Descriptive Statistics

Descriptive statistics involves summarizing and organizing the data so they can be easily understood

Descriptive statistics, unlike inferential statistics, seeks to describe the data, but do not attempt to ma whole population. Here, we typically describe the data in a sample. This generally means that descrip is not developed on the basis of probability theory.

```
# Importing Libraries
import math

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import scipy.stats as stats

import warnings
warnings.filterwarnings('ignore')
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning:
  import pandas.util.testing as tm
```

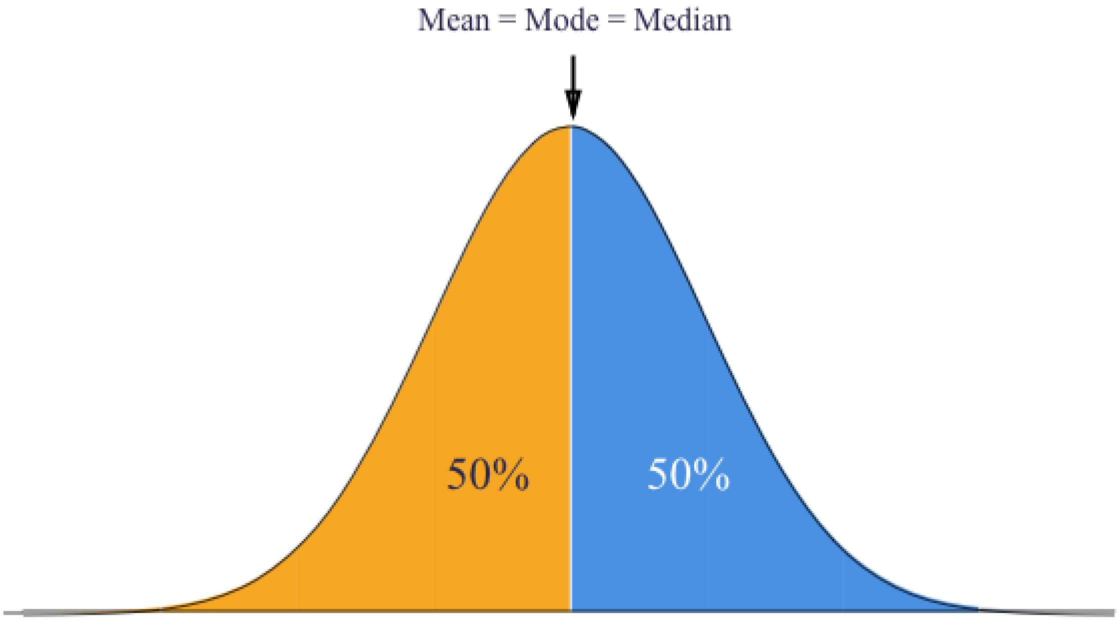
Normal Distribution

The normal distribution is one of the most important concepts in statistics since nearly all statistical basically describes how large samples of data look like when they are plotted. It is sometimes called The bell curve is symmetrical. Half of the data will fall to the left of the mean; half will fall to the right.

Inferential statistics and the calculation of probabilities require that a normal distribution is given. Thi normally distributed, you need to be very careful what statistical tests you apply to it since they could

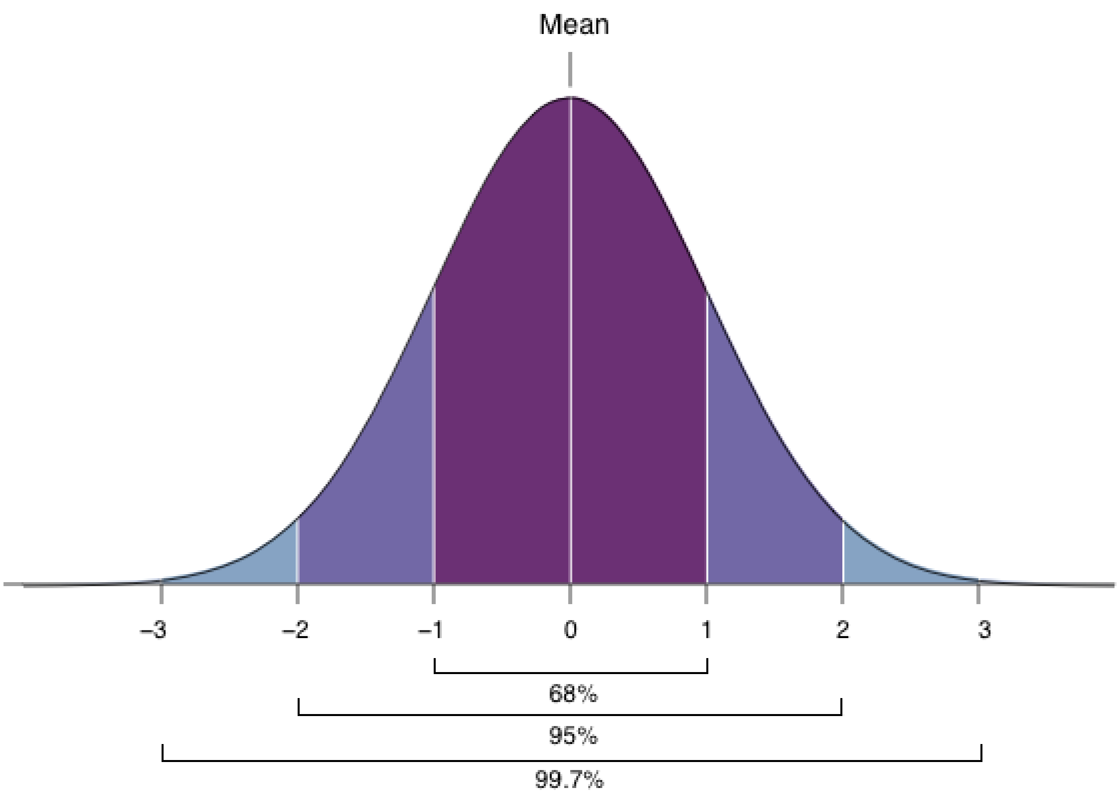
A Normal Distribution is given if your data is symmetrical, bell-shaped, centered and unimodal.

In a perfect normal distribution, each side is an exact mirror of the other. It should look like the distrib

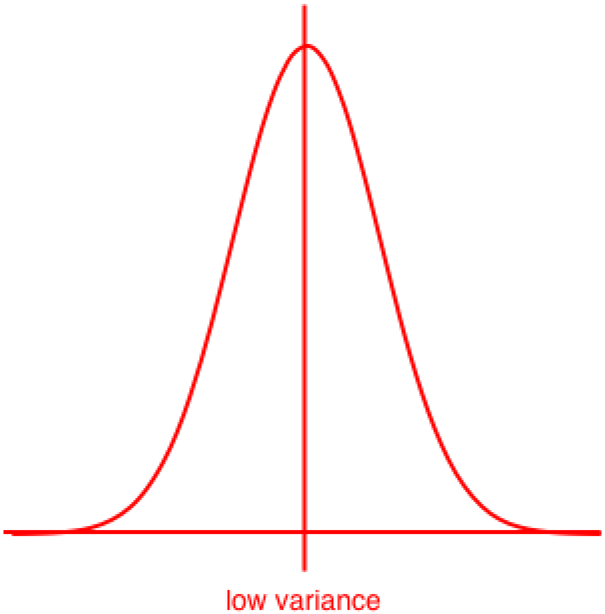


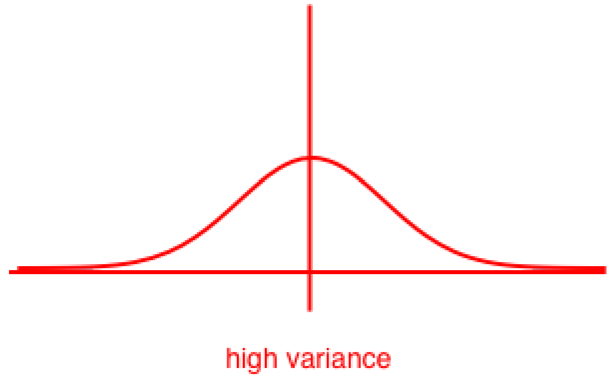
The **Empirical Rule** (also called the 68-95-99 7 Rule or the Three Sigma Rule) tells you what percentag
number of standard deviations from the mean:

- 68% of the data falls within one standard deviation of the mean.
- 95% of the data falls within two standard deviations of the mean.
- 99.7% of the data falls within three standard deviations of the mean.



The standard deviation controls the spread of the distribution. A smaller standard deviation indicates
the mean; the normal distribution will be taller. A larger standard deviation indicates that the data is s
distribution will be flatter and wider.





Properties of a Normal Distribution:

1. The mean, mode and median are all equal.
2. The curve is symmetric at the center (i.e. around the mean, μ).
3. Exactly half of the values are to the left of center and exactly half the values are to the right.
4. The total area under the curve is 1.

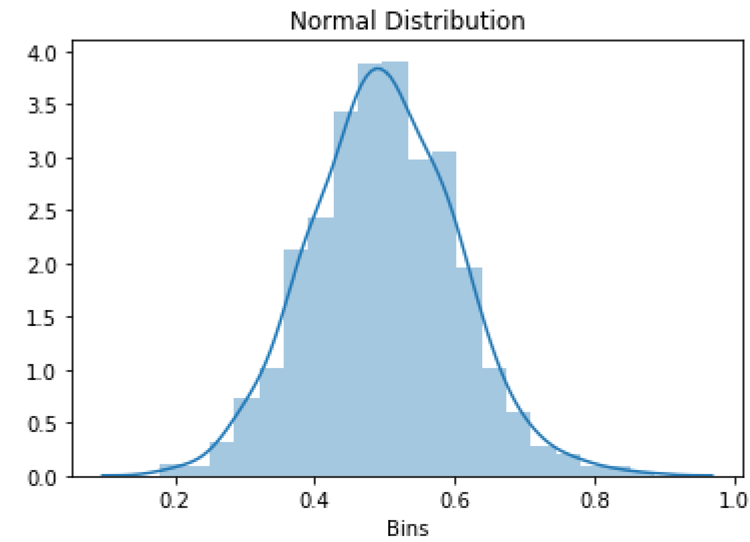
A Standard Normal Disctribution is a normal distribution with a mean of 0 and a standard deviation o

```
# Creating Normal Distribution with mean = 0.5 and sd = 0.1

np.random.seed(100)

mu, sigma = 0.5, 0.1
Distribution = np.random.normal(mu, sigma, 1000)

# Create the bins and histogram
sns.distplot(Distribution, bins= 20, axlabel= 'Bins', label= 'Normal Distribution')
plt.title('Normal Distribution')
plt.show()
```




```
## Reading Dataset - IPL Dataset from Kaggle: https://www.kaggle.com/nowke9/ipldata
```

```
import io

deliveries = pd.read_csv('/content/deliveries.csv')
matches = pd.read_csv('/content/matches.csv')
```

```
print('Number of rows    =', matches.shape[0])
print('Number of columns =', matches.shape[1])
matches.head()
```

```
Number of rows    = 756
Number of columns = 18
```

	id	season	city	date	team1	team2	toss_winner	toss_decision	resu
0	1	2017	Hyderabad	2017-04-05	Sunrisers Hyderabad	Royal Challengers Bangalore	Royal Challengers Bangalore	field	norm
1	2	2017	Pune	2017-04-06	Mumbai Indians	Rising Pune Supergiant	Rising Pune Supergiant	field	norm
2	3	2017	Rajkot	2017-04-07	Gujarat Lions	Kolkata Knight Riders	Kolkata Knight Riders	field	norm
3	4	2017	Indore	2017-04-08	Rising Pune Supergiant	Kings XI Punjab	Kings XI Punjab	field	norm

Measures of Frequency

Frequency statistics include absolute frequencies (raw counts) for each category of the discrete varia
percentages of the total number of observations).etc.

Count

```
# Total non null values
matches.winner.count()
```

```
752
```

```
# Total Value count
len(matches.winner)
```

```
756
```

```
# NA values
matches.winner.isna().sum()
```

```
4
```

```
# Total size of column
matches.winner.size
```

```
756
```

```
# Total unique Values
matches.winner.unique()
```

```
array(['Sunrisers Hyderabad', 'Rising Pune Supergiant',
       'Kolkata Knight Riders', 'Kings XI Punjab',
       'Royal Challengers Bangalore', 'Mumbai Indians',
       'Delhi Daredevils', 'Gujarat Lions', 'Chennai Super Kings',
       'Rajasthan Royals', 'Deccan Chargers', 'Pune Warriors',
       'Kochi Tuskers Kerala', nan, 'Rising Pune Supergiants',
       'Delhi Capitals'], dtype=object)
```

```
# Total unique Values count
matches.winner.nunique()
```

```
15
```

Value Counts

```
# Value Counts
matches.winner.value_counts()
```

```
Mumbai Indians                 109
Chennai Super Kings            100
Kolkata Knight Riders           92
Royal Challengers Bangalore     84
Kings XI Punjab                 82
Rajasthan Royals                75
Delhi Daredevils                67
Sunrisers Hyderabad             58
Deccan Chargers                 29
Gujarat Lions                   13
Pune Warriors                   12
Delhi Capitals                  10
Rising Pune Supergiant          10
Kochi Tuskers Kerala             6
Rising Pune Supergiants          5
Name: winner, dtype: int64
```

```
# Normalized Value Counts
matches.winner.value_counts(normalize= True,sort= True, ascending= False)
```

```
Mumbai Indians                 0.144947
Chennai Super Kings            0.132979
Kolkata Knight Riders          0.122340
Royal Challengers Bangalore    0.111702
Kings XI Punjab                0.109043
Rajasthan Royals               0.099734
Delhi Daredevils               0.089096
Sunrisers Hyderabad            0.077128
Deccan Chargers                0.038564
Gujarat Lions                  0.017287
Pune Warriors                  0.015957
Delhi Capitals                 0.013298
Rising Pune Supergiant         0.013298
Kochi Tuskers Kerala           0.007979
Rising Pune Supergiants        0.006649
Name: winner, dtype: float64
```

CrossTabs

```
# Crosstabbed Value Count
pd.crosstab(matches.season, matches.winner)
```

winner	Chennai Super Kings	Deccan Chargers	Delhi Capitals	Delhi Daredevils	Gujarat Lions	Kings XI Punjab	K
season							
2008	9	2	0	7	0	10	
2009	8	9	0	10	0	7	
2010	9	8	0	7	0	4	
2011	11	6	0	4	0	7	
2012	10	4	0	11	0	8	
2013	12	0	0	3	0	8	
2014	10	0	0	2	0	12	
2015	10	0	0	5	0	3	
2016	0	0	0	7	9	4	
2017	0	0	0	6	4	7	
2018	11	0	0	5	0	6	
2019	10	0	10	0	0	6	

```
results = pd.crosstab(matches.toss_decision, matches.result, margins = True)
results.columns = ['no result','normal','tie', 'Row_sum']
results.index = ['bat','field', 'Col_sum']
results
```

	no result	normal	tie	Row_sum
bat	1	288	4	293
field	3	455	5	463
Col_sum	4	743	9	756

```
results/results.loc["Col_sum"]
```

	no result	normal	tie	Row_sum
bat	0.25	0.387618	0.444444	0.387566
field	0.75	0.612382	0.555556	0.612434
Col_sum	1.00	1.000000	1.000000	1.000000

```
results/results.loc["Col_sum",'Row_sum']
```

	no result	normal	tie	Row_sum
bat	0.001323	0.380952	0.005291	0.387566
field	0.003968	0.601852	0.006614	0.612434
Col_sum	0.005291	0.982804	0.011905	1.000000

Measure of Central Tendency (Mean, Median, Mode)

Central tendency refers to the idea that there is one number that best summarizes the entire set of m
way "central" to the set.

Mean / Average

Mean or Average is a central tendency of the data i.e. a number around which a whole data is spread
can estimate the value of whole data set.

1. Mean (usuallly refered to Arithmetic Mean, also called Average) is calculated as sum of all num
total number of values.
2. Another type of mean is geometric mean. It is calculated as Nth root of product of all the numb
in the dataset.

Arithmetic Mean

$$Arithmetic\,mean = \frac{Sum\,of\,all\,numbers}{No.\;of\,values\,in\,the\,set} \quad or$$

$$\bar{x} = \frac{\sum_{i=i}^{n} x_i}{n}$$

Geometric Mean

$$Geometric\,mean = \sqrt[n]{product\,of\,all\,numbers}$$

$$\bar{x}_{geom} = \sqrt[n]{\prod_{i=1}^{n} x_i}$$

```
win_by_runs_data = matches[matches['win_by_runs'] > 0].win_by_runs
print('Number of rows =',len(win_by_runs_data))
win_by_runs_data.head()
```

```
Number of rows = 337
0     35
4     15
8     97
13    17
14    51
Name: win_by_runs, dtype: int64
```

```
# Calculating arithmetic mean

win_by_runs_arithmetic_mean = sum(win_by_runs_data) / len(win_by_runs_data)

print('Arithmetic mean =' ,win_by_runs_arithmetic_mean)
```

```
Arithmetic mean = 29.798219584569733
```

```
win_by_runs_data.mean()
```

```
29.798219584569733
```

```
# Calculating geometric mean : https://docs.scipy.org/doc/scipy/reference/stats.mstats.h

win_by_runs_geometric_mean = stats.mstats.gmean(win_by_runs_data)

print('Geometric mean =' ,win_by_runs_geometric_mean)
```

```
Geometric mean = 19.24102896835606
```

Median

Median is the value which divides the data in 2 equal parts i.e. number of terms on right side of it is sa when data is arranged in either ascending or descending order.

- Median will be a middle term, if number of terms is odd
- Median will be average of middle 2 terms, if number of terms is even.

```
win_by_runs_10 = list(win_by_runs_data[:10])
print(win_by_runs_10)
print(sorted(win_by_runs_10))
```

```
[35, 15, 97, 17, 51, 27, 5, 21, 15, 14]
[5, 14, 15, 15, 17, 21, 27, 35, 51, 97]
```

```
[5, 14, 15, 15, 17, 21, 27, 35, 51, 97]
              ^^  ^^
          (middle numbers)
```

```
Median = (17 + 21)/2 = 19
```

```
win_by_runs_10_median = win_by_runs_data[:10].median()
print('Median (first 10) =', win_by_runs_10_median)

win_by_runs_median = win_by_runs_data.median()
print('Median =', win_by_runs_median)
```

```
Median (first 10) = 19.0
Median = 22.0
ERROR! Session/line number was not unique in database. History logging moved to new sess
```

Mode

Mode is the term appearing maximum time in data set i.e. term that has highest frequency.

But there could be a data set where there is no mode at all as all values appears same number of time more than the rest of the values then the data set is bimodal. If three values appeared same time and data set is trimodal and for n modes, that data set is multimodal.

```
# Retrieve frequency (sorted, descending order)
win_by_runs_data.value_counts(sort=True, ascending=False).head()
```

```
14    13
10    11
4     11
1     10
13     9
Name: win_by_runs, dtype: int64
```

```
win_by_runs_data_mode = win_by_runs_data.mode()
print('Mode =', list(win_by_runs_data_mode))
```

```
Mode = [14]
```

```
print('Mode =', stats.mstats.mode(win_by_runs_data))
```

```
Mode = ModeResult(mode=array([14.]), count=array([13.]))
```

Measure of Spread / Dispersion

By just measuring the center of the data, one wouldn't get much idea about the dataset. Measure of S variability/spread within your data.

Standard Deviation & Variance

Variance is a square of average distance between each quantity and mean. That is it is square of star

Standard deviation is the measurement of average distance between each quantity and mean. That is standard deviation indicates that the data points tend to be close to the mean of the data set, while a data points are spread out over a wider range of values. There are situations when we have to choose Deviation. When we are asked to find SD of some part of a population, a segment of population; then

$$\text{S.D.} = \sqrt{\frac{1}{n-1}\sum_{i=0}^{n}(x-\bar{x})^{2}}$$

where $\bar{x}$ is mean of a sample. But when we have to deal with a whole population, then we use populat

$$\text{S.D.} = \sqrt{\frac{1}{n}\sum_{i=0}^{n}(x-\mu)^{2}}$$

where μ is mean of a population. Though sample is a part of a population, their SD formulas should h observed values fall (on average) closer to the sample mean instead of the entire population one. Thu underestimates the real one and dividing by (n−1) corrects the result - ***(Bessel's Correction)***

Comparison with IQR: IQR is calculated with respect to median, Standard deviation is calculated with

```
win_by_wickets_data = matches[matches.win_by_wickets > 0].win_by_wickets
win_by_wickets_data.head()
```

```
1     7
2    10
3     6
5     9
6     4
Name: win_by_wickets, dtype: int64
```

```
# Step 1: calculate mean(μ)
win_by_wickets_mean = win_by_wickets_data.mean()
print(f'Mean = {win_by_wickets_mean}')

# Step 2: calculate numerator part - sum of (x - mean)
win_by_wickets_var_numerator = sum([(x - win_by_wickets_mean) ** 2 for x in win_by_wicke

# Step 3: calculate variane
```

```
win_by_wickets_variance = win_by_wickets_var_numerator / len(win_by_wickets_data)
print(f'Variance = {win_by_wickets_variance}')

# Step 4: calculate standard deviation
win_by_wickets_standard_deviation = math.sqrt(win_by_wickets_variance)
print(f'Standard deviation = {win_by_wickets_standard_deviation}')
```

```
Mean = 6.238916256157635
Variance = 3.3246924215583893
Standard deviation = 1.8233739116150558
```

```
win_by_wickets_standard_deviation_verify = win_by_wickets_data.std(ddof = 0) ### ddof =
print(f'Standard deviation = {win_by_wickets_standard_deviation_verify}')
```

```
Standard deviation = 1.8233739116150558
```

i.e. matches are won by an average of 6.23 wickets with standard deviation of 1.83 (spread = 6.23 ± 1

What is a Z-Score?

Simply put, a z-score (also called a standard score) gives you an idea of how far from the mean a data measure of how many standard deviations below or above the population mean a raw score is.

A z-score can be placed on a normal distribution curve. Z-scores range from -3 standard deviations (w normal distribution curve) up to +3 standard deviations (which would fall to the far right of the normal score, you need to know the mean μ and also the population standard deviation σ.

The basic z score formula for a sample is:

```
z = (x – μ) / σ
```

```
# Z-Score for a match won by 10 Wickets
print('Z-score = ',(10-win_by_wickets_data.mean())/win_by_wickets_data.std())
```

```
Z-score =  2.0601638466597523
```

so a match won by 10 wickets falls 2.06 std away from the mean.

Mean Deviation / Mean Absolute Deviation

Mean absolute deviation is the average distance between mean and each data point.

$$Mean\,absolute\,deviation\,(MAD) = \frac{\sum |x_i - \bar{x}|}{n}$$

```
win_by_runs_mad = win_by_runs_data.mad()
```

```
win_by_runs_mad = win_by_runs_data.mad()
print(f'Mean absolute deviation = {win_by_runs_mad}')
```

```
Mean absolute deviation = 20.089144044589645
```

Measure of Position

Range

Range is the simplest form of measuring variability. It is the difference between largest number and s

1 cell hidden

Percentile

Percentile is a way to represent position of a values in data set. To calculate percentile, values in data

In general, if k is nth percentile, it implies that n% of the total terms are less than k.

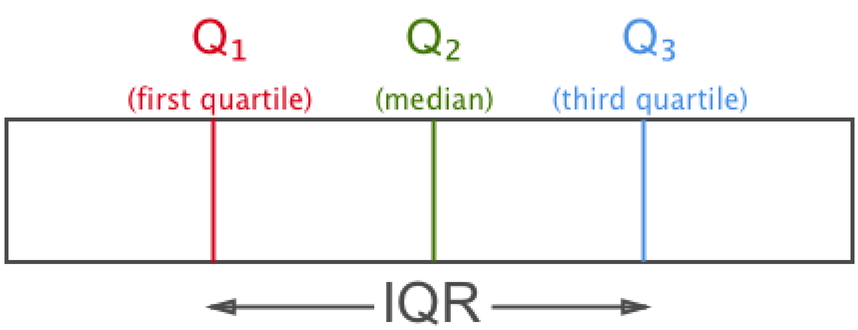
2 cells hidden

Interquartile Range (IQR)

Interquartile range or IQR is the amount spread in middle 50% of the dataset or the distance between

- First Quartile (Q_1) = Median of data points to left of the median in ordered list (25th percentile)
- Second Quartile (Q_2) = Median of data (50th percentile)
- Third Quartile (Q_3) = Median of data points to right of the median in ordered list (75th percentile)

IQR = Q_3 - Q_1



The range gives us a measurement of how spread out the entirety of our data set is. The interquartile first and third quartile are, indicates how spread out the middle 50% of our set of data is.

The primary advantage of using the interquartile range rather than the range for the measurement of t interquartile range is not sensitive to outliers.

Note: If you sort data in descending order, the magnitude will be same, just sign will differ. Negative IQ order. It just we negate smaller values from larger values, we prefer ascending order (Q3 - Q1).

```
quant = np.quantile(win_by_runs_data,[0.25,0.75])
print('IQR =', (quant[1] - quant[0]))
```

```
IQR = 28.0
```

```
print('IQR =', stats.iqr(win_by_runs_data))
```

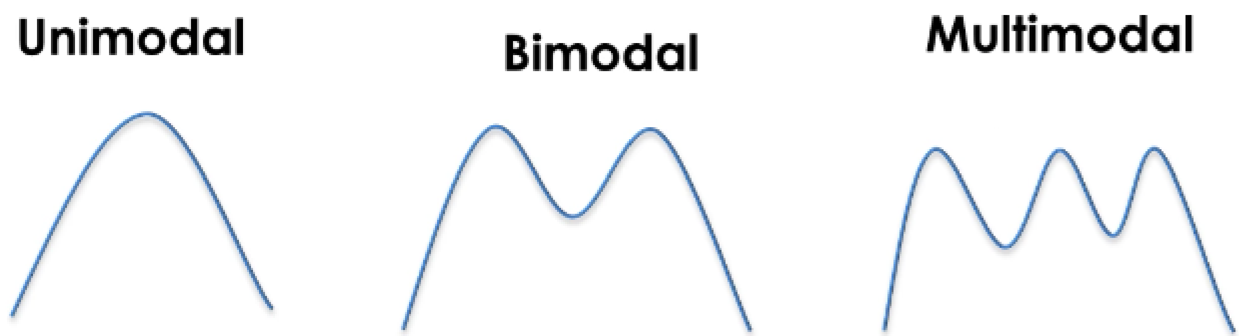
```
IQR = 28.0
```

```
print('IQR =',(stats.scoreatpercentile(win_by_runs_data,75) - stats.scoreatpercentile(wi
```

```
IQR = 28.0
```

Modality

The modality of a distribution is determined by the number of peaks it contains. Most distributions ha encounter distributions with two or more peaks. The picture below shows visual examples of the thre



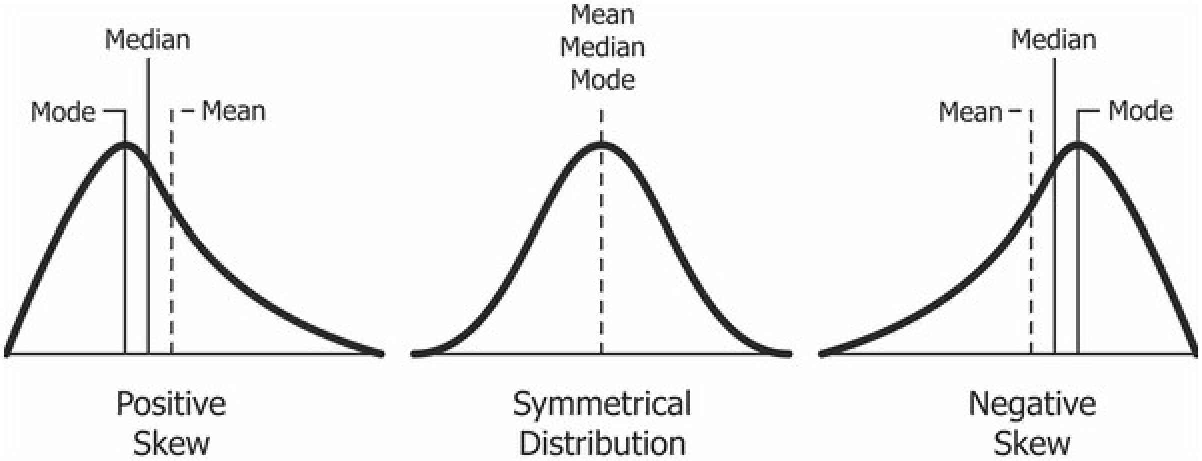
Unimodal means that the distribution has only one peak, which means it has only one frequently occu bimodal distribution has two values that occur frequently (two peaks) and a multimodal has two or se

Skewness

Skewness is a measurement of the symmetry of a distribution.

Therefore it describes how much a distribution differs from a normal distribution, either to the left or t either positive, negative or zero. Note that a perfect normal distribution would have a skewness of zer

Below you can see an illustration of the different types of skewness:

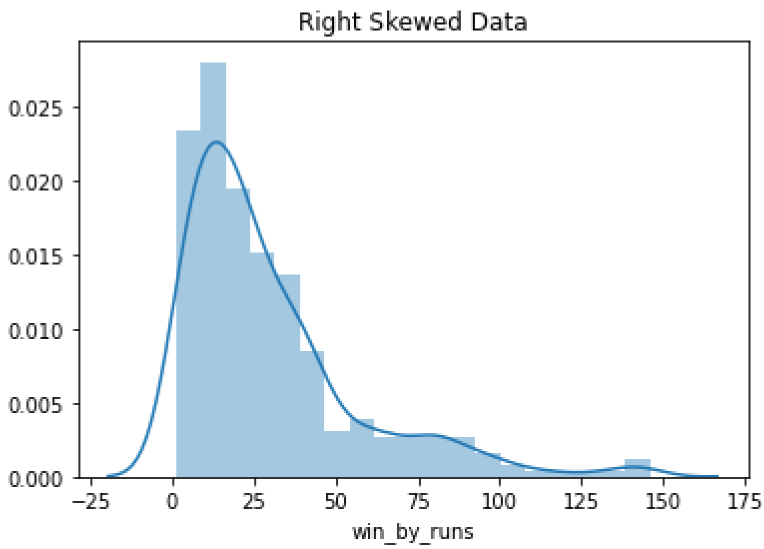



```
win_by_runs_data.skew()
```

```
1.7570395489658672
```

```
sns.distplot(win_by_runs_data)
plt.title('Right Skewed Data')
```

```
Text(0.5, 1.0, 'Right Skewed Data')
```



Transforming the skewed data to normal distribution discussed in Transformation in data Processing

```
# Also Checking Normalaity Tests
stats.normaltest(win_by_runs_data) #D’Agostino’s K^2 Test
#The D’Agostino’s K^2 test calculates summary statistics from the data, namely kurtosis
```

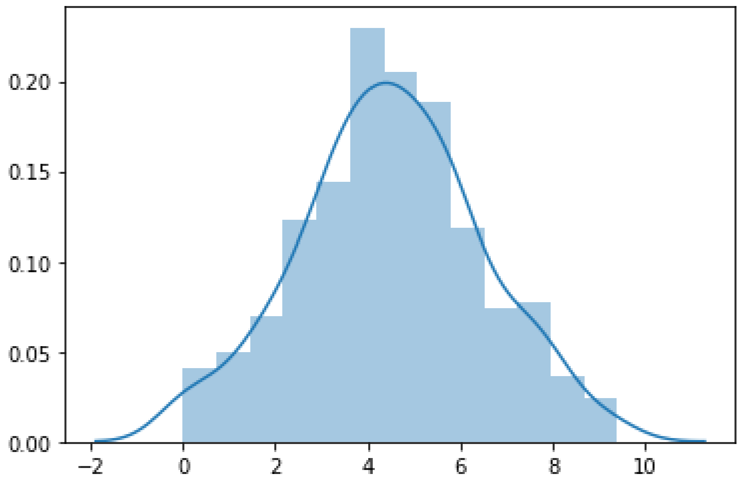
```
NormaltestResult(statistic=123.09813352575927, pvalue=1.8602869461677737e-27)
```

```
stats.shapiro(win_by_runs_data) # Shapiro-Wilk Test
# The Shapiro-Wilk test evaluates a data sample and quantifies how likely it is that the
```

```
(0.8269662857055664, 9.895202878381175e-19)
```

```
normalized_data, lambda_val = stats.boxcox(win_by_runs_data)
sns.distplot(normalized_data)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4597c56668>
```



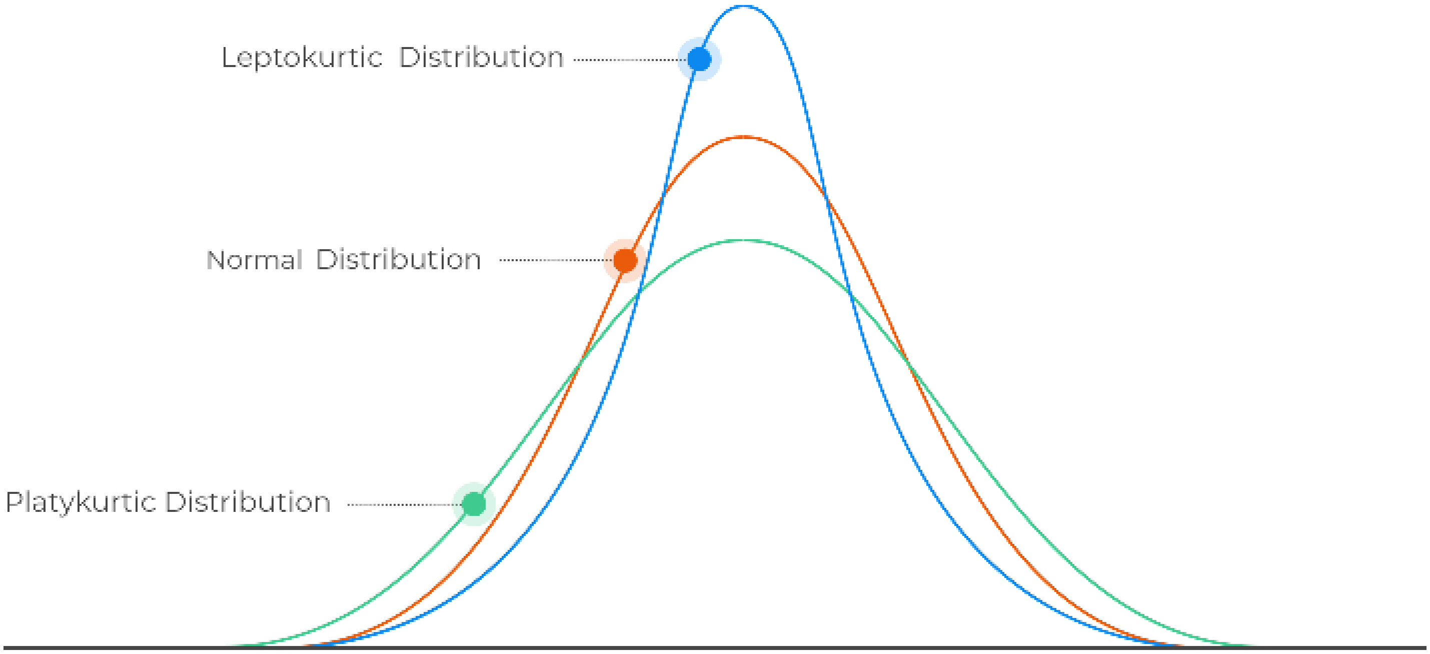

```
from scipy.special import inv_boxcox
original_data = inv_boxcox(normalized_data,lambda_val)
```

Kurtosis

Kurtosis measures whether your dataset is heavy-tailed or light-tailed compared to a normal distributi heavy tails and more outliers and data sets with low kurtosis tend to have light tails and fewer outliers

- A histogram is an effective way to show both the skewness and kurtosis of a data set because y with your data.
- A probability plot is also a great tool because a normal distribution would just follow the straigh

A good way to mathematically measure the kurtosis of a distribution is fishers measurement of kurto



The three most common types of kurtosis:

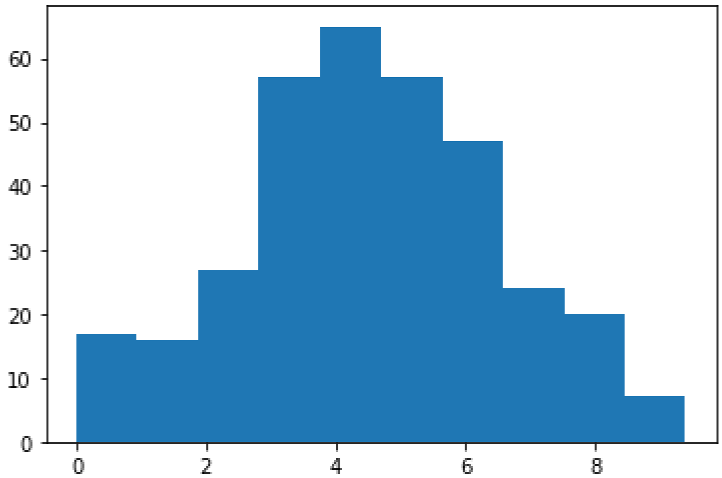
- A normal distribution is called mesokurtic and has kurtosis of or around zero.
- A platykurtic distribution has negative kurtosis and tails are very thin compared to the normal di
- Leptokurtic distributions have kurtosis greater than 3 and the fat tails mean that the distribution has a relatively small standard deviation.

```
pd.DataFrame(normalized_data).kurtosis() #platykurtic distribution
```

```
0   -0.175594
dtype: float64
```

```
plt.hist(normalized_data)
```

```
(array([17., 16., 27., 57., 65., 57., 47., 24., 20.,  7.]),
 array([0.        , 0.94012898, 1.88025795, 2.82038693, 3.7605159 ,
        4.70064488, 5.64077385, 6.58090283, 7.5210318 , 8.46116078,
        9.40128975]),
 <a list of 10 Patch objects>)
```



Probability Plots

To fully understand the concepts of probability plots let's quickly go over a few definitions from proba

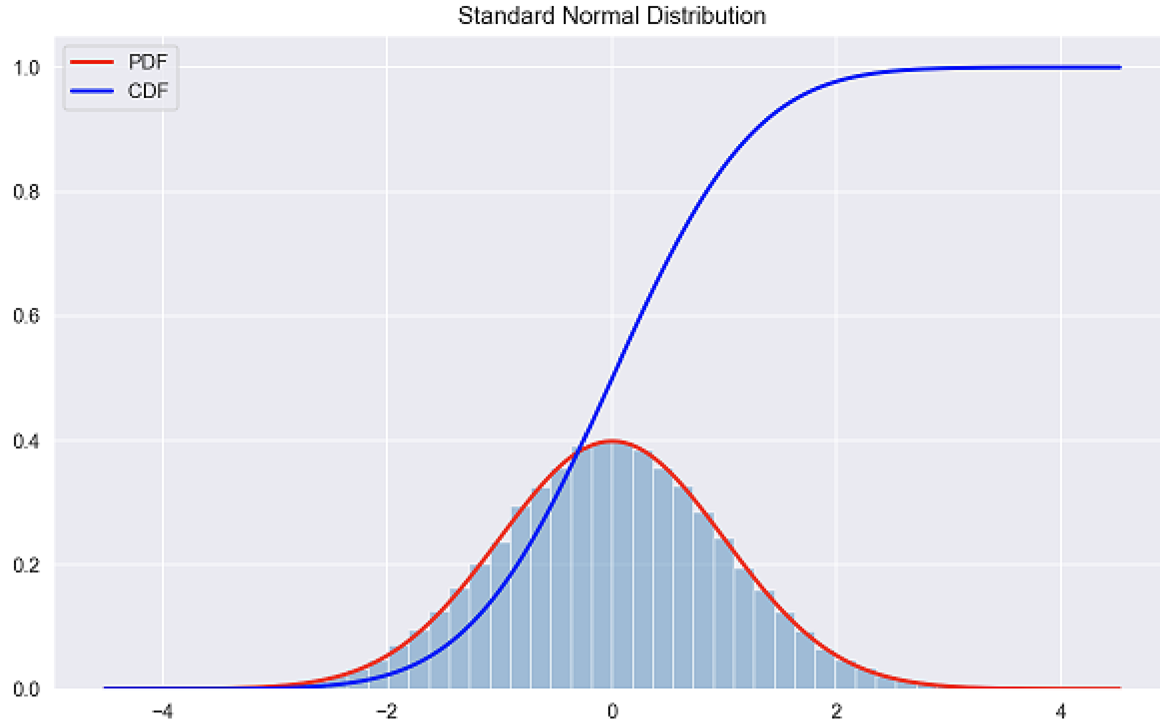
- ***Probability Density Function (PDF)*** — A function that allows us to calculate probabilities of findir belongs to the sample space. It is important to remember that the probability of a continuous ra equal to 0.

PDF of Gaussian Distribution

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- ***Cumulative Distribution Function (CDF)*** — A function that provides the probability of a random va given value x. When we are dealing with continuous variables, the CDF is the area under the PDF

$$F_X(x) = \mathrm{P}(X \leq x)$$



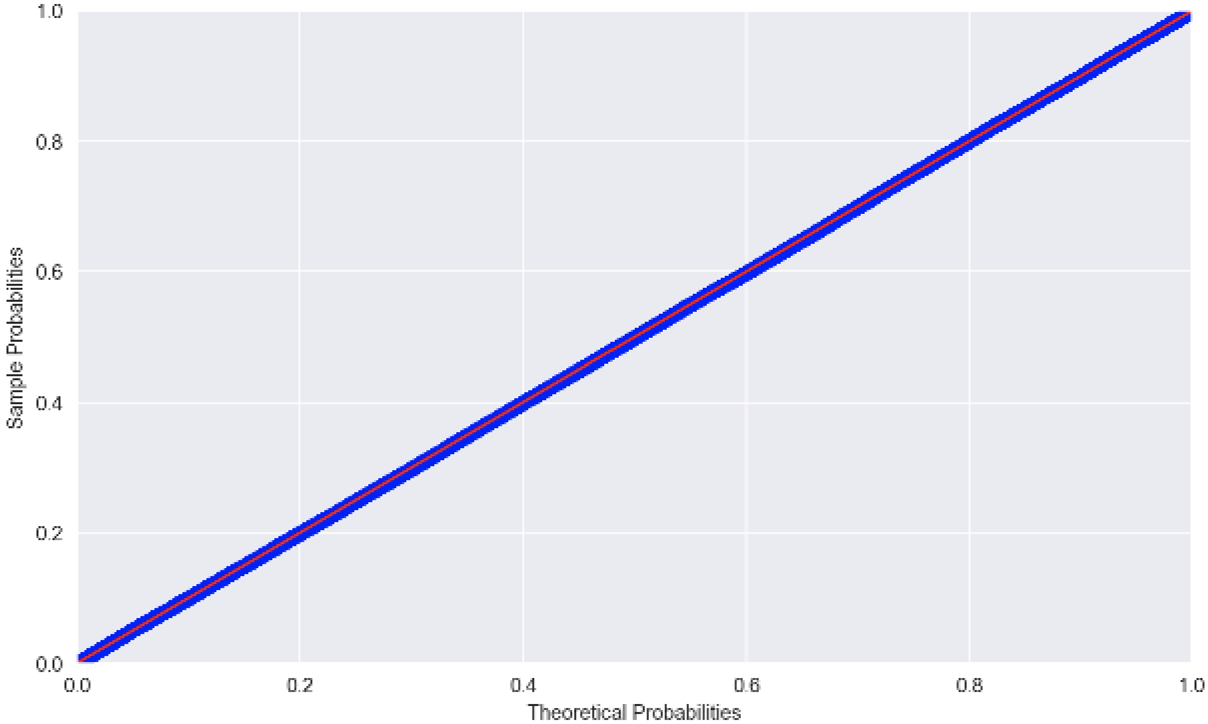
We use probability plots to visually compare data coming from different datasets (distributions). The

- two empirical sets
- one empirical and one theoretical set
- two theoretical sets

The most common use for probability plots is the middle one, when we compare observed (empirical) theoretical distribution like Gaussian.

P-P Plot

P-P (probability–probability) plot is a visualization that plots CDFs of the two distributions (empirical a



Some key information on P-P plots:

1. Interpretation of the points on the plot: assuming we have two distributions (f and g) and a poin
   the plot indicates what percentage of data lies at or below z in both f and g (as per definition of
2. To compare the distributions we check if the points lie on a 45-degree line (x=y). In case they de
3. P-P plots are well suited to compare regions of high probability density (center of distribution) b
   theoretical CDFs change more rapidly than in regions of low probability density.
4. P-P plots require fully specified distributions, so if we are using Gaussian as the theoretical distr
   scale parameters. Changing the location or scale parameters does not necessarily preserve the
5. P-P plots can be used to visually evaluate the skewness of a distribution.

Q-Q Plot

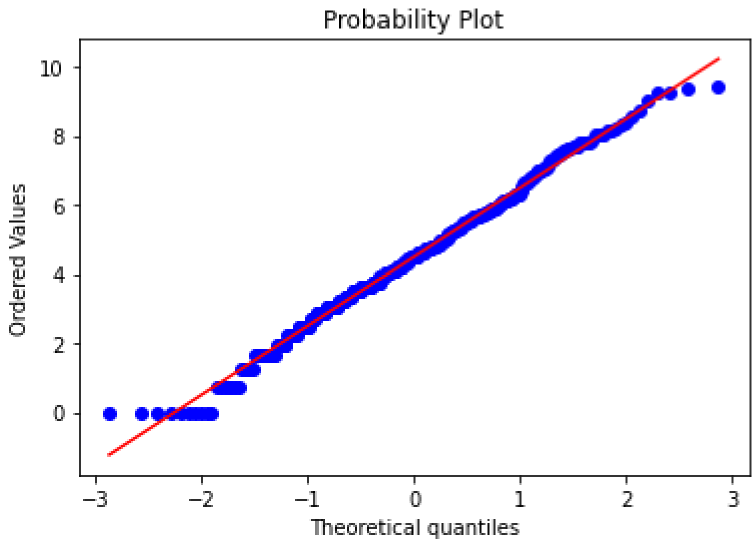
Similarly to P-P plots, Q-Q (quantile-quantile) plots allow us to compare distributions by plotting their

Some key information on Q-Q plots:

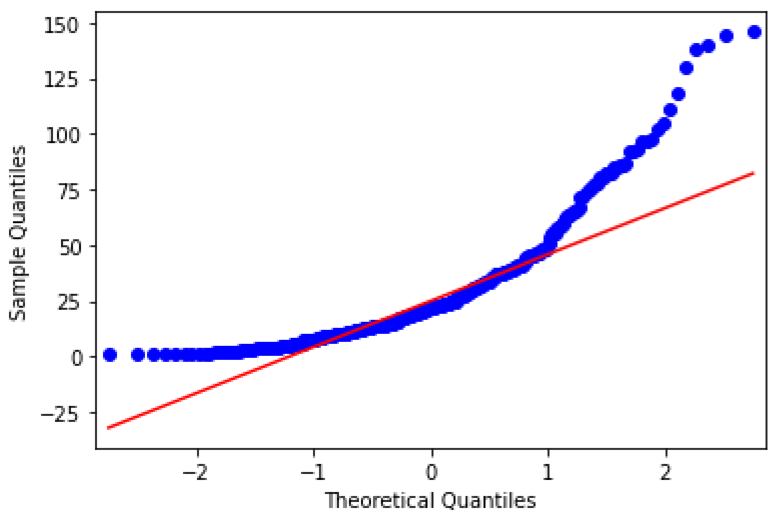
1. Interpretation of the points on the plot: a point on the chart corresponds to a certain quantile co
   most cases empirical and theoretical).
2. On a Q-Q plot, the reference line is dependent on the location and scale parameters of the theor
   are equal to the location and scale parameters respectively.
3. A linear pattern in the points indicates that the given family of distributions reasonably describe
4. Q-Q plot gets very good resolution at the tails of the distribution but worse in the center (where
5. Q-Q plots do not require specifying the location and scale parameters of the theoretical distribu
   computed from a standard distribution within the specified family.

6. The linearity of the point pattern is not affected by changing location or scale parameters.
7. Q-Q plots can be used to visually evaluate the similarity of location, scale, and skewness of the t

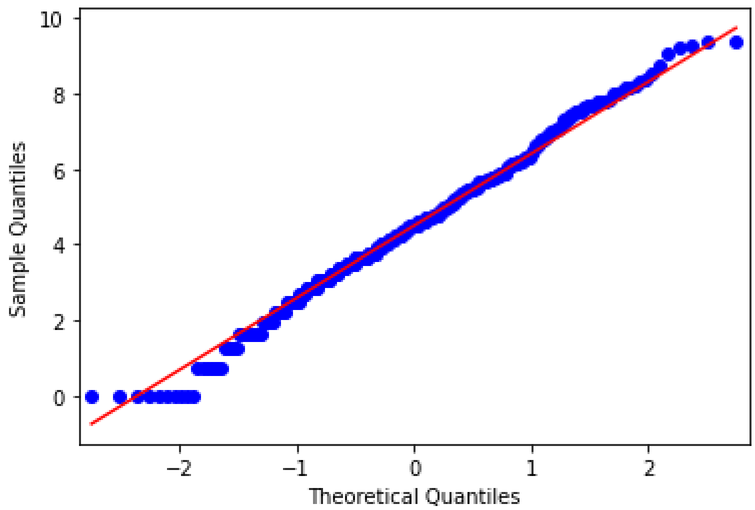
```
stats.probplot(normalized_data,  plot=plt)
plt.show()
```




```
## Q-Q plots of Original data
from statsmodels.api import qqplot
qqplot(win_by_runs_data, line = 'q');
```




```
## Q-Q plots of Normalized Data
from statsmodels.api import qqplot
qqplot(normalized_data, line = 'q');
```


Sample Quantiles
Theoretical Quantiles
10
8
6
4
2
0
−2
−1
0
1
2