In [1]:

```
import numpy as np
import pandas as pd
import gc

import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from sklearn import model_selection
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import RobustScaler
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import StandardScaler
from sklearn.manifold import TSNE
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.preprocessing import normalize

import lightgbm as lgb
import xgboost as xgb
from xgboost import plot_importance


from catboost import CatBoostRegressor


from IPython.display import display # Allows the use

import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
import catboost
from catboost import CatBoostRegressor
```

[load Data]

In [2]:

```
train_df = pd.read_csv('csv/train.csv')
test_df = pd.read_csv('csv/test.csv')
```

```
import pandas_profiling
pandas_profiling.ProfileReport(train_df)
```

In [3]:

```
X_train = train_df.drop(["ID", "target"], axis=1)
y_train = np.log1p(train_df["target"].values)

X_test = test_df.drop(["ID"], axis=1)
```

In [5]:
```python
print("Train set size: {}".format(X_train.shape))
print("Test set size: {}".format(X_test.shape))
```

```
Train set size: (4459, 4991)
Test set size: (49342, 4991)
```

In [4]:
```python
# check and remove constant columns
colsToRemove = []
for col in X_train.columns:
    if X_train[col].std() == 0:
        colsToRemove.append(col)

# remove constant columns in the training set
X_train.drop(colsToRemove, axis=1, inplace=True)

# remove constant columns in the test set
X_test.drop(colsToRemove, axis=1, inplace=True)

print("Removed `{}` Constant Columns\n".format(len(co
print(colsToRemove)
```

```
Removed `256` Constant Columns

['d5308d8bc', 'c330f1a67', 'eeac16933', '7df8788e8',
'5b91580ee', '6f29fbbc7', '46dafc868', 'ae41a98b6', '
f416800e9', '6d07828ca', '7ac332a1d', '70ee7950a', '8
33b35a7c', '2f9969eab', '8b1372217', '68322788b', '22
88ac1a6', 'dc7f76962', '467044c26', '39ebfbfd9', '9a5
ff8c23', 'f6fac27c8', '664e2800e', 'ae28689a2', 'd87d
cac58', '4065efbb6', 'f944d9d43', 'c2c4491d5', 'a4346
e2e2', '1af366d4f', 'cfff5b7c8', 'da215e99e', '5acd26
139', '9be9c6cef', '1210d0271', '21b0a54cb', 'da35e79
2b', '754c502dd', '0b346adbd', '0f196b049', 'b603ed95
d', '2a50e001c', '1e81432e7', '10350ea43', '3c7c7e24c
', '7585fce2a', '64d036163', 'f25d9935c', 'd98484125'
, '95c85e227', '9a5273600', '746cdb817', '6377a6293',
'7d944fb0c', '87eb21c50', '5ea313a8c', '0987a65a1', '
2fb7c2443', 'f5dde409b', '1ae50d4c3', '2b21cd7d8', '0
db8a9272', '804d8b55b', '76f135fa6', '7d7182143', 'f8
8e61ae6', '378ed28e0', 'ca4ba131e', '1352ddae5', '2b6
01ad67', '6e42ff7c7', '22196a84c', '0e410eb3d', '992e
6d1d3', '90a742107', '08b9ec4ae', 'd95203ded', '58ad5
1def', '9f69ae59f', '863de8a31', 'be10df47c', 'f006d9
618', 'a7e39d23d', '5ed0abe85', '6c578fe94', '7fa4fce
e9', '5e0571f07', 'fd5659511', 'e06b9f40f', 'c506599c
8', '99de8c2dc', 'b05f4b229', '5e0834175', 'eb1cc0d9c
', 'b281a62b9', '00fcf67e4', 'e37b65992', '2308e2b29'
, 'c342e8709', '708471ebf', 'f614aac15', '15ecf7b68',
'3bfe540f1', '7a0d98f3c', 'e642315a5', 'c16d456a7', '
0c9b5bcfa', 'b778ab129', '2ace87cdd', '697a566f0', '9
7b1f84fc', '34eff114b', '5281333d7', 'c89f3ba7e', 'cd
6d3c7e6', 'fc7c8f2e8', 'abbbf9f82', '24a233e8f', '8e2
6b560e', 'a28ac1049', '504502ce1', 'd9a8615f3', '4efd
6d283', '34cc56e83', '93e98252a', '2b6cef19e', 'c7f70
a49b', '0d29ab7eb', 'e4a0d39b7', 'a4d1a8409', 'bc694f
c8f', '3a36fc3a2', '4ffba44d3', '9bfdec4bc', '66a866d
```

```
2f', 'f941e9df7', 'e7af4dbf3', 'dc9a54a3e', '748168a0
4', 'bba8ce4bb', 'ff6f62aa4', 'b06fe66ba', 'ae87ebc42
', 'f26589e57', '963bb53b1', 'a531a4bf0', '9fc79985d'
, '9350d55c1', 'de06e884c', 'fc10bdf18', 'e0907e883',
'c586d79a1', 'e15e1513d', 'a06067897', '643e42fcb', '
217cd3838', '047ebc242', '9b6ce40cf', '3b2c972b3', '1
7a7bf25a', 'c9028d46b', '9e0473c91', '6b041d374', '78
3c50218', '19122191d', 'ce573744f', '1c4ea481e', 'fbd
6e0a0b', '69831c049', 'b87e3036b', '54ba515ee', 'a09b
a0b15', '90f77ec55', 'fb02ef0ea', '3b0cccd29', 'fe9ed
417c', '589e8bd6f', '17b5a03fd', '80e16b49a', 'a3d5c2
c2a', '1bd3a4e92', '611d81daa', '3d7780b1c', '113fd02
06', '5e5894826', 'cb36204f9', 'bc4e3d600', 'c66e2deb
0', 'c25851298', 'a7f6de992', '3f93a3272', 'c1b95c2ec
', '6bda21fee', '4a64e56e7', '943743753', '20854f8bf'
, 'ac2e428a9', '5ee7de0be', '316423a21', '2e52b0c6a',
'8bdf6bc7e', '8f523faf2', '4758340d5', '8411096ec', '
9678b95b7', 'a185e35cc', 'fa980a778', 'c8d90f7d7', '0
80540c81', '32591c8b4', '5779da33c', 'bb425b41e', '01
599af81', '1654ab770', 'd334a588e', 'b4353599c', '51b
53eaec', '2cc0fbc52', '45ffef194', 'c15ac04ee', '5b05
5c8ea', 'd0466eb58', 'a80633823', 'a117a5409', '7ddac
276f', '8c32df8b3', 'e5649663e', '6c16efbb8', '9118fd
5ca', 'ca8d565f1', '16a5bb8d2', 'fd6347461', 'f5179fb
9c', '97428b646', 'f684b0a96', 'e4b2caa9f', '2c2d9f26
7', '96eb14eaf', 'cb2cb460c', '86f843927', 'ecd16fc60
', '801c6dc8e', 'f859a25b8', 'ae846f332', '2252c7403'
, 'fb9e07326', 'd196ca1fd', 'a8e562e8e', 'eb6bb7ce1',
'5beff147e', '52b347cdc', '4600aadcf', '6fa0b9dab', '
43d70cc4d', '408021ef8', 'e29d22b59']
```

In [5]:

```
gc.collect()
print("Train set size: {}".format(X_train.shape))
print("Test set size: {}".format(X_test.shape))
```

```
Train set size: (4459, 4735)
Test set size: (49342, 4735)
```

In [5]:

```
%%time
# Check and remove duplicate columns
colsToRemove = []
colsScaned = []
dupList = {}

columns = X_train.columns

for i in range(len(columns)-1):
    v = X_train[columns[i]].values
    dupCols = []
    for j in range(i+1,len(columns)):
        if np.array_equal(v, X_train[columns[j]].valu
            colsToRemove.append(columns[j])
            if columns[j] not in colsScaned:
                dupCols.append(columns[j])
                colsScaned.append(columns[j])
                dupList[columns[i]] = dupCols

# remove duplicate columns in the training set
X_train.drop(colsToRemove, axis=1, inplace=True)

# remove duplicate columns in the testing set
X_test.drop(colsToRemove, axis=1, inplace=True)

print("Removed `{}` Duplicate Columns\n".format(len(d
print(dupList)
```

```
Removed `4` Duplicate Columns

{'34ceb0081': ['d60ddde1b'], '8d57e2749': ['acc5b709d
', 'f333a5f60'], '168b3e5bc': ['f8d75792f'], 'a765da8
bc': ['912836770']}
CPU times: user 5min 34s, sys: 1.71 s, total: 5min 36
s
Wall time: 5min 36s
```

In [7]:

```
gc.collect()
print("Train set size: {}".format(X_train.shape))
print("Test set size: {}".format(X_test.shape))
```

```
Train set size: (4459, 4730)
Test set size: (49342, 4730)
```

In [6]:

```
# Drop Sparse Data

def drop_sparse(train, test):
    flist = [x for x in train.columns if not x in ['I
    for f in flist:
        if len(np.unique(train[f]))<2:
            train.drop(f, axis=1, inplace=True)
            test.drop(f, axis=1, inplace=True)
    return train, test
```

In [7]:

```
%%time
X_train, X_test = drop_sparse(X_train, X_test)
```

```
CPU times: user 740 ms, sys: 2.03 ms, total: 742 ms
Wall time: 741 ms
```

In [10]:

```
gc.collect()
print("Train set size: {}".format(X_train.shape))
print("Test set size: {}".format(X_test.shape))
```

```
Train set size: (4459, 4730)
Test set size: (49342, 4730)
```

In [8]:

```
#  Add Features
## SumZeros

def add_SumZeros(train, test, features):
    flist = [x for x in train.columns if not x in ['I
    if 'SumZeros' in features:
        train.insert(1, 'SumZeros', (train[flist] ==
        test.insert(1, 'SumZeros', (test[flist] == 0)
    flist = [x for x in train.columns if not x in ['I

    return train, test
```

In [9]:

```
%%time
X_train, X_test = add_SumZeros(X_train, X_test, ['Sum
```

```
CPU times: user 7.75 s, sys: 3.47 s, total: 11.2 s
Wall time: 11.5 s
```

In [13]:

```
gc.collect()
print("Train set size: {}".format(X_train.shape))
print("Test set size: {}".format(X_test.shape))
```

```
Train set size: (4459, 4731)
Test set size: (49342, 4731)
```

In [10]:

```
## SumValues

def add_SumValues(train, test, features):
    flist = [x for x in train.columns if not x in ['I
    if 'SumValues' in features:
        train.insert(1, 'SumValues', (train[flist] !=
        test.insert(1, 'SumValues', (test[flist] != 0
    flist = [x for x in train.columns if not x in ['I

    return train, test
```

In [11]:

```
%%time
X_train, X_test = add_SumValues(X_train, X_test, ['Su
```

```
CPU times: user 7.6 s, sys: 3.65 s, total: 11.3 s
Wall time: 11.5 s
```

In [16]:

```
gc.collect()
print("Train set size: {}".format(X_train.shape))
print("Test set size: {}".format(X_test.shape))
```

```
Train set size: (4459, 4732)
Test set size: (49342, 4732)
```

In [12]:

```
## Other Aggregates

def add_OtherAgg(train, test, features):
    flist = [x for x in train.columns if not x in ['I
    if 'OtherAgg' in features:
        train['Mean']   = train[flist].mean(axis=1)
        train['Median'] = train[flist].median(axis=1)
        train['Mode']   = train[flist].mode(axis=1)
        train['Max']    = train[flist].max(axis=1)
        train['Var']    = train[flist].var(axis=1)
        train['Std']    = train[flist].std(axis=1)
        train['Skew']    = train[flist].skew(axis=1)
        train['kurt']    = train[flist].kurt(axis=1)

        test['Mean']   = test[flist].mean(axis=1)
        test['Median'] = test[flist].median(axis=1)
        test['Mode']   = test[flist].mode(axis=1)
        test['Max']    = test[flist].max(axis=1)
        test['Var']    = test[flist].var(axis=1)
        test['Std']    = test[flist].std(axis=1)
        test['Skew']    = test[flist].skew(axis=1)
        test['kurt']    = test[flist].kurt(axis=1)
    flist = [x for x in train.columns if not x in ['I

    return train, test
```

In [13]:

```
%%time
X_train, X_test = add_OtherAgg(X_train, X_test, ['Oth
```

```
CPU times: user 1min 56s, sys: 37.6 s, total: 2min 33
s
Wall time: 2min 36s
```

In [14]:

```
gc.collect()
print("Train set size: {}".format(X_train.shape))
print("Test set size: {}".format(X_test.shape))
```

```
Train set size: (4459, 4740)
Test set size: (49342, 4740)
```

In [15]:
```
%%time

# K-Means

flist = [x for x in X_train.columns if not x in ['ID'

flist_kmeans = []
for ncl in range(2,21):
    cls = KMeans(n_clusters=ncl)
    cls.fit_predict(X_train[flist].values)
    X_train['kmeans_cluster_'+str(ncl)] = cls.predict
    X_test['kmeans_cluster_'+str(ncl)] = cls.predict(
    flist_kmeans.append('kmeans_cluster_'+str(ncl))
print(flist_kmeans)
```

```
['kmeans_cluster_2', 'kmeans_cluster_3', 'kmeans_clus
ter_4', 'kmeans_cluster_5', 'kmeans_cluster_6', 'kmea
ns_cluster_7', 'kmeans_cluster_8', 'kmeans_cluster_9'
, 'kmeans_cluster_10', 'kmeans_cluster_11', 'kmeans_c
luster_12', 'kmeans_cluster_13', 'kmeans_cluster_14',
'kmeans_cluster_15', 'kmeans_cluster_16', 'kmeans_clu
ster_17', 'kmeans_cluster_18', 'kmeans_cluster_19', '
kmeans_cluster_20']
CPU times: user 7min 29s, sys: 1min 23s, total: 8min
52s
Wall time: 7min
```

In [16]:
```
gc.collect()
print("Train set size: {}".format(X_train.shape))
print("Test set size: {}".format(X_test.shape))
```

```
Train set size: (4459, 4759)
Test set size: (49342, 4759)
```

In [17]:
```
%%time
# PCA

flist = [x for x in X_train.columns if not x in ['ID'

n_components = 100
flist_pca = []
pca = PCA(n_components=n_components)
x_train_projected = pca.fit_transform(normalize(X_tra
x_test_projected = pca.transform(normalize(X_test[fli
for npca in range(0, n_components):
    X_train.insert(1, 'PCA_'+str(npca+1), x_train_pro
    X_test.insert(1, 'PCA_'+str(npca+1), x_test_proje
    flist_pca.append('PCA_'+str(npca+1))
print(flist_pca)
```

```
['PCA_1', 'PCA_2', 'PCA_3', 'PCA_4', 'PCA_5', 'PCA_6'
, 'PCA_7', 'PCA_8', 'PCA_9', 'PCA_10', 'PCA_11', 'PCA
_12', 'PCA_13', 'PCA_14', 'PCA_15', 'PCA_16', 'PCA_17
', 'PCA_18', 'PCA_19', 'PCA_20', 'PCA_21', 'PCA_22',
'PCA_23', 'PCA_24', 'PCA_25', 'PCA_26', 'PCA_27', 'PC
A_28', 'PCA_29', 'PCA_30', 'PCA_31', 'PCA_32', 'PCA_3
3', 'PCA_34', 'PCA_35', 'PCA_36', 'PCA_37', 'PCA_38',
'PCA_39', 'PCA_40', 'PCA_41', 'PCA_42', 'PCA_43', 'PC
A_44', 'PCA_45', 'PCA_46', 'PCA_47', 'PCA_48', 'PCA_4
9', 'PCA_50', 'PCA_51', 'PCA_52', 'PCA_53', 'PCA_54',
'PCA_55', 'PCA_56', 'PCA_57', 'PCA_58', 'PCA_59', 'PC
A_60', 'PCA_61', 'PCA_62', 'PCA_63', 'PCA_64', 'PCA_6
5', 'PCA_66', 'PCA_67', 'PCA_68', 'PCA_69', 'PCA_70',
'PCA_71', 'PCA_72', 'PCA_73', 'PCA_74', 'PCA_75', 'PC
A_76', 'PCA_77', 'PCA_78', 'PCA_79', 'PCA_80', 'PCA_8
1', 'PCA_82', 'PCA_83', 'PCA_84', 'PCA_85', 'PCA_86',
'PCA_87', 'PCA_88', 'PCA_89', 'PCA_90', 'PCA_91', 'PC
A_92', 'PCA_93', 'PCA_94', 'PCA_95', 'PCA_96', 'PCA_9
7', 'PCA_98', 'PCA_99', 'PCA_100']
CPU times: user 22.2 s, sys: 7.81 s, total: 30.1 s
Wall time: 17.5 s
```

In [18]:
```
gc.collect()
print("Train set size: {}".format(X_train.shape))
print("Test set size: {}".format(X_test.shape))
```

```
Train set size: (4459, 4859)
Test set size: (49342, 4859)
```

In [29]:
```
X_train.head()
```

Out[29]:

	48df886f9	PCA_30	PCA_29	PCA_28	PCA_27	PCA_26	PCA
0	0.0	-0.006539	-0.006042	0.010928	-0.002797	0.012687	0.006
1	0.0	0.002054	-0.001779	-0.001739	0.005825	0.005565	0.002
2	0.0	-0.001189	0.003169	-0.000672	0.001723	-0.002263	0.005
3	0.0	-0.002196	0.007200	-0.000559	0.006277	-0.005844	0.004
4	0.0	-0.001708	0.002251	-0.000263	0.002030	-0.002663	0.007

5 rows × 4787 columns

In [30]:
```
X_test.head()
```

Out[30]:

	48df886f9	PCA_30	PCA_29	PCA_28	PCA_27	PCA_26	PC
0	0.0	0.005431	0.006147	-0.013688	-0.001139	-0.000280	0.01
1	0.0	-0.003066	0.002949	-0.000819	0.000769	-0.000934	0.00
2	0.0	-0.001286	-0.000462	-0.000460	-0.003330	0.005027	0.00
3	0.0	0.000883	-0.000061	-0.000949	0.004407	-0.001299	-0.00
4	0.0	0.003144	0.005251	0.002261	-0.000274	-0.004972	0.00

5 rows × 4787 columns

In [19]:
```
X_train.to_csv("csv/X_train-knn20-pca100.csv",index=F
```

In [20]:
```
X_test.to_csv("csv/X_test-knn20-pca100.csv",index=Fal
```

In [21]:
```
y_train
```

Out[21]:
```
array([17.45309674, 13.3046866 , 16.11809575, ..., 14
.84513033,
       16.11809575, 16.81124288])
```

In [22]:
```
np.save('csv/y_train-knn20-pca100.npy', y_train)
```

In [33]:
```

```

1 Reload Data

In [3]:
```
X_train = pd.read_csv("csv/X_train-knn20-pca30.csv")
X_test = pd.read_csv("csv/X_test-knn20-pca30.csv")
y_train = np.load("csv/y_train-knn20-pca30.npy")
```

In [4]:
```
X_train_sparse=X_train.replace(0, np.nan).to_sparse()
X_test_sparse=X_test.replace(0, np.nan).to_sparse()
```

In [2]:
```
# Save Ram 95%

test_df = pd.read_csv('csv/test.csv', index_col='ID')
test_size_mb = test_df.memory_usage().sum() / 1024 /
print("Test memory size: %.2f MB" % test_size_mb)
# Test memory size: 1879.24 MB

test_df_sparse = test_df.replace(0, np.nan).to_sparse
test_sparse_size_mb = test_df_sparse.memory_usage().s
print("Test sparse memory size: %.2f MB" % test_spars
```

```
Test memory size: 1879.24 MB
Test sparse memory size: 26.78 MB
```

In [14]:
```
dev_X, val_X, dev_y, val_y = train_test_split(X_train
```

In [5]:
```
dev_X, val_X, dev_y, val_y = train_test_split(X_train
```

2 Random Forest Regressor

In [103]:
```
#Import Library
from sklearn.ensemble import RandomForestRegressor
```

In [56]:
```
from sklearn.model_selection import RandomizedSearchC
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 2
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf nod
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
```

In [57]:
```
rf = RandomForestRegressor()
# Random search of parameters, using 3 fold cross val
# search across 100 different combinations, and use a
rf_random = RandomizedSearchCV(estimator = rf, param_
                               verbose=2, random_stat
```

In [58]:
```
rf_random.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 100 candidates, totalling
300 fits
[CV] n_estimators=400, min_samples_split=5, min_sampl
es_leaf=1, max_features=sqrt, max_depth=30, bootstrap
=True
[CV] n_estimators=400, min_samples_split=5, min_sampl
es_leaf=1, max_features=sqrt, max_depth=30, bootstrap
=True
[CV] n_estimators=400, min_samples_split=5, min_sampl
es_leaf=1, max_features=sqrt, max_depth=30, bootstrap
=True
[CV] n_estimators=2000, min_samples_split=5, min_samp
les_leaf=1, max_features=sqrt, max_depth=10, bootstra
p=True
[CV] n_estimators=2000, min_samples_split=5, min_samp
les_leaf=1, max_features=sqrt, max_depth=10, bootstra
p=True
[CV] n_estimators=2000, min_samples_split=5, min_samp
les_leaf=1, max_features=sqrt, max_depth=10, bootstra
```

In [59]:
```
rf_random.best_params_
```

Out[59]:
```
{'bootstrap': True,
 'max_depth': 40,
 'max_features': 'auto',
 'min_samples_leaf': 4,
 'min_samples_split': 2,
 'n_estimators': 600}
```

In [ ]:
```
def evaluate(model, test_features, test_labels):
    predictions = model.predict(test_features)
    errors = abs(predictions - test_labels)
    mape = 100 * np.mean(errors / test_labels)
    accuracy = 100 - mape
    print('Model Performance')
    print('Average Error: {:0.4f} degrees.'.format(np
    print('Accuracy = {:0.2f}%.'.format(accuracy))

    return accuracy
```

In [62]:
```
best_random_rc = rf_random.best_estimator_
```

In [63]:
```
pred_final_test_rc= np.expm1(best_random_rc.predict(X
```

In [66]:
```
sub_rf_random = pd.read_csv('csv/sample_submission.cs
sub_rf_random["target"] = pred_final_test_rc

sub_rf_random.to_csv('result/sub_rf_07-01-1.csv', ind
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```
from sklearn.model_selection import GridSearchCV
# Create the parameter grid based on the results of r
param_grid = {
    'bootstrap': [True],
    'max_depth': [80, 90, 100, 110],
    'max_features': [2, 3],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [100, 200, 300, 1000]
}
# Create a based model
rf = RandomForestRegressor()
# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf, param_grid
                          cv = 3, n_jobs = -1, verbos
```

In [ ]:
```
# Fit the grid search to the data
grid_search.fit(train_features, train_labels)
grid_search.best_params_
```

In [ ]:
```
best_grid = grid_search.best_estimator_
grid_accuracy = evaluate(best_grid, test_features, te
```

In [ ]:
```
print('Improvement of {:0.2f}%.'.format( 100 * (grid_
```

In [ ]:

In [ ]:

In [37]:
```
RFModel= RandomForestRegressor(n_estimators=1000,  n_
                              max_features = "auto",mi
```

In [38]:
```
RFModel.fit(X_train, y_train)
```

Out[38]:
```
RandomForestRegressor(bootstrap=True, criterion='mse'
, max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_sp
lit=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators
=1000, n_jobs=-1,
           oob_score=False, random_state=50, verbose=
0, warm_start=False)
```

In [49]:
```
pred_final_test= np.expm1(RFModel.predict(X_test) )
```

In [50]:
```
# Predictions

sub_rf = pd.read_csv('csv/sample_submission.csv')
sub_rf["target"] = pred_final_test

sub_rf.to_csv('result/sub_rf_06-30-1.csv', index=Fals
```

In [67]:
```
sub_rf_random.tail()
```

Out[67]:

	ID	target
49337	fff73b677	1.430668e+06
49338	fff7b5923	1.116732e+07
49339	fff7c698f	1.403312e+06
49340	fff8dba89	1.118543e+06
49341	fffbe2f6f	2.774964e+06

In [51]:
```
sub_rf.tail()
```

Out[51]:

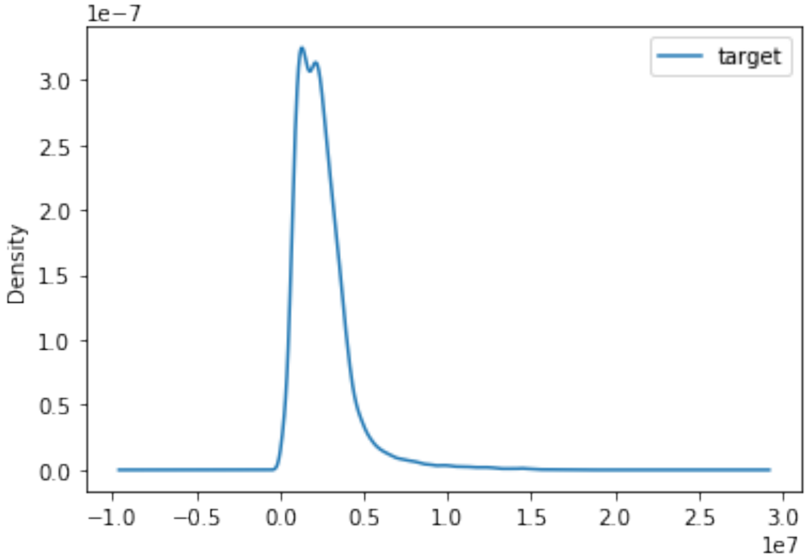
	ID	target
49337	fff73b677	1.664626e+06
49338	fff7b5923	1.162498e+07
49339	fff7c698f	1.515234e+06
49340	fff8dba89	1.266355e+06
49341	fffbe2f6f	3.031582e+06

In [52]:
```
from sklearn.metrics import mean_squared_error
from math import sqrt

print("RMSE : ", sqrt(mean_squared_error(y_train,  RF
```

```
RMSE :  0.5044603124644393
```

In [53]:
```
sub_rf.plot(kind="kde")
```

Out[53]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x12b5fec18
>
```



In [ ]:

In [ ]:

In [34]:
```
def run_rf(train_X, train_y, val_X, val_y, test_X):
    params = {
        "n_estimators": 1000,
        "n_jobs" : -1,
        "random_state" : 50,
        "max_features" : "auto",
        "min_samples_leaf" : 5,
        "max_depth": 20,
        "bootstrap": "bootstrap"
    }

    rftrain = rfr.Dataset(train_X, label=train_y)
    lgval = rfr.Dataset(val_X, label=val_y)
    evals_result = {}
    rfr.train
    model = rfr.train(params, lgtrain, 1000, valid_se
                      verbose_eval=200, evals_result=

    pred_test_y = model.predict(test_X, num_iteration
    return pred_test_y, model, evals_result
```

In [55]:
```
rfr.
```

```
Object `rfr.train` not found.
```

Build Train and Test Data for Modeling

LightGBM

In [ ]:

In [ ]:

In [83]:

```python
def run_lgb(train_X, train_y, val_X, val_y, test_X):
    params = {
        "objective" : "regression",
        "metric" : "rmse",
        "num_leaves" : 30,
        "learning_rate" : np.random.rand()*0.002+0.00
        "bagging_fraction" : np.random.rand()*0.03+0.
        "feature_fraction" : np.random.rand()*0.03+0.
        "bagging_frequency" : 5,
        "bagging_seed" : 2018,
        "verbosity" : -1
    }

    lgtrain = lgb.Dataset(train_X, label=train_y)
    lgval = lgb.Dataset(val_X, label=val_y)
    evals_result = {}
    model = lgb.train(params, lgtrain, 1000, valid_se
                      verbose_eval=200, evals_result=

    pred_test_y = model.predict(test_X, num_iteration
    return pred_test_y, model, evals_result
```

In [ ]:

In [84]:

```python
%%time
# Training LGB
seeds = [42, 2018]
pred_test_full_seed = 0
for seed in seeds:
    kf = model_selection.KFold(n_splits=5, shuffle=Tr
    pred_test_full = 0
    for dev_index, val_index in kf.split(X_train):
        dev_X, val_X = X_train.loc[dev_index,:], X_tr
        dev_y, val_y = y_train[dev_index], y_train[va
        pred_test, model, evals_result = run_lgb(dev_
        pred_test_full += pred_test
    pred_test_full /= 5.
    pred_test_full = np.expm1(pred_test_full)
    pred_test_full_seed += pred_test_full
    print("Seed {} completed....".format(seed))
pred_test_full_seed /= np.float(len(seeds))

print("LightGBM Training Completed...")
```

```
Training until validation scores don't improve for 10
0 rounds.
[200]   training's rmse: 1.34625        valid_1's rms
e: 1.43238
[400]   training's rmse: 1.1639 valid_1's rmse: 1.371
72
[600]   training's rmse: 1.04511        valid_1's rms
```

```
e: 1.35941
[800]   training's rmse: 0.956577       valid_1's rms
e: 1.35483
[1000]  training's rmse: 0.885017       valid_1's rms
e: 1.35167
Did not meet early stopping. Best iteration is:
[1000]  training's rmse: 0.885017       valid_1's rms
e: 1.35167
Training until validation scores don't improve for 10
0 rounds.
[200]   training's rmse: 1.28659        valid_1's rms
e: 1.47138
[400]   training's rmse: 1.09646        valid_1's rms
e: 1.41291
[600]   training's rmse: 0.972954       valid_1's rms
e: 1.40162
[800]   training's rmse: 0.882767       valid_1's rms
e: 1.40034
Early stopping, best iteration is:
[816]   training's rmse: 0.876469       valid_1's rms
e: 1.40018
Training until validation scores don't improve for 10
0 rounds.
[200]   training's rmse: 1.31946        valid_1's rms
e: 1.46462
[400]   training's rmse: 1.13648        valid_1's rms
e: 1.40465
[600]   training's rmse: 1.01635        valid_1's rms
e: 1.39126
[800]   training's rmse: 0.928433       valid_1's rms
e: 1.38829
[1000]  training's rmse: 0.857084       valid_1's rms
e: 1.38773
Did not meet early stopping. Best iteration is:
[1000]  training's rmse: 0.857084       valid_1's rms
e: 1.38773
Training until validation scores don't improve for 10
0 rounds.
[200]   training's rmse: 1.337  valid_1's rmse: 1.473
17
[400]   training's rmse: 1.15782        valid_1's rms
e: 1.40692
[600]   training's rmse: 1.03992        valid_1's rms
e: 1.38586
[800]   training's rmse: 0.950469       valid_1's rms
e: 1.37918
Early stopping, best iteration is:
[818]   training's rmse: 0.943375       valid_1's rms
e: 1.37898
Training until validation scores don't improve for 10
0 rounds.
[200]   training's rmse: 1.29295        valid_1's rms
e: 1.41247
[400]   training's rmse: 1.10413        valid_1's rms
e: 1.36331
[600]   training's rmse: 0.977646       valid_1's rms
e: 1.35632
```

```
Early stopping, best iteration is:
[630]   training's rmse: 0.962113       valid_1's rms
e: 1.35588
Seed 42 completed....
Training until validation scores don't improve for 10
0 rounds.
[200]   training's rmse: 1.30475        valid_1's rms
e: 1.44745
[400]   training's rmse: 1.12024        valid_1's rms
e: 1.38553
[600]   training's rmse: 0.997091       valid_1's rms
e: 1.36889
[800]   training's rmse: 0.905037       valid_1's rms
e: 1.36561
Early stopping, best iteration is:
[870]   training's rmse: 0.877396       valid_1's rms
e: 1.36485
Training until validation scores don't improve for 10
0 rounds.
[200]   training's rmse: 1.30655        valid_1's rms
e: 1.5274
[400]   training's rmse: 1.12258        valid_1's rms
e: 1.46459
[600]   training's rmse: 1.00188        valid_1's rms
e: 1.45065
[800]   training's rmse: 0.911771       valid_1's rms
e: 1.44773
[1000]  training's rmse: 0.839054       valid_1's rms
e: 1.44684
Did not meet early stopping. Best iteration is:
[1000]  training's rmse: 0.839054       valid_1's rms
e: 1.44684
Training until validation scores don't improve for 10
0 rounds.
[200]   training's rmse: 1.26982        valid_1's rms
e: 1.41552
[400]   training's rmse: 1.0796 valid_1's rmse: 1.365
01
[600]   training's rmse: 0.955231       valid_1's rms
e: 1.35059
[800]   training's rmse: 0.86393        valid_1's rms
e: 1.3457
[1000]  training's rmse: 0.78912        valid_1's rms
e: 1.34439
Did not meet early stopping. Best iteration is:
[1000]  training's rmse: 0.78912        valid_1's rms
e: 1.34439
Training until validation scores don't improve for 10
0 rounds.
[200]   training's rmse: 1.28006        valid_1's rms
e: 1.41389
[400]   training's rmse: 1.0911 valid_1's rmse: 1.359
68
[600]   training's rmse: 0.966417       valid_1's rms
e: 1.3494
[800]   training's rmse: 0.873186       valid_1's rms
e: 1.34701
```

```
Early stopping, best iteration is:
[894]   training's rmse: 0.835737       valid_1's rms
e: 1.34621
Training until validation scores don't improve for 10
0 rounds.
[200]   training's rmse: 1.35628        valid_1's rms
e: 1.40659
[400]   training's rmse: 1.17456        valid_1's rms
e: 1.35257
[600]   training's rmse: 1.0561 valid_1's rmse: 1.342
96
[800]   training's rmse: 0.966723       valid_1's rms
e: 1.34004
Early stopping, best iteration is:
[866]   training's rmse: 0.941514       valid_1's rms
e: 1.33904
Seed 2018 completed....
LightGBM Training Completed...
CPU times: user 52min 21s, sys: 1min 42s, total: 54mi
n 4s
Wall time: 9min 38s
```

In [36]:

```
# feature importance
print("Features Importance...")
gain = model.feature_importance('gain')
featureimp = pd.DataFrame({'feature':model.feature_na
                   'split':model.feature_importance('
                   'gain':100 * gain / gain.sum()}).s
print(featureimp[:15])
```

```
Features Importance...
          feature  split       gain
4765          Max    108  10.981523
31      SumValues    324  10.358513
30          PCA_1    187   8.169489
4766          Var     77   5.647445
4163    f190486d6     92   5.428007
25          PCA_6     97   3.108202
32       SumZeros     58   2.126455
4762         Mean     93   1.939694
4767          Std     17   1.775125
2410    58e2e02e6     46   1.670494
11         PCA_20     46   1.589677
29          PCA_2     64   1.458935
9          PCA_22     51   1.343045
28          PCA_3     48   1.052109
1582    26ab20ff9     34   0.975456
```

In [85]:
```
# feature importance
print("Features Importance...")
gain = model.feature_importance('gain')
featureimp = pd.DataFrame({'feature':model.feature_na
                           'split':model.feature_importance('
                           'gain':100 * gain / gain.sum()}).s
print(featureimp[:15])
```

```
Features Importance...
         feature  split      gain
4835         Max    379  9.843301
101    SumValues   1183  7.388539
100        PCA_1    604  6.345557
4233   f190486d6    362  4.770581
4836         Var    229  4.330360
102     SumZeros    490  3.421039
95         PCA_6    254  2.097547
4837         Std     82  1.874990
4832        Mean    279  1.645605
99         PCA_2    247  1.625822
2480   58e2e02e6    172  1.326224
79        PCA_22    183  1.274365
4839        kurt    345  1.242536
4838        Skew    293  0.913440
73        PCA_28    209  0.829514
```

In [ ]:

In [ ]:

In [86]:
```
# Predictions

sub_lgb = pd.read_csv('csv/sample_submission.csv')
sub_lgb["target"] = pred_test_full_seed
```

In [97]:
```
print(sub_lgb.head())

sub_lgb.to_csv('result/sub_lgb100_07-03-2.csv', index
```

```
          ID        target
0  000137c73  2.568800e+06
1  00021489f  2.315777e+06
2  0004d7953  1.251207e+06
3  00056a333  3.346352e+06
4  00056d8eb  2.898810e+06
```

In [ ]:

```
booster = "gbtree",
           eval_metric = "rmse",
           nthread = 4,
           eta = 0.007,
           max_depth = 22,
           min_child_weight = 57,
           gamma = 1.444155,
           subsample = 0.6731153,
           colsample_bytree = 0.05427895,
           colsample_bylevel = 0.70762806,
           alpha = 0,
           lambda = 0,
           nrounds = 6000)
```

In [15]:

```
params1={ 'objective': 'reg:linear',
           'n_estimators': 1000,
           'booster' : 'gbtree',
           'eval_metric' : 'rmse',
           'nthread' : 4,
           'eta' : np.random.rand()*0.0001+0.005,
           'learning_rate': np.random.rand()*0.001+0.0
           'max_depth' : 22,
           'min_child_weight' : 56,
           'gamma' : np.random.rand()*0.01+1.44,
           'subsample' : np.random.rand()*0.005+0.6713
           'colsample_bytree' : np.random.rand()*0.002
           'colsample_bylevel' : np.random.rand()*0.01
           'alpha' : 0.0,
           'lambda' : 0.0,
           'nround' : 5000,
           'random_state': 42,
           'silent': True
        }
```

In [117]:

```
params1={ 'objective': 'reg:linear',
           'n_estimators': 1000,
           'booster' : 'gbtree',
           'eval_metric' : 'rmse',
           'nthread' : 4,
           'eta' : np.random.rand()*0.0001+0.005,
           'learning_rate': np.random.rand()*0.001+0.0
           'max_depth' : 22,
           'min_child_weight' : 56,
           'gamma' : np.random.rand()*0.01+1.44,
           'subsample' : np.random.rand()*0.005+0.6713
           'colsample_bytree' : np.random.rand()*0.002
           'colsample_bylevel' : np.random.rand()*0.01
           'alpha' : 0.0,
           'lambda' : 0.0,
           'nround' : 5000,
           'random_state': 42,
           'silent': True
        }
```

In [110]:
```
params11={ 'objective': 'reg:linear',
           'n_estimators': 400,
           'booster' : 'gbtree',
           'eval_metric' : 'rmse',
           'nthread' : 4,
           'eta' : np.random.rand()*0.0001+0.007,
           'learning_rate': np.random.rand()*0.001+0.0
           'max_depth' : 30,
           'min_child_weight' : 30,
           'gamma' : np.random.rand()*0.001,#+1.44,
           'subsample' : np.random.rand()*0.005+0.75,
           'colsample_bytree' : np.random.rand()*0.02+
           'colsample_bylevel' : np.random.rand()*0.01
           'alpha' : 0.0,
           'lambda' : 0.0,
           'reg_lambda':0.1,
           'nround' : 5000,
           'random_state': 42,
           'silent': True
         }
```

In [94]:
```
params11
```

Out[94]:
```
{'alpha': 0.0,
 'booster': 'gbtree',
 'colsample_bylevel': 0.7064166782317791,
 'colsample_bytree': 0.20090522203553685,
 'eta': 0.0070676292001709985,
 'eval_metric': 'rmse',
 'gamma': 6.011734204378005e-05,
 'lambda': 0.0,
 'learning_rate': 0.010895448915269808,
 'max_depth': 30,
 'min_child_weight': 30,
 'n_estimators': 400,
 'nround': 5000,
 'nthread': 4,
 'objective': 'reg:linear',
 'random_state': 42,
 'reg_lambda': 0.1,
 'silent': True,
 'subsample': 0.7538877716372883}
```

In [ ]:
```
params2 = {'objective': 'reg:linear',
           'eval_metric': 'rmse',
           'eta': 0.005,
           'max_depth': 15,
           'subsample': 0.7,
           'colsample_bytree': 0.5,
           'alpha':0,
           'random_state': 42,
           'silent': True}
```

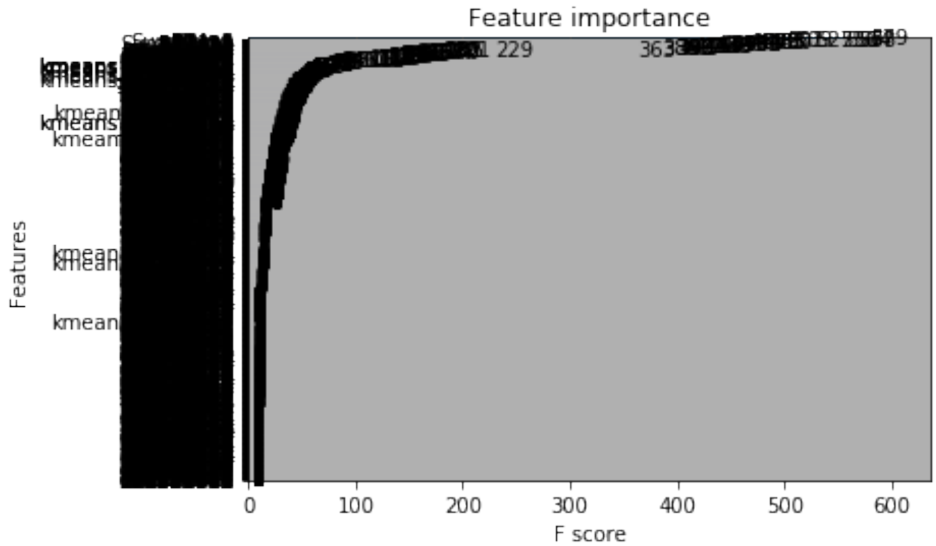
In [16]:
```python
def run_xgb(train_X, train_y, val_X, val_y, test_X):

    params = params1;
    tr_data = xgb.DMatrix(train_X, train_y)
    va_data = xgb.DMatrix(val_X, val_y)

    watchlist = [(tr_data, 'train'), (va_data, 'valid

    model_xgb = xgb.train(params, tr_data, 2000, watc

    dtest = xgb.DMatrix(test_X)
    xgb_pred_y = np.expm1(model_xgb.predict(dtest, nt

    return xgb_pred_y, model_xgb
```

In [17]:
```python
%%time
pred_test_xgb, model_xgb = run_xgb(dev_X, dev_y, val_
print("XGB Training Completed...")
```

```
[0]     train-rmse:13.8121      valid-rmse:13.8019
Multiple eval metrics have been passed: 'valid-rmse'
will be used for early stopping.

Will train until valid-rmse hasn't improved in 100 ro
unds.
[100]   train-rmse:2.17182      valid-rmse:2.21333
[200]   train-rmse:1.27186      valid-rmse:1.39993
[300]   train-rmse:1.17214      valid-rmse:1.36996
[400]   train-rmse:1.1028       valid-rmse:1.36637
Stopping. Best iteration:
[383]   train-rmse:1.11285      valid-rmse:1.36552

XGB Training Completed...
CPU times: user 1min, sys: 7.19 s, total: 1min 8s
Wall time: 1min 8s
```

In [77]:
```python
pred_test_xgb, model_xgb = run_xgb(dev_X, dev_y, val_
print("XGB Training Completed...")
```

```
[0]     train-rmse:13.811       valid-rmse:13.8008
Multiple eval metrics have been passed: 'valid-rmse'
will be used for early stopping.

Will train until valid-rmse hasn't improved in 100 ro
unds.
[100]   train-rmse:2.14054      valid-rmse:2.20523
[200]   train-rmse:1.2096       valid-rmse:1.40202
[300]   train-rmse:1.08588      valid-rmse:1.37207
[400]   train-rmse:0.995841     valid-rmse:1.36257
[500]   train-rmse:0.920355     valid-rmse:1.36
Stopping. Best iteration:
[445]   train-rmse:0.960342     valid-rmse:1.35973

XGB Training Completed...
```

In [113]:
```
plot_importance(model_xgb)
```
Out[113]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x127dddd30
>
```


In [41]:
```
def create_feature_map(features):
    outfile = open('xgb.fmap', 'w')
    i = 0
    for feat in features:
        outfile.write('{0}\t{1}\tq\n'.format(i, feat)
        i = i + 1

    outfile.close()
```

3 Catboost

In [56]:
```
CatBoostRegressor?
```
In [89]:
```
cb_model = CatBoostRegressor(iterations=500,
                             learning_rate=0.05,
                             depth=10,
                             eval_metric='RMSE',
                             random_seed = 42,
                             #bootstrap_type="Bayesia
                             #bagging_temperature = 0
                             od_type='Iter',
                             metric_period = 50,
                             od_wait=20)
```

In [91]:
```
cb_model.fit(dev_X, dev_y,
             eval_set=(val_X, val_y),
             use_best_model=True,
             verbose=True)
```

```
0:      learn: 13.8909919       test: 13.8532047
best: 13.8532047 (0)    total: 2.7s     remaining: 22
m 27s
50:     learn: 1.9823213        test: 1.9318554 best:
1.9318554 (50)  total: 2m 15s   remaining: 19m 53s
100:    learn: 1.4537572        test: 1.4411852 best:
1.4411852 (100) total: 4m 30s   remaining: 17m 49s
150:    learn: 1.3589251        test: 1.3946405 best:
1.3946405 (150) total: 6m 46s   remaining: 15m 40s
200:    learn: 1.2762128        test: 1.3744637 best:
1.3740578 (199) total: 9m 3s    remaining: 13m 28s
250:    learn: 1.2055244        test: 1.3674497 best:
1.3670947 (248) total: 11m 21s  remaining: 11m 16s
300:    learn: 1.1533054        test: 1.3636515 best:
1.3633247 (298) total: 13m 39s  remaining: 9m 1s
Stopped by overfitting detector  (20 iterations wait)

bestTest = 1.362469913
bestIteration = 311

Shrink model to first 312 iterations.
```

Out[91]:
```
<catboost.core.CatBoostRegressor at 0x126b075f8>
```

In [58]:

```
cb_model.fit(dev_X, dev_y,
             eval_set=(val_X, val_y),
             use_best_model=True,
             verbose=True)
```

```
0:      learn: 13.8898858       test: 13.8517139
best: 13.8517139 (0)    total: 3.27s    remaining: 27
m 14s
50:     learn: 1.9951918        test: 1.9511898 best:
1.9511898 (50)  total: 2m 57s   remaining: 26m 6s
100:    learn: 1.4630882        test: 1.4634321 best:
1.4634321 (100) total: 5m 45s   remaining: 22m 45s
150:    learn: 1.3788610        test: 1.4244734 best:
1.4244734 (150) total: 8m 37s   remaining: 19m 55s
200:    learn: 1.3016245        test: 1.4097275 best:
1.4096771 (199) total: 11m 31s  remaining: 17m 8s
250:    learn: 1.2435555        test: 1.4048450 best:
1.4044234 (242) total: 14m 29s  remaining: 14m 22s
300:    learn: 1.1824698        test: 1.3992065 best:
1.3977661 (284) total: 17m 32s  remaining: 11m 36s
Stopped by overfitting detector  (20 iterations wait)

bestTest = 1.397766112
bestIteration = 284

Shrink model to first 285 iterations.
```

Out[58]: <catboost.core.CatBoostRegressor at 0x1190f7358>

In [92]:

```
pred_test_cat = np.expm1(cb_model.predict(X_test))

sub_cat = pd.read_csv('csv/sample_submission.csv')

#sub_cat = pd.DataFrame()
sub_cat["target"] = pred_test_cat
```

In [95]:

```
sub_cat.to_csv('result/sub_cat100-2018-07-03-3.csv',
```

In [ ]:

```

```

```
%%time
# Training LGB
seeds = [42, 2018]
pred_test_full_seed_xgb = 0
for seed in seeds:
    kf = model_selection.KFold(n_splits=5,
shuffle=True, random_state=seed)
    pred_test_full_xgb = 0
    for dev_index, val_index in kf.split(X_train):
        dev_X, val_X = X_train.loc[dev_index,:],
X_train.loc[val_index,:]
        dev_y, val_y = y_train[dev_index],
y_train[val_index]
        pred_test, model, evals_result =
run_xgb(dev_X, dev_y, val_X, val_y, X_test)
        pred_test_full_xgb += pred_test
    pred_test_full_xgb /= 5.
    pred_test_full_xgb = np.expm1(pred_test_full_xgb)
    pred_test_full_seed_xgb += pred_test_full_xgb
    print("Seed {} completed....".format(seed))
pred_test_full_seed_xgb /= np.float(len(seeds))

print("Xgboost Training Completed...")
```

In [114]:

```
sub_xgb = pd.read_csv('csv/sample_submission.csv')
sub_xgb["target"] = pred_test_xgb
```

In [108]:

```
sub_xgb.tail()
```

Out[108]:

	ID	target
49337	fff73b677	62844.117188
49338	fff7b5923	95076.906250
49339	fff7c698f	73857.093750
49340	fff8dba89	61265.140625
49341	fffbe2f6f	70642.710938

In [79]:

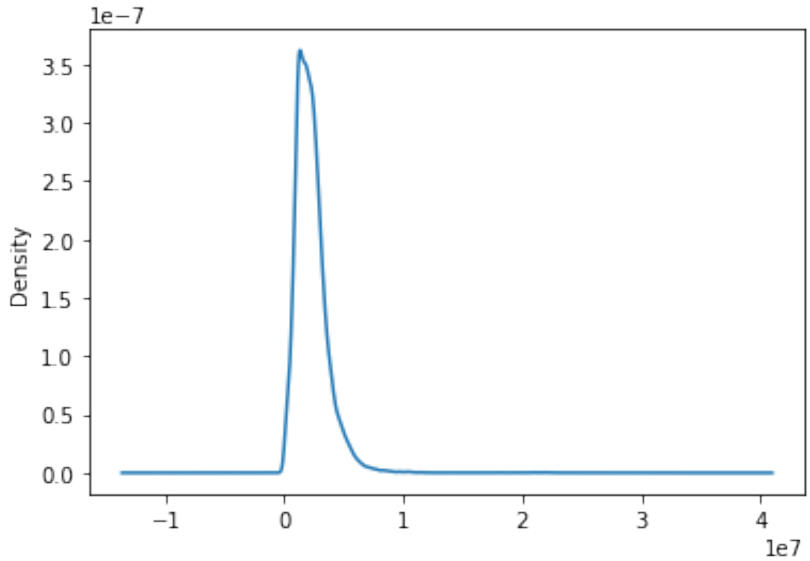
```
sub_xgb.tail()
```

Out[79]:

	ID	target
49337	fff73b677	2.270990e+06
49338	fff7b5923	4.171136e+06
49339	fff7c698f	2.942894e+06
49340	fff8dba89	7.673914e+05
49341	fffbe2f6f	2.392380e+06

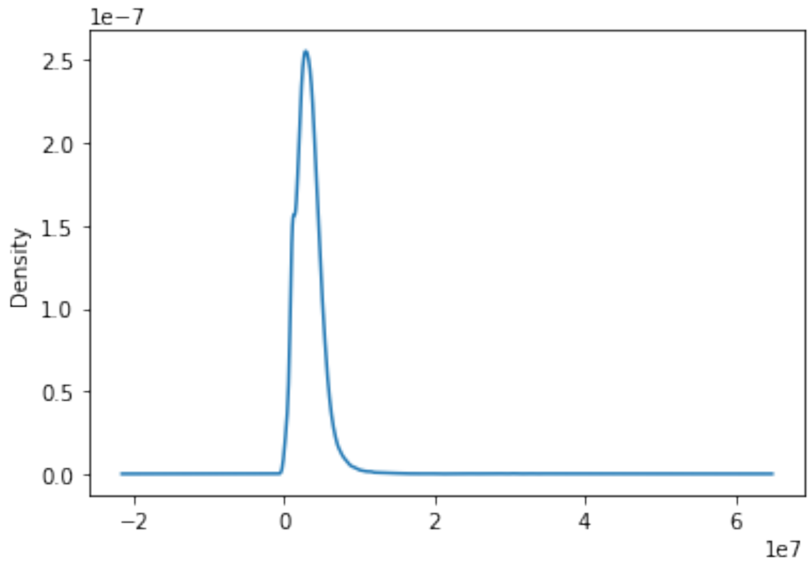
In [94]:
```
sub_cat["target"].plot(kind="kde")
```

Out[94]: <matplotlib.axes._subplots.AxesSubplot at 0x1276204e0
>



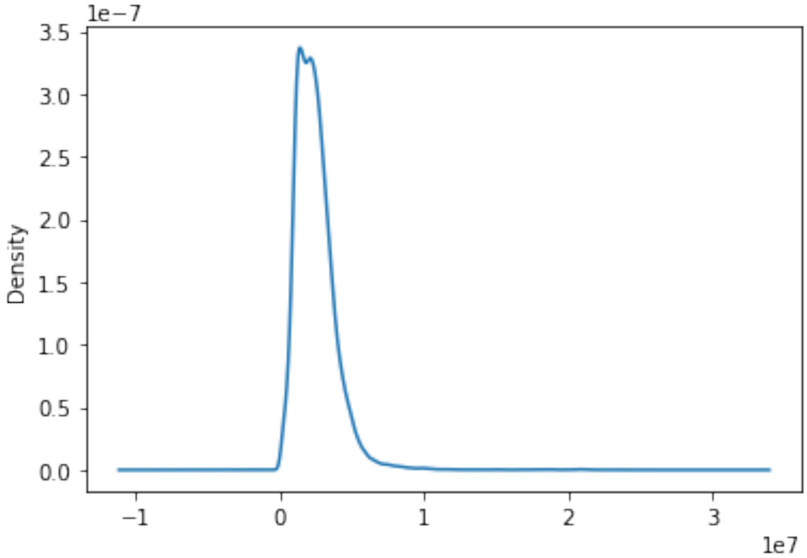
In [34]:
```
sub_xgb["target"].plot(kind="kde")
```

Out[34]: <matplotlib.axes._subplots.AxesSubplot at 0x114fe17f0
>



In [88]:
```
sub_lgb["target"].plot(kind="kde")
```

Out[88]: <matplotlib.axes._subplots.AxesSubplot at 0x128b96f28
>



4 Combine Predictions

In [33]:
```
sub_xgb = pd.read_csv('csv/sample_submission.csv')


#sub_xgb = pd.DataFrame()

sub_xgb["target"] =pred_test_xgb*1.3
sub_xgb.to_csv('result/sub_xgb100-2018-07-04-1.csv',
```

In [35]:
```
sub_xgb["target"].tail()
```

Out[35]:
```
49337    3.796644e+06
49338    4.787528e+06
49339    3.588330e+06
49340    6.984828e+05
49341    3.572033e+06
Name: target, dtype: float32
```

In [30]:
```
sub_xgb["target"].tail()
```

Out[30]:
```
49337    3.504595e+06
49338    4.419256e+06
49339    3.312304e+06
49340    6.447534e+05
49341    3.297262e+06
Name: target, dtype: float32
```

In [100]:
```
sub = pd.read_csv('csv/sample_submission.csv')

sub_lgb = pd.DataFrame()
sub_lgb["target"] = pred_test_full_seed

sub_xgb = pd.DataFrame()
sub_xgb["target"] =pred_test_xgb

sub_cat = pd.DataFrame()
sub_cat["target"] = pred_test_cat

sub["target"] = (sub_lgb["target"] + sub_xgb["target"
```

In [68]:
```
sub_cat.to_csv('result/sub_cat-2018-06-28.35.csv', in
```

In [69]:
```
sub_lgb.to_csv('result/sub_lgb-2018-06-28.csv', index
```

In [101]:
```
sub.to_csv('result/sub_stack-2018-07-03-1.15.csv', in
```

In [65]:
```
sub_xgb.tail()
```

Out[65]:

	target
49337	2.581754e+06
49338	3.772445e+06
49339	2.769820e+06
49340	5.214266e+05
49341	2.545089e+06

In [66]:
```
sub_lgb.tail()
```

Out[66]:

	target
49337	1.211180e+06
49338	6.347431e+06
49339	1.048917e+06
49340	8.867132e+05
49341	3.123992e+06

In [67]:
```
sub_cat.tail()
```

Out[67]:

	target
49337	6.493444e+05
49338	4.131360e+06
49339	1.072969e+06
49340	1.537915e+06
49341	3.602614e+06

In [ ]:
```

```

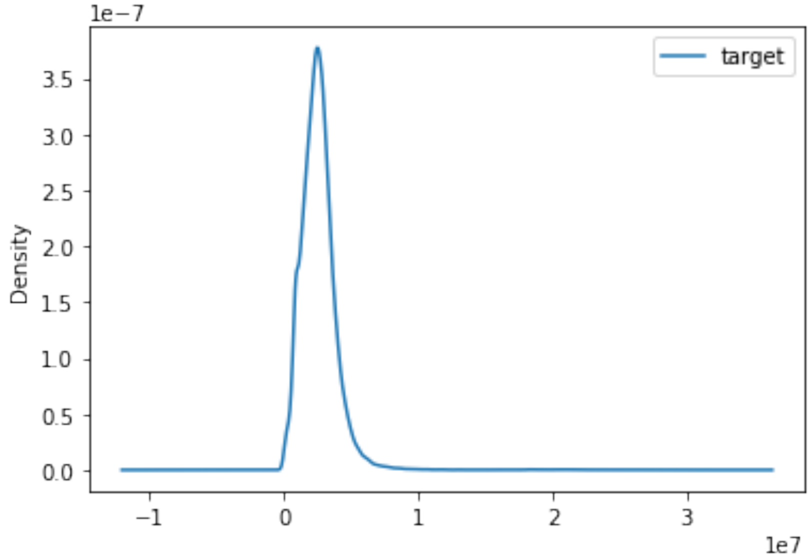
In [68]:
```
sub.tail()
```

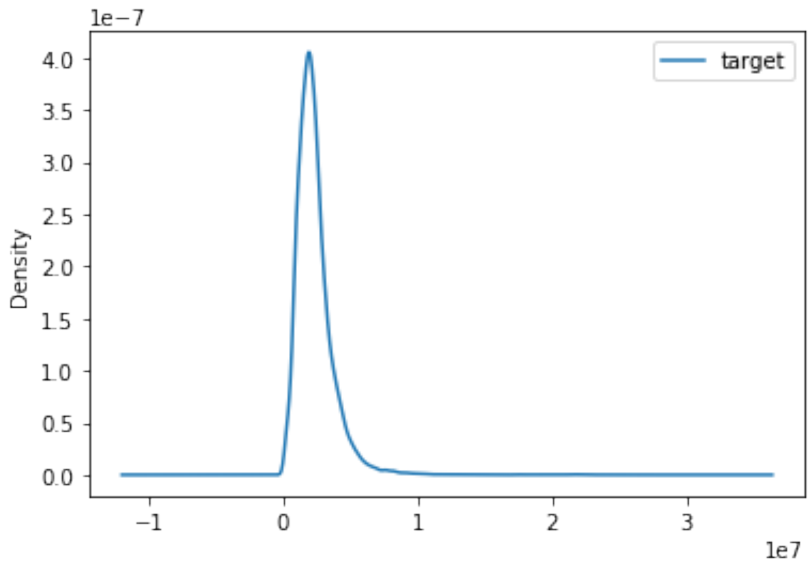
Out[68]:

	ID	target
49337	fff73b677	1.521492e+06
49338	fff7b5923	5.814285e+06
49339	fff7c698f	1.028849e+06
49340	fff8dba89	8.218056e+05
49341	fffbe2f6f	3.009544e+06

In [30]:
```
sub_xgb.plot(kind="kde")
sub_lgb.plot(kind="kde")
```

Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0x1240569e8
>





In [31]:
```
# 06-26, 0.95,0.15
sub.tail()
```

Out[31]:

	ID	target
49337	fff73b677	1.459404e+06
49338	fff7b5923	6.402822e+06
49339	fff7c698f	1.010917e+06
49340	fff8dba89	9.920673e+05
49341	fffbe2f6f	3.261140e+06

In [44]:
```python
# 06-25, 0.95,0.15
sub.tail()
```

Out[44]:

	ID	target
49337	fff73b677	1.471745e+06
49338	fff7b5923	5.967550e+06
49339	fff7c698f	1.115998e+06
49340	fff8dba89	1.061934e+06
49341	fffbe2f6f	3.580716e+06

In [72]:
```python
sub["target"] = (9.5*sub_lgb["target"] + 1.5*sub_xgb[
sub.to_csv('result/sub_lgb_xgb-2018-06-27-1-0.95.csv'
```

In [73]:
```python
sub["target"] = (1.5*sub_lgb["target"] + 9.5*sub_xgb[
sub.to_csv('result/sub_lgb_xgb-2018-06-27-1-0.15.csv'
```

In [ ]: