데이터 관리론

Data Engineering with Apache Airflow

Data Engineering with Apache Airflow

Crash Course for Airflow

- File and Stream
- Bash shell
- Decorator Function
- Class

File and Stream

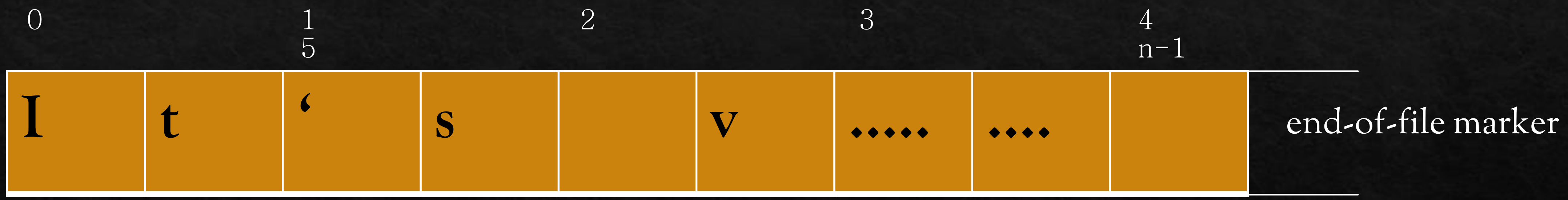
- In Python, "streams" and "file-like objects" refer to entities that allow sequential access to data.

- Streams
  - A sequence of data elements made available over time.
  - In Python, streams are often associated with reading or writing data in a continuous or sequential manner.

- File-Like Objects
  - Objects that behave like files but might not necessarily be actual files.
  - Objects that implement methods such as read(), write(), and others, making them compatible with operations that would typically be performed on files.

Files and Streams

- Python views files as sequential streams of bytes.
- Each File ends with an end-of-file marker.
- Opening a file creates an object associated with a stream.



Python’s view a file of n bytes

Files and Streams

Text Encoding

```
with open("sample.txt", "w") as file:
    file.write("It's simple code.")
```


I	t	‘	s		v	.....	....	

```
with open("sample.txt", "w") as file:
    file.write("It's simple code.")

if file.closed:
    print('File is closed')
else:
    print('File is still open')
```

→ **True or False ?**

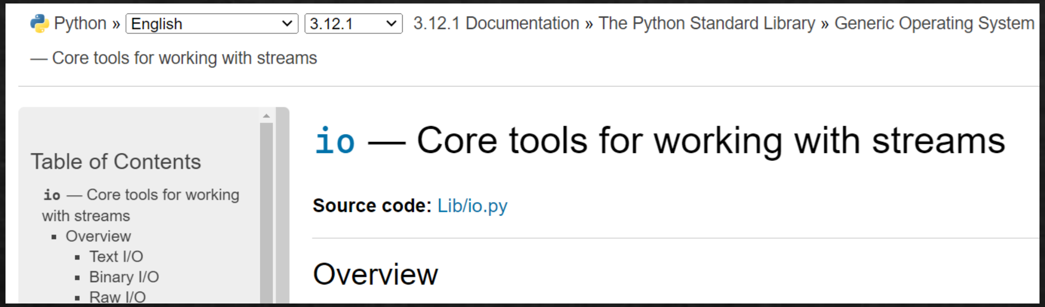
Files and Streams

```
chunk_size = 1  # Define your preferred chunk size
num = 0

with open('sample.txt', 'rb') as file:
    while True: )       # None 은 false  즉 모두 읽으면
False
        chunk = file.read(chunk_size)

        num += 1

        if not chunk:
            break
        # Process the chunk (e.g., print, analyze, etc.)
        print(chunk)

print(num)
```

```
import os
os.path.getsize('./sample.txt’)
#
17
```

영문자 1개는 1 chunk(byte)

File and Stream



Python » English 3.12.1 3.12.1 Documentation » The Python Standard Library » Generic Operating System — Core tools for working with streams

Table of Contents

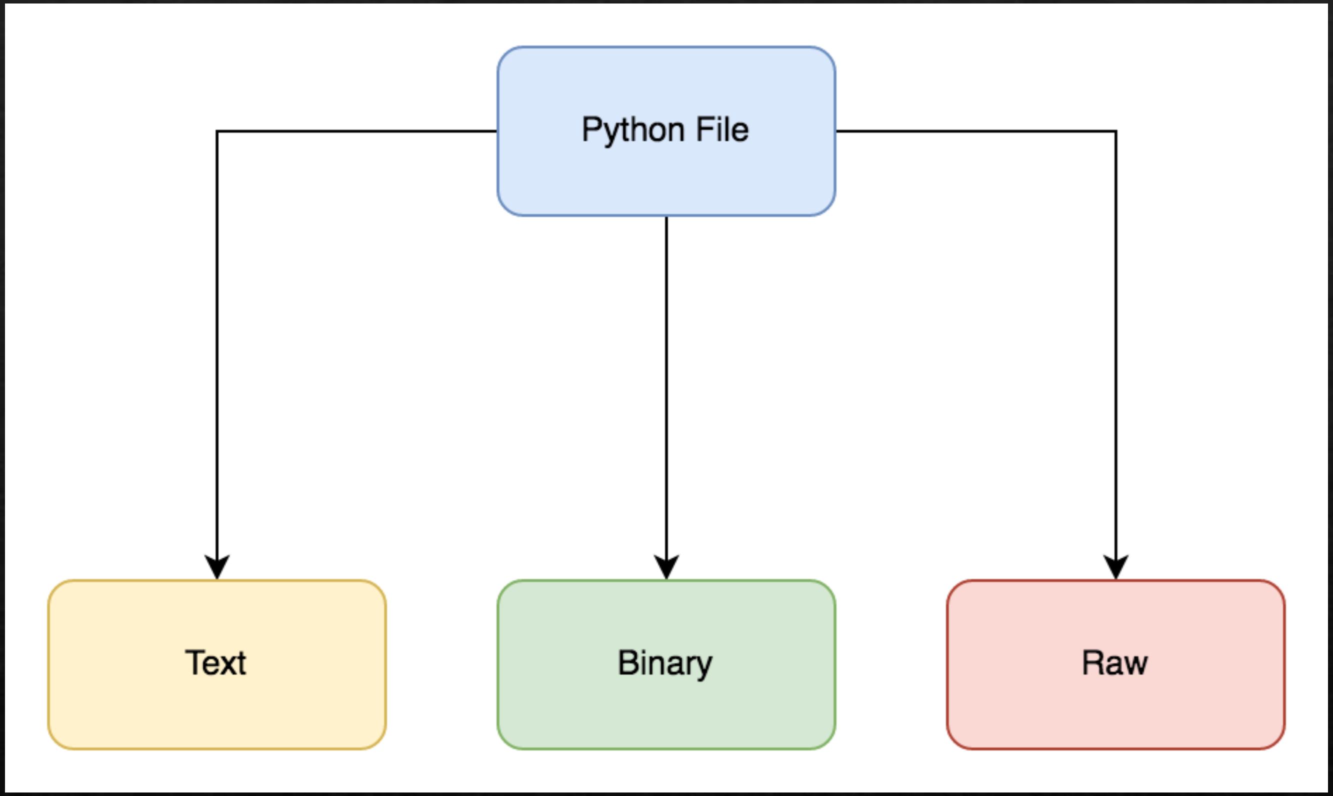
io — Core tools for working with streams
- Overview
  - Text I/O
  - Binary I/O
  - Raw I/O

io — Core tools for working with streams

Source code: Lib/io.py

Overview

https://docs.python.org/ko/3/library/io.html



https://medium.com/dev-bits/ultimate-guide-for-working-with-i-o-streams-and-zip-archives-in-python-3-6f3cf96dca50

File and Stream

```
with open("sample.txt", "r") as file:
    content = file.read()
    print(content)
```

```
import io
# Open a file-like object
file_content = "It's simple code."
file_like_object = io.StringIO(file_content)
# Open the file in write mode and write the content
with open("sample.txt", "w") as file:
    file.write(file_like_object.getvalue())
```

This approach is useful when you want to work with file-like objects **in-memory** without dealing with actual files on disk.

실습 chapter 2. Python Script

```
import json
import requests
from pathlib import Path
```

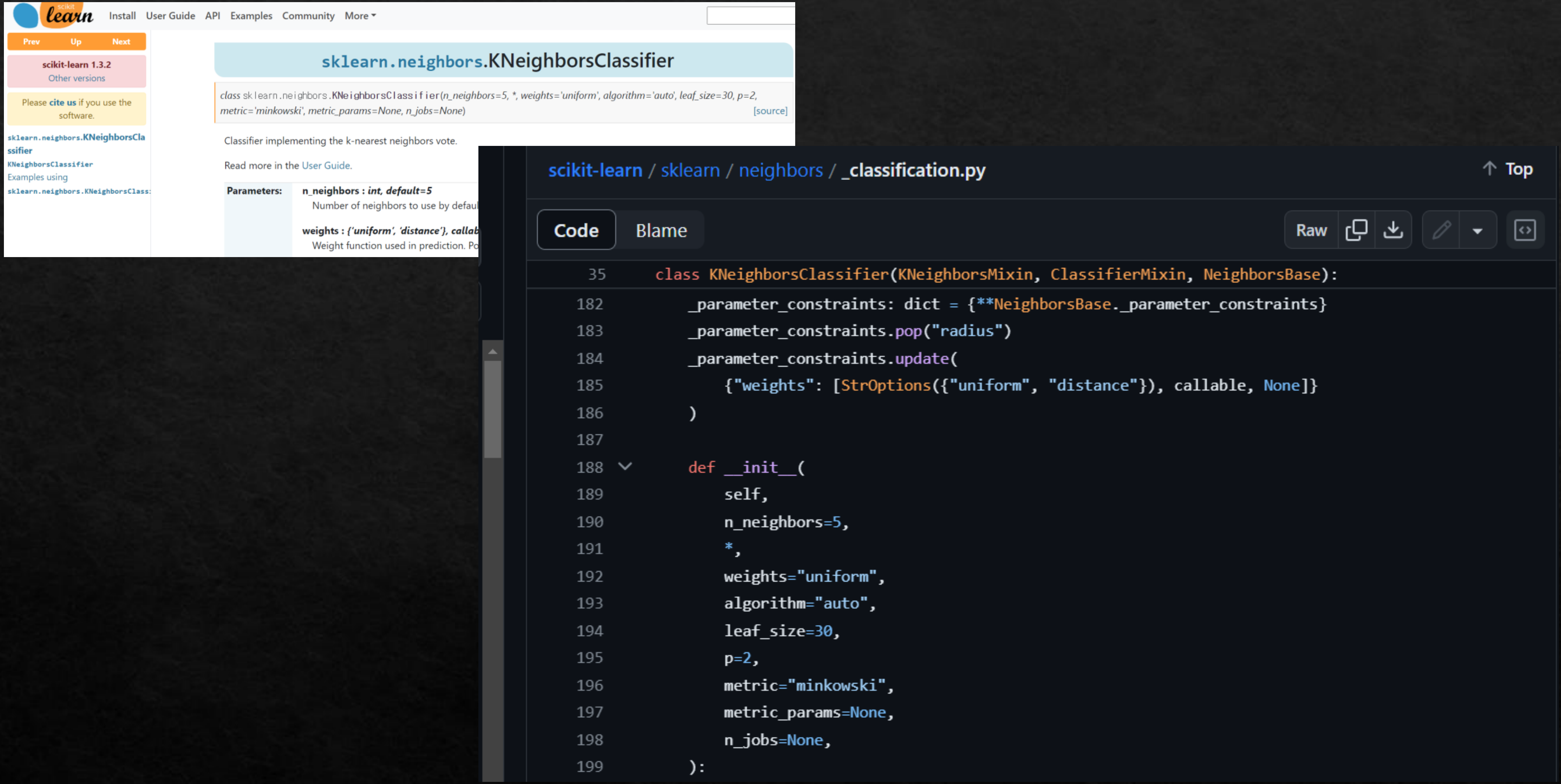
```
def _get_pictures():
    colab_directory = '/content/tmp/images'
    Path(colab_directory).mkdir(parents=True, exist_ok=True)

    with open("/content/tmp/launches.json") as f:
        launches = json.load(f)
        image_urls = [launch["image"] for launch in launches["results"]]
        for image_url in image_urls:
            try:
                response = requests.get(image_url)
                response.raise_for_status()
                image_filename = image_url.split("/")[-1]
                target_file = f"{colab_directory}/{image_filename}"
                with open(target_file, "wb") as file:
                    file.write(response.content)
                print(f"Downloaded {image_url} to {target_file}")
            except requests.exceptions.RequestException as e:
                print(f"Error downloading {image_url}: {e}")
```

```
# Call the function
_get_pictures()
```

Scikit-learn's Class parameters(1/2)

```
X = [[0], [1], [2], [3]]
y = [0, 0, 1, 1]
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(X, y)
print(neigh.predict([[1.1]]))
print(neigh.predict_proba([[0.9]]))

neigh.get_params()
#
{'algorithm': 'auto',
 'leaf_size': 30,
 'metric': 'minkowski',
 'metric_params': None,
 'n_jobs': None,
 'n_neighbors': 3,
 'p': 2,
 'weights': 'uniform'}
```

Scikit-learn's Class parameters(2/2)



scikit-learn 1.3.2
Other versions

Please **cite us** if you use the software.

sklearn.neighbors.KNeighborsClassifier
KNeighborsClassifier
Examples using
sklearn.neighbors.KNeighborsClass:

sklearn.neighbors.KNeighborsClassifier

class sklearn.neighbors.KNeighborsClassifier(*n_neighbors=5*, ***, *weights='uniform'*, *algorithm='auto'*, *leaf_size=30*, *p=2*, *metric='minkowski'*, *metric_params=None*, *n_jobs=None*) [source]

Classifier implementing the k-nearest neighbors vote.

Read more in the User Guide.

Parameters: **n_neighbors : *int, default=5***
Number of neighbors to use by defaul

weights : *{'uniform', 'distance'}, callab*
Weight function used in prediction. Po

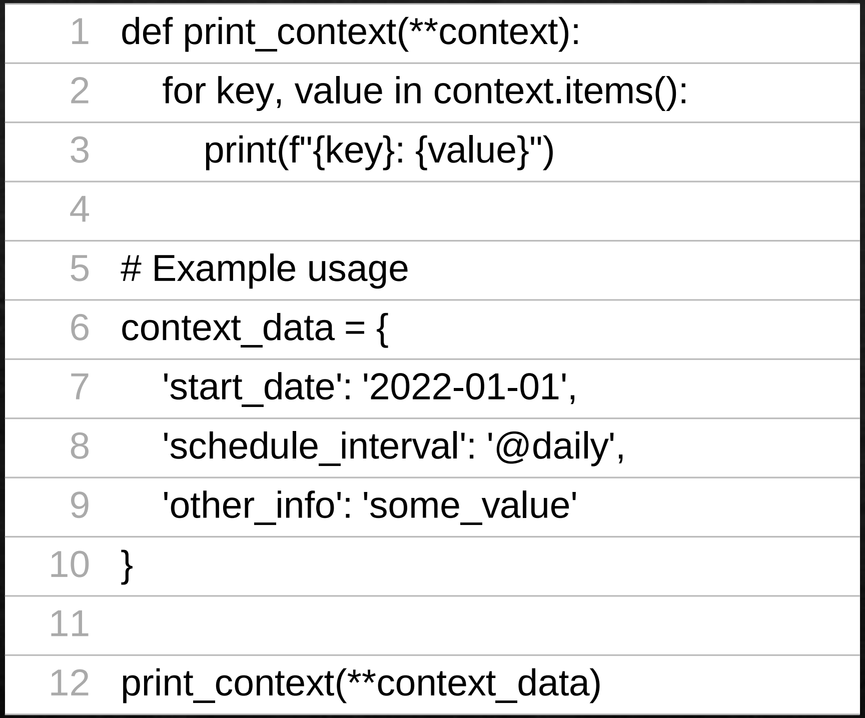
scikit-learn / sklearn / neighbors / _classification.py

```python
class KNeighborsClassifier(KNeighborsMixin, ClassifierMixin, NeighborsBase):
    _parameter_constraints: dict = {**NeighborsBase._parameter_constraints}
    _parameter_constraints.pop("radius")
    _parameter_constraints.update(
        {"weights": [StrOptions({"uniform", "distance"}), callable, None]}
    )

    def __init__(
        self,
        n_neighbors=5,
        *,
        weights="uniform",
        algorithm="auto",
        leaf_size=30,
        p=2,
        metric="minkowski",
        metric_params=None,
        n_jobs=None,
    ):
```

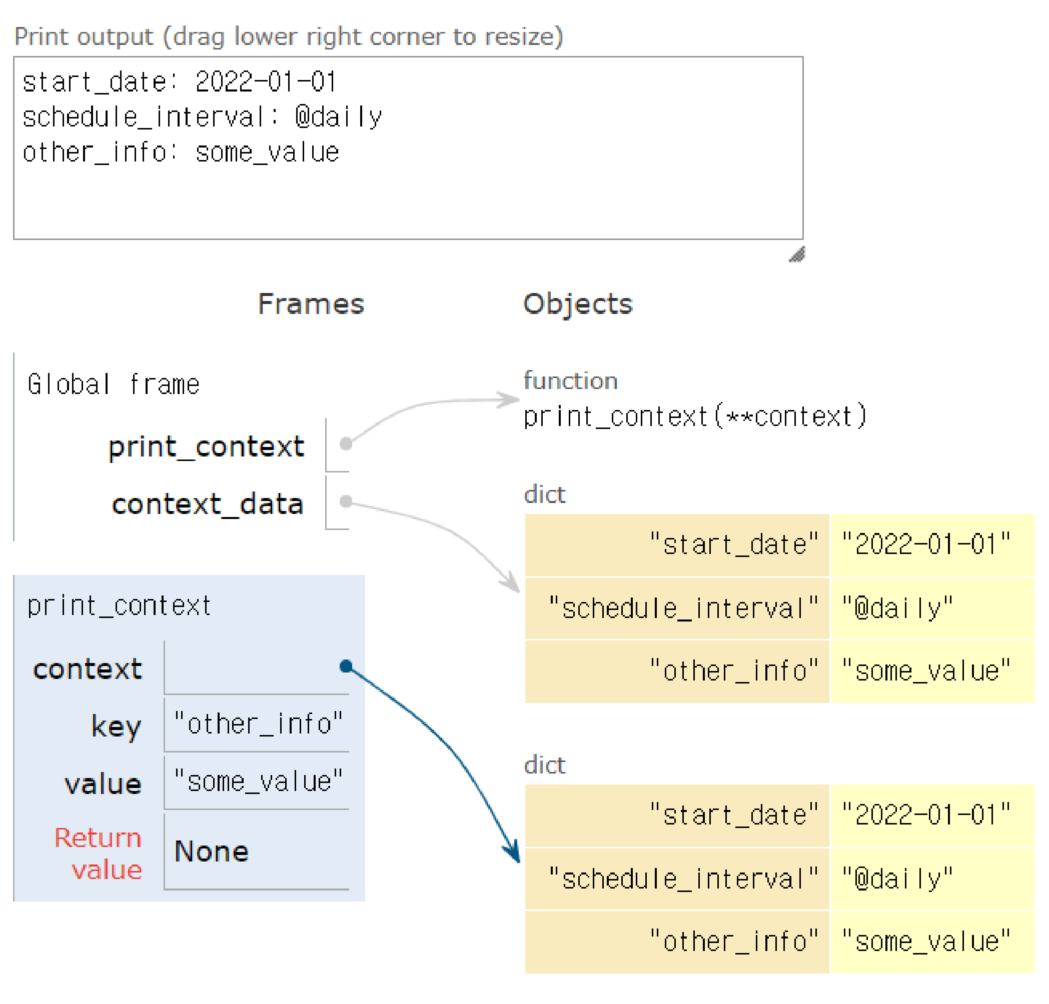
*arg, **kwargs

```
def greet(*name, **kwargs):
    print(f"{kwargs}, {name}!")

greet("Carol", "Sanggoo", "Breece", greet1='Hi', greet2='Bye’)

#
{'greet1': 'Hi', 'greet2': 'Bye'}, ('Carol', 'Sanggoo', 'Breece')!
```

**kwargs



```python
def print_context(**context):
    for key, value in context.items():
        print(f"{key}: {value}")

# Example usage
context_data = {
    'start_date': '2022-01-01',
    'schedule_interval': '@daily',
    'other_info': 'some_value'
}

print_context(**context_data)
```



Print output (drag lower right corner to resize)

```
start_date: 2022-01-01
schedule_interval: @daily
other_info: some_value
```

key	"other_info"
value	"some_value"
Return value	None

"start_date"	"2022-01-01"
"schedule_interval"	"@daily"
"other_info"	"some_value"

"start_date"	"2022-01-01"
"schedule_interval"	"@daily"
"other_info"	"some_value"

```
print_context(start_date= '2022-01-01',
              schedule_interval= '@daily',
              other_info= 'some_value’)

print_context(start_date= '2022-01-01',
              other_info= 'some_value’)
```

*args

```
def print_context(*context):
    print(context)

# Example usage
context_data = ['start_date', 'schedule_interval', 'other_info']

print_context(*context_data)
```

*args

```
class MyClass:
    def __init__(self, name, age, city):
        self.name = name
        self.age = age
        self.city = city

# Creating an instance of MyClass
obj = MyClass(name="John", age=25, city="New York")

vars(obj)
#
{'name': 'John', 'age': 25, 'city': 'New York'}
```

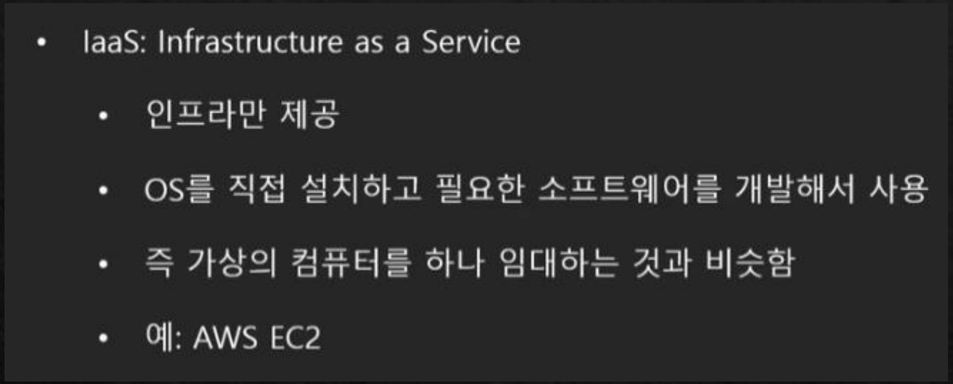
```
# Using the ** operator to pass attributes of the class as keyword arguments to the function
my_function(**vars(obj))
#
name: John age: 25 city: New York
```

- The vars() function in Python returns the __dict__ attribute of an object, which is a dictionary containing the object's attributes and their values.
- It's a built-in function that allows you to inspect the namespace of an object.

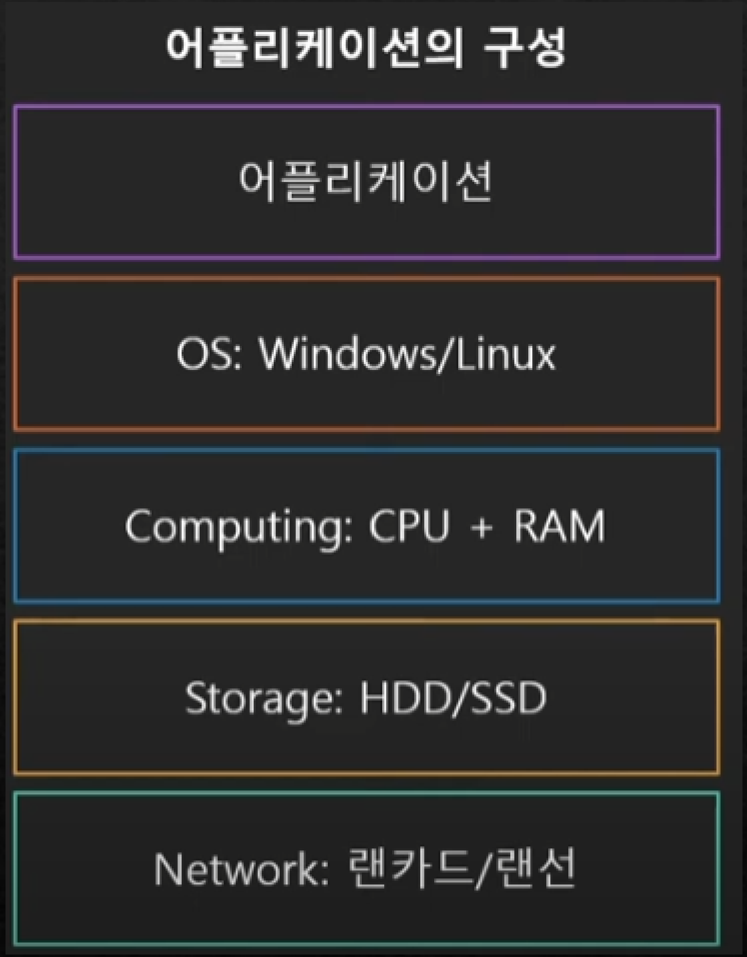
Bash shell script

```
Linux/Ubuntu$ curl -o /tmp/launches.json -L ’https://ll.thespacedevs.com/2.0.0/launch/upcoming’
```

클라우드서비스

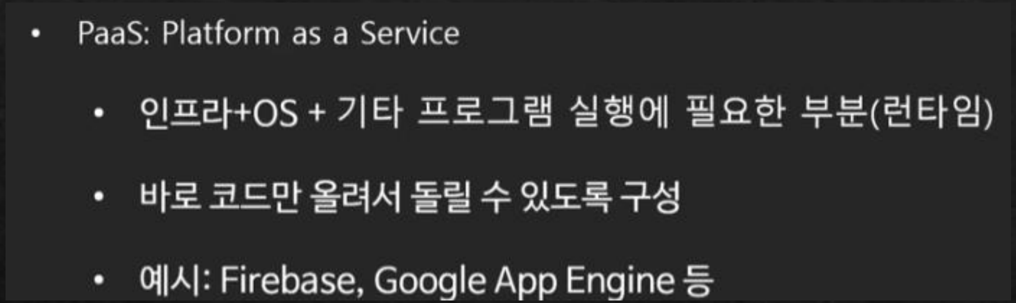


- IaaS: Infrastructure as a Service
  - 인프라만 제공
  - OS를 직접 설치하고 필요한 소프트웨어를 개발해서 사용
  - 즉 가상의 컴퓨터를 하나 임대하는 것과 비슷함
  - 예: AWS EC2

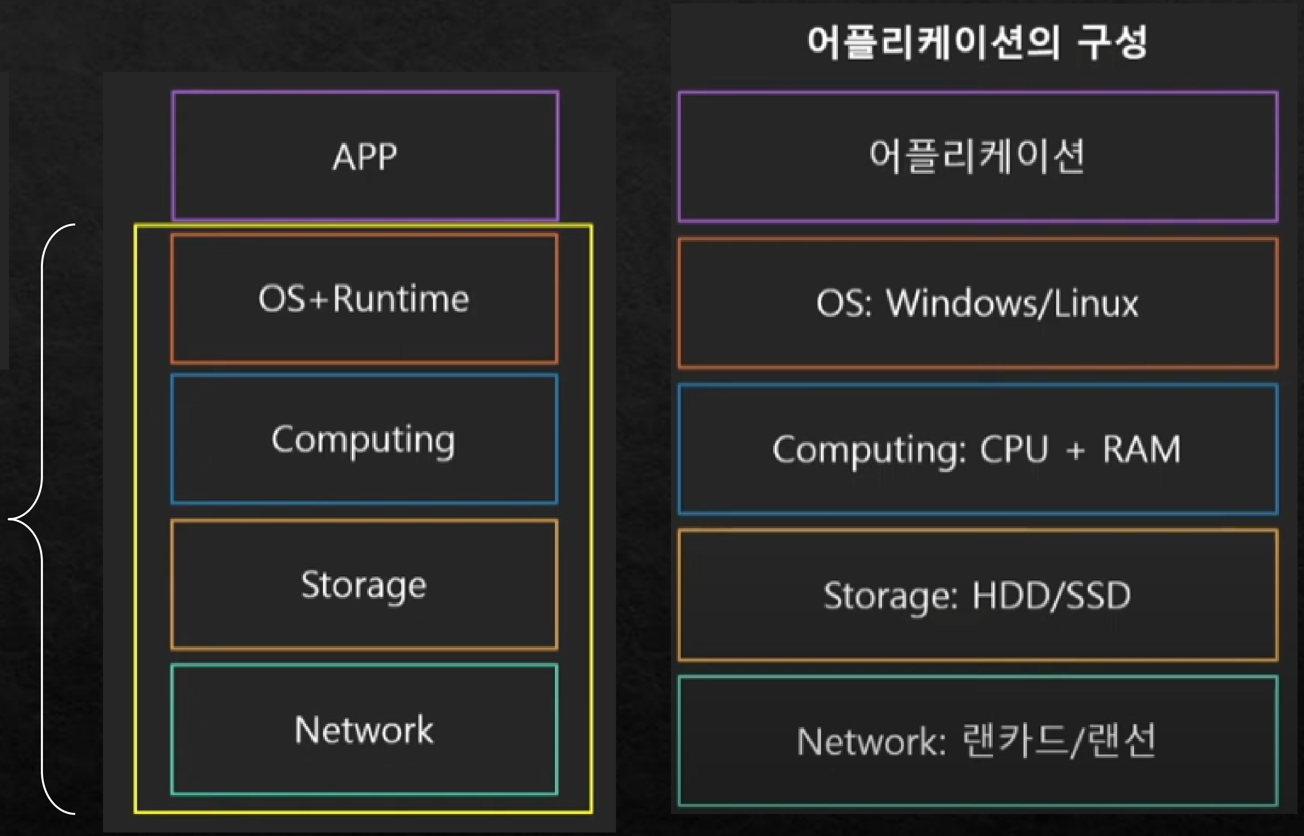


https://www.youtube.com/watch?v=s75iONF6XFw&list=PLfth0bK2MgIan-SzGpHIbfnCnjj583K2m&index=3

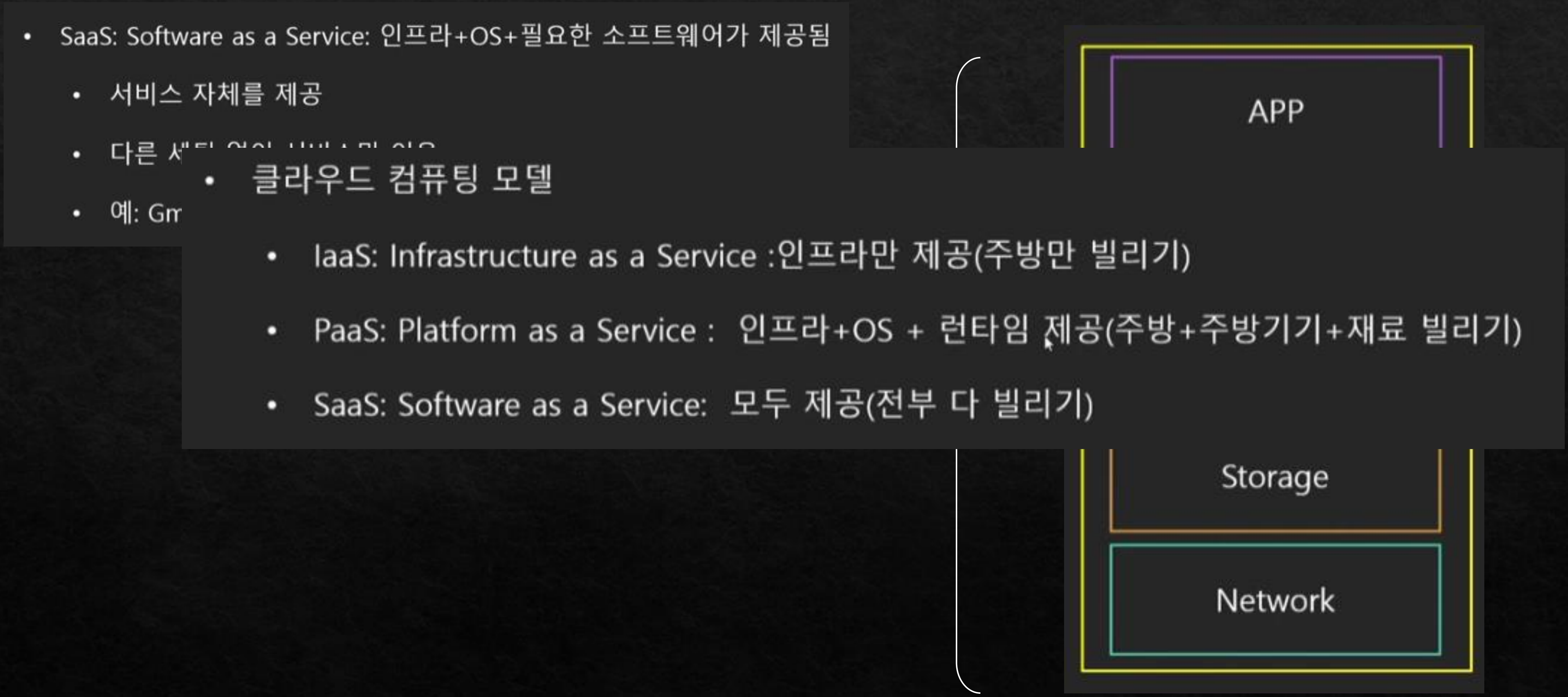
클라우드서비스



- PaaS: Platform as a Service
  - 인프라+OS + 기타 프로그램 실행에 필요한 부분(런타임)
  - 바로 코드만 올려서 돌릴 수 있도록 구성
  - 예시: Firebase, Google App Engine 등



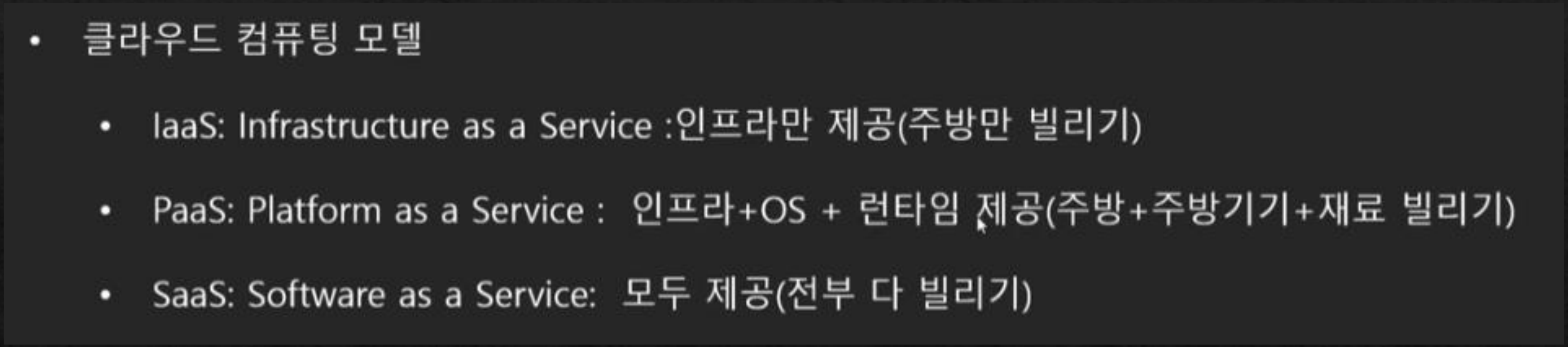
클라우드서비스



- SaaS: Software as a Service: 인프라+OS+필요한 소프트웨어가 제공됨
  - 서비스 자체를 제공
  - 다른 세
  - 예: Gm

- 클라우드 컴퓨팅 모델
  - IaaS: Infrastructure as a Service :인프라만 제공(주방만 빌리기)
  - PaaS: Platform as a Service : 인프라+OS + 런타임 제공(주방+주방기기+재료 빌리기)
  - SaaS: Software as a Service: 모두 제공(전부 다 빌리기)

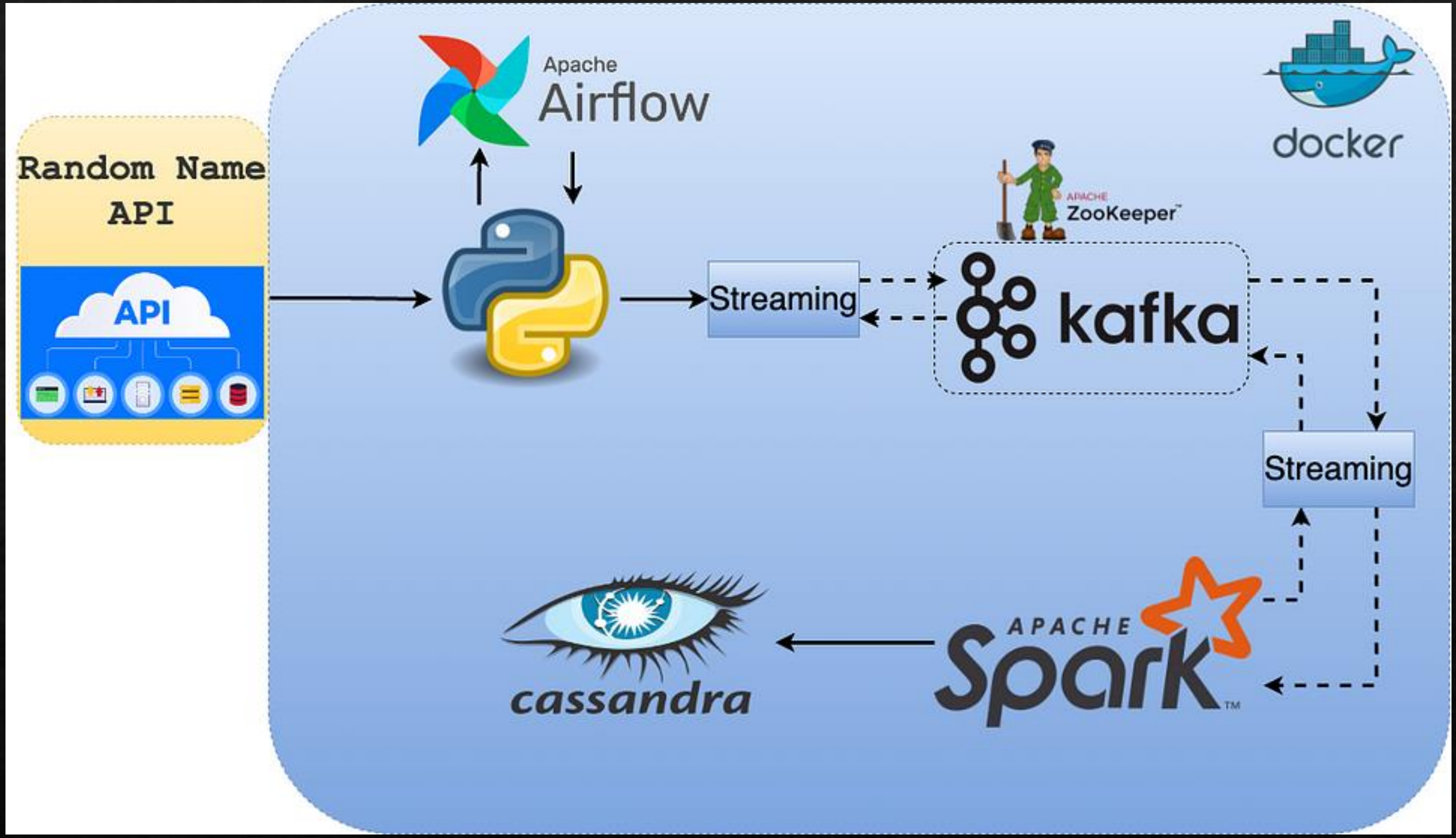
클라우드서비스



- 클라우드 컴퓨팅 모델
  - IaaS: Infrastructure as a Service :인프라만 제공(주방만 빌리기)
  - PaaS: Platform as a Service : 인프라+OS + 런타임 제공(주방+주방기기+재료 빌리기)
  - SaaS: Software as a Service: 모두 제공(전부 다 빌리기)

Spark, Kafka, Airflow, Docker, Cassandra, Python

- Data Engineering End-to-End Project



https://www.youtube.com/watch?v=0Ssx7jJJADI

data-engineering-end-to-end-project

data-engineering-end-end-project1/2 (with github)