DATA 601
Fundamentals of Statistics for Data Science
Part I

Ergun Simsek

Outline

- Why do we need statistics?
  - A very gentle introduction to Linear Regression
- Descriptive vs. Inferential Statistics
- Central Tendency and Variability
- Standard Deviation and z-Score
- Distribution of Sample Means
- Central Limit Theorem

Introduction

- Statistical knowledge helps you use the proper methods to collect the data, employ the correct analyses, and effectively present the results.
- Statistics is a crucial process behind how we make
  - discoveries in science,
  - decisions based on data,
  - predictions.
- Statistics has always been heavily used in business!

Examples of Business Questions

Simple (descriptive) Stats

- "Who are the most profitable customers?"

Hypothesis Testing

- "Is there a difference in value to the company of these customers?"

Segmentation/Classification

- What are the common characteristics of these customers?

Prediction

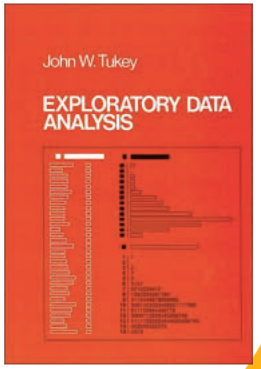
- Will this new customer become a profitable customer? If so, how profitable?

adapted from Provost and Fawcett, "Data Science for Business"

Exploratory Data Analysis 1977

- Based on insights developed at Bell Labs in the 60's, he introduced many basic techniques for visualizing and summarizing data
  - 5-number summary, box plots, stem and leaf diagrams,…
- Instead of confirming observations with data, he was more interested in learning from the data
- 5 Number summary: min, max, median and two quartiles

John Tukey
(also father of FFT algorithm)



Before we talk about these, let's have a gentle introduction to Linear Regression!

Intro to Linear Regression

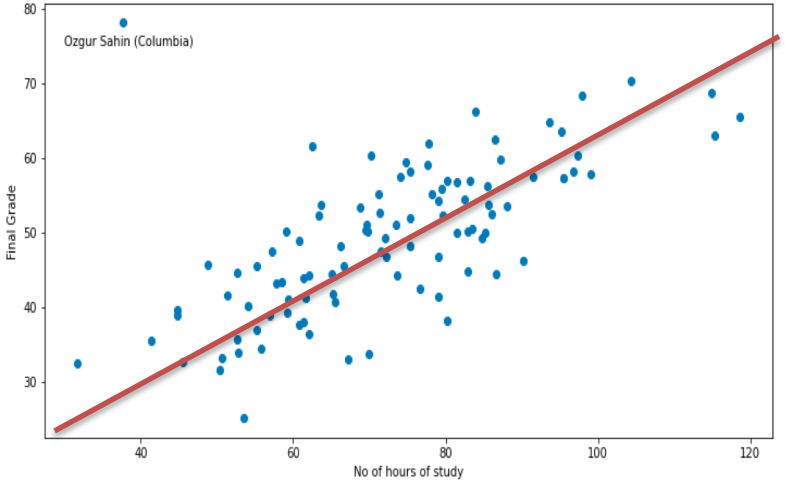
- Assume that we have a simple dataset with N pairs of x and y values, e.g.

$$(x_0,y_0),(x_1,y_1),\cdots,(x_N,y_N)$$

- We want to formulate a linear model to describe the relationship between x and y. The linear model can be express as:

$$y=\beta_0+\beta_1 x$$

- For a linear function, β_0 is the intercept, and β_1 is the slope. These are two parameters that we want to estimate.



Dataset: Grade vs Number of Hours

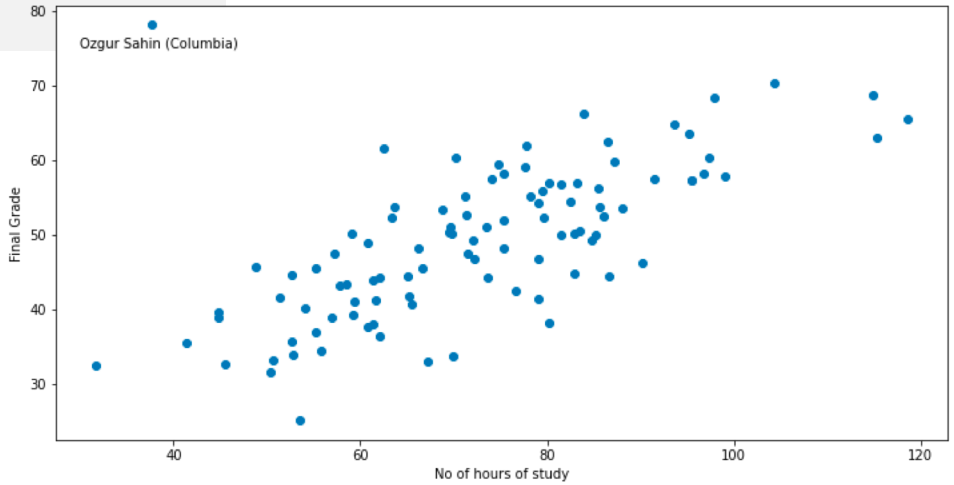
```
# let's load a sample dataset

gvh = pd.read_csv("grade_vs_no_of_hours.csv")

gvh.head()
```

Note that this dataset is available in our class repo

	grade	no_of_hours
0	32.502345	31.707006
1	53.426804	68.777596
2	61.530358	62.562382
3	47.475640	71.546632
4	59.813208	87.230925

Dataset: Grade vs Number of Hours

```
# Let's see what we have

fig = plt.figure(figsize=(12,6))

plt.scatter(gvh.no_of_hours, gvh.grade)

plt.xlabel('No of hours of study')

plt.ylabel('Final Grade')

plt.text(30,75,'Ozgur Sahin (Columbia)')

plt.show()
```



LR on Grade vs Number of Hours

Numpy has a `polyfit` function which can determine β_0 and β_1 for us. Later we'll learn how we calculate them.

Degree of the polynomial fit

```
Lin_Reg_Coefficients = np.polyfit(gvh.no_of_hours, gvh.grade, 1)

Lin_Reg_Coefficients
```

Output: array([0.3980811 , 20.43090502])

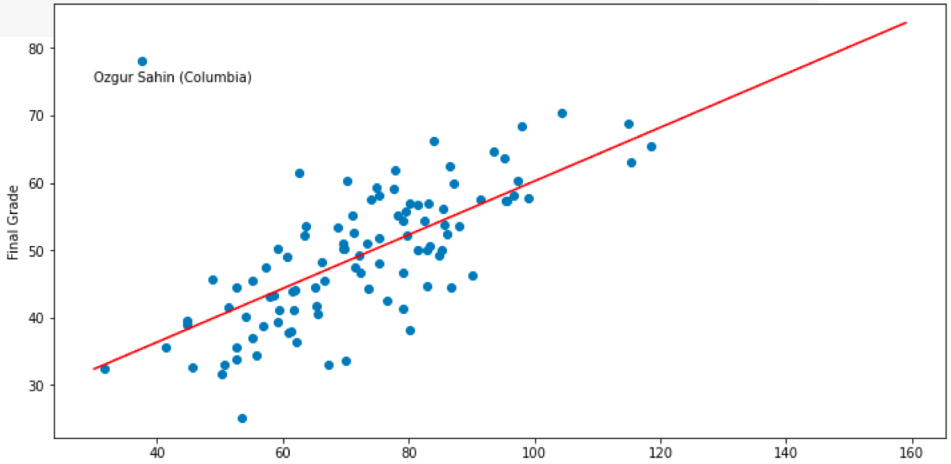
Slope

Intercept

LR on Grade vs Number of Hours

```
x = np.arange(30,160)
y_prediction = Lin_Reg_Coefficients[0]*x+Lin_Reg_Coefficients[1]
```

```
fig = plt.figure(figsize=(12,6))
plt.scatter(gvh.no_of_hours, gvh.grade)
plt.plot(x,y_prediction,'r')
plt.xlabel('No of hours of study')
plt.ylabel('Final Grade')
plt.text(30,75,'Ozgur Sahin (Columbia)')
plt.show()
```



Prediction with LR

- Guess the grade of a student who works for 160 hours in total

```
x_guess = 160
y_guess = Lin_Reg_Coefficients[0]*x_guess+Lin_Reg_Coefficients[1]
print(y_guess)
```

```
84.12388093958388
```

- Do you see a problem (hint what if the student works for 1600 hours??)

```
x_guess = 1600
y_guess = Lin_Reg_Coefficients[0]*x_guess+Lin_Reg_Coefficients[1]
print(y_guess)
```

```
657.3606641915338
```

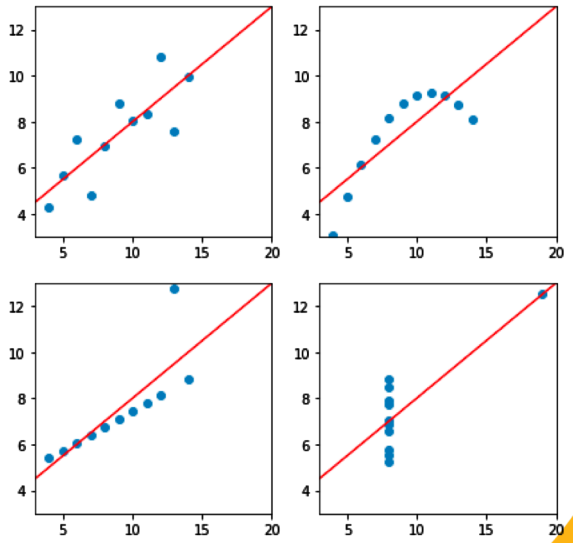
Another Example: Anscombe's Quarted

This famous dataset was created by statistician Francis Anscombe in 1973. There are four groups each consisting of 11 pairs of x and y coordinates.

```
anscombe = pd.read_csv("https://raw.githubusercontent.com/simsekergun/DATA601/main/Datasets/anscombes.csv")
```

	xA	yA	xB	yB	xC	yC	xD	yD
0	10	8.04	10	9.14	10	7.46	8	6.58
1	8	6.95	8	8.14	8	6.77	8	5.76
2	13	7.58	13	8.74	13	12.74	8	7.71
3	9	8.81	9	8.77	9	7.11	8	8.84
4	11	8.33	11	9.26	11	7.81	8	8.47
5	14	9.96	14	8.10	14	8.84	8	7.04
6	6	7.24	6	6.13	6	6.08	8	5.25
7	4	4.26	4	3.10	4	5.39	19	12.50
8	12	10.84	12	9.13	12	8.15	8	5.56
9	7	4.82	7	7.26	7	6.42	8	7.91
10	5	5.68	5	4.74	5	5.73	8	6.89

```
fig = plt.figure(figsize=(7,7))

ax1 = fig.add_subplot(221)
ax1.scatter(anscombe.xA, anscombe.yA)
ax1.set_xlim(3,20)
ax1.set_ylim(3,13)

ax2 = fig.add_subplot(222)
ax2.scatter(anscombe.xB, anscombe.yB)
ax2.set_xlim(3,20)
ax2.set_ylim(3,13)

ax3 = fig.add_subplot(223)
ax3.scatter(anscombe.xC, anscombe.yC)
ax3.set_xlim(3,20)
ax3.set_ylim(3,13)

ax4 = fig.add_subplot(224)
ax4.scatter(anscombe.xD, anscombe.yD)
ax4.set_xlim(3,20)
ax4.set_ylim(3,13)
```



Need for Statistics

- Linear regression is very useful and powerful but there are certain limits where it could be applied.
- We heavily use statistics to determine these limits.
- We had two very simple cases where we could determine the issue visually but when there are tens of variables, things will get complicated.
- We really need to understand distributions, how to characterize them, and how to transform them (when needed).

Descriptive vs. Inferential Statistics

- **Descriptive:** e.g., median; describes data you have but can't be generalized beyond that
- **Inferential:** e.g., t-test, that enable inferences about the population beyond our data

We'll use both approaches in Machine Learning

Supervised Learning:

kNN (k Nearest Neighbors)
Naïve Bayes
Logistic Regression
Support Vector Machines
Random Forests

Unsupervised Learning:

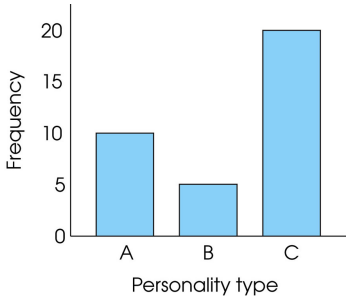
Clustering
Factor analysis
Latent Dirichlet Allocation

Frequency Distributions

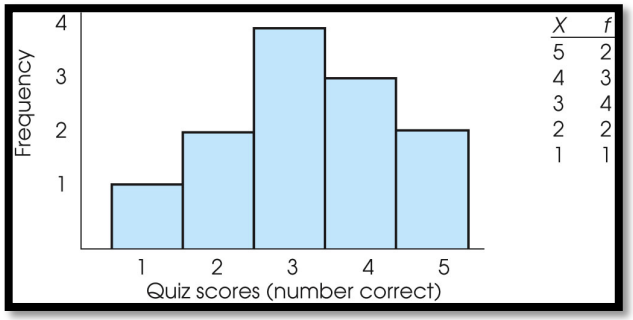
Frequency Distributions

- After data collection and cleaning, we can get a general overview of the dataset with descriptive statistics
- Step-1: Construct a **frequency distribution**.
  - Show exactly how many individuals are located in each category on the scale of measurement

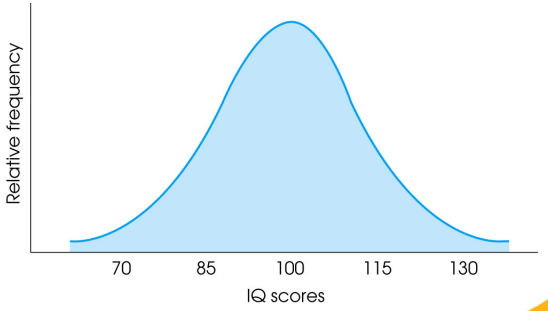
Bar plot



Histogram

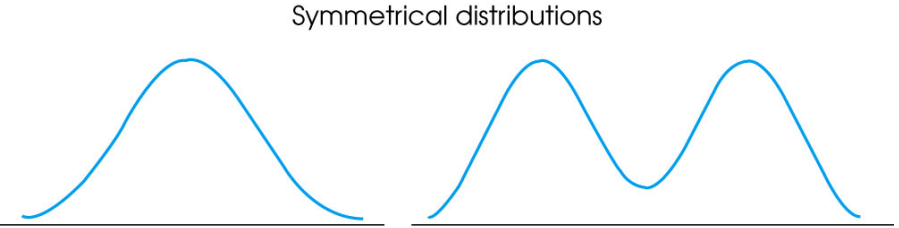


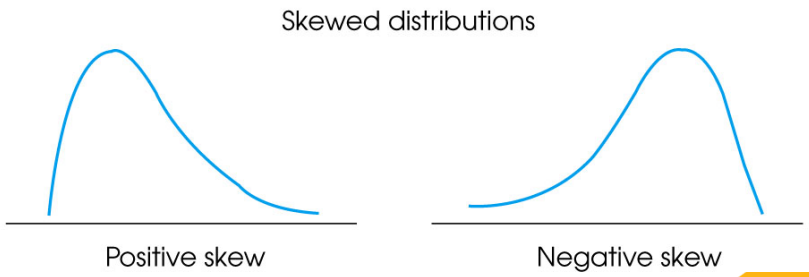
Smooth Curves



Frequency distribution graphs

- Frequency distribution graphs are useful because they show the entire set of scores.
- At a glance, you can determine the highest score, the lowest score, and where the scores are centered.
- The graph also shows whether the scores are clustered together or scattered over a wide range.
- In a positively (negatively) skewed distribution, the scores tend to pile up on the left (right) side of the distribution with the tail tapering off to the right (left).





Dataset: Seaborn - Diamonds

Before we discuss Central Tendency, let's take a look at the Diamonds dataset with Seaborn

```
## Import pandas
import pandas as pd

## import seaborn
import seaborn as sns

## import numpy
import numpy as np

## Load the "diamonds" dataset from the seaborn
diamonds = sns.load_dataset("diamonds")
```

Dataset: Seaborn - Diamonds

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
...	...	...	...	...	...	...	...	...	...	...
53935	0.72	Ideal	D	SI1	60.8	57.0	2757	5.75	5.76	3.50
53936	0.72	Good	D	SI1	63.1	55.0	2757	5.69	5.75	3.61
53937	0.70	Very Good	D	SI1	62.8	60.0	2757	5.66	5.68	3.56
53938	0.86	Premium	H	SI2	61.0	58.0	2757	6.15	6.12	3.74
53939	0.75	Ideal	D	SI2	62.2	55.0	2757	5.83	5.87	3.64

53940 rows × 10 columns

Dataset: Seaborn - Diamonds

Column	Range
price price in US dollars	(326 – –18,823)
carat weight of the diamond	(0.2--5.01)
cut quality of the cut	(Fair, Good, Very Good, Premium, Ideal)
color diamond colour	from J (worst) to D (best)
clarity a measurement of how clear the diamond is	(I1 (worst), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (best))
x length in mm	(0--10.74)
y width in mm	(0--58.9)
z depth in mm	(0--31.8)
depth total depth percentage = $\frac{z}{mean(x,y)} = \frac{2 \cdot z}{(x+y)}$	(43--79)
table width of top of diamond relative to widest point	(43--95)

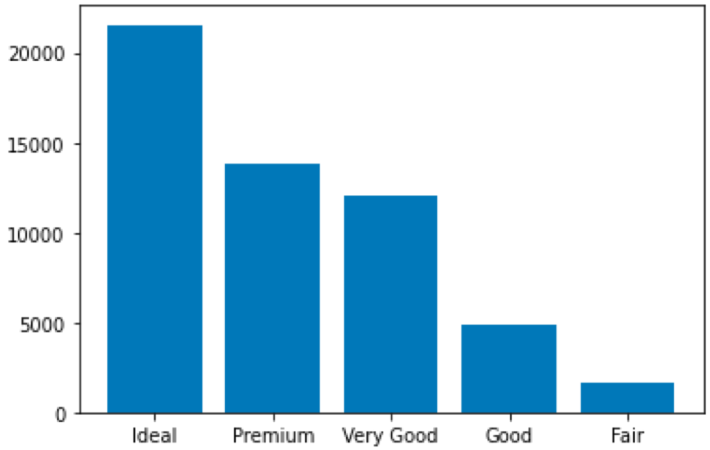
Features and Types

```
diamonds.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53940 entries, 0 to 53939
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   carat    53940 non-null  float64
 1   cut      53940 non-null  category
 2   color    53940 non-null  category
 3   clarity  53940 non-null  category
 4   depth    53940 non-null  float64
 5   table    53940 non-null  float64
 6   price    53940 non-null  int64
 7   x        53940 non-null  float64
 8   y        53940 non-null  float64
 9   z        53940 non-null  float64
dtypes: category(3), float64(6), int64(1)
memory usage: 3.0 MB
```

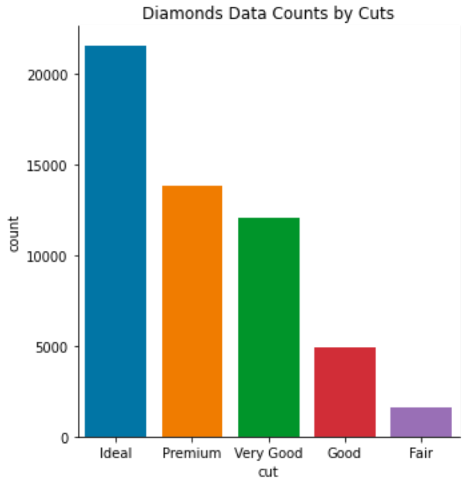
Category Data Visualization with Matplotlib

```
## Let's see the number of diamonds for each different type of cuts

cuts = diamonds.cut.value_counts()

import matplotlib.pyplot as plt
%matplotlib inline

plt.bar(x = cuts.index, height = cuts.values)
plt.draw()
```

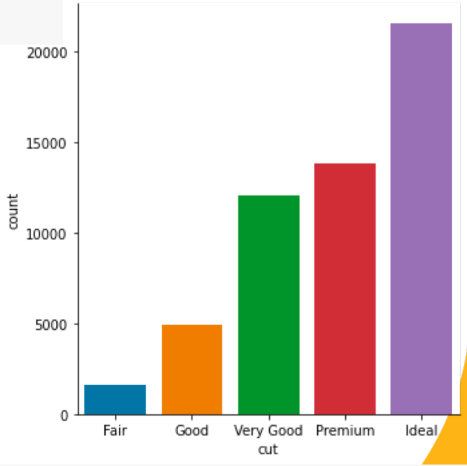


Category Data Visualization with Seaborn

```
## we can use catplot for showing the count
plot = sns.catplot(data= diamonds, kind = 'count', x = 'cut')

## Note that since seaborn is built on top of matplotlib
## we can use matplotlib features like adding a title

plot.ax.set_title('Diamonds Data Counts by Cuts')
```



YOU CAN CHANGE THE ORDER




```
plot = sns.catplot(data= diamonds, kind = 'count', x = 'cut',
                   order= ['Fair','Good','Very Good','Premium', 'Ideal'])
```

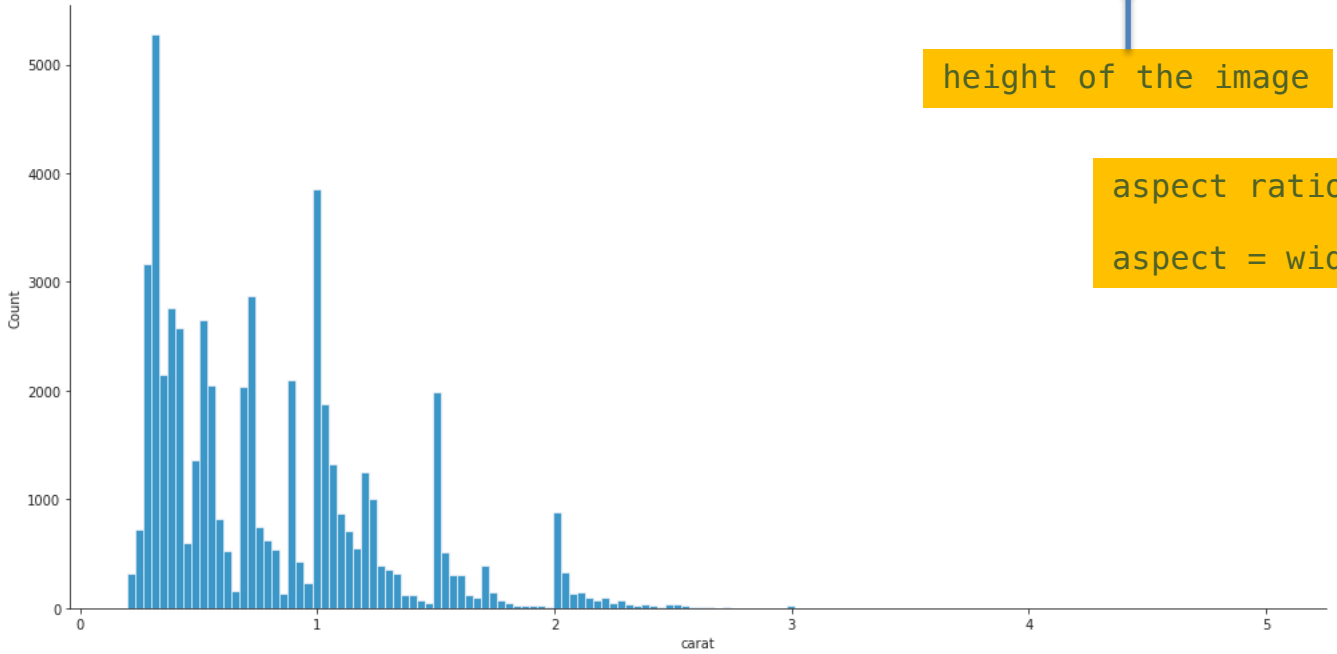
Distribution Visualization with Seaborn-1

```
sns.set_style('ticks')

## we can use displot to see the distribution of 'carat' colum.
plot = sns.displot(data = diamonds, x= 'carat', kind = 'hist', height= 7, aspect = 2)
```

height of the image

aspect ratio of the image

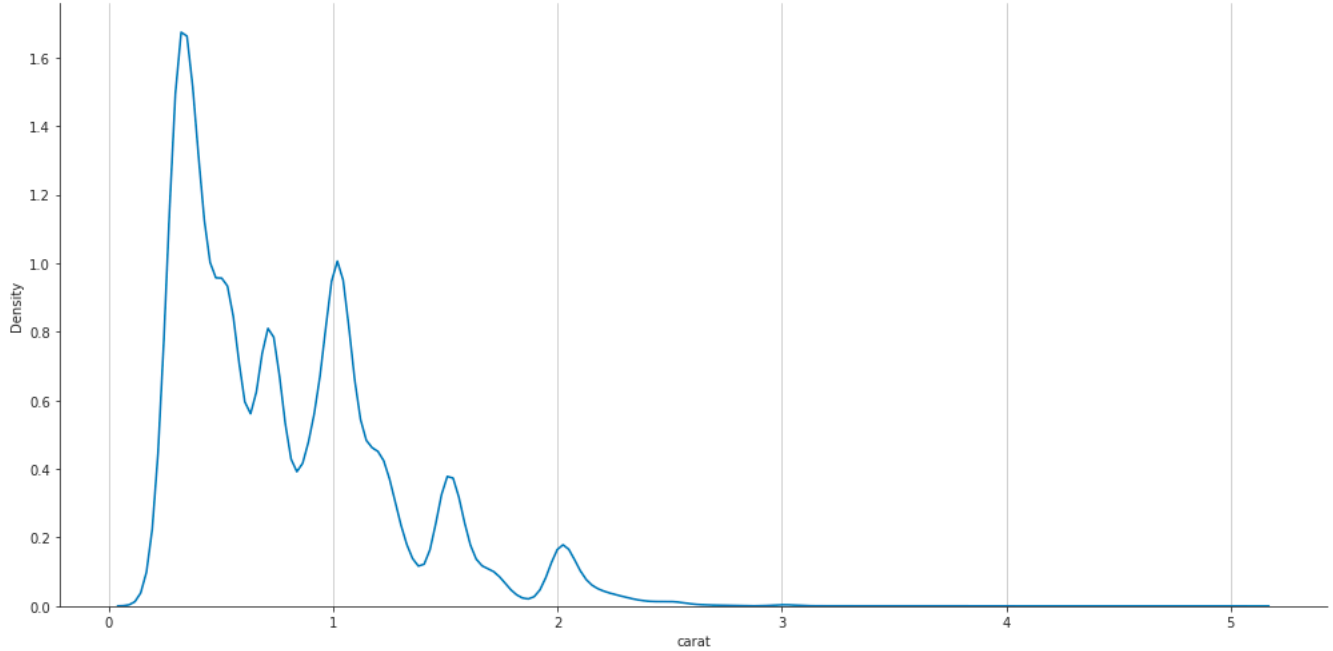
aspect = width/heigt



Distribution Visualization with Seaborn-2

```
## or we can just take a look at the densities
plot = sns.displot(data = diamonds, x= 'carat', kind = 'kde', height= 7, aspect = 2)

## note that we can add a grid also
plot.ax.grid(axis = 'x')
```

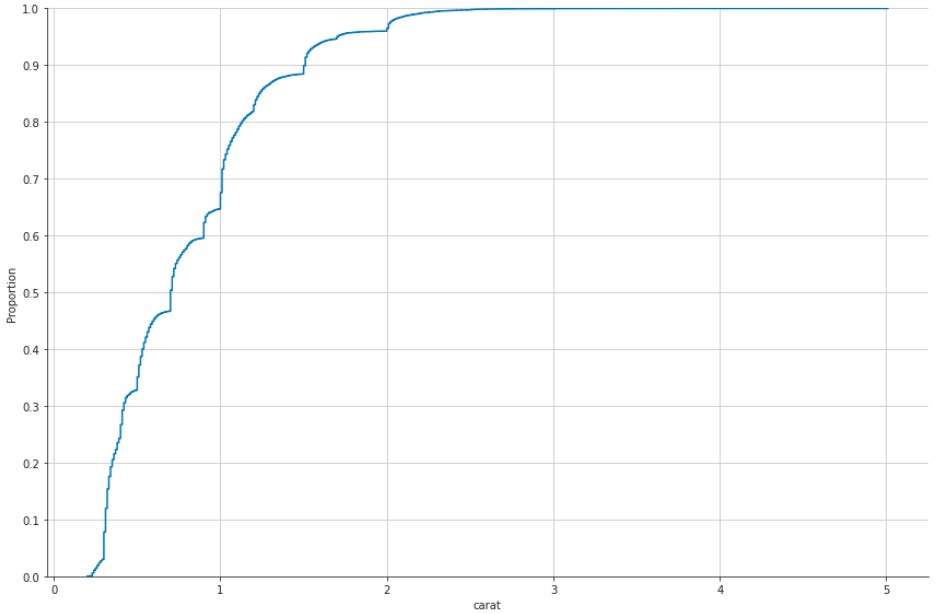
kernel density estimate



Distribution Visualization with Seaborn-2

empirical cumulative distribution function

```
plot= sns.displot(data=diamonds, kind="ecdf", x="carat",height= 8, aspect = 1.5)
plot.ax.set_yticks(ticks = np.arange(0,1.1,0.1))
plot.ax.grid()
```



ECDF represents the proportion or count of observations falling below each unique value in a dataset

Central Tendency

Central Tendency

- **Central tendency** is a statistical measure that determines a single value that accurately describes the center of the distribution and represents the entire distribution of scores.
- The goal of central tendency is to identify the single value that is the best representative for the entire set of data.
  - Candidate A (GPA: 3.8)
  - Candidate B (GPA: 2.8)

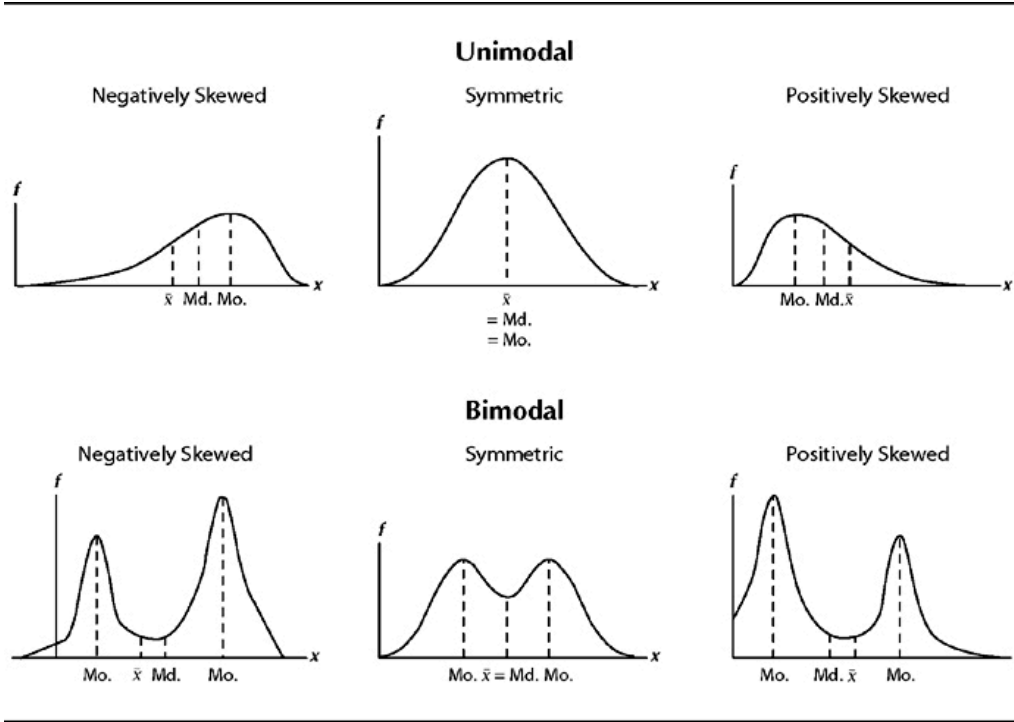
Mean, Median, and Mode

- Mean: sum of scores/num of scores
  - Not useful when a distribution contains a few extreme scores
    - Example: 1985 Salary Survey - New Geography Graduates
  - Inappropriate when data are measured on an ordinal scale (ranks)
- Median: Find the one at the center when sort from smallest to biggest
  - Advantage: the median is relatively unaffected by extreme scores
  - How to find the median from frequency distributions?
- Mode: most frequently occurring category or score in the distribution
  - corresponds to the peak or high point of the distribution
  - Possible for a distribution to have more than one mode (e.g. bimodal).
    - A distribution may have a major mode at the highest peak and a minor mode at a secondary peak in a different location
    - But do not forget a distribution can have only one mean and only one median.

Distributions vs Mean, Median, & Mode

- Symmetrical distributions vs. mean and median
  If there is only one mode, then
  the mode, mean, and median are all the same value.

- Skewed distributions vs. mean, median, and mode
  - The mode will be located at the peak on one side and the mean usually will be displaced toward the tail on the other side.
  - The median is usually located between the mean and the mode.



Variability

Variability

- We use variability to obtain a measure of how spread out the scores are in a distribution.
- Variability serves both as a descriptive measure and as an important component of most inferential statistics.
  - [Descriptive] Variability measures the degree to which the scores are spread out or clustered together in a distribution.
  - [Inferential] Variability provides a measure of how accurately any individual score or sample represents the entire population.

Measuring Variability

- Variability can be measured with
  - the range
  - the interquartile range
  - the standard deviation/variance.
- In each case, variability is determined by measuring *distance*.

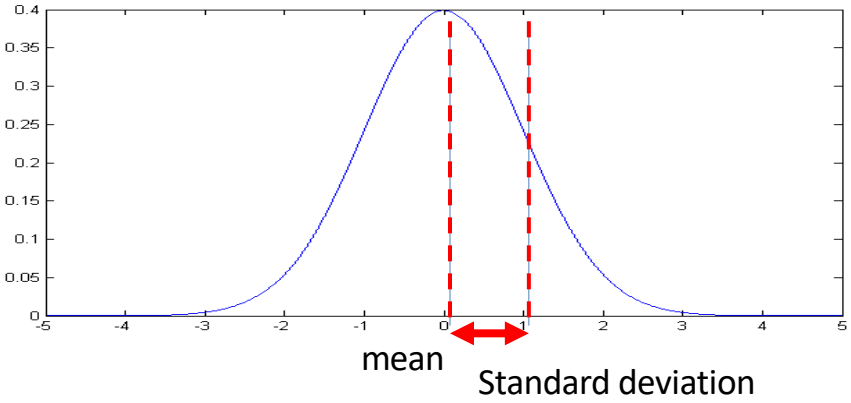
The Range

- The **range** is the total distance covered by the distribution, from the highest score to the lowest score (using the upper and lower real limits of the range).
- In this case, the variance is the mean squared deviation of samples from the sample mean:

$$Var(X) = \frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^2$$

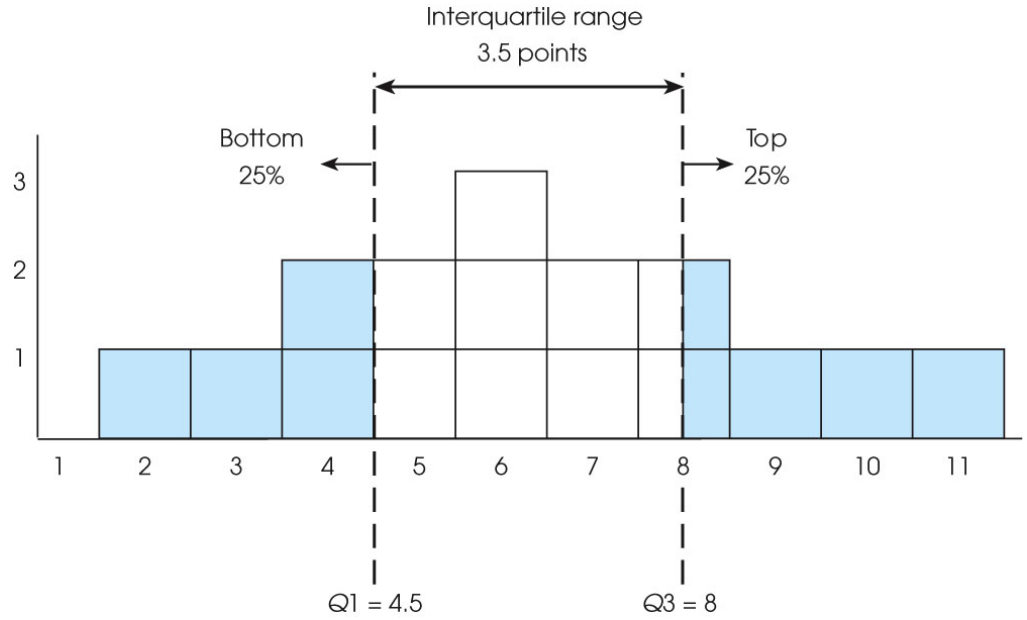
The standard deviation is the square root of variance.

The normal distribution is completely characterized by mean and variance



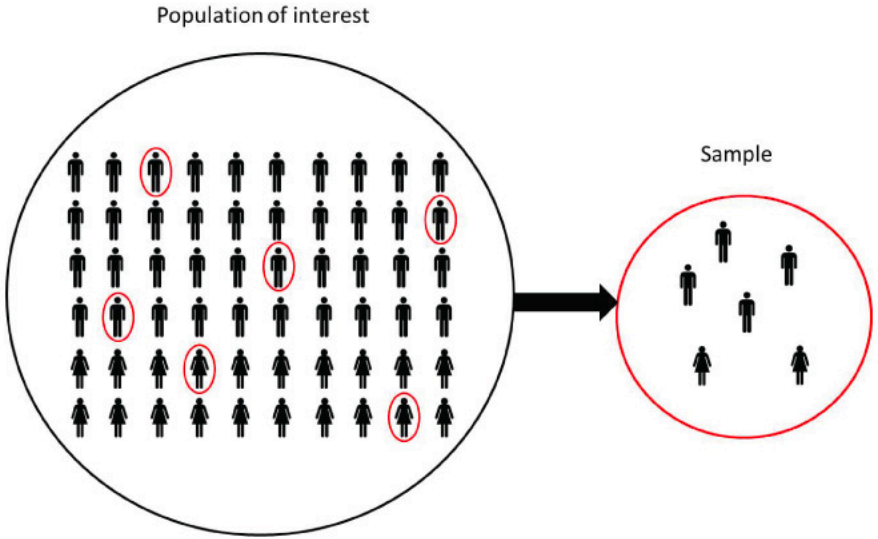
The Interquartile Range

- The **interquartile range** is the distance covered by the middle 50% of the distribution (the difference between Q1 and Q3).



Population vs Sample

- Population: The entire group that we want to draw conclusions about
- Sample: A specific group that you will collect data from. The size of the sample is always less than the total size of the population.



The Standard Deviation

SD measures the standard distance between a score and the mean.

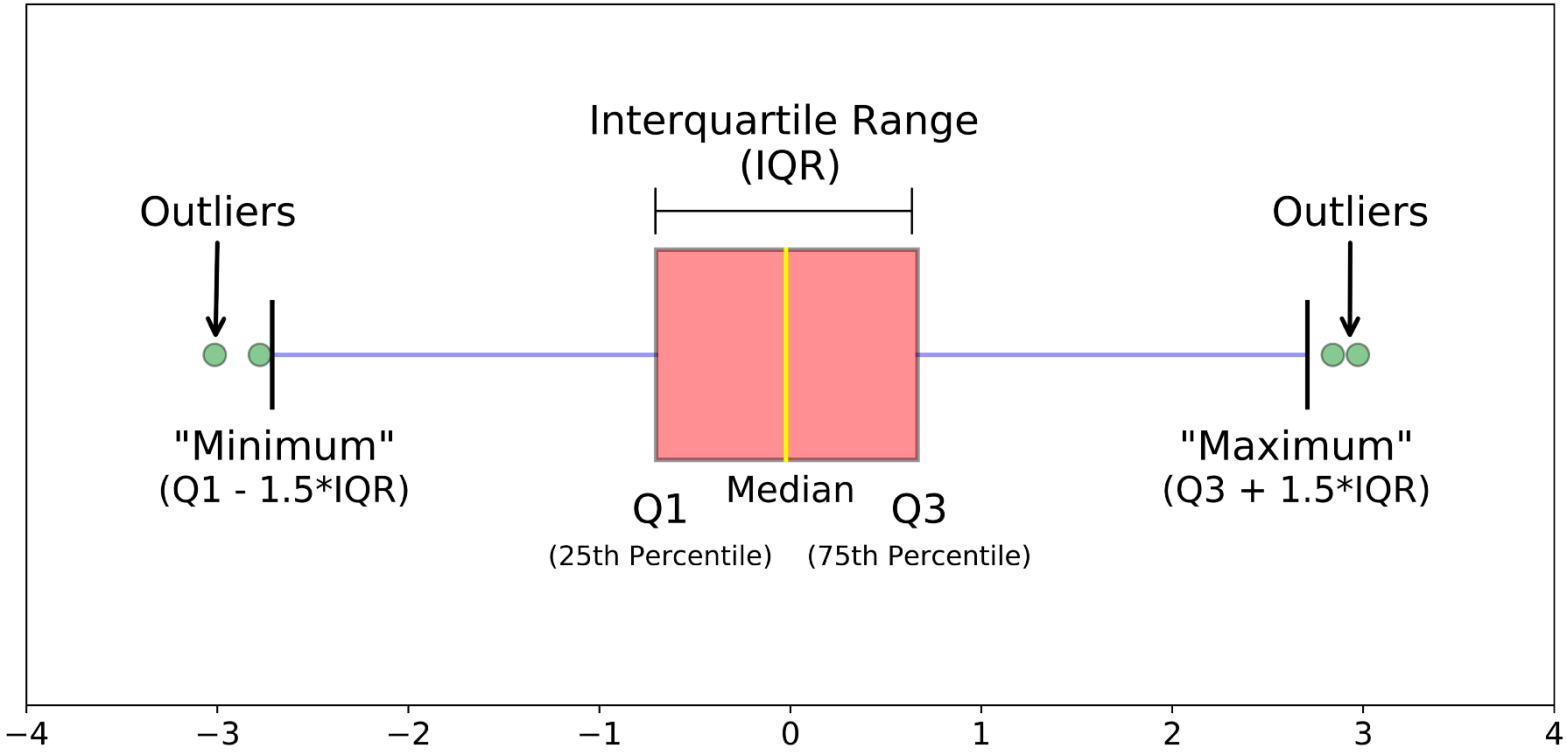
	Sample	Population	
calculated from only some of the individuals in a population	$S = \sqrt{\frac{\sum(x_i - \bar{x})^2}{n-1}}$	$\sigma = \sqrt{\frac{\sum(x_i - \mu)^2}{N}}$	calculated from every individual in the population

Note: If a constant is added to every score in a distribution, the SD will *not* be changed

General rule: roughly

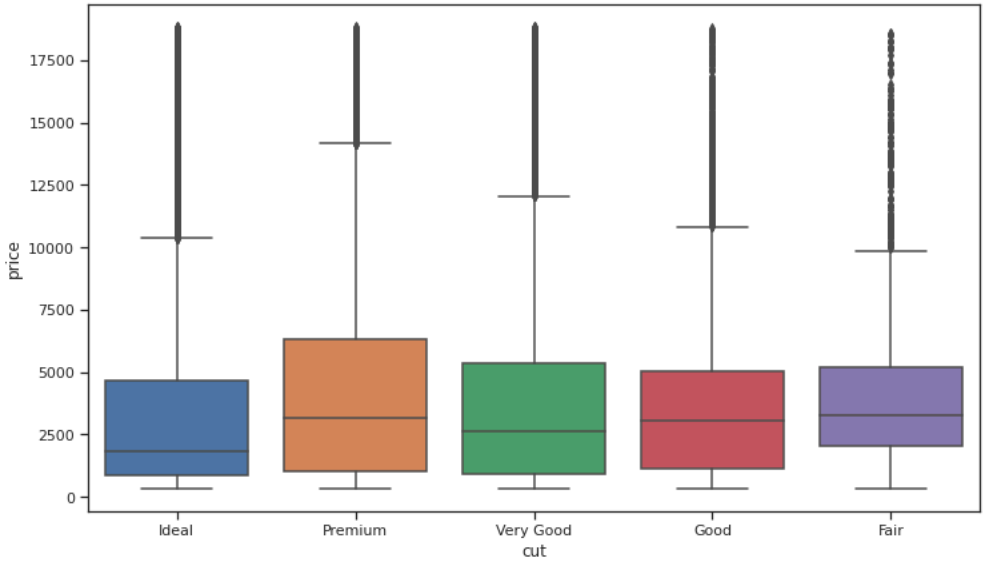
- 68% of the scores will be within 1 standard deviation of the mean,
- 95% of the scores will be within a distance of 2 standard deviations of the mean
- 99.7% of the scores will be within a distance of 3 standard deviations of the mean

Boxplot


Interquartile Range
(IQR)
Outliers
Outliers
"Minimum"
(Q1 - 1.5*IQR)
Q1
(25th Percentile)
Median
Q3
(75th Percentile)
"Maximum"
(Q3 + 1.5*IQR)
−4
−3
−2
−1
0
1
2
3
4

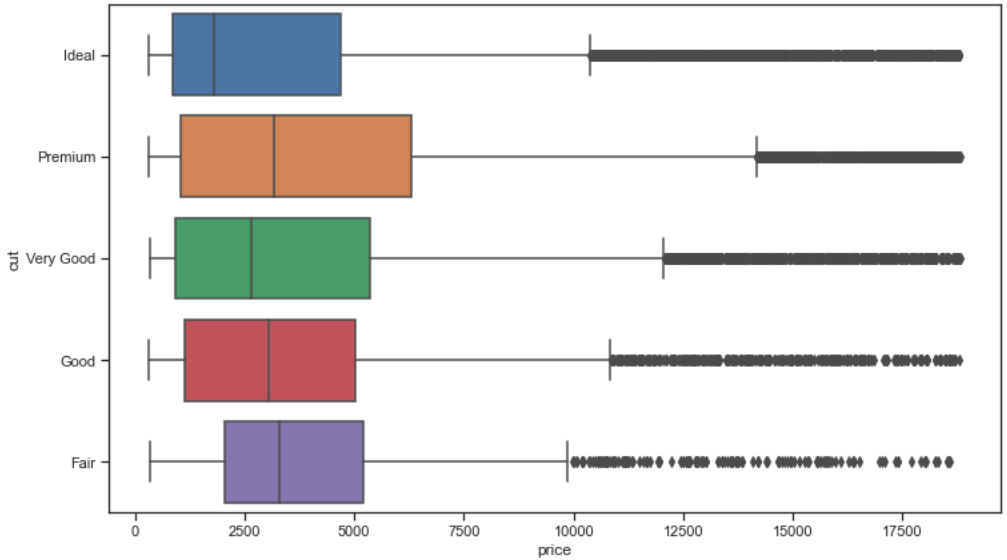
Diamond Dataset and Boxplot-1

```
sns.set(rc={"figure.figsize":(12, 7)}) #width=12, #height=7
sns.set_style('ticks')

plot = sns.boxplot(x = 'cut', y = 'price' , data = diamonds)
```



Diamond Dataset and Boxplot-2

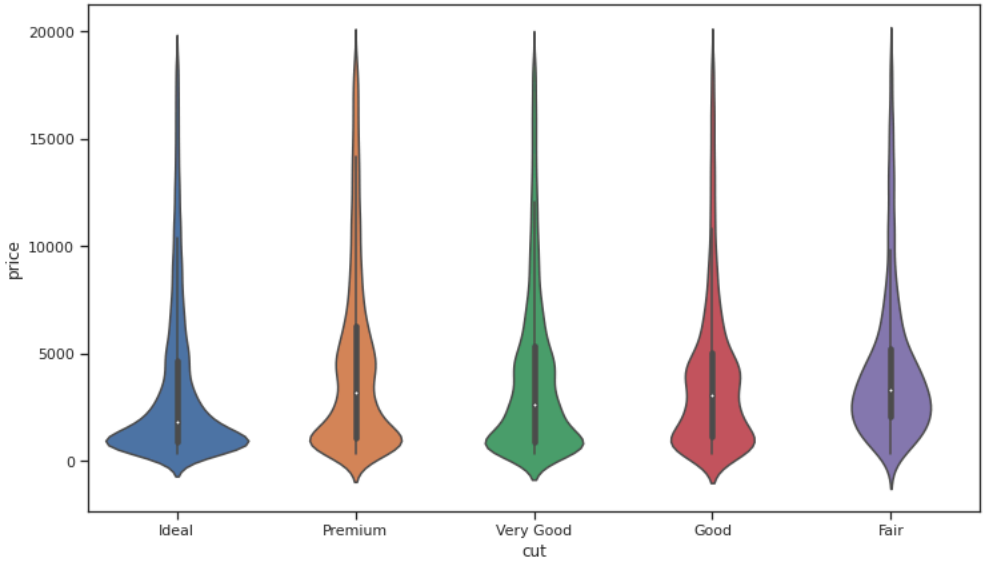
```
plot = sns.boxplot(y = 'cut', x = 'price', data = diamonds)
```



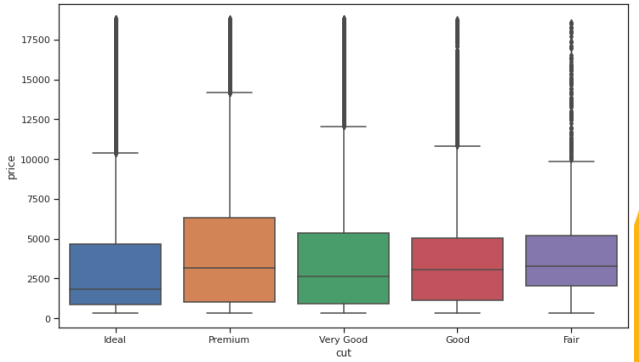
Diamond Dataset and Violinplot

Similar to boxplot but the violin plot features a kernel density estimation of the underlying distribution.

```
sns.violinplot(data = diamonds, x= 'cut', y = 'price')
```



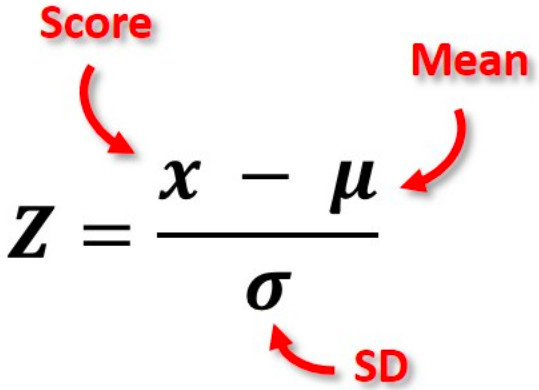
Remember boxplot



z-Scores and Probability

z-Scores and Location

- By itself, a raw score (or X value) provides very little information about how that particular score compares with other values in the distribution.
- We transform a raw score into a z-score to understand exactly where the score is located relative to all the other scores in the distribution

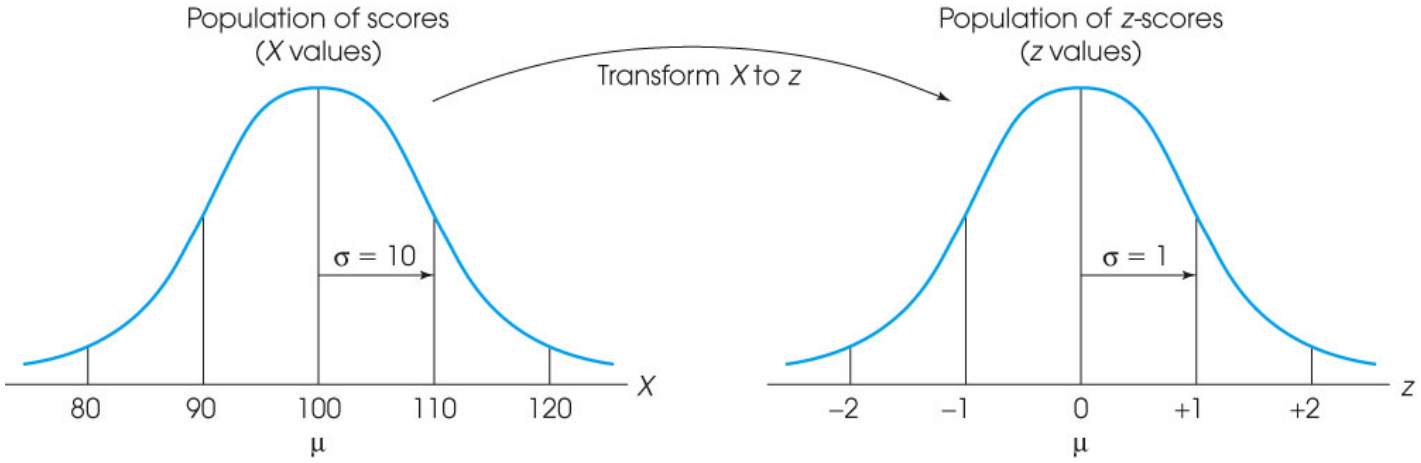


[Descriptive] z-Scores describe exactly where each individual is located.

[Inferential] z-scores determine whether a specific sample is representative of its population, or is extreme and unrepresentative

z-Scores and Location (cont.)

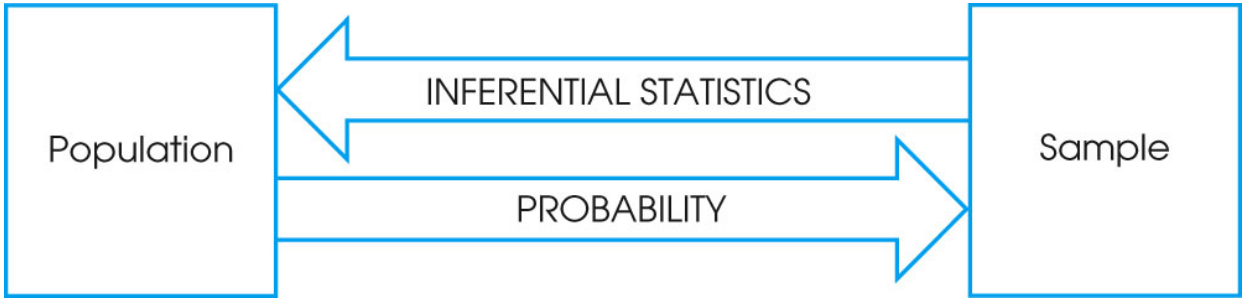
- A score that is located two standard deviations above the mean will have a z-score of +2.00.



z-scores always have a mean of zero and a standard deviation of one

Probability

- Probability: Measuring and quantifying the likelihood of obtaining a specific sample from a specific population.
- We define probability as a fraction or a proportion.
- The probability of any specific outcome is determined by a ratio comparing the **frequency of occurrence** for that outcome relative to the **total number of possible outcomes**.

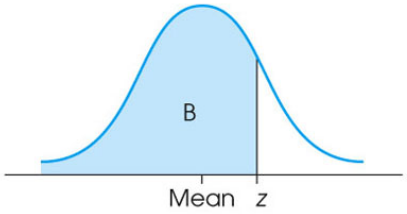


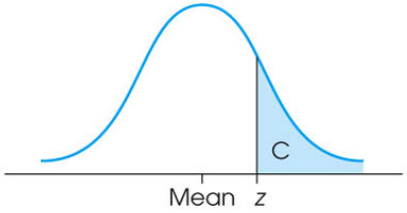
Probability establishes a connection between samples and populations

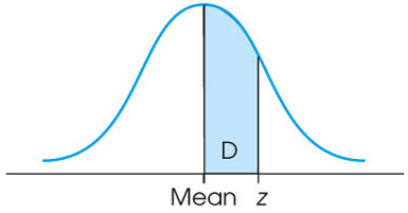
Probability and Normal Distribution

unit normal table

(A) z	(B) Proportion in body	(C) Proportion in tail	(D) Proportion between mean and z
0.00	.5000	.5000	.0000
0.01	.5040	.4960	.0040
0.02	.5080	.4920	.0080
0.03	.5120	.4880	.0120
0.21	.5832	.4168	.0832
0.22	.5871	.4129	.0871
0.23	.5910	.4090	.0910
0.24	.5948	.4052	.0948
0.25	.5987	.4013	.0987
0.26	.6026	.3974	.1026
0.27	.6064	.3936	.1064
0.28	.6103	.3897	.1103
0.29	.6141	.3859	.1141
0.30	.6179	.3821	.1179
0.31	.6217	.3783	.1217
0.32	.6255	.3745	.1255
0.33	.6293	.3707	.1293
0.34	.6331	.3669	.1331

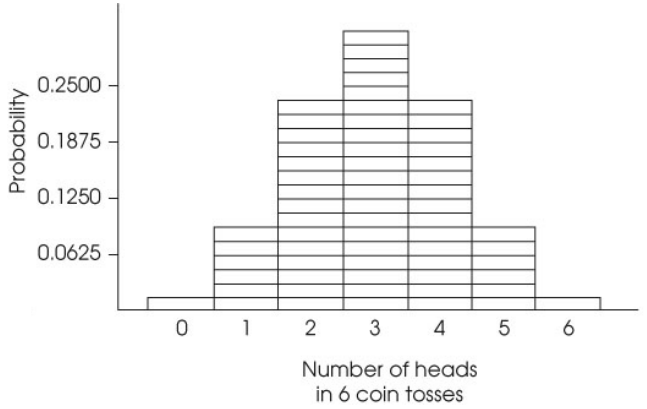






Probability and the Binomial Distribution

- Binomial distributions are formed by a series of observations (for example, 100 coin tosses) for which there are exactly two possible outcomes (heads and tails).
- The two outcomes are identified as A and B, with probabilities of p(A) = p and p(B) = q.
- The distribution shows the probability for each value of X, where X is the number of occurrences of A in a series of n observations.



$n \in \{0, 1, 2, \ldots\}$: number of trials
$p \in [0, 1]$: success probability of each trial
$q = 1 - p$

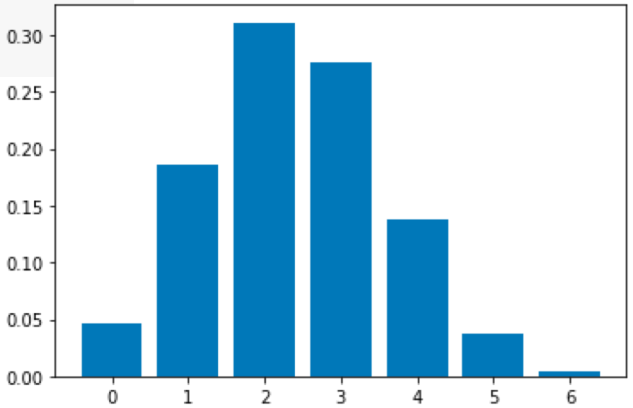
$$P_x = \binom{n}{x} p^x q^{n-x}$$

probability of *x* number of times for a specific outcome within *n* trials

Probability and the Binomial Distribution

What would happen to this curve if p = 0.4? ➔

```
from scipy.stats import binom
import matplotlib.pyplot as plt
n = 6
p = 0.4
# defining list of r values
r_values = list(range(n + 1))
# list of pmf values
dist = [binom.pmf(r, n, p) for r in r_values ]
# plotting the graph
plt.bar(r_values, dist)
plt.show()
```



Some other important distributions: Poisson

For events with an expected separation λ the Poisson distribution $f(k;\lambda)$ describes the probability of k events occurring within the observed interval λ.

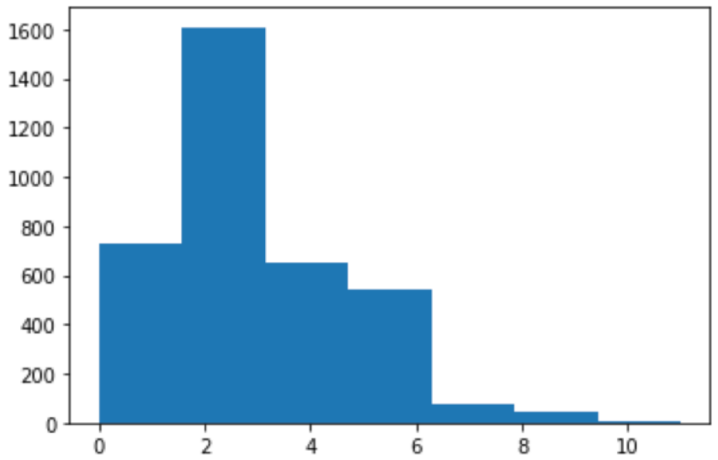
$$f(k;\lambda) = \frac{\lambda^k e^{-\lambda}}{k!}$$

Examples

- Observed frequency of a given term in a corpus.
- Number of visits to a web site in a fixed time interval.
- Number of web site clicks in an hour.

```
s1 = np.random.poisson(3, 365*10)
count, bins, ignored = plt.hist(s1, 7, density=False)
plt.show()
```

Let's say, I typically call my brother once in three days but sometimes I call him two days in a row. If my call frequency is Poisson distributed, this will be the number of days between my phone calls in the following 10 years



Some other important distributions: Poisson

For events with an expected separation λ the Poisson distribution $f(k;\lambda)$ describes the probability of k events occurring within the observed interval λ.

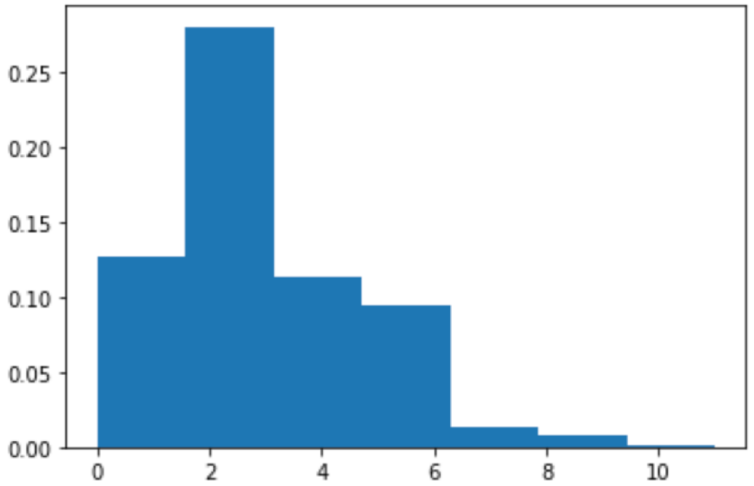
$$f(k;\lambda) = \frac{\lambda^k e^{-\lambda}}{k!}$$

Examples

- Observed frequency of a given term in a corpus.
- Number of visits to a web site in a fixed time interval.
- Number of web site clicks in an hour.

```
count, bins, ignored = plt.hist(s1, 7, density=True)
plt.show()
```

If **density = True,** draw and return a probability density: each bin will display the bin's raw count divided by the total number of counts and the bin width

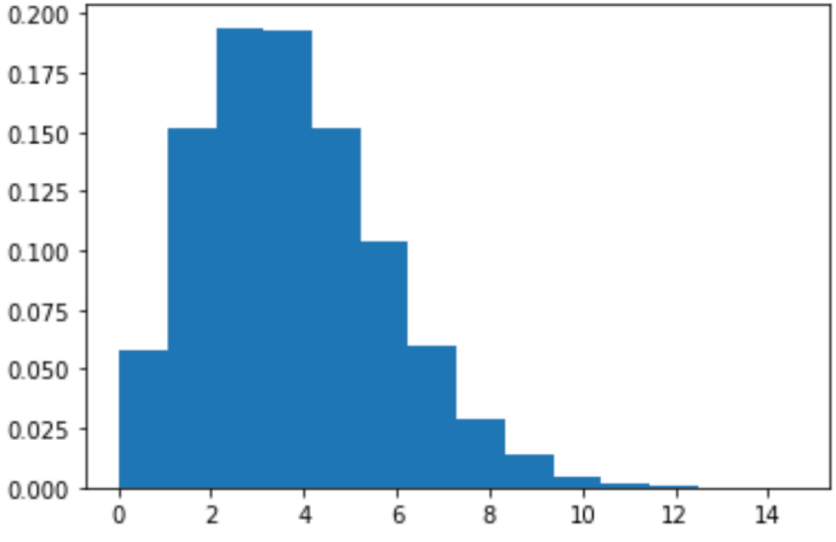


Some other important distributions: Rayleigh

$$P(x; scale) = \frac{x}{scale^2} e^{\frac{-x^2}{2 \cdot scale^2}}$$

```
scale = 3

N = 100000
```

```
s2 = np.random.rayleigh(3, 100000)
count, bins, ignored = plt.hist(s2, 14, density=True)
plt.show()
```



Some other important distributions: Gamma

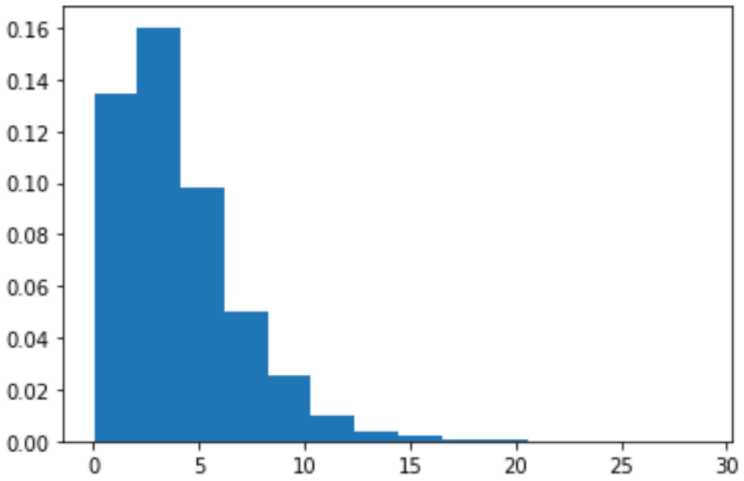
The probability density

$$p(x) = x^{k-1}\frac{e^{-x/\theta}}{\theta^k \Gamma(k)}$$

k: shape

theta: scale

```
s5 = np.random.gamma(2, 2, 10000)
count, bins, ignored = plt.hist(s5, 14, density=True)
plt.show()
```



Example:

Times to failure of electronic components

Some other important distributions: Exponential

The probability density

$$f(x; \frac{1}{\beta}) = \frac{1}{\beta}\exp(-\frac{x}{\beta}),$$

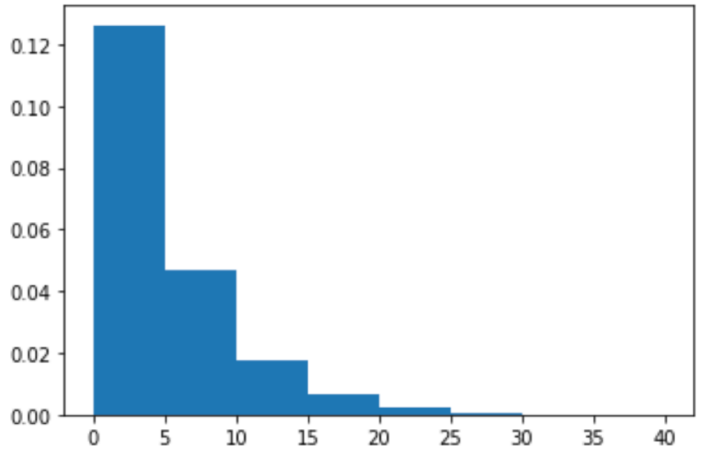
for x > 0 and 0 elsewhere.

$$\lambda = 1/\beta$$

Example: The amount of time (beginning now) until an earthquake occurs has an exponential distribution

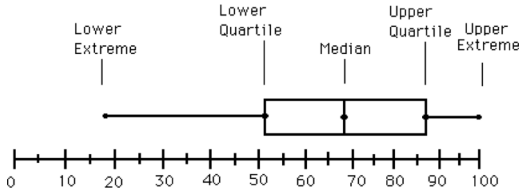
Let's say, typically an earthquake happens once in every 5 years. This graph says that the chance of having no earthquake in 30 years in a row is almost ZERO!

```
s4 = np.random.exponential(5, 1000)
count, bins, ignored = plt.hist(s4, bins=np.arange(0,45,5), density=True)
plt.show()
```



Correcting distributions

Many statistical tools, including mean and variance, t-test, ANOVA etc. **assume data are normally distributed**.

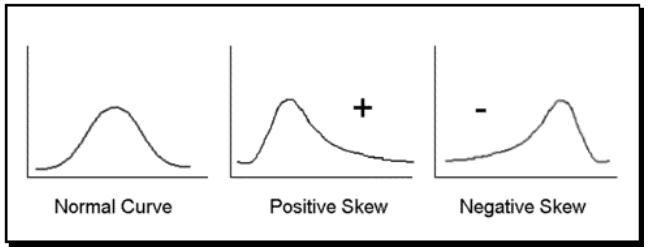


Very often the data is not normally distributed.

Examine boxplot

or histogram (better)

If it's asymmetric, then the data is normal.



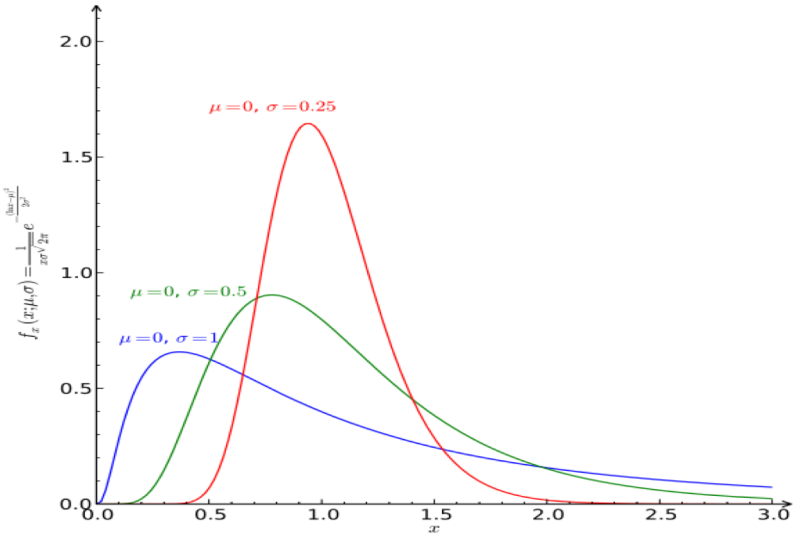
Then, you better correct the distribution

Approach-1: You analyze and understand the distribution of your data before applying any model. You correct the distribution and then apply models!

Approach-2: This is 2022, you let Python do the transformation!

Correcting distributions: Approach-1

- X satisfies a log-normal distribution,
  - Then, Y=log(X) has a normal dist.



- X is Poisson distributed with mean *k* and SD of sqrt(*k*).
  - Then sqrt(X) is approximately normally distributed with SD of 1.

Correcting distributions: Approach-2

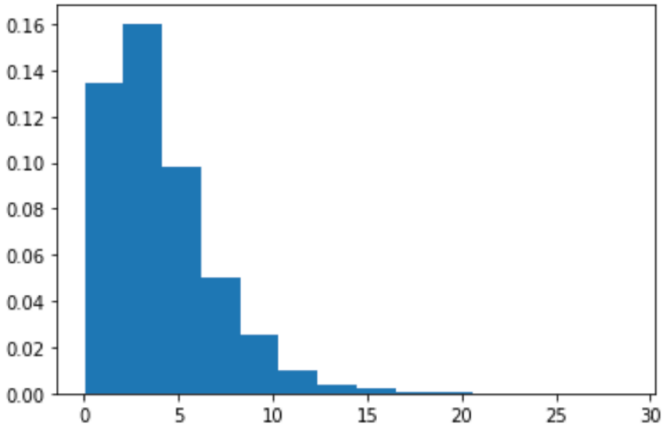
scipy.stats.boxcox

scipy.stats.boxcox(*x*, *lmbda=None*, *alpha=None*, *optimizer=None*)

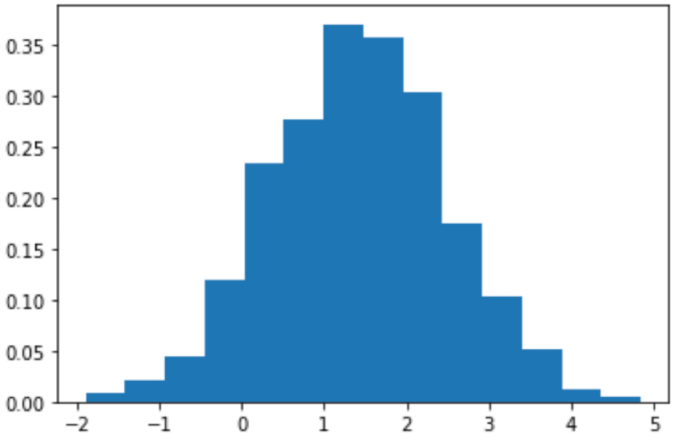
Return a dataset transformed by a Box-Cox power transformation.

IMPORTANT NOTE: boxcox requires the input data to be positive.

```
s5 = np.random.gamma(2, 2, 10000)
count, bins, ignored = plt.hist(s5, 14, density=True)
plt.show()
```



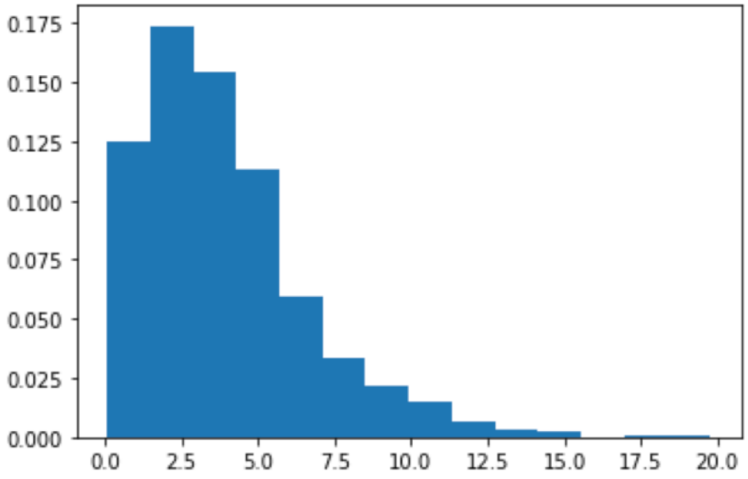

```
s5transformed, lmbd = stats.boxcox(s5)
count, bins, ignored = plt.hist(s5transformed, 14, density=True)
plt.show()
```



Correcting distributions: Approach-2

How do we inverse-Box-Cox?

```
from scipy.special import boxcox, inv_boxcox
s5inverted = inv_boxcox(s5transformed, lmbd)
count, bins, ignored = plt.hist(s5inverted, 14, density=True)
plt.show()
```



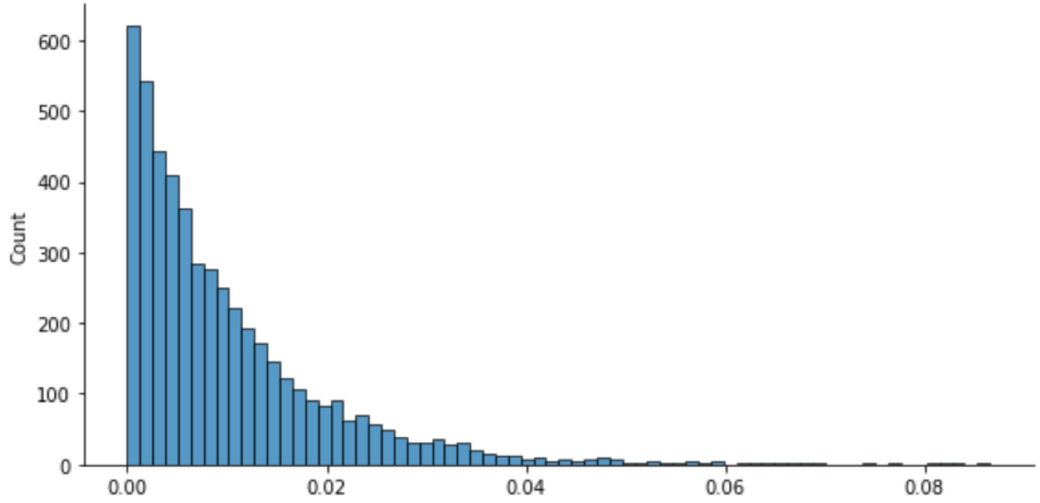
Central Limit Theorem

The Central Limit Theorem

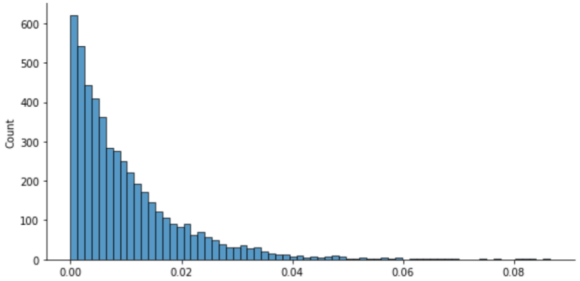
The distribution of the sum (or mean) of a set of M identically-distributed random variables approaches a normal distribution as $M \rightarrow \infty$.

```
Let's say we have a exponentially distributed data (population)
```

```
population = np.random.exponential(0.01, 10000)
plot = sns.displot(x = population , kind='hist', height = 4, aspect = 2)
```



The Central Limit Theorem



- Let's say we randomly chose N samples and we calculate their mean.
- Then we take another randomly chosen group of N samples and we calculate their mean.
- Then we just repeat. Let's say M is the total number of these random group selection and mean calculation.
- According to the CLT, the "means" we have calculated is normally distributed for large M no matter what N is (assuming N is not so small).

CLT in Action

Let's define a function that will create random samples from a distribution.

We can use sample function of pandas that will select random elements without replacement.

a dataframe M N

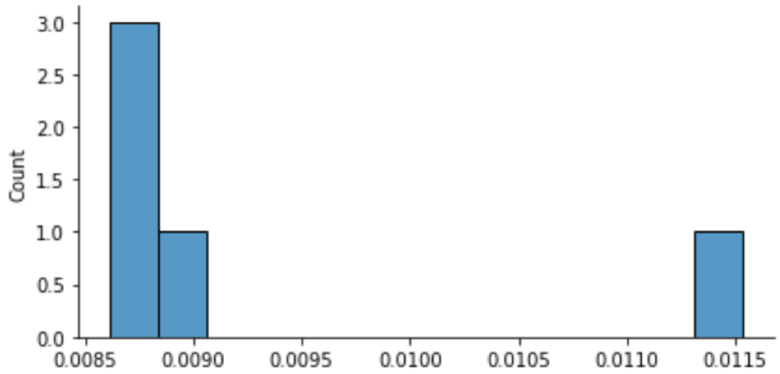
```
def random_samples(population, sample_qty, sample_size):
    sample_means = []
    for i in range(sample_qty):
        sample = population.sample(n=sample_size)
        sample_mean = np.array(sample).mean()
        sample_means.append(sample_mean)
    return sample_means
```

CLT in Action-1

N = 50

M = 5

```
[82] population_dataframe = pd.Series(population)
     samples_from_normal = random_samples(population_dataframe, 5, 50)
     plot = sns.displot(samples_from_normal, kind='hist', height = 3, aspect = 2)
```

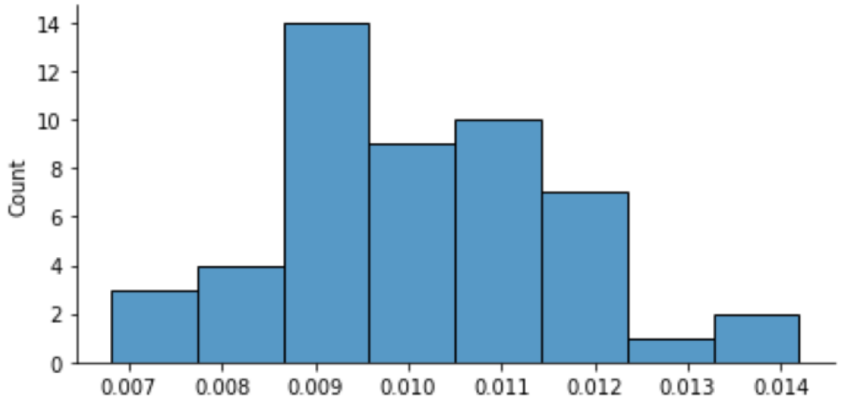


CLT in Action-2

N = 50

M = 50

```
[83] samples_from_normal = random_samples(population_dataframe, 50, 50)
     plot = sns.displot(samples_from_normal, kind='hist', height = 3, aspect = 2)
```

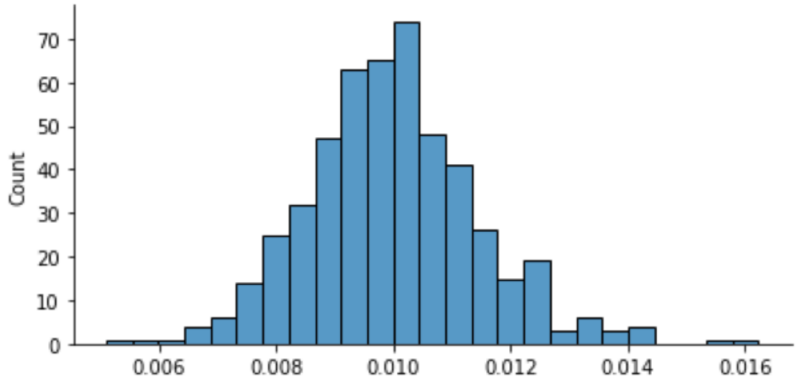


CLT in Action-3

N = 50

M = 500

```
samples_from_normal = random_samples(population_dataframe, 500, 50)
plot = sns.displot(samples_from_normal, kind='hist', height = 3, aspect = 2)
```



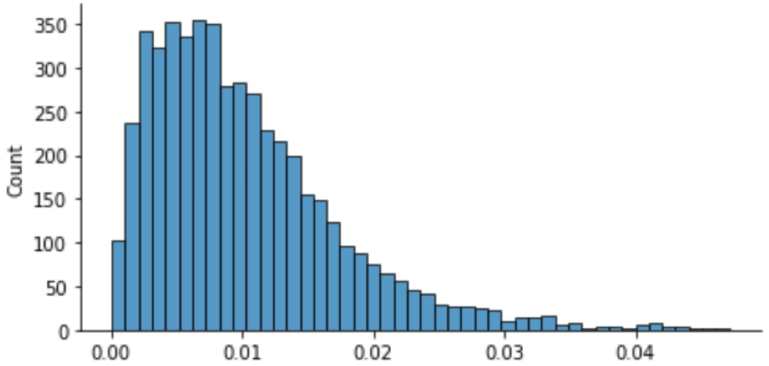
Note that

But if we choose a ridiciously small *N*, then it won't work. For example,

N = 2

M = 5000

```
[86] samples_from_normal = random_samples(population_dataframe, 5000, 2)
     plot = sns.displot(samples_from_normal, kind='hist', height = 3, aspect = 2)
```



The shape of the distribution of sample means tends to be normal. It is guaranteed to be normal if

- either the population from which the samples are obtained is normal,
- or the sample size is n = 30 or more.

The Central Limit Theorem

The distribution of the sum (or mean) of a set of *M* identically-distributed random variables approaches a normal distribution as $M \rightarrow \infty$.

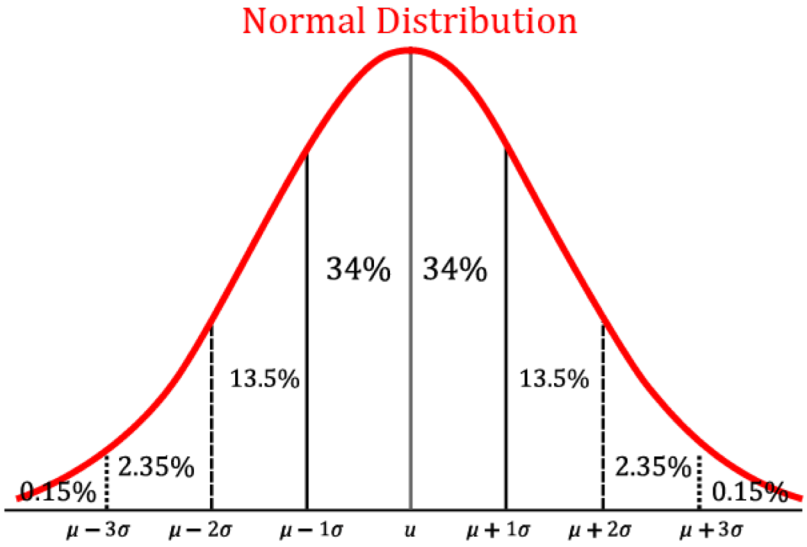
- **The mean** of the distribution of **sample means** is called the **Expected Value** and is always equal to the population mean μ.
- An **individual sample mean** probably will not be identical to its population mean; there will be some "error"
  - Some sample means will be relatively close to μ and others will be relatively far away.
  - The **standard error** provides a measure of the standard distance between the sample mean and population mean μ.

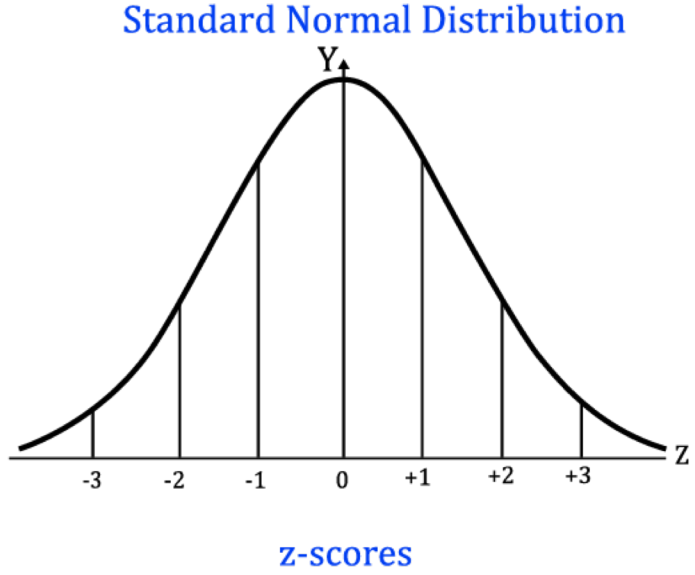
$$SE = \frac{\sigma}{\sqrt{N}}$$

$\frac{\text{sample standard deviation}}{\text{square root of number of samples}}$

z-Scores and Location within the Distribution of Sample Means

- Within the distribution of sample means, the location of each sample mean can be specified by a z-score
- A positive z-score indicates a sample mean that is greater than μ
- The numerical value of the z-score indicates the distance between sample mean and μ measured in terms of the standard error



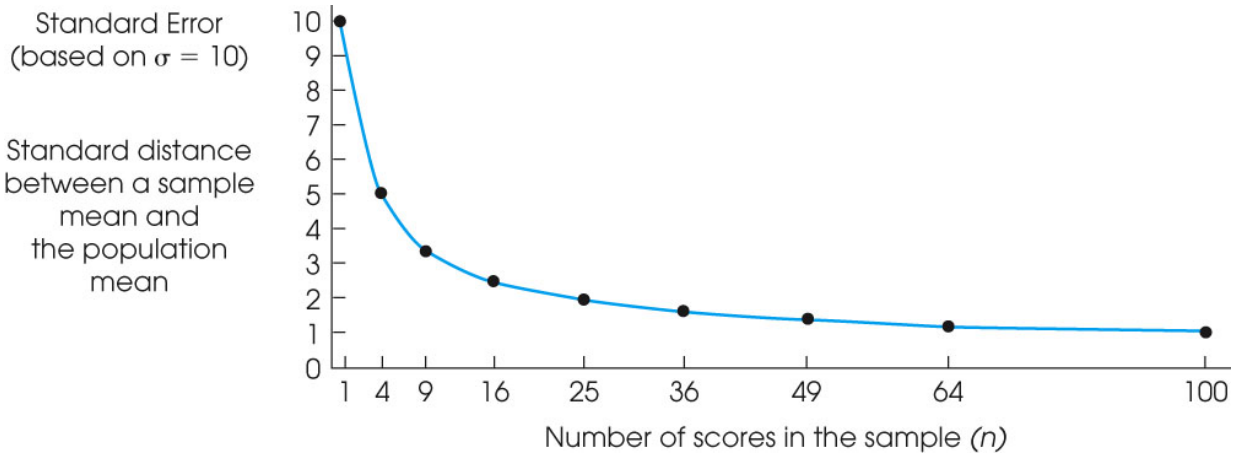


Probability and Sample Means

- Because the distribution of sample means tends to be normal, the z-score value obtained for a sample mean can be used with the unit normal table to obtain probabilities.
- The procedures for computing z-scores and finding probabilities for sample means are essentially the same as we used for individual scores
- However, when you are using sample means, you must remember to consider the sample size (*N*) and compute the standard error before you start any other computations.
- Also, you must be sure that the distribution of sample means satisfies at least one of the criteria for normal shape before you can use the unit normal table.

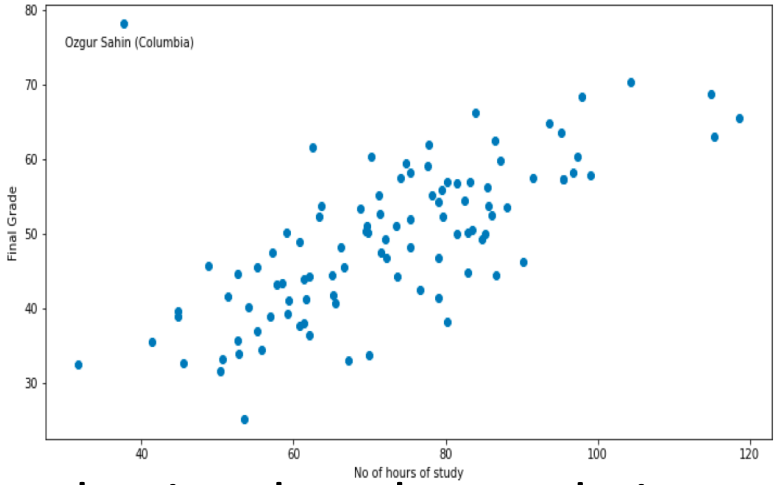
The Standard Error of M

- The magnitude of the standard error is determined by two factors: σ and n.
- The population standard deviation, σ, measures the standard distance between a single score and the population mean.
- Thus, the standard deviation provides a measure of the "error" that is expected for the smallest possible sample, when N = 1.



Outlier Detection with z-Scores

Remember our Grade vs. Number of Hours of Study

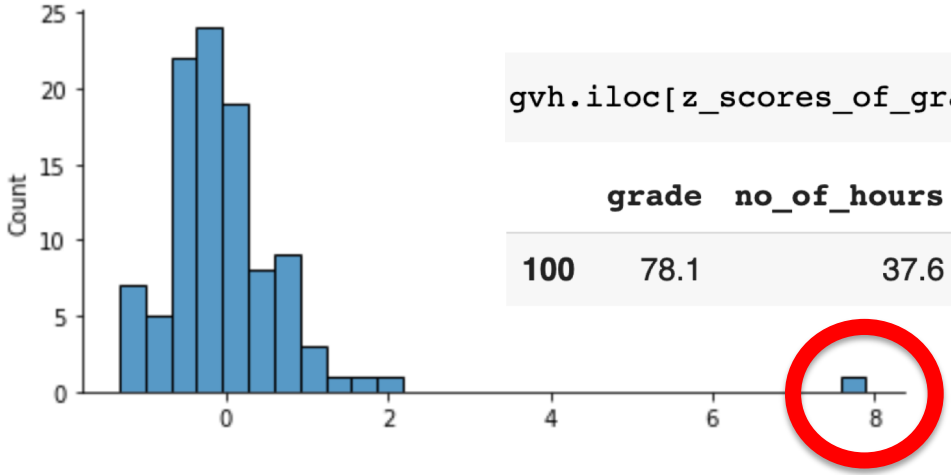


Let's hypothesize that the grade is somehow proportional to the number of study hours

Outlier Detection with z-Scores

We don’t have to plot but let’s compute the z-Scores and plot them

```
grade_normalized_by_hour = gvh.iloc[:,0].values/gvh.iloc[:,1].values
z_scores_of_grade_normalized_by_hour = stats.zscore(grade_normalized_by_hour)
plot = sns.displot(z_scores_of_grade_normalized_by_hour, kind='hist', height = 3, aspect = 2)
```




```
gvh.iloc[z_scores_of_grade_normalized_by_hour>2]
```

	grade	no_of_hours
100	78.1	37.6

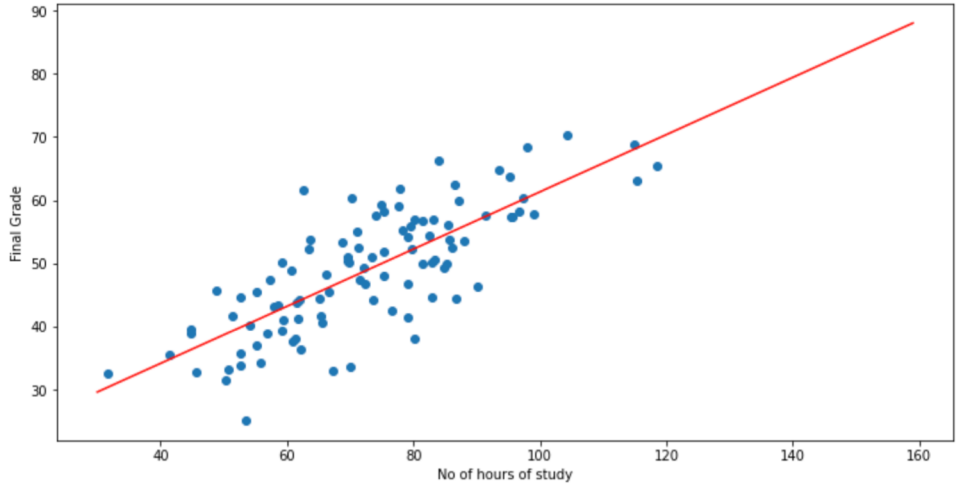
Outlier Detection with z-Scores

```
Lin_Reg_Coefficients = np.polyfit(gvh.no_of_hours, gvh.grade, 1)
Lin_Reg_Coefficients
```

```
array([ 0.4526934 , 16.03166437])
```

```
x = np.arange(30,160)
y_prediction = Lin_Reg_Coefficients[0]*x+Lin_Reg_Coefficients[1]
```

```
fig = plt.figure(figsize=(12,6))
plt.scatter(gvh.no_of_hours, gvh.grade)
plt.plot(x,y_prediction,'r')
plt.xlabel('No of hours of study')
plt.ylabel('Final Grade')
plt.show()
```



What happened to Ozgur Sahin?

COLUMBIA | BIOLOGICAL SCIENCES

About Faculty Ph.D. Program M.A. Biotechnology Undergraduate Courses News Events

Ozgur Sahin

Associate Professor
908 NWC Building, M.C. 4830,
New York, NY, 10027
Email: os2246@columbia.edu
Office Phone: (212) 851-9285
Website: http://www.extremebio.org/

Structure & Biophysics | Sahin Lab Members

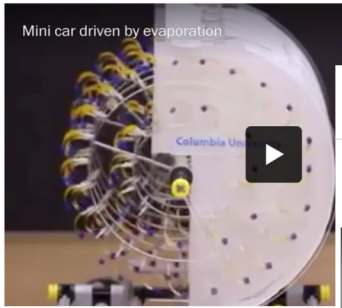
Short Research Description:

Imaging and Modeling Biology at Physical Extremes.

Sections

The Washington Post
Democracy Dies in Darkness

Sahin has shown in previous studies that the spores are capable of harvesting large quantities of energy from their interaction with water. But he realized he'd need to show the energy storage and release in action if he wanted the work to move forward.



WORLD ECONOMIC FORUM

Join us Sign in

Microbe-powered machines | Ozgur Sahin

https://www.youtube.com/watch?v=5zmcItrWJOM