Toronto Metropolitan University

LEVERAGING DATA ANALYTICS FOR SALES OPTIMIZATION AND REVENUE GROWTH

Source Codes

Prepared by

Rabia Danish

Toronto Metropolitan University

PYSPARK CODES

Data statistics:

```
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)
df_amazon = sqlContext.read.format('com.databricks.spark.csv').options(header='true',
inferSchema='true').load('/user/root/project/amazonv2_normalised.csv')

df_amazon.select("actual_price").summary("count", "mean", "stddev", "min", "max").show()
df_amazon.select("discounted_price").summary("count", "mean", "stddev", "min",
"max").show()
df_amazon.select("discount_percentage").summary("count", "mean", "stddev", "min",
"max").show()
df_amazon.select("rating").summary("count", "mean", "stddev", "min", "max").show()
df_amazon.select("rating_count").summary("count", "mean", "stddev", "min", "max").show()
df_amazon.select("Sales_volume").summary("count", "mean", "stddev", "min", "max").show()
df_amazon.select("revenue").summary("count", "mean", "stddev", "min", "max").show()
df_amazon.select("discount_effectiveness").summary("count", "mean", "stddev", "min",
"max").show()
df_amazon.select("inventory_performance").summary("count", "mean", "stddev", "min",
"max").show()
df_amazon.select("NormalizedSalesVolume").summary("count", "mean", "stddev", "min",
"max").show()
df_amazon.select("NormalizedRevenue").summary("count", "mean", "stddev", "min",
"max").show()
```

Data cleaning:

```
spark.sql("""
CREATE OR REPLACE TEMP VIEW amazon_new AS
SELECT
product_id,
product_name,
category,
CAST(REGEXP_REPLACE(discounted_price, '[₹,]', '') AS FLOAT) AS discounted_price,
CAST(REGEXP_REPLACE(actual_price, '[₹,]', '') AS FLOAT) AS actual_price,
CAST(REGEXP_REPLACE(discount_percentage, '[%]', '') AS FLOAT) AS discount_percentage,

CAST(REGEXP_REPLACE(rating_count, '[,]', '') AS FLOAT) AS rating_count,

rating
FROM Amazon
""").show()
```

Deriving new columns:

```
spark.sql("""
CREATE OR REPLACE TEMP VIEW amazonv2_derived_columns AS
SELECT *,
(rating * rating_count) AS Sales_volume,
(discounted_price * (rating * rating_count)) AS revenue,
FROM amazon_sales_update
""").show()
```

Data normalization:

```
spark.sql("""
CREATE OR REPLACE TEMP VIEW amazon_normalised_data AS
SELECT
product_id,
category,
discounted_price,
actual_price,
discount_percentage,
rating,
rating_count,
Sales_volume,
revenue,
(Sales_volume - MIN(Sales_volume) OVER ()) /
(MAX(Sales_volume) OVER () - MIN(Sales_volume) OVER ()) AS NormalizedSalesVolume,
(revenue - MIN(revenue) OVER ()) /
(MAX(revenue) OVER () - MIN(revenue) OVER ()) AS NormalizedRevenue,
(inventory_performance - MIN(inventory_performance) OVER ()) /
FROM amazon_sales
""").show()
```

Category analysis:

```
spark.sql("""
CREATE OR REPLACE TEMP VIEW category_analysis AS
SELECT
category,
AVG(NormalizedSalesVolume) AS AvgNormalizedSalesVolume,
AVG(NormalizedRevenue) AS AvgNormalizedRevenue,
AVG(NormalizedInventoryPerformance) AS AvgNormalizedInventoryPerformance,
AVG(rating) AS AvgRating,
AVG(discount_percentage) AS AvgDiscountPercentage
```

```
FROM amazon_sales
GROUP BY category
""").show()
```

Discount analysis:

```
spark.sql("""
CREATE OR REPLACE TEMP VIEW discount_analysis_new AS
SELECT
category,
CASE
WHEN discount_percentage <= 0.1 THEN '0-10%'
WHEN discount_percentage > 0.1 AND discount_percentage <= 0.2 THEN '10-20%'
WHEN discount_percentage > 0.2 AND discount_percentage <= 0.3 THEN '20-30%'
WHEN discount_percentage > 0.3 AND discount_percentage <= 0.4 THEN '30-40%'
WHEN discount_percentage > 0.4 AND discount_percentage <= 0.5 THEN '40-50%'
ELSE '50%+'
END AS DiscountRange,
AVG(NormalizedSalesVolume) AS AvgNormalizedSalesVolume,
AVG(NormalizedInventoryPerformance) AS AvgNormalizedInventoryPerformance,
AVG(rating) AS AvgRating,
AVG(discount_percentage) AS AvgDiscountPercentage,
AVG(NormalizedRevenue) AS AvgNormalizedRevenue
FROM amazon_sales
GROUP BY category, DiscountRange
""").show()
```

Rating analysis:

```
spark.sql("""
CREATE OR REPLACE TEMP VIEW rating_analysis AS
SELECT
category,
CASE
WHEN rating <= 2 THEN 'Low Ratings (1.0-2.0)'
WHEN rating > 2 AND rating <= 4 THEN 'Medium Ratings (2.1-4.0)'
ELSE 'High Ratings (4.1-5.0)'
END AS RatingRange,
COUNT(*) AS ProductCount,
AVG(NormalizedSalesVolume) AS AvgNormalizedSalesVolume,
AVG(NormalizedRevenue) AS AvgNormalizedRevenue,
AVG(NormalizedInventoryPerformance) AS AvgNormalizedInventoryPerformance,
```

AVG(discount_percentage) AS AvgDiscountPercentage
FROM amazon_sales
GROUP BY category, RatingRange
""").show()

HIVE CODES

Count of products sold per category:
select category, count(product_id) AS count_of_products_sold from amazon_norm_data_2024 group by category order by count_of_products_sold DESC limit 10;

Categories with the highest average revenue:
select category, cast(avg(revenue) as bigint) AS avg_revenue from amazon_norm_data_2024 group by category order by avg_revenue DESC limit 10;

Categories with highest average ratings:
select category, avg(rating) AS average_rating from amazon_norm_data_2024 group by category order by average_rating DESC limit 15;

Categories with highest average discount_percentage:
select category, avg(discount_percentage) AS avg_discount_percentage from amazon_norm_data_2024 group by category order by avg_discount_percentage DESC limit 15;

Most expensive categories:
select category, avg(discounted_price) AS avg_discounted_price from amazon_norm_data_2024 group by category order by avg_discounted_price DESC limit 15;