변수 변환

- **로그 변환 (Log Transformation)**
    - 데이터에 자연로그(ln)나 상용로그(log10)를 취하는 방법
    - 큰 값은 크게 줄이고, 작은 값은 작게 줄인다.
        - 오른쪽으로 길게 꼬리를 갖는(`right-skewed`) 분포(`Positive Skewed`)를 정규분포에 가깝게 만들어주는 효과
    - 데이터의 편차를 줄여주고, 곱셈 관계를 덧셈 관계로 변환하여 해석을 용이하게 함
- **제곱근 변환 (Square Root Transformation)**
    - 데이터에 제곱근을 취하는 방법
    - 로그 변환보다는 효과가 약하지만, 마찬가지로 `Positive Skewed`를 완화하는 데 사용
    - 특히, 데이터가 포아송 분포(카운트 데이터)를 따를 때 분산을 안정화시키는 효과
- **지수 변환 (Power Transformation)**
    - `Negative Skewed`/`left-skewed`를 완화시키는데 사용
    - 제곱이나 거듭제곱을 적용해서, 작은 값은 조금 키우고 큰 값은 많이 키운다.

적용 가능한 상황

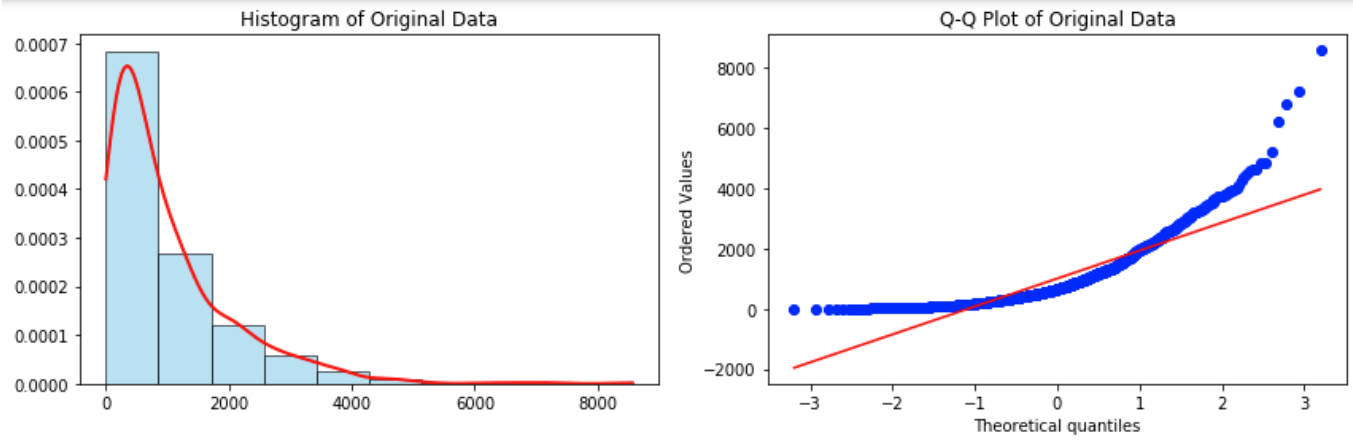
- **분포의 왜도(Skewness) 완화**: 히스토그램을 그렸을 때 데이터가 한쪽으로 심하게 쏠려 있는 경우, 로그 변환이나 제곱근 변환을 통해 분포를 대칭적으로 만들 수 있습니다.
- **선형 회귀 모델의 가정 충족**:
    - **선형성**: 독립변수와 종속변수의 관계가 비선형적일 때, 한쪽 또는 양쪽에 변환을 적용하여 관계를 선형적으로 만들 수 있습니다.
    - **등분산성**: 잔차의 분산이 일정하지 않을 때(이분산성), 종속변수에 변환을 적용하여 잔차의 분산을 안정시킬 수 있습니다.
    - **정규성**: 잔차가 정규분포를 따르지 않을 때, 종속변수 변환을 통해 잔차의 분포를 정규분포에 가깝게 만들 수 있습니다.
- **값의 범위(Scale) 축소**: 매우 큰 값을 갖는 데이터의 범위를 줄여 다른 변수와의 스케일 차이를 완화하고, 이상치의 영향을 줄일 수 있습니다. (지수 변환은 반대)

예제 데이터 생성

- `seaborn` 라이브러리가 `0.11.2` 이상이라면, `sns.histplot`에 `kde=True`만 적용해서 해결 가능
- 아래 코드의 시각화는 `seaborn` 라이브러리가 `0.9.0`일 때, `matplot`으로 `histogram` 그리는 상황

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from scipy import stats
from scipy.stats import gaussian_kde

# 지수분포를 이용하여 right-skewed 데이터 생성
np.random.seed(0)
```

```
data = np.random.exponential(scale=1000, size=1000)
df = pd.DataFrame(data, columns=['value'])

def plot_distribution(df, column, title):
    plt.figure(figsize=(12, 4))

    # (1) Histogram + KDE
    plt.subplot(1, 2, 1)
    data = df[column].dropna()

    # 히스토그램 (density=True → 확률밀도로 표시)
    plt.hist(data, density=True, alpha=0.6, color='skyblue', edgecolor='black')

    # KDE 추가
    kde = gaussian_kde(data)
    x = np.linspace(data.min(), data.max(), 200)
    plt.plot(x, kde(x), color='red', linewidth=2)

    plt.title(f'Histogram of {title}')

    # (2) Q-Q Plot
    plt.subplot(1, 2, 2)
    stats.probplot(data, dist="norm", plot=plt)
    plt.title(f'Q-Q Plot of {title}')

    plt.tight_layout()
    plt.show()

# 원본 데이터 분포 확인
plot_distribution(df, 'value', 'Original Data')
```



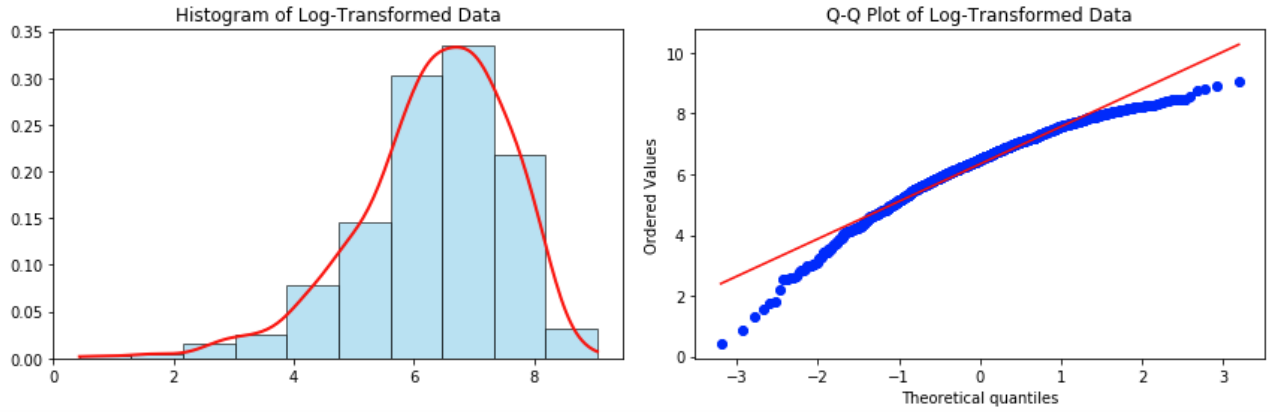
- **원본 데이터**: 히스토그램이 오른쪽으로 길게 꼬리를 가지며, Q-Q 플롯의 점들이 직선에서 크게 벗어나 있어 정규분포를 따르지 않음을 보여줍니다.

1. 로그 변환 (Log Transformation)

- **용도**: right-skewed 분포를 정규분포에 가깝게 변환
- **주의사항**
  - 로그 함수는 0이나 음수 값을 입력으로 받을 수 없음

    - 데이터에 0이 포함된 경우, 아주 작은 값(e.g., 1)을 더한 후 로그 변환 (`np.log1p` 또는 `np.log(x + 1)`).
- **코드 예시**

```
# 로그 변환 (log1p는 log(1+x)를 계산하여 0을 처리)
df['log_value'] = np.log1p(df['value'])

# 변환 후 데이터 분포 확인
plot_distribution(df, 'log_value', 'Log-Transformed Data')
```

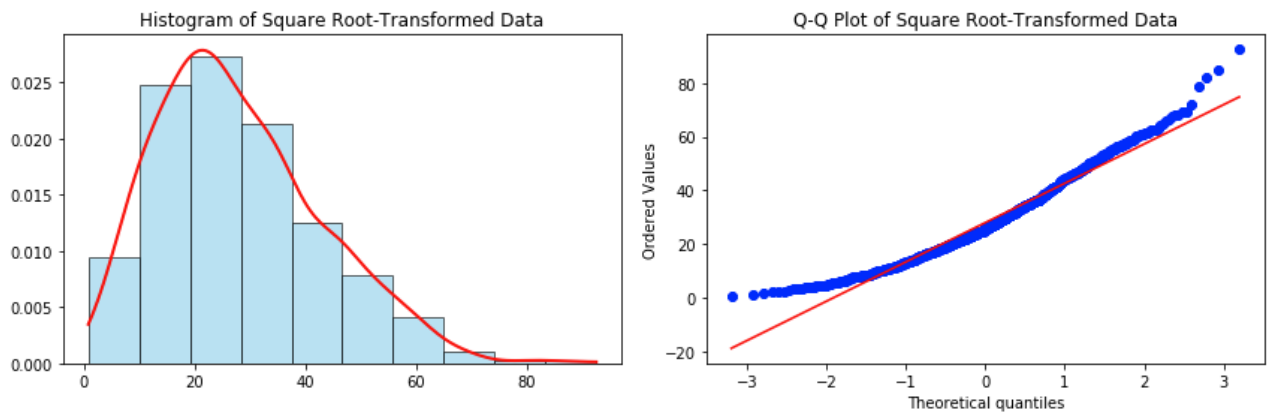


- **로그 변환**: 히스토그램이 원본보다 훨씬 대칭적인 종 모양에 가까워지고, Q-Q 플롯의 점들이 직선에 가깝게 정렬됩니다. 이는 데이터가 정규분포에 더 가까워졌음을 의미합니다.

2. 제곱근 변환 (Square Root Transformation)

- **용도**: right-skewed 분포를 완화 (로그 변환보다 효과는 약함)
- **주의사항**: 음수 값에는 적용 불가
- **코드 예시**

```
# 제곱근 변환
df['sqrt_value'] = np.sqrt(df['value'])

# 변환 후 데이터 분포 확인
plot_distribution(df, 'sqrt_value', 'Square Root-Transformed Data')
```

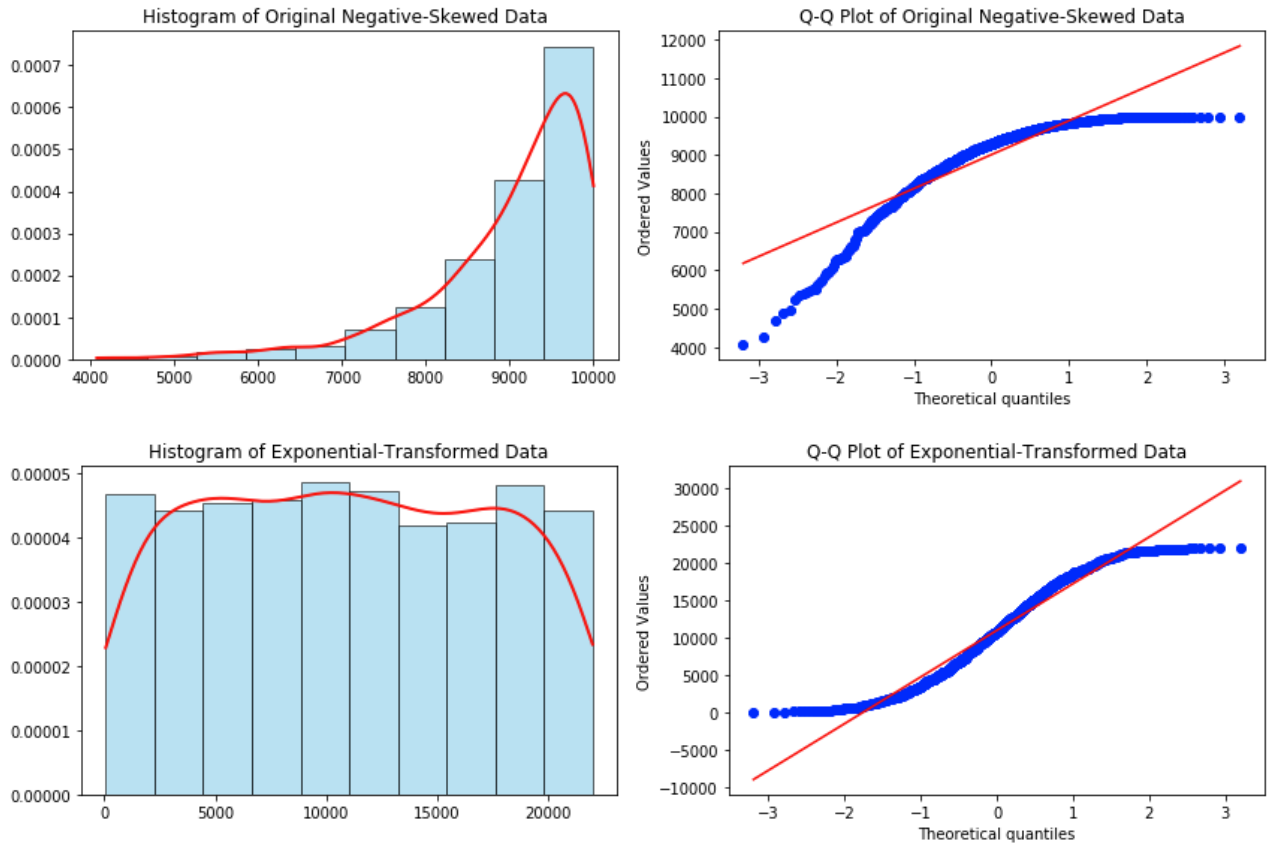


- **제곱근 변환**: 원래 데이터보다 왜도가 감소하고 분포가 더 대칭적으로 변한 것을 확인할 수 있습니다. 하지만 이 예제 데이터처럼 왜도가 매우 심한 경우에는 로그 변환만큼 효과적이지는 않을 수 있습니다.

3. 지수 변환 (Exponential Transformation)

- **용도**: left-skewed(좌측 치우친) 분포를 정규분포에 가깝게 변환
- **주의사항**
  - 지수 변환은 데이터의 큰 값을 상대적으로 더 늘려주기 때문에, **좌측 치우친 분포(꼬리가 왼쪽)** 에 적합
  - 값이 너무 큰 경우 overflow 발생 가능 → 필요하다면 스케일 조정 후 적용
- **코드 예시**

```
# 좌측으로 치우친 데이터 생성
np.random.seed(1)
neg_data = -np.random.exponential(scale=1000, size=1000)+10000
df_neg = pd.DataFrame(neg_data, columns=['value'])

# 원본 데이터 분포 확인
plot_distribution(df_neg, 'value', 'Original Negative-Skewed Data')

# 지수 변환
df_neg['exp_value'] = np.exp(df_neg['value'] / 1000)  # 스케일 조정 후 exp 적용

# 변환 후 데이터 분포 확인
plot_distribution(df_neg, 'exp_value', 'Exponential-Transformed Data')
```



- **원본 데이터**: 히스토그램이 왼쪽으로 길게 꼬리를 가지며, Q-Q 플롯에서도 점들이 직선에서 크게 벗어나 정규성을 만족하지 않음을 보여줍니다.
- **지수 변환**: 히스토그램은 종 모양에 가까워지고 Q-Q 플롯의 점들이 직선에 근접합니다. 데이터가 정규분포에 더 가까워졌고, 좌측 치우침을 완화하는 데 효과적인 방법임을 확인할 수 있습니다.

장단점 및 대안

변환 방법	장점	단점	대안
로그 변환	오른쪽으로 심하게 치우친 분포를 정규화하는 데 매우 효과적. 변수 간의 곱셈 관계를 덧셈 관계로 변환하여 해석을 용이하게 함.	0 또는 음수 값에 적용할 수 없음. 변환된 결과의 해석이 원래 단위가 아니므로 직관적이지 않을 수 있음.	**Box-Cox 변환**: 데이터로부터 최적의 람다(λ) 값을 찾아 자동으로 최적의 거듭제곱 변환을 수행하는 방법 (`scipy.stats.boxcox`). 람다가 0일 때 로그 변환과 유사한 효과를 냄. 0보다 큰 양수 데이터에만 적용 가능.
제곱근 변환	구현이 간단하고, 카운트 데이터의 분산을 안정시키는 데 효과적.	로그 변환보다 효과가 약함. 음수 값에 적용할 수 없음.	**거듭제곱 변환 (Power Transform)**: `x^λ` 형태로, 람다 값에 따라 다양한 변환이 가능 (λ=0.5는 제곱근 변환, λ=-1은 역수 변환 등). **Yeo-Johnson 변환**: Box-Cox 변환을 확장하여 0과 음수 값을 포함하는 데이터에도 적용할 수 있도록 만든 방법 (`sklearn.preprocessing.PowerTransformer`).

어떤 변환을 선택할 것인가?

- 변환의 선택은 정해진 규칙이 있기보다는, 데이터의 특성과 분석의 목적에 따라 달라집니다.
- 먼저 데이터의 분포를 시각화(히스토그램, Q-Q 플롯)하여 왜도의 정도를 파악합니다.
- 왜도가 심하지 않으면 **제곱근 변환**을, 왜도가 심하면 **로그 변환**을 우선적으로 시도해볼 수 있습니다.
- 여러 변환을 시도해보고, 변환 후의 분포가 가장 대칭적이고 정규분포에 가까워지는 방법을 선택하는 것이 일반적입니다.
- `scikit-learn`의 `PowerTransformer`를 사용하면 Box-Cox 또는 Yeo-Johnson 변환을 쉽게 적용하여 최적의 변환을 자동으로 찾을 수 있어 편리합니다.