

數據分析團隊

Spark 教育訓練

01 Hadoop 架構及資料處理

數據經營部

李嘉桓



國泰人壽

Cathay Life Insurance

教材文件：

\\cxlsvr174\國壽商業分析教育資源(期限20221231)\01 課程資料\2020課程資料\20200818 Spark教育訓練

教材 Gitlab：

http://10.95.42.31:8282/i9h00211/hadoop_spark_practice

課堂聊天室：

<https://tlk.io/i9h00>



一般使用者

- 分析最新的即時數據
- 商業分析報告
- 使用業務管轄內的資料表



企劃分析人員



分析資源

重度使用者

- 分析現在及大量過往歷史數據
- AI 模型建置及技術應用
- 使用多個跨業務別資料表



資料科學家

問題痛點

- TD 好用但成本太高，要有地方存放歷史資料隨時供資料科學家使用
- 越來越多非結構化資料，如文本、圖檔、音檔有分析需求
- 現有的分析資源是足夠的，但人壽的自主分析能力強，連線數容易滿載，需要分流
- 集團架構對齊需求



集團 Hadoop 由來

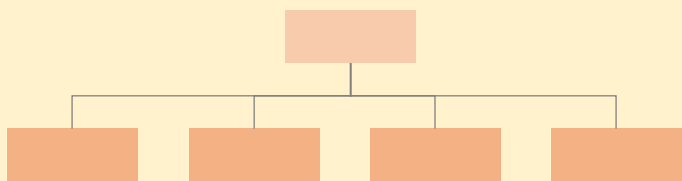
銀行數據部為了解決類似問題，進行組織轉型並建立以 Hadoop 為中心的開發環境，提供數據團隊使用。並成立金控數數發團隊，專門處理集團數據相關問題。



HAP 及 Datalake 環境

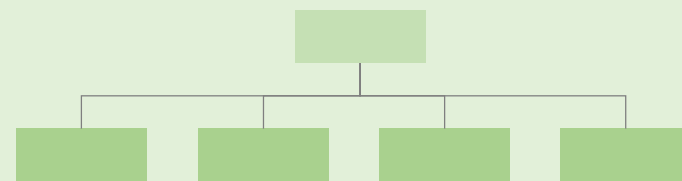
集團 Hadoop 共有兩座，分別叫 HAP (Hadoop Analytics Platform) 及 Datalake，兩者的差別在於，一個為開發探索用，一個為上線營運用。後者只有排程要上線才會用到。

HAP 環境



- 使用者為數據團隊開發者
- 資料皆為遮蔽後暗碼
- 有比較大的 CPU、RAM 運算資源
- 有 GPU 可以使用

Datalake 環境



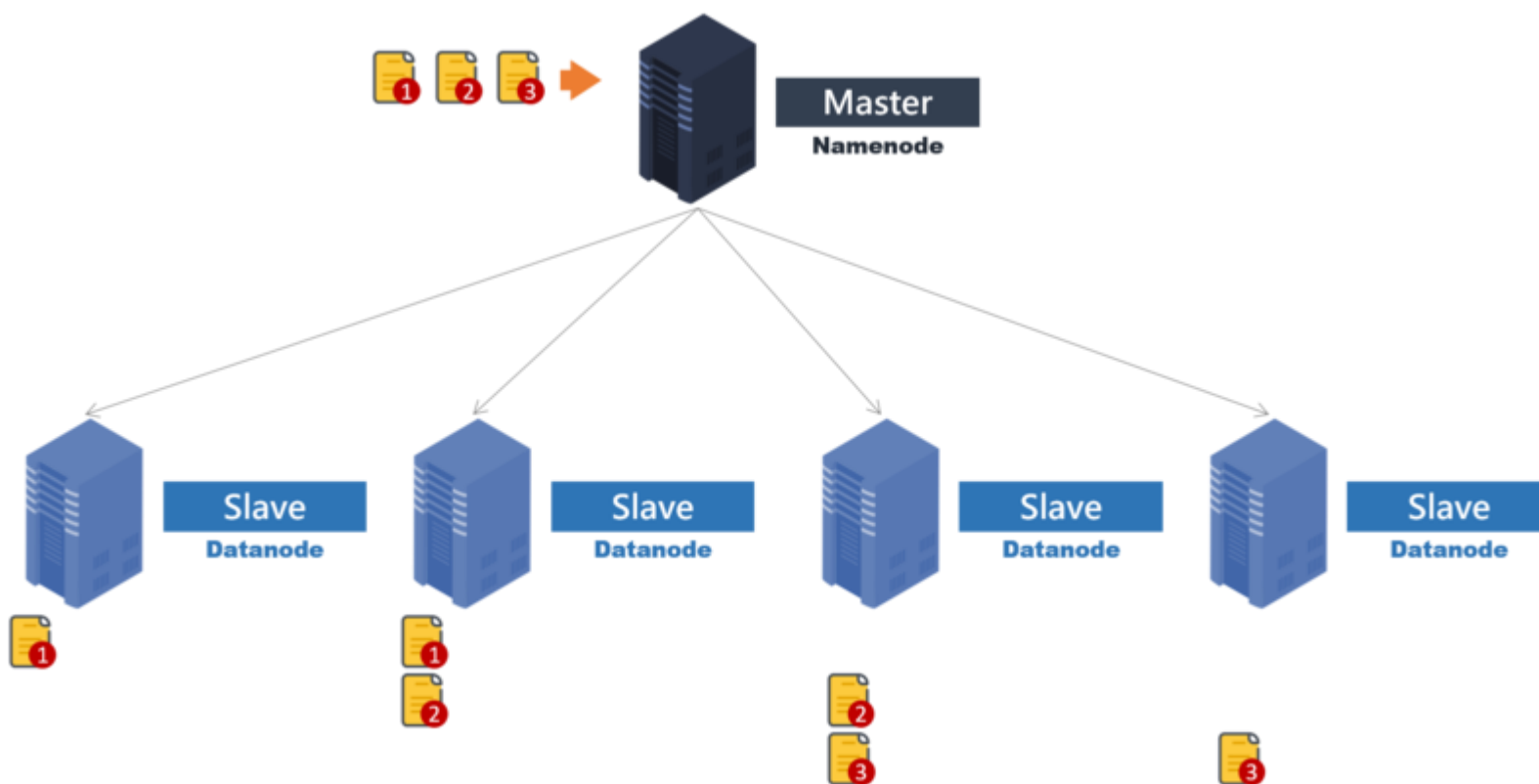
- 使用者為 Hadoop 平台管理員
- 資料與 HAP 一致，但明暗碼共存
- 僅須滿足營運用硬體資源即可
- 目前無 GPU 可使用

後面皆以HAP環境做為範例說明



Hadoop 架構

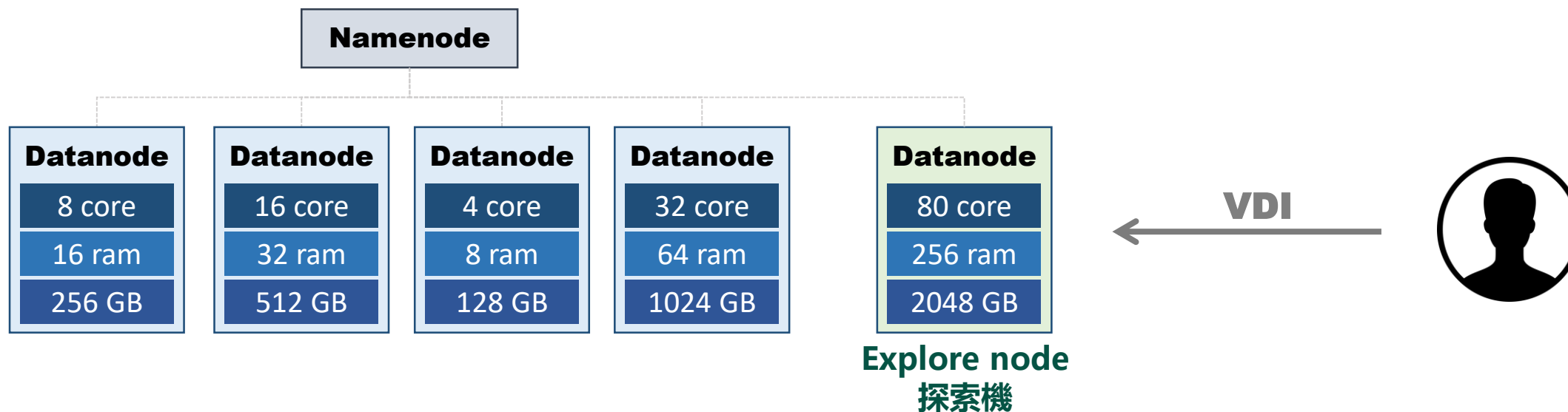
Hadoop 是叢集的架構，又叫 Cluster 架構，由一個 Master 大腦節點與多個 Slave 工作節點組成。最早的功能是分散式儲存 (HDFS架構)，透過低成本的方式備援資料。



站在系統角度通常會用 Master 和 Slave 稱呼節點，若為資料科學使用，會習慣稱 Namenode 及 Datanode。

分散式儲存到分散式運算

每個工作節點除了儲存資源，同時也有閒置的運算資源，為了效益最大化，延伸出分散式運算架構，讓叢集除了儲存外也能進行開發，並且讓使用者可以連線進去叢集中進行操作。



以上圖為例，整個叢集有 140 core + 376 ram 的運算能力



Explore Node 探索機

每個子集團在 HAP 都有一台主要的 Explore Node 探索機，提供使用者連線到 Hadoop 進行開發，也是 HAP 起 Python server 及 R server 服務的地方。



使用Pyspark、Rspark、MLlib開發
(整個叢集資源)

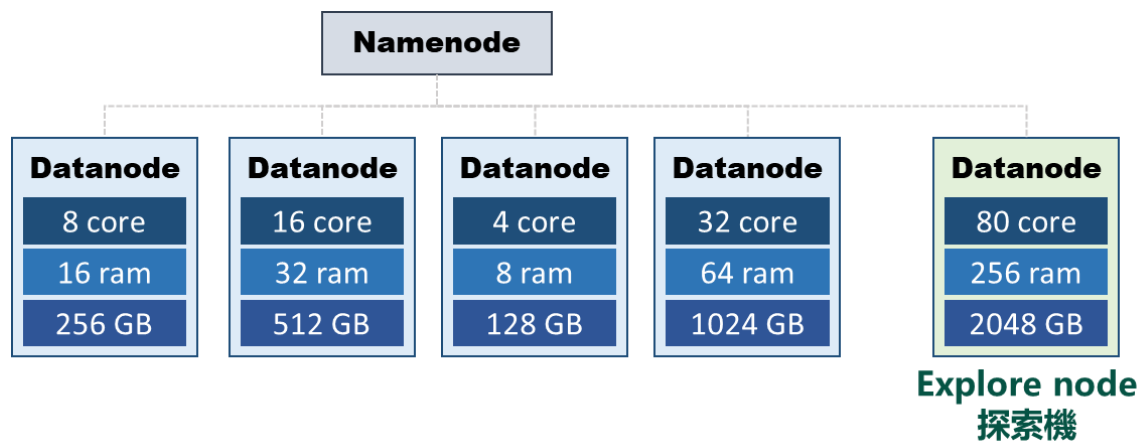


使用 Python、R、sklearn開發
(Explore node 單機資源)

如果要使用到 Hadoop 的資源，就必須使用相關的語言或套件開發！！

Hadoop 運算框架

Hadoop 最基礎的分散式運算框架為 MapReduce，會將每個工作分配到各個 Datanode 協助運算，提高系統效率。



執行 Hadoop 運算的 framework



主要透過Hue連接使用，可以透過 SQL 語法輕鬆進行資料篩選
(公司現有的 IMPALA 有不明 BUG，建議不要用)

Hue介紹-1

Hue 是目前使用 Hadoop 資料最簡易的途徑，定位為類似 SAS-EG 的工具，使用前提必須向平台管理者提出帳號申請，並透過 VDI 連線使用。(連結網址詳見 Wiki Hue)

The image shows two parts of the Hue interface. On the left is the login page with a 'Welcome to Hue' message, a sign-in button, and a list of databases including 'default', 'life_user_dm', and 'life_v_dm'. On the right is the main query editor interface. The top navigation bar includes 'Query Editors' and 'Data Browsers'. The 'Query Editors' dropdown menu is open, showing options like 'Hive', 'Impala', 'DB Query', 'Pig', and 'Job Designer'. A red annotation '選擇 Hive 使用，其他功能有問題建議別用' points to the 'Hive' option. Below the menu is a text area for writing queries, with a red annotation '打 Hive SQL 的區塊' pointing to it. Below the text area are buttons for 'Execute', 'Save as...', 'Explain', 'Format', and 'New query'. Below these buttons is a table of recent queries with columns for '時間' (Time) and 'Query'. A red annotation '顯示結果的區塊' points to the 'Query' column. The table contains several queries, including 'SELECT * FROM life_user_dm.AG_BUYTAG' and 'Load Data Inpath ... OVERWRITE INTO TABLE life_user_dm.AG_BUYTAG'.

選擇 Hive 使用，其他功能有問題建議別用

打 Hive SQL 的區塊

所有人都會有兩個資料視圖可以使用

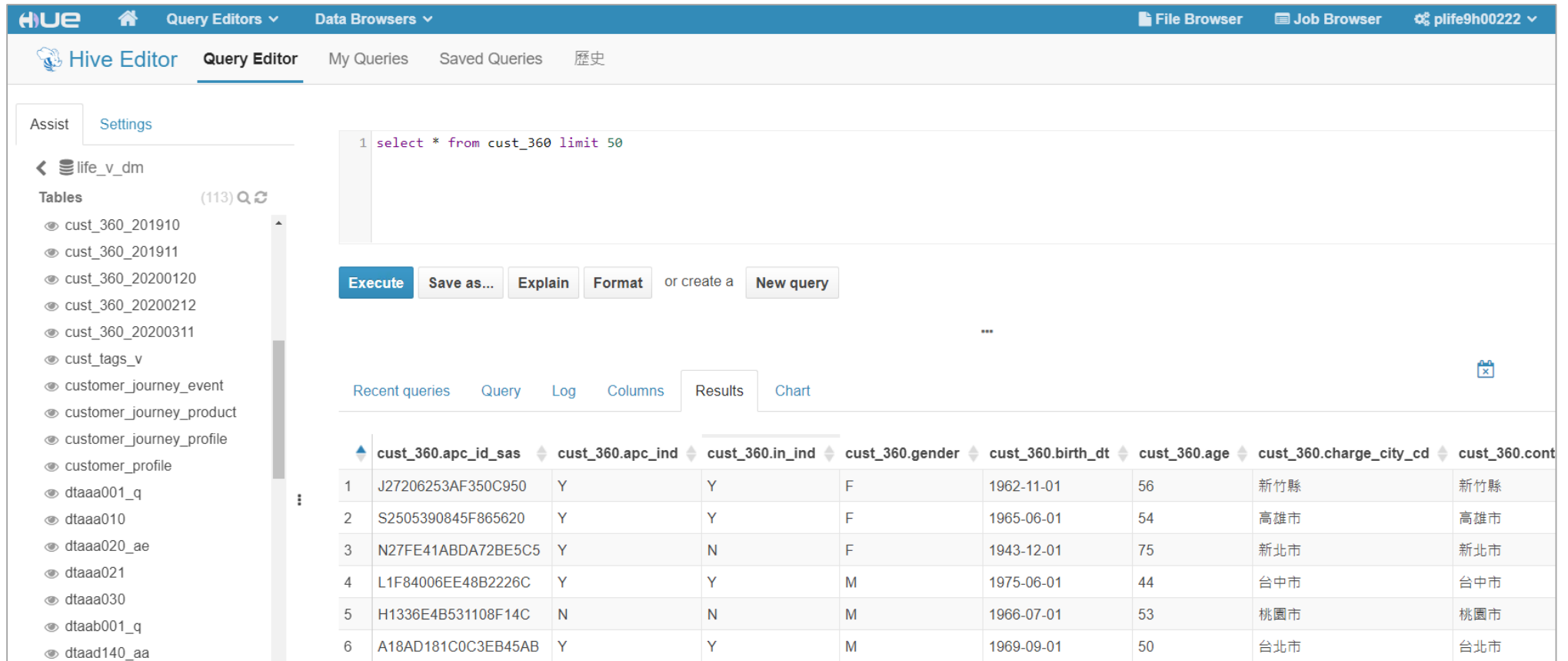
1. View 資料視圖 (life_v_dm)
→ TD 現有自主/CR的資料
2. User 資料視圖 (life_user_dm)
→ 開發者自建的 Table

顯示結果的區塊



Hue介紹-2

Hive SQL 與一般 SQL 無太大差異，且在 User 資料視圖可以自行 Create / Drop Table，因資料視圖跟 TD 一樣以科為單位開權限，務必注意不要誤刪科內同仁的 Table。



The screenshot displays the Hue web interface. At the top, there's a navigation bar with 'HUE', a home icon, and tabs for 'Query Editors', 'Data Browsers', 'File Browser', and 'Job Browser'. Below this, the 'Hive Editor' is active, showing a 'Query Editor' tab with a SQL query: `1 select * from cust_360 limit 50`. Below the query editor are buttons for 'Execute', 'Save as...', 'Explain', 'Format', and 'New query'. The 'Results' tab is selected, showing a table with 8 columns: `cust_360.apc_id_sas`, `cust_360.apc_ind`, `cust_360.in_ind`, `cust_360.gender`, `cust_360.birth_dt`, `cust_360.age`, `cust_360.charge_city_cd`, and `cust_360.cont`. The table contains 6 rows of data. On the left, a sidebar shows a tree view of tables under 'life_v_dm', including 'cust_360_201910', 'cust_360_201911', and others.

	<code>cust_360.apc_id_sas</code>	<code>cust_360.apc_ind</code>	<code>cust_360.in_ind</code>	<code>cust_360.gender</code>	<code>cust_360.birth_dt</code>	<code>cust_360.age</code>	<code>cust_360.charge_city_cd</code>	<code>cust_360.cont</code>
1	J27206253AF350C950	Y	Y	F	1962-11-01	56	新竹縣	新竹縣
2	S2505390845F865620	Y	Y	F	1965-06-01	54	高雄市	高雄市
3	N27FE41ABDA72BE5C5	Y	N	F	1943-12-01	75	新北市	新北市
4	L1F84006EE48B2226C	Y	Y	M	1975-06-01	44	台中市	台中市
5	H1336E4B531108F14C	N	N	M	1966-07-01	53	桃園市	桃園市
6	A18AD181C0C3EB45AB	Y	Y	M	1969-09-01	50	台北市	台北市

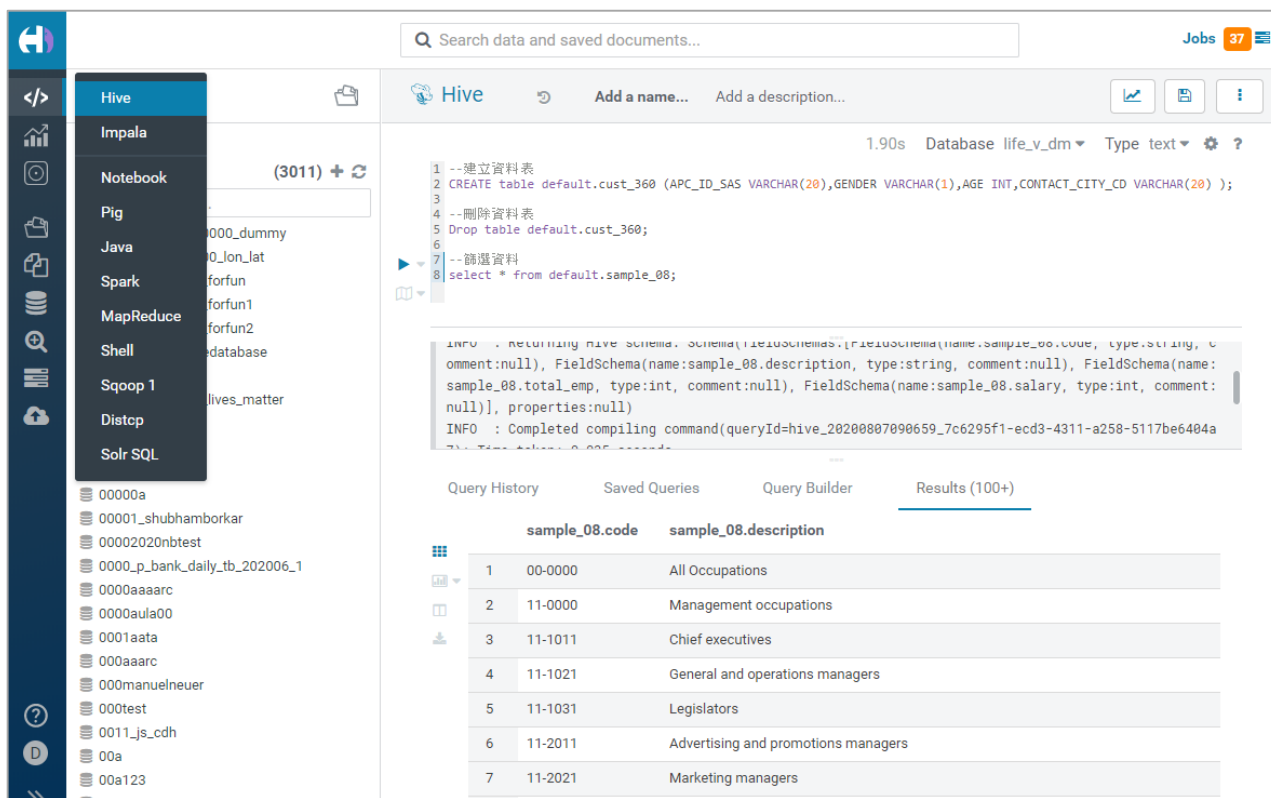
從 Cust_360 篩選前 50 筆資料範例



Hue Demo 環境

請各位使用外部的 Hue Demo 環境，嘗試 Hive SQL 的撰寫，感受一下與 SAS-EG 的差異。

Demo 網址：<http://demo.gethue.com/>



The screenshot displays the Hue Demo web interface. On the left is a sidebar with navigation icons and a file browser. The main area is titled 'Hive' and shows a SQL editor with the following code:

```
1 --建立資料表
2 CREATE table default.cust_360 (APC_ID_SAS VARCHAR(20),GENDER VARCHAR(1),AGE INT,CONTACT_CITY_CD VARCHAR(20) );
3
4 --刪除資料表
5 Drop table default.cust_360;
6
7 --篩選資料
8 select * from default.sample_08;
```

Below the editor, the 'Results (100+)' tab is active, showing a table with 7 rows of data:

	sample_08.code	sample_08.description
1	00-0000	All Occupations
2	11-0000	Management occupations
3	11-1011	Chief executives
4	11-1021	General and operations managers
5	11-1031	Legislators
6	11-2011	Advertising and promotions managers
7	11-2021	Marketing managers

--建立資料表

Create table default.cust_360 (APC_ID_SAS VARCHAR(20),GENDER VARCHAR(1),AGE INT,CONTACT_CITY_CD VARCHAR(20));

--刪除資料表

Drop table default.cust_360;

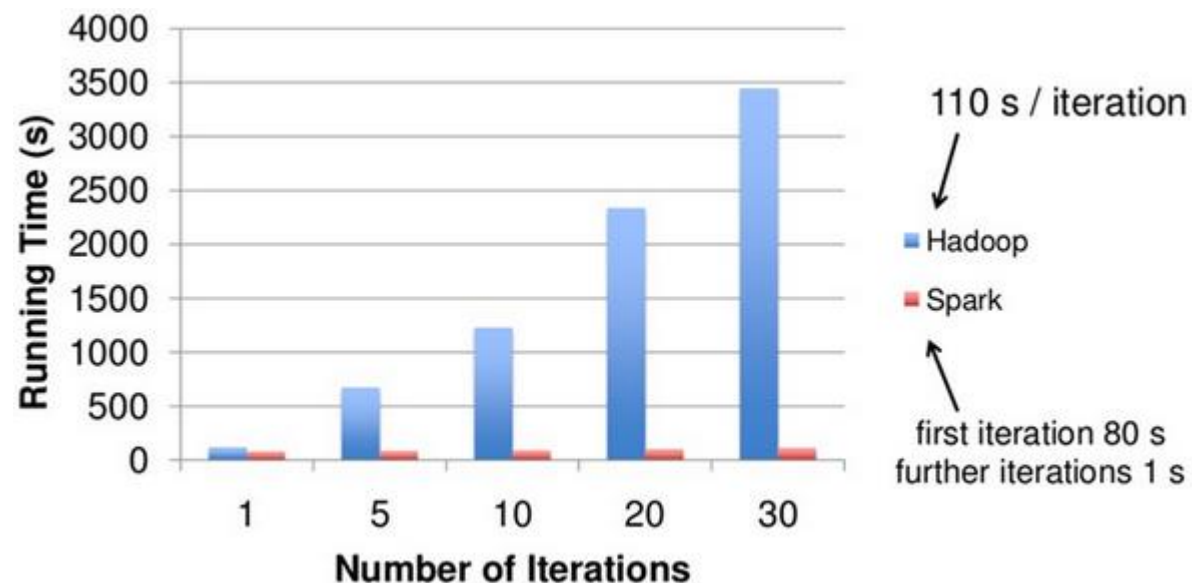
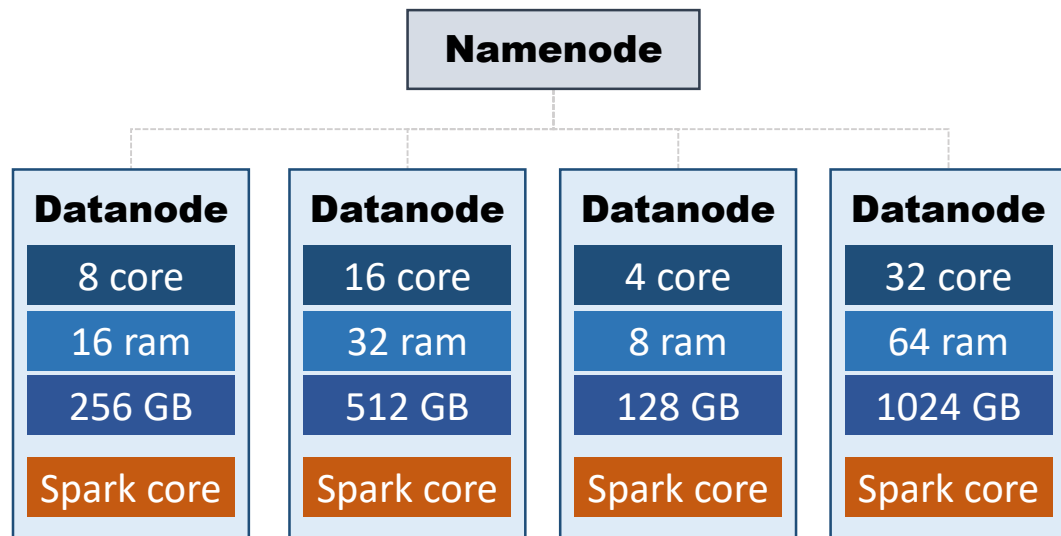
--篩選資料

Select * from default.sample_08;



Spark 架構

MapReduce 雖然可以解決一般的資料篩選需求，但遇到大量資料時，運算效率明顯不足，隨著架構演進，誕生了市場上最流行的 Hadoop 運算框架 - Spark。



Spark core

Spark 的核心由 Scala 驅動，但學習曲線高且無法快速支援機器學習需求，故衍伸出 Pyspark、Rspark 等支援框架，輔助資料科學家使用 Hadoop 資源。

架構概念示意圖

	Pyspark	TF
開發語言	Python	Python
運算框架	Spark	Tensorflow
框架語言	Scala	CUDA
底層系統	HDFS	GPU

	Scala	Python
Performance	10X faster than Python	Slower
Learning Curve	Not easy for Java people; Less programmer;	Easier than Scala; More examples; More popular;
Concurrency	Based on JVM (better)	Does not support heavy-weight process fork
Type Safety	Statically typed	Dynamically typed
Advanced Features	Spark Streaming	Machine Learning



Spark 處理資料的方法有三種，分別為 RDD、DataFrame、Spark SQL：

RDD

RDD 的數據沒有定義 Schema，使用上要有 Map/Reduce 的概念，屬於高階程式技巧，但相對功能最齊全，擁有所有 Spark 功能。然而 Python 用 RDD 會比 Scala 慢，而且有另外兩種替代方法，所以不需要特別去學。

DataFrame

使用 DataFrame 時，Python 與 Scala 的速度沒差多少，而且比 RDD 快，所以是使用者最常使用的資料處理方式之一，缺點就是語法跟 Pandas 略有差異，許多操作需要重新學習，而且建立時需要建立 Schema。

Spark SQL

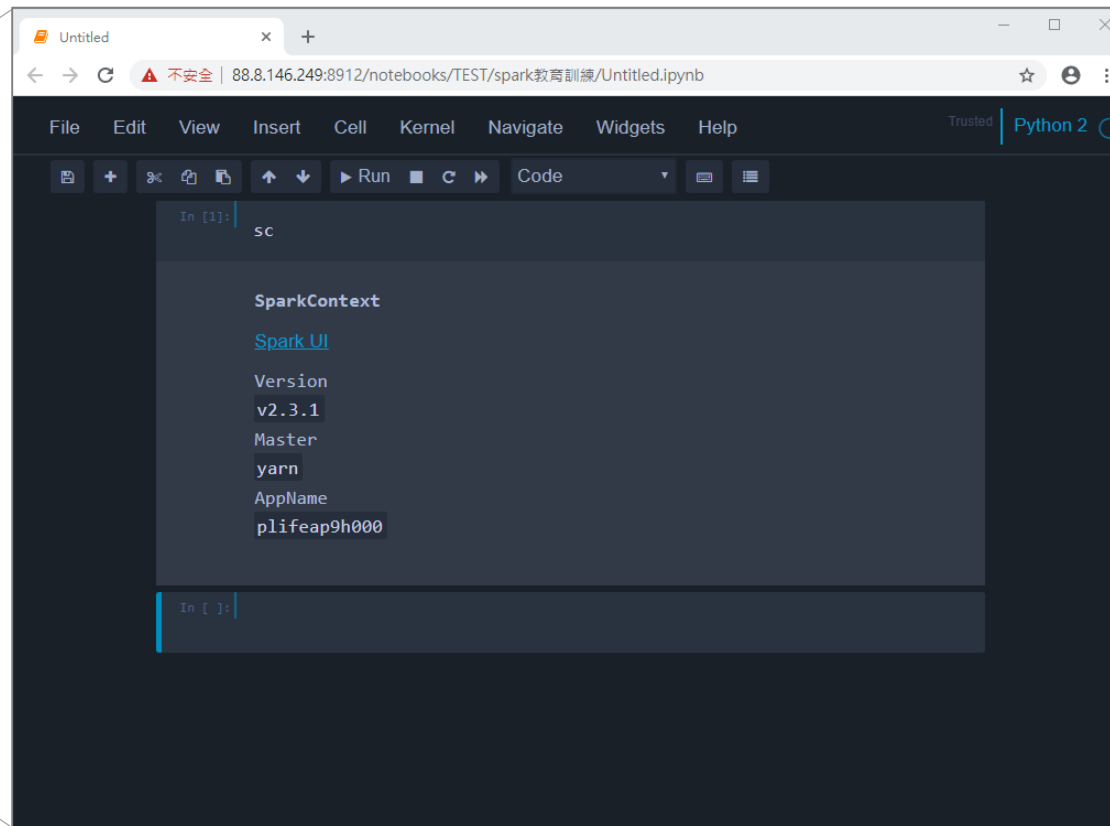
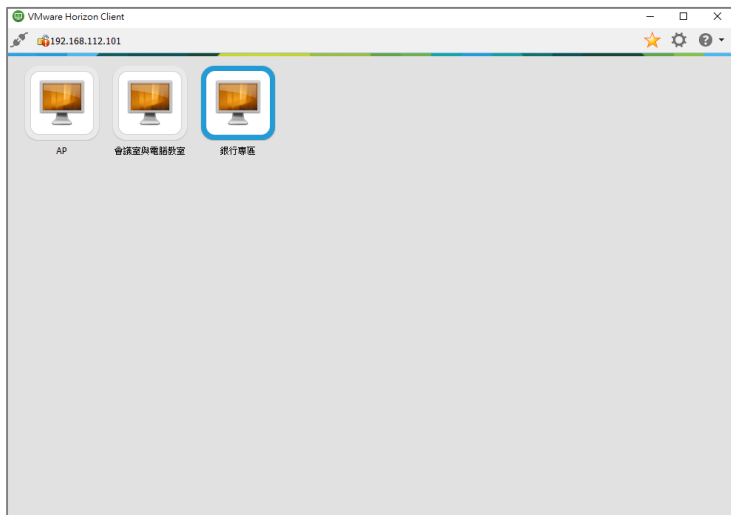
從 DataFrame 衍伸而來，透過登錄 Spark SQL temp table，就可以快速使用 SQL 語法探索資料，幾乎沒有學習難度。

使用難易度： SparkSQL > DataFrame > RDD



開啟開發環境

Hadoop 系統的所有操作，與公司現有的開發環境一樣，都要進去 VDI (虛擬桌面) 使用，唯一不同的是，連線的桌面要選擇銀行專區，原因是 Hadoop 系統為銀行代管。



瀏覽器打上 URL 即可連線 HAP 開發環境，可以下 sc 確認目前 Spark 運行狀態
URL 資訊詳見 Wiki 或工程科人員



篩選第一筆資料

跟在Hue上面的操作一樣，把 Hive SQL 語法打在 spark.sql() 使用，即可從 Hive Table篩選想要的資料。

```
In [2]: sql = '''
select
APC_ID_SAS,
GENDER,AGE,
CONTACT_CITY_CD,
EDUCATION_CD,
MARRIAGE_CD,
STAFF_IND,
LIFE_INSD_CNT,
HEIGHT,
WEIGHT
from life_v_dm.cust_360 limit 100 '''
```

```
In [3]: df = spark.sql(sql)
df.show()
```

APC_ID_SAS	GENDER	AGE	CONTACT_CITY_CD	EDUCATION_CD	MARRIAGE_CD	STAFF_IND	LIFE_INSD_CNT	HEIGHT	WEIGHT
J27206253AF350C950	F	56.0	新竹縣		1	1	0	2.0	158.0
S2505390845F865620	F	54.0	高雄市				0	1.0	null
N27FE41ABDA72BE5C5	F	75.0	新北市		1	1	0	0.0	null
L1F84006EE48B2226C	M	44.0	台中市		3	0	0	1.0	null
H1336E4B531108F14C	M	53.0	桃園市		1	1	0	0.0	null
A18AD181C0C3EB45AB	M	50.0	台北市				0	2.0	null
K1EBB8A445322D39A5	M	44.0	台中市		3	1	0	2.0	null
E2C32CE02CBF92A96C	F	58.0	高雄市		1	1	0	1.0	null
E1E8C1C4B25610BD7D	M	46.0	高雄市		3	1	0	0.0	182.0

--篩選Cust360 前100筆資料

Select APC_ID_SAS, GENDER, AGE, CONTACT_CITY_CD,
EDUCATION_CD, MARRIAGE_CD, STAFF_IND,
LIFE_INSD_CNT, HEIGHT,WEIGHT
From life_v_dm.cust_360 limit 100



Spark 練習環境

因 VDI 只能透過公司內網連線，不易所有人同時練習，故請遵從以下步驟，啟用 Colab 開發環境練習後續課程，筆記本會自動儲存在自身的 Google drive 中。

1 使用 Chrome 上網搜尋 Colab，打開並新增筆記本，命名可以自訂

2 安裝 pyspark

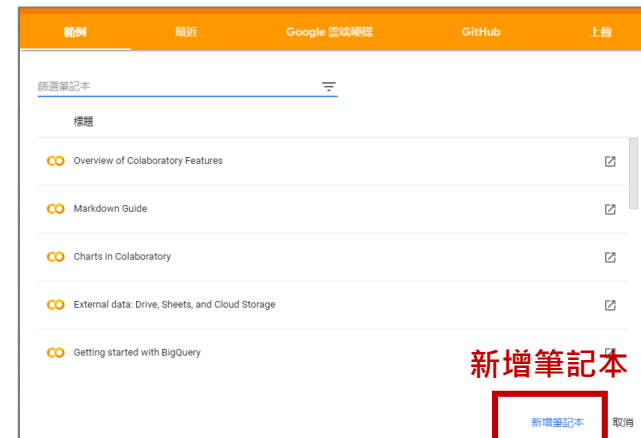
```
!pip install pyspark
```

3 啟動 spark 服務

```
from pyspark.sql import SparkSession
from pyspark import SparkContext
spark = SparkSession.builder.master("local").getOrCreate()
sc = SparkContext.getOrCreate()
```

4 打上 sc 確認環境運行正常

Colab 一段時間沒動就會重啟連線，需要重複上述步驟。



因為沒辦法連線公司資料庫，故從外部載入測試用資料：

從外部抓CUST360的測試資料

```
from pyspark import SparkFiles
```

```
url = 'https://raw.githubusercontent.com/chia313339/Spark_practice/master/CUST_360.csv'
```

```
spark.sparkContext.addFile(url)
```

```
df = spark.read.csv(SparkFiles.get("CUST_360.csv"), header=True, inferSchema=True)
```

將資料集讀取在memory上，DataFrame跟SparkSQL都要下此語法，如果沒下，資料每次都會重新load

```
df.cache()
```

執行這句才會真的運行Spark，將資料存在記憶體

```
df.show()
```

COL NAME	欄位名稱
APC_ID_SAS	SAS ID
GENDER	性別
AGE	年齡(年)
CONTACT_CITY_CD	聯絡地址_縣市
EDUCATION_CD	教育程度/學歷
MARRIAGE_CD	婚姻狀況
STAFF_IND	是否為公司員工
LIFE_INSD_CNT	目前主約被保有效件數(件)
HEIGHT	身高
WEIGHT	體重

APC_ID_SAS	GENDER	AGE	CONTACT_CITY_CD	EDUCATION_CD	MARRIAGE_CD	STAFF_IND	LIFE_INSD_CNT	HEIGHT	WEIGHT
Q2D129E954AD477523	F	50	台中市	2	1	0	4	158	54
A2AFDA91C172B15A77	F	35	高雄市	3	0	0	2	162	48
F1D4EF7FFBE44A51FE	M	33	新北市	3	0	0	2	172	70
T117FCDED3FAE1C1F3	M	31	屏東縣	3	0	0	3	167	58
L26284A55F56B398BA	F	55	高雄市	3	1	0	2	161	55
E1F5DBF55F6B1427DD	M	60	高雄市	1	0	0	1	167	72
F20481473EEB92E8D8	F	37	新北市	3	0	0	3	160	54
H20BE9CA7E806E0551	F	71	基隆市	3	1	0	2	150	47
F1B81805D82921E3B9	M	27	新北市	null	null	0	1	182	64
N1ED4D2C4D60B78323	M	57	彰化縣	2	1	0	8	182	76
F12A8F9FE08045F1A6	M	42	新北市	3	1	0	6	174	63
S2FBC9D681C459E262	F	46	高雄市	1	1	0	2	160	53
T24571F44004FE8145	F	59	屏東縣	4	1	0	2	163	58
P2A8A6EE60E936FA24	F	53	雲林縣	2	1	0	1	153	57
J1E82E68A12B9D234D	M	54	高雄市	3	1	0	2	180	80
J1882AEE9E2183B202	M	56	新竹縣	null	1	0	1	173	68
Q221C6DF46F85DA0D3	F	59	嘉義縣	1	1	0	5	157	58
B29C7859D7C9244C3A	F	45	新北市	null	1	0	2	160	52
A2663738EDFF7839CA	F	37	台北市	2	1	0	1	166	65
R24A96ABFCF4742B5C	F	32	台南市	3	0	0	3	159	62

only showing top 20 rows



Spark cache()

Spark 的 Dataframe 與常用的 Pandas 完全不一樣，是特殊的資料表型態，僅有[執行運算](#)後，Spark 才會開始動作。

Spark 沒反應

```
df = spark.sql("select * from life_v_dm.cust_360")
```

Spark 沒反應

```
df.cache()
```

Spark 開始運行

```
df.show()
```

執行運算才會開始做資料篩選及邏輯處理，否則只是單純將指令派發給Spark，不會開始運算。

常見的有 `show()`、`count()`、`head()`、`describe()`



Spark Dataframe

使用 spark.sql 從 Hive 讀取出來的資料皆為 Spark Dataframe，可以使用 pyspark 語法進行簡單的資料處理。

篩選年齡大於25歲的客戶

```
df.select('APC_ID_SAS', 'AGE').where(df.AGE > 25).show()
```

```
+-----+-----+
|      APC_ID_SAS | AGE |
+-----+-----+
| Q2D129E954AD477523 | 50 |
| A2AFDA91C172B15A77 | 35 |
| F1D4EF7FFBE44A51FE | 33 |
| T117FCDED3FAE1C1F3 | 31 |
| L26284A55F56B398BA | 55 |
| E1F5DBF55F6B1427DD | 60 |
| F20481473EEB92E8D8 | 37 |
| H20BE9CA7E806E0551 | 71 |
| F1B81805D82921E3B9 | 27 |
| N1ED4D2C4D60B78323 | 57 |
| F12A8F9FE08045F1A6 | 42 |
| S2FBC9D681C459E262 | 46 |
| T24571F44004FE8145 | 59 |
| P2A8A6EE60E936FA24 | 53 |
| J1E82E68A12B9D234D | 54 |
| J1882AEE9E2183B202 | 56 |
| Q221C6DF46F85DA0D3 | 59 |
| B29C7859D7C9244C3A | 45 |
| A2663738EDFF7839CA | 37 |
| R24A96ABFCF4742B5C | 32 |
+-----+-----+
only showing top 20 rows
```

計算各學歷客戶數

```
df.select('EDUCATION_CD').groupby(df.EDUCATION_CD).count().show()
```

```
+-----+-----+
| EDUCATION_CD | count |
+-----+-----+
|          null | 19138 |
|              1 | 22184 |
|              3 | 30428 |
|              4 |  3967 |
|              2 | 24283 |
+-----+-----+
```



Spark Dataframe

Spark Dataframe 使用上會跟 Pandas 差異頗大，例如新增欄位要使用 withColume 語法。

新增欄位以公尺為單位計算身高，並從高到低排序

```
df.select('APC_ID_SAS', 'HEIGHT').withColumn('HEIGHT_M', df.HEIGHT/100).orderBy('HEIGHT_M', ascending=False).show()
```

APC_ID_SAS	HEIGHT	HEIGHT_M
F1057CE5E949E6EFC1	201	2.01
ACE8FCF8C6B46669D3	198	1.98
H1B5DCDA759441F52C	196	1.96
F13ACDDC993E9E53D9	196	1.96
P1C1B81800B3987528	196	1.96
C1E1615E25FC949658	195	1.95
H1804B3E772A5FE82E	195	1.95
R111E1AA083CCA93D9	195	1.95
F1C81BFF76B64F7029	195	1.95
F166C8AE8B99A075F0	195	1.95
E12E48BB533DBC264E	195	1.95
E17A0620F60388202F	195	1.95
M1FB9517B8D97BEC30	193	1.93
F1980A9D354E0DE785	193	1.93
R157828AF5AE7D5593	193	1.93
L1252D0A9147B3B57C	193	1.93
E152DE2A6C1C9D7C52	193	1.93
R195BE7E6A9A99F9BB	193	1.93
A1F2427D4A6C93A26B	193	1.93
B18167CA3279E1551B	193	1.93

only showing top 20 rows



Spark SQL 是將 Spark Dataframe 轉換為可以用 SQL 語法操作的 Temp Table，只要跟 Dataframe 一樣有先 `cache()`，運算速度是一樣的。

```
# register temp table
df.registerTempTable('cust_360')
```

篩選年齡大於25歲的客戶

```
spark.sql('select APC_ID_SAS, AGE from cust_360 where AGE > 25').show()
```

APC_ID_SAS	AGE
Q2D129E954AD477523	50
A2AFDA91C172B15A77	35
F1D4EF7FFBE44A51FE	33
T117FCDED3FAE1C1F3	31
L26284A55F56B398BA	55
E1F5DBF55F6B1427DD	60
F20481473EEB92E8D8	37
H20BE9CA7E806E0551	71
F1B81805D82921E3B9	27
N1ED4D2C4D60B78323	57
F12A8F9FE08045F1A6	42
S2FBC9D681C459E262	46
T24571F44004FE8145	59
P2A8A6EE60E936FA24	53
J1E82E68A12B9D234D	54
J1882AEE9E2183B202	56
Q221C6DF46F85DA0D3	59
B29C7859D7C9244C3A	45
A2663738EDFF7839CA	37
R24A96ABFCF4742B5C	32

only showing top 20 rows

計算各學歷客戶數

```
spark.sql('select EDUCATION_CD, count(1) from cust_360 group by EDUCATION_CD').show()
```

EDUCATION_CD	count(1)
null	19138
1	22184
3	30428
4	3967
2	24283

Spark SQL 就跟一般的 SQL 一樣，非常適合 pyspark 不熟的使用者，可以透過 Spark SQL 組出要分析的 ABT。

新增欄位以公尺為單位計算身高，並從高到低排序

```
spark.sql('select APC_ID_SAS, HEIGHT, HEIGHT/100 as HEIGHT_M from cust_360 order by HEIGHT_M desc').show()
```

APC_ID_SAS	HEIGHT	HEIGHT_M
F1057CE5E949E6EFC1	201	2.01
ACE8FCF8C6B46669D3	198	1.98
H1B5DCDA759441F52C	196	1.96
F13ACDDC993E9E53D9	196	1.96
P1C1B81800B3987528	196	1.96
C1E1615E25FC949658	195	1.95
H1804B3E772A5FE82E	195	1.95
R111E1AA083CCA93D9	195	1.95
F1C81BFF76B64F7029	195	1.95
F166C8AE8B99A075F0	195	1.95
E12E48BB533DBC264E	195	1.95
E17A0620F60388202F	195	1.95
M1FB9517B8D97BEC30	193	1.93
F1980A9D354E0DE785	193	1.93
R157828AF5AE7D5593	193	1.93
L1252D0A9147B3B57C	193	1.93
E152DE2A6C1C9D7C52	193	1.93
R195BE7E6A9A99F9BB	193	1.93
A1F2427D4A6C93A26B	193	1.93
B18167CA3279E1551B	193	1.93

only showing top 20 rows

Spark 清除 cache()

Spark 建立的 cache，可以透過以下語法清除記憶體의 暫存。

DataFrame

load

```
df = spark.sql("select * from life_v_dm.cust_360")  
或 df = spark.read.csv(SparkFiles.get("CUST_360.csv"), header=True, inferSchema=True)
```

delete

```
del df
```

Spark SQL

load

```
df.registerTempTable('cust_360')
```

delete

```
spark.catalog.dropTempView("cust_360")
```



嘗試用 Dataframe 及 Spark SQL 解以下問題：

1 篩選客戶ID、身高、體重，並計算每個客戶 BMI 值，由小排到大顯示。

2 根據上面的 BMI 欄位，新增欄位 BMI_L 顯示 BMI 等級，規則如下。

過輕： $BMI < 18.5$

正常： $18.5 \leq BMI < 24$

過重： $24 \leq BMI$



真的很簡單的

資料篩選練習

1

篩選客戶ID、身高、體重，並計算每個客戶 BMI 值，由小排到大顯示。

DataFrame

```
df.select('APC_ID_SAS', 'HEIGHT', 'WEIGHT').withColumn('BMI', df.WEIGHT / (df.HEIGHT/100)**2).orderBy('BMI', ascending=True).show()
```

Spark SQL

```
spark.sql('select APC_ID_SAS, HEIGHT, WEIGHT, WEIGHT/sqrt(HEIGHT/100) as BMI from cust_360 order by BMI asc').show()
```

APC_ID_SAS	HEIGHT	WEIGHT	BMI
E105DCAFC305033417	49	0	0.0
H29DC5A204152934EE	43	0	0.0
A126C93E33B4D4CB31	172	0	0.0
A15C24ACBA604E7241	49	0	0.0
Q19F6BBF162DFE9D3E	151	3	1.3157317661506074
A236E76AECC53C9FDC	130	3	1.7751479289940826
P210C79C71D3C77171	155	5	2.0811654526534857
A268ED34D77A261522	165	6	2.203856749311295
P135C8C1E11AE16244	172	7	2.3661438615467825
A1110745FF4F29418D	55	1	3.305785123966942
A1E7457FDC4C53B285	53	1	3.5599857600569593
N11FE8FC93D938A2D9	51	1	3.8446751249519417
H2E3A538DEC2F497A8	50	1	4.0
H2FDF7168B2AC5F914	50	1	4.0
E1033E78F40656C84A	48	1	4.340277777777778
F1083B205D44EC8DE3	48	1	4.340277777777778
A1ECCB57B3EBAFFE96	47	1	4.526935264825713
E24ED6669B9EC94CD1	47	1	4.526935264825713
Q2A12E52284107B5CB	105	5	4.535147392290249
J26B1BE57AA148FE8C	80	3	4.687499999999999

only showing top 20 rows



資料篩選練習

2

根據上面的 BMI 欄位，新增欄位 BMI_L 顯示 BMI 等級，規則如下。

過輕：BMI<18.5

正常：18.5<=BMI<24

過重：24<=BMI

Spark SQL

- WAY 1 -

```
spark.sql('select APC_ID_SAS, HEIGHT, WEIGHT, WEIGHT/sqrt(HEIGHT/100) as BMI,
  case when WEIGHT/sqrt(HEIGHT/100) < 18.5 then "過輕"
when WEIGHT/sqrt(HEIGHT/100) < 24 then "正常"
else "過重" end as BMI_L from cust_360').show()
```

- WAY 2 - registerTempTable()

```
spark.sql('select APC_ID_SAS, HEIGHT, WEIGHT, WEIGHT/sqrt(HEIGHT/100) as
  BMI from cust_360').registerTempTable('cust_360_tmp')
```

```
spark.sql('select *,
  case when BMI < 18.5 then "過輕"
when BMI < 24 then "正常"
else "過重" end as BMI_L
from cust_360_tmp').show()
```

APC_ID_SAS	HEIGHT	WEIGHT	BMI	BMI_L
E105DCAFC305033417	49	0	0.0	過輕
H29DC5A204152934EE	43	0	0.0	過輕
A126C93E33B4D4CB31	172	0	0.0	過輕
A15C24ACBA604E7241	49	0	0.0	過輕
A1110745FF4F29418D	55	1	1.348399724926484	過輕
A1E7457FDC4C53B285	53	1	1.3736056394868903	過輕
N11FE8FC93D938A2D9	51	1	1.4002800840280099	過輕
H2FDF7168B2AC5F914	50	1	1.414213562373095	過輕
H2E3A538DEC2F497A8	50	1	1.414213562373095	過輕
E1033E78F40656C84A	48	1	1.4433756729740643	過輕
F1083B205D44EC8DE3	48	1	1.4433756729740643	過輕
E24ED6669B9EC94CD1	47	1	1.4586499149789456	過輕
A1ECCB57B3EBAFFE96	47	1	1.4586499149789456	過輕
S2FCC785A60EE77479	44	1	1.5075567228888183	過輕
Q19F6BBF162DFE9D3E	151	3	2.4413653763134784	過輕
A236E76AEC53C9FDC	130	3	2.6311740579210876	過輕
J2260DE7F4BF665AA0	57	2	2.6490647141300876	過輕
P2EEDBCB64F8559B50	53	2	2.7472112789737806	過輕
K2B23B42CA97EA5D74	52	2	2.7735009811261455	過輕
H2C482C6D7A4C03EAA	51	2	2.8005601680560197	過輕

only showing top 20 rows



2

根據上面的 BMI 欄位，新增欄位 BMI_L 顯示 BMI 等級，規則如下。

過輕：BMI<18.5

正常：18.5<=BMI<24

過重：24<=BMI

DataFrame

```
df_tmp = df.select('APC_ID_SAS', 'HEIGHT', 'WEIGHT').withColumn('BMI', df.WEIGHT / (df.HEIGHT / 100) ** 2)
```

- WAY 1 - withColumn()

```
from pyspark.sql.functions import when
df_tmp.withColumn("BMI_L", when(df_tmp.BMI < 18.5, '過輕').when(df_tmp.BMI < 24, '正常').otherwise('過重')).show()
```

- WAY 2 - select()

```
from pyspark.sql.functions import when
df_tmp.select('*', when(df_tmp.BMI < 18.5, '過輕').when(df_tmp.BMI < 24, '正常').otherwise('過重').alias('BMI_L')).show()
```

- WAY 3 - selectExpr()

```
df_tmp.selectExpr('*', 'case when BMI < 18.5 then "過輕" when BMI < 24 then "正常" else "過重" end as BMI_L').show()
```



2

根據上面的 BMI 欄位，新增欄位 BMI_L 顯示 BMI 等級，規則如下。

過輕：BMI<18.5

正常：18.5<=BMI<24

過重：24<=BMI

DataFrame

```
df_tmp = df.select('APC_ID_SAS', 'HEIGHT', 'WEIGHT').withColumn('BMI', df.WEIGHT / (df.HEIGHT / 100) ** 2)
```

- WAY 4 - select() with expr function

```
from pyspark.sql.functions import expr
df_tmp.select('*', expr('case when BMI < 18.5 then "過輕" when BMI < 24 then "正常" else "過重" end as BMI_L')).show()
```

- WAY 5 - withColumn() with expr function

```
from pyspark.sql.functions import expr
df_tmp.withColumn("BMI_L", expr('case when BMI < 18.5 then "過輕" when BMI < 24 then "正常" else "過重" end as BMI_L')).show()
```



請善用大絕招，把資料表轉回 Pandas Dataframe，再進行操作，但同時就吃不到 Hadoop 叢集資源，建議至少做完 Spark SQL 後再轉。

- 使用 `toPandas()` 就可以快速轉回熟悉的 dataframe 型態

```
import pandas as pd
df_pd = df.toPandas()
df_pd.head()
```

	APC_ID_SAS	GENDER	AGE	CONTACT_CITY_CD	EDUCATION_CD	MARRIAGE_CD	STAFF_IND	LIFE_INSD_CNT	HEIGHT	WEIGHT
0	Q2D129E954AD477523	F	50.0	台中市	2.0	1.0	0	4	158	54
1	A2AFDA91C172B15A77	F	35.0	高雄市	3.0	0.0	0	2	162	48
2	F1D4EF7FFBE44A51FE	M	33.0	新北市	3.0	0.0	0	2	172	70
3	T117FCDED3FAE1C1F3	M	31.0	屏東縣	3.0	0.0	0	3	167	58
4	L26284A55F56B398BA	F	55.0	高雄市	3.0	1.0	0	2	161	55

- Pandas Dataframe 型態當然也可以轉回 Dpark Dataframe 型態

```
df_sd = spark.createDataFrame(df_pd)
df_sd.show(5)
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|      APC_ID_SAS|GENDER|  AGE|CONTACT_CITY_CD|EDUCATION_CD|MARRIAGE_CD|STAFF_IND|LIFE_INSD_CNT|HEIGHT|WEIGHT|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|Q2D129E954AD477523|    F|50.0|      台中市|      2.0|      1.0|      0|         4|   158|   54|
|A2AFDA91C172B15A77|    F|35.0|      高雄市|      3.0|      0.0|      0|         2|   162|   48|
|F1D4EF7FFBE44A51FE|    M|33.0|      新北市|      3.0|      0.0|      0|         2|   172|   70|
|T117FCDED3FAE1C1F3|    M|31.0|      屏東縣|      3.0|      0.0|      0|         3|   167|   58|
|L26284A55F56B398BA|    F|55.0|      高雄市|      3.0|      1.0|      0|         2|   161|   55|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows
```

Spark Dataframe 資料表訊息

Spark Dataframe 含有 schema 訊息，所以可以查看每個欄位的基本訊息，但沒有shape()可用。

```
# 檢視基本訊息
```

```
df.printSchema()
```

```
root
|-- APC_ID_SAS: string (nullable = true)
|-- GENDER: string (nullable = true)
|-- AGE: integer (nullable = true)
|-- CONTACT_CITY_CD: string (nullable = true)
|-- EDUCATION_CD: integer (nullable = true)
|-- MARRIAGE_CD: integer (nullable = true)
|-- STAFF_IND: integer (nullable = true)
|-- LIFE_INSD_CNT: integer (nullable = true)
|-- HEIGHT: integer (nullable = true)
|-- WEIGHT: integer (nullable = true)
```

```
# 欄位清單
```

```
df.columns
```

```
['APC_ID_SAS',
 'GENDER',
 'AGE',
 'CONTACT_CITY_CD',
 'EDUCATION_CD',
 'MARRIAGE_CD',
 'STAFF_IND',
 'LIFE_INSD_CNT',
 'HEIGHT',
 'WEIGHT']
```

```
# 查看資料維度
```

```
print(df.count(), len(df.columns))
```

```
100000 10
```

```
# 各欄位基本訊息
```

```
df.describe().show()
```

summary	APC_ID_SAS	GENDER	AGE	CONTACT_CITY_CD	EDUCATION_CD	MARRIAGE_CD	STAFF_IND	LIFE_INSD_CNT	HEIGHT	WEIGHT
count	100000	99571	99971	99885	80862	99086	100000	100000	100000	100000
mean	null	null	37.788008522471515	null	2.2000692537904083	0.4749409603778536	0.06458	3.29753	153.37791	54.49487
stddev	null	null	18.661418949622714	null	0.898252300077991	0.5368330466579324	0.35052910336592774	3.032440907269028	29.38724910800615	19.80406109088385
min	057DEB2BD27B30E75C	F	0	南投縣	1	0	0	1	5	0
max	Z2F1778CD4D4E88C29	M	120	高雄市	4	2	3	82	201	175



Spark Dataframe 新增數字欄位

因為有 schema 訊息，所以要注意新增欄位的資料型態。

```
from pyspark.sql import functions as F
# 加一欄都是 0 的 LABEL
df.withColumn("LABEL", F.lit(0)).show()
df.withColumn("LABEL", F.lit(0)).printSchema()
```

APC_ID_SAS	GENDER	AGE	CONTACT_CITY_CD	EDUCATION_CD	MARRIAGE_CD	STAFF_IND	LIFE_INSD_CNT	HEIGHT	WEIGHT	LABEL
Q2D129E954AD477523	F	50	台中市	2	1	0	4	158	54	0
A2AFDA91C172B15A77	F	35	高雄市	3	0	0	2	162	48	0
F1D4EF7FFBE44A51FE	M	33	新北市	3	0	0	2	172	70	0
T117FCDED3FAE1C1F3	M	31	屏東縣	3	0	0	3	167	58	0
L26284A55F56B398BA	F	55	高雄市	3	1	0	2	161	55	0
E1F5DBF55F6B1427DD	M	60	高雄市	1	0	0	1	167	72	0
F20481473EEB92E8D8	F	37	新北市	3	0	0	3	160	54	0
H20BE9CA7E806E0551	F	71	基隆市	3	1	0	2	150	47	0
F1B81805D82921E3B9	M	27	新北市	null	null	0	1	182	64	0
N1ED4D2C4D60B78323	M	57	彰化縣	2	1	0	8	182	76	0
F12A8F9FE08045F1A6	M	42	新北市	3	1	0	6	174	63	0
S2FBC9D681C459E262	F	46	高雄市	1	1	0	2	160	53	0
T24571F44004FE8145	F	59	屏東縣	4	1	0	2	163	58	0
P2A8A6EE60E936FA24	F	53	雲林縣	2	1	0	1	153	57	0
J1E82E68A12B9D234D	M	54	高雄市	3	1	0	2	180	80	0
J1882AEE9E2183B202	M	56	新竹縣	null	1	0	1	173	68	0
Q221C6DF46F85DA0D3	F	59	嘉義縣	1	1	0	5	157	58	0
B29C7859D7C9244C3A	F	45	新北市	null	1	0	2	160	52	0
A2663738EDFF7839CA	F	37	台北市	2	1	0	1	166	65	0
R24A96ABFCF4742B5C	F	32	台南市	3	0	0	3	159	62	0

only showing top 20 rows

```
root
-- APC_ID_SAS: string (nullable = true)
-- GENDER: string (nullable = true)
-- AGE: integer (nullable = true)
-- CONTACT_CITY_CD: string (nullable = true)
-- EDUCATION_CD: integer (nullable = true)
-- MARRIAGE_CD: integer (nullable = true)
-- STAFF_IND: integer (nullable = true)
-- LIFE_INSD_CNT: integer (nullable = true)
-- HEIGHT: integer (nullable = true)
-- WEIGHT: integer (nullable = true)
-- LABEL: integer (nullable = false)
```

```
from pyspark.sql import functions as F
# 加一欄都是 null 的 LABEL
df.withColumn("LABEL", F.lit(None)).show()
df.withColumn("LABEL", F.lit(None)).printSchema()
```

APC_ID_SAS	GENDER	AGE	CONTACT_CITY_CD	EDUCATION_CD	MARRIAGE_CD	STAFF_IND	LIFE_INSD_CNT	HEIGHT	WEIGHT	LABEL
Q2D129E954AD477523	F	50	台中市	2	1	0	4	158	54	null
A2AFDA91C172B15A77	F	35	高雄市	3	0	0	2	162	48	null
F1D4EF7FFBE44A51FE	M	33	新北市	3	0	0	2	172	70	null
T117FCDED3FAE1C1F3	M	31	屏東縣	3	0	0	3	167	58	null
L26284A55F56B398BA	F	55	高雄市	3	1	0	2	161	55	null
E1F5DBF55F6B1427DD	M	60	高雄市	1	0	0	1	167	72	null
F20481473EEB92E8D8	F	37	新北市	3	0	0	3	160	54	null
H20BE9CA7E806E0551	F	71	基隆市	3	1	0	2	150	47	null
F1B81805D82921E3B9	M	27	新北市	null	null	0	1	182	64	null
N1ED4D2C4D60B78323	M	57	彰化縣	2	1	0	8	182	76	null
F12A8F9FE08045F1A6	M	42	新北市	3	1	0	6	174	63	null
S2FBC9D681C459E262	F	46	高雄市	1	1	0	2	160	53	null
T24571F44004FE8145	F	59	屏東縣	4	1	0	2	163	58	null
P2A8A6EE60E936FA24	F	53	雲林縣	2	1	0	1	153	57	null
J1E82E68A12B9D234D	M	54	高雄市	3	1	0	2	180	80	null
J1882AEE9E2183B202	M	56	新竹縣	null	1	0	1	173	68	null
Q221C6DF46F85DA0D3	F	59	嘉義縣	1	1	0	5	157	58	null
B29C7859D7C9244C3A	F	45	新北市	null	1	0	2	160	52	null
A2663738EDFF7839CA	F	37	台北市	2	1	0	1	166	65	null
R24A96ABFCF4742B5C	F	32	台南市	3	0	0	3	159	62	null

only showing top 20 rows

```
root
-- APC_ID_SAS: string (nullable = true)
-- GENDER: string (nullable = true)
-- AGE: integer (nullable = true)
-- CONTACT_CITY_CD: string (nullable = true)
-- EDUCATION_CD: integer (nullable = true)
-- MARRIAGE_CD: integer (nullable = true)
-- STAFF_IND: integer (nullable = true)
-- LIFE_INSD_CNT: integer (nullable = true)
-- HEIGHT: integer (nullable = true)
-- WEIGHT: integer (nullable = true)
-- LABEL: null (nullable = true)
```



Spark Dataframe 定義欄位型態

因為有 schema 訊息，所以要注意新增欄位的資料型態，可以用 `cast()` 來修改。

指定新欄位String型態

```
from pyspark.sql.types import StringType
df.withColumn("LABEL", F.lit(0).cast(StringType())).printSchema()
```

```
root
|-- APC_ID_SAS: string (nullable = true)
|-- GENDER: string (nullable = true)
|-- AGE: integer (nullable = true)
|-- CONTACT_CITY_CD: string (nullable = true)
|-- EDUCATION_CD: integer (nullable = true)
|-- MARRIAGE_CD: integer (nullable = true)
|-- STAFF_IND: integer (nullable = true)
|-- LIFE_INSD_CNT: integer (nullable = true)
|-- HEIGHT: integer (nullable = true)
|-- WEIGHT: integer (nullable = true)
-- LABEL: string (nullable = false)
```

改變現有欄位型態

```
df.withColumn("AGE", df.AGE.cast(StringType())).printSchema()
```

```
root
|-- APC_ID_SAS: string (nullable = true)
|-- GENDER: string (nullable = true)
|-- AGE: string (nullable = true)
|-- CONTACT_CITY_CD: string (nullable = true)
|-- EDUCATION_CD: integer (nullable = true)
|-- MARRIAGE_CD: integer (nullable = true)
|-- STAFF_IND: integer (nullable = true)
|-- LIFE_INSD_CNT: integer (nullable = true)
|-- HEIGHT: integer (nullable = true)
|-- WEIGHT: integer (nullable = true)
```

欄位型態列表：

- BinaryType: binary
- BooleanType: boolean
- ByteType: tinyint
- DateType: date
- DecimalType: decimal(10,0)
- DoubleType: double
- FloatType: float
- IntegerType: int
- LongType: bigint
- ShortType: smallint
- StringType: string
- TimestampType: timestamp



Spark Dataframe 資料欄位合併

欄位合併的方法很多，簡單一點可以用 Spark SQL 進行合併，或使用 withColumn() 和 concat() 進行合併。

```
from pyspark.sql import functions as F
df_tmp = df[['AGE']]
df.withColumn("NEW_AGE", F.concat(df_tmp.AGE)).show()
```

APC_ID_SAS	GENDER	AGE	CONTACT_CITY_CD	EDUCATION_CD	MARRIAGE_CD	STAFF_IND	LIFE_INSD_CNT	HEIGHT	WEIGHT	NEW_AGE
Q2D129E954AD477523	F	50	台中市	2	1	0	4	158	54	50
A2AFDA91C172B15A77	F	35	高雄市	3	0	0	2	162	48	35
F1D4EF7FFBE44A51FE	M	33	新北市	3	0	0	2	172	70	33
T117FCDED3FAE1C1F3	M	31	屏東縣	3	0	0	3	167	58	31
L26284A55F56B398BA	F	55	高雄市	3	1	0	2	161	55	55
E1F5DBF55F6B1427DD	M	60	高雄市	1	0	0	1	167	72	60
F20481473EEB92E8D8	F	37	新北市	3	0	0	3	160	54	37
H20BE9CA7E806E0551	F	71	基隆市	3	1	0	2	150	47	71
F1B81805D82921E3B9	M	27	新北市	null	null	0	1	182	64	27
N1ED4D2C4D60B78323	M	57	彰化縣	2	1	0	8	182	76	57
F12A8F9FE08045F1A6	M	42	新北市	3	1	0	6	174	63	42
S2FBC9D681C459E262	F	46	高雄市	1	1	0	2	160	53	46
T24571F44004FE8145	F	59	屏東縣	4	1	0	2	163	58	59
P2A8A6EE60E936FA24	F	53	雲林縣	2	1	0	1	153	57	53
J1E82E68A12B9D234D	M	54	高雄市	3	1	0	2	180	80	54
J1882AEE9E2183B202	M	56	新竹縣	null	1	0	1	173	68	56
Q221C6DF46F85DA0D3	F	59	嘉義縣	1	1	0	5	157	58	59
B29C7859D7C9244C3A	F	45	新北市	null	1	0	2	160	52	45
A2663738EDFF7839CA	F	37	台北市	2	1	0	1	166	65	37
R24A96ABFCF4742B5C	F	32	台南市	3	0	0	3	159	62	32

only showing top 20 rows



Spark Dataframe 資料欄位移除

移除欄位一樣可以使用 Spark SQL 操作，或是使用 drop() 即可。

```
df.drop(df.AGE).show()
```

APC_ID_SAS	GENDER	CONTACT_CITY_CD	EDUCATION_CD	MARRIAGE_CD	STAFF_IND	LIFE_INSD_CNT	HEIGHT	WEIGHT
Q2D129E954AD477523	F	台中市	2	1	0	4	158	54
A2AFDA91C172B15A77	F	高雄市	3	0	0	2	162	48
F1D4EF7FFBE44A51FE	M	新北市	3	0	0	2	172	70
T117FCDED3FAE1C1F3	M	屏東縣	3	0	0	3	167	58
L26284A55F56B398BA	F	高雄市	3	1	0	2	161	55
E1F5DBF55F6B1427DD	M	高雄市	1	0	0	1	167	72
F20481473EEB92E8D8	F	新北市	3	0	0	3	160	54
H20BE9CA7E806E0551	F	基隆市	3	1	0	2	150	47
F1B81805D82921E3B9	M	新北市	null	null	0	1	182	64
N1ED4D2C4D60B78323	M	彰化縣	2	1	0	8	182	76
F12A8F9FE08045F1A6	M	新北市	3	1	0	6	174	63
S2FBC9D681C459E262	F	高雄市	1	1	0	2	160	53
T24571F44004FE8145	F	屏東縣	4	1	0	2	163	58
P2A8A6EE60E936FA24	F	雲林縣	2	1	0	1	153	57
J1E82E68A12B9D234D	M	高雄市	3	1	0	2	180	80
J1882AEE9E2183B202	M	新竹縣	null	1	0	1	173	68
Q221C6DF46F85DA0D3	F	嘉義縣	1	1	0	5	157	58
B29C7859D7C9244C3A	F	新北市	null	1	0	2	160	52
A2663738EDFF7839CA	F	台北市	2	1	0	1	166	65
R24A96ABFCF4742B5C	F	台南市	3	0	0	3	159	62

only showing top 20 rows



Spark Dataframe 合併資料列

資料列合併使用 `union()` 即可，可以看到下面筆數變兩倍。

```
df_tmp = df.union(df)
print(df_tmp.count())
df_tmp.show()
```

200000

APC_ID_SAS	GENDER	AGE	CONTACT_CITY_CD	EDUCATION_CD	MARRIAGE_CD	STAFF_IND	LIFE_INSD_CNT	HEIGHT	WEIGHT
Q2D129E954AD477523	F	50	台中市	2	1	0	4	158	54
A2AFDA91C172B15A77	F	35	高雄市	3	0	0	2	162	48
F1D4EF7FFBE44A51FE	M	33	新北市	3	0	0	2	172	70
T117FCDED3FAE1C1F3	M	31	屏東縣	3	0	0	3	167	58
L26284A55F56B398BA	F	55	高雄市	3	1	0	2	161	55
E1F5D8F55F6B1427DD	M	60	高雄市	1	0	0	1	167	72
F20481473EEB92E8D8	F	37	新北市	3	0	0	3	160	54
H20BE9CA7E806E0551	F	71	基隆市	3	1	0	2	150	47
F1B81805D82921E3B9	M	27	新北市	null	null	0	1	182	64
N1ED4D2C4D60B78323	M	57	彰化縣	2	1	0	8	182	76
F12A8F9FE08045F1A6	M	42	新北市	3	1	0	6	174	63
S2FBC9D681C459E262	F	46	高雄市	1	1	0	2	160	53
T24571F44004FE8145	F	59	屏東縣	4	1	0	2	163	58
P2A8A6EE60E936FA24	F	53	雲林縣	2	1	0	1	153	57
J1E82E68A12B9D234D	M	54	高雄市	3	1	0	2	180	80
J1882AEE9E2183B202	M	56	新竹縣	null	1	0	1	173	68
Q221C6DF46F85DA0D3	F	59	嘉義縣	1	1	0	5	157	58
B29C7859D7C9244C3A	F	45	新北市	null	1	0	2	160	52
A2663738EDFF7839CA	F	37	台北市	2	1	0	1	166	65
R24A96ABFCF4742B5C	F	32	台南市	3	0	0	3	159	62

only showing top 20 rows



Spark Dataframe 資料串聯

為了學習資料串聯，首先讀取另一張資料表，台灣各縣市衛生所資料 health_center.csv。

```
# 從外部抓衛生所的測試資料
from pyspark import SparkFiles
url = 'https://raw.githubusercontent.com/chia313339/Spark_practice/master/health_center.csv'
spark.sparkContext.addFile(url)
df2 = spark.read.csv(SparkFiles.get("health_center.csv"), header=True, inferSchema=True)
```

```
# 將資料集讀取在memory上，DataFrame跟SparkSQL都要下此語法，如果沒下，資料每次都會重新load
df2.cache()
```

```
# 執行這句才會真的運行Spark，將資料存在記憶體
df2.show()
```

COL_NAME	欄位名稱
CITY_CD	縣市
HC_CNT	衛生所數量
HCP_CNT	衛生所員工數

```
+-----+-----+-----+
|CITY_CD|HC_CNT|HCP_CNT|
+-----+-----+-----+
| 新北市|    29|    430|
| 台北市|    12|    303|
| 桃園市|    13|    257|
| 台中市|    30|    332|
| 台南市|    37|    342|
| 高雄市|    38|    474|
| 宜蘭縣|    12|    139|
| 新竹縣|    13|    169|
| 苗栗縣|    18|    184|
| 彰化縣|    27|    212|
| 南投縣|    13|    181|
| 雲林縣|    20|    219|
| 嘉義縣|    18|    209|
| 屏東縣|    33|    370|
| 台中縣|    16|    209|
| 花蓮縣|    13|    161|
| 澎湖縣|    11|     87|
| 基隆市|     7|     62|
| 新竹市|     3|     31|
| 嘉義市|     2|     23|
+-----+-----+-----+
only showing top 20 rows
```



Spark Dataframe 資料串聯

資料表的串聯使用 `join()` 函數即可，並於 `how` 參數內寫下 `join` 方式，當然也可以使用 Spark SQL 會更簡單一些。

```
df.join(df2, df.CONTACT_CITY_CD == df2.CITY_CD, how = "left_outer").show()
```

APC_ID_SAS	GENDER	AGE	CONTACT_CITY_CD	EDUCATION_CD	MARRIAGE_CD	STAFF_IND	LIFE_INSD_CNT	HEIGHT	WEIGHT	CITY_CD	HC_CNT	HCP_CNT
Q2D129E954AD477523	F	50	台中市	2	1	0	4	158	54	台中市	30	332
A2AFDA91C172B15A77	F	35	高雄市	3	0	0	2	162	48	高雄市	38	474
F1D4EF7FFBE44A51FE	M	33	新北市	3	0	0	2	172	70	新北市	29	430
T117FCDED3FAE1C1F3	M	31	屏東縣	3	0	0	3	167	58	屏東縣	33	370
L26284A55F56B398BA	F	55	高雄市	3	1	0	2	161	55	高雄市	38	474
E1F5DBF55F6B1427DD	M	60	高雄市	1	0	0	1	167	72	高雄市	38	474
F20481473EEB92E8D8	F	37	新北市	3	0	0	3	160	54	新北市	29	430
H20BE9CA7E806E0551	F	71	基隆市	3	1	0	2	150	47	基隆市	7	62
F1B81805D82921E3B9	M	27	新北市	null	null	0	1	182	64	新北市	29	430
N1ED4D2C4D60B78323	M	57	彰化縣	2	1	0	8	182	76	彰化縣	27	212
F12A8F9FE08045F1A6	M	42	新北市	3	1	0	6	174	63	新北市	29	430
S2FBC9D681C459E262	F	46	高雄市	1	1	0	2	160	53	高雄市	38	474
T24571F44004FE8145	F	59	屏東縣	4	1	0	2	163	58	屏東縣	33	370
P2A8A6EE60E936FA24	F	53	雲林縣	2	1	0	1	153	57	雲林縣	20	219
J1E82E68A12B9D234D	M	54	高雄市	3	1	0	2	180	80	高雄市	38	474
J1882AEE9E2183B202	M	56	新竹縣	null	1	0	1	173	68	新竹縣	13	169
Q221C6DF46F85DA0D3	F	59	嘉義縣	1	1	0	5	157	58	嘉義縣	18	209
B29C7859D7C9244C3A	F	45	新北市	null	1	0	2	160	52	新北市	29	430
A2663738EDFF7839CA	F	37	台北市	2	1	0	1	166	65	台北市	12	303
R24A96ABFCF4742B5C	F	32	台南市	3	0	0	3	159	62	台南市	37	342

only showing top 20 rows

how –

default inner. Must be one of: inner, cross, outer, full, fullouter, full_outer, left, leftouter, left_outer, right, rightouter, right_outer, semi, leftsemi, left_semi, anti, leftanti and left_anti.



資料串聯練習

嘗試用 Dataframe 或 Spark SQL 解以下問題，方法不拘：

1

建立一張資料表，包含 CUST_360 所有欄位，並使用 CONTACT_CITY_CD 透過 left join 串聯 HEALTH_CENTER 衛生所資料的 CITY_CD 欄位。計算並顯示各縣市衛生所的平均員工數，命名為 HC_MEAN，最後結果顯示如下圖：

APC_ID_SAS	GENDER	AGE	CONTACT_CITY_CD	EDUCATION_CD	MARRIAGE_CD	STAFF_IND	LIFE_INSD_CNT	HEIGHT	WEIGHT	HC_MEAN
Q2D129E954AD477523	F	50	台中市	2	1	0	4	158	54	11.066666666666666
A2AFDA91C172B15A77	F	35	高雄市	3	0	0	2	162	48	12.473684210526315
F1D4EF7FFBE44A51FE	M	33	新北市	3	0	0	2	172	70	14.827586206896552
T117FCDED3FAE1C1F3	M	31	屏東縣	3	0	0	3	167	58	11.212121212121213
L26284A55F56B398BA	F	55	高雄市	3	1	0	2	161	55	12.473684210526315
E1F5DBF55F6B1427DD	M	60	高雄市	1	0	0	1	167	72	12.473684210526315
F20481473EEB92E8D8	F	37	新北市	3	0	0	3	160	54	14.827586206896552
H20BE9CA7E806E0551	F	71	基隆市	3	1	0	2	150	47	8.857142857142858
F1B81805D82921E3B9	M	27	新北市	null	null	0	1	182	64	14.827586206896552
N1ED4D2C4D60B78323	M	57	彰化縣	2	1	0	8	182	76	7.851851851851852
F12A8F9FE08045F1A6	M	42	新北市	3	1	0	6	174	63	14.827586206896552
S2FBC9D681C459E262	F	46	高雄市	1	1	0	2	160	53	12.473684210526315
T24571F44004FE8145	F	59	屏東縣	4	1	0	2	163	58	11.212121212121213
P2A8A6EE60E936FA24	F	53	雲林縣	2	1	0	1	153	57	10.95
J1E82E68A12B9D234D	M	54	高雄市	3	1	0	2	180	80	12.473684210526315
J1882AEE9E2183B202	M	56	新竹縣	null	1	0	1	173	68	13.0
Q221C6DF46F85DA0D3	F	59	嘉義縣	1	1	0	5	157	58	11.611111111111111
B29C7859D7C9244C3A	F	45	新北市	null	1	0	2	160	52	14.827586206896552
A2663738EDFF7839CA	F	37	台北市	2	1	0	1	166	65	25.25
R24A96ABFCF4742B5C	F	32	台南市	3	0	0	3	159	62	9.243243243243244

only showing top 20 rows



1

建立一張資料表，包含 CUST_360 所有欄位，並使用 CONTACT_CITY_CD 透過 left join 串聯 HEALTH_CENTER 衛生所資料的 CITY_CD 欄位。計算並顯示各縣市衛生所的平均員工數，命名為 HC_MEAN：

DataFrame

```
df.join(df2,df.CONTACT_CITY_CD == df2.CITY_CD,
how = "left_outer")
.withColumn("HC_MEAN",df2.HCP_CNT/df2.HC_CNT)
.drop(*df2.columns).show()
```

Spark SQL

```
df2.registerTempTable('health_center')
df2.cache()

spark.sql('select APC_ID_SAS, GENDER, AGE, CONTACT_CITY_CD,
EDUCATION_CD, MARRIAGE_CD,
STAFF_IND, LIFE_INSD_CNT, HEIGHT, WEIGHT,
HCP_CNT/HC_CNT as HC_MEAN
from cust_360 left join health_center on CONTACT_CITY_CD = CITY_CD').show()
```



寫回 Hive Table

HAP 的 Hive 資料庫中，有 life_user 的空間可以自行建立使用者 Table，可以透過 mode 選擇寫入方式，語法如下：

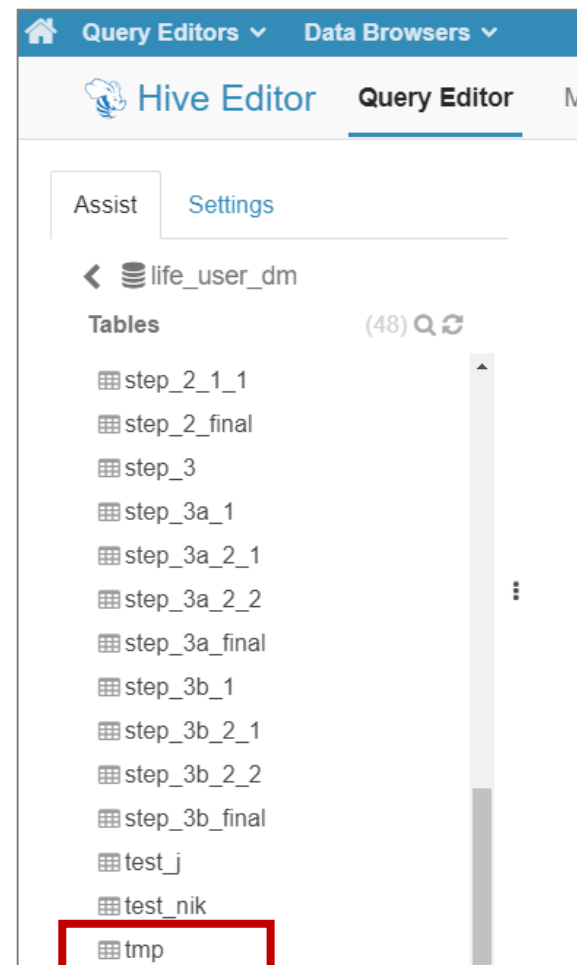
公司 HAP 環境

```
df.write.mode("append").saveAsTable("life_user_dm.tmp")
```

mode –

specifies the behavior of the save operation when data already exists.

- append: Append contents of this DataFrame to existing data.
- overwrite: Overwrite existing data.
- ignore: Silently ignore this operation if data already exists.
- error (default case): Throw an exception if data already exists.



Colab 寫回 Hive Table

Colab 雖然沒有 Hue 可以操作，但一樣可以進行 Table 的儲存，因為沒有 Database，直接帶資料表名稱即可：

Colab 環境

```
df.write.mode("append").saveAsTable("tmp")
```

```
# 儲存好的 Table 可以用 spark.sql() 直接呼叫  
spark.sql('select * from tmp').show()
```

APC_ID_SAS	GENDER	AGE	CONTACT_CITY_CD	EDUCATION_CD	MARRIAGE_CD	STAFF_IND	LIFE_INSD_CNT	HEIGHT	WEIGHT
Q2D129E954AD477523	F	50	台中市	2	1	0	4	158	54
A2AFDA91C172B15A77	F	35	高雄市	3	0	0	2	162	48
F1D4EF7FF8E44A51FE	M	33	新北市	3	0	0	2	172	70
T117FCDED3FAE1C1F3	M	31	屏東縣	3	0	0	3	167	58
L26284A55F56B398BA	F	55	高雄市	3	1	0	2	161	55
E1F5DBF55F6B1427DD	M	60	高雄市	1	0	0	1	167	72
F20481473EEB92E8D8	F	37	新北市	3	0	0	3	160	54
H20BE9CA7E806E0551	F	71	基隆市	3	1	0	2	150	47
F1B81805D82921E3B9	M	27	新北市	null	null	0	1	182	64
N1ED4D2C4D60B78323	M	57	彰化縣	2	1	0	8	182	76
F12A8F9FE08045F1A6	M	42	新北市	3	1	0	6	174	63
S2FBC9D681C459E262	F	46	高雄市	1	1	0	2	160	53
T24571F44004FE8145	F	59	屏東縣	4	1	0	2	163	58
P2A8A6EE60E936FA24	F	53	雲林縣	2	1	0	1	153	57
J1E82E68A12B9D234D	M	54	高雄市	3	1	0	2	180	80
J1882AEE9E2183B202	M	56	新竹縣	null	1	0	1	173	68
Q221C6DF46F85DA0D3	F	59	嘉義縣	1	1	0	5	157	58
B29C7859D7C9244C3A	F	45	新北市	null	1	0	2	160	52
A2663738EDFF7839CA	F	37	台北市	2	1	0	1	166	65
R24A96ABFCF4742B5C	F	32	台南市	3	0	0	3	159	62

only showing top 20 rows



進階 - 資料篩選最大痛點 Join

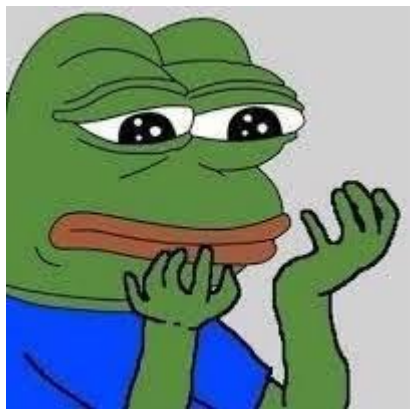
所以的資料篩選操作都很單純，唯獨 Join 是最複雜的，尤其數據團隊的資料動輒幾百萬筆互串，容易導致記憶體不足，甚至是跑到天荒地老的狀況發生。

Spark 提供三種不同的 join 方式，解決不同場景的困境：

Broadcast hash join

Shuffle hash join

Sort-merge join



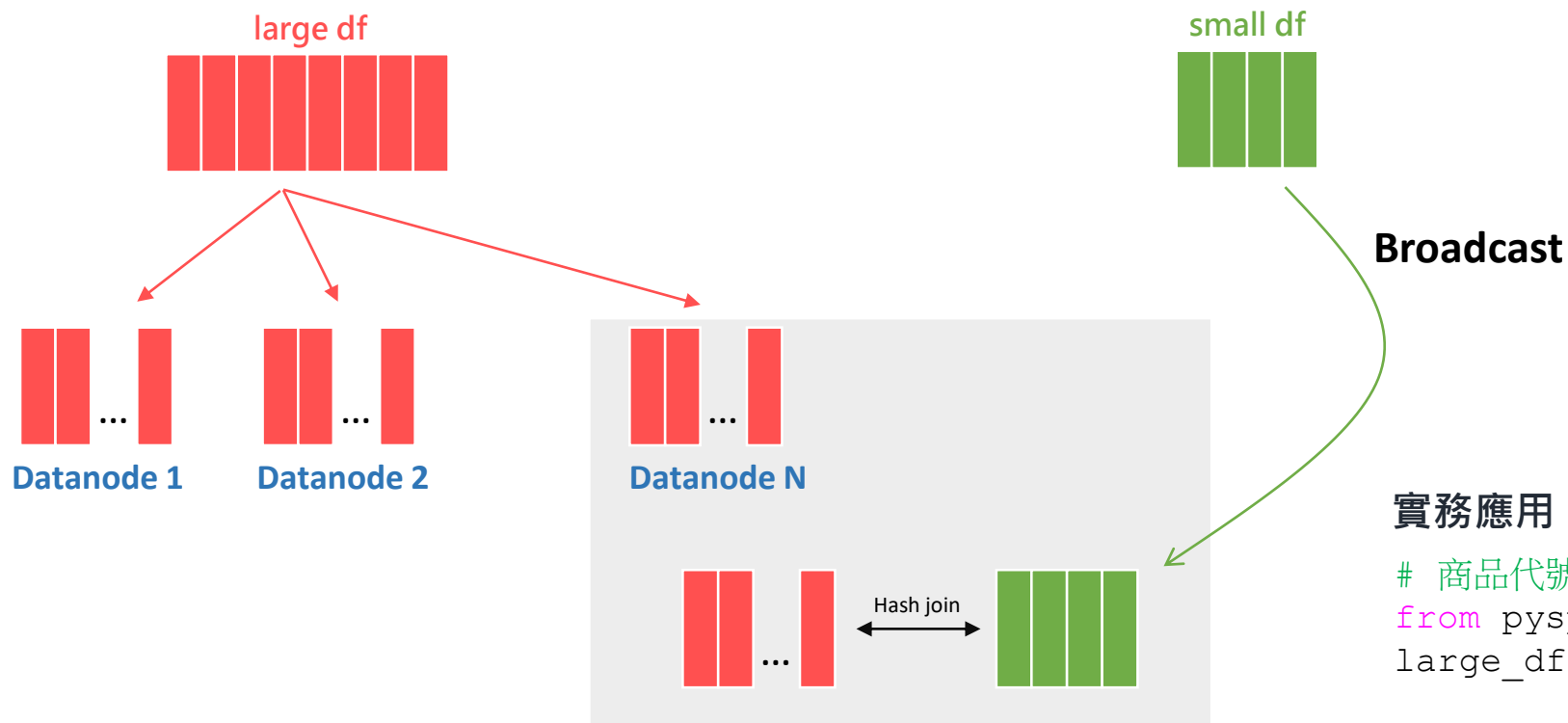
當你發現要拿客戶服務歷程串 DTAAB001 理賠紀錄檔...



進階 – Broadcast Hash Join

Broadcast hash join

Broadcast 的方法原理，是將其中一張小表 (通常非常小，小於10M)，廣播到叢集每張大表存在的節點，如此運算可以提高運算速度。(開發者學這個就好)



實務應用：

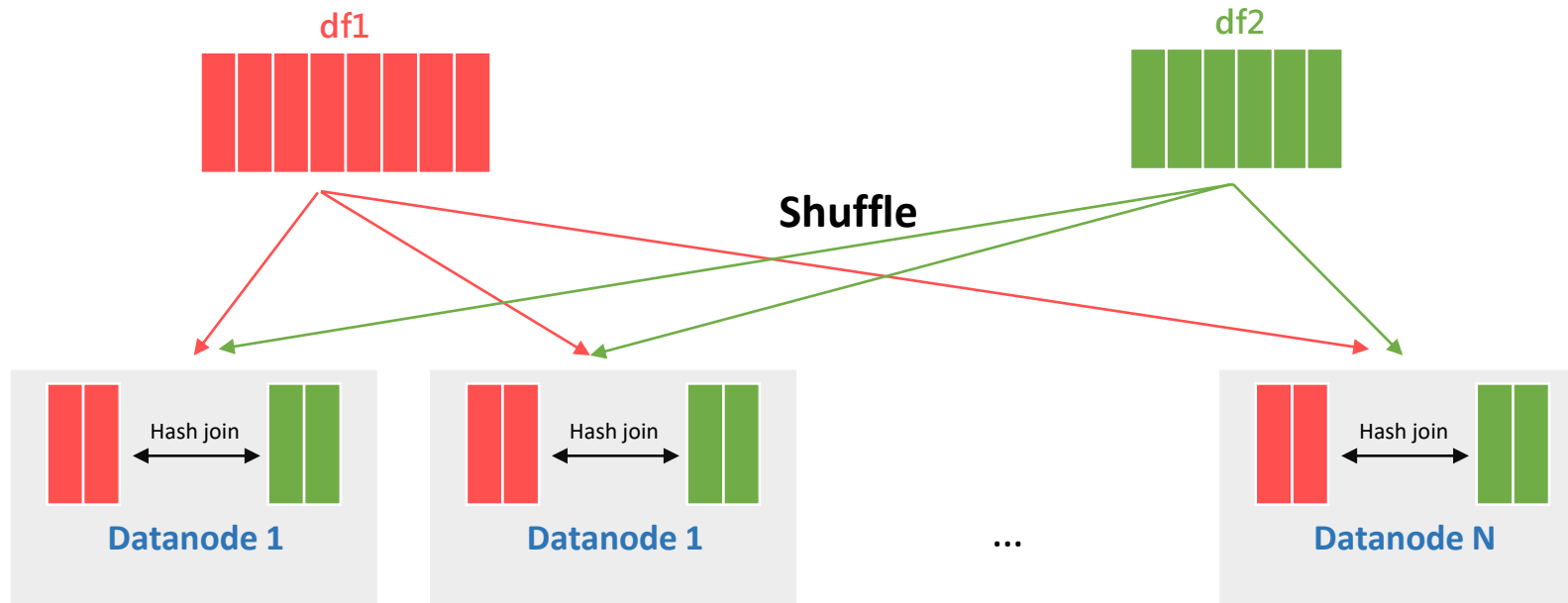
```
# 商品代號串商品分類
from pyspark.sql.functions import broadcast
large_df.join(broadcast(small_df))
```



進階 – Shuffle hash join

Shuffle hash join

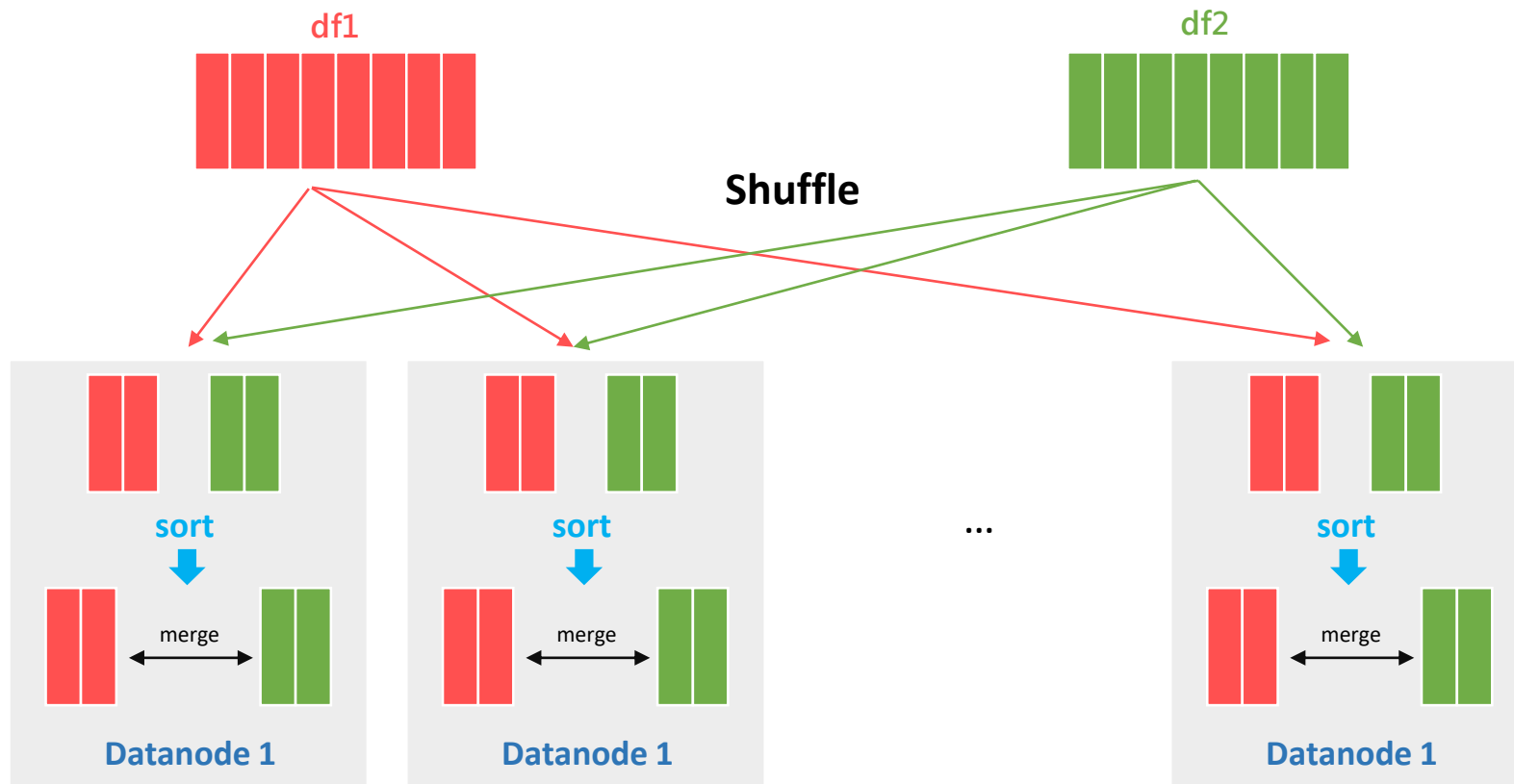
當將其中一張表沒這麼小時，可以用該方法處理，Shuffle 會根據 key 進行分區，相同 join key 會在同一個節點進行運算，也因為如此 IO 變多，資源 cost 更高。



進階 – Sort-Merge join

Sort-merge join

適合兩張大表串聯的場景，流程是將兩張大表根據 key 重新分區，遍佈到叢集之中後再進行 sort，各節點碰到相同 join key 就 merge，資源 cost 超大。



Spark 在資料處理的運算能力無庸置疑，但語法學習曲線較高，導致部分資料科學家會排斥學習，並資料工程師鴻溝越來越大。



```
import pandas as pd
df = read.csv('data.csv')
df.columns = ['x', 'y', 'z']
```

```
df['x2'] = df.x*df.x
```

1. 讀取 csv
2. 重新命名欄位
3. 新增欄位



```
df = (spark.read.schema(schema).options(header='true',
inferredschema='true').load('data.csv'))
df = df.toDF('x', 'y', 'z')
```

```
df = df.withColumn('x2', df.x*df.x)
```


Spark 未來發展

因此 Spark 的新套件也在積極發展中，Spark 3 開始有 Koalas Spark 可以使用，使用方法跟 Pandas 無異，但背後可以吃 Spark 資源，大幅降低學習曲線。



```
import pandas as pd
df = pd.read.csv('data.csv')
df.columns = ['x', 'y', 'z']

df['x2'] = df.x*df.x
```

1. 讀取 csv
2. 重新命名欄位
3. 新增欄位



```
import databricks.koalas as ks
df = ks.read.csv('data.csv')
df.columns = ['x', 'y', 'z']

df['x2'] = df.x*df.x
```





程式改了三週還出來



直接找工程科

END

