Project Overview

The SQL portfolio project aims to analyze sales data and derive insights using SQL queries. The project includes 10 queries that focus on different aspects of the data.

Technologies Used

SSMS(SQL Server Management Studio)

Project Structure

The project is organized into separate SQL queries, each addressing a specific analysis objective. The queries are self-contained and can be executed independently.

Data Source

The data used for the analysis is sourced from the [AdventureWorksDW2019] database. It includes tables such as [FactInternetSales], [DimCustomer], [DimGeography], and [DimProduct]. The data has been preprocessed and cleaned, ready for analysis.

Query 1: Sales by Customer Age Group

In this query, I aim to conduct a comprehensive analysis of sales data based on customer age groups for each country. To achieve this, I calculate the customer age by determining the difference between their birthdate and the order date. Once the results are obtained, I organize and group them by country and age group. This analysis will provide me with valuable insights into the purchasing behavior of different age segments in each country, enabling me to identify any variations or similarities in consumer preferences and trends across different regions. Understanding how customer age impacts sales patterns in various countries will allow me to tailor marketing strategies and product offerings to effectively cater to the unique needs and preferences of diverse customer demographics.

```sql
WITH CTE_MAIN AS
(
SELECT T3.EnglishCountryRegionName,
DATEDIFF(YEAR, BirthDate, ORDERDATE) AS AGE, ---Shows customer's ages when they ordered
SalesOrderNumber
FROM [AdventureWorksDW2019].[dbo].[FactInternetSales] T1
JOIN AdventureWorksDW2019.DBO.DimCustomer T2
ON T1.CustomerKey = T2.CustomerKey
JOIN AdventureWorksDW2019.DBO.DimGeography T3
ON T2.GeographyKey = T3.GeographyKey
)
```

```
---Creating the age groups using CASE WHEN Statements to show the sales by age
groups---
SELECT EnglishCountryRegionName,
    CASE WHEN AGE < 30 THEN 'a: Under 30'
    WHEN AGE BETWEEN 30 AND 40 THEN 'b: 30 - 40'
    WHEN AGE BETWEEN 40 AND 50 THEN 'c: 40 - 50'
    WHEN AGE BETWEEN 50 AND 60 THEN 'd: 50 - 60'
    WHEN AGE > 60 THEN 'e: over 60'
    ELSE 'Other'
    END AS Age_group,
    COUNT(SalesOrderNumber) AS Sales
FROM CTE_MAIN
GROUP BY EnglishCountryRegionName,
    CASE WHEN AGE < 30 THEN 'a: Under 30'
    WHEN AGE BETWEEN 30 AND 40 THEN 'b: 30 - 40'
    WHEN AGE BETWEEN 40 AND 50 THEN 'c: 40 - 50'
    WHEN AGE BETWEEN 50 AND 60 THEN 'd: 50 - 60'
    WHEN AGE > 60 THEN 'e: over 60'
    ELSE 'Other'
    END
    ORDER BY EnglishCountryRegionName, Age_group
```

Query 2: Product Sales by Age Group

The purpose of this query is to perform an in-depth analysis of product sales based on customer age groups. To achieve this, I calculate the customer age by determining the difference between their birthdate and the order date. The results obtained from this query are then organized and grouped by product type and age group. This analysis will provide valuable insights into the purchasing behavior of different age segments and help identify any potential trends or preferences based on customer age. By understanding how various age groups interact with different product types, I can tailor marketing strategies and product offerings to better meet the needs and preferences of our diverse customer base.

```
WITH CTE_MAIN AS
(
SELECT T4.EnglishProductName,
    DATEDIFF(YEAR, BirthDate, ORDERDATE) AS AGE,
---Shows customer's when they ordered---
```

```
    SalesOrderNumber,
    T5.EnglishProductSubcategoryName
FROM [AdventureWorksDW2019].[dbo].[FactInternetSales] T1
JOIN AdventureWorksDW2019.DBO.DimCustomer T2
ON T1.CustomerKey = T2.CustomerKey
JOIN AdventureWorksDW2019.DBO.DimGeography T3
ON T2.GeographyKey = T3.GeographyKey
JOIN AdventureWorksDW2019.DBO.DimProduct T4
ON T1.ProductKey = T4.ProductKey
JOIN [AdventureWorksDW2019].[DBO].[DimProductSubcategory] T5
ON T4.ProductSubcategoryKey = T5.ProductSubcategoryKey
)
---Creating the age groups using CASE WHEN Statements to show the product sales
by age groups---
SELECT
       EnglishProductSubcategoryName AS Product_Type,
    CASE WHEN AGE < 30 THEN 'a: Under 30'
    WHEN AGE BETWEEN 30 AND 40 THEN 'b: 30 - 40'
    WHEN AGE BETWEEN 40 AND 50 THEN 'c: 40 - 50'
    WHEN AGE BETWEEN 50 AND 60 THEN 'd: 50 - 60'
    WHEN AGE > 60 THEN 'e: over 60'
    ELSE 'Other'
    END AS Age_Group,
    COUNT(SalesOrderNumber) AS Sales
FROM CTE_MAIN
GROUP BY
    EnglishProductSubcategoryName,
    CASE WHEN AGE < 30 THEN 'a: Under 30'
    WHEN AGE BETWEEN 30 AND 40 THEN 'b: 30 - 40'
    WHEN AGE BETWEEN 40 AND 50 THEN 'c: 40 - 50'
    WHEN AGE BETWEEN 50 AND 60 THEN 'd: 50 - 60'
    WHEN AGE > 60 THEN 'e: over 60'
    ELSE 'Other'
    END
    ORDER BY Product_Type, Age_Group
```

Query 3: Monthly sales for Australia and USA compared for year 2012

The purpose of this query is to compare the monthly sales between Australia and the USA specifically for the year 2012. My objective is to conduct an in-depth analysis to comprehend the sales trends and performance of these two countries during this particular period. To achieve this, I designed the query to filter and include only the sales data from Australia and the USA while restricting the dataset to the year 2012. This analysis will provide valuable insights into the sales dynamics and patterns of these two countries, allowing me to make informed decisions and strategies to optimize sales and market performance in both regions.

```
SELECT SUBSTRING(CAST ([OrderDateKey] as char),1,6) AS Month_Key
---The month and year of the sales (formatted as YYYYMM)
      ,[SalesOrderNumber]---The unique identifier of the sales order
      ,[OrderDate]---The date of the sales order
      ,T2.SalesTerritoryCountry---The country of the sales territory
  FROM [AdventureWorksDW2019].[dbo].[FactInternetSales] T1
  JOIN [AdventureWorksDW2019].[dbo].DimSalesTerritory T2
  ON T1.SalesTerritoryKey = T2.SalesTerritoryKey
  WHERE SalesTerritoryCountry IN ('Australia', 'United States')
  AND SUBSTRING(CAST ([OrderDateKey] as char),1,4) = '2012'
---Limits to year 2012
```

Query 4: First Reorder Date for Each Product

The purpose of this query is to determine the first reorder date for each product. My main objective is to identify the specific point in time when a product requires reordering based on the sales data. By running this query, I will be able to obtain valuable insights into the optimal timing for initiating reorders, ensuring that we maintain sufficient stock levels to meet customer demand without excess inventory. This analysis will aid in enhancing our inventory management practices, optimizing supply chain operations, and ultimately improving customer satisfaction through timely product availability.

```
--- Query Utilizes two Common Table Expressions (CTEs) - 'MAIN_CTE' and
'REORDER_DATE '
WITH MAIN_CTE AS
--- Calculates the total sales quantity for each product by aggregating the
data from the `DimProduct` and `FactInternetSales` tables
(
SELECT
```

```sql
      [EnglishProductName] --- Product name in english
      ,OrderDateKey
      ,[SafetyStockLevel]
      ,[ReorderPoint]
      ,SUM(T2.OrderQuantity) AS SALES
  FROM [AdventureWorksDW2019].[dbo].[DimProduct] T1
  JOIN DBO.FactInternetSales T2 ON T1.ProductKey = T2.ProductKey
  GROUP BY
    [EnglishProductName]
    ,OrderDateKey
      ,[SafetyStockLevel]
      ,[ReorderPoint]
),
REORDER_DATE
---Determines the first reorder date for each product based on the safety stock
level and running total sales quantity
AS
(
        SELECT *, CASE WHEN (SAFETYSTOCKLEVEL - RUNNING_TOTAL_SALES) <=
REORDERPOINT THEN 1 ELSE 0 END AS REORDER_FLAG
        FROM
(SELECT *, SUM(SALES) OVER (PARTITION BY ENGLISHPRODUCTNAME ORDER BY
ORDERDATEKEY) AS RUNNING_TOTAL_SALES
FROM MAIN_CTE
GROUP BY
     [EnglishProductName]
      ,OrderDateKey
      ,[SafetyStockLevel]
      ,[ReorderPoint]
      ,SALES) MAIN_SQ
      )
SELECT EnglishProductName, MIN(ORDERDATEKEY) AS First_Reorder_Date---The first
date when the product needs to be reordered
FROM REORDER_DATE
WHERE REORDER_FLAG = 1---identifies if the safety stock level is lower than or
equal to the running total sales quantity. If the flag is 1, it indicates that
a reorder is required.
GROUP BY EnglishProductName
```

Query 5: Days Between First Order and First Reorder Date

The purpose of this query is to calculate the number of days between a product's first order date and its first reorder date. My main objective is to identify products that took over one year to need reordering, as this could indicate excessive stock levels for those items. To achieve this, I built upon the results from Query 4 and added a new column to display the number of days between the product's first order and first reorder date. My focus lies on products that required over one year to be reordered. The final result set includes products with a difference of more than 365 days between their first order and first reorder date, allowing me to identify potential stock excess and pinpoint potential issues with inventory management. This analysis will provide valuable insights into optimizing inventory levels and enhancing our overall supply chain efficiency.

```
---Query Utilizes two Common Table Expressions (CTEs) - 'MAIN_CTE' and
'REORDER_DATE '
WITH MAIN_CTE AS
---retrieves sales and relevant details for each product by joining the
DimProduct and FactInternetSales tables and grouping the data
(
SELECT
      [EnglishProductName]
      ,OrderDateKey
      ,[SafetyStockLevel]
      ,[ReorderPoint]
      ,SUM(T2.OrderQuantity) AS SALES
  FROM [AdventureWorksDW2019].[dbo].[DimProduct] T1
  JOIN DBO.FactInternetSales T2 ON T1.ProductKey = T2.ProductKey
  GROUP BY
    [EnglishProductName]
    ,OrderDateKey
      ,[SafetyStockLevel]
      ,[ReorderPoint]
),
REORDER_DATE
---determines the first reorder date for each product based on the safety stock
level and running total sales quantity. It includes a REORDER_FLAG column to
identify if a reorder is needed.
AS
(
SELECT *,
```

```sql
CASE WHEN (SAFETY STOCK LEVEL - RUNNING_TOTAL_SALES) <= REORDERPOINT
THEN 1 ELSE 0 END AS REORDER_FLAG
FROM
(SELECT *, SUM(SALES) OVER (PARTITION BY ENGLISHPRODUCTNAME ORDER BY
ORDERDATEKEY) AS RUNNING_TOTAL_SALES
FROM MAIN_CTE
GROUP BY
     [EnglishProductName]
      ,OrderDateKey
      ,[SafetyStockLevel]
      ,[ReorderPoint]
      ,SALES) MAIN_SQ
      )
SELECT EnglishProductName, MAX(products_first_orderdate) AS
products_first_orderdate, MAX(first_reorder_date) AS first_reorder_date,
datediff(day, MAX(CAST(CAST(products_first_orderdate AS char) AS date)),
MAX(CAST(CAST(first_reorder_date AS char) AS date))) AS days_to_reorder
---Retrieves the number of days between the first order and first reorder date
FROM
(
SELECT EnglishProductName, MIN(ORDERDATEKEY) AS products_first_orderdate,NULL
AS first_reorder_date
FROM MAIN_CTE
GROUP BY EnglishProductName
UNION ALL---Vertically Combines results from the two subqueries into a single
result
SELECT EnglishProductName, NULL AS products_first_orderdate, MIN(ORDERDATEKEY)
AS first_reorder_date
FROM REORDER_DATE
WHERE REORDER_FLAG = 1---identifies if the safety stock level is lower than or
equal to the running total sales quantity. If the flag is 1, it indicates that
a reorder is required.
GROUP BY EnglishProductName
) main_subquery
GROUP BY EnglishProductName
HAVING DATEDIFF(day, MAX(CAST(CAST(products_first_orderdate AS char) AS date)),
MAX(CAST(CAST(first_reorder_date AS char) AS date))) > 365
--DATEDIFF operator used to calculate the difference in days between first
order date and first reorder date
```

Query 6: Sales Promotion Discount Analysis

The purpose of this SQL query is to retrieve all sales transactions that are associated with a promotion and calculate the new sales value after applying a 25% discount.

```sql
-- Select the OrderDate, SalesReasonName, SalesOrderNumber, and calculate the
sales_amount_after_discount
SELECT OrderDate, SalesReasonName, t1.SalesOrderNumber, (SalesAmount*0.75) AS
sales_amount_after_discount
FROM [AdventureWorksDW2019].[dbo].[FactInternetSales] t1
-- Join with the FactInternetSalesReason table to link sales transactions with
reasons
JOIN [AdventureWorksDW2019].[dbo].[FactInternetSalesReason] t2
    ON t1.SalesOrderNumber = t2.SalesOrderNumber
-- Join with the DimSalesReason table to retrieve the sales reason name
JOIN [AdventureWorksDW2019].[dbo].[DimSalesReason] t3
    ON t2.SalesReasonKey = t3.SalesReasonKey
-- Filter the results to include only sales transactions associated with the
promotion
WHERE SalesReasonName = 'On Promotion'
```

Query 7: Customer First and Last Sale Analysis

The purpose of this SQL query is to analyze sales data for each customer, with a specific focus on their first and last sales. I designed the query to retrieve essential information, including the customer key, the sales value of their initial purchase, the sales value of their latest purchase, and the calculated difference between the two. By comparing customers' first and last purchases, this query provides valuable insights into their sales behavior. The identified differences can be instrumental in pinpointing customers who have undergone significant changes in their purchase patterns or overall sales trends, allowing for a more comprehensive understanding of customer behavior and preferences.

```sql
WITH first_purchase AS
```

```sql
(
---Retrieves the customer key, sales amount, and order date for the first
purchase of each customer
SELECT
    [CustomerKey],
    [SalesAmount],
    [OrderDate],
    ROW_NUMBER() OVER (PARTITION BY [CustomerKey] ORDER BY [OrderDate] ASC) AS
purchase_num
FROM [AdventureWorksDW2019].[dbo].[FactInternetSales]
),
last_purchase AS
(
---Retrieve the customer key, sales amount, and order date for the last
purchase of each customer
SELECT
    [CustomerKey],
    [SalesAmount],
    [OrderDate],
    ROW_NUMBER() OVER (PARTITION BY [CustomerKey] ORDER BY [OrderDate] ASC) AS
purchase_num
FROM [AdventureWorksDW2019].[dbo].[FactInternetSales]
)

SELECT
    customerkey,
    SUM(first_purchase_value) AS first_purchase_value,
    SUM(last_purchase_value) AS last_purchase_value,
    (SUM(first_purchase_value) - SUM(last_purchase_value)) AS difference
FROM
(
--- Combine the results of the first and last purchases using union all join
SELECT
    CustomerKey,
    SalesAmount AS first_purchase_value, --- Sales value of the first purchase
    NULL AS last_purchase_value --- Set as NULL since this represents the first
purchase
 FROM first_purchase
 WHERE purchase_num = 1 --- Filter to select only the first purchase records

UNION ALL
SELECT
    CustomerKey,
    NULL AS first_purchase_value, --- Set as NULL since this represents the
last purchase
    SalesAmount AS last_purchase_value --- Sales value of the last purchase
FROM last_purchase
WHERE purchase_num = 1 --- Filter to select only the last purchase records
) main_subquery
GROUP BY customerkey
HAVING (SUM(first_purchase_value) - SUM(last_purchase_value)) <> 0 --- Exclude
customers with equal first and last purchase values
ORDER BY customerkey
```

Query 8: Filtered Bike Sales Analysis by Commute Distance and Country

I built upon the previous Query to create this one. In this Query, I focused on filtering bike sales based on the income brackets of our customers. To achieve this, I categorized customer income and calculated the count of purchases for each income bracket. The analysis obtained from this Query will offer valuable insights into our customers' purchasing behaviors in relation to their income, allowing us to identify trends and patterns that can aid in better understanding our target audience.

```
WITH bike_sales AS
(
---Retrieves relevant columns for bike sales, joining necessary tables and
applying filters
SELECT
    orderdatekey,
    OrderDate,
    CustomerKey,
    BirthDate,
    YearlyIncome,
    TotalChildren,
    CommuteDistance,
    englishcountryregionname AS Country,
    SalesAmount,
    SalesOrderNumber
FROM [AdventureWorksDW2019].[dbo].[FactInternetSales] T1
```

```
 JOIN [AdventureWorksDW2019].[dbo].[DimCustomer] T2 ON T1.CustomerKey =
T2.CustomerKey
 JOIN [AdventureWorksDW2019].[dbo].[DimGeography] T3 ON T2.Geographykey =
T3.Geographykey
 JOIN [AdventureWorksDW2019].[dbo].[DimProduct] T4 ON T1.ProductKey =
T4.ProductKey
 JOIN [AdventureWorksDW2019].[dbo].[DimProductSubcategory] T5 ON
T4.ProductSubcategoryKey = T5.ProductSubcategoryKey
 WHERE EnglishProductSubcategoryName IN ('Mountain Bikes', 'Touring Bikes',
'Road Bikes')
)

SELECT
    Country,
    CommuteDistance,
    COUNT(DISTINCT SalesOrderNumber) AS sales
FROM bike_sales
GROUP BY Country, CommuteDistance
ORDER BY Country, CommuteDistance
```

Query 9: Bike Sales Analysis by Customer and Income Brackets.

This Query builds upon the previous Query.This Query filters bike sales based on the customer's income brackets. This is done by creating categories for customer's income and calculating the count of purchases for each bracket. This analysis can be used to provide insights about purchasing behaviors relating to their income by helping identify trends and patterns.

```
SELECT CustomerKey,
---Income brackets created using the CASE Statement based on their yearly
income
CASE WHEN YearlyIncome < 50000 THEN 'Less than 50K'
WHEN YearlyIncome BETWEEN 50000 AND 75000 THEN '50K - 75K'
WHEN YearlyIncome BETWEEN 75000 AND 100000 THEN '75K - 100K'
WHEN YearlyIncome > 100000 THEN 'Over 100K'
ELSE 'Other'
END AS Income,
COUNT(salesordernumber) AS purchases
FROM bike_sales
GROUP BY CustomerKey,
YearlyIncome
```

Query 10: Analysis of Customers with and without Children

I created this query with the purpose of identifying customers who have children and those who do not. By retrieving their sales numbers and corresponding dates, I aim to perform a comparison analysis on their purchasing behaviors. This analysis will help uncover potential patterns or differences in sales across customer segments for the year 2012, providing valuable insights for further understanding our customer base.

```
--- Selects the substring of the Orderdatekey column, representing the month,
and aliases it as "month_key"
--- Determines whether the customer has children or not based on the
TotalChildren column and aliases it as "Has Children"
--- Calculates the count of sales orders and aliases it as "sales"
SELECT
 SUBSTRING(CAST(Orderdatekey AS char), 1, 6) AS month_key,
 CASE
    WHEN TotalChildren = 0 THEN 'No Children'
    ELSE 'Has Children'
 END AS 'Has Children',
 COUNT(salesordernumber) AS sales
--- Retrieves data from the "bike_sales" CTE
--- Filters the data to include only sales orders from the year 2012 based on
the substring of the Orderdatekey column
WHERE SUBSTRING(CAST(Orderdatekey AS char), 1, 4) = '2012'
--- Groups the results by the month_key and TotalChildren
GROUP BY
 SUBSTRING(CAST(Orderdatekey AS char), 1, 6),
 TotalChildren
```

Last Steps:

I inputted the data results into Google Sheets to create visualizations that represent the analysis carried out using the SQL queries. Additionally, I prepared the data for Tableau by creating a comprehensive table with all the required data using JOINS. This data was then used to build an interactive dashboard on Tableau, leveraging the same dataset for further insights and data exploration.

```
SELECT
OrderDateKey,
```

```sql
OrderDate,
SalesOrderNumber,
SalesAmount,
BirthDate,
YearlyIncome,
TotalChildren,
TotalProductCost,
EnglishCountryRegionName,
EnglishProductName,
EnglishProductSubcategoryName
FROM AdventureWorksDW2019.DBO.FactInternetSales T1
JOIN DBO.DimCustomer T2
ON T1.CustomerKey = T2.CustomerKey
JOIN DBO.DimGeography T3
ON T2.GEOGRAPHYKEY = T3.GeographyKey
JOIN DBO.DimProduct T4
ON T1.ProductKey = T4.ProductKey
JOIN DBO.DimProductSubcategory T5
ON T4.ProductSubcategoryKey = T5.ProductSubcategoryKey
WHERE OrderDate BETWEEN '20130101' AND '20131231'
```