강사 : 신 인 호 교수

1. 스파크란?

- 아파치 스파크(apache spark)는 2011년 버클리 대학의 AMPLab에서 개발되어 현재는 아파치 재단의 오픈소스로 관리되고 있는 인메모리 기반의 대용량 데이터 고속 처리 엔진으로 범용 분산 클러스터 컴퓨팅 프레임워크입니다.

- 2014년 5월 정식 출시되었고, 2020년 5월 현재 2.4.5버전이 가장 최신 버전입니다. 또한, 2019년 12월 스파크 3.0의 프리뷰 버전이 출시 되었습니다.

1. Spark 개요

- Speed

  인메모리(In-Memory) 기반의 빠른 처리

- Ease of Use

  다양한 언어 지원(Java, Scala, Python, R, SQL)을 통한 사용의 편이성

- Generality

  SQL, Streaming, 머신러닝, 그래프 연산 등 다양한 컴포넌트 제공

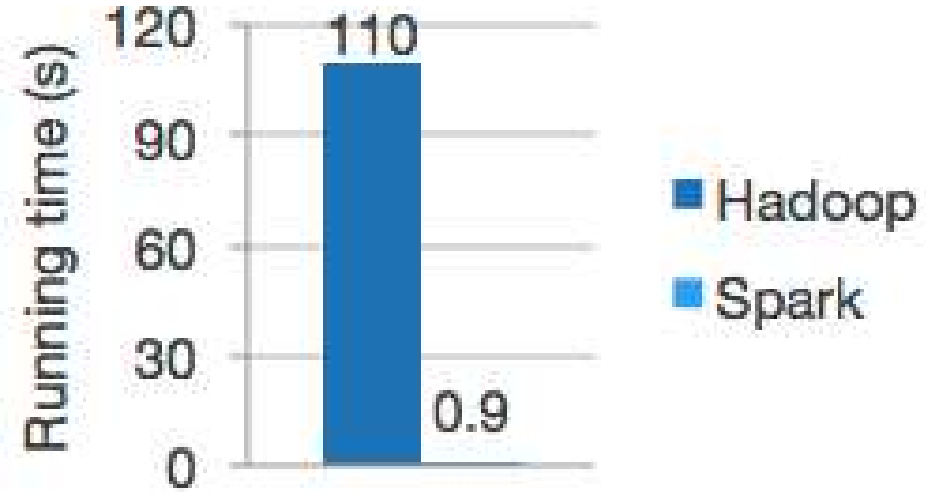
- Run Everywhere

  YARN, Mesos, Kubernetes 등 다양한 클러스터에서 동작 가능

  HDFS, Casandra, HBase 등 다양한 파일 포맷 지원

2. 특징 - 인메모리 기반의 빠른 처리

- 스파크는 인메모리 기반의 처리로 맵리듀스 작업처리에 비해 디스크는 10배, 메모리 작업은 100배 빠른 속도1를 가지고 있습니다. 맵리듀스는 작업의 중간 결과를 디스크에 쓰기 때문에 IO로 인하여 작업 속도에 제약이 생깁니다. 스파크는 메모리에 중간 결과를 메모리에 저장하여 반복 작업의 처리 효율이 높습니다.



2. 특징 - 다양한 컴포넌트 제공

- 스파크는 단일 시스템 내에서
- 스파크 스트리밍을 이용한 스트림 처리,
- 스파크 SQL을 이용한 SQL 처리,
- MLib 를 이용한 머신러닝 처리,
- GraphX를 이용한 그래프 프로세싱을 지원합니다.
- 추가적인 소프트웨어의 설치 없이도 다양한 애플리케이션을 구현할 수 있고,
- 각 컴포넌트간의 연계를 이용한 애플리케이션의 생성도 쉽게 구현할 수 있습니다.

2. 특징 - 다양한 언어지원

- 스파크는 자바, 스칼라, , R 인터페이스등
- 다양한 언어를 지원하여 개발자에게 작업의 편의성을 제공합니다. 하지만 언어마다 처리하는 속도가 다릅니다.
- 따라서 성능을 위해서는 Scala 로 개발을 진행하는 것이 좋습니다

2. 특징 - 다양한 파일 포맷 지원 및 Hbase, Hive 등과 연동 가능

- 스파크는 기본적으로 TXT, Json, ORC, Parquet 등의 파일 포맷을 지원합니다.
- S3, HDFS 등의 파일 시스템과 연동도 가능하고,
- HBase, Hive 와도 간단하게 연동할 수 있습니다.

🔥S3 (Simple Storage Service)

- S3는 최고의 확장성, 데이터 가용성 및 보안과 성능을 제공하는 객체 스토리지 서비스 입니다

S3 참고URL

PySpark 개요

1. PySpark 개요

- **PySpark**는 **Apache Spark**를 **Python**에서 사용할 수 있도록 만든 라이브러리입니다.
- **대용량 데이터를 빠르게 처리하고 분석할 수 있는 분산 컴퓨팅 프레임워크**입니다.
- PySpark = **Apache Spark + Python**

2. PySpark 특징

특징	설명
분산 처리	여러 노드(서버)에서 데이터를 나누어 병렬로 처리
빠른 속도	MapReduce보다 최대 100배 빠른 성능
확장성	단일 머신부터 클러스터까지 확장 가능
다양한 데이터 소스 지원	CSV, JSON, Parquet, HDFS, MySQL, Cassandra 등 지원
실시간 데이터 처리	Spark Streaming을 이용해 실시간 데이터 분석 가능

3. PySpark vs Pandas vs Dask 차이점

	PySpark	Pandas	Dask
데이터 크기	TB~PB (빅데이터)	MB~GB (작은 데이터)	GB~TB (중간 규모)
속도	분산 병렬 처리 (빠름)	단일 머신 (느림)	병렬 처리 (빠름)
확장성	클러스터 (확장 가능)	단일 머신 (확장 불가능)	멀티코어/멀티머신 (확장 가능)
API 스타일	Spark DataFrame	Pandas DataFrame	Pandas와 유사

5. PySpark의 적용

- 대량의 데이터를 빠르게 처리해야 할 때
- 하둡(Hadoop)이나 클러스터 환경에서 데이터 분석할 때
- 실시간 데이터 스트리밍이 필요할 때
- 머신러닝 모델을 빅데이터로 학습할 때

PySpark 기능및 장점

1. PySpark 주요 기능

1. Python API: Spark와 상호 작용하기 위한 Python API를 제공하여 Python 개발자가 Spark의 분산 컴퓨팅 기능을 활용할 수 있도록 합니다.
2. 분산 컴퓨팅: PySpark는 Spark의 분산 컴퓨팅 프레임워크를 활용하여 컴퓨터 클러스터에서 대규모 데이터를 처리하여 작업을 병렬로 실행할 수 있도록 합니다.
3. Fault Tolerance: RDD(복원력 있는 분산 데이터 세트)를 유지 관리하여 내 결함성을 자동으로 처리하여 오류를 정상적으로 복구할 수 있습니다.
4. 지연 평가: PySpark는 지연 평가를 사용하는데, 이는 데이터에 대한 변환이 즉시 실행되지 않고 작업이 트리거될 때까지 계산의 DAG(방향성 비순환 그래프)로 저장됨을 의미합니다.
5. Python 에코시스템과 통합: Python 에코시스템과 원활하게 통합되어 사용자가 데이터 조작 및 기계 학습 작업을 위해 pandas, NumPy 및 scikit-learn과 같은 인기 있는 Python 라이브러리를 활용할 수 있습니다.
6. 대화형 데이터 분석: PySpark는 Jupyter Notebook 및 대화형 Python 셸과의 통합 덕분에 대화형 데이터 분석 및 탐색에 적합합니다.

1. PySpark 주요 기능

7. **기계 학습:** PySpark에는 분류, 회귀, 클러스터링 등을 위한 광범위한 기계 학습 알고리즘을 제공하는 Spark의 확장 가능한 기계 학습 라이브러리인 MLlib가 포함되어 있습니다.
8. **스트리밍 처리:** Spark Streaming을 통한 스트리밍 처리를 지원하여 연속 데이터 스트림에 대한 실시간 데이터 처리 및 분석을 가능하게 합니다.
9. **제공합SQL 지원:** 사용자가 Spark SQL을 사용하여 분산 데이터 세트에서 SQL 쿼리를 수행할 수 있도록 하여 구조화된 데이터 작업을 위한 친숙한 인터페이스를 사용합니다.

1. PySpark 주요 기능

- **Spark SQL** → SQL을 사용하여 데이터 처리
- **DataFrame API** → Pandas와 비슷한 방식으로 데이터 처리
- **Spark Streaming** → 실시간 데이터 처리
- **MLlib** → 머신러닝 지원
- **GraphX** → 그래프 데이터 처리

2. PySpark 장점

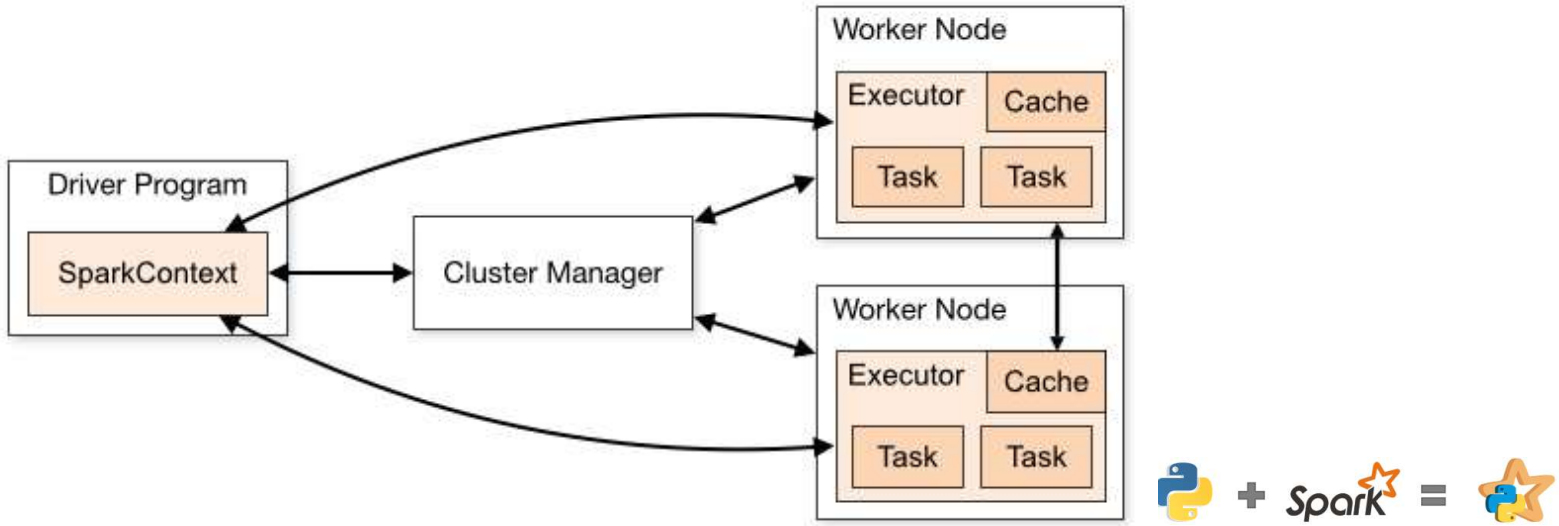
1. **확장성**: PySpark는 분산 컴퓨팅의 기능을 활용하여 머신 클러스터 전반에서 대규모 데이터 세트를 처리할 수 있도록 하여 증가하는 데이터 요구 사항을 수용할 수 있습니다.
2. **성능**: PySpark는 인메모리 컴퓨팅 및 병렬 처리를 활용하여 고성능을 달성하여 기존 단일 노드 처리에 비해 더 빠른 데이터 처리 및 분석을 가능하게 합니다.
3. **사용 편의성**: 사용자 친화적인 Python API를 제공하여 언어 구문 및 에코시스템에 익숙한 Python 개발자가 액세스할 수 있도록 합니다. 또한 데이터 분석 및 기계 학습을 위한 인기 있는 Python 라이브러리와 잘 통합됩니다.
4. **내결함성**: RDD(Resilient Distributed Dataset)를 통해 내결함성을 자동으로 처리하여 수동 개입 없이 데이터 안정성과 장애 복구를 보장합니다.

2. PySpark 장점

5. **통합 플랫폼**: 배치 처리, 대화형 데이터 분석, 스트리밍 처리 및 기계 학습을 포함한 다양한 데이터 처리 작업을 위한 통합 플랫폼을 제공하여 개발 및 배포 워크플로를 간소화합니다.
6. **실시간 처리**: PySpark는 스트리밍 및 구조적 스트리밍을 통해 데이터 스트림을 실시간으로 처리할 수 있도록 하여 변화하는 데이터에 대한 시기적절한 통찰력과 응답을 용이하게 합니다.
7. **기계 학습 기능**: 기계 학습 라이브러리인 MLlib가 포함되어 있어 인기 있는 기계 학습 알고리즘의 확장 가능한 구현을 제공하여 대규모 모델 학습 및 배포가 가능합니다.
8. **커뮤니티 및 에코시스템**: PySpark는 활발한 커뮤니티 및 에코시스템의 이점을 활용하여 광범위한 문서, 자습서 및 타사 패키지를 제공할 뿐만 아니라 커뮤니티의 지속적인 개발 및 지원을 제공합니다.

3. PySpark 아키텍처

- PySpark 아키텍처는 작업을 조정하고 클러스터 관리자와 상호 작용하여 리소스를 할당하는 드라이버 프로그램으로 구성됩니다.
- 드라이버는 작업자 노드와 통신하며, 여기서 작업은 실행기의 JVM 내에서 실행됩니다. SparkContext는 실행 환경을 관리하는 반면, DataFrame API는 데이터 조작을 위한 높은 수준의 추상화를 가능하게 합니다. SparkSession은 Spark 기능에 대한 통합 진입점을 제공합니다. 그 아래에서 클러스터 관리자는 노드 전체의 리소스 할당 및 작업 스케줄링을 감독하여 대규모 데이터를 효율적으로 처리하기 위한 병렬 계산을 용이

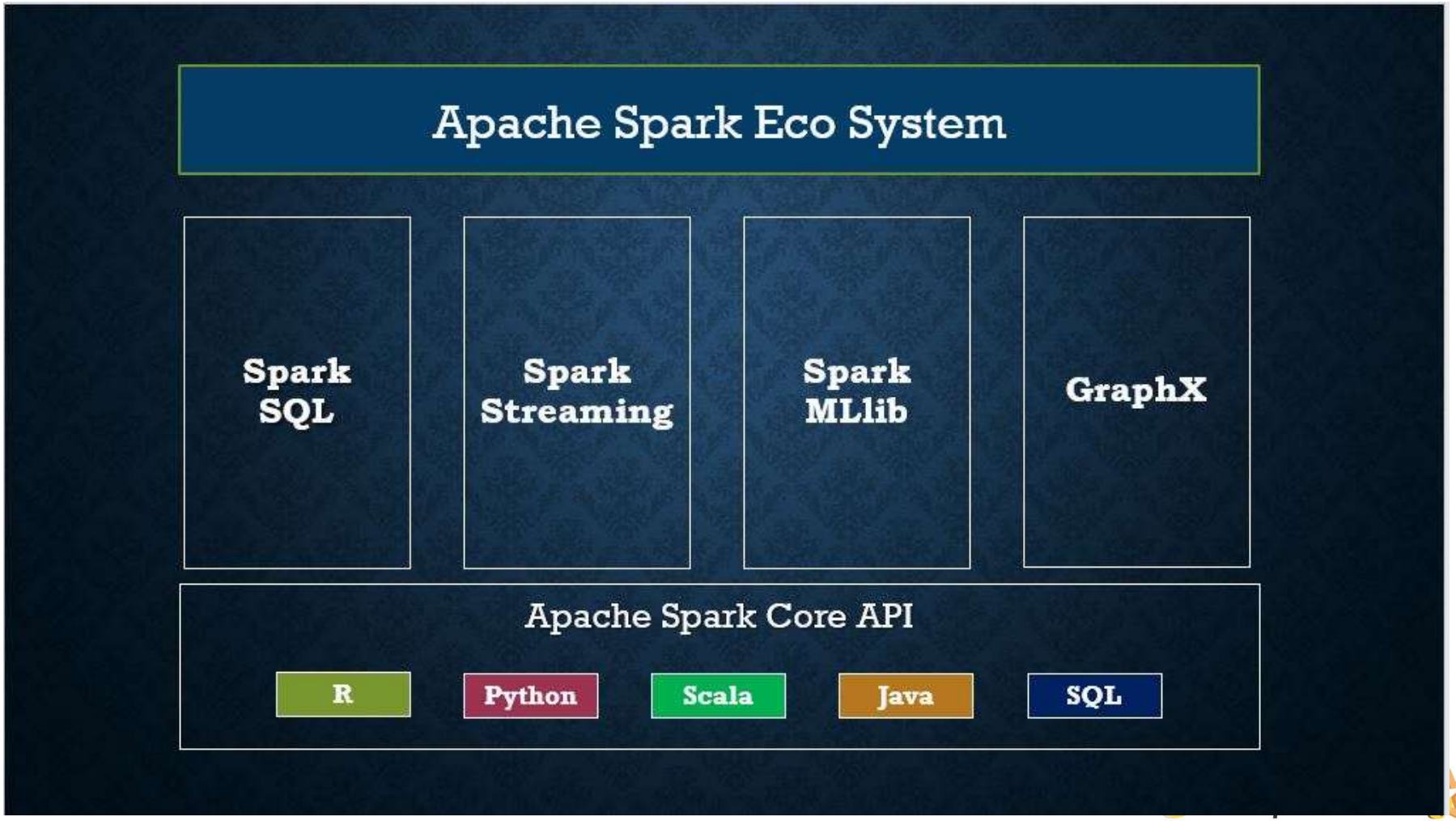


3. PySpark 아키텍처 – 클러스터 관리자

1. **독립 실행형**: 독립 실행형 클러스터 관리자는 응용 프로그램의 리소스를 관리하는 Spark와 함께 번들로 제공되는 간단한 독립 실행형 솔루션입니다. 중소 규모 클러스터에 적합하며 추가 소프트웨어 설치가 필요하지 않습니다.
2. **Mesos**: Mesos는 클러스터 전체에서 CPU, 메모리, 스토리지 및 기타 컴퓨팅 리소스를 추상화하는 분산 시스템 커널입니다. PySpark는 Mesos를 클러스터 관리자로 활용하여 Spark, Hadoop 등과 같은 여러 프레임워크 간에 리소스를 효율적으로 공유할 수 있습니다.
3. **하둡 YARN**(Yet Another Resource Negotiator): YARN은 하둡의 리소스 관리 계층으로, 하둡 클러스터 전반에서 리소스를 관리하고 스케줄링하는 역할을 합니다. PySpark는 YARN에서 실행할 수 있으므로 기존 Hadoop 에코시스템과 원활하게 통합하고 YARN의 리소스 관리 기능을 활용할 수 있습니다.
4. **쿠버네티스**: 쿠버네티스는 컨테이너화된 애플리케이션의 배포, 확장 및 관리를 자동화하는 컨테이너 오케스트레이션 플랫폼입니다. PySpark는 쿠버네티스에서 실행할 수 있으므로 컨테이너화된 환경에서 동적 리소스 할당과 효율적인 리소스 활용이 가능합니다.

3. pySpark 기능 및 장점

4. PySpark 모듈 & 패키지



5. 마무리

- PySpark = Apache Spark + Python
- TB~PB급 대용량 데이터를 분산 병렬 처리
- 데이터프레임, SQL, 스트리밍, 머신러닝 지원

Spark 실습

PySpark 다운로드 & 설치

- Java 설치
- Apache Spark 설치
- Spark 다운로드 페이지로 이동하여 사용하려는 Spark 버전을 선택한 다음 패키지 유형을 선택합니다. 포인트 3의 URL이 선택한 버전으로 변경됩니다. 3번 항목의 링크를 클릭하여 다운로드합니다.

PySpark 다운로드 & 설치

spark 설치

spark-3.5.4-bin-hadoop3

tar -xvf spark-3.5.4-bin-hadoop3.tgz

mv spark-3.5.3-bin-hadoop3 ~/spark

환경설정

vi .bashrc 에 설정추가

export SPARK_HOME=~/spark

export

PATH=$PATH:$SPARK_HOME/bin:$SPARK

_HOME/sbin

source .bashrc

echo $SPARK_HOME

spark-submit --version

spark-submit --help

실습에 필요한 프로그램 설치

🔥 spark 환경설정

cd ~/spark/conf

cp spark-env.sh.template spark-env.sh

nano ~/spark/conf/spark-env.sh **에 설정추가**

export SPARK_WORKER_INSTANCES=2

🔥 pip 설치

sudo apt-get install software-properties-common -y

sudo add-apt-repository universe

sudo apt-get update

sudo apt-get install python3-pip

🔥 pandas 설치

pip install pandas --break-system-packages

🔥 실시간 WordCount실습을 위한 netcat 설치(7.1)

sudo apt-get install netcat-openbsd 1.226-1ubuntu2

sudo apt-get install netcat

PySpark 다운로드 & 설치

```
# Set windows environment variables
SPARK_HOME  = C:₩apps₩spark-3.5.0-bin-hadoop3
HADOOP_HOME = C:₩apps₩spark-3.5.0-bin-hadoop3
PATH = %PATH%;C:₩apps₩spark-3.0.5-bin-hadoop3₩bin
```

1. SparkSession

- SparkSession은 스파크 작업을 수행하는 기본 진입점(entry point) 입니다.
- 즉, 데이터프레임을 생성하고, SQL을 실행하며, 클러스터에서 분산 연산을 수행하려면 무조건 SparkSession을 생성해야 합니다.

🔥 SparkSession의 역할

기 능	설 명
1. 스파크 실행 환경 초기화	SparkContext, Config, UI 등을 자동 설정
2. 데이터프레임 생성	CSV, JSON, Parquet, JDBC 등의 파일을 로드하여 DataFrame 생성
3. SQL 및 Hive 지원	spark.sql()을 사용하여 SQL 실행 가능
4. 분산 연산 관리	클러스터에서 분산 데이터 처리 수행
5. 캐시 및 메모리 관리	데이터프레임을 메모리에 캐시하여 성능 최적화

4.2 SparkSession

SparkSession은 스파크 작업을 수행하는 기본 진입점(entry point) 입니다.
즉, 데이터프레임을 생성하고, SQL을 실행하며, 클러스터에서 분산 연산을 수행하려면 무조건 SparkSession을 생성해야 합니다.

🔥 SparkSession 생성

```
from pyspark.sql import SparkSession

spark = SparkSession.builder ₩
        .appName('SparkByExamples.com') ₩
        .getOrCreate()
```

4.3 정의된 변수확인

방 법	설 명
locals()	현재 정의된 모든 지역 변수 확인
globals()	현재 정의된 모든 전역 변수 확인
dir()	모든 변수와 객체의 목록 확인
%whos, %who	Jupyter Notebook에서 변수 확인
isinstance()	특정 타입의 변수만 필터링하여 확인
spark.catalog.listTables()	PySpark에서 사용 가능한 테이블 목록 확인

4.4 정의된 DataFrame 확인

```
from pyspark.sql import DataFrame
```

🔥 **현재 정의된 변수 중에서 DataFrame 객체만 출력**

```
for var_name, obj in locals().items():
    if isinstance(obj, DataFrame):
        print(f"{var_name}: {obj}")
```

Data 처리

1. DataFrame 활용

샘플 데이터

```
emp = [( ' James ' , '  ' , ' Smith ' , ' 2020-04-01 ' , ' M ' ,300000),
       ( ' Michael ' , ' Rose ' , '  ' , ' 2024-05-19 ' , ' M ' ,400000),
       ( ' Robert ' , '  ' , ' Williams ' , ' 2023-09-05 ' , ' M ' ,400000),
       ( ' Maria ' , ' Anne ' , ' Jones ' , ' 2019-12-01 ' , ' F ' ,400000),
       ( ' Jen ' , ' Mary ' , ' Brown ' , ' 2018-02-17 ' , ' F ' ,-1)]
columns = [ " firstname " , " middlename " , " lastname " , " ipsail " ,"sex " , " salary " ]
```

DataFrame 생성

```
df_emp = spark.createDataFrame(data=emp, schema = columns)
df_emp.show()
df_emp.dtypes
```

2. DataFrame – CSV 파일 읽기

```
hakjum_st = StructType([
            StructField("team"   ,T.StringType(), True),
            StructField("irum"   ,T.StringType(), True),
            StructField("gwamok" ,T.StringType(), True),
            StructField("jumsu"  ,T.StringType(), True)
])
```

```
df_hakjum=spark.read.csv("./hakjum.dat", header=True, inferSchema=True)
```

```
df_hakjum=spark.read.csv("./hakjum.dat", header=True, schema= hakjum_st)
```

3. Data 조회 –함수

- df.count() – 각 열의 개수를 반환합니다(개수에는 null이 아닌 값만 포함됨).
- df.corr() – 데이터 프레임의 열 간의 상관 관계를 반환합니다.
- df.head(n) – 맨 위에서 처음 n개 행을 반환합니다.
- df.max() – 각 열의 최대값을 반환합니다.
- df.mean() – 각 열의 평균을 반환합니다.
- df.median() – 각 열의 중앙값을 반환합니다.
- df.min() – 각 열의 최소값을 반환합니다.
- df.std() – 각 열의 표준 편차를 반환합니다.
- df.tail(n) – 마지막 n개 행을 반환합니다.

3. Data 조회 –함수

- select() – DataFrame에서 특정 열을 선택합니다.
- filter() – 조건에 따라 행을 필터링합니다.
- groupBy() – 하나 이상의 열을 기준으로 행을 그룹화합니다.
- agg() – 그룹화된 데이터에 대해 집계 함수(예: 합계, 평균)를 수행합니다.
- orderBy() – 하나 이상의 열을 기준으로 행을 정렬합니다.
- dropDuplicates() – DataFrame에서 중복 행을 제거합니다.
- withColumn() – 새 열을 추가하거나 기존 열을 수정된 데이터로 바꿉니다.
- drop() – DataFrame에서 하나 이상의 열을 제거합니다.
- join() – 공통 열 또는 인덱스를 기반으로 두 개의 DataFrame을 병합합니다.
- pivot() – DataFrame을 피벗하여 열 값을 기반으로 데이터를 재구성합니다.

4. Data 조회 – select, groupby

```
from pyspark.sql.functions import mean, max, min
from pyspark.sql.functions import col, when
```

🔥 select

```
df_hakjum.select('team','gwamok',jumsu').show()
```

🔥 전체 평균점수 확인

```
df_hakjum.select(mean('jumsu') ).show()
```

🔥 그룹의 사용

```
df_hakjum.groupby('team','gwamok').count().show()

df_hakjum.groupby("team","gwamok")₩
        .agg(mean("jumsu"),max("jumsu"), min("jumsu"))₩
        .show()
df_hakjum.groupby('team','gwamok').mean('jumsu').show()
```

4. Data 조회 – filter, join

🔥 Filter 사용

```
df_hakjum.filter(df_hakjum['gwamoK'] == 'java').show()
df_hakjum.filter(df_hakjum['jumsu'].between(70, 80)).show()
```

🔥 Join 사용

```
from pyspark.sql import SparkSession
```

🔥 Join SparkSession 생성

```
spark = SparkSession.builder.appName("Join Example").getOrCreate()

data1 = [(1, "Alice", 25), (2, "Bob", 30), (3, "Charlie", 35)]
data2 = [(1, "HR"), (2, "Engineering"), (4, "Marketing")]
columns1 = ["id", "name", "age"]
columns2 = ["id", "department"]

df1 = spark.createDataFrame(data1, columns1)
df2 = spark.createDataFrame(data2, columns2)
```

4. Data 조회 – filter, join

🔥 LEFT JOIN

```
df_left = df1.join(df2, "id", "left")
df_left.show()
```

🔥 RIGHT JOIN

```
df_right = df1.join(df2, "id", "right")
df_right.show()
```

🔥 OUTER JOIN

```
df_outer = df1.join(df2, "id", "outer")
df_outer.show()
```

🔥 **필요한 컬럼만 선택**

```
df1.join(df2, "id").select(df1.id, df1.name, df2.department).show()
```

5. Table, view 생성

🔥 Table 생성

```
df_hakjum.write.saveAsTable('hakjum_tbl')
```

🔥 view 생성

```
df_hakjum.createOrReplaceTempView('hakjum_view')
```

6. Table 확인

```
spark.catalog.listTables()
spark.sql("SHOW TABLES").show()
spark.catalog.listTables("default")
spark.sql("DESCRIBE hakjum_tbl ").show()
spark.catalog.listDatabases()
spark.catalog.tableExists(“hakjum_tbl")
spark.catalog.listColumns(" hakjum_tbl ")
spark.sql("SELECT current_database()").show()
```

spark.sql

1. Spark.sql – 조회

- Table로 조회

```
spark.sql('select team, irum, gwamok,jumsu from hakjum_tbl').show()
spark.sql('select team, gwamok,avg(jumsu) from hakjum_tbl \
          group by team,gwamok').show()
```

- view로 조회

```
spark.sql('select team, irum, gwamok,jumsu from hakjum_view').show()
```

- dataframe으로 조회

```
spark.sql('select team, irum, gwamok,jumsu from {df} ', df=df_hakjum).show()
```

Data의 변환(Pandas, PySpark)

1. Pyspark에서 Pandas를 활용

```
import pandas as pd


# Pandas DataFrame 생성
data = {"id": [1, 2, 3], "name": ["Alice", "Bob", "Charlie"]}
pdf = pd.DataFrame(data)


# Pandas DataFrame 출력
print(pdf)
```

1. PySpark DataFrame → Pandas DataFrame 변환

```
from pyspark.sql import SparkSession

# SparkSession 생성
spark = SparkSession.builder.appName("PandasExample").getOrCreate()

# PySpark DataFrame 생성
data = [(1, "Alice"), (2, "Bob"), (3, "Charlie")]
columns = ["id", "name"]

df = spark.createDataFrame(data, columns)

# PySpark DataFrame을 Pandas DataFrame으로 변환
pdf = df.toPandas()

print(pdf)  # Pandas DataFrame 출력
```

2. Pandas DataFrame → PySpark DataFrame 변환

```
# Pandas DataFrame을 PySpark DataFrame으로 변환
spark_df = spark.createDataFrame(pdf)

spark_df.show()  # PySpark DataFrame 출력
```

3. PySpark에서 pandas UDF(User Defined Function) 사용

```
from pyspark.sql.functions import pandas_udf
from pyspark.sql.types import IntegerType

# pandas UDF 정의
@pandas_udf(IntegerType())
def squared_udf(s: pd.Series) -> pd.Series:
    return s * s

# PySpark DataFrame에 UDF 적용
df = spark.createDataFrame([(1,), (2,), (3,)], ["value"])
df.withColumn("squared", squared_udf(df["value"])).show()
```

3. PySpark에서 pandas UDF(User Defined Function) 사용

작업	코드
pandas 임포트	import pandas as pd
PySpark DF → Pandas DF	df.toPandas()
Pandas DF → PySpark DF	spark.createDataFrame(pdf)
Pandas UDF 사용	@pandas_udf(IntegerType())

실시간 WordCount

1. 관련 S/W설치

apt-get update
apt-get install netcat-openbsd 1.226-1ubuntu2

네트워크 소켓 서버 구동

nc -lk 9999

- nc: netcat 프로그램 실행 명령어
- l: listening 모드 활성화
  이 옵션을 통해 netcat은 연결을 기다리는 서버로 동작
- k: 연결이 끊어진 후에도 계속해서 리스닝을 유지한다.
- 9999: netcat이 연결을 수신할 네트워크 포트 번호

2. 실시간 Streaming WC

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import explode, split
```

🔥 SparkSession 초기화

```
spark = SparkSession.builder ₩
    .appName("StructuredNetworkWordCount") ₩
    .getOrCreate()
```

2. 실시간 Streaming WC

- 스트리밍 데이터 소스로부터 데이터 읽기 설정

```
lines = spark.readStream ₩
    .format("socket") ₩
    .option("host", "localhost") ₩
    .option("port", 9999) ₩
    .load()
```

- 입력 데이터에서 단어 분할

```
words = lines.select(
    explode(split(lines.value, " ")
    ).alias("word")
)
```

2. 실시간 Streaming WC

- 단어의 출현 빈도 계산

```
wordCounts = words.groupBy("word").count()
```

- 결과를 콘솔에 출력하는 스트리밍 쿼리 시작

```
query = wordCounts.writeStream ₩
    .outputMode("complete") ₩
    .format("console") ₩
    .start()
```

- 종료

```
query.awaitTermination()
```

감사합니다.