## Documentación del Código

El código proporcionado es un flujo de procesamiento de datos usando **Apache Spark** en **Python**, que incluye la limpieza, transformación y carga de datos desde un archivo CSV a una base de datos SQL Server. A continuación, se describen los pasos y las funcionalidades de cada sección del código:

---

### 1. **Importación de Librerías**

```
import pandas as pd
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, to_date, regexp_extract
from pyspark.sql.types import DoubleType
from pyspark.sql import functions as F
from pyspark.sql.types import StructType, StructField, StringType,
DateType
import pymssql
```

- **pandas**: Utilizada para la manipulación de datos en forma de DataFrames (aunque no se utiliza directamente en este código).
- **pyspark**: Se utiliza para el procesamiento de grandes volúmenes de datos en paralelo mediante RDDs (Resilient Distributed Datasets) y DataFrames.
- **pymssql**: Para la conexión con una base de datos SQL Server desde Python.

---

### 2. **Inicialización de Spark**

```
spark =
SparkSession.builder.master("local").appName("DataCleaning").getOrCreate()
```

Se crea una sesión de Spark con el nombre `DataCleaning`, configurando el entorno de ejecución en modo local.

---

### 3. **Lectura del Archivo CSV**

```
file_path =
"C:\\Users\\Usuario\\Documents\\WILLIAM\\DOCUMENTOS\\PERSONALES\\MONOKERA\\se
data_3 = spark.read.csv(file_path, header=True, inferSchema=True)
```

- Se lee un archivo CSV desde el sistema local.
- `header=True` asegura que la primera fila del archivo sea considerada como nombres de las columnas.
- `inferSchema=True` permite que Spark infiera automáticamente los tipos de datos de cada columna.

---

## 4. **Verificación y Limpieza de Datos**

- **Extracción y limpieza de la columna 'id'**:

```
data_3 = data_3.withColumn('id', regexp_extract(col('id'), r'\d+', 0))
data_3_cleaned = data_3.filter(col('id').rlike('^\d+$'))
```

- Se extraen solo los números de la columna `'id'` (si contienen otros caracteres).
- Se filtran las filas donde `'id'` contiene solo números.

---

## 5. **Conversión de Fechas**

```
date_columns = ["claim_date", "payment_date", "policy_start_date",
"policy_end_date"]
for col_name in date_columns:
    if col_name in data_3_cleaned.columns:
        data_3_cleaned = data_3_cleaned.withColumn(col_name,
to_date(col(col_name), "M/d/yyyy"))
```

Se verifica si las columnas de fecha existen y, en caso afirmativo, se convierten a tipo `Date` en el formato `"M/d/yyyy"`.

---

## 6. **Manejo de Valores Nulos**

```
data_cleaned = data_3_cleaned.fillna({
    "insured_age": 0,
    "insured_gender": "Unknown",
    "claim_status": "Unknown",
    "payment_status": "Unknown",
    "payment_method": "Unknown"
})
```

Se rellenan los valores nulos de algunas columnas con valores predeterminados.

---

## 7. **Conversión de Columnas Numéricas**

```
numeric_columns = ["claim_amount", "payment_amount"]
for col_name in numeric_columns:
    if col_name in data_cleaned.columns:
        data_cleaned = data_cleaned.withColumn(col_name,
col(col_name).cast(DoubleType()))
```

Las columnas `claim_amount` y `payment_amount` se convierten al tipo `DoubleType` para asegurar que se manejen como valores numéricos.

---

## 8. **Filtrado y Selección de Columnas**

- **Data de reclamaciones**:

```
data_claim = data_cleaned.filter(F.col('claim_date').isNotNull()) \
    .select(
```

```
        F.col('policy_number').alias('id'),
        'claim_status',
        'claim_date',
        'claim_amount',
        'claim_description'
    )
```

- **Data del asegurado**:

```
data_insured = data_cleaned.groupBy("policy_number").agg(
    F.first('insured_name').alias('insured_name'),
    F.first('insured_gender').alias('insured_gender'),
    F.first('insured_age').alias('insured_age'),
    F.first('insured_address').alias('insured_address'),
    F.first('insured_city').alias('insured_city'),
    F.first('insured_state').alias('insured_state'),
    F.first('insured_postal_code').alias('insured_postal_code'),
    F.first('insured_country').alias('insured_country')
).withColumnRenamed("policy_number", "id")
```

Se filtran y agrupan los datos según diferentes columnas necesarias para crear DataFrames limpios.

---

### 9. **Agrupación y Filtrado de Pagos**

```
payment_columns = [col_name for col_name in data_cleaned.columns if
"payment" in col_name]
data_payments = data_cleaned.select(
    F.col('policy_number').alias('id'),
    *payment_columns
)
data_payments_filtered = data_payments.filter(F.col('payment_status') !=
'Unknown')
```

Se seleccionan solo las columnas que contienen la palabra `"payment"` y se filtran las filas con `payment_status` diferente a `"Unknown"`.

---

### 10. **Agrupación por Póliza**

```
data_premium = data_cleaned.groupBy("policy_number").agg(
    F.first('premium_amount').alias('premium_amount'),
    F.first('deductible_amount').alias('deductible_amount'),
    F.first('coverage_limit').alias('coverage_limit')
).withColumnRenamed("policy_number", "id")
```

Se agrupan los datos por `policy_number` y se obtienen los primeros valores de las columnas `premium_amount`, `deductible_amount` y `coverage_limit`.

---

### 11. **Conexión a SQL Server y Carga de Datos**

```
conn = pymssql.connect(
    server="LAPTOP-O07NV287",
```

```
    user="williamxlp",
    password="ADDDDDDD",
    database="monokera"
)
```

Se establece una conexión con la base de datos SQL Server utilizando las credenciales correspondientes.

**Función de carga de datos**:

```
def load_to_sql(df, table_name):
    try:
        for row in df.collect():
            query = f"INSERT INTO {table_name} ({', '.join(df.columns)})
VALUES ({', '.join(['%s'] * len(df.columns))})"
            cursor.execute(query, tuple(row))
        conn.commit()
        print(f"Datos cargados correctamente en la tabla {table_name}.")
    except Exception as e:
        print(f"Error al cargar datos en la tabla {table_name}: {e}")
        conn.rollback()
```

La función `load_to_sql` carga los datos de un DataFrame en una tabla de SQL Server.

---

### 12. **Cierre de Conexión**

```
cursor.close()
conn.close()
```

Finalmente, se cierra la conexión con la base de datos SQL Server.

---

## Resumen

Este flujo de trabajo en PySpark realiza la lectura de datos desde un archivo CSV, limpia y transforma los datos, y luego los carga en diferentes tablas en una base de datos SQL Server. Los pasos incluyen la conversión de tipos de datos, el manejo de valores nulos, y la agrupación de datos por póliza.