EDA_FE_model_final

December 27, 2020

Business Problem : we have to build a recommendation system that can predict whether a user will listen to a song again within one month after the user's very first observable listening event in KKbox. If the user did not listen to the song again within one month, the target variable will be 0, and 1 otherwise. This helps the company to recommend songs to users, to apply rating to songs and to determine the taste in songs of users.

ML formulation : Building a recommendation system using a collaborative filtering based algorithm with matrix factorization and word embedding.

performance metric : AUC ROC Score ROC Curve is the metric which calculates TPR and FPR at each thresholds and plots them and AUC ROC score is the area under the curve of ROC curve. This metric works best when data is balanced. while other metrics like F1-Score calculates precision and recall at a particular thresholds and considers predicted classes while AUC ROC Curve uses predicted scores. This predicted scores and thresholds helps us in determing the correct threshold that should be choosen for evaluation. while Plotting ROC Curve we make use of ordered predicted scores, we make use of each predicted score as a threshold and calculate TPR and FPR at each threshold. Then we plot ROC Curve on collected TPR and FPR values obtained by thresholding over predicted scores. The Point at which we have high TPR and low FPR this point can be used as right threshold at the time of inference. compared to f1 score and accuracy this metrics uses 0.5 as there threshold which is not right all the time. When your predictions overfit : ie y_true = [0,1,0,1,1,0,1,0,0,1] and y_pred = [1,1,1,1,1,1,1,1,1,1] (M1) as well as y_pred = [0,0,0,0,0,0,0,0,0,0] (M2) In this AUC(M1) is equal to AUC(M2) because when we apply thresholding on both model then TPR and FPR in the both the cases will be same. So AUC ROC Score doesn't make sense here. so we should apply F1-Score when we biased or overfitted or class imbalance problem when y_true and y_pred are inverted auc score varies from 0 to 0.5 this problem arises due to invertion of labels. can be solved by just inverting labels. Calculation of F1-Score : threshold = 0.5 precision = TP/(TP+FP) recall = TP/(TP+FN) F1-Score = 2xprecisionxRecall/(Precision+Recall)

Calculation of ROC Curve : 1.sort scores 2.for threshold in scores: calculate TPR and FPR list.add(TPR, FPR) 3.Plot TPR vs FPR curve 4.Calculate Area under the curve.

[1]:
```
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
import numpy as np
%matplotlib inline

df = pd.read_csv('train.csv')
df.head()
```

[2]:
```
                                           msno  \
0  FGtllVqz18RPiwJj/edr2gV78zirAiY/9SmYvia+kCg=   
1  Xumu+NIjS6QYVxDS4/t3SawvJ7viT9hPKXmf0RtLNx8=   
2  Xumu+NIjS6QYVxDS4/t3SawvJ7viT9hPKXmf0RtLNx8=   
3  Xumu+NIjS6QYVxDS4/t3SawvJ7viT9hPKXmf0RtLNx8=   
4  FGtllVqz18RPiwJj/edr2gV78zirAiY/9SmYvia+kCg=   

                                        song_id source_system_tab  \
0  BBzumQNXUHKdEBOB7mAJuzok+IJA1c2Ryg/yzTF6tik=           explore   
1  bhp/MpSNoqoxOIB+/l8WPqu6jldth4DIpCm3ayXnJqM=        my library   
2  JNWfrrC7zNN7BdMpsISKa4Mw+xVJYNnxXh3/Epw7QgY=        my library   
3  2A87tzfnJTSWqD7gIZHisolhe4DMdzkbd6LzO1KHjNs=        my library   
4  3qm6XTZ6MOCU11x8FIVbAGH5l5uMkT3/ZalWG1oo2Gc=           explore   

    source_screen_name      source_type  target  
0              Explore  online-playlist       1  
1  Local playlist more   local-playlist       1  
2  Local playlist more   local-playlist       1  
3  Local playlist more   local-playlist       1  
4              Explore  online-playlist       1  
```

[5]:
```
songs = pd.read_csv('songs.csv')
songs.head()
```

[5]:
```
                                        song_id  song_length genre_ids  \
0  CXoTN1eb7AI+DntdU1vbcwGRV4SCIDxZu+YD8JP8r4E=       247640       465   
1  o0kFgae9QtnYgRkVPqLJwa05zIhRlUjfF7O1tDw0ZDU=       197328       444   
2  DwVvVurfpuz+XPuFvucclVQEyPqcpUkHR0ne1RQzPs0=       231781       465   
3  dKMBWoZyScdxSkihKG+Vf47nc18N9q4m58+b4e7dSSE=       273554       465   
4  W3bqWd3T+VeHFzHAUfARgW9AvVRaF4N5Yzm4Mr6Eo/o=       140329       726   

       artist_name                              composer      lyricist  language  
0     (Jeff Chang)                                                           3.0  
1        BLACKPINK  TEDDY|   FUTURE BOUNCE|  Bekuh BOOM         TEDDY      31.0  
2     SUPER JUNIOR                                   NaN           NaN      31.0  
3            S.H.E                                                          3.0  
4                                          Traditional   Traditional      52.0  
```

[6]:
```
members = pd.read_csv('members.csv')
members.head()
```

[6]:
```
                                           msno  city  bd gender  \
0  XQxgAYj3klVKjR3oxPPXYYFp4soD4TuBghkhMTD4oTw=     1   0    NaN
1  UizsfmJb9mV54qE9hCYyU07Va97c0lCRLEQX3ae+ztM=     1   0    NaN
2  D8nEhsIOBSoE6VthTaqDX8U6lqjJ7dLdr72mOyLya2A=     1   0    NaN
3  mCuD+tZ1hERA/o5GPqk38e041J8ZsBaLcu7nGoIIvhI=     1   0    NaN
4  q4HRBfVSssAFS9iRfxWrohxuk9kCYMKjHOEagUMV6rQ=     1   0    NaN

   registered_via  registration_init_time  expiration_date
0               7                20110820         20170920
1               7                20150628         20170622
2               4                20160411         20170712
3               9                20150906         20150907
4               4                20170126         20170613
```

1 EDA

[5]:
```python
df['msno'].unique().shape[0], df['song_id'].unique().shape[0], df.shape[0]
```

[5]:
```
(30755, 359966, 7377418)
```

[6]:
```python
len(set(df['song_id'].unique()).intersection(set(songs['song_id'].unique())))
```

[6]:
```
359914
```

[7]:
```python
len(set(df['msno'].unique()).intersection(set(members['msno'].unique())))
```

[7]:
```
30755
```

Conclusion : 1. Out Of 359966 song_ids we have only information of 359914 song_ids.
2. Out Of 30755 members we have all information of 30755 members.

[5]:
```python
df = df.merge(members, on = 'msno', how='left')
df = df.merge(songs, on = 'song_id', how = 'left')
```

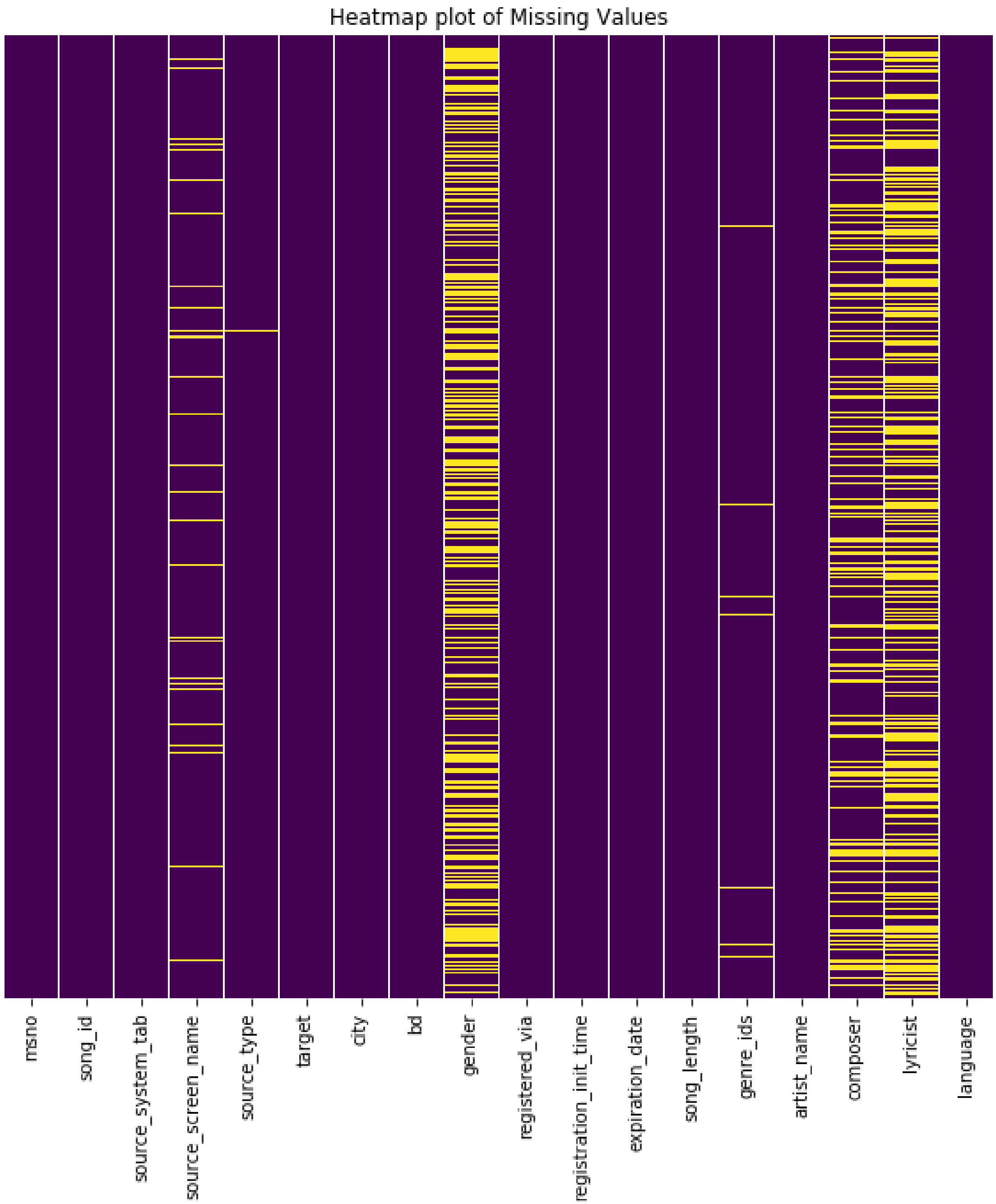
[9]:
```python
df.describe()
```

[9]:
```
             target          city            bd  registered_via  \
count  7.377418e+06  7.377418e+06  7.377418e+06    7.377418e+06
mean   5.035171e-01  7.511399e+00  1.753927e+01    6.794068e+00
std    4.999877e-01  6.641624e+00  2.155447e+01    2.275774e+00
min    0.000000e+00  1.000000e+00 -4.300000e+01    3.000000e+00
25%    0.000000e+00  1.000000e+00  0.000000e+00    4.000000e+00
50%    1.000000e+00  5.000000e+00  2.100000e+01    7.000000e+00
75%    1.000000e+00  1.300000e+01  2.900000e+01    9.000000e+00
max    1.000000e+00  2.200000e+01  1.051000e+03    1.300000e+01

       registration_init_time  expiration_date   song_length      language
count            7.377418e+06     7.377418e+06  7.377304e+06  7.377268e+06
```

```
mean          2.012810e+07    2.017157e+07  2.451210e+05  1.860933e+01
std           3.017281e+04    3.869831e+03  6.734471e+04  2.117681e+01
min           2.004033e+07    1.970010e+07  1.393000e+03 -1.000000e+00
25%           2.011070e+07    2.017091e+07  2.147260e+05  3.000000e+00
50%           2.013102e+07    2.017093e+07  2.418120e+05  3.000000e+00
75%           2.015102e+07    2.017101e+07  2.721600e+05  5.200000e+01
max           2.017013e+07    2.020102e+07  1.085171e+07  5.900000e+01
```

[6]:
```python
import re
df['artist_name_processed'] = df['artist_name'].astype(str).apply(lambda x: ' '.
 ↪join(re.sub('[^a-zA-Z ]',' ', x).lower().split()[:3]))
obj = df['artist_name_processed'].astype('category').cat
artist_map = dict(enumerate(obj.categories))
df['artist_name_processed'] = obj.codes
```

[10]:
```python
plt.figure(figsize = (10, 10))
sns.heatmap(df.isnull(), cbar = False, cmap = 'viridis', yticklabels = False)
plt.title('Heatmap plot of Missing Values')
plt.show()
```

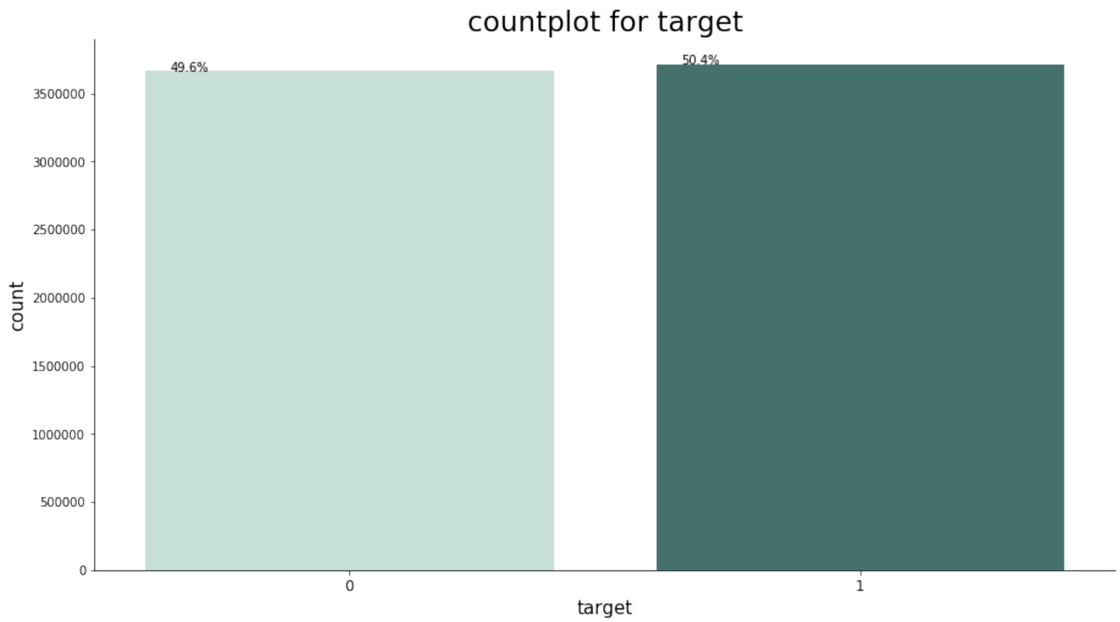


Conclusion : Gender, composer and lyricist are highly sparsed ie. they have high missing value rate. So, we need to handle them carefully.

1.1 EDA Of Categorical Variables

[209]:
```
a = pd.DataFrame(df.groupby('target').size().reset_index())
a = a.sort_values(by = 0, ascending = False)
```

```
plt.figure(figsize = (15,8))
ax = sns.barplot(x= 'target',y = 0, data = a, palette='ch:2.5,-.10,dark=.4')
plt.xticks(fontsize = 12)
plt.xlabel('target', fontdict = {'fontsize':15})
plt.ylabel('count', fontdict = {'fontsize':15})
plt.title('countplot for target', fontdict = {'fontsize': 23})

for p in ax.patches:
        ax.annotate('{:.1f}%'.format( 100*p.get_height()/df.shape[0]), (p.
 ↪get_x()+0.05, p.get_height()+0.5), fontsize = 10)
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
plt.show()
```

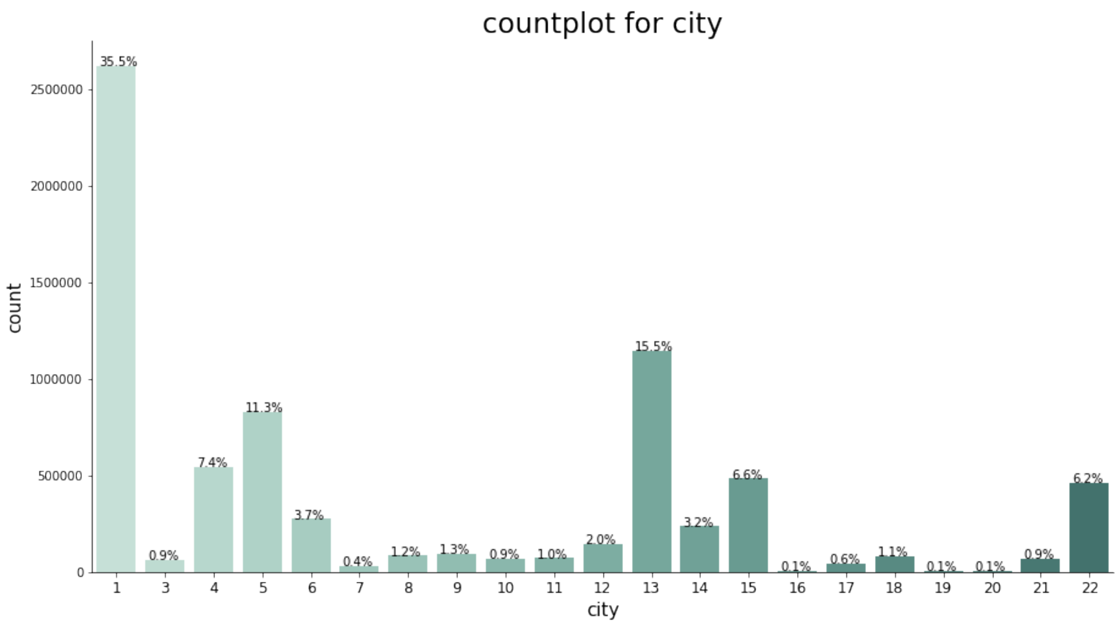


Aim :Plot for determining the distribution of Target variable.

Conclusion : Dataset is balanced for taget variable.

[123]:
```
a = pd.DataFrame(df.groupby('city').size().reset_index())
a = a.sort_values(by = 0, ascending = False)

plt.figure(figsize = (15,8))
ax = sns.barplot(x= 'city',y = 0, data = a, palette='ch:2.5,-.10,dark=.4')
plt.xticks(fontsize = 12)
plt.xlabel('city', fontdict = {'fontsize':15})
plt.ylabel('count', fontdict = {'fontsize':15})
```

```
plt.title('countplot for city', fontdict = {'fontsize': 23})

for p in ax.patches:
        ax.annotate('{:.1f}%'.format( 100*p.get_height()/df.shape[0]), (p.
 ↪get_x()+0.05, p.get_height()+0.5), fontsize = 10)

ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
plt.show()
```

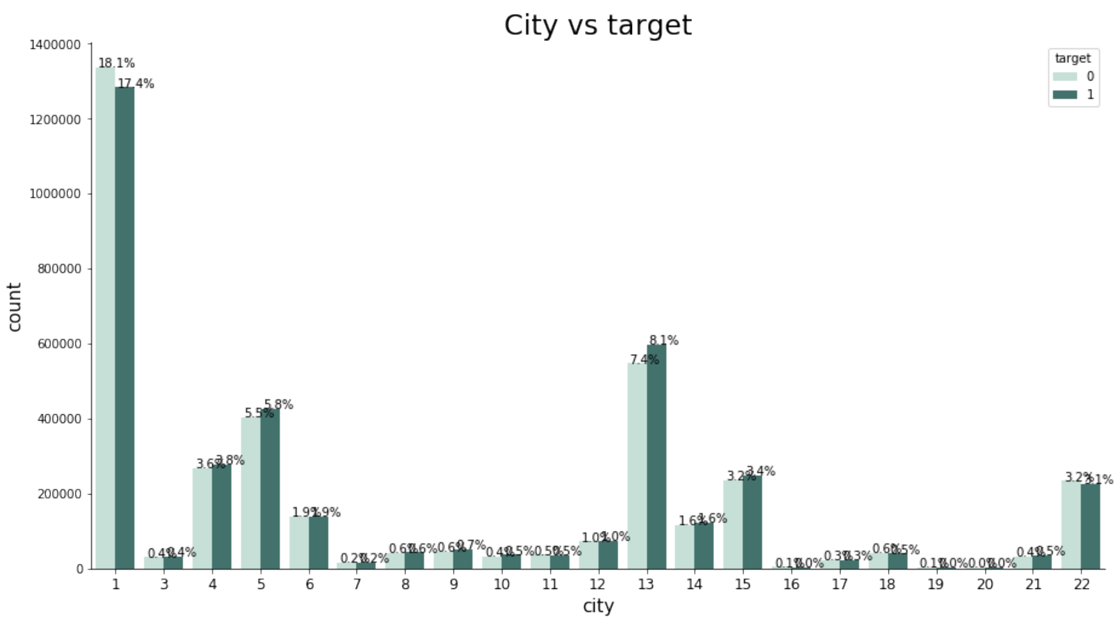


Aim for city :Plot for determining the distribution of users according to cities.

Conclusion for city : 1. There are majority of users using KKBox app from city 1 and 13. 2. City 1 and 13 must be cities with high population as there are more users in city.

[8]:
```
a = pd.DataFrame(df.groupby(['city', 'target']).size().reset_index())
a = a.sort_values(by = 0, ascending = False)

plt.figure(figsize = (15,8))
ax = sns.barplot(x= 'city',y = 0, data = a, hue = 'target', palette='ch:2.5,-.
 ↪10,dark=.4')
plt.xticks(fontsize = 12)
plt.xlabel('city', fontdict = {'fontsize':15})
plt.ylabel('count', fontdict = {'fontsize':15})
plt.title('City vs target', fontdict = {'fontsize': 23})

for p in ax.patches:
```

```
        ax.annotate('{:.1f}%'.format( 100*p.get_height()/df.shape[0]), (p.
↪get_x()+0.05, p.get_height()+0.5), fontsize = 10)

ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
plt.show()
```

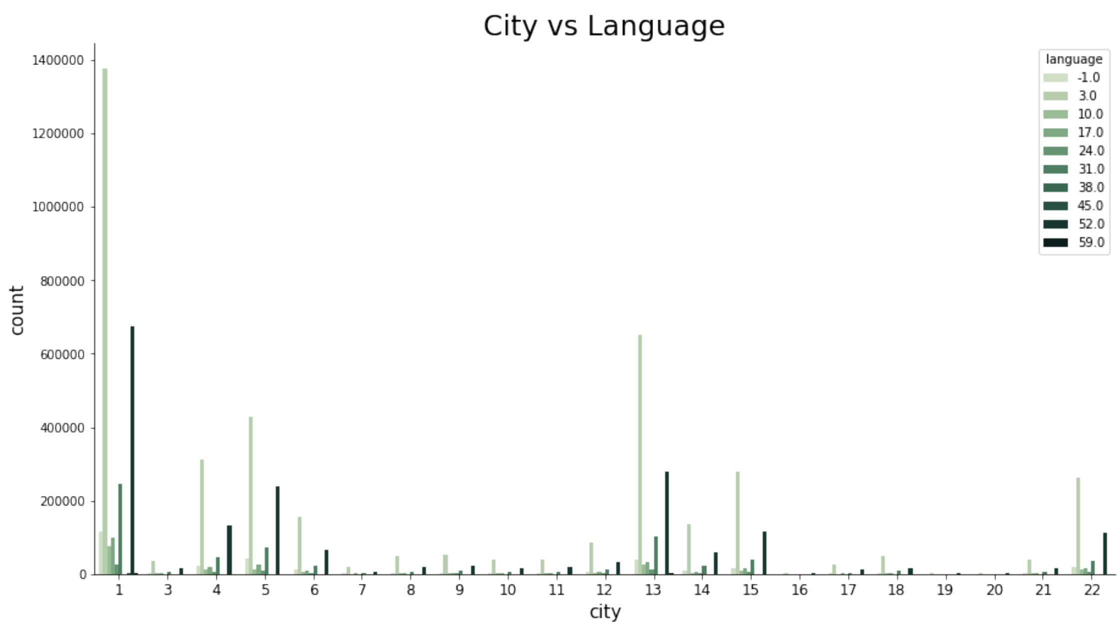


[25]:
```
a = pd.DataFrame(df.groupby(['city', 'registered_via']).size().reset_index())
a = a.sort_values(by = 0, ascending = False)

plt.figure(figsize = (15,8))
ax = sns.barplot(x= 'city',y = 0, data = a, hue = 'registered_via', palette='ch:
↪2.5,-.10,dark=.4')
plt.xticks(fontsize = 12)
plt.xlabel('city', fontdict = {'fontsize':15})
plt.ylabel('count', fontdict = {'fontsize':15})
plt.title('City vs Registered via', fontdict = {'fontsize': 23})

ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
plt.show()
```

Aim for city vs registered via :Plot for determining the distribution of users according to cities and mode through which they have registered.

Conclusion for city vs registered via: 1. Except city 1 all cities mostly prefer to registration mode as 13 only city 1 prefer registration mode 7.

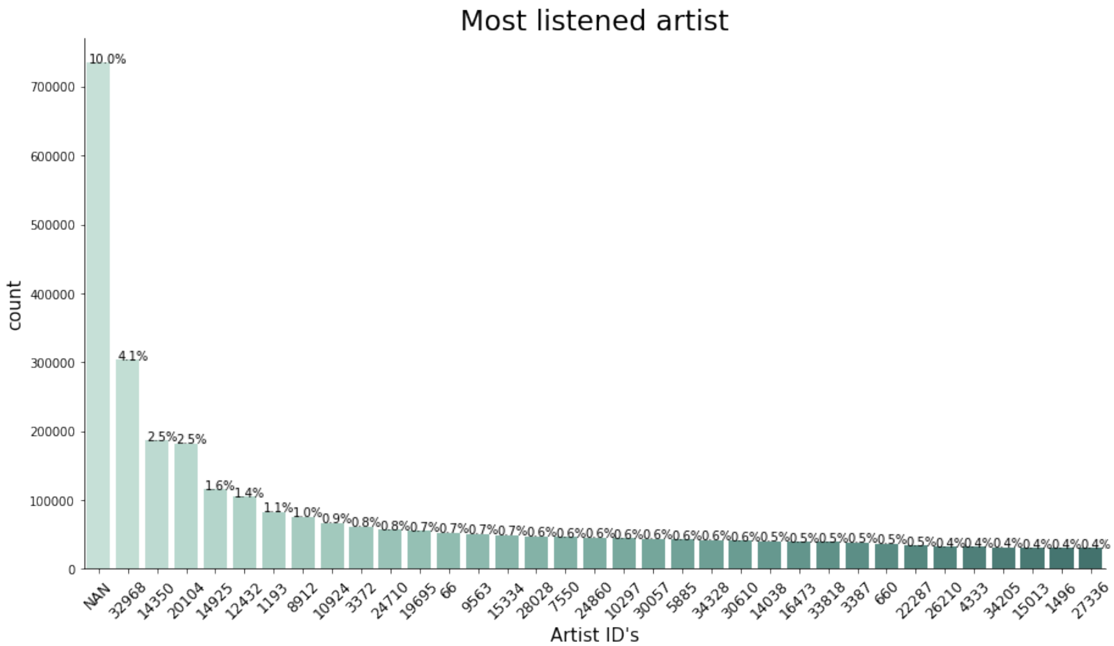
[28]:
```
a = pd.DataFrame(df.groupby(['city', 'language']).size().reset_index())
a = a.sort_values(by = 0, ascending = False)

plt.figure(figsize = (15,8))
ax = sns.barplot(x= 'city',y = 0, data = a, hue = 'language', palette='ch:2.5,-.
↪30,dark=.1')
plt.xticks(fontsize = 12)
plt.xlabel('city', fontdict = {'fontsize':15})
plt.ylabel('count', fontdict = {'fontsize':15})
plt.title('City vs Language', fontdict = {'fontsize': 23})

ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
plt.show()
```



Aim for city vs Language :Plot for determining the most prefered languages of city..

Conclusion for city vs Language: 1. Most popular two languages are 3.0 and 59.0 . 2. All cities belongs to same country.

[78]:
```
a = pd.DataFrame(df.groupby('artist_name_processed').size().reset_index())
a = a.sort_values(by = 0, ascending = False)
a = a.iloc[:35, :]
a.replace(0, 'NAN', inplace = True)
plt.figure(figsize = (15,8))
ax = sns.barplot(x= 'artist_name_processed',y = 0, data = a, palette='ch:2.5,-.
↪10,dark=.4')
plt.xticks(rotation= 45, fontsize = 12)
plt.xlabel('Artist ID\'s', fontdict = {'fontsize':15})
plt.ylabel('count', fontdict = {'fontsize':15})
plt.title('Most listened artist', fontdict = {'fontsize': 23})

for p in ax.patches:
        ax.annotate('{:.1f}%'.format( 100*p.get_height()/df.shape[0]), (p.
↪get_x()+0.05, p.get_height()+0.5), fontsize = 10)

ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)

plt.show()
```



[86]:
```
from prettytable import PrettyTable

myTable = PrettyTable(["Artist ID", "Artist Name", "Artist  ID ", "Artist ␣
↪Name"])
a = np.array(a)
for i in range(1, a.shape[0], 2):
    myTable.add_row([a[i][0], artist_map[a[i][0]], a[i+1][0],␣
↪artist_map[a[i+1][0]]])
print(myTable)
```

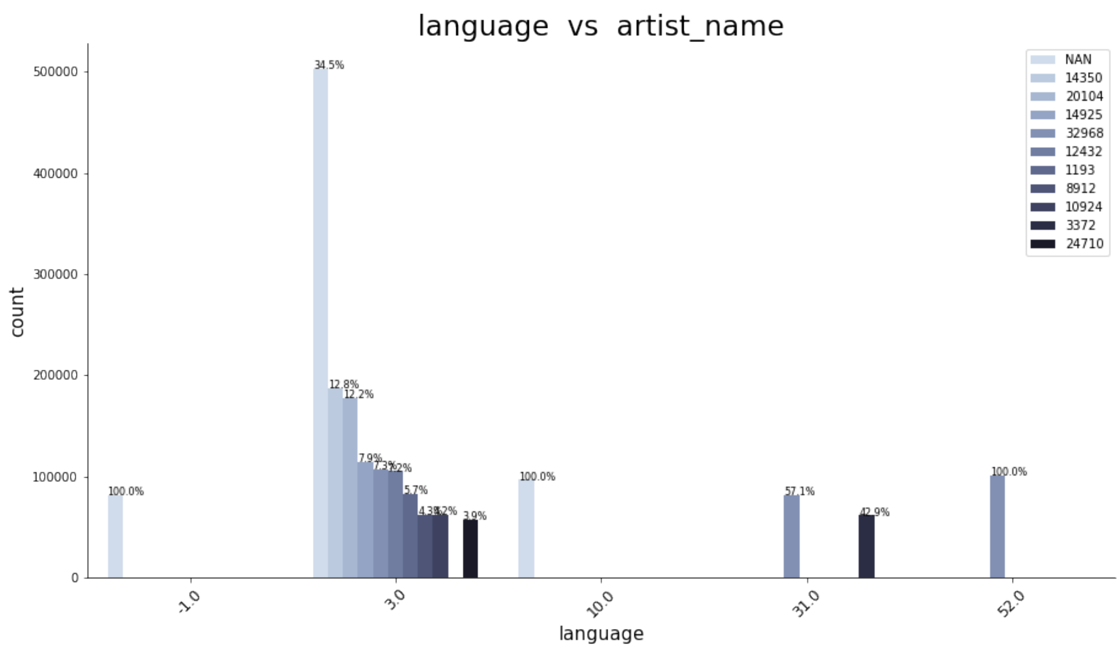
```
+-----------+------------------+-------------+--------------+
| Artist ID |    Artist Name   | Artist  ID  | Artist  Name |
+-----------+------------------+-------------+--------------+
|   32968   | various artists  |    14350    |   jay chou   |
|   20104   |      mayday      |    14925    |    jj lin    |
|   12432   |       hebe       |     1193    |     amei     |
|    8912   |    eason chan    |    10924    |    g e m     |
|    3372   |     bigbang      |    24710    |   r chord    |
|   19695   |      maroon      |      66     |    a lin     |
|    9563   |       eric       |    15334    |  jolin tsai  |
|   28028   |    sodagreen     |     7550    |    della     |
|   24860   |   rainie yang    |    10297    |  fish leong  |
|   30057   | the chainsmokers |     5885    |  claire kuo  |
|   34328   |     yoga lin     |    30610    | the last day |
|   14038   |    jam hsiao     |    16473    |   kenji wu   |
|   33818   |   william wei    |     3387    |     bii      |
```

```
|    660    |   alan walker    |    22287    | nickthereal  |
|   26210   |      s h e       |     4333    |  bruno mars  |
|   34205   |    yanzi sun     |    15013    |  jody jiang  |
|    1496   |    andrew tan    |    27336    |   shi shi    |
+-----------+------------------+-------------+--------------+
```

Aim for Artist Name :Plot for determining the most Listened artist.

Conclusion for Artist Name: 1. Most popular two Artist are jay chou and mayday. 2. There are 10% of artist names are missing. 3. 4.1% of Artists are unknown with given name as "various artists".

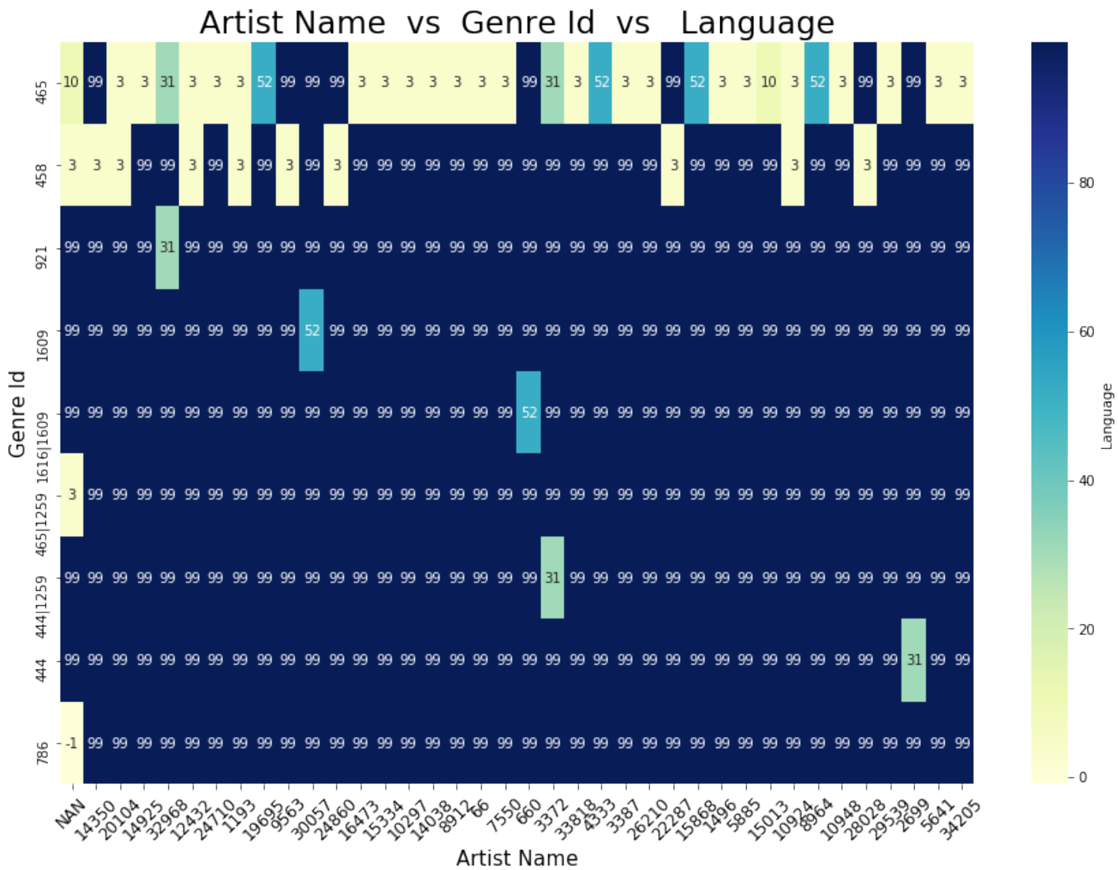
[53]:
```
a = pd.DataFrame(df.groupby(['language', 'artist_name_processed']).size().
↪reset_index())
a = a.sort_values(by = 0, ascending = False)
a = a.iloc[:15, :]
a.replace(0, 'NAN', inplace = True)
b = np.array(a.groupby('language')[0].sum().reset_index())
plt.figure(figsize = (15,8))
ax = sns.barplot(hue= 'artist_name_processed',y = 0, data = a, x = 'language',␣
↪palette='ch:30.0,-.10,dark=.10')
plt.xticks(rotation = 45, fontsize = 12)
plt.xlabel('language', fontdict = {'fontsize':15})
plt.ylabel('count', fontdict = {'fontsize':15})
plt.title('language  vs  artist_name', fontdict = {'fontsize': 23})
plt.legend(loc = 'upper right')
count = 0
for p in ax.patches:
    if count == b.shape[0]:
            count = 0
    ax.annotate('{:.1f}%'.format( 100*p.get_height()/b[count, 1]), (p.get_x(),␣
↪p.get_height()+0.5), fontsize = 8)
    count+=1

ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
plt.show()
```



Aim for Artist Name vs Language :Plot for determining the Most popular artists in Language.

Conclusion for Artist Name vs Language: 1. jay chou and mayday are two most popular singers in language 3.0 2. bigbang is the most popular singers in language 31.0 3. Maroon is the most popular artist in language 52.0 4. All the artist names in language -1.0 and 10.0 are missing.

[12]:
```
a = pd.DataFrame(df.groupby(['language', 'genre_ids', 'artist_name_processed']).size().reset_index())
a = a.sort_values(by = 0, ascending = False)
a = a.iloc[:50, :]
a.replace(0, 'NAN', inplace = True)
f = pd.DataFrame(columns = a['genre_ids'].unique(), index = a['artist_name_processed'].unique())
a = np.array(a)
for i in a:
    f[i[1]].loc[i[2]] = i[0]
f.replace(np.nan, 99, inplace = True)
plt.figure(figsize = (15, 10))
sns.heatmap(f.T, annot = True, cmap = 'YlGnBu',cbar_kws={'label': 'Language'})
plt.xticks(rotation = 45, fontsize = 12)
plt.xlabel('Artist Name', fontdict = {'fontsize':15})
plt.ylabel('Genre Id', fontdict = {'fontsize':15})
plt.title('Artist Name  vs  Genre Id  vs   Language', fontdict = {'fontsize':23})
plt.show()
```

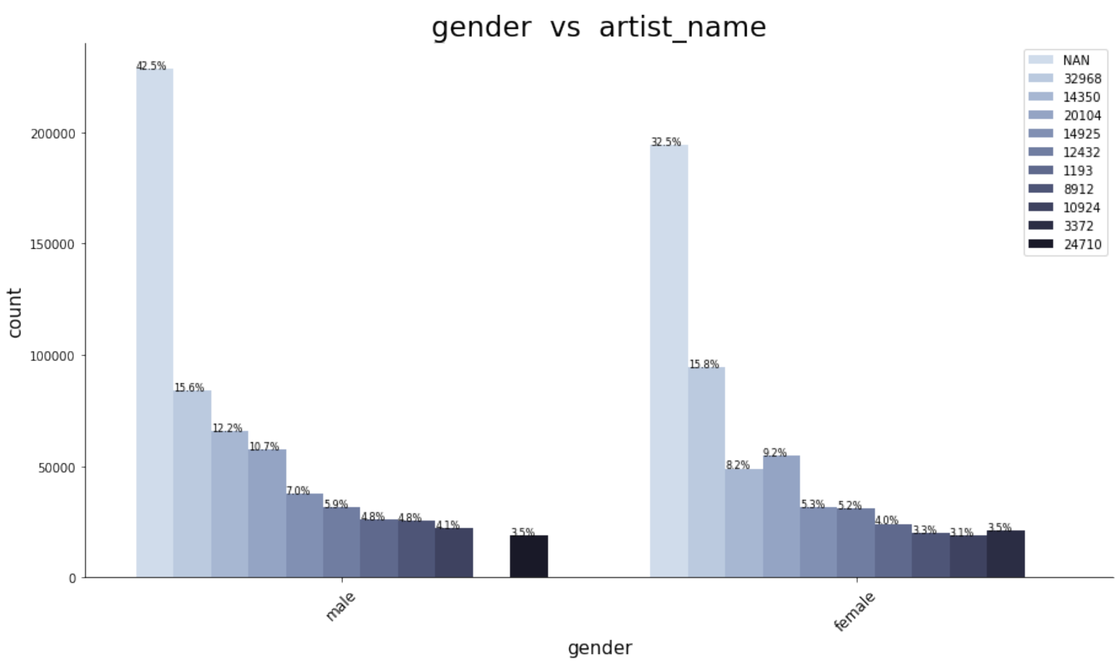


Aim for Artist Name vs Genrr ID vs Language :Plot for determining the Most popular artists of particular genre in particular Language.

Conclusion for Artist Name vs Genre ID vs Language: 1. Jay Chou sings only with genre id 458 and language 3.0 . 2. mayday sings only in 458 and 465 genre with only language 3.0 . 3. The Chainsmoker is the only artist which uses genre id 1609 with language 52.0 4. Alan Walker is the only artist which uses genre id 1616 and 1609 with languages 52.0.

[18]:
```
a = pd.DataFrame(df.groupby(['gender', 'artist_name_processed']).size().
↪reset_index())
a = a.sort_values(by = 0, ascending = False)
a = a.iloc[:20, :]
a.replace(0, 'NAN', inplace = True)
b = np.array(a.groupby('gender')[0].sum().reset_index())
plt.figure(figsize = (15,8))
ax = sns.barplot(hue= 'artist_name_processed',y = 0, data = a, x = 'gender',␣
↪palette='ch:30.0,-.10,dark=.10')
plt.xticks(rotation = 45, fontsize = 12)
plt.xlabel('gender', fontdict = {'fontsize':15})
```

```
plt.ylabel('count', fontdict = {'fontsize':15})
plt.title('gender  vs  artist_name', fontdict = {'fontsize': 23})
plt.legend(loc = 'upper right')
count = 0
for p in ax.patches:
    if count == b.shape[0]:
            count = 0
    ax.annotate('{:.1f}%'.format( 100*p.get_height()/b[count, 1]), (p.get_x(),␣
↪p.get_height()+0.5), fontsize = 8)
    count+=1

ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
plt.show()
```

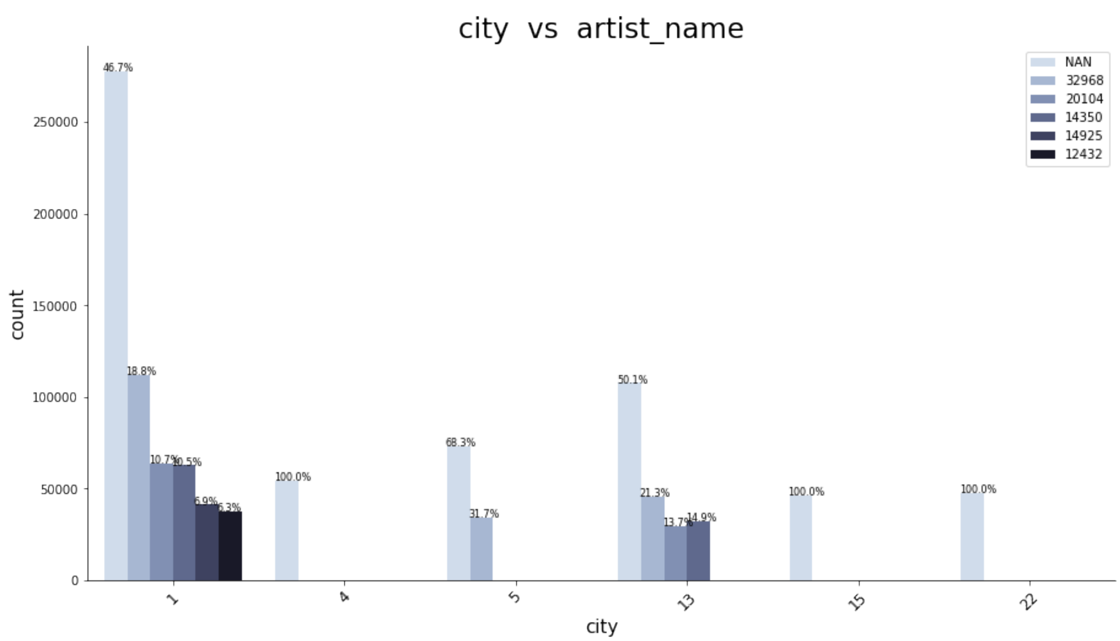


Aim for Artist Name vs Gender :Plot for determining the Most popular artists among males and females.

Conclusion for Artist Name vs Gender : 1. jay chou is more popular in males than in females. 2. Bigbang is only popular in males and not in females. 3. Sodagreen is only popular in females and not in males.

[54]:
```
a = pd.DataFrame(df.groupby(['city', 'artist_name_processed']).size().
↪reset_index())
a = a.sort_values(by = 0, ascending = False)
a = a.iloc[:15, :]
```

```python
a.replace(0, 'NAN', inplace = True)
b = np.array(a.groupby('city')[0].sum().reset_index())
plt.figure(figsize = (15,8))
ax = sns.barplot(hue= 'artist_name_processed',y = 0, data = a, x = 'city', palette='ch:30.0,-.10,dark=.10')
plt.xticks(rotation = 45, fontsize = 12)
plt.xlabel('city', fontdict = {'fontsize':15})
plt.ylabel('count', fontdict = {'fontsize':15})
plt.title('city  vs  artist_name', fontdict = {'fontsize': 23})
plt.legend(loc = 'upper right')
count = 0
for p in ax.patches:
    if count == b.shape[0]:
            count = 0
    ax.annotate('{:.1f}%'.format( 100*p.get_height()/b[count, 1]), (p.get_x()-0.01, p.get_height()+1), fontsize = 8)
    count+=1

ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
plt.show()
```



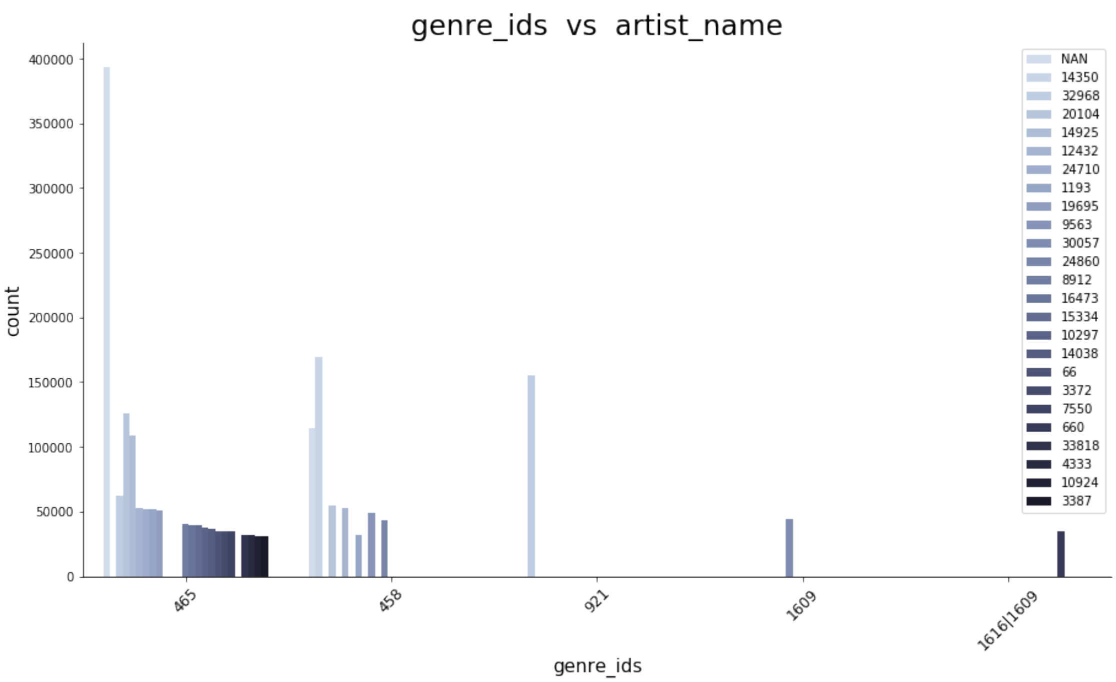
Aim for Artist Name vs City :Plot for determining the Most popular artists in City.

Conclusion for Artist Name vs City: 1. jay chou and mayday are two most popular artist in city 1 and 13. 2. All the artist names in city 4, 6, 15 and 22 are missing.

```
[37]: a = pd.DataFrame(df.groupby(['genre_ids', 'artist_name_processed']).size().
      ↪reset_index())
      a = a.sort_values(by = 0, ascending = False)
      a = a.iloc[:30, :]
      a.replace(0, 'NAN', inplace = True)
      b = np.array(a.groupby('genre_ids')[0].sum().reset_index().sort_values(by = 0,␣
      ↪ascending = False))
      plt.figure(figsize = (15,8))
      ax = sns.barplot(hue= 'artist_name_processed',y = 0, data = a, x = 'genre_ids',␣
      ↪palette='ch:30.0,-.10,dark=.10')
      plt.xticks(rotation = 45, fontsize = 12)
      plt.xlabel('genre_ids', fontdict = {'fontsize':15})
      plt.ylabel('count', fontdict = {'fontsize':15})
      plt.title('genre_ids  vs  artist_name', fontdict = {'fontsize': 23})
      plt.legend(loc = 'upper right')

      ax.spines['right'].set_visible(False)
      ax.spines['top'].set_visible(False)
      from prettytable import PrettyTable

      myTable = PrettyTable(["Artist ID", "Artist Name", "Genre ID", "Count", "Artist␣
      ↪ ID ", "Artist  Name",  "Genre  ID", "Count "])
      a = np.array(a)
      for i in range(1, a.shape[0], 2):
          try:
              myTable.add_row([a[i][1], artist_map[a[i][1]], a[i][0], a[i][2],␣
      ↪a[i+1][1], artist_map[a[i+1][1]], a[i+1][0], a[i+1][2]])
          except:
              pass
      print(myTable)

      plt.show()
```

```
+-----------+-------------+----------+--------+-------------+-----------------+
-----------+--------+
| Artist ID | Artist Name | Genre ID | Count  | Artist  ID  |   Artist  Name  |
Genre  ID | Count  |
+-----------+-------------+----------+--------+-------------+-----------------+
-----------+--------+
|   14350   |   jay chou  |   458    | 168959 |    32968    | various artists |
921     | 154734 |
|   14925   |    jj lin   |   465    | 108495 |    32968    | various artists |
465     | 61923  |
|   20104   |    mayday   |   458    | 54941  |    12432    |       hebe      |
458     | 52653  |
|   12432   |     hebe    |   465    | 52293  |    24710    |     r chord     |
465     | 51700  |
```

1193	amei	465	51292	19695	maroon	465	50546
9563	eric	458	48721	30057	the chainsmokers	1609	44149
24860	rainie yang	458	43097	8912	eason chan	465	40472
16473	kenji wu	465	39312	15334	jolin tsai	465	39065
10297	fish leong	465	37279	14038	jam hsiao	465	36853
66	a lin	465	34880	3372	bigbang	465	34565
7550	della	465	34508	660	alan walker	1616|1609	34480
33818	william wei	465	31995	4333	bruno mars	465	31817
1193	amei	458	31507	10924	g e m	465	31321

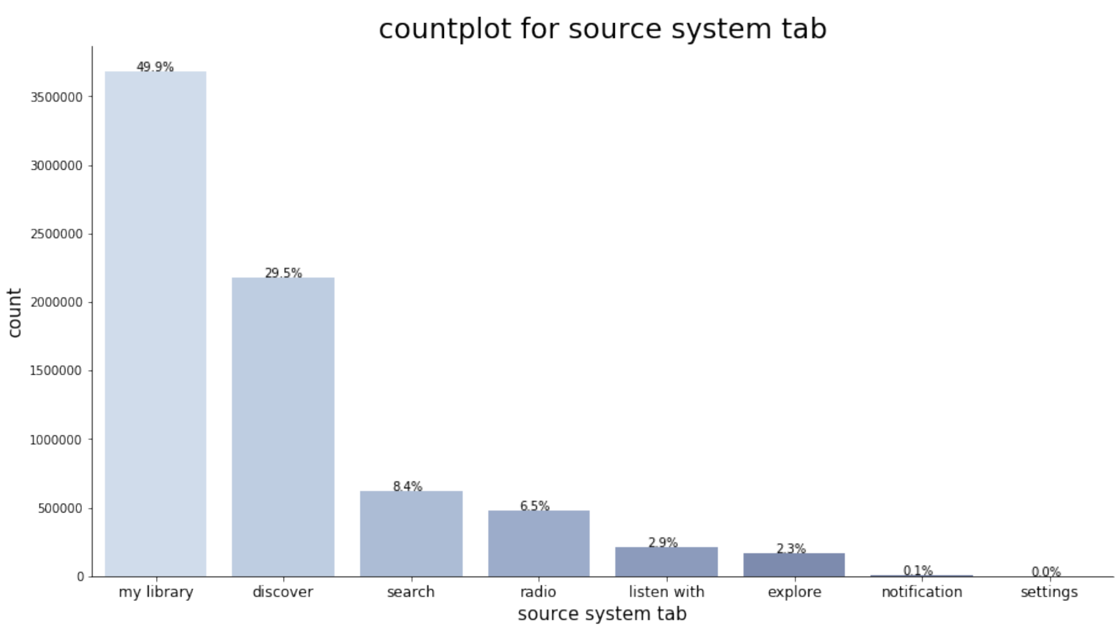


Aim for Artist Name vs genre ids :Plot for determining the Most popular artists among genre.

Conclusion for Artist Name vs genre ids: 1. mayday and jj lin are most popular artists in genre id 465. 2. jay chou and mayday are most popular artists in genre id 458. 3.

The Chainsmoker is the only popular artist in genre id 1609. 4. Alan walker is popular among genre id 1616 and 1609.

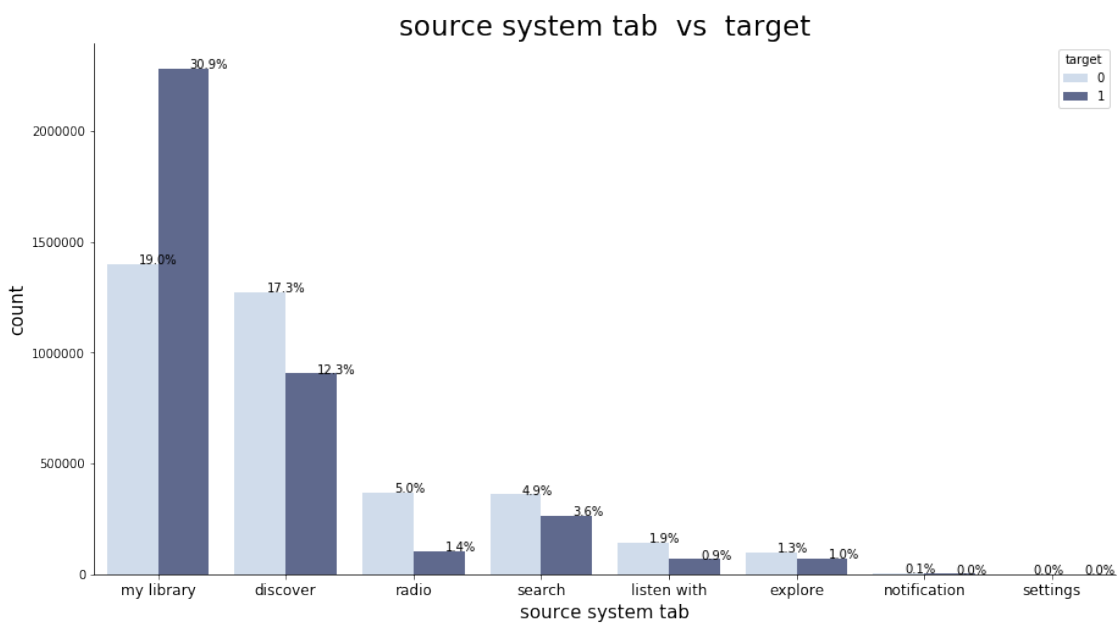
[192]:
```
a = pd.DataFrame(df.groupby('source_system_tab').size().reset_index())
a = a.sort_values(by = 0, ascending = False)

plt.figure(figsize = (15,8))
ax = sns.barplot(x= 'source_system_tab',y = 0, data = a, palette='ch:30.0,-.
↪10,dark=.4')
plt.xticks(fontsize = 12)
plt.xlabel('source system tab', fontdict = {'fontsize':15})
plt.ylabel('count', fontdict = {'fontsize':15})
plt.title('countplot for source system tab', fontdict = {'fontsize': 23})

for p in ax.patches:
        ax.annotate('{:.1f}%'.format( 100*p.get_height()/df.shape[0]), (p.
↪get_x()+0.25, p.get_height()+0.5), fontsize = 10)
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
plt.show()
```



Aim for source system tab :Plot for determining the distribution of Source system tab used by users in KKBox app while listening songs.

Conclusion for source system tab : 1.There are majority of users using KKBox app by using local storage ie my library. 2. Most of the peoples discover songs using discover, search then they try to use there my library for playing songs.

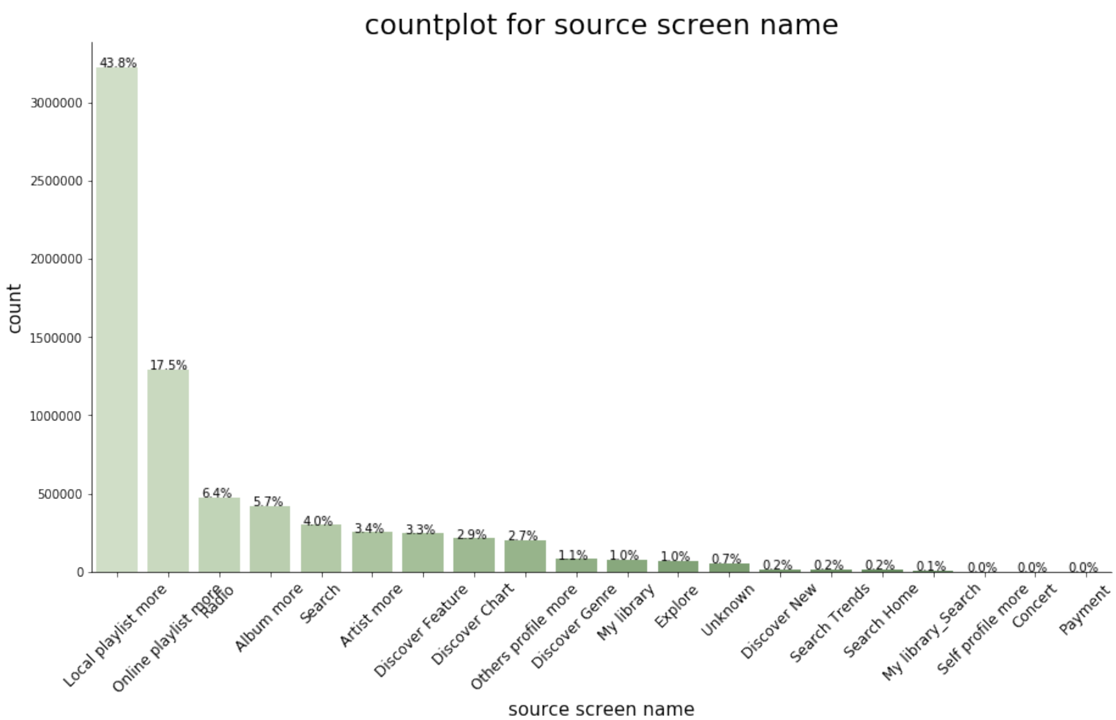
[33]:
```
a = pd.DataFrame(df.groupby(['source_system_tab', 'target']).size().
 ↪reset_index())
a = a.sort_values(by = 0, ascending = False)

plt.figure(figsize = (15,8))
ax = sns.barplot(x= 'source_system_tab',y = 0, data = a, hue = 'target',␣
 ↪palette='ch:30.0,-.10,dark=.4')
plt.xticks(fontsize = 12)
plt.xlabel('source system tab', fontdict = {'fontsize':15})
plt.ylabel('count', fontdict = {'fontsize':15})
plt.title('source system tab  vs  target', fontdict = {'fontsize': 23})

for p in ax.patches:
        ax.annotate('{:.1f}%'.format( 100*p.get_height()/df.shape[0]), (p.
 ↪get_x()+0.25, p.get_height()+0.5), fontsize = 10)
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
plt.show()
```



Aim for target vs source system tab : Plot for determining the distribution of Source system tab used by users in KKBox app while listening songs with respect to variation in target variable.

Conclusion for target vs source system tab : 1. 30.9% of users listening songs repeatedly from my library. 2. 17.3% of users discovered songs where not listend more than once.

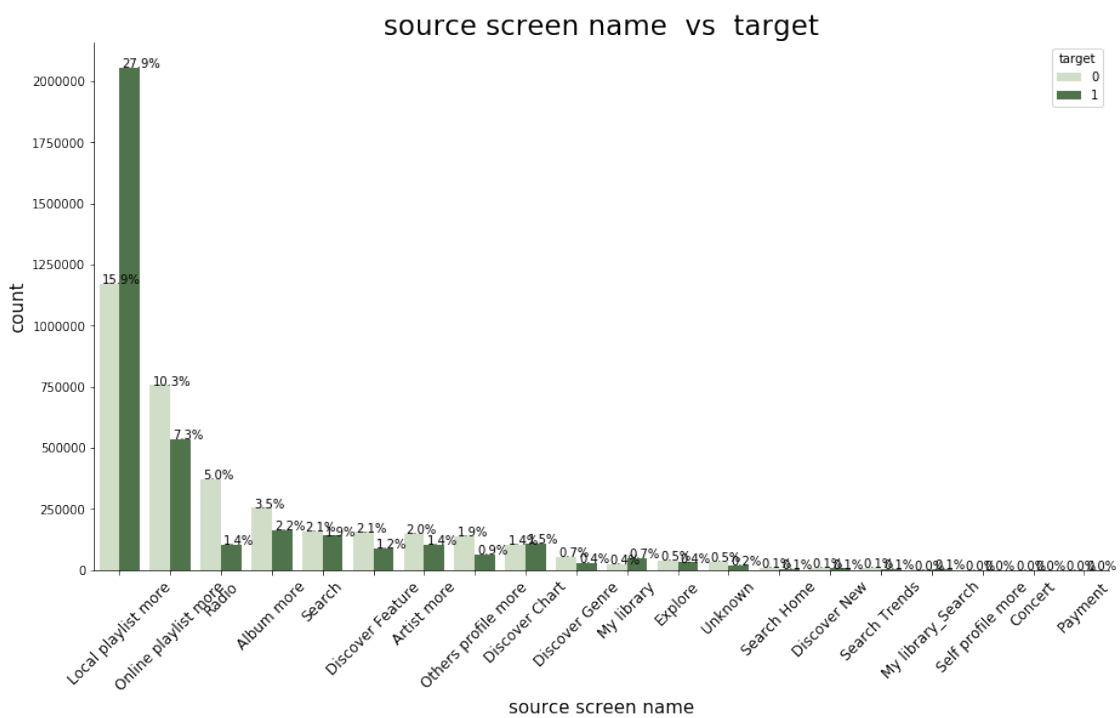
```
[137]: a = pd.DataFrame(df.groupby('source_screen_name').size().reset_index())
a = a.sort_values(by = 0, ascending = False)

plt.figure(figsize = (15,8))
ax = sns.barplot(x= 'source_screen_name',y = 0, data = a, palette='ch:23.0,-.
 ↪10,dark=.4')
plt.xticks(rotation = 45, fontsize = 12)
plt.xlabel('source screen name', fontdict = {'fontsize':15})
plt.ylabel('count', fontdict = {'fontsize':15})
plt.title('countplot for source screen name', fontdict = {'fontsize': 23})

for p in ax.patches:
        ax.annotate('{:.1f}%'.format( 100*p.get_height()/df.shape[0]), (p.
 ↪get_x()+0.05, p.get_height()+0.5), fontsize = 10)
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
plt.show()
```



Aim for source screen name : Plot for determining the distribution of Source screen name used by users in KKBox app while listening songs.

Conclusion for source screen name : 1. 43.8% of the users are using local playlist for playing songs. 2. we can also conclude that most of the users are specific about there songs as they don't even use my library search.

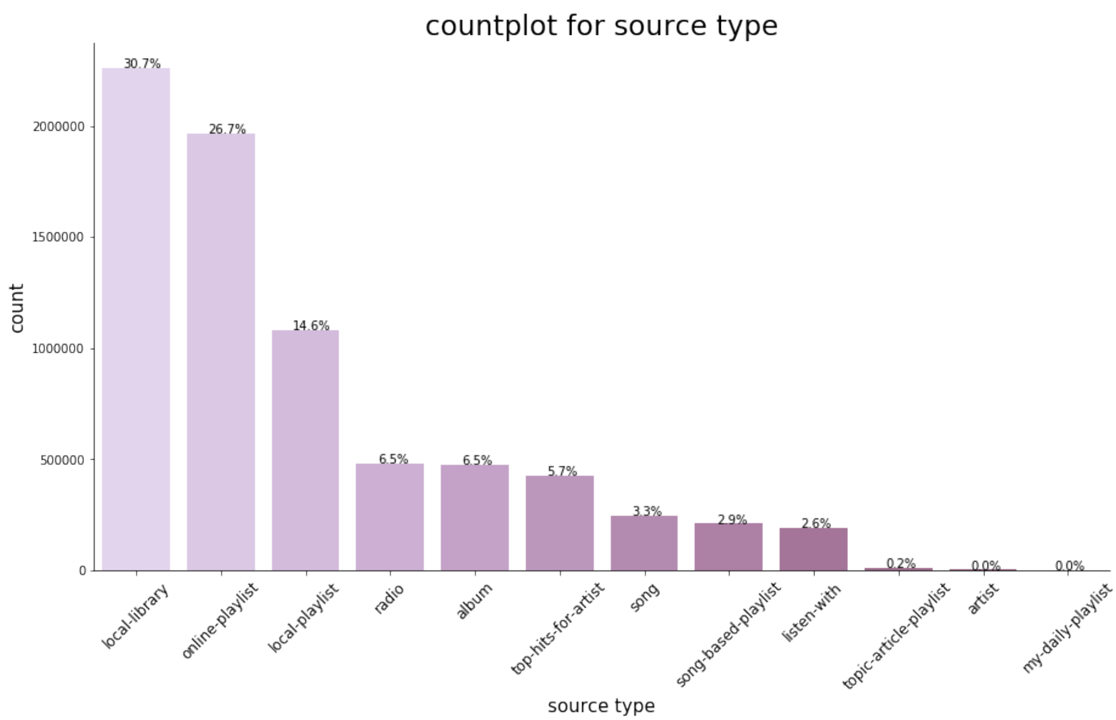
```python
[53]: a = pd.DataFrame(df.groupby(['source_screen_name', 'target']).size().
       ↪reset_index())
      a = a.sort_values(by = 0, ascending = False)

      plt.figure(figsize = (15,8))
      ax = sns.barplot(x= 'source_screen_name',y = 0, hue = 'target', data = a,␣
       ↪palette='ch:23.0,-.10,dark=.4')
      plt.xticks(rotation = 45, fontsize = 12)
      plt.xlabel('source screen name', fontdict = {'fontsize':15})
      plt.ylabel('count', fontdict = {'fontsize':15})
      plt.title('source screen name  vs  target', fontdict = {'fontsize': 23})

      for p in ax.patches:
              ax.annotate('{:.1f}%'.format( 100*p.get_height()/df.shape[0]), (p.
       ↪get_x()+0.05, p.get_height()+0.5), fontsize = 10)
      ax.spines['right'].set_visible(False)
      ax.spines['top'].set_visible(False)
      plt.show()
```



Aim for source screen name : Plot for determining the distribution of Source screen name used by users in KKBox app while listening songs with respect to variation in target variable.

Conclusion for source screen name vs target : 1. 27.9% of users listening songs repeatedly from Local Playlist. 2. 5.0% of users Listening songs from radio where not listend more than once.

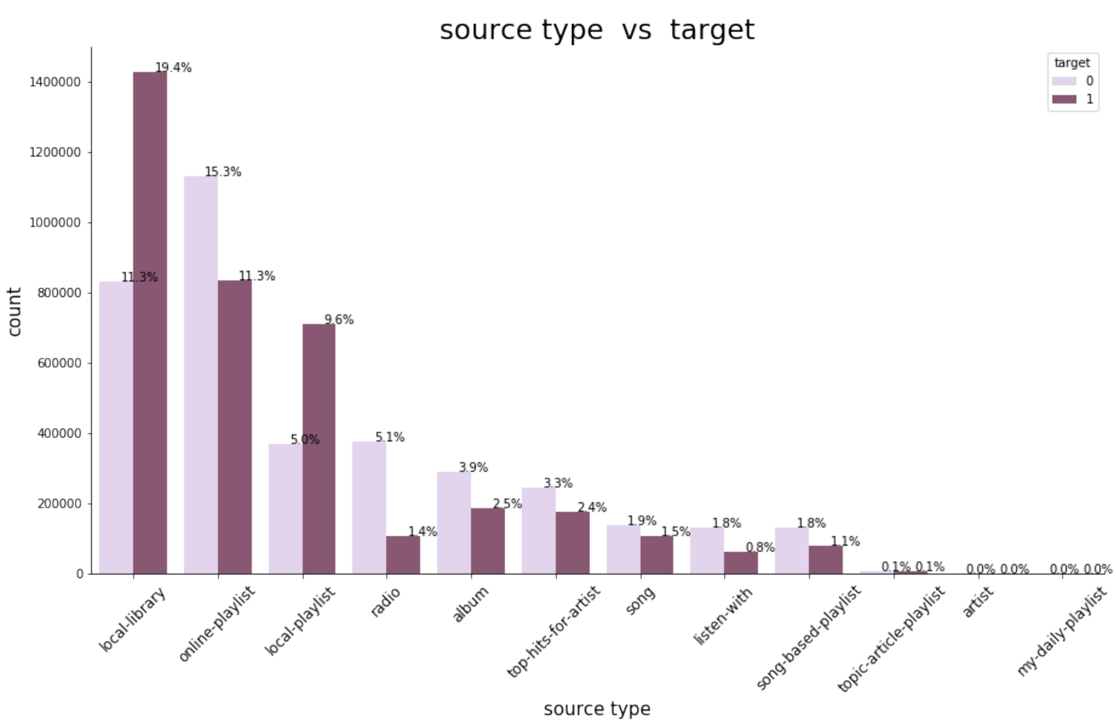
```
[200]: a = pd.DataFrame(df.groupby('source_type').size().reset_index())
a = a.sort_values(by = 0, ascending = False)

plt.figure(figsize = (15,8))
ax = sns.barplot(x= 'source_type',y = 0, data = a, palette='ch:4.0,-.30,dark=.
 ↪4')
plt.xticks(rotation = 45, fontsize = 12)
plt.xlabel('source type', fontdict = {'fontsize':15})
plt.ylabel('count', fontdict = {'fontsize':15})
plt.title('countplot for source type', fontdict = {'fontsize': 23})

for p in ax.patches:
        ax.annotate('{:.1f}%'.format( 100*p.get_height()/df.shape[0]), (p.
 ↪get_x()+0.25, p.get_height()+0.5), fontsize = 10)
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
plt.show()
```



Aim for source type : Plot for determining the distribution of Source type used by users in KKBox app while listening songs.

Conclusion for source system type : 1. 72% of users are using app using local-library, online-playlist and local-playlist.

```
[55]: a = pd.DataFrame(df.groupby(['source_type', 'target']).size().reset_index())
a = a.sort_values(by = 0, ascending = False)

plt.figure(figsize = (15,8))
ax = sns.barplot(x= 'source_type',y = 0, data = a, hue = 'target', palette='ch:
 ↪4.0,-.30,dark=.4')
plt.xticks(rotation = 45, fontsize = 12)
plt.xlabel('source type', fontdict = {'fontsize':15})
plt.ylabel('count', fontdict = {'fontsize':15})
plt.title('source type  vs  target', fontdict = {'fontsize': 23})

for p in ax.patches:
        ax.annotate('{:.1f}%'.format( 100*p.get_height()/df.shape[0]), (p.
 ↪get_x()+0.25, p.get_height()+0.5), fontsize = 10)
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
plt.show()
```

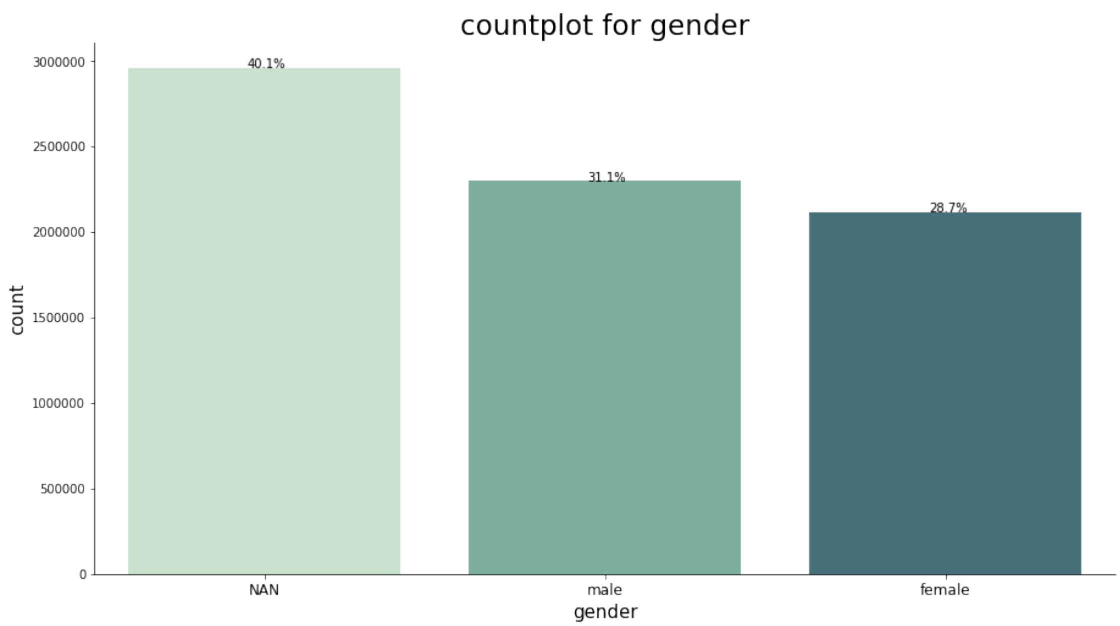


Aim for source type vs target : Plot for determining the distribution of Source type used by users in KKBox app while listening songs with respect to variation in target variable.

Conclusion for source type vs target : 1. 19.4% of users listening songs repeatedly from Local library. 2. 5.1% of users Listening songs from radio where not listend more than once.

[204]:

```
a = df['gender']
a.replace(np.nan, 'NAN',inplace = True)
a = pd.DataFrame(df.groupby('gender').size().reset_index())
a = a.sort_values(by = 0, ascending = False)

plt.figure(figsize = (15,8))
ax = sns.barplot(x= 'gender',y = 0, data = a, palette='ch:3.0,-.40,dark=.4')
plt.xticks( fontsize = 12)
plt.xlabel('gender', fontdict = {'fontsize':15})
plt.ylabel('count', fontdict = {'fontsize':15})
plt.title('countplot for gender', fontdict = {'fontsize': 23})

for p in ax.patches:
        ax.annotate('{:.1f}%'.format( 100*p.get_height()/df.shape[0]), (p.
 ↪get_x()+0.35, p.get_height()+0.5), fontsize = 10)
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
plt.show()
```



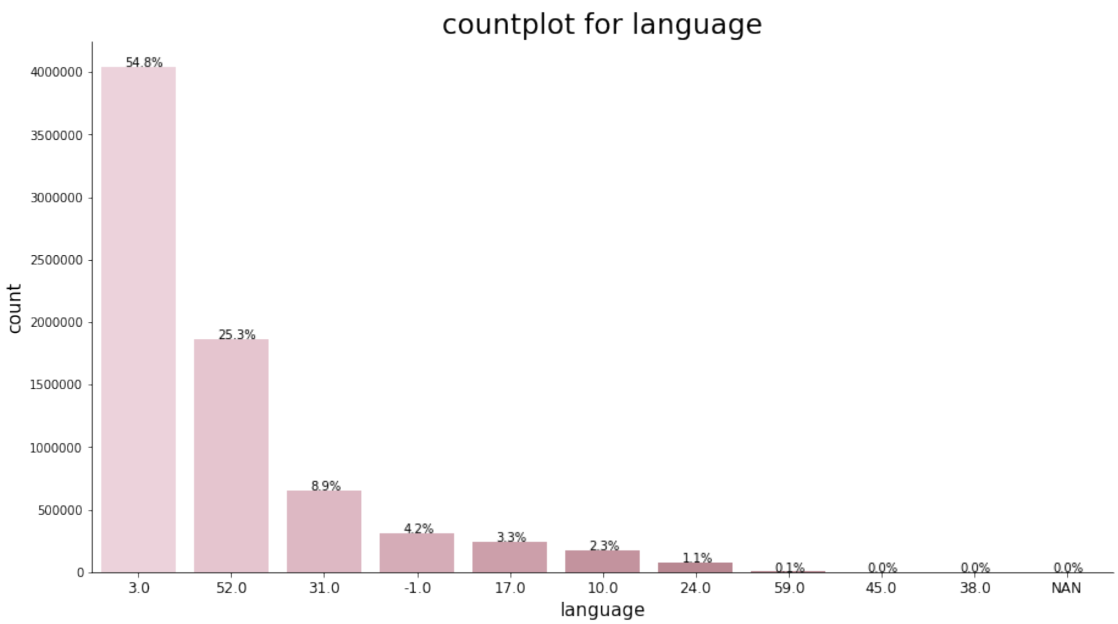
Aim for gender : Plot for determining the distribution of genders of users.

Conclusion for gender : 1.There are majority of missing values. 2. Nothing can be judged due to high missing value rate.

[292]:

```
a = df['language']
a.replace(np.nan, 'NAN',inplace = True)
a = pd.DataFrame(df.groupby('language').size().reset_index())
```

```
a = a.sort_values(by = 0, ascending = False)

plt.figure(figsize = (15,8))
ax = sns.barplot(x= 'language',y = 0, data = a, palette='ch:1.0,-.1,dark=.4')
plt.xticks( fontsize = 12)
plt.xlabel('language', fontdict = {'fontsize':15})
plt.ylabel('count', fontdict = {'fontsize':15})
plt.title('countplot for language', fontdict = {'fontsize': 23})

for p in ax.patches:
        ax.annotate('{:.1f}%'.format( 100*p.get_height()/df.shape[0]), (p.
 ↪get_x()+0.25, p.get_height()+0.5), fontsize = 10)
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
plt.show()
```

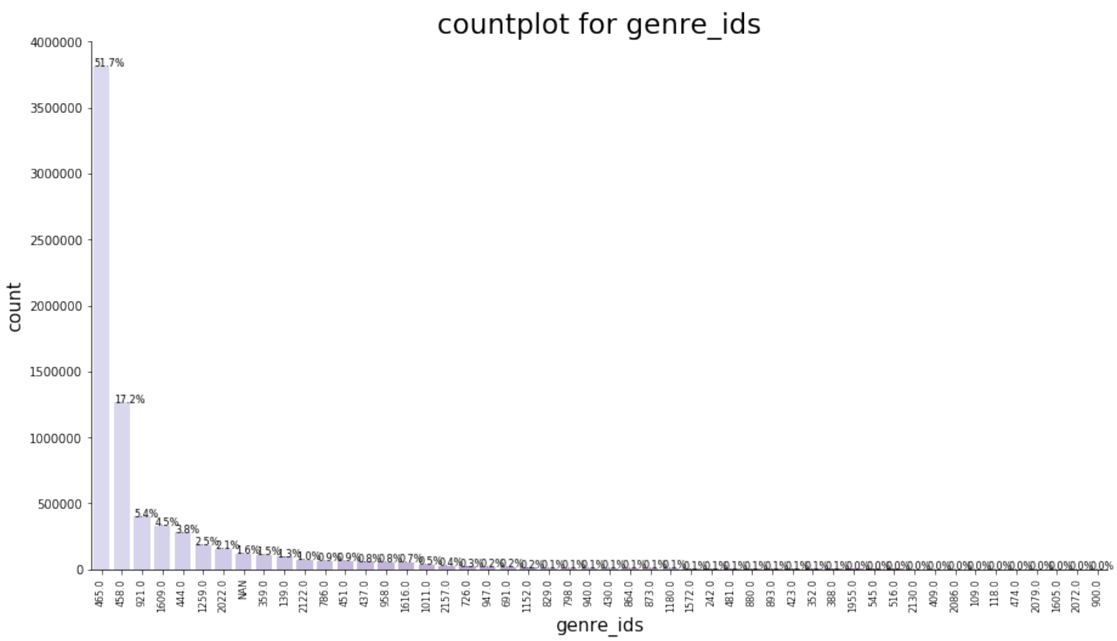


Aim for Language : Plot for determining the distribution of Language of songs most listened by users.

Conclusion for Language : 1. 54.8% of users have song language 3.0 2. It can be concluded that language 3.0 is local language of that region/country where the app is used.

[293]:
```
a = df['genre_ids']
a.replace(np.nan, 'NAN',inplace = True)
a = pd.DataFrame(df.groupby('genre_ids').size().reset_index())
a = a.sort_values(by = 0, ascending = False)
```

```
a = a[:50]

plt.figure(figsize = (15,8))
ax = sns.barplot(x= 'genre_ids',y = 0, data = a, palette='ch:1.0,-.40,dark=.4')
plt.xticks(rotation = 90,  fontsize = 8)
plt.xlabel('genre_ids', fontdict = {'fontsize':15})
plt.ylabel('count', fontdict = {'fontsize':15})
plt.title('countplot for genre_ids', fontdict = {'fontsize': 23})

for p in ax.patches:
        ax.annotate('{:.1f}%'.format( 100*p.get_height()/df.shape[0]), (p.
 ↪get_x()+0.01, p.get_height()+0.1), fontsize = 8)
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
plt.show()
```



Aim for genre ids : Plot for determining the most liked genre id by users.

Conclusion for genre ids : 1. 51.7% of users listen songs with genre id 465.

[228]:
```
a = df['registered_via']
a.replace(np.nan, 'NAN',inplace = True)
a = pd.DataFrame(df.groupby('registered_via').size().reset_index())
a = a.sort_values(by = 0, ascending = False)

plt.figure(figsize = (15,8))
```

```
ax = sns.barplot(x= 'registered_via',y = 0, data = a, palette='ch:1.0,-.
↪40,dark=.4')
plt.xticks( fontsize = 12)
plt.xlabel('registered via', fontdict = {'fontsize':15})
plt.ylabel('count', fontdict = {'fontsize':15})
plt.title('countplot for registered via', fontdict = {'fontsize': 23})

for p in ax.patches:
        ax.annotate('{:.1f}%'.format( 100*p.get_height()/df.shape[0]), (p.
↪get_x()+0.35, p.get_height()+0.5), fontsize = 10)
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
plt.show()
```

Aim for registered via : Plot for determining the distribution of mode used by users for registering in KKBox app.

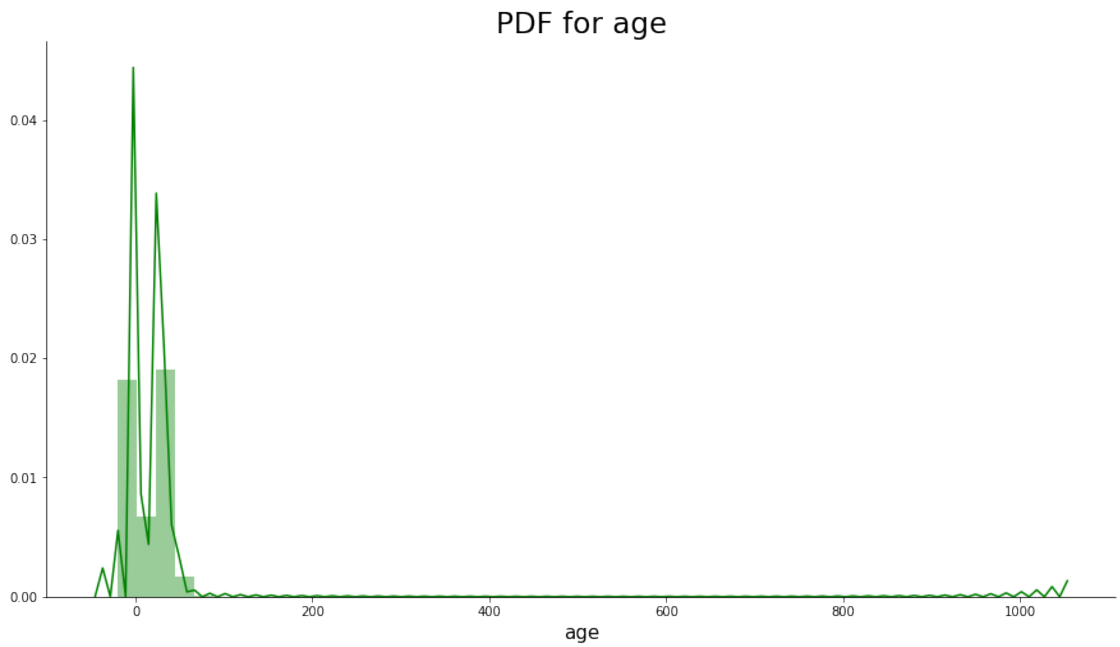
Conclusion for registered via : 1. 72.8% of users have registered via 7 and 9 registeration process.

1.2 EDA Of Continuous Variables

[297]:
```
print(df['bd'].describe())
plt.figure(figsize = (15,8))
ax = sns.distplot(a = df['bd'], color='green')
plt.xlabel('age', fontdict = {'fontsize':15})
plt.title('PDF for age', fontdict = {'fontsize': 23})
ax.spines['right'].set_visible(False)
```

```
ax.spines['top'].set_visible(False)
```

```
count    7.377418e+06
mean     1.753927e+01
std      2.155447e+01
min     -4.300000e+01
25%      0.000000e+00
50%      2.100000e+01
75%      2.900000e+01
max      1.051000e+03
Name: bd, dtype: float64
```

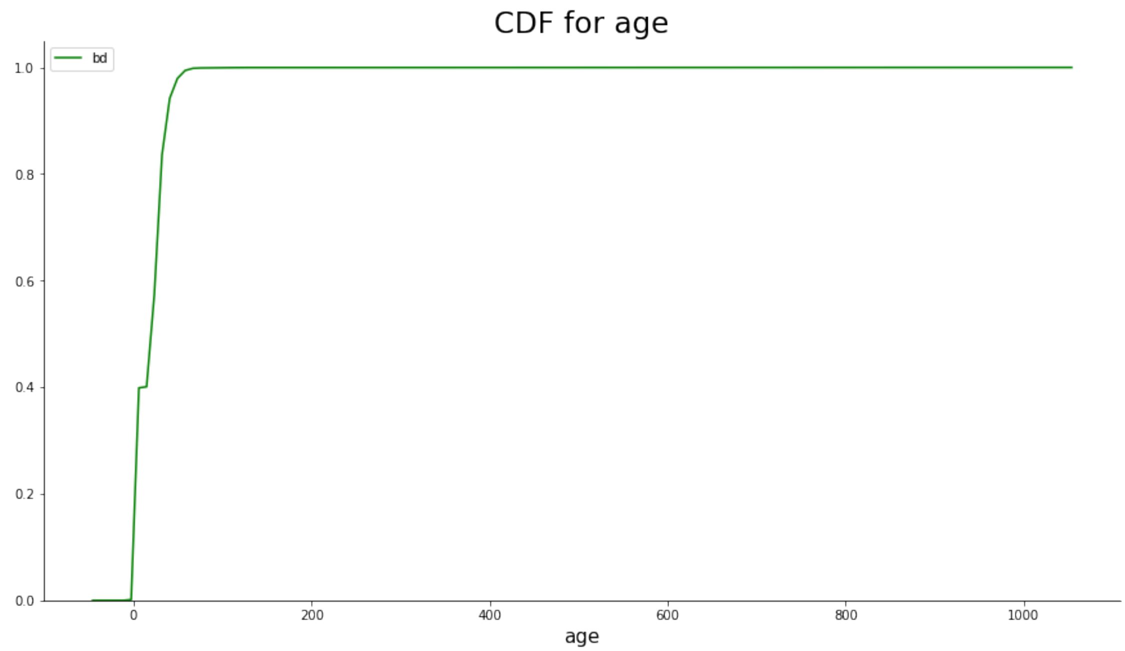


[299]:

```
print(df['bd'].describe())
plt.figure(figsize = (15,8))
ax = sns.kdeplot(data = df['bd'], color='green', cumulative = True)
plt.xlabel('age', fontdict = {'fontsize':15})
plt.title('CDF for age', fontdict = {'fontsize': 23})
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
```

```
count    7.377418e+06
mean     1.753927e+01
std      2.155447e+01
min     -4.300000e+01
25%      0.000000e+00
50%      2.100000e+01
75%      2.900000e+01
```

```
max        1.051000e+03
Name: bd, dtype: float64
```



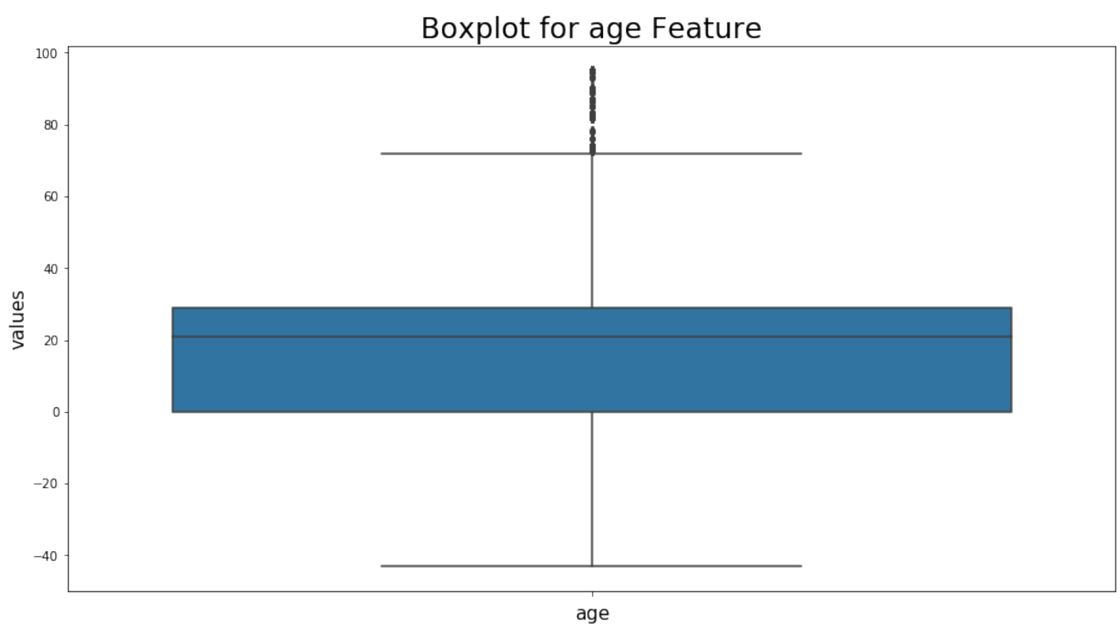
[81]:
```python
for i in range(0, 101, 10):
    print(str(i)+'th Percentile of age is '+str(np.nanpercentile(df['bd'], i)))
for i in range(90, 101, 1):
    print(str(i)+'th Percentile of age is '+str(np.nanpercentile(df['bd'], i)))
for i in [99.0, 99.1, 99.2, 99.3, 99.4, 99.5, 99.6, 99.7, 99.8, 99.9, 100]:
    print(str(i)+'th Percentile of age is '+str(np.nanpercentile(df['bd'], i)))
```

```
0th Percentile of age is -43.0
10th Percentile of age is 0.0
20th Percentile of age is 0.0
30th Percentile of age is 0.0
40th Percentile of age is 15.0
50th Percentile of age is 21.0
60th Percentile of age is 24.0
70th Percentile of age is 27.0
80th Percentile of age is 30.0
90th Percentile of age is 36.0
100th Percentile of age is 1051.0
90th Percentile of age is 36.0
91th Percentile of age is 37.0
92th Percentile of age is 38.0
93th Percentile of age is 39.0
94th Percentile of age is 40.0
95th Percentile of age is 42.0
```

```
96th Percentile of age is 44.0
97th Percentile of age is 46.0
98th Percentile of age is 50.0
99th Percentile of age is 54.0
100th Percentile of age is 1051.0
99.0th Percentile of age is 54.0
99.1th Percentile of age is 55.0
99.2th Percentile of age is 55.0
99.3th Percentile of age is 56.0
99.4th Percentile of age is 58.0
99.5th Percentile of age is 59.0
99.6th Percentile of age is 60.0
99.7th Percentile of age is 63.0
99.8th Percentile of age is 66.0
99.9th Percentile of age is 82.0
100th Percentile of age is 1051.0
```

[87]:

```
plt.figure(figsize = (15, 8))
a = df['bd'] >= 0
b = df['bd'] <= 100
a = df[a.values.tolist() and b.values.tolist()]
sns.boxplot(y = a['bd'])
plt.title('Boxplot for age Feature', fontdict = {'fontsize': 23})
plt.xlabel('age', fontdict = {'fontsize':15})
plt.ylabel('values', fontdict = {'fontsize':15})
plt.show()
```

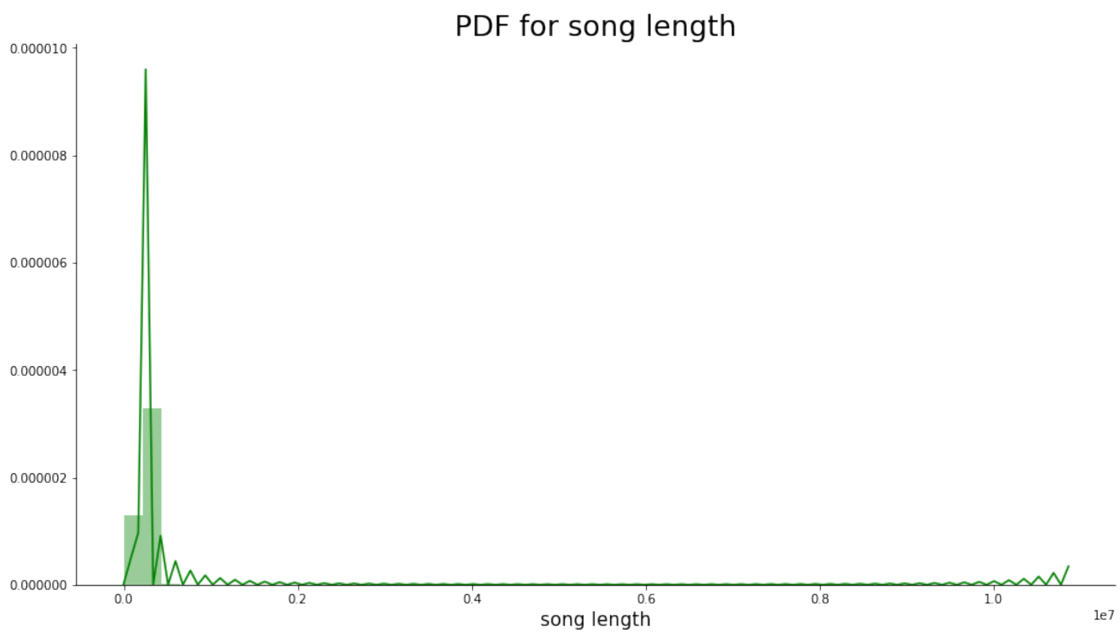


Aim for Age : Plot for determing the distribution of age feature.

Conclusion for Age : 1. Feature has outliers. 2. Majority of values are lying between 0 to 100.

[72]:
```
print(df['song_length'].describe())
plt.figure(figsize = (15,8))
ax = sns.distplot(a = df['song_length'], color='green')
plt.xlabel('song length', fontdict = {'fontsize':15})
plt.title('PDF for song length', fontdict = {'fontsize': 23})
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
```

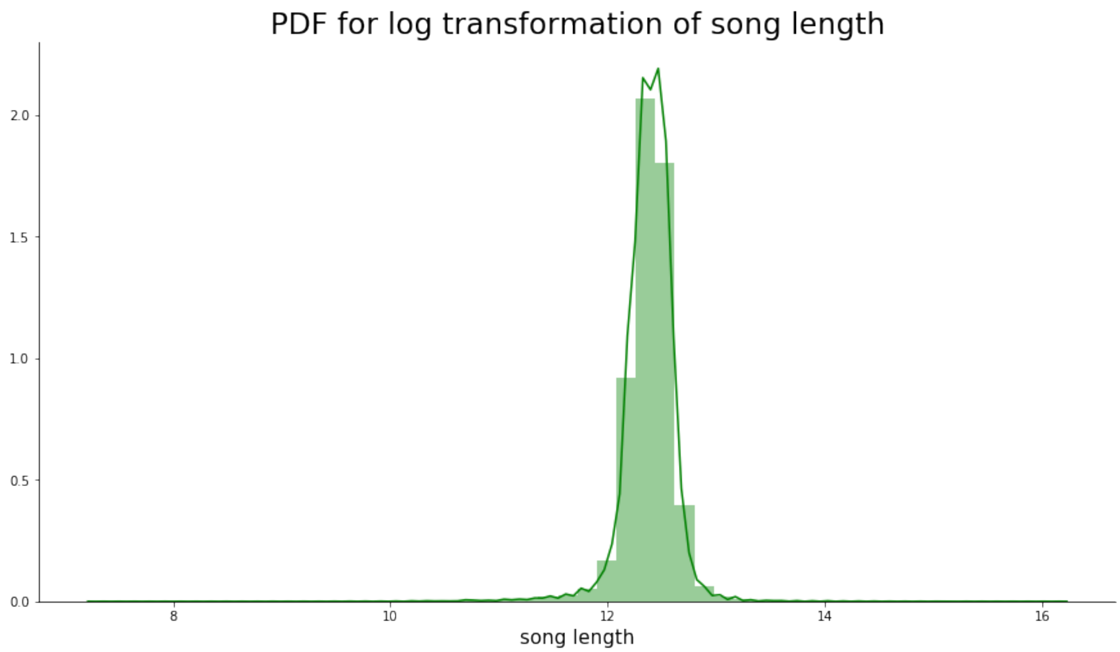
```
count    7.377304e+06
mean     2.451210e+05
std      6.734471e+04
min      1.393000e+03
25%      2.147260e+05
50%      2.418120e+05
75%      2.721600e+05
max      1.085171e+07
Name: song_length, dtype: float64
```



[67]:
```
a = np.log(df['song_length'])
print(a.describe())
plt.figure(figsize = (15,8))
ax = sns.distplot(a = np.log(df['song_length']), color='green')
plt.xlabel('song length', fontdict = {'fontsize':15})
```

```
plt.title('PDF for log transformation of song length', fontdict = {'fontsize':␣
↪23})
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
```

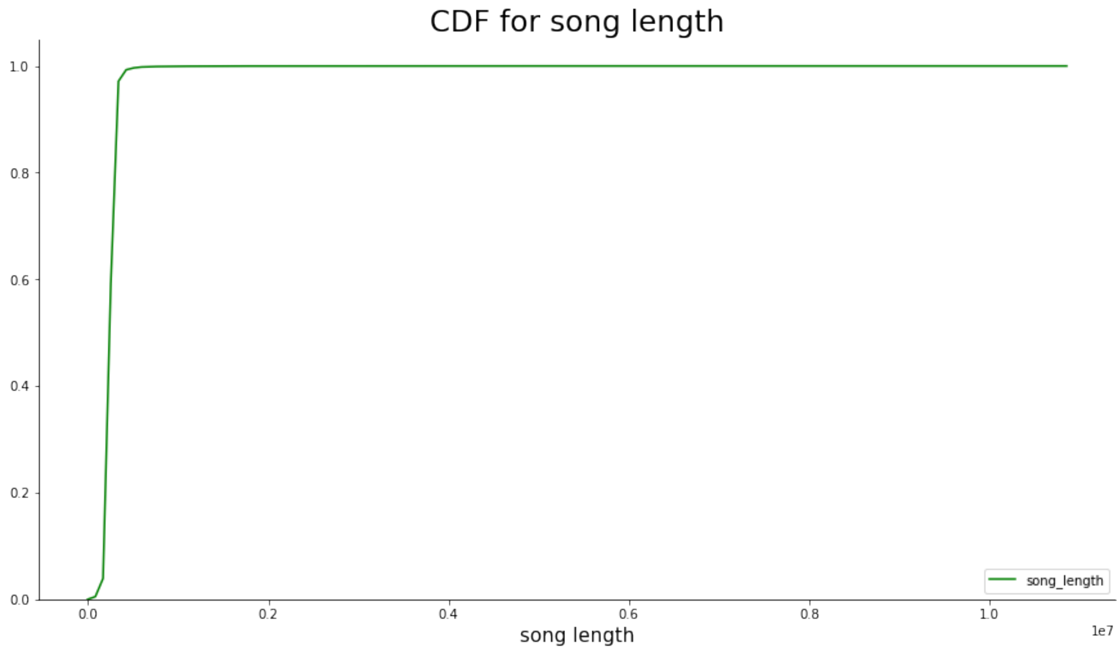
```
count    7.377304e+06
mean     1.238304e+01
std      2.356609e-01
min      7.239215e+00
25%      1.227712e+01
50%      1.239592e+01
75%      1.251415e+01
max      1.619983e+01
Name: song_length, dtype: float64
```



[18]:
```
print(df['song_length'].describe())
plt.figure(figsize = (15,8))
ax = sns.kdeplot(data = df['song_length'], color='green', cumulative = True)
plt.xlabel('song length', fontdict = {'fontsize':15})
plt.title('CDF for song length', fontdict = {'fontsize': 23})
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
```

```
count    7.377304e+06
mean     2.451210e+05
std      6.734471e+04
min      1.393000e+03
```

```
25%      2.147260e+05
50%      2.418120e+05
75%      2.721600e+05
max      1.085171e+07
Name: song_length, dtype: float64
```

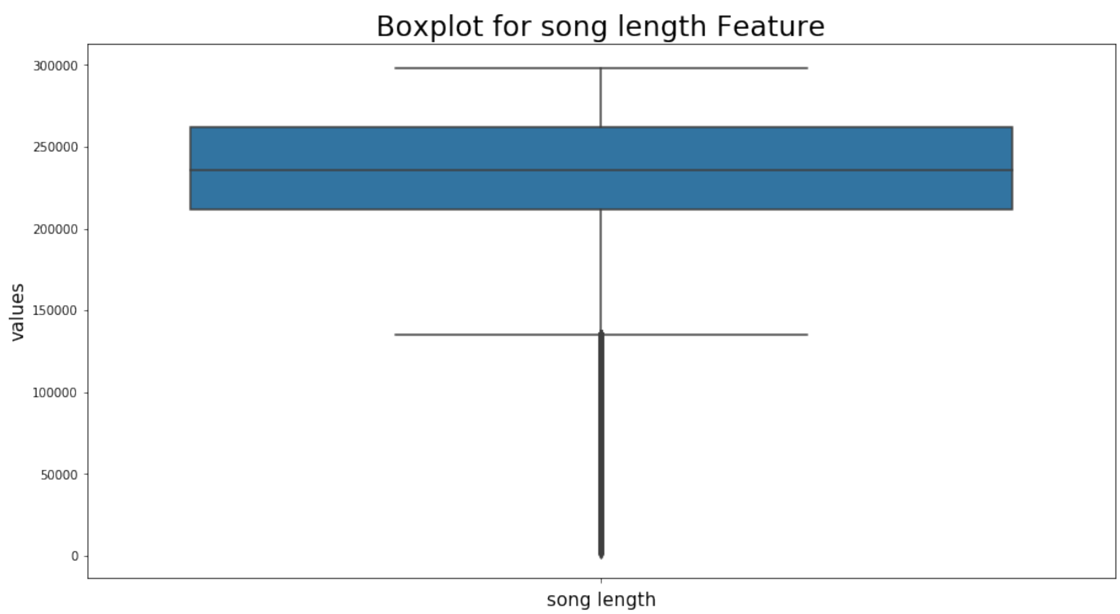


[25]:
```
for i in range(0, 101, 10):
    print(str(i)+'th Percentile of song length is '+str(np.
 ↪nanpercentile(df['song_length'], i)))
for i in range(90, 101, 1):
    print(str(i)+'th Percentile of song length is '+str(np.
 ↪nanpercentile(df['song_length'], i)))
```

```
0th Percentile of song length is 1393.0
10th Percentile of song length is 191518.0
20th Percentile of song length is 208236.0
30th Percentile of song length is 220160.0
40th Percentile of song length is 231132.0
50th Percentile of song length is 241812.0
60th Percentile of song length is 253469.0
70th Percentile of song length is 265508.0
80th Percentile of song length is 279928.0
90th Percentile of song length is 298260.0
100th Percentile of song length is 10851706.0
90th Percentile of song length is 298260.0
91th Percentile of song length is 300826.0
92th Percentile of song length is 304227.0
```

```
93th Percentile of song length is 307983.0
94th Percentile of song length is 311902.0
95th Percentile of song length is 319190.0
96th Percentile of song length is 325465.0
97th Percentile of song length is 334471.0
98th Percentile of song length is 352653.0
99th Percentile of song length is 395947.0
100th Percentile of song length is 10851706.0
```

[78]:
```python
plt.figure(figsize = (15, 8))
a = df['song_length'] >= 191518
b = df['song_length'] <= 298260
a = df[a.values.tolist() and b.values.tolist()]
sns.boxplot(y = a['song_length'])
plt.title('Boxplot for song length Feature', fontdict = {'fontsize': 23})
plt.xlabel('song length', fontdict = {'fontsize':15})
plt.ylabel('values', fontdict = {'fontsize':15})
plt.show()
```

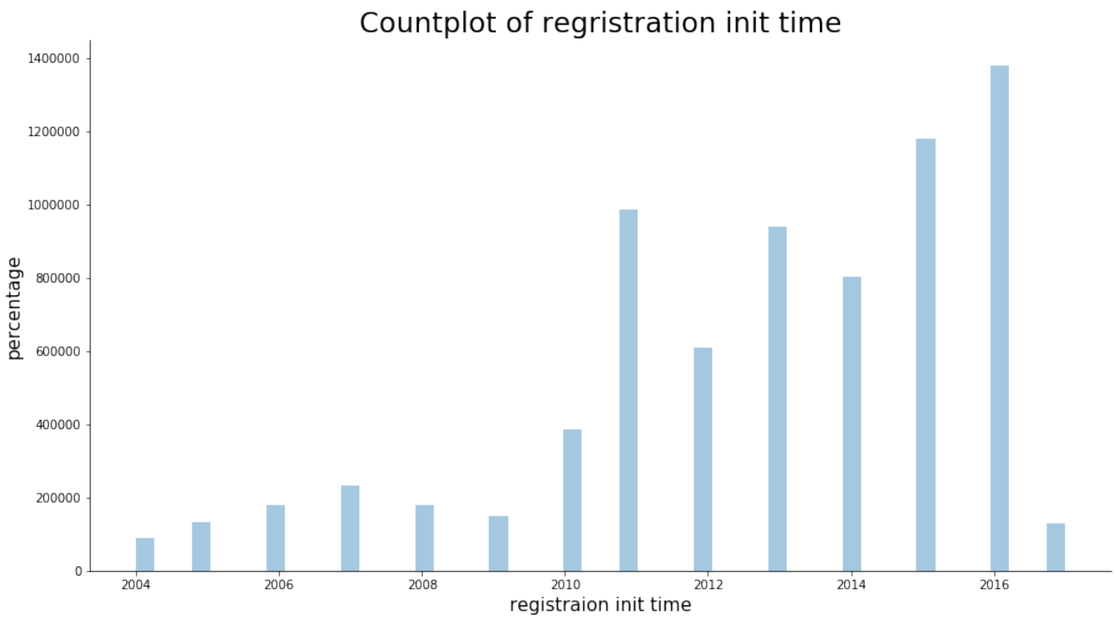


Aim for Song Length : Plots for determing the distribution of lengths of songs.

Conclusion for Song Length : 1. Feature has outliers. 2. song length are lying between 191518ms to 395947ms. 3. Applying log transformation on song length makes sense in avoiding outliers.

[118]:
```python
plt.figure(figsize = (15, 8))
ax = sns.distplot(df['registration_init_time'].astype(str).apply(lambda x:␣
↪int(x[:4])), kde = False)
```

```
plt.title('Countplot of regristration init time', fontdict = {'fontsize': 23})
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
plt.xlabel('registraion init time', fontdict = {'fontsize':15})
plt.ylabel('percentage', fontdict = {'fontsize':15})
plt.show()
```



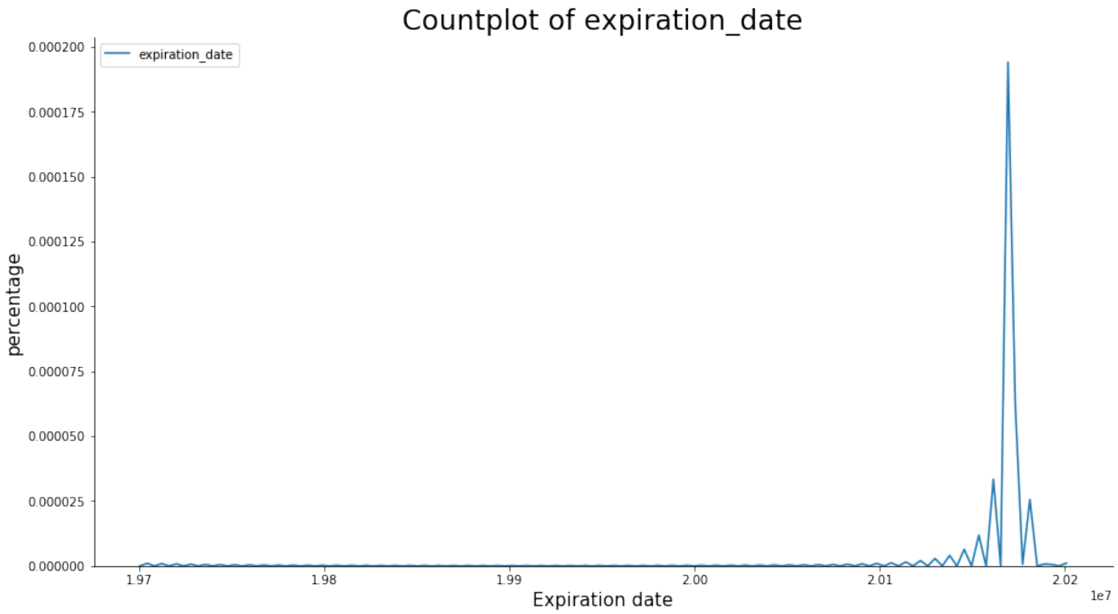
Aim for registration init time : Plot for determing the distribution of when the users registers on KKBox app.

Conclusion for registration init time : 1. Registration of users increases as time passes.

[112]:
```
print(df['expiration_date'].describe())
plt.figure(figsize = (15, 8))
ax = sns.kdeplot(df['expiration_date'])
plt.title('Countplot of expiration_date', fontdict = {'fontsize': 23})
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
plt.xlabel('Expiration date', fontdict = {'fontsize':15})
plt.ylabel('percentage', fontdict = {'fontsize':15})
plt.show()
```

```
count    7.377418e+06
mean     2.017157e+07
std      3.869831e+03
min      1.970010e+07
25%      2.017091e+07
50%      2.017093e+07
```

```
75%      2.017101e+07
max      2.020102e+07
Name: expiration_date, dtype: float64
```



Aim for expiration date : Plot for determining the distribution of expiration date of plans of users.

Conclusion for expiration date : 1. There are outliers in the features. 2. Most of plans of users expiration are in the year of 2017.

2 EDA on Feature Engineered Features

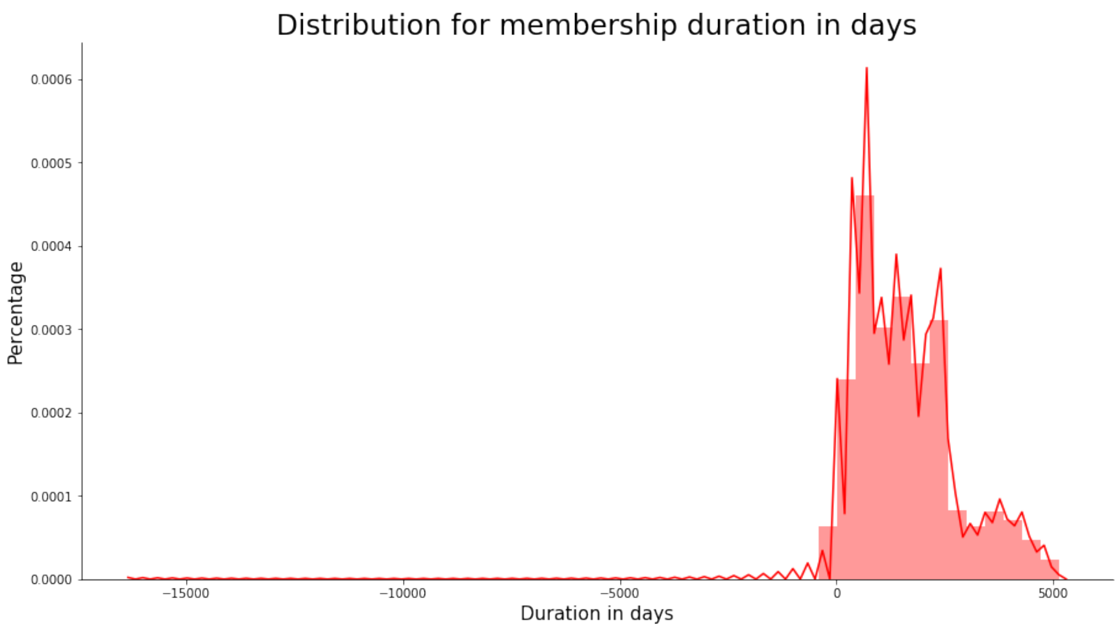
1. Duration of subscription

[20]:
```
a = pd.to_datetime(df['expiration_date'], format = '%Y%m%d') - pd.
 ↪to_datetime(df['registration_init_time'], format = '%Y%m%d')
a =list(a)
a = [i.days for i in a]
df['membership_duration'] = a
```

[23]:
```
print(df['membership_duration'].describe())
plt.figure(figsize = (15,8))
ax = sns.distplot(a = df['membership_duration'], color='red')
plt.xlabel('Duration in days', fontdict = {'fontsize':15})
plt.ylabel('Percentage', fontdict = {'fontsize': 15})
plt.title('Distribution for membership duration in days', fontdict =␣
 ↪{'fontsize': 23})
```

```
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
```

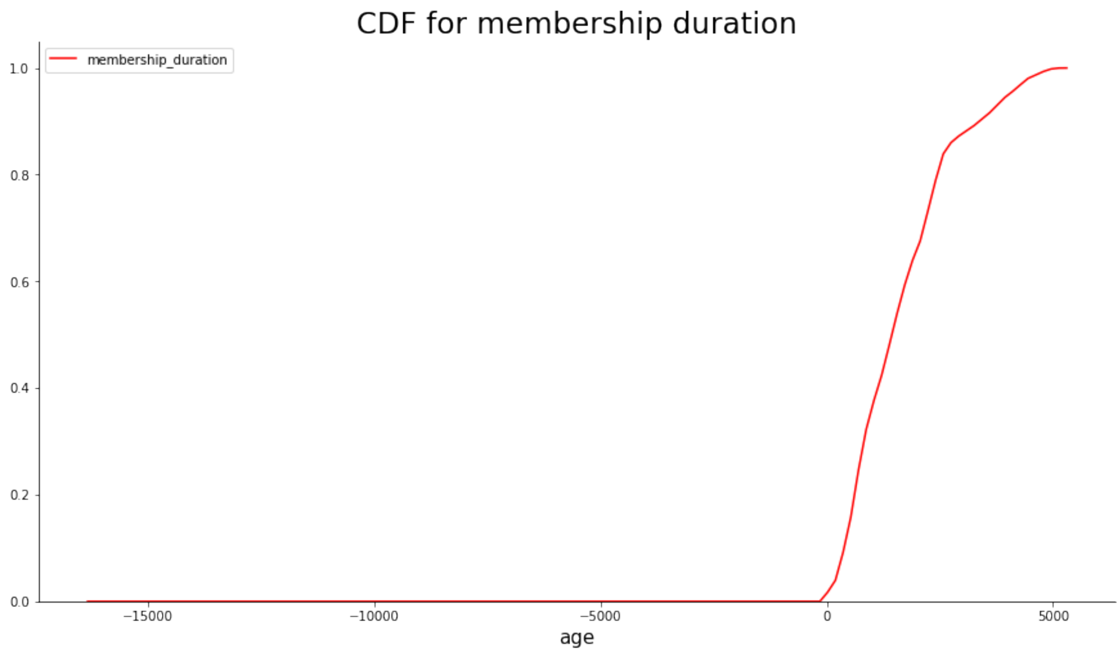
```
count    7.377418e+06
mean     1.627961e+03
std      1.128673e+03
min     -1.619100e+04
25%      7.010000e+02
50%      1.433000e+03
75%      2.286000e+03
max      5.149000e+03
Name: membership_duration, dtype: float64
```



[204]:
```
print(df['membership_duration'].describe())
plt.figure(figsize = (15,8))
ax = sns.kdeplot(data = df['membership_duration'], color='red', cumulative =␣
 ↪True)
plt.xlabel('age', fontdict = {'fontsize':15})
plt.title('CDF for membership duration', fontdict = {'fontsize': 23})
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
```

```
count    7.377418e+06
mean     1.627961e+03
std      1.128673e+03
min     -1.619100e+04
25%      7.010000e+02
```

```
50%      1.433000e+03
75%      2.286000e+03
max      5.149000e+03
Name: membership_duration, dtype: float64
```



Aim for Duration in days : Plot for determining the distribution of duration in days of subscription.

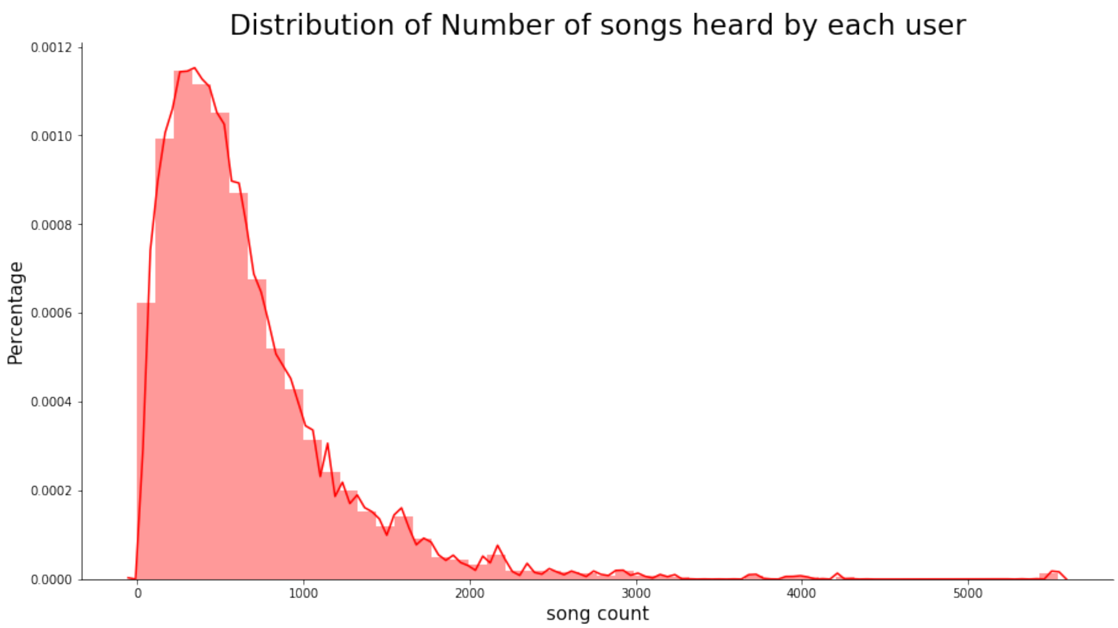
Conclusion for Duration in days : 1. There are outliers in the features. 2. Range of membership duration varies from 0 to 5000.

[190]:
```
a = pd.DataFrame(df['msno'].value_counts().reset_index())
a.rename(columns = {'index':'msno', 'msno':'song_count'}, inplace=True)
df = df.merge(a, how = 'left',on = 'msno')
```

[201]:
```
print(df['song_count'].describe())
plt.figure(figsize = (15,8))
ax = sns.distplot(a = df['song_count'], color='red')
plt.xlabel('song count', fontdict = {'fontsize':15})
plt.ylabel('Percentage', fontdict = {'fontsize': 15})
plt.title('Distribution of Number of songs heard by each user', fontdict =␣
 ↪{'fontsize': 23})
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
```

```
count    7.371599e+06
mean     6.471541e+02
std      5.573472e+02
```

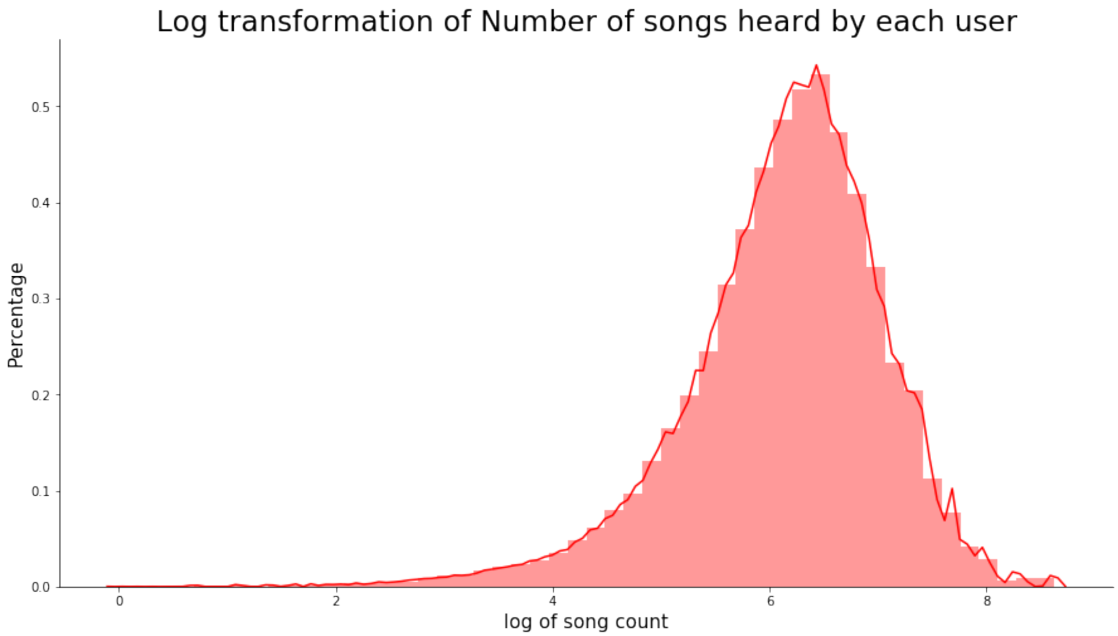
```
min      1.000000e+00
25%      2.860000e+02
50%      5.090000e+02
75%      8.360000e+02
max      5.537000e+03
Name: song_count, dtype: float64
```



[202]:

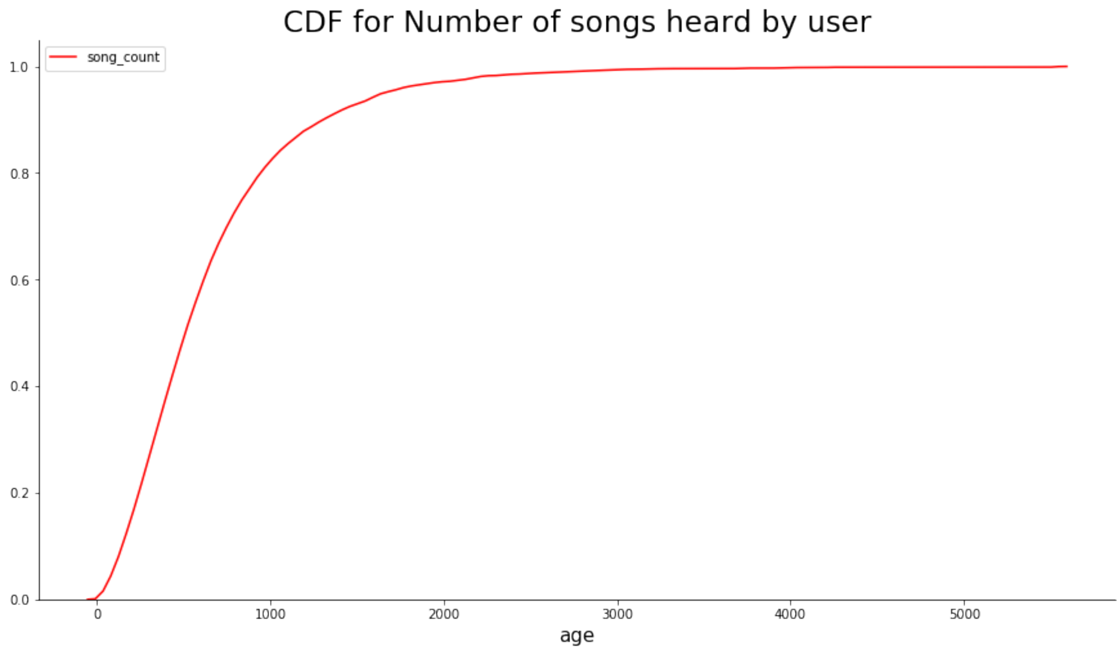
```
print(np.log(df['song_count']).describe())
plt.figure(figsize = (15,8))
ax = sns.distplot(a = np.log(df['song_count']), color='red')
plt.xlabel('log of song count', fontdict = {'fontsize':15})
plt.ylabel('Percentage', fontdict = {'fontsize': 15})
plt.title('Log transformation of Number of songs heard by each user', fontdict␣
 ↪= {'fontsize': 23})
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
```

```
count    7.371599e+06
mean     6.128115e+00
std      9.168676e-01
min      0.000000e+00
25%      5.655992e+00
50%      6.232448e+00
75%      6.728629e+00
max      8.619208e+00
Name: song_count, dtype: float64
```



[203]:
```
print(df['song_count'].describe())
plt.figure(figsize = (15,8))
ax = sns.kdeplot(data = df['song_count'], color='red', cumulative = True)
plt.xlabel('age', fontdict = {'fontsize':15})
plt.title('CDF for Number of songs heard by user', fontdict = {'fontsize': 23})
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
```

```
count    7.371599e+06
mean     6.471541e+02
std      5.573472e+02
min      1.000000e+00
25%      2.860000e+02
50%      5.090000e+02
75%      8.360000e+02
max      5.537000e+03
Name: song_count, dtype: float64
```



Aim for song count : Plot for determining the distribution of Number of songs heard by a particular user.

Conclusion for song count : 1. There are some outliers. 2. Log transformation of song count makes sense in avoiding outliers. 3. Range of song count varies from 0 to 2500.

1. Create Embedding for Users and Songs

[7]:
```
map_song_id = {key: id for id, key in enumerate(df.song_id.unique())}
map_msno = {key: id for id, key in enumerate(df.msno.unique())}
```

[8]:
```
row = []
cols = []
ids = df['song_id'].values
ms = df['msno'].values
for i in range(df.shape[0]):
    song_id = ids[i]
    msno = ms[i]
    row.append(map_song_id[song_id])
    cols.append(map_msno[msno])
row = np.array(row)
cols = np.array(cols)
```

[9]:
```
from scipy.sparse import csc_matrix
Y = csc_matrix((df.target.values, (row, cols)), dtype = np.int8)
```

```
[10]: def predict(df, emb_user, emb_song):
          df['prediction'] = np.sum(np.multiply(emb_song[df['song_id'].apply(lambda x:
       ↪ map_song_id[x])],emb_user[df['msno'].apply(lambda x: map_msno[x])]), axis=1)
          return df
```

```
[11]: lmbda = 0.0002
      def cost(df, Y,emb_user, emb_anime):
          predicted = predict(df, emb_user, emb_song)
          predicted = csc_matrix((df.prediction.values, (row, cols)), dtype = np.int8)
          return np.sum((Y-predicted).power(2))/df.shape[0]
```

```
[12]: def create_embeddings(n, K):
          return np.random.random((n, K))
```

```
[14]: def gradient(df, Y, emb_user, emb_song):
          predicted = predict(df, emb_user, emb_song)
          predicted = csc_matrix((df.prediction.values, (row, cols)), dtype = np.int8)
          delta = (Y-predicted)
          grad_user = (-2/df.shape[0])*(delta.T*emb_song) + 2*lmbda*emb_user
          grad_song = (-2/df.shape[0])*(delta*emb_user) + 2*lmbda*emb_song
          return grad_user, grad_song
```

```
[15]: emb_user = create_embeddings(30755, 30)
      emb_song = create_embeddings(359966, 30)
```

```
[16]: beta = 0.9
      grad_user, grad_song = gradient(df, Y,  emb_user, emb_song)
      v_user = grad_user
      v_song = grad_song
      for i in range(500):
          grad_user, grad_song = gradient(df, Y, emb_user, emb_song)
          v_user = beta*v_user + (1-beta)*grad_user
          v_song = beta*v_song + (1-beta)*grad_song
          emb_user = emb_user - 1*v_user
          emb_song = emb_song - 1*v_song
          print("\niteration", i+1, ":")
          print("train mse:",  cost(df, Y, emb_user, emb_song))
```

```
iteration 1 :
train mse: 43.37502700809416

iteration 2 :
train mse: 42.910836013358605

iteration 3 :
train mse: 42.45490766552742
```

```
iteration 4 :
train mse: 42.004536953172504

iteration 5 :
train mse: 41.56056075445366

iteration 6 :
train mse: 41.123853087896066

iteration 7 :
train mse: 40.69462351191162

iteration 8 :
train mse: 40.27064496006597

iteration 9 :
train mse: 39.8566118932125

iteration 10 :
train mse: 39.4504421465613

iteration 11 :
train mse: 39.05086942342158

iteration 12 :
train mse: 38.65748897514008

iteration 13 :
train mse: 38.274843990133135

iteration 14 :
train mse: 37.8969231511621

iteration 15 :
train mse: 37.52631083124204

iteration 16 :
train mse: 37.16426329645412

iteration 17 :
train mse: 36.80839136402465

iteration 18 :
train mse: 36.458558807431004

iteration 19 :
train mse: 36.11678896329312
```

```
iteration 20 :
train mse: 35.782108726928584

iteration 21 :
train mse: 35.454441242179854

iteration 22 :
train mse: 35.13069504805069

iteration 23 :
train mse: 34.814451207726066

iteration 24 :
train mse: 34.50443054738121

iteration 25 :
train mse: 34.20083801676955

iteration 26 :
train mse: 33.90267516358704

iteration 27 :
train mse: 33.608184055722475

iteration 28 :
train mse: 33.32009491667681

iteration 29 :
train mse: 33.03800082901633

iteration 30 :
train mse: 32.762038425909985

iteration 31 :
train mse: 32.48803388936346

iteration 32 :
train mse: 32.22053393206132

iteration 33 :
train mse: 31.958337591824133

iteration 34 :
train mse: 31.699166429230388

iteration 35 :
train mse: 31.444748826757547
```

```
iteration 36 :
train mse: 31.194280302403904

iteration 37 :
train mse: 30.948994485604583

iteration 38 :
train mse: 30.707528026743233

iteration 39 :
train mse: 30.469849885149518

iteration 40 :
train mse: 30.235357275404485

iteration 41 :
train mse: 30.00478622737657

iteration 42 :
train mse: 29.777944126251217

iteration 43 :
train mse: 29.55461897373851

iteration 44 :
train mse: 29.336374596098526

iteration 45 :
train mse: 29.12015301288337

iteration 46 :
train mse: 28.9073487499285

iteration 47 :
train mse: 28.698008300465013

iteration 48 :
train mse: 28.491077772738375

iteration 49 :
train mse: 28.287603738869073

iteration 50 :
train mse: 28.087751703915924

iteration 51 :
train mse: 27.891058226604486
```

```
iteration 52 :
train mse: 27.695981575125607

iteration 53 :
train mse: 27.504513096587452

iteration 54 :
train mse: 27.31509167570551

iteration 55 :
train mse: 27.12785367455118

iteration 56 :
train mse: 26.94378236396528

iteration 57 :
train mse: 26.761380201040527

iteration 58 :
train mse: 26.58213727350138

iteration 59 :
train mse: 26.405799156290183

iteration 60 :
train mse: 26.231187930519866

iteration 61 :
train mse: 26.05887913630487

iteration 62 :
train mse: 25.889261934188898

iteration 63 :
train mse: 25.72147518278075

iteration 64 :
train mse: 25.555245751291306

iteration 65 :
train mse: 25.391288117333193

iteration 66 :
train mse: 25.22984152450085

iteration 67 :
train mse: 25.07019664603524
```

```
iteration 68 :
train mse: 24.912327592119627

iteration 69 :
train mse: 24.755823107759383

iteration 70 :
train mse: 24.60112006666831

iteration 71 :
train mse: 24.44877611652207

iteration 72 :
train mse: 24.298524226226576

iteration 73 :
train mse: 24.149396441952998

iteration 74 :
train mse: 24.00169612186811

iteration 75 :
train mse: 23.856044621573563

iteration 76 :
train mse: 23.712268709730154

iteration 77 :
train mse: 23.569920668721767

iteration 78 :
train mse: 23.429129812083307

iteration 79 :
train mse: 23.290184723164664

iteration 80 :
train mse: 23.151678405642734

iteration 81 :
train mse: 23.015464217968944

iteration 82 :
train mse: 22.88034228235407

iteration 83 :
train mse: 22.747103525921943
```

```
iteration 84 :
train mse: 22.615054616669408

iteration 85 :
train mse: 22.483707307895525

iteration 86 :
train mse: 22.35514417103653

iteration 87 :
train mse: 22.22705721703718

iteration 88 :
train mse: 22.10104375270589

iteration 89 :
train mse: 21.97611115433611

iteration 90 :
train mse: 21.85168659278897

iteration 91 :
train mse: 21.729251345118307

iteration 92 :
train mse: 21.606861506288514

iteration 93 :
train mse: 21.486379652067974

iteration 94 :
train mse: 21.367260469719895

iteration 95 :
train mse: 21.249171864736415

iteration 96 :
train mse: 21.131456832187087

iteration 97 :
train mse: 21.01558621186979

iteration 98 :
train mse: 20.90053986367588

iteration 99 :
train mse: 20.786536021139103
```

```
iteration 100 :
train mse: 20.673873840414085

iteration 101 :
train mse: 20.56223559516351

iteration 102 :
train mse: 20.45135073002506

iteration 103 :
train mse: 20.342324103094064

iteration 104 :
train mse: 20.233774065669046

iteration 105 :
train mse: 20.126185340182705

iteration 106 :
train mse: 20.019500182855303

iteration 107 :
train mse: 19.913393276617917

iteration 108 :
train mse: 19.80828766920893

iteration 109 :
train mse: 19.704321349285074

iteration 110 :
train mse: 19.601417731786377

iteration 111 :
train mse: 19.49957153031047

iteration 112 :
train mse: 19.398597585225616

iteration 113 :
train mse: 19.29833201805835

iteration 114 :
train mse: 19.19898411612301

iteration 115 :
train mse: 19.100426328018827
```

```
iteration 116 :
train mse: 19.002385658505457

iteration 117 :
train mse: 18.905528194281523

iteration 118 :
train mse: 18.809259553952344

iteration 119 :
train mse: 18.71417249232726

iteration 120 :
train mse: 18.619941827886127

iteration 121 :
train mse: 18.52598090551464

iteration 122 :
train mse: 18.433082956665867

iteration 123 :
train mse: 18.340853127747405

iteration 124 :
train mse: 18.249884309117363

iteration 125 :
train mse: 18.159066898473153

iteration 126 :
train mse: 18.069942763172698

iteration 127 :
train mse: 17.981068173173867

iteration 128 :
train mse: 17.892292669332278

iteration 129 :
train mse: 17.804876855290022

iteration 130 :
train mse: 17.718017875630743

iteration 131 :
train mse: 17.631271130360243
```

```
iteration 132 :
train mse: 17.54555021824709

iteration 133 :
train mse: 17.45989965595009

iteration 134 :
train mse: 17.375635893208166

iteration 135 :
train mse: 17.291637670523752

iteration 136 :
train mse: 17.20816768143001

iteration 137 :
train mse: 17.126065786159874

iteration 138 :
train mse: 17.044529535943333

iteration 139 :
train mse: 16.964028065103538

iteration 140 :
train mse: 16.88315939262219

iteration 141 :
train mse: 16.802873173242997

iteration 142 :
train mse: 16.723626070801465

iteration 143 :
train mse: 16.644642475185762

iteration 144 :
train mse: 16.566184944380268

iteration 145 :
train mse: 16.488281943628515

iteration 146 :
train mse: 16.410986201405425

iteration 147 :
train mse: 16.334089514786882
```

```
iteration 148 :
train mse: 16.25853137235819

iteration 149 :
train mse: 16.18302704279465

iteration 150 :
train mse: 16.108843229433386

iteration 151 :
train mse: 16.034715939912854

iteration 152 :
train mse: 15.960928471180567

iteration 153 :
train mse: 15.88749790238265

iteration 154 :
train mse: 15.81450230961564

iteration 155 :
train mse: 15.742244237753642

iteration 156 :
train mse: 15.67006586857353

iteration 157 :
train mse: 15.598270695790857

iteration 158 :
train mse: 15.527106909219459

iteration 159 :
train mse: 15.456835846904703

iteration 160 :
train mse: 15.387242528483542

iteration 161 :
train mse: 15.318363145479896

iteration 162 :
train mse: 15.24912699809066

iteration 163 :
train mse: 15.18094935111444
```

```
iteration 164 :
train mse: 15.113280418704756

iteration 165 :
train mse: 15.045802203426728

iteration 166 :
train mse: 14.978647543083502

iteration 167 :
train mse: 14.912025724989421

iteration 168 :
train mse: 14.845728410671592

iteration 169 :
train mse: 14.780090270064676

iteration 170 :
train mse: 14.714384219519621

iteration 171 :
train mse: 14.649858527739651

iteration 172 :
train mse: 14.585028800049015

iteration 173 :
train mse: 14.521236562710694

iteration 174 :
train mse: 14.45786154451327

iteration 175 :
train mse: 14.394590763326681

iteration 176 :
train mse: 14.331387214334338

iteration 177 :
train mse: 14.269334881119654

iteration 178 :
train mse: 14.207687296558227

iteration 179 :
train mse: 14.145817140902142
```

```
iteration 180 :
train mse: 14.084872105660816

iteration 181 :
train mse: 14.024021276820697

iteration 182 :
train mse: 13.963623994194174

iteration 183 :
train mse: 13.903299501261824

iteration 184 :
train mse: 13.843877221000627

iteration 185 :
train mse: 13.785208049753992

iteration 186 :
train mse: 13.72650702454436

iteration 187 :
train mse: 13.66787648469966

iteration 188 :
train mse: 13.609471633571529

iteration 189 :
train mse: 13.551747101763787

iteration 190 :
train mse: 13.494353308976121

iteration 191 :
train mse: 13.437200928563353

iteration 192 :
train mse: 13.380437708694288

iteration 193 :
train mse: 13.324469889058747

iteration 194 :
train mse: 13.268438361497207

iteration 195 :
train mse: 13.213258215814802
```

```
iteration 196 :
train mse: 13.157779998367992

iteration 197 :
train mse: 13.103039979570088

iteration 198 :
train mse: 13.048169969493392

iteration 199 :
train mse: 12.993816535812394

iteration 200 :
train mse: 12.939658563470308

iteration 201 :
train mse: 12.885543966737414

iteration 202 :
train mse: 12.832055063167086

iteration 203 :
train mse: 12.779152408064718

iteration 204 :
train mse: 12.726580898628761

iteration 205 :
train mse: 12.673899594682043

iteration 206 :
train mse: 12.621520835609424

iteration 207 :
train mse: 12.569702570736808

iteration 208 :
train mse: 12.518200676713723

iteration 209 :
train mse: 12.466905765675742

iteration 210 :
train mse: 12.41644271749276

iteration 211 :
train mse: 12.365667093826051
```

```
iteration 212 :
train mse: 12.315913101304549

iteration 213 :
train mse: 12.266171714819466

iteration 214 :
train mse: 12.21666265894111

iteration 215 :
train mse: 12.167309077511943

iteration 216 :
train mse: 12.118113681507541

iteration 217 :
train mse: 12.069499654215065

iteration 218 :
train mse: 12.021185867467453

iteration 219 :
train mse: 11.972778687611303

iteration 220 :
train mse: 11.924675001470705

iteration 221 :
train mse: 11.877083283067329

iteration 222 :
train mse: 11.829432329847652

iteration 223 :
train mse: 11.78233428009637

iteration 224 :
train mse: 11.735766361618658

iteration 225 :
train mse: 11.689050423874585

iteration 226 :
train mse: 11.642608159114747

iteration 227 :
train mse: 11.59661957611728
```

```
iteration 228 :
train mse: 11.550862645982646

iteration 229 :
train mse: 11.505608872914616

iteration 230 :
train mse: 11.460374076675606

iteration 231 :
train mse: 11.414856254586631

iteration 232 :
train mse: 11.370012516574226

iteration 233 :
train mse: 11.325412766363517

iteration 234 :
train mse: 11.28102311675982

iteration 235 :
train mse: 11.237022763248605

iteration 236 :
train mse: 11.19344586412211

iteration 237 :
train mse: 11.149836026642383

iteration 238 :
train mse: 11.107010881042664

iteration 239 :
train mse: 11.064017112762215

iteration 240 :
train mse: 11.020957196677754

iteration 241 :
train mse: 10.978391220342944

iteration 242 :
train mse: 10.936060963334327

iteration 243 :
train mse: 10.894195096441601
```

```
iteration 244 :
train mse: 10.852409474425876

iteration 245 :
train mse: 10.810885054906743

iteration 246 :
train mse: 10.769746813858182

iteration 247 :
train mse: 10.728253570558154

iteration 248 :
train mse: 10.687461249992884

iteration 249 :
train mse: 10.646464250771746

iteration 250 :
train mse: 10.605932997154289

iteration 251 :
train mse: 10.565522924145005

iteration 252 :
train mse: 10.5253206474135

iteration 253 :
train mse: 10.485236975863371

iteration 254 :
train mse: 10.44547428382125

iteration 255 :
train mse: 10.406139790371103

iteration 256 :
train mse: 10.366906822956215

iteration 257 :
train mse: 10.327689308102103

iteration 258 :
train mse: 10.288507035930458

iteration 259 :
train mse: 10.249998576737823
```

```
iteration 260 :
train mse: 10.211284490047873

iteration 261 :
train mse: 10.172851531524985

iteration 262 :
train mse: 10.134652665742946

iteration 263 :
train mse: 10.096413541973629

iteration 264 :
train mse: 10.05812345186351

iteration 265 :
train mse: 10.020397109124087

iteration 266 :
train mse: 9.98325389723071

iteration 267 :
train mse: 9.946208822653128

iteration 268 :
train mse: 9.90892423338355

iteration 269 :
train mse: 9.871924161000502

iteration 270 :
train mse: 9.835156825870515

iteration 271 :
train mse: 9.79905896615862

iteration 272 :
train mse: 9.762475570721355

iteration 273 :
train mse: 9.726259919120755

iteration 274 :
train mse: 9.69029652379735

iteration 275 :
train mse: 9.65495692395361
```

```
iteration 276 :
train mse: 9.619314237040655

iteration 277 :
train mse: 9.584309578229131

iteration 278 :
train mse: 9.549055645213542

iteration 279 :
train mse: 9.514003544329466

iteration 280 :
train mse: 9.479214136978547

iteration 281 :
train mse: 9.444655569197787

iteration 282 :
train mse: 9.410413101169

iteration 283 :
train mse: 9.376180934847396

iteration 284 :
train mse: 9.3420366041344

iteration 285 :
train mse: 9.308047205675482

iteration 286 :
train mse: 9.273928493681664

iteration 287 :
train mse: 9.240246248755323

iteration 288 :
train mse: 9.206694537302889

iteration 289 :
train mse: 9.173429240419887

iteration 290 :
train mse: 9.140164892378337

iteration 291 :
train mse: 9.107121082199761
```

```
iteration 292 :
train mse: 9.074303096286533

iteration 293 :
train mse: 9.041709579150863

iteration 294 :
train mse: 9.009145069453838

iteration 295 :
train mse: 8.976675443901918

iteration 296 :
train mse: 8.944717379440883

iteration 297 :
train mse: 8.912222270718564

iteration 298 :
train mse: 8.880455736681858

iteration 299 :
train mse: 8.848640405084815

iteration 300 :
train mse: 8.817075703179622

iteration 301 :
train mse: 8.785574980297985

iteration 302 :
train mse: 8.754308079059639

iteration 303 :
train mse: 8.72284151447024

iteration 304 :
train mse: 8.69163832115789

iteration 305 :
train mse: 8.660796500889607

iteration 306 :
train mse: 8.63003451885199

iteration 307 :
train mse: 8.5993118459602
```

```
iteration 308 :
train mse: 8.568561521117552

iteration 309 :
train mse: 8.53832153742678

iteration 310 :
train mse: 8.507917268616202

iteration 311 :
train mse: 8.477933743214766

iteration 312 :
train mse: 8.44798220732511

iteration 313 :
train mse: 8.4182100024697

iteration 314 :
train mse: 8.388530241881373

iteration 315 :
train mse: 8.3589826413523

iteration 316 :
train mse: 8.329541988809636

iteration 317 :
train mse: 8.300551764858653

iteration 318 :
train mse: 8.271405795360925

iteration 319 :
train mse: 8.242412453788033

iteration 320 :
train mse: 8.213820472148928

iteration 321 :
train mse: 8.184967016915675

iteration 322 :
train mse: 8.156643557407213

iteration 323 :
train mse: 8.128362524666489
```

```
iteration 324 :
train mse: 8.100117547900906

iteration 325 :
train mse: 8.072085924913026

iteration 326 :
train mse: 8.044105674912279

iteration 327 :
train mse: 8.016336203262442

iteration 328 :
train mse: 7.988536639783729

iteration 329 :
train mse: 7.960916949534377

iteration 330 :
train mse: 7.933471439465677

iteration 331 :
train mse: 7.90592846982508

iteration 332 :
train mse: 7.878559680365136

iteration 333 :
train mse: 7.851397873890296

iteration 334 :
train mse: 7.824699237592339

iteration 335 :
train mse: 7.797735467883208

iteration 336 :
train mse: 7.770634658358792

iteration 337 :
train mse: 7.744141920655709

iteration 338 :
train mse: 7.717697980512965

iteration 339 :
train mse: 7.691296602686739
```

```
iteration 340 :
train mse: 7.66481619992252

iteration 341 :
train mse: 7.638626007093539

iteration 342 :
train mse: 7.612944257733532

iteration 343 :
train mse: 7.586965114353016

iteration 344 :
train mse: 7.561182652250421

iteration 345 :
train mse: 7.535628860937526

iteration 346 :
train mse: 7.510187032915852

iteration 347 :
train mse: 7.4847203994676725

iteration 348 :
train mse: 7.459437570163437

iteration 349 :
train mse: 7.434388426953712

iteration 350 :
train mse: 7.409261614293781

iteration 351 :
train mse: 7.384129108585144

iteration 352 :
train mse: 7.359412059883281

iteration 353 :
train mse: 7.334576677097597

iteration 354 :
train mse: 7.309768404067656

iteration 355 :
train mse: 7.2853997157271015
```

```
iteration 356 :
train mse: 7.261036584886474

iteration 357 :
train mse: 7.236721438313513

iteration 358 :
train mse: 7.212524490275595

iteration 359 :
train mse: 7.18838433717596

iteration 360 :
train mse: 7.164218565357148

iteration 361 :
train mse: 7.1402779400597876

iteration 362 :
train mse: 7.11617099641094

iteration 363 :
train mse: 7.092459448549614

iteration 364 :
train mse: 7.06862089148263

iteration 365 :
train mse: 7.044926287218645

iteration 366 :
train mse: 7.021449238744504

iteration 367 :
train mse: 6.99816399179225

iteration 368 :
train mse: 6.97520433300648

iteration 369 :
train mse: 6.952043519833091

iteration 370 :
train mse: 6.928890839586424

iteration 371 :
train mse: 6.90632630549062
```

```
iteration 372 :
train mse: 6.883679222188576

iteration 373 :
train mse: 6.860979817057946

iteration 374 :
train mse: 6.838559235765142

iteration 375 :
train mse: 6.816067084717173

iteration 376 :
train mse: 6.793548230559797

iteration 377 :
train mse: 6.770905349269894

iteration 378 :
train mse: 6.748661252486981

iteration 379 :
train mse: 6.726563819482643

iteration 380 :
train mse: 6.704422739771557

iteration 381 :
train mse: 6.6825977598124435

iteration 382 :
train mse: 6.660754616316982

iteration 383 :
train mse: 6.639376811778863

iteration 384 :
train mse: 6.617907105168773

iteration 385 :
train mse: 6.596327061852805

iteration 386 :
train mse: 6.575064609325376

iteration 387 :
train mse: 6.553907884845349
```

```
iteration 388 :
train mse: 6.532412966162416

iteration 389 :
train mse: 6.511238755889933

iteration 390 :
train mse: 6.490095044092662

iteration 391 :
train mse: 6.468867292052585

iteration 392 :
train mse: 6.447875937082594

iteration 393 :
train mse: 6.426951001014176

iteration 394 :
train mse: 6.406289165125251

iteration 395 :
train mse: 6.3856890039306435

iteration 396 :
train mse: 6.365133980479349

iteration 397 :
train mse: 6.344683058490111

iteration 398 :
train mse: 6.324128170587596

iteration 399 :
train mse: 6.303684568232408

iteration 400 :
train mse: 6.283387494107017

iteration 401 :
train mse: 6.263161989736789

iteration 402 :
train mse: 6.2430770494500925

iteration 403 :
train mse: 6.2229823496513275
```

```
iteration 404 :
train mse: 6.203004763997376

iteration 405 :
train mse: 6.1829953243804265

iteration 406 :
train mse: 6.163115062749596

iteration 407 :
train mse: 6.143261233130616

iteration 408 :
train mse: 6.123492663693449

iteration 409 :
train mse: 6.1040347720571075

iteration 410 :
train mse: 6.084393618471937

iteration 411 :
train mse: 6.064975849274096

iteration 412 :
train mse: 6.045753676963946

iteration 413 :
train mse: 6.026619069164849

iteration 414 :
train mse: 6.007459655939246

iteration 415 :
train mse: 5.988443111126413

iteration 416 :
train mse: 5.969363807229033

iteration 417 :
train mse: 5.95047955802423

iteration 418 :
train mse: 5.931549222234662

iteration 419 :
train mse: 5.912900692356052
```

```
iteration 420 :
train mse: 5.894222070648565

iteration 421 :
train mse: 5.875529758514428

iteration 422 :
train mse: 5.8572599519235595

iteration 423 :
train mse: 5.838750359543136

iteration 424 :
train mse: 5.820351510514925

iteration 425 :
train mse: 5.802087125875205

iteration 426 :
train mse: 5.783782076601868

iteration 427 :
train mse: 5.765467132267685

iteration 428 :
train mse: 5.747313355431399

iteration 429 :
train mse: 5.729315595239418

iteration 430 :
train mse: 5.711441184436072

iteration 431 :
train mse: 5.693491544060537

iteration 432 :
train mse: 5.675541903685002

iteration 433 :
train mse: 5.657752888612249

iteration 434 :
train mse: 5.640116365915555

iteration 435 :
train mse: 5.622432807792645
```

```
iteration 436 :
train mse: 5.605075515580112

iteration 437 :
train mse: 5.587584978918098

iteration 438 :
train mse: 5.570134429145807

iteration 439 :
train mse: 5.552688081385655

iteration 440 :
train mse: 5.535504020512326

iteration 441 :
train mse: 5.518300982809975

iteration 442 :
train mse: 5.500971478097079

iteration 443 :
train mse: 5.483855327161888

iteration 444 :
train mse: 5.4666247730574575

iteration 445 :
train mse: 5.4497400038875385

iteration 446 :
train mse: 5.4328174166083585

iteration 447 :
train mse: 5.416173110971887

iteration 448 :
train mse: 5.399339850337882

iteration 449 :
train mse: 5.382567993300637

iteration 450 :
train mse: 5.365838834128688

iteration 451 :
train mse: 5.349260540747454
```

```
iteration 452 :
train mse: 5.3327451419995455

iteration 453 :
train mse: 5.316459633980344

iteration 454 :
train mse: 5.300113671205834

iteration 455 :
train mse: 5.283828840930526

iteration 456 :
train mse: 5.267602432178847

iteration 457 :
train mse: 5.251404759768255

iteration 458 :
train mse: 5.235378692111522

iteration 459 :
train mse: 5.21929325408971

iteration 460 :
train mse: 5.2031068322277525

iteration 461 :
train mse: 5.186971512255372

iteration 462 :
train mse: 5.171200954046524

iteration 463 :
train mse: 5.1555079297391035

iteration 464 :
train mse: 5.139752146347137

iteration 465 :
train mse: 5.124192230940419

iteration 466 :
train mse: 5.108557763705405

iteration 467 :
train mse: 5.093071044639195
```

```
iteration 468 :
train mse: 5.077451758867398

iteration 469 :
train mse: 5.061965039801188

iteration 470 :
train mse: 5.046467476832681

iteration 471 :
train mse: 5.0312731364821675

iteration 472 :
train mse: 5.015816102598497

iteration 473 :
train mse: 5.000516983042035

iteration 474 :
train mse: 4.985410749397689

iteration 475 :
train mse: 4.970250838436971

iteration 476 :
train mse: 4.955001329733519

iteration 477 :
train mse: 4.9399780519417495

iteration 478 :
train mse: 4.925030410368506

iteration 479 :
train mse: 4.910284194280438

iteration 480 :
train mse: 4.895383045938295

iteration 481 :
train mse: 4.8805550939366595

iteration 482 :
train mse: 4.865756691568785

iteration 483 :
train mse: 4.851200379319702
```

```
iteration 484 :
train mse: 4.83659540505906

iteration 485 :
train mse: 4.82207162451687

iteration 486 :
train mse: 4.807714162326169

iteration 487 :
train mse: 4.793320779709107

iteration 488 :
train mse: 4.779069045565806

iteration 489 :
train mse: 4.764531303499409

iteration 490 :
train mse: 4.7502836358194696

iteration 491 :
train mse: 4.736078666004827

iteration 492 :
train mse: 4.722048011919617

iteration 493 :
train mse: 4.707905258994407

iteration 494 :
train mse: 4.693944954725352

iteration 495 :
train mse: 4.680008507041353

iteration 496 :
train mse: 4.665959418322237

iteration 497 :
train mse: 4.651983390394851

iteration 498 :
train mse: 4.638081372100645

iteration 499 :
train mse: 4.624276813378339
```

```
iteration 500 :
train mse: 4.61072125776254
```

[43]:
```
from tqdm import tqdm
df_user = []
for key in map_msno:
    val = map_msno[key]
    df_user.append([key] +list(emb_user[val]))
```

[44]:
```
df_song = []
for key in map_song_id:
    val = map_song_id[key]
    df_song.append([key] +list(emb_song[val]))
```

[48]:
```
df_user = pd.DataFrame(df_user)
df_song = pd.DataFrame(df_song)

df_user = df_user.rename(columns = {0: 'msno'})
df_song = df_song.rename(columns = {0: 'song_id'})

df_user.to_csv('user_embedding.csv')
df_song.to_csv('song_embedding.csv')
```

[7]:
```
df_user = pd.read_csv('user_embedding.csv', index_col = 0)
df_song = pd.read_csv('song_embedding.csv', index_col = 0)

df = df.merge(df_user, how = 'left', on = 'msno')
df = df.merge(df_song, how = 'left', on = 'song_id')
```

2. Creating Expiration and registration year, Month, day
3. Creating Membership Duration

[ ]:
```
df['expiration_year'] = df.expiration_date.astype(str).apply(lambda x: x[:4]).
↪astype(np.uint16)
df['expiration_month'] = df.expiration_date.astype(str).apply(lambda x: x[4:6]).
↪astype(np.uint8)
df['expiration_day'] = df.expiration_date.astype(str).apply(lambda x: x[6:]).
↪astype(np.uint8)
df['registration_year'] = df.registration_init_time.astype(str).apply(lambda x:␣
↪x[:4]).astype(np.uint16)
df['registration_month'] = df.registration_init_time.astype(str).apply(lambda x:
↪ x[4:6]).astype(np.uint8)
df['registration_day'] = df.registration_init_time.astype(str).apply(lambda x:␣
↪x[6:]).astype(np.uint8)

a = list(pd.to_datetime(df['expiration_date'], format = '%Y%m%d') - pd.
↪to_datetime(df['registration_init_time'], format = '%Y%m%d'))
df['membership_duration'] = [i.days for i in a]
```

```
df.drop(columns = ['expiration_date', 'registration_init_time'], inplace =True)
```

4. Creating Number of songs heard by users

```
[ ]: a = pd.DataFrame(df['msno'].value_counts().reset_index())
a.rename(columns = {'index':'msno', 'msno':'song_count'}, inplace=True)
df = df.merge(a, how = 'left',on = 'msno')
df['song_count'] = np.log(df['song_count'])
df.rename(columns = {'song_count':'log_song_count'}, inplace=True)
```

4. Handling Outliers of age

```
[ ]: def clean(x):
    if x>=0 and x<=100:
        return x
    elif x <0:
        return 0
    elif x>100:
        return 100
df['bd'] = df.bd.apply(lambda x: clean(x)).astype(np.uint8)
```

5. Handling Most popular Genre id

```
[ ]: most_popular_genre = ['465', '458', '921', '1609', '444', '1259', '2022',␣
↪'359', '139']
def clean(x):
    b = re.findall('[0-9]+', x)
    if len(b) == 0:
        return np.nan
    if len(b) == 1:
        return int(b[0])
    if len(b)>1:
        for i in most_popular_genre:
            if i in b:
                return int(i)
        return b[0]
df['genre_ids'] = df['genre_ids'].astype(str).apply(lambda x: clean(x))
df['genre_ids'] = df['genre_ids'].astype(str).apply(lambda x: x if x in␣
↪most_popular_genre else '0').astype(np.uint16)
```

```
[ ]: df.drop(columns = ['msno', 'song_id', 'artist_name'], inplace = True)
```

```
[ ]: Y = df['target']
x = df.drop(columns = ['target'])
```

```
[ ]: from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, Y, random_state = 0,␣
↪stratify = Y, test_size = 0.2)
```

```
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train,␣
↪random_state = 0, stratify = y_train, test_size = 0.2)
```

6. Applying One Hot Encoding on categorical Feature

```
[ ]: categorical_features = ['source_system_tab', 'source_screen_name',␣
↪'source_type', 'city', 'registered_via', 'language', 'gender']
x_train_ohe = pd.get_dummies(x_train[categorical_features])
x_val_ohe = pd.get_dummies(x_val[categorical_features])
x_test_ohe = pd.get_dummies(x_test[categorical_features])
```

```
[ ]: x_train = pd.concat([x_train, x_train_ohe], axis = 1)
x_val = pd.concat([x_val, x_val_ohe], axis = 1)
x_test = pd.concat([x_test, x_test_ohe], axis = 1)
x_train.drop(columns = ['city', 'registered_via', 'language',␣
↪'source_system_tab', 'source_screen_name', 'source_type', 'gender'], inplace␣
↪= True)
x_val.drop(columns = ['city', 'registered_via', 'language',␣
↪'source_system_tab', 'source_screen_name', 'source_type', 'gender'], inplace␣
↪= True)
x_test.drop(columns = ['city', 'registered_via', 'language',␣
↪'source_system_tab', 'source_screen_name', 'source_type', 'gender'], inplace␣
↪= True)
```

```
[ ]: x_train.drop(columns = ['composer', 'lyricist'], inplace = True)
x_val.drop(columns = ['composer', 'lyricist'], inplace = True)
x_test.drop(columns = ['composer', 'lyricist'], inplace = True)
```

```
[ ]: from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(x_train['song_length'])
x_train['song_length'] = scaler.transform(x_train['song_length'].values.
↪reshape(-1, 1))
x_val['song_length'] = scaler.transform(x_val['song_length'].values.reshape(-1,␣
↪1))
x_test['song_length'] = scaler.transform(x_test['song_length'].values.
↪reshape(-1, 1))
```

7. Changing data type of feature to reduce space

```
[ ]: import numpy as np
x_train['bd'] = x_train['bd'].astype(np.uint8)
x_val['bd'] = x_val['bd'].astype(np.uint8)
x_test['bd'] = x_test['bd'].astype(np.uint8)
```

```
[ ]: x_train['genre_ids'] = x_train['genre_ids'].astype(np.uint16)
x_val['genre_ids'] = x_val['genre_ids'].astype(np.uint16)
x_test['genre_ids'] = x_test['genre_ids'].astype(np.uint16)
```

8. Changing missing values to median

```
[ ]: median = x_train['song_length'].median()
     x_train['song_length'].fillna(median, inplace = True)
     x_test['song_length'].fillna(median, inplace = True)
     x_val['song_length'].fillna(median, inplace = True)
```

```
[ ]: x_train.to_csv('x_train.csv'), y_train.to_csv('y_train.csv')
     x_test.to_csv('x_test.csv'), y_test.to_csv('y_test.csv')
     x_val.to_csv('x_val.csv'), y_val.to_csv('y_val.csv')
```

3 Feature Engineering Conclusion

1.Creating 30 Features of user embedding which helps in determining the user. 2.Creating 30 Features of song embedding which helps in determing the song.3.Creating 1 feature of Number of days of membership. 4.Creating 1 feature of Number of songs heard by user. 5. Creating 6 features of Expiration and registration year, month and day of users.6. Applying log Transformation of song count feature as it follows normal distribution.7. keeping most popular genre id among all.8. Applying one hot encoding to all categorical features.9. Treating missing values as a new category and dropping it in applying one hot encoding for categorical features.10. filling missing values as median value in continuous features.

4 Modeling

```
[10]: def plot_confusion_matrix(test_y, predict_y):
          C = confusion_matrix(test_y, predict_y)
          print("Percentage of misclassified points ",round((len(test_y)-np.trace(C))/
       ↪len(test_y)*100, 3), '%')
          A =(((C.T)/(C.sum(axis=1))).T)
          B =(C/C.sum(axis=0))
          labels = [1,2]
          cmap=sns.light_palette("green")
          # representing A in heatmap format
          print("-"*50, "Confusion matrix", "-"*50)
          plt.figure(figsize=(10,5))
          sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels,␣
       ↪yticklabels=labels)
          plt.xlabel('Predicted Class')
          plt.ylabel('Original Class')
          plt.show()

          print("-"*50, "Precision matrix", "-"*50)
          plt.figure(figsize=(10,5))
          sns.heatmap(B, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels,␣
       ↪yticklabels=labels)
```

```
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
    print("Sum of columns in precision matrix",B.sum(axis=0))

    # representing B in heatmap format
    print("-"*50, "Recall matrix"    , "-"*50)
    plt.figure(figsize=(10,5))
    sns.heatmap(A, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels,␣
 ↪yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
    print("Sum of rows in precision matrix",A.sum(axis=1))
```

[2]:
```
import dask.dataframe as pd
x_train = pd.read_csv('E:/x_train.csv')
x_test = pd.read_csv('x_test.csv')
x_val = pd.read_csv('x_val.csv')
y_train = pd.read_csv('y_train.csv')
y_test = pd.read_csv('y_test.csv')
y_val = pd.read_csv('y_val.csv')
```

[3]:
```
x_train = x_train.drop(columns = ['Unnamed: 0'])
x_test = x_test.drop(columns = ['Unnamed: 0'])
x_val = x_val.drop(columns = ['Unnamed: 0'])
y_train = y_train.drop(columns = ['Unnamed: 0'])
y_test = y_test.drop(columns = ['Unnamed: 0'])
y_val = y_val.drop(columns = ['Unnamed: 0'])
```

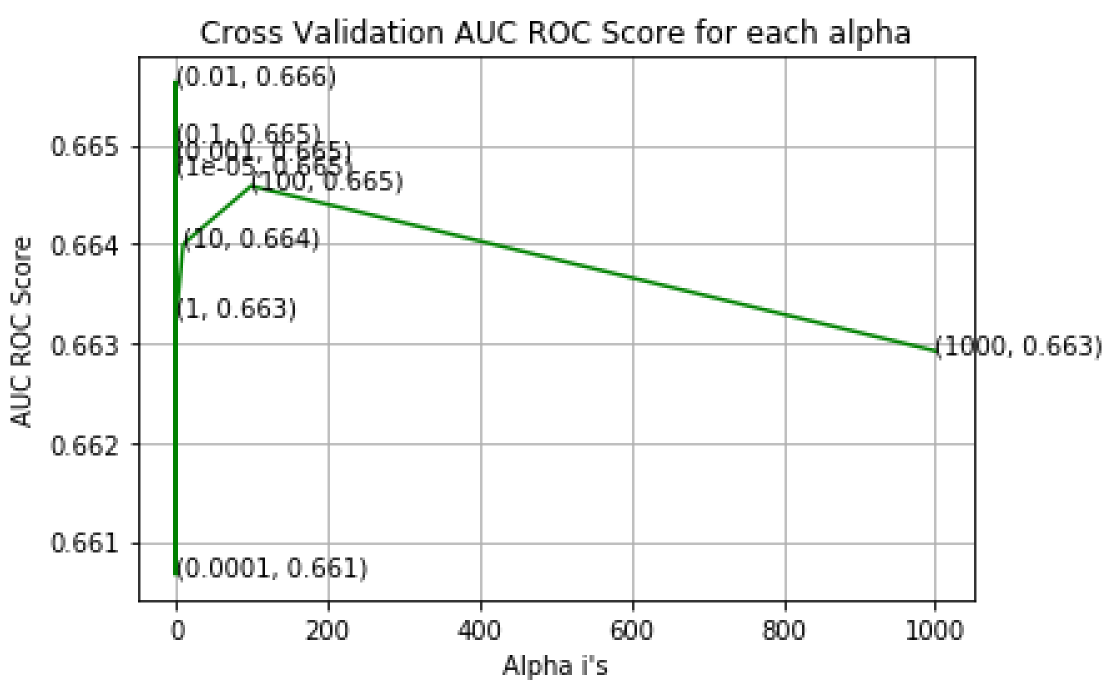
[7]:
```
import warnings
import numpy as np
warnings.filterwarnings("ignore")
```

1. Logistic Regression

[5]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

alphas = [10 ** x for x in range(-5, 4)]
cv_logs = []
for i in alphas:
    regressor = LogisticRegression(penalty='l2',C=i,class_weight='balanced')
    regressor.fit(x_train, y_train)
    sig_clf = CalibratedClassifierCV(regressor, method="sigmoid")
    sig_clf.fit(x_train, y_train)
```

```
    y_pred = sig_clf.predict_proba(x_val)
    score = roc_auc_score(y_val, y_pred[:, 1])
    print ('AUC ROC Score for c = ',i,'is',score)
    cv_logs.append(score)

fig, ax = plt.subplots()
ax.plot(alphas, cv_logs,c='g')
for i, txt in enumerate(np.round(cv_logs,3)):
    ax.annotate((alphas[i],np.round(txt,3)), (alphas[i],cv_logs[i]))
plt.grid()
plt.title("Cross Validation AUC ROC Score for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("AUC ROC Score")
plt.show()
```

```
AUC ROC Score for c =  1e-05 is 0.6647200724158513
AUC ROC Score for c =  0.0001 is 0.66066237329238
AUC ROC Score for c =  0.001 is 0.6648746192677282
AUC ROC Score for c =  0.01 is 0.6656384442856027
AUC ROC Score for c =  0.1 is 0.6650492350905566
AUC ROC Score for c =  1 is 0.6632797240083415
AUC ROC Score for c =  10 is 0.6639936211617927
AUC ROC Score for c =  100 is 0.6645825333981991
AUC ROC Score for c =  1000 is 0.6629217742893395
```



[5]:

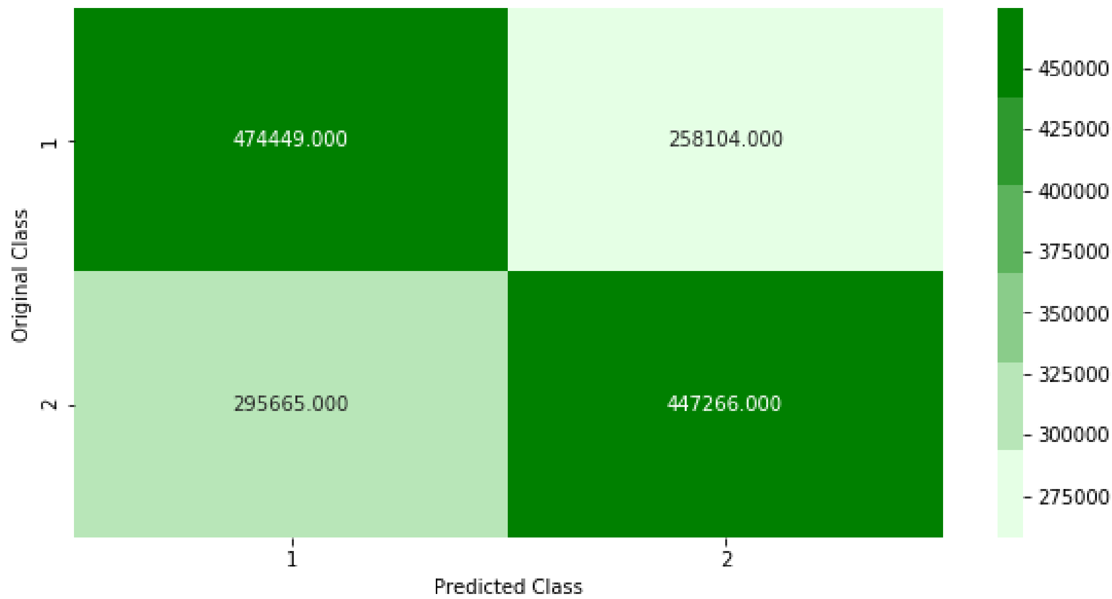
```
from sklearn.linear_model import LogisticRegression
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

regressor = LogisticRegression(penalty='l2',C=0.01,class_weight='balanced')
regressor.fit(x_train, y_train)
sig_clf = CalibratedClassifierCV(regressor, method="sigmoid")
sig_clf.fit(x_train, y_train)

y_pred = sig_clf.predict_proba(x_train)
score = roc_auc_score(y_train, y_pred[:, 1])
print ('Train AUC ROC Score for c = ',0.01,'is',score)
y_pred = sig_clf.predict_proba(x_val)
score = roc_auc_score(y_val, y_pred[:, 1])
print ('validation AUC ROC Score for c = ',0.01,'is',score)
y_pred = sig_clf.predict_proba(x_test)
score = roc_auc_score(y_test, y_pred[:, 1])
print ('Test AUC ROC Score for c = ',0.01,'is',score)
```

```
Train AUC ROC Score for c =  0.01 is 0.6652841982097703
validation AUC ROC Score for c =  0.01 is 0.6647462700597071
Test AUC ROC Score for c =  0.01 is 0.6650204873576647
```

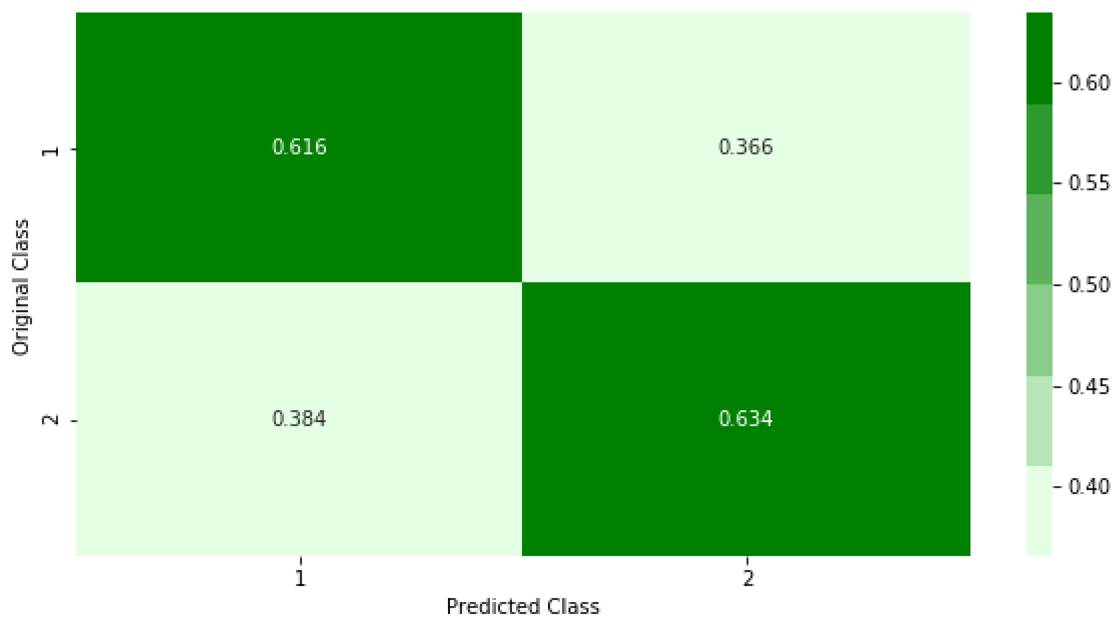
[6]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
plot_confusion_matrix(y_test, sig_clf.predict(x_test))
```

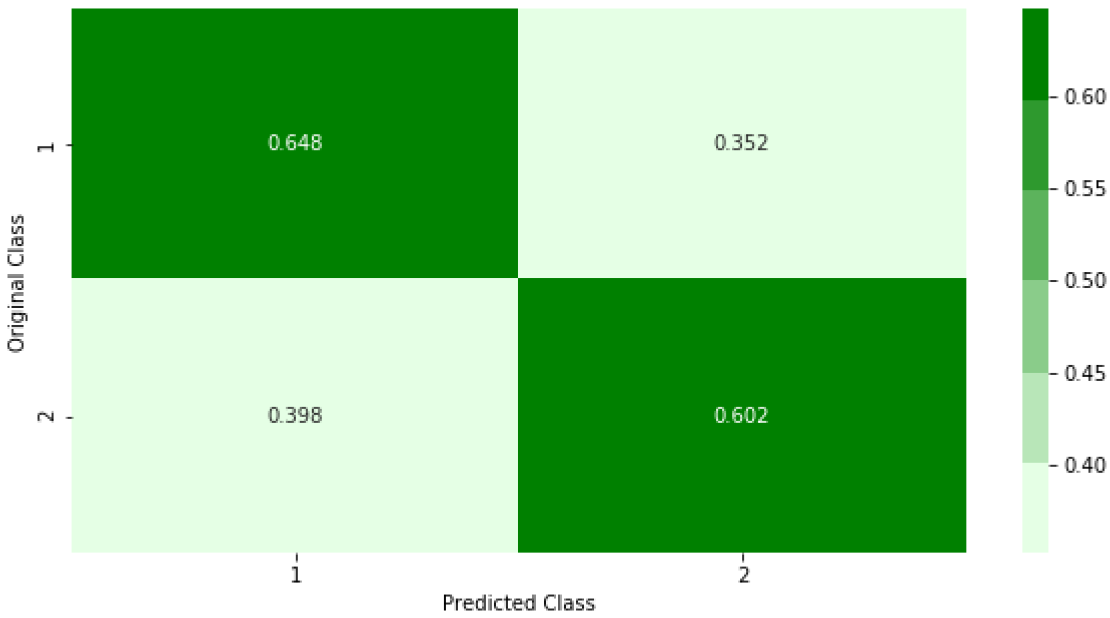
```
Percentage of misclassified points  37.531 %
-------------------------------------------------- Confusion matrix
--------------------------------------------------
```



-- Precision matrix



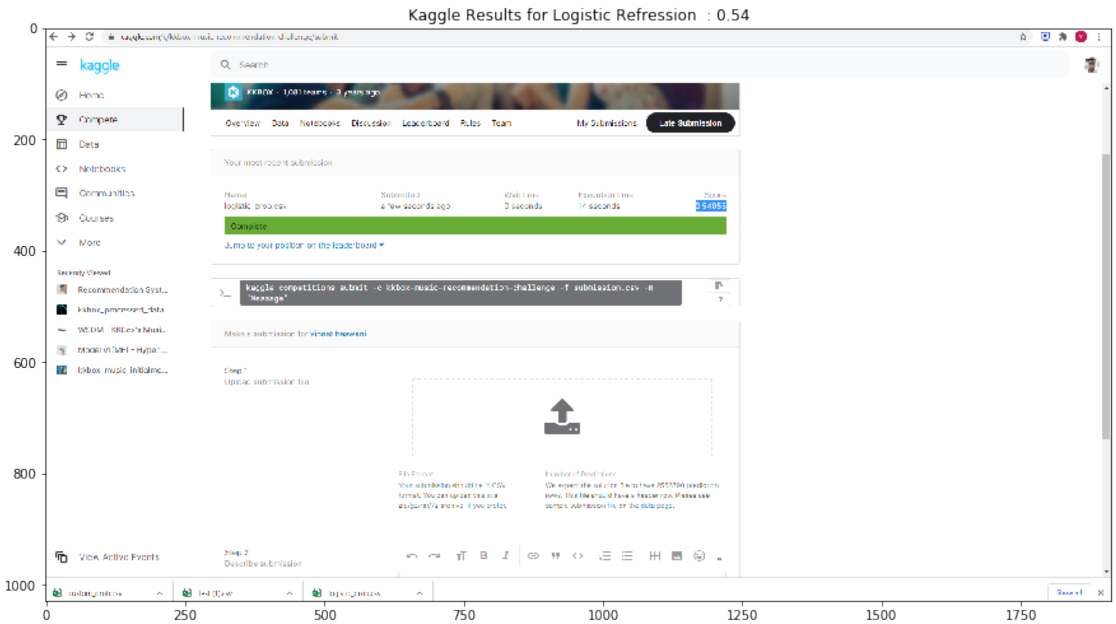
Sum of columns in precision matrix [1. 1.]
-- Recall matrix




```
Sum of rows in precision matrix [1. 1.]
```

[16]:
```
import matplotlib.pyplot as plt
import cv2
plt.figure(figsize = (15, 20))
plt.title('Kaggle Results for Logistic Refression  : 0.54')
plt.imshow(cv2.cvtColor(cv2.imread('logistic Regression prob.png'), cv2.
 ↪COLOR_BGR2RGB))
```

[16]:
```
<matplotlib.image.AxesImage at 0x1d7531dc208>
```

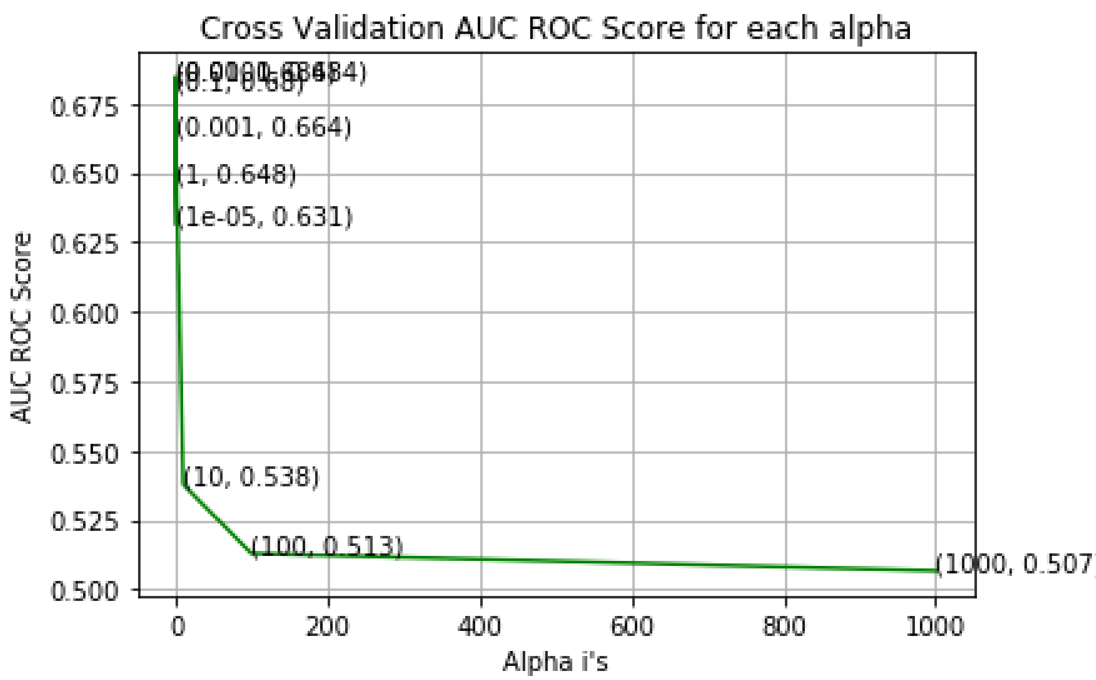


2. SGDClassifier with log loss

[6]:
```
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

alphas = [10 ** x for x in range(-5, 4)]
cv_logs = []
for i in alphas:
    regressor = SGDClassifier(penalty='l2',alpha=i,class_weight='balanced')
    regressor.fit(x_train, y_train)
    sig_clf = CalibratedClassifierCV(regressor, method="sigmoid")
    sig_clf.fit(x_train, y_train)
    y_pred = sig_clf.predict_proba(x_val)
    score = roc_auc_score(y_val, y_pred[:, 1])
    print ('AUC ROC Score for c = ',i,'is',score)
    cv_logs.append(score)

fig, ax = plt.subplots()
ax.plot(alphas, cv_logs,c='g')
for i, txt in enumerate(np.round(cv_logs,3)):
    ax.annotate((alphas[i],np.round(txt,3)), (alphas[i],cv_logs[i]))
plt.grid()
plt.title("Cross Validation AUC ROC Score for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("AUC ROC Score")
```

```
plt.show()
```

```
AUC ROC Score for c =  1e-05 is 0.6314927694977276
AUC ROC Score for c =  0.0001 is 0.6843973579421669
AUC ROC Score for c =  0.001 is 0.6641265100237393
AUC ROC Score for c =  0.01 is 0.6843594993560639
AUC ROC Score for c =  0.1 is 0.6802011765466669
AUC ROC Score for c =  1 is 0.647585506803966
AUC ROC Score for c =  10 is 0.5380404000104143
AUC ROC Score for c =  100 is 0.5129237882688955
AUC ROC Score for c =  1000 is 0.5067033561296438
```

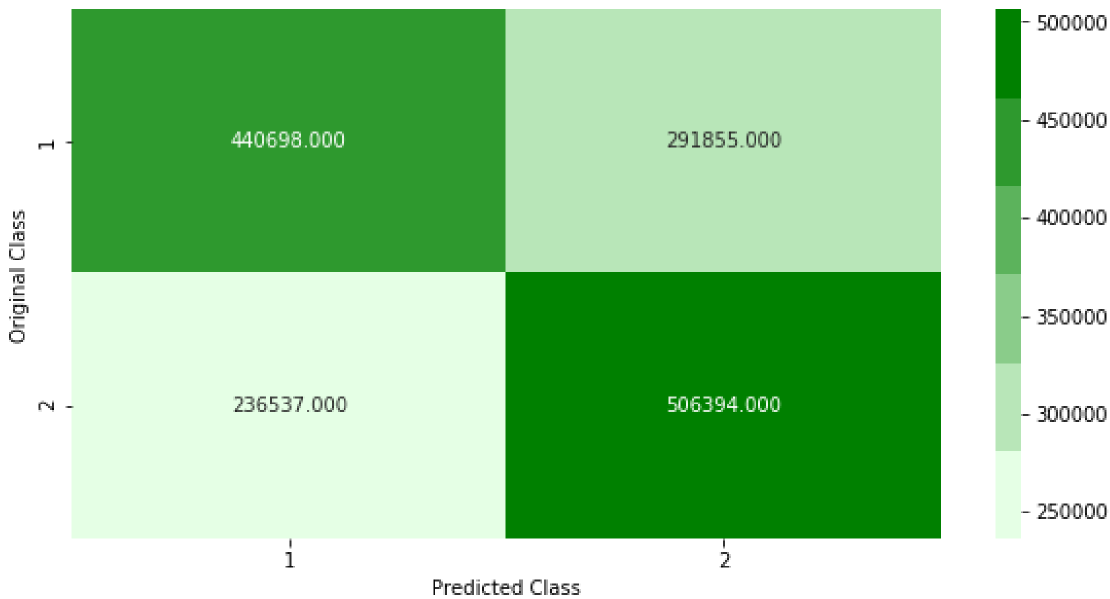


[7]:
```
regressor = SGDClassifier(penalty='l2',alpha=0.01,class_weight='balanced')
regressor.fit(x_train, y_train)
sig_clf = CalibratedClassifierCV(regressor, method="sigmoid")
sig_clf.fit(x_train, y_train)
y_pred = sig_clf.predict_proba(x_val)
score = roc_auc_score(y_val, y_pred[:, 1])
print ('validation AUC ROC Score for c = ',0.01,'is',score)
y_pred = sig_clf.predict_proba(x_test)
score = roc_auc_score(y_test, y_pred[:, 1])
print ('Test AUC ROC Score for c = ',0.01,'is',score)
```

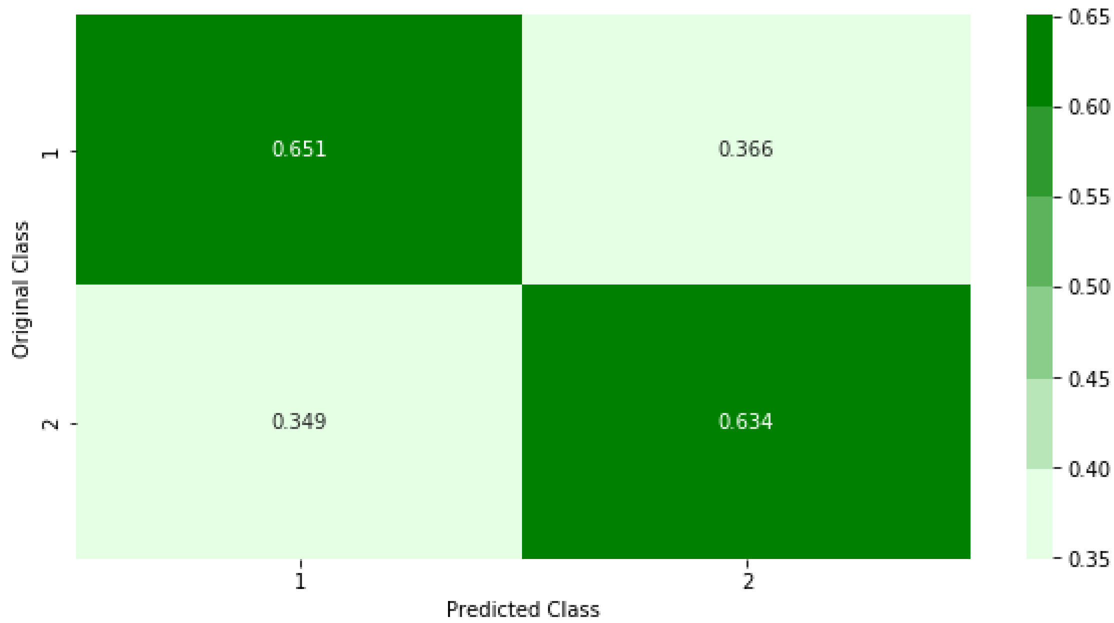
```
validation AUC ROC Score for c =  0.01 is 0.6843594993560639
Test AUC ROC Score for c =  0.01 is 0.6846739588359931
```

[20]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
plot_confusion_matrix(y_test, sig_clf.predict(x_test))
```

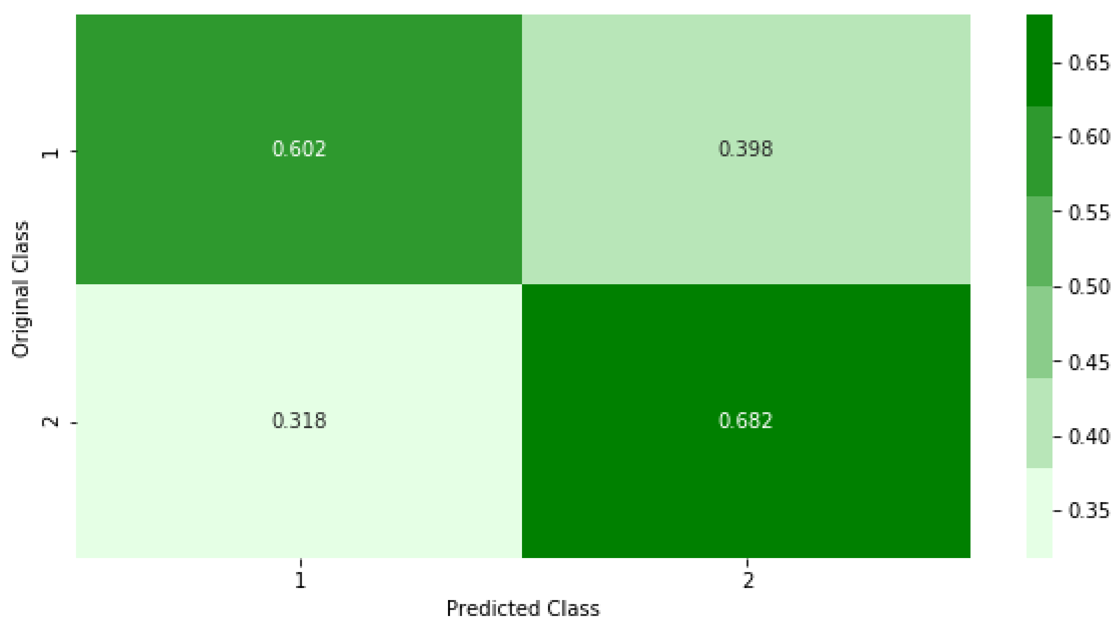
```
Percentage of misclassified points  35.811 %
-------------------------------------------------- Confusion matrix
--------------------------------------------------
```




```
-------------------------------------------------- Precision matrix
--------------------------------------------------
```



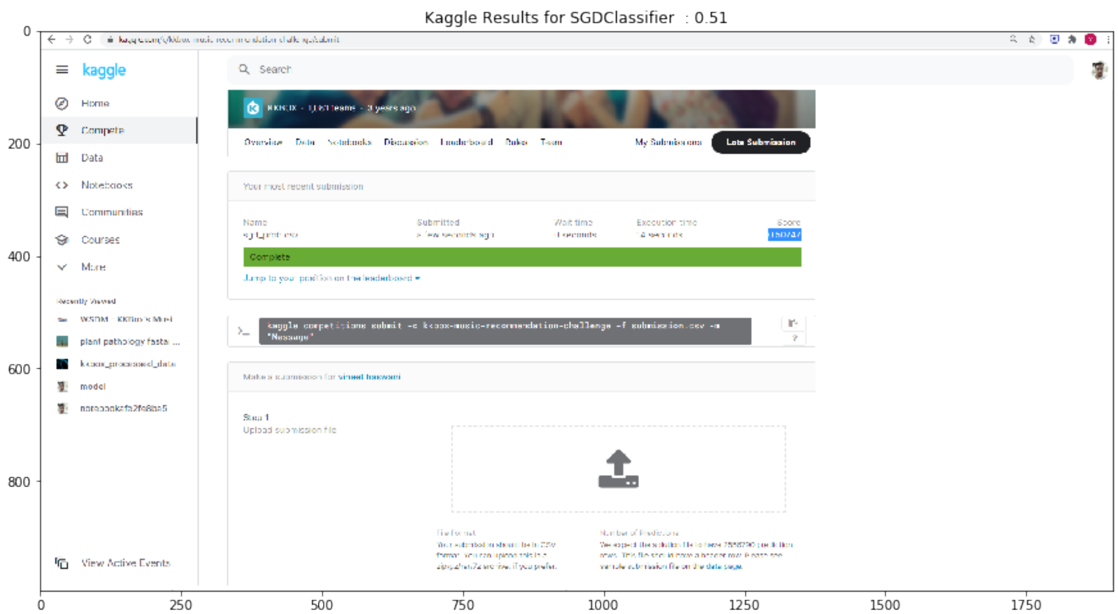

```
Sum of columns in precision matrix [1. 1.]
--------------------------------------------------- Recall matrix
---------------------------------------------------
```




```
Sum of rows in precision matrix [1. 1.]
```

```
[18]: import matplotlib.pyplot as plt
      import cv2
      plt.figure(figsize = (15, 20))
      plt.title('Kaggle Results for SGDClassifier  : 0.51')
      plt.imshow(cv2.cvtColor(cv2.imread('sgd_prob.png'), cv2.COLOR_BGR2RGB))
```

```
[18]: <matplotlib.image.AxesImage at 0x1d7577b0388>
```

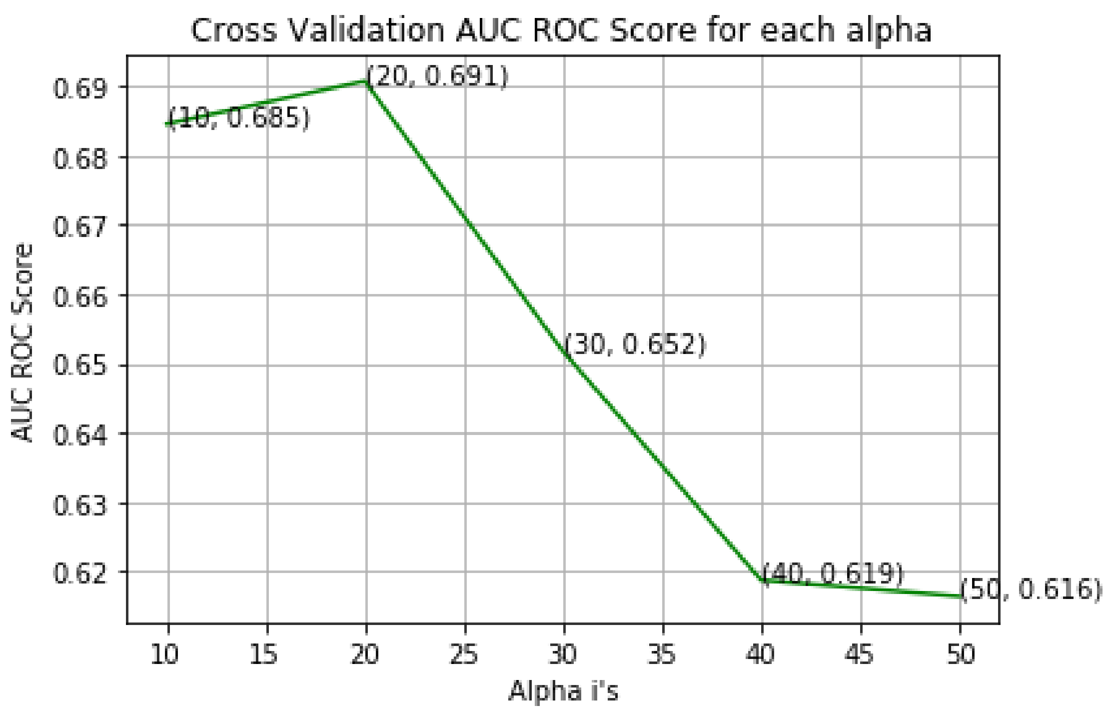


3. Decision Tree

```
[6]: from sklearn.tree import DecisionTreeClassifier
     from sklearn.calibration import CalibratedClassifierCV
     from sklearn.metrics import roc_auc_score
     import matplotlib.pyplot as plt

     depth=[10, 20, 30, 40, 50]
     cv_logs=[]
     for i in depth:
         classifier=DecisionTreeClassifier(max_depth=i,random_state=42)
         classifier.fit(x_train, y_train)
         y_pred = classifier.predict_proba(x_val)
         score = roc_auc_score(y_val, y_pred[:, 1])
         print ('AUC ROC Score for depth = ',i,'is',score)
         cv_logs.append(score)

     fig, ax = plt.subplots()
     ax.plot(alphas, cv_logs,c='g')
```

```
for i, txt in enumerate(np.round(cv_logs,3)):
    ax.annotate((alphas[i],np.round(txt,3)), (alphas[i],cv_logs[i]))
plt.grid()
plt.title("Cross Validation AUC ROC Score for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("AUC ROC Score")
plt.show()
```

```
AUC ROC Score for depth =  10 is 0.6845244398493214
AUC ROC Score for depth =  20 is 0.6907042320141951
AUC ROC Score for depth =  30 is 0.6517873490908819
AUC ROC Score for depth =  40 is 0.6187848824044917
AUC ROC Score for depth =  50 is 0.6164559012685721
```

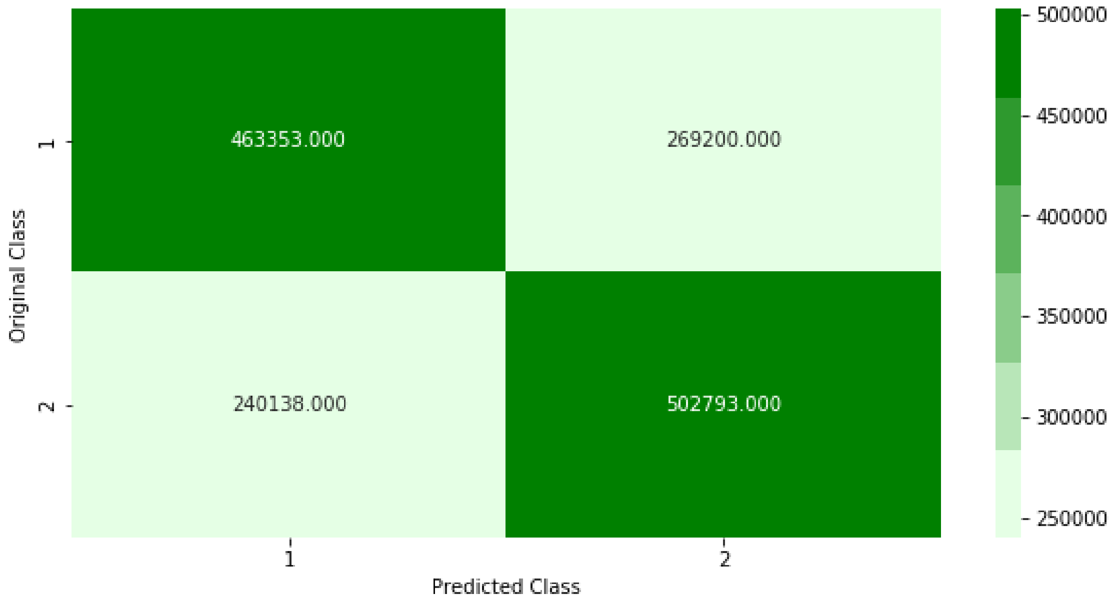


[13]:
```
classifier=DecisionTreeClassifier(max_depth=20,random_state=42)
classifier.fit(x_train, y_train)
y_pred = classifier.predict_proba(x_val)
score = roc_auc_score(y_val, y_pred[:, 1])
print ('validation AUC ROC Score for depth = ',20,'is',score)
y_pred = classifier.predict_proba(x_test)
score = roc_auc_score(y_test, y_pred[:, 1])
print ('Test AUC ROC Score for depth = ',20,'is',score)
```

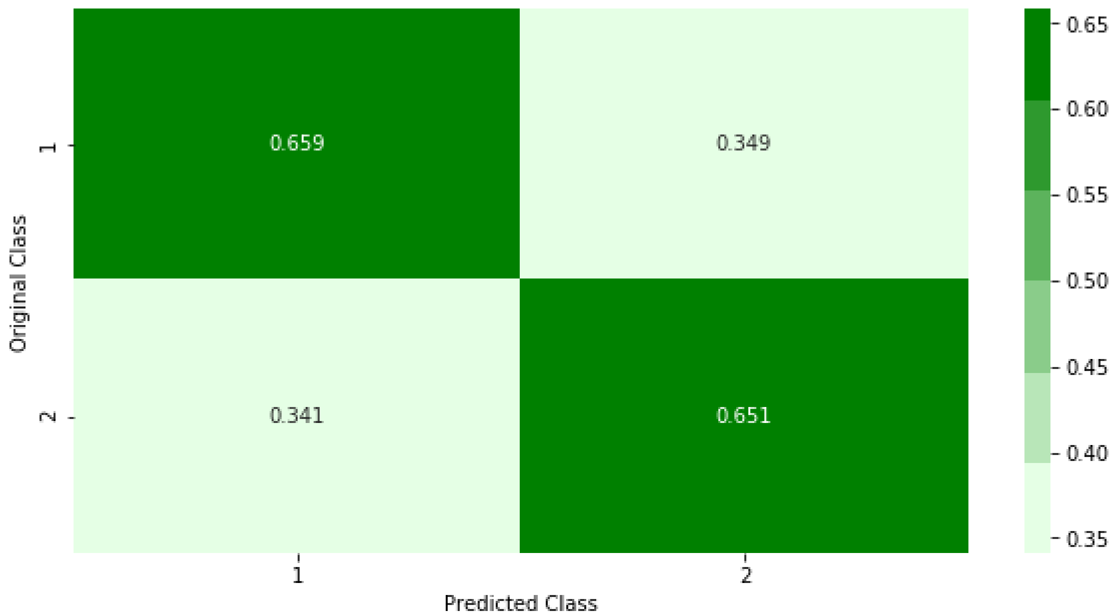
```
validation AUC ROC Score for depth =  20 is 0.6907042320141951
Test AUC ROC Score for depth =  20 is 0.6924008796996413
```

[24]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
plot_confusion_matrix(y_test, classifier.predict(x_test))
```

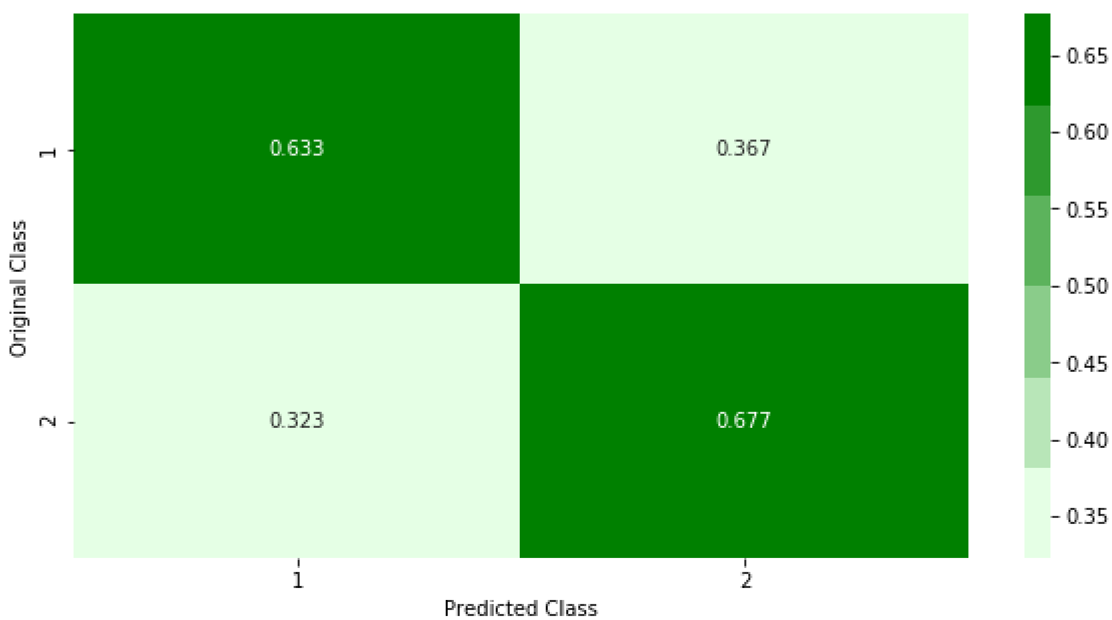
```
Percentage of misclassified points  34.52 %
-------------------------------------------------- Confusion matrix
--------------------------------------------------
```




```
-------------------------------------------------- Precision matrix
--------------------------------------------------
```



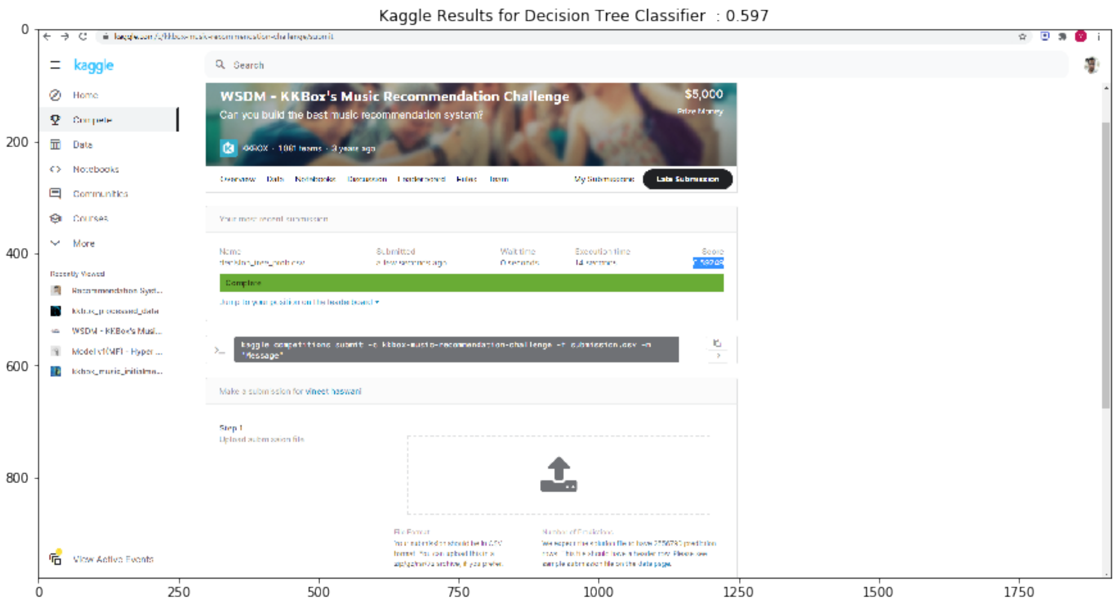

```
Sum of columns in precision matrix [1. 1.]
-------------------------------------------------- Recall matrix
--------------------------------------------------
```




```
Sum of rows in precision matrix [1. 1.]
```

```
[20]: import matplotlib.pyplot as plt
      import cv2
      plt.figure(figsize = (15, 20))
      plt.title('Kaggle Results for Decision Tree Classifier  : 0.597')
      plt.imshow(cv2.cvtColor(cv2.imread('Decision Tree prob.png'), cv2.
       ↪COLOR_BGR2RGB))
```

```
[20]: <matplotlib.image.AxesImage at 0x1d7575283c8>
```

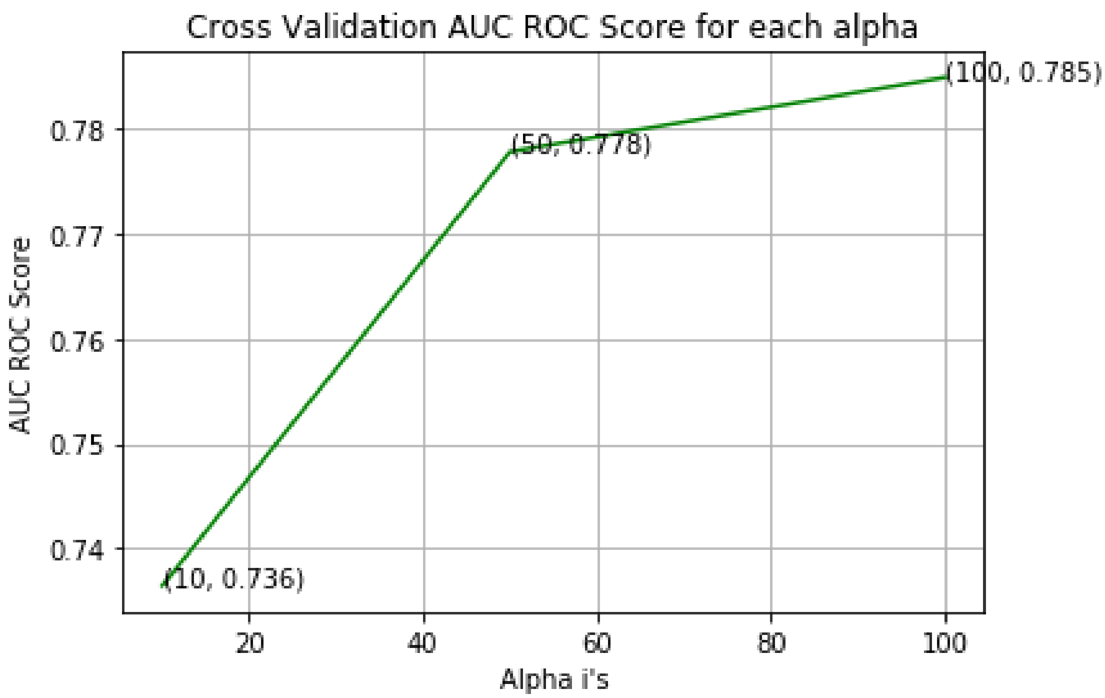


4. Random Forest

```
[9]: from sklearn.ensemble import RandomForestClassifier
     from sklearn.calibration import CalibratedClassifierCV
     from sklearn.metrics import roc_auc_score
     import matplotlib.pyplot as plt

     alphas=[10,50,100]
     cv_logs=[]
     for i in alphas:
         classifier=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1)
         classifier.fit(x_train, y_train)
         y_pred = classifier.predict_proba(x_val)
         score = roc_auc_score(y_val, y_pred[:, 1])
         print ('AUC ROC Score for c = ',i,'is',score)
         cv_logs.append(score)

     fig, ax = plt.subplots()
```

```
ax.plot(alphas, cv_logs,c='g')
for i, txt in enumerate(np.round(cv_logs,3)):
    ax.annotate((alphas[i],np.round(txt,3)), (alphas[i],cv_logs[i]))
plt.grid()
plt.title("Cross Validation AUC ROC Score for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("AUC ROC Score")
plt.show()
```

```
AUC ROC Score for c =  10 is 0.736468057712352
AUC ROC Score for c =  50 is 0.7777900131752978
AUC ROC Score for c =  100 is 0.7848667134629947
```



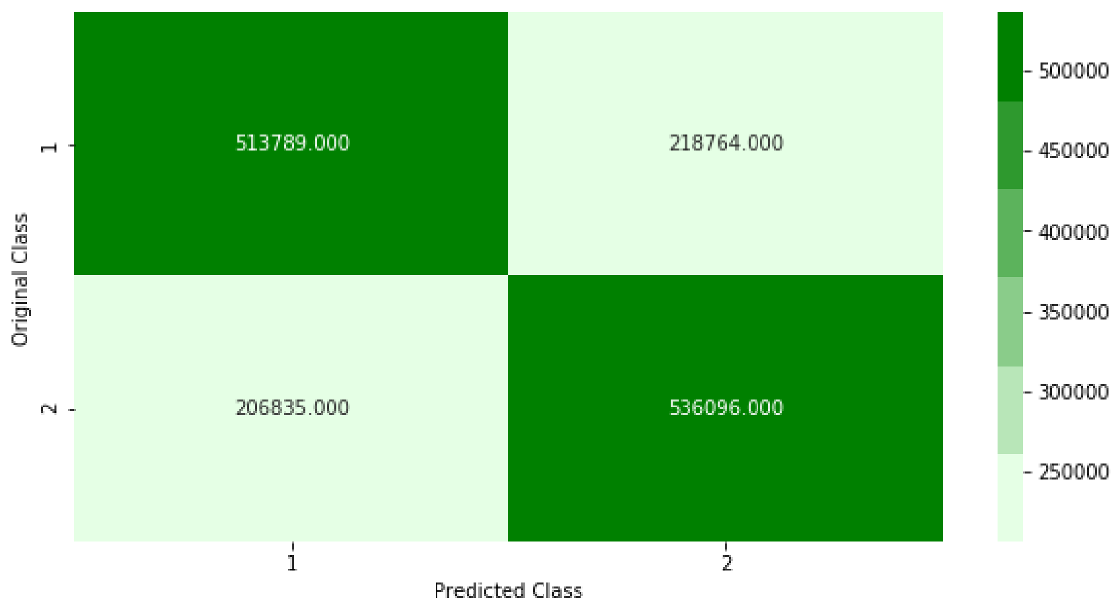
[1]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

classifier=RandomForestClassifier(n_estimators=100,random_state=42, n_jobs = 6)
classifier.fit(x_train, y_train)
y_pred = classifier.predict_proba(x_val)
score = roc_auc_score(y_val, y_pred[:, 1])
print ('validation AUC ROC Score for c = ',100,'is',score)
y_pred = classifier.predict_proba(x_test)
```

```
score = roc_auc_score(y_test, y_pred[:, 1])
print ('Test AUC ROC Score for c = ',100,'is',score)
```

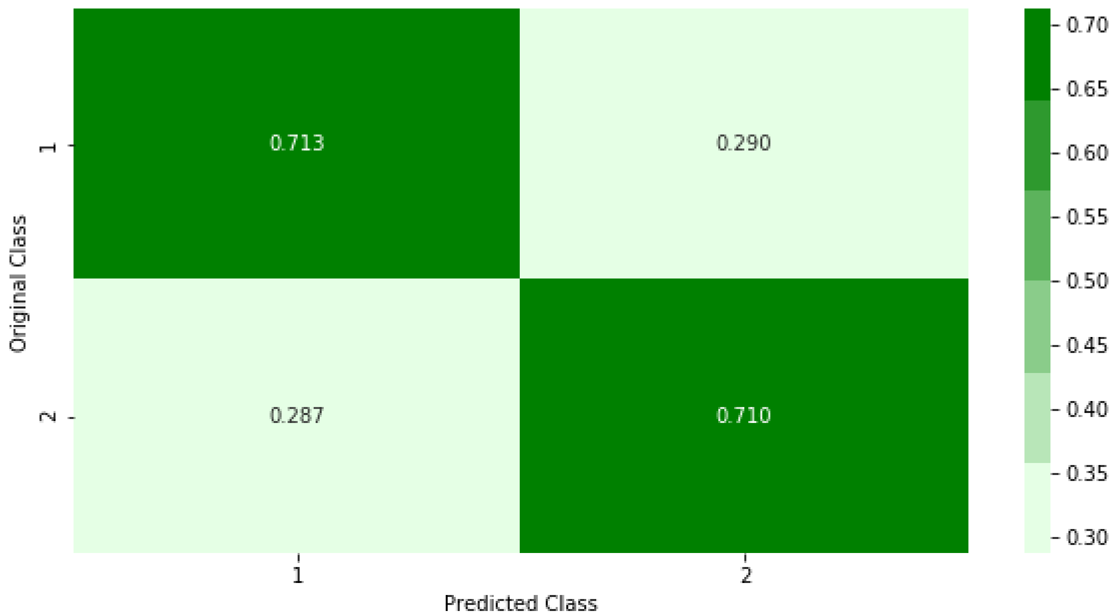
```
validation AUC ROC Score for c =  100 is 0.7848667134629947
Test AUC ROC Score for c =  100 is 0.7856958254261837
```

[22]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
plot_confusion_matrix(y_test, classifier.predict(x_test))
```

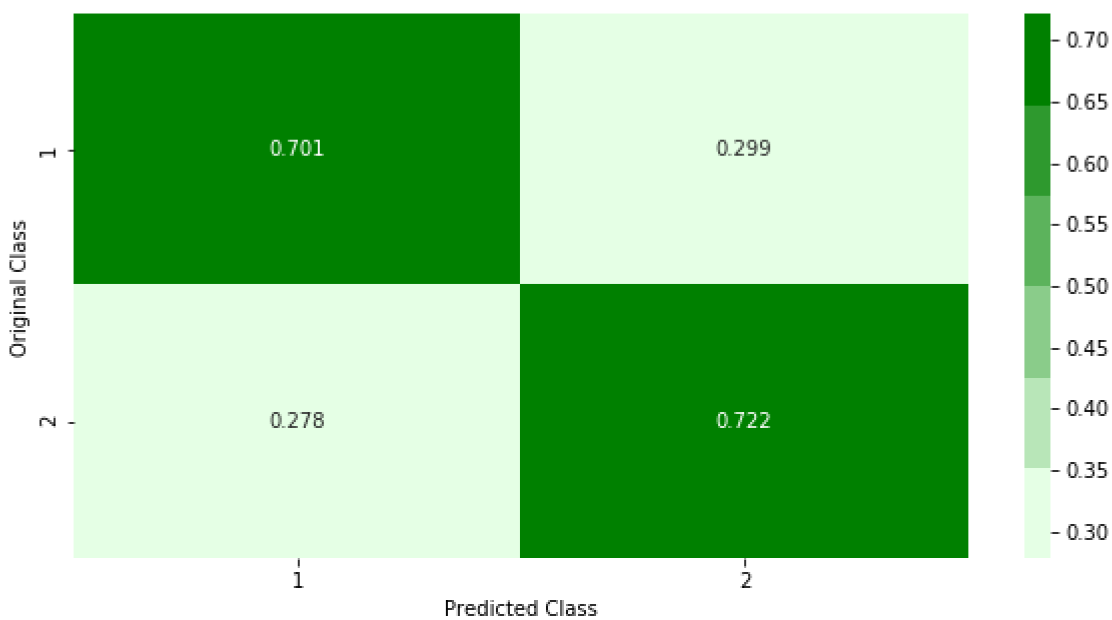
```
Percentage of misclassified points  28.845 %
-------------------------------------------------- Confusion matrix
--------------------------------------------------
```




```
-------------------------------------------------- Precision matrix
--------------------------------------------------
```



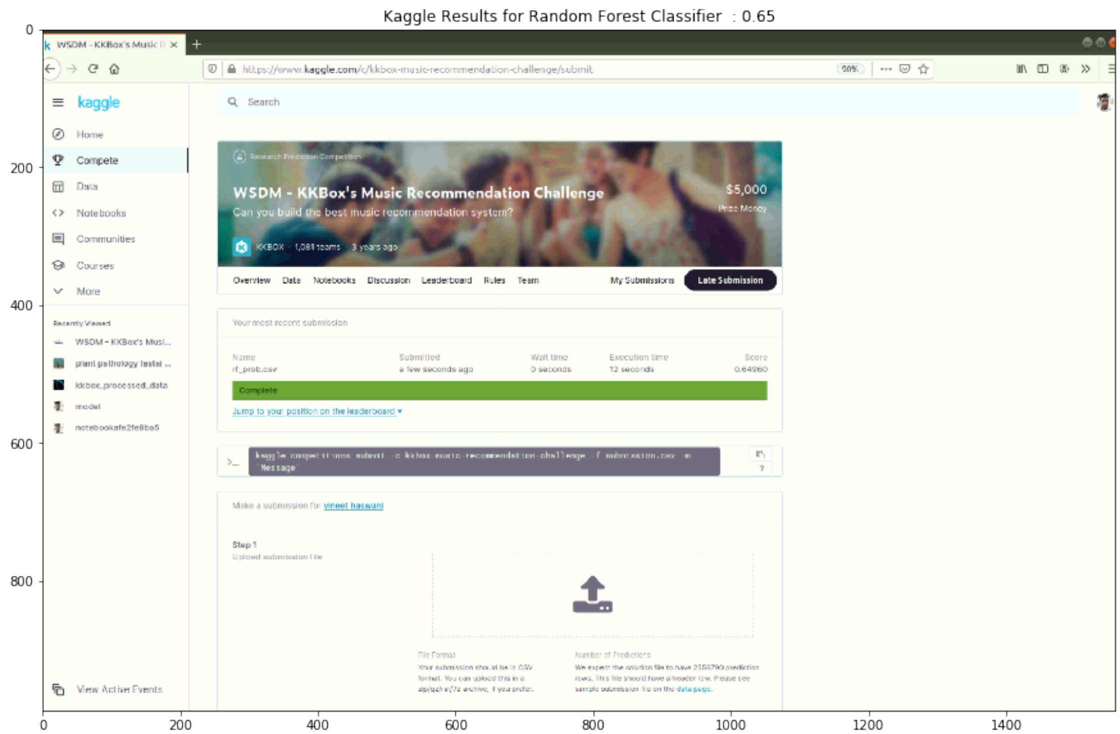

```
Sum of columns in precision matrix [1. 1.]
--------------------------------------------------- Recall matrix
---------------------------------------------------
```




```
Sum of rows in precision matrix [1. 1.]
```

```
[28]: import matplotlib.pyplot as plt
      import cv2
      plt.figure(figsize = (15, 20))
      plt.title('Kaggle Results for Random Forest Classifier  : 0.65')
      plt.imshow(cv2.cvtColor(cv2.imread('Random Forest prob.png'), cv2.
       ↪COLOR_BGR2RGB))
```

```
[28]: <matplotlib.image.AxesImage at 0x1d757fa4cc8>
```

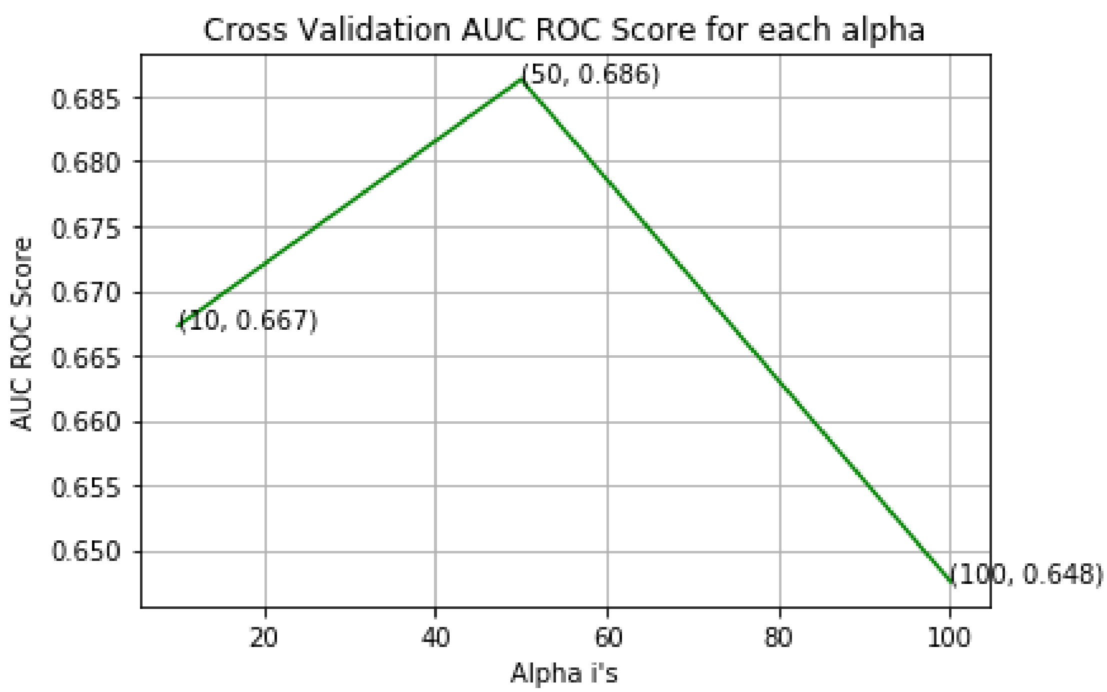


5. AdaBoost

```
[5]: from sklearn.ensemble import AdaBoostClassifier
     from sklearn.calibration import CalibratedClassifierCV
     from sklearn.metrics import roc_auc_score
     import matplotlib.pyplot as plt

     alphas=[10,50,100]
     cv_logs=[]
     for i in alphas:
         classifier=AdaBoostClassifier(n_estimators=i,random_state=42,n_jobs=-1)
         classifier.fit(x_train, y_train)
         y_pred = classifier.predict_proba(x_val)
         score = roc_auc_score(y_val, y_pred[:, 1])
         print ('AUC ROC Score for c = ',i,'is',score)
```

```
    cv_logs.append(score)

fig, ax = plt.subplots()
ax.plot(alphas, cv_logs,c='g')
for i, txt in enumerate(np.round(cv_logs,3)):
    ax.annotate((alphas[i],np.round(txt,3)), (alphas[i],cv_logs[i]))
plt.grid()
plt.title("Cross Validation AUC ROC Score for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("AUC ROC Score")
plt.show()
```

```
AUC ROC Score for c =  10 is 0.6672817891319418
AUC ROC Score for c =  50 is 0.6863191808701756
AUC ROC Score for c =  100 is 0.6902886908284588
```



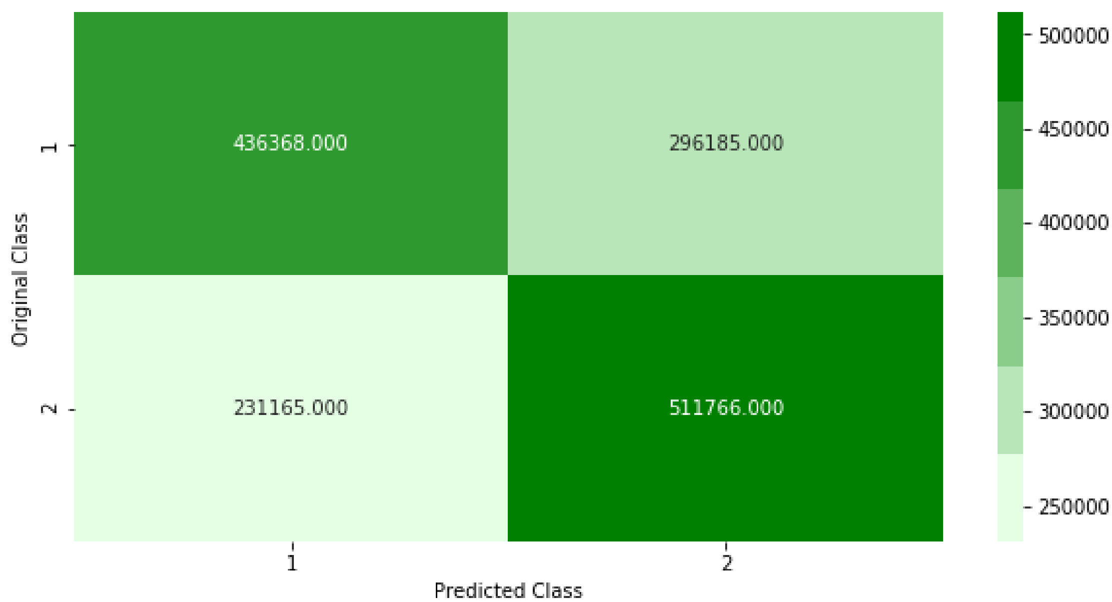
[8]:
```
classifier=AdaBoostClassifier(n_estimators=50,random_state=42)
classifier.fit(x_train, y_train)
y_pred = classifier.predict_proba(x_train)

score = roc_auc_score(y_train, y_pred[:, 1])
print ('Train AUC ROC Score for c = ',50,'is',score)
y_pred = classifier.predict_proba(x_val)
score = roc_auc_score(y_val, y_pred[:, 1])
print ('validation AUC ROC Score for c = ',50,'is',score)
```

```
y_pred = classifier.predict_proba(x_test)
score = roc_auc_score(y_test, y_pred[:, 1])
print ('Test AUC ROC Score for c = ',50,'is',score)
```

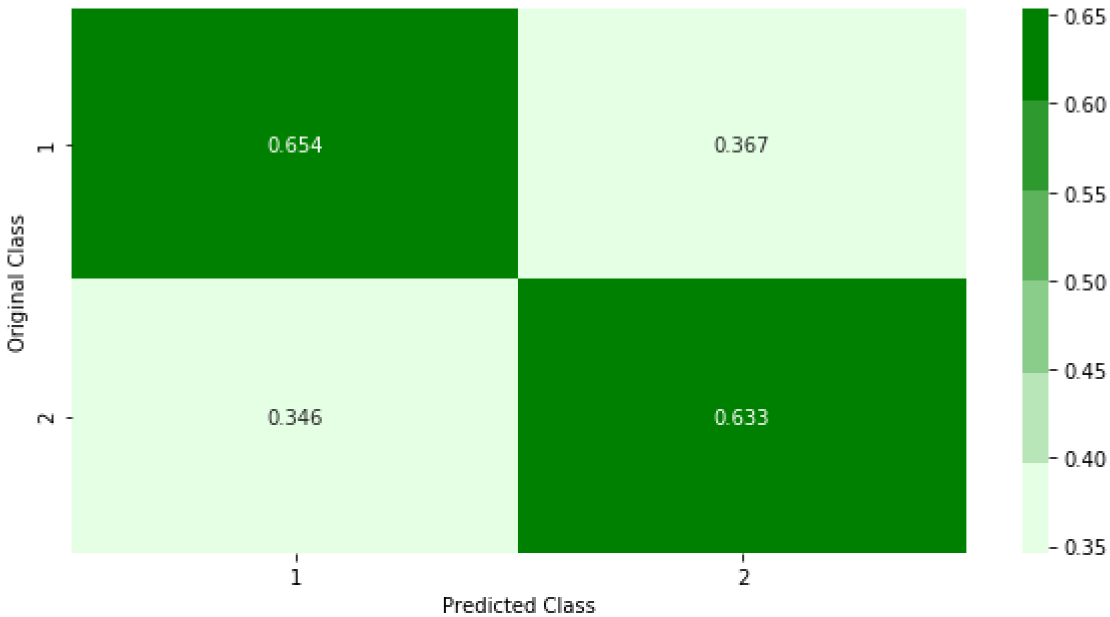
```
Train AUC ROC Score for c =  50 is 0.6865621097439154
validation AUC ROC Score for c =  50 is 0.6863191808701756
Test AUC ROC Score for c =  50 is 0.6863646809616059
```

[18]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
plot_confusion_matrix(y_test, classifier.predict(x_test))
```

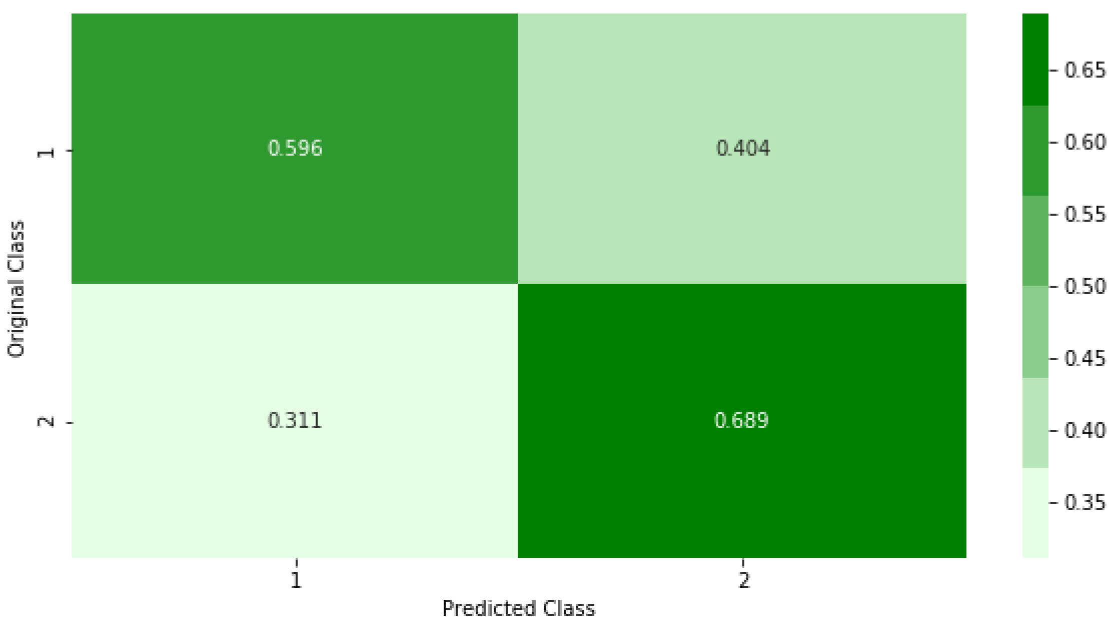
```
Percentage of misclassified points  35.741 %
-------------------------------------------------- Confusion matrix
--------------------------------------------------
```




```
-------------------------------------------------- Precision matrix
--------------------------------------------------
```



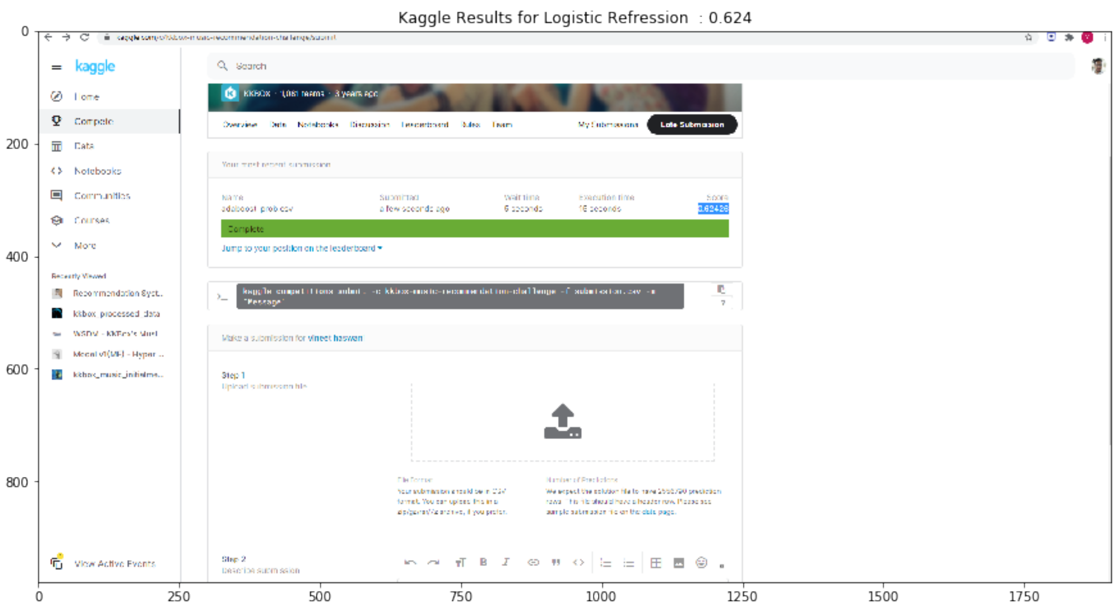
Sum of columns in precision matrix [1. 1.]
--- Recall matrix



Sum of rows in precision matrix [1. 1.]

[4]:
```python
import matplotlib.pyplot as plt
import cv2
plt.figure(figsize = (15, 20))
plt.title('Kaggle Results for Logistic Refression  : 0.624')
plt.imshow(cv2.cvtColor(cv2.imread('Adaboost prob.png'), cv2.COLOR_BGR2RGB))
```

[4]:
```
<matplotlib.image.AxesImage at 0x158a4dd6508>
```

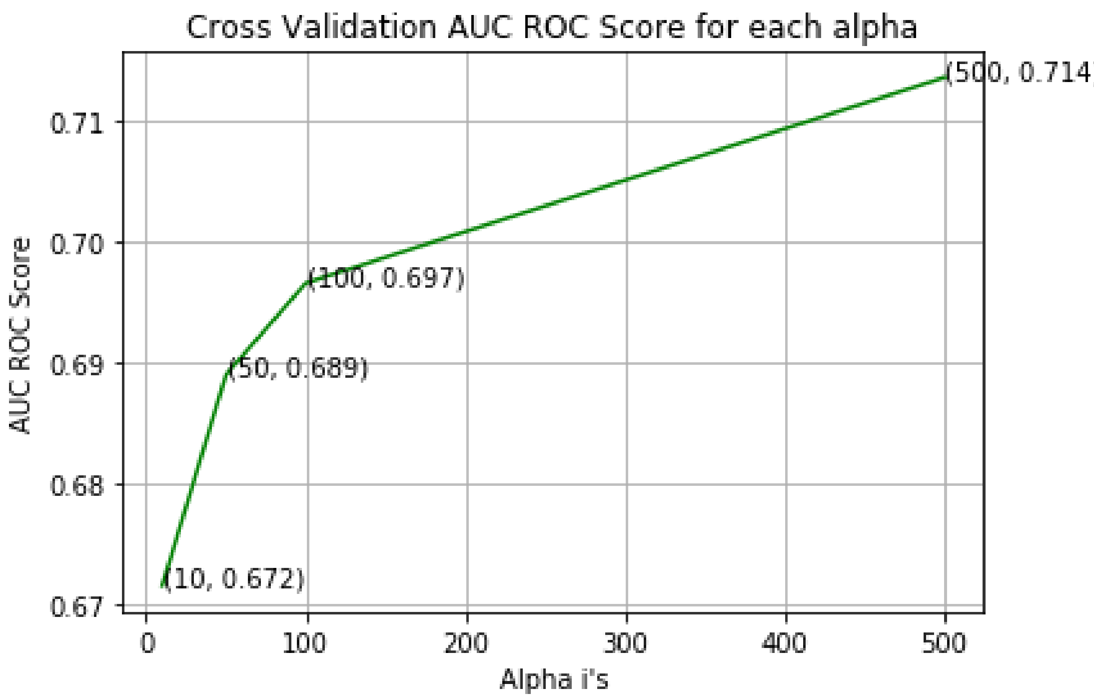


6. XGBoost

[21]:
```python
from xgboost import XGBClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

alpha=[10,50,100,500]
cv_logs=[]
for i in alpha:
    classifier=XGBClassifier(n_estimators=i,nthread=-1)
    classifier.fit(x_train, y_train)
    y_pred = classifier.predict_proba(x_val)
    score = roc_auc_score(y_val, y_pred[:, 1])
    print ('AUC ROC Score for c = ',i,'is',score)
    cv_logs.append(score)

fig, ax = plt.subplots()
ax.plot(alphas, cv_logs,c='g')
```

```
for i, txt in enumerate(np.round(cv_logs,3)):
    ax.annotate((alphas[i],np.round(txt,3)), (alphas[i],cv_logs[i]))
plt.grid()
plt.title("Cross Validation AUC ROC Score for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("AUC ROC Score")
plt.show()
```

```
AUC ROC Score for c =  10 is 0.6715350819922914
AUC ROC Score for c =  50 is 0.6890097897657351
AUC ROC Score for c =  100 is 0.6965871653598837
AUC ROC Score for c =  500 is 0.7135917108237587
```



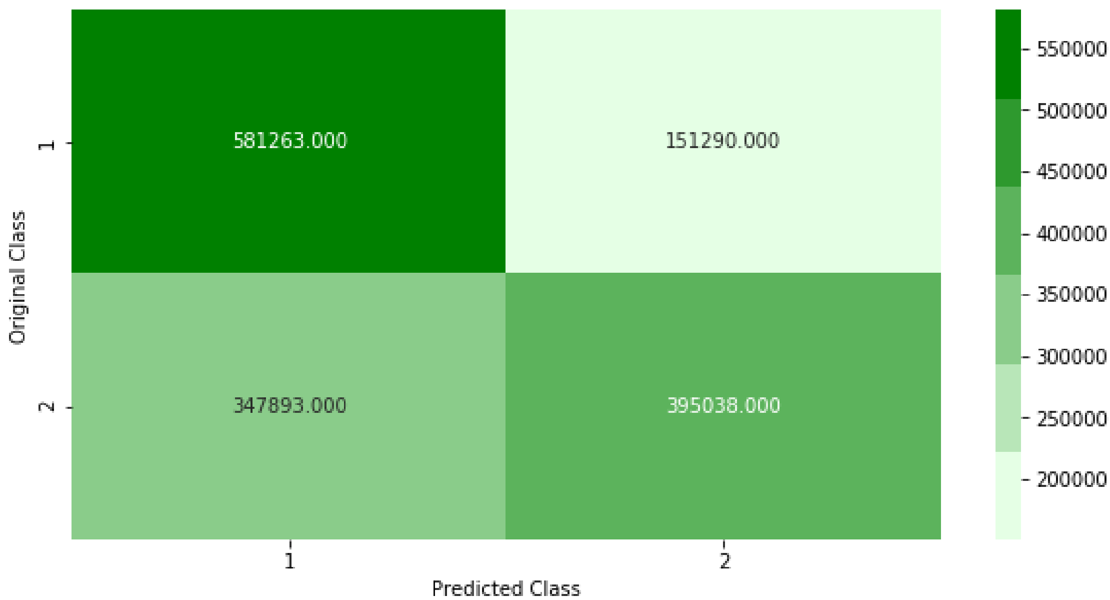
[23]:
```
from xgboost import XGBClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

classifier=XGBClassifier(n_estimators=500,nthread=-1)
classifier.fit(x_train, y_train)

y_pred = sig_clf.predict_proba(x_val)
score = roc_auc_score(y_val, y_pred[:, 1])
print ('validation AUC ROC Score for c = ',0.01,'is',score)
```

```
y_pred = sig_clf.predict_proba(x_test)
score = roc_auc_score(y_test, y_pred[:, 1])
print ('Test AUC ROC Score for c = ',0.01,'is',score)
```

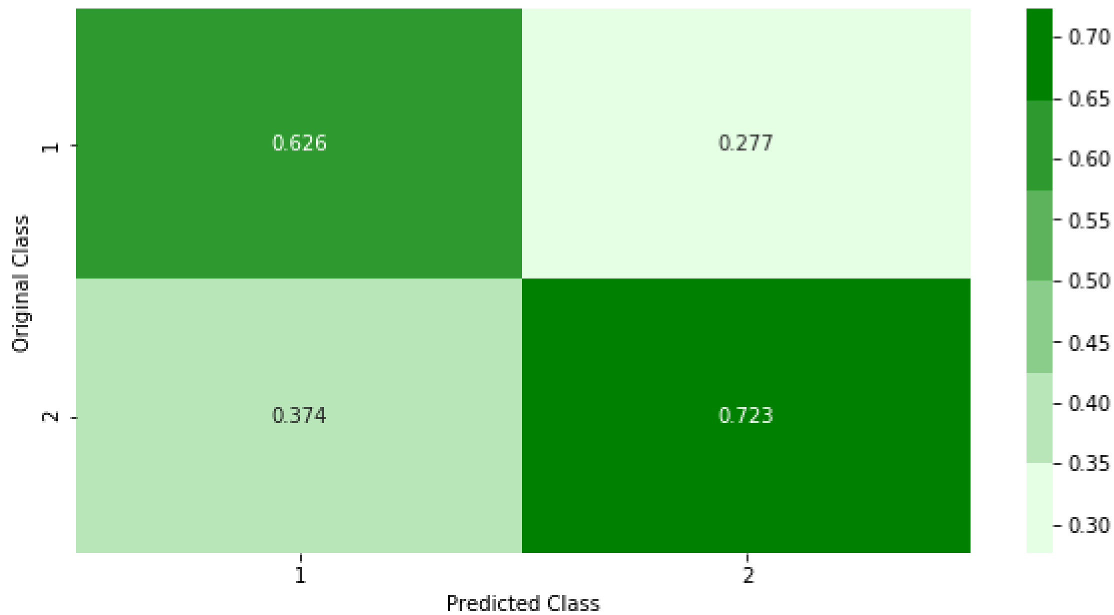
```
validation AUC ROC Score for c =  500 is 0.7135917108237587
Test AUC ROC Score for c =  500 is 0.7123181532419605
```

[11]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
plot_confusion_matrix(y_test, classifier.predict(x_test))
```

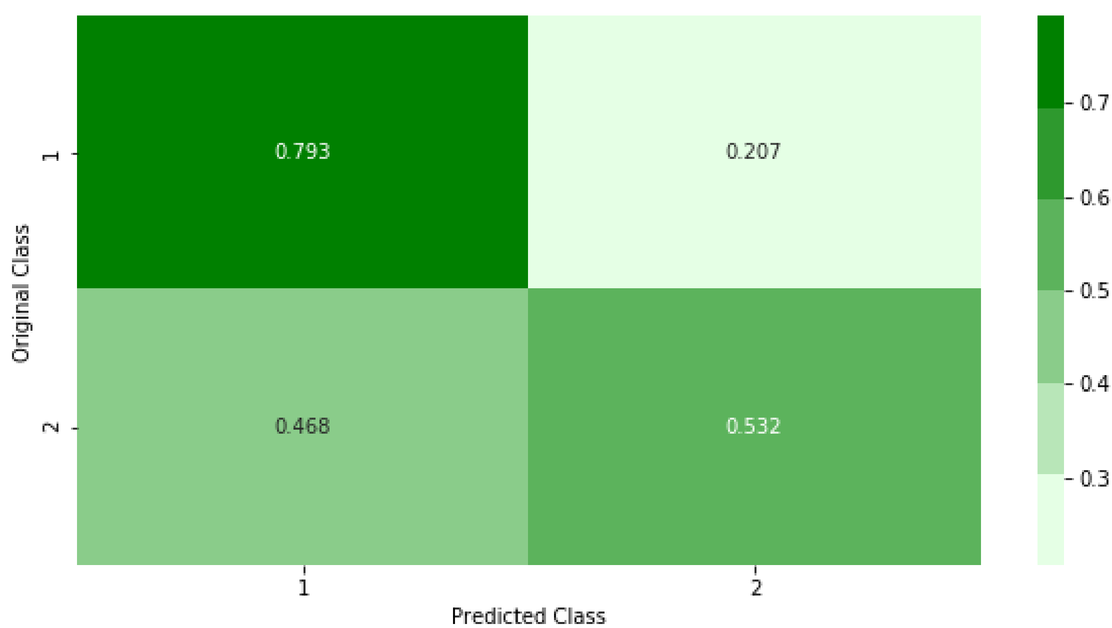
```
Percentage of misclassified points  33.832 %
-------------------------------------------------- Confusion matrix
--------------------------------------------------
```




```
-------------------------------------------------- Precision matrix
--------------------------------------------------
```



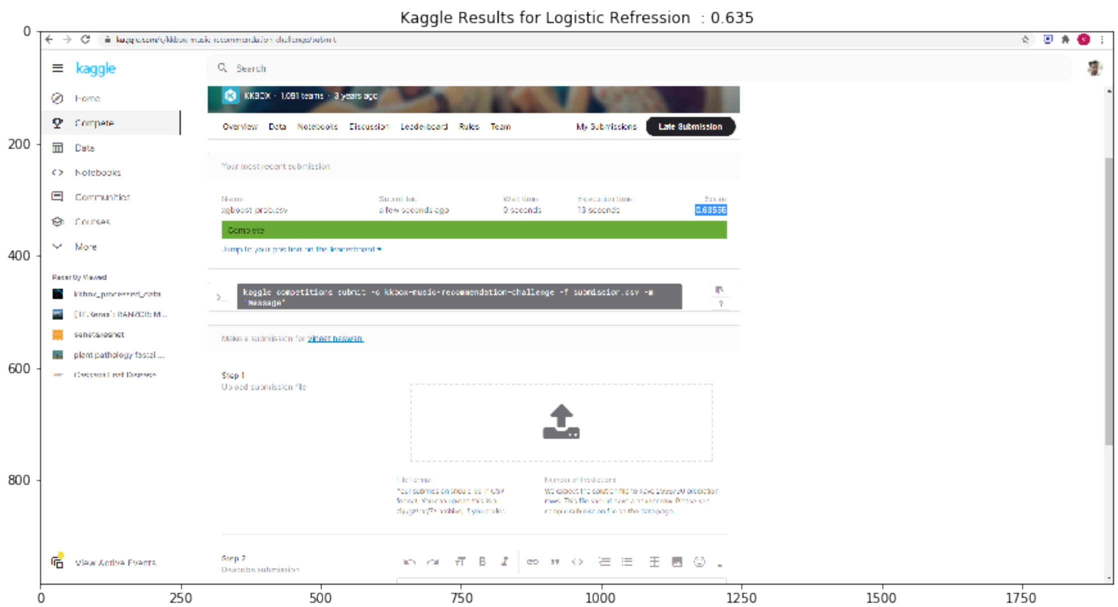

```
Sum of columns in precision matrix [1. 1.]
-------------------------------------------------- Recall matrix
--------------------------------------------------
```




```
Sum of rows in precision matrix [1. 1.]
```

[24]:
```python
import matplotlib.pyplot as plt
import cv2
plt.figure(figsize = (15, 20))
plt.title('Kaggle Results for Logistic Refression  : 0.635')
plt.imshow(cv2.cvtColor(cv2.imread('xgboost_prob.png'), cv2.COLOR_BGR2RGB))
```

[24]: <matplotlib.image.AxesImage at 0x1d75939a7c8>



7. Custom Ensemble Model

[ ]:
```python
import numpy as np
def generating_samples(input_data, target_data):
    selected_rows = np.sort(np.random.choice(input_data.shape[0], int(input_data.shape[0]*0.6), replace = True))

    sampled_input_data = input_data.iloc[selected_rows, :]
    sampled_target_data = target_data.iloc[selected_rows, :]

    return sampled_input_data , sampled_target_data, selected_rows
```

[1]:
```python
from scipy.stats import mode
from tqdm import tqdm
from sklearn.metrics import roc_auc_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

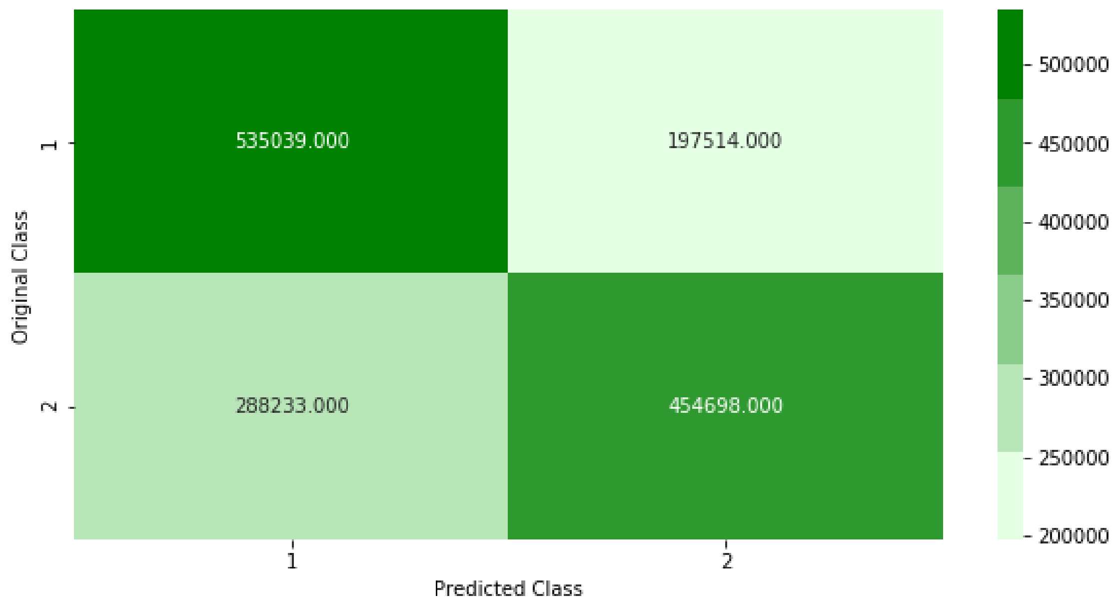
```python
def custom_model(x_train, y_train, x_test, y_test, n_estimators = 10, alpha =␣
↪100, max_depth = 40):
    # x_train, x_test, y_train, y_test = train_test_split(x_train, y_train,␣
↪test_size = 0.5, random_state = 0, stratify = y_train)
    d1_x_train, d2_x_train, d1_y_train, d2_y_train = train_test_split(x_train,␣
↪y_train, test_size = 0.5, random_state = 0, stratify = y_train)
    predictions = []
    base_models = []
    for i in tqdm(range(n_estimators)):
        ## generating samples
        x, y, rows = generating_samples(d1_x_train, d1_y_train)
        ## training base model
        base_model = DecisionTreeClassifier(max_depth=max_depth,random_state=42)
        base_model.fit(x, y)
        pred = base_model.predict_proba(d2_x_train)[:, 1].reshape(-1, 1)
        predictions.append(pred)
        base_models.append(base_model)
    predictions = np.array(predictions).T
    predictions = predictions.reshape(-1, n_estimators)

    ## training meta model
    meta_model =␣
↪LogisticRegression(penalty='l2',C=alpha,class_weight='balanced')
    meta_model.fit(predictions, d2_y_train)
    y_pred = meta_model.predict_proba(predictions)

    score = roc_auc_score(d2_y_train, y_pred[:, 1])
    print('AUC Score of Model on train set is :', score )

    ### Calculate AUC ROC Score on test set
    predictions = []
    for base_model in base_models:
        pred = base_model.predict_proba(x_test)[:, 1].reshape(-1, 1)
        predictions.append(pred)
    predictions = np.array(predictions).T
    predictions = predictions.reshape(-1, n_estimators)
    y_pred = meta_model.predict_proba(predictions)
    score = roc_auc_score(y_test, y_pred[:, 1])
    print('AUC Score of Model on test set is :', score )
    return base_models, meta_model, y_pred

base_models, meta_model, y_pred = custom_model(x_train, y_train, x_test,␣
↪y_test, n_estimators = 100, alpha = 100, max_depth = 10)
```

```
AUC Score of Model on train set is : 0.6959068940143411
AUC Score of Model on test set is : 0.6961223438481663
```

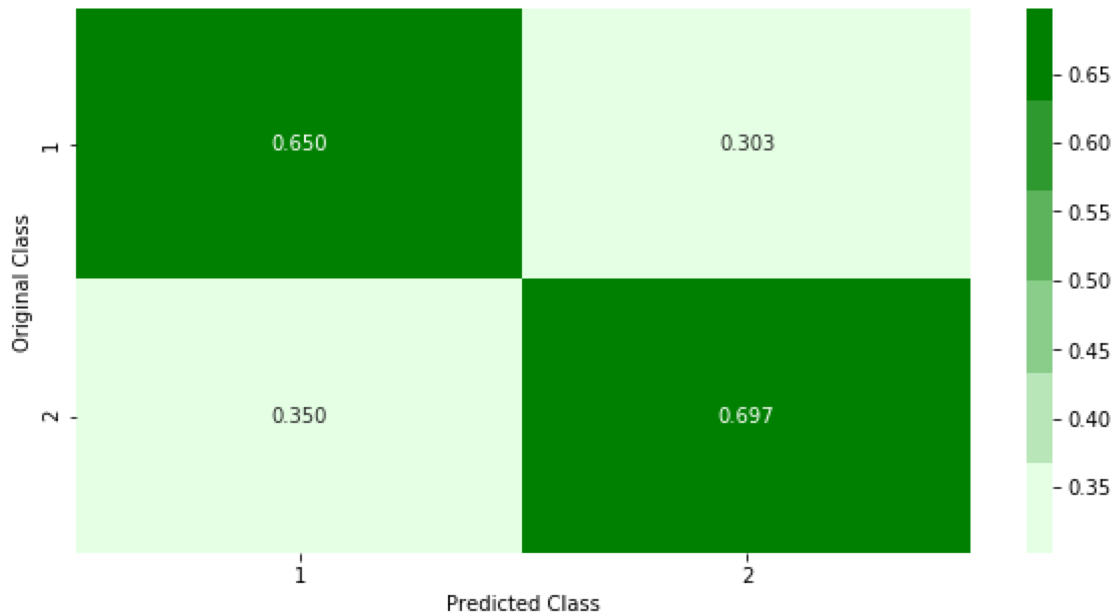
[11]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
plot_confusion_matrix(y_test, y_pred[:, 1] >0.5)
```

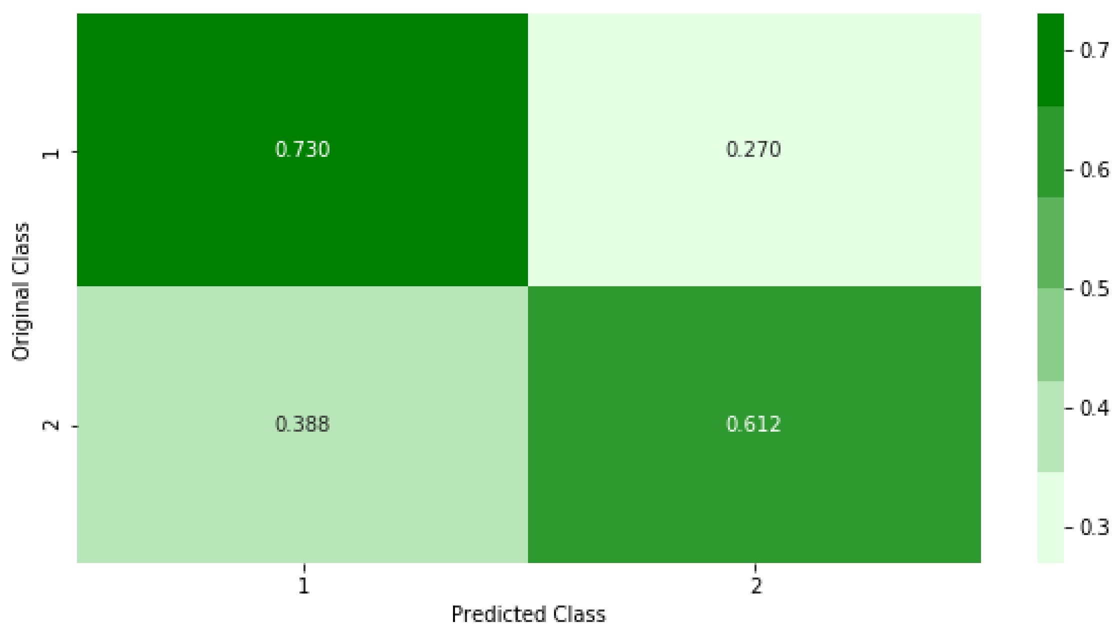
```
Percentage of misclassified points  32.921 %
-------------------------------------------------- Confusion matrix
--------------------------------------------------
```




```
-------------------------------------------------- Precision matrix
--------------------------------------------------
```



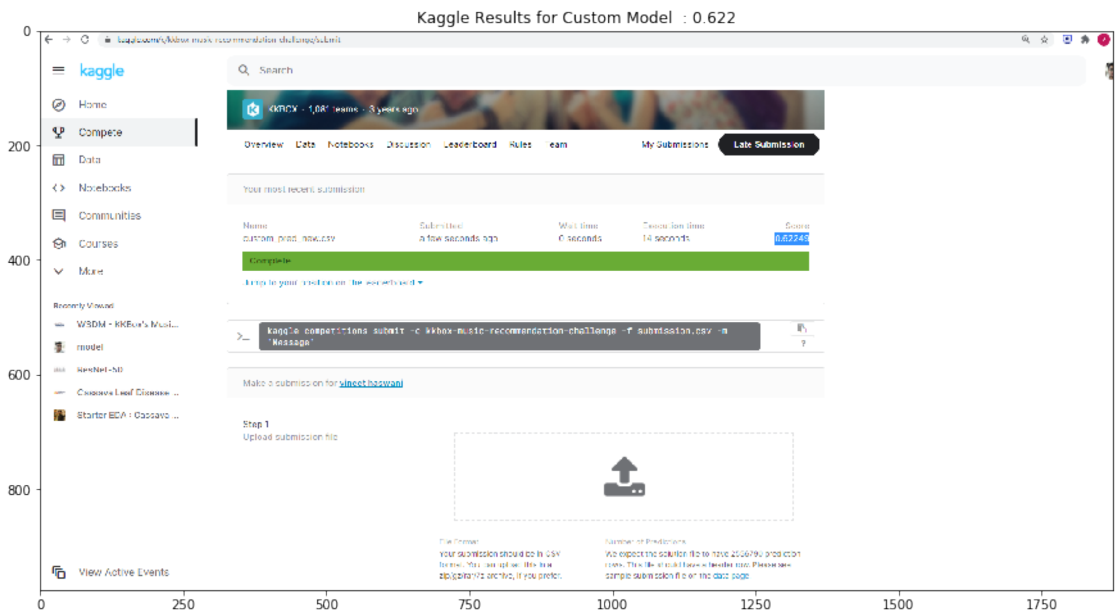
Sum of columns in precision matrix [1. 1.]
-- Recall matrix



Sum of rows in precision matrix [1. 1.]

```
[5]: import matplotlib.pyplot as plt
     import cv2
     plt.figure(figsize = (15, 20))
     plt.title('Kaggle Results for Custom Model  : 0.622')
     plt.imshow(cv2.cvtColor(cv2.imread('custom_prob.png'), cv2.COLOR_BGR2RGB))
```

```
[5]: <matplotlib.image.AxesImage at 0x158a4d9ad08>
```



5 Conclusion for Modeling :

```
[6]: from prettytable import PrettyTable

     myTable = PrettyTable(["Model Name", "HyperParameter", "AUC Score on Test"])
     myTable.add_row(["Logistic Regression", "alpha : 0.01", "66.50 %"])
     myTable.add_row(["SGDClassifier with log loss", "alpha : 0.01", "68.46 %"])
     myTable.add_row(["Decision Tree Classifier", "max_depth :20", "69.24 %"])
     myTable.add_row(["Random Forest Classifier", "n_estimators : 100", "78.56 %"])
     myTable.add_row(["AdaBoost Regression", "n_estimators : 50", "68.63 %"])
     myTable.add_row(["XGBoost Classifier", "n_estimators : 500", "71.23 %"])
     myTable.add_row(["Custom Model", "alpha :100 max_depth :40", "69.61 %"])

     print(myTable)
```

```
+------------------------------+--------------------------+-------------------+
|          Model Name          |      HyperParameter      | AUC Score on Test |
+------------------------------+--------------------------+-------------------+
|     Logistic Regression      |       alpha : 0.01       |      66.50 %      |
```

SGDClassifier with log loss	alpha : 0.01	68.46 %
Decision Tree Classifier	max_depth :20	69.24 %
Random Forest Classifier	n_estimators : 100	78.56 %
AdaBoost Regression	n_estimators : 50	68.63 %
XGBoost Classifier	n_estimators : 500	71.23 %
Custom Model	alpha :100 max_depth :40	69.61 %

Best Model is Random Forest classifier with number of trees as 100 with AUC Score of 78.56%. Further it can be improved by applying Calibration on top of classifier but it is very expensive to apply need more than 35 GB of ram for computation