

# Day 1: Ingest and Index Your Data

Welcome to our crash course!

In this course, you'll learn how to build intelligent systems that can understand and interact with your data.

We'll create a conversational agent that can answer questions about any GitHub repository - think of it as your personal AI assistant for documentation and code. If you know [DeepWiki](#), it's something similar, but tailored to your GitHub repo.

For that, we need to:

- Download and process data from the repo
- Put it inside a search engine
- Make the search engine available to our agent

In the first half of the course, we will focus on data preparation. Today, we will do the first part: downloading the data.

## GitHub Repo Data

On the first day, we will learn how to download and process data from any GitHub repository. We will download the data as a zip archive, process all the text data from there, and make it available for ingesting it later into a search engine.

Think of this as preparing a meal - we need to gather and prep all our ingredients (the data) before we can cook (build our AI agent).

Today, we will deal with simple cases, when documents are not large.

Tomorrow we will deal with more complex cases when documents are big and we also have code.

## Environment Setup

First, let's prepare the environment. We need Python 3.10 or higher.

We will use [uv](#) as the package manager. If you don't have [uv](#), let's install it:

```
Shell
pip install uv
```

Next, create a folder **aihero** with two subfolders:

- **course** - here you will reproduce all the examples from this email course
- **project** - here you will create your own project

Now go to **course** and run:

```
Shell
uv init
uv add requests python-frontmatter
uv add --dev jupyter
```

This will initialize an empty Python project with **uv** and install multiple libraries:

- **requests** for downloading data from GitHub
- **python-frontmatter** for parsing structured metadata in markdown files
- **jupyter** (in dev mode)

The reason we need jupyter in dev mode is because it's only used for development and experimentation, not in the final production code.

Let's start Jupyter:

```
Shell
uv run jupyter notebook
```

# Understanding Frontmatter

We will also need a library for parsing frontmatter - a popular documentation format commonly used for modern frameworks like Jekyll, Hugo, and Next.js.

It looks like this:

```
None
---
title: "Getting Started with AI"
author: "John Doe"
date: "2024-01-15"
tags: ["ai", "machine-learning", "tutorial"]
difficulty: "beginner"
---

# Getting Started with AI

This is the main content of the document written in **Markdown**.

You can include code blocks, links, and other formatting here.
```

This format is called "frontmatter". The section between the `---` markers contains YAML metadata that describes the document, while everything below is regular Markdown content. This is very useful because we can extract structured information (like title, tags, difficulty level) along with the content.

This is how we read it:

```
Python
import frontmatter

with open('example.md', 'r', encoding='utf-8') as f:
    post = frontmatter.load(f)

# Access metadata
print(post.metadata['title']) # "Getting Started with AI"
print(post.metadata['tags'])  # ["ai", "machine-learning", "tutorial"]

# Access content
print(post.content) # The markdown content without frontmatter
```

We can also get all the metadata and content at the same time using the `post.to_dict()` method.

## Sample Repositories

Now that we know how to process a single markdown file, let's find a repo with multiple files that we will use as our knowledge base.

We will work with multiple repositories:

- <https://github.com/DataTalksClub/faq> (source for <https://datatalks.club/faq/>) - FAQ for DataTalks.Club courses
- <https://github.com/evidentlyai/docs/> - docs for Evidently AI library

There are multiple ways you can download a GitHub repo.

First, you can clone it using git, then we process each file and prepare it for ingestion into our search system.

Alternatively, we can download the entire repository as a zip file and process all the content.

## Working with Zip Archives

The second option is easier and more efficient for our use case.

We don't even need to save the zip archive - we can load it into our Python process memory and extract all the data we need from there.

So the plan:

- Use `requests` for downloading the zip archive from GitHub
- Open the archive using built-in `zipfile` and `io` modules
- Iterate over all `.md` and `.mdx` files in the repo
- Collect the results into a list

Let's implement it step by step.

First, we import the necessary libraries:

```
Python
import io
import zipfile
import requests
import frontmatter
```

Next, we download the repository as a zip file. GitHub provides a convenient URL format for this:

```
Python
url = 'https://codeload.github.com/DataTalksClub/faq/zip/refs/heads/main'
resp = requests.get(url)
```

Now we process the zip file in memory without saving it to disk:

```
Python
repository_data = []

# Create a ZipFile object from the downloaded content
zf = zipfile.ZipFile(io.BytesIO(resp.content))

for file_info in zf.infolist():
    filename = file_info.filename.lower()

    # Only process markdown files
    if not filename.endswith('.md'):
        continue

    # Read and parse each file
    with zf.open(file_info) as f_in:
        content = f_in.read()
        post = frontmatter.loads(content)
        data = post.to_dict()
        data['filename'] = filename
        repository_data.append(data)

zf.close()
```

Let's look at what we got:

```
Python
print(repository_data[1])
```

Output:

```
Python
{'id': '9e508f2212',
 'question': 'Course: When does the course start?',
 'sort_order': 1,
 'content': "...'}
```

For processing Evidently docs we also need `.mdx` files (React markdown), so we can modify the code like this:

```
Python
for file_info in zf.infolist():
    filename = file_info.filename.lower()

    if not (filename.endswith('.md') or filename.endswith('.mdx')):
        continue

    # rest remains the same...
```

## Complete Implementation

Let's now put everything together into a reusable function:

```

Python
import io
import zipfile
import requests
import frontmatter

def read_repo_data(repo_owner, repo_name):
    """
    Download and parse all markdown files from a GitHub repository.

    Args:
        repo_owner: GitHub username or organization
        repo_name: Repository name

    Returns:
        List of dictionaries containing file content and metadata
    """
    prefix = 'https://codeload.github.com'
    url = f'{prefix}/{repo_owner}/{repo_name}/zip/refs/heads/main'
    resp = requests.get(url)

    if resp.status_code != 200:
        raise Exception(f"Failed to download repository: {resp.status_code}")

    repository_data = []
    zf = zipfile.ZipFile(io.BytesIO(resp.content))

    for file_info in zf.infolist():
        filename = file_info.filename
        filename_lower = filename.lower()

        if not (filename_lower.endswith('.md')
                or filename_lower.endswith('.mdx')):
            continue

        try:
            with zf.open(file_info) as f_in:
                content = f_in.read().decode('utf-8', errors='ignore')
                post = frontmatter.loads(content)
                data = post.to_dict()
                data['filename'] = filename
                repository_data.append(data)
        except Exception as e:
            print(f"Error processing {filename}: {e}")
            continue

    zf.close()
    return repository_data

```

We can now use this function for different repositories:

```
Python
dtc_faq = read_repo_data('DataTalksClub', 'faq')
evidently_docs = read_repo_data('evidentlyai', 'docs')

print(f"FAQ documents: {len(dtc_faq)}")
print(f"Evidently documents: {len(evidently_docs)}")
```

## Data Processing Considerations

For FAQ, the data is ready to use. These are small records that we can index (put into a search engine) as is.

For Evidently docs, the documents are very large. We need extra processing called "chunking" - breaking large documents into smaller, manageable pieces. This is important because:

1. Search relevance: Smaller chunks are more specific and relevant to user queries
2. Performance: AI models work better with shorter text segments
3. Memory limits: Large documents might exceed token limits of language models

We will cover chunking techniques in tomorrow's lesson.

If you have any suggestions about the course content or want to improve something, let me know!

## Homework

- Create a new `uv` project in the `project` directory
- Select a GitHub repo with documentation (preferably with `.md` files)
- Download the data from there using the techniques we've learned
- Make a post on social media about what you're building



# Learning in Public


We encourage everyone to share what they learned. This is called "learning in public".

Learning in public is one of the most effective ways to accelerate your growth. Here's why:

1. Accountability: Sharing your progress creates commitment and motivation to continue
2. Feedback: The community can provide valuable suggestions and corrections
3. Networking: You'll connect with like-minded people and potential collaborators
4. Documentation: Your posts become a learning journal you can reference later
5. Opportunities: Employers and clients often discover talent through public learning




Don't worry about being perfect. Everyone starts somewhere, and people love following genuine learning journeys!

## Example post for LinkedIn

 Day 1 of AI Agents crash course by @Alexey Grigorev complete!

Just built my first data ingestion pipeline that can download and parse any GitHub repository's documentation.

Today I learned how to:

-  Download repos as zip archives
-  Parse frontmatter metadata
-  Extract content from markdown files

Here's my repo:

Next up: Chunking the data for better search performance!

Following along with this amazing course - who else is building AI agents?

You can sign up here: <https://alexeygrigorev.com/aihero/>

## Example post for Twitter/X

 Built my first AI data pipeline in a course from @AI\_Grigor

- ✨ Downloads any GitHub repo
- 📝 Parses markdown
- 🔍 Prepares data for AI search

Here's my repo:

Tomorrow: Chunking the data for search performance!

Join me too: <https://alexeygrigorev.com/aihero/>

## Community

Have questions about this lesson or suggestions for improvement? You can find me and other learners in **DataTalks.Club Slack**:

- [Join DataTalks.Club](#)
- Find us in the [#course-ai-bootcamp channel](#)

In the community channel you can:

- Ask questions about the course content
- Share your implementation and get feedback
- Show off your GitHub repositories
- Suggest improvements to the course materials
- Connect with other course participants

Don't hesitate to reach out - the community is here to help each other succeed!