```python
import requests
from pyspark.sql import SparkSession

# Crear una sesión de Spark
spark = SparkSession.builder \
    .appName("CSV File Reading") \
    .getOrCreate()

# URL del archivo CSV (ejemplo: dataset de Iris)
csv_url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

# Descargar el archivo CSV
response = requests.get(csv_url)
with open("/tmp/iris.csv", "wb") as f:
    f.write(response.content)

# Leer el archivo CSV descargado
iris_df = spark.read.csv("/tmp/iris.csv",
                         header=False,
                         inferSchema=True)

# Asignar nombres a las columnas
column_names = ["sepal_length", "sepal_width", "petal_length", "petal_width",
"class"]
iris_df = iris_df.toDF(*column_names)

# Mostrar el esquema y los primeros registros
iris_df.printSchema()
iris_df.show(5)

# Escritura de CSV
iris_df.write.csv("hdfs://localhost:9000/user/hdfs/iris", header=True,
mode="overwrite")
```

Este código demuestra cómo leer un archivo CSV desde una URL, descargarlo localmente y luego cargarlo en un DataFrame de PySpark. Finalmente, intenta escribir el DataFrame en HDFS. Vamos a analizarlo paso a paso:

1. **`import requests`**: Importa la librería `requests`, que se utiliza para realizar solicitudes HTTP, como descargar el archivo CSV desde la URL.
2. **`from pyspark.sql import SparkSession`**: Importa la clase `SparkSession`, el punto de entrada para trabajar con DataFrames en PySpark.
3. **`spark = SparkSession.builder \`**: Inicia la construcción de una `SparkSession`.
    - **`.appName("CSV File Reading") \`**: Establece el nombre de la aplicación Spark.
    - **`.getOrCreate()`**: Obtiene una `SparkSession` existente o crea una nueva.

4. **`csv_url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"`**: Define la URL del archivo CSV que se va a leer (el famoso dataset de Iris).
5. **`response = requests.get(csv_url)`**: Realiza una solicitud HTTP GET a la URL del archivo CSV y guarda la respuesta en la variable `response`.
6. **`with open("/tmp/iris.csv", "wb") as f:`**: Abre un archivo local llamado `iris.csv` en modo de escritura binaria (`"wb"`) dentro del directorio `/tmp/`. El uso de `with open(...)` asegura que el archivo se cierre correctamente después de la escritura.
7. **`f.write(response.content)`**: Escribe el contenido de la respuesta HTTP (el contenido del archivo CSV) en el archivo local `/tmp/iris.csv`.
8. **`iris_df = spark.read.csv(...)`**: Utiliza el lector de CSV de Spark (`spark.read.csv()`) para leer el archivo CSV local que se acaba de descargar.
    - **`"/tmp/iris.csv"`**: Especifica la ruta del archivo CSV que se va a leer.
    - **`header=False`**: Indica que el archivo CSV no tiene una fila de encabezado.
    - **`inferSchema=True`**: Le dice a Spark que intente inferir el tipo de datos de cada columna basándose en su contenido. Esto puede ser útil pero a veces puede no ser completamente preciso, especialmente para tipos de datos ambiguos.
9. **`column_names = [...]`**: Define una lista de nombres de columna que se asignarán al DataFrame. Dado que el archivo CSV no tiene encabezado, es necesario proporcionar los nombres de las columnas explícitamente.
10. **`iris_df = iris_df.toDF(*column_names)`**: Renombra las columnas del DataFrame `iris_df` utilizando los nombres definidos en la lista `column_names`. El operador `*` se utiliza para expandir la lista en argumentos separados para la función `toDF()`.
11. **`iris_df.printSchema()`**: Imprime el esquema del DataFrame, mostrando los nombres de las columnas y sus tipos de datos inferidos.
12. **`iris_df.show(5)`**: Muestra las primeras 5 filas del DataFrame para verificar que los datos se hayan leído correctamente.
13. **`iris_df.write.csv("hdfs://localhost:9000/user/hdfs/iris", header=True, mode="overwrite")`**: Intenta escribir el DataFrame `iris_df` en formato CSV en el sistema de archivos distribuido Hadoop (HDFS).
    - **`"hdfs://localhost:9000/user/hdfs/iris"`**: Especifica la ruta en HDFS donde se escribirán los archivos CSV. `hdfs://localhost:9000` indica la dirección del NameNode de HDFS (asumiendo que se está ejecutando localmente en el puerto 9000). `/user/hdfs/iris` es el directorio dentro de HDFS donde se guardarán los archivos.
    - **`header=True`**: Indica que se debe incluir una fila de encabezado en los archivos CSV escritos, utilizando los nombres de las columnas del DataFrame.
    - **`mode="overwrite"`**: Especifica el modo de escritura. `"overwrite"` significa que si el directorio de destino ya existe, su contenido se eliminará antes de escribir los nuevos archivos.

Consideraciones Importantes (especialmente en tu contexto en España):

- **Dependencia de `requests`:** Asegúrate de que la librería `requests` esté instalada en tu entorno de Python. Si no lo está, puedes instalarla con `pip install requests`.
- **Acceso a la URL:** Tu entorno debe tener acceso a la URL especificada para descargar el archivo CSV.
- **Permisos de Escritura en `/tmp/`:** El proceso de Python debe tener permisos para escribir en el directorio `/tmp/` de tu sistema local.

- **Configuración de HDFS:** La parte de la escritura a HDFS (`iris_df.write.csv(...)`) solo funcionará si tienes un clúster Hadoop en funcionamiento y accesible en `hdfs://localhost:9000`.
    - Si no tienes un clúster Hadoop configurado localmente, esta parte del código generará un error de conexión o un error relacionado con el sistema de archivos HDFS.
    - Asegúrate de que el servicio NameNode de HDFS esté en ejecución y accesible en la dirección y puerto especificados.
    - Verifica que el usuario que ejecuta el script de PySpark tenga permisos para escribir en el directorio `/user/hdfs/iris` en HDFS.

Si no tienes un clúster Hadoop configurado y solo quieres trabajar con el DataFrame localmente, puedes comentar o eliminar la línea de escritura a HDFS.

Este es un flujo de trabajo común para leer datos externos en PySpark, ya sea desde la web o desde un sistema de archivos local, realizar transformaciones y análisis, y luego, potencialmente, escribir los resultados en un sistema de almacenamiento distribuido como HDFS.