In [1]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
# Install packages
!pip3 install pymysql
!pip3 install plotly
!pip3 install cufflinks
```

```
Requirement already satisfied: pymysql in c:\users\60357\anaconda3\lib\site-packages (1.0.2)
Requirement already satisfied: plotly in c:\users\60357\anaconda3\lib\site-packages (5.3.1)
Requirement already satisfied: tenacity>=6.2.0 in c:\users\60357\anaconda3\lib\site-packages (from plotly) (8.0.1)
Requirement already satisfied: six in c:\users\60357\anaconda3\lib\site-packages (from plotly) (1.15.0)
Requirement already satisfied: cufflinks in c:\users\60357\anaconda3\lib\site-packages (0.17.3)
Requirement already satisfied: pandas>=0.19.2 in c:\users\60357\anaconda3\lib\site-packages (from cufflinks) (1.1.3)
Requirement already satisfied: ipython>=5.3.0 in c:\users\60357\anaconda3\lib\site-packages (from cufflinks) (7.19.0)
Requirement already satisfied: colorlover>=0.2.1 in c:\users\60357\anaconda3\lib\site-packages (from cufflinks) (0.3.0)
Requirement already satisfied: plotly>=4.1.1 in c:\users\60357\anaconda3\lib\site-packages (from cufflinks) (5.3.1)
Requirement already satisfied: six>=1.9.0 in c:\users\60357\anaconda3\lib\site-packages (from cufflinks) (1.15.0)
Requirement already satisfied: ipywidgets>=7.0.0 in c:\users\60357\anaconda3\lib\site-packages (from cufflinks) (7.5.1)
Requirement already satisfied: setuptools>=34.4.1 in c:\users\60357\anaconda3\lib\site-packages (from cufflinks) (50.3.1.post20201107)
Requirement already satisfied: numpy>=1.9.2 in c:\users\60357\anaconda3\lib\site-packages (from cufflinks) (1.19.2)
Requirement already satisfied: pytz>=2017.2 in c:\users\60357\anaconda3\lib\site-packages (from pandas>=0.19.2->cufflinks) (2020.1)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\60357\anaconda3\lib\site-packages (from pandas>=0.19.2->cufflinks) (2.8.1)
Requirement already satisfied: colorama; sys_platform == "win32" in c:\users\60357\anaconda3\lib\site-packages (from ipython>=5.3.0->cufflinks) (0.4.4)
Requirement already satisfied: backcall in c:\users\60357\anaconda3\lib\site-packages (from ipython>=5.3.0->cufflinks) (0.2.0)
Requirement already satisfied: traitlets>=4.2 in c:\users\60357\anaconda3\lib\site-packages (from ipython>=5.3.0->cufflinks) (5.0.5)
Requirement already satisfied: decorator in c:\users\60357\anaconda3\lib\site-packages (from ipython>=5.3.0->cufflinks) (4.4.2)
Requirement already satisfied: pickleshare in c:\users\60357\anaconda3\lib\site-packages (from ipython>=5.3.0->cufflinks) (0.7.5)
```

In [3]:
```
#import statements
import pymysql
import pandas as pd
```

Part B

Question 1

In [ ]:
```
######## QUESTION 1 ######## - { 10 Points }
```

In [5]:
```
# Create a function for easily querying data.
def qry(sql):

    # Open database connection
    connection = pymysql.connect(host="localhost",user="root",password="rootroot",database="sakila" )
    df = pd.read_sql(sql, connection)

    # disconnect from server
    connection.close()

    # return data.
    return df
```

In [6]:
```
#!/usr/bin/python3

# Open database connection
db = pymysql.connect(host="localhost",user="root",password="rootroot",database="sakila" )

# prepare a cursor object using cursor() method
cursor = db.cursor()


# a) Show the list of databases.
Q1_a = "SHOW databases;"
# execute SQL query using execute() method.
cursor.execute(Q1_a)

# Fetch a single row using fetchone() method.
Q1_a = cursor.fetchall()
print(Q1_a)

# disconnect from server
db.close()
```

```
(('classicmodels',), ('information_schema',), ('mysql',), ('performance_schema',), ('sakila',), ('sys',), ('world',))
```

In [7]:
```
# b) Select sakila database.
# select database
#!/usr/bin/python3

# Open database connection
db = pymysql.connect(host="localhost",user="root",password="rootroot",database="sakila" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

Q1_b = "USE sakila;"
# execute SQL query using execute() method.
cursor.execute(Q1_b)


# disconnect from server
db.close()
```

In [37]:
```
# c) Show all tables in the sakila database.
Q1_c = "SHOW tables;"
qry(Q1_c)
```

Out[37]:

	Tables_in_sakila
0	actor
1	actor_info
2	address
3	category
4	city
5	country
6	customer
7	customer_list
8	film
9	film_actor
10	film_category
11	film_list
12	film_text
13	inventory
14	language
15	nicer_but_slower_film_list
16	payment
17	payment_types
18	rental
19	sales_by_film_category
20	sales_by_store
21	staff
22	staff_list
23	store

In [30]:
```
# d) Show each of the columns along with their data types for the actor table.
Q1_d = "DESCRIBE actor;"
qry(Q1_d)
```

Out[30]:

	Field	Type	Null	Key	Default	Extra
0	actor_id	smallint unsigned	NO	PRI	None	auto_increment
1	first_name	varchar(45)	NO		None	
2	last_name	varchar(45)	NO	MUL	None	
3	last_update	timestamp	NO		CURRENT_TIMESTAMP	DEFAULT_GENERATED on update CURRENT_TIMESTAMP

In [31]:
```
# e) Show the total number of records in the actor table.
Q1_e = "SELECT COUNT(*) \
FROM actor;"
qry(Q1_e)
```

Out[31]:

	COUNT(*)
0	200

In [32]:
```
# f) What is the first name and last name of all the actors in the actor table ?
Q1_f = "SELECT first_name, last_name \
FROM actor;"
qry(Q1_f)
```

Out[32]:

	first_name	last_name
0	PENELOPE	GUINESS
1	NICK	WAHLBERG
2	ED	CHASE
3	JENNIFER	DAVIS
4	JOHNNY	LOLLOBRIGIDA
...	...	...
195	BELA	WALKEN
196	REESE	WEST
197	MARY	KEITEL
198	JULIA	FAWCETT
199	THORA	TEMPLE

200 rows × 2 columns

In [42]:
```
# g) Insert your first name (in first name column) and middle initial ( in the last name
# column ) into the actors table.
#!/usr/bin/python3

# Open database connection
db = pymysql.connect(host="localhost",user="root",password="rootroot",database="sakila" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

Q1_g = "INSERT INTO `actor` \
        (`actor_id`, `first_name`, `last_name`, `last_update`) \
        VALUES \
        (null, 'Ruoxin', 'S', DEFAULT);"

# execute SQL query using execute() method.
cursor.execute(Q1_g)


# disconnect from server
db.close()
```

In [44]:
```
# h) Update your middle initial with your last name (in last name column) in the actors
# table.
#!/usr/bin/python3

# Open database connection
db = pymysql.connect(host="localhost",user="root",password="rootroot",database="sakila" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

Q1_h = "UPDATE actor \
        SET \
        last_name = 'Shi' \
        WHERE \
        last_name = 'S';"

# execute SQL query using execute() method.
cursor.execute(Q1_h)

# disconnect from server
db.close()
```

In [45]:
```
# i) Delete the new record from the actor table where the first name and last name matches
# yours.
#!/usr/bin/python3

# Open database connection
db = pymysql.connect(host="localhost",user="root",password="rootroot",database="sakila" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

Q1_i = "DELETE FROM actor \
        WHERE \
        first_name = 'Ruoxin' \
        AND last_name = 'Shi';"

# execute SQL query using execute() method.
cursor.execute(Q1_i)

# disconnect from server
db.close()
```

In [52]:
```
# j) Create a table payment_type with the following specifications and appropriate data types
# Table Name : “Payment_type”
# Primary Key: "payment_type_id”
# Column: “Type”
# Insert following rows in to the table:
# 1, “Credit Card” ; 2, “Cash”; 3, “Paypal” ; 4 , “Cheque”

#!/usr/bin/python3

# Open database connection
db = pymysql.connect(host="localhost",user="root",password="rootroot",database="sakila" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

q = "DROP TABLE IF EXISTS `Payment_type`"

Q1_j_1 = "CREATE TABLE `Payment_type` ( \
          `payment_type_id` INT(10) NOT NULL, \
          `type` VARCHAR(50) NOT NULL, \
          PRIMARY KEY (`payment_type_id`) \
            );"


Q1_j_2 = "INSERT INTO `Payment_type` \
        (`payment_type_id`, `type`) \
        VALUES \
        (1, 'Credit Card'), \
        (2, 'Cash'), \
        (3, 'Paypal'), \
        (4, 'Cheque');"


# execute SQL query using execute() method.
cursor.execute(q)
cursor.execute(Q1_j_1)
cursor.execute(Q1_j_2)

# disconnect from server
db.close()
```

In [53]:
```
# k) Rename table payment_type to payment_types.
Q1_k = "RENAME TABLE Payment_type TO payment_types; "

#!/usr/bin/python3
# Open database connection
db = pymysql.connect(host="localhost",user="root",password="rootroot",database="sakila" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# execute SQL query using execute() method.
cursor.execute(Q1_k)

# disconnect from server
db.close()
```

In [54]:
```
# l) Drop the table payment_types
Q1_l = "DROP TABLE payment_types;"

#!/usr/bin/python3
# Open database connection
db = pymysql.connect(host="localhost",user="root",password="rootroot",database="sakila" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# execute SQL query using execute() method.
cursor.execute(Q1_l)

# disconnect from server
db.close()
```

Question 2

In [55]:
```
######## QUESTION 2 ######## - { 10 Points }
# a) List all the movies ( title & description ) that are rated PG-13 ?
# select * from film;
Q2_a = "SELECT title, description \
        FROM film \
        WHERE rating = 'PG-13';"
qry(Q2_a)
```

Out[55]:

	title	description
0	AIRPLANE SIERRA	A Touching Saga of a Hunter And a Butler who m...
1	ALABAMA DEVIL	A Thoughtful Panorama of a Database Administra...
2	ALTER VICTORY	A Thoughtful Drama of a Composer And a Feminis...
3	ANTHEM LUKE	A Touching Panorama of a Waitress And a Woman ...
4	APOLLO TEEN	A Action-Packed Reflection of a Crocodile And ...
...	...	...
218	WHALE BIKINI	A Intrepid Story of a Pastry Chef And a Databa...
219	WHISPERER GIANT	A Intrepid Story of a Dentist And a Hunter who...
220	WORLD LEATHERNECKS	A Unbelieveable Tale of a Pioneer And a Astron...
221	WRONG BEHAVIOR	A Emotional Saga of a Crocodile And a Sumo Wre...
222	WYOMING STORM	A Awe-Inspiring Panorama of a Robot And a Boat...

223 rows × 2 columns

In [56]:
```
# b) List all movies that are either PG OR PG-13 using IN operator ?
Q2_b = "SELECT * \
        FROM film \
        WHERE rating IN ('PG', 'PG-13');"
qry(Q2_b)
```

Out[56]:

	film_id	title	description	release_year	language_id	original_language_id	rental_duration	rental_rate	length	replacement_cost	rating	special_features	last_update
0	1	ACADEMY DINOSAUR	A Epic Drama of a Feminist And a Mad Scientist...	2006	6	None	6	0.99	86	20.99	PG	Deleted Scenes,Behind the Scenes	2021-10-20 19:55:57
1	6	AGENT TRUMAN	A Intrepid Panorama of a Robot And a Boy who m...	2006	1	None	3	2.99	169	17.99	PG	Deleted Scenes	2006-02-15 05:03:42
2	7	AIRPLANE SIERRA	A Touching Saga of a Hunter And a Butler who m...	2006	1	None	6	4.99	62	28.99	PG-13	Trailers,Deleted Scenes	2006-02-15 05:03:42
3	9	ALABAMA DEVIL	A Thoughtful Panorama of a Database Administra...	2006	1	None	3	2.99	114	21.99	PG-13	Trailers,Deleted Scenes	2006-02-15 05:03:42
4	12	ALASKA PHANTOM	A Fanciful Saga of a Hunter And a Pastry Chef ...	2006	1	None	6	0.99	136	22.99	PG	Commentaries,Deleted Scenes	2006-02-15 05:03:42
...	...	...	...	...	...	...	...	...	...	...	...	...	...
412	987	WORDS HUNTER	A Action-Packed Reflection of a Composer And a...	2006	1	None	3	2.99	116	13.99	PG	Trailers,Commentaries,Deleted Scenes	2006-02-15 05:03:42
413	990	WORLD LEATHERNECKS	A Unbelieveable Tale of a Pioneer And a Astron...	2006	1	None	3	0.99	171	13.99	PG-13	Trailers,Behind the Scenes	2006-02-15 05:03:42
414	991	WORST BANGER	A Thrilling Drama of a Madman And a Dentist wh...	2006	1	None	4	2.99	185	26.99	PG	Deleted Scenes,Behind the Scenes	2006-02-15 05:03:42
415	993	WRONG BEHAVIOR	A Emotional Saga of a Crocodile And a Sumo Wre...	2006	1	None	6	2.99	178	10.99	PG-13	Trailers,Behind the Scenes	2006-02-15 05:03:42
416	994	WYOMING STORM	A Awe-Inspiring Panorama of a Robot And a Boat...	2006	1	None	6	4.99	100	29.99	PG-13	Deleted Scenes	2006-02-15 05:03:42

417 rows × 13 columns

In [59]:
```
# c) Report all payments greater than and equal to 2$ and Less than equal to 7$ ?
# Note : write 2 separate queries conditional operator and BETWEEN keyword
Q2_c_1 = "SELECT * \
        FROM payment \
        WHERE amount >= 2 AND amount <= 7;"

qry(Q2_c_1)
```

Out[59]:

	payment_id	customer_id	staff_id	rental_id	amount	payment_date	last_update
0	1	1	1	76.0	2.99	2005-05-25 11:30:37	2006-02-15 22:12:30
1	3	1	1	1185.0	5.99	2005-06-15 00:54:12	2006-02-15 22:12:30
2	6	1	1	1725.0	4.99	2005-06-16 15:18:57	2006-02-15 22:12:30
3	7	1	1	2308.0	4.99	2005-06-18 08:41:48	2006-02-15 22:12:30
4	9	1	1	3284.0	3.99	2005-06-21 06:24:45	2006-02-15 22:12:30
...	...	...	...	...	...	...	...
10868	16042	599	2	9679.0	2.99	2005-07-31 06:41:19	2006-02-15 22:24:11
10869	16043	599	2	11522.0	3.99	2005-08-17 00:05:05	2006-02-15 22:24:11
10870	16045	599	1	14599.0	4.99	2005-08-21 17:43:42	2006-02-15 22:24:12
10871	16048	599	2	15719.0	2.99	2005-08-23 11:08:46	2006-02-15 22:24:13
10872	16049	599	2	15725.0	2.99	2005-08-23 11:25:00	2006-02-15 22:24:13

10873 rows × 7 columns

In [60]:
```
Q2_c_2 = "SELECT * \
        FROM payment \
        WHERE amount BETWEEN 2 and 7;"

qry(Q2_c_2)
```

Out[60]:

	payment_id	customer_id	staff_id	rental_id	amount	payment_date	last_update
0	1	1	1	76.0	2.99	2005-05-25 11:30:37	2006-02-15 22:12:30
1	3	1	1	1185.0	5.99	2005-06-15 00:54:12	2006-02-15 22:12:30
2	6	1	1	1725.0	4.99	2005-06-16 15:18:57	2006-02-15 22:12:30
3	7	1	1	2308.0	4.99	2005-06-18 08:41:48	2006-02-15 22:12:30
4	9	1	1	3284.0	3.99	2005-06-21 06:24:45	2006-02-15 22:12:30
...	...	...	...	...	...	...	...
10868	16042	599	2	9679.0	2.99	2005-07-31 06:41:19	2006-02-15 22:24:11
10869	16043	599	2	11522.0	3.99	2005-08-17 00:05:05	2006-02-15 22:24:11
10870	16045	599	1	14599.0	4.99	2005-08-21 17:43:42	2006-02-15 22:24:12
10871	16048	599	2	15719.0	2.99	2005-08-23 11:08:46	2006-02-15 22:24:13
10872	16049	599	2	15725.0	2.99	2005-08-23 11:25:00	2006-02-15 22:24:13

10873 rows × 7 columns

In [63]:
```
# d) List all addresses that have phone number that contain digits 589. A separate query
# for phone numbers that start with 140, and a third query that ends with 589
# Note : write 3 different queries

# contain 589
Q2_d_1 = "SELECT * \
        FROM address \
        WHERE phone LIKE '%589%';"
qry(Q2_d_1)
```

Out[63]:

	address_id	address	address2	district	city_id	postal_code	phone	location	last_update
0	4	1411 Lillydale Drive	None	QLD	576		6172235589	b'\x00\x00\x00\x00\x01\x01\x00\x00\x00[\r\xe44...	2014-09-25 22:30:09
1	153	782 Mosul Street		Massachusetts	94	25545	885899703621	b'\x00\x00\x00\x00\x01\x01\x00\x00\x00\xe9\xc4...	2014-09-25 22:33:46
2	333	1860 Taguig Loop		West Java	119	59550	38158430589	b'\x00\x00\x00\x00\x01\x01\x00\x00\x00B\xac\xa...	2014-09-25 22:31:32
3	388	368 Hunuco Boulevard		Namibe	360	17165	106439158941	b'\x00\x00\x00\x00\x01\x01\x00\x00\x00\xb5\x85...	2014-09-25 22:30:03
4	492	185 Mannheim Lane		Stavropol	408	23661	589377568313	b'\x00\x00\x00\x00\x01\x01\x00\x00\x003>\x82\x...	2014-09-25 22:32:56
5	504	1 Valle de Santiago Avenue		Apulia	93	86208	465897838272	b'\x00\x00\x00\x00\x01\x01\x00\x00\x00\xca\xfe...	2014-09-25 22:31:43

In [64]:
```
# start with 140
Q2_d_2 = "SELECT * \
        FROM address \
        WHERE phone LIKE '140%';"
qry(Q2_d_2)
```

Out[64]:

	address_id	address	address2	district	city_id	postal_code	phone	location	last_update
0	3	23 Workhaven Lane	None	Alberta	300		14033335568	b'\x00\x00\x00\x00\x01\x01\x00\x00\x00\xcd\xc4...	2014-09-25 22:30:27

In [65]:
```
# end with 589
Q2_d_3 = "SELECT * \
        FROM address \
        WHERE phone LIKE '%589';"
qry(Q2_d_3)
```

Out[65]:

	address_id	address	address2	district	city_id	postal_code	phone	location	last_update
0	4	1411 Lillydale Drive	None	QLD	576		6172235589	b'\x00\x00\x00\x00\x01\x01\x00\x00\x00[\r\xe44...	2014-09-25 22:30:09
1	333	1860 Taguig Loop		West Java	119	59550	38158430589	b'\x00\x00\x00\x00\x01\x01\x00\x00\x00B\xac\xa...	2014-09-25 22:31:32

In [66]:
```
# e) List all staff members ( first name, last name, email ) whose password is NULL ?
Q2_e = "SELECT first_name, last_name, email \
        FROM staff \
        WHERE password IS NULL;"
qry(Q2_e)
```

Out[66]:

	first_name	last_name	email
0	Jon	Stephens	Jon.Stephens@sakilastaff.com

In [67]:
```
# f) Select all films that have title names like ZOO and rental duration greater than or equal
# to 4
Q2_f = "SELECT * \
        FROM film \
        WHERE title LIKE '%ZOO%' AND rental_duration >= 4;"
qry(Q2_f)
```

Out[67]:

	film_id	title	description	release_year	language_id	original_language_id	rental_duration	rental_rate	length	replacement_cost	rating	special_features	last_update
0	568	MEMENTO ZOOLANDER	A Touching Epistle of a Squirrel And a Explore...	2006	1	None	4	4.99	77	11.99	NC-17	Behind the Scenes	2006-02-15 05:03:42
1	924	UNFORGIVEN ZOOLANDER	A Taut Epistle of a Monkey And a Sumo Wrestler...	2006	1	None	7	0.99	129	15.99	PG	Trailers,Commentaries,Behind the Scenes	2006-02-15 05:03:42
2	999	ZOOLANDER FICTION	A Fateful Reflection of a Waitress And a Boat ...	2006	1	None	5	2.99	101	28.99	R	Trailers,Deleted Scenes	2006-02-15 05:03:42

In [68]:
```
# g) What is the cost of renting the movie ACADEMY DINOSAUR for 2 weeks ?
# Note : use of column alias and watch for rental_duration value
Q2_g = "SELECT *, rental_rate/rental_duration * 14 as rate_per_2week \
        FROM film \
        WHERE title = 'ACADEMY DINOSAUR';"
qry(Q2_g)

# The cost is 2.31
```

Out[68]:

	film_id	title	description	release_year	language_id	original_language_id	rental_duration	rental_rate	length	replacement_cost	rating	special_features	last_update	rate_per_2week
0	1	ACADEMY DINOSAUR	A Epic Drama of a Feminist And a Mad Scientist...	2006	6	None	6	0.99	86	20.99	PG	Deleted Scenes,Behind the Scenes	2021-10-20 19:55:57	2.31

In [69]:
```
# h) List all unique districts where the customers, staff, and stores are located
# Note : check for NOT NULL values
Q2_h = "SELECT DISTINCT district \
        FROM address \
        WHERE district is NOT NULL;"
qry(Q2_h)
```

Out[69]:

	district
0	Alberta
1	QLD
2	Nagasaki
3	California
4	Attika
...	...
373	al-Manama
374	Fejr
375	West Greece
376	Piura
377	Vaud

378 rows × 1 columns

In [70]:
```
# i) List the top 10 newest customers across all stores based on customer_id
Q2_i = "SELECT * \
        FROM customer \
        ORDER BY customer_id DESC \
        LIMIT 10;"
qry(Q2_i)
```

Out[70]:

	customer_id	store_id	first_name	last_name	email	address_id	active	create_date	last_update
0	599	2	AUSTIN	CINTRON	AUSTIN.CINTRON@sakilacustomer.org	605	1	2006-02-14 22:04:37	2006-02-15 04:57:20
1	598	1	WADE	DELVALLE	WADE.DELVALLE@sakilacustomer.org	604	1	2006-02-14 22:04:37	2006-02-15 04:57:20
2	597	1	FREDDIE	DUGGAN	FREDDIE.DUGGAN@sakilacustomer.org	603	1	2006-02-14 22:04:37	2006-02-15 04:57:20
3	596	1	ENRIQUE	FORSYTHE	ENRIQUE.FORSYTHE@sakilacustomer.org	602	1	2006-02-14 22:04:37	2006-02-15 04:57:20
4	595	1	TERRENCE	GUNDERSON	TERRENCE.GUNDERSON@sakilacustomer.org	601	1	2006-02-14 22:04:37	2006-02-15 04:57:20
5	594	1	EDUARDO	HIATT	EDUARDO.HIATT@sakilacustomer.org	600	1	2006-02-14 22:04:37	2006-02-15 04:57:20
6	593	2	RENE	MCALISTER	RENE.MCALISTER@sakilacustomer.org	599	1	2006-02-14 22:04:37	2006-02-15 04:57:20
7	592	1	TERRANCE	ROUSH	TERRANCE.ROUSH@sakilacustomer.org	598	0	2006-02-14 22:04:37	2006-02-15 04:57:20
8	591	1	KENT	ARSENAULT	KENT.ARSENAULT@sakilacustomer.org	597	1	2006-02-14 22:04:37	2006-02-15 04:57:20
9	590	2	SETH	HANNON	SETH.HANNON@sakilacustomer.org	596	1	2006-02-14 22:04:37	2006-02-15 04:57:20

Question 3

In [71]:
```
######## QUESTION 3 ########
# a) Show total number of movies
Q3_a = "SELECT COUNT(DISTINCT (film_id)) \
        FROM film;"
qry(Q3_a)
```

Out[71]:

	COUNT(DISTINCT (film_id))
0	1000

In [72]:
```
# b) What is the minimum payment received and max payment received across all
# transactions ?
Q3_b = "SELECT MIN(amount), MAX(amount) \
        FROM payment;"
qry(Q3_b)
```

Out[72]:

	MIN(amount)	MAX(amount)
0	0.0	11.99

In [73]:
```
# c) Number of customers that rented movies between Feb-2005 & May-2005 ( based on
# paymentDate ).
Q3_c = "SELECT COUNT(customer_id) \
        FROM payment \
        WHERE payment_date BETWEEN '2005-02-01' AND '2005-05-31';"
qry(Q3_c)
```

Out[73]:

	COUNT(customer_id)
0	994

In [74]:
```
# d) List all movies where replacement_cost is greater than 15$ or rental_duration is
# between 6 & 10 days
Q3_d = "SELECT * \
        FROM film \
        WHERE replacement_cost > 15 or (rental_duration BETWEEN 6 AND 10);"
qry(Q3_d)
```

Out[74]:

	film_id	title	description	release_year	language_id	original_language_id	rental_duration	rental_rate	length	replacement_cost	rating	special_features	last_update
0	1	ACADEMY DINOSAUR	A Epic Drama of a Feminist And a Mad Scientist...	2006	6	None	6	0.99	86	20.99	PG	Deleted Scenes,Behind the Scenes	2021-10-20 19:55:57
1	3	ADAPTATION HOLES	A Astounding Reflection of a Lumberjack And a ...	2006	1	None	7	2.99	50	18.99	NC-17	Trailers,Deleted Scenes	2006-02-15 05:03:42
2	4	AFFAIR PREJUDICE	A Fanciful Documentary of a Frisbee And a Lumb...	2006	1	None	5	2.99	117	26.99	G	Commentaries,Behind the Scenes	2006-02-15 05:03:42
3	5	AFRICAN EGG	A Fast-Paced Documentary of a Pastry Chef And ...	2006	1	None	6	2.99	130	22.99	G	Deleted Scenes	2006-02-15 05:03:42
4	6	AGENT TRUMAN	A Intrepid Panorama of a Robot And a Boy who m...	2006	1	None	3	2.99	169	17.99	PG	Deleted Scenes	2006-02-15 05:03:42
...	...	...	...	...	...	...	...	...	...	...	...	...	...
814	994	WYOMING STORM	A Awe-Inspiring Panorama of a Robot And a Boat...	2006	1	None	6	4.99	100	29.99	PG-13	Deleted Scenes	2006-02-15 05:03:42
815	996	YOUNG LANGUAGE	A Unbelieveable Yarn of a Boat And a Database ...	2006	1	None	6	0.99	183	9.99	G	Trailers,Behind the Scenes	2006-02-15 05:03:42
816	998	ZHIVAGO CORE	A Fateful Yarn of a Composer And a Man who mus...	2006	1	None	6	0.99	105	10.99	NC-17	Deleted Scenes	2006-02-15 05:03:42
817	999	ZOOLANDER FICTION	A Fateful Reflection of a Waitress And a Boat ...	2006	1	None	5	2.99	101	28.99	R	Trailers,Deleted Scenes	2006-02-15 05:03:42
818	1000	ZORRO ARK	A Intrepid Panorama of a Mad Scientist And a B...	2006	1	None	3	4.99	50	18.99	NC-17	Trailers,Commentaries,Behind the Scenes	2006-02-15 05:03:42

819 rows × 13 columns

In [75]:
```
# e) What is the total amount spent by customers for movies in the year 2005 ?
Q3_e = "SELECT SUM(amount) \
        FROM payment \
        WHERE payment_date BETWEEN '2005-01-01 00:00:00' AND '2005-12-31 23:59:59';"
qry(Q3_e)
```

Out[75]:

	SUM(amount)
0	66902.33

In [76]:
```
# f) What is the average replacement cost across all movies ?
Q3_f = "SELECT AVG(replacement_cost) \
        FROM film;"
qry(Q3_f)
```

Out[76]:

	AVG(replacement_cost)
0	19.984

In [77]:
```
# g) What is the standard deviation of rental rate across all movies ?
Q3_g = "SELECT STDDEV(rental_rate) \
        FROM film;"
qry(Q3_g)
```

Out[77]:

	STDDEV(rental_rate)
0	1.64557

In [78]:
```
# h) What is the midrange of the rental duration for all movies
Q3_h = "SELECT MAX(rental_duration) + MIN(rental_duration) / 2 as midrange \
        FROM film;"
qry(Q3_h)
```

Out[78]:

	midrange
0	8.5

Question 4

In [79]:
```
# a) Customers sorted by first Name and last name in ascending order.
Q4_a = "SELECT * \
        FROM customer \
        ORDER BY first_name, last_name ASC;"
qry(Q4_a)
```

Out[79]:

	customer_id	store_id	first_name	last_name	email	address_id	active	create_date	last_update
0	375	2	AARON	SELBY	AARON.SELBY@sakilacustomer.org	380	1	2006-02-14 22:04:37	2006-02-15 04:57:20
1	367	1	ADAM	GOOCH	ADAM.GOOCH@sakilacustomer.org	372	1	2006-02-14 22:04:37	2006-02-15 04:57:20
2	525	2	ADRIAN	CLARY	ADRIAN.CLARY@sakilacustomer.org	531	1	2006-02-14 22:04:37	2006-02-15 04:57:20
3	217	2	AGNES	BISHOP	AGNES.BISHOP@sakilacustomer.org	221	1	2006-02-14 22:04:36	2006-02-15 04:57:20
4	389	1	ALAN	KAHN	ALAN.KAHN@sakilacustomer.org	394	1	2006-02-14 22:04:37	2006-02-15 04:57:20
...	...	...	...	...	...	...	...	...	...
594	359	2	WILLIE	MARKHAM	WILLIE.MARKHAM@sakilacustomer.org	364	1	2006-02-14 22:04:37	2006-02-15 04:57:20
595	212	2	WILMA	RICHARDS	WILMA.RICHARDS@sakilacustomer.org	216	1	2006-02-14 22:04:36	2006-02-15 04:57:20
596	190	2	YOLANDA	WEAVER	YOLANDA.WEAVER@sakilacustomer.org	194	1	2006-02-14 22:04:36	2006-02-15 04:57:20
597	174	2	YVONNE	WATKINS	YVONNE.WATKINS@sakilacustomer.org	178	1	2006-02-14 22:04:36	2006-02-15 04:57:20
598	479	1	ZACHARY	HITE	ZACHARY.HITE@sakilacustomer.org	484	1	2006-02-14 22:04:37	2006-02-15 04:57:20

599 rows × 9 columns

In [80]:
```
# b) Count of movies that are either G/NC-17/PG-13/PG/R grouped by rating.
Q4_b = "SELECT rating, COUNT(film_id) \
        FROM film \
        GROUP BY rating;"
qry(Q4_b)
```

Out[80]:

	rating	COUNT(film_id)
0	PG	194
1	G	178
2	NC-17	210
3	PG-13	223
4	R	195

In [81]:
```
# c) Number of addresses in each district.
Q4_c = "SELECT district, COUNT(address_id) \
        FROM address \
        GROUP BY district;"
qry(Q4_c)
```

Out[81]:

	district	COUNT(address_id)
0	Alberta	2
1	QLD	2
2	Nagasaki	1
3	California	9
4	Attika	1
...	...	...
373	al-Manama	1
374	Fejr	1
375	West Greece	1
376	Piura	1
377	Vaud	1

378 rows × 2 columns

In [82]:
```
# d) Find the movies where rental rate is greater than 1$ and order result set by descending
# order.
Q4_d = "SELECT * \
        FROM film \
        WHERE rental_rate > 1 \
        ORDER BY rental_rate DESC;"
qry(Q4_d)
```

Out[82]:

	film_id	title	description	release_year	language_id	original_language_id	rental_duration	rental_rate	length	replacement_cost	rating	special_features	last_update
0	2	ACE GOLDFINGER	A Astounding Epistle of a Database Administrat...	2006	1	None	3	4.99	48	12.99	G	Trailers,Deleted Scenes	2006-02-15 05:03:42
1	7	AIRPLANE SIERRA	A Touching Saga of a Hunter And a Butler who m...	2006	1	None	6	4.99	62	28.99	PG-13	Trailers,Deleted Scenes	2006-02-15 05:03:42
2	8	AIRPORT POLLOCK	A Epic Tale of a Moose And a Girl who must Con...	2006	1	None	6	4.99	54	15.99	R	Trailers	2006-02-15 05:03:42
3	10	ALADDIN CALENDAR	A Action-Packed Tale of a Man And a Lumberjack...	2006	1	None	6	4.99	63	24.99	NC-17	Trailers,Deleted Scenes	2006-02-15 05:03:42
4	13	ALI FOREVER	A Action-Packed Drama of a Dentist And a Croco...	2006	1	None	4	4.99	150	21.99	PG	Deleted Scenes,Behind the Scenes	2006-02-15 05:03:42
...	...	...	...	...	...	...	...	...	...	...	...	...	...
654	987	WORDS HUNTER	A Action-Packed Reflection of a Composer And a...	2006	1	None	3	2.99	116	13.99	PG	Trailers,Commentaries,Deleted Scenes	2006-02-15 05:03:42
655	988	WORKER TARZAN	A Action-Packed Yarn of a Secret Agent And a T...	2006	1	None	7	2.99	139	26.99	R	Trailers,Commentaries,Behind the Scenes	2006-02-15 05:03:42
656	991	WORST BANGER	A Thrilling Drama of a Madman And a Dentist wh...	2006	1	None	4	2.99	185	26.99	PG	Deleted Scenes,Behind the Scenes	2006-02-15 05:03:42
657	993	WRONG BEHAVIOR	A Emotional Saga of a Crocodile And a Sumo Wre...	2006	1	None	6	2.99	178	10.99	PG-13	Trailers,Behind the Scenes	2006-02-15 05:03:42
658	999	ZOOLANDER FICTION	A Fateful Reflection of a Waitress And a Boat ...	2006	1	None	5	2.99	101	28.99	R	Trailers,Deleted Scenes	2006-02-15 05:03:42

659 rows × 13 columns

In [83]:
```
# e) Top 2 movies that are rated R with the highest replacement cost ?
Q4_e = "SELECT * \
        FROM film \
        WHERE rating = 'R' \
        ORDER BY replacement_cost DESC \
        LIMIT 2;"
qry(Q4_e)
```

Out[83]:

	film_id	title	description	release_year	language_id	original_language_id	rental_duration	rental_rate	length	replacement_cost	rating	special_features	last_update
0	138	CHARIOTS CONSPIRACY	A Unbelieveable Epistle of a Robot And a Husba...	2006	1	None	5	2.99	71	29.99	R	Deleted Scenes,Behind the Scenes	2006-02-15 05:03:42
1	525	LOATHING LEGALLY	A Boring Epistle of a Pioneer And a Mad Scient...	2006	1	None	4	0.99	140	29.99	R	Deleted Scenes	2006-02-15 05:03:42

In [84]:
```
# f) Find the most frequently occurring (mode) rental rate across products.
Q4_f = "SELECT rental_rate, COUNT(*) as frequency \
        FROM film \
        GROUP BY rental_rate \
        ORDER BY frequency DESC \
        LIMIT 1;"
qry(Q4_f)
```

Out[84]:

	rental_rate	frequency
0	0.99	341

In [85]:
```
# g) Find the 2 longest movies with movie length greater than 50mins and which has
# commentaries as a special features.
Q4_g = "SELECT * \
        FROM film \
        WHERE length > 50 and special_features LIKE '%commentaries%' \
        ORDER BY length DESC \
        LIMIT 2;"
qry(Q4_g)
```

Out[85]:

	film_id	title	description	release_year	language_id	original_language_id	rental_duration	rental_rate	length	replacement_cost	rating	special_features	last_update
0	182	CONTROL ANTHEM	A Fateful Documentary of a Robot And a Student...	2006	1	None	7	4.99	185	9.99	G	Commentaries	2006-02-15 05:03:42
1	817	SOLDIERS EVOLUTION	A Lacklusture Panorama of a A Shark And a Pion...	2006	1	None	7	4.99	185	27.99	R	Trailers,Commentaries,Deleted Scenes,Behind th...	2006-02-15 05:03:42

In [87]:
```
# h) List the years which has more than 2 movies released.
Q4_h = "SELECT release_year, COUNT(*) \
        FROM film \
        GROUP BY release_year \
        HAVING COUNT(*) > 2;"
qry(Q4_h)
```

Out[87]:

	release_year	COUNT(*)
0	2006	1000

Part C

Question 1

In [88]:
```
# a) List the actors (firstName, lastName) who acted in more then 25 movies.
# Note: Also show the count of movies against each actor

q1_a = "SELECT a.first_name, a.last_name, COUNT(*) as count \
        FROM actor as a \
        LEFT JOIN film_actor as fa ON a.actor_id = fa.actor_id \
        LEFT JOIN film as f ON fa.film_id = f.film_id \
        GROUP BY a.first_name, a.last_name \
        HAVING count > 25;"
qry(q1_a)
```

Out[88]:

	first_name	last_name	count
0	JOHNNY	LOLLOBRIGIDA	29
1	GRACE	MOSTEL	30
2	KARL	BERRY	31
3	UMA	WOOD	35
4	VIVIEN	BERGEN	30
...	...	...	...
120	JOHN	SUVARI	29
121	JAYNE	SILVERSTONE	27
122	BELA	WALKEN	30
123	REESE	WEST	33
124	MARY	KEITEL	40

125 rows × 3 columns

In [90]:
```
# b) List the actors who have worked in the German language movies.
# Note: Please execute the below SQL before answering this question.

#!/usr/bin/python3

# Open database connection
db = pymysql.connect(host="localhost",user="root",password="rootroot",database="sakila" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

q = "SET SQL_SAFE_UPDATES=0;"
q2 = "UPDATE film SET language_id=6 WHERE title LIKE '%ACADEMY%';"

# execute SQL query using execute() method.
cursor.execute(q)
cursor.execute(q2)

# disconnect from server
db.close()


q1_b = "SELECT a.first_name, a.last_name, a.actor_id, l.name \
        FROM actor as a \
        LEFT JOIN film_actor as fa on a.actor_id = fa.actor_id \
        LEFT JOIN film as f on fa.film_id = f.film_id \
        LEFT JOIN language as l on l.language_id = f.language_id \
        WHERE l.name = 'German';"
qry(q1_b)
```

Out[90]:

	first_name	last_name	actor_id	name
0	PENELOPE	GUINESS	1	German
1	CHRISTIAN	GABLE	10	German
2	LUCILLE	TRACY	20	German
3	SANDRA	PECK	30	German
4	JOHNNY	CAGE	40	German
5	GARY	PHOENIX	51	German
6	MENA	TEMPLE	53	German
7	KENNETH	PESCI	88	German
8	WARREN	NOLTE	108	German
9	OPRAH	KILMER	162	German
10	DEBBIE	AKROYD	182	German
11	ROCK	DUKAKIS	188	German
12	CUBA	BIRCH	189	German
13	MERYL	ALLEN	194	German
14	MARY	KEITEL	198	German

In [91]:
```
# c) List the actors who acted in horror movies.
# Note: Show the count of movies against each actor in the result set.
q1_c = "SELECT a.first_name, a.last_name, COUNT(f.title) as count \
        FROM actor as a \
        LEFT JOIN film_actor as fa on a.actor_id = fa.actor_id \
        LEFT JOIN film as f on fa.film_id = f.film_id \
        LEFT JOIN film_category as fc on f.film_id = fc.film_id \
        LEFT JOIN category as c on fc.category_id = c.category_id \
        WHERE c.name = 'horror' \
        GROUP BY a.first_name, a.last_name;"
qry(q1_c)
```

Out[91]:

	first_name	last_name	count
0	BOB	FAWCETT	1
1	MINNIE	ZELLWEGER	3
2	SEAN	GUINESS	1
3	CHRIS	DEPP	2
4	JODIE	DEGENERES	2
...	...	...	...
150	ALBERT	NOLTE	1
151	JAYNE	NEESON	1
152	CHRIS	BRIDGES	1
153	SIDNEY	CROWE	1
154	SALMA	NOLTE	1

155 rows × 3 columns

In [93]:
```
# d) List all customers who rented more than 3 horror movies.
q1_d = "SELECT c.first_name, c.last_name, COUNT(DISTINCT(f.film_id)) as count \
        FROM customer as c \
        LEFT JOIN rental as r ON r.customer_id = c.customer_id \
        LEFT JOIN inventory as i ON r.inventory_id = i.inventory_id \
        LEFT JOIN film as f ON i.film_id = f.film_id \
        LEFT JOIN film_category as fc on f.film_id = fc.film_id \
        LEFT JOIN category as ca on fc.category_id = ca.category_id \
        WHERE ca.name = 'horror' \
        GROUP BY c.first_name, c.last_name \
        HAVING count > 3;"
qry(q1_d)
```

Out[93]:

	first_name	last_name	count
0	ANA	BRADLEY	4
1	ANDREA	HENDERSON	4
2	BERTHA	FERGUSON	4
3	BYRON	BOX	4
4	CARL	ARTIS	4
5	CLAUDE	HERZOG	4
6	CONNIE	WALLACE	4
7	DUANE	TUBBS	4
8	EDGAR	RHOADS	4
9	ELEANOR	HUNT	4
10	EMMA	BOYD	5
11	JOSE	ANDREW	4
12	KARL	SEAL	5
13	KRISTINA	CHAMBERS	4
14	LOUIS	LEONE	4
15	NANCY	THOMAS	4
16	NEIL	RENNER	4
17	NINA	SOTO	4
18	OSCAR	AQUINO	4
19	RACHEL	BARNES	4
20	RAYMOND	MCWHORTER	4
21	REGINA	BERRY	4
22	RICHARD	MCCRARY	4
23	ROSEMARY	SCHMIDT	5
24	SONIA	GREGORY	4
25	SUE	PETERS	4
26	THELMA	MURRAY	5
27	THERESA	WATSON	4
28	TINA	SIMMONS	4
29	TOMMY	COLLAZO	4
30	WARREN	SHERROD	4
31	ZACHARY	HITE	4

In [94]:
```
# e) List all customers who rented the movie which starred SCARLETT BENING
q1_e = "SELECT distinct(c.customer_id), c.first_name, c.last_name \
        FROM customer as c \
        LEFT JOIN rental as r ON r.customer_id = c.customer_id \
        LEFT JOIN inventory as i ON r.inventory_id = i.inventory_id \
        LEFT JOIN film as f ON i.film_id = f.film_id \
        LEFT JOIN film_actor as fa ON f.film_id = fa.film_id \
        LEFT JOIN actor as a ON fa.actor_id = a.actor_id \
        WHERE a.first_name = 'SCARLETT' and a.last_name = 'BENING';"
qry(q1_e)
```

Out[94]:

	customer_id	first_name	last_name
0	59	CHERYL	MURPHY
1	339	WALTER	PERRYMAN
2	1	MARY	SMITH
3	128	MARJORIE	TUCKER
4	29	ANGELA	HERNANDEZ
...	...	...	...
257	324	GARY	COY
258	483	VERNON	CHAPA
259	522	ARNOLD	HAVENS
260	320	ANTHONY	SCHWAB
261	439	ALEXANDER	FENNELL

262 rows × 3 columns

In [95]:
```
# f) Which customers residing at postal code 62703 rented movies that were
# Documentaries.
q1_f = "SELECT c.first_name, c.last_name, a.postal_code \
        FROM customer as c \
        LEFT JOIN rental as r ON r.customer_id = c.customer_id \
        LEFT JOIN inventory as i ON r.inventory_id = i.inventory_id \
        LEFT JOIN film as f ON i.film_id = f.film_id \
        LEFT JOIN film_category as fc on f.film_id = fc.film_id \
        LEFT JOIN category as ca on fc.category_id = ca.category_id \
        LEFT JOIN address as a on a.address_id = c.address_id \
        WHERE a.postal_code = '62703' and ca.name = 'Documentary';"
qry(q1_f)
```

Out[95]:

	first_name	last_name	postal_code
0	ANDY	VANHORN	62703

In [96]:
```
# g) Find all the addresses where the second address line is not empty (i.e., contains some
# text), and return these second addresses sorted.

# All the address2 are empty

q1_g = "SELECT * \
        FROM address \
        WHERE address2 IS NOT NULL and address2 <> '';"
qry(q1_g)
```

Out[96]:

address_id	address	address2	district	city_id	postal_code	phone	location	last_update

In [98]:
```
# h) How many films involve a “Crocodile” and a “Shark” based on film description ?
q1_h = 'SELECT COUNT(f.title) as count \
        FROM film as f \
        LEFT JOIN film_text as ft ON ft.film_id = f.film_id \
        WHERE ft.description LIKE "%Crocodile%" and ft.description LIKE "%Shark%";'
qry(q1_h)
```

Out[98]:

	count
0	10

In [99]:
```
# i) List the actors who played in a film involving a "Crocodile" and a "Shark", along with
# the release year of the movie, sorted by the actors' last names.
ql_i = 'SELECT f.title, a.first_name, a.last_name, f.release_year \
        FROM film as f \
        LEFT JOIN film_text as ft ON ft.film_id = f.film_id \
        LEFT JOIN film_actor as fa ON f.film_id = fa.film_id \
        LEFT JOIN actor as a ON fa.actor_id = a.actor_id \
        WHERE ft.description LIKE "%Crocodile%" and ft.description LIKE "%Shark%" \
        ORDER BY a.last_name;'
qry(ql_i)
```

Out[99]:

	title	first_name	last_name	release_year
0	SLACKER LIAISONS	None	None	2006
1	MADIGAN DORADO	KIRSTEN	AKROYD	2006
2	CLEOPATRA DEVIL	KIM	ALLEN	2006
3	WARLOCK WEREWOLF	AUDREY	BAILEY	2006
4	SHOOTIST SUPERFLY	JULIA	BARRYMORE	2006
5	CONNECTICUT TRAMP	VIVIEN	BASINGER	2006
6	EXCITEMENT EVE	VIVIEN	BERGEN	2006
7	CLEOPATRA DEVIL	KARL	BERRY	2006
8	CONNECTICUT TRAMP	KARL	BERRY	2006
9	CONNECTICUT TRAMP	HENRY	BERRY	2006
10	DANCING FEVER	LAURA	BRODY	2006
11	HONEY TIES	ZERO	CAGE	2006
12	CONNECTICUT TRAMP	JOHNNY	CAGE	2006
13	MADIGAN DORADO	JON	CHASE	2006
14	CLEOPATRA DEVIL	FRED	COSTNER	2006
15	CONNECTICUT TRAMP	FRED	COSTNER	2006
16	EXCITEMENT EVE	SUSAN	DAVIS	2006
17	SPLASH GUMP	JENNIFER	DAVIS	2006
18	EXCITEMENT EVE	GINA	DEGENERES	2006
19	HONEY TIES	JULIANNE	DENCH	2006
20	DANCING FEVER	ROCK	DUKAKIS	2006
21	WARLOCK WEREWOLF	HUMPHREY	GARLAND	2006
22	SPLASH GUMP	AL	GARLAND	2006
23	CLEOPATRA DEVIL	EWAN	GOODING	2006
24	SPLASH GUMP	PENELOPE	GUINESS	2006
25	CLEOPATRA DEVIL	WILLIAM	HACKMAN	2006
26	CONNECTICUT TRAMP	MEG	HAWKE	2006
27	SHOOTIST SUPERFLY	WOODY	HOFFMAN	2006
28	EXCITEMENT EVE	MORGAN	HOPKINS	2006
29	DANCING FEVER	JANE	JACKMAN	2006
30	CONNECTICUT TRAMP	ALBERT	JOHANSSON	2006
31	SPLASH GUMP	RAY	JOHANSSON	2006
32	HONEY TIES	ALBERT	JOHANSSON	2006
33	MADIGAN DORADO	RAY	JOHANSSON	2006
34	MADIGAN DORADO	MILLA	KEITEL	2006
35	SHOOTIST SUPERFLY	FAY	KILMER	2006
36	DANCING FEVER	MATTHEW	LEIGH	2006
37	MADIGAN DORADO	GENE	MCKELLEN	2006
38	EXCITEMENT EVE	GRACE	MOSTEL	2006
39	WARLOCK WEREWOLF	GRACE	MOSTEL	2006
40	SPLASH GUMP	CHRISTIAN	NEESON	2006
41	HONEY TIES	ALBERT	NOLTE	2006
42	DANCING FEVER	JAYNE	NOLTE	2006
43	EXCITEMENT EVE	KENNETH	PALTROW	2006
44	HONEY TIES	KIRSTEN	PALTROW	2006
45	EXCITEMENT EVE	SANDRA	PECK	2006
46	EXCITEMENT EVE	PENELOPE	PINKETT	2006
47	SPLASH GUMP	CAMERON	STREEP	2006
48	CLEOPATRA DEVIL	JOHN	SUVARI	2006
49	DANCING FEVER	KENNETH	TORN	2006
50	WARLOCK WEREWOLF	HELEN	VOIGHT	2006
51	SPLASH GUMP	MORGAN	WILLIAMS	2006
52	WARLOCK WEREWOLF	GROUCHO	WILLIAMS	2006
53	SPLASH GUMP	GENE	WILLIS	2006
54	SPLASH GUMP	HUMPHREY	WILLIS	2006
55	WARLOCK WEREWOLF	WILL	WILSON	2006
56	CLEOPATRA DEVIL	FAY	WOOD	2006
57	SPLASH GUMP	UMA	WOOD	2006
58	MADIGAN DORADO	MINNIE	ZELLWEGER	2006

In [100]:
```
# j) Find all the film categories in which there are between 55 and 65 films. Return the
# names of categories and the number of films per category, sorted from highest to lowest by
# the number of films.
q1_j = "SELECT ca.name, count(*) as count \
        FROM category as ca \
        LEFT JOIN film_category as fa ON fa.category_id = ca.category_id \
        LEFT JOIN film as f ON f.film_id = fa.film_id \
        GROUP BY ca.name \
        HAVING count BETWEEN 55 AND 65 \
        ORDER BY count DESC;"
qry(q1_j)
```

Out[100]:

	name	count
0	Action	64
1	New	63
2	Drama	62
3	Games	61
4	Sci-Fi	61
5	Children	60
6	Comedy	58
7	Classics	57
8	Travel	57
9	Horror	56

In [101]:
```
# k) In which of the film categories is the average difference between the film replacement
# cost and the rental rate larger than 17$?
q1_k = "SELECT ca.name, ((SUM(f.replacement_cost) - SUM(f.rental_rate)) / COUNT(*)) as avg_difference \
        FROM category as ca \
        LEFT JOIN film_category as fa ON fa.category_id = ca.category_id \
        LEFT JOIN film as f ON f.film_id = fa.film_id \
        GROUP BY ca.name \
        HAVING avg_difference > 17;"
qry(q1_k)
```

Out[101]:

	name	avg_difference
0	Action	18.265625
1	Animation	17.318182
2	Children	17.166667
3	Classics	18.263158
4	Drama	18.064516
5	Games	17.032787
6	Sci-Fi	17.934426
7	Sports	17.270270

In [102]:
```
# l) Many DVD stores produce a daily list of overdue rentals so that customers can be
# contacted and asked to return their overdue DVDs. To create such a list, search the rental
# table for films with a return date that is NULL and where the rental date is further in the
# past than the rental duration specified in the film table. If so, the film is overdue and we
# should produce the name of the film along with the customer name and phone number.
q1_l = "SELECT r.return_date, r.rental_date, f.rental_duration, f.title, c.first_name, c.last_name, a.phone \
        FROM rental as r \
        LEFT JOIN inventory as i ON i.inventory_id = r.inventory_id \
        LEFT JOIN film as f ON f.film_id = i.film_id \
        LEFT JOIN customer as c ON r.customer_id = c.customer_id \
        LEFT JOIN address as a ON a.address_id = c.address_id \
        WHERE r.return_date IS NULL and DATEDIFF(CURRENT_DATE(), r.rental_date) > f.rental_duration;"
qry(q1_l)
```

Out[102]:

	return_date	rental_date	rental_duration	title	first_name	last_name	phone
0	None	2006-02-14 15:16:03	5	HYDE DOCTOR	GAIL	KNIGHT	904253967161
1	None	2006-02-14 15:16:03	6	HUNGER ROOF	GREGORY	MAULDIN	80303246192
2	None	2006-02-14 15:16:03	6	FRISCO FORREST	LOUISE	JENKINS	800716535041
3	None	2006-02-14 15:16:03	4	TITANS JERK	WILLIE	HOWELL	991802825778
4	None	2006-02-14 15:16:03	6	CONNECTION MICROCOSMOS	EMILY	DIAZ	333339908719
...	...	...	...	...	...	...	...
178	None	2006-02-14 15:16:03	3	DANCES NONE	JESSIE	BANKS	352679173732
179	None	2006-02-14 15:16:03	4	CONVERSATION DOWNHILL	RAFAEL	ABNEY	82671830126
180	None	2006-02-14 15:16:03	7	SHOCK CABIN	STEPHANIE	MITCHELL	42384721397
181	None	2006-02-14 15:16:03	3	WEDDING APOLLO	REGINA	BERRY	201705577290
182	None	2006-02-14 15:16:03	3	WINDOW SIDE	JEREMY	HURTADO	600264533987

183 rows × 7 columns

In [103]:
```
# m) Find the list of all customers and staff given a store id
# Note : use a set operator, do not remove duplicates
q1_m = '(SELECT s.store_id, c.first_name, c.last_name, "customer" as label \
        FROM store as s \
        LEFT JOIN customer as c on s.store_id = c.store_id) UNION ALL \
        (SELECT s.store_id, stf.first_name, stf.last_name, "staff" as label \
        FROM store as s \
        LEFT JOIN staff as stf on s.store_id = stf.store_id);'
qry(q1_m)
```

Out[103]:

	store_id	first_name	last_name	label
0	1	MARY	SMITH	customer
1	1	PATRICIA	JOHNSON	customer
2	1	LINDA	WILLIAMS	customer
3	1	ELIZABETH	BROWN	customer
4	1	MARIA	MILLER	customer
...	...	...	...	...
596	2	SETH	HANNON	customer
597	2	RENE	MCALISTER	customer
598	2	AUSTIN	CINTRON	customer
599	1	Mike	Hillyer	staff
600	2	Jon	Stephens	staff

601 rows × 4 columns

Question 2

In [117]:
```
######## QUESTION 2 ######## - { 10 Points }
# a) List actors and customers whose first name is the same as the first name of the actor
# with ID 8.
q2_a = "(SELECT a.actor_id as id, a.first_name, a.last_name, 'actor' as label \
        FROM actor as a \
        WHERE a.first_name = (SELECT first_name FROM actor WHERE actor_id = 8)) \
        UNION ALL \
        (SELECT c.customer_id as id, c.first_name, c.last_name, 'customer' as label \
        FROM customer as c \
        WHERE c.first_name = (SELECT first_name FROM actor WHERE actor_id = 8));"
qry(q2_a)
```

Out[117]:

	id	first_name	last_name	label
0	8	MATTHEW	JOHANSSON	actor
1	103	MATTHEW	LEIGH	actor
2	181	MATTHEW	CARREY	actor
3	323	MATTHEW	MAHAN	customer

In [105]:
```
# b) List customers and payment amounts, with payments greater than average the
# payment amount
q2_b = "SELECT c.first_name, c.last_name, p.amount \
        FROM customer as c \
        LEFT JOIN payment as p ON c.customer_id = p.customer_id \
        WHERE p.amount > (SELECT AVG(amount) FROM payment);"
qry(q2_b)
```

Out[105]:

	first_name	last_name	amount
0	MARY	SMITH	5.99
1	MARY	SMITH	9.99
2	MARY	SMITH	4.99
3	MARY	SMITH	4.99
4	MARY	SMITH	5.99
...	...	...	...
7741	AUSTIN	CINTRON	6.99
7742	AUSTIN	CINTRON	9.99
7743	AUSTIN	CINTRON	6.99
7744	AUSTIN	CINTRON	4.99
7745	AUSTIN	CINTRON	8.99

7746 rows × 3 columns

In [106]:
```
# c) List customers who have rented movies at least once
# Note: use IN clause
q2_c = "SELECT first_name, last_name \
        FROM customer \
        WHERE customer_id IN (SELECT r.customer_id FROM rental as r);"
qry(q2_c)
```

Out[106]:

	first_name	last_name
0	MARY	SMITH
1	PATRICIA	JOHNSON
2	LINDA	WILLIAMS
3	BARBARA	JONES
4	ELIZABETH	BROWN
...	...	...
594	TERRENCE	GUNDERSON
595	ENRIQUE	FORSYTHE
596	FREDDIE	DUGGAN
597	WADE	DELVALLE
598	AUSTIN	CINTRON

599 rows × 2 columns

In [108]:
```
# d) Find the floor of the maximum, minimum and average payment amount
q2_d = "SELECT FLOOR(MAX(amount)), FLOOR(MIN(amount)), FLOOR(AVG(amount)) \
        FROM payment;"
qry(q2_d)
```

Out[108]:

	FLOOR(MAX(amount))	FLOOR(MIN(amount))	FLOOR(AVG(amount))
0	11	0	4

Question 3

In [109]:
```
######## QUESTION 3 ######## - { 5 Points }
# a) Create a view called actors_portfolio which contains information about actors and
# films ( including titles and category).


#!/usr/bin/python3

# Open database connection
db = pymysql.connect(host="localhost",user="root",password="rootroot",database="sakila" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

q = "DROP view actors_portfolio;"
q3_a = "CREATE VIEW actors_portfolio AS \
        SELECT a.actor_id, a.first_name, a.last_name, f.title, c.name as category \
        FROM actor as a \
        LEFT JOIN film_actor as fa ON a.actor_id = fa.actor_id \
        LEFT JOIN film as f ON fa.film_id = f.film_id \
        LEFT JOIN film_category as fc ON fc.film_id = f.film_id \
        LEFT JOIN category as c ON fc.category_id = c.category_id;"

# execute SQL query using execute() method.
cursor.execute(q)
cursor.execute(q3_a)

# disconnect from server
db.close()
```

In [110]:
```
# b) Describe the structure of the view and query the view to get information on the actor
# ADAM GRANT
q3_b_1 = "DESCRIBE actors_portfolio;"

qry(q3_b_1)
```

Out[110]:

	Field	Type	Null	Key	Default	Extra
0	actor_id	smallint unsigned	NO		0	
1	first_name	varchar(45)	NO		None	
2	last_name	varchar(45)	NO		None	
3	title	varchar(128)	YES		None	
4	category	varchar(25)	YES		None	

In [111]:
```
q3_b_2 = 'SELECT * \
        FROM actors_portfolio \
        WHERE first_name = "ADAM" AND last_name = "GRANT";'
qry(q3_b_2)
```

Out[111]:

	actor_id	first_name	last_name	title	category
0	71	ADAM	GRANT	ANNIE IDENTITY	Sci-Fi
1	71	ADAM	GRANT	BALLROOM MOCKINGBIRD	Foreign
2	71	ADAM	GRANT	DISCIPLE MOTHER	Travel
3	71	ADAM	GRANT	FIREBALL PHILADELPHIA	Comedy
4	71	ADAM	GRANT	GLADIATOR WESTWARD	Family
5	71	ADAM	GRANT	GLORY TRACY	Games
6	71	ADAM	GRANT	GROUNDHOG UNCUT	Comedy
7	71	ADAM	GRANT	HAPPINESS UNITED	Foreign
8	71	ADAM	GRANT	IDOLS SNATCHERS	Children
9	71	ADAM	GRANT	LOSER HUSTLER	Sports
10	71	ADAM	GRANT	MARS ROMAN	Games
11	71	ADAM	GRANT	MIDNIGHT WESTWARD	Action
12	71	ADAM	GRANT	OPERATION OPERATION	Comedy
13	71	ADAM	GRANT	SEABISCUIT PUNK	Sports
14	71	ADAM	GRANT	SPLENDOR PATTON	Children
15	71	ADAM	GRANT	TADPOLE PARK	Classics
16	71	ADAM	GRANT	TWISTED PIRATES	Children
17	71	ADAM	GRANT	WANDA CHAMBER	Games

In [ ]:
```
# c) Insert a new movie titled Data Hero in Sci-Fi Category starring ADAM GRANT
# Note: this is feasible
q3_c = "INSERT INTO actors_portfolio(actor_id, first_name, last_name, title, category) \
        values(71, 'ADAM','GRANT', 'Data Hero', 'Sci-Fi');"

# we cannot insert new value into a view. We can only update the base table.
```

Question 4

In [121]:
```
######## QUESTION 4 ######## - { 5 Points }
# a) Extract the street number ( characters 1 through 4 ) from customer addressLine1
# Note: this is a compound query
q4_a = "SELECT \
        SUBSTRING( \
            a.address, \
            1, \
            LOCATE(' ', address) - 1 \
        ) AS street_number, a.address \
        FROM address as a \
        WHERE a.address_id IN (SELECT c.address_id FROM customer as c);"
qry(q4_a)
```

Out[121]:

	street_number	address
0	1913	1913 Hanoi Way
1	1121	1121 Loja Avenue
2	692	692 Joliet Street
3	1566	1566 Inegl Manor
4	53	53 Idfu Parkway
...	...	...
594	844	844 Bucuresti Place
595	1101	1101 Bucuresti Boulevard
596	1103	1103 Quilmes Boulevard
597	1331	1331 Usak Boulevard
598	1325	1325 Fukuyama Street

599 rows × 2 columns

In [113]:
```
# b) Find out actors whose last name starts with character A, B or C.
q4_b = "SELECT actor_id, first_name, last_name \
        FROM actor \
        WHERE last_name REGEXP '^(A|B|C)';"
qry(q4_b)
```

Out[113]:

	actor_id	first_name	last_name
0	3	ED	CHASE
1	11	ZERO	CAGE
2	12	KARL	BERRY
3	14	VIVIEN	BERGEN
4	16	FRED	COSTNER
5	25	KEVIN	BLOOM
6	26	RIP	CRAWFORD
7	37	VAL	BOLGER
8	39	GOLDIE	BRODY
9	40	JOHNNY	CAGE
10	47	JULIA	BARRYMORE
11	49	ANNE	CRONYN
12	57	JUDE	CRUISE
13	58	CHRISTIAN	AKROYD
14	60	HENRY	BERRY
15	67	JESSICA	BAILEY
16	76	ANGELINA	ASTAIRE
17	80	RALPH	CRUZ
18	86	GREG	CHAPLIN
19	91	CHRISTOPHER	BERRY
20	92	KIRSTEN	AKROYD
21	98	CHRIS	BRIDGES
22	104	PENELOPE	CRONYN
23	105	SIDNEY	CROWE
24	112	RUSSELL	BACALL
25	115	HARRISON	BALE
26	118	CUBA	ALLEN
27	121	LIZA	BERGMAN
28	124	SCARLETT	BENING
29	129	DARYL	CRAWFORD
30	145	KIM	ALLEN
31	158	VIVIEN	BASINGER
32	159	LAURA	BRODY
33	167	LAURENCE	BULLOCK
34	174	MICHAEL	BENING
35	176	JON	CHASE
36	181	MATTHEW	CARREY
37	182	DEBBIE	AKROYD
38	183	RUSSELL	CLOSE
39	185	MICHAEL	BOLGER
40	187	RENEE	BALL
41	189	CUBA	BIRCH
42	190	AUDREY	BAILEY
43	194	MERYL	ALLEN

In [114]:
```
# c) Find film titles that contains exactly 10 characters
q4_c = "SELECT title \
        FROM film \
        WHERE title REGEXP '^.{10}$';"
qry(q4_c)
```

Out[114]:

	title
0	ALONE TRIP
1	BASIC EASY
2	BUGSY SONG
3	CAUSE DATE
4	CHILL LUCK
...	...
61	WAGON JAWS
62	WAIT CIDER
63	WARS PLUTO
64	WRATH MILE
65	YOUTH KICK

66 rows × 1 columns

In [115]:
```
# d) Format a payment_date using the following format e.g "22/1/2016"
q4_d = "SELECT payment_id, DATE_FORMAT(payment_date, '%d/%m/%Y') AS 'DD/MM/YYYY' FROM payment;"
qry(q4_d)
```

Out[115]:

	payment_id	DD/MM/YYYY
0	1	25/05/2005
1	2	28/05/2005
2	3	15/06/2005
3	4	15/06/2005
4	5	15/06/2005
...	...	...
16044	16045	21/08/2005
16045	16046	21/08/2005
16046	16047	23/08/2005
16047	16048	23/08/2005
16048	16049	23/08/2005

16049 rows × 2 columns

In [116]:
```
# e) Find the number of days between two date values rental_date & return_date
q4_e = "SELECT *, DATEDIFF(return_date, rental_date) AS numberOfDays FROM rental;"
qry(q4_e)
```

Out[116]:

	rental_id	rental_date	inventory_id	customer_id	return_date	staff_id	last_update	numberOfDays
0	1	2005-05-24 22:53:30	367	130	2005-05-26 22:04:30	1	2006-02-15 21:30:53	2.0
1	2	2005-05-24 22:54:33	1525	459	2005-05-28 19:40:33	1	2006-02-15 21:30:53	4.0
2	3	2005-05-24 23:03:39	1711	408	2005-06-01 22:12:39	1	2006-02-15 21:30:53	8.0
3	4	2005-05-24 23:04:41	2452	333	2005-06-03 01:43:41	2	2006-02-15 21:30:53	10.0
4	5	2005-05-24 23:05:21	2079	222	2005-06-02 04:33:21	1	2006-02-15 21:30:53	9.0
...	...	...	...	...	...	...	...	...
16039	16045	2005-08-23 22:25:26	772	14	2005-08-25 23:54:26	1	2006-02-15 21:30:53	2.0
16040	16046	2005-08-23 22:26:47	4364	74	2005-08-27 18:02:47	2	2006-02-15 21:30:53	4.0
16041	16047	2005-08-23 22:42:48	2088	114	2005-08-25 02:48:48	2	2006-02-15 21:30:53	2.0
16042	16048	2005-08-23 22:43:07	2019	103	2005-08-31 21:33:07	1	2006-02-15 21:30:53	8.0
16043	16049	2005-08-23 22:50:12	2666	393	2005-08-30 01:01:12	2	2006-02-15 21:30:53	7.0

16044 rows × 8 columns

Question 5

In [8]:
```
# Visualize Data
from plotly.offline import init_notebook_mode,iplot
import plotly.graph_objects as go
import plotly.graph_objects as go
import cufflinks as cf
```

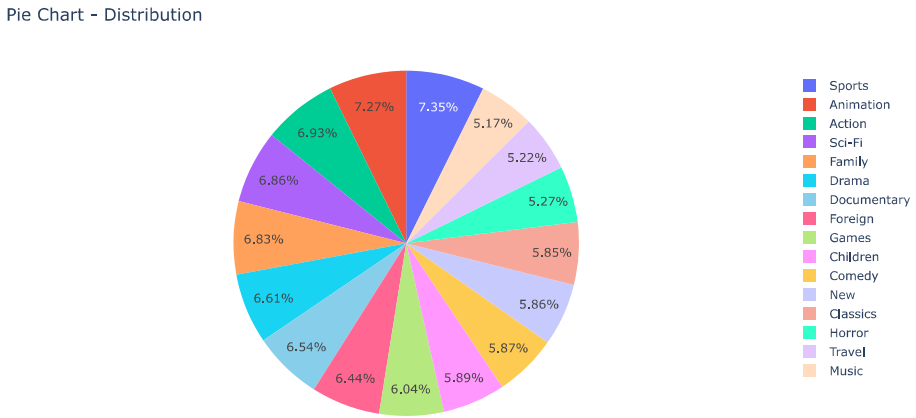
In [41]:
```
# 1. check the distribution of the rental counts of each film category
#    To see which kind of movie is relatively more popular
q5_1 = "SELECT c.name, COUNT(*) \
FROM rental as r \
LEFT JOIN inventory as i on i.inventory_id = r.inventory_id \
LEFT JOIN film as f on f.film_id = i.film_id \
LEFT JOIN film_category as fc on fc.film_id = f.film_id \
LEFT JOIN category as c on fc.category_id = c.category_id \
GROUP BY c.name;"
df = qry(q5_1)
```

In [42]:
```
df
```

Out[42]:

	name	COUNT(*)
0	Documentary	1050
1	Horror	846
2	Family	1096
3	Foreign	1033
4	Comedy	941
5	Sports	1179
6	Music	830
7	Animation	1166
8	Action	1112
9	New	940
10	Sci-Fi	1101
11	Classics	939
12	Games	969
13	Children	945
14	Travel	837
15	Drama	1060

In [43]:
```
labels=df['name']
values=df['COUNT(*)']

trace=go.Pie(labels=labels,values=values, marker=dict(colors=['skyblue']),hoverinfo="value")
data = [trace]
layout = go.Layout(title="Pie Chart - Distribution")
fig = go.Figure(data = data,layout = layout)

iplot(fig)
```



In [ ]:
```
# Insights:
# The above pie chart shows that "Sports" is the category that is rent most while "Music" is the least.
# This result might indicate that movies in "Sports" category have relatively more popularity while those
# in "Music" have relatively less popularity.
# So it might be better to get more sports movie for rental in order to create more rental revenue.
```

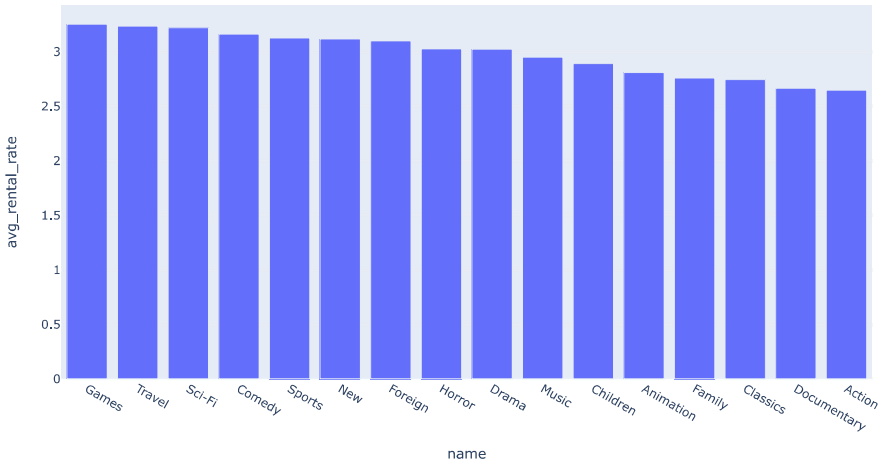
In [20]:
```
# 2. Check the averafe rental rate for each film catergory
#    Combine the result with the rental count for each category
q5_2 = "SELECT c.name, AVG(f.rental_rate) as avg_rental_rate \
FROM film as f \
LEFT JOIN film_category as fc on f.film_id = fc.film_id \
LEFT JOIN category as c on c.category_id = fc.category_id \
GROUP BY c.name \
ORDER BY avg_rental_rate DESC;"
df = qry(q5_2)
```

In [21]:
```
df
```

Out[21]:

	name	avg_rental_rate
0	Games	3.252295
1	Travel	3.235614
2	Sci-Fi	3.219508
3	Comedy	3.162414
4	Sports	3.125135
5	New	3.116984
6	Foreign	3.099589
7	Horror	3.025714
8	Drama	3.022258
9	Music	2.950784
10	Children	2.890000
11	Animation	2.808182
12	Family	2.758116
13	Classics	2.744386
14	Documentary	2.666471
15	Action	2.646250

In [22]:
```
import plotly.express as px
fig = px.bar(df, x='name', y='avg_rental_rate')
fig.show()
```



In [ ]:
```
# Insight:
# From the plot we could know that the "Games" category has the highest average rental rate while the
# "Action" category lowest average rental rate. If we combine this information with the above rental counts
# for each category, we could see that "Action" has a high total rental count (3rd highest) while "Games"
# has a medium count.
# The "Sports" has highest rental count while "Music" has the lowest count. But their average rental rate
# are both around the middle, with that of "Sports" higher than that of "Music".
# So this might indicate that the rental rate might be a factor that have influnce on the rental count
# (e.g. "Action") but not on every category.
```

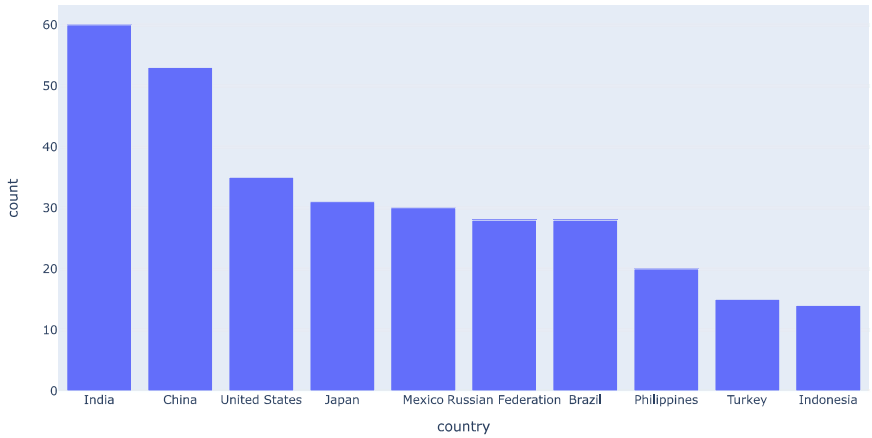
In [30]:
```
# 3. check the number of customers from each country, focus on the top 10 countries.
#    to see people from which country are more likely to rent a film
q5_3 = "SELECT co.country, count(*) as count \
FROM customer as c \
LEFT JOIN address as a ON c.address_id = a.address_id \
LEFT JOIN city ON city.city_id = c.customer_id \
LEFT JOIN country as co ON city.country_id = co.country_id \
GROUP BY co.country \
ORDER BY count DESC \
LIMIT 10;"

df = qry(q5_3)
```

In [31]:
```
df
```

Out[31]:

	country	count
0	India	60
1	China	53
2	United States	35
3	Japan	31
4	Mexico	30
5	Russian Federation	28
6	Brazil	28
7	Philippines	20
8	Turkey	15
9	Indonesia	14

In [32]:
```
import plotly.express as px
fig = px.bar(df, x='country', y='count')
fig.show()
```



In [ ]:
```
# Insight:
# From the above histogram we could know that the top 5 country with highest rental counts are:
# 1. India, 2. China, 3. U.S., 4. Japan, 5. Mexico.
# We then know that the number of potential customers are greatest from the above country and we could consider to
# focus more on these countries customer, to analyze their behaviors in order to create more rentals.
```

In [35]:

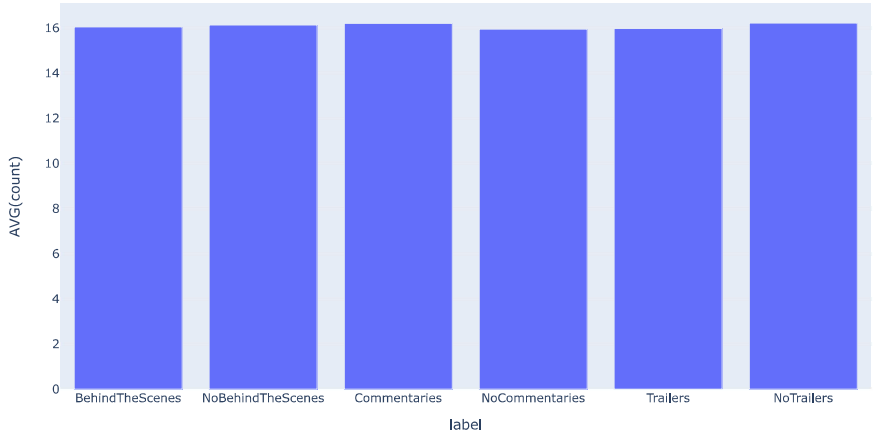
```
# 4. Check the influnce of special feature "Behind the Scenes" on rental count

#!/usr/bin/python3

# Open database connection
db = pymysql.connect(host="localhost",user="root",password="rootroot",database="sakila" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

q1 = "DROP VIEW IF EXISTS BehindTheScenes;"
q2 = "DROP VIEW IF EXISTS NoBehindTheScenes;"
q3 = "DROP VIEW IF EXISTS Commentaries;"
q4 = "DROP VIEW IF EXISTS NoCommentaries;"
q5 = "DROP VIEW IF EXISTS Trailers;"
q6 = "DROP VIEW IF EXISTS NoTrailers;"

q7 = 'CREATE VIEW BehindTheScenes AS \
SELECT f.title, f.special_features, COUNT(*) as count \
FROM film as f \
LEFT JOIN inventory as i ON i.film_id = f.film_id \
LEFT JOIN rental as r ON r.inventory_id = i.inventory_id \
WHERE special_features LIKE "%Behind the Scenes%" \
GROUP BY f.title \
ORDER BY count DESC;'

q8 = 'CREATE VIEW NoBehindTheScenes AS \
SELECT f.title, f.special_features, COUNT(*) as count \
FROM film as f \
LEFT JOIN inventory as i ON i.film_id = f.film_id \
LEFT JOIN rental as r ON r.inventory_id = i.inventory_id \
WHERE special_features NOT LIKE "%Behind the Scenes%" \
GROUP BY f.title \
ORDER BY count DESC;'


q9 = 'CREATE VIEW Commentaries AS \
SELECT f.title, f.special_features, COUNT(*) as count \
FROM film as f \
LEFT JOIN inventory as i ON i.film_id = f.film_id \
LEFT JOIN rental as r ON r.inventory_id = i.inventory_id \
WHERE special_features LIKE "%commentaries%" \
GROUP BY f.title \
ORDER BY count DESC;'

q10 = 'CREATE VIEW NoCommentaries AS \
SELECT f.title, f.special_features, COUNT(*) as count \
FROM film as f \
LEFT JOIN inventory as i ON i.film_id = f.film_id \
LEFT JOIN rental as r ON r.inventory_id = i.inventory_id \
WHERE special_features NOT LIKE "%commentaries%" \
GROUP BY f.title \
ORDER BY count DESC;'

q11 = 'CREATE VIEW Trailers AS \
SELECT f.title, f.special_features, COUNT(*) as count \
FROM film as f \
LEFT JOIN inventory as i ON i.film_id = f.film_id \
LEFT JOIN rental as r ON r.inventory_id = i.inventory_id \
WHERE special_features LIKE "%trailers%" \
GROUP BY f.title \
ORDER BY count DESC;'

q12 = 'CREATE VIEW NoTrailers AS \
SELECT f.title, f.special_features, COUNT(*) as count \
FROM film as f \
LEFT JOIN inventory as i ON i.film_id = f.film_id \
LEFT JOIN rental as r ON r.inventory_id = i.inventory_id \
WHERE special_features NOT LIKE "%trailers%" \
GROUP BY f.title \
ORDER BY count DESC;'

# execute SQL query using execute() method.
query = [q1, q2, q3, q4, q5, q6, q7, q8, q9, q10, q11, q12]
for q in query:
    cursor.execute(q)

# disconnect from server
db.close()



q5_4 = '(SELECT "BehindTheScenes" as label, AVG(count) \
        FROM BehindTheScenes) \
        UNION ALL \
        (SELECT "NoBehindTheScenes" as label, AVG(count) \
        FROM NoBehindTheScenes) \
        UNION ALL \
        (SELECT "Commentaries" as label, AVG(count) \
        FROM Commentaries) \
        UNION ALL \
        (SELECT "NoCommentaries" as label, AVG(count) \
        FROM NoCommentaries) \
        UNION ALL \
        (SELECT "Trailers" as label, AVG(count) \
        FROM Trailers) \
        UNION ALL \
        (SELECT "NoTrailers" as label, AVG(count) \
        FROM NoTrailers);'
df = qry(q5_4)
```

In [36]:
```
df
```

Out[36]:

	label	AVG(count)
0	BehindTheScenes	16.0446
1	NoBehindTheScenes	16.1364
2	Commentaries	16.2041
3	NoCommentaries	15.9501
4	Trailers	15.9738
5	NoTrailers	16.2172

In [38]:
```
fig = px.bar(df, x='label', y='AVG(count)')
fig.show()
```



In [ ]:
```
# Insight:
# From the plot we could see that the average rental count for the movies with or without a specific special
# features listed above are about the same.
# This might indicate that these special features would not be a relatively important factor for people
# when making rental decisions.
```

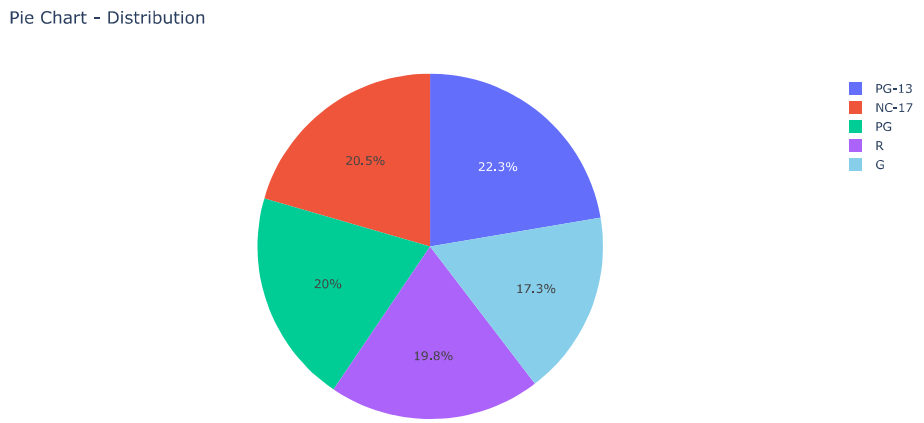
In [51]:
```
# 5. Check the rental count for each rating
q5_5 = "SELECT f.rating, COUNT(DISTINCT(r.rental_id)) as count \
FROM film as f \
LEFT JOIN inventory as i on f.film_id = i.film_id \
LEFT JOIN rental as r on i.inventory_id = r.inventory_id \
GROUP BY f.rating;"

df = qry(q5_5)
```

In [52]:
```
df
```

Out[52]:

	rating	count
0	G	2773
1	PG	3212
2	PG-13	3585
3	R	3181
4	NC-17	3293

In [53]:
```
labels=df['rating']
values=df['count']

trace=go.Pie(labels=labels,values=values, marker=dict(colors=['skyblue']),hoverinfo="value")
data = [trace]
layout = go.Layout(title="Pie Chart - Distribution")
fig = go.Figure(data = data,layout = layout)

iplot(fig)
```



In [ ]:
```
# Insight:
# From the pie chart we could see that the rating PG-13 has the highest rental count while the rating
# G has the lowest rental count.
# This might indicate that more customers prefer to watch film with rating PG-13 while less customers
# are willing to rent movie in rating G,
# which makes sense since films with rating G would contain some elements that are not
# well-accepted by the public or the younger customers.
# But overall, the difference in the rental count of each rating is not significant.
```