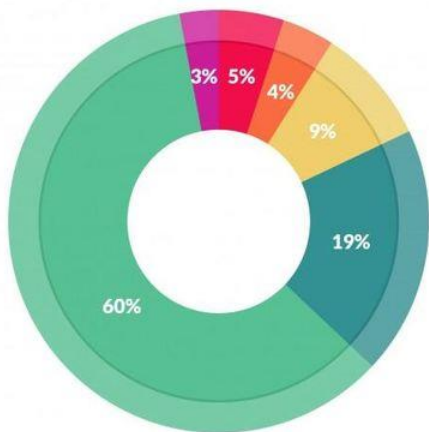


Chapter 2: Data Cleansing

ทำความสะอาดข้อมูลให้พร้อมใช้งานด้วยเครื่องมือ Distributed Processing: Apache Spark

What is Data Cleansing & Data Quality

Data Scientist ใช้เวลาส่วนใหญ่ไปกับการทำความสะอาดข้อมูล



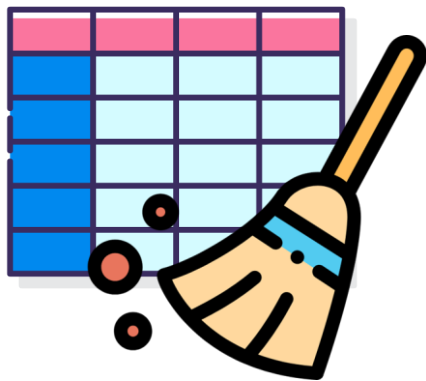
What data scientists spend the most time doing

- Building training sets: 3%
- Cleaning and organizing data: 60%
- Collecting data sets: 19%
- Mining data for patterns: 9%
- Refining algorithms: 4%
- Other: 5%

ผลสำรวจจาก Forbes บอกว่า Data Scientist ใช้เวลากว่า 60% เพื่อเตรียมและทำความสะอาดข้อมูล

<https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/>

Data Cleansing คืออะไร



Data Cleansing การทำความสะอาดข้อมูล

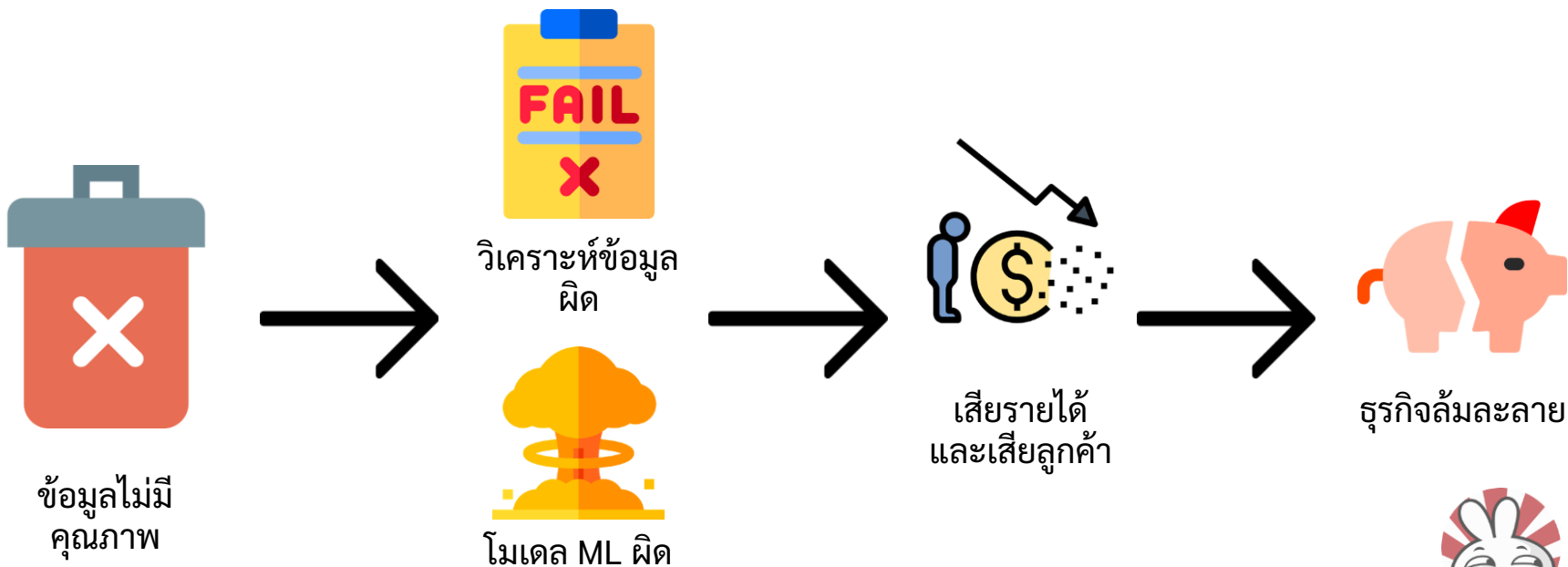
เป็นการพัฒนาคุณภาพของข้อมูล โดยการค้นหาและแก้ไขความผิดพลาดของข้อมูล เช่น

- ข้อมูลไม่ถูกต้องตามโครงสร้าง (Format Error) เช่น อายุติดลบ
- ข้อมูลสูญหาย (Missing Data)
- ข้อมูลสูงกว่าค่าปกติ (Outlier) เช่น อายุ 670 ปี

Tip: ควรทำ Data Cleansing คู่กับผู้เชี่ยวชาญข้อมูลด้านนั้นด้วย เช่น ผู้เชี่ยวชาญทางการแพทย์ จะบอกเราได้ว่าเลือดมีกี่ประเภท หรือค่าความดันแบบไหนที่ผิดปกติ

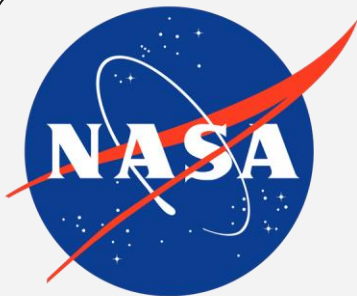
ทำไมต้องทำความสะอาดข้อมูล

Garbage in, Garbage out



Data ที่ไม่สะอาด มีราคาแพง

เศรษฐกิจอเมริกาเสียเงิน **\$3.1 พันล้าน** จากข้อมูลที่ไม่สะอาด ในปี 2016



Case Study: NASA เสียเงิน \$125 ล้าน จากข้อมูลไม่สะอาดในโปรเจก Mars Climate Orbiter

“The peer review preliminary findings indicate that **one team used English units (e.g., inches, feet and pounds)** while the **other used metric units (e.g. centimeter, kilometer, and kilogram)** for a key spacecraft operation.”

ทำไม Data Cleaning ถึงยาก



กระบวนการที่
ไม่มีวันจบสิ้น

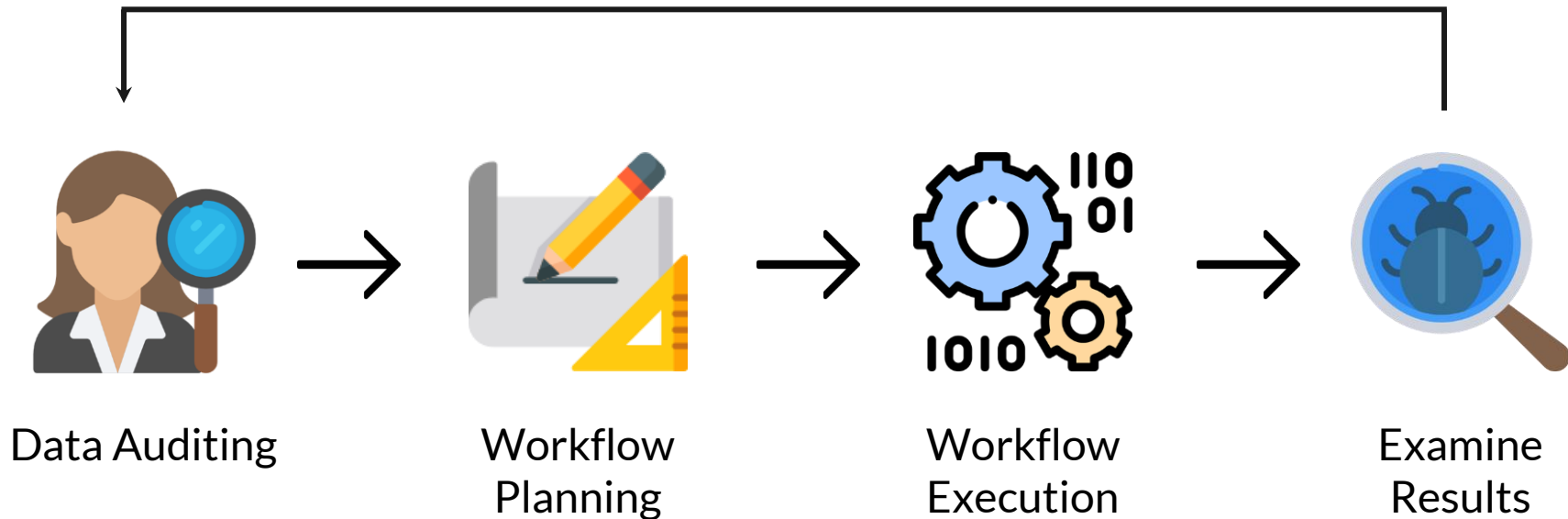


ยากที่จะรู้ว่าเกิด
จากอะไร

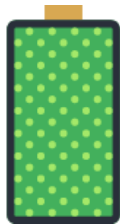


แต่ละ Source มักมี
ข้อมูลโครงสร้าง
แตกต่างกัน

Data cleansing เป็นกระบวนการที่ไม่มีวันจบสิ้น



Data Quality



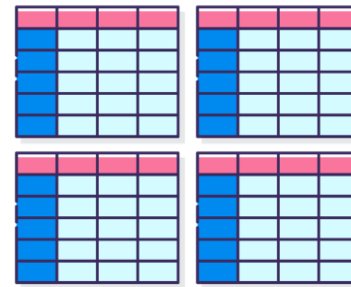
Completeness

ข้อมูลครบ ไม่มี
Missing Values



Validity

ข้อมูลไม่ผิดข้อจำกัด
เช่น อายุไม่ควรติดลบ



Consistency

ข้อมูลจากหลาย Data Source
ควรจะใช้โครงสร้างคล้ายกัน

Data Dictionary

ไฟล์ที่รวบรวมรายละเอียดของทุกคอลัมน์ในตารางข้อมูล เพื่อให้ทุกคนในองค์กรเข้าใจตรงกันและสำคัญมากในการทำความสะดวกข้อมูล

คอลัมน์สำคัญใน Data Dictionary:

- ชื่อคอลัมน์
- ประเภทข้อมูลในคอลัมน์
- ตัวอย่างข้อมูลในคอลัมน์
- คำอธิบายของคอลัมน์

employees

Column	Data type	Nullable	Description
emp_no	integer	not null	Unique employee number
first_name	varchar(100)	not null	Employee first name
last_name	varchar(100)	not null	Employee last name
dob	date	null	Date of birth if known
card_no	char(6)	null	Employee access control card number
edu	integer	not null	Education level 9 - unknown 4 - elementary 3 - Middle school 2 - High school 1 - University
dept_id	integer	not null	Employee department ID. Ref: departments
eval	date	null	Date of last performance review
current	bit	not null	1 - current employee, 0 - past employee

Data Lineage

การเดินทางของข้อมูลตั้งแต่ต้นจนจบ

มีประโยชน์มากในการตรวจสอบ Error ของข้อมูล ว่ามาจากจุดไหนในระบบ



Apache Atlas



Data Catalog

แหล่งรวมข้อมูล และรายละเอียดเกี่ยวกับข้อมูลทั้งหมดให้ฝ่าย Business และ Data Analyst นำข้อมูลไปใช้ต่อได้ง่าย เช่น

- ระบบค้นหาข้อมูล
- Data Dictionary
- ตัวอย่างข้อมูล

ตัวอย่าง: Talend Data Catalog, IBM Watson Knowledge Catalog, Alation

Video: <https://www.talend.com/resources/introduction-to-talend-data-catalog/>



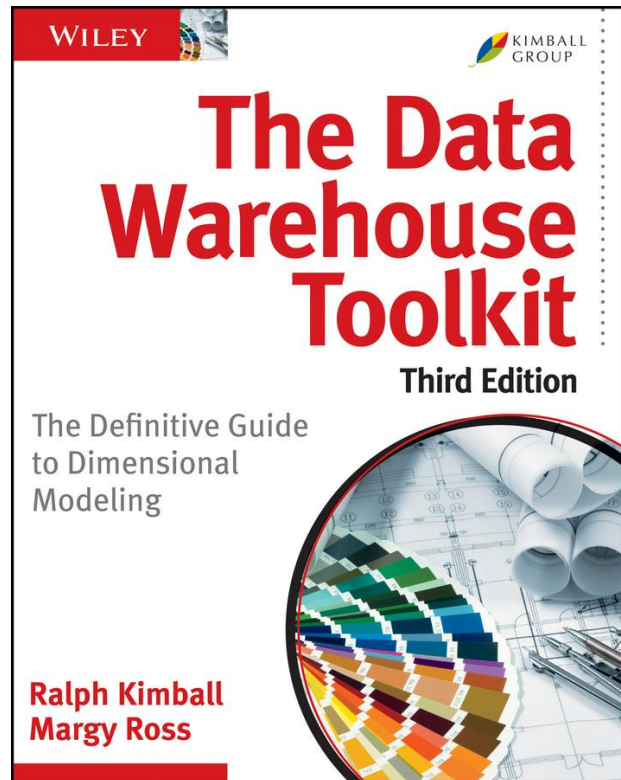
Exploratory Data Analysis

Data Profiling

“Data Profiling ใช้การดึงข้อมูลจากแหล่งข้อมูล เพื่อค้นหาและความสัมพันธ์ แทนที่จะดูจากข้อมูลที่เราจะไม่อัปเดตจากเอกสารการใช้งาน”

- Ralph Kimball, Margy Ross

วิธีการดูภาพรวมของข้อมูล เพื่อหาว่ามีปัญหาอะไรกับข้อมูลมัย



ตัวอย่างของการทำ Data Profiling

ใช้ข้อมูล Hotel Booking
Demand บน Kaggle

<https://www.kaggle.com/jessemestipak/hotel-booking-demand>

	is_canceled	lead_time	arrival_date_year	arrival_date_week_number	arrival_date_day_of_month
count	119390.000000	119390.000000	119390.000000	119390.000000	119390.000000
mean	0.370416	104.011416	2016.156554	27.165173	15.798241
std	0.482918	106.863097	0.707476	13.605138	8.780829
min	0.000000	0.000000	2015.000000	1.000000	1.000000
25%	0.000000	18.000000	2016.000000	16.000000	8.000000
50%	0.000000	69.000000	2016.000000	28.000000	16.000000
75%	1.000000	160.000000	2017.000000	38.000000	23.000000
max	1.000000	737.000000	2017.000000	53.000000	31.000000

EDA

วิธีการเจาะลึกเพื่อดูรายละเอียดของข้อมูล และความสัมพันธ์ของแต่ละคอลัมน์ / แถว

ใช้ตัวเลขหรือกราฟฟิก

1. **แบบใช้ตัวเลข** - ใช้ตัวเลขทางสถิติ เช่น หาค่า Min, Max, Mean
2. **แบบใช้กราฟฟิก** - ใช้การพลอตกราฟ (Data Visualisation) เช่น Boxplot, Histogram

ดูรายละเอียดตัวแปรที่ตัว

1. **ดูทีละตัว (Univariate)** - เช่น ดูค่า Mean ของคอลัมน์เดียว
2. **ดูทีละหลายตัว (Multivariate)** - เช่น การคำนวณ Covariance หรือ การวาด Scatterplot เปรียบเทียบระหว่าง 2 ตัวแปร



EDA แบบใช้ตัวเลข

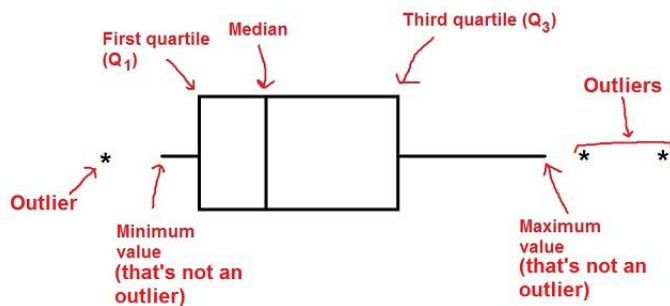
ใช้สถิติเชิงพรรณนา (Descriptive Statistics)

- การกระจายตัวของข้อมูล (Distribution): ค่าเฉลี่ย
- ความแตกต่างของข้อมูล (Variability): ค่าสูงสุด - ต่ำสุด, ค่าความเบี่ยงเบนมาตรฐาน (Standard Deviation) หรือ variance
- ระยะ Interquartile

	is_canceled	lead_time
count	119390.000000	119390.000000
mean	0.370416	104.011416
std	0.482918	106.863097
min	0.000000	0.000000
25%	0.000000	18.000000
50%	0.000000	69.000000
75%	1.000000	160.000000
max	1.000000	737.000000



EDA แบบใช้กราฟฟิก

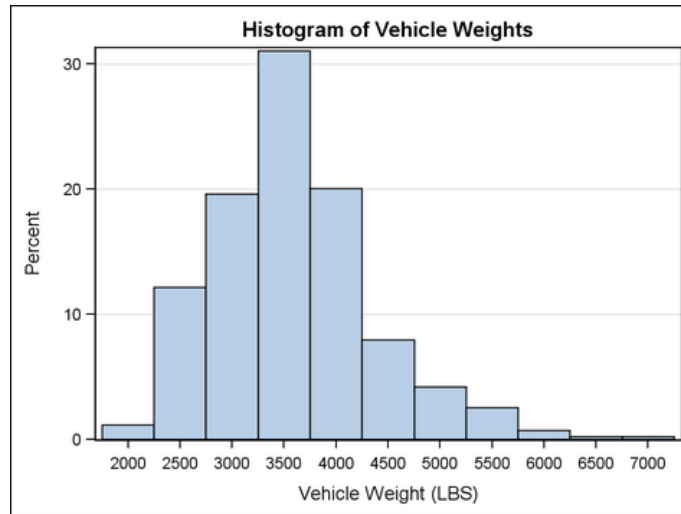


Boxplot

แสดงการกระจายตัวของข้อมูล
และค่าที่เกินจากปกติมาก (Outliers)

https://mrgiomini.blogspot.com/2017_03_20_archive.html

<https://support.sas.com/documentation/cdl/en/grstatgraph/65377/HTML/default/p1sxw5gidyzrygn1ibkzfm5c93m.htm>



Histogram

แสดงการกระจายตัวของข้อมูล (เป็นลักษณะ
ยอดเขาเดียว หรือหลายยอด หรือไม่มียอด
เลย) และความเบี่ยงเบนของข้อมูล
(Skewness)

DataTH.com



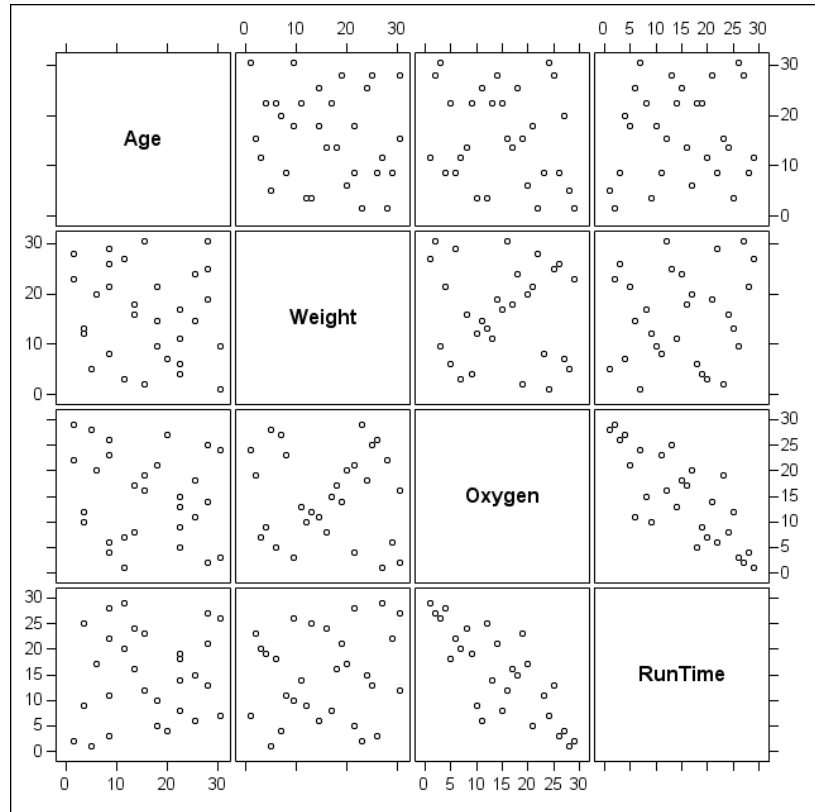
EDA แบบใช้กราฟฟีก (2)

Scatterplot

ใช้หาความสัมพันธ์ระหว่าง 2 ตัวแปร

Scatterplot Matrix

ใช้หาความสัมพันธ์ระหว่างทุกคู่ตัวแปร



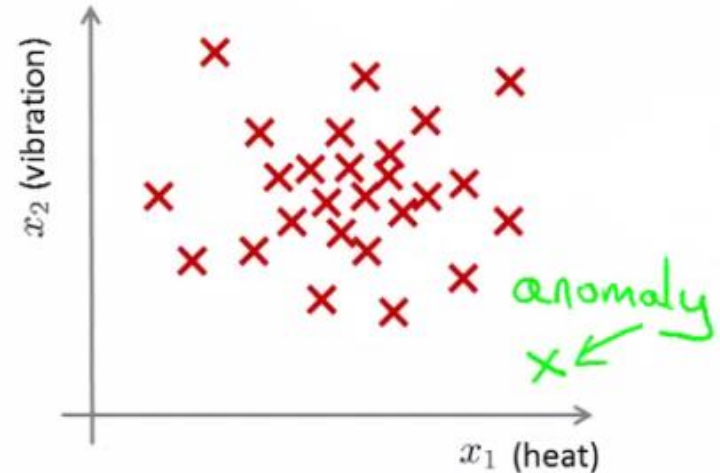
Data Anomaly

What is Data Anomaly

ความผิดปกติของข้อมูลที่เกิดขึ้นได้จากตอนเก็บข้อมูล ซึ่งทำให้ข้อมูลไม่สมบูรณ์

ตัวอย่าง Data Anomaly:

- พิมพ์ผิด (Lexical Error)
- ข้อมูลซ้ำ (Duplication)
- ข้อมูลไม่สม่ำเสมอ (Inconsistency)
- ข้อมูลหายบางส่วน (Missing Values)
- ข้อมูลเกินค่าปกติ (Outliers)
- ฯลฯ



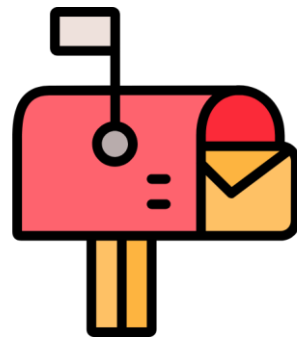
Types of Anomaly

1. **Syntactical Anomalies** - เกิดจากข้อผิดพลาดในการกรอกข้อมูล เช่น พิมพ์ผิด (spelling mistake), ประเภทข้อมูลผิด (domain format error), มีตัวอักษรผิดปกติ (syntactical error), มีค่าแปลก (irregularity)
2. **Semantic Anomalies**: เกิดจากข้อผิดพลาดในการเก็บข้อมูล เช่น เก็บข้อมูลซ้ำ (duplication), เก็บข้อมูลไม่ตรงตามเงื่อนไข (integrity constraint violation)
3. **Coverage Anomalies**: เกิดจากข้อผิดพลาดในความสมบูรณ์ของข้อมูล เช่น ค่าหาย (missing value)

วิธีค้นหาและแก้ไข Syntactical Anomalies

เช่น พิมพ์ผิด (Spelling Mistake) หรือคำแปลก (Irregularities)

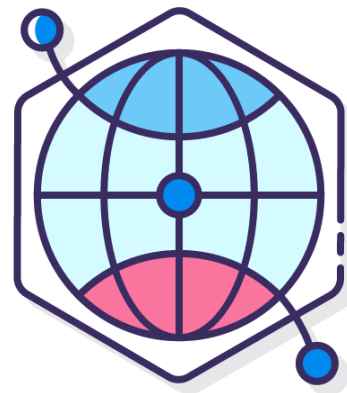
วิธีแก้: หาคำที่ผิดหลักการ เช่น รหัสไปรษณีย์ 6 หลัก และแก้ไขโดยหาแหล่งข้อมูลที่ถูกต้องถ้าสามารถทำได้



Tip: ใช้ข้อมูลออนไลน์ที่มีความถูกต้อง เช่น ข้อมูลชื่อเมือง ใช้ Wikipedia หรือ Google Maps API

วิธีค้นหาและแก้ไข Semantic Anomalies

- ค่าไม่ตรงตามเงื่อนไข (Integrity Constraints)
เช่น อายุติดลบ หาค่าที่ผิด และแก้ไขให้ถูก
- ค่าซ้ำ (Duplication)
เช็คก่อนว่าคอลัมน์นั้นซ้ำได้มั้ย แล้วลบหรือแก้ไขตามความเหมาะสม
- ค่าขัดกัน (Contradictions)
เช่น วันจบมาก่อนวันเริ่ม แก้ไขเหมือนค่าซ้ำ



วิธีหา Syntactical และ Semantics Anomalies

Regular Expression (regex) = pattern สำหรับค้นหาตัวหนังสือ

ทำไม regex ถึงมีประโยชน์

- ฟังก์ชันเยอะ และยืดหยุ่น
- เขียนได้สั้น ๆ
- ง่ายสำหรับการค้นหาและแทนที่
- เรียนครั้งเดียว นำไปใช้ได้หลายภาษาโปรแกรมมิ่ง

ตัวอย่าง

[A-Z] = ค้นหา 1 ตัวอักษร จาก A ถึง Z

[A-Z]+ = ค้นหา 1+ ตัวอักษร จาก A ถึง Z

[a-zA-Z0-9]* = ค้นหา 0+ ตัวอักษร จาก a ถึง z, A ถึง Z, และ 0 ถึง 9

Tip: ใช้ <https://regex101.com/> สำหรับเขียน Regular Expression



Missing Values

ข้อมูลที่หายไปจากระบบ สามารถเกิดโดยบังเอิญ (MAR - Missing At Random) หรือไม่บังเอิญ (MNAR - Missing Not At Random)

สำหรับ Data Analyst, Data Scientist



- มองหาข้อมูลหาย
- ตรวจสอบเช็คข้อมูลหายเพราะอะไร
- หาวิธีแก้ไขปัญหาข้อมูลหายที่เหมาะสมที่สุด เช่น ตัดทิ้งจากการวิเคราะห์ หรือ แทนค่าที่หายด้วยค่ากลาง, Regression Model

สำหรับ Data Engineer



- มองหาข้อมูลหาย
- แจ้งทีมที่เก็บข้อมูล จบ

Tip: ปัญหาถ้าแก้ได้ที่ต้นเหตุ (ทีมเก็บข้อมูล) จะดีกับงานมากกว่าในระยะยาว



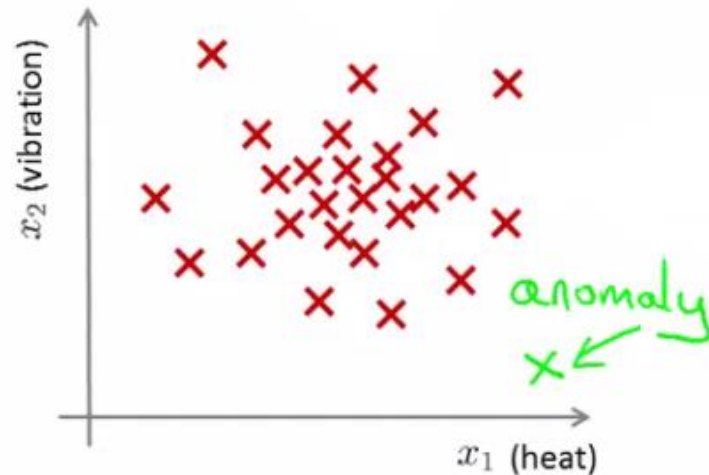
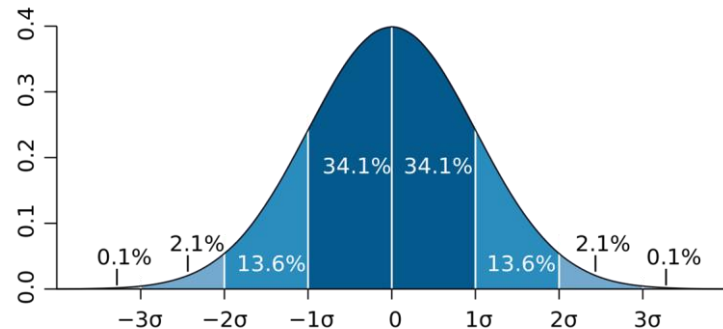
What is Outliers?

Outliers คือ ค่าที่ห่างจากค่าส่วนใหญ่มาก ซึ่งแปลว่าค่า อาจจะผิดปกติ

มีประโยชน์มากสำหรับการตรวจสอบการโกง ในบัตรเครดิต (Fraud Detection)

วิธีการเช็ค Outliers

1. Boxplot
2. กฎ 3 sigma



Distributed Data Processing

Hadoop MapReduce

Popular distributed processing programming paradigm found in 2006 by Yahoo! (based on Google's paper in 2003)



Run on commodity
hardware



Big ecosystem

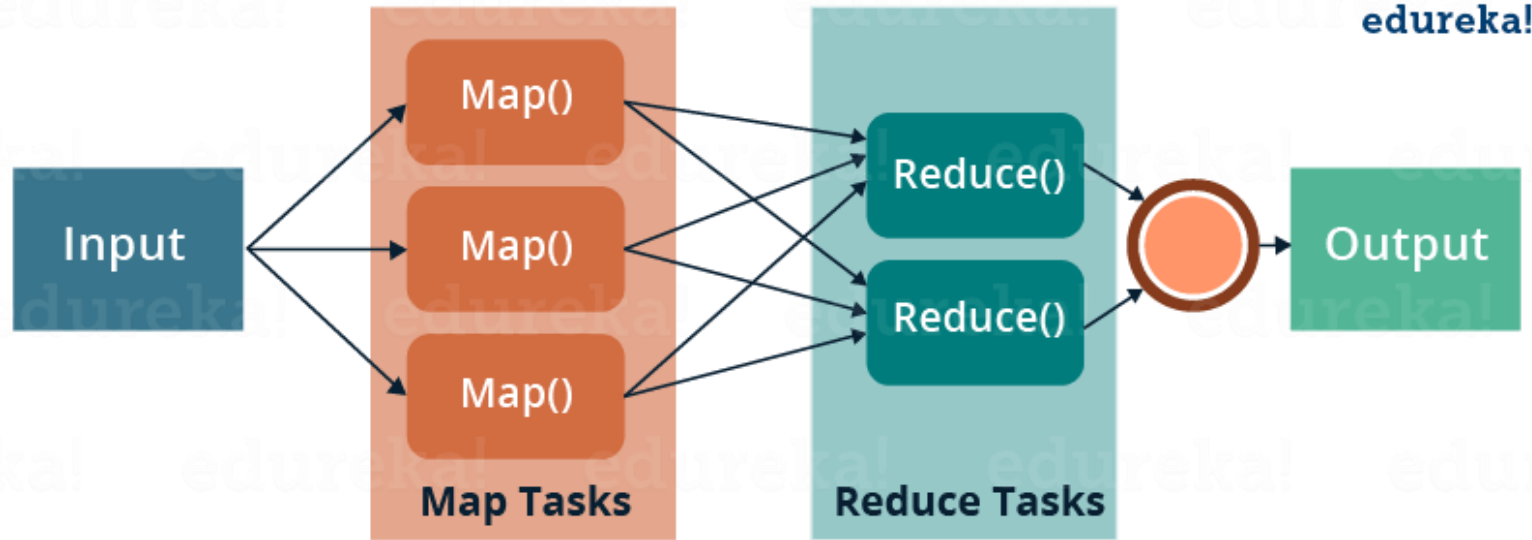


Hard to write &
debug

DataTH.com



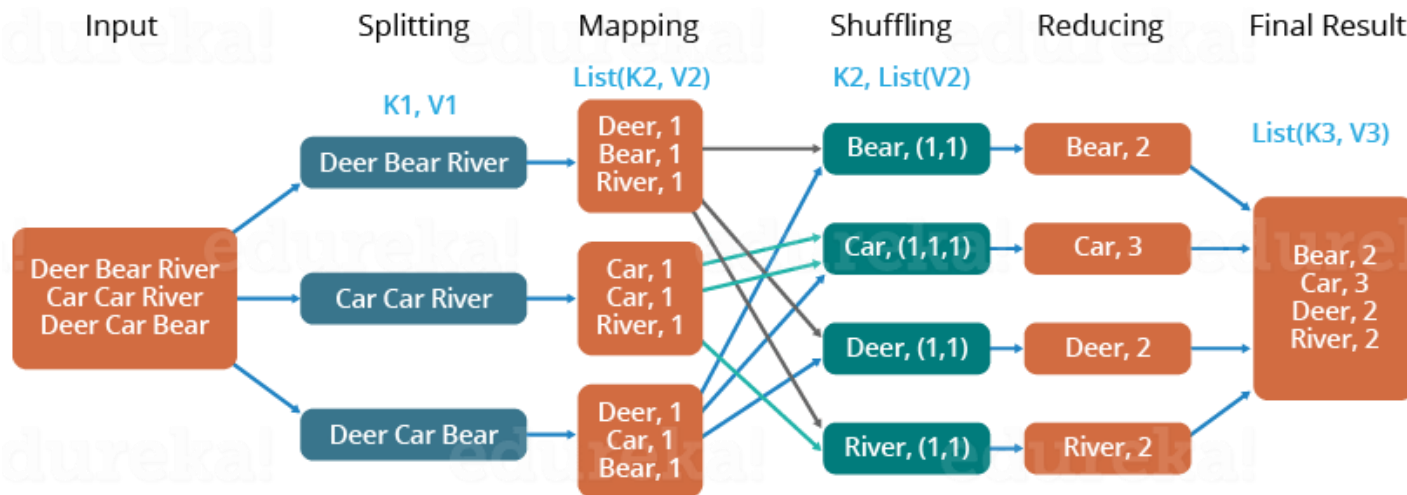
การทำงานของ Hadoop MapReduce



ตัวอย่าง Hadoop MapReduce

The Overall MapReduce Word Count Process

edureka!



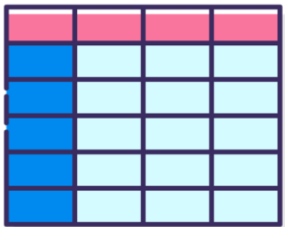
Apache Spark

In-memory distributed data processing found in 2014 at University of California

- เร็วกว่า และใช้พื้นที่ในการประมวลผลน้อยกว่า Hadoop
- Fault-tolerant ด้วย RDD
- มี Component ให้ใช้งานเยอะ เช่น MLlib, Spark SQL, GraphX

	Hadoop MR Record	Spark Record	Spark 1 PB
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400 physical	6592 virtualized	6080 virtualized
Cluster disk throughput	3150 GB/s (est.)	618 GB/s	570 GB/s
Sort Benchmark Daytona Rules	Yes	Yes	No
Network	dedicated data center, 10Gbps	virtualized (EC2) 10Gbps network	virtualized (EC2) 10Gbps network
Sort rate	1.42 TB/min	4.27 TB/min	4.27 TB/min
Sort rate/node	0.67 GB/min	20.7 GB/min	22.5 GB/min

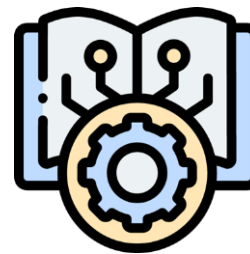
ส่วนเสริมของ Spark



Spark DataFrame
& Spark SQL



GraphX



MLlib

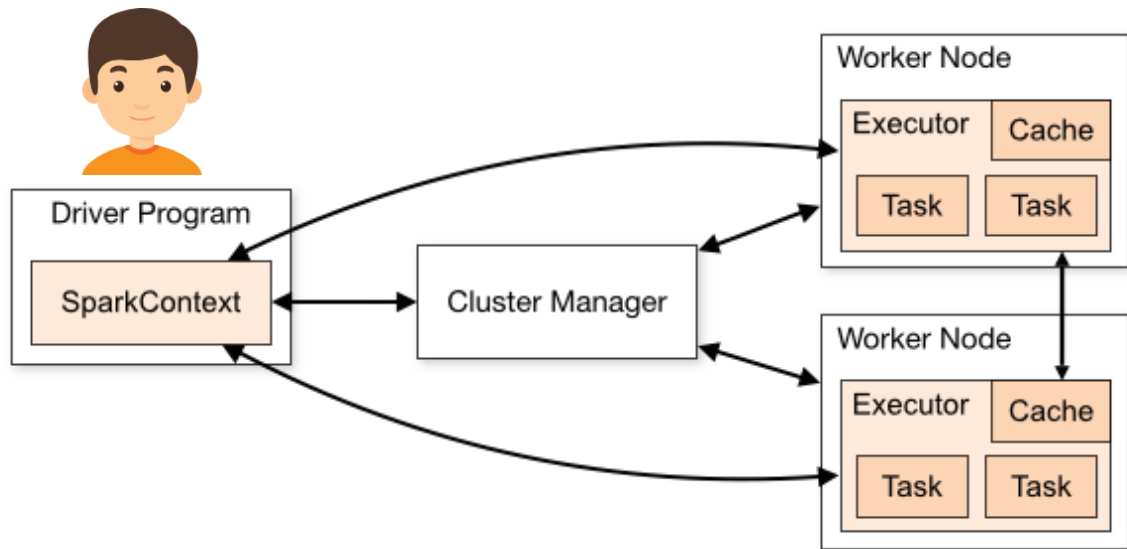


Spark Streaming



SparkR

3 วิธี ในการรันคำสั่ง Spark



1. Spark Shell
2. Programming e.g. PySpark, SparkR
3. Spark Submit





databricks®

Databricks

Data Platform ออนไลน์ โดยทีมพัฒนา Spark

- มี UI แบบ Notebook ให้ใช้งานง่าย รองรับ Python, R, SQL
- รองรับการรัน Spark job ด้วย Spark submit หรือ Pyspark

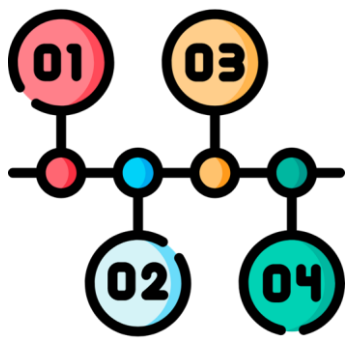
The screenshot shows the Databricks Python notebook interface. The left sidebar contains navigation icons for Home, Workspace, Recent, Data, Clusters, Jobs, and Search. The main area displays three code cells:

- Cmd 1:** Contains valid Python code: `a = 1`, `b = 2`, and `a + b`. The output is `Out[4]: 3`.
- Cmd 2:** Contains `raise new ValueError()`, which results in a `SyntaxError: invalid syntax`.
- Cmd 3:** Contains `err`, which results in a `NameError: name 'err' is not defined`.

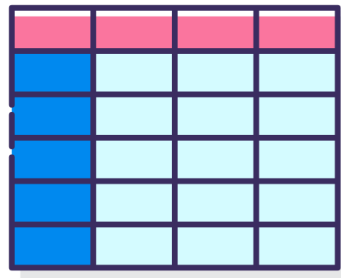


DataTH.com

ประเภทข้อมูลใน Spark



**Resilient Distributed
Datasets (RDD)**
(Java / Scala / Python)
วิธีทำงานกับข้อมูลแบบ
พื้นฐาน มีคำสั่งซับซ้อน



**Spark DataFrame &
Spark SQL (Python / R)**
วิธีทำงานกับข้อมูลแบบ
ตาราง Relational Database



DataSet (Java / Scala)
ข้อมูลแบบ DataFrame ที่มี
การเช็คโค้ดตอน Compile
และบังคับกำหนด Type ข้อมูล

RDD

Resilient Distributed Datasets รองรับการทำงานกับข้อมูลแบบพื้นฐาน
แบ่งเป็นคำสั่งแบบ Transformation กับ Action

Transformation

- ยังไม่ทำงานทันที
- สร้าง RDD ใหม่ทุกครั้ง (ข้อมูล RDD ไม่สามารถเขียนทับได้ - Immutable)
- เช่น map(), filter()

Action

- ทำงานทันที และบังคับ Transformation ก่อนหน้านี้ให้ทำงานด้วย
- ผลลัพธ์ไม่ได้เป็นค่า RDD
- เช่น count(), collect(), take(n)

Tip: Transformation ไม่ทำงานทันที ทำให้เวลารันคำสั่ง Spark เร็วมาก



RDD: Transformation vs Action

Transformation:

5.33 ms

DOES NOT
require calculating
output

```
▶ %%timeit

from pyspark.sql.functions import when

dt.withColumn("CountryNoEIRE", when(dt['Country'] == 'EIRE', 'Ireland').otherwise(dt['Country']))

📄 100 loops, best of 3: 5.33 ms per loop
```

Action:

63.1 ms

require calculating
output

```
▶ %%timeit

from pyspark.sql.functions import when

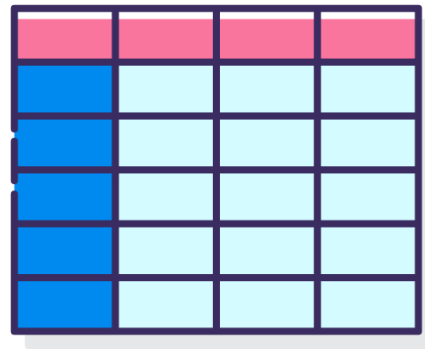
dt.withColumn("CountryNoEIRE", when(dt['Country'] == 'EIRE', 'Ireland').otherwise(dt['Country'])).take(1)

📄 10 loops, best of 3: 63.1 ms per loop
```

Spark DataFrame

แปลงข้อมูลเป็นตาราง เพิ่มวิธีการทำงานกับข้อมูลแบบง่าย ๆ
คล้าย DataFrame ใน Pandas และ R

- คำสั่ง **select** สำหรับเลือกข้อมูลด้วยเงื่อนไขต่าง ๆ
- คำสั่ง **withColumn** สำหรับเพิ่มคอลัมน์ใหม่
- คำสั่ง **na.fill** สำหรับเติมข้อมูล Missing Values
- แปลงเป็น Pandas DataFrame เพื่อทำ Data Visualisation ได้
- พร้อมใช้งานกับ SQL ด้วย Spark SQL



Tip: บางอย่างทำใน DataFrame ง่ายกว่า แต่เราอยากได้ RDD
เราก็สามารถแปลง RDD -> DataFrame -> RDD ได้

Spark SQL

ใช้ **SQL** ในการดึงข้อมูลจาก Spark DataFrame

มีประโยชน์มากสำหรับการเขียน ETL ด้วย Spark



แปลง Spark DataFrame เป็น **TempView** หรือ **GlobalTempView**



ใช้คำสั่ง **SQL** ในการดึงหรือแปลงข้อมูลบน TempView



Spark Cheatsheets

By DataCamp

RDD:

<https://www.datacamp.com/community/blog/pyspark-cheat-sheet-python>

DataFrame:

<https://www.datacamp.com/community/blog/pyspark-sql-cheat-sheet>

Python For Data Science Cheat Sheet

PySpark - SQL Basics

Learn Python for data science [Interactively](#) at [www.DataCamp.com](#)

PySpark & Spark SQL
Spark SQL is Apache Spark's module for working with structured data.

Initializing SparkSession
A SparkSession can be created DataFrame, register DataFrame as tables, execute SQL over tables, cache tables, and read parquet files.

```
>>> from pyspark.sql import SparkSession
>>> spark = SparkSession \
    .builder \
    .appName("Python Spark SQL basic example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

Creating DataFrames

Retrieving RDD Information

Basic Information

Count RDD instances
Count RDD instances by key
Return (key,value) pairs as a dictionary
Sum of RDD elements
Check whether RDD is empty

```
>>> rdd.count()
>>> rdd.countByKey()
>>> rdd.collectAsMap()
>>> rdd.sum()
>>> rdd.isEmpty()
```

Summary

Maximum value of RDD elements
Minimum value of RDD elements
Mean value of RDD elements
Standard deviation of RDD elements
Compute variance of RDD elements
Compute histogram by bins

```
>>> rdd.max()
>>> rdd.min()
>>> rdd.mean()
>>> rdd.stdev()
>>> rdd.var()
>>> rdd.histogram(bins)
```

Applying Functions

Apply a function to each RDD element
Apply a function to each RDD element and return the result
Apply a Python function to each RDD element and return the result

```
>>> rdd.map(lambda x: x*2)
>>> rdd.flatMap(lambda x: x.split())
>>> rdd.foreach(lambda x: print(x))
>>> rdd.foreachPartition(lambda iter: iter.collect())
```

Getting

Return a list with all RDD elements
Take first RDD element
Take first RDD element
Take top RDD elements

```
>>> rdd.collect()
>>> rdd.first()
>>> rdd.take(1)
>>> rdd.takeOrdered(10)
```

Sampling

Return sampled subset of RDD

```
>>> rdd.sample(False, 0.1, 42, 1)
```

Filtering

Filter the RDD
Return distinct RDD values
Return (key,value) RDD keys

```
>>> rdd.filter(lambda x: x % 2 == 0)
>>> rdd.distinct()
>>> rdd.keys()
```

Iterating

Apply a function to all RDD elements

```
>>> rdd.foreach(lambda x: print(x))
>>> rdd.foreachPartition(lambda iter: iter.collect())
```

Duplicate Values

```
>>> df = df.dropDuplicates()
```

Queries

Show all entries in DataFrame column
Show all entries in DataFrame, age and type
Show all entries in DataFrame and age, add 1 to the entries of age
Show all entries where age >= 24
Show DataFrame and count depending on age >= 30
Show DataFrame if in the given options like

```
>>> from pyspark.sql import functions as F
>>> df.select("first_name", "last_name") \
    .show()
>>> df.select("first_name", "last_name") \
    .select(F.col("first_name").alias("first_name_output"),
           F.col("last_name").alias("last_name_output")) \
    .show()
>>> df.select((F.col("first_name") + df("age") + 1) \
    .show()
>>> df.select((df("age") > 24).show())
>>> df.select("first_name",
           F.when(df.age > 30, 1) \
    .show()
>>> df(df.first_name.inList("James", "Maria") \
    .collect())
```

Reshaping Data

Reducing

Merge the RDD values for each key
Return the RDD values

```
>>> rdd.reduce(lambda x, y: x + y)
>>> rdd.reduceByKey(lambda x, y: x + y)
>>> rdd.reduceByKey(lambda x, y: x + y)
```

Grouping by

Return RDD of grouped values
Group by column key

```
>>> rdd.groupBy(lambda x: x % 2) \
    .collect()
>>> rdd.groupByKey() \
    .collect()
>>> rdd.groupByKey().mapValues(lambda x: x.sum()) \
    .collect()
```

Aggregating

Aggregate RDD elements of each partition and then the results
Aggregate the elements of each partition, and then the results
Create summary of RDD elements by applying a function

```
>>> rdd.aggregate(lambda x: x, (0, 0), lambda x, y: x + y)
>>> rdd.aggregateByKey(lambda x, y: x + y, lambda x, y: x + y)
>>> rdd.aggregateByKey(lambda x, y: x + y, lambda x, y: x + y)
>>> rdd.aggregateByKey(lambda x, y: x + y, lambda x, y: x + y)
>>> rdd.aggregateByKey(lambda x, y: x + y, lambda x, y: x + y)
```

Mathematical Operations

Return each RDD value not contained in RDD
Return each RDD value not contained in RDD
Return the Cartesian product of RDD and RDD
Return the Cartesian product of RDD and RDD

```
>>> rdd.subtract(rdd2)
>>> rdd.subtract(rdd2)
>>> rdd.subtract(rdd2)
>>> rdd.subtract(rdd2)
>>> rdd.subtract(rdd2)
```

Sort

Sort RDD by given function
Sort RDD by value RDD key

```
>>> rdd.sortBy(lambda x: x)
>>> rdd.sortBy(lambda x: x)
>>> rdd.sortBy(lambda x: x)
>>> rdd.sortBy(lambda x: x)
>>> rdd.sortBy(lambda x: x)
```

Repartitioning

New RDD with partitions
Decrease the number of partitions in the RDD to 1

```
>>> rdd.repartition(1)
>>> rdd.repartition(1)
```

Saving

Save RDD as Parquet file
Save RDD as Parquet file
Save RDD as Parquet file
Save RDD as Parquet file
Save RDD as Parquet file

```
>>> rdd.saveAsParquetFile("path")
>>> rdd.saveAsParquetFile("path")
>>> rdd.saveAsParquetFile("path")
>>> rdd.saveAsParquetFile("path")
>>> rdd.saveAsParquetFile("path")
```

Stopping SparkSession

```
>>> sc.stop()
```

Execution

Run Python code in the Spark shell

```
>>> %run ./python_code.py
```

GroupBy

Group by age, count the members in the groups

```
>>> df.groupBy("age") \
    .count() \
    .show()
```

Filter

Filter entries of age, only keep those records of which the values are >= 24

```
>>> df.filter(df("age") >= 24).show()
```

Sort

Sort RDD by given function
Sort RDD by value RDD key

```
>>> rdd.sortBy(lambda x: x)
>>> rdd.sortBy(lambda x: x)
>>> rdd.sortBy(lambda x: x)
>>> rdd.sortBy(lambda x: x)
>>> rdd.sortBy(lambda x: x)
```

Missing & Replacing Values

Replace null values
Return new RDD omitting rows with null values
Return new RDD of replacing one value with another

```
>>> df.na.fill(0)
>>> df.na.drop()
>>> df.na.replace(1, 20)
```

Repartitioning

at with to partitions
at with to partitions
at with to partitions
at with to partitions
at with to partitions

```
>>> df.repartition(10) \
    .show()
>>> df.coalesce(1).rdd.getPartitions() \
    .show()
```

Running SQL Queries Programmatically

Registering DataFrames as Views

Register DataFrame as view
Register DataFrame as view
Register DataFrame as view
Register DataFrame as view
Register DataFrame as view

```
>>> df.createOrReplaceTempView("temp")
>>> df.createOrReplaceTempView("temp")
>>> df.createOrReplaceTempView("temp")
>>> df.createOrReplaceTempView("temp")
>>> df.createOrReplaceTempView("temp")
```

Query Views

Return the results of the query
Return the results of the query
Return the results of the query
Return the results of the query
Return the results of the query

```
>>> sql = spark.sql("SELECT * FROM customers")
>>> sql.show()
>>> sql.show()
>>> sql.show()
>>> sql.show()
```

Output

Data Structures

Convert RDD to DataFrame
Convert RDD to DataFrame
Convert RDD to DataFrame
Convert RDD to DataFrame
Convert RDD to DataFrame

```
>>> rdd = df.rdd
>>> df.toLocalIterator()
>>> df.toLocalIterator()
>>> df.toLocalIterator()
>>> df.toLocalIterator()
```

Write & Save to Files

Save RDD to file
Save RDD to file
Save RDD to file
Save RDD to file
Save RDD to file

```
>>> df.save("path")
>>> df.save("path")
>>> df.save("path")
>>> df.save("path")
>>> df.save("path")
```

Stopping SparkSession

```
>>> spark.stop()
```

DataCamp

Learn Python for Data Science [Interactively](#)

Workshop 2:

Data Cleansing with Spark



Workshop 2 - Data Cleansing with Spark

มาลองใช้ PySpark, Spark SQL, และ Pandas เพื่อให้ข้อมูลของเรามีคุณภาพ

Input:

- ข้อมูลที่เราดึงมาแล้ว (CSV)
- Notebook ที่รองรับ Spark, PySpark, Pandas



Output:

- ข้อมูลที่ทำความสะอาดเรียบร้อยแล้ว (CSV)