```
import org.apache.spark.sql.{SparkSession, DataFrame}
import org.apache.spark.sql.functions._
import org.apache.spark.storage.StorageLevel
import java.net.URL
import java.io.{File, FileOutputStream}

object CompleteExample {
  def main(args: Array[String]): Unit = {
    // Crear una sesión de Spark
    val spark = SparkSession.builder()
      .appName("CompleteExample")
      .master("local[*]")
      .config("spark.sql.warehouse.dir", "spark-warehouse")
      .config("spark.executor.memory", "2g")
      .config("spark.driver.memory", "2g")
      .config("spark.sql.shuffle.partitions", "8")
      .getOrCreate()

    import spark.implicits._

    // URL del archivo Parquet a descargar (yellow taxi trip records)
    val url = "https://d37ci6vzurychx.cloudfront.net/trip-data/yellow_tripdata_2022-
01.parquet"

    // Descargar el archivo Parquet
    val localFilePath = "yellow_tripdata_2022-01.parquet"
    downloadFile(url, localFilePath)
    println(s"Archivo Parquet descargado en: $localFilePath")

    // Leer el archivo Parquet
    val dfTaxi = spark.read.parquet(localFilePath)
```

```
// Mostrar el esquema y algunas estadísticas básicas
dfTaxi.printSchema()
dfTaxi.describe().show()

// Cachear el DataFrame para mejorar el rendimiento de operaciones subsecuentes
dfTaxi.persist(StorageLevel.MEMORY_AND_DISK)

// Realizar algunas transformaciones y consultas
val dfProcessed = dfTaxi
  .withColumn("trip_duration_minutes",
    (unix_timestamp($"tpep_dropoff_datetime") -
unix_timestamp($"tpep_pickup_datetime")) / 60)
  .withColumn("price_per_mile", when($"trip_distance" > 0, $"total_amount" /
$"trip_distance").otherwise(0))
  .withColumn("day_of_week", date_format($"tpep_pickup_datetime", "EEEE"))

// Crear una vista temporal para usar SQL
dfProcessed.createOrReplaceTempView("taxi_trips")

// Realizar una consulta SQL compleja
val resultSQL = spark.sql("""
  SELECT
    day_of_week,
    AVG(trip_duration_minutes) as avg_duration,
    AVG(trip_distance) as avg_distance,
    AVG(total_amount) as avg_amount,
    AVG(price_per_mile) as avg_price_per_mile
  FROM taxi_trips
  WHERE trip_distance > 0 AND trip_duration_minutes BETWEEN 5 AND 120
  GROUP BY day_of_week
  ORDER BY avg_amount DESC
```

```
    """)

    println("Resumen por dia de la semana:")
    resultSQL.show()

    // Análisis adicional usando la API de DataFrame
    val topPickupLocations = dfProcessed
      .groupBy("PULocationID")
      .agg(
        count("*").as("total_pickups"),
        avg("total_amount").as("avg_fare")
      )
      .orderBy(desc("total_pickups"))
      .limit(5)

    println("Top 5 ubicaciones de recogida:")
    topPickupLocations.show()

    // Guardar resultados en formato Parquet
    resultSQL.write.mode("overwrite").parquet("taxi_summary_by_day.parquet")
    topPickupLocations.write.mode("overwrite").parquet("top_pickup_locations.parquet")

    // Liberar el caché
    dfTaxi.unpersist()

    // Detener la sesión de Spark
    spark.stop()
  }

  def downloadFile(url: String, localFilePath: String): Unit = {
    val connection = new URL(url).openConnection()
```

```
    val inputStream = connection.getInputStream
    val outputStream = new FileOutputStream(new File(localFilePath))

    try {
      val buffer = new Array[Byte](4096)
      var bytesRead = inputStream.read(buffer)
      while (bytesRead != -1) {
        outputStream.write(buffer, 0, bytesRead)
        bytesRead = inputStream.read(buffer)
      }
    } finally {
      inputStream.close()
      outputStream.close()
    }
  }
}
```

Explicación del Código `CompleteExample`

Este código Scala utiliza la API de Apache Spark para realizar un análisis de un conjunto de datos de viajes en taxi amarillo de la ciudad de Nueva York, almacenado en formato Parquet. El programa realiza las siguientes acciones:

1. **Configuración de la Sesión de Spark:**
   - Crea o obtiene una sesión de Spark (`SparkSession`), que es el punto de entrada para trabajar con Spark SQL.
   - Configura varios parámetros de Spark, como el nombre de la aplicación, el modo de ejecución (local), el directorio del almacén de Spark SQL, la memoria asignada a los executors y al driver, y el número de particiones para las operaciones de shuffle.
2. **Descarga del Archivo Parquet:**
   - Define la URL del archivo Parquet que contiene los datos de los viajes en taxi de enero de 2022.
   - Llama a la función `downloadFile` para descargar el archivo desde la URL y guardarlo localmente con el nombre `yellow_tripdata_2022-01.parquet`.

- La función `downloadFile` utiliza las clases `java.net.URL` y `java.io.{File, FileOutputStream}` para abrir una conexión a la URL, leer el contenido y escribirlo en un archivo local.

3. **Lectura del Archivo Parquet:**
   - Utiliza `spark.read.parquet(localFilePath)` para leer el archivo Parquet descargado y crear un DataFrame llamado `dfTaxi`. Spark infiere automáticamente el esquema del archivo Parquet.
4. **Inspección del DataFrame:**
   - `dfTaxi.printSchema()` muestra la estructura del DataFrame, incluyendo los nombres y tipos de datos de las columnas.
   - `dfTaxi.describe().show()` calcula y muestra estadísticas descriptivas básicas (recuento, media, desviación estándar, mínimo y máximo) para las columnas numéricas del DataFrame.
5. **Cacheo del DataFrame:**
   - `dfTaxi.persist(StorageLevel.MEMORY_AND_DISK)` persiste el DataFrame `dfTaxi` en la memoria y en el disco. Esto es una optimización para mejorar el rendimiento de las operaciones posteriores que utilicen este DataFrame, ya que los datos se podrán acceder más rápidamente desde la memoria (si está disponible) en lugar de tener que leerlos del disco nuevamente.
6. **Transformaciones del DataFrame:**
   - Se crea un nuevo DataFrame llamado `dfProcessed` a partir de `dfTaxi` añadiendo tres nuevas columnas:
     - `trip_duration_minutes`: Calcula la duración del viaje en minutos restando la marca de tiempo de recogida de la marca de tiempo de entrega y dividiendo el resultado por 60. Se utiliza la función `unix_timestamp` para convertir las marcas de tiempo a segundos.
     - `price_per_mile`: Calcula el precio por milla dividiendo el importe total (`total_amount`) por la distancia del viaje (`trip_distance`). Se utiliza una condición `when` para evitar la división por cero en caso de que la distancia del viaje sea cero, estableciendo el precio por milla en 0 en ese caso.
     - `day_of_week`: Extrae el nombre del día de la semana (por ejemplo, "Monday", "Tuesday") de la marca de tiempo de recogida (`tpep_pickup_datetime`) utilizando la función `date_format`.
7. **Creación de una Vista Temporal:**
   - `dfProcessed.createOrReplaceTempView("taxi_trips")` crea una vista temporal llamada `taxi_trips` basada en el DataFrame `dfProcessed`. Esto permite ejecutar consultas SQL directamente contra el DataFrame utilizando la sintaxis SQL estándar.
8. **Consulta SQL Compleja:**
   - Se ejecuta una consulta SQL contra la vista temporal `taxi_trips` para calcular el promedio de la duración del viaje, la distancia, el importe total y el precio por milla, agrupado por el día de la semana.
   - Se aplica un filtro (`WHERE`) para considerar solo los viajes con una distancia mayor que 0 y una duración entre 5 y 120 minutos (para eliminar posibles valores atípicos o errores en los datos).

    - Los resultados se agrupan por `day_of_week` y se ordenan por el importe promedio (`avg_amount`) en orden descendente.
    - Los resultados de esta consulta se almacenan en el DataFrame `resultSQL` y luego se muestran en la consola.
9. **Análisis Adicional con la API de DataFrame:**
    - Se realiza un análisis adicional utilizando la API de DataFrame para encontrar las 5 principales ubicaciones de recogida (`PULocationID`) en función del número total de recogidas y el importe promedio de la tarifa para cada ubicación.
    - Se agrupan los datos por `PULocationID`, se cuenta el número de recogidas (`count("*")`) y se calcula el promedio del importe total (`avg("total_amount")`).
    - Los resultados se ordenan por el número total de recogidas en orden descendente y se limitan a las 5 primeras filas.
    - Los resultados de este análisis se almacenan en el DataFrame `topPickupLocations` y luego se muestran en la consola.
10. **Guardado de Resultados en Formato Parquet:**
    - Los DataFrames resultantes del análisis (`resultSQL` y `topPickupLocations`) se guardan en formato Parquet en los directorios `taxi_summary_by_day.parquet` y `top_pickup_locations.parquet`, respectivamente.
    - Se utiliza el modo `"overwrite"` para que si estos directorios ya existen, se eliminen y se creen nuevos con los resultados actuales.
11. **Liberación del Caché:**
    - `dfTaxi.unpersist()` elimina el DataFrame `dfTaxi` de la caché de memoria y disco, liberando los recursos utilizados.
12. **Detención de la Sesión de Spark:**
    - `spark.stop()` detiene la sesión de Spark, liberando todos los recursos asociados a la aplicación.

Posible Resultado del Código

El resultado del código se mostrará en la consola en dos partes principales:

1. Resumen por día de la semana:

Esta sección mostrará una tabla con el promedio de la duración del viaje, la distancia, el importe total y el precio por milla, agrupado por cada día de la semana. Los días se mostrarán ordenados de forma descendente según el importe promedio del viaje. El resultado podría tener un formato similar a este:

```
Resumen por dia de la semana:
+-----------+------------+------------+----------+------------------+
|day_of_week|avg_duration|avg_distance|avg_amount|avg_price_per_mile|
+-----------+------------+------------+----------+------------------+
|   Saturday|       17.05|        3.52|     18.75|              5.32|
|     Sunday|       16.80|        3.45|     17.90|              5.19|
|    Friday|       17.50|        3.60|     17.85|              5.05|
|    Tuesday|       15.95|        3.20|     16.50|              5.15|
|  Wednesday|       16.10|        3.25|     16.40|              5.08|
|   Thursday|       16.30|        3.30|     16.35|              4.95|
```

```
|     Monday|       16.00|        3.15|     16.20|                5.10|
+-----------+------------+------------+----------+-------------------+
```

Nota: Los valores exactos variarán dependiendo de los datos del archivo Parquet descargado.

2. Top 5 ubicaciones de recogida:

Esta sección mostrará una tabla con las 5 ubicaciones de recogida (`PULocationID`) con el mayor número total de recogidas y el importe promedio de la tarifa para cada una de estas ubicaciones. El resultado podría tener un formato similar a este:

```
Top 5 ubicaciones de recogida:
+------------+-------------+--------+
|PULocationID|total_pickups|avg_fare|
+------------+-------------+--------+
|         237|        35000|   15.50|
|         161|        32000|   18.20|
|         186|        30500|   12.75|
|         230|        29000|   16.80|
|         170|        27500|   14.30|
+------------+-------------+--------+
```

Nota: Los `PULocationID` y los valores exactos de `total_pickups` y `avg_fare` dependerán de los datos del archivo Parquet.

Además de la salida en la consola, el programa también creará dos directorios en el sistema de archivos local (en la ubicación desde donde se ejecute el programa):

- `taxi_summary_by_day.parquet`: Contendrá los resultados del resumen por día de la semana guardados en formato Parquet.
- `top_pickup_locations.parquet`: Contendrá los resultados de las 5 principales ubicaciones de recogida guardados en formato Parquet.

Estos archivos Parquet podrán ser utilizados posteriormente para realizar más análisis o integrarse con otros sistemas de procesamiento de datos.