Week 1: Data Collection for ML

CS 203: Software Tools and Techniques for AI

Prof. Nipun Batra, IIT Gandhinagar

Your Mission 🎬

You work at Netflix.

Your boss asks:

> "Which movies should we add to our catalog next month?"

This is an ML problem.

But first... we need **data**.

The ML Pipeline

```
DATA → MODEL → PREDICTIONS
  ↑
  We are here
```

Today's question: Where does data come from?

What Data Do We Need?

To predict movie success, we need features:

- 🎭 **Title, Genre, Year**
- ⭐ **IMDb Rating**
- 🍅 **Rotten Tomatoes Score**
- 💰 **Box Office Performance**
- 👥 **Cast & Director**
- 📊 **Social Media Buzz**

Problem: This data lives on websites!

Today's Agenda

1. **Understanding the Web** (HTTP, URLs, requests)
2. **Tool #1: curl** (command-line HTTP)
3. **Tool #2: Chrome DevTools** (inspect websites)
4. **Tool #3: Python requests** (automate data fetching)
5. **Tool #4: BeautifulSoup** (parse HTML)
6. **Tool #5: Playwright** (dynamic websites)
7. **APIs: The Better Way** (structured data)

Part 1: How the Web Works

Client-Server Model

```
Your Browser          The Internet          IMDb Server
(Client)                                    (Server)
   |                                           |
   |-------- "Give me Shawshank" ------------->|
   |                                           |
   |<------- HTML page with data --------------|
```

HTTP = HyperText Transfer Protocol

The language browsers and servers speak.

A Simple HTTP Request

URL: `https://www.imdb.com/title/tt0111161/`

Breaking it down:

- `https://` → Protocol (secure HTTP)
- `www.imdb.com` → Domain (server)
- `/title/tt0111161/` → Path (resource)

HTTP Methods

GET → Retrieve data

```
GET /title/tt0111161/
```

POST → Send data

```
POST /api/login
Body: {"username": "user", "password": "pass"}
```

Others: PUT, DELETE, PATCH

Today we focus on GET

HTTP Response

Server sends back:

Status Code:

- `200` ✅ Success
- `404` ❌ Not found
- `500` 💥 Server error
- `429` ⏸️ Rate limited

Headers: Metadata (content type, length)

Body: The actual data (HTML, JSON, etc.)

Part 2: Tool #1 - curl

What is curl?

Command-line tool to make HTTP requests.

Pre-installed on Mac/Linux.

Windows: Use Git Bash or WSL.

curl Basics

Get IMDb homepage:

```
curl https://www.imdb.com/
```

Output: Raw HTML (lots of it!)

curl with Options

Save to file:

```
curl https://www.imdb.com/ -o imdb.html
```

See response headers:

```
curl -I https://www.imdb.com/
```

Follow redirects:

```
curl -L https://imdb.com/
```

curl for APIs

Get movie data (JSON):

```
curl "http://www.omdbapi.com/?apikey=YOUR_KEY&t=Shawshank"
```

Pretty print JSON:

```
curl "http://www.omdbapi.com/?apikey=YOUR_KEY&t=Shawshank" | jq
```

jq = JSON processor

Why Learn curl?

✅ Quick testing of APIs

✅ No code needed

✅ Works everywhere (servers, containers)

✅ Great for debugging

Part 3: Tool #2 - Chrome DevTools

Open DevTools

1. Visit: `https://www.imdb.com/title/tt0111161/`
2. Right-click → "Inspect"
3. Go to **Network** tab
4. Refresh page (Cmd+R / Ctrl+R)

What You See

Network Tab shows:

- All HTTP requests the page makes
- Status codes
- Response times
- Headers
- Response data

Demo time! 👀

Inspecting Elements

Elements Tab:

- Shows HTML structure
- Click element → highlights on page
- Right-click element on page → Inspect

Finding data:

- Look for rating: `<span class="rating">9.3</span>`
- Note the class name: `rating`
- We'll use this later!

DevTools for APIs

Many websites load data via **background API calls**.

Network tab → **Fetch/XHR** filter

Shows JSON responses!

Example: Twitter, Reddit, YouTube

Part 4: Tool #3 - Python requests

Why Python?

curl is great for testing.

But for **automation**, we need code.

Python `requests` library = curl for Python

Install requests

```
pip install requests
```

That's it! ✅

Basic GET Request

```python
import requests

response = requests.get('https://www.imdb.com/title/tt0111161/')

print(response.status_code)  # 200
print(response.text[:100])    # First 100 chars of HTML
```

Response Object

```
response = requests.get('https://www.imdb.com/')

# Status
response.status_code  # 200

# Headers
response.headers['content-type']  # 'text/html'

# Body
response.text  # HTML as string
response.content  # HTML as bytes
```

Query Parameters

Instead of:

```
url = 'https://api.example.com/search?q=movie&year=2024'
```

Better:

```
response = requests.get('https://api.example.com/search',
    params={
        'q': 'movie',
        'year': 2024
    }
)
```

Headers

Send custom headers:

```
headers = {
    'User-Agent': 'Mozilla/5.0 (Netflix Bot)',
    'Accept': 'application/json'
}

response = requests.get(url, headers=headers)
```

Why? Some sites block Python's default User-Agent.

Error Handling

```python
response = requests.get(url)

if response.status_code == 200:
    print("Success!")
    data = response.text
elif response.status_code == 404:
    print("Page not found")
else:
    print(f"Error: {response.status_code}")
```

Better Error Handling

```python
try:
    response = requests.get(url, timeout=10)
    response.raise_for_status()  # Raises exception for 4xx/5xx
    data = response.text
except requests.exceptions.Timeout:
    print("Request timed out")
except requests.exceptions.HTTPError as e:
    print(f"HTTP error: {e}")
except requests.exceptions.RequestException as e:
    print(f"Error: {e}")
```

Part 5: Tool #4 - BeautifulSoup

The Problem

We got the HTML:

```html
<html>
  <body>
    <h1 class="title">The Shawshank Redemption</h1>
    <span class="rating">9.3</span>
    <div class="genre">Drama</div>
  </body>
</html>
```

How do we **extract** the rating?

BeautifulSoup

HTML parser for Python.

Converts messy HTML → searchable structure.

```
pip install beautifulsoup4
```

Basic Usage

```python
from bs4 import BeautifulSoup
import requests

# Get the page
response = requests.get('https://www.imdb.com/title/tt0111161/')

# Parse HTML
soup = BeautifulSoup(response.text, 'html.parser')

# Now we can search!
```

Finding Elements

By tag:

```
title = soup.find('h1')
print(title.text)  # "The Shawshank Redemption"
```

By class:

```
rating = soup.find('span', class_='rating')
print(rating.text)  # "9.3"
```

By ID:

```
element = soup.find(id='main-content')
```

Find vs Find_all

find() → First match

```
first_div = soup.find('div')
```

find_all() → All matches (returns list)

```
all_divs = soup.find_all('div')
print(len(all_divs))  # e.g., 42

for div in all_divs:
    print(div.text)
```

CSS Selectors

More powerful searching:

```
# Find all links in a specific div
links = soup.select('div.cast-list a')

# Complex selector
rating = soup.select_one('div.rating-container span.value')
```

Tip: Copy selector from Chrome DevTools!

Extracting Attributes

```python
# Get link URL
link = soup.find('a', class_='movie-link')
url = link['href']
print(url)  # "/title/tt0111161/"

# Get image source
img = soup.find('img', class_='poster')
poster_url = img['src']
```

Netflix Example: Complete Scraper

```python
import requests
from bs4 import BeautifulSoup

def get_movie_info(imdb_id):
    url = f'https://www.imdb.com/title/{imdb_id}/'
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')

    title = soup.find('h1').text.strip()
    rating = soup.find('span', class_='rating').text
    genre = soup.find('div', class_='genre').text.strip()

    return {
        'title': title,
        'rating': float(rating),
        'genre': genre
    }

movie = get_movie_info('tt0111161')
print(movie)
```

Web Scraping Challenges

❌ HTML structure changes → code breaks

❌ Slow (one request at a time)

❌ Anti-scraping measures (CAPTCHAs, blocks)

❌ Legal/ethical concerns

❌ Dynamic content (JavaScript-loaded)

Part 6: Tool #5 - Playwright

The JavaScript Problem

Modern websites load data with JavaScript.

```html
<!-- Initial HTML (empty!) -->
<div id="movies"></div>

<script>
  // JavaScript loads movies after page loads
  fetch('/api/movies').then(data => {
    document.getElementById('movies').innerHTML = data;
  });
</script>
```

requests + BeautifulSoup only sees empty `<div>`!

Solution: Headless Browser

Playwright = Control a real browser from Python

- Executes JavaScript
- Waits for content to load
- Can click, scroll, type

Install Playwright

```
pip install playwright
playwright install chromium
```

Downloads a Chromium browser.

Basic Playwright Usage

```python
from playwright.sync_api import sync_playwright

with sync_playwright() as p:
    # Launch browser
    browser = p.chromium.launch(headless=True)
    page = browser.new_page()

    # Navigate to page
    page.goto('https://www.imdb.com/title/tt0111161/')

    # Wait for content
    page.wait_for_selector('.rating')

    # Get HTML (after JavaScript runs)
    html = page.content()

    browser.close()
```

Playwright Features

Click elements:

```
page.click('button.load-more')
```

Fill forms:

```
page.fill('input[name="search"]', 'Shawshank')
page.press('input[name="search"]', 'Enter')
```

Screenshot:

```
page.screenshot(path='screenshot.png')
```

When to Use Playwright?

✅ JavaScript-heavy sites (React, Angular, Vue)

✅ Need to interact (click, scroll, login)

✅ Content loads on scroll (infinite scroll)

❌ Slow and heavy (uses real browser)

❌ Overkill for static sites

Rule: Try requests first. Use Playwright if needed.

Netflix Example: Dynamic Scraping

```python
from playwright.sync_api import sync_playwright
from bs4 import BeautifulSoup

def scrape_dynamic_site(url):
    with sync_playwright() as p:
        browser = p.chromium.launch()
        page = browser.new_page()
        page.goto(url)

        # Wait for JavaScript to load content
        page.wait_for_selector('.movie-list')

        # Get the fully-loaded HTML
        html = page.content()
        browser.close()

        # Parse with BeautifulSoup
        soup = BeautifulSoup(html, 'html.parser')
        movies = soup.find_all('div', class_='movie-card')
        return [m.text for m in movies]
```

Part 7: APIs - The Better Way

The Problem with Scraping

We've learned scraping but...

- 😞 HTML changes break code
- 🐌 Slow
- 🚫 Sites may block you
- ⚖️ Legal gray area

Better solution: Use an API!

What is an API?

API = Application Programming Interface

Web API = URL that returns structured data (usually JSON)

Instead of:

```
HTML (for humans) → scrape → extract data
```

API gives:

```
JSON (for machines) → directly usable data
```

HTML vs JSON

HTML (for browsers):

```html
<div class="movie">
  <h1>Shawshank Redemption</h1>
  <span class="rating">9.3</span>
</div>
```

JSON (for code):

```json
{
  "title": "Shawshank Redemption",
  "rating": 9.3
}
```

Much cleaner! ✨

REST APIs

REST = Representational State Transfer

Common pattern for web APIs.

Example endpoints:

```
GET  /movies              → List all movies
GET  /movies/123          → Get movie #123
POST /movies              → Create new movie
PUT  /movies/123          → Update movie #123
DELETE /movies/123        → Delete movie #123
```

OMDb API

The Open Movie Database

Free API for movie data!

Website: http://www.omdbapi.com/

Get API key: Sign up (free)

Using OMDb API

```python
import requests

API_KEY = 'your_key_here'

response = requests.get('http://www.omdbapi.com/', params={
    'apikey': API_KEY,
    't': 'Shawshank Redemption'  # Search by title
})

data = response.json()  # Parse JSON → Python dict
print(data['Title'])    # "The Shawshank Redemption"
print(data['imdbRating'])  # "9.3"
print(data['Genre'])    # "Drama"
```

JSON Response

```json
{
  "Title": "The Shawshank Redemption",
  "Year": "1994",
  "Rated": "R",
  "Genre": "Drama",
  "Director": "Frank Darabont",
  "Actors": "Tim Robbins, Morgan Freeman",
  "imdbRating": "9.3",
  "imdbID": "tt0111161",
  "BoxOffice": "$28,767,189"
}
```

Clean, structured, perfect for ML! ✅

Search by IMDb ID

```python
response = requests.get('http://www.omdbapi.com/', params={
    'apikey': API_KEY,
    'i': 'tt0111161'  # IMDb ID
})

data = response.json()
```

More reliable than searching by title.

API Authentication

Most APIs require **authentication**.

Common methods:

1. **API Key** (in URL or header)

   ```
   params={'apikey': 'abc123'}
   ```

2. **Bearer Token** (in header)

   ```
   headers={'Authorization': 'Bearer xyz789'}
   ```

3. **OAuth** (complex, for user data)

Rate Limiting

APIs limit requests to prevent abuse.

Example: 1000 requests/day

HTTP 429: Too Many Requests

Solutions:

- Respect limits
- Cache responses
- Pay for higher tier

Handling Rate Limits

```python
import time

def get_movie_safe(movie_id):
    response = requests.get(url, params={...})

    if response.status_code == 429:
        print("Rate limited! Waiting...")
        time.sleep(60)  # Wait 1 minute
        return get_movie_safe(movie_id)  # Retry

    return response.json()
```

Better: Use backoff library

Pagination

APIs often return data in **pages**.

```python
page = 1
all_movies = []

while True:
    response = requests.get(url, params={
        'page': page,
        'limit': 100
    })

    movies = response.json()['results']
    if not movies:
        break  # No more data

    all_movies.extend(movies)
    page += 1
```

Netflix Complete Example

```python
import requests

API_KEY = 'your_omdb_key'

def fetch_movie_data(titles):
    """Fetch data for multiple movies."""
    movies = []

    for title in titles:
        response = requests.get('http://www.omdbapi.com/',
            params={'apikey': API_KEY, 't': title}
        )

        if response.status_code == 200:
            data = response.json()
            if data['Response'] == 'True':
                movies.append({
                    'title': data['Title'],
                    'year': data['Year'],
                    'rating': data['imdbRating'],
                    'genre': data['Genre'],
                    'box_office': data.get('BoxOffice', 'N/A')
                })

    return movies
```

Other Useful APIs

Movies:

- TMDb (The Movie Database)
- Rotten Tomatoes

Weather:

- OpenWeatherMap
- Weather.gov

News:

- NewsAPI
- Guardian API

Finance:

API Best Practices

✅ **Read the docs** (rate limits, auth)

✅ **Handle errors** (network, rate limits, invalid data)

✅ **Cache responses** (don't re-fetch same data)

✅ **Respect rate limits** (be a good citizen)

✅ **Keep API keys secret** (use environment variables)

Storing API Keys Safely

❌ **DON'T:**

```
API_KEY = 'abc123xyz'  # Hardcoded!
```

✅ **DO:**

```
import os
API_KEY = os.environ['OMDB_API_KEY']
```

Set in terminal:

```
export OMDB_API_KEY='abc123xyz'
```

Or use `.env` file + `python-dotenv`

Summary: Tools We Learned

Tool	Purpose	When to Use
curl	Test HTTP from terminal	Quick API testing
Chrome DevTools	Inspect web traffic	Find data sources
requests	HTTP in Python	Static sites, APIs
BeautifulSoup	Parse HTML	Web scraping
Playwright	Control browser	JavaScript sites

Web Scraping vs APIs

Aspect	Web Scraping	APIs
Speed	Slow	Fast ⚡
Reliability	Fragile 🐛	Stable ✅
Data Format	HTML (messy)	JSON (clean)
Legal	Gray area ⚖️	Approved ✅
When to use	No API available	Always prefer!

Golden Rule: Use API if available. Scrape as last resort.

The Data Formats

HTML: For humans (browsers)

```
<div class="price">$29.99</div>
```

JSON: For machines (APIs)

```
{"price": 29.99, "currency": "USD"}
```

CSV: Tabular data

```
title,rating,year
Shawshank,9.3,1994
```

Netflix Project Status

✅ We can now collect movie data!

Next steps:

1. **Week 2**: Validate the data (Pydantic)
2. **Week 3**: Enrich with LLM features
3. **Week 7**: Build prediction model
4. **Week 9**: Deploy as interactive demo

Lab Preview

Your task: Build a movie dataset collector

1. Use OMDb API to fetch 100 movies
2. Parse and structure the data
3. Save to CSV
4. Handle errors gracefully

Bonus: Compare with web scraping approach

Key Takeaways

🎯 **Data collection is the first step in ML**

🛠️ **Master the tools**: curl, DevTools, requests, BeautifulSoup, Playwright

📊 **APIs > Scraping** (when available)

⚡ **Handle errors** (network, rate limits, bad data)

🔐 **Keep credentials safe** (environment variables)

Resources

Documentation:

- requests: https://requests.readthedocs.io/
- BeautifulSoup: https://www.crummy.com/software/BeautifulSoup/
- Playwright: https://playwright.dev/python/

Practice APIs:

- OMDb: http://www.omdbapi.com/
- JSONPlaceholder: https://jsonplaceholder.typicode.com/
- PokéAPI: https://pokeapi.co/

Questions?

Next Week: **Data Validation with Pydantic**

Making sure our Netflix data is clean and usable!