파생 변수 생성 (Feature Engineering)

- 기존에 주어진 변수(feature)들을 조합하거나 가공하여 새로운 변수를 만드는 과정
- **피처 엔지니어링(Feature Engineering)**의 핵심적인 부분
- 파생 변수는 데이터에 대한 깊은 이해(domain knowledge)를 바탕으로 만들어지며, 모델이 데이터의 패턴을 더 쉽게 학습하도록 돕는 역할을 합니다.

주요 파생 변수 생성 방법

- **수치형 변수 간의 연산**: 두 변수를 더하거나, 빼거나, 곱하거나, 나누어 새로운 의미를 갖는 변수를 생성합니다. (e.g., '수입'과 '지출'로 '순수익'을 계산)
- **날짜/시간 변수 분해**: 날짜/시간 데이터에서 년, 월, 일, 요일, 시간, 분기 등 세부 정보를 추출하여 새로운 변수로 만듭니다. (e.g., '가입일'에서 '가입 요일'을 추출하여 요일별 패턴 분석)
- **범주형 변수 조합**: 두 개 이상의 범주형 변수를 조합하여 새로운 카테고리를 만듭니다. (e.g., '성별'과 '연령대'를 조합하여 '20대 남성', '30대 여성' 등의 그룹 생성)
- **데이터의 통계량 활용**: 특정 그룹(e.g., 사용자 ID) 내의 데이터에 대한 통계량(평균, 합계, 개수, 표준편차 등)을 계산하여 새로운 변수로 사용합니다. (e.g., 각 사용자의 '평균 구매 금액')

적용 가능한 상황

- **모델 성능 개선**: 기존 변수만으로는 모델이 데이터의 복잡한 관계를 충분히 학습하지 못할 때, 파생 변수를 통해 숨겨진 패턴을 명시적으로 드러내어 모델의 예측력을 높일 수 있습니다.
- **도메인 지식 활용**: 해당 비즈니스나 데이터에 대한 전문가의 지식을 모델에 반영하고 싶을 때. (e.g., '키'와 '몸무게'로 비만도를 나타내는 'BMI' 지수를 만드는 것)
- **데이터 정보 압축**: 여러 변수에 흩어져 있는 정보를 하나의 의미 있는 변수로 요약하여 표현하고 싶을 때.
- **추세 및 주기성 반영**: 시계열 데이터에서 시간의 흐름에 따른 변화(trend)나 주기적인 패턴(seasonality)을 변수로 만들어 모델이 시간적 특성을 학습하도록 할 때.

예제 데이터프레임 생성

```
import pandas as pd

data = {'order_id': [1, 2, 3, 4, 5],
        'user_id': [101, 102, 101, 103, 102],
        'order_date': ['2023-01-15 10:30', '2023-01-16 14:00', '2023-02-20 11:00',
'2023-03-10 20:45', '2023-03-12 13:20'],
        'price': [100, 250, 80, 120, 300],
        'quantity': [2, 1, 3, 5, 2]}
df = pd.DataFrame(data)
df['order_date'] = pd.to_datetime(df['order_date'])
```

1. 수치형 변수 간 연산

- **용도**: 기존 수치형 변수들을 사칙연산하여 새로운 변수를 만듭니다.

```
# '총 구매 금액' 파생 변수 생성
df['total_price'] = df['price'] * df['quantity']

print("--- Derived from numeric operations ---")
print(df[['price', 'quantity', 'total_price']])
'''
--- Derived from numeric operations ---
    price  quantity  total_price
0     100         2          200
1     250         1          250
2      80         3          240
3     120         5          600
4     300         2          600
'''
```

2. 날짜/시간 변수 분해

- **용도**: `datetime` 타입의 변수에서 년, 월, 요일 등 다양한 시간 관련 정보를 추출합니다.
- **주의사항**: `dt` 접근자(accessor)를 사용하기 위해서는 해당 열이 반드시 `datetime` 타입이어야 합니다.

```
# 날짜/시간 관련 파생 변수 생성
df['order_year'] = df['order_date'].dt.year
df['order_month'] = df['order_date'].dt.month
df['order_dayofweek'] = df['order_date'].dt.dayofweek # 월요일=0, 일요일=6
df['order_hour'] = df['order_date'].dt.hour

print("\n--- Derived from datetime ---")
print(df[['order_date', 'order_year', 'order_month', 'order_dayofweek',
'order_hour']])
```

- **결과 해석**
  - `order_date`에서 년, 월, 요일, 시간 정보가 각각 새로운 변수로 추출됨
  - 모델은 월별, 요일별, 시간대별 구매 패턴을 학습 가능

Series.dt 접근자 기본 메서드

메서드	내용
`df['date'].dt.date`	YYYY-MM-DD (문자)
`df['date'].dt.year`	연 (4자리 숫자)
`df['date'].dt.month`	월 (숫자)
`df['date'].dt.month_name()`	월 (문자)
`df['date'].dt.day`	일 (숫자)
`df['date'].dt.time`	HH:MM:SS (문자)

메서드	내용
`df['date'].dt.hour`	시 (숫자)
`df['date'].dt.minute`	분 (숫자)
`df['date'].dt.second`	초 (숫자)
`df['date'].dt.quarter`	분기 (숫자)
`df['date'].dt.day_name()`	요일 이름 (문자)
`df['date'].dt.weekday`	요일 숫자 (0=월, 6=일)
`df['date'].dt.dayofyear`	연 기준 몇 일째 (숫자)
`df['date'].dt.days_in_month`	월 일수 (=daysinmonth) (숫자)

Series.dt vs Series.dt.isocalendar()

구분	`air['Date'].dt`	`air['Date'].dt.isocalendar()`
일	`day`	X
월	`month`	X
연	`year`	`year`
주차	X	`week`
요일	`weekday` : 0~6 (월~일)	`day` : 1~7 (월~일)

3. 그룹별 통계량 활용

- **용도**: 특정 그룹(e.g., 사용자)을 기준으로 데이터를 집계하여 그룹의 특성을 나타내는 변수를 만듭니다.
- **주의사항**: `groupby()`와 `transform()`을 함께 사용하면, 집계된 값을 원래 데이터프레임의 각 행에 맞게 브로드캐스팅하여 쉽게 새로운 열로 추가할 수 있습니다.

```
# 사용자별 총 구매 금액 및 평균 구매 금액 파생 변수 생성
df['user_total_spent'] = df.groupby('user_id')['total_price'].transform('sum')
df['user_avg_spent'] = df.groupby('user_id')['total_price'].transform('mean')

# 사용자별 구매 횟수 파생 변수 생성
df['user_order_count'] = df.groupby('user_id')['order_id'].transform('count')

print("\n--- Derived from group statistics ---")
print(df[['user_id', 'total_price', 'user_total_spent', 'user_avg_spent',
'user_order_count']])
```

- **결과 해석**
  - 각 주문 행에 해당 주문을 한 사용자의 총 구매액, 평균 구매액, 총 주문 횟수 정보가 추가
    - `user_id`가 101인 모든 행의 `user_total_spent`가 440 (200+240)로 생성됨
    - 각 사용자의 충성도나 구매력을 나타내는 강력한 변수로 해석 가능

장단점 및 대안

장점	단점
모델 성능 향상: 데이터에 숨겨진 비선형적, 복합적 패턴을 모델이 쉽게 학습하도록 하여 예측 정확도를 크게 높일 수 있습니다.	**도메인 지식 의존성**: 효과적인 파생 변수를 만들기 위해서는 해당 데이터가 생성된 분야(도메인)에 대한 깊은 이해가 필요합니다.
모델 해석력 증대: 잘 만들어진 파생 변수는 그 자체로 의미를 가지므로(e.g., 'BMI', '재구매율'), 모델의 예측 결과를 더 쉽게 해석할 수 있게 도와줍니다.	**오버피팅 위험**: 너무 많은 파생 변수를 만들거나, 타겟 변수의 정보를 과도하게 사용하여 파생 변수를 만들 경우 모델이 학습 데이터에만 과적합(overfitting)될 수 있습니다.
알고리즘 한계 극복: 단순한 선형 모델도 좋은 파생 변수가 있다면 복잡한 모델 못지않은 성능을 낼 수 있습니다.	**시간과 노력**: 어떤 변수를 어떻게 조합할지 탐색하고 검증하는 데 많은 시간과 노력이 소요될 수 있습니다.

대안: 자동화된 피처 엔지니어링 (Automated Feature Engineering)

- 수동으로 파생 변수를 만드는 작업은 많은 노력을 요구하기 때문에, 이를 자동화하려는 시도도 있습니다.
- `featuretools` 와 같은 라이브러리는 "Deep Feature Synthesis"라는 알고리즘을 사용하여 데이터프레임 간의 관계를 기반으로 자동으로 수많은 파생 변수를 생성해줍니다.
- 이러한 도구는 탐색 단계에서 잠재적으로 유용한 변수 아이디어를 얻는 데 도움이 될 수 있지만, 생성된 변수들의 의미를 해석하고 그중에서 유용한 것을 선별하는 과정은 여전히 분석가의 몫입니다.