rb-121450094

May 28, 2024

1 TUGAS TEKNOLOGI BASIS DATA

Nama: Syifa Firnanda
NIM: 121450094
Kelas: RB

- Title: Three Ways of Storing and Accessing Lots of Images in Python
- URL: https://realpython.com/storing-images-in-python/ Description:
- Re-implementaikan artikel tersebut dan buatlah laporan dalam bentuk markdown
- Gunakan bahasa yang mudah dimengerti dalm menjelaskan setiap Langkah yang anda kerjakan
- File format: markdown dengan nama "Class-NIM.md" Contoh: "RA- 1234.md"
- Media pengumpulan Github Dokumen Sains Data | 2023-Institut Teknologi Sumatera.

1.1 Setup

Langkah ini merupakan langkah untuk mengakses data yang terdapat pada penyimpanan lokal komputer. Program ini memuat dataset CIFAR-10 dari beberapa batch file dalam direktori komputer. CIFAR-10 adalah dataset gambar yang sering digunakan untuk pelatihan model machine learning dan deep learning. Program akan membaca data dari setiap batch file menggunakan fungsi `unpickle`, lalu mengonversi data gambar yang diflatkan menjadi format gambar asli dengan ukuran 32x32 piksel dan tiga saluran warna (RGB). Setelah itu, program menyimpan gambar yang telah dikonversi dan labelnya masing-masing ke dalam dua daftar, `images` dan `labels`. Setelahnya, program mencetak ukuran (shape) dari daftar gambar dan label guna memastikan data telah berhasil dimuat.

Berdasarkan hasil output program, diketahui bahwa: - np.shape(images) (50000, 32, 32, 3): berarti daftar images berisi 50000 gambar. - np.shape(labels) (50000,): berarti bahwa daftar labels berisi 50000 label.

[36]:
```
import numpy as np
import pickle
from pathlib import Path

# Path to the unzipped CIFAR data
data_dir = Path("C:/Users/Syifa Firnanda/Downloads/cifar-10-batches-py")

# Unpickle function provided by the CIFAR hosts
def unpickle(file):
```

```
    with open(file, "rb") as fo:
        dict = pickle.load(fo, encoding="bytes")
    return dict

images, labels = [], []
for batch in data_dir.glob("data_batch_*"):
    batch_data = unpickle(batch)
    for i, flat_im in enumerate(batch_data[b"data"]):
        im_channels = []
        # Each image is flattened, with channels in order of R, G, B
        for j in range(3):
            im_channels.append(
                flat_im[j * 1024 : (j + 1) * 1024].reshape((32, 32))
            )
        # Reconstruct the original image
        images.append(np.dstack((im_channels)))
        # Save the label
        labels.append(batch_data[b"labels"][i])

print("Loaded CIFAR-10 training set:")
print(f" - np.shape(images)     {np.shape(images)}")
print(f" - np.shape(labels)     {np.shape(labels)}")
```

```
Loaded CIFAR-10 training set:
 - np.shape(images)     (50000, 32, 32, 3)
 - np.shape(labels)     (50000,)
```

1.1.1 Setup for Storing Images on Disk

Langkah ini melakukan instalasi beberapa hal untuk membangun environment agar dapat melakukan penyimpanan dan pengaksesan gambar-gambar dalam disk, seperti Pillow untuk memanipulasi gambar, LMDB (Lightning Memory-Mapped Database), dan HDF5 (Hierarchical Data Format).

[37]:
```
pip install Pillow
```

```
Requirement already satisfied: Pillow in c:\users\syifa
firnanda\anaconda3\lib\site-packages (9.4.0)
Note: you may need to restart the kernel to use updated packages.
```

[4]:
```
pip install lmdb
```

```
Collecting lmdbNote: you may need to restart the kernel to use updated packages.

  Downloading lmdb-1.4.1-cp310-cp310-win_amd64.whl (100 kB)
     ---------------------------------------- 100.1/100.1 kB 21.5 kB/s eta 0:00:00
Installing collected packages: lmdb
Successfully installed lmdb-1.4.1
```

[34]:
```
conda install -c conda-forge pillow
```

```
Note: you may need to restart the kernel to use updated packages.

usage: conda-script.py [-h] [-V] command …
conda-script.py: error: unrecognized arguments: pillow
```

1.1.2 Getting Started With HDF5

[35]:
```
pip install h5py
```

```
Requirement already satisfied: h5py in c:\users\syifa
firnanda\anaconda3\lib\site-packages (3.7.0)
Requirement already satisfied: numpy>=1.14.5 in c:\users\syifa
firnanda\anaconda3\lib\site-packages (from h5py) (1.23.5)
Note: you may need to restart the kernel to use updated packages.
```

1.2 Storing a Single Image

Langkah ini bertujuan untuk menyiapkan direktori penyimpanan untuk gambar yang akan disimpan dalam tiga format yang berbeda, yaitu disk (sebagai file biasa), LMDB, dan HDF5.

[6]:
```
from pathlib import Path

disk_dir = Path("data/disk/")
lmdb_dir = Path("data/lmdb/")
hdf5_dir = Path("data/hdf5/")
```

[9]:
```
disk_dir.mkdir(parents=True, exist_ok=True)
lmdb_dir.mkdir(parents=True, exist_ok=True)
hdf5_dir.mkdir(parents=True, exist_ok=True)
```

1.2.1 Storing to disk

Pada langkah ini program menyimpan sebuah gambar dan labelnya ke dalam format file pada disk. Gambar disimpan sebagai file `.png`, sementara labelnya disimpan sebagai file `.csv`. Fungsi `store_single_disk` mengambil tiga parameter: `image` (array gambar berukuran 32x32x3), `image_id` (ID unik untuk gambar), dan `label` (label gambar). Gambar disimpan di direktori yang ditentukan dengan nama file berdasarkan `image_id`, dan label disimpan dalam file `.csv` yang terpisah di direktori yang sama.

[11]:
```
from PIL import Image
import csv

def store_single_disk(image, image_id, label):
    """ Stores a single image as a .png file on disk.
        Parameters:
        ---------------
```

```
        image       image array, (32, 32, 3) to be stored
        image_id    integer unique ID for image
        label       image label
    """
    Image.fromarray(image).save(disk_dir / f"{image_id}.png")

    with open(disk_dir / f"{image_id}.csv", "wt") as csvfile:
        writer = csv.writer(
            csvfile, delimiter=" ", quotechar="|", quoting=csv.QUOTE_MINIMAL
        )
        writer.writerow([label])
```

1.2.2 Storing to LMDB

Hal yang sama dilakukan dengan menggunakan LMDB. Program akan menyimpan sebuah gambar dan labelnya ke dalam database LMDB. Fungsi store_single_lmdb mengambil tiga parameter, yaitu image (array gambar berukuran 32x32x3), image_id (ID unik untuk gambar), dan label (label gambar).

[12]:
```
class CIFAR_Image:
    def __init__(self, image, label):
        # Dimensions of image for reconstruction - not really necessary
        # for this dataset, but some datasets may include images of
        # varying sizes
        self.channels = image.shape[2]
        self.size = image.shape[:2]

        self.image = image.tobytes()
        self.label = label

    def get_image(self):
        """ Returns the image as a numpy array. """
        image = np.frombuffer(self.image, dtype=np.uint8)
        return image.reshape(*self.size, self.channels)
```

[13]:
```
import lmdb
import pickle

def store_single_lmdb(image, image_id, label):
    """ Stores a single image to a LMDB.
        Parameters:
        ---------------
        image       image array, (32, 32, 3) to be stored
        image_id    integer unique ID for image
        label       image label
    """
    map_size = image.nbytes * 10
```

```
    # Create a new LMDB environment
    env = lmdb.open(str(lmdb_dir / f"single_lmdb"), map_size=map_size)

    # Start a new write transaction
    with env.begin(write=True) as txn:
        # All key-value pairs need to be strings
        value = CIFAR_Image(image, label)
        key = f"{image_id:08}"
        txn.put(key.encode("ascii"), pickle.dumps(value))
    env.close()
```

1.2.3 Storing with HDF5

Kemudian dilakukan hal yang sama juga dengan menggunakan HDF5. Program akan menyimpan sebuah gambar dan labelnya ke dalam file HDF5. Fungsi store_single_hdf5 menerima tiga parameter: image (array gambar berukuran 32x32x3), image_id (ID unik untuk gambar), dan label (label gambar).

[14]:
```
import h5py

def store_single_hdf5(image, image_id, label):
    """ Stores a single image to an HDF5 file.
        Parameters:
        ---------------
        image       image array, (32, 32, 3) to be stored
        image_id    integer unique ID for image
        label       image label
    """
    # Create a new HDF5 file
    file = h5py.File(hdf5_dir / f"{image_id}.h5", "w")

    # Create a dataset in the file
    dataset = file.create_dataset(
        "image", np.shape(image), h5py.h5t.STD_U8BE, data=image
    )
    meta_set = file.create_dataset(
        "meta", np.shape(label), h5py.h5t.STD_U8BE, data=label
    )
    file.close()
```

1.2.4 Experiment single storage

Program berikut ini bertujuan untuk mengukur waktu yang dibutuhkan dalam melakukan penyimpanan sebuah gambar dan labelnya menggunakan tiga metode penyimpanan yang berbeda, yaitu disk, LMDB, dan HDF5.

Berdasarkan hasil yang didapatkan, diketahui bahwa metode LMDB memili waktu eksekusi yang

lebih cepat (0.005552500020712614) daripada metode HDF5 (0.029452199989464134) dan metode disk (0.09228749998146668).

[38]:
```
_store_single_funcs = dict(
    disk=store_single_disk, lmdb=store_single_lmdb, hdf5=store_single_hdf5
)
```

[16]:
```
from timeit import timeit

store_single_timings = dict()

for method in ("disk", "lmdb", "hdf5"):
    t = timeit(
        "_store_single_funcs[method](image, 0, label)",
        setup="image=images[0]; label=labels[0]",
        number=1,
        globals=globals(),
    )
    store_single_timings[method] = t
    print(f"Method: {method}, Time usage: {t}")
```

```
Method: disk, Time usage: 0.09228749998146668
Method: lmdb, Time usage: 0.005552500020712614
Method: hdf5, Time usage: 0.029452199989464134
```

1.3 Storage many image

1.3.1 adjusting the code for many image

Langkah ini bertujuan untuk menyimpan sejumlah besar gambar beserta labelnya ke dalam format yang berbeda sesuai dengan kebutuhan atau preferensinya.

[17]:
```
def store_many_disk(images, labels):
    """ Stores an array of images to disk
        Parameters:
        ---------------
        images       images array, (N, 32, 32, 3) to be stored
        labels       labels array, (N, 1) to be stored
    """
    num_images = len(images)

    # Save all the images one by one
    for i, image in enumerate(images):
        Image.fromarray(image).save(disk_dir / f"{i}.png")

    # Save all the labels to the csv file
    with open(disk_dir / f"{num_images}.csv", "w") as csvfile:
        writer = csv.writer(
            csvfile, delimiter=" ", quotechar="|", quoting=csv.QUOTE_MINIMAL
```

```
        )
        for label in labels:
            # This typically would be more than just one value per row
            writer.writerow([label])

def store_many_lmdb(images, labels):
    """ Stores an array of images to LMDB.
        Parameters:
        ---------------
        images       images array, (N, 32, 32, 3) to be stored
        labels       labels array, (N, 1) to be stored
    """
    num_images = len(images)

    map_size = num_images * images[0].nbytes * 10

    # Create a new LMDB DB for all the images
    env = lmdb.open(str(lmdb_dir / f"{num_images}_lmdb"), map_size=map_size)

    # Same as before - but let's write all the images in a single transaction
    with env.begin(write=True) as txn:
        for i in range(num_images):
            # All key-value pairs need to be Strings
            value = CIFAR_Image(images[i], labels[i])
            key = f"{i:08}"
            txn.put(key.encode("ascii"), pickle.dumps(value))
    env.close()

def store_many_hdf5(images, labels):
    """ Stores an array of images to HDF5.
        Parameters:
        ---------------
        images       images array, (N, 32, 32, 3) to be stored
        labels       labels array, (N, 1) to be stored
    """
    num_images = len(images)

    # Create a new HDF5 file
    file = h5py.File(hdf5_dir / f"{num_images}_many.h5", "w")

    # Create a dataset in the file
    dataset = file.create_dataset(
        "images", np.shape(images), h5py.h5t.STD_U8BE, data=images
    )
    meta_set = file.create_dataset(
        "meta", np.shape(labels), h5py.h5t.STD_U8BE, data=labels
    )
```

```
    file.close()
```

1.3.2 Preparing Dataset

Langkah ini bertujuan untuk mempersiapkan data hingga 100000 gambar.

[18]:
```
cutoffs = [10, 100, 1000, 10000, 100000]

# Let's double our images so that we have 100,000
images = np.concatenate((images, images), axis=0)
labels = np.concatenate((labels, labels), axis=0)

# Make sure you actually have 100,000 images and labels
print(np.shape(images))
print(np.shape(labels))
```

```
(100000, 32, 32, 3)
(100000,)
```

1.3.3 Experiment for Many Storage Dataset

Program berikut bertujuan untuk mengukur dan mencatat waktu yang dibutuhkan guna menyimpan berbagai jumlah gambar dan label menggunakan tiga metode penyimpanan yang berbeda, yaitu disk, LMDB, dan HDF5.

Berdasarakan hasil yang didapatkan, diketahui bahwa waktu yang dibutuhkan metode LMDB adalah yang paling cepat, diikuti oleh HDF5, dan metode disk adalah yang paling lambat.

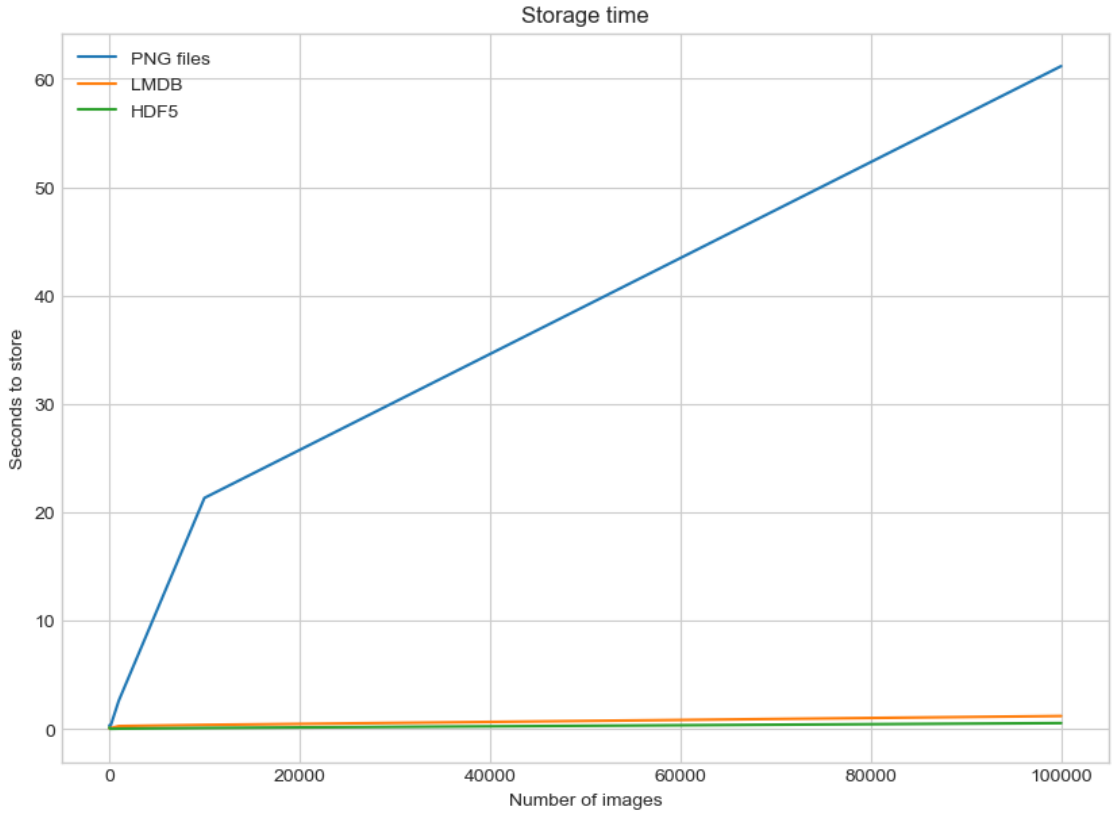
[39]:
```
_store_many_funcs = dict(
    disk=store_many_disk, lmdb=store_many_lmdb, hdf5=store_many_hdf5
)

from timeit import timeit

store_many_timings = {"disk": [], "lmdb": [], "hdf5": []}

for cutoff in cutoffs:
    for method in ("disk", "lmdb", "hdf5"):
        t = timeit(
            "_store_many_funcs[method](images_, labels_)",
            setup="images_=images[:cutoff]; labels_=labels[:cutoff]",
            number=1,
            globals=globals(),
        )
        store_many_timings[method].append(t)

        # Print out the method, cutoff, and elapsed time
        print(f"Method: {method}, Time usage: {t}")
```

```
Method: disk, Time usage: 0.33790660003433004
Method: lmdb, Time usage: 0.0764377000159584
Method: hdf5, Time usage: 0.11575830000219867
Method: disk, Time usage: 0.17462219996377826
Method: lmdb, Time usage: 0.0703932999749668
Method: hdf5, Time usage: 0.02166530000977218
Method: disk, Time usage: 2.609621200012043
Method: lmdb, Time usage: 0.27234099997440353
Method: hdf5, Time usage: 0.036609900009352714
Method: disk, Time usage: 21.317625700030476
Method: lmdb, Time usage: 0.3838267999817617
Method: hdf5, Time usage: 0.09900480002397671
Method: disk, Time usage: 61.15604639996309
Method: lmdb, Time usage: 1.2013721999828704
Method: hdf5, Time usage: 0.5469745000009425
```

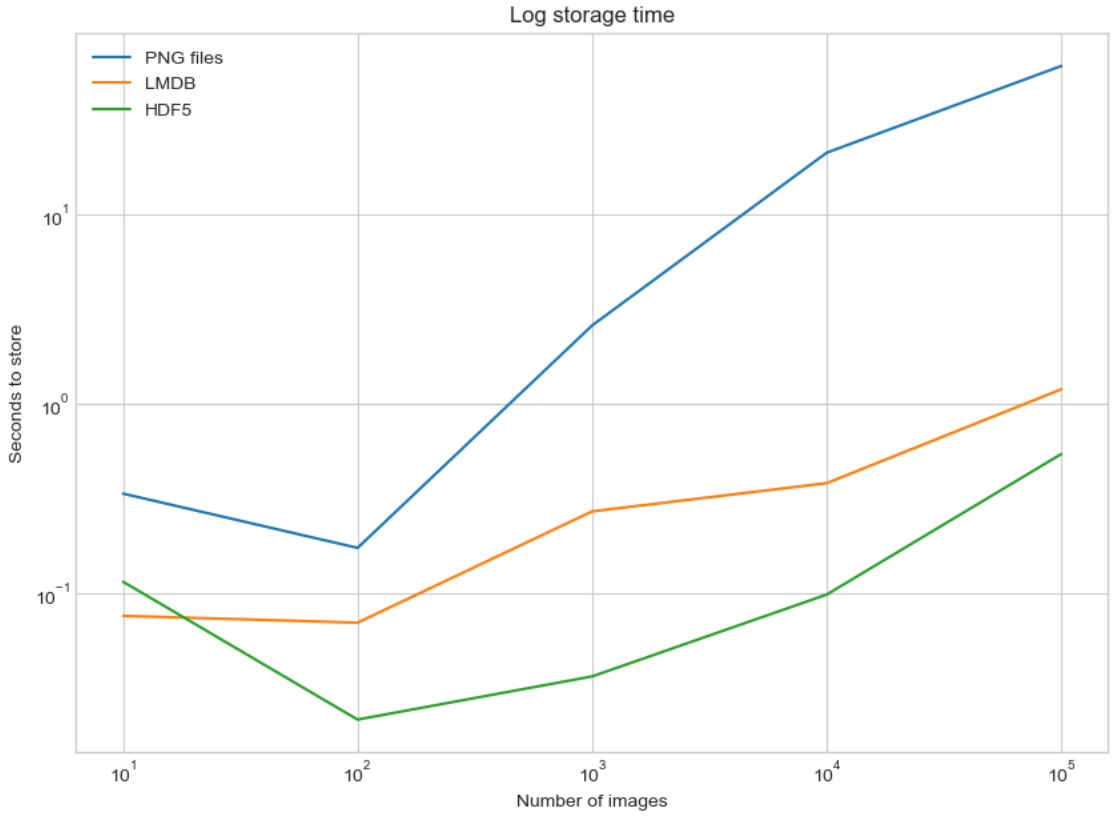
[40]:
```python
import matplotlib.pyplot as plt

def plot_with_legend(
    x_range, y_data, legend_labels, x_label, y_label, title, log=False
):
    """ Displays a single plot with multiple datasets and matching legends.
        Parameters:
        --------------
        x_range         list of lists containing x data
        y_data          list of lists containing y values
        legend_labels   list of string legend labels
        x_label         x axis label
        y_label         y axis label
    """
    plt.style.use("seaborn-whitegrid")
    plt.figure(figsize=(10, 7))

    if len(y_data) != len(legend_labels):
        raise TypeError(
            "Error: number of data sets does not match number of labels."
        )

    all_plots = []
    for data, label in zip(y_data, legend_labels):
        if log:
            temp, = plt.loglog(x_range, data, label=label)
        else:
            temp, = plt.plot(x_range, data, label=label)
        all_plots.append(temp)

    plt.title(title)
```

```
    plt.xlabel(x_label)
    plt.ylabel(y_label)
    plt.legend(handles=all_plots)
    plt.show()

# Getting the store timings data to display
disk_x = store_many_timings["disk"]
lmdb_x = store_many_timings["lmdb"]
hdf5_x = store_many_timings["hdf5"]

plot_with_legend(
    cutoffs,
    [disk_x, lmdb_x, hdf5_x],
    ["PNG files", "LMDB", "HDF5"],
    "Number of images",
    "Seconds to store",
    "Storage time",
    log=False,
)

plot_with_legend(
    cutoffs,
    [disk_x, lmdb_x, hdf5_x],
    ["PNG files", "LMDB", "HDF5"],
    "Number of images",
    "Seconds to store",
    "Log storage time",
    log=True,
)
```

```
C:\Users\Syifa Firnanda\AppData\Local\Temp\ipykernel_15036\2568719458.py:15:
MatplotlibDeprecationWarning: The seaborn styles shipped by Matplotlib are
deprecated since 3.6, as they no longer correspond to the styles shipped by
seaborn. However, they will remain available as 'seaborn-v0_8-<style>'.
Alternatively, directly use the seaborn API instead.
  plt.style.use("seaborn-whitegrid")
```




```
C:\Users\Syifa Firnanda\AppData\Local\Temp\ipykernel_15036\2568719458.py:15:
MatplotlibDeprecationWarning: The seaborn styles shipped by Matplotlib are
deprecated since 3.6, as they no longer correspond to the styles shipped by
seaborn. However, they will remain available as 'seaborn-v0_8-<style>'.
Alternatively, directly use the seaborn API instead.
  plt.style.use("seaborn-whitegrid")
```



Berdasarkan visualisasi di atas, dapat diketahui bahwa untuk grafik "Storage Time" metode LMDB dan HDF5 menghasilkan waktu penyimpanan yang lebih baik dari pada PNG File, begitu juga untuk hasil yang tergambarkan pada grafik "Log storage time".

1.4 Reading Single Image

Ketiga program berikut ini bertujuan untuk membaca dan mengembalikan gambar beserta labelnya dari tiga format penyimpanan yang berbeda: disk, LMDB, dan HDF5.

1.4.1 Reading from Disk

[41]:
```
def read_single_disk(image_id):
    """ Stores a single image to disk.
        Parameters:
        ---------------
        image_id    integer unique ID for image

        Returns:
        ----------
        image       image array, (32, 32, 3) to be stored
        label       associated meta data, int label
    """
```

```
    image = np.array(Image.open(disk_dir / f"{image_id}.png"))

    with open(disk_dir / f"{image_id}.csv", "r") as csvfile:
        reader = csv.reader(
            csvfile, delimiter=" ", quotechar="|", quoting=csv.QUOTE_MINIMAL
        )
        label = int(next(reader)[0])

    return image, label
```

1.4.2 Reading from LMDB

[43]:
```
def read_single_lmdb(image_id):
    """ Stores a single image to LMDB.
        Parameters:
        ---------------
        image_id    integer unique ID for image

        Returns:
        ----------
        image       image array, (32, 32, 3) to be stored
        label       associated meta data, int label
    """
    # Open the LMDB environment
    env = lmdb.open(str(lmdb_dir / f"single_lmdb"), readonly=True)

    # Start a new read transaction
    with env.begin() as txn:
        # Encode the key the same way as we stored it
        data = txn.get(f"{image_id:08}".encode("ascii"))
        # Remember it's a CIFAR_Image object that is loaded
        cifar_image = pickle.loads(data)
        # Retrieve the relevant bits
        image = cifar_image.get_image()
        label = cifar_image.label
    env.close()

    return image, label
```

1.4.3 Reading from HDF5

[44]:
```
def read_single_hdf5(image_id):
    """ Stores a single image to HDF5.
        Parameters:
        ---------------
        image_id    integer unique ID for image
```

```
        Returns:
        ----------
        image       image array, (32, 32, 3) to be stored
        label       associated meta data, int label
    """
    # Open the HDF5 file
    file = h5py.File(hdf5_dir / f"{image_id}.h5", "r+")

    image = np.array(file["/image"]).astype("uint8")
    label = int(np.array(file["/meta"]).astype("uint8"))

    return image, label
```

[24]:
```
_read_single_funcs = dict(
    disk=read_single_disk, lmdb=read_single_lmdb, hdf5=read_single_hdf5
)
```

1.4.4 Experiment for Reading a Single Image

Program berikut ini bertujuan untuk mengukur waktu yang dibutuhkan dalam membaca satu gambar dan labelnya menggunakan tiga metode penyimpanan yang berbeda: disk, LMDB, dan HDF5.

Berdasarkan hasil yang didapatkan, diketahui bahwa metode LMDB (0.017994800000451505) adalah metode yang paling efisien dalam hal kecepatan untuk membaca data dalam kasus ini dibandingkan dengan metode disk (0.0492454000050202) dan HDF5 (0.06455030001234263).

[45]:
```
from timeit import timeit

read_single_timings = dict()

for method in ("disk", "lmdb", "hdf5"):
    t = timeit(
        "_read_single_funcs[method](0)",
        setup="image=images[0]; label=labels[0]",
        number=1,
        globals=globals(),
    )
    read_single_timings[method] = t
    print(f"Method: {method}, Time usage: {t}")
```

```
Method: disk, Time usage: 0.0492454000050202
Method: lmdb, Time usage: 0.017994800000451505
Method: hdf5, Time usage: 0.06455030001234263
```

1.5 Reading Many Image

1.5.1 Adjusting the Code for Many Images

Program berikut ini bertujuan untuk mendefinisikan tiga fungsi guna membaca banyak gambar dan label dari tiga format penyimpanan yang berbeda, yaitu disk, LMDB, dan HDF5. Setiap fungsi menerima jumlah gambar yang ingin dibaca dan mengembalikan array gambar dan label yang sesuai.

[46]:
```
def read_many_disk(num_images):
    """ Reads image from disk.
        Parameters:
        ---------------
        num_images   number of images to read

        Returns:
        ----------
        images      images array, (N, 32, 32, 3) to be stored
        labels      associated meta data, int label (N, 1)
    """
    images, labels = [], []

    # Loop over all IDs and read each image in one by one
    for image_id in range(num_images):
        images.append(np.array(Image.open(disk_dir / f"{image_id}.png")))

    with open(disk_dir / f"{num_images}.csv", "r") as csvfile:
        reader = csv.reader(
            csvfile, delimiter=" ", quotechar="|", quoting=csv.QUOTE_MINIMAL
        )
        for row in reader:
            labels.append(int(row[0]))
    return images, labels

def read_many_lmdb(num_images):
    """ Reads image from LMDB.
        Parameters:
        ---------------
        num_images   number of images to read

        Returns:
        ----------
        images      images array, (N, 32, 32, 3) to be stored
        labels      associated meta data, int label (N, 1)
    """
    images, labels = [], []
    env = lmdb.open(str(lmdb_dir / f"{num_images}_lmdb"), readonly=True)
```

```
    # Start a new read transaction
    with env.begin() as txn:
        # Read all images in one single transaction, with one lock
        # We could split this up into multiple transactions if needed
        for image_id in range(num_images):
            data = txn.get(f"{image_id:08}".encode("ascii"))
            # Remember that it's a CIFAR_Image object
            # that is stored as the value
            cifar_image = pickle.loads(data)
            # Retrieve the relevant bits
            images.append(cifar_image.get_image())
            labels.append(cifar_image.label)
    env.close()
    return images, labels

def read_many_hdf5(num_images):
    """ Reads image from HDF5.
        Parameters:
        ---------------
        num_images   number of images to read

        Returns:
        ----------
        images      images array, (N, 32, 32, 3) to be stored
        labels      associated meta data, int label (N, 1)
    """
    images, labels = [], []

    # Open the HDF5 file
    file = h5py.File(hdf5_dir / f"{num_images}_many.h5", "r+")

    images = np.array(file["/images"]).astype("uint8")
    labels = np.array(file["/meta"]).astype("uint8")

    return images, labels

_read_many_funcs = dict(
    disk=read_many_disk, lmdb=read_many_lmdb, hdf5=read_many_hdf5
)
```

1.5.2 Experiment for Reading Many Images

Program berikut ini berfungsi untuk mengetahui tentang waktu yang diperlukan untuk membaca sejumlah gambar dari berbagai format penyimpanan (disk, LMDB, dan HDF5).

Berdasarkan hasil yang didapatkan, diketahui bahwa HDF5 cenderung memiliki waktu yang sedikit lebih kecil dibandingkan dengan disk dan LMDB. Metode LMDB sering kali sedikit lebih cepat dibandingkan dengan disk, namun perbedaannya sangat kecil.

[48]:
```
from timeit import timeit

read_many_timings = {"disk": [], "lmdb": [], "hdf5": []}

for cutoff in cutoffs:
    for method in ("disk", "lmdb", "hdf5"):
        t = timeit(
            "_read_many_funcs[method](num_images)",
            setup="num_images=cutoff",
            number=0,
            globals=globals(),
        )
        read_many_timings[method].append(t)

        # Print out the method, cutoff, and elapsed time
        print(f"Method: {method}, No. images: {cutoff}, Time usage: {t}")
```

```
Method: disk, No. images: 10, Time usage: 3.00002284348011e-07
Method: lmdb, No. images: 10, Time usage: 2.9994407668709755e-07
Method: hdf5, No. images: 10, Time usage: 2.00001522898674e-07
Method: disk, No. images: 100, Time usage: 2.00001522898674e-07
Method: lmdb, No. images: 100, Time usage: 1.00000761449337e-07
Method: hdf5, No. images: 100, Time usage: 1.00000761449337e-07
Method: disk, No. images: 1000, Time usage: 2.00001522898674e-07
Method: lmdb, No. images: 1000, Time usage: 1.00000761449337e-07
Method: hdf5, No. images: 1000, Time usage: 2.00001522898674e-07
Method: disk, No. images: 10000, Time usage: 2.00001522898674e-07
Method: lmdb, No. images: 10000, Time usage: 1.00000761449337e-07
Method: hdf5, No. images: 10000, Time usage: 9.994255378842354e-08
Method: disk, No. images: 100000, Time usage: 2.00001522898674e-07
Method: lmdb, No. images: 100000, Time usage: 2.00001522898674e-07
Method: hdf5, No. images: 100000, Time usage: 1.00000761449337e-07
```

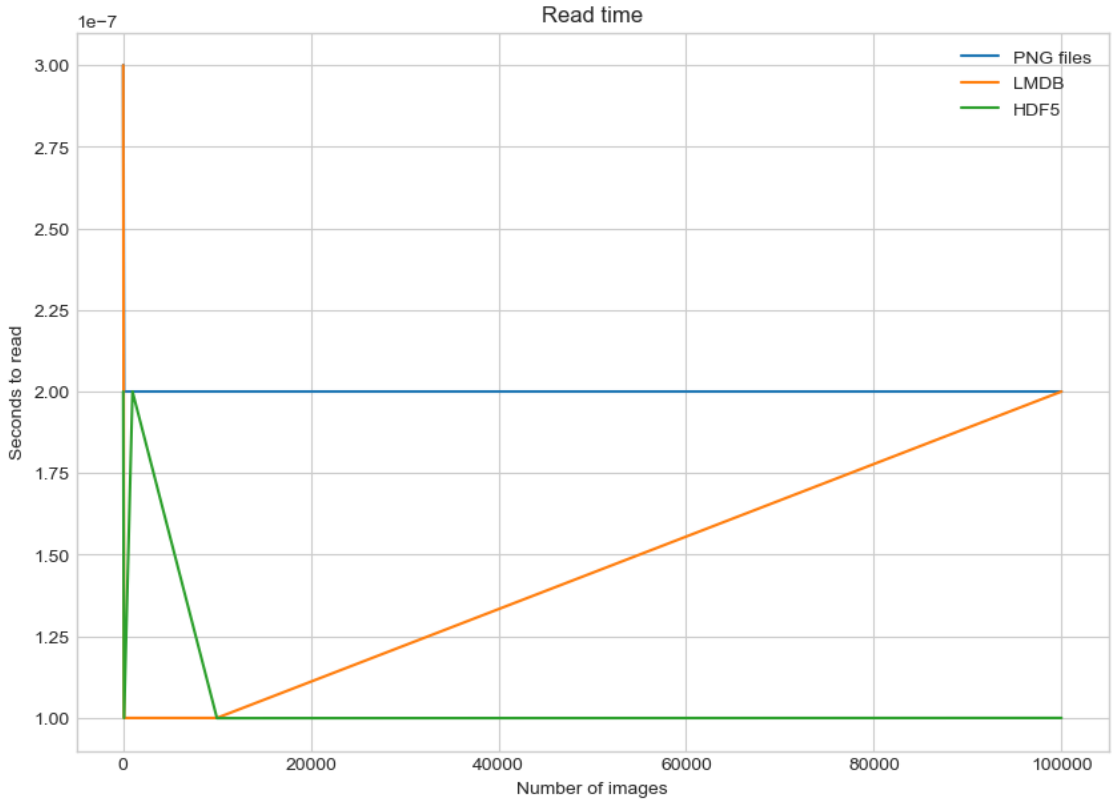
1.5.3 Plot the Read Timing

Berikut ini adalah program yang bertujuan untuk memvisualisasikan waktu yang dibutuhkan untuk membaca data gambar.

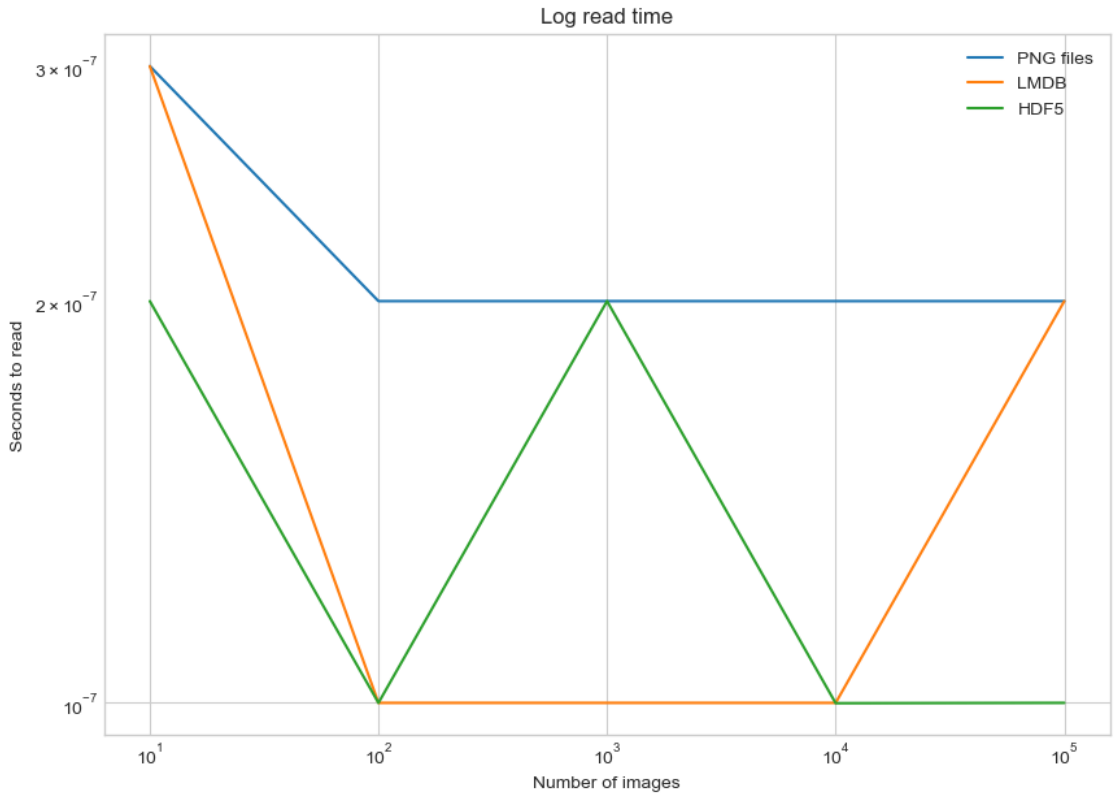
[49]:
```
# Extract read timing data for each method
disk_read_times = read_many_timings["disk"]
lmdb_read_times = read_many_timings["lmdb"]
hdf5_read_times = read_many_timings["hdf5"]

# Plot read times without log scale
plot_with_legend(
    cutoffs,
    [disk_read_times, lmdb_read_times, hdf5_read_times],
    ["PNG files", "LMDB", "HDF5"],
```

```
    "Number of images",
    "Seconds to read",
    "Read time",
    log=False,
)

# Plot read times with log scale
plot_with_legend(
    cutoffs,
    [disk_read_times, lmdb_read_times, hdf5_read_times],
    ["PNG files", "LMDB", "HDF5"],
    "Number of images",
    "Seconds to read",
    "Log read time",
    log=True,
)

# Plot both read and write times
plot_with_legend(
    cutoffs,
    [disk_read_times, lmdb_read_times, hdf5_read_times, disk_x, lmdb_x, hdf5_x],
    [
        "Read PNG",
        "Read LMDB",
        "Read HDF5",
        "Write PNG",
        "Write LMDB",
        "Write HDF5",
    ],
    "Number of images",
    "Seconds",
    "Store and Read Times",
    log=False,
)
```

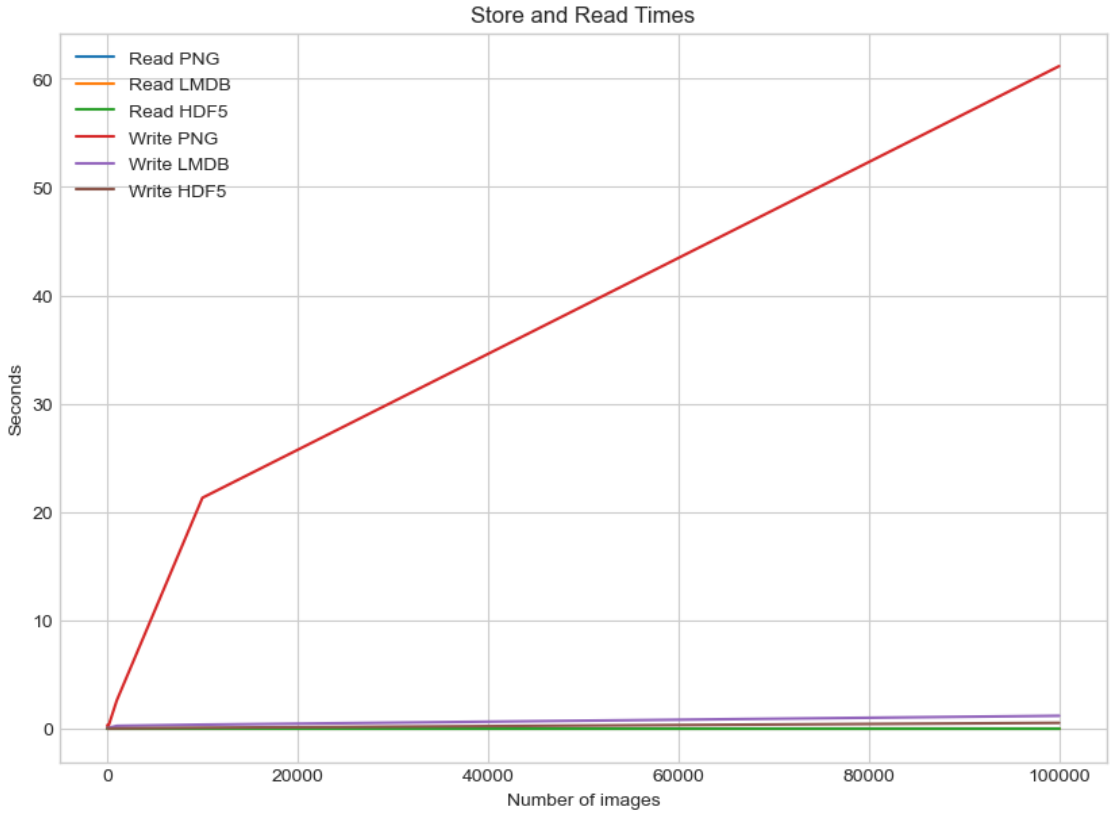
```
C:\Users\Syifa Firnanda\AppData\Local\Temp\ipykernel_15036\2568719458.py:15:
MatplotlibDeprecationWarning: The seaborn styles shipped by Matplotlib are
deprecated since 3.6, as they no longer correspond to the styles shipped by
seaborn. However, they will remain available as 'seaborn-v0_8-<style>'.
Alternatively, directly use the seaborn API instead.
  plt.style.use("seaborn-whitegrid")
```




```
C:\Users\Syifa Firnanda\AppData\Local\Temp\ipykernel_15036\2568719458.py:15:
MatplotlibDeprecationWarning: The seaborn styles shipped by Matplotlib are
deprecated since 3.6, as they no longer correspond to the styles shipped by
seaborn. However, they will remain available as 'seaborn-v0_8-<style>'.
Alternatively, directly use the seaborn API instead.
  plt.style.use("seaborn-whitegrid")
```




```
C:\Users\Syifa Firnanda\AppData\Local\Temp\ipykernel_15036\2568719458.py:15:
MatplotlibDeprecationWarning: The seaborn styles shipped by Matplotlib are
deprecated since 3.6, as they no longer correspond to the styles shipped by
seaborn. However, they will remain available as 'seaborn-v0_8-<style>'.
Alternatively, directly use the seaborn API instead.
  plt.style.use("seaborn-whitegrid")
```



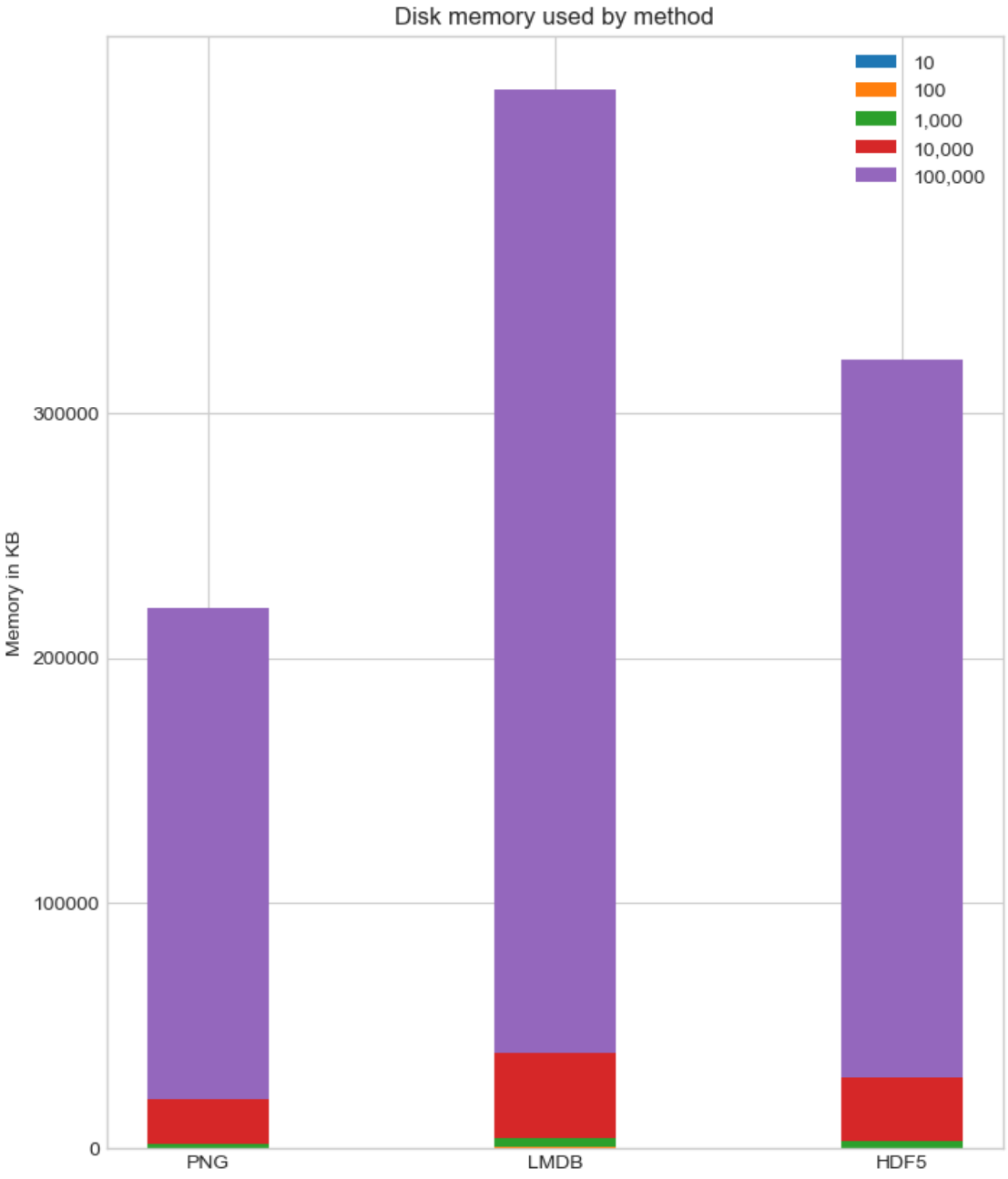
Berdasarkan 3 visualisasi grafik di atas, dapat dilihat bahwa untuk setiap metode penyimpanan data, terdapat kecenderungan bahwa kecepatan membaca (read) data lebih lambat dibandingkan dengan menulis (write) data, kecuali untuk format file PNG yang membutuhkan waktu untuk membaca file jauh lebih lama daripada menulisnya. Secara keseluruhan, format penyimpanan data HDF5 dan LMDB lebih baik dibandingkan dengan format file PNG.

1.6 Considering Disk Image

Program berikut ini bertujuan untuk memvisualisasikan penggunaan memori (dalam kilobyte) dari tiga metode penyimpanan data yang berbeda (disk, LMDB, dan HDF5) dengan jumlah data yang bervariasi (10, 100, 1.000, 10.000, dan 100.000). Dengan menggunakan pustaka matplotlib di Python, dibuat sebuah grafik batang bertumpuk yang menunjukkan penggunaan memori untuk setiap metode dan jumlah data yang berbeda. Dengan visualisasi ini dapat dengan mudah untuk membandingkan efisiensi penggunaan memori dari ketiga metode tersebut dalam konteks jumlah data yang berbeda.

[32]:
```
# Data memory usage in KB
disk_mem = [24, 204, 2004, 20032, 200296]
lmdb_mem = [60, 420, 4000, 39000, 393000]
hdf5_mem = [36, 304, 2900, 29000, 293000]

# Combine the data into a single list
```

```
memory_data = [disk_mem, lmdb_mem, hdf5_mem]

# Indices for the groups
indices = np.arange(3)
# Bar width
bar_width = 0.35

# Create a new figure
plt.figure(figsize=(8, 10))

# Plot the first set of bars
bar_plots = [plt.bar(indices, [data[0] for data in memory_data], bar_width)]

# Plot subsequent sets of bars stacked on top of the previous bars
for i in range(1, len(disk_mem)):
    bar_plots.append(
        plt.bar(
            indices,
            [data[i] for data in memory_data],
            bar_width,
            bottom=[data[i - 1] for data in memory_data]
        )
    )

# Labeling the plot
plt.ylabel("Memory in KB")
plt.title("Disk memory used by method")
plt.xticks(indices, ("PNG", "LMDB", "HDF5"))
plt.yticks(np.arange(0, 400000, 100000))

# Adding the legend
plt.legend(
    [bar[0] for bar in bar_plots],
    ("10", "100", "1,000", "10,000", "100,000")
)

# Display the plot
plt.show()
```



Berdasarkan hasil representasi penggunaan memori disk oleh tiga metode penyimpanan data yang berbeda: PNG, LMDB, dan HDF5, dengan variasi jumlah data yang berbeda (10, 100, 1.000, 10.000, dan 100.000), dapat diketahui bahwa metode PNG menggunakan memori paling sedikit jika dibandingkan LMDB dan HDF5 untuk semua variasi jumlah data. Namun, semakin banyak data yang disimpan, penggunaan memori untuk PNG juga meningkat secara signifikan. Di sisi lain, LMDB dan HDF5 memiliki penggunaan memori yang lebih tinggi dibandingkan PNG untuk jumlah data yang sedikit, tetapi peningkatannya lebih terkendali saat jumlah data bertambah.