

# Tidymodel의 기본 철학: Chatgpt시대의 한계와 가능성을 중심으로



공병규

유타대학교 아시아 캠퍼스,  
경영대 정보시스템학과  
조교수

Byoung-gyu.Gong@utah.edu


## 발표의 동기

Tidymodels에 대한 책을 번역하다가, ChatGPT로 분석을 해보고 현타가 옴

- 근원적인 질문: 데이터 분석가라는 직업은 미래에 살아 남을 것인가?
- 당장의 현실적 질문: Tidymodels를 한국에 널리 소개하고 보급하는게 아직 유효하고 의미 있는 일인가?
- 즉, Tidymodels는 ChatGPT의 시대에 어떠한 의미를 가지는가? 왜 내가 이걸 해야하지?

소결론: 예전만큼은 아니지만 아직도 시장에 분명 필요로 하는 사람들이 있을 것 같다.

# 데이터 분석을 해보자



## R Core 언어의 철학(1)

### 가능성


“한 가지 결과를 가져오는데 여러가지 다양한 방식이 있을 수 있다”.

- 개발자 편의주의이기도 하지만, 전체 개발 참여자의 폭을 넓혀서 생태계 확장에 기여.
- 프로그래밍의 유연성과 다양성을 가져옴.
- 일종의 오픈 생태계.

### 한 계

지나친 확장성은 생태계를 교란하고 혼탁하게 만들 수 있음

- 사용자들은 지나치게 다양한 Syntax 옵션을 두고 혼란스러워 하거나 고민하게 됨.
- Naming Convention이 통일되지 않아 불필요한 중복 함수가 지속적으로 발생.



## R Core 언어의 철학(2)

### 가능성

#### “Computational efficiency”.

- 다양한 R의 코어 언어 Syntax들은 컴퓨팅적으로 효율적이게 작동함.
- 이러한 언어들은 성능면에서 매우 효율적이기도 함.
- 소수의 개발자와 분석가들이 활용하고 발전시켜 나가기에 유리한 구조.

### 한 계

#### “Computationally efficient, but not human readable.

- 성능이 뛰어난 코드들이라도 결국 인간이 언어적으로 이해하기 힘든 형식이라면 활용 가능성이 떨어져.
- 개발자 중심적인 사고로는 분석의 대중화를 이룰 수 없음.

## R Core 언어의 철학(예시)

“지나친 확장성이 초래한 중복 함수의 부작용.”

약간씩 실제 사용문법과 세부 사항은 다르지만, 궁극적으로 유사한 기능을 수행

Function	Package	Code
<code>lda()</code>	MASS	<code>predict(object)</code>
<code>glm()</code>	stats	<code>predict(object, type = "response")</code>
<code>gbm()</code>	gbm	<code>predict(object, type = "response", n.trees)</code>
<code>mda()</code>	mda	<code>predict(object, type = "posterior")</code>
<code>rpart()</code>	rpart	<code>predict(object, type = "prob")</code>
<code>various</code>	RWeka	<code>predict(object, type = "probability")</code>
<code>logitboost()</code>	LogitBoost	<code>predict(object, type = "raw", nIter)</code>
<code>pamr.train()</code>	pamr	<code>pamr.predict(object, type = "posterior")</code>

# 파이썬과 R을 비교해보자

# R vs Python

## R Core

```

data <- read.csv("data.csv")

# 데이터 분리 (훈련 및 테스트 세트)
set.seed(42)
train_index <- sample(1:nrow(data), 0.8 * nrow(data))
train_data <- data[train_index, ]
test_data <- data[-train_index, ]

# 데이터 전처리 (표준화)
train_data$Gr_Liv_Area <- scale(train_data$Gr_Liv_Area)
train_data$Garage_Area <- scale(train_data$Garage_Area)
train_data$Year_Built <- scale(train_data$Year_Built)

test_data$Gr_Liv_Area <- scale(test_data$Gr_Liv_Area)
test_data$Garage_Area <- scale(test_data$Garage_Area)
test_data$Year_Built <- scale(test_data$Year_Built)

# 선형 회귀 모델 생성 및 훈련
model <- lm(Sale_Price ~ Gr_Liv_Area + Garage_Area + Year_Built,
            data = train_data)

# 예측 수행
y_pred <- predict(model, newdata = test_data)

# R² 계산
sst <- sum((test_data$Sale_Price - mean(test_data$Sale_Price))^2)
sse <- sum((test_data$Sale_Price - y_pred)^2)
r2 <- 1 - sse / sst

# RMSE 계산
rmse <- sqrt(mean((test_data$Sale_Price - y_pred)^2))

# 결과 출력

```



## Python Scikit-learn

```

data = pd.read_csv("data.csv")

# 독립 변수와 종속 변수 설정
X = data[['Gr_Liv_Area', 'Garage_Area', 'Year_Built']]
y = data['Sale_Price']

# 데이터 분리 (훈련 및 테스트 세트)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 데이터 전처리 (표준화)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 선형 회귀 모델 생성 및 훈련
model = LinearRegression()
model.fit(X_train_scaled, y_train)


# 예측 수행
y_pred = model.predict(X_test_scaled)

# R² 및 RMSE 계산
r2 = r2_score(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

print(f"Python: R² = {r2}, RMSE = {rmse}")

```





## R Core의 비교 우위

-Human Readable  
Fitting Function-

### R Core

- 별도의 x, y 설정 없이 모델 피팅 시에 설정과 피팅을 동시 수행
- 실제 회귀분석의 수식을 그대로 옮겨 놓은 듯한 syntax: Human Readable.

```
model <- lm(Sale_Price ~ Gr_Liv_Area + Garage_Area + Year_Built, data)
```

### Python Scikit-Learn

- 시작부터 x와 y를 설정해주어야 함
- 모델 피팅 부분이 다소 추상적으로 설정됨.

```
X = data[['Gr_Liv_Area', 'Garage_Area', 'Year_Built']]  
y = data['Sale_Price']  
  
model = LinearRegression()  
model.fit(X_train_scaled, y_train)
```

## Python Scikit-learn의 비교 우위

-Human Readable Naming  
Convention-

### R Core

- Train과 test데이터 생성 과정에 개입하는 함수에서 함수의 기능을 유추하기 불가능

```
# 데이터 분리 (훈련 및 테스트 세트)
set.seed(42)
train_index <- sample(1:nrow(data), 0.8 * nrow(data))
train_data <- data[train_index, ]
test_data <- data[-train_index, ]
```

### Python Scikit-Learn

- train\_test\_split이라는 naming convention은 human readability를 고려하고 있음.
- 전혀 지식이 없는 사람이 보아도 그 기능을 유추가능
- 이는 Python 언어 전반의 특징에 해당하며 상당히 일관성 있게 적용.

```
# 데이터 분리 (훈련 및 테스트 세트)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_st
```

## Python Scikit-learn의 비교 우위

-확장성-

## R Core

- 변수 표준화 과정의 복잡성과 많은 비효율성(물론 고급 사용자들은 극복가능하지만)

```
# 데이터 전처리 (표준화)
train_data$Gr_Liv_Area <- scale(train_data$Gr_Liv_Area)
train_data$Garage_Area <- scale(train_data$Garage_Area)
train_data$Year_Built <- scale(train_data$Year_Built)

test_data$Gr_Liv_Area <- scale(test_data$Gr_Liv_Area)
test_data$Garage_Area <- scale(test_data$Garage_Area)
test_data$Year_Built <- scale(test_data$Year_Built)
```

## Python Scikit-Learn

- 기존 코드에 단 한 줄 추가로 추가적인 작업을 수행가능.
- 데이터 전처리를 위해 별도의 객체 생성을 할 필요가 없음.

```
# 데이터 전처리 (표준화)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

# Tidy Universe

## 기본 철학



### Tidyverse의 출현과 철학

- Design for Human-
- 무한한 확장성-
- 통일성-

At a high level, the tidyverse is a language for solving data science challenges with R code. **Its primary goal is to facilitate a conversation between a human and a computer about data.** Less abstractly, the tidyverse is a collection of R packages that share a high-level design philosophy and low-level grammar and data structures, so that learning one package makes it easier to learn the next.

### 인간과 기계의 대화를 촉진하는 것이 목표.

- 사용자들이 이해하기 쉬운 인간의 언어에 가까운 언어 사용.
- 추상성 최소화 내지는 배제.
- 디자인 우선 주의. 성능보다 일단 쉽게 읽고 사용할 수 있는 언어를 추구.

### 동시에 추가적인 작업을 더하기 쉽게 구조화.

- 패키지 별로 파편화된 syntax 구조의 통일을 추구.

## 코드예시

### Tidyverse의 출현과 철학

- Design for Human-
- 무한한 확장성-
- 통일성-

```
result <- data %>%  
  filter(Sale_Price > 100000) %>%  
  select(Sale_Price, Gr_Liv_Area, Garage_Area, Year_Built) %>%  
  mutate(House_Age = 2024 - Year_Built) %>%  
  group_by(Gr_Liv_Area) %>%  
  summarize( Avg_Sale_Price =  
    mean(Sale_Price, na.rm = TRUE),  
    Avg_Garage_Area =  
    mean(Garage_Area, na.rm = TRUE),  
    Avg_House_Age =  
    mean(House_Age, na.rm = TRUE) ) %>%  
  arrange(desc(Avg_Sale_Price))
```

# Tidymodels

## Tidymodels의 철학

### -Design for Human

```
# 데이터 로드
data <- read.csv("파일경로/ames.csv")

# 데이터 분리 (훈련 및 테스트 세트)
set.seed(42)
data_split <- initial_split(data, prop = 0.8)
train_data <- training(data_split)
test_data <- testing(data_split)

# Recipe 생성 (데이터 전처리 및 표준화)
rec <- recipe(Sale_Price ~ Gr_Liv_Area + Garage_Area + Year_Built, data = train_data)
  step_normalize(all_predictors())

# 모델 정의 (선형 회귀)
lm_model <- linear_reg() %>%
  set_engine("lm")

# Workflow 생성 (Recipe와 모델 연결)
lm_workflow <- workflow() %>%
  add_recipe(rec) %>%
  add_model(lm_model)

# 모델 훈련
lm_fit <- lm_workflow %>%
  fit(data = train_data)

# 예측 수행
lm_predictions <- predict(lm_fit, new_data = test_data) %>%
  bind_cols(test_data)

# R² 및 RMSE 계산
metrics <- lm_predictions %>%
  metrics(truth = Sale_Price, estimated = .pred)
```

함수명이 길어지더라도  
확실히 인간의 언어에  
가까운 naming을 사용.

코드를 말그대로 읽을 수 있게  
구성.

“If a practitioner were to  
include these names in a plot  
or table, would the people  
viewing those results  
understand the name?”



## Tidyverse의 출현과 철학

### -확장성

```
# 데이터 로드
data <- read.csv("파일경로/ames.csv")

# 데이터 분리 (훈련 및 테스트 세트)
set.seed(42)
data_split <- initial_split(data, prop = 0.8)
train_data <- training(data_split)
test_data <- testing(data_split)

# Recipe 생성 (데이터 정리 및 표준화)
rec <- recipe(Sale_Price ~ Gr_Liv_Area + Garage_Area + Year_Built, data = train_data)
  step_normalize(all_predictors())

# 모델 정의 (선형 회귀)
lm_model <- linear_reg() %>%
  set_engine("lm")

# Workflow 생성 (Recipe와 모델 연결)
lm_workflow <- workflow() %>%
  add_recipe(rec) %>%
  add_model(lm_model)

# 모델 훈련
lm_fit <- lm_workflow %>%
  fit(data = train_data)

# 예측 수행
lm_predictions <- predict(lm_fit, new_data = test_data) %>%
  bind_cols(test_data)

# R² 및 RMSE 계산
metrics <- lm_predictions %>%
  metrics(truth = Sale_Price, estimate = .pred)
```

Recipe와 Workflow라는 특유 객체를 통해 무한한 확장성과 readability를 동시에 확보하고 있음.

물론 recipe와 workflow의 문법을 추가로 배워야하는 번거로움이 초기 비용으로 발생하나 확장을 지속할 수록 그러한 비용을 상쇄 가능.

## Tidyverse의 출현과 철학

### -통일성

```
# 데이터 로드
data <- read.csv("파일경로/ames.csv")

# 데이터 분리 (훈련 및 테스트 세트)
set.seed(42)
data_split <- initial_split(data, prop = 0.8)
train_data <- training(data_split)
test_data <- testing(data_split)

# Recipe 생성 (데이터 정리 및 표준화)
rec <- recipe(Sale_Price ~ Gr_Liv_Area + Garage_Area + Year_Built, data = train_data)
  step_normalize(all_predictors())

# 모델 정의 (선형 회귀)
lm_model <- linear_reg() %>%
  set_engine("lm")

# Workflow 생성 (Recipe와 모델 연결)
lm_workflow <- workflow() %>%
  add_recipe(rec) %>%
  add_model(lm_model)

# 모델 훈련
lm_fit <- lm_workflow %>%
  fit(data = train_data)

# 예측 수행
lm_predictions <- predict(lm_fit, new_data = test_data) %>%
  bind_cols(test_data)

# R² 및 RMSE 계산
metrics <- lm_predictions %>%
  metrics(truth = Sale_Price, estimate = .pred)
```

Recipe와 Workflow라는 특유 객체를 통해 무한한 확장성과 readability를 동시에 확보하고 있음.

물론 recipe와 workflow의 문법을 추가로 배워야하는 번거로움이 초기 비용으로 발생하나 확장을 지속할 수록 그러한 비용을 상쇄 가능.

# ChatGPT로 분석하기



## ChatGPT로 분석하기 - Tidymodels가 필요 없는 100가지 이유

### 1. 전혀 통계적 지식이 없는 사람의 경우

## 해줘분석


그냥 단순히 “분석해줘” 혹은 “insight”를 도출해줘 라는 프롬프트 명령문만으로도 EDA를 끝낼 수 있다.  
제대로만 되면 상당한 수준의 분석을 단 한마디로 할 수 있다.

## 문제점

### 재현가능성

- 아무리 프롬프트 엔지니어링이 발달했다고 하나, 여전히 Generative AI는 기존의 직접 코딩보다 재현 가능성이 낮다.
- 동일한 요구문을 여러 번 반복 입력하였을 때, 결과가 조금씩 다르게 나타나며, 이러한 차이를 보정하기 위해서는 보다 복잡한 프롬프트 전략을 짜야 한다.
- 이러한 재현가능성의 문제는 결국 예측 가능성과 신뢰도의 하향으로 이어진다.





### Data Analyst

By ChatGPT

Drop in any files and I can help analyze and visualize your data.

**Other**  
Category


**Capabilities**

- ✓ Code Interpreter & Data Analysis

**Ratings**

Not enough ratings yet

**More by ChatGPT**




**Web Browser**  
I can browse the web to help you gather information or conduct research  
By ChatGPT

[Start Chat](#)



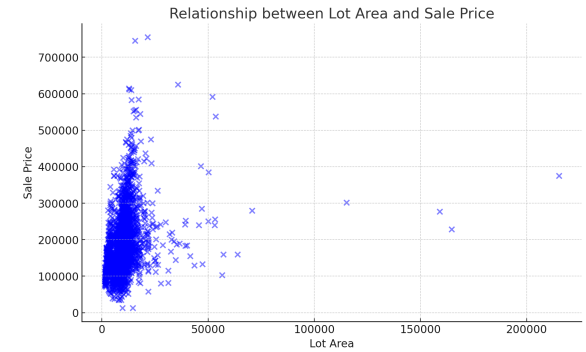
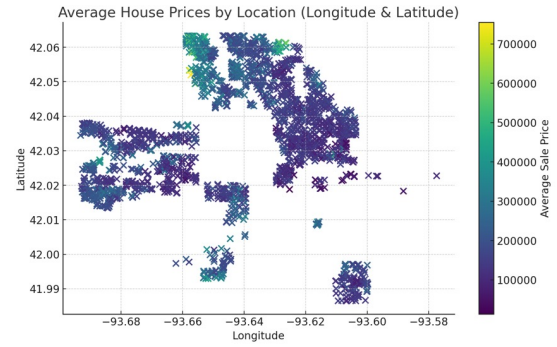
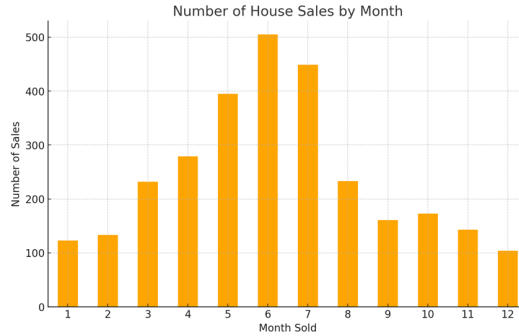
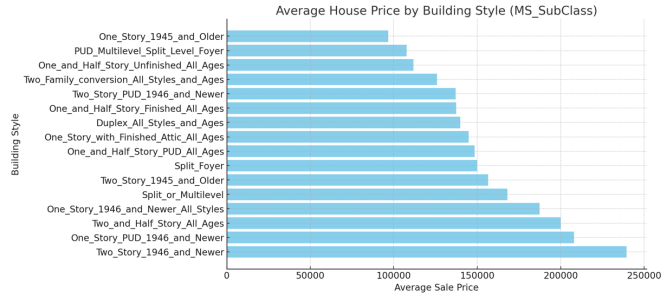
**ames.csv**  
Spreadsheet

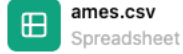
 첨부된 데이터를 가지고 비즈니스 전략을 짜고 싶어. 어떠한 insight가 도출되는지 분석해줘.



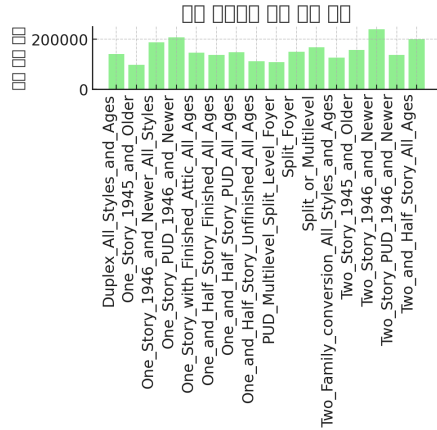
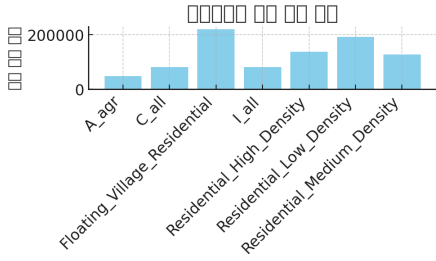
ames.csv  
 Spreadsheet

📄 첨부된 데이터를 가지고 비즈니스 전략을 짜고 싶어. 어떠한 insight가 도출되는지 분석해줘.





📄 첨부된 데이터를 가지고 비즈니스 전략을 짜고 싶어. 어떠한 insight가 도출되는지 분석해줘.



주택 판매 가격(Sale\_Price)과 상관관계가 높은 상위 10개의 변수는 다음과 같습니다:

- Gr\_Liv\_Area** (거주 가능 면적): 상관계수 0.71 – 가장 큰 영향을 미치는 변수로, 주택의 실내 거주 공간 면적이 클수록 판매 가격이 높아지는 경향이 있습니다.
- Garage\_Cars** (차고에 주차 가능한 차량 수): 상관계수 0.65 – 차고에 주차 가능한 차량 수가 많을수록 가격이 상승합니다.
- Garage\_Area** (차고 면적): 상관계수 0.64 – 차고 면적 또한 판매 가격과 밀접한 연관이 있습니다.
- Total\_Bsmt\_SF** (지하실 면적): 상관계수 0.63 – 지하실 면적이 클수록 가격이 높습니다.
- First\_Flr\_SF** (1층 면적): 상관계수 0.62 – 1층 면적 역시 중요한 요인입니다.
- Year\_Built** (건축 연도): 상관계수 0.56 – 주택이 최근에 지어질수록 가격이 높아지는 경향이 있습니다.
- Full\_Bath** (완전 욕실 수): 상관계수 0.55 – 욕실의 수가 많을수록 가격에 긍정적인 영향을 줍니다.
- Year\_Remod\_Add** (리모델링 연도): 상관계수 0.53 – 리모델링을 최근에 했을 경우 가격이 상승합니다.
- Mas\_Vnr\_Area** (벽돌 베니어 면적): 상관계수 0.50 – 벽돌 베니어로 장식된 면적이 클수록 가격이 상승합니다.



## ChatGPT로 분석하기- Tidymodels가 필요 없는 100가지 이유

### 1. 전혀 통계적 지식이 없는 사람의 경우

## 해줘분석

그냥 단순히 “분석해줘” 혹은 “insight”를 도출해줘 라는 프롬프트 명령문만으로도 EDA를 끝낼 수 있다.  
제대로만 되면 상당한 수준의 분석을 단 한마디로 할 수 있다.

## 문제점

### 재현가능성

- 이러한 재현가능성 문제로 인해서 물론 일반적이지 않지만 전혀 통계적인 분석 방법에 대한 사전 지식이 없는 사람은 물론 어느 정도 결론 도출은 가능하나 신뢰성 높은 분석을 지속하기에는 한계가 있을 수밖에 없다.

### 고급 통계 구현

- 단순히 “분석해줘”라는 명령문으로 고급 통계 분석에 접근하지 않으며, 어떠한 통계적인 방법이 필요한지 프롬프트 입력자가 스스로 알고 있어야한다.





## 해썬코딩

### ChatGPT로 분석하기- Tidymodels가 필요 없는 100가지 이유

#### 2. 통계적인 지식은 해박하나 코딩에 대한 지식은 부족한 경우

구체적으로 어떠한 방향으로 통계분석을 진행할지 프롬프트로 입력할 경우 상당한 수준의 코딩 실력을 뽐내며 정확히 분석을 진행해준다.

#### 재현가능성

- 여전히 재현 가능성의 문제가 남아있다. 매번 시도할 때마다 약간씩 대답이 다를 것이다.
- 그러나 프롬프팅의 구체성이 높아질 수록 이러한 문제를 피해갈 수 있다.
- 또한 코드 자체가 한번 생성되면 그 코드 자체로 반복 재생산 가능하기 때문에 코드를 생성할 경우 연구나 분석 수준에서의 재현가능성 문제는 사라지게 된다.
- 즉 코드의 재현가능성은 문제가 될지 모르지만 분석과 연구의 재현가능성 문제는 사라지게 된다.

## 해썬코딩 + 설명

# ChatGPT로 분석하기- Tidymodels가 필요 없는 100가지 이유

## 2. 통계적인 지식은 해박하나 코딩에 대한 지식은 부족한 경우

```
set.seed(42)  
train_index <- sample(1:nrow(data), 0.8 * nrow(data))  
train_data <- data[train_index, ]  
test_data <- data[-train_index, ]
```

- 데이터를 나누는 이유: 우리가 데이터를 사용해 모델(기계가 배우는 방법)을 만들려면 **훈련 데이터**와 **테스트 데이터**가 필요합니다. 훈련 데이터는 모델이 배우는 데 사용되고, 테스트 데이터는 모델이 얼마나 잘 배웠는지 시험하는 데 사용됩니다.

심지어 요청할 경우 매우 친절하게 코드에 대한 주석도 달아주어서 패키지 매뉴얼을 찾아볼 필요가 없게 만들어줍니다.

1. `set.seed(42)`: 이 명령어는 난이자를 무작위로 섞을 때 매번 똑같은 결과를 얻기 위해 사용됩니다. 숫자 42는 아무 숫자나 넣을 수 있습니다.
2. `train_index <- sample(1:nrow(data), 0.8 * nrow(data))`:
  - `nrow(data)`: 데이터의 총 행(row) 개수를 셉니다. 예를 들어, 데이터에 100개의 집이 있다면 `nrow(data)`는 100입니다.
  - `sample(1:nrow(data), 0.8 * nrow(data))`: 이 부분은 데이터의 80%를 무작위로 뽑아냅니다. `0.8 * nrow(data)`는 데이터의 80%를 계산하는 것입니다.
  - `train_index`: 훈련 데이터로 사용할 행 번호들이 여기에 저장됩니다.
3. `train_data <- data[train_index, ]`: 이 명령어는 데이터를 `train_index`에서 선택된 행(즉, 80%)으로 구성된 훈련 데이터를 만듭니다.
4. `test_data <- data[-train_index, ]`: 이 부분은 훈련 데이터에 선택되지 않은 나머지 20%를 테스트 데이터로 만듭니다. `-train_index`는 `train_index`에 해당하지 않는 행을 의미합니다.



## ChatGPT로 분석하기- Tidymodels가 필요 없는 100가지 이유

### 소결론

코딩의 죽음을 다시 목격할 수 있었다.

- Human Readable Code는 ChatGPT가 가장 잘 구현한다.
- 심지어 기본적인 통계분석은 파이썬으로 해주는데, 이는 파이썬 언어를 잘 몰라도 분석을 이해하게 해준다.
- 파이썬 코드를 심지어 R코드로 그대로 번역해주기도 하기 때문에 통합성의 문제를 간단히 해결해 버린다.
- 코딩을 잘 모르는 분석가도 친절한 설명을 들으며 천천히 코드를 읽고 매우 기본적인 부분은 이해할 수 있다.
- 즉, Tidyverse와 Tidymodels의 기본 철학은 ChatGPT가 대부분 구현하였고 혁신을 이루어 낸 것이 맞다.

# Tidymodels 살리기

# ChatGPT로 분석하기- Tidymodels가 여전히 필요한 단 한가지 이유

보이시나요...

```
# 데이터 분리 (훈련 및 테스트 세트)
# 이제 데이터를 두 개의 상자로 나눌 겁니다. 하나는 '훈련 데이터', 다른 하나는 '테스트 데이터'입니다.
# '훈련 데이터'는 우리가 모델을 훈련시키는 데 사용합니다. 모델이 데이터를 보고 배울 수 있게 하는 과정입니다
# '테스트 데이터'는 모델이 배운 내용을 얼마나 잘 적용하는지 시험하는 데 사용됩니다.
# 'set.seed(42)'는 데이터를 무작위로 섞는데, 숫자 42는 같은 결과를 얻기 위해 매번 똑같은 순서로 섞으라
# 'initial_split()'은 데이터를 80%는 훈련용, 20%는 테스트용으로 나누어줍니다.
# 'training()'은 훈련 데이터를 가져오는 명령어입니다.
# 'testing()'은 테스트 데이터를 가져오는 명령어입니다.

set.seed(42) # 데이터를 무작위로 섞되, 매번 같은 결과가 나오게 합니다.
data_split <- initial_split(data, prop = 0.8) # 데이터를 80% 훈련 데이터, 20% 테스트 데이터로 나눕니다.
train_data <- training(data_split) # 훈련 데이터를 가져옵니다.
test_data <- testing(data_split) # 테스트 데이터를 가져옵니다.
```

```
set.seed(42)
train_index <- sample(1:nrow(data), 0.8 * nrow(data))
train_data <- data[train_index, ]
test_data <- data[-train_index, ]
```

• 데이터를 나누는 이유: 우리가 데이터를 사용해 모델(기계가 배우는 방법)을 만들려면 훈련 데이터와 테스트 데이터가 필요합니다. 훈련 데이터는 모델이 배우는 데 사용되고, 테스트 데이터는 모델이 얼마나 잘 배웠는지 시는 데 사용됩니다.

1. `set.seed(42)`: 이 명령어는 데이터를 무작위로 섞을 때 매번 똑같은 결과를 얻기 위해 사용됩니다. 숫자 42는 아무 숫자나 넣을 수 있습니다.
2. `train_index <- sample(1:nrow(data), 0.8 * nrow(data))`:
  - `nrow(data)`: 데이터의 총 행(row) 개수를 셉니다. 예를 들어, 데이터에 100개의 집이 있다면 `nrow(data)`는 100입니다.
  - `sample(1:nrow(data), 0.8 * nrow(data))`: 이 부분은 데이터의 80%를 무작위로 뽑아냅니다. `0.8 * nrow(data)`는 데이터의 80%를 계산하는 것입니다.
  - `train_index`: 훈련 데이터로 사용할 행 번호들이 여기에 저장됩니다.
3. `train_data <- data[train_index, ]`: 이 명령어는 데이터를 `train_index`에서 선택된 행(즉, 80%)으로 구성된 훈련 데이터를 만듭니다.
4. `test_data <- data[-train_index, ]`: 이 부분은 훈련 데이터에 선택되지 않은 나머지 20%를 테스트 데이터로 만듭니다. `-train_index`는 `train_index`에 해당하지 않는 행을 의미합니다.

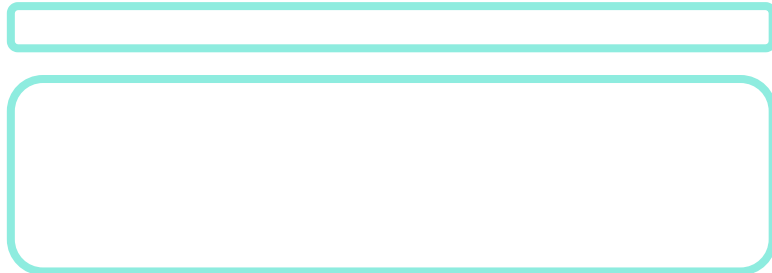


## ChatGPT로 분석하기- Tidymodels가 여전히 필요한 단 한가지 이유

### 소결론

사실 아래 네모 크기의 차이만큼이 바로 Tidymodels가 구현해낸 코드의 간결성이라는 사실... 여전히 초심자에게 Core R의 코드는 복잡하고 설명해야 할 부분이 많은 것이 사실.

Tidymodels는 초심자가 코드를 이해하기 위해서 읽어야하는 주석의 길이를 줄여준다. 즉, 이미 코드의 철학 자체에서 가지고 있는 human readability가 ChatGPT를 이용한 학습 과정에서도 불필요한 정보를 줄여주고 있다는 점.





**Thank you**

---