**Polars 활용 레시피 100제**

**Polars 데이터 분석 레시피 100제 (초판)**

**발 행**｜2025년 04월 01일

**저 자**｜Evan

**펴낸이**｜한건희

**펴낸곳**｜주식회사 부크크

**출판사등록**｜2014.07.15(제2014-16호)

**주 소**｜서울특별시 금천구 가산디지털1로 119 SK트윈타워 A동 305호

**전 화**｜1670-8316

**이메일**｜info@bookk.co.kr

**ISBN**｜979-11-410-0000-0

**www.bookk.co.kr**

**ⓒ Evan 2025**

본 책은 저작자의 지적 재산으로서 무단 전재와 복제를 금합니다.

Polars 데이터 분석 레시피 100제

(초판)

Evan 지음

Table of Contents

[Chapter 1. 책을 출간하면서 5](#_Toc190694691)

[1. 책을 집필하면서 느끼는 소회 5](#_Toc190694692)

[2. 무엇을 배울 수 있는가 5](#_Toc190694693)

[3. 저자소개 6](#_Toc190694694)

[Chapter 2. 개발환경설정 7](#_Toc190694695)

[1. Google Colab 7](#_Toc190694696)

[Colab에서 패키지 설치 및 업그레이드 7](#_Toc190694697)

[Colab에서 패키지 버전 확인 7](#_Toc190694698)

[2. 주요 라이브러리 8](#_Toc190694699)

[Local에서의 패키지 버전 확인 10](#_Toc190694700)

[Chapter 3. polar 시작하기 12](#_Toc190694701)

[레시피 1 – 데이터프레임 생성 12](#_Toc190694702)

[레시피 2 – 데이터 불러오기 15](#_Toc190694703)

[레시피 3 – polars 데이터프레임 주요 속성 16](#_Toc190694704)

[데이터프레임 크기 확인 16](#_Toc190694705)

[컬럼명 확인 16](#_Toc190694706)

[데이터 타입 확인 17](#_Toc190694707)

[데이터프레임의 기본 정보 17](#_Toc190694708)

[레시피 4 – 데이터 호환성 18](#_Toc190694709)

[NumPy 배열과의 호환 18](#_Toc190694710)

[Pandas DataFrame과의 호환 19](#_Toc190694711)

[레시피 5 – Polars Series 생성 20](#_Toc190694712)

[레시피 6 – Polars Series 생성 22](#_Toc190694713)

[레시피 7 – Polars DataFrame에서 Series로 변형 23](#_Toc190694714)

취업 준비생 분들에게 이 책을 바칩니다.

IT/빅데이터로 전업하고자 하는 많은 사람이 있습니다. 이 글의 저자, Sara, Evan도 전공자는 아니었습니다. 열심히 독학을 하면서, 이 분야로 진입하기 위해 매우 열심히 공부를 하며, 취업 준비를 하고 있습니다.

Evan은 IT/빅데이터와 관련된 첫 직장을 구하기 위해 100여군데의 이력서를 썼고, 같은 회사에 3년 내내 지원하고 떨어지기기를 반복한적도 있습니다. 정규직도 아니고 계약직에. 그래서, Evan은 취업 준비생의 고뇌와 고충을 매우 잘 이해하고 있습니다. 2011년 필리핀에서 만났던 그들도 생존의 위협을 느꼈듯이, 현재 취업 준비생들도 생존의 위협을 느끼고 있음을. Evan 역시 생존의 위협속에서 매일 하루 생존하기 위해 열심히 살고 있습니다.

Sara의 블로그 타이틀은 Grit 입니다. Grit의 핵심은 열정과 끈기, 단 3-6개월이 아닌 3-6년간의 노력이라고 합니다. 이 책이 취업 준비생 분들의 직업의 성공, 인생의 성공에 하나의 디딤돌이 되는 좋은 입문서가 되기를 기원합니다.

# Chapter 1. 책을 출간하면서

## 책을 집필하면서 느끼는 소회

## 무엇을 배울 수 있는가

## 저자소개

# Chapter 2. 개발환경설정

본 교재의 독자는 기본적으로 Pandas를 활용할 수 있다고 가정하므로, 별도의 Python 개발환경 설정 설명은 생략한다.[[1]](#footnote-1) 다만, 원활한 실습을 위해 Google Colab에서도 실행할 수 있도록 코드를 구성하였다. Google Colab은 브라우저 기반의 Jupyter Notebook 환경으로, 별도의 설치 없이 Python을 실행할 수 있는 장점이 있다. 또한, 실행 환경이 클라우드에 존재하므로 로컬 컴퓨터의 환경 설정과 무관하게 일관된 실행 결과를 얻을 수 있다.

## Google Colab

### Colab에서 패키지 설치 및 업그레이드

본 교재에서 활용하는 주요 패키지는 **Pandas**와 함께 **Polars**이다. Polars는 Pandas보다 속도가 빠르고 메모리 사용량이 적어 대량의 데이터를 처리하는 데 유리하다. Colab에서 최신 버전을 사용하려면 아래 명령어를 실행한 후 세션을 다시 시작해야 한다.Google Colab이나Jupyter Notebook에서 라이브러리를 설치 할 때는 ‘!’를 앞에 명시적으로 붙여줘야 한다.위 명령어를 실행한 후, [런타임] → [세션 다시 시작] 을 선택하면 변경된 버전이 적용된다.

|  |
| --- |
| !pip install polars==1.22 --quiet |

### Colab에서 패키지 버전 확인

설치된 패키지의 버전을 확인하려면 다음 명령어를 사용하면 된다. 2025년 2월 기준, Google Colab의 기본 Polars 버전은 **1.9.0**이다. 최신 버전의 변경 사항이 필요할 경우 공식 문서를 참고하면 된다.

|  |
| --- |
| import polars as pl  pl.\_\_version\_\_  [결과]  ‘1.22.0’ |

## 주요 라이브러리

본 교재에서 활용한 주요 라이브러리는 다음과 같다.

파일명 : requirements.txt

|  |
| --- |
| polars==1.22.0  pandas==2.2.3  numpy==2.0.2  matplotlib==3.9.4  seaborn==0.13.2  scikit-learn==1.6.1  graphviz==0.20.3  statsmodels==0.14.4  jupyterlab==4.3.5  pyarrow==19.0.0  deltalake==0.24.0  xlsx2csv==0.8.4  xlsxwriter==3.2.2  openpyxl==3.1.5  xlrd==2.0.1 |

데이터 분석을 수행할 때 사용되는 라이브러리는 매우 다양하다. 본 교재에서는 **데이터 처리, 분석, 시각화, 머신러닝**을 포함한 다양한 작업을 수행할 수 있도록 필요한 패키지를 정리하였다. 특히, 로컬 환경에서도 동일한 코드를 실행할 수 있도록 주요 라이브러리의 버전 정보를 명시하였다. 각 라이브러리에 대한 설명은 다음 표에서 대략적으로 확인 할 수 있다.

|  |  |
| --- | --- |
| 라이브러리 | 주요기능 |
| polars | 대용량 데이터 처리를 위한 고속 데이터프레임 연산 |
| pandas | 전통적인 데이터 분석 및 테이블 형태 데이터 조작 |
| numpy | 벡터 및 행렬 연산을 위한 과학 연산 라이브러리 |
| matplotlib | 기본적인 데이터 시각화 도구 |
| seaborn | 통계적 데이터 시각화 |
| scikit-learn | 머신러닝 모델 구축 및 평가 |
| graphviz | 그래프 및 네트워크 시각화 |
| statsmodels | 통계적 분석 및 회귀 모델링 |
| jupyterlab | Jupyter Notebook의 확장형 개발 환경 |
| pyarrow | Apache Arrow 기반의 고속 데이터 처리 |
| deltalake | Delta Lake 지원 (데이터 레이크 관리) |
| xlsx2csv | XLSX 파일을 CSV로 변환 |
| xlsxwriter | Excel 파일 생성 및 편집 |
| openpyxl | Excel 파일 읽기/쓰기 |
| xlrd | XLS 파일 읽기 지원 |

각 개인 Local에서 개발환경을 설정 할 때는 프로젝트 경로에서 다음과 같이 명령어를 실행하면 된다.

|  |
| --- |
| pip install -r requirements |

### Local에서의 패키지 버전 확인

주요 라이브러리의 설치 버전을 확인하고 싶다면 다음과 같은 코드를 작성하면 확인할 수 있다.

|  |
| --- |
| import pkg\_resources  # requirements.txt 파일에 명시된 패키지들의 설치 여부와 버전을 확인하는 코드  # 각 패키지에 대해 현재 환경에 설치된 버전을 출력하며, 설치되지 않은 경우 메시지 표시  packages = [  'polars', 'pandas', 'numpy', 'matplotlib', 'seaborn',  'scikit-learn', 'graphviz', 'statsmodels', 'jupyterlab',  'pyarrow', 'deltalake', 'xlsx2csv', 'xlsxwriter', 'openpyxl', 'xlrd'  ]  # 각 패키지를 순회하면서 버전 정보 확인  for package in packages:  try:  # pkg\_resources를 사용하여 설치된 패키지의 버전 정보 조회  version = pkg\_resources.get\_distribution(package).version  print(f"{package}: {version}")  except pkg\_resources.DistributionNotFound:  # 패키지가 설치되지 않은 경우 예외 처리  print(f"{package}: 설치되지 않음")  [결과]  polars: 1.22.0  pandas: 2.2.3  numpy: 2.0.2  matplotlib: 3.9.4  seaborn: 0.13.2  scikit-learn: 1.6.1  graphviz: 0.20.3  statsmodels: 0.14.4  jupyterlab: 4.3.5  pyarrow: 19.0.0  deltalake: 0.24.0  xlsx2csv: 0.8.4  xlsxwriter: 3.2.2  openpyxl: 3.1.5  xlrd: 2.0.1 |

# Chapter 3. polar 시작하기

## 레시피 1 – 데이터프레임 생성

**파일명 : ch01\_get\_started.ipynb**

데이터 분석의 첫 걸음은 데이터를 효과적으로 다룰 수 있는 구조를 만드는 것이다. Polars는 강력하고 직관적인 데이터프레임 기능을 제공하여 데이터 처리를 쉽고 빠르게 할 수 있다. 이번 레시피에서는 Polars 데이터프레임을 생성하는 세 가지 기본적인 방법을 살펴보도록 한다.

첫 번째는 파이썬 딕셔너리를 사용하는 방법이다. 딕셔너리의 키는 컬럼명이 되고 값은 리스트 형태의 데이터가 된다. 이 방법은 데이터가 이미 컬럼 단위로 정리되어 있을 때 유용하다.

|  |
| --- |
| df1 = pl.DataFrame({  "A": [1, 2, 3, 4, 5],  "B": ["a", "b", "c", "d", "e"],  "C": [1.1, 2.2, 3.3, 4.4, 5.5]  })  print("\n딕셔너리로 생성한 데이터프레임:")  print(df1)  [결과] |

텍스트, 스크린샷, 도표, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

두 번째는 중첩된 리스트를 사용하는 방법이다. 각 내부 리스트는 하나의 행을 나타내며, schema 매개변수를 통해 컬럼명을 별도로 지정해준다. 행 단위로 데이터가 구성되어 있을 때 적합한 방법이다.

|  |
| --- |
| df2 = pl.DataFrame([  [1, "a", 1.1],  [2, "b", 2.2],  [3, "c", 3.3],  [4, "d", 4.4],  [5, "e", 5.5]  ], schema=["A", "B", "C"])  print("\n리스트로 생성한 데이터프레임:")  print(df2)  [결과] |

텍스트, 스크린샷, 번호, 도표이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

마지막으로 Polars의 Series 객체들로 데이터프레임을 구성할 수 있다. 각 Series는 이름과 데이터를 가지며, 이들을 리스트로 묶어 데이터프레임을 생성한다. 이 방법은 개별 컬럼에 대한 세밀한 제어가 필요할 때 유용하다.

|  |
| --- |
| df3 = pl.DataFrame([  pl.Series("A", [1, 2, 3, 4, 5]),  pl.Series("B", ["a", "b", "c", "d", "e"]),  pl.Series("C", [1.1, 2.2, 3.3, 4.4, 5.5])  ])  print("\nSeries로 생성한 데이터프레임:")  print(df3)  [결과] |

텍스트, 스크린샷, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

## 레시피 2 – 데이터 불러오기

polars는 다양한 형식의 데이터를 불러올 수 있다. 문법은 기존의 pandas와 비슷하기 때문에 코드 작성하는 데 큰 어려움은 없다.

|  |  |
| --- | --- |
| 파일 형식 | 메서드 |
| CSV | pl.read\_csv() |
| Excel | pl.read\_excel() |
| Parquet | pl.read\_parquet() |
| JSON | pl.read\_json() |

아래 예제에서는 CSV 파일을 불러오는 방법을 살펴본다. 데이터셋은 소스코드에서 확인이 가능하다.

|  |
| --- |
| df = pl.read\_csv("data/titanic\_dataset.csv")  print("\ntitanic 데이터셋:")  df.head(1)  [결과] |

텍스트, 폰트, 라인, 대수학이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

## 레시피 3 – polars 데이터프레임 주요 속성

pandas와 polars의 주요 기본 문법은 유사한 것이 많다. 본 장에서는 데이터프레임의 크기, 컬럼명, 데이터 타입, 기본 통계량을 확인하는 메서드를 사용한다. 기존의 pandas를 활용한 독자들은 어렵지 않게 이해할 수 있다.

### 데이터프레임 크기 확인

pandas와 동일한 문법을 사용하며 (행 수, 열 수)를 튜플로 반환한다.

|  |
| --- |
| print("\n데이터프레임의 크기(shape):")  df.shape  [결과]  데이터프레임의 크기(shape):  (891, 12) |

### 컬럼명 확인

pandas와 동일한 문법을 사용하며 리스트로 반환한다.

|  |
| --- |
| print("\n데이터프레임의 컬럼명:")  print(df.columns)  [결과]  데이터프레임의 컬럼명:  ['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp', 'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'] |

### 데이터 타입 확인

pandas와 동일한 문법을 사용하며, 리스트로 반환한다.

|  |
| --- |
| print("\n데이터프레임의 데이터 타입:")  print(df.dtypes)  [결과]  데이터프레임의 데이터 타입:  [Int64, Int64, Int64, String, String, Float64, Int64, Int64, String, Float64, String, String] |

### 데이터프레임의 기본 정보

pandas와 동일한 문법을 사용하며, 리스트로 반환한다.

|  |
| --- |
| print("\n데이터프레임의 기본 정보:")  print(df.describe())  [결과] |

텍스트, 번호, 평행이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

## 레시피 4 – 데이터 호환성

다른 데이터 형식과의 호환성을 알아보도록 한다. polars는 다른 데이터 관련 라이브러리인 NumPy 배열과의 호환, pandas 데이터프레임과의 호환성이 뛰어나다.

### NumPy 배열과의 호환

NumPy 배열을 polars의 DataFrame으로 변환하는 방법은 **from\_numpy()** 메서드를 활용하는 것이다. **polars.from\_numpy()** 메서드는 NumPy 배열을 **polars.DataFrame**으로 변환하는 기능을 제공한다. 이때, **schema** 매개변수를 사용하면 데이터프레임의 컬럼명을 지정할 수 있다.

|  |
| --- |
| # numpy 배열 생성  import numpy as np  arr = np.array([[1, 2, 3], [4, 5, 6]])  print("\nNumPy 배열:")  print(arr)  # numpy 배열을 polars 데이터프레임으로 변환  df\_from\_np = pl.from\_numpy(arr, schema=['col1', 'col2', 'col3'])  print("\nNumPy 배열로부터 생성된 polars 데이터프레임:")  print(df\_from\_np)  [결과] |

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

### Pandas DataFrame과의 호환

polars는 pandas와의 호환성을 제공하여 두 라이브러리 간 데이터 변환을 쉽게 할 수 있다. pandas.DataFrame을 polars.DataFrame으로 변환하거나 그 반대로 변환하는 기능을 지원한다. Pandas 데이터프레임을 polars 데이터프레임으로 변환할 때는 **polars.from\_pandas()** 메서드를 사용한다.

|  |
| --- |
| # pandas 데이터프레임 생성  import pandas as pd  pd\_df = pd.DataFrame({  'A': [1, 2, 3],  'B': ['a', 'b', 'c']  })  print("\nPandas 데이터프레임:")  print(pd\_df)  # pandas 데이터프레임을 polars로 변환  pl\_from\_pd = pl.from\_pandas(pd\_df)  print("\nPandas 데이터프레임으로부터 생성된 polars 데이터프레임:")  print(pl\_from\_pd)  [결과] |

텍스트, 스크린샷, 도표, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

## 레시피 5 – Polars Series 생성

**polars.Series**는 **pandas.Series**와 유사한 1차원 데이터 구조이며, polars.DataFrame의 개별 컬럼을 표현하는 객체이다. 하나의 데이터 타입을 가지며, 벡터 연산을 지원하여 빠르고 효율적으로 데이터를 처리할 수 있다. **polars.Series** 객체를 생성하는 방법은 다양하게 있지만, 본 장에서는 예시로 직접 생성, NumPy 배열을 활용하여 생성, 데이터 타입을 직접 지정하여 생성하는 방법을 소개한다.

먼저 직접 생성하는 방법은 다음과 같다.

|  |
| --- |
| # Series 생성  s1 = pl.Series("a", [1, 2, 3])  print("\nSeries s1:")  print(s1)  [결과]  shape: (3,)  Series: 'B' [f64]  [  1  2  3  ] |

이번에는 NumPy 배열을 이용하여 Series를 생성한다.

|  |
| --- |
| # NumPy 배열을 이용하여 Series 생성  s\_np = pl.Series("B", np.array([0.1, 0.2, 0.3]))  print(s\_np)  [결과]  shape: (3,)  Series: 'B' [f64]  [  0.1  0.2  0.3  ] |

이번에는 데이터 타입을 Float32로 지정해서 생성하도록 한다. 이 때, polars.datatypes.Float32 클래스를 호출한다.

|  |
| --- |
| from polars.datatypes import Float32  # 데이터 타입을 Float32로 지정  s\_typed = pl.Series("C", [1.1, 2.2, 3.3], dtype=Float32)  print(s\_typed)  [결과]  shape: (3,)  Series: 'C' [f32]  [  1.1  2.2  3.3  ] |

## 레시피 6 – Polars Series 생성

위 코드는 **polars.Series**의 기본 속성을 확인하는 예제이다. 먼저, "a"라는 이름을 가진 polars.Series 객체 s1을 생성하고, [1, 2, 3, 4, 5]의 정수 리스트를 데이터로 설정한다. 이후, Series의 주요 속성을 출력하는데, **s1.name**을 통해 시리즈의 이름을 확인하고, **s1.len()**을 사용하여 시리즈의 길이를 출력한다. 또한, **s1.dtype**을 이용해 데이터 타입을 확인하며, **s1.null\_count()**를 통해 결측값(Null)의 개수를 확인할 수 있다. 마지막으로, **s1.n\_unique()**를 사용하여 시리즈 내 고유값 개수를 출력한다. 이를 통해 polars.Series의 기본 속성을 쉽게 파악할 수 있으며, 데이터 분석 과정에서 유용하게 활용할 수 있다.

|  |
| --- |
| # Series의 기본 속성들  s1 = pl.Series("a", [1, 2, 3, 4, 5])  print("\nSeries의 기본 속성들:")  print(f"이름: {s1.name}")  print(f"길이: {s1.len()}")  print(f"데이터 타입: {s1.dtype}")  print(f"Null 값 개수: {s1.null\_count()}")  print(f"고유값 개수: {s1.n\_unique()}")  [결과]  Series의 기본 속성들:  이름: a  길이: 5  데이터 타입: Int64  Null 값 개수: 0  고유값 개수: 5 |

## 레시피 7 – Polars DataFrame에서 Series로 변형

아래 코드는 polars.DataFrame을 생성하고, 개별 컬럼을 polars.Series로 추출하는 예제이다. 먼저, pl.DataFrame()을 사용하여 df라는 데이터프레임을 생성하며, 컬럼 "a"에는 정수 리스트 [1, 2, 3], 컬럼 "b"에는 문자열 리스트 ["x", "y", "z"], 컬럼 "c"에는 불리언 값 [True, False, True]가 들어간다. 그 후, df["컬럼명"]을 이용해 개별 컬럼을 Series로 변환한다. "a" 컬럼은 series\_a, "b" 컬럼은 series\_b, "c" 컬럼은 series\_c로 저장되며, 각각 정수, 문자열, 불리언 타입의 polars.Series가 된다. 마지막으로, 원본 데이터프레임과 변환된 Series 객체들을 출력하여, DataFrame에서 컬럼을 추출하는 방법과 Series로 변환된 형태를 확인할 수 있다. 이를 활용하면 DataFrame의 특정 컬럼을 독립적으로 다루거나, 벡터 연산 및 데이터 분석을 수행할 수 있다.

|  |
| --- |
| # 데이터프레임 생성  df = pl.DataFrame({  "a": [1, 2, 3],  "b": ["x", "y", "z"],  "c": [True, False, True]  })  print("\n원본 데이터프레임:")  print(df) |

텍스트, 스크린샷, 도표, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

|  |
| --- |
| # 데이터프레임의 컬럼을 Series로 변환  series\_a = df["a"]  series\_b = df["b"]  series\_c = df["c"]  print("\n추출된 Series들:")  print("Series a:")  print(series\_a)  print("\nSeries b:")  print(series\_b)  print("\nSeries c:")  print(series\_c)  [결과]  추출된 Series들:  Series a:  shape: (3,)  Series: 'a' [i64]  [  1  2  3  ]  Series b:  shape: (3,)  Series: 'b' [str]  [  "x"  "y"  "z"  ]  Series c:  shape: (3,)  Series: 'c' [bool]  [  true  false  true  ] |

## 레시피 8 – Polars LazyFrame 생성

### LazyFrame

LazyFrame은 polars에서 제공하는 **지연 실행(Lazy Execution) 데이터프레임**으로, DataFrame과 달리 즉시 연산을 수행하지 않고, 최적화된 실행 계획을 세운 후 연산을 실행하는 방식이다. 즉, 데이터를 다룰 때 모든 연산을 바로 적용하는 것이 아니라, 필요할 때 한꺼번에 실행하여 성능을 극대화할 수 있다. 이는 대용량 데이터 처리에서 특히 유용하다.

### LazyFrame vs. DataFrame

|  |  |  |
| --- | --- | --- |
| 비교항목 | LazyFrame | DataFrame |
| 실행방식 | 지연 실행(Lazy Execution) | 즉시 실행(Eager Execution) |
| 속도 | 대용량 데이터 최적화 | 작은 데이터에 적합 |
| 메모리 사용 | 최적화 후 실행되어 효율적 | 더 많이 사용 |
| API 지원 | 동일한 연산 지원  .collect() 호출해야 실행 | .filter(), .select() 등 지원 |

아래 코드는 polars.LazyFrame을 사용하여 딕셔너리 데이터를 기반으로 지연 실행(Lazy Execution) 방식의 데이터프레임을 생성하는 예제이다. 먼저, data라는 이름의 딕셔너리를 정의하고, "a", "b", "c"라는 세 개의 키를 가진 리스트를 값으로 설정한다. "a"는 정수 리스트 [1, 2, 3], "b"는 문자열 리스트 ["x", "y", "z"], "c"는 불리언 값 [True, False, True]로 구성된다. 그 후, pl.LazyFrame(data)를 사용하여 해당 데이터를 LazyFrame 객체 lf로 변환한다.

|  |
| --- |
| # 딕셔너리에서 LazyFrame 직접 생성  data = {  "a": [1, 2, 3],  "b": ["x", "y", "z"],  "c": [True, False, True]  }  lf = pl.LazyFrame(data)  print("\n딕셔너리로부터 직접 생성된 LazyFrame:")  print(lf)  [결과]  딕셔너리로부터 직접 생성된 LazyFrame:  naive plan: (run LazyFrame.explain(optimized=True) to see the optimized plan)  DF ["a", "b", "c"]; PROJECT \*/3 COLUMNS |

 Polars에서 LazyFrame을 DataFrame으로 변환하는 예시를 보여주고 있다. LazyFrame은 실제 연산을 지연시키고 쿼리 최적화를 수행할 수 있게 해주는 Polars의 중요한 기능입니다. 대규모 데이터셋을 처리할 때 특히 유용하다.

|  |
| --- |
| # LazyFrame을 DataFrame으로 수집  df = lf.collect()  print("\n수집된 DataFrame:")  print(df)  [결과] |

텍스트, 도표, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

|  |
| --- |
| # 기본 데이터프레임 생성  df = pl.DataFrame({  "a": [1, 2, 3],  "b": ["x", "y", "z"],  "c": [True, False, True]  })  # LazyFrame으로 변환  lf = df.lazy()  print("\nLazyFrame으로 변환:")  print(lf)  [결과]  LazyFrame으로 변환:  naive plan: (run LazyFrame.explain(optimized=True) to see the optimized plan)  DF ["a", "b", "c"]; PROJECT \*/3 COLUMNS |

위 코드는 polars.DataFrame에서 **.lazy()**를 사용하여 LazyFrame으로 변환하는 방법을 보여준다.

## 레시피 9 – Polars LazyFrame로 파일 읽어오기

### scan\_csv()

scan\_csv()는 Polars의 **Lazy API** 기능을 활용하여 CSV 파일을 즉시 로드하는 것이 아니라, **지연 평가(Lazy Evaluation) 방식**으로 읽어들이는 함수이다. 이 메서드를 활용하면, 파일 전체를 즉시 읽어들이지 않고, 필요한 데이터만 읽도록 최적화되어 있다.

1. 개발환경설정이 궁금하다면 다음 필자의 유투브에 접속해서 확인한다. https://youtu.be/BqkwJ5zCMb0?si=ylqt185WSSPJ-x9q [↑](#footnote-ref-1)