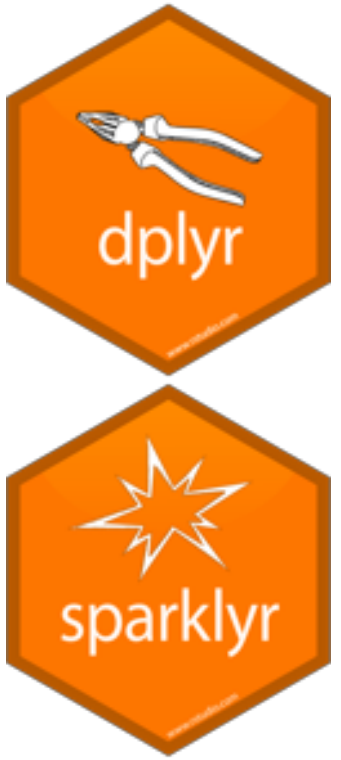
cloudera

sparklyr을 활용한 R 코드 분산 처리

임상배/Solutions Engineer
Cloudera



Apache Spark 개요(빠르게 대량의 데이터를 처리/분석)

- 인-메모리 기반의 고속의 범용 클러스터 컴퓨팅 엔진 (The leading candidate for “successor to MapReduce”)
- 맵리듀스(MapReduce) 모델을 대화형 명령어 쿼리나 스트리밍으로 처리 가능하도록 확장
- 성능 : 기존 하둡 맵리듀스에 비해 빠른 성능 제공
- 활용 범위 : 개별 분산 시스템에서 수행되는 일괄 처리, 반복 알고리즘, 대화형 쿼리, 스트리밍을 단일 시스템에서 지원
- 스칼라 언어로 구현되었으며 다양한 언어(파이선, 자바, 스칼라, R)를 지원

Apache Spark 개요(빠르게 대량의 데이터를 처리/분석)

- Cluster computing platform designed to be fast and general-purpose
  - Speed : RDD라는 방식을 통해 메모리를 사용을 극대화하여 interactive, streaming이 빠르게 처리
  - Generality : 분산처리, 배치처리, 대화형 작업, 스트리밍처리 등을 모두 같은 엔진에서 지원하여 쉬운 작업 연계

스파크 생태계

Apache Spark™ is a **unified analytics engine** for large-scale data processing.

R 사용자가 Spark을 알게 되면 생기는 좋은 일들…

- 하둡과 연계할 수 있음(요즘 많은 기업에서 이미 하둡에 데이터를 저장)
- 익숙한 SQL 문장으로 빅데이터를 다룰 수 있음(기존 SQL 지식 및 코딩 구현체 재활용)
- 내장형 머신 러닝 기능을 지원
- 성능, 성능, 성능
- But NFL(No Free Lunch)

분석 프로젝트에서 데이터를 다루는 방법

Data Engineering + Data Science

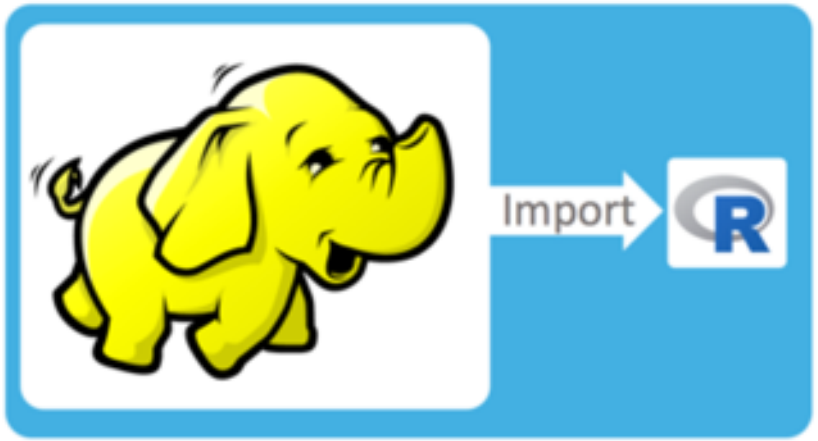
- 기존 분석 소프트웨어 기반 환경 고도화(대량의 데이터 활용, 기존 보유 기술 활용, 고비용-고객은 무엇을 원할까?)
- 소형화 방법(Big Data를 Small 데이터로 만들어 활용, 분석 알고리즘은 그대로)
- 데이터 병렬 처리 방법(데이터 처리 구간을 오픈 소스 기반 병렬 처리 엔진으로 대체, Big Data를 다수의 Medium/Small 데이터로 나누고, 분석 알고리즘은 그대로 혹은 약간 수정)
- 알고리즘 병렬 처리 방법(Big Data 기반 병렬 처리 알고리즘-e.q. Spark MLlib)

데이터 레이크(Data Lake)를 활용하여 데이터 분석하기

데이터 소스로 혹은 분석 엔진으로

Hadoop as a Data Source

Problem: Data is too large to download into memory



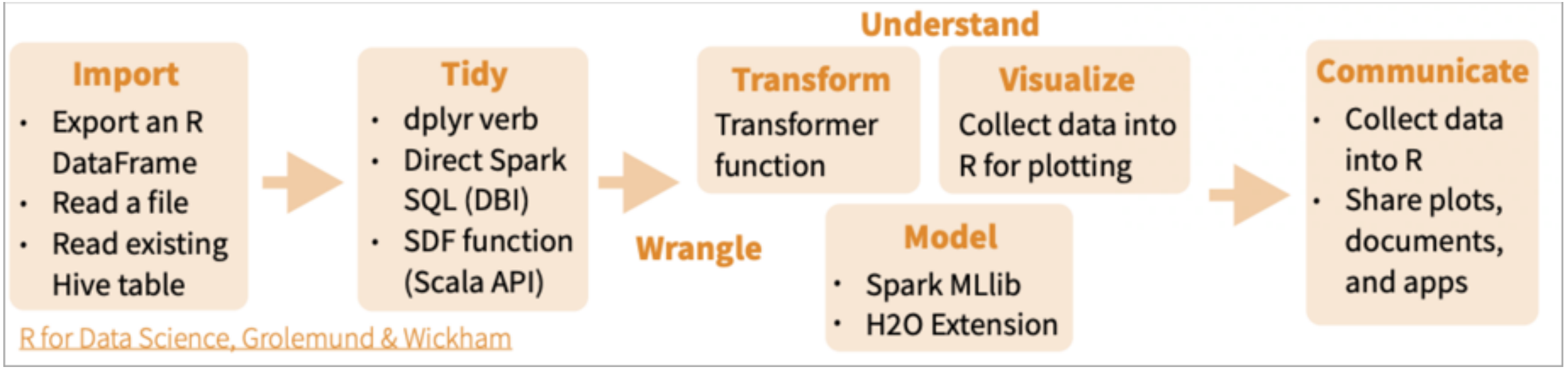
Workaround: Use a very small sample or download as much data as possible

Spark as an Analysis Engine

Solution: Use sparklyr to access & analyze the data inside Spark.
Only bring results into R.

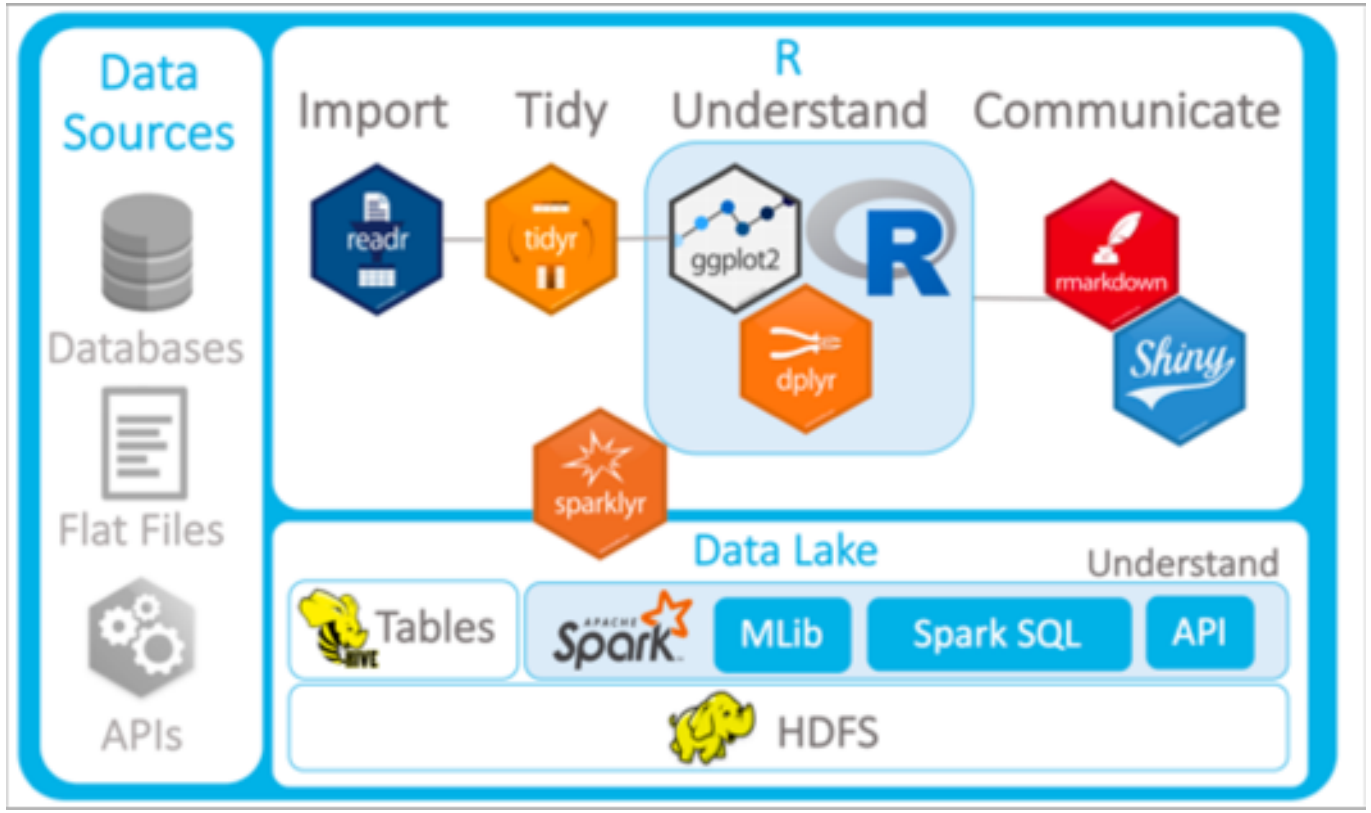
Source : https://spark.rstudio.com/guides/data-lakes/

Data Science Toolchain with Spark + sparklyr

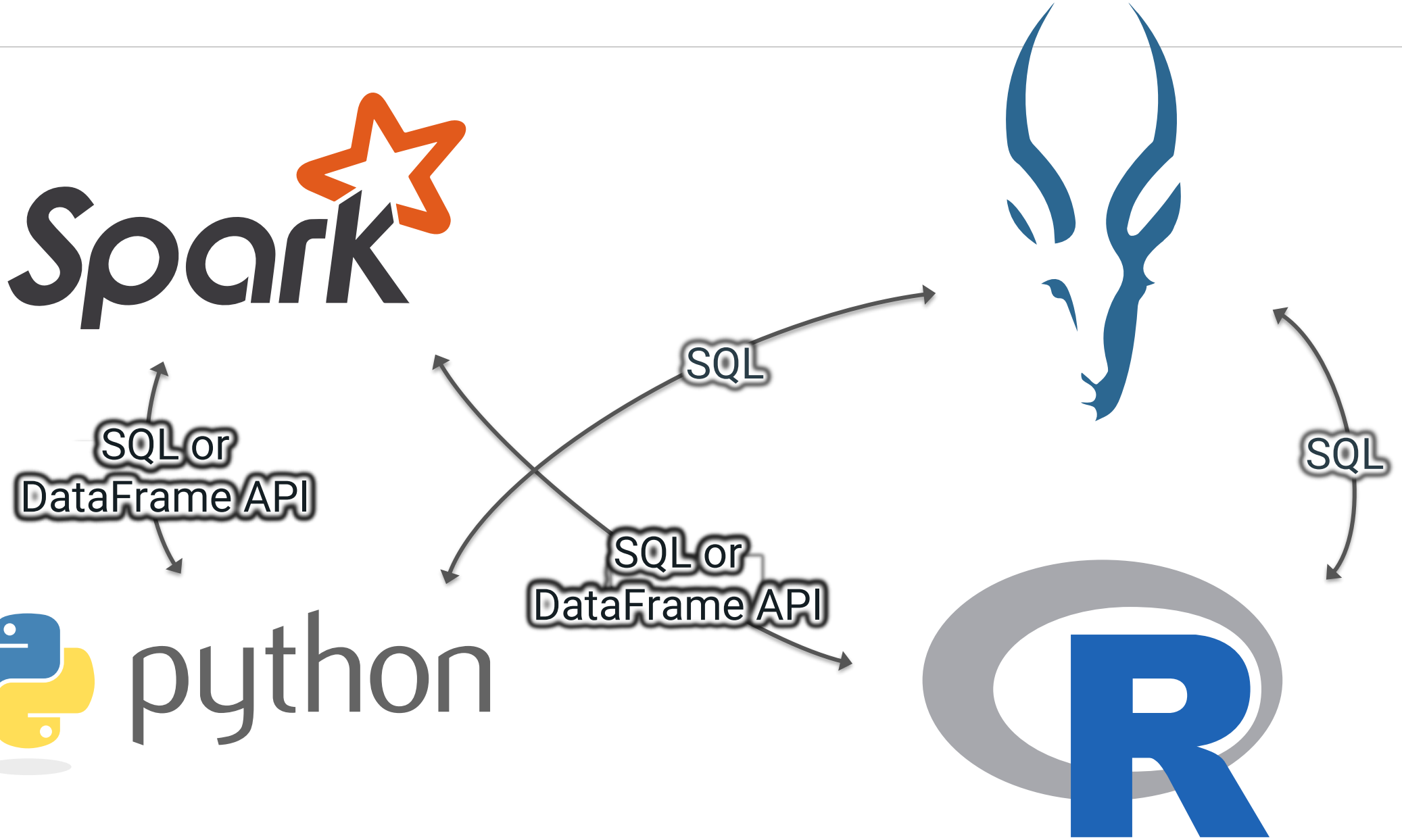


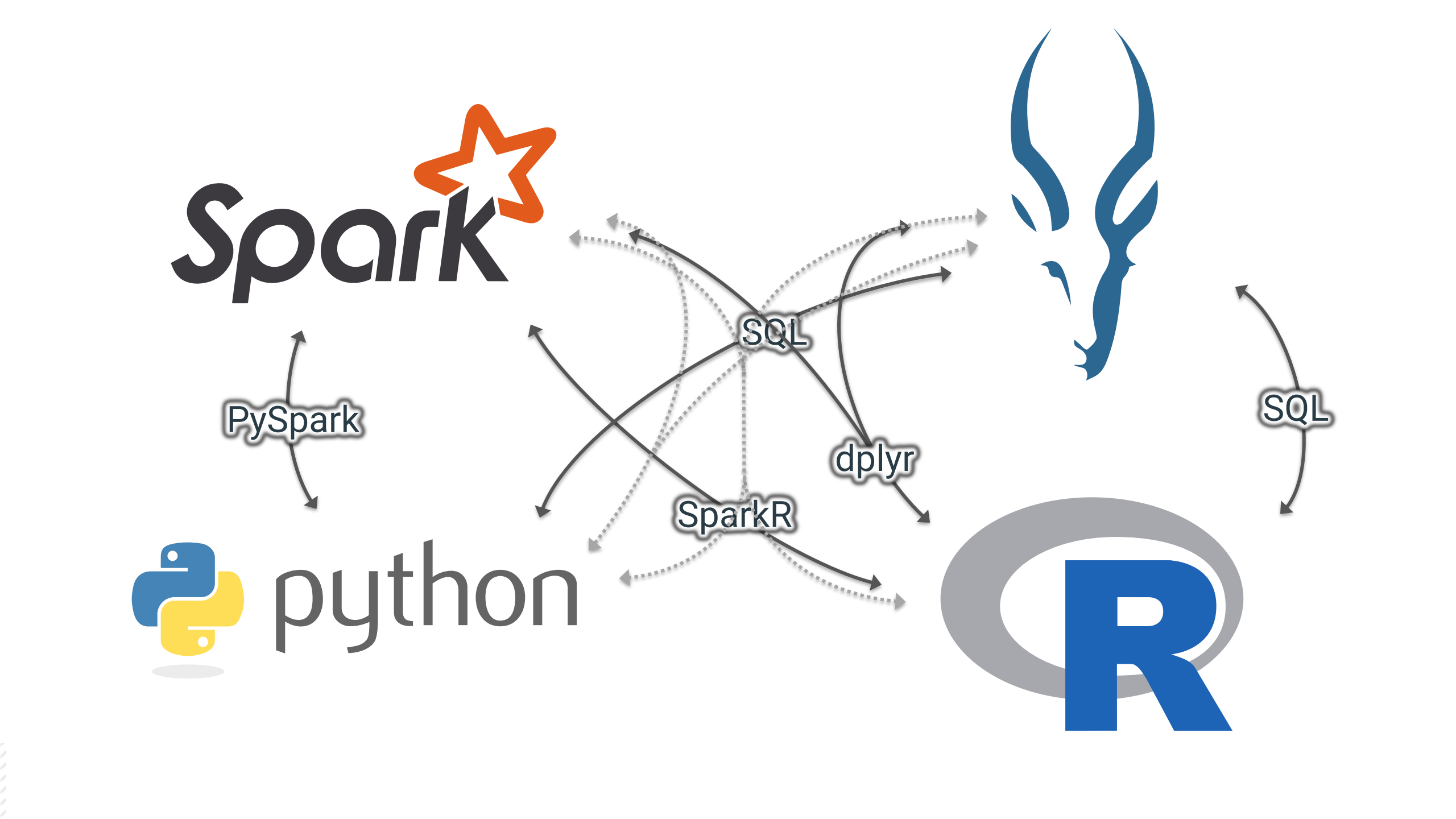
source : R for Data Science Grolemund & Wickham, https://github.com/rstudio/cheatsheets/raw/master/sparklyr.pdf

R과 Spark를 활용한 데이터 분석



Source : https://spark.rstudio.com/guides/data-lakes/




Spark
PySpark
python
SQL
dplyr
SparkR
SQL
R

dplyr

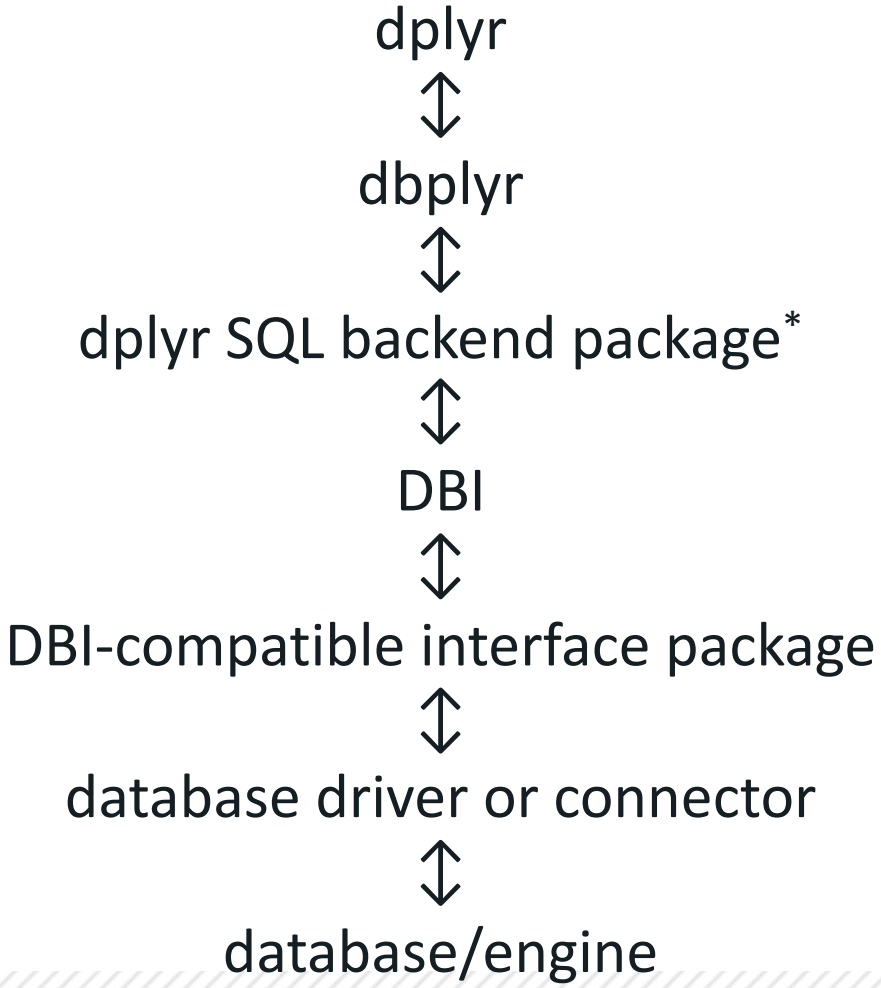
dplyr은 R에서 효율적으로 데이터 처리를 하는 도구를 제공

- select() to select columns
- filter() to filter rows
- arrange() to order rows
- mutate() to create new columns
- summarise() to aggregate
- group_by() to perform operations by group

dplyr는 로컬 데이터와 원격 데이터 소스에서 작동

- 원격 데이터 소스를 사용하면 dplyr은 해당 연산을 SQL로 변환

dplyr SQL backends

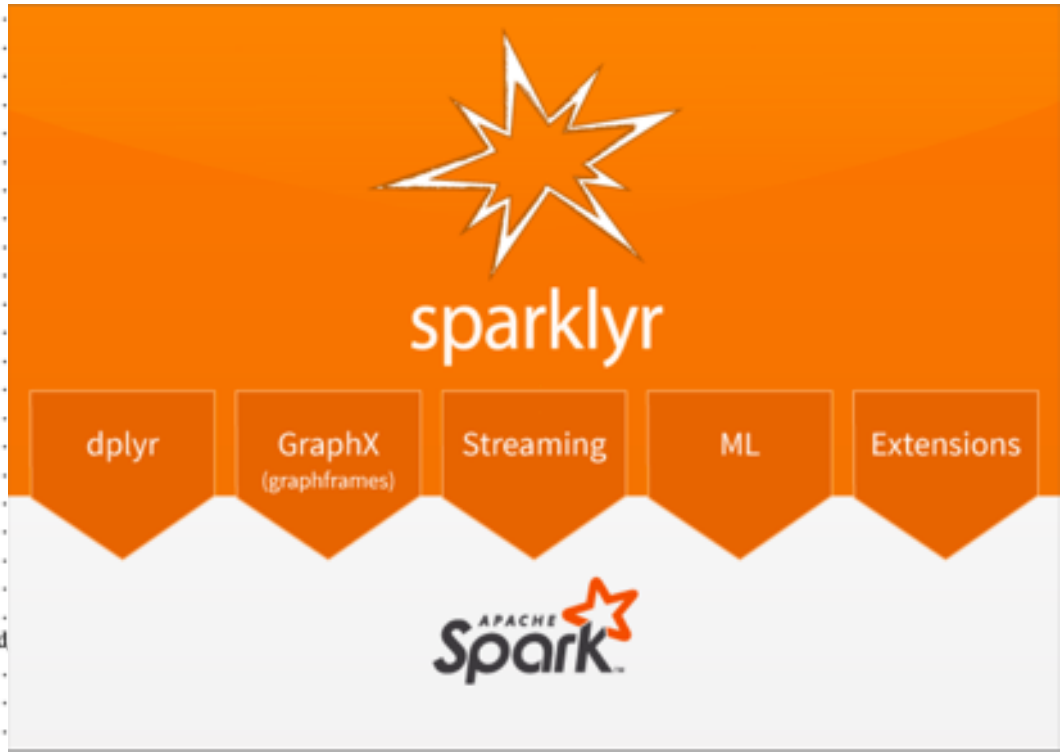


* optional

sparklyr 개요

- Apache Spark를 사용하기 위한 인터페이스를 제공하는 R 패키지
- dplyr에 Spark용 SQL backend를 제공
- Apache Spark의 MLlib API 제공
- Apache Spark의 DataFrames API 일부 제공(sdf_xxx)
- RStudio가 개발

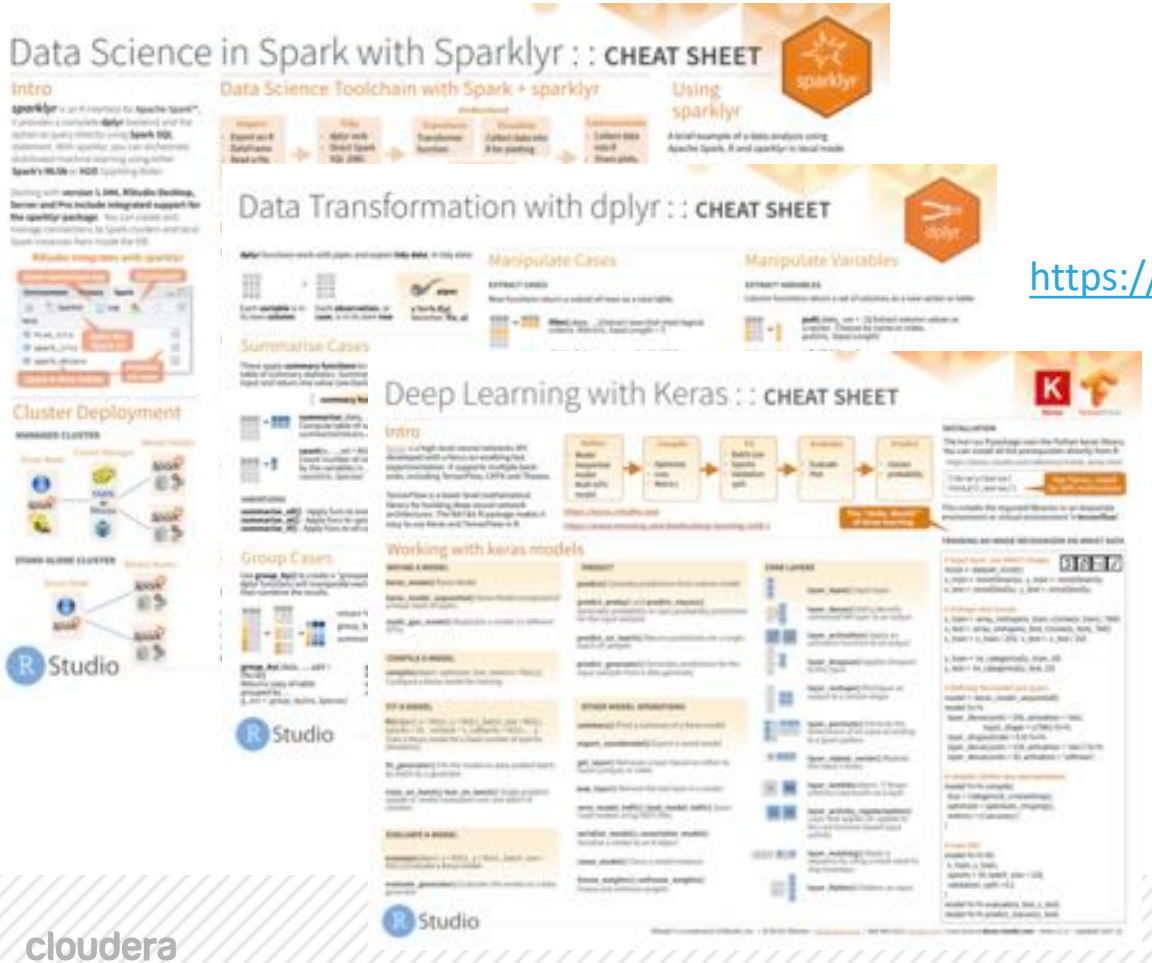
sdf-saveload
sdf-transform-methods
sdf_along
sdf_bind
sdf_broadcast
sdf_checkpoint
sdf_coalesce
sdf_collect
sdf_copy_to
sdf_debug_string
sdf_describe
sdf_dim
sdf_is_streaming
sdf_last_index
sdf_len
sdf_num_partitions
sdf_partition
sdf_persist
sdf_pivot
sdf_project
sdf_quantile
sdf_read_column
sdf_register
sdf_repartition
sdf_residuals.ml_model_generalized
sdf_sample
sdf_schema
sdf_separate_column
sdf_seq
sdf_sort
sdf_sql
sdf_with_sequential_id
sdf_with_unique_id



참고 자료

Data Science in Spark with Sparklyr :: CHEAT SHEET

https://www.rstudio.com/resources/cheatsheets/



https://github.com/rstudio/cheatsheets/raw/master/sparklyr.pdf

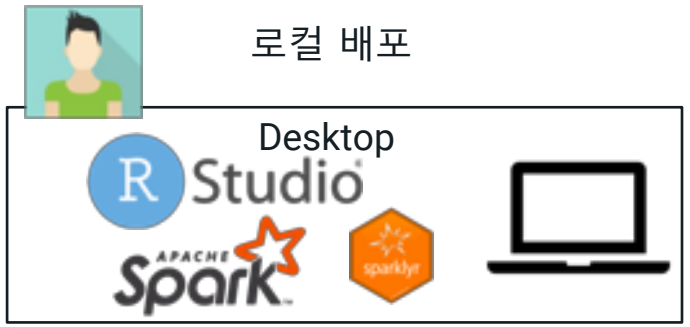
https://github.com/rstudio/cheatsheets/raw/master/data-transformation.pdf

https://github.com/rstudio/cheatsheets/raw/master/keras.pdf

sparklyr 배포 옵션

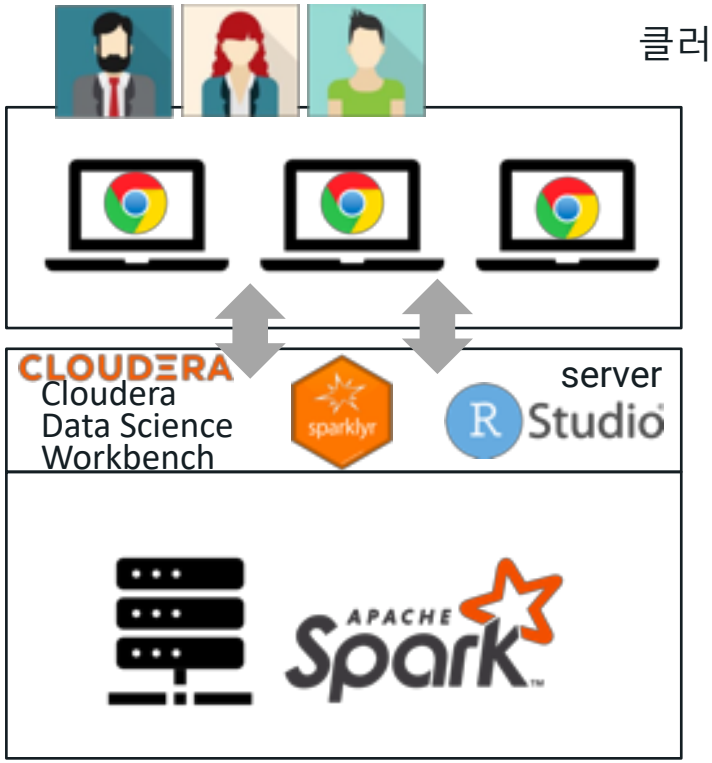
로컬 배포(RStudio Desktop), 클러스터 배포(RStudio Server or RStudio Desktop)

로컬 배포

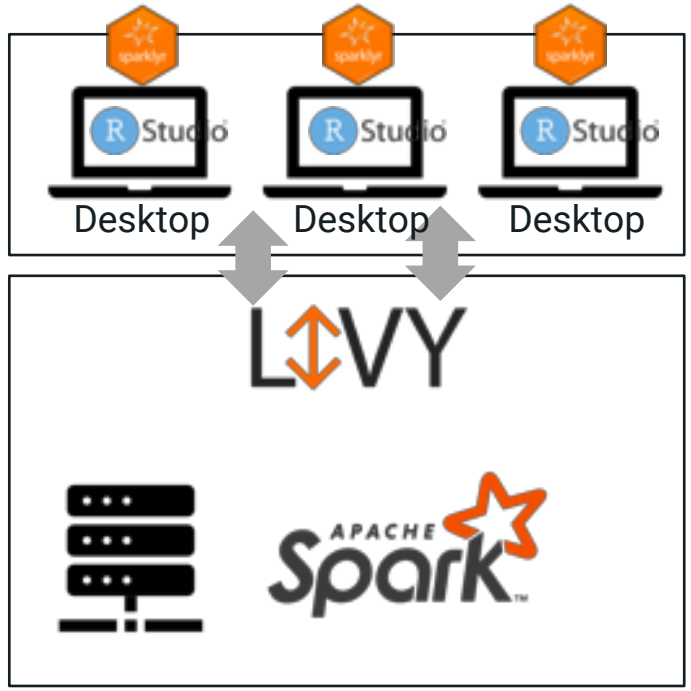


- 소규모 데이터를 대상
- 학습 및 단순 테스트 용도

클러스터 배포



- 일반적인 배포 방식
- 전용 분석 서버 도입/웹 GUI



- 실험적 방식
- 운영 시스템에 권장하지 않음

R코드 분산 수행(Distributing R Computations)

spark_apply()

- spark_apply()는 Spark 객체(일반적으로 Spark DataFrame)에 R 함수를 적용
- spark 객체를 파티션하여 클러스터 전체에 분포 시킴
- 기본 파티션 방식 혹은 group_by 인수를 사용하여 사용자 지정 파티셔닝

```
17  trees_tbl <- sdf_copy_to(sc, trees, repartition = 3,overwrite=TRUE)
18
19  trees_tbl %>%
20    spark_apply(function(e) head(e, 1))
21
17:1  (Top Level)                                                R Script

Console  Terminal  Jobs
~/
> trees_tbl <- sdf_copy_to(sc, trees, repartition = 3,overwrite=TRUE)
>
> trees_tbl %>%
+   spark_apply(function(e) head(e, 1))
# Source: spark<?> [?? x 3]
  Girth Height Volume
  <dbl>  <dbl>  <dbl>
1   8.3     70   10.3
2  11.3     79   24.2
3  14       78   34.5
```

source : https://spark.rstudio.com/guides/distributed-r/

R코드 분산 수행(Distributing R Computations)

spark_apply()

```
22  trees_tbl <- sdf_copy_to(sc, trees, repartition = 2,overwrite=TRUE)
23
24  trees_tbl %>%
25    spark_apply(function(e) head(e, 1))
26
```

22:1 (Top Level) R S

Console Terminal Jobs

```
> trees_tbl <- sdf_copy_to(sc, trees, repartition = 2,overwrite=TRUE)
>
> trees_tbl %>%
+   spark_apply(function(e) head(e, 1))
# Source: spark<?> [?? x 3]
  Girth Height Volume
  <dbl>  <dbl>  <dbl>
1   8.3     70   10.3
2  12.9     74   22.2
```

```
> iris_tbl %>%
+   spark_apply(nrow, group_by = "Species")
# Source: spark<?> [?? x 2]
  Species    result
  <chr>       <int>
1 versicolor     50
2 virginica      50
3 setosa         50
>
> iris_tbl %>%
+   spark_apply(
+     function(e) summary(lm(Petal_Length ~ Petal_Width, e))$r.squared,
+     names = "r.squared",
+     group_by = "Species")
# Source: spark<?> [?? x 2]
  Species    r.squared
  <chr>          <dbl>
1 versicolor     0.619
2 virginica      0.104
3 setosa         0.110
```

source : https://spark.rstudio.com/guides/distributed-r/

R코드 분산 수행(Distributing R Computations)

spark_apply()

```
> spark_apply(
+   iris_tbl,
+   function(e) broom::tidy(lm(Petal_Length ~ Petal_Width, e)),
+   names = c("term", "estimate", "std.error", "statistic", "p.value"),
+   group_by = "Species")
# Source: spark<?> [?? x 6]
  Species    term        estimate std.error statistic  p.value
  <chr>      <chr>          <dbl>     <dbl>     <dbl>    <dbl>
1 versicolor (Intercept)    1.78     0.284      6.28  9.48e- 8
2 versicolor Petal_Width    1.87     0.212      8.83  1.27e-11
3 virginica  (Intercept)    4.24     0.561      7.56  1.04e- 9
4 virginica  Petal_Width    0.647    0.275      2.36  2.25e- 2
5 setosa     (Intercept)    1.33     0.0600    22.1   7.68e-27
6 setosa     Petal_Width    0.546    0.224      2.44  1.86e- 2
```

With spark_apply() you can use any R package inside Spark. For instance, you can use the broom package to create a tidy data frame from linear regression output.

source : https://spark.rstudio.com/guides/distributed-r/

Apache Spark의 MLlib 활용

ml_로 시작하는 함수

```
# copy mtcars into spark
mtcars_tbl <- copy_to(sc, mtcars)

# transform our data set, and then partition into 'training', 'test'
partitions <- mtcars_tbl %>%
  filter(hp >= 100) %>%
  mutate(cyl8 = cyl == 8) %>%
  sdf_partition(training = 0.5, test = 0.5, seed = 1099)

# fit a linear model to the training dataset
fit <- partitions$training %>%
  ml_linear_regression(response = "mpg", features = c("wt", "cyl"))
fit
```

```
## Formula: mpg ~ wt + cyl
##
## Coefficients:
## (Intercept)          wt         cyl
##   33.499452   -2.818463   -0.923187
```

```
summary(fit)
```

```
## Deviance Residuals:
##    Min     1Q Median     3Q    Max
## -1.752 -1.134 -0.499  1.296  2.282
##
## Coefficients:
## (Intercept)          wt         cyl
##   33.499452   -2.818463   -0.923187
##
## R-Squared: 0.8274
## Root Mean Squared Error: 1.422
```

source : https://github.com/rstudio/sparklyr

간단한 Tip

show_query() 활용, collect는 사용 시 주의

```
> flights_tbl %>%
+   filter(dest %in% c("SJC", "SFO")) %>%
+   group_by(origin, dest) %>%
+   summarise(
+     num_departures = n(),
+     avg_dep_delay = mean(depdelay, na.rm = TRUE)
+   ) %>%
+   arrange(avg_dep_delay) %>%
+   show_query()
<SQL>
SELECT `origin`, `dest`, count(*) AS `num_departures`, AVG(`depdelay`)
AS `avg_dep_delay`
FROM `flights_p`
WHERE (`dest` IN ("SJC", "SFO"))
GROUP BY `origin`, `dest`
ORDER BY `avg_dep_delay`
```

수행 전에 show_query 기능을 사용하여 작성한 내용이
정확한지 확인

```
# using sparklyr:
delays_sample <- flights_spark %>%
    select(dep_delay, arr_delay) %>%
    na.omit() %>%
     sdf_sample(fraction = 0.05, replacement = FALSE) %>%
    collect()
```

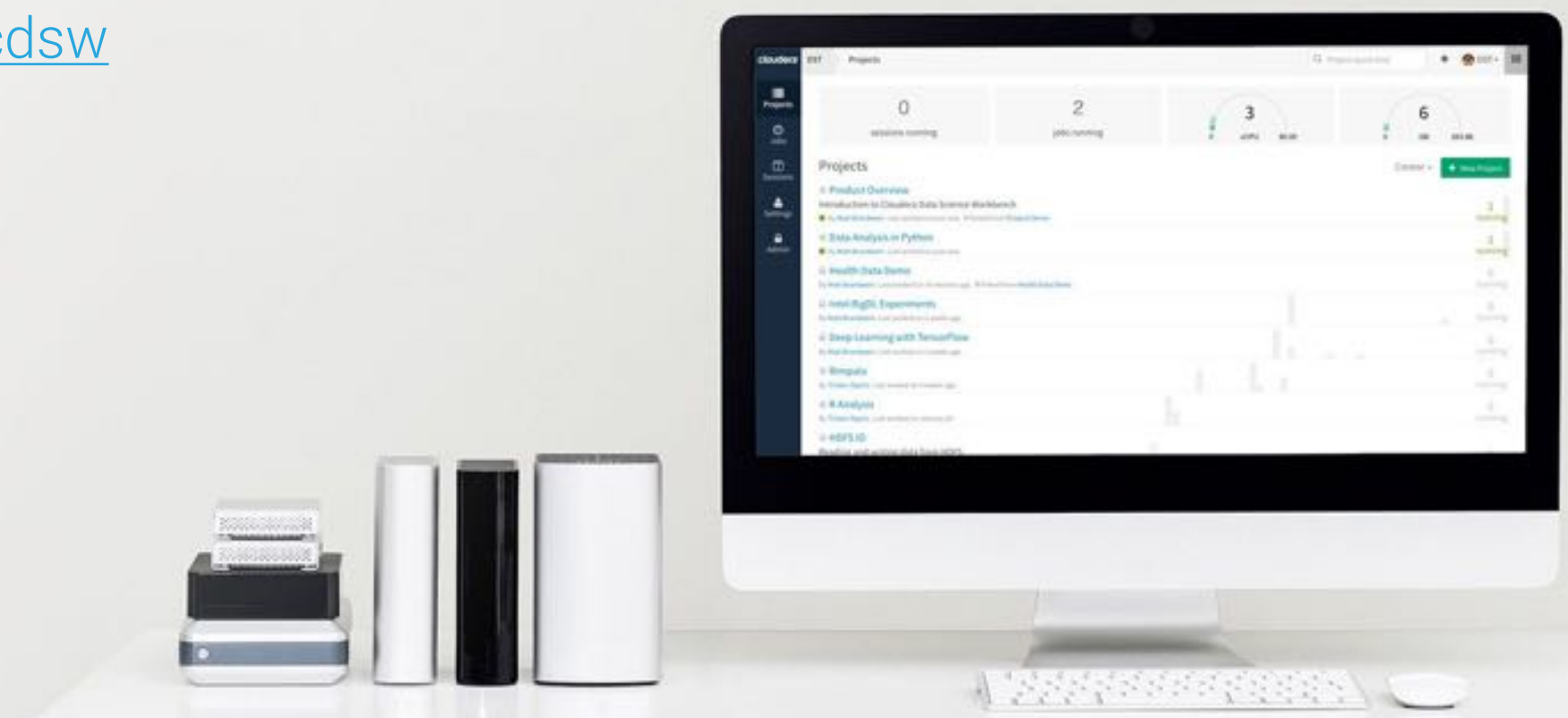
시각화를 위해 collect 하기 전에 필요한
데이터만으로 구성했는지 확인 필요

source : https://github.com/ianmcook/dplyr-examples/tree/master/five-tips

Cloudera Data Science Workbench

More information

tiny.cloudera.com/cdsw

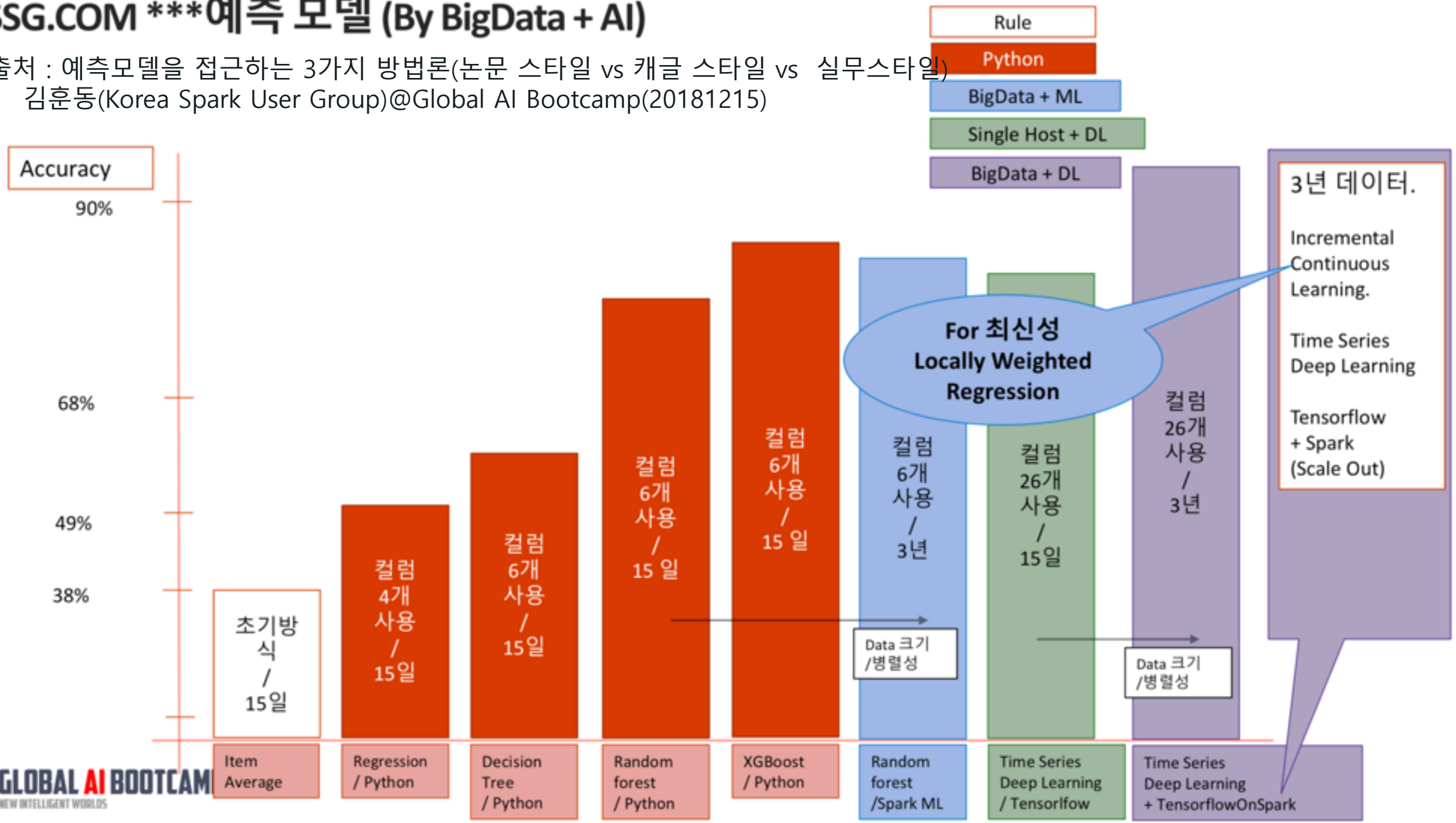


AI 실무 프로젝트 수행 전문가의 경험

(김훈동-KOREA SPARK USER GROUP)

SSG.COM ***예측 모델 (By BigData + AI)

출처 : 예측모델을 접근하는 3가지 방법론(논문 스타일 vs 캐글 스타일 vs 실무스타일)
- 김훈동(Korea Spark User Group)@Global AI Bootcamp(20181215)



Production A/B Test 에서 중요한 것!

논문 쓸 때랑, Kaggle 할 때와 달라요!

Sample Data, Selected Feature 정교한 모델		**더 많은 Data**(or 전수 Data)에 적용하는 Simple 모델
주1회 or 일1회	VS	**최신성** (매 시간 or 준실시간)
모두에게 적용하는 정교한 단일 모델		**개인화** 모델
최고 정확도		덜 정확해도 빠르고, **다수**에게, **실시간**으로...

Machine Learning vs Deep Learning

[장점]

- 보다 긴 역사를 가지고 있다.
- 많은 구현체, 검증된 방법론들이 있다.
- 보다 다양한 요건에 대한 방법론이 존재한다.
- 작은 데이터로 빠른 성능에 도달하기도 한다.

- 특정 분야에 있어서는 Dramatic한 성능 향상을 꽤 할 수 있다.
- Feature Selection 등에 큰 공을 들이지 않더라도, 모델이 end to end 로 학습하고 선별하는 경우가 많다.

[단점]

- 특정 분야에서는 Deep Learning 보다 정확도가 많이 떨어진다.
- Feature selection , 검증 등에 있어, Deep Learning 의 end to end 방식에 비하여 손이 많이 가고, feature 가 매우 많을 때에는 현실적이지 않을 수 있다.

- 상당히 많은 Label 데이터가 필요 할 때가 많다.
- 작은 양의 데이터로는 Under fit 되는 경우가 ML의 경우보다 더 많다.
- GPU 및 BigData Scale Computing 환경 등, 고사양의 Compute 자원을 필요로 한다.

출처 : Spark Deep learning A to Z, 김훈동(Korea Spark User Group), 201709 멜팅팡

Machine Learning vs Deep Learning
Sentimental Analysis for 한글

	한글문장 긍부정 Sentimental 분류
Naïve Bayes	83.2%
Word2Vec + CNN	85.4% ✓

작은 크기 데이터.
긍 부정 등 쉬운 분류 문제에서는 Machine Learning 도 매우 정확하고, 훨씬 Training 이 빠름!

1. Naïve Bayes
2. Word2Vec + CNN

출처 : Spark Deep learning A to Z, 김훈동(Korea Spark User Group), 201709 멜팅팡
- 긍부정 정확도 Score 출처. (by 송치성(바벨피쉬))

Machine Learning vs Deep Learning
Multi class Text Classification for 한글

1. Multinomial Naïve Bayes
2. Count Vector + SVM
3. TF-IDF + SVM
4. Word2Vec + CNN

	한글문장 138지 분류 (1:1고객응대 Data)
Multinomial Naïve Bayes	32.31%
Count Vector + SVM	17.28%
TF-IDF + SVM	51.21%
Word2Vec + CNN	59.00%

데이터 충분히 많은 경우.
좀 어려운 다지 분류 문제로 가보자. 

출처 : Spark Deep learning A to Z, 김훈동(Korea Spark User Group), 201709 멜팅팡

- 사용데이타 SSG.COM 1:1 고객응대 CS Data, 138지 분류 문제 (Top 1 맞추기)
- Training Data : 1,649,415건

Machine Learning vs Deep Learning
Multi class Text Classification for 한글

1.	Word2Vec + CNN (Batch Normalize + Augmentation)	72.30%
2.	Word2Vec + LSTM	73.94%
3.	Word2Vec + CNN + LSTM	72.97%
4.	Word2Vec + Bidirectional GRU	74.36%
5.	Word2Vec + Bidirectional GRU + Attention Network	73.15%
6.	FastText	72.50%
7.	Glove + LSTM (BigDL on Spark Cluster)	75.25%

출처 : Spark Deep learning A to Z, 김훈동(Korea Spark User Group), 201709 멜팅팡

- 사용데이타 : SSG.COM 1:1 고객응대 CS Data 총 31지 분류 (top 1 맞추기) – 고객 라벨링
- Training Data : 1,649,415건

THANK YOU