Project: Investigate a Dataset(no-show appointments)

1 Contents

2 Summary

I have chosen this dataset for no-show appointments 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. One remark in the description of this dataset is the encoding of the last column: it says ‘No’ if the patient showed up to their appointment, and ‘Yes’ if they did not. This dataset’s major records were taken in May and some others in March and April. The dataset is a good example of data cleaning and data analysis.

The data has 14 columns:

Column	**Records**	**Is Null**	**Type**
PatientId	110527		float64
AppointmentID	110527		int64
Gender	110527		object
ScheduledDay	110527		object
AppointmentDay	110527		object
Age	110527		int64
Neighbourhood	110527		object
Scholarship	110527		int64
Hipertension	110527		int64
Diabetes	110527		int64
Alcoholism	110527		int64
Handcap	110527		int64
SMS_received	110527		int64
No-show	110527		object
types: float64(1), int64(8), object(5)			

From this Dataset we can ask the following questions:

- What is the relation between the date and showing for the appointment?
- What are the percentage records in each month?
- What is the relation between the age and showing for the appointment?
- How age is distributed in the dataset?
- Is showing for the appointment related to some time in the day rather than another?
- What is the percentage of scholarship holders in this dataset?
- What is the relation between scholarship and attendance?

Notebook and report objectives:

- Practice data cleaning techniques
- Practice EDA technique
- Practice python libraries related to EDA

3 Importing necessary libraries

```
import numpy as np
import pandas as pd

import seaborn as sns

import plotly.express as px
import plotly.graph_objects as go

import holoviews as hv
from holoviews import opts
hv.extension('bokeh')

#to plot within notebook
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('fivethirtyeight')
```

4 Loading Dataset

Since I worked with Google coolab I needed to upload the dataset from the local machine

```
from google.colab import files

uploaded = files.upload()

for fn in uploaded.keys():
  print('User uploaded file "{name}" with length {length} bytes'.format(
      name=fn, length=len(uploaded[fn])))
```

The next image shows how the data file is red. However, on_bad_lines='skip' was added just because sometimes when uploading the file some data gets corrupted because of the low internet speed.

Reading file

+ Code + Text

```
filepath = "noshowappointments-kagglev2-may-2016.csv"
df = pd.read_csv(filepath, on_bad_lines='skip')
df.head(3)
```

	PatientId	AppointmentID	Gender	ScheduledDay	AppointmentDay	Age	Neighbourhood	Scholarship	Hipertension	Diabetes	Alcoholism	Handcap	SMS_received	No-show
0	2.987250e+13	5642903	F	2016-04-29T18:38:08Z	2016-04-29T00:00:00Z	62	JARDIM DA PENHA	0	1	0	0	0	0	No
1	5.589978e+14	5642503	M	2016-04-29T16:08:27Z	2016-04-29T00:00:00Z	56	JARDIM DA PENHA	0	0	0	0	0	0	No
2	4.262962e+12	5642549	F	2016-04-29T16:19:04Z	2016-04-29T00:00:00Z	62	MATA DA PRAIA	0	0	0	0	0	0	No

5 Data Cleaning

5.1 Data Shape & info

First of all, we see data shape the data description

```
df.shape # Rows & Columns
```

```
(110527, 14)
```

```
df.describe() # Data Describtion
```

	PatientId	AppointmentID	Age	Scholarship	Hipertension	Diabetes	Alcoholism	Handcap	SMS_received
count	1.105270e+05	1.105270e+05	110527.000000	110527.000000	110527.000000	110527.000000	110527.000000	110527.000000	110527.000000
mean	1.474963e+14	5.675305e+06	37.088874	0.098266	0.197246	0.071865	0.030400	0.022248	0.321026
std	2.560949e+14	7.129575e+04	23.110205	0.297675	0.397921	0.258265	0.171686	0.161543	0.466873
min	3.921784e+04	5.030230e+06	-1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	4.172614e+12	5.640286e+06	18.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	3.173184e+13	5.680573e+06	37.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	9.439172e+13	5.725524e+06	55.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000
max	9.999816e+14	5.790484e+06	115.000000	1.000000	1.000000	1.000000	1.000000	4.000000	1.000000

And here we see data info where there are 0 null values and data types as mentioned in the summary

```
df.info() # checking types and null values
```

```
<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
```

5.2 Checking data redundancy and uniqueness

I created a function to check the unique values in the dataset since our id or index must be unique

```python
def get_data_summary(data):
  '''
  Calculating unique value count
  '''

  data_unq_val_count = pd.DataFrame(data.nunique(), columns=['unq_val_count'])
  data_unq_val_count.reset_index(inplace=True)
  data_unq_val_count.rename(columns = {'index':'variable'}, inplace = True)
  data_unq_val_count = data_unq_val_count.merge(
      data.dtypes.reset_index().rename(columns = {'index': 'variable', 0:'dtype'}),
      on= 'variable'
                                              )
  data_unq_val_count = data_unq_val_count.sort_values(by= 'unq_val_count', ascending= False)

  return data_unq_val_count
```

```python
print(get_data_summary(df))
```

```
          variable  unq_val_count    dtype
1    AppointmentID         110527    int64
3     ScheduledDay         103549   object
0        PatientId          62299  float64
5              Age            104    int64
6    Neighbourhood             81   object
4   AppointmentDay             27   object
11         Handcap              5    int64
2           Gender              2   object
7      Scholarship              2    int64
8     Hipertension              2    int64
9         Diabetes              2    int64
10      Alcoholism              2    int64
12    SMS_received              2    int64
13         No-show              2   object
```

Then I checked duplicated data

```python
df[df.duplicated()]
```

PatientId	AppointmentID	Gender	ScheduledDay	AppointmentDay	Age	Neighbourhood	Scholarship	Hipertension	Diabetes	Alcoholism	Handcap	SMS_received	No-show

As a summary of the previous steps, we can conclude multiple issues starting with dates columns, ScheduledDay changes from date time to date and time adding some other features, and dropping the appointment date since when don't need it in further operations. Also The column name No-show isn't clear we can change the name to attended and correct the data and make it binary (0,1). Another operation we can do is drop the PatientID and AppointmentID since we don't need them later.

5.3 Dropping columns

```
df.drop(['AppointmentID', 'PatientId', 'AppointmentDay'], axis = 1, inplace = True)
df.head()
```

	Gender	ScheduledDay	Age	Neighbourhood	Scholarship	Hipertension	Diabetes	Alcoholism	Handcap	SMS_received	No-show
0	F	2016-04-29T18:38:08Z	62	JARDIM DA PENHA	0	1	0	0	0	0	No
1	M	2016-04-29T16:08:27Z	56	JARDIM DA PENHA	0	0	0	0	0	0	No
2	F	2016-04-29T16:19:04Z	62	MATA DA PRAIA	0	0	0	0	0	0	No
3	F	2016-04-29T17:29:31Z	8	PONTAL DE CAMBURI	0	0	0	0	0	0	No
4	F	2016-04-29T16:07:23Z	56	JARDIM DA PENHA	0	1	1	0	0	0	No

5.4 Features extraction

Datetime treatment, schedule day column has so much info we can get the season day part-time and so many as shown in next steps. The first thing we checked was if there is any null values in the column. Then we converted the column value to a Datetime type then extracted all the shown values from it.

eving the scheduled date

```
df['date'] = pd.to_datetime(df['ScheduledDay'], errors='coerce')
print (df[df.date.isnull()])
```

```
Empty DataFrame
Columns: [Gender, ScheduledDay, Age, Neighbourhood, Scholarship, Hipertension, Diabetes, Alcoholism, Handcap, SMS_received, No-show, date]
Index: []
```

```
df['date'] = pd.to_datetime(df['ScheduledDay'], format='%Y-%m-%d %H:%M:%S')
df['year'] = df['date'].apply(lambda x : x.year)
df['month'] = df['date'].apply(lambda x : x.month)
df['day'] = df['date'].apply(lambda x : x.day)
df['weekday'] = df['date'].apply(lambda x : x.day_name())
df['weekofyear'] = df['date'].apply(lambda x : x.weekofyear)
df['hour'] = df['date'].apply(lambda x : x.hour)
df['minute'] = df['date'].apply(lambda x : x.minute)
df.head(3)
```

Dropped the old column since we retrieved all features from it

```
df.drop(['ScheduledDay'], axis = 1, inplace = True)
df.head()
```

	Gender	Age	Neighbourhood	Scholarship	Hipertension	Diabetes	Alcoholism	Handcap	SMS_received	No-show	date	year	month	day	weekday	weekofyear	hour	minute
0	F	62	JARDIM DA PENHA	0	1	0	0	0	0	No	2016-04-29 18:38:08+00:00	2016	4	29	Friday	17	18	38
1	M	56	JARDIM DA PENHA	0	0	0	0	0	0	No	2016-04-29 16:08:27+00:00	2016	4	29	Friday	17	16	8
2	F	62	MATA DA PRAIA	0	0	0	0	0	0	No	2016-04-29 16:19:04+00:00	2016	4	29	Friday	17	16	19
3	F	8	PONTAL DE CAMBURI	0	0	0	0	0	0	No	2016-04-29 17:29:31+00:00	2016	4	29	Friday	17	17	29
4	F	56	JARDIM DA PENHA	0	1	1	0	0	0	No	2016-04-29 16:07:23+00:00	2016	4	29	Friday	17	16	7

More info we can get from the date, seasons, a. Winter (December, January, and February). b. Spring (March, April, May). c. Summer (June to September). d. Autumn period (October to November).

Creating Seasons

```python
def define_seasons(month_val):
    if month_val in [12, 1, 2]:
        season_val = 'Winter'
    elif month_val in [3, 4, 5]:
        season_val = 'Spring'
    elif month_val in [6, 7, 8, 9]:
        season_val = 'Summer'
    elif month_val in [10, 11]:
        season_val = 'Autumn'

    return season_val
```

Further info we can find in the date column is to get the day part (night, morning, afternoon, noon)

Creating day parts

```python
def hours_to_day_time(hour):
    if hour in [22,23,0,1,2,3]:
        timing = 'Night'
    elif hour in range(4, 12):
        timing = 'Morning'
    elif hour in range(12, 17):
        timing = 'Afternoon'
    elif hour in range(17, 22):
        timing = 'Evening'
    else:
        timing = 'NaN'
    return timing
```

```python
[114] df['day_part'] = df['hour'].apply(hours_to_day_time)
      df.head()
```

	Gender	Age	Neighbourhood	Scholarship	Hipertension	Diabetes	Alcoholism	Handcap	SMS_received	No-show	...	year	month	day	weekday	weekofyear	hour	minute	season	month_name	day_part
0	F	62	JARDIM DA PENHA	0	1	0	0	0	0	No	...	2016	4	29	Friday	17	18	38	Spring	April	Evening
1	M	56	JARDIM DA PENHA	0	0	0	0	0	0	No	...	2016	4	29	Friday	17	16	8	Spring	April	Afternoon
2	F	62	MATA DA PRAIA	0	0	0	0	0	0	No	...	2016	4	29	Friday	17	16	19	Spring	April	Afternoon
3	F	8	PONTAL DE CAMBURI	0	0	0	0	0	0	No	...	2016	4	29	Friday	17	17	29	Spring	April	Evening
4	F	56	JARDIM DA PENHA	0	1	1	0	0	0	No	...	2016	4	29	Friday	17	16	7	Spring	April	Afternoon

The last thing is changing the No-show column name and values to binary.

Changing No-Show to attandance and its values to binary

```python
[115] df.rename(columns={'No-show': 'Attendance'}, inplace=True)
      df.head()
```

	Gender	Age	Neighbourhood	Scholarship	Hipertension	Diabetes	Alcoholism	Handcap	SMS_received	Attendance	...	year	month	day	weekday	weekofyear	hour	minute	season	month_name	day_part
0	F	62	JARDIM DA PENHA	0	1	0	0	0	0	No	...	2016	4	29	Friday	17	18	38	Spring	April	Evening
1	M	56	JARDIM DA PENHA	0	0	0	0	0	0	No	...	2016	4	29	Friday	17	16	8	Spring	April	Afternoon
2	F	62	MATA DA PRAIA	0	0	0	0	0	0	No	...	2016	4	29	Friday	17	16	19	Spring	April	Afternoon
3	F	8	PONTAL DE CAMBURI	0	0	0	0	0	0	No	...	2016	4	29	Friday	17	17	29	Spring	April	Evening
4	F	56	JARDIM DA PENHA	0	1	1	0	0	0	No	...	2016	4	29	Friday	17	16	7	Spring	April	Afternoon

5 rows × 21 columns

```python
[116] def status_to_binary(status):
          if status == 'No':
              val = 1
          elif status == 'Yes':
              val = 0

          return val
```

```python
[117] df['Attendance'] = df['Attendance'].apply(status_to_binary)
      df.head()
```

	Gender	Age	Neighbourhood	Scholarship	Hipertension	Diabetes	Alcoholism	Handcap	SMS_received	Attendance	...	year	month	day	weekday	weekofyear	hour	minute	season	month_name	day_part
0	F	62	JARDIM DA PENHA	0	1	0	0	0	0	1	...	2016	4	29	Friday	17	18	38	Spring	April	Evening
1	M	56	JARDIM DA PENHA	0	0	0	0	0	0	1	...	2016	4	29	Friday	17	16	8	Spring	April	Afternoon
2	F	62	MATA DA PRAIA	0	0	0	0	0	0	1	...	2016	4	29	Friday	17	16	19	Spring	April	Afternoon
3	F	8	PONTAL DE CAMBURI	0	0	0	0	0	0	1	...	2016	4	29	Friday	17	17	29	Spring	April	Evening
4	F	56	JARDIM DA PENHA	0	1	1	0	0	0	1	...	2016	4	29	Friday	17	16	7	Spring	April	Afternoon

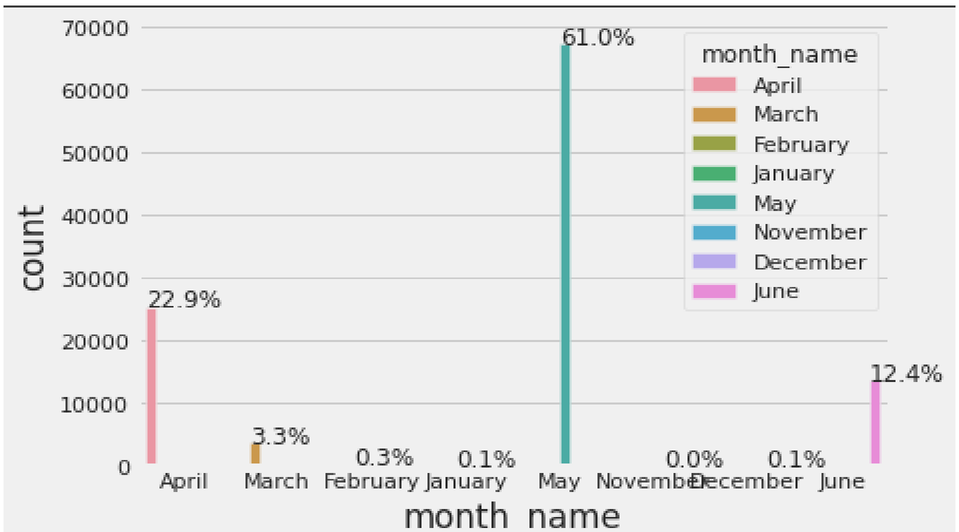
6 EDA

6.1 Univariate Analysis

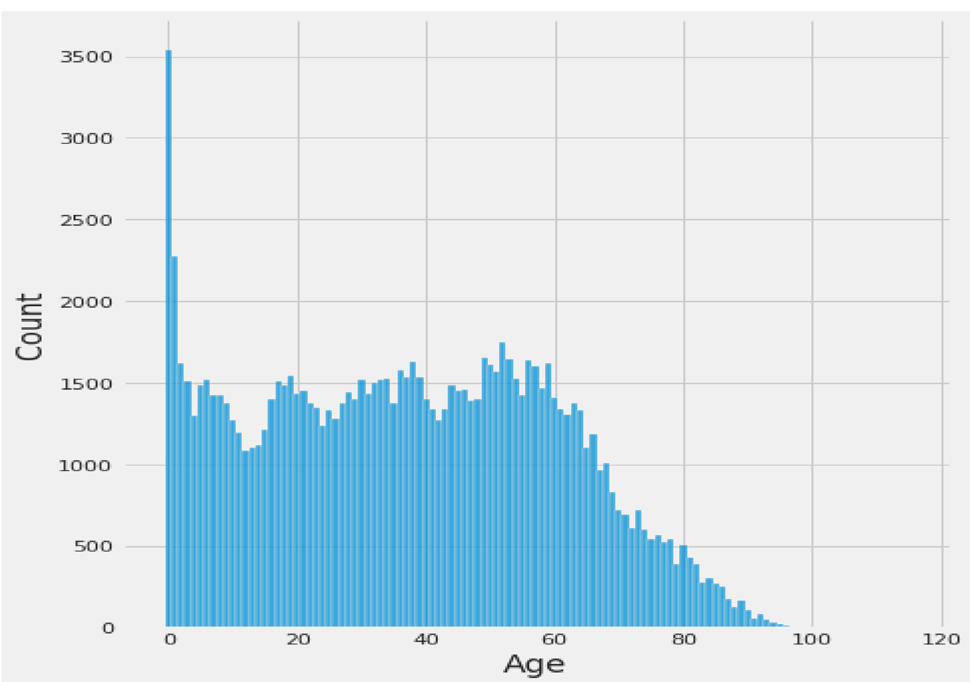
Here we will try to answer the question related to one variable such as

- What are the percentage records in each month?
- How age is distributed in the dataset?
- What is the percentage of scholarship holders in this dataset?

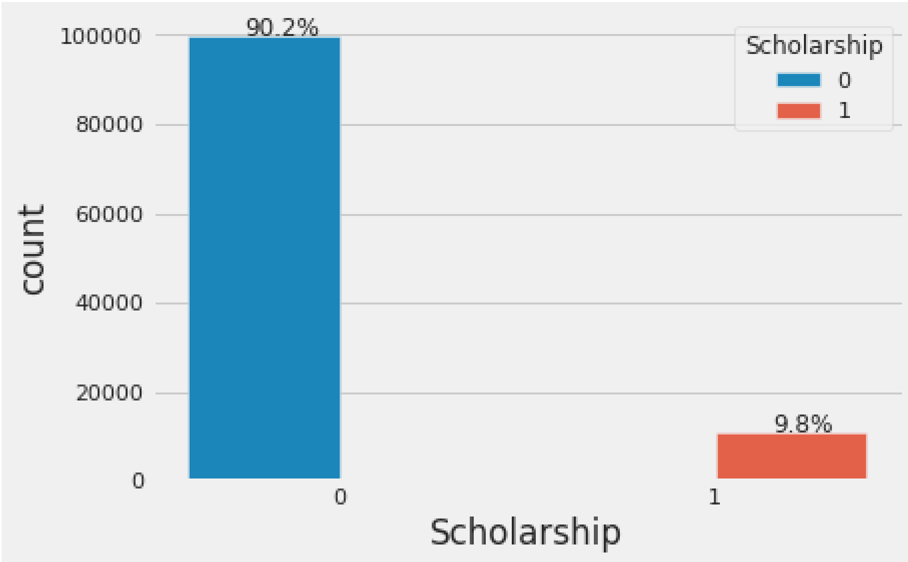
Months record percentage



Age Distribution



Scholarship holders

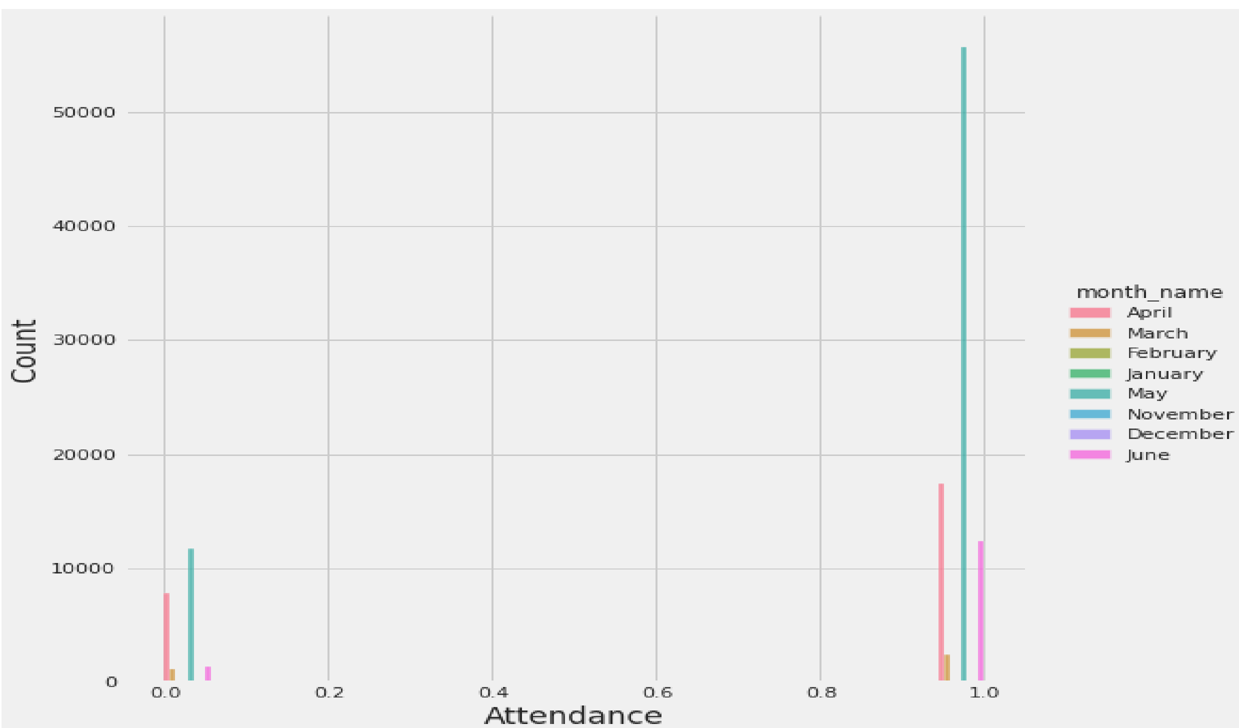


6.2 Multivariate Analysis

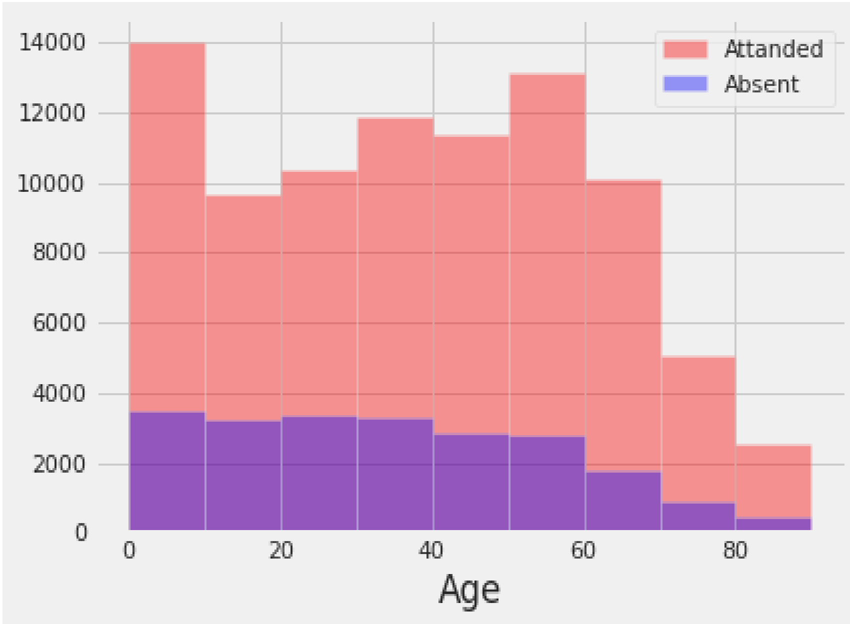
Here we will try to answer the questions related to multi variables such as

- What is the relation between the date and showing for the appointment?
- What is the relation between the age and showing for the appointment?
- Is showing for the appointment related to sometime in the day rather than another?
- What is the relation between scholarship and attendance?

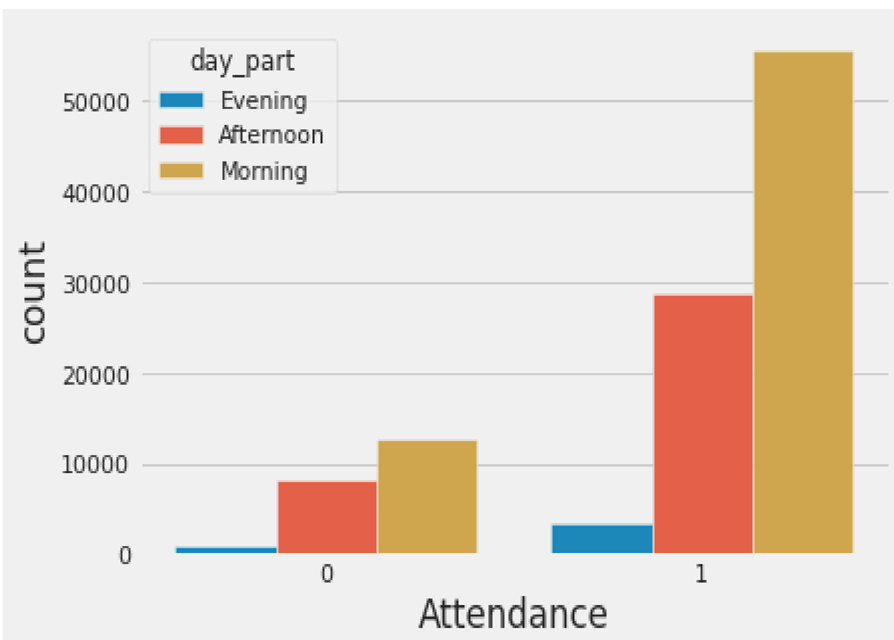
Date and show for the appointment relation



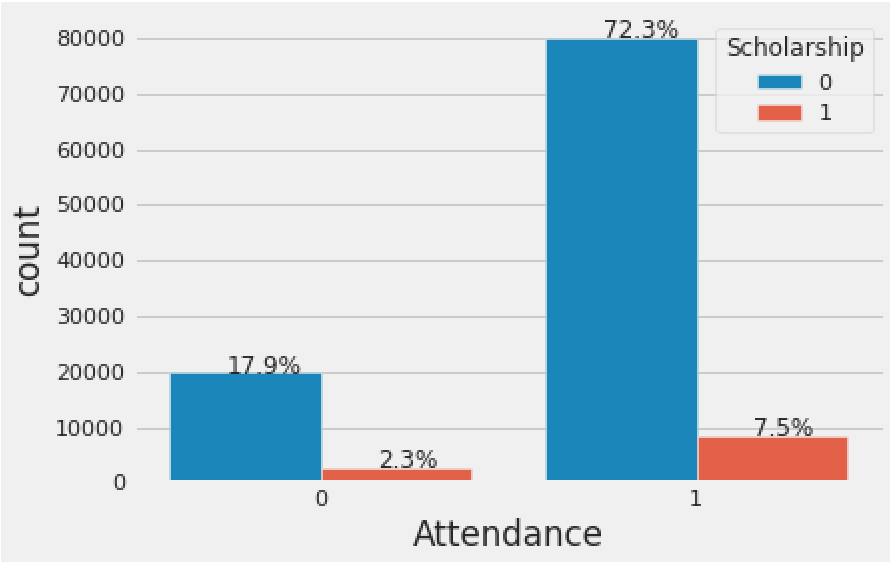
Age and showing for the appointment relation



Day part-time and showing for an appointment



Scholarship holder and showing for an appointment



6.4 Conclusion of the section:

This data even seems somehow organized and can get some valuable data from it. But there are so many questions that can be asked concerning diabetes and their discipline, the age of diabetes, men or women who are more discipline… So, we can conclude that the data has various uses and can lead to multiple results. In this report, we tried to benefit from applying what learned in data cleaning and EDA.

7 Suggestions for the next steps

The next for this data analysis is to study and plot the relationship between diabetes and appointment, gender and appointment, diabetes and gender, … and so on. No matter how many examples I give there will be another relationship between this dataset features that can be formed. However, this dataset will be a great asset to machine learning training.

Name: Abid Mohamed Nadhir