

▼ DESCRIPTIVE STATISTICS

by Dr Juan H Klopper

- Research Fellow
- School for Data Science and Computational Thinking
- Stellenbosch University



▼ INTRODUCTION

Datasets hide information. There is a hidden story inside of data. It is our duty to learn as much about this information as possible.

We need a strategy to extract the information in the data. Unfortunately, humans are not good at seeing this information when staring at rows and columns of random variables. Instead, we start the journey of discovery by summarising the data. Calculating single values that are representative of the whole. Values that we understand and can interpret.

We start the journey in this notebook, which is all about **descriptive statistics** (summary statistics), as well as **comparative descriptive statistics**.

▼ PACKAGES FOR THIS NOTEBOOK

The following packages will be used in this notebook.

```
1 import pandas as pd # Data import and manipulation
```

```
1 from scipy import stats # A great statistical module from the scipy package
```

```
1 from google.colab import drive # Import a file in Google Drive
```

```
https://colab.research.google.com/drive/1PU1FJ-kQwW\_ZLzrKqUOZDQ\_7kWpXpYuu#scrollTo=5gFaqrCcGo7&printMode=true
```

```
1 # Format tables printed to the screen (don't put this on the same line as the code)
2 %load_ext google.colab.data_table
```

▼ DATA IMPORT

▼ CONNECTING TO GOOGLE DRIVE AND IMPORTING DATA

We mount our Google Drive and navigate to the directory containing the data.

```
1 # Log on and list files in the DATA directory of your Google Drive
2 drive.mount('/gdrive')
3 %cd '/gdrive/My Drive/Stellenbosch University/School for Data Science and Computat
4 %ls
```

```
Mounted at /gdrive
/gdrive/My Drive/Stellenbosch University/School for Data Science and Computat
australia_rain.csv      crops.csv              DefaultMissingData.csv
bitcoin_ethereum.csv  customers.csv         kaggle_survey_2020_responses.csv
breast_cancer.csv     data.csv              MissingData.csv
client_data.csv        DatesTimes.csv       montague_gardens_construction.csv
```

The `data.csv` file in the current folder is imported and assigned to the computer variable `df`.

```
1 df = pd.read_csv('client_data.csv') # Import the csv file
```

```
1 df # Display the dataframe to the screen
```

index	first_name	last_name	age	children	email	job_title	home_lo
0	Murry	Benfield	27	4	mbenfield0@youtube.com	Senior Financial Analyst	false
1	Coleman	Saladino	43	1	csaladino1@trellian.com	Physical Therapy Assistant	false
2	Cinderella	Kermannes	65	3	ckermannes2@webeden.co.uk	Health Coach III	true
3	Wenona	Voss	26	3	wvoss3@storify.com	Quality Control Specialist	true
4	Frayda	Burles	32	1	fburles4@baidu.com	Programmer Analyst III	true
5	Ealasaid	Dewen	42	0	edewen5@dagondesign.com	Accountant IV	true
6	Courtenay	Jacqueme	63	2	cjacqueme6@thetimes.co.uk	VP Quality Control	true
7	Clara	Piggen	65	2	cpiggen7@alibaba.com	VP Accounting	true
8	Dulciana	Janata	29	3	djanata8@wunderground.com	Computer Systems Analyst IV	true
9	Dionysus	Lambeth	25	1	dlambeth9@woothemes.com	Sales Representative	true
10	Davidde	Lamont	50	1	dlamonta@cdc.gov	Database Administrator IV	true
11	Ilsa	Woolbrook	46	2	iwoolbrookb@twitter.com	Design Engineer	false
12	Riley	Gentry	44	2	rgentryc@printfriendly.com	Administrative Officer	true
13	Casey	Roughan	52	0	croughand@noaa.gov	Staff Accountant IV	false
14	Ragnar	Poletto	60	0	rpolettoe@aboutads.info	Database Administrator I	true
15	Ailyn	Biasetti	53	1	abiasettif@gmpg.org	Nurse Practitioner	false
16	Had	Stanlike	40	1	hstanlike@theatlantic.com	Research	true

The data contains customer information.

17	Lisha	Jarratt	30	4	ljarratt@sourceforge.net	Design	true
----	-------	---------	----	---	--------------------------	--------	------

EXAMINING THE DATAFRAME OBJECT

18	De	Callahan	40	0	dcallahan@...	Environmental	...
----	----	----------	----	---	---------------	---------------	-----

We investigate the data by looking at the number of subjects (rows) and the number of statistical variables (columns).

19						Associate	
----	--	--	--	--	--	-----------	--

`1 df.shape # Using the shape attribute`

`(1000, 11)`

24	Wvalls	Base	32	4	wvalls@seesaa.net	Executive	false
----	--------	------	----	---	-------------------	-----------	-------

We see 1000 customers in our data set, with data collected for 11 variables.

Let's have a look at all the statistical variables using the `columns` attribute of the dataframe object.

```
1 df.columns # Name of each column

Index(['first_name', 'last_name', 'age', 'children', 'email', 'job_title',
      'home_loan', 'more_than_one_vehicle', 'fin_lit_review', 'savings',
      'invest'],
      dtype='object')
```

Finally, we can view the data types of each variable (column).

```
1 df.dtypes # Pandas data type of each column

first_name          object
last_name           object
age                 int64
children            int64
email               object
job_title           object
home_loan           bool
more_than_one_vehicle  bool
fin_lit_review      object
savings             float64
invest              int64
dtype: object
```

The `object` notation indicates a categorical data type, `bool` indicates a Boolean data type (`True` or `False`), `int64` indicates an numerical data type (64-bit integer), and `float64` is also a numerical data type (64-bit floating point value / a number with decimal places).

▼ COUNTING

▼ FREQUENCIES

Counting how many times a sample space element of a categorical variable occurs is a good start in data analysis. In our dataframe object, we have an `invest` variable, indicating how likely a customer is to make an investment based on a questionnaire that they completed. Let's first see what the sample space elements are in this variable. The `unique` method will return an array of the unique elements it finds in a specified column.

```
1 df.invest.unique() # The .unique() method returns the sample space elements of

array([3, 1, 2, 5, 4])
```

Data on the customers indicates how likely they are to make an investment on a scale from 1 to 5.

We can now count how many times each of these elements appear in the `invest` column, using the `value_counts` method. This gives us the **frequency** of each element.

```
1 df.invest.value_counts() # Counting the number of times the unique values appear
2      2      210
3      5      208
4      4      201
5      1      192
6      3      189
7      Name: invest, dtype: int64
```

We can express the counts as a fraction, called a **relative frequency**, dividing the frequency by the total number of observations (rows with data). This is done by setting the `normalize` argument to `True`.

```
1 df.invest.value_counts(normalize=True) # Expressing the relative frequency
2      2      0.210
3      5      0.208
4      4      0.201
5      1      0.192
6      3      0.189
7      Name: invest, dtype: float64
```

We can multiply this by 100 to get a percentage.

```
1 df.invest.value_counts(normalize=True) * 100 # Expressing the relative frequency as a percentage
2      2      21.0
3      5      20.8
4      4      20.1
5      1      19.2
6      3      18.9
7      Name: invest, dtype: float64
```

▼ Exercise

The `children` column indicates how many children a customer has. Calculate the frequency with which each element appears.

```
1 # Frequency of number of children
```

▼ Solution

```
1 df.children.value_counts()


3    218
2    214
1    194
4    192
0    182
Name: children, dtype: int64
```

▼ GROUPED FREQUENCIES

We can calculate *combined frequencies* to consider comparative descriptive statistics. As an example, consider the `home_loan` and the `more_than_one_vehicle` variables. Both are binary categorical variables. We can create a **contingency table** to summarize both variables.

We can do this with the pandas `crosstab()` function.

```
1 pd.crosstab(df.home_loan, df.more_than_one_vehicle) # Row and column variable
```

1 to 2 of 2 entries 

home_loan	False	True
false	268	247
true	239	246

Show per page

The left-hand side of the table contains the sample space elements of the `home_loan` variable (listed first in the `crosstab` function). For each of the two elements, we note frequencies across the `more_than_one_vehicle` variable's two sample space elements.

▼ MEASURES OF CENTRAL TENDENCY (POINT ESTIMATES)

Measures of central tendency or **point estimates** are single values that are representative of a list of continuous numerical values or of a categorical variable. Here we discuss the arithmetic mean, the median, and the mode.

▼ ARITHMETIC MEAN (AVERAGE)

The **mean** or the **average** is simply the sum of all the continuous numerical variables divided by

Let's start learning about the information in our data by asking some questions.

- What is the mean age of all the customers?

A pandas series object has many useful methods that are geared towards summary statistics.

The `.mean()` method calculates the mean.

```
1 # Using the .mean() method
2 df.age.mean()
```

```
44.739
```

- What is the mean value of the customers' savings?

```
1 # Using alternative column (variable) reference
2 df['savings'].mean()
```

```
30136.492399999974
```

- What is the mean age of the customers who have a home loan (indicated as *True* in the `home_loan` column)?

We looked at conditional in the previous notebook, where we selected only certain rows in a pandas series.

```
1 # Using a conditional on the Smoke column
2 df[df.home_loan == True]['age'].mean()
```

```
44.37731958762887
```

- What about the mean age of the customers without a home loan?

```
1 # Using a conditional on the Smoke column
2 df[df.home_loan == False].age.mean()
```

```
45.07961165048544
```

We can save a lot of time and typing by calculating the age means for both home loan group, using the `groupby` method.

The `groupby` method can create groups from the unique elements in a column and then call a

```
1 # Use the .groupby() method
2 df.groupby('home_loan')['age'].mean()

home_loan
False    45.079612
True     44.377320
Name: age, dtype: float64
```

The mean age for each of the sample space elements in the `home_loan` column will be calculated. Very useful.

By the way, the `mean` method has some useful arguments. We can use `skipna=True` to skip over any missing values (this is the default behaviour of this method). We can also use `numeric_only=True` if there are data values that were not captured as numbers.

▼ GEOMETRIC MEAN

The **geometric mean** multiplies all the continuous numerical variables and then takes the n -th root of that product, where n is the number of values. At the beginning of this notebook we imported the `stats` module from the `scipy` library. It contains many functions that we will use in the statistical analysis of our data. The `gmean` function calculates the geometric mean. It can take a pandas series as argument.

```
1 stats.gmean(df.age)

43.13095250169657
```

▼ MEDIAN

The mean makes an assumption of the data values and that is that they should be normally distributed. We will learn much more about distributions later. For now, we can view the normal distribution as the familiar bell-shaped curve.

Not all data value for a continuous numerical value follow a nice bell-shaped curve. We can have quite different *shapes* (distributions) or many outliers (values that are way-off from all the others). In these cases, the mean is not a good representative summary of all the data values. Here, we rather use the median.

The **median** puts all the values in a sorted order. If there are an odd number of values, then the median is the middle value, such that half of all the values are less than and the other half

greater than the median. If there are an even number of values, then the mean of the middle two values is taken.

- What is the median savings of customers older than 50?

```
1 # Using the .median() function
2 df[df.age > 50]['savings'].median()

30177.08
```

This syntax can take some getting used to. We have learned about it in the previous notebook. Square bracket notation denotes indexing. The syntax then indicates the use of the `df` dataframe object. The indices that are included are decided by the `df.age > 50` code, called a conditional. Wherever this conditional returns a `True` value (age is greater than 50) the index value of that row is included. The `savings` column is then used to return a series object. Finally, we call the `median` method on the series object.

What about the median savings of those with a home loan and more than one vehicle?

We can use conditionals more than once by separating each by a pair of parentheses and an `&` or `|` symbol. This is called a compound statement. The question we ask of the data therefor also involves logic. More precisely, we have **and** and **or** questions. When using **and**, `&`, both sets of arguments must be true for the **compound statement** to be true.

```
1 df[(df.home_loan == True) & (df.more_than_one_vehicle == True)].savings.median()

30090.425
```

If we are interested in the median savings of those with either a home loan **or** with more than one vehicle, only one of these have to be true to be included. Note that if both are true, it also leads to inclusion. The `|` is used for *or* statements.

```
1 df[(df.home_loan == True) | (df.more_than_one_vehicle == True)].savings.median()

30350.53
```

▼ Exercise

Calculate the median age of the customers who have a home loan and no children.

```
1 df.columns
```

```
Index(['first_name', 'last_name', 'age', 'children', 'email', 'job_title',
      'home_loan', 'more_than_one_vehicle', 'fin_lit_review', 'savings',
      'invest'],
      dtype='object')
```

▼ Solution

```
1 df[(df.home_loan == True) & (df.children == 0)].age.median()

40.5
```

There is an alternative solution to this problem. The `loc` property takes as second argument, the column of interest. The first argument is reserved to select the rows.

```
1 df.loc[(df.home_loan == True) & (df.children == 0), 'age'].median()

40.5
```

▼ MODE

The last measure of central tendency that we will take a look at is the mode. The **mode** is used for categorical or discrete data types. It returns the value(s) that occurs most commonly (and is hence not fit for continuous numerical variables). If a single sample space element occurs most commonly, there is a single mode. Sometimes more than one sample space element shares the spoils. This variable is then **bimodal**. As you might imagine, there are terms such as **tri-modal** and **multi-modal**.

- What is the mode of the `children` variable?

We use the `value_counts` method to calculate the frequency. The `ascending` argument is set to `False` by default and the `sort` is set to `True`, such that we get the mode at the top.

```
1 df.children.value_counts()

3    218
2    214
1    194
4    192
0    182
Name: children, dtype: int64
```

▼ MEASURES OF DISPERSION (SPREAD)

Measure of dispersion give us an indication of how spread out our numerical data values are.

▼ STANDARD DEVIATION AND VARIANCE

The **standard deviation** can be understood as the average difference between each continuous numerical data value and the mean of that variable. Difference infers subtraction. As some values will be larger than the mean and some smaller, subtraction from the mean will lead to positive numbers and negative numbers. In fact, from the way we calculate the mean, if we sum up all these differences (so as to calculate a mean difference), we will get 0. To mitigate this, we square all of the differences. Squaring (multiplying by itself) returns positive values. Now we can sum all these values and divide by the number of values. This gives us the **variance**.

Variances are very useful in statistics. We need to express the spread in the same units as our variable for it to make sense as a summary statistic. The *age* variable had a unit of years. What then, is a years². Instead, we take the square root of the variance to get the standard deviation, now expressed in the same units as the variable and a true measure of the average difference between all the values and the mean.

The `std` method returns the standard deviation of a series object and the `var` method returns the variance.

- What is the standard deviation of the age of customers who have a home loan versus those who do not?

```
1 # Group by the Smoke column
2 df.groupby('home_loan')['age'].std()

home_loan
False      11.612228
True       11.833028
Name: age, dtype: float64
```

▼ Exercise

What is the variance in savings for those with a home loan or more than one vehicle?

▼ Solution

```
1 df.loc[(df.home_loan == True) | (df.more_than_one_vehicle == True), 'savings'].var()
```

23785369.166071285

▼ RANGE

The **range** is the difference between the minimum and the maximum value of a continuous numerical variable. The `min` and the `max` methods for series objects give these values. Let's see then how old our youngest and oldest customers are.

- What is the minimum age of all the customers?

```
1 # Using the .min() function
2 df.age.min()

25
```

- What is the maximum age of all the customers?

```
1 # Using the .max() function
2 df.age.max()

65
```

- What is the range in the age of all the customers?

We simply subtract the minimum value from the maximum value.

```
1 # Difference between maximum and minimum ages
2 df.age.max() - df.age.min()

40
```

▼ QUANTILES

We divided our continuous numerical variables up into two halves for the median. We can divide them up into quarters as well. In fact, we can divide it up at any percentage level from 0% to 100% (fraction of 0.0 to 1.0). Here 0.0 or 0% would be the minimum value and 1.0 or 100% would be the maximum value. Dividing the values up into these *bins* give us **quantiles**.

We can divide the values up into four bins with three values. These values are the **quartiles**.

The lowest of these three values (the **first quartile**), divides the data into two parts, with a quarter being lower than that value and three-quarters being higher. The second divides the data values equally (the median or **second quartile**). The third is a value that has three-quarters of the values less than and a quarter more than it (the **third quartile**)

- What are the quartile values for the age of all the customers?

The `quantile` method allows us to choose, as a fraction, any of these *cut-off* values. For the quartiles, we create a list `[0.25, 0.5, 0.75]`.

```
1 # Specifying the quartiles as fractions
2 df.age.quantile([0.25, 0.5, 0.75])

0.25    34.0
0.50    45.0
0.75    55.0
Name: age, dtype: float64
```

- What is the 95th percentile values in age of the customers with a home loan versus those that do not have a home loan?

```
1 df.groupby('home_loan')['age'].quantile(0.95)

home_loan
False    63.0
True     63.0
Name: age, dtype: float64
```

The **interquartile range** is the difference between the third and the first quartile.

- What is the interquartile range of the age of all the customers?

```
1 df.age.quantile(0.75) - df.age.quantile(0.25)

21.0
```

▼ Conclusion

We now know a lot more about our data. Be encouraged to learn even more by asking some questions about this mock business data set and see if you can calculate the required values.

✓ 0s completed at 13:54

