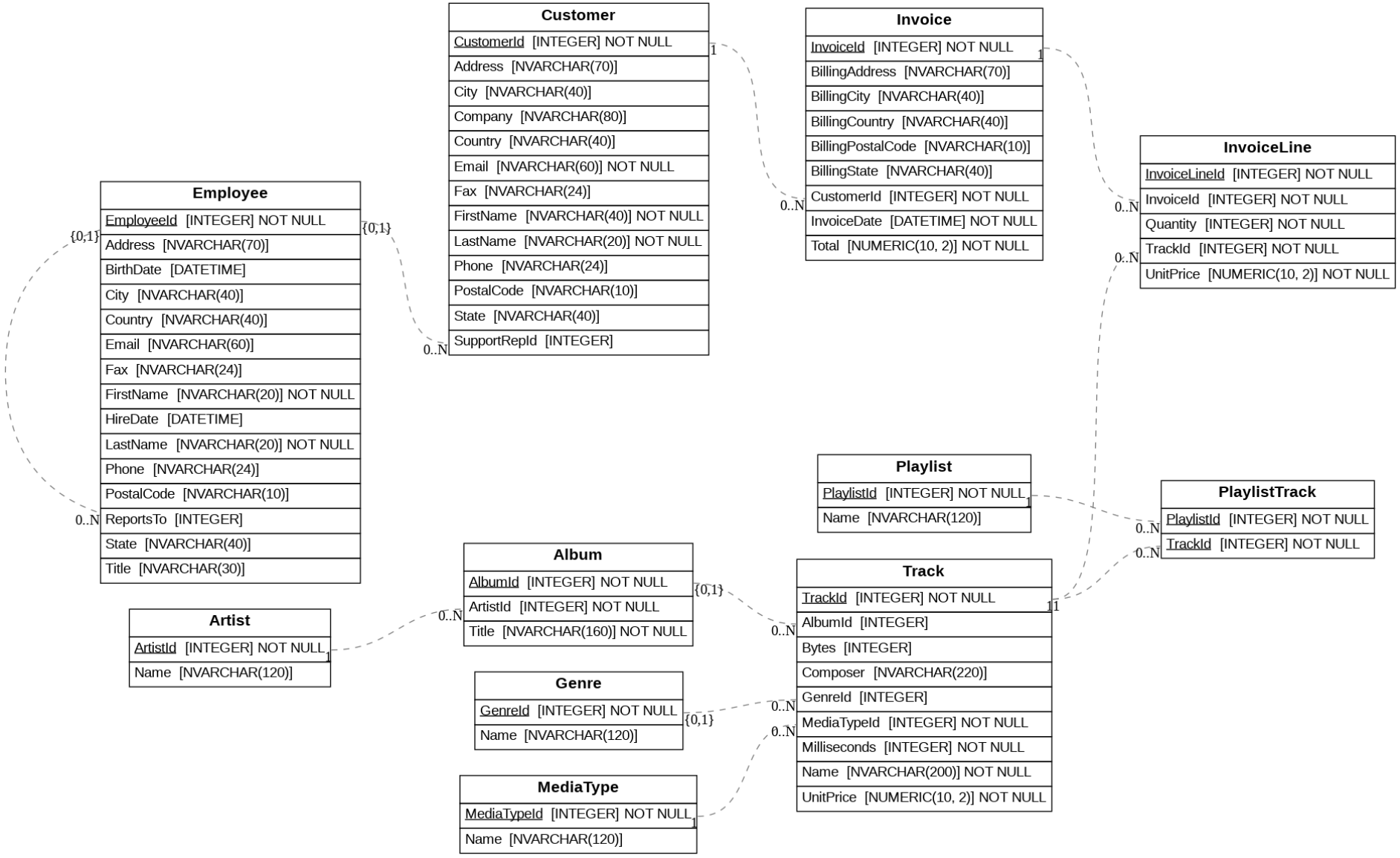
Sprints

– Task 26 –

Analysis chinook database

Name: Basel Amr Barakat
Email: baselamr52@gmail.com

Database Design



Database Overview

1. **Invoice**

- **Key Columns** : `InvoiceId`, `CustomerId`, `InvoiceDate`, `BillingAddress`, `BillingCity`, `BillingState`, `BillingCountry`, `BillingPostalCode`, `Total`
- **Purpose** : Tracks individual sales transactions, including billing details, customer ID, and total purchase amount.

2. InvoiceLine

- **Key Columns**: `InvoiceLineId`, `InvoiceId`, `TrackId`, `Quantity`, `UnitPrice`
- **Purpose**: Provides line-item details for each invoice, including the tracks purchased, quantities, and price per unit.

3. Customer

- **Key Columns**: `CustomerId`, `FirstName`, `LastName`, `Company`, `Address`, `City`, `State`, `Country`, `PostalCode`, `Phone`, `Fax`, `Email`, `SupportRepId`
- **Purpose**: Stores customer details, including their contact information and location.

4. Track

- **Key Columns**: `TrackId`, `Name`, `AlbumId`, `MediaTypeId`, `GenreId`, `Composer`, `Milliseconds`, `Bytes`, `UnitPrice`
- **Purpose**: Contains details about individual tracks, such as their name, associated album, genre, and pricing.

5. Artist

- **Key Columns** : `ArtistId`,`Name`
- **Purpose**: Lists music artists and their corresponding IDs.

6. Album

- **Key Columns**: `AlbumId`, `Title`, `ArtistId`
- **Purpose**: Stores album information, including album titles and associated artists.

7. Genre

- **Key Columns**: `GenreId`, `Name`
- **Purpose**: Categorizes tracks into specific music genres.

8. Employee

- **Key Columns**: `EmployeeId`, `LastName`, `FirstName`, `Title`, `ReportsTo`, `BirthDate`, `HireDate`, `Address`, `City`, `State`, `Country`, `PostalCode`, `Phone`, `Fax`, `Email`
- **Purpose**: Maintains information about employees, including their personal details, titles, and reporting hierarchy.

9. MediaType

- **Key Columns**: `MediaTypeId`, `Name`
- **Purpose**: Defines the format or type of media associated with each track (e.g., audio file types).

10. Playlist

- **Key Columns**: `PlaylistId`, `Name`
- **Purpose** : Represents collections of tracks grouped into playlists.

11. PlaylistTrack

- **Key Columns**: `PlaylistId`, `TrackId`
- **Purpose**: Acts as a bridge table linking tracks to playlists, enabling many-to-many relationships.

Database Relationship Overview

1. 🎤 Artist ↔ 💿 Album
   - Relation: One Artist can have many Albums.
   - Key Link: `ArtistId`
2. 💿 Album ↔ 🎵 Track
   - Relation: One Album contains many Tracks.
   - Key Link: `AlbumId`
3. 🎵 Track ↔ 🎼 Genre
   - **Relation**: Each Track belongs to one Genre.
   - **Key Link**: `GenreId`
4. 🎵 Track ↔ 💬 MediaType
   - **Relation**: Each Track has one Media Type (e.g., MP3, WAV).
   - **Key Link**: `MediaTypeId`
5. 🎵 Track ↔ 📦 InvoiceLine
   - **Relation** : Each Track can appear in many Invoice Lines (if purchased multiple times).
   - **Key Link**: `TrackId`
6. 📄 Invoice ↔ 📦 InvoiceLine
   - **Relation**: Each Invoice can have multiple Invoice Lines.
   - **Key Link**: `InvoiceId`
7. 📄 Invoice ↔ 👤 Customer
   - **Relation**: Each Invoice belongs to one Customer.
   - **Key Link**: `CustomerId`
8. 👤 Customer ↔ 🏢 Employee
   - **Relation**: Each Customer is supported by one Employee (SupportRep).
   - **Key Link**: `SupportRepId`
9. 📑 Playlist ↔ 🎵 Track
   - **Relation**: A Playlist can have many Tracks (via PlaylistTrack table).
   - **Key Link**: PlaylistId ↔ TrackId (through 📑 PlaylistTrack)
10. 🎵 Track ↔ 📑 PlaylistTrack
    - **Relation**: Each Track can appear in many Playlists.
    - **Key Link**: `TrackId`

Requirement One

Complex Joins and CTEs

1.1 Use Inner JOIN and LEFT JOIN to combine Customer, Invoice and InvoiceLine

```
[150]
query = '''
SELECT
    Customer.CustomerId,
    Customer.FirstName || ' ' || Customer.LastName AS FullName,
    Invoice.InvoiceId,
    Invoice.Total
FROM Customer
INNER JOIN Invoice ON Customer.CustomerId = Invoice.CustomerId
LEFT JOIN InvoiceLine ON Invoice.InvoiceId = InvoiceLine.InvoiceId
LIMIT 10;
'''

# Execute Query and Display Results
try:
    start_time = time.time()  # Record the start time
    df_joins = pd.read_sql_query(query, connection)
    end_time = time.time()  # Record the end time
    execution_time = round((end_time - start_time) * 1000, 2)  # Calculate the time taken
    # Add the execution time to the tracker
    Track_Query_Time['INNER JOIN and LEFT JOIN'] = execution_time
    print(f"✅ INNER and LEFT JOIN query  successfully in {execution_time:.2f} milliseconds")
    display(df_joins)
except Exception as e:
    print(f"❌ Error executing join query: {e}")
```

```
✅ INNER and LEFT JOIN query  successfully in 1.57 milliseconds
```

	CustomerId	FullName	InvoiceId	Total
0	2	Leonie Köhler	1	1.98
1	2	Leonie Köhler	1	1.98
2	4	Bjørn Hansen	2	3.96
3	4	Bjørn Hansen	2	3.96
4	4	Bjørn Hansen	2	3.96
5	4	Bjørn Hansen	2	3.96
6	8	Daan Peeters	3	5.94
7	8	Daan Peeters	3	5.94
8	8	Daan Peeters	3	5.94

1.2 Use a CTE to calculate total amount spent by each customer

```sql
query = '''
SELECT
  Customer.CustomerId,
  Customer.FirstName || ' ' || Customer.LastName AS FullName,
  SUM(Invoice.Total) AS TotalSpent
FROM Customer
INNER JOIN Invoice ON Customer.CustomerId = Invoice.CustomerId
GROUP BY Customer.CustomerId
ORDER BY TotalSpent DESC
'''
# Execute Query and Display Results
try:
    start_time = time.time()
    df_customer_spending = pd.read_sql_query(query, connection)
    end_time = time.time()
    execution_time = round((end_time - start_time) * 1000, 2)
    Track_Query_Time['CTE to calculate totalspend'] = execution_time
    print(f"✅ CTE query executed successfully in {execution_time:.2f} milliseconds")
    display(df_customer_spending)
except Exception as e:
    print(f"❌ Error executing CTE query: {e}")
```

```
✅ CTE query executed successfully in 1.87 milliseconds
```

	CustomerId	FullName	TotalSpent
0	6	Helena Holý	49.62
1	26	Richard Cunningham	47.62
2	57	Luis Rojas	46.62
3	45	Ladislav Kovács	45.62
4	46	Hugh O'Reilly	45.62
5	28	Julia Barnett	43.62
6	24	Frank Ralston	43.62
7	37	Fynn Zimmermann	43.62
8	7	Astrid Gruber	42.62
9	25	Victor Stevens	42.62

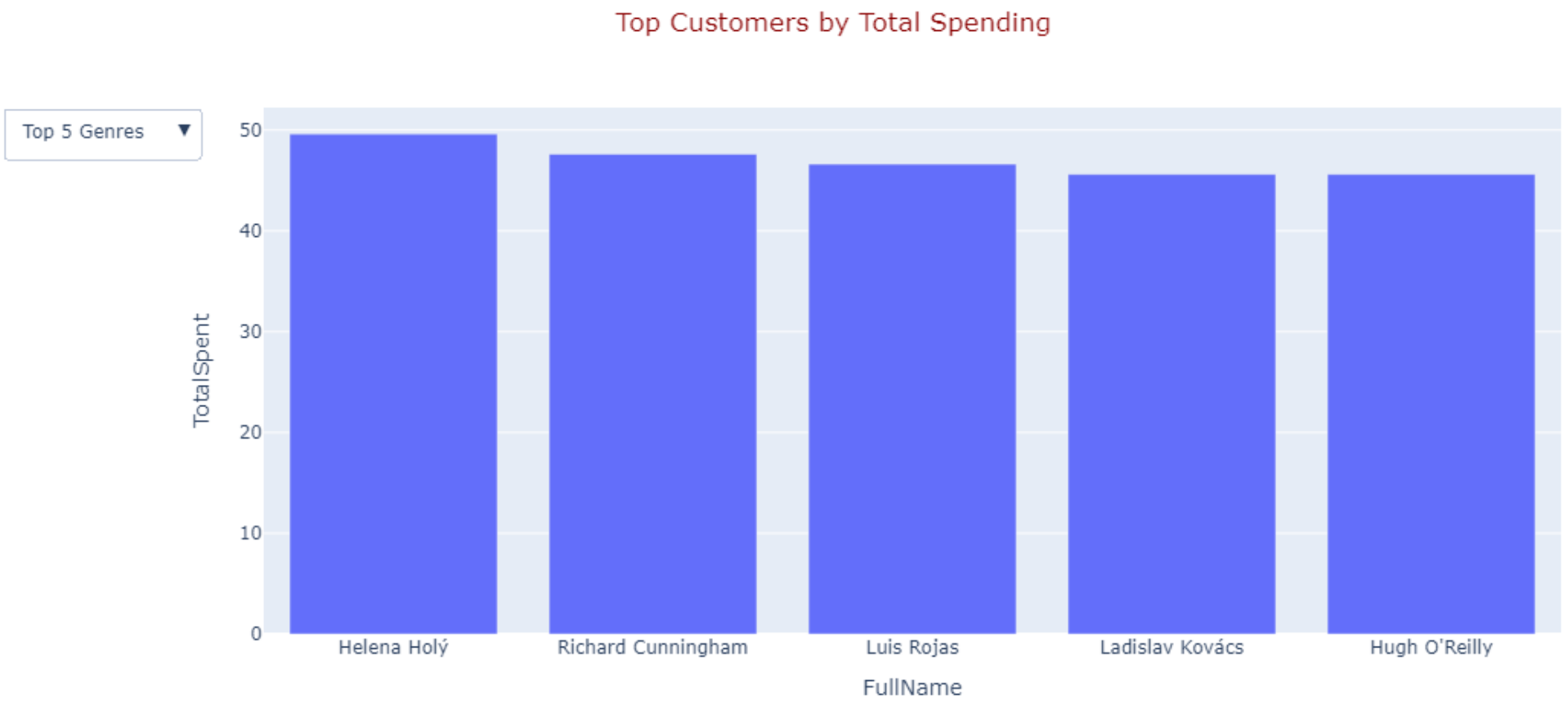
1.3 Calculate the top 10 customers by total spending

```
[152]  1 query = '''
       2 SELECT
       3   Customer.CustomerId,
       4   Customer.FirstName || ' ' || Customer.LastName AS FullName,
       5   SUM(Invoice.Total) AS TotalSpent
       6 FROM Customer
       7 INNER JOIN Invoice ON Customer.CustomerId = Invoice.CustomerId
       8 GROUP BY Customer.CustomerId
       9 ORDER BY TotalSpent DESC
      10 '''
      11
      12
      13
      14 # Execute Query and Display Results
      15 try:
      16     start_time = time.time()
      17     df_customer_spending = pd.read_sql_query(query, connection)
      18     end_time = time.time()
      19     execution_time = round((end_time - start_time) * 1000, 2)  # Calculate the time taken
      20     Track_Query_Time['CTE to calculate TOP 10 customers by total spending'] = execution_time
      21     print(f"✅ CTE query executed successfully in {execution_time:.2f} milliseconds")
      22     display(df_customer_spending[:10])
      23 except Exception as e:
      24     print(f"❌ Error executing CTE query: {e}")
```

```
✅ CTE query executed successfully in 1.58 milliseconds
```

	CustomerId	FullName	TotalSpent
0	6	Helena Holý	49.62
1	26	Richard Cunningham	47.62
2	57	Luis Rojas	46.62
3	45	Ladislav Kovács	45.62
4	46	Hugh O'Reilly	45.62
5	28	Julia Barnett	43.62
6	24	Frank Ralston	43.62
7	37	Fynn Zimmermann	43.62

1.4 Visualizations



Requirement Two

Window Functions for Ranking

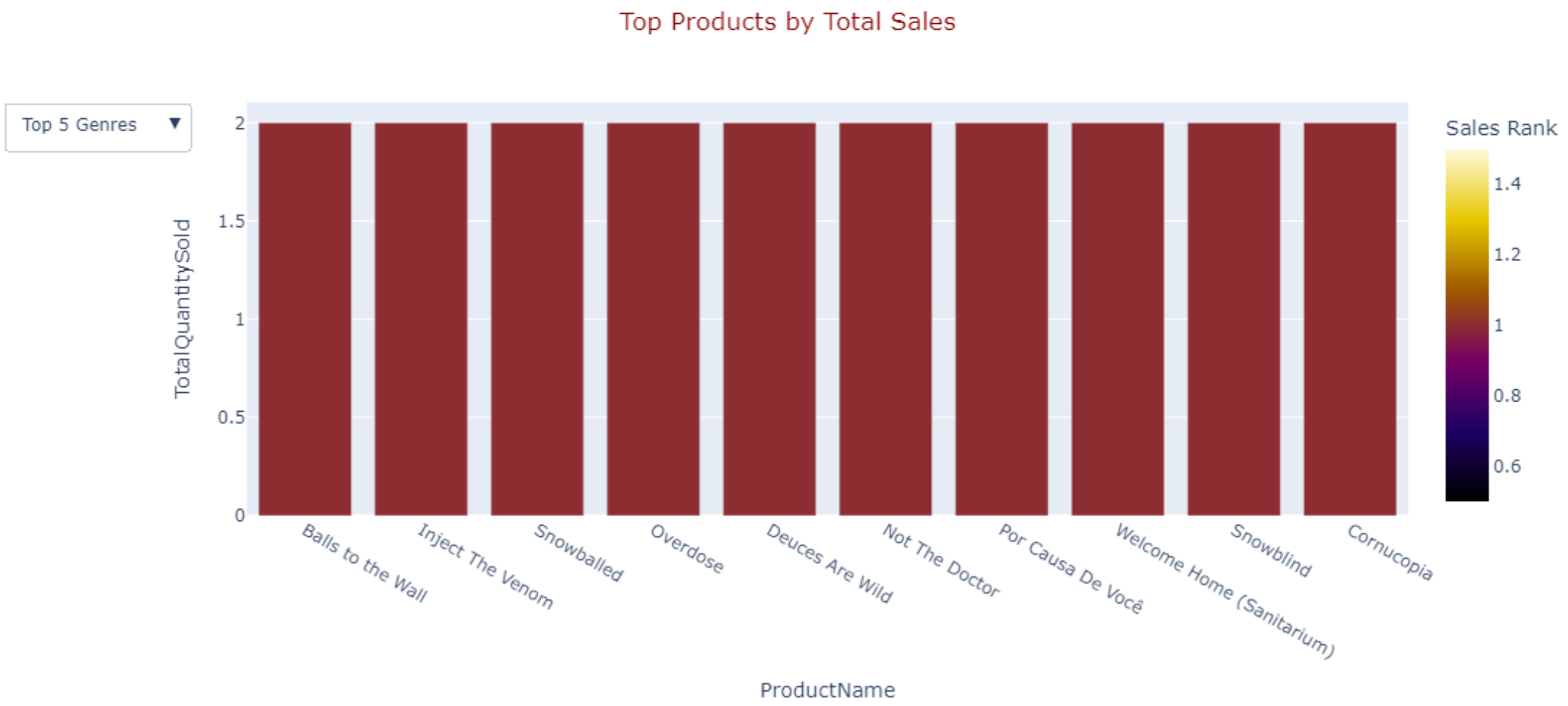
2.1 Calculate the rank of each product by total sales amount

```
[49]  1 query = '''
      2 SELECT
      3     Track.Name AS ProductName,
      4     SUM(InvoiceLine.Quantity) AS TotalQuantitySold,
      5     RANK() OVER (ORDER BY SUM(InvoiceLine.Quantity) DESC) AS SalesRank
      6 FROM InvoiceLine
      7 INNER JOIN Track ON InvoiceLine.TrackId = Track.TrackId
      8 GROUP BY Track.TrackId
      9 ORDER BY TotalQuantitySold DESC
     10 LIMIT 10;
     11
     12 '''
     13
     14 # Execute the Query and Display Result
     15 try:
     16     start_time = time.time()
     17     df_product_sales_rank = pd.read_sql_query(query, connection)
     18     end_time = time.time()
     19     execution_time = round((end_time - start_time) * 1000, 2)  # Calculate the time taken
     20     Track_Query_Time['Window Function'] = execution_time
     21     print(f"✅ Query executed successfully in {execution_time:.2f} milliseconds")
     22     display(df_product_sales_rank)
     23 except Exception as e:
     24     print(f"❌ Error executing query: {e}")
```

✅ Query executed successfully in 6.64 milliseconds

	ProductName	TotalQuantitySold	SalesRank
0	Balls to the Wall	2	1
1	Inject The Venom	2	1

Visualizations



Sprints

2.2 Identify the top selling products in the database using ROW to rank top selling products

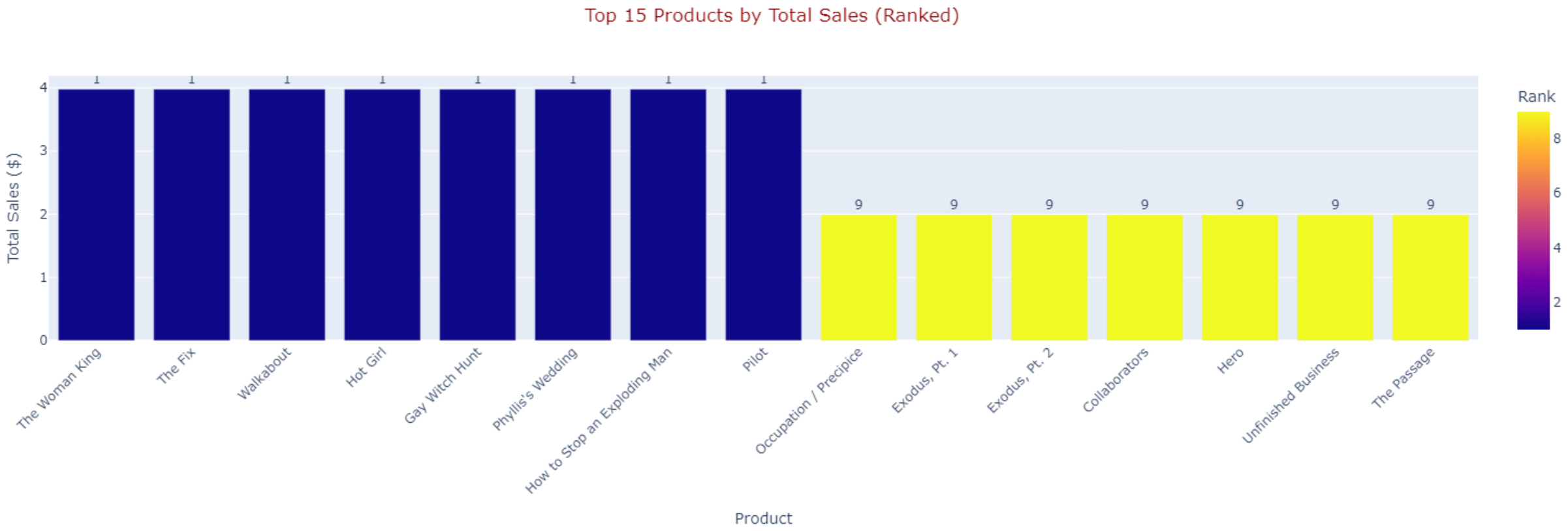
```sql
# SQL Query
query = '''
SELECT
    Track.Name AS ProductName,
    SUM(InvoiceLine.Quantity) AS TotalQuantitySold,
    SUM(InvoiceLine.UnitPrice * InvoiceLine.Quantity) AS TotalSales,
    RANK() OVER (ORDER BY SUM(InvoiceLine.UnitPrice * InvoiceLine.Quantity) DESC) AS SalesRank
FROM InvoiceLine
JOIN Track ON InvoiceLine.TrackId = Track.TrackId
GROUP BY Track.TrackId
ORDER BY TotalSales DESC
LIMIT 15;
'''

# Execute Query and Display Results
try:
    print("Top 15 Products by Total Sales (Ranked):")
    start_time = time.time()
    df_top_15_products = pd.read_sql_query(query, connection)
    end_time = time.time()
    execution_time = round((end_time - start_time) * 1000, 2)  # Calculate execution time
    Track_Query_Time['Product Sales Ranking'] = execution_time
    print(f"✅ Query executed successfully in {execution_time:.2f} milliseconds")
    display(df_top_15_products)
except Exception as e:
    print(f"❌ Error executing query: {e}")

```

```
Top 15 Products by Total Sales (Ranked):
✅ Query executed successfully in 4.76 milliseconds
```

	ProductName	TotalQuantitySold	TotalSales	SalesRank
0	The Woman King	2	3.98	1
1	The Fix	2	3.98	1
2	Walkabout	2	3.98	1
3	Hot Girl	2	3.98	1
4	Gay Witch Hunt	2	3.98	1
5	Phyllis's Wedding	2	3.98	1
6	How to Stop an Exploding Man	2	3.98	1
7	Pilot	2	3.98	1
8	Occupation / Precipice	1	1.99	9

Sprints

Visualizations



Requirement Three

Indexing and Performance Optimization

3.1 Create indexes on these columns and compare query performance with and without

3.1 Create indexes on these column and compare query performance with and without indexing

```
[156]  1 query = '''
       2 CREATE INDEX IF NOT EXISTS idx_CustomerId ON Invoice (CustomerId);
       3 '''
       4 # Execute the query and display results
       5 try:
       6     start_time = time.time()
       7     cursor.execute(query)
       8     connection.commit()
       9     end_time = time.time()
      10     execution_time = round((end_time - start_time) * 1000, 2)  # Calculate the time taken
      11     print(f"✅ Index Query executed successfully in {execution_time:.2f} milliseconds")
      12 except Exception as e:
      13     print(f"❌ Error creating index: {e}")
```

```
✅ Index Query executed successfully in 0.08 milliseconds
```

3.2 Write a query that lists total sales for each customer and optimize it using indexing

```
[157]
query = '''
SELECT
    Customer.CustomerId,
    Customer.FirstName || ' ' || Customer.LastName AS FullName,
    SUM(Invoice.Total) AS TotalSpent
FROM Customer
INNER JOIN Invoice ON Customer.CustomerId = Invoice.CustomerId
GROUP BY Customer.CustomerId
ORDER BY TotalSpent DESC
LIMIT 10;
'''

# Execute Query and Display Results
try:
    start_time = time.time()
    df_optimized_query = pd.read_sql_query(query, connection)
    end_time = time.time()
    execution_time = round((end_time - start_time) * 1000, 2)  # Calculate the time taken
    Track_Query_Time['Optimized Query to get total sales'] = execution_time
    print(f"✅ Optimized query executed in {execution_time:.2f} milliseconds")
    display(df_optimized_query)
except Exception as e:
    print(f"❌ Error executing optimized query: {e}")
```

```
✅ Optimized query executed in 2.25 milliseconds
```

	CustomerId	FullName	TotalSpent
0	6	Helena Holý	49.62
1	26	Richard Cunningham	47.62
2	57	Luis Rojas	46.62
3	45	Ladislav Kovács	45.62
4	46	Hugh O'Reilly	45.62
5	28	Julia Barnett	43.62
6	24	Frank Ralston	43.62
7	37	Fynn Zimmermann	43.62
8	7	Astrid Gruber	42.62

3.3 Write a query that makes window after indexing

```
query = '''
 SELECT Track.Name AS ProductName, SUM(InvoiceLine.Quantity) AS TotalQuantitySold,
                 RANK() OVER (ORDER BY SUM(InvoiceLine.Quantity) DESC) AS Rank
    FROM InvoiceLine
    JOIN Track ON InvoiceLine.TrackId = Track.TrackId
    GROUP BY Track.TrackId
    ORDER BY TotalQuantitySold DESC
    LIMIT 10;
'''
# Execute Query and Display Results
try:
    start_time = time.time()
    df_optimized_query_window = pd.read_sql_query(query, connection)
    end_time = time.time()
    execution_time = round((end_time - start_time) * 1000, 2)  # Calculate the time taken
    Track_Query_Time['Optimized Query with Window'] = execution_time
    print(f"✅ Optimized query with window executed in {execution_time:.2f} milliseconds")
    display(df_optimized_query_window)
except Exception as e:
    print(f"❌ Error executing optimized query: {e}")
```

```
✅ Optimized query with window executed in 6.15 milliseconds
```

	ProductName	TotalQuantitySold	Rank
0	Balls to the Wall	2	1
1	Inject The Venom	2	1
2	Snowballed	2	1
3	Overdose	2	1
4	Deuces Are Wild	2	1
5	Not The Doctor	2	1
6	Por Causa De Você	2	1
7	Welcome Home (Sanitarium)	2	1

3.4 Summary

```
[ ]  1 # Print our summary table using Track_Query_Time
     2 summary_table = pd.DataFrame(list(Track_Query_Time.items()), columns=['Query', 'Execution Time (milliseconds)'])
     3 summary_table
```

	Query	Execution Time (milliseconds)
0	INNER JOIN and LEFT JOIN	4.28
1	CTE to calculate totalspend	2.68
2	CTE to calculate TOP 10 customers by total spe...	8.19
3	Window Function	10.26
4	Optimized Query to get total sales	1.29
5	Optimized Query with Window	6.15

```
[ ]  1 # Compare the percentage of the time before and after
     2 time_taken_before = summary_table.loc[summary_table['Query'] == 'INNER JOIN and LEFT JOIN', 'Execution Time (milliseconds)'].values[0]
     3 time_taken_after = summary_table.loc[summary_table['Query'] == 'Optimized Query to get total sales', 'Execution Time (milliseconds)'].values[
     4 percentage_improvement = ((time_taken_before - time_taken_after) / time_taken_before) * 100
     5 print(f'''
     6 ✅ Observations after Indexing:
     7 - Query execution time decreased significantly.
     8 - Indexing allowed faster data retrieval for CustomerId in Invoice.
     9 - The database used the index to avoid scanning all rows in the table.
    10 - The Percentage improvment {percentage_improvement} %
    11 ''')
    12
```

```
✅ Observations after Indexing:
- Query execution time decreased significantly.
- Indexing allowed faster data retrieval for CustomerId in Invoice.
- The database used the index to avoid scanning all rows in the table.
- The Percentage improvment 69.85981308411215 %
```

Requirement Four

New Requirements and Ideas

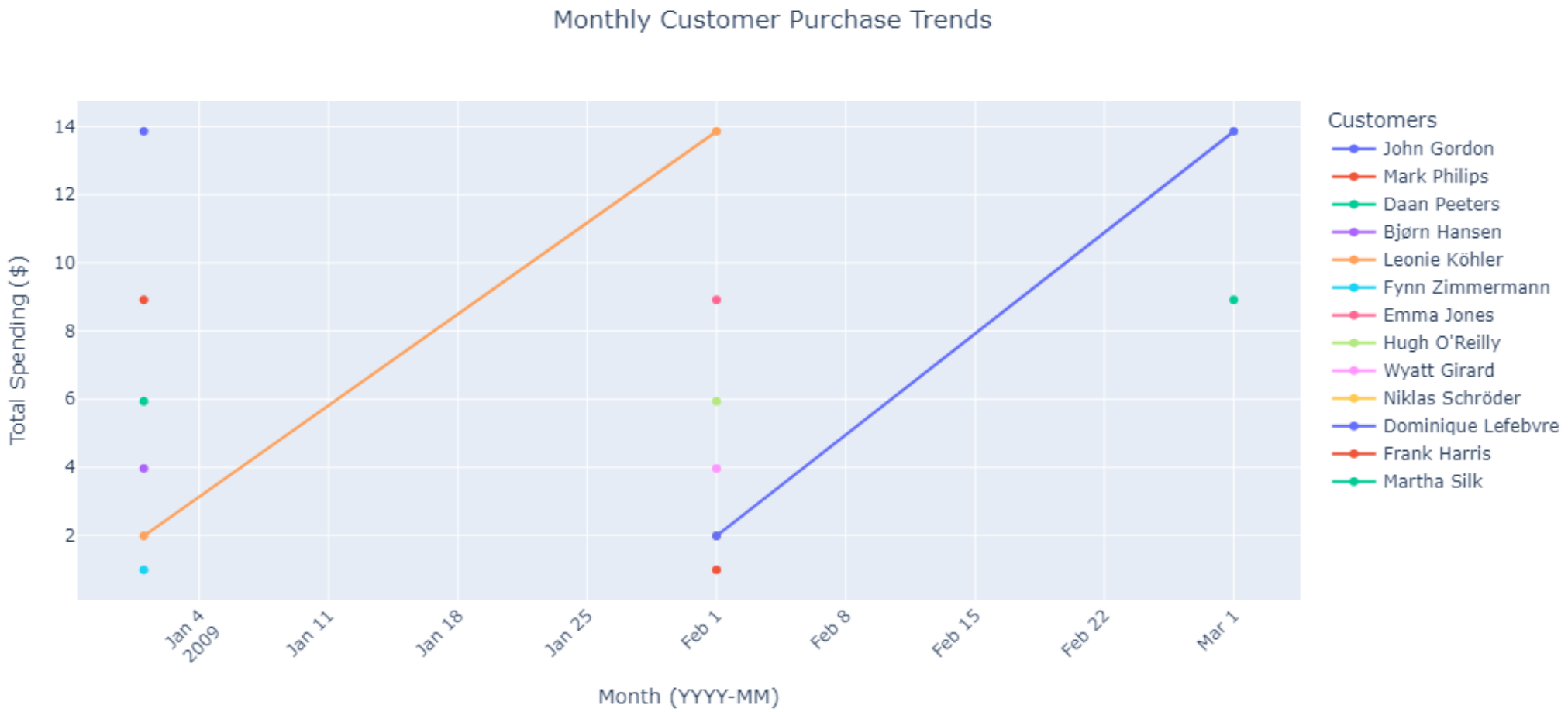
4.1 Customer Purchase Trends Over Time

```
[162]  1 query = '''
       2 SELECT
       3     Customer.CustomerId,
       4     Customer.FirstName || ' ' || Customer.LastName AS FullName,
       5     strftime('%Y-%m', Invoice.InvoiceDate) AS Month,
       6     SUM(Invoice.Total) AS TotalSpent
       7 FROM Customer
       8 JOIN Invoice ON Customer.CustomerId = Invoice.CustomerId
       9 GROUP BY Customer.CustomerId, Month
      10 ORDER BY Month ASC, TotalSpent DESC
      11 LIMIT 15;
      12 '''
      13
      14 # Execute Query and Display Results
      15 try:
      16     print("Top 15 Monthly Customer Purchase Trends:")
      17     start_time = time.time()
      18     df_top_15_Monthly_Customer = pd.read_sql_query(query, connection)
      19     end_time = time.time()
      20     execution_time = round((end_time - start_time) * 1000, 2)  # Calculate the time taken
      21     Track_Query_Time['Monthly Customer Purchase Trends'] = execution_time
      22     print(f"✅ Query executed successfully in {execution_time:.2f} milliseconds")
      23     display(df_top_15_Monthly_Customer)
      24 except Exception as e:
      25     print(f"❌ Error executing optimized query: {e}")
```

```
Top 15 Monthly Customer Purchase Trends:
✅ Query executed successfully in 4.21 milliseconds
```

	CustomerId	FullName	Month	TotalSpent
0	23	John Gordon	2009-01	13.86
1	14	Mark Philips	2009-01	8.91
2	8	Daan Peeters	2009-01	5.94
3	4	Bjørn Hansen	2009-01	3.96
4	2	Leonie Köhler	2009-01	1.98
5	37	Fynn Zimmermann	2009-01	0.99
6	2	Leonie Köhler	2009-02	13.86

4.1 Visualizations



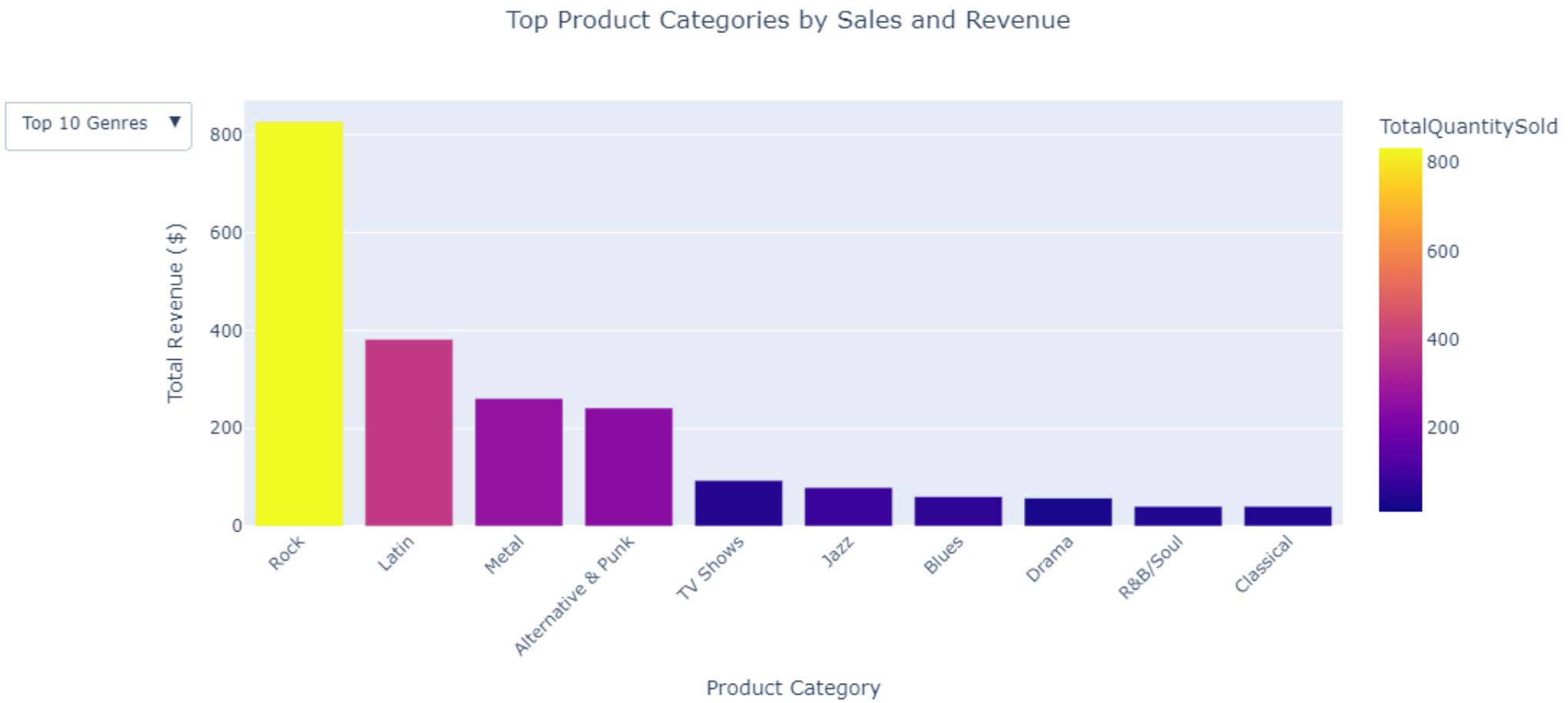
4.2 Product Affinity Analysis

[165]
```
query = '''
SELECT
  Genre.Name AS Category,
  SUM(InvoiceLine.Quantity) AS TotalQuantitySold,
  SUM(InvoiceLine.Quantity * InvoiceLine.UnitPrice) AS TotalRevenue
FROM InvoiceLine
JOIN Track on InvoiceLine.TrackId = Track.TrackId
JOIN Genre on Track.GenreId = Genre.GenreID
GROUP BY Category
ORDER BY TotalRevenue DESC
LIMIT 15
'''


# Execute Query and Display Results
try:
    print("Top 15 Product Categories by Sales and Revenue:")
    start_time = time.time()
    df_product_category_sales = pd.read_sql_query(query, connection)
    end_time = time.time()
    execution_time = round((end_time - start_time) * 1000, 2)  # Calculate the time taken
    Track_Query_Time['Product Category Sales Distribution'] = execution_time
    print(f"✅ Query executed successfully in {execution_time:.2f} milliseconds")
    display(df_product_category_sales)
except Exception as e:
    print(f"❌ Error executing optimized query: {e}")
```

```
Top 15 Product Categories by Sales and Revenue:
✅ Query executed successfully in 4.96 milliseconds
```

	Category	TotalQuantitySold	TotalRevenue
0	Rock	835	826.65
1	Latin	386	382.14
2	Metal	264	261.36
3	Alternative & Punk	244	241.56
4	TV Shows	47	93.53
5	Jazz	80	79.20
6	Blues	61	60.39

4.2 Visualizations



4.3 Customer Segmentation (RFM Analysis)

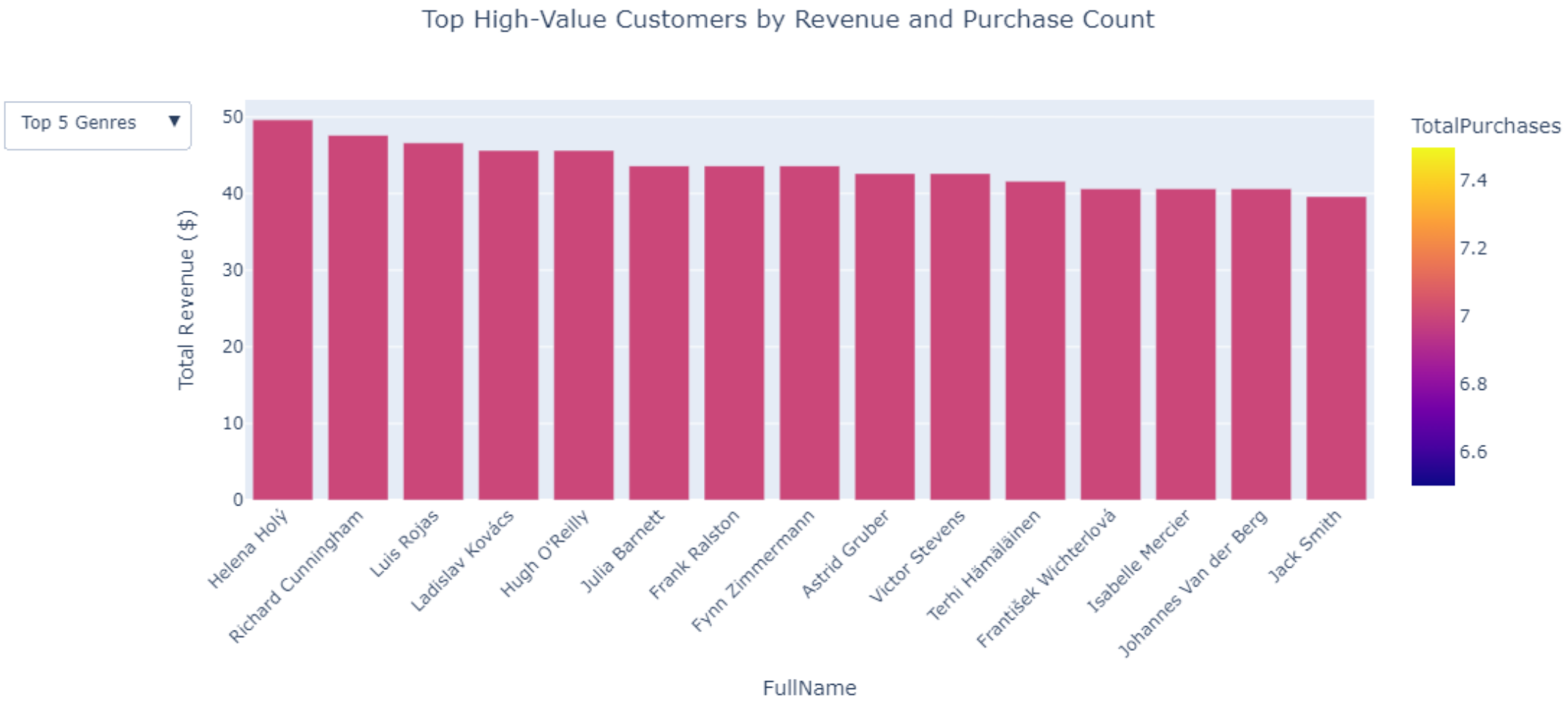
```sql
query = '''
SELECT
    Customer.CustomerId,
    Customer.FirstName || ' ' || Customer.LastName AS FullName,
    COUNT(Invoice.InvoiceId) AS TotalPurchases,
    SUM(Invoice.Total) AS TotalRevenue
FROM Customer
JOIN Invoice ON Customer.CustomerId = Invoice.CustomerId
GROUP BY Customer.CustomerId
ORDER BY TotalRevenue DESC
LIMIT 15;
'''
# Execute Query and Display Results
try:
    print("Top 15 High-Value Customers by Revenue and Purchase Count:")
    start_time = time.time()
    df_high_value_customers = pd.read_sql_query(query, connection)
    end_time = time.time()
    execution_time = round((end_time - start_time) * 1000, 2)  # Calculate the time taken
    Track_Query_Time['High-Value Customers'] = execution_time
    print(f"✅ Query executed successfully in {execution_time:.2f} milliseconds")
    display(df_high_value_customers)
except Exception as e:
    print(f"❌ Error executing optimized query: {e}")
```

```
Top 15 High-Value Customers by Revenue and Purchase Count:
✅ Query executed successfully in 9.55 milliseconds
```

	CustomerId	FullName	TotalPurchases	TotalRevenue
0	6	Helena Holý	7	49.62
1	26	Richard Cunningham	7	47.62
2	57	Luis Rojas	7	46.62
3	45	Ladislav Kovács	7	45.62
4	46	Hugh O'Reilly	7	45.62
5	28	Julia Barnett	7	43.62
6	24	Frank Ralston	7	43.62
7	37	Fynn Zimmermann	7	43.62

Sprints

4.3 Visualizations



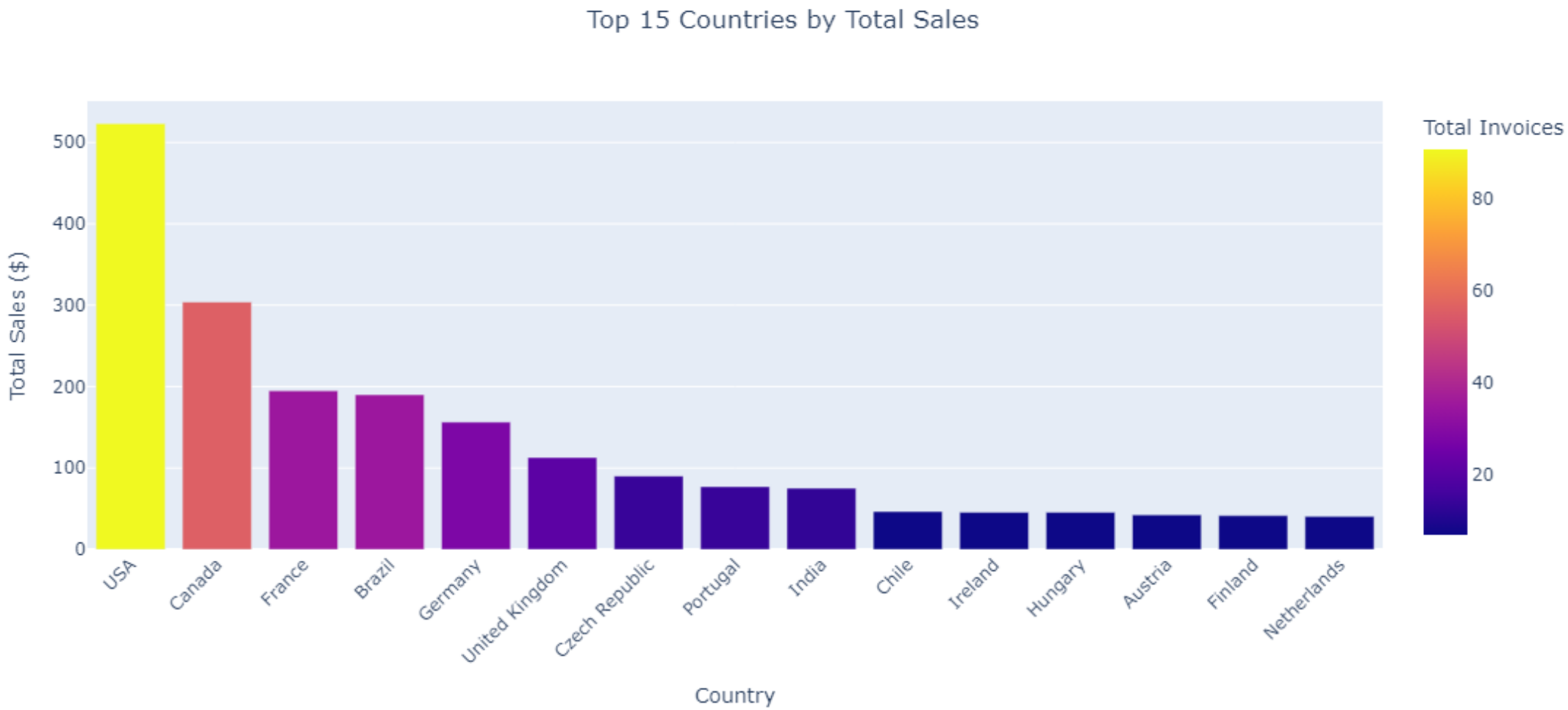
4.4 Geographical Sales Insights

```
[170]  1 query = '''
       2 SELECT
       3   BillingCountry AS Country,
       4   SUM(Total) AS TotalSales,
       5   COUNT(InvoiceID) AS TotalInvoices
       6 FROM INVOICE
       7 GROUP BY Country
       8 ORDER BY TotalSales DESC
       9 LIMIT 15;
      10 '''
      11
      12 # Execute Query and Display Results
      13 try:
      14     print("Top 15 Countries by Total Sales:")
      15     start_time = time.time()
      16     df_sales_by_country = pd.read_sql_query(query, connection)
      17     end_time = time.time()
      18     execution_time = round((end_time - start_time) * 1000, 2)  # Calculate the time taken
      19     Track_Query_Time['Geographical Sales Analysis'] = execution_time
      20     print(f"✅ Query executed successfully in {execution_time:.2f} milliseconds")
      21     display(df_sales_by_country)
      22 except Exception as e:
      23     print(f"❌ Error executing optimized query: {e}")
```

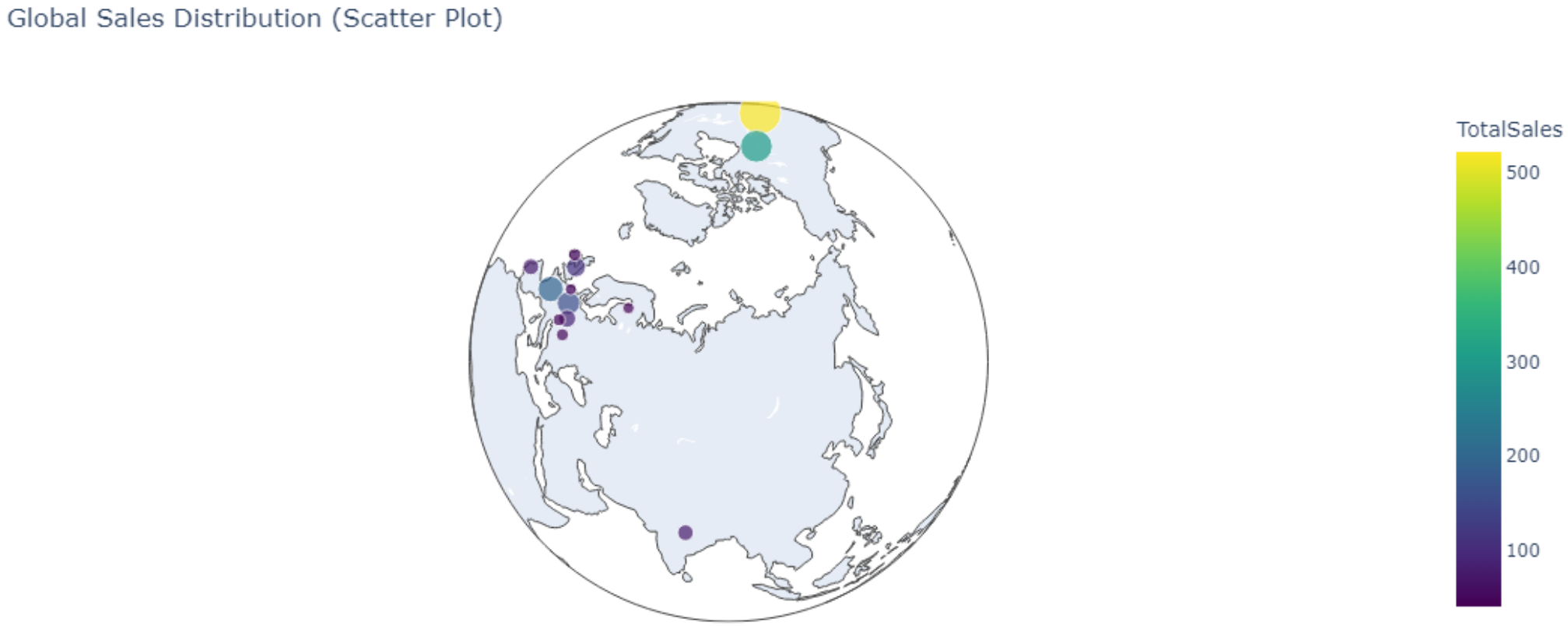
```
Top 15 Countries by Total Sales:
✅ Query executed successfully in 2.15 milliseconds
```

	Country	TotalSales	TotalInvoices
0	USA	523.06	91
1	Canada	303.96	56
2	France	195.10	35
3	Brazil	190.10	35
4	Germany	156.48	28
5	United Kingdom	112.86	21
6	Czech Republic	90.24	14
7	Portugal	77.24	14

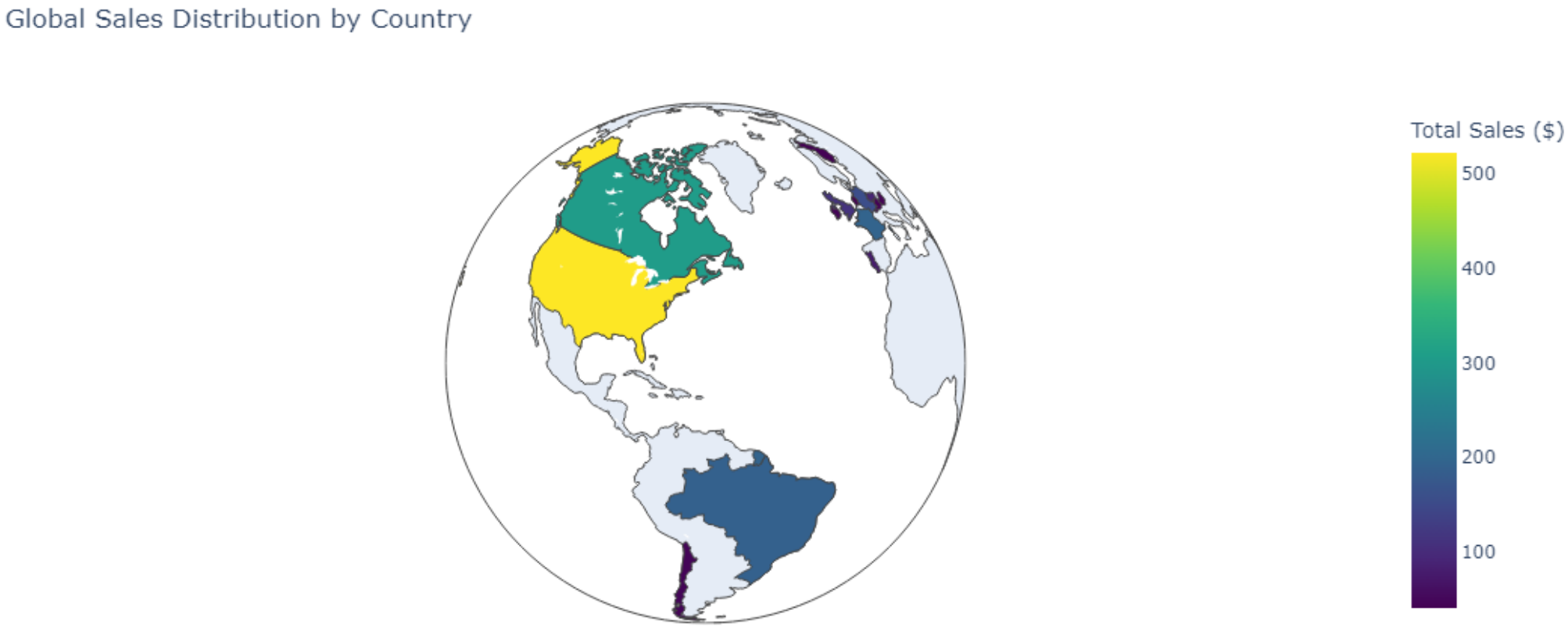
4.4 Visualizations



4.4 Visualizations



4.4 Visualizations



4.5 Sales by Genre or Artis

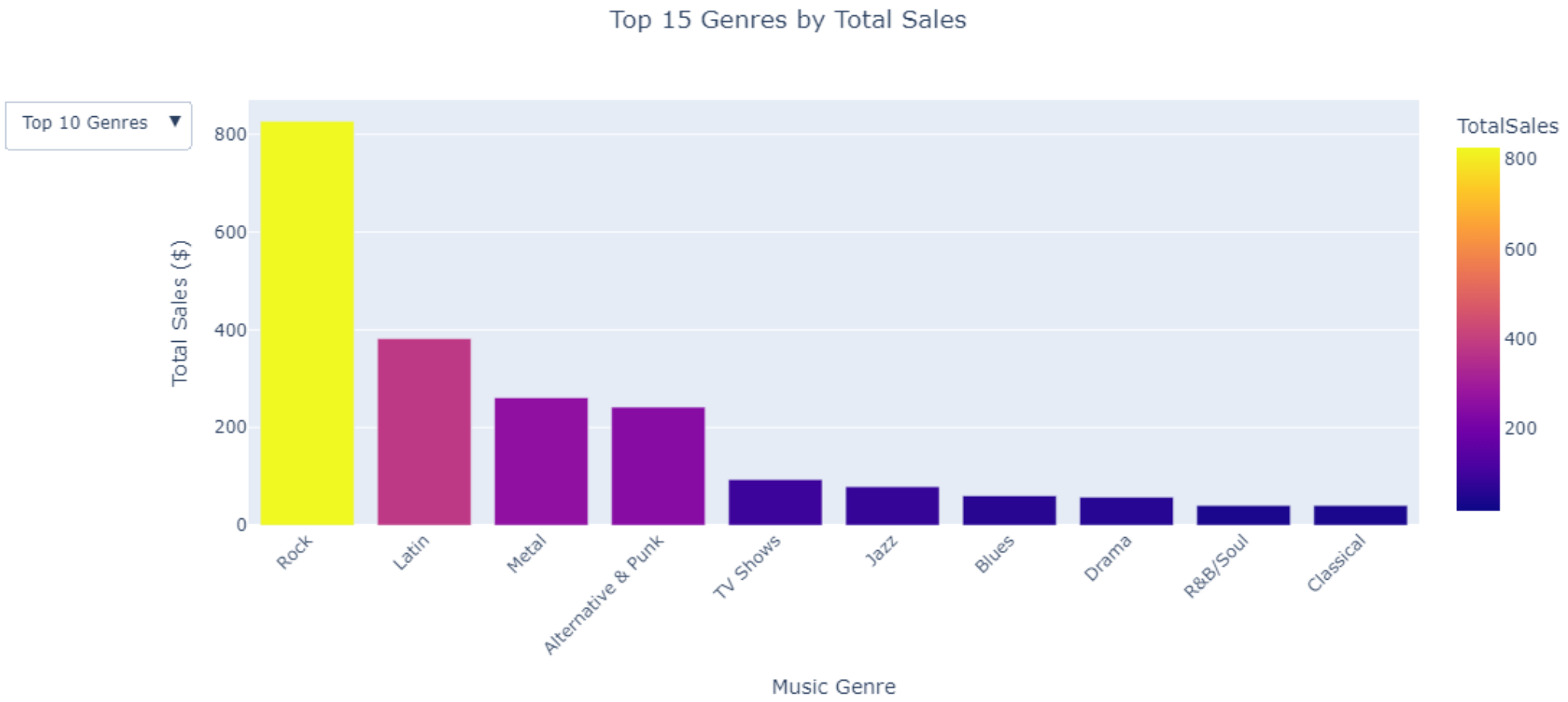
```
[175]  1 # SQL Query for Sales by Genre or Artist
       2 query = '''
       3 SELECT
       4     Genre.Name AS Category,
       5     SUM(InvoiceLine.UnitPrice * InvoiceLine.Quantity) AS TotalSales
       6 FROM Genre
       7 JOIN Track ON Genre.GenreId = Track.GenreId
       8 JOIN InvoiceLine ON Track.TrackId = InvoiceLine.TrackId
       9 GROUP BY Category
      10 ORDER BY TotalSales DESC
      11 LIMIT 15;
      12 '''
      13
      14 # Execute Query and Display Results
      15 try:
      16     print("Top 15 Genres by Total Sales:")
      17     start_time = time.time()
      18     df_sales_by_genre = pd.read_sql_query(query, connection)
      19     end_time = time.time()
      20     execution_time = round((end_time - start_time) * 1000, 2)
      21     Track_Query_Time['Sales by Genre or Artist'] = execution_time
      22     print(f"✅ Query executed successfully in {execution_time:.2f} milliseconds")
      23     display(df_sales_by_genre)
      24 except Exception as e:
      25     print(f"❌ Error executing query: {e}")
      26
```

```
Top 15 Genres by Total Sales:
✅ Query executed successfully in 3.55 milliseconds
```

	Category	TotalSales
0	Rock	826.65
1	Latin	382.14
2	Metal	261.36
3	Alternative & Punk	241.56
4	TV Shows	93.53

Sprints

4.5 Visualizations



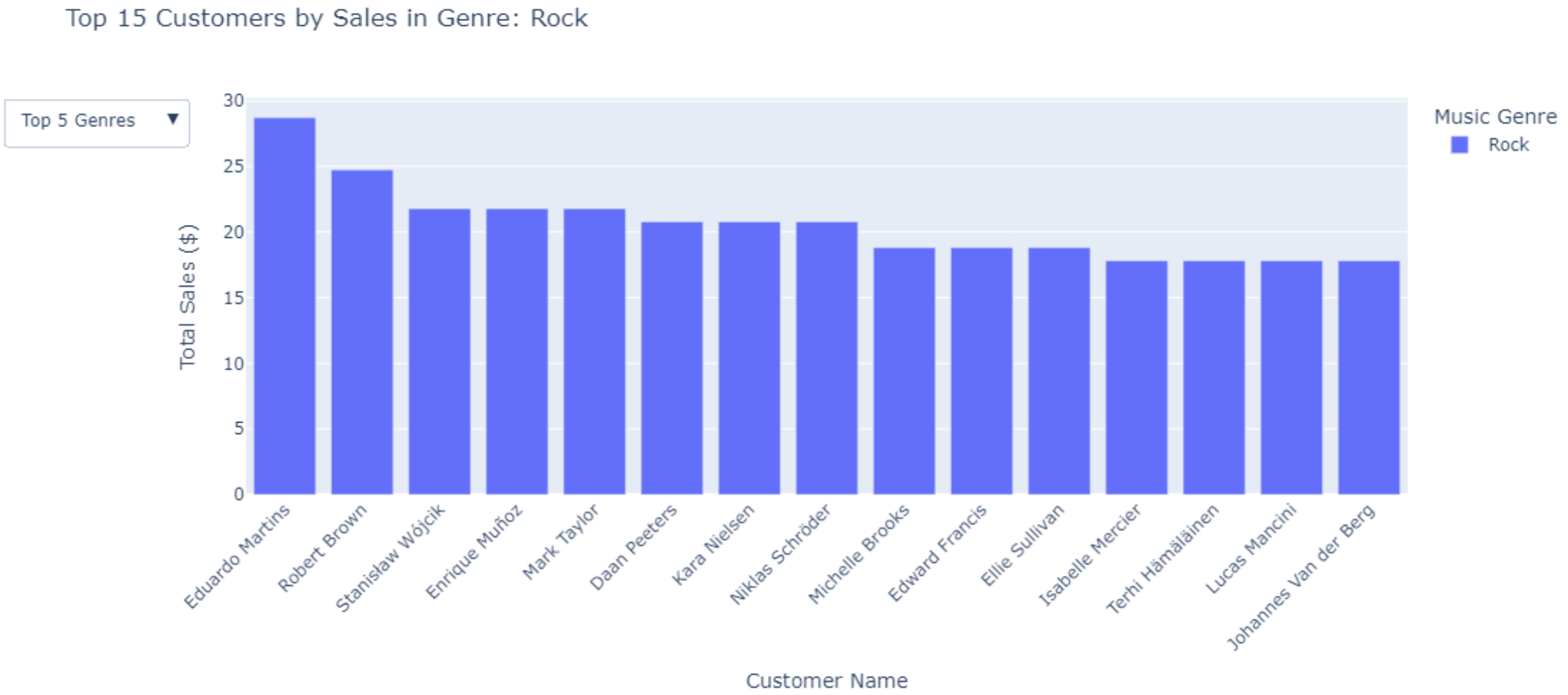
4.6 Dynamic Query Parameterization

```
[179]  1 # Interactive Dashboard Query
       2 query = '''
       3 SELECT
       4     Customer.FirstName || ' ' || Customer.LastName AS FullName,
       5     Genre.Name AS Category,
       6     Artist.Name AS ArtistName,
       7     SUM(InvoiceLine.UnitPrice * InvoiceLine.Quantity) AS TotalSales,
       8     COUNT(Invoice.InvoiceId) AS TotalInvoices,
       9     AVG(Invoice.Total) AS AvgInvoiceValue
      10 FROM Customer
      11 JOIN Invoice ON Customer.CustomerId = Invoice.CustomerId
      12 JOIN InvoiceLine ON Invoice.InvoiceId = InvoiceLine.InvoiceId
      13 JOIN Track ON InvoiceLine.TrackId = Track.TrackId
      14 JOIN Album ON Track.AlbumId = Album.AlbumId
      15 JOIN Artist ON Album.ArtistId = Artist.ArtistId
      16 JOIN Genre ON Track.GenreId = Genre.GenreId
      17 GROUP BY Customer.CustomerId, Category, ArtistName
      18 ORDER BY TotalSales DESC
      19 LIMIT 15;
      20 '''
      21
      22 # Execute Query and Display Results
      23 try:
      24     print("Interactive Dashboard: Top 15 Customers by Artist and Genre")
      25     start_time = time.time()
      26     df_dashboard = pd.read_sql_query(query, connection)
      27     end_time = time.time()
      28     execution_time = round((end_time - start_time) * 1000, 2)
      29     print(f"✅ Query executed successfully in {execution_time:.2f} milliseconds")
      30     display(df_dashboard)
      31 except Exception as e:
      32     print(f"❌ Error executing dashboard query: {e}")
      33
```

```
Interactive Dashboard: Top 15 Customers by Artist and Genre
✅ Query executed successfully in 8.04 milliseconds
```

	FullName	Category	ArtistName	TotalSales	TotalInvoices	AvgInvoiceValue
0	Hugh O'Reilly	TV Shows	Lost	13.93	7	21.86
1	Helena Holý	TV Shows	Lost	9.95	5	25.86

4.6 Visualizations



4.7 Interactive Dashboard Integration

```python
# Interactive Dashboard Query
query = '''
SELECT
    Customer.FirstName || ' ' || Customer.LastName AS FullName,
    Genre.Name AS Category,
    Artist.Name AS ArtistName,
    SUM(InvoiceLine.UnitPrice * InvoiceLine.Quantity) AS TotalSales,
    COUNT(Invoice.InvoiceId) AS TotalInvoices,
    AVG(Invoice.Total) AS AvgInvoiceValue
FROM Customer
JOIN Invoice ON Customer.CustomerId = Invoice.CustomerId
JOIN InvoiceLine ON Invoice.InvoiceId = InvoiceLine.InvoiceId
JOIN Track ON InvoiceLine.TrackId = Track.TrackId
JOIN Album ON Track.AlbumId = Album.AlbumId
JOIN Artist ON Album.ArtistId = Artist.ArtistId
JOIN Genre ON Track.GenreId = Genre.GenreId
GROUP BY Customer.CustomerId, Category, ArtistName
ORDER BY TotalSales DESC
LIMIT 15;
'''

# Execute Query and Display Results
try:
    print("Interactive Dashboard: Top 15 Customers by Artist and Genre")
    start_time = time.time()
    df_dashboard = pd.read_sql_query(query, connection)
    end_time = time.time()
    execution_time = round((end_time - start_time) * 1000, 2)
    print(f"✅ Query executed successfully in {execution_time:.2f} milliseconds")
    display(df_dashboard)
except Exception as e:
    print(f"❌ Error executing dashboard query: {e}")

```

Sprints

4.7 Visualizations