Project by Faraz

Table of Content

1. Introduction:

In the ever-evolving entertainment industry, movie rental business strives to maintain its presence and relevance amidst the rise of streaming services. This scenario is about a Video Rental Store company. The database is called “sakila.”

2. Business Objective(s):

- The first objective is to hire a renowned actor or actress to appear in a series of advertisements that will air on local television and social media platforms. By enlisting a well-known figure, the movie rental business anticipates generating increased buzz and interest around its services, subsequently driving up foot traffic and revenue. Management is keen to identify the perfect candidate who not only possesses excellent acting abilities but also resonates with their target audience.

- The second objective revolves around the promotional strategy for the rental services, whether to focus on promoting long-term or short-term rentals. Long-term rentals could potentially result in a steadier revenue stream and greater customer loyalty, while short-term rentals might attract a wider range of customers seeking quick entertainment solutions. Based on the profitability we will steer our marketing towards one of the rental services.

3. Business Question(s):

1. Which actors and actresses are most popular given our rental history?
2. What is our average rental duration?
3. What is more profitable for us long or short-term rentals?

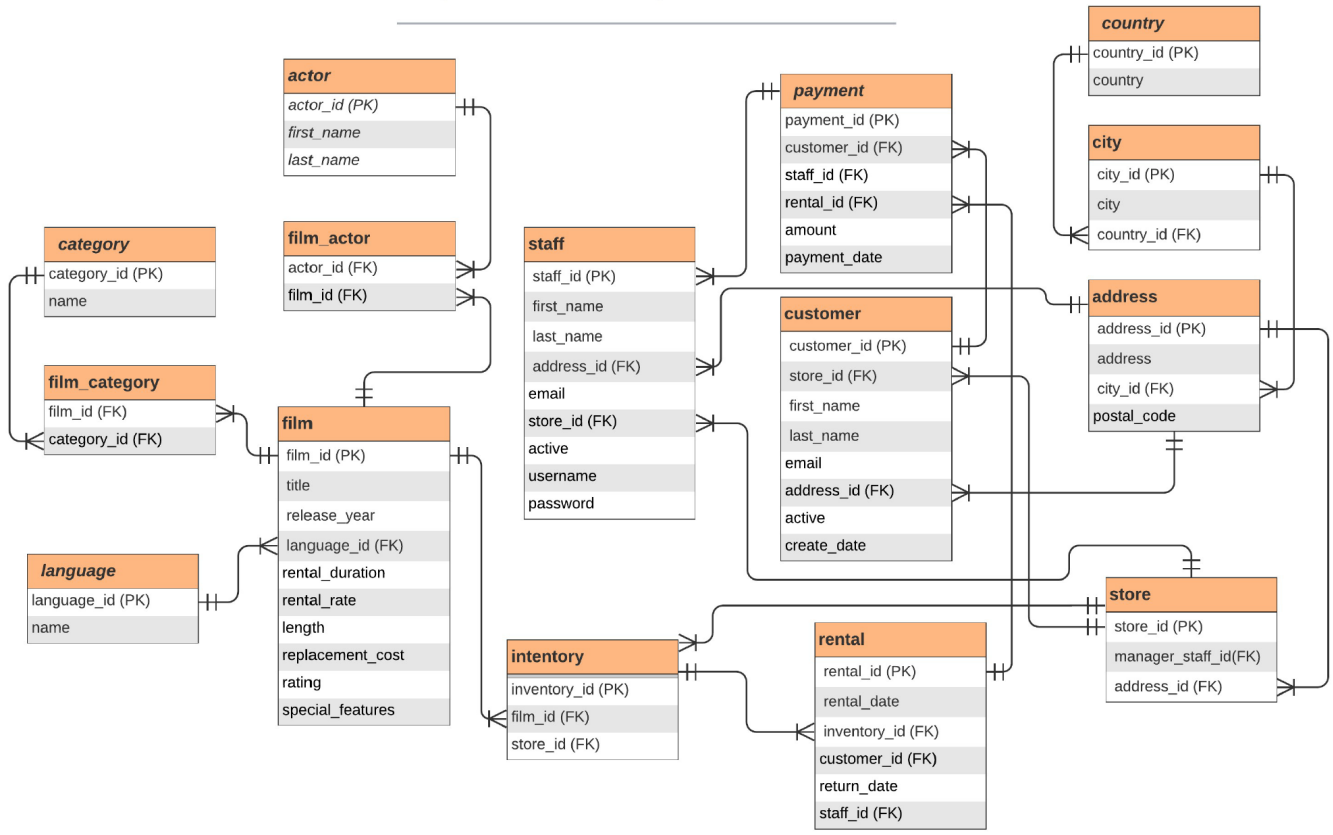
4. Data Exploration:

The data is stored in an SQLite relational database, accessible using SQL that is coded within a Python program. This database will be implemented using the SQLite database. The physical database is stored on the local computer and accessed using SQL commands via Python.

In [2]:
```
# Importing all libraries
import sqlite3
import pandas as pd
import pprint as pp
from pyspark.sql.functions import *
```

In [3]:
```
#importing database
conn = sqlite3.connect('sqlite-sakila.db')
#The Cursor object allows you to execute SQL statements and fetch results from the datab
cur=conn.cursor()
```

In [4]:
```
df=pd.DataFrame()
```



In [5]:
```
df=pd.read_sql("""
select * from rental
""", conn)
df.head(10)
```

Out[5]:

	rental_id	rental_date	inventory_id	customer_id	return_date	staff_id	last_update
0	1	2005-05-24 22:53:30.000	367	130	2005-05-26 22:04:30.000	1	2021-03-06 15:53:41
1	2	2005-05-24 22:54:33.000	1525	459	2005-05-28 19:40:33.000	1	2021-03-06 15:53:41
2	3	2005-05-24 23:03:39.000	1711	408	2005-06-01 22:12:39.000	1	2021-03-06 15:53:41
3	4	2005-05-24 23:04:41.000	2452	333	2005-06-03 01:43:41.000	2	2021-03-06 15:53:41
4	5	2005-05-24 23:05:21.000	2079	222	2005-06-02 04:33:21.000	1	2021-03-06 15:53:41
5	6	2005-05-24 23:08:07.000	2792	549	2005-05-27 01:32:07.000	1	2021-03-06 15:53:41
6	7	2005-05-24 23:11:53.000	3995	269	2005-05-29 20:34:53.000	2	2021-03-06 15:53:41
7	8	2005-05-24 23:31:46.000	2346	239	2005-05-27 23:33:46.000	2	2021-03-06 15:53:41
8	9	2005-05-25 00:00:40.000	2580	126	2005-05-28 00:22:40.000	1	2021-03-06 15:53:41
9	10	2005-05-25 00:02:21.000	1824	399	2005-05-31 22:44:21.000	2	2021-03-06 15:53:41

In [7]:
```
df=pd.read_sql("""
select * from actor
""", conn)
df.head(10)
```

Out[7]:

	actor_id	first_name	last_name	last_update
0	1	PENELOPE	GUINESS	2021-03-06 15:51:59
1	2	NICK	WAHLBERG	2021-03-06 15:51:59
2	3	ED	CHASE	2021-03-06 15:51:59
3	4	JENNIFER	DAVIS	2021-03-06 15:51:59
4	5	JOHNNY	LOLLOBRIGIDA	2021-03-06 15:51:59
5	6	BETTE	NICHOLSON	2021-03-06 15:51:59
6	7	GRACE	MOSTEL	2021-03-06 15:51:59
7	8	MATTHEW	JOHANSSON	2021-03-06 15:51:59
8	9	JOE	SWANK	2021-03-06 15:51:59
9	10	CHRISTIAN	GABLE	2021-03-06 15:51:59

In [8]:
```
df=pd.read_sql("""
select * from film
""", conn)
df.head(10)
```

Out[8]:

	film_id	title	description	release_year	language_id	original_language_id	rental_duration	rental_r
0	1	ACADEMY DINOSAUR	A Epic Drama of a Feminist And a Mad Scientist...	2006	1	None	6	0
1	2	ACE GOLDFINGER	A Astounding Epistle of a Database Administrat...	2006	1	None	3	4
2	3	ADAPTATION HOLES	A Astounding Reflection of a Lumberjack And a ...	2006	1	None	7	2
3	4	AFFAIR PREJUDICE	A Fanciful Documentary of a Frisbee And a Lumb...	2006	1	None	5	2
4	5	AFRICAN EGG	A Fast-Paced Documentary of a Pastry Chef And ...	2006	1	None	6	2
5	6	AGENT TRUMAN	A Intrepid Panorama of a Robot And a Boy who m...	2006	1	None	3	2
6	7	AIRPLANE SIERRA	A Touching Saga of a Hunter And a Butler who m...	2006	1	None	6	4
7	8	AIRPORT POLLOCK	A Epic Tale of a Moose And a Girl who must Con...	2006	1	None	6	4
8	9	ALABAMA DEVIL	A Thoughtful Panorama of a Database Administra...	2006	1	None	3	2
9	10	ALADDIN CALENDAR	A Action-Packed Tale of a Man And a Lumberjack...	2006	1	None	6	4

5. Database Queries & Results:

- ### 5.1 Query for first business question

In [46]:

```
query = """
SELECT
  actor_id
  ,first_name
  ,last_name
  ,COUNT(DISTINCT film_id) AS number_of_films
FROM film_actor
```

```
  INNER JOIN actor USING(actor_id)
GROUP BY actor_id
ORDER BY 4 DESC  -- number_of_films
LIMIT 10
"""
df = pd.read_sql(query, conn)
df
```

Out[46]:

	actor_id	first_name	last_name	number_of_films
0	107	GINA	DEGENERES	42
1	102	WALTER	TORN	41
2	198	MARY	KEITEL	40
3	181	MATTHEW	CARREY	39
4	23	SANDRA	KILMER	37
5	81	SCARLETT	DAMON	36
6	13	UMA	WOOD	35
7	37	VAL	BOLGER	35
8	60	HENRY	BERRY	35
9	106	GROUCHO	DUNST	35

This code is querying a database to retrieve information about actors and their film credits. Specifically, the query is selecting the actor ID, first name, and last name from the actor table and the number of distinct film IDs from the film_actor table. The INNER JOIN keyword is used to join the film_actor and actor tables on the actor_id column. The GROUP BY clause is used to group the results by actor ID, and the COUNT function is used to count the number of distinct film IDs for each actor. The results are then sorted in descending order by the number of films and limited to the top 10 results using the ORDER BY and LIMIT keywords, respectively. Finally, the code uses pd.read_sql() to execute the query and retrieve the results as a pandas DataFrame, which is then assigned to the variable df.

- ### 5.2 Query for second business question

In [39]:

```
query = """
SELECT AVG(julianday(return_date) - julianday(rental_date)) AS avg_rental_time
FROM rental
"""

df = pd.read_sql(query, conn)
df.head(10)
```

Out[39]:

	avg_rental_time
0	5.02533

This code is querying a database to calculate the average rental time for movie rentals. Specifically, the query is selecting the average value of the difference between the return_date and rental_date columns in the rental table. The julianday() function is used to convert the dates to Julian day numbers, which represent the number of days since November 24, 4714 BC. This makes it easier to perform calculations with dates, such as finding the difference between two dates. The results are then returned as a single row with a single column named avg_rental_time. Finally, the code uses pd.read_sql() to execute the query and retrieve the result as a pandas DataFrame, which is then assigned to the variable df.

- ### Query for third business question

In [37]:
```
query = """
SELECT SUM(amount)
FROM payment
  INNER JOIN rental USING(rental_id)
WHERE julianday(return_date) - julianday(rental_date) >= 5.02533;
"""

df = pd.read_sql(query, conn)
df.head(10)
```

Out[37]:

	SUM(amount)
0	42488.77

In [38]:
```
query = """
SELECT SUM(amount)
FROM payment
  INNER JOIN rental USING(rental_id)
WHERE julianday(return_date) - julianday(rental_date) < 5.02533;
"""

df = pd.read_sql(query, conn)
df.head(10)
```

Out[38]:

	SUM(amount)
0	24399.62

This code calculates the rental period in days by using the julianday function to convert each date to a Julian day number and then subtracting the Julian day number of the rental date from the Julian day number of the return date. The resulting time difference is then compared to the rental period threshold of 5.0252 days. The SUM function calculates the total amount paid for rental transactions that meet the rental period criteria. The output is stored in a Pandas DataFrame called df and is displayed using the head method.

The store makes more money from long term rentals

Total short term rental and long term rental = 42488.77+24399.62 = 66,888.39 To check if our findings are correct we will calculate the sum of total payments

In [36]:
```
query = """
SELECT SUM(amount)
FROM payment
"""
```

```
df = pd.read_sql(query, conn)
df.head(10)
```

Out[36]:

	SUM(amount)
0	67416.51

Finding the missing amount we assume in some cases the payment has been made in advanced but the rental is not return to confirm this we can run following queries

In [41]:
```
query = """
select SUM(amount)
FROM payment
  INNER JOIN rental USING(rental_id)
WHERE rental_date IS NULL OR return_date IS NULL
"""

df = pd.read_sql(query, conn)
df.head(10)
```

Out[41]:

	SUM(amount)
0	518.17

In [42]:
```
query = """
SELECT SUM(amount)
FROM payment
WHERE rental_id IS NULL
;
"""

df = pd.read_sql(query, conn)
df.head(10)
```

Out[42]:

	SUM(amount)
0	9.95

To find if we have the right amount we will add out missing payments in total
66,888.39+9.95+518.17=67416.51

This totals equals to our actual total.

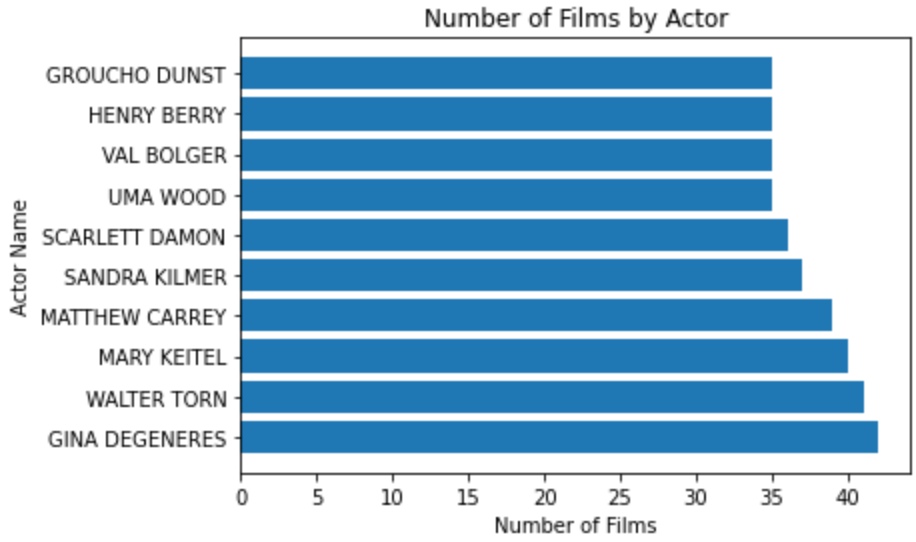
6. Analysis:

Visualizing top actors/acteresses

In [44]:
```
# Importing all libraries
import os as os
import matplotlib.pyplot as plt # Matplot library for visulization
import seaborn as sns
import numpy as np
```

In [47]:
```
# Create a bar chart using matplotlib
plt.barh(df['first_name'] + ' ' + df['last_name'], df['number_of_films'])

# Add labels and title
lumber of Films')
```

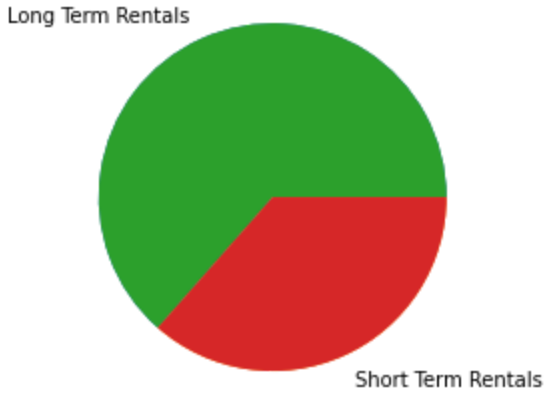
```
plt.ylabel('Actor Name')
plt.title('Number of Films by Actor')

# Show the plot

plt.show()
```



Visualizing total market share of short or long term rentals

In [59]:

```
query = """
SELECT SUM(amount)
FROM payment
  INNER JOIN rental USING(rental_id)
WHERE julianday(return_date) - julianday(rental_date) >= 5.02533;
"""

df1 = pd.read_sql(query, conn)
df1 = df1.iloc[0, 0]

query = """
SELECT SUM(amount)
FROM payment
  INNER JOIN rental USING(rental_id)
WHERE julianday(return_date) - julianday(rental_date) < 5.02533;
"""

df2 = pd.read_sql(query, conn)
df2 = df2.iloc[0, 0]
# library
import matplotlib.pyplot as plt

# create random data
names='Long Term Rentals', 'Short Term Rentals',
values=[df1,df2]

# Create a pieplot
plt.pie(values)

# Label distance: gives the space between labels and the center of the pie
plt.pie(values, labels=names, labeldistance=1.15);
plt.show();
```



7. Decisions:

- Based on the data, it appears that Gina Degeneres (actor_id 107) has the most film credits, followed by Walter Torn (actor_id 102) and Mary Keitel (actor_id 198). If the store is looking to hire an actor or actress for their ads, I would suggest considering these three individuals as potential candidates.
- Long terms rentals are generating more profit for us we should steer our marketing towards long term rentals which is avg of 5 days or more.

8. Conclusions:

- In conclusion, to boost interest and drive foot traffic to the movie rental business, it is crucial to hire a renowned actor or actress for a series of advertisements on local television and social media platforms. Based on the data available, the top candidates to consider for this role are Gina Degeneres (actor_id 107), Walter Torn (actor_id 102), and Mary Keitel (actor_id 198), who have the most film credits and are likely to resonate with the target audience. By selecting the right candidate with excellent acting abilities and a strong connection to the audience, the movie rental business can expect increased buzz and revenue growth.
- Long-term rentals offer a unique opportunity for our business to establish a more stable revenue stream and foster enduring customer loyalty. By focusing on this segment, we can generate increased profitability and build lasting relationships with our clientele. As a strategic decision, we should shift our marketing efforts towards promoting long-term rentals, accentuating the benefits for both our customers and our organization. This targeted approach will enable us to capitalize on the potential for sustained growth and enhance our competitive advantage in the marketplace.