L3-BCU_KBP-Part2_3

April 8, 2021

1 TP 3 : Scientific operations tools

1.1 Part II. - Animals database

1.1.1 II.1) Create notebook + read and create a dataframe from database.

[38]:
```
import pandas as pd
import numpy as np ## .. for np.arange().
import statistics ## .. for mean + median method().
from matplotlib import pyplot as plt ## .. for histogram.
from math import log10, floor
```

[39]:
```
## Read file & create dataframe :
df_ani = pd.read_csv("PanTHERIA_1-0_WR05_Aug2008.txt", sep='\t')
```

[40]:
```
## Show content :
df_ani.describe()
```

[40]:
```
       1-1_ActivityCycle  5-1_AdultBodyMass_g  8-1_AdultForearmLen_mm  \
count        5416.000000         5.416000e+03             5416.000000
mean         -692.433346         1.159401e+05             -823.276128
std           461.384103         2.638103e+06              393.076695
min          -999.000000        -9.990000e+02             -999.000000
25%          -999.000000        -9.990000e+02             -999.000000
50%          -999.000000         2.312500e+01             -999.000000
75%             1.000000         2.819150e+02             -999.000000
max             3.000000         1.543213e+08              246.000000

       13-1_AdultHeadBodyLen_mm  2-1_AgeatEyeOpening_d  3-1_AgeatFirstBirth_d  \
count               5416.000000            5416.000000            5416.000000
mean                -441.370303            -909.936621            -848.912626
std                 1125.326888             286.985807             552.433154
min                 -999.000000            -999.000000            -999.000000
25%                 -999.000000            -999.000000            -999.000000
50%                 -999.000000            -999.000000            -999.000000
75%                  131.005000            -999.000000            -999.000000
max                30480.000000             153.500000            5456.750000
```

	18-1_BasalMetRate_mLO2hr	5-2_BasalMetRateMass_g	6-1_DietBreadth	\
count	5416.000000	5416.00000	5416.000000	
mean	-694.539337	-42.32449	-599.361337	
std	2775.817235	12409.88190	490.519927	
min	-999.000000	-999.00000	-999.000000	
25%	-999.000000	-999.00000	-999.000000	
50%	-999.000000	-999.00000	-999.000000	
75%	-999.000000	-999.00000	2.000000	
max	113712.000000	407000.00000	8.000000	

	7-1_DispersalAge_d	…	26-6_GR_MinLong_dd	26-7_GR_MidRangeLong_dd	\
count	5416.000000	…	5416.000000	5416.000000	
mean	-961.650096	…	-124.978658	-116.195251	
std	241.549875	…	359.006349	361.919556	
min	-999.000000	…	-999.000000	-999.000000	
25%	-999.000000	…	-93.272500	-79.335000	
50%	-999.000000	…	3.055000	16.930000	
75%	-999.000000	…	89.640000	99.427500	
max	5248.980000	…	172.340000	175.750000	

	27-1_HuPopDen_Min_n/km2	27-2_HuPopDen_Mean_n/km2	\
count	5416.000000	5416.000000	
mean	-124.672821	-69.937411	
std	354.244136	390.450236	
min	-999.000000	-999.000000	
25%	0.000000	6.000000	
50%	1.000000	28.850000	
75%	5.000000	75.602500	
max	1119.000000	2060.500000	

	27-3_HuPopDen_5p_n/km2	27-4_HuPopDen_Change	28-1_Precip_Mean_mm	\
count	5416.000000	5416.000000	5416.000000	
mean	-122.373892	-145.818923	-61.841064	
std	355.359663	352.869151	419.305365	
min	-999.000000	-999.000000	-999.000000	
25%	0.000000	0.030000	29.832500	
50%	2.000000	0.090000	91.065000	
75%	8.125000	0.120000	154.607500	
max	1119.000000	1.000000	461.000000	

	28-2_Temp_Mean_01degC	30-1_AET_Mean_mm	30-2_PET_Mean_mm
count	5416.000000	5416.000000	5416.000000
mean	-6.574847	580.167897	885.371204
std	443.838735	896.708034	1009.305361
min	-999.000000	-999.000000	-999.000000
25%	68.150000	269.327500	639.625000
50%	199.655000	843.705000	1379.115000

```
75%                 239.215000       1276.722500       1577.025000
max                 350.000000       1858.560000       2107.000000

[8 rows x 49 columns]
```

1.1.2 II.2) Take weight of animals as adults ("5-1_AdultBodyMass_g")

[41]:
```
## Values reckoning of "5-1_AdultBodyMass_g" column
print(df_ani["5-1_AdultBodyMass_g"])
```

```
0       492714.47
1        10392.49
2         9658.70
3        11989.10
4        31756.51
          …
5411        40.42
5412        93.99
5413       123.00
5414       100.00
5415        95.02
Name: 5-1_AdultBodyMass_g, Length: 5416, dtype: float64
```

1.1.3 II.3) Take a look to the values distribution, find :

- MINIMUM

- MAXIMUM

- AVERAGE

- MEDIAN

1.1.4 And control if they are some unrealistic values, missing datas, etc …

[42]:
```
## Check missing values (defined by <NA> label)
df_ani.isna().sum()
```

[42]:
```
MSW05_Order                     0
MSW05_Family                    0
MSW05_Genus                     0
MSW05_Species                   0
MSW05_Binomial                  0
1-1_ActivityCycle               0
5-1_AdultBodyMass_g             0
8-1_AdultForearmLen_mm          0
13-1_AdultHeadBodyLen_mm        0
2-1_AgeatEyeOpening_d           0
```

```
3-1_AgeatFirstBirth_d             0
18-1_BasalMetRate_mLO2hr          0
5-2_BasalMetRateMass_g            0
6-1_DietBreadth                   0
7-1_DispersalAge_d                0
9-1_GestationLen_d                0
12-1_HabitatBreadth               0
22-1_HomeRange_km2                0
22-2_HomeRange_Indiv_km2          0
14-1_InterbirthInterval_d         0
15-1_LitterSize                   0
16-1_LittersPerYear               0
17-1_MaxLongevity_m               0
5-3_NeonateBodyMass_g             0
13-2_NeonateHeadBodyLen_mm        0
21-1_PopulationDensity_n/km2      0
10-1_PopulationGrpSize            0
23-1_SexualMaturityAge_d          0
10-2_SocialGrpSize                0
24-1_TeatNumber                   0
12-2_Terrestriality               0
6-2_TrophicLevel                  0
25-1_WeaningAge_d                 0
5-4_WeaningBodyMass_g             0
13-3_WeaningHeadBodyLen_mm        0
References                        0
5-5_AdultBodyMass_g_EXT           0
16-2_LittersPerYear_EXT           0
5-6_NeonateBodyMass_g_EXT         0
5-7_WeaningBodyMass_g_EXT         0
26-1_GR_Area_km2                  0
26-2_GR_MaxLat_dd                 0
26-3_GR_MinLat_dd                 0
26-4_GR_MidRangeLat_dd            0
26-5_GR_MaxLong_dd                0
26-6_GR_MinLong_dd                0
26-7_GR_MidRangeLong_dd           0
27-1_HuPopDen_Min_n/km2           0
27-2_HuPopDen_Mean_n/km2          0
27-3_HuPopDen_5p_n/km2            0
27-4_HuPopDen_Change              0
28-1_Precip_Mean_mm               0
28-2_Temp_Mean_01degC             0
30-1_AET_Mean_mm                  0
30-2_PET_Mean_mm                  0
dtype: int64
```

No missing values : **Checked !**

```
[43]: ## MINIMUM :␣
 ↪---------------------------------------------------------------------->
print(f'Minimum = {df_ani["5-1_AdultBodyMass_g"].min():.3f}[g]')
LOWER_LIMIT = 0
if df_ani["5-1_AdultBodyMass_g"].min() < LOWER_LIMIT:
    ## List with only realistic values (under 0 deleted) :
    ls_neg_filter = [x for x in df_ani["5-1_AdultBodyMass_g"] if x >=␣
 ↪LOWER_LIMIT]
```

You can filter directly using pandas, it's simpler.

```
    ## Check filtered list :
    #print(ls_neg_filter)
    print(f'Minimum (filtered) = {min(ls_neg_filter):.3f}[g]')

## MAXIMUM :␣
 ↪---------------------------------------------------------------------->
print(f'Maximum = {max(ls_neg_filter):.3f}[g] = \
{max(ls_neg_filter)/1000:.3f}[kg] = \
{max(ls_neg_filter)/1000000:.3f}[t]')

## AVERAGE :␣
 ↪---------------------------------------------------------------------->
print(f'Average = {statistics.mean(ls_neg_filter):.3f}[g]    = \
{statistics.mean(ls_neg_filter)/1000:.3f}[kg]    = \
{statistics.mean(ls_neg_filter)/1000000:.3f}[t]')

## MEDIAN  :␣
 ↪---------------------------------------------------------------------->
print(f'Median  = {statistics.median(ls_neg_filter):.3f}[g]')
```

```
Minimum = -999.000[g]
Minimum (filtered) = 1.960[g]
Maximum = 154321304.500[g] = 154321.304[kg] = 154.321[t]
Average = 177810.182[g]    = 177.810[kg]    = 0.178[t]
Median  = 104.465[g]
```

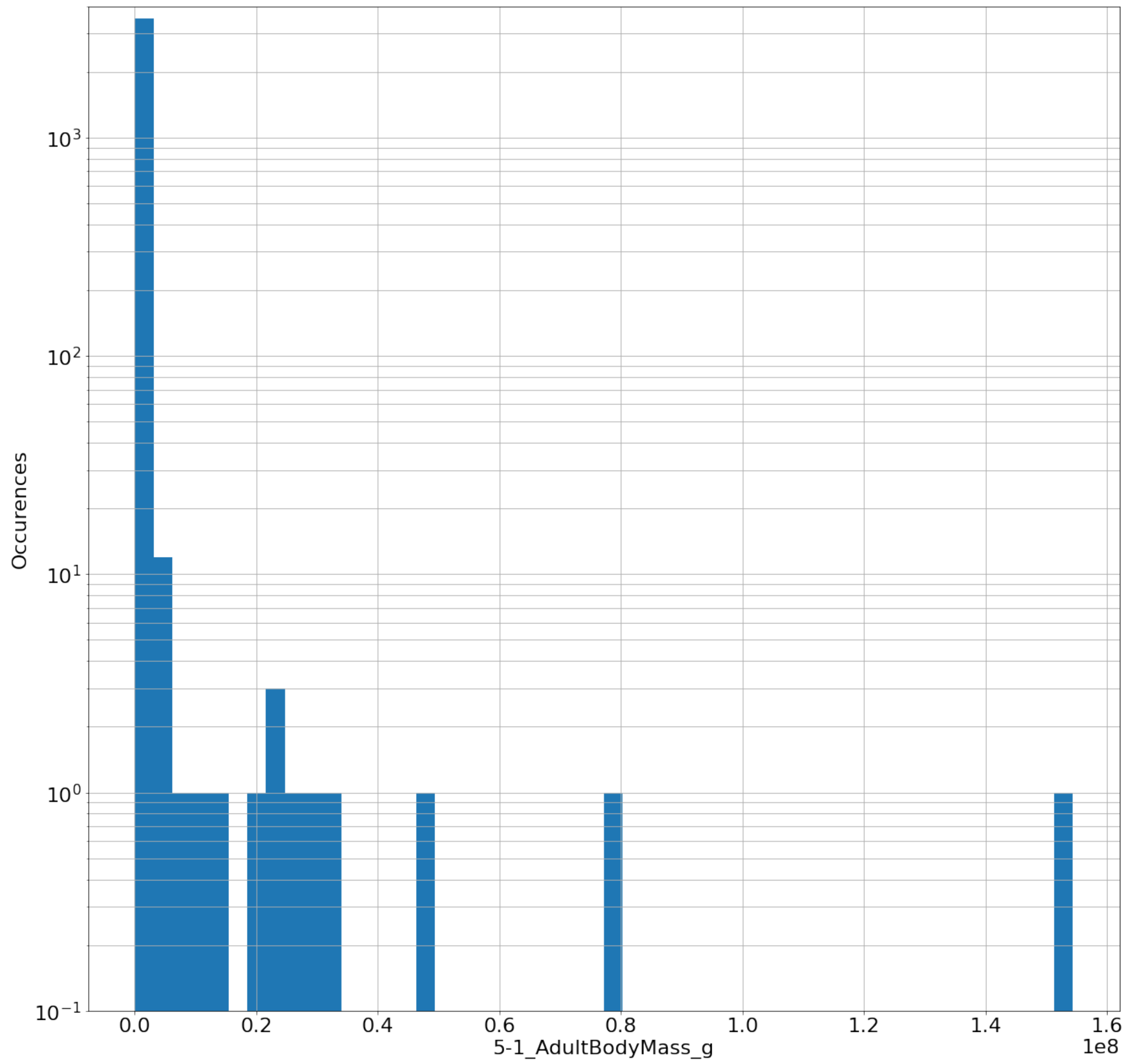
Observations : - **No missing values detected.** - Some values are **not realistic**, like the minimum one. It was a **negative value.** So we make a filter to keep positive values only. - A **blue whale** as an adult can weight up to **150**[**t**], so the **maximum is correct**. - We can see that **the median and the average are different**, due to the fact that the **AVERAGE** value is calculated with every values in the list. And the **MEDIAN** is counting the number of values above and below a certain point and if the number of values above is equal to the number of value below that point, the median is found.

1.1.5 II.4) Histogram of Body masses

```
[44]: ## Set size of the graph
plt.figure(figsize=(20,20))

## Set font size around graph
```

```
plt.rcParams.update({'font.size': 22})

## Show grid through graph
plt.grid(True, which="both")

## Set y scale
plt.ylim(0.1, 4000)

## X/Y titles :
plt.xlabel("5-1_AdultBodyMass_g")
plt.ylabel("Occurences")

## Show
plt.hist(ls_neg_filter, bins=50, log=True)
```

```
[44]: (array([3.517e+03, 1.200e+01, 1.000e+00, 1.000e+00, 1.000e+00, 0.000e+00,
              1.000e+00, 3.000e+00, 1.000e+00, 1.000e+00, 1.000e+00, 0.000e+00,
              0.000e+00, 0.000e+00, 0.000e+00, 1.000e+00, 0.000e+00, 0.000e+00,
              0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
              0.000e+00, 1.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
              0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
              0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
              0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
              0.000e+00, 1.000e+00]),
       array([1.96000000e+00, 3.08642801e+06, 6.17285406e+06, 9.25928011e+06,
              1.23457062e+07, 1.54321322e+07, 1.85185583e+07, 2.16049843e+07,
              2.46914104e+07, 2.77778364e+07, 3.08642625e+07, 3.39506885e+07,
              3.70371146e+07, 4.01235406e+07, 4.32099667e+07, 4.62963927e+07,
              4.93828188e+07, 5.24692448e+07, 5.55556709e+07, 5.86420969e+07,
              6.17285230e+07, 6.48149490e+07, 6.79013751e+07, 7.09878011e+07,
              7.40742272e+07, 7.71606532e+07, 8.02470793e+07, 8.33335053e+07,
              8.64199314e+07, 8.95063574e+07, 9.25927835e+07, 9.56792095e+07,
              9.87656356e+07, 1.01852062e+08, 1.04938488e+08, 1.08024914e+08,
              1.11111340e+08, 1.14197766e+08, 1.17284192e+08, 1.20370618e+08,
              1.23457044e+08, 1.26543470e+08, 1.29629896e+08, 1.32716322e+08,
              1.35802748e+08, 1.38889174e+08, 1.41975600e+08, 1.45062026e+08,
              1.48148452e+08, 1.51234878e+08, 1.54321304e+08]),
       <BarContainer object of 50 artists>)
```



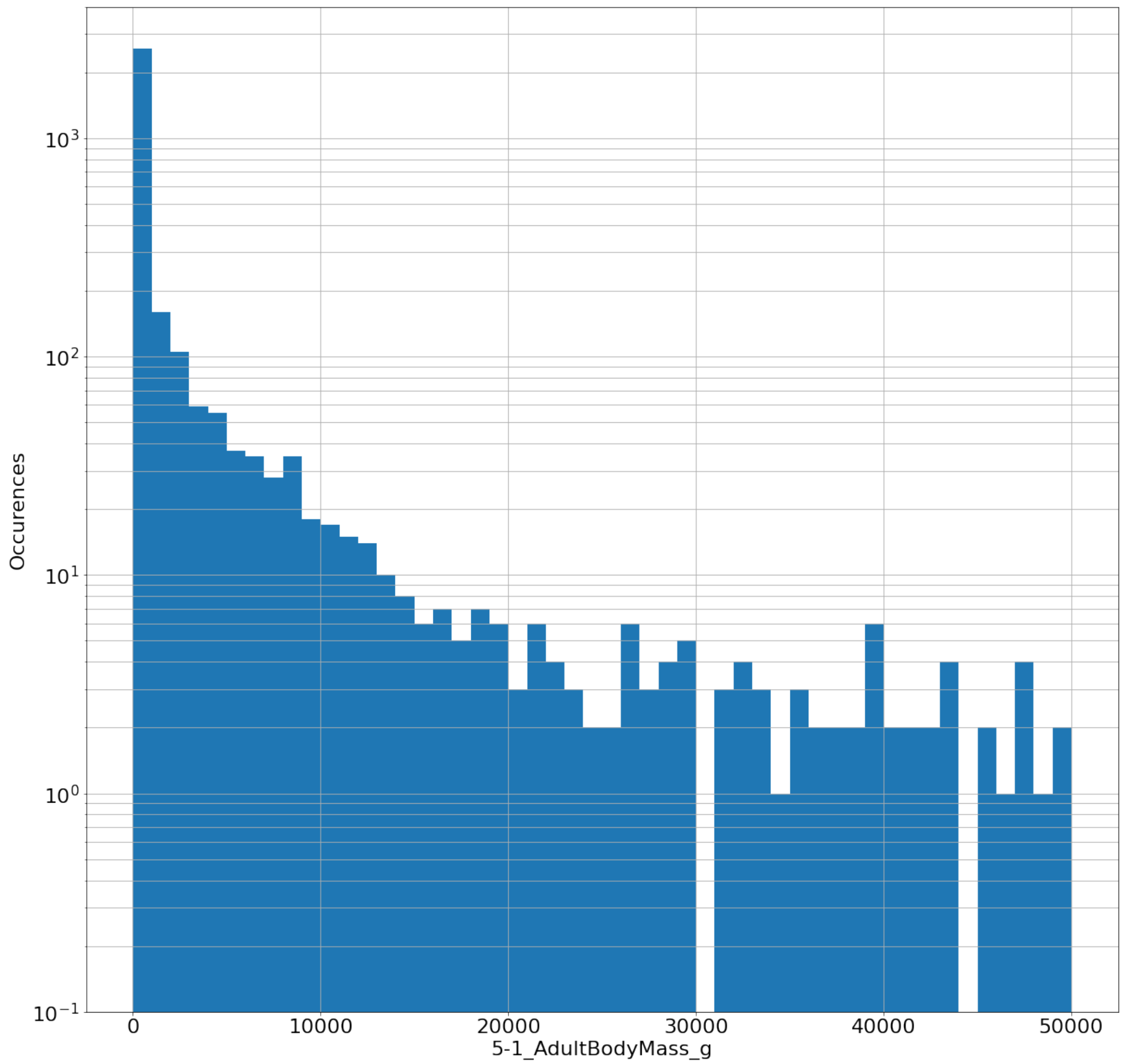
Observations **"Q6 : What are your conclusion about this histogram ?"** : - With it, we can see that there is a bigger quantity of light animals than heavy ones. - Also, most of the measures have been taken with animals lighter than 10 tonnes.

1.1.6 II.5) Histogram of Body masses that don't exceed 50[kg]

[45]:
```
## Set size of the graph
plt.figure(figsize=(20,20))

## Set font size around graph
plt.rcParams.update({'font.size': 22})

## Show grid through graph
plt.grid(True, which="both")
```

```python
## Set y scale
plt.ylim(0.1, 4000)

## X/Y titles :
plt.xlabel("5-1_AdultBodyMass_g")
plt.ylabel("Occurences")

## Show
plt.hist(ls_neg_filter, bins=50, log=True, range=[0, 50000])
```

[45]:
```
(array([2.577e+03, 1.590e+02, 1.050e+02, 5.900e+01, 5.500e+01, 3.700e+01,
        3.500e+01, 2.800e+01, 3.500e+01, 1.800e+01, 1.700e+01, 1.500e+01,
        1.400e+01, 1.000e+01, 8.000e+00, 6.000e+00, 7.000e+00, 5.000e+00,
        7.000e+00, 6.000e+00, 3.000e+00, 6.000e+00, 4.000e+00, 3.000e+00,
        2.000e+00, 2.000e+00, 6.000e+00, 3.000e+00, 4.000e+00, 5.000e+00,
        0.000e+00, 3.000e+00, 4.000e+00, 3.000e+00, 1.000e+00, 3.000e+00,
        2.000e+00, 2.000e+00, 2.000e+00, 6.000e+00, 2.000e+00, 2.000e+00,
        2.000e+00, 4.000e+00, 0.000e+00, 2.000e+00, 1.000e+00, 4.000e+00,
        1.000e+00, 2.000e+00]),
 array([    0.,  1000.,  2000.,  3000.,  4000.,  5000.,  6000.,  7000.,
         8000.,  9000., 10000., 11000., 12000., 13000., 14000., 15000.,
        16000., 17000., 18000., 19000., 20000., 21000., 22000., 23000.,
        24000., 25000., 26000., 27000., 28000., 29000., 30000., 31000.,
        32000., 33000., 34000., 35000., 36000., 37000., 38000., 39000.,
        40000., 41000., 42000., 43000., 44000., 45000., 46000., 47000.,
        48000., 49000., 50000.]),
 <BarContainer object of 50 artists>)
```



Observations **“Q7 : What are your conclusion about those histograms ?”** : - We can see that between ~**15kg and 50kg**, we have **a balanced magnitude** of occurences. - Under 10kg, there is a lot more occurences. - **The peak is under 1kg**.

1.2 Part III. - Benford’s law

1.2.1 III.1) Take animals’ body masses as adult (5-1__AdultBodyMass__g) after filter of missing and unrealistic values.

[46]:
```
ls_neg_filter[:10]
```

[46]:
```
[492714.47,
 10392.49,
 9658.7,
 11989.1,
```

```
31756.51,
800143.05,
500000.0,
635974.34,
1117.02,
897.67]
```

1.2.2 III.2) Extract most significant digits

[47]:
```
## Recover most significant digit of each value in <ls_neg_filter>
ls_msd = [int(x) // (10**floor(log10(x))) for x in ls_neg_filter]
## Check result list
print(ls_msd[:10])
```

```
[4, 1, 9, 1, 3, 8, 5, 6, 1, 8]
```

Most Significant Digit : **Checked with the 2 last cells !**

1.2.3 III.3) Reckon frequency of msd (most significant digit)

[48]:
```
## ls of each percent by index 1 to 9
ls_percent = [(ls_msd.count(x) / len(ls_msd) * 100) for x in range(1,10)]

## print list
for x in range(1, 10):
    print(f'{x} : {ls_percent[x-1]:.2f}%')
```

```
1 : 27.58%
2 : 18.52%
3 : 10.70%
4 : 11.32%
5 : 7.51%
6 : 6.86%
7 : 7.03%
8 : 6.13%
9 : 4.35%
```

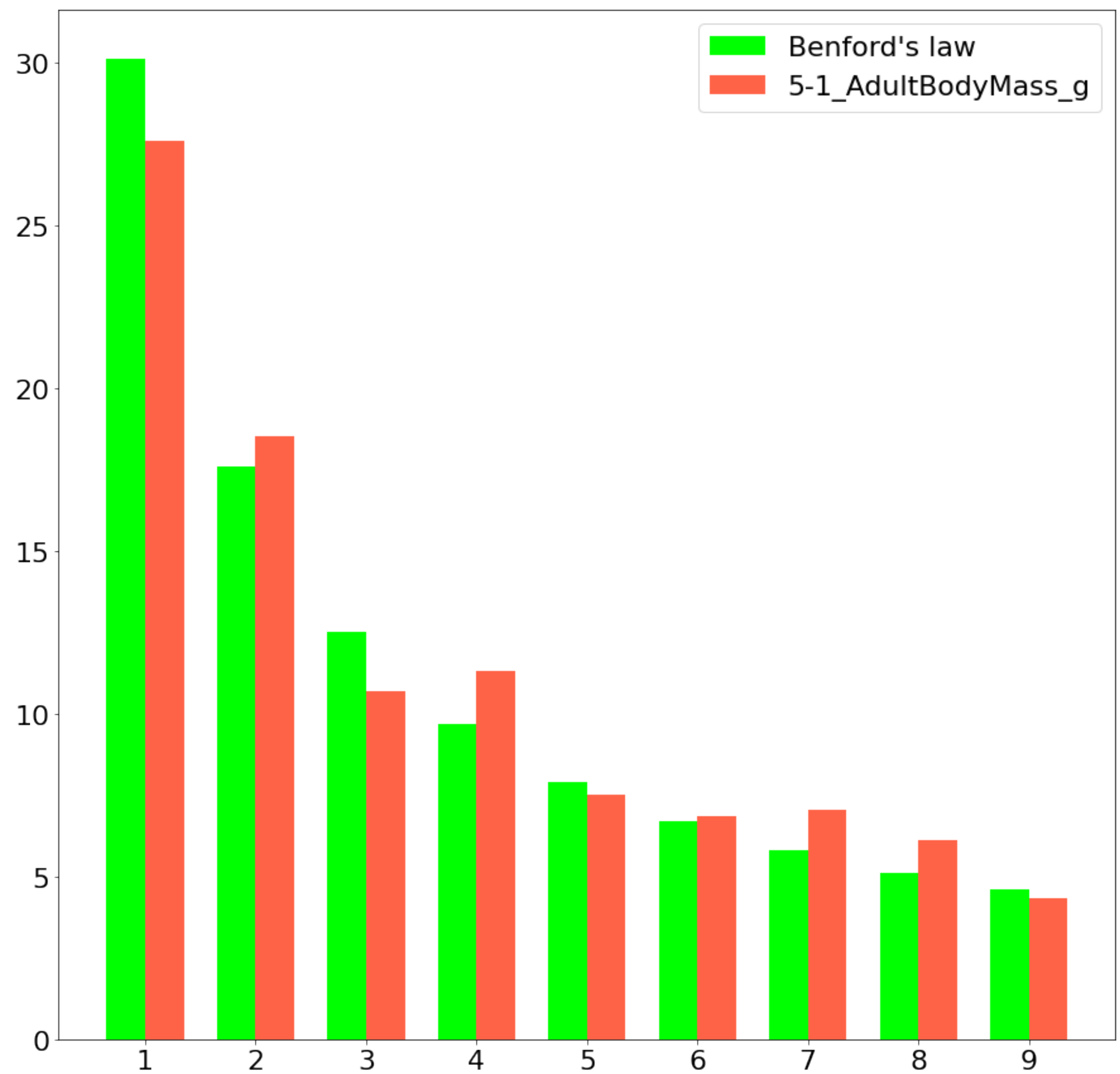
1.2.4 III.4) barchart : Comparison of our result and Belford's law

[49]:
```
ls_Benford = [30.1, 17.6, 12.5, 9.7, 7.9, 6.7, 5.8, 5.1, 4.6]
#ls_idx = [*range(1,10,1)]
ls_idx = np.arange(1,10,1)
## Checked index list
#print(ls_idx)
```

[50]:
```
## Set bar width
barWidth=0.35
```

```python
## Set plot size
plt.figure(figsize=(15,15))

## Bar datas :
plt.bar(ls_idx - barWidth/2, ls_Benford, color='lime', width=barWidth,␣
 ↪label='Benford\'s law')
plt.bar(ls_idx + barWidth/2, ls_percent, color='tomato', width=barWidth,␣
 ↪label='5-1_AdultBodyMass_g')

## Legend :
ax = plt.gca()
ax.set_xticks(ls_idx)
plt.legend()
```

[50]: <matplotlib.legend.Legend at 0x7ff4adecb400>



Observation **"Q8 : What can be observed ? Comment your result."** :
We are pretty close to the Benford's law values. They are not exactly the same, but we have the same magnitude than the law.

1.2.5 III.5) Comparison with another column (5-3_NeonateBodyMass_g for example)

[51]:
```python
## MINIMUM :␣
↪--------------------------------------------------------------------------->
print(f'Minimum = {df_ani["5-3_NeonateBodyMass_g"].min():.3f}[g]')
LOWER_LIMIT = 0
if df_ani["5-1_AdultBodyMass_g"].min() < LOWER_LIMIT:
    ## List with only realistic values (under 0 deleted) :
    ls_neg_5_3 = [x for x in df_ani["5-3_NeonateBodyMass_g"] if x >=␣
↪LOWER_LIMIT]
    ## Check filtered list :
    #print(ls_neg_filter)
    print(f'Minimum (filtered) = {min(ls_neg_5_3):.3f}[g]')

## MAXIMUM :␣
↪--------------------------------------------------------------------------->
print(f'Maximum = {max(ls_neg_5_3):.3f}[g] = \
{max(ls_neg_5_3)/1000:.3f}[kg] = \
{max(ls_neg_5_3)/1000000:.3f}[t]')

## AVERAGE :␣
↪--------------------------------------------------------------------------->
print(f'Average = {statistics.mean(ls_neg_5_3):.3f}[g]   = \
{statistics.mean(ls_neg_5_3)/1000:.3f}[kg]   = \
{statistics.mean(ls_neg_5_3)/1000000:.3f}[t]')

## MEDIAN  :␣
↪--------------------------------------------------------------------------->
print(f'Median  = {statistics.median(ls_neg_5_3):.3f}[g]')
```

```
Minimum = -999.000[g]
Minimum (filtered) = 0.004[g]
Maximum = 2738612.790[g] = 2738.613[kg] = 2.739[t]
Average = 10553.737[g]   = 10.554[kg]   = 0.011[t]
Median  = 13.690[g]
```

[52]:
```python
ls_neg_5_3[:10]
```

[52]:
```
[36751.19,
 211.82,
 200.01,
```

```
412.31,
22977.05,
73.6,
0.2,
34346.64,
5875.0,
177.2]
```

[53]:
```
## Recover most significant digit of each value in <ls_neg_5_3>
ls_msd_5_3 = [int(x) // (10**floor(log10(x))) for x in ls_neg_5_3]
## Check result list
print(ls_msd_5_3[:10])
```

```
[3, 2, 2, 4, 2, 7, 0.0, 3, 5, 1]
```

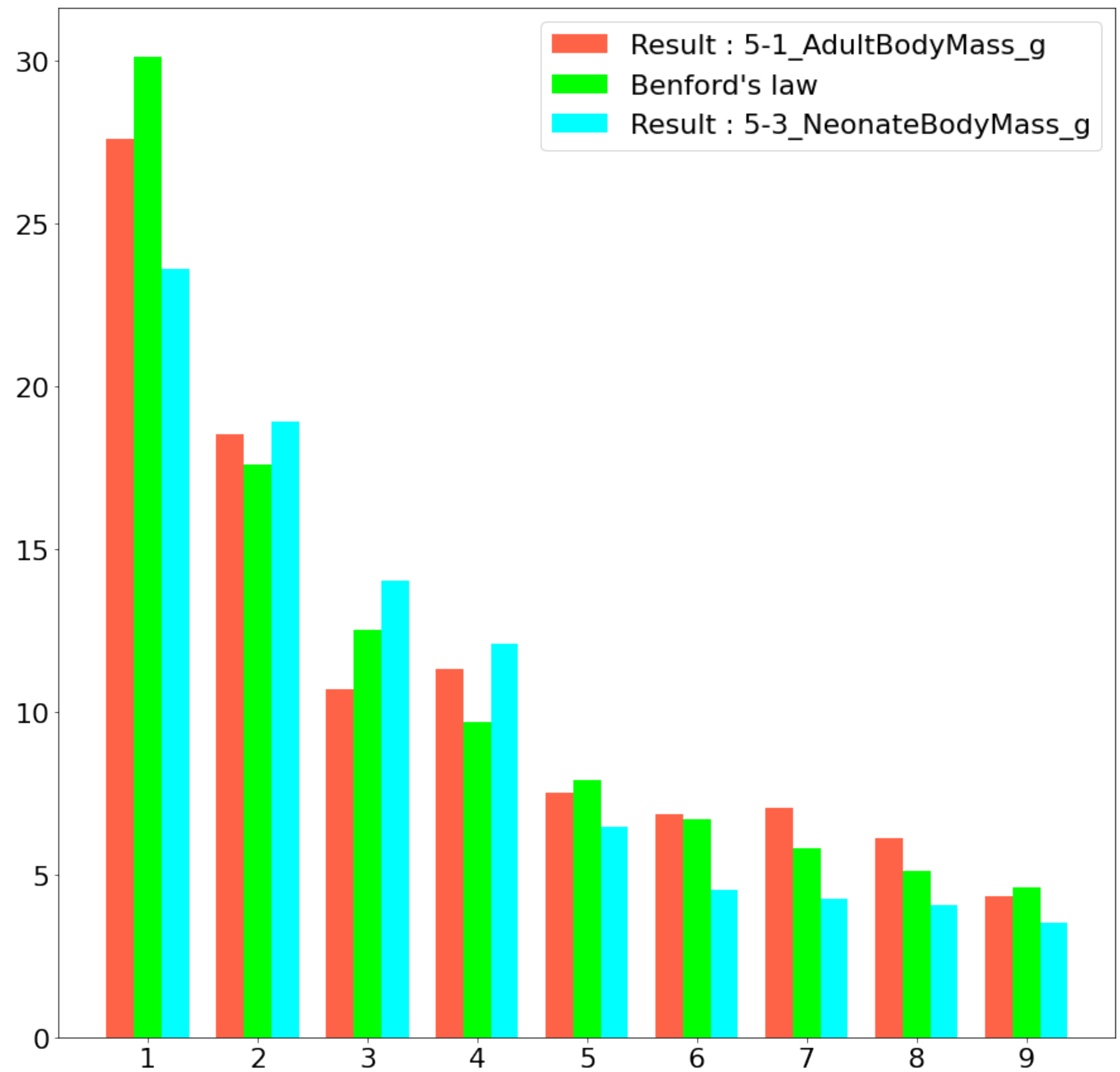
[54]:
```
## ls of each percent by index 1 to 9
ls_percent_5_3 = [(ls_msd_5_3.count(x) / len(ls_msd_5_3) * 100) for x in␣
↪range(1,10)]

## Print list
for x in range(1, 10):
    print(f'{x} : {ls_percent_5_3[x-1]:.2f}%')
```

```
1 : 23.59%
2 : 18.89%
3 : 14.01%
4 : 12.07%
5 : 6.45%
6 : 4.52%
7 : 4.24%
8 : 4.06%
9 : 3.50%
```

[55]:
```
## Reset bar width
barWidth=0.25

## Reset plot size
plt.figure(figsize=(15,15))

## Bar datas :
plt.bar(ls_idx - barWidth, ls_percent, color='tomato', width=barWidth,␣
↪label='Result : 5-1_AdultBodyMass_g')
plt.bar(ls_idx, ls_Benford, color='lime', width=barWidth, label='Benford\'s␣
↪law')
plt.bar(ls_idx + barWidth, ls_percent_5_3, color='cyan', width=barWidth,␣
↪label='Result : 5-3_NeonateBodyMass_g')

## Legend :
```

```
ax = plt.gca()
ax.set_xticks(ls_idx)
plt.legend()
```

[55]: <matplotlib.legend.Legend at 0x7ff4ade317f0>



Observation **"Q9 : Comment your result."** :
Like before, values are **not identical**, but they're **close to Benford's law**.

```
[ ]:
```

You could do functions to avoid copy-pasting all the code a second time.