test_ids = test_df['id']
```

Data Exploration

In [5]:
```python
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300000 entries, 0 to 299999
Data columns (total 32 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   id      300000 non-null  int64
 1   cat0    300000 non-null  object
 2   cat1    300000 non-null  object
 3   cat2    300000 non-null  object
 4   cat3    300000 non-null  object
 5   cat4    300000 non-null  object
 6   cat5    300000 non-null  object
 7   cat6    300000 non-null  object
 8   cat7    300000 non-null  object
 9   cat8    300000 non-null  object
 10  cat9    300000 non-null  object
 11  cat10   300000 non-null  object
 12  cat11   300000 non-null  object
 13  cat12   300000 non-null  object
 14  cat13   300000 non-null  object
 15  cat14   300000 non-null  object
 16  cat15   300000 non-null  object
 17  cat16   300000 non-null  object
 18  cat17   300000 non-null  object
 19  cat18   300000 non-null  object
 20  cont0   300000 non-null  float64
 21  cont1   300000 non-null  float64
 22  cont2   300000 non-null  float64
 23  cont3   300000 non-null  float64
 24  cont4   300000 non-null  float64
 25  cont5   300000 non-null  float64
 26  cont6   300000 non-null  float64
 27  cont7   300000 non-null  float64
 28  cont8   300000 non-null  float64
 29  cont9   300000 non-null  float64
 30  cont10  300000 non-null  float64
 31  target  300000 non-null  int64
dtypes: float64(11), int64(2), object(19)
memory usage: 73.2+ MB
```

In [29]:
```python
df.shape
```

Out[29]:
```
(500000, 31)
```

In [34]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500000 entries, 0 to 499999
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   id      500000 non-null  int64
 1   cat0    500000 non-null  object
 2   cat1    500000 non-null  object
 3   cat2    500000 non-null  object
 4   cat3    500000 non-null  object
 5   cat4    500000 non-null  object
 6   cat5    500000 non-null  object
 7   cat6    500000 non-null  object
 8   cat7    500000 non-null  object
 9   cat8    500000 non-null  object
 10  cat9    500000 non-null  object
 11  cat10   500000 non-null  object
 12  cat11   500000 non-null  object
 13  cat12   500000 non-null  object
 14  cat13   500000 non-null  object
 15  cat14   500000 non-null  object
 16  cat15   500000 non-null  object
 17  cat16   500000 non-null  object
 18  cat17   500000 non-null  object
 19  cat18   500000 non-null  object
 20  cont0   500000 non-null  float64
 21  cont1   500000 non-null  float64
 22  cont2   500000 non-null  float64
 23  cont3   500000 non-null  float64
 24  cont4   500000 non-null  float64
 25  cont5   500000 non-null  float64
 26  cont6   500000 non-null  float64
 27  cont7   500000 non-null  float64
 28  cont8   500000 non-null  float64
 29  cont9   500000 non-null  float64
 30  cont10  500000 non-null  float64
dtypes: float64(11), int64(1), object(19)
memory usage: 118.3+ MB
```

In [ ]:
```python
y_train.shape
```

In [6]:
```python
train_df['cat0'].value_counts
```

Out[6]:
```
<bound method IndexOpsMixin.value_counts of 0         A
1         A
2         A
3         A
4         A
         ..
299995    A
299996    A
299997    A
299998    B
299999    A
Name: cat0, Length: 300000, dtype: object>
```

In [7]:
```python
test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   id      200000 non-null  int64
 1   cat0    200000 non-null  object
 2   cat1    200000 non-null  object
 3   cat2    200000 non-null  object
 4   cat3    200000 non-null  object
 5   cat4    200000 non-null  object
 6   cat5    200000 non-null  object
 7   cat6    200000 non-null  object
 8   cat7    200000 non-null  object
 9   cat8    200000 non-null  object
 10  cat9    200000 non-null  object
 11  cat10   200000 non-null  object
 12  cat11   200000 non-null  object
 13  cat12   200000 non-null  object
 14  cat13   200000 non-null  object
 15  cat14   200000 non-null  object
 16  cat15   200000 non-null  object
 17  cat16   200000 non-null  object
 18  cat17   200000 non-null  object
 19  cat18   200000 non-null  object
 20  cont0   200000 non-null  float64
 21  cont1   200000 non-null  float64
 22  cont2   200000 non-null  float64
 23  cont3   200000 non-null  float64
 24  cont4   200000 non-null  float64
 25  cont5   200000 non-null  float64
 26  cont6   200000 non-null  float64
 27  cont7   200000 non-null  float64
 28  cont8   200000 non-null  float64
 29  cont9   200000 non-null  float64
 30  cont10  200000 non-null  float64
dtypes: float64(11), int64(1), object(19)
memory usage: 47.3+ MB
```

Feature Engineering

In [28]:
```python
df_dummies = pd.get_dummies(df).drop(columns='id')
df_dummies.head()
```

Out[28]:

	cont0	cont1	cont2	cont3	cont4	cont5	cont6	cont7	cont8	cont9	...	cat16_C	cat16_D	cat17_A
0	0.629858	0.855349	0.759439	0.795549	0.681917	0.621672	0.592184	0.791921	0.815254	0.965006	...	0	1	0
1	0.370727	0.328929	0.386385	0.541366	0.388982	0.357778	0.600044	0.408701	0.399353	0.927406	...	0	0	0
2	0.502272	0.322749	0.343255	0.616352	0.793687	0.552877	0.352113	0.388835	0.412303	0.292696	...	0	1	0
3	0.934242	0.707663	0.831147	0.807807	0.800032	0.619147	0.221789	0.897617	0.633669	0.760318	...	0	1	0
4	0.254427	0.274514	0.338818	0.277308	0.610578	0.128291	0.578764	0.279167	0.351103	0.357084	...	0	0	0

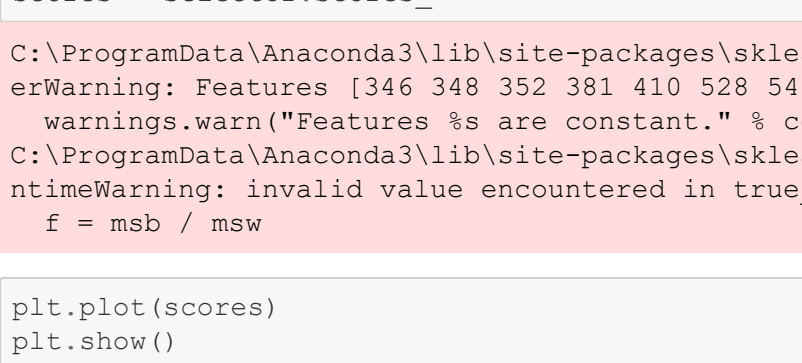
5 rows × 642 columns

In [29]:
```python
X_train = df_dummies.iloc[0:300000,0:642]

X_test = df_dummies.iloc[300000:500000,0:642]
```

In [30]:
```python
selector = SelectKBest(f_classif, k=10)
selected_features = selector.fit_transform(X_train, y_train)

scores = selector.scores_
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\feature_selection\_univariate_selection.py:114: Us
erWarning: Features [346 348 352 381 410 528 544 552] are constant.
  warnings.warn("Features %s are constant." % constant_features_idx,
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\feature_selection\_univariate_selection.py:116: Ru
ntimeWarning: invalid value encountered in true_divide
  f = msb / msw
```

In [8]:
```python
plt.plot(scores)
plt.show()
```



In [9]:
```python
f_score_indexes = (-selector.scores_).argsort()[:50]
f_score_indexes
```

Out[9]:
```
array([631, 633, 629, 627, 639, 625, 624, 641, 618, 619,  11,  12, 637,
       640, 636,   5,  43, 622, 623,  21, 279,  24,   6,   8,  64,   1,
       186, 292,  67, 164,   3,  40,  28,   2,  19,  20, 635, 296,  68,
       260, 355,  42, 368, 188, 237,  13,  30, 170, 208, 171], dtype=int64)
```

In [90]:
```python
scores[464]
```

Out[90]:
```
0.005542254653787618
```

In [13]:
```python
df_dummies.head()
```

Out[13]:

	cont0	cont1	cont2	cont3	cont4	cont5	cont6	cont7	cont8	cont9	...	cat16_C	cat16_D	cat17_A
0	0.629858	0.855349	0.759439	0.795549	0.681917	0.621672	0.592184	0.791921	0.815254	0.965006	...	0	1	0
1	0.370727	0.328929	0.386385	0.541366	0.388982	0.357778	0.600044	0.408701	0.399353	0.927406	...	0	0	0
2	0.502272	0.322749	0.343255	0.616352	0.793687	0.552877	0.352113	0.388835	0.412303	0.292696	...	0	1	0
3	0.934242	0.707663	0.831147	0.807807	0.800032	0.619147	0.221789	0.897617	0.633669	0.760318	...	0	1	0
4	0.254427	0.274514	0.338818	0.277308	0.610578	0.128291	0.578764	0.279167	0.351103	0.357084	...	0	0	0

5 rows × 642 columns

In [31]:
```python
cols = df_dummies.columns

for j in range(642):
    if scores[j] < 3:
        df_dummies = df_dummies.drop(cols[j], axis=1)
```

In [32]:
```python
df_dummies.shape
```

Out[32]:
```
(500000, 502)
```

In [33]:
```python
X_train = df_dummies.iloc[0:300000,0:502]

X_test = df_dummies.iloc[300000:500000,0:502]
```

In [37]:
```python
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300000 entries, 0 to 299999
Data columns (total 22 columns):
 #   Column    Non-Null Count   Dtype
---  ------    --------------   -----
 0   cat0_A    300000 non-null  uint8
 1   cat0_B    300000 non-null  uint8
 2   cat10_BU  300000 non-null  uint8
 3   cat10_BW  300000 non-null  uint8
 4   cat10_CA  300000 non-null  uint8
 5   cat10_DG  300000 non-null  uint8
 6   cat10_EJ  300000 non-null  uint8
 7   cat10_JM  300000 non-null  uint8
 8   cat10_KE  300000 non-null  uint8
 9   cat10_KM  300000 non-null  uint8
 10  cat11_A   300000 non-null  uint8
 11  cat11_B   300000 non-null  uint8
 12  cat14_A   300000 non-null  uint8
 13  cat14_B   300000 non-null  uint8
 14  cat15_B   300000 non-null  uint8
 15  cat15_D   300000 non-null  uint8
 16  cat16_B   300000 non-null  uint8
 17  cat16_D   300000 non-null  uint8
 18  cat17_D   300000 non-null  uint8
 19  cat18_B   300000 non-null  uint8
 20  cat18_C   300000 non-null  uint8
 21  cat18_D   300000 non-null  uint8
dtypes: uint8(22)
memory usage: 6.3 MB
```

In [34]:
```python
X_train_df, X_test_df, y_train_df, y_test_df = train_test_split(X_train, y_train, test_size=0.4)
```

In [35]:
```python
X_train_df.shape
```

Out[35]:
```
(180000, 502)
```

Model Selection

In [98]:
```python
MLA = [

ensemble.AdaBoostClassifier(),
ensemble.BaggingClassifier(),
ensemble.ExtraTreesClassifier(),
ensemble.GradientBoostingClassifier(),
ensemble.RandomForestClassifier(),

linear_model.LogisticRegressionCV(),
linear_model.PassiveAggressiveClassifier(),
linear_model.RidgeClassifierCV(),
linear_model.SGDClassifier(),
linear_model.Perceptron(),

naive_bayes.BernoulliNB(),
naive_bayes.GaussianNB(),

neighbors.KNeighborsClassifier(),

svm.SVC(probability=True),
svm.NuSVC(probability=True),
svm.LinearSVC(),

tree.DecisionTreeClassifier(),
tree.ExtraTreeClassifier(),

discriminant_analysis.LinearDiscriminantAnalysis(),
discriminant_analysis.QuadraticDiscriminantAnalysis(),

#XGBClassifier()

]
```

In [48]:
```python
model = Adamodel.fit(X_train,y_train)
```

In [56]:
```python
formod = forest.fit(X_train, y_train)
```

In [23]:
```python
logmod = log.fit(X_train, y_train)
```

In [39]:
```python
nbmod = nb.fit(X_train, y_train)
```

Model Testing

In [99]:
```python
def MLA_test(X_initial, y_initial, f, t, verbose=0):
    X = X_initial[f:t]
    y = y_initial[f:t]

    #split dataset in cross-validation with this splitter class: http://scikit-learn.org/stable/module
s/generated/sklearn.model_selection.ShuffleSplit.html#sklearn.model_selection.ShuffleSplit
    #note: this is an alternative to train_test_split
    cv_split = model_selection.ShuffleSplit(n_splits = 10, test_size = .3, train_size = .6, random_stat
e = 0 ) # run model 10x with 60/30 split intentionally leaving out 10%

    #create table to compare MLA metrics
    MLA_columns = ['MLA Name', 'MLA Parameters','MLA Train Accuracy Mean', 'MLA Test Accuracy Mean', 'M
LA Test Accuracy 3*STD' ,'MLA Time']
    MLA_compare = pd.DataFrame(columns = MLA_columns)

    #index through MLA and save performance to table
    row_index = 0
    for alg in MLA:
        #set name and parameters
        MLA_name = alg.__class__.__name__
        MLA_compare.loc[row_index, 'MLA Name'] = MLA_name
        MLA_compare.loc[row_index, 'MLA Parameters'] = str(alg.get_params())
        #score model with cross validation: http://scikit-learn.org/stable/modules/generated/sklearn.mo
del_selection.cross_validate.html#sklearn.model_selection.cross_validate
        cv_results = model_selection.cross_validate(alg, X, y, cv  = cv_split, n_jobs=-1, verbose=0, re
turn_train_score=True)

        training_score = cv_results['train_score'].mean()
        test_score = cv_results['test_score'].mean()
        if verbose == 1:
            print('{}/{}'.format(row_index+1, len(MLA)), MLA_name, " - ", training_score, test_score)

        MLA_compare.loc[row_index, 'MLA Time'] = cv_results['fit_time'].mean()
        MLA_compare.loc[row_index, 'MLA Train Accuracy Mean'] = training_score
        MLA_compare.loc[row_index, 'MLA Test Accuracy Mean'] = test_score
        #if this is a non-bias random sample, then +/-3 standard deviations (std) from the mean, should
 statistically capture 99.7% of the subsets
        MLA_compare.loc[row_index, 'MLA Test Accuracy 3*STD'] = cv_results['test_score'].std()*3   #le
t's know the worst that can happen!

        #save MLA predictions - see section 6 for usage
        #alg.fit(data1[data1_x_bin], df[Target])
        #MLA_predict[MLA_name] = alg.predict(df[data1_x_bin])

        row_index+=1

    #print and sort table: https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.sort
_values.html
    MLA_compare.sort_values(by = ['MLA Test Accuracy Mean'], ascending = False, inplace = True)


    return MLA_compare
```

In [101]:
```python
MLA_compare = MLA_test(X_train, y_train, 0, 10000, verbose=1)
MLA_compare
```

```
1/21 AdaBoostClassifier  -  0.8054666666666666 0.7969
2/21 BaggingClassifier  -  0.9829166666666665 0.7838
3/21 ExtraTreesClassifier  -  1.0 0.8038000000000001
4/21 GradientBoostingClassifier  -  0.8313666666666666 0.8056333333333334
5/21 RandomForestClassifier  -  0.9999499999999999 0.8059333333333333
6/21 LogisticRegressionCV  -  0.7961166666666666 0.7946
7/21 PassiveAggressiveClassifier  -  0.7028333333333333 0.7012
8/21 RidgeClassifierCV  -  0.7933833333333333 0.7908666666666667
9/21 SGDClassifier  -  0.7904 0.7868999999999999
10/21 Perceptron  -  0.72455 0.7271333333333333
11/21 BernoulliNB  -  0.7308499999999999 0.7277999999999999
12/21 GaussianNB  -  0.6931 0.6927333333333332
13/21 KNeighborsClassifier  -  0.8389666666666666 0.7825333333333333
14/21 SVC  -  0.8028333333333334 0.7969
15/21 NuSVC  -  0.7922833333333333 0.7876666666666666
16/21 LinearSVC  -  0.79745 0.794
17/21 DecisionTreeClassifier  -  1.0 0.7227666666666667
18/21 ExtraTreeClassifier  -  1.0 0.7187
19/21 LinearDiscriminantAnalysis  -  0.7956 0.7926333333333332
20/21 QuadraticDiscriminantAnalysis  -  0.6650666666666666 0.6648666666666666
21/21 XGBClassifier  -  0.9830500000000001 0.7987666666666666
```

Out[101]:

	MLA Name	MLA Parameters	MLA Train Accuracy Mean	MLA Test Accuracy Mean	MLA Test Accuracy 3*STD	MLA Time
4	RandomForestClassifier	{'bootstrap': True, 'ccp_alpha': 0.0, 'class_w...	0.99995	0.805933	0.0190252	4.24012
3	GradientBoostingClassifier	{'ccp_alpha': 0.0, 'criterion': 'friedman_mse'...	0.831367	0.805633	0.0217368	6.58936
2	ExtraTreesClassifier	{'bootstrap': False, 'ccp_alpha': 0.0, 'class_...	1	0.8038	0.0152525	2.36166
20	XGBClassifier	{'objective': 'binary:logistic', 'base_score':...	0.98305	0.798767	0.0102961	4.89539
13	SVC	{'C': 1.0, 'break_ties': False, 'cache_size': ...	0.802833	0.7969	0.0207415	22.9352
0	AdaBoostClassifier	{'algorithm': 'SAMME.R', 'base_estimator': Non...	0.805467	0.7969	0.0166436	1.87583
5	LogisticRegressionCV	{'Cs': 10, 'class_weight': None, 'cv': None, '...	0.796117	0.7946	0.0187286	2.0944
15	LinearSVC	{'C': 1.0, 'class_weight': None, 'dual': True,...	0.79745	0.794	0.0160873	0.457899
18	LinearDiscriminantAnalysis	{'n_components': None, 'priors': None, 'shrink...	0.7956	0.792633	0.0187481	0.0859158
7	RidgeClassifierCV	{'alphas': array([0.1, 1. , 10.]), 'class_w...	0.793383	0.790867	0.0193142	0.042682
14	NuSVC	{'break_ties': False, 'cache_size': 200, 'clas...	0.792283	0.787667	0.0209189	25.6482
8	SGDClassifier	{'alpha': 0.0001, 'average': False, 'class_wei...	0.7904	0.7869	0.0181497	0.115682
1	BaggingClassifier	{'base_estimator': None, 'bootstrap': True, 'b...	0.982917	0.7838	0.0185213	1.67855
12	KNeighborsClassifier	{'algorithm': 'auto', 'leaf_size': 30, 'metric...	0.838967	0.782533	0.019653	0.281363
10	BernoulliNB	{'alpha': 1.0, 'binarize': 0.0, 'class_prior':...	0.73085	0.7278	0.018145	0.020067
9	Perceptron	{'alpha': 0.0001, 'class_weight': None, 'early...	0.72455	0.727133	0.221373	0.0345262
16	DecisionTreeClassifier	{'ccp_alpha': 0.0, 'class_weight': None, 'crit...	1	0.722767	0.0159502	0.310905
17	ExtraTreeClassifier	{'ccp_alpha': 0.0, 'class_weight': None, 'crit...	1	0.7187	0.0176094	0.0375465
6	PassiveAggressiveClassifier	{'C': 1.0, 'average': False, 'class_weight': N...	0.702833	0.7012	0.266824	0.0426862
11	GaussianNB	{'priors': None, 'var_smoothing': 1e-09}	0.6931	0.692733	0.0290923	0.0212302
19	QuadraticDiscriminantAnalysis	{'priors': None, 'reg_param': 0.0, 'store_cova...	0.665067	0.664867	0.19889	0.0280566

TensorFlow Model

In [36]:
```python
df_dummies.shape
```

Out[36]:
```
(500000, 502)
```

In [37]:
```python
y_train.shape
```

Out[37]:
```
(300000,)
```

In [38]:
```python
f = 0
t = 180000
X = X_train_df[f:t]
y = y_train_df[f:t]
```

In [39]:
```python
X_train_df.shape
```

Out[39]:
```
(180000, 502)
```

In [77]:
```python
model = keras.Sequential([

    keras.layers.Dense(60, activation='relu', input_shape= (502,)),
    keras.layers.Dense(40, activation='relu'),
    keras.layers.Dense(30, activation='relu'),
    keras.layers.Dense(20, activation='relu'),
    keras.layers.Dense(10, activation='relu'),
    keras.layers.Dense(1, activation='relu')
])


model.compile(optimizer='adam', loss='mse', metrics=['accuracy'])
history = model.fit(X, y, epochs=2, validation_split=0.3, shuffle=True)

#best model: layers: 50-25-10-1  502 features 'adam' 'mse' 2 epochs
```

```
Epoch 1/2
3938/3938 [==============================] - 17s 4ms/step - loss: 0.1257 - accuracy: 0.8326 - val_los
s: 0.1125 - val_accuracy: 0.8455
Epoch 2/2
3938/3938 [==============================] - 15s 4ms/step - loss: 0.1116 - accuracy: 0.8466 - val_los
s: 0.1109 - val_accuracy: 0.8478
```

In [78]:
```python
test_loss, test_acc = model.evaluate(X_test_df, y_test_df, verbose=1)

print('Test accuracy:', test_acc)
```

```
3750/3750 [==============================] - 12s 3ms/step - loss: 0.1113 - accuracy: 0.8470
Test accuracy: 0.8470083475112915
```

In [62]:
```python
y_pred = model.predict(X_test)
```

In [63]:
```python
y_pred.shape
```

Out[63]:
```
(200000, 1)
```

In [64]:
```python
test_ids.shape
```

Out[64]:
```
(200000,)
```

In [65]:
```python
df_results = pd.DataFrame({'id':test_ids, 'target':y_pred.reshape(y_pred.shape[0])})
df_results.head()
```

Out[65]:

	id	target
0	5	0.139930
1	6	0.434369
2	8	0.050159
3	9	0.119325
4	11	0.116143

In [66]:
```python
compression_opts = dict(method='zip',
                        archive_name='results19.csv')

df_results.to_csv('results19.zip', index=False,
          compression=compression_opts)
```

In [ ]: