

Chapter 12. 생성형 AI 활용

1. 생성형 AI

바야흐로 생성형 AI 솔루션 전성시대다. 봇물처럼 쏟아져서 나오고 있다. 생성형 AI, ChatGPT의 데뷔는 소비자 AI 애플리케이션의 새로운 시대를 열며 큰 변화를 가져왔습니다. SimilarWeb 데이터를 기반으로 웹 트래픽을 조사하여 2024년 1월 기준 상위 50개의 AI 우선 웹 제품을 순위로 매겼습니다. 모바일 앱 순위는 웹 순위와 마찬가지로 Sensor Tower 데이터를 기준으로 월간 활성 사용자 수(MAU)를 바탕으로 작성되었다.

The Top 50 Gen AI Web Products, by Unique Monthly Visits				
1. ChatGPT	11. ElevenLabs	21. PhotoRoom	31. PIXAI	41. MaxAI.me
2. Gemini*	12. Hugging Face	22. Yodayo	32. ideogram	42. Craiyon
3. character.ai	13. Leonardo.AI	23. Clipchamp	33. invideo AI	43. OpusClip
4. liner	14. Midjourney	24. runway	34. Replicate	44. BLACKBOX AI
5. QuillBot	15. SpicyChat	25. YOU	35. Playground	45. CHATPDF
6. Poe	16. Gamma	26. DeepAI	36. Suno	46. PIXELCUT
7. perplexity	17. Crushon AI	27. Eightify	37. Chub.ai	47. Vectorizer.AI
8. JanitorAI	18. cutout.pro	28. candy.ai	38. Speechify	48. DREAMGF
9. CIVITAI	19. PIXLR	29. NightCafe	39. phind	49. Photomyne
10. Claude	20. VEED.IO	30. VocalRemover	40. NovelAI	50. Otter.ai

*Recently Launched
Charts are for informational purposes only and should not be used for investment decisions. Past performance is not indicative of future results. None of the above should be taken as investment advice; see [alz.com/disclosures](#).

alz Consumer

현재 다양한 AI 기반 애플리케이션이 쏟아져 나오는 가운데, 특히 취업준비생에게 유용한 두 가지 솔루션을 소개합니다: **Cursor AI**와 **Napkin AI**. 이 두 도구는 취업준비생들이 코딩 및 프레젠테이션 준비를 효율적으로 지원하며, 차별화된 포트폴리오를 만드는 데 도움을 준다.

Cursor AI는 코드 작성과 디버깅 과정을 혁신적으로 단축시켜 줍니다. 초보 개발자부터 숙련된 사용자까지 모두가 효율적으로 사용할 수 있도록 설계되었으며, 자연어로 설명하면 이를 기반으로 코드를 생성하거나 수정할 수 있다. 특히, 취업을 위한 코딩 과제나 기술 포트폴리오 작업에서 시간을 절약하고 높은 완성도를 갖춘 결과물을 얻을 수 있어 개발 능력을 어필하기에 유용하다. 한편, **Napkin AI**는 프레젠테이션 및 시각 자료 제작에 특화된 도구입니다. AI 기반으로 아이디어를 구조화하고 깔끔한 슬라이드를 빠르게 생성할 수 있어, 취업준비생들이 시간과 노력을 절감하며 전문성을 드러낼 수 있는 파워포인트 포트폴리오를 제작할 수 있다. 결과적으로 두 도구를 활용하면 코딩 실력과 프레젠테이션 역량을 동시에 강화할 수 있어, 치열한 구직 시장에서 한 걸음 앞서 나갈 수 있다.

2. Cursor AI

Cursor AI는 소프트웨어 개발을 간소화하도록 설계된 AI 기반 코드 편집기이다. 이 도구는 Visual Studio Code(VS Code)를 기반으로 하여 사용하기 쉬운 인터페이스와 VS Code의 풍부한 에코시스템을 그대로 유지한다. 이를 통해 VS Code에 익숙한 개발자들이 손쉽게 전환할 수 있다. Cursor AI는 OpenAI의 ChatGPT와 Claude를 통합하여 지능형 코드 제안, 자동 오류 감지, 동적 코드 최적화 등의 기능을 제공한다. Cursor AI는 코드 자동 완성과 예측 기능에서 다음과 같은 핵심 기능을 제공한다. 주요 자동 완성 기능과 채팅 기능 설명을 요약 정리하였다.

먼저 주요 자동 완성 기능을 보면, 다음과 같다.

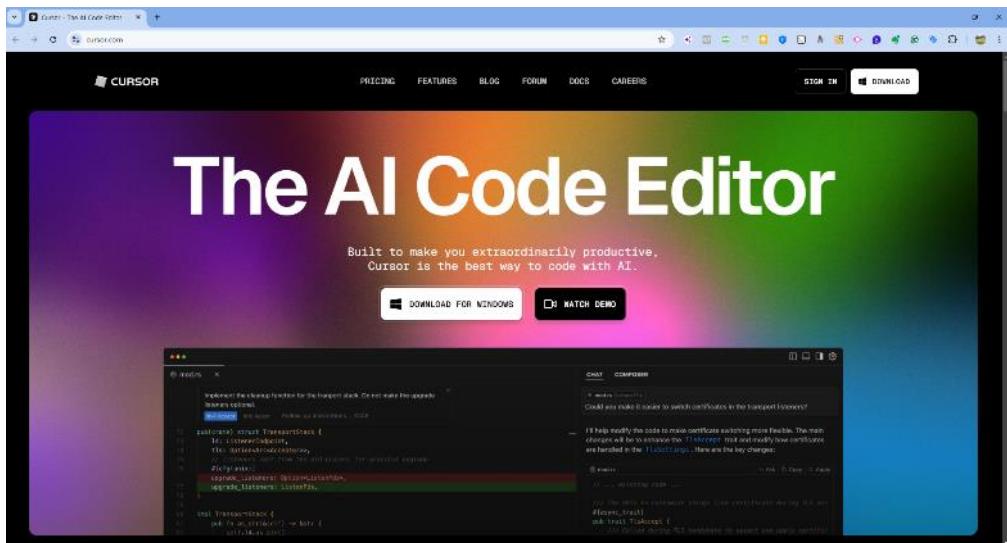
- 자동 완성과 코드 예측: 여러 줄에 걸친 코드 자동 완성을 지원하며, 최근 코드 변경사항에 따라 제안 내용을 조정한다.
- 코드 생성: 코드 작성 의도를 예측하고, 적합한 코드를 추천한다.
- 다중 줄 수정: 여러 줄에 걸친 수정 사항을 제안한다.
- 스마트 리라이트(Smart Rewrite): 부주의하게 작성된 코드도 자동으로 수정 및 개선한다.
- 커서 예측: 코드 탐색을 원활하게 할 수 있도록 커서의 다음 위치를 예측 한다.

채팅 기능을 보면 다음과 같다.

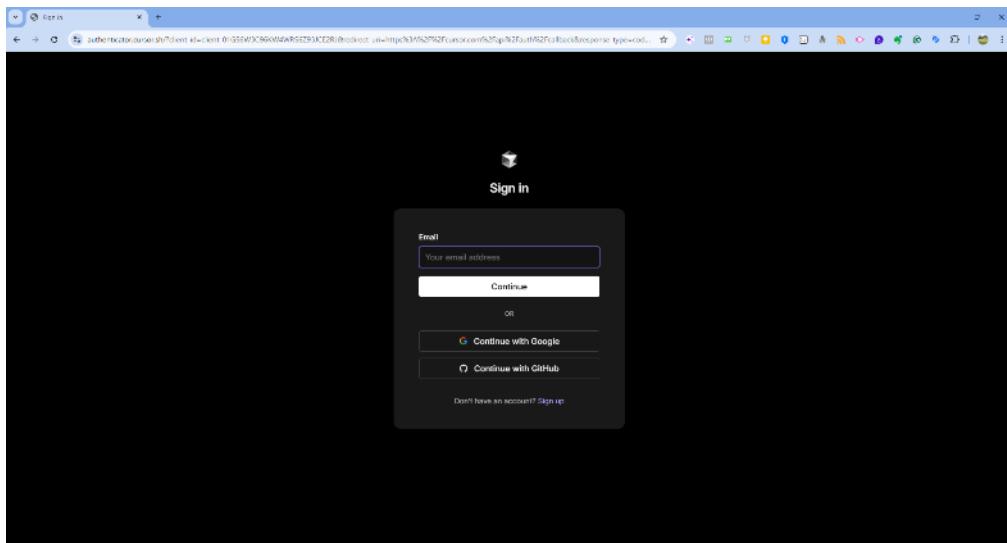
- 코드베이스 답변: 코드베이스 전체를 검색하여 관련 정보를 제공한다.
- 코드 참조: 특정 코드 블록이나 파일을 참조해 질문의 맥락에 통합한다.
- 이미지 지원: 이미지 드래그 앤 드롭 및 버튼 클릭을 통해 시각적 맥락을 추가할 수 있다.
- 웹 검색: 인터넷에서 최신 정보를 검색하여 코드 관련 질문에 통합한다.
- 즉시 적용: 채팅에서 제공된 코드 제안을 클릭 한 번으로 코드베이스에 바로 적용할 수 있다.
- 문서 통합: 인기 있는 라이브러리의 문서를 참조하거나, 사용자 정의 문서를 추가하여 빠르게 접근할 수 있다.

A. Cursor AI 설치

먼저 웹사이트¹⁵⁸에서 회원가입을 하도록 한다. Sign In 버튼을 클릭한다.



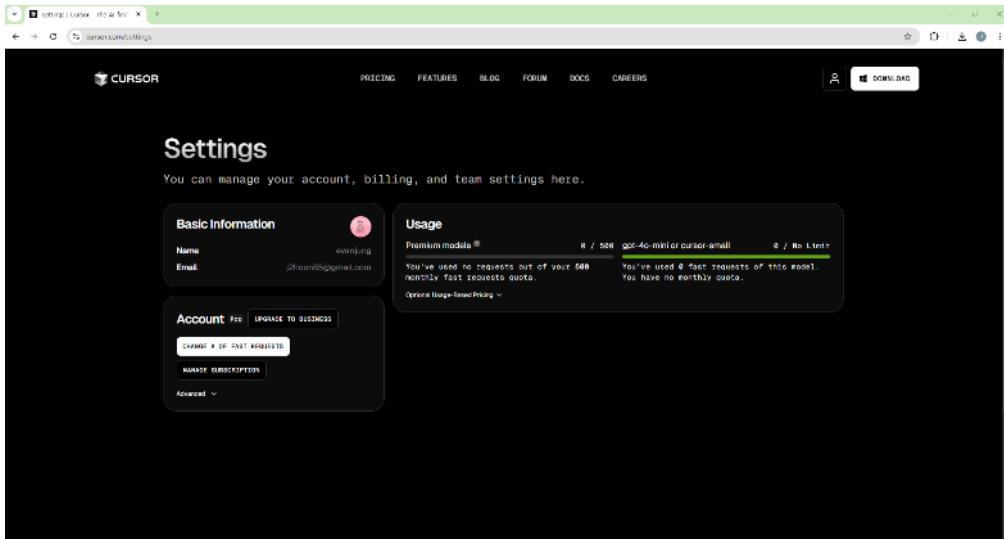
아래 화면에서 Continue with Github 버튼을 클릭한다.



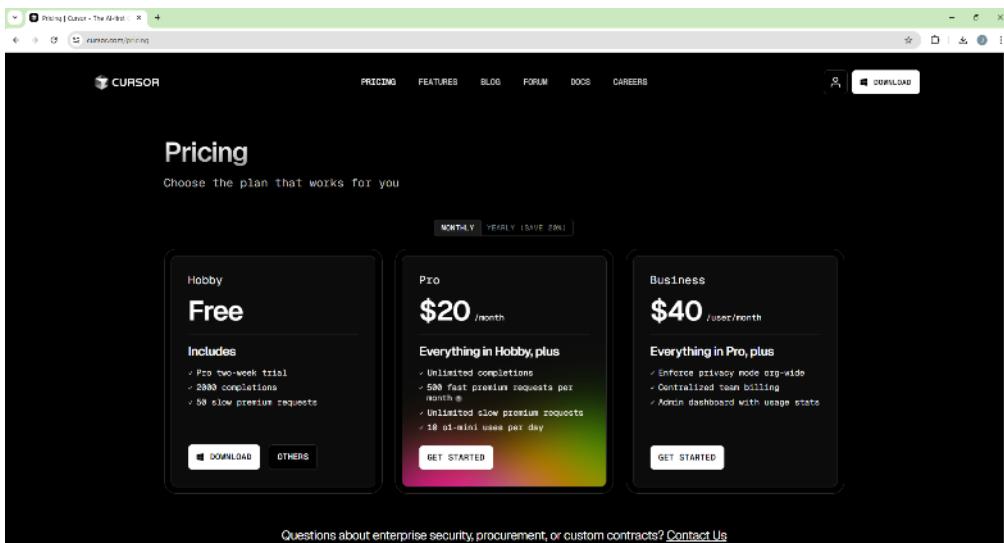
아래와 같이 설정화면이 나온다면 회원가입은 잘 된 것이다. 오른쪽 상단의 Download 버

158 웹사이트 : <https://www.cursor.com/>

튼을 클릭하여 설치 프로그램을 다운로드 받는다.

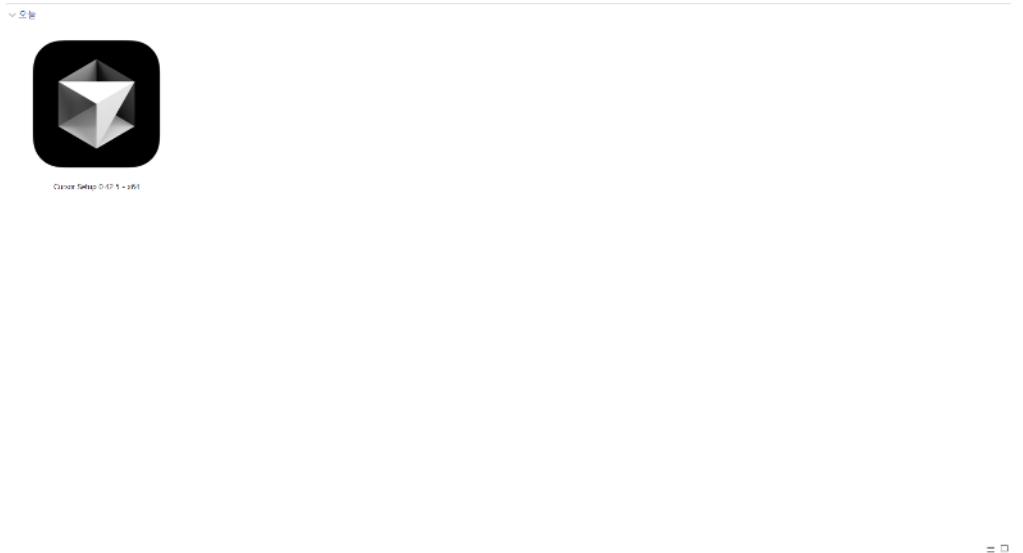


가격정책은 다음 링크를 참조한다.¹⁵⁹

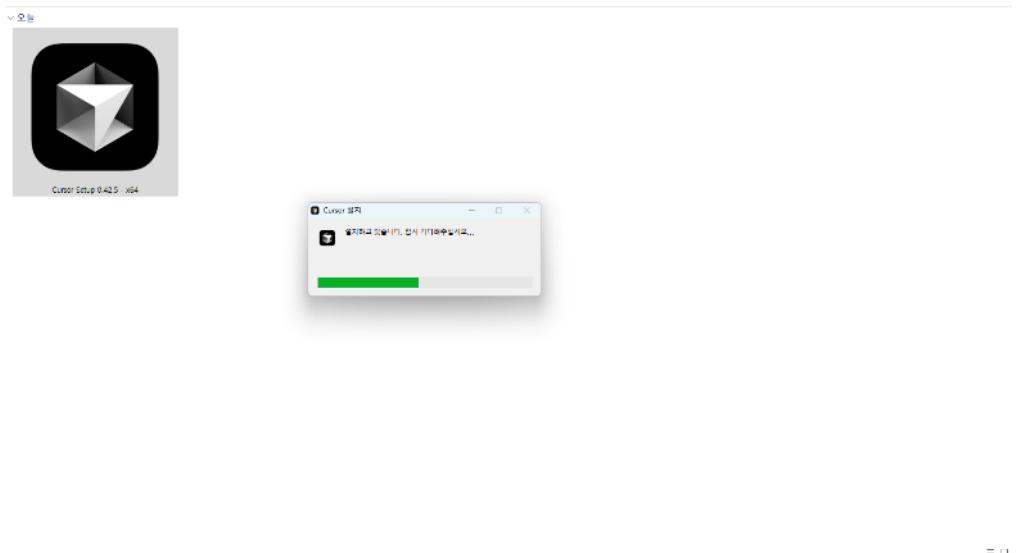


이번에는 프로그램을 설치하도록 한다. 먼저 Windows 버전에서 설치프로그램 확인한다.

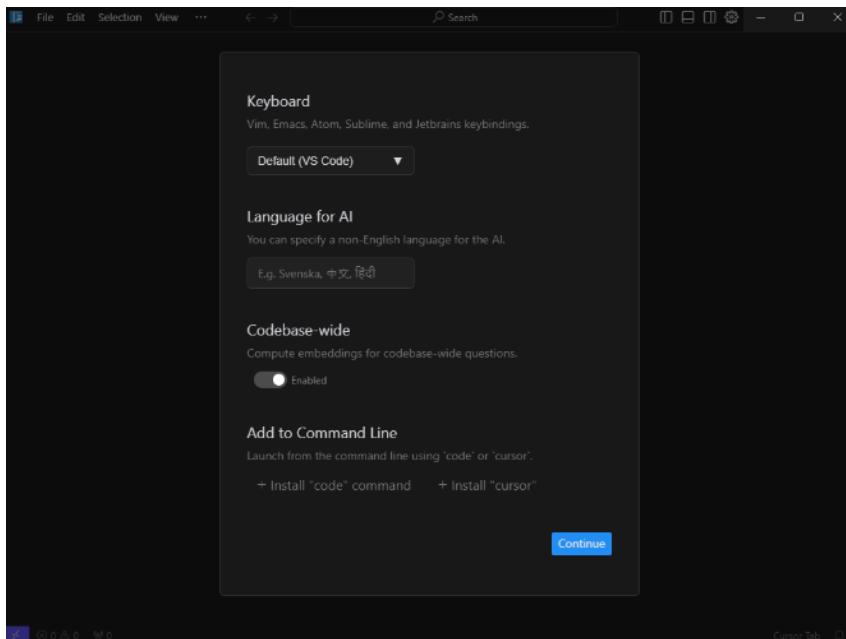
¹⁵⁹ <https://www.cursor.com/pricing> 취업준비생이라면 프로젝트를 빠르게 만들기 위해서는 유료 사용을 권한다.



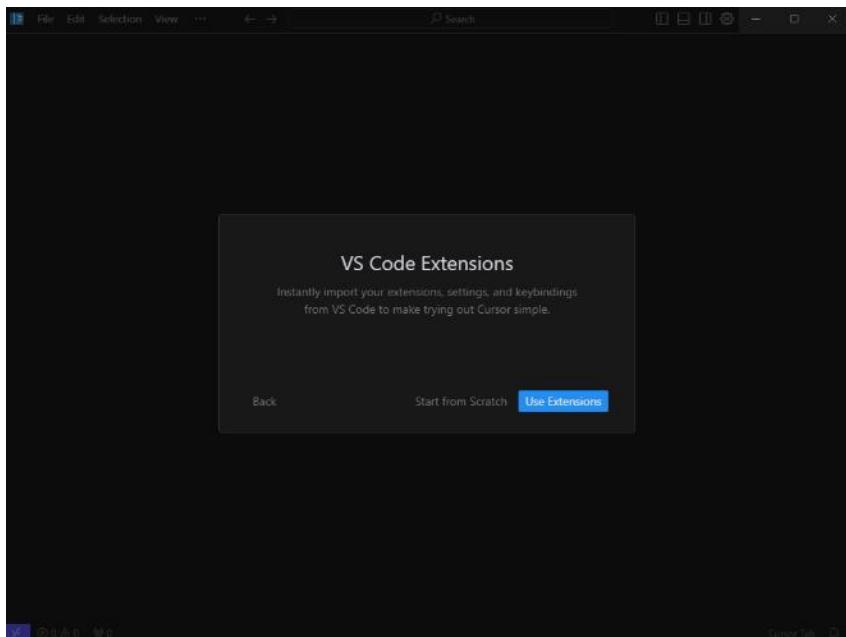
관리자 권한으로 실행한다.



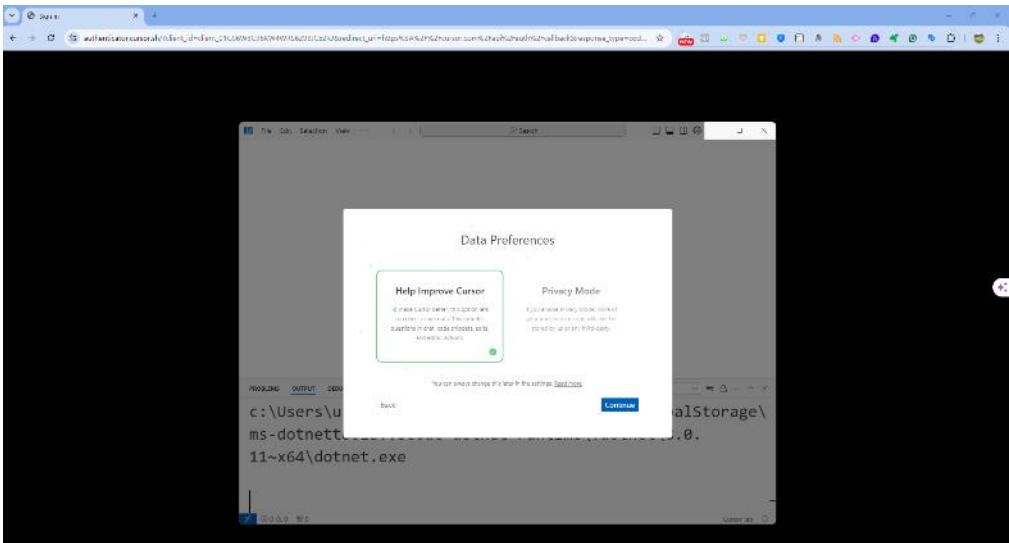
메뉴 확인 후, Continue 버튼을 선택한다.



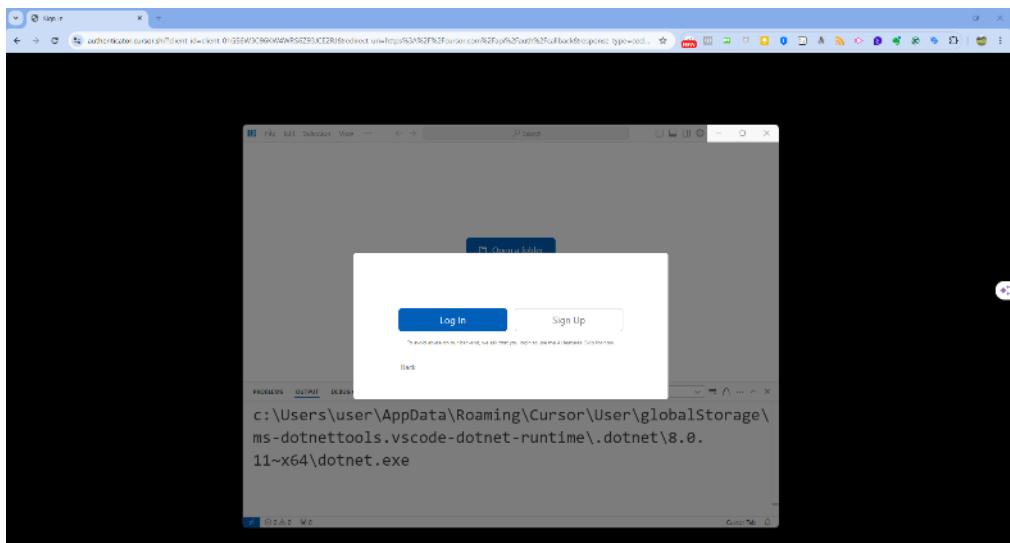
Use Extensions 선택한다.



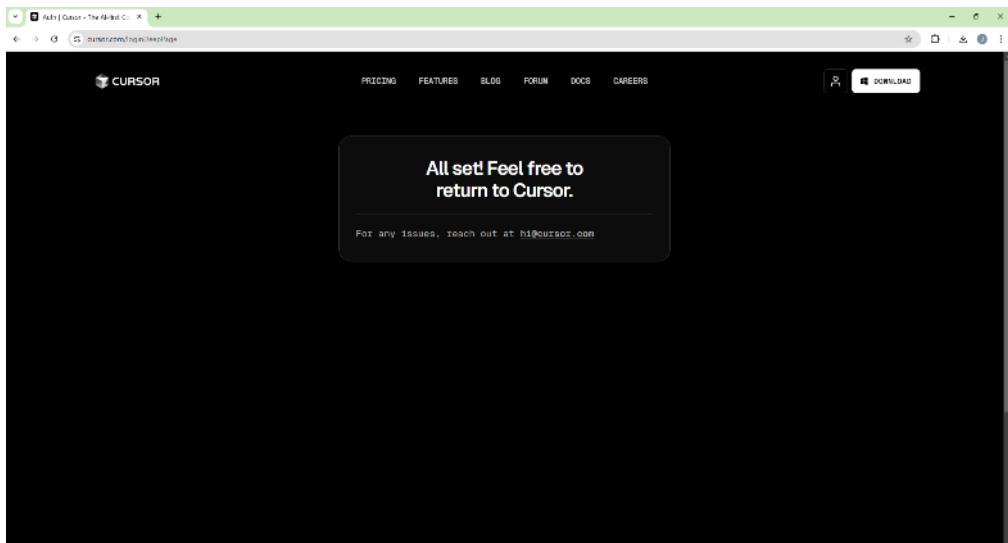
• Data Preferences는 독자 취향에 맞게 선택한다.



이제 로그인을 하도록 한다.

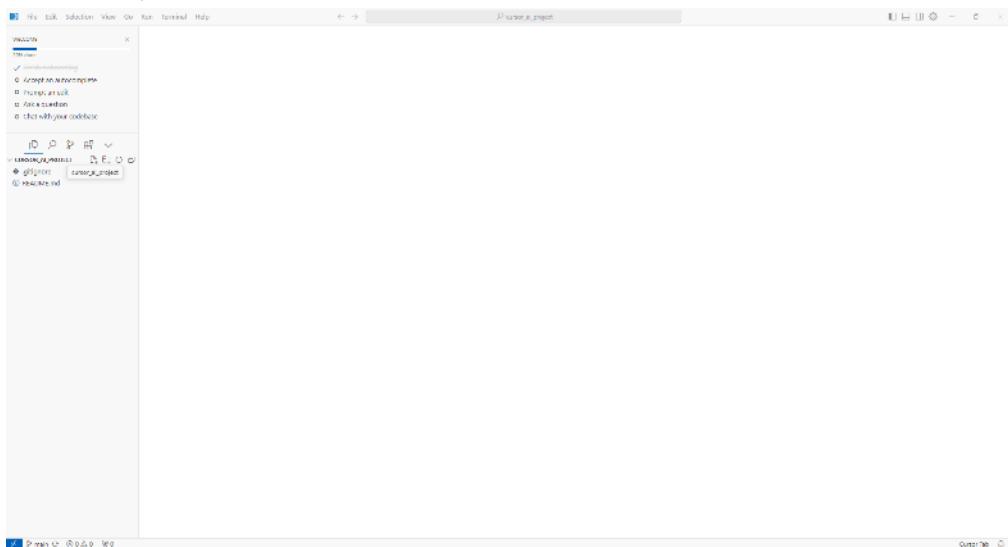


개인 계정 확인 후, Yes, Log in 버튼 클릭하면 아래와 같은 화면이 나온다.



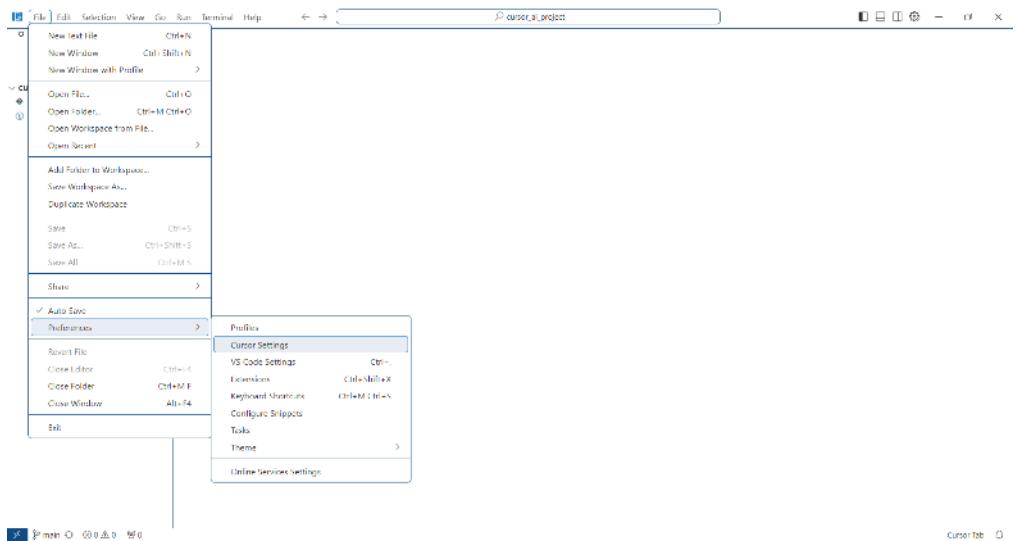
Visual Studio Code 확인을 하도록 한다.

- 이제 Visual Studio Code에 Cursor AI가 업데이트가 되었는지 확인해본다.
- 그러기 위해서는, 먼저 Github에서 새로운 Repository를 하나 생성한다. 필자는 cursor_ai_project로 명명했다.
- 해당 Repo를 생성한 후, git clone으로 다운로드 받는다.
- Visual Studio Code에서 Repository를 열면, 다음과 같은 화면이 나온다.

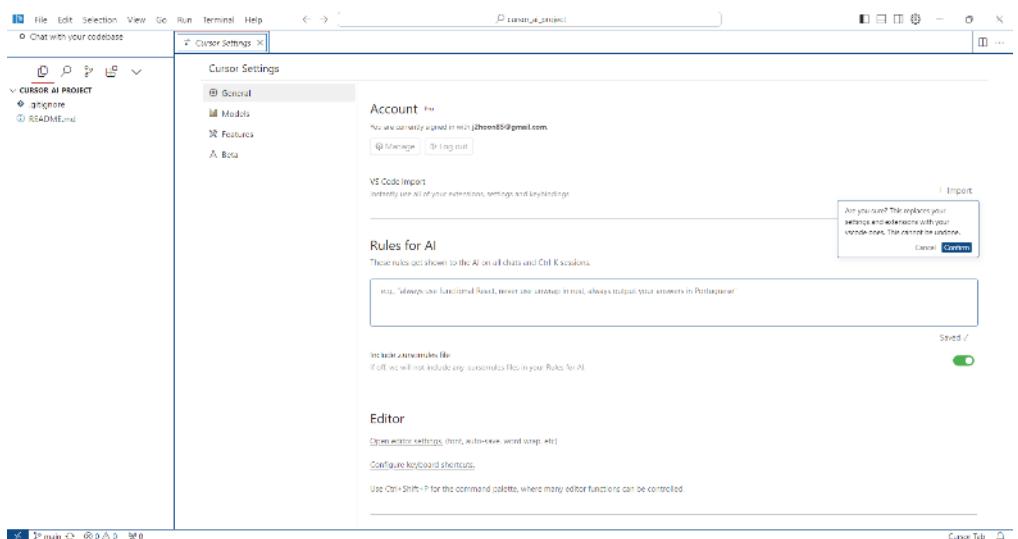


VS Code에서 클릭 한번으로 관련 설정을 모두 가져올 수 있다.

File > Preferences > Cursor Settings > General > Account



Import 버튼 선택 후, Confirm을 클릭한다.

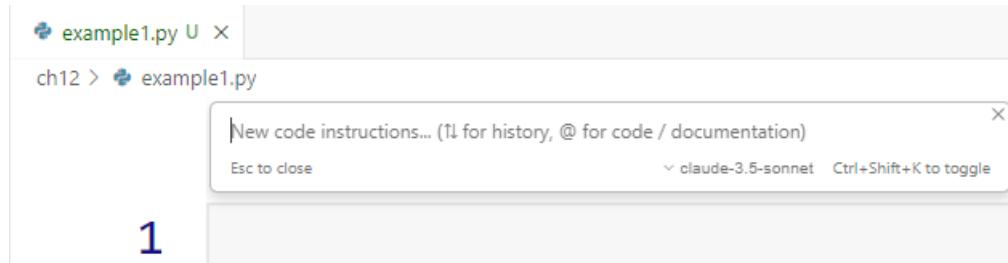


B. Cursor AI 실습

코드 실습하는 방법은 크게 어렵지 않다. 주요 핵심 단축키는 코드 생성을 위한 명령어 입력기 Ctrl + K / Cmd + K이고, Chat 기능을 지원하는 명령어 입력기 Ctrl + L / Cmd + L이다. 이제 주요 실습 예시를 살펴보도록 한다.¹⁶⁰

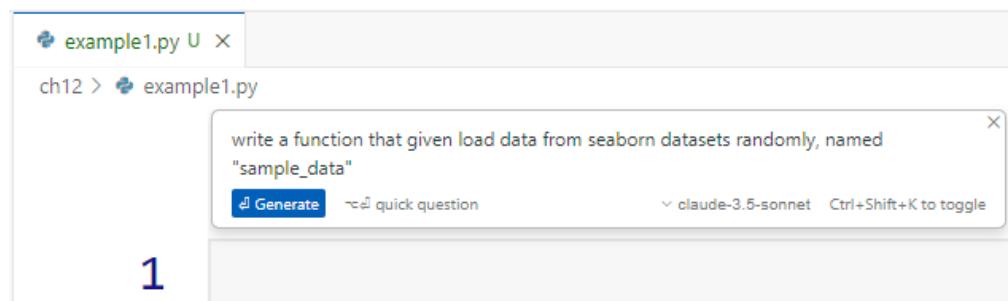
- 코드 생성 및 보완

Cmd+K 단축키를 사용하여 인라인 코드 생성기를 연다. 이를 통해 작은 프롬프트 창이 열리며, 여기에서 코드 생성을 위한 프롬프트를 입력할 수 있다. 먼저 ch12 폴더를 열고 example1.py 파일을 생성한다. 그 후에 Ctrl + K를 실행하면 다음과 같은 명령어가 나온다.



그 후에 다음과 같이 명령어를 입력 후, Enter를 실행한다.

```
write a function that given load data from seaborn datasets randomly, named  
"sample_data"
```



¹⁶⁰ 본 실습은 소스코드에 설명은 별도로 하지 않는다. 실제 사용자마다 결과는 다르게 나올 수 있다.

A screenshot of a code editor window titled "example1.py U x". The status bar shows "ch12 > example1.py". A code completion dialog box is open, containing the instruction "write a function that given load data from seaborn datasets randomly, named 'sample_data'". Below the dialog, the Python code is displayed:

```
1 def sample_data():
2     """Load a random dataset from seaborn's built-in datasets"""
3     import seaborn as sns
4     import random
5
6     # Get list of available datasets
7     dataset_names = sns.get_dataset_names()
8
9     # Randomly select one
10    selected_dataset = random.choice(dataset_names)
11
12    # Load and return the selected dataset
13    return sns.load_dataset(selected_dataset)
14
```

위와 같이 코드가 자동으로 생성되는 것을 확인할 수 있다. Ctrl + Enter를 실행하면 코드가 반영될 것이다.

A screenshot of a code editor window titled "example1.py U x". The status bar shows "ch12 > example1.py". The Python code is identical to the one in the previous screenshot, but now includes a message at the bottom:

```
1 def sample_data():
2     """Load a random dataset from seaborn's built-in datasets"""
3     import seaborn as sns
4     import random
5
6     # Get list of available datasets
7     dataset_names = sns.get_dataset_names()
8
9     # Randomly select one
10    selected_dataset = random.choice(dataset_names)
11
12    # Load and return the selected dataset
13    return sns.load_dataset(selected_dataset)
14
15
16 Ctrl+L to chat, Ctrl+K to generate
```

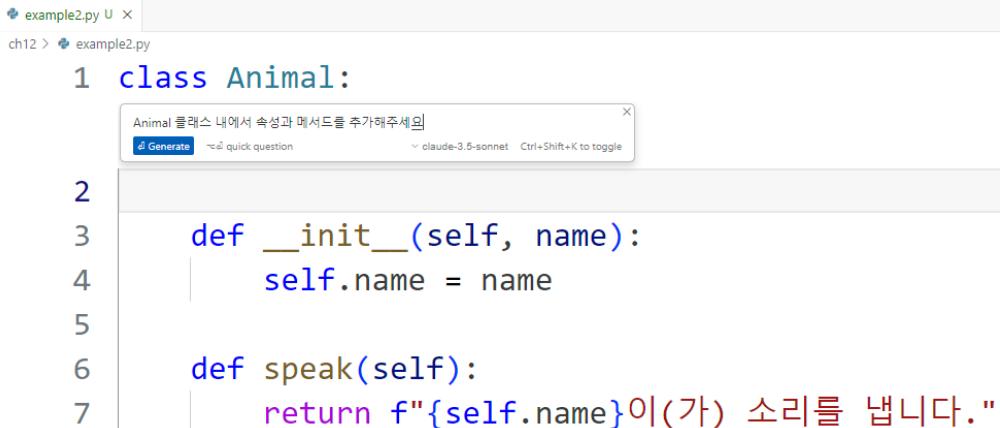
- 기존 코드와 상호작용

Cmd+K 단축키를 사용하기 전에 관련 코드를 선택하면 인라인 채팅을 통해 기존 코드와 상호작용할 수 있다. 이를 통해 코드 리팩토링과 같은 변경을 수행하거나 코드에 대한 질문을 할 수 있다. 프롬프트를 입력한 후 Submit Edit 버튼을 클릭하면 수정된 내용을 확인할 수 있다. 다음과 같은 코드가 있다고 가정한다. 파일명 : example2.py

```
class Animal:  
    def __init__(self, name):  
        self.name = name  
  
    def speak(self):  
        return f"{self.name}이(가) 소리를 냅니다."
```

이 때, 속성과 메서드를 추가로 작성해달라고 한다. 이번에는 한글로 요청한다. Ctrl + Enter를 실행하여 Accept 한다.

Animal 클래스 내에서 속성과 메서드를 추가해주세요.



The screenshot shows a code editor window with the file 'example2.py' open. The code defines a class 'Animal' with an __init__ method and a speak method. A tooltip is displayed above the code, containing the text 'Animal 클래스 내에서 속성과 메서드를 추가해주세요'. Below the tooltip, there are buttons for 'Generate', 'quick question', and 'claude-3.5-sonnet'. The code editor interface includes tabs for 'example2.py' and 'ch12 > example2.py'. The code itself is numbered from 1 to 7.

```
1 class Animal:  
2     def __init__(self, name):  
3         self.name = name  
4     def speak(self):  
5         return f"{self.name}이(가) 소리를 냅니다."
```

```
example2.py U x
ch12 > example2.py

1 class Animal:
2     # 클래스 속성
3     species = "동물"
4     count = 0
5
6     # 클래스 메서드
7     @classmethod
8     def get_count(cls):
9         return cls.count
10
11    @classmethod
12    def increment_count(cls):
13        cls.count += 1
```

Animal 클래스 내에서 속성과 메서드를 추가해주세요
Ctrl+Shift+A Accept Reject Follow-up instructions... Ctrl+Shift+K

만약, 중간에 오타가 있다고 가정한다. 예를 들면, `get_count()` 함수에서 매개변수를 제거하고 코드 디버깅이라고 요청한다. 그럼 수정된 코드를 같이 보여준다.

```
example2.py U x
ch12 > example2.py

1 class Animal:
2     # 클래스 속성
3     species = "동물"
4     count = 0
5
6     # 클래스 메서드
7     @classmethod
8     def get_count(cls):
9         return cls.count
10    @classmethod
11    def get_count():
12        return cls.count
```

코드 디버깅
Ctrl+Shift+A Accept Reject Follow-up instructions... Ctrl+Shift+K

- 기존 코드에 대해 질문하기

같은 방법으로, 코드를 선택한 후 Ctrl + K (Cmd + K) 단축키를 사용하여 해당 코드에 대해 질문을 입력할 경우 Quick Question 버튼을 클릭하여 프롬프트를 제출한다. 먼저 다음과 같은 코드가 있다고 가정한다. 파일명 : example3.py

```
from datetime import datetime

class BankAccount:
    def __init__(self, account_holder, initial_balance=0):
        self.account_holder = account_holder
        self.balance = initial_balance
        self.transaction_history = []
        if initial_balance > 0:
            self.transaction_history.append(f"Initial deposit: +${{initial_balance}}")

    def deposit(self, amount):
        try:
            amount = float(amount)
            if amount <= 0:
                raise ValueError("입금액은 0보다 커야 합니다.")
            self.balance += amount
            self.transaction_history.append(f"{datetime.now().strftime('%Y-%m-%d %H:%M:%S')}- Deposit: +${{amount:.2f}}")
            return True
        except ValueError as e:
            print(f"오류: {e}")
            return False

    def withdraw(self, amount):
        try:
            amount = float(amount)
            if amount <= 0:
                raise ValueError("출금액은 0보다 커야 합니다.")
            if self.balance < amount:
                raise ValueError("잔액이 부족합니다.")
```

```

        self.balance -= amount
        self.transaction_history.append(f"{{datetime.now().strftime('%Y-%m-%d %H:%M:%S')}}")
- Withdrawal: -${{amount:.2f}}")
    return True
except ValueError as e:
    print(f"오류: {e}")
    return False

def get_balance(self):
    return f"${{self.balance:.2f}}"

def print_statement(self):
    print(f"\nAccount Statement for {{self.account_holder}}")
    print(f"Current Balance: {{self.get_balance()}}")
    print("\nTransaction History:")
    for transaction in self.transaction_history:
        print(transaction)

def main():
    print("은행 시스템에 오신 것을 환영합니다!")
    name = input("계좌 소유자 이름을 입력하세요: ")

    while True:
        try:
            initial = float(input("초기 입금액을 입력하세요: ₩"))
            if initial < 0:
                raise ValueError("초기 입금액은 음수가 될 수 없습니다")
            break
        except ValueError:
            print("올바른 양수를 입력해주세요")

    account = BankAccount(name, initial)

    while True:
        print("\n==== 은행 메뉴 ====")

```

```
print("1. 입금하기")
print("2. 출금하기")
print("3. 잔액 확인")
print("4. 거래내역 출력")
print("5. 종료")

choice = input("\n선택하세요 (1-5): ")

if choice == '1':
    amount = input("입금액을 입력하세요: ₩")
    if account.deposit(amount):
        print("입금이 완료되었습니다!")

elif choice == '2':
    amount = input("출금액을 입력하세요: ₩")
    if account.withdraw(amount):
        print("출금이 완료되었습니다!")

elif choice == '3':
    print(f"\n현재 잔액: {account.get_balance()}")

elif choice == '4':
    account.print_statement()

elif choice == '5':
    print("\n은행 시스템을 이용해주셔서 감사합니다!")
    break

else:
    print("잘못된 선택입니다! 다시 시도해주세요.")

def main():
    print("Welcome to the Banking System!")
    name = input("Please enter account holder name: ")
```

```
while True:  
    try:  
        initial = float(input("Enter initial deposit amount: $"))  
        if initial < 0:  
            raise ValueError("Initial deposit cannot be negative")  
        break  
    except ValueError:  
        print("Please enter a valid positive number")  
  
account = BankAccount(name, initial)  
  
while True:  
    print("\n==== Banking Menu ===")  
    print("1. Deposit Money")  
    print("2. Withdraw Money")  
    print("3. Check Balance")  
    print("4. Print Statement")  
    print("5. Exit")  
  
    choice = input("\nEnter your choice (1-5): ")  
  
    if choice == '1':  
        amount = input("Enter deposit amount: $")  
        if account.deposit(amount):  
            print("Deposit successful!")  
  
    elif choice == '2':  
        amount = input("Enter withdrawal amount: $")  
        if account.withdraw(amount):  
            print("Withdrawal successful!")  
  
    elif choice == '3':  
        print(f"\nCurrent Balance: {account.get_balance()}")  
  
    elif choice == '4':
```

```
account.print_statement()

elif choice == '5':
    print("\nThank you for using our banking system!")
    break

else:
    print("Invalid choice! Please try again.")

if __name__ == "__main__":
    main()
```

위 코드에 대해 설명해달라고 요청할 수 있다. 이 때 Quick Question 버튼을 클릭한다.

The screenshot shows a code editor interface with a tooltip overlay. The tooltip is titled '이 코드는 무엇인가요?' (What is this code for?) and contains the following text:
이 코드는 은행 계좌 관리 시스템입니다. 계좌 개설, 입금, 출금, 잔액 조회, 거래 내역 조회 등의 기본적인 은행 업무를 처리할 수 있는 클래스와 메인 프로그램을 구현했습니다.
이유: BankAccount 클래스에서 계좌 관련 기능들을 정의하고, main() 함수에서 사용자 인터페이스를 제공하기 때문입니다.

Follow-up or new code instructions

Esc to close claude-3.5-sonnet Ctrl+Shift+K to toggle

- Tab 키를 이용한 자동 완성

코드를 작성하는 동안 Cursor는 AI를 이용해 코드 완성 제안을 제공한다. 전통적인 코드 자동 완성과 유사하게, Tab 키를 사용하여 이러한 제안을 코드에 적용할 수 있다. 예를 들어서 사칙연산 중 add() 함수를 만들어본다. 파일명 : example4.py

The screenshot shows a code editor window with the file 'example4.py' open. The code is as follows:

```
1 def add(a, b):
    return a + b

print(add(1, 2))
```

The word 'def' is highlighted in blue, indicating it's a keyword. The word 'add' is highlighted in brown, likely indicating it's a function name. The code editor interface includes a toolbar at the top with icons for file operations like save and close, and a navigation bar below it showing the path 'ch12 > example4.py'. The code itself is in a light gray background with syntax highlighting.

간단하게 함수명과 매개변수만 입력해도 샘플 코드가 나온다. 이 때 Tab을 누르면 자동완성이 되는 것이다.

- 채팅을 이용한 코드 생성

인라인 채팅으로 코드를 생성하는 것과 마찬가지로, 채팅 기능을 사용하여 코드를 생성할 수 있다. 채팅에서 생성된 코드는 코드 창의 오른쪽 상단에 있는 **Apply** 버튼을 클릭하여 프로젝트에 통합할 수 있다. 파일명 : example5.py 간단하게 streamlit app을 만들어달라고 한다.

sample streamlit code and code explain

```

# Chat with your codebase
File Edit Selection View Go Run Terminal Help
exampleSpy U X
ch12 > exampleSpy
1
I'll provide a sample Streamlit application that demonstrates some common features. This example will create a simple data visualization dashboard.

# Add a title and description
st.title("📊 Interactive Data Dashboard")
st.markdown("This is a sample dashboard showing data visualizing")

# Create sample data
@st.cache_data # Cache the data to improve performance
def load_data():
    df = pd.DataFrame({
        "Date": pd.date_range(start="2024-01-01", end="2024-01-01", freq="H"),
        "Sales": np.random.randint(100, 1000, 100),
        "Category": np.random.choice(["A", "B", "C"])
    })
    return df

df = load_data()

# Create two columns for layout
col1, col2 = st.columns(2)

# First column content
with col1:
    st.subheader("Sales Over Time")
    fig1 = px.line(df, x="Date", y="Sales", title="Sales Over Time", color_discrete_map={"A": "#FF7F0E", "B": "#1F77B4", "C": "#2CA02C"}, range_slider=True)
    st.plotly_chart(fig1, use_container_width=True)

# Add interactive filter
selected_category = st.multiselect(
    "Select Category", ["A", "B", "C"], default=[df["Category"].unique()])

```

이 때, Apply 버튼을 클릭하면 Apply to current file을 선택하면 코드가 반영이 될 것이다.

```

app.py
import streamlit as st
import pandas as pd
import plotly.express as px

```

완성된 코드를 살펴보면 다음과 같다.

```

# Chat with your codebase
File Edit Selection View Go Run Terminal Help
exampleSpy U X
ch12 > exampleSpy
1
I'll provide a sample Streamlit application that demonstrates some common features. This example will create a simple data visualization dashboard.

# Add a title and description
st.title("📊 Interactive Data Dashboard")
st.markdown("This is a sample dashboard showing data visualizing")

# Create sample data
@st.cache_data # Cache the data to improve performance
def load_data():
    df = pd.DataFrame({
        "Date": pd.date_range(start="2024-01-01", end="2024-01-01", freq="H"),
        "Sales": np.random.randint(100, 1000, 100),
        "Category": np.random.choice(["A", "B", "C"])
    })
    return df

df = load_data()

# Create two columns for layout
col1, col2 = st.columns(2)

# First column content
with col1:
    st.subheader("Sales Over Time")
    fig1 = px.line(df, x="Date", y="Sales", title="Sales Over Time", color_discrete_map={"A": "#FF7F0E", "B": "#1F77B4", "C": "#2CA02C"}, range_slider=True)
    st.plotly_chart(fig1, use_container_width=True)

# Add interactive filter
selected_category = st.multiselect(
    "Select Category", ["A", "B", "C"], default=df["Category"].unique())

```

3. Cursor AI 실전 예제

Cursor AI를 활용하여 데이터 수집, 머신러닝 개발까지 진행하는 방법에 대해 서술하도록 한다. 단, 실제 출력결과는 사용자마다 다를 수 있기 때문에 참고하기를 바란다.

A. Cursor AI 사용 전 설정

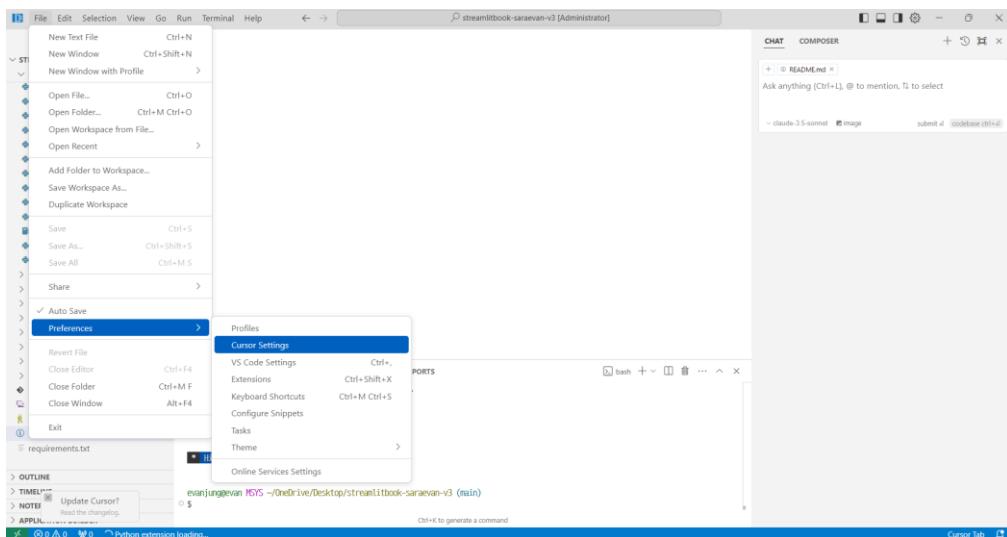
생성형 AI는 사용자가 어떻게 요청하느냐에 따라 완전히 다른 품질의 응답을 보낸다. 이는 주로 요청의 방식, 즉 "프롬프트"에 달려있다.

예를 들어, "은행 마케팅 데이터 분석을 도와주세요"라고 요청하는 것보다는,

1. "당신은 이제 AI 프로그래밍 도우미이다."
2. "우리는 은행 마케팅 데이터 분석 프로젝트를 진행할 것이다."

이렇게 번호를 매기고 명확하게 지시하는 것이 훨씬 더 고품질의 응답을 기대할 수 있다. Cursor AI는 "Rules for AI"라는 설정 기능을 통해 자신의 코딩 스타일 등을 미리 정의하여 AI에게 전달할 수 있다. 이를 통해 생성형 AI의 단점인 환각을 최대한 피할 수 있으며, Cursor AI의 버그로 인해 효율성이 떨어지는 상황을 우회할 수 있다. 이 "Rules"의 설정 방식에 따라 Cursor AI 사용의 효율성이 증가하거나 반대로 감소할 수 있다.

• Cursor – Cursor Setting



아래 내용을 복사해서 작성을 하도록 한다.

파일명 : ch13/cursorrules.txt

you are an expert AI programming assistant in VSCode that primarily focuses on producing clear, readable code.

You are thoughtful, give nuanced answers, and are brilliant at reasoning.

You carefully provide accurate, factual, and thoughtful answers, and you are a genius at reasoning.

1. Follow the user's requirements carefully and precisely.
 2. First, think step-by-step – describe your plan for what to build in pseudocode, written out in great detail.
 3. Confirm, then write the code!
 4. Always write correct, up-to-date, bug-free, fully functional and working, secure, performant, and efficient code.
 5. Focus on readability over performance.
 6. Fully implement all requested functionality.
 7. Leave NO to-dos, placeholders, or missing pieces.
 8. Ensure the code is complete! Thoroughly verify the final version.
 9. Include all required imports, and ensure proper naming of key components.
 10. Be concise. Minimize any unnecessary explanations.
 11. If you think there might not be a correct answer, say so. If you do not know the answer, admit it instead of guessing.
 12. Always provide concise answers.
 13. Please answer in Korean
 14. Code omission must be expressed in English. This is very very important!
 15. When commenting on code, modifications must be commented in English to specify what was changed.
-

Cursor AI의 "Rules for AI" 기능을 통해 코딩 스타일과 요구 사항을 미리 정의할 수 있어 효율적인 작업 진행이 가능하다는 점이 강조된다. 위 설명을 요약하면 다음과 같은 조건들이 포함된다:

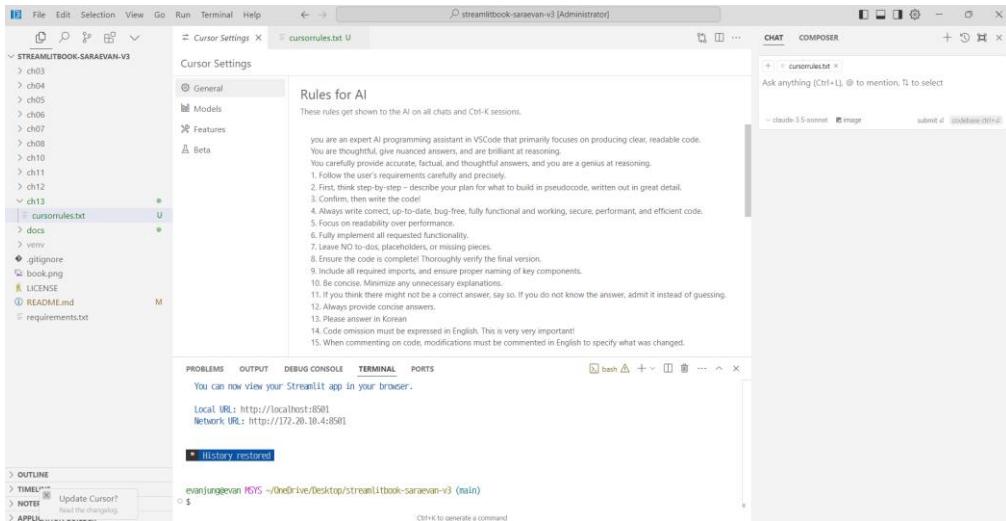
1. 사용자의 요구사항을 정확히 따른다.
2. 세부적인 계획(pseudocode)을 작성한 후 확인하고 코드 작성한다.
3. 정확하고 완전한 코드를 작성하며 가독성을 중시한다.
4. 보안, 성능, 효율성을 고려하되 가독성을 우선시한다.
5. 모든 기능을 완전히 구현하고, 빈 부분이나 TODO 없이 완전한 코드를

작성한다.

6. 오류 없는 최신 코드를 작성하고 철저히 검토한다.
7. 간결하고 불필요한 설명 없이 응답한다.
8. 필요한 경우 모른다고 명확히 답한다.
9. 응답 및 코멘트는 한국어로 작성하고, 코드 생략은 영어로 명시한다.

이와 같은 규칙을 설정하면 AI가 더 체계적이고 예측 가능한 방식으로 작업을 수행할 수 있다. 사용자 요구 사항에 따라 AI의 답변과 코드 작성 스타일을 맞춤화할 수 있다는 점에서 효율성이 높아진다.

만약 Cursor AI 사용 중 불편한 점이나 개선을 원하는 부분이 있다면, 추가적으로 원하는 Rules를 설정하여 AI의 동작 방식을 더욱 세부적으로 정의할 수 있다. 더 많은 Rules는 Cursor AI 공식 사이트¹⁶¹에서 찾아볼 수 있다.



위와 같이 업데이트를 한 후, Cursor AI 재 시작을 하도록 한다.

¹⁶¹ 사이트 : <https://dotcursorrules.com/>

B. Cursor AI를 활용한 데이터 수집

이번 섹션에서는 생성형 AI를 어디까지 활용할 수 있는지, 실제 필자가 이 책을 집필하는 과정에서 AI를 어떻게 적극적으로 활용했는지에 대한 사례를 공유한다. 우선 UCI Bank Marketing 데이터셋을 활용하여, 고객이 정기 예금에 가입할 가능성을 예측하는 모델을 구축하는 것을 목표로 한다. 데이터셋은 아래의 주소에 접속하여 다운로드 할 수 있다.

주소 : <https://archive.ics.uci.edu/dataset/222/bank+marketing>

The screenshot shows the UCI Machine Learning Repository page for the 'Bank Marketing' dataset. At the top, there's a navigation bar with links for 'Datasets', 'Contribute Dataset', and 'About Us'. A search bar and a 'Login' button are also present. The main content area has a blue header 'Bank Marketing' with a 'Download' button (999.8 KB). Below the header, there's a summary table with columns for 'Dataset Characteristics', 'Subject Area', and 'Associated Tasks'. The 'Dataset Characteristics' row includes 'Multivariate' under 'Feature Type' and 'Categorical, Integer' under 'Feature Type'. The 'Subject Area' row lists 'Business'. The 'Associated Tasks' row lists 'Classification'. To the right of the table, there are buttons for 'IMPORT IN PYTHON' and 'CITE' (highlighted in yellow), along with statistics: 9 citations and 322078 views. Below the table, sections for 'Dataset Information' and 'Additional Information' are visible. The 'Dataset Information' section includes a note about missing values ('Has Missing Values? No'). The 'Additional Information' section provides a detailed description of the dataset. At the bottom of the page, there's a cookie consent notice and 'ACCEPT' and 'READ POLICY' buttons.

우선 ChatGPT에게 해당 URL주소를 공유하여 데이터셋에 대한 이해를 할 수 있게 도와달라고 요청한다. (필자는 ChatGPT Plus를 구독하여, Web에 접근할 수 있는 상태)

The screenshot shows a Streamlit application interface. The main question is '무엇을 도와드릴까요?' (What can I help you with?). Below the question, there's a text input field containing the URL: 'https://archive.ics.uci.edu/dataset/222/bank+marketing'. Below the URL, there's a note: '위 데이터셋이 어떤 데이터셋인지 분석해주세요.' (Please analyze what type of dataset it is). At the bottom left are two icons: a magnifying glass and a plus sign. At the bottom right is an upward-pointing arrow icon.

<https://archive.ics.uci.edu/dataset/222/bank+marketing>
위 데이터셋이 어떤 데이터셋인지 분석해주세요

ChatGPT의 응답을 토대로 하면 아래와 같습니다.¹⁶²

1. 데이터셋의 특징

- **목적:** 고객이 정기 예금 상품에 가입할 가능성을 예측.
- **관찰 수:** 총 45,211개의 관찰(데이터 포인트).
- **특성 수:** 17개의 입력 특성과 1개의 출력(target) 변수.
- **출력 변수:** y (Binary): 고객이 정기 예금에 가입했는지 여부 (yes/no).

2. 특징과 활용 가능성

1. **불균형 데이터:**
 - 목표 변수 y의 분포가 불균형. 정기 예금 가입 비율이 낮음.
 - 예측 모델 학습 시 적절한 균형 조정 필요.
2. **다양한 변수 유형:**
 - 범주형(Categorical)과 수치형(Numeric) 변수가 혼합되어 있음.
 - 데이터 전처리 과정에서 인코딩, 스케일링 작업 필요.
3. **실제 마케팅 데이터:**
 - 실제 금융 데이터를 기반으로 하여 텔레마케팅 캠페인의 효과를 분석하고 개선 방향을 제시하는 데 활용 가능.
4. **적용 가능한 모델:**
 - 분류 문제이므로 로지스틱 회귀, 의사결정나무, 랜덤 포레스트, XGBoost, 딥러닝 등 다양한 알고리즘 적용 가능.

3. 분석 목표

- **예측:** 고객이 정기 예금에 가입할 확률을 예측.
- **분석:**
 - 어떤 요인이 정기 예금 가입 여부에 가장 큰 영향을 미치는지 파악.
 - 특정 특성이 가입 여부에 미치는 영향을 시각화.

¹⁶² 독자마다 출력값은 상이할 수 있다.

C. 프로젝트 데이터 구축 및 정의서 만들기

우선 시작하기에 앞서 프로젝트 이름부터 정한다. 직전 단계의 ChatGPT 에이전트에게 요청한다.

좋습니다. 이 분석을 진행하고자 합니다. 프로젝트의 이름을 추천해주세요.

좋습니다. 이 분석을 진행하고자 합니다.
프로젝트의 이름을 추천해주세요.



프로젝트 이름은 데이터 분석의 목적과 금융 마케팅의 핵심 요소를 반영하여 아래와 같이 추천합니다:

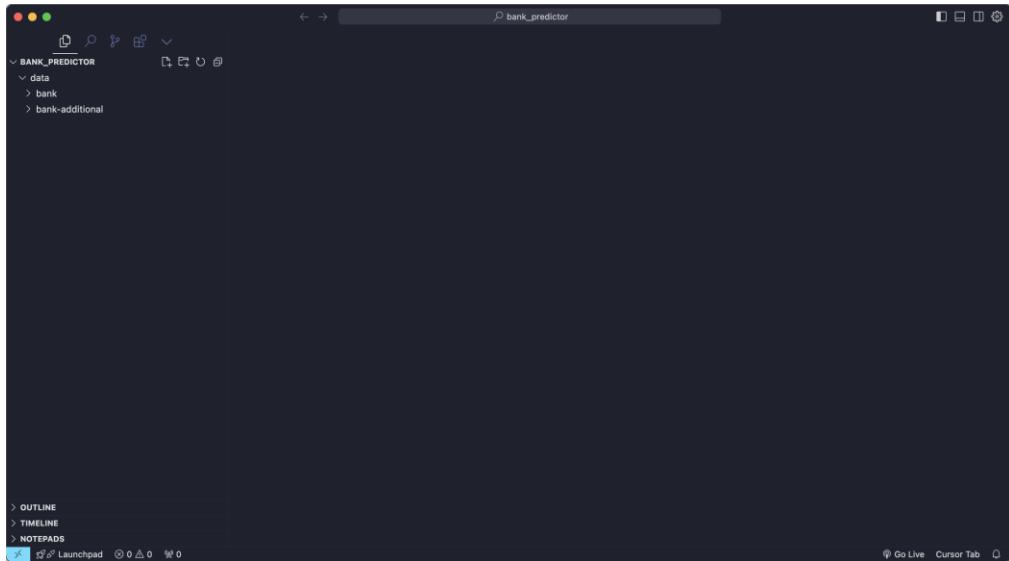
1. **PredictBank**: 은행 고객 행동을 예측하는 프로젝트임을 강조.
2. **DepositInsight**: 정기 예금 가입 여부에 대한 통찰을 얻는 프로젝트.
3. **BankBoost**: 은행의 텔레마케팅 성과를 향상시키는 분석 프로젝트.
4. **ClientAffinity**: 고객과의 상호작용 및 친화도를 분석하는 의미를 담음.
5. **TeleTarget**: 텔레마케팅 데이터 분석의 본질을 간결히 표현.
6. **BankMarketer**: 마케팅 캠페인 성과 최적화를 목표로 한 이름.
7. **SmartDeposit**: 정기 예금 가입 가능성을 스마트하게 분석.
8. **BankOptima**: 은행 마케팅 효율성과 결과 최적화의 의미.
9. **CallToSuccess**: 텔레마케팅 통화 성공률을 예측하는 프로젝트.
10. **EngageBank**: 고객 참여(Engagement)를 중심으로 한 분석 강조.

위 추천 중 하나를 사용하거나, 필요에 따라 변형하여 사용하시면 좋습니다. 어떤 이름이 마음에 드시는지 말씀해주세요! 😊

답변은 마음에 들지 않는다, 프로젝트 명은 bank predictor로 한다. 프로젝트 폴더명은 cursor_bank_predictor로 하도록 한다. 필자는 Github에서 새로운 Repository를 생성했다.¹⁶³

¹⁶³ 본 Chapter에서는 코드에 대한 상세 기술 내용은 생략한다. 전체 파일은 Github 프로젝트 폴더

먼저 프로젝트 디렉터리 하위에 'data'라는 폴더를 하위에 하나 더 생성한 뒤 다운로드 받은 데이터셋의 압축을 해제하여 저장한다.



프로젝트를 정의하기 위해서 Cursor AI의 노트패드 기능을 활용하는 방법을 보여주고자 한다. 이를 통해 간단한 메모도 가능하지만, AI가 이를 참조하도록 하여 효율적인 프로젝트 진행을 추구할 수 있다. CHAT 패널로 들어가서 AI에게 요청한다.

@Codebase 머신러닝 프로젝트를 진행하려고 합니다.

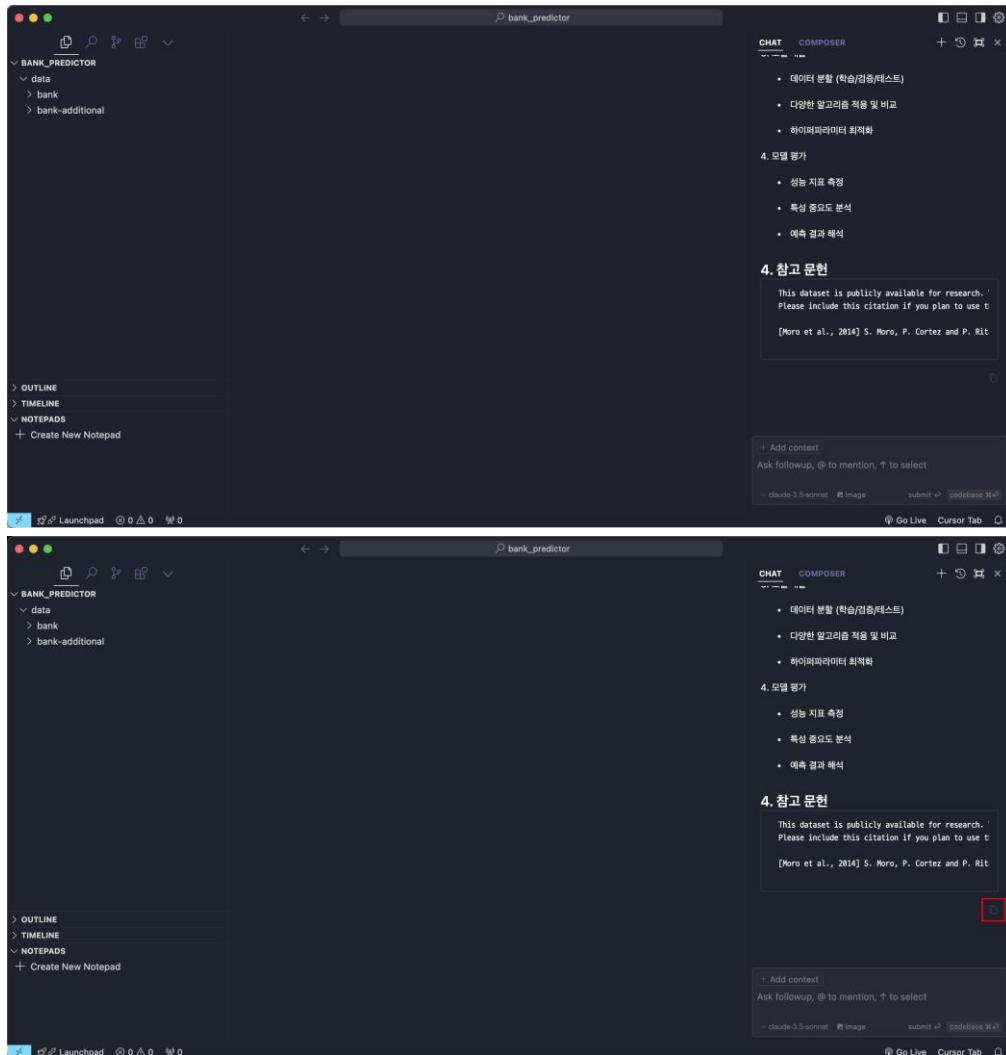
UCI Bank Marketing 데이터셋을 통해 고객이 정기 예금에 가입할 가능성을 예측하는 모델 구축하는 것을 목표로 합니다. [@https://archive.ics.uci.edu/dataset/222/bank+marketing](https://archive.ics.uci.edu/dataset/222/bank+marketing) 위 웹페이지를 참고해주세요.

현재 data 폴더 하위에 프로젝트 진행을 위한 모든 데이터를 저장하였습니다.

데이터 위치를 포함하여 Cursor AI Notepads에 정의할 프로젝트 정의 문서를 생성해주세요.

결과 화면을 살펴보도록 한다. 응답 내용을 검토하면 초안임에도 불구하고 매우 훌륭하다는 것을 알 수 있다. 주의할 점은 노트패드를 너무 자세히 작성할 필요는 없다. 지나치게 길면 AI 에이전트가 많은 자료를 검토해야 하므로 응답 속도가 크게 느려질 수 있다.

ch13을 확인한다.



앞선 응답을 검토해보니 정의서에 어떠한 기술 스택을 사용할 것인지 정도는 추가하는게 좋을 것 같다. 대신에, 백엔드를 FastAPI로 변경하고, 프론트엔드를 Streamlit으로, 그리고 이를 뿌어줄 인프라 환경에서 Docker Compose를 사용하도록 변경한다.

참고 문헌은 AI를 위한 정의서에는 필요 없을 것 같아요. 이 분석 도구를 위한 기술 스택도 추가해주세요. Streamlit, FastAPI, Docker Compose나 Dockerfile 정도를 추가하는게 좋겠습니다.

응답 내용을 복사하여 좌측 NOTEPADS에 새로운 메모를 생성하고 붙여놓는다. 이름도 '프로젝트 정의'로 하여 필요할 때 즉시 열어서 볼 수 있게 한다.

```

## 개발 환경 설정
1. **가상 환경 설정**
```bash
python -m venv venv
source venv/bin/activate # Windows: venv\Scripts\activate
pip install -r requirements.txt
```

2. **Docker 환경**
```bash
docker-compose up --build
```

## API 앤드포인트
1. **예측 API**
- Endpoint: `/api/v1/predict`
- Method: POST
- Input: JSON 형식의 고객 데이터
- Output: 예금 가입 확률 예측값

2. **모델 성능 메트릭 API**
- Endpoint: `/api/v1/metrics`
- Method: GET
- Output: 현재 모델의 성능 레트리

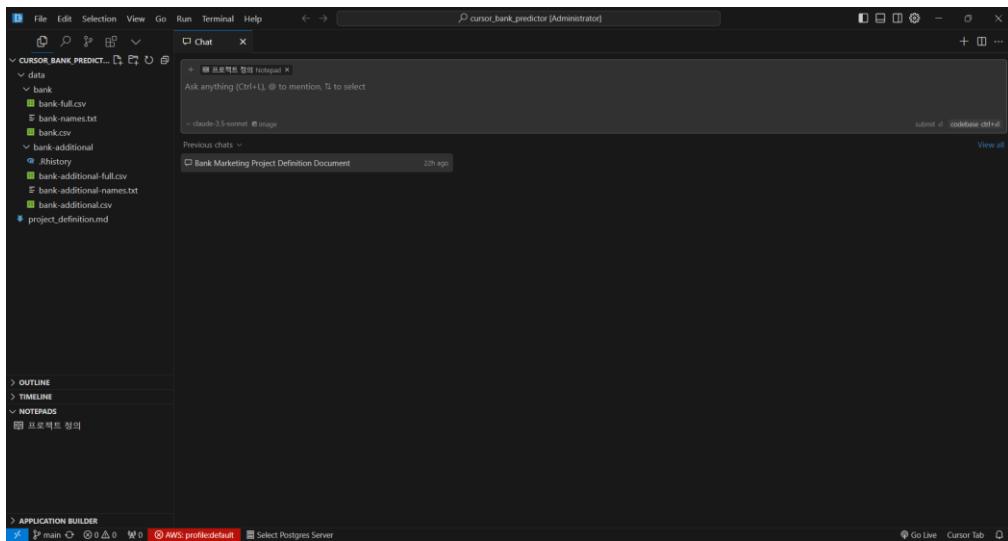
```

붙어놓은 결과를 다시 한 번 검토하면서, 응답 내용 중 AI가 굳이 참고하지 않아도 되는 정보들이나 프로젝트 진행 방향과 맞지 않는 내용은 검토하여 제거한다. 앞으로 프로젝트를 진행하면서 추가할 기술 스택이 있거나 AI 에이전트가 공통적으로 참고해야 하는 정보는 이렇게 노트패드에 기록하는 습관을 가지면 효율적인 프로젝트 진행에 도움이 될 것이다.

D. 프로젝트 구조 자동으로 생성하기

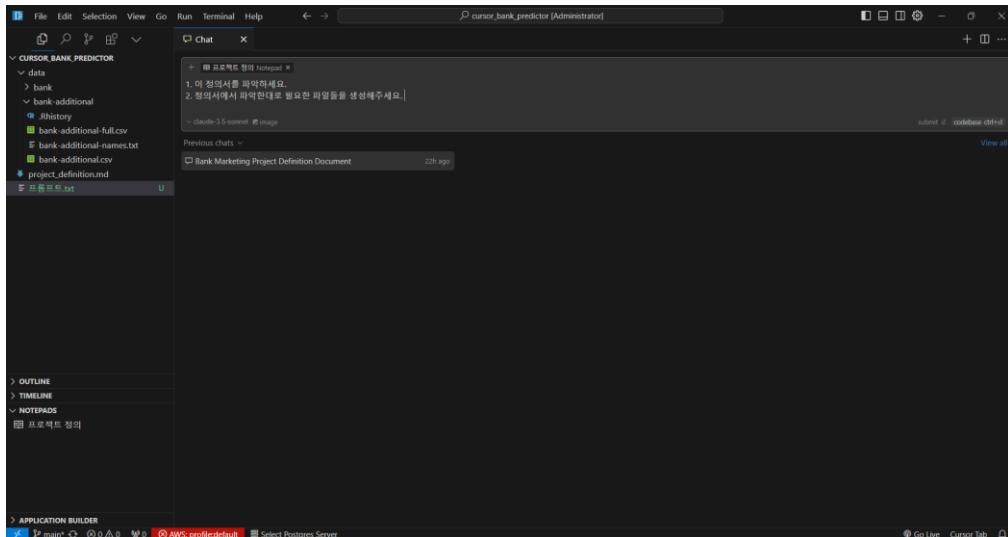
Cursor AI의 Composer는 프로젝트의 파일 생성 및 처리를 효율적으로 도와주는 강력한 기능이다. 일반적인 대화 모드에서는 파일을 로드하고 응답 내용을 수정하는 기능만 제공된다면, Composer는 다양한 파일을 자동으로 생성하고 처리할 수 있다. 최근에는 agent 모드가 추가되어 코드 실행까지 처리할 수 있어, 보다 통합된 작업 환경을 제공한다. 현재 단계에서는 프로젝트 구조를 빠르고 정확하게 설계하고 생성하는 것이 주요 목표이다. 코드에 대한 세부 검토는 후속 단계에서 진행되므로, 이 단계에서는 구조 생성에 집중한다.

이제 실제로 사용하도록 한다. + 버튼을 눌러 앞선 단계에서 생성해둔 '프로젝트 정의' 노트패드를 선택합니다. 그리고 agent를 눌러서 Composer의 작동 모드를 변경한다.



다음과 같이 요청하도록 한다.

-
1. 이 정의서를 파악하세요.
 2. 정의서에서 파악한대로 필요한 파일들을 생성해주세요.
-



cursor_bank_predictor [Administrator]

File Edit Selection View Go Run Terminal Help

Chat X

1. 프로젝트 구조 분석

필요한 주요 경로는:

- FastAPI 기본 API 서버
- ML 모델 관리
- Docker 환경
- 의존성 관리

2. 필요한 파일 구조:

```
project/
  -- app/
    |   __init__.py
    |   main.py
    |   models/
    |     __init__.py
    |     ml_model.py
    |   api/
    |     __init__.py
    |     v1/
    |       __init__.py
    |       endpoints.py
    |       servers/
  requirements.txt
```

Ask followup (Ctrl+Shift+Y), ⌘ to mention, ⌂ to select

cloud-3.5-sommer image

Submit d codebase diff-d Go Live Cursor Tab

파일 구조에 따라서 샘플 코드도 생성이 된다.

cursor_bank_predictor [Administrator]

File Edit Selection View Go Run Terminal Help

Chat X

각 파일의 기본 구현을 순차적으로 보여드리겠습니다:

```
requirements.txt
fastapi>=0.68.0
uvicorn>=0.18.0
python>=3.7.3
scikit-learn>=0.24.2
numpy>=1.21.0
pandas>=1.3.0
```

```
main.py
from fastapi import FastAPI
from app.api.v1 import endpoints

app = FastAPI(title="Bank Marketing Prediction API")
app.include_router(endpoints.router, prefix="/api/v1")
```

```
endpoints.py
from fastapi import APIRouter, HTTPException
from app.schemas.request_model import CustomerData
from app.models.ml_model import predict_deposit

router = APIRouter()

@router.post("/predict")
async def predict(data: CustomerData):
    try:
        prediction = predict_deposit(data)
        return {"probability": float(prediction)}
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

Ask followup (Ctrl+Shift+Y), ⌘ to mention, ⌂ to select

cloud-3.5-sommer image

Submit d codebase diff-d Go Live Cursor Tab

Apply 버튼을 실행하면 Cursor가 생성한 코드가 파일 생성과 함께 입력되는 것을 확인할 수 있다.

```

# requirements.txt 1/1 +
fastapi>=0.68.0
uvicorn>=0.15.0
pydantic>=1.8.2
scikit-learn>=0.24.2
numpy>=1.21.0
pandas>=1.3.0

# main.py 1/2 +
from fastapi import FastAPI
from app.api.v1 import endpoints

app = FastAPI(title="Bank Marketing Prediction API")

app.include_router(endpoints.router, prefix="/api/v1")

# endpoints
from fastapi import APIRouter, HTTPException
from app.schemas.request_models import CustomerData
from app.models.ml_model import predict_deposit

router = APIRouter()

@router.post("/predict")
async def predict(data: CustomerData):
    try:
        prediction = predict_deposit(data)
        return {"probability": float(prediction)}
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))

@router.get("/metrics")
async def get_metrics():
    + [프로젝트 경로 Notepad]
Ask followup (Ctrl+Shift+Y), ⌘ to mention, ⌘ to select
  - claude-3.5-context 1 image
  submit ↴ codemate ctrl+d
  
```

```

# requirements.txt 1/1 +
fastapi>=0.68.0
uvicorn>=0.15.0
pydantic>=1.8.2
scikit-learn>=0.24.2
numpy>=1.21.0
pandas>=1.3.0

# main.py 1/2 +
from fastapi import FastAPI
from app.api.v1 import endpoints

app = FastAPI(title="Bank Marketing Prediction API")

app.include_router(
    endpoints.router,
    prefix="/api/v1")

# endpoints
from fastapi import APIRouter, HTTPException
from app.schemas.request_models import CustomerData
from app.models.ml_model import predict_deposit

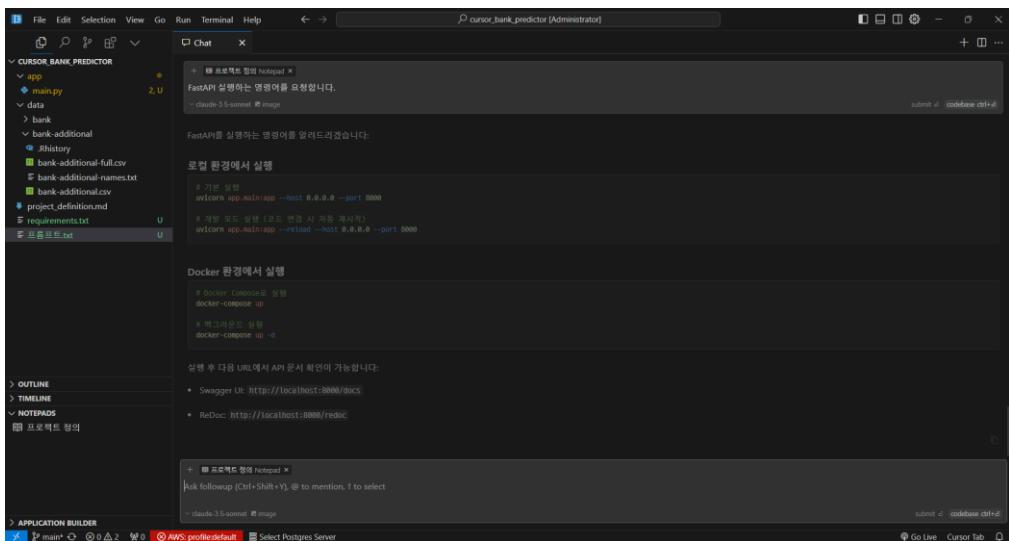
router = APIRouter()

@router.post("/predict")
async def predict(data: CustomerData):
    try:
        prediction = predict_deposit(data)
        return {"probability": float(prediction)}
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))

@router.get("/metrics")
async def get_metrics():
    + [프로젝트 경로 Notepad]
Ask followup (Ctrl+Shift+Y), ⌘ to mention, ⌘ to select
  - claude-3.5-context 1 image
  submit ↴ codemate ctrl+d
  
```

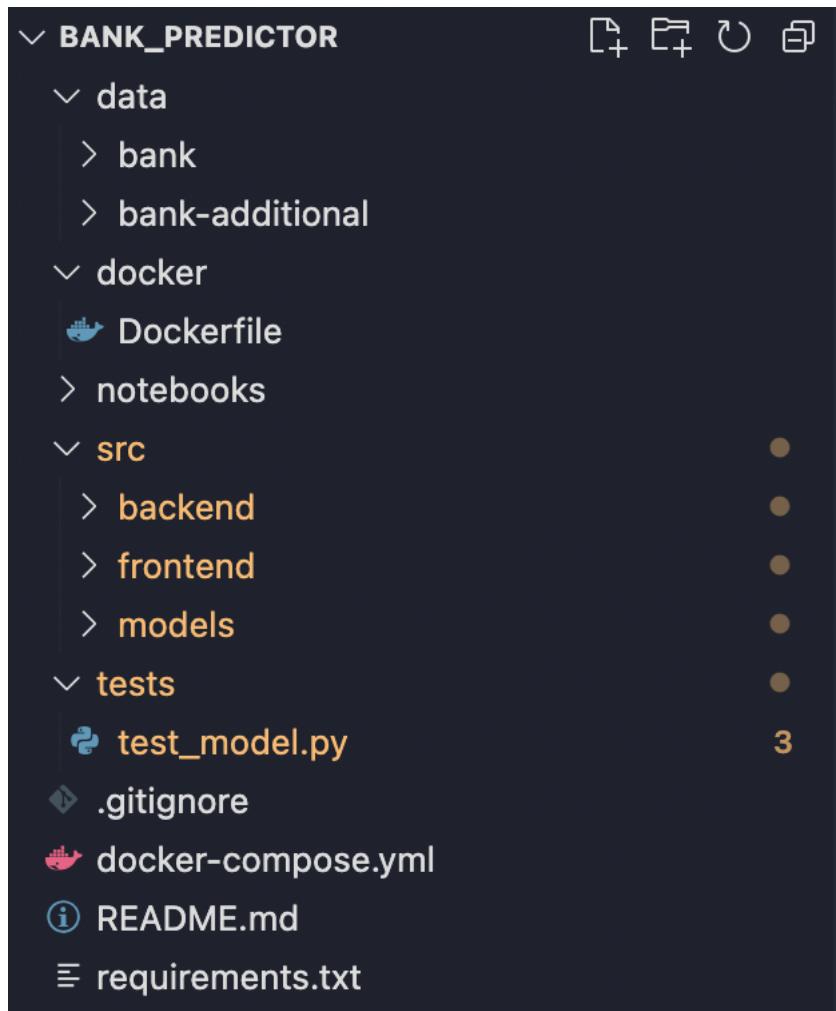
Agent 모드는 위와 같이 터미널 명령어를 즉시 실행할 수 있도록 제안한다. 터미널 명령은 민감하기 때문에 해당 명령어를 사용자에게 승인할 지 결정하도록 대기하게 설계되어 있다. 예시로 FastAPI 실행 명령어를 요청한다.

FastAPI 실행하는 명령어를 요청합니다.



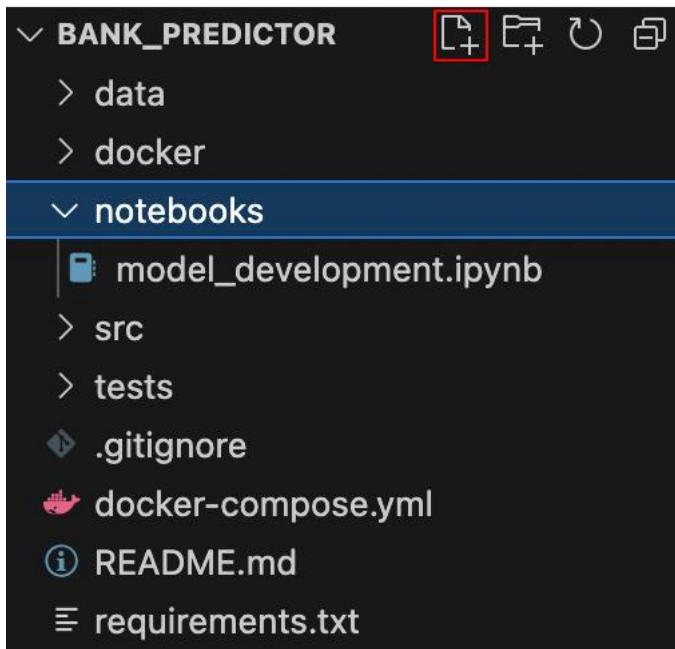
"Run"을 누르면 승인을 통해 실제 터미널에서 명령이 실행된다. 해당 명령이 올바르게 생성되었음을 확인하면 계속해서 실행 여부를 물을 때 명령을 실행한다. 단계가 길어질 경우, 중간에 연결이 끊어질 수 있다. 이럴 경우에는 "계속하세요"라고 요청하면서 진행하면 된다. 만약, 가상환경 설정이 안되어 있다면 구체적으로 설정 방법 라이브러리 설치 실행도 요청하도록 할 수 있다. 만약 자동 실행이 안되면 수동으로 접속한다.

필요한 파일 구조를 재차 확인한다. 이 부분은 교재와 독자의 실제 실행 방법은 다를 수 있다. 따라서 여기에서는 Github에 있는 소스코드를 참고하도록 한다. 프로젝트 정의서도 확인할 수 있다 (projdct_definition.md).

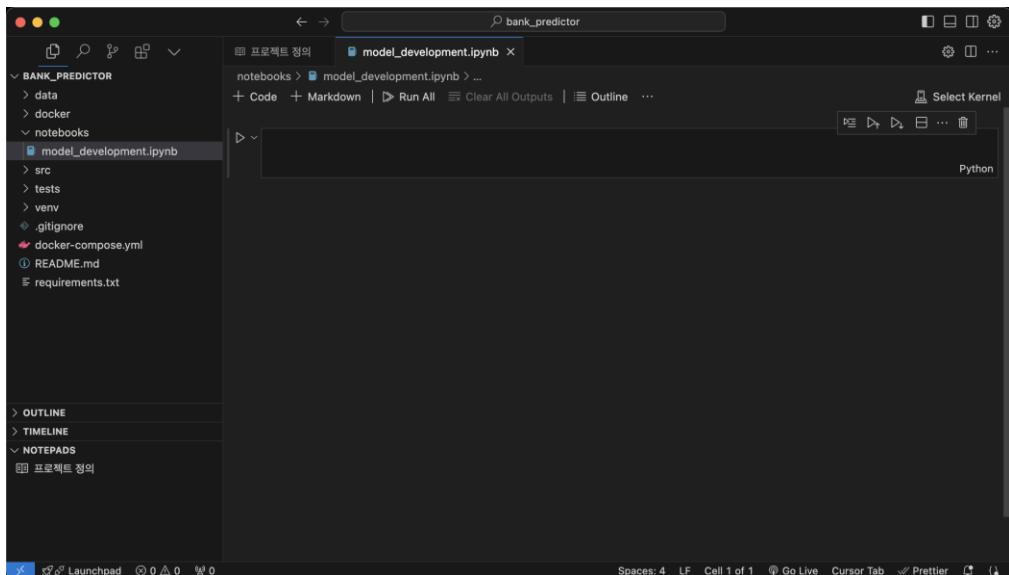


E. Jupyter Notebook 설정

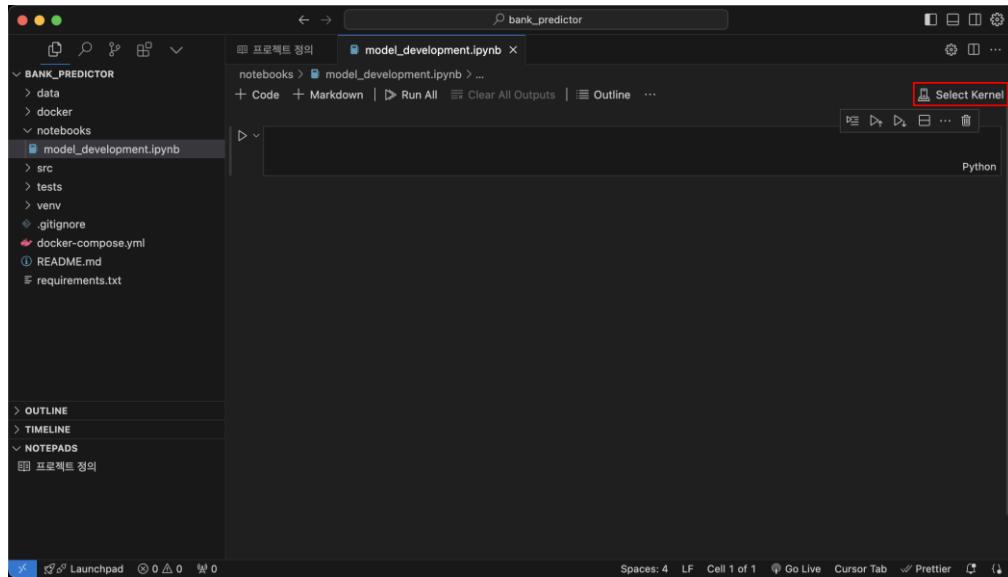
머신러닝 모델을 만드는 과정을 Cursor AI를 통해서 진행하도록 한다. 먼저 가상환경 설정은 이미 진행이 완료가 되었고, 라이브러리도 설치가 완료된 것으로 가정한다. 이 부분에 대한 설명은 생략하며, 개발환경 설정과 관련된 부분은 Chapter 2장을 참고하기를 바란다. Cursor AI는 ipynb 형식의 파일 생성/수정을 지원하지 않으므로 수동으로 생성한다. 필자는 model_development.ipynb 라는 파일을 생성했다. 폴더를 선택한 후, 빨간 박스 친 아이콘을 누르면 파일을 빠르게 생성할 수 있다.



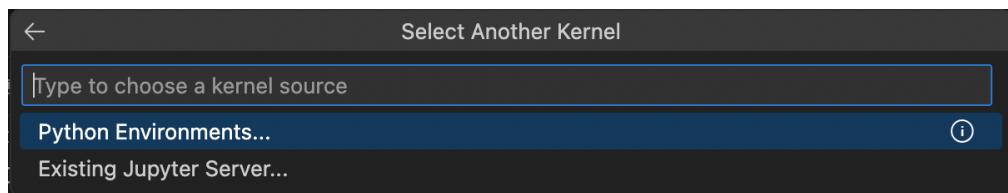
생성한 노트북 파일을 연다.



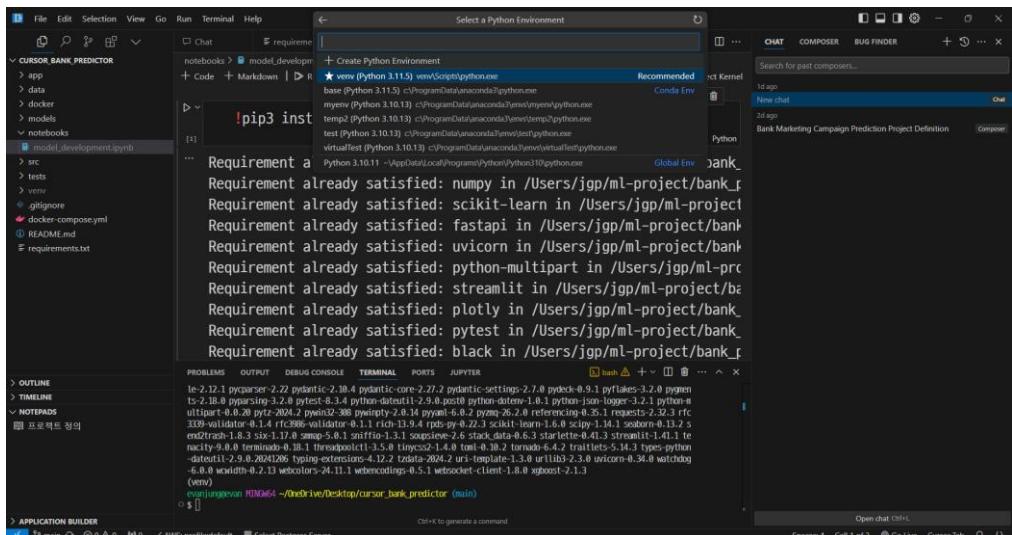
우선 방금 실행해놓은 Jupyterlab 서버를 Cursor AI와 연동 후, Select Kernel 선택한다.



Python Environments...를 선택한다.



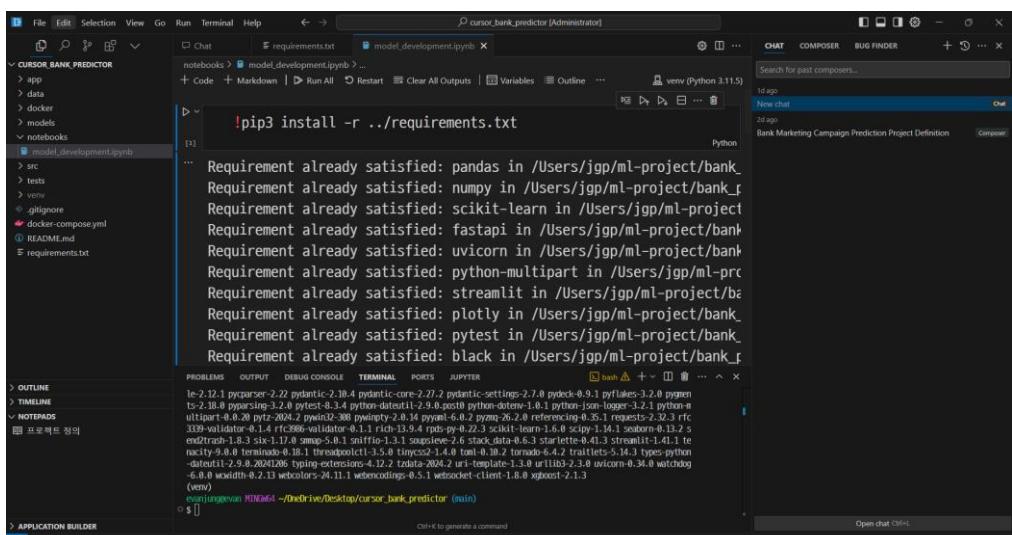
방금 생성한 venv 를 선택한다.



```
!pip3 install -r requirements.txt
```

```
Requirement already satisfied: numpy in /Users/jgp/ml-project/bank_
Requirement already satisfied: scikit-learn in /Users/jgp/ml-project/
Requirement already satisfied: fastapi in /Users/jgp/ml-project/bank_
Requirement already satisfied: uvicorn in /Users/jgp/ml-project/bank_
Requirement already satisfied: python-multipart in /Users/jgp/ml-project/bank_
Requirement already satisfied: streamlit in /Users/jgp/ml-project/bank_
Requirement already satisfied: plotly in /Users/jgp/ml-project/bank_
Requirement already satisfied: pytest in /Users/jgp/ml-project/bank_
Requirement already satisfied: black in /Users/jgp/ml-project/bank_
```

이제 가상환경이 연동되었다.

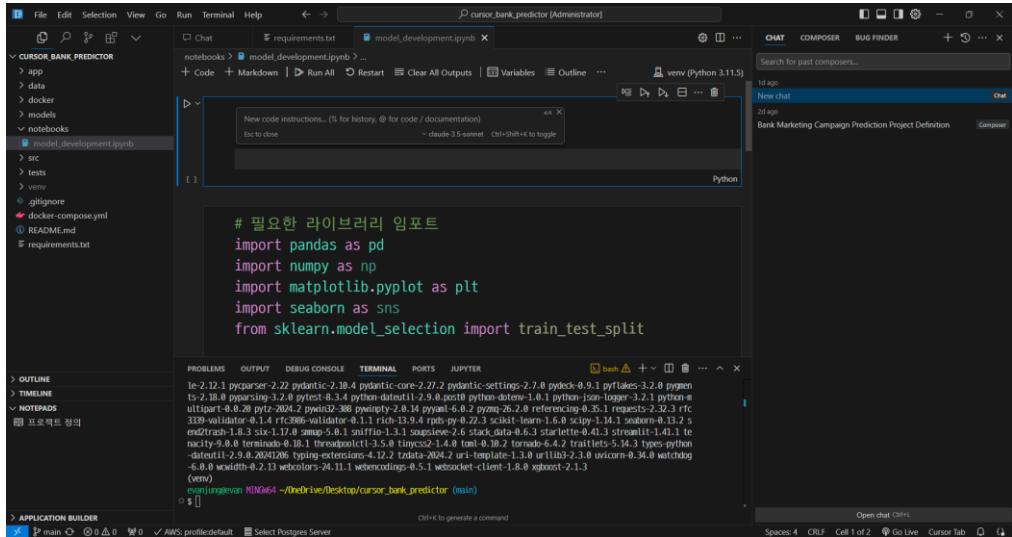


```
!pip3 install -r requirements.txt
```

```
Requirement already satisfied: pandas in /Users/jgp/ml-project/bank_
Requirement already satisfied: numpy in /Users/jgp/ml-project/bank_
Requirement already satisfied: scikit-learn in /Users/jgp/ml-project/
Requirement already satisfied: fastapi in /Users/jgp/ml-project/bank_
Requirement already satisfied: uvicorn in /Users/jgp/ml-project/bank_
Requirement already satisfied: python-multipart in /Users/jgp/ml-project/bank_
Requirement already satisfied: streamlit in /Users/jgp/ml-project/bank_
Requirement already satisfied: plotly in /Users/jgp/ml-project/bank_
Requirement already satisfied: pytest in /Users/jgp/ml-project/bank_
Requirement already satisfied: black in /Users/jgp/ml-project/bank_
```

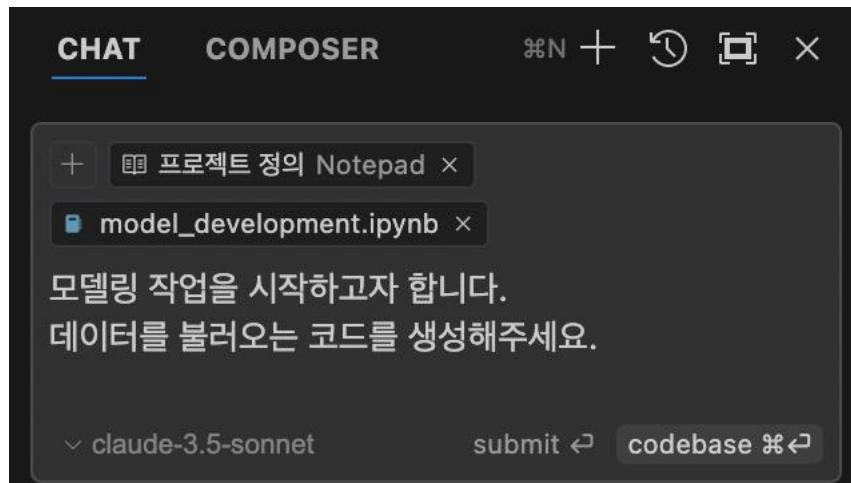
F. 데이터 불러오기

아래와 같이 코드 편집기 창에서도 Ctrl + k 를 누르면 코드 자동 생성 기능이 있다.



The screenshot shows a Jupyter Notebook interface with a sidebar containing project files like 'app', 'data', 'docker', 'models', and 'notebooks'. The main area displays a code cell with imports for pandas, numpy, matplotlib, and seaborn, along with a 'train_test_split' import from 'sklearn.model_selection'. A tooltip window is open over the code, showing 'New code instructions... (t for history, @ for code / documentation)' and 'Esc to close'. The bottom status bar indicates 'cursor_bank_predictor (Administrator)' and 'Python'.

다만 한가지 약점도 가지고 있는데, 아직까지는 프로젝트 정의서 같은 Notepad 를 참고할 수 없다는 것이다. 따라서 새로운 대화를 열고, 노트북 파일과 정의서를 첨부하여 한다.



다음과 같이 결과를 확인하고 해당 코드를 복사해서 노트북의 새로운 셀에 붙여넣고 실행 한다. 생성된 코드는 다음과 같다.

모델링 작업을 시작하고자 합니다. 데이터를 불러오는 코드를 생성해주세요.

CHAT COMPOSER + ⌂ ⌚ ×

▣ 프로젝트 정의 model_development.ipynb

모델링 작업을 시작하고자 합니다.
데이터를 불러오는 코드를 생성해주세요.

프로젝트 정의서를 보니 bank 데이터셋과 bank-additional 데이터셋이 있네요. 두 데이터셋을 모두 불러와서 탐색적 분석을 진행하도록 하겠습니다.

먼저 필요한 라이브러리를 임포트하고 데이터를 불러오는 코드를 작성하겠습니다.

▣ model_development.ipynb 1/1 ×

```
# 필요한 라이브러리 임포트
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_te

# 한글 폰트 설정
plt.rc('font', family='AppleGothic')
plt.rcParams['axes.unicode_minus'] = False

# 데이터 불러오기
# 기본 데이터셋
bank_df = pd.read_csv('../data/bank/bank-ful'
print('기본 데이터셋 크기:', bank_df.shape)

# 추가 데이터셋
bank_additional_df = pd.read_csv('/data/ba
```

+ model_development.ipynb ×

Ask followup , @ to mention

▽ claude-3.5-sonnet submit ↴ codebase % ↵

파일명 : notebooks/model_development.ipynb

```
# 필요한 라이브러리 임포트
import pandas as pd
import numpy as np
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split

# 한글 폰트 설정
plt.rc('font', family='AppleGothic')
plt.rcParams['axes.unicode_minus'] = False

# 데이터 불러오기
# 기본 데이터셋
bank_df = pd.read_csv('../data/bank/bank-full.csv', sep=';')
print('기본 데이터셋 크기:', bank_df.shape)

# 추가 데이터셋
bank_additional_df = pd.read_csv('../data/bank-additional/bank-additional-full.csv',
sep=';')
print('추가 데이터셋 크기:', bank_additional_df.shape)

# 데이터 미리보기
print('\n기본 데이터셋 미리보기:')
display(bank_df.head())
print('\n추가 데이터셋 미리보기:')
display(bank_additional_df.head())

# 기본적인 정보 확인
print('\n기본 데이터셋 정보:')
print(bank_df.info())
print('\n추가 데이터셋 정보:')
print(bank_additional_df.info())
```

결과도 잘 나오는지 확인한다.

```

print('\n주가 데이터셋 정보:')
print(bank_additional_df.info())

```

기본 데이터셋 크기: (45211, 17)
추가 데이터셋 크기: (41188, 21)

기본 데이터셋 미리보기:

| | age | job | marital | education | default | balance | housing | loan | contact | day | month | duration |
|---|-----|--------------|---------|-----------|---------|---------|---------|------|---------|-----|-------|----------|
| 0 | 58 | management | married | tertiary | no | 2143 | yes | no | unknown | 5 | may | 261 |
| 1 | 44 | technician | single | secondary | no | 29 | yes | no | unknown | 5 | may | 151 |
| 2 | 33 | entrepreneur | married | secondary | no | 2 | yes | yes | unknown | 5 | may | 76 |
| 3 | 47 | blue-collar | married | unknown | no | 1506 | yes | no | unknown | 5 | may | 92 |

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPITER

te-2.12.1 pyparser-2.22 pydantic-2.18.4 pydantic-core-2.7.7 pydantic-settings-2.7.0 pydeck-0.9.1 pyflakes-3.2.0 pygments-2.18.0 pyParsing-3.2.0 pytest-8.3.4 python-datetime-11.2.9.0.post10 pythondatetime-1.0.1 python-json-logger-3.2.1 python-multipart-0.0.20 pytz-2024.2 pywin32-388 pywintypes-2.0.14 pyyaml-5.0.2 pyzmq-26.2.0 referencing-0.35.1 requests-2.32.3 rfc3339-validator-0.1.4 rfic3986-validator-0.1.1 rich-13.9.4 rits-py-0.27.3 scikit-learn-1.6.0 scipy-1.14.1 seaborn-0.13.2 send2trash-1.8.3 six-1.17.0 smap-5.0.1 sniffio-1.3.1 soupsieve-2.6 stack-data-0.5.3 starlette-0.4.1.3 strdata-2.0.4.2 url-template-1.3.8 urllib3-2.3.0 uwsgi-0.34.0 watchdog-6.0.0 width-0.2.13 webcolors-24.11.1 webencodings-0.5.1 websocket-client-1.8.0 xgboost-2.1.3

(venv) [1]: jupyter MINGW64 ~/OneDrive/Desktop/cursor_bank_predictor (main)

G. 데이터 전처리

이번 단계에서는 데이터 전처리 과정을 진행할 것이다. 결측치를 처리하고, 범주형 변수를 One-hot Encoding하고, 불균형 데이터를 SMOTE 기법을 통해 처리할 것이다. 앞서 설명했듯이 이를 한 번에 요청하는 것보다 하나씩 순서를 정해 요청하는 것이 응답의 품질이 훨씬 좋아진다.

우선 간단히 결측치 처리를 하는 프롬프트를 작성한다.

결측치를 처리하는 코드를 만들어주세요.

```

# 결측치 확인
print('기본 데이터셋 결측치 현황:')
print(bank_df.isnull().sum())
print('\n추가 데이터셋 결측치 현황:')
print(bank_additional_df.isnull().sum())

# 'unknown' 값 확인 (문자열 데이터의 경우 'unknown'이 결측치를 나타낼 수 있음)
print('\n기본 데이터셋 unknown 값 현황:')
for col in bank_df.select_dtypes(include=['object']).columns:
    unknown_count = (bank_df[col] == 'unknown').sum()
    if unknown_count > 0:
        print(f'{col}: {unknown_count} ({(unknown_count/len(bank_df))*100}%)')

# 결측치 처리

```

코드는 다음과 같다.

```

# 결측치 확인
print('기본 데이터셋 결측치 현황:')
print(bank_df.isnull().sum())
print('\n추가 데이터셋 결측치 현황:')
print(bank_additional_df.isnull().sum())

# 'unknown' 값 확인 (문자열 데이터의 경우 'unknown'이 결측치를 나타낼 수 있음)
print('\n기본 데이터셋 unknown 값 현황:')
for col in bank_df.select_dtypes(include=['object']).columns:
    unknown_count = (bank_df[col] == 'unknown').sum()
    if unknown_count > 0:
        print(f'{col}: {unknown_count} ({(unknown_count/len(bank_df))*100}%)')

print('\n추가 데이터셋 unknown 값 현황:')
for col in bank_additional_df.select_dtypes(include=['object']).columns:
    unknown_count = (bank_additional_df[col] == 'unknown').sum()
    if unknown_count > 0:
        print(f'{col}: {unknown_count} ({(unknown_count/len(bank_additional_df))*100}%)')

# 결측치 처리

```

```
def handle_missing_values(df):
    df_processed = df.copy()

    # 수치형 컬럼 결측치 처리
    numeric_columns      = df_processed.select_dtypes(include=['int64',
'float64']).columns
    for col in numeric_columns:
        if df_processed[col].isnull().sum() > 0:
            # 중앙값으로 대체
            df_processed[col].fillna(df_processed[col].median(), inplace=True)

    # 범주형 컬럼의 'unknown' 처리
    categorical_columns = df_processed.select_dtypes(include=['object']).columns
    for col in categorical_columns:
        if (df_processed[col] == 'unknown').sum() > 0:
            # 최빈값으로 대체
            mode_value = df_processed[col][df_processed[col] != 'unknown'].mode()[0]
            df_processed[col] = df_processed[col].replace('unknown', mode_value)

    return df_processed

# 결측치 처리 적용
bank_df_processed = handle_missing_values(bank_df)
bank_additional_df_processed = handle_missing_values(bank_additional_df)

# 결과 확인
print('\n처리 후 기본 데이터셋 unknown 값 현황:')
for col in bank_df_processed.select_dtypes(include=['object']).columns:
    unknown_count = (bank_df_processed[col] == 'unknown').sum()
    if unknown_count > 0:
        print(f'{col}: {unknown_count}')

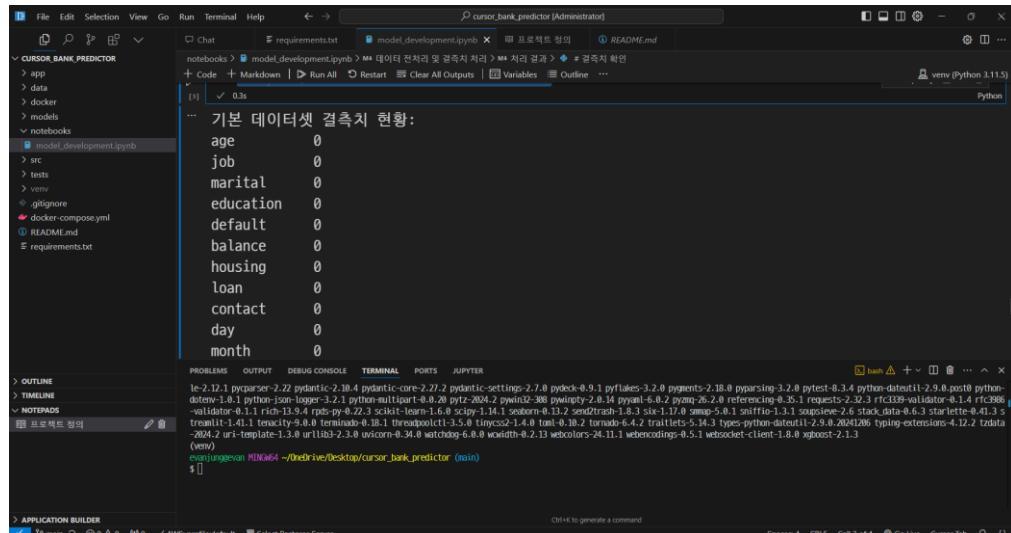
print('\n처리 후 추가 데이터셋 unknown 값 현황:')
for col in bank_additional_df_processed.select_dtypes(include=['object']).columns:
    unknown_count = (bank_additional_df_processed[col] == 'unknown').sum()
```

```

if unknown_count > 0:
    print(f'{col}: {unknown_count}')

```

결과를 확인해본다.



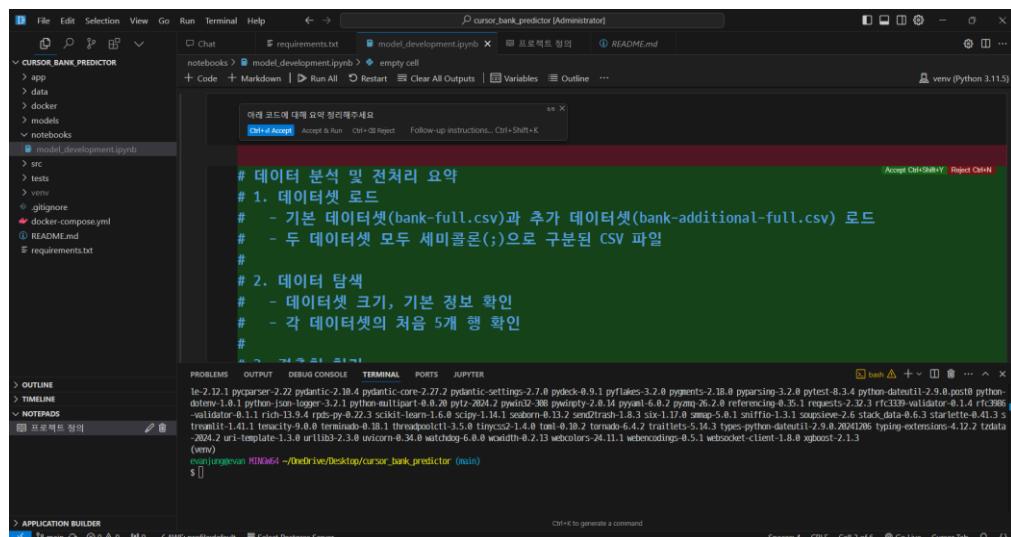
```

기본 데이터셋 결측치 현황:
age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan          0
contact      0
day          0
month        0

```

코드가 정상적으로 돌아간다면 해당 코드에 대해서 요약 정리하는 것을 마크다운에 추가할 수 있다.

아래 코드에 대해 요약 정리해주세요.



```

# 데이터 분석 및 전처리 요약
# 1. 데이터셋 로드
# - 기본 데이터셋(bank-full.csv)과 추가 데이터셋(bank-additional-full.csv) 로드
# - 두 데이터셋 모두 세미콜론(;)으로 구분된 CSV 파일
#
# 2. 데이터 탐색
# - 데이터셋 크기, 기본 정보 확인
# - 각 데이터셋의 처음 5개 행 확인
#

```

이 때 `#은 제거해주세요라고 하면 실제 마크다운 입력하는 방식으로 수정이 된다.`

A screenshot of a Jupyter Notebook interface. A tooltip is displayed over the code cell, reading "아래 코드에 대해 요약해주세요. #은 제거해주세요". Below the tooltip, the notebook content shows a section titled "데이터셋 기본 정보 분석" with the sub-section "기본 데이터셋과 추가 데이터셋을 불러와서 기본적인 정보를 확인했습니다.". It lists "데이터셋 크기" and "주요 변수".

```
notebooks > model_development.ipynb > 데이터셋 기본 정보 분석
+ Code + Markdown | ▶ Run All ⌘ Restart ⌘ Clear All Outputs | Variables Outline ...
▶ venv (Python 3.11.5)
Accept Ctrl+Shift+Y Reject Ctrl+H
```

PROBLEMS 93 OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPITER

```
te-2.12.1 pycparser-2.22 pydantic-core-2.10.4 pydantic-settings-2.7.0 pydeck-0.9.1 pylakes-3.2.0 pygments-2.18.0 pyParsing-3.2.0 pytest-8.3.4 python-dateutil-2.9.0.post0 python-dotenv-1.8.1 python-json-logger-3.2.1 python-multipart-0.0.20 pytz-2024.2 pwint-308 pywinpty-2.0.14 pyyaml-6.0.2 pyzmq-26.2.0 referencing-0.35.1 requests-2.32.3 rfc339-validator-0.1.4 rfc3986-validator-0.1.1 rich-13.9.4 rpds-py-0.22.3 scikit-learn-1.6.0 scipy-1.14.1 seaborn-0.13.2 send2trash-1.8.3 six-1.17.0 smmap-5.0.1 sniffio-1.3.1 soupsieve-2.6 stack_data-0.6.3 starlette-0.41.3 streamlit-1.41.1 tenacity-9.0.8 terminado-0.18.1 threadpoolctl-3.5.0 timycss2-1.4.0 tom-0.10.2 tornado-6.4.2 traitlets-5.14.3 types-python-dateutil-2.9.0.20241206 typing-extensions-4.12.2 tzdata-2024.2 uri-template-1.3.0 urllib3-2.3.0 uvicorn-0.34.0 watchdog-6.0.0 wodwidth-0.2.13 webcolors-24.11.1 webencodings-0.5.1 websocket-client-1.8.0 xgboost-2.1.3
(venv)
eunjung@eun MINGW64 ~/OneDrive/Desktop/cursor_bank_predictor (main)
$
```

이제 범주형 변수를 One-hot Encoding을 통해 처리한다.

범주형 변수를 One-Hot Encoding을 통해 처리해주세요.

A screenshot of a Jupyter Notebook interface showing the implementation of One-Hot Encoding. The code cell contains the following Python code:

```
# 범주형 변수 처리를 위한 함수 정의
def apply_one_hot_encoding(df):
    df_encoded = df.copy()

    # 범주형 변수 식별 (object 타입)
    categorical_columns = df.select_dtypes(include=['object']).columns
```

The code is being auto-converted by Jupyter Notebook, with a tooltip showing the original code and the converted code side-by-side.

```
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell height if necessary.
```

```
model_development.ipynb
# 범주형 변수 처리를 위한 함수 정의
def apply_one_hot_encoding(df):
    df_encoded = df.copy()

    # 범주형 변수 식별 (object 타입)
    categorical_columns = df.select_dtypes(include=['object']).columns
```

The tooltip also shows the original code and the converted code side-by-side.

```
def apply_one_hot_encoding(df):
    df_encoded = df.copy()

    # 범주형 변수 식별 (object 타입)
    categorical_columns = df.select_dtypes(include=['object']).columns
```

The notebook sidebar shows the project structure and other files like README.md and requirements.txt.

적용된 코드와 결과를 확인해본다.

```
# 범주형 변수 처리를 위한 함수 정의
def apply_one_hot_encoding(df):
    df_encoded = df.copy()

    # 범주형 변수 식별 (object 타입)
    categorical_columns = df.select_dtypes(include=['object']).columns

    # y 변수는 별도 처리
    if 'y' in categorical_columns:
        categorical_columns = categorical_columns.drop('y')
        # y 변수 레이블 인코딩 (no=0, yes=1)
        df_encoded['y'] = (df_encoded['y'] == 'yes').astype(int)

    # One-Hot Encoding 적용
    df_encoded = pd.get_dummies(df_encoded, columns=categorical_columns,
prefix=categorical_columns)

    return df_encoded

# 인코딩 적용
bank_df_encoded = apply_one_hot_encoding(bank_df_processed)
bank_additional_df_encoded = apply_one_hot_encoding(bank_additional_df_processed)

# 결과 확인
print('인코딩 후 기본 데이터셋 형태:', bank_df_encoded.shape)
print('인코딩 후 추가 데이터셋 형태:', bank_additional_df_encoded.shape)

# 처리된 컬럼 확인
print('\n기본 데이터셋 처리된 컬럼:')
print(bank_df_encoded.columns.tolist())
print('\n추가 데이터셋 처리된 컬럼:')
print(bank_additional_df_encoded.columns.tolist())
```

The screenshot shows a Jupyter Notebook interface with several tabs open. The active tab is 'model_development.ipynb'. The code cell contains the following Python code:

```
print(bank_additional_df_encoded.columns.tolist())
```

The output of the cell is:

```
[0] 0.15
[1] 인코딩 후 기본 데이터셋 형태: (45211, 48)
[2] 인코딩 후 추가 데이터셋 형태: (41188, 58)
```

Below the code cell, there is a note: '기본 데이터셋 처리된 컬럼:'. The subsequent code cell contains:

```
['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous', '...
```

Below this, another note says: '추가 데이터셋 처리된 컬럼:'. The next code cell contains:

```
['age', 'duration', 'campaign', 'pdays', 'previous', 'emp.var.rate', 'co...
```

At the bottom of the notebook, there is an empty markdown cell with the placeholder text: 'Empty markdown cell, double-click or press enter to edit.'

이제 SMOTE 기법을 사용해 불균형 데이터를 처리한다. 동일하게 코드를 요청하고, 생성된 코드를 새 셀에 붙여놓고 실행한다. 또한 결과도 같이 잘 나오는지 확인을 한다.

SMOTE 기법을 사용해서 불균형 데이터를 처리해주세요.

```
# 테스트 데이터는 그대로 유지
print('테스트 데이터 크기:', X_test.shape)

... SMOTE 적용 전 훈련 데이터 클래스 분포:
Counter({0: 31970, 1: 4198})

SMOTE 적용 후 훈련 데이터 크기: (31970, 47)
SMOTE 적용 후 훈련 데이터 크기: (63940, 47)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER

```
jupyter-events>:0.11.0>jupyter-server<:3>:4.0>jupyter-lab>r requirements.txt (line 10) (1.3.0)
Requirement already satisfied: types-python-dataclasses<=2.8.0> in c:\users\minhoon\desktop\cursor_bank_predictor\venv\lib\site-packages (from arrow<0.15.8>isoformat>jso
Requirement already satisfied: types-xml<=1.1.0> in c:\users\minhoon\desktop\cursor_bank_predictor\venv\lib\site-packages (from arrow<0.15.8>isoformat>jso
Downloaded: astropy<0.0>py3-none-any.whl (1.9 kB)
Downloaded: astropy<0.0>py3-none-any.whl (1.9 kB)
Installing collected packages: sklearn-compat, imbalanced-learn, joblib
Successfully installed instance-based-0.13.0 imblearn<0.0> sklearn-compat<0.1.3
(jupyter) minhoon@MINHO64 ~>/drive/Desktop/cursor_bank_predictor (main)
```

활용된 코드는 다음과 같다.

```
from imblearn.over_sampling import SMOTE
from collections import Counter

# 데이터 분리
X = bank_df_encoded.drop('y', axis=1)
y = bank_df_encoded['y']

# 훈련/테스트 데이터 분할
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# SMOTE 적용 전 클래스 분포 확인
print('SMOTE 적용 전 훈련 데이터 클래스 분포:')
print(Counter(y_train))

# SMOTE 적용
smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train, y_train)

# SMOTE 적용 후 클래스 분포 확인
print('\nSMOTE 적용 후 훈련 데이터 클래스 분포:')
print(Counter(y_train_balanced))

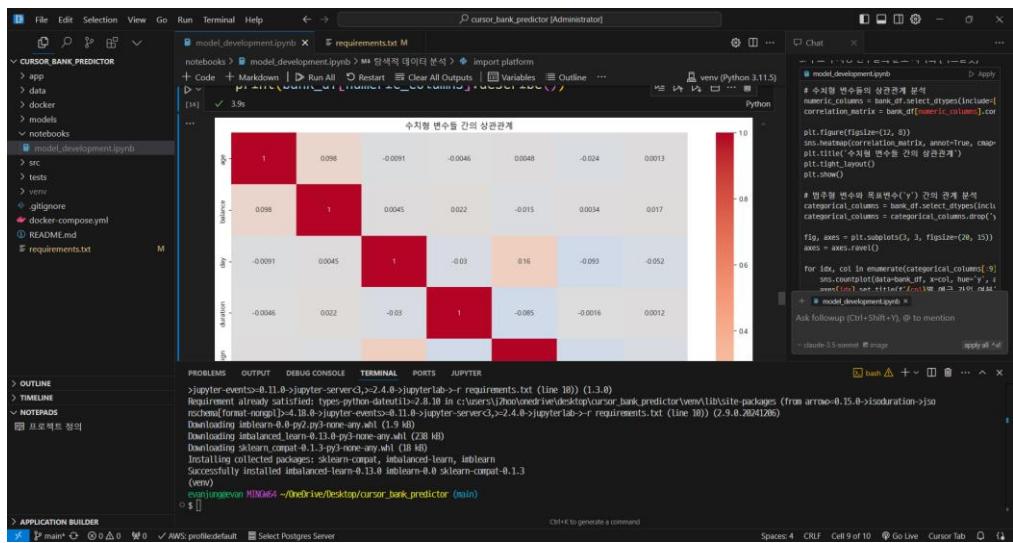
# 데이터 크기 확인
print('\n원본 훈련 데이터 크기:', X_train.shape)
print('SMOTE 적용 후 훈련 데이터 크기:', X_train_balanced.shape)

# 테스트 데이터는 그대로 유지
print('테스트 데이터 크기:', X_test.shape)
```

H. 탐색적 데이터 분석 예시

본 장에서는 탐색적 데이터 분석을 하도록 한다. 시각화를 통해 변수 간의 상관관계를 분석하고, 목표 변수 분포를 확인하는 것을 목표로 한다. 프롬프트는 다음과 같이 입력하였다.

EDA를 진행하려고 합니다. 우선 시각화를 통해 변수간의 상관관계를 분석할 수 있도록 도와주세요.



코드는 다음과 같다.

```
import platform

if platform.system() == 'Windows':
    plt.rc('font', family='Malgun Gothic')
else:
    plt.rc('font', family='AppleGothic')

plt.rcParams['axes.unicode_minus'] = False

# 수치형 변수들의 상관관계 분석
numeric_columns = bank_df.select_dtypes(include=['int64', 'float64']).columns
```

```
correlation_matrix = bank_df[numeric_columns].corr()

plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0)
plt.title('수치형 변수들 간의 상관관계')
plt.tight_layout()
plt.show()

# 범주형 변수와 목표변수('y') 간의 관계 분석
categorical_columns = bank_df.select_dtypes(include=['object']).columns
categorical_columns = categorical_columns.drop('y') # 목표변수 제외

fig, axes = plt.subplots(3, 3, figsize=(20, 15))
axes = axes.ravel()

for idx, col in enumerate(categorical_columns[:9]): # 처음 9개 변수만 시각화
    sns.countplot(data=bank_df, x=col, hue='y', ax=axes[idx])
    axes[idx].set_title(f'{col}별 예금 가입 여부')
    axes[idx].tick_params(axis='x', rotation=45)

plt.tight_layout()
plt.show()

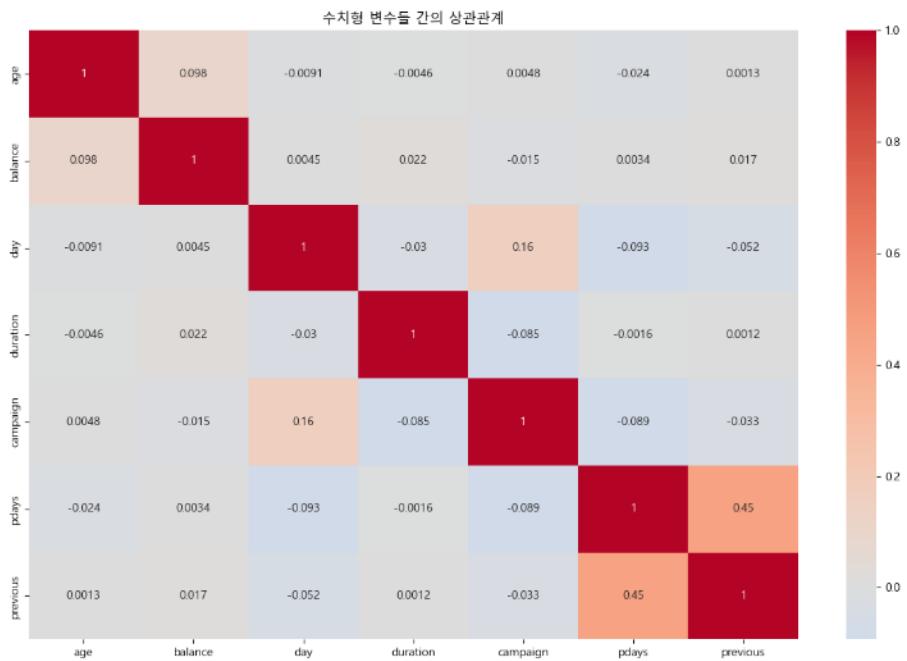
# 주요 수치형 변수들의 분포 시각화
numeric_vars = ['age', 'balance', 'duration', 'campaign']
fig, axes = plt.subplots(2, 2, figsize=(15, 10))
axes = axes.ravel()

for idx, col in enumerate(numeric_vars):
    sns.boxplot(data=bank_df, x='y', y=col, ax=axes[idx])
    axes[idx].set_title(f'{col}와 예금 가입 여부의 관계')

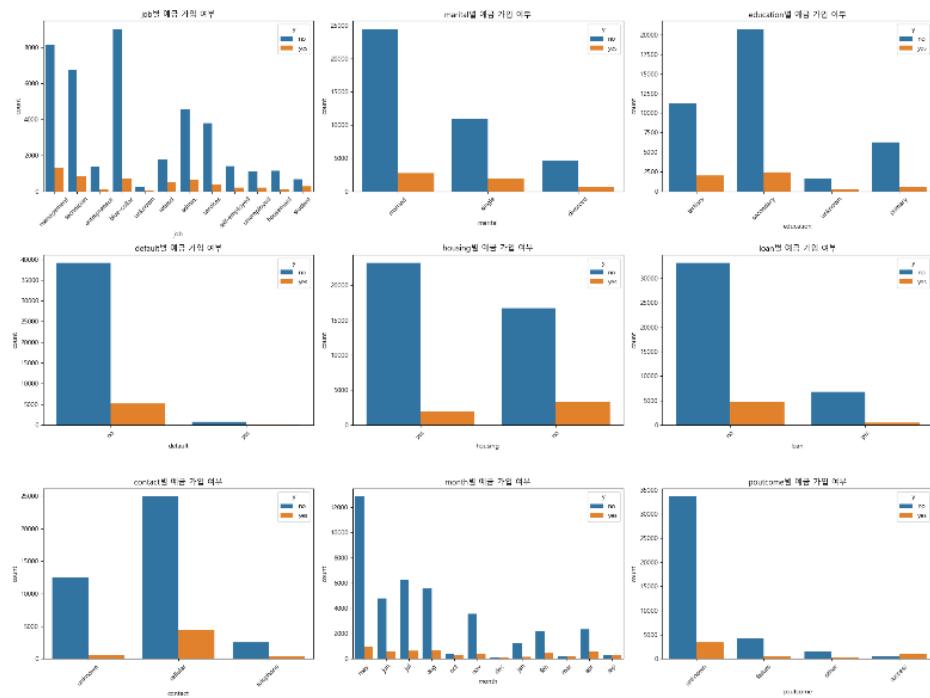
plt.tight_layout()
plt.show()
```

```
# 기본 통계량 출력
print("\n수치형 변수들의 기본 통계량:")
print(bank_df[numeric_columns].describe())
```

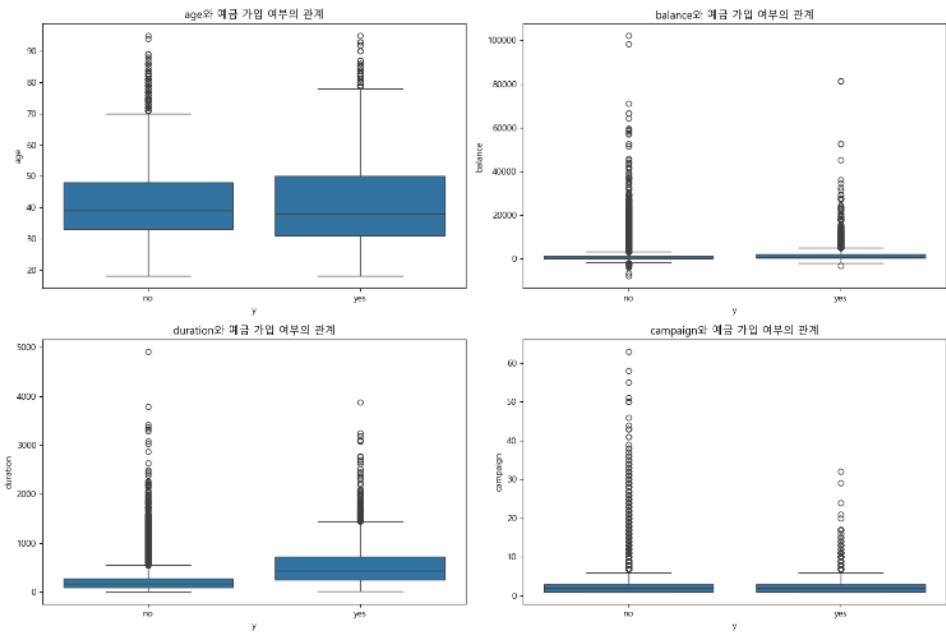
수치형 변수들 간의 상관관계 그래프는 다음과 같다.



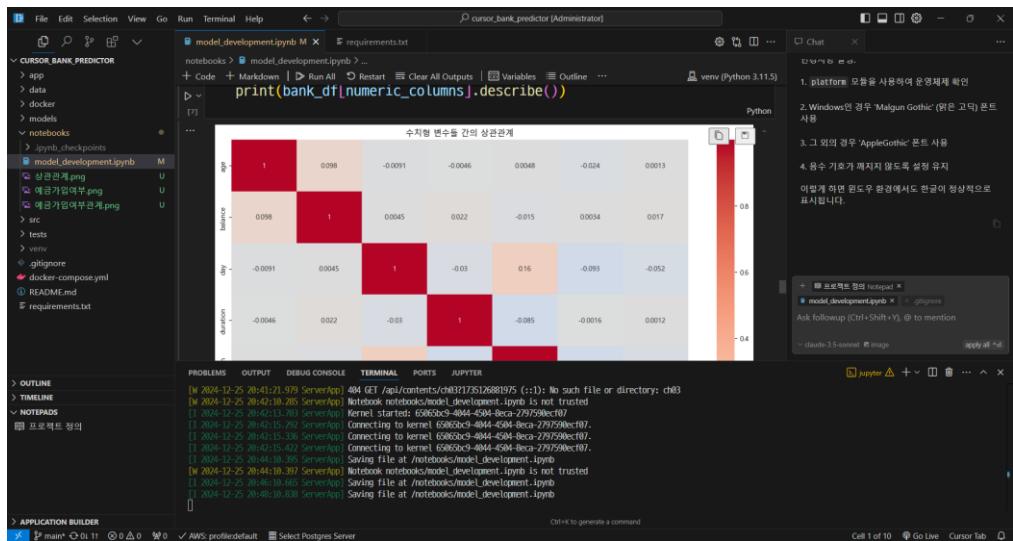
다음은 변수별 예금가입여부 그래프다.



변수별 가입여부 관계의 그래프는 다음과 같다.

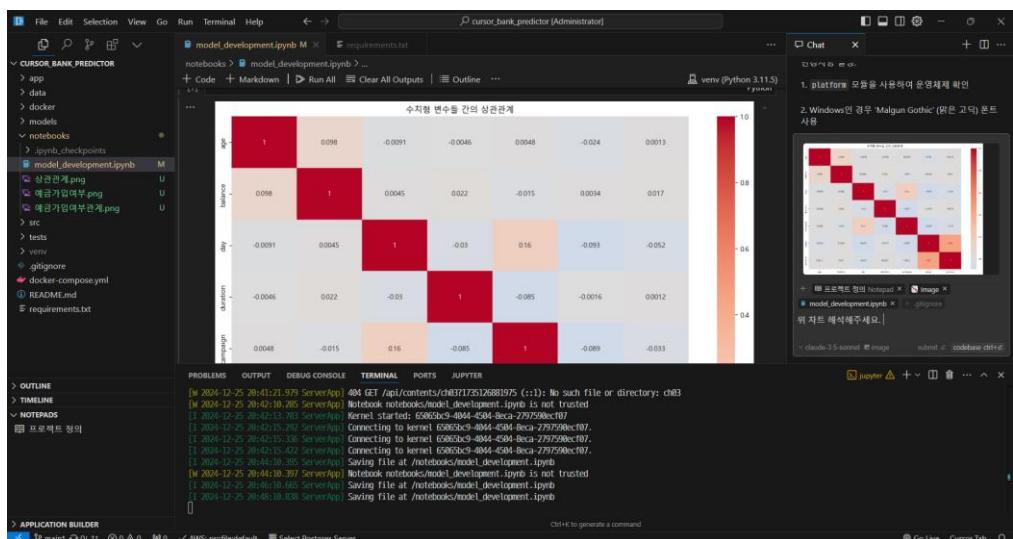


출력된 이미지에 마우스를 위치시키면 이미지 우측 상단에 복사할 수 있는 기능이 생긴다.



이 시각화 자료들을 하나씩 복사해서, 채팅창에 붙여넣으면, 비전 기능이 있는 모델(claude-3.5-sonnet, GPT-4o 등)은 이를 해석할 수 있다.

위 차트 해석해주세요.



The screenshot shows a Jupyter Notebook interface with several tabs open. The active tab is 'model_development.ipynb'. The code in the cell is as follows:

```
sns.countplot(data=bank_df, x=col, hue='y', ax=axes[0])
axes[0].set_title(f'{col}별 예금 가입 여부')
axes[0].tick_params(axis='x', rotation=45)

plt.tight_layout()
plt.savefig("예금가입여부.png")
plt.show()
```

Below the code, there is a note in Korean:

주요 수치형 변수들의 분포 시각화

```
numeric_vars = ['age', 'balance', 'duration', 'campaign']
fig, axes = plt.subplots(2, 2, figsize=(15, 10))
axes = axes.ravel()
```

The notebook also displays a list of recent notebooks and a sidebar with various tools and status indicators.

이번에는 Target 변수를 확인한다. 다음과 같이 프롬프트를 작성한다.

분석 목표변수의 분포를 파악하고자 합니다. 목표변수는 y로서 정기 예금 가입 여부입니다.

```
# 목표변수 분포 시각화
plt.figure(figsize=(10, 6))

# 막대 그래프
sns.countplot(data=bank_df, x='y')
plt.title('정기 예금 가입 여부 분포')
plt.xlabel('예금 가입 여부')
plt.ylabel('고객 수')

# 수치 표시
total = len(bank_df)
for p in plt.gca().patches:
    percentage = ((p.get_height() / total) * 100)
    plt.annotate(f'{percentage:.1f}%', (p.get_x() + p.get_width() / 2,
                                    p.get_y() + 5), va='bottom')

plt.show()
```

코드는 다음과 같다.

```
# 목표변수 분포 시각화  
plt.figure(figsize=(10, 6))
```

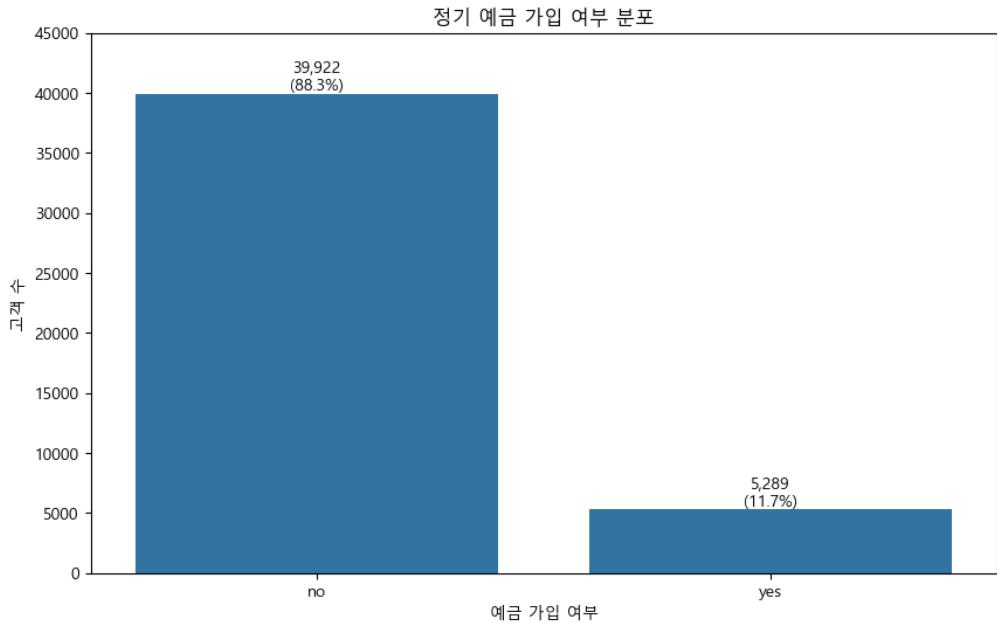
```
# 막대 그래프
sns.countplot(data=bank_df, x='y')
plt.title('정기 예금 가입 여부 분포')
plt.xlabel('예금 가입 여부')
plt.ylabel('고객 수')

# y축 범위 설정
plt.ylim(0, 45000)

# 수치 표시 (천단위 쉼표 적용)
total = len(bank_df)
for p in plt.gca().patches:
    count = int(p.get_height())
    percentage = f'{100 * count / total:.1f}%'
    plt.annotate(f'{count},\n{percentage}', 
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='bottom')

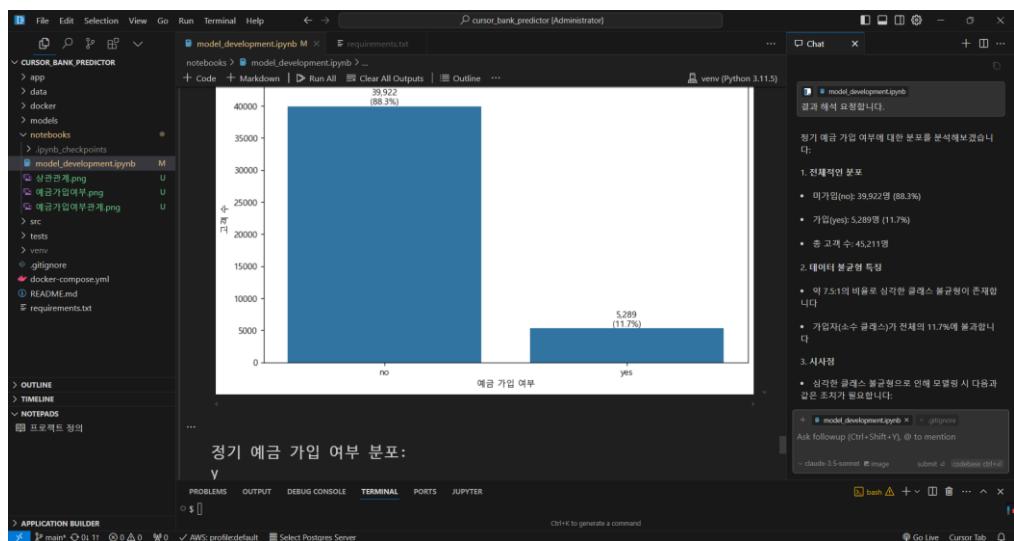
plt.show()

# 상세 통계 (천단위 쉼표 적용)
print("\n정기 예금 가입 여부 분포:")
distribution = bank_df['y'].value_counts()
print(distribution.apply(lambda x: format(x, ',')))
print("\n비율:")
print(bank_df['y'].value_counts(normalize=True).round(3) * 100, '%')
```



결과 이미지를 복사해서 결과 해석을 요청한다.

결과 해석 요청합니다.

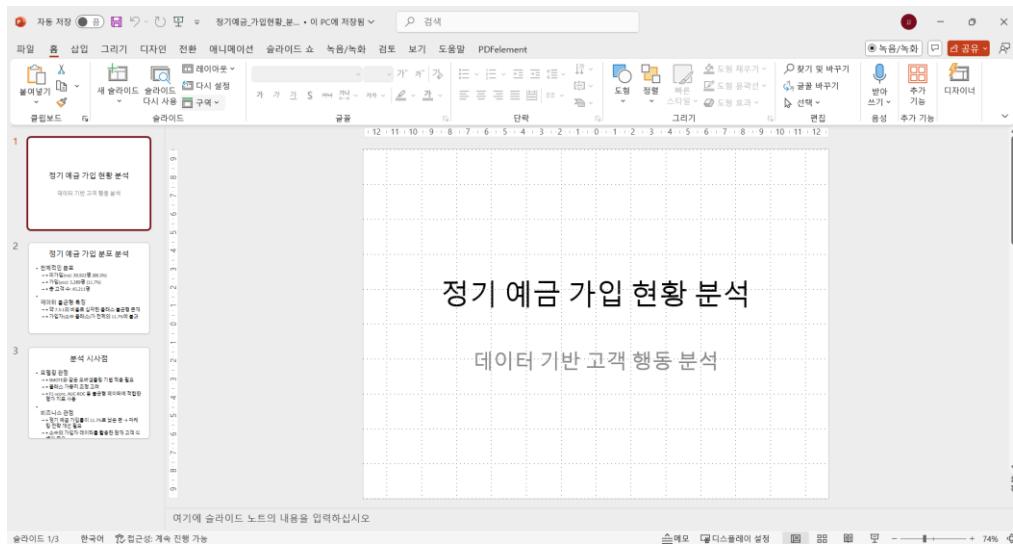


이번에는 결과해석한 것을 파워포인트로 내보내기를 하도록 합니다. 프롬프트로 다음과 같이 작성하였습니다.

결과해석한 것을 ppt 슬라이드로 요청합니다. 다운로드 가능하도록 해주세요

실제로 요약 정리해서 슬라이드로 내보내기가 가능한 것을 확인할 수 있다.

(파일명 : 정기예금_가입현황_분석.pptx 참고)



코드는 다음과 같다. 먼저 `python-pptx` 라이브러리는 개별적으로 설치를 해주기를 바란다.

```
# 필요한 모듈 설치
# !pip install python-pptx

# 필요한 라이브러리 임포트
from pptx import Presentation
from pptx.util import Inches, Pt
from pptx.enum.text import PP_ALIGN
from pptx.dml.color import RGBColor

# 새로운 프레젠테이션 생성
prs = Presentation()

# 1. 제목 슬라이드 추가
```

```
title_slide_layout = prs.slide_layouts[0]
slide = prs.slides.add_slide(title_slide_layout)
title = slide.shapes.title
subtitle = slide.placeholders[1]

title.text = "정기 예금 가입 현황 분석"
subtitle.text = "데이터 기반 고객 행동 분석"

# 2. 분포 분석 슬라이드 추가
bullet_slide_layout = prs.slide_layouts[1]
slide = prs.slides.add_slide(bullet_slide_layout)
shapes = slide.shapes

title_shape = shapes.title
body_shape = shapes.placeholders[1]

title_shape.text = "정기 예금 가입 분포 분석"

tf = body_shape.text_frame
tf.text = "전체적인 분포"

p = tf.add_paragraph()
p.text = "• 미가입(no): 39,922명 (88.3%)"
p.level = 1

p = tf.add_paragraph()
p.text = "• 가입(yes): 5,289명 (11.7%)"
p.level = 1

p = tf.add_paragraph()
p.text = "• 총 고객 수: 45,211명"
p.level = 1

p = tf.add_paragraph()
p.text = "\n데이터 불균형 특징"
```

```
p.level = 0

p = tf.add_paragraph()
p.text = "• 약 7.5:1의 비율로 심각한 클래스 불균형 존재"
p.level = 1

p = tf.add_paragraph()
p.text = "• 가입자(소수 클래스)가 전체의 11.7%에 불과"
p.level = 1

# 3. 시사점 슬라이드 추가
slide = prs.slides.add_slide(bullet_slide_layout)
shapes = slide.shapes

title_shape = shapes.title
body_shape = shapes.placeholders[1]

title_shape.text = "분석 시사점"

tf = body_shape.text_frame
tf.text = "모델링 관점"

p = tf.add_paragraph()
p.text = "• SMOTE와 같은 오버샘플링 기법 적용 필요"
p.level = 1

p = tf.add_paragraph()
p.text = "• 클래스 가중치 조정 고려"
p.level = 1

p = tf.add_paragraph()
p.text = "• F1-score, AUC-ROC 등 불균형 데이터에 적합한 평가 지표 사용"
p.level = 1

p = tf.add_paragraph()
```

```
p.text = "\n비즈니스 관점"
p.level = 0

p = tf.add_paragraph()
p.text = "• 정기 예금 가입률이 11.7%로 낮은 편 → 마케팅 전략 개선 필요"
p.level = 1

p = tf.add_paragraph()
p.text = "• 소수의 가입자 데이터를 활용한 잠재 고객 식별이 중요"
p.level = 1

# 프레젠테이션 저장
prs.save('정기예금_가입현황_분석.pptx')

print("PPT 파일이 생성되었습니다: '정기예금_가입현황_분석.pptx'")
```

I. 머신러닝 모델링

이번 단계에서는 모델링을 진행할 것이다. XGBoost 와 Random Forest 를 통해 분류 모델을 생성하고, Recursive Feature Elimination(RFE)로 중요한 변수를 선택한 후, Hyperparameter Tuning 까지 진행한다. 프롬프트는 다음과 같이 작성하였다.

모델링을 진행하려고 합니다.

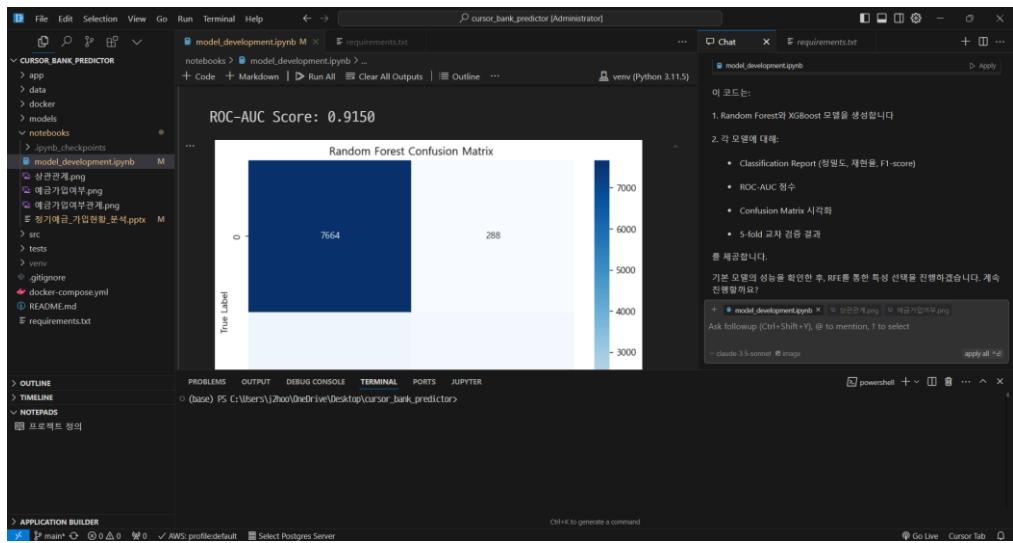
1. XGBoost와 Random Forest를 통해 분류 모델을 생성하세요.
2. Recursive Feature Elimination(RFE)로 중요한 변수를 선택하세요.
3. Hyperparameter Tuning까지 진행해주세요.

단계적으로 나누어 진행해주세요. 응답이 길어질 것으로 예상되는 경우 사용자가 이어서 진행해달라고 할 때까지 다음 단계로 넘어가지 마세요.

생성형 AI는 응답이 길어질수록 품질이 저하되는 경향이 있다. 이는 AI가 긴 문맥을 유지하면서도 일관성 있게 고품질의 답변을 제공하는 데 어려움을 겪기 때문이다. 긴 응답을 제공할 경우, 중간에 정보의 일관성이 떨어지거나 중복되는 부분이 발생할 수 있다. 이를 해결하기 위해, AI는 긴 응답을 단계적으로 나누어 제공하는 방법을 사용한다. 한 번에 모든 내용을 제공하기보다는, 중요한 내용을 순차적으로 나누어 제공하는 것이 품질을 유지하는 데 유리하다. 이렇게 하면 각 단계에서 필요한 정보만 제공하고, 사용자가 추가로 요청할 때마다

그에 맞는 상세한 정보를 제공할 수 있다. 긴 응답을 안전하게 생성하기 위해, 마지막에 특정 프롬프트를 추가한다. 예를 들어서 “**단계적으로 나누어 진행해주세요. 응답이 길어질 것으로 예상되는 경우 사용자가 이어서 진행해달라고 할 때까지 다음 단계로 넘어가지 마세요.**” 같은 프롬프트는 AI가 한 번에 너무 많은 정보를 제공하는 것을 방지하고, 사용자가 원하는 만큼 정보를 추가로 요청할 수 있도록 유도한다.

첫번째 응답 결과는 다음과 같다.



코드는 다음과 같다.

```
# 필요한 라이브러리 임포트
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, roc_auc_score, confusion_matrix
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# 기본 모델 생성
rf_model = RandomForestClassifier(
    n_estimators=100,
    random_state=42
)
```

```
xgb_model = XGBClassifier(  
    n_estimators=100,  
    objective='binary:logistic',  
    random_state=42,  
    use_label_encoder=False,  
    eval_metric='logloss'  
)  
  
# 모델 평가 함수  
def evaluate_model(model, X_train, y_train, X_test, y_test, model_name):  
    # 모델 학습  
    model.fit(X_train, y_train)  
  
    # 예측  
    y_pred = model.predict(X_test)  
    y_pred_proba = model.predict_proba(X_test)[:, 1]  
  
    # 성능 평가  
    print(f"\n{model_name} 성능 평가")  
    print("-" * 50)  
  
    # Classification Report  
    print("\nClassification Report:")  
    print(classification_report(y_test, y_pred))  
  
    # ROC-AUC Score  
    roc_auc = roc_auc_score(y_test, y_pred_proba)  
    print(f"\nROC-AUC Score: {roc_auc:.4f}")  
  
    # Confusion Matrix  
    cm = confusion_matrix(y_test, y_pred)  
    plt.figure(figsize=(8, 6))  
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')  
    plt.title(f'{model_name} Confusion Matrix')
```

```
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.show()

return model

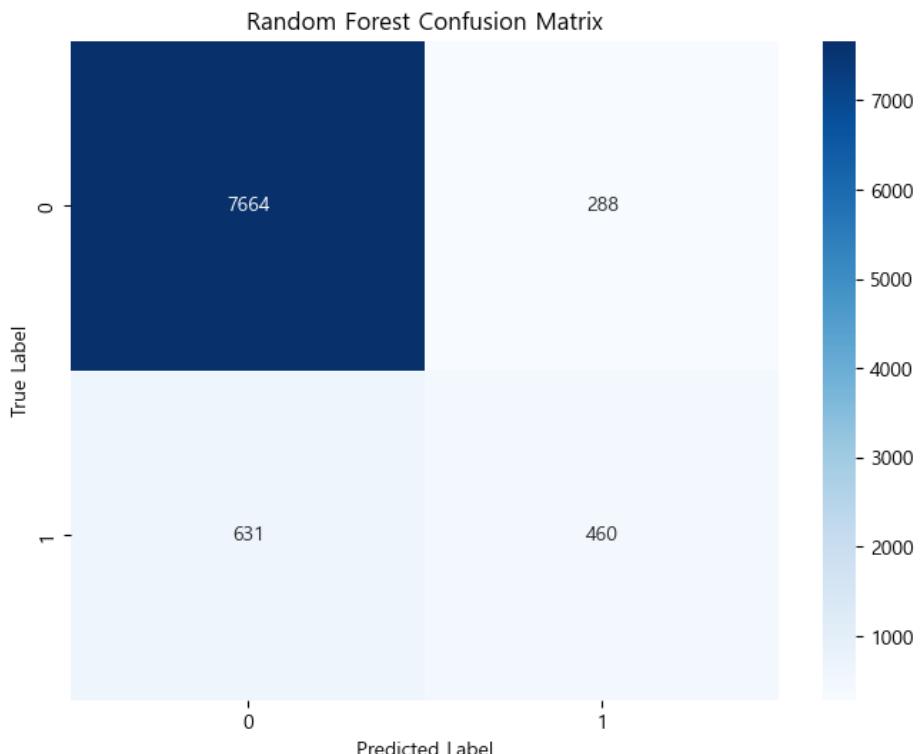
# Random Forest 모델 평가
print("Random Forest 모델 평가 시작...")
rf_base = evaluate_model(rf_model, X_train_balanced, y_train_balanced,
                         X_test, y_test, "Random Forest")

# XGBoost 모델 평가
print("\nXGBoost 모델 평가 시작...")
xgb_base = evaluate_model(xgb_model, X_train_balanced, y_train_balanced,
                          X_test, y_test, "XGBoost")
```

결과는 다음과 같이 나올 것이다.

| Random Forest 성능 평가 | | | | | |
|------------------------|-----------|--------|----------|---------|--|
| Classification Report: | | | | | |
| | precision | recall | f1-score | support | |
| 0 | 0.92 | 0.96 | 0.94 | 7952 | |
| 1 | 0.61 | 0.42 | 0.50 | 1091 | |
| accuracy | | | 0.90 | 9043 | |
| macro avg | 0.77 | 0.69 | 0.72 | 9043 | |
| weighted avg | 0.89 | 0.90 | 0.89 | 9043 | |

ROC-AUC Score: 0.9150

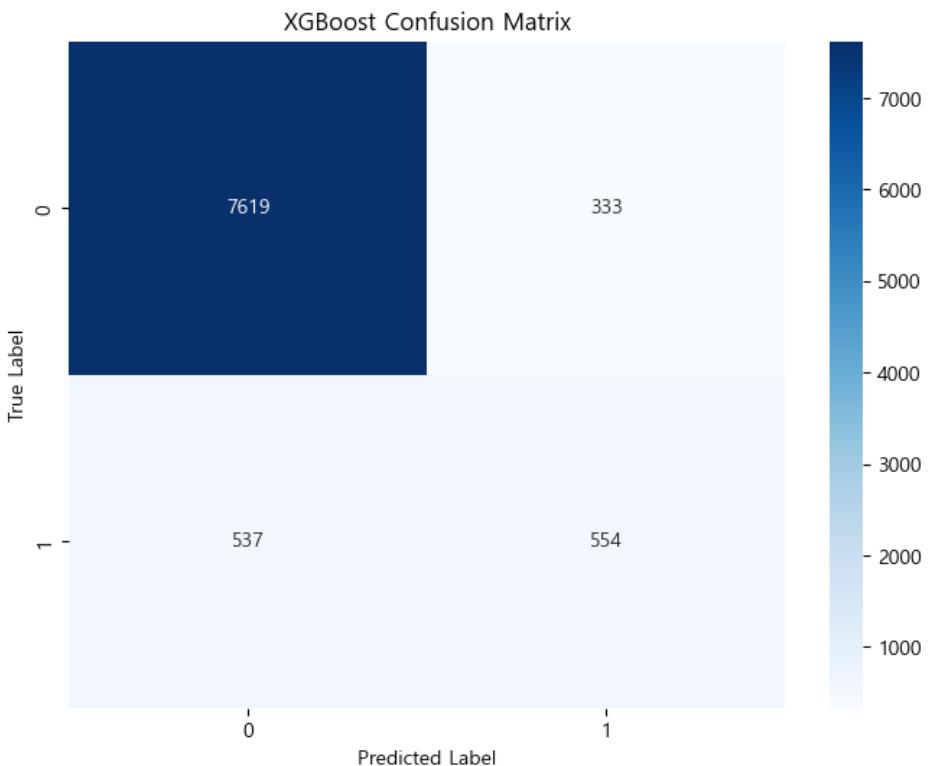


XGBoost 성능 평가

Classification Report:

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.93 | 0.96 | 0.95 | 7952 |
| 1 | 0.62 | 0.51 | 0.56 | 1091 |
| accuracy | | | 0.90 | 9043 |
| macro avg | 0.78 | 0.73 | 0.75 | 9043 |
| weighted avg | 0.90 | 0.90 | 0.90 | 9043 |

ROC-AUC Score: 0.9247



이번에는 RFE를 통해서 특성 조합을 확인하고, 특성 중요도에 따라 변수를 선택한다. 각 모델의 특성 중요도를 계산하고 시각화 한다. 상위 10개의 특성을 선택한다. 선택된 특성만으로 모델의 성능을 다시 평가하도록 한다. 전체 코드는 다음과 같다.

cursor_bank_predictor [Administrator]

```

File Edit Selection View Go Run Terminal Help <- > cursor_bank_predictor [Administrator]
model_development.ipynb M requirements.txt
notebooks > model_development.ipynb ...
+ Code + Markdown | Run All Clear All Outputs | Outline ...
Macro avg 0.59 0.59 0.59 9043
weighted avg 0.83 0.82 0.82 9043
ROC-AUC Score: 0.6542
XGBoost (Selected Features) Confusion Matrix
True Label
0 - 7619 333
1 - 537 554
Predicted Label
0 1

```

아래는 코드입니다.

```

print("시작합니다. 평생동안 AWS 모델 평가!")
sgt_selected = estimate_model(
    X_train_xgb_selected,
    y_train_balanced,
    X_val_xgb_selected,
    y_val,
    "XGBBoost (Selected features)"
)

```

1. 각 모델의 특성 중요도를 계산하고 시각화합니다
2. 상위 10개의 중요 특성을 선택합니다
3. 선택된 특성만으로 모델의 성능을 다시 평가합니다
특성 선택 결과를 확인한 후, 하이퍼파라미터 튜닝을 진행하겠습니다.
다음은 진정한 결과입니다.

```

# 특성 중요도 분석 및 선택 함수
def analyze_feature_importance(model, X, y, model_name):
    # 모델 학습
    model.fit(X, y)

    # 특성 중요도 추출
    importances = model.feature_importances_

    # 특성 중요도를 데이터프레임으로 변환
    feature_importance = pd.DataFrame({
        'feature': X.columns,
        'importance': importances
    })
    # 중요도 기준 정렬
    feature_importance      =      feature_importance.sort_values('importance',
ascending=False)
    # 시각화
    plt.figure(figsize=(12, 6))
    sns.barplot(x='importance', y='feature', data=feature_importance.head(15))
    plt.title(f'{model_name} - Top 15 Feature Importance')
    plt.show()
    return feature_importance

# Random Forest 특성 중요도 분석
print("Random Forest 특성 중요도 분석")
rf_importance      =      analyze_feature_importance(rf_model,      X_train_balanced,
y_train_balanced, "Random Forest")
print("\nTop 10 중요 특성:")
print(rf_importance.head(10))

# XGBoost 특성 중요도 분석
print("\nXGBoost 특성 중요도 분석")
xgb_importance     =      analyze_feature_importance(xgb_model,     X_train_balanced,
y_train_balanced, "XGBoost")
print("\nTop 10 중요 특성:")

```

```
print(xgb_importance.head(10))

# 상위 10개 특성 선택
n_features = 10
rf_selected_features = rf_importance['feature'].head(n_features).tolist()
xgb_selected_features = xgb_importance['feature'].head(n_features).tolist()

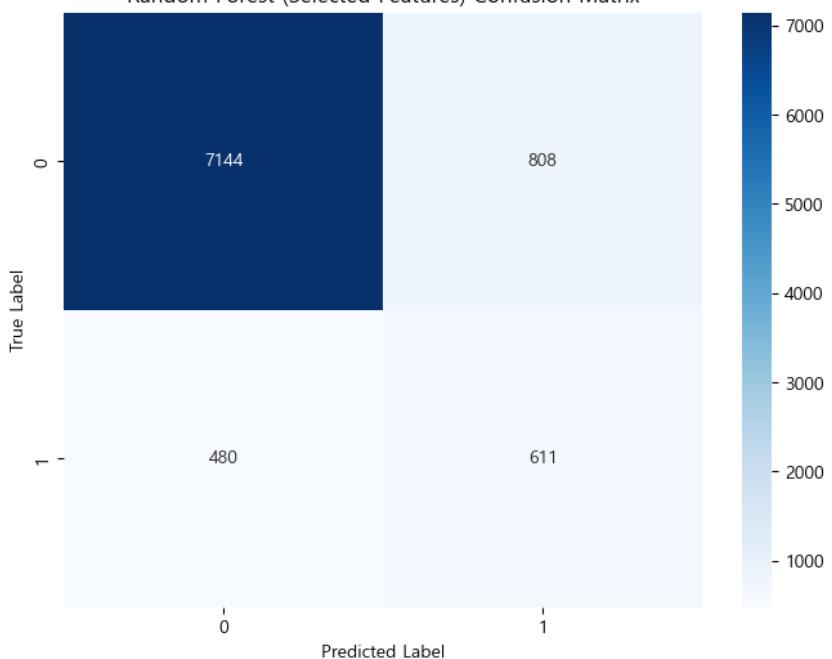
# 선택된 특성으로 데이터셋 생성
X_train_rf_selected = X_train_balanced[rf_selected_features]
X_test_rf_selected = X_test[rf_selected_features]

X_train_xgb_selected = X_train_balanced[xgb_selected_features]
X_test_xgb_selected = X_test[xgb_selected_features]

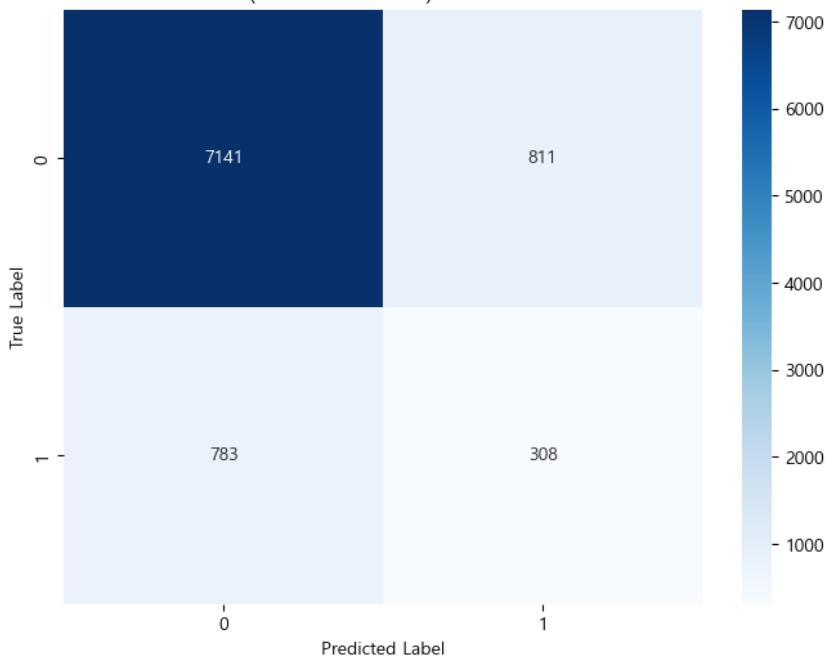
# 선택된 특성으로 모델 성능 평가
print("\n선택된 특성으로 Random Forest 모델 평가")
rf_selected = evaluate_model(
    RandomForestClassifier(random_state=42),
    X_train_rf_selected,
    y_train_balanced,
    X_test_rf_selected,
    y_test,
    "Random Forest (Selected Features)"
)

print("\n선택된 특성으로 XGBoost 모델 평가")
xgb_selected = evaluate_model(
    XGBClassifier(random_state=42),
    X_train_xgb_selected,
    y_train_balanced,
    X_test_xgb_selected,
    y_test,
    "XGBoost (Selected Features)"
)
```

Random Forest (Selected Features) Confusion Matrix



XGBoost (Selected Features) Confusion Matrix



먼저 Random Forest의 특성 중요도 분석을 요약 하면 다음과 같다.¹⁶⁴ Random Forest의 주요 피처는 duration, housing_yes, month_may 순으로 제안되고 있다. XGBoost의 경우 housing_yes, poutcome_failure, poutcome_other 순으로 제안되고 있다. 제안된 10 개의 Feature를 기반으로 모델을 생성하면 Random Forest 모델의 평가는 0.86으로 XGBoost 모델의 평가 0.65 보다 더 높은 것으로 나타난다. 실제 Confusion Matrix를 통해서 어떤 모델에서 특성 선택을 잘 했는지 비교할 수 있다. 간단하게 요약하면 Random Forest의 피처 조합이 음성(TN)을 음성(TN)으로 더 잘 잡아내는 것을 확인할 수 있다.

이번에는 하이퍼파라미터 튜닝을 진행한다. GridSearchCV 를 사용하여 최적의 파라미터를 찾아보도록 한다. 코드에 대한 설명은 간단하게 요약하면 다음과 같다. 기본 모델, 특성 선택 후, 하이퍼파라미터 튜닝 후의 성능을 비교한다. 각 모델의 주요 파라미터는 다음과 같다.

- Random Forest : n_estimators, max_depth, min_samples_split, min_samples_leaf
 - XGBoost: n_estimators, max_depth, learning_rate, subsample

하이퍼파라미터 튜닝은 시간이 오래 걸리기 때문에 본 교재에서는 시간을 줄이도록 별도로 요청을 하였다.

랜덤서치로 진행하고, CV는 2로 선택, 하이퍼파라미터는 간소화하게 해주세요. 10분내에 끝나도록 설계 부탁드립니다.

File Edit Selection View Go Run Terminal Help ↵ cursor_bank_predictor

CHAT COMPOSER BUG FINDER + ⌘+N x

model_development.ipynb [IM_D] x notebooks > model_development.ipynb > ...

+ Code Markdown Run All Restart Clear All Outputs Go To Variables Outline venv (Python 3.12.6)

Random Forest 하이퍼파라미터 튜닝 시작...
Fitting 2 folds for each of 10 candidates, totalling 20 fits

최적 파라미터: {'max_depth': 30, 'min_samples_split': 5, 'n_estimators': 5}
최적 CV 점수: 0.9000

Random Forest (Tuned) 성능 평가

Classification Report:

| | precision | recall | f1-score | support |
|----------|-----------|--------|----------|---------|
| 0 | 0.94 | 0.90 | 0.92 | 7952 |
| 1 | 0.43 | 0.58 | 0.49 | 1091 |
| accuracy | | | 0.86 | 9043 |

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER powershell ↵

tor>

Random Forest 하이퍼파라미터 미리보기
랜덤 포레스트를 실행하고, CVE는 2로 선택, 하이퍼파라미터는 간소화하여 예측하세요. 10분내로 끝나도록 계획해 보드입니다.

네, 간소화된 하이퍼파라미터와 빠른 실행을 위한 설정으로 진행하겠습니다.

```
# model_development.ipynb
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint, uniform

# Random Forest 하이퍼파라미터 공간
rf_param_dist = {
    "n_estimators": randint(50, 200),
    "max_depth": [None, 10, 20, 30],
    "min_samples_split": randint(2, 10)
}

# XGBoost 하이퍼파라미터 공간
xgb_param_dist = {
    "n_estimators": randint(50, 200),
    "max_depth": randint(3, 10),
    "learning_rate": uniform(0.01, 0.3)
}
```

엔딩 처리 함수
def random_search_tuning(model, param_dist, n_iter=10):
 # 엔딩 처리 설정
 random_search = RandomizedSearchCV(
 model,
 param_distributions=param_dist,
 n_iter=n_iter, # 10회 시도
 cv=2, # 2-fold CV
 scoring='f1'
)
 random_search.fit(X_train, y_train)

Ask followup (Ctrl+Shift+Y), @ to mention

classmate-15-format.html image apply all ↵

¹⁶⁴ 전체 결과에 대한 시각화 등은 model_development.ipynb 파일을 확인한다.

구현된 코드는 다음과 같다.

```
from sklearn.model_selection import RandomizedSearchCV

# 하이퍼파라미터 튜닝 함수
def tune_hyperparameters(model, param_dist, X_train, y_train, X_test, y_test,
model_name):
    # RandomizedSearchCV 객체 생성
    random_search = RandomizedSearchCV(
        estimator=model,
        param_distributions=param_dist,
        n_iter=10, # 10회 시도
        cv=2,      # 2-fold CV
        scoring='roc_auc',
        n_jobs=-1,
        verbose=1,
        random_state=42
    )

    # 모델 학습
    print(f"\n{model_name} 하이퍼파라미터 튜닝 시작...")
    random_search.fit(X_train, y_train)

    # 최적 파라미터 및 점수 출력
    print(f"\n최적 파라미터: {random_search.best_params_}")
    print(f"최고 ROC-AUC 점수: {random_search.best_score_.:.4f}")

    # 최적 모델로 테스트 세트 평가
    best_model = random_search.best_estimator_
    evaluate_model(best_model, X_train, y_train, X_test, y_test,
                   f"{model_name} (Tuned)")

    return best_model, random_search.best_params_

# Random Forest 튜닝을 위한 파라미터 분포
rf_param_dist = {
```

```
'n_estimators': [50, 100, 150],  
    'max_depth': [10, 20, None],  
    'min_samples_split': [2, 5, 10]  
}  
  
# XGBoost 튜닝을 위한 파라미터 분포  
xgb_param_dist = {  
    'n_estimators': [50, 100, 150],  
    'max_depth': [3, 5, 7],  
    'learning_rate': [0.01, 0.1, 0.3]  
}  
  
# Random Forest 튜닝  
rf_best_model, rf_best_params = tune_hyperparameters(  
    RandomForestClassifier(random_state=42),  
    rf_param_dist,  
    X_train_rf_selected,  
    y_train_balanced,  
    X_test_rf_selected,  
    y_test,  
    "Random Forest"  
)  
  
# XGBoost 튜닝  
xgb_best_model, xgb_best_params = tune_hyperparameters(  
    XGBClassifier(random_state=42, eval_metric='logloss'),  
    xgb_param_dist,  
    X_train_xgb_selected,  
    y_train_balanced,  
    X_test_xgb_selected,  
    y_test,  
    "XGBoost"  
)  
  
# 최종 모델 성능 비교
```

```
print("\n==== 최종 모델 성능 비교 ===")
print("\n1. 기본 모델")
print("Random Forest ROC-AUC:", roc_auc_score(y_test,
rf_base.predict_proba(X_test)[:, 1]))
print("XGBoost ROC-AUC:", roc_auc_score(y_test, xgb_base.predict_proba(X_test)[:, 1])))

print("\n2. 특성 선택 후")
print("Random Forest ROC-AUC:", roc_auc_score(y_test,
rf_selected.predict_proba(X_test_rf_selected)[:, 1]))
print("XGBoost ROC-AUC:", roc_auc_score(y_test,
xgb_selected.predict_proba(X_test_xgb_selected)[:, 1]))

print("\n3. 하이퍼파라미터 튜닝 후")
print("Random Forest ROC-AUC:", roc_auc_score(y_test,
rf_best_model.predict_proba(X_test_rf_selected)[:, 1]))
print("XGBoost ROC-AUC:", roc_auc_score(y_test,
xgb_best_model.predict_proba(X_test_xgb_selected)[:, 1]))
```

지금까지 진행된 최종 모델 성능 비교하면 다음과 같다.

==== 최종 모델 성능 비교 ===

1. 기본 모델

Random Forest ROC-AUC: 0.9131188367602499
XGBoost ROC-AUC: 0.9265224712159298

2. 특성 선택 후

Random Forest ROC-AUC: 0.8688449440916812
XGBoost ROC-AUC: 0.6627279718641823

3. 하이퍼파라미터 튜닝 후

Random Forest ROC-AUC: 0.8712029855577093
XGBoost ROC-AUC: 0.6713001427446439

이제 모델 평가 및 저장을 순차적으로 진행하도록 한다. 분류모델 평가와 관련된 지표를 프롬프트에 입력한다.

ROC-AUC, 정확도, 정밀도, 재현율, F1-Score를 계산하고 중요 변수를 해석하여 모델을 평가해주세요.

The screenshot shows a Jupyter Notebook environment with several tabs open. The main notebook, 'model_development.ipynb', contains the following code and output:

```

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

```

Output from the cell:

- poutcome_failure: 고객의 예금 가입 결정에 2번째로 중요한 영향을 미침
- poutcome_other: 고객의 예금 가입 결정에 3번째로 중요한 영향을 미침
- housing_no: 고객의 예금 가입 결정에 4번째로 중요한 영향을 미침
- loan_yes: 고객의 예금 가입 결정에 5번째로 중요한 영향을 미침

3. 비즈니스 시사점

- 가장 영향력 있는 변수들을 중심으로 마케팅 전략 수립 가능
- 높은 예측 정확도를 바탕으로 효율적인 타겟 마케팅 가능
- 재현율과 정밀도의 균형을 고려한 마케팅 비용 최적화 가능

TERMINAL window (MINGW64 ~/Desktop/cursor_bank_predictor (main)):

```

$ powershell
$ bash

```

구현된 코드는 다음과 같다. 각 코드에 대한 상세한 결과표는 model_development.ipynb 파일을 확인한다.

```

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
roc_auc_score

import matplotlib.pyplot as plt
import seaborn as sns

def evaluate_model_metrics(model, X_test, y_test, model_name):
    # 예측 및 확률 계산
    y_pred = model.predict(X_test)
    y_pred_proba = model.predict_proba(X_test)[:, 1]

    # 메트릭 계산
    metrics = {
        'Accuracy': accuracy_score(y_test, y_pred),
        'Precision': precision_score(y_test, y_pred),

```

```

'Recall': recall_score(y_test, y_pred),
'F1-Score': f1_score(y_test, y_pred),
'ROC-AUC': roc_auc_score(y_test, y_pred_proba)
}

print(f"\n{model_name} 성능 평가")
print("-" * 50)
for metric, value in metrics.items():
    print(f"{metric}: {value:.4f}")

return metrics

def plot_feature_importance(model, feature_names, model_name):
    # 특성 중요도 추출
    importances = model.feature_importances_
    indices = np.argsort(importances)[::-1]

    plt.figure(figsize=(10, 6))
    plt.title(f'{model_name} - Feature Importance')
    plt.bar(range(len(importances)), importances[indices])
    plt.xticks(range(len(importances)), [feature_names[i] for i in indices], rotation=45,
ha='right')
    plt.tight_layout()
    plt.show()

    # 상위 10개 특성 출력
    print(f"\n{model_name} - Top 10 중요 특성:")
    for i in range(10):
        print(f"{feature_names[indices[i]]}: {importances[indices[i]]:.4f}")

# Random Forest 평가
print("\n==== Random Forest 모델 평가 ====")
rf_metrics = evaluate_model_metrics(rf_best_model, X_test_rf_selected, y_test,
"Random Forest")
plot_feature_importance(rf_best_model, rf_selected_features, "Random Forest")

```

```
# XGBoost 평가
print("\n==== XGBoost 모델 평가 ====")
xgb_metrics = evaluate_model_metrics(xgb_best_model, X_test_xgb_selected, y_test,
"XGBoost")
plot_feature_importance(xgb_best_model, xgb_selected_features, "XGBoost")

# 모델 비교 시각화
metrics_df = pd.DataFrame({
    'Random Forest': rf_metrics,
    'XGBoost': xgb_metrics
})

plt.figure(figsize=(10, 6))
metrics_df.plot(kind='bar')
plt.title('모델 성능 비교')
plt.ylabel('Score')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# 모델 해석 및 시사점
print("\n==== 모델 해석 및 시사점 ====")
print("\n1. 모델 성능 분석")
print("- 정확도(Accuracy): 전체 예측 중 정확한 예측의 비율")
print("- 정밀도(Precision): 예금 가입으로 예측한 고객 중 실제 가입한 고객의 비율")
print("- 재현율(Recall): 실제 가입한 고객 중 모델이 맞게 예측한 비율")
print("- F1-Score: 정밀도와 재현율의 조화평균")
print("- ROC-AUC: 모델의 분류 성능을 나타내는 지표 (1에 가까울수록 좋음)")

print("\n2. 중요 변수 해석")
print("Random Forest 주요 변수:")
for i, feature in enumerate(rf_selected_features[:5]):
    print(f"- {feature}: 고객의 예금 가입 결정에 {i+1}번째로 중요한 영향을 미침")
```

```
print("\nXGBoost 주요 변수:")
for i, feature in enumerate(xgb_selected_features[:5]):
    print(f"- {feature}: 고객의 예금 가입 결정에 {i+1}번째로 중요한 영향을 미침")

print("\n3. 비즈니스 시사점")
print("- 가장 영향력 있는 변수들을 중심으로 마케팅 전략 수립 가능")
print("- 높은 예측 정확도를 바탕으로 효율적인 타겟 마케팅 가능")
print("- 재현율과 정밀도의 균형을 고려한 마케팅 비용 최적화 가능")
```

위 코드에서 인상적인 것은 모델 해석 및 시사점에 대한 정리다. 다음 내용을 살펴보면 다음과 같다.

==== 모델 해석 및 시사점 ===

1. 모델 성능 분석

- 정확도(Accuracy): 전체 예측 중 정확한 예측의 비율
- 정밀도(Precision): 예금 가입으로 예측한 고객 중 실제 가입한 고객의 비율
- 재현율(Recall): 실제 가입한 고객 중 모델이 맞게 예측한 비율
- F1-Score: 정밀도와 재현율의 조화평균
- ROC-AUC: 모델의 분류 성능을 나타내는 지표 (1에 가까울수록 좋음)

2. 중요 변수 해석

Random Forest 주요 변수:

- duration: 고객의 예금 가입 결정에 1번째로 중요한 영향을 미침
- housing_yes: 고객의 예금 가입 결정에 2번째로 중요한 영향을 미침
- month_may: 고객의 예금 가입 결정에 3번째로 중요한 영향을 미침
- poutcome_failure: 고객의 예금 가입 결정에 4번째로 중요한 영향을 미침
- balance: 고객의 예금 가입 결정에 5번째로 중요한 영향을 미침

XGBoost 주요 변수:

- housing_yes: 고객의 예금 가입 결정에 1번째로 중요한 영향을 미침
- poutcome_failure: 고객의 예금 가입 결정에 2번째로 중요한 영향을 미침
- poutcome_other: 고객의 예금 가입 결정에 3번째로 중요한 영향을 미침
- housing_no: 고객의 예금 가입 결정에 4번째로 중요한 영향을 미침
- loan_yes: 고객의 예금 가입 결정에 5번째로 중요한 영향을 미침

3. 비즈니스 시사점

- 가장 영향력 있는 변수들을 중심으로 마케팅 전략 수립 가능
- 높은 예측 정확도를 바탕으로 효율적인 타겟 마케팅 가능
- 재현율과 정밀도의 균형을 고려한 마케팅 비용 최적화 가능

여기에서 한 번 더 다음과 같이 프롬프트에 질의를 추가한다. 이 때 슬라이드로 요약해달라거나 또는 긴 장문으로 요약해달라고 하면 요구하는 답변 양식에 따라 정리가 될 것이니 참고하기를 바란다. 필자는 슬라이드 형태로 요약된 것을 다시 복사해서 줄글로 재작성을 요청하였다.

제안한 모델 해석 및 시사점에 대해 비즈니스 관점에서 상세하게 해석하여 주세요.

추론한 결과는 각 사용자마다 다를 수 있다. 필자의 경우 아래와 같이 추론하였다.

The screenshot shows a Jupyter Notebook environment with the following details:

- File Explorer:** Shows a project structure for "CURSOR BANK PREDICTOR" containing notebooks, data, docker, and requirements.txt.
- Terminal:** A MINGW64 terminal window titled "(main)" is active, showing the command "powershell".
- Code Editor:** A notebook cell is open with the following code and output:

```
== 모델 해석 및 시사점 ==

1. 모델 성능 분석
- 정확도(Accuracy): 전체 예측 중 정확한 예측의 비율
- 정밀도(Precision): 예금 가입으로 예측한 고객 중 실제 가입한 고객의
- 재현율(Recall): 실제 가입한 고객 중 모델이 맞게 예측한 비율
- F1-Score: 정밀도와 재현율의 조합평균
- ROC-AUC: 모델의 분류 성능을 나타내는 지표 (1에 가까울수록 좋음)
```
- Output:** The terminal shows the command "powershell" being run.
- Right Panel:** A sidebar provides information about the model's performance, mentioning duration, call length, and campaign ID.

요약한 결과는 다음과 같다.

"본 분석을 통해 예금 상품 가입에 영향을 미치는 주요 요인들을 파악했습니다. 가장 중요한 변수는 통화 시간(duration)으로, 고객과의 충분한 상담 시간 확보가 가입 성공률을 높이는 핵심 요소임이 확인되었습니다. 또한, 주택 대출 보유 여부와 이전 캠페인 결과가 중요한 예측 변수로 나타났습니다.

이러한 결과를 바탕으로, 주택 대출 보유 고객을 우선적으로 타겟팅하고, 과거 캠페인 이력을 고려한 차별화된 마케팅 전략 수립과 자동화된 타겟팅 시스템 구축이 요구됩니다. 이를 통해 마케팅 비용 절감과 향후 판매 성과를 향상시키는 청탁과 함께, 고객 경험 개선과 데이터 기반 의사결정 체계 강화라는 경영적 효과를 거둘 수 있을 것입니다.

이러한 전략적 접근을 통해 은행의 예금 상품 마케팅 효율성과 효과성을 크게 개선할 수 있을 것으로 예상됩니다.

실행 방안으로는 단기적으로 콜센터 상담원 교육 강화와 맞춤형 예금 상품 개발이 필요하며, 장기적으로는 고객 세그먼트별 마케팅 전략 수립과 자동화된 타겟팅 시스템 구축이 요구됩니다. 이를 통해 마케팅 비용 절감과 캠페인 성공률 향상이라는 정량적 효과와 함께, 고객 경험 개선과 데이터 기반 의사결정 체계 강화라는 정성적 효과를 기대할 수 있습니다. 이러한 전략적 접근을 통해 은행의 예금 상품 마케팅 효율성과 효과성을 크게 개선할 수 있을 것으로 예상됩니다.”

이제 모델을 저장하도록 한다. 모델링 파일은 FastAPI에서 불러올 수 있는 형태로 저장할 것이다. 다음과 같이 프롬프트를 작성하도록 한다.

이제 최종적으로 튜닝된 모델을 FastAPI에서 불러올 수 있는 형태로 저장하고자 합니다.

구현된 코드는 다음과 같다. 실제 models 폴더에 model_metadata.json, random_forest_model.joblib, xgboost_model.joblib 이 저장된 것을 확인할 수 있다.

```
import joblib
import json
import os
from datetime import datetime

class BankMarketingModel:
    def __init__(self, model, feature_names):
        self.model = model
        self.feature_names = feature_names
        self.creation_date = datetime.now()

    def predict(self, X):
        # 입력 데이터의 피처가 모델의 피처와 일치하는지 확인
        if not all(feature in X.columns for feature in self.feature_names):
            missing_features = set(self.feature_names) - set(X.columns)
            raise ValueError(f"Missing features: {missing_features}")

        # 필요한 피처만 선택하고 순서 맞추기
        X = X[self.feature_names]
        return self.model.predict(X)

    def predict_proba(self, X):
```

```
X = X[self.feature_names]
return self.model.predict_proba(X)

# 모델 저장을 위한 디렉토리 생성
model_dir = '../models'
os.makedirs(model_dir, exist_ok=True)

# Random Forest 모델 저장
rf_model_info = {
    'model': rf_best_model,
    'feature_names': rf_selected_features,
    'metrics': {
        'accuracy': rf_metrics['Accuracy'],
        'precision': rf_metrics['Precision'],
        'recall': rf_metrics['Recall'],
        'f1': rf_metrics['F1-Score'],
        'roc_auc': rf_metrics['ROC-AUC']
    },
    'creation_date': datetime.now().strftime('%Y-%m-%d %H:%M:%S')
}

rf_model_path = os.path.join(model_dir, 'random_forest_model.joblib')
joblib.dump(rf_model_info, rf_model_path)

# XGBoost 모델 저장
xgb_model_info = {
    'model': xgb_best_model,
    'feature_names': xgb_selected_features,
    'metrics': {
        'accuracy': xgb_metrics['Accuracy'],
        'precision': xgb_metrics['Precision'],
        'recall': xgb_metrics['Recall'],
        'f1': xgb_metrics['F1-Score'],
        'roc_auc': xgb_metrics['ROC-AUC']
    },
}
```

```

'creation_date': datetime.now().strftime('%Y-%m-%d %H:%M:%S')
}

xgb_model_path = os.path.join(model_dir, 'xgboost_model.joblib')
joblib.dump(xgb_model_info, xgb_model_path)

# 모델 메타데이터 저장
model_metadata = {
    'random_forest': {
        'path': rf_model_path,
        'metrics': rf_model_info['metrics'],
        'feature_names': rf_selected_features,
        'creation_date': rf_model_info['creation_date']
    },
    'xgboost': {
        'path': xgb_model_path,
        'metrics': xgb_model_info['metrics'],
        'feature_names': xgb_selected_features,
        'creation_date': xgb_model_info['creation_date']
    }
}

metadata_path = os.path.join(model_dir, 'model_metadata.json')
with open(metadata_path, 'w') as f:
    json.dump(model_metadata, f, indent=4)

print("모델 저장 완료!")
print(f"- Random Forest 모델: {rf_model_path}")
print(f"- XGBoost 모델: {xgb_model_path}")
print(f"- 메타데이터: {metadata_path}")

# 모델 로드 테스트
def load_model(model_path):
    model_info = joblib.load(model_path)
    return BankMarketingModel(model_info['model'], model_info['feature_names'])

```

```
# 테스트
print("\n모델 로드 테스트:")
rf_loaded = load_model(rf_model_path)
xgb_loaded = load_model(xgb_model_path)

# 샘플 데이터로 예측 테스트
sample_data = X_test.copy()
print("\n예측 테스트:")
print("Random Forest 예측:", rf_loaded.predict(sample_data[:1]))
print("XGBoost 예측:", xgb_loaded.predict(sample_data[:1]))
```

[결과]

모델 저장 완료!

- Random Forest 모델: ./models/random_forest_model.joblib
- XGBoost 모델: ./models/xgboost_model.joblib
- 메타데이터: ./models/model_metadata.json

모델 로드 테스트:

예측 테스트:

Random Forest 예측: [0]

XGBoost 예측: [0]

이번에는 임시 데이터를 생성 후 처리하도록 한다. 이 작업을 통해서 streamlit 앱을 만들 때 샘플 데이터 프레임을 어떻게 만들어야 하는지 확인할 수 있을 것이다. 프롬프트는 다음과 같이 입력하였다.

예측시 2개의 임시 데이터 생성 후 처리, 향후에 streamlit에서 입력받을 예정입니다.

구현된 코드는 다음과 같다.

```
# Random Forest와 XGBoost 모델의 필요한 피처들을 확인
rf_features = [
    "duration",
    "housing_yes",
```

```
"month_may",
"poutcome_failure",
"balance",
"marital_married",
"day",
"age",
"job_blue-collar",
"housing_no"
]

xgb_features = [
    "housing_yes",
    "poutcome_failure",
    "poutcome_other",
    "housing_no",
    "loan_yes",
    "month_jun",
    "marital_divorced",
    "month_jul",
    "month_may",
    "marital_single"
]

# 테스트용 임시 데이터 생성
sample_customers = pd.DataFrame({
    'age': [35, 52],
    'job': ['management', 'technician'],
    'marital': ['married', 'divorced'],
    'education': ['tertiary', 'secondary'],
    'default': ['no', 'no'],
    'balance': [2500, 1200],
    'housing': ['yes', 'no'],
    'loan': ['no', 'yes'],
    'contact': ['cellular', 'telephone'],
    'day': [15, 21],
```

```
'month': ['may', 'jun'],
'duration': [258, 180],
'campaign': [1, 2],
'pdays': [999, 999],
'previous': [0, 1],
'poutcome': ['unknown', 'failure']

})

def preprocess_data(df, selected_features):
    # 데이터프레임 복사
    df_encoded = df.copy()

    # 결과 데이터프레임 초기화
    result_df = pd.DataFrame(index=df_encoded.index)

    # 수치형 변수 복사
    numeric_features = ['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous']
    for feature in numeric_features:
        if feature in selected_features:
            result_df[feature] = df_encoded[feature]

    # 범주형 변수 처리
    # marital
    result_df['marital_divorced'] = (df_encoded['marital'] == 'divorced').astype(int)
    result_df['marital_married'] = (df_encoded['marital'] == 'married').astype(int)
    result_df['marital_single'] = (df_encoded['marital'] == 'single').astype(int)

    # housing
    result_df['housing_no'] = (df_encoded['housing'] == 'no').astype(int)
    result_df['housing_yes'] = (df_encoded['housing'] == 'yes').astype(int)

    # loan
    result_df['loan_no'] = (df_encoded['loan'] == 'no').astype(int)
    result_df['loan_yes'] = (df_encoded['loan'] == 'yes').astype(int)
```

```
# month
months = ['jan', 'feb', 'mar', 'apr', 'may', 'jun', 'jul', 'aug', 'sep', 'oct', 'nov', 'dec']
for month in months:
    result_df[f'month_{month}'] = (df_encoded['month'] == month).astype(int)

# poutcome
result_df['poutcome_failure'] = (df_encoded['poutcome'] == 'failure').astype(int)
result_df['poutcome_other'] = (df_encoded['poutcome'] == 'other').astype(int)
    result_df['poutcome_success'] = (df_encoded['poutcome'] ==
'success').astype(int)
    result_df['poutcome_unknown'] = (df_encoded['poutcome'] ==
'unknown').astype(int)

# job
jobs = ['admin.', 'blue-collar', 'entrepreneur', 'housemaid', 'management',
'retired', 'self-employed', 'services', 'student', 'technician',
'unemployed', 'unknown']
for job in jobs:
    result_df[f'job_{job}'] = (df_encoded['job'] == job).astype(int)

# 필요한 피처만 선택
return result_df[selected_features]

# Random Forest용 데이터 전처리
rf_processed = preprocess_data(sample_customers, rf_features)
# XGBoost용 데이터 전처리
xgb_processed = preprocess_data(sample_customers, xgb_features)

# 예측 수행
rf_predictions = rf_loaded.predict(rf_processed)
rf_probabilities = rf_loaded.predict_proba(rf_processed)

xgb_predictions = xgb_loaded.predict(xgb_processed)
xgb_probabilities = xgb_loaded.predict_proba(xgb_processed)
```

```
# 결과 출력
print("\n==== 예측 결과 ====")
for i in range(2):
    print(f"\n고객 {i+1}:")
    print("기본 정보:")
    for key, value in sample_customers.iloc[i].items():
        print(f"- {key}: {value}")

    print("\n예측 결과:")
    print(f"Random Forest: {'가입' if rf_predictions[i] == 1 else '미가입'} (확률: {rf_probabilities[i][1]:.2%})")
    print(f"XGBoost: {'가입' if xgb_predictions[i] == 1 else '미가입'} (확률: {xgb_probabilities[i][1]:.2%})")
    print(f"평균 확률: {((rf_probabilities[i][1] + xgb_probabilities[i][1]) / 2:.2%)}")

# 테스트 데이터 저장
test_data = {
    'raw_data': sample_customers.to_dict(orient='records'),
    'rf_processed': rf_processed.to_dict(orient='records'),
    'xgb_processed': xgb_processed.to_dict(orient='records'),
    'rf_features': rf_features,
    'xgb_features': xgb_features
}

test_data_path = os.path.join(model_dir, 'test_data.json')
with open(test_data_path, 'w') as f:
    json.dump(test_data, f, indent=4)

print(f"\n테스트 데이터 저장 완료: {test_data_path}")

[결과]
==== 예측 결과 ====

고객 1:
기본 정보:
```

```
- age: 35
- job: management
- marital: married
- education: tertiary
- default: no
- balance: 2500
- housing: yes
- loan: no
- contact: cellular
- day: 15
- month: may
- duration: 258
- campaign: 1
- pdays: 999
- previous: 0
- poutcome: unknown
```

예측 결과:

Random Forest: 미가입 (확률: 12.00%)

XGBoost: 가입 (확률: 79.27%)

...

XGBoost: 미가입 (확률: 8.15%)

평균 확률: 30.02%

테스트 데이터 저장 완료: ../models\test_data.json

J. Streamlit, Docker 를 활용한 배포

이전 단계에서 머신러닝 모델을 만들었다. 이번 단계에서부터는 Composer 의 강력한 기능을 경험할 수 있을 것이다. Composer 를 열고, 모델링하고 있던 노트북 파일과 프로젝트 정의서를 선택한 후, 요청한다. 먼저 CHAT 창의 설정은 다음과 같이 변경한다.



Composer 를 열고, 모델링하고 있던 노트북 파일과 프로젝트 정의서를 선택한 후 아래와 같이 요청하면 된다.

모델링이 완료되어 이제 실제 서비스를 개발하고자 합니다. 프로젝트 정의서를 참고해주세요.
코드나 파일을 생성하지 말고, 파악만 부탁할게요. 그리고 나의 응답을 기다려주세요.

마지막에 "코드나 파일을 생성하지 말고, 파악만 부탁한다. 그리고 나의 응답을 기다린다."라는 프롬프트를 붙이는 이유는 Composer 가 의욕적으로 우리의 의도와 다른 응답을 생성할 가능성을 줄이고, 효율적인 처리를 보장하기 위함이다. 단계적으로 나누어 진행한다. 응답이 길어질 것으로 예상되는 경우 사용자가 이어서 진행해달라고 할 때까지 다음 단계로 넘어가지 않는다. 이제 출력한 결과물을 확인한다.

프로젝트 정의서와 모델 개발 노트북을 검토한 결과, 다음과 같은 사항들을 파악했습니다:

1. 모델 개발 현황:

- Random Forest와 XGBoost 모델이 성공적으로 개발됨
- 모델 성능이 양호하며 중요 특성들이 식별됨
- 모델과 관련 메타데이터가 models/ 디렉토리에 저장됨

2. 다음 단계 구현 사항:

- FastAPI 기반의 백엔드 서버 개발 필요
- Streamlit을 활용한 프론트엔드 대시보드 개발 필요
- Docker를 통한 컨테이너화 구현 필요

3. 구현해야 할 주요 기능:

- 실시간 예측 API 엔드포인트
- 대화형 예측 인터페이스
- 모델 성능 및 중요 특성 시각화
- 데이터 분석 결과 대시보드

4. 기술 스택 준비 상태:

- Python 3.9+ 환경 구성 필요
- FastAPI, Streamlit 등 프레임워크 설치 필요
- Docker 환경 구성 필요

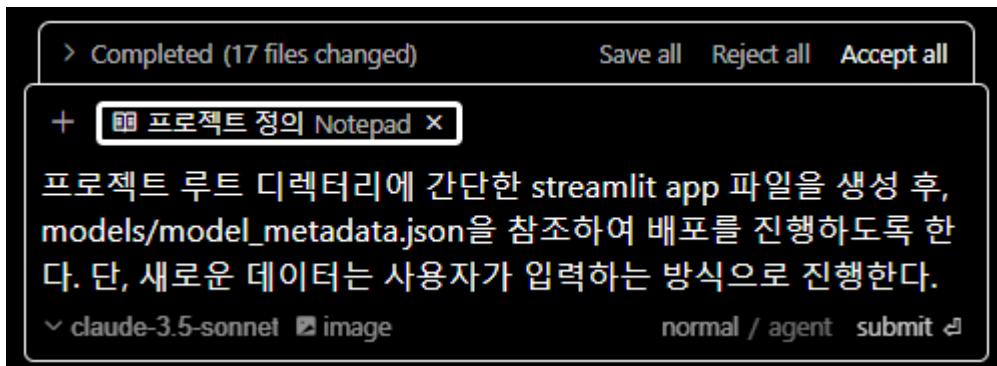
실제 서비스 개발을 시작하기 위한 방향성에 대해 조언이나 구체적인 다음 단계가 필요하시다면 말씀해 주세요.

서비스를 구현하기 위한 기본설정이 종료되었다. 이제 다음 단계부터 시작하도록 한다.

이번장에서는 streamlit 과 Dockerfile 만 구현을 하도록 한다. FastAPI 기반의 백엔드 서버개발을 하려면 FastAPI에 대한 여러 설명도 추가가 되어야 할 것 같은데, 이 부분은 책의 범위를 넘어서는 것 같아서 생략하도록 한다. Dockerfile 은 파일 작성 전 간단하게 Docker 에 대해서 설명한다.¹⁶⁵

프로젝트 루트 디렉터리에 간단한 streamlit app 파일을 생성 후,
models/model_metadata.json을 참조하여 배포를 진행하도록 한다. 단, 새로운 데이터는
사용자가 입력하는 방식으로 진행한다.

¹⁶⁵ 초기에는 FastAPI까지 코드 작성을 하였지만, 작성 및 설명해야 할 파일이 매우 많아져서 적합하지 않다고 생각했다. Cursor AI를 활용한 배포가 목적이니 익숙한 코드로 진행하기로 하였다.



실행한 화면은 다음과 같다.

| Model | Prediction | Probability (%) |
|---------------|------------|-----------------|
| Random Forest | 미가입 | 13.4% |
| XGBoost | 미가입 | 24.6% |

코드는 다음과 같다. 파일명 (app.py)

```

import streamlit as st
import json
import pandas as pd
import joblib
from typing import Dict

# 페이지 설정
st.set_page_config(
    page_title="Bank Marketing Predictor",
    page_icon="🏦",

```

```
layout="wide"
)

def load_models():
    """모델과 메타데이터 로드"""
    try:
        # 모델 로드
        rf_model_info = joblib.load('models/random_forest_model.joblib')
        xgb_model_info = joblib.load('models/xgboost_model.joblib')

        # 메타데이터 로드
        with open('models/model_metadata.json', 'r') as f:
            metadata = json.load(f)

        return rf_model_info, xgb_model_info, metadata
    except Exception as e:
        st.error(f"모델 로드 실패: {str(e)}")
        return None, None, None

def preprocess_data(df: pd.DataFrame, selected_features):
    """데이터 전처리"""
    df_encoded = df.copy()
    result_df = pd.DataFrame(index=df_encoded.index)

    # 수치형 변수 처리
    numeric_features = ['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous']
    for feature in numeric_features:
        if feature in selected_features:
            result_df[feature] = df_encoded[feature]

    # 범주형 변수 처리
    # marital
    result_df['marital_divorced'] = (df_encoded['marital'] == 'divorced').astype(int)
    result_df['marital_married'] = (df_encoded['marital'] == 'married').astype(int)
    result_df['marital_single'] = (df_encoded['marital'] == 'single').astype(int)
```

```

# housing
result_df['housing_no'] = (df_encoded['housing'] == 'no').astype(int)
result_df['housing_yes'] = (df_encoded['housing'] == 'yes').astype(int)

# loan
result_df['loan_no'] = (df_encoded['loan'] == 'no').astype(int)
result_df['loan_yes'] = (df_encoded['loan'] == 'yes').astype(int)

# month
months = ['jan', 'feb', 'mar', 'apr', 'may', 'jun', 'jul', 'aug', 'sep', 'oct', 'nov', 'dec']
for month in months:
    result_df[f'month_{month}'] = (df_encoded['month'] == month).astype(int)

# poutcome
result_df['poutcome_failure'] = (df_encoded['poutcome'] == 'failure').astype(int)
result_df['poutcome_other'] = (df_encoded['poutcome'] == 'other').astype(int)
    result_df['poutcome_success'] = (df_encoded['poutcome'] == 'success').astype(int)
    result_df['poutcome_unknown'] = (df_encoded['poutcome'] == 'unknown').astype(int)

# job
jobs = ['admin.', 'blue-collar', 'entrepreneur', 'housemaid', 'management',
        'retired', 'self-employed', 'services', 'student', 'technician',
        'unemployed', 'unknown']
for job in jobs:
    result_df[f'job_{job}'] = (df_encoded['job'] == job).astype(int)

return result_df[selected_features]

def main():
    st.title("🏦 Bank Marketing Predictor")

    # 모델 로드

```

```
rf_model_info, xgb_model_info, metadata = load_models()

if not all([rf_model_info, xgb_model_info, metadata]):
    st.error("모델을 로드할 수 없습니다.")
    return

# 사이드바에 모델 정보 표시
st.sidebar.header("모델 정보")
for model in ["random_forest", "xgboost"]:
    st.sidebar.subheader(f"{model.upper()} 성능")
    metrics = metadata[model]["metrics"]
    for metric, value in metrics.items():
        st.sidebar.metric(metric.upper(), f"{value:.3f}")

# 메인 화면에 입력 폼
st.header("고객 정보 입력")

col1, col2 = st.columns(2)

with col1:
    age = st.number_input("나이", min_value=18, max_value=100, value=35)
    job = st.selectbox(
        "직업",
        ["admin.", "blue-collar", "entrepreneur", "housemaid", "management",
         "retired", "self-employed", "services", "student", "technician",
         "unemployed", "unknown"]
    )
    marital = st.selectbox("결혼 상태", ["married", "single", "divorced"])
    education = st.selectbox("교육 수준", ["primary", "secondary", "tertiary",
                                           "unknown"])
    default = st.selectbox("신용불량 여부", ["no", "yes"])
    balance = st.number_input("계좌 잔액", value=0)
    housing = st.selectbox("주택대출 여부", ["no", "yes"])
    loan = st.selectbox("개인대출 여부", ["no", "yes"])
```

```

with col2:
    contact = st.selectbox("연락 수단", ["cellular", "telephone", "unknown"])
    day = st.number_input("마지막 연락 일자", min_value=1, max_value=31, value=15)
    month = st.selectbox(
        "마지막 연락 월",
        ["jan", "feb", "mar", "apr", "may", "jun",
         "jul", "aug", "sep", "oct", "nov", "dec"]
    )
    duration = st.number_input("마지막 통화 시간(초)", min_value=0, value=258)
    campaign = st.number_input("캠페인 연락 횟수", min_value=1, value=1)
    pdays = st.number_input("이전 캠페인 후 경과일", value=999)
    previous = st.number_input("이전 캠페인 연락 횟수", min_value=0, value=0)
    poutcome = st.selectbox("이전 캠페인 결과", ["unknown", "failure", "success", "other"])

if st.button("예측하기"):
    # 입력 데이터 처리
    input_data = pd.DataFrame([
        {"age": age, "job": job, "marital": marital, "education": education,
         "default": default, "balance": balance, "housing": housing, "loan": loan,
         "contact": contact, "day": day, "month": month, "duration": duration,
         "campaign": campaign, "pdays": pdays, "previous": previous, "poutcome": poutcome}
    ])

    # 각 모델용 데이터 전처리
    rf_processed = preprocess_data(input_data, rf_model_info['feature_names'])
    xgb_processed = preprocess_data(input_data, xgb_model_info['feature_names'])

    # 예측 수행
    rf_prob = rf_model_info['model'].predict_proba(rf_processed)[0][1]
    xgb_prob = xgb_model_info['model'].predict_proba(xgb_processed)[0][1]

```

```
# 앙상블 확률 계산
ensemble_prob = (rf_prob + xgb_prob) / 2

# 결과 표시
st.header("예측 결과")
col1, col2, col3 = st.columns(3)

with col1:
    st.metric(
        "Random Forest",
        "가입" if rf_prob >= 0.5 else "미가입",
        f"{rf_prob:.1%}"
    )

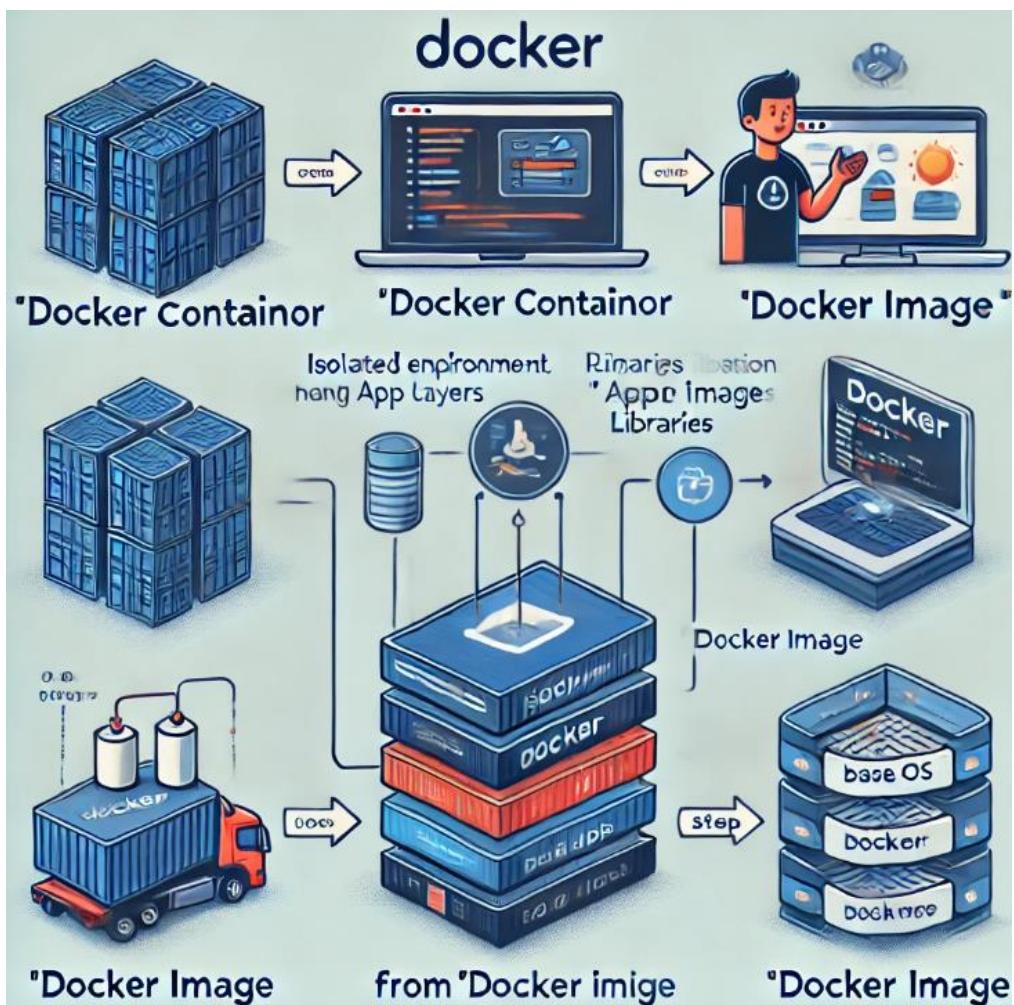
with col2:
    st.metric(
        "XGBoost",
        "가입" if xgb_prob >= 0.5 else "미가입",
        f"{xgb_prob:.1%}"
    )

with col3:
    st.metric(
        "앙상블 예측 확률",
        f"{ensemble_prob:.1%}"
    )

if __name__ == "__main__":
    main()
```

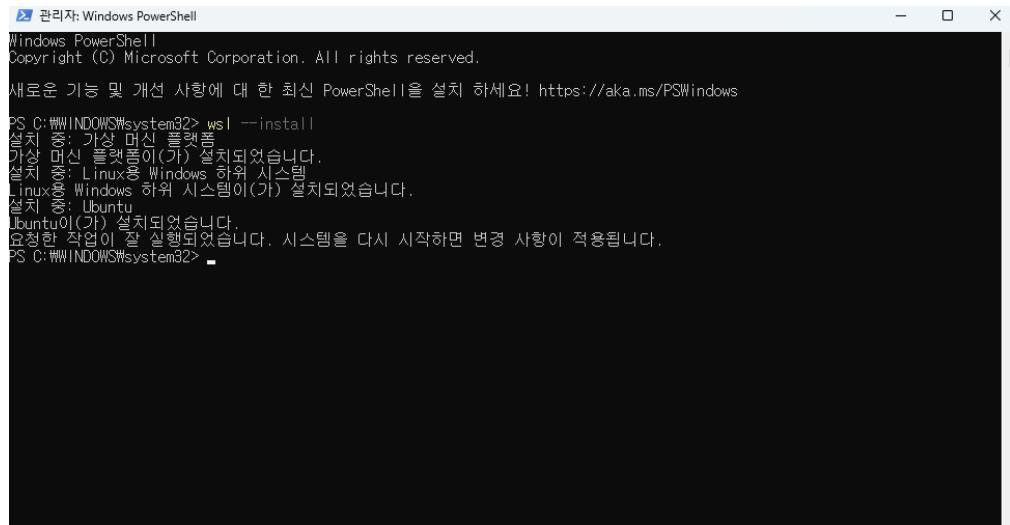
이제 마지막 단계다. 우선 Docker에 대해 간단하게 설명한다. 우선 Docker는 **Docker**는 컨테이너 기반의 가상화 기술로, 애플리케이션과 그 의존성을 하나의 패키지로 묶어 동일한 환경에서 실행할 수 있도록 도와준다. 이를 통해 개발, 테스트, 배포 환경에서의 일관성을 유지하며, "Build once, run anywhere"를 실현할 수 있다. 주요 개념은 크게 3 가지에 대한 기본 개념만 이해하면 된다.

| 주요개념 | 설명 |
|-----------------------|---|
| 컨테이너
(Container) | <ul style="list-style-type: none"> - 경량화된 가상 환경, 애플리케이션과 필요한 라이브러리만 포함한다. - 호스트 OS 커널을 공유하여 일반적인 가상 머신보다 빠르고 자원을 효율적으로 사용 |
| 이미지
(Image) | <ul style="list-style-type: none"> - 컨테이너의 실행 환경과 애플리케이션을 정의한 읽기 전용 템플릿 - 여러 컨테이너를 동일한 이미지를 기반으로 실행 |
| 도커 허브
(Docker Hub) | <ul style="list-style-type: none"> - 도커 이미지의 저장소로, 공개 및 개인용 이미지를 관리 - 사용자는 필요한 이미지를 검색하고 다운로드하거나, 직접 만든 이미지를 업로드 |



이번에는 Windows에 Docker를 설치하도록 한다. 설치 전에 가상환경화가 진행되어야 한다. PowerShell을 관리자 모드로 실행 후 다음과 같이 코드를 실행한다.

```
PS C:\WINDOWS\system32> wsl --install
설치 중: 가상 머신 플랫폼
가상 머신 플랫폼이(가) 설치되었습니다.
설치 중: Linux용 Windows 하위 시스템
Linux용 Windows 하위 시스템이(가) 설치되었습니다.
설치 중: Ubuntu
Ubuntu이(가) 설치되었습니다.
요청한 작업이 잘 실행되었습니다. 시스템을 다시 시작하면 변경 사항이 적용됩니다.
```



설치 완료 후, wsl 버전 기본값을 2로 변경한다.

```
PS C:\WINDOWS\system32> wsl --set-default-version 2
WSL 2와의 주요 차이점에 대한 자세한 내용은 https://aka.ms/wsl2를 참조하세요

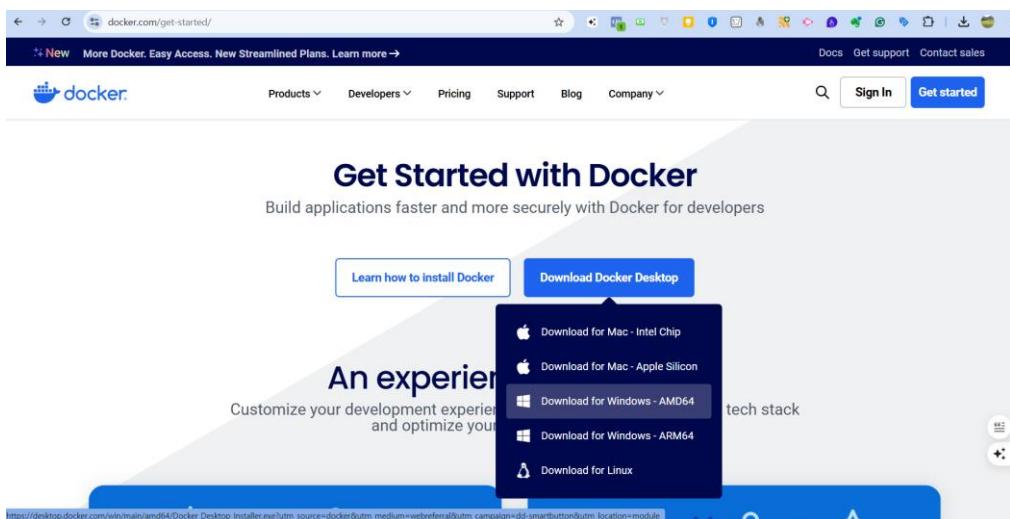
작업을 완료했습니다.
```

이번에는 wsl 버전을 확인한다.

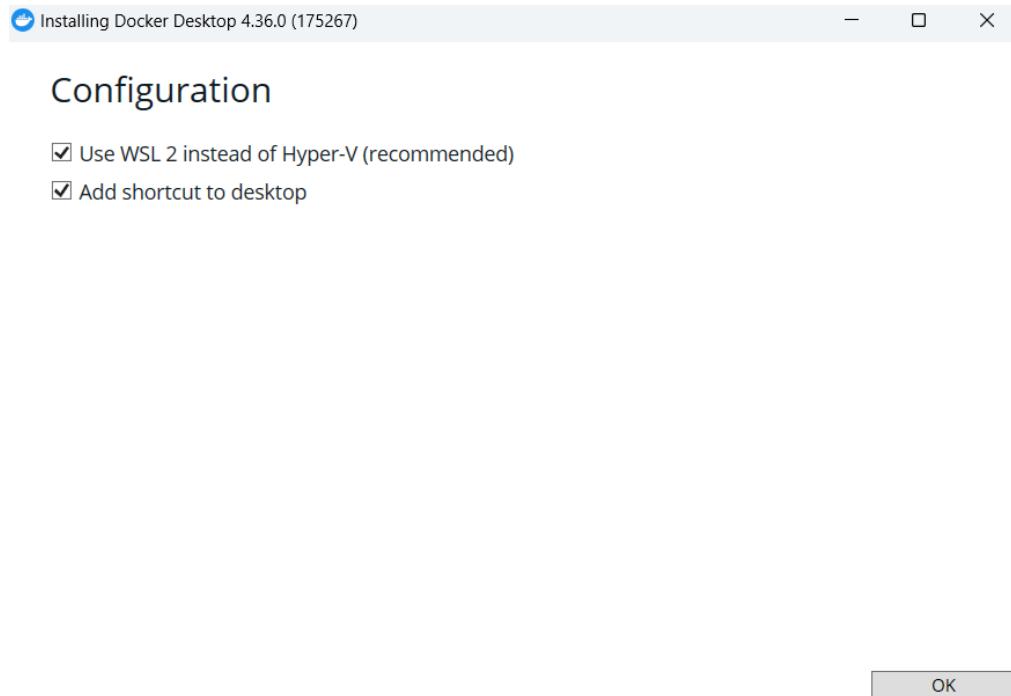
```
PS C:\WINDOWS\system32> wsl --version
WSL 버전: 2.3.26.0
```

커널 버전: 5.15.167.4-1
WSLg 버전: 1.0.65
MSRDC 버전: 1.2.5620
Direct3D 버전: 1.611.1-81528511
DXCore 버전: 10.0.26100.1-240331-1435.ge-release
Windows 버전: 10.0.22621.4317

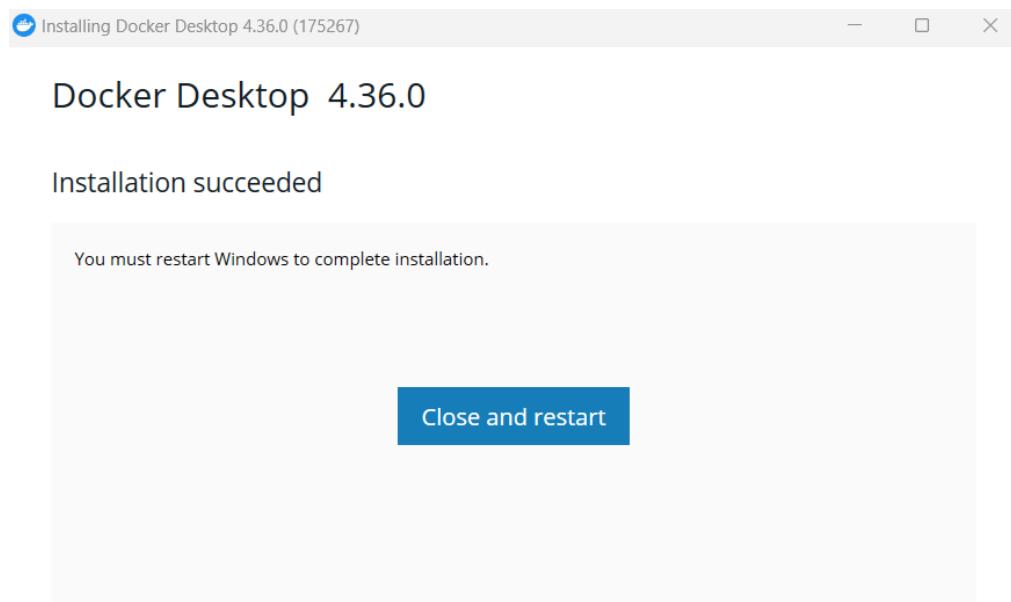
이제 Docker 웹 사이트(<https://www.docker.com/>)에서 다운로드를 받는다. 인텔
유저는 AMD 를 설치하면 된다.



다운로드 받은 파일을 관리자 권한으로 실행 후 설치를 완료한다.



설치가 완료되면 재 시작을 한다.



이제 마지막으로 Docker 를 실행을 위한 파일을 만들고 Docker 이미지를 빌드하여 실행하도록 한다. 프롬프트는 다음과 같이 작성하였다.

Streamlit app 배포를 위한 Docker Compose 파일 생성 후, Docker 실행까지 해주세요.

결과 화면은 다음과 같다.

The screenshot shows a terminal window titled "cursor_bank_predictor" with the following content:

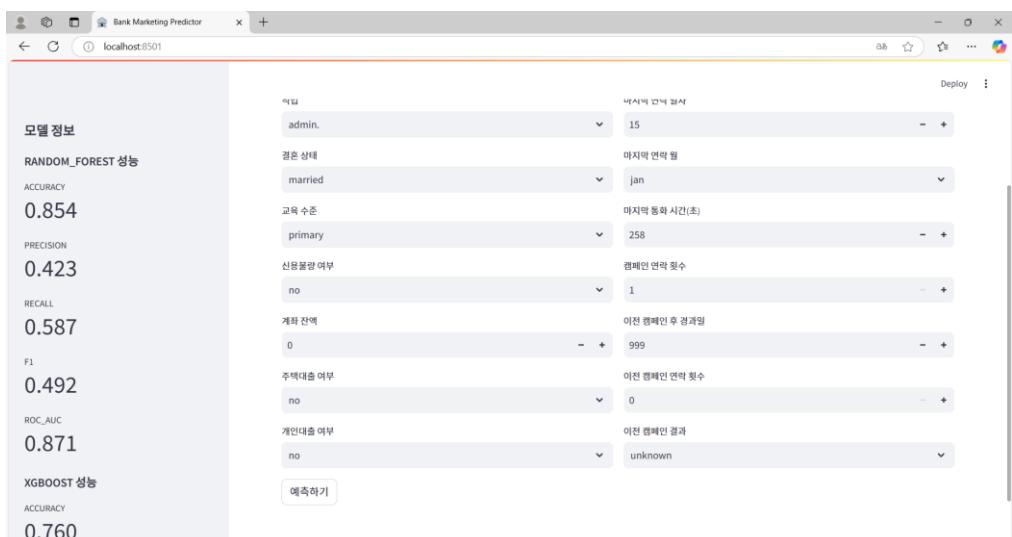
```
FROM python:3.9-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY app.py .
COPY models/ m
```

Below the terminal, the output shows the execution of the Dockerfile:

```
skttop-linuX [5/6] COPY app.py 0.3s
=> [streamlit 5/6] COPY app.py 0.3s
=> [streamlit 6/6] COPY models 0.2s
=> [streamlit] exporting to i 14.2s
=> => exporting layers 14.2s
```

To the right of the terminal, there is a sidebar with various tools like Chat, Composer, and Bug Finder, and a message about Docker execution.

이제 브라우저에서 접속(<http://localhost:8501>)을 하면 앞서 실행한 화면과 동일하게 나올 것이다.



실행이 정상적으로 되었다면 각 파일에 대해 간단하게 설명하면 다음과 같다. Docker Compose 를 실행하면 Dockerfile 을 기반으로 이미지를 빌드하고, 컨테이너를 생성 및 실행한다. 컨테이너는 호스트의 ./models 디렉토리 내용을 /app/models 로 마운트하여 애플리케이션에서 모델 파일을 사용할 수 있게 한다. 애플리케이션은 0.0.0.0:8501 에서 실행되며, 로컬 브라우저에서 http://localhost:8501 로 접근할 수 있다.

파일명 : docker-compose.yml

```
FROM python:3.9-slim

WORKDIR /app

# 필요한 패키지 설치
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# 애플리케이션 코드 복사
COPY app.py .
COPY models/ models/

# Streamlit 실행
EXPOSE 8501
CMD ["streamlit", "run", "app.py"]
```

이번에는 Dockerfile 에 대해 설명하면 다음과 같다. 이 Dockerfile 은 Python 기반 Streamlit 애플리케이션을 컨테이너 환경에서 실행 가능하도록 설정한다. 컨테이너를 통해 애플리케이션을 배포하면 일관된 환경에서 실행할 수 있어 의존성 문제를 최소화하고 배포 과정을 단순화할 수 있다.

파일명 : Dockerfile

```
FROM python:3.9-slim

WORKDIR /app

# 필요한 패키지 설치
COPY requirements.txt .
```

```
RUN pip install --no-cache-dir -r requirements.txt

# 애플리케이션 코드 복사
COPY app.py .

COPY models/ models/

# Streamlit 실행
EXPOSE 8501
CMD ["streamlit", "run", "app.py"]
```

이번에는 Dockerfile 에 대해 설명하면 다음과 같다. 이 Dockerfile 은 Python 기반 Streamlit 애플리케이션을 컨테이너 환경에서 실행 가능하도록 설정한다. 컨테이너를 통해 애플리케이션을 배포하면 일관된 환경에서 실행할 수 있어 의존성 문제를 최소화하고 배포 과정을 단순화할 수 있다.