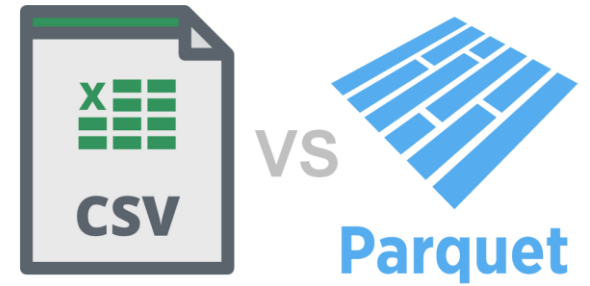


PARQUET

*.csv 포맷으로는 만족하지 못하는 경우



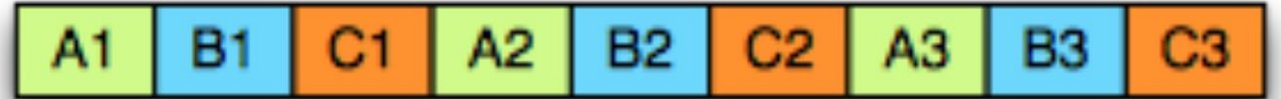
- Data Type이 저장되지 않는다.
- 너무 많은 데이터는 저장해도 csv의 이점을 살리지 못한다.
- 특정 Column만 선택하는 것이 불가능하다.
- (압축을 하지 않은 경우) 용량이 상대적으로 작지만 크다.
- Escaping이 잘 되지 않은 경우에는 파일 Parsing이 깨진다.
- 한글이 들어간 csv의 경우 UTF-8을 제대로 인식하지 못한다.

CSV vs. JSON vs. PARQUET

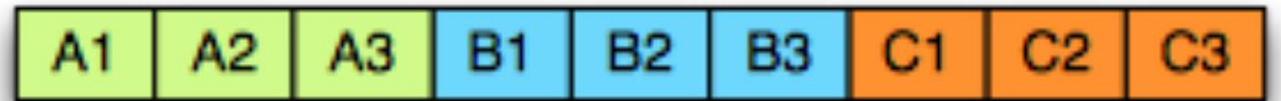
Spark Format Showdown		File Format		
		<u>CSV</u>	<u>JSON</u>	<u>Parquet</u>
A t t r i b u t e	Columnar	No	No	Yes
	Compressable	Yes	Yes	Yes
	Splittable	Yes*	Yes**	Yes
	Human Readable	Yes	Yes	No
	Nestable	No	Yes	Yes
	Complex Data Structures	No	Yes	Yes
	Default Schema: Named columns	Manual	Automatic (full read)	Automatic (instant)
	Default Schema: Data Types	Manual (full read)	Automatic (full read)	Automatic (instant)

컬럼기반 저장 포맷

A	B	C
A1	B1	C1
A2	B2	C2
A3	B3	C3



<실제 데이터 저장 방식>



<Parquet 형식의 데이터 저장 방식>

컬럼기반 저장 포맷인 Parquet의 목적도
필요한 데이터만 디스크로부터 읽어 I/O를 최소화하고, 데이터 크기를 줄이는 것이다.

Parquet의 장점

1. 압축률이 더 좋다.

유사한 데이터들이 모여 있기 때문에 데이터들이 많이 모이면 압축하기가 좋습니다.

2. I/O 사용률이 줄어든다.

데이터를 읽어 들일 때 일부 컬럼만을 스캔하기 때문입니다.

3. 컬럼별로 적합한 인코딩을 사용할 수 있다.

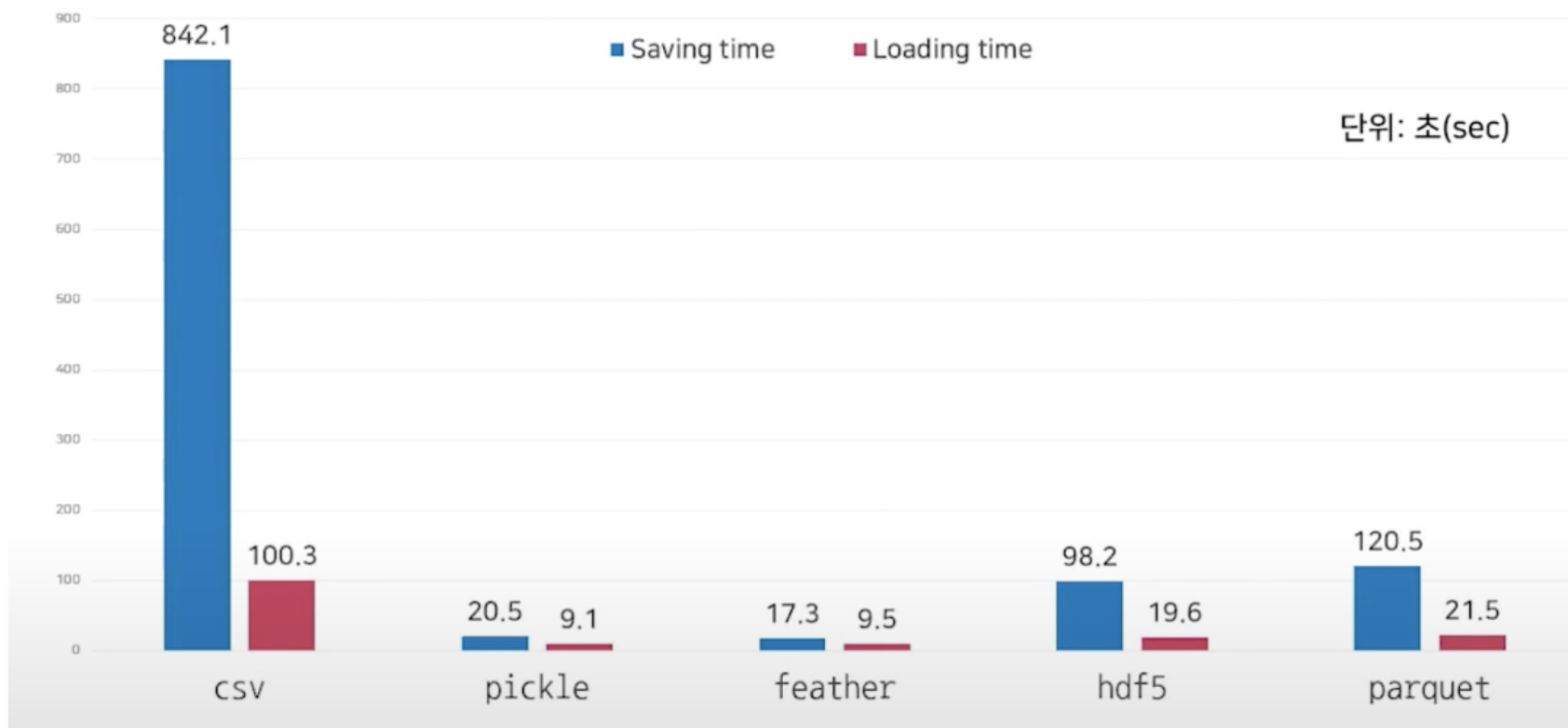
각 컬럼의 데이터들은 동일한 유형의 데이터를 저장하기 때문입니다. 따라서 컬럼마다 서로 다른 (데이터형에 유리한) 인코딩을 사용할 수 있습니다.



Parquet

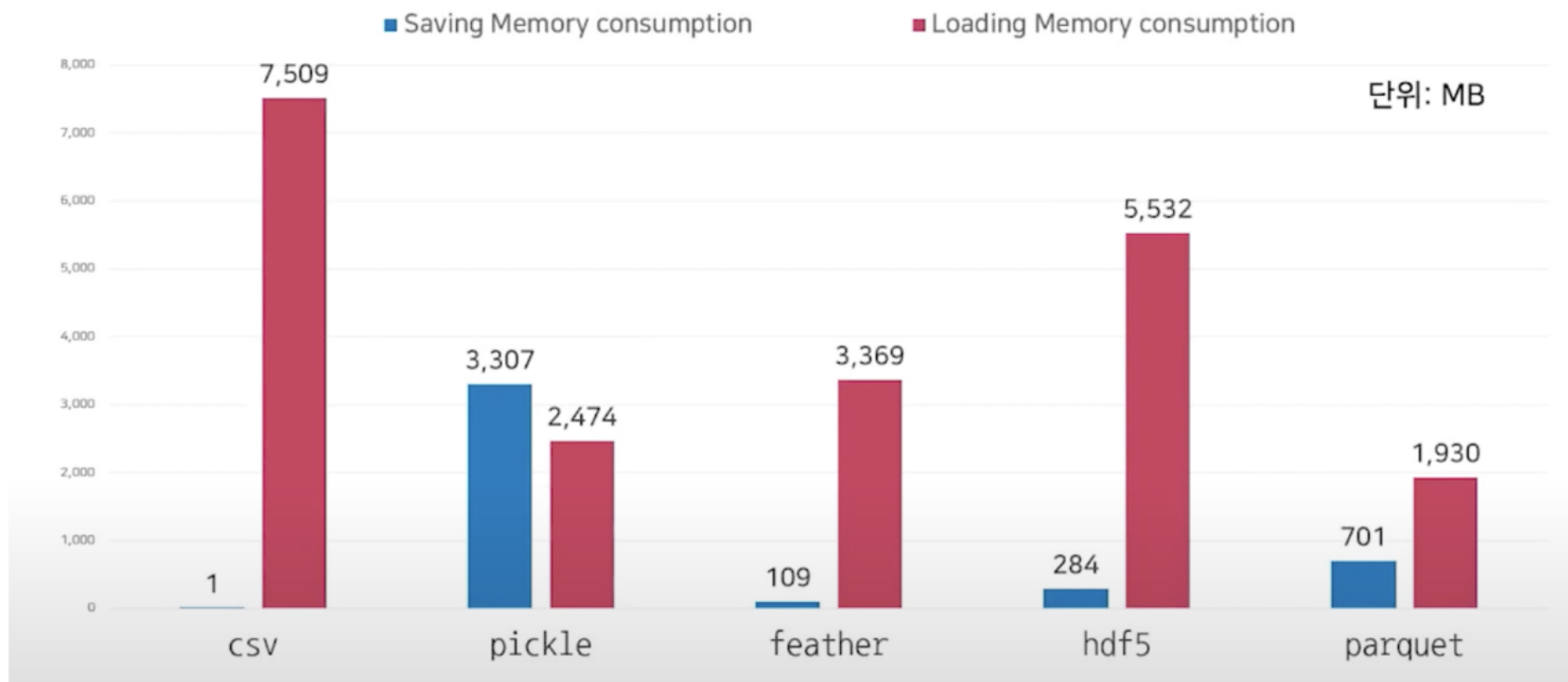
파일 형식에 따른 비교 - 저장/로드 시간

pickle, feather < hdf5, parquet <<< csv



파일 형식에 따른 비교 - 저장/로드 메모리 사용량

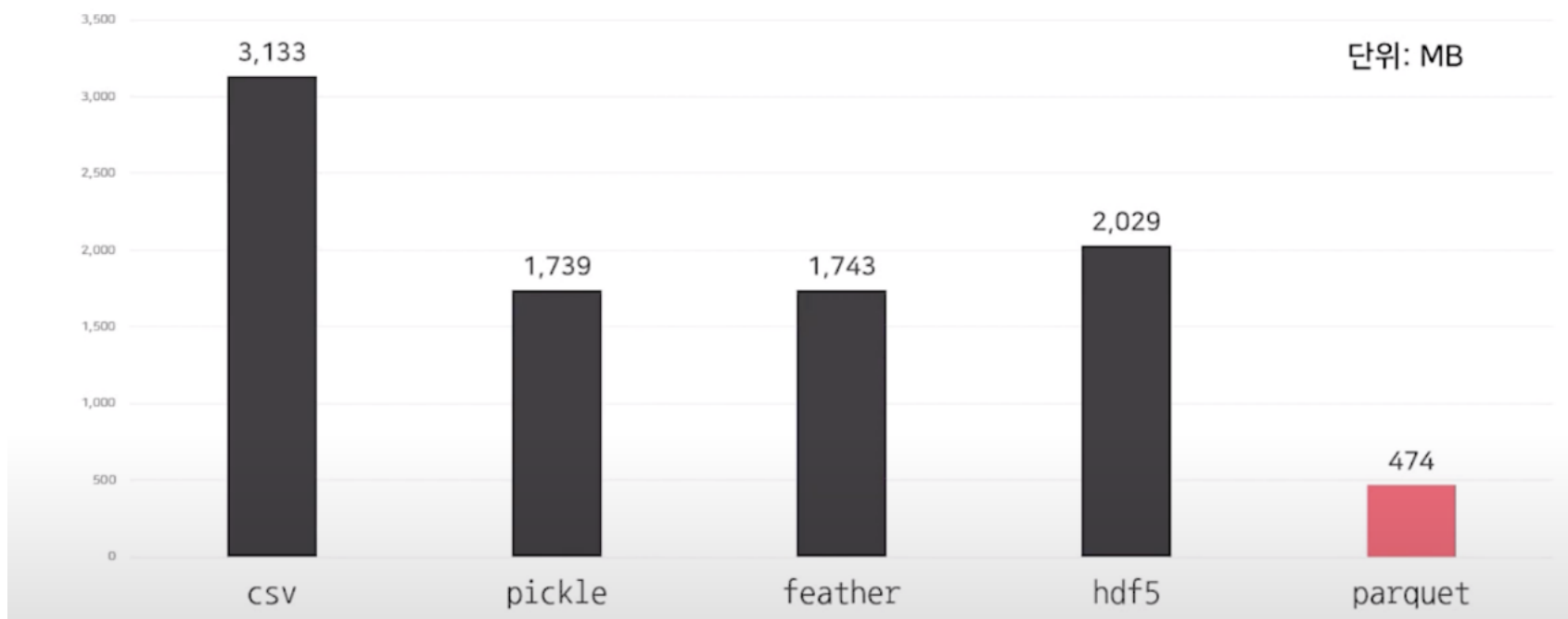
parquet < pickle, feather < hdf5 < csv



파일 형식에 따른 비교 - 저장 파일 크기

parquet < pickle < feather < hdf5 < csv

■ csv ■ pickle ■ feather ■ hdf5 ■ parquet



Parquet 사용 예제

```
# !pip install fastparquet

# save
import pandas as pd

df = pd.DataFrame()
df.to_parquet('sample.parquet', compression='gzip')

# read
df = pd.read_parquet('sample.parquet')
print(df)
```

Snappy

Parquet형식은 Pandas에서 기본 옵션으로 **Snappy** 압축을 사용

Gzip

Snappy 압축이 좋기는 하지만 위와 같이 빌드 관련한 의존 패키지
도 설치해야하고, 때로는 의존성 라이브러리도 이슈가 종종 있어
사용하기 까다로운 측면이 있음

따라서 시스템에서 보통 잘 지원하는 gzip 형식을 이용하기도 함

압축률은 $\text{gzip} > \text{snappy}$ 이며, 압축 속도는 $\text{gzip} < \text{snappy}$ 로 약간의 차이는 있다.

Parquet 사용 예제

```
df = pd.DataFrame()  
df.to_parquet('sample.parquet', compression='gzip')
```

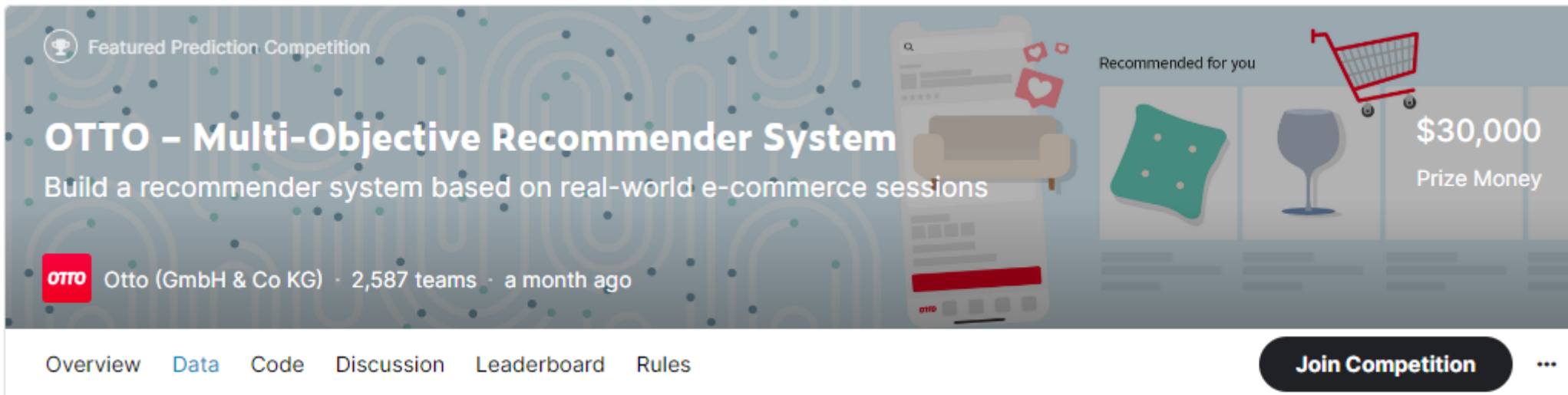
한편, 어떤 압축 방식(Gzip/Snappy/Uncompressed)을 사용하든 파일을 읽는 방식은 동일하다.
이때는 압축 방식을 알아서 유추해서 풀기 때문에 별도의 옵션을 주지 않아도 된다.

```
df = pd.read_parquet('sample.parquet')
```

또한 특정 컬럼만 읽으려고 한다면
아래와 같이 **columns** 인자를 전달하면 파일 전체 대신 해당 컬럼만 읽어서 DataFrame을 생성한다.

```
df = pd.read_parquet('sample.parquet', columns=['a', 'b'])
```

Parquet 적용 사례



The banner for the OTTO competition features a dark blue background with a pattern of white dots and lines. On the left, it says "Featured Prediction Competition" with a trophy icon. The main title "OTTO - Multi-Objective Recommender System" is in large white letters, followed by the subtitle "Build a recommender system based on real-world e-commerce sessions". Below this, the OTTO logo is shown next to "Otto (GmbH & Co KG) · 2,587 teams · a month ago". On the right, there's a graphic of a smartphone displaying a shopping app, a shopping cart icon, and a prize money amount of "\$30,000". At the bottom, there are navigation links: "Overview", "Data" (highlighted with a blue underline), "Code", "Discussion", "Leaderboard", and "Rules". A black button with white text says "Join Competition", followed by three dots.

Featured Prediction Competition

OTTO – Multi-Objective Recommender System

Build a recommender system based on real-world e-commerce sessions

OTTO Otto (GmbH & Co KG) · 2,587 teams · a month ago

Overview Data Code Discussion Leaderboard Rules

[Join Competition](#) ...

Dataset Description

The goal of this competition is to predict e-commerce clicks, cart additions, and orders. You'll build a multi-objective recommender system based on previous events in a user session.

The training data contains full e-commerce `session` information. For each `session` in the test data, your task is to predict the `aid` values for each session `type` that occurs after the last timestamp `ts` in the test session. In other words, the test data contains sessions truncated by timestamp, and you are to predict what occurs after the point of truncation.

For additional background, please see the published [OTTO Recommender Systems Dataset](#) GitHub.

Files

3 files

Size

11.89 GB

Type

jsonl, csv

Parquet 적용 사례

```
import os

import numpy as np
import pandas as pd

from pathlib import Path
from tqdm import tqdm

data_path = Path('/kaggle/input/otto-recommender-system/')
chunksize = 100_000
```

```
chunks = pd.read_json(data_path / 'train.jsonl', lines=True, chunksize=chunksize)
os.mkdir('train_parquet')

for e, chunk in enumerate(tqdm(chunks, total=129)):
    event_dict = {
        'session': [],
        'aid': [],
        'ts': [],
        'type': [],
    }

    for session, events in zip(chunk['session'].tolist(), chunk['events'].tolist()):
        for event in events:
            event_dict['session'].append(session)
            event_dict['aid'].append(event['aid'])
            event_dict['ts'].append(event['ts'])
            event_dict['type'].append(event['type'])

    # save DataFrame
    start = str(e*chunksize).zfill(9)
    end = str(e*chunksize+chunksize).zfill(9)
    pd.DataFrame(event_dict).to_parquet(f"train_parquet/{start}_{end}.parquet")
```

```
chunks = pd.read_json(data_path / 'test.jsonl', lines=True, chunksize=chunksize)
os.mkdir('test_parquet')

for e, chunk in enumerate(tqdm(chunks, total=17)):
    event_dict = {
        'session': [],
        'aid': [],
        'ts': [],
        'type': [],
    }

    for session, events in zip(chunk['session'].tolist(), chunk['events'].tolist()):
        for event in events:
            event_dict['session'].append(session)
            event_dict['aid'].append(event['aid'])
            event_dict['ts'].append(event['ts'])
            event_dict['type'].append(event['type'])

    # save DataFrame
    start = str(e*chunksize).zfill(9)
    end = str(e*chunksize+chunksize).zfill(9)
    pd.DataFrame(event_dict).to_parquet(f"test_parquet/{start}_{end}.parquet")
```

Data Explorer

11.89 GB

sample_submission.csv
test.jsonl
train.jsonl

Data Explorer

Version 1 (2.31 GB)

test_parquet
train_parquet