```
import os
import warnings
warnings.simplefilter(action = 'ignore', category=FutureWarning)
warnings.filterwarnings('ignore')
def ignore_warn(*args, **kwargs):
    pass

warnings.warn = ignore_warn #ignore annoying warning (from sklearn and seaborn)

import pandas as pd
import numpy as np
import time
import matplotlib.pyplot as plt; plt.rcdefaults()
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from scipy.sparse import coo_matrix
import math as mt
from scipy.sparse.linalg import * #used for matrix multiplication
from scipy.sparse.linalg import svds
from scipy.sparse import csc_matrix
from scipy.stats import skew, norm, probplot
import seaborn as sns
sns.set(style="ticks", color_codes=True, font_scale=1.5)
color = sns.color_palette()
```

```
sns.set_style('darkgrid')
#Class for Popularity based Recommender System model
class popularity_recommender_py():
    def __init__(self):
        self.train_data = None
        self.user_id = None
        self.item_id = None
        self.popularity_recommendations = None

    #Create the popularity based recommender system model
    def create(self, train_data, user_id, item_id):
        self.train_data = train_data
        self.user_id = user_id
        self.item_id = item_id

        #Get a count of user_ids for each unique song as recommendation score
        train_data_grouped =
train_data.groupby([self.item_id]).agg({self.user_id: 'count'}).reset_index()
        train_data_grouped.rename(columns = {'user_id': 'score'},inplace=True)

        #Sort the songs based upon recommendation score
        train_data_sort = train_data_grouped.sort_values(['score', self.item_id],
ascending = [0,1])

        #Generate a recommendation rank based upon score
        train_data_sort['Rank'] = train_data_sort['score'].rank(ascending=0,
method='first')
```

```
        #Get the top 10 recommendations
        self.popularity_recommendations = train_data_sort.head(10)

    #Use the popularity based recommender system model to
    #make recommendations
    def recommend(self, user_id):
        user_recommendations = self.popularity_recommendations

        #Add user_id column for which the recommendations are being generated
        user_recommendations['user_id'] = user_id

        #Bring user_id column to the front
        cols = user_recommendations.columns.tolist()
        cols = cols[-1:] + cols[:-1]
        user_recommendations = user_recommendations[cols]

        return user_recommendations

track_metadata_df = pd.read_csv('../input/song_data.csv')
count_play_df = pd.read_csv('../input/10000.txt', sep='\t', header=None,
names=['user','song','play_count'])

print('First see of track metadata:')
print('Number of rows:', track_metadata_df.shape[0])
print('Number of unique songs:', len(track_metadata_df.song_id.unique()))
display(track_metadata_df.head())
print('Note the problem with repeated track metadata. Let\'s see of counts play
song by users:')
display(count_play_df.shape, count_play_df.head())

unique_track_metadata_df =
track_metadata_df.groupby('song_id').max().reset_index()

print('Number of rows after unique song Id treatment:',
unique_track_metadata_df.shape[0])
print('Number of unique songs:', len(unique_track_metadata_df.song_id.unique()))
display(unique_track_metadata_df.head())

user_song_list_count = pd.merge(count_play_df,
                                unique_track_metadata_df, how='left',
                                left_on='song',
                                right_on='song_id')
user_song_list_count.rename(columns={'play_count':'listen_count'},inplace=True)
del(user_song_list_count['song_id'])

display(user_song_list_count.head())
user_song_list_count.listen_count.describe().reset_index().T

print('{:d} users, {:.2%} of total play counts, listening a single more than 200
times'.format(
    count_play_df.user[count_play_df.play_count>200].unique().shape[0],
```

```
count_play_df.play_count[count_play_df.play_count>200].count()/count_play_df.shap
e[0]))
display(count_play_df.play_count[count_play_df.play_count>200].describe().reset_i
ndex().T)

def create_popularity_recommendation(train_data, user_id, item_id, n=10):
    #Get a count of user_ids for each unique song as recommendation score
    train_data_grouped = train_data.groupby([item_id]).agg({user_id:
'count'}).reset_index()
    train_data_grouped.rename(columns = {user_id: 'score'},inplace=True)

    #Sort the songs based upon recommendation score
    train_data_sort = train_data_grouped.sort_values(['score', item_id],
ascending = [0,1])

    #Generate a recommendation rank based upon score
    train_data_sort['Rank'] = train_data_sort.score.rank(ascending=0,
method='first')

    #Get the top n recommendations
    popularity_recommendations = train_data_sort.head(n)
    return popularity_recommendations

recommendations =
create_popularity_recommendation(user_song_list_count,'user','title', 15)
display(recommendations)

display(create_popularity_recommendation(user_song_list_count,'user','artist_name
', 10))
```