Managing ML Data and Models for Spotify Music Recommendation

Final Report

Name: Kehan Bi
USC ID: 2608579504
Course: DSCI 551, Tue, Fall 2021
Date: December 1, 2021

I. About Me:

My background is Mathematics and minor in Business in Undergraduate school. This is one-person group and I did not have computer science background or UI background.

II. Project Introduction and Motivation:

Music is an indispensable part of life. Music accompanies people on the way to and get off work, time before going to bed, and even an alarm to wake up. I love listening to music, but I often feel troubled because there is usually not enough music in my playlist. The music software recommended me music that I might like every day. Everyone has a specific type of music they like. So how does Spotify recommend music to us? The goal of this project is to recommend few artists that users may like based on genre of music which given by users.

III. Parts and Functions in project:

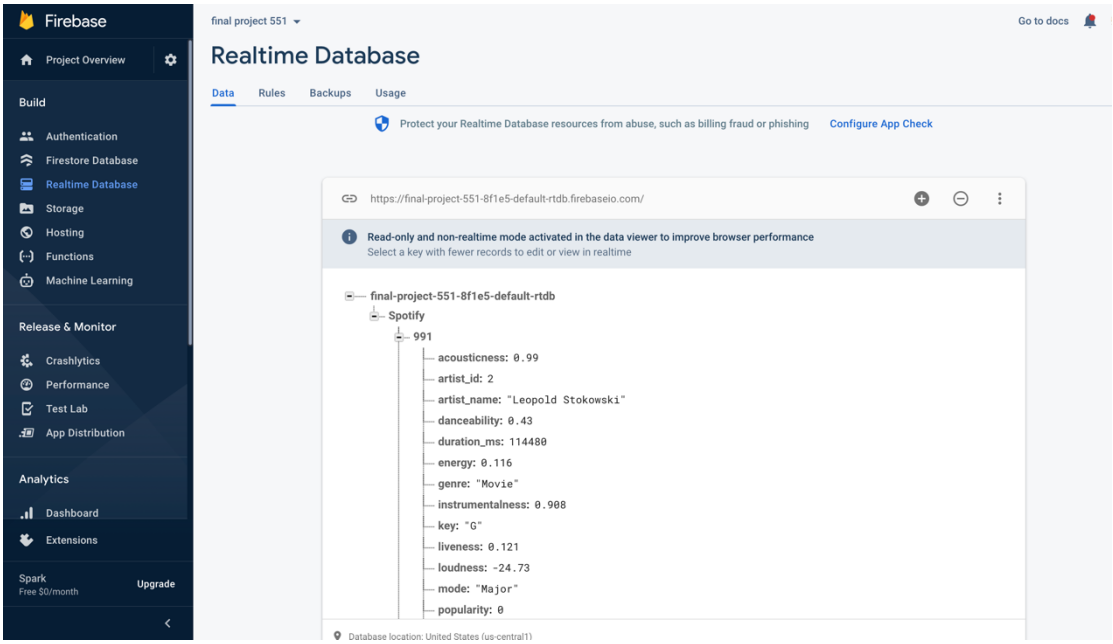
(The details will be explained later)

1. **Cloud Database**
2. **Data preparation**
   a) Clean and transform data
   b) Extract features and metadata from raw data
   c) Upload raw data, metadata, and features to Firebase
3. **Interface for user to upload file and upload metadata**
   a) User can upload .csv or .json files
   b) User can customize the special split symbols for their .csv files
   c) User can type in the description of dataset, the input from users will storage in dataset.
   d) Metadata and features are auto extraction by app when user click “upload” to upload the file.
   e) Metadata, features, description, and dataset upload to Firebase at same time.
4. **Interface for user to view the dataset from Firebase:**
   a) User can select the datasets, explore the description, metadata, features, first 10 lines, last 10 lines, and random 10 lines of datasets.
   b) The new dataset which uploaded by users show in the list with their metadata. Users able to explore them.

5. **Machine Learning Method - LightFM:**
   a) Set up ML model
   b) Set the parameter
6. **Show recommended result**
   a) User can type in the genre of music they like

IV. Cloud Database:

The data is storage in Firebase Realtime Database:



V. Datasets:

- SpotifyFeature.csv download from Kaggle
- Features: Genre; Artist_name; Track_name; Track_id; Popularity; Acousticness; Danceability; Duration_ms; Energy; Instrumentalness; Key; Liveness; Loudness; Mode; Speechiness; Tempo; Time_signature; Valence; Artist_id
- Genre, artist_name, popularity, and artist_id will be used in my model
- I choose the data with the Artist who has music records more than 50 times. Because the filtered artists are more popular and make the train dataset in more general condition. The result of recommendation will be more reliability.
- The dataset is too large to upload to and request from Firebase, I selected top 10000 rows for upload and use in model.

Data preparation:

Link of Firebase: https://final-project-551-8f1e5-default-rtdb.firebaseio.com/.json

1. Upload dataset to Firebase:
   I give each “artist_name” an “actist_id”, because integer type of data is easier to run in machine learning model.

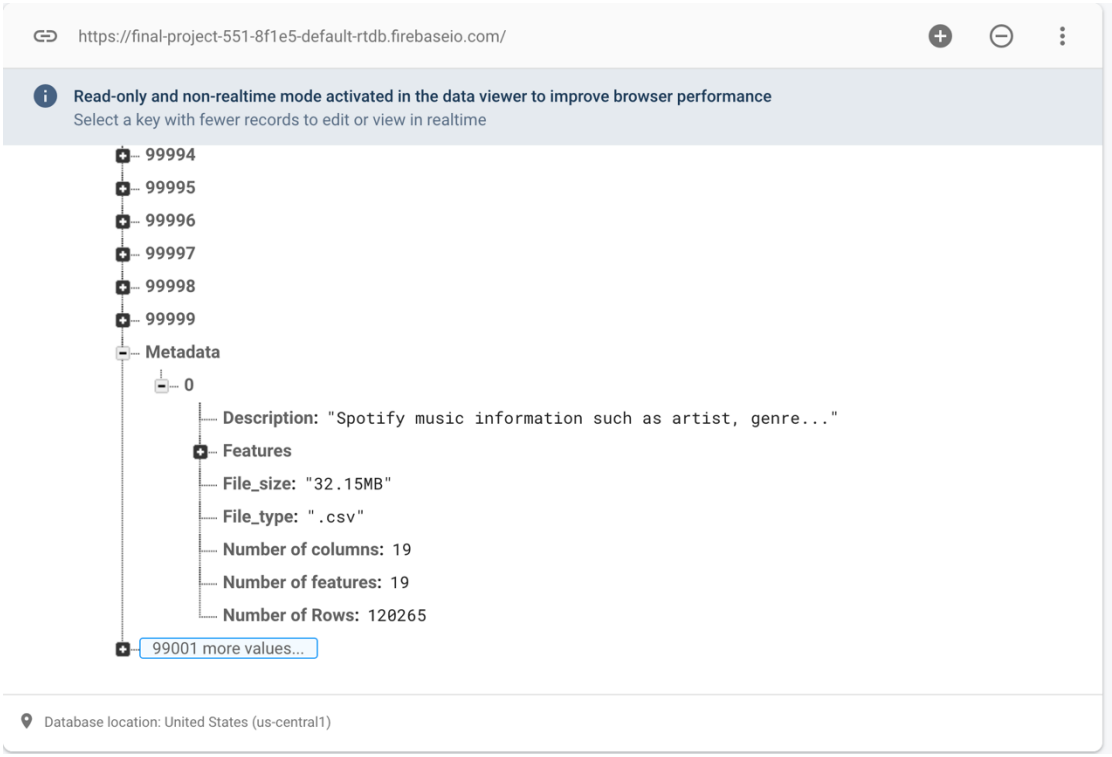
```
#upload dataset to Firebase Realtime database
spotify = pd.read_csv("data/SpotifyFeatures.csv", error_bad_lines=False, sep = ',')
#use data which the artist accurs more than 50 times
spotify = spotify.groupby('artist_name').filter(lambda x : len(x)>=50)
df_artist = pd.DataFrame(spotify["artist_name"].unique())
df_artist = df_artist.reset_index()
df_artist = df_artist.rename(columns={'index':'artist_id', 0:'artist_name'})
df_artist
spotify  = pd.merge(spotify , df_artist, how='inner', on='artist_name')

jsondata = spotify.head(100000).to_json(orient = "records")
url = "https://final-project-551-8f1e5-default-rtdb.firebaseio.com/Spotify.json"
response = requests.put(url, jsondata)

print("-----Data Uploaded To Firebase Successful-----")
```

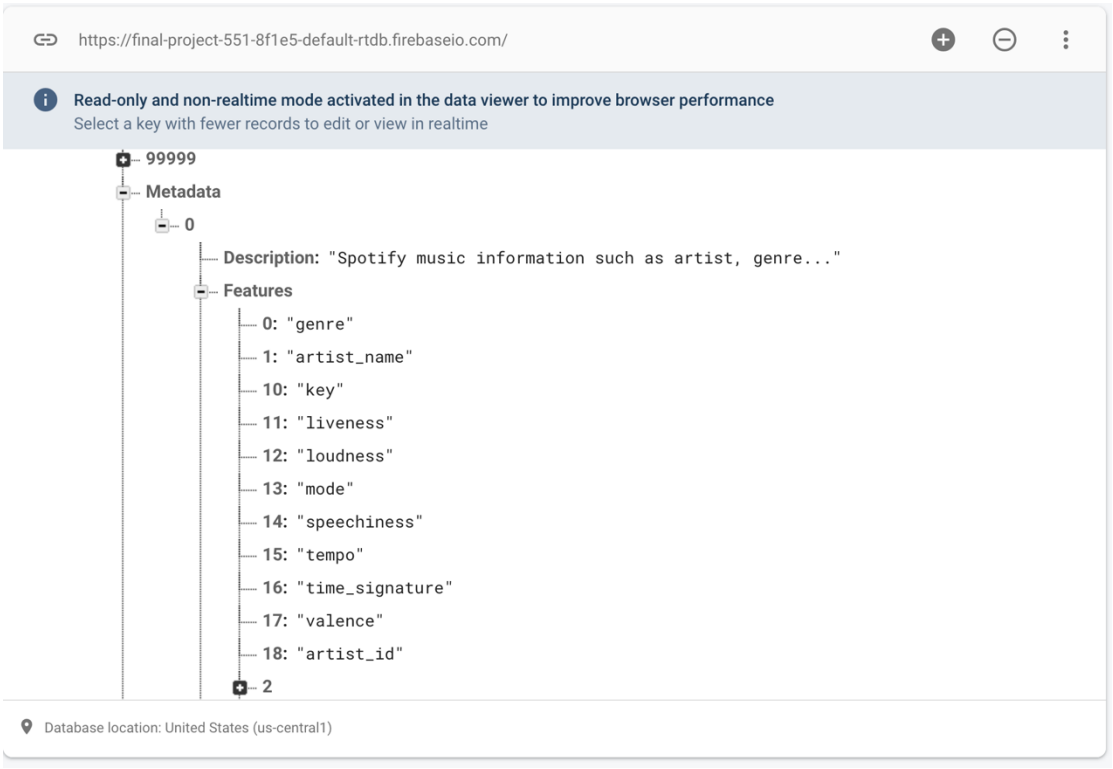
```
-----Data Uploaded To Firebase Successful-----
```

2. Extract metadata and features from dataset and upload to Firebase: (The users could view data and metadata through interface, I will explain it later).
   This part gets the file size, file type, number of rows and columns, and how many features and what they are.

```
#upload metadata to firebase
url2 = "https://final-project-551-8f1e5-default-rtdb.firebaseio.com/Spotify/Metadata.json"
path = "data/SpotifyFeatures.csv"

fsize = os.path.getsize(path)
fsize = fsize/float(1024*1024)
fsize = str(round(fsize,2)) + 'MB'
ext = os.path.splitext(path)[-1]
rows_num = spotify.shape[0]
columns_num = spotify.shape[1]
features_num = len(spotify.columns.values)
features = spotify.columns.values

meta_dic = {"File_size":[fsize], "File_type": [ext], "Number of Rows": [rows_num],
            "Number of columns" : [columns_num], "Number of features": [features_num], "Features": [features],
             "Description" : ["Spotify music information such as artist, genre of music, track name, popularity, etc
describe = pd.DataFrame(meta_dic)
describe = describe.to_json(orient = "records")
des = requests.put(url2, describe)
```

The metadata shows in Firebase as following:





VI. Interface for user to upload file and upload metadata:

1. This part allows the users upload .csv and .json files to Firebase.
2. When user click on the upper “Upload” button, user can choose file from local. When user click on the lower “Upload” button, the file will be upload to Firebase, and the metadata, features, and description will be upload as well.
3. I set the split symbol of csv file as “;” by default. User can customize the special split symbols for their .csv files, i.e “\t” in film1.csv from our HW5.
4. User allow to type in the description of their dataset. The content will be upload with metadata to Firebase by click lower “Upload” button.
5. Metadata and features of their files are auto extraction by my app when user click “Upload” to upload the file.
6. Metadata, features, description, and dataset upload to Firebase at same time.

Code shows as follow:

```python
# user can upload file here
def on_button_upload_file(b):
    with output1:
        output1.clear_output()
        uploaded_file = uploader.value
        print("Successful upload file", list(uploaded_file.keys())[0])
        #print(uploaded_file)
        metadata = list(uploaded_file.values())[0]['metadata']
        ext = os.path.splitext(metadata['name'])[-1]
        print("Metadata:\nFile type:",ext)
        fsize = metadata['size']
        fsize = fsize/float(1024*1024)
        print("File size:", str(round(fsize,2)), 'MB')
        file_name = os.path.splitext(metadata['name'])[0]
        #print(file_name)
        content = list(uploaded_file.values())[0]['content']
        describe = str(text_file_description.value)
        if ext == '.csv':  #csv file
            if len(str(text_split_symbol.value))==0:
                split = ";"
            else:
                split = str(text_split_symbol.value)
            upload_csv = pd.read_csv(io.BytesIO(content), sep = split)
            rows_num = upload_csv.shape[0]
            columns_num = upload_csv.shape[1]
            features_num = len(upload_csv.columns)
            features = upload_csv.columns.values
            print(features)
            print(type(features))
            print("There are {} rows and {} columns in dataset.".format(rows_num, columns_num))
            print("There are {} features in dataset".format(features_num))
            print("The features and their datatypes:")
            for i in range(len(upload_csv.columns)):
                print(" |-" + upload_csv.columns[i] + " - " + str(upload_csv.dtypes[i]))

            #upload dataset to firebase
            url_upload_csv = "https://final-project-551-8f1e5-default-rtdb.firebaseio.com/" \
                                                + str(file_name) + ".json"
            csv_to_json = upload_csv.to_json(orient = "records")
            #print(csv_to_json)
            response = requests.put(url_upload_csv, csv_to_json)
            print("{} has been uploaded to Firebase.".format(list(uploaded_file.keys())[0]))

            #upload metadata
            url_upload_csv_metadata = "https://final-project-551-8f1e5-default-rtdb.firebaseio.com/" \
                                                + str(file_name) + "/Metadata.json"
            meta_dic_csv = {"File_size":[fsize], "File_type": [ext], "Number of Rows": [rows_num],
                       "Number of columns" : [columns_num], "Number of features": [features_num],
                            "Features": [features], "Description" : [describe]}
            describe_csv = pd.DataFrame(meta_dic_csv)
            describe_csv = describe_csv.to_json(orient = "records")
            des = requests.put(url_upload_csv_metadata, describe_csv)
        elif ext == '.json':  #json file
            json_content = json.loads(str(content,'utf-8'))
            rows_num_json = len(json_content)
            json_feature = list(json_content[0].keys())
            features_num_json = len(json_feature)

            print("There are {} rows and {} columns in dataset.".format(rows_num_json, features_num_json))
            print("There are {} features in dataset".format(features_num_json))
            print("The features and their datatypes:")
            for key, value in json_content[0].items():
                print(" |-" , key, " - ", str(type(value)))

            #upload dataset to firebase
            upload_json = str(content,'utf-8')
            url_upload_json = "https://final-project-551-8f1e5-default-rtdb.firebaseio.com/" \
                                                + str(file_name) + ".json"
            response_json = requests.put(url_upload_json, upload_json)
            print("{} has been uploaded to Firebase.".format(list(uploaded_file.keys())[0]))

            #upload metadata to firebase
            url_upload_json_metadata = "https://final-project-551-8f1e5-default-rtdb.firebaseio.com/" \
                                                + str(file_name) + "/Metadata.json"
            meta_dic_json = {"File_size":[fsize], "File_type": [ext], "Number of Rows": [rows_num_json],
                       "Number of columns" : [features_num_json], "Number of features": [features_num_json],
                             "Features": [json_feature], "Description" : [describe]}
            describe_json = pd.DataFrame(meta_dic_json)
            describe_json = describe_json.to_json(orient = "records")
            des = requests.put(url_upload_json_metadata, describe_json)
        else:
            print("The file type is uncorrect, please upload other file.")
```

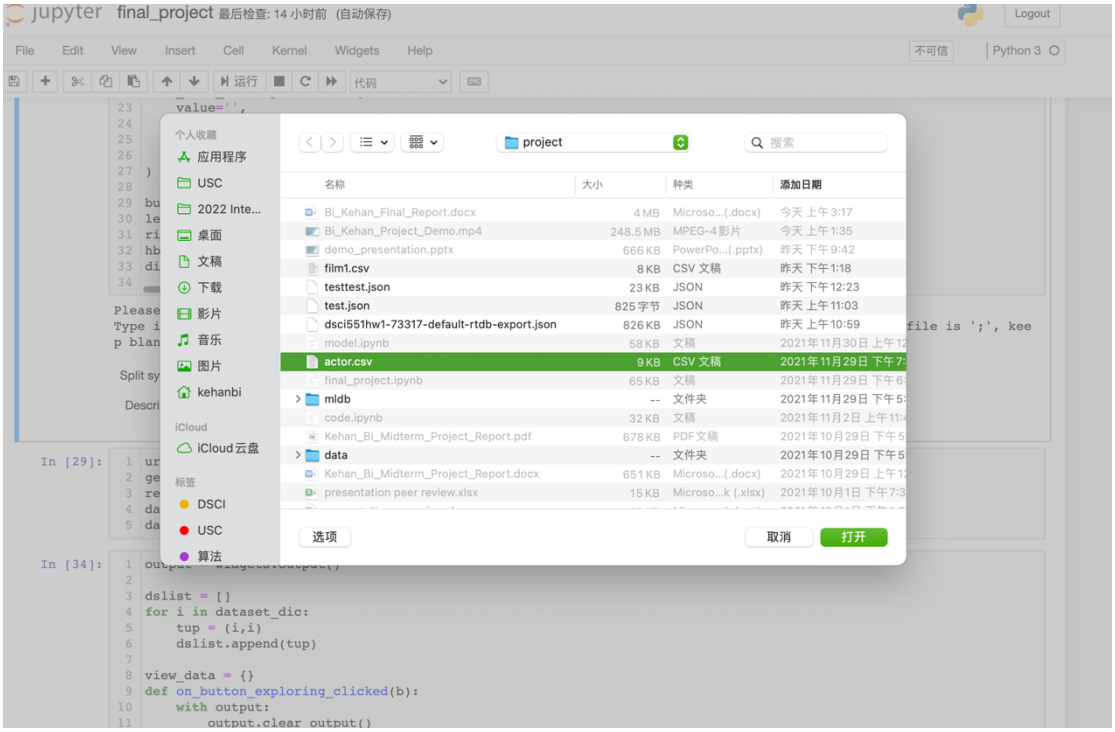
```
print("Please upload .csv or .json files here.")
print("Type in the special split symbol of .csv files here, e.g. ',';'\\t'; etc. If the split symbol of your file i
output1 = widgets.Output()
uploader = widgets.FileUpload(
    accept='.csv,.json',  # Accepted file extension e.g. '.txt', '.pdf', 'image/*', 'image/*,.pdf'
    multiple=True  # True to accept multiple files upload else False
)
button = widgets.Button(
    description='Upload',
    disabled=False,
    button_style='', # 'success', 'info', 'warning', 'danger' or ''
    tooltip='Click me2',
    icon='check' # (FontAwesome names without the `fa-` prefix)
)
text_split_symbol = widgets.Text(
    value='',
    placeholder='Type in the special split symbol',
    description='Split symbol:',
    disabled=False
)

text_file_description = widgets.Text(
    value='',
    placeholder='Describe your file in short sentence',
    description='Description:',
    disabled=False
)

button.on_click(on_button_upload_file)
left_box = widgets.VBox([text_split_symbol, text_file_description])
right_box = widgets.VBox([uploader, button])
hbox = widgets.HBox([left_box, right_box])
display(hbox, output1)
```

```
Please upload .csv or .json files here.
Type in the special split symbol of .csv files here, e.g. ',';'\t'; etc. If the split symbol of your file is ';', kee
p blank.
```

Split symbol: Type in the special split symbol | Upload (0)

Description: Describe your file in short sentenc | Upload

- **When click upper “Upload” button:**



- **Type in description and click upload:**

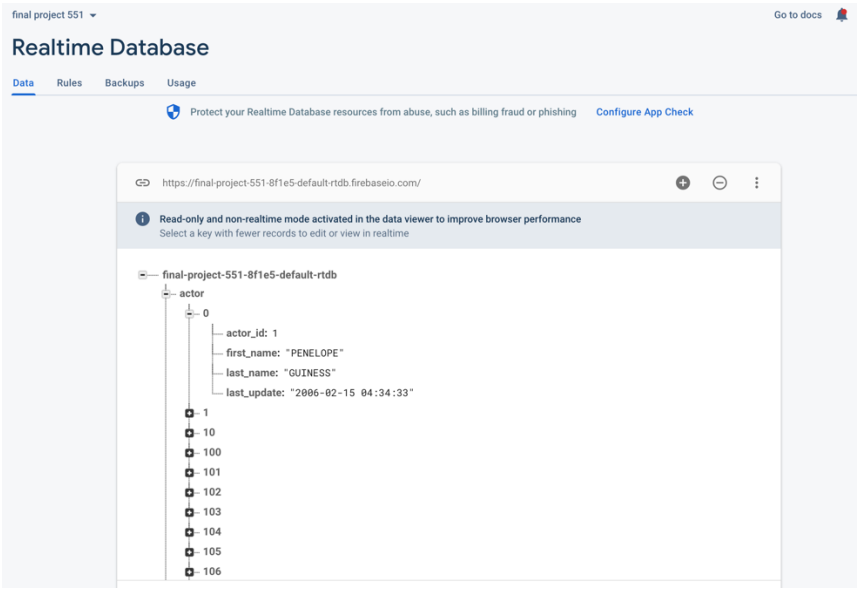
```
Please upload .csv or .json files here.
Type in the special split symbol of .csv files here, e.g. ',';'\t'; etc. If the split symbol of your file is ';', kee
p blank.
```

Split symbol: Type in the special split symbol | Upload (1)

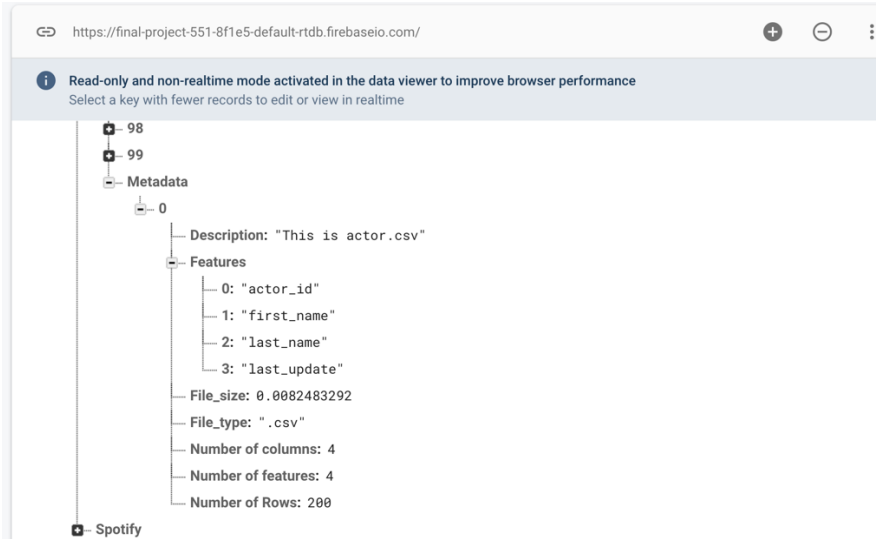
Description: This is actor.csv | ✔ Upload

```
Successful upload file actor.csv
Metadata:
File type: .csv
File size: 0.01 MB
['actor_id' 'first_name' 'last_name' 'last_update']
<class 'numpy.ndarray'>
There are 200 rows and 4 columns in dataset.
There are 4 features in dataset
The features and their datatypes:
 |-actor_id - int64
 |-first_name - object
 |-last_name - object
 |-last_update - object
actor.csv has been uploaded to Firebase.
```

- **The data of actor.csv shows in Firebase:**



- **The metadata shows in Firebase as well:**



VII. Interface for user to view the dataset from Firebase:

a) User can select the datasets, explore the description, metadata, features, first 10 lines, last 10 lines, and random 10 lines of datasets.

b) The new dataset which uploaded by users show in the list with their metadata. Users able to explore them.

1. Request data from Firebase:

```
url_get_data = "https://final-project-551-8f1e5-default-rtdb.firebaseio.com/.json"
get_data_response = requests.get(url_get_data)
reply = get_data_response.json()
dataset_dic = reply.keys()
dataset_dic = list(dataset_dic)
```

2. Set up definition of user interface, I use ipywidget package.

```
output = widgets.Output()

dslist = []
for i in dataset_dic:
    tup = (i,i)
    dslist.append(tup)

view_data = {}
def on_button_exploring_clicked(b):
    with output:
        output.clear_output()
        view_data = reply[dropdown_ds.value]
        meta = pd.DataFrame(view_data["Metadata"])
        #print(meta)
        df2 = view_data.copy()
        del df2['Metadata']
        line_data = pd.DataFrame(df2)
        if b.description == 'View Dataset':
            print("Dataset: {}".format(dropdown_ds.value))
            print("Description:", view_data["Metadata"][0]['Description'])
        elif b.description == 'Metadata':
            print(meta.T)
            print("\nFeatures are: ")
            for i in list(meta["Features"])[0]:
                print("|-", i)
            #, list(meta["Features"])[0])
            #print("Dataset has {} rows and {} columns\nNumber of features = {}\nFeatures are: {}".format(valdic[0]
        elif b.description == 'First 10 lines':
            print(line_data.head(10).T)
            #print(valdic[0][1].head(10))
        elif b.description == 'Last 10 lines':
            print(line_data.tail(10).T)
            #print(valdic[0][1].tail(10))
        else:
            print(line_data.sample(n=2).T)
```

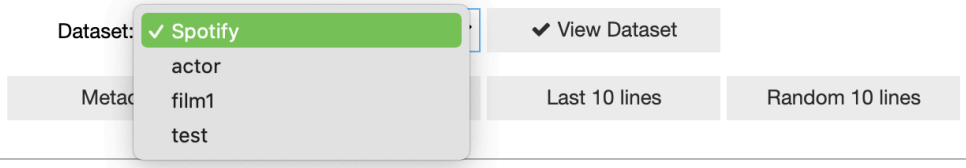
3. Interface show as follow:

```
dropdown_ds = widgets.Dropdown(
    options = dslist,
    value = "Spotify",
    description='Dataset:',
)
button_exploring = widgets.Button(
    description='View Dataset',
    disabled=False,
    button_style='',
    tooltip='Click me2',
    icon='check'
)
button_metadata = widgets.Button(
    description='Metadata',
    disabled=False,
    button_style='',
)
button_head10 = widgets.Button(
    description='First 10 lines',
    disabled=False,
    button_style='',
)
button_tail10 = widgets.Button(
    description='Last 10 lines',
    disabled=False,
    button_style='',
)
button_rand10 = widgets.Button(
    description='Random 10 lines',
    disabled=False,
    button_style='',
)
radiobuttons = widgets.RadioButtons(
    options=['First 10 lines', 'Last 10 lines', 'Random 10 lines'],
    description='Preview of data:',
    disabled=False
)

button_exploring.on_click(on_button_exploring_clicked)
button_metadata.on_click(on_button_exploring_clicked)
button_head10.on_click(on_button_exploring_clicked)
button_tail10.on_click(on_button_exploring_clicked)
button_rand10.on_click(on_button_exploring_clicked)

left_box = widgets.VBox([dropdown_ds])
right_box = widgets.VBox([button_exploring])
hbox1 = widgets.HBox([left_box, right_box])
hbox2 = widgets.HBox([button_metadata, button_head10,button_tail10,button_rand10])
display(hbox1,hbox2,output)
```


```
Dataset: Spotify
Description: Spotify music information such as artist, genre of music, track name, popularity, etc.
```

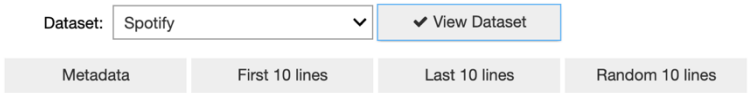
4. The drop-down box shows the dataset storage in Firebase. “Actor”, “film1”, and “test” is test file that I upload in previous interface:

```
left_box = widgets.VBox([dropdown_ds])
right_box = widgets.VBox([button_exploring])
hbox1 = widgets.HBox([left_box, right_box])
hbox2 = widgets.HBox([button_metadata, button_head10,button_tail10,button_
display(hbox1,hbox2,output)
```




```
32]:
def  on_button_ml_model(b):
    with output2:
        output2.clear_output()
        if b.description == 'Upload':
            rec_list = sample_recommendation_user(model = model,
```

5. When click “View Dataset”, the dataset name and description will be show in the output:



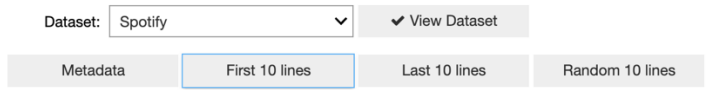

```
Dataset: Spotify
Description: Spotify music information such as artist, genre of music, track name, popularity, etc.
```

---

6. When click “Metadata”, the metadata and features will be show as follow:


```
                                                          0
Description          Spotify music information such as artist, genr...
Features             [genre, artist_name, track_name, track_id, pop...
File_size                                                32.15MB
File_type                                                   .csv
Number of Rows                                           120265
Number of columns                                            19
Number of features                                           19

Features are:
|- genre
|- artist_name
|- track_name
|- track_id
|- popularity
|- acousticness
|- danceability
|- duration_ms
|- energy
|- instrumentalness
|- key
|- liveness
|- loudness
|- mode
|- speechiness
|- tempo
|- time_signature
|- valence
|- artist_id
```

7. When click on “First 10 lines” button, user can preview the dataset. “Last 10 lines” and “Random 10 lines” are similar:




```
       acousticness artist_id     artist_name danceability duration_ms  energy  \
0             0.611         0  Henri Salvador        0.389       99373    0.91
1             0.703         0  Henri Salvador         0.24      152427   0.326
2             0.749         0  Henri Salvador        0.578      160627  0.0948
3             0.381         0  Henri Salvador        0.451      194360   0.491
4             0.689         0  Henri Salvador        0.704      161773   0.804
...             ...       ...             ...          ...         ...     ...
99995         0.249       930  The Cat Empire        0.672      313053   0.603
99996         0.189       930  The Cat Empire        0.524      225467   0.765
99997         0.019       930  The Cat Empire        0.419      195547   0.575
99998         0.249       930  The Cat Empire        0.672      313053   0.603
99999       0.00727       930  The Cat Empire        0.613      258573   0.762

       genre instrumentalness key liveness
0      Movie                0  C#    0.346
1      Movie                0  C#   0.0985
2      Movie                0  C#    0.107
3      Movie                0   D    0.152
4      Movie           0.0422   C     0.18
...      ...              ...  ..      ...
99995   Jazz          0.00277   D    0.127
99996    Ska                0   F    0.153
99997    Ska         4.83e-06  A#    0.158
99998    Ska          0.00277   D    0.127
99999    Ska                0   B    0.188

[100000 rows x 10 columns]
```

VIII. Machine Learning Method - LightFM:

- LightFM is a Python implementation of a number of popular recommendation algorithms for both implicit and explicit feedback.
- The model learns embeddings (latent representations in a high-dimensional space) for users and items in a way that encodes user preferences over items. When multiplied together, these representations produce scores for every item for a given user; items scored highly are more likely to be interesting to the user.
- When user give a genre of music, the model will give recommended artists base on the genre, artists, and popularity. The model will give each music record score, and sort the score list until the model get the correct one. The top n of list will be output as results.

- Code:
  1. Get data from Firebase:

```
In [11]:
url_model = "https://final-project-551-8f1e5-default-rtdb.firebaseio.com/Spotify.json"
data_model = requests.get(url_model)
reply_model = data_model.json()
del reply_model['Metadata']
spotify_firebase = pd.DataFrame(reply_model)
spotify_firebase = spotify_firebase.T
```

  2. Create_interaction_matrix(): Group the dataset by genre, artist_id, and rating as popularity. I set the parameters for user_col = “genre”, item_col = “artist_id”, and rating_col = “popularity”.

     Create_user_dict(); create_item_dict(): Generate the dictionary by “genre” and “artist_id” separately.

     runMF(): Main model of lightFM, run lightFM model and fit it on my data.

     Parameters:

     - n_components – the dimensionality of the feature latent embeddings.
     - loss – one of (‘logistic’, ‘bpr’, ‘warp’, ‘warp-kos’): the loss function.

     WARP: Weighted Approximate-Rank Pairwise loss. Maximises the rank of positive examples by repeatedly sampling negative examples until rank violating one is found.

     - k – for k-OS training, the k-th positive example will be selected from the n positive examples sampled for every genre.

```
def create_interaction_matrix(df,user_col, item_col, rating_col, norm= False, threshold = None):
    interactions = df.groupby([user_col, item_col])[rating_col].sum().unstack().reset_index().fillna(0) \
    .set_index(user_col)
    if norm:
        interactions = interactions.applymap(lambda x: 1 if x > threshold else 0)
    return interactions

def create_user_dict(interactions):
    user_id = list(interactions.index)
    user_dict = {}
    counter = 0
    for i in user_id:
        user_dict[i] = counter
        counter += 1
    return user_dict

def create_item_dict(df,id_col,name_col):
    item_dict ={}
    for i in range(df.shape[0]):
        item_dict[(df.loc[i,id_col])] = df.loc[i,name_col]
    return item_dict

def runMF(interactions, n_components=30, loss='warp', k=15, epoch=30,n_jobs = 4):
    model = LightFM(no_components= n_components, loss=loss,k=k)
    model.fit(x,epochs=epoch,num_threads = n_jobs)
    return model
```

```
def sample_recommendation_user(model, interactions, actor_id, user_dict,
                               item_dict,threshold = 0,nrec_items = 10, show = True):
    n_users, n_items = interactions.shape
    user_x = user_dict[actor_id]
    scores = pd.Series(model.predict(user_x,np.arange(n_items)))
    scores.index = interactions.columns
    scores = list(pd.Series(scores.sort_values(ascending=False).index))

    known_items = list(pd.Series(interactions.loc[actor_id,:][interactions.loc[actor_id,:] > threshold].index) \
                       .sort_values(ascending=False))

    scores = [x for x in scores if x not in known_items]
    return_score_list = scores[0:nrec_items]
    known_items = list(pd.Series(known_items).apply(lambda x: item_dict[x]))
    scores = list(pd.Series(return_score_list).apply(lambda x: item_dict[x]))
    if show == True:
        print("\n Recommended:")
        counter = 1
        for i in scores:
            print(str(counter) + '- ' + i)
            counter+=1
    return return_score_list
```

3. Run all model by set up the parameters:

```
interactions = create_interaction_matrix(df = spotify_firebase, user_col = "genre", item_col = 'artist_id',
                                         rating_col = 'popularity', norm= False, threshold = None)
interactions
```

artist_id genre	0	1	2	3	4	5	6	7	8	9	...	1144	1145	1146	1147	1148	1149	1150	1151	1152
Alternative	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Anime	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Blues	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Children's Music	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Children's Music	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Classical	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Comedy	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Country	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Dance	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Electronic	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Folk	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Hip-Hop	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Indie	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Jazz	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Movie	2205.0	266.0	713.0	9730.0	165.0	1495.0	1.0	601.0	996.0	1123.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Opera	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Pop	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

```
user_dict = create_user_dict(interactions=interactions)
artists_dict = create_item_dict(df = df_artist, id_col = 'artist_id', name_col = 'artist_name')
x = sparse.csr_matrix(interactions.values)
train, test = lightfm.cross_validation.random_train_test_split(x, test_percentage=0.2, random_state=None)

```

```
%time
model = runMF(interactions = train,
                  n_components = 30,
                  loss = 'warp',
                  k = 15,
                  epoch = 30,
                  n_jobs = 4)
```

```
CPU times: user 2 µs, sys: 1 µs, total: 3 µs
Wall time: 5.96 µs
```

```
train_auc = auc_score(model, train, num_threads=4).mean()
print('Train AUC: %s' % train_auc)
test_auc = auc_score(model, test, train_interactions=train, num_threads=4).mean()
print('Test AUC: %s' % test_auc)
train_precision = precision_at_k(model, train, k=10).mean()
test_precision = precision_at_k(model, test, k=10, train_interactions=train).mean()
print('train Precision %.2f, test Precision %.2f.' % (train_precision, test_precision))
```

```
Train AUC: 0.9879274
Test AUC: 0.99997014
train Precision 0.78, test Precision 0.92.
```

4. Interface that let user type in the genre of music:

```
def  on_button_ml_model(b):
    with output2:
        output2.clear_output()
        if b.description == 'Upload':
            rec_list = sample_recommendation_user(model = model,
                                      interactions = interactions,
                                      actor_id = text_song_type.value,
                                      user_dict = user_dict,
                                      item_dict = artists_dict,
                                      threshold = 0,
                                      nrec_items = 10,
                                      show = True)
```

```
output2 = widgets.Output()
button = widgets.Button(
    description='Upload',
    disabled=False,
    button_style='', # 'success', 'info', 'warning', 'danger' or ''
    tooltip='Click me2',
    icon='check' # (FontAwesome names without the `fa-` prefix)
)

text_song_type = widgets.Text(
    value='',
    placeholder='Type in music type',
    description='Type:',
    disabled=False
)
text_song_type.value=''

button.on_click(on_button_ml_model)

left_box = widgets.VBox([text_song_type])
right_box = widgets.VBox([button])
hbox = widgets.HBox([left_box, right_box])
display(hbox, output2)
```

5. The model output the recommended artists: There are two examples:


```
 Recommended:
1- El Alfa
2- Keyshia Cole
3- Reik
4- Papa Roach
5- J Balvin
6- Godsmack
7- Creed
8- The Lacs
9- Ari Lennox
10- Akon
```


```
 Recommended:
1- Old Town School of Folk Music
2- Tom's Music Box
3- Yann Tiersen
4- Jóhann Jóhannsson
5- Umberto Giordano
6- Junkie XL
7- Gaetano Donizetti
8- Shigeru Umebayashi
9- Masakatsu Takagi
10- Dora The Explorer
```

IX. Learning Experiences

What are the lessons, challenges, experiences, and skills gained from the project?

After this project, I fully understand how to use the cloud databases and

what it can do for us to make the value. I learned how to set up the user interface, this is really helpful because the programs aim to help the people who do not understand the coding. I also learned how to structure a whole project. The logical relationship between modules are really important for a project. Need to keep mind very clearly.