```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, avg, dense_rank, sum
from pyspark.sql.window import Window
from datetime import date

# Crear una sesión de Spark
spark = SparkSession.builder \
    .appName("DataFrameExample") \
    .master("local[*]") \
    .getOrCreate()

# Crear un DataFrame a partir de una lista de tuplas
data = [
    (date(2024, 9, 1), "ProductoA", 10, 20.5),
    (date(2024, 9, 1), "ProductoB", 5, 10.0),
    (date(2024, 9, 2), "ProductoA", 7, 20.5),
    (date(2024, 9, 2), "ProductoB", 8, 10.0),
    (date(2024, 9, 3), "ProductoA", 15, 20.5),
    (date(2024, 9, 3), "ProductoB", 6, 10.0),
    (date(2024, 9, 4), "ProductoA", 14, 20.5),
    (date(2024, 9, 4), "ProductoB", 10, 10.0),
    (date(2024, 9, 5), "ProductoA", 12, 20.5),
    (date(2024, 9, 5), "ProductoB", 9, 10.0),
    (date(2024, 9, 6), "ProductoA", 11, 20.5),
    (date(2024, 9, 6), "ProductoB", 7, 10.0),
    (date(2024, 9, 7), "ProductoA", 9, 20.5),
    (date(2024, 9, 7), "ProductoB", 5, 10.0)
]

columns = ["fecha", "producto", "cantidad", "precio"]
df = spark.createDataFrame(data, columns)

# Realizar operaciones complejas
windowSpec = Window.partitionBy("producto").orderBy("fecha").rowsBetween(-6, 0)

resultadoDF = df \
    .groupBy("fecha", "producto") \
    .agg(
        sum("cantidad").alias("total_cantidad"),
        sum(col("precio") * col("cantidad")).alias("total_ventas")
    ) \
    .withColumn("promedio_7_dias",
                avg("total_ventas").over(windowSpec)) \
    .filter(col("total_ventas") > col("promedio_7_dias"))

# Mostrar el resultado
resultadoDF.show()
```

```

El código que has proporcionado es un excelente ejemplo de cómo trabajar con DataFrames en PySpark para realizar análisis de datos complejos. Vamos a desglosarlo paso a paso:

1. **`from pyspark.sql import SparkSession`**: Importa la clase `SparkSession`, el punto de entrada para trabajar con DataFrames y Spark SQL.
2. **`from pyspark.sql.functions import col, avg, dense_rank, sum`**: Importa varias funciones integradas de Spark SQL que se utilizarán para realizar transformaciones en el DataFrame.
    - `col`: Se utiliza para referirse a una columna por su nombre.
    - `avg`: Calcula el promedio de una columna.
    - `dense_rank`: Asigna rangos dentro de una partición de ventana sin huecos. (Aunque no se usa directamente en este ejemplo, se importa).
    - `sum`: Calcula la suma de los valores de una columna.
3. **`from pyspark.sql.window import Window`**: Importa la clase `Window`, que se utiliza para definir ventanas sobre las particiones de los datos para realizar cálculos como promedios móviles.
4. **`from datetime import date`**: Importa la clase `date` del módulo `datetime` para crear objetos de fecha para los datos del DataFrame.
5. **`spark = SparkSession.builder \`**: Inicia la construcción de una `SparkSession`.
    - **`.appName("DataFrameExample") \`**: Establece el nombre de la aplicación Spark.
    - **`.master("local[*]") \`**: Configura Spark para que se ejecute en modo local utilizando todos los núcleos disponibles.
    - **`.getOrCreate()`**: Obtiene una `SparkSession` existente o crea una nueva.
6. **`data = [...]`**: Define una lista de tuplas que representan los datos que se cargarán en el DataFrame. Cada tupla contiene:
    - Una fecha (`datetime.date` object).
    - El nombre de un producto (`str`).
    - La cantidad vendida (`int`).
    - El precio unitario (`float`).
7. **`columns = [...]`**: Define una lista de nombres de columna que se asignarán a los datos de la lista de tuplas al crear el DataFrame.
8. **`df = spark.createDataFrame(data, columns)`**: Crea un DataFrame de PySpark llamado `df` utilizando los datos y los nombres de columna definidos.
9. **`windowSpec = Window.partitionBy("producto").orderBy("fecha").rowsBetween(-6, 0)`**: Define una especificación de ventana.
    - **`.partitionBy("producto")`**: Divide los datos del DataFrame en particiones basadas en los valores únicos de la columna "producto". Los cálculos de ventana se realizarán independientemente dentro de cada partición de producto.
    - **`.orderBy("fecha")`**: Ordena los datos dentro de cada partición por la columna "fecha" en orden ascendente. Esto es importante para calcular el promedio móvil de los últimos 7 días.
    - **`.rowsBetween(-6, 0)`**: Define el marco de la ventana. Para cada fila, considerará la fila actual (0) y las 6 filas precedentes (-6). Esto crea una ventana de los últimos 7 días (incluyendo el día actual).
10. **`resultadoDF = df \`**: Comienza una serie de transformaciones en el DataFrame `df`.
```

- `.groupBy("fecha", "producto") \`: Agrupa el DataFrame por las columnas "fecha" y "producto".
- `.agg(...)`: Realiza funciones de agregación dentro de cada grupo.
  - `sum("cantidad").alias("total_cantidad")`: Calcula la suma de la columna "cantidad" para cada grupo y la renombra como "total_cantidad".
  - `sum(col("precio") * col("cantidad")).alias("total_ventas")`: Calcula el total de ventas para cada grupo multiplicando el "precio" por la "cantidad" y sumando los resultados, renombrándolo como "total_ventas".
- `.withColumn("promedio_7_dias", avg("total_ventas").over(windowSpec))`: Agrega una nueva columna llamada "promedio_7_dias". Para cada fila, calcula el promedio de la columna "total_ventas" dentro de la ventana definida por `windowSpec`. Esto efectivamente calcula el promedio de las ventas totales de los últimos 7 días para cada producto en cada fecha.
- `.filter(col("total_ventas") > col("promedio_7_dias"))`: Filtra el DataFrame, manteniendo solo las filas donde el valor de la columna "total_ventas" es mayor que el valor de la columna "promedio_7_dias".

11. `resultadoDF.show()`: Muestra el contenido del DataFrame resultante (`resultadoDF`) en la consola. Por defecto, muestra las primeras 20 filas y trunca las columnas que son demasiado anchas.

En resumen, este código realiza las siguientes operaciones:

- Crea un DataFrame con datos de ventas de dos productos a lo largo de varios días.
- Agrupa los datos por fecha y producto para calcular la cantidad total vendida y las ventas totales para cada día y producto.
- Utiliza una función de ventana para calcular el promedio de las ventas totales de los últimos 7 días para cada producto.
- Filtra el DataFrame para mostrar solo los registros donde las ventas totales de un día para un producto son mayores que el promedio de ventas de los 7 días anteriores para ese mismo producto.
- Finalmente, muestra el DataFrame resultante.

Este es un ejemplo muy útil que demuestra cómo realizar análisis de series de tiempo y comparaciones utilizando las funcionalidades de DataFrames y ventanas en PySpark.