

## Data Transformation Techniques

- Generally used for to convert Normal distribution
- Because all statistical math analysis by assumption Data follows Normal distribution
- It is also avoid skew ness also
- We have some important transformation
  - Log transformation
  - Exponential transformation
  - Reciprocal transformation
  - Square root transformation
  - Power transformaton

*Step – 1*

### Read the required packages

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

*Step – 2*

### Read the data

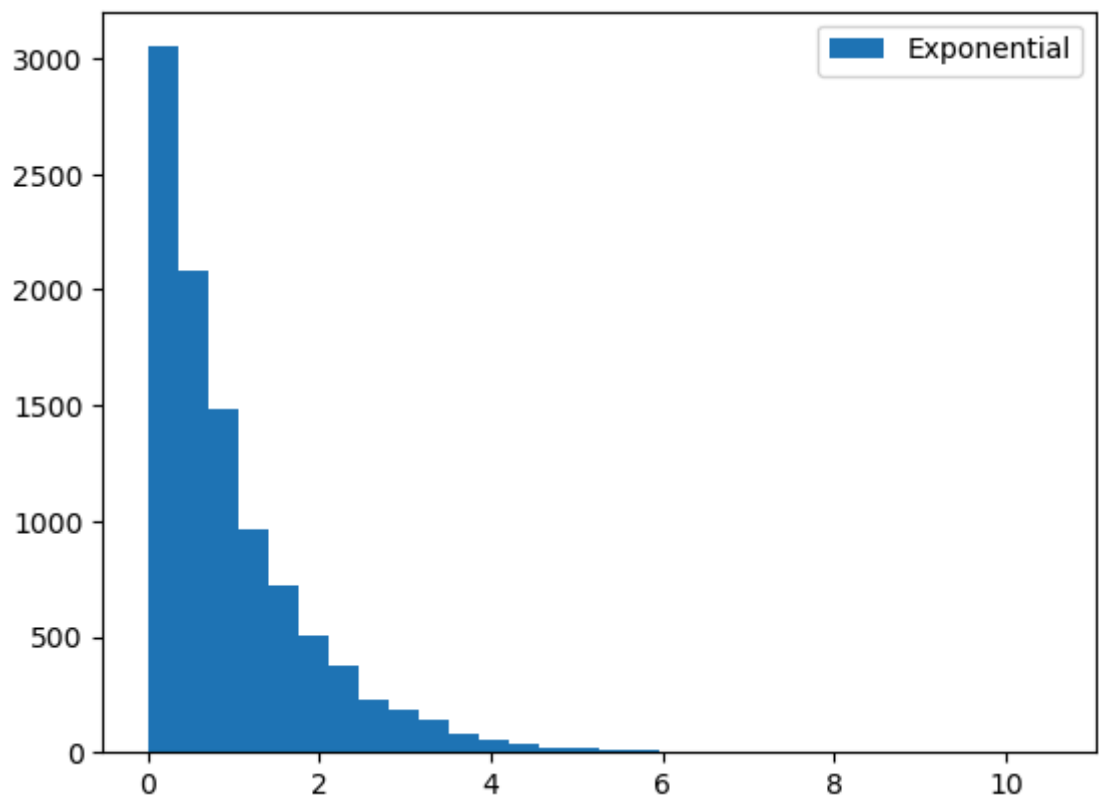
*Exponential – data*

```
In [2]: exp_data=np.random.exponential(size=10000)
exp_data[:10]

# We are consider exponential data set with random 1k observations
# using numpy package
```

```
Out[2]: array([7.73011522e-01, 4.44471406e+00, 1.37768693e+00, 1.41605528e+00,
1.46048573e-01, 5.19345332e-01, 5.48210064e-01, 2.51184417e-04,
7.64626973e-01, 5.34139109e-01])
```

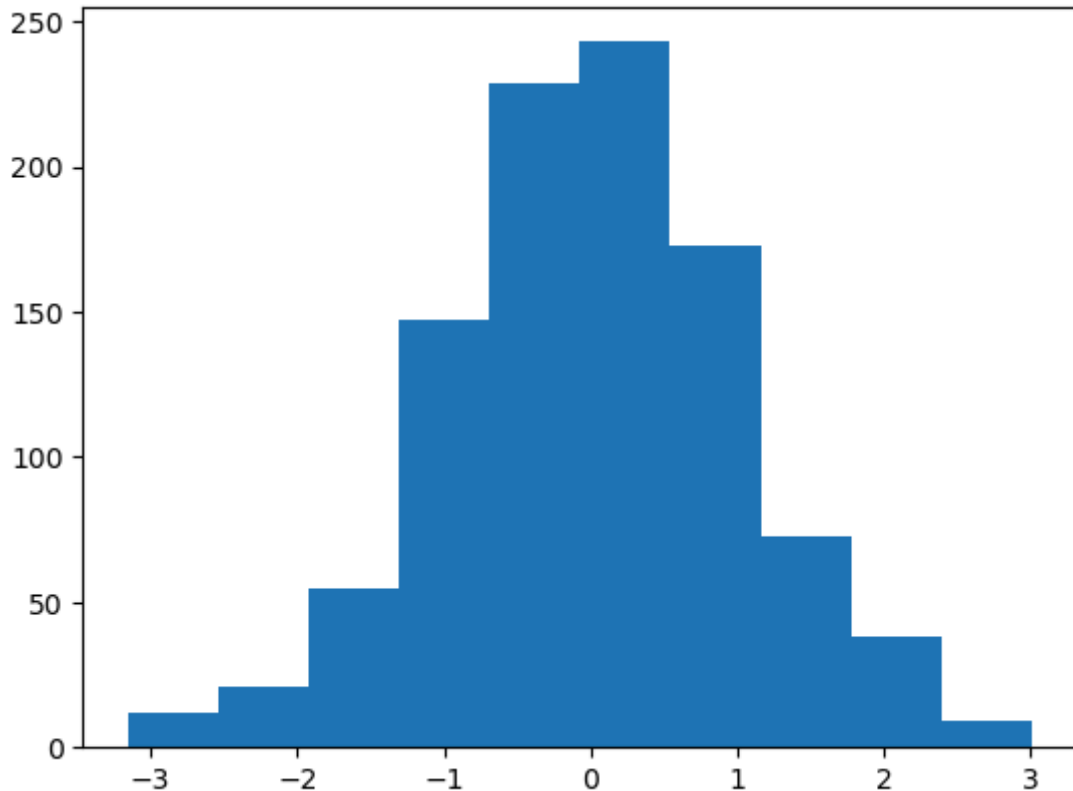
```
In [3]: plt.hist(exp_data,bins=30,label='Exponential')  
plt.legend()  
plt.show()
```



*Norm – data*

```
In [4]: norm_data=np.random.normal(size=1000)
plt.hist(norm_data)
```

```
Out[4]: (array([ 12.,  21.,  55., 147., 229., 243., 173.,  73.,  38.,   9.]),
array([-3.15424458, -2.53824956, -1.92225454, -1.30625952, -0.6902645 ,
        -0.07426948,  0.54172554,  1.15772056,  1.77371558,  2.3897106 ,
         3.00570562])),
<BarContainer object of 10 artists>)
```



*Step – 3*

### Log Transformaton

- np.log is used for log transformation
- Generally log transformation will not convert data into normal
- It avoids skew ness
- np.log means natural logarithm base=e

```
In [5]: x=2
import numpy as np
np.log(2)
```

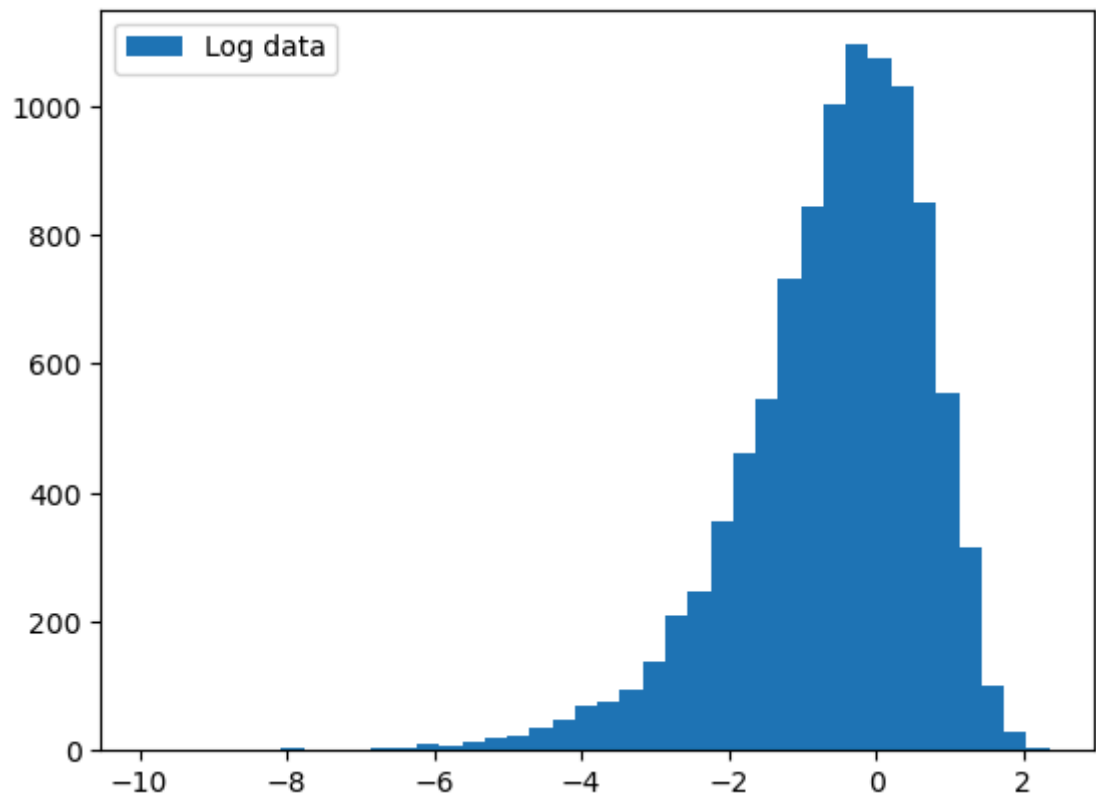
```
Out[5]: 0.6931471805599453
```

```
In [6]: log_data=np.log(exp_data)
log_data[:10]
```

```
Out[6]: array([-0.25746132,  1.49171554,  0.32040596,  0.34787504, -1.92381602,
        -0.65518624, -0.60109674, -8.28932316, -0.26836718, -0.62709897])
```

```
In [ ]: exp_data[:10]
```

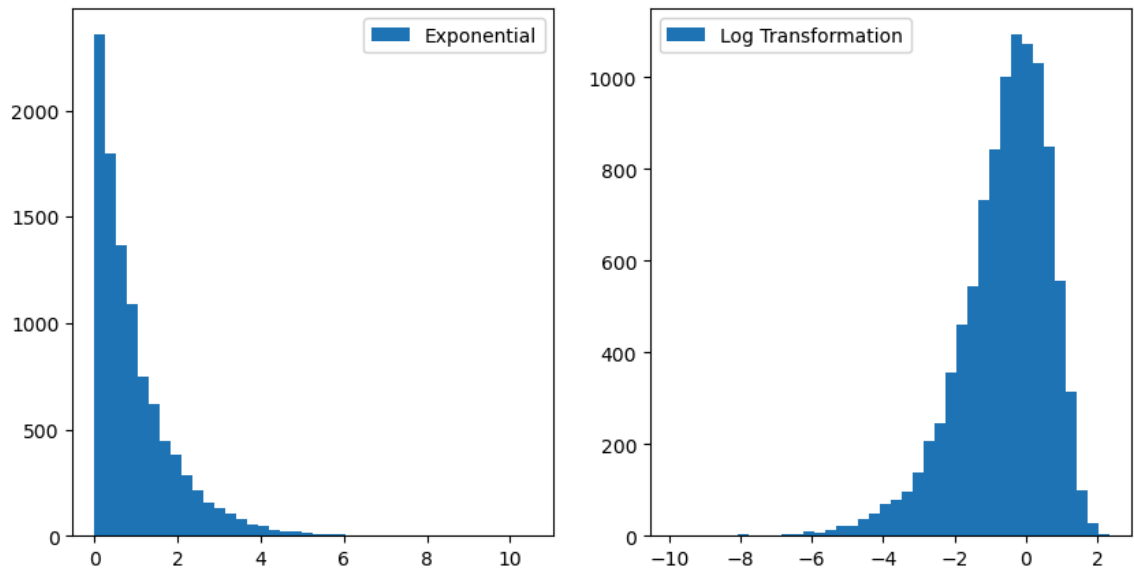
```
In [7]: plt.hist(log_data,bins=40,label='Log data')  
plt.legend()  
plt.show()
```



```
In [8]: plt.figure(figsize=(10,5))
plt.subplot(1,2,1).hist(exp_data,
                        bins=40,
                        label='Exponential')

plt.legend()
plt.subplot(1,2,2).hist(log_data,
                        bins=40,
                        label='Log Transformation')

plt.legend()
plt.show()
```



Step – 4

### Reciprocal transformation

- Reciprocal transformation fails when data has zero value

```
In [ ]: # Take the exponential data
# rec_data= 1/exp_data
# plot the histogram and check it
```

```
In [11]: x=3
np.reciprocal(x)
```

Out[11]: 0

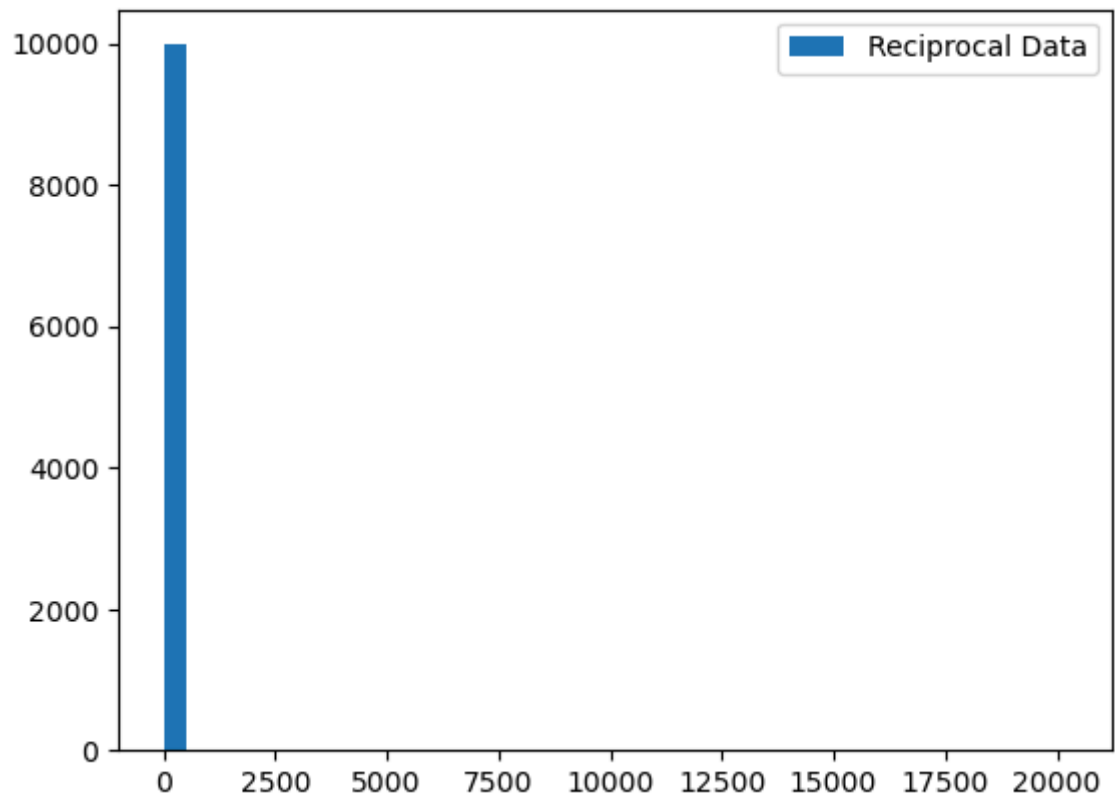
```
In [14]: exp_data,rec_data
```

```
Out[14]: (array([0.77301152, 4.44471406, 1.37768693, ..., 0.12437243, 2.2502534 ,
0.43547471]),
array([1.29364178, 0.22498635, 0.72585431, ..., 8.04036702, 0.44439439,
2.29634461]))
```

```
In [15]: 1/0.77
```

Out[15]: 1.2987012987012987

```
In [12]: rec_data=np.reciprocal(exp_data)
plt.hist(rec_data,bins=40,label='Reciprocal Data')
plt.legend()
plt.show()
```



```
In [ ]: exp_data[:2]
```

```
In [ ]: 1/1.687,1/0.88
```

*Step – 5*

### **Square root transformation**

```
In [16]: print(25**2) # square,
print(25**(1/2)) # square root
print(np.sqrt(25))
```

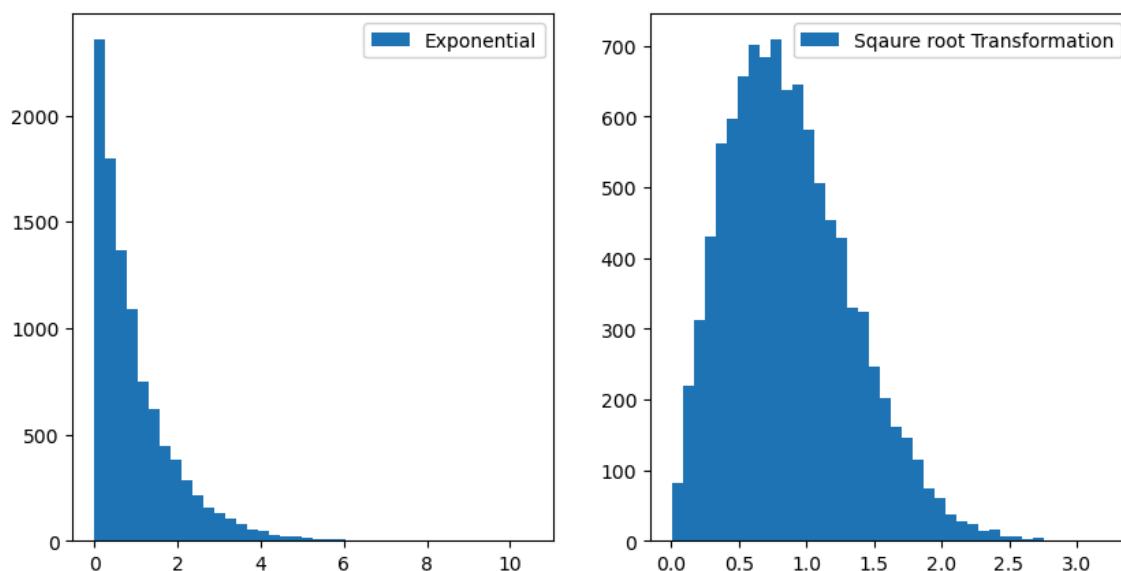
```
625
5.0
5.0
```

```
In [17]: sqrt_data=np.sqrt(exp_data)
```

```
In [18]: plt.figure(figsize=(10,5))
plt.subplot(1,2,1).hist(exp_data,
                        bins=40,
                        label='Exponential')

plt.legend()
plt.subplot(1,2,2).hist(sqrt_data,
                        bins=40,
                        label='Sqaure root Transformation')

plt.legend()
plt.show()
```



```
In [ ]: import seaborn as sns
sns.displot(sqrt_data)
```

*Step – 6*

### Power transformer

- It is related to sklearn package
- Package name: sklearn.preprocessing
- Method name: Power Transformer
- Inside Box-Cox , yeo-jhonson

```
In [21]: exp_data
```

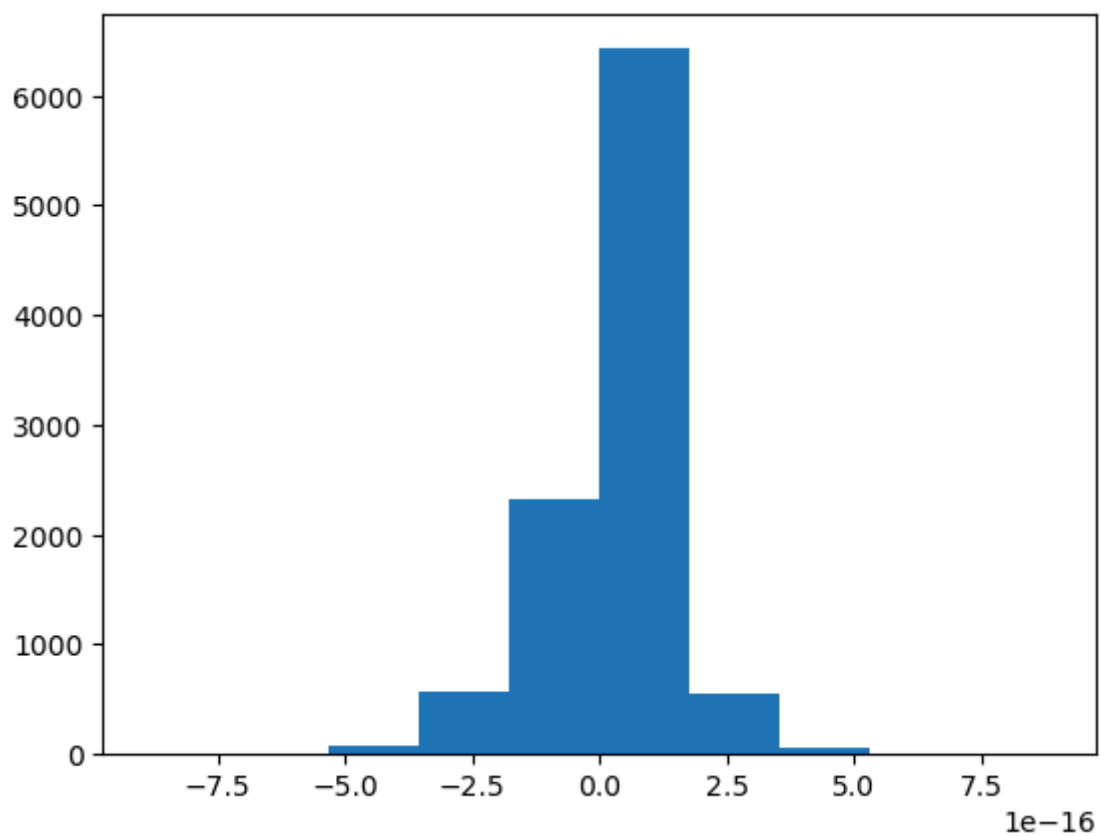
```
Out[21]: array([3.57645714, 0.45701262, 0.36380708, ..., 1.27247759, 0.4684463 ,
                2.48816533])
```

```
In [19]: from sklearn.preprocessing import PowerTransformer
pt=PowerTransformer()
pt_data=pt.fit([exp_data]).transform([exp_data])
```

```
In [27]: pt_data
```

```
Out[27]: array([[ -4.44089210e-16,  5.55111512e-17,  5.55111512e-17, ...,
                  0.00000000e+00,  0.00000000e+00,  0.00000000e+00]])
```

```
In [20]: plt.hist(pt_data[0])  
plt.show()
```



```
In [ ]:
```

```
In [ ]:
```