Introduction to Computer Programming for the Physical Sciences

Joseph F. Hennawi

Spring 2024

☐ Open a new Jupyter notebook

☐ Name your notebook with your name and Homework 1

☐ Open a Markdown cell at the top and write your name and Homework 1

☐ Open a Markdown cell before each problem and write e.g. Problem 1, Problem 2(a), etc.

☐ Please abide by the **Policy and Guidelines on Using AI Tools**

☐ Once you finish the problems: 1) Restart the Python kernel and clear all cell outputs. 2) Rerun the notebook from start to finish so that all answers/outputs show up. 3) Save your notebook as a single .pdf file and upload it to Gradescope on Canvas by the deadline. **No late homeworks will be accepted except for illness accompanied by a doctor's note.**

☐ For parts of problems that require analytical solutions you can perform your calculations using a pencil and paper. Then photograph your work and convert the photograph to a .pdf file using an online

tool. Homework assignments can only be submitted as a single .pdf file, so you will also need to figure out how to concatenate your photo .pdf file and your notebook .pdf file into a single .pdf file that you can submit. Online websites can do this for you. Alternatively, you can code up the analytical solution to your problems in a notebook Markdown cell using the LaTeX mathematical rendering language. This is harder but a chatbot can help you learn it.

Homework 4

Problem 1: Benford's Law

Benford's law refers to the observation that in many real-life datasets, the frequency of the first digits of each number is not uniform. If the digits were distributed uniformly, they would each occur as the first digit with equal probability, i.e. 1/9 per digit, or 11.1% of the time. Instead, there tend to be many more numbers with the first significant (decimal) digit of 1 as compared to 9. This applies best to datsets where the numbers span several orders of magnitude. Familiarize yourself with Benford's law by reading the Wikipedia page here. Incidentally, Benford's law is used in forensic accounting to detect fraud, see here.

Download the file census_data.csv at this link: https://github.com/enigma-igm/Phys29/blob/main/Phys29/homework/HW4/data/census_data.csv This is a comma-separated values or **csv** file containing data from the United States Census Bureau. It lists the number of inhabitants for every town in the USA. As we discussed in the Week4 lecture on Reading and Writing Files, a CSV file is like a simple spreadsheet, where cells in each row are separated by a specific delimiter (in this case, a comma). As is indicated in the first header line of the CSV, the census data has three columns: State,Town,Population.

Note that Problem 2 will require you to reuse some of the code that you develop for this problem. Make sure that you write general functions for this problem that you can then use for both problems.

a) Write a Python program to read the census data file using the `pandas` package.

b) Extract the most significant digit from the Population column data for each town. For example for `California,Santa Barbara city,86353`, the most significant digit of the Population is 8. Store the most significant digits for each

town in a `numpy` array. This array should have integer type and shape `shape=(ntown,)` where `ntown` is the number of towns in the dataset.

c) Construct a numpy array of shape `(9,)` that contains the integer number of times that each of the most significant digits 1-9 appears in the dataset. For this part of the problem, you are allowed to loop over the numbers (1,2,...9), **but no other loops are allowed!!, i.e. you cannot loop over the towns in the dataset** Hint: Use `numpy` Boolean indexing.

d) Write a program to plot a histogram of the **number of times** that each significant digit appears. The x-axis should be the digits (1,2,...9), and the y-axis should be the number of times that each digit appears. Plot the expectation for the case where the distribution across the digits is uniform, i.e. 11.1% per digit, as a horizontal line on the same plot. I realize we have not covered plotting yet in the lectures, but you can use Google or AI to assist you.

e) Using your results from **c)** and **d)** print to the screen the **percentage** of the time that each of the most significant digits 1-9 appears in the dataset.

f) Do the populations in this dataset obey Benford's law? Explain your reasoning.

g) Make a histogram of the town Populations using the following code:

```
from matplotlib import pyplot as plt
log10_bins = np.logspace(np.log10(np.min(populations)),
np.log10(np.max(populations)), 100)
plt.hist(populations, bins=log10_bins)
plt.xscale('log')
plt.yscale('log')
plt.xlabel('Population')
plt.ylabel('Number of Towns')
plt.show()
```

Note that we have chosen our histogram bins to be linearly spaced in the $\log_{10}$ of the population and made our histogram plot with a log scale on the x-axis to better illustrate the full dynamic range of the populations. We similarly chose a log scale on the y-axis to illustrate the large range of the number of cities in each histogram bin.

Based on the *distribution* of populations illustrated by the histogram, comment on why or why not you would expect the population data to obey Benford's law.

In [ ]:
```
#Problem 1
# a)
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# import the required lib
census_data = pd.read_csv('data/census_Data.csv')
#use pandas to read the csv we need
state_data = census_data['State'].values
town_data = census_data['Town'].values
```

```
population_data = census_data['Population'].values
#make several list for the ease of later use
```

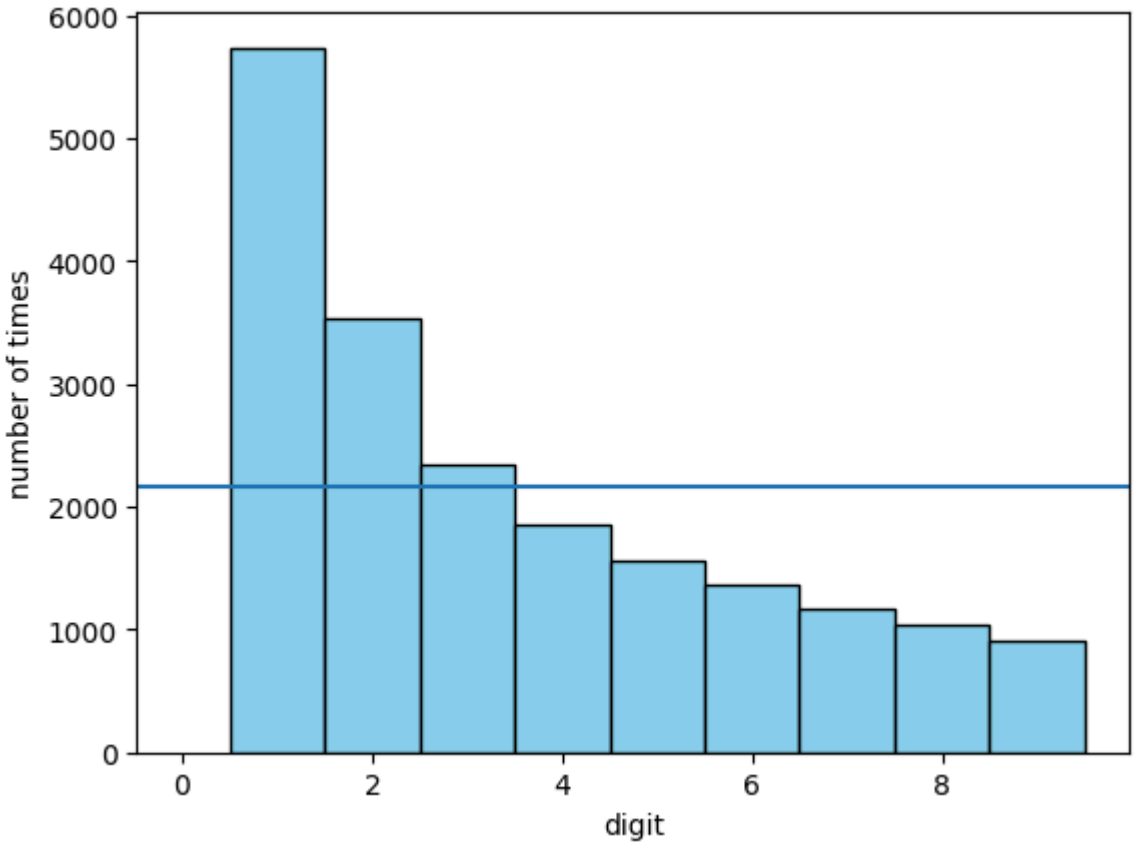
In [ ]:
```
# b)

def extract_significant_digit(input_data):
    # make a function that can make the array with all first digits
    output = np.zeros(len(input_data),int)
    # set the output as a all 0 array with 0s.
    # and make sure that they are all int type
    for i in range(len(input_data)):
            # go through all the elements in the input_data
            output[i] = int(str(input_data[i])[0])
            # first make input_data[i] a string
            # since sting can be use as a list. thus, we can use the indicater [
            # since we already have the the string of the first digit, we can co
    return output
    # return the output.

significant_digit_of_population_data = extract_significant_digit(population_data
# find the need array
print(significant_digit_of_population_data)
# print the value to the screen
```

```
[2 4 7 ... 5 1 1]
```

In [ ]:
```
# c)
def find_frequecy_of_significant_data(input):
    frequency = np.zeros(9,int)
    # set a empty array with assigned shape for later use
    for i in range(9):
        # use the for loop asked in question
        frequency[i] = list(input).count(i+1)
        #the frequce[i] can represent the value of
    return frequency


frequency = find_frequecy_of_significant_data(significant_digit_of_population_da
print(frequency)
```

```
[5738 3540 2342 1847 1559 1370 1166 1043  904]
```

In [ ]:
```
# d)
def frequency_histogram(input):
    #draw the graph as reqired
    plt.bar(range(1,10),height=input,width=1, edgecolor='black', color='skyblue'
    # since we already have the frequecy values, we can use the bar function dir
    plt.xlabel('digit')
    # labal the x axis as digit
    plt.ylabel('number of times')
    # labal y axis as number of times
    plt.axline((0,sum(input)/len(input)),(9,sum(input)/len(input)))
    # plot a line asked, showing the uniform distribution

frequency_histogram(frequency)
# draw from the given array
```



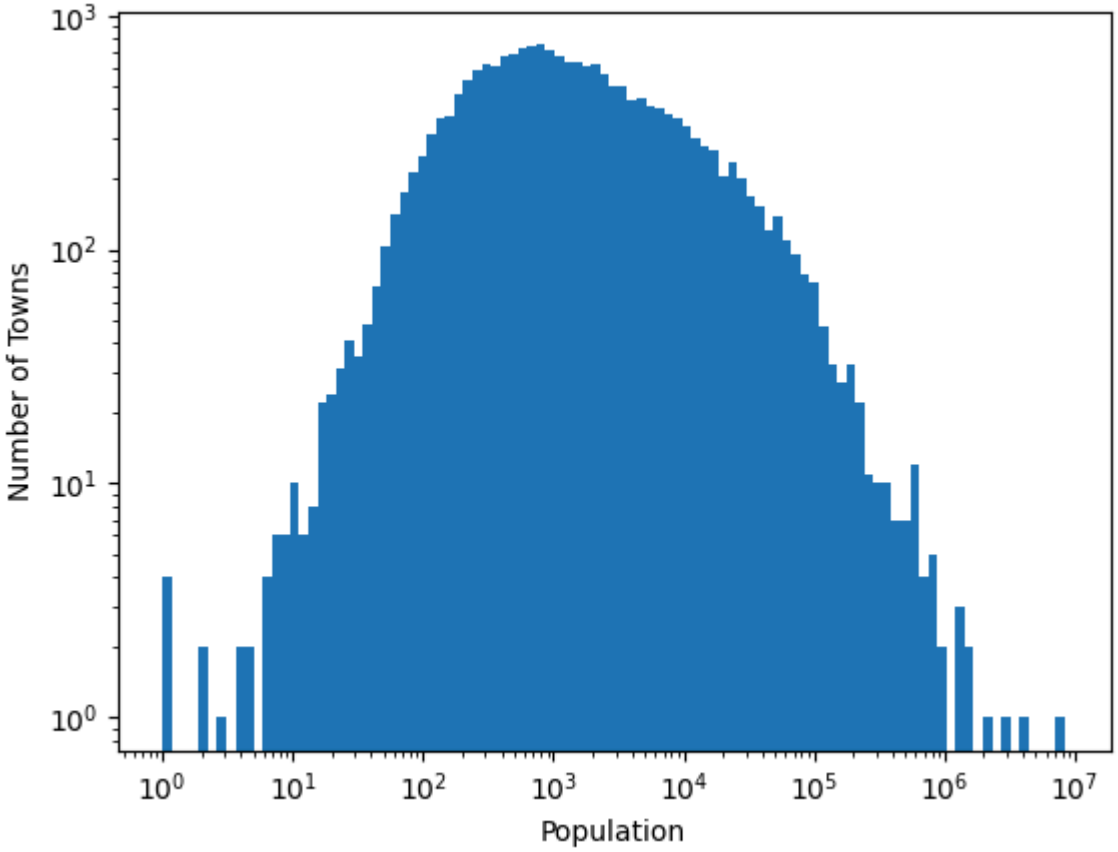
In [ ]:
```
# e)
def frequency_in_percentgae(frequency,significant_digit):
    return frequency/len(significant_digit)

print(frequency_in_percentgae(frequency,significant_digit_of_population_data))
```

```
[0.29412066 0.18145471 0.12004716 0.09467425 0.07991184 0.070224
 0.05976729 0.0534625  0.04633759]
```

f)

The data sheet does follow the Benford's Law, since the distribution given by the law is 30%, 17.6%, 12.5%, 9.7%, 7.9%, 6.7%, 5.8%, 5.1%, 4.6% which is relative similar to the result we get.

In [ ]:
```
# g)

log10_bins = np.logspace(np.log10(np.min(population_data)), np.log10(np.max(popu
plt.hist(population_data, bins=log10_bins)
plt.xscale('log')
plt.yscale('log')
plt.xlabel('Population')
plt.ylabel('Number of Towns')
plt.show()
```



from the graph we can find out that excetp sevral exceptions laying out saide the middle, most of the data falls in to the first few blocks in each 10^x which represents the first digits to be 1. we can easily find out that the larger the significant value is, the samller it is on the x axis. Thus, we can find out that the Benford's law occurs.

Problem 2: Munich Temperatures

Download the file munich_temperatures.txt at this link: https://github.com/enigma-igm/Phys29/blob/main/Phys29/homework/HW4/data/munich_temperatures.txt This is a text file containing the temperature (averaged over a 24 hour day) in Munich, Germany for each day between 1995-2013. The dates are represented by a floating point number where the integer part is the year and the decimal part is the fraction of the year. In other words, 1995.75 would be three quarters into the year 1995, or 10/01/1995. The average temperatures are in degrees Celsius.

a) Write a Python program to read this file using the `pandas` package. Assign the date and temperature columns of the `pandas` `DataFrame` to two arrays of shape `(ndays,)` where `ndays` is the number of days in the dataset.

b) The temperature data contains bad values. Analyze the temperatures and figure out how to identify which values are the bad ones (remember these are temperatures in degrees Celsius not Fahrenheit!). Use `numpy` boolean indexing to

create two new date and temperature `numpy` arrays that contain only the good data.

c) Using the good data from part **b)**, compute the average (i.e. use `numpy.mean`) temperature over the entire time period covered by the dataset and print the value to the screen. **You are not allowed to perform any loops in this part of the problem.**

d) Compute the minimum, maximum and average temperature as a function of the year for each year 1995, 1996, ..., 2012 (ignore 2013 since it is incompletely covered). **You are allowed to loop over the years** (i.e. `range(1995,2013)`), **but no other loops are allowed**. Print the minimum, maximum, and average temperature for each year to the screen.

e) Convert the good temperatures in the Munich temperatures dataset onto the *absolute temperature* scale in degrees Kelvin (**no loops allowed!**).

f) Repeat Problem1(c) but for the good temperatures in the Munich temperature dataset **in degrees Kelvin** using the functions that you wrote for Problem 1. **Do not repeat code, i.e. use the general functions that you wrote in Problem 1 to solve this part and the rest of the parts of this Problem below.**

g) Repeat Problem1(d) but for the good temperatures in degrees Kelvin using the functions that you wrote for Problem 1.

h) Repeat Problem1(e) but for the good temperatures in degrees Kelvin using the functions that you wrote for Problem 1.

i) Do the daily temperatures in Munich in degrees Kelvin obey Benford's law? Explain your reasoning.

j) Make a histogram of good temperatures in degrees Kelvin as in Problem1(g). Think about the best choices for the histogram bins (i.e. logarthmic or linear) and the scale of the x-axis (i.e. linear or logarithmic) and y-axis given the dynamic range of this data.

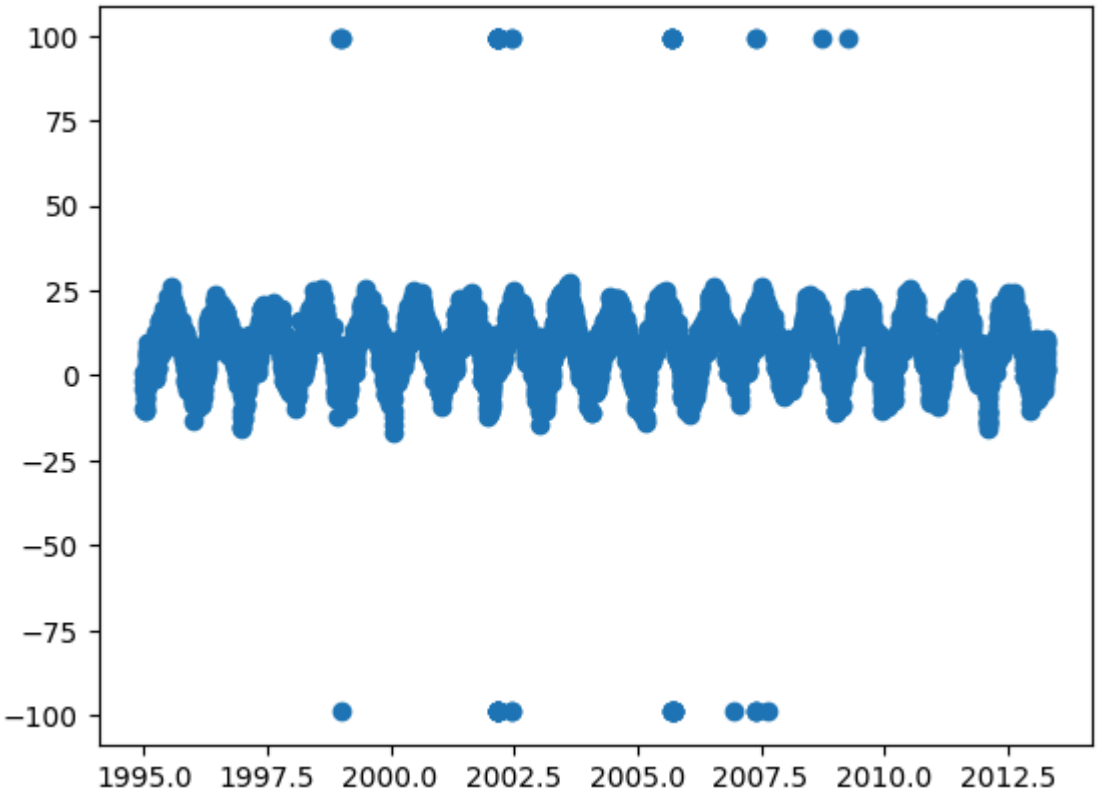
Based on the *distribution* of temperatures, comment on why or why not you would expect the temperature data to obey Benford's law.

In [ ]:
```
#problem 2
# a)
munich_temperatures = pd.read_csv('data/munich_temperatures.txt', sep='\s+', com
# use panda to read the file.
date_munich_temperatures = np.array(munich_temperatures['date (yr)'].values,floa
temperature_munich_temperatures = np.array(munich_temperatures['Temp (deg C)'].v
# make two arrays which are asked in the question
print(date_munich_temperatures)
print(temperature_munich_temperatures)
```

```
[1995.00274 1995.00548 1995.00821 ... 2013.27926 2013.282   2013.28474]
[ 0.944444 -1.61111  -3.55556  ... 10.5556    8.94444  11.1667  ]
```

In [ ]:
```
# b)
plt.scatter(date_munich_temperatures,temperature_munich_temperatures)
```

Out[ ]:
```
<matplotlib.collections.PathCollection at 0x26e135f75d0>
```



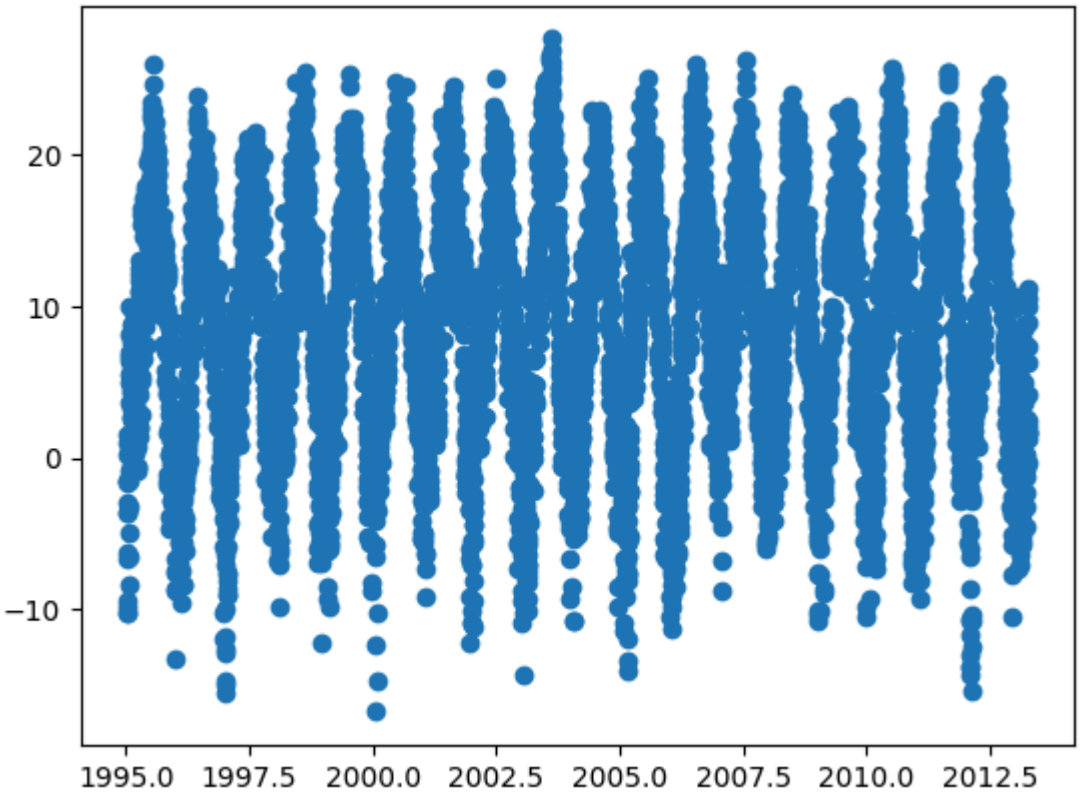
from the scatter we get, we can find out that the bad data is vary far away from the normal data. Thus, we can make all data to be good data we want by exclude the points 50 from median.

In [ ]:
```
upper_limit = 50
lower_limit = -50
# setting the limit of good data

'''
new_date_munich_temperatures = np.array([])
new_temperature_munich_temperatures = np.array([])
# create empty array for later use

for i in range(len(temperature_munich_temperatures)):
    #loop over all the elements
    if(bool(temperature_munich_temperatures[i] < float(upper_limit)) and bool(te
        # find the temperatrue which looks good
        new_date_munich_temperatures = np.append(new_date_munich_temperatures,da
        new_temperature_munich_temperatures = np.append(new_temperature_munich_t
        # add the good data to the end of the good data array
'''
a = temperature_munich_temperatures < upper_limit
# list the bool to be true for all elements lower than the upper limit
b = temperature_munich_temperatures > lower_limit
# list the bool to be true for all elements greater than teh loewr limit
correct_index = np.logical_and(a,b)
# use and so that only if the number is greater than lower limit and smaller tha
new_date_munich_temperatures = date_munich_temperatures[correct_index]
new_temperature_munich_temperatures = temperature_munich_temperatures[correct_in
```

```
# use the boolean index to create the correct array

print(new_temperature_munich_temperatures)
print(new_date_munich_temperatures)
# print the good data
plt.scatter(new_date_munich_temperatures,new_temperature_munich_temperatures)
# make a graph just to make sure that no expcetion is lost
```

```
[ 0.944444 -1.61111  -3.55556  ... 10.5556    8.94444  11.1667  ]
[1995.00274 1995.00548 1995.00821 ... 2013.27926 2013.282   2013.28474]
```

Out[ ]: <matplotlib.collections.PathCollection at 0x26e14ca60d0>



In [ ]:
```
# c)
print(np.mean(new_temperature_munich_temperatures))
# use np to calculate the mean
```

```
8.933222104668378
```

In [ ]:
```
# d)
for i in range(1995,2013):
    # loop over the years
    a = new_date_munich_temperatures > i
    # find the date which is larger than the starting date
    b = new_date_munich_temperatures < i+1
    # find the date which is smaller than the ending data
    correct_index = np.logical_and(a,b)
    # only if both are satified it can be the correct index
    print(i,np.max(new_temperature_munich_temperatures[correct_index]),np.min(ne
    # print the year, max, min, average in order, using the boolean indexing
```

```
1995 25.9444 -13.2778 8.76560005479452
1996 23.8333 -15.5 7.229833593424658
1997 21.5556 -12.8889 8.548421924590164
1998 25.5 -12.2222 9.245702324861877
1999 25.3333 -9.83333 9.110651741758241
2000 24.7778 -16.7778 9.765446401643835
2001 24.5556 -12.1667 9.00713250874317
2002 25.1111 -11.1111 9.881714284084085
2003 27.6667 -14.3333 9.398325638356164
2004 23.0 -10.7778 8.8770157030137
2005 25.1111 -14.1111 8.22475459088319
2006 25.9444 -11.2778 9.163765467582417
2007 26.2778 -8.83333 9.768647250696377
2008 24.0556 -5.11111 9.660867162637363
2009 23.2778 -10.7778 9.372297064657534
2010 25.7222 -9.33333 8.321307882191782
2011 25.5 -9.27778 9.691631063013697
2012 24.7222 -15.4444 9.225267140821916
```

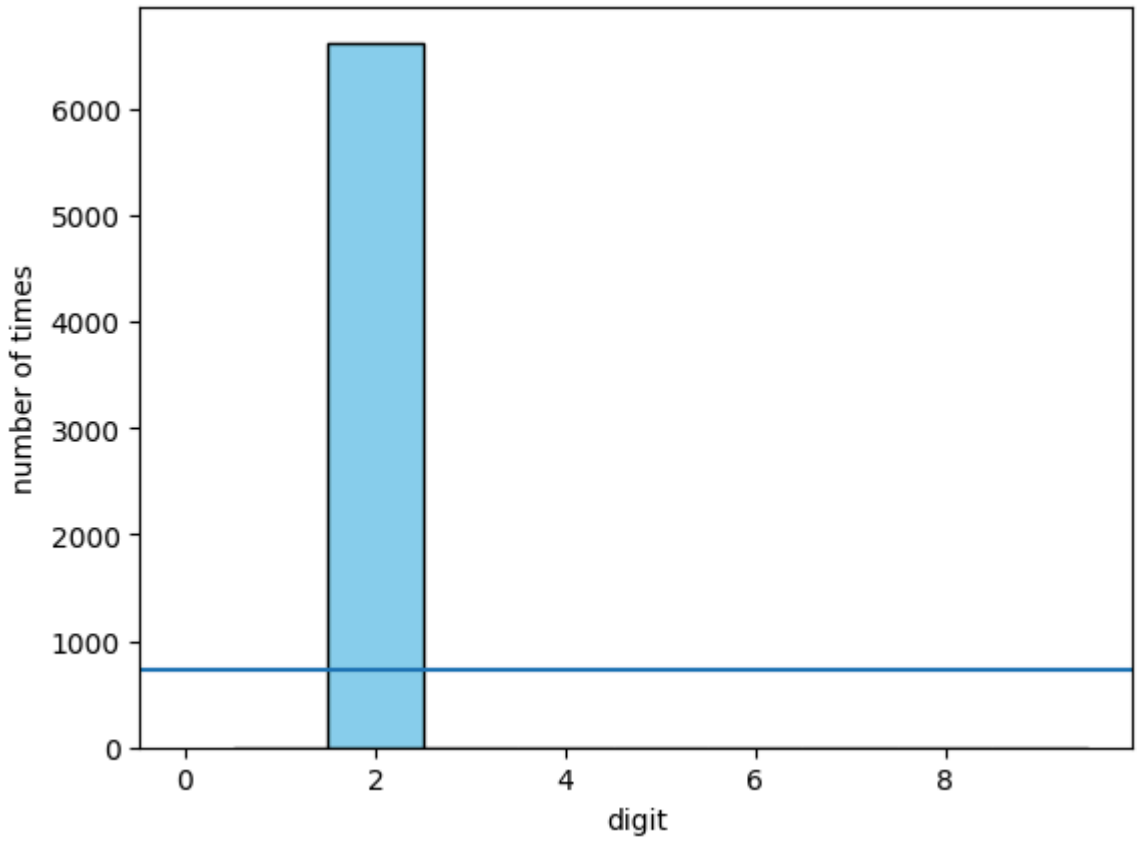
In [ ]:
```
# e)

new_temperature_munich_temperatures_K = new_temperature_munich_temperatures + 27
# using the feature of np array
print(new_temperature_munich_temperatures_K)
```

```
[274.094444 271.53889  269.59444  ... 283.7056   282.09444  284.3167  ]
```

In [ ]:
```
# f)

significant_digit_of_temperature_data = extract_significant_digit(new_temperatur
frequency = find_frequecy_of_significant_data(significant_digit_of_temperature_d
print(frequency)
```

```
[   0 6617    2    0    0    0    0    0    0]
```

In [ ]:
```
# g)

frequency_histogram(frequency)
```



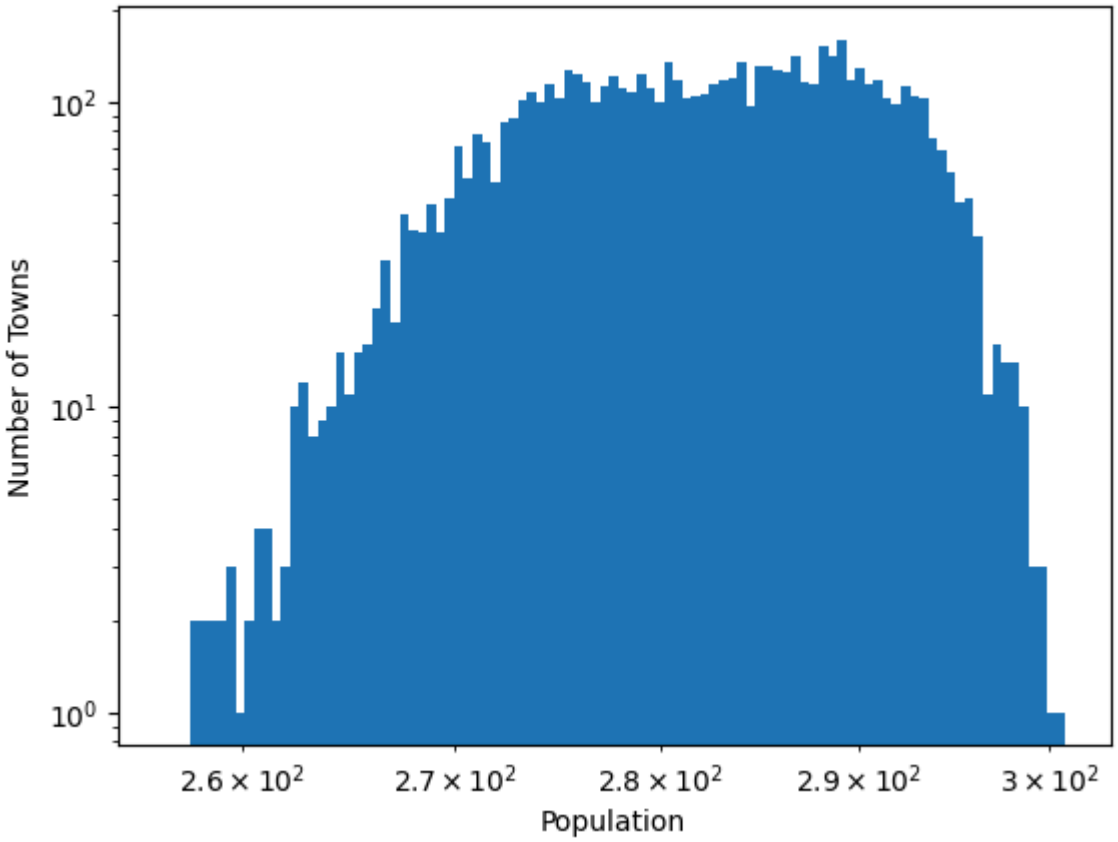
In [ ]:
```
# h)
print(frequency_in_percentgae(frequency,significant_digit_of_temperature_data))
```

```
[0.00000000e+00 9.99697840e-01 3.02160447e-04 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00]
```

i)

since bonford's law focus on the first digit, it will not work in K temperature in this, since the change of temperature will not affect the first digit.

In [ ]:
```
# j)

log10_bins = np.logspace(np.log10(np.min(new_temperature_munich_temperatures_K))
plt.hist(new_temperature_munich_temperatures_K, bins=log10_bins)
plt.xscale('log')
plt.yscale('log')
plt.xlabel('Population')
plt.ylabel('Number of Towns')
plt.show()
```

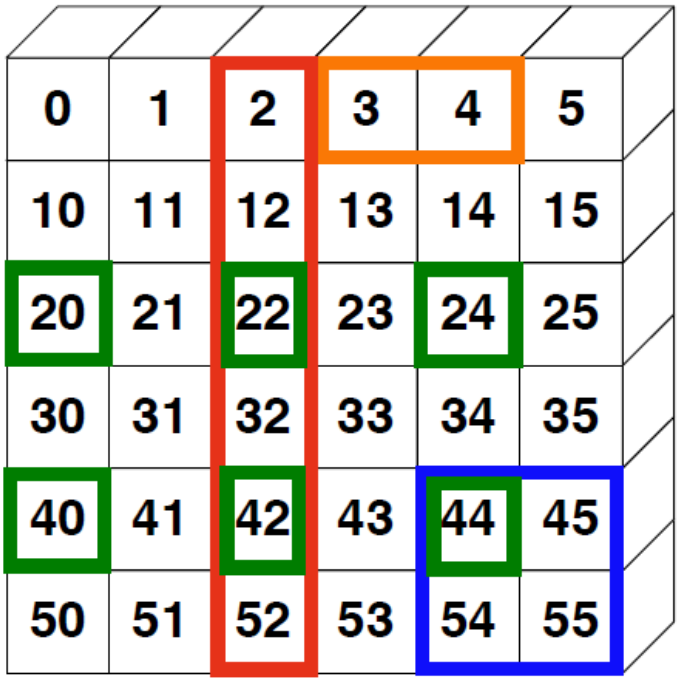


I still choose the log form since the bonford's law is caused by the different space occupied by log. in the daigram we can find out that though the shape is till like a bell shape, but the range of it is so small that benford's law may not able to work

Problem 3: Array Slicing

Download the data slicing.txt at this link: https://github.com/enigma-igm/Phys29/blob/main/Phys29/homework/HW4/data/slicing.txt. This is a text file containing a 2D numpy array of `shape=(6,6,)`. Read the data file into a `numpy` array using `np.loadtxt`. See the Week4 lecture for an example of how to use `np.loadtxt`.

The colors in the image below indicate different sub-arrays that can be extracted from the main array. For each of the colors (sub-arrays), obtain the new numpy sub-array with a single `numpy` command using the slice/stride syntax. Print the shape of the sub-arrays and the contents of the sub-arrays to the screen to verify that your results are correct.



In [ ]:
```
# problem 3

data = np.loadtxt('data/slicing.txt',comments='#')
# read the data we need.

print(data[0:6,2])
# red

print(data[0,3:5])
# organ

print(data[2:5:2,0:5:2])
# green

print(data[4:6,4:6])
# blue
```

```
[ 2. 12. 22. 32. 42. 52.]
[3. 4.]
[[20. 22. 24.]
 [40. 42. 44.]]
[[44. 45.]
 [54. 55.]]
```