**BIG DATA PROJECT**

**Books Survey**

Technion - Israel Institute of Technology

Continuing and Professional Education

Course: Big Data Analyst

Handed by: Haim Zimbovsky, Slava Bakman, Gabriel Halimi

**Table of Contents**

[1 Overview 3](#_Toc85960788)

[2 Data Cleaning and Organization (PySpark) 4](#_Toc85960789)

[2.1 Data Source 4](#_Toc85960790)

[2.2 Environment preparation 5](#_Toc85960791)

[2.3 Reading and cleaning “Users” table 6](#_Toc85960792)

[2.3.1 Reading 6](#_Toc85960793)

[2.3.2 Cleaning 6](#_Toc85960794)

[2.4 Reading and cleaning “Books” table 8](#_Toc85960795)

[2.4.1 Reading 8](#_Toc85960796)

[2.4.2 Cleaning 9](#_Toc85960797)

[2.5 Reading and cleaning “Rating” table 9](#_Toc85960798)

[2.5.1 Reading 9](#_Toc85960799)

[2.5.2 Cleaning 10](#_Toc85960800)

[2.6 Create CSV File for Tableau 10](#_Toc85960801)

[2.6.1 Joining all data frames to one 10](#_Toc85960802)

[2.6.2 Creating CSV file 11](#_Toc85960803)

[3 Tableau 12](#_Toc85960804)

[3.1 Tableau Prep Builder 12](#_Toc85960805)

[3.2 Tableau Prep Desktop – Visualizations 13](#_Toc85960806)

[3.2.1 Books Average Rating in Each Country 13](#_Toc85960807)

[3.2.2 Author books by publisher Average Rating 13](#_Toc85960808)

# Overview

The purpose of the project is to analyze the dataset of book survey was collected by Cai-Nicolas Ziegler from the Book-Crossing community with kind permission from Ron Hombaker, CTO of Humanking System.

This analysis is based on the usage of two main tools: Python Spark Library (pyspark) and Tableau Commercial Software.

We use pyspark library for data cleaning and organization and Tableau application – for BI process.

# Data Cleaning and Organization (PySpark)

## Data Source

Our data source includes three files in CSV format:

|  |  |
| --- | --- |
| **File name** | **Columns** |
| BX-Book-Ratings.csv | User-ID  ISBN  Book-Rating |
| BX-Books.csv | ISBN  Book-Title  Book-Author  Year-Of-Publication  Publisher  Image-URL-S  Image-URL-M  Image-URL-L |
| BX-Users.csv | User-ID  Location  Age |

The data cleaning and organization processes include the following steps:

* Environment preparation: Virtual Machine installation in Google Colab application, install necessary software and python libraries and mounting google drive disk on this VM,
* Users table analytics,
* Books table analytics,
* Rating table analytics,
* CSV file creating for Tableau,

## Environment preparation

The first step is included operating system update and Spark application installation.

|  |
| --- |
| !apt update  !apt-get install openjdk-8-jdk-headless -qq > /dev/null  qq > /dev/null  !wget -q https://downloads.apache.org/spark/spark-3.1.2/spark-3.1.2-bin-hadoop3.2.tgz  !tar xf spark-3.1.2-bin-hadoop3.2.tgz  # !tar --> like unzip  !pip install -q findspark |

Where:

* **apt update -** update operating system to last release.
* **apt-get install openjdk-8-jdk-headless -qq > /dev/null** - Java Development Kit installation
* **wget -q**[**https://downloads.apache.org/spark/spark-3.1.2/spark-3.1.2-bin-hadoop3.2.tgz**](https://downloads.apache.org/spark/spark-3.1.2/spark-3.1.2-bin-hadoop3.2.tgz)- download Spark version 3.1.2
* **tar xf spark-3.1.2-bin-hadoop3.2.tgz** and **pip install -q findspark** – Spark installation

After this we need mount google drive on virtual machine, set environments and import all necessary Python packages.

Set environments:

# Set enviroments

import os

os.environ["JAVA\_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"

os.environ["SPARK\_HOME"] = "/content/spark-3.1.2-bin-hadoop3.2"

Mount google drive:

from google.colab import drive

drive.mount('/content/drive')

Import Python packages:

import findspark

findspark.init("spark-3.1.2-bin-hadoop3.2") # SPARK\_HOME

from pyspark.sql import SparkSession

spark = SparkSession.builder.master("local[\*]").getOrCreate()

from pyspark.sql import Row

from pyspark.sql import functions as F

from pyspark.sql.types import IntegerType, DecimalType, StringType

import shutil

Set data path and data files:

data\_path = "/content/drive/MyDrive/BDA/Big\_Data\_Project/Data"

file\_bx\_users = data\_path + "/" + "BX-Users.csv"

file\_bx\_books = data\_path + "/" + "BX-Books.csv"

file\_bx\_rating = data\_path + "/" + "BX-Book-Ratings.csv"

## Reading and cleaning “Users” table

### Reading

Spark SQL provides spark.read.csv("path") to read a CSV file into Spark DataFrame. We used this function for creating a DataFrame of “Users” data.

df\_bx\_users = spark.read.csv(file\_bx\_users, sep=";", header=True, inferSchema=True, encoding="ISO-8859-1")

Where:

* **df\_bx\_users** – data frame,
* **file\_bx\_users** – path to CSV file,
* **sep=”**;” – separator “;”,
* **header=True** – read the first row as headers,
* **inferSchema=True** – infer the schema of each column automatically,
* **encoding="ISO-8859-1** – set encoding to “ISO-8859-1”

### Cleaning

Delete dash symbol from column name

# Rename Column

df\_bx\_users = df\_bx\_users.withColumnRenamed('User-ID', 'UserID')

df\_bx\_users.printSchema()

Delete rows where Age equal to null

df\_bx\_users.registerTempTable("bx\_users")

spark.sql('select \* from bx\_users where Age is null').show()

df\_bx\_users = spark.sql('select \* from bx\_users where Age is not null')

We found 366 users with ages more than 100 years old. For example

8782|calgary, alberta,...|239

We suppose that the data was typed incorrectly and the original age should be divided to 10.

df\_bx\_users = df\_bx\_users.withColumn("Age", F.when(df\_bx\_users.Age > 100, df\_bx\_users.Age/10).otherwise(df\_bx\_users.Age))

df\_bx\_users

df\_bx\_users.filter("Age>100").show()

Replace “NULL” string to average user’s age in “Age” column

spark.sql('select \* from bx\_users where Age like "%NULL%"').count()

df\_bx\_users = df\_bx\_users.withColumn("Age", F.when(df\_bx\_users.Age == "NULL", F.lit(0)).otherwise(df\_bx\_users.Age))

df\_bx\_users = df\_bx\_users.withColumn("Age", df\_bx\_users["Age"].cast(IntegerType()))

df\_bx\_users.printSchema()

age\_avg = int(df\_bx\_users.agg({"Age":"avg"}).collect()[0][0])

print("Average age is " + str(age\_avg))

df\_bx\_users = df\_bx\_users.withColumn("Age", F.when(df\_bx\_users.Age == 0, age\_avg).otherwise(df\_bx\_users.Age))

df\_bx\_users.show(3)

The user’s average age is 20.

Convert “Age” column datatype to decimal.

df\_bx\_users = df\_bx\_users.withColumn('Age', df\_bx\_users.Age.cast(DecimalType()))

df\_bx\_users.filter("UserID==8782").show()

df\_bx\_users.printSchema()

The location column includes city, area, and country are separated by a comma. We will take from the string city and country. The “Location” column will be dropped. We will create two new columns – “UserCity” and “UserCountry”

split\_location = F.split(df\_bx\_users.Location, ',')

df\_bx\_users = df\_bx\_users.withColumn('UserCity', split\_location.getItem(0))

df\_bx\_users = df\_bx\_users.withColumn('UserCountry', split\_location.getItem(2))

df\_bx\_users = df\_bx\_users.drop('Location')

Clean “UserCity” and “UserCountry” columns

df\_bx\_users = df\_bx\_users.withColumn('UserCity', F.regexp\_replace('UserCity', '\d+', ''))

df\_bx\_users = df\_bx\_users.withColumn('UserCity', F.regexp\_replace('UserCity', '&', ''))

df\_bx\_users = df\_bx\_users.withColumn('UserCity', F.regexp\_replace('UserCity', '#', ''))

df\_bx\_users = df\_bx\_users.withColumn('UserCity', F.regexp\_replace('UserCity', ';', ''))

df\_bx\_users = df\_bx\_users.withColumn('UserCity', F.regexp\_replace('UserCity', ',', ''))

df\_bx\_users = df\_bx\_users.withColumn('UserCity', F.regexp\_replace('UserCity', '(currently living in england)', ''))

df\_bx\_users = df\_bx\_users.withColumn('UserCity',

    F.when((df\_bx\_users.UserCity == 'nyc') | (df\_bx\_users.UserCity == 'ny'),

            'new york').otherwise(df\_bx\_users.UserCity))

df\_bx\_users = df\_bx\_users.withColumn('UserCountry', F.regexp\_replace('UserCountry', '&', ''))

df\_bx\_users = df\_bx\_users.withColumn('UserCountry', F.regexp\_replace('UserCountry', '#', ''))

df\_bx\_users = df\_bx\_users.withColumn('UserCountry', F.regexp\_replace('UserCountry', ';', ''))

df\_bx\_users = df\_bx\_users.withColumn('UserCountry', F.regexp\_replace('UserCountry', ',', ''))

Set “n/a” to other in “UserCity” and “UserCountry” columns

df\_bx\_users.filter(df\_bx\_users.UserCity == 'n/a').show()

df\_bx\_users.filter(df\_bx\_users.UserCountry == 'n/a').show()

df\_bx\_users = df\_bx\_users.withColumn("UserCity",

  F.when((df\_bx\_users.UserCity == 'n/a') |

         (df\_bx\_users.UserCity == ''), 'other').otherwise(df\_bx\_users.UserCity))

df\_bx\_users = df\_bx\_users.withColumn("UserCountry",

  F.when((df\_bx\_users.UserCountry == 'n/a') |

         (df\_bx\_users.UserCountry == ''), 'other').otherwise(df\_bx\_users.UserCountry))

## Reading and cleaning “Books” table

### Reading

We used “spark.read.csv” function to read data from CSV file and create a due data frame.

df\_bx\_books = spark.read.csv(file\_bx\_books, sep=";", header=True, inferSchema=True, encoding="ISO-8859-1")

Where:

* **df\_bx\_books** – data frame,
* **file\_bx\_books** – path to CSV file,
* **sep=”**;” – separator “;”,
* **header=True** – read the first row as headers,
* **inferSchema=True** – infer the schema of each column automatically,
* **encoding="ISO-8859-1** – set encoding to “ISO-8859-1”

### Cleaning

Delete dash symbol from column name and delete unnecessary columns.

df\_bx\_books = df\_bx\_books.withColumnRenamed('Book-Title', 'BookTitle')

df\_bx\_books = df\_bx\_books.withColumnRenamed('Book-Author', 'BookAuthor')

df\_bx\_books = df\_bx\_books.withColumnRenamed('Year-Of-Publication', 'YearOfPublication')

df\_bx\_books = df\_bx\_books.drop(df\_bx\_books['Image-URL-S'])

df\_bx\_books = df\_bx\_books.drop(df\_bx\_books['Image-URL-M'])

df\_bx\_books = df\_bx\_books.drop(df\_bx\_books['Image-URL-L'])

## Reading and cleaning “Rating” table

### Reading

After CSV file investigation we decided to read the data and at the same time perform the cleaning process. On the one hand, it takes more time than reading the data by “spark.read.csv” function but on the second hand the cleaning process is easier and flexibility and we did almost all cleaning steps.

Function “parsingInput” allows to us clean to each line and create from it one data list.

# Get Columns Function

def parsingInput(line):

    data = []

    fields = line.split(";")

    for field in fields:

      field = field.replace('"', '')

      field = field.replace(',', '')

      field = field.replace('\n', '')

      data.append(field)

    return data

columns = []

data = []

with open(file\_bx\_rating, mode='r', encoding="ISO-8859-1") as file:

  columns = parsingInput(file.readline())

  other\_lines = file.readlines()

file.close()

for line in other\_lines:

  data.append(parsingInput(line))

The first row of the CSV file will be parsed as column names and all other lines will be added to the data frame as data.

After this, we create spark data frame by using a data list and columns list.

df\_bx\_rating = spark.createDataFrame(data, columns)

Where:

* **df\_bx\_rating** – data frame,
* **data** – data list,
* **columns** – columns list

### Cleaning

Delete dash symbol from column name and change the datatype of “BookRating” column to integer.

df\_bx\_rating = df\_bx\_rating.withColumnRenamed('User-ID', 'UserID')

df\_bx\_rating = df\_bx\_rating.withColumnRenamed('Book-Rating', 'BookRating')

df\_bx\_rating = df\_bx\_rating.withColumn('BookRating', df\_bx\_rating.BookRating.cast(IntegerType()))

df\_bx\_rating.printSchema()

df\_bx\_rating.show(5)

## Create CSV File for Tableau

### Joining all data frames to one

We found that the Tableau application knows to read data from the CSV file. Therefore, we will create one CSV file from all tables.

First able, we need to create one data frame from our three data frames: df\_bx\_users, df\_bx\_books, and df\_bx\_rating. For this, we will use the “join” function of the spark library.

Let’s make a join between df\_bx\_books and df\_bx\_rating data frames.

df\_books\_rating = df\_bx\_books.join(df\_bx\_rating, df\_bx\_books.ISBN == df\_bx\_rating.ISBN, "inner")

df\_books\_rating = df\_books\_rating.drop(df\_bx\_rating.ISBN)

df\_books\_rating.show(5)

Pay attention, after joining, we need to drop df\_bx\_rating.ISBN column from new data frame df\_books\_rating, otherwise the ISBN column will be presented twice.

Let’s, make a join between df\_books\_rating and df\_bx\_users data frames.

df\_result = df\_books\_rating.join(df\_bx\_users, df\_books\_rating.UserID == df\_bx\_users.UserID, "inner")

df\_result = df\_result.drop(df\_bx\_users.UserID)

df\_result.show(5)

Check again, that the received data frame does not include “NULL” and “Not Available” data.

df\_result.select([F.count(F.when(F.isnan(c) | F.col(c).isNull(), c)).alias(c) for c in df\_result.columns]).show()

### Creating CSV file

We use the “write” of the spark library for creating CSV files. Before CSV file creating the destination directory will be erased if exists.

result\_directory = data\_path + "/../result"

if os.path.exists(result\_directory) and os.path.isdir(result\_directory):

    shutil.rmtree(result\_directory)

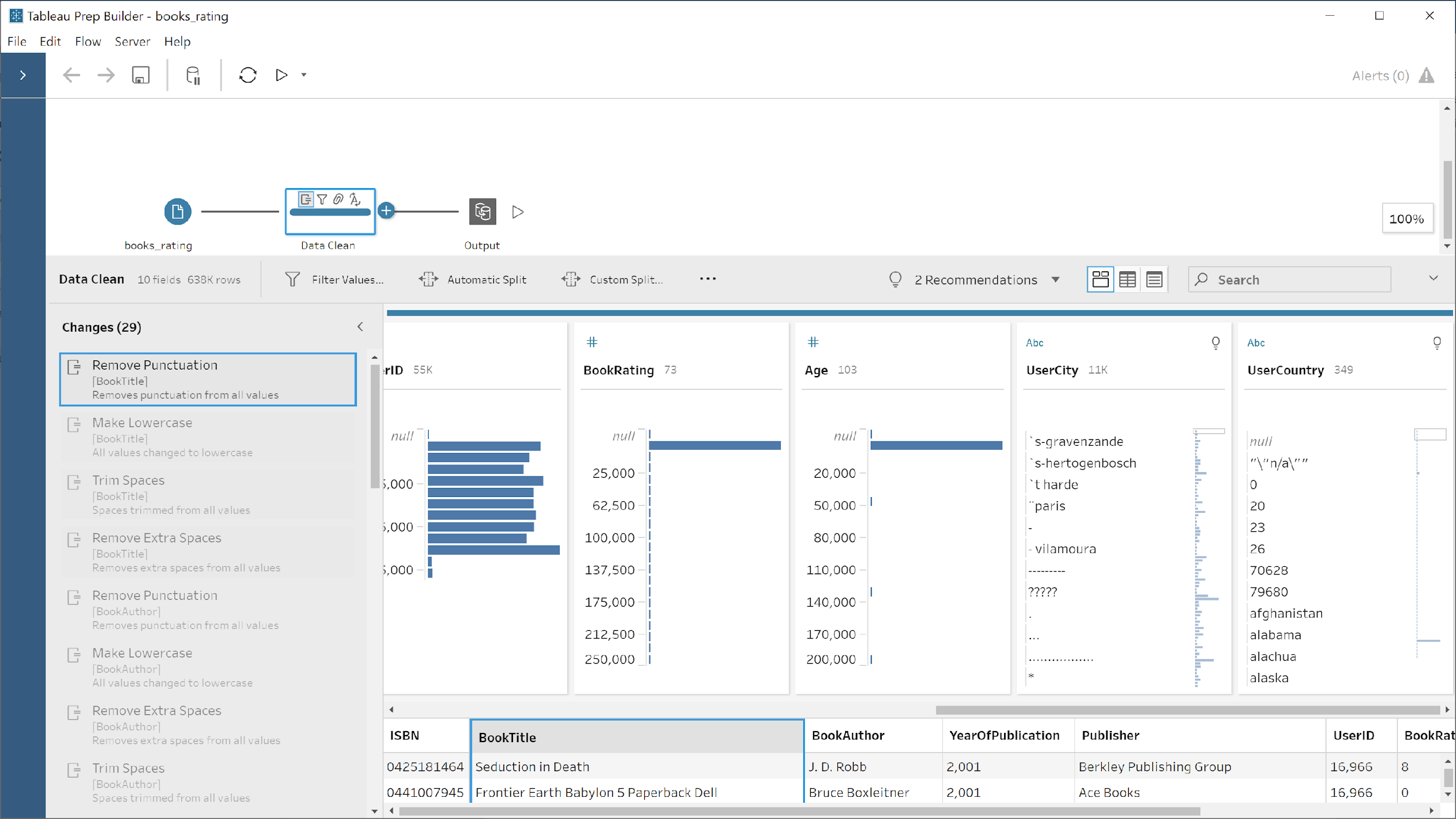
df\_result.repartition(1).write.format("com.databricks.spark.csv").option("header", "true").option("encoding", "ISO-8859-1").save(result\_directory)

Where:

* **df\_result** – data frame,
* **result\_directory** – path to destination directory,
* **columns** – columns list

# Tableau

## Tableau Prep Builder



Cleaning steps (all the following steps same for all columns):

1. Remove punctuation
2. Make lowercase
3. Trim spaces
4. Remove extra spaces
5. Change datatype (if necessary)

## Tableau Prep Desktop – Visualizations

### Books Average Rating in Each Country

The visualization displays the rating of the books title in each country that is in the data base. For example, the highest rated book title in Albania is “Life of Pi” with average rating of 9.00. The Author of that book is Yann Martel.

Chart

Description automatically generated with medium confidence

### Author books by publisher Average Rating

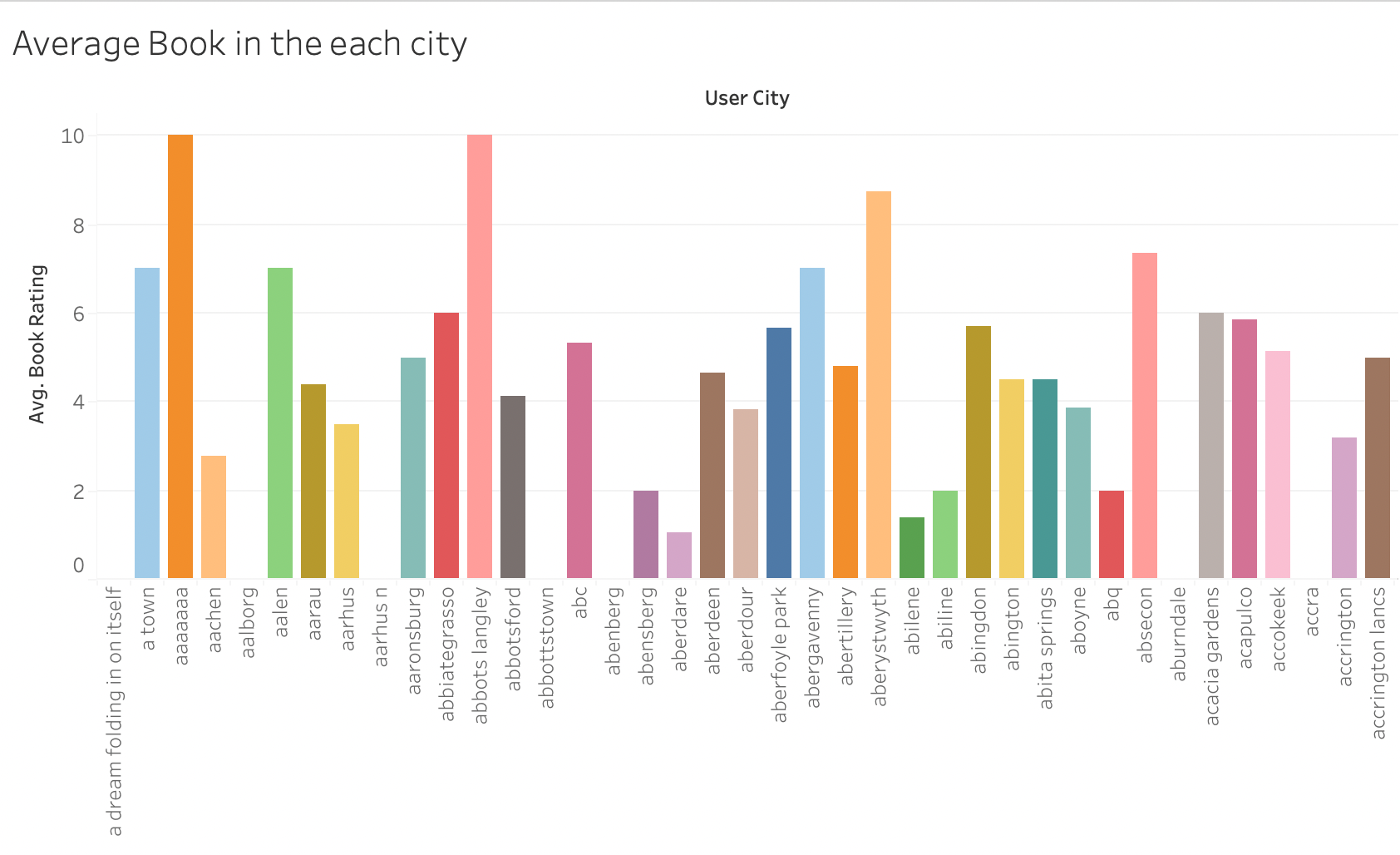
This visualization shows the average rating of each author and the distinct amount of book he published. As it can see, there is two abnormal observations, the two authors with the highest number of published books are with very low average books rating. The author Francine Pascal published by the publisher “Sweet Valley” 264 different books and has very low average rating of 1.21. The author Ann M.Martin published by the publisher “Schalatics” 269 different books and has very low average rating of 1.00.

Chart, scatter chart

Description automatically generated

3.2.3 **Average Book in the each city**

Here, the visualization displays the average book rationg given according to the cities to the different books. This allows us to see in which cities readers are more demanding than in other cities. We can see it very well with the city of Abendare where the average rating is very low 1.08.



3.2.4  **Average age depending on the country**

In this visualization, we can see the average age of readers depending on the country. This allows us in particular to see if the readers are rather young or old depending on the country. This will influence the type of book read, or the date of publication. Here, we see when Albania, the average age is young, 24 years old.

