In [1]:
```python
import os
from pyspark.sql import SparkSession
import findspark
findspark.init()
import pyspark
```

In [2]:
```python
jdbc_driver_path = os.path.abspath(r"C:\Users\sqljdbc_12.8.0.0_fra\sqljdbc_12.8\fra\jars
```

Lancer une session Spark

In [3]:
```python
spark = SparkSession.builder \
       .appName("TelecomETL1") \
       .config("spark.driver.extraClassPath", jdbc_driver_path) \
       .config("spark.executor.extraClassPath", jdbc_driver_path) \
       .getOrCreate()

print("SparkSession créée avec succès!")
```

```
SparkSession créée avec succès!
```

Configuration des connexions

In [4]:
```python
# Configuration de la connexion à la base de données source (telecom2)
jdbc_url_source = "jdbc:sqlserver://[servername]:1433;databaseName=telecom2;encrypt=true

# Configuration de la connexion à la base de données cible (spy)
jdbc_url_target = "jdbc:sqlserver://[servername]:1433;databaseName=spy;encrypt=true;trus

properties = {
    "user": "[user]",
    "password": "[pwd]",
    "driver": "com.microsoft.sqlserver.jdbc.SQLServerDriver"
}
print("connecion établie avec succès")
```

```
connecion établie avec succès
```

Apperçu sur les données

In [5]:
```python
# Liste des tables à extraire
tables = [
    "department", "worklocation", "employee", "employeeworklocation",
    "salesperson", "customer", "orders", "planinclusions", "plans",
    "billinginformation", "phonenumber", "callrecords", "salary",
    "simdata", "tracking"
]

try:
    for table_name in tables:
        print(f"\nExtraction des données de la table: {table_name}")

        # Lire les données de la table
        df = spark.read.jdbc(url=jdbc_url_source, table=table_name, properties=propertie

        print(f"Aperçu des données de {table_name}:")
        df.show(5, truncate=False)
```

Loading [MathJax]/extensions/Safe.js

```python
        print(f"Schéma de la table {table_name}:")
        df.printSchema()

        row_count = df.count()
        print(f"Nombre total de lignes dans {table_name}: {row_count}")

except Exception as e:
    print("Erreur lors de l'extraction des données:", str(e))
```

```
Extraction des données de la table: department
Aperçu des données de department:
+------------+--------------------+------+
|DepartmentId|DepartmentName      |Salary|
+------------+--------------------+------+
|1           |Information Technology|10000 |
|2           |Sales & Marketing   |5000  |
|3           |Finance             |2500  |
|4           |Human Resource      |7500  |
|5           |Customer Care       |1000  |
+------------+--------------------+------+

Schéma de la table department:
root
 |-- DepartmentId: integer (nullable = true)
 |-- DepartmentName: string (nullable = true)
 |-- Salary: integer (nullable = true)

Nombre total de lignes dans department: 5

Extraction des données de la table: worklocation
Aperçu des données de worklocation:
+----------+-------------+-----------------+
|LocationId|LocationName |NumberOfEmployees|
+----------+-------------+-----------------+
|1         |Seattle      |5                |
|2         |Washington DC|5                |
|3         |New York     |5                |
|4         |Boston       |5                |
+----------+-------------+-----------------+

Schéma de la table worklocation:
root
 |-- LocationId: integer (nullable = true)
 |-- LocationName: string (nullable = true)
 |-- NumberOfEmployees: integer (nullable = true)

Nombre total de lignes dans worklocation: 4

Extraction des données de la table: employee
Aperçu des données de employee:
+----------+---------------------+---------+---+------------+------+
|EmployeeId|Employee_Name        |SSN      |Age|DepartmentId|Salary|
+----------+---------------------+---------+---+------------+------+
|1         |Ojas Phansekar       |123456789|24 |1           |1000  |
|2         |Shreyas Kalayanaraman|245987675|24 |1           |1000  |
|3         |Saurabh Kulkarni     |734756953|24 |1           |1000  |
|4         |Vivek Shetye         |572364526|26 |1           |1000  |
|5         |Mihir Patil          |238745784|27 |1           |1000  |
+----------+---------------------+---------+---+------------+------+
only showing top 5 rows

Schéma de la table employee:
root
 |-- EmployeeId: integer (nullable = true)
 |-- Employee_Name: string (nullable = true)
 |-- SSN: string (nullable = true)
 |-- Age: integer (nullable = true)
 |-- DepartmentId: integer (nullable = true)
 |-- Salary: integer (nullable = true)

Nombre total de lignes dans employee: 37

Extraction des données de la table: employeeworklocation
```

```
Aperçu des données de employeeworklocation:
+--------------+----------+
|WorkEmployeeId|LocationId|
+--------------+----------+
|1             |2         |
|2             |4         |
|3             |3         |
|4             |1         |
|5             |2         |
+--------------+----------+
only showing top 5 rows

Schéma de la table employeeworklocation:
root
 |-- WorkEmployeeId: integer (nullable = true)
 |-- LocationId: integer (nullable = true)

Nombre total de lignes dans employeeworklocation: 20

Extraction des données de la table: salesperson
Aperçu des données de salesperson:
+-------------+--------------------+
|SalesPersonId|IdEmployeeSalesPerson|
+-------------+--------------------+
|1            |5                   |
|2            |6                   |
|3            |7                   |
|4            |8                   |
+-------------+--------------------+

Schéma de la table salesperson:
root
 |-- SalesPersonId: integer (nullable = true)
 |-- IdEmployeeSalesPerson: integer (nullable = true)

Nombre total de lignes dans salesperson: 4

Extraction des données de la table: customer
Aperçu des données de customer:
+----------+---------------+---+---+-----------+--------------------+------------------
---+
|CustomerId|CustomerName    |Sex|Age|DateOfBirth|SocialSecurityNumber|CustomerSalesPerso
nId|
+----------+---------------+---+---+-----------+--------------------+------------------
---+
|1         |Jishnu Vasudevan|M  |24 |1993-12-28 |232498675           |1
|
|2         |Harsh Shah      |M  |24 |1993-09-12 |456498675           |2
|
|3         |Rachana Rambhad |F  |24 |1993-08-19 |543498675           |3
|
|4         |Lagan Gupta     |F  |24 |1993-08-08 |765498675           |4
|
|5         |Neha Verma      |F  |24 |1993-08-27 |987498675           |1
|
+----------+---------------+---+---+-----------+--------------------+------------------
---+
only showing top 5 rows

Schéma de la table customer:
root
 |-- CustomerId: integer (nullable = true)
 |-- CustomerName: string (nullable = true)
 |-- Sex: string (nullable = true)
```
Loading [MathJax]/extensions/Safe.js

```
 |-- Age: integer (nullable = true)
 |-- DateOfBirth: date (nullable = true)
 |-- SocialSecurityNumber: integer (nullable = true)
 |-- CustomerSalesPersonId: integer (nullable = true)

Nombre total de lignes dans customer: 20

Extraction des données de la table: orders
Aperçu des données de orders:
+-------+-----------------+------------------+---------------+
|OrderId|OrderType        |OrderStatus       |OrderCustomerId|
+-------+-----------------+------------------+---------------+
|1      |2 day shipping   |Shipped           |1              |
|2      |Priority Shipping|Partially Shipped |2              |
|3      |Standard         |Payment Incomplete|3              |
|4      |2 day shipping   |Order Cancelled   |4              |
|5      |Standard         |Pending           |5              |
+-------+-----------------+------------------+---------------+
only showing top 5 rows

Schéma de la table orders:
root
 |-- OrderId: integer (nullable = true)
 |-- OrderType: string (nullable = true)
 |-- OrderStatus: string (nullable = true)
 |-- OrderCustomerId: integer (nullable = true)

Nombre total de lignes dans orders: 20

Extraction des données de la table: planinclusions
Aperçu des données de planinclusions:
+------+-----+-----------+------------+
|PlanId|Data |Talktime   |TextMessages|
+------+-----+-----------+------------+
|1     |500MB|60 Minutes |100         |
|2     |500MB|120 Minutes|200         |
|3     |500MB|180 Minutes|300         |
|4     |500MB|240 Minutes|400         |
|5     |500MB|300 Minutes|500         |
+------+-----+-----------+------------+
only showing top 5 rows

Schéma de la table planinclusions:
root
 |-- PlanId: integer (nullable = true)
 |-- Data: string (nullable = true)
 |-- Talktime: string (nullable = true)
 |-- TextMessages: string (nullable = true)

Nombre total de lignes dans planinclusions: 20

Extraction des données de la table: plans
Aperçu des données de plans:
+-------+---------+------------------+---------------+
|PlansId|PlansType|PlanName          |PlanInclusionId|
+-------+---------+------------------+---------------+
|1      |Prepaid  |Basic Plan        |1              |
|2      |Prepaid  |Every Minute Counts|2             |
|3      |Postpaid |Family            |3              |
|4      |Postpaid |Enjoy Data        |4              |
|5      |Postpaid |Finger tips       |5              |
+-------+---------+------------------+---------------+
only showing top 5 rows
```

```
Schéma de la table plans:
root
 |-- PlansId: integer (nullable = true)
 |-- PlansType: string (nullable = true)
 |-- PlanName: string (nullable = true)
 |-- PlanInclusionId: integer (nullable = true)

Nombre total de lignes dans plans: 10

Extraction des données de la table: billinginformation
Aperçu des données de billinginformation:
+----------+------------+--------+------------+-----+
|BillNumber|IncludedData|DataUsed|BalancedData|Tax  |
+----------+------------+--------+------------+-----+
|1         |50          |40      |10          |15.10|
|2         |100         |60      |40          |20.30|
|3         |200         |100     |100         |30.30|
|4         |500         |200     |300         |40.30|
|5         |500         |100     |400         |50.00|
+----------+------------+--------+------------+-----+
only showing top 5 rows

Schéma de la table billinginformation:
root
 |-- BillNumber: integer (nullable = true)
 |-- IncludedData: integer (nullable = true)
 |-- DataUsed: integer (nullable = true)
 |-- BalancedData: integer (nullable = true)
 |-- Tax: decimal(18,2) (nullable = true)

Nombre total de lignes dans billinginformation: 15

Extraction des données de la table: phonenumber
Aperçu des données de phonenumber:
+-------------+-----------+---------------+
|AccountNumber|PhoneNumber|PhoneBillNumber|
+-------------+-----------+---------------+
|9            |1235465768 |3              |
|10           |1235465768 |4              |
|11           |1675849305 |5              |
|12           |1345267859 |6              |
|13           |1578893409 |7              |
+-------------+-----------+---------------+
only showing top 5 rows

Schéma de la table phonenumber:
root
 |-- AccountNumber: integer (nullable = true)
 |-- PhoneNumber: long (nullable = true)
 |-- PhoneBillNumber: integer (nullable = true)

Nombre total de lignes dans phonenumber: 11

Extraction des données de la table: callrecords
Aperçu des données de callrecords:
+------+-------------------+-------------------+-------------------+-----------------+
|CallId|CallStartTime      |CallEndTime        |CallDuration       |CallAccountNumber|
+------+-------------------+-------------------+-------------------+-----------------+
|2     |1970-01-01 12:20:20|1970-01-01 12:21:20|1970-01-01 00:01:00|10               |
|3     |1970-01-01 11:23:24|1970-01-01 15:40:30|1970-01-01 04:17:06|10               |
|4     |1970-01-01 08:30:10|1970-01-01 08:32:20|1970-01-01 00:02:10|11               |
|5     |1970-01-01 21:45:30|1970-01-01 21:50:34|1970-01-01 00:05:04|14               |
|6     |1970-01-01 12:32:21|1970-01-01 12:34:20|1970-01-01 00:01:59|10               |
+------+-------------------+-------------------+-------------------+-----------------+
```

```
only showing top 5 rows

Schéma de la table callrecords:
root
 |-- CallId: integer (nullable = true)
 |-- CallStartTime: timestamp (nullable = true)
 |-- CallEndTime: timestamp (nullable = true)
 |-- CallDuration: timestamp (nullable = true)
 |-- CallAccountNumber: integer (nullable = true)

Nombre total de lignes dans callrecords: 19

Extraction des données de la table: salary
Aperçu des données de salary:
+----------+-------------+------------+------+
|EmployeeId|EmployeeName  |DepartmentId|Salary|
+----------+-------------+------------+------+
|27        |Devdip Sen    |5           |10000 |
|28        |Alpana Sharan |3           |2500  |
|29        |Priyanka Singh|3           |2500  |
|30        |Ranjani Iyer  |2           |5000  |
|31        |Amlan Bhuyan  |4           |7500  |
+----------+-------------+------------+------+
only showing top 5 rows

Schéma de la table salary:
root
 |-- EmployeeId: integer (nullable = true)
 |-- EmployeeName: string (nullable = true)
 |-- DepartmentId: integer (nullable = true)
 |-- Salary: integer (nullable = true)

Nombre total de lignes dans salary: 12

Extraction des données de la table: simdata
Aperçu des données de simdata:
+----------------+--------+-------------+----------------+-------------+
|SimNumber       |SimType |SimCustomerId|SimAccountNumber|SimPlanNumber|
+----------------+--------+-------------+----------------+-------------+
|1234567890123460|Postpaid|2            |19              |3            |
|1234567890123461|Prepaid |16           |10              |1            |
|1234567890123462|Postpaid|14           |12              |3            |
|1234567890123463|Postpaid|1            |14              |5            |
|1234567890123464|Prepaid |12           |16              |8            |
+----------------+--------+-------------+----------------+-------------+
only showing top 5 rows

Schéma de la table simdata:
root
 |-- SimNumber: long (nullable = true)
 |-- SimType: string (nullable = true)
 |-- SimCustomerId: integer (nullable = true)
 |-- SimAccountNumber: integer (nullable = true)
 |-- SimPlanNumber: integer (nullable = true)

Nombre total de lignes dans simdata: 10

Extraction des données de la table: tracking
Aperçu des données de tracking:
+----------+---------------------------------+---------------+
|TrackingId|TrackingStatus                   |TrackingOrderId|
+----------+---------------------------------+---------------+
|1         |On the way                       |10             |
|2         |Arrived to courier service       |9              |
```

```
|3         |Near by closest dispatch location|14             |
|4         |Arrived to courier service       |16             |
|5         |Arrived to courier service       |17             |
+----------+---------------------------------+---------------+
only showing top 5 rows

Schéma de la table tracking:
root
 |-- TrackingId: integer (nullable = true)
 |-- TrackingStatus: string (nullable = true)
 |-- TrackingOrderId: integer (nullable = true)

Nombre total de lignes dans tracking: 9
```

Connexion à toutes les tables

In [6]:

```
employee_df = spark.read.jdbc(url=jdbc_url_source, table="employee", properties=properti
department_df = spark.read.jdbc(url=jdbc_url_source, table="department", properties=prop
customer_df = spark.read.jdbc(url=jdbc_url_source, table="customer", properties=properti
orders_df = spark.read.jdbc(url=jdbc_url_source, table="orders", properties=properties)
callrecords_df = spark.read.jdbc(url=jdbc_url_source, table="callrecords", properties=pr
plans_df = spark.read.jdbc(url=jdbc_url_source, table="plans", properties=properties)
simdata_df = spark.read.jdbc(url=jdbc_url_source, table="simdata", properties=properties
worklocation_df = spark.read.jdbc(url=jdbc_url_source, table="worklocation", properties=
employeeworklocation_df = spark.read.jdbc(url=jdbc_url_source, table="employeeworklocati
salesperson_df = spark.read.jdbc(url=jdbc_url_source, table="salesperson", properties=pr
planinclusions_df = spark.read.jdbc(url=jdbc_url_source, table="planinclusions", propert
billinginformation_df = spark.read.jdbc(url=jdbc_url_source, table="billinginformation",
phonenumber_df = spark.read.jdbc(url=jdbc_url_source, table="phonenumber", properties=pr
salary_df = spark.read.jdbc(url=jdbc_url_source, table="salary", properties=properties)
tracking_df = spark.read.jdbc(url=jdbc_url_source, table="tracking", properties=properti

# Vérifier que les DataFrames ont été correctement chargés
print("Nombre de lignes dans chaque table:")
print("employee:", employee_df.count())
print("department:", department_df.count())
print("customer:", customer_df.count())
print("orders:", orders_df.count())
print("callrecords:", callrecords_df.count())
print("plans:", plans_df.count())
print("simdata:", simdata_df.count())
print("worklocation:", worklocation_df.count())
print("employeeworklocation:", employeeworklocation_df.count())
print("salesperson:", salesperson_df.count())
print("planinclusions:", planinclusions_df.count())
print("billinginformation:", billinginformation_df.count())
print("phonenumber:", phonenumber_df.count())
print("salary:", salary_df.count())
print("tracking:", tracking_df.count())
```

```
Nombre de lignes dans chaque table:
employee: 37
department: 5
customer: 20
orders: 20
callrecords: 19
plans: 10
simdata: 10
worklocation: 4
employeeworklocation: 20
salesperson: 4
planinclusions: 20
billinginformation: 15
phonenumber: 11
salary: 12
tracking: 9
```

In [7]:

```
# Analyse des performances des commerciaux
from pyspark.sql.functions import col, sum, count, datediff, current_date, when, avg, ro
from pyspark.sql.window import Window
salesperson_performance = customer_df.join(salesperson_df, customer_df.CustomerSalesPers
    .join(employee_df, salesperson_df.IdEmployeeSalesPerson == employee_df.EmployeeId) \
    .join(simdata_df, customer_df.CustomerId == simdata_df.SimCustomerId) \
    .join(plans_df, simdata_df.SimPlanNumber == plans_df.PlansId) \
    .groupBy("IdEmployeeSalesPerson", "Employee_Name") \
    .agg(
        count("CustomerId").alias("TotalCustomers"),
        sum(when(col("PlansType") == "Postpaid", 1).otherwise(0)).alias("PostpaidPlans")
        sum(when(col("PlansType") == "Prepaid", 1).otherwise(0)).alias("PrepaidPlans"),
        avg("Salary").alias("AvgSalary")
    ) \
    .withColumn("PostpaidRatio", round(col("PostpaidPlans") / col("TotalCustomers"), 2))
salesperson_performance.show()
```

```
+--------------------+---------------+--------------+-------------+------------+-------
--+-------------+
|IdEmployeeSalesPerson|  Employee_Name|TotalCustomers|PostpaidPlans|PrepaidPlans|AvgSala
ry|PostpaidRatio|
+--------------------+---------------+--------------+-------------+------------+-------
--+-------------+
|                    8|Shantanu Sawant|             2|            0|           2|   750
0.0|          0.0|
|                    6|   Karan Thevar|             3|            3|           0|   750
0.0|          1.0|
|                    5|    Mihir Patil|             3|            2|           1|   100
0.0|         0.67|
|                    7|  Chetan Mistry|             2|            1|           1|   750
0.0|          0.5|
+--------------------+---------------+--------------+-------------+------------+-------
--+-------------+
```

Analyse des tendances d'utilisation des données par plan et par mois

In [8]:

```
data_usage_trends = callrecords_df.join(simdata_df, callrecords_df.CallAccountNumber ==
    .join(plans_df, simdata_df.SimPlanNumber == plans_df.PlansId) \
    .withColumn("CallMonth", month("CallStartTime")) \
    .withColumn("CallYear", year("CallStartTime")) \
    .groupBy("PlansType", "PlanName", "CallYear", "CallMonth") \
    .agg(
        ...col("CallDuration")).alias("TotalCallDuration"),
```

```
        count("CallId").alias("TotalCalls")
    ) \
    .orderBy("CallYear", "CallMonth", "PlansType")
data_usage_trends.show()
```

```
+---------+-----------------+--------+---------+-----------------+----------+
|PlansType|         PlanName|CallYear|CallMonth|TotalCallDuration|TotalCalls|
+---------+-----------------+--------+---------+-----------------+----------+
| Postpaid|           Family|    1970|        1|           3893.0|         3|
| Postpaid|     Do not disturb|    1970|        1|           5816.0|         4|
| Postpaid|       Finger tips|    1970|        1|            304.0|         1|
|  Prepaid|        Basic Plan|    1970|        1|          33317.0|         8|
|  Prepaid|Continuous Texting|    1970|        1|           5567.0|         2|
|  Prepaid|    Talk For Hours|    1970|        1|            449.0|         1|
+---------+-----------------+--------+---------+-----------------+----------+
```

Analyse de la rotation du personnel et son impact sur les ventes

In [9]:

```
employee_turnover = employee_df.join(salesperson_df, employee_df.EmployeeId == salespers
    .join(customer_df, salesperson_df.SalesPersonId == customer_df.CustomerSalesPersonId
    .join(orders_df, customer_df.CustomerId == orders_df.OrderCustomerId, "left_outer")
    .groupBy("EmployeeId", "Employee_Name", "DepartmentId") \
    .agg(
        count("CustomerId").alias("TotalCustomers"),
        count("OrderId").alias("TotalOrders"),
        sum(when(col("OrderStatus") == "Cancelled", 1).otherwise(0)).alias("CancelledOrd
    ) \
    .join(salary_df, "EmployeeId") \
    .withColumn("OrderCancellationRate", round(col("CancelledOrders") / col("TotalOrders
employee_turnover.show()
```

```
+----------+--------------+-----------+-------------+-----------+--------------+----
-----------+-----------+------+--------------------+
|EmployeeId|  Employee_Name|DepartmentId|TotalCustomers|TotalOrders|CancelledOrders|   E
mployeeName|DepartmentId|Salary|OrderCancellationRate|
+----------+--------------+-----------+-------------+-----------+--------------+----
-----------+-----------+------+--------------------+
|        27|     Devdip Sen|           5|            0|          0|             0|
Devdip Sen|          5| 10000|                NULL|
|        28|  Alpana Sharan|           3|            0|          0|             0|  Al
pana Sharan|          3|  2500|                NULL|
|        29| Priyanka Singh|           3|            0|          0|             0| Pri
yanka Singh|          3|  2500|                NULL|
|        30|    Ranjani Iyer|           2|            0|          0|             0|   R
anjani Iyer|          2|  5000|                NULL|
|        31|    Amlan Bhuyan|           4|            0|          0|             0|   A
mlan Bhuyan|          4|  7500|                NULL|
|        32|Manoj Prabhakar|           1|            0|          0|             0|Mano
j Prabhakar|          1|  1000|                NULL|
|        33|      Raj Phadke|           5|            0|          0|             0|
Raj Phadke|          5| 10000|                NULL|
|        34|     Priya Yadav|           1|            0|          0|             0|
Priya Yadav|          1|  1000|                NULL|
|        35|    Sayali Joshi|           4|            0|          0|             0|   S
ayali Joshi|          4|  7500|                NULL|
|        36|    Pranav Patil|           5|            0|          0|             0|   P
ranav Patil|          5| 10000|                NULL|
|        37|     Rohit Patil|           3|            0|          0|             0|
Rohit Patil|          3|  2500|                NULL|
|        38|  Swanand Sapre|           5|            0|          0|             0|  Sw
anand Sapre|          5| 10000|                NULL|
+----------+--------------+-----------+-------------+-----------+--------------+----
-----------+-----------+------+--------------------+
```

Analyse géographique des performances de vente

In [10]:

```
geo_sales_performance = employeeworklocation_df.join(employee_df, employeeworklocation_d
    .join(worklocation_df, employeeworklocation_df.LocationId == worklocation_df.Locatio
    .join(salesperson_df, employee_df.EmployeeId == salesperson_df.IdEmployeeSalesPerson
    .join(customer_df, salesperson_df.SalesPersonId == customer_df.CustomerSalesPersonId
    .join(orders_df, customer_df.CustomerId == orders_df.OrderCustomerId) \
    .groupBy("LocationName") \
    .agg(
        count("OrderId").alias("TotalOrders"),
        sum(when(col("OrderStatus") == "Shipped", 1).otherwise(0)).alias("CompletedOrder
        avg("Salary").alias("AvgSalary")
    ) \
    .withColumn("OrderCompletionRate", round(col("CompletedOrders") / col("TotalOrders")
geo_sales_performance.show()
```

```
+-------------+-----------+---------------+---------+-------------------+
| LocationName|TotalOrders|CompletedOrders|AvgSalary|OrderCompletionRate|
+-------------+-----------+---------------+---------+-------------------+
|      Seattle|          5|              0|   7500.0|                0.0|
|Washington DC|          5|              1|   1000.0|                0.2|
|     New York|          5|              0|   7500.0|                0.0|
|       Boston|          5|              2|   7500.0|                0.4|
+-------------+-----------+---------------+---------+-------------------+
```

In [11]:

```
# Analyse du réseau social des clients (qui appelle qui)
from pyspark.sql.functions import col, sum, count, datediff, current_date, when, avg, ro
Loading [MathJax]/extensions/Safe.js sql.window import Window
```

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.clustering import KMeans
social_network_analysis = callrecords_df.alias("caller") \
    .join(callrecords_df.alias("receiver"), col("caller.CallAccountNumber") != col("rece
    .groupBy("caller.CallAccountNumber", "receiver.CallAccountNumber") \
    .agg(
        count("*").alias("CallFrequency"),
        avg(unix_timestamp(col("caller.CallEndTime")) - unix_timestamp(col("caller.CallS
    ) \
    .orderBy(col("CallFrequency").desc())
social_network_analysis.show()
```

```
+-----------------+-----------------+-------------+---------------------+
|CallAccountNumber|CallAccountNumber|CallFrequency|AvgCallDurationSeconds|
+-----------------+-----------------+-------------+---------------------+
|               10|               11|           32|             4164.625|
|               11|               10|           32|               1454.0|
|               13|               10|           16|               2783.5|
|               10|               13|           16|             4164.625|
|               10|               12|           16|             4164.625|
|               12|               10|           16|                956.5|
|               17|               10|            8|                449.0|
|               10|               19|            8|             4164.625|
|               11|               13|            8|               1454.0|
|               13|               11|            8|               2783.5|
|               10|               17|            8|             4164.625|
|               12|               11|            8|                956.5|
|               10|               14|            8|             4164.625|
|               19|               10|            8|               1980.0|
|               11|               12|            8|               1454.0|
|               14|               10|            8|                304.0|
|               14|               11|            4|                304.0|
|               11|               19|            4|               1454.0|
|               17|               11|            4|                449.0|
|               11|               14|            4|               1454.0|
+-----------------+-----------------+-------------+---------------------+
only showing top 20 rows
```

In [12]:

```
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
from pyspark.sql.functions import col, count, sum, avg, when, datediff, current_date, to
from pyspark.sql.window import Window
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator

# Préparation des données pour la segmentation
customer_features = customer_df.join(simdata_df, customer_df.CustomerId == simdata_df.Si
    .join(callrecords_df, simdata_df.SimAccountNumber == callrecords_df.CallAccountNumbe
    .join(orders_df, customer_df.CustomerId == orders_df.OrderCustomerId, "left") \
    .groupBy("CustomerId") \
    .agg(
        count("OrderId").alias("TotalOrders"),
        sum(when(col("CallEndTime").isNotNull() & col("CallStartTime").isNotNull(),
                 col("CallEndTime").cast("long") - col("CallStartTime").cast("long"))
             .otherwise(0)).alias("TotalCallDurationSeconds"),
        count("CallId").alias("TotalCalls"),
        avg("Age").alias("Age")
    ) \
    .withColumn("AvgOrderValue", col("TotalOrders") * 50)  # Supposons un montant moyen

# Gestion des valeurs nulles
customer_features = customer_features.na.fill({
    ders": 0,
```

Loading [MathJax]/extensions/Safe.js

```python
    "TotalCallDurationSeconds": 0,
    "TotalCalls": 0,
    "Age": customer_features.select(avg("Age")).first()[0],
    "AvgOrderValue": 0
})

# Préparation des caractéristiques pour le clustering
feature_cols = ["TotalOrders", "TotalCallDurationSeconds", "TotalCalls", "Age", "AvgOrde
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")
customer_features_vector = assembler.transform(customer_features)

# Application du clustering K-means
kmeans = KMeans(k=5, seed=1)  # 5 segments
model = kmeans.fit(customer_features_vector)
customer_segments = model.transform(customer_features_vector)

# Évaluation du modèle
evaluator = ClusteringEvaluator()
silhouette = evaluator.evaluate(customer_segments)
print(f"Silhouette avec carré de la distance euclidienne = {silhouette}")

# Affichage des résultats de segmentation
print("Résultats de la segmentation des clients:")
customer_segments.select("CustomerId", "prediction").show()

#  Analyse de l'évolution des plans des clients dans le temps
plan_evolution = simdata_df.join(plans_df, simdata_df.SimPlanNumber == plans_df.PlansId)
    .join(customer_df, simdata_df.SimCustomerId == customer_df.CustomerId) \
    .withColumn("CustomerAge", datediff(current_date(), to_date(col("DateOfBirth"))) / 3
    .withColumn("PlanRank", dense_rank().over(Window.partitionBy("SimCustomerId").orderB
    .groupBy("SimCustomerId", "CustomerName", "CustomerAge") \
    .agg(
        count("PlansId").alias("TotalPlansUsed"),
        collect_list("PlanName").alias("PlanSequence")
    ) \
    .orderBy(col("TotalPlansUsed").desc())

print("Analyse de l'évolution des plans des clients:")
plan_evolution.show(truncate=False)
```

```
Silhouette avec carré de la distance euclidienne = 0.9318914716713715
Résultats de la segmentation des clients:
+----------+----------+
|CustomerId|prediction|
+----------+----------+
|        12|         0|
|         1|         4|
|        13|         2|
|         6|         0|
|        16|         1|
|         3|         0|
|        20|         0|
|         5|         4|
|        19|         0|
|        15|         2|
|         9|         0|
|        17|         0|
|         4|         0|
|         8|         0|
|         7|         0|
|        10|         0|
|        11|         0|
|        14|         3|
|         2|         3|
|        18|         0|
+----------+----------+

Analyse de l'évolution des plans des clients:
+-------------+----------------+------------------+--------------+--------------------+
|SimCustomerId|CustomerName    |CustomerAge       |TotalPlansUsed|PlanSequence        |
+-------------+----------------+------------------+--------------+--------------------+
|12           |Kal Bugrara     |63.663244353182755|1             |[Enjoy surfing]     |
|1            |Jishnu Vasudevan |30.663928815879533|1             |[Finger tips]       |
|13           |Neeraj Rajput   |33.831622176591374|1             |[Do not disturb]    |
|16           |Vijayshree Uppili|33.01300479123888 |1             |[Basic Plan]        |
|5            |Neha Verma      |31.000684462696782|1             |[Talk For Hours]    |
|15           |Sameer Goel     |34.65845311430527 |1             |[Continuous Texting]|
|7            |Anubhav Gupta   |33.66461327857632 |1             |[Enjoy Data]        |
|10           |Dharit Shah     |30.666666666666668|1             |[Powerful Speed]    |
|14           |Shruti Mehta    |32.695414099931554|1             |[Family]            |
|2            |Harsh Shah      |30.95687885010267 |1             |[Family]            |
+-------------+----------------+------------------+--------------+--------------------+
```

In [14]:

```python
from pyspark.sql.functions import col, when, count, sum, avg, datediff, current_date, to
from pyspark.sql.window import Window
from pyspark.sql.types import DoubleType


# Fonction UDF pour calculer le coût des dépassements
@udf(returnType=DoubleType())
def calculate_overage_cost(data_used, included_data, balanced_data):
    if data_used > included_data:
        return (data_used - included_data - balanced_data) * 0.1  # Supposons 0.1 par un
    return 0.0

customer_usage = customer_df.join(
    simdata_df, customer_df.CustomerId == simdata_df.SimCustomerId
).join(
    plans_df, simdata_df.SimPlanNumber == plans_df.PlansId
).join(
    planinclusions_df, plans_df.PlanInclusionId == planinclusions_df.PlanId
).join(
    callrecords_df, simdata_df.SimAccountNumber == callrecords_df.CallAccountNumber
```

Loading [MathJax]/extensions/Safe.js

```python
).join(
    billinginformation_df, simdata_df.SimAccountNumber == billinginformation_df.BillNumb
).join(
    orders_df, customer_df.CustomerId == orders_df.OrderCustomerId
)

# Analyse de la rentabilité des clients
customer_profitability = customer_usage.groupBy(
    "CustomerId", "CustomerName", "SimType", "PlansType", "PlanName"
).agg(
    sum("Tax").alias("TotalTax"),
    sum("IncludedData").alias("TotalIncludedData"),
    sum("DataUsed").alias("TotalDataUsed"),
    sum("BalancedData").alias("TotalBalancedData"),
    count("CallId").alias("TotalCalls"),
    sum(when(col("CallEndTime").isNotNull() & col("CallStartTime").isNotNull(),
             col("CallEndTime").cast("long") - col("CallStartTime").cast("long"))
        .otherwise(0)).alias("TotalCallDurationSeconds"),
    count("OrderId").alias("TotalOrders"),
    sum(when(col("OrderStatus") == "Shipped", 1).otherwise(0)).alias("CompletedOrders")
    # Retirez la ligne faisant référence à "Salary"
).withColumn(
    "OverageCost", calculate_overage_cost(col("TotalDataUsed"), col("TotalIncludedData")
).withColumn(
    "TotalRevenue", lit(50) * 12 + col("OverageCost") + col("TotalTax")
).withColumn(
    "AvgCallDurationMinutes", when(col("TotalCalls") > 0, round(col("TotalCallDurationSe
).withColumn(
    "OrderCompletionRate", when(col("TotalOrders") > 0, round(col("CompletedOrders") / c
).withColumn(
    "CustomerLifetimeValue", col("TotalRevenue") * 3 - (col("TotalCalls") * 0.05 + col("
)

# Calcul des métriques de rentabilité
window_spec = Window.orderBy(col("CustomerLifetimeValue").desc())
customer_profitability = customer_profitability.withColumn(
    "ProfitabilityRank", dense_rank().over(window_spec)
).withColumn(
    "ProfitabilityScore",
    (col("CustomerLifetimeValue") / 1000 * 0.4) +
    (col("OrderCompletionRate") * 0.3) +
    (col("AvgCallDurationMinutes") / 10 * 0.3)
).withColumn(
    "ProfitabilityCategory",
    when(col("ProfitabilityScore") >= 0.8, "High Value")
    .when(col("ProfitabilityScore") >= 0.6, "Medium Value")
    .when(col("ProfitabilityScore") >= 0.4, "Average Value")
    .otherwise("Low Value")
)

# Affichage des résultats
print("Analyse de la rentabilité des clients:")
customer_profitability.select(
    "CustomerName", "SimType", "PlanName", "TotalRevenue", "OverageCost",
    "TotalCalls", "TotalOrders", "AvgCallDurationMinutes", "OrderCompletionRate",
    format_number("CustomerLifetimeValue", 2).alias("CustomerLifetimeValue"),
    "ProfitabilityRank", format_number("ProfitabilityScore", 2).alias("ProfitabilityScor
    "ProfitabilityCategory"
).show(truncate=False)

# Analyse des caractéristiques des clients les plus rentables
top_customers = customer_profitability.filter(col("ProfitabilityCategory") == "High Valu

print("\nCaractéristiques des clients les plus rentables:")
```

```
top_customers.groupBy("SimType", "PlansType").agg(
    count("CustomerId").alias("CustomerCount"),
    avg("TotalRevenue").alias("AvgRevenue"),
    avg("TotalCalls").alias("AvgCalls"),
    avg("TotalOrders").alias("AvgOrders"),
    avg("AvgCallDurationMinutes").alias("AvgCallDuration"),
    avg("OrderCompletionRate").alias("AvgOrderCompletionRate")
).orderBy(col("CustomerCount").desc()).show(truncate=False)

# Analyse de l'impact des dépassements sur la rentabilité
print("\nImpact des dépassements sur la rentabilité:")
customer_profitability.groupBy("ProfitabilityCategory").agg(
    avg("OverageCost").alias("AvgOverageCost"),
    avg("TotalRevenue").alias("AvgRevenue"),
    avg("CustomerLifetimeValue").alias("AvgLifetimeValue"),
    (avg("OverageCost") / avg("TotalRevenue") * 100).alias("OverageCostPercentage")
).orderBy("ProfitabilityCategory").show(truncate=False)
```

Analyse de la rentabilité des clients:

CustomerName	SimType	PlanName	TotalRevenue	OverageCost	TotalCalls	TotalOrders	AvgCallDurationMinutes	OrderCompletionRate	CustomerLifetimeValue	ProfitabilityRank	ProfitabilityScore	ProfitabilityCategory
Vijayshree Uppili	Prepaid	Basic Plan	3000.0	0.0	8	8	69.41	0.0	8,983.60	1	5.68	High Value
Neeraj Rajput	Postpaid	Do not disturb	1320.0	0.0	4	4	24.23	0.0	3,951.80	2	2.31	High Value
Shruti Mehta	Postpaid	Family	960.0	0.0	2	2	15.94	1.0	2,875.90	3	1.93	High Value
Sameer Goel	Prepaid	Continuous Texting	940.0	0.0	2	2	46.39	0.0	2,815.90	4	2.52	High Value
Jishnu Vasudevan	Postpaid	Finger tips	800.0	0.0	1	1	5.07	1.0	2,397.95	5	1.41	High Value

Caractéristiques des clients les plus rentables:

SimType	PlansType	CustomerCount	AvgRevenue	AvgCalls	AvgOrders	AvgCallDuration	AvgOrderCompletionRate
Postpaid	Postpaid	3	1026.6666666666667	2.3333333333333335	2.3333333333333335	15.08	0.6666666666666666
Prepaid	Prepaid	2	1970.0	5.0	5.0	57.9	0.0

Impact des dépassements sur la rentabilité:

ProfitabilityCategory	AvgOverageCost	AvgRevenue	AvgLifetimeValue	OverageCostPercentage
High Value	0.0	1404.0	4205.030000000001	0.0

Performances de vente par localisation et département

```
In [15]: from pyspark.sql.functions import count, sum, avg, col

sales_performance = (
    customer_df
    .join(salesperson_df, customer_df.CustomerSalesPersonId == salesperson_df.SalesPerso
    .join(employee_df, salesperson_df.IdEmployeeSalesPerson == employee_df.EmployeeId)
    .join(department_df, employee_df.DepartmentId == department_df.DepartmentId)
    .join(employeeworklocation_df, employee_df.EmployeeId == employeeworklocation_df.Wor
    .join(worklocation_df, employeeworklocation_df.LocationId == worklocation_df.Locatio
    .join(orders_df, customer_df.CustomerId == orders_df.OrderCustomerId)
)

# Analyser les performances
performance_metrics = (
    sales_performance
    .groupBy("LocationName", "DepartmentName")
    .agg(
        count("CustomerId").alias("TotalCustomers"),
        count("OrderId").alias("TotalOrders"),
        sum(when(col("OrderStatus") == "Shipped", 1).otherwise(0)).alias("CompletedOrder
        avg(when(col("OrderStatus") == "Shipped", 1).otherwise(0)).alias("OrderCompletio
    )
    .orderBy(col("TotalOrders").desc())
)

performance_metrics.show()
```

```
+-------------+--------------------+--------------+-----------+---------------+---------
----------+
| LocationName|      DepartmentName|TotalCustomers|TotalOrders|CompletedOrders|OrderComp
letionRate|
+-------------+--------------------+--------------+-----------+---------------+---------
----------+
|     New York|      Human Resource|             5|          5|              0|
0.0|
|Washington DC|Information Techn...|             5|          5|              1|
0.2|
|       Boston|      Human Resource|             5|          5|              2|
0.4|
|      Seattle|      Human Resource|             5|          5|              0|
0.0|
+-------------+--------------------+--------------+-----------+---------------+---------
----------+
```

Analyse des plans en termes d'utilisation des données et de contribution aux revenus

```
In [16]: from pyspark.sql.functions import sum, avg, col, round

data_usage_revenue = (
    simdata_df
    .join(plans_df, simdata_df.SimPlanNumber == plans_df.PlansId)
    .join(planinclusions_df, plans_df.PlanInclusionId == planinclusions_df.PlanId)
    .join(phonenumber_df, simdata_df.SimAccountNumber == phonenumber_df.AccountNumber)
    .join(billinginformation_df, phonenumber_df.PhoneBillNumber == billinginformation_df
)

usage_revenue_metrics = (
    data_usage_revenue
    .groupBy("PlansType", "PlanName")
```

Loading [MathJax]/extensions/Safe.js

```
    .agg(
        count("SimNumber").alias("TotalSubscribers"),
        round(avg("IncludedData"), 2).alias("AvgIncludedData"),
        round(avg("DataUsed"), 2).alias("AvgDataUsed"),
        round(avg(col("DataUsed") / col("IncludedData")), 2).alias("AvgDataUsageRate"),
        round(sum("Tax"), 2).alias("TotalRevenue")
    )
    .orderBy(col("TotalRevenue").desc())
)

usage_revenue_metrics.show()
```

```
+---------+-----------------+----------------+---------------+-----------+-------------
---+------------+
|PlansType|         PlanName|TotalSubscribers|AvgIncludedData|AvgDataUsed|AvgDataUsageR
ate|TotalRevenue|
+---------+-----------------+----------------+---------------+-----------+-------------
---+------------+
|  Prepaid|    Enjoy surfing|               1|         1500.0|      600.0|
0.4|      300.00|
| Postpaid|           Family|               2|         1500.0|      300.0|
0.18|      240.00|
|  Prepaid|   Talk For Hours|               1|         1500.0|      800.0|
0.53|      180.00|
| Postpaid|   Powerful Speed|               1|         2000.0|      900.0|
0.45|      180.00|
| Postpaid|      Finger tips|               1|         1200.0|      400.0|
0.33|      100.00|
| Postpaid|       Enjoy Data|               1|         1500.0|      400.0|
0.27|      100.00|
|  Prepaid|Continuous Texting|              1|          800.0|      300.0|
0.38|       60.00|
| Postpaid|   Do not disturb|               1|          500.0|      100.0|
0.2|       50.00|
|  Prepaid|       Basic Plan|               1|          500.0|      200.0|
0.4|       40.30|
+---------+-----------------+----------------+---------------+-----------+-------------
---+------------+
```

la relation entre les interactions d'appels et la satisfaction des client

In [17]:

```
from pyspark.sql.functions import count, sum, avg, col, datediff, to_timestamp, when

# Joindre les tables nécessaires
call_customer_data = (
    callrecords_df
    .join(phonenumber_df, callrecords_df.CallAccountNumber == phonenumber_df.AccountNumb
    .join(simdata_df, phonenumber_df.AccountNumber == simdata_df.SimAccountNumber)
    .join(customer_df, simdata_df.SimCustomerId == customer_df.CustomerId)
    .join(orders_df, customer_df.CustomerId == orders_df.OrderCustomerId)
)

# Calculer les métriques d'appels et de satisfaction client
call_satisfaction_metrics = (
    call_customer_data
    .withColumn("CallDurationMinutes", (col("CallDuration").cast("long") / 60))
    .withColumn("CustomerAge", datediff(to_timestamp(lit("2023-05-23")), col("DateOfBirt
    .groupBy("CustomerId", "CustomerName", "Sex")
    .agg(
        count("CallId").alias("TotalCalls"),
```

Loading [MathJax]/extensions/Safe.js

```
        round(sum("CallDurationMinutes"), 2).alias("TotalCallDurationMinutes"),
        round(avg("CallDurationMinutes"), 2).alias("AvgCallDurationMinutes"),
        sum(when(col("OrderStatus") == "Shipped", 1).otherwise(0)).alias("CompletedOrder
        sum(when(col("OrderStatus") == "Order Cancelled", 1).otherwise(0)).alias("Cancel
    )
    .withColumn("CustomerSatisfactionScore",
                when(col("CompletedOrders") > col("CancelledOrders"), "High")
                .when(col("CompletedOrders") == col("CancelledOrders"), "Medium")
                .otherwise("Low"))
    .orderBy(col("TotalCallDurationMinutes").desc())
)

call_satisfaction_metrics.show()
```

```
+----------+----------------+---+----------+------------------------+------------------
----+---------------+---------------+-------------------------+
|CustomerId|    CustomerName|Sex|TotalCalls|TotalCallDurationMinutes|AvgCallDurationMin
utes|CompletedOrders|CancelledOrders|CustomerSatisfactionScore|
+----------+----------------+---+----------+------------------------+------------------
----+---------------+---------------+-------------------------+
|        16|Vijayshree Uppili|  F|         8|                  555.28|                 6
9.41|              0|              8|                      Low|
|        13|    Neeraj Rajput|  M|         4|                   96.93|                 2
4.23|              0|              0|                   Medium|
|        15|      Sameer Goel|  M|         2|                   92.78|                 4
6.39|              0|              0|                   Medium|
|         2|       Harsh Shah|  M|         1|                    33.0|                  
33.0|              0|              0|                   Medium|
|        14|     Shruti Mehta|  F|         2|                   31.88|                 1
5.94|              2|              0|                     High|
|         5|       Neha Verma|  F|         1|                    7.48|                  
7.48|              0|              0|                   Medium|
|         1| Jishnu Vasudevan|  M|         1|                    5.07|                  
5.07|              1|              0|                     High|
+----------+----------------+---+----------+------------------------+------------------
----+---------------+---------------+-------------------------+
```

Envoyer les résultats vers la destination

In [18]:

```
def send_to_sql_server(df, table_name, mode="append"):
    """
    Envoie un DataFrame PySpark vers une table SQL Server.

    :param df: Le DataFrame PySpark à envoyer
    :param table_name: Le nom de la table dans SQL Server
    :param mode: Le mode d'écriture ('overwrite', 'append', 'ignore', 'error')
    """
    df.write.jdbc(url=jdbc_url_target, table=table_name, mode=mode, properties=propertie
    print(f"Les données ont été envoyées avec succès à la table {table_name} dans la bas


# 1. Analyse des performances de vente
send_to_sql_server(performance_metrics, "sales_performance_analysis")

# 2. Analyse de l'utilisation des données et des revenus
send_to_sql_server(usage_revenue_metrics, "data_usage_revenue_analysis")

# 3. Analyse des tendances d'appels et de la satisfaction client
send_to_sql_server(call_satisfaction_metrics, "call_trends_customer_satisfaction")
```

```
Les données ont été envoyées avec succès à la table sales_performance_analysis dans la b
ase de données spy.
Les données ont été envoyées avec succès à la table data_usage_revenue_analysis dans la
base de données spy.
Les données ont été envoyées avec succès à la table call_trends_customer_satisfaction da
ns la base de données spy.
```

Analyse complète

In [19]:

```python
# Fonction pour charger un DataFrame depuis la base de données source (telecom2)
def load_dataframe(table_name):
    return spark.read.jdbc(url=jdbc_url_source, table=table_name, properties=properties)

# Fonction pour envoyer les données à la base de données cible (spy)
def send_to_sql_server(df, table_name, mode="append"):
    df.write.jdbc(url=jdbc_url_target, table=table_name, mode=mode, properties=propertie
    print(f"Les données ont été envoyées avec succès à la table {table_name} dans la bas

# Test de connexion
print("Test de connexion réussi. Aperçu de la table customer de telecom2 :")
customer_df.show(5)

try:
    # 1. Joindre les tables nécessaires
    comprehensive_analysis = (
        customer_df
        .join(simdata_df, customer_df.CustomerId == simdata_df.SimCustomerId)
        .join(phonenumber_df, simdata_df.SimAccountNumber == phonenumber_df.AccountNumbe
        .join(billinginformation_df, phonenumber_df.PhoneBillNumber == billinginformatio
        .join(plans_df, simdata_df.SimPlanNumber == plans_df.PlansId)
        .join(planinclusions_df, plans_df.PlanInclusionId == planinclusions_df.PlanId)
        .join(orders_df, customer_df.CustomerId == orders_df.OrderCustomerId)
        .join(callrecords_df, phonenumber_df.AccountNumber == callrecords_df.CallAccount
        .join(salesperson_df, customer_df.CustomerSalesPersonId == salesperson_df.SalesP
        .join(employee_df, salesperson_df.IdEmployeeSalesPerson == employee_df.EmployeeI
        .join(department_df, employee_df.DepartmentId == department_df.DepartmentId)
    )

    # 2. Calculer les métriques
    comprehensive_metrics = (
        comprehensive_analysis
        .groupBy("CustomerId", "CustomerName", "Employee_Name", "DepartmentName", "Plans
        .agg(
            count("OrderId").alias("TotalOrders"),
            sum(when(col("OrderStatus") == "Shipped", 1).otherwise(0)).alias("CompletedO
            round(avg("DataUsed"), 2).alias("AvgDataUsed"),
            round(sum("Tax"), 2).alias("TotalRevenue"),
            count("CallId").alias("TotalCalls"),
            round(avg(col("CallDuration").cast("long") / 60), 2).alias("AvgCallDurationM
            round(avg(col("DataUsed") / col("IncludedData")), 2).alias("DataUsageRate")
        )
        .withColumn("CustomerSatisfactionScore",
                    when(col("CompletedOrders") / col("TotalOrders") > 0.8, "High")
                    .when(col("CompletedOrders") / col("TotalOrders") > 0.5, "Medium")
                    .otherwise("Low"))
        .withColumn("RevenuePerCall", round(col("TotalRevenue") / col("TotalCalls"), 2))
        .withColumn("RevenuePerDataUnit", round(col("TotalRevenue") / col("AvgDataUsed")
    )

    # 3. Ajouter des indicateurs de performance
    final_analysis = comprehensive_metrics.withColumn(
        "PerformanceIndicator",
```

```
        when((col("CustomerSatisfactionScore") == "High") & (col("RevenuePerDataUnit") >
        .when((col("CustomerSatisfactionScore") == "High") | (col("RevenuePerDataUnit")
        .when((col("CustomerSatisfactionScore") == "Low") & (col("RevenuePerDataUnit") <
        .otherwise("Average")
    )

    # Afficher les résultats
    print("Analyse complète :")
    final_analysis.show(truncate=False)

    # 4. Envoyer les résultats à la base de données spy
    send_to_sql_server(final_analysis, "comprehensive_telecom_analysis")

except Exception as e:
    print("Erreur lors de l'analyse :", str(e))
```

```
Test de connexion réussi. Aperçu de la table customer de telecom2 :
+----------+----------------+---+---+-----------+--------------------+------------------
---+
|CustomerId|    CustomerName|Sex|Age|DateOfBirth|SocialSecurityNumber|CustomerSalesPerso
nId|
+----------+----------------+---+---+-----------+--------------------+------------------
---+
|         1|Jishnu Vasudevan|  M| 24| 1993-12-28|           232498675|
1|
|         2|      Harsh Shah|  M| 24| 1993-09-12|           456498675|
2|
|         3| Rachana Rambhad|  F| 24| 1993-08-19|           543498675|
3|
|         4|     Lagan Gupta|  F| 24| 1993-08-08|           765498675|
4|
|         5|      Neha Verma|  F| 24| 1993-08-27|           987498675|
1|
+----------+----------------+---+---+-----------+--------------------+------------------
---+
only showing top 5 rows

Analyse complète :
+----------+-----------------+---------------+----------------------+---------+---------
---------+-----------+---------------+-----------+------------+----------+--------------
--------+-------------+-------------------------+--------------+------------------+-----
---------------+
|CustomerId|CustomerName     |Employee_Name  |DepartmentName        |PlansType|PlanName 
|TotalOrders|CompletedOrders|AvgDataUsed|TotalRevenue|TotalCalls|AvgCallDurationMinutes|
DataUsageRate|CustomerSatisfactionScore|RevenuePerCall|RevenuePerDataUnit|PerformanceInd
icator|
+----------+-----------------+---------------+----------------------+---------+---------
---------+-----------+---------------+-----------+------------+----------+--------------
--------+-------------+-------------------------+--------------+------------------+-----
---------------+
|15        |Sameer Goel      |Chetan Mistry  |Human Resource        |Prepaid  |Continuou
s Texting|2          |0              |300.0      |120.00      |2         |46.39
|0.38         |Low                      |60.00         |0.4               |Poor
|
|13        |Neeraj Rajput    |Mihir Patil    |Information Technology|Postpaid |Do not di
sturb    |4          |0              |100.0      |200.00      |4         |24.23
|0.2          |Low                      |50.00         |2.0               |Poor
|
|5         |Neha Verma       |Mihir Patil    |Information Technology|Prepaid  |Talk For 
Hours    |1          |0              |800.0      |180.00      |1         |7.48
|0.53         |Low                      |180.00        |0.23              |Poor
|
|2         |Harsh Shah       |Karan Thevar   |Human Resource        |Postpaid |Family
|1          |0              |500.0      |170.00      |1         |33.0                  |
0.25         |Low                      |170.00        |0.34              |Poor
|
|14        |Shruti Mehta     |Karan Thevar   |Human Resource        |Postpaid |Family
|2          |2              |100.0      |140.00      |2         |15.94                 |
0.1          |High                     |70.00         |1.4               |Good
|
|16        |Vijayshree Uppili|Shantanu Sawant|Human Resource        |Prepaid  |Basic Pla
n        |8          |0              |200.0      |322.40      |8         |69.41
|0.4          |Low                      |40.30         |1.61              |Poor
|
|1         |Jishnu Vasudevan |Mihir Patil    |Information Technology|Postpaid |Finger ti
ps       |1          |1              |400.0      |100.00      |1         |5.07
|0.33         |High                     |100.00        |0.25              |Good
|
+----------+-----------------+---------------+----------------------+---------+---------
---------+-----------+---------------+-----------+------------+----------+--------------
```

```
--------+------------+-----------------------+-------------+----------------+-----
---------------+

Les données ont été envoyées avec succès à la table comprehensive_telecom_analysis dans
la base de données spy.
```

envoyer les résultats en format csv

In [20]:

```python
import os
import csv
from pyspark.sql import SparkSession

# Configurer HADOOP_HOME (ajustez le chemin si nécessaire)
os.environ['HADOOP_HOME'] = r"C:\hadoop"
os.environ['PATH'] = r"C:\hadoop\bin;" + os.environ['PATH']


# Charger la table employee depuis SQL Server
query = "(SELECT * FROM employee) as employee_data"
df_employee = spark.read.jdbc(url=jdbc_url_source, table=query, properties=properties)

# Afficher le schéma et quelques lignes pour vérification
df_employee.printSchema()
df_employee.show(5)

# Convertir le DataFrame en RDD et collecter les résultats
results = df_employee.rdd.collect()

# Chemin de sortie spécifié
output_path = r"C:\Users\ELITEBOOK\Desktop\stage\jupy\employee_dataa1.csv"

def write_csv(path):
    try:
        # Créer le répertoire parent si nécessaire
        os.makedirs(os.path.dirname(path), exist_ok=True)

        with open(path, 'w', newline='') as csvfile:
            writer = csv.writer(csvfile)
            writer.writerow(df_employee.columns)
            for row in results:
                writer.writerow(row)
        print(f"Résultats sauvegardés dans {path}")
        return True
    except PermissionError:
        print(f"Erreur de permission pour {path}. Essayez d'exécuter le script en tant q
        return False
    except Exception as e:
        print(f"Erreur lors de l'écriture dans {path}: {str(e)}")
        return False

# Écrire le fichier CSV
write_csv(output_path)
```

Loading [MathJax]/extensions/Safe.js

```
root
 |-- EmployeeId: integer (nullable = true)
 |-- Employee_Name: string (nullable = true)
 |-- SSN: string (nullable = true)
 |-- Age: integer (nullable = true)
 |-- DepartmentId: integer (nullable = true)
 |-- Salary: integer (nullable = true)

+----------+--------------------+---------+---+------------+------+
|EmployeeId|       Employee_Name|      SSN|Age|DepartmentId|Salary|
+----------+--------------------+---------+---+------------+------+
|         1|      Ojas Phansekar|123456789| 24|           1|  1000|
|         2|Shreyas Kalayanar...|245987675| 24|           1|  1000|
|         3|    Saurabh Kulkarni|734756953| 24|           1|  1000|
|         4|        Vivek Shetye|572364526| 26|           1|  1000|
|         5|         Mihir Patil|238745784| 27|           1|  1000|
+----------+--------------------+---------+---+------------+------+
only showing top 5 rows

Résultats sauvegardés dans C:\Users\ELITEBOOK\Desktop\stage\jupy\employee_dataa1.csv
```

Out[20]:
```
True
```

In [21]:
```python
import os
import csv
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, when, avg, round, lit, concat, substring

# Configurer HADOOP_HOME (ajustez le chemin si nécessaire)
os.environ['HADOOP_HOME'] = r"C:\hadoop"
os.environ['PATH'] = r"C:\hadoop\bin;" + os.environ['PATH']



# Charger la table employee depuis SQL Server
query = "(SELECT * FROM employee) as employee_data"
df_employee = spark.read.jdbc(url=jdbc_url_source, table=query, properties=properties)

# Afficher le schéma initial
print("Schéma initial:")
df_employee.printSchema()

# Transformations
df_transformed = df_employee \
    .withColumn("SalaryCategory", when(col("Salary") < 5000, "Low")
                                  .when((col("Salary") >= 5000) & (col("Salary") < 10000)
                                  .otherwise("High")) \
    .withColumn("AdjustedSalary", round(col("Salary") * 1.1, 2)) \
    .withColumn("AgeBracket", when(col("Age") < 30, "Young")
                              .when((col("Age") >= 30) & (col("Age") < 50), "Middle-aged
                              .otherwise("Senior")) \
    .withColumn("MaskedSSN", concat(substring(col("SSN"), 1, 3), lit("******"))) \
    .withColumn("FullName", concat(col("Employee_Name"), lit(" (ID: "), col("EmployeeId"
    .drop("SSN")  # Supprimer la colonne SSN originale pour des raisons de confidentiali

# Calculer le salaire moyen par département
avg_salary_by_dept = df_transformed.groupBy("DepartmentId").agg(round(avg("Salary"), 2).

# Joindre le salaire moyen du département
df_final = df_transformed.join(avg_salary_by_dept, "DepartmentId")

# Afficher le nouveau schéma et quelques lignes
print("\nNouveau schéma après transformations:")
df_final.printSchema()
rçu des données transformées:")
```

```python
df_final.show(5, truncate=False)

# Convertir le DataFrame en RDD et collecter les résultats
results = df_final.rdd.collect()

# Chemin de sortie spécifié
output_path = r"C:\Users\ELITEBOOK\Desktop\stage\jupy\employee_data_transformed11.csv"

def write_csv(path, data, columns):
    try:
        os.makedirs(os.path.dirname(path), exist_ok=True)
        with open(path, 'w', newline='') as csvfile:
            writer = csv.writer(csvfile)
            writer.writerow(columns)
            for row in data:
                writer.writerow(row)
        print(f"Résultats transformés sauvegardés dans {path}")
        return True
    except PermissionError:
        print(f"Erreur de permission pour {path}. Essayez d'exécuter le script en tant q
        return False
    except Exception as e:
        print(f"Erreur lors de l'écriture dans {path}: {str(e)}")
        return False

# Écrire le fichier CSV
write_csv(output_path, results, df_final.columns)
```

```
Schéma initial:
root
 |-- EmployeeId: integer (nullable = true)
 |-- Employee_Name: string (nullable = true)
 |-- SSN: string (nullable = true)
 |-- Age: integer (nullable = true)
 |-- DepartmentId: integer (nullable = true)
 |-- Salary: integer (nullable = true)


Nouveau schéma après transformations:
root
 |-- DepartmentId: integer (nullable = true)
 |-- EmployeeId: integer (nullable = true)
 |-- Employee_Name: string (nullable = true)
 |-- Age: integer (nullable = true)
 |-- Salary: integer (nullable = true)
 |-- SalaryCategory: string (nullable = false)
 |-- AdjustedSalary: double (nullable = true)
 |-- AgeBracket: string (nullable = false)
 |-- MaskedSSN: string (nullable = true)
 |-- FullName: string (nullable = true)
 |-- AvgDeptSalary: double (nullable = true)


Aperçu des données transformées:
+------------+----------+---------------------+---+------+--------------+--------------+
----------+---------+-----------------------------+-------------+
|DepartmentId|EmployeeId|Employee_Name        |Age|Salary|SalaryCategory|AdjustedSalary|
AgeBracket|MaskedSSN|FullName                     |AvgDeptSalary|
+------------+----------+---------------------+---+------+--------------+--------------+
----------+---------+-----------------------------+-------------+
|1           |1         |Ojas Phansekar       |24 |1000  |Low           |1100.0        |
Young     |123******|Ojas Phansekar (ID: 1)       |1000.0       |
|1           |2         |Shreyas Kalayanaraman|24 |1000  |Low           |1100.0        |
Young     |245******|Shreyas Kalayanaraman (ID: 2)|1000.0       |
|1           |3         |Saurabh Kulkarni     |24 |1000  |Low           |1100.0        |
Young     |734******|Saurabh Kulkarni (ID: 3)     |1000.0       |
|1           |4         |Vivek Shetye         |26 |1000  |Low           |1100.0        |
Young     |572******|Vivek Shetye (ID: 4)         |1000.0       |
|1           |5         |Mihir Patil          |27 |1000  |Low           |1100.0        |
Young     |238******|Mihir Patil (ID: 5)          |1000.0       |
+------------+----------+---------------------+---+------+--------------+--------------+
----------+---------+-----------------------------+-------------+
only showing top 5 rows

Résultats transformés sauvegardés dans C:\Users\ELITEBOOK\Desktop\stage\jupy\employee_da
ta_transformed11.csv
```

Out[21]:

```
True
```

envoyer les résultats en format JSON

In [22]:

```python
import os
import json
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, when, avg, round, lit, concat, substring, to_json


query = "(SELECT * FROM employee) as employee_data"
df_employee = spark.read.jdbc(url=jdbc_url_source, table=query, properties=properties)

 initial:")
```

```
df_employee.printSchema()

# Transformations
df_transformed = df_employee \
    .withColumn("performance_score", (col("Salary") / 1000 + col("Age") / 10).cast("int"
    .withColumn("experience_level", when(col("Age") < 25, "Junior")
                                   .when((col("Age") >= 25) & (col("Age") < 35), "Interm
                                   .when((col("Age") >= 35) & (col("Age") < 45), "Senior
                                   .otherwise("Expert")) \
    .withColumn("salary_bracket", when(col("Salary") < 3000, "Entry")
                                  .when((col("Salary") >= 3000) & (col("Salary") < 6000)
                                  .when((col("Salary") >= 6000) & (col("Salary") < 9000)
                                  .otherwise("Executive")) \
    .withColumn("department_size", when(col("DepartmentId").isin([1, 2]), "Small")
                                   .when(col("DepartmentId").isin([3, 4]), "Medium")
                                   .otherwise("Large")) \
    .withColumn("employee_code", concat(substring(col("Employee_Name"), 1, 3), lit("-"),

# Calculer des statistiques par département
dept_stats = df_transformed.groupBy("DepartmentId") \
    .agg(round(avg("Salary"), 2).alias("avg_salary"),
         round(avg("Age"), 2).alias("avg_age"))

# Joindre les statistiques du département
df_final = df_transformed.join(dept_stats, "DepartmentId")

# Afficher le nouveau schéma et quelques lignes
print("\nNouveau schéma après transformations:")
df_final.printSchema()
print("\nAperçu des données transformées:")
df_final.show(5, truncate=False)

# Chemin de sortie spécifié
output_path = r"C:\Users\ELITEBOOK\Desktop\stage\jupy\employee_data2_tansformed11.json"

# Écriture manuelle du JSON
try:
    data = df_final.toJSON().collect()
    with open(output_path, 'w') as f:
        json.dump(data, f)
    print(f"Résultats transformés sauvegardés en JSON dans {output_path}")
except Exception as e:
    print(f"Erreur lors de l'écriture du JSON: {str(e)}")
```

```
Schéma initial:
root
 |-- EmployeeId: integer (nullable = true)
 |-- Employee_Name: string (nullable = true)
 |-- SSN: string (nullable = true)
 |-- Age: integer (nullable = true)
 |-- DepartmentId: integer (nullable = true)
 |-- Salary: integer (nullable = true)


Nouveau schéma après transformations:
root
 |-- DepartmentId: integer (nullable = true)
 |-- EmployeeId: integer (nullable = true)
 |-- Employee_Name: string (nullable = true)
 |-- SSN: string (nullable = true)
 |-- Age: integer (nullable = true)
 |-- Salary: integer (nullable = true)
 |-- performance_score: integer (nullable = true)
 |-- experience_level: string (nullable = false)
 |-- salary_bracket: string (nullable = false)
 |-- department_size: string (nullable = false)
 |-- employee_code: string (nullable = true)
 |-- avg_salary: double (nullable = true)
 |-- avg_age: double (nullable = true)


Aperçu des données transformées:
+------------+----------+---------------------+---------+---+------+-----------------+--
--------------+--------------+----------------+-------------+----------+-------+
|DepartmentId|EmployeeId|Employee_Name        |SSN      |Age|Salary|performance_score|ex
perience_level|salary_bracket|department_size|employee_code|avg_salary|avg_age|
+------------+----------+---------------------+---------+---+------+-----------------+--
--------------+--------------+----------------+-------------+----------+-------+
|1           |1         |Ojas Phansekar       |123456789|24 |1000  |3                |Ju
nior          |Entry         |Small           |Oja-1        |1000.0    |25.57  |
|1           |2         |Shreyas Kalayanaraman|245987675|24 |1000  |3                |Ju
nior          |Entry         |Small           |Shr-2        |1000.0    |25.57  |
|1           |3         |Saurabh Kulkarni     |734756953|24 |1000  |3                |Ju
nior          |Entry         |Small           |Sau-3        |1000.0    |25.57  |
|1           |4         |Vivek Shetye         |572364526|26 |1000  |3                |In
termediate    |Entry         |Small           |Viv-4        |1000.0    |25.57  |
|1           |5         |Mihir Patil          |238745784|27 |1000  |3                |In
termediate    |Entry         |Small           |Mih-5        |1000.0    |25.57  |
+------------+----------+---------------------+---------+---+------+-----------------+--
--------------+--------------+----------------+-------------+----------+-------+
only showing top 5 rows

Résultats transformés sauvegardés en JSON dans C:\Users\ELITEBOOK\Desktop\stage\jupy\emp
loyee_data2_tansformed11.json
```

analyse de la rétention des clients

In [23]:
```python
from pyspark.sql.functions import col, lag, datediff, current_date, count, avg, to_date,
from pyspark.sql.window import Window

# Charger les données
orders_df = spark.read.jdbc(url=jdbc_url_source, table="orders", properties=properties)
customer_df = spark.read.jdbc(url=jdbc_url_source, table="customer", properties=properti

# Ajouter une colonne de date à orders_df
orders_with_date = orders_df.withColumn("OrderDate", to_date(from_unixtime(col("OrderId"
```

```python
# Analyse de la rétention des clients
window_spec = Window.partitionBy("OrderCustomerId").orderBy("OrderId")

customer_retention = orders_with_date.withColumn("PreviousOrderDate", lag("OrderDate").o
    .withColumn("DaysSinceLastOrder", datediff(col("OrderDate"), col("PreviousOrderDate"
    .groupBy("OrderCustomerId") \
    .agg(
        count("OrderId").alias("TotalOrders"),
        avg("DaysSinceLastOrder").alias("AvgDaysBetweenOrders")
    ) \
    .join(customer_df, orders_df.OrderCustomerId == customer_df.CustomerId) \
    .withColumn("CustomerLifetime", datediff(current_date(), col("DateOfBirth")))

print("Analyse de la rétention des clients:")
customer_retention.show()

# Quelques statistiques supplémentaires
print("\nStatistiques globales:")
customer_retention.select(
    avg("TotalOrders").alias("AvgOrdersPerCustomer"),
    avg("AvgDaysBetweenOrders").alias("OverallAvgDaysBetweenOrders"),
    avg("CustomerLifetime").alias("AvgCustomerLifetime")
).show()
```

```
Analyse de la rétention des clients:
+---------------+-----------+--------------------+----------+-----------------+---+---+-
----------+--------------------+---------------------+----------------+
|OrderCustomerId|TotalOrders|AvgDaysBetweenOrders|CustomerId|     CustomerName|Sex|Age|D
ateOfBirth|SocialSecurityNumber|CustomerSalesPersonId|CustomerLifetime|
+---------------+-----------+--------------------+----------+-----------------+---+---+-
----------+--------------------+---------------------+----------------+
|             12|          1|                NULL|        12|      Kal Bugrara|  M| 57| 
1960-12-28|           145498675|                    4|           23253|
|              1|          1|                NULL|         1| Jishnu Vasudevan|  M| 24| 
1993-12-28|           232498675|                    1|           11200|
|             13|          1|                NULL|        13|    Neeraj Rajput|  M| 27| 
1990-10-28|           232555675|                    1|           12357|
|              6|          1|                NULL|         6|      Aniel Patel|  M| 24| 
1993-11-28|           235468675|                    2|           11230|
|             16|          1|                NULL|        16|Vijayshree Uppili|  F| 26| 
1991-08-23|           654498675|                    4|           12058|
|              3|          1|                NULL|         3|  Rachana Rambhad|  F| 24| 
1993-08-19|           543498675|                    3|           11331|
|             20|          1|                NULL|        20|      Simmah Kazi|  F| 22| 
1995-12-28|           232834675|                    4|           10470|
|              5|          1|                NULL|         5|       Neha Verma|  F| 24| 
1993-08-27|           987498675|                    1|           11323|
|             19|          1|                NULL|        19|  Komal Shirodkar|  F| 26| 
1991-02-27|           678498675|                    3|           12235|
|             15|          1|                NULL|        15|      Sameer Goel|  M| 28| 
1989-12-30|           276578675|                    3|           12659|
|              9|          1|                NULL|         9|     Parnal Dighe|  F| 24| 
1993-09-28|           232498765|                    1|           11291|
|             17|          1|                NULL|        17|     Rohit Kamble|  M| 24| 
1993-06-28|           453498675|                    1|           11383|
|              4|          1|                NULL|         4|      Lagan Gupta|  F| 24| 
1993-08-08|           765498675|                    4|           11342|
|              8|          1|                NULL|         8|     Aditya Joshi|  M| 24| 
1993-10-28|           232434575|                    4|           11261|
|              7|          1|                NULL|         7|    Anubhav Gupta|  M| 27| 
1990-12-28|           555698675|                    3|           12296|
|             10|          1|                NULL|        10|      Dharit Shah|  M| 24| 
1993-12-27|           123498675|                    2|           11201|
|             11|          1|                NULL|        11|     Girish Sanai|  M| 24| 
1993-07-22|           645498675|                    3|           11359|
|             14|          1|                NULL|        14|     Shruti Mehta|  F| 26| 
1991-12-17|           232444375|                    2|           11942|
|              2|          1|                NULL|         2|       Harsh Shah|  M| 24| 
1993-09-12|           456498675|                    2|           11307|
|             18|          1|                NULL|        18|   Priyanka Desai|  F| 26| 
1991-04-23|           189498675|                    2|           12180|
+---------------+-----------+--------------------+----------+-----------------+---+---+-
----------+--------------------+---------------------+----------------+


Statistiques globales:
+--------------------+---------------------------+-------------------+
|AvgOrdersPerCustomer|OverallAvgDaysBetweenOrders|AvgCustomerLifetime|
+--------------------+---------------------------+-------------------+
|                 1.0|                       NULL|            12183.9|
+--------------------+---------------------------+-------------------+
```

Analyse de la performance des vendeurs

In [24]:

```
salesperson_performance = customer_df.join(salesperson_df, customer_df.CustomerSalesPers
    .join(orders_df, customer_df.CustomerId == orders_df.OrderCustomerId) \
    .join(employee_df, salesperson_df.IdEmployeeSalesPerson == employee_df.EmployeeId) \
    .groupBy("IdEmployeeSalesPerson", "Employee_Name") \
    .agg(
        count("CustomerId").alias("TotalCustomers"),
        count("OrderId").alias("TotalOrders"),
        sum(when(col("OrderStatus") == "Shipped", 1).otherwise(0)).alias("CompletedOrder
    ) \
    .withColumn("OrderCompletionRate", round(col("CompletedOrders") / col("TotalOrders")


print("Performance des vendeurs:")
salesperson_performance.show()
```

```
Performance des vendeurs:
+---------------------+--------------+--------------+-----------+---------------+------
-------------+
|IdEmployeeSalesPerson|  Employee_Name|TotalCustomers|TotalOrders|CompletedOrders|OrderC
ompletionRate|
+---------------------+--------------+--------------+-----------+---------------+------
-------------+
|                    5|   Mihir Patil|             5|          5|              1|
0.2|
|                    6|  Karan Thevar|             5|          5|              2|
0.4|
|                    7| Chetan Mistry|             5|          5|              0|
0.0|
|                    8|Shantanu Sawant|             5|          5|              0|
0.0|
+---------------------+--------------+--------------+-----------+---------------+------
-------------+
```

Analyse des salaires par département

In [26]:

```
from pyspark.sql.functions import avg, count, desc, col

salary_analysis = employee_df.join(department_df, "DepartmentId") \
    .groupBy("DepartmentName") \
    .agg(
        avg(employee_df.Salary).alias("AvgSalary"),
        count("EmployeeId").alias("EmployeeCount")
    ) \
    .orderBy(desc("AvgSalary"))

print("Analyse des salaires par département:")
salary_analysis.show()
```

```
Analyse des salaires par département:
+--------------------+---------+-------------+
|      DepartmentName|AvgSalary|EmployeeCount|
+--------------------+---------+-------------+
|       Customer Care|  10000.0|           10|
|      Human Resource|   7500.0|            7|
|   Sales & Marketing|   5000.0|            6|
|             Finance|   2500.0|            7|
|Information Techn...|   1000.0|            7|
+--------------------+---------+-------------+
```

Catégorisation des employés par âge

In [27]:
```
from pyspark.sql.functions import col, expr, when, concat, lit, datediff, current_date,
from pyspark.sql.window import Window
from pyspark.sql.functions import col, when

def display_df(df, n=10):
    return df.limit(n).toPandas()

employee_category = employee_df.withColumn(
    "age_category",
    when(col("Age") < 30, "Junior")
    .when((col("Age") >= 30) & (col("Age") < 45), "Mid-level")
    .when(col("Age") >= 45, "Senior")
    .otherwise("Unknown")
)

print("Catégorisation des employés par âge:")
print(display_df(employee_category))
```

```
Catégorisation des employés par âge:
   EmployeeId          Employee_Name        SSN  Age  DepartmentId  Salary  \
0           1         Ojas Phansekar  123456789   24             1    1000
1           2  Shreyas Kalayanaraman  245987675   24             1    1000
2           3       Saurabh Kulkarni  734756953   24             1    1000
3           4           Vivek Shetye  572364526   26             1    1000
4           5            Mihir Patil  238745784   27             1    1000
5           6           Karan Thevar  968374657   28             4    7500
6           7          Chetan Mistry  623784983   30             4    7500
7           8         Shantanu Sawant  527473298   24             4    7500
8           9            Pooja Patil  286436778   24             4    7500
9          10      Kalpita Malvankar  863476236   34             4    7500

  age_category
0       Junior
1       Junior
2       Junior
3       Junior
4       Junior
5       Junior
6    Mid-level
7       Junior
8       Junior
9    Mid-level
```

Classement des employés par salaire dans chaque département

In [30]:
```
salary_ranking = employee_df.withColumn(
    "SalaryRank",
    rank().over(Window.partitionBy("DepartmentId").orderBy(col("Salary").desc()))
)

print("\n3. Classement des employés par salaire dans chaque département:")
print(display_df(salary_ranking))
```

```
3. Classement des employés par salaire dans chaque département:
   EmployeeId           Employee_Name        SSN  Age  DepartmentId  Salary  \
0           1          Ojas Phansekar  123456789   24             1    1000
1           2  Shreyas Kalayanaraman  245987675   24             1    1000
2           3       Saurabh Kulkarni  734756953   24             1    1000
3           4           Vivek Shetye  572364526   26             1    1000
4           5            Mihir Patil  238745784   27             1    1000
5          32         Manoj Prabhakar  444787654   21             1    1000
6          34            Priya Yadav  228787654   33             1    1000
7          11          Vaibhav Parkar  123456789   24             2    5000
8          12        Sayali Sakhalkar  674378987   24             2    5000
9          13           Khushi Chavan  652134897   45             2    5000

   SalaryRank
0           1
1           1
2           1
3           1
4           1
5           1
6           1
7           1
8           1
9           1
```

Identification des employés les mieux payés par département (Top 3)

In [31]:

```
top_earners = employee_df.withColumn(
    "SalaryRank",
    dense_rank().over(Window.partitionBy("DepartmentId").orderBy(col("Salary").desc()))
).filter(col("SalaryRank") <= 3)

print("\n5. Top 3 des employés les mieux payés par département:")
print(display_df(top_earners))
```

```
5. Top 3 des employés les mieux payés par département:
   EmployeeId           Employee_Name        SSN  Age  DepartmentId  Salary  \
0           1          Ojas Phansekar  123456789   24             1    1000
1           2  Shreyas Kalayanaraman  245987675   24             1    1000
2           3       Saurabh Kulkarni  734756953   24             1    1000
3           4           Vivek Shetye  572364526   26             1    1000
4           5            Mihir Patil  238745784   27             1    1000
5          32         Manoj Prabhakar  444787654   21             1    1000
6          34            Priya Yadav  228787654   33             1    1000
7          11          Vaibhav Parkar  123456789   24             2    5000
8          12        Sayali Sakhalkar  674378987   24             2    5000
9          13           Khushi Chavan  652134897   45             2    5000

   SalaryRank
0           1
1           1
2           1
3           1
4           1
5           1
6           1
7           1
8           1
9           1
```

Analyse de la distribution des âges

In [32]:
```
age_distribution = employee_df.withColumn(
    "AgeGroup",
    when(col("Age") < 25, "18-24")
    .when((col("Age") >= 25) & (col("Age") < 35), "25-34")
    .when((col("Age") >= 35) & (col("Age") < 45), "35-44")
    .when((col("Age") >= 45) & (col("Age") < 55), "45-54")
    .otherwise("55+")
).groupBy("AgeGroup").count().orderBy("AgeGroup")

print("\n6. Distribution des âges:")
print(display_df(age_distribution))
```

```
6. Distribution des âges:
  AgeGroup  count
0    18-24     13
1    25-34      9
2    35-44      3
3    45-54      4
4      55+      8
```

Analyse des employés par tranche de salaire

In [33]:
```
salary_brackets = employee_df.withColumn(
    "SalaryBracket",
    when(col("Salary") < 3000, "Low")
    .when((col("Salary") >= 3000) & (col("Salary") < 6000), "Medium")
    .when((col("Salary") >= 6000) & (col("Salary") < 9000), "High")
    .otherwise("Very High")
).groupBy("SalaryBracket").count().orderBy("SalaryBracket")

print("\n8. Analyse des employés par tranche de salaire:")
print(display_df(salary_brackets))
```

```
8. Analyse des employés par tranche de salaire:
  SalaryBracket  count
0          High      7
1           Low     14
2        Medium      6
3     Very High     10
```

Analyse du nombre d'employés par département

In [34]:
```
employees_per_dept = employee_df.join(department_df, "DepartmentId") \
    .groupBy("DepartmentName") \
    .agg(count("EmployeeId").alias("EmployeeCount")) \
    .orderBy(col("EmployeeCount").desc())

print("\n9. Nombre d'employés par département:")
print(display_df(employees_per_dept))
```

```
9. Nombre d'employés par département:
           DepartmentName  EmployeeCount
0            Customer Care             10
1  Information Technology              7
2                  Finance              7
3           Human Resource              7
4        Sales & Marketing              6
```

Calcul de la masse salariale totale par département

In [35]:
```
from pyspark.sql.functions import col, sum as sum_

total_salary_by_dept = employee_df.join(department_df, "DepartmentId") \
    .groupBy(department_df.DepartmentName) \
    .agg(sum_(employee_df.Salary).alias("TotalSalary")) \
    .orderBy(col("TotalSalary").desc())

print("\n10. Masse salariale totale par département:")
print(display_df(total_salary_by_dept))
```

```
10. Masse salariale totale par département:
           DepartmentName  TotalSalary
0            Customer Care       100000
1           Human Resource        52500
2        Sales & Marketing        30000
3                  Finance        17500
4  Information Technology          7000
```

normalisation des noms d'employés

In [36]:
```
from pyspark.sql.functions import col, when, regexp_replace, trim, lower, upper, to_date
from pyspark.sql.window import Window

cleaned_employee_df = employee_df.withColumn(
    "CleanedName",
    trim(regexp_replace(lower(col("Employee_Name")), r'[^\w\s]', ''))
)
cleaned_employee_df.show()
```

```
+----------+------------------+---------+---+------------+------+-------------------+
|EmployeeId|     Employee_Name|      SSN|Age|DepartmentId|Salary|        CleanedName|
+----------+------------------+---------+---+------------+------+-------------------+
|         1|    Ojas Phansekar|123456789| 24|           1|  1000|     ojas phansekar|
|         2|Shreyas Kalayanar...|245987675| 24|           1|  1000|shreyas kalayanar...|
|         3|   Saurabh Kulkarni|734756953| 24|           1|  1000|   saurabh kulkarni|
|         4|       Vivek Shetye|572364526| 26|           1|  1000|       vivek shetye|
|         5|        Mihir Patil|238745784| 27|           1|  1000|        mihir patil|
|         6|       Karan Thevar|968374657| 28|           4|  7500|       karan thevar|
|         7|      Chetan Mistry|623784983| 30|           4|  7500|      chetan mistry|
|         8|     Shantanu Sawant|527473298| 24|           4|  7500|     shantanu sawant|
|         9|        Pooja Patil|286436778| 24|           4|  7500|        pooja patil|
|        10|   Kalpita Malvankar|863476236| 34|           4|  7500|   kalpita malvankar|
|        11|     Vaibhav Parkar|123456789| 24|           2|  5000|     vaibhav parkar|
|        12|    Sayali Sakhalkar|674378987| 24|           2|  5000|    sayali sakhalkar|
|        13|      Khushi Chavan|652134897| 45|           2|  5000|      khushi chavan|
|        14|       Pratik Patre|677435432| 24|           2|  5000|       pratik patre|
|        15|            Pushkar|564321879| 43|           2|  5000|            pushkar|
|        16|       Tushar Gupta|444777651| 24|           5| 10000|       tushar gupta|
|        17|  Pranav Swaminathan|990077663| 34|           3|  2500|  pranav swaminathan|
|        18|             Victor|563477778| 44|           3|  2500|             victor|
|        19|        Yusuf Ozbek|995912563| 45|           3|  2500|        yusuf ozbek|
|        20|  Sudharshan Poojary|763459876| 24|           3|  2500|  sudharshan poojary|
+----------+------------------+---------+---+------------+------+-------------------+
only showing top 20 rows
```

Gestion des valeurs manquantes dans la colonne Salary

In [37]:

```python
salary_stats = employee_df.agg(
    avg("Salary").alias("avg_salary"),
    stddev("Salary").alias("stddev_salary")
)
avg_salary = salary_stats.collect()[0]["avg_salary"]
stddev_salary = salary_stats.collect()[0]["stddev_salary"]

imputed_salary_df = employee_df.withColumn(
    "ImputedSalary",
    when(col("Salary").isNull(),
         when(col("Age") < 30, avg_salary - stddev_salary)
         .when(col("Age") >= 30, avg_salary + stddev_salary)
         .otherwise(avg_salary)
    ).otherwise(col("Salary"))
)
print("\nSalaires imputés :")
imputed_salary_df.select("EmployeeId", "Salary", "ImputedSalary").show(5)
```

```
Salaires imputés :
+----------+------+-------------+
|EmployeeId|Salary|ImputedSalary|
+----------+------+-------------+
|         1|  1000|       1000.0|
|         2|  1000|       1000.0|
|         3|  1000|       1000.0|
|         4|  1000|       1000.0|
|         5|  1000|       1000.0|
+----------+------+-------------+
only showing top 5 rows
```

Détection des valeurs aberrantes de salaire

In [38]:

```
salary_outliers_df = imputed_salary_df.withColumn(
    "IsSalaryOutlier",
    abs(col("ImputedSalary") - avg_salary) > (3 * stddev_salary)
)
print("\nDétection des valeurs aberrantes de salaire :")
salary_outliers_df.select("EmployeeId", "ImputedSalary", "IsSalaryOutlier").show(5)
```

```
Détection des valeurs aberrantes de salaire :
+----------+-------------+---------------+
|EmployeeId|ImputedSalary|IsSalaryOutlier|
+----------+-------------+---------------+
|         1|       1000.0|          false|
|         2|       1000.0|          false|
|         3|       1000.0|          false|
|         4|       1000.0|          false|
|         5|       1000.0|          false|
+----------+-------------+---------------+
only showing top 5 rows
```

Gestion des valeurs vides dans la colonne Salary

In [39]:

```
from pyspark.sql.functions import col, when, regexp_replace, trim, lower, length, isnan,
from pyspark.sql.window import Window

salary_stats = employee_df.agg(
    avg("Salary").alias("avg_salary"),
    stddev("Salary").alias("stddev_salary")
)
avg_salary = salary_stats.collect()[0]["avg_salary"]
stddev_salary = salary_stats.collect()[0]["stddev_salary"]

df_with_imputed_salary = employee_df.withColumn(
    "ImputedSalary",
    when(col("Salary").isNull() | isnan("Salary"),
         when(col("Age") < 35, lit(avg_salary - stddev_salary))
         .when(col("Age") >= 35, lit(avg_salary + stddev_salary))
         .otherwise(lit(avg_salary))
    ).otherwise(col("Salary"))
)
df_with_imputed_salary.show()
```

```
+----------+------------------+---------+---+------------+------+-------------+
|EmployeeId|     Employee_Name|      SSN|Age|DepartmentId|Salary|ImputedSalary|
+----------+------------------+---------+---+------------+------+-------------+
|         1|    Ojas Phansekar|123456789| 24|           1|  1000|       1000.0|
|         2|Shreyas Kalayanar...|245987675| 24|           1|  1000|       1000.0|
|         3|  Saurabh Kulkarni|734756953| 24|           1|  1000|       1000.0|
|         4|      Vivek Shetye|572364526| 26|           1|  1000|       1000.0|
|         5|       Mihir Patil|238745784| 27|           1|  1000|       1000.0|
|         6|      Karan Thevar|968374657| 28|           4|  7500|       7500.0|
|         7|     Chetan Mistry|623784983| 30|           4|  7500|       7500.0|
|         8|    Shantanu Sawant|527473298| 24|           4|  7500|       7500.0|
|         9|       Pooja Patil|286436778| 24|           4|  7500|       7500.0|
|        10|  Kalpita Malvankar|863476236| 34|           4|  7500|       7500.0|
|        11|    Vaibhav Parkar|123456789| 24|           2|  5000|       5000.0|
|        12|   Sayali Sakhalkar|674378987| 24|           2|  5000|       5000.0|
|        13|     Khushi Chavan|652134897| 45|           2|  5000|       5000.0|
|        14|      Pratik Patre|677435432| 24|           2|  5000|       5000.0|
|        15|           Pushkar|564321879| 43|           2|  5000|       5000.0|
|        16|      Tushar Gupta|444777651| 24|           5| 10000|      10000.0|
|        17| Pranav Swaminathan|990077663| 34|           3|  2500|       2500.0|
|        18|            Victor|563477778| 44|           3|  2500|       2500.0|
|        19|       Yusuf Ozbek|995912563| 45|           3|  2500|       2500.0|
|        20| Sudharshan Poojary|763459876| 24|           3|  2500|       2500.0|
+----------+------------------+---------+---+------------+------+-------------+
only showing top 20 rows
```

Calcul de l'ancienneté

In [40]:

```
from pyspark.sql.functions import col, when, current_date, datediff, round, lit
from pyspark.sql.types import DateType

print("Schéma actuel du DataFrame:")
df_with_imputed_salary.printSchema()

if "HireDate" not in df_with_imputed_salary.columns:
    df_with_hire_date = df_with_imputed_salary.withColumn(
        "HireDate",
        lit("2020-01-01").cast(DateType())  # Date fictive, à ajuster selon vos besoins
    )
else:
    df_with_hire_date = df_with_imputed_salary

df_with_tenure = df_with_hire_date.withColumn(
    "HireDate",
    when(col("HireDate").isNull(), current_date()).otherwise(col("HireDate"))
).withColumn(
    "TenureYears",
    round(datediff(current_date(), col("HireDate")) / 365, 2)
)

print("DataFrame avec ancienneté calculée:")
df_with_tenure.show()
```

```
Schéma actuel du DataFrame:
root
 |-- EmployeeId: integer (nullable = true)
 |-- Employee_Name: string (nullable = true)
 |-- SSN: string (nullable = true)
 |-- Age: integer (nullable = true)
 |-- DepartmentId: integer (nullable = true)
 |-- Salary: integer (nullable = true)
 |-- ImputedSalary: double (nullable = true)

DataFrame avec ancienneté calculée:
+----------+--------------------+---------+---+------------+------+-------------+-------
---+-----------+
|EmployeeId|       Employee_Name|      SSN|Age|DepartmentId|Salary|ImputedSalary|  HireD
ate|TenureYears|
+----------+--------------------+---------+---+------------+------+-------------+-------
---+-----------+
|         1|      Ojas Phansekar|123456789| 24|           1|  1000|       1000.0|2020-01
-01|       4.66|
|         2|Shreyas Kalayanar...|245987675| 24|           1|  1000|       1000.0|2020-01
-01|       4.66|
|         3|    Saurabh Kulkarni|734756953| 24|           1|  1000|       1000.0|2020-01
-01|       4.66|
|         4|        Vivek Shetye|572364526| 26|           1|  1000|       1000.0|2020-01
-01|       4.66|
|         5|         Mihir Patil|238745784| 27|           1|  1000|       1000.0|2020-01
-01|       4.66|
|         6|        Karan Thevar|968374657| 28|           4|  7500|       7500.0|2020-01
-01|       4.66|
|         7|       Chetan Mistry|623784983| 30|           4|  7500|       7500.0|2020-01
-01|       4.66|
|         8|     Shantanu Sawant|527473298| 24|           4|  7500|       7500.0|2020-01
-01|       4.66|
|         9|         Pooja Patil|286436778| 24|           4|  7500|       7500.0|2020-01
-01|       4.66|
|        10|   Kalpita Malvankar|863476236| 34|           4|  7500|       7500.0|2020-01
-01|       4.66|
|        11|      Vaibhav Parkar|123456789| 24|           2|  5000|       5000.0|2020-01
-01|       4.66|
|        12|    Sayali Sakhalkar|674378987| 24|           2|  5000|       5000.0|2020-01
-01|       4.66|
|        13|       Khushi Chavan|652134897| 45|           2|  5000|       5000.0|2020-01
-01|       4.66|
|        14|        Pratik Patre|677435432| 24|           2|  5000|       5000.0|2020-01
-01|       4.66|
|        15|             Pushkar|564321879| 43|           2|  5000|       5000.0|2020-01
-01|       4.66|
|        16|        Tushar Gupta|444777651| 24|           5| 10000|      10000.0|2020-01
-01|       4.66|
|        17|  Pranav Swaminathan|990077663| 34|           3|  2500|       2500.0|2020-01
-01|       4.66|
|        18|              Victor|563477778| 44|           3|  2500|       2500.0|2020-01
-01|       4.66|
|        19|         Yusuf Ozbek|995912563| 45|           3|  2500|       2500.0|2020-01
-01|       4.66|
|        20|  Sudharshan Poojary|763459876| 24|           3|  2500|       2500.0|2020-01
-01|       4.66|
+----------+--------------------+---------+---+------------+------+-------------+-------
---+-----------+
only showing top 20 rows
```

Durée moyenne des appels par compte

In [41]:
```
avg_call_duration = callrecords_df \
    .withColumn("DurationSeconds",
                expr("hour(CallDuration) * 3600 + minute(CallDuration) * 60 + second(Cal
    .groupBy("CallAccountNumber") \
    .agg(
        avg("DurationSeconds").alias("AvgCallDurationSeconds"),
        count("CallId").alias("CallCount")
    ) \
    .orderBy(col("AvgCallDurationSeconds").desc())

print("4. Durée moyenne des appels par compte:")
avg_call_duration.show()
```

```
4. Durée moyenne des appels par compte:
+-----------------+----------------------+---------+
|CallAccountNumber|AvgCallDurationSeconds|CallCount|
+-----------------+----------------------+---------+
|               10|              4164.625|        8|
|               13|                2783.5|        2|
|               19|                1980.0|        1|
|               11|                1454.0|        4|
|               12|                 956.5|        2|
|               17|                 449.0|        1|
|               14|                 304.0|        1|
+-----------------+----------------------+---------+
```

Analyse des commandes par statut

In [42]:
```
order_status = orders_df.groupBy("OrderStatus") \
    .agg(count("OrderId").alias("OrderCount")) \
    .orderBy(col("OrderCount").desc())

print("3. Analyse des commandes par statut:")
order_status.show()
```

```
3. Analyse des commandes par statut:
+------------------+----------+
|       OrderStatus|OrderCount|
+------------------+----------+
|   Order Cancelled|         4|
|           Shipped|         3|
| Partially Shipped|         3|
|           Pending|         3|
|        On The way|         2|
|  Refund Initiated|         2|
|Payment Incomplete|         2|
|    Order Decilned|         1|
+------------------+----------+
```

Top 5 des employés avec le plus de clients

In [43]:
```
top_salespeople = customer_df.join(employee_df, customer_df.CustomerSalesPersonId == emp
    .groupBy("EmployeeId", "Employee_Name") \
    .agg(count("CustomerId").alias("CustomerCount")) \
    .orderBy(col("CustomerCount").desc()) \
    .limit(5)
```

```
print("6. Top 5 des employés avec le plus de clients:")
top_salespeople.show()
```

```
6. Top 5 des employés avec le plus de clients:
+----------+-------------------+-------------+
|EmployeeId|      Employee_Name|CustomerCount|
+----------+-------------------+-------------+
|         1|     Ojas Phansekar|            5|
|         2|Shreyas Kalayanar...|            5|
|         3|   Saurabh Kulkarni|            5|
|         4|        Vivek Shetye|            5|
+----------+-------------------+-------------+
```

Calcul du salaire moyen par département et du ratio de salaire

In [44]:

```
window_spec = Window.partitionBy("DepartmentId")
df_with_salary_ratio = df_with_tenure.withColumn(
    "AvgDeptSalary", avg("Salary").over(window_spec)
).withColumn(
    "SalaryRatio", col("Salary") / col("AvgDeptSalary")
)
df_with_salary_ratio.show()
```

```
+----------+------------------+---------+---+------------+------+-------------+-------
---+-----------+------------+----------+
|EmployeeId|     Employee_Name|      SSN|Age|DepartmentId|Salary|ImputedSalary|  HireD
ate|TenureYears|AvgDeptSalary|SalaryRatio|
+----------+------------------+---------+---+------------+------+-------------+-------
---+-----------+------------+----------+
|         1|    Ojas Phansekar|123456789| 24|           1|  1000|       1000.0|2020-01
-01|       4.66|      1000.0|       1.0|
|         2|Shreyas Kalayanar...|245987675| 24|           1|  1000|       1000.0|2020-01
-01|       4.66|      1000.0|       1.0|
|         3|  Saurabh Kulkarni|734756953| 24|           1|  1000|       1000.0|2020-01
-01|       4.66|      1000.0|       1.0|
|         4|      Vivek Shetye|572364526| 26|           1|  1000|       1000.0|2020-01
-01|       4.66|      1000.0|       1.0|
|         5|       Mihir Patil|238745784| 27|           1|  1000|       1000.0|2020-01
-01|       4.66|      1000.0|       1.0|
|        32|   Manoj Prabhakar|444787654| 21|           1|  1000|       1000.0|2020-01
-01|       4.66|      1000.0|       1.0|
|        34|       Priya Yadav|228787654| 33|           1|  1000|       1000.0|2020-01
-01|       4.66|      1000.0|       1.0|
|        11|    Vaibhav Parkar|123456789| 24|           2|  5000|       5000.0|2020-01
-01|       4.66|      5000.0|       1.0|
|        12|  Sayali Sakhalkar|674378987| 24|           2|  5000|       5000.0|2020-01
-01|       4.66|      5000.0|       1.0|
|        13|     Khushi Chavan|652134897| 45|           2|  5000|       5000.0|2020-01
-01|       4.66|      5000.0|       1.0|
|        14|      Pratik Patre|677435432| 24|           2|  5000|       5000.0|2020-01
-01|       4.66|      5000.0|       1.0|
|        15|           Pushkar|564321879| 43|           2|  5000|       5000.0|2020-01
-01|       4.66|      5000.0|       1.0|
|        30|      Ranjani Iyer|777787654| 34|           2|  5000|       5000.0|2020-01
-01|       4.66|      5000.0|       1.0|
|        17|Pranav Swaminathan|990077663| 34|           3|  2500|       2500.0|2020-01
-01|       4.66|      2500.0|       1.0|
|        18|            Victor|563477778| 44|           3|  2500|       2500.0|2020-01
-01|       4.66|      2500.0|       1.0|
|        19|       Yusuf Ozbek|995912563| 45|           3|  2500|       2500.0|2020-01
-01|       4.66|      2500.0|       1.0|
|        20|Sudharshan Poojary|763459876| 24|           3|  2500|       2500.0|2020-01
-01|       4.66|      2500.0|       1.0|
|        28|     Alpana Sharan|987787654| 45|           3|  2500|       2500.0|2020-01
-01|       4.66|      2500.0|       1.0|
|        29|    Priyanka Singh|238787654| 43|           3|  2500|       2500.0|2020-01
-01|       4.66|      2500.0|       1.0|
|        37|       Rohit Patil|222787654| 45|           3|  2500|       2500.0|2020-01
-01|       4.66|      2500.0|       1.0|
+----------+------------------+---------+---+------------+------+-------------+-------
---+-----------+------------+----------+
only showing top 20 rows
```

Création d'une catégorie de performance

In [46]:

```python
df_with_performance = df_with_salary_ratio.withColumn(
    "PerformanceCategory",
    when((col("SalaryRatio") > 1.2) & (col("TenureYears") > 5), "High Performer")
    .when((col("SalaryRatio") < 0.8) & (col("TenureYears") <= 2), "Needs Improvement")
    .otherwise("Average Performer")
)

# Affichage des résultats
```

```
print("Employés avec catégories de performance:")
df_with_performance.select("EmployeeId", "Employee_Name", "Salary", "TenureYears", "Sala
```

```
Employés avec catégories de performance:
+----------+-------------------+------+-----------+-----------+-------------------+
|EmployeeId|      Employee_Name|Salary|TenureYears|SalaryRatio|PerformanceCategory|
+----------+-------------------+------+-----------+-----------+-------------------+
|         1|      Ojas Phansekar|  1000|       4.66|        1.0|  Average Performer|
|         2|Shreyas Kalayanar...|  1000|       4.66|        1.0|  Average Performer|
|         3|    Saurabh Kulkarni|  1000|       4.66|        1.0|  Average Performer|
|         4|        Vivek Shetye|  1000|       4.66|        1.0|  Average Performer|
|         5|         Mihir Patil|  1000|       4.66|        1.0|  Average Performer|
|        32|      Manoj Prabhakar|  1000|       4.66|        1.0|  Average Performer|
|        34|         Priya Yadav|  1000|       4.66|        1.0|  Average Performer|
|        11|      Vaibhav Parkar|  5000|       4.66|        1.0|  Average Performer|
|        12|     Sayali Sakhalkar|  5000|       4.66|        1.0|  Average Performer|
|        13|       Khushi Chavan|  5000|       4.66|        1.0|  Average Performer|
|        14|        Pratik Patre|  5000|       4.66|        1.0|  Average Performer|
|        15|             Pushkar|  5000|       4.66|        1.0|  Average Performer|
|        30|        Ranjani Iyer|  5000|       4.66|        1.0|  Average Performer|
|        17|  Pranav Swaminathan|  2500|       4.66|        1.0|  Average Performer|
|        18|              Victor|  2500|       4.66|        1.0|  Average Performer|
|        19|         Yusuf Ozbek|  2500|       4.66|        1.0|  Average Performer|
|        20|  Sudharshan Poojary|  2500|       4.66|        1.0|  Average Performer|
|        28|       Alpana Sharan|  2500|       4.66|        1.0|  Average Performer|
|        29|      Priyanka Singh|  2500|       4.66|        1.0|  Average Performer|
|        37|         Rohit Patil|  2500|       4.66|        1.0|  Average Performer|
+----------+-------------------+------+-----------+-----------+-------------------+
only showing top 20 rows
```

Statistiques sur les catégories de performance

In [47]:

```
performance_stats = df_with_performance.groupBy("PerformanceCategory").agg(
    count("*").alias("EmployeeCount"),
    avg("Salary").alias("AvgSalary"),
    avg("TenureYears").alias("AvgTenure")
)

print("\nStatistiques par catégorie de performance:")
performance_stats.show()
```

```
Statistiques par catégorie de performance:
+-------------------+-------------+----------------+------------------+
|PerformanceCategory|EmployeeCount|       AvgSalary|         AvgTenure|
+-------------------+-------------+----------------+------------------+
|  Average Performer|           37|5594.594594594595|4.6599999999999975|
+-------------------+-------------+----------------+------------------+
```

standarizer ssn

In [49]:

```
from pyspark.sql.functions import col, when, avg, regexp_replace, concat, substring, lit

employee_cleaned = employee_df.na.fill({
    "DepartmentId": 0,
    "Salary": employee_df.select(avg("Salary")).collect()[0][0]
})

employee_cleaned = employee_cleaned.withColumn(
```

```
    regexp_replace(col("SSN"), r'[^0-9]', '')
).withColumn(
    "SSN",
    when(length(col("SSN")) == 9,
         concat(substring(col("SSN"), 1, 3), lit("-"),
                substring(col("SSN"), 4, 2), lit("-"),
                substring(col("SSN"), 6, 4)))
    .otherwise(lit(None))
)

print("Données des employés nettoyées :")
employee_cleaned.show(truncate=False)
```

```
Données des employés nettoyées :
+----------+--------------------+-----------+---+------------+------+
|EmployeeId|Employee_Name       |SSN        |Age|DepartmentId|Salary|
+----------+--------------------+-----------+---+------------+------+
|1         |Ojas Phansekar      |123-45-6789|24 |1           |1000  |
|2         |Shreyas Kalayanaraman|245-98-7675|24 |1           |1000  |
|3         |Saurabh Kulkarni    |734-75-6953|24 |1           |1000  |
|4         |Vivek Shetye        |572-36-4526|26 |1           |1000  |
|5         |Mihir Patil         |238-74-5784|27 |1           |1000  |
|6         |Karan Thevar        |968-37-4657|28 |4           |7500  |
|7         |Chetan Mistry       |623-78-4983|30 |4           |7500  |
|8         |Shantanu Sawant     |527-47-3298|24 |4           |7500  |
|9         |Pooja Patil         |286-43-6778|24 |4           |7500  |
|10        |Kalpita Malvankar   |863-47-6236|34 |4           |7500  |
|11        |Vaibhav Parkar      |123-45-6789|24 |2           |5000  |
|12        |Sayali Sakhalkar    |674-37-8987|24 |2           |5000  |
|13        |Khushi Chavan       |652-13-4897|45 |2           |5000  |
|14        |Pratik Patre        |677-43-5432|24 |2           |5000  |
|15        |Pushkar             |564-32-1879|43 |2           |5000  |
|16        |Tushar Gupta        |444-77-7651|24 |5           |10000 |
|17        |Pranav Swaminathan  |990-07-7663|34 |3           |2500  |
|18        |Victor              |563-47-7778|44 |3           |2500  |
|19        |Yusuf Ozbek         |995-91-2563|45 |3           |2500  |
|20        |Sudharshan Poojary  |763-45-9876|24 |3           |2500  |
+----------+--------------------+-----------+---+------------+------+
only showing top 20 rows
```

Clean customer data

In [51]:

```
customer_cleaned = customer_df.withColumn(
    "Sex", upper(trim(col("Sex")))
).withColumn(
    "DateOfBirth", to_date(col("DateOfBirth"))
).withColumn(
    "Age", round(datediff(current_date(), col("DateOfBirth")) / 365.25)
)
customer_cleaned.show()
```

```
+----------+---------------+---+----+----------+-------------------+----------------
-----+
|CustomerId|   CustomerName|Sex| Age|DateOfBirth|SocialSecurityNumber|CustomerSalesPer
sonId|
+----------+---------------+---+----+----------+-------------------+----------------
-----+
|         1| Jishnu Vasudevan|  M|31.0| 1993-12-28|           232498675|
1|
|         2|       Harsh Shah|  M|31.0| 1993-09-12|           456498675|
2|
|         3|  Rachana Rambhad|  F|31.0| 1993-08-19|           543498675|
3|
|         4|      Lagan Gupta|  F|31.0| 1993-08-08|           765498675|
4|
|         5|       Neha Verma|  F|31.0| 1993-08-27|           987498675|
1|
|         6|      Aniel Patel|  M|31.0| 1993-11-28|           235468675|
2|
|         7|    Anubhav Gupta|  M|34.0| 1990-12-28|           555698675|
3|
|         8|     Aditya Joshi|  M|31.0| 1993-10-28|           232434575|
4|
|         9|     Parnal Dighe|  F|31.0| 1993-09-28|           232498765|
1|
|        10|      Dharit Shah|  M|31.0| 1993-12-27|           123498675|
2|
|        11|     Girish Sanai|  M|31.0| 1993-07-22|           645498675|
3|
|        12|      Kal Bugrara|  M|64.0| 1960-12-28|           145498675|
4|
|        13|    Neeraj Rajput|  M|34.0| 1990-10-28|           232555675|
1|
|        14|     Shruti Mehta|  F|33.0| 1991-12-17|           232444375|
2|
|        15|      Sameer Goel|  M|35.0| 1989-12-30|           276578675|
3|
|        16|Vijayshree Uppili|  F|33.0| 1991-08-23|           654498675|
4|
|        17|     Rohit Kamble|  M|31.0| 1993-06-28|           453498675|
1|
|        18|   Priyanka Desai|  F|33.0| 1991-04-23|           189498675|
2|
|        19|  Komal Shirodkar|  F|33.0| 1991-02-27|           678498675|
3|
|        20|      Simmah Kazi|  F|29.0| 1995-12-28|           232834675|
4|
+----------+---------------+---+----+----------+-------------------+----------------
-----+
```

gérer les incohérences dans order status

In [52]:

```
orders_cleaned = orders_df.withColumn(
    "OrderStatus",
    when(col("OrderStatus").isin("Shipped", "On The way"), "In Transit")
    .when(col("OrderStatus") == "Order Cancelled", "Cancelled")
    .when(col("OrderStatus").isin("Pending", "Payment Incomplete"), "Pending")
    .otherwise(col("OrderStatus"))
)
orders_cleaned.show()
```

```
+-------+----------------+----------------+---------------+
|OrderId|       OrderType|     OrderStatus|OrderCustomerId|
+-------+----------------+----------------+---------------+
|      1|   2 day shipping|      In Transit|              1|
|      2|Priority Shipping|Partially Shipped|              2|
|      3|         Standard|         Pending|              3|
|      4|   2 day shipping|       Cancelled|              4|
|      5|         Standard|         Pending|              5|
|      6|Priority Shipping| Refund Initiated|              6|
|      7|   2 day shipping|       Cancelled|              7|
|      8|         Standard|         Pending|              8|
|      9|Priority Shipping|Partially Shipped|              9|
|     10|   2 day shipping|      In Transit|             10|
|     11|         Standard|       Cancelled|             11|
|     12|Priority Shipping|Partially Shipped|             12|
|     13|   2 day shipping|         Pending|             13|
|     14|         Standard|      In Transit|             14|
|     15|Priority Shipping|      In Transit|             15|
|     16|   2 day shipping|       Cancelled|             16|
|     17|         Standard|   Order Decilned|             17|
|     18|Priority Shipping| Refund Initiated|             18|
|     19|   2 day shipping|         Pending|             19|
|     20|         Standard|      In Transit|             20|
+-------+----------------+----------------+---------------+
```

identifie les doublons potentiels

In [53]:
```
window_spec = Window.partitionBy("Employee_Name", "SSN").orderBy("EmployeeId")

employee_deduped = employee_cleaned.withColumn(
    "IsPotentialDuplicate",
    row_number().over(window_spec) > 1
)

print("Employés avec identification des doublons potentiels:")
employee_deduped.show(truncate=False)

# Afficher uniquement les doublons potentiels
print("\nDoublons potentiels:")
employee_deduped.filter(col("IsPotentialDuplicate") == True).show(truncate=False)
```

```
Employés avec identification des doublons potentiels:
+----------+-----------------+-----------+---+------------+------+--------------------+
|EmployeeId|Employee_Name    |SSN        |Age|DepartmentId|Salary|IsPotentialDuplicate|
+----------+-----------------+-----------+---+------------+------+--------------------+
|28        |Alpana Sharan    |987-78-7654|45 |3           |2500  |false               |
|31        |Amlan Bhuyan     |555-78-7654|23 |4           |7500  |false               |
|7         |Chetan Mistry    |623-78-4983|30 |4           |7500  |false               |
|22        |Devdip Sen       |458-78-7654|56 |5           |10000 |false               |
|23        |Devdip Sen       |458-78-7654|56 |5           |10000 |true                |
|24        |Devdip Sen       |458-78-7654|56 |5           |10000 |true                |
|26        |Devdip Sen       |458-78-7654|56 |5           |10000 |true                |
|27        |Devdip Sen       |458-78-7654|56 |5           |10000 |true                |
|10        |Kalpita Malvankar|863-47-6236|34 |4           |7500  |false               |
|6         |Karan Thevar     |968-37-4657|28 |4           |7500  |false               |
|13        |Khushi Chavan    |652-13-4897|45 |2           |5000  |false               |
|32        |Manoj Prabhakar  |444-78-7654|21 |1           |1000  |false               |
|5         |Mihir Patil      |238-74-5784|27 |1           |1000  |false               |
|1         |Ojas Phansekar   |123-45-6789|24 |1           |1000  |false               |
|21        |Parth Mehta      |458-78-7654|56 |5           |10000 |false               |
|9         |Pooja Patil      |286-43-6778|24 |4           |7500  |false               |
|36        |Pranav Patil     |658-78-7654|25 |5           |10000 |false               |
|17        |Pranav Swaminathan|990-07-7663|34 |3           |2500  |false               |
|14        |Pratik Patre     |677-43-5432|24 |2           |5000  |false               |
|34        |Priya Yadav      |228-78-7654|33 |1           |1000  |false               |
+----------+-----------------+-----------+---+------------+------+--------------------+
only showing top 20 rows


Doublons potentiels:
+----------+-------------+-----------+---+------------+------+--------------------+
|EmployeeId|Employee_Name|SSN        |Age|DepartmentId|Salary|IsPotentialDuplicate|
+----------+-------------+-----------+---+------------+------+--------------------+
|23        |Devdip Sen   |458-78-7654|56 |5           |10000 |true                |
|24        |Devdip Sen   |458-78-7654|56 |5           |10000 |true                |
|26        |Devdip Sen   |458-78-7654|56 |5           |10000 |true                |
|27        |Devdip Sen   |458-78-7654|56 |5           |10000 |true                |
+----------+-------------+-----------+---+------------+------+--------------------+
```

Analyse de la performance des vendeurs

In [54]:

```
salesperson_performance = customer_df.join(salesperson_df, customer_df.CustomerSalesPers
    .join(orders_df, customer_df.CustomerId == orders_df.OrderCustomerId) \
    .join(employee_df, salesperson_df.IdEmployeeSalesPerson == employee_df.EmployeeId) \
    .groupBy("IdEmployeeSalesPerson", "Employee_Name") \
    .agg(
        count("CustomerId").alias("TotalCustomers"),
        count("OrderId").alias("TotalOrders"),
        sum(when(col("OrderStatus") == "Shipped", 1).otherwise(0)).alias("CompletedOrder
    ) \
    .withColumn("OrderCompletionRate", round(col("CompletedOrders") / col("TotalOrders")


print("Performance des vendeurs:")
salesperson_performance.show()
```

```
Performance des vendeurs:
+--------------------+--------------+-------------+-----------+--------------+------
-------------+
|IdEmployeeSalesPerson|  Employee_Name|TotalCustomers|TotalOrders|CompletedOrders|OrderC
ompletionRate|
+--------------------+--------------+-------------+-----------+--------------+------
-------------+
|                    5|    Mihir Patil|            5|          5|              1|
0.2|
|                    6|   Karan Thevar|            5|          5|              2|
0.4|
|                    7|  Chetan Mistry|            5|          5|              0|
0.0|
|                    8|Shantanu Sawant|            5|          5|              0|
0.0|
+--------------------+--------------+-------------+-----------+--------------+------
-------------+
```

Analyse des employés par département

In [55]:
```
employee_dept = employee_df.join(department_df, "DepartmentId")
employee_dept.groupBy("DepartmentName").count().show()
```

```
+--------------------+-----+
|      DepartmentName|count|
+--------------------+-----+
|Information Techn...|    7|
|             Finance|    7|
|      Human Resource|    7|
|   Sales & Marketing|    6|
|       Customer Care|   10|
+--------------------+-----+
```

Analyse des ventes par vendeur

In [56]:
```
sales_analysis = customer_df.join(salesperson_df, customer_df.CustomerSalesPersonId == s
sales_analysis = sales_analysis.join(employee_df, salesperson_df.IdEmployeeSalesPerson =
sales_analysis.groupBy("Employee_Name").count().orderBy("count", ascending=False).show()
```

```
+---------------+-----+
|  Employee_Name|count|
+---------------+-----+
|   Karan Thevar|    5|
|Shantanu Sawant|    5|
|    Mihir Patil|    5|
|  Chetan Mistry|    5|
+---------------+-----+
```

Analyse des appels par client

In [57]:
```
call_analysis = callrecords_df.join(phonenumber_df, callrecords_df.CallAccountNumber ==
call_analysis = call_analysis.join(simdata_df, phonenumber_df.AccountNumber == simdata_d
call_analysis = call_analysis.join(customer_df, simdata_df.SimCustomerId == customer_df.

from pyspark.sql.functions import sum, col
```

Loading [MathJax]/extensions/Safe.js

```
call_duration = call_analysis.groupBy("CustomerName").agg(sum(col("CallDuration").cast("
call_duration.orderBy("TotalDuration", ascending=False).show()
```

```
+----------------+-------------+
|    CustomerName|TotalDuration|
+----------------+-------------+
|Vijayshree Uppili|       33317|
|   Neeraj Rajput|        5816|
|     Sameer Goel|        5567|
|      Harsh Shah|        1980|
|    Shruti Mehta|        1913|
|      Neha Verma|         449|
| Jishnu Vasudevan|         304|
+----------------+-------------+
```

Analyse des plans les plus populaires

In [58]:

```
popular_plans = simdata_df.join(plans_df, simdata_df.SimPlanNumber == plans_df.PlansId)
popular_plans.groupBy("PlanName").count().orderBy("count", ascending=False).show()
```

```
+-----------------+-----+
|         PlanName|count|
+-----------------+-----+
|           Family|    2|
|    Do not disturb|    1|
|      Finger tips|    1|
|     Enjoy surfing|    1|
|    Talk For Hours|    1|
|        Enjoy Data|    1|
|Continuous Texting|    1|
|    Powerful Speed|    1|
|        Basic Plan|    1|
+-----------------+-----+
```

In [59]:

```
# Calcul de l'écart salarial par rapport à la moyenne du département
avg_salary_by_dept = employee_df.groupBy("DepartmentId").agg(avg("Salary").alias("AvgDep
salary_comparison = employee_df.join(avg_salary_by_dept, "DepartmentId") \
    .withColumn("SalaryDifference", col("Salary") - col("AvgDeptSalary")) \
    .withColumn("SalaryDifferencePercent", (col("Salary") - col("AvgDeptSalary")) / col(

print("\n4. Comparaison des salaires avec la moyenne du département:")
print(display_df(salary_comparison))
```

```
4. Comparaison des salaires avec la moyenne du département:
   DepartmentId  EmployeeId          Employee_Name        SSN  Age  Salary  \
0             1           1          Ojas Phansekar  123456789   24    1000
1             1           2  Shreyas Kalayanaraman  245987675   24    1000
2             1           3        Saurabh Kulkarni  734756953   24    1000
3             1           4            Vivek Shetye  572364526   26    1000
4             1           5             Mihir Patil  238745784   27    1000
5             1          32         Manoj Prabhakar  444787654   21    1000
6             1          34             Priya Yadav  228787654   33    1000
7             3          17      Pranav Swaminathan  990077663   34    2500
8             3          18                  Victor  563477778   44    2500
9             3          19             Yusuf Ozbek  995912563   45    2500

   AvgDeptSalary  SalaryDifference  SalaryDifferencePercent
0         1000.0               0.0                      0.0
1         1000.0               0.0                      0.0
2         1000.0               0.0                      0.0
3         1000.0               0.0                      0.0
4         1000.0               0.0                      0.0
5         1000.0               0.0                      0.0
6         1000.0               0.0                      0.0
7         2500.0               0.0                      0.0
8         2500.0               0.0                      0.0
9         2500.0               0.0                      0.0
```

In [60]:

```python
# Calcul du ratio salaire/âge
salary_age_ratio = employee_df.withColumn("SalaryAgeRatio", col("Salary") / col("Age"))

print("\n7. Ratio salaire/âge:")
print(display_df(salary_age_ratio))
```

```
7. Ratio salaire/âge:
   EmployeeId          Employee_Name        SSN  Age  DepartmentId  Salary  \
0           1          Ojas Phansekar  123456789   24             1    1000
1           2  Shreyas Kalayanaraman  245987675   24             1    1000
2           3        Saurabh Kulkarni  734756953   24             1    1000
3           4            Vivek Shetye  572364526   26             1    1000
4           5             Mihir Patil  238745784   27             1    1000
5           6            Karan Thevar  968374657   28             4    7500
6           7           Chetan Mistry  623784983   30             4    7500
7           8         Shantanu Sawant  527473298   24             4    7500
8           9             Pooja Patil  286436778   24             4    7500
9          10        Kalpita Malvankar  863476236   34             4    7500

   SalaryAgeRatio
0       41.666667
1       41.666667
2       41.666667
3       38.461538
4       37.037037
5      267.857143
6      250.000000
7      312.500000
8      312.500000
9      220.588235
```

In [ ]: