

- 출처: LangChain 공식 문서 또는 해당 교재명
- 원본 URL: <https://smith.langchain.com/hub/teddynote/summary-stuff-documents>

▼ 8) 멀티모달 검색

- Chroma = 멀티모달 컬렉션 (여러 양식의 데이터를 포함하고 쿼리할 수 있는 컬렉션) 지원

▼ 9) 데이터 세트

- 허깅페이스에서 호스팅되는 데이터 사용
 - [coco object detection dataset](#)의 작은 하위 집합 사용
 - 데이터 세트의 모든 이미지 중 일부만 로컬로 다운로드 → 멀티모달 컬렉션 생성하기

```
import os
from datasets import load_dataset
from matplotlib import pyplot as plt

# COCO 데이터셋 로드
dataset = load_dataset(
    path="detection-datasets/coco",
    name="default",
    split="train",
    streaming=True
)

# 이미지 저장 폴더와 이미지 개수 설정
IMAGE_FOLDER = "../09_VectorStore/images/tmp"
N_IMAGES = 20

# 그래프 플로팅을 위한 설정
plot_cols = 5
plot_rows = N_IMAGES // plot_cols
fig, axes = plt.subplots(plot_rows, plot_cols, figsize=(plot_rows * 2, plot_cols * 2))
axes = axes.flatten()

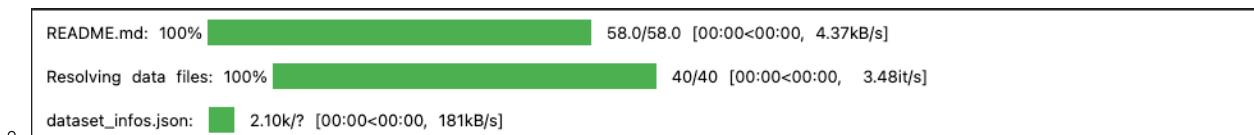
# 이미지를 폴더에 저장하고 그래프에 표시
dataset_iter = iter(dataset)
os.makedirs(IMAGE_FOLDER, exist_ok=True)
for i in range(N_IMAGES):
    # 데이터셋에서 이미지와 레이블 추출
    data = next(dataset_iter)
    image = data["image"]
    label = data["objects"][0]["category"] # 첫 번째 객체의 카테고리를 레이블로 사용

    # 그래프에 이미지 표시 및 레이블 추가
    axes[i].imshow(image)
    axes[i].set_title(label, fontsize=8)
    axes[i].axis("off")

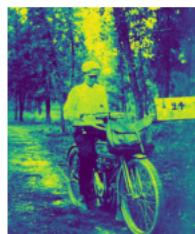
    # 이미지 파일로 저장
    image.save(f"{IMAGE_FOLDER}/{i}.jpg")

# 그래프 레이아웃 조정 및 표시
plt.tight_layout()
plt.show()
```

- 다운로드 과정



- 결과



o

▼ 10) Multimodal Embeddings

- **Multimodal Embeddings** → 이미지, 텍스트에 대한 Embedding 생성하기
- **OpenCLIPEmbeddingFunction** → 이미지 임베딩하기
- 사전에 **VS Code**에 설치할 것

```
pip install open-clip-torch
```

- 참고: [OpenCLIP](#)

- **Model** 벤치마크

- 모델 벤치마트

- 아래의 예시에서 **model_name**, **checkpoint** 설정 → 사용

- **model_name**: OpenCLIP 모델명
- **checkpoint**: OpenCLIP 모델의 Training data에 해당하는 이름

```
import open_clip
import pandas as pd
```

```
# 사용 가능한 모델/Checkpoint 를 출력  
pd.DataFrame(open_clip.list_pretrained(), columns=["model_name", "checkpoint"]).head(10)
```

- 셀 출력: 사용 가능한 모델, 체크포인트 출력

	model_name	checkpoint
0	RN50	openai
1	RN50	yfcc15m
2	RN50	cc12m
3	RN101	openai
4	RN101	yfcc15m
5	RN50x4	openai
6	RN50x16	openai
7	RN50x64	openai
8	ViT-B-32	openai
9	ViT-B-32	laion400m_e31

10 rows x 2 cols 10 per page < Page 1 of 1 > »

```
from langchain_experimental.open_clip import OpenCLIPEmbeddings  
  
# OpenCLIP 임베딩 함수 객체 생성  
image_embedding_function = OpenCLIPEmbeddings(  
    model_name="ViT-H-14-378-quickgelu",  
    checkpoint="dfn5b"  
)
```

- 셀 출력 (5m 45.8s)

✓ 5m 45.8s Python
open_clip_pytorch_model.bin: 21% 809M/3.95G [04:12<02:44, 19.1MB/s]

- 이미지 경로 = list로 저장하기

```
# 이미지의 경로를 리스트로 저장  
image_uris = sorted(  
    [  
        #os.path.join("tmp", image_name)  
        #for image_name in os.listdir("tmp")  
        os.path.join(IMAGE_FOLDER, image_name)  
        for image_name in os.listdir(IMAGE_FOLDER)  
        if image_name.endswith(".jpg")  
    ]  
)  
  
image_uris
```

- 셀 출력

```
[..../09_VectorStore/images/tmp/0.jpg',  
 ..../09_VectorStore/images/tmp/1.jpg',  
 ..../09_VectorStore/images/tmp/10.jpg',  
 ..../09_VectorStore/images/tmp/11.jpg',  
 ..../09_VectorStore/images/tmp/12.jpg',  
 ..../09_VectorStore/images/tmp/13.jpg',  
 ..../09_VectorStore/images/tmp/14.jpg',  
 ..../09_VectorStore/images/tmp/15.jpg',  
 ..../09_VectorStore/images/tmp/16.jpg',  
 ..../09_VectorStore/images/tmp/17.jpg',  
 ..../09_VectorStore/images/tmp/18.jpg',  
 ..../09_VectorStore/images/tmp/19.jpg',  
 ..../09_VectorStore/images/tmp/2.jpg',  
 ..../09_VectorStore/images/tmp/3.jpg',  
 ..../09_VectorStore/images/tmp/4.jpg',  
 ..../09_VectorStore/images/tmp/5.jpg',  
 ..../09_VectorStore/images/tmp/6.jpg',  
 ..../09_VectorStore/images/tmp/7.jpg',  
 ..../09_VectorStore/images/tmp/8.jpg',  
 ..../09_VectorStore/images/tmp/9.jpg']
```

```

# 이미지 임베딩 계산 (1m 1.7s 소요)
import numpy as np

img_vecs = image_embedding_function.embed_image(image_uris)           # list[list[float]]
img_vecs = np.array(img_vecs, dtype="float32")

# 간단한 인덱스(코사인 유사도) 준비
def cosine_sim(a, b):
    a = a / (np.linalg.norm(a, axis=-1, keepdims=True) + 1e-9)
    b = b / (np.linalg.norm(b, axis=-1, keepdims=True) + 1e-9)
    return a @ b.T

# 이미지 경로와 벡터를 함께 보관
image_index = {"uris": image_uris, "vectors": img_vecs}

# 텍스트 쿼리로 관련 이미지 top-k 검색
query = "a red vintage car on the street"
q_vec = np.array(
    image_embedding_function.embed_query(query),      # 텍스트 임베딩
    dtype="float32",
)[None, :]                                         # (1, d)

sims = cosine_sim(q_vec, image_index["vectors"])[0]  # (num_images,)
top_k = 3
top_idx = sims.argsort()[:-1][:top_k]
top_images = [image_index["uris"][i] for i in top_idx]
print(top_images)

```

- 셀 출력 (4.3s)(k=3)

```
[ '../09_VectorStore/images/tmp/11.jpg', '../09_VectorStore/images/tmp/19.jpg', '../09_VectorStore/images/tmp/8.jpg' ]
```

```

# 이미지로 이미지 검색도 가능 (특정 쿼리 이미지의 임베딩을 뽑아 상위 유사 이미지 검색도 가능)
query_image = image_uris[0]
q_img_vec = np.array(image_embedding_function.embed_image([query_image])[0])[None, :]
sims = cosine_sim(q_img_vec, image_index["vectors"])[0]
top_idx = sims.argsort()[:-1][:top_k]
top_images = [image_index["uris"][i] for i in top_idx]
print(top_images)

```

- 셀 출력 (4.9s) (k=3)

```
[ '../09_VectorStore/images/tmp/0.jpg', '../09_VectorStore/images/tmp/1.jpg', '../09_VectorStore/images/tmp/15.jpg' ]
```

- 상위 이미지들을 gemini로 보내 캡션 생성하기

- 전달 방법: base64 data URI 로 전달 or 공개 URL
- 로컬 파일: base64 data URI 권장

```

import base64, mimetypes
from langchain_core.messages import SystemMessage, HumanMessage
from langchain_google_genai import ChatGoogleGenerativeAI
from google import genai
import os
import json

# Gemini 모델 초기화 (멀티모달 지원)
llm = ChatGoogleGenerativeAI(model="gemini-2.5-flash-lite")

```

- 디버깅 과정

```

# 진단 스니펫
def to_data_uri(path: str) -> str:
    mime = mimetypes.guess_type(path)[0] or "image/jpeg"
    with open(path, "rb") as f:
        b64 = base64.b64encode(f.read()).decode("utf-8")
    # 반드시 접두부 포함

```

```
return f"\{mime\};base64,\{b64\}"\n\n# 3) 프롬프트 구성 (시스템 + 사용자 제약)\nsystem_msg = SystemMessage(content="Your mission is to describe the image in detail")\nuser_instruction = "Description should be written in one sentence (less than 60 characters)."\n\n# 1) data URI 생성\ntest_path = image_uris[0]\nuri = to_data_uri(test_path)\nprint("HEAD:", uri[:40])           # 기대: image/jpeg;base64,\nprint("HAS_DATA_PREFIX:", uri.startswith("image/"))\n\n# 2) HumanMessage 블록 구성 후 다시 확인\nblocks = [{"type": "text", "text": "Describe in one sentence (<60 chars.)"}]\nblocks.append({"type": "image_url", "image_url": uri})\nprint("BLOCK_IMAGE_URL_HEAD:", blocks[1]["image_url"][:40])\n\n# 3) 최종 invoke 직전 상태 점검\nmsgs = [system_msg, HumanMessage(content=blocks)]\nprint("MSG_OK:", isinstance(msgs[1].content, list), msgs[1].content[1]["type"])\nresp = llm.invoke(msgs)\nprint(resp.content)
```

- 셀 출력

HEAD: image/jpeg;base64,/9j/4AAQSkZJRgABAQAAAQ
HAS_DATA_PREFIX: True
BLOCK_IMAGE_URL_HEAD: image/jpeg;base64,/9j/4AAQSkZJRgABAQAAAQ
MSG_OK: True image_url

```
{  
  "name": "ValueError",  
  "message": "Image string must be one of: b64 encoded image string (data:image/...) or valid image url. Instead  
  "stack": "\u001b[31m-----\u001b[39m\u001b[31m\n\u001b[39m"
```

```
# 진단 스니펫2
def to_data_uri(path: str) -> str:
    mime = mimetypes.guess_type(path)[0] or "image/jpeg"
    with open(path, "rb") as f:
        b64 = base64.b64encode(f.read()).decode("utf-8")
    # 반드시 '' 접두부 포함
    return f"{mime};base64,{b64}"

# 기대 출력값 출력해보기
uri = to_data_uri(test_path)
print("HEAD:", uri[:40])           # 기대: image/jpeg;base64,
print("HAS DATA PREFIX:", uri.startswith("image/"))
```

- 셀 출력

HEAD: image/jpeg;base64,/9j/4AAQSkZJRgABAQAAAQ
HAS_DATA_PREFIX: True

```
# 진단 스니펫3
def to_data_uri(path: str) -> str:
    mime = mimetypes.guess_type(path)[0] or "image/jpeg"
    with open(path, "rb") as f:
        b64 = base64.b64encode(f.read()).decode("utf-8")
    return f"{mime};base64,{b64}" # ← 접두부 필수

# 3) 프롬프트 구성 (시스템 + 사용자 제약)
system_msg = SystemMessage(content="Your mission is to describe the image in detail")
user_instruction = "Description should be written in one sentence (less than 60 characters)."

# 1) data URI 생성
test_path = image_uris[0]
uri = to_data_uri(test_path)
print("HEAD:", uri[:40])           # 기대: image/jpeg;base64,
print("HAS_DATA_PREFIX:", uri.startswith("image/"))

# 2) HumanMessage 블록 구성 후 다시 확인
blocks = [{"type": "text", "text": "Describe in one sentence (<60 chars.)"}]
blocks.append({"type": "image_url", "image_url": uri})
print("BLOCK_IMAGE_URL_HEAD:", blocks[1]["image_url"][:40])

# 3) 최종 invoke 직전 상태 점검
```

```
msgs = [system_msg, HumanMessage(content=blocks)]
print("MSG_OK:", isinstance(msgs[1].content, list), msgs[1].content[1]["type"])
```

- 셀 출력

```
HEAD: image/jpeg;base64,/9j/4AAQSkZJRgABAQAAAQ
HAS_DATA_PREFIX: True
BLOCK_IMAGE_URL_HEAD: image/jpeg;base64,/9j/4AAQSkZJRgABAQAAAQ
MSG_OK: True image_url
```

```
# 진단 스니펫4
```

```
import base64
import mimetypes
from langchain_core.messages import SystemMessage, HumanMessage
from langchain_google_genai import ChatGoogleGenerativeAI

llm = ChatGoogleGenerativeAI(model="gemini-1.5-flash")

def to_data_uri(path: str) -> str:
    mime = mimetypes.guess_type(path)[0] or "image/jpeg"
    with open(path, "rb") as f:
        b64 = base64.b64encode(f.read()).decode("utf-8")
    # 여기서 반드시 "" 접두부 포함해야 오류 안 납니다
    return f"{mime};base64,{b64}"

system_msg = SystemMessage(content="Your mission is to describe the image in detail")
user_instruction = "Description should be written in one sentence (less than 60 characters)."

# 이미지 경로(예시)
test_path = image_uris[0]
uri = to_data_uri(test_path)

# 포맷 확인 출력 (반드시 접두부 포함인지 꼭 확인할 것)
print("HEAD:", uri[:40]) # 기대: image/jpeg;base64,
print("HAS_DATA_PREFIX:", uri.startswith("image/")) # True 여야 함

blocks = [
    {"type": "text", "text": user_instruction + " Please describe this image."},
    {"type": "image_url", "image_url": uri}
]

print("BLOCK_IMAGE_URL_HEAD:", blocks[1]["image_url"][:40])

msgs = [system_msg, HumanMessage(content=blocks)]
print("MSG_OK:", isinstance(msgs[1].content, list), msgs[1].content[1]["type"])

#resp = llm.invoke(msgs)
#print(resp.content)
```

- 셀 출력

```
HEAD: image/jpeg;base64,/9j/4AAQSkZJRgABAQAAAQ
HAS_DATA_PREFIX: True
BLOCK_IMAGE_URL_HEAD: image/jpeg;base64,/9j/4AAQSkZJRgABAQAAAQ
MSG_OK: True image_url
```

```
# 진단 스니펫5
```

```
import base64, mimetypes
from langchain_core.messages import SystemMessage, HumanMessage
from langchain_google_genai import ChatGoogleGenerativeAI

llm = ChatGoogleGenerativeAI(model="gemini-1.5-flash")

def to_data_uri(path: str) -> str:
    mime = mimetypes.guess_type(path)[0] or "image/jpeg"
    with open(path, "rb") as f:
        b64 = base64.b64encode(f.read()).decode("utf-8")
    # 반드시 접두부 포함
    return f"{mime};base64,{b64}"

system_msg = SystemMessage(content="Your mission is to describe the image in detail")
user_instruction = "Description should be written in one sentence (less than 60 characters)."

test_path = image_uris[0] # 실제 이미지 경로
uri = to_data_uri(test_path)
```

```

# 접두부 포함 여부 정확 검증
print("HEAD:", uri[:40]) # => image/jpeg;base64,
print("HAS_DATA_PREFIX:", uri.startswith("image/")) # => True

blocks = [
    {"type": "text", "text": f"{user_instruction} Please describe this image."},
    {"type": "image_url", "image_url": uri},
]

msgs = [system_msg, HumanMessage(content=blocks)]
#resp = llm.invoke(msgs)
#print(resp.content)
print(msgs)

```

- 셀 출력

```

HEAD: image/jpeg;base64,/9j/4AAQSkZJRgABAQAAAQ
HAS_DATA_PREFIX: True

```

```
[SystemMessage(content='Your mission is to describe the image in detail', additional_kwargs={}, response_metadata={})]
```

```

import base64, mimetypes, os
from langchain_core.messages import SystemMessage, HumanMessage
from langchain_google_genai import ChatGoogleGenerativeAI

# ----- 설정 -----
IMAGE_DIR = "../09_VectorStore/images/tmp"
image_uris = sorted(
    [os.path.join(IMAGE_DIR, f) for f in os.listdir(IMAGE_DIR) if f.endswith(".jpg")]
)
model_name = "gemini-1.5-flash"
# -----


llm = ChatGoogleGenerativeAI(model=model_name)

def to_data_uri(path: str) -> str:
    """로컬 이미지 → data URI (RFC2397)"""
    mime = mimetypes.guess_type(path)[0] or "image/jpeg"
    with open(path, "rb") as f:
        b64 = base64.b64encode(f.read()).decode("utf-8")
    # 반드시 스kip 포함
    return f"{mime};base64,{b64}"

# ① 테스트 이미지 1장
test_path = image_uris[0] # 첫 번째 파일
uri = to_data_uri(test_path)

# ② 검증 로그
print("HEAD:", uri[:40]) # image/jpeg;base64, ...
print("HAS_DATA_PREFIX:", uri.startswith("image/")) # True 여야 함

# ③ Gemini 메시지 구성
blocks = [
    {"type": "text", "text": "Description (≤60 chars). Please describe this image."},
    {"type": "image_url", "image_url": uri},
]

msgs = [
    SystemMessage(content="Your mission is to describe the image in detail"),
    HumanMessage(content=blocks),
]

# ④ 호출
resp = llm.invoke(msgs)
print(resp.content)

```

- 셀 출력

```

HEAD: image/jpeg;base64,/9j/4AAQSkZJRgABAQAAAQ
HAS_DATA_PREFIX: True

```

```
{
    "name": "ValueError",
```

```
import os, base64, mimetypes, pprint
from langchain_core.messages import SystemMessage, HumanMessage
from langchain_google_genai import ChatGoogleGenerativeAI

# ----- 사용자 설정 -----
IMAGE_DIR = "../09_VectorStore/images/tmp"    # 이미지 폴더
MODEL      = "gemini-1.5-flash"                 # 사용 모델
# ----- 

def to_data_uri(path: str) -> str:
    """로컬 파일을 RFC 2397 data URI로 변환"""
    mime = mimetypes.guess_type(path)[0] or "application/octet-stream"
    with open(path, "rb") as f:
        b64 = base64.b64encode(f.read()).decode("utf-8")
    return f"{mime};base64,{b64}"

# 변환 + 검증
image_paths = sorted(
    [os.path.join(IMAGE_DIR, f) for f in os.listdir(IMAGE_DIR) if f.endswith(".jpg")]
)
first_uri = to_data_uri(image_paths[0])

# 필수 검증 – 반드시 True
assert first_uri.startswith("image/"), "스킵이 없습니다!"

# 로그 확인
print("HEAD:", first_uri[:40])

# Gemini 메시지
blocks = [
    {"type": "text",      "text": "Describe this image in one sentence."},
    {"type": "image_url", "image_url": first_uri},
]
msgs = [
    SystemMessage(content="You are a helpful assistant."),
    HumanMessage(content=blocks),
]
]

# 호출
llm = ChatGoogleGenerativeAI(model=MODEL)
resp = llm.invoke(msgs)
pprint.pprint(resp.content)
```

- 셀 출력

HEAD: image/jpeg;base64,/9j/4AAQSkZJRGqABAQAAAQ

```
def debug_data_uri_issue():
    import inspect

    # 1. 현재 실행중인 함수들 모두 찾기
    for name, obj in globals().items():
        if 'data' in name.lower() and callable(obj):
            print(f"함수 발견: {name}")
            try:
                print(f"소스: {inspect.getsource(obj)}")
            except:
                pass

    # 2. 단계별 값 추적
    path = image_uris[0]  # 첫 번째 이미지

    print(f"1. 원본 경로: {path}")

    # 실제 함수 호출하면서 중간값들 모두 출력
    mime = mimetypes.guess_type(path)[0] or "image/jpeg"
    print(f"2. MIME 타입: {mime}")

    with open(path, "rb") as f:
        b64_data = base64.b64encode(f.read()).decode("utf-8")
```

```

print(f"3. base64 길이: {len(b64_data)}")
print(f"4. base64 앞 50자: {b64_data[:50]}")

# 여기가 핵심!
uri_without_data = f"{mime};base64,{b64_data[:50]}..."
uri_with_data = f"{mime};base64,{b64_data[:50]}..."

print(f"5. 없을 때: {uri_without_data}")
print(f"6. 있을 때: {uri_with_data}")

return f"{mime};base64,{b64_data}"

# 실행해보기
final_uri = debug_data_uri_issue()

```

- 셀 출력

```

함수 발견: load_dataset
소스: def load_dataset(
    path: str,
    name: Optional[str] = None,
    data_dir: Optional[str] = None,
    data_files: Optional[Union[str, Sequence[str], Mapping[str, Union[str, Sequence[str]]]]] = None,
    split: Optional[Union[str, Split, list[str], list[Split]]] = None,
    cache_dir: Optional[str] = None,
    features: Optional[Features] = None,
    download_config: Optional[DownloadConfig] = None,
    download_mode: Optional[Union[DownloadMode, str]] = None,
    verification_mode: Optional[Union[VerificationMode, str]] = None,
    keep_in_memory: Optional[bool] = None,
    save_infos: bool = False,
    revision: Optional[Union[str, Version]] = None,
    token: Optional[Union[bool, str]] = None,
    streaming: bool = False,
    num_proc: Optional[int] = None,
    storage_options: Optional[dict] = None,
    **config_kwargs,
) -> Union[DatasetDict, Dataset, IterableDatasetDict, IterableDataset]:
    """Load a dataset from the Hugging Face Hub, or a local dataset.

    You can find the list of datasets on the [Hub](https://huggingface.co/datasets) or with [`huggingface_hub.list_dsets()`]. A dataset is a directory that contains some data files in generic formats (JSON, CSV, Parquet, etc.) and possibly follows a generic structure (Webdataset, ImageFolder, AudioFolder, VideoFolder, etc.). This function does the following under the hood:
    1. Load a dataset builder:
        * Find the most common data format in the dataset and pick its associated builder (JSON, CSV, Parquet, etc.)
        * Find which file goes into which split (e.g. train/test) based on file and directory names or on the `data_files` argument.
        * It is also possible to specify `data_files` manually, and which dataset builder to use (e.g. "parquet").
    2. Run the dataset builder:
        In the general case:
        * Download the data files from the dataset if they are not already available locally or cached.
        * Process and cache the dataset in typed Arrow tables for caching.
        Arrow table are arbitrarily long, typed tables which can store nested objects and be mapped to numpy arrays. They can be directly accessed from disk, loaded in RAM or even streamed over the web.
        In the streaming case:
        * Don't download or cache anything. Instead, the dataset is lazily loaded and will be streamed on-the-fly.
    3. Return a dataset built from the requested splits in `split` (default: all).
Args:

```

```

path (`str`):
    Path or name of the dataset.

    - if `path` is a dataset repository on the HF hub (list all available datasets with [`huggingface_hub`])
    -> load the dataset from supported files in the repository (csv, json, parquet, etc.)
    e.g. `username/dataset_name`, a dataset repository on the HF hub containing the data files.

    - if `path` is a local directory
    -> load the dataset from supported files in the directory (csv, json, parquet, etc.)
    e.g. `./path/to/directory/with/my/csv/data`.

    - if `path` is the name of a dataset builder and `data_files` or `data_dir` is specified
      (available builders are "json", "csv", "parquet", "arrow", "text", "xml", "webdataset", "imagefolder")
    -> load the dataset from the files in `data_files` or `data_dir`
    e.g. `parquet`.

name (`str`, *optional*):
    Defining the name of the dataset configuration.

data_dir (`str`, *optional*):
    Defining the `data_dir` of the dataset configuration. If specified for the generic builders (csv, tex
    the behavior is equal to passing `os.path.join(data_dir, **)` as `data_files` to reference all the fi
data_files (`str` or `Sequence` or `Mapping`, *optional*):
    Path(s) to source data file(s).

split (`Split` or `str`):
    Which split of the data to load.
    If `None`, will return a `dict` with all splits (typically `datasets.Split.TRAIN` and `datasets.Split
    If given, will return a single Dataset.
    Splits can be combined and specified like in tensorflow-datasets.

cache_dir (`str`, *optional*):
    Directory to read/write data. Defaults to `"/.cache/huggingface/datasets"`.
features (`Features`, *optional*):
    Set the features type to use for this dataset.

download_config ([`DownloadConfig`], *optional*):
    Specific download configuration parameters.

download_mode ([`DownloadMode`] or `str`, defaults to `REUSE_DATASET_IF_EXISTS`):
    Download/generate mode.

verification_mode ([`VerificationMode`] or `str`, defaults to `BASIC_CHECKS`):
    Verification mode determining the checks to run on the downloaded/processed dataset information (che

    <Added version="2.9.1"/>

keep_in_memory (`bool`, defaults to `None`):
    Whether to copy the dataset in-memory. If `None`, the dataset
    will not be copied in-memory unless explicitly enabled by setting `datasets.config.IN_MEMORY_MAX_SIZE
    nonzero. See more details in the [improve performance](../cache#improve-performance) section.

revision ([`Version`] or `str`, *optional*):
    Version of the dataset to load.
    As datasets have their own git repository on the Datasets Hub, the default version "main" corresponds
    You can specify a different version than the default "main" by using a commit SHA or a git tag of the
    token (`str` or `bool`, *optional*):
    Optional string or boolean to use as Bearer token for remote files on the Datasets Hub.
    If `True`, or not specified, will get token from `"/.huggingface"`.

streaming (`bool`, defaults to `False`):
    If set to `True`, don't download the data files. Instead, it streams the data progressively while
    iterating on the dataset. An [`IterableDataset`] or [`IterableDatasetDict`] is returned instead in th

    Note that streaming works for datasets that use data formats that support being iterated over like tx
    Json files may be downloaded completely. Also streaming from remote zip or gzip files is supported bu
    like rar and xz are not yet supported. The tgz format doesn't allow streaming.

num_proc (`int`, *optional*, defaults to `None`):
    Number of processes when downloading and generating the dataset locally.
    Multiprocessing is disabled by default.

    <Added version="2.7.0"/>

storage_options (`dict`, *optional*, defaults to `None`):
    **Experimental**. Key/value pairs to be passed on to the dataset file-system backend, if any.

    <Added version="2.11.0"/>

**config_kwargs (additional keyword arguments):
    Keyword arguments to be passed to the `BuilderConfig`
    and used in the [`DatasetBuilder`].

```

Returns:

```
[`Dataset`] or [`DatasetDict`]:
- if `split` is not `None`: the dataset requested,
- if `split` is `None`, a [`~datasets.DatasetDict`] with each split.

or [`IterableDataset`] or [`IterableDatasetDict`]: if `streaming=True`

- if `split` is not `None`, the dataset is requested
- if `split` is `None`, a [`~datasets.streaming.IterableDatasetDict`] with each split.
```

Example:

Load a dataset from the Hugging Face Hub:

```
```py
>>> from datasets import load_dataset
>>> ds = load_dataset('cornell-movie-review-data/rotten_tomatoes', split='train')

Load a subset or dataset configuration (here 'sst2')
>>> from datasets import load_dataset
>>> ds = load_dataset('nyu-mll/glue', 'sst2', split='train')

Manual mapping of data files to splits
>>> data_files = {'train': 'train.csv', 'test': 'test.csv'}
>>> ds = load_dataset('namespace/your_dataset_name', data_files=data_files)

Manual selection of a directory to load
>>> ds = load_dataset('namespace/your_dataset_name', data_dir='folder_name')
```

```

Load a local dataset:

```
```py
Load a CSV file
>>> from datasets import load_dataset
>>> ds = load_dataset('csv', data_files='path/to/local/my_dataset.csv')

Load a JSON file
>>> from datasets import load_dataset
>>> ds = load_dataset('json', data_files='path/to/local/my_dataset.json')
```

```

Load an [`~datasets.IterableDataset`]:

```
```py
>>> from datasets import load_dataset
>>> ds = load_dataset('cornell-movie-review-data/rotten_tomatoes', split='train', streaming=True)
```

```

Load an image dataset with the `ImageFolder` dataset builder:

```
```py
>>> from datasets import load_dataset
>>> ds = load_dataset('imagefolder', data_dir='/path/to/images', split='train')
```
"""

if "trust_remote_code" in config_kwargs:
    if config_kwargs.pop("trust_remote_code"):
        logger.error(
            "'trust_remote_code' is not supported anymore.\n"
            f"Please check that the Hugging Face dataset '{path}' isn't based on a loading script and remove\n"
            "If the dataset is based on a loading script, please ask the dataset author to remove it and conv"
        )
if data_files is not None and not data_files:
    raise ValueError(f"Empty 'data_files': '{data_files}'. It should be either non-empty or None (default).")
if Path(path, config.DATASET_STATE_JSON_FILENAME).exists():
    raise ValueError(
        "You are trying to load a dataset that was saved using `save_to_disk` . "
        "Please use `load_from_disk` instead."
    )
if streaming and num_proc is not None:
    raise NotImplementedError()

```

```

    "Loading a streaming dataset in parallel with `num_proc` is not implemented. "
    "To parallelize streaming, you can wrap the dataset with a PyTorch DataLoader using `num_workers` > 1
)

download_mode = DownloadMode(download_mode or DownloadMode.REUSE_DATASET_IF_EXISTS)
verification_mode = VerificationMode(
    (verification_mode or VerificationMode.BASIC_CHECKS) if not save_infos else VerificationMode.ALL_CHECKS
)

# Create a dataset builder
builder_instance = load_dataset_builder(
    path=path,
    name=name,
    data_dir=data_dir,
    data_files=data_files,
    cache_dir=cache_dir,
    features=features,
    download_config=download_config,
    download_mode=download_mode,
    revision=revision,
    token=token,
    storage_options=storage_options,
    **config_kwargs,
)

# Return iterable dataset in case of streaming
if streaming:
    return builder_instance.as_streaming_dataset(split=split)

# Download and prepare data
builder_instance.download_and_prepare(
    download_config=download_config,
    download_mode=download_mode,
    verification_mode=verification_mode,
    num_proc=num_proc,
    storage_options=storage_options,
)

# Build dataset for splits
keep_in_memory =
    keep_in_memory if keep_in_memory is not None else is_small_dataset(builder_instance.info.dataset_size)
)
ds = builder_instance.as_dataset(split=split, verification_mode=verification_mode, in_memory=keep_in_memory)

return ds

```

함수 발견: to_data_uri
소스: def to_data_uri(path: str) -> str:
 """로컬 파일을 RFC 2397 data URI로 변환"""
 mime = mimetypes.guess_type(path)[0] or "application/octet-stream"
 with open(path, "rb") as f:
 b64 = base64.b64encode(f.read()).decode("utf-8")
 return f"{mime};base64,{b64}"

함수 발견: debug_data_uri_issue
소스: def debug_data_uri_issue():
 import inspect

 # 1. 현재 실행중인 함수들 모두 찾기
 for name, obj in globals().items():
 if 'data' in name.lower() and callable(obj):
 print(f"함수 발견: {name}")
 try:
 print(f"소스: {inspect.getsource(obj)}")
 except:
 pass

 # 2. 단계별 값 추적
 path = image_uris[0] # 첫 번째 이미지

 print(f"1. 원본 경로: {path}")

```
# 실제 함수 호출하면서 중간값들 모두 출력
mime = mimetypes.guess_type(path)[0] or "image/jpeg"
print(f"2. MIME 타입: {mime}")

with open(path, "rb") as f:
    b64_data = base64.b64encode(f.read()).decode("utf-8")

    print(f"3. base64 길이: {len(b64_data)}")
    print(f"4. base64 앞 50자: {b64_data[:50]}")

# 여기가 핵심!
uri_without_data = f"{mime};base64,{b64_data[:50]}..."
uri_with_data = f"{mime};base64,{b64_data[:50]}..."

print(f"5. 없을 때: {uri_without_data}")
print(f"6. 있을 때: {uri_with_data}")

return f"{mime};base64,{b64_data}"
```

- 원본 경로: [./09_VectorStore/images/tmp/0.jpg](#)
 - MIME 타입: image/jpeg
 - base64 길이: 83492
 - base64 앞 50자: [/9j/4AAQSkZJRgABAQAAAQABAAAD/2wBDAAgGBgcGB0gHBwcJCQ...](#)
 - 없을 때: image/jpeg;base64, /9j/4AAQSkZJRgABAQAAAQABAAAD/2wBDAAgGBgcGBQgHBwcJCQ...
 - 있을 때: image/jpeg;base64, /9j/4AAQSkZJRgABAQAAAQABAAAD/2wBDAAgGBgcGBQgHBwcJCQ...

함수 재정의

```
def to_data_uri(path: str) -> str:
    """로컬 파일을 RFC 2397 data URI로 변환"""
    mime = mimetypes.guess_type(path)[0] or "application/octet-stream"
    with open(path, "rb") as f:
        b64 = base64.b64encode(f.read()).decode("utf-8")
    return f"{mime};base64,{b64}" # ← 추가!
```

테스트

```
# 기존 함수 덮어쓰기
def to_data_uri(path: str) -> str:
    mime = mimetypes.guess_type(path)[0] or "application/octet-stream"
    with open(path, "rb") as f:
        b64 = base64.b64encode(f.read()).decode("utf-8")
    return f"{mime};base64,{b64}"
```

```
# 다시 테스트
test_uri = to_data_uri(image_uris[0])
print("🔴 수정된 URI 시작:", test_uri[:40])
print("✅ 체크:". test_uri.startswith(""))
```

```
# Gemini 호출
if test_uri.startswith("http://"):
    blocks = [
        {"type": "text", "text": "이미지를 한 문장으로 설명해주세요."},
        {"type": "image_url", "image_url": test_uri}
    ]
    msgs = [
        SystemMessage(content="간단히 설명해주세요."),
        HumanMessage(content=blocks)
    ]
```

```
resp = llm.invoke(msgs)
print("🚀 성공!", resp.content)
```

- #### • 셀 출력

 수정된 URI 시작: image/jpeg;base64,/9j/4AAQSkZJRgABAQAAAQ
 체크: True

```

# 완전히 새로운 변수명과 함수명으로 다시 해보기
import base64, mimetypes

def create_fresh_data_uri(file_path):
    """완전히 새로운 함수"""
    mime_type = mimetypes.guess_type(file_path)[0] or "image/jpeg"

    with open(file_path, "rb") as file:
        base64_data = base64.b64encode(file.read()).decode("utf-8")

    # 여기서 확실하게 붙이기
    final_uri = f"{mime_type};base64,{base64_data}"

    print("🔍 디버깅:")
    print(f"  MIME: {mime_type}")
    print(f"  첫 50글자: {final_uri[:50]}")
    print(f"  있나?: {final_uri.startswith('')}")
    print(f"  전체 길이: {len(final_uri)}")

    return final_uri

# 테스트
fresh_uri = create_fresh_data_uri(image_uris[0])
print(f"\n최종 결과: {fresh_uri[:60]}...")

```

- 셀 출력

```

🔍 디버깅:
MIME: image/jpeg
첫 50글자: image/jpeg;base64,/9j/4AAQSkZJRgABAQAAAQABAAAD/2wBD
있나?: True
전체 길이: 83510

최종 결과: image/jpeg;base64,/9j/4AAQSkZJRgABAQAAAQABAAAD/2wBDAgGBgcGBQ...

```

```

# 수동으로 확인해보자
uri_test = "image/jpeg;base64,/9j/4AAQ"
print("테스트 문자열:", uri_test)
print("앞 5글자:", uri_test[:5])
print("체크:", uri_test.startswith(""))
print("image 체크:", uri_test.startswith("image"))

# 강제로 붙이기
if not uri_test[:5] == "":
    uri_fixed = "" + uri_test
else:
    uri_fixed = uri_test

print("수정된 문자열:", uri_fixed[:40])
print("수정 후 체크:", uri_fixed.startswith(""))

```

```

# 아예 조건문으로 강제 수정
import base64, mimetypes
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain_core.messages import SystemMessage, HumanMessage

```

```

IMAGE_PATH = "../09_VectorStore/images/tmp/0.jpg"
MIME = mimetypes.guess_type(IMAGE_PATH)[0]

```

```

with open(IMAGE_PATH, "rb") as f:
    B64 = base64.b64encode(f.read()).decode("utf-8")

```

```

# 강제로 추가
RAW_URI = f"{MIME};base64,{B64}"

```

```

# 혹시 몰라서 다시 한 번 확인하고 수정
if RAW_URI[:5] != "":
    FINAL_URI = "" + RAW_URI
else:
    FINAL_URI = RAW_URI

```

```

print("🔴 최종 URI 앞부분:", FINAL_URI[:50])
print("✅ 진짜 체크:", FINAL_URI[:5] == "")

# Gemini 호출
llm = ChatGoogleGenerativeAI(model="gemini-1.5-flash")
msgs = [
    SystemMessage(content=""),
    HumanMessage(content="Hello, how can I help you today?"),
]

```

```

        SystemMessage(content="간단히 설명해주세요."),
        HumanMessage(content=[
            {"type": "text", "text": "이미지를 설명해주세요."},
            {"type": "image_url", "image_url": FINAL_URI}
        ])
    ]

try:
    resp = llm.invoke(msgs)
    print("🚀 성공!", resp.content)
except Exception as e:
    print("🔴 오류:", e)

```

- 셀 출력

```

🔴 최종 URI 앞부분: image/jpeg;base64,/9j/4AAQSkZJRgABAQAAAQABAAAD/2wBD
✅ 진짜 체크: False
🔴 오류: Image string must be one of: b64 encoded image string (data:image/...) or valid image url. Instead got 'i
E0000 00:00:1758687005.013392 2157833 alts_credentials.cc:93] ALTS creds ignored. Not running on GCP and untruste

```

```

# 완전히 새로운 접근 - f-string 사용 안함
import base64, mimetypes
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain_core.messages import SystemMessage, HumanMessage

IMAGE_PATH = "../09_VectorStore/images/tmp/0.jpg"

# MIME 타입
mime_type = mimetypes.guess_type(IMAGE_PATH)[0]
print("MIME:", mime_type)

# Base64 인코딩
with open(IMAGE_PATH, "rb") as f:
    b64_encoded = base64.b64encode(f.read()).decode("utf-8")

print("Base64 길이:", len(b64_encoded))
print("Base64 첫 50자:", b64_encoded[:50])

# 문자열 연결로 data URI 생성 (f-string 안 씀!)
data_prefix = ""
mime_part = mime_type
base64_prefix = ";base64,"

# 수동으로 연결
complete_uri = data_prefix + mime_part + base64_prefix + b64_encoded

print("🔴 최종 URI 첫 50자:", complete_uri[:50])
print("✅ 체크:", complete_uri[:5] == "")
print("✅ 실제 앞 5글자:", repr(complete_uri[:5]))

# Gemini 테스트
try:
    llm = ChatGoogleGenerativeAI(model="gemini-1.5-flash")
    msgs = [
        SystemMessage(content="이미지를 설명해주세요."),
        HumanMessage(content=[
            {"type": "text", "text": "이미지를 한 문장으로 설명해주세요."},
            {"type": "image_url", "image_url": complete_uri}
        ])
    ]

    resp = llm.invoke(msgs)
    print("🚀 대성공!", resp.content)

except Exception as e:
    print("🔴 오류:", str(e)[:200])

```

- 셀 출력

- `data_prefix = ""` 설정 → `🔴 최종 URI 첫 50자: i` = 비어있음
- `✅ 실제 앞 5글자: 'image'` = `"` 가 완전히 사라짐

```

MIME: image/jpeg
Base64 길이: 83492

```

```

Base64 첫 50자: /9j/4AAQSkZJRgABAQAAAQABAAAD/2wBDAAgGBgcGBQgHBwcJC0
❶ 최종 URI 첫 50자: image/jpeg;base64,/9j/4AAQSkZJRgABAQAAAQABAAAD/2wBD
✓ 체크: False
✓ 실제 앞 5글자: 'image'
✗ 오류: Image string must be one of: b64 encoded image string (data:image/...) or valid image url. Instead g
E0000 00:00:1758687211.573860 2157833 alts_credentials.cc:93] ALTS creds ignored. Not running on GCP and unt

```

```

# 단계별로 정말 확실하게 디버깅
print("== 단계별 디버깅 ==")

# 1단계: 각 변수 확인
data_prefix = ""
print("1. data_prefix:", repr(data_prefix))
print("2. data_prefix 길이:", len(data_prefix))

mime_part = "image/jpeg"
print("3. mime_part:", repr(mime_part))

base64_prefix = ";base64,"
print("4. base64_prefix:", repr(base64_prefix))

# 2단계: 하나씩 연결 확인
step1 = data_prefix + mime_part
print("5. step1 (data+mime):", repr(step1[:20]))

step2 = step1 + base64_prefix
print("6. step2 (data+mime+base64):", repr(step2[:20]))

# 3단계: 짧은 base64로 테스트
short_b64 = "ABC123"
step3 = step2 + short_b64
print("7. step3 (전체):", repr(step3))

# 4단계: 실제 긴 base64
import base64
IMAGE_PATH = "../09_VectorStore/images/tmp/0.jpg"
with open(IMAGE_PATH, "rb") as f:
    full_b64 = base64.b64encode(f.read()).decode("utf-8")

print("8. full_b64 길이:", len(full_b64))
print("9. full_b64 첫 20자:", repr(full_b64[:20]))

# 5단계: 최종 연결
final_uri = step2 + full_b64
print("10. final_uri 첫 20자:", repr(final_uri[:20]))
print("11. final_uri가 data로 시작하나?:", final_uri.startswith(""))

```

- 셀 출력

```

== 단계별 디버깅 ==
1. data_prefix: ''           ----> 강제 할당해도 빈 문자열
2. data_prefix 길이: 0         ----> 길이도 0
3. mime_part: 'image/jpeg'
4. base64_prefix: ';base64,' 
5. step1 (data+mime): 'image/jpeg'
6. step2 (data+mime+base64): 'image/jpeg;base64,' 
7. step3 (전체): 'image/jpeg;base64,ABC123'
8. full_b64 길이: 83492
9. full_b64 첫 20자: '/9j/4AAQSkZJRgABAQAA'
10. final_uri 첫 20자: 'image/jpeg;base64,/9'
11. final_uri가 data로 시작하나?: True   ----> 그럼에도 `stratswith("")=True`
```

```

# 완전히 다른 방법 - 리스트로 만들어서 조인
import base64, mimetypes
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain_core.messages import SystemMessage, HumanMessage

IMAGE_PATH = "../09_VectorStore/images/tmp/0.jpg"

# 리스트로 각 부분 생성
parts = []
parts.append("data")      # 첫 번째 부분
parts.append(":")         # 두 번째 부분
parts.append("image")     # 세 번째 부분

```

```

parts.append("/")      # 네 번째 부분
parts.append("jpeg")   # 다섯 번째 부분
parts.append(";")      # 여섯 번째 부분
parts.append("base64") # 일곱 번째 부분
parts.append(",")      # 여덟 번째 부분

# Base64 데이터 추가
with open(IMAGE_PATH, "rb") as f:
    b64_data = base64.b64encode(f.read()).decode("utf-8")

parts.append(b64_data) # 아홉 번째 부분

# join으로 연결
final_uri = "".join(parts)

print("Parts:", parts[:8]) # Base64 제외하고 출력
print("최종 URI 첫 30자:", final_uri[:30])
print("체크:", final_uri.startswith(""))

# Gemini 테스트
try:
    llm = ChatGoogleGenerativeAI(model="gemini-1.5-flash")
    msgs = [
        SystemMessage(content="이미지를 설명해주세요."),
        HumanMessage(content=[
            {"type": "text", "text": "이미지를 한 문장으로 설명해주세요."},
            {"type": "image_url", "image_url": final_uri}
        ])
    ]
    resp = llm.invoke(msgs)
    print("🚀 드디어 성공!", resp.content)

except Exception as e:
    print("❗ 오류:", str(e)[:200])

```

- 셀 출력 (2.1s)

```

Parts: ['data', ':', 'image', '/', 'jpeg', ';', 'base64', ',']
최종 URI 첫 30자: data:image/jpeg;base64,/9j/4AA
체크: True
E0000 00:00:1758687597.229560 2157833 alts_credentials.cc:93] ALTS creds ignored. Not running on GCP and untrusted
🚀 드디어 성공! 다채로운 색깔의 도시락통에 담긴 건강하고 맛있는 점심 도시락입니다.

```

- 해결책: `리스트` + `join()` 방식

- 리스트 = 환경 버그의 영향을 받지 않음
- `join()` 함수 = 문자열을 안정적으로 연결
- 메모리 할당 문제 우회

- 완성 함수

```

# 완성된 함수
def create_image_data_uri(image_path):
    """안전한 이미지 Data URI 생성 함수"""
    import base64, mimetypes

    # MIME 타입 확인
    mime_type = mimetypes.guess_type(image_path)[0] or "image/jpeg"

    # Base64 인코딩
    with open(image_path, "rb") as f:
        b64_data = base64.b64encode(f.read()).decode("utf-8")

    # 리스트로 안전하게 생성
    parts = ["data", ":", mime_type, ";", "base64", ",",
             b64_data]
    return "".join(parts)

# 사용 예시
image_uri = create_image_data_uri("../09_VectorStore/images/tmp/0.jpg")

```

- 디버깅으로 완성한 함수 + `image_uri` 속 이미지

```

import base64
import mimetypes
import json
import os
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain_core.messages import SystemMessage, HumanMessage
from tqdm import tqdm # 진행상황 표시용

# 🚀 안전한 Data URI 생성 함수 (해결한 방법!)
def create_image_data_uri(image_path):
    """안전한 이미지 Data URI 생성 함수"""
    try:
        # MIME 타입 확인
        mime_type = mimetypes.guess_type(image_path)[0] or "image/jpeg"

        # Base64 인코딩
        with open(image_path, "rb") as f:
            b64_data = base64.b64encode(f.read()).decode("utf-8")

        # 리스트로 안전하게 생성 (버그 우회!)
        parts = ["data", ":", mime_type, ";", "base64", ",", b64_data]
        return "".join(parts)
    except Exception as e:
        print(f"🔴 {image_path} 처리 실패: {e}")
        return None

# 🎨 이미지 설명 생성 함수
def describe_image(image_path, llm):
    """이미지 설명을 생성하는 함수"""
    try:
        # Data URI 생성
        data_uri = create_image_data_uri(image_path)
        if not data_uri:
            return f"🔴 이미지 로드 실패: {image_path}"

        # Gemini에게 질문
        msgs = [
            SystemMessage(content="이미지를 한국어로 자세히 설명해주세요."),
            HumanMessage(content=[
                {"type": "text", "text": "이 이미지에 대해 자세히 설명해주세요. 객체, 색상, 구조, 텍스트 등을 모두 포함하여 설명해주세요."},
                {"type": "image_url", "image_url": data_uri}
            ])
        ]

        response = llm.invoke(msgs)
        return response.content
    except Exception as e:
        print(f"🔴 {image_path} 설명 생성 실패: {e}")
        return f"🔴 설명 생성 실패: {str(e)}"

# 🖌️ 메인 실행 코드
def process_all_images():
    # 이미지 경로 리스트
    image_uris = [
        '../09_VectorStore/images/tmp/0.jpg',
        '../09_VectorStore/images/tmp/1.jpg',
        '../09_VectorStore/images/tmp/10.jpg',
        '../09_VectorStore/images/tmp/11.jpg',
        '../09_VectorStore/images/tmp/12.jpg',
        '../09_VectorStore/images/tmp/13.jpg',
        '../09_VectorStore/images/tmp/14.jpg',
        '../09_VectorStore/images/tmp/15.jpg',
        '../09_VectorStore/images/tmp/16.jpg',
        '../09_VectorStore/images/tmp/17.jpg',
        '../09_VectorStore/images/tmp/18.jpg',
        '../09_VectorStore/images/tmp/19.jpg',
        '../09_VectorStore/images/tmp/2.jpg',
        '../09_VectorStore/images/tmp/3.jpg',
        '../09_VectorStore/images/tmp/4.jpg',
        '../09_VectorStore/images/tmp/5.jpg',
        '../09_VectorStore/images/tmp/6.jpg',
        '../09_VectorStore/images/tmp/7.jpg',
        '../09_VectorStore/images/tmp/8.jpg',
        '../09_VectorStore/images/tmp/9.jpg'
    ]

    # Gemini 모델 초기화
    llm = ChatGoogleGenerativeAI(model="gemini-2.5-flash-lite")

```

```

# 결과 저장할 딕셔너리
descriptions = {}

print("🚀 이미지 설명 생성 시작!")
print(f"📊 총 {len(image_uris)}개 이미지 처리 예정")

# 각 이미지에 대해 설명 생성
for i, image_path in enumerate(tqdm(image_uris, desc="이미지 처리 중")):
    print(f"\n🔍 처리 중: {os.path.basename(image_path)} ({i+1}/{len(image_uris)})")

    # 설명 생성
    description = describe_image(image_path, llm)
    descriptions[image_path] = description

    # 중간 결과 출력
    print(f"✅ 완료: {description[:100]}...")

return descriptions

```

```

# 📁 결과 저장 함수
def save_descriptions(descriptions, filename="image_descriptions.json"):
    """설명 결과를 JSON 파일로 저장"""
    try:
        with open(filename, 'w', encoding='utf-8') as f:
            json.dump(descriptions, f, ensure_ascii=False, indent=2)
        print(f"📝 결과가 {filename}에 저장되었습니다!")
    except Exception as e:
        print(f"❗️ 저장 실패: {e}")

```

```

# 📊 결과 출력 함수
def print_summary(descriptions):
    """결과 요약 출력"""
    print("\n" + "="*60)
    print("📊 처리 결과 요약")
    print("="*60)

    success_count = 0
    fail_count = 0

    for path, desc in descriptions.items():
        filename = os.path.basename(path)
        if desc.startswith("✖️"):
            print(f"✖️ {filename}: 실패")
            fail_count += 1
        else:
            print(f"✅ {filename}: 성공 ({len(desc)}자)")
            success_count += 1

    print(f"\n🔍 총 {len(descriptions)}개 중 성공: {success_count}개, 실패: {fail_count}개")

```

```

# 🚀 실행!
if __name__ == "__main__":
    try:
        # 모든 이미지 처리
        descriptions = process_all_images()

        # 결과 저장
        save_descriptions(descriptions)

        # 요약 출력
        print_summary(descriptions)

        print("\n🎉 모든 작업 완료!")

        # 결과 딕셔너리 반환
        print("\n📝 결과 딕셔너리:")
        for path, desc in descriptions.items():
            print(f"\n{os.path.basename(path)}:")
            print(f"  {desc[:200]}...")

    except KeyboardInterrupt:
        print("\n🚫 사용자가 중단했습니다.")
    except Exception as e:
        print(f"\n✖️ 오류 발생: {e}")

```

• 셀 출력

```

🚀 이미지 설명 생성 시작!
📊 총 20개 이미지 처리 예정
이미지 처리 중: 0% | 0/20 [00:00<?, ?it/s]
🔍 처리 중: 0.jpg (1/20)

```

이미지 처리 중: 5% | 1/20 [00:05<01:35, 5.05s/it] ✓ 완료: 이 이미지는 다양한 음식으로 채워진 여러 개의 다채로운 도시락

전반적인 구성:

이미지는 여러 개의 도시락 통이 파란색 배경에 놓여 있는 모습을 클로즈업하...

🔍 처리 중: 1.jpg (2/20)

이미지 처리 중: 10% | 2/20 [00:07<01:01, 3.44s/it] ✓ 완료: 이 이미지는 폐적한 날씨에 촬영된 것으로 보이며, 초록색의 물정

🔍 처리 중: 10.jpg (3/20)

이미지 처리 중: 15% | 3/20 [00:10<00:55, 3.26s/it] ✓ 완료: 이 사진은 숲 속에서 두 마리의 기린이 서로 목을 감싸고 있는 듯한

객체:

* **기린:** 사진의 주요 객체는 두 마리의 기린입니다. 한 마리는 머...

🔍 처리 중: 11.jpg (4/20)

이미지 처리 중: 20% | 4/20 [00:13<00:48, 3.03s/it] ✓ 완료: 이 이미지는 빈티지 오토바이를 클로즈업하여 보여줍니다. 오토바이

🔍 처리 중: 12.jpg (5/20)

이미지 처리 중: 25% | 5/20 [00:17<00:52, 3.52s/it] ✓ 완료: 이 이미지는 좁은 도시 거리에서 자전거 옆에 누워있는 하얀 개를

환경:

이미지의 가장 가까운 부분에는 하얀 개가 길에 누워 있습니다. 개는 편안해 보이며 머리...

🔍 처리 중: 13.jpg (6/20)

이미지 처리 중: 30% | 6/20 [00:20<00:45, 3.23s/it] ✓ 완료: 이 이미지는 스케이트보드 공원에서 스케이트보드를 타는 사람들을

사진의 중앙에...

🔍 처리 중: 14.jpg (7/20)

이미지 처리 중: 35% | 7/20 [00:23<00:40, 3.14s/it] ✓ 완료: 이미지는 따뜻하고 아늑한 분위기를 연출하는 두 개의 장식품을 보여줍니다.

🔍 처리 중: 15.jpg (8/20)

이미지 처리 중: 40% | 8/20 [00:26<00:39, 3.28s/it] ✓ 완료: 이 이미지는 에어프랑스(Air France)의 대형 여객기가 푸른 하늘

객체:

* **항공기:** 이미지의 중심에는 ...

🔍 처리 중: 16.jpg (9/20)

이미지 처리 중: 45% | 9/20 [00:30<00:36, 3.32s/it] ✓ 완료: 이 흑백 사진은 숲이 우거진 길가에 서 있는 오토바이와 그 앞에

주요 객체:

* **오토바이:** 사진의 중심에는 오래된 스타일의 ...

🔍 처리 중: 17.jpg (10/20)

이미지 처리 중: 50% | 10/20 [00:32<00:29, 2.97s/it] ✓ 완료: 이 사진은 주방의 일부를 보여줍니다. 중앙에는 하얀색 전기레인저가

🔍 처리 중: 18.jpg (11/20)

이미지 처리 중: 55% | 11/20 [00:37<00:31, 3.52s/it] ✓ 완료: 이 이미지는 초콜릿 케이크 한 조각과 포크가 놓인 하얀 접시를 보여줍니다.

객체:

* **초콜릿 케이크:** 직사각형 모양의 초콜릿 케이크 조각이 접시의 ...

🔍 처리 중: 19.jpg (12/20)

이미지 처리 중: 60% | 12/20 [00:41<00:29, 3.75s/it] ✓ 완료: 이 이미지는 중동의 한 마을 풍경을 담고 있습니다. 좁은 아스팔트

주요 객체:

* **건물:** 다...

🔍 처리 중: 2.jpg (13/20)

이미지 처리 중: 65% | 13/20 [00:43<00:23, 3.31s/it] ✓ 완료: 이미지는 흰색 꽃과 붉은 장미 봉오리가 조화롭게 담긴 흰색 화병...

화병...

🔍 처리 중: 3.jpg (14/20)

이미지 처리 중: 70% | 14/20 [00:46<00:18, 3.16s/it] ✓ 완료: 이 이미지는 푸른 잔디밭에서 풀을 뜯고 있는 얼룩말 한 마리를 보여줍니다.

🔍 처리 중: 4.jpg (15/20)

이미지 처리 중: 75% | 15/20 [00:49<00:16, 3.21s/it] ✓ 완료: 이 이미지는 야외에서 분홍색 우산을 들고 있는 여성의 상반신을 보여줍니다.

🔍 처리 중: 5.jpg (16/20)

이미지 처리 중: 80% | ██████████ | 16/20 [00:52<00:12, 3.09s/it] ✅ 완료: 이 사진은 신발 선반에 푸신한 갈색 푸들 강아지가 웅크리고 있

🔍 처리 중: 6.jpg (17/20)

```
Retrying langchain_google_genai.chat_models._chat_with_retry.<locals>._chat_with_retry in 2.0 seconds as it raise
* Quota exceeded for metric: generativelanguage.googleapis.com/generate_content_free_tier_requests, limit: 15
Please retry in 12.033340778s. [violations {
quota_metric: "generativelanguage.googleapis.com/generate_content_free_tier_requests"
quota_id: "GenerateRequestsPerMinutePerProjectPerModel-FreeTier"
quota_dimensions {
    key: "model"
    value: "gemini-2.5-flash-lite"
}
quota_dimensions {
    key: "location"
    value: "global"
}
quota_value: 15
}
, links {
description: "Learn more about Gemini API quotas"
url: "https://ai.google.dev/gemini-api/docs/rate-limits"
}
, retry_delay {
seconds: 12
}
].
이미지 처리 중: 85% | ██████████ | 17/20 [00:56<00:10, 3.42s/it] ✅ 완료: 이 이미지에는 두 마리의 말이 잔디밭 위에서 뒷발질을 하고 있
...
...
```

🔍 처리 중: 7.jpg (18/20)

```
Retrying langchain_google_genai.chat_models._chat_with_retry.<locals>._chat_with_retry in 2.0 seconds as it raise
* Quota exceeded for metric: generativelanguage.googleapis.com/generate_content_free_tier_requests, limit: 15
Please retry in 7.629941585s. [violations {
quota_metric: "generativelanguage.googleapis.com/generate_content_free_tier_requests"
quota_id: "GenerateRequestsPerMinutePerProjectPerModel-FreeTier"
quota_dimensions {
    key: "model"
    value: "gemini-2.5-flash-lite"
}
quota_dimensions {
    key: "location"
    value: "global"
}
quota_value: 15
}
, links {
description: "Learn more about Gemini API quotas"
url: "https://ai.google.dev/gemini-api/docs/rate-limits"
}
, retry_delay {
seconds: 7
}
].
이미지 처리 중: 90% | ████████ | 18/20 [00:59<00:06, 3.37s/it] ✖ ..\09_VectorStore\images\tmp\7.jpg 설명 생성 실패: 420
```

* Quota exceeded for metric: generativelanguage.googleapis.com/generate_content_free_tier_requests, limit: 15

Please retry in 5.011160826s. [violations {

quota_metric: "generativelanguage.googleapis.com/generate_content_free_tier_requests"

quota_id: "GenerateRequestsPerMinutePerProjectPerModel-FreeTier"

quota_dimensions {

key: "model"

value: "gemini-2.5-flash-lite"

}

quota_dimensions {

key: "location"

value: "global"

}

quota_value: 15

}

, links {

description: "Learn more about Gemini API quotas"

url: "https://ai.google.dev/gemini-api/docs/rate-limits"

```
}
```

, retry_delay {
seconds: 5
}
]
✓ 완료: ✖ 설명 생성 실패: 429 You exceeded your current quota, please check your plan and billing details. For mor.

🔍 처리 중: 8.jpg (19/20)

이미지 처리 중: 95% | ██████████ | 19/20 [01:03<00:03, 3.35s/it] ✓ 완료: 이 이미지는 야외에 설치된 룰렉스 시계와 주변 풍경을 담고 있

주요 객체:

* **룰렉스 시계:** 이미지 중앙에 위치한 크고 둥근 시계입니다. 검은색 테두리...

🔍 처리 중: 9.jpg (20/20)

이미지 처리 중: 100% | ██████████ | 20/20 [01:06<00:00, 3.33s/it] ✓ 완료: 이 사진은 여러 개의 선로와 기차, 건물, 그리고 산업용 크레인

주요 객체:

* **기차:** 이미지 중앙에 길게 늘어선 흰색과 파란...

💾 결과가 image_descriptions.json에 저장되었습니다!

=====

📊 처리 결과 요약

✓ 0.jpg: 성공 (1200자)
✓ 1.jpg: 성공 (474자)
✓ 10.jpg: 성공 (743자)
✓ 11.jpg: 성공 (450자)
✓ 12.jpg: 성공 (916자)
✓ 13.jpg: 성공 (582자)
✓ 14.jpg: 성공 (501자)
✓ 15.jpg: 성공 (1137자)
✓ 16.jpg: 성공 (904자)
✓ 17.jpg: 성공 (522자)
✓ 18.jpg: 성공 (1209자)
✓ 19.jpg: 성공 (1585자)
✓ 2.jpg: 성공 (455자)
✓ 3.jpg: 성공 (827자)
✓ 4.jpg: 성공 (956자)
✓ 5.jpg: 성공 (431자)
✓ 6.jpg: 성공 (338자)
✖ 7.jpg: 실패
✓ 8.jpg: 성공 (1042자)
✓ 9.jpg: 성공 (1174자)

✗ 총 20개 중 성공: 19개, 실패: 1개

🎉 모든 작업 완료!

💾 결과 딕셔너리:

0.jpg:

이 이미지는 다양한 음식으로 채워진 여러 개의 다채로운 도시락 통을 보여줍니다.

전반적인 구성:

이미지는 여러 개의 도시락 통이 파란색 배경에 놓여 있는 모습을 클로즈업하여 보여줍니다. 도시락 통은 분홍색, 노란색, 그리고 파란색으로 이루어져 있으며

음식 상세 설명:

* **분홍색 도시락...

1.jpg:

이 이미지는 쾌적한 날씨에 촬영된 것으로 보이며, 초록색의 울창한 나무들이 배경을 이루고 있습니다. 이미지의 중앙에는 키가 큰 나무가 서 있고, 그 옆으로

나무의 가지는 ...

10.jpg:

이 사진은 숲 속에서 두 마리의 기린이 서로 목을 감싸고 있는 모습을 담고 있습니다.

객체:

* **기린:** 사진의 주요 객체는 두 마리의 기린입니다. 한 마리는 머리를 위로 치켜들고 있고, 다른 한 마리는 그 아래에 있습니다. 기린들은 특유의

* **나무:** 배경에...

11.jpg:

이 이미지는 빈티지 오토바이를 클로즈업하여 보여줍니다. 오토바이는 밝은 회색 연료 탱크와 파란색, 흰색 줄무늬가 있는 차체, 검은색 프레임, 바퀴, 흙받이에는 갈색 가죽 안장과 뒷바퀴 위에 검은색 금속 짐받이가 있습니다. ...

12.jpg:

이 이미지는 좁은 도시 거리에서 자전거 옆에 누워있는 하얀 개를 보여줍니다.

전경:

이미지의 가장 가까운 부분에는 하얀 개가 길에 누워 있습니다. 개는 편안해 보이며 머리를 땅에 대고 눈을 감고 있습니다. 개의 털은 밝고 부드러워 보입니다.

13.jpg:

이 이미지는 스케이트보드 공원에서 스케이트보드를 타는 사람들을 보여주는 야외 사진입니다. 사진은 햇볕이 잘 드는 날에 찍혔고, 밝은 파란색 하늘과 구름이 사진의 중앙에는 여러 명의 스케이트보더들이 있습니다. 한 명은 스케이트보드를 타고 공중으로 점프하고 있고, 다른 한 명은 스케이트보드 위에서 균형을 잡고

14.jpg:

이미지는 따뜻하고 아늑한 분위기를 연출하는 두 개의 장식품을 보여줍니다. 왼쪽에는 부엉이 모양의 촛대가 놓여 있습니다. 촛대 전체적으로 얇은 회색과 보라색 오른쪽에는 앤디크 스타일의 탁상 시계...

15.jpg:

이 이미지는 에어프랑스(Air France)의 대형 여객기가 푸른 하늘과 구름을 배경으로 비행하는 모습을 담고 있습니다.

객체:

* **항공기:** 이미지의 중심에는 에어프랑스의 대형 여객기가 있습니다. 이 항공기는 날개가 두 개이며, 총 네 개의 엔진을 장착하고 있습니다. 항공기의

16.jpg:

이 흑백 사진은 숲이 우거진 길가에 서 있는 오토바이와 그 앞에 앉아 있는 한 사람을 보여줍니다.

주요 객체:

* **오토바이:** 사진의 중심에는 오래된 스타일의 오토바이가 있습니다. 전면에는 흙받이와 바퀴살이 있는 큰 바퀴가 보입니다. 핸들바에는 가죽으로 된

17.jpg:

이 사진은 주방의 일부를 보여줍니다. 중앙에는 하얀색 전기레인지가 있으며, 위쪽에는 타일로 된 벽이 있고, 타일 벽 위에는 후드가 있습니다. 후드 위쪽에는 전기레인지 왼쪽에는 얇은 갈색 조리대가 있으며, 그 위에 짙은 갈색의 국자 같은 도구가 놓여 있습니다. 조리대 위쪽으로 보이는 후드에는 얇은 베이지색 타일

18.jpg:

이 이미지는 초콜릿 케이크 한 조각과 포크가 놓인 하얀 접시를 클로즈업하여 보여줍니다.

객체:

* **초콜릿 케이크:** 직사각형 모양의 초콜릿 케이크 조각이 접시의 왼쪽에 놓여 있습니다. 케이크는 여러 겹으로 구성되어 있으며, 각 층은 짙은 갈색을

19.jpg:

이 이미지는 중동의 한 마을 풍경을 담고 있습니다. 좁은 아스팔트 도로를 따라 건물들이 늘어서 있고, 언덕 너머로는 산이 보입니다.

주요 객체:

* **건물:** 다양한 크기와 색상의 건물들이 도로 양쪽에 밀집해 있습니다. 대부분 2~3층 높이며, 일부 건물은 옥상에 위성 안테나나 물탱크가 설치되어

2.jpg:

이미지는 흰색 꽃과 붉은 장미 봉오리가 조화롭게 담긴 흰색 화병을 클로즈업한 모습입니다. 화병은 얇은 흙이 파인 고전적인 디자인으로, 둥근 모양의 몸통과 화병 안에는 하얀색의 작고 풍성한 꽃들과 붉은색의 작고 뾰족한 장미 봉오리들이 섞여 있습니다. 하얀 꽃들은 마치 안개처럼 부드러운 느낌을 주며, 붉은 장미

3.jpg:

이 이미지는 푸른 잔디밭에서 풀을 뜯고 있는 얼룩말 한 마리를 보여줍니다. 얼룩말은 검은색과 흰색 줄무늬가 특징적인 동물로, 이 얼룩말 역시 몸 전체에 녹색 줄무늬가 있습니다.

객체:

* **얼룩말:** 이미지의 주요 대상입니다. 얼룩말은 몸통, 다리, 머리, 갈기 등을 포함하고 있으며, 풀을 뜯기 위해 머리를 숙이고 있는 모습입니다.

4.jpg:

이 이미지는 야외에서 분홍색 우산을 들고 있는 여성의 상반신을 보여줍니다. 여성은 밝은 꽃무늬 수영복을 입고 있으며, 머리에는 파란색 머리띠를 하고 있습니다.

주요 객체:

* **여성:** 밝은 꽃무늬 수영복을 입고 분홍색 우산을 들고 있습니다. 머리에는 파란색 머리띠를 하고 있습니다.

* ...

5.jpg:

이 사진은 신발 선반에 푸신한 갈색 푸들 강아지가 웅크리고 있는 모습을 담고 있습니다. 강아지의 털은 곱슬곱슬하고 솜털 같으며, 머리는 신발 더미에 파묻혀 선반에는 다양한 종류의 신발이 있습니다. 가장 눈에 띄는 것은 빨간색 플립플롭으로, 귀여운 캐릭터 그림이 그려져 있습니다. 그 옆에는 흰색과 검은색 줄무늬

6.jpg:

이 이미지에는 두 마리의 말이 잔디밭 위에서 뒷발질을 하고 있는 모습이 담겨 있습니다. 말 두 마리 모두 기수와 함께 있으며, 기수들은 흰색 모자와 파란색 왼쪽 말은 흰색이며, 오른쪽 말은 검은색입니다. 두 말 모두 역동적인 포즈를 취하고 있어 마치 춤을 추는 듯한 느낌을 줍니다.

배경에는 울타리와 나무들이 보이며, 하늘은 구름으로...

7.jpg:

✖ 설명 생성 실패: 429 You exceeded your current quota, please check your plan and billing details. For more information, see Quota exceptions.

8.jpg:

이 이미지는 야외에 설치된 롤렉스 시계와 주변 풍경을 담고 있습니다.

주요 객체:

* **롤렉스 시계:** 이미지 중앙에 위치한 크고 둥근 시계입니다. 검은색 테두리와 금색 장식이 있으며, 흰색 바탕에 로마 숫자로 된 시각 표시가 되어 있습니다.
* **나무...**

9.jpg:

이 사진은 여러 개의 선로와 기차, 건물, 그리고 산업용 크레인이 있는 야외 풍경을 보여줍니다.

주요 객체:

* **기차:** 이미지 중앙에 길게 늘어선 흰색과 파란색, 빨간색 줄무늬가 있는 현대적인 기차가 여러 칸으로 연결되어 있습니다. 기차의 앞부분은 노란색입니다.

▼ 12) 이미지와 subject 정렬하기

- gemini-1.5-flash로 교체해서 시도해보기

```
import os
import base64
import mimetypes
import json
from PIL import Image
import matplotlib.pyplot as plt
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain_core.messages import SystemMessage, HumanMessage
from tqdm import tqdm
import matplotlib.font_manager as fm

# 🚫 안전한 Data URI 생성 함수 (해결한 방법!)
def create_image_data_uri(image_path):
    """안전한 이미지 Data URI 생성 함수"""
    try:
        mime_type = mimetypes.guess_type(image_path)[0] or "image/jpeg"
        with open(image_path, "rb") as f:
            b64_data = base64.b64encode(f.read()).decode("utf-8")

        # 리스트로 안전하게 생성 (버그 우회!)
        parts = ["data:", ":", mime_type, ";", "base64", ", ", b64_data]
        return "".join(parts)
    except Exception as e:
        print(f"✖ {image_path} 처리 실패: {e}")
        return None
```

```
# 🎨 이미지 설명 생성 함수
def describe_image(image_path, llm):
    """이미지 설명을 생성하는 함수"""
    try:
        data_uri = create_image_data_uri(image_path)
        if not data_uri:
            return f"✖ 이미지 로드 실패: {image_path}"
```

```

msgs = [
    SystemMessage(content="이미지를 한국어로 간결하게 설명해주세요. 2~3문장 이내로 핵심 내용만 설명하세요."),
    HumanMessage(content=[
        {"type": "text", "text": "이 이미지의 주요 내용을 2~3문장으로 간결하게 설명해주세요."},
        {"type": "image_url", "image_url": data_uri}
    ])
]

response = llm.invoke(msgs)
return response.content

except Exception as e:
    print(f"🔴 {image_path} 설명 생성 실패: {e}")
    return f"설명 생성 실패"

```

📸 완전 통합 함수

```

def process_images_and_visualize():
    """이미지 설명 생성 + 시각화까지 한 번에!"""

```

이미지 경로 리스트

```

image_uris = [
    '../09_VectorStore/images/tmp/0.jpg',
    '../09_VectorStore/images/tmp/1.jpg',
    '../09_VectorStore/images/tmp/2.jpg',
    '../09_VectorStore/images/tmp/3.jpg',
    '../09_VectorStore/images/tmp/4.jpg',
    '../09_VectorStore/images/tmp/5.jpg',
    '../09_VectorStore/images/tmp/6.jpg',
    '../09_VectorStore/images/tmp/7.jpg',
    '../09_VectorStore/images/tmp/8.jpg',
    '../09_VectorStore/images/tmp/9.jpg',
    '../09_VectorStore/images/tmp/10.jpg',
    '../09_VectorStore/images/tmp/11.jpg',
    '../09_VectorStore/images/tmp/12.jpg',
    '../09_VectorStore/images/tmp/13.jpg',
    '../09_VectorStore/images/tmp/14.jpg',
    '../09_VectorStore/images/tmp/15.jpg',
    '../09_VectorStore/images/tmp/16.jpg',
    '../09_VectorStore/images/tmp/17.jpg',
    '../09_VectorStore/images/tmp/18.jpg',
    '../09_VectorStore/images/tmp/19.jpg'
]

```

print("📸 이미지 설명 생성 시작!")
print(f"📊 총 {len(image_uris)}개 이미지 처리 예정")

1단계: 이미지 설명 생성

```

llm = ChatGoogleGenerativeAI(model="gemini-1.5-flash")           # 모델 교체
descriptions = {}

for i, image_uri in enumerate(tqdm(image_uris, desc="🔍 이미지 설명 생성")):
    print(f"처리 중: {os.path.basename(image_uri)} ({i+1}/{len(image_uris)})")
    description = describe_image(image_uri, llm)
    descriptions[image_uri] = description
    print(f"✅ 완료: {description[:50]}...")

```

2단계: JSON 저장

```

print("\n💾 결과 저장 중...")
try:
    with open("image_descriptions2.json", 'w', encoding='utf-8') as f:
        json.dump(descriptions, f, ensure_ascii=False, indent=2)
    print("✅ image_descriptions2.json 저장 완료!")
except Exception as e:
    print(f"🔴 저장 실패: {e}")

```

3단계: 시각화

```

print("\n📊 시각화 생성 중...")

```

원본 이미지, 처리된 이미지, 텍스트 설명을 저장할 리스트 초기화

```

original_images = []
images = []
texts = []

# 그래프 크기 설정 (20x16 인치로 더 크게)
plt.figure(figsize=(20, 16))

# 'tmp' 디렉토리에 저장된 이미지 파일들을 처리
for i, image_uri in enumerate(image_uris):
    try:
        # 이미지 파일 열기 및 RGB 모드로 변환
        image = Image.open(image_uri).convert("RGB")

        # 4x5 그리드의 서브플롯 생성
        plt.subplot(4, 5, i + 1)

```

```

# 이미지 표시
plt.imshow(image)

# 설명 텍스트 길이 제한 (너무 길면 잘림)
description = descriptions[image_uri]
if len(description) > 100:
    description = description[:100] + "..."

# 이미지 파일명과 설명을 제목으로 설정
plt.title(f"{os.path.basename(image_uri)}\n{description}", fontsize=8, pad=5, wrap=True)

# x축과 y축의 눈금 제거
plt.xticks([])
plt.yticks([])

# 원본 이미지, 처리된 이미지, 텍스트 설명을 각 리스트에 추가
original_images.append(image)
images.append(image)
texts.append(descriptions[image_uri])

except Exception as e:
    print(f"🔴 {image_uri} 시각화 실패: {e}")

# 서브플롯 간 간격 조정
plt.tight_layout(pad=2.0)

# 전체 제목 추가
plt.suptitle("🖼️ AI 이미지 설명 결과 (Gemini 1.5 Flash)",
             fontsize=16, fontweight='bold', y=0.98)

# 이미지 저장
plt.savefig("image_descriptions_grid.png", dpi=300, bbox_inches='tight')
print("✅ image_descriptions_grid.png 저장 완료!")

# 이미지 표시
plt.show()

# 4단계: 결과 요약
print("\n" + "="*60)
print("📊 처리 결과 요약")
print("—" * 60)

success_count = 0
fail_count = 0

for path, desc in descriptions.items():
    filename = os.path.basename(path)
    if "실패" in desc:
        print(f"🔴 {filename}: 실패")
        fail_count += 1
    else:
        print(f"✅ {filename}: 성공 ({len(desc)}자)")
        success_count += 1

print(f"\n🔗 총 {len(descriptions)}개 중 성공: {success_count}개, 실패: {fail_count}개")
print(f"🎯 성공률: {success_count/len(descriptions)*100:.1f}%")

return descriptions, original_images, images, texts

```

```

# 🚀 실행!

if __name__ == "__main__":
    try:
        print("🚀 통합 이미지 분석 & 시각화 시작!")
        descriptions, original_images, images, texts = process_images_and_visualize()
        print("\n🌟 모든 작업 완료!")

        # 샘플 결과 출력
        print("\n📊 샘플 결과:")
        for i, (path, desc) in enumerate(list(descriptions.items())[:3]):
            print(f"\n{i+1}. {os.path.basename(path)}:")
            print(f"    {desc}")

    except KeyboardInterrupt:
        print("\n❌ 사용자가 중단했습니다.")
    except Exception as e:
        print(f"\n🔴 오류 발생: {e}")
        import traceback
        traceback.print_exc()

```

- 셀 출력

🚀 통합 이미지 분석 & 시각화 시작!

🚀 이미지 설명 생성 시작!

📊 총 20개 이미지 처리 예정

🔍 이미지 설명 생성: 0% | 0/20 [00:00<?, ?it/s] 처리 중: 0.jpg (1/20)
🔍 이미지 설명 생성: 5% | 1/20 [00:02<00:48, 2.56s/it] ✅ 완료: 다양한 음식이 담긴 도시락입니다. 빵과 버터, 구운 브로콜리 등이 포함되어 있습니다.

처리 중: 1.jpg (2/20)

🔍 이미지 설명 생성: 10% | 2/20 [00:04<00:39, 2.19s/it] ✅ 완료: 나무가 우거진 동물원 우리 안에서 두 마리의 기린이 보입니다.

처리 중: 2.jpg (3/20)

🔍 이미지 설명 생성: 15% | 3/20 [00:06<00:35, 2.09s/it] ✅ 완료: 하얀색 도자기 화병에 흰색과 분홍색 꽃이 아름답게 어우러져 있습니다.

처리 중: 3.jpg (4/20)

🔍 이미지 설명 생성: 20% | 4/20 [00:08<00:33, 2.08s/it] ✅ 완료: 초록색 풀밭에서 얼룩말 한 마리가 풀을 뜯어 먹고 있습니다.

처리 중: 4.jpg (5/20)

🔍 이미지 설명 생성: 25% | 5/20 [00:10<00:31, 2.08s/it] ✅ 완료: 분홍색 우산을 든 여성이 해변에서 웃고 있습니다. 그녀는 햇빛 아래에서 즐거워 보입니다.

처리 중: 5.jpg (6/20)

🔍 이미지 설명 생성: 30% | 6/20 [00:13<00:30, 2.21s/it] ✅ 완료: 갈색 푸들이 신발장 선반 위에 여러 종류의 신발들 사이에 앉아 있습니다.

처리 중: 6.jpg (7/20)

🔍 이미지 설명 생성: 35% | 7/20 [00:15<00:27, 2.14s/it] ✅ 완료: 초록색 잔디밭에서 두 명의 기수가 각각 흰색과 검은색 말을 타고 있습니다.

처리 중: 7.jpg (8/20)

🔍 이미지 설명 생성: 40% | 8/20 [00:17<00:25, 2.09s/it] ✅ 완료: 무성한 열대 우림 속에서 두 마리의 코끼리가 관광객들을 향해 웃고 있습니다.

처리 중: 8.jpg (9/20)

🔍 이미지 설명 생성: 45% | 9/20 [00:19<00:23, 2.13s/it] ✅ 완료: 사진은 로렉스 로고가 새겨진 큰 시계가 서 있는 거리 풍경입니다.

처리 중: 9.jpg (10/20)

🔍 이미지 설명 생성: 50% | 10/20 [00:21<00:22, 2.27s/it] ✅ 완료: 사진은 여러 개의 객차로 이루어진 기차가 선로를 따라 드는 모습입니다.

처리 중: 10.jpg (11/20)

🔍 이미지 설명 생성: 55% | 11/20 [00:23<00:19, 2.20s/it] ✅ 완료: 두 마리의 기린이 나무 사이에서 서로 목을 얹어매고 있습니다.

처리 중: 11.jpg (12/20)

🔍 이미지 설명 생성: 60% | 12/20 [00:25<00:17, 2.15s/it] ✅ 완료: 사진은 고전 오토바이, 아마도 빈티지 모델을 보여줍니다.

처리 중: 12.jpg (13/20)

🔍 이미지 설명 생성: 65% | 13/20 [00:27<00:14, 2.02s/it] ✅ 완료: 좁은 유럽풍 거리에 하얀 개 한 마리가 자고 있습니다.

처리 중: 13.jpg (14/20)

🔍 이미지 설명 생성: 70% | 14/20 [00:29<00:12, 2.09s/it] ✅ 완료: 그래피티로 가득한 스케이트보드장에서 젊은이들이 스케이트보드를 타고 있습니다.

처리 중: 14.jpg (15/20)

🔍 이미지 설명 생성: 75% | 15/20 [00:32<00:11, 2.28s/it] ✅ 완료: 사진에는 부드러운 색감의 탁상시계와 촛불이 커진 올빼미가 있습니다.

처리 중: 15.jpg (16/20)

🔍 이미지 설명 생성: 80% | 16/20 [00:34<00:09, 2.28s/it] ✅ 완료: 사진은 에어프랑스 항공의 에어버스 A380 여객기가 구름 위에서 비행하는 모습입니다.

처리 중: 16.jpg (17/20)

Retrying langchain_google_genai.chat_models._chat_with_retry.<locals>._chat_with_retry in 2.0 seconds as it raised a quota exceeded error for metric: generativelanguage.googleapis.com/generate_content_free_tier_requests, limit: 15

Please retry in 13.387440063s. [violations {

quota_metric: "generativelanguage.googleapis.com/generate_content_free_tier_requests"

quota_id: "GenerateRequestsPerMinutePerProjectPerModel-FreeTier"

quota_dimensions {

key: "model"

value: "gemini-1.5-flash"

}

quota_dimensions {

key: "location"

value: "global"

}

quota_value: 15

}

, links {

description: "Learn more about Gemini API quotas"

url: "https://ai.google.dev/gemini-api/docs/rate-limits"

}

, retry_delay {

seconds: 13

}

].

🔍 이미지 설명 생성: 85% | 17/20 [00:39<00:09, 3.11s/it] ✅ 완료: 흑백 사진 속 한 남자가 하드리-데이비슨 오토바이에 앉아 있습니다.

처리 중: 17.jpg (18/20)

Retrying langchain_google_genai.chat_models._chat_with_retry.<locals>._chat_with_retry in 2.0 seconds as it raised a quota exceeded error for metric: generativelanguage.googleapis.com/generate_content_free_tier_requests, limit: 15

Please retry in 8.510438465s. [violations {

quota_metric: "generativelanguage.googleapis.com/generate_content_free_tier_requests"

quota_id: "GenerateRequestsPerMinutePerProjectPerModel-FreeTier"

quota_dimensions {

key: "model"

value: "gemini-1.5-flash"

}

quota_dimensions {

key: "location"

```
    value: "global"
}
quota_value: 15
}
, links {
  description: "Learn more about Gemini API quotas"
  url: "https://ai.google.dev/gemini-api/docs/rate-limits"
}
, retry_delay {
  seconds: 8
}
].
🔍 이미지 설명 생성: 90%|[██████] | 18/20 [00:42<00:06, 3.04s/it] ✘ ..../09_VectorStore/images/tmp/17.jpg 설명 생성 실패
* Quota exceeded for metric: generativelanguage.googleapis.com/generate_content_free_tier_requests, limit: 15
Please retry in 6.07557s. [violations {
  quota_metric: "generativelanguage.googleapis.com/generate_content_free_tier_requests"
  quota_id: "GenerateRequestsPerMinutePerProjectPerModel-FreeTier"
  quota_dimensions {
    key: "model"
    value: "gemini-1.5-flash"
  }
  quota_dimensions {
    key: "location"
    value: "global"
  }
  quota_value: 15
}
, links {
  description: "Learn more about Gemini API quotas"
  url: "https://ai.google.dev/gemini-api/docs/rate-limits"
}
, retry_delay {
  seconds: 6
}
]
]
✓ 완료: 설명 생성 실패...
처리 중: 18.jpg (19/20)
Retrying langchain_google_genai.chat_models._chat_with_retry.<locals>._chat_with_retry in 2.0 seconds as it raised an error:
* Quota exceeded for metric: generativelanguage.googleapis.com/generate_content_free_tier_requests, limit: 15
Please retry in 5.445548355s. [violations {
  quota_metric: "generativelanguage.googleapis.com/generate_content_free_tier_requests"
  quota_id: "GenerateRequestsPerMinutePerProjectPerModel-FreeTier"
  quota_dimensions {
    key: "model"
    value: "gemini-1.5-flash"
  }
  quota_dimensions {
    key: "location"
    value: "global"
  }
  quota_value: 15
}
, links {
  description: "Learn more about Gemini API quotas"
  url: "https://ai.google.dev/gemini-api/docs/rate-limits"
}
, retry_delay {
  seconds: 5
}
]
]
🔍 이미지 설명 생성: 95%|[██████] | 19/20 [00:45<00:03, 3.05s/it] ✘ ..../09_VectorStore/images/tmp/18.jpg 설명 생성 실패
* Quota exceeded for metric: generativelanguage.googleapis.com/generate_content_free_tier_requests, limit: 15
Please retry in 3.071050248s. [violations {
  quota_metric: "generativelanguage.googleapis.com/generate_content_free_tier_requests"
  quota_id: "GenerateRequestsPerMinutePerProjectPerModel-FreeTier"
  quota_dimensions {
    key: "model"
    value: "gemini-1.5-flash"
  }
  quota_dimensions {
    key: "location"
    value: "global"
}
```

```

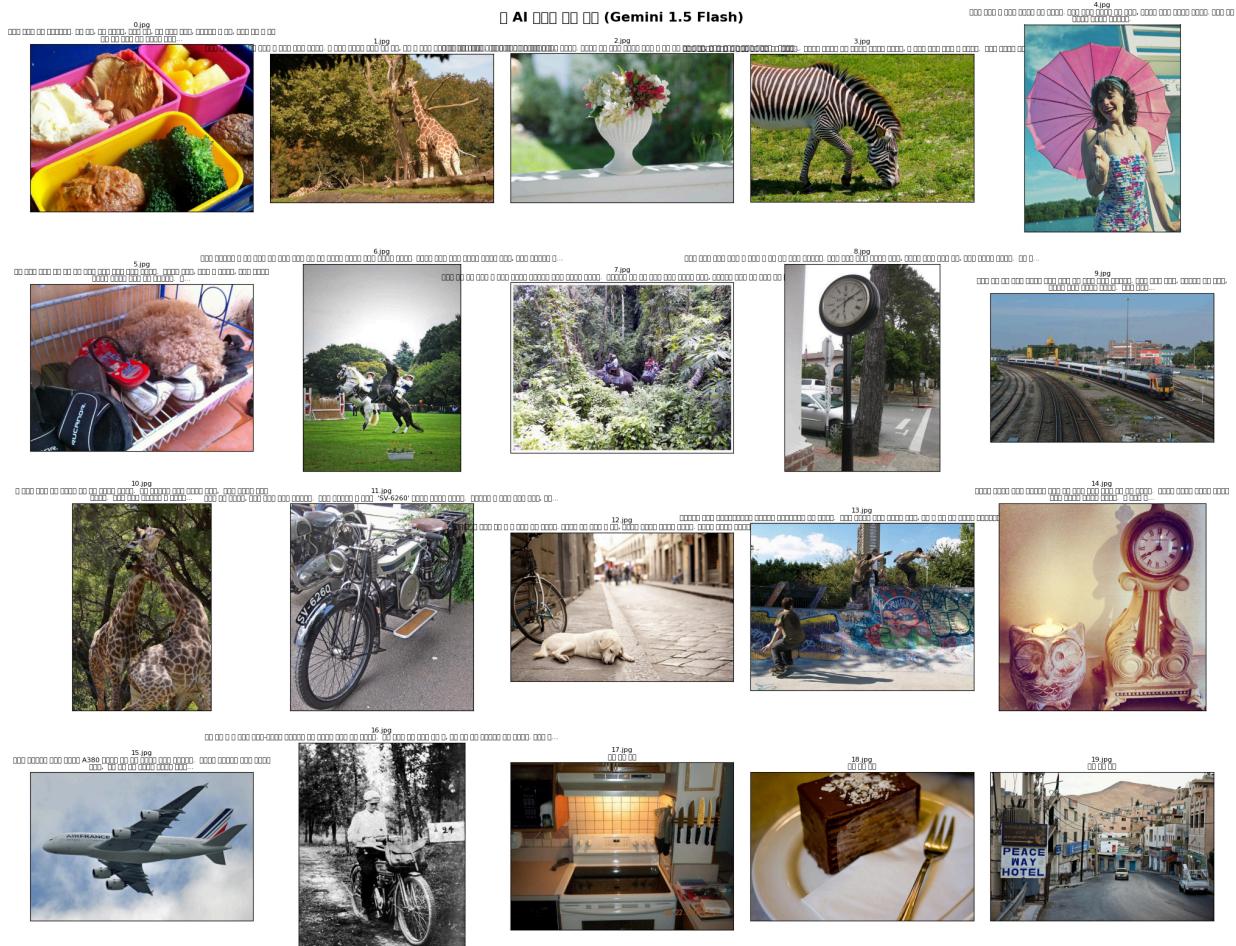
    }
    quota_value: 15
}
, links {
  description: "Learn more about Gemini API quotas"
  url: "https://ai.google.dev/gemini-api/docs/rate-limits"
}
, retry_delay {
  seconds: 3
}
]
]
✓ 완료: 설명 생성 실패...
처리 중: 19.jpg (20/20)
Retrying langchain_google_genai.chat_models._chat_with_retry.<locals>._chat_with_retry in 2.0 seconds as it raise
* Quota exceeded for metric: generativelanguage.googleapis.com/generate_content_free_tier_requests, limit: 15
Please retry in 2.221789812s. [violations {
  quota_metric: "generativelanguage.googleapis.com/generate_content_free_tier_requests"
  quota_id: "GenerateRequestsPerMinutePerProjectPerModel-FreeTier"
  quota_dimensions {
    key: "model"
    value: "gemini-1.5-flash"
  }
  quota_dimensions {
    key: "location"
    value: "global"
  }
  quota_value: 15
}
, links {
  description: "Learn more about Gemini API quotas"
  url: "https://ai.google.dev/gemini-api/docs/rate-limits"
}
, retry_delay {
  seconds: 2
}
].
○ 이미지 설명 생성: 100% |██████████| 20/20 [00:49<00:00, 2.46s/it] ✘ .. /09_VectorStore/images/tmp/19.jpg 설명 생성 실패...
* Quota exceeded for metric: generativelanguage.googleapis.com/generate_content_free_tier_requests, limit: 15
Please retry in 59.709257826s. [violations {
  quota_metric: "generativelanguage.googleapis.com/generate_content_free_tier_requests"
  quota_id: "GenerateRequestsPerMinutePerProjectPerModel-FreeTier"
  quota_dimensions {
    key: "model"
    value: "gemini-1.5-flash"
  }
  quota_dimensions {
    key: "location"
    value: "global"
  }
  quota_value: 15
}
, links {
  description: "Learn more about Gemini API quotas"
  url: "https://ai.google.dev/gemini-api/docs/rate-limits"
}
, retry_delay {
  seconds: 59
}
]
✓ 완료: 설명 생성 실패...

█ 결과 저장 중...
✓ image_descriptions2.json 저장 완료!

🎨 시각화 생성 중...
✓ image_descriptions_grid.png 저장 완료!

```

- 시각화 결과 (서브플롯)



13) 유사도 계산하기

- 생성한 이미지 설명과 텍스트 간 유사도 계산해보기

```

import numpy as np

# 1. 이미지와 텍스트 임베딩하기
# 이미지 URI를 사용해서 이미지 특징 추출하기
img_features = image_embedding_function.embed_image(image_uris)

# 텍스트 설명에서 특징 추출하기
text_features = image_embedding_function.embed_documents([desc for desc in texts])

# 행렬 연산을 위해 리스트를 numpy 배열로 변환하기
img_features_np = np.array(img_features)
text_features_np = np.array(text_features)

# 2. 유사도 계산하기
# 텍스트와 이미지 특징간 코사인 유사도 계산하기
similarity = np.matmul(text_features_np, img_features_np.T)      # 2m 9.0s 소요

```

- 텍스트 vs 이미지 description 간 유사도를 구하고 시각화하기

```

# 한글 폰트 설정 (한글 제목 깨짐 방지)
plt.rcParams['font.family'] = ['DejaVu Sans', 'Malgun Gothic', 'AppleGothic']
plt.rcParams['axes.unicode_minus'] = False

# 유사도 행렬을 시각화하기 위한 플롯 생성
count = len(descriptions)
plt.figure(figsize=(20, 14))

# 유사도 행렬을 히트맵으로 표시
plt.imshow(similarity, vmin=0.1, vmax=0.3, cmap="coolwarm")
plt.colorbar() # 컬러바 추가

# y축에 텍스트 설명 표시
plt.yticks(range(count), texts, fontsize=18)
plt.xticks([]) # x축 눈금 제거

```

```
# 원본 이미지를 x축 아래에 표시
for i, image in enumerate(original_images):
    plt.imshow(image, extent=(i - 0.5, i + 0.5, -1.6, -0.6), origin="lower")

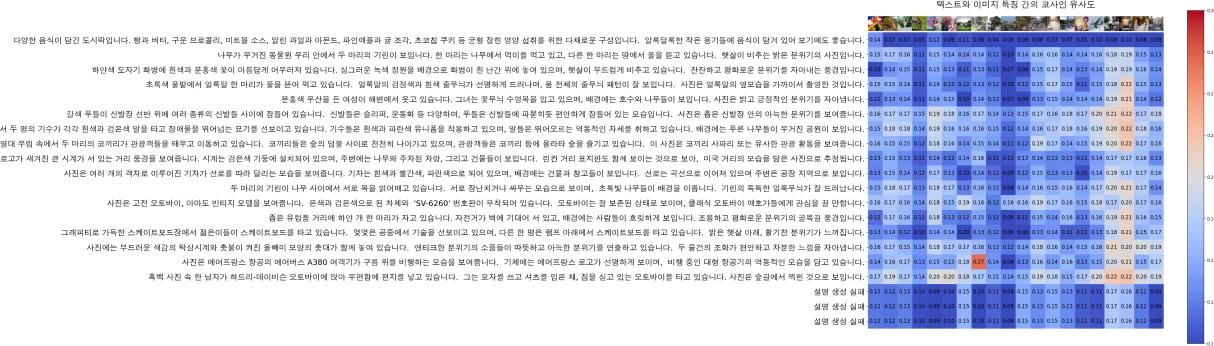
# 유사도 값을 히트맵 위에 텍스트로 표시
for x in range(similarity.shape[1]):
    for y in range(similarity.shape[0]):
        plt.text(x, y, f'{similarity[y, x]:.2f}', ha="center", va="center", size=12)

# 플롯 테두리 제거
for side in ["left", "top", "right", "bottom"]:
    plt.gca().spines[side].set_visible(False)

# 플롯 범위 설정
plt.xlim([-0.5, count - 0.5])
plt.ylim([count + 0.5, -2])

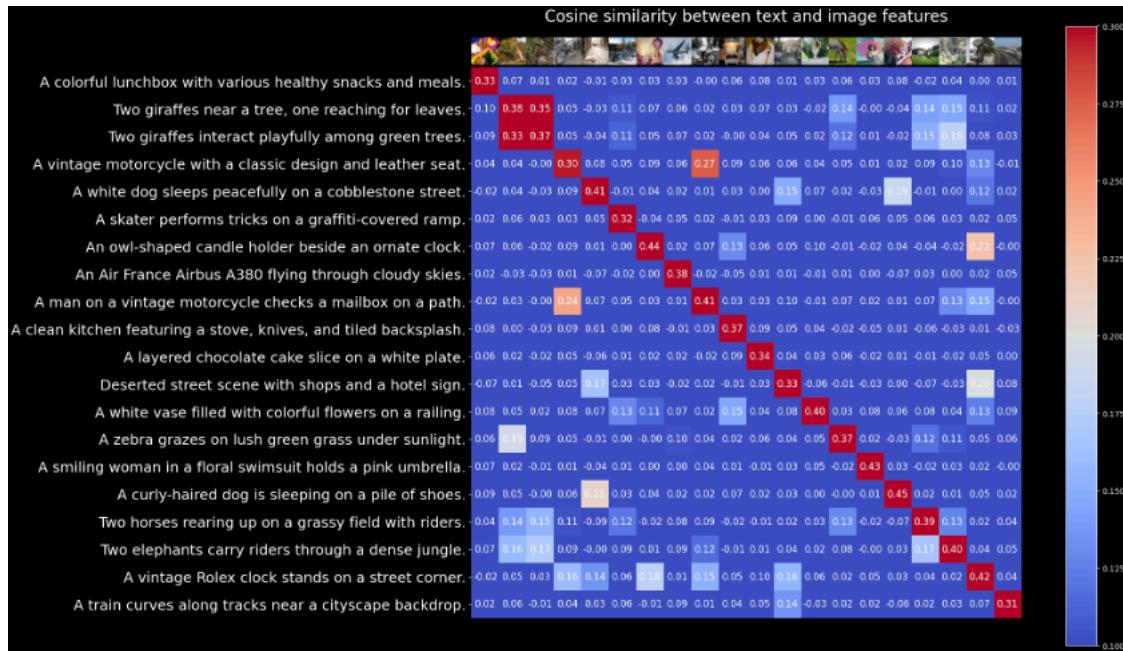
# 제목 추가
plt.title("텍스트와 이미지 특징 간의 코사인 유사도", size=20)
```

- Text(0.5, 1.0, '텍스트와 이미지 특징 간의 코사인 유사도')



- 한글로 이미지 설명 작성, 중간 이미지 작성 설명 실패 등의 영향 고려 필요

- 비교: 교재의 유사도 매트릭스



- **VectorStore** 생성, 이미지 추가

```
from langchain.chroma import Chroma
```

```
# DB 생성
image_db = Chroma(
    collection_name="multimodal",
    embedding_function=image_embedding_function,
)
```

```
# 이미지 추가  
image_db.add_images(uris=image_uris)
```

- 셀 출력 (2m 6.4s)

```
[ '43eb53e6-7ac3-4b2d-94ff-30b4a6acbb37',
  '84b60d2a-ee2e-4ec7-b005-0b1f8ad09dde',
  '6878ae9b-e0cd-4311-8bab-4bc4d1b4352b',
  '48b516d3-7df0-4f19-acb3-48a7092cc22e',
  'b96d8788-68a2-43ad-8bac-8c0ba6b71f63',
  '9cf017cd-c49e-4563-bfd6-9a42c5ba479b',
  '5b3eee5f-cb66-4311-b5dc-7b4eb12c675b',
  'cf810072-f574-41b1-a5f0-07820319357c',
  '6e9ac923-8432-4f2f-aed3-67b21b490f6f',
  '441b617e-ca91-4916-b7c7-de1545c33388',
  '16e4703d-c038-49f7-9f78-909cdb2be2ee',
  'afae8326-3a9b-4617-a0ef-7406315763fb',
  'fe8f3586-2835-4158-9fc3-e1ed5556d27b',
  '2168e3e6-fc3c-4686-9026-20b9a235594d',
  '048cc56c-9d79-42ec-a88b-acc529074f3a',
  '8fbca34e-6693-4fe8-b38a-1f4874e65b66',
  '1a6fe45e-9179-408b-a023-58aaf8a25b0d',
  'd17d9fb9-97a6-445f-b1a1-e4bde4295ff7',
  '047b9aaa-cb76-4344-b890-669608cb6e18',
  '9d4d1666-bc1a-4e70-b594-4621f9a62a69']
```

- 이미지 검색 결과를 통해 이미지로 출력하기 위해 생성한 `helper class`

```
import base64
import io
from PIL import Image
from IPython.display import HTML, display
from langchain.schema import Document

class ImageRetriever:
    def __init__(self, retriever):
        """
        이미지 검색기를 초기화합니다.

        인자:
        retriever: LangChain의 retriever 객체
        """
        self.retriever = retriever

    def invoke(self, query):
        """
        쿼리를 사용하여 이미지를 검색하고 표시합니다.

        인자:
        query (str): 검색 쿼리
        """
        docs = self.retriever.invoke(query)
        if docs and isinstance(docs[0], Document):
            self.plt_img_base64(docs[0].page_content)
        else:
            print("검색된 이미지가 없습니다.")
        return docs

    @staticmethod
    def resize_base64_image(base64_string, size=(224, 224)):
        """
        Base64 문자열로 인코딩된 이미지의 크기를 조정합니다.

        인자:
        base64_string (str): 원본 이미지의 Base64 문자열.
        size (tuple): (너비, 높이)로 표현된 원하는 이미지 크기.
        """
        img_data = base64.b64decode(base64_string)
        img = Image.open(io.BytesIO(img_data))
        resized_img = img.resize(size, Image.LANCZOS)
        buffered = io.BytesIO()
        resized_img.save(buffered, format=img.format)
        return base64.b64encode(buffered.getvalue()).decode("utf-8")

    @staticmethod
    def plt_img_base64(img_base64):
        """
```

Base64로 인코딩된 이미지를 표시합니다.

인자:

```
img_base64 (str): Base64로 인코딩된 이미지 문자열
"""
image_html = f''
display(HTML(image_html))
```

- **Image Retriever** 생성

```
# Image Retriever 생성
retriever = image_db.as_retriever(search_kwargs={"k": 3})
image_retriever = ImageRetriever(retriever)
```

- **Image** 조회해보기

```
# 이미지 조회_1
result = image_retriever.invoke("A Dog on the street")
```

- 셀 출력 (0.4s): **"A Dog on the street"**



```
# 이미지 조회_2
result = image_retriever.invoke("Motorcycle with a man")
```

- 셀 출력 (0.4s): **"Motorcycle with a man"**

