

Dataset Name: **Northwind Traders Database**

Kaggle Link: <https://www.kaggle.com/datasets/jealousleopard/northwind>

Why this dataset?

- Contains 8 related tables (perfect for joins)
- Medium-sized dataset (realistic for practice)
- Classic relational database structure
- Includes customers, orders, products, employees, etc.

Dataset Tables: Customers - Customer information

- Orders - Order headers
- OrderDetails - Order line items
- Products - Product information
- Employees - Employee data
- Categories - Product categories
- Shippers - Shipping companies

Initialize Spark Session and Load Data

```
1 from pyspark.sql import SparkSession
2 from pyspark.sql.functions import *
3 from pyspark.sql.window import Window
4
5 spark = SparkSession.builder \
6     .appName("NorthwindAnalysis") \
7     .getOrCreate()

1 # Load all tables
2 customers = spark.read.csv("customers.csv", header=True, inferSchema=True)
3 orders = spark.read.csv("orders.csv", header=True, inferSchema=True)
4 order_details = spark.read.csv("order_details.csv", header=True, inferSchema=True)
5 products = spark.read.csv("products.csv", header=True, inferSchema=True)
6 employees = spark.read.csv("employees.csv", header=True, inferSchema=True)
7 categories = spark.read.csv("categories.csv", header=True, inferSchema=True)
8 shippers = spark.read.csv("shippers.csv", header=True, inferSchema=True)
9 suppliers = spark.read.csv("suppliers.csv", header=True, inferSchema=True)
```

Transformation Examples with Northwind Data

Basic Transformations

```
1 # Filter - German customers
2 german_customers = customers.filter(col("Country") == "Germany")
3
4 german_customers.show(5)
```

```

+-----+-----+-----+-----+-----+
|customerID|      companyName|  contactName|  contactTitle|      city|country|
+-----+-----+-----+-----+-----+
|ALFKI| Alfreds Futterkiste| Maria Anders| Sales Representative| Berlin|Germany|
|BLAUS| Blauer See Delika...| Hanna Moos| Sales Representative| Mannheim|Germany|
|DRACD| Drachenblut Delik...| Sven Ottlieb| Order Administrator| Aachen|Germany|
|FRANK| Frankenversand| Peter Franken| Marketing Manager| München|Germany|
|KOENE| Königlich Essen| Philip Cramer| Sales Associate| Brandenburg|Germany|
+-----+-----+-----+-----+-----+
only showing top 5 rows
```

```
1 # Select - Specific columns
2 customer_contact = customers.select("CustomerID", "CompanyName", "ContactName")
3
4 customer_contact.show(5)
```

```

+-----+-----+-----+
|CustomerID|      CompanyName|      ContactName|
+-----+-----+-----+
|ALFKI| Alfreds Futterkiste| Maria Anders|
|ANATR| Ana Trujillo Empa...| Ana Trujillo|
|ANTON| Antonio Moreno Ta...| Antonio Moreno|
|AROUT| Around the Horn| Thomas Hardy|
+-----+-----+-----+
```

```
| BERGS| Berglunds snabbk...p|Christina Berglund|
+-----+-----+-----+
only showing top 5 rows
```

```
1 # WithColumn - Calculate order age
2 from pyspark.sql.functions import datediff, current_date
3 orders_with_age = orders.withColumn(
4     "OrderAgeDays",
5     datediff(current_date(), col("OrderDate"))
6 )
7
8 orders_with_age.show(5)
```

```
↗ +-----+-----+-----+-----+-----+-----+-----+-----+
|orderID|customerID|employeeID| orderDate|requiredDate|shippedDate|shipperID|freight|OrderAgeDays|
+-----+-----+-----+-----+-----+-----+-----+-----+
| 10248| VINET| 5|2013-07-04| 2013-08-01| 2013-07-16| 3| 32.38| 4415|
| 10249| TOMSP| 6|2013-07-05| 2013-08-16| 2013-07-10| 1| 11.61| 4414|
| 10250| HANAR| 4|2013-07-08| 2013-08-05| 2013-07-12| 2| 65.83| 4411|
| 10251| VICTE| 3|2013-07-08| 2013-08-05| 2013-07-15| 1| 41.34| 4411|
| 10252| SUPRD| 4|2013-07-09| 2013-08-06| 2013-07-11| 2| 51.3| 4410|
+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows
```

```
1 # Drop - Remove unnecessary columns
2 products_clean = products.drop("QuantityPerUnit")
3 products_clean.show(5)
```

```
↗ +-----+-----+-----+-----+-----+
|productID| productname|unitPrice|discontinued|categoryID|
+-----+-----+-----+-----+-----+
| 1| Chai| 18.0| 0| 1|
| 2| Chang| 19.0| 0| 1|
| 3| Aniseed Syrup| 10.0| 0| 2|
| 4|Chef Anton's Cajun...| 22.0| 0| 2|
| 5|Chef Anton's Gumb...| 21.35| 1| 2|
+-----+-----+-----+-----+-----+
only showing top 5 rows
```

1 Start coding on [generate](#) with AI.

Aggregations

```
1 # Product sales aggregation
2 product_sales = order_details.groupBy("ProductID") \
3     .agg(
4         sum("Quantity").alias("TotalUnitsSold"),
5         sum(col("Quantity") * col("UnitPrice")).alias("TotalRevenue"),
6         avg("UnitPrice").alias("AvgUnitPrice")
7     )
8
9 product_sales.show(5)
```

```
↗ +-----+-----+-----+-----+
|ProductID|TotalUnitsSold| TotalRevenue| AvgUnitPrice|
+-----+-----+-----+-----+
| 31| 1397| 16172.5| 11.666666666666666|
| 65| 745| 14606.999999999998| 19.456249999999997|
| 53| 722| 21510.2| 30.159999999999982|
| 34| 506| 6678.0| 12.968421052631578|
| 28| 640| 26865.6| 41.975757575757555|
+-----+-----+-----+-----+
only showing top 5 rows
```

```
1 # Employee order count
2 employee_performance = orders.groupBy("EmployeeID") \
3     .agg(
4         count("OrderID").alias("OrderCount"),
5         min("OrderDate").alias("FirstOrderDate"),
6         max("OrderDate").alias("LastOrderDate")
7     )
8
9 employee_performance.show(5)
```

```
↗ +-----+-----+-----+-----+
|EmployeeID|OrderCount|FirstOrderDate|LastOrderDate|
+-----+-----+-----+-----+
| 1| 123| 2013-07-17| 2015-05-06|
```

6	67	2013-07-05	2015-04-23
3	127	2013-07-08	2015-04-30
5	42	2013-07-04	2015-04-22
9	43	2013-07-12	2015-04-29

only showing top 5 rows

Join Operations

```
1 # Complete order information
2 complete_orders = orders.join(order_details, "OrderID") \
3     .join(products.withColumnRenamed("unitPrice", "product_unitPrice"), "ProductID") \
4     .join(customers.withColumnRenamed("country", "customer_country"), "CustomerID") \
5     .join(employees.withColumnRenamed("city", "employee_city").withColumnRenamed("country", "employee_country"), '
6
7 complete_orders.show(5)
```

employeeID	customerID	productID	orderID	orderDate	requiredDate	shippedDate	shipperID	freight	unitPrice	quantity	discount
5	VINET	72	10248	2013-07-04	2013-08-01	2013-07-16	3	32.38	34.8	5	0.0
5	VINET	42	10248	2013-07-04	2013-08-01	2013-07-16	3	32.38	9.8	10	0.0
5	VINET	11	10248	2013-07-04	2013-08-01	2013-07-16	3	32.38	14.0	12	0.0
6	TOMSP	51	10249	2013-07-05	2013-08-16	2013-07-10	1	11.61	42.4	40	0.0
6	TOMSP	14	10249	2013-07-05	2013-08-16	2013-07-10	1	11.61	18.6	9	0.0

only showing top 5 rows

```
1 # Products with category names
2 products_with_categories = products.join(categories, "CategoryID")
3
4 products_with_categories.show(5)
```

categoryID	productID	productName	quantityPerUnit	unitPrice	discontinued	categoryName	description
1	1	Chai	10 boxes x 20 bags	18.0	0	Beverages	Soft drinks, coff...
1	2	Chang	24 - 12 oz bottles	19.0	0	Beverages	Soft drinks, coff...
2	3	Aniseed Syrup	12 - 550 ml bottles	10.0	0	Condiments	Sweet and savory ...
2	4	Chef Anton's Cajun...	48 - 6 oz jars	22.0	0	Condiments	Sweet and savory ...
2	5	Chef Anton's Gumb...	36 boxes	21.35	1	Condiments	Sweet and savory ...

only showing top 5 rows

```
1 # Left join to find unsold products
2 unsold_products = products.join(
3     order_details,
4     "ProductID",
5     "left"
6 ).filter(col("OrderID").isNull())
7
8
9 unsold_products.show(5)
```

productID	productName	quantityPerUnit	unitPrice	discontinued	categoryID	orderID	unitPrice	quantity	discount
-----------	-------------	-----------------	-----------	--------------	------------	---------	-----------	----------	----------

1 Start coding or [generate](#) with AI.

Window Functions

```
1 # Customer order ranking
2 customer_window = Window.partitionBy("CustomerID").orderBy(col("OrderDate").desc())
3 customer_orders_ranked = orders.withColumn(
4     "OrderRank",
5     rank().over(customer_window)
6 )
7
8 customer_orders_ranked.show(5)
```

orderID	customerID	employeeID	orderDate	requiredDate	shippedDate	shipperID	freight	OrderRank
11011	ALFKI	3	2015-04-09	2015-05-07	2015-04-13	1	1.21	1

10952	ALFKI	1	2015-03-16	2015-04-27	2015-03-24	1	40.42	2
10835	ALFKI	1	2015-01-15	2015-02-12	2015-01-21	3	69.53	3
10702	ALFKI	4	2014-10-13	2014-11-24	2014-10-21	1	23.94	4
10692	ALFKI	4	2014-10-03	2014-10-31	2014-10-13	2	61.02	5

only showing top 5 rows

```

1 # Monthly sales growth
2 monthly_sales = orders.join(order_details, "OrderID") \
3     .groupBy(month("OrderDate").alias("Month")) \
4     .agg(sum(col("Quantity") * col("UnitPrice")).alias("MonthlySales"))
5
6 sales_window = Window.orderBy("Month")
7 monthly_growth = monthly_sales.withColumn(
8     "PrevMonthSales",
9     lag("MonthlySales").over(sales_window)
10 ).withColumn(
11     "GrowthPct",
12     (col("MonthlySales") - col("PrevMonthSales")) / col("PrevMonthSales") * 100
13 )
14
15 monthly_growth.show(5)

```

Month	MonthlySales	PrevMonthSales	GrowthPct
1	167547.52	NULL	NULL
2	145769.15000000002	167547.52	-12.998324296295143
3	149805.35	145769.15000000002	2.7688986318435567
4	190329.95	149805.35	27.051503834809644
5	76722.36	190329.95	-59.689812349554025

only showing top 5 rows

✓ Action Examples

```

1 # Show results
2 complete_orders.show(5)
3
4 # Count records
5 print(f"Total customers: {customers.count()}")
6 print(f"Total orders: {orders.count()}")
7
8 # Collect top products to driver
9 top_products = product_sales.orderBy(col("TotalRevenue").desc()).take(5)
10
11 # Save results
12 complete_orders.write.mode("overwrite").parquet("output/complete_orders.parquet")
13 product_sales.write.mode("overwrite").csv("output/product_sales.csv", header=True)

```

employeeID	customerID	productID	orderID	orderDate	requiredDate	shippedDate	shipperID	freight	unitPrice	quantity	discount
5	VINET	72	10248	2013-07-04	2013-08-01	2013-07-16	3	32.38	34.8	5	0.0
5	VINET	42	10248	2013-07-04	2013-08-01	2013-07-16	3	32.38	9.8	10	0.0
5	VINET	11	10248	2013-07-04	2013-08-01	2013-07-16	3	32.38	14.0	12	0.0
6	TOMSP	51	10249	2013-07-05	2013-08-16	2013-07-10	1	11.61	42.4	40	0.0
6	TOMSP	14	10249	2013-07-05	2013-08-16	2013-07-10	1	11.61	18.6	9	0.0

only showing top 5 rows

Total customers: 91
Total orders: 830

✓ Complex Business Queries

Top 5 Customers by Revenue

```

1 top_customers = complete_orders.groupBy(
2     "CustomerID", "CompanyName"
3 ).agg(
4     sum(col("Quantity") * col("UnitPrice")).alias("TotalSpent"),
5     countDistinct("OrderID").alias("OrderCount")
6 ).orderBy(
7     col("TotalSpent").desc()

```

```
8 ).limit(5)
```

CustomerID	CompanyName	TotalSpent	OrderCount
QUICK	QUICK-Stop	117483.39	28
SAVEA	Save-a-lot Markets	115673.39	31
ERNSH	Ernst Handel	113236.68	30
HUNGO	Hungry Owl All-Ni...	57317.390000000014	19
RATTC	Rattlesnake Canyo...	52245.899999999994	18

Employee Sales Performance

```
1 employee_sales = complete_orders.groupBy(
2     "EmployeeID",
3     "employeeName"
4 ).agg(
5     sum(col("Quantity") * col("UnitPrice")).alias("TotalSales"),
6     countDistinct("OrderID").alias("OrderCount"),
7     avg(col("Quantity") * col("UnitPrice")).alias("AvgOrderValue")
8 ).orderBy(
9     col("TotalSales").desc()
10 )
11 employee_sales.show()
```

EmployeeID	employeeName	TotalSales	OrderCount	AvgOrderValue
4	Margaret Peacock	250187.44999999998	156	595.6844047619047
3	Janet Leverling	213051.3	127	663.711214953271
1	Nancy Davolio	202143.71000000002	123	585.9237971014493
2	Andrew Fuller	177749.26	96	737.548796680498
7	Robert King	141295.99000000002	72	802.8181250000001
8	Laura Callahan	133301.03	104	512.6962692307692
9	Anne Dodsworth	82963.99999999999	43	775.3644859813082
6	Michael Suyama	78198.09999999999	67	465.4648809523809
5	Steven Buchanan	75567.75	42	645.8782051282051

Product Category Analysis

```
1 category_performance = products_with_categories.join(
2     order_details.withColumnRenamed("unitPrice", "order_unitPrice"), "ProductID"
3 ).groupBy(
4     "CategoryID", "CategoryName"
5 ).agg(
6     sum(col("Quantity") * col("order_unitPrice")).alias("CategoryRevenue"),
7     avg(col("order_unitPrice")).alias("AvgProductPrice"),
8     countDistinct("ProductID").alias("ProductCount")
9 )
10 category_performance.show()
```

CategoryID	CategoryName	CategoryRevenue	AvgProductPrice	ProductCount
7	Produce	105268.59999999998	35.19448529411767	5
4	Dairy Products	251330.5	26.983060109289614	10
8	Seafood	141623.09	19.0629696969697	12
1	Beverages	286526.94999999995	29.236757425742578	12
5	Grains & Cereals	100726.8	21.24642857142858	7
3	Confections	177099.1	22.60269461077844	13
6	Meat & Poultry	178188.80000000002	42.874739884393065	6
2	Condiments	113694.75	21.320833333333333	12

1 Start coding or [generate](#) with AI.

This notebook demonstrates various PySpark transformations and actions applied to the Northwind Traders database. The goal is to showcase common data manipulation techniques using Spark DataFrames.

The dataset tables used include:

- Customers
- Orders

- OrderDetails
- Products
- Employees
- Categories
- Shippers

✓ PySpark Transformations

Transformations are lazy operations that define the data manipulation logic. They do not execute immediately but build a plan that is executed when an action is called.

Filtering Data

Filtering selects rows based on a condition. Below is an example of filtering the `customers` DataFrame to get only customers from Germany.

Selecting Columns

- `Selecting` chooses specific columns from a DataFrame. Here, we select the `CustomerID`, `CompanyName`, and `ContactName` from the `customers` DataFrame.

Adding New Columns (`withColumn`)

- `withColumn` is used to add a new column to a DataFrame or replace an existing one. In this example, we calculate the age of each order in days.

Dropping Columns (`drop`)

- The `drop` transformation removes specified columns from a DataFrame. Here, we remove the `QuantityPerUnit` column from the `products` DataFrame.
-

✓ Aggregations (`groupBy` and `agg`)

Aggregations are used to group data by one or more columns and then perform aggregate functions (like sum, count, average, min, max) on other columns.

Below is an example of calculating total units sold, total revenue, and average unit price for each product.

Here, we aggregate the `orders` DataFrame to find the order count, first order date, and last order date for each employee.

✓ Join Operations

Joins combine data from two or more DataFrames based on related columns. Different types of joins exist, such as inner, outer, left, and right joins.

This example demonstrates joining multiple tables (`orders`, `order_details`, `products`, `customers`, `employees`) to create a comprehensive view of orders.

This code joins `products` with `categories` to add category names to the product information.

This example uses a left join to identify products that have not been sold by checking for null `OrderID` values in the joined result.

✓ Window Functions

Window functions perform calculations across a set of DataFrame rows that are related to the current row. They are used for tasks like ranking, calculating moving averages, and accessing previous or subsequent rows.

This example uses a window function to rank orders for each customer based on the order date.

This code calculates the monthly sales and then uses a window function (`lag`) to determine the previous month's sales and calculate the monthly growth percentage.

✓ PySpark Actions

Actions are operations that trigger the execution of the transformations plan and return a result to the driver program or write data to storage.

Displaying Results (`show`)

- The `show()` action displays the top rows of a `DataFrame`.

Counting Records (`count`)

- The `count()` action returns the number of rows in a `DataFrame`.

Collecting Data (`collect`, `take`)

- Actions like `collect()` and `take()` return data from the `DataFrame` to the driver program. `collect()` brings all data (use with caution on large datasets), while `take(n)` brings the first `n` rows. Here, we collect the top 5 products by revenue.

✓ Saving Data (`write`)

- The `write` action saves the contents of a `DataFrame` to various data sources (e.g., Parquet, CSV, JSON). The `mode("overwrite")` option is used here to replace the file if it already exists.

1 Start coding or [generate](#) with AI.