

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/345213754>

Descriptive Analysis

Chapter · January 2017

DOI: 10.4018/978-1-68318-016-6.ch004

CITATION

1

READS

14,436

2 authors:



Rui Sarmiento

79 PUBLICATIONS 210 CITATIONS

SEE PROFILE



Vera Costa

University of Porto

24 PUBLICATIONS 120 CITATIONS

SEE PROFILE

Descriptive Analysis

INTRODUCTION

Descriptive statistics is the initial stage of analysis used to describe and summarize data. The availability of a large amount of data and very efficient computational methods strengthened this area of the statistic.

R VS. PYTHON

In order to make a descriptive analysis correctly, the first step should be to identify the variable type. In the following sections, suggestions for this analysis are presented, in R and Python languages.

Categorical variables

Categorical variables are qualitative variables and cannot be represented by a number. In the data set in the study, the Gender of the researchers, R_user (yes vs. no) and Python_user (yes vs. no) are categorical variables. In particular, they are nominal categorical variables. The difference between nominal and ordinal categorical variables is that, regarding their presentation, while nominal variables may be presented randomly or in the preferred order of the analyst, the ordinal variables must be presented in the order that is more easily understood. For example, the school levels must be shown by order, i.e., from the lowest level to the highest.

In case of categorical variables, the analysis that can be done is the frequency of each category. In R, this count is given by the *table* function. In Python, the *value_counts()* function gives these values. A suggestion of the descriptive analysis for the Gender variable, in the programming languages mentioned above would be:

	In R
Code	<pre>### Gender's descriptive analysis # Count of each factor level of the Gender variable (presented in a data frame "data.df" with a "Gender" #column), and conversion of the count to numeric values Freq.Gender <- as.numeric(table(data.df[, "Gender"])) # Cumulative frequencies of the "Freq.Gender" object CFreq.Freq.Gender <- cumsum(Freq.Gender) # Relative frequencies for each factor level of the Gender variable and conversion of the frequencies as # numeric values Rel.Freq.Gender <- as.numeric(prop.table(Freq.Gender)) # Data frame with the realized analysis Freqs.Gender <- data.frame(Gender = levels(factor(data.df[, "Gender"])), Frequency = Freq.Gender, Cumulative.Frequency = CFreq.Freq.Gender, Relative.Frequency = Rel.Freq.Gender) # Output the previous results Freqs.Gender</pre>
Output	<pre>Gender Frequency Cumulative.Frequency Relative.Frequency 1 female 87 87 0.435 2 male 113 200 0.565</pre>

Table 3

In Python	
Code	<pre> #### Gender descriptive analysis # Count of each factor level of the Gender variable print(datadf['Gender'].value_counts()) # Filtering Gender data gender_datadf = datadf['Gender'] # Group by Gender value gender_datadf = pd.DataFrame(gender_datadf.value_counts(sort=True)) # Create new column with cumulative sum gender_datadf['cum_sum'] = gender_datadf['Gender'].cumsum() # Create new column with relative frequency gender_datadf['cum_perc'] = 100*gender_datadf['cum_sum']/gender_datadf['Gender'].sum() gender_datadf </pre>
Output	<pre> Out[130]: Gender cum_sum cum_perc male 113 113 56.5 female 87 200 100.0 </pre>

Table 4

From the previous outputs (frequency column in R or Gender column in Python), there are 87 female and 113 male cases in the study, corresponding to 43.5% and 56.5% (relative frequencies column in R or cum_perc column in Python) respectively. The cumulative frequencies column shows that, in total, there are 200 elements in the study. The mode (most common element) of this variable is “male”.

To analyze the Python_user variable (a categorical variable), a similar study to the study of the Gender variable was also done.

In R	
Code	<pre> #### Python_user descriptive analysis # Count of each factor level of the Python_user variable and conversion of the count as numeric values Freq.Python <- as.numeric(table(data.df[, "Python_user"])) # Cumulative frequencies of the “Freq.Python” object CFreq.Freq.Python <- cumsum(Freq.Python) # Relative frequencies for each factor level of the Python_user variable and conversion of the frequencies as numeric values Rel.Freq.Python <- as.numeric(prop.table(Freq.Python)) # Data frame with the executed analysis Freqs.Python <- data.frame(Python.user = levels(factor(data.df[, "Python_user"])), Frequency = Freq.Python, Cumulative.Frequency = CFreq.Freq.Python, Relative.Frequency = Rel.Freq.Python) Freqs.Python </pre>
Output	<pre> Python.user Frequency Cumulative.Frequency Relative.Frequency 1 no 92 92 0.4623116 2 yes 107 199 0.5376884 </pre>

Table 5

	In Python
Code	<pre> #### Python_user descriptive analysis # Filtering Python_user data python_datadf = datadf['Python_user'] # Group by Python_user python_datadf = pd.DataFrame(python_datadf.value_counts(sort=True)) # Create new column with cumulative sum python_datadf['cum_sum'] = python_datadf['Python_user'].cumsum() # Create new column with relative frequency python_datadf['cum_perc'] = 100*python_datadf['cum_sum']/python_datadf['Python_user'].sum() python_datadf Freqs.Python </pre>
Output	<pre> Out[131]: Python_user cum_sum cum_perc yes 107 107 53.768844 no 92 92 100.000000 </pre>

Table 6

As it is possible to observe in the previous output, the Python programming language has 107 users. The number of non-users is 92. So, as the total number of researchers (Python users and non-users) is 199, the conclusion is that there is one missing.

To calculate the relative frequency, in R, the *prop.table* function is used. The function ignores missing values (or NA's). Thus, there are 46.2% of non-users and 53.8% of users. To include NA as a category in counts, the option *exclude=NULL* should be included in the *table* function. In Python, by default, the NA's are also excluded. To include NA in the counts, in function *value_counts*, the option *dropna=False* should be added. Thus, the previous code should be replaced by the next.

	In R
Code	<pre> #### Python_user descriptive analysis Freq.Python <- as.numeric(table(data.df[, "Python_user"], exclude=NULL)) CFreq.Freq.Python <- cumsum(Freq.Python) Rel.Freq.Python <- as.numeric(prop.table(Freq.Python)) Freqs.Python <- data.frame(Python.user = levels(factor(data.df[, "Python_user"], exclude = NULL)), Frequency = Freq.Python, Cumulative.Frequency = CFreq.Freq.Python, Relative.Frequency = Rel.Freq.Python) Freqs.Python </pre>
Output	<pre> Python.user Frequency Cumulative.Frequency Relative.Frequency 1 no 92 92 0.460 2 yes 107 199 0.535 3 <NA> 1 200 0.005 </pre>

Table 7

	In Python
--	-----------

Code	<pre> ### Python_user descriptive analysis print(datadf['Python_user'].value_counts()) python_datadf = datadf['Python_user'] python_datadf = pd.DataFrame(python_datadf.value_counts(sort=True, dropna =False)) python_datadf['cum_sum'] = python_datadf['Python_user'].cumsum() python_datadf['cum_perc'] = 100*python_datadf['cum_sum']/python_datadf['Python_user'].sum() python_datadf </pre>
Output	<pre> Out[8]: Python_user cum_sum cum_perc yes 107 53.5 no 92 99.5 NaN 1 100.0 </pre>

Table 8

With the previous code, the number of missing values and corresponding relative frequency is given. Thus, the Python_user variable has one missing, corresponding to 0.5% of the sample. Also, there are 92 non-users (46%) and 107 users (53.5%) of Python.

The mode (most frequent element) of this variable is “yes”.

Similar to the previous variables, the analysis of the R_user variable could be done as presented below.

In R	
Code	<pre> ### R_user descriptive analysis # Count of each factor level of the R_user variable Freq.R <- as.numeric(table(data.df[, "R_user"])) # Cumulative frequencies of the “Freq.R” object CFreq.Freq.R <- cumsum(Freq.R) # Relative frequencies of each factor level of the R_user variable and conversion of the frequencies as numeric values Rel.Freq.R <- as.numeric(prop.table(Freq.R)) # Data frame with the realized analysis Freqs.R <- data.frame(R.user = levels(factor(data.df[, "R_user"])), Frequency = Freq.R, Cumulative.Frequency = CFreq.Freq.R, Relative.Frequency = Rel.Freq.R) Freqs.R </pre>
Output	<pre> R.user Frequency Cumulative.Frequency Relative.Frequency 1 no 91 91 0.455 2 yes 109 200 0.545 </pre>

Table 9

In Python

Code	<pre> ### R_user descriptive analysis print(datadf['R_user'].value_counts()) # Filtering R_user data r_datadf = datadf['R_user'] # Group by R_user r_datadf = pd.DataFrame(r_datadf.value_counts(sort=True)) # Create new column with cumulative sum r_datadf['cum_sum'] = r_datadf['R_user'].cumsum() # Create new column with relative frequency r_datadf['cum_perc'] = 100*r_datadf['cum_sum']/r_datadf['R_user'].sum() r_datadf </pre>
Output	<pre> out[132]: R_user cum_sum cum_perc yes 109 109 54.5 no 91 200 100.0 </pre>

Table 10

Regarding the R_user variable, there are 109 users (54.5%) and 91 non-users (45.5%). The mode (most frequent element) of this variable is “yes”.

	In R
Code	<pre> ### Tasks descriptive analysis # Count of each factor level of the variable Tasks Freq.Tasks <- as.numeric(table(data.df[, "Tasks"])) # Cumulative frequencies of the “Freq.Tasks” object CFreq.Freq.Tasks <- cumsum(Freq.Tasks) # Relative frequencies each factor level of the R_user variable and conversion of the frequencies as numeric values Rel.Freq.Tasks <- as.numeric(prop.table(Freq.Tasks)) # Data frame with the realized analysis Freqs.Tasks <- data.frame(Tasks = levels(factor(data.df[, "Tasks"])), Frequency = Freq.Tasks, Cumulative.Frequency = CFreq.Freq.Tasks, Relative.Frequency = Rel.Freq.Tasks) Freqs.R </pre>
Output	<pre> Tasks Frequency Cumulative.Frequency Relative.Frequency 1 PhD_Student 78 78 0.39 2 Phd_Supervisor 56 134 0.28 3 Postdoctoral_research 66 200 0.33 </pre>

Table 11

	In Python
--	-----------

Code	<pre>### Tasks descriptive analysis # Filtering Tasks data tasks_datadf = datadf['Tasks'] # Group by tasks tasks_datadf = pd.DataFrame(tasks_datadf.value_counts(sort=True)) # Create new column with cumulative sum tasks_datadf['cum_sum'] = tasks_datadf['Tasks'].cumsum() # Create new column with relative frequency tasks_datadf['cum_perc'] = 100*tasks_datadf['cum_sum']/tasks_datadf['Tasks'].sum() tasks_datadf</pre>																
Output	<pre>Out[133]:</pre> <table><thead><tr><th></th><th>Tasks</th><th>cum_sum</th><th>cum_perc</th></tr></thead><tbody><tr><td>PhD_Student</td><td>78</td><td>78</td><td>39.0</td></tr><tr><td>Postdoctoral_research</td><td>66</td><td>144</td><td>72.0</td></tr><tr><td>Phd_Supervisor</td><td>56</td><td>200</td><td>100.0</td></tr></tbody></table>		Tasks	cum_sum	cum_perc	PhD_Student	78	78	39.0	Postdoctoral_research	66	144	72.0	Phd_Supervisor	56	200	100.0
	Tasks	cum_sum	cum_perc														
PhD_Student	78	78	39.0														
Postdoctoral_research	66	144	72.0														
Phd_Supervisor	56	200	100.0														

Table 12

Regarding the individual's tasks, there are 78 Ph.D. Students, 56 Ph.D. Supervisors, and 66 Postdoc researchers, corresponding to 39%, 28%, and 33%, respectively.

Discrete Numerical Variables

As mentioned in the Statistics chapter, a discrete numeric variable only has distinct integer values. In this case, information such as mean, standard deviation, quartiles and median, are crucial since they show the general behavior of the variable in the study. In this context, this type of descriptive analysis was applied to Age and Publication variables.

In R, some commands can be used, namely, *mean*, *median*, *min*, *max*, *quantile* (*x*, 0.25), *quantile* (*x*, 0.75) and *sd*. However, the *summary* function gives all these values.

Thus, if the reader only needs the mean (for example), he/she can use the corresponding command. But, if he/she need all information about the variable, the *summary* function is more convenient. Please note that the function does not give the standard deviation of the numeric variable.

In Python, each measure should be calculated individually. Some functions like *len()*, *min()*, *max()*, *mean()*, *var()*, and *std()* are very useful.

	In R
Code	<pre> ### Age and number of publications descriptive analysis # Summary description of Age and Publications variables, selecting the columns by the corresponding name summary(data.df[,c("Age", "Publications")]) OR # Summary description of Age and Publications variables, selecting the columns by the corresponding position, i.e., fifth and sixth column summary(data.df[,c(5, 6)]) # Standard deviation of Age variable, removing missing values, represented by NA's sd(data.df[, "Age"], na.rm=TRUE) # Standard deviation of Publications variable, removing missing values, represented by NA's sd(data.df[, "Publications"], na.rm=TRUE) </pre>

Outputs	# Summary of Age and Publications variables
	Age Publications
	Min. :24.00 Min. :11.00
	1st Qu.:33.00 1st Qu.:25.00
	Median :37.00 Median :31.00
	Mean :37.06 Mean :29.65
	3rd Qu.:41.00 3rd Qu.:35.00
	Max. :52.00 Max. :70.00
	NA's :2
	# Standard deviation of Age variable and Publications variable
	[1] 5.637253
	[1] 7.743209

Table 13

	In Python
Code	<pre> ### Age and number of publications descriptive analysis # Import panda package import pandas as pd # Read data datadf = pd.read_csv('data.csv', sep=',') ## Age # To write the name of the output list print("\nAge Variable: \n") # Dimension of the Age variable print("Number of elements: {0:8.0f}".format(len(datadf['Age']))) # Minimum and maximum of the Age variable print("Minimum: {0:8.3f} Maximum: {1:8.3f}".format(datadf['Age'].min(), datadf['Age'].max())) # Mean of the Age variable print("Mean: {0:8.3f}".format(datadf['Age'].mean())) # Variance of the Age variable print("Variance: {0:8.3f}".format(datadf['Age'].var())) # Standard deviation of the Age variable print("Standard Deviation : {0:8.3f}".format(datadf['Age'].std())) ##Publications # To write the name of the output list print("\nPublications: \n") # Dimension of the Publications variable print("Number of elements: {0:8.0f}".format(len(datadf['Publications']))) # Minimum and maximum of the Publications variable print("Minimum: {0:8.3f} Maximum: {1:8.3f}".format(datadf['Publications'].min(), datadf['Publications'].max())) # Mean of the Publications variable print("Mean: {0:8.3f}".format(datadf['Publications'].mean())) # Variance of the Publications variable print("Variance: {0:8.3f}".format(datadf['Publications'].var())) # Standard deviation of the Publications variable print("Standard Deviation : {0:8.3f}".format(datadf['Publications'].std())) </pre>

Outputs	<p>Age Variable: Number of elements: 200 Minimum: 24.000 Maximum: 52.000 Mean: 37.056 Variance: 31.779 Standard Deviation: 5.637</p> <p>Publications: Number of elements: 200 Minimum: 11.000 Maximum: 70.000 Mean: 29.650 Variance: 59.957 Standard Deviation: 7.743</p>
---------	---

Table 14

With the previous outputs, it is possible to conclude that the variable Age has two missing values (NA's). For the valid values, the age of researchers varies between 24 and 52 years old. The mean (37.06 years) is quite close to the median (37 years), which suggests the non-existence of outliers. The mean is a measure greatly influenced by “large” or “small” values even if these values arise in small number in the sample. When these values (outliers) exist, the mean assumes very different/distant values from the median. This is not true in the case of Age variable.

The first and third quartiles are 33 and 41 years old, respectively. This means that 50% (half) of researchers present in the sample have ages between 33 and 41 years old.

The standard deviation is 5.64 years, that is, on average, the age of researchers varies about six years of the mean (37.06) of their ages.

Regarding the Publications variable, there are no missing values. The number of publications ranges from 11 to 70. It is also possible to check that the mean is 29.65, and the median is 31 publications per researcher. The mean and median are farther apart from each other than in the previous case (Age variable). This suggests the possibility of outliers or suspected outliers existence.

The first and third quartiles are 25 publications and 35, respectively, i.e., half of the researchers have between 25 and 35 publications.

The standard deviation of this variable is approximately 7.74 publications.

Continuous Numerical variables

A continuous numerical variable can take any numeric value within a specified interval. If the 200 researchers in this analysis were asked to indicate their height, the values should vary a lot. In this case, it is common to proceed with the creation of a set of intervals to group some values. The reader needs to define the size of these ranges. For example, if 500 height records are varying between 1.51m and 1.70m, the amplitude of each range should be small. Otherwise, all values fall into the same interval. If there are 500 salary records between € 1,000 and € 10,000, the amplitude of the interval should be, at least, 1000 or 2000 units.

To show how to work with continuous variables, the Publications variable will be regarded. This variable has been analyzed before. However, the frequencies of each number of publications are unknown. Thus, the frequency analysis (already presented for discrete variables) is provided below.

In R

Code	<pre> ### Frequency analysis of Publications variable # # Count of each factor level of the Publications variable Freq.Publications<-sort(as.numeric(table(data.df[, "Publications"])), decreasing=TRUE) # Cumulative frequencies of the "Freq.Publications" object CFreq.Freq. Publications <- cumsum(Freq. Publications) # Relative frequencies each factor level of the Publications variable Rel.Freq <- as.numeric(prop.table(Freq. Publications)) # Data frame with the realized analysis Freqs. Publications <- data.frame(Publications = levels(factor(data.df[, "Publications "])), Frequency = Freq. Publications, Cumulative.Frequency = CFreq.Freq. Publications, Relative.Frequency = Rel.Freq) </pre>				
	Output	Publications	Frequency	Cumulative.Frequency	Relative.Frequency
	1	11	15	15	0.075
	2	12	14	29	0.070
	3	13	13	42	0.065
	4	15	13	55	0.065
	5	16	12	67	0.060
	6	17	12	79	0.060
	7	18	10	89	0.050
	8	19	10	99	0.050
	9	21	9	108	0.045
	10	22	9	117	0.045
	11	23	8	125	0.040
	12	24	7	132	0.035
	13	25	7	139	0.035
	14	26	7	146	0.035
	15	27	6	152	0.030
	16	28	6	158	0.030
	17	29	5	163	0.025
	18	30	4	167	0.020
	19	31	4	171	0.020
	20	32	4	175	0.020
	21	33	4	179	0.020
	22	34	3	182	0.015
	23	35	2	184	0.010
	24	36	2	186	0.010
	25	37	2	188	0.010
	26	38	2	190	0.010
	27	39	2	192	0.010
	28	40	2	194	0.010
	29	41	2	196	0.010
	30	42	1	197	0.005
	31	44	1	198	0.005
	32	45	1	199	0.005
	33	70	1	200	0.005

Table 15

	In Python
--	-----------

Code	<pre> #### Frequency analysis of Publications variable # Filtering Publications data pubs_datadf = datadf['Publications'] # Group by publications pubs_datadf = pd.DataFrame(pubs_datadf.value_counts(sort=True)) # Create new column with cumulative sum pubs_datadf['cum_sum'] = pubs_datadf['Publications'].cumsum() # Create new column with relative frequency pubs_datadf['cum_perc'] = 100*pubs_datadf['cum_sum']/pubs_datadf['Publications'].sum() pubs_datadf </pre>
Output	<pre> Out[2]: Publications cum_sum cum_perc 31 15 15 7.5 33 14 29 14.5 29 13 42 21.0 25 13 55 27.5 26 12 67 33.5 36 12 79 39.5 39 10 89 44.5 34 10 99 49.5 35 9 108 54.0 32 9 117 58.5 21 8 125 62.5 22 7 132 66.0 24 7 139 69.5 28 7 146 73.0 30 6 152 76.0 38 6 158 79.0 18 5 163 81.5 37 4 167 83.5 40 4 171 85.5 16 4 175 87.5 15 4 179 89.5 19 3 182 91.0 12 2 184 92.0 27 2 186 93.0 41 2 188 94.0 23 2 190 95.0 42 2 192 96.0 44 2 194 97.0 17 2 196 98.0 13 1 197 98.5 70 1 198 99.0 45 1 199 99.5 11 1 200 100.0 </pre>

Table 16

As it is possible to observe, the number of publications per researcher varies widely. Therefore, two suggestions of the division with intervals and corresponding frequencies are presented below.

	In R
Code	<pre> #### Division at intervals and corresponding frequencies a) # Division of the interval in 11 equal parts, and the interval closed on right classIntervals(data.df[, "Publications"], n=11, style = "equal", rtimes = 3, intervalClosure = c("right"), dataPrecision = NULL) b) # Division at fixed intervals, i.e., the fixedBreaks indicates the limits of the intervals. The intervals should be closed on right. classIntervals(data.df[, "Publications"], n=11, style = "fixed", fixedBreaks=c(10, 20, 30, 40, 70), rtimes = 3, intervalClosure = c("right"), dataPrecision = NULL) </pre>

Outputs	<p>a) style: equal one of 64,512,240 possible partitions of this variable into 11 classes</p> <pre>[11,16.36364] (16.36364,21.72727] (21.72727,27.09091] (27.09091,32.45455] (32.45455,37.81818] (37.81818,43.18182] (43.18182,48.54545] (48.54545,53.90909] (53.90909,59.27273] (59.27273,64.63636] (64.63636,70]</pre> <p>b) style: fixed one of 4,960 possible partitions of this variable into 4 classes</p> <pre>[10,20] (20,30] (30,40] (40,70]</pre>
---------	---

Table 17

	In Python
Code	<pre>### Division at intervals and corresponding frequencies a) # Division of the interval in 11 equal parts, and the interval closed on right table = np.histogram(datadf['Publications'], bins=11, range=(0, 70)) print(table) b) # Division at fixed intervals, i.e., the buckets object indicates the limits of the intervals. The intervals should be closed on right. buckets = [0,10, 20, 30, 40,70] table = np.histogram(datadf['Publications'], bins=buckets) print(table)</pre>
Outputs	<p>a) (array([0, 3, 19, 37, 55, 64, 20, 1, 0, 0, 1], dtype=int64), array([0., 6.36363636, 12.72727273, 19.09090909, 25.45454545, 31.81818182, 38.18181818, 44.54545455, 50.90909091, 57.27272727, 63.63636364, 70.]))</p> <p>b) array([0, 22, 71, 95, 12], dtype=int64), array([0, 10, 20, 30, 40, 70]))</p>

Table 18

In R, in the first case a), 11 intervals are defined. In this case, the amplitude of each interval is fixed. The reader needs only to indicate the number of ranges that he/she wants to consider. To point out that symbol “(” means interval opened, and “]” means range closed. The intervals [11,16], [16,22], [22,27], [27,32], [32,38], [38,43], [43,49], [49,54], [54,59], [59,65], [65,70] could be considered (round to units). However, this is not a good solution because there are many classes with frequencies equal to one. Thus, some intervals, which seem to make more sense, must be considered.

In the second case b), four intervals are defined. Although it appears that the intervals have different dimensions, it is contemplated that the ends (first and last) include the corresponding infinities, that is, the intervals are $]-\infty, 20]$, $[20, 30]$, $[30, 40]$, $[40, +\infty[$.

Similar to R, in Python, in the first case a), 11 intervals with equal dimension are considered. The first array gives the frequencies of publications in each interval. The defined intervals (rounded to units) are [0,6], [6,13], [13,19], [19,25], [25,32], [32,38], [38,45], [45,51], [51,57], [57,64], [64,70]. As there are many classes with frequencies equal to one, the better solution is to set the intervals (presented in b)) manually.

Graphical Representation

To summarize data in a visual way, charts and/or graphs are a good option. Depending on the data type, different graphs should be used. We will explain with some examples.

Pie chart

The pie chart to represent nominal variables graphically. The data in the study has some variables of this type, to remember, Gender, Python_user, R_user, and Tasks. We provide the pie charts of some of these variables.

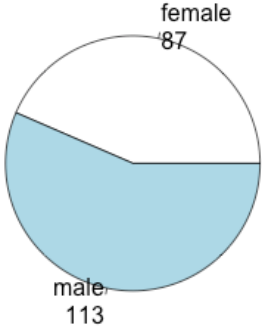
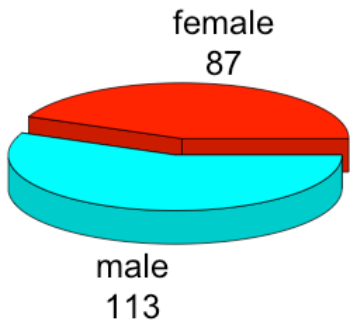
	In R
Code	<pre> #### Gender Pie Chart # Frequency of the Gender variable mytable <- table(data.df[, "Gender"]) # Labels of each pie slice. Past gender classes with their frequencies lbls <- paste(names(mytable), "\n", mytable, sep="") a) # Graph with labels lbls and a name of the pie chart pie(mytable, labels = lbls, main="Pie Chart of Gender Variable\n (with sample sizes)") OR b) # A 3D Graph with labels lbls, spacing between the slices (input <i>explode</i>) and a name of the pie chart library(plotrix) pie3D(mytable, labels = lbls, explode=0.1, main="Pie Chart of Gender Variable\n (with sample sizes)") </pre>
Outputs	<div style="display: flex; justify-content: space-around;"> <div style="width: 45%;"> <p>a)</p> <p style="text-align: center;">Pie Chart of Gender Variable (with sample sizes)</p>  <p style="text-align: center;"><i>Figure 39 Pie Chart of Gender Variable in R</i></p> </div> <div style="width: 45%;"> <p>b)</p> <p style="text-align: center;">Pie Chart of Gender Variable (with sample sizes)</p>  <p style="text-align: center;"><i>Figure 40 3D Pie Chart of Gender Variable in R</i></p> </div> </div>

Table 19

	In Python
--	-----------

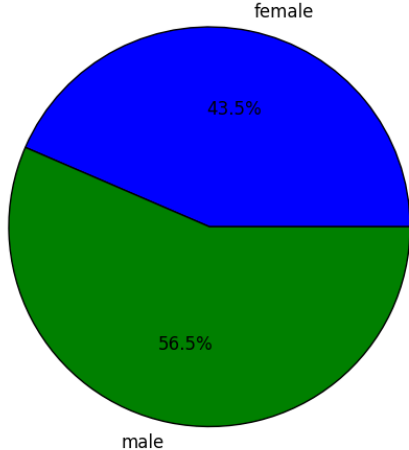
Code	<pre> #### Pie Chart Gender # Import packages matplotlib.pyplot and pandas import matplotlib.pyplot as plt import pandas as pd # Pie chart labels label_list = datadf['Gender'].value_counts(sort=False).index # Plot pie chart axis plt.axis("equal") # The pie chart is oval by default. To make it a circle use pyplot.axis("equal") # To show the percentage of each pie slice, pass an output format to the autopct parameter (rounding to 1 decimal place). plt.pie(datadf['Gender'].value_counts(sort=False), labels=label_list, autopct="%1.1f%%") plt.title("Researchers Gender") plt.show() </pre>
Output	<p style="text-align: center;">Researchers Gender</p>  <p style="text-align: center;">male</p> <p style="text-align: center;">Figure 41 Pie Chart of Gender Variable in Python</p>

Table 20

In R, the more traditional version of a pie chart is shown in output a). Note that the names of each slice should be expressly mentioned. Otherwise, an image without a caption is displayed.

Some packages have been created to make better illustrations of this type of graph. This is the case of package *plotrix* that allows doing 3D pie charts, as shown in output b).

As it is possible to visualize, the size of each slice is proportional to the length of each factor level of the variable. The biggest slice represents the number of male researchers, and the smaller shows the number of female researchers. Thus, the graph indicates that there are 113 males (56.5%) and 87 females (43.5%). Similarly to R, in Python, the specification of the legend is also required. Also, the pie chart is oval by default. To change it, just indicate that both axes have equal scales. This is a condition that provides a circle pie chart.

The values of frequencies (and percentage), as expected, coincide with the relative frequencies analyzed at the beginning of this chapter.

Regarding the Python_user variable, similar pie charts will be done next.

	In R
--	------

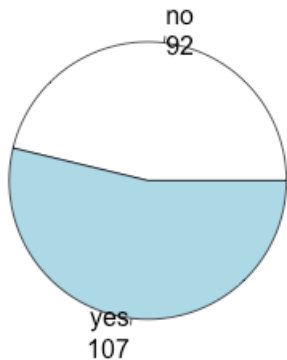
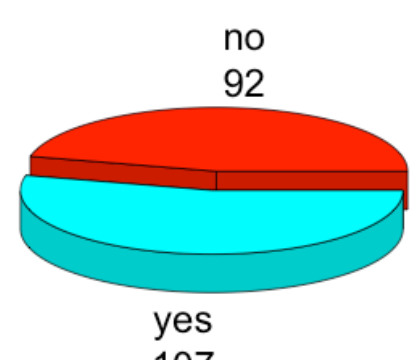
Code	<pre> #### Python_user a) # Graph with labels lbls and a name of the pie chart mytable <- table(data.df[, "Python_user"]) lbls <- paste(names(mytable), "\n", mytable, sep="") pie(mytable, labels = lbls, main="Pie Chart of Python users Variable\n (with sample sizes)") b) # A 3D Graph with labels lbls, spacing between the slices (input <i>explode</i>) and a name of the pie chart library(plotrix) pie3D(mytable, labels = lbls, explode=0.1, main="Pie Chart of Python users Variable\n (with sample sizes)") </pre>
Outputs	<div style="display: flex; justify-content: space-around;"> <div style="width: 45%;"> <p>a)</p> <p>Pie Chart of Python users Variable (with sample sizes)</p>  <p>Figure 42 Pie Chart of Python_user Variable in R</p> </div> <div style="width: 45%;"> <p>b)</p> <p>Pie Chart of Python users Variable (with sample sizes)</p>  <p>Figure 43 3D Pie Chart of Python_user Variable in R</p> </div> </div>

Table 21

Code	<pre> In Python #### Python_user # Pie Chart Python_user import matplotlib.pyplot as plt import pandas as pd # Pie chart labels label_list = datadf['Python_user'].value_counts(sort=False).index plt.axis("equal") #The pie chart is oval by default. To make it a circle use pyplot.axis("equal") # To show the percentage of each pie slice, pass an output format to the autopct parameter plt.pie(datadf['Python_user'].value_counts(sort=False), labels=label_list, autopct=" %1.1f%%") plt.title("Researchers Python Users") plt.show() </pre>
------	---

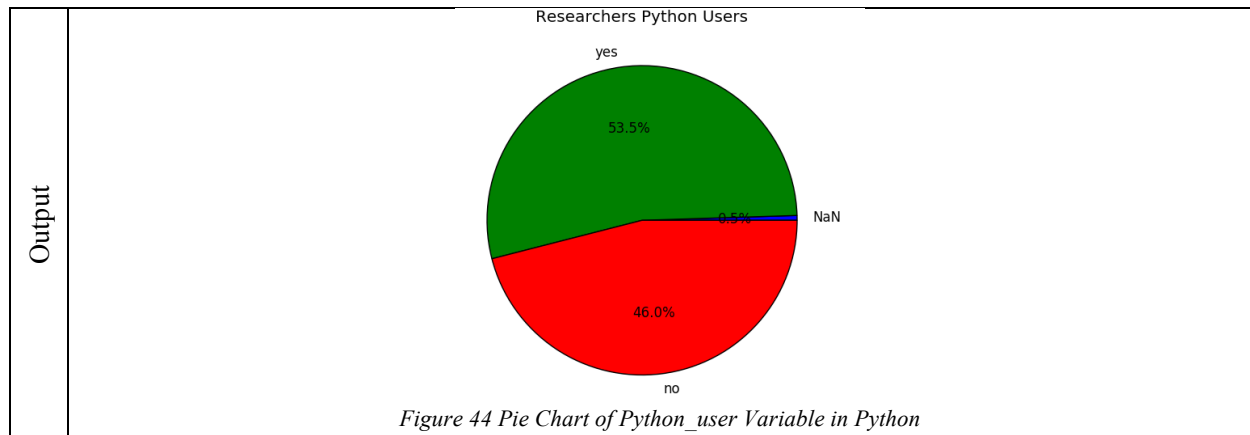


Table 22

Regarding the Python_users variable, the R pie charts above show that there are 107 users ($\approx 54\%$) and 92 non-users ($\approx 46\%$). We should point out that missing values are not represented. They are just eliminated. In Python, missing values are represented in the pie chart. Thus, the Python output shows that there are 53.5% of users, 46% of non-users and 0.5% of missing values. If the number of absence answers is not of reader's interest, the correspondent users could be deleted, or a condition in the plot should be inserted. A similar pie chart for the R_users variable is provided next:

	In R
Code	<pre>#### R_user a) # Graph with labels lbls and a name of the pie chart mytable <- table(data.df[, "R_user"]) lbls <- paste(names(mytable), "\n", mytable, sep="") pie(mytable, labels = lbls, main="Pie Chart of R users Variable\n (with sample sizes)") b) # A 3D Graph with labels lbls, spacing between the slices (input <i>explode</i>) and a name of the pie chart library(plotrix) pie3D(mytable, labels = lbls, explode=0.1, main="Pie Chart of R users Variable\n (with sample sizes)")</pre>

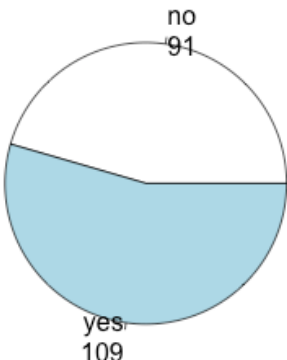
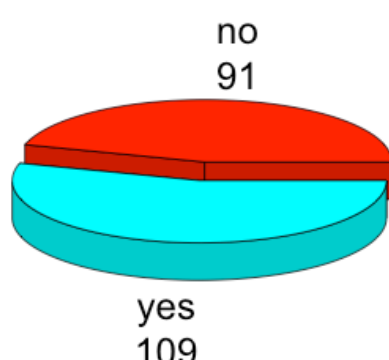
Outputs	<div style="display: flex; justify-content: space-around;"> <div style="text-align: center;"> <p>a)</p> <p>Pie Chart of R users Variable (with sample sizes)</p>  <p><i>Figure 45 Pie Chart of R_user Variable in R</i></p> </div> <div style="text-align: center;"> <p>b)</p> <p>Pie Chart of R users Variable (with sample sizes)</p>  <p><i>Figure 46 3D Pie Chart of R_user Variable in R</i></p> </div> </div>
---------	--

Table 23

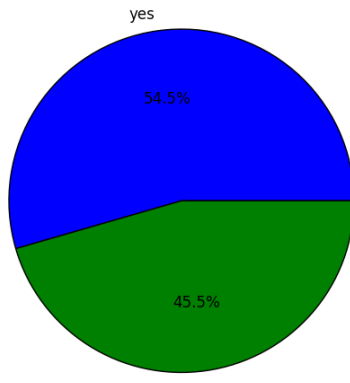
Code	<p>In Python</p> <pre> ### R_user # Pie Chart R_user import matplotlib.pyplot as plt import pandas as pd # Pie chart labels label_list = datadf['R_user'].value_counts(sort=False).index plt.axis("equal") #The pie chart is oval by default. To make it a circle use pyplot.axis("equal") # To show the percentage of each pie slice, pass an output format to the autopct parameter plt.pie(datadf['R_user'].value_counts(sort=False), labels=label_list, autopct="%1.1f%%") plt.title("Researchers R Users") plt.show()</pre>
Output	<p style="text-align: center;">Researchers R Users</p>  <p style="text-align: center;"><i>Figure 47 Pie Chart of R_user Variable in Python</i></p>

Table 24

The R pie charts above show that there are 109 R users (54.5%) and 91 non-users (45.5%). In the case of this variable (R_users), there are no missing values. The results presented in the pie chart correspond to the entire sample. As expected, the values are the same as initially obtained.

Bar graph

Similar to pie charts, the bar graphs are very useful in case of discrete variables, namely nominal variables. In the case of Tasks variable, either a pie chart or a bar graph is suitable. An example of a bar graph might be:

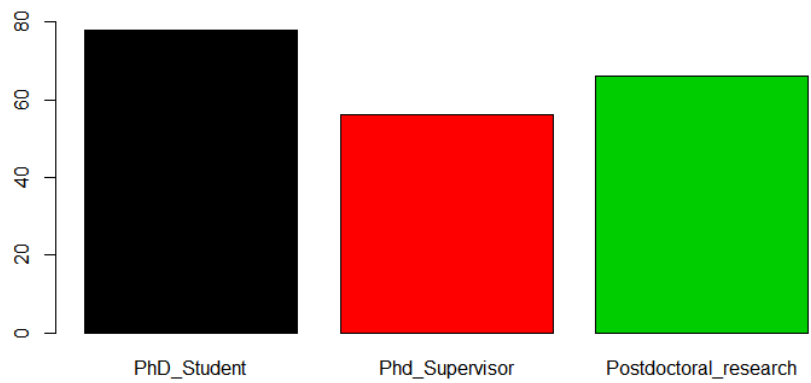
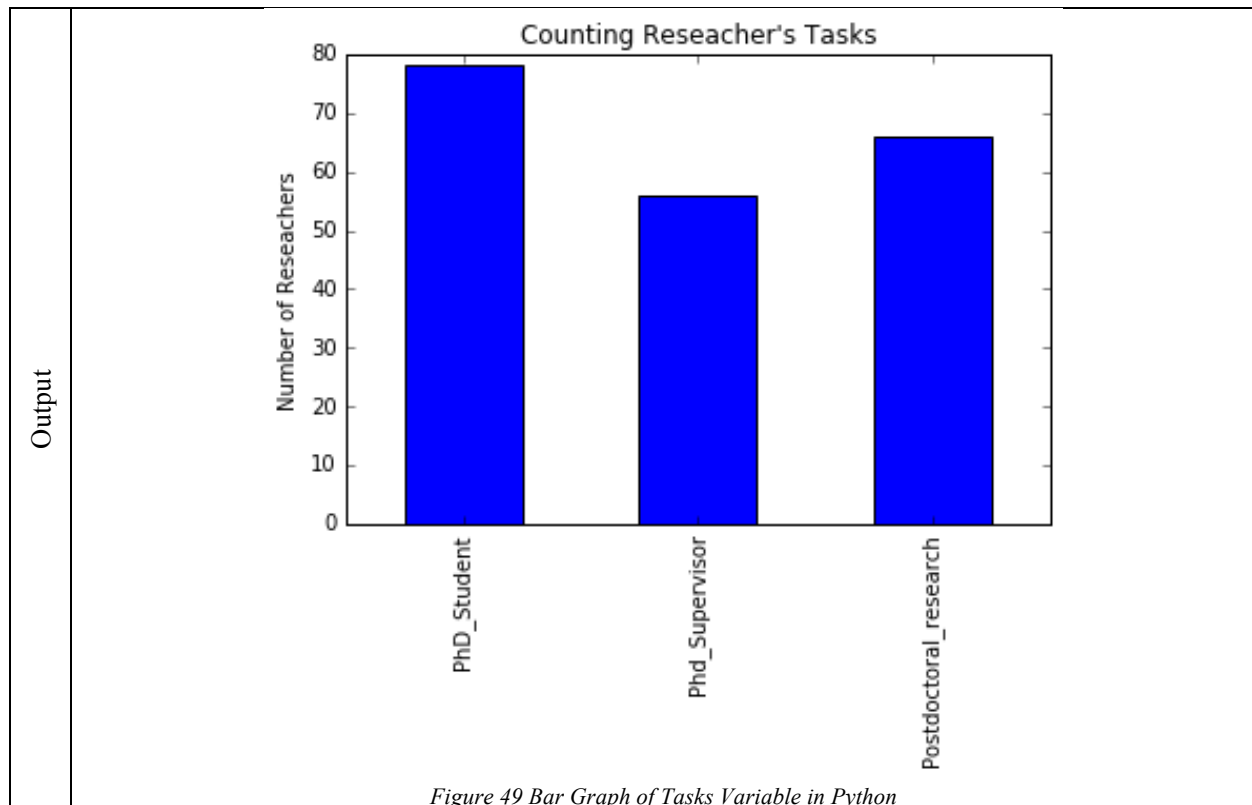
	In R
Code	<pre>### Tasks Bar Graph # Graph with different colors in the bars barplot(table(data.df[, "Tasks"]), col=c(1,2,3), ylim = c(0, 80))</pre>
Output	 <p>Figure 48 Bar Graph of Tasks Variable in R</p>

Table 25

	In Python
Code	<pre>### Tasks Bar Graph # Grouped sum of Tasks var = datadf['Tasks'].value_counts(sort=False) # Tasks Bar Graph plt.figure() # Setting y-axis label plt.ylabel('Number of Reseachers') # Setting graph title plt.title("Counting Reseachers's Tasks") # Trigger Bar Graph var.plot.bar() # Show Bar Graph plt.show()</pre>



In the previous outputs, a bar graph of the Tasks variable is represented. The x-axis represents the different factor levels of the Tasks variable. In the y-axis, the frequencies of each factor level. The code to create this bar graph specifies the length of the y-axis and the maximum is 78. Thus, the largest bar represents the number of Ph.D. Students. The Ph.D. Supervisors are shown in the shortest bar, in which the maximum frequency is near 60. Finally, the number of Postdoctoral researchers is near 70.

Boxplots

One of the best ways to represent numerical variables is using a boxplot. This type of graph allows creating a general idea about the variable since some critical values are given. Median, quartiles, maximum, minimum and outliers (if it exists) can be observed. We provide the boxplot of Age and Publications variables.

	In R
Code	<pre>### Boxplot of Age variable # Boxplot with title and name of the y-axis boxplot(data.df[, "Age"], data=data.df, main="Scientific Researchers Data", xlab="", ylab="Age of Researchers")</pre>

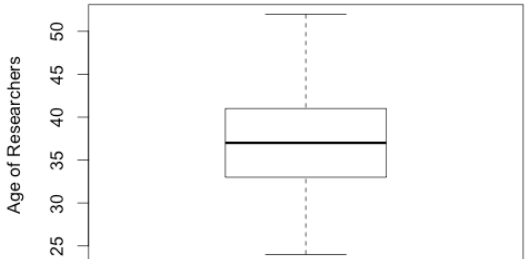
Output	<p style="text-align: center;">Scientific Researchers Data</p>  <p style="text-align: center;">Figure 50 Boxplot of Age Variable in R</p>
--------	--

Table 27

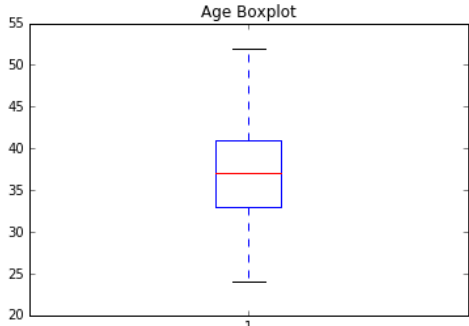
In Python	
Code	<pre> ### Boxplot of Age variable # First we have to take NaN values and substitute them for the mean of the variable datadf['Age']=datadf['Age'].replace('nan',datadf['Age'].mean()) # Importing modules import matplotlib.pyplot as plt import pandas as pd # Setting Figure fig=plt.figure() ax = fig.add_subplot(1,1,1) # Triggering Boxplot Chart ax.boxplot(datadf['Age'],showfliers=True, flierprops = dict(marker='o', markerfacecolor='green', markersize=12,linestyle='none')) # Setting Plot Title plt.title('Age Boxplot') # Show Plot plt.show() </pre>
Output	 <p style="text-align: center;">Figure 51 Boxplot of Age Variable in Python</p>

Table 28

In R, a boxplot is created using the *boxplot* function. If there are no other input parameters in addition to the data being analyzed, the output only returns the graph without subtitles. To be given the chart or axes names, some inputs such as *main*, *xlab* and *ylab* should be set. In Python, first we have to take NaN values and remove or substitute them for the mean of the variable.

In the boxplots above, it appears that Age variables distributed more or less evenly among near 25 and 50 years. Furthermore, the median is slightly over to 35 years. A boxplot of this type may indicate a high probability of this variable having a normal distribution.

Following this, we provide the boxplot of the Publications variable.

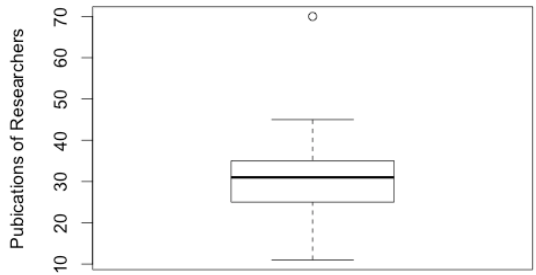
	In R
Code	<pre>### Boxplot of Publications variable # Boxplot with title and name of the y-axis boxplot(data.df[, "Publications"], data=data.df, main="Scientific Researchers Data", xlab="", ylab="Publications of Researchers")</pre>
Output	 <p>Figure 52 Boxplot of Publications Variable in R</p>

Table 29

	In Python
Code	<pre>### Boxplot of Publications variable # Import packages matplotlib.pyplot and pandas import matplotlib.pyplot as plt import pandas as pd # Plot boxplot and create one or more subplots using add_subplot, because you can't create blank figure fig=plt.figure() ax = fig.add_subplot(1,1,1) # Triggering Boxplot Chart ax.boxplot(data['Publications'], showfliers=True, flierprops = dict(marker='o', markerfacecolor='green', markersize=12, linestyle='none')) # Boxplot Title plt.title('Publications Boxplot') # Show Boxplot plt.show()</pre>

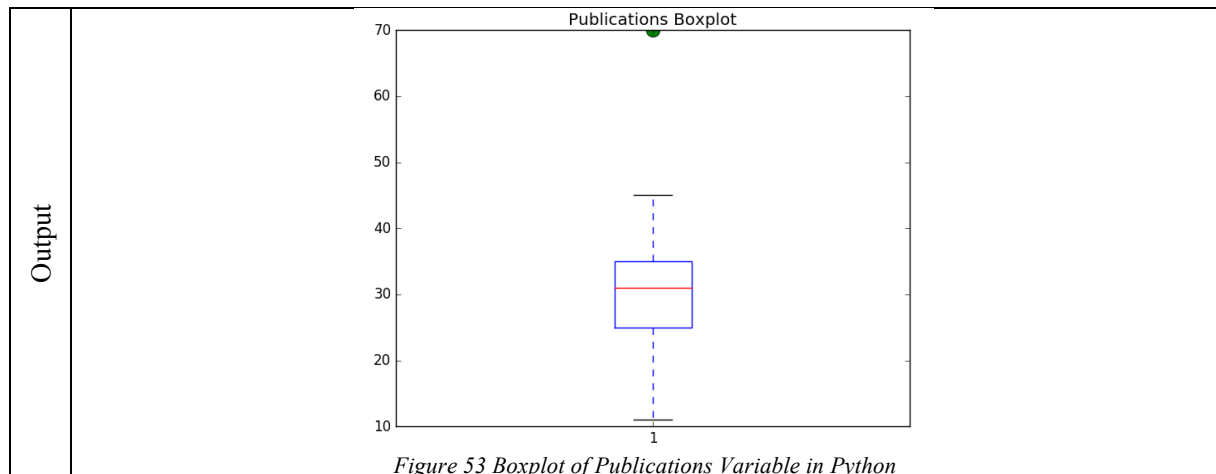


Table 30

In R, the boxplot of the Publications variable shows an outlier, represented by a small circle. This means that there is only one person with a high number of publications when compared with the others. Also, the average of publications is around 30.

The distribution of this variable seems to be quite irregular.

In Python, by default, the outliers are removed. If the analysts need to visualize them, the command `showfliers=True` and some particularities of the `flierprops` command should be included.

The Python's boxplot above, similar to R, shows the variation of the Publications variable, such as median, maximum, minimum and quartiles.

A boxplot is not only useful to represent variables individually. It is possible to cross two variables x_1 and x_2 to check the behavior of x_2 , depending on x_1 , or vice versa. In this sense, we represented a boxplot showing the number of Publications depending on the Age of the researchers.

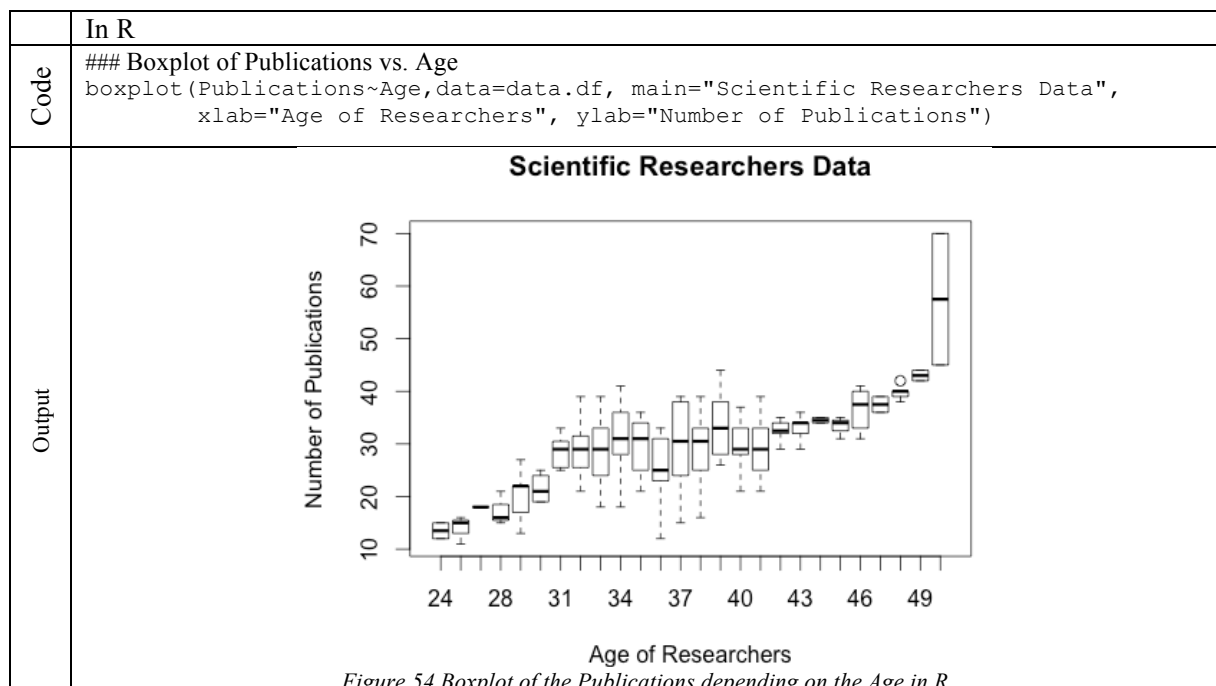


Table 31

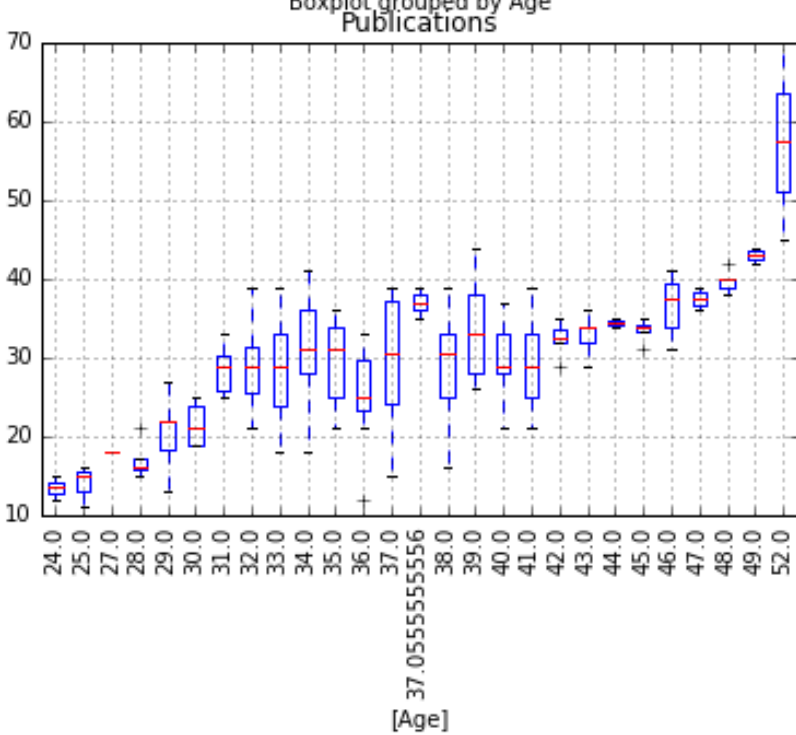
	In Python
Code	<pre> #### Boxplot of Publications vs. Age # Import pandas package import pandas as pd # First we have to take NaN values and substitute them for the mean of the variable datadf['Age']=datadf['Age'].replace('nan',datadf['Age'].mean()) # Get only Publications and Age variables new_datadf = datadf.ix[:,['Age','Publications']] # Boxplot new_datadf.boxplot(by='Age',rot = 90) </pre>
Output	 <p style="text-align: center;">Figure 55 Boxplot of the Publications depending on the Age in Python</p>

Table 32

The figures above show that there is an increasing trend in the number of publications along the age, i.e., the senior researchers have a tendency to have a higher number of publications than younger researchers. Also, we can additionally see that the number of publications of researchers between 30 and 40 years old, and more than 50 years old, varies more than in other age intervals.

Histogram

As the intervals of the number of Publications are continuous, a histogram should be used as a visual representation. The big difference between the histogram and the bar graph essentially is in the x-axis. Instead, on the bar graph, the frequencies are represented in separate bars, the histogram has continuous bars. In both programming languages some particularities can be specified, namely the axis labels and the main label of the histogram. Otherwise, the output only returns the picture without labels.

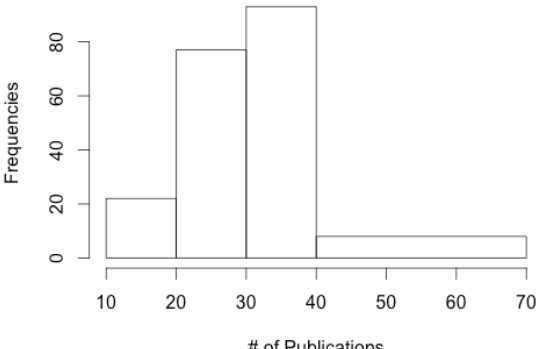
In R	
Code	<pre>### Histogram of the number of Publications (defined by classes) hist(data.df[, "Publications"], breaks=c(10, 20, 30, 40, 70), right = TRUE, freq=TRUE, ylab = "Frequencies", xlab="# of Publications", main="Histogram for Publications", include.lowest=TRUE)</pre>
Output	<p style="text-align: center;">Histogram for Publications</p>  <p style="text-align: center;"><i>Figure 56 Histogram of the Publications in R</i></p>

Table 33

In Python	
Code	<pre>### Histogram of the number of Publications # Publications Histogram fig=plt.figure() #Plots in matplotlib reside within a figure object, use plt.figure to create new figure # Create one or more subplots using add_subplot, because you can't create blank figure ax = fig.add_subplot(1,1,1) # Variable ax.hist(datadf['Publications'], facecolor='green') # Here it is possible to change the number of bins # Limits of x axis ax.set_xlim(0, 70) # Limits of y axis ax.set_ylim(0, 80) # Set grid on ax.grid(True) # Labels and Title plt.title('Publications distribution') plt.xlabel('Publications') plt.ylabel('#Reseachers') # Finally, show plot plt.show()</pre>

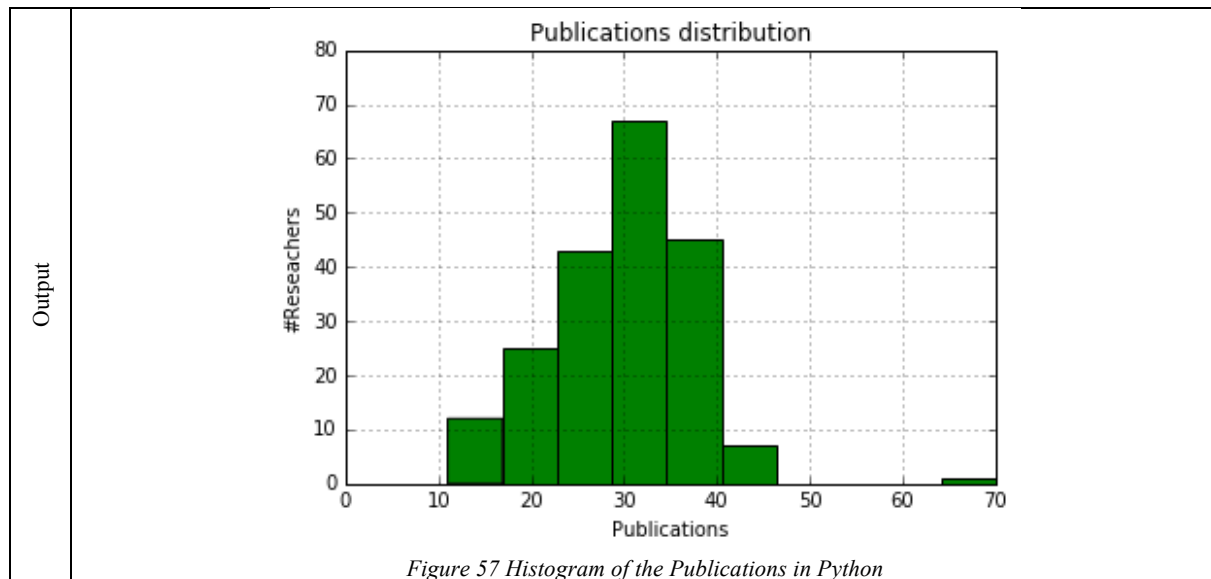


Table 34

The histograms show the distribution of frequencies along intervals of the number of Publications. It is possible to verify that the highest frequency corresponds to the interval of 30 and 40 publications. Since the last bar includes researchers publishing from 40 to 70 articles, this bar is wider (in R's output). However, the frequency of researchers with 40 or more publications is only around 10.

How to present results?

Regardless the programming language, the presentation of results is crucial. Mostly, the outputs provide a lot of information. Some of this information is not of the reader's interest, or it is not necessary to be demonstrated. On the other hand, depending on how this information is synthesized, it can become easier or more difficult to read.

Thus, in this sense, a presentation of the analysis made throughout this chapter is proposed. Note that, graphically, the user only presents the chart that considers more appropriate. The remaining analysis can be systematized in a table like the following.

	N	%
Gender		
Male	113	56.5
Female	87	43.5
Python users*	107	53.7
R users	109	54.5
Tasks		
PhD Student	78	39
PhD Supervisor	56	28
Postdoctoral Research	66	33
Age** (min-max)	24 – 52	
Mean (SD)	37.06 (5.64)	
Median (IQR)	37.00 (8.00)	
Number of Publications (min-max)	11 – 70	
Mean (SD)	29.65 (7.74)	
Median (IQR)	31.00 (10.00)	

Number of Publications (intervals)		
$]-\infty, 20]$	22	11.0
$[20, 30]$	77	38.5
$[30, 40]$	93	46.5
$[40, +\infty[$	8	4.0

* N=199; **N=198; IQR – Interquartile Range; SD – Standard Deviation

Table 35

CONCLUSIONS

At the end of this chapter, the reader should be able to identify the type of variable he/she wants to study. Moreover, the reader should be able to make a descriptive analysis of variables, in both R and Python. Additionally, a way to visualize each of several variables type is presented. Succinctly, we approached the following concepts:

- Variables Frequencies' Analysis
- Measures of Central Tendency and Dispersion
 - Median
 - Mean
 - Standard Deviation
 - Max
 - Min
 - Quartiles
- Outlier and missing values
- Graphical representation of variables
 - Pie Chart
 - Bar Graph
 - Boxplot
 - Histogram