```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType

spark = SparkSession.builder.appName("UDFExample").getOrCreate()

# Definir una UDF para categorizar salarios
@udf(returnType=StringType())
def categorize_salary(salary):
    if salary < 3500:
        return "Low"
    elif salary < 4500:
        return "Medium"
    else:
        return "High"

# Crear un DataFrame de ejemplo
data = [("Alice", 3000),
        ("Bob", 4000),
        ("Charlie", 5000)]
df = spark.createDataFrame(data, ["name", "salary"])

# Usar la UDF en una transformación de DataFrame
result = df.withColumn("salary_category", categorize_salary(df.salary))

result.show()

spark.stop()
```

Python

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType

spark = SparkSession.builder.appName("UDFExample").getOrCreate()

# Definir una UDF para categorizar salarios
@udf(returnType=StringType())
def categorize_salary(salary):
    if salary < 3500:
        return "Low"
    elif salary < 4500:
        return "Medium"
    else:
        return "High"

# Crear un DataFrame de ejemplo
data = [("Alice", 3000),
        ("Bob", 4000),
```

```
        ("Charlie", 5000)]
df = spark.createDataFrame(data, ["name", "salary"])

# Usar la UDF en una transformación de DataFrame
result = df.withColumn("salary_category", categorize_salary(df.salary))

result.show()

spark.stop()
```

Este código demuestra el uso de **UDFs (User-Defined Functions)** en PySpark SQL. Las UDFs permiten extender la funcionalidad de Spark SQL definiendo funciones personalizadas en Python (u otros lenguajes soportados) y utilizándolas en las transformaciones de DataFrames.

Desglose del Código:

1. **`from pyspark.sql import SparkSession`**: Importa la clase `SparkSession`.
2. **`from pyspark.sql.functions import udf`**: Importa el decorador `udf`, que se utiliza para registrar una función Python como una UDF de Spark.
3. **`from pyspark.sql.types import StringType`**: Importa el tipo de datos `StringType`, que se utilizará para especificar el tipo de retorno de la UDF.
4. **`spark = SparkSession.builder.appName("UDFExample").getOrCreate()`**: Crea o obtiene una sesión de Spark con el nombre de aplicación "UDFExample".
5. **Definición de la UDF (`categorize_salary`):**
    - **`@udf(returnType=StringType())`**: Este decorador marca la función `categorize_salary` como una UDF de Spark. El argumento `returnType=StringType()` indica que esta función Python devolverá un valor de tipo String cuando se utilice en Spark.
    - **`def categorize_salary(salary):`**: Define una función Python que toma un argumento `salary`.
    - La función contiene una lógica condicional (`if`, `elif`, `else`) para categorizar el salario en tres niveles:
        - Si `salary` es menor que 3500, devuelve "Low".
        - Si `salary` es menor que 4500 (y no menor que 3500), devuelve "Medium".
        - En cualquier otro caso (si `salary` es 4500 o mayor), devuelve "High".
6. **Creación del DataFrame de Ejemplo:**
    - Se crea una lista de tuplas `data` con nombres y salarios.
    - Se crea un DataFrame `df` a partir de esta lista, con las columnas "name" y "salary".
7. **Uso de la UDF:**
    - **`result = df.withColumn("salary_category", categorize_salary(df.salary))`**: Se utiliza el método `withColumn()` para agregar una nueva columna al DataFrame llamada "salary_category". Los valores de esta nueva columna se obtienen aplicando la UDF `categorize_salary` a cada valor de la columna existente "salary". Spark ejecutará la función Python `categorize_salary` para cada fila del DataFrame.
8. **`result.show()`**: Muestra el DataFrame `result`, que ahora incluye la nueva columna "salary_category" con la categoría de salario asignada a cada empleado.
9. **`spark.stop()`**: Detiene la sesión de Spark.

Salida Esperada del Código:

```
+-------+------+---------------+
|   name|salary|salary_category|
+-------+------+---------------+
|  Alice|  3000|            Low|
|    Bob|  4000|         Medium|
|Charlie|  5000|           High|
+-------+------+---------------+
```

Puntos Clave sobre las UDFs:

- **Flexibilidad:** Las UDFs permiten implementar lógica de negocio compleja o personalizada que no está disponible directamente a través de las funciones integradas de Spark SQL.
- **Rendimiento:** Es importante tener en cuenta que las UDFs basadas en Python pueden ser menos eficientes que las funciones integradas de Spark SQL. Esto se debe a la necesidad de serializar y deserializar datos entre la JVM de Spark y el intérprete de Python para cada fila. Para operaciones de alto rendimiento en grandes conjuntos de datos, a menudo es preferible utilizar las funciones integradas de Spark SQL siempre que sea posible.
- **Tipos de Retorno:** Al definir una UDF, es necesario especificar explícitamente el tipo de retorno utilizando el argumento `returnType` del decorador `@udf`. Esto asegura que Spark pueda manejar correctamente los resultados de la UDF dentro del DataFrame.

En resumen, las UDFs son una herramienta poderosa para extender las capacidades de PySpark SQL cuando se requiere lógica personalizada, pero su uso debe considerarse cuidadosamente en términos de rendimiento, especialmente en aplicaciones de gran escala.