

Music Recommendation System Python Project with Source Code

Why use Python for this project?

Over the years, Python has earned a superstar status in data science. All data fans adore it, and it offers a simple introduction to statistical science and machine learning. There are several built-in libraries for challenging data science projects, which are simple to create. Python's popularity is also due to its straightforward code readability. When compared to other complex languages, its syntax is skeletal and minimal. The list of libraries available for usage in Python for data science is incomplete, but it includes seaborn, Matlab, sci-kit learn, SciPy, pandas, requests, regex, NumPy, etc.

- A python is a good option for beginners to start learning data science, and rightly so.
- We all utilize audio streaming services, including Gaana, Jio Saavn, Spotify, and iTunes.
- Do you ponder how these platforms propose songs to you based on your preferences when listening to music on them? These services use machine learning algorithms.
- This is so that they can give you the songs they believe you'll like. We'll discuss these models in this article and use them to create a music recommendation engine.

To assist you in beginning your educational path and gaining the practical experience required for a data science career, here is a selection of vetted Python data science projects. It's time to play around with Data Science Projects and experiment with different strategies for solving a business problem to gain data-driven insights. By offering visual and straightforward methods to organize and clean your data, software like Tableau Prep may assist you in fostering a great data culture.

Tableau Prep features two products: Mondrian Prep Director for scheduling, managing, and monitoring flows throughout your company and Tableau Prep Builder for creating data flows. A data cleaning tool can let analysts or administrators begin their studies more quickly and with greater confidence in the data, saving an administrator a lot of time. Making successful and efficient business decisions requires understanding data quality and the tools taken to construct, manage, and transform data.

This vital procedure will help your firm further establish a data culture. You can learn how design firm Tinuiti grew their advanced analytics for 500+ clients and centralized 100plus data sets in Tableau Prep by reading about how they did it.

Music Recommendation System on KKBox Dataset

Today's world is surrounded by music. By 2021, more than 70 million songs will be available on Spotify alone, proving how accessible music is. Other services include KKBox, Gaana, Saavn, and Apple Music. How can fresh content become found in a field with so much content currently available? The Recommendations system is how users discover new songs and develop new musical likes. Recommendation algorithms are profitable for music streaming businesses as well. They benefit from increased engagement and audience growth on their platform.

In Asia, it is one of the largest streaming services is KKBox.

Dataset Description

The information includes music and user metadata. User- and song-specific information, such as the user ID'S, user Registration date, song ID'S, song genre, song ArtistName, and song release date, is included in the metadata. The information includes when a user plays a piece of music only once. For each song-user combination, this information is in particular.

The dataset consists of three files:

- csv - This file contains information on user-song pairs, including user IDs, source system tab, source type, source screentime, and target.
- The target indicates if a user tuned to the same music within one month.
- csv - It includes information about song IDs, song genre, song artist, song lyricist, etc. Target = 1 indicates that the user played the song within 30 days; Target = 0 signifies that the user did not play the song.
- individ'suals .csv - It contains client account information like user_name, user_age, user_gender, user_subscription_plan, and so forth.

Remove duplicate or pointless observations as well as undesirable observations from your dataset. The majority of duplicate observations will occur during data gathering. Duplicate data can be produced when you merge data sets from several sources, scrape data, or get from clients or other departments. One of the most important factors to consider in this procedure is de-duplication. Those observations are deemed irrelevant when you observe data that do not pertain to the particular issue you are attempting to study.

For instance, you might eliminate those useless observations if you wish to examine data about millennial clients, but your dataset also includes observations from earlier generations. In addition to providing a more understandable and effective dataset, this can increase analysis efficiency and reduce divergence from your main aim.

Step 1: Data Cleaning

Anomalies, outliers, and missing values may be present in the dataset. These situations may affect how accurately and efficiently algorithms are implemented. A dataset contains 30-50% outliers or missing values. The data needs to be uniformly normalized throughout.

Data cleansing is removing information from your dataset that does not belong there. The methods used to sanitize that data are the main topic of this essay. The process of changing data from one place or architecture to another is known as data transformation. Transformation activities are also known as data wrangling or data munging

when translating data from one "hard" data form into another format for archiving and analysis.

We employ the following methods to clean the data: 1. Outlier Recognition and Treatment.

Outliers are ludicrous values and are outside the acceptable range for a label. For instance, a user's age below zero and above one hundred can be deemed ludicrous. It might be stricter in specific circumstances, such as when buying alcohol between 18 and 100.

Benefits of data cleaning

- When you have clean data, you can make decisions using the highest-quality information and eventually boost productivity. Benefits comprise:
- Removal of inaccuracies when several data sources are involved.
- Clients are happier, and employees are less annoyed when there are fewer mistakes.
- The capacity to map out your data's many functions and planned uses.
- Monitoring mistakes and improving reporting make resolving incorrect or damaged data easier for future operations by allowing users to identify where issues are coming from.

Step 2: Addition of Missing Values

Imputing entails substituting a different value for any lacking values in the dataset.

We divide user-song pairings into two distinct categories: repetitions and non-repeats.

- Replacing empty values in the dataset with the proper data - The median or the average of the values is used to fill in the gaps left by the missing values.
- Eliminating all null values - In this situation, all the pieces of information with missing data are eliminated, which causes data loss. Following this process, the dataset file's size decreases.
- Creating a new Missing label - Data points with missing values are placed in a new section labeled "missing." It categorizes the lacking resources into a single group.
- Finally, change the numerical counterparts of string labels.

Step 3: Correct structural issues

When you test or exchange data and find odd naming practices, typos, or wrong capitalization, such are structural faults. Mislabeled sections or classes may result from these inconsistencies. For instance, you might see both "N/A" and "Not Applicable," but they must be assessed as belonging to the same category.

Step 4: Eliminate unwelcome outliers

There will frequently be isolated findings that, at first look, seem to need to fit the data you are evaluating. Removing an outlier if you have a good reason to, such as incorrect data entry, will enhance the accuracy of the data you are currently working with. However, occasionally an outlier's appearance will support a theory you're working on. Remember that an outlier doesn't necessarily indicate that something needs to be fixed. To determine the reliability of the number, this step is necessary. If an outlier turns out to be incorrect or unimportant for the analysis, you should remove it.

Step 5: Data standardization

Data standardization allows you to recognize and transform data from many formats into standardized one. Data standardization can be useful if you don't have data restrictions at data entry or if company data have conflicting forms. In contrast to data validation, standardization procedures can be used on data that has already been gathered. This entails creating scripts to transform your soiled data into reliable ones.

Matching strings: Use rigorous or fuzzy string-matching techniques to find exact and close matches with your data and legitimate values to standardize inconsistent data. By comparing your data characters to the expected valid values, you can eliminate or modify the strings that don't match.

Libraries

NumPy, Sklearn, and Pandas

To develop a music recommendation system, the following 4 modelling strategies will be assessed in this project:

- Regression with logit
 - Of all the algorithms, regression is the most straightforward. It can be found in the Sklearn package of Python as little more than a linear model.
- Determination Tree
 - The decision tree uses the tree structure to arrive at decisions or outcomes. Each level offers the option of choosing one of the two branches. The result is output by the tree after each iteration.
- Forest Random

Decision Trees are gathered into a Random Forest.

Recommended models

As I've already mentioned, these music streaming sites use ML models to offer the songs you enjoy. These models are referred to as classes in the Recommendation Python package. We must import Pandas and Numpy libraries into this package:

```
import numpy as npn
```

```
import pandas
```

Let's now talk about the models that are applied to recommendations:

Recommendation for Popularity:

This algorithm suggests music to you that is trending or popular in your area. This model is based on the songs well-liked in your area or constantly played by system users.

The source code of popularity recommendation

```
class popularity_recommender():
    def __init__(s):
        s.t_data = None
        s.u_id's = None
    #ID'S of users
        s.i_id's = None
    #ID'S of Song users are listening to
        s.pop_recommendations = None
    #getting all recommendations according to the popularity
    #Creating the system models
        def create_p(s, t_data, u_id's, i_id's):
            s.t_data = t_data
            s.u_id's = u_id's
            s.i_id's = i_id's
    # Get a recommendation score based on the number of times each music has been listened to.
        t_data_grouped = t_data.groupby([s.i_id's]).agg({s.u_id's: 'count'}).reset_index()
        t_data_grouped.rename(columns = {'user_id's': 'score'},inplace=True)

    #Sort the songs as per user ratings.
        t_data_sort = t_data_grouped.sort_values(['score', s.i_id's], ascending = [0,1])

    #Create a ranking for recommendations based on score
        t_data_sort['Rank'] = t_data_sort['score'].rank(ascending=0, method='first')

    # top 10 recommendations are here
```

```

s.pop_recommendations = t_data_sort.head(10)
#To give recommendations using the system model
def fits(dfs, algo, flag=0):
    if flag:
        algo.fits(df)
    else:
        algo.partial_fit(df)
    dfs['label'] = algo.labels_
    return (dfs, algo)
In [ ]:
def predict(t, Y):
    y_preds = t[1].predict(Y)
    modes = pd.Series(y_preds).mode()
    return t[0][t[0]['label'] == mode.loc[0]]
In [ ]:
def recommend(recommendations, meta, Y):
    dats = []
    for i in Y['track_id']:
        dats.append(i)
    genre_modes = meta.loc[dats]['genres'].mode()
    artist_modes = meta.loc[dats]['artist_name'].mode()
    return metas[meta['genre'] == genre_mode.iloc[0]], meta[meta['artist_name'] == artist_modes.iloc[0]], meta.loc[recomm
In [ ]:
t = fit(X, kmeans, 1)
In [ ]:
recommendations = predict(t, Y)
In [ ]:
output = recommend(recommendations, m)
def recommend_p(s, u_id's):
    u1_recommendations = s.pop_recommendations

    # Add the column for the user id where the music recommendations are generated.
    u1_recommendations['user_id's'] = u_id's

    #Bringing user_id's column to the upper front
    cols = u1_recommendations.columns.tolist()

```

```
cols = cols[-1:] + cols[:-1]
u1_recommendations = u1_recommendations[cols]

return u1_recommendations
```

Similarity Recommendation:

Your daily music listening habits are taken into account by this model.

For instance: Let's say you use Spotify to listen to Linkin Park's song Numb. After watching the songs, you may be given music recommendations such as Boulevard of Shattered Dreams by Green Day or At the End by Linkin Park because these songs have the same artist or genre.

Source code:

```
#python Class for Items similar in nature, Recommender System model
```

```
class similarity_recommender1():
```

```
    def __init__(s):
```

```
        s.t_data = None
```

```
        s.u_id's = None
```

```
        s.i_id's = None
```

```
        s.co_matrix = None
```

```
        s.songs_dic = None
```

```
        s.rev_songs_dic = None
```

```
        s.i_similarity_recommendations11 = None
```

```
    def get_u_items(s, u):
```

```
        u_data = s.t_data[s.t_data[s.u_id's] == u]
```

```
        u_items = list(u_data[s.i_id's].unique())
```

```
        return u_items
```

```
    def get_i_users(s, i):
```

```
        i_data11 = s.t_data[s.t_data[s.i_id's] == i]
```

```
        i_users = set(i_data[s.u_id's].unique())
```

```
        return i_users
```

```
#Get unique songs in the training data
```

```
def get_all_items1_t_data(s):
```

```
    all_items1 = list(s.t_data[s.i_id's].unique())
```



```

    return all_items1
#Construct a co-occurrence matrix
def construct_co_matrix(s, u_songs, a_songs):
    #Get users for all songs in user_songs11.
    u_songs_users = []
    for i in range(0, len(u_songs)):
        u_songs_users.append(s.get_i_users(u_songs[i]))

    #Initializing the item co-occurrence matrix of size len(user_songs11) X len(songs)
    co_matrix = npn.matrix(npn.zeros(shape=(len(u_songs), len(a_songs))), float)

    # Determine how comparable the songs the user has been listening to are to all the other songs in the training set.
    for i in range(0, len(a_songs)):
        #Calculating unique listeners of songs.
        songs_i_data11 = s.t_data[s.t_data[s.i_id's] == a_songs[i]]
        users_i = set(songs_i_data[s.u_id's].unique())

        for j in range(0, len(u_songs)):

            #Getting unique users means listeners of songs (item) j
            users_j = u_songs_users[j]

            # Calculate the songs in common listened to by listeners i & j
            users_intersection11 = users_i.intersection(users_j)

            #Calculate co-occurrence_matrix[i,j] as Jaccard Index
            if len(users_intersection) != 0:
                #Calculate all the songs listened by i & j
                users_union = users_i.union(users_j)

                co_matrix[j,i] = float(len(users_intersection))/float(len(users_union))
            else:
                co_matrix[j,i] = 0

    return co_matrix

```

#Use the co-occurrence matrix to make top recommendations

def generate_top_r(s, user, co-occurrence_matrix, a_songs, u_songs):

print("Non zero values in co-occurrence_matrix :%d" % npn.count_nonzero(co-occurrence_matrix))

#Calculate the average of the scores in the co-occurrence matrix for all songs listened to by the user.

user_sim_scores1 = co-occurrence_matrix.sum(axis=0)/float(co-occurrence_matrix.shape[0])

user_sim_scores1 = npn.array(user_sim_scores)[0].tolist()

#Sort the indices of user_sim_scores1 based on their value and also maintain the corresponding score

s_index = sorted(((e,i) **for** i,e **in** enumerate(list(user_sim_scores))), reverse=True)

#Create a dataframe from the following

columns = ['user_id's', 'songs', 'score', 'rank']

#index = npn.arange(1) # array of numbers for the number of samples

df1 = pandas.DataFrame(columns=columns)

#Filling the dataframe with the top 20 songs

rank = 1

for i **in** range(0,len(s_index)):

if ~npn.isnan(s_index[i][0]) **and** a_songs[s_index[i][1]] **not in** u_songs **and** rank <= 10:

df1.loc[len(df1)]=[user,a_songs[s_index[i][1]],s_index[i][0],rank]

rank = rank+1

#Handling the case where no recommendation

if df1.shape[0] == 0:

print("There are no songs available for the current user's similarity-based recommendation algorithm.")

return -1

else:

return df1

#Create the system model

def create_s(s, t_data, u_id's, i_id's):

s.t_data = t_data

s.u_id's = u_id's

s.i_id's = i_id's

#Use the model to make recommendations

```

def recommend_s(s, u):

    #A. Getting all unique songs for this user
    u_songs = s.getting_u_items(u)

    print("No. of songs for the user: %d" % len(u_songs))

    # Getting all the songs in the data
    a_songs1 = s.getting_all_items1_t_data()

    print("No. of songs in the list: %d" % len(a_songs))

    #C. Make the co-occurrence matrix of size len(user_songs11) X len(songs)
    co_matrix = s.construct_co_matrix(u_songs, a_songs)
    df_r = s.generate_top_r(u, co_matrix, a_songs, u_songs)
    return df_

def similar_items(s, i_list):

    u_songs = i_list
    a_songs1 = s.getting_all_items1_t_data()

    print("no. of unique songs in the set: %d" % len(a_songs))

    # Make the co-occurrence matrices of size len(user_songs11) X len(songs)
    co_matrix = s.construct_co_matrix(u_songs, a_songs)
    #C. Use the matrix to make recommendations
    u = ""
    df_r = s.generate_top_r(u, co_matrix, a_songs, u_songs)
    return df_r

```

We now integrate them into a new file by using Python libraries necessary for this task and the Guid'seline package:

Source code:

```

import pandas
from sklearn.model_selection import train_test_split

```

```
import numpy as npn
import time
import Recommenders as Recommenders
```

After that, we'll open an a.csv file with the data and obtain the number of occasions a user has listened to each song in line five.

Source code:

```
#Read user_id's, songs_id's, listen_count
# The process of downloading data from outside sources could take some time.
triplets = 'https://static.turi.com/datasets/millionsongs/10000.txt'
songs_metadata = 'https://static.turi.com/datasets/millionsongs/songs_data.csv'

songs_df_a = pandas.read_table(triplets,header=None)
songs_df_a.columns = ['user_id's', 'songs_id's', 'listen_count']

#Read songs metadata
songs_df_b = pandas.read_csv(songs_metadata)

# Combine the columns for song titles and artists to create a new column.
songs_df1 = pandas.merge(songs_df_a, songs_df_b.drop_duplicates(['songs_id's']), on="songs_id's", how="left")
songs_df1.head()
```

Output:

[*]:

	usrz_id	song_id	listen_count	title	release	artist_name	year
0	b08344d063b5ccb3212f76538f3d9e43d87dca9e	SO8YHA12A67018F1D	1	The Sea	Wolfgang Amadeus	Wolfgang Amadeus	0
1	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SO8BMDR12A8C13253B	2	Entre Dos Aguas	Flamenco Para Niños	Paco De Lucia	1976
2	b08344d063b5ccb3212f76538f3d9e43d87dca9e	SO8YHA12A67018F1D	1	Stanger	Clayton Kopp	Clayton Kopp	2007
3	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SO8YHA12A67018F1D	1	Constellations	In Between Dreams	Jack Johnson	2005
4	b08344d063b5ccb3212f76538f3d9e43d87dca9e	SO8YHA12A67018F1D	1	Learn To Fly	There Is Nothing Left To Learn	Free Flight	1999

For better understanding, we will now display the number of songs, or the number of rows, in the dataset in the file.

Source code:

```
print("Total no of songs:",len(songs_df1))
```

Output:

```
Total no of songs: 2000000
```

Next, we'll build a dataframe from a portion of the provided dataset.

Source code:

```
songs_df1 = songs_df1.head(10000)

# Combine the columns for song titles and artists to create a new column.
songs_df1['songs'] = songs_df1['title'].map(str) + " - " + songs_df1['artist_name']

The column listen_count denotes the no of times the songs have been listened to. Using this column, we'll find the datafram
songs_gr = songs_df1.groupby(['songs']).agg({'listen_count': 'count'}).reset_index()
grouped_sum = songs_gr['listen_count'].sum()
songs_gr['percentage'] = songs_gr['listen_count'].div(grouped_sum)*100
songs_gr.sort_values(['listen_count', 'songs'], ascending = [0,1])
```

Because the output is too long enough to display in full, I've excerpted it below.

Output:

```
Anyway - Armand Van Helden & A-TRAK Present Du.
5139 high fives - Four Tets
5142 in white rooms - Booka Shades
5132 paranoid's androids - Christophers O'Riley
5192 ¿LoVes? [Piano ¿LoVes? [Piano Y Voz] - Alehjandro Sanz
51502 Época - Gotan Project
512 rows × 3 columns
518 Your Love - The Outfield
7121 Your Mouth - Telefon Tel Aviv
Ze Rook Naar Rozen - Rob De Nijs
7131 Zebra - Beach Houses
7132 Zebra - Man Mans
71332 Zero - The Pain Machinery
71352 Zopf: pigtail - Penguin Café Orchestra
7137s2
5123 Your Songs (Alternate Take 10) - Cilla Black2
7126 Your Visits Are Getting Shorter - Bloc Party2
7127 Your Woman - White Towns
7130 Ze Rook Naar Rozen - Rob De Nijs
7131 Zebra - Beach Houses
7132 Zebra - Man Mans
7133 Zero - The Pain Machinery
7132 Zopf: pigtail - Penguin Café Orchestra
5137s2 Anyway - Armand Van Helden & A-TRAK Present Du.
51392 high fives - Four Tets
51402 in white rooms - Booka Shades
51432 paranoid's androids - Christophers O'Riley
51492 LoVes [Piano ¿LoVes? [Piano Y Voz] - Alehjandro Sanz
51502 Época - Gotan Project
51512 rows × 3 columns
```

 [For Videos Join Our Youtube Channel: Join Now](#)

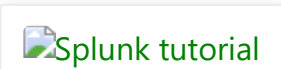
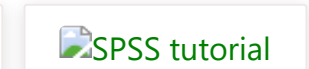
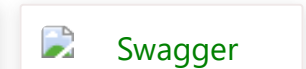
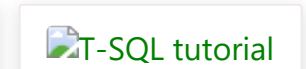

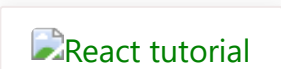
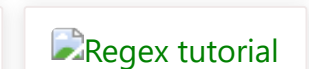


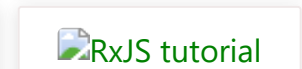
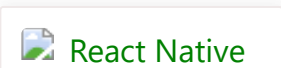



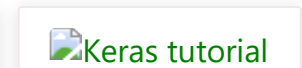
Feedback

- Send your Feedback to feedback@javatpoint.com

Help Others, Please Share



Learn Latest Tutorials


 Splunk	 SPSS	 Swagger	 Transact-SQL	 Tumblr
 ReactJS	 Regex	 Reinforcement Learning	 R Programming	 RxJS
 React Native	 Python Design Patterns	 Python Pillow	 Python Turtle	 Keras

Preparation

 Aptitude	 Logical Reasoning	 Verbal Ability	 Interview Questions
---	--	---	---


Reasoning

Interview Questions


 Company Interview Questions

Company Questions


Trending Technologies

 Artificial Intelligence


Artificial Intelligence

 AWS Tutorial


AWS

 Selenium tutorial


Selenium

 Cloud Computing


Cloud Computing

 Hadoop tutorial


Hadoop

 ReactJS Tutorial


ReactJS

 Data Science Tutorial


Data Science

 Angular 7 Tutorial


Angular 7

 Blockchain Tutorial


Blockchain

 Git Tutorial

Git


 Machine Learning Tutorial

Machine Learning


 DevOps Tutorial

DevOps


B.Tech / MCA

 DBMS tutorial


DBMS

 Data Structures tutorial


Data Structures

 DAA tutorial


DAA

 Operating System


Operating System

 Computer Network tutorial


Computer Network

 Compiler Design tutorial


Compiler Design

 Computer Organization and Architecture


Computer Organization

 Discrete Mathematics Tutorial














Discrete Mathematics

 Ethical Hacking

Ethical Hacking

 Computer Graphics Tutorial

Computer Graphics

 <div>Software Engineering</div> <div>Software Engineering</div>	 <div>html tutorial</div> <div>Web Technology</div>	 <div>Cyber Security tutorial</div> <div>Cyber Security</div>	 <div>Automata Tutorial</div> <div>Automata</div>	 <div>C Language tutorial</div> <div>C Programming</div>
 <div>C++ tutorial</div> <div>C++</div>	 <div>Java tutorial</div> <div>Java</div>	 <div>.Net Framework tutorial</div> <div>.Net</div>	 <div>Python tutorial</div> <div>Python</div>	 <div>List of Programs</div> <div>Programs</div>
 <div>Control Systems tutorial</div> <div>Control System</div>	 <div>Data Mining Tutorial</div> <div>Data Mining</div>	 <div>Data Warehouse Tutorial</div> <div>Data Warehouse</div>		