Pareto Distribution & Transformations

The **Pareto Distribution** describes situations where **a small number of causes contribute to most of the effect**. It is also called the "**80/20 rule**" or "**Power Law Distribution**" because in many cases:

- **20% of people own 80% of the wealth.**
- **20% of customers generate 80% of the revenue.**
- **20% of bugs cause 80% of software crashes.**

- It is a **skewed** distribution with a **long tail on the right** (large values are rare but possible).

💡 **80/20 rule is applicable for α=1.16**

Power Law:

- It's a situation where small changes in one variable cause large proportional changes in another.

$$P(x) \propto x^{-\alpha}$$

- P(x) is the probability of the event,
- x is the size or magnitude of the event,
- α is a constant that dictates the steepness of the distribution (often called the **power-law exponent**).

🔷 Real-World Applications

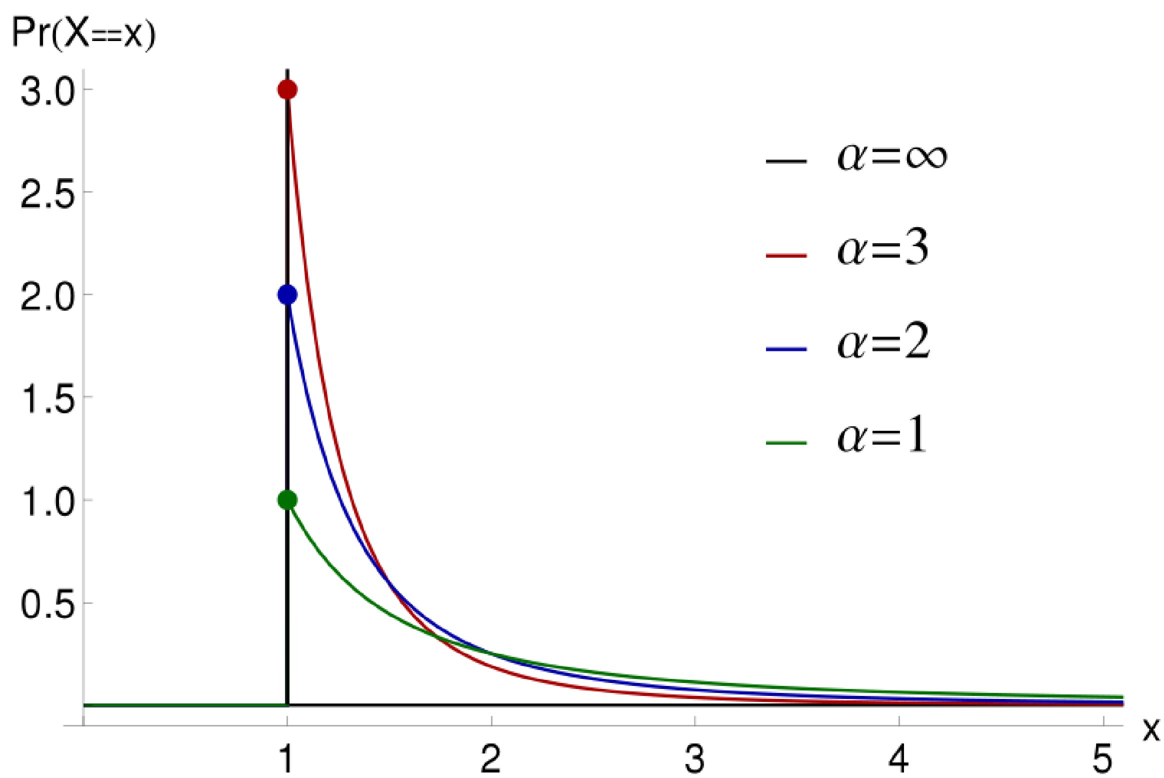
- **Economics** → Income & wealth distribution.
- **Business** → Sales revenue distribution among customers.
- **Internet** → 20% of websites get 80% of the traffic.
- **Social Media** → A few influencers get most of the engagement.
- **Natural Sciences** → City sizes, earthquake magnitudes, etc.

🔷 How It Works Internally?

The Pareto distribution is controlled by two parameters:

1. **Shape Parameter (`alpha`)/Pareto index**: Controls how steep the drop is.
    - High `alpha` → More equal distribution.
    - Low `alpha` → More extreme inequality.
        - the tail of the distribution is **much heavier**.
2. **Scale Parameter (`xm`)**: The minimum value (everything starts from here).
    - e.g., poverty line in wealth data

📌 **The smaller `alpha` is, the more extreme the inequality!**



Pareto distribution's PDF:

$$f(x; \alpha, x_m) = \frac{\alpha x_m^\alpha}{x^{\alpha+1}} \quad \text{for} \quad x \geq x_m$$

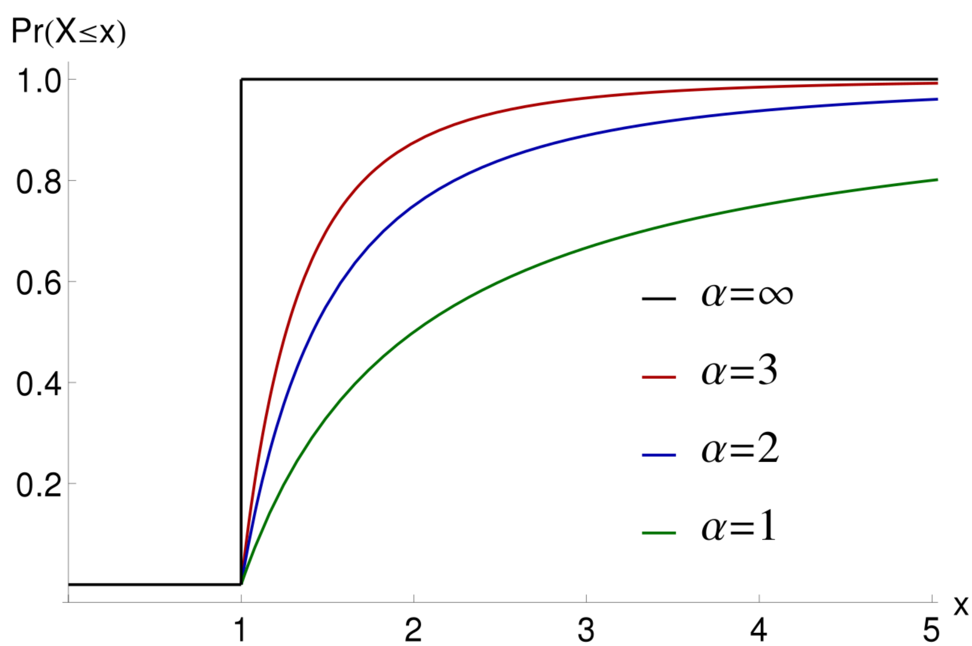
- x_m is the **scale parameter** (the minimum possible value of x)
- α is the **shape parameter** (the power-law exponent, which determines the steepness of the tail)
- x is the value of the random variable.

This formula applies for values of $x \geq x$ and for $x < x$, the probability is zero.

CDF of the Pareto distribution:

$$F(x; \alpha, x_m) = 1 - \left(\frac{x_m}{x}\right)^{\alpha} \quad \text{for} \quad x \geq x_m$$

- **α**: The shape parameter controls the steepness of the distribution.
- x_m: The scale parameter represents the minimum value of the distribution.



- As the α reduces, it takes more time to reach 1
  - cuz Higher alpha = More equality
  - **High α:** You're more likely to pick up larger values quickly, so you reach 1 faster.
  - **Low α:** You're more likely to get stuck with lots of small values, making the journey to 1 a longer one.

Q. How to detect if a distribution is Pareto Distribution?

Log-log plot

- You take log of x & y
- & plot a graph
- If the graph is a straight line, it's a pareto distribution.

Plot a pareto distribution plot:

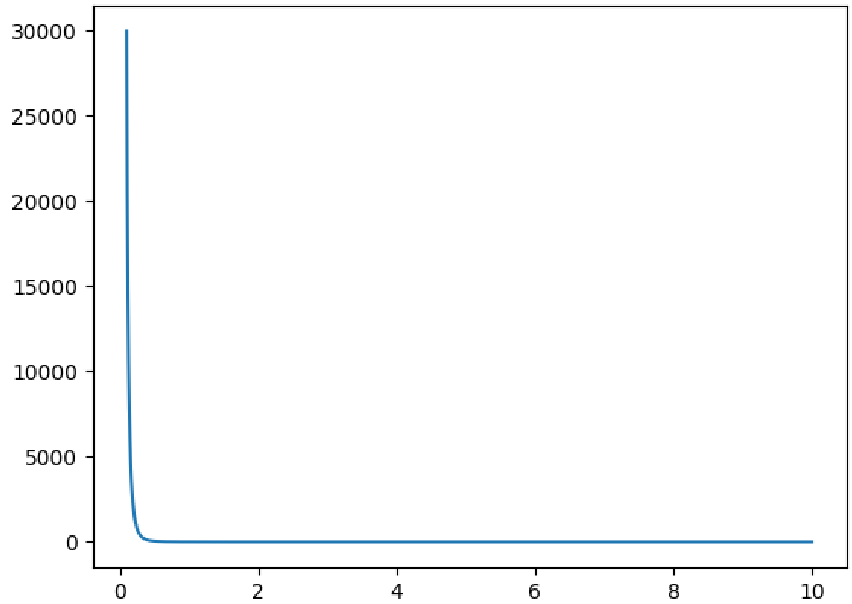
```
import numpy as np
import matplotlib.pyplot as plt

# Define the parameters of the Pareto distribution
alpha = 3
xm = 1

# Create an array of x values
x = np.linspace(0.1, 10, 1000)

# Calculate the y values of the Pareto distribution
y = alpha * (xm**alpha) / (x**(alpha+1))


plt.plot(x,y)
```



- `y = alpha * (xm**alpha) / (x**(alpha+1))` is the formula for pareto distribution PDF

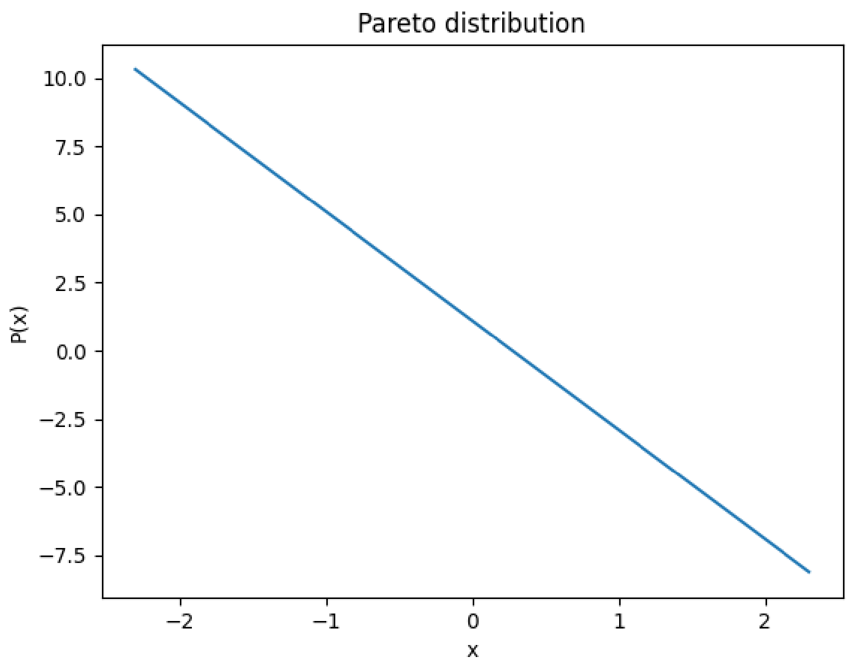
$$f(x;\alpha,x_m) = \frac{\alpha x_m^\alpha}{x^{\alpha+1}} \quad \text{for} \quad x \geq x_m$$

- Now take log of x & y

```
# Create the log-log plot
plt.plot(np.log(x),np.log(y))

# Add labels and a title
plt.xlabel('x')
plt.ylabel('P(x)')
plt.title('Pareto distribution')
```

```
# Show the plot
plt.show()
```



- Since it was a pareto distribution, you get a straight line

Q-Q Plot

- Generate random data from `stats.pareto`

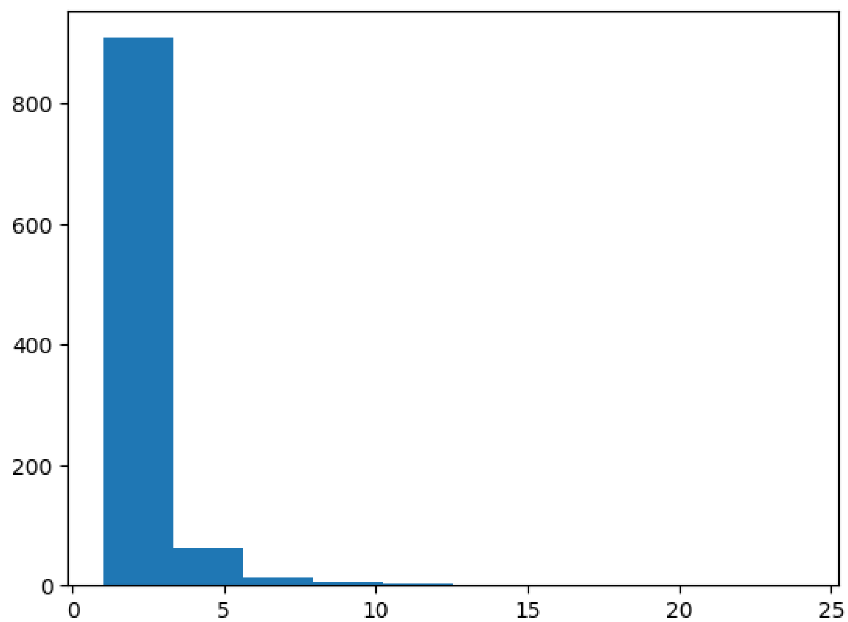
```
import numpy as np
import scipy.stats as stats
import statsmodels.api as sm
import matplotlib.pyplot as plt

# Define the parameters of the Pareto distribution
alpha = 2
xm = 1
```

```
# Generate a set of random data from the Pareto distribution
x = stats.pareto.rvs(b=alpha, scale=xm, size=1000)
```

- `b=alpha` : The shape parameter α.
- `scale=xm` : The scale parameter x_m

```
plt.hist(x)
```



- Fit a Pareto distribution to the data

```
# Fit a Pareto distribution to the data
params = stats.pareto.fit(x, floc=0)

# Create a Pareto distribution object with the fitted paramet
ers
dist = stats.pareto(b=params[0], scale=params[2])
```

- `floc=0` parameter is used to fix the location parameter to 0 (i.e., it ensures that the Pareto distribution starts at 0, which is typical for a Pareto distribution where the minimum value x_m is set to 0).
- `params[0]`: Estimated shape parameter α.
- `params[2]`: Estimated scale parameter x_m.

```
params

Output: (2.026531179662769, 0, 1.0002654443811565)
```

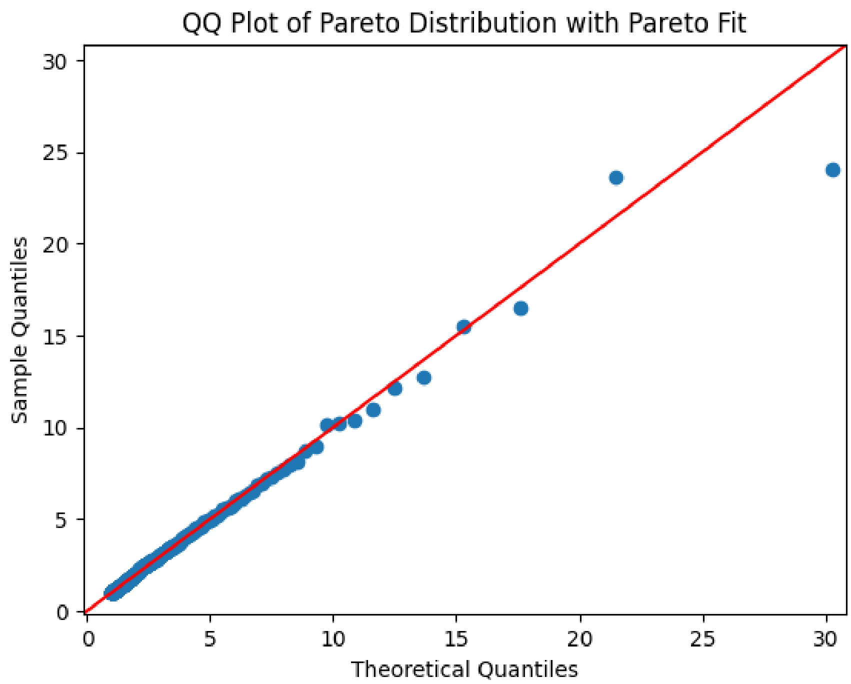
- `params[0]`: Estimated **shape parameter α** (approximated to **2.03**).
- `params[1]`: Fixed **location parameter μ** (set to **0** due to `floc=0`).
- `params[2]`: Estimated **scale parameter** x_m (approximated to **1.0003**, close to the true value of 1).

Now create a Q-Q plot

```
# Create a QQ plot of the data using the Pareto distribution
fig = sm.qqplot(x, dist=dist, line='45')

# Add a title and labels to the plot
plt.title('QQ Plot of Pareto Distribution with Pareto Fit')
plt.xlabel('Theoretical Quantiles')
plt.ylabel('Sample Quantiles')

# Show the plot
plt.show()
```



- Even if few points are going outside the line we can say that it's pareto distribution.

Transformations

- Mathematical function to a dataset to change its distribution, scale, or other characteristics
- You try to convert the data into normal distribution

▼ **DeepSeek Response**

What Are Transformations?

Transformations are mathematical operations applied to data to:

1. **Simplify analysis** (e.g., make data linear or symmetric).
2. **Meet assumptions** of statistical models (e.g., normality, homoscedasticity).
3. **Improve interpretability** or visualization.

Common Transformations

1. Log Transformation

- **Formula**: $Y = \log(X)$
- **Use Case**:
  - Reduces **right skew** (e.g., income, Pareto-distributed data).
  - Converts multiplicative relationships to additive.
- **Example**:

```
import numpy as np
x = np.array([1, 10, 100, 1000])
y = np.log10(x)  # Transformed: [0, 1, 2, 3]
```

2. Square Root Transformation (RARE)

- **Formula**: $Y = \sqrt{X}$
- **Use Case**:
  - Mild right skew.
  - Count data (e.g., Poisson-like distributions).

3. Box-Cox Transformation

- **Formula**:

$$Y = \begin{cases} \frac{X^{\lambda}-1}{\lambda} & \text{if } \lambda \neq 0, \\ \ln(X) & \text{if } \lambda = 0. \end{cases}$$

- **Use Case**:
  - Automatically selects the best λ to stabilize variance/normality.
- **Python Example**:

```
from scipy.stats import boxcox
transformed_data, lambda_val = boxcox(original_data)
```

4. Reciprocal Transformation

- **Formula**:

$$Y = \frac{1}{X}$$

- **Use Case**:
  - Severe right skew.
  - Inverse relationships (e.g., speed vs. time).
  - Used for **Left-skewed data**

Log Transformation

- This is used to transform highly skewed data to make it more normally distributed. It's particularly useful for data with **exponential growth** or **outliers**.
- It is useful when the data contains **very large values** or is **right-skewed**.

$$x_{\log} = \log(x)$$

- This transformation is typically applied when the values are **strictly positive**.
    - Cuz you can't calculate log of a negative number.
- If the data includes zeros, you might need to add a small constant to avoid taking the log of zero.

Box-Cox Transformation

- The **Box-Cox transformation** is often used to stabilize **variance** and make the data more **normal**.
- It's ideal for **positively skewed** data and helps normalize the data for more accurate statistical modeling.
- The transformation works by applying a **power transformation** to the data, which can make the distribution more symmetric (normal).
    - We try to figure out the appropriate value of x^{λ} →$x^2/x^3/x^4$, etc
    - **λ varies from -5 to +5**
        - Could be 1.6, 2.78, etc.

Mathematical Formula

$$y(\lambda) = \frac{x^{\lambda} - 1}{\lambda}, \quad \lambda \neq 0$$

When $\lambda = 0$, the transformation becomes the **logarithm** transformation:

$$y(0) = \log(x)$$

- **x** is the original data value.

- **λ** is the transformation parameter (which can be estimated from the data).
- If $\lambda=0$, the transformation is equivalent to the log transformation.
- **y(λ)** is the transformed value.

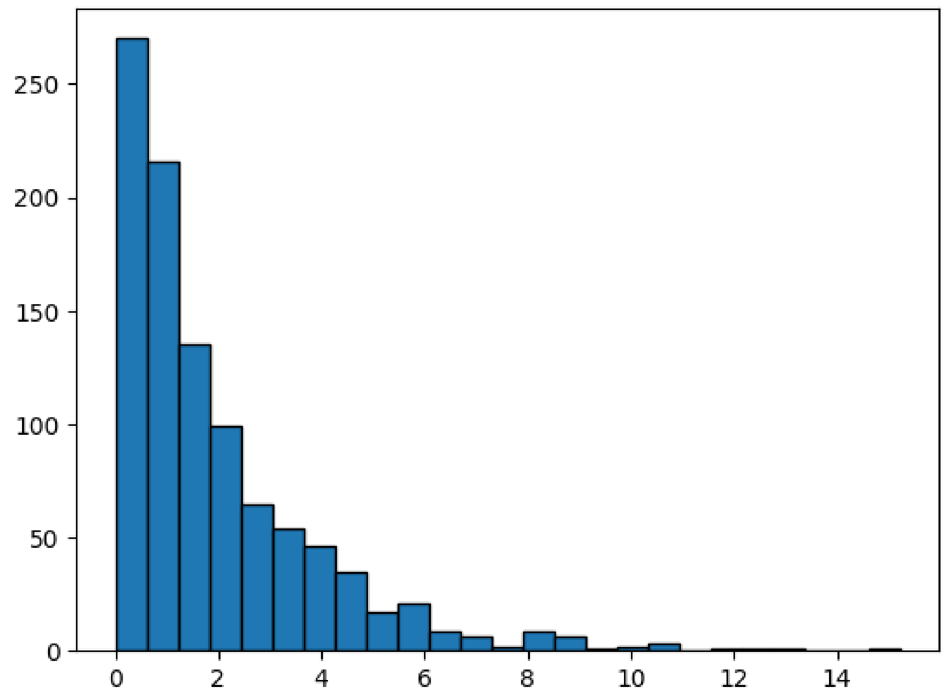
Key Features of Box-Cox Transformation

- **λ Parameter**: The parameter λ controls the transformation. The value of λ can be **estimated** based on the data or manually chosen.
    - $\lambda=1$ corresponds to no transformation (identity transformation).
    - $\lambda=0$ corresponds to a log transformation.
    - $\lambda>1$ or $\lambda<0$ applies a power transformation.
- **Data Requirements**: Box-Cox can only be applied to **positive data** (i.e., all values of x must be greater than 0).

Limitation of only positive numbers is solved by **Yeo-Johnson Transform**.

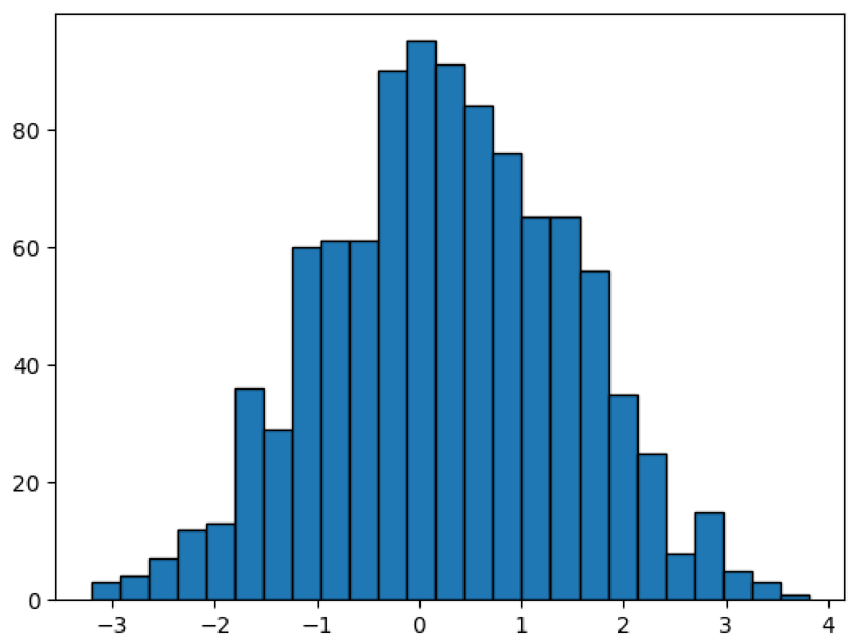
Let's apply the Box-Cox transformation to positively skewed data.

```
data= np.random.exponential(2, 1000)
plt.hist(data, edgecolor='k', bins=25);
```



- Apply Box-Cox transformation on this

```
transformed_data, lambda_est = stats.boxcox(data)
plt.hist(transformed_data, edgecolor='k', bins=25);
```



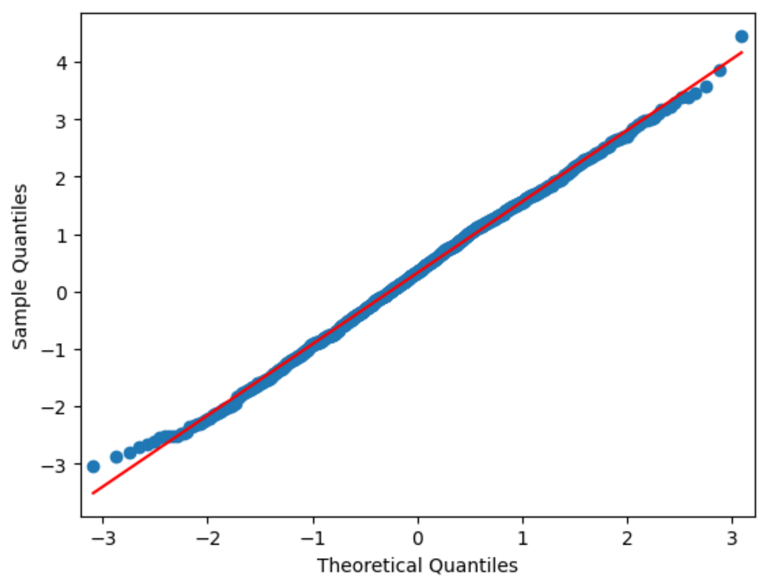
- `np.random.exponential(scale=2, size=1000)`: Generates 1000 random values following an **exponential distribution** (which is positively skewed)
    - mean=~2

```
lambda_est

Output: 0.2348188810505703
```

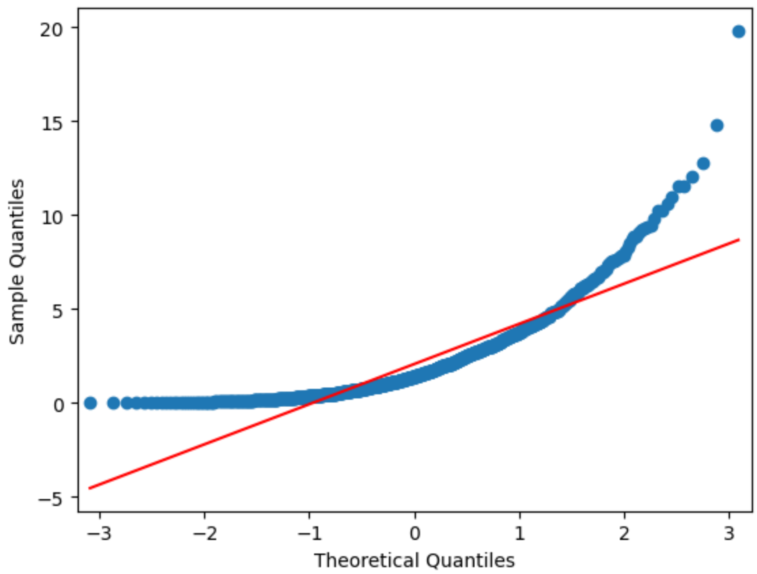
- Now, **Check Normality with Q-Q Plot**

```
sm.qqplot(transformed_data, line='s')
```



Without transformation:

```
sm.qqplot(data, line='s')
```



20
15
10
5
0
−5
−3
−2
−1
0
1
2
3
Sample Quantiles
Theoretical Quantiles