Data Science

Data Preprocessing

홍익대학교 소프트웨어융합학과

노기섭 교수

(kafa46@hongik.ac.kr)

Lecture Goals

- 결측치 처리 (Missing Value Handling)
- 이상치 탐지 (Outlier Detection)
- 데이터 정규화 (Data Normalization/Scaling)
- 범주형 데이터 처리 (Categorical Data Handling)

데이터 전처리(Data Preprocessing)

- **분석 성패를 좌우하는 핵심 단계**
- **원시(raw) 데이터를 분석·학습에 적합하게 변환하는 과정**
- **필요성**
  - 입력 데이터가 불완전·왜곡 → 모델 성능 저하
  - 꼼꼼한 전처리 → 분석 품질 보장
- **주요 요소**
  - 결측치 처리: 누락 데이터 보완
  - 이상치 탐지: 분포에서 벗어난 값 식별·처리
  - 정규화/표준화: 변수 간 크기 차이 조정
  - 범주형 데이터 처리: 문자열 → 수치형 변환
- **“데이터 청소”를 넘어 신뢰성과 해석 가능성 확보분석가가 가장 많은 시간 투자하는 영역**

결측치 처리

결측치(Missing Value)란?

- 특정 변수 값이 비어 있거나 기록되지 않은 상태
- 단순한 빈칸이 아니라 분석 전반에 큰 영향
- 불완전한 데이터 사용 시 → 잘못된 분석 결과
- 결측치 처리는 데이터 전처리의 핵심 과제

데이터 분석에서 결측치를 무시하면 어떤 문제가 생기는가?

통계적 왜곡

- 평균, 분산, 상관계수 등 기초 통계량 왜곡
- 예: 고소득층 응답 누락 → 평균 소득이 실제보다 낮게 계산
- 표본이 모집단을 제대로 대표하지 못하는 문제 발생

모델 정확도 저하

- 머신러닝 알고리즘 대부분 결측치 직접 처리 불가
- 결측치 포함 데이터 학습 시 예측력 크게 저하
- 특정 집단에 집중 발생 시 → 편향된 학습 결과

학습 불안정성

- 무작위 제거 → 데이터 크기 축소, 검정력 약화
- 잘못된 값 대체 → 실제 패턴 반영 실패, 불안정한 결과 초래

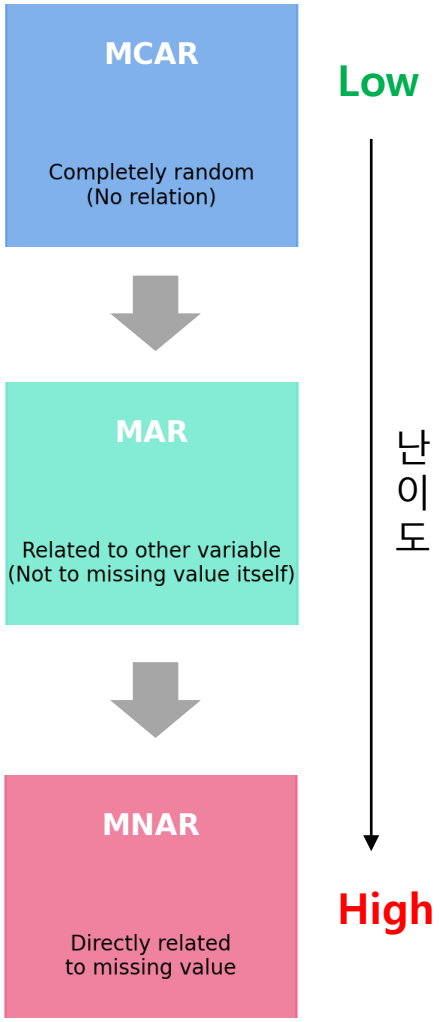
결측치 발생 원인

- **데이터 수집 오류**
  - 입력 실수, 네트워크 지연, 저장 매체 손상 등
  - 예: 나이 입력 누락, 잘못된 형식
- **기록설문조사 응답 거부**
  - 민감한 문항(소득, 재산, 건강 등) 응답 누락
  - 단순 누락 처리 시 → 데이터 편향 초래 가능
- **센서/시스템 고장**
  - IoT 센서 전원 꺼짐, 네트워크 끊김, 장비 오작동
  - 특정 시간대 데이터 전체 누락 발생
- **데이터 통합 불일치**
  - DB/파일 병합 시 키 값 불일치
  - 예: 한 DB에 이메일 있음, 다른 DB에 없음 → 통합 후 결측치 발생

결측치 유형

- **MCAR (Missing Completely at Random) – 완전히 무작위 누락**
  - 결측 발생이 다른 변수/해당 값과 무관
    - 예: 설문지 인쇄 손상으로 응답 항목 유실
  - 분석에 체계적 편향 없음, 단순 삭제도 큰 문제 X
  - 실제 데이터에서는 드문 경우
- **MAR (Missing at Random) – 특정 변수와 관련된 누락**
  - 결측 발생이 다른 변수와 관련 있음
    - 예: 고소득자일수록 자신의 소득수준 조사에 답변 X

      하지만, 같은 소득 수준 내에서는 결측이 무작위적
  - "소득수준" 변수 데이터의 단순 삭제보다 다른 변수 활용한 추정/보완 필요
- **MNAR (Missing Not at Random) – 값 자체와 관련된 누락**
  - 결측 발생이 해당 값과 직접적으로 연결
    - 예: 비만자가 체중을 의도적으로 체중 기입 질문에 답변 X
  - 결측이 "체중"이라는 변수 자체에 밀접하게 연관, 평균 대체 시 심각한 왜곡 발생
  - 가장 다루기 어려움 → 도메인 지식·고급 통계 기법 필요



결측치 처리 방법

- 제거법 (Deletion)
- 대체법 (Imputation)
- 도메인 지식 활용

제거법 (Deletion)

개념:

- 결측치가 포함된 데이터를 제외하고, 완전한 데이터만 활용

방법

- 행 제거 (Listwise deletion)
  - 결측치 포함된 전체 행 삭제
  - 장점: 구현 단순, 남은 데이터 완전
  - 단점: 데이터 급격히 줄어듦, 특정 집단 편향 가능
- 열 제거 (Variable deletion)
  - 결측치가 많은 변수 자체 삭제
  - 장점: 단순, 불필요 변수 제거 효과
  - 단점: 중요한 변수 손실 시 설명력 저하

장점

구현 간단, 추가 보완 불필요

결측치가 적으면 큰 영향 없음

단점

데이터 손실 및 표본 수 부족

특정 집단 결측 시 분석 왜곡 위험

중요한 변수 제거 시 성능 · 해석력 저하

제거법 (Deletion) 실습

```python
import pandas as pd
from typing import Tuple, Optional

def make_sample_dataframe() -> pd.DataFrame:
    """실습용 샘플 데이터프레임 생성"""
    data = {
        "id":      [1, 2, 3, 4, 5, 6, 7],
        "age":     [23, None, 31, 29, None, 41, 36],
        "income":  [52000, 61000, None, 58000, 60000, None, 72000],
        "city":    ["Seoul", "Busan", None, "Daejeon", "Seoul", "Seoul", None],
        "hobby":   [None, None, "Run", None, "Music", None, None],
    }
    return pd.DataFrame(data)

def missing_report(df: pd.DataFrame) -> pd.DataFrame:
    """컬럼별 결측 개수와 결측률 리포트"""
    n = len(df)
    report = pd.DataFrame({
        "missing_count": df.isna().sum(),
        "missing_rate":  (df.isna().mean() * 100).round(2)
    }).sort_values("missing_rate", ascending=False)
    return report

def listwise_delete(
    df: pd.DataFrame,
    subset: Optional[list[str]] = None,
    keep_threshold: Optional[int] = None
) -> pd.DataFrame:
    """
    행 제거(listwise deletion)
    - subset: 지정된 컬럼들 중 하나라도 NaN이면 해당 행 삭제
    - keep_threshold(thresh): NaN이 아닌 값의 최소 개수 미만인 행을 삭제
      (예: keep_threshold=3 -> 최소 3개 이상의 유효 값이 있는 행만 유지)
    """
    if keep_threshold is not None:
        # thresh는 NaN이 아닌 값의 최소 개수 기준
        return df.dropna(thresh=keep_threshold)
    if subset is not None:
        return df.dropna(subset=subset)
    # 기본: 하나라도 NaN이 있으면 행 삭제 (모든 컬럼 기준)
    return df.dropna()

def variable_delete_by_missing_ratio(
    df: pd.DataFrame,
    threshold: float = 0.4
) -> Tuple[pd.DataFrame, pd.Series]:
    """
    열 제거(variable deletion)
    - threshold: 결측률이 threshold 이상인 컬럼을 제거 (0~1 사이)
      예: 0.4 -> 결측률 40% 이상 컬럼 삭제
    반환: (제거 후 DF, 제거된 컬럼 목록 시리즈)
    """
    miss_rate = df.isna().mean()
    drop_cols = miss_rate[miss_rate >= threshold].index
    return df.drop(columns=drop_cols), miss_rate[drop_cols]
```

```python
def main():
    # 0) 샘플 데이터 준비
    df = make_sample_dataframe()
    print("원본 데이터\n", df, "\n")
    print("결측 리포트(원본)\n", missing_report(df), "\n")

    # 1) 행 제거: 모든 컬럼 기준으로 결측 포함 행 제거 (가장 보수적)
    df_listwise_all = listwise_delete(df)
    print(f"[행 제거 - 전체 컬럼 기준] shape: {df.shape} -> {df_listwise_all.shape}")
    print(df_listwise_all, "\n")

    # 2) 행 제거: subset 기준 (예: age, income 중 하나라도 NaN이면 삭제)
    df_listwise_subset = listwise_delete(df, subset=["age", "income"])
    print(f"[행 제거 - subset=['age','income']] shape: {df.shape} -> {df_listwise_subset.shape}")
    print(df_listwise_subset, "\n")

    # 3) 행 제거: keep_threshold 사용 (유효값이 4개 미만이면 삭제)
    df_listwise_thresh = listwise_delete(df, keep_threshold=4)
    print(f"[행 제거 - keep_threshold=4] shape: {df.shape} -> {df_listwise_thresh.shape}")
    print(df_listwise_thresh, "\n")

    # 4) 열 제거: 결측률 40% 이상인 컬럼 제거
    df_col_drop_40, dropped_40 = variable_delete_by_missing_ratio(df, threshold=0.40)
    print("[열 제거 - 결측률 40% 이상 삭제] 제거된 컬럼:")
    print(dropped_40.apply(lambda r: f"{round(r*100,2)}%"), "\n")
    print(f"shape: {df.shape} -> {df_col_drop_40.shape}")
    print(df_col_drop_40, "\n")
    print("결측 리포트(열 제거 후)\n", missing_report(df_col_drop_40), "\n")

    # 5) 열 제거: 결측률 60% 이상인 컬럼만 더 강하게 제거
    df_col_drop_60, dropped_60 = variable_delete_by_missing_ratio(df, threshold=0.60)
    print("[열 제거 - 결측률 60% 이상 삭제] 제거된 컬럼:")
    print(dropped_60.apply(lambda r: f"{round(r*100,2)}%"), "\n")
    print(f"shape: {df.shape} -> {df_col_drop_60.shape}")
    print(df_col_drop_60, "\n")

    # 6) 파이프라인 예시: 먼저 결측률 높은 열 제거 -> 그 다음 subset 기준 행 제거
    df_pipe, _ = variable_delete_by_missing_ratio(df, threshold=0.5)
    df_pipe = listwise_delete(df_pipe, subset=["age", "income"])
    print("[파이프라인] (열 제거 50%) -> (subset 행 제거)")
    print(f"최종 shape: {df.shape} -> {df_pipe.shape}")
    print(df_pipe, "\n")


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

제거법 (Deletion) 실행 결과

원본 데이터

	id	age	income	city	hobby
0	1	23.0	52000.0	Seoul	None
1	2	NaN	61000.0	Busan	None
2	3	31.0	NaN	None	Run
3	4	29.0	58000.0	Daejeon	None
4	5	NaN	60000.0	Seoul	Music
5	6	41.0	NaN	Seoul	None
6	7	36.0	72000.0	None	None

결측 리포트(원본)

	missing_count	missing_rate
hobby	5	71.43
income	2	28.57
age	2	28.57
city	2	28.57
id	0	0.00

[행 제거 - 전체 컬럼 기준] shape: (7, 5) -> (0, 5)

Empty DataFrame

Columns: [id, age, income, city, hobby]

Index: []

[행 제거 - subset=['age','income']] shape: (7, 5) -> (3, 5)

	id	age	income	city	hobby
0	1	23.0	52000.0	Seoul	None
3	4	29.0	58000.0	Daejeon	None
6	7	36.0	72000.0	None	None

[행 제거 - keep_threshold=4] shape: (7, 5) -> (3, 5)

	id	age	income	city	hobby
0	1	23.0	52000.0	Seoul	None
3	4	29.0	58000.0	Daejeon	None
4	5	NaN	60000.0	Seoul	Music

[열 제거 - 결측률 40% 이상 삭제] 제거된 컬럼:

hobby 71.43%

dtype: object

shape: (7, 5) -> (7, 4)

	id	age	income	city
0	1	23.0	52000.0	Seoul
1	2	NaN	61000.0	Busan
2	3	31.0	NaN	None
3	4	29.0	58000.0	Daejeon
4	5	NaN	60000.0	Seoul
5	6	41.0	NaN	Seoul
6	7	36.0	72000.0	None

결측 리포트(열 제거 후)

	missing_count	missing_rate
age	2	28.57
income	2	28.57
city	2	28.57
id	0	0.00

[열 제거 - 결측률 60% 이상 삭제] 제거된 컬럼:

hobby 71.43%

dtype: object

shape: (7, 5) -> (7, 4)

	id	age	income	city
0	1	23.0	52000.0	Seoul
1	2	NaN	61000.0	Busan
2	3	31.0	NaN	None
3	4	29.0	58000.0	Daejeon
4	5	NaN	60000.0	Seoul
5	6	41.0	NaN	Seoul
6	7	36.0	72000.0	None

[파이프라인] (열 제거 50%) -> (subset 행 제거)

최종 shape: (7, 5) -> (3, 4)

	id	age	income	city
0	1	23.0	52000.0	Seoul
3	4	29.0	58000.0	Daejeon
6	7	36.0	72000.0	None

- 단순한 전체 행 제거는 실무에서 적절하지 않음.
- 결측률, 분석 목적 등을 고려하여 행/열 제거 방식을 선택
- 데이터 품질을 유지하면서 분석 가능 데이터를 뽑아내는 것이 관건!

대체법 (Imputation)

개념

- 결측치를 특정 값이나 예측 값으로 채워 데이터 크기 유지
- 잘못 적용 시 새로운 왜곡 발생 가능

방법

구 분	기 법	설 명	장 점	단 점
단순 대체	평균/ 최빈값 대체	수치형은 평균(또는 중앙값), 범주형은 최빈값으로 결측치를 채움	구현이 매우 간단, 계산이 빠름	데이터 분산이 줄고, 분포 왜곡 발생 가능
통계적 대체	KNN 대체	결측치가 있는 샘플과 유사한 K개의 이웃 데이터를 찾아 평균/다수결로 대체	데이터 패턴을 잘 반영, 단순 대체보다 현실성 높음	계산 비용이 크고, K 값 선택에 따라 결과 달라짐
고급 대체	Iterative Imputer	각 변수를 다른 변수로 회귀 예측하여 반복적으로 대체	변수 간 상관관계 반영, 정교한 대체 가능, 불확실성 고려 가능	계산량 많음, 잘못 적용 시 비현실적 값 생성, 초기값·설정에 민감

회귀 기반 대체법

회귀 기반 대체법이란?

- 결측치(누락된 값)가 있는 변수를 다른 변수들의 관계를 활용해 예측하고 채워 넣는 방법
- 누락된 값을 "예측"해서 채우는 방식

예시 1: 학생 성적 데이터

- 학생들의 성적 데이터가 있다고 가정
  - 변수: 수학 점수, 영어 점수, 국어 점수
  - 문제: 일부 학생의 영어 점수가 결측치로 빠져 있음
- 이때 영어 점수를 그냥 평균으로 채우는 대신, 수학 점수와 국어 점수를 이용해 회귀모형을 만든 뒤 예측값으로 채움.

“수학과 국어 성적이 비슷하면 영어 점수도 비슷하겠지?”라는 가정을 활용

- 결과적으로 단순 평균보다 더 개별 학생의 특성을 반영한 대체가 가능

Iterative Imputer란 무엇인가?

개념

- 결측 변수를 다른 변수들의 함수로 가정

  → 반복 회귀모형으로 추정

장점

- 평균/최빈값 대체보다 변수 간 상관관계·

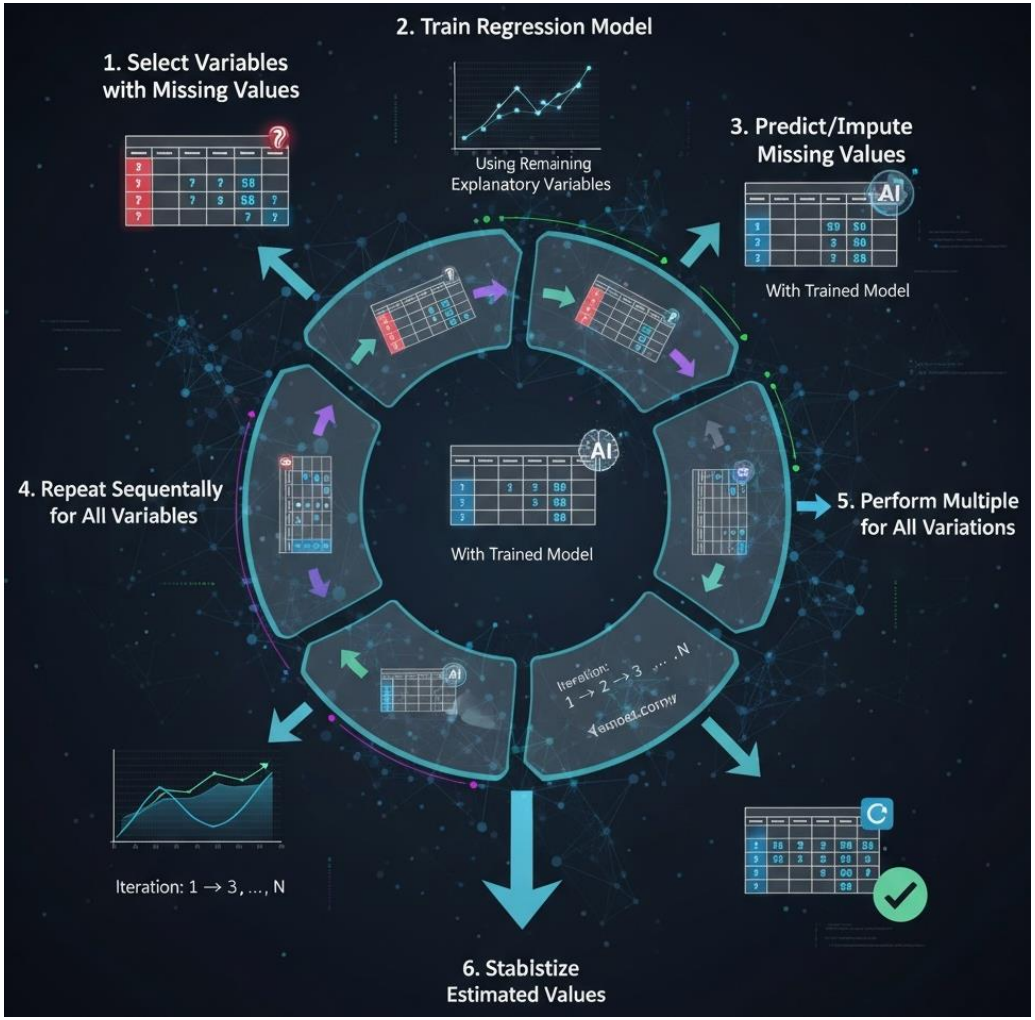
  구조적 특성 반영

- 더 현실성 있는 값 예측 가능

  (예: 소득 ~ 나이·학력·지역 기반 추정)

한계

- 데이터 적거나 상관성 약하면 부정확한 값 생성
- 반복 횟수·초기값에 따라 결과 달라짐
- 예측 값이 실제 범위 벗어날 위험 있음



대체법 (Imputation) 실습

```python
import numpy as np
import pandas as pd
from sklearn.impute import SimpleImputer, KNNImputer
from sklearn.experimental import enable_iterative_imputer  # IterativeImputer 사용 가능하게 함
from sklearn.impute import IterativeImputer
from sklearn.linear_model import BayesianRidge


# ========================
# 데이터 준비
# ========================
def make_sample_dataframe() -> pd.DataFrame:
    """실습용 샘플 데이터프레임 생성"""
    return pd.DataFrame({
        "id": [1, 2, 3, 4, 5, 6, 7],
        "age": [23, None, 31, 29, None, 41, 36],
        "income": [52000, 61000, None, 58000, 60000, None, 72000],
        "city": ["Seoul", "Busan", None, "Daejeon", "Seoul", "Seoul", None],
        "hobby": [None, None, "Run", None, "Music", None, None],
    })

# ========================
# 유틸리티 함수
# ========================
def normalize_missing(df: pd.DataFrame) -> pd.DataFrame:
    """None → np.nan 변환"""
    return df.replace({None: np.nan})


def detect_dtypes(df: pd.DataFrame, exclude: list[str] | None = None) -> tuple[list[str], list[str]]:
    """수치형/범주형 컬럼 구분"""
    exclude = exclude or []
    num_cols = [c for c in df.select_dtypes(include=[np.number]).columns if c not in exclude]
    cat_cols = [c for c in df.select_dtypes(exclude=[np.number]).columns if c not in exclude]
    return num_cols, cat_cols



def missing_report(df: pd.DataFrame) -> pd.DataFrame:
    """결측 개수와 결측률 리포트"""
    return pd.DataFrame({
        "missing_count": df.isna().sum(),
        "missing_rate": (df.isna().mean() * 100).round(2)
    }).sort_values("missing_rate", ascending=False)

# ========================
# 결측치 대체 방법
# ========================
def simple_impute_mean_mode(df: pd.DataFrame) -> pd.DataFrame:
    """단순 대체: 수치형=평균, 범주형=최빈값"""
    df = normalize_missing(df.copy())
    num_cols, cat_cols = detect_dtypes(df, exclude=["id"])
    if num_cols:
        df[num_cols] = SimpleImputer(strategy="mean").fit_transform(df[num_cols])
    if cat_cols:
        df[cat_cols] = SimpleImputer(strategy="most_frequent").fit_transform(df[cat_cols])
    return df
```

```python
def knn_impute(df: pd.DataFrame, n_neighbors: int = 3) -> pd.DataFrame:
    """KNN 기반 대체"""
    df = normalize_missing(df.copy())
    num_cols, cat_cols = detect_dtypes(df, exclude=["id"])
    out = df.copy()
    if num_cols:
        out[num_cols] = KNNImputer(n_neighbors=n_neighbors).fit_transform(df[num_cols])
    if cat_cols:
        out[cat_cols] = SimpleImputer(strategy="most_frequent").fit_transform(df[cat_cols])
    return out



def iterative_impute(df: pd.DataFrame) -> pd.DataFrame:
    """회귀 기반 대체 (Iterative Imputer)"""
    df = normalize_missing(df.copy())
    num_cols, cat_cols = detect_dtypes(df, exclude=["id"])
    out = df.copy()
    if num_cols:
        it = IterativeImputer(
            estimator=BayesianRidge(),
            max_iter=10,
            random_state=42,
            sample_posterior=True
        )
        out[num_cols] = it.fit_transform(df[num_cols])
    if cat_cols:
        out[cat_cols] = SimpleImputer(strategy="most_frequent").fit_transform(out[cat_cols])
    return out



# ========================
# 실행 예제
# ========================
def main() -> None:
    df = make_sample_dataframe()


    print("========== 원본 데이터 ==========\n")
    print(f"{df}\n")
    print("결측 리포트(원본)\n")
    print(f"{missing_report(df)}\n")


    print("---------- 단순 대체 (평균/최빈값) ---------\n")
    df1 = simple_impute_mean_mode(df)
    print(f"{df1}\n")
    print("결측 리포트(단순 대체)\n")
    print(f"{missing_report(df1)}\n")


    print("---------- KNN 기반 대체 ---------\n")
    df2 = knn_impute(df, n_neighbors=3)
    print(f"{df2}\n")
    print("결측 리포트(KNN)\n")
    print(f"{missing_report(df2)}\n")


    print("---------- 회귀 기반 대체 (Iterative) ---------\n")
    df3 = iterative_impute(df)
    print(f"{df3.round(2)}\n")  # 보기 좋게 소수점 반올림
    print("결측 리포트(Iterative)\n")
    print(f"{missing_report(df3)}\n")



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

대체법 (Imputation) 실행 결과

========== 원본 데이터 ==========

	id	age	income	city	hobby
0	1	23.0	52000.0	Seoul	None
1	2	NaN	61000.0	Busan	None
2	3	31.0	NaN	None	Run
3	4	29.0	58000.0	Daejeon	None
4	5	NaN	60000.0	Seoul	Music
5	6	41.0	NaN	Seoul	None
6	7	36.0	72000.0	None	None

결측 리포트(원본)

	missing_count	missing_rate
hobby	5	71.43
income	2	28.57
age	2	28.57
city	2	28.57
id	0	0.00

---------- 단순 대체 (평균/최빈값) ---------

	id	age	income	city	hobby
0	1	23.0	52000.0	Seoul	Music
1	2	32.0	61000.0	Busan	Music
2	3	31.0	60600.0	Seoul	Run
3	4	29.0	58000.0	Daejeon	Music
4	5	32.0	60000.0	Seoul	Music
5	6	41.0	60600.0	Seoul	Music
6	7	36.0	72000.0	Seoul	Music

결측 리포트(단순 대체)

	missing_count	missing_rate
id	0	0.0
age	0	0.0
income	0	0.0
city	0	0.0
hobby	0	0.0

---------- KNN 기반 대체 ---------

	id	age	income	city	hobby
0	1	23.000000	52000.000000	Seoul	Music
1	2	29.333333	61000.000000	Busan	Music
2	3	31.000000	60666.666667	Seoul	Run
3	4	29.000000	58000.000000	Daejeon	Music
4	5	29.333333	60000.000000	Seoul	Music
5	6	41.000000	60666.666667	Seoul	Music
6	7	36.000000	72000.000000	Seoul	Music

결측 리포트(KNN)

	missing_count	missing_rate
id	0	0.0
age	0	0.0
income	0	0.0
city	0	0.0
hobby	0	0.0

---------- 회귀 기반 대체 (Iterative) ---------

	id	age	income	city	hobby
0	1	23.00	52000.00	Seoul	Music
1	2	21.57	61000.00	Busan	Music
2	3	31.00	63716.89	Seoul	Run
3	4	29.00	58000.00	Daejeon	Music
4	5	7.87	60000.00	Seoul	Music
5	6	41.00	59621.37	Seoul	Music
6	7	36.00	72000.00	Seoul	Music

결측 리포트(Iterative)

	missing_count	missing_rate
id	0	0.0
age	0	0.0
income	0	0.0
city	0	0.0
hobby	0	0.0

도메인 지식 활용

- **도메인 지식: 해당 분야의 전문가적 배경지식 → 단순 통계 처리 한계를 보완**
- **의미 구분 필요**
  - 환자 데이터
    - 검사 결과 0 → 실제 증상 없음
    - 값 없음 (미 기록) → 검사 누락/기록 오류
  - 금융 데이터
    - 거래 금액 0원 → 실제 거래 없음
    - 값 없음 → 입력 오류/시스템 누락 가능
- **핵심**
  - "없음" vs "미기록"을 명확히 구분해야 정확한 해석 가능
  - 데이터 맥락·수집 배경 고려 → 분석 신뢰도와 현장 적용성 강화
  - 필요 시 도메인 전문가와 협업 필수

도메인 지식 활용 실습

컬럼명	데이터 유형	의미	일반적 범위/값
sbp	연속형	수축기 혈압 (Systolic BP)	정상: 약 90 ₩~ 120 mmHg
dbp	연속형	이완기 혈압 (Diastolic BP)	정상: 약 60 ₩~ 80 mmHg
glucose	연속형	혈당 수치 (Blood Glucose)	정상 공복 혈당: 약 70 ₩~ 100 mg/dL
symptom_present	이진	증상 존재 여부	0 = 증상 없음, 1 = 증상 있음
diagnosis	범주형	진단 결과	Hypertension, Diabetes, None 등

도메인 지식 실습 코드

- **실습 코드는 웹 페이지 참조**
  - https://www.deepshark.org/courses/data_science/w/04_data_preprocessing

이상치 탐지

이상치란 무엇인가?

- **정의**
  - 데이터의 일반적 분포·패턴에서 크게 벗어난 값
  - 예: 키 조사에서 300cm, 소비 금액이 갑자기 1억 원
- **발생 원인**
  - 데이터 입력·수집 오류 (센서 오작동, 입력 실수)
  - 시스템 통합 시 단위 변환 오류
  - 실제 극단적 사건 발생 (사기 거래, 특이 이벤트 등)
- **의미와 해석**
  - 단순 오류 → 제거 필요 (예: 신체 데이터 측정 오류)
  - 중요한 신호 → 분석 대상 (예: 금융 사기, 기계 고장 조짐)
  - 잡음인지 핵심 정보인지 맥락에 따라 판단해야 함
- **영향**
  - 통계적 왜곡: 평균, 표준편차 등 요약값 크게 흔들림
  - 모델 성능 저하: 회귀선 왜곡, 결정 경계 학습 오류, 예측력 저하
- **중요성: 이상치 탐지·해석·처리 여부가 분석 신뢰도를 좌우데이터 과학 필수 단계**

이상치를 무시했을 때 발생하는 문제점

■ 통계적 왜곡

- 평균, 표준편차, 분산 등 기본 통계량이 왜곡됨
- 예: 시험 점수 대부분 70~90점인데 0점 하나 → 평균 급격히 낮아짐, 표준편차 과대 추정
- 기본 통계량부터 잘못 → 이후 분석 전반에 부정적 영향

■ 모델 성능 저하

- 잘못된 패턴 학습, 극단 값에 끌려감
- 회귀: 단일 이상치가 전체 회귀선 왜곡
- 분류: 이상치가 결정 경계 구부려 과적합 초래
- 정확도와 일반화 성능 모두 저하

■ 학습 불안정성

- 특히 딥러닝 모델에서 심각이상치 → 손실 함수 과도하게 커짐 → 기울기 폭발(gradient explosion) 발생
- 학습이 발산하거나 수렴 불안정 → 훈련 시간 증가, 신뢰도 저하

이상치 탐지 방법

구분	주요 기법	설명	장점	단점
통계적 방법	Z-Score, IQR(사분위 범위)	데이터 분포(평균, 표준편차, 사분위수)를 기준으로 임계값 이상 벗어난 관측치를 이상치로 간주	계산이 간단하고 직관적임	분포가 정규분포가 아닐 경우 정확도 저하
거리 기반 방법	k-NN Distance, DBSCAN	데이터 간의 거리(distance)를 계산해 밀도가 낮거나 주변과 동떨어진 점을 이상치로 판단	다양한 데이터 분포에 적용 가능	고차원 데이터에서는 거리 계산이 비효율적
머신러닝 기반 방법	Isolation Forest, One-Class SVM	학습 알고리즘을 이용해 정상 데이터 패턴을 학습하고, 그와 크게 다른 점을 이상치로 탐지	복잡한 패턴의 이상치 탐지가 가능	계산 비용이 크고 모델 선택·튜닝이 필요함

이상치 처리 방법

제거 (Deletion)

- 입력 오류, 센서 고장 → 삭제
- 장점: 간단, 분포 정리 용이
- 단점: 데이터 손실 ↑, 중요한 신호까지 제거 위험

대체 (Imputation / Replacement)

- 평균·중앙값·최빈값 또는 회귀·KNN 등으로 교체
- 장점: 데이터 손실 최소화
- 단점: 변동성 왜곡, 인위적 값 생성

변환 (Transformation)

- 로그 변환, 제곱근 변환, 표준화 등
- 장점: 구조 유지하며 이상치 영향 완화
- 단점: 변수 의미 변형 가능, 모든 상황에 효과적이지 않음

핵심:
이상치 처리법(제거·대체·변환)은 **데이터 특성과 분석 목적**에 따라 신중히 선택해야 함.

이상치 변환 실습

- **실습 코드는 웹 페이지 참조**
  - https://www.deepshark.org/courses/data_science/w/04_data_preprocessing

데이터 정규화

정규화란 무엇인가?

정의

- 데이터의 크기·범위를 일정한 기준으로 조정하는 과정
- 서로 다른 단위·범위를 가진 변수를 공정하게 비교 가능

필요성

- 변수 단위 차이 → 모델 학습 시 불균형 발생
- 경사 하강법 기반 모델: 스케일 차이로 학습 속도·안정성 저하
- 시각화 시 범위 큰 변수가 지배 → 패턴 왜곡

주요 방법

- Min-Max Scaling: 0~1 범위로 변환
- Z-score Standardization: 평균 0, 표준편차 1로 변환
- Robust Scaling: 중앙값·사분위수 활용, 이상치 영향 감소

효과

- 변수 간 공정한 비교, 모델 학습 효율성·안정성 향상, 해석·시각화 용이성 증대

정규화가 필요한 이유

■ 변수 단위 차이 조정

- 키(cm), 몸무게(kg), 소득(원) 등 단위·범위 다름
- 큰 값 가진 변수가 모델에 과도한 영향 → 예측 정확도 저하
- 정규화 → 변수 간 동등한 조건에서 학습 기여

■ 학습 속도와 안정성 확보

- 경사 하강법(Gradient Descent) 최적화 시 변수 크기 차이 → 기울기 불균형
- 수렴 경로가 지그재그로 비효율적, 속도 저하·수렴 실패 위험
- 정규화 → 균형 잡힌 스케일 → 빠르고 안정적인 학습

■ 이상치와 분포 왜곡 완화

- 극단적으로 큰 값 → 전체 스케일 확대, 다른 데이터 중요성 감소
- 정규화 → 이상치 영향 완화, 전체 분포 반영 강화

주요 정규화 기법

구분	기법	설명	장점	단점
최소-최대 정규화	Min-Max Scaling	데이터 값을 0~1 범위로 선형 변환	직관적이고 계산이 간단 값의 상대적 비율 유지	이상치에 민감하여 범위가 왜곡될 수 있음
Z-점수 표준화	Z-score Standardization	평균을 0, 표준편차를 1로 변환하여 표준 정규분포 기반으로 스케일 조정	분포가 다른 변수들을 비교 가능 경사 하강법 기반 모델에 적합	이상치에 민감 정규분포 가정이 잘 맞지 않으면 효과 제한
로버스트 스케일링	Robust Scaling	중앙값을 0, IQR(사분위 범위)을 1로 변환	이상치에 강건함 비대칭적 분포에도 안정적	데이터의 세밀한 변화를 과도하게 단순화할 수 있음
로그 변환	Log Transformation	로그 함수를 적용하여 큰 값을 압축	긴 꼬리 분포 완화 이상치 영향 완화	0 이하 값에는 적용 불가 해석이 직관적이지 않을 수 있음
제곱근 변환	Square Root Transformation	제곱근 함수를 적용하여 분산을 줄임	분산이 큰 데이터 완화 로그 변환보다 완만한 압축 효과	음수 데이터에는 적용 불가 효과가 제한적일 수 있음

최소-최대 정규화 (Min-Max Normalization)

정의

- 데이터 최소값 → 0, 최대값 → 1로 변환모든 값을 0~1 범위로 선형 변환

공식

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

장점

- 결과 범위가 항상 0~1
- 서로 다른 단위·범위의 변수 비교 가능
- 거리 기반 알고리즘(KNN, K-means 등)에 효과적

단점

- 이상치에 민감 → 극단값이 최소·최대로 잡히면 나머지 데이터가 0 근처에 몰려 변별력 저하

Z-점수 표준화 (Z-score Standardization)

정의

- 데이터 → 평균 0, 표준편차 1 분포로 변환
- Z-점수: 평균으로부터 몇 개의 표준편차 떨어져 있는지 의미

공식

$$x' = \frac{x - \mu}{\sigma}$$

장점

- 데이터의 상대적 위치 해석 용이, 서로 다른 시험/변수 비교 가능
- 경사 하강법 기반 모델 학습 안정성 ↑, 변수 간 공정한 반영 가능

단점

- 이상치에 민감
- 극단값이 평균·표준편차 왜곡 → 변환 결과 왜곡 발생
- 이상치 많을 경우 → 로버스트 스케일링 권장

로버스트 스케일링 (Robust Scaling)

- **정의: 중앙값(Median)과 사분위 범위(IQR, Inter-Quantile Range = Q3 - Q1) 기준으로 변환**
- **공식**

$$x' = \frac{x - Median}{IQR}$$

- **특징**
  - 평균·표준편차 대신 중앙값·사분위수 활용, 극단값 영향 최소화 → 이상치에 강건(Robust)
- **장점**
  - 이상치가 많은 데이터에서도 안정적, 일반적인 데이터 패턴을 잘 반영
- **단점**
  - 정규분포·대칭적 데이터에는 효율 ↓, 범위를 0~1로 제한하지 않아 비교 직관성 ↓
- **활용**
  - 극단값 많은 실제 데이터셋에서 유용
  - Min-Max, Z-score보다 이상치에 유연한 대안

로그/제곱근 변환 (Log/Square Root Transformation)

공통 특징

- 데이터 크기를 압축(Compression)
- 분포 안정화(Stabilization)
- 이상치(극단값) 영향 완화

로그 변환 (Log Transformation)

- 큰 값을 더 강하게 압축 $x' = \log(x + c)$
- 소득·거래액처럼 양수의 비대칭 분포에 적합

제곱근 변환 (Square Root Transformation)

$$x' = \sqrt{x + c}$$

- 비교적 완만하게 압축
- 카운트 데이터(예: 방문자 수, 사건 발생 횟수)에 적합

정규화 실습

- **실습 코드는 웹 페이지 참조**
  - https://www.deepshark.org/courses/data_science/w/04_data_preprocessing

범주형 데이터 처리

범주형 데이터란 무엇인가?

- **정의**
  - 집단·범주 구분을 위한 데이터
  - 값은 문자열·라벨·기호 형태 → 크기 비교 불가
  - 예: 성별, 혈액형, 도시, 학년
- **종류**
  - 명목형 데이터 (Nominal)
    - 순서·크기 없음
    - 예: 성별(남/여), 혈액형(A/B/O/AB), 국적
  - 순서형 데이터 (Ordinal)
    - 순서 개념 있음 (정량적 차이는 없음)
    - 예: 학년(1학년 < 2학년 < 3학년), 고객 만족도
- **분석 활용**
  - 집단 비교 기준 (예: 도시별 매출)
  - 순서형 변수는 예측 모델에서 중요한 설명 변수
- **처리 필요성**
  - 대부분의 알고리즘은 숫자 데이터만 처리 가능하므로, 범주형 데이터 → 수치형으로 변환 필요 (Categorical Data Handling)

범주형 데이터 처리 방법

- **라벨 인코딩 (Label Encoding) ← 순서형 (ordinal) 데이터**
  - 범주 → 정수 변환 (예: 서울=0, 부산=1, 대전=2)
  - 장점: 구현 간단, 메모리 효율 ↑
  - 단점: 순서 없는 범주도 숫자 크기로 오해될 수 있음
- **원-핫 인코딩 (One-Hot Encoding) ← 명목형 (nominal) 데이터**
  - 각 범주 → 새로운 열 생성, 해당 값 1 / 나머지 0
  - 장점: 잘못된 순서 해석 문제 없음
  - 단점: 범주 수 많으면 차원 폭발(차원의 저주)
- **더미 변수 (Dummy Variables) ← 명목형 (nominal) 데이터**
  - 원-핫 인코딩에서 1개 범주 제외 → 중복 방지
  - 장점: 다중공선성 완화, 회귀모델 안정성 ↑
  - 단점: 기준 범주 선택에 따라 해석 달라짐
- **고급 기법**
  - 타깃 인코딩: 범주 → 평균 목표 값으로 치환 (데이터 누수 주의)
  - 임베딩(Embedding): 범주형 변수를 벡터로 매핑, 딥러닝 활용 (고차원 → 저 차원 변환, 의미 보존)

범주형 데이터 처리 실습

- **실습 코드는 웹 페이지 참조**
  - https://www.deepshark.org/courses/data_science/w/04_data_preprocessing

수고하셨습니다 ..^^..