Practice Project: Furniture Sales Data Analysis

Scenario

Furniture World, a global furniture retail company, operates across multiple countries, selling a wide range of furniture products like sofas, beds, chairs, and tables. Over the past few years, the company has seen rapid growth, leading to an influx of sales data from different regions. However, Furniture World is now facing significant challenges in understanding sales patterns, optimizing inventory management, and improving customer targeting across its markets. The company lacks a comprehensive data-driven approach to analyze its sales trends, understand regional performance, and identify best-selling products.

Problem Statement

The primary objective of this project is to leverage PySpark SQL to analyze Furniture World's sales data for actionable insights. The analysis will optimize product inventory, identify sales trends, and enhance customer segmentation. Key tasks include calculating total sales per product, examining regional performance, and analyzing customer spending patterns. Ultimately, these insights will drive data-driven decisions to improve overall business performance and revenue.

0. Preparación de entorno

```
#importado de librerías
from pyspark.sql import SparkSession
from pyspark.sql.window import Window
from pyspark.sql.functions import col, to_date, sum, month, year
from pyspark.sql.functions import asc_nulls_last, avg, rank, desc, weekday
from pyspark.sql.functions import udf, max, round, date_format
import plotly.express as px


#Creación de sesión Spark
sparkSession = SparkSession.builder.appName('Furniture Sales').getOrCreate()
```

1. Load the sales data into a PySpark DataFrame so we can work with it.

```
path = '/content/drive/MyDrive/Colab Notebooks/data/Furniture-Sales-Data.csv'
dfSales = sparkSession.read.csv(path,
                                sep = ',',
                                header = True,
                                inferSchema = True)
dfSales.printSchema()
dfSales.show(5)
```

```
root
 |-- OrderID: string (nullable = true)
 |-- CustomerID: string (nullable = true)
 |-- ProductNames: string (nullable = true)
 |-- Quantity: integer (nullable = true)
 |-- Price: double (nullable = true)
 |-- OrderDate: date (nullable = true)
 |-- Region: string (nullable = true)

+---------+----------+------------+--------+-------+----------+------+
|  OrderID|CustomerID|ProductNames|Quantity|  Price| OrderDate|Region|
+---------+----------+------------+--------+-------+----------+------+
|ORD100000|  CUST9055|   Bookshelf|       2|1865.65|2023-03-12|France|
|ORD100001|  CUST1538|        Sofa|       4|1726.87|2024-03-08| China|
|ORD100002|  CUST6940|Coffee Table|       4| 760.89|2022-06-20| China|
|ORD100003|  CUST1625|     Dresser|       5| 368.03|2023-05-01|France|
|ORD100004|  CUST9204|    TV Stand|       1| 753.76|2023-07-03| India|
+---------+----------+------------+--------+-------+----------+------+
only showing top 5 rows
```

2. Clean the data by fixing missing values and making sure the dates are in the right format.

```
rowsBeforeDeleteNulls = dfSales.count()
dfSales = dfSales.dropna()
print(f'Filas antes de borrado de nulls: {rowsBeforeDeleteNulls}; filas después del borrado: {dfSales.count()}')
```

```
Filas antes de borrado de nulls: 1500; filas después del borrado: 1500
```

```
dfDateCheck = dfSales.select('OrderDate').withColumn('ConvertedDate', to_date(col('OrderDate'), 'yyyy-MM-dd'))
dfDateCheck = dfDateCheck.filter(dfDateCheck.OrderDate != dfDateCheck.ConvertedDate)
print(f'Filas con formato de fecha incorrecto: {dfDateCheck.count()}')
```

```
Filas con formato de fecha incorrecto: 0
```

3.Calculate the total sales for each product to see which products bring in the most money.

```
dfSalesByProduct = dfSales.groupBy('ProductNames').agg(sum('Price').alias('TotalSales'))\
                          .orderBy('TotalSales', ascending = False).limit(1)
dfSalesByProduct.show()
```

```
+------------+------------------+
|ProductNames|        TotalSales|
+------------+------------------+
|    Recliner|176088.30000000008|
+------------+------------------+
```

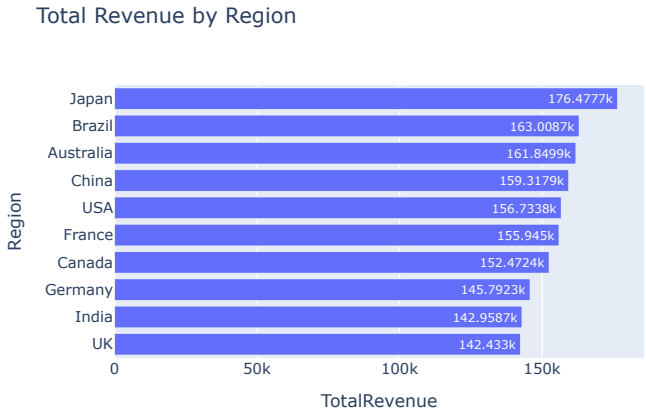
4.Find out which products have sold the most units overall.

```
dfTop5BestSellingProducts = dfSales.groupBy('ProductNames').agg(sum('Quantity').alias('TotalQuantity'))\
                                   .orderBy('TotalQuantity', ascending = False).limit(5)
dfTop5BestSellingProducts.show()
```

```
+------------+-------------+
|ProductNames|TotalQuantity|
+------------+-------------+
|Office Chair|          921|
|   Bookshelf|          882|
|    Wardrobe|          882|
|    Recliner|          872|
|Coffee Table|          856|
+------------+-------------+
```

5.Analyze how much revenue each region generates to compare performance across locations.

```
dfRevenueByRegion = dfSales.groupBy('Region').agg(sum('Price').alias('TotalRevenue'))
pdRevenueByRegion = dfRevenueByRegion.orderBy('TotalRevenue', ascending = True).toPandas()
fig = px.bar(pdRevenueByRegion, y='Region',
             x='TotalRevenue',
             title='Total Revenue by Region',
             orientation = 'h',
             text_auto = True)
fig.update_layout(width = 600, height = 400)
fig.show()
```



6.Look at monthly and yearly sales patterns to understand which times of the year sales are highest.

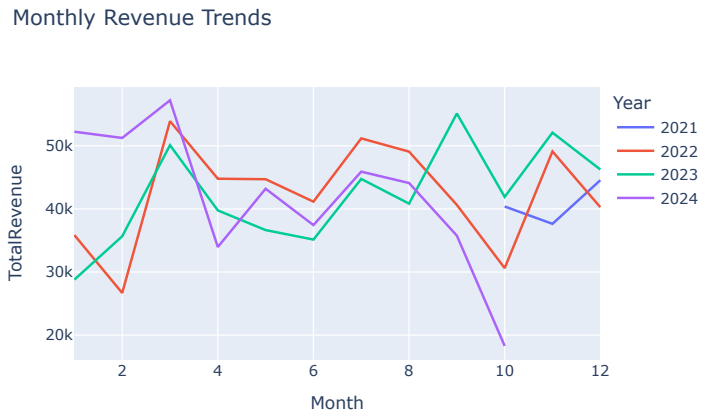
```
#Preparación de dataframe para cálculos de cubo y gráfico
dfTrendsByDate = dfSales.withColumn('Year', year(col('OrderDate')))\
                        .withColumn('Month', month(col('OrderDate')))\
                        .groupBy('Year', 'Month')\
                        .agg(sum('Price').alias('TotalRevenue'))\
                        .orderBy('Year','Month')\
                        .withColumn('TotalRevenue', col('TotalRevenue').cast('int'))

#Creación del cubo de cálculo
dfTrendsByDateCube = dfTrendsByDate.cube('Year', 'Month').agg(sum('TotalRevenue').alias('TotalRevenue'))
dfTrendsByDateCube = dfTrendsByDateCube.orderBy(asc_nulls_last('Year'), asc_nulls_last('Month'))

#Creación de matriz de meses y años y formateo de totales
dfTrendsByDateCube = dfTrendsByDateCube.groupBy('Month').pivot('Year').sum('TotalRevenue')\
                                       .orderBy(asc_nulls_last('Month'))
dfTrendsByDateCube = dfTrendsByDateCube.withColumn('Month', col('Month').cast('string'))\
                                       .fillna({'Month':'Total'})\
                                       .withColumnRenamed('null', 'Total')
dfTrendsByDateCube.show()
```

```
+-----+-------+------+------+------+------+
|Month|  Total|  2021|  2022|  2023|  2024|
+-----+-------+------+------+------+------+
|    1| 116874|  NULL| 35851| 28786| 52237|
|    2| 113597|  NULL| 26682| 35666| 51249|
|    3| 161222|  NULL| 53903| 50103| 57216|
|    4| 118513|  NULL| 44785| 39763| 33965|
|    5| 124551|  NULL| 44709| 36644| 43198|
|    6| 113727|  NULL| 41153| 35139| 37435|
|    7| 141835|  NULL| 51177| 44752| 45906|
|    8| 134024|  NULL| 49083| 40838| 44103|
|    9| 131526|  NULL| 40638| 55126| 35762|
|   10| 131191| 40387| 30614| 41907| 18283|
|   11| 138838| 37634| 49111| 52093|  NULL|
|   12| 131072| 44547| 40275| 46250|  NULL|
|Total|1556970|122568|507981|507067|419354|
+-----+-------+------+------+------+------+
```

```
#Gráfico de lineas
pdTrendsByDate = dfTrendsByDate.toPandas()
fig = px.line(pdTrendsByDate, x = 'Month', y = 'TotalRevenue', color = 'Year', title = 'Monthly Revenue Trends')
fig.update_layout(width = 600, height = 400)
fig.show()
```



7.Group customers based on how much they spend to identify the most valuable ones.

```
dfSpendingByCustomer = dfSales.groupby('CustomerID')\
                              .agg(sum('Price')\
                                   .alias('TotalSpending'))\
                                   .orderBy('TotalSpending', ascending = False)\
                                   .withColumn('TotalSpending', col('TotalSpending').cast('int'))
dfSpendingByCustomer.show(5)
```

```
+----------+-------------+
|CustomerID|TotalSpending|
+----------+-------------+
|  CUST4297|         4488|
|  CUST9121|         4237|
|  CUST8423|         3836|
|  CUST4630|         3709|
|  CUST2640|         3706|
+----------+-------------+
only showing top 5 rows
```

8. Calculate the average amount spent on each order to understand typical customer spending.

```
#Preparación de vista sobre el dataframe dfSales
dfSales.createOrReplaceTempView('sales')
#Declaración de sentencia sql de consulta
sqlAverageSpendingByOrder = '''SELECT ROUND(AVG(Price),2)
                               FROM sales'''
#Ejecución de sentencia sql y salida de resultado
print(f'El promedio de gasto por orden es: {sparkSession.sql(sqlAverageSpendingByOrder).collect()[0][0]}')
```

```
El promedio de gasto por orden es: 1037.99
```

9.Compare how well each product sells in different regions to find out local preferences

```
#Cálculo de ventas por producto y Region
dfSalesByProductAndRegion = dfSales.groupBy('ProductNames', 'Region')\
                                   .agg(sum('Quantity').alias('TotalQuantity'))\
```

```
                                        .orderBy('ProductNames', 'TotalQuantity')

#Creación de ventana para ranqueo de producto por región
window = Window.partitionBy('Region').orderBy(desc('TotalQuantity'))
dfSalesByProductAndRegion = dfSalesByProductAndRegion.withColumn('Rank', rank().over(window))

#Filtrado del producto con mayores ventas en cada región
dfBestProductByRegion = dfSalesByProductAndRegion.filter('Rank == 1')\
                                                 .drop('Rank')\
                                                 .select('Region','ProductNames','TotalQuantity')
dfBestProductByRegion.show()
```

```
+---------+------------+-------------+
|   Region|ProductNames|TotalQuantity|
+---------+------------+-------------+
|Australia|    Wardrobe|          142|
|   Brazil|   Bookshelf|          119|
|   Canada|   Bed Frame|          123|
|    China|   Bed Frame|          132|
|   France|   Bookshelf|          127|
|  Germany|     Dresser|          112|
|    India|   Bed Frame|           99|
|    India|Coffee Table|           99|
|    Japan|Coffee Table|          147|
|       UK|    Recliner|          113|
|      USA|Coffee Table|          127|
+---------+------------+-------------+
```

10.Identify products with consistently low sales to decide if any changes are needed

```
#Cálculo de total de unidades vendidas por cada producto
dfLowSalesProducts = dfSales.groupBy('ProductNames')\
                            .agg(sum('Quantity').alias('TotalQuantity'))\
                            .orderBy('TotalQuantity')

#Cálculo de la media de unidades vendidas para todos los productos
avgSales = dfLowSalesProducts.agg(avg('TotalQuantity')).collect()[0][0]

#Filtrado de productos con pocas ventas: productos con ventas inferior a la media
dfLowSalesProducts = dfLowSalesProducts.filter(dfLowSalesProducts.TotalQuantity < avgSales)
dfLowSalesProducts.show()
```

```
+------------+-------------+
|ProductNames|TotalQuantity|
+------------+-------------+
|     Dresser|          707|
|    TV Stand|          709|
|Dining Table|          744|
|        Sofa|          783|
+------------+-------------+
```

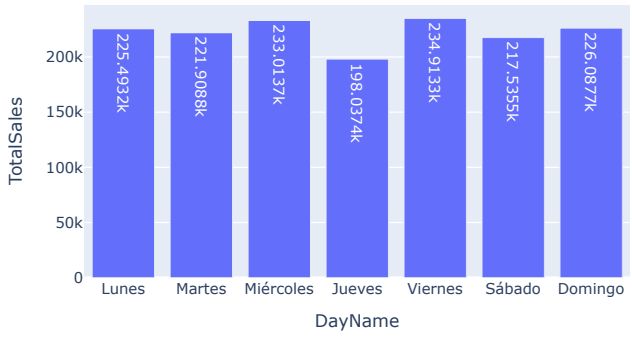
11.Find the specific days with the highest sales to understand when demand peaks

```
#Calculo de agregado de ventas para cada día de la semana
dfSalesByWeekDay = dfSales.withColumn('WeekDay', weekday('OrderDate'))\
                            .groupBy('WeekDay').agg(sum('Price').alias('TotalSales'))\
                            .orderBy('WeekDay')

#Definición de función de usuario para nombres de los días en castellano
#Para nombres de los días de la semana en inglés, usar date_format
def getDayName(dayNumber):
  names = ['Lunes', 'Martes', 'Miércoles', 'Jueves', 'Viernes', 'Sábado', 'Domingo']
  return names[dayNumber]
#End getDayName
getDayNameUDF = sparkSession.udf.register('getDayNameUDF', getDayName)

#Formato del día de la semana y generación de gráfico
dfSalesByWeekDay = dfSalesByWeekDay.withColumn('DayName', getDayNameUDF(col('WeekDay')))
pdSalesByWeekDay = dfSalesByWeekDay.toPandas()
fig = px.bar(pdSalesByWeekDay, y='TotalSales', x='DayName', title='Ventas acumuladas por día', text_auto = True)
fig.update_layout(width = 600, height = 400)
fig.show()
```

Ventas acumuladas por día



12.Suggest the right stock levels for each product based on past sales patterns

Show code

ProductNames	AverageQuantity	MaxQuantity	SuggestedStock
TV Stand	19.0	44	31
Office Chair	26.0	57	41
Coffee Table	24.0	45	34
Bed Frame	23.0	47	35
Sofa	21.0	53	37
Bookshelf	24.0	54	39
Wardrobe	24.0	54	39
Recliner	25.0	71	48
Dresser	19.0	45	32
Dining Table	21.0	49	35