

## Non-Linear Data Transformation

Square Root

Logarithmic

Cubic Root

Box-Cox

Yeo-Johnson

by Franklin Garcia

## Introduction

At the stage of data processing in Data Science it is fundamental to **transform our data, which do not follow a normal distribution**, to obtain a **better result in our models** 😊. In this deepnote we will see how to transform those non-linear data with the square root, logarithmic, cube root, Box-Cox and Yeo-Jhonson methods.

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

```
df=pd.read_csv('data.csv')
df.head(3)
```

	Moderate Positi...	Highly Positive S...	Moderate Negat...	Highly Negative ...	
0	0.8999903238	2.895073786	11.18074757	9.027484718	
1	1.1135538	2.962384883	10.84293804	9.009761974	
2	1.156830061	2.966377528	10.81793375	9.006134119	

## Data set

This is a data set from GitHub by Marsja you can see [here](#).

So this data set have four variables:

- Positive Bias Moderate (right bias)
- High Positive Bias (right bias)
- Negative Bias Moderate (left bias)
- High Negative Bias (left bias)

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 4 columns):
 #   Column                                Non-Null Count  Dtype  
---  -
 0   Moderate Positive Skew               10000 non-null  float64
 1   Highly Positive Skew                 10000 non-null  float64
 2   Moderate Negative Skew               10000 non-null  float64
 3   Highly Negative Skew                 10000 non-null  float64
dtypes: float64(4)
memory usage: 312.6 KB
```

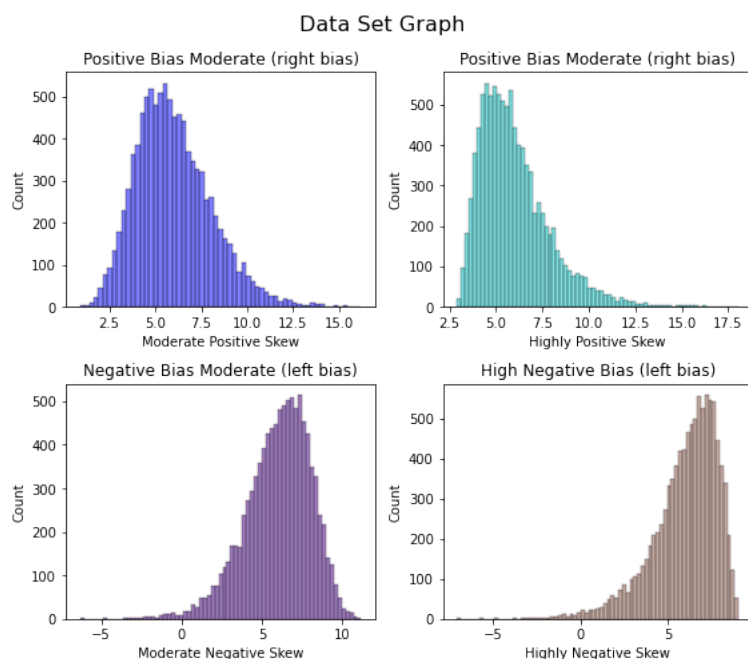
```
fig, axes = plt.subplots(2,2, figsize=(8,7))
plt.suptitle('Data Set Graph', fontsize=16)

# Positive Bias Moderate (right bias)
sns.histplot(ax=axes[0,0], data=df['Moderate Positive Skew'], color='blue', alpha=0.5)
axes[0,0].set_title('Positive Bias Moderate (right bias)')

# High Positive Bias (right bias)
sns.histplot(ax=axes[0,1], data=df['Highly Positive Skew'],color='c', alpha=0.5)
axes[0,1].set_title('Positive Bias Moderate (right bias)')

# title
sns.histplot(ax=axes[1,0], data=df['Moderate Negative Skew'],color='indigo', alpha=0.5)
axes[1,0].set_title('Negative Bias Moderate (left bias)')

# title
sns.histplot(ax=axes[1,1], data=df['Highly Negative Skew'],color='tab:brown', alpha=0.5)
axes[1,1].set_title('High Negative Bias (left bias)')
fig.tight_layout()
plt.show()
```



## Asymmetry

Other way to see if our data set is asymmetric is check it the Asymmetry.

Formally, we say that a frequency distribution is symmetric if the arithmetic mean  $\bar{X}$  is equal to the median  $\tilde{X}$ . This means that in any other case, the distribution is asymmetric.

**Interpretation:**

```
skewness=0 : Symmetric distribution
skewness>0 : Asymmetric distribution to the right
skewness<0 : Asymmetric distribution to the left
```

```
# Calculation
df.skew()
```

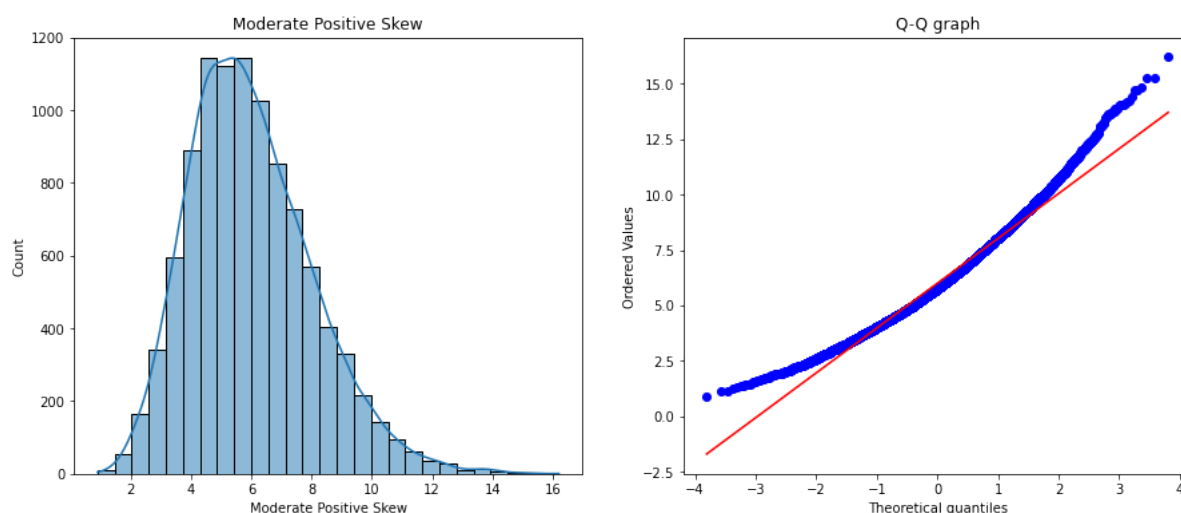
```
Moderate Positive Skew    0.656308
Highly Positive Skew      1.271249
Moderate Negative Skew   -0.690244
Highly Negative Skew     -1.201891
dtype: float64
```

## Square Root

Normally used when the data are moderately skewed. Generally used to reduce right skewed data. 

```
def diagnostic_graph(data_set, variable):
    plt.figure(figsize=(15,6))
    plt.subplot(1,2,1)
    plt.title(variable)
    sns.histplot(data=data_set, x=variable, kde=True, bins=27);
    plt.subplot(1,2,2)
    stats.probplot(data_set[variable], dist='norm', plot=plt);
    plt.title('Q-Q graph')
    plt.show()
```

```
diagnostic_graph(df, 'Moderate Positive Skew');
```

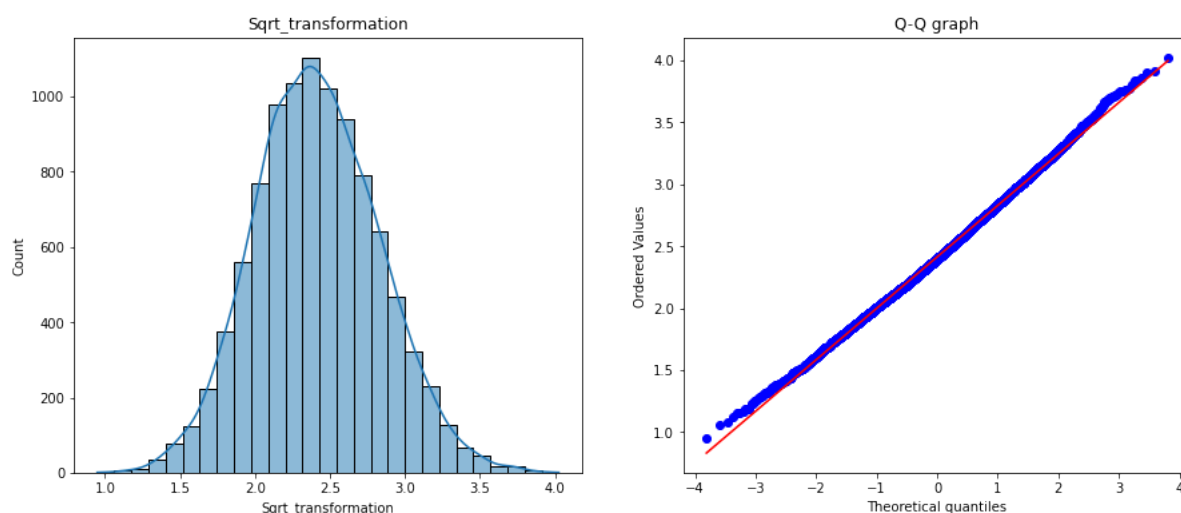


## Data Transformation

```
df_transformed = df.copy()
# Transformation
df_transformed['Sqrt_transformation'] = np.sqrt(df['Moderate Positive Skew'])
df_transformed.head(3)
```

	Moderate Positi...	Highly Positive S...	Moderate Negat...	Highly Negative ...	Sqrt_transforma...	
0	0.8999903238	2.895073786	11.18074757	9.027484718	0.9486781982	
1	1.1135538	2.962384883	10.84293804	9.009761974	1.055250586	
2	1.156830061	2.966377528	10.81793375	9.006134119	1.075560347	

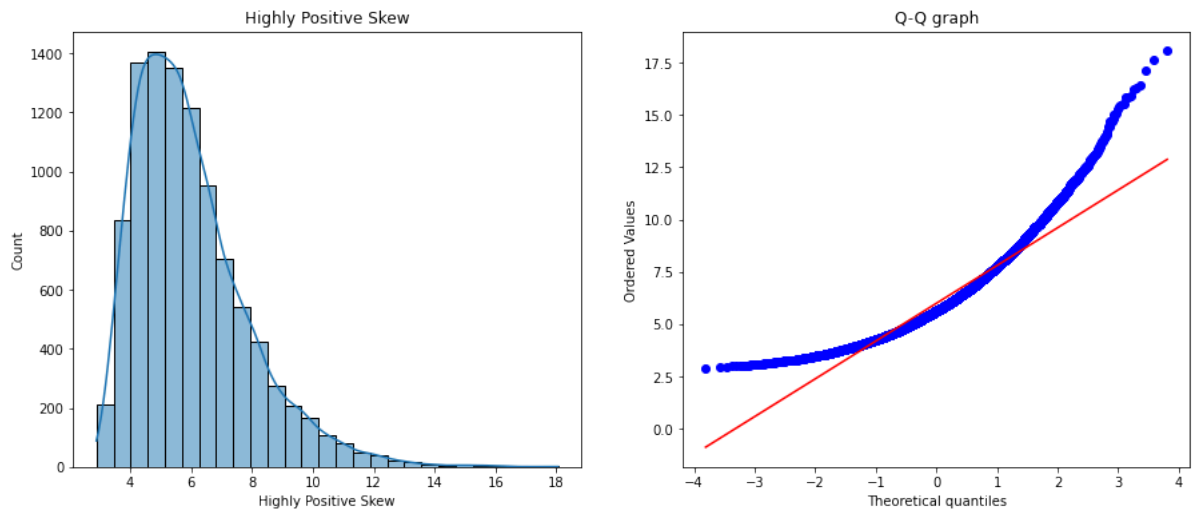
```
# Graph after Square Root transformation
diagnostic_graph(df_transformed, 'Sqrt_transformation')
```



# Logarithmic

The logarithm,  $x$  to log base 10 of  $x$ , or  $x$  to log base  $e$  of  $x$  ( $\ln x$ ), or  $x$  to log base 2 of  $x$ , is a strong transformation with a major effect on distribution shape. It is commonly used for reducing right skewness and is often appropriate for measured variables. It can not be applied to zero or negative values. One unit on a logarithmic scale means a multiplication by the base of logarithms being used. Exponential growth or decline. **!**

```
diagnostic_graph(df_transformed, 'Highly Positive Skew')
```

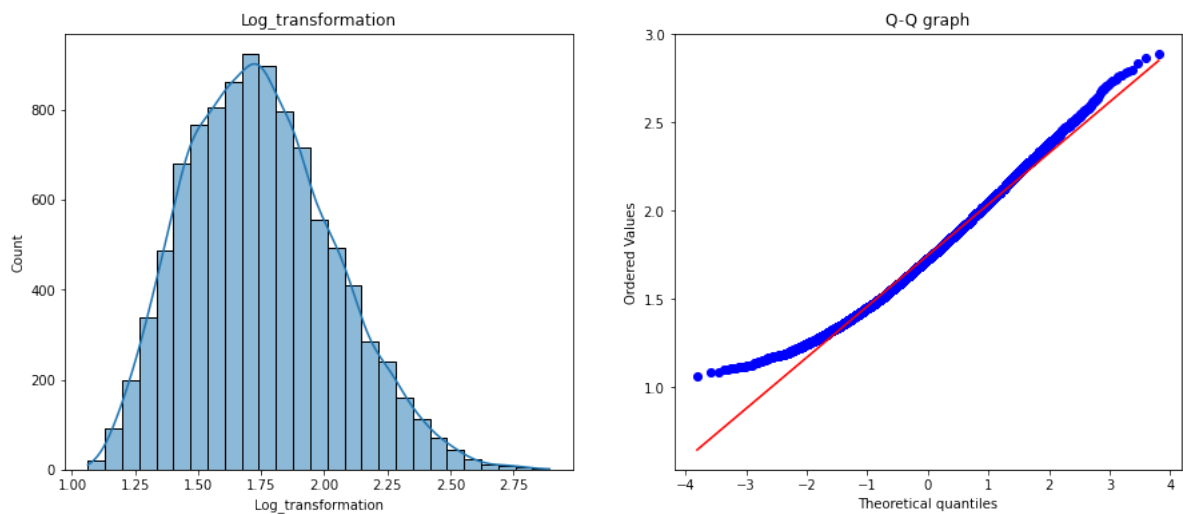


## Data Transformation


```
df_transformed['Log_transformation'] = np.log(df['Highly Positive Skew'])
df_transformed.head(3)
```

	Moderate Positi...	Highly Positive S...	Moderate Negat...	Highly Negative ...	Sqrt_transforma...	Log_transformat...	
0	0.8999903238	2.895073786	11.18074757	9.027484718	0.9486781982	1.063010598	
1	1.1135538	2.962384883	10.84293804	9.009761974	1.055250586	1.085994648	
2	1.156830061	2.966377528	10.81793375	9.006134119	1.075560347	1.087341521	

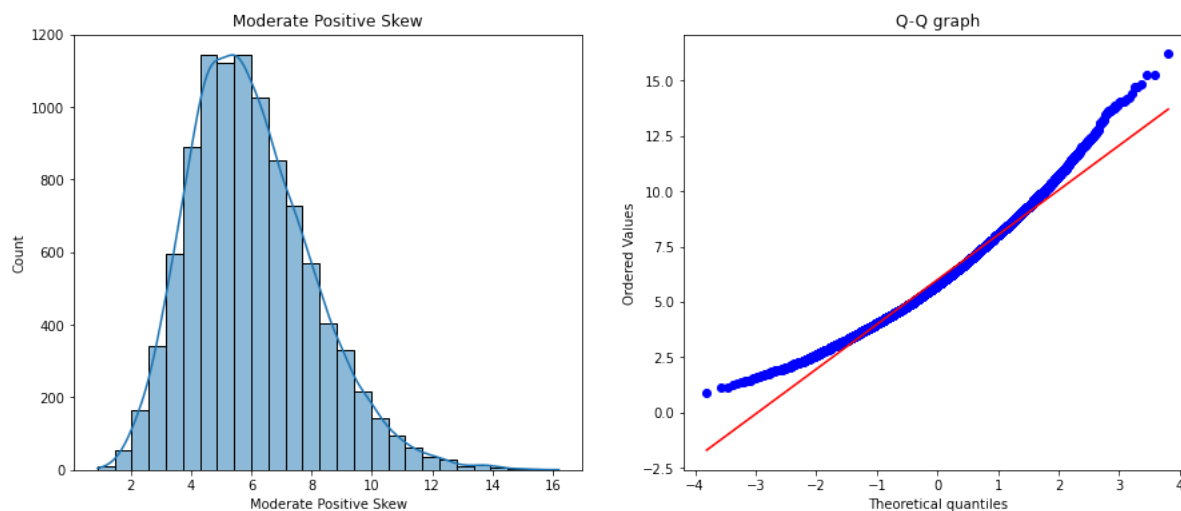
```
# Graph after Log transformation
diagnostic_graph(df_transformed, 'Log_transformation')
```



## Cubit Root

The cube root,  $x$  to  $x^{1/3}$ . This is a fairly strong transformation with a substantial effect on distribution shape: it is weaker than the logarithm. It is also used for reducing right skewness, and has the advantage that it can be applied to zero and negative values. Note that the cube root of a volume has the units of a length. It is commonly applied to rainfall data. 

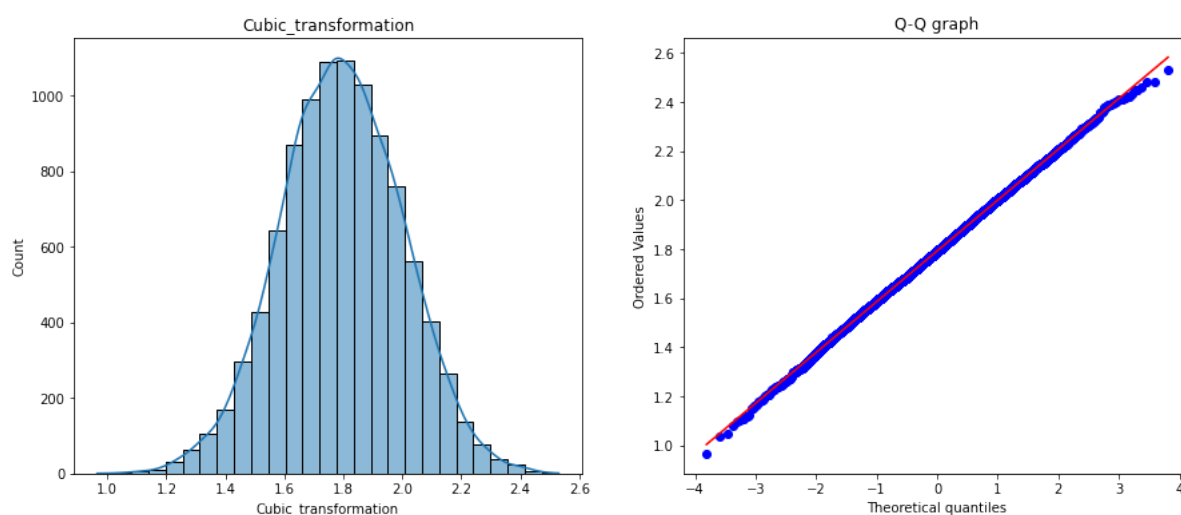
```
diagnostic_graph(df_transformed, 'Moderate Positive Skew')
```



## Data Transformation

```
# Add a new column with the transformation
df_transformed['Cubic_transformation'] = np.cbrt(df['Moderate Positive Skew'])
```

```
# After Cubic transformation
diagnostic_graph(df_transformed, 'Cubic_transformation')
```



## Power Transformer

**Power Transformer** is used to convert feature data to a more Gaussian (normal) distribution. This transformer is very useful for heterocedasticity data (non-constant variance). It supports two methods:

- Box-Cox transformer
- Yeo-Johnson

The **Box-Cox** transformation requires the data to be **strictly positive**, while the **Yeo-Johnson** transformation can handle both **positive and negative**. **!!**

## Yeo-Johnson

```
from sklearn.preprocessing import PowerTransformer
df_pt = df.copy()

# Model Creation
p_scaler = PowerTransformer(method='yeo-johnson')

# fitting and transforming the model
df_yjt = pd.DataFrame(p_scaler.fit_transform(df_pt), columns=['Moderate Positive Skew', 'Highly Positive Skew', 'Moderate
```

```
# Transformed Data Set
df_yjt.head()
```

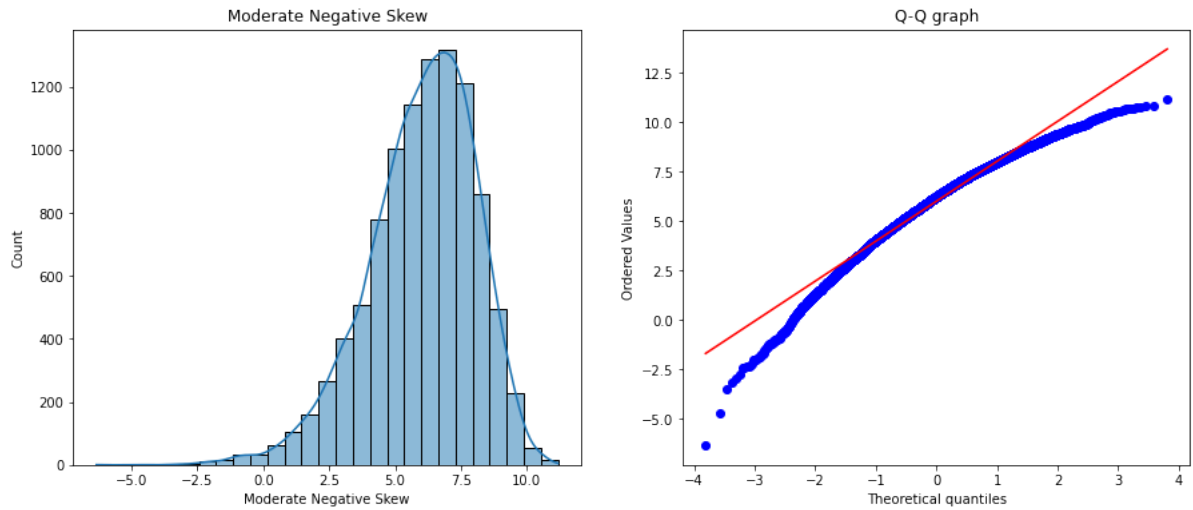
	Moderate Positi...	Highly Positive S...	Moderate Negat...	Highly Negative ...	
0	-3.764951581	-2.745615469	3.089553627	2.131558782	
1	-3.48940587	-2.638128635	2.855629137	2.115778806	
2	-3.436218346	-2.631854811	2.838452506	2.112552128	
3	-3.307816456	-2.578963324	2.801858298	2.107210459	
4	-3.238306511	-2.560786962	2.794015907	2.090492644	

```
df_yjt.describe()
```

	Moderate Positi...	Highly Positive S...	Moderate Negat...	Highly Negative ...	
cou...	10000	10000	10000	10000	
me...	9.549694369e-16	4.547473509e-17	9.094947018e-17	-2.273736754e-16	
std	1.000050004	1.000050004	1.000050004	1.000050004	
min	-3.764951581	-2.745615469	-2.970698007	-2.439111143	
25%	-0.6826636919	-0.7351637354	-0.6925935485	-0.7016429737	
50%	-0.01112970907	0.009128949241	0.036185216	0.07484809464	
75%	0.6911628527	0.701639102	0.7174149283	0.7757420776	
max	3.537717563	3.015598786	3.089553627	2.131558782	

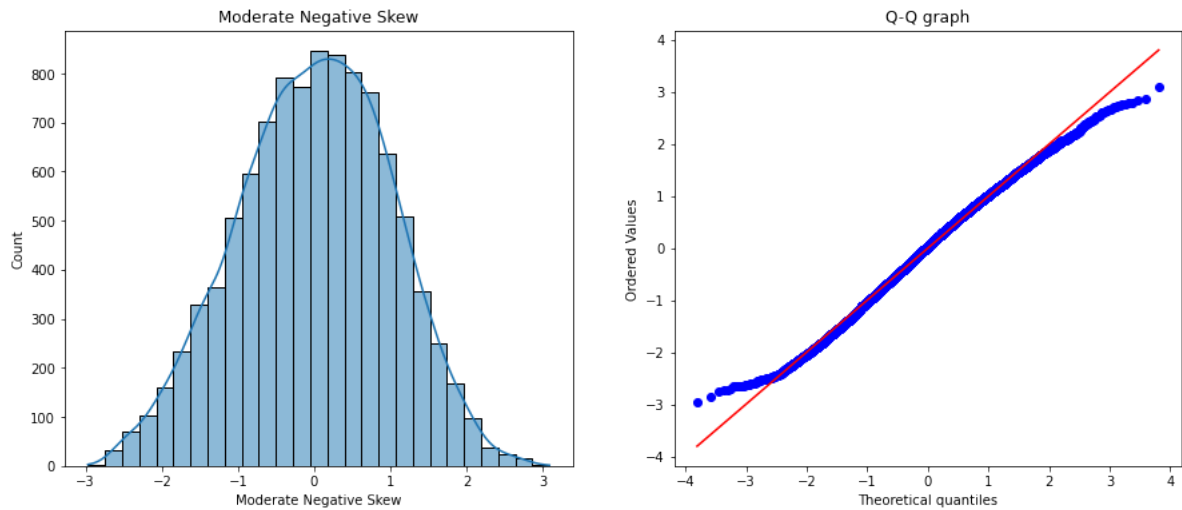
## Before Data transformation

```
diagnostic_graph(df_pt, 'Moderate Negative Skew')
```



After Data Transformation

```
diagnostic_graph(df_yjt, 'Moderate Negative Skew')
```



Box-Cox ✓

Box-Cox can only be applied to strictly positive data. In both methods, the transformation is parameterized by  $\lambda$  which is determined through maximum likelihood estimation !

```
pt = PowerTransformer(method='box-cox', standardize=False)

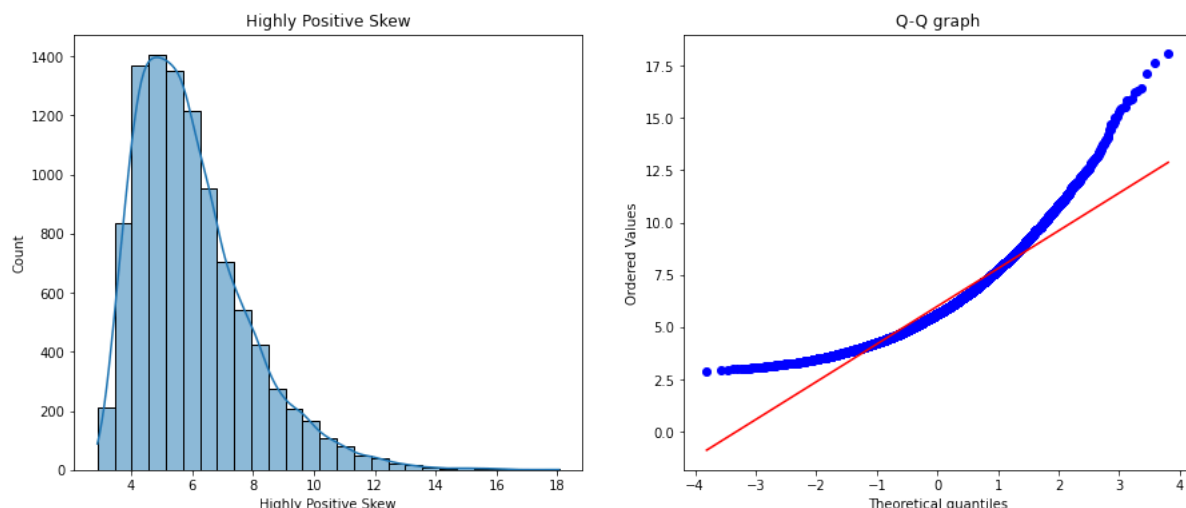
# Only positive data
df_bxcx = pd.DataFrame(p_scaler.fit_transform(df_pt.iloc[:, 0:2]), columns=['Moderate Positive Skew', 'Highly Positive Skew'])
```

```
df_bxcx.head(3)
```

	Moderate Positi...	Highly Positive S...	
0	-3.764951581	-2.745615469	
1	-3.48940587	-2.638128635	
2	-3.436218346	-2.631854811	

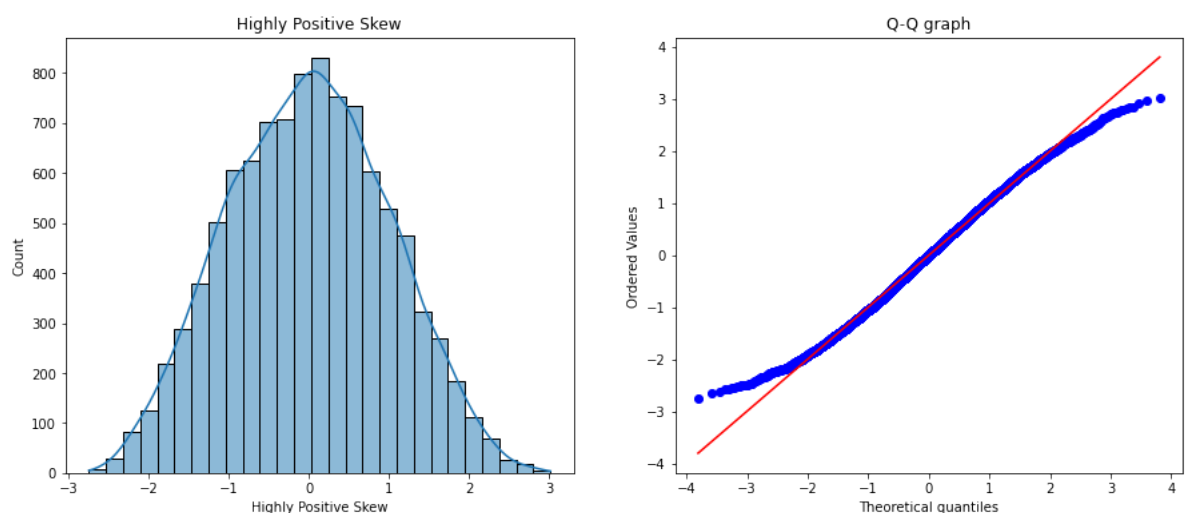
Before Data Transformation

```
diagnostic_graph(df_pt, 'Highly Positive Skew')
```



## After Data Transformation

```
diagnostic_graph(df_bxcx, 'Highly Positive Skew')
```



## Conclusions

As we could see there are many transformations, some for positive data, others for negative data or even some for both. Likewise, there are transformations for data skewed to the right or to the left. As a data scientist we should know these transformations and use them depending on the context of our data. Transformations are an essential part of data science knowledge since data are usually not normalized. With this knowledge we add one more tool in our toolbox to face new challenges.

## References

Boston College. *Transformations: an introduction*. <http://fmwww.bc.edu/repec/bocode/t/transint.html>

Naren Castellon. *Transformación de Datos con Python (2022)*. [https://www.youtube.com/watch?v=2lyrjUu1X\\_I&list=LL&index=2](https://www.youtube.com/watch?v=2lyrjUu1X_I&list=LL&index=2)

Scikit-Learn. *Preprocessing data*. <https://scikit-learn.org/stable/modules/preprocessing.html>