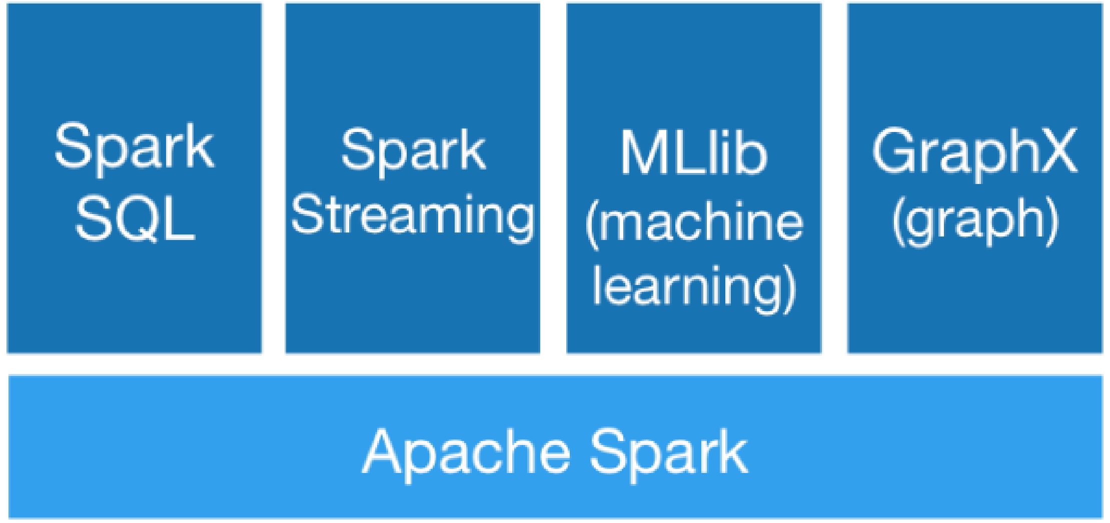
Basic introduction to PySpark

BUILDING DATA ENGINEERING PIPELINES IN PYTHON

Oliver Willekens
Data Engineer at Data Minded

What is Spark?

- A fast and general engine for large-scale data processing
- 4 libraries built on top of Spark core:



- API in several languages
  - Java, Scala, Python (“PySpark”), R

When to use Spark

Spark is used for:

- Data processing at scale
- Interactive analytics
- Machine learning

Spark is **not** used for:

- When you have only little data
- When you have only simple operations

Business case: finding the perfect diaper

Find the perfect diaper based on:

- qua**l**itative attributes e.g. comfort
- qua**nt**itative attributes e.g. price

Scraped data available:

- *prices.csv*: pricing details per model per store
- *ratings.csv*: user ratings per model

Starting the Spark analytics engine

```
from pyspark.sql import SparkSession


spark = SparkSession.builder.getOrCreate()
```

Reading a CSV file

```
prices = spark.read.csv("mnt/data_lake/landing/prices.csv")
prices.show()
```

```
+---------+-----------+------------+-----+--------+--------+----------+
|      _c0|        _c1|         _c2|  _c4|     _c5|     _c6|       _c7|
+---------+-----------+------------+-----+--------+--------+----------+
|    store|countrycode|       brand|price|currency|quantity|      date|
|     Aldi|         BE|Diapers-R-Us|  6.8|     EUR|      40|2019-02-03|
|Carrefour|         FR|     Nappy-k|  5.7|     EUR|      30|2019-02-06|
|    Tesco|        IRL|     Pampers|  6.3|     EUR|      35|2019-02-07|
|       DM|         DE|     Huggies|  6.8|     EUR|      40|2019-02-01|
+---------+-----------+------------+-----+--------+--------+----------+
```

Reading a CSV file with headers

```
prices = spark.read.options(header="true").csv("mnt/data_lake/landing/prices.csv")
prices.show()
```

```
+---------+-----------+------------+-----+--------+--------+----------+
|    store|countrycode|       brand|price|currency|quantity|      date|
+---------+-----------+------------+-----+--------+--------+----------+
|     Aldi|         BE|Diapers-R-Us|  6.8|     EUR|      40|2019-02-03|
|Carrefour|         FR|     Nappy-k|  5.7|     EUR|      30|2019-02-06|
|    Tesco|        IRL|     Pampers|  6.3|     EUR|      35|2019-02-07|
|       DM|         DE|     Huggies|  6.8|     EUR|      40|2019-02-01|
+---------+-----------+------------+-----+--------+--------+----------+
```

Automatically inferred data types

```python
from pprint import pprint
pprint(prices.dtypes)
```

```
[('store', 'string'),
('countrycode', 'string'),
('brand', 'string'),
('price', 'string'),
('currency', 'string'),
('quantity', 'string'),
('date', 'string')]
```

Enforcing a schema

```
schema = StructType([StructField("store", StringType(), nullable=False),
                     StructField("countrycode", StringType(), nullable=False),
                     StructField("brand", StringType(), nullable=False),
                     StructField("price", FloatType(), nullable=False),
                     StructField("currency", StringType(), nullable=True),
                     StructField("quantity", IntegerType(), nullable=True),
                     StructField("date", DateType(), nullable=False)])

prices = spark.read.options(header="true").schema(schema).csv("mnt/data_lake/landing/prices.csv")
print(prices.dtypes)
```

```
[('store', 'string'), ('countrycode', 'string'), ('brand', 'string'),
('price', 'float'), ('currency', 'string'), ('quantity', 'int'), ('date', 'date')]
```

Let's practice!

BUILDING DATA ENGINEERING PIPELINES IN PYTHON

Cleaning data

BUILDING DATA ENGINEERING PIPELINES IN PYTHON

Oliver Willekens
Data Engineer at Data Minded

Reasons to clean data

Most data sources are not ready for analytics. This could be due to:

- Incorrect data types
- Invalid rows
- Incomplete rows
- Badly chosen placeholders

Can we automate data cleaning?

Data cleaning depends on the context

- Can our system cope with data that is 95% clean and 95% complete?
- What are the implicit standards in the company?
  - regional datetimes vs. UTC
  - column naming conventions
  - ...
- What are the low-level details of the systems?
  - representation of unknown / incomplete data
  - ranges for numerical values
  - meaning of fields

Selecting data types

Data type	Value type in Python
`ByteType`	Good for numbers that are within the range of -128 to 127.
`ShortType`	Good for numbers that are within the range of -32768 to 32767.
`IntegerType`	Good for numbers that are within the range of-2147483648 to 2147483647.
`FloatType`	float
`StringType`	string
`BooleanType`	bool
`DateType`	datetime.date

Badly formatted source data

```
cat bad_data.csv  # prints the entire file on stdout
```

```
store,countrycode,brand,price,currency,quantity,date
Aldi,BE,Diapers-R-Us,6.8,EUR,40,2019-02-03
  -----------------------------------
Kruidvat,NL,Nappy-k,5.6,EUR,40,2019-02-15
DM,AT,Huggies,7.2,EUR,40,2019-02-01
```

Spark's default handling of bad source data

```
prices = spark.read.options(header="true").csv('landing/prices.csv')
prices.show()
```

```
+--------------------+-----------+------------+-----+--------+--------+----------+
|               store|countrycode|       brand|price|currency|quantity|      date|
+--------------------+-----------+------------+-----+--------+--------+----------+
|                Aldi|         BE|Diapers-R-Us|  6.8|     EUR|      40|2019-02-03|
|-----------------...|       null|        null| null|    null|    null|      null|
|            Kruidvat|         NL|     Nappy-k|  5.6|     EUR|      40|2019-02-15|
|                  DM|         AT|     Huggies|  7.2|     EUR|      40|2019-02-01|
+--------------------+-----------+------------+-----+--------+--------+----------+
```

Handle invalid rows

```
prices = (spark
          .read
          .options(header="true", mode="DROPMALFORMED")
          .csv('landing/prices.csv'))
```

```
+--------+-----------+------------+-----+--------+--------+----------+
|   store|countrycode|       brand|price|currency|quantity|      date|
+--------+-----------+------------+-----+--------+--------+----------+
|    Aldi|         BE|Diapers-R-Us|  6.8|     EUR|      40|2019-02-03|
|Kruidvat|         NL|     Nappy-k|  5.6|     EUR|      40|2019-02-15|
|      DM|         AT|     Huggies|  7.2|     EUR|      40|2019-02-01|
+--------+-----------+------------+-----+--------+--------+----------+
```

The significance of null

```
store,countrycode,brand,price,currency,quantity,date
Aldi,BE,Diapers-R-Us,6.8,EUR,40,2019-02-03
Kruidvat,,Nappy-k,5.6,EUR,,2019-02-15
```

```
prices = (spark.read.options(header="true")
          .schema(schema)
          .csv('/landing/prices_with_incomplete_rows.csv'))
prices.show()
```

```
+--------+-----------+------------+-----+--------+--------+----------+
|   store|countrycode|       brand|price|currency|quantity|      date|
+--------+-----------+------------+-----+--------+--------+----------+
|    Aldi|         BE|Diapers-R-Us|  6.8|     EUR|      40|2019-02-03|
|Kruidvat|       null|     Nappy-k|  5.6|     EUR|    null|2019-02-15|
+--------+-----------+------------+-----+--------+--------+----------+
```

Supplying default values for missing data

```
prices.fillna(25, subset=['quantity']).show()
```

```
+--------+-----------+------------+-----+--------+--------+----------+
|   store|countrycode|       brand|price|currency|quantity|      date|
+--------+-----------+------------+-----+--------+--------+----------+
|    Aldi|         BE|Diapers-R-Us|  6.8|     EUR|      40|2019-02-03|
|Kruidvat|       null|     Nappy-k|  5.6|     EUR|      25|2019-02-15|
+--------+-----------+------------+-----+--------+--------+----------+
```

Badly chosen placeholders

Example: contracts of employees

```
employees = spark.read.options(header="true").schema(schema).csv('employees.csv')
```

```
+-------------+----------+----------+----------+
|employee_name|department|start_date|  end_date|
+-------------+----------+----------+----------+
|          Bob| marketing|2012-06-01|2016-05-02|
|        Alice|        IT|2018-04-03|9999-12-31|
+-------------+----------+----------+----------+
```

Conditionally replace values

```
from pyspark.sql.functions import col, when
from datetime import date, timedelta
one_year_from_now = date.today().replace(year=date.today().year + 1)
better_frame = employees.withColumn("end_date",
    when(col("end_date") > one_year_from_now, None).otherwise(col("end_date")))
better_frame.show()
```

```
+-------------+----------+----------+----------+
|employee_name|department|start_date|  end_date|
+-------------+----------+----------+----------+
|          Bob| marketing|2012-06-01|2016-05-02|
|        Alice|        IT|2018-04-03|      null|
+-------------+----------+----------+----------+
```

Let's practice!

BUILDING DATA ENGINEERING PIPELINES IN PYTHON

Transforming data with Spark

BUILDING DATA ENGINEERING PIPELINES IN PYTHON

Oliver Willekens
Data Engineer at Data Minded

Why do we need to transform data?

Process:

1. Collect data
2. “Massage” data: involves cleaning and business logic
3. **Derive insights**

Example:

1. Collect data from *booking.com* and *hotels.com*.
2. Standardize hotel names, normalizing review scores.
3. Join datasets, filter on location and rank results.

Common data transformations

1. **Filtering rows**

```
country | purchase_order
________|_______________
India   | 87254800912
Ukraine | 32498562223
```

European purchases?

```
country | purchase_order
________|_______________
Ukraine | 32498562223
```

Common data transformations

1. **Filtering rows**
2. **Selecting and renaming columns**

```
country | purchase_order | store_keep
________|________________|___________
Ukraine | 32498562223    | Oksana D.
Spain   | 74398221190    | Pedro R.
```

->

```
country_of_purchase | purchase_order
____________________|________________
Ukraine             | 32498562223
Spain               | 74398221190
```

Common data transformations

1. **Filtering rows**
2. **Selecting and renaming columns**
3. **Grouping and aggregation**

```
country | purchase_order | price            country | total_revenue
________|________________|______            ________|______________
Ukraine | 32498562223    | $12      =>      Ukraine | $12
Spain   | 74398221190    | $54              Spain   | $80
Spain   | 49876776100    | $26
```

Common data transformations

1. **Filtering rows**
2. **Selecting and renaming columns**
3. **Grouping and aggregation**
4. **Joining multiple datasets**

```
country | purchase_order | price          purchase_order | category
________|________________|______          _______________|_________
Ukraine | 32498562223    | $12      +     32498562223    | food
Spain   | 74398221190    | $54            49876776100    | electronics
Spain   | 49876776100    | $26            74398221190    | clothing
```

Common data transformations

1. Filtering rows
2. Selecting and renaming columns
3. Grouping and aggregation
4. Joining multiple datasets
5. Ordering results

```
country | purchase_order | price          country | purchase_order | price
________|________________|______          ________|________________|______
Spain   | 74398221190    | $26      =>    Ukraine | 32498562223    | $12
Ukraine | 32498562223    | $12            Spain   | 74398221190    | $26
Spain   | 49876776100    | $54            Spain   | 49876776100    | $54
```

Recall the prices dataset

```
prices = spark.read.options(header="true").schema(schema).csv('landing/prices.csv')
```

```
+---------+-----------+------------+-----+--------+--------+----------+
|    store|countrycode|       brand|price|currency|quantity|      date|
+---------+-----------+------------+-----+--------+--------+----------+
|     Aldi|         BE|Diapers-R-Us|  6.8|     EUR|      40|2019-02-03|
| Kruidvat|         BE|     Nappy-k|  4.8|     EUR|      30|2019-01-28|
|Carrefour|         FR|     Nappy-k|  5.7|     EUR|      30|2019-02-06|
|    Tesco|        IRL|     Pampers|  6.3|     EUR|      35|2019-02-07|
|       DM|         DE|     Huggies|  6.8|     EUR|      40|2019-02-01|
+---------+-----------+------------+-----+--------+--------+----------+
```

Filtering and ordering rows

```
prices_in_belgium = prices.filter(col('countrycode') == 'BE').orderBy(col('date'))
```

```
+--------+-----------+------------+-----+--------+--------+----------+
|   store|countrycode|       brand|price|currency|quantity|      date|
+--------+-----------+------------+-----+--------+--------+----------+
|Kruidvat|         BE|     Nappy-k|  4.8|     EUR|      30|2019-01-28|
|    Aldi|         BE|Diapers-R-Us|  6.8|     EUR|      40|2019-02-03|
+--------+-----------+------------+-----+--------+--------+----------+
```

- Function `col` creates Column objects
- Method `orderBy` sorts values by a certain column.

Selecting and renaming columns

```
prices.select(



)
```

Selecting and renaming columns

```
prices.select(
    col("store"),
    col("brand")
)
```

Selecting and renaming columns

```
prices.select(
    col("store"),
    col("brand").alias("brandname")
)
```

```
+---------+------------+
|    store|   brandname|
+---------+------------+
|     Aldi|Diapers-R-Us|
|  Kruidvat|     Nappy-k|
|Carrefour|     Nappy-k|
|  Kruidvat|     Nappy-k|
|    Tesco|     Pampers|
|       DM|     Huggies|
|       DM|     Huggies|
+---------+------------+
```

Reducing duplicate values

```
prices.select(
    col("store"),
    col("brand").alias("brandname")
).distinct()
```

```
+---------+------------+
|    store|   brandname|
+---------+------------+
|       DM|     Huggies|
| Kruidvat|     Nappy-k|
|Carrefour|     Nappy-k|
|     Aldi|Diapers-R-Us|
|    Tesco|     Pampers|
+---------+------------+
```

Grouping and aggregating with mean()

```
(prices
    .groupBy(col('brand'))
    .mean('price')
).show()
```

```
+-----------+------------------+
|      brand|        avg(price)|
+-----------+------------------+
|Diapers-R-Us| 6.800000190734863|
|    Pampers| 6.300000190734863|
|    Huggies|               7.0|
|    Nappy-k|5.3666666348775225|
+-----------+------------------+
```

Grouping and aggregating with agg()

```
(prices
    .groupBy(col('brand'))
    .agg(
        avg('price').alias('average_price'),
        count('brand').alias('number_of_items')
        )
).show()
```

```
+------------+------------------+---------------+
|       brand|     average_price|number_of_items|
+------------+------------------+---------------+
|Diapers-R-Us| 6.800000190734863|              1|
|     Pampers| 6.300000190734863|              1|
|     Huggies|               7.0|              2|
|     Nappy-k|5.3666666348775225|              3|
+------------+------------------+---------------+
```

Joining related data

```
+---------+-----------+------------+-------+-----+--------+--------+----------+
|    store|countrycode|       brand|  model|price|currency|quantity|      date|
+---------+-----------+------------+-------+-----+--------+--------+----------+
|     Aldi|         BE|Diapers-R-Us|6months|  6.8|     EUR|      40|2019-02-03|
| Kruidvat|         BE|     Nappy-k|2months|  4.8|     EUR|      30|2019-01-28|
|Carrefour|         FR|     Nappy-k|2months|  5.7|     EUR|      30|2019-02-06|
|    Tesco|        IRL|     Pampers|3months|  6.3|     EUR|      35|2019-02-07|
|       DM|         DE|     Huggies|newborn|  6.8|     EUR|      40|2019-02-01|
+---------+-----------+------------+-------+-----+--------+--------+----------+
+------------+-------+---------------+-------+
|       brand|  model|absorption_rate|comfort|
+------------+-------+---------------+-------+
|Diapers-R-Us|6months|              2|      3|
|     Nappy-k|2months|              3|      4|
|     Pampers|3months|              4|      4|
|     Huggies|newborn|              3|      5|
+------------+-------+---------------+-------+
```

Executing a join with 2 foreign keys

```
ratings_with_prices = ratings.join(prices, ["brand", "model"])
```

```
+-----------+-------+---------------+-------+---------+-----------+-----+--------+--------+--------
|      brand|  model|absorption_rate|comfort|    store|countrycode|price|currency|quantity|      da
+-----------+-------+---------------+-------+---------+-----------+-----+--------+--------+--------
|Diapers-R-Us|6months|              2|      3|     Aldi|         BE|  6.8|     EUR|      40|2019-02-
|     Nappy-k|2months|              3|      4| Kruidvat|         BE|  4.8|     EUR|      30|2019-01-
|     Nappy-k|2months|              3|      4|Carrefour|         FR|  5.7|     EUR|      30|2019-02-
|     Pampers|3months|              4|      4|    Tesco|        IRL|  6.3|     EUR|      35|2019-02-
|     Huggies|newborn|              3|      5|       DM|         DE|  6.8|     EUR|      40|2019-02-
+-----------+-------+---------------+-------+---------+-----------+-----+--------+--------+--------
```

Let's practice!

BUILDING DATA ENGINEERING PIPELINES IN PYTHON

Packaging your application

BUILDING DATA ENGINEERING PIPELINES IN PYTHON

Oliver Willekens
Data Engineer at Data Minded

Running your pipeline locally

Running a Python program:

```
python hello_world.py  # script does something
```

Running a PySpark program *locally* is no different:

```
python my_pyspark_data_pipeline.py  # script starts at least a SparkSession
```

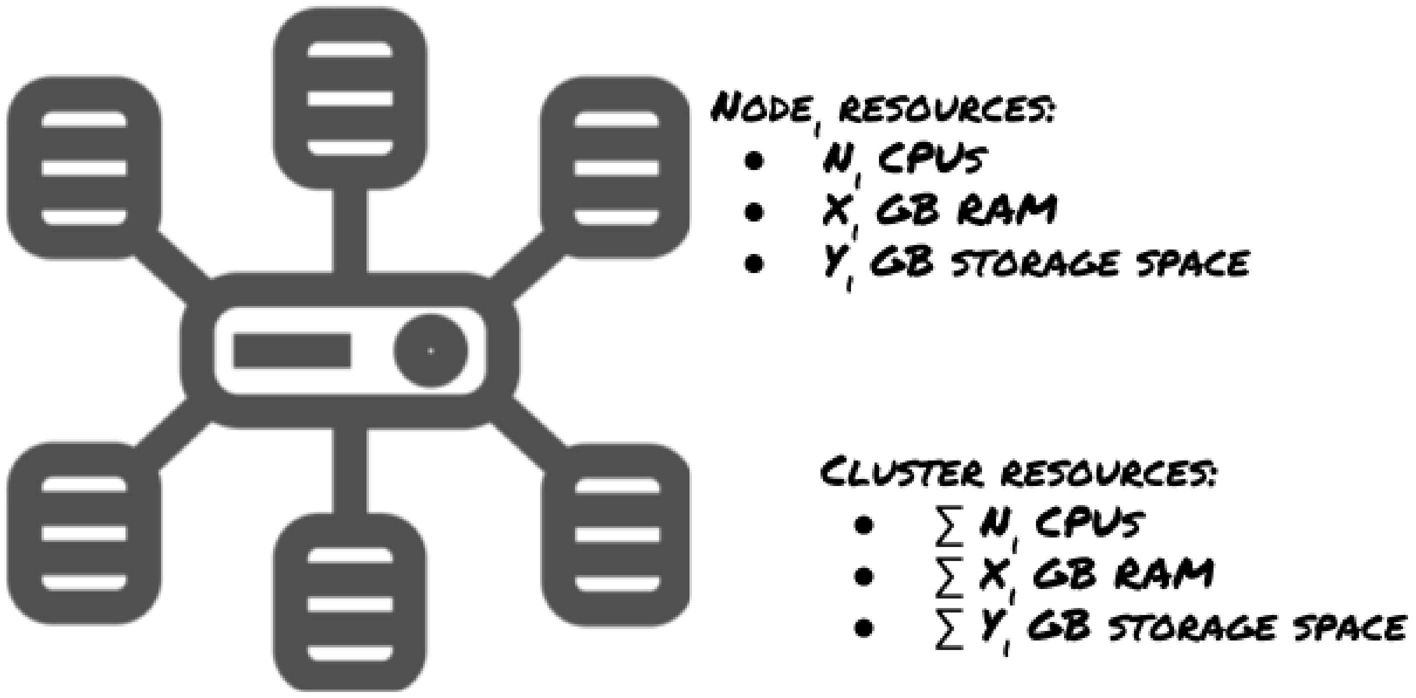
Conditions:

- local installation of Spark
- access to referenced resources
- classpath is properly configured

Using the "spark-submit" helper program

`spark-submit` comes with any Spark installation

1. sets up launch environment for use with the *cluster manager* and the selected *deploy mode*
2. invokes main class/app/module/function



CLUSTER MANAGER:
"THESE ARE THE AVAILABLE RESOURCES.
WHO NEEDS SOMETHING?"

Basic arguments of “spark-submit”

```
spark-submit \
  --master "local[*]" \
  --py-files PY_FILES \
  MAIN_PYTHON_FILE \
  app_arguments
```

```
On your path, if Spark is installed
URL of the cluster manager
Comma-separated list of zip, egg or py
Path to the module to be run
Optional arguments parsed by main script
```

Collecting all dependencies in one archive

```
.
├── pydiaper
│   ├── cleaning
│   │   ├── clean_prices.py
│   │   ├── clean_ratings.py
│   │   └── __init__.py
│   ├── data_catalog
│   │   ├── catalog.py
```

```
zip \
  --recurse-paths \
  dependencies.zip \
  pydiaper
```

```
spark-submit \
  --py-files dependencies.zip \
  pydiaper/cleaning/clean_prices.py
```

Let's practice!

BUILDING DATA ENGINEERING PIPELINES IN PYTHON