Opzet

Spark-object aanmaken + enkel de nodige tekst in de console uitprinten:

```
spark = SparkSession.builder().appName("WineSQL").master("local[*]").getOrCreate();
spark.sparkContext().setLogLevel("ERROR");
```

CSV-inlezen met delimiter ';' + schema overnemen + header overnemen

```
Dataset<Row> df = spark.read()
        .option("header", true)
        .option("delimiter", ";")
        .option("inferSchema", true)
        .csv(INPUT_PATH);

System.out.println(df.schema());
```

Schema niet overnemen: 1. Structured fields maken met 'fields' 2. Schema maken 3. Dataframe maken (read / readStream)

```
List<StructField>  fields = Arrays.asList(
    new StructField []  {
        DataTypes.createStructField("loglevel", DataTypes.StringType, false),
        DataTypes.createStructField("source", DataTypes.IntegerType, false),
        DataTypes.createStructField("day", DataTypes.StringType, false),
        DataTypes.createStructField("time", DataTypes.StringType, false),
        DataTypes.createStructField("message", DataTypes.StringType, true)
        }
    );

StructType schema = DataTypes.createStructType(fields);


Dataset<Row> messages = spark.readStream()
    .schema(schema)
    .option("header", false)
    .csv(CSV_PATH);
```

In-memory dataset maken: 1. Structured Fields maken 2. Schema maken met structured fields 3. Rijen toevoegen aan een ArrayList van Row-object 4. Dataframe maken met rijen(3) en het schema(2).

```
private static Dataset<Row> createInMemoryDataset(SparkSession spark) {

        List<StructField>  fields = Arrays.asList(
                new StructField []  {
                    DataTypes.createStructField("source", DataTypes.IntegerType, false),
                    DataTypes.createStructField("sourceString", DataTypes.StringType, false),
                }
                );

        StructType schema = DataTypes.createStructType(fields);

        List<Row> rows = new ArrayList<Row>();
        rows.add(RowFactory.create(0, "web"));
        rows.add(RowFactory.create(1, "e-mail"));
        rows.add(RowFactory.create(2, "database"));
        rows.add(RowFactory.create(3, "firewall"));
        rows.add(RowFactory.create(4, "ml-model"));
```

```
        return spark.createDataFrame(rows, schema);
    }
```

Analyse

Aantal rijen uitprinten (kan je ook gebruiken na een filter)

```
System.out.println("Het aantal in de dataframe is: " + df.count());
```

```
filtered_df = df.where(...)
System.out.println("Er zijn " + filtered_df.count() + " aantal rijen in de gefilterde dataset");
```

JOIN maken

```
messages = sourceSystemsDf.join(messages, "source")
            .drop("source");
```

Bewerkingen op Dataframe

Rijen uitfilteren (string-waarde)

```
df = df.where(not(col("alcohol").like("??")));
```

Rijen uitfilteren (tussen range van waarden)

```
df = df.where(col("pH").$greater$eq(2.5).and(col("pH").$less$eq(4)));
```

Getal afronden naar op 3 cijfers van de komma

```
df.groupBy(col("kolom")).agg(bround(col("kolom").cast("double")), 3)
```

Waarde aanpassen als het in een lijst van waarden behoort: * Als loglevel error of fatal is, dan high. * Als loglevel iets anders is, dan is het low.

```
messages = messages.withColumn("loglevel",
    when(col("loglevel").isin("ERROR", "FATAL"), "HIGH").otherwise("LOW"));
```

Watermarking: 1. Datum-kolom casten naar Timestamp-type 2. Watermark toevoegen 3. Window-kolom toevoegen

```
messages = messages.withColumn("dayAndTime", concat_ws(" ", col("day"), col("time")).cast("TimeStamp"))
        .withWatermark("watermark",  WATERMARK_DELAY)
        .withColumn("window", window(col("dayAndTime"), "5 minutes"));
```

Groeperen

Groeperen op categorie en aantal voorkomens (count) per categorie tellen. Rangschikken volgens aantal (count)

```
df.groupBy(col("categories"))
                .agg(count("categories").as("aantalVoorkomens"))
                .orderBy(desc("aantalVoorkomens"))
                .show();
```

Groeperen op twee kolommen:

```
df.groupBy(col("kolom1","kolom2")).agg(count("kolom1"));
```

Groeperen per time-window (watermarking)

```
messages = messages
    .withColumn("HIGH", when(col("loglevel").equalTo("HIGH"), 1).otherwise(0))
    .withColumn("LOW", when(col("loglevel").equalTo("LOW"), 1).otherwise(0))
    .groupBy(col("window"))
    .agg(sum("LOW").as("LOW"), sum("HIGH").as("HIGH"));
```

UDF

UDF maken (registreren)

```
spark.udf().register("convertPH", createUDF(), DataTypes.StringType);
```

UDF schrijven: - 1 invoerveld (getal); 1 uitvoerveld (tekst) - invoerveld moet in de call(..) functie staan - type van het uitvoerveld is de return-waarde (voor call(...))

```
private static UDF1<Double, String> createUDF() {
        return new UDF1<Double, String>() {
            public String call(Double pH) throws Exception {

                if (pH >= 3.25) {
                    return "HIGH";
                } else if (pH >= 3.1) {
                    return "MEDIUM";
                } else {
                    return "LOW";
                }
            }
        };
    }
```

UDF gebruiken voor een (nieuwe) kolom.

```
private static void showPHReport(Dataset<Row> df) {
df.withColumn("pH_CAT", call_udf("convertPH", col("pH")))
        .groupBy("pH_CAT")
        .agg(count("pH_CAT").as("aantalVoorkomensPerCat"))
        .show();
}
```

Dataframe uitschrijven / opslaan

Dataframe naar bestand uitschrijven (hier parquet)

```
        df.write()
            .mode(SaveMode.Overwrite)
            .parquet(OUTPUT_PATH); // aanpassen naar csv / txt
```

Stream uitschrijven

```
StreamingQuery query = null;
query = messages
    .writeStream()
    .format("console") // in console
    .outputMode(OutputMode.Append()) // nieuwe lijnen
    .trigger(Trigger.ProcessingTime(10, TimeUnit.SECONDS)) // iedere tien seconden
    .start();

query.awaitTermination();
```