SPRAWOZDANIE

Zajęcia: Eksploracja i wizualizacja danych

Prowadzący: prof. dr hab. inż. Vasyl Martsenyuk

Laboratorum nr 3 Data: 24.11.2021 Temat: Użycie biblioteki „PySpark” dla dużych zbiorów danych Wariant: 1	Piotr Rybka Informatyka II stopień, niestacjonarne, III semestr, gr. 1

Repozytorium z kodem programu:

`https://colab.research.google.com/drive/102PjrGMDmssVFKQKxz3Sehfyq7MA4nKL?usp=sharing`

Polecenie

Na podstawie danych ze zbioru [1] wypróbować podstawowe metody biblioteki „pyspark”.

Zadanie 1. Zainstalować pakiet „pyspark”

Instalacja biblioteki w notatniku Google Colab (`https://research.google.com/colaboratory/`):

```
!pip install pyspark==3.0.1 py4j==0.10.9
```

Zadanie 2. Utworzyć sesję „SparkSession”

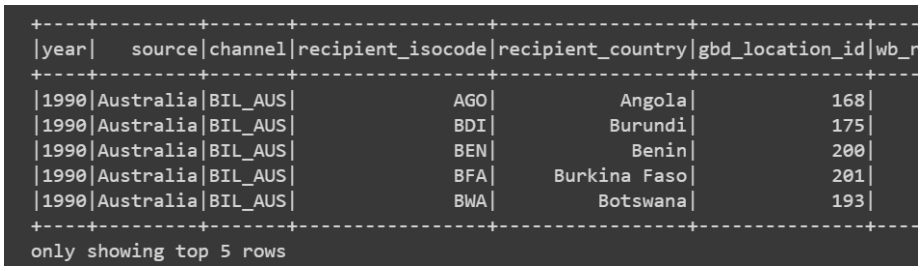
```
from pyspark.sql import SparkSession

spark = SparkSession\
        .builder\
        .master('local[4]')\
        .appName('zadanie_4')\
        .getOrCreate()
```

Zadanie 3. Wczytać dane z pliku CSV i wyświetlić pierwsze 5 wierszy

```
csv_file = r"/content/IHME_DAH_DATABASE_1990_2020_Y2021M09D22.CSV"

data = spark.read.csv(csv_file, header=True)

# sposob 1
data.show(5)

# sposob 2
data.head(5)
```

Wynik działania obu funkcji: 11 i 2.

Rysunek 1: Wynik działania funkcji `show()` (dane na podstawie [1])



```
+----+---------+-------+----------------+----------------+--------------+----
|year|   source|channel|recipient_isocode|recipient_country|gbd_location_id|wb_r
+----+---------+-------+----------------+----------------+--------------+----
|1990|Australia|BIL_AUS|             AGO|          Angola|           168|
|1990|Australia|BIL_AUS|             BDI|         Burundi|           175|
|1990|Australia|BIL_AUS|             BEN|           Benin|           200|
|1990|Australia|BIL_AUS|             BFA|    Burkina Faso|           201|
|1990|Australia|BIL_AUS|             BWA|        Botswana|           193|
+----+---------+-------+----------------+----------------+--------------+----
only showing top 5 rows
```

Rysunek 2: Wynik działania funkcji `head()` (dane na podstawie [1])

```
[Row(year=1990, source='Australia', channel='BIL_AUS', recipient_
 Row(year=1990, source='Australia', channel='BIL_AUS', recipient_
 Row(year=1990, source='Australia', channel='BIL_AUS', recipient_
 Row(year=1990, source='Australia', channel='BIL_AUS', recipient_
 Row(year=1990, source='Australia', channel='BIL_AUS', recipient_
```

Zadanie 4. Utworzyć schemat danych i zastosować go na zbiorze danych

```
# wyswietlenie pierwotnego schematu danych
data.printSchema()

from pyspark.sql.types import *

# utworzenie nowego schematu danych
data_schema = [
      StructField('year', IntegerType(), True),
      StructField('source', StringType(), True),
      StructField('channel', StringType(), True),
      StructField('recipient_isocode', StringType(), True),
      StructField('recipient_country', StringType(), True),
      StructField('gbd_location_id', IntegerType(), True),
      StructField('wb_regioncode', StringType(), True),
      StructField('wb_location_id', IntegerType(), True),
      StructField('gbd_region', StringType(), True),
      StructField('gbd_region_id', IntegerType(), True),
      StructField('gbd_superregion', StringType(), True),
      StructField('gbd_superregion_id', IntegerType(), True),
      StructField('elim_ch', IntegerType(), True),
      StructField('prelim_est', IntegerType(), True),
      StructField('dah_20', IntegerType(), True),
      StructField('rmh_fp_dah_20', IntegerType(), True),
      StructField('rmh_mh_dah_20', IntegerType(), True),
      StructField('rmh_hss_other_dah_20', IntegerType(), True),
      StructField('rmh_hss_hrh_dah_20', IntegerType(), True),
      StructField('rmh_other_dah_20', IntegerType(), True),
      StructField('nch_cnn_dah_20', IntegerType(), True),
      StructField('nch_cnv_dah_20', IntegerType(), True),
      StructField('nch_other_dah_20', IntegerType(), True),
      StructField('nch_hss_other_dah_20', IntegerType(), True),
      StructField('nch_hss_hrh_dah_20', IntegerType(), True),
      StructField('hiv_treat_dah_20', IntegerType(), True),
      StructField('hiv_prev_dah_20', IntegerType(), True),
      StructField('hiv_pmtct_dah_20', IntegerType(), True),
      StructField('hiv_other_dah_20', IntegerType(), True),
      StructField('hiv_ct_dah_20', IntegerType(), True),
      StructField('hiv_ovc_dah_20', IntegerType(), True),
      StructField('hiv_care_dah_20', IntegerType(), True),
      StructField('hiv_hss_other_dah_20', IntegerType(), True),
      StructField('hiv_hss_hrh_dah_20', IntegerType(), True),
      StructField('hiv_amr_dah_20', IntegerType(), True),
      StructField('mal_diag_dah_20', IntegerType(), True),
```

```
      StructField('mal_hss_other_dah_20', IntegerType(), True),
      StructField('mal_hss_hrh_dah_20', IntegerType(), True),
      StructField('mal_con_nets_dah_20', IntegerType(), True),
      StructField('mal_con_irs_dah_20', IntegerType(), True),
      StructField('mal_con_oth_dah_20', IntegerType(), True),
      StructField('mal_treat_dah_20', IntegerType(), True),
      StructField('mal_comm_con_dah_20', IntegerType(), True),
      StructField('mal_other_dah_20', IntegerType(), True),
      StructField('mal_amr_dah_20', IntegerType(), True),
      StructField('tb_other_dah_20', IntegerType(), True),
      StructField('tb_treat_dah_20', IntegerType(), True),
      StructField('tb_diag_dah_20', IntegerType(), True),
      StructField('tb_hss_other_dah_20', IntegerType(), True),
      StructField('tb_hss_hrh_dah_20', IntegerType(), True),
      StructField('tb_amr_dah_20', IntegerType(), True),
      StructField('oid_hss_other_dah_20', IntegerType(), True),
      StructField('oid_hss_hrh_dah_20', IntegerType(), True),
      StructField('oid_ebz_dah_20', IntegerType(), True),
      StructField('oid_zika_dah_20', IntegerType(), True),
      StructField('oid_covid_dah_20', IntegerType(), True),
      StructField('oid_other_dah_20', IntegerType(), True),
      StructField('oid_amr_dah_20', IntegerType(), True),
      StructField('ncd_hss_other_dah_20', IntegerType(), True),
      StructField('ncd_hss_hrh_dah_20', IntegerType(), True),
      StructField('ncd_tobac_dah_20', IntegerType(), True),
      StructField('ncd_mental_dah_20', IntegerType(), True),
      StructField('ncd_other_dah_20', IntegerType(), True),
      StructField('swap_hss_other_dah_20', IntegerType(), True),
      StructField('swap_hss_hrh_dah_20', IntegerType(), True),
      StructField('swap_hss_pp_dah_20', IntegerType(), True),
      StructField('other_dah_20', IntegerType(), True),
      StructField('rmh_dah_20', IntegerType(), True),
      StructField('nch_dah_20', IntegerType(), True),
      StructField('ncd_dah_20', IntegerType(), True),
      StructField('hiv_dah_20', IntegerType(), True),
      StructField('mal_dah_20', IntegerType(), True),
      StructField('tb_dah_20', IntegerType(), True),
      StructField('swap_hss_total_dah_20', IntegerType(), True),
      StructField('oid_dah_20', IntegerType(), True),
      StructField('unalloc_dah_20', IntegerType(), True),
]

data_struc = StructType(fields = data_schema)

# dodanie schematu do danych
data2 = spark.read.csv(csv_file, header=True, schema=data_struc)

data2.printSchema()
```

Pierwotny schemat danych tuż po załadowaniu danych z plik CSV – rys. 3a i po dodaniu

schematu danych – rys. 3b.

```
|-- rmh_other_dah_20: string (nullable = true)
|-- nch_cnn_dah_20: string (nullable = true)
|-- nch_cnv_dah_20: string (nullable = true)
|-- nch_other_dah_20: string (nullable = true)
|-- nch_hss_other_dah_20: string (nullable = true)
|-- nch_hss_hrh_dah_20: string (nullable = true)
|-- hiv_treat_dah_20: string (nullable = true)
|-- hiv_prev_dah_20: string (nullable = true)
|-- hiv_pmtct_dah_20: string (nullable = true)
|-- hiv_other_dah_20: string (nullable = true)
|-- hiv_ct_dah_20: string (nullable = true)
|-- hiv_ovc_dah_20: string (nullable = true)
|-- hiv_care_dah_20: string (nullable = true)
|-- hiv_hss_other_dah_20: string (nullable = true)
|-- hiv_hss_hrh_dah_20: string (nullable = true)
|-- hiv_amr_dah_20: string (nullable = true)
|-- mal_diag_dah_20: string (nullable = true)
|-- mal_hss_other_dah_20: string (nullable = true)
|-- mal_hss_hrh_dah_20: string (nullable = true)
|-- mal_con_nets_dah_20: string (nullable = true)
|-- mal_con_irs_dah_20: string (nullable = true)
|-- mal_con_oth_dah_20: string (nullable = true)
|-- mal_treat_dah_20: string (nullable = true)
|-- mal_comm_con_dah_20: string (nullable = true)
|-- mal_other_dah_20: string (nullable = true)
```

(a) Domyślny schemat danych

```
|-- rmh_other_dah_20: integer (nullable = true)
|-- nch_cnn_dah_20: integer (nullable = true)
|-- nch_cnv_dah_20: integer (nullable = true)
|-- nch_other_dah_20: integer (nullable = true)
|-- nch_hss_other_dah_20: integer (nullable = true)
|-- nch_hss_hrh_dah_20: integer (nullable = true)
|-- hiv_treat_dah_20: integer (nullable = true)
|-- hiv_prev_dah_20: integer (nullable = true)
|-- hiv_pmtct_dah_20: integer (nullable = true)
|-- hiv_other_dah_20: integer (nullable = true)
|-- hiv_ct_dah_20: integer (nullable = true)
|-- hiv_ovc_dah_20: integer (nullable = true)
|-- hiv_care_dah_20: integer (nullable = true)
|-- hiv_hss_other_dah_20: integer (nullable = true)
|-- hiv_hss_hrh_dah_20: integer (nullable = true)
|-- hiv_amr_dah_20: integer (nullable = true)
|-- mal_diag_dah_20: integer (nullable = true)
|-- mal_hss_other_dah_20: integer (nullable = true)
|-- mal_hss_hrh_dah_20: integer (nullable = true)
|-- mal_con_nets_dah_20: integer (nullable = true)
|-- mal_con_irs_dah_20: integer (nullable = true)
|-- mal_con_oth_dah_20: integer (nullable = true)
|-- mal_treat_dah_20: integer (nullable = true)
|-- mal_comm_con_dah_20: integer (nullable = true)
|-- mal_other_dah_20: integer (nullable = true)
```

(b) Nowy schemat danych

Rysunek 3: Dane przed dodaniem i po dodaniu schematu danych (dane na podstawie [1])

Zadanie 5. Wyświetlić typy danych w poszczególnych kolumnach

Funkcja:

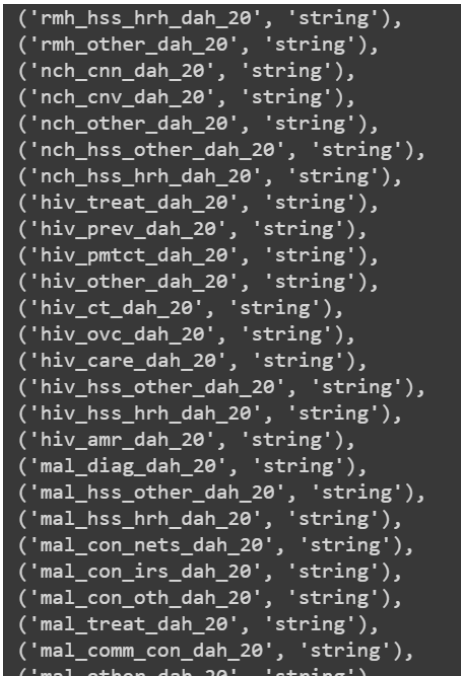
```
data.dtypes
```

Wynik działania funkcji – rys. 4.

Zadanie 6. Pokazać 2 pierwsze i ostatnie rekordy danych

```
data2.head(2)

data2.tail(2)
```

Rysunek 4: Typy danych (dane na podstawie [1])



```
('rmh_hss_hrh_dah_20', 'string'),
('rmh_other_dah_20', 'string'),
('nch_cnn_dah_20', 'string'),
('nch_cnv_dah_20', 'string'),
('nch_other_dah_20', 'string'),
('nch_hss_other_dah_20', 'string'),
('nch_hss_hrh_dah_20', 'string'),
('hiv_treat_dah_20', 'string'),
('hiv_prev_dah_20', 'string'),
('hiv_pmtct_dah_20', 'string'),
('hiv_other_dah_20', 'string'),
('hiv_ct_dah_20', 'string'),
('hiv_ovc_dah_20', 'string'),
('hiv_care_dah_20', 'string'),
('hiv_hss_other_dah_20', 'string'),
('hiv_hss_hrh_dah_20', 'string'),
('hiv_amr_dah_20', 'string'),
('mal_diag_dah_20', 'string'),
('mal_hss_other_dah_20', 'string'),
('mal_hss_hrh_dah_20', 'string'),
('mal_con_nets_dah_20', 'string'),
('mal_con_irs_dah_20', 'string'),
('mal_con_oth_dah_20', 'string'),
('mal_treat_dah_20', 'string'),
('mal_comm_con_dah_20', 'string'),
```

Zadanie 7. Dodać nową kolumnę zawierającą zera, zmienić jej nazwę, a następnie usunąć

```
# dodanie kolumny
res = data2.withColumn('Nowa kolumna', data2.year*0 + 1000)

# zmiana nazwy kolumny
res = res.withColumnRenamed('Nowa kolumna', 'col')

# usuniecie kolumny
res = data2.drop('col')
```

Zadanie 8. Dodać do danych kolumnę zawierającą numer wiersza

```python
from pyspark.sql.functions import udf

# funkcja zwracajaca kolejne liczby naturalne od 0
i = -1
def incr():
    global i
    i = i+1
    return i

# utworzenie nowej kolumny
newCol = udf(incr, IntegerType())

# dodanie nowej kolumny
data3 = data2.withColumn('id', newCol())

data3.show(5)
```

Uzyskany wynik – rys. 5

Rysunek 5: Dodatkowa kolumna z numerem wiersza (dane na podstawie [1])

```
---+--------------------+---------+-------------+---+
h_20|swap_hss_total_dah_20|oid_dah_20|unalloc_dah_20| id|
---+--------------------+---------+-------------+---+
  0|                   0|        0|         null|  0|
  0|                   0|        0|            0|  1|
  0|                   0|        0|            0|  2|
  0|                   0|        0|            0|  3|
  0|                   0|        0|         null|  4|
---+--------------------+---------+-------------+---+
```

Zadanie 9. Sprawdzić liczbę wierszy danych przed i po usunięciu wierszy bez danych dwoma sposobami

```
# poczatkowa liczba rekordow
data3.count()                    # wynik: 384 306

# usuniecie wierszy bez danych (sposob 1.)
data4 = data3.na.drop()

data4.count()                    # wynik: 136 560

# wstawienie zera w miejsce braku danych (sposob 2.)
data5 = data3.na.fill(data3.select(0 * data3.year).collect()[0][0])

data5.count()                    # wynik: 384 306
```

Zadanie 10. Wyświetlić wybrane kolumny danych

```
data5.select(['year', 'source', 'dah_20']).show(5)
```

Wyodrębnione kolumny – rys.6.

Rysunek 6: Pięć pierwszych wierszy wybranych kolumn danych (dane na podstawie [1])

```
+----+---------+------+
|year|   source|dah_20|
+----+---------+------+
|1990|Australia|    14|
|1990|Australia|    12|
|1990|Australia|    12|
|1990|Australia|    13|
|1990|Australia|    25|
+----+---------+------+
only showing top 5 rows
```

Zadanie 11. Odfiltrować dane zawierające dane od roku 2000

```
from pyspark.sql.functions import col

data5.filter((col('year') >= 2000) & (col('dah_20') > 20)).select(['year',
    'source', 'dah_20']).show(5)
```

Rezultat odfiltrowania danych – rys. 7

Rysunek 7: Wynik filtrowania danych (rok 2000 i później, wartość `dah_20` powyżej 20; dane na podstawie [1])

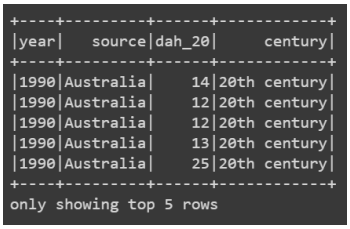
```
+----+---------+------+
|year|   source|dah_20|
+----+---------+------+
|2000|Australia|   333|
|2000|Australia|    25|
|2000|Australia|    46|
|2000|Australia|   106|
|2000|Australia|    22|
+----+---------+------+
only showing top 5 rows
```

Zadanie 12. Dodać kolumnę zawierającą wynik sprawdzenia warunku

```
# dodanie kolumny zawierajacej wynik sprawdzenia, czy rok jest wiekszy niz
    2000
from pyspark.sql import functions as f

data5.select('year', 'source', 'dah_20', f.when(data5.year > 2000, '21st
    century').otherwise('20th century').alias('century')).show(5)

# dodanie kolumny zawierajacej wynik sprawdzenia, czy nazwa kraju zaczyna
    sie od litery 'A'
data5.select('year', 'source', 'dah_20', data5.source.rlike('^A').alias('A-
    country')).show(5)
```

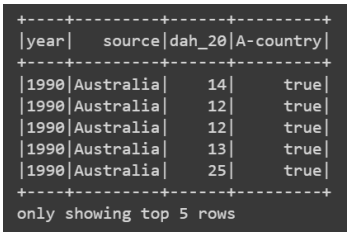
Uzyskane dane – rys. 8 i

Rysunek 8: Nowa kolumna `century` zawierająca dane dyskretne (dane na podstawie [1])



```
+----+---------+------+------------+
|year|   source|dah_20|     century|
+----+---------+------+------------+
|1990|Australia|    14|20th century|
|1990|Australia|    12|20th century|
|1990|Australia|    12|20th century|
|1990|Australia|    13|20th century|
|1990|Australia|    25|20th century|
+----+---------+------+------------+
only showing top 5 rows
```

Rysunek 9: Nowa kolumna `A-country` zawierająca dane typu logicznego (dane na podstawie [1])



```
+----+---------+------+---------+
|year|   source|dah_20|A-country|
+----+---------+------+---------+
|1990|Australia|    14|     true|
|1990|Australia|    12|     true|
|1990|Australia|    12|     true|
|1990|Australia|    13|     true|
|1990|Australia|    25|     true|
+----+---------+------+---------+
only showing top 5 rows
```

Zadanie 13. Pogrupować dane wg kraju i obliczyć liczybę danych, średnią wartość oraz wartości minimalne i maksymalne

```
# pogrupowanie danych wg kraju
from pyspark.sql.functions import mean, count, min, max

data5\
    .select(['year', 'source', 'dah_20'])\
    .groupBy('source')\
    .agg(
        count('year').alias('number of countries'),
        mean('dah_20').alias('meah dah_20'),
        min('year').alias('min year'),
        max('year').alias('max year'),
    ).show(5)
```

Uzyskany wynik – 10.

Rysunek 10: Przykład grupowania danych względem kolumny `source` (dane na podstawie [1])

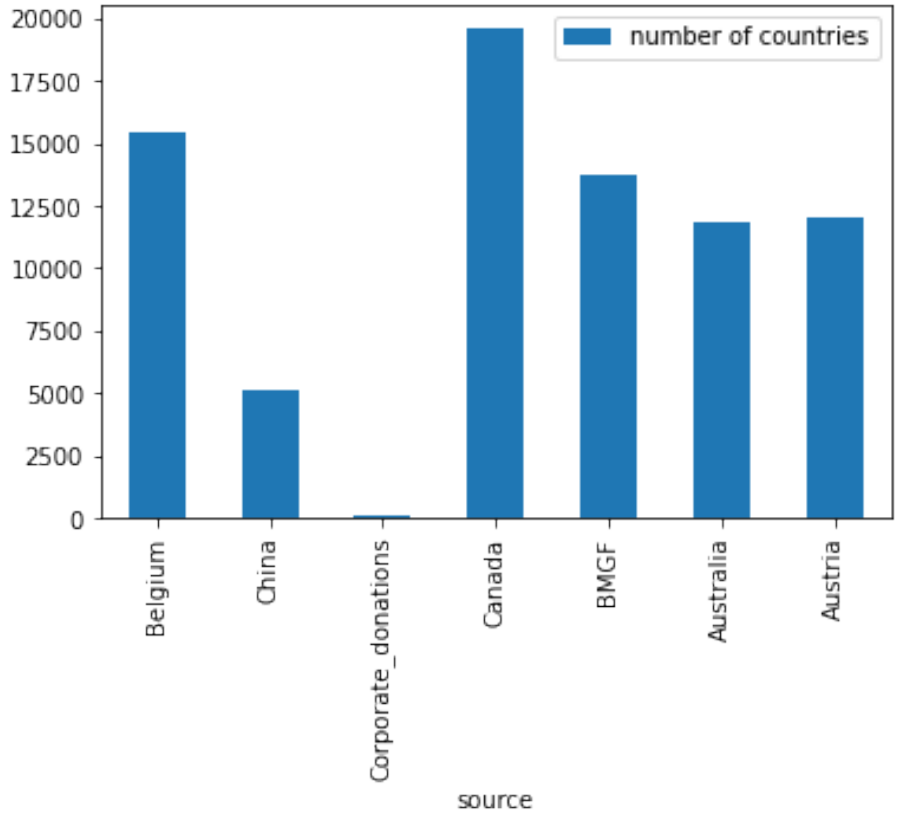
```
+--------------+------------------+-----------------+--------+--------+
|        source|number of countries|       meah dah_20|min year|max year|
+--------------+------------------+-----------------+--------+--------+
|        Sweden|             19691| 945.3851505764054|    1990|    2020|
|Debt_repayments|              4893| 8464.654199877376|    1990|    2020|
|       Germany|             17489|1851.7729429927383|    1990|    2020|
|        France|             17156|1562.2891116810445|    1990|    2020|
|        Greece|              8178| 80.97407679139154|    1990|    2020|
+--------------+------------------+-----------------+--------+--------+
only showing top 5 rows
```

Zadanie 14. Wygenerować wykres słupkowy na podstawie pogrupowanych danych

```
from matplotlib import pyplot as plt

res = data5\
    .filter(data5.source.rlike('^[ABC]'))\
    .select(['year', 'source', 'dah_20'])\
    .groupBy('source')\
    .agg(
        count('year').alias('number of countries'),
        mean('dah_20').alias('mean dah_20'),
        mean('year').alias('mean year'),
        max('year').alias('max year'))\
    .toPandas()

res\
    .plot(kind='bar', x='source', y='number of countries')

plt.show()
```

Uzyskany wykres – 11.

Rysunek 11: Wykres słupkowy liczby rekordów w wybranych krajach (dane na podstawie [1])



Wnioski

Biblioteka „pyspark” jest wygodną alternatywą dla pakietu „pandas”, zwłaszcza jeśli analizowane są duże zbiory danych.

Wszelkie operacje na danych wymagają utworzenia sesji „SparkSession” przy użyciu biblioteki `builder`, np.

```
from pyspark.sql import SparkSession

spark = SparkSession\
        .builder\
        .master('local[4]')\    # liczba rdzeni procesora
        .appName('zadanie')\    # nazwa sesji
        .getOrCreate()
```

Biblioteka „pyspark” pozwala wczytywać dane z różnych źródeł danych:

```
# pliki CSV
data = spark.read.csv(csv_file_path, sep=',', header=True)

# pliki JSON
data = spark.read.json(json_data_file)

# pliki Parquet
data = spark.read.parquet(parquet_data_file)
```

Zapis wymaga użycia jednej z 2 funkcji:

```
# pliki CSV
spark.write.csv(csv_file_path, sep=',', header=True)

# pliki JSON i Parquet
data = spark.write.save(json_data_file)
data = spark.write.save(parquet_data_file)
```

Wczytane dane są zwykle w formacie „string”, ale można nadarzucić im typ przy użyciu schematu, np.

```
from pyspark.sql.types import *

data_schema = [
                StructField('Id', IntegerType(), False),
```

```
                StructField('Miasto', StringType(), True),
                StructField('Populacja', DoubleType(), True)
    ]
data_struc = StructType(fields = data_schema)

data = spark.read.csv(csv_file, sep=',', header=True, schema=data_struc)

# wyswietlenie schematu
data.printSchema()

# wyswietlenie typow danych
data.dtypes
```

Wybieranie wierszy odbywa się analogicznie jak w bibliotece „pandas”:

```
data.head(1)        # pierwszy wiersz
data.show(1)        # jw.
data.tail(1)        # ostatni wiersz
```

Możliwe jest dodanie kolumny, zmiana jej nazwy oraz usunięcie, np.

```
# dodanie kolumny
data = data.withColumn('Nowa kolumna', 1000000 + data.Id)

# zmiana nazwy kolumny
data = data.withColumnRenamed('Nowa kolumna', 'Id2')

# usuniecie kolumny
data = data.drop('Id2')
```

Wartości w nowych kolumnach można też dodawać przy użyciu osobnej funkcji, np.

```
from pyspark.sql.functions import udf

i = -1
def incr():
  global i
  i = i+1
  return i

newCol = udf(incr, IntegerType())

csv_data = csv_data.withColumn('id', newCol())
```

Wiersze niezawierające danych mogą być łatwo usunięte na kilka sposobów:

- przez zwykłe usunięcie wierszy:
  `new_data = data.na.drop()`;
- przez wstawienie innych danych:
  `new_data = data.na.fill(data.select(data['kolumna']).collect()[0][0])`;

- przez zastąpienie:
  `new_data = data.na.replace('stara wartosc', 'nowa wartosc')`.

Liczbę wierszy zwraca funkcja `data.count()`.
Istnieje kilka sposobów odwołania się do kolumny:

- składnia kropkowa: `dane.kolumna` (nazwa kolumny nie może zawierać spacji);
- składnia indeksowa: `dane['kolumna']` (nazwa kolumny może być dowolna);
- funkcją `select`, podając jako jej parametr listę nazw kolumn, które mają być wyodrębnione.

Odfiltrowaniu danych służy funkcja `filter(c)`, zawierająca listę warunków połączonych spójnikami `&` lub `|`. Zakres danych można wskazać funkcją `dane.kolumna.between(a, b)`.

Wśród przydatnych funkcji w klasie `pyspark.sql.functions` można wymienić:

- `udf(f, t)` – pozwala zdefiniować kolumnę użytkownika i zapełnić ją danymi w typie `t` i zwracanymi przez funkcję `f`;
- `mean(c)` – średnia danych z kolumny `c`;
- `col('nazwa')` – kolumna, której wartości mają być porównywane;
- `when(c, a).otherwise(b)` – zwraca wartość `a`, jeśli spełniony jest warunek `c` (w przeciwnym razie zwracane jest `b`);
- `dane.kolumna.rlike(regex)` – zwraca `true`, jeśli wartość w wierszu kolumny pasuje do wyrażenia regularnego.

Nazwy kolumn można aliasować funkcją `dane.kolumna.alias('nowa nazwa')`.

Grupowanie odbywa się przez użycie funkcji `groupBy`, przyjmującej listę kolumn, względem któych dane mają być pogrupowane. Funkcja `agg` pozwala przekazać funkcje agregujące dane z przekazanych im kolumn.

```
dane.select(['kol1']).groupBy('kol1').agg(f1('kol2'), f2('kol3'))
```

Funkcją `toPandas()` można każdy zbiór danych przekształcić do ramki danych, którą z kolei można przetwarzać przy użyciu biblioteki `pandas` i innych.

Strony internetowe

[1] *Development Assistance for Health Database 1990-2020*. URL: `http://ghdx.healthdata.org/record/ihme-data/development-assistance-health-database-1990-2020` (term. wiz. 21. 10. 2021).