Medical Appointment No Shows

Ahmed Khaled

September 6, 2023

Abstract

This analysis is part of the Udacity Data Analysis Nanodegree program and aims to explore a dataset containing aproximately 100k medical appointments from the Brazilian public health system. It is basically divided into four sections: (1) Introduction, where the investigation problem is set and the dataset is described; (2) Data Wrangling, where the acquired data are cleaned and parsed; (3) Exploratory Data Analysis, in which some intuition about the data are built based on the found patterns; (4) Conclusion, where the found insights about the problem are reviewed and communicated. Although the main techniques involved in data analysis are applied, we highlight the use of heatmaps to gain insights regarding the study topic.

1 Introduction

The Brazilian public health system, known as SUS for Unified Health System in its acronym in Portuguese, is one of the largest health system in the world[1], representing government investment of more than 9% of GDP. However, its operation is not homogeneous and there are distinct perceptions of quality from citizens in different regions of the country.

This analysis will use the no-show appointments dataset which collects information from 100k medical appointments in Brazil and is focused on the question of whether or not patients show up for their appointment, including a set of characteristics about the patient in each row:

- 'ScheduledDay': tells us on which day the patient set up their appointment.
- 'Neighborhood': indicates the location of the hospital.
- 'Scholarship': indicates whether or not the patient is enrolled in Brasilian welfare program called Bolsa Família.
- 'No-show': it says 'No' if the patient showed up to their appointment, and 'Yes' if they did not show up.

This analysis aims to outline some possible reasons for patient no-showing at the scheduled appointments, as well as get insights about the Brazilian public health system. To accomplish this, we will first try to understand the data and the context on which they were collected. We will then proceed to data wrangling and exploratory data analysis, in an iterative process, in order to draw some conclusions about the subject.

[1] Wikipedia contributors. Brazil. Wikipedia, The Free Encyclopedia. February 9, 2018, 22:06 UTC. Available at: `https://en.wikipedia.org/w/index.php?title=Brazil&oldid=824851232`. Accessed in February 10, 2018.

1.1 Import Library

This section sets up import statements for all the packages that will be used throughout this python notebook.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import numpy as np
```

1.2 Read Data

```
data = pd.read_csv('archive/KaggleV2-May-2016.csv')
print(data.head()) # Just make sure we already read it
```

```
      PatientId  AppointmentID Gender  ... Handcap SMS_received  No-show
0  2.987250e+13        5642903      F  ...       0            0       No
1  5.589978e+14        5642503      M  ...       0            0       No
2  4.262962e+12        5642549      F  ...       0            0       No
3  8.679512e+11        5642828      F  ...       0            0       No
4  8.841186e+12        5642494      F  ...       0            0       No

[5 rows x 14 columns]
```

2 Explore Data & Data Wrangling

Dataset columns contain the following informaiton

```
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 110527 entries, 0 to 110526
Data columns (total 14 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   PatientId       110527 non-null  float64
 1   AppointmentID   110527 non-null  int64
 2   Gender          110527 non-null  object
 3   ScheduledDay    110527 non-null  object
 4   AppointmentDay  110527 non-null  object
 5   Age             110527 non-null  int64
 6   Neighbourhood   110527 non-null  object
 7   Scholarship     110527 non-null  int64
 8   Hipertension    110527 non-null  int64
 9   Diabetes        110527 non-null  int64
 10  Alcoholism      110527 non-null  int64
 11  Handcap         110527 non-null  int64
 12  SMS_received    110527 non-null  int64
 13  No-show         110527 non-null  object
dtypes: float64(1), int64(8), object(5)
```

```
memory usage: 11.8+ MB
None
```

so basically 110527 row, and 14 column. All columns have their data (no missing data). To check if values are consistent we run the following.

2.1 PatientId

PatientID originally numerical (float64 type), to prevent any numerical operation we transform it to str type

```
data['PatientId'] = data['PatientId'].apply(lambda x: str(int(x)));
```

now let's get how many uniqur patients are in our dataset

```
print(len(data['PatientId'].unique()))
```

```
62299
```

Therefore some patients had more than one appointment which sound reasonable

2.2 AppointmentID

same as before ID should be str type, and we get unique ones.

```
data['AppointmentID'] = data['AppointmentID'].apply(lambda x: str(int(x)));
print(len(data['AppointmentID'].unique()))
```

```
110527
```

there is $110,527$ appointments which exactly the same rows of our dataset. Therefore we will index our dataset with this ID

```
data.set_index('AppointmentID', inplace=True) # inplace: modify the DataFrame rather than c
```

2.3 Gender

We supposed to have Male and Female gender only.

```
print(len(data['Gender'].unique()))
```

```
2
```

Seems nice[2], let's get how many males & females and the ration

```
# Note: easier way is to use `data.Gender.value_counts()`
print("Num of Male appointments = ", len(data[data['Gender'] == 'M']))
print("Num of Females appointments = ", len(data[data['Gender'] == 'F']))
```

```
Num of Male appointments =  38687
Num of Females appointments =  71840
```

[2] they are unbalanced. Since each instance represents the appointment and not the patient, this fact can be treated in a further analysis

2.4 Age

Only positive age is possible, so we check for negative one.

```
print(data[data['Age'] < 0])
```

```
                     PatientId Gender          ScheduledDay  ... Handcap  SMS_received No-s
AppointmentID                                                ...
5775010        465943158731293      F  2016-06-06T08:58:13Z  ...       0             0

[1 rows x 13 columns]
```

for lucky we got only one, so we drop it

```
data.drop('5775010', inplace=True) # Num is AppointmentId, our dataset index
```

2.5 Handcap

this attribute assumes values from 0 to 4, probably indicating the handicap number for each patient. In this analysis we gonna use bool(True for >0, False otherwise)

```
data['Handcap'] = np.where(data['Handcap'] > 0, True, False)
```

Now we can explore little about Handcap patients

```
print(data.Handcap.value_counts())
```

```
Handcap
False    108285
True       2241
Name: count, dtype: int64
```

the sum of both of them are are 110526. which is expected

2.6 Dates. Both ScheduledDay & AppointmentDay

Reviewing the information table 2 before we decide to convert it to datetime type

```
data['ScheduledDay'] = pd.to_datetime(data.ScheduledDay)
data['AppointmentDay'] = pd.to_datetime(data.AppointmentDay)
```

2.6.1 Adding Waiting days

a very relevant informaiton to add is the waiting time from AppointmentDay to Scheduled-Day. we will add it to new column. We SHOULD normalize both time FIRST as hourse doesn't matter for us

```
data['WaitingDays'] = \
    data['AppointmentDay'].dt.normalize() - \
    data['ScheduledDay'].dt.normalize()
```

Let's check the data

```
neg_waiting_days = data[data['WaitingDays'].dt.days < 0]
print(neg_waiting_days)
```

```
                    PatientId Gender              ScheduledDay  ... SMS_received  No-show
AppointmentID                                                   ...
5679978          7839272661752      M 2016-05-10 10:51:53+00:00  ...            0      Yes
5715660          7896293967868      F 2016-05-18 14:50:41+00:00  ...            0      Yes
5664962         24252258389979      F 2016-05-05 13:43:58+00:00  ...            0      Yes
5686628        998231581612122      F 2016-05-11 13:49:20+00:00  ...            0      Yes
5655637          3787481966821      M 2016-05-04 06:50:57+00:00  ...            0      Yes

[5 rows x 14 columns]
```

We found 5 rows, which will be droped. then we just extract the day value.

```
data.drop(neg_waiting_days.index, inplace=True)

data['WaitingDays'] = data.WaitingDays.dt.days
```

2.6.2 Waiting Days DataFrame

```
waitingdays = data.groupby(by=['WaitingDays','No-show'])
waitingdays = waitingdays.count()['PatientId'].unstack()
waitingdays.fillna(value=0, inplace=True)
waitingdays.reset_index(drop=False, inplace=True)
waitingdays.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 129 entries, 0 to 128
Data columns (total 3 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   WaitingDays  129 non-null    int64
 1   No           129 non-null    float64
 2   Yes          129 non-null    float64
dtypes: float64(2), int64(1)
memory usage: 3.2 KB
```

2.6.3 Category DataFrame

```
categories = pd.Series(['Same day: 0', 'Short: 1-3', 'Week: 4-7', 'Fortnight: 8-15', 'Month

waitingdays['WaitingDays'] = pd.cut(waitingdays.WaitingDays, bins = [-1,0,3,7,15,30,90,180,
data['WaitingCategories'] = pd.cut(data.WaitingDays, bins = [-1,0,3,7,15,30,90,180, 10000],

# Grouping the dataset by the waiting categories, returning the sum of all instances:
waitingdays = waitingdays.groupby('WaitingDays').sum()
## Creating a new attribute, "No-showing rate", relating how many patients did not show up
waitingdays['No-showing rate'] = (waitingdays.Yes / waitingdays.No)*100

print(waitingdays)
```

```
No-show                No     Yes  No-showing rate
WaitingDays
Same day: 0         36770.0  1792.0         4.873538
Short: 1-3          11316.0  3359.0        29.683634
Week: 4-7           13097.0  4413.0        33.694739
Fortnight: 8-15      9362.0  4166.0        44.499039
Month: 16-30        10709.0  5159.0        48.174433
Quarter: 31-90       6792.0  3369.0        49.602473
Semester: 91-180      161.0    56.0        34.782609
Very long: >180         0.0     0.0              NaN
```

2.7 Misc

little things to change before we start analysis

- Rename column names

```
data.rename(columns={'Hipertension': 'Hypertension'}, inplace=True)
```

- Re-ordering columns

```
data = data.reindex(\
    columns=['PatientId', 'Gender', 'Age', 'Scholarship',
             'Hipertension', 'Diabetes', 'Alcoholism', 'Handcap', 'ScheduledDay', 'Schedule


print(data.describe())
```

```
                 Age    Scholarship  Hipertension  ...  ScheduleTime    WaitingDays   SMS_r
count  110521.000000  110521.000000           0.0  ...           0.0  110521.000000  110521
mean       37.089386       0.098271           NaN  ...           NaN      10.184345       0
std        23.109885       0.297682           NaN  ...           NaN      15.255153       0
min         0.000000       0.000000           NaN  ...           NaN       0.000000       0
25%        18.000000       0.000000           NaN  ...           NaN       0.000000       0
50%        37.000000       0.000000           NaN  ...           NaN       4.000000       0
75%        55.000000       0.000000           NaN  ...           NaN      15.000000       1
max       115.000000       1.000000           NaN  ...           NaN     179.000000       1

[8 rows x 8 columns]
```

3 Exploratory Analysis

An advice to use some helpher function like these use some help from this stackoverflow question: python - Pandas sum across columns and divide each cell from that value - Sta...

```
def get_statistics(data, bins=20):
    '''Prints basic statistics from the input data.
    Syntax: get_statistics(data, bins=20), where:
        data = the input data series;
        bins = the number of bins to the histogram.
    '''
```

```
    total = data.values
    print('Mean:', np.mean(total))
    print('Standard deviation:', np.std(total))
    print('Minimum:', np.min(total))
    print('Maximum:', np.max(total))
    print('Median:', np.median(total))
    return plt.hist(data, bins=bins);

def get_total(dataframe):
    '''Return the total sum of each numerical attribute of a pandas.Dataframe.'''
    return dataframe.sum(axis=1)

def df_row_normalize(dataframe):
    '''
    Normalizes the values of a given pandas.Dataframe by the total sum of each line.
    '''
    return dataframe.div(dataframe.sum(axis=1), axis=0)

def df_column_normalize(dataframe, percent=False):
    '''
    Normalizes the values of a given pandas.Dataframe by the total sum of each column.
    If percent=True, multiplies the final value by 100.
    '''
    if percent:
        return dataframe.div(dataframe.sum(axis=0), axis=1)*100
    else:
        return dataframe.div(dataframe.sum(axis=0), axis=1)
```

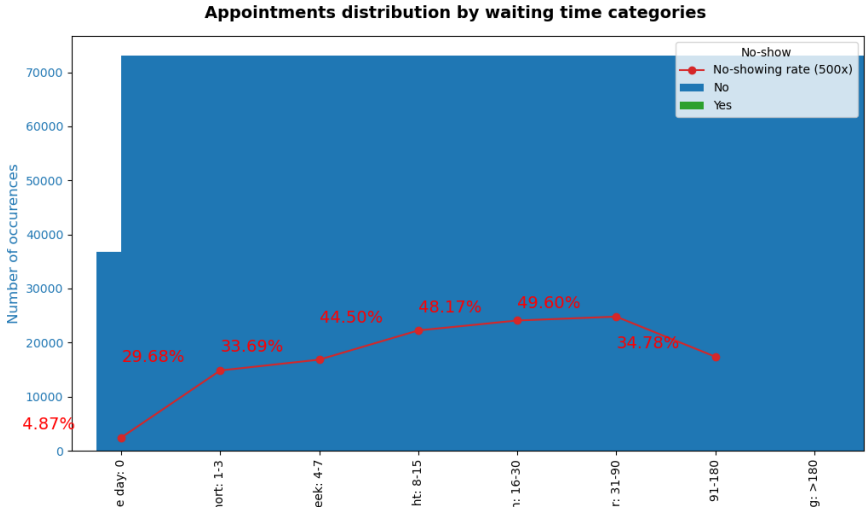
3.1 Exploring the no-showing appointments

- Questions
  - What is the average waiting time between the scheduling date and the appointment date?
  - Is there any relation between the waiting time and the no-showing appointments?
- Waiting time between the scheduling and the appointment date

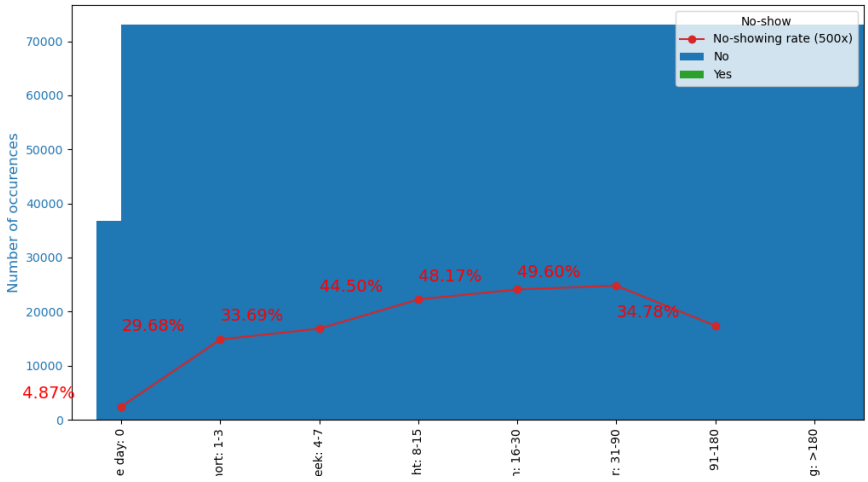
```
waitingdays_plt = get_statistics(data.WaitingDays)
```

```
Mean: 10.184345056595578
Standard deviation: 15.255084482611368
Minimum: 0
Maximum: 179
Median: 4.0
```

```
plt.savefig('waitingdays_statistics.png')
'waitingdays_statistics.png'
```



posx and posy should be finite values]]



the average is about 10days, with SD of 15, but the mean isn't centered in histogram. we investigate this issue.

1. making use of another dataframe, these data will be parsed appropriately making use of the groupBy method, which returns a pandas.groupby object with the selected attributes as index.

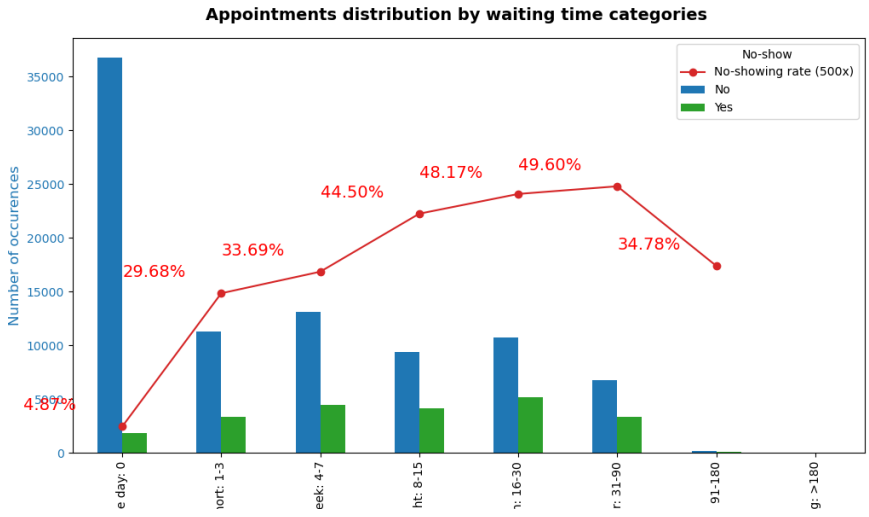
2. Since we are interested in the number of instances grouped either by 'WaitingDays' as by 'No_{show}' attributes, we will then use the count() method. To avoid redundancy we choose the 'PatientId' as reference, but it could be applied in any other attribute. We make use of the unstack() method to transform the hierarchical index as dataframe columns again.

3. Since for a given number of waiting days there is no correspondent values for No_{show}: Yes or No_{show}: No, the last operation will result in a NumPy.NaN value. However, in order to be able to plot these data, we will replace those NaN for 0. We will also reset the index, moving its values to a new dataframe column:

4. The categorized waiting days information was updated in the main dataset. However, it would be useful to parse the auxiliary dataset (waitingdays) in order to find out how the no-showing rate is distributed among the waiting categories.

```
## 1. Grouping by the 'WaitingDays' and 'No_show' values:
waitingdays = data.groupby(by=['WaitingDays','No-show'])


## 2.
waitingdays = waitingdays.count()['PatientId'].unstack()

## 3.
waitingdays.fillna(value=0, inplace=True)
waitingdays.reset_index(drop=False, inplace=True)


## Defining the categories label:
categories = pd.Series(['Same day: 0', 'Short: 1-3', 'Week: 4-7', 'Fortnight: 8-15', 'Month
## Applying these categories both to the auxiliary and to the working datasets:
waitingdays['WaitingDays'] = pd.cut(waitingdays.WaitingDays, bins = [-1,0,3,7,15,30,90,180,
data['WaitingCategories'] = pd.cut(data.WaitingDays, bins = [-1,0,3,7,15,30,90,180, 10000],

## 4.
## Grouping the dataset by the waiting categories, returning the sum of all instances:
waitingdays = waitingdays.groupby('WaitingDays').sum()
## Creating a new attribute, "No-showing rate", relating how many patients did not show up
waitingdays['No-showing rate'] = (waitingdays.Yes / waitingdays.No)*100



eda_waitingdays = waitingdays.copy()
eda_waitingdays.reset_index(drop=False, inplace=True) # to be plotted

## Adding now column
#Transforming the 'No-showing rate' into strings with the percentual values:
eda_waitingdays['No-show percentual'] = eda_waitingdays['No-showing rate'].apply(lambda x:

#Multiplying the rate values by 500 times in order to be plotted in the same scale:
eda_waitingdays['No-showing rate (500x)'] = eda_waitingdays['No-showing rate']*500

print(eda_waitingdays)

posx and posy should be finite values
No-show       WaitingDays       No  ...  No-show percentual  No-showing rate (500x)
0             Same day: 0  36770.0  ...               4.87%             2436.769105
1              Short: 1-3  11316.0  ...              29.68%            14841.816896
2              Week: 4-7   13097.0  ...              33.69%            16847.369627
3         Fortnight: 8-15   9362.0  ...              44.50%            22249.519333
4            Month: 16-30  10709.0  ...              48.17%            24087.216360
5          Quarter: 31-90   6792.0  ...              49.60%            24801.236749
6        Semester: 91-180    161.0  ...              34.78%            17391.304348
7         Very long: >180      0.0  ...                nan%                    NaN
```

```
[8 rows x 6 columns]

## Setting the graph parameters:
fig1, ax = plt.subplots(figsize=[12,6])  #Defines the graph window size
fig1.subplots_adjust(top=0.92)
plt.suptitle('Appointments distribution by waiting time categories', fontsize=14, fontweigh

colors = ['tab:blue', 'tab:green', 'tab:red']  #Defines the colors to be used

ax.set_ylabel('Number of occurences', color=colors[0], fontsize=12)  #Set the y-axis color
ax.tick_params(axis='y', labelcolor=colors[0])

## Plotting the line chart:
eda_waitingdays[['WaitingDays', 'No-showing rate (500x)']].plot(x='WaitingDays', linestyle=
#Setting the line chart marker labels
x = ax.get_xticks()  #Getting the x-axis ticks to plot the label
for a,b,c in zip(x,eda_waitingdays['No-showing rate (500x)'], eda_waitingdays['No-show perc
    plt.text(a,b+1500,c, color='red', fontsize=14)

## Plotting the bar chart:
eda_waitingdays[['WaitingDays', 'No', 'Yes']].plot(x='WaitingDays', kind='bar', ax=ax, colo

ax.set_xlabel('Waiting time categories', fontsize=12)  #Set the y-axis color and label

# org-mode show plot
plt.savefig('appointment_distribution.png')
'appointment_distribution.png'
```



Through the chart above, **it becomes evident that the no-showing rate increases as the waiting** It reaches the lower rates when the attendance occurs in the same day it was scheduled.