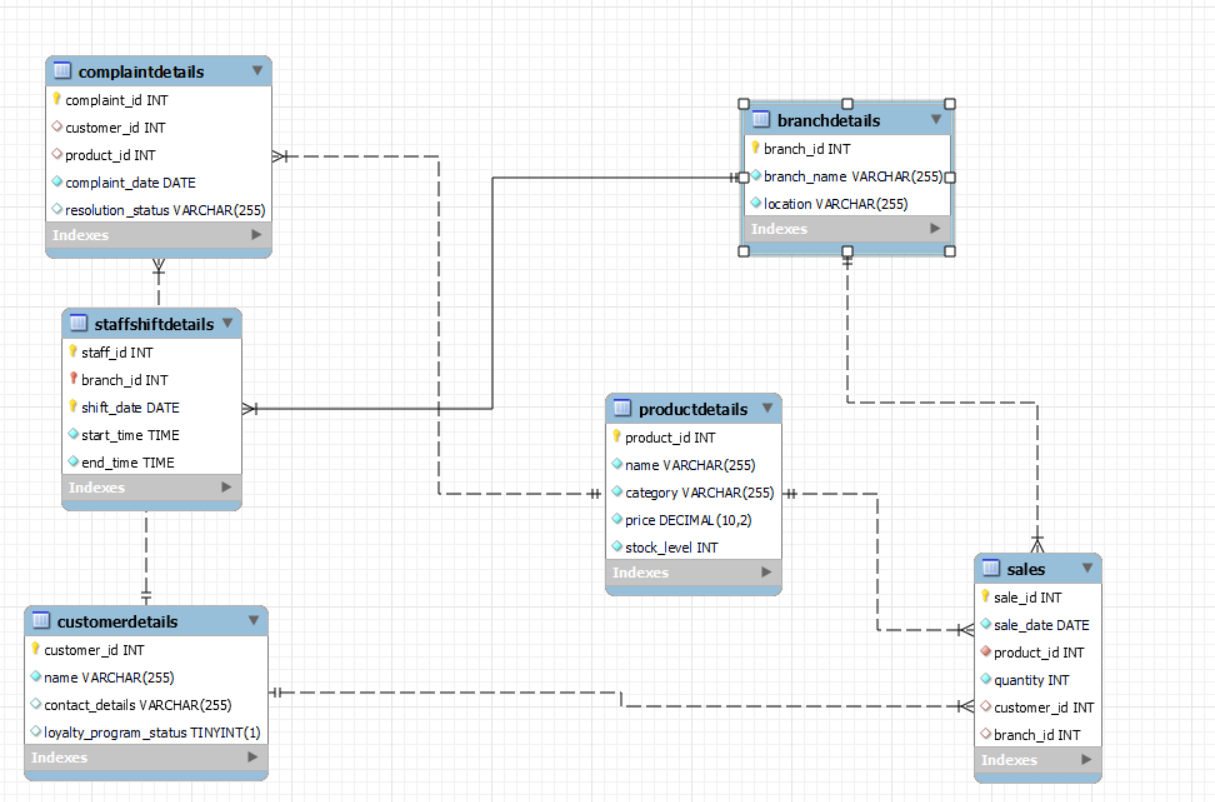
Aritha Mindula Jayaratne

s4683873  Data mining and warehousing for Business

Assessment 2 – Project (Ausieretailors).

ER diagram.



 Entity-relationship Diagram (ERD) taken from the database designed is described. An Entity Relationship Diagram (ERD) is a visual aid that illustrates the connections among individuals, items, locations, ideas, and occurrences in an information system. This is a summary based on the typical elements of an ERD.

Tables in the database.

* complaintdetails: Complaints from the customers clients.
* Staff work shift: staff shift details are stored in this table.
* customerdetails: Contains data about the customers.
* productdetails:  products' specifications.
* branchdetails: Lists the different supermarket branches of the supermarket.
* sales: Records all the sales transactions.

Primary keys of the tables.

These keys serve as each table's distinct record identifiers. Usually, a key symbol is placed next to the attribute name to indicate them. Complaint\_id in complaintdetails and product\_id in productdetails are two examples based on the created database.

Foreign Keys of the tables.

Foreign keys are characteristics in one table that create a relationship with the primary key in another table. They let you link related information between tables. Some examples from the database are given here for better understanding.

* Product\_id in complaintdetails indicates the product about which a complaint is made, and links to product\_id in productdetails.
* The buyer's identity is indicated by the link between customer\_id in sales and customer\_id in customerdetails.
* The staff member's place of employment is indicated by the link between branch\_id in branchdetails and branch\_id in staffshiftdetails.

Relationships based on the diagram.

One-to-Many: Multiple complaints can be filed by a single customer (from customerdetails), but each complaint is linked to a specific customer.One-to-Many: Multiple complaints (in complaint details) may be made about a single product (from productdetails), but each complaint relates to a different product.  
  
Many-to-One: Several shifts (found in staffshiftdetails) may take place at a single branch (found in branchdetails); however, every shift is linked to a specific branch.  
  
  
Many-to-One: Each sale record relates to a single product, but multiple sales (in sales) may involve the same product (based on product details).  
  
Many-to-One: A customer can make several purchases (in sales) (from customerdetails), but every sale is associated with a single customer.  
Many-to-One: Using branch details, it is possible to make multiple sales (in sales) at a single branch, with each sale record pointing to a different branch.

|  |  |  |  |
| --- | --- | --- | --- |
| Question. | Code of the Query | Screenshot | Analysis |
| 1. List all products and their categories with sales greater than 100 units. | SELECT  p.product\_id,  p.name AS product\_name,  p.category,  SUM(s.quantity) AS total\_quantity\_sold  FROM  productdetails p  JOIN  sales s ON p.product\_id = s.product\_id  GROUP BY  p.product\_id  HAVING  SUM(s.quantity) > 100; |  | After creating the table there is no product which has sold more than 100 units in the given tables. There for it is safe to come into a conclusion that no product in the supermarket has been sold more than 100 times. |
| 1. Calculate the total revenue per branch, considering the quantity sold and product prices. | SELECT  s.branch\_id,  SUM(s.quantity \* p.price) AS total\_revenue  FROM  sales s  INNER JOIN  productdetails p ON s.product\_id = p.product\_id  GROUP BY  s.branch\_id  ORDER BY  total\_revenue DESC  LIMIT 10; |  | By taking into consideration the quantity sold and the product price columns, and by calculating the total revenue per branch these are the results gained by the given code. The results show the top 10 highest branches with highest revenue. By analyzing the table generated the branch with highest total revenue is branch 38 with 818.00 and the second and third is branch 18 and 40. |
| 1. Identify customers who have made purchases in more than 3 different branches. | SELECT  cd.customer\_id,  cd.name AS customer\_name,  COUNT(DISTINCT s.branch\_id) AS branches\_visited  FROM  sales s  JOIN  customerdetails cd ON s.customer\_id = cd.customer\_id  GROUP BY  cd.customer\_id, cd.name  HAVING  COUNT(DISTINCT s.branch\_id) > 3; |  | The customer id shows a unique identifier for the customer. Customer name shows the customer’s name and the branches visited by each customer. Customer 51 has visited 7 different branches. Customer 52,53,54 has visited different branches 4 times each. Customer 55,59 and 60 has visited 5 different branches each. And customer 56 and 58 has visited 6 different branches. This table can be used to understand the customer engagement across different geographical locations relating to the business. |
| 1. Determine the average sale quantity of products by category for sales made in the last quarter. | SELECT pd.category, AVG(s.quantity) AS AverageQuantity  FROM sales s  JOIN productdetails pd ON s.product\_id = pd.product\_id  WHERE s.sale\_date BETWEEN '2022-10-01' AND '2022-12-31'  GROUP BY pd.category; |  | The query calculates the average sale quantity of the products a, grouped by the category for the sales made in the last quarter. The table generated has two columns. Category column represents different categories in the supermarket. These represents different categories which are mentioned in the database. Average quantity shows the average quantity of each product sold in the relating category. When analyzing the data in the generated table, category 7 and 9 has the highest average sale quantity, both categories are averaging 9.0000 units sold. Category 10 has the lowest average sales quantity and the other categories falls under the other two extremes ranging from 4.0000 – 8.0000. The table shows the product categories are performing well in terms of the units sold which can be used in inventory planning for the supermarket. |
| 1. Rank products within each category based on the total sales quantity using a window function. | SELECT  product\_id,  name,  category,  total\_quantity,  RANK() OVER(PARTITION BY category ORDER BY total\_quantity DESC) AS rank\_within\_category  FROM (  SELECT  p.product\_id,  p.name,  p.category,  SUM(s.quantity) AS total\_quantity  FROM  productdetails p  LEFT JOIN  sales s ON p.product\_id = s.product\_id  GROUP BY  p.product\_id, p.name, p.category  ) AS product\_sales; |  | From the query a table was generated with following columns, product ID, name, category, total quantity, and the rank within category. When analyzing the table from a business analytics perspective the above could be derived, product 20 is the best-selling product in category 1 with a highest rank of 1. Also, there is a possibility the ranking between some products is the same. For example, in category 10 there are products with the same rank of 3. Another main thing to notice is the null value for the product 60, this means there may be a lack of sales data for the product, in this case there should be further investigation as to why this happened. This type of table and data can be used to identify the top performing products and categories. |
| 1. Show the month-over-month percentage growth in sales for the top 5 products by total quantity sold. | WITH ranked\_products AS (  -- Selecting the top 5 products by total quantity sold  SELECT  product\_id,  SUM(quantity) AS total\_quantity,  RANK() OVER (ORDER BY SUM(quantity) DESC) AS product\_rank  FROM  sales  GROUP BY  product\_id  ORDER BY  total\_quantity DESC  LIMIT 5  ),  monthly\_sales AS (  -- Calculating total quantity sold for each of the top 5 products for each month  SELECT  YEAR(sale\_date) AS year,  MONTH(sale\_date) AS month,  product\_id,  SUM(quantity) AS total\_quantity  FROM  sales  WHERE  product\_id IN (SELECT product\_id FROM ranked\_products)  GROUP BY  YEAR(sale\_date),  MONTH(sale\_date),  product\_id  )  -- Calculating month-over-month percentage growth in sales for the top 5 products  SELECT  m1.year,  m1.month,  m1.product\_id,  (m1.total\_quantity - COALESCE(LAG(m1.total\_quantity) OVER (PARTITION BY m1.product\_id ORDER BY m1.year, m1.month), 0)) / COALESCE(LAG(m1.total\_quantity) OVER (PARTITION BY m1.product\_id ORDER BY m1.year, m1.month), 1) AS percentage\_growth  FROM  monthly\_sales m1  JOIN  ranked\_products rp ON m1.product\_id = rp.product\_id  WHERE  rp.product\_rank <= 5  ORDER BY  m1.year,  m1.month,  m1.product\_id; |  | The query generates a table with 4 columns. They are year, month, product id and percentage growth. The data on the table shows several products which are identified by the product id column and show how the products performed and if they have been shrunk or grown from one month to the next 2022 year. Product 34 has a growth rate of 14% in the month of march and gain the sales have dropped by 0.50% in the month of April. And another analysis we can perform is about the product 51 which had a 10% growth rate in April and there is a 0% growth in May. Product 17 and 55 shows a fluctuations in their month over month sales percentage. There are some growths and some decline across the different months. The information can be used understand sales trends and seasonal impacts on the products. |
| 1. Find all products that have not been sold in the last 6 months (from June to December 2022) but have stock levels above 50. | SELECT \*  FROM productdetails  WHERE product\_id NOT IN (  SELECT DISTINCT product\_id  FROM sales  WHERE sale\_date BETWEEN '2022-06-01' AND '2022-12-31'  )  AND stock\_level > 50; |  | The following query produces a table with the mentioned columns. Product id, name, category, price, and the stock level. When analyzing the table produce by the query, it shows the details about each product, and the category which the product belongs to and its stock level. This table is beneficial for supply chain department when making new orders and making predictions based on the products which are sold in the past for make future predictions. |
| 1. Calculate the total number of complaints lodged against products in each category. | SELECT  p.category,  COUNT(c.complaint\_id) AS total\_complaints  FROM  productdetails p  LEFT JOIN  complaints c ON p.product\_id = c.product\_id  GROUP BY  p.category; |  | The query creates a table with 2 columns. The aim is to see how many complaints have been filed for each product. When analyzing the data in the table. The may conclusions I came into conclusion is that category 6 has highest number of complaints with a total of 11. Categories 3,2,and 8 have the similar number of complaints which is 8. And another thing to mention is that category 7 and 4 has the lowest number of complaints. The data can be used identify categories which may need major attention to improve product quality. |
| 1. List the top 10 customers by total spending and show their most frequently bought product category. | SELECT  cd.customer\_id,  cd.name AS customer\_name,  SUM(s.quantity \* p.price) AS total\_spending,  MAX(p.category) AS most\_frequent\_category  FROM  sales s  JOIN  customerdetails cd ON s.customer\_id = cd.customer\_id  JOIN  productdetails p ON s.product\_id = p.product\_id  GROUP BY  cd.customer\_id, cd.name  ORDER BY  total\_spending DESC  LIMIT  10; |  | The query generates a table with following columns, customer id, customer name, total spending , most frequent category. This query will generate and display the top 10 customers by their total spendings and also the most frequently bought item by the customer. According to the table customer 51 has the highest spending which is 1666 in currency the most frequently bought item is from the category 9. And the next is the customers 58,56 and 60 and they have spent nearly 800 in currency. The most frequent item by these customers is from the category 8. Customer s 52 and 55 ha the same frequently bought items from category 6 but customer 52 has the most spent in total. This data can be used understand customer behaviors, preferences, and the effectiveness of the sales strategies. |
| 1. Identify days of the week with the highest sales transactions volume. | SELECT  DAYNAME(s.sale\_date) AS day\_of\_week,  COUNT(\*) AS transaction\_volume  FROM  sales s  GROUP BY  DAYNAME(s.sale\_date)  ORDER BY  transaction\_volume DESC; |  | According to the table generated by the query the table has two columns. One is days of week. The days listed in the column is from Monday to Sunday. And the other column is the transaction volume. This shows the number of sales happened on each day. Tuesday has the highest number sales transactions and Thursday has the lowest number of transactions occurred. And the lowest number of transactions occurred is Friday which is a total of 5. The data generated suggest that the beginning and middle of the weeks are busier and end of the week is low. But this could be wrong based on other factors taken into consideration. Business ca use to improve staff allocation, stock allocation and promotional activities. |
| 1. Determine the branch with the lowest stock levels across all products. | SELECT  s.branch\_id,  SUM(p.stock\_level - IFNULL(sq.total\_quantity\_sold, 0)) AS estimated\_remaining\_stock  FROM  branchdetails b  LEFT JOIN sales s ON b.branch\_id = s.branch\_id  LEFT JOIN (  SELECT  product\_id,  SUM(quantity) AS total\_quantity\_sold  FROM  sales  GROUP BY  product\_id  ) sq ON s.product\_id = sq.product\_id  LEFT JOIN productdetails p ON s.product\_id = p.product\_id  GROUP BY  s.branch\_id  ORDER BY  estimated\_remaining\_stock ASC  LIMIT 1; |  | The query creates a table with two columns. The columns are Branch Id to identify each branch by a ID. Estimated remaining stock column calculates the remaining stock in each branch. The list is listed in the ascending order of the remaining stocks of each branch. If we perform an analysis on the table branch 23 has the lowest stock level when compared with other branches. The data generated from this query is real helpful in inventory management cause this highlights which branches need replenishing urgently before next week. Also, a further investigation needs to be done as to why there are null values on the table. |
| 1. Analyze the correlation between loyalty program status and the average transaction value per customer. | SELECT  cd.loyalty\_program\_status,  COUNT(DISTINCT cd.customer\_id) AS number\_of\_customers,  AVG(s.transaction\_value) AS average\_transaction\_value  FROM  customerdetails cd  JOIN  (SELECT  customer\_id,  sale\_id,  SUM(quantity \* p.price) AS transaction\_value  FROM  sales  JOIN  productdetails p ON sales.product\_id = p.product\_id  GROUP BY  customer\_id, sale\_id) s ON cd.customer\_id = s.customer\_id  GROUP BY  cd.loyalty\_program\_status; |  | The query creates a table which displays the customers average transaction values in relation to the loyalty program status of the customers. Loyalty program status shows if the customers are a part of the loyalty program or not. Number of customers show the count of each customer who are a part of the program of not. Average transaction value gives an average value of transaction for each group. When analyzing the data provided 15 customers are not loyalty program with an average transaction percentage of 129.482143. And the customers with the loyalty program have a count of 22 and the average transaction value of 153.984375. in conclusion customers who has a loyalty program spend more than customers who does not have a loyalty program. |
| 1. Calculate the average duration between complaint registration and resolution. | SELECT  AVG(DATEDIFF(complaint\_date, complaint\_date)) AS average\_resolution\_duration  FROM  complaintdetails  WHERE  resolution\_status = 'Resolved'; |  | The number that is shown is 0.0000, which means that the query's average resolution duration is 0. This result indicates that the query may not have been written correctly since it appears to deduct the complaint registration date (complaint\_date) from the total time, which is zero. There should be two distinct date columns in your dataset—complaint\_date and resolution\_date—to accurately determine the average time between the filing of a complaint and its resolution. Next, for each complaint, the SQL query should calculate the difference between these two dates and determine the average of these disparities. |
| 1. Identify staff members with shifts longer than 8 hours and list their corresponding branches and shift dates. | SELECT  ss.staff\_id,  b.branch\_name,  ss.shift\_date,  TIMEDIFF(ss.end\_time, ss.start\_time) AS shift\_duration  FROM  staffshiftdetails ss  JOIN  branchdetails b ON ss.branch\_id = b.branch\_id  WHERE  TIMEDIFF(ss.end\_time, ss.start\_time) > '08:00:00'; |  | SQL query that has most likely determined the shifts of employees. The following columns are visible:staff\_id: A staff member's identifier. branch\_name: The employee's name and the name of the branch they were employed by. shift\_date: The employee's date of work for the shift. shift\_duration: The amount of time the employee put in during that particular shift. The findings seem to indicate that different employees work different shifts at different branches and on different dates, with varying shift durations. Look for shift\_duration times that are more than '08:00:00' to find the employees who work shifts longer than eight hours. As an illustration:On 2022-12-28, a staff member at Branch 15 with staff\_id 1 put in a 9 hours and 39 minute shift. On May 31, 2022, Staff Member 24 worked a shift at Branch 37 that lasted for ten hours and twenty-one minutes.  These are only a few examples from the table when the shifts are longer than eight hours. A shift that is longer than eight hours is indicated by any shift\_duration value greater than '08:00:00'. |
| 1. Find the product with the highest number of complaints and detail the nature of these complaints. | SELECT  pd.product\_id,  pd.name AS product\_name,  COUNT(c.complaint\_id) AS number\_of\_complaints  FROM  complaintdetails c  JOIN  productdetails pd ON c.product\_id = pd.product\_id  GROUP BY  pd.product\_id  ORDER BY  number\_of\_complaints DESC; |  | The SQL query generates a table comprising the subsequent columns:product\_id: Each product's special identification number. product\_name: The product's name. number\_of\_complaints: The total number of grievances filed against a product. The number of complaints column is most often used to sort the list in descending order. With four complaints, the product at the top, Product 15 with product\_id 15, has the most. Another SQL query would be needed to select the complaint data for that product in order to describe the nature of these complaints for Product 15. This would entail filtering for product\_id 15 and combining the complaints table with the product table on product\_id. A description or categorization of the complaint may be included in the complaint details and listed as part of the output. |
| 1. What is the average stock level for each product category? | SELECT  category, -- The product category  AVG(stock\_level) AS average\_stock\_level -- The average stock level for each category  FROM  productdetails -- From the product details table  GROUP BY  category; -- Grouping the results by product category |  | Most likely, the outcome of a SQL query displaying the average stock level for each product category is displayed in the image you posted. Based on the typical structure of these result grids, the following is an explanation: category: The numerous product classes or types found in the database are indicated by the listing of the product categories (e.g., Category 1, Category 2, etc.). average\_stock\_level: The average stock level of each category's products is displayed in this column. The values appear to be numerical, and they most likely indicate the average quantity of a product that is offered across all products in a category. Based on the values shown: With 95.0000 units, Category 10 has the highest average stock level.At 50.0000 units, Category 1 has the lowest average stock level. The average stock levels for the other categories range from 55.0000 to 90.0000 units.  Understanding which categories may be overstocked and which may need replenishment is made easier with the use of this data, which is beneficial for supply chain optimization and inventory management. It also helps to spot patterns in the availability of products across various inventory segments. |
| 1. Which product has the highest average sales price within each category? | SELECT  pp.category,  pp.product\_name,  pp.average\_price AS highest\_average\_price  FROM (  SELECT  p.category,  p.name AS product\_name,  AVG(s.quantity \* p.price) / AVG(s.quantity) AS average\_price,  ROW\_NUMBER() OVER(PARTITION BY p.category ORDER BY AVG(s.quantity \* p.price) / AVG(s.quantity) DESC) AS rn  FROM  sales s  JOIN  productdetails p ON s.product\_id = p.product\_id  GROUP BY  p.product\_id, p.name  ) AS pp  WHERE pp.rn = 1; |  | The table listing the products with the highest average sales price within each product category, like SQL query result grids. This is an overview of the data that is normally shown: category: The product categories listed in this column are denoted by the labels Category 1, Category 2, and so forth.product\_name: Product names that, based on average sales price, are probably the priciest in their respective categories.highest\_average\_price: Displays, as a function of time or recorded transactions, the average sales price of the product that is listed alongside it. Based on the given structure, it seems that: At 45.00000000, Product 70 has the highest average price for Category 1.  The average price of Product 69 in Category 10 and Product 68 in Category 9 are around 44.00000000.  With Category 4 having the lowest average price among the highest in each category, the other categories' products have the highest average prices, which range from roughly 36.50000000 to 43.4997825011.  This information can be used to guide pricing strategies, advertising campaigns, and inventory decisions by illuminating which products in each category are in high demand. Given the prices' precision, it is possible that they were calculated with a high degree of accuracy, which is required for some financial analyses. |
| 1. How many customers are repeat buyers, and what is their average spending? | SELECT  COUNT(customer\_id) AS repeat\_buyers,  AVG(total\_spending) AS average\_spending  FROM (  SELECT  s.customer\_id,  SUM(p.price \* s.quantity) AS total\_spending,  COUNT(s.sale\_id) AS number\_of\_purchases  FROM  sales s  JOIN  productdetails p ON s.product\_id = p.product\_id  GROUP BY  s.customer\_id  HAVING  COUNT(s.sale\_id) > 1  ) AS RepeatBuyers; |  | The sql quesry genrates Two-column SQL query result grid:  repeat\_buyers: 21 represents the number of customers who have made multiple purchases. average\_spending: The mean total of the purchases made by these loyal customers. The average spending in this instance is 525.047619. According to this result set, there are 21 customers who have made purchases more than once, and they have collectively spent about 525.05 units of currency on average. Businesses need this kind of information to identify their core customer base, which is a major contributor to sales. To improve customer retention and raise the lifetime value of each customer, it can be helpful to understand repeat buying behavior and apply it to customer loyalty programs, targeted marketing campaigns, and strategic decision-making. |
| 1. What is the month with the highest number of complaints, and what are the common themes? | SELECT  EXTRACT(YEAR\_MONTH FROM complaint\_date) AS month,  COUNT(complaint\_id) AS number\_of\_complaints  FROM  complaintdetails  GROUP BY  month  ORDER BY  number\_of\_complaints DESC  LIMIT 1; |  | SQL query using the enumerated columns.month: The month and year are displayed in this column together in a combined numerical format; in this example, the month is January 2021, or 202201.number\_of\_complaints: In this case, 10, the total number of complaints recorded for that month. According to this data, 10 complaints were made in January 2021. Considering the context of your question, this is probably the highest number for a single month. |
| 1. Compare the sales performance of loyalty program members versus non-members. | SELECT  cd.loyalty\_program\_status,  COUNT(s.sale\_id) AS total\_sales\_transactions,  SUM(s.quantity) AS total\_units\_sold,  SUM(s.quantity \* pd.price) AS total\_revenue  FROM  sales s  JOIN  customerdetails cd ON s.customer\_id = cd.customer\_id  JOIN  productdetails pd ON s.product\_id = pd.product\_id  GROUP BY  cd.loyalty\_program\_status; |  | SQL query result comparing members of loyalty programs' sales performance to that of non-participants. What each column stands for is as follows loyalty\_program\_status: Returns a value of 1 (for members, 0 for non-members) indicating the customer's status in the loyalty program.total\_sales\_transactions: The total amount of sales that each group has accomplished. total\_units\_sold: The total quantity of merchandise sold to every category. total\_revenue: The overall amount of money made by every group. Based on the information provided.Members of the loyalty program (loyalty\_program\_status = 1) have completed 64 sales transactions as opposed to non-members' (28). In total, members have bought 354 units as opposed to non-members' 147. Revenue from members is substantially higher than revenue from non-members (3625.50) at 9855.00. According to this data, members of loyalty programs appear to be more engaged and make greater contributions to sales and revenue. |