□□□□□□□□□ □□□□□□□□□□□□
□□□□□□□□□□□□ □□□□□□□□□□□□□□□□
□□□□□□

Contents

바이브 코딩으로 시작하는 생명정보학 실습

저자: 박정빈

이 책에 대하여

이 책은 AI 코딩 에이전트를 활용하여 생명정보학 분석과 웹 도구 개발을 실습하는 안내서이다. 코드를 한 줄 한 줄 외우는 대신, 개발과 분석의 핵심 개념을 이해하고 AI에게 정확히 지시하는 **바이브 코딩(Vibe Coding)** 방식을 다룬다.

개발 환경 구성부터 Python 데이터 분석, 단일세포 RNA-seq 분석, 워크플로우 자동화, 그리고 SvelteKit 기반 생명정보학 웹 도구 제작까지 실무에 필요한 전 과정을 단계별로 안내한다.

1장. 개발 환경 구성

1.1 Visual Studio Code 설치

Visual Studio Code(VS Code)는 Microsoft에서 개발한 무료 코드 편집기로, 다양한 프로그래밍 언어를 지원하며 확장 프로그램을 통해 기능을 추가할 수 있다. 이 책에서는 VS Code를 기본 개발 환경으로 사용한다.

설치 방법

1. https://code.visualstudio.com 에 접속한다
2. 운영체제에 맞는 설치 파일을 다운로드한다 (Windows, macOS, Linux)
3. 다운로드한 파일을 실행하여 설치를 완료한다

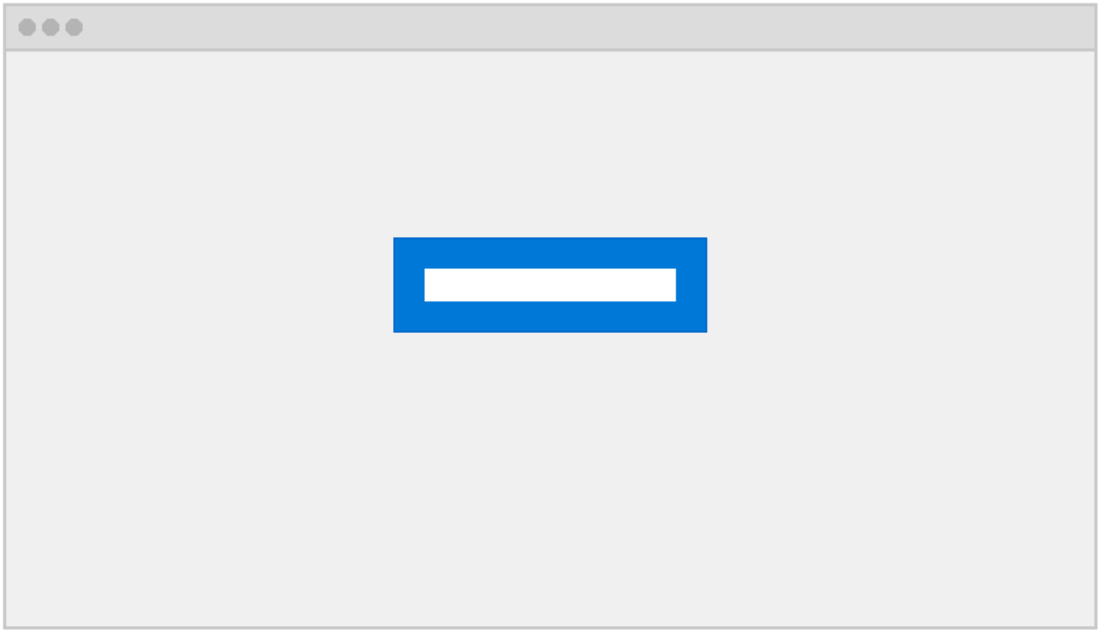

Figure 1: VS Code 공식 웹사이트에서 다운로드 버튼이 보이는 메인 페이지 스크린샷

설치 확인

설치가 완료되면 VS Code를 실행하여 정상적으로 동작하는지 확인한다.

유용한 확장 프로그램

VS Code의 강력한 장점 중 하나는 확장 프로그램(Extension)이다. 왼쪽 사이드바의 확장 프로그램 아이콘을 클릭하거나 `Cmd+Shift+X` (Mac) / `Ctrl+Shift+X` (Windows/Linux)를 눌러 확장 프로그램 마켓플레이스를 열 수 있다. 앞으로 필요한 확장 프로그램은 이곳에서 검색하여 설치하면 된다.

1.2 WSL (Windows Subsystem for Linux)

Windows 사용자의 경우, 웹 개발 및 생명정보학 도구 대부분이 리눅스 환경을 기반으로 하므로 **WSL(Windows Subsystem for Linux)**을 설치하여 리눅스 환경에서 개발을 진행하는 것을 권장한다. macOS나 Linux 사용자는 이 절을

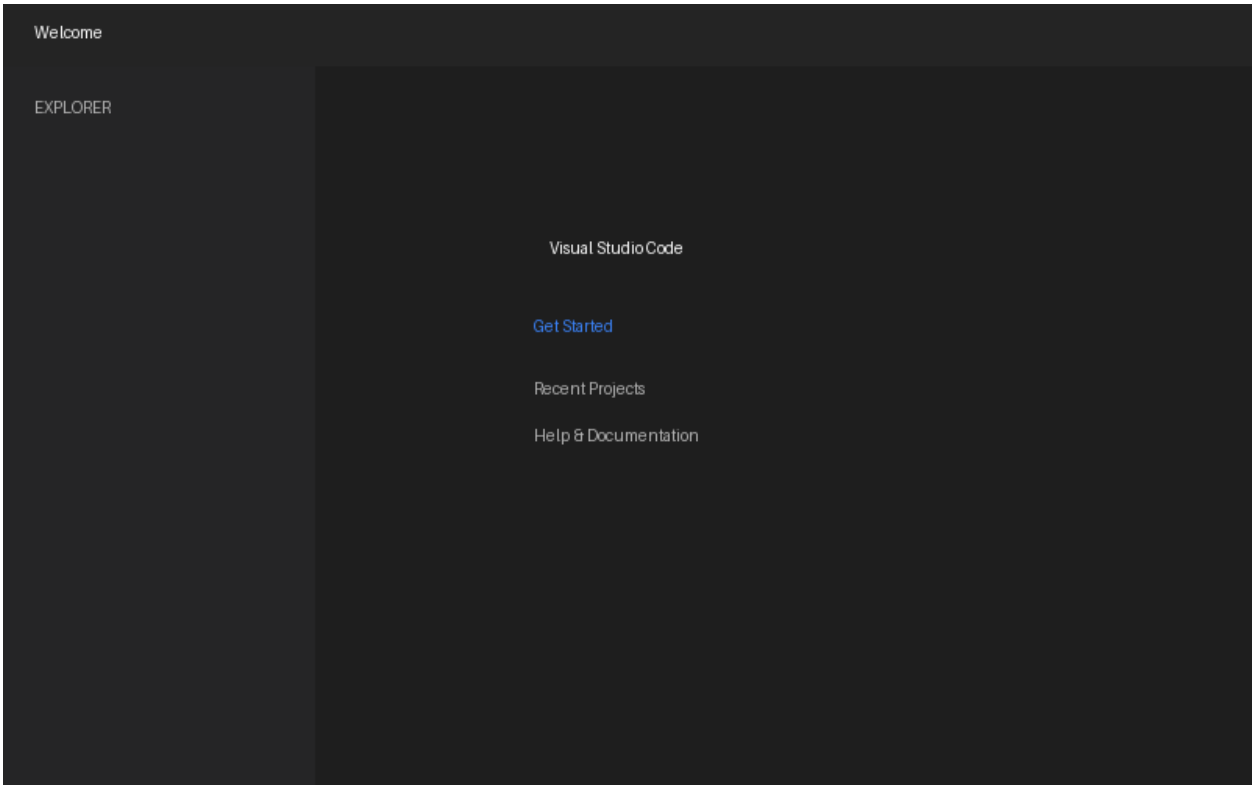

Figure 2: VS Code를 처음 실행했을 때의 Welcome 탭 화면 스크린샷

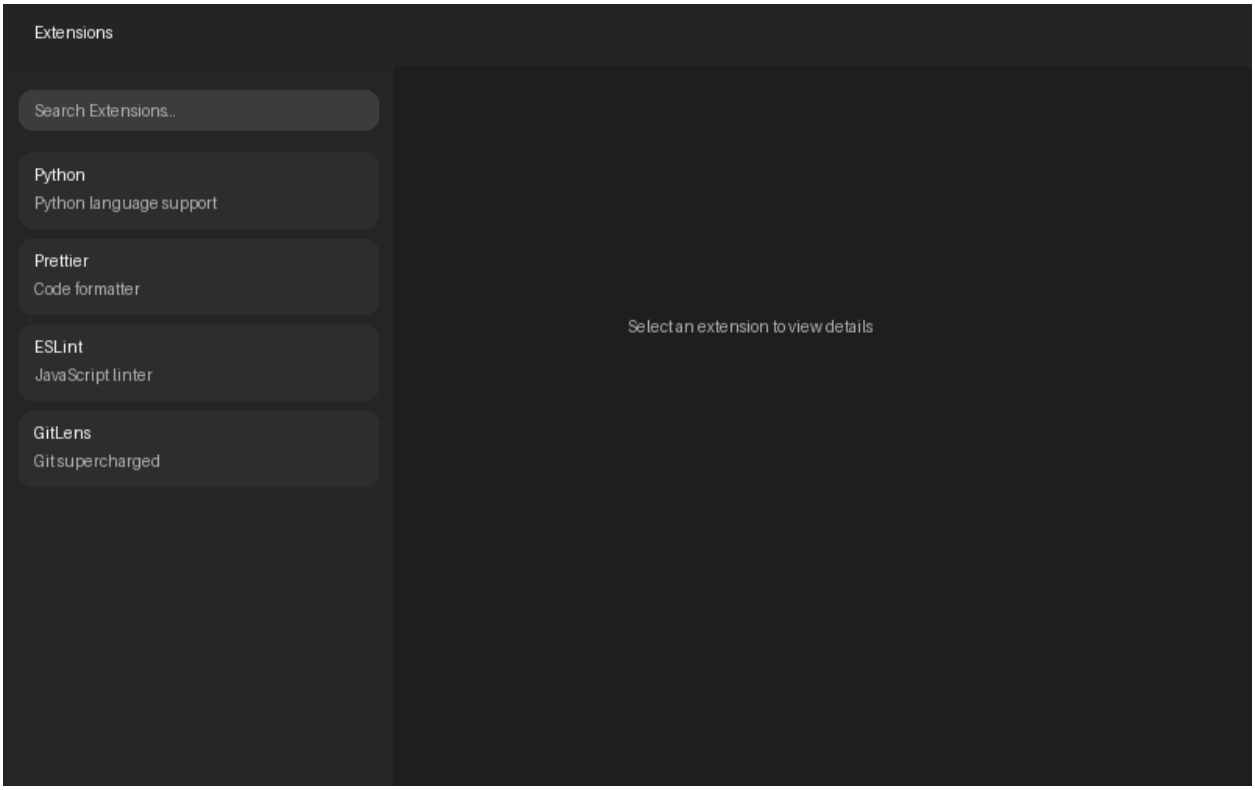

Figure 3: VS Code 확장 프로그램 마켓플레이스 화면 스크린샷

건너뛰어도 된다.

WSL이란?

WSL은 Windows 위에서 리눅스 배포판을 직접 실행할 수 있게 해주는 기능이다. 별도의 가상 머신 없이도 리눅스 터미널과 명령어를 사용할 수 있으며, Windows의 파일 시스템과도 자연스럽게 연동된다.

WSL 설치

PowerShell을 **관리자 권한**으로 실행한 뒤 다음 명령을 입력한다:

```
wsl --install
```

이 명령 하나로 WSL 기능 활성화와 Ubuntu 배포판 설치가 자동으로 진행된다. 설치가 완료되면 컴퓨터를 재부팅한다.

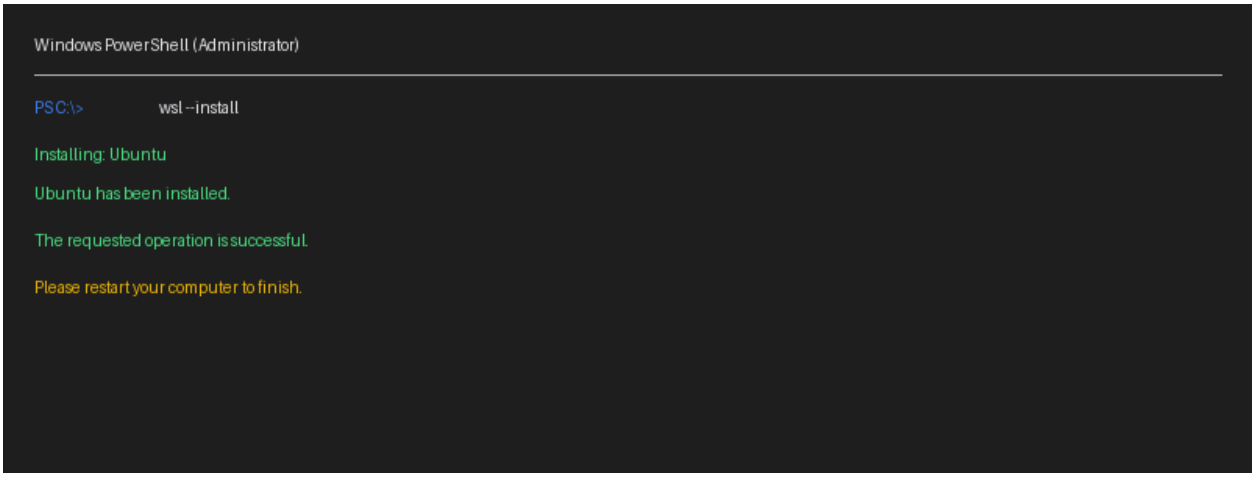

Figure 4: PowerShell 관리자 권한으로 wsl –install 실행하는 화면 스크린샷

재부팅 후 자동으로 Ubuntu 터미널이 열리며, 리눅스 사용자 이름과 비밀번호를 설정하라는 메시지가 나타난다.

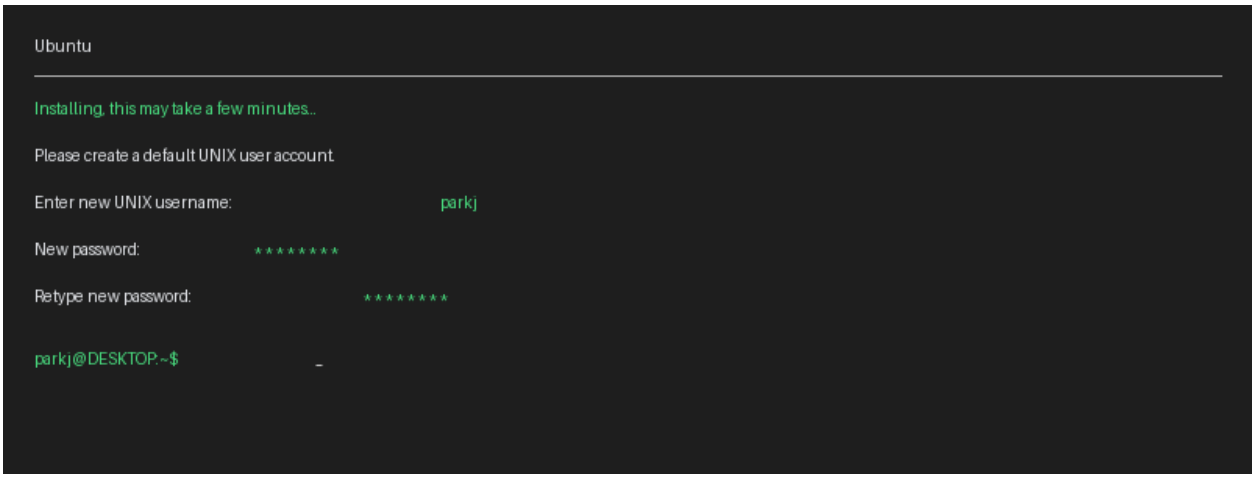

Figure 5: WSL Ubuntu 초기 설정 — 사용자 이름과 비밀번호 입력 화면 스크린샷

VS Code에서 WSL 연동

VS Code에서 WSL 환경을 사용하려면 **WSL** 확장 프로그램을 설치한다.

1. VS Code 확장 프로그램 마켓플레이스에서 “WSL”을 검색하여 설치
2. VS Code 좌측 하단의 파란색 >< 아이콘을 클릭
3. **“Connect to WSL”**을 선택

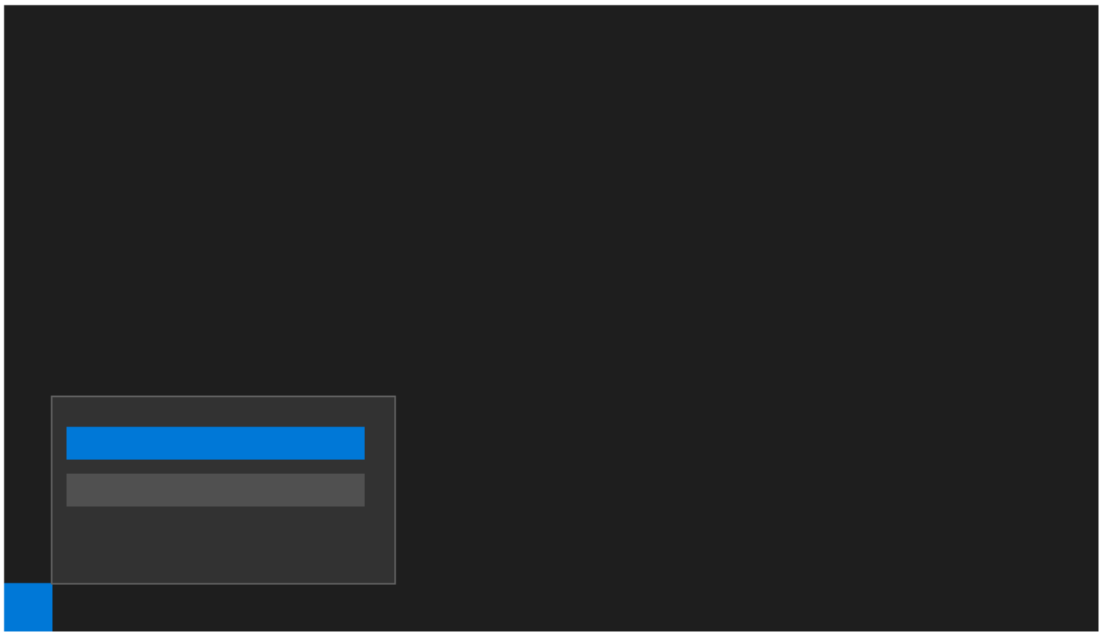

Figure 6: VS Code에서 WSL 확장 프로그램 설치 및 WSL 연결 화면 스크린샷

연결이 완료되면 VS Code 좌측 하단에 **“WSL: Ubuntu”**라고 표시되며, 터미널을 열면 리눅스 셸이 실행된다. 이후 이 책의 모든 터미널 명령은 WSL 환경에서 실행하면 된다.

1.3 Python 환경 설정: uv와 micromamba

WSL 환경이 준비되었으면, 생명정보학 작업에 필요한 Python 패키지 관리 도구를 설치한다. 이 책에서는 **uv**(Python 패키지 매니저)와 **micromamba**(Conda 호환 환경 매니저)를 사용한다.

uv 설치

uv는 Rust로 작성된 초고속 Python 패키지 매니저이다. 기존의 pip보다 10~100배 빠르며, 가상 환경 생성과 패키지 설치를 한 번에 처리할 수 있다.

```
curl -LsSf https://astral.sh/uv/install.sh | sh
```

설치 후 터미널을 재시작하거나 다음 명령으로 환경을 갱신한다:

```
source ~/.bashrc
```

기본 사용법:

```
#       +
uv venv
uv pip install pandas numpy matplotlib
```

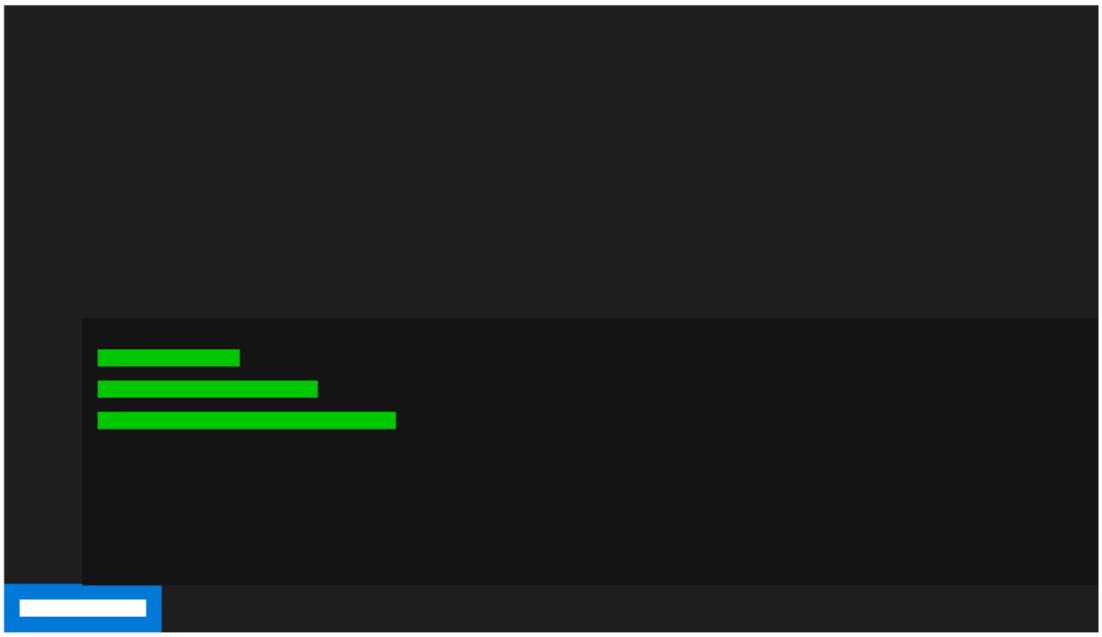

Figure 7: VS Code가 WSL에 연결된 상태 — 좌측 하단에 “WSL: Ubuntu” 표시와 리눅스 터미널이 열린 모습

```
#   requirements.txt
uv pip install -r requirements.txt

# Python        (       )
uv run script.py
```

micromamba 설치

micromamba는 Conda의 경량화 버전으로, Conda와 동일한 패키지 저장소(conda-forge, bioconda)를 사용하지만 훨씬 빠르고 가볍다. 생명정보학 도구 중 상당수가 Conda/Bioconda 채널을 통해 배포되므로 micromamba가 필요하다.

```
"${SHELL}" <(curl -L micro.mamba.pm/install.sh)
```

설치 과정에서 나오는 질문에는 기본값(Enter)으로 응답하면 된다. 설치 완료 후 터미널을 재시작한다.

micromamba를 conda로 사용하기

생명정보학의 많은 문서와 튜토리얼은 `conda` 명령을 기준으로 작성되어 있다. AI 에이전트도 `conda` 명령을 생성하는 경우가 많다. **alias를 설정하면 `conda` 명령을 입력해도 micromamba가 실행되도록** 할 수 있다.

다음 명령을 실행하면 `~/.bashrc`에 alias가 추가된다:

```
echo 'alias conda="micromamba"' >> ~/.bashrc
source ~/.bashrc
```

이제 `conda`와 `micromamba`를 동일하게 사용할 수 있다:

```
#
conda create -n bioinfo python=3.11
```

```
micromamba create -n bioinfo python=3.11

#
conda activate bioinfo
#
micromamba activate bioinfo

# Bioconda
conda install -c bioconda -c conda-forge samtools
```

uv vs micromamba: 언제 무엇을 쓸까?

	uv	micromamba (conda)
용도	Python 패키지 설치	Python + 비Python 도구 설치
속도	매우 빠름	빠름 (conda보다 훨씬 빠름)
사용 상황	pandas, matplotlib 등 순수 Python 패키지	samtools, STAR, snakemake 등 바이너리 도구
채널	PyPI	conda-forge, bioconda

> **팁**: 일반적인 Python 패키지는 uv로 설치하고, 생명정보학 전용 도구(samtools, STAR, bedtools 등)는 micromamba로 설치하는 것이 좋다.

1.4 VS Code에서 Claude Code 사용하기

Claude Code는 VS Code에 직접 통합되는 AI 코딩 도구이다. Github Copilot과 마찬가지로 VS Code 확장 프로그램으로 설치하여 사용할 수 있으며, 코드 작성, 디버깅, 리팩토링 등 다양한 작업을 AI의 도움을 받아 수행할 수 있다.

필수 조건

- VS Code 1.98.0 이상
- Anthropic 계정 (확장 프로그램을 처음 열 때 로그인)

설치 방법

VS Code에서 `Cmd+Shift+X` (Mac) 또는 `Ctrl+Shift+X` (Windows/Linux)를 눌러 확장 프로그램 보기를 열고, "Claude Code"를 검색한 후 **설치**를 클릭한다.

시작하기

1단계: Claude Code 패널 열기 편집기 오른쪽 위 모서리의 **Spark 아이콘**을 클릭하여 Claude Code 패널을 연다. 또는 다음 방법으로 열 수 있다:

- **명령 팔레트**: `Cmd+Shift+P` (Mac) 또는 `Ctrl+Shift+P` (Windows/Linux)를 누르고 "Claude Code"를 입력
- **상태 표시줄**: 창 오른쪽 아래의 **Claude Code**를 클릭

2단계: 프롬프트 보내기 Claude에게 코드에 대해 질문하거나, 디버깅을 요청하거나, 변경 사항 작성을 요청한다. 편집기에서 텍스트를 선택하면 Claude가 해당 코드를 자동으로 인식한다.

- `@` 뒤에 파일명을 입력하면 특정 파일을 참조할 수 있다 (예: `@auth.js`)
- `Option+K` (Mac) / `Alt+K` (Windows/Linux)로 현재 파일과 선택 영역의 참조를 삽입할 수 있다

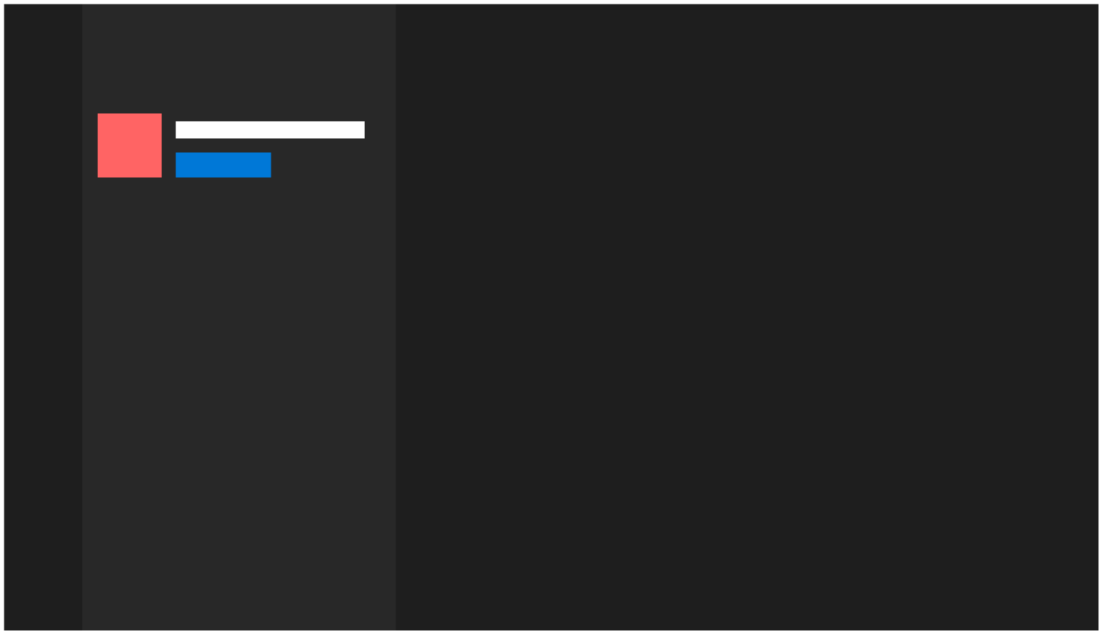

Figure 8: VS Code 확장 프로그램 마켓플레이스에서 Claude Code를 검색하여 설치하는 화면 스크린샷

Figure 9: VS Code 편집기에서 Spark 아이콘 위치와 Claude Code 패널이 열린 모습 스크린샷

Figure 10: Claude Code 패널에 프롬프트를 입력하고 @-멘션으로 파일을 참조하는 모습 스크린샷

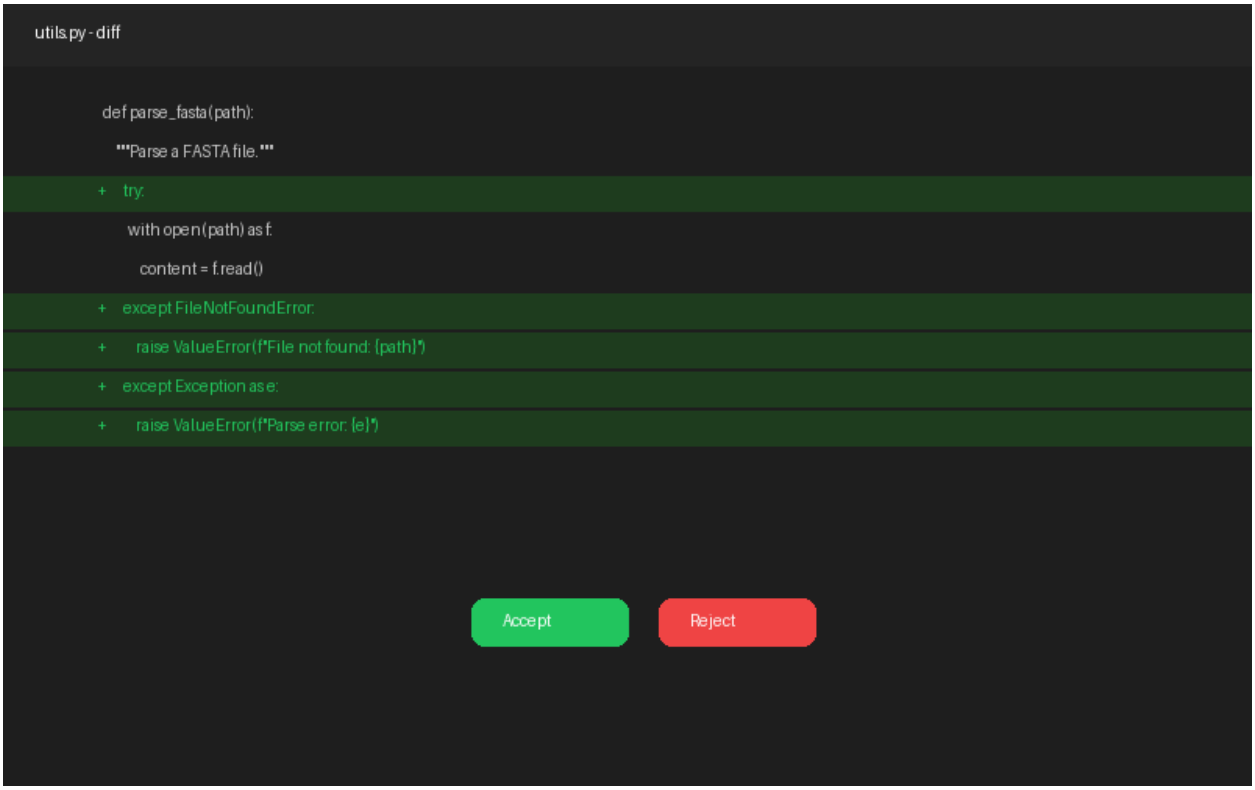

Figure 11: Claude가 제안한 코드 변경 사항의 diff 화면과 수락/거부 버튼이 있는 스크린샷

3단계: 변경 사항 검토 Claude가 파일을 편집하려고 하면, 원본과 제안된 변경 사항을 나란히 비교하는 diff 화면이 나타난다. 수락하거나 거부하거나, Claude에게 다른 방법으로 수정하도록 요청할 수 있다.

주요 기능

권한 모드 프롬프트 상자 하단의 모드 표시기를 클릭하여 전환한다:

모드	설명
일반 모드	각 작업 전에 권한을 요청
Plan Mode	수행할 작업을 설명하고 승인을 기다림
자동 수락 모드	요청하지 않고 바로 편집

파일 및 폴더 참조 `@-` 을 사용하여 특정 파일이나 폴더에 대한 컨텍스트를 Claude에게 제공한다:

```
> @auth.js
> @src/components/
```

여러 대화 실행 명령 팔레트에서 **새 탭에서 열기** 또는 **새 창에서 열기**를 사용하여 여러 대화를 동시에 실행할 수 있다. 각 대화는 독립적인 기록과 컨텍스트를 유지한다.

Git 통합 Claude Code는 Git과 통합되어 VS Code에서 직접 버전 관리 작업을 수행할 수 있다:

```
>
>        PR
```

주요 단축키

명령	단축키 (Mac / Windows·Linux)	설명
포커스 전환	`Cmd+Esc` / `Ctrl+Esc`	편집기와 Claude 사이 전환
새 탭에서 열기	`Cmd+Shift+Esc` / `Ctrl+Shift+Esc`	새 대화를 탭으로 열기
새 대화	`Cmd+N` / `Ctrl+N`	새 대화 시작 (Claude 포커스 시)
@-멘션 삽입	`Option+K` / `Alt+K`	현재 파일 및 선택 영역 참조 삽입

1.5 정리

- **VS Code 설치 및 기본 사용법 숙지**
  - 확장 프로그램 마켓플레이스 활용법 익히기
- **Windows 사용자는 WSL 설치**
  - VS Code에서 WSL 연동하여 리눅스 환경에서 개발
- **uv와 micromamba로 Python 환경 구성**
  - uv: 빠른 Python 패키지 설치
  - micromamba: Conda 호환 환경 매니저 (bioconda 채널 활용)
  - `alias conda="micromamba"`로 편의성 확보
- **Claude Code 활용하기**
  - VS Code에 Claude Code 확장 프로그램을 설치하고 코딩에 활용해보기
  - @-멘션, 권한 모드 등 주요 기능 익히기

2장. Git & Github

2.1 버전 관리 시스템 (Version Control System)

버전 관리가 없다면?

버전 관리 시스템 없이 파일을 관리하면 어떻게 될까? 아마 다음과 같은 경험이 있을 것이다.

Figure 12: 버전 관리 없이 파일을 관리하는 모습

졸업논문, 졸업논문수정1, 졸업논문수정2, 졸업논문완성본, 졸업논문완성본final, 졸업논문완성본final1, 졸업논문최종완성본-final, 졸업논문최종완성본final1… 파일 이름만으로는 어떤 것이 최신 버전인지, 어떤 변경이 있었는지 알 수 없다. 이러한 문제를 해결하기 위해 **버전 관리 시스템(VCS, Version Control System)**이 등장했다.

버전 관리 시스템의 종류

버전 관리 시스템은 크게 두 가지로 나뉜다.

- **중앙집중형**: SVN, CVS 등
- **분산형**: Git

중앙집중형 VCS 중앙집중형 VCS에서는 하나의 중앙 서버에 버전 트리가 저장된다. 사용자들은 중앙 서버에 접속하여 파일을 가져오고, 수정한 내용을 다시 서버에 올린다.

중앙집중형의 가장 큰 단점은 **항상 온라인이어야 버전 트리에 접근 가능**하다는 점이다. 네트워크 연결이 끊기면 버전 관리 작업을 할 수 없다.

분산형 VCS 분산형 VCS에서는 각 사용자가 버전 트리의 전체 사본을 로컬에 가지고 있다. 따라서 **오프라인에서도 버전 트리 수정이 가능**하다.

2.2 Git이란?

Git은 **분산형 버전 컨트롤 시스템**이다. 프로젝트 폴더 안의 `.git` 디렉토리에 버전 트리가 저장된다.

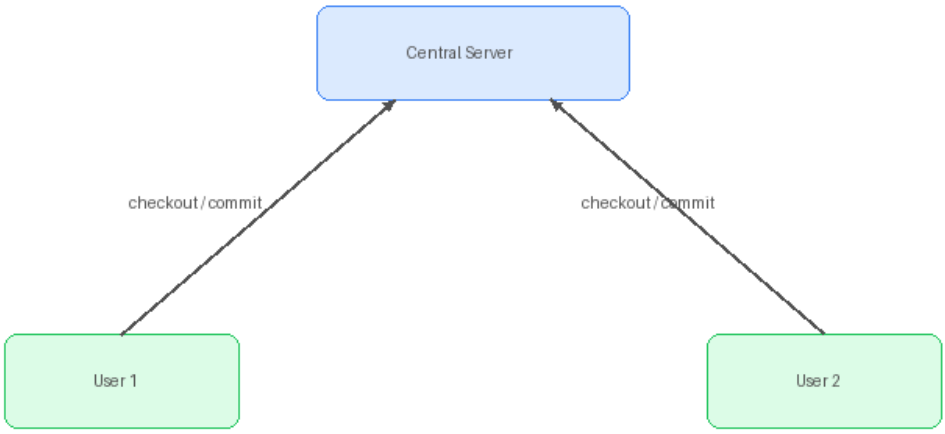

Figure 13: 중앙집중형 VCS 다이어그램 — 중앙 서버에 버전 트리가 있고, User1과 User2가 서버에 연결되어 있는 구조

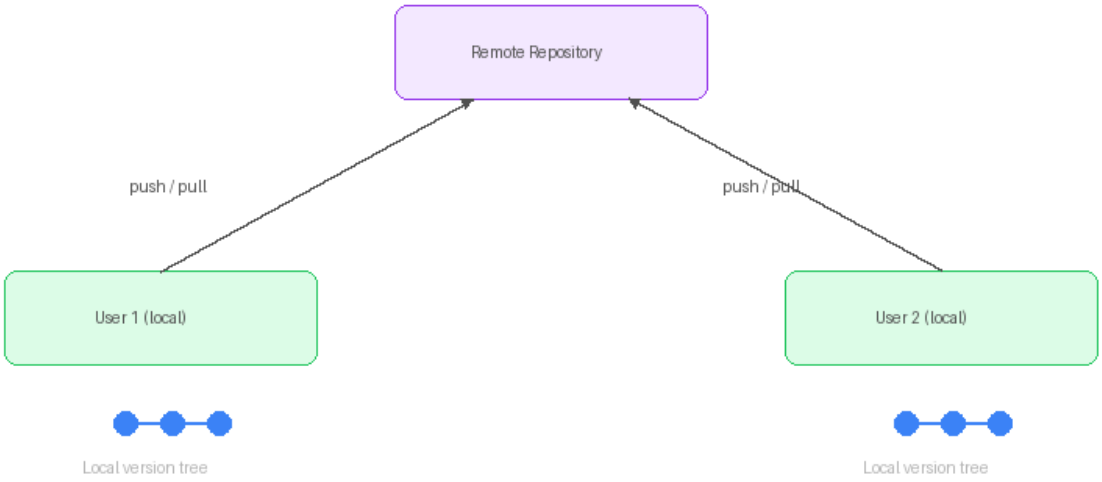

Figure 14: 분산형 VCS 다이어그램 — User1과 User2가 각각 로컬에 버전 트리를 가지고 있고, 원격 저장소와 동기화하는 구조

Figure 15: Git 로고 이미지

Git 기본 용어

용어	설명
Repository (Repo, 리포지토리)	Git 버전 트리의 저장소
Clone (클론)	원격의 버전 트리를 로컬에 처음으로 가져옴
Commit (커밋)	버전 스냅샷
HEAD (헤드)	가장 최신 커밋
Push (푸시)	로컬의 버전 트리를 원격에 업로드
Pull (풀)	원격의 버전 트리를 로컬로 다운로드
Stash (스태시)	로컬의 변경사항을 저장해두고 HEAD로 되돌림
.gitignore	원격에 업로드하지 말아야 할 파일 정의 (바이너리, 비밀번호 등)

Github 기본 용어

용어	설명
Github (깃헙)	원격 Git 저장소들을 모아놓은 현재 가장 유명한 웹사이트
Fork (포크)	다른 사람의 Git 저장소를 내 계정으로 복제
Pull Request (PR)	Fork된 저장소의 변경 사항을 원본 저장소에 merge 해달라고 요청하는 것
Github Actions	Github상에서 동작하는 자동화 CI 도구
Github Copilot	Github에서 개발한 AI 기반 코드 어시스트 도구

Github에 저장소 만들기

저장소를 만드는 방법은 두 가지가 있다.

1. 로컬에 Git 저장소 생성 후 Github에 업로드
2. **Github에 저장소 생성 후 로컬에 clone** (이 방법이 더 쉬움)

물론 반드시 원격 저장소로 Github을 쓸 필요는 없다. Bitbucket, Gitlab, Phorge 등 다른 서비스도 활용 가능하다.

2.3 Git 사용 시나리오

시나리오 1: 혼자 작업하기 (Clone → Commit → Push)

가장 기본적인 워크플로우이다. 원격 저장소를 Clone한 뒤, 파일을 수정하고 Commit, 그리고 Push하는 과정이다.

1단계: Clone

원격 저장소를 로컬로 복제한다.

2단계: Commit

파일을 수정한 뒤, 변경 사항을 커밋한다.

3단계: Push

커밋한 내용을 원격 저장소에 업로드한다.

시나리오 2: 함께 작업하기 (충돌 해결)

두 명 이상이 같은 저장소에서 작업할 때 발생할 수 있는 상황이다.

1단계: User1과 User2가 각각 Clone

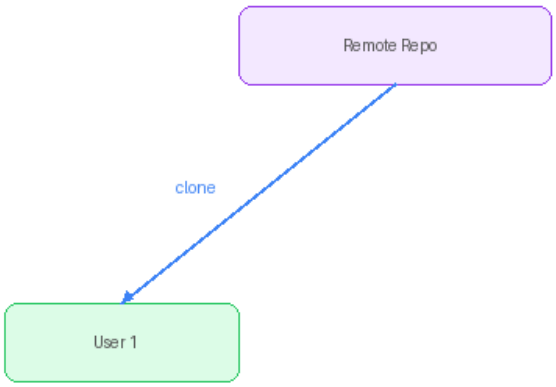

Figure 16: 시나리오 1 다이어그램 — User1이 원격 저장소에서 Clone하는 모습

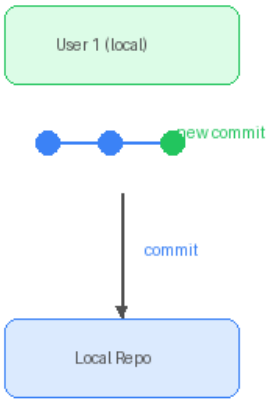

Figure 17: 시나리오 1 다이어그램 — User1이 로컬에서 Commit하는 모습

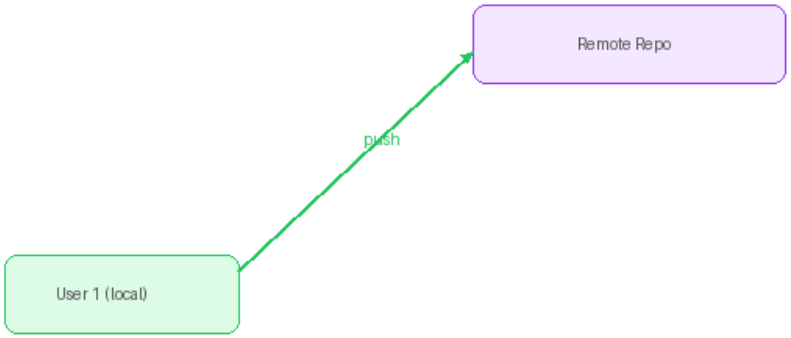

Figure 18: 시나리오 1 다이어그램 — User1이 원격 저장소로 Push하는 모습

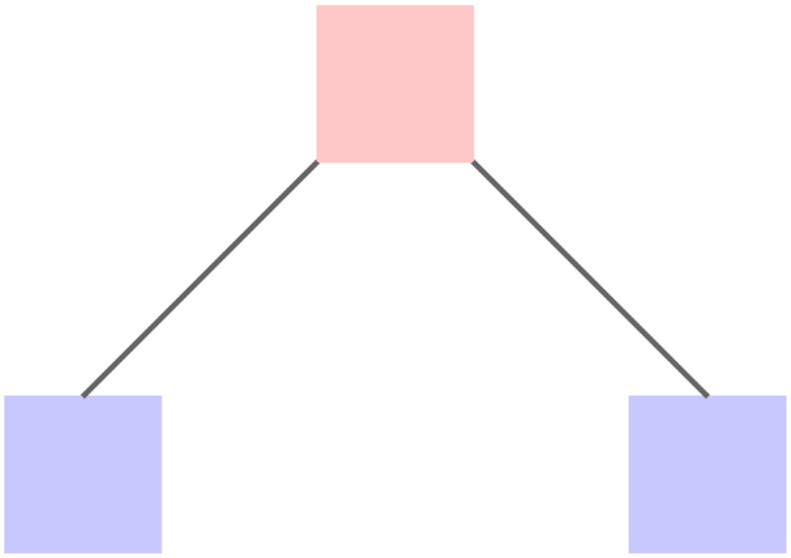

Figure 19: 시나리오 2 다이어그램 — User1과 User2가 동일한 원격 저장소를 Clone하는 모습

2단계: User1이 먼저 Commit & Push

User1이 파일을 수정하고 커밋한 뒤, 원격 저장소에 Push한다.

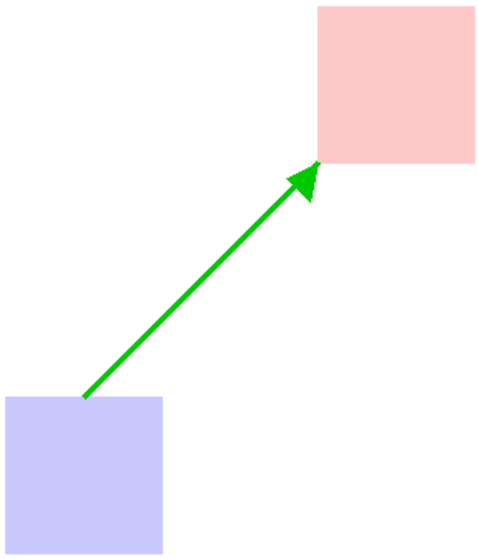

Figure 20: 시나리오 2 다이어그램 — User1이 Commit 후 Push하는 모습

3단계: User2도 Commit 후 Push 시도 — 실패!

User2도 파일을 수정하고 커밋한 뒤 Push를 시도하지만, User1이 이미 Push한 변경 사항이 있으므로 **Push가 거부**된다.

4단계: User2가 Pull

먼저 원격의 변경 사항을 Pull로 가져온다.

5단계: 충돌(Conflict) 해결

같은 파일의 같은 부분을 수정했다면 **merge conflict(병합 충돌)**이 발생한다. 충돌이 발생하면 해당 파일에 다음과 같은 표시가 나타난다.

충돌을 수동으로 해결한 뒤, 다시 커밋하고 Push하면 된다.

> 참고: Github 웹 인터페이스에서도 merge conflict를 해결할 수 있다. https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/addressing-merge-conflicts/resolving-a-merge-conflict-on-github

6단계: Merge 후 Push

충돌을 해결하고 merge 커밋을 생성한 뒤, 최종적으로 Push한다.

시나리오 3: Stash 활용하기

파일을 수정했으나 아직 Commit하지 않은 상태에서, Pull을 깜빡 잊었을 때 사용하는 방법이다.

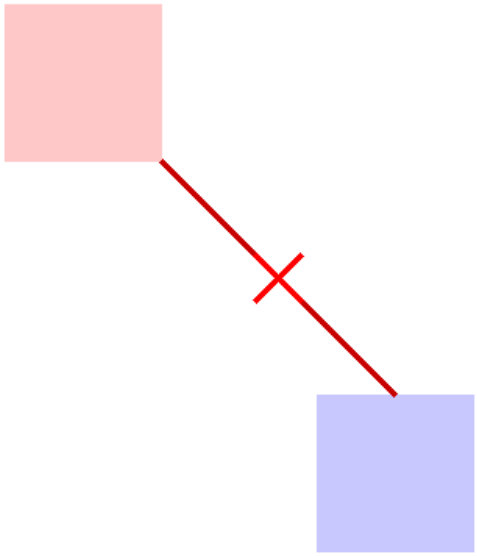

Figure 21: 시나리오 2 다이어그램 — User2의 Push가 거부되는 모습

Figure 22: Git Push 거부 시 터미널 에러 메시지 스크린샷

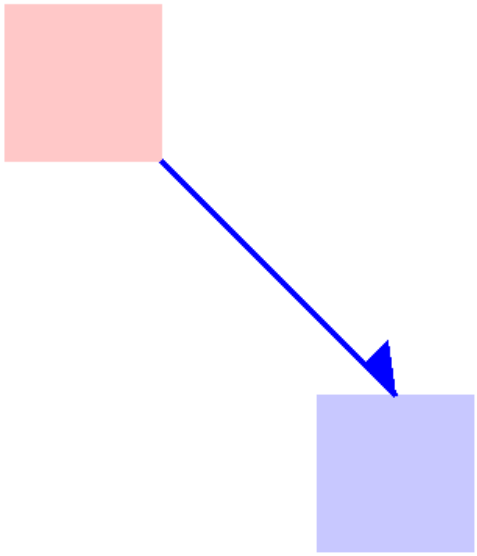

Figure 23: 시나리오 2 다이어그램 — User2가 Pull하는 모습

Figure 24: Merge conflict 표시가 된 파일 내용 스크린샷 — ««««‹ HEAD, =======, »»»»› 마커가 보이는 모습

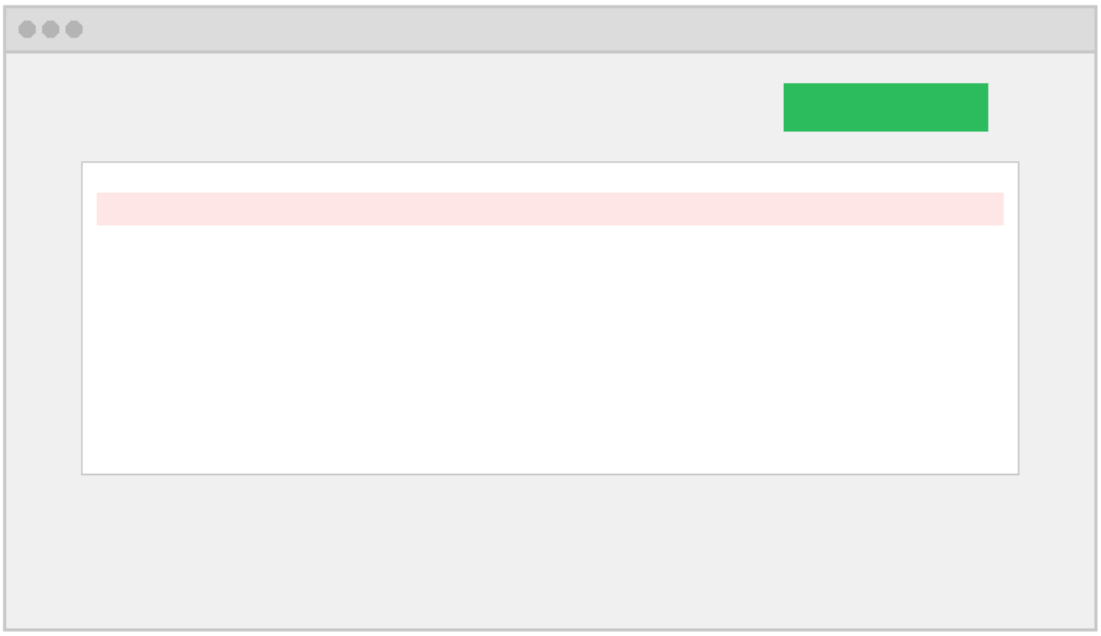

Figure 25: Github 웹에서 merge conflict 해결하는 화면 스크린샷

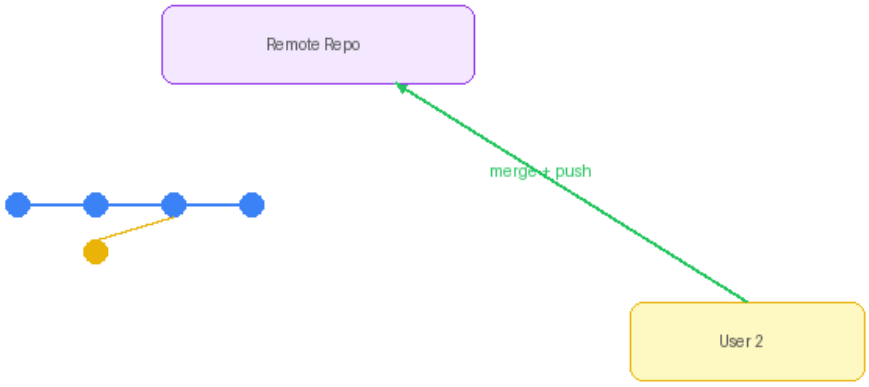

Figure 26: 시나리오 2 다이어그램 — User2가 Merge 후 Push하는 모습, 버전 트리에 merge branch가 표시됨

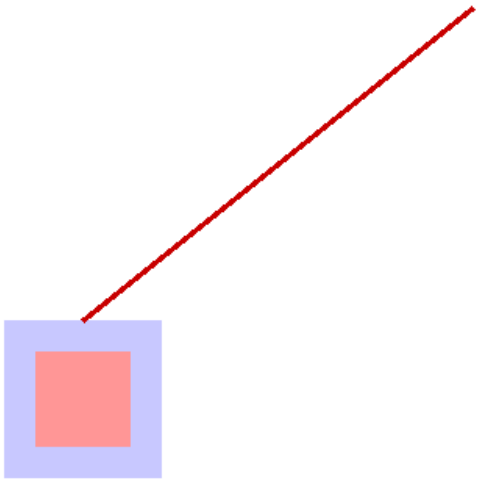

Figure 27: 시나리오 3 다이어그램 — User2가 파일을 수정했으나 Commit 전에 Pull이 필요한 상황

Git stash: 로컬의 변경사항을 임시로 저장하고 HEAD로 되돌린다. 로컬 변경사항이 없어지므로 `git pull`이 가능해진다.

이제 `git stash pop` 명령으로 임시로 저장했던 변경 사항을 다시 불러온다. 만약 conflict가 발생하면 시나리오 2와 동일한 방식으로 해결할 수 있다.

2.4 Github에서 협업하기

다른 개발자의 프로젝트에 기여하기

다른 개발자의 프로젝트에 기여하는 과정은 다음과 같다.

1. 다른 개발자의 저장소를 **Fork**한다
2. Fork한 저장소를 로컬에 **Clone**한다
3. 자유롭게 수정한 후 Github에 **Push**한다 (이때 Push되는 저장소는 내가 Fork한 저장소)
4. Github 상에서 **Pull Request**를 보낸다
5. 상대방이 변경 사항을 **Merge**해 주면 기여 완료

Fork

Fork는 다른 개발자의 프로젝트(쓰기 권한 없음)를 내 계정의 프로젝트(쓰기 권한 있음)로 복제하는 것이다.

Pull Request

Fork한 저장소에서 변경 사항을 Push한 뒤, 원본 저장소에 Pull Request를 열어 변경 사항의 반영을 요청한다.

Figure 28: git stash 실행 전후 터미널 스크린샷

Figure 29: git stash pop 실행 후 터미널 스크린샷

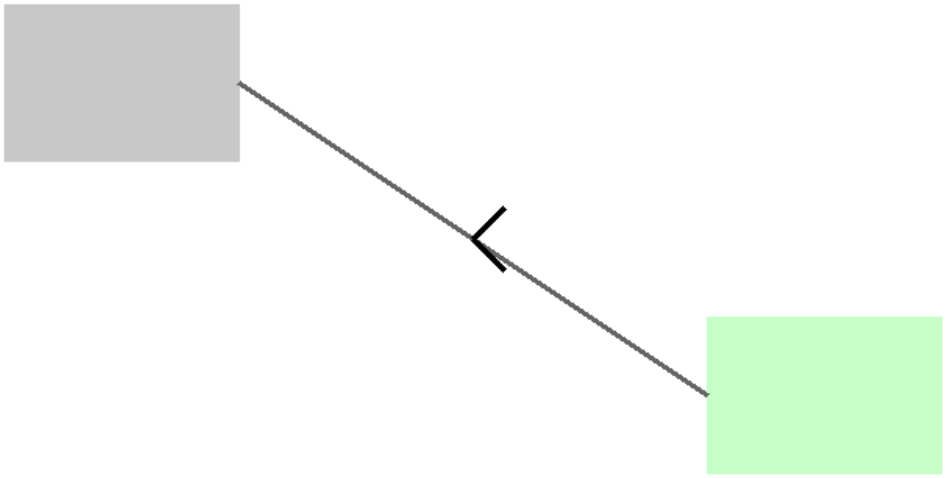

Figure 30: Github Fork 개념 다이어그램 — 원본 저장소에서 내 계정으로 Fork되는 구조

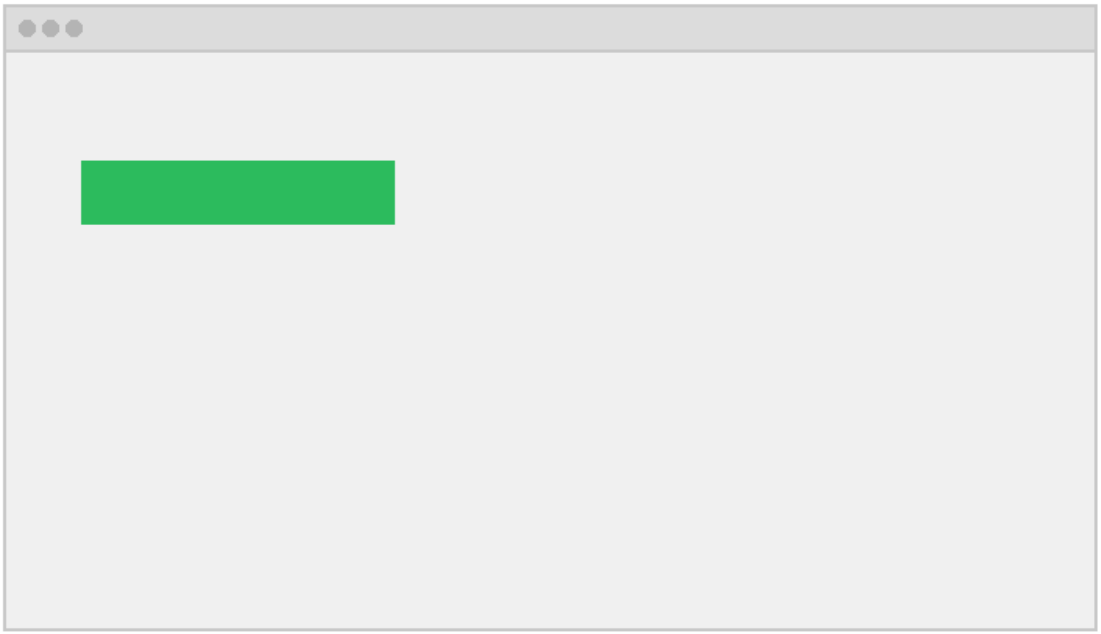

Figure 31: Github에서 Pull Request 생성 화면 스크린샷

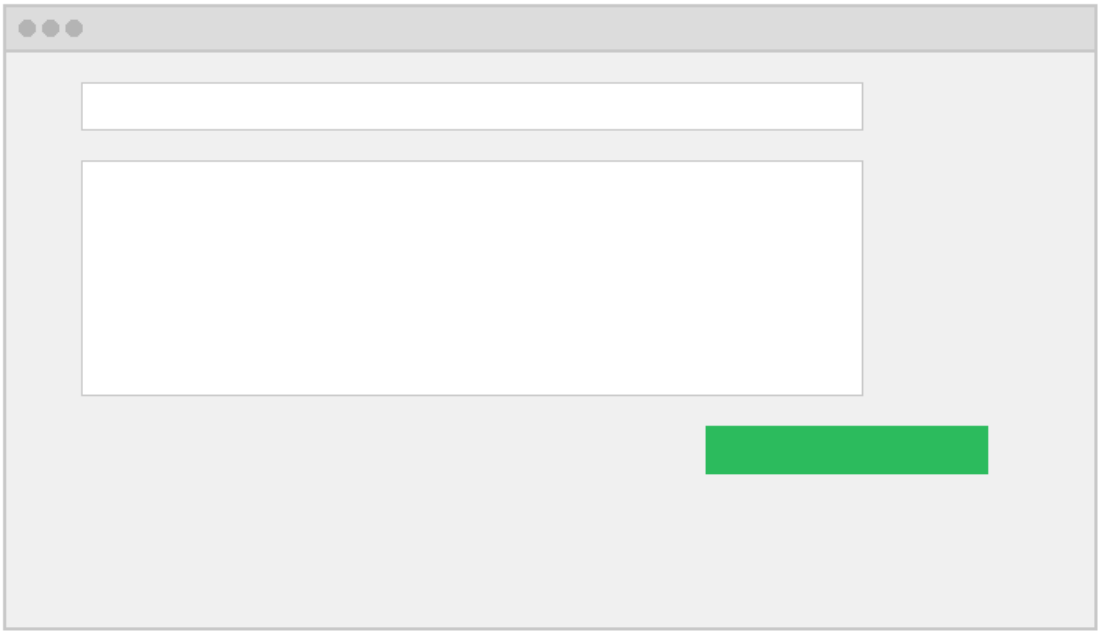

Figure 32: Pull Request 상세 내용 작성 화면 스크린샷

Figure 33: Pull Request가 생성된 후의 화면 스크린샷

2.5 정리

- **Git 사용법을 확실히 숙지할 것**
  - Clone, Push, Pull 등 기본 활용법
  - 일하기 시작 전 항상 `git pull`부터 실행하기
  - Commit은 자주 하기 (특히 task를 마치고 나서는 반드시 commit)
- **Github 상에서 협업하는 방법 알기**
  - 서로의 repository에 Pull Request를 날려보고 merge 해보기

3장. Docker

3.1 Docker란?

Docker는 애플리케이션을 **컨테이너(Container)**라는 격리된 환경에서 실행할 수 있게 해주는 도구이다. 컨테이너는 운영체제, 라이브러리, 설정 파일 등을 모두 포함하고 있어, 어떤 컴퓨터에서든 동일한 환경으로 프로그램을 실행할 수 있다.

왜 Docker가 필요한가?

생명정보학 도구를 개발하다 보면 다음과 같은 문제를 자주 겪게 된다:

- "내 컴퓨터에서는 되는데 다른 컴퓨터에서는 안 돼요"
- 파이썬 버전, 라이브러리 버전 충돌
- 운영체제마다 설치 방법이 다름
- 데이터베이스, 웹 서버 등 여러 서비스를 동시에 관리해야 함

Docker를 사용하면 이러한 문제를 해결할 수 있다. 개발 환경을 코드로 정의하여 누구나 동일한 환경을 재현할 수 있다.

Figure 34: Docker 없이 개발할 때의 환경 차이 문제 vs Docker를 사용할 때의 일관된 환경을 비교하는 다이어그램

가상 머신과의 차이

Docker 컨테이너는 가상 머신(VM)과 비슷해 보이지만, 중요한 차이가 있다. 가상 머신은 운영체제 전체를 포함하므로 무겁고 느린 반면, 컨테이너는 호스트 운영체제의 커널을 공유하므로 가볍고 빠르다.

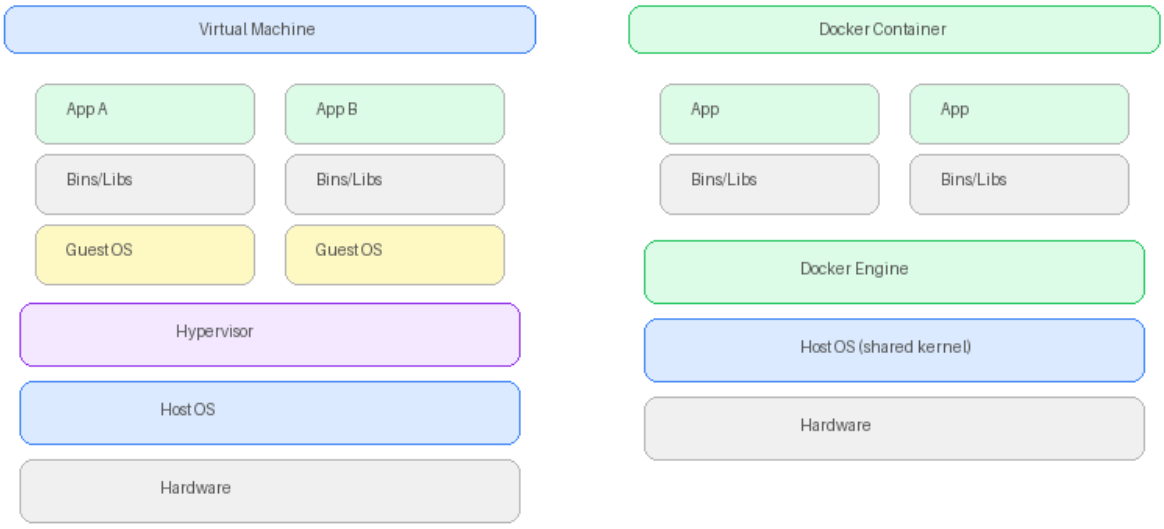

Figure 35: 가상 머신 vs Docker 컨테이너 아키텍처 비교 다이어그램 — VM은 Guest OS 포함, Container는 커널 공유

3.2 Docker 설치

이 책에서는 모든 개발을 WSL(Windows) 또는 네이티브 리눅스/macOS 환경에서 진행한다. Docker도 WSL 내에서 직접 설치한다. Windows 사용자는 1장에서 설치한 WSL Ubuntu 터미널을 열고 진행한다.

터미널에서 다음 명령을 순서대로 실행한다:

```
# Docker   GPG
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Docker
echo \\
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docke
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \\
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

# Docker
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

#      docker      (sudo   docker    )
sudo usermod -aG docker $USER
```

설치 후 WSL 터미널을 재시작한다.

설치 확인

터미널에서 다음 명령을 실행하여 Docker가 정상적으로 설치되었는지 확인한다:

```
docker --version
docker run hello-world
```

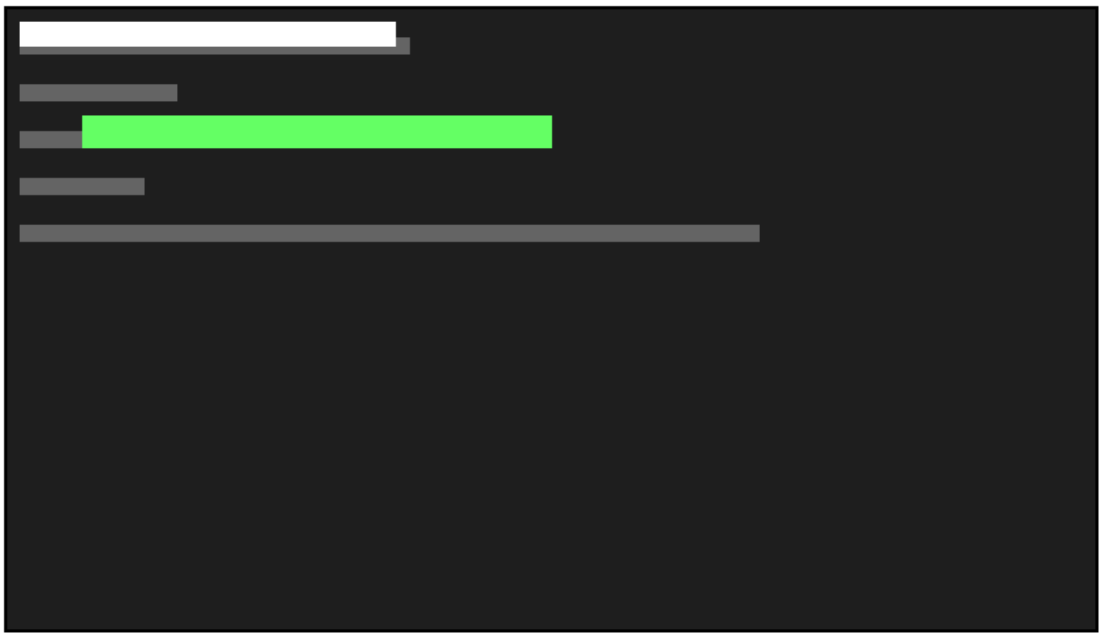

Figure 36: docker run hello-world 실행 시 “Hello from Docker!” 메시지가 출력되는 터미널 스크린샷

3.3 Docker 기본 개념

이미지 (Image)

Docker 이미지는 컨테이너를 만들기 위한 **설계도**이다. 운영체제, 프로그램, 설정 파일 등이 모두 포함되어 있다. Docker Hub(https://hub.docker.com/)에서 다양한 공식 이미지를 다운로드할 수 있다.

컨테이너 (Container)

컨테이너는 이미지를 기반으로 **실제로 실행되는 인스턴스**이다. 하나의 이미지로 여러 개의 컨테이너를 만들 수 있다.

Dockerfile

Dockerfile은 Docker 이미지를 만들기 위한 **레시피 파일**이다. 어떤 기반 이미지를 사용하고, 어떤 파일을 복사하고, 어떤 명령을 실행할지 순서대로 기술한다.

```
FROM node:20-alpine
WORKDIR /app
COPY package.json .
RUN npm install
COPY . .
CMD ["npm", "run", "dev"]
```

Docker Compose

여러 개의 컨테이너를 함께 관리해야 할 때 사용하는 도구이다. 예를 들어 웹 서버와 데이터베이스를 동시에 실행해야 하는 경우, `compose.yml` 파일 하나로 모든 서비스를 정의하고 한 번에 실행할 수 있다.

```yaml
services:
  web:
    build: .
    ports:
      - "3000:3000"
  db:
    image: postgres:16
    environment:
      POSTGRES_PASSWORD: mysecret
```

실행은 다음 명령 하나로 가능하다:

```
docker compose up
```

Figure 37: docker compose up 실행 시 웹 서버와 데이터베이스가 동시에 시작되는 터미널 출력 스크린샷

3.4 자주 사용하는 Docker 명령어

명령어	설명
`docker compose up`	compose.yml에 정의된 모든 서비스 시작
`docker compose up -d`	백그라운드에서 서비스 시작
`docker compose down`	모든 서비스 종료
`docker compose logs`	서비스 로그 확인

명령어	설명
`docker ps`	실행 중인 컨테이너 목록
`docker exec -it <  > bash`	실행 중인 컨테이너에 접속

3.5 정리

- **Docker는 개발 환경을 코드로 정의하여 일관된 환경을 재현하는 도구**
  - "내 컴퓨터에서는 되는데" 문제를 해결
- **Dockerfile로 이미지를 정의하고, Docker Compose로 여러 서비스를 관리**
  - 웹 서버 + 데이터베이스 등을 한 번에 실행 가능
- **Windows 사용자는 WSL 위에 Docker를 설치하여 사용**

4장. Python 데이터 분석 기초

4.1 왜 Python인가?

생명정보학에서 Python은 사실상 표준 프로그래밍 언어이다. 데이터 처리, 시각화, 통계 분석, 머신러닝까지 거의 모든 작업을 Python 생태계 안에서 수행할 수 있다.

바이브 코딩에서는 이 패키지들의 문법을 암기할 필요가 없다. 대신 **각 패키지가 무엇을 할 수 있는지**를 알고, AI에게 정확히 요청하는 것이 중요하다. 이 장에서는 핵심 패키지들의 역할과 주요 개념을 소개한다.

4.2 패키지 설치

WSL 환경에서 필요한 패키지를 설치한다:

```
pip install pandas numpy matplotlib seaborn scipy
```

> **팁**: AI에게 "pandas로 CSV 파일 읽어서 히스토그램 그려줘"라고 요청하면, AI가 알아서 import문과 코드를 작성해준다. 하지만 **어떤 패키지가 어떤 역할을 하는지** 알아야 AI의 결과물이 맞는지 판단할 수 있다.

4.3 pandas — 테이블 데이터 처리

pandas는 표(테이블) 형태의 데이터를 다루는 패키지이다. 엑셀 스프레드시트를 Python에서 다룬다고 생각하면 된다.

핵심 개념

- **DataFrame**: 행과 열로 이루어진 2차원 테이블. pandas의 핵심 자료구조이다.
- **Series**: DataFrame의 한 열(column). 1차원 데이터이다.
- **Index**: 각 행을 식별하는 라벨.

주요 작업

```
import pandas as pd

# CSV
df = pd.read_csv("gene_expression.csv")

#
df.head()          #   5
df.shape           # (  ,   )
df.describe()      #
```

```
#
df["gene_name"]                #
df[["gene_name", "logFC"]]     #

#
significant = df[df["pvalue"] < 0.05]

#
df.sort_values("logFC", ascending=False)

#
df["neg_log_p"] = -np.log10(df["pvalue"])
```

	gene_id	gene_name	logFC	pvalue
0	ENSG001	GAPDH	1.23	0.0012
1	ENSG002	ACTB	-0.54	0.1200
2	ENSG003	TP53	2.41	0.0001
3	ENSG004	MT-CO1	0.82	0.0340
4	ENSG005	IL6	-1.80	0.0050

[5 rows x 4 columns]

Figure 38: pandas DataFrame을 Jupyter Notebook에서 출력한 예시 스크린샷

AI에게 요청하는 예시

> "gene_expression.csv 파일을 읽어서 pvalue가 0.05 미만인 유전자만 필터링하고, logFC 기준으로 내림차순 정렬해서 상위 20개를 보여줘"

이 요청을 하려면 **pvalue, logFC가 무엇인지, 필터링과 정렬이라는 개념**을 알아야 한다. 코드 문법은 몰라도 되지만, 데이터의 의미는 사람이 이해하고 있어야 한다.

4.4 NumPy — 수치 연산

NumPy는 대규모 수치 데이터를 빠르게 처리하는 패키지이다. pandas의 내부에서도 NumPy를 사용한다.

핵심 개념

- **ndarray**: N차원 배열. 같은 타입의 데이터를 담는 고성능 자료구조이다.
- **브로드캐스팅**: 크기가 다른 배열 간 연산을 자동으로 확장하는 기능.
- **벡터 연산**: 반복문 없이 배열 전체에 연산을 한 번에 적용한다.

주요 작업

```
import numpy as np

#
arr = np.array([1, 2, 3, 4, 5])
matrix = np.zeros((100, 100))    # 100x100

#
arr.mean()      #
arr.std()       #
arr.max()       #

#     (    )
log_values = np.log2(arr + 1)    #      log2

#
random_data = np.random.normal(0, 1, size=1000)  #
```

> **바이브 코딩에서의 활용**: NumPy 자체를 직접 쓸 일은 많지 않지만, AI가 생성하는 코드에 자주 등장한다. `np.log2`, `np.mean` 같은 표현이 나왔을 때 무엇을 하는 코드인지 이해할 수 있으면 충분하다.

4.5 Matplotlib — 기본 시각화

Matplotlib은 Python의 가장 기본적인 시각화 패키지이다. 거의 모든 종류의 그래프를 그릴 수 있다.

핵심 개념

- **Figure**: 전체 그림 영역. 하나의 Figure 안에 여러 그래프를 배치할 수 있다.
- **Axes**: 개별 그래프 영역. 실제로 데이터가 그려지는 공간이다.
- **Subplot**: Figure를 격자로 나누어 여러 그래프를 배치하는 방식.

주요 그래프 유형

```
import matplotlib.pyplot as plt

#    (Scatter plot)
plt.scatter(df["logFC"], df["neg_log_p"])
plt.xlabel("Log Fold Change")
plt.ylabel("-log10(p-value)")
plt.title("Volcano Plot")
plt.savefig("volcano.png", dpi=300)
plt.show()

#
plt.hist(df["logFC"], bins=50)
plt.xlabel("Log Fold Change")
plt.ylabel("Frequency")
plt.show()
```

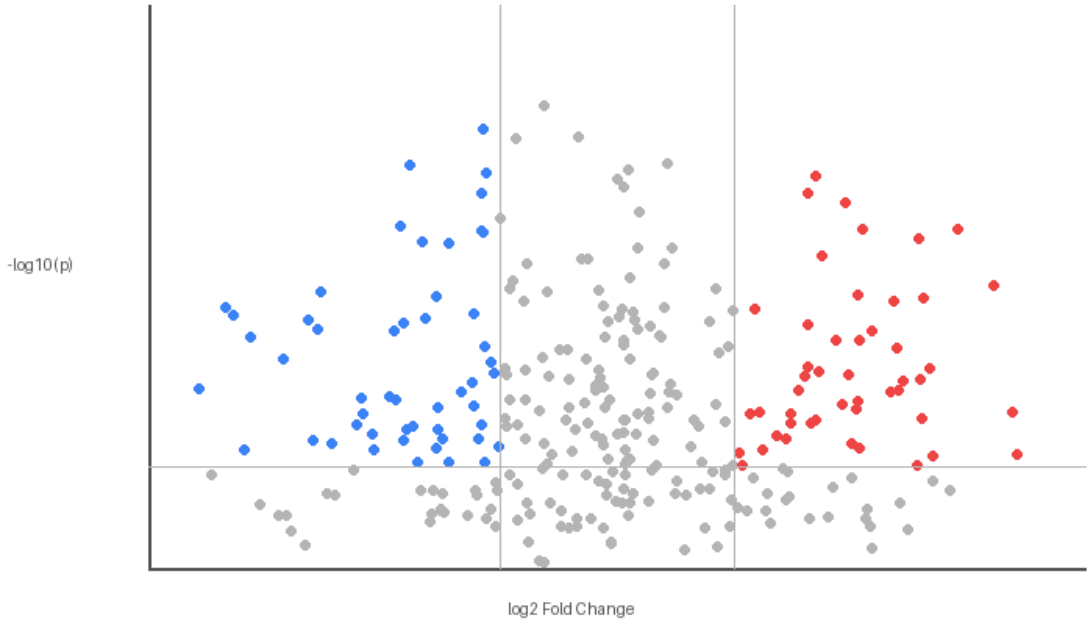

Figure 39: Matplotlib으로 그린 Volcano Plot 예시

```
#          (Subplot)
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
axes[0].hist(df["logFC"], bins=50)
axes[0].set_title("Distribution of logFC")
axes[1].scatter(df["logFC"], df["neg_log_p"], s=1)
axes[1].set_title("Volcano Plot")
plt.tight_layout()
plt.savefig("combined.png", dpi=300)
```

AI에게 요청하는 예시

> "logFC와 −log10(pvalue)로 volcano plot 그려줘. significant한 유전자(pvalue 〈 0.05, |logFC| 〉 1)는 빨간색으로 표시하고, 나머지는 회색으로. 그래프 해상도는 300 dpi로 저장해줘"

4.6 Seaborn — 통계 시각화

Seaborn은 Matplotlib 위에 구축된 **통계 시각화 전문 패키지**이다. 더 적은 코드로 보기 좋은 통계 그래프를 만들 수 있다.

Matplotlib과의 차이

	Matplotlib	Seaborn
수준	저수준 (세밀한 제어)	고수준 (간결한 코드)
스타일	기본 스타일 단순함	기본 스타일이 깔끔함
통계 기능	직접 구현 필요	회귀선, 분포 등 내장

	Matplotlib	Seaborn
DataFrame 연동	수동	직접 지원

주요 그래프 유형

```
import seaborn as sns

#     -
sns.boxplot(data=df, x="cell_type", y="expression")
plt.xticks(rotation=45)
plt.show()

#    -
sns.heatmap(expression_matrix, cmap="RdBu_r", center=0)
plt.title("Gene Expression Heatmap")
plt.show()

#      -
sns.violinplot(data=df, x="condition", y="expression")
plt.show()

#    +
sns.regplot(data=df, x="gene_A", y="gene_B")
plt.show()
```

AI에게 요청하는 예시

> "cell_type별 gene expression의 분포를 violin plot으로 비교해줘. 색상은 pastel 팔레트 사용하고, 각 그룹의 데이터 포인트도 strip plot으로 겹쳐서 보여줘"

4.7 SciPy — 과학 계산과 통계 검정

SciPy는 과학 계산에 필요한 다양한 알고리즘을 제공한다. 생명정보학에서는 주로 **통계 검정** 기능을 사용한다.

자주 사용하는 통계 검정

```
from scipy import stats

# t-test -
t_stat, p_value = stats.ttest_ind(group_a, group_b)
print(f"t-statistic: {t_stat:.4f}, p-value: {p_value:.4e}")

# Mann-Whitney U test -      (       )
u_stat, p_value = stats.mannwhitneyu(group_a, group_b)

# Pearson
corr, p_value = stats.pearsonr(gene_a_expression, gene_b_expression)
print(f"Correlation: {corr:.4f}, p-value: {p_value:.4e}")

#       (Benjamini-Hochberg)
from scipy.stats import false_discovery_control
adjusted_pvalues = false_discovery_control(p_values, method='bh')
```

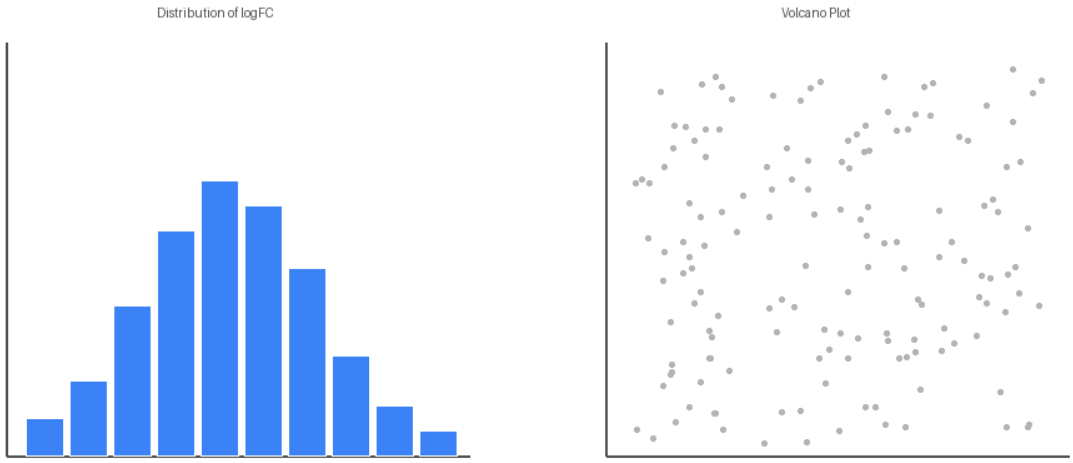

Figure 40: Subplot으로 두 개의 그래프를 나란히 배치한 예시

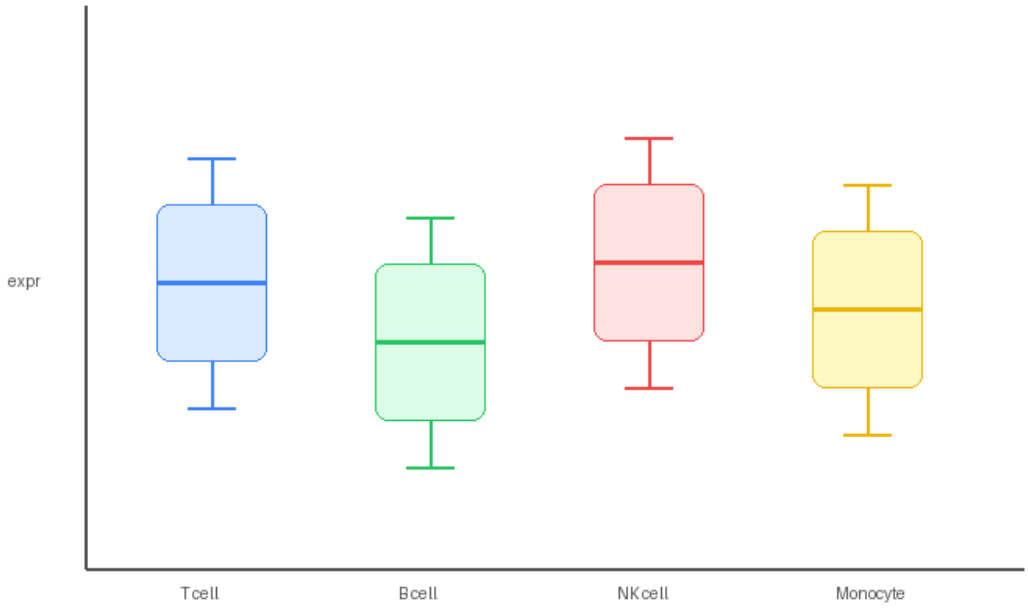

Figure 41: Seaborn 박스 플롯 예시 — 세포 유형별 유전자 발현량 분포

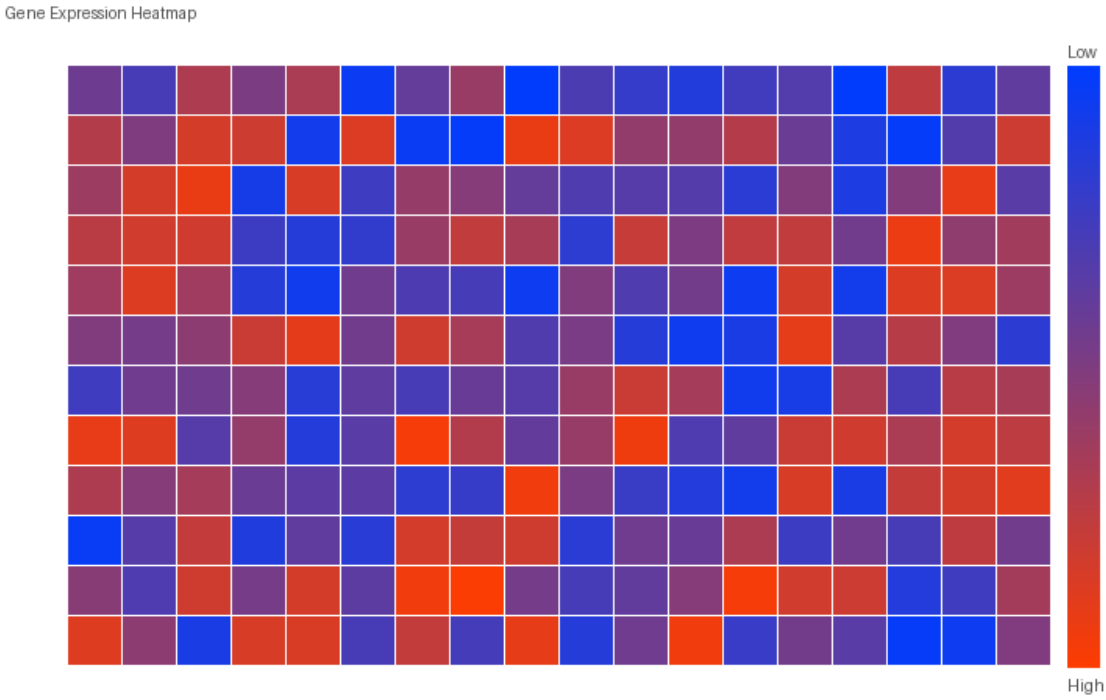

Figure 42: Seaborn 히트맵 예시 — 유전자 발현 매트릭스

AI에게 요청하는 예시

> "treatment 그룹과 control 그룹의 gene expression을 t-test로 비교해줘. p-value가 0.05 미만인 유전자 목록을 뽑아주고, Benjamini-Hochberg 보정도 적용해줘"

이 요청에서 **t-test가 무엇인지**, **다중 검정 보정이 왜 필요한지**를 이해하고 있어야 AI의 결과를 올바르게 해석할 수 있다.

4.8 바이브 코딩 실전: AI와 함께하는 데이터 분석

워크플로우

바이브 코딩으로 데이터 분석을 할 때의 일반적인 흐름이다:

1. **데이터 확인**: "이 CSV 파일의 구조를 보여줘" → AI가 `pd.read_csv`와 `df.head()`, `df.describe()` 실행
2. **전처리**: "결측값 제거하고, gene_name 열을 인덱스로 설정해줘" → AI가 `dropna()`, `set_index()` 적용
3. **분석**: "두 그룹 간 차이가 있는 유전자를 찾아줘" → AI가 통계 검정 수행
4. **시각화**: "결과를 volcano plot으로 그려줘" → AI가 Matplotlib/Seaborn으로 시각화
5. **해석**: 결과를 사람이 확인하고, 추가 분석 방향을 지시

핵심 포인트

사람이 해야 할 일	AI가 해주는 일
분석 목표 설정	코드 작성
적절한 분석 방법 선택	패키지 import 및 함수 호출
결과 해석	그래프 생성 및 통계량 계산
생물학적 의미 판단	데이터 전처리 및 변환

핵심: 코드를 외울 필요는 없다. 하지만 **"박스 플롯은 분포를 비교할 때 쓴다"**, **"t-test는 두 그룹의 평균을 비교한다"**, **"p-value가 작을수록 통계적으로 유의하다"** 같은 개념은 반드시 이해해야 한다. 이것이 AI에게 올바른 지시를 내리고, AI의 결과를 검증하는 힘이 된다.

4.9 정리

- **pandas**: 테이블 데이터(CSV, TSV) 읽기, 필터링, 정렬, 집계
- **NumPy**: 수치 배열 연산, 수학 함수 (log, mean, std 등)
- **Matplotlib**: 기본 그래프 (산점도, 히스토그램, 서브플롯)
- **Seaborn**: 통계 시각화 (박스 플롯, 히트맵, 바이올린 플롯)
- **SciPy**: 통계 검정 (t-test, 상관분석, 다중 검정 보정)
- **바이브 코딩의 핵심**: 문법이 아닌 **개념**을 이해하고, AI에게 정확히 요청하는 것

5장. Scanpy를 이용한 단일세포 데이터 분석

5.1 단일세포 분석이란?

전통적인 bulk RNA-seq은 수천~수백만 개의 세포를 한꺼번에 분석하여 **평균적인** 유전자 발현을 측정한다. 반면, 단일세포 RNA-seq(scRNA-seq)은 **개별 세포 하나하나**의 유전자 발현을 측정한다.

이를 통해 다음과 같은 질문에 답할 수 있다:

- 이 조직에는 어떤 종류의 세포들이 있는가?
- 각 세포 유형의 비율은 어떻게 되는가?
- 특정 질환에서 어떤 세포 유형이 변화하는가?

Scanpy는 이러한 단일세포 데이터를 분석하기 위한 Python 패키지이다.

5.2 패키지 설치

```
pip install scanpy anndata mudata
```

- **scanpy**: 단일세포 분석의 핵심 패키지
- **anndata**: h5ad 파일 형식을 다루는 패키지
- **mudata**: h5mu 파일(멀티오믹스 데이터)을 다루는 패키지

5.3 AnnData — 단일세포 데이터 구조

h5ad 파일이란?

h5ad는 단일세포 데이터의 **표준 저장 형식**이다. HDF5 기반으로, 대용량 데이터를 효율적으로 저장하고 읽을 수 있다.

하나의 h5ad 파일에는 다음 정보가 모두 담겨 있다:

속성	설명	예시
`X`	유전자 발현 매트릭스 (세포 × 유전자)	10,000 세포 × 20,000 유전자
`obs`	세포(행)에 대한 메타데이터	cell_type, sample_id, condition
`var`	유전자(열)에 대한 메타데이터	gene_name, highly_variable
`obsm`	세포의 임베딩 좌표	UMAP, t-SNE, PCA 좌표
`uns`	비구조화 데이터	색상 팔레트, 분석 파라미터

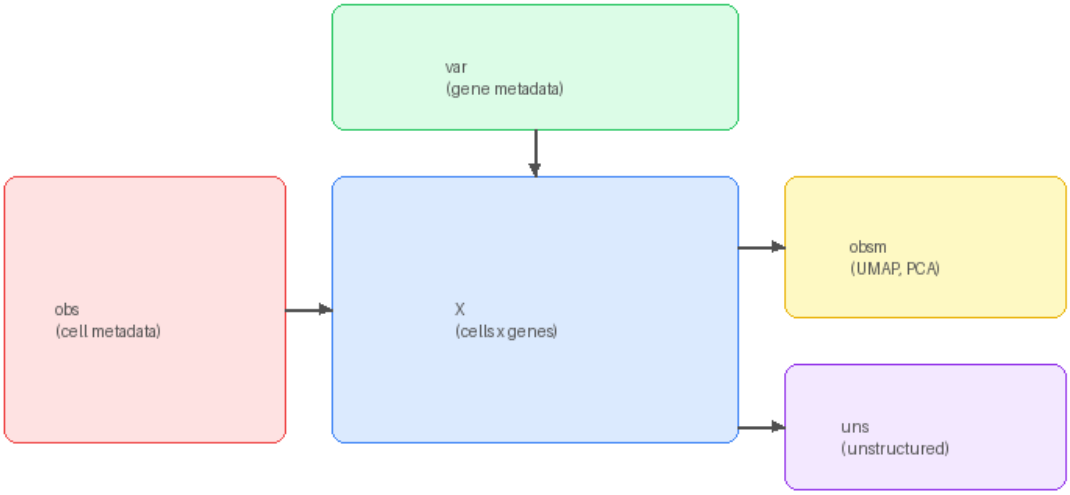

Figure 43: AnnData 객체의 구조를 나타낸 다이어그램

기본 사용법

```
import scanpy as sc
import anndata as ad

# h5ad
adata = sc.read_h5ad("pbmc_10k.h5ad")

#
print(adata)
# AnnData object with n_obs × n_vars = 10000 × 20000
#     obs: 'cell_type', 'sample', 'condition'
#     var: 'gene_name', 'highly_variable'
#     obsm: 'X_pca', 'X_umap'

#         (pandas DataFrame)
adata.obs.head()

#
adata.var.head()

#
t_cells = adata[adata.obs["cell_type"] == "T cell"]

#
adata[:, "CD3E"].X.toarray()
```

AI에게 요청하는 예시

> “pbmc_10k.h5ad 파일을 읽어서 어떤 cell_type이 있는지, 각 유형별 세포 수를 보여줘”

5.4 MuData — 멀티오믹스 데이터

h5mu 파일이란?

h5mu는 **멀티오믹스 데이터**를 저장하는 형식이다. 하나의 파일에 여러 종류의 데이터(예: RNA + ATAC, RNA + Protein)를 함께 담을 수 있다.

```
import mudata as md

# h5mu
mdata = md.read_h5mu("multiome.h5mu")

#   modality
print(mdata.mod)
# {'rna': AnnData object with n_obs × n_vars = 5000 × 20000,
#  'atac': AnnData object with n_obs × n_vars = 5000 × 100000}

#   modality
rna = mdata.mod["rna"]    # RNA    (AnnData)
atac = mdata.mod["atac"]  # ATAC    (AnnData)
```

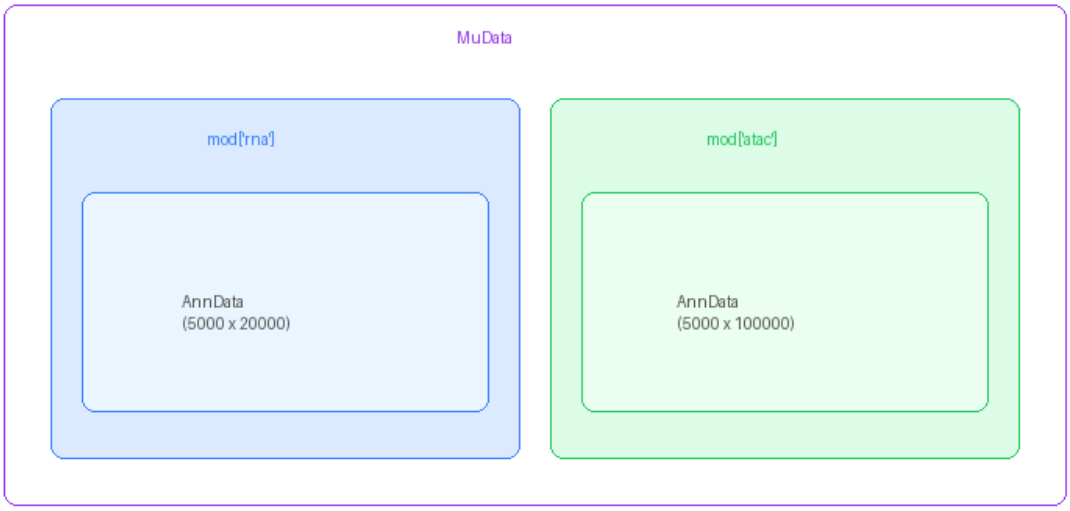

Figure 44: MuData 객체의 구조 다이어그램

5.5 Scanpy 분석 워크플로우

단일세포 분석은 보통 다음 순서로 진행된다:

1단계: 품질 관리 (Quality Control)

```
#
adata.var["mt"] = adata.var_names.str.startswith("MT-")
sc.pp.calculate_qc_metrics(adata, qc_vars=["mt"], inplace=True)

# QC
sc.pl.violin(adata, ["n_genes_by_counts", "total_counts", "pct_counts_mt"])
```

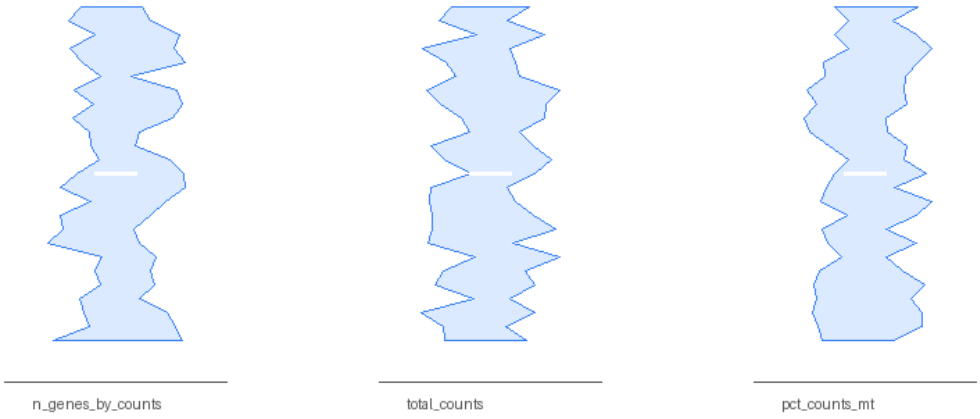

Figure 45: QC 바이올린 플롯

```
#
adata = adata[adata.obs["n_genes_by_counts"] > 200]
adata = adata[adata.obs["pct_counts_mt"] < 20]
```

2단계: 정규화 및 전처리

```
#
sc.pp.normalize_total(adata, target_sum=1e4)
sc.pp.log1p(adata)

#
sc.pp.highly_variable_genes(adata, n_top_genes=2000)
sc.pl.highly_variable_genes(adata)
```

3단계: 차원 축소

```
# PCA
sc.tl.pca(adata)
sc.pl.pca_variance_ratio(adata, n_pcs=50)

#        + UMAP
sc.pp.neighbors(adata, n_pcs=30)
sc.tl.umap(adata)
```

4단계: 클러스터링

```
# Leiden
sc.tl.leiden(adata, resolution=0.5)

# UMAP
sc.pl.umap(adata, color="leiden")
```

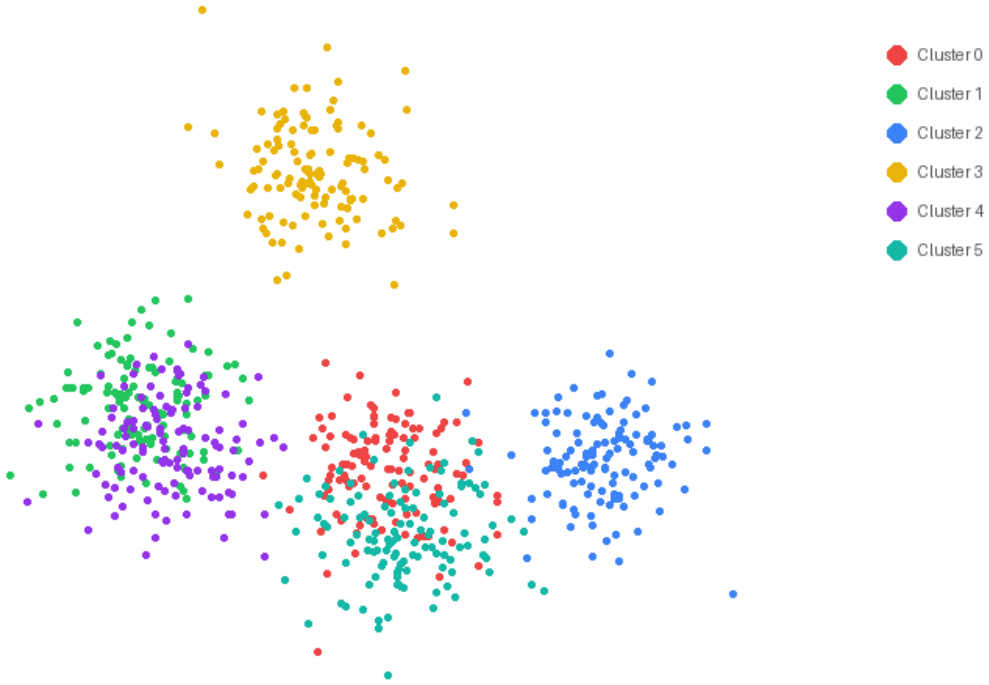

Figure 46: UMAP 플롯

5단계: 세포 유형 주석

```
#
sc.tl.rank_genes_groups(adata, groupby="leiden")
sc.pl.rank_genes_groups(adata, n_genes=10)

#         UMAP
sc.pl.umap(adata, color=["CD3E", "CD14", "MS4A1", "NKG7"])
```

6단계: 결과 저장

```
# h5ad
adata.write_h5ad("pbmc_analyzed.h5ad")
```

5.6 바이브 코딩으로 단일세포 분석하기

실전 대화 예시

단일세포 분석의 전체 과정을 AI와 대화하며 진행할 수 있다:

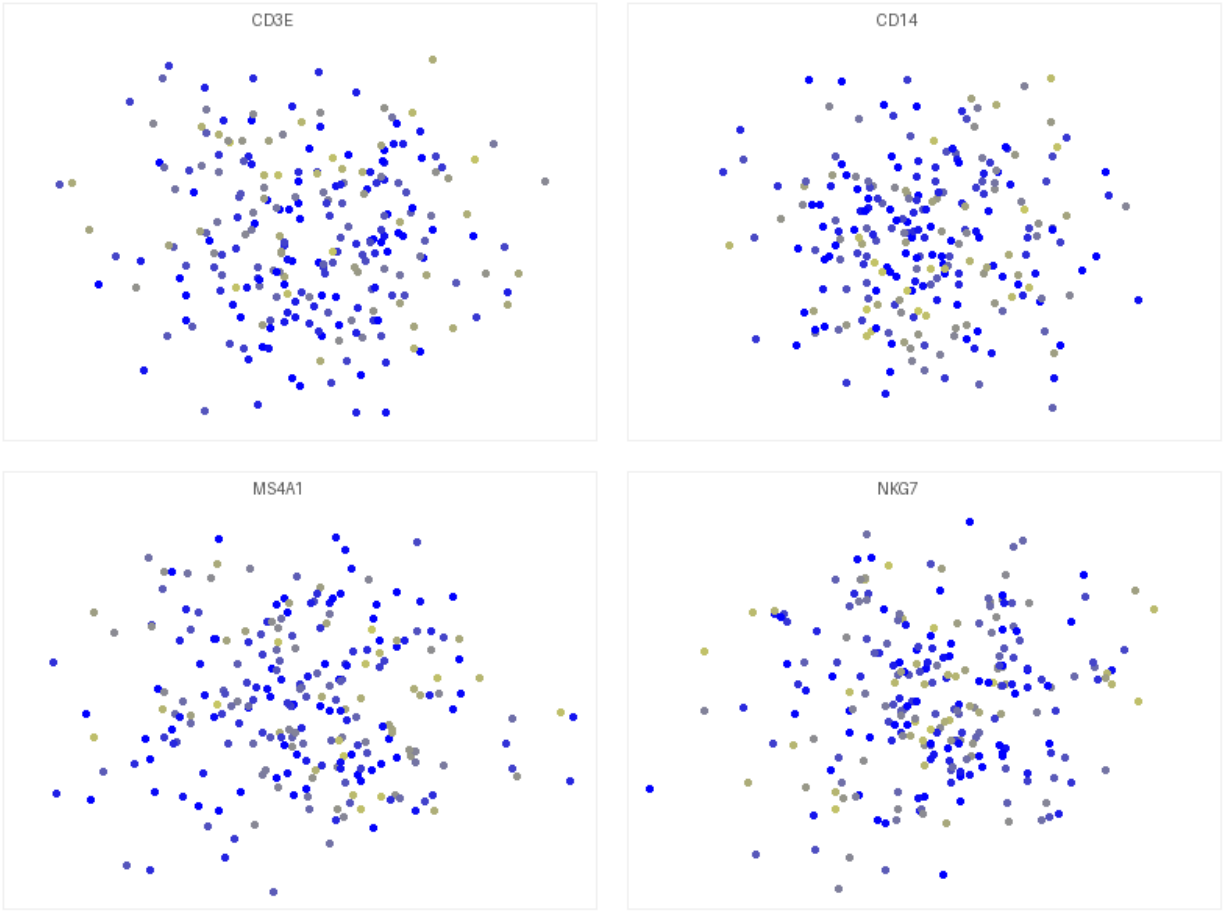

Figure 47: 마커 유전자별 UMAP 플롯

사용자: "pbmc_10k.h5ad 파일을 읽고 QC 해줘. 미토콘드리아 비율 20% 이상, 유전자 수 200개 미만인 세포는 제거해줘"

사용자: "정규화하고 고변동 유전자 2000개 선택해서 PCA, UMAP 해줘"

사용자: "Leiden 클러스터링 해주고, 각 클러스터의 마커 유전자 상위 5개씩 보여줘"

사용자: "CD3E, CD14, MS4A1, NKG7 마커로 봤을 때 각 클러스터가 어떤 세포 유형인지 UMAP에 표시해줘"

알아야 할 핵심 개념

코드를 외울 필요는 없지만, AI에게 올바른 지시를 내리려면 다음 개념들을 이해해야 한다:

개념	설명
QC (Quality Control)	품질이 낮은 세포를 걸러내는 과정
정규화 (Normalization)	세포 간 시퀀싱 깊이 차이를 보정
고변동 유전자 (HVG)	세포 간 발현 차이가 큰 유전자. 분석의 핵심
PCA	고차원 데이터를 주요 성분으로 압축
UMAP	고차원 데이터를 2D로 시각화
클러스터링	유사한 세포를 그룹으로 묶는 과정
마커 유전자	특정 세포 유형을 구분하는 유전자

5.7 정리

- **h5ad**: 단일세포 데이터의 표준 파일 형식. 발현 매트릭스, 세포/유전자 메타데이터, 임베딩 좌표를 하나의 파일에 저장
- **h5mu**: 멀티오믹스 데이터 형식. 여러 modality(RNA, ATAC 등)를 하나의 파일에 통합
- **AnnData**: h5ad의 Python 객체. `adata.obs`(세포 정보), `adata.var`(유전자 정보), `adata.X`(발현 매트릭스)로 구성
- **Scanpy 워크플로우**: QC → 정규화 → 고변동 유전자 선택 → PCA → UMAP → 클러스터링 → 세포 유형 주석
- **바이브 코딩의 핵심**: 분석 파이프라인의 각 단계가 **왜 필요한지**를 이해하고, AI에게 단계별로 지시하는 것

6장. Snakemake를 이용한 워크플로우 관리

6.1 워크플로우란?

생명정보학 분석은 보통 여러 단계를 순서대로 수행한다. 예를 들어 RNA-seq 분석은 다음과 같은 흐름을 따른다:

1. FASTQ 파일 품질 확인 (FastQC)
2. 어댑터 트리밍 (Trim Galore)
3. 레퍼런스 게놈에 정렬 (STAR)
4. 발현량 정량 (featureCounts)
5. 차등 발현 분석 (DESeq2)

이 과정을 매번 수동으로 실행하면 시간이 오래 걸리고, 실수가 생기기 쉽다. **Snakemake는 이러한 분석 파이프라인을 자동화하는 워크플로우 관리 도구**이다.

Snakemake의 장점

- **재현성**: 같은 Snakefile로 언제든 동일한 분석을 반복할 수 있다
- **자동 의존성 관리**: 어떤 단계를 먼저 실행해야 하는지 자동으로 판단한다
- **병렬 실행**: 독립적인 단계는 동시에 실행하여 시간을 절약한다
- **부분 재실행**: 중간에 실패하면 실패한 단계부터 다시 시작한다

6.2 설치

```
pip install snakemake
```

> **참고**: 복잡한 생명정보학 파이프라인에서는 Conda와 함께 사용하는 것이 일반적이다. 각 rule에 독립적인 Conda 환경을 지정할 수 있다.

6.3 Snakemake 핵심 개념

Rule (규칙)

Snakemake의 기본 단위는 **rule**이다. 하나의 rule은 **입력 → 처리 → 출력**을 정의한다.

```
rule fastqc:
    input:
        "data/{sample}.fastq.gz"
    output:
        "results/fastqc/{sample}_fastqc.html"
    shell:
        "fastqc {input} -o results/fastqc/"
```

Wildcard (와일드카드)

`{sample}` 같은 와일드카드를 사용하면 **여러 샘플에 같은 규칙을 자동 적용**할 수 있다. 예를 들어 `sample_A.fastq.gz`, `sample_B.fastq.gz`, `sample_C.fastq.gz`가 있으면, 위 rule이 세 파일 모두에 자동으로 실행된다.

DAG (방향성 비순환 그래프)

Snakemake는 rule 간의 의존 관계를 **DAG**로 자동 구성한다. 출력 파일이 다른 rule의 입력 파일이면 자동으로 순서가 결정된다.

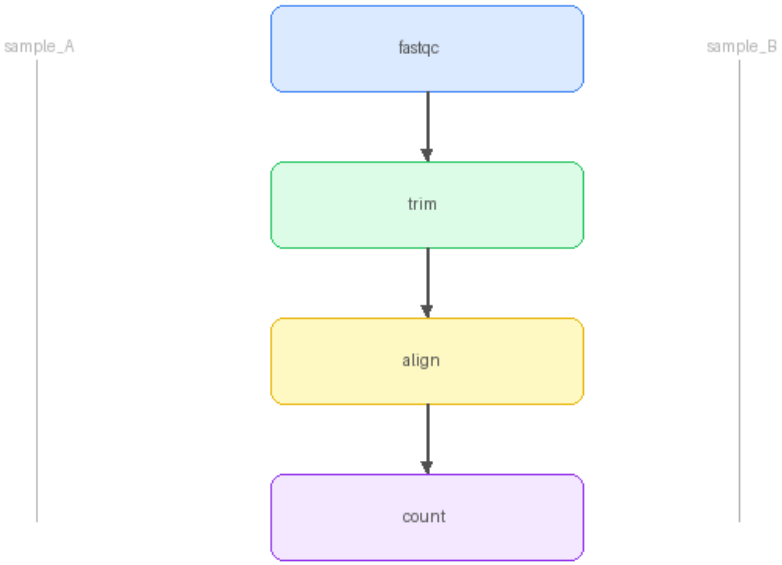

Figure 48: Snakemake DAG 시각화 예시

6.4 Snakefile 작성

기본 구조

Snakefile은 프로젝트 루트에 `Snakefile`이라는 이름으로 작성한다. `rule all`은 최종적으로 원하는 결과 파일을 지정하는 특수 rule이다.

```
#
SAMPLES = ["sample_A", "sample_B", "sample_C"]

#
rule all:
    input:
        expand("results/counts/{sample}.counts.txt", sample=SAMPLES)

# 1 :
rule fastqc:
    input:
        "data/{sample}.fastq.gz"
    output:
        "results/fastqc/{sample}_fastqc.html"
    shell:
        "fastqc {input} -o results/fastqc/"

# 2 :
rule trim:
    input:
        "data/{sample}.fastq.gz"
    output:
        "results/trimmed/{sample}_trimmed.fastq.gz"
    shell:
        "trim_galore {input} -o results/trimmed/"

# 3 :
rule align:
    input:
        fastq="results/trimmed/{sample}_trimmed.fastq.gz",
        index="ref/genome_index"
    output:
        "results/aligned/{sample}.bam"
    threads: 4
    shell:
        "STAR --runThreadN {threads} "
        "--genomeDir {input.index} "
        "--readFilesIn {input.fastq} "
        "--outSAMtype BAM SortedByCoordinate "
        "--outFileNamePrefix results/aligned/{wildcards.sample}"

# 4 :
rule count:
    input:
        bam="results/aligned/{sample}.bam",
        gtf="ref/genes.gtf"
    output:
        "results/counts/{sample}.counts.txt"
```

```
    shell:
        "featureCounts -a {input.gtf} -o {output} {input.bam}"
```

실행

```
#    (          )
snakemake -n

#   (4     )
snakemake --cores 4

# DAG
snakemake --dag | dot -Tpng > dag.png
```

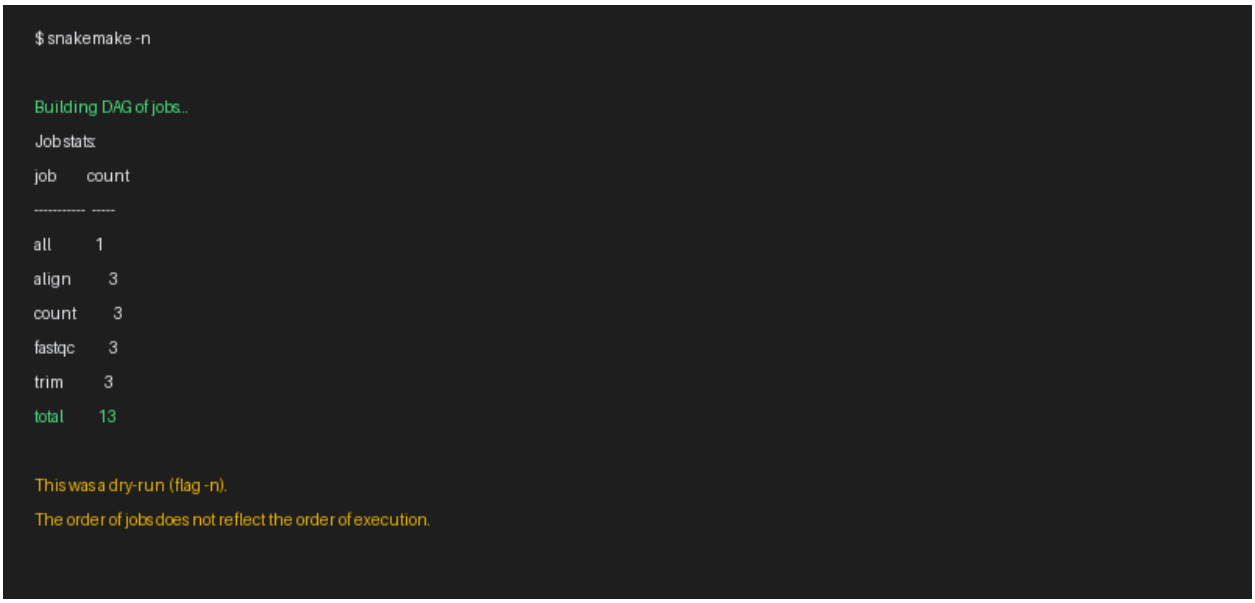

Figure 49: snakemake -n 드라이런 결과

6.5 주요 기능

Conda 환경 통합

각 rule에 독립적인 Conda 환경을 지정할 수 있다. 도구 간 의존성 충돌을 방지한다.

```
rule deseq2:
    input:
        counts=expand("results/counts/{sample}.counts.txt", sample=SAMPLES)
    output:
        "results/deseq2/results.csv"
    conda:
        "envs/deseq2.yaml"
    script:
        "scripts/deseq2.R"
```

```
# envs/deseq2.yaml
name: deseq2
channels:
```

```
  - conda-forge
  - bioconda
dependencies:
  - r-base=4.3
  - bioconductor-deseq2
```

Config 파일

분석 파라미터를 Snakefile과 분리하여 관리할 수 있다.

```
# config.yaml
samples:
  - sample_A
  - sample_B
  - sample_C

reference:
  genome: "ref/genome.fa"
  gtf: "ref/genes.gtf"

params:
  threads: 4
  min_quality: 20
```

```
# Snakefile  config
configfile: "config.yaml"

SAMPLES = config["samples"]

rule trim:
    input:
        "data/{sample}.fastq.gz"
    output:
        "results/trimmed/{sample}_trimmed.fastq.gz"
    params:
        quality=config["params"]["min_quality"]
    shell:
        "trim_galore -q {params.quality} {input} -o results/trimmed/"
```

로그 파일

각 rule의 실행 로그를 별도로 저장한다.

```
rule align:
    input:
        "results/trimmed/{sample}_trimmed.fastq.gz"
    output:
        "results/aligned/{sample}.bam"
    log:
        "logs/align/{sample}.log"
    shell:
        "STAR --readFilesIn {input} > {log} 2>&1"
```

6.6 바이브 코딩으로 Snakefile 작성하기

Snakemake의 문법을 완벽하게 익히는 것보다, **분석 파이프라인의 흐름을 이해하고 AI에게 정확히 설명하는 것**이 중요하다.

AI에게 요청하는 예시

> "RNA-seq 파이프라인을 Snakemake로 만들어줘. 입력은 data/ 폴더의 FASTQ 파일이고, FastQC → Trim Galore → STAR 정렬 → featureCounts 순서로 처리해줘. 샘플 목록은 config.yaml에서 읽어오게 하고, 각 단계마다 로그 파일을 남겨줘"

이 요청을 하려면 다음을 알아야 한다: - **분석 단계의 순서**: 왜 트리밍 다음에 정렬을 하는지 - **각 도구의 역할**: FastQC는 QC, STAR는 정렬, featureCounts는 정량 - **입출력 파일 형식**: FASTQ → BAM → counts

Snakefile 수정 요청 예시

> "align rule에서 STAR 대신 HISAT2를 사용하도록 바꿔줘"
>
> "count rule 다음에 DESeq2로 차등 발현 분석하는 rule 추가해줘. R 스크립트는 별도 파일로 분리하고, Conda 환경도 만들어줘"
>
> "sample_D가 추가됐으니까 config.yaml에 반영해줘"

디버깅 요청 예시

> "snakemake 실행하면 align 단계에서 에러가 나. logs/align/sample_A.log 보고 원인 알려줘"
>
> "드라이런 결과를 보여주고, DAG 그래프도 생성해줘"

6.7 실전: 단일세포 분석 파이프라인

Scanpy 분석도 Snakemake로 자동화할 수 있다:

```
SAMPLES = ["pbmc_10k", "pbmc_5k"]

rule all:
    input:
        expand("results/{sample}/umap.png", sample=SAMPLES)

rule qc:
    input:
        "data/{sample}.h5ad"
    output:
        "results/{sample}/qc_filtered.h5ad"
    script:
        "scripts/01_qc.py"

rule normalize:
    input:
        "results/{sample}/qc_filtered.h5ad"
    output:
        "results/{sample}/normalized.h5ad"
    script:
        "scripts/02_normalize.py"

rule cluster:
    input:
```

```
        "results/{sample}/normalized.h5ad"
    output:
        "results/{sample}/clustered.h5ad",
        "results/{sample}/umap.png"
    script:
        "scripts/03_cluster.py"
```

팁: AI에게 "이 Scanpy 분석을 Snakemake 파이프라인으로 변환해줘"라고 요청하면, 기존 분석 코드를 자동으로 rule 단위로 분리하고 Snakefile을 생성해준다.

6.8 정리

- **Snakemake**: 생명정보학 분석 파이프라인을 자동화하는 워크플로우 관리 도구
- **Rule**: 입력 → 처리 → 출력을 정의하는 기본 단위
- **Wildcard**: `{sample}` 같은 패턴으로 여러 샘플에 동일 규칙 적용
- **DAG**: rule 간 의존 관계를 자동으로 파악하여 실행 순서 결정
- **Config 파일**: 분석 파라미터를 Snakefile과 분리하여 관리
- **바이브 코딩의 핵심**: 분석 파이프라인의 **흐름과 각 도구의 역할**을 이해하고, AI에게 단계별로 지시하는 것

7장. 프로젝트 디렉토리 설정

7.1 이 책에서 사용하는 기술 스택

이 책에서는 다음과 같은 기술 스택을 사용하여 생명정보학 웹 도구를 개발한다.

기술	역할	설명
SvelteKit	프론트엔드 프레임워크	빠르고 가벼운 웹 앱 개발 프레임워크
Tailwind CSS	CSS 프레임워크	유틸리티 클래스 기반의 스타일링 도구
PostgreSQL	데이터베이스	오픈소스 관계형 데이터베이스
Docker	컨테이너	개발 환경 통합 관리

SvelteKit

SvelteKit은 Svelte를 기반으로 한 풀스택 웹 프레임워크이다. React, Vue 등 다른 프레임워크와 비교했을 때 다음과 같은 장점이 있다:

- **컴파일 타임 최적화**: 런타임에 가상 DOM을 사용하지 않아 매우 빠름
- **적은 코드량**: 동일한 기능을 더 적은 코드로 구현 가능
- **서버 사이드 렌더링(SSR)** 기본 지원
- **파일 기반 라우팅**: 파일 구조가 곧 URL 구조

Tailwind CSS

Tailwind CSS는 유틸리티 클래스 기반의 CSS 프레임워크이다. 별도의 CSS 파일을 작성하지 않고, HTML 요소에 직접 클래스를 적용하여 스타일링한다.

```
<!--    CSS    -->
<div class="card"> </div>

<!-- Tailwind CSS    -->
<div class="bg-white rounded-lg shadow-md p-6"> </div>
```

AI 에이전트와 함께 사용하기에 특히 적합한데, 스타일이 HTML과 같은 파일에 있어 컨텍스트를 파악하기 쉽기 때문이다.

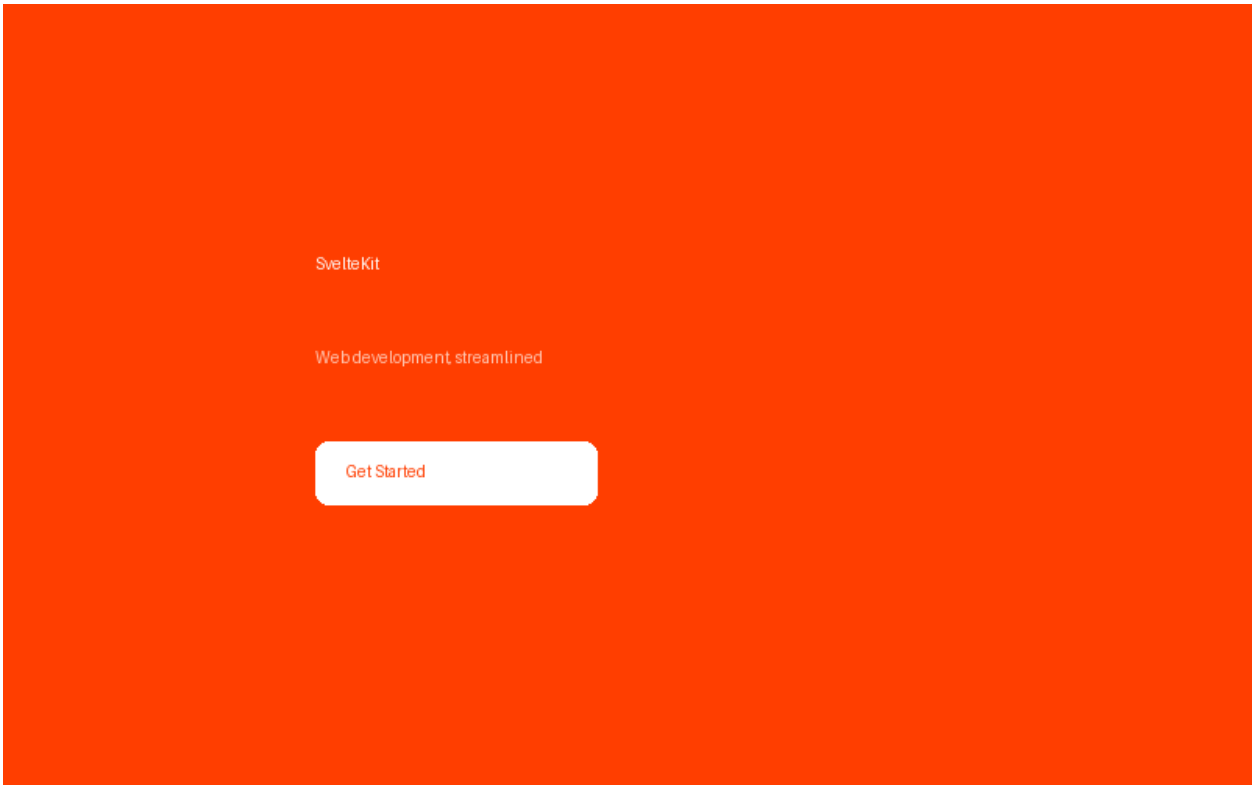

Figure 50: SvelteKit 공식 웹사이트

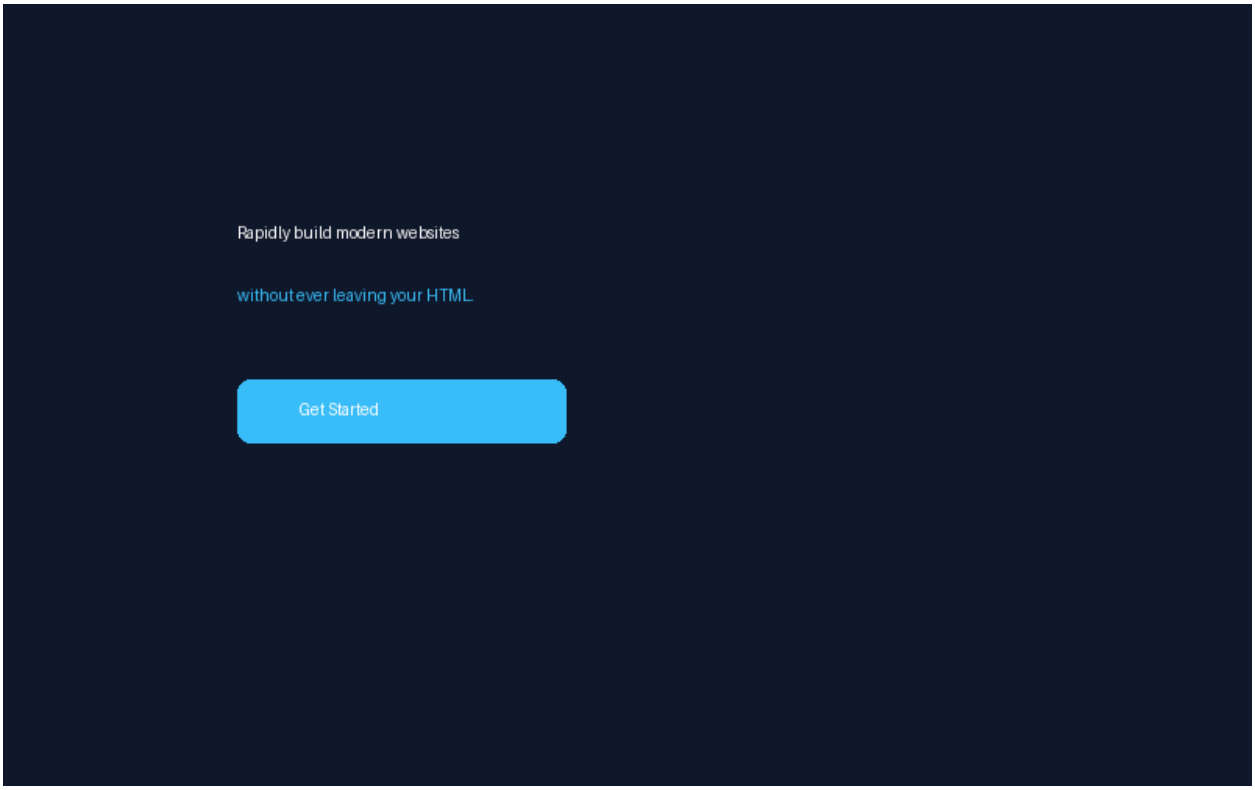

Figure 51: Tailwind CSS 공식 웹사이트

PostgreSQL

PostgreSQL은 세계에서 가장 많이 사용되는 오픈소스 관계형 데이터베이스이다. 사용자 데이터, 분석 결과 등을 저장하고 조회하는 데 사용한다. 이 책에서는 Docker를 통해 PostgreSQL을 실행한다.

7.2 프로젝트 생성

Node.js 설치

SvelteKit을 사용하려면 Node.js가 필요하다. 다음 웹사이트에서 설치한다:

https://nodejs.org/en/download

설치 시 **nvm**(Node Version Manager)과 **pnpm**(패키지 매니저)을 선택한 후 터미널에 표시되는 명령을 입력한다.

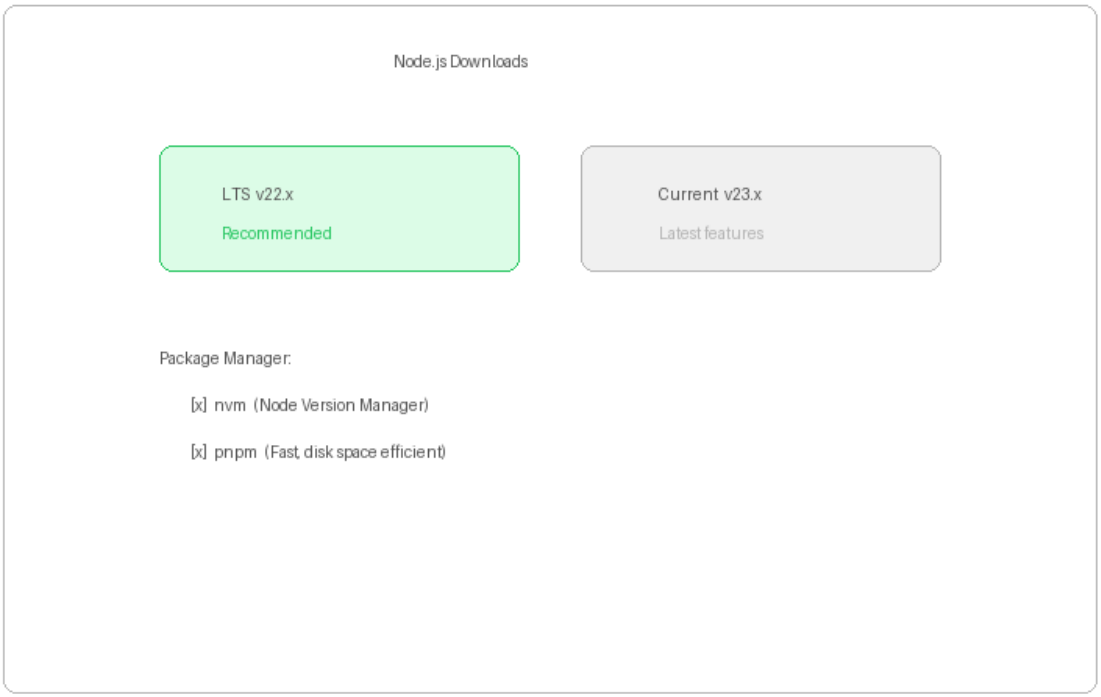

Figure 52: Node.js 다운로드 페이지

설치 확인:

```
node --version
pnpm --version
```

SvelteKit 프로젝트 초기화

터미널에서 다음 명령을 실행하여 SvelteKit 프로젝트를 생성한다:

```
pnpm create svelte@latest my-bioinfo-app
cd my-bioinfo-app
pnpm install
```

프로젝트 생성 시 다음 옵션을 선택한다:

- Template: **Skeleton project**
- Type checking: **Yes, using TypeScript syntax**
- Additional options: 방향키와 스페이스바로 선택 가능. **Tailwind CSS**, **Typography**, **Forms**를 선택할 것

  참고: 프로젝트 초기 생성은 AI 에이전트에 맡기기보다 직접 수행하는 것이 좋다. AI는 초기화 명령을 사용하기보다 기존 코드를 직접 작성하려는 경향이 있어, 최신 버전이 아닌 코드를 생성할 수 있다.

Tailwind CSS

Tailwind CSS는 SvelteKit 프로젝트 생성 시 옵션으로 함께 설치할 수 있다. 별도로 수동 설치할 필요 없이, 프로젝트 초기화 과정에서 Tailwind CSS를 선택하면 자동으로 설정된다.

7.3 Docker 환경 구성

compose.yml 작성

프로젝트 루트에 `compose.yml` 파일을 생성한다. 이 파일은 SvelteKit 개발 서버와 PostgreSQL 데이터베이스를 함께 관리한다.

```yaml
services:
  app:
    build: .
    ports:
      - "5173:5173"
    volumes:
      - .:/app
      - /app/node_modules
    environment:
      - DATABASE_URL=postgresql://postgres:postgres@db:5432/bioinfo
    depends_on:
      - db

  db:
    image: postgres:16-alpine
    ports:
      - "5432:5432"
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
      POSTGRES_DB: bioinfo
    volumes:
      - pgdata:/var/lib/postgresql/data

volumes:
  pgdata:
```

Dockerfile 작성

프로젝트 루트에 `Dockerfile`을 생성한다:

```
FROM node:20-alpine

WORKDIR /app

# pnpm
RUN corepack enable && corepack prepare pnpm@latest --activate
```

```
#
COPY package.json pnpm-lock.yaml ./
RUN pnpm install

#
COPY . .

#
CMD ["pnpm", "dev", "--host"]
```

7.4 환경 변수 (.env)

프로젝트에서 데이터베이스 접속 정보, API 키 등 민감한 설정값은 **환경 변수**로 관리한다. 프로젝트 루트에 `.env` 파일을 생성한다:

```
DATABASE_URL=postgresql://postgres:postgres@localhost:5432/bioinfo
PUBLIC_SITE_NAME=My Bioinfo App
```

SvelteKit에서 환경 변수는 두 가지로 나뉜다:

접두사	접근 범위	용도
`PUBLIC_`	서버 + 클라이언트	사이트 이름 등 공개 정보
(없음)	서버만	데이터베이스 비밀번호, API 키 등 민감 정보

> **주의**: `.env` 파일은 `.gitignore`에 반드시 추가하여 Git에 커밋되지 않도록 한다. 대신 `.env.example` 파일을 만들어 어떤 환경 변수가 필요한지 안내한다.

```
# .gitignore
echo ".env" >> .gitignore
```

7.5 프로젝트 디렉토리 구조

최종적으로 프로젝트 디렉토리는 다음과 같은 구조를 가진다:

```
my-bioinfo-app/
   compose.yml      # Docker
   Dockerfile              # Docker
   .env                    #     (Git      )
   .env.example            #
   .gitignore
   package.json
   pnpm-lock.yaml
   svelte.config.js        # SvelteKit
   vite.config.ts          # Vite
   tailwind.config.ts      # Tailwind CSS
   CLAUDE.md               # AI
   README.md
   src/
       app.css             #      (Tailwind   )
       app.html            # HTML
       lib/                #
           components/     #
           server/         #       (DB    )
       routes/             #
```

```
        +layout.svelte  #
        +page.svelte    #
        tools/
            +page.svelte  #
static/                 #     (  ,    )
```

CLAUDE.md 작성

CLAUDE.md는 AI 에이전트가 참조하는 프로젝트 명세 파일이다. Claude Code는 작업을 시작할 때 이 파일을 가장 먼저 읽고, 프로젝트의 구조와 규칙을 파악한다.

핵심 원칙: CLAUDE.md에 정보를 많이 넣을수록 AI는 더 똑똑해진다.

AI 코딩 에이전트는 사람과 달리 프로젝트의 맥락을 스스로 알지 못한다. "Navbar에 로고를 추가해줘"라고 요청했을 때, AI가 올바른 위치에 올바른 방식으로 코드를 작성하려면 다음을 알아야 한다:

- 이 프로젝트가 SvelteKit을 사용하는지, React를 사용하는지
- Navbar 컴포넌트가 어디에 있는지
- 스타일링에 Tailwind CSS를 쓰는지, 일반 CSS를 쓰는지
- 로고 이미지는 어디에 저장되어 있는지

이러한 정보가 CLAUDE.md에 없으면 AI는 매번 프로젝트 구조를 탐색하고 추측해야 한다. 반면, 이 정보가 잘 정리되어 있으면 AI는 곧바로 정확한 코드를 작성할 수 있다.

왜 배경지식이 필요한가? 바이브 코딩은 "AI가 다 해주니까 개발을 몰라도 된다"는 뜻이 **아니다**. 오히려 그 반대이다. CLAUDE.md를 잘 작성하려면 **사람이 먼저 프로젝트를 이해하고 있어야 한다**.

예를 들어 이런 CLAUDE.md를 작성한다고 하자:

```
#



#
- SvelteKit (    +  )
- Tailwind CSS (  )
- PostgreSQL (    )

#
- src/routes/ :       (      )
- src/lib/components/ :       UI
- src/lib/server/ :        (DB    )

#
-       PascalCase ( : NavBar.svelte)
-   API  +server.ts
-      $env/static/private   $env/static/public

#
-  : blue-600  primary color
-   :     max-w-7xl,
-   : mobile-first
```

이 파일을 작성하려면 SvelteKit의 파일 기반 라우팅이 무엇인지, Tailwind의 유틸리티 클래스가 어떻게 동작하는지, 컴포넌트와 레이아웃의 개념이 무엇인지 알아야 한다. **이 책에서 배우는 웹 개발 배경지식이 바로 이를 위한 것이다.**

> **핵심**: 바이브 코딩에서 사람의 역할은 코드를 직접 작성하는 것이 아니라, **AI가 올바른 코드를 작성할 수 있도록 정확한 지시를 내리는 것**이다. 정확한 지시를 내리려면 기본적인 개발 개념을 이해하고 있어야 한다.

CLAUDE.md에 포함할 내용

항목	설명	예시
프로젝트 개요	프로젝트의 목적과 대상 사용자	"생명정보학 연구자를 위한 웹 기반 시퀀스 분석 도구"
기술 스택	사용하는 프레임워크, 라이브러리	"SvelteKit, Tailwind CSS, PostgreSQL"
디렉토리 구조	주요 폴더의 역할	"src/routes/는 페이지, src/lib/은 공유 코드"
코딩 컨벤션	파일명 규칙, 코드 스타일	"컴포넌트는 PascalCase, 변수는 camelCase"
디자인 가이드라인	색상, 폰트, 레이아웃 규칙	"primary color는 blue-600, 최대 너비 max-w-7xl"
비즈니스 로직	도메인 특화 규칙	"FASTA 형식은 〉로 시작하는 헤더 + 시퀀스"

> **팁**: CLAUDE.md는 한 번 작성하고 끝이 아니다. 프로젝트가 발전할수록 새로운 규칙과 패턴을 추가하여 AI 에이전트가 항상 최신 상태를 파악할 수 있도록 유지한다.

7.6 개발 서버 실행

Docker를 사용하는 경우

```
docker compose up
```

브라우저에서 `http://localhost:5173`으로 접속하면 SvelteKit 앱을 확인할 수 있다.

Docker 없이 로컬에서 실행하는 경우

```
pnpm dev
```

7.7 정리

- **SvelteKit + Tailwind CSS + PostgreSQL이 이 책의 기본 기술 스택**
  - SvelteKit: 빠르고 가벼운 풀스택 프레임워크
  - Tailwind CSS: AI와 함께 사용하기 좋은 유틸리티 CSS
  - PostgreSQL: 오픈소스 관계형 데이터베이스
- **Docker Compose로 개발 환경을 통합 관리**
  - 웹 앱과 데이터베이스를 한 번에 실행
- **환경 변수(.env)로 민감한 정보를 분리 관리**
  - `.gitignore`에 반드시 추가
- **CLAUDE.md에 프로젝트 명세를 작성하여 AI 에이전트 활용도를 높이기**

8장. 랜딩 페이지 디자인

8.1 랜딩 페이지란?

랜딩 페이지(Landing Page)는 웹 사이트의 **가장 첫 페이지**이다. 방문자가 URL에 접속했을 때 가장 먼저 보게 되는 화면으로, 웹 사이트의 첫인상을 결정한다. 따라서 방문자의 이목을 끌 수 있도록 화려하고 매력적인 디자인이 중요하다.

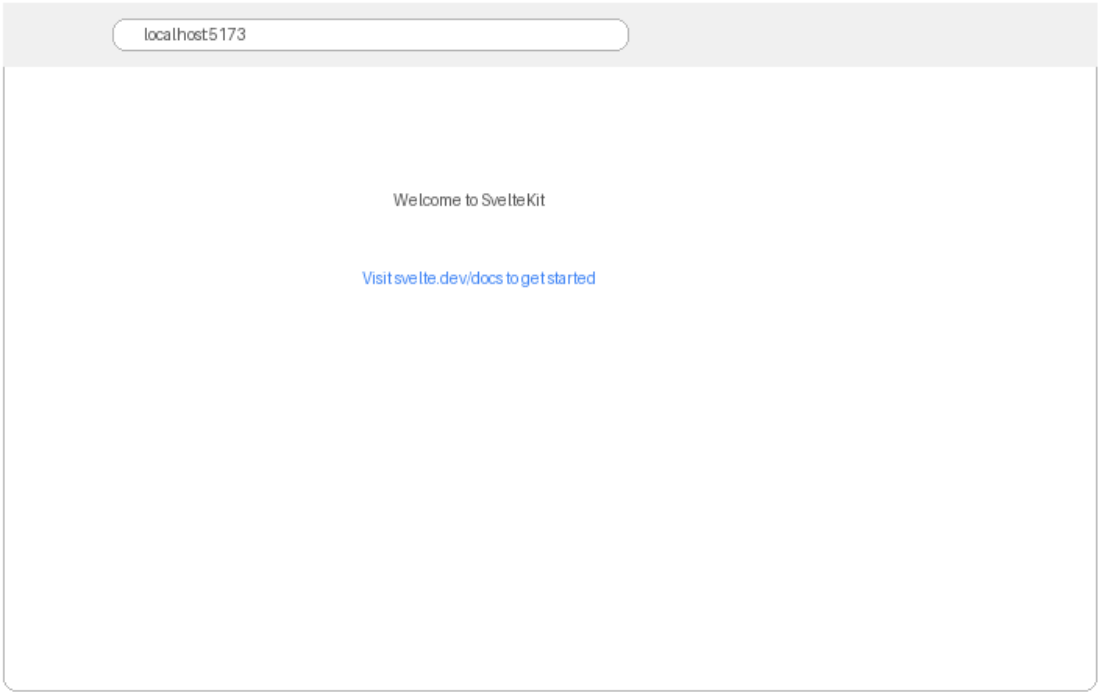

Figure 53: SvelteKit 개발 서버 실행 화면

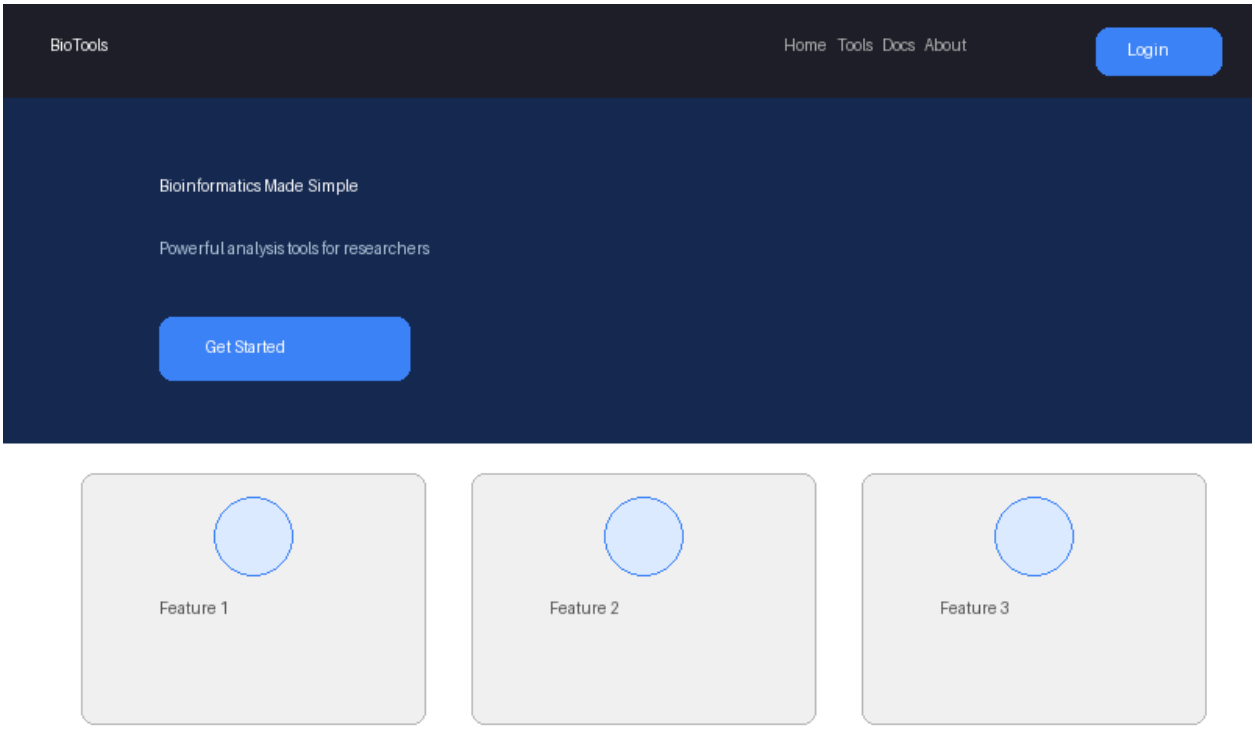

Figure 54: 잘 디자인된 랜딩 페이지 예시

8.2 랜딩 페이지의 구조

Header, Body, Footer

웹 페이지는 크게 세 영역으로 구성된다:

Figure 55: 웹 페이지 구조

영역	포함 요소
Header	로고, 메인 메뉴(Navigation Bar), Hero 섹션
Body	주 콘텐츠 (기능 소개, 특징, 사용 방법 등)
Footer	로고(주로 흑백), 하단 메뉴, 연락처, 주소, 법적 정보, 저작권 정보 등

실제 구현 시 각각을 별도의 컴포넌트 파일로 나누어서 구현한다.

Navigation Bar (Navbar)

Navigation Bar는 웹 사이트의 상단에 위치하는 메뉴이다. 로고와 주요 페이지 링크를 포함하며, 사용자가 사이트 내를 이동할 수 있게 한다.

Hero Section

Hero Section은 랜딩 페이지의 가장 **눈에 띄는 첫 번째 영역**이다. 화면의 좌우 전체 너비를 사용하며, 다음 요소들을 포함한다:

- **헤드라인**: 사이트의 핵심 가치를 한 문장으로 전달
- **서브 헤드라인**: 헤드라인을 보충하는 설명
- **CTA(Call to Action) 버튼**: 사용자가 취해야 할 행동 (예: “시작하기”, “도구 사용하기”)

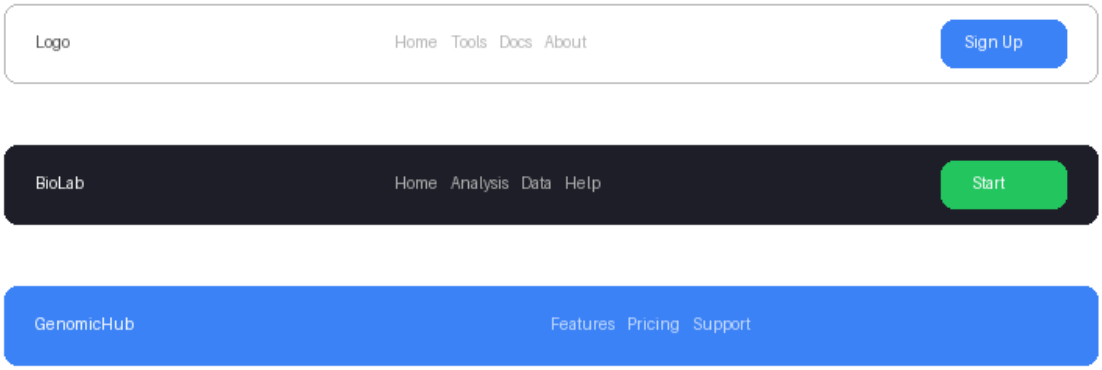

Figure 56: Navbar 디자인 예시

- **배경 이미지 또는 동영상**: 시각적 임팩트

  참고: Hero Section 디자인 패턴에 대한 자세한 내용은 https://www.awebco.com/blog/hero-section/ 에서 확인할 수 있다.

Carousel

Carousel(캐러셀)은 여러 이미지나 콘텐츠를 **슬라이드 형태**로 보여주는 요소이다. 여러 기능이나 특징을 순차적으로 소개할 때 유용하다.

Features Section

웹 도구의 주요 기능을 소개하는 영역이다. 보통 **카드(Card)** 형태로 3~4개의 핵심 기능을 나열한다.

8.3 UI 컴포넌트

자주 사용되는 웹 구성 요소를 패턴화한 것을 **UI 컴포넌트**라 한다. 2011년 등장한 트위터 부트스트랩 프레임워크(https://getbootstrap.com) 유래된 것들이 많다.

랜딩 페이지에서 자주 사용되는 컴포넌트:

컴포넌트	설명
Navbar	상단 네비게이션 바
Hero	첫 화면의 대형 배너 영역
Card	카드형 콘텐츠 블록
Carousel	슬라이드 형태의 콘텐츠
Button	클릭 가능한 버튼 (Primary, Secondary, Ghost 등)
Badge	작은 라벨/태그
Footer	하단 정보 영역

8.4 AI를 활용한 디자인 목업 생성

나노바나나(Nanobanana)를 활용한 디자인

Google Gemini 등의 AI를 통해 디자인 목업을 생성할 수 있다. 앞서 배운 Header, Body, Footer 및 컴포넌트 개념을 활용하여 프롬프트를 구체적으로 작성하면 더 정확한 결과를 얻을 수 있다.

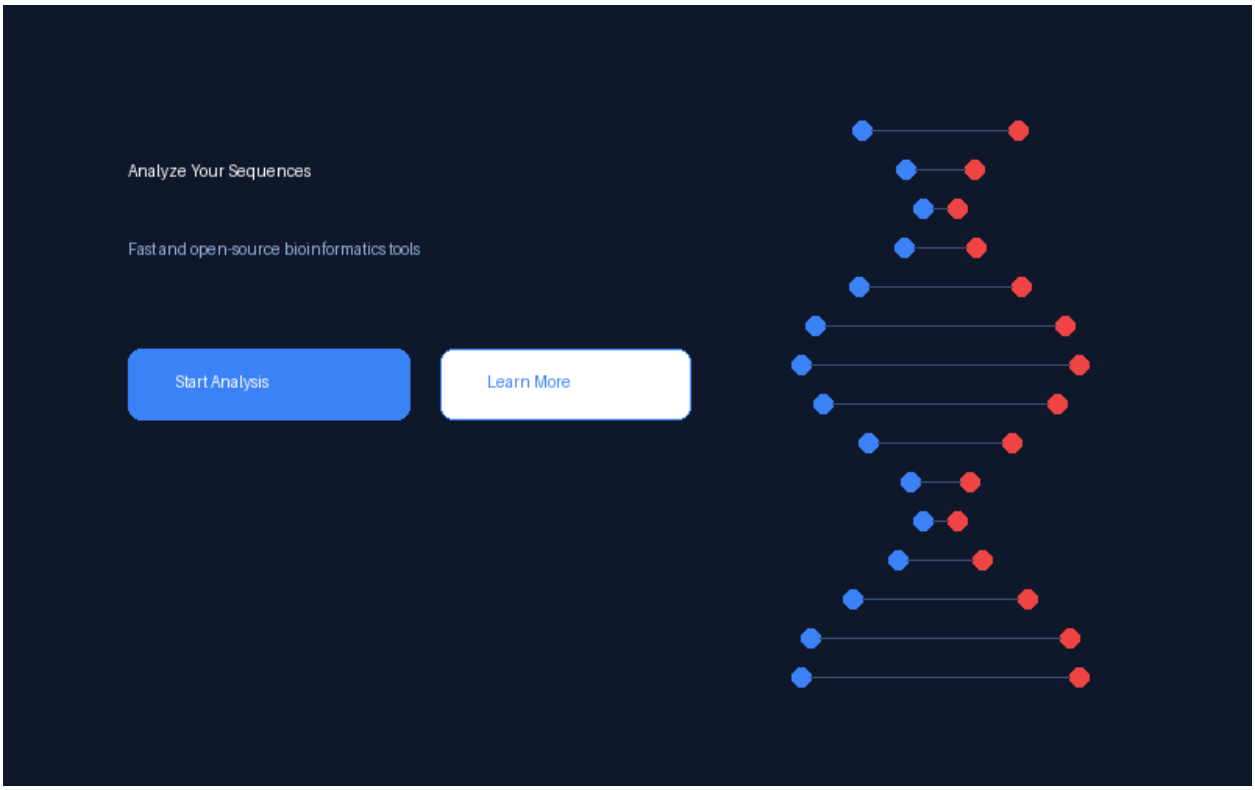

Figure 57: Hero Section 예시

Figure 58: Carousel 컴포넌트 예시

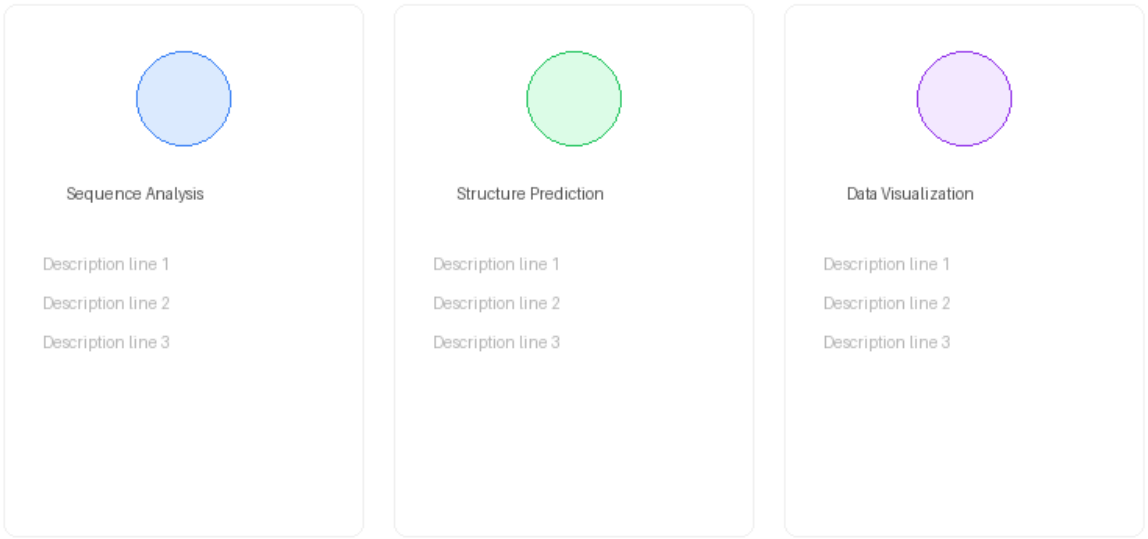

Figure 59: Features Section 예시

기본 프롬프트 예시:

```
                .
     Navbar,   Hero Section  DNA
"Sequence Analysis Made Simple"    , "   "     ,
   3           .
```

여러 번 생성해보고 마음에 드는 것을 선택한다.

더 구체적인 프롬프트 예시:

```
Header:     ,    Home/Tools/About/Contact      Navbar.
Hero Section:       ,       "Bioinformatics Tools for Everyone",
   "AI-powered sequence analysis", CTA   "Get Started".
   DNA      .
Features: 3  Card - Reverse Complement, Sequence Alignment, BLAST Search.
Footer:  ( ),     ,     .
```

Claude Code의 design 스킬

Claude Code에는 **design 스킬**이 내장되어 있어, 디자인 목업 이미지를 기반으로 실제 코드를 생성할 수 있다. 사용 방법은 다음과 같다:

1. AI로 생성한 디자인 목업 이미지를 프로젝트 폴더에 저장
2. Claude Code에서 디자인 이미지를 참조하며 구현을 요청

```
>                SvelteKit + Tailwind CSS    .
> (          )
```

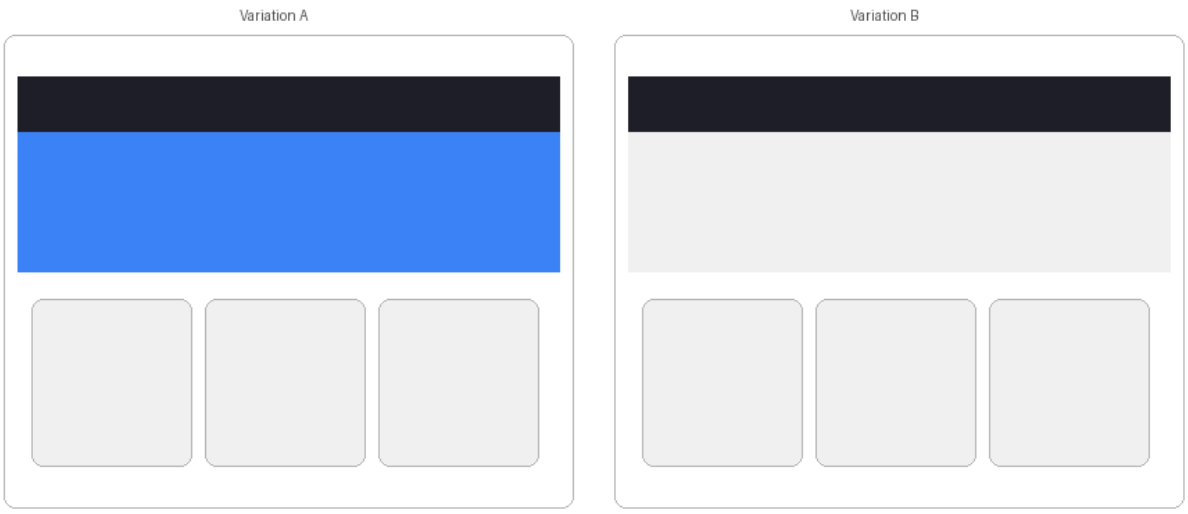

Figure 60: Gemini 디자인 목업

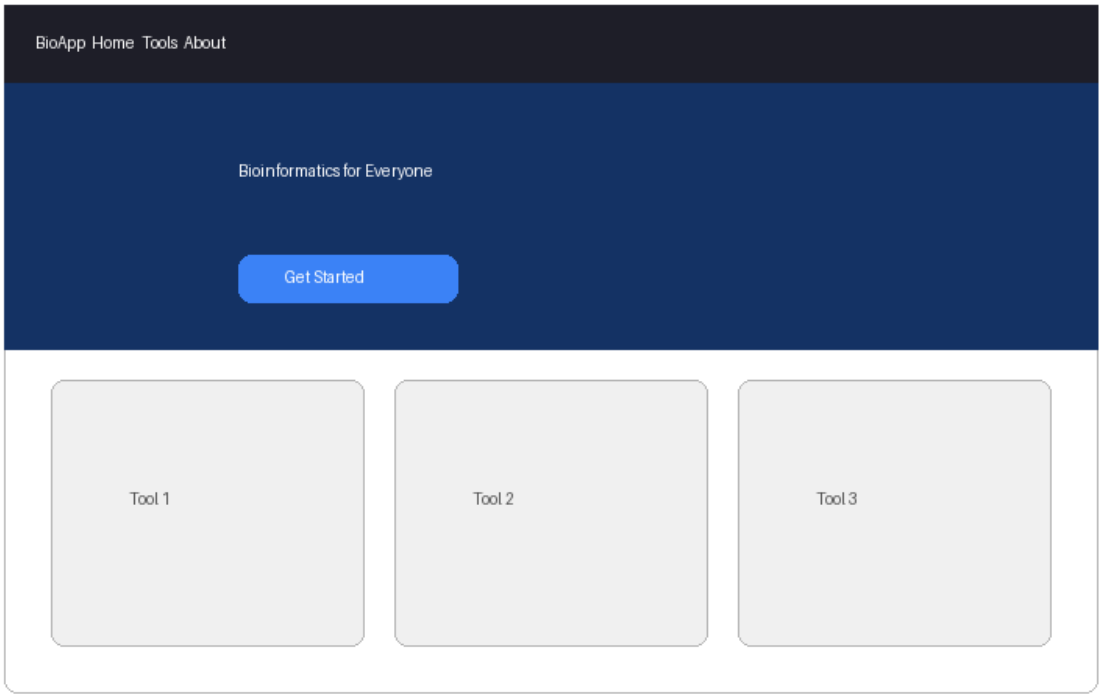

Figure 61: 상세 디자인 목업

Claude Code는 이미지를 분석하여 레이아웃, 색상, 컴포넌트 구조를 파악하고 코드를 생성한다.

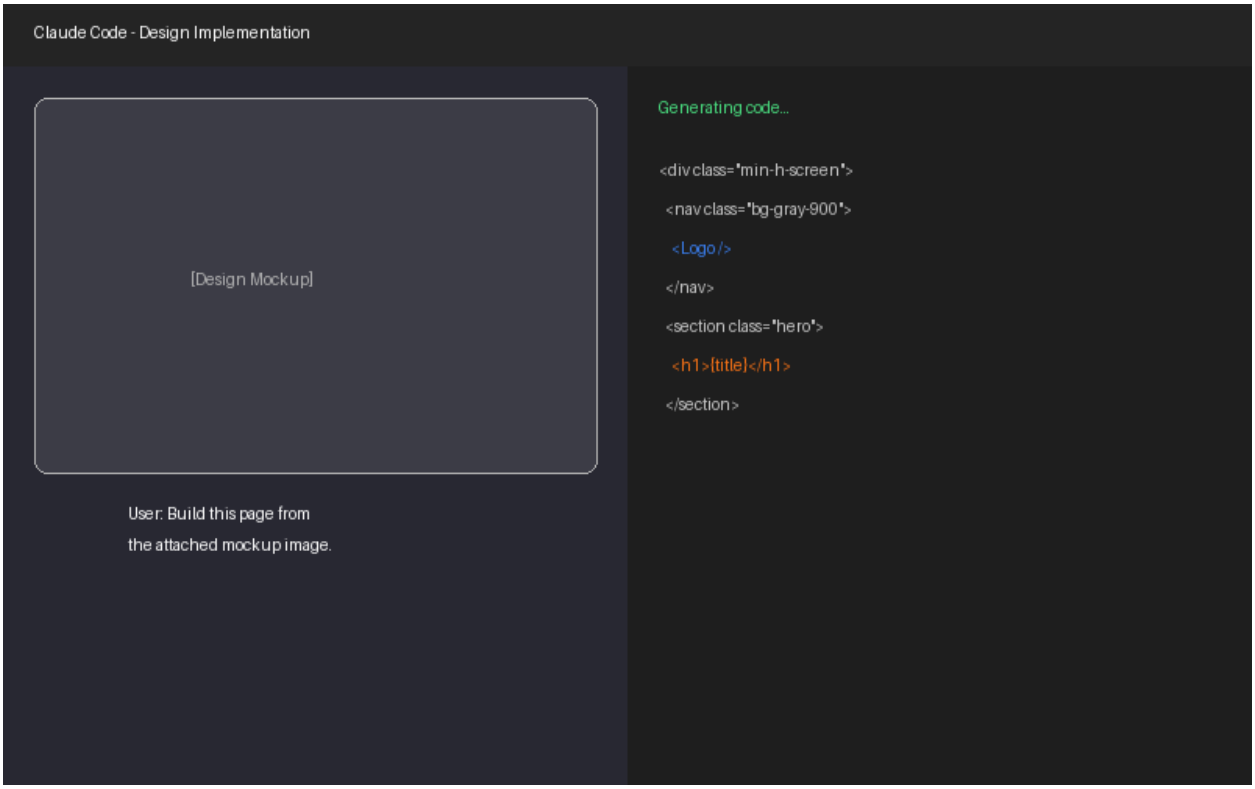

Figure 62: Claude 디자인 요청

8.5 SvelteKit에서 랜딩 페이지 구현

레이아웃 구성

SvelteKit에서는 `src/routes/+layout.svelte` 파일이 모든 페이지에 공통으로 적용되는 레이아웃을 정의한다. Header와 Footer를 여기에 배치한다.

```
<!-- src/routes/+layout.svelte -->
<script>
  import Header from '$lib/components/Header.svelte';
  import Footer from '$lib/components/Footer.svelte';
  import '../app.css';
</script>

<Header />
<main>
  <slot />
</main>
<Footer />
```

컴포넌트 분리

각 UI 요소를 별도의 Svelte 컴포넌트로 분리하여 `src/lib/components/` 디렉토리에 저장한다:

```
src/lib/components/
```

```
  Header.svelte       # Navbar
  Footer.svelte       #
  Hero.svelte         # Hero
  FeatureCard.svelte  #
  Carousel.svelte     #    (   )
```

랜딩 페이지 조립

src/routes/+page.svelte에서 컴포넌트들을 조립하여 랜딩 페이지를 완성한다:

```
<!-- src/routes/+page.svelte -->
<script>
  import Hero from '$lib/components/Hero.svelte';
  import FeatureCard from '$lib/components/FeatureCard.svelte';
</script>

<Hero />

<section class="py-16 px-4 max-w-6xl mx-auto">
  <h2 class="text-3xl font-bold text-center mb-12">   </h2>
  <div class="grid grid-cols-1 md:grid-cols-3 gap-8">
    <FeatureCard
      title="Reverse Complement"
      description="DNA            ."
    />
    <FeatureCard
      title="Sequence Alignment"
      description="            ."
    />
    <FeatureCard
      title="BLAST Search"
      description="             ."
    />
  </div>
</section>
```

8.6 정리

- **랜딩 페이지는 웹 사이트의 첫인상을 결정하는 중요한 페이지**
  - Header(Navbar) + Hero Section + Features + Footer 구조
- **UI 컴포넌트의 명칭과 역할을 이해하면 AI에게 더 정확한 지시가 가능**
  - Navbar, Hero, Card, Carousel, Button, Badge, Footer
- **나노바나나로 디자인 목업을 생성하고, Claude Code로 구현**
  - 프롬프트에 컴포넌트 명칭을 사용하여 구체적으로 요청
  - Claude Code의 design 스킬로 이미지 기반 코드 생성 가능
- **SvelteKit에서는 컴포넌트를 분리하여 레이아웃에 조립하는 방식으로 구현**

9장. 일반 페이지 디자인

9.1 일반 페이지란?

일반 페이지는 랜딩 페이지를 제외한 **나머지 모든 페이지**를 의미한다. 실제 웹 도구의 기능을 제공하는 핵심 페이지들이다. 랜딩 페이지의 화려한 Hero Section, Carousel 등의 디자인 요소를 축소하고, **모든 페이지에 공통적인 일관된 디자인**을 적용한다.

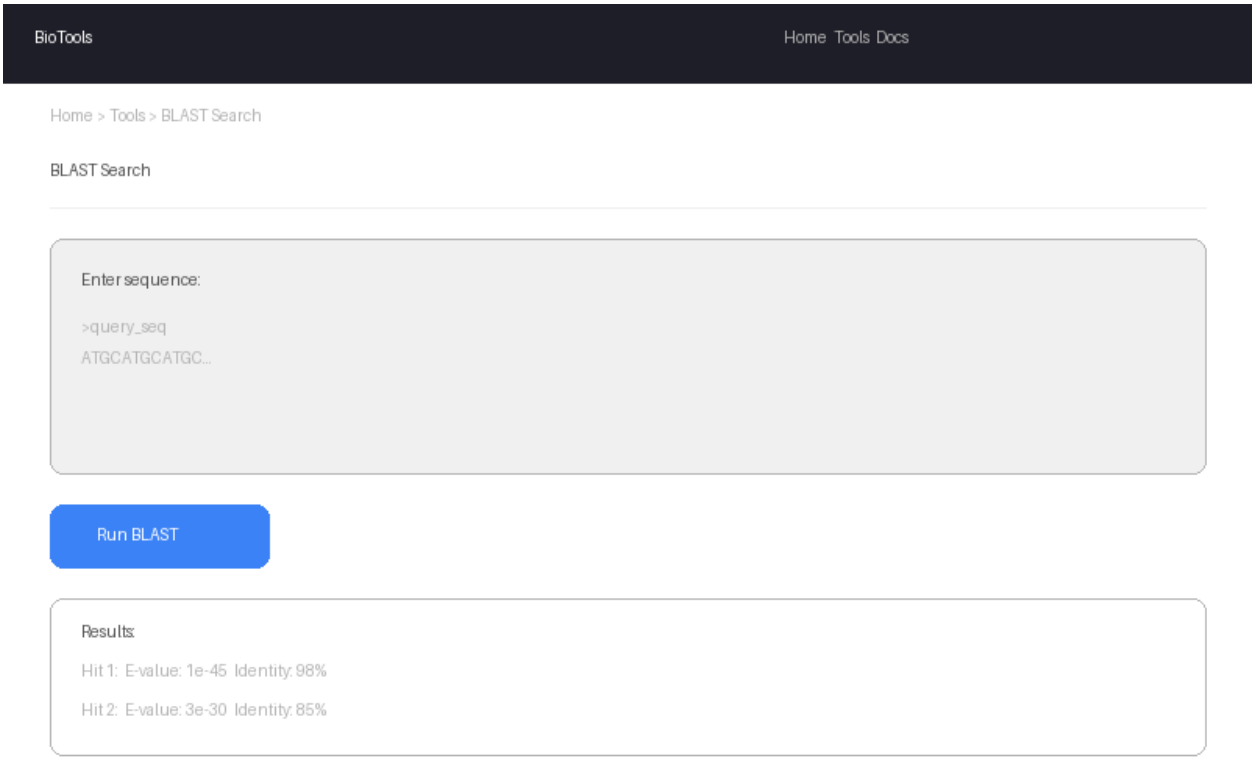

Figure 63: 일반 페이지 예시

9.2 일반 페이지의 구조

공통 레이아웃

일반 페이지는 랜딩 페이지와 동일한 Header(Navbar)와 Footer를 공유하되, Hero Section의 높이를 크게 줄여 간결한 형태로 사용한다. 랜딩 페이지의 Hero가 화면 좌우 전체 너비에 큰 높이를 차지하는 것과 달리, 일반 페이지의 Hero는 페이지 제목 정도만 표시하는 얇은 배너 형태이다.

영역	랜딩 페이지	일반 페이지
Header	Navbar + 큰 Hero Section	Navbar + 축소된 Hero (페이지 타이틀 배너)
Body	기능 소개, 특징 카드 등	실제 도구 인터페이스, 콘텐츠
Footer	동일	동일

Page Header (Breadcrumb 포함)

일반 페이지의 상단에는 현재 위치를 알려주는 **Breadcrumb**과 **페이지 제목(Heading)**을 배치한다.

```
Home > Tools > Reverse Complement
```

Sidebar

도구가 여러 개이거나 설정 옵션이 많은 경우, **Sidebar**를 활용하여 네비게이션을 제공한다.

9.3 도구 페이지에서 자주 사용되는 컴포넌트

생명정보학 웹 도구의 일반 페이지에서 자주 사용되는 UI 컴포넌트들이다.

입력 폼 컴포넌트

컴포넌트	용도	예시
Textarea	긴 텍스트 입력 (시퀀스 등)	FASTA 시퀀스 입력
Input	단일 값 입력	파라미터 값, 검색어

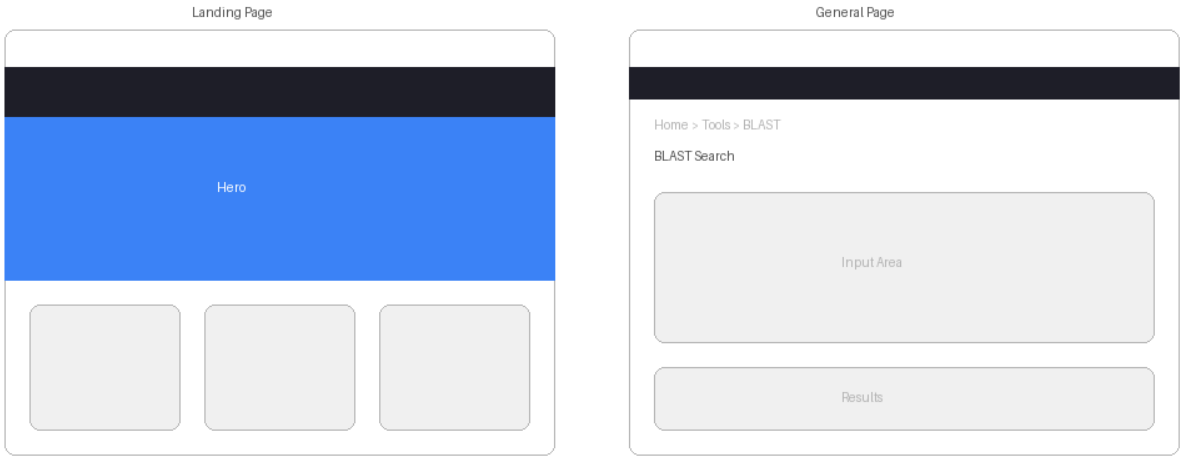

Figure 64: 레이아웃 비교

Figure 65: 페이지 헤더와 Breadcrumb

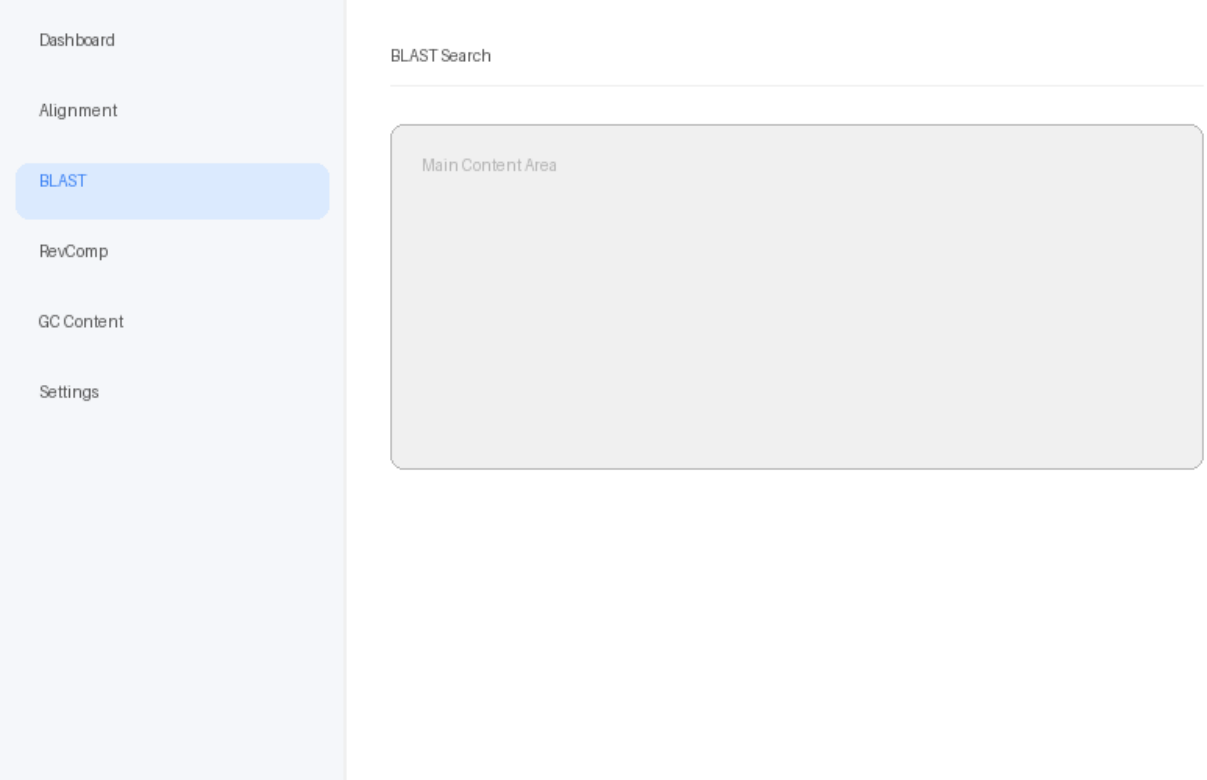

Figure 66: 사이드바 레이아웃

시각화 컴포넌트

컴포넌트	용도
Chart	데이터 시각화 (막대, 선, 파이 차트 등)
Heatmap	유전자 발현 히트맵 등
Sequence Viewer	시퀀스 정렬 결과 시각화

9.4 일반 페이지 디자인 패턴

패턴 1: 단일 도구 페이지

하나의 도구만 제공하는 간단한 페이지이다. 입력 → 실행 → 결과의 흐름을 하나의 페이지에서 처리한다.

```
Navbar

Breadcrumb
     (Heading)



   Textarea (    )

[   ]
```

```
        (Table/Code)

  [  ]

  Footer
```

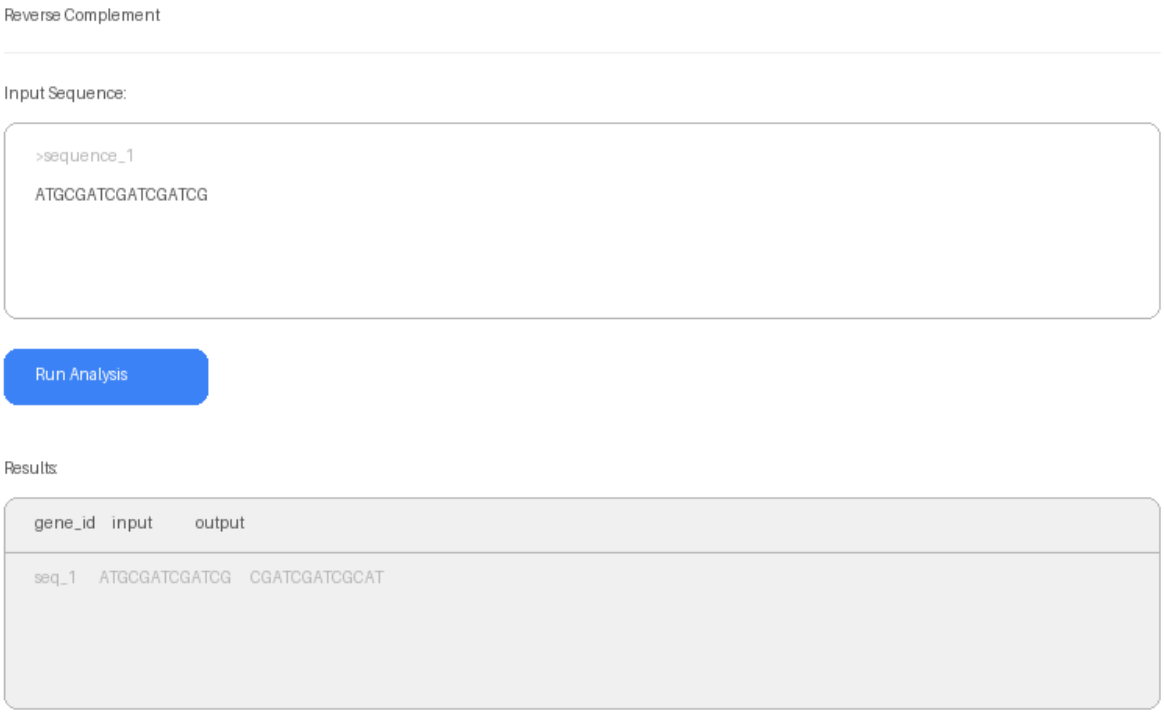

Figure 67: 단일 도구 페이지 예시

패턴 2: 다중 도구 페이지 (Sidebar 활용)

여러 도구를 제공하는 경우, Sidebar로 도구를 선택하고 우측에서 사용하는 구조이다.

```
  Navbar

         Breadcrumb
 Side
 bar

  1
  2      /
  3


  Footer
```

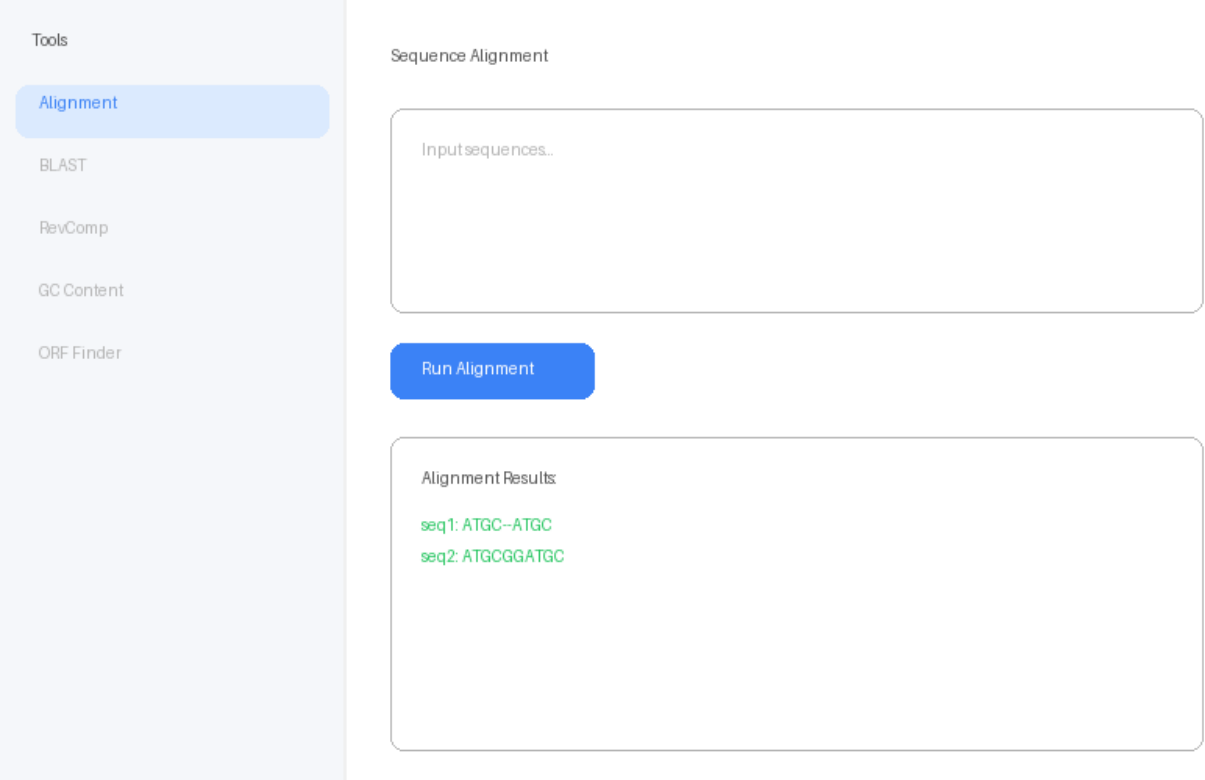

Figure 68: 다중 도구 사이드바

패턴 3: 탭 기반 결과 표시

분석 결과를 여러 형태로 보여줘야 할 때, Tab 컴포넌트를 활용한다.

9.5 SvelteKit에서 일반 페이지 구현

파일 기반 라우팅

SvelteKit은 **파일 기반 라우팅**을 사용한다. `src/routes/` 디렉토리의 폴더 구조가 URL 구조가 된다.

```
src/routes/
  +page.svelte              → /          (    )
  +layout.svelte            →
  about/
    +page.svelte            → /about
  tools/
    +page.svelte             → /tools    (    )
    +layout.svelte           → tools
    revcomp/
      +page.svelte           → /tools/revcomp
    alignment/
      +page.svelte           → /tools/alignment
```

도구 페이지 예시: Reverse Complement

```
<!-- src/routes/tools/revcomp/+page.svelte -->
<script>
```

Figure 69: 탭 기반 결과 표시

```
  let inputSequence = '';
  let result = '';

  function reverseComplement(seq) {
    const complement = { A: 'T', T: 'A', G: 'C', C: 'G' };
    return seq
      .toUpperCase()
      .split('')
      .reverse()
      .map(base => complement[base] || base)
      .join('');
  }

  function handleSubmit() {
    // FASTA
    const lines = inputSequence.split('\\n');
    const seq = lines.filter(l => !l.startsWith('>')).join('');
    result = reverseComplement(seq);
  }
</script>

<div class="max-w-4xl mx-auto p-6">
  <nav class="text-sm text-gray-500 mb-4">
    <a href="/" class="hover:underline">Home</a> &gt;
    <a href="/tools" class="hover:underline">Tools</a> &gt;
    <span>Reverse Complement</span>
  </nav>

  <h1 class="text-3xl font-bold mb-6">Reverse Complement</h1>

  <div class="space-y-4">
```

```
    <label class="block">
      <span class="text-lg font-medium">Input Sequence (FASTA)</span>
      <textarea
        bind:value={inputSequence}
        class="mt-2 w-full h-40 p-3 border rounded-lg font-mono"
        placeholder=">sequence1&#10;ATCGATCG"
      ></textarea>
    </label>

    <button
      on:click={handleSubmit}
      class="bg-blue-600 text-white px-6 py-2 rounded-lg hover:bg-blue-700"
    >

    </button>

    {#if result}
      <div class="mt-6">
        <h2 class="text-xl font-semibold mb-2"> </h2>
        <pre class="bg-gray-100 p-4 rounded-lg font-mono overflow-x-auto">{result}</pre>
        <button
          on:click={() => navigator.clipboard.writeText(result)}
          class="mt-2 text-blue-600 hover:underline"
        >

        </button>
      </div>
    {/if}
  </div>
</div>
```

9.6 AI를 활용한 일반 페이지 디자인

일반 페이지도 랜딩 페이지와 마찬가지로 AI를 활용하여 디자인할 수 있다. 프롬프트에 컴포넌트 명칭을 정확히 사용하면 더 좋은 결과를 얻을 수 있다.

프롬프트 예시:

```
Reverse Complement         .
   Breadcrumb (Home > Tools > Reverse Complement)      .
      FASTA             Textarea  "    " Button.
      Code Block          , " "    "   "   .
               .
```

Claude Code에서는 디자인 목업 이미지를 참조하여 구현을 요청할 수 있다:

```
>           /tools/revcomp       .
> SvelteKit + Tailwind CSS    ,  /     Card       .
```

9.7 정리

- **일반 페이지는 랜딩 페이지와 동일한 Header/Footer를 공유하되, Hero Section을 축소**
  - Breadcrumb + Heading으로 현재 위치와 페이지 제목을 표시
- **도구 페이지에서 자주 사용되는 컴포넌트를 숙지**
  - 입력: Textarea, Input, Select, File Upload, Button
  - 출력: Table, Code Block, Tab, Spinner, Alert

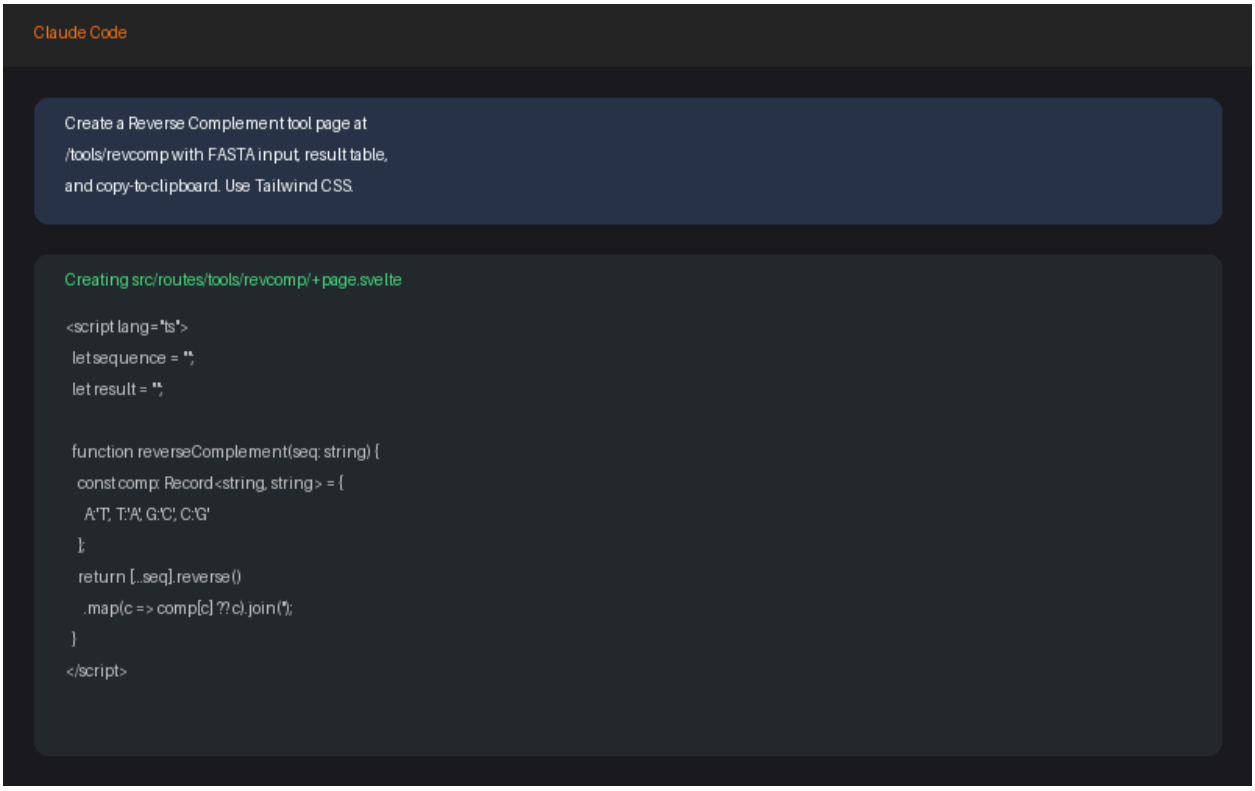

Figure 70: Claude 일반 페이지 디자인 요청

  - 시각화: Chart, Heatmap, Sequence Viewer
- **디자인 패턴을 상황에 맞게 선택**
  - 단일 도구: 입력 → 실행 → 결과 흐름
  - 다중 도구: Sidebar 활용
  - 복합 결과: Tab 기반 전환
- **SvelteKit의 파일 기반 라우팅으로 페이지를 구성**
  - 폴더 구조 = URL 구조
- **AI에게 컴포넌트 명칭을 정확히 사용하여 디자인 및 구현을 요청**