Project: Vitória No-show Medical Appointment

Table of Contents

Student Tags

Author: Anderson Hitoshi Uyekita
Course: Data Science - Foundation I
COD: ND110
Date: 13/12/2018
Dataset: No-show appointments
Version: 1.0

Introduction

I decided to face this specific problem because I am from Brazil and it will be a big challenge for me to make any conclusions using this data. I am an Electrical Engineer, and I have never been to Vitória, even more, I do not anything about the public health care systems.

It's a bit complicated to address any kind of question (beforehand) if I do not have a clue about the variables in this dataset. Obviously, (Now) I have already loaded the data and printed some summaries, and my approach for this Project is aggregate new variables (all these new information are public and free to access) to enhance my analysis.

With these new variables, such as Average Income per Month, Number of inhabitants in each Neighborhood (divided into male and female), Neighborhood, and Regional Administration.

I hope at the end this document I have made good questions, and the most important, I have "answered" properly.

Objectivies

Although I do not have a question yet, I wonder to this project an opportunity to practice all the content learned "in class". So, my objectives for this project are:

- Having practised Data Wrangling
  - Gathering;
  - Assessing;
  - Cleaning;
- Having practised EDA;
- Having practised in Python Coding;
  - Also in Jupyter Notebook;
- Making a reproducible research;
- Having fun.

Synopsis

I have made along this document three questions:

1. Are patients with many appointments are more likely to show-up then the patients with less than two appointments?

> Seems to be true, but I lack information to affirm it, what I could say is when I analyse patient with a higher number of appointments, they tend to have a better rate of show-up.

2. Are patients with many appointments in average older than those with few appointments?

> I also identify a positive correlation, when I analyse patients with higher number of appointments they tend to have an age average higher than the population.

3. Are patients with disabilities tend to have better rates of show-up? Is the number of disabilities raises the show-up rates?

> Due to the small number of patients with disabilities, it is difficult to answer this question, but I have found a better rate of show-up in the group of patients with disabilities, however, patients with many disabilities tend to have worse show-up rates.

All these questions I have answered using the original data and additional data collected on the web. My report is linear and I start with a Data Wrangling (Gather and Assess), after that I start with the EDA, in this step I stopped a while to gather new data on the internet, and I came back to finish the EDA and to pose and answer the questions.

In the final of the document, I have inserted an appendix to segregate from the main document the Additional Data I have found.

Obs.: I really do not know if I am doing great and for this reason, I have decided to submit my report to receive a feedback. I am not sure If I am on the "right path". I know I have not used all the resources I have gathered in the web, and I do want to use them to pose a new question, but first I need a preliminary feedback.

Reproducibility

I have written this report using the Jupyter Notebook to allow anyone the reproducibility of each step. Despite I have created some dataset to enhance my analysis, which turns everything much harder to reproduce, I have stored it in Github, so it's available to anyone to download and to use it.

Work envinronment

All this research is performed using:

- Dell Notebook Inspiron 7348;
- Intel(R) Core(TM) i7-5500U CPU @ 2.40GHz 2.40GHz;
- 8.00 GB;
- Windows 10 Pro 64-bits.

Softwares

To be honest, I did not use the Python IDLE in this project, because I wrote directly in Jupyter Notebook, so to perform study I code everything directly in Opera.

- Opera
- Atom

I have used the Atom to push to Github repository, and nothing more than this.

Packages

I kindly ask you to install each of this packages before you run the next steps.

- Jupyter Notebook
- Pandas
- Numpy
- Matplotlib

Repository

You can access all files of this report in this repository:

- https://github.com/AndersonUyekita/udacity_data_science_foundation_01 (https://github.com/AndersonUyekita/udacity_data_science_foundation_01)

Data Wrangling

This file is available at Kaggle and maintained by @JoniHoppen (https://www.kaggle.com/joniarroba), is also available to download at GDrive.

- Kaggle (uploaded by JoniHoppen) (https://www.kaggle.com/joniarroba/noshowappointments), and;
- GDrive (https://d17h27t6h515a5.cloudfront.net/topher/2017/October/59dd2e9a_noshowappointments-kagglev2-may-2016/noshowappointments-kagglev2-may-2016.csv)

Brief about Vitória City

This dataset records medical appointments from March to june of 2016, realized in publics hospitals in Vitória (https://en.wikipedia.org/wiki/Vitória,_Espírito_Santo), a medium size city (almost 320.000 inhabitants) of Brazil located in the southeast region.

Figure 1a - Brazil Map with Espírito Santo State Highlighted.

Figure 1b - Vitória City, Capital of Espírito Santo State.

(**) Both pictures extracted from Wikipedia.

Vitória is the Capital of Espírito Santo State, and has almost 320.000 inhabitants (also according with the Wikipedia) spread in 9 Regional Administration areas, which has a total of 80 neighborhoods.

The public health systems are compounded by Basic Units of Health (*Unidade Básica de Saúde*, for the sake of this document UBS) spread all over the city, the Figure 2 shows how these UBS are located in the city.

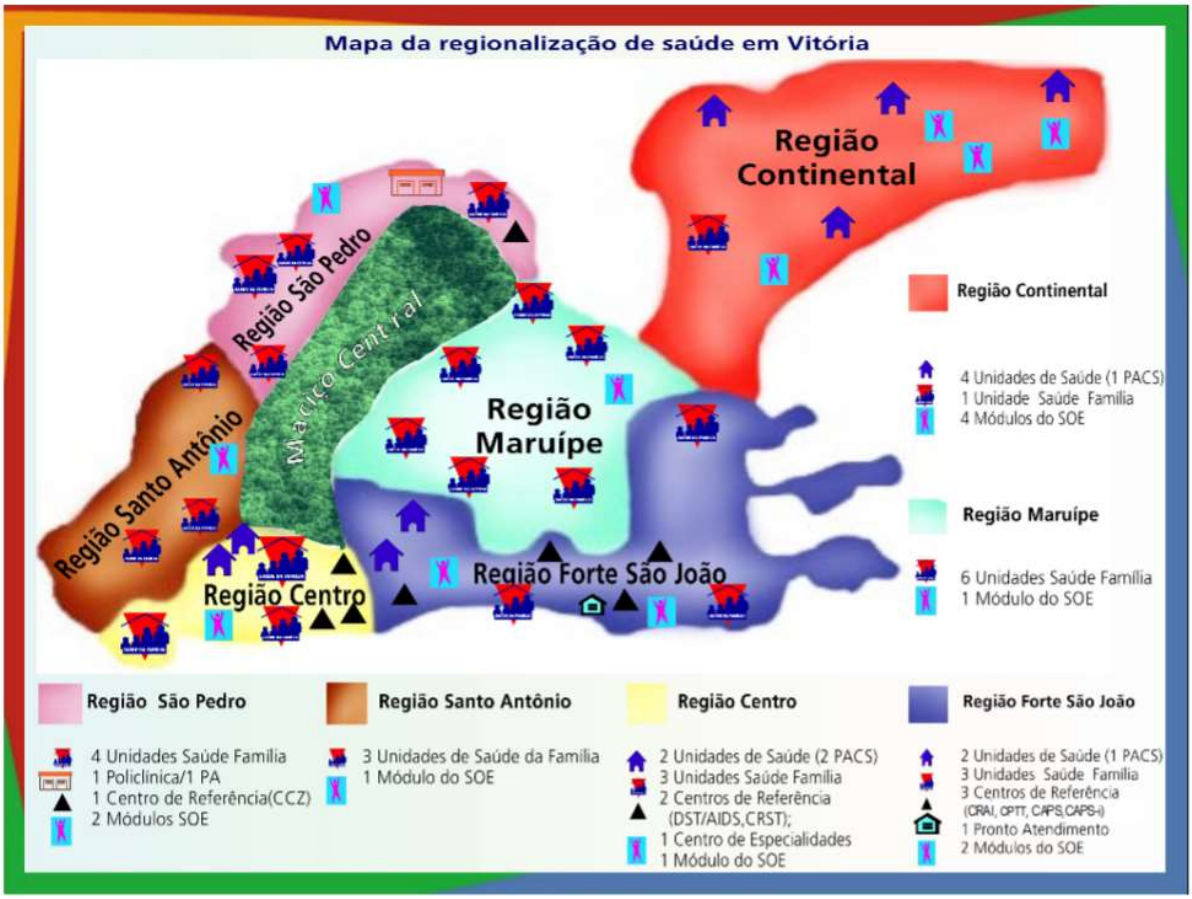


Figure 2 - UBS in Vitória City.

I have extracted this illustration from the document *Relatório de Gestão 2008* (http://www.vitoria.es.gov.br/arquivos/20100519_saude_relator_gestao_2008.pdf) from the *Sistema Único de Saúde* of Vitória Council Health Secretary, based on this document and on these two:

- Equipamento de Saúde (http://sistemas7.vitoria.es.gov.br/GeoWebApi/Downloads/pdf/saude/Equipamentos_de_Saude.pdf): This document shows each UBS address;
- Município de Vitória - Territórios de Saúde (http://sistemas7.vitoria.es.gov.br/GeoWebApi/Downloads/pdf/saude/Regioes_Territoriais_de_Saude.pdf): Summarize the number of each kind of UBS.

After a briefly compilation of info from these three sources, I have summed up 38 (See details here) UBS (or similars) in Vitória City.

- Basic Units of Health with Family Strategy(US Família): 19 units;
- Community Agent Health Program (US PACS): 8 units;
- Polyclinic, Specialized or Reference Center: 11.

I will assume for this report all appointments could be done in any of this 38 premises listed above.

Observation: This is not important to the project, but could be important to someone who wants to learn more about this dataset. The Regional Administration Area is slightly different from the Health Territory Area because the Regional Administration Area has 9 areas and the Health Territory Area has 6, which means some neighbourhood were relocated to other regions. It could be much explicit if you compare these two files (Regional Administrative Area (http://sistemas7.vitoria.es.gov.br/GeoWebApi/Downloads/pdf/politicos/Regionais_Administrativas.pdf) and Health Territory Area (http://sistemas7.vitoria.es.gov.br/GeoWebApi/Downloads/pdf/saude/Regioes_Territoriais_de_Saude.pdf)).

If you are still curious about the location of each UBS, you can access the GeoWeb (http://geoweb.vitoria.es.gov.br/), this is a web-based tool with a thematic map.

Data Structure

Unfortunately, this dataset lacks a codebook, despite this hurdle, much of the problems could be solved reading the Kaggle Discussions Channel. This dataset has:

- 110.527 rows (observations).
- 14 columns (variable or features);

Each of these variables could be interpreted by using the Table 1.

Table 1 - Variables Types Description

Variable	Type	Description
PatientId	int	Internal ID to track the patient
AppointmentID	date	Internal ID to track the appointment
Gender	cat	F: Female and M: Male
ScheduledDay	date	Planned date to appointment
AppointmentDay	date	When the appointment occurs
Age	int	Person Age, if Age is negative the person is pregnant
Neighbourhood	str	In Brazil, each city is divided into Regional Administration (*Regiões Administrativas*) and then in neighborhood (*bairros*). Read more in Appendix A.
Scholarship	bool	0: No and 1: Yes
Hipertension	bool	0: No and 1: Yes
Diabetes	bool	0: No and 1: Yes
Alcoholism	bool	0: No and 1: Yes
Handcap	int	Means the quantity of disabilities of this person (0,1,2,3,4).
SMS_received	int	Number of SMS sent to the patient
No-show	cat	No: Show-up and Yes: No-show

Additional Data

I have found some good information available on the internet, and I decided to aggregate for my analysis. In the final of this document, in Appendix chapter, I have added some details from where I get this info. As results of it, I have written two datasets:

- Vitória Regional Administration Data;
- Vitória Neighborhood Data.
- Vitória Age Data.
- Vitória UBS Dataset.

Data Assess

Let's start analysing the number of rows, columns, number of null values, number of duplicated rows, and data types. I am also interested in the first and last rows to assess if there is any problem with a header or a non-standard value in the final of the archive.

Requirements

These are all the libraries I will use during this report these libraries:

In [1]:

```
# Importing Libraries.
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
% matplotlib inline
```

Let's load the `noshowappointments-kagglev2-may-2016.csv` file, according with the instructions:

> *This dataset collects information from **100k medical appointments** in Brazil and is focused on the question of whether or not patients show up for their appointment. A number of characteristics about the patient are included in each row.*

Has 100.000 medical appointments.

In [2]:

```
# Loading the dataset.
df_vit = pd.read_csv('noshowappointments-kagglev2-may-2016.csv')

# Plot the dimensions
print("Rows: {}\nColumns: {}".format(*df_vit.shape))
```

```
Rows: 110527
Columns: 14
```

As I expected, the number of rows is roughly 110k and 14 columns.

Now, I want to see the first and the last 10 rows, just to ensure if everything is ok.

In [3]:

```
df_vit.head(5)  # Print the first 5 rows.
```

Out[3]:

	PatientId	AppointmentID	Gender	ScheduledDay	AppointmentDay	Age	Neighbourho
0	2.987250e+13	5642903	F	2016-04-29T18:38:08Z	2016-04-29T00:00:00Z	62	JARDIM I PENI
1	5.589978e+14	5642503	M	2016-04-29T16:08:27Z	2016-04-29T00:00:00Z	56	JARDIM I PENI
2	4.262962e+12	5642549	F	2016-04-29T16:19:04Z	2016-04-29T00:00:00Z	62	MATA I PRA
3	8.679512e+11	5642828	F	2016-04-29T17:29:31Z	2016-04-29T00:00:00Z	8	PONTAL I CAMBU
4	8.841186e+12	5642494	F	2016-04-29T16:07:23Z	2016-04-29T00:00:00Z	56	JARDIM I PENI

In [4]:

```
df_vit.tail(5)  # Print the last 5 rows.
```

Out[4]:

	PatientId	AppointmentID	Gender	ScheduledDay	AppointmentDay	Age	Nei
110522	2.572134e+12	5651768	F	2016-05-03T09:15:35Z	2016-06-07T00:00:00Z	56	M
110523	3.596266e+12	5650093	F	2016-05-03T07:27:33Z	2016-06-07T00:00:00Z	51	M
110524	1.557663e+13	5630692	F	2016-04-27T16:03:52Z	2016-06-07T00:00:00Z	21	M
110525	9.213493e+13	5630323	F	2016-04-27T15:09:23Z	2016-06-07T00:00:00Z	38	M
110526	3.775115e+14	5629448	F	2016-04-27T13:30:56Z	2016-06-07T00:00:00Z	54	M

After check the first and last rows, and confirm nothing is wrong with it. I will analyse the number of duplicated entries.

In [5]:

```
# Sum the number of rows with duplicated entries.
dupe = sum(df_vit.duplicated())

print("Number of duplicated rows: {}".format(dupe))
```

```
Number of duplicated rows: 0
```

Nice dataframe, no duplicated entries (identical row). Now, let's find any NA value or any kind of strange value using the `.info()` method. Note with results of `.info()` I can check if there is inconsistency in the data type of each variable.

In [6]:

```
# Check if any rows has NA values.
df_vit.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 110527 entries, 0 to 110526
Data columns (total 14 columns):
PatientId         110527 non-null float64
AppointmentID     110527 non-null int64
Gender            110527 non-null object
ScheduledDay      110527 non-null object
AppointmentDay    110527 non-null object
Age               110527 non-null int64
Neighbourhood     110527 non-null object
Scholarship       110527 non-null int64
Hipertension      110527 non-null int64
Diabetes          110527 non-null int64
Alcoholism        110527 non-null int64
Handcap           110527 non-null int64
SMS_received      110527 non-null int64
No-show           110527 non-null object
dtypes: float64(1), int64(8), object(5)
memory usage: 11.8+ MB
```

Based on the `.info()` results, I have found some problem, so I will only comment which I think is wrong.

- `PatientId` : ID to identify the patient, must be int;
- `ScheduledDay` and `AppointmentDay` : Are dates, I will need to fix it;
- `Scholarship` , `Hipertension` , `Diabetes` , `No-show` , and `Alcoholism` : Boolean variables, I also need to fix it later.

Let's see the columns names, maybe it will be necessary an adjustment in the upper case to lower case.

In [7]:

```
# Print the columns names.
df_vit.columns
```

Out[7]:

```
Index(['PatientId', 'AppointmentID', 'Gender', 'ScheduledDay',
       'AppointmentDay', 'Age', 'Neighbourhood', 'Scholarship', 'Hipertens
ion',
       'Diabetes', 'Alcoholism', 'Handcap', 'SMS_received', 'No-show'],
      dtype='object')
```

Fortunately, I did not find any space between the variable's name, however, there is a plenty of variable with One uppercase in the middle of the variable. To put everything uniformly, later I will apply the `.lower()` .

The `.nunique()` method prints the number of unique values for each variable.

In [8]:

```
# Print the number of unique values for each variable.
df_vit.nunique()
```

Out[8]:

```
PatientId          62299
AppointmentID     110527
Gender                 2
ScheduledDay      103549
AppointmentDay        27
Age                  104
Neighbourhood         81
Scholarship            2
Hipertension           2
Diabetes               2
Alcoholism             2
Handcap                5
SMS_received           2
No-show                2
dtype: int64
```

The `.nunique()` method results are very clarifying because I realize some `PatientId` appointed more than once.

In [9]:

```
unique_patientid = round(100*(110527-62299)/110527,2),round(100*(62299)/110527,2)

print("Appointed more than once: {}%\nAppointed one time: {}%".format(*unique_patientid
))
```

```
Appointed more than once: 43.63%
Appointed one time: 56.37%
```

Let's take a look in the `Age` variable, I want to know the `.max()`, `.min()`, `.mean()` etc, the `.decribe()` shows a very good summary.

In [10]:

```
# Shows the mean, max, min etc.
df_vit['Age'].describe()
```

Out[10]:

```
count    110527.000000
mean         37.088874
std          23.110205
min          -1.000000
25%          18.000000
50%          37.000000
75%          55.000000
max         115.000000
Name: Age, dtype: float64
```

The minimum age of -1 is quite weird. So, I will print all the observations with less than zero age.

In [11]:

```
# Cases of age less than zero.
df_vit[df_vit.Age < 0]
```

Out[11]:

	PatientId	AppointmentID	Gender	ScheduledDay	AppointmentDay	Age	Neighbo
99832	4.659432e+14	5775010	F	2016-06-06T08:58:13Z	2016-06-06T00:00:00Z	-1	R

Only one observation has age below of zero, for this reason, in the next chapter (Data Cleaning) I will remove this observation.

I would also like to record when this appointment happens.

In [12]:

```
nov = sum(df_vit['ScheduledDay'].str.contains('2015-11'))
dez = sum(df_vit['ScheduledDay'].str.contains('2015-12'))
jan = sum(df_vit['ScheduledDay'].str.contains('2016-01'))
fev = sum(df_vit['ScheduledDay'].str.contains('2016-02'))
mar = sum(df_vit['ScheduledDay'].str.contains('2016-03'))
aph = sum(df_vit['ScheduledDay'].str.contains('2016-04'))
may = sum(df_vit['ScheduledDay'].str.contains('2016-05'))
jun = sum(df_vit['ScheduledDay'].str.contains('2016-06'))

print("nov: {}\ndez: {}\njan: {}\nfev: {}\nmar: {}\nmar: {}\naph: {}\njun: {}".format(n
ov,dez,jan,fev,mar,aph,may,jun))
```

```
nov: 1
dez: 61
jan: 60
fev: 281
mar: 3614
mar: 25339
aph: 67421
jun: 13750
```

The appointment distribution is very left-skewed, most of the record observation was made in April, March, and June.

Move on to `Handcap` variable, which has 5 levels:

- 0: No disabilities;
- 1: One disability;
- 2: Two disabilities, and so on.

In [13]:

```
# If a person has disabilities (blind, dumb, etc.)
df_vit['Handcap'].value_counts() # This is cummulative and I do not know what kind o di
sabilities each patient has.
```

Out[13]:

```
0    108286
1      2042
2       183
3        13
4         3
Name: Handcap, dtype: int64
```

After that, most of the patients do not have disabilities.

Let's see the patient gender.

In [14]:

```
# Gender
df_vit.Gender.value_counts()
```

Out[14]:

```
F    71840
M    38687
Name: Gender, dtype: int64
```

Women are predominant in this medical appointments.

The next variable is `SMS_received` , adopting the interpretation of the creator/maintainer of the dataset, this variable is the number of messages sent/received by the patient.

In [15]:

```
df_vit.SMS_received.value_counts()
```

Out[15]:

```
0    75045
1    35482
Name: SMS_received, dtype: int64
```

This variable looks like a boolean, but let's assume as quantity because the maintainer said this is the number of messages sent to the patient.

The next 4 variables are all boolean, so I will bind the results into a single table.

In [16]:

```
# Just o save some lines of report
b_family = df_vit['Scholarship'].value_counts() # Counts the number of Scholarship
b_alcohol = df_vit['Alcoholism'].value_counts() # Counts the number of Alcoholism
b_hiper = df_vit['Hipertension'].value_counts() # Counts the number of Hipertension
b_diab = df_vit['Diabetes'].value_counts()      # Counts the number of Diabetes

pd.concat([b_family,b_alcohol,b_hiper,b_diab],axis = 1)
```

Out[16]:

	Scholarship	Alcoholism	Hipertension	Diabetes
0	99666	107167	88726	102584
1	10861	3360	21801	7943

After this extense Data Assessing, let start the Data Clean.

Data Cleaning

In this chapter I will edit the `df_vit` dataframe modifying some variables types, removing non-desireable observations, etc.

Based on the long Data Assessing, this dataframe is quite clean, I just need to convert some variable and remove a non-standard negative age.

Renaming Columns names

Renaming the columns will turn the study much easier and fast, due to the uppercase in the middle of the variable names, it's annoying because you must record the exact form to type.

In [17]:

```
# The rename methods deals with changing dash to underscores and lower casing everythin
g.
df_vit.rename(columns=lambda x: x.strip().lower().replace("-", "_"), inplace=True)

# Print columns name to ensure the modification.
df_vit.columns
```

Out[17]:

```
Index(['patientid', 'appointmentid', 'gender', 'scheduledday',
       'appointmentday', 'age', 'neighbourhood', 'scholarship', 'hipertens
ion',
       'diabetes', 'alcoholism', 'handcap', 'sms_received', 'no_show'],
      dtype='object')
```

This minor changing will save me time in future, avoiding me to commit typos problems.

Converting Data Types

Now, let's convert `patientid`, `scheduledday`, and `appointmentday`.

In [18]:

```
# Convert float to int
df_vit.patientid = df_vit.patientid.astype(np.int64) # I am coercing to int64, because
 the default converts to int32.
```

The conversion of `scheduledday` and `appointmentday` will be performed by the `numpy` package.

In [19]:

```
# Converting the ScheduledDay to date.
df_vit.scheduledday = df_vit.scheduledday.apply(np.datetime64)

# Converting the AppointmentDay to date.
df_vit.appointmentday = df_vit.appointmentday.apply(np.datetime64)

# Print the first 5 rows.
df_vit.head()
```

Out[19]:

	patientid	appointmentid	gender	scheduledday	appointmentday	age	neighb
0	29872499824296	5642903	F	2016-04-29 18:38:08	2016-04-29	62	JAI
1	558997776694438	5642503	M	2016-04-29 16:08:27	2016-04-29	56	JAI
2	4262962299951	5642549	F	2016-04-29 16:19:04	2016-04-29	62	N
3	867951213174	5642828	F	2016-04-29 17:29:31	2016-04-29	8	POI C
4	8841186448183	5642494	F	2016-04-29 16:07:23	2016-04-29	56	JAI

Both `datetime` variables are converted but have minor differences.

- `scheduledday` : It is OK;
- `appointmentday` : Lose the hours, minutes, and seconds, probably is due to all the cases have the same time `00:00:00`.

Now, let's convert the categorical variable, such as:

- `gender` : male and female categories;
- `no-show` : yes or no.

In [20]:

```
# Converting the No-show variable as categorical
df_vit['no_show'] = pd.Categorical(df_vit['no_show'])

# Converting the Gender variable as categorical
df_vit['gender'] = pd.Categorical(df_vit['gender'])
```

Finally, I will convert all boolean variables.

In [21]:

```
# Converting int64 variables to bool.
df_vit.scholarship = df_vit.scholarship.apply(bool)
df_vit.hipertension = df_vit.hipertension.apply(bool)
df_vit.diabetes = df_vit.diabetes.apply(bool)
df_vit.alcoholism = df_vit.alcoholism.apply(bool)
```

Let's print the `.info()` once again to ensure the conversion.

In [22]:

```
df_vit.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 110527 entries, 0 to 110526
Data columns (total 14 columns):
patientid         110527 non-null int64
appointmentid     110527 non-null int64
gender            110527 non-null category
scheduledday      110527 non-null datetime64[ns]
appointmentday    110527 non-null datetime64[ns]
age               110527 non-null int64
neighbourhood     110527 non-null object
scholarship       110527 non-null bool
hipertension      110527 non-null bool
diabetes          110527 non-null bool
alcoholism        110527 non-null bool
handcap           110527 non-null int64
sms_received      110527 non-null int64
no_show           110527 non-null category
dtypes: bool(4), category(2), datetime64[ns](2), int64(5), object(1)
memory usage: 7.4+ MB
```

Data Filtering

I will remove all observations with age below of zero because it is not correct. I know there is only one observation with this characteristics, it is not likely this deletion will affect all the analysis.

In [23]:

```
# Cases of Age less than zero. I have updated my variable name to lowercase age.
df_vit = df_vit[df_vit.age >= 0]

# Print the number of rows
print("Number of rows: {}".format(df_vit.shape[0]))
```

```
Number of rows: 110526
```

As I expected, the number of rows has decreased one unit.

Here, I finished the Data Wrangling.

Exploratory Data Analysis

Sadly, I did not come up with a good and interesting question, but as a kickstart to my Exploratory Analysis, I will try to find any relationship between the patient who has appointed more than one time and no-showed.

> My Question: Is there a relationship between patients with many appointments to do show-up more frequently?

First, I want to know how is the percentage of a no-show of all appointments.

In [24]:

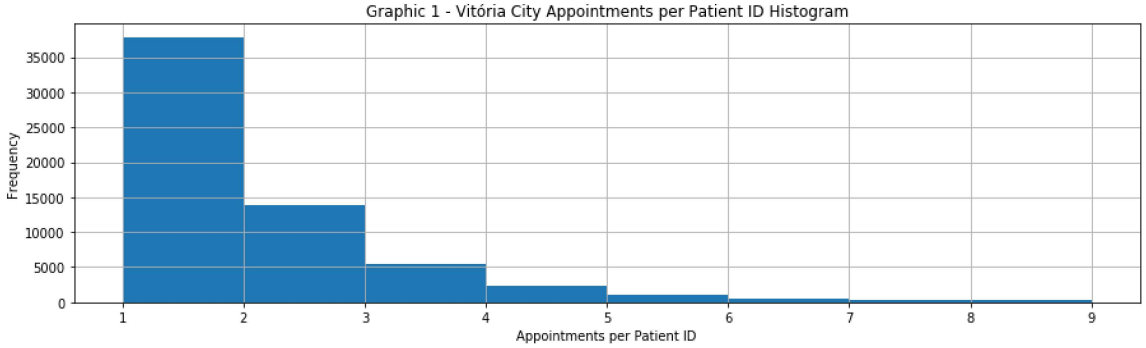
```
no_show_p = df_vit['no_show'].value_counts() # First show-up and second no-show.

print(no_show_p)

print("Appointments with Show-up: {}\nAppointments with No-show: {}\n".format(*no_show_
p))  # No-show: Yes : 22319
print("No-show Percentage: {}%".format(round(100*no_show_p[1]/sum(no_show_p)),2))
     # Show-up: No  : 88207
```

```
No     88207
Yes    22319
Name: no_show, dtype: int64
Appointments with Show-up: 88207
Appointments with No-show: 22319

No-show Percentage: 20.0%
```

The no-show rate is 20.0% for all appointments but I know that some patients have an appointment more than once. Let's get deep in this question plotting the histogram.

In [25]:

```
# Histogram of patient id.
df_vit['patientid'].value_counts().hist(bins = [1, 2, 3, 4, 5, 6, 7, 8, 9], # 9 Levels
 of bons to be showed
                                        figsize= [15,4],                   # Strech a
 image to horizontal
                                        cumulative = False);               # Do not cu
mulative

plt.title('Graphic 1 - Vitória City Appointments per Patient ID Histogram') # Add title
plt.xlabel('Appointments per Patient ID')                                   # X axis La
bel
plt.ylabel('Frequency');                                                    # Y axis La
bel
```



Although it seems to be **not** clear that only a few patients have several appointments. I will plot the `patientid` with most appointments to analyse this in details.

In [26]:

```
# This is the patientid with multiple appointments.
df_vit['patientid'].value_counts().head(5)
```

Out[26]:

```
822145925426128    88
99637671331        84
26886125921145     70
33534783483176     65
258424392677       62
Name: patientid, dtype: int64
```

As you can see, there are some patients with high frequency, for this reason, I would like to test the theory of [80/20 (https://en.wikipedia.org/wiki/Pareto_principle)](https://en.wikipedia.org/wiki/Pareto_principle), so I will `.sum()` the first 20% of `patientid` with higher frequency.

In [27]:

```
# Number of unique patient id
unq_pat = df_vit['patientid'].nunique()

# Summation of the 20% patients with higher number of appointments
res = sum(df_vit['patientid'].value_counts().head(int(unq_pat * 0.20)))   # I use the i
nt() to ensure a integer number as index.

# Print
print('Number of appointments    : {}'.format(res))
print('Number of appointments (%): {}%'.format(round(100*res/110526),2))
```

```
Number of appointments    : 48767
Number of appointments (%): 44%
```

The 20% with higher frequency (`patientid` with many appointments) are responsible for 44% of the total appointments, it is far from the 80%, but it shows a kind of "asymmetry".

Question 1

> **Are patients with many appointments are more likely to show-up then the patients with less than two appointments?**

Let's divide the dataset into two groups using a threshold of at least 3 appointments to classify each patientid.

- First group (`df_gp1`) that one with a higher number of appointments, and;
- Second group (`df_gp2`) that one with a lower number of appointments.

In [28]:

```
# Counts the number of appointment for each patient id.                    # I have
 decided to split in several lines
n_appoin_per_pat_id = df_vit['patientid'].value_counts()                   # because
 in the future this is much easier for
                                                                           # me to r
evise this code, and to understand.
# Creates a "vector" of boolean (True or False) for each patient id.
is_greater_3 = n_appoin_per_pat_id >= 3
is_not_great_3 = n_appoin_per_pat_id < 3

# Quantity of Patients with more or equal 3 appointments.
is_greater_3.value_counts()
```

Out[28]:

```
False    51814
True     10484
Name: patientid, dtype: int64
```

There are 10,484 patients with more than or equal to 3 appointments and 51,814 patients with less than 3 appointments.

Obs.: I have read [this (https://stackoverflow.com/questions/26640145/python-pandas-how-to-get-the-row-names-from-index-of-a-dataframe)](https://stackoverflow.com/questions/26640145/python-pandas-how-to-get-the-row-names-from-index-of-a-dataframe) thread in Stack overflow to understand how to extract the "rows names."

In [29]:

```
# Creating a list of patientid with more than 3 appointments.
more_3 = list(n_appoin_per_pat_id[is_greater_3].index)      # List of patient id with m
ore than 3 appointments
less_3 = list(n_appoin_per_pat_id[is_not_great_3].index)    # List of patient id less t
han 3 appointments

# First 10 patients id with more than 3 appointments.
more_3[:10] # Slicing an Example
```

Out[29]:

```
[822145925426128,
 99637671331,
 26886125921145,
 33534783483176,
 258424392677,
 75797461494159,
 871374938638855,
 6264198675331,
 66844879846766,
 872278549442]
```

The `more_3` and `less_3` are two lists of `patientid`, they will be used to select the rows of `df_vit`.

In [30]:

```
# Testing if any less of information.
print(df_vit['patientid'].nunique() == len(more_3) + len(less_3)) # The total must be e
qual to these two subsets.
```

```
True
```

I have inserted a test to ensure if these two subsetted data frames have the same quantity of observation from the original dataframe. If the results are `True`, is everything OK!

In [31]:

```
# Print the number of patient id in each list.
print("Dataframe with higher number appointments: {}\nDataframe with lower number appoi
ntments: {}".format(len(more_3),len(less_3)))
```

```
Dataframe with higher number appointments: 10484
Dataframe with lower number appointments: 51814
```

These two vectors (`more_3` and `less_3`) will be used to "filter" and separate which rows pertain to `df_gp1` or `df_gp2` .

Obs.: I have read [this (https://stackoverflow.com/questions/12065885/filter-dataframe-rows-if-value-in-column-is-in-a-set-list-of-values)](https://stackoverflow.com/questions/12065885/filter-dataframe-rows-if-value-in-column-is-in-a-set-list-of-values) Stack overflow thread to learn how to use `.isin()` .

In [32]:

```
# Subsetting the Dataframe with the Patients with more than 3 appointments
df_gp1 = df_vit[df_vit['patientid'].isin(more_3)]

# Subsetting the Dataframe with the Patients with less than 3 appointments
df_gp2 = df_vit[df_vit['patientid'].isin(less_3)]

# Printing the dimension of each dataframe
print("df_gp1\nObservations: {}\nColumns: {}\n\ndf_gp2\nObservations: {}\nColumns: {}".
format(*df_gp1.shape,*df_gp2.shape))
```

```
df_gp1
Observations: 44817
Columns: 14

df_gp2
Observations: 65709
Columns: 14
```

Both data frames are ready to calculate the no-show rate.

In [33]:

```
# Assign the results of value count to a variable, I need it to make calculations
df_gp1_res_no_show = df_gp1.no_show.value_counts()

# Percentage Calculation
df_gp1_res_no_show_p = 100 * df_gp1_res_no_show[1]/(df_gp1_res_no_show[0] + df_gp1_res_
no_show[1])

# Print results and rounding.
print("No-show: {}%".format(round(df_gp1_res_no_show_p,2)))
```

```
No-show: 21.09%
```

In [34]:

```
# Assign the results of value count to a variable, I need it to make calculations
df_gp2_res_no_show = df_gp2.no_show.value_counts()

# Percentage Calculation
df_gp2_res_no_show_p = 100 * df_gp2_res_no_show[1]/(df_gp2_res_no_show[0] + df_gp2_res_
no_show[1])

print("No-show: {}%".format(round(df_gp2_res_no_show_p,2)))
```

```
No-show: 19.58%
```

Although the difference between these two results is slight, patients from the group `df_gp1` have a higher percentage of a no-show from the `df_gp2` . Going to the opposite way from my expectations (and negating my question).

I will make a sensible analysis, varying the threshold of 3 appointments to 4, 5 and so on, therefore I will create a function to encapsulate this lines of codes.

In [35]:

```
# I decided to encapsulate in one function because it will save me line codes.
def attendance(data, n, complete = False):
    """
    ------------------------------------------------------------------------------
-----------------
    |DESCRIPTION:
                |
    |
                |
    |     This function calculate the no-show rate separating the dataframe into two gr
oup based on a   |
    |     threshold n.
                |
    |
                |
    ------------------------------------------------------------------------------
-----------------
    |INPUT:
                |
    |
                |
    |     VARIABLE   TYPE        DESCRIPTION
                |
    |
                |
    |     data       dataframe   This dataset must be the 'noshowappointments-kagglev2-
may-2016.csv'    |
    |                            imported file but it is required an treatment before u
se this function.|
    |                            The columns name should be renamed to lower case.
                |
    |
                |
    |     n          int         The threshold number of appointment to classify each p
atient id.       |
    |
                |
    |     complete   bool        If False return percentage and If True also returns da
ta frames.       |
    |
                |
    ------------------------------------------------------------------------------
-----------------
    |OUTPUT:
                |
    |
                |
    |     VARIABLE         TYPE   DESCRIPTION
                |
    |
                |
    |     [m_no_show_p,    list   The first element of the list is the percentage of no
-show of patients|
    |     l_no_show_p]            with more than or equal to n appointments in the data
 , the second    |
    |                             element is the percentage of the group with less than
 n appointments. |
    |
                |
```

```
    |     df_more,   data frame   The former with patientid with more or equal n appoin
tments and the   |
    |     df_less                 latter with patientid with less than n appointments.
                |
    |
                |
    ------------------------------------------------------------------------------
------------------
    """
    greater = data['patientid'].value_counts() >= n    # Creating a vector to filter th
e rows
    lesser = data['patientid'].value_counts() < n      # Creating a vector to filter th
e rows

    # Extracting the patient id as list
    more = list((data['patientid'].value_counts())[greater].index)
    less = list((data['patientid'].value_counts())[lesser].index)

    # Subsetting the data into two dataframes.
    df_more = data[data['patientid'].isin(more)]
    df_less = data[data['patientid'].isin(less)]

    # Calculation of no-show from the first dataframe (Patients with more than or equal
 to n appointments)
    m_no_show = df_more.no_show.value_counts()
    m_no_show_p = 100 * m_no_show[1]/(m_no_show[0] + m_no_show[1])

    # Calculation of no-show from the first dataframe (Patients with less than n appoin
tments)
    l_no_show = df_less.no_show.value_counts()
    l_no_show_p = 100 * l_no_show[1]/(l_no_show[0] + l_no_show[1])

    if complete == False:
        return [round(m_no_show_p,2), round(l_no_show_p,2)]  # Return results in percen
tage and in a list.
    else:
        return [round(m_no_show_p,2), round(l_no_show_p,2), df_more, df_less]
```

The function `attendance` will calculate for each quantity of appointments the probability of these two groups.

In [36]:

```
# Variable Initialization
res_no_show_2_88 = []

# Loop to generate for each index value (threshold) the probabilities.
for index in range(2,88):
    res_no_show_2_88.append(attendance(df_vit, index))
```

The variable `res_no_show_2_88` is a list with each element a pair of no-show percentage. For this reason, I need to unpack this list to plot in the graphic.

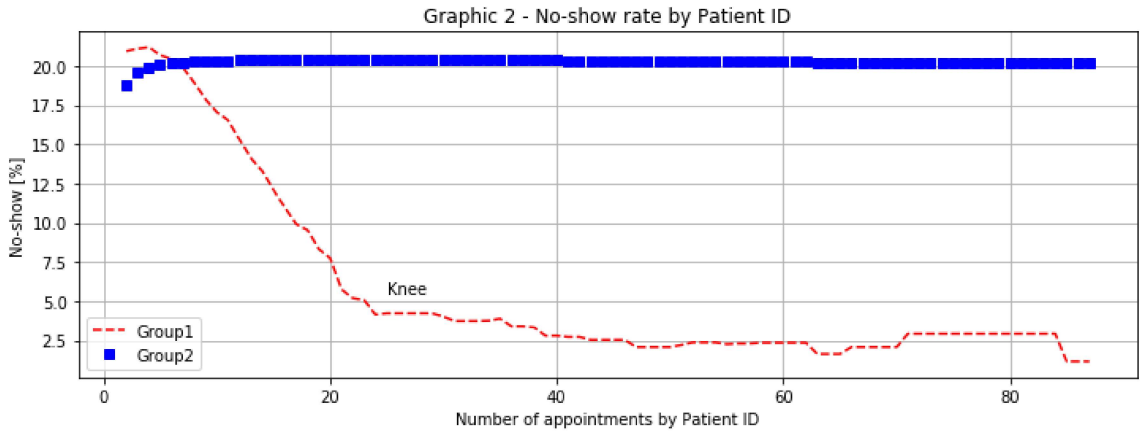
In [37]:

```
# Unpacking the res_no_show_2_88 variable
no_show_2_88, show_up_2_88 = zip(*res_no_show_2_88)

# X axis variable
threshold = np.arange(2,88)
```

I have created two variables (`no_show_2_88` and `show_up_2_88`) to be plotted and a third variable to simulate my threshold number (`threshold`).

In [38]:

```
# Defining the size of the Plot.
plt.figure(figsize= [12,4])

# red dashes, blue squares and green triangles
plt.plot(threshold, no_show_2_88, 'r--',threshold,show_up_2_88, 'bs')

plt.xlabel('Number of appointments by Patient ID') # X label
plt.ylabel('No-show [%]')                          # Y label
plt.title('Graphic 2 - No-show rate by Patient ID')     # Graphic Title
plt.text(25, 5.5, r'Knee')                         # Text
plt.grid(True)                                     # Grid
plt.legend(('Group1', 'Group2'))                   # Legend

plt.show() # Plot the graphic
```



The graphic above shows the behaviour of the no-show rate when the threshold varies from 2 to 88. Keep in mind, that I defined threshold as the number of appointments by each patient.

Analyzing the red line (Group 1):

- Roughly, from 7 to 25 there is a negative linear relationship between the patient with more appointment and no-show rate;
  - Patient tend to do not miss an appointment;
- Patients with more than 25 appointments have a very low no-show rate.

Analyzing the blue line (Group 2):

- Do not show any changes varying the threshold.

Answer Question 1

> As I expected the higher appointment frequency given of a `patientid` leads a higher shop-ups (or lower no-show), this is probably a consequence from the illness severity (chemotherapy, radiotherapy, dialysis, etc.), which obliges the patient to go several times to the hospital.

I have recognised a shortage of information, that is why I will stop here this analysis to collect new data. Later, I will come back to a deeper analysis because I think there is an opportunity for more exploratory analysis.

Data Merging

I have gathered some additional data from Vitória Council Website (http://legado.vitoria.es.gov.br/regionais/home.asp), let's load them to make a new analysis. I have not yet figure out the second question and binding this additional data could lead me a new insight.

In [39]:

```
# Loading the Vitoria Auxiliary data.
vit_aux = pd.read_csv('02-Datasets/vit_aux.csv', sep=";");

# Print the first 5 rows.
vit_aux.head()
```

Out[39]:

	id_ra	reg_adm	nbh	pop_2000	pop_2010	male	female	avg_inc_mon
0	1	Centro	Centro	9240	9838	4416	5422	1791,15
1	1	Centro	Do Moscoso	854	795	369	426	647,05
2	1	Centro	Fonte Grande	1459	1231	592	639	706,03
3	1	Centro	Ilha do Príncipe	2810	2613	1194	1419	623,21
4	1	Centro	Parque Moscoso	1708	1773	791	982	1754,79

In [40]:

```
df_vit.head(2)
```

Out[40]:

	patientid	appointmentid	gender	scheduledday	appointmentday	age	neighbourh
0	29872499824296	5642903	F	2016-04-29 18:38:08	2016-04-29	62	JARDIM PEI
1	558997776694438	5642503	M	2016-04-29 16:08:27	2016-04-29	56	JARDIM PEI

Comparing these two datasets, I realized the only variable in common is the neighborhood (or neighbourhood in Britain English), sadly the variable name and the content is not equivalent. It means I need to edit the column from both data frames. Again, I will use a tailored function to deal with this problem.

In [41]:

```
def lower_ahu(data):
    """
    -----------------------------------------------------------------------------
-----------------
    |DESCRIPTION:
                |
    |
                |
    |     This function converts any uppercase letter to lowercase, converts space into
 underscore and  |
    |     change any non-standard letter such as: á,é,í,ó,ú,ã,õ,ç etc. to a "regular" l
etter without the|
    |     accent.
                |
    |
                |
    -----------------------------------------------------------------------------
-----------------
    |INPUT:
                |
    |
                |
    |     VARIABLE    TYPE    DESCRIPTION
                |
    |
                |
    |     data        list    The input expected a list with strings.
                |
    |
                |
    |
                |
    -----------------------------------------------------------------------------
-----------------
    |OUTPUT:
                |
    |
                |
    |     VARIABLE    TYPE    DESCRIPTION
                |
    |
                |
    |     data        list    The converted version of the list, without any word with we
ird accent.      |
    |
                |
    -----------------------------------------------------------------------------
-----------------
    """
    # This helps to reduce the number of "types" of letters.
    data = [x.lower() for x in data]

    # My list of non standard letter and its accents.
    non_standard = [['ã','a'],['á','a'],['à','a'],['â','a'],
                    ['é','e'],['è','e'],['ê','e'],
                    ['í','i'],['ì','i'],
                    ['õ','o'],['ó','o'],['ò','o'],['ô','o'],
                    ['ú','u'],['ù','u'],
                    ['ç','c'],
```

```
                    [" ",'_'],["-",'_'],["'",'_'],["`",'_'],["´",'_'],[",",'.']]

    for wrong,correct in non_standard:
        data = [x.replace(wrong,correct) for x in data] # Where the magic happens!

    return data # Return the data without acccents.
```

After defining a function, let's use it.

In [42]:

```
# Converting the content of the nbh varible to lowercase and removing portuguese accent
s.
vit_aux['nbh'] = lower_ahu(vit_aux['nbh'])
vit_aux['reg_adm'] = lower_ahu(vit_aux['reg_adm'])

# Converting the content of the neighbourhood varible to lowercase and removing portugu
ese accents.
df_vit['neighbourhood'] = lower_ahu(df_vit['neighbourhood'])

# Let's compare both categories.
sorted(list(vit_aux.nbh.unique())) is sorted(list(df_vit.neighbourhood.unique()))
```

Out[42]:

```
False
```

Unfortunately, is `False` . Let's dig to find out the problem of different numbers of neighborhoods.

In [43]:

```
# Number of neighborhood in each dataframe.
nbh_comparison = len(vit_aux.nbh.unique()),len(df_vit.neighbourhood.unique())

print("Auxiliary Dataframe: {}\nVitória Hospital Appointment: {}".format(*nbh_compariso
n))
```

```
Auxiliary Dataframe: 80
Vitória Hospital Appointment: 81
```

So, there is one neighborhood lacking in `vit_aux` data frame, and this neighborhood, after few minutes comparing these two lists, is `ilhas_oceanicas_de_trindade` . Curiously, this island is 1,200 kilometers far from the coast, it means this is an ultra-marine territory, a place where only 32 Marines lives and no one else.

In [44]:

```
# Print all observation from ilhas_oceanicas_de_trindade.
df_vit[df_vit['neighbourhood'] == 'ilhas_oceanicas_de_trindade']
```

Out[44]:

	patientid	appointmentid	gender	scheduledday	appointmentday	age	
48754	534986855114	5583947	F	2016-04-14 12:25:43	2016-05-13	51	ilhas_oce
48765	7256429752481	5583948	F	2016-04-14 12:26:13	2016-05-13	58	ilhas_oce

As you can see, there are only 2 observations, from two different patients, and both no-showed. Due to the inconsistency of this two observations, I will remove them because I am convinced doing it will not affect my results.

In [45]:

```
# Creating a vector to remove only the observations with the ilhas_oceanicas_de_trindad
e as neighbourhood.
rm_trindade = list(map(lambda x : not(x), df_vit['neighbourhood'] == 'ilhas_oceanicas_d
e_trindade'))

# Subsetting using the vector rm_trindade to remove this observations.
df_vit = df_vit[rm_trindade]

# Print dimensions
print("Rows: {}\nColumns: {}".format(*df_vit.shape))
```

```
Rows: 110524
Columns: 14
```

Finally, I can compare the neighborhood columns of this two data frames.

In [46]:

```
# Comparing the categories of these two columns. If true, I could join them.
sorted(list(df_vit.neighbourhood.unique())) == sorted(list(vit_aux.nbh.unique()))
```

Out[46]:

```
True
```

It is OK, so I will continue to rename the columns to have the same name (`neighbourhood` will be replaced by `nbh`).

In [47]:

```
# Creating a vector to assign a new name to the variable neighbourhood
new_name_column = list(df_vit.columns)

# Renaming the columns names
new_name_column[6] = 'nbh'

# Defining variables names.
df_vit.columns = new_name_column
```

Before any merge, I will convert the `avg_inc_mon` to `float64`. You can see `avg_inc_mon` is originally imported as `str/obj`.

In [48]:

```
vit_aux.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 80 entries, 0 to 79
Data columns (total 8 columns):
id_ra          80 non-null int64
reg_adm        80 non-null object
nbh            80 non-null object
pop_2000       80 non-null int64
pop_2010       80 non-null int64
male           80 non-null int64
female         80 non-null int64
avg_inc_mon    80 non-null object
dtypes: int64(5), object(3)
memory usage: 5.1+ KB
```

In [49]:

```
# Converting the number Brazillian formatting to USA - Changing , to .
vit_aux['avg_inc_mon'] = lower_ahu(vit_aux['avg_inc_mon'])

# Converting str/obj to float
vit_aux['avg_inc_mon'] = vit_aux['avg_inc_mon'].astype(float)
```

Everything is set-up and ready to the merge.

Obs.: I have read in pandas website how to merge, [this (https://pandas.pydata.org/pandas-docs/stable/merging.html#database-style-dataframe-joining-merging)](https://pandas.pydata.org/pandas-docs/stable/merging.html#database-style-dataframe-joining-merging) is the thread I have read.

In [50]:

```
# Merging the two datasets.
df_vit = pd.merge(df_vit, vit_aux, how='left', on=['nbh', 'nbh'])

# Print the first 5 rows
df_vit.head()
```

Out[50]:

	patientid	appointmentid	gender	scheduledday	appointmentday	age	
0	29872499824296	5642903	F	2016-04-29 18:38:08	2016-04-29	62	jardim_da
1	558997776694438	5642503	M	2016-04-29 16:08:27	2016-04-29	56	jardim_da
2	4262962299951	5642549	F	2016-04-29 16:19:04	2016-04-29	62	mata_d
3	867951213174	5642828	F	2016-04-29 17:29:31	2016-04-29	8	pontal_de_c
4	8841186448183	5642494	F	2016-04-29 16:07:23	2016-04-29	56	jardim_da

5 rows × 21 columns

Now, the `df_vit` has few more variables, such as:

- Region Administration Name;
- Population in 2000 and 2010 this Regional Administration;
- Number of male residents;
- Number of female residents, and;
- Monthly Average Income in BRL.

Let's see the `.describe()` and `.info()` to ensure if there are any problems during the merging process.

In [51]:

```
df_vit.describe()
```

Out[51]:

	patientid	appointmentid	age	handcap	sms_received	id
count	1.105240e+05	1.105240e+05	110524.000000	110524.000000	110524.000000	110524.000
mean	1.474960e+14	5.675306e+06	37.088904	0.022249	0.321034	4.600
std	2.560959e+14	7.129503e+04	23.110111	0.161545	0.466876	2.309
min	3.921700e+04	5.030230e+06	0.000000	0.000000	0.000000	1.000
25%	4.172693e+12	5.640287e+06	18.000000	0.000000	0.000000	3.000
50%	3.173184e+13	5.680574e+06	37.000000	0.000000	0.000000	4.000
75%	9.439068e+13	5.725523e+06	55.000000	0.000000	1.000000	7.000
max	9.999816e+14	5.790484e+06	115.000000	4.000000	1.000000	9.000

In [52]:

```
df_vit.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 110524 entries, 0 to 110523
Data columns (total 21 columns):
patientid          110524 non-null int64
appointmentid      110524 non-null int64
gender             110524 non-null category
scheduledday       110524 non-null datetime64[ns]
appointmentday     110524 non-null datetime64[ns]
age                110524 non-null int64
nbh                110524 non-null object
scholarship        110524 non-null bool
hipertension       110524 non-null bool
diabetes           110524 non-null bool
alcoholism         110524 non-null bool
handcap            110524 non-null int64
sms_received       110524 non-null int64
no_show            110524 non-null category
id_ra              110524 non-null int64
reg_adm            110524 non-null object
pop_2000           110524 non-null int64
pop_2010           110524 non-null int64
male               110524 non-null int64
female             110524 non-null int64
avg_inc_mon        110524 non-null float64
dtypes: bool(4), category(2), datetime64[ns](2), float64(1), int64(10), ob
ject(2)
memory usage: 14.1+ MB
```

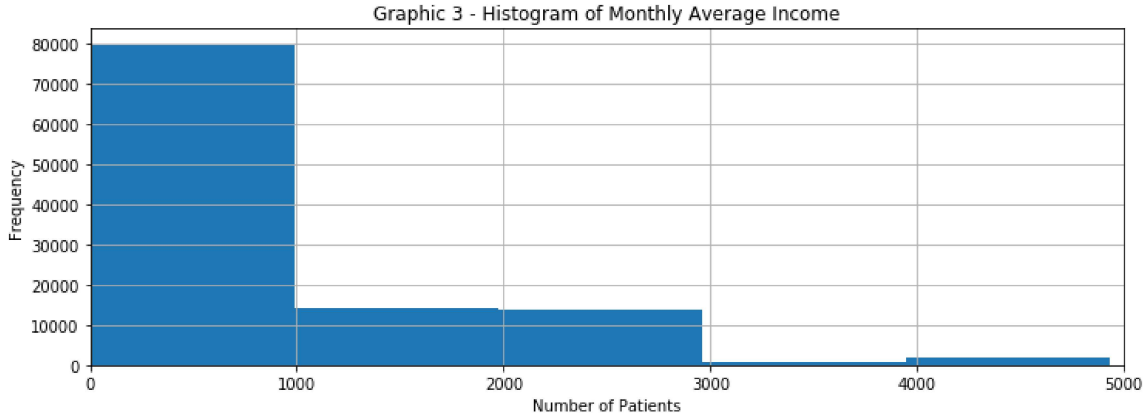
Everything looks pretty good but before any question, I would like to point out some consideration:

- The Monthly Average Income is an information based on the residents of the neighbourhood, and I assume all patient will choose the closest UBS from its residents to go, and;
- Male and Female have the same Monthly Average Income.

So let's start some interesting analysis, with these new features.

In [53]:

```
# Monthly Average Income Histogram
df_vit['avg_inc_mon'].hist(bins = 11, figsize = [12,4]);
plt.xlim(0,5000)

plt.title('Graphic 3 - Histogram of Monthly Average Income')        # Add title
plt.xlabel('Number of Patients')                                    # X axis Label
plt.ylabel('Frequency');                                            # Y axis Label
```



Concretely, the majority of the public attended by the public health systems in Vitória has average income bearing 1000 BRL/month.

However, in some cases one `patientid` has multiple entries, it means this patient went several times to the hospital, and so I must remove these duplicated `patientid` to a further analysis, for instance, to calculate the `.mean()` and `.count()`.

Let's remove the duplicated `patientid` from the `data_vit`.

In [54]:

```
# Creating a vector to sinalize if there is a duplicated patientid
dupe = df_vit['patientid'].duplicated()

# Converting to Not -> True to False.
dupe = list(map( lambda x : not(x) , dupe ) )

# Selecting the unique patients, no duplicated.
df_clean = df_vit[dupe]

# Dimensions of the cleaned data set.
df_clean.shape  # The number of rows from shape is the same from len(df_vit['patienti
d'].unique())
```

Out[54]:

```
(62296, 21)
```

I want to calculated the average age and the average income from all differents patients.

In [55]:

```
# Average Age Calculation
print("{} years".format(round(df_clean['age'].mean(),2)))
print("{} BRL".format(round(df_clean['avg_inc_mon'].mean(),2)))
```

```
36.7 years
1068.71 BRL
```

I will encapsulate a function to deal with removing duplicated `patientid` .

In [56]:

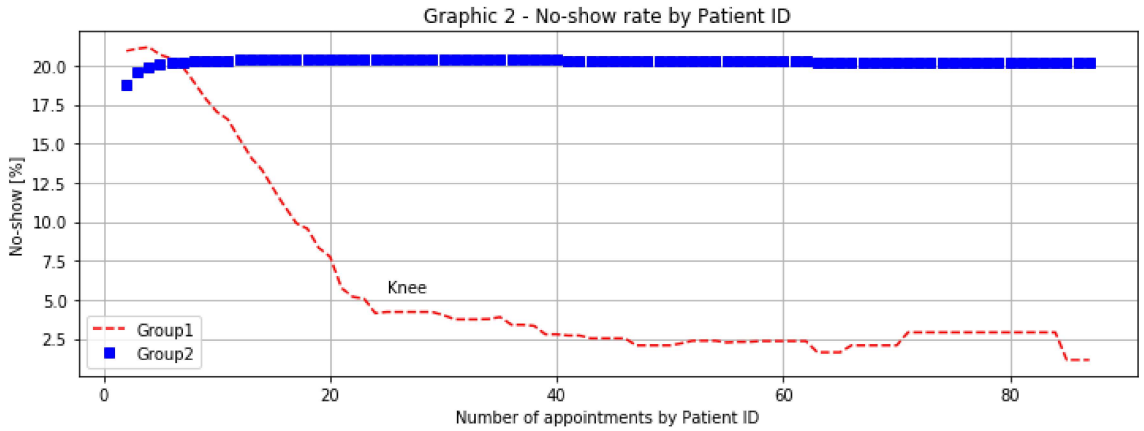
```
# Defining a function to eliminate the duplicates patientid.
def no_dupe_pat(data_dupe):
    """
    -----------------------------------------------------------------------------------
-----------------
    |DESCRIPTION:
                |
    |
                |
    |     This function removes all duplicated rows of a given patientid.
                |
    |
                |
    -----------------------------------------------------------------------------------
-----------------
    |INPUT:
                |
    |
                |
    |     VARIABLE    TYPE         DESCRIPTION
                |
    |
                |
    |     data_dupe   data frame   Any data frame with at least the patientid column.
                |
    |
                |
    |
                |
    -----------------------------------------------------------------------------------
-----------------
    |OUTPUT:
                |
    |
                |
    |     VARIABLE    TYPE         DESCRIPTION
                |
    |
                |
    |     data_dupe   data frame   A data frame with no duplicated patientid.
                |
    |
                |
    -----------------------------------------------------------------------------------
-----------------
    """
    # Creating a vector to sinalize if there is a duplicated patientid
    dupe_less = data_dupe['patientid'].duplicated()

    # Converting to Not -> True to False.
    dupe_less = list(map( lambda x : not(x) , dupe_less ) )

    # Selecting the unique patients, no duplicated and return.
    return data_dupe[dupe_less]
```

With this function, I can calculate more accurate values of the average mean, average income, etc.

I will come back to the first question I have made to go deeper and to answer some new questions. I will reproduce the Graphic 2 as a starter.

In [57]:

```
# Defining the size of the Plot.
plt.figure(figsize= [12,4])

# red dashes, blue squares and green triangles
plt.plot(threshold, no_show_2_88, 'r--',threshold,show_up_2_88, 'bs')

plt.xlabel('Number of appointments by Patient ID') # X label
plt.ylabel('No-show [%]')                          # Y label
plt.title('Graphic 2 - No-show rate by Patient ID')     # Graphic Title
plt.text(25, 5.5, r'Knee')                         # Text
plt.grid(True)                                     # Grid
plt.legend(('Group1', 'Group2'))                   # Legend

plt.show() # Plot the graphic
```



I am going to split the dataset approximately in the knee (I guess 25 is a good approximation) value of the graphic.

In [58]:

```
# Calculating the percentage of no-show and data frames.
no_show_more_25,no_show_less_25,df_more_25,df_less_25 = attendance(df_vit, 25, complete
= True)
```

After munging the data, I come up with a question.

Question 2

> **Are patients with many appointments in average older than those with few appointments?**

In [59]:

```
# Create a list of the average age of each data frame.
avg_age_25 = [round(df_more_25['age'].mean(),2), round(df_less_25['age'].mean(),2)]

# Print
print("Average Age for Patients with more or equal to 25 appointments: {} years".format
(avg_age_25[0]))
print("Average Age for Patients with less than 25 appointments: {} years".format(avg_ag
e_25[1]))
```

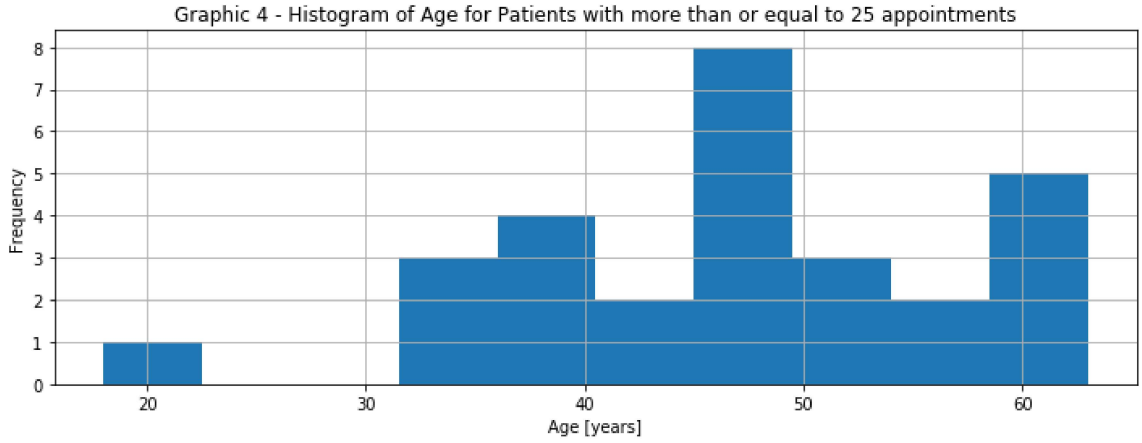
```
Average Age for Patients with more or equal to 25 appointments: 46.99 year
s
Average Age for Patients with less than 25 appointments: 36.96 years
```

Well, let's take a look in the distribution.

Before it, I will explain my function `no_dupe_pat()` , this function aims to remove any duplicated rows given a `patientid` , it means: If a `patientid` has 88 appointments, 87 will be excluded.

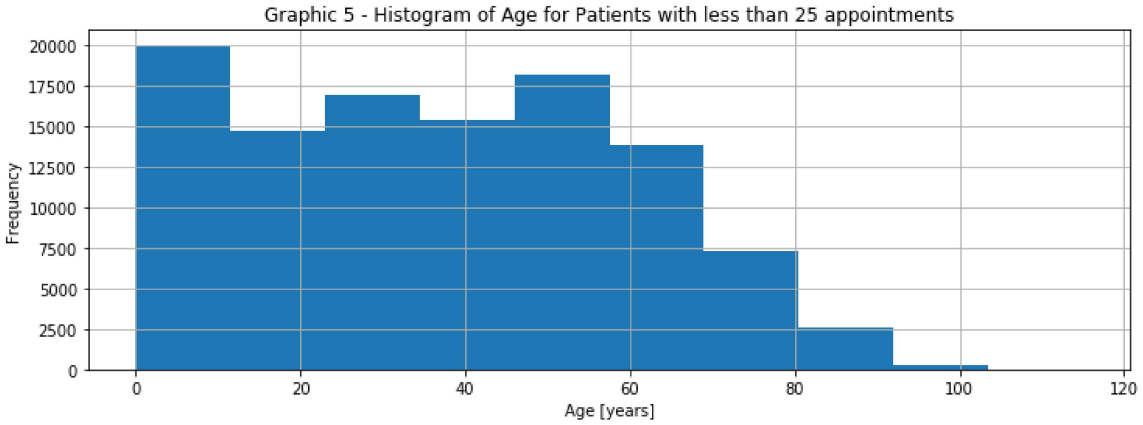
In [60]:

```
# Remove duplicates entries in patientid and plot the histogram of ages - Only patients
 with more or equal 25 apppointments.
no_dupe_pat(df_more_25).age.hist(figsize =[12,4]);

plt.title('Graphic 4 - Histogram of Age for Patients with more than or equal to 25 appo
intments')        # Add title
plt.xlabel('Age [years]')                                    # X axis Label
plt.ylabel('Frequency');                                     # Y axis Label
```



In [61]:

```
# Remove duplicates entries in patientid and plot the histogram of ages - Only patients
 with less than 25 apppointments.
df_less_25.age.hist(figsize =[12,4]);

plt.title('Graphic 5 - Histogram of Age for Patients with less than 25 appointments')
      # Add title
plt.xlabel('Age [years]')                                    # X axis Label
plt.ylabel('Frequency');                                     # Y axis Label
```



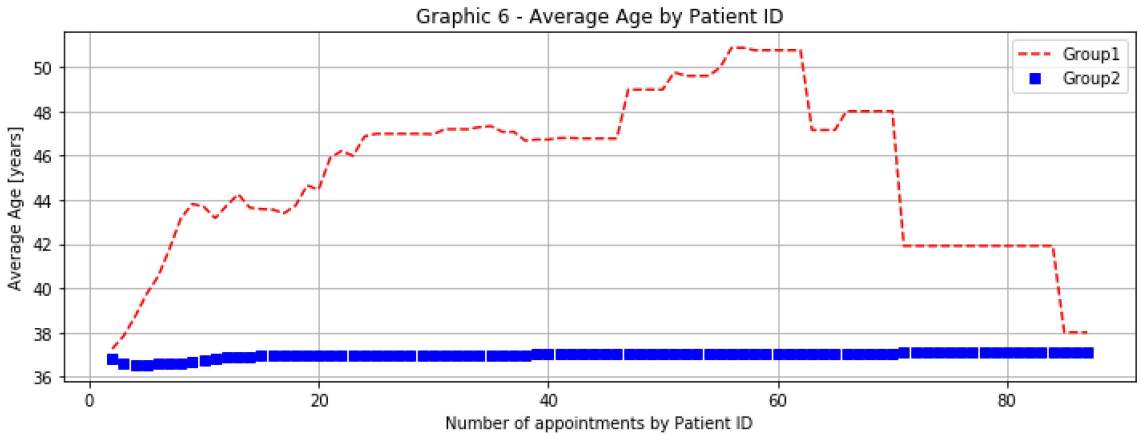
Based on these two histograms:

- There are few patients with more than or equal to 25 appointments (only 28 patients to be extactly);
  - The majority os these patients has more than 40 years;
- The group with less than 25 appointments has some similarities from 0 to 60 years, roughly, with the same frequency. After 60 years the frequency goes down steadily.

Only one point does not explain much of this question, so let's varies the number of appointments to plot a graph.

In [62]:

```
# Initializing variable
list_avg_age = []

# Loop to calculate the average age varying the number of appointment cummulate by pati
entid
for index in range(2,88):
    no_show_more, no_show_less, df_more, df_less = attendance(df_vit, index, complete=
True)
    list_avg_age.append([round(df_more['age'].mean(),2), round(df_less['age'].mean(),2
)]) # Calculate the .mean()

# Unpacking the list_avg_age variable.
avg_age_more, avg_age_less = zip(*list_avg_age)

# Generator of x axis.
threshold = np.arange(2,88)

# Defining the size of the Plot.
plt.figure(figsize= [12,4])

# red dashes, blue squares.
plt.plot(threshold, avg_age_more, 'r--',threshold,avg_age_less, 'bs')

plt.xlabel('Number of appointments by Patient ID')    # X label
plt.ylabel('Average Age [years]')                     # Y label
plt.title('Graphic 6 - Average Age by Patient ID')    # Graphic Title
plt.grid(True)                                        # Grid
plt.legend(('Group1', 'Group2'))                      # Legend

plt.show() # Plot the graphic
```



Based on the graphic, there is a positive relationship between the number of appointments by the patient and the average, however, this could be a trap, because the sample (the number of observation in the data frame with more than "x" appointments) became too small when I raises the Number of appointments by a patientid, Table 2 shows how the number of observations drop in the data frame so-called `df_more` .

Table 2 - Number of Patients for each Number of Cumulate Appointments.

k	Observations of df_more	Observations of df_less
2	24,379	37,917
3	10,484	51,812
4	4,984	57,312
5	2,617	59,679
10	333	61,963
25	28	62,268
50	13	62,283
88	1	62,295

Answer Question 2

> Seems the average age increases with the increase of the appointments, and it's also true to say the show-up rate increases with the increase in the number of appointment.

Question 3

> **Are patients with disabilities tend to have better rates of show-up? Is the number of disabilities raises the show-up rates?**

Let's start analysing the number of patients with disabilities.

In [63]:

```
# Number of appointments of Disabled Patient.
df_vit.handcap.value_counts()
```

Out[63]:

```
0    108283
1      2042
2       183
3        13
4         3
Name: handcap, dtype: int64
```

In [64]:

```
# Calculation of total appointments with Disabled Patient.
tot_dis = sum(df_vit.handcap.value_counts()) - df_vit.handcap.value_counts()[0]

print("The total of appointments with patients with disabilities: {}".format(tot_dis))
```

```
The total of appointments with patients with disabilities: 2241
```

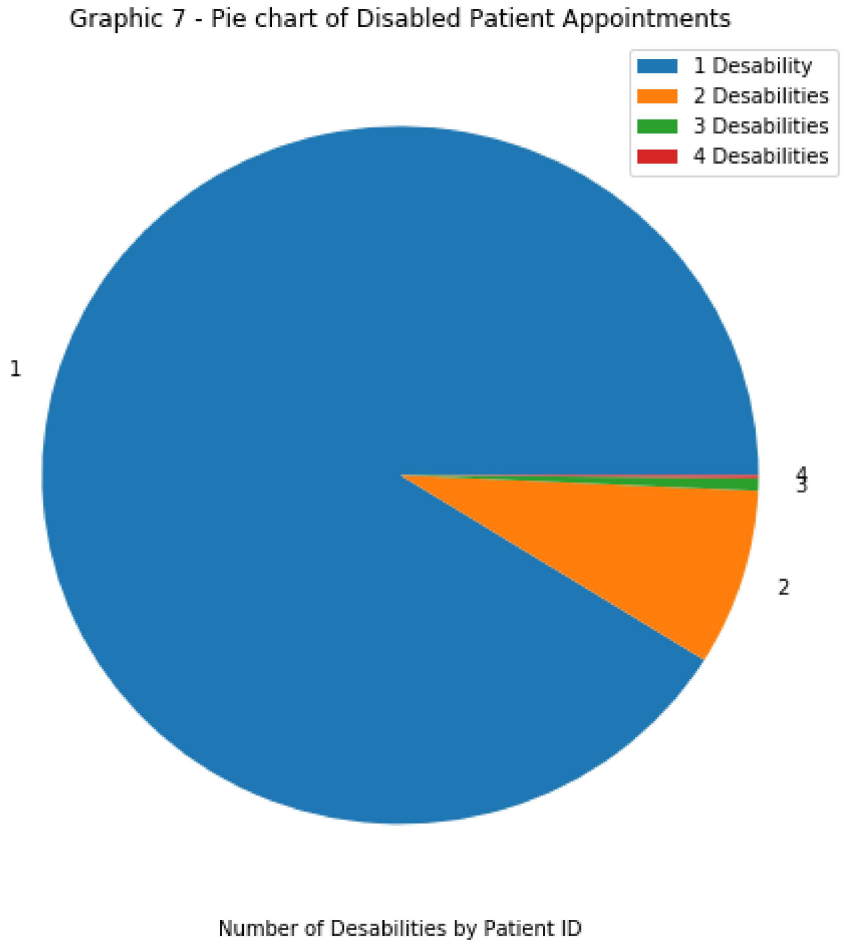
In [65]:

```
# Calculation of total Disabled Patient.
tot_pat_dis = sum(df_clean.handcap.value_counts()) - df_clean.handcap.value_counts()[0]

print("The total of patients with disabilities: {}".format(tot_pat_dis))
```

```
The total of patients with disabilities: 1133
```

In [66]:

```
# Subsetting to plot a graphic.
dis_gp = df_vit.handcap.value_counts();

# Creating the Pie Chart.
dis_gp[1:].plot(kind='pie',figsize = (8,8), title = 'Graphic 7 - Pie chart of Disabled
 Patient Appointments');
plt.xlabel('Number of Desabilities by Patient ID');    # X label
plt.ylabel('');                                        # Y label
plt.legend(('1 Desability', '2 Desabilities','3 Desabilities','4 Desabilities'));    #
 Legend
```



As you can see, almost 91% of the patient with disabilities has at least one disability, 8.1% has two disabilities, 0.6% has three disabilities, and 0.01% has four disabilities.

Now, I will calculate the no-show rate.

In [67]:

```
# variable initialization
no_show_dis_all = []
no_show_dis_all_p = []

# Loop to calculte the number of no-show and show-up.
for index in range(1,5):
    no_show_dis_all.append(list(df_vit[df_vit.handcap == index].no_show.value_counts
()))

# Unpacking the no_show_dis_all
no_show_dis,show_up_dis = zip(*no_show_dis_all)

# Loop to calculate the percentage.
for index in range(0,4):
    no_show_dis_all_p.append(round(100 * show_up_dis[index]/(no_show_dis[index] + show_
up_dis[index]),2))

# Print the percentage
print("1 Disability:   {}% no-show\n2 Disabilities: {}% no-show\n3 Disabilities: {}% no
-show\n4 Disabilities: {}% no-show".format(*no_show_dis_all_p))
```

```
1 Disability:   17.92% no-show
2 Disabilities: 20.22% no-show
3 Disabilities: 23.08% no-show
4 Disabilities: 33.33% no-show
```

In [68]:

```
# No-show and show-up of any patient with more than or equal to 1 disability.
dis_more_one = df_vit[df_vit.handcap >= 1].no_show.value_counts()

# Percentage
dis_more_one_p = 100 * dis_more_one[1]/(dis_more_one[0]+dis_more_one[1])

# No-show to the all sample of patients with more than or equal to 1 disability.
print("Rate of No-show for any level of handcap: {}%".format(round(dis_more_one_p,2)))
```

```
Rate of No-show for any level of handcap: 18.16%
```

Analysing each group of disability separated (1, 2, 3, and 4) the no-show increase if the number of disability increases. Keep in mind, for patients with 4 disabilities there are 3 patients, with 3 disabilities there are 6, with 2 disabilities there are 9, and with 1 disability there are 1,025 patients.

Answer Question 3

> Using all sample of disabilities the no-show rate (18.16%) is slightly lower than the role dataset (20%), so in average a patient with a disability could have a better rate of show-up.

Conclusions

Throughout of this document, I have pose three questions. I will repeat the same text from the Synopsis:

1. Are patients with many appointments are more likely to show-up then the patients with less than two appointments?

> Seems to be true, but I lack information to affirm it, what I could say is when I analyse patient with a higher number of appointments, they tend to have a better rate of show-up.

2. Are patients with many appointments in average older than those with few appointments?

> I also identify a positive correlation, when I analyse patients with higher number of appointments they tend to have an age average higher than the population.

3. Are patients with disabilities tend to have better rates of show-up? Is the number of disabilities raises the show-up rates?

> Due to the small number of patients with disabilities, it is difficult to answer this question, but I have found a better rate of show-up in the group of patients with disabilities, however, patients with many disabilities tend to have worse show-up rates.

Appendix A

This appendix aims to go in depth to find new data available on internet. Mostly of this information is public and disclosure in the Vitória Council Website.

Regional Administration Dataset

The city of Vitória is divided into 9 (nine) Regional Administration, according with the Vitória City Council Website (http://www.vitoria.es.gov.br/prefeitura/gerencias-regionais-veja-os-enderecos), do not expected to read any information in english, so the most important thing to gather in this page are the address of each suburbs administration building and the oficial name of suburbs administration.

Table A1 - Regional Administration Names and Address.

Regional Administation	Address	General Information
Central 1 - Centro	Praça Américo Poli Monjardim, Forte São João	See more (http://legado.vitoria.es.gov.br/regionais/dados_regiao/regiao_1/regiao1.asp)
Central 2 - Santo Antônio	Avenida Santo Antônio, 1400, Santo Antônio	See more (http://legado.vitoria.es.gov.br/regionais/dados_regiao/regiao_2/regiao2.asp)
Central 3 - Jucutuquara	Rua Santa Rita de Cássia, S/N, De Lourdes	See more (http://legado.vitoria.es.gov.br/regionais/dados_regiao/regiao_3/regiao3.asp)
Central 4 - Maruípe	Rua Marechal Floriano, 709, Maruípe	See more (http://legado.vitoria.es.gov.br/regionais/dados_regiao/regiao_4/regiao4.asp)
Central 5 - Praia do Canto	Avenida Rio Branco, 80, Santa Lúcia	See more (http://legado.vitoria.es.gov.br/regionais/dados_regiao/regiao_5/regiao5.asp)
Central 6 - Goiabeiras	Rua Desembargador Cassiano Castelo, 65, Goiabeiras	See more (http://legado.vitoria.es.gov.br/regionais/dados_regiao/regiao_6/regiao6.asp)
Central 7 - São Pedro	Avenida Beira Mar, 360, São Pedro	See more (http://legado.vitoria.es.gov.br/regionais/dados_regiao/regiao_7/regiao7.asp)
Central 8 - Jardim Camburi	Avenida Santos Evangelista, 15, Jardim Camburi	See more (http://legado.vitoria.es.gov.br/regionais/dados_regiao/regiao_8/regiao8.asp)
Central 9 - Jardim da Penha	Praça Philogomiro Lannes, S/N, Jardim da Penha	See more (http://legado.vitoria.es.gov.br/regionais/dados_regiao/regiao_9/regiao9.asp)

Going depth to find more information in the Vitória Council Website, I have found a map of each Regional administration (Table A1 Column General Information), with general information, such as:

- Number of Neighboor;
- Area (m²);
- Population (in 2010);
- Demographic density (inhabitants/km²);
- Numbers of home (in 2010);
- Average Income (in 2010);
- Economic Activities (in 2012).

I have recorded these information into one dataset (I do not know how to "webscrap" these infos, and probabily this is going to take too much time of my stydies to learn how to do it. I am new in Python). This dataset, co called `vitoria_reg_adm.csv` is available in my Github Repository. The Table A2 shows the description variables.

Table A2 - Data type of the vit_reg_adm.csv file.

Variable	Type	
id_ra	int	ID of the Regional Administration (Following the sequence stablished in Table A1
reg_adm	str	Regional Administration Name
nbh_qty	int	Quantity of Neighborhood in each Regional Administration
area_m2	float	Regional Administration Area in m²
pop_2010	int	Population in 2010
den_km2	int	Demographic Density in km²
home_2010	int	number of homes
avg_inc_2010	float	Monthly Average Income (in BRL)
eco_act_2012	int	Commercial Stores Quantity

Keep in mind, in Brazil the "." is the thousands separator, whereas "," is the decimals separator.

In [69]:

```
# Loading the Vitoria Regional Administration data.
vit_reg_adm = pd.read_csv('02-Datasets/vit_reg_adm.csv', sep=";");

# Print
vit_reg_adm.head()
```

Out[69]:

	id_ra	reg_adm	nbh_qty	area_m2	pop_2010	den_km2	home_2010	avg_inc_2010
0	1	Centro	8	2072047	19611	13140	6952	1425,82
1	2	Santo Antônio	12	4425494	35261	2179	10796	649,84
2	3	Jucutuquara	14	4793706	34141	6418	10580	1217,69
3	4	Maruípe	12	5684216	54402	7122	17009	806,72
4	5	Praia do Canto	9	5334352	34236	9570	12133	3844,97

In [70]:

```
# Number of Neighborhood and Variables
vit_reg_adm.shape
```

Out[70]:

```
(9, 9)
```

Neighborhood Dataset

Vitória City has 80 (eighty) neighborhoods, as you can check in the Vitória Council Website (http://legado.vitoria.es.gov.br/regionais/geral/bairros.asp), in this link you can access all neighborhood boundaries, even more in each of this `pdf` 's you can see the roads and streets and a short resume of each neighborhood. Figure A1, shows an example of information in each `pdf` .

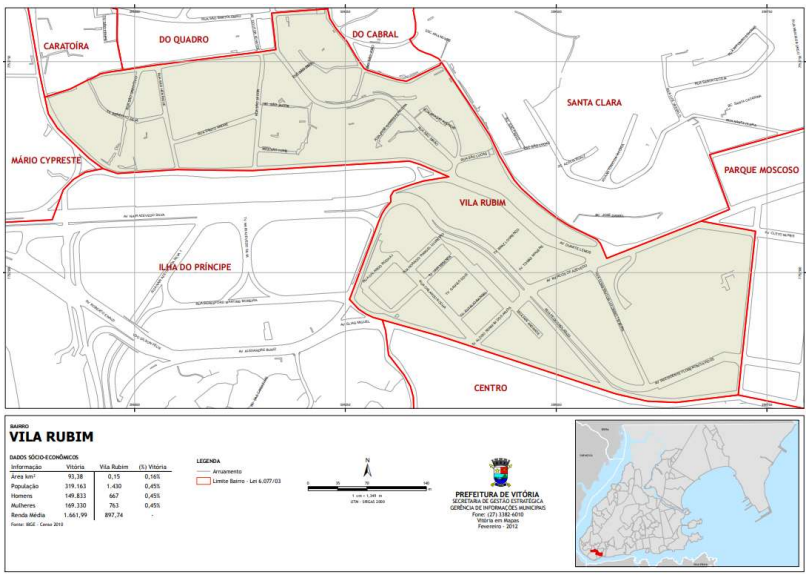


Figura A1 - Vila Rubim Boundaries and Some Descriptive Analysis on bottom left.

Based on the Table A1 Column General Informations links, and list of all neighborhoods (http://legado.vitoria.es.gov.br/regionais/geral/bairros.asp), I will create an auxiliary dataset to record additional data, so-called `vit_aux.csv` .

In this auxililary dataset I have followed the neighborhood classification of Vitória Council Website (See Table A1, each rows has own on URL, in it you can see the picture of each Regional Administration), in some cases I have found neighborhood in differents Regional Administration. The Table A3 shows the Description Types of each variable/features.

Table A3 - Description types of vit_aux.csv.

Variable	Type	Description
id_ra	int	ID of the Regional Administration (Following the sequence stablished in Table A1)
reg_adm	str	Regional Administration Name
nbh	str	Neighborhood Name
pop_2000	int	Population in Census 2000
pop_2010	int	Population in Census 2010
male	int	Quantity of male in given neighborhood
female	int	Quantity of female in given neighborhood
avg_inc_mon	float	Average Income per month (in BRL)

Keep in mind, in Brazil the "." is the thousands separator, whereas "," is the decimals separator.

In this link (http://legado.vitoria.es.gov.br/regionais/bairros/Mapa_bairros/LIMITE_BAIRROS.pdf) you can see with details a big map of Vitória and its Regional Administration Areas.

In [71]:

```
# Loading the Vitoria Auxiliary data.
vit_aux = pd.read_csv('02-Datasets/vit_aux.csv', sep=";");

# Print
vit_aux.head()
```

Out[71]:

	id_ra	reg_adm	nbh	pop_2000	pop_2010	male	female	avg_inc_mon
0	1	Centro	Centro	9240	9838	4416	5422	1791,15
1	1	Centro	Do Moscoso	854	795	369	426	647,05
2	1	Centro	Fonte Grande	1459	1231	592	639	706,03
3	1	Centro	Ilha do Príncipe	2810	2613	1194	1419	623,21
4	1	Centro	Parque Moscoso	1708	1773	791	982	1754,79

In [72]:

```
# Number of Neighborhood and Variables
vit_aux.shape
```

Out[72]:

```
(80, 8)
```

Age Dataset

I have found a table in the Vitória Council Website (click here to visit the original) (http://legado.vitoria.es.gov.br/regionais/dados_socioeconomicos/populacao/2000_2010/tab2.asp) with 20 categories (cut by 5 year each one) and by gender.

Table A4 - First rows of the vit_age.csv

age	male	female
0 to 04 years	9932	9666
05 to 09 years	10165	9727
10 to 14 years	11944	11686
15 to 19 years	12450	12932
20 to 24 years	15297	15970
25 to 29 years	15511	16621

This dataset is important to understand the population distribution by age, and how it affects the appointment.

In [73]:

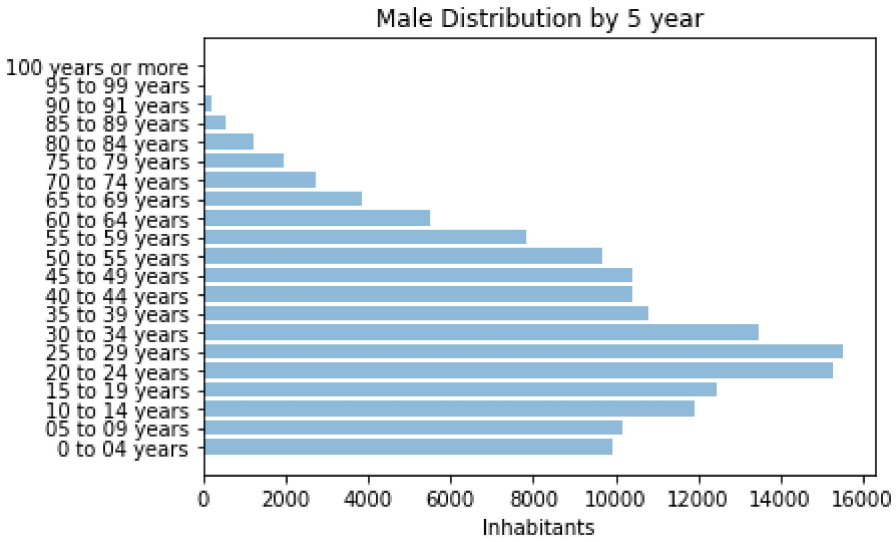
```
# Loading the Vitoria Age data.
vit_age = pd.read_csv('02-Datasets/vit_age.csv', sep=";");

# Print
vit_age.head()
```

Out[73]:

	age	male	female
0	0 to 04 years	9932	9666
1	05 to 09 years	10165	9727
2	10 to 14 years	11944	11686
3	15 to 19 years	12450	12932
4	20 to 24 years	15297	15970

In [74]:

```
y = np.arange(len(vit_age['male']))
x = vit_age['male']
objects = vit_age['age']

plt.barh(y, x, align='center', alpha=0.5)
plt.yticks(y , objects)
plt.xlabel('Inhabitants')
plt.title('Male Distribution by 5 year')

plt.show()
```



Vitória UBS Dataset

Vitória City has 38 UBS.

Table A5 - Number of USB in each Health Territory Area.

Health Territory Area	Number of Neighborhood Covered	UBS Family	US PACS	Reference Center	Specialized Center	Polyclinic	Total
Centro	11	3	2	2	1	0	8
Forte São João	20	2	2	4	0	0	8
Santo Antônio	9	3	0	0	0	0	3
Penha	14	1	4	0	0	0	5
Maruípe	18	6	0	0	0	0	6
São Pedro	10	4	0	2	1	1	8

I have not created any external document for this information.

References

Throughout the report I have inserted many links to different sources, for this reason, this chapter does not have too many links. I do not think it is necessary to insert the same link here. So, in this chapter, I only put the most important URL to this research.

- Kaggle; (https://www.kaggle.com/joniarroba/noshowappointments)
- IBGE; (https://sidra.ibge.gov.br/home/pmc/brasil)
- Vitória Council Website. (http://legado.vitoria.es.gov.br/regionais/home.asp)