- Converting skewd distribution to normal distribution
- All maths developed by assumption as data follows normal diribution
- Methods
    - log transformation
    - sqrt transformation
    - Reciprocal transformation
    - exponential transformation
    - box-cox transformation
    - yeo-jhanosn transformation

Read the packages

$step - 1$

In [1]:

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

$step - 2$

Read the Data

In [4]:

```
data=np.random.exponential(size=1000)
# we are taking a random values from exponential distribution
# 1000 samples we are taking
```

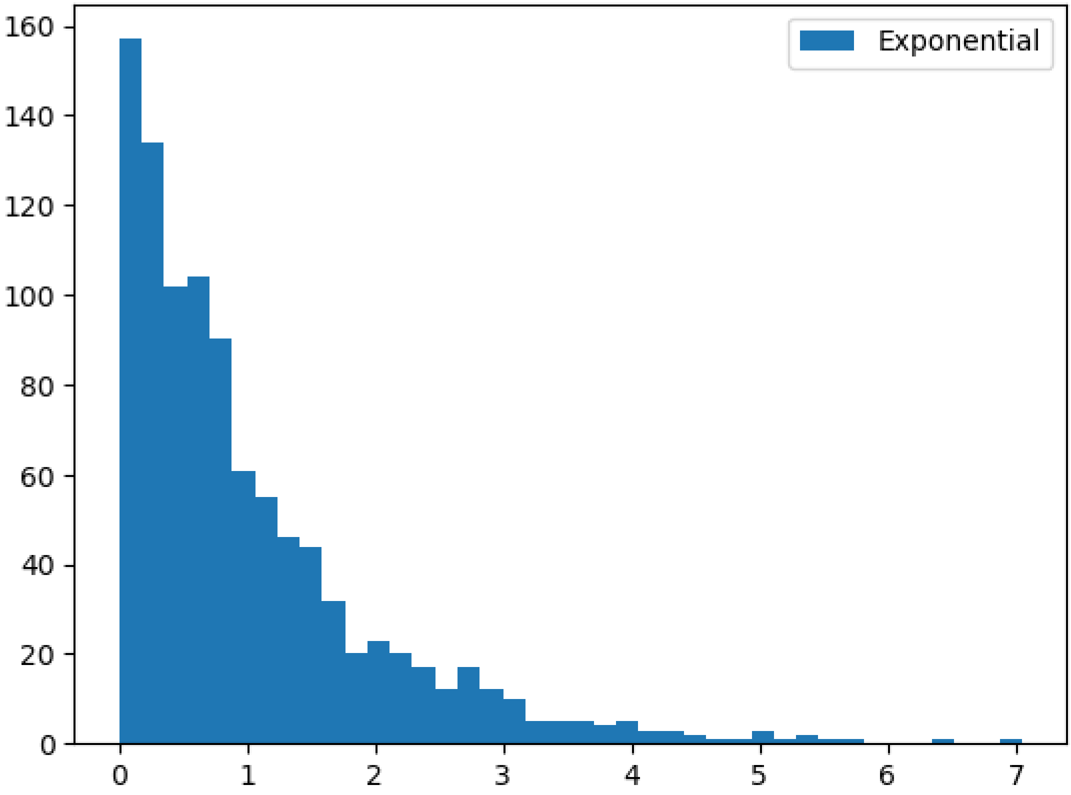
In [5]:

```
data[:10]
```

Out[5]:

```
array([1.71644918, 0.44852568, 0.68748503, 0.31956718, 1.80053973,
       2.52660219, 2.31418607, 0.96207246, 0.11398711, 0.71066586])
```

In [18]:
```
plt.hist(data,bins=40,label='Exponential')
plt.legend()
plt.show()
```

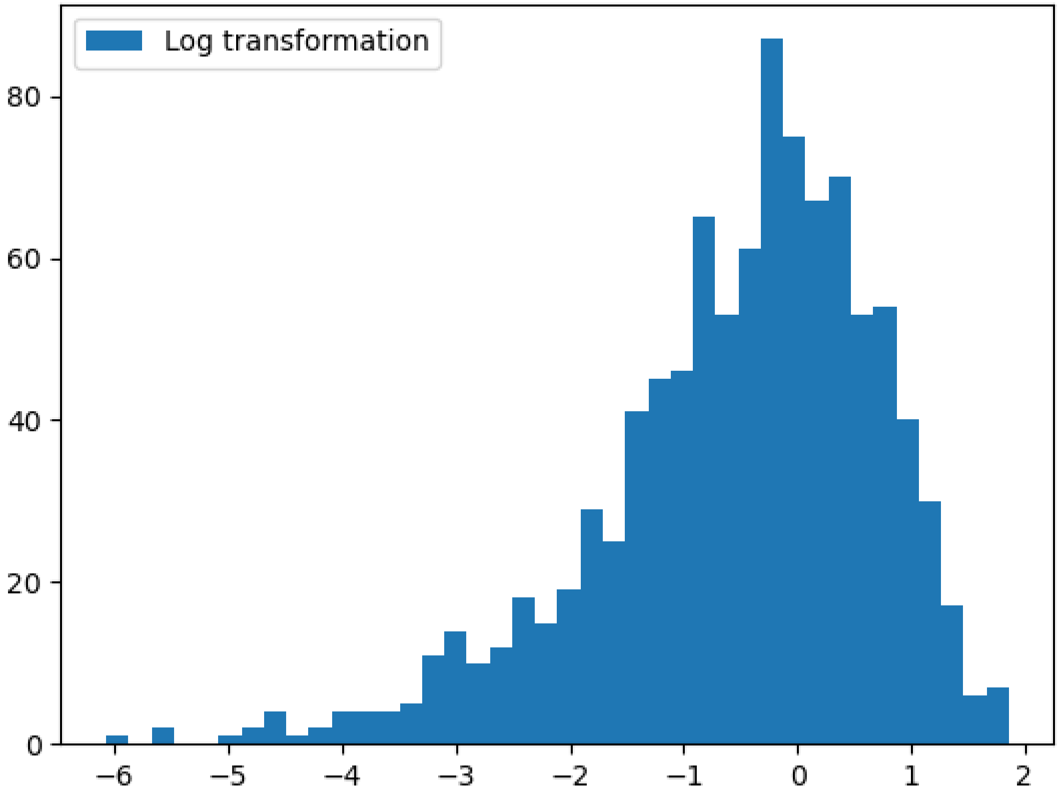


$step - 3$

Log transformation

- np.log represents natural logorithm
- natural logorithm means base e
- exponential will multiply with log base e
- Natural logorithms will works postive data
- log transformation will remove the skew
- It will not convert into Normal distribution

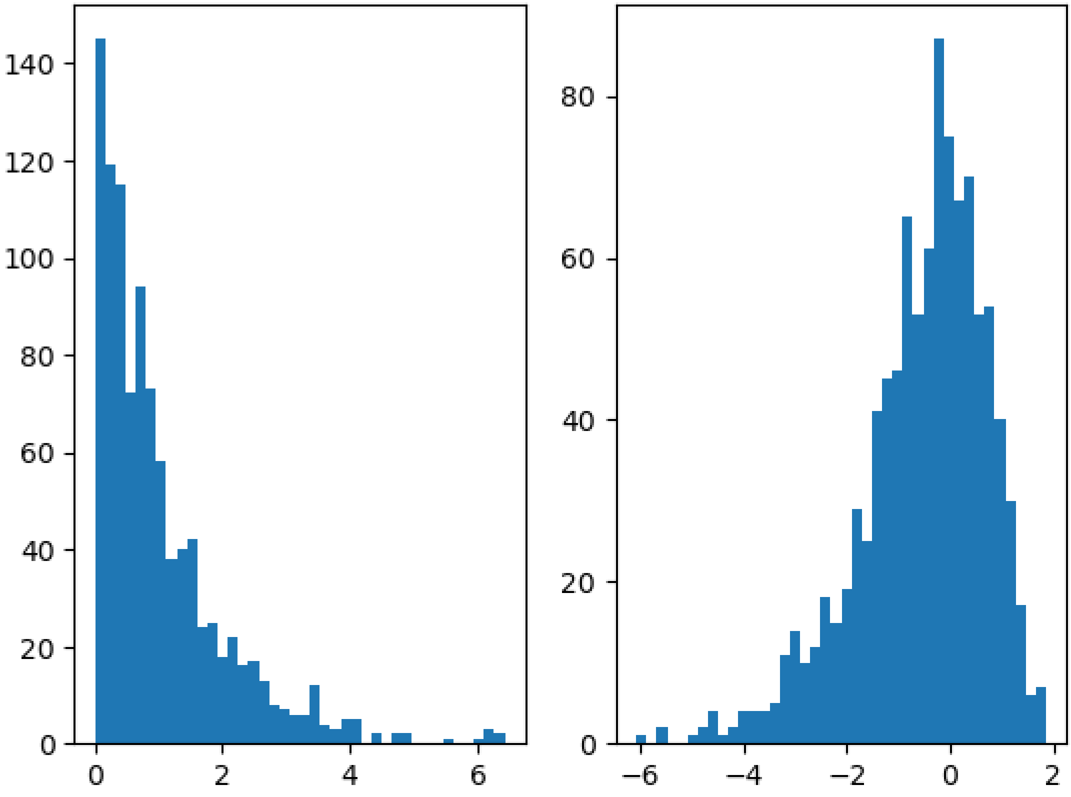
In [6]:
```
log_data=np.log(data)
plt.hist(log_data,bins=40,label='Log transformation')
plt.legend()
plt.show()
```



In [7]:
```
log_data
```

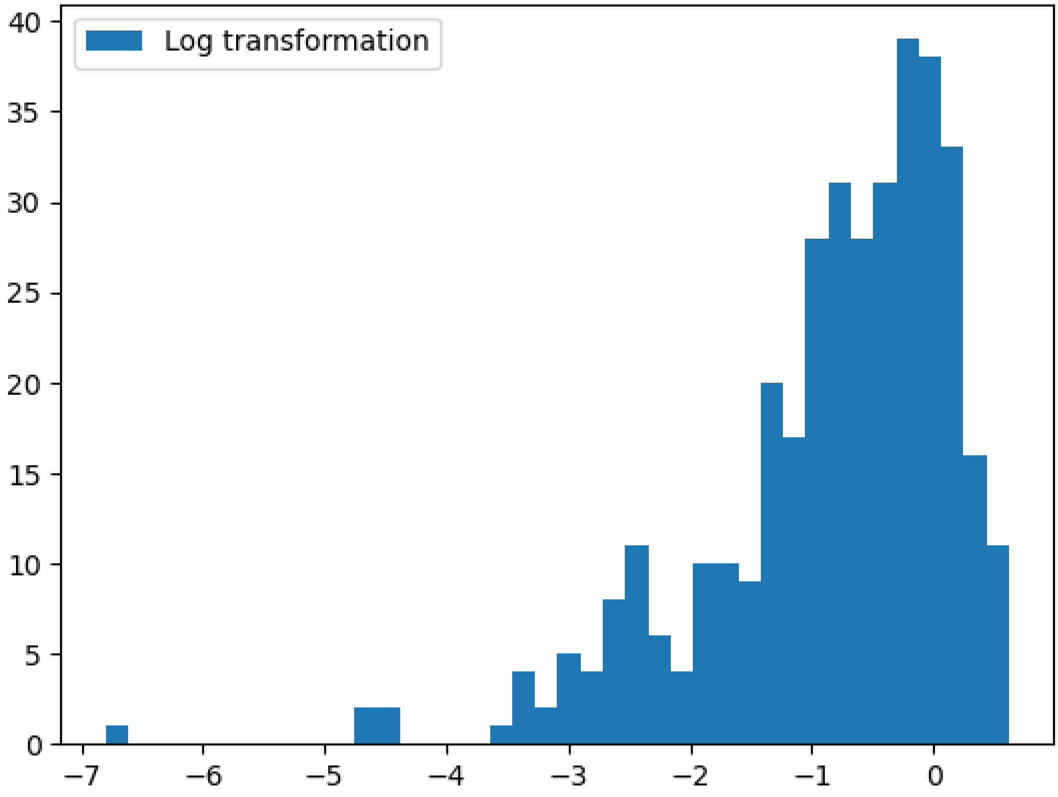
Out[7]:
```
array([ 5.40257727e-01, -8.01789340e-01, -3.74715230e-01, -1.14078776e+
00,
        5.88086470e-01,  9.26875393e-01,  8.39058034e-01, -3.86655041e-
02,
       -2.17166992e+00, -3.41552915e-01, -1.59668971e+00, -7.16723412e-
02,
       -8.21478505e-03, -1.12521247e+00,  9.51118412e-02, -1.90170417e+
00,
       -4.67874507e+00, -1.34493752e+00, -1.17146060e+00,  3.61163679e-
01,
        2.58636119e-01, -2.55245004e-01, -1.18913422e-01,  3.05619208e-
01,
        4.53148352e-01, -1.48973804e-01,  9.47808636e-02, -6.39814397e-
02,
        1.02864183e-01, -8.75161398e-01,  1.87926274e-01, -2.77388723e-
01,
        6.11053718e-01, -3.31758230e+00,  2.46962398e-01, -2.36437953e-
01,
       -2.67048776e+00, -7.19599879e-02,  1.48266572e-01, -4.95805583e-
01
```

In [8]:
```
plt.subplot(1,2,1).hist(data,bins=40)
plt.subplot(1,2,2).hist(log_data,bins=40)
plt.show()
```



In [9]:
```
log_log_data=np.log(log_data)
plt.hist(log_log_data,bins=40,label='Log transformation')
plt.legend()
plt.show()
```

```
C:\Users\kurre\AppData\Local\Temp\ipykernel_5720\4251579950.py:1: RuntimeW
arning: invalid value encountered in log
  log_log_data=np.log(log_data)
```



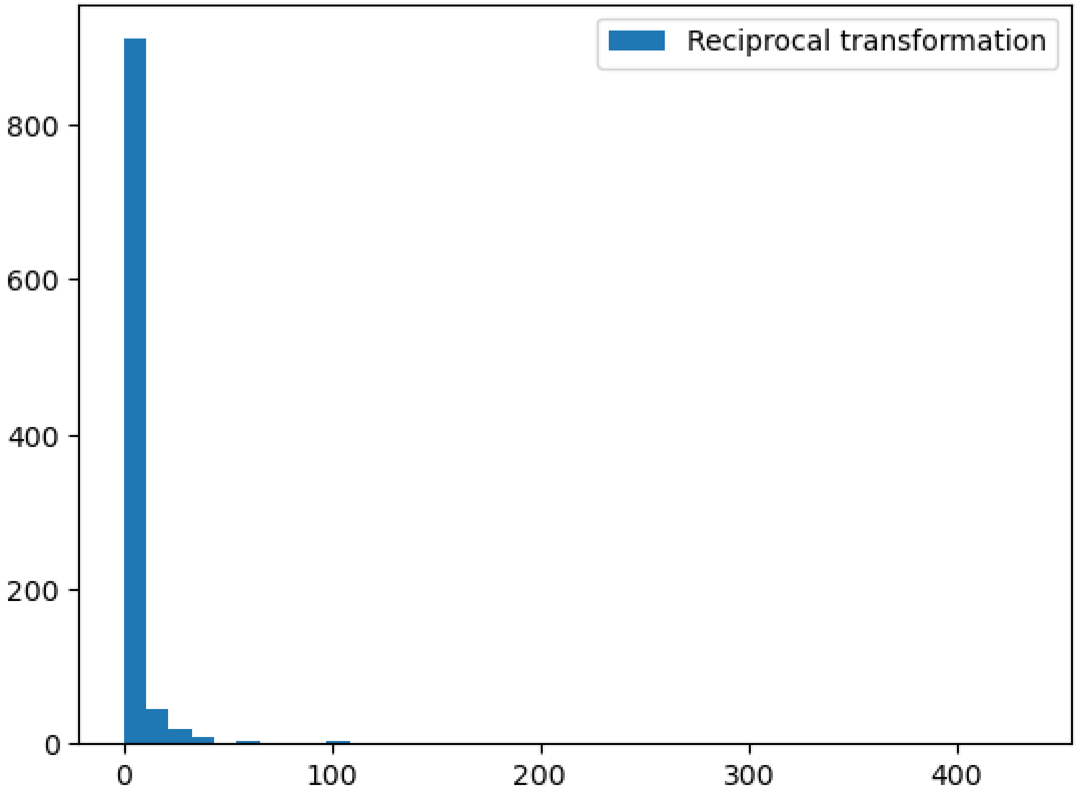
$Step-4$

- if data has zero,it will fail

In [10]:
```
print(data[:5])
print(1/data[:5])
```

```
[1.71644918 0.44852568 0.68748503 0.31956718 1.80053973]
[0.58259808 2.22952674 1.45457714 3.12923248 0.55538902]
```

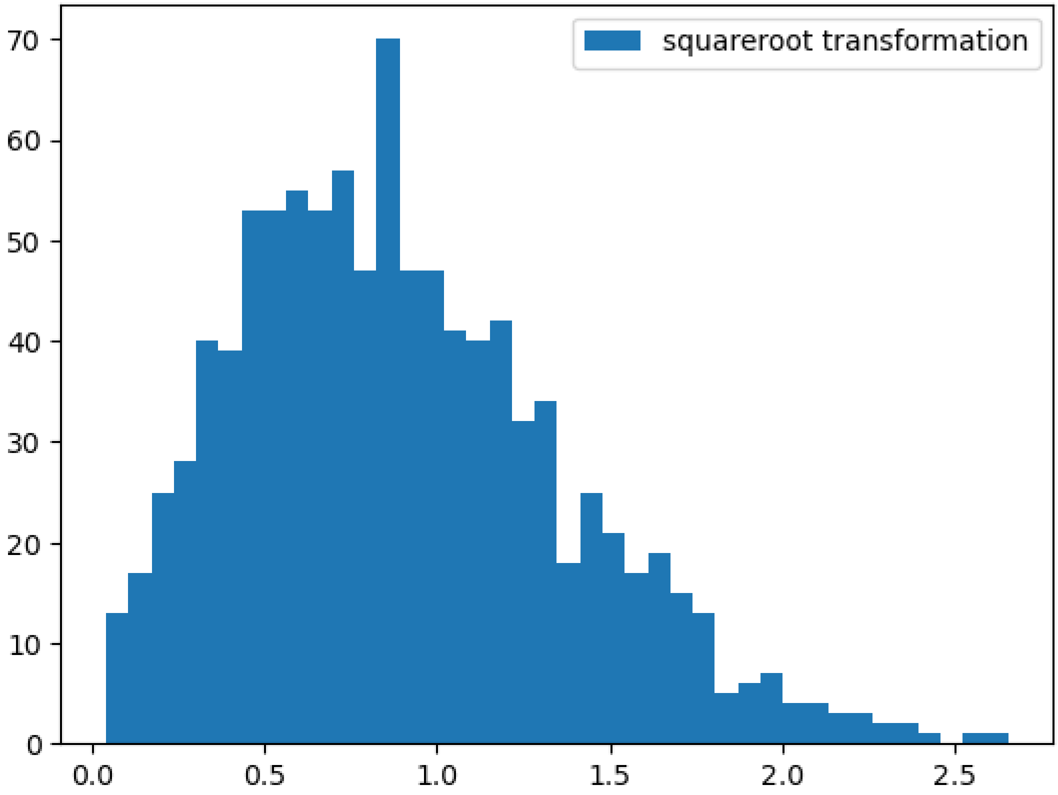
In [12]:
```
rec_data=1/data
plt.hist(rec_data,bins=40,label='Reciprocal transformation')
plt.legend()
plt.show()
```



$step - 5$

Square root transformation

In [26]:
```
srt_data=np.sqrt(data)
plt.hist(srt_data,bins=40,label='squareroot transformation')
plt.legend()
plt.show()
```



Power Transformer

it is in sklearn.preprocessing

method argument

box-cox

ye-jhonson

In [ ]:

In [32]:

```
# Qustion- distance between two point

#(1,2) (6,7)

x1=1
x2=2
y1=6
y2=7
sub=x2-x1
sub1=y2-y1
s1=sub*sub
s2=sub1*sub1
c=s1+s2
p=np.sqrt(c)
print(p)
```

```
2
1.4142135623730951
```

In [35]:

```
import math

# Coordinates of the two points
x1, y1 = 1, 2
x2, y2 = 6, 7

# Calculate the distance
distance = math.sqrt((x2 - x1)**2 + (y2 - y1)**2)

print(f"The distance between the points ({x1},{y1}) and ({x2},{y2}) is: {di
```

```
The distance between the points (1,2) and (6,7) is: 7.0710678118654755
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: