Before you turn this problem in, make sure everything runs as expected. First, **restart the kernel** (in the menubar, select Kernel→Restart) and then **run all cells** (in the menubar, select Cell→Run All).

Make sure you fill in any place that says `YOUR CODE HERE` / `raise NotImplementedError` or "YOUR ANSWER HERE", as well as your name and collaborators below:

Processing and SQL for Relational Database Project

Author: Yanni Guo

I. 1.1 Development:

In [1]:
```
# This global variable controls whether to use "sqlite" versus "mysql" c
onnections

db_source = "sqlite"

import pandas as pd
import os
import os.path
import json
import sqlalchemy as sa           #builds a prepared object
import csv

def getsqlite_info(dirname=".",filename="creds.json"):
    """ Using directory and filename parameters, open a credentials file
        and obtain the four parts needed for a connection string to
        a remote provider using the "mysql" dictionary within
        an outer dictionary.

        Return a scheme, server, user, and password
    """
    assert os.path.isfile(os.path.join(dirname, filename))
    with open(os.path.join(dirname, filename)) as f:
        D = json.load(f)
    sqlite = D["sqlite"]
    return sqlite["scheme"], sqlite["basepath"], sqlite["database"]

if db_source == "sqlite":
    scheme, basepath, db = getsqlite_info()
    template = '{}:///{}.db'
    cstring = template.format(scheme, os.path.join(basepath, db))   # Co
nnection string for SQLite
elif db_source == "mysql":
    scheme, server, user, password, db = getmysql_creds()
    template = '{}://{}:{}@{}/{}'
    cstring = template.format(scheme, user, password, server,db)
else:
    raise ValueEror

# establish connection string:

engine=sa.create_engine(cstring)  # create engine object
connection = engine.connect()     # establish connection

cstring
```

Out[1]:
```
'sqlite+pysqlite:///./imdb3.db'
```

1.2 Load Notebook Extension to Enable "SQL Magic" and Establish connection from client to server

In [2]:
```
%load_ext sql
%sql $cstring
```

Out[2]:
```
'Connected: @./imdb3.db'
```

II. Question 1: What are the Average Rating of Each Genre Movie for Each Decade?

1. Create query to find range of votes for average movie rates:

In [3]:
```
query_votes = """
SELECT MAX(VOTES), MIN(VOTES), AVG(VOTES)
FROM Ratings
"""
vote_result = %sql $query_votes
vote_resultdf = vote_result.DataFrame()
vote_resultdf.head(10)
```

```
 * sqlite+pysqlite:///./imdb3.db
Done.
```

Out[3]:

	MAX(VOTES)	MIN(VOTES)	AVG(VOTES)
0	2200984	5	3261.147122

2. Create function that builds a query to find the average rate for each movie genre for each decade

In [4]:
```
def rating_genre_decade(dbcon1, year1, year2, year3, year4, year5, vote_
min, vote_max):
    """
    This function makes a query which contains the fields for movie rele
ase year, movie genre, and the average movie rating.

    Parameters:
    dbcon1: connects to database
    year1: variable for first year value
    year2: variable for second year value
    year3: variable for third year value
    year4: variable for fourth year value
    year5: variable for fifth year value
    vote_min: variable for minimum value of votes
    vote_max = variable for maximum value of votes

    Return value: Execute query_q1 to a pandas dataframe with binded SQL
variables
    """
    #create a query that contains fields for movie release year, movie g
enre, and average rating of movies with constraints

    query_q1 = """ SELECT M.ReleaseYear, G.Genre, AVG(R.AvgRating) AS av
erage_rating
                   FROM Ratings AS R LEFT JOIN Movies AS M
                        ON R.ID = M.MovieID
                        LEFT JOIN Movie_Genre AS MG
                        USING(MovieID)
                        LEFT JOIN Genres AS G
                        USING(GenreID)
                    WHERE M.ReleaseYear IN (:y1, :y2, :y3, :y4, :y5) AND
VOTES BETWEEN :v_min AND :v_max
                    GROUP BY M.ReleaseYear, G.Genre
                    ORDER BY M.ReleaseYear DESC """

    prepare_stmtv = sa.sql.text(query_q1)  #prepare statement object
    bound_stmtv = prepare_stmtv.bindparams(y1=year1, y2=year2, y3=year3,
y4=year4, y5=year5, v_min=vote_min, v_max=vote_max) #bound statement to
 bind named parameters
    df1 = pd.read_sql_query(bound_stmtv, con=dbcon1)  #execute query to
 dataframe

    return df1  #return dataframe

rating_genre_decade(connection, 1970, 1980, 1990, 2000, 2010, 261, 2621)
#call rating_genre_decade function
```

Out[4]:

	ReleaseYear	Genre	average_rating
0	2010	Action	5.225581
1	2010	Adventure	5.487500
2	2010	Animation	6.318750
3	2010	Biography	6.870370
4	2010	Comedy	5.595979
...	...	...	...
101	1970	Sci-Fi	4.735714
102	1970	Sport	6.550000
103	1970	Thriller	5.896429
104	1970	War	6.453846
105	1970	Western	5.786957

106 rows × 3 columns

III. Question 2: What is the Correalation Between Average Movie Rating vs. Movie Runtime and Movie Genre?

1. Create a query to find the max, min, and average movie runtime:

In [5]:
```
query_runtime ="""
SELECT MAX(MovieRunTime), MIN(MovieRunTime), AVG(MovieRunTime)
FROM Movies
"""
r = %sql $query_runtime
rdf = r.DataFrame()
rdf.head()
```

```
 * sqlite+pysqlite:///./imdb3.db
Done.
```

Out[5]:

	MAX(MovieRunTime)	MIN(MovieRunTime)	AVG(MovieRunTime)
0	51420	1	88.862854

2. Create function that builds query to find the average rating, genre, and runtime for each movie

```
In [6]: def rating_genre_runtime(dbcon2, time_min, time_max, vote_min, vote_max
        ):
            """
            This function creates a query that contains the fields for average m
        ovie rating, movie genre, and movie runtime

            Parameters:
            dbcon2 = connects to database
            time_min = variable for minimum value of movie runtime
            time_max = variable for maximum value of movie runtime
            vote_min = variable for minimum value of votes
            vote_max = variable for maximum value of votes

            Return value: execute query_q2 to a pandas dataframe with binded SQL
        variables

            """
            #creates a query that contains field for average movie rating, movie
        genre, and average movie runtime with constraints

            query_q2 = """
            SELECT R.AvgRating, G.Genre, M.MovieRunTime
            FROM Ratings AS R LEFT JOIN Movies AS M
                ON R.ID = M.MovieID
                LEFT JOIN Movie_Genre AS MG
                USING(MovieID)
                LEFT JOIN Genres AS G
                USING(GenreID)
            WHERE M.MovieRunTime BETWEEN :tmin AND :tmax AND VOTES BETWEEN :vmin
        2 AND :vmax2
            ORDER BY R.AvgRating ASC
            """
            prepare_stmt2 = sa.sql.text(query_q2)  #prepare statement object for
        query_q2
            bound_stmt2 = prepare_stmt2.bindparams(tmin=time_min, tmax=time_max,
        vmin2=vote_min, vmax2=vote_max)  #bound statement to bind named paramete
        rs
            df2 = pd.read_sql_query(bound_stmt2, con=dbcon2)  #execute query to
         dataframe
            return df2                                  #return dataframe


        rating_genre_runtime(connection, 40, 300, 261, 2621) #call rating_genre_
        runtime function
```

Out[6]:

	AvgRating	Genre	MovieRuntime
0	1.0	Comedy	82
1	1.0	Mystery	82
2	1.0	Comedy	81
3	1.0	Documentary	119
4	1.0	Documentary	70
...	...	...	...
68525	9.7	Crime	100
68526	9.7	Documentary	100
68527	9.7	Comedy	100
68528	9.7	Comedy	121
68529	10.0	Drama	94

68530 rows × 3 columns

IV. Question 3: What are the numbers of actors and actresses in each movie genre?

1. Find the max, min, and average number of actors for a movie

In [7]:
```
query_mcount = """
SELECT MAX(actor_count), MIN(actor_count), AVG(actor_count)
FROM (SELECT COUNT(Job) AS actor_count
      FROM Movie_Person
      WHERE Job = 'actor'
      GROUP BY MovieID) AS MP
"""
resulta = %sql $query_mcount
resultdfa = resulta.DataFrame()
resultdfa.head()
```

```
 * sqlite+pysqlite:///./imdb3.db
Done.
```

Out[7]:

	MAX(actor_count)	MIN(actor_count)	AVG(actor_count)
0	10	1	2.764824

2. Find the max, min, and average number of actresses for a movie

In [8]:
```
query_fcount = """
SELECT MAX(actress_count), MIN(actress_count), AVG(actress_count)
FROM (SELECT COUNT(Job) AS actress_count
      FROM Movie_Person
      WHERE Job = 'actress'
      GROUP BY MovieID) AS MF
"""

resultb = %sql $query_fcount
resultdfb = resultb.DataFrame()
resultdfb.head()
```

```
 * sqlite+pysqlite:///./imdb3.db
Done.
```

Out[8]:

	MAX(actress_count)	MIN(actress_count)	AVG(actress_count)
0	10	1	1.895742

3. Create query to count the number of jobs in each movie that is an actor

```
In [9]: query_jobcount1 = """
        SELECT MovieID, COUNT(Job) AS count
        FROM Movie_Person
        WHERE Job = 'actor'
        GROUP BY MovieID
        HAVING count >= 1 AND count <= 3
        ORDER BY count DESC
        """

        resultc = %sql $query_jobcount1
        resultdfc = resultc.DataFrame()
        resultdfc.head(5)
```

```
 * sqlite+pysqlite:///./imdb3.db
Done.
```

Out[9]:

	MovieID	count
0	tt0016906	3
1	tt0035423	3
2	tt0054724	3
3	tt0058950	3
4	tt0059900	3

4. Create a query to count the number of jobs in each movie that is an actress

In [10]:
```
query_jobcount2 = """
SELECT MovieID, COUNT(Job) AS count
FROM Movie_Person
WHERE Job = 'actress'
GROUP BY MovieID
HAVING count >= 1 AND count <= 3
ORDER BY count DESC
"""

resultd = %sql $query_jobcount2
resultdfd = resultd.DataFrame()
resultdfd.head(5)
```

```
 * sqlite+pysqlite:///./imdb3.db
Done.
```

Out[10]:

	MovieID	count
0	tt0031458	3
1	tt0060967	3
2	tt0061876	3
3	tt0062847	3
4	tt0063498	3

5. Create query to count the number of actor jobs for each movie genre

In [11]:
```
queryMale = """
SELECT G.Genre, COUNT(*) AS Male_count
FROM (SELECT MovieID, COUNT(Job) AS count
        FROM Movie_Person
        WHERE Job = 'actor'
        GROUP BY MovieID
        HAVING count >= 1 AND count <= 3
        ORDER BY count DESC) AS MP
    LEFT JOIN Movies AS M
    USING(MovieID)
    LEFT JOIN Movie_Genre AS MG
    USING(MovieID)
    LEFT JOIN Genres AS G
    USING(GenreID)
GROUP BY G.Genre
ORDER BY Male_count ASC
"""
male_result = %sql $queryMale
male_resultdf = male_result.DataFrame()
male_resultdf.head()
```

```
 * sqlite+pysqlite:///./imdb3.db
Done.
```

Out[11]:

	Genre	Male_count
0	Game-Show	2
1	Short	9
2	Talk-Show	20
3	Reality-TV	47
4	Adult	64

6. Create query to count the number of actress jobs for each movie genre

In [12]:
```
queryFemale = """
SELECT G.Genre, COUNT(*) AS Female_count
FROM (SELECT MovieID, COUNT(Job) AS count
        FROM Movie_Person
        WHERE Job = 'actress'
        GROUP BY MovieID
        HAVING count >= 1 AND count <= 3) AS MP
    LEFT JOIN Movies AS M
    USING(MovieID)
    LEFT JOIN Movie_Genre AS MG
    USING(MovieID)
    LEFT JOIN Genres AS G
    USING(GenreID)
GROUP BY G.Genre
ORDER BY Female_count ASC
"""

female_result = %sql $queryFemale
female_resultdf = female_result.DataFrame()
female_resultdf.head()
```

```
 * sqlite+pysqlite:///./imdb3.db
Done.
```

Out[12]:

	Genre	Female_count
0	Game-Show	3
1	Short	5
2	Talk-Show	10
3	Reality-TV	37
4	Adult	60

7. Create a function that combines actor and actress query to form a table that includes the count of actors and actress in each movie genre

In [13]:
```
def gender_movie_genre(dbcon3, job1, cast_min, cast_max, job2, cast2_min
, cast2_max):
    """
    This function creates a query that contains fields for movie genre,
 count for actresses in each movie genre, and count for
    actors in each movie genre.

    Parameters:
    dbcon3 = connects to database
    job1 = variable for job specification from first table
    cast_min = variable for minimum value of specific job position from
 first table
    cast_max = variable for maximum value of specific job position form
 first table
    job2 = variable for second job specification for second table
    cast2_min = variable for minimum value of specific job position from
second table
    cast2_max = variable for maximum value of specific job position form
second table

    Return value: execute query_q3 to a pandas dataframe with binded SQL
variables

    """
    #creates query for joining of actor and actress dataframe, contains
 fields for movie genre, actress count and actor count per genre, with c
onstraints

    query_q3 = """
    SELECT *
    FROM (
    SELECT G.Genre, COUNT(*) AS count1
    FROM (SELECT MovieID, COUNT(Job) AS count
            FROM Movie_Person
            WHERE Job = :j1
            GROUP BY MovieID
            HAVING count >= :c1_min AND count <= :c1_max) AS MP
        LEFT JOIN Movies AS M
        USING(MovieID)
        LEFT JOIN Movie_Genre AS MG
        USING(MovieID)
        LEFT JOIN Genres AS G
        USING(GenreID)
    GROUP BY G.Genre) AS left_query

    INNER JOIN
    (SELECT G.Genre, COUNT(*) AS count2
    FROM (SELECT MovieID, COUNT(Job) AS count
            FROM Movie_Person
            WHERE Job = :j2
            GROUP BY MovieID
            HAVING count >= :c2_min AND count <= :c2_max
            ORDER BY count DESC) AS MP
        LEFT JOIN Movies AS M
        USING(MovieID)
        LEFT JOIN Movie_Genre AS MG
        USING(MovieID)
        LEFT JOIN Genres AS G
        USING(GenreID)
    GROUP BY G.Genre) AS right_query

    USING(Genre)
    ORDER BY count1, count2 DESC
    """

    prepare_stmt3 = sa.sql.text(query_q3)    #prepare statement object f
or query_q3
    bound_stmt3 = prepare_stmt3.bindparams(j1=job1, c1_min=cast_min, c1_
max=cast_max, j2=job2, c2_min=cast2_min, c2_max=cast2_max) #bound statem
ent to bind named parameters for query_q3
    df3 = pd.read_sql_query(bound_stmt3, con=dbcon3)  #executes query an
d binded variables to dataframe

    return df3  #return dataframe

gender_movie_genre(connection, "actress", 1, 3, "actor", 1, 3)   #call g
ender_movie_genre function
```

Out[13]:

	Genre	count1	count2
0	Game-Show	3	2
1	Short	5	9
2	Talk-Show	10	20
3	Reality-TV	37	47
4	Adult	60	64
5	News	153	405
6	Western	998	873
7	Sport	1318	1589
8	War	2305	2235
9	Music	2551	3007
10	Animation	2898	2897
11	Musical	2960	2680
12	History	3284	3957
13	Biography	3574	4757
14	Sci-Fi	5056	4811
15	Fantasy	6273	5923
16	Mystery	7209	6966
17	Family	7761	7638
18	Documentary	8524	16265
19	Adventure	9225	8692
20	Crime	14272	12957
21	Horror	15460	14915
22	Thriller	18844	17700
23	Romance	22077	20589
24	Action	22810	19065
25	Comedy	46959	43862
26	Drama	93618	87641

V. Create a function that exports dataframes to csv files

In [14]:
```
def export_to_csv():
    """
    This function exports the pandas dataframe to csv files to be used f
or vizualization

    Parameters: None

    Return value: csv files of the corresponding data results
    """

    a = rating_genre_decade(connection, 1970, 1980, 1990, 2000, 2010, 26
1, 2621)  #assisgns variable a to rating_genre_decade function
    b = rating_genre_runtime(connection, 40, 300, 261, 2621)
#assigns variable b to rating_genre_runtime function
    c = gender_movie_genre(connection, "actress", 1, 3, "actor", 1, 3)
#assigns variable c to gender_movie_genre function

    a.to_csv(r'figures/question1_last_final.csv')   #executes pandas dat
aframe for question 1 to csv file
    b.to_csv(r'figures/question2_last_final.csv')   #executes pandas dat
aframe for question 2 to csv file
    c.to_csv(r'figures/question3_last_final.csv')   #executes pandas dat
aframe for question 3 to csv file
```

IV. Create a main function to execute all functions

In [15]:
```
def main():

    rating_genre_decade(connection, 1970, 1980, 1990, 2000, 2010, 261, 2
621)  #calls function for question 1

    rating_genre_runtime(connection, 40, 300, 261, 2621)
#calls function for question 2

    gender_movie_genre(connection, "actress", 1, 3, "actor", 1, 3)
#calls function for question 3

    export_to_csv()
#calls export_to_csv function


main()
```

In [16]:
```
try:
    connection.close()  #close connection
except:
    pass
del engine              #delete engine
```