Biostatistics Fundamental

By: Mildred Monsivais

February 21, 2020

Table of Contents

Biostatistics Fundamental

Introduction:

- Python development very flexible
- Web development, app development

How to install the program?
- intel distribution for python
-select Mac OS either choose python 3.6 or 2.6
-https://software.intel.com/en-us/distribution-for-python/choose-download (link)
-or can also download it from anaconda between python 3.6 or 2.6

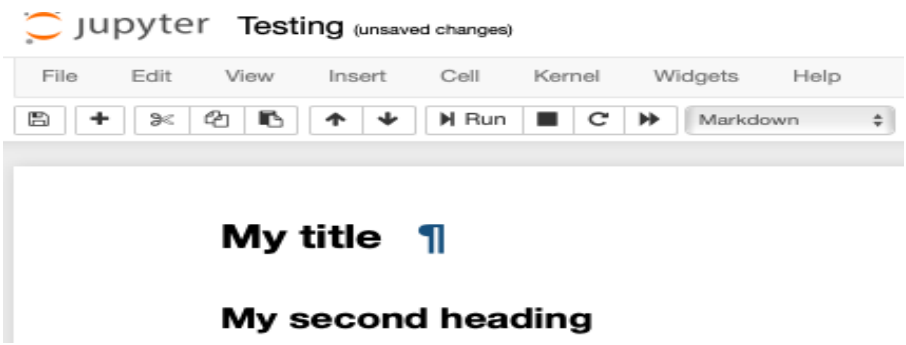
Opening Jupyter Notebook:
How to open your default browser>

- open your terminal
- Jupyter notebook

Jupyter notebook?
The notebook is an environment where all the coding is done. There are 4 dropdown next to the command palette. The 4 dropdown menu are code, markdown, raw NBConvert, and heading. The dropdown determine the behavior of the cell.

1. Code: cell in which code is written
2. Markdown: result are HTML and LaTeX outputs, can create a Title as it is unaffected cells. The markdown by hashtags(#) with a space makes the text size bigger.
    A. Hashtags

    Example below: # -> space -> write text
    - Can also make a second heading by adding 2 hashtags(##) which makes the text smaller then only one hashtag(#)
    - up to 6 hashtags(#)



.

B. Bullet points

- Markdown -> * -> space -> write text
- How to make sub-bullets: markdown -> tab -> * -> write text

It is like a word document where you can type in any notes between the line of cell.

Final results:

My title

My second heading

This is a normal line of text

- First bullet point
- Second bullet point
  - First subbullet point

3. Raw NBConvert
4. heading

What can you download the jupyter notebook as?

1. Notebook(.ipynb)
2. python(.py)
3. HTML(.html)
4. Reveal.js slides(.html)
5. markdown (.md)
6. reST(.rst)
7. LaTeX(.tex)
8. PDF via LaTex(.pdf)

Shutting down Jupyter Notebook:
A web server, in which the server is being connected to the websites using javascript.
The circle means that the notebook is being open at the moment.

- Save and Checkpoint -> Close and Halt
- it should say in terminal that shutting down 0 kernels

Packages vs Libraries:
1)Module: a file containing python statements and definitions
-a file with some python code with the ending of a .py
2)Package: A package is normally installed in python. Is a directory for sub packages and modules, collections of modules
-to help organize modules use the concept of packages
-think of a package as a directory and a module is a file within the directory
-a bunch of python file that ends with a .py
3)Library: A collection of a bunch of packages

Ex: Matplotlib, scipy, numpy, pandas
Module: a file containing python statements and definitions
-a file with some python code

Introducing Google Colab:

-Google's version of Jupyter notebook and looks like slightly modified jupyter notebook

- Google drive -> new -> more -> google colab

In order to use new library you have to upload it to the cloud first. In order to use pandas

How to import a file?

data_file = files.upload()

- choose the file you want to upload
- import pandas as pd
- df = pd.read_cvs('name of the file')
- df.head() -> shows you the first 5 columns

What If you want to use plotly?

```
[ ]  # Define a function for the use of Plotly in Colab
     def configure_plotly_browser_state():
       import IPython
       display(IPython.core.display.HTML('''
             <script src="/static/components/requirejs/require.js"></script>
             <script>
               requirejs.config({
                 paths: {
                   base: '/static/base',
                   plotly: 'https://cdn.plot.ly/plotly-1.5.1.min.js?noext',
                 },
               });
             </script>
             '''))
```

```
# Required code to run plotly in a cell
configure_plotly_browser_state()
init_notebook_mode(connected=False)

# Plotting
```

The advantage of using Google Colab?
Don't have to install anything in your hard drive and inside in the google drive. Can collaborate and share with others as they can add little comments.
Plotly:
Are interactive browsers and open inside of default browsers.

Doing Simple Arithmetic

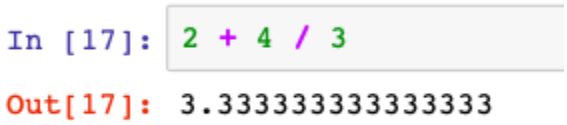
-Statistical analysis is arithmetic
Simple Arithmetic:
Instead of always typing in numpy you can abbreviate it as np.
Import numpy as np (abbreviation numpy as np makes retyping it easier)

-You have a function and have to pass the information to the function to know what to do. The information that you pass are called argument.

```
2 + 2
4

2 + 2 + 3
7

2-3
-1

2 + (-3)
-1

2 * 4
8

2 * 4 * 3
24

4 / 2
2.0

3 / 2
1.5

10 / 3
3.3333333333333335

2 ** 4
16

import numpy as np


32 / 6
5.333333333333333

np.round(32/ 6)
5.0

np.floor(32 / 6)
5.0

np.ceil(32 / 6)
6.0
```

Ex:
np.floor -> the floor of the largest integer
np.ceil -> round to the highest integer
np.round-> round to the nearest integer

Done in order of execution(PEMDOTS)



Functions inside of python:

```
1. type(1)
2. int
```

-The attribute is located inside the parentheses to pass the function. Type is a function that returns the class type of argument inside the parenthesis. used to indicate what kind of number it is.
type(object)

```
1. type(1.0)
2. float
```

Output is float meaning the number has a decimal

```
1. type("1")
2. str
```

Output is a string meaning it the object has a quotations(could have single or double quotations)

Two different types of arguments can be passed:

1) Single arguments: type(obj) is passed
2) Three arguments: type(name, bases, dict) it returns

Type is mostly used for debugging purposes. Debugging is used to locate the mistake of the code.

Collection:

A collection is a group, set of elements and ways to collect elements in python. Used to store collections of data like list, dict, set and a tuple.

Incorporating the type function in a list of numbers.

```
type([1,2,3,4,5])
list
```

Doesn't matter if you use single or double quotations in strings the output will always be single quotations.
Ex:

```
["Group A", "Group B", "Control 'Group'"]
['Group A', 'Group B', 'Control 'Group']
```

Naming a list:
Created something to hold an object as the object is a list .

```
list_1 = [1,2,3,4,5]
```

Lets retrieve the object -> call out the object

```
list_1
[1, 2, 3, 4, 5]
```

- Shows the ability to store something on the computer.

Extracting elements from that list:

```
len(list_1)
5
```

len(list_1) = indicates how many objects are inside the list.
-Attributes always go inside of parentheses to be able to pass the function.

Lets import an numpy library into python:

```
import numpy as np

np.sum(list_1) / len(list_1)
3.0
```

Different way to find the average of the list add up all the values then divide by the total number that the list contains.

Indexing in list:
The elements in a list are indexed and 0 is the first index. How to call out which an integers position in the list. indicates the object you want which is position 0.

```
list_1[0]
1

list_1[2]
3

list_1[-1]
```

The [-1] index indicates to index the last number of the list.

Indexing certain numbers in list:
-Up to but not including the last number. That is why 4 isn't included into the output of the list.

```
1. list_1[2:3]
2. [3]
3.
4. list_1[2:4]
5. [3, 4]
```

Appending instances to list"

```
1. list_1.append(6)
```

- Adding 6 to list_1

Return the list:

```
1. list_1
2. [1,2,3,4,5,6]
```

- the 6 has permanently been added to the list

Adding instances to the list but not permanently:

```
1. list_1 + [7]
2. [1,2,3,4,5,6,7]
```

Return the list:

```
3. list_1
4. [1,2,3,4,5,6]
```

- The 7 has not be added

Removing instances from a list:
Use the remove function to remove 5 integer.

```
1. list_1.remove(5)
2.
3. list_1
4. [1,2,3,4,6]
```

- 5 has been removed from the list

Collections: Dictionaries
A dictionary is a set of objects(key) to another object(value). In python its mapping unique keys to value. Dictionaries are mutable, meaning that they can be changed at any point.

Ex: "key": value

```
1. dict_1 = {"Pneumonia": 10, "Hypertension": 5, " Diabetes": 4}
```

```
What type of attribute is it?
type(dict_1)
dict
```

What if you want to figure out the items in the dictionary?

```
dict_1.items()
dict_items(['Pneumonia' , 10)( 'Hypertension' , 5)('Diabetes', 4 )])
```

- output is the key, value associated with the dictionary

What if you want the keys of the dictionary?

```
dict_1.keys()
(['Pneumonia', 'Hypertension', 'Diabetes])
```

- Keys of the dictionary

```
dict_1.values()
dict_values([10,5,4])
```

- values of the dictionary

```
dict_1["Hypertension"]
5
```

- What if you want the value of the key?
  Place the object inside the [ ] to return the value.

Adding(update) / Removing(pop) items from the list:

1. Adding items to a list:

```
dict_1.update({"Bronchitis":6})
```

```
dict_1
{'Pneumonia':10, ' Hypertension': 5, 'Diabetes': 4, 'Bronchitis': 6}
```

2. Removing an item from the list:
   - use .pop(key) then its going to return the value

```
dict_1.pop("Hypertension")
5

dict_1
{'Pneumonia': 10, 'Diabetes':4,  'Bronchitis': 6}
```

Making a dictionary between a dictionary:
key : value

```
dict_2 = {"Diabetes": {"IDDM":10, "NIDDM":12}, "Hypertension": 12 }

dict_2
{'Diabetes': {'IDDM': 10, 'NIDDM': 12}, 'Hypertension': 12}

dict_2.values()
dict_values([{'IDDM' : 10, 'NIDDM': 12}, 12])
```

Working with Data: Importing external data

-working with data using excel spreadsheet import it and use it
-columns = make up the different variables in the
-When doing spreadsheets don't have space in between them
numerical values = age, temperature
Categorical = grade, diabetes(Y,N), logistics, pneumonia, pain(1-10 scale)

1)Import an excel file on jupyter notebook
Importing an Excel file into python by using pandas.

```
import pandas as pd

df = pd.read_excel("name of the file")
```

-Make sure the type in the “name of the file.xlsx”

```
type(df) pandas.core.frame.DataFrame

df.head()
```

	Age	Grade	Logistics	Temperature	Hypertension	Pneumonia	DiabetesMellitus
0	77	B	Immediate Surgical Intervention(then Ward/ ICU...	37.4	No	No	No
1	64	A	Admit to Ward for Care	38.0	Yes	No	No
2	87	A	Admit to Ward for Care	38.5	No	No	No
3	70	A	Admit to Ward for Care	38.2	No	No	No
4	85	B	admit to Ward for Care	38.1	Yes	No	No

- This displays the first 5 rows

What about displaying the last 5 rows?:

```
1. df.tail()
```

	Age	Grade	Logistics	Temperature	Hypertension	Pneumonia	DiabetesMellitus
15	82	A	Admit to Ward for Care	37.3	Yes	Yes	No
16	84	B	Admit to Ward for Care	38.3	No	No	No
17	27	B	Admit to Ward for Care	37.1	No	No	No
18	22	A	Admit to Ward for Care	38.3	Yes	no	No
19	58	B	Immediate Surgical Intervention(then Ward/ ICU...	37.5	Yes	Yes	No

- Displays the last 5 rows

Column names:

```
1. df.columns
2. Index(['Age ', 'Grade ', 'Logistics ', 'Temperature', 'Hypertension',
3.        'Pneumonia ', 'DiabetesMellitus '],
4.       dtype='object')
```

- returns the column names

What about how many rows the data contains?

```
1. len(df)
2. 20
```

- There are 20 rows in the data frame

Alternative way to display the number of rows in the data:

```
• df.shape[0]
• 20
```

How many rows and columns are in the data frame?

```
1. df.shape
2. (20, 7)
```

- 20 rows and 7 columns

How many columns are in the data?

```
1. df.shape[1]
2. 7
```

Indexing a data frame to figure out the number of rows and columns?
[0] rows -> columns [1]

```
1.df.shape[0]
2.20
```

20 rows

```
3. df.shape[1]
4. 7
```

7 columns

Looking at specific columns in the data frame?

```
1. df.Temperature
2.
3. 0     37.4
4. 1     38.0
5. 2     38.5
6. 3     38.2
7. 4     38.1
8. 5     37.1
9. 6     38.0
10.7     38.1
11.8     37.9
12.9     37.8
13.10    36.6
14.11    39.1
15.12    36.4
16.13    39.3
17.14    36.1
18.15    37.3
19.16    38.3
20.17    37.1
21.18    38.3
22.19    37.5
23.Name: Temperature, dtype: float64
```

- called a panda series

You can use the dot notation because of the name of the variable doesn't contain a space between the words.

Now being more selective with the column you want to include:

```
1. df[df.Temperature >= 37.0]
```

	Age	Grade	Logistics	Temperature	Hypertension	Pneumonia	DiabetesMellitus
0	77	B	Immediate Surgical Intervention(then Ward/ ICU...	37.4	No	No	No
1	64	A	Admit to Ward for Care	38.0	Yes	No	No
2	87	A	Admit to Ward for Care	38.5	No	No	No
3	70	A	Admit to Ward for Care	38.2	No	No	No
4	85	B	admit to Ward for Care	38.1	Yes	No	No
5	30	B	Immediate Surgical Intervention(then Ward/ ICU...	37.1	Yes	No	Yes
6	60	B	Admit to Ward for Care	38.0	Yes	No	No
7	29	A	Admit to Ward for Care	38.1	Yes	No	No
8	73	A	admit to Ward for Care	37.9	No	No	No
9	68	A	Admit to Ward for Care	37.8	Yes	No	Yes
11	74	B	Admit to Ward for Care	39.1	Yes	Yes	Yes
13	88	B	Admit to Ward for Care	39.3	No	No	No
15	82	A	Admit to Ward for Care	37.3	Yes	Yes	No
16	84	B	Admit to Ward for Care	38.3	No	No	No
17	27	B	Admit to Ward for Care	37.1	No	No	No
18	22	A	Admit to Ward for Care	38.3	Yes	no	No
19	58	B	Immediate Surgical Intervention(then Ward/ ICU...	37.5	Yes	Yes	No

- **include all the rows that have a temperature higher than 37**

Selecting more than 1 column:

```
1. df[(df.hypertension == "No") & (df.Temperature >= 37)
```

	Age	Grade	Logistics	Temperature	Hypertension	Pneumonia	DiabetesMellitus
0	77	B	Immediate Surgical Intervention(then Ward/ ICU...	37.4	No	No	No
2	87	A	Admit to Ward for Care	38.5	No	No	No
3	70	A	Admit to Ward for Care	38.2	No	No	No
8	73	A	admit to Ward for Care	37.9	No	No	No
13	88	B	Admit to Ward for Care	39.3	No	No	No
16	84	B	Admit to Ward for Care	38.3	No	No	No
17	27	B	Admit to Ward for Care	37.1	No	No	No

	Age	Grade	Logistics	Temperature	Hypertension	Pneumonia	DiabetesMellitus
0	77	B	Immediate Surgical Intervention(then Ward/ ICU...	37.4	No	No	No
2	87	A	Admit to Ward for Care	38.5	No	No	No
3	70	A	Admit to Ward for Care	38.2	No	No	No
8	73	A	admit to Ward for Care	37.9	No	No	No
13	88	B	Admit to Ward for Care	39.3	No	No	No
16	84	B	Admit to Ward for Care	38.3	No	No	No
17	27	B	Admit to Ward for Care	37.1	No	No	No

- **Include all the rows with “No” hypertension and a temperature higher than 37**

```
1. df[df.Hypertension != "No"
```

	Age	Grade	Logistics	Temperature	Hypertension	Pneumonia	DiabetesMellitus
1	64	A	Admit to Ward for Care	38.0	Yes	No	No
4	85	B	admit to Ward for Care	38.1	Yes	No	No
5	30	B	Immediate Surgical Intervention(then Ward/ ICU...	37.1	Yes	No	Yes
6	60	B	Admit to Ward for Care	38.0	Yes	No	No
7	29	A	Admit to Ward for Care	38.1	Yes	No	No
9	68	A	Admit to Ward for Care	37.8	Yes	No	Yes
11	74	B	Admit to Ward for Care	39.1	Yes	Yes	Yes
15	82	A	Admit to Ward for Care	37.3	Yes	Yes	No
18	22	A	Admit to Ward for Care	38.3	Yes	no	No
19	58	B	Immediate Surgical Intervention(then Ward/ ICU...	37.5	Yes	Yes	No

- **exclude all the rows with “No” hypertension**

Creating Simulated Data:

1) import different libraries needed to perform the function

```
import pandas as pd
import numpy as np
from scipy import stats

from plotly.offline import iplot, init_notebook_mode
init_notebook_mode()

import plotly.graph_objs as go
```

What is a pseudo-random number?

It is a mathematical function that generates a sequence of nearly random numbers. It is done by taking a parameter to start off the sequence called seed. Producing a random number by using deterministic system in which no randomness model in which it will always produce the same initial condition will always produce the same output. It takes a parameter called the see and since the function is deterministic (given the same seed) it will produce the same sequence of random numbers for the same initial input.

```
np.random.seed()  ->the number you place inside the parenthesis is called a parameter, since its a deterministic system the same parameter will always give the same output.
```

Randomness can be applied by using the pseudorandom number generator.

How to generate a pseudo random number?

```
np.random.seed(123)
np.random.rand()
0.6964691855978616
```

- generating a random number from 0 to 1.
- This is called a pseudo-random generator is called

Creating more than 1 random number?

```
np.random.seed(123)
np.random.rand(10)
array([0.69646919, 0.28613933, 0.22685145, 0.55131477, 0.71946897,
       0.42310646, 0.9807642 , 0.68482974, 0.4809319 , 0.39211752])
```

- generating 10 random numbers from 0 to 1

Now what if we want rows and columns?

```
np.random.seed(123)

np.random.rand(5,10)
array([[0.69646919, 0.28613933, 0.22685145],
       [0.55131477, 0.71946897, 0.42310646],
       [0.9807642 , 0.68482974, 0.4809319 ],
       [0.39211752, 0.34317802, 0.72904971],
       [0.43857224, 0.0596779 , 0.39804426]])
```

- 5 rows and 10 columns
  np.random.seed(123)

Being more selective of the list:

```
np.random.seed(123)
np.random.randint(low = 15,
                  high = 86,
                  size = 10)
array([81, 32, 72, 62, 47, 61, 40, 51, 83, 64])

```

- you need commas to separate the list
- lowest number 15, highest 86 and size -> want 10 numbers has output

Generate 10 random numbers:

```
np.random.seed(123)

np.random.randn( 10 )
array([-1.0856306 ,  0.99734545,  0.2829785 , -1.50629471, -0.57860025,
        1.65143654, -2.42667924, -0.42891263,  1.26593626, -0.8667404 ])

```

- when its only one argument you don't have to be specific on you want, since you only want 10 random numbers you don't have to be specific in what you want the argument to be like above.

Scipy stats:
Is a continuous random variable that generates a random sample from a normal random variable with
-uses numpy and doesn't have a random number generator, random numbers are obtained by three ways

1) directly from numpy. random
2) By transformation of other random numbers that are available in numpy.random
3) using ppf(obtained by an equation solver) to transform uniform random numbers

```
np.random.seed(123)

stats.norm.rvs(loc = 15,
               Size = 1)
array([13.9143694])
```

- Loc is the location specifies the mean
- Scale is the standard deviation
- the comma to separate the list
- Generating the standard deviation by using loc number

```
np.random.seed(123)

stats.norm.rvs(2,3, size 10)
```

- Mean of 2, standard deviation of 3

  Two different ways to express a mean:
  1) typing the integral

```
np.random.seed(123)

stats.norm.rvs (2,
                Size = 10
array([ 0.9143694 ,  2.99734545,  2.2829785 ,  0.49370529,
1.42139975,
```

```
        3.65143654, -0.42667924,  1.57108737,  3.26593626,
1.1332596 ])
```

- The mean is 2 with output of 10 numbers.

2)the loc(location) of the numbers

```
4. np.random.seed(123)
5.
6. stats.norm.rvs (loc = 2,
```

```
                Size = 10
array([ 0.9143694 ,  2.99734545,  2.2829785 ,  0.49370529,
1.42139975,
        3.65143654, -0.42667924,  1.57108737,  3.26593626,
1.1332596 ])
```

- the mean is 2 with output of 10 numbers

-get the exact same array of numbers for the 2 methods

Cleaning up a data frame:

Import library to clean up data

```
1. import panadas as pd
2. import numpy as np
3.
4. from plotly.offline import plot, init_notebook_mode
5. init_notebook_mode()
6.
7. import plotly.graph_objs as go
```

import data frame :

```
1. df =  pd.read_excel("ProblemData.xlsx")
```

Just to make sure data is imported call out the first 5 rows:

```
8.   df.head()
```

	Group	Treatment	Age	Cholesterol
0	I	A	74.0	6.7
1	I	A	74.0	6.7
2	I	A	68.0	6.7
3	II	B	21.0	5
4	I	B	66.0	3.7

Group and Treatment are categorical variable
Lets detect if there's any problems:

```
1. df.shape
2. (301,4)
```

- 301 rows and 4 columns

The one thing you can do is see there's any duplicates but YOU have to understand your data as it could be that there are 2 patients that have the same output information.

```
3. df.drops_duplicates(keep = "first",
4.                     inplace = True)
```

- Its going to find duplicates, -> go row by row to find duplicates and 2 arguments being passed is
  1) keep = "first"
    - Default argument meaning that you only need to keep the first set of data (no more duplication)
  2)inplace = "True"
    - makes the changes permanent

Lets see the changes:

```
5. df.shape
6. (299, 4)
```

- you had 2 rows that had the same data -> two duplications got deleted

```
7. df.dtypes
```

```
Group           object
Treatment       object
Age            float64
Cholesterol     object
dtype: object
```

- Theres a mistake in the data frame it seems to be that cholesterol is an object but its suppose to be a int(numbers)

```
8. df.Group.value_counts()
```

```
II     100
I       99
III     99
Name: Group, dtype: int64
```

- if you add the up the columns its 298 but there's supposed to be 299 rows

```
9. len(df.Group)
299
```

Why doesn't the group match the length of the rows?
This might be due to the fact that there could be an empty cell. This happens a lot when designing your own data frame.

Count how many cells are empty?

```
df.Group.isna().sum()
1
```

- there is one empty cell

```
10. len(df.Grou
```

- minute 6:10
- 21:Working with Data; Cleaning up data

Describing categorical data:

```
df.Age.describe()
```

```
count    298.000000
mean      51.463087
std       19.735805
min      -10.000000
25%       37.000000
50%       52.000000
75%       65.000000
max      104.000000
Name: Age, dtype: float64
```

What's the problem with this data set?

- The minimum is negative which doesn't make sense
- the max is 104 which also doesn't make sense

Lets see if theres an empty cell?:

```
1)df.Age.isna().sum()
1
```

- yes there is an empty cell in the data spread sheet

```
1. df.Cholesterol.describe()
```

```
count     298
unique     86
top         5
freq       10
Name: Cholesterol, dtype: int64
```

- This doesn’t describe a numerical value , we should see means standard deviations, Have to ma

Lambda function:

```
1. a= 3
   b= 4
   a + b
   7
2. addition = lambda x, y: x +y
```

- Lambda function take two variables then what do you want to do with them :(which is the semi colon) add x + y
- Semi colon suggest what you want to do with the attribute

```
3. addition(a, b)
```
```
7
```

Isinstance function:

```
1. isinstance(3, int)
```
```
True
```

- Isinstance checks what you pass, is 3 an integer -> true

```
1. df[df.Cholesterol.apply(lambda x: is instance(x, str))]
```

- what the [ ] is that its addressing something, so its going to go down the cholesterol column and look for all the strings true or false

	Group	Treatment	Age	Cholesterol
26	III	B	42.0	X

- now we found an X in the cholesterol column meaning that the cholesterol for this person was not found at all -> the person put an X in the cholesterol value but it throws off the data set

Coerce function: is to delete everything that you find an error towards

```
1. df.Cholesterol = pd.to_numeric(df.Cholesterol, errors = "coerce")
2. df.Cholesterol.describe()
```
```
count     297.000000
mean        9.349158
std        57.709475
min         1.100000
25%         4.800000
50%         6.000000
75%         7.200000
max      1000.000000
Name: Cholesterol, dtype: float64
```

- now you can tell that the X value is gone from cholesterol and the data doesn't have weird values like before
- we clean up the string something that is not a numerical value

```
1. df.Cholesterol.isna().sum
2. 2
```

```
1. df.Cholesterol.isna()
25    False
26     True
27    False
28    False
29    False
30    False
      ...
271   False
272   False
273   False
274   False
275   False
276   False
277   False
278   False
279   False
280   False
281   False
282   False
283   False
284   False
285   False
286   False
287   False
288   False
```

- go down all the cell and the true lets you know its a missing cell

What can we do with the missing cells?

- we can replace the empty values

```
3. df.Cholesterol.fillna(df.Cholesterol.dropna().mean(),
                         Inplace = True)
```

  - inplace = true (means to make this permanent)

- the solution: is to get all the empty cells(True) and fill them out with the mean

  Checking to make sure there are no empty cells?

```
1. df.Cholesterol.isna(). sum()
0
```

  - yes there are no empty cells now 0

```
2. df.Gruop.fillna(method = "ffill"
                   inplace  = True)
```

- fillna = for all the not available cells -> is going to look at the value that is above the empty cell and copy that value in

What if you just want to delete all the values and not fill them up:

```
3. df.dropna(axis = 0
             how = "any"
             inplace = True)
```

- look at the rows across the columns and if any one is missing drop the cell
- if all the variables in that row are empty then drop them

```
4. df.shape
(297, 4)
```

- you can see how it dropped two rows now you know all the rows have data within them.

Now the Age:

```
5. df.Age.describe()
```

```
count    298.000000
mean      51.463087
std       19.735805
min      -10.000000
25%       37.000000
50%       52.000000
75%       65.000000
max      104.000000
Name: Age, dtype: float64
```

Lets look at all the patients older than 18

```
1. df[df.Age < 18]
```

	Group	Treatment	Age	Cholesterol
8	III	B	1.0	9.1
77	III	A	10.0	8.6
98	III	A	14.0	5.5
111	II	B	-6.0	4.6
119	I	A	-10.0	5.7
174	II	A	13.0	6.7
281	I	A	17.0	5.0
286	II	A	-7.0	7.6
299	I	A	2.0	4.3

```
2. df = [df.Age >= 18]
```

- now lets keep the ones with only ages that are higher than 18 in the column

```
3. df.Age.describe()
```

	Group	Treatment	Age	Cholesterol
8	III	B	1.0	9.1
77	III	A	10.0	8.6
98	III	A	14.0	5.5
111	II	B	-6.0	4.6
119	I	A	-10.0	5.7
174	II	A	13.0	6.7
281	I	A	17.0	5.0
286	II	A	-7.0	7.6
299	I	A	2.0	4.3

```
4. df = df[df.Age <= 85]
5. df.shape
   (279, 4)
```

Descriptive statistics:

-Summary of the data that tells us the story

Measuring a central tendency:

Range:

```
1. df.Age.max()
   90
```

```
2. df.Age.min()
   18
3. df.Age.max()- df.Age.min()
   72
```

- This is the range of the values

```
4. df.Age.std()
   21.75
```

- going to be the standard deviation
- its the square root of the varia

```
5. df.Age.var()
   471.43
```

```
6. df.Age.quantile([0, 0.25, 0.5, 0.75, 1])
```

	Group	Treatment	Age	Cholesterol
8	III	B	1.0	9.1
77	III	A	10.0	8.6
98	III	A	14.0	5.5
111	II	B	-6.0	4.6
119	I	A	-10.0	5.7
174	II	A	13.0	6.7
281	I	A	17.0	5.0
286	II	A	-7.0	7.6
299	I	A	2.0	4.3

- first quantile, second quantile, third quantile

Theres a numpy function to find the percentages of the quantiles:

```
1. np.percentile(df.Age.values , [0,25, 50, 75, 90, 100])
```

```
array([18., 35., 56., 75., 84., 90.])
```

```
1. df.Age.quartile([0,0.25,0.50,0.75,1])
```

```
0.00    18.0
0.25    35.0
0.50    56.0
0.75    75.0
1.00    90.0
Name: Age, dtype: float64
```

Sub-library:

```
1. from scripy import stats
```

```
2. stats.iqr(df["Age"])
   40.0
```

- The third quartile minus(-) the second quartile

Shortcut for numerical values:

```
3. df.Age.describe()
```

```
count    597.000000
mean      55.103853
std       21.712566
min       18.000000
25%       35.000000
50%       56.000000
75%       75.000000
max       90.000000
Name: Age, dtype: float64
```

- much simpler way to describe numerical values

For categorical values:

```
4. df.DiabetesMellitus.value_counts()
```

```
No     532
Yes     65
Name: DiabetesMellitus, dtype: int64
```

- use value counts for categorical values 4

Data Visualization: scatter plots

Cube Cube plot: its going to plot every quantile and it has a distribution or a normal distribution
Seeing the normality -> passing the list to a normal distribution
-statistical test to see the assumption of the test
-assumption of normality

Linear Model:
Steps towards analysis data?

1) import libraries needed:

```
1. import pandas as pd
2. import numpy as np
3. from scipy import stats
```

```
1. import plotly.graph_objects as go
2. import plotly.express as px
3. import plotly.io as pio
4. pio.templates.default = "plotly_white"
5. import statsmodels.api as sm
```

- import template of plotly as a white figure

- statsmodel contains a lot of statistical functions (used for linear models)

Data Import: Using a csv file

```
1. df = pd.read_excel('Donations.csv)
2. df.head()
```

- using a spreadsheet file as a csv file

	NO2	PM2_5	PM10	LDL
count	290.000000	290.000000	290.000000	290.000000
mean	38.698276	72.800345	131.490690	3.863793
std	3.179285	2.326551	5.665682	0.460667
min	30.400000	66.400000	112.400000	2.400000
25%	36.450000	71.300000	127.700000	3.600000
50%	38.450000	72.800000	131.400000	3.800000
75%	40.900000	74.375000	135.175000	4.200000
max	47.700000	80.300000	145.600000	5.000000

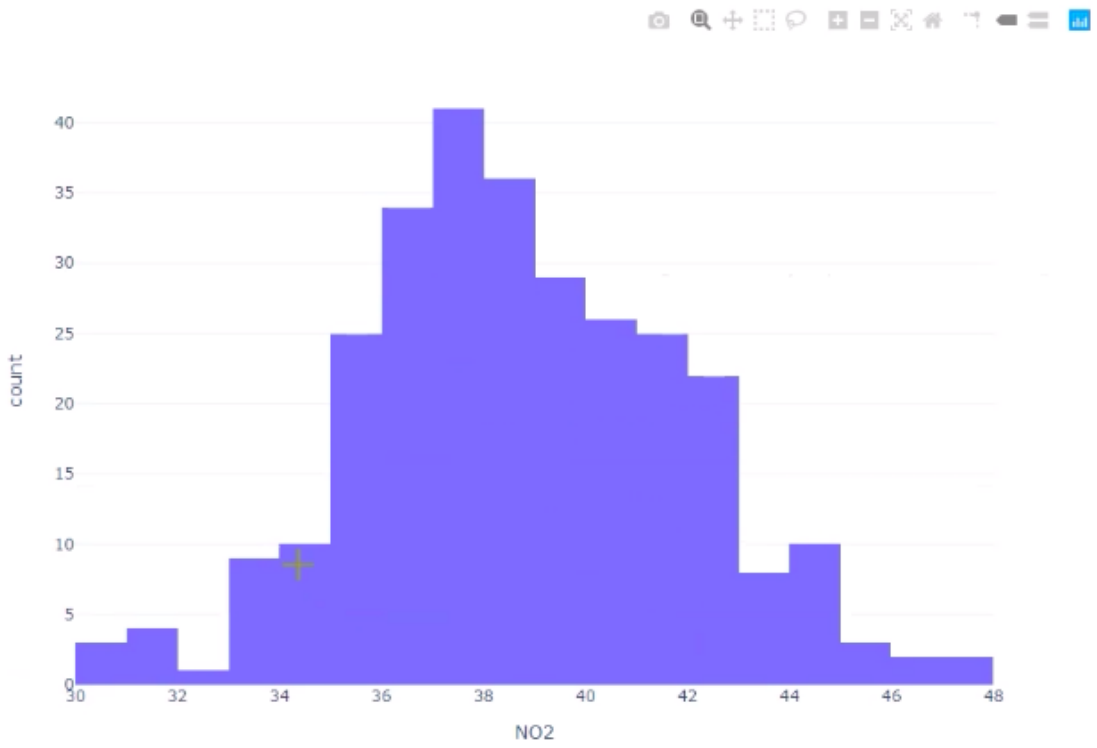
2) Descriptive statistics

```
1. df.describe()
```

	NO2	PM2_5	PM10	LDL
count	290.000000	290.000000	290.000000	290.000000
mean	38.698276	72.800345	131.490690	3.863793
std	3.179285	2.326551	5.665682	0.460667
min	30.400000	66.400000	112.400000	2.400000
25%	36.450000	71.300000	127.700000	3.600000
50%	38.450000	72.800000	131.400000	3.800000
75%	40.900000	74.375000	135.175000	4.200000
max	47.700000	80.300000	145.600000	5.000000

3) Data Visualization: one way to visualize data is through creating separate histograms

```
1. fig_no2 = px.histogram(df.'NO2')
2. fig_no2.show();
```



Correlations

```
1. stats.pearsonr(df.NO2, df.LDL)
(0.739538000442933, 1.95590711199)
```

-To test the correlation between two variables without plotting them
-you can do this with any two variables
-strong correlation between NO2 and LDL

Linear Model: can we predict someone's cholesterol, need the independent and the dependent variables to be separated in computer variables

Need two variables,
1) one to hold the data frame(the independent variable)
independent variable = stands alone, isn't changed by other variables and is the thing that is being measured

```
1. X = df.drop('LDL', axis = 'columns').to_numpy ()
```

Drop the LDL column and save it has a numpy array -> so its going to delete the data frame which includes the column and variables

```
1. y= df.'LDL.to_numpy()
```

- dependent variable so you have the numpy array there as well
- to do linear regression you need a column where all the values are 1, which represents the y-intercept

Benefits of using csv(Comma Separated Values) files?

- can be opened or edited by any text editors
- any programmer can use a csv file
- doesn't manipulate data
- can be opened in any text editor like microsoft words, excel, Microsoft
- known as flat files(text file) -> just stores data but does not contain the formatting of formulas

Benefits of using Excel(xlsx):

- stores data and operates on data
- any programming language library to parse Excel data
- Does consume more memory while importing data

Data Import:

```
1. X = df.drop("LDL", axis = 'columns'). to_numpy()
2. y = df.LDL.to_numby()
```

- your going to drop the "LDL" column, have to tell pandas what it is, since its being eliminated completely you use the sublibrary of numpy

```
1. X = sm.add_constant(X)
2.
3. X[0:5]
```

- to do linear regression it requires a column where all of the values are 1. Which represents the y intercept
-

```
1. X[0:5]
```

```
array([[  1. ,  37.7,  71.6, 129.7],
       [  1. ,  37.1,  74.3, 128.8],
       [  1. ,  37.3,  78.1, 133.4],
       [  1. ,  33.5,  74.1, 128.4],
       [  1. ,  33. ,  70. , 123.7]])
```

- To view the first five subjects

- need to the 1 to calculate the y-intercept
- This time I used indexing instead of .head() function

Now to create a model using the ordinary least squares method in statsmodels

```
1. model = sm.OLS(y,X)
```

- given this model we can fit the data now
- give the result = model.fit (command) ->then call it out

```
1. result = model.fit()
2. result.summary()
```

OLS Regression Results

Dep. Variable:	y	R-squared:	0.548
Model:	OLS	Adj. R-squared:	0.543
Method:	Least Squares	F-statistic:	115.5
Date:	Thu, 28 Nov 2019	Prob (F-statistic):	5.22e-49
Time:	13:26:52	Log-Likelihood:	-71.150
No. Observations:	290	AIC:	150.3
Df Residuals:	286	BIC:	165.0
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.0978	0.734	0.133	0.894	-1.347	1.542
x1	0.1075	0.006	18.491	0.000	0.096	0.119
x2	-0.0012	0.008	-0.148	0.883	-0.017	0.014
x3	-0.0024	0.003	-0.725	0.469	-0.009	0.004

Omnibus:	0.565	Durbin-Watson:	1.896
Prob(Omnibus):	0.754	Jarque-Bera (JB):	0.668
Skew:	-0.096	Prob(JB):	0.716
Kurtosis:	2.864	Cond. No.	6.23e+03

- What you can do? is multiple the coefficient values X1, X2, X3
- in the sense of (0.0978 + 0.1065 * 37.7 +(-0.0012) * 71.5 +(-0.0024) *129.7
- 3.7534 but our true model LDL level of 4.1 / the prediction were a value of 3.8
- What the coefficient are? are the slope on the plane, all of them have slopes and highly correlates with the p value
-
- x1 = NO2 column
- x2 =
- x3 =PM10 column

- p value (not very predictive)

```
df.iloc[0]
```

```
NO2       37.7
PM2_5     71.5
PM10     129.7
LDL        4.1
Name: 0, dtype: float64
```

- we can use the coefficients to calculate and predict the LDL levels.

```
list(zip(np.round(result.predict(X)[0 : 5], 1), y[0:5]))
```

- X only contained the independent values, and its predicted value is inside of a list and

```
[(3.8, 4.1), (3.7, 4.0), (3.7, 3.7), (3.3, 3.6), (3.3, 3.3)]
```

- our predicted values for the coefficients that will give us the lowest possible error

Comparing Means: (comparing two means a T test)
a variety of T test in python

Logistic Regression

Libraries:
as usual we start by importing the libraries that we use in the notebook

```
import pandas as pd
import numpy as np
from scipy import stats
```

```
import plotly.graph_objects as go
import plotly.express as px
import plotly.io as pio
pio.templates.default = 'plotly_white'

import  statsmodels.api as sm
```

- Usage of plotly

Data import:

```
df = pf.read_csv('Logistics.csv')
df.head()
```

	Group	Age	CRP	WCC	Target
0	Two	61	0.2	14.0	B
1	Two	43	0.3	13.6	A
2	Two	37	0.2	11.5	A
3	One	36	2.1	4.5	A
4	Four	44	2.7	7.3	A

- Shows the first 5 rows

Lets look at how big our data set is:

```
df.shape
```

(500, 5)

- 500 rows and 5 columns

```
df.dtypes
```

```
Group      object
Age         int64
CRP       float64
WCC       float64
Target     object
dtype: object
```

- the dependent variable values must be numeric so we need to convert the A and B values. We choose B as our class of choice. We are going to encode B as 1 and A as 0.

```
df.Target.replace({'A':0 'B':1},
                    Inplace = True)
```

- lets look at the data frame object again and make sure that the target variable now holds integers
- Use a dictionary to change the outcomes, inplace = True because we want to over right the argument

```
1. df.dtypes
2.
```

	Group	Age	CRP	WCC	Target
0	Two	61	0.2	14.0	1
1	Two	43	0.3	13.6	0
2	Two	37	0.2	11.5	0
3	One	36	2.1	4.5	0
4	Four	44	2.7	7.3	0

- to see if the dictionary replaced the group

```
3. Df.dtypes
```

```
Group      object
Age         int64
CRP       float64
WCC       float64
Target      int64
dtype: object
```

- replaced target to integers

Now lets do the same for the group variable: want to replace the written numbers in words with 1,2,3,4(just numbers)

```
1. df.Group.replace({'One':1 , 'Two':2, 'Three':3, 'Four':4}
                    Inplace = True)
   df.head()
```

	Group	Age	CRP	WCC	Target
0	2	61	0.2	14.0	1
1	2	43	0.3	13.6	0
2	2	37	0.2	11.5	0
3	1	36	2.1	4.5	0
4	4	44	2.7	7.3	0

Descriptive statistics:

- always start by summarizing statistics
- The first variable is **Group** and even though it is encoded in integer values it represents four classes for a categorical variable
- Count how many of each of the classes are present in the dataset

```
1. df.Group.value_counts()
```

```
1    138
4    123
3    120
2    119
Name: Group, dtype: int64
```

```
1. df.Group.value_counts(normalize = True)
```

```
1    0.276
4    0.246
3    0.240
2    0.238
Name: Group, dtype: float64
```

- An equal distribution of this variable, next we look at the age variable

```
2. df.Age.describe()
```

```
count    500.000000
mean      44.524000
std       10.377901
min       15.000000
25%       37.000000
50%       44.000000
75%       52.000000
max       71.000000
Name: Age, dtype: float64
```

- patients are between 15 and 71 years old -> with the various point estimates and measures of dispersion as shown by the .describe() function.
- Let's Look at CRP (c-reactive protein) and WCC

```
1. df.CRP.describe()
```

```
count    500.000000
mean      44.524000
std       10.377901
min       15.000000
25%       37.000000
50%       44.000000
75%       52.000000
max       71.000000
Name: Age, dtype: float64
```

```
1. df.groupby('Target').Group.value_counts()
2. pd.crosstab(df.Target,
3.              df.Group)
```

Target	Group 1	2	3	4
0	66	66	55	56
1	72	53	65	67

```
1. df.groupby('Target').Age.describe()
```

Target	count	mean	std	min	25%	50%	75%	max
0	243.0	38.288066	8.401553	15.0	33.0	38.0	43.0	68.0
1	257.0	50.420233	8.449439	27.0	45.0	51.0	56.0	71.0

```
2. df.groupby('Target')['CRP'].describe()
```

Target	count	mean	std	min	25%	50%	75%	max
0	243.0	1.660494	1.592078	0.0	0.6	1.1	2.4	9.1
1	257.0	1.999611	1.855261	0.0	0.6	1.5	2.9	9.6

The frequency of the dependent variable, need to know many of the classes are present

-it is important to have a class balance. An imbalance makes it difficult for a model to be accurate than more that 95% of cases in one class. A very simple model can predict that a class at all times and be 95% accurate.
Important situations: look for NaN values(missing data), logistic regression models require the data to be present

```
1. df.isna().sum()
```

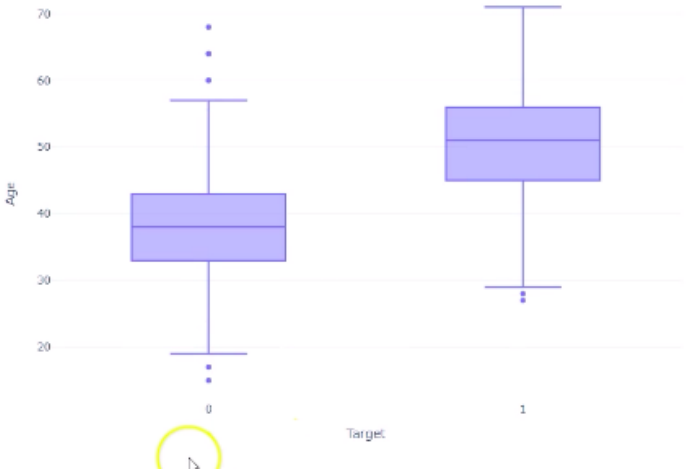
- check for missing data

Data visualization: this is part of our initial analysis, by creating a bar chart of the frequencies of the Group variable for each of the target classes. We can then create a list objects to hold the frequency counts which will indicate the height of each bar

```
2. Target_0_group_counts = df[df.Target == 0].Group.value.counts().to_list
3. Target_1_group_counts = df[df.Target == 1].Group.value.counts().to_list
```

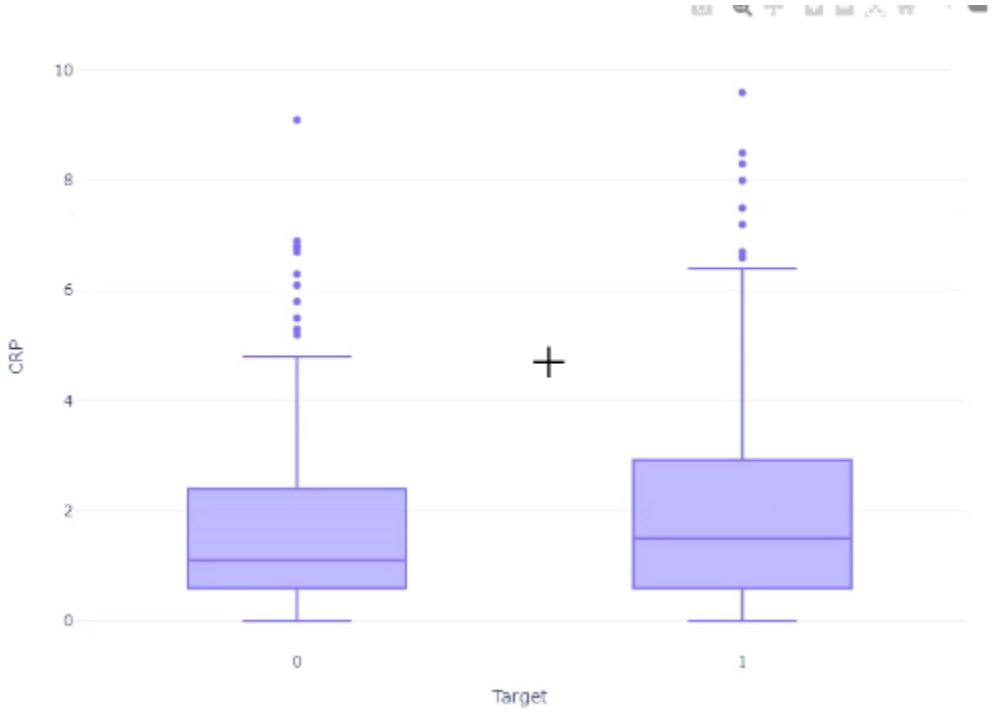
```
4. Fig_group = go.Figure()
   fig_group.add_trace(go.Bar(
   X = ['One','Two', 'Three', 'Four'
   y = target_0_group_counts,
   Name = Target 0'))

   fig_group.add_trace(go.Bar(
   X = ['One', 'Two', 'Three', 'Four'],
   y= target_1_group_counts,
   Name= 'Target 1'))

   fig_group.update_layout(
   Barmode = 'group',
   Title = 'Number of subjects in each group of the target'
   xaxis= dict(title = 'Groups'),
   yaxis=dict(title='Frequency
```


```
fig_age = px.box(df,
                 Y = 'Age'
                 X = 'Target')
 fig_age.show() ;
```



- there seems to be a difference in ages. This might make it a good predictor in a logistic regression model. We can do the same for the CRP and WCC variables

```
fig_crp = px.box(df,
                 Y = 'CRP',
                 x= 'Target)
fig_crp.show()
```



- Theres a lot of outliners, data is not evenly distributed

Odds Ratios:

- in a logistic regression we choose one of the classes in the dependent variable as the class of interest. We use the independent variable and see how its contributing to the subject ending up in the class. A logistics regression model with calculate an odds ratio for the independent variable
- less than 1 means that the independent variable lowers the odd of being specific independent variable lowers the odd of being selected class of the dependent variable
- the value of 1 increases the chances of being selected to the independent variable

Two different groups:
1) categorical independent variable: one of the sample space is chosen at the baseline and
-for odd ratios the sample space elements is compared to that base sample space element
-chances that you will get a 1 “win”outcome for t

2)for numerical values: the odd ratio refers to an increase in 1 unit value of that variable

independent variable = believed to influence the outcome(dependent variable)
dependent variable = is the outcome variable (the predicted value)

Simple regression model vs multiple regression model?
Simple regression model: is there is one independent variable

multiple regression model: if theres 2 or more independent variables
Managing categorical variables:
-the group variable has a sample space of 4 elements, 1,2,3 ,4. We will choose 1 as our base-case and compare being
-you to choose a base then you compare the variables with that base -> lets choose 1 as the base then compare it with the other variables like 2,3,4 (categorical variables are converted to integers
-we have to create a dummy variable and we have to drop the base variable liked

```
drop_first = True
```

Categorical variables:
more than a binary independent variable, choose one of them as a base and does that increase of odds in the target variable in group 1.

```
dummies_group = pd.get_dummies(df.Group,
                               Prefix = 'Group',
                               Drop_first = True
Df = pd.concat([df, dummies_group],
               Axis = 'columns'
df.head()
```

	Group	Age	CRP	WCC	Target	Group_2	Group_3	Group_4
0	2	61	0.2	14.0	1	1	0	0
1	2	43	0.3	13.6	0	1	0	0
2	2	37	0.2	11.5	0	1	0	0
3	1	36	2.1	4.5	0	0	0	0
4	4	44	2.7	7.3	0	0	0	1

- the zero means that they were in group 1
- Lets look at the first 5 columns

```
df.drop('Group', axis = 'columns', inplace = True)
```

- We first look at the first subject -> note that they are in group 2
- Dummy variables will have to be under group_two and now to drop the variable
- referred to as one hot encoding: refers to splitting the column which contains numerical categorical data to many columns depending on the number of categories present in that column. Each column contains “0’ or “1” corresponding to which is has been placed
- binary vector =

-in order to put the words in machine learning algorithm the data should be converted into vector representations
the process of converting text to number is called vectorization
One-Hot encoding:

-one hot encoding is representative of categorical variables as binary vectors
ex: rome = 1
As paris = 0

- It is important to set the base-class to 1 when categorical variables are converted into integers. Since we are making a comparison choosing 1 for the group variable we remove it from the data frame.

- Pd.get_dummies takes the columns with data type object -> then encodes them and returns a new data frame that replaces the old one
-use the new function of pd.concat(the data frame)

What are the assumptions to be made?

- the dependent variable is binary(means that they can only have 2 possible types “0” or “1”) which can represent “win” or loss”
- if there is more than categories the type of outcome a multinomial logistic regression should be used
  now independent variables:
- lack of outliers
- absence of multicollinearity: absence of perfect multicollinearity -> linear relation among the predictors
- cannot be a repeated measures design,collecting outcomes at two different time points

Ex: of data used
Question: how does the GRE score,GPA and prestige of the undergraduate institution effect admission into graduate school?
Independent variable: GRE scores, GPA, and undergraduate prestige
Dependent variables: is admission status(binary)

Managing categorical variables:

Creating the logistic regression model:
-to create a logistic regression model we need to specify the independent variables ,and the dependent variables
-we create two computer variables X and y to hold the former and the latter

```
8.  X = df.drop('Target'
9.                axis  = 'columns')
10. Y = df.Target
```

- need an intercept -> to do this we need to add a column of constants(being all l’s)

```
11. x = sm.add_constant(X);
```

- adding the intercept so we add a column of 1.

```
X.head()
```

	const	Age	CRP	WCC	Group_2	Group_3	Group_4
0	1.0	61	0.2	14.0	1	0	0
1	1.0	43	0.3	13.6	1	0	0
2	1.0	37	0.2	11.5	1	0	0
3	1.0	36	2.1	4.5	0	0	0
4	1.0	44	2.7	7.3	0	0	1

- Let's look at the first 5 columns and is the format that you are going to finally input in you model

```
12. y.head()
```

```
0    1
1    0
2    0
3    0
4    0
Name: Target, dtype: int64
```

- let's look at the dependent variable(outcome predictions)

$y_i = \alpha + x_i\beta + \varepsilon_i$

-expect the dependent variables to take a nonzero value

Creating the model:

the logit() function takes as arguments our dependent variable and our independent variable

-storing it into the computer variable called model

-logistic regression 5 from the beginning

Creating the logistic regression model: In order to create a logistic regression model -> you need to specify the independent variables, and the dependent variable -> two computer variables, X & Y to hold the former and the latter

```
X = df.drop('Target',
                        axis = 'columns')
Y = df.Target
```

- For proper calculations, we need an intercept. To do this we add a column of constants (the constant is one column all with 1's)
- we have to add all the constants

```
X = sm.add_constant(X);
```

```
C:\Users\juank\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:2389: FutureWarnin
g:

Method .ptp is deprecated and will be removed in a future version. Use numpy.ptp instea
d.
```

```
X.head()
```

	const	Age	CRP	WCC	Group_2	Group_3	Group_4
0	1.0	61	8.2	14.0	1	0	0
1	1.0	43	8.3	13.6	1	0	0
2	1.0	37	8.2	11.5	1	0	0
3	1.0	36	2.1	4.5	0	0	0
4	1.0	44	2.7	7.3	0	0	1

```
result.summary()
```

Dep. Variable:	Target	No. Observations:	500
Model:	Logit	Df Residuals:	493
Method:	MLE	Df Model:	6
Date:	Thu, 28 Nov 2019	Pseudo R-squ.:	0.3955
Time:	13:59:11	Log-Likelihood:	-209.37
converged:	True	LL-Null:	-346.38
Covariance Type:	nonrobust	LLR p-value:	3.011e-56

	coef	std err	z	P>|z|	[0.025	0.975]
const	-11.6056	1.086	-10.686	0.000	-13.734	-9.477
Age	0.1979	0.018	10.833	0.000	0.162	0.234
CRP	0.2017	0.069	2.912	0.004	0.066	0.337
WCC	0.1916	0.029	6.677	0.000	0.135	0.248
Group_2	0.1113	0.346	0.322	0.747	-0.566	0.789
Group_3	0.4817	0.347	1.389	0.165	-0.198	1.161
Group_4	0.4222	0.341	1.238	0.216	-0.246	1.091

- being in group_2, group_3, and group_4 is insignificant has being compared to 1
- This summary contains a lot of information as the pseudo -R^2 value of 0.3955 is a derivative the slope of the R^2 value in linear regression.

The p values and 95% confidence intervals are the most interest. The coefficients(p values) are our odds ratios. We don't need to take the exponents of these values.

```
1. np.exp(result.params.values)
```

```
array([9.11527875e-06, 1.21886504e+00, 1.22346976e+00, 1.21117578e+00,
       1.11775905e+00, 1.61890104e+00, 1.52524397e+00])
```

- The odds ratios of each of the independent variable, in the order of their listing in the summary. For every 1 unit increase in age, we note an increase in the odds of being in class 1 of 22%. We got this by subtracting 1 from the OR(1.21886505 - 1) and expressing the value as a percentage .

```
1. np.exp(result.conf_int().values)
```

```
array([[1.08480715e-06, 7.65926979e-05],
       [1.17599220e+00, 1.26330088e+00],
       [1.06818061e+00, 1.40133441e+00],
       [1.14493481e+00, 1.28124918e+00],
       [5.67680216e-01, 2.20886109e+00],
       [8.20518246e-01, 3.19412832e+00],
       [7.81584822e-01, 2.97647689e+00]])
```

- Age variable we note that there is a 95% confidence interval is above 1 This gives us a significant p value for within the confidence limits we always find an increase in the odds.

In group_2 variable the confidence interval is above 1, as it is 1.118 shows a 12% increase. The odds for being in 1, the p value is not significant because the confidence intervals shows that both values decrease and increase the odds. If the odds ratio is less than 1 then we subtract it.

confidence intervals: its is a estimates that measures the parameter value, you except 90% of the confidence intervals that will include the unknown true values of the parameters -> in most cases the confidence levels is taken as 95%. How did they get this value? -> by the error model and also depends on the parameters.

Categorical Variables:

Since its a string each element goes inside of a square bracket [ ]. You want 10 random choices and replace the default. 10 random choices which is size =10 from the two. Replace = true which means that you pick you then replace it then pick another choice either treatment or control group. Set the probability of each 0.5 and 0.5 is 50%, 50% and has to add up to 1. You can skew the probability of the choices.

Hiding code:

```
from IPython.display import HTML

HTML('''<script>
code_show=true;
function code_toggle() {
 if (code_show){
 $('div.input').hide();
 } else {
 $('div.input').show();
 }
 code_show = !code_show
}
$( document ).ready(code_toggle);
</script>
<form action="javascript:code_toggle()"><input type="submit" value="Click here to toggle on/off the raw code."></form>''')
```