# CSDA 1120 - Group 4 - Sprint 1 - Scenario 1

## 1.0 PREPARING SYSTEM

### Import SQLite, Pandas and Pretty Print into the notebook

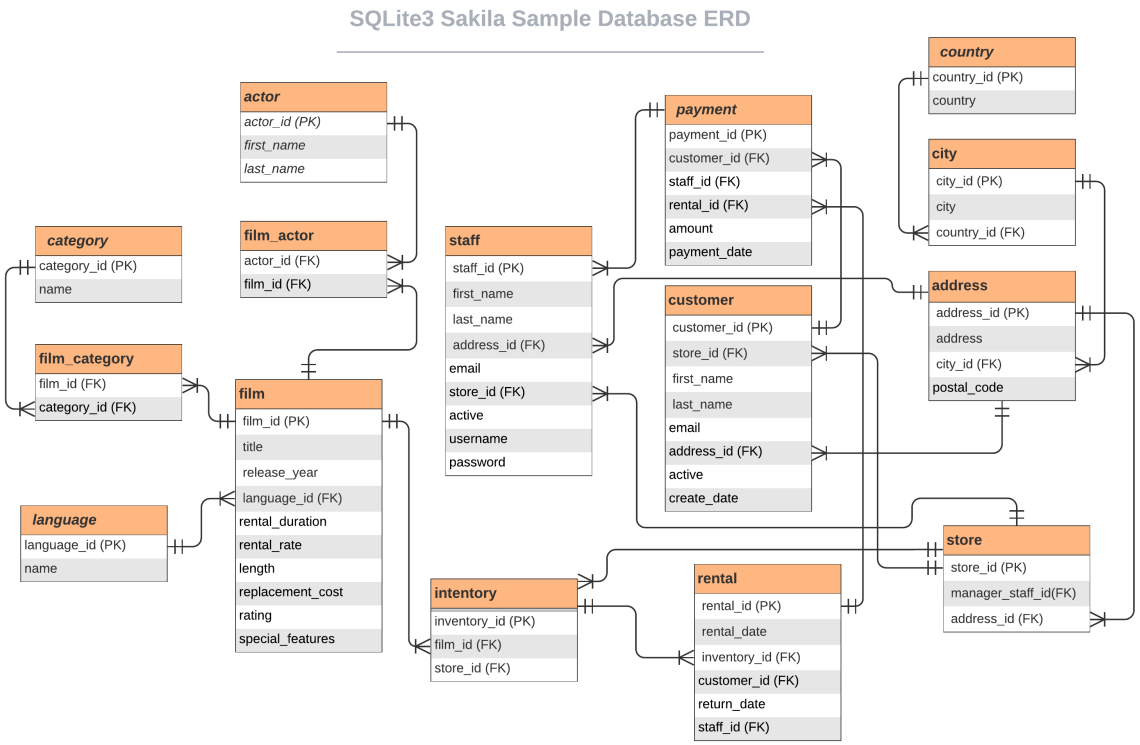
In [22]:
```
import sqlite3
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import locale
locale.setlocale(locale.LC_ALL, 'en_US')
import warnings
warnings.filterwarnings("ignore")
```

### Setting the connection information and a cursor object to interact with the database

### The data base being referred for the analysis is stored as a file "sqlite-sakila.db'

In [2]:
```
conn = sqlite3.connect('sqlite-sakila.db')
cur = conn.cursor()
```

### Reviewing the Sakila Database Schema below. It contains data for a Video Rental Store Company




## Setting Bold Variables

In [3]:
```
start = "\033[1m"
end = "\033[0;0m"
```

# 1.1 DATA EXPLORATION

## 1.1.1 Totals

### 1.1.1.1 Revenue Generated by the Video Rental Company

In [4]:
```
df_revenue = pd.read_sql("""

    SELECT   SUM(amount) AS "Total Revenue"

    FROM     payment

""", conn)

print("Total Revenue Generated by the Video Rental Company is: ", start +
      locale.currency(round(df_revenue["Total Revenue"][0],2), grouping=True, symbol=T
```

```
Total Revenue Generated by the Video Rental Company is:  $67,416.51
```

### 1.1.1.2 Number of Films

In [5]:
```
df_numfilms = pd.read_sql("""
```

```
    SELECT   COUNT(film_id) AS "Total Number of Films"

    FROM     film

""", conn)

print("Total Number of Films for Rental are: ", start + df_numfilms["Total Number of F
```

Total Number of Films for Rental are:  **1000**

#### 1.1.1.3 Number of Rentals

In [6]:
```
df_numrentals = pd.read_sql("""

    SELECT   COUNT(rental_id) AS "Total Number of Rentals"

    FROM     rental

""", conn)

print("Total Number of Rental Made are: ", start + df_numrentals["Total Number of Rent
```

Total Number of Rental Made are:  **16044**

#### 1.1.1.4 Number of Customers

In [7]:
```
df_numcustomers = pd.read_sql("""

    SELECT   COUNT(customer_id) AS "Total Number of Customers"

    FROM     customer

""", conn)

print("Total Number of Customers are: ", start + df_numcustomers["Total Number of Cust
```

Total Number of Customers are:  **599**

#### 1.1.1.5 Number of Stores

In [8]:
```
df_numstores = pd.read_sql("""

    SELECT   COUNT(store_id) AS "Total Number of Stores"

    FROM     store

""", conn)

print("Total Number of Stores are: ", start + df_numstores["Total Number of Stores"][0
```

Total Number of Stores are:  **2**

#### 1.1.1.6 Number of Staff Members

In [9]:
```
df_numstaff = pd.read_sql("""

    SELECT   COUNT(staff_id) AS "Total Number of Staff"
```

```
    FROM     staff

""", conn)

print("Total Number of Staff Members are: ", start + df_numstaff["Total Number of Staf
```

```
Total Number of Staff Members are:  2
```

#### 1.1.1.7 Number of Countries Reached

In [10]:
```
df_numcountries = pd.read_sql("""

    SELECT   COUNT(country_id) AS "Total Number of Countries"

    FROM     country

""", conn)

print("Total Number of Countries Reached are: ", start + df_numcountries["Total Number
```

```
Total Number of Countries Reached are:  109
```

#### 1.1.1.8 Number of Cities Reached

In [11]:
```
df_numcity = pd.read_sql("""

    SELECT   COUNT(city_id) AS "Total Number of City"

    FROM     city

""", conn)

print("Total Number of Cities Reached are: ", start + df_numcity["Total Number of City
```

```
Total Number of Cities Reached are:  600
```

### 1.1.2 Highest and Least

#### 1.1.2.1 Highest and Least Grossing Actor

In [12]:
```
## Highest Grossing Actor
df_bestactor = pd.read_sql("""

    SELECT  a.first_name || ' ' || a.last_name AS Name, a.actor_id AS ActorID, SUM(p.a

    FROM    actor AS a, film_actor AS fa, film AS f, inventory as i, rental as r, paym

    WHERE   a.actor_id = fa.actor_id
            AND fa.film_id = f.film_id
            AND f.film_id = i.film_id
            AND i.inventory_id = r.inventory_id
            AND r.rental_id = p.rental_id

    GROUP BY a.actor_id

    ORDER BY SUM(p.amount) DESC

    LIMIT 1
```

```
""", conn)

print("Highest Grossing Actor is: ", start + df_bestactor["Name"][0] + end, " with Tot
      start + locale.currency(round(df_bestactor["Amount"][0],2), grouping=True, symbo

## Least Grossing Actor
df_worstactor = pd.read_sql("""

    SELECT  a.first_name || ' ' || a.last_name AS Name, a.actor_id AS ActorID, SUM(p.a

    FROM    actor AS a, film_actor AS fa, film AS f, inventory as i, rental as r, paym

    WHERE   a.actor_id = fa.actor_id
            AND fa.film_id = f.film_id
            AND f.film_id = i.film_id
            AND i.inventory_id = r.inventory_id
            AND r.rental_id = p.rental_id

    GROUP BY a.actor_id

    ORDER BY SUM(p.amount) ASC

    LIMIT 1

""", conn)

print("Least Grossing Actor is: ", start + df_worstactor["Name"][0] + end, " with Tota
      start + locale.currency(round(df_worstactor["Amount"][0],2), grouping=True, symb
```

Highest Grossing Actor is:  **GINA DEGENERES**  with Total Revenue for Business being : **$3,442.49**
Least Grossing Actor is:  **EMILY DEE**  with Total Revenue for Business being : **$883.85**

### 1.1.2.2 Highest and Least Performing Genre

In [13]:
```
## Highest Performing Genre
df_bestgenre = pd.read_sql("""

    SELECT  c.name AS Genre, c.category_id AS CategoryID, SUM(p.amount) AS Amount

    FROM    category AS c, film_category AS fc, film AS f, inventory as i, rental as r

    WHERE   c.category_id = fc.category_id
            AND fc.film_id = f.film_id
            AND f.film_id = i.film_id
            AND i.inventory_id = r.inventory_id
            AND r.rental_id = p.rental_id

    GROUP BY c.category_id

    ORDER BY SUM(p.amount) DESC

    LIMIT 1

""", conn)

print("Highest Performing Genre is: ", start + df_bestgenre["Genre"][0] + end, " with
      start + locale.currency(round(df_bestgenre["Amount"][0],2), grouping=True, symbo
```

```
## Least Performing Genre
df_worstgenre = pd.read_sql("""

    SELECT  c.name AS Genre, c.category_id AS CategoryID, SUM(p.amount) AS Amount

    FROM    category AS c, film_category AS fc, film AS f, inventory as i, rental as r

    WHERE   c.category_id = fc.category_id
            AND fc.film_id = f.film_id
            AND f.film_id = i.film_id
            AND i.inventory_id = r.inventory_id
            AND r.rental_id = p.rental_id

    GROUP BY c.category_id

    ORDER BY SUM(p.amount) ASC

    LIMIT 1

""", conn)

print("Least Performing Genre is: ", start + df_worstgenre["Genre"][0] + end, " with T
      start + locale.currency(round(df_worstgenre["Amount"][0],2), grouping=True, symb
```

```
Highest Performing Genre is:  Sports  with Total Revenue for Business being : $5,314.
21
Least Performing Genre is:  Music  with Total Revenue for Business being : $3,417.72
```

## 1.2 BUSINESS OBJECTIVES

1. Reward the top 10 most loyal active customers with the highest value of purchases made to date.
2. Improve sales and revenue from the 5 best and worst performing movies.

## 1.3 DECISIONS

1. Select the top 10 active customers who should be rewarded during the program and the form of reward to be given to loyal customers.
2. Select the 10 worst performing movies to be offered for clearance and the amount of discount to be given.
3. Select the 10 best performing movies for which premium needs to be charged and decide on the amount of premium.

## 1.4 BUSINESS QUESTIONS

1. What is the first name, last name, and email address of the top 10 most loyal active customers with the highest value of purchases made to date to whom 40% discount on the next purchase made is to be given (maximum discount $50)?

2. What is the title of the 10 worst performing movies on which 40% discount is to be offered?
3. What is the title of the 10 best performing movies on which 40% premium is to be charged?

# 1.5 DATABASE QUERIES

## 1.5.1 Identify 10 Highest Spending Customers

```
    SELECT   c.customer_id AS "Customer ID", c.first_name AS "First
Name", c.last_name AS "Last Name",
        c.email AS "Email ID", SUM(p.amount) as "Total Spend",
Count(p.amount) as "Number of Transactions"

    FROM     customer as c, payment as p

    WHERE    c.customer_id = p.customer_id
             AND   c.active = 1

    GROUP BY c.customer_id

    ORDER by SUM(p.amount) DESC

    LIMIT 10
```

## 1.5.2 Identify 10 Worst Performing Movies

```
    SELECT f.film_id AS "Film ID", f.title AS "Film Title",
SUM(p.amount) AS "Total Earnings",
        f.rental_rate AS "Existing Rental Rate", ROUND(f.rental_rate
* 0.6, 2) AS "New Rental Rate", c.name as Genre

    FROM film as f, inventory as i, rental as r, payment as p,
film_category as fc, category as c

    WHERE f.film_id = i.film_id
            AND i.inventory_id = r.inventory_id
            AND r.rental_id = p.rental_id
            AND f.film_id = fc.film_id
            AND fc.category_id = c.category_id

    GROUP BY f.film_id

    ORDER BY SUM(p.amount) ASC

    LIMIT 10
```

### 1.5.3 Identify 10 Best Performing Movies

```
SELECT f.film_id AS "Film ID", f.title AS "Film Title", SUM(p.amount)
AS "Total Earnings",
    f.rental_rate AS "Existing Rental Rate", ROUND(f.rental_rate *
1.4, 2) AS "New Rental Rate", c.name as Genre

FROM film as f, inventory as i, rental as r, payment as p,
film_category as fc, category as c

WHERE f.film_id = i.film_id
        AND i.inventory_id = r.inventory_id
        AND r.rental_id = p.rental_id
        AND f.film_id = fc.film_id
        AND fc.category_id = c.category_id

GROUP BY f.film_id

ORDER BY SUM(p.amount) DESC

LIMIT 10
```

## 1.6 ANALYSIS

### Creating Dataframe (df_bestcustomers) to Output result from Query to Question 1

In [14]:
```
df_bestcustomers = df = pd.read_sql("""

SELECT   c.customer_id AS "Customer ID", c.first_name AS "First Name", c.last_name AS
           c.email AS "Email ID", SUM(p.amount) as "Total Spend", Count(p.amount) as

FROM     customer as c, payment as p

WHERE    c.customer_id = p.customer_id
         AND   c.active = 1

GROUP BY c.customer_id

ORDER by SUM(p.amount) DESC

LIMIT 10

""", conn)
```

### Creating Dataframe (df_worstmovies) to Output result from Query to Question 2

In [15]:
```
df_worstmovies = df = pd.read_sql("""

    SELECT f.film_id AS "Film ID", f.title AS "Film Title", SUM(p.amount) AS "Total Ea
```

```
            f.rental_rate AS "Existing Rental Rate", ROUND(f.rental_rate * 0.6, 2) AS

    FROM film as f, inventory as i, rental as r, payment as p, film_category as fc, ca

    WHERE f.film_id = i.film_id
            AND i.inventory_id = r.inventory_id
            AND r.rental_id = p.rental_id
            AND f.film_id = fc.film_id
            AND fc.category_id = c.category_id

    GROUP BY f.film_id

    ORDER BY SUM(p.amount) ASC

    LIMIT 10

""", conn)
```

### Creating Dataframe (df_bestmovies) to Output result from Query to Question 3

In [16]:
```
df_bestmovies = df = pd.read_sql("""

    SELECT f.film_id AS "Film ID", f.title AS "Film Title", SUM(p.amount) AS "Total Ea
        f.rental_rate AS "Existing Rental Rate", ROUND(f.rental_rate * 1.4, 2) AS "New

    FROM film as f, inventory as i, rental as r, payment as p, film_category as fc, ca

    WHERE f.film_id = i.film_id
            AND i.inventory_id = r.inventory_id
            AND r.rental_id = p.rental_id
            AND f.film_id = fc.film_id
            AND fc.category_id = c.category_id

    GROUP BY f.film_id

    ORDER BY SUM(p.amount) DESC

    LIMIT 10

""", conn)
```

# 1.7 RESULT

## 1.7.1 10 Best Customers

### List of 10 Highest Spending Customers to whom a Personalized Reward Email containing 40% discount coupon is to be Sent

In [17]:
```
df_bestcustomers
```

Out[17]:

| | Customer ID | First Name | Last Name | Email ID | Total Spend | Number of Transactions |
|---|---|---|---|---|---|---|
| **0** | 526 | KARL | SEAL | KARL.SEAL@sakilacustomer.org | 221.55 | 45 |
| **1** | 148 | ELEANOR | HUNT | ELEANOR.HUNT@sakilacustomer.org | 216.54 | 46 |
| **2** | 144 | CLARA | SHAW | CLARA.SHAW@sakilacustomer.org | 195.58 | 42 |
| **3** | 137 | RHONDA | KENNEDY | RHONDA.KENNEDY@sakilacustomer.org | 194.61 | 39 |
| **4** | 178 | MARION | SNYDER | MARION.SNYDER@sakilacustomer.org | 194.61 | 39 |
| **5** | 459 | TOMMY | COLLAZO | TOMMY.COLLAZO@sakilacustomer.org | 186.62 | 38 |
| **6** | 469 | WESLEY | BULL | WESLEY.BULL@sakilacustomer.org | 177.60 | 40 |
| **7** | 468 | TIM | CARY | TIM.CARY@sakilacustomer.org | 175.61 | 39 |
| **8** | 236 | MARCIA | DEAN | MARCIA.DEAN@sakilacustomer.org | 175.58 | 42 |
| **9** | 181 | ANA | BRADLEY | ANA.BRADLEY@sakilacustomer.org | 174.66 | 34 |

## Plotting and Highlighting Top 10 Customers

### Creating Dataframe (df_customers) with details of all Customers

In [18]:

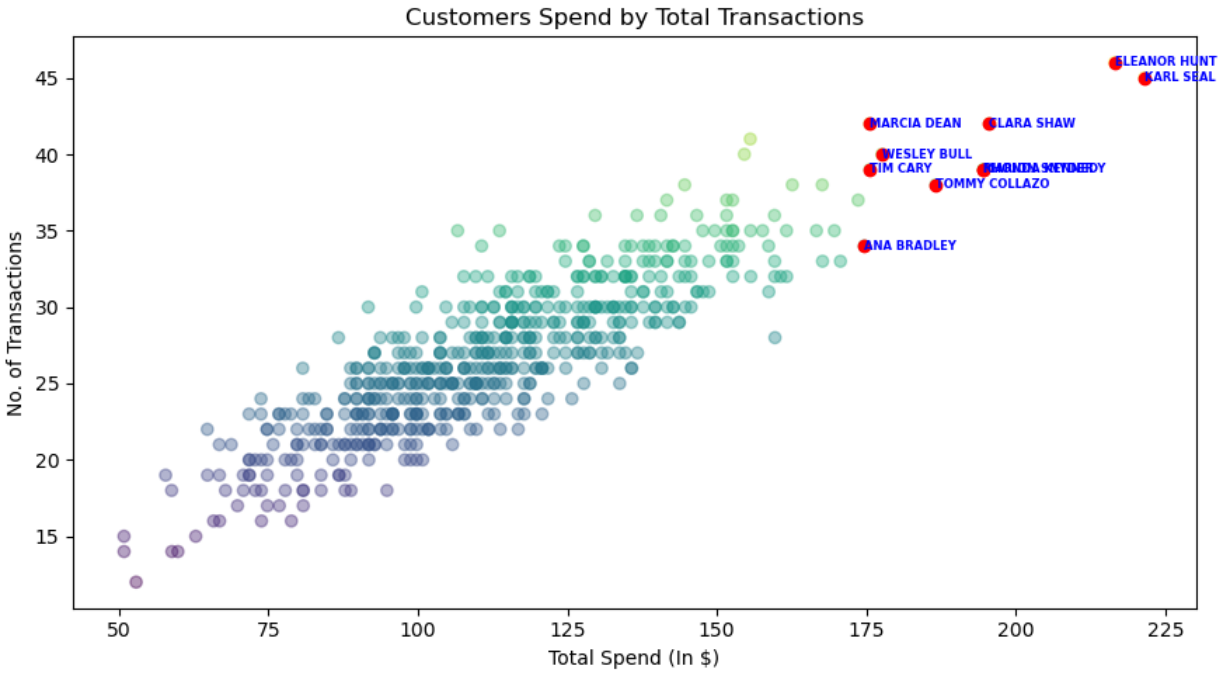
```
df_customers = df = pd.read_sql("""

SELECT   c.customer_id AS "Customer ID", c.first_name AS "First Name", c.last_name AS
           c.email AS "Email ID", SUM(p.amount) as "Total Spend", Count(p.amount) as

FROM     customer as c, payment as p

WHERE    c.customer_id = p.customer_id
         AND   c.active = 1

GROUP BY c.customer_id

ORDER by SUM(p.amount) DESC

""", conn)
```

### Creating Plot

In [28]:

```
plt.figure(figsize=(9,5))
plt.scatter(df_customers["Total Spend"], df_customers["Number of Transactions"],
    alpha=0.4, c=df_customers["Number of Transactions"], cmap='viridis')
# set x-axis label and specific size
plt.xlabel('Total Spend (In $)',size=10)
# set y-axis label and specific size
plt.ylabel('No. of Transactions',size=10)
# set plot title with specific size
plt.title('Customers Spend by Total Transactions',size=12)

# plot top 10 customers
plt.scatter(df_bestcustomers["Total Spend"], df_bestcustomers["Number of Transactions'

# loop through to annotate top 10 customers
```

```
for i in range(df_bestcustomers.shape[0]):
    plt.annotate((df_bestcustomers["First Name"] +" "+ df_bestcustomers["Last Name"]).
                 (df_bestcustomers["Total Spend"].tolist()[i], df_bestcustomers["Numbe
                 color='Blue', weight='bold', ha='left', va='center', size=6)

plt.tight_layout()

# plot
plt.show()
```




## 1.7.2 10 Worst Movies

### List of 10 Worst Revenue Making Movies on which 40% Discount is to be Offered

In [30]:
```
df_worstmovies
```

Out[30]:

| | Film ID | Film Title | Total Earnings | Existing Rental Rate | New Rental Rate | Genre |
|---|---|---|---|---|---|---|
| **0** | 635 | OKLAHOMA JUMANJI | 5.94 | 0.99 | 0.59 | New |
| **1** | 885 | TEXAS WATCH | 5.94 | 0.99 | 0.59 | Horror |
| **2** | 335 | FREEDOM CLEOPATRA | 5.95 | 0.99 | 0.59 | Comedy |
| **3** | 261 | DUFFEL APOCALYPSE | 6.93 | 0.99 | 0.59 | Documentary |
| **4** | 996 | YOUNG LANGUAGE | 6.93 | 0.99 | 0.59 | Documentary |
| **5** | 718 | REBEL AIRPORT | 7.92 | 0.99 | 0.59 | Music |
| **6** | 196 | CRUELTY UNFORGIVEN | 7.93 | 0.99 | 0.59 | Classics |
| **7** | 475 | JAPANESE RUN | 7.94 | 0.99 | 0.59 | Horror |
| **8** | 839 | STALLION SUNDANCE | 8.93 | 0.99 | 0.59 | Sci-Fi |
| **9** | 401 | HAROLD FRENCH | 9.92 | 0.99 | 0.59 | Drama |

## Plotting Genres of Worst 10 Movies

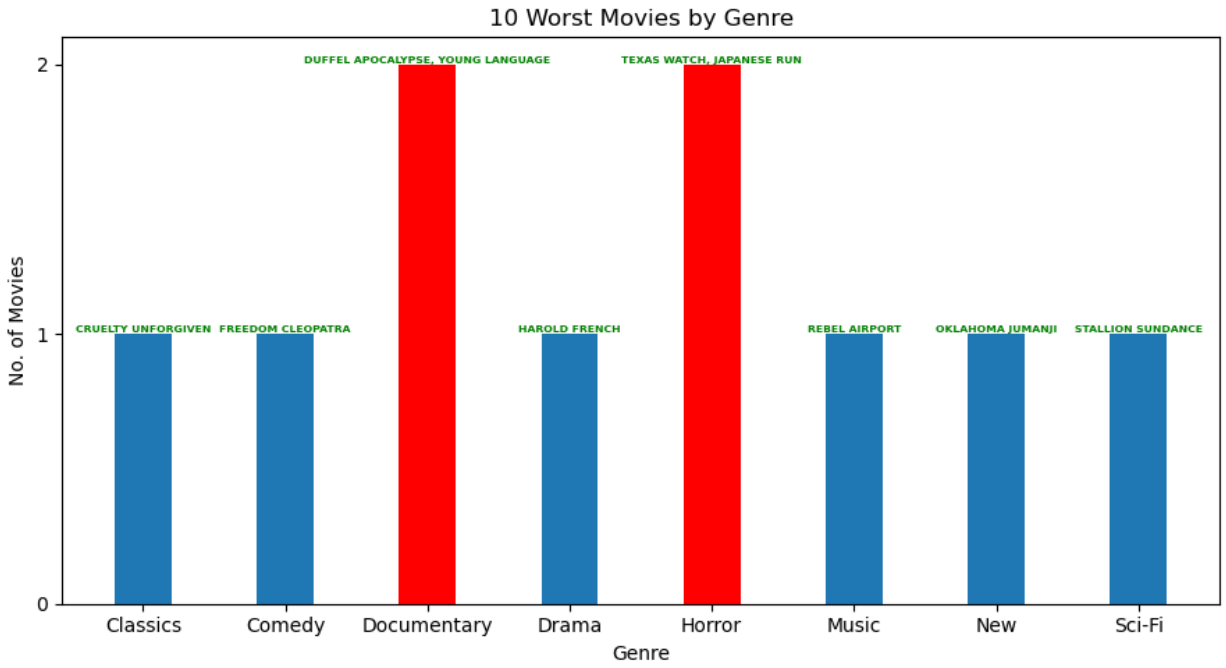
### Creating Dataframe (df_worstmovies_detail) with expanded details of 10 worst movies

In [31]:

```
df_worstmovies_detail = df = pd.read_sql("""

WITH worst_movies(id, title, earning) AS
    (SELECT f.film_id as id, f.title as title, SUM(p.amount) as earning

    FROM film as f, inventory as i, rental as r, payment as p

    WHERE f.film_id = i.film_id
        AND i.inventory_id = r.inventory_id
        AND r.rental_id = p.rental_id

    GROUP BY f.film_id

    ORDER BY SUM(p.amount) ASC

    LIMIT 10)

SELECT w.id, w.title, c.name as genre, count(w.id) as genrewise_count

FROM worst_movies as w, film_category as fc, category as c

WHERE w.id = fc.film_id
        AND fc.category_id = c.category_id

GROUP BY c.name

""", conn)
```

### Creating Plot

In [43]:
```
plt.figure(figsize=(9,5))
plt.bar(df_worstmovies_detail["genre"], df_worstmovies_detail["genrewise_count"], widt
# set x-axis label and specific size
plt.xlabel('Genre',size=10)
# set y-axis label and specific size
plt.locator_params(axis='y', nbins=3)
plt.ylabel('No. of Movies',size=10)
# set plot title with specific size
plt.title('10 Worst Movies by Genre',size=12)

# loop through to annotate worst 10 movies
for i in range(df_worstmovies_detail.shape[0]):
    if(df_worstmovies_detail["genrewise_count"].tolist()[i]>1):
        plt.bar(df_worstmovies_detail["genre"][i], df_worstmovies_detail["genrewise_co
        a=""
        for j in range (df_worstmovies.shape[0]):
            if(df_worstmovies["Genre"][j] == df_worstmovies_detail["genre"][i]):
                a = a + ", " + df_worstmovies["Film Title"][j]
        a = a[2:]
        df_worstmovies_detail["title"][i] = a
    plt.annotate(df_worstmovies_detail["title"].tolist()[i],
        (df_worstmovies_detail["genre"].tolist()[i], df_worstmovies_detail["genrewise_
        color='Green', weight='bold', ha='center', va='bottom', size=5.5)

plt.tight_layout()

# plot
plt.show()
```




## 1.7.3 10 Best Movies

### List of 10 Best Revenue Making Movies on which 40% Premium is to be Charged

In [44]:
```
df_bestmovies
```

Out[44]:

| | Film ID | Film Title | Total Earnings | Existing Rental Rate | New Rental Rate | Genre |
|---|---|---|---|---|---|---|
| **0** | 879 | TELEGRAPH VOYAGE | 231.73 | 4.99 | 6.99 | Music |
| **1** | 973 | WIFE TURN | 223.69 | 4.99 | 6.99 | Documentary |
| **2** | 1000 | ZORRO ARK | 214.69 | 4.99 | 6.99 | Comedy |
| **3** | 369 | GOODFELLAS SALUTE | 209.69 | 4.99 | 6.99 | Sci-Fi |
| **4** | 764 | SATURDAY LAMBS | 204.72 | 4.99 | 6.99 | Sports |
| **5** | 893 | TITANS JERK | 201.71 | 4.99 | 6.99 | Sci-Fi |
| **6** | 897 | TORQUE BOUND | 198.72 | 4.99 | 6.99 | Drama |
| **7** | 403 | HARRY IDAHO | 195.70 | 4.99 | 6.99 | Drama |
| **8** | 460 | INNOCENT USUAL | 191.74 | 4.99 | 6.99 | Foreign |
| **9** | 444 | HUSTLER PARTY | 190.78 | 4.99 | 6.99 | Comedy |

## Plotting Genres of Best 10 Movies

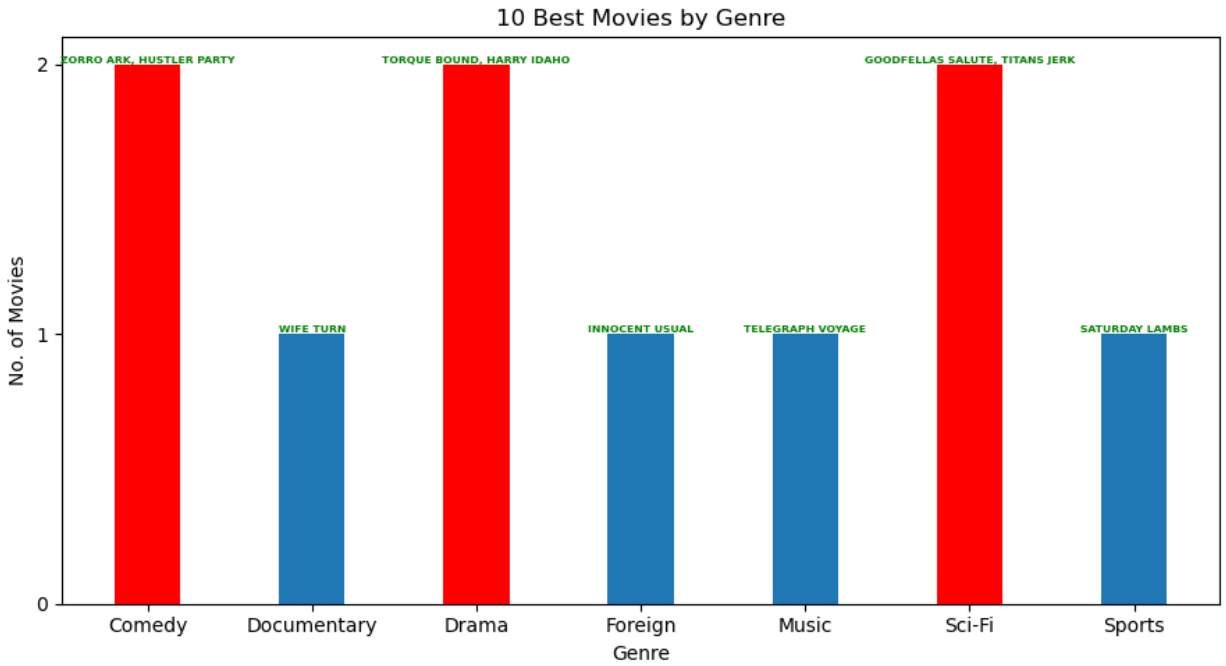
### Creating Dataframe (df_bestmovies_detail) with expanded details of 10 best movies

In [45]:

```
df_bestmovies_detail = df = pd.read_sql("""

WITH best_movies(id, title, earning) AS
    (SELECT f.film_id as id, f.title as title, SUM(p.amount) as earning

    FROM film as f, inventory as i, rental as r, payment as p

    WHERE f.film_id = i.film_id
        AND i.inventory_id = r.inventory_id
        AND r.rental_id = p.rental_id

    GROUP BY f.film_id

    ORDER BY SUM(p.amount) DESC

    LIMIT 10)

SELECT b.id, b.title, c.name as genre, count(b.id) as genrewise_count

FROM best_movies as b, film_category as fc, category as c

WHERE b.id = fc.film_id
        AND fc.category_id = c.category_id

GROUP BY c.name

""", conn)
```

### Creating Plot

In [46]:
```
plt.figure(figsize=(9,5))
plt.bar(df_bestmovies_detail["genre"], df_bestmovies_detail["genrewise_count"], width
# set x-axis label and specific size
plt.xlabel('Genre',size=10)
# set y-axis label and specific size
plt.locator_params(axis='y', nbins=3)
plt.ylabel('No. of Movies',size=10)
# set plot title with specific size
plt.title('10 Best Movies by Genre',size=12)

# loop through to annotate top 10 Movies
for i in range(df_bestmovies_detail.shape[0]):
    if(df_bestmovies_detail["genrewise_count"].tolist()[i]>1):
        plt.bar(df_bestmovies_detail["genre"][i], df_bestmovies_detail["genrewise_cour
        a=""
        for j in range (df_bestmovies.shape[0]):
            if(df_bestmovies["Genre"][j] == df_bestmovies_detail["genre"][i]):
                a = a + ", " + df_bestmovies["Film Title"][j]
        a = a[2:]
        df_bestmovies_detail["title"][i] = a
    plt.annotate(df_bestmovies_detail["title"].tolist()[i],
        (df_bestmovies_detail["genre"].tolist()[i], df_bestmovies_detail["genrewise_co
        color='Green', weight='bold', ha='center', va='bottom', size=5.5)

plt.tight_layout()

# plot
plt.show()
```




## 1.8 CONCLUSION

During the project, the following three business tasks were undertaken.

1. Identify Top 10 Active Customers
2. Identify Worst 10 Performing Movies

3. Identify Top 10 Performing Movies

In line with the business objectives, it is recommended that the Top 10 Active Customers, as listed in Section-1.7.1, be sent a personalized Email with the 40% discount coupon.

Further, it is recommended that the New Rental Rates, as listed in Section 1.7.2 and 1.7.3, for the worst 10 movies and the best 10 movies respectively be made effective on the rentals henceforth.

In [ ]:
```
# print script output to html
!jupyter nbconvert CSDA_1120-Group_4-Scenario_1.ipynb --to 'html'
```

# Group4-Scenario2

May 27, 2023

## 1 CSDA 1120 - Group 4 - Sprint 1 - Scenario 2

### 1.1 2.0 PREPARING SYSTEM

[4]:
```
import sqlite3
import pandas as pd
from pandas import *
!pip install mysql.connector
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Collecting mysql.connector
  Downloading mysql-connector-2.2.9.tar.gz (11.9 MB)
                             11.9/11.9 MB
39.8 MB/s eta 0:00:00
  Preparing metadata (setup.py) … done
Building wheels for collected packages: mysql.connector
  Building wheel for mysql.connector (setup.py) … done
  Created wheel for mysql.connector:
filename=mysql_connector-2.2.9-cp310-cp310-linux_x86_64.whl size=247953
sha256=3bc8cd03a058155a73259c46ba99bd7d9431946058fb231dc5f77daf0e837206
  Stored in directory: /root/.cache/pip/wheels/76/48/9b/da67ff1a18fe8e9d428f9b1a
177716d4a7d363d2bbe83bf6cf
Successfully built mysql.connector
Installing collected packages: mysql.connector
Successfully installed mysql.connector-2.2.9
```

[9]:
```
# Connect to the MYSQL Server and show tables
cnx = mysql.connector.connect(
    user='root',
    password='',
    host='34.121.31.247',
    database='restaurants'
)

# Create a cursor object
cursor = cnx.cursor()
```

```
# Execute the SHOW TABLES query
cursor.execute("SHOW TABLES")

# Fetch all table names
tables = cursor.fetchall()

# Print the table names
for table in tables:
    print(table[0])
```

```
bytearray(b'family')
bytearray(b'market')
bytearray(b'measures')
bytearray(b'population')
bytearray(b'product')
bytearray(b'productdim')
bytearray(b'region')
bytearray(b'sales')
bytearray(b'salesfact')
bytearray(b'scenario')
bytearray(b'supplier')
```

[ ]:
```
# Create a cursor object
cursor = cnx.cursor()

# Execute the SHOW TABLES query
cursor.execute("SHOW TABLES")

# Fetch all table names
tables = cursor.fetchall()

# Convert byte arrays to strings and print the table names
for table in tables:
    table_name = table[0].decode('utf-8')
    print(table_name)
```

```
family
market
measures
population
product
productdim
region
sales
salesfact
scenario
supplier
```

## 1.2 2.1 DATA EXPLORATION

[12]:
```python
# Create a cursor object
cursor = cnx.cursor()

# Get the table names
query = "SHOW TABLES"
cursor.execute(query)

# Fetch the table names
tables = cursor.fetchall()

# Print tables with columns
for table in tables:
    table_name = table[0].decode('utf-8')  # Decode bytes to string
    print("Table:", table_name)

    # Get the column names for the current table
    query = f"SHOW COLUMNS FROM `{table_name}`"  # Use backticks for table name
    cursor.execute(query)

    # Fetch the column names
    columns = cursor.fetchall()

    # Create a list of column names
    column_names = [column[0] for column in columns]

    # Print the column names in a single line
    print("  Columns:", ", ".join(column_names))

    print()  # Add a newline after each table
```

```
Table: family
  Columns: FAMILYID, FAMILY, FAMILY_ALIAS, INTRODATE

Table: market
  Columns: STATEID, REGIONID, STATE, UDAMKTSIZE, UDAMKTTYPE, POPULATIONID

Table: measures
  Columns: SORTKEY, MEASURESID, PARENT, CHILD, MEASURES_ALIAS, CONSOLIDATION,
TWOPASSCALC, STORAGE, VARIANCEREPORTING, TIMEBALANCE, SKIP, UDA, FORMULA,
COMMENT_ESSBASE

Table: population
  Columns: POPULATIONID, POPGROUP, POPULATION, POPULATION_ALIAS

Table: product
  Columns: PRODUCTID, FAMILYID, SKU, SKU_ALIAS, CAFFEINATED, OUNCES, PKGTYPE,
```

```
INTRODATE

Table: productdim
  Columns: FAMILY, FAMILY_ALIAS, CONSOLIDATION, SKU, SKU_ALIAS

Table: region
  Columns: REGIONID, REGION, UDA, DIRECTOR

Table: sales
  Columns: STATEID, PRODUCTID, SCENARIOID, MEASURESID, SUPPLIERID, TRANSDATE,
AMOUNT

Table: salesfact
  Columns: STATEID, PRODUCTID, SCENARIOID, SUPPLIERID, TRANSDATE, SALES, COGS,
MARKETING, PAYROLL, MISC, OPENINGINVENTORY, ADDITIONS

Table: scenario
  Columns: SCENARIOID, SCENARIO, CONSOLIDATION

Table: supplier
  Columns: SUPPLIERID, SUPPLIER_ALIAS, ADDRESS, CITY, STATE, ZIP, COUNTRY
```

[ ]:
```python
# Create a cursor object
cursor = cnx.cursor()

# Execute the query to fetch the top 5 highest sold products
query = "SELECT SKU_ALIAS, SALES FROM salesfact \
         JOIN product ON salesfact.PRODUCTID = product.PRODUCTID \
         ORDER BY SALES DESC LIMIT 5"

# Execute the query
cursor.execute(query)

# Fetch the results
results = cursor.fetchall()

# Print the top 5 highest sold products
print("Top 5 highest sold products:")
for result in results:
    product_name = result[0]
    sales = result[1]
    print("Product:",product_name)
    print("Sales:", sales)
```

```
Top 5 highest sold products:
Product: Cola
Sales: 1130.0
```

```
Product: Cola
Sales: 910.0
Product: Cola
Sales: 902.4
Product: Cola
Sales: 866.4
Product: Cola
Sales: 860.0
```

[13]:
```python
# Create a cursor object
cursor = cnx.cursor()

# Execute the query to fetch the products with the highest sales in each family
query = "SELECT f.FAMILY, p.SKU_ALIAS, MAX(s.SALES) AS MaxSales \
         FROM salesfact s \
         JOIN product p ON s.PRODUCTID = p.PRODUCTID \
         JOIN productdim pd ON p.SKU_ALIAS = pd.SKU_ALIAS \
         JOIN family f ON pd.FAMILY = f.FAMILY \
         GROUP BY f.FAMILY, p.SKU_ALIAS"

# Execute the query
cursor.execute(query)

# Fetch the results
results = cursor.fetchall()

# Print the products with the highest sales in each family
print("Products with highest sales in each family:")
for result in results:
    family = result[0]
    product_name = result[1]
    max_sales = result[2]

    # Display the information in a compact format
    print(f"Family: {family}, Product: {product_name}, Max Sales: {max_sales}")
    print()
```

```
Products with highest sales in each family:
Family: 100, Product: Cola, Max Sales: 1130.0

Family: 100, Product: Diet Cola, Max Sales: 390.0

Family: 100, Product: Caffeine Free Cola, Max Sales: 310.0

Family: 200, Product: Old Fashioned, Max Sales: 690.0

Family: 200, Product: Diet Root Beer, Max Sales: 687.0
```

```
Family: 200, Product: Sasparilla, Max Sales: 532.0

Family: 200, Product: Birch Beer, Max Sales: 765.0

Family: 300, Product: Dark Cream, Max Sales: 678.0

Family: 300, Product: Vanilla Cream, Max Sales: 376.0

Family: 300, Product: Diet Cream, Max Sales: 678.0

Family: 400, Product: Grape, Max Sales: 687.0

Family: 400, Product: Orange, Max Sales: 654.0

Family: 400, Product: Strawberry, Max Sales: 282.0
```

[ ]:
```python
# Create a cursor object
cursor = cnx.cursor()

# Execute the query to fetch the top sales by supplier
query = "SELECT su.SUPPLIER_ALIAS, SUM(sf.SALES) AS TotalSales \
         FROM salesfact sf \
         JOIN supplier su ON sf.SUPPLIERID = su.SUPPLIERID \
         GROUP BY su.SUPPLIER_ALIAS \
         ORDER BY TotalSales DESC"

# Execute the query
cursor.execute(query)

# Fetch the results
results = cursor.fetchall()

# Print the top sales by supplier
print("Top sales by supplier:")
for result in results:
    supplier_alias = result[0]
    total_sales = result[1]
    print("Supplier:", supplier_alias)
    print("Total Sales:", total_sales)
    print()
```

```
Top sales by supplier:
Supplier: High Tech Drinks
Total Sales: 266839.26999999967

Supplier: East Coast Beverage
Total Sales: 255253.6400000003
```

```
Supplier: Cool Canadian
Total Sales: 251842.08999999988
```

## 1.3 2.2 BUSINESS OBJECTIVES

Business Objectives 1. Increase Sales Revenue 2. Expand Market Presence

## 1.4 2.3 DECISIONS

Decisions for Objective 1: Increase Sales Revenue

1. Analyze sales data to identify high-potential products and develop targeted marketing and promotion strategies to boost their sales.
2. Implement a customer retention program to enhance customer loyalty and encourage repeat purchases, leading to increased sales revenue.

Decisions for Objective 2: Expand Market Presence

1. Conduct market research and analysis to identify new target markets and consumer segments with growth potential.
2. Develop and launch new product variants or flavors to cater to the preferences and demands of different market segments.

## 1.5 2.4 BUSINESS QUESTIONS

1. Which products have shown the highest sales growth potential based on the analysis of sales data, and what targeted marketing and promotion strategies can be implemented to maximize their sales?
2. Which new target markets and consumer segments show the most growth potential based on market research and analysis, and what strategies should be employed to penetrate these markets effectively?
3. What are the highest and lowest selling categories within the different types of drinks?

## 1.6 2.5 DATABASE QUERIES

[ ]:
```
# Q1
# Fetch products with the highest sales growth potential (GROWTH POTENTIAL IS␣
 ↪INDICATED BY No. of Units sold)
cursor = cnx.cursor()
query1a = "SELECT p.SKU_ALIAS \
           FROM salesfact sf \
           JOIN product p ON sf.PRODUCTID = p.PRODUCTID \
           GROUP BY p.SKU_ALIAS \
           ORDER BY SUM(sf.SALES) DESC"

# Execute the query
cursor.execute(query1a)
```

```
# Fetch the results
results1a = cursor.fetchall()

# Print the products with the highest sales growth potential
print("Products with highest sales growth potential:")
for result in results1a:
    product_alias = result[0]
    print("Product Alias:", product_alias)
```

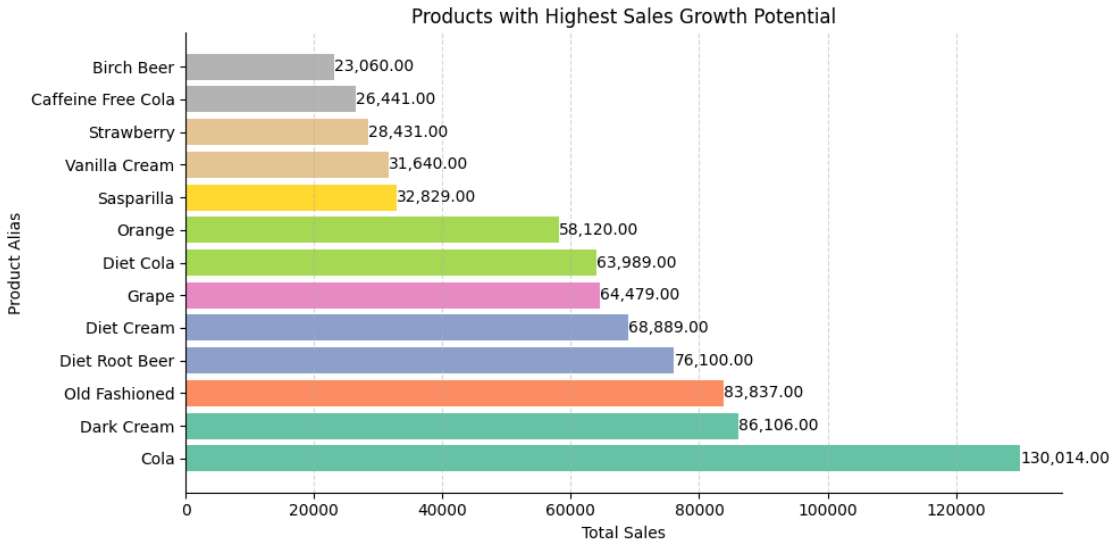
```
Products with highest sales growth potential:
Product Alias: Cola
Product Alias: Dark Cream
Product Alias: Old Fashioned
Product Alias: Diet Root Beer
Product Alias: Diet Cream
Product Alias: Grape
Product Alias: Diet Cola
Product Alias: Orange
Product Alias: Sasparilla
Product Alias: Vanilla Cream
Product Alias: Strawberry
Product Alias: Caffeine Free Cola
Product Alias: Birch Beer
```

[14]:
```
#Q1
# creating a plot
import matplotlib.pyplot as plt
import numpy as np

# Fetch products with the highest sales growth potential (1GROWTH POTENTIAL IS␣
 ↪INDICATED BY No. of Units sold)
cursor = cnx.cursor()
query1a = "SELECT p.SKU_ALIAS, SUM(sf.SALES) AS total_sales \
           FROM salesfact sf \
           JOIN product p ON sf.PRODUCTID = p.PRODUCTID \
           GROUP BY p.SKU_ALIAS \
           ORDER BY total_sales DESC"

# Execute the query
cursor.execute(query1a)

# Fetch the results
results1a = cursor.fetchall()

# Extract the product aliases and total sales
product_aliases = [result[0] for result in results1a]
```

```
total_sales = [result[1] for result in results1a]

# Print the products with the highest sales growth potential
print("Products with highest sales growth potential:")
for result in results1a:
    product_alias = result[0]
    print("Product Alias:", product_alias)

# Generate a custom color scheme
colors = plt.cm.Set2(np.linspace(0, 1, len(product_aliases)))

# Create a figure and axes
fig, ax = plt.subplots(figsize=(10, 5))

# Create a horizontal bar chart with custom colors
bar_plot = ax.barh(product_aliases, total_sales, color=colors)

# Add annotations to the bars
for i, bar in enumerate(bar_plot):
    sales = total_sales[i]
    ax.text(bar.get_width(), bar.get_y() + bar.get_height() / 2, f'{sales:,.
 ↪2f}', ha='left', va='center', fontsize=10)

# Add labels and title
ax.set_xlabel('Total Sales')
ax.set_ylabel('Product Alias')
ax.set_title('Products with Highest Sales Growth Potential')

# Remove the spines
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

# Add a grid
ax.grid(axis='x', linestyle='--', alpha=0.5)

# Show the plot
plt.tight_layout()
plt.show()
```

```
Products with highest sales growth potential:
Product Alias: Cola
Product Alias: Dark Cream
Product Alias: Old Fashioned
Product Alias: Diet Root Beer
Product Alias: Diet Cream
Product Alias: Grape
Product Alias: Diet Cola
Product Alias: Orange
```

```
Product Alias: Sasparilla
Product Alias: Vanilla Cream
Product Alias: Strawberry
Product Alias: Caffeine Free Cola
Product Alias: Birch Beer
```



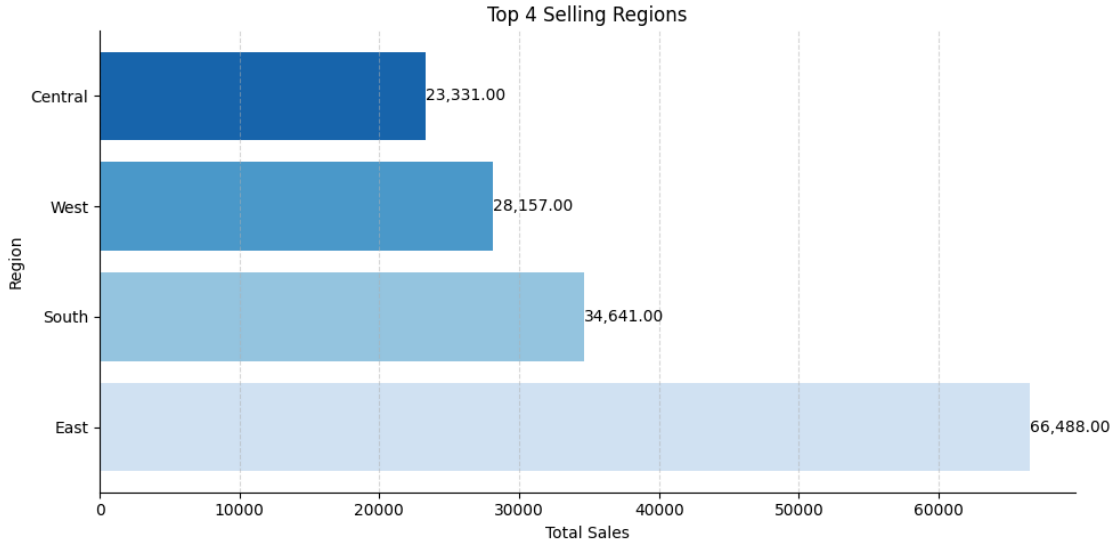

In this query below, we join the 'region' table with the 'salesfact' table based on matching 'REGIONID' and 'STATEID'. Then we calculate the total sales for each region by SUM(sf.SALES) up the 'SALES' values from the 'salesfact' table for each region. Then we group them by region and and sort the results in descending order based on the total sales and then we limit the result to 5.

[15]:
```
#Q2
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

# Create a cursor object
cursor = cnx.cursor()

# Execute the query to fetch the top 4 selling regions ( AS WE FOUND THAT THERE␣
 ↪ARE ONLY 4 REGIONS, BUT WE LIMIT BY 5 JUST TO MAKE SURE WE GET ALL THE DATA)
query = "SELECT r.REGION, SUM(sf.SALES) AS total_sales \
         FROM region r \
         JOIN salesfact sf ON r.REGIONID = sf.STATEID \
         GROUP BY r.REGION \
         ORDER BY total_sales DESC \
         LIMIT 5"

# Execute the query
```

```
cursor.execute(query)

# Fetch the results
results = cursor.fetchall()

# Extract the regions and total sales
regions = [result[0] for result in results]
total_sales = [result[1] for result in results]

# Create a color palette for the bars
colors = sns.color_palette('Blues', len(regions))

# Create a figure and axes
fig, ax = plt.subplots(figsize=(10, 5))

# Create a horizontal bar chart with color gradients
bar_plot = ax.barh(regions, total_sales, color=colors)

# Add labels and title
ax.set_xlabel('Total Sales')
ax.set_ylabel('Region')
ax.set_title('Top 4 Selling Regions')

# Add data labels to the bars
for i, bar in enumerate(bar_plot):
    width = bar.get_width()
    ax.text(width, bar.get_y() + bar.get_height() / 2, f'{width:,.2f}',␣
 ↪ha='left', va='center', fontsize=10)

# Remove the spines
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

# Add a grid
ax.grid(axis='x', linestyle='--', alpha=0.5)

# Print the top 4 selling products
top_4_products = results[:4]
print("Top 4 Selling Regions:")
for product in top_4_products:
    region, sales = product
    print(f"Region: {region}, Total Sales: {sales}")

# Show the plot
plt.tight_layout()
plt.show()
```

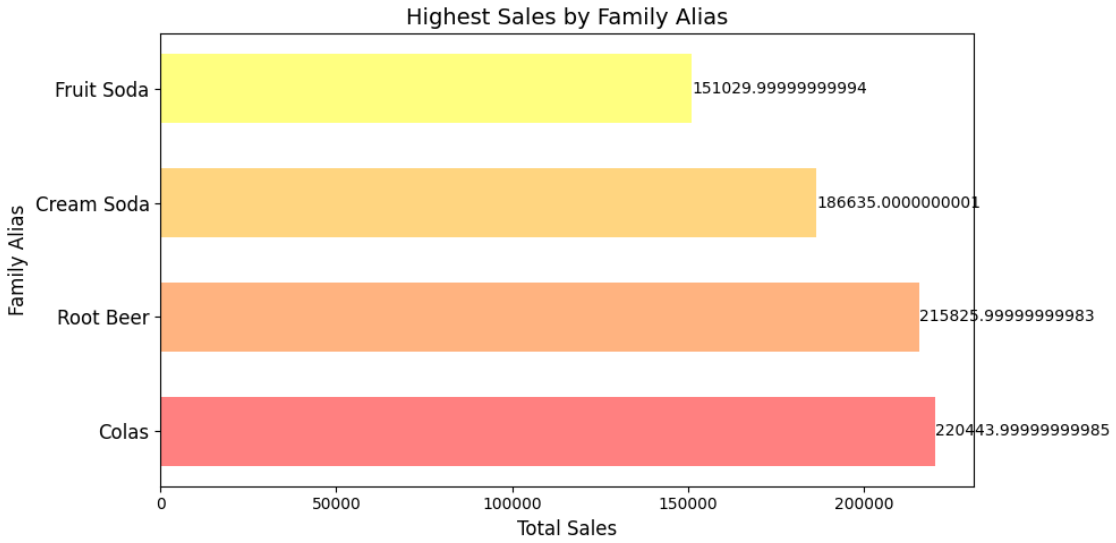
```
Top 4 Selling Regions:
```

```
Region: East, Total Sales: 66488.00000000006
Region: South, Total Sales: 34641.00000000001
Region: West, Total Sales: 28156.99999999999
Region: Central, Total Sales: 23331.0
```



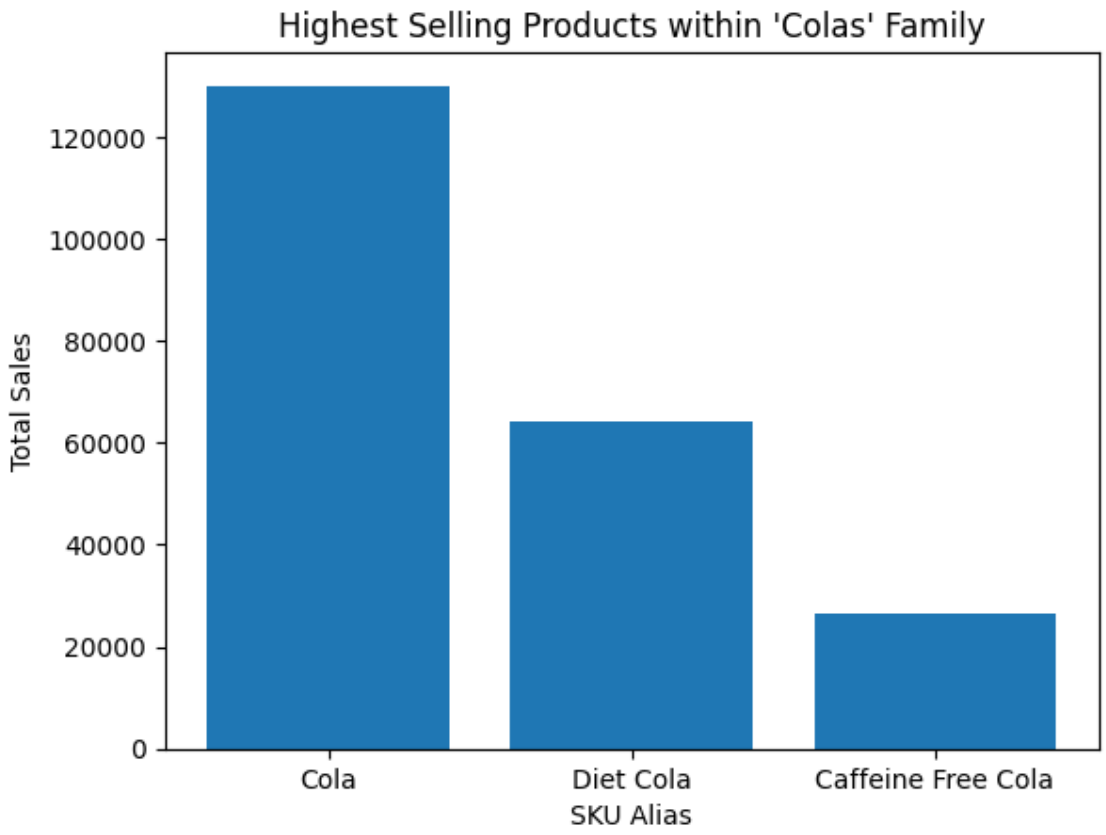

[16]:
```
#Q3
import matplotlib.pyplot as plt
import numpy as np

# Fetch family alias with the highest sales (WE DO THIS AS WE WANT TO FIND␣
 ↪HISHEST SOLD FAMILY)
cursor = cnx.cursor()
query1a = """
    SELECT f.FAMILY_ALIAS, SUM(sf.SALES) AS TotalSales
    FROM family f
    JOIN product p ON f.FAMILYID = p.FAMILYID
    JOIN salesfact sf ON p.PRODUCTID = sf.PRODUCTID
    GROUP BY f.FAMILY_ALIAS
    ORDER BY TotalSales DESC
"""

# Execute the query
cursor.execute(query1a)

# Fetch the results
results1a = cursor.fetchall()

# Extract the family aliases and total sales from the results
```

```
family_aliases = [result[0] for result in results1a]
total_sales = [result[1] for result in results1a]

# Set up custom colors and styles
colors = ['#FF8080', '#FFB380', '#FFD580', '#FFFF80', '#B3FF80']
bar_width = 0.6

# Create a horizontal bar chart to visualize the highest sales by family alias
fig, ax = plt.subplots(figsize=(10, 5))
bar_positions = np.arange(len(family_aliases))

ax.barh(bar_positions, total_sales, height=bar_width, color=colors)
ax.set_yticks(bar_positions)
ax.set_yticklabels(family_aliases, fontsize=12)
ax.set_xlabel('Total Sales', fontsize=12)
ax.set_ylabel('Family Alias', fontsize=12)
ax.set_title('Highest Sales by Family Alias', fontsize=14)

# Add data labels to the bars
for i, v in enumerate(total_sales):
    ax.text(v + 50, i, str(v), color='black', fontsize=10, va='center')

plt.tight_layout()
plt.show()
```



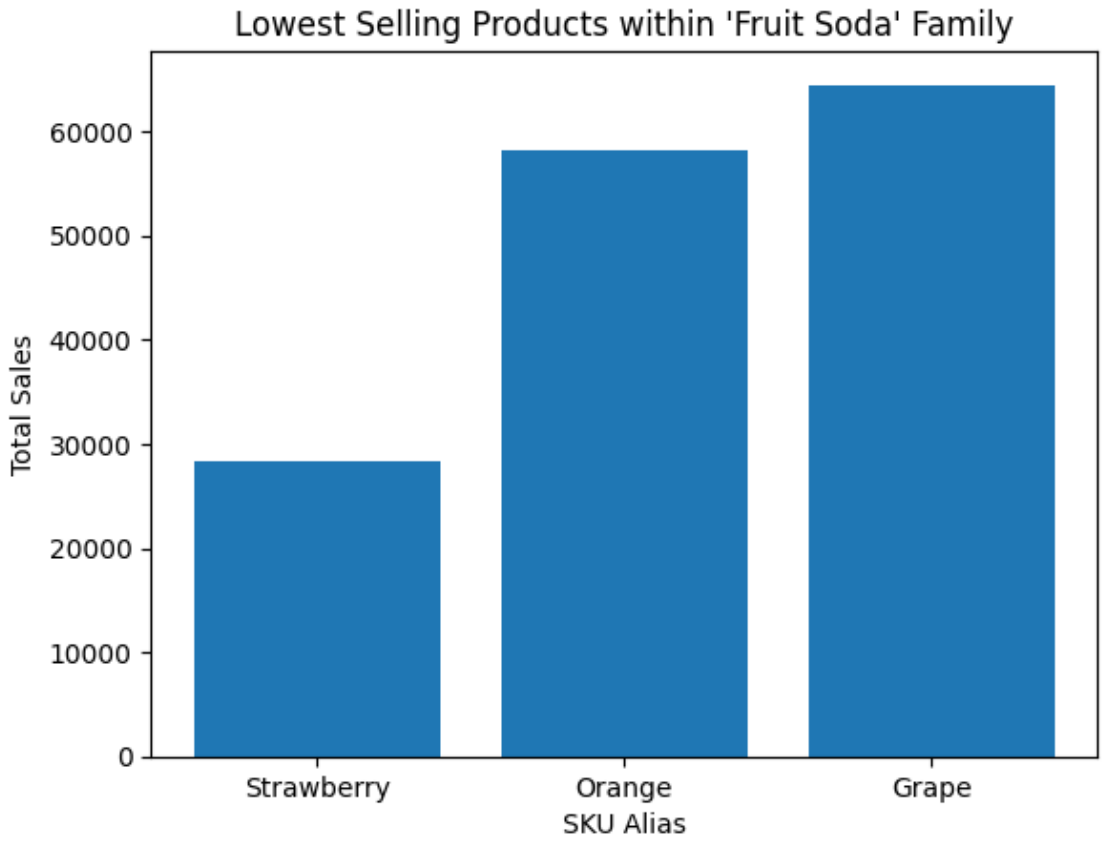

[17]:
```
import matplotlib.pyplot as plt
import numpy as np
```

```
# Fetch highest selling products within 'Colas' family  (WE DO THIS AS WE WANT␣
 ↪TO FIND HISHEST SOLD FAMILY)
cursor = cnx.cursor()
query_highest_colas = """
    SELECT p.SKU_ALIAS, SUM(sf.SALES) AS TotalSales
    FROM product p
    JOIN family f ON p.FAMILYID = f.FAMILYID
    JOIN salesfact sf ON p.PRODUCTID = sf.PRODUCTID
    WHERE f.FAMILY_ALIAS = 'colas'
    GROUP BY p.SKU_ALIAS
    ORDER BY TotalSales DESC
    LIMIT 5
"""

cursor.execute(query_highest_colas)
results_highest_colas = cursor.fetchall()

# Extract the SKU aliases and total sales from the results
sku_aliases = [result[0] for result in results_highest_colas]
total_sales = [result[1] for result in results_highest_colas]

# Create a bar chart to visualize the highest selling products within 'Colas'␣
 ↪family
plt.bar(sku_aliases, total_sales)
plt.xlabel('SKU Alias')
plt.ylabel('Total Sales')
plt.title("Highest Selling Products within 'Colas' Family")

plt.show()
```



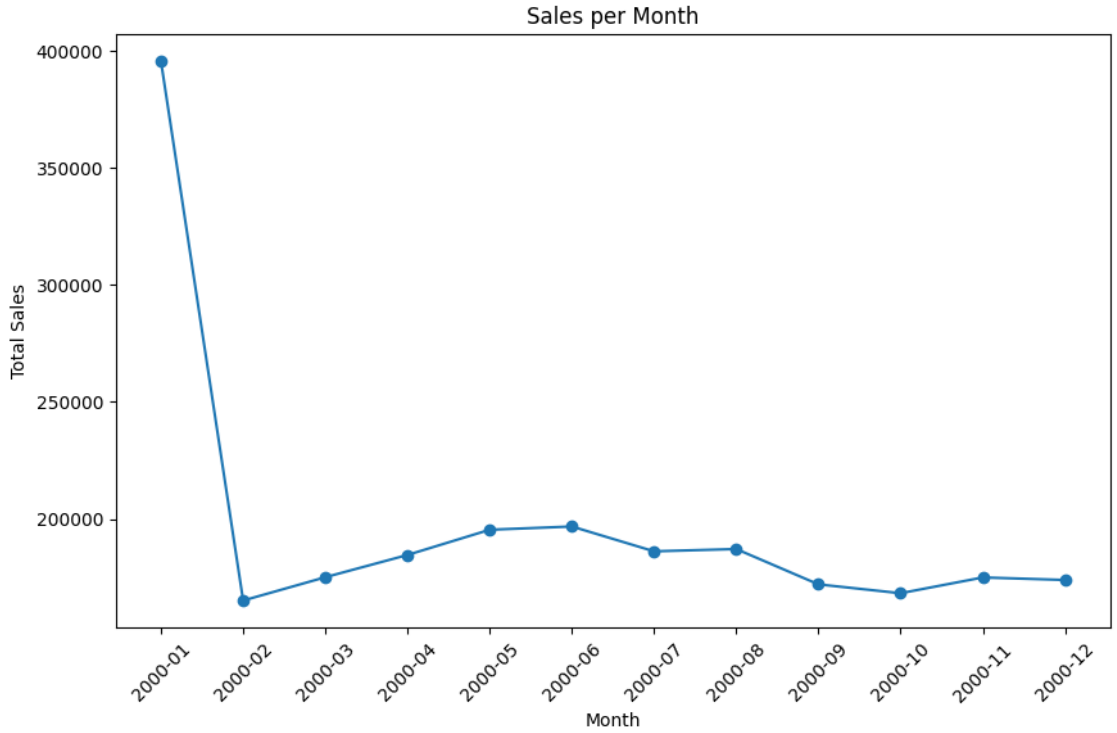

[20]:
```
import matplotlib.pyplot as plt
import numpy as np

# Fetch lowest selling products within 'Fruit Soda' family (WE DO THIS AS WANT␣
 ↪TO FIND OUT FOR WHICH PRODUCTS NEW STRATEGIES CAN BE IMPLEMENTED)
cursor = cnx.cursor()
query_lowest_fruit_soda = """
    SELECT p.SKU_ALIAS, SUM(sf.SALES) AS TotalSales
    FROM product p
    JOIN family f ON p.FAMILYID = f.FAMILYID
    JOIN salesfact sf ON p.PRODUCTID = sf.PRODUCTID
    WHERE f.FAMILY_ALIAS = 'fruit soda'
    GROUP BY p.SKU_ALIAS
    ORDER BY TotalSales ASC
    LIMIT 5
"""

cursor.execute(query_lowest_fruit_soda)
results_lowest_fruit_soda = cursor.fetchall()
```

```
# Extract the SKU aliases and total sales from the results
sku_aliases = [result[0] for result in results_lowest_fruit_soda]
total_sales = [result[1] for result in results_lowest_fruit_soda]

# Create a bar chart to visualize the lowest selling products within 'Fruit␣
 ↪Soda' family
plt.bar(sku_aliases, total_sales)
plt.xlabel('SKU Alias')
plt.ylabel('Total Sales')
plt.title("Lowest Selling Products within 'Fruit Soda' Family")

plt.show()
```




[21]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# Create a cursor object
cursor = cnx.cursor()
```

```
# Execute the query to fetch the transdate and amount (WE CARRY THIS OUT AS WE␣
 ↪WANT TO SEE WHAT KIND OF SEASONS ARE SELLING THE MOST)
query = "SELECT DATE_FORMAT(s.TRANSDATE, '%Y-%m') AS month, SUM(s.AMOUNT) AS␣
 ↪total_sales \
        FROM sales s \
        GROUP BY month \
        ORDER BY month ASC"

# Execute the query
cursor.execute(query)

# Fetch the results
results = cursor.fetchall()

# Extract the month and total sales
months = [result[0] for result in results]
total_sales = [result[1] for result in results]

# Create a DataFrame from the results
df = pd.DataFrame({'Month': months, 'Total Sales': total_sales})

# Plot the sales per month
plt.figure(figsize=(10, 6))
plt.plot(df['Month'], df['Total Sales'], marker='o')
plt.xlabel('Month')
plt.ylabel('Total Sales')
plt.title('Sales per Month')
plt.xticks(rotation=45)
plt.show()
```




## 1.7 2.6 Conclusion

Conclusion:

Based on the analysis of sales data and market research, several key findings have emerged, which provide valuable insights for maximizing sales growth potential and market penetration. The following three-part conclusion summarizes the findings and provides recommendations for each area:

1. Product-Specific Sales Growth Potential and Marketing Strategies:
   - The analysis revealed that the products with the highest sales growth potential are Cola, Dark Cream, Old Fashioned, and Diet Root Beer.
   - To maximize their sales, targeted marketing and promotion strategies should be implemented.
   - It is recommended to focus on marketing these top-selling products in regions that are currently underperforming in terms of overall sales.
   - This strategy will not only diversify the client's locations but also leverage the market-tested success of these products to boost sales in the targeted regions.
2. New Target Markets and Consumer Segments with Growth Potential and Penetration Strategies:
   - The analysis identified the East and South regions as the markets with the most growth potential, with significant sales volumes of approximately 66,000 and 34,000 units, respectively.
   - To effectively penetrate these markets, the client should diversify their supplier base.

- By partnering with additional suppliers or distributors in the West and North areas, the client can expand their reach and tap into untapped consumer segments.
- This expansion strategy will allow the client to capitalize on the existing success in the East and South regions while unlocking new growth opportunities in previously untapped markets.

3. Highest and Lowest Selling Categories within Different Drink Types:
    - The analysis revealed that Colas are the highest selling category, while fruit Soda is the lowest selling category.
    - Within the Colas category, regular cola, diet cola, and caffeine-free cola were identified as the top performers.
    - Among fruit sodas, strawberry, orange, and grape showed lower sales performance.
    - To address the underperformance of the fruit soda category, a rebranding effort and alternative marketing methods are recommended.
    - By targeting and reaching out to consumers who are interested in fruit-based sodas, the client can increase sales within this category.
    - Additionally, expanding and seeking recommendations from other areas will help reach a broader customer base and enhance the overall performance of the fruit soda category.

By implementing the recommended strategies, such as targeting underperforming regions with top-selling products, diversifying suppliers to penetrate new markets, and rebranding and marketing efforts for the fruit soda category, the client can drive sales growth, increase market share, and capitalize on untapped opportunities in their industry.