**Data transformations**

- Generally data has 3 types of distributions
  - Normal distribution
  - Postive Skewed
  - Negative Skewed
- All the maths concepts developed by assumption as Data follows Normal distribution
- Data skew happens because of many reasons, in that one reason is ouliers
- Once we perform the outlier analysis then we can apply transformation methods , if the data does not follows normality
- We have some transformation methods mentioned below
  - Log Transformation
  - Exponential Transformation
  - Reciprocal Transformation
  - Square root Transformation
  - Power Transformation

$Step-1$

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

$Step-2$

**Read the data**

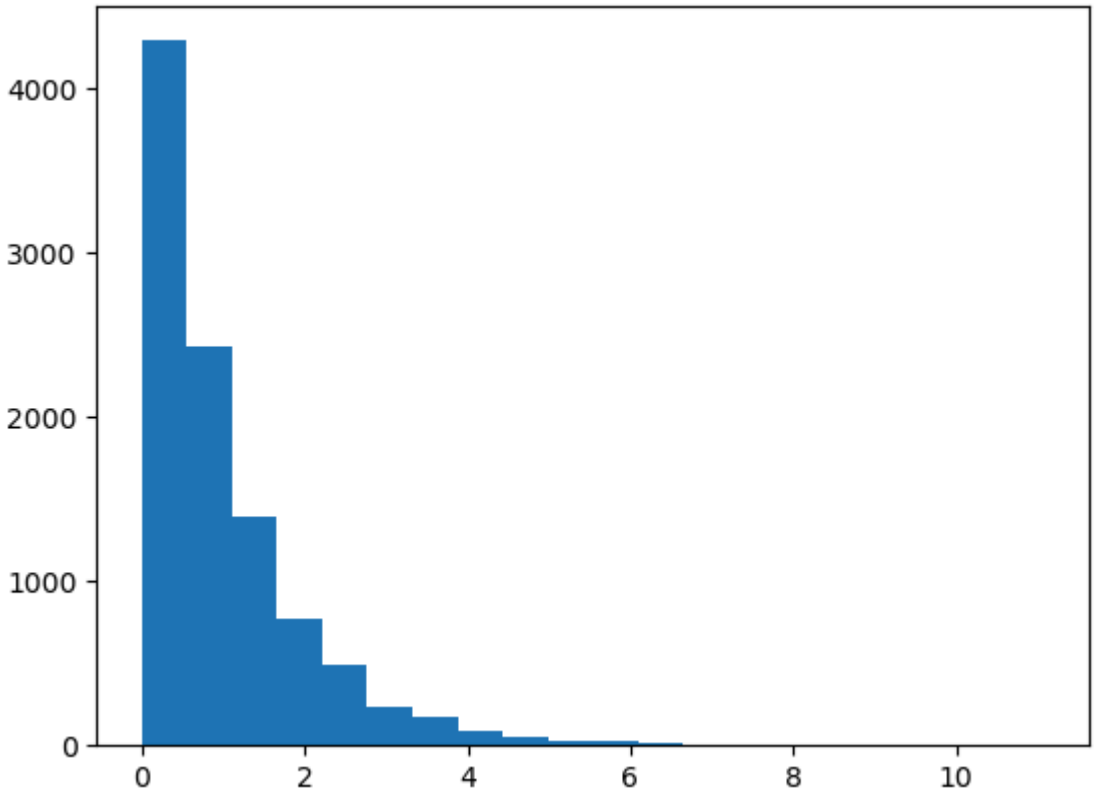
In [3]:
```
exp_data= np.random.exponential(size=10000)
exp_data
```

Out[3]:
```
array([1.85085097, 0.70753286, 0.59755911, ..., 0.23733074, 0.42797443,
       1.52496464])
```

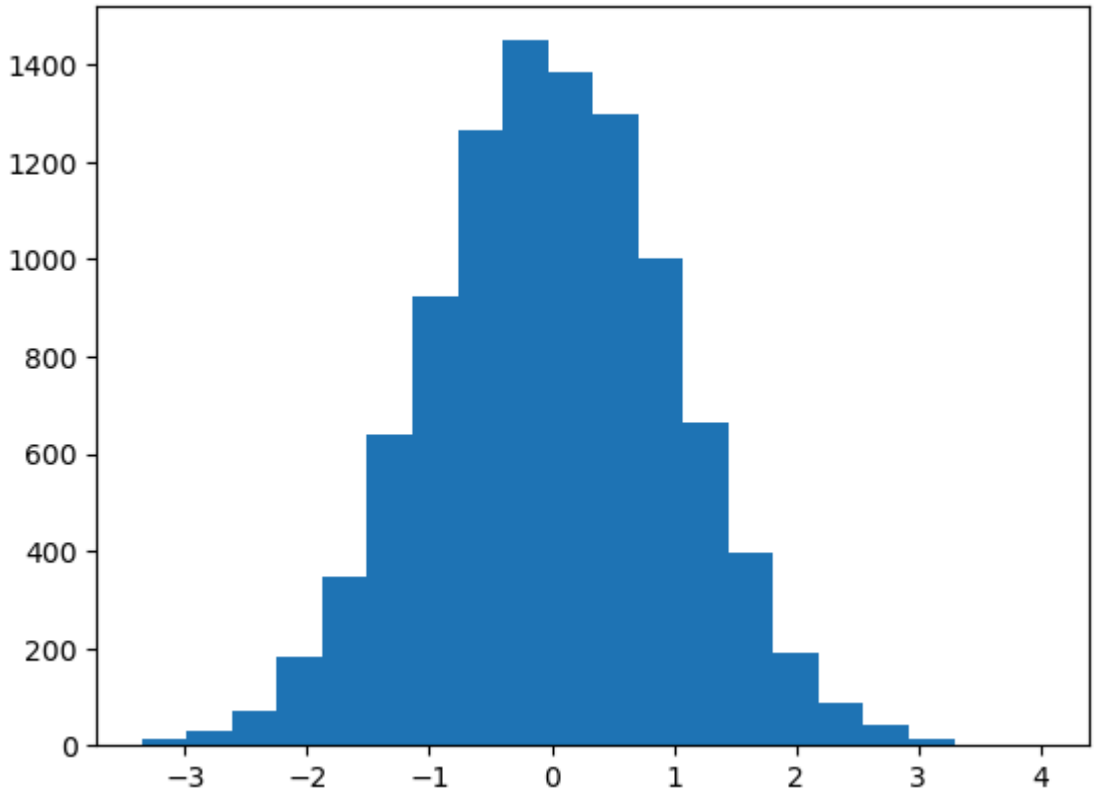
$Step-3$

**plot the data**

In [4]:
```
plt.hist(exp_data,bins=20,label='Exponential')
plt.show()
```




In [5]:
```
norm_data= np.random.normal(size=10000)
plt.hist(norm_data,bins=20,label='Exponential')
plt.show()
```



**Goal**

**Convert exp_data to Normal distribution**

bins= interval

$Step - 4$

**Log transformation**

- Log transformation means performing logarithm operations on origianl data
- It is an approach to convert into Noraml distribution
- Log means natural logarithm base=e

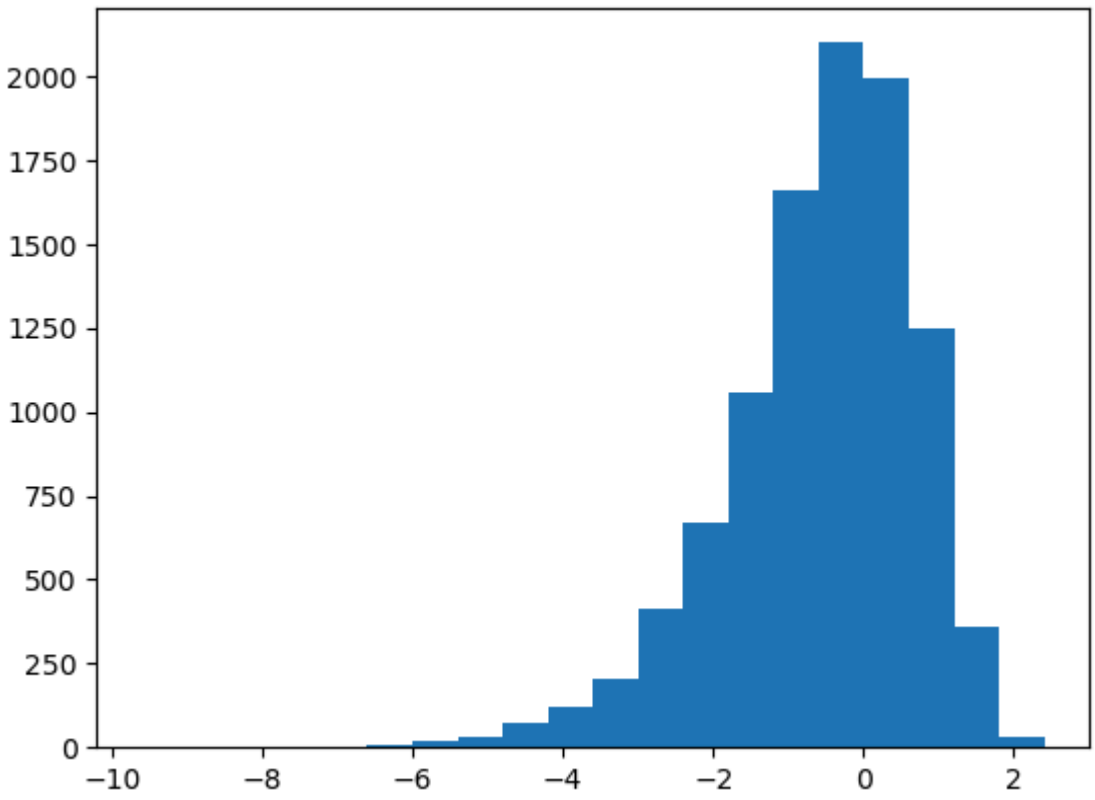
In [6]:
```
x=2
np.log(x)
```

Out[6]:
```
0.6931471805599453
```

In [7]:
```
log_data=np.log(exp_data)
log_data
```

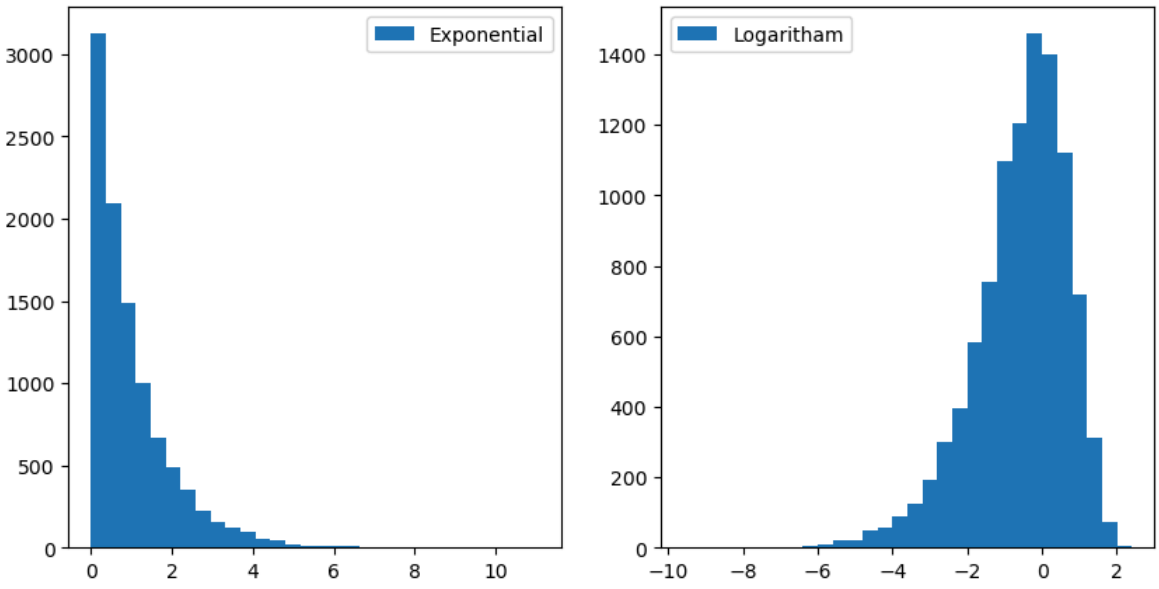
Out[7]:
```
array([ 0.61564552, -0.3459712 , -0.51490208, ..., -1.43830058,
       -0.84869182,  0.42197122])
```

In [8]:
```
plt.hist(log_data,bins=20,label='Logarithm')
plt.show()
```




In [11]:
```
plt.figure(figsize=(10,5))
plt.subplot(1,2,1).hist(exp_data,bins=30,label='Exponential')
```

```
plt.legend()
plt.subplot(1,2,2).hist(log_data,bins=30,label='Logaritham')
plt.legend()
plt.show()
```



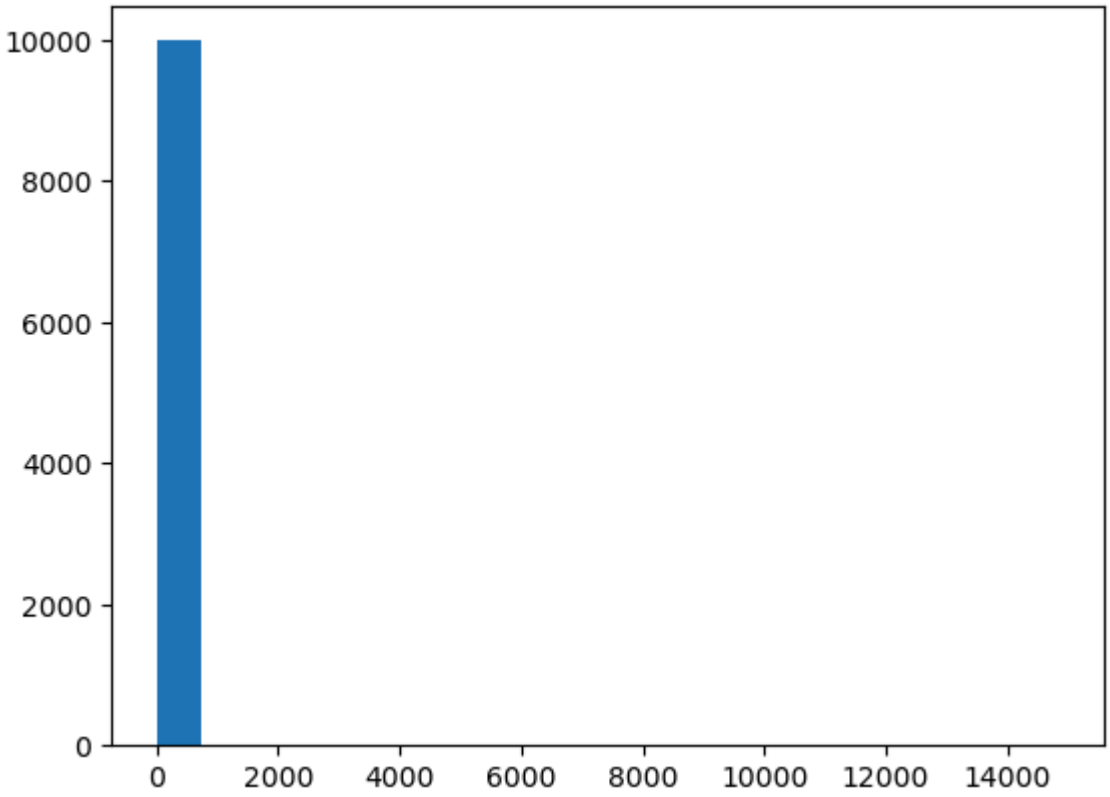

**Reciprocal Transformations**

- Reciprocal will fail when values are low
- For examples x=0 then 1/0 fail

In [12]:
```
x=2
1/x
```

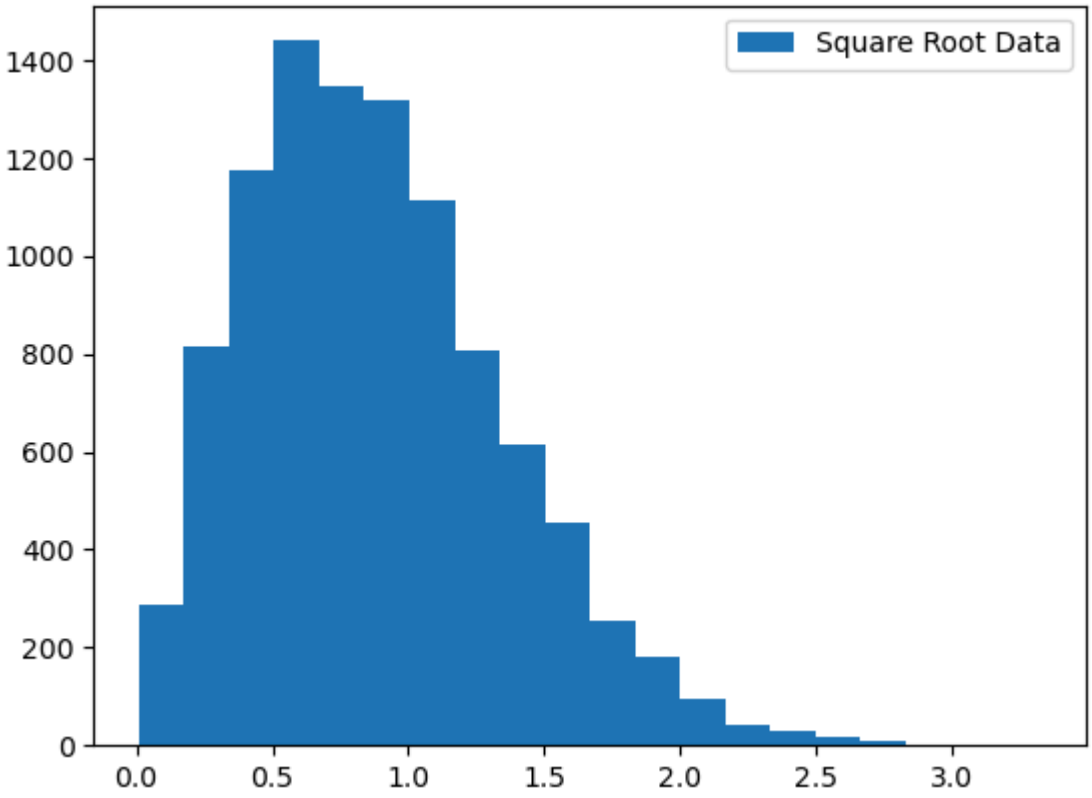
Out[12]:
```
0.5
```

In [14]:
```
resiprocal_data=np.reciprocal(exp_data)
plt.hist(resiprocal_data,bins=20,label="Reciprocal")
plt.show()
```



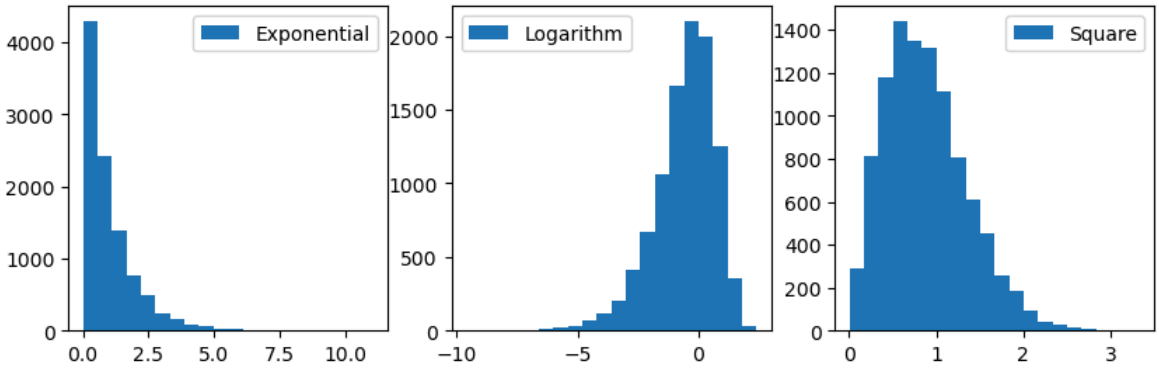

**Square root transformation**

In [15]:
```
square_root_data = np.sqrt(exp_data)
plt.hist(square_root_data, bins=20, label="Square Root Data")
plt.legend()
plt.show()
```




In [20]:
```
plt.figure(figsize=(10,3))
log_data = np.log(exp_data)
```

```
plt.subplot(1,3,1).hist(exp_data,bins=20,label='Exponential')
plt.legend()
log_data = np.log(exp_data)
plt.subplot(1,3,2).hist(log_data,bins=20,label='Logarithm')
plt.legend()
square_data = np.sqrt(exp_data)
plt.subplot(1,3,3).hist(square_data,bins=20,label='Square')
plt.legend()
plt.show()
```



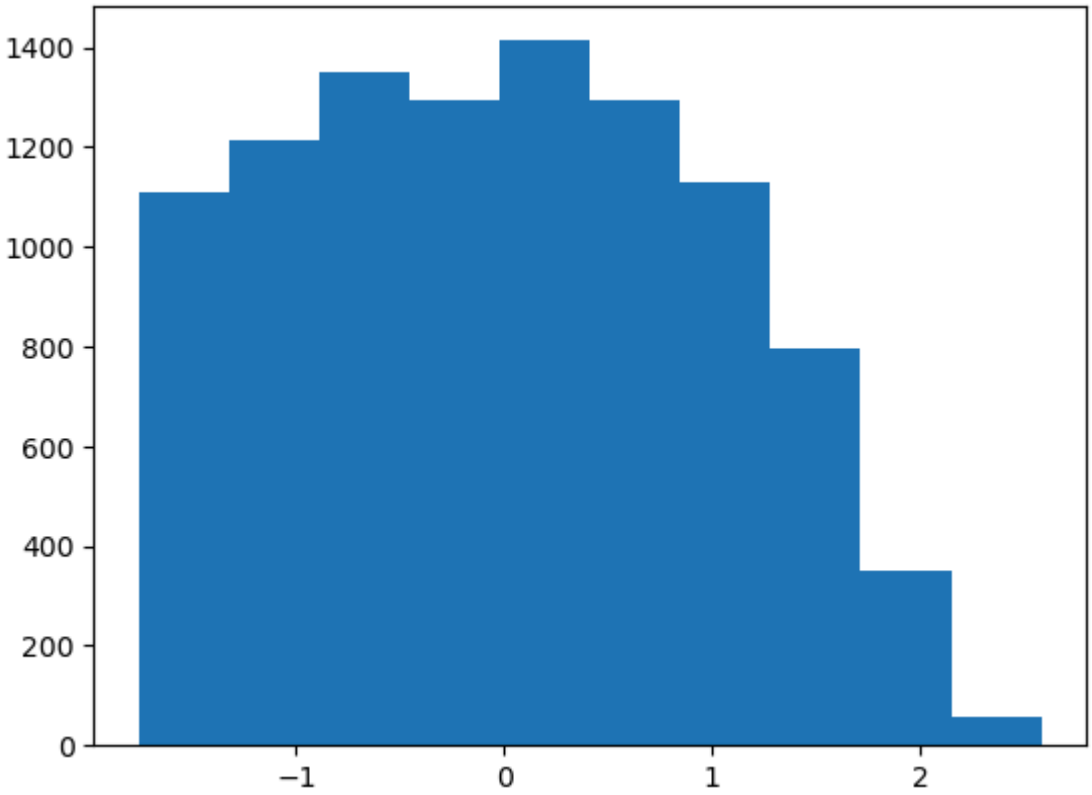

**Power Transformation**

- Box cox
- ye-jhonson
- It is under sklearn preprocessing

In [ ]:
```
import numpy as np
from sklearn.preprocessing import PowerTransformer
# save
# apply fit transform
```

In [27]:
```
from sklearn.preprocessing import PowerTransformer
power = PowerTransformer(method='yeo-johnson')
exp_data1=exp_data.reshape(-1,1)
data_trans = power.fit_transform(exp_data1)
plt.hist(data_trans)
```

Out[27]:
```
(array([1111., 1212., 1348., 1292., 1413., 1294., 1128.,  794.,  351.,
         57.]),
 array([-1.75640778, -1.32230905, -0.88821031, -0.45411157, -0.02001283,
         0.41408591,  0.84818465,  1.28228339,  1.71638213,  2.15048087,
         2.58457961]),
 <BarContainer object of 10 artists>)
```




In [23]:
```
exp_data
```

Out[23]:
```
array([1.85085097, 0.70753286, 0.59755911, ..., 0.23733074, 0.42797443,
       1.52496464])
```

In [25]:
```
exp_data.reshape(-1,1)
```

Out[25]:
```
array([[1.85085097],
       [0.70753286],
       [0.59755911],
       ...,
       [0.23733074],
       [0.42797443],
       [1.52496464]])
```

In [ ]:
```
# Apply box cox share the output
```