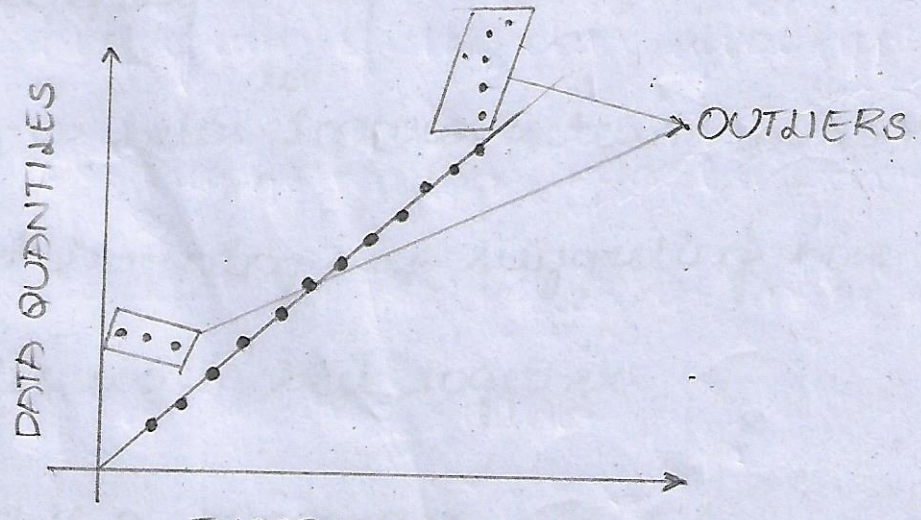
LECTURE - 32

Q-Q PLOT (QUANTILE - QUANTILE):

A Q-Q plot is a plot of the quantiles of two distributions against each other, or a plot based on estimates of the quantiles.

→ The pattern of points in the plot is used to compare the two distributions.

→ A 45° reference line is also plotted. If the two sets come with the same distribution, the points should fall approximately along this reference line.



Q - Q PLOT FIGURE

If you are given a random variable and the plot of PDF looks like bell-shaped curve, then is that distribution is a Normal Distribution or not?

Ans: We look into the Q-Q plot and also on the given random variable, the points falls on line which is at 45°

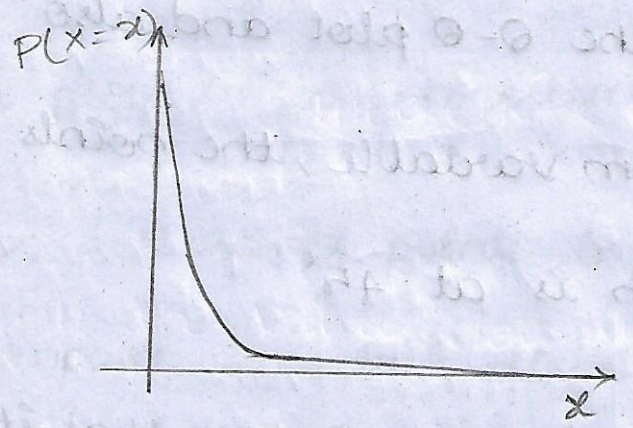
From the above plot, we can say that it is a normal distribution.

→ the Outliers states that they don't follow Normal Distribution.

THEORETICAL QUANTILES - Standard Normal Distribution of a random variable.

PARETO DISTRIBUTION:

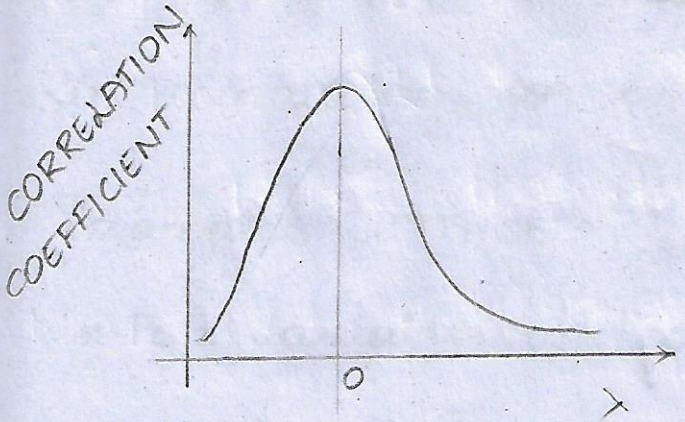
It is a skewed distribution with heavy or slowly decaying tails i.e., much of the data is in the tails.



BOX-COX PLOT:

The Box-Cox normality plot is a plot of these correlation coefficients for various values of the lambda parameter.

The histogram of the data after applying the Box-Cox transformation with lambda = -0.3 shows a data set for which the normality assumption is reasonable.



In Machine learning, we use Q-Q PLOT & BOX-COX PLOT to transform to a Normal Distribution.

HOW TO PLOT?

1. Sort all the values of x which is a Standard Normal Distribution.

$\Rightarrow X \sim N(0, 1)$ where $X \sim N$ should hold TRUE

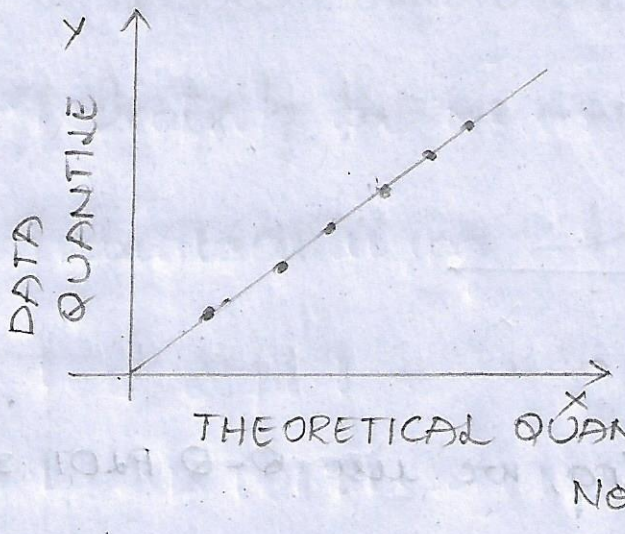
2. Observed Quantity of a given distribution

- GIVEN RANDOM VARIABLE

↳ Sort all the values of Y

$Y \sim$ DISTRIBUTION (Does 'Y' follows Normal distribution or not)

3. As per the data, Q-Q plot is plotted.



4. If the values/dots lie on the reference line i.e., on 45° line, then Y is said to be followed a Normal Distribution.

SNAKE CASE:

It is stylized as snake_case refers to the style of writing in which each space is replaced by an underscore (_) character, and the first letter of each word written in lowercase.

→ It is a commonly used naming convention in computing, for e

Example: PYTHON — mu_normal_100
sigma_normal_100

* For variables and subroutine names and for filenames.

CAMELCASE:

Java uses camelcase as a practice for writing names of methods, variables, classes, packages and constants.

↳ Camel case in Java Programming:

consists of compound words or phrases such that each word or abbrevation begins with a capital letter or first word with a lowercase letter, rest all with capital.

HOW TO CHECK THE NORMALITY FOR A DISTRIBUTI-
-ON?

Ans:→ We check the Normality for a distribution using

↳ Q-Q plot

↳ AD (ANDERSON DARLING) Test

↳ KS (KOLMOGOROV SMIRNOV) Test.

→ Then we transform into a Normal Distribution by using BOX-COX

BOX-COX(pareto_rv) $\xrightarrow[l=0.1]{}$ x_t then

$\lambda = 0 \rightarrow \log_e(\text{pareto\_rv})$ and

$\lambda \neq 0 \rightarrow \frac{(\text{Pareto\_rv})^{\lambda} - 1}{\lambda}$ (takes one value at a time)

I/p - import warnings
warnings.filterwarnings('ignore')

I/p - import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats.

PLOTTING UNIFORM & NORMAL DISTRIBUTION:

I/p - uniform_100 = np.random.uniform(low=10,
high=40, size=100)
sns.distplot(uniform_100)

I/p - uniform_1000 = np.random.uniform(low=10,
high=40, size=1000)
sns.distplot(uniform_1000)

I/p - normal_100 = np.random.normal(loc=20,
scale=5, size=100)
sns.distplot(normal_100)
# mean = loc, Standard deviation = scale

I/p - normal_1000 = np.random.normal(loc=20,
scale=5, size=1000)

sns.distplot(normal_1000)

I/p - normal_10000 = np.random.normal(loc=20,
scale=5, size=10000)

sns.distplot(normal_10000)

→ As the size value increases, there is a change in graph which results a perfect bell-shaped curve i.e., a Normal Distribution.

I/p - mu_normal_10000 = normal_10000.mean()
sigma_normal_10000 = normal_10000.std()
Print(mu_normal_10000, sigma_normal_10000)

O/p - 20.015628 , 5.016727

```
i/p - one_std_right = mu_normal_10000+
                      (1 * sigma_normal_10000)

one_std_left = mu_normal_10000-
               (1* sigma_normal_10000)

two_std_right= mu_normal_10000+
               (2* sigma_normal_10000)

two_std_left = mu_normal_10000 -
               (2 * sigma_normal_10000)

three_std_right = mu_normal_10000 +
                  (3* sigma_normal_10000)

three_std_left = mu_normal_10000 -
                 (3* sigma_normal_10000)


i/p - plt.figure(figsize=(20,10))
      sns.set_style("darkgrid")
      sns.distplot(normal_10000)
      plt.axvline(mu_normal_10000, color='coral',
                  label='Mean')
      Plt.axvline(one_std_right, color='yellow',
                  Label='Mean + 1SD')
```

```
Plt.axvline(one_std_left, color = 'Yellow',
                    label = 'Mean-1SD')

Plt.axvline(two_std_right, color = 'green',
                    label='Mean+2SD')

Plt.axvline(two_std_left, color = 'green',
                    label = 'Mean-2SD')

Plt.axvline(three_std_right, color = 'blue',
                    label = 'Mean+3SD')

Plt.axvline(three_std_left, color = 'blue',
                    label = 'Mean-3SD')

Plt.legend();
```

65-95-99.7 RULE:

```
I/p - type(normal_10000)

O/p - numpy.ndarray

I/p - normal_10000 < one_std_right

O/p - array([True, True, ....., True, True])
```

I/p - (normal_10000< one_std_right).sum()

O/p - 8442

I/p - ((one_std_left < normal_10000) &
(normal_10000 < one_std_right)).sum()

O/p - 6856

I/p - ((one_std_left<normal_10000) &
(normal_10000< one_std_right)).sum()/
normal_10000.size

O/p - 0.6856.

I/p - ((two_std_left<normal_10000)&(normal_10000<
two_std_right)).sum()/normal_10000.size

O/p - 0.9528

I/p - ((three_std_left<normal_10000)&(normal_10000<
three_std_right)).sum()/normal_10000.size

O/p - 0.9975

SAVING RANDOM GENERATED VALUES IN .CSV:

```
I/p - df_100 = pd.DataFrame({'uniform': uniform_
      100, 'normal': normal_100})

      df_1000 = pd.DataFrame({'uniform': uniform_
      1000, 'normal': normal_1000})

      df_10000 = pd.DataFrame({'uniform': uniform_
      10000, 'normal': normal_10000})

I/p - df_100.to_csv('E:\data_100.csv', index='False')

      df_1000.to_csv('E:\data_1000.csv', index='False')

      df_10000.to_csv('E:\data_10000.csv', index='False')
```

CHECKING SKEWNESS & KURTOSIS:

```
I/p - df_100['normal'].skew()

O/p - 0.287

I/p - df_100['normal'].kurt()

O/p - 0.476
```

I/p - print('100 Data points', df_100['normal'].skew())

Print('1000 Data points', df_1000['normal'].skew())

Print('10000 Data points, df_10000['normal'].skew())

O/p - 100 Data points 0.287

1000 Data points -0.014

10000 Data points -0.010

I/p - Print('100 Data points', df_100['normal'].kurt())

Print('1000 Data Points', df_1000['normal'].kurt())

Print('10000Data points', df_10000['normal'].kurt())

O/p - 100 Data points 0.476

1000 Data points 0.225

10000 Data points 0.058

CHECKING NORMALITY WITH Q-Q PLOT :

```
I/p - stats.probplot(df_100['uniform'],
                     dist="norm", plot=plt)
      Plt.grid()

I/p - stats.probplot(df_100['normal'], dist="norm",
                     Plot=plt)
      Plt.grid()

I/p - plt.figure(figsize=(12,8))
      # Creating a plot with 1 row and 2 cols
      Plt.subplot(2, 3, 1)
      stats.probplot(df_100['uniform'], dist="norm",
                     Plot=plt)
      Plt.subplot(2,3,2)
      stats.probplot(df_1000['uniform'], dist="norm",
                     Plot=plt)
      Plt.subplot(2,3,3)
      stats.probplot(df_10000['uniform'], dist="norm",
                     Plot=plt)
```

```
plt.subplot(2, 3, 4)
stats.probplot(df_100['normal'], dist="norm",
               plot=plt)

plt.subplot(2, 3, 5)
stats.probplot(df_1000['normal'], dist="norm",
               plot=plt)

plt.subplot(2,3,6)
stats.probplot(df_10000['normal'], dist="norm",
               plot=plt)

plt.show()
```

BOX-COX TRANSFORMATION:

```
I/p - pareto_cv = np.random.pareto(a=2,
                                   size=1000)

      sns.distplot(pareto_cv)

# a = alpha
# Normality Test
I/p - stats.probplot(pareto_cv, dist="norm",
                     plot=plt)

      plt.grid()
```

I/P - # X_t = Transformed box-cox, l = lambda

```
x_t, l = stats.boxcox(pareto_rv)

print(l)
```

O/P - 0.1038

I/P - `stats.probplot(x_t, dist="norm", plot=plt)`

```
plt.grid()
```