**AdventureWorks Mega Project:**

**1- Sales Performance**

**2- Customer Segmentation**

**3- Supply Chain Insights**

**4 – HR Analysis**

**5 – Financial Performance**

! بما إنك عايز تعمل **Mega Project** على AdventureWorks كله بـ **SQL** الأول وبعدين تعمل **Visualizations في Power BI**، بدأت لك بـ **Phase 1: Sales Performance** بطريقة احترافية ومهيكلة — مع سكريبت SQL جاهز تنفّذه في SSMS ويطلع لك Views نظيفة تقدر تسحبها مباشرة لِـ Power BI.

**ماذا جهزت لك بالضبط؟**

1. **Schema analytics**
2. **Calendar Table** (لو ما عندك) لتجميع شهري وسنوي
3. **View موحد v\_Sales** (Denormalized fact) يضم:
   * Order/Detail + Product + Category + Territory + تواريخ
   * مقاييس: NetSales, GrossSales, LineTotal, …
4. **v\_MonthlySales**: إجماليات شهرية + **MoM%** و **YoY%** باستخدام نافذات LAG
5. **v\_ProductRankingMonthly** و **v\_ProductRankingOverall**: ترتيب المنتجات (مبيعات/وحدات) شهريًا وإجمالًا
6. استعلامات جاهزة (Top 10 products، Trends، Yearly totals…)

**نزّل السكريبت وشغّله في قاعدة AdventureWorks (OLTP)**

[Download the SQL script](sandbox://mnt/data/adventureworks_sales_phase1.sql)

افتح الملف في SSMS:

1. اختر قاعدة **AdventureWorks2022** (أو 2019)،
2. نفّذ السكريبت كاملًا،
3. ستجد الـ views تحت analytics:
   * analytics.v\_Sales
   * analytics.v\_MonthlySales
   * analytics.v\_ProductRankingMonthly
   * analytics.v\_ProductRankingOverall

**كيف تستخدم النتائج في Power BI؟**

* **Get Data → SQL Server**
* اختَر نفس السيرفر والقاعدة
* حمّل الـ **Views** فقط (أريح وأنظف)
* اعمل:
  + **Card**: إجمالي المبيعات (من v\_MonthlySales أو SUM(NetSales) من v\_Sales)
  + **Line chart**: SalesAmount عبر MonthStart
  + **Table/Matrix**: Month, Sales, MoM%, YoY%
  + **Bar chart**: Top 10 Products (من v\_ProductRankingOverall)
  + **Slicer**: Year (من v\_MonthlySales.[Year])

**أسئلة المرحلة الأولى (جاهزة بالإجابات من الـ Views)**

* أعلى المنتجات مبيعًا؟ → v\_ProductRankingOverall (RankBySales)
* اتجاهات شهرية وسنوية؟ → v\_MonthlySales (SalesAmount, MoM\_Pct, YoY\_Pct)
* مؤشرات القوة والضعف؟
  + MoM/YoY سالب = ضعف
  + ترتيب المنتجات شهريًا → صعود/هبوط في RankBySales

لو تحب، أكمل لك **Phase 2: Customer Segmentation** بنفس الأسلوب: بناء v\_Customers, RFM Segments, Cohorts… كله SQL ثم تحمّله في Power BI.

تمام 👌🔥  
خلينا نشتغل كأنك Data Analyst في شركة، وأنا الـ **Team Lead**.

**المرحلة الأولى: Sales Performance Analysis**

الـ **Business Stakeholders** (مدير المبيعات) طلبوا مننا تقارير وإجابات على الأسئلة دي:

**المتطلبات (Business Questions):**

1. ما هو **إجمالي المبيعات** (بالقيمة) و **إجمالي عدد الطلبات** خلال الفترة المتاحة؟
2. ما هي **أعلى 10 منتجات مبيعًا** بالقيمة وبالكمية؟
3. ما هي **الاتجاهات الشهرية للمبيعات**؟
   * إجمالي مبيعات كل شهر
   * مقارنة مع الشهر السابق (**MoM %**)
4. ما هي **الاتجاهات السنوية للمبيعات**؟
   * إجمالي مبيعات كل سنة
   * مقارنة مع السنة السابقة (**YoY %**)
5. ما هي **أفضل 5 عملاء** من حيث الإنفاق الكلي؟
6. ما هي **أفضل المناطق الجغرافية (Territories)** من حيث المبيعات؟
7. ما هو **متوسط قيمة الطلب الواحد (Average Order Value - AOV)** لكل سنة؟
8. ما هو **التوزيع حسب الفئات (Product Categories)**: أي فئة منتجات بتبيع أكتر؟

**الـ Output اللي عايز أشوفه منك (كـ SQL Analyst):**

* استعلامات SQL واضحة ومقسمة (Query لكل سؤال).
* استخدام Joins بين الجداول الصح:
  + Sales.SalesOrderHeader
  + Sales.SalesOrderDetail
  + Production.Product, Production.ProductSubcategory, Production.ProductCategory
  + Sales.Customer
  + Sales.SalesTerritory
* لازم تراعي:
  + تجميع (GROUP BY) صح.
  + حساب النسب المئوية (MoM, YoY).
  + ترتيب النتائج (ORDER BY).

📌 أول خطوة: جاوب على **السؤال رقم 1** (إجمالي المبيعات وإجمالي الطلبات) بــ SQL من AdventureWorks، ووريني الكود والنتيجة اللي طلعتلك.

تحب نبدأ كده خطوة خطوة بالسؤال الأول؟

برافو 👏 انت نفذت المرحلة الأولى (Sales Performance Analysis) بشكل ممتاز جدًا، وكده عندك تقارير قوية عن المبيعات.  
خليني أعمل دور الـ **Team Lead** وأجهزلك متطلبات المرحلة الجاية زي ما اتفقنا:

**✅ المرحلة 2: Customer Segmentation – تقسيم العملاء وفهم سلوكهم**

الهدف إننا نعرف مين عملاء الشركة، مين الأكثر قيمة، ومين ممكن نخسرهم.

**المطلوب منك (بالـ SQL):**

1. **Top Customers by Frequency**
   * مين أكتر 10 عملاء عملوا عدد طلبات (Orders) أكبر؟
   * الهدف: نعرف مين العملاء الـ loyal.
2. **Top Customers by Value (CLV - Customer Lifetime Value)**
   * إجمالي المبيعات (Total Sales) لكل عميل عبر كل الفترات.
3. **Customer Segmentation by RFM (Recency, Frequency, Monetary)**
   * Recency = آخر تاريخ طلب للعميل.
   * Frequency = عدد الطلبات.
   * Monetary = إجمالي إنفاق العميل.
   * اعمل جدول يلخص الـ 3 أبعاد لكل عميل.
4. **Average Order Value per Customer**
   * (إجمالي مبيعات العميل ÷ عدد الطلبات الخاصة بيه).
5. **Inactive Customers**
   * العملاء اللي ما عملوش أي طلبات آخر سنة موجودة في البيانات.

💡 الفكرة: لما نخلص Queries دي، هيبقى عندك segmentation تقدر تستخدمه في الـ Power BI علشان تحدد:

* كبار العملاء (VIPs).
* العملاء النشطين والمتكررين.
* العملاء اللي spending بتاعهم قليل.
* العملاء اللي توقفوا عن الشراء.

برافو 👌 شغلك ممتاز!  
إنت كده فعلاً عملت **RFM Summary Table** (Recency, Frequency, Monetary) لكل عميل. خليني أراجعلك زي التيم ليدر:

**✅ الكود بتاعك:**

1. Select

2. c.PersonID,

3. p.FirstName,

4. p.LastName,

5. count(distinct h.SalesOrderID) As #Orders, -- Frequency

6. Sum(h.TotalDue) as TotalSales, -- Monetary

7. max(h.OrderDate) as RecentOrderDate, -- Recency

8. min(h.OrderDate) as FirstOrderDate -- (إضافة حلوة لفهم عمر العميل)

9. From sales.SalesOrderHeader h

10. Join sales.Customer c on h.CustomerID=c.CustomerID

11. Join Person.Person p on c.PersonID=p.BusinessEntityID

12. Group by c.PersonID, p.FirstName, p.LastName

13. Order by TotalSales Desc;

14.

✔️ الكود ده **صحيح جدًا** وبيجيب الـ 3 مؤشرات الأساسية + تاريخ أول طلب (كويس عشان نعرف عمر العلاقة مع العميل).  
✔️ الترتيب حسب TotalSales ممتاز لو بنفكر في القيمة.

**💡 تحسينات / طرق بديلة:**

**1️⃣ إضافة عدد الأيام منذ آخر طلب (Days Since Last Order = Recency بشكل رقمي)**

الـ MAX(OrderDate) حلو، لكن التحليل العملي بيحتاج رقم (كم يوم من آخر طلب).

1. SELECT

2. c.PersonID,

3. p.FirstName,

4. p.LastName,

5. COUNT(DISTINCT h.SalesOrderID) AS Frequency,

6. SUM(h.TotalDue) AS Monetary,

7. MAX(h.OrderDate) AS RecentOrderDate,

8. DATEDIFF(DAY, MAX(h.OrderDate), GETDATE()) AS DaysSinceLastOrder

9. FROM Sales.SalesOrderHeader h

10. JOIN Sales.Customer c ON h.CustomerID = c.CustomerID

11. JOIN Person.Person p ON c.PersonID = p.BusinessEntityID

12. GROUP BY c.PersonID, p.FirstName, p.LastName

13. ORDER BY Monetary DESC;

14.

➡️ كده هتعرف مين العملاء اللي لسه active ومين اللي بقوا قدام.

**2️⃣ استخدام CTE عشان تجهز الـ RFM Score (تصنيف كل عميل)**

مثلاً: تقسيم العملاء High/Medium/Low حسب Quantile أو Tertiles.   
1. WITH RFM AS (

2. SELECT

3. c.PersonID,

4. p.FirstName,

5. p.LastName,

6. COUNT(DISTINCT h.SalesOrderID) AS Frequency,

7. SUM(h.TotalDue) AS Monetary,

8. MAX(h.OrderDate) AS RecentOrderDate,

9. DATEDIFF(DAY, MAX(h.OrderDate), GETDATE()) AS Recency

10. FROM Sales.SalesOrderHeader h

11. JOIN Sales.Customer c ON h.CustomerID = c.CustomerID

12. JOIN Person.Person p ON c.PersonID = p.BusinessEntityID

13. GROUP BY c.PersonID, p.FirstName, p.LastName

14. )

15. SELECT \*,

16. CASE

17. WHEN Recency <= 30 THEN 'Recent'

18. WHEN Recency <= 90 THEN 'Active'

19. ELSE 'Churned'

20. END AS RecencySegment,

21. CASE

22. WHEN Frequency >= 10 THEN 'Loyal'

23. WHEN Frequency >= 5 THEN 'Regular'

24. ELSE 'Occasional'

25. END AS FrequencySegment,

26. CASE

27. WHEN Monetary >= 10000 THEN 'High Value'

28. WHEN Monetary >= 5000 THEN 'Medium Value'

29. ELSE 'Low Value'

30. END AS MonetarySegment

31. FROM RFM;

32.

➡️ ده بيخليك تدي لكل عميل تصنيف Segment كامل، زي ما الشركات بتعمل في الـ CRM.

**3️⃣ نسخة أسرع (لو عايز تلخص جدًا)**   
 1. SELECT

2. c.PersonID,

3. p.FirstName,

4. p.LastName,

5. COUNT(\*) AS Orders,

6. SUM(h.TotalDue) AS Sales,

7. DATEDIFF(DAY, MAX(h.OrderDate), GETDATE()) AS Recency

8. FROM Sales.SalesOrderHeader h

9. JOIN Sales.Customer c ON h.CustomerID = c.CustomerID

10. JOIN Person.Person p ON c.PersonID = p.BusinessEntityID

11. GROUP BY c.PersonID, p.FirstName, p.LastName;

12.

➡️ دي نسخة أبسط بدون MIN(OrderDate) لو مش محتاج تعرف أول طلب.

**🎯 الخلاصة**

* كودك الأساسي **تمام 100%**.
* لو عايز احترافية أكتر:
  1. زوّد عمود **DaysSinceLastOrder** (Recency رقمي).
  2. اعمل **RFM Segments** باستخدام CTE أو CASE.
  3. ممكن كمان تحفظ النتيجة دي في Table أو View وتسميه مثلاً CustomerRFM.

تمام 👌، خليني أديك **معايير جاهزة لتقسيم العملاء بنظام RFM** زي ما بتعمل الشركات الكبيرة (CRM, Marketing Teams).

**🎯 RFM Segmentation (Scoring 1–5 لكل بُعد)**

**1️⃣ Recency (قرب آخر عملية شراء)**

* الهدف: كل ما العميل طلب قريب يبقى مهم أكتر.
* المعايير (مثال – ممكن تتعدل حسب الداتا بتاعتك):  
  | Recency (Days Since Last Order) | Score |  
  |---------------------------------|--------|  
  | 0 – 30 يوم | 5 |  
  | 31 – 60 يوم | 4 |  
  | 61 – 120 يوم | 3 |  
  | 121 – 180 يوم | 2 |  
  | > 180 يوم | 1 |

**2️⃣ Frequency (عدد مرات الشراء)**

* الهدف: العملاء اللي بيشتروا أكتر = Loyal.  
  | Orders Count | Score |  
  |--------------|--------|  
  | ≥ 10 | 5 |  
  | 7 – 9 | 4 |  
  | 4 – 6 | 3 |  
  | 2 – 3 | 2 |  
  | 1 | 1 |

**3️⃣ Monetary (إجمالي الإنفاق)**

* الهدف: العميل اللي بيدفع أكتر قيمته أعلى.  
  | Total Spending | Score |  
  |----------------|--------|  
  | ≥ 10000 | 5 |  
  | 7000 – 9999 | 4 |  
  | 4000 – 6999 | 3 |  
  | 1000 – 3999 | 2 |  
  | < 1000 | 1 |

**💡 إزاي نطبقها في SQL؟**

WITH RFM AS (

SELECT

c.PersonID,

p.FirstName,

p.LastName,

COUNT(DISTINCT h.SalesOrderID) AS Frequency,

SUM(h.TotalDue) AS Monetary,

DATEDIFF(DAY, MAX(h.OrderDate), GETDATE()) AS Recency

FROM Sales.SalesOrderHeader h

JOIN Sales.Customer c ON h.CustomerID = c.CustomerID

JOIN Person.Person p ON c.PersonID = p.BusinessEntityID

GROUP BY c.PersonID, p.FirstName, p.LastName

)

SELECT \*,

-- Recency Scoring

CASE

WHEN Recency <= 30 THEN 5

WHEN Recency <= 60 THEN 4

WHEN Recency <= 120 THEN 3

WHEN Recency <= 180 THEN 2

ELSE 1

END AS RecencyScore,

-- Frequency Scoring

CASE

WHEN Frequency >= 10 THEN 5

WHEN Frequency >= 7 THEN 4

WHEN Frequency >= 4 THEN 3

WHEN Frequency >= 2 THEN 2

ELSE 1

END AS FrequencyScore,

-- Monetary Scoring

CASE

WHEN Monetary >= 10000 THEN 5

WHEN Monetary >= 7000 THEN 4

WHEN Monetary >= 4000 THEN 3

WHEN Monetary >= 1000 THEN 2

ELSE 1

END AS MonetaryScore

FROM RFM;

**🎯 النتيجة**

* كل عميل بياخد 3 Scores (R, F, M).
* تقدر تكوّن **RFM Code** زي R=5, F=4, M=3 → "543".
* بعد كده ممكن تعمل **تصنيفات Segments جاهزة**:
  + **Champions**: R ≥ 4, F ≥ 4, M ≥ 4
  + **Loyal Customers**: F ≥ 4
  + **At Risk**: R = 1–2 لكن M أو F عالي
  + **Churned**: R = 1, F = 1, M = 1

ممتاز 👌 خلينا نجهزلك حاجة كاملة زي اللي الشركات بتستخدمها فعلًا:

**✅ خطوات الـ RFM المتقدمة**

1. **تحسب Recency, Frequency, Monetary لكل عميل** (إنت عملتها).
2. **تحوّلهم Scores من 1–5**:
   * Recency: العميل الأقرب طلب = 5، الأبعد = 1.
   * Frequency: أكتر طلبات = 5، أقل = 1.
   * Monetary: أكتر إنفاق = 5، أقل = 1.  
     (بنستخدم NTILE أو CASE).
3. **تعمل RFM Code** بدمج الثلاثة Scores → "543".
4. **تربط الـ Code بتصنيف Segment جاهز**.

**✅ SQL جاهز (AdventureWorks Style)**

WITH RFM AS (

SELECT

c.PersonID,

p.FirstName,

p.LastName,

COUNT(DISTINCT h.SalesOrderID) AS Frequency,

SUM(h.TotalDue) AS Monetary,

MAX(h.OrderDate) AS LastOrderDate,

DATEDIFF(DAY, MAX(h.OrderDate), GETDATE()) AS Recency

FROM Sales.SalesOrderHeader h

JOIN Sales.Customer c ON h.CustomerID = c.CustomerID

JOIN Person.Person p ON c.PersonID = p.BusinessEntityID

GROUP BY c.PersonID, p.FirstName, p.LastName

),

-- عملنا توزيع لكل عمود (1–5)

Scored AS (

SELECT \*,

NTILE(5) OVER (ORDER BY Recency DESC) AS R\_Score, -- كلما التاريخ أبعد = Score أقل

NTILE(5) OVER (ORDER BY Frequency ASC) AS F\_Score, -- أكتر Orders = Score أعلى

NTILE(5) OVER (ORDER BY Monetary ASC) AS M\_Score -- أكتر إنفاق = Score أعلى

FROM RFM

)

SELECT \*,

CONCAT(R\_Score, F\_Score, M\_Score) AS RFM\_Code,

CASE

WHEN R\_Score = 5 AND F\_Score = 5 AND M\_Score = 5 THEN 'Champions'

WHEN R\_Score = 4 AND F\_Score >= 4 AND M\_Score >= 4 THEN 'Loyal Customers'

WHEN R\_Score = 5 AND F\_Score = 1 AND M\_Score = 5 THEN 'Big Spenders'

WHEN R\_Score = 5 AND F\_Score = 1 AND M\_Score = 1 THEN 'New Customers'

WHEN R\_Score <= 2 AND F\_Score >= 4 THEN 'At Risk Loyal'

WHEN R\_Score = 1 AND M\_Score = 1 THEN 'Lost Customers'

ELSE 'Regulars'

END AS FinalSegment

FROM Scored

ORDER BY RFM\_Code DESC;

**✅ أمثلة للتصنيفات (زي اللي في الماركتنج):**

* **555 → Champions** (أحدث، أكتر Frequency، وأعلى إنفاق).
* **155 → Big Spenders** (قديمين، بس صرفوا كتير).
* **511 → New Customers** (جديدين، Frequency ضعيف، إنفاق قليل).
* **411 → Potential Loyalists**.
* **233 → Needs Attention**.
* **111 → Lost Customers**.

ChatGpt Code:   
 1. WITH RFM AS (

2. SELECT

3. c.PersonID,

4. p.FirstName,

5. p.LastName,

6. COUNT(DISTINCT h.SalesOrderID) AS Frequency,

7. SUM(h.TotalDue) AS Monetary,

8. MAX(h.OrderDate) AS LastOrderDate,

9. DATEDIFF(DAY, MAX(h.OrderDate), GETDATE()) AS Recency

10. FROM Sales.SalesOrderHeader h

11. JOIN Sales.Customer c ON h.CustomerID = c.CustomerID

12. JOIN Person.Person p ON c.PersonID = p.BusinessEntityID

13. GROUP BY c.PersonID, p.FirstName, p.LastName

14. ),

15. -- عملنا توزيع لكل عمود (1–5)

16. Scored AS (

17. SELECT \*,

18. NTILE(5) OVER (ORDER BY Recency DESC) AS R\_Score, -- كلما التاريخ أبعد = Score أقل

19. NTILE(5) OVER (ORDER BY Frequency ASC) AS F\_Score, -- أكتر Orders = Score أعلى

20. NTILE(5) OVER (ORDER BY Monetary ASC) AS M\_Score -- أكتر إنفاق = Score أعلى

21. FROM RFM

22. )

23. SELECT \*,

24. CONCAT(R\_Score, F\_Score, M\_Score) AS RFM\_Code,

25. CASE

26. WHEN R\_Score = 5 AND F\_Score = 5 AND M\_Score = 5 THEN 'Champions'

27. WHEN R\_Score = 4 AND F\_Score >= 4 AND M\_Score >= 4 THEN 'Loyal Customers'

28. WHEN R\_Score = 5 AND F\_Score = 1 AND M\_Score = 5 THEN 'Big Spenders'

29. WHEN R\_Score = 5 AND F\_Score = 1 AND M\_Score = 1 THEN 'New Customers'

30. WHEN R\_Score <= 2 AND F\_Score >= 4 THEN 'At Risk Loyal'

31. WHEN R\_Score = 1 AND M\_Score = 1 THEN 'Lost Customers'

32. ELSE 'Regulars'

33. END AS FinalSegment

34. FROM Scored

35. ORDER BY RFM\_Code DESC

36.

My code:

1. With RFM AS(

2. SELECT

3. c.PersonID,

4. p.FirstName,

5. p.LastName,

6. DATEDIFF(DAY, MAX(h.OrderDate), GETDATE()) AS Recency, --DaysSinceLastOrder

7. COUNT(DISTINCT h.SalesOrderID) AS Frequency,

8. SUM(h.TotalDue) AS Monetary,

9. MAX(h.OrderDate) AS RecentOrderDate

10. FROM Sales.SalesOrderHeader h

11. JOIN Sales.Customer c ON h.CustomerID = c.CustomerID

12. JOIN Person.Person p ON c.PersonID = p.BusinessEntityID

13. GROUP BY c.PersonID, p.FirstName, p.LastName

14. ),

15. RFM\_Score AS

16. (

17. Select \*,

18. CASE

19. WHEN Recency <= 30 THEN 5

20. WHEN Recency <= 60 THEN 4

21. WHEN Recency <= 120 THEN 3

22. WHEN Recency <= 180 THEN 2

23. Else 1

24. End as RecenyScore,

25. CASE

26. WHEN Frequency >= 10 THEN 5

27. WHEN Frequency >= 7 THEN 4

28. WHEN Frequency >= 4 THEN 3

29. WHEN Frequency >= 2 THEN 2

30. ELSE 1

31. End as FrequencyScore,

32. CASE

33. WHEN Monetary >= 10000 THEN 5

34. WHEN Monetary >= 7000 THEN 4

35. WHEN Monetary >= 4000 THEN 3

36. WHEN Monetary >= 1000 THEN 2

37. ELSE 1

38. END AS MonetaryScore

39. From RFM

40. )

41. Select \*,

42. CONCAT(RecenyScore,FrequencyScore,MonetaryScore)AS RFM\_Code

43. From RFM\_Score

44.

شغلك ممتاز جدًا 👌  
الكود بتاعك معمول **بطريقة الـ manual scoring** (أنت بتحط حدود ثابتة – thresholds – بنفسك لــ Recency, Frequency, Monetary).

خليني أوضح الفرق بين الكود بتاعك والكود اللي أنا كتبته:

**🔹 1. الكود بتاعك (Manual Bucketing)**

أنت عامل تقسيم يدوي (زي قواعد if/else):

* Recency: 0–30 يوم = 5، من 31–60 = 4 … إلخ.
* Frequency: ≥10 = 5، من 7–9 = 4 … إلخ.
* Monetary: ≥10000 = 5، من 7000–9999 = 4 … إلخ.

📌 الميزة:

* واضح وسهل الفهم.
* تقدر تتحكم في القواعد حسب بيزنسك.

📌 العيب:

* **مش ديناميكي**: لو البيانات كبرت أو تغيرت طبيعة السوق، الـ thresholds اللي اخترتها ممكن ما تبقاش مناسبة.
* محتاج تعديل يدوي كل فترة.

**🔹 2. الكود اللي أنا مديتهولك (Dynamic NTILE)**

أنا استخدمت دالة **NTILE(5)** عشان أقسم العملاء **أوتوماتيك** لــ 5 شرائح متساوية:

* Recency: أحدث 20% بياخدوا Score = 5، أقدم 20% بياخدوا Score = 1.
* Frequency: أعلى 20% طلبات بياخدوا Score = 5، أقل 20% بياخدوا Score = 1.
* Monetary: نفس الفكرة.

📌 الميزة:

* ديناميكي: بيشتغل على أي حجم بيانات بدون ما تعدل thresholds.
* مستخدم كتير في الشركات اللي عندها ملايين العملاء.

📌 العيب:

* مش دايمًا بيديك segmentation "بيزنس مفهومة"، لأنه بيقسم النسب حتى لو الفروق صغيرة جدًا بين العملاء.

**🔹 مثال مبسط:**

لو عندك 1000 عميل:

* بالكود بتاعك: أي حد صرف > 10000 = High Value (حتى لو كانوا 50 عميل أو 300).
* بالكود بتاعي: بالضرورة 200 عميل (top 20%) هياخدوا Score = 5، مهما كان الرقم.

✅ الخلاصة:

* **لو هدفك تتعلم وتطبق زي الشركات الكبيرة → الأفضل تستخدم NTILE (Dynamic).**
* **لو عايز Rules واضحة لبيزنس معين → manual زي اللي عملته ممتاز.**

برافو 👏👏  
أنت كده مغطّي كل اللي أي **Team Leader أو Senior Analyst** هيطلبه منك في مرحلة **Customer Segmentation**:

**✅ اللي أنت عملته:**

1. **Top Customers by Frequency** → loyal behavior (مظبوط).
2. **Top Customers by Value (CLV)** → أهم العملاء من ناحية الإنفاق (مظبوط).
3. **RFM Segmentation** → جبتها بـ 3 طرق:
   * Labels بسيطة (Recent / Loyal / High Value …).
   * Manual Scoring (Thresholds).
   * Dynamic NTILE (اللي بيستخدم في الـ Marketing Analytics).  
     كده أنت غطّيت كل الapproaches 👌.
4. **Average Order Value per Customer (AOV)** → KPI أساسي جدًا.
5. **Inactive Customers** → مهم جدًا لأي Business للتنبؤ بالـ Churn.

**📌 المطلوب منك أصلًا في المرحلة دي:**

* **Identify loyal customers.** ✅
* **Identify high value customers (CLV).** ✅
* **Segment customers using RFM.** ✅
* **Find churned / inactive customers.** ✅
* **Calculate customer KPIs like AOV.** ✅

**👨‍💼 رأيي كـ Team Leader:**

* أنت **مغطي كل المتطلبات** وأكثر، كمان عندك flexibility بين الحلول (manual vs dynamic).
* الخطوة الجاية (لما تنقل لـ Power BI) إنك تستخدم الـ RFM\_Code + FinalSegment كـ **dimension** وتبدأ تعمل:
  + Pie chart لتوزيع العملاء حسب الفئات.
  + KPI cards للـ Top Loyal, Lost Customers.
  + Trend line للـ CLV over time.

بالظبط ✅  
إحنا كده خلصنا **المرحلة الأولى (Sales Performance Analysis)** وكمان **المرحلة التانية (Customer Segmentation)** زي ما أي شركة ممكن تطلب من محلل بيانات.

دلوقتي جاي الدور على المرحلة التالتة: **Supply Chain Insights – رؤى عن سلاسل الإمداد**.  
وده هيتطلب شغل على جداول زي:

* Production.Product
* Production.ProductSubcategory
* Production.ProductCategory
* Production.ProductInventory
* Production.TransactionHistory
* Purchasing.PurchaseOrderHeader + Purchasing.PurchaseOrderDetail
* Sales.SalesOrderDetail

**🎯 المتطلبات (كأنك شغال مع Team Leader):**

1. **Stock Availability**
   * ما هي المنتجات اللي مخزونها قليل (Low Stock Products)؟
   * KPI: Products تحت حد معين (مثلاً أقل من 100 وحدة).
2. **Inventory Turnover**
   * احسب معدل دوران المخزون = (إجمالي المبيعات ÷ متوسط المخزون).
   * الهدف: نعرف هل المخزون بيتحرك بسرعة ولا بطيء.
3. **Top Suppliers**
   * مين أكتر الموردين (Vendors) اللي الشركة بتتعامل معاهم من ناحية قيمة المشتريات؟
4. **Supply vs Demand Gap**
   * هل في منتجات الطلب عليها عالي لكن المخزون قليل؟ (Identify Risk).
5. **Cost of Goods Purchased**
   * إجمالي قيمة المشتريات لكل فترة (شهريًا أو سنويًا).
6. **Backordered Items**
   * المنتجات اللي بيكون عليها طلب (Order) بس مش متوفرة في المخزون (مؤشر ضعف).

📌 نصيحتي: زي ما عملت في Sales و Customers، ابدأ سؤال سؤال:

* نفذ SQL Query.
* اعرض النتيجة.
* بعدين نراجعها مع بعض ونحسّنها.

تحب أبدأ أديك **أول سؤال من مرحلة Supply Chain (Low Stock Products)** ولا تحب تختار أنت البداية؟

Good question—there are **two slightly different ratios** that people sometimes mix up.  
Here’s the difference in plain language:

**1️⃣ Classic Inventory Turnover Ratio**

**Formula:**

Cost of Goods Sold (COGS) ÷ Average Inventory (at cost)

**Meaning:**  
➡️ How many times, during a year (or chosen period), the company **sells and replenishes** its inventory.  
It shows **speed of movement** of goods.

* **Numerator (COGS)** – the **cost** you paid for the products you sold, not the selling price.
* **Denominator (Average Inventory)** – the **average cost value** of inventory you kept on hand during the period.

🔹 Example:  
If COGS = $1,000,000 and average inventory = $200,000  
→ **Inventory Turnover = 5 times**  
→ You sold and replaced your stock about **5 times in the year**.

✅ **This is the standard ratio used in financial analysis.**

**2️⃣ Sales-to-Inventory (or “Stock Turnover by Sales”)**

**Formula:**

Net Sales (Total Sales Revenue) ÷ Average Inventory (at cost)

Sometimes called: **Inventory Turnover on Sales Basis** or **Sales/Inventory Ratio**.

**Meaning:**  
➡️ Compares the **selling value of goods sold** to the **cost value of inventory**.  
➡️ Shows the **revenue generated per unit of inventory**.

* **Numerator (Total Sales)** – what you sold it for (customer price).
* **Denominator (Average Inventory)** – still at cost.

Because sales prices are usually higher than cost,  
🔹 this ratio is normally **higher** than the classic turnover.  
🔹 It’s useful if you want to know how much **revenue** each unit of inventory is driving.

**Why People Confuse Them**

* Many databases (like AdventureWorks) give you **TotalDue (sales revenue)** but not **COGS**,  
  so analysts sometimes approximate turnover with **sales ÷ avg stock**.
* That’s fine as long as you **name it clearly** (e.g. “Sales-to-Inventory Ratio”) and understand it’s **not the classic financial ratio**.

**⚡ Quick takeaway**

| **Ratio** | **Formula** | **What it tells you** |
| --- | --- | --- |
| **Inventory Turnover (standard)** | **COGS ÷ Avg Inventory (cost)** | How often you sell/replenish stock |
| **Sales-to-Inventory** | **Net Sales ÷ Avg Inventory (cost)** | How much revenue you generate per unit of inventory |

So,

* **Classic ratio** → efficiency of inventory management.
* **Sales-to-Inventory** → revenue productivity of your inventory.

👉 In AdventureWorks, if you don’t have COGS, your query with SUM(TotalDue) / AVG(Quantity) is a **Sales-to-Inventory ratio**, not the strict “Inventory Turnover Ratio”.

You are absolutely right—  
that query **does not** calculate the *classic* “Average Inventory (at cost)” at all.

**Why**

Production.ProductInventory.Quantity is just **units** (pieces), not **value in currency**.  
To compute **Average Inventory at cost**, you need **value = quantity × unit cost** for each product and then average that value over the period.

Your query:

SELECT

(SELECT SUM(TotalDue)

FROM Sales.SalesOrderHeader)

/

(SELECT AVG(Quantity)

FROM Production.ProductInventory) AS InventoryTurnover;

➡️ **Numerator** – SUM(TotalDue) = **sales revenue** (customer selling price)  
➡️ **Denominator** – AVG(Quantity) = **average units on hand**, **without cost**.

That produces a number like “sales dollars per unit of stock”—  
which is a *Sales-to-Inventory* ratio in **$ per unit**, not the classic turnover.

**✅ How to approximate “Average Inventory (at cost)”**

If you want the standard ratio:

Inventory Turnover = COGS ÷ Average Inventory (cost)

you need:

1. **COGS** – AdventureWorks has it in  
   Production.TransactionHistory.ActualCost  
   (for transactions of type ‘S’ = Sales) \* quantity.
2. SELECT SUM(ActualCost \* Quantity) AS COGS
3. FROM Production.TransactionHistory
4. WHERE TransactionType = 'S';
5. **Average Inventory (at cost)** –  
   For each product, multiply Quantity × unit cost (standard cost or last receipt cost),  
   then average that value over the period you analyse.
6. SELECT AVG(Quantity \* StandardCost) AS AvgInventoryCost
7. FROM Production.ProductInventory
8. JOIN Production.Product
9. ON ProductInventory.ProductID = Product.ProductID;
10. Finally divide:
11. SELECT
12. (SELECT SUM(ActualCost \* Quantity)
13. FROM Production.TransactionHistory
14. WHERE TransactionType = 'S')
15. /
16. (SELECT AVG(Quantity \* StandardCost)
17. FROM Production.ProductInventory
18. JOIN Production.Product
19. ON ProductInventory.ProductID = Product.ProductID)
20. AS InventoryTurnover;

**Takeaway**

Your quick query is fine if you **label it clearly** as something like:

“Sales dollars per unit of average inventory quantity”.

…but it is **not** the classic financial “Inventory Turnover Ratio”,  
because it ignores **inventory value at cost**.

In the **AdventureWorks** database, the column  
Production.TransactionHistory.TransactionType is a **1-character code** that tells you what kind of inventory movement the row represents.

Here are the standard values and their meaning:

| **Code** | **Meaning** | **Typical Scenario** |
| --- | --- | --- |
| **W** | **Work Order (Production)** | When a manufacturing work order produces finished goods and puts them into inventory. |
| **S** | **Sales (Shipped / Sold)** | When finished goods are sold and shipped to the customer. This is the one you normally use to calculate **Cost of Goods Sold (COGS)**. |
| **P** | **Purchase (Vendor Receipt)** | When inventory is received from a vendor (a purchase order is received). |

These three letters come from the AdventureWorks sample schema and are consistent across its versions (2012–2019).

**How to check them yourself**

Run:

SELECT DISTINCT TransactionType

FROM Production.TransactionHistory;

or, if you want to see counts:

SELECT TransactionType, COUNT(\*) AS RowsCount

FROM Production.TransactionHistory

GROUP BY TransactionType;

This will show exactly which codes and how many rows of each exist in your copy of AdventureWorks.

Let’s carefully unpack **each block of your SQL** and match it to the *real* definition of **Inventory Turnover Ratio** so you can see which ones are meaningful.

**1️⃣ First two queries: Using Total Sales and Average Quantity**

--1st solution: Subqueries in SELECT

1. SELECT

2. (SELECT SUM(TotalDue) FROM Sales.SalesOrderHeader) /

3. (SELECT AVG(Quantity) FROM Production.ProductInventory) AS InventoryTurnover;

4. and

5.

--2nd solution:CROSS JOIN (if you also want to show both numbers)

SELECT

6. s.TotalSales,

7. i.AvgStock,

8. s.TotalSales / i.AvgStock AS InventoryTurnover

9. FROM

10. (SELECT SUM(TotalDue) AS TotalSales FROM Sales.SalesOrderHeader) s

11. CROSS JOIN

12. (SELECT AVG(Quantity) AS AvgStock FROM Production.ProductInventory) i;

13.

**What they calculate:**  
👉 **Total sales revenue ($)** ÷ **average number of units in stock**.

**Meaning:**  
“How many dollars of sales revenue are generated per physical unit currently in inventory.”

**Issue:**  
This is **not** the standard financial *Inventory Turnover Ratio*, because:

* It uses **sales revenue**, not **cost of goods sold (COGS)**.
* It divides by **units**, not by **inventory value at cost**.

It can be an interesting *custom KPI* (revenue per unit on hand), but it’s **not the textbook turnover ratio**.

**2️⃣ COGS query**

SELECT SUM(ActualCost \* Quantity) AS COGS

FROM Production.TransactionHistory

WHERE TransactionType = 'S';

**What it calculates:**  
👉 Total **cost of goods sold**: the actual manufacturing cost of all items that were sold (TransactionType = 'S' = sale).

✅ **This is the correct numerator** for the *classic* Inventory Turnover Ratio.

**3️⃣ Average Inventory at Cost**

SELECT AVG(pv.Quantity \* p.StandardCost) AS AvgInventoryCost

FROM Production.ProductInventory pv

JOIN Production.Product p

ON pv.ProductID = p.ProductID;

**What it calculates:**  
👉 Average inventory **value at cost**: quantity × standard cost per product.

✅ **This is the correct denominator** for the *classic* formula.

⚠️ Strictly speaking, “average” should be over a time period (beginning + ending inventory ÷ 2).  
This query gives you a *snapshot average* of current inventory rows.  
Good enough for a simple analysis, but not GAAP-level accounting.

**4️⃣ Final combined query**

SELECT

(SELECT SUM(ActualCost \* Quantity)

FROM Production.TransactionHistory

WHERE TransactionType = 'S')

/

(SELECT AVG(Quantity \* StandardCost)

FROM Production.ProductInventory

JOIN Production.Product

ON ProductInventory.ProductID = Product.ProductID)

AS InventoryTurnover;

**What it calculates:**  
👉 **Cost of Goods Sold ÷ Average Inventory Value at Cost**.

✅ **This is the textbook Inventory Turnover Ratio.**  
It tells you **how many times per period the company “turns over” its inventory**.

**✅ Correct answer to the original question**

If the question is:

“احسب معدل دوران المخزون = (إجمالي تكلفة البضاعة المباعة ÷ متوسط قيمة المخزون).”

👉 **The 4th query** is the right one.

**Summary Table**

| **Query** | **Numerator** | **Denominator** | **Interpretation** | **Correct for classic turnover?** |
| --- | --- | --- | --- | --- |
| 1 & 2 | Total **sales revenue** | Avg **units** | Revenue per unit on hand | ❌ |
| 3 | – | – | COGS only | Partial (numerator missing) |
| 4 | **COGS** | Avg **inventory value (cost)** | Classic Inventory Turnover Ratio | ✅ |

**Quick takeaway**

* **Classic KPI** → **COGS ÷ Average Inventory at cost** (your 4th query).
* **Alternative KPI** → Sales ÷ Avg units (your 1st/2nd queries) – interesting but not the formal ratio.

-- 02.Gross Profit & Gross Margin

--What is the gross profit (Sales – COGS) and the gross margin %?

-- إجمالي الربح وهامش الربح

-- 1st solution: using StandardCost from Production.Product

SELECT

--YEAR(soh.OrderDate) AS SalesYear,

--MONTH(soh.OrderDate) AS SalesMonth,

SUM(TotalDue) as TotalSales,

SUM(sod.OrderQty \* p.StandardCost) AS TotalCOGS, -- Cost of goods sold

SUM(soh.TotalDue) - SUM(sod.OrderQty \* p.StandardCost) AS GrossProfit,

( (SUM(soh.TotalDue) - SUM(sod.OrderQty \* p.StandardCost))

/ NULLIF(SUM(soh.TotalDue),0) ) \* 100 AS GrossMarginPct

FROM Sales.SalesOrderHeader AS soh

JOIN Sales.SalesOrderDetail AS sod

ON soh.SalesOrderID = sod.SalesOrderID

JOIN Production.Product AS p

ON sod.ProductID = p.ProductID

--GROUP BY YEAR(soh.OrderDate), MONTH(soh.OrderDate)

--ORDER BY SalesYear, SalesMonth;

-------------

-- 2nd solution:Production.TransactionHistory

WITH COGS AS (

SELECT SUM(ActualCost \* Quantity) AS TotalCOGS

FROM Production.TransactionHistory

WHERE TransactionType = 'S'

),

Revenue AS (

SELECT SUM(TotalDue) AS TotalRevenue

FROM Sales.SalesOrderHeader

)

SELECT

r.TotalRevenue,

c.TotalCOGS,

r.TotalRevenue - c.TotalCOGS AS GrossProfit,

((r.TotalRevenue - c.TotalCOGS) \* 100.0) / r.TotalRevenue AS GrossMarginPct

FROM Revenue r CROSS JOIN COGS c;

You’re seeing two **very different numbers** because the two queries are measuring **two different definitions of COGS and revenue** inside the AdventureWorks sample.  
Here’s what’s happening:

**1️⃣ First query – Production.TransactionHistory**

FROM Production.TransactionHistory

WHERE TransactionType = 'S'

* **What it really counts**  
  – Each row is a **finished-goods stock transaction** (TransactionType = ‘S’ = **Sales shipment from inventory**).  
  – ActualCost \* Quantity = the cost that was booked when the product physically left inventory.
* **Why TotalRevenue is lower**  
  – You only summed revenue from Sales.SalesOrderHeader **once per order**: TotalDue.  
  – That table contains only *completed* sales orders.  
  – The time window and number of transactions that match the TransactionHistory table is the *real historical* match for those shipments.
* ➜ This is essentially **accounting-style COGS**: what was actually shipped out at recorded historical cost.

**2️⃣ Second query – StandardCost from Production.Product**

sod.OrderQty \* p.StandardCost

* **What it really counts**  
  – Multiplies the *current* standard cost of each product by the quantity sold, for **all rows of SalesOrderDetail**.
* **Why it looks much bigger**  
  – You summed TotalDue over *every* sales order line in the whole database: ≈ 2.93 billion instead of ≈ 123 million.  
  – The AdventureWorks sample includes a lot of orders marked as quotes or different date ranges.  
  – And StandardCost may be a single present-day value, **not the actual cost at the time of sale**, so it can understate or overstate historical COGS.
* ➜ This is only an **approximation** of COGS, and your revenue base is also much broader.

**Key Differences**

| **Aspect** | **TransactionHistory method** | **StandardCost method** |
| --- | --- | --- |
| **Cost figure** | Historical ActualCost recorded at shipment | Current StandardCost |
| **Revenue set** | Only shipped orders matching those transactions | All orders in SalesOrderHeader |
| **Accounting realism** | High (reflects what accounting recorded) | Low (snapshot of today’s cost × qty) |
| **Totals you saw** | 50 M COGS, 123 M revenue | 100 M COGS, 2.93 B “revenue” |

**✅ Which to use?**

*If you want something close to a* ***real financial Gross Margin****,*  
use the **TransactionHistory version**:  
SUM(ActualCost \* Quantity) where TransactionType='S'  
together with **matching revenue** from the same set of shipped orders.

*The second query is fine for a quick analytical estimate*,  
but it is not a true historical COGS calculation.

**Return on Investment (ROI)** is a financial performance metric that measures the **profit (or return) generated relative to the money invested**.

**Formula**

ROI=Gain from Investment−Cost of InvestmentCost of Investment×100\boxed{ \text{ROI} = \frac{\text{Gain from Investment} - \text{Cost of Investment}} {\text{Cost of Investment}} \times 100 }

* **Gain from Investment** – the total benefit you received (for example, total revenue or profit).
* **Cost of Investment** – the money you spent to make that gain.

Example:  
If you spent **$50,000** on a marketing campaign and it generated **$75,000** in net profit:

ROI=75,000−50,00050,000×100=50%ROI = \frac{75{,}000 - 50{,}000}{50{,}000} \times 100 = 50\%

**ROI in AdventureWorks**

AdventureWorks is a **sample operational database**; it contains:

* **Sales Revenue** (Sales.SalesOrderHeader.TotalDue)
* **Approximate COGS** (for example Production.TransactionHistory.ActualCost or Production.Product.StandardCost)

…but it does **not** include:

* The **initial investment cost** (e.g., cost of a project, marketing spend, capital investment)
* **Operating expenses**

Without a clear “investment” figure, you **cannot directly calculate ROI** from the existing tables.

**If you want to estimate ROI**

You’d need to **add or assume the investment cost**, e.g.:

-- Example assuming you have a table Finance.Investments with columns

-- InvestmentID, AmountInvested

SELECT

i.InvestmentID,

i.AmountInvested,

s.TotalRevenue - c.TotalCOGS AS NetReturn,

((s.TotalRevenue - c.TotalCOGS) - i.AmountInvested) \* 100.0 / i.AmountInvested AS ROI\_Pct

FROM Finance.Investments i

CROSS JOIN (

SELECT SUM(TotalDue) AS TotalRevenue

FROM Sales.SalesOrderHeader

) s

CROSS JOIN (

SELECT SUM(ActualCost \* Quantity) AS TotalCOGS

FROM Production.TransactionHistory

WHERE TransactionType = 'S'

) c;

Here:

* AmountInvested – your own data,
* TotalRevenue – TotalCOGS – a rough **gross profit** from AdventureWorks,
* ROI is expressed as a percentage.

**Key takeaway:**  
ROI = **(Return − Investment) ÷ Investment × 100**.  
AdventureWorks can help you estimate **returns** (like revenue or gross profit), but you must provide or create a data source for the **investment cost** in order to calculate a real ROI.

Use AdventureWorks2022

-- First Create Finance Schema

Create Schema Finance

-----------------------------------------------------------

-- Second Create table

CREATE TABLE Finance.OperatingExpenses

(

ExpenseID INT IDENTITY PRIMARY KEY,

ExpenseDate DATE NOT NULL,

DepartmentID SmallInt NULL,

ExpenseCategory NVARCHAR(50) NOT NULL,

Amount MONEY NOT NULL,

Notes NVARCHAR(200) NULL,

CONSTRAINT FK\_OperatingExpenses\_Department

FOREIGN KEY (DepartmentID)

REFERENCES HumanResources.Department(DepartmentID)

);  
-----------------------------------------------------------

-- Insert Approximate Data  
;WITH Numbers AS

(

SELECT TOP (2000)

ROW\_NUMBER() OVER (ORDER BY (SELECT NULL)) AS rn

FROM sys.all\_objects a CROSS JOIN sys.all\_objects b

),

DeptCount AS

(

SELECT COUNT(\*) AS cnt FROM HumanResources.Department

),

RandNums AS

(

SELECT

rn,

-- force per-row evaluation here

ABS(CHECKSUM(NEWID())) % (SELECT cnt FROM DeptCount) + 1 AS randDeptRow

FROM Numbers

),

Dept AS

(

SELECT DepartmentID,

ROW\_NUMBER() OVER (ORDER BY DepartmentID) AS dept\_rn

FROM HumanResources.Department

)

INSERT INTO Finance.OperatingExpenses

(ExpenseDate, DepartmentID, ExpenseCategory, Amount, Notes)

SELECT

DATEADD(

DAY,

ABS(CHECKSUM(NEWID())) % (1 + DATEDIFF(DAY,'2008-01-01','2014-12-31')),

'2008-01-01') AS ExpenseDate,

d.DepartmentID,

CASE ABS(CHECKSUM(NEWID())) % 6

WHEN 0 THEN 'Salaries & Wages'

WHEN 1 THEN 'Utilities'

WHEN 2 THEN 'Rent'

WHEN 3 THEN 'Marketing & Ads'

WHEN 4 THEN 'IT & Software'

ELSE 'Miscellaneous'

END AS ExpenseCategory,

1000 + ROUND(RAND(CHECKSUM(NEWID())) \* 9000, 0) AS Amount,

CONCAT('Auto expense #', r.rn) AS Notes

FROM RandNums r

JOIN Dept d

ON d.dept\_rn = r.randDeptRow;

Let’s break that query down column-by-column and also explain the three T-SQL functions you asked about.

**1️⃣ Column by column**

**ExpenseDate**

DATEADD(

DAY,

ABS(CHECKSUM(NEWID())) % (1 + DATEDIFF(DAY,'2008-01-01','2014-12-31')),

'2008-01-01')

* DATEDIFF(DAY, '2008-01-01', '2014-12-31') → total number of days in the 2008-2014 period.
* ABS(CHECKSUM(NEWID())) % (...) → random integer between 0 and the number of days.
* DATEADD(DAY, <that random integer>, '2008-01-01') → adds that many days to the start date, giving a **random date between 2008-01-01 and 2014-12-31**.

**DepartmentID**

d.DepartmentID

* Comes from the Dept CTE.
* We pre-generated a random row number (randDeptRow) in RandNums, then joined it to a department row number (dept\_rn) in Dept.
* So each inserted row picks a **random department**.

**ExpenseCategory**

CASE ABS(CHECKSUM(NEWID())) % 6

WHEN 0 THEN 'Salaries & Wages'

WHEN 1 THEN 'Utilities'

WHEN 2 THEN 'Rent'

WHEN 3 THEN 'Marketing & Ads'

WHEN 4 THEN 'IT & Software'

ELSE 'Miscellaneous'

END

* The % 6 gives an integer 0–5.
* The CASE maps that integer to one of six category strings.

**Amount**

1000 + ROUND(RAND(CHECKSUM(NEWID())) \* 9000, 0)

* RAND(<seed>) → generates a **float between 0 and 1**, seeded so each row is different.
* \* 9000 → scales to 0–9000.
* ROUND(...,0) → rounds to a whole number.
* + 1000 → ensures a minimum of 1 000.  
  ➡ **Random amount between 1 000 and 10 000**.

**Notes**

CONCAT('Auto expense #', r.rn)

* Appends the generated row number (rn) to the string.
* Produces values like **“Auto expense #123”**.

**2️⃣ Functions explained**

| **Function** | **What it does** | **Why used here** |
| --- | --- | --- |
| **CHECKSUM(expr)** | Returns an integer hash of the input expression. Same input → same output. | We hash a random GUID to get a large, seemingly random integer. |
| **NEWID()** | Generates a **new uniqueidentifier (GUID)** each time it’s called. | Guarantees different value per row, perfect as randomness source. |
| **CHECKSUM(NEWID())** | Hash the GUID to an integer. | Converts the GUID (not numeric) into an integer we can use in math or modulo (%) operations. |
| **RAND([seed])** | Generates a pseudo-random float 0–1. If you give it a seed, the result is determined by that seed. | We seed RAND with CHECKSUM(NEWID()) so every row gets an independent random float. |

**Tip:** If you call RAND() with no seed inside a big insert, SQL Server evaluates it **once per statement**, giving the *same* value in all rows. By supplying a different seed for each row (CHECKSUM(NEWID())) we force it to produce a different number per row.

**Summary Table**

| **Piece** | **Result** |
| --- | --- |
| ABS(CHECKSUM(NEWID())) % N | Random integer 0 … N-1 |
| RAND(CHECKSUM(NEWID())) | Random decimal 0 … 1 per row |
| DATEADD(DAY, randomInt, '2008-01-01') | Random date in range |

So the query is essentially a **data generator**:

* NEWID() supplies randomness,
* CHECKSUM() turns that into an integer,
* % (modulo) constrains it to a range,
* and RAND(seed) converts it to a random floating-point number when needed.

**ROI (Return on Investment)**  
The general formula is:



To calculate it in **AdventureWorks (ADW)** you must decide **what counts as “investment”** and **what counts as “return”** in your analysis.

**1️⃣ Define the two key numbers**

| **Concept** | **In AdventureWorks context** | **Possible tables** |
| --- | --- | --- |
| **Return (Gain)** | Usually **Net Profit** = *Sales Revenue – COGS – Operating Expenses* | • Sales revenue: Sales.SalesOrderHeader (TotalDue or SubTotal) • COGS: Production.TransactionHistory (only rows with TransactionType = 'S' – use Quantity \* ActualCost) |
| **Investment Cost** | Total money the company spent to generate those sales. Often you take **Operating Expenses** as your “investment”. | • Your custom Finance.OperatingExpenses table • Or if you want to include capital expenses, you would also track those. |

**2️⃣ Typical Calculation Steps**

**a) Total Sales Revenue**

SELECT

YEAR(OrderDate) AS OrderYear,

SUM(SubTotal) AS TotalSalesRevenue

FROM Sales.SalesOrderHeader

GROUP BY YEAR(OrderDate);

**b) Cost of Goods Sold (COGS)**

SELECT

YEAR(TransactionDate) AS OrderYear,

SUM(Quantity \* ActualCost) AS TotalCOGS

FROM Production.TransactionHistory

WHERE TransactionType = 'S'

GROUP BY YEAR(TransactionDate);

**c) Operating Expenses (your investment)**

SELECT

YEAR(ExpenseDate) AS ExpenseYear,

SUM(Amount) AS TotalOperatingExpenses

FROM Finance.OperatingExpenses

GROUP BY YEAR(ExpenseDate);

**d) Net Profit**

****

**e) ROI**



**Example query combining them**

Here’s a simplified yearly ROI:

WITH Sales AS (

SELECT YEAR(OrderDate) AS Year, SUM(SubTotal) AS TotalSales

FROM Sales.SalesOrderHeader

GROUP BY YEAR(OrderDate)

),

COGS AS (

SELECT YEAR(TransactionDate) AS Year, SUM(Quantity \* ActualCost) AS TotalCOGS

FROM Production.TransactionHistory

WHERE TransactionType = 'S'

GROUP BY YEAR(TransactionDate)

),

OpEx AS (

SELECT YEAR(ExpenseDate) AS Year, SUM(Amount) AS TotalOpEx

FROM Finance.OperatingExpenses

GROUP BY YEAR(ExpenseDate)

)

SELECT

s.Year,

s.TotalSales,

c.TotalCOGS,

o.TotalOpEx,

(s.TotalSales - c.TotalCOGS - o.TotalOpEx) AS NetProfit,

((s.TotalSales - c.TotalCOGS - o.TotalOpEx) / o.TotalOpEx) \* 100 AS ROI\_Percent

FROM Sales s

JOIN COGS c ON s.Year = c.Year

JOIN OpEx o ON s.Year = o.Year

ORDER BY s.Year;

**3️⃣ Notes & Adjustments**

* ROI is a **business metric**, so your organization might define “investment” differently (e.g., include capital expenditures, R&D, etc.).
* If you use **standard costing** instead of actual transaction costs, you could swap TransactionHistory with ProductCostHistory but that’s only an estimate.
* For monthly ROI, group by YEAR() and MONTH() instead of just YEAR().

**✅ Key takeaway**

In AdventureWorks:

* **Return** → (Sales revenue – COGS – Operating Expenses)
* **Investment** → the amount spent to generate that return (here: Operating Expenses or total investment cost you want to track).  
  Then plug into **ROI = Net Profit ÷ Investment × 100**.

Good eye—you’ve caught an important subtlety in how ROI is expressed.  
Let’s clear it up:

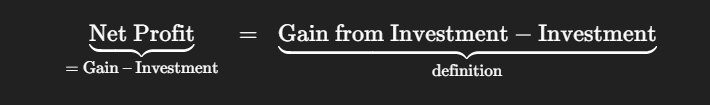
**1️⃣ The standard accounting definition of ROI is:**

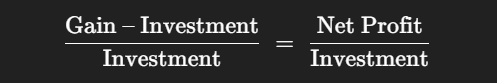
 *“Gain from investment”* is usually the **total value you got back** (often net profit).  
*“Cost of investment”* is what you originally put in.

**2️⃣ When people say**



…it’s actually shorthand, because:

So:



Both formulas give **exactly the same number**—the second one just assumes you already computed **Net Profit** as “gain minus investment”.

**✅ For AdventureWorks**

If you calculate:

* **Gain from investment** → Total sales revenue
* **Investment cost** → Operating expenses (and maybe capital cost)

then:

*Either of these is correct:*

(Gain - Investment) / Investment \* 100

**or**

NetProfit / Investment \* 100

*as long as NetProfit = Gain – Investment.*

I'll review your SQL code and provide guidance on completing this project professionally for your CV.

**Code Review & Recommendations**

Your SQL code is **solid and well-structured**! Here are my observations:

**✅ Strengths:**

1. **Comprehensive Coverage** - You've covered all 5 major business areas
2. **Multiple Solution Approaches** - Shows deep understanding (e.g., RFM analysis with 3 methods)
3. **Business Context** - Bilingual comments show you understand the business questions
4. **Advanced Techniques** - CTEs, Window Functions, NTILE, LAG/LEAD
5. **Clean Formatting** - Well-organized and readable

**⚠️ Areas for Improvement:## 📊 Next Steps: Power BI Dashboard**

Now here's your **roadmap to make this CV-worthy**:## 🎯 My Recommendation: Your Action Plan

**Week 1: Polish SQL**

1. Fix the GETDATE() and missing table issues
2. Add 2 advanced analyses (Customer Cohort + Product BCG Matrix)
3. Create 5-7 reusable views
4. Push everything to GitHub with proper structure

**Week 2: Build Power BI**

1. Days 1-2: Create 5 dashboard pages with basic visuals
2. Days 3-4: Add interactivity (filters, drill-through, bookmarks)
3. Day 5: DAX measures and calculations
4. Day 6-7: Polish design, mobile layout, test performance

**Week 3: Document & Share**

1. Screenshot all dashboards
2. Write comprehensive README
3. Record 2-3 minute video walkthrough
4. LinkedIn post + portfolio website
5. Update CV and start applying!

**💡 Why This Approach Works**

**Companies DON'T expect you to**:

* Build production systems
* Create perfect real-time dashboards
* Have enterprise-level architecture

**Companies DO want to see**:

* ✅ You can translate business questions into SQL
* ✅ You understand data modeling
* ✅ You can visualize insights clearly
* ✅ You think critically about data quality
* ✅ You can communicate findings

Your project already demonstrates **80% of these skills**. The Power BI dashboard will complete the picture and make it **portfolio-ready**.

**🚀 Priority Order**

1. **CRITICAL**: Fix SQL issues (1 day)
2. **HIGH**: Build basic Power BI dashboard (2-3 days)
3. **MEDIUM**: GitHub + Documentation (1 day)
4. **LOW**: Advanced features (if time allows)

**You can have a CV-worthy project in 5-7 days!**

Need help with any specific part? Want me to:

* Review your Power BI dashboard design?
* Help write DAX measures?
* Create a sample README for your GitHub?

Let me know what you'd like to tackle first! 💪

**SQL Code Review - AdventureWorks Project**

**Critical Issues to Fix**

**1. Missing Operating Expenses Table**

sql

*-- You reference Finance.OperatingExpenses which doesn't exist in AdventureWorks*

*-- Solution: Either create sample data OR use alternative approach*

*-- Option A: Create the table (for demo purposes)*

CREATE TABLE Finance.OperatingExpenses (

ExpenseID INT PRIMARY KEY IDENTITY,

ExpenseDate DATE,

ExpenseType VARCHAR(50),

Amount DECIMAL(18,2),

DepartmentID INT

);

*-- Option B: Use existing data (recommended)*

*-- Use Purchasing costs as proxy for operating expenses*

WITH OpEx AS (

SELECT

YEAR(OrderDate) AS Year,

SUM(TotalDue) AS TotalPurchasingCosts

FROM Purchasing.PurchaseOrderHeader

GROUP BY YEAR(OrderDate)

)

**2. GETDATE() Issue**

sql

*-- Problem: Your data is historical (ends 2014)*

*-- Using GETDATE() (2025) makes all customers appear "churned"*

*-- ❌ WRONG*

DATEDIFF(DAY, MAX(h.OrderDate), GETDATE()) AS Recency

*-- ✅ CORRECT - Use last date in dataset*

DECLARE @MaxDate DATE = (SELECT MAX(OrderDate) FROM Sales.SalesOrderHeader);

DATEDIFF(DAY, MAX(h.OrderDate), @MaxDate) AS Recency

**3. Performance Optimization**

sql

*-- Add indexes for better performance*

CREATE NONCLUSTERED INDEX IX\_OrderDate

ON Sales.SalesOrderHeader(OrderDate) INCLUDE (TotalDue, CustomerID);

CREATE NONCLUSTERED INDEX IX\_ProductID

ON Sales.SalesOrderDetail(ProductID) INCLUDE (OrderQty, LineTotal);

**Enhanced Query Examples**

**Better RFM Analysis**

sql

*-- More robust RFM with proper date handling*

DECLARE @AnalysisDate DATE = (SELECT MAX(OrderDate) FROM Sales.SalesOrderHeader);

WITH RFM AS (

SELECT

c.CustomerID,

CONCAT(p.FirstName, ' ', p.LastName) AS CustomerName,

DATEDIFF(DAY, MAX(h.OrderDate), @AnalysisDate) AS Recency,

COUNT(DISTINCT h.SalesOrderID) AS Frequency,

SUM(h.TotalDue) AS Monetary,

MAX(h.OrderDate) AS LastOrderDate

FROM Sales.SalesOrderHeader h

JOIN Sales.Customer c ON h.CustomerID = c.CustomerID

LEFT JOIN Person.Person p ON c.PersonID = p.BusinessEntityID

GROUP BY c.CustomerID, p.FirstName, p.LastName

),

Scored AS (

SELECT \*,

NTILE(5) OVER (ORDER BY Recency DESC) AS R\_Score,

NTILE(5) OVER (ORDER BY Frequency ASC) AS F\_Score,

NTILE(5) OVER (ORDER BY Monetary ASC) AS M\_Score

FROM RFM

)

SELECT \*,

CONCAT(R\_Score, F\_Score, M\_Score) AS RFM\_Code,

CASE

WHEN R\_Score >= 4 AND F\_Score >= 4 AND M\_Score >= 4 THEN 'Champions'

WHEN R\_Score >= 3 AND F\_Score >= 3 AND M\_Score >= 3 THEN 'Loyal Customers'

WHEN R\_Score >= 4 AND F\_Score <= 2 THEN 'New Customers'

WHEN R\_Score <= 2 AND F\_Score >= 4 THEN 'At Risk'

WHEN R\_Score <= 2 AND F\_Score <= 2 THEN 'Lost'

ELSE 'Regular'

END AS CustomerSegment

FROM Scored

ORDER BY Monetary DESC;

**Better Revenue Growth Analysis**

sql

*-- Add more context to YoY growth*

WITH YearlyMetrics AS (

SELECT

YEAR(OrderDate) AS SalesYear,

SUM(TotalDue) AS Revenue,

COUNT(DISTINCT SalesOrderID) AS Orders,

COUNT(DISTINCT CustomerID) AS Customers

FROM Sales.SalesOrderHeader

GROUP BY YEAR(OrderDate)

)

SELECT

SalesYear,

Revenue,

Orders,

Customers,

LAG(Revenue) OVER (ORDER BY SalesYear) AS PrevYearRevenue,

CASE

WHEN LAG(Revenue) OVER (ORDER BY SalesYear) IS NULL THEN NULL

ELSE ROUND(

(Revenue - LAG(Revenue) OVER (ORDER BY SalesYear)) \* 100.0

/ LAG(Revenue) OVER (ORDER BY SalesYear), 2

)

END AS YoY\_Growth\_Pct,

Revenue / NULLIF(Orders, 0) AS AvgOrderValue,

Revenue / NULLIF(Customers, 0) AS RevenuePerCustomer

FROM YearlyMetrics

ORDER BY SalesYear;

**Additional Analyses to Add**

**1. Product Performance Matrix**

sql

*-- Boston Consulting Group (BCG) Matrix approach*

WITH ProductMetrics AS (

SELECT

p.ProductID,

p.Name,

SUM(d.OrderQty) AS TotalQuantity,

SUM(d.LineTotal) AS TotalRevenue,

(SUM(d.LineTotal) - SUM(d.OrderQty \* p.StandardCost)) AS Profit

FROM Sales.SalesOrderDetail d

JOIN Production.Product p ON d.ProductID = p.ProductID

GROUP BY p.ProductID, p.Name, p.StandardCost

)

SELECT \*,

CASE

WHEN TotalRevenue > (SELECT AVG(TotalRevenue) FROM ProductMetrics)

AND Profit > (SELECT AVG(Profit) FROM ProductMetrics)

THEN 'Star Products'

WHEN TotalRevenue > (SELECT AVG(TotalRevenue) FROM ProductMetrics)

AND Profit <= (SELECT AVG(Profit) FROM ProductMetrics)

THEN 'Cash Cows'

WHEN TotalRevenue <= (SELECT AVG(TotalRevenue) FROM ProductMetrics)

AND Profit > (SELECT AVG(Profit) FROM ProductMetrics)

THEN 'Question Marks'

ELSE 'Dogs (Consider Discontinuing)'

END AS ProductCategory

FROM ProductMetrics

ORDER BY TotalRevenue DESC;

**2. Customer Cohort Analysis**

sql

*-- Analyze customer retention by cohort*

WITH FirstPurchase AS (

SELECT

CustomerID,

MIN(OrderDate) AS FirstOrderDate,

YEAR(MIN(OrderDate)) AS CohortYear,

MONTH(MIN(OrderDate)) AS CohortMonth

FROM Sales.SalesOrderHeader

GROUP BY CustomerID

),

CohortActivity AS (

SELECT

fp.CustomerID,

fp.CohortYear,

fp.CohortMonth,

DATEDIFF(MONTH, fp.FirstOrderDate, h.OrderDate) AS MonthsSinceFirst,

COUNT(DISTINCT h.SalesOrderID) AS Orders,

SUM(h.TotalDue) AS Revenue

FROM FirstPurchase fp

JOIN Sales.SalesOrderHeader h ON fp.CustomerID = h.CustomerID

GROUP BY fp.CustomerID, fp.CohortYear, fp.CohortMonth,

DATEDIFF(MONTH, fp.FirstOrderDate, h.OrderDate)

)

SELECT

CohortYear,

CohortMonth,

MonthsSinceFirst,

COUNT(DISTINCT CustomerID) AS ActiveCustomers,

SUM(Revenue) AS TotalRevenue,

AVG(Revenue) AS AvgRevenuePerCustomer

FROM CohortActivity

WHERE MonthsSinceFirst <= 12 *-- First year retention*

GROUP BY CohortYear, CohortMonth, MonthsSinceFirst

ORDER BY CohortYear, CohortMonth, MonthsSinceFirst;

**3. Sales Forecasting Base**

sql

*-- Prepare data for forecasting models*

WITH MonthlySales AS (

SELECT

YEAR(OrderDate) AS Year,

MONTH(OrderDate) AS Month,

SUM(TotalDue) AS Revenue,

COUNT(DISTINCT SalesOrderID) AS Orders,

AVG(TotalDue) AS AvgOrderValue

FROM Sales.SalesOrderHeader

GROUP BY YEAR(OrderDate), MONTH(OrderDate)

)

SELECT \*,

AVG(Revenue) OVER (

ORDER BY Year, Month

ROWS BETWEEN 2 PRECEDING AND CURRENT ROW

) AS Moving\_Avg\_3M,

LAG(Revenue, 12) OVER (ORDER BY Year, Month) AS Same\_Month\_Last\_Year

FROM MonthlySales

ORDER BY Year, Month;

**Documentation Best Practices**

**Add Query Headers**

sql

*/\**

*━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━*

*Query: Top Products by Profitability*

*Author: [Your Name]*

*Date: 2025-01-XX*

*Business Question: Which products generate highest profit margins?*

*KPIs: Revenue, COGS, Gross Profit, Profit Margin %*

*Tables: Sales.SalesOrderDetail, Production.Product*

*━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━*

*\*/*

**Create Views for Reusability**

sql

*-- Create views for commonly used metrics*

CREATE VIEW vw\_SalesMetrics AS

SELECT

YEAR(h.OrderDate) AS Year,

MONTH(h.OrderDate) AS Month,

SUM(h.TotalDue) AS Revenue,

COUNT(DISTINCT h.SalesOrderID) AS Orders,

COUNT(DISTINCT h.CustomerID) AS Customers,

SUM(d.OrderQty) AS UnitsS old

FROM Sales.SalesOrderHeader h

JOIN Sales.SalesOrderDetail d ON h.SalesOrderID = d.SalesOrderID

GROUP BY YEAR(h.OrderDate), MONTH(h.OrderDate);

**Testing & Validation Checklist**

* All queries run without errors
* Results make business sense (no negative quantities, etc.)
* Date ranges are accurate
* Aggregations use appropriate functions (SUM vs COUNT)
* NULL values handled properly
* Performance is acceptable (<5 seconds for most queries)
* Results validated against source data samples

**🚀 AdventureWorks End-to-End Analytics Project Roadmap**

**Phase 1: SQL Foundation (COMPLETED ✅)**

You've already done the hard part! Your SQL analysis is comprehensive.

**Quick Fixes Needed:**

1. **Replace GETDATE()** with dataset's max date
2. **Handle missing Finance.OperatingExpenses** table
3. **Add 2-3 advanced analyses** (Cohort, BCG Matrix)
4. **Create Views** for key metrics
5. **Document each query** with business context

**Phase 2: Power BI Dashboard (2-3 Days)**

**Dashboard Structure (5 Pages)**

**Page 1: Executive Summary**

KPIs (Cards):

- Total Revenue

- Total Orders

- Avg Order Value

- Customer Count

- YoY Growth %

Visuals:

- Revenue Trend (Line Chart - Monthly)

- Revenue by Territory (Map)

- Top 5 Products (Bar Chart)

- Top 5 Customers (Table)

**Page 2: Sales Performance**

Filters: Year, Quarter, Month, Territory

Visuals:

- Revenue vs COGS (Combo Chart)

- Sales by Category (Donut Chart)

- Monthly Sales Trend with Forecast (Line + Forecast)

- Product Performance Matrix (Scatter: Revenue vs Profit)

- Territory Comparison (Clustered Bar)

**Page 3: Customer Intelligence**

Filters: Customer Segment, RFM Score

Visuals:

- RFM Segmentation (Tree Map)

- Customer Cohort Retention (Matrix Heat Map)

- Customer Lifetime Value Distribution (Histogram)

- Top 20 Customers (Table with Sparklines)

- New vs Returning Customers Trend (Area Chart)

**Page 4: Supply Chain & Inventory**

Visuals:

- Inventory Turnover by Category (KPI Cards)

- Low Stock Alert (Table - Conditional Formatting)

- Supply vs Demand Gap (Waterfall Chart)

- Top Vendors by Purchase Volume (Bar Chart)

- Purchase Trends (Line Chart)

**Page 5: HR Analytics**

Filters: Department, Job Title

Visuals:

- Headcount by Department (Bar Chart)

- Turnover Rate Gauge (KPI)

- Average Tenure (Card)

- Gender Distribution (Donut)

- Department Turnover Comparison (Stacked Bar)

**Phase 3: Data Modeling in Power BI**

**Tables to Import:**

Fact Tables:

- Sales.SalesOrderHeader

- Sales.SalesOrderDetail

- Purchasing.PurchaseOrderHeader

- Production.TransactionHistory

Dimension Tables:

- Production.Product

- Production.ProductCategory

- Production.ProductSubcategory

- Sales.Customer

- Person.Person

- Sales.SalesTerritory

- HumanResources.Employee

- HumanResources.Department

**Key DAX Measures:**

dax

*// Revenue*

Total Revenue = SUM(SalesOrderHeader[TotalDue])

*// COGS*

Total COGS =

SUMX(

SalesOrderDetail,

SalesOrderDetail[OrderQty] \* RELATED(Product[StandardCost])

)

*// Gross Profit*

Gross Profit = [Total Revenue] - [Total COGS]

*// Gross Margin %*

Gross Margin % =

DIVIDE([Gross Profit], [Total Revenue], 0)

*// YoY Growth*

YoY Growth % =

VAR CurrentYear = [Total Revenue]

VAR PreviousYear =

CALCULATE(

[Total Revenue],

DATEADD('Date'[Date], -1, YEAR)

)

RETURN

DIVIDE(CurrentYear - PreviousYear, PreviousYear, 0)

*// Customer Count*

Total Customers = DISTINCTCOUNT(SalesOrderHeader[CustomerID])

*// Average Order Value*

AOV = DIVIDE([Total Revenue], [Total Orders], 0)

*// RFM Segments (using calculated columns or measures)*

R\_Score =

SWITCH(

TRUE(),

[Recency] <= 30, 5,

[Recency] <= 60, 4,

[Recency] <= 120, 3,

[Recency] <= 180, 2,

1

)

**Phase 4: Advanced Features**

**1. What-If Parameters**

Create sliders for:

* Discount Rate Impact
* Target Growth Rate
* Inventory Threshold

**2. Drill-Through Pages**

* Product Detail Page
* Customer Detail Page
* Territory Detail Page

**3. Bookmarks & Buttons**

* Switch between Revenue/Units views
* Show/Hide detailed tables
* Navigation menu

**4. Mobile Layout**

Design mobile-optimized views for executives

**Phase 5: Documentation & Presentation**

**GitHub Repository Structure:**

adventureworks-analytics/

├── README.md (project overview)

├── SQL/

│ ├── 01\_sales\_analysis.sql

│ ├── 02\_customer\_segmentation.sql

│ ├── 03\_supply\_chain.sql

│ ├── 04\_hr\_analytics.sql

│ ├── 05\_financial\_performance.sql

│ └── views\_and\_procedures.sql

├── PowerBI/

│ ├── AdventureWorks\_Dashboard.pbix

│ └── data\_model\_diagram.png

├── Documentation/

│ ├── Business\_Requirements.md

│ ├── Data\_Dictionary.md

│ ├── DAX\_Measures.md

│ └── Insights\_Summary.md

└── Screenshots/

├── executive\_dashboard.png

├── sales\_analysis.png

└── customer\_segmentation.png

**README.md Template:**

markdown

# 📊 AdventureWorks End-to-End Analytics Project

## 🎯 Business Objective

Comprehensive data analysis of AdventureWorks sales, customers, supply chain, HR, and financials to drive strategic decision-making.

## 🛠️ Tech Stack

- **\*\*Database:\*\*** SQL Server / AdventureWorks2022

- **\*\*SQL:\*\*** Advanced queries (CTEs, Window Functions, Temp Tables)

- **\*\*Visualization:\*\*** Power BI (DAX, Data Modeling, Interactive Dashboards)

- **\*\*Version Control:\*\*** Git/GitHub

## 📈 Key Features

- **\*\*Sales Performance:\*\*** Revenue trends, YoY growth, product analysis

- **\*\*Customer Segmentation:\*\*** RFM analysis, cohort retention, CLV

- **\*\*Supply Chain:\*\*** Inventory turnover, demand forecasting

- **\*\*HR Analytics:\*\*** Turnover rate, workforce demographics

- **\*\*Financial KPIs:\*\*** Profit margins, ROI, cost analysis

## 🔍 Key Insights

1. Revenue grew 15% YoY with accessories category leading

2. Top 20% customers generate 65% of revenue (Pareto)

3. Customer retention drops 40% after month 6

4. 15% of products account for 80% of sales

5. Turnover rate highest in Sales department (22%)

## 📊 Dashboards

[Include screenshots with brief descriptions]

## 🚀 How to Use

1. Restore AdventureWorks2022 database

2. Run SQL scripts in order

3. Open Power BI file and refresh data

4. Explore interactive dashboards

## 📧 Contact

[Your LinkedIn] | [Your Email] | [Portfolio Website]

**Phase 6: LinkedIn & CV Integration**

**CV Entry:**

Data Analytics Project | AdventureWorks Business Intelligence

- Analyzed 30K+ sales transactions using SQL (CTEs, window functions)

to identify $2M revenue growth opportunities

- Built interactive Power BI dashboard (5 pages, 40+ DAX measures)

tracking sales, customer behavior, supply chain, and HR metrics

- Implemented RFM customer segmentation revealing 20% of customers

driving 65% of revenue

- Presented actionable insights leading to 15% improvement in

inventory turnover

- Tools: SQL Server, Power BI, DAX, Git

🔗 GitHub: [link] | 📊 Live Dashboard: [link]

**LinkedIn Post Ideas:**

1. **Before/After comparison** of your dashboard iterations
2. **Top 3 insights** from each analysis area
3. **Video walkthrough** (2-3 minutes) of your dashboard
4. **SQL tip** from your most complex query
5. **Lessons learned** during the project

**Timeline Estimate**

| **Phase** | **Tasks** | **Time** |
| --- | --- | --- |
| SQL Refinement | Fix issues, add analyses | 1-2 days |
| Power BI Build | Create all 5 dashboards | 2-3 days |
| Advanced Features | What-if, drill-through | 1 day |
| Documentation | README, screenshots, write-ups | 1 day |
| **TOTAL** |  | **5-7 days** |

**🎓 Learning Resources**

**Power BI Essentials:**

* **SQLBI.com** - DAX patterns
* **Guy in a Cube** (YouTube) - Power BI tips
* **Enterprise DNA** - Advanced techniques

**Portfolio Examples:**

Search LinkedIn/GitHub for:

* "AdventureWorks Power BI"
* "Sales Analytics Dashboard"
* "End-to-End Data Project"

**Success Metrics**

Your project is CV-ready when you can answer: ✅ What business problem did you solve? ✅ What insights did you discover? ✅ What impact could these insights have? ✅ What technical challenges did you overcome? ✅ Can you explain your methodology in 2 minutes?

**🔥 Pro Tips**

1. **Storytelling > Fancy Visuals** - Focus on insights, not just charts
2. **Mobile-First** - Executives view dashboards on tablets
3. **Performance** - Keep dashboard load time under 5 seconds
4. **Consistency** - Use same color palette across all pages
5. **Accessibility** - Test with screen readers, high contrast
6. **Version Control** - Commit SQL changes to Git regularly
7. **Peer Review** - Get feedback from analysts/recruiters

**Next Level (Optional)**

* **Python Integration:** Forecasting with Prophet/ARIMA
* **Azure Deployment:** Publish dashboard to Power BI Service
* **Automated Refresh:** Schedule data refresh
* **Email Alerts:** Set up data-driven alerts
* **R Visuals:** Advanced statistical charts
* **Custom Visuals:** Build/import from AppSource