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([3.57645714, 0.45701262, 0.36380708, 0.65021855, 0.14204027,
       1.22165698, 0.33590313, 0.0232194 , 1.89293601, 0.19725694])
```

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

$Norm - data$

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

$Step - 3$

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 logorithm base=e

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

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

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

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

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

In [ ]:
```
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
```

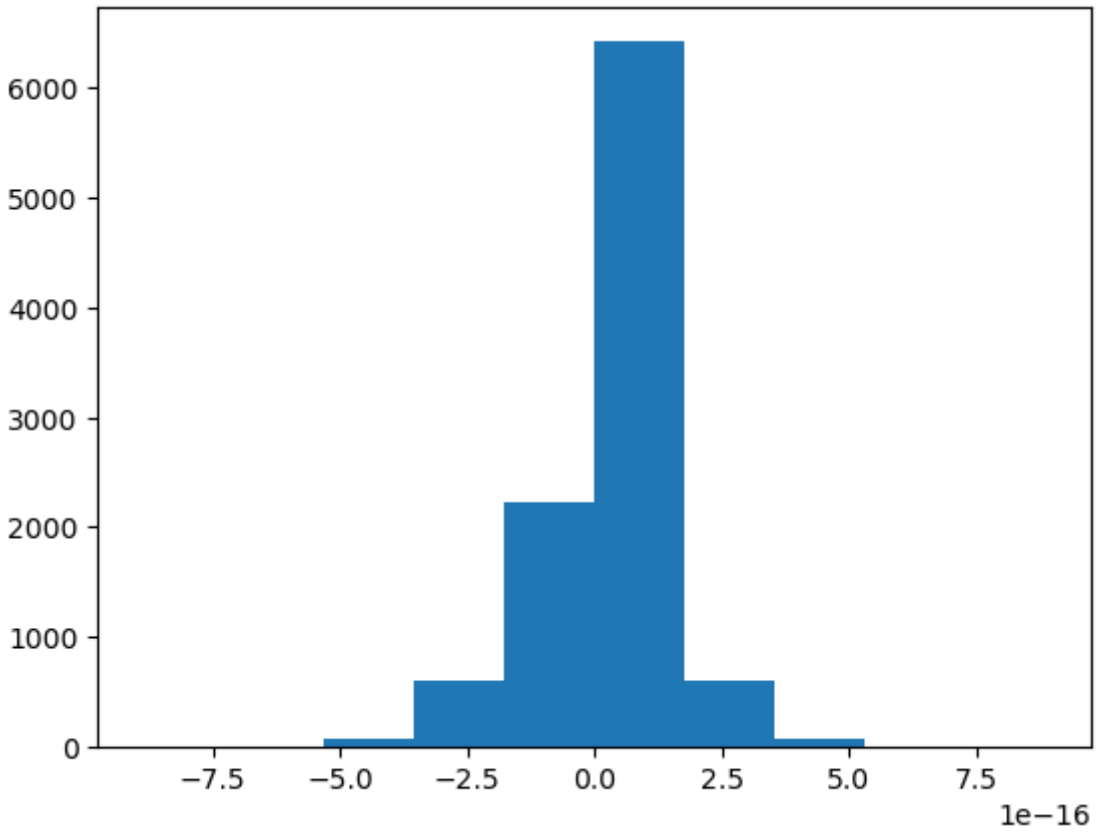
Out[21]:
```
array([3.57645714, 0.45701262, 0.36380708, ..., 1.27247759, 0.4684463 ,
       2.48816533])
```

In [26]:
```
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 [28]:
```
plt.hist(pt_data[0])
plt.show()
```



In [ ]:

In [ ]: