Introduction

In this paper, I'll be presenting an amazing distribution of an interesting problem! Well, we'll be looking into **the distribution of the leading digit** *(first digit from the left)* **of 0!, 1!, 2!, 3!, 4!, ... and up to (n - 1)!, where** $n \geq 1$ **is any integer of your choosing.**
For example, for $\mathbf{n = 16}$ we have:

- $0! = \mathbf{1}$
- $1! = \mathbf{1}$
- $2! = 2 \times 1 = \mathbf{2}$
- $3! = 3 \times 2 \times 1 = \mathbf{6}$
- $4! = 4 \times 3 \times 2 \times 1 = \mathbf{2}4$
- $5! = 5 \times 4 \times 3 \times 2 \times 1 = \mathbf{1}20$
- $6! = 6 \times 5 \times 4 \times 3 \times 2 \times 1 = \mathbf{7}20$
- $7! = 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = \mathbf{5},040$
- $8! = 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = \mathbf{4}0,320$
- $9! = 9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = \mathbf{3}62,880$
- $10! = 10 \times 9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = \mathbf{3},628,800$
- $11! = 11 \times 10 \times 9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = \mathbf{3}9,916,800$
- $12! = 12 \times 11 \times 10 \times 9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = \mathbf{4}79,001,600$
- $13! = 13 \times 12 \times 11 \times 10 \times 9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = \mathbf{6},227,020,800$
- $14! = 14 \times 13 \times 12 \times 11 \times 10 \times 9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = \mathbf{8}7,178,291,200$
- $15! = 15 \times 14 \times 13 \times 12 \times 11 \times 10 \times 9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = \mathbf{1},307,674,368,000$

and the following table represents how many times each digit shows up as the leading one:

leading digit	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
frequency	4	2	3	2	1	2	1	1	0

Now this is exactly what we're going to do, only for much bigger values of n
(n = 10000, n = 20000 and n = 50000), and for each case, we'll visualise and examine this unexpected distribution, **which turns out to be a famous one!**
Furthermore, we will apply and verify different statistical procedures in order to make significant inferences and conclusions.

Important note

This document contains code segments written in *Python*, so feel free to copy, use and modify them, as it will help you to better understand and grasp the provided solutions and plots. However, you should know that a simple copy/paste from this PDF file won't do the trick, instead you'll find all code segments in this GitHub repository: https://github.com/sam-chisholm/the-leading-digit-of-factorials

Part 1: examining our population

Well first of all, **we are really lucky,** because in *Python* it's possible to calculate 100000!. To prove this to you, try executing the following code:

```
factorial = 1
for i in range(1, 100001):
    factorial *= i

# this print statement will print 100,000! which is a huge number!!
print(f"{factorial:,}")
```

So calculating factorials is not a problem anymore, now in order to get the leading digit of all computed values, well we simply convert them into strings and get the first element, so to wrap up, here's the actual code for computing the frequencies of the population:

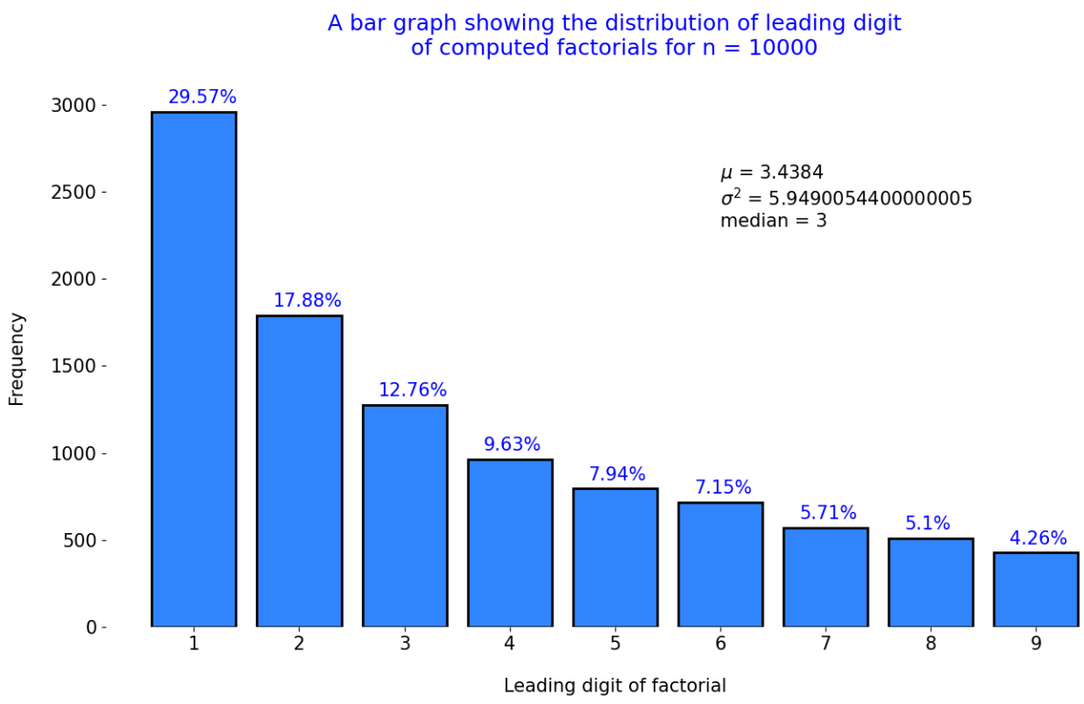
```
# this dictionary stores the frequency of each digit
# note that initially digits_freq["1"] = 1 since 0! = 1
digits_freq = {"1": 1, "2": 0, "3": 0, "4": 0, "5": 0, "6": 0, "7": 0, "8": 0, "9": 0}

# you could change the value of n depending on your choice
# just note that as n gets bigger, your code takes a lot of time for execution!
n = 16

factorial = 1
for x in range(1, n):
    factorial *= x

    # after computing factorial, we convert it into a string and get its first element
    # then we use it to access the dictionary and increment the corresponding value
    digits_freq[str(factorial)[0]] += 1

# here n = 16, so the printed dictionary will be exactly the table in the introduction
print(digits_freq)
```

In addition, and for large values of n we want to calculate our population's mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\sigma^2}$ and **the median** as well, and of course we want to visualise and plot our population distribution, to achieve all of this try executing the following code:

```
# make sure to install the following libraries before importing them!
import matplotlib.pyplot as plt
import numpy as np

digits_freq = {"1": 1, "2": 0, "3": 0, "4": 0, "5": 0, "6": 0, "7": 0, "8": 0, "9": 0}

# don't forget to change the value of n here later to 20000 !!!
n = 10000

factorial = 1
for x in range(1, n):
    factorial *= x

    digits_freq[str(factorial)[0]] += 1

# now we simply create 2 "arrays" to compute all parameters needed
# and to plot our population distribution
leading_digit = np.arange(1, 10)
frequency_values = np.array(list(digits_freq.values()))

# now to compute the mean we first multiply the 2 previous "arrays" element by element
# then we sum all values of resulting "array" and divide by n
# since the total values computed from 0!, 1!, ... and up to (n - 1)! is exactly n
mean = np.sum(leading_digit * frequency_values) / n

# now to calculate the variance we subtract the mean from leading_digit values
# then we square each element of resulting "array"
# then we multiply by frequency_values, sum all values of resulting "array"
# and finally divide by n
variance = np.sum(frequency_values * np.square(leading_digit - mean)) / n

# since we are interested in n = 10000, 20000, and 50000 which are all even
# the median will be the average of the 2 values at positions n / 2 and (n + 2) / 2
# this is not true if n is odd
cumulative_freq = median = 0
for i, freq in enumerate(frequency_values, 1):
    cumulative_freq += freq
    if cumulative_freq == n / 2:
```

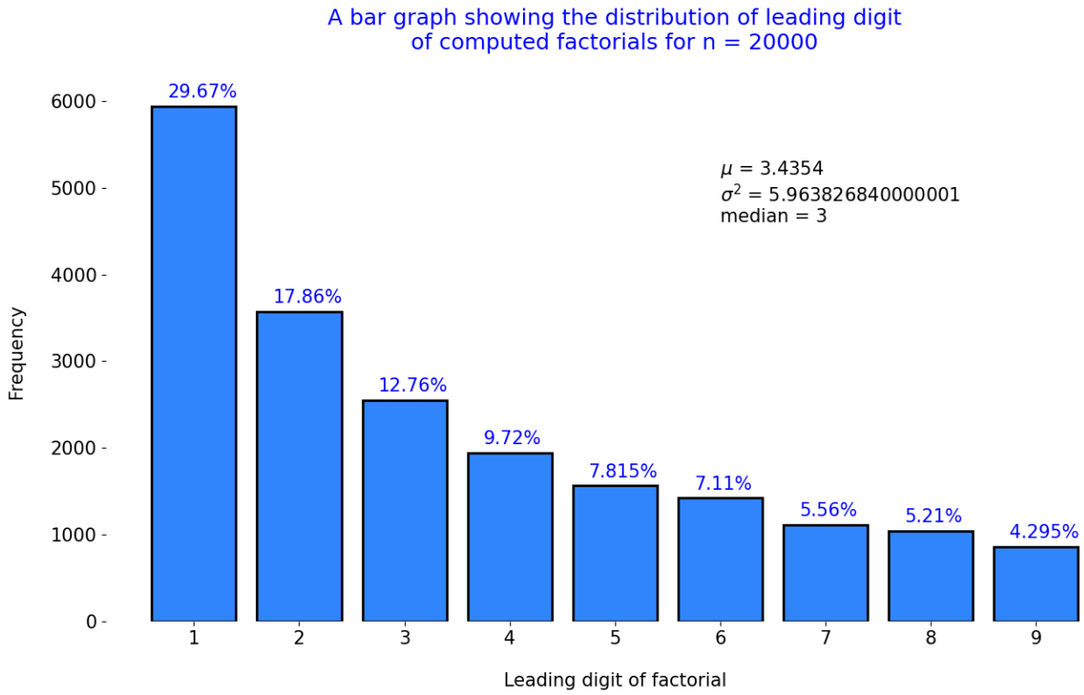
```
            median = (i + i + 1) / 2
            break
        elif cumulative_freq > n / 2:
            median = i
            break

# Set general font size of the graph
plt.rcParams.update({"font.size": 15})
plt.figure(figsize=(15, 8))
plt.box(False)

plt.bar(leading_digit, frequency_values, color="#3185fc", edgecolor="black", linewidth=2)
plt.xticks(leading_digit)

# setting the x and y label of the graph
plt.xlabel("\nLeading digit of factorial")
plt.ylabel("Frequency\n")

# setting the plot's title
title = "A bar graph showing the distribution of leading digit\n"
title += "of computed factorials for n = " + str(n) + "\n"
plt.title(title, color="blue")

# first we fill text with the values of all computed parameters
text = "$\\mu $ = " + str(mean)
text += "\n$\\sigma^2$ = " + str(variance)
text += "\nmedian = " + str(median)

# then we display it on the graph
plt.text(6, 0.23 * n, text)

# here we calculate the percentage of each digit then we display it on top of each bar
for i, freq in enumerate(frequency_values, 1):
    plt.text(i - 0.25, freq + 0.005 * n, str(freq * 100 / n) + "%", color="blue")

plt.show()
```

After executing this code segment *(which takes on my computer 1 minute and 12 seconds to finish execution)* we get the following graph:



Interesting, right? I mean, normally we would expect for bigger values of n that **our distribution will be almost uniform,** in other words, all digits have **roughly the same frequency,** but it's definitely not the case here, maybe n = 10000 isn't big enough, so let's try the previous code

segment on n = 20000 *(which takes on my computer 11 minutes to finish execution)* and see the results:



Again, even for n = 20000 we still get the same previous distribution and almost the same percentage values. Well, turns out that this distribution is quite famous under the name of Benford's law named after physicist Frank Benford, and it was actually applied and tested on many data sets like lengths of rivers **(measured in any unit you want!)**, street addresses, population numbers, physical constants, stock prices and many more sets. In addition, this law is highly used to detect fraud and verify the integrity of collected data. However, Benford's law only works **when data values are distributed across multiple orders of magnitude,** in other words, our values should spread out on a very very large range *(which is the case of our population of factorials when n is very big!)* **and it definitely shouldn't be restricted in a small one,** so for example, Benford's law can't be applied on people's height in meters, **since human heights are between 1 and 3 meters.**

Furthermore, once a data set is satisfying Benford's law, we can easily compute the relative frequency of each leading digit f_i where $i \in \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ with the following formula:

$$\boldsymbol{f_i} = \frac{\text{frequency of digit i}}{\text{total observed values}} = \boldsymbol{\log_{10}\left(1 + \frac{1}{i}\right)}$$

therefore we could actually calculate the following parameters for that data set and here's how:

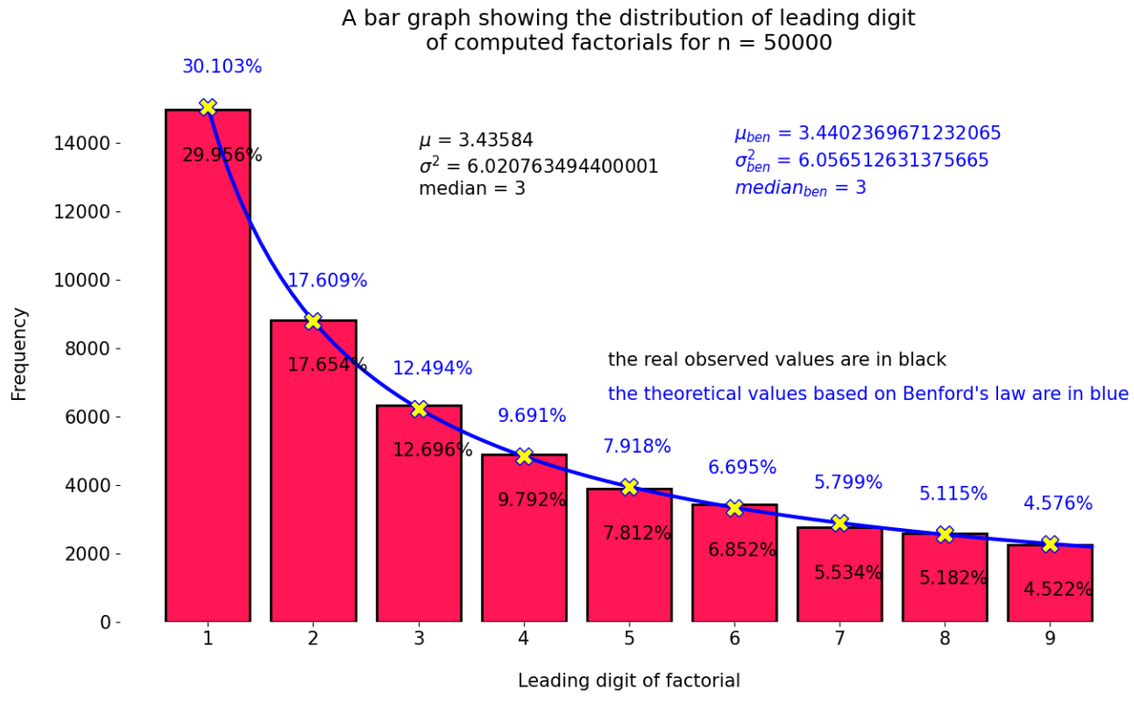
$$\begin{aligned}
\mu_{ben} &= \sum_{i=1}^{9} i \times f_i = \sum_{i=1}^{9} i \times \log_{10}\left(1 + \frac{1}{i}\right) = \sum_{i=1}^{9} \log_{10}\left(\left(1 + \frac{1}{i}\right)^i\right) = \log_{10}\left(\prod_{i=1}^{9} \frac{(i+1)^i}{i^i}\right) \\
&= \log_{10}\left(\frac{2^1}{1^1} \times \frac{3^2}{2^2} \times \frac{4^3}{3^3} \times \frac{5^4}{4^4} \times \frac{6^5}{5^5} \times \frac{7^6}{6^6} \times \frac{8^7}{7^7} \times \frac{9^8}{8^8} \times \frac{10^9}{9^9}\right) \\
&= \log_{10}\left(\frac{10^9}{1 \times 2 \times 3 \times 4 \times 5 \times 6 \times 7 \times 8 \times 9}\right) = 9 - \log_{10}(9!) \approx 3.44 \\
\sigma^2_{ben} &= \sum_{i=1}^{9} (i - \mu_{ben})^2 \times f_i = \sum_{i=1}^{9} (i - \mu_{ben})^2 \times \log_{10}\left(1 + \frac{1}{i}\right) \quad \approx 6.06 \\
median_{ben} &= 3 \text{ since } \log_{10}\left(1 + \frac{1}{1}\right) + \log_{10}\left(1 + \frac{1}{2}\right) \quad \approx 0.48 < 0.5 \\
&\qquad \text{and } \log_{10}\left(1 + \frac{1}{1}\right) + \log_{10}\left(1 + \frac{1}{2}\right) + \log_{10}\left(1 + \frac{1}{3}\right) \quad \approx 0.60 > 0.5
\end{aligned}$$

So to conclude Part 1, we're actually going to plot our distribution for n = 50000, and see how well does it satisfy Benford's law, and we're going to compare all observed values to the theoretical ones based on the same law, and here's the entire code segment to achieve all of this:

```
# make sure to install the following libraries before importing them!
import matplotlib.pyplot as plt
import numpy as np

digits_freq = {"1": 1, "2": 0, "3": 0, "4": 0, "5": 0, "6": 0, "7": 0, "8": 0, "9": 0}

# I' m only testing this code segment for n = 50000
# but, you could use it as well for the 2 previous values of n
n = 50000

factorial = 1
for x in range(1, n):
    factorial *= x

    digits_freq[str(factorial)[0]] += 1

# now we simply create 2 "arrays" to compute all parameters needed
# and to plot our population distribution
leading_digit = np.arange(1, 10)
frequency_values = np.array(list(digits_freq.values()))

# here we compute the real observed parameters of the population
# in the same way we did in the previous code segment
mean = np.sum(leading_digit * frequency_values) / n
variance = np.sum(frequency_values * np.square(leading_digit - mean)) / n

cumulative_freq = median = 0
for i, freq in enumerate(frequency_values, 1):
    cumulative_freq += freq
    if cumulative_freq == n / 2:
        median = (i + i + 1) / 2
        break
    elif cumulative_freq > n / 2:
        median = i
        break

# now we calculate the theoretical parameters based on Benford's law
# the theoretical median = 3
benford_freq = np.log10(1 + 1 / leading_digit)
benford_mean = np.sum(benford_freq * leading_digit)
benford_variance = np.sum(benford_freq * np.square(leading_digit - benford_mean))

# Set general font size of the graph
plt.rcParams.update({"font.size": 15})
plt.figure(figsize=(15, 8))
plt.box(False)

plt.bar(leading_digit, frequency_values, color="#ff1654", edgecolor="black", linewidth=2)

# setting the x-coordinates of the curve
x = np.arange(1, 9.5, 0.1)
# setting the corresponding y-coordinates
y = np.log10(1 + 1 / x) * n

# plotting Benford's curve
# note that the yellow X in Benford's curve corresponds to leading_digit values
plt.plot(x, y, color="blue", linewidth=3, markevery=(leading_digit - 1) * 10, marker="X",
         markerfacecolor="yellow", markersize=15)

# if you don't use this line of code, not all digits will be printed on the x-axis
plt.xticks(leading_digit)
```

```
# setting the x and y label of the graph
plt.xlabel("\nLeading digit of factorial")
plt.ylabel("Frequency\n")

# setting the plot's title
title = "A bar graph showing the distribution of leading digit\n"
title += "of computed factorials for n = " + str(n) + "\n"
plt.title(title)

# first we fill text with the values of all calculated parameters
text = "$\\mu$ = " + str(mean)
text += "\n$\\sigma^2$ = " + str(variance)
text += "\nmedian = " + str(median)
# then we display it on the graph
plt.text(3, 0.25 * n, text)

# now we fill text1 with the theoretical values based on Benford's law
text1 = "$\\mu_{ben}$ = " + str(benford_mean)
text1 += "\n$\\sigma_{ben}^2$ = " + str(benford_variance)
text1 += "\n$median_{ben}$ = 3"
# then we display it on the graph
plt.text(6, 0.25 * n, text1, color="blue")

# here we calculate the percentage of each digit then we display it inside each bar
for i, freq in enumerate(frequency_values, 1):
    plt.text(i - 0.25, freq - 0.03 * n, str(freq * 100 / n) + "%")

# here we display the theoretical percentages based on Benford's law
for i, freq in enumerate(benford_freq, 1):
    plt.text(i - 0.25, (freq + 0.02) * n, str(round(freq * 100, 3)) + "%", color="blue")

plt.text(4.8, 0.15 * n, "the real observed values are in black")
plt.text(4.8, 0.13 * n, "the theoretical values based on Benford's law are in blue",
         color="blue")
plt.show()
```

And after executing this code segment *(which takes on my computer almost 4 hours and 15 minutes to finish execution)* we get the following graph:



So with that we conclude Part 1 *(by the way, I'll leave YOU to judge if our population does satisfy Benford's law)* and as promised, in the next and final Part, we'll be testing and verifying some well known statistical theorems and plotting some well known distributions as well.

Part 2: samples, samples and even more samples!

Before we go any further, let me explain basically what we'll do, first we're gonna consider our population to be the leading digit of factorials from 0!, 1!, ... and up to (10000 - 1)! *(the first population we've plotted in Part 1)* which can be summarized in the following table:

leading digit	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
frequency	2957	1788	1276	963	794	715	571	510	426

this table could be easily found by executing the second code snippet in Part 1
frequencies_of_population.py

then we're gonna draw **n random values** *(a single sample of size n)* out of this population and compute a certain **statistic,** again we'll draw another n random values and calculate that same statistic, and we'll keep repeating that many many times and finally study and plot the resulting distribution of the chosen statistic. Luckily in *Python*, this is pretty straightforward to achieve and here's how:

```
import numpy as np


# first, we create 2 arrays to store our population
leading_digit = np.arange(1, 10)
relative_frequencies = [.2957, .1788, .1276, .0963, .0794, .0715, .0571, .0510, .0426]

# this line of code is made just to make sure that we get the same samples
np.random.seed(0)

# here we chose the value of n
# which is the number of random values drawn in each time

n = 10

# here we'll be drawing 5 samples, each sample of size n
# and for each sample, we're calculating some basic statistics
for i in range(1, 6):
    sample = np.random.choice(leading_digit,size=n,replace=True,p=relative_frequencies)
    print("sample " + str(i) + ":", sample, np.mean(sample),
          np.median(sample), np.var(sample, ddof=1), sep="\t\t")
```

And after executing this code, you'll always get these results:
(note that $\overline{x}$ is the sample mean and s^2 is the sample variance)

		$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$	median	$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2$
sample 1:	[3 5 4 3 2 4 2 7 9 2]	4.1	3.5	5.433333333333334
sample 2:	[6 3 3 8 1 1 1 6 6 7]	4.2	4.5	7.28888888888889
sample 3:	[9 6 2 6 1 4 1 8 3 2]	4.2	3.5	8.399999999999999
sample 4:	[1 5 2 3 1 4 4 4 8 4]	3.6	4.0	4.2666666666666675
sample 5:	[2 2 4 1 4 4 1 1 2 2]	2.3	2.0	1.5666666666666667

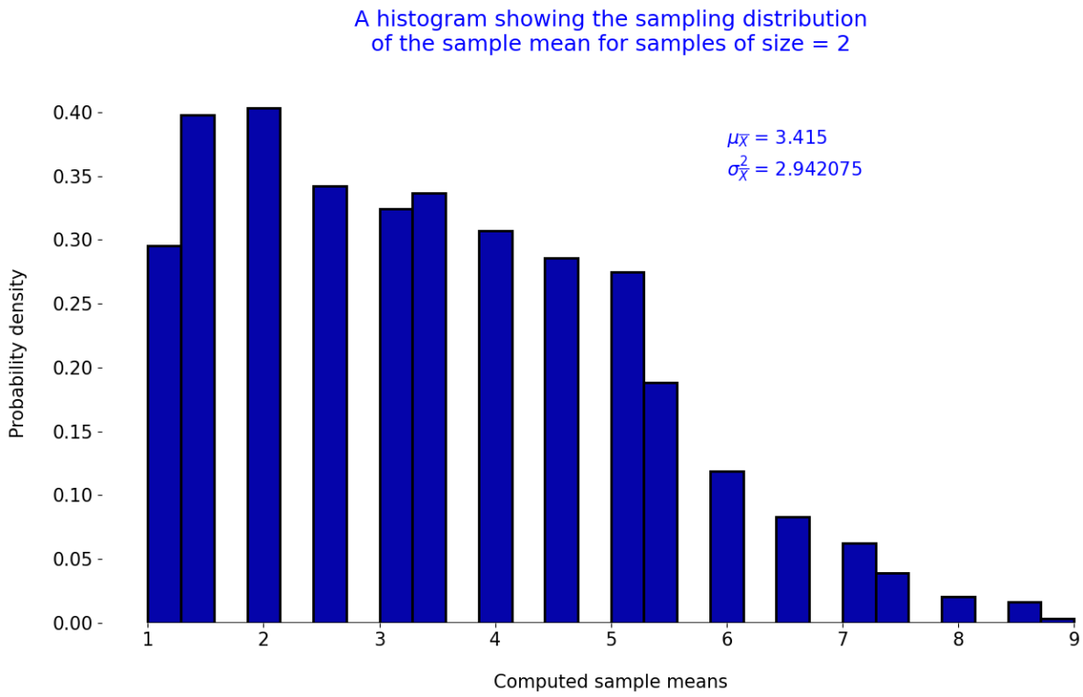
Now this is exactly what we'll do, but we'll be studying and visualising each statistic one at a time, so let's get started!

1) The sampling distribution of the sample mean

In this section, we'll basically be drawing many many samples of size n, for each sample we'll compute its mean and store it, and finally we'll study and plot **the sampling distribution of the sample mean** which we'll call $\overline{\mathbf{X}}$, now we'll be studying 2 cases, when n = 2 and when n = 100 and see what happens, and here's the code segment for n = 2:

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

# first, we create 2 arrays to store our population
leading_digit = np.arange(1, 10)
relative_frequencies = [.2957, .1788, .1276, .0963, .0794, .0715, .0571, .0510, .0426]

# this line of code is made just to make sure that we get the same samples
np.random.seed(0)

# we execute this for n = 2
n = 2

# here we'll be drawing 5000 samples, each sample of size n
# and for each sample, we're calculating its mean
# note that we're storing the sample means in this array
sample_means = np.empty(5000)
for i in range(5000):
    sample = np.random.choice(leading_digit,size=n,replace=True,p=relative_frequencies)

    sample_means[i] = np.mean(sample)

# here we compute the mean and variance of resulting distribution
sampling_distribution_mean = np.mean(sample_means)
sampling_distribution_variance = np.var(sample_means)

# Set general font size of the graph
plt.rcParams.update({"font.size": 15})

# Set width and height of the plot's window
plt.figure(figsize=(15, 8))
plt.box(False)

# now we simply plot the histogram that represents
# the sampling distribution of the sample mean
# note that density=True means that the sum of
# area under histogram equals 1
plt.hist(sample_means, bins="auto", density=True, color="#0504aa",
         edgecolor="black", linewidth=2)

# setting the x label of the graph
plt.xlabel("\nComputed sample means")

# setting the y label of the graph
plt.ylabel("Probability density\n")

# setting the plot's title
title = "A histogram showing the sampling distribution\n"
title += "of the sample mean for samples of size = " + str(n) + "\n"
plt.title(title, color="blue")

# first we fill text with the values of all calculated parameters
text = "$\\mu_{\\overline{X}}$ = " + str(sampling_distribution_mean)
text += "\n$\\sigma^2_{\\overline{X}}$ = " + str(sampling_distribution_variance)

# then we display it on the graph
plt.text(6, 0.35, text, color="blue")

plt.show()
```

and after execution we get the following histogram:



Now there are a couple of interesting things to observe in this plot. Firstly, notice that all computed sample means are between 1 and 9, this is quite obvious since the population which we're sampling from is **digits from 1 to 9 (a discrete distribution)**, thus $\overline{\mathbf{X}}$ will take **its lowest value 1** when our drawn sample for n = 2 is [1, 1], and $\overline{\mathbf{X}}$ will take **its highest value 9** when our drawn sample is [9, 9], so in general and for any value n, we have $1 \leq \overline{\mathbf{X}} \leq 9$.

Secondly, you might be wondering now about the reason those blank spaces exist in the histogram. Well, since again the population distribution is discrete, then $\overline{\mathbf{X}}$ is definitely **discrete** as well, matter of fact, let's try to find and list all possible values of $\overline{\mathbf{X}}$ when n = 2. To do that, we should think about all possible drawn samples, specifically, the total number of samples is $9 \times 9 = 81$, although when evaluating their mean we **don't care about the order**, for example the samples [2, 8] and [8, 2] have the same sample mean, so the total number is reduced to:

$$\binom{9+2-1}{2} = \frac{10!}{2! \times 8!} = 45 \qquad \textit{(remember repetition is allowed!)}$$

Now let's list them all here and see all possible values of $\overline{\mathbf{X}}$:

sample	$\overline{x}$	sample	$\overline{x}$	sample	$\overline{x}$	sample	$\overline{x}$	sample	$\overline{x}$
[1, 1]	1	[1, 2]	1.5	[1, 3]	2	[1, 4]	2.5	[1, 5]	3
[1, 6]	3.5	[1, 7]	4	[1, 8]	4.5	[1, 9]	5	[2, 2]	2
[2, 3]	2.5	[2, 4]	3	[2, 5]	3.5	[2, 6]	4	[2, 7]	4.5
[2, 8]	5	[2, 9]	5.5	[3, 3]	3	[3, 4]	3.5	[3, 5]	4
[3, 6]	4.5	[3, 7]	5	[3, 8]	5.5	[3, 9]	6	[4, 4]	4
[4, 5]	4.5	[4, 6]	5	[4, 7]	5.5	[4, 8]	6	[4, 9]	6.5
[5, 5]	5	[5, 6]	5.5	[5, 7]	6	[5, 8]	6.5	[5, 9]	7
[6, 6]	6	[6, 7]	6.5	[6, 8]	7	[6, 9]	7.5	[7, 7]	7
[7, 8]	7.5	[7, 9]	8	[8, 8]	8	[8, 9]	8.5	[9, 9]	9

Now I know I know, this is starting to get really *mathy*, and unfortunately we're not done yet with formulas, although what we did so far is actually really important, not only do we have insights now on the behavior of the sampling distribution, but we could also improve our initial

approach with an enhanced algorithm that stores the sample means in a dictionary where indices are the blue values in the previous table, instead of creating a huge array of 5000 elements, thus minimizing memory usage, furthermore what we did here for n = 2 could actually be generalized, specifically for any given value n, we have $\overline{\mathbf{X}}$ is always a discrete distribution and its sample space is the following set:

$$\left\{\frac{n}{n}, \frac{n+1}{n}, \frac{n+2}{n}, \frac{n+3}{n}, ..., \frac{9n-2}{n}, \frac{9n-1}{n}, \frac{9n}{n}\right\}$$

Thirdly, notice that if we compare $\mu_{\overline{X}}$ and $\sigma^2_{\overline{X}}$ to the population parameters $\mu = 3.4384$ and $\sigma^2 = 5.9490054400000005$ which we computed in the previous part we get:

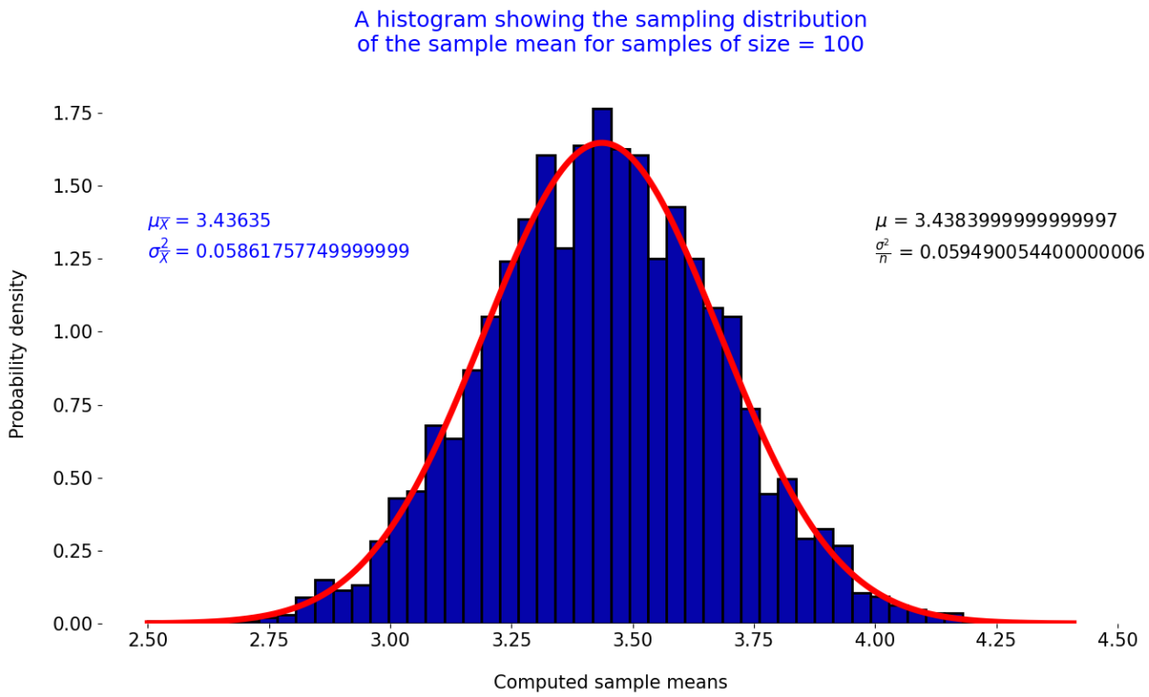
$$\boldsymbol{\mu_{\overline{X}} \approx \mu}$$
$$\boldsymbol{\sigma^2_{\overline{X}} \approx \frac{\sigma^2}{n} \approx \frac{\sigma^2}{2}}$$

(Note that the variance formula is true only because our random values are independent from each other!)

Lastly, we know that for large sample sizes (usually when $n \geq 30$) the sampling distribution of the sample mean $\overline{X}$ **will be approximately normal** thanks to the **Central Limit Theorem,** one of the most fundamental concepts in statistics, and to prove this to you try executing the following code segment for n = 100 and see the resulting plot:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

# first, we create 2 arrays to store our population
leading_digit = np.arange(1, 10)
relative_frequencies = [.2957, .1788, .1276, .0963, .0794, .0715, .0571, .0510, .0426]

# here we compute the population parameters
population_mean = np.sum(leading_digit * relative_frequencies)
population_variance = np.sum(relative_frequencies*(leading_digit-population_mean)** 2)

# this line of code is made just to make sure that we get the same samples
np.random.seed(0)

# we execute this for n = 100
n = 100

# here we'll be drawing 5000 samples, each sample of size n
# and for each sample, we're calculating its mean
# note that we're storing the sample means in this array
sample_means = np.empty(5000)
for i in range(5000):
    sample = np.random.choice(leading_digit,size=n,replace=True,p=relative_frequencies)

    sample_means[i] = np.mean(sample)

# here we compute the mean and variance of resulting distribution
sampling_distribution_mean = np.mean(sample_means)
sampling_distribution_variance = np.var(sample_means)

# Set general font size of the graph
plt.rcParams.update({"font.size": 15})

# Set width and height of the plot's window
plt.figure(figsize=(15, 8))
plt.box(False)

# now we simply plot the histogram that represents
# the sampling distribution of the sample mean
plt.hist(sample_means, bins="auto", density=True, color="#0504aa",
         edgecolor="black", linewidth=2)
```

```
# In the same time we draw the normal curve on top of the histogram
x = np.arange(np.amin(sample_means), np.amax(sample_means), 0.001)
plt.plot(x, norm.pdf(x, sampling_distribution_mean,
         np.sqrt(sampling_distribution_variance)),color="red", linewidth=5)

# setting the x label of the graph
plt.xlabel("\nComputed sample means")

# setting the y label of the graph
plt.ylabel("Probability density\n")

# setting the plot's title
title = "A histogram showing the sampling distribution\n"
title += "of the sample mean for samples of size = " + str(n) + "\n"
plt.title(title, color="blue")

# first we fill text with the values of all calculated parameters
text = "$\\mu_{\\overline{X}}$ = " + str(sampling_distribution_mean)
text += "\n$\\sigma^2_{\\overline{X}}$ = " + str(sampling_distribution_variance)

# then we display it on the graph
plt.text(2.5, 1.25, text, color="blue")

# then we fill text again with the theoretical values
text = "$\\mu$ = " + str(population_mean)
text += "\n$\\frac{\\sigma^2}{n}$ = " + str(population_variance / n)

# then we display it on the graph
plt.text(4, 1.25, text)

plt.show()
```



With that we conclude this section, but before moving to the next one, I encourage you to keep executing the code segment for other values of n and see the results.

(for different values of n, you' ll have to modify the plt.text() so that the values are printed inside the graph!!)

2) The sampling distribution of the sample proportion

In this section and as usual, we'll still keep sampling from the same population, and for each sample of size n we'll compute **the proportion of 3 in it,** and we'll keep doing that many many

times and finally study and plot the resulting distribution which we'll call $\hat{\boldsymbol{P}}$, now it can be proven that:

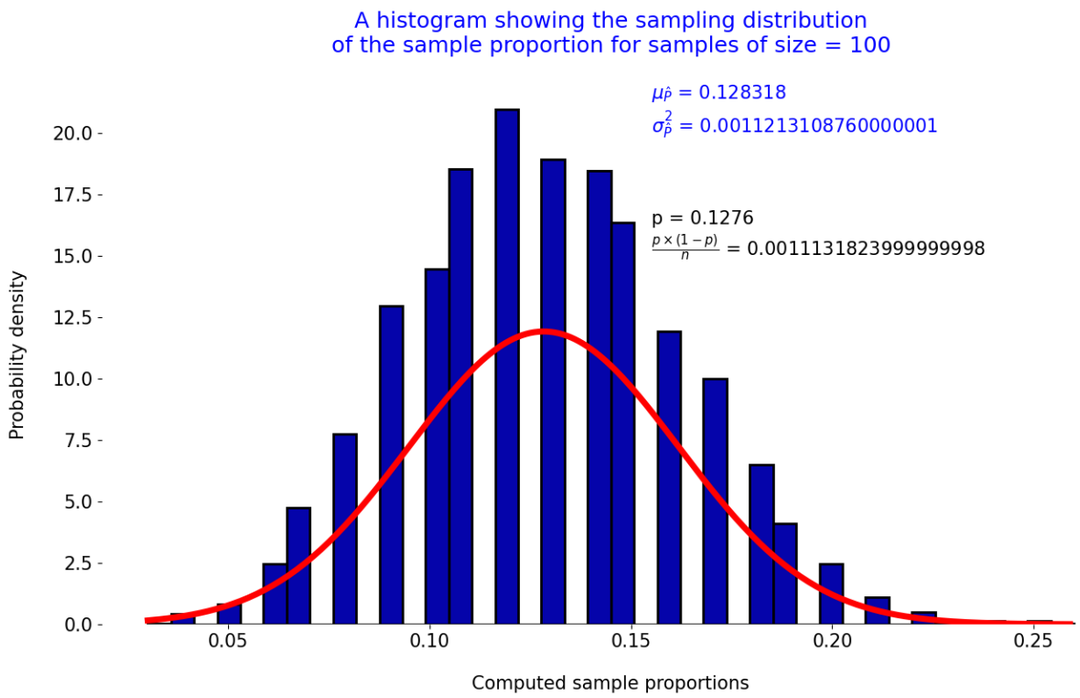
$$\boldsymbol{\mu_{\hat{P}} = p} \qquad \text{where p = 0.1276 is the proportion of 3 in the population}$$

$$\boldsymbol{\sigma^2_{\hat{P}} = \frac{p \times (1-p)}{n}} \qquad \text{since our samples are truly random}$$

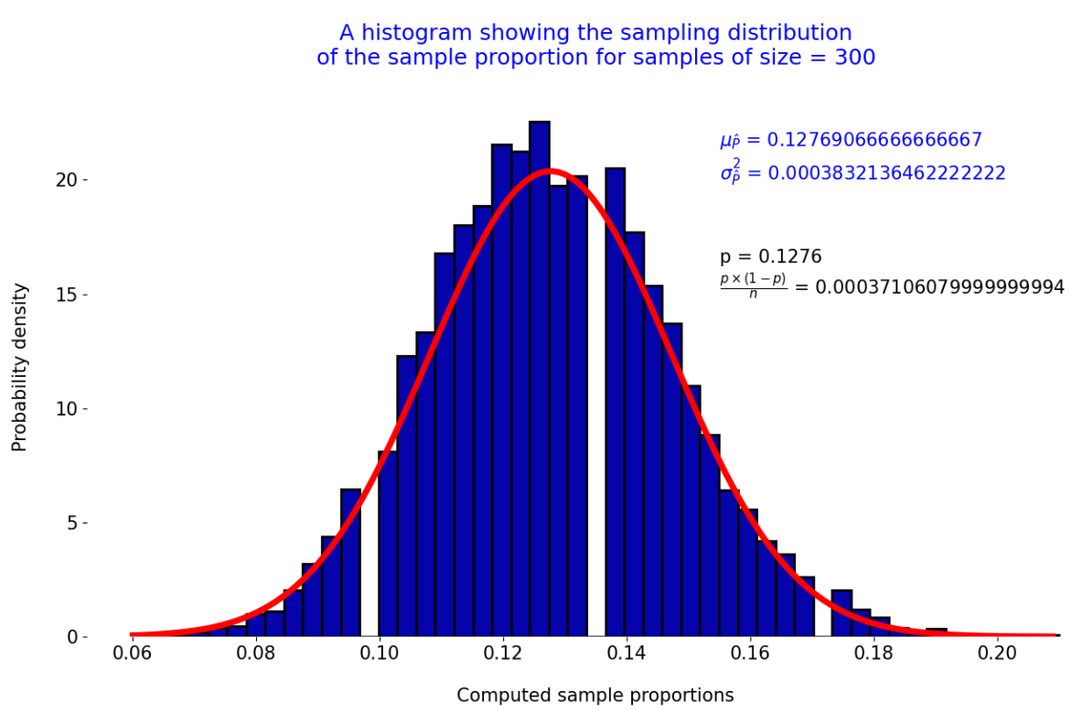
and when $n \geq 30$, the sampling distribution of the sample proportion **will be approximately normal** thanks to the Central Limit Theorem, so let's test this for n = 100 by executing the following code and see the results:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

leading_digit = np.arange(1, 10)
relative_frequencies = [.2957, .1788, .1276, .0963, .0794, .0715, .0571, .0510, .0426]
# this line of code is made just to make sure that we get the same samples
np.random.seed(0)
# we execute this for n = 100 and later on for n = 300
n = 100

# here we'll be drawing 5000 samples, each sample of size n
# and for each sample, we're calculating the proportion of 3 in it
# note that we're storing the sample proportions in this array
sample_proportions = np.empty(5000)

for i in range(5000):
    sample = np.random.choice(leading_digit,size=n,replace=True,p=relative_frequencies)
    # this line of code helps us determine the number of 3 in the sample
    count_3 = np.sum(sample == 3)
    sample_proportions[i] = count_3 / n

# here we compute the mean and variance of resulting distribution
sampling_distribution_mean = np.mean(sample_proportions)
sampling_distribution_variance = np.var(sample_proportions)

# Set general font size of the graph
plt.rcParams.update({"font.size": 15})
# Set width and height of the plot's window
plt.figure(figsize=(15, 8))
plt.box(False)

# now we simply plot the histogram that represents
# the sampling distribution of the sample proportion

plt.hist(sample_proportions, bins="auto", density=True, color="#0504aa",
         edgecolor="black", linewidth=2)

# In the same time we draw the normal curve on top of the histogram
x = np.arange(np.amin(sample_proportions), np.amax(sample_proportions), 0.001)
plt.plot(x, norm.pdf(x, sampling_distribution_mean,
         np.sqrt(sampling_distribution_variance)), color="red", linewidth=5)

# setting the x and y label of the graph
plt.xlabel("\nComputed sample proportions")
plt.ylabel("Probability density\n")

# setting the plot's title
title = "A histogram showing the sampling distribution\n"
title += "of the sample proportion for samples of size = " + str(n) + "\n"
plt.title(title, color="blue")

# first we fill text with the values of all observed parameters
text = "$\\mu_{\\hat{P}}$ = " + str(sampling_distribution_mean)
```

```
text += "\n$\\sigma^2_{\\hat{P}}$ = " + str(sampling_distribution_variance)

# then we display it on the graph
plt.text(0.155, 20, text, color="blue")

# then we fill text again with the theoretical values
text = "p = " + str(0.1276)
text += "\n$\\frac{p \\times (1 - p)}{n}$ = " + str(0.1276 * 0.8724 / n)

# then we display it on the graph
plt.text(0.155, 15, text)
plt.show()
```



Now that's quite odd, normally we would expect that the sampling distribution of the sample proportion would be approximately normal since $100 \geq 30$, but it's definitely not the case here, well turns out that we also need to check that $\boldsymbol{n \times p \geq 15}$ and $\boldsymbol{n \times (1-p) \geq 15}$ especially when the parameter p is very close to 0 or to 1 *(in our case, $p = 0.1276$ is close to 0)*. Note that this rule is based only on practical experience rather than theory, that's why you'll find that other statistical resources may recommend a number different than 15 in the inequality, now let's try to test the same previous code on n = 300 and see the results: *(notice that $300 \times 0.1276 = 38.28 \geq 15$ and $300 \times (1 - 0.1276) = 261.72 \geq 15$)*



Now our sampling distribution seems to be approximately normal, still, maybe for bigger values of n we could actually get even better results, but I'll leave you to check that on your own.

3) The sampling distribution of the sample variance

In this section, we'll be interested in examining the variance of each drawn sample from the population and keep repeating that many many times in order to study the resulting distribution which we'll call $\boldsymbol{S^2}$. Turns out it can be proven that:

$$\boldsymbol{\mu_{S^2} = \sigma^2}$$

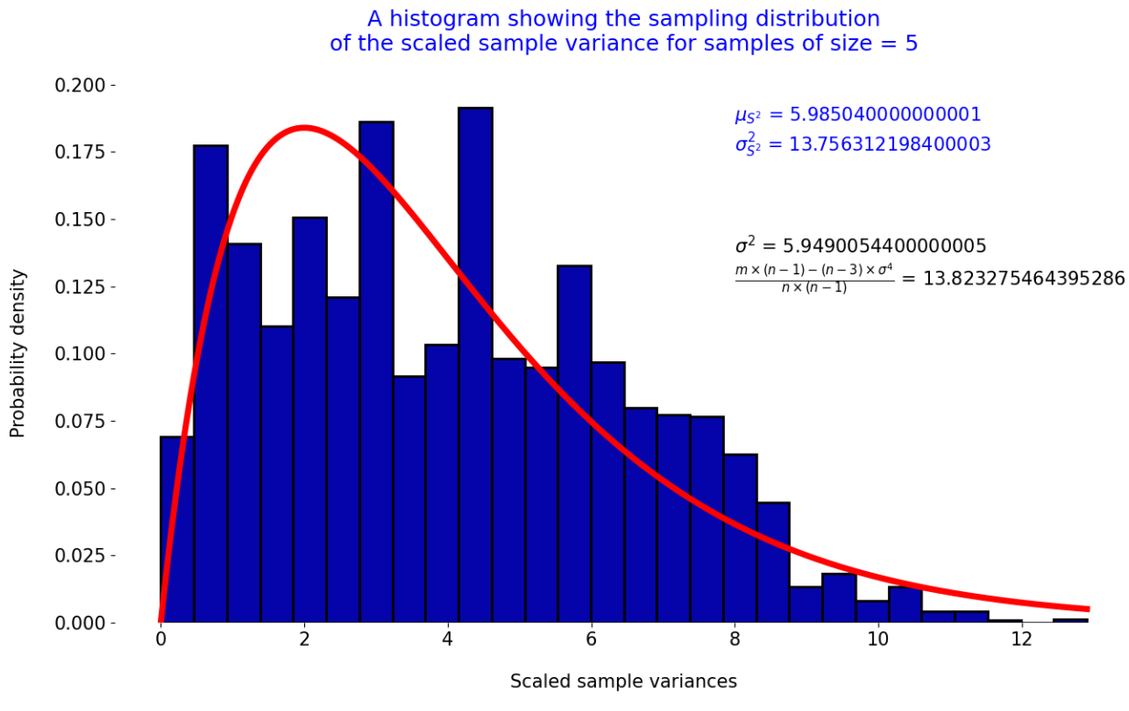
$$\boldsymbol{\sigma^2_{S^2}} = \frac{\boldsymbol{m \times (n-1) - (n-3) \times \sigma^4}}{\boldsymbol{n \times (n-1)}} \text{ where m in our case is}$$

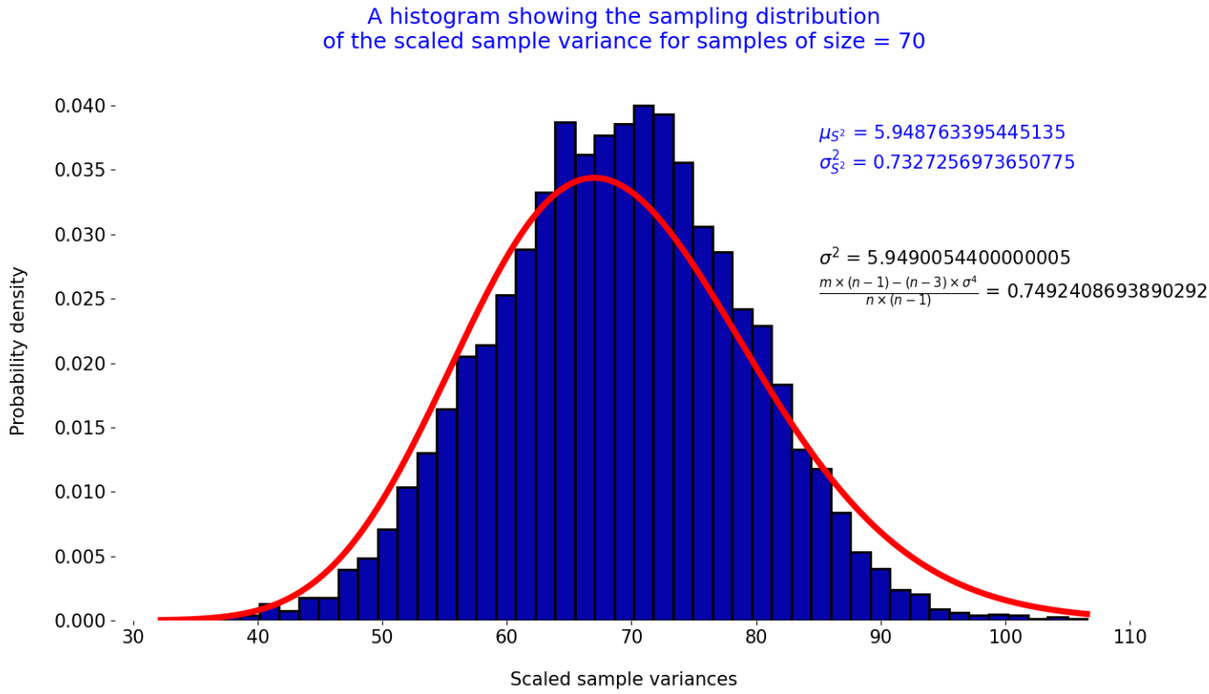
$$\boldsymbol{m = \sum_{i=1}^{9} f_i \times (i - \mu)^4} \text{ and } \boldsymbol{f_i} \text{ are the relative frequencies}$$

In addition, if our population was normal, then $\frac{(n-1) \times S^2}{\sigma^2}$ has **the chi-squared distribution** χ^2 with n - 1 degrees of freedom, unfortunately we can't say the same if we're sampling from a non normal distribution *(which is the case of our population)*, still for large sample sizes the sampling distribution of the sample variance $\boldsymbol{S^2}$ will be approximately normal **regardless of the distribution of the population we're sampling from**, this is true because as n increases, in other words, as the degrees of freedom increase, the χ^2 distribution looks more and more like a normal distribution, to prove this to you try executing the following code for n = 5 and n = 70 and see the resulting plots:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import chi2

# first, we create 2 arrays to store our population
leading_digit = np.arange(1, 10)
relative_frequencies = [.2957, .1788, .1276, .0963, .0794, .0715, .0571, .0510, .0426]

# here we compute the population parameters and the value of m
population_mean = np.sum(leading_digit * relative_frequencies)
population_variance = np.sum(relative_frequencies*(leading_digit - population_mean)**2)
m = np.sum(relative_frequencies * (leading_digit - population_mean) ** 4)

# this line of code is made just to make sure that we get the same samples
np.random.seed(0)

# we execute this for n = 5 and later on for n = 70

n = 5

# here we'll be drawing 5000 samples, each sample of size n
# and for each sample, we're calculating its variance
# note that we're storing the sample variances in this array
sample_variances = np.empty(5000)
for i in range(5000):
    sample = np.random.choice(leading_digit,size=n,replace=True,p=relative_frequencies)
    sample_variances[i] = np.var(sample, ddof=1)

# here we compute the mean and variance of resulting distribution
sampling_distribution_mean = np.mean(sample_variances)
sampling_distribution_variance = np.var(sample_variances)

# Set general font size of the graph
plt.rcParams.update({"font.size": 15})

# Set width and height of the plot's window
plt.figure(figsize=(15, 8))
plt.box(False)
```

```
# REMEMBER we're plotting (n - 1) * S^2 / sigma^2
# note that density=True means that the sum of
# area under histogram is integrated to 1
plt.hist(sample_variances * (n - 1) / population_variance, bins="auto", density=True,
color="#0504aa",edgecolor="black", linewidth=2)

# In the same time we draw the chi-squared curve on top of the histogram
x = np.arange(np.amin(sample_variances) * (n - 1) / population_variance,
              np.amax(sample_variances) * (n - 1) / population_variance, 0.001)
plt.plot(x, chi2.pdf(x, df=n - 1), color="red", linewidth=5)

# setting the x and label of the graph
plt.xlabel("\nScaled sample variances")
plt.ylabel("Probability density\n")

# setting the plot's title
title = "A histogram showing the sampling distribution\n"
title += "of the scaled sample variance for samples of size = " + str(n) + "\n"
plt.title(title, color="blue")

# first we fill text with the values of all observed parameters
text = "$\\mu_{S^2}$ = " + str(sampling_distribution_mean)
text += "\n$\\sigma^2_{S^2}$ = " + str(sampling_distribution_variance)

# then we display it on the graph
plt.text(8, 0.175, text, color="blue")

# USE THIS LINE OF CODE WHEN n = 70 INSTEAD of the previous one
# plt.text(85, 0.035, text, color="blue")

# then we fill text again with the theoretical values
text = "$\\sigma^2$ = " + str(population_variance)
text +="\n$\\frac{m\\times(n - 1)-(n - 3)\\times\\sigma^4}{n \\times (n - 1)}$ = "
text += str((m * (n - 1) - (n -  3) * population_variance ** 2) / (n **2 - n))

# then we display it on the graph
plt.text(8, 0.125, text)

# USE THIS LINE OF CODE WHEN n = 70 INSTEAD of the previous one
# plt.text(85, 0.025, text)

plt.show()
```



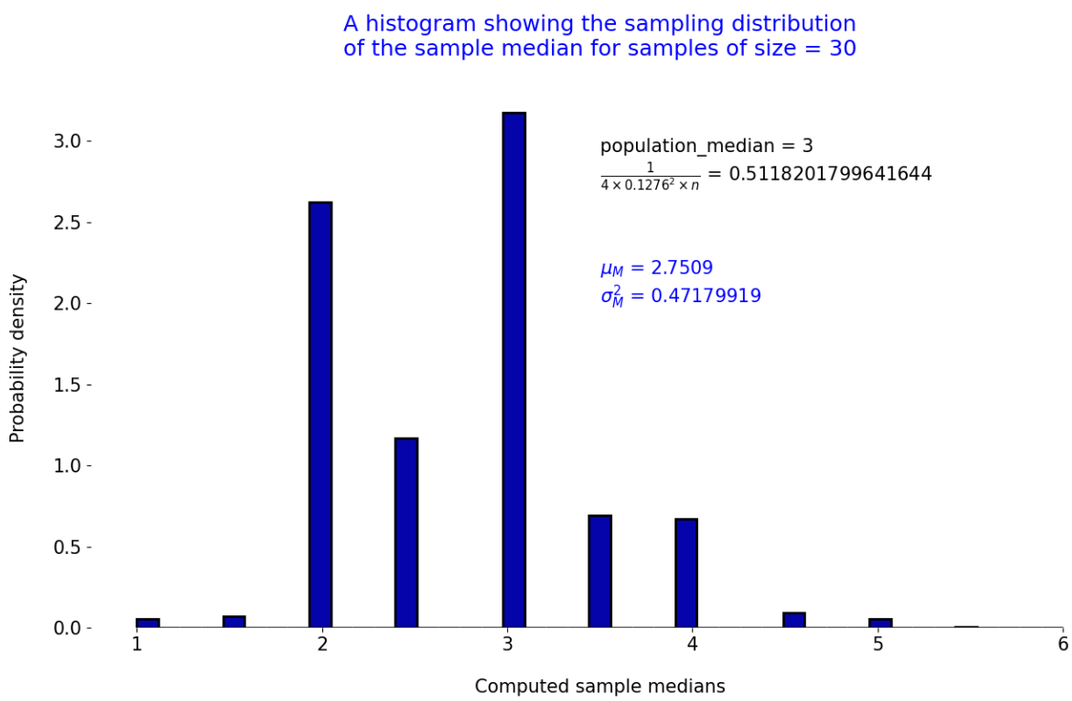


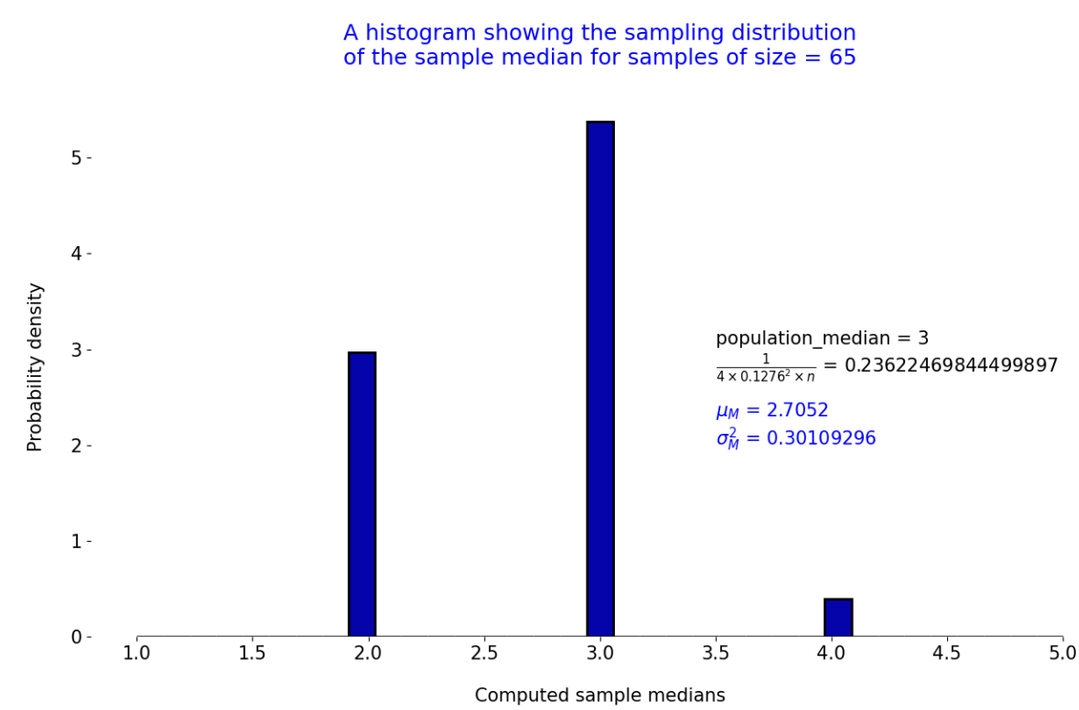
4) A little bonus!

If you're somehow not tired or bored yet from sampling from our population, then try to execute this code segment so you could visualise the sampling distribution of the sample median:

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

leading_digit = np.arange(1, 10)
relative_frequencies = [.2957, .1788, .1276, .0963, .0794, .0715, .0571, .0510, .0426]
# this line of code is made just to make sure that we get the same samples
np.random.seed(0)
# we execute this for n = 30 and n = 65
n = 30
# here we'll be drawing 5000 samples, each sample of size n
# and for each sample, we're calculating its median
# note that we're storing the sample medians in this array
sample_medians = np.empty(5000)
for i in range(5000):
    sample = np.random.choice(leading_digit,size=n,replace=True,p=relative_frequencies)
    sample_medians[i] = np.median(sample)

# here we compute the mean and variance of resulting distribution
sampling_distribution_mean = np.mean(sample_medians)
sampling_distribution_variance = np.var(sample_medians)

# Set general font size of the graph
plt.rcParams.update({"font.size": 15})
plt.figure(figsize=(15, 8))
plt.box(False)

# we plot the histogram of the sampling
# distribution of the sample median
plt.hist(sample_medians, bins="auto", density=True, color="#0504aa",
         edgecolor="black", linewidth=2)

# setting the x and y label of the graph
plt.xlabel("\nComputed sample medians")
plt.ylabel("Probability density\n")

# setting the plot's title
```

```
title = "A histogram showing the sampling distribution\n"
title += "of the sample median for samples of size = " + str(n) + "\n"
plt.title(title, color="blue")

# first we fill text with the values of all observed parameters
text = "$\\mu_{M}$ = " + str(sampling_distribution_mean)
text += "\n$\\sigma^2_{M}$ = " + str(sampling_distribution_variance)

# then we display it on the graph
plt.text(3.5, 2, text, color="blue")

# then we fill text again with the theoretical values
text = "population_median = 3"
text += "\n$\\frac{1}{4 \\times 0.1276^2 \\times n}$ = "
text += str(1 / (4 * n * 0.1276 ** 2))

# then we display it on the graph
plt.text(3.5, 2.75, text)

plt.show()
```





Epilogue

I hope that you've been able to understand all content and plots provided in this statistical study, executed and modified some code segments as well, but most importantly **appreciate the fact that statistical procedures and inferences don't provide 100% accuracy and their results will almost always be at least a little bit different from what's really observed.** So in conclusion, Thank You for taking your time to check the content, and please I will definitely appreciate any suggestion or feedback related to this study and of course let me know if you find any sort of mistake via the following links, as this will help me to learn from them and make better articles in the future.

Click for LinkedIn profile

My Email: **ridazouga@gmail.com**