In [4]:
```
import numpy as np
import pandas as pd
import re
```

In [2]:
```
train_data = pd.read_json('train_data.json',orient="index")
test_data = pd.read_json('test_data.json',orient='index')
```

In [3]:
```
#set index
train_data.reset_index(level = 0, inplace = True)
train_data.rename(columns={'index':'ID'}, inplace=True)

test_data.reset_index(level = 0, inplace = True)
test_data.rename(columns={'index':'ID'}, inplace=True)
```

In [4]:
```
#check data
print ('Train data has {} rows and {} columns'.format(train_data.shape[0],trai
n_data.shape[1]))
print ('test_data data has {} rows and {} columns'.format(test_data.shape[0],t
est_data.shape[1]))
```

```
Train data has 200000 rows and 7 columns
test_data data has 100000 rows and 6 columns
```

In [5]:
```
#Encode Target Variable
train_data = train_data.replace({'segment':{'pos':1,'neg':0}})
```

In [6]:
```
#check target variable count
train_data['segment'].value_counts()/train_data.shape[0]
```

Out[6]:
```
0    0.923725
1    0.076275
Name: segment, dtype: float64
```

In [7]:
```
train_data['g1'] = [re.sub(pattern='\:\d+',repl='',string=x) for x in train_da
ta['genres']]
train_data['g1'] = train_data['g1'].apply(lambda x: x.split(','))

train_data['g2'] = [re.sub(pattern='\:\d+', repl='', string = x) for x in trai
n_data['dow']]
train_data['g2'] = train_data['g2'].apply(lambda x: x.split(','))

t1 = pd.Series(train_data['g1']).apply(frozenset).to_frame(name='t_genre')
t2 = pd.Series(train_data['g2']).apply(frozenset).to_frame(name='t_dow')
```

In [8]:
```
# using frozenset trick - might take few minutes to process
for t_genre in frozenset.union(*t1.t_genre):
    t1[t_genre] = t1.apply(lambda _: int(t_genre in _.t_genre), axis=1)
```

In [9]:
```
train_data = pd.concat([train_data.reset_index(drop=True), t1], axis=1)
train_data = pd.concat([train_data.reset_index(drop=True), t2], axis=1)
```

In [10]:
```
test_data['g1'] = [re.sub(pattern='\:\d+',repl='',string=x) for x in
test_data['genres']]
test_data['g1'] = test_data['g1'].apply(lambda x: x.split(','))

test_data['g2'] = [re.sub(pattern='\:\d+', repl='', string = x) for x in test_
data['dow']]
test_data['g2'] = test_data['g2'].apply(lambda x: x.split(','))

t1_te = pd.Series(test_data['g1']).apply(frozenset).to_frame(name='t_genre')
t2_te = pd.Series(test_data['g2']).apply(frozenset).to_frame(name='t_dow')
```

In [11]:
```
for t_genre in frozenset.union(*t1_te.t_genre):
    t1_te[t_genre] = t1_te.apply(lambda _: int(t_genre in _.t_genre), axis=1)
```

In [12]:
```
for t_dow in frozenset.union(*t2_te.t_dow):
    t2_te[t_dow] = t2_te.apply(lambda _: int(t_dow in _.t_dow), axis = 1)
```

In [13]:
```
test_data = pd.concat([test_data.reset_index(drop=True), t1_te], axis=1)
test_data = pd.concat([test_data.reset_index(drop=True), t2_te], axis=1)
```

In [14]:
```
#the rows aren't list exactly. They are object, so we convert them to list and
 extract the watch time
w1 = train_data['titles']
w1 = w1.str.split(',')

#create a nested list of numbers
main = []
for i in np.arange(train_data.shape[0]):
    d1 = w1[i]
    nest = []
    nest = [re.sub(pattern = '.*\:', repl=' ', string= d1[k]) for k in
list(np.arange(len(d1)))]
    main.append(nest)
```

In [15]:
```
blanks = []
for i in np.arange(len(main)):
    if '' in main[i]:
        print ("{} blanks found".format(len(blanks)))
        blanks.append(i)

#replacing blanks with 0
for i in blanks:
    main[i] = [x.replace('','0') for x in main[i]]

#converting string to integers
main = [[int(y) for y in x] for x in main]

#adding the watch time
tosum = []
for i in np.arange(len(main)):
    s = sum(main[i])
    tosum.append(s)
```

```
0 blanks found
1 blanks found
2 blanks found
3 blanks found
4 blanks found
5 blanks found
6 blanks found
7 blanks found
8 blanks found
9 blanks found
10 blanks found
11 blanks found
12 blanks found
13 blanks found
14 blanks found
15 blanks found
16 blanks found
17 blanks found
18 blanks found
19 blanks found
20 blanks found
21 blanks found
22 blanks found
23 blanks found
24 blanks found
25 blanks found
26 blanks found
27 blanks found
28 blanks found
29 blanks found
30 blanks found
31 blanks found
32 blanks found
33 blanks found
34 blanks found
35 blanks found
36 blanks found
37 blanks found
38 blanks found
39 blanks found
40 blanks found
41 blanks found
42 blanks found
43 blanks found
44 blanks found
45 blanks found
```

In [16]:
```
train_data['title_sum'] = tosum
```

In [17]:
```
#making changes in test data
w1_te = test_data['titles']
w1_te = w1_te.str.split(',')
```

In [18]:
```
main_te = []
for i in np.arange(test_data.shape[0]):
    d1 = w1_te[i]
    nest = []
    nest = [re.sub(pattern = '.*\:', repl=' ', string= d1[k]) for k in
list(np.arange(len(d1)))]
    main_te.append(nest)
```

In [19]:
```
blanks_te = []
for i in np.arange(len(main_te)):
    if '' in main_te[i]:
        print ("{} blanks found".format(len(blanks_te)))
        blanks_te.append(i)

#replacing blanks with 0
for i in blanks_te:
    main_te[i] = [x.replace('','0') for x in main_te[i]]

#converting string to integers
main_te = [[int(y) for y in x] for x in main_te]

#adding the watch time
tosum_te = []
for i in np.arange(len(main_te)):
    s = sum(main_te[i])
    tosum_te.append(s)
```

```
0 blanks found
1 blanks found
2 blanks found
3 blanks found
4 blanks found
5 blanks found
6 blanks found
7 blanks found
8 blanks found
9 blanks found
10 blanks found
11 blanks found
```

In [20]:
```
test_data['title_sum'] = tosum_te
```

In [21]:
```
#count variables
def wcount(p):
    return p.count(',')+1
```

In [22]:
```
train_data['title_count'] = train_data['titles'].map(wcount)
train_data['genres_count'] = train_data['genres'].map(wcount)
train_data['cities_count'] = train_data['cities'].map(wcount)
train_data['tod_count'] = train_data['tod'].map(wcount)
train_data['dow_count'] = train_data['dow'].map(wcount)


test_data['title_count'] = test_data['titles'].map(wcount)
test_data['genres_count'] = test_data['genres'].map(wcount)
test_data['cities_count'] = test_data['cities'].map(wcount)
test_data['tod_count'] = test_data['tod'].map(wcount)
test_data['dow_count'] = test_data['dow'].map(wcount)
```

In [23]:
```
test_id = test_data['ID']
train_data.drop(['ID','cities','dow','genres','titles','tod','g1','g2','t_genr
e','t_dow'], inplace=True, axis=1)
test_data.drop(['ID','cities','dow','genres','titles','tod','g1','g2','t_genr
e','t_dow'], inplace=True, axis=1)
```

In [ ]:
```

```

In [24]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [25]:
```
target = train_data['segment']
train_data.drop('segment',axis=1, inplace=True)
```

In [ ]:
```
#train final model
rf_model = RandomForestClassifier(n_estimators=500,max_depth=12,max_features=1
0)
rf_model.fit( train_data,target)
```

In [31]:
```
#make prediction
rf_pred = rf_model.predict_proba(test_data)
```

```
---------------------------------------------------------------------------
NotFittedError                            Traceback (most recent call last)
<ipython-input-31-ebc0a26ed790> in <module>()
      1 #make prediction
----> 2 rf_pred = rf_model.predict(test_data)

C:\Users\SIDDHARTH\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py in
predict(self, X)
    496             The predicted classes.
    497         """
--> 498         proba = self.predict_proba(X)
    499
    500         if self.n_outputs_ == 1:

C:\Users\SIDDHARTH\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py in
predict_proba(self, X)
    535         """
    536         # Check data
--> 537         X = self._validate_X_predict(X)
    538
    539         # Assign chunk of trees to jobs

C:\Users\SIDDHARTH\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py in
_validate_X_predict(self, X)
    314         """Validate X whenever one tries to predict, apply, predict_p
roba"""
    315         if self.estimators_ is None or len(self.estimators_) == 0:
--> 316             raise NotFittedError("Estimator not fitted, "
    317                                  "call `fit` before exploiting the mo
del.")
    318

NotFittedError: Estimator not fitted, call `fit` before exploiting the model.
```

In [149]:
```
print(rf_pred)
```

```
[[ 0.99262971  0.00737029]
 [ 0.99408008  0.00591992]
 [ 0.95159633  0.04840367]
 ...,
 [ 0.98684896  0.01315104]
 [ 0.9885485   0.0114515 ]
 [ 0.98937364  0.01062636]]
```

In [150]:
```
#make submission file and submit
columns = ['segment']
sub = pd.DataFrame(data=rf_pred[:,1], columns=columns)
sub['ID'] = test_id
sub = sub[['ID','segment']]
sub.to_csv("sub_hot.csv", index=False)
```

In [ ]:
```

```

In [ ]: