No_Show_Appointments

March 29, 2021

1 Project: Investigate a Dataset (No Show Medical Appointments in Brazil)

1.1 Table of Contents

Introduction

In this project, I have analysed "No show appointments" dataset, which is available on Kaggle. This dataset collects information from 100k medical appointments in Brazil.

The dataset includes, - Patient details - Patient ID, Gender, Age, Neighbourhood, Scholarship, Whether the patient has Hipertension, Diabetes and Whether the patient is Alcoholic and Handicap - Appointment details - Appointment ID, Appointment Scheduled Day, Appointment Day, SMS_received and No-show (whether the patient showed up or not)

Note: Findings are tentative as inferential statistics or machine learning are not used to complete this project

Questions focused to be answered,

Patients with which disease schedules appoinments and doesn't shows up the most?

How does the number of SMS received affects the show-up for scheduled appoinments?

How does Gender affects the No Show occurence?

How does Scholarship affects the No Show occurence?

How does Age affects the No Show rate?

Does the gap between scheduled date and appointment date affect patient's no-show rate?

Does the Day of the week affect patient no-show probability?

What factors are important for us to know in order to predict if a patient will show up for their scheduled appointment?

[1]:
```
# Import required packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import requests
import datetime as dt

# To plot visualizations inline with the notebook
%matplotlib inline
```

Data Wrangling

Data Gathering > - Download the csv file (“no_show_appointments”) from Kaggle programmatically > - Load the downloaded file into DataFrame

[2]:
```
# Request csv file from the url
url = "https://d17h27t6h515a5.cloudfront.net/topher/2017/October/
 ↪59dd2e9a_noshowappointments-kagglev2-may-2016/
 ↪noshowappointments-kagglev2-may-2016.csv"
response = requests.get(url)

# Write the response into csv file
with open('no_show_appointments.csv', mode='wb') as file:
    file.write(response.content)

# Load data from the file into DataFrame and print out a few lines.
df = pd.read_csv('no_show_appointments.csv')
df.head()
```

[2]:
```
      PatientId  AppointmentID Gender          ScheduledDay  \
0  2.987250e+13        5642903      F  2016-04-29T18:38:08Z
1  5.589978e+14        5642503      M  2016-04-29T16:08:27Z
2  4.262962e+12        5642549      F  2016-04-29T16:19:04Z
3  8.679512e+11        5642828      F  2016-04-29T17:29:31Z
4  8.841186e+12        5642494      F  2016-04-29T16:07:23Z

         AppointmentDay  Age      Neighbourhood  Scholarship  Hipertension  \
0  2016-04-29T00:00:00Z   62    JARDIM DA PENHA            0             1
1  2016-04-29T00:00:00Z   56    JARDIM DA PENHA            0             0
2  2016-04-29T00:00:00Z   62      MATA DA PRAIA            0             0
3  2016-04-29T00:00:00Z    8  PONTAL DE CAMBURI            0             0
4  2016-04-29T00:00:00Z   56    JARDIM DA PENHA            0             1

   Diabetes  Alcoholism  Handcap  SMS_received No-show
```

0	0	0	0	0	No
1	0	0	0	0	No
2	0	0	0	0	No
3	0	0	0	0	No
4	1	0	0	0	No

Data Assessing > - Number of samples and features > - Perform operations to inspect data types > - Look for instances of missing or possibly errant data > - Look for instances of duplicates > - Perform operations to inspect number of uniques values in each column > - Perform operations to inspect uniques values in each column

[3]:
```python
# Number of samples and features
print('Number of samples :', df.shape[0])
print('Number of features :', df.shape[1])
```

```
Number of samples : 110527
Number of features : 14
```

[4]:
```python
# Perform operations to inspect data types
df.dtypes
```

[4]:
```
PatientId          float64
AppointmentID        int64
Gender              object
ScheduledDay        object
AppointmentDay      object
Age                  int64
Neighbourhood       object
Scholarship          int64
Hipertension         int64
Diabetes             int64
Alcoholism           int64
Handcap              int64
SMS_received         int64
No-show             object
dtype: object
```

- checking the datatype of values which are of object in above step

[5]:
```python
print('Gender :', type(df.Gender[0]))
print('ScheduledDay :', type(df.ScheduledDay[0]))
print('AppointmentDay :', type(df.AppointmentDay[0]))
print('Neighbourhood :', type(df.Neighbourhood[0]))
```

```
Gender : <class 'str'>
ScheduledDay : <class 'str'>
AppointmentDay : <class 'str'>
Neighbourhood : <class 'str'>
```

Note : - Wrong column naming conventions for "No-show" column name - 'ScheduledDay' & 'AppointmentDay' is of type string - 'Handicap' mispelled as 'Handcap' - Lowercase columns names would be more easier during analysis

[6]:
```
# look for instances of missing or possibly errant data
df.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
```

[7]:
```
# Look for instances of duplicates
df.duplicated().value_counts()
```

[7]:
```
False    110527
dtype: int64
```

[8]:
```
# Perform operations to inspect number of uniques values in each column
df.nunique()
```

[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
```

Checking the unique values in each column, > - As per the dataset definition in Kaggle, Handcap must have values either True/False. But in the assessed DataFrame, 5 unique values are found. > - And also, SMS_received can have more than 1 unique values. > - So, checking the unique values of each column for data consistency > - **Avoided columns :** PatientId & AppointmentID as it will be not used in analysis

[9]:
```python
# Unique values in Handcap
df.Handcap.unique()
```

[9]:
```
array([0, 1, 2, 3, 4], dtype=int64)
```

[10]:
```python
# Unique values in SMS_received
df.SMS_received.unique()
```

[10]:
```
array([0, 1], dtype=int64)
```

> **Note :** It is clear that the Handcap & SMS_received column values may have swapped, as per the dataset definition in Kaggle

[11]:
```python
# Unique values in Gender
df.Gender.unique()
```

[11]:
```
array(['F', 'M'], dtype=object)
```

[12]:
```python
# Unique values in Age
df.Age.unique()
```

[12]:
```
array([ 62,  56,   8,  76,  23,  39,  21,  19,  30,  29,  22,  28,  54,
        15,  50,  40,  46,   4,  13,  65,  45,  51,  32,  12,  61,  38,
        79,  18,  63,  64,  85,  59,  55,  71,  49,  78,  31,  58,  27,
         6,   2,  11,   7,   0,   3,   1,  69,  68,  60,  67,  36,  10,
        35,  20,  26,  34,  33,  16,  42,   5,  47,  17,  41,  44,  37,
        24,  66,  77,  81,  70,  53,  75,  73,  52,  74,  43,  89,  57,
        14,   9,  48,  83,  72,  25,  80,  87,  88,  84,  82,  90,  94,
        86,  91,  98,  92,  96,  93,  95,  97, 102, 115, 100,  99,  -1],
      dtype=int64)
```

> **Note :** Calculating the proportion of Age (102,115,100) in order to decide whether it is normal

[13]:
```python
# Number of patients is of Age (102,115,100)
count = df.query('Age in [102,115,100]').Age.count()
count
```

[13]: 11

[14]:
```python
# Total samples
total_samples = df.shape[0]
total_samples
```

[14]: 110527

[15]:
```python
# Proportion of Age (102,115,100)
proportion = count/total_samples
proportion
```

[15]: 9.952319342785021e-05

Note : Age must be greater than 0 - (0,-1) inaccurate data - Considering the Ages (102,115,100) as its proportion is too low and we know some humans live after 100 years old.

[16]:
```python
# Unique values in Scholarship
df.Scholarship.unique()
```

[16]: array([0, 1], dtype=int64)

[17]:
```python
# Unique values in Hipertension
df.Hipertension.unique()
```

[17]: array([1, 0], dtype=int64)

[18]:
```python
# Unique values in Diabetes
df.Diabetes.unique()
```

[18]: array([0, 1], dtype=int64)

[19]:
```python
# Unique values in Alcoholism
df.Alcoholism.unique()
```

[19]: array([0, 1], dtype=int64)

[20]:
```python
# Unique values in ScheduledDay
df.ScheduledDay.unique()
```

[20]:
```
array(['2016-04-29T18:38:08Z', '2016-04-29T16:08:27Z',
       '2016-04-29T16:19:04Z', …, '2016-04-27T16:03:52Z',
       '2016-04-27T15:09:23Z', '2016-04-27T13:30:56Z'], dtype=object)
```

[21]:
```python
# Unique values in AppointmentDay
df.AppointmentDay.unique()
```

[21]:
```
array(['2016-04-29T00:00:00Z', '2016-05-03T00:00:00Z',
       '2016-05-10T00:00:00Z', '2016-05-17T00:00:00Z',
       '2016-05-24T00:00:00Z', '2016-05-31T00:00:00Z',
       '2016-05-02T00:00:00Z', '2016-05-30T00:00:00Z',
       '2016-05-16T00:00:00Z', '2016-05-04T00:00:00Z',
       '2016-05-19T00:00:00Z', '2016-05-12T00:00:00Z',
       '2016-05-06T00:00:00Z', '2016-05-20T00:00:00Z',
       '2016-05-05T00:00:00Z', '2016-05-13T00:00:00Z',
       '2016-05-09T00:00:00Z', '2016-05-25T00:00:00Z',
       '2016-05-11T00:00:00Z', '2016-05-18T00:00:00Z',
       '2016-05-14T00:00:00Z', '2016-06-02T00:00:00Z',
       '2016-06-03T00:00:00Z', '2016-06-06T00:00:00Z',
       '2016-06-07T00:00:00Z', '2016-06-01T00:00:00Z',
       '2016-06-08T00:00:00Z'], dtype=object)
```

Observations based on Assessing the dataset : > **Columns :** > - Unrequired columns, PatientId & AppointmentID > - Wrong column naming conventions for "No-show" column name > - 'Handicap' mispelled as 'Handcap' > - Lowercase columns names would be more easier during analysis

Datatype :

```
> - 'ScheduledDay' & 'AppointmentDay' is of type string, should be changed to DateTime
```

Data :

```
> - It is clear that the Handcap & SMS_received column values may have swapped, as per the data
> - Inaccurate data in Age (0,-1)
> - Because of wrong column name conventions, couldn't assess the unique values in 'no_show'. I
```

Data Cleaning

Columns :

```
> - Drop Unrequired columns, 'PatientId' & 'AppointmentID'
> - Change the column name of <i>'No-show' - 'no_show'</i> and <i>'Handcap' - 'handicap'</i>
> - Change columns names to Lowercase & Insert _ before day in ScheduledDay & AppointmentDay
```

Datatype :

```
> - Change datatype of 'ScheduledDay' & 'AppointmentDay', from <i>string</i> to <i>DateTime</i>
```

Data :

```
> - Swap the values among 'handicap' & 'sms_received' columns
> - Fill in inaccurate data in 'age' (0,-1) with age mean
> - Change the value in 'no_show' column 'No'-1 & 'Yes'-0 of integer type, as it is in encrypte
```

Feature Engineering :

```
> - Create a column named 'awaiting_days' for easier analysis, which is the difference between
```

1.1.1 Issue 1

Define :

Drop unrequired columns, 'PatientId' & 'AppointmentID', as it will be not used in analysis

Clean :

[22]:
```
# Check columns before drop
print('Number of columns before drop :', df.shape[1])
df.columns
```

```
Number of columns before drop : 14
```

[22]:
```
Index(['PatientId', 'AppointmentID', 'Gender', 'ScheduledDay',
       'AppointmentDay', 'Age', 'Neighbourhood', 'Scholarship', 'Hipertension',
       'Diabetes', 'Alcoholism', 'Handcap', 'SMS_received', 'No-show'],
      dtype='object')
```

[23]:
```
# Drop the columns from DataFrame
df.drop(['PatientId','AppointmentID'], axis=1, inplace=True)
```

Test :

[24]:
```
# Check whether the columns are dropped from DataFrame
df.columns.all() in ['PatientId', 'AppointmentID']  # should return false
```

[24]:
```
False
```

[25]:
```
# Columns after drop
print('Number of columns after drop :', df.shape[1])
df.columns
```

```
Number of columns after drop : 12
```

[25]:
```
Index(['Gender', 'ScheduledDay', 'AppointmentDay', 'Age', 'Neighbourhood',
       'Scholarship', 'Hipertension', 'Diabetes', 'Alcoholism', 'Handcap',
       'SMS_received', 'No-show'],
      dtype='object')
```

1.1.2 Issue 2

Define :

Change the column name of 'No-show' - 'no_show' (Wrong naming convention) and 'Handcap' - 'hanc

Clean :

[26]:
```
# Renaming column names
df.rename(columns = {
    'No-show' : 'no_show',
    'Handcap' : 'handicap'
}, inplace=True);
```

Test :

[27]:
```
# Check whether the old name is not present
df.columns.all() in ['No-show', 'Handcap']  # should return false
```

[27]: False

[28]:
```
# columns names
df.columns
```

[28]:
```
Index(['Gender', 'ScheduledDay', 'AppointmentDay', 'Age', 'Neighbourhood',
       'Scholarship', 'Hipertension', 'Diabetes', 'Alcoholism', 'handicap',
       'SMS_received', 'no_show'],
      dtype='object')
```

[29]:
```
# Check for the unique data, as we couldn't earlier (Because of naming␣
↪conventions)
df.no_show.unique()
```

[29]:
```
array(['No', 'Yes'], dtype=object)
```

1.1.3 Issue 3

Define :

- Change columns names to Lowercase
- Insert _ before day in ScheduledDay & AppointmentDay

Clean :

[30]:
```
# Column names
df.columns
```

[30]:
```
Index(['Gender', 'ScheduledDay', 'AppointmentDay', 'Age', 'Neighbourhood',
       'Scholarship', 'Hipertension', 'Diabetes', 'Alcoholism', 'handicap',
       'SMS_received', 'no_show'],
      dtype='object')
```

[31]:
```
# Rename columns names - to lowercase & insert underscore '_' before Day in␣
↪ScheduledDay & AppointmentDay
df.rename(columns = lambda x : (x.split('Day')[0]+'_day').lower() if 'Day' in x␣
↪else x.lower(), inplace=True)
```

Test :

[32]:
```
# Check whether the names are changed
df.columns
```

[32]:
```
Index(['gender', 'scheduled_day', 'appointment_day', 'age', 'neighbourhood',
       'scholarship', 'hipertension', 'diabetes', 'alcoholism', 'handicap',
       'sms_received', 'no_show'],
      dtype='object')
```

1.1.4 Issue 4

Define :

Change datatype of 'scheduled_day' & 'appointment_day', from string to Timestamp

Clean :

[33]:
```
# Change datatype from string to timestamp using datetime package
df.scheduled_day = df.scheduled_day.apply(lambda x : dt.datetime.
↪strptime(x,"%Y-%m-%dT%H:%M:%SZ"))
df.appointment_day = df.appointment_day.apply(lambda x : dt.datetime.
↪strptime(x,"%Y-%m-%dT%H:%M:%SZ"))
```

Test :

[34]:
```
# Check if the datatype is changed
type(df.scheduled_day[0]) , df.appointment_day[0]
```

[34]:
```
(pandas._libs.tslibs.timestamps.Timestamp, Timestamp('2016-04-29 00:00:00'))
```

1.1.5 Issue 5

Define :

Swap the values among 'handicap' & 'sms_received' columns

Clean :

[35]:
```
# list out few lines
df.head()
```

[35]:
```
  gender        scheduled_day appointment_day  age      neighbourhood  \
0      F 2016-04-29 18:38:08      2016-04-29   62    JARDIM DA PENHA
1      M 2016-04-29 16:08:27      2016-04-29   56    JARDIM DA PENHA
2      F 2016-04-29 16:19:04      2016-04-29   62      MATA DA PRAIA
3      F 2016-04-29 17:29:31      2016-04-29    8  PONTAL DE CAMBURI
4      F 2016-04-29 16:07:23      2016-04-29   56    JARDIM DA PENHA

   scholarship  hipertension  diabetes  alcoholism  handicap  sms_received  \
0            0             1         0           0         0             0
1            0             0         0           0         0             0
2            0             0         0           0         0             0
3            0             0         0           0         0             0
4            0             1         1           0         0             0

  no_show
0      No
1      No
2      No
3      No
4      No
```

[36]:
```python
# Unique values in 'handicap' & 'sms_received' before swap
df.handicap.unique() , df.sms_received.unique()
```

[36]:
```
(array([0, 1, 2, 3, 4], dtype=int64), array([0, 1], dtype=int64))
```

[37]:
```python
# Swap by storing the copy of one column value temporarily
temp = df.handicap.copy()
df.handicap = df.sms_received
df.sms_received = temp
```

Test :

[38]:
```python
# Unique values after swap
df.handicap.unique() , df.sms_received.unique()
```

[38]:
```
(array([0, 1], dtype=int64), array([0, 1, 2, 3, 4], dtype=int64))
```

1.1.6 Issue 6

Define :

`Fill in inaccurate data in 'age' (0,-1) with age mean`

Clean :

[39]:
```python
# Number of samples
df.shape[0]
```

[39]:
```
110527
```

[40]:
```python
# Counts of each age
df.age.value_counts()
```

[40]:
```
 0      3539
 1      2273
 52     1746
 49     1652
 53     1651
        …
 115       5
 100       4
 102       2
 99        1
-1         1
Name: age, Length: 104, dtype: int64
```

[41]:
```python
# Number of inaccurate data
print('Number of inaccurate data for age[0,-1] :',df.query('age in [0,-1]').
 ↪count().age)
```

```
# Proportion of inaccurate data
print('Proportion of inaccurate data for age[0,-1] :',df.query('age in [0,-1]').
↪count().age / df.shape[0])
```

```
Number of inaccurate data for age[0,-1] : 3540
Proportion of inaccurate data for age[0,-1] : 0.03202837315768998
```

Note : - Age 0 has sample count of 3539 out of 110527, which has a proportion of 0.0320 - As the proportion is low, fill in with mean age

[42]:
```
# Change value of age in [0,-1] to age.mean
mean_age = int(df.age.mean())
df.age = df.age.apply(lambda x : mean_age if x in [0,-1] else x)
```

[43]:
```
# The below query should be returned empty after drop
df.query('age in [0,-1]')
```

[43]:
```
Empty DataFrame
Columns: [gender, scheduled_day, appointment_day, age, neighbourhood,
scholarship, hipertension, diabetes, alcoholism, handicap, sms_received,
no_show]
Index: []
```

Test :

[44]:
```
# number of samples after dropping the patient details
df.shape[0]
```

[44]:
```
110527
```

[45]:
```
# Check that the column 'age' has no value of 0/-1
df.age.all() in [0,-1]     # should return False
```

[45]:
```
False
```

1.1.7 Issue 7

Define :

Change the data in 'no_show' column as, 'No'-1 & 'Yes'-0 of integer type

Clean :

[46]:
```
# number of unique values in 'no_show' before change
df.no_show.value_counts()
```

[46]:
```
No     88208
Yes    22319
Name: no_show, dtype: int64
```

[47]:
```python
# Change value 'No'=1 & 'Yes'=0 of type int
df.no_show = df.no_show.apply(lambda x : int(1) if x=='No' else int(0))
```

1.1.8 Test :

[48]:
```python
# Check the number of unique value after value change
df.no_show.value_counts()
```

[48]:
```
1    88208
0    22319
Name: no_show, dtype: int64
```

[49]:
```python
# Check the datatype
df.no_show.dtype
```

[49]:
```
dtype('int64')
```

1.1.9 Feature Engineering :

Define :

Create a column named 'awaiting_days' for easier analysis, which is the difference between sche

Create :

[50]:
```python
# Difference between scheduled_day & appointment_day
df.scheduled_day - df.appointment_day
```

[50]:
```
0           0 days 18:38:08
1           0 days 16:08:27
2           0 days 16:19:04
3           0 days 17:29:31
4           0 days 16:07:23
                   …
110522   -35 days +09:15:35
110523   -35 days +07:27:33
110524   -41 days +16:03:52
110525   -41 days +15:09:23
110526   -41 days +13:30:56
Length: 110527, dtype: timedelta64[ns]
```

Note : As we can see the result in previous step as,

> - Some results are positive, some are negative
> - So, create a column <i>'awaiting_days'</i>, which is the result of absolute difference betw

[51]:
```python
# Feature creation
df['awaiting_days'] = abs(df.scheduled_day - df.appointment_day).dt.days
```

Test :

[52]:
```
# Check if the feature is created
df.tail()
```

[52]:
```
        gender       scheduled_day appointment_day  age neighbourhood  \
110522      F 2016-05-03 09:15:35      2016-06-07   56   MARIA ORTIZ
110523      F 2016-05-03 07:27:33      2016-06-07   51   MARIA ORTIZ
110524      F 2016-04-27 16:03:52      2016-06-07   21   MARIA ORTIZ
110525      F 2016-04-27 15:09:23      2016-06-07   38   MARIA ORTIZ
110526      F 2016-04-27 13:30:56      2016-06-07   54   MARIA ORTIZ

        scholarship  hipertension  diabetes  alcoholism  handicap  \
110522            0             0         0           0         1
110523            0             0         0           0         1
110524            0             0         0           0         1
110525            0             0         0           0         1
110526            0             0         0           0         1

        sms_received  no_show  awaiting_days
110522             0        1             34
110523             0        1             34
110524             0        1             40
110525             0        1             40
110526             0        1             40
```

[53]:
```
# Check the datatype of newly created feature
type(df.awaiting_days[0])
```

[53]: numpy.int64

Store the cleaned data in new file,

[54]:
```
df.to_csv('no_show_cleaned.csv', index=False)
```

Exploratory Data Analysis

[55]:
```
# Load cleaned data into the DataFrame
df = pd.read_csv('no_show_cleaned.csv',
                 parse_dates=['scheduled_day', 'appointment_day'],
                 infer_datetime_format=True)
```

Research Question 1 : Patients with which disease schedules appoinments and doesn't shows up the most?

[56]:
```
# Diseases formed as an numpy array for further calculation
diseases = np.array(['alcoholism','diabetes','hipertension','handicap'])
```

[57]:
```
# total number of patients appointments
total_count = df.shape[0]
total_count
```

[57]: 110527

[58]:
```
# total number of patients didn't show up for appointments
total_noshowup = df.no_show.value_counts()[0]

# total number of patients show up
total_showup = df.no_show.value_counts()[1]
```

Process : - Now that, we have value of total number of patients, total no show & total show up - We have to calculate number of patients, no show & show up counts specific for each disease - Also, those who don't have any disease - As, the count may vary rapidly, we have to calculate proportions for the same set of feature conditions

[59]:
```
# Function to calculate number/proportions of patients having specific disease␣
↪booked for an appointment/noShow/ShowUp
def patient_disease_count(df, diseases, case, divide=1):
    result = []

    # number of patients having specific disease booked an appointment,␣
↪divide=1 (default)
    # proportions of patients having specific disease booked an appointment,␣
↪when 'divide' is passed from func call
    if case == 'total_count':
        for disease in diseases:
            result.append(df[disease].value_counts()[1]/divide)

    # number of patients didn't show, divide=1 (default)
    # proportions of patients didn't show, when 'divide' is passed from func␣
↪call
    elif case == 'noshow_count':
        for disease in diseases:
            result.append(df.query('{}==1 and no_show==0'.format(disease)).
↪shape[0]/divide)

    # number of patients show up, divide=1 (default)
    # proportions of patients show up, when 'divide' is passed from func call
    else:
        for disease in diseases:
            result.append(df.query('{}==1 and no_show==1'.format(disease)).
↪shape[0]/divide)
    return result
```

- Calculating the above said process by calling the function written above, and storing the values as a DataFrame

[60]:
```
# Create a DataFrame to store the values
df_dis = pd.DataFrame({
    'disease' : diseases,
```

```
    'patient_count' : np.array(patient_disease_count(df, diseases,␣
↪'total_count')),
    'patient_proportion' : np.array(patient_disease_count(df, diseases,␣
↪'total_count', total_count)),
    'no_show_count' : np.array(patient_disease_count(df, diseases,␣
↪'noshow_count')),
    'no_show_proportion' : np.array(patient_disease_count(df, diseases,␣
↪'noshow_count', total_noshowup)),
    'show_up_count' : np.array(patient_disease_count(df, diseases,␣
↪'show_count')),
    'show_up_proportion' : np.array(patient_disease_count(df, diseases,␣
↪'show_count', total_showup))
})
df_dis
```

[60]:

```
        disease  patient_count  patient_proportion  no_show_count  \
0    alcoholism         3360.0            0.030400          677.0
1      diabetes         7943.0            0.071865         1430.0
2  hipertension        21801.0            0.197246         3772.0
3      handicap        35482.0            0.321026         9784.0

   no_show_proportion  show_up_count  show_up_proportion
0            0.030333         2683.0            0.030417
1            0.064071         6513.0            0.073837
2            0.169004        18029.0            0.204392
3            0.438371        25698.0            0.291334
```

Further Process : - The same process followed for those who didn't have any disease, but booked for an appointment

[61]:

```
# Query for those who don't have any disease
no_dis = df.query('hipertension==0 & diabetes==0 & alcoholism==0 & handicap==0')
```

[62]:

```
# No_disease patients - show up & no show (value counts)
no_dis.no_show.value_counts()
```

[62]:

```
1    47721
0     9890
Name: no_show, dtype: int64
```

- Calculate number and proportion for those who don't have any disease
- Add it to the DataFrame

[63]:

```
# Adding the No_disease patient info as a row in df_dis DataFrame
df_dis.loc[len(df_dis.index)] = ['no_disease',
                                  no_dis.shape[0], no_dis.shape[0]/total_count,
                                  no_dis.no_show.value_counts()[0], no_dis.
↪no_show.value_counts()[0]/total_noshowup,
```

```
                                 no_dis.no_show.value_counts()[1], no_dis.
↪no_show.value_counts()[1]/total_showup]
df_dis
```

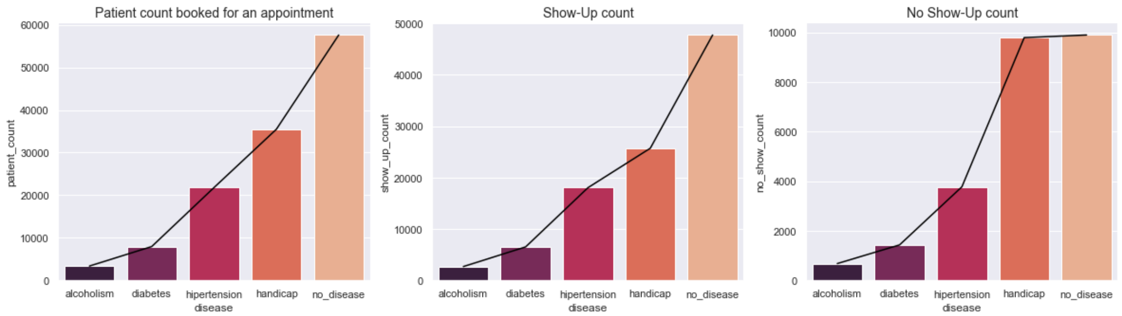
[63]:

```
        disease  patient_count  patient_proportion  no_show_count  \
0    alcoholism         3360.0            0.030400          677.0
1      diabetes         7943.0            0.071865         1430.0
2  hipertension        21801.0            0.197246         3772.0
3      handicap        35482.0            0.321026         9784.0
4    no_disease        57611.0            0.521239         9890.0

   no_show_proportion  show_up_count  show_up_proportion
0            0.030333         2683.0            0.030417
1            0.064071         6513.0            0.073837
2            0.169004        18029.0            0.204392
3            0.438371        25698.0            0.291334
4            0.443120        47721.0            0.541005
```

- Let's plot the counts of patients for each disease and no disease
- show up & no show counts

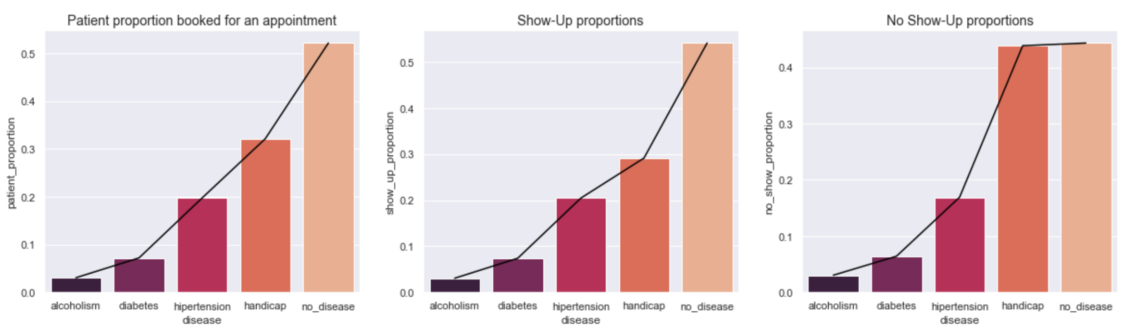
[64]:

```
# set background theme as darkgrid
sns.set_theme(style="darkgrid")

# set the figure size
plt.figure(figsize=(20,5))

# Plot the responses for patient counts
plt.subplot(1,3,1)
plt.title('Patient count booked for an appointment', fontsize=14)
sns.lineplot(x=df_dis.disease, y=df_dis.patient_count, color='black')
sns.barplot(x=df_dis.disease, y=df_dis.patient_count, palette="rocket")

# Plot the responses for show up count
plt.subplot(1,3,2)
plt.title('Show-Up count', fontsize=14)
sns.lineplot(x=df_dis.disease, y=df_dis.show_up_count, color='black')
sns.barplot(x=df_dis.disease, y=df_dis.show_up_count, palette="rocket")

# Plot the responses for no-show up count
plt.subplot(1,3,3)
plt.title('No Show-Up count', fontsize=14)
sns.lineplot(x=df_dis.disease, y=df_dis.no_show_count, color='black')
sns.barplot(x=df_dis.disease, y=df_dis.no_show_count, palette="rocket");
```



Note : As the count vary in large amount for show-up and no-show, let us plot the proportions

[65]:
```
# set the figure size
plt.figure(figsize=(20,5))

# Plot the responses for patient counts proportions
plt.subplot(1,3,1)
plt.title('Patient proportion booked for an appointment', fontsize=14)
sns.lineplot(x=df_dis.disease, y=df_dis.patient_proportion, color='black')
sns.barplot(x=df_dis.disease, y=df_dis.patient_proportion, palette="rocket");

# Plot the responses for show up proportions
plt.subplot(1,3,2)
plt.title('Show-Up proportions', fontsize=14)
sns.lineplot(x=df_dis.disease, y=df_dis.show_up_proportion, color='black')
sns.barplot(x=df_dis.disease, y=df_dis.show_up_proportion, palette="rocket");

# Plot the responses for no-show up count
plt.subplot(1,3,3)
plt.title('No Show-Up proportions', fontsize=14)
sns.lineplot(x=df_dis.disease, y=df_dis.no_show_proportion, color='black')
sns.barplot(x=df_dis.disease, y=df_dis.no_show_proportion, palette="rocket");
```



conclusion : - People with **no diseases** books for an appointment the most, has greater proportions in no-show up among others, followed by **handicap** - However, **Handicaps** are those who fails to show up for an fixed appointment the most, as proportions of all disease and no-disease for show-up is more than no-show, except handicap. - And also, there is a drastic difference between no-show & show-up proportion for 'handicap', where no-show occurence is greater

Research Question 2 : How does the number of SMS received affects the show-up for scheduled appoinments?

Process : - Let's group the dataset by **sms_received** & **no_show** - Calculate proportions and load into a DataFrame

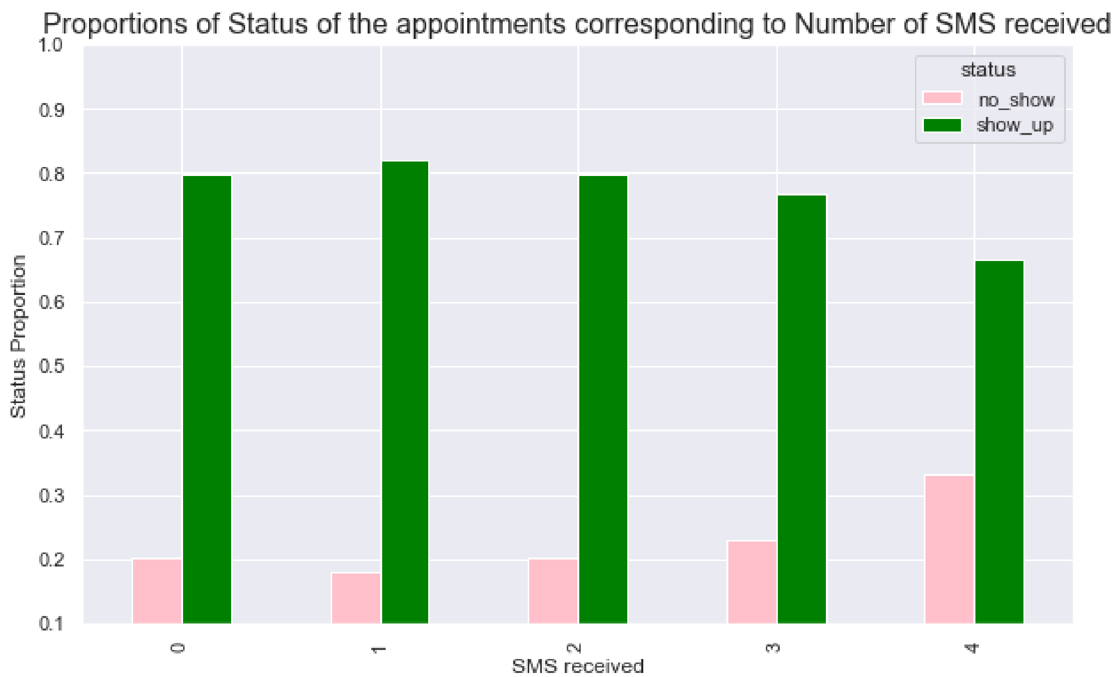
[66]:
```
# Group the dataset, calculate propotion & load into dataframe
sms_propor = pd.DataFrame((df.groupby(['sms_received','no_show']).size() / df.
↪groupby('sms_received').size()).unstack(level=[1]))
sms_propor
```

[66]:
```
no_show              0         1
sms_received
0             0.202353  0.797647
1             0.179236  0.820764
2             0.202186  0.797814
3             0.230769  0.769231
4             0.333333  0.666667
```

[67]:
```
# Set the common column name as status & rename each column name
sms_propor.columns.name = 'status'
sms_propor.rename(columns={
    sms_propor.columns[0] :'no_show',
    sms_propor.columns[1] :'show_up'
},inplace=True)
sms_propor
```

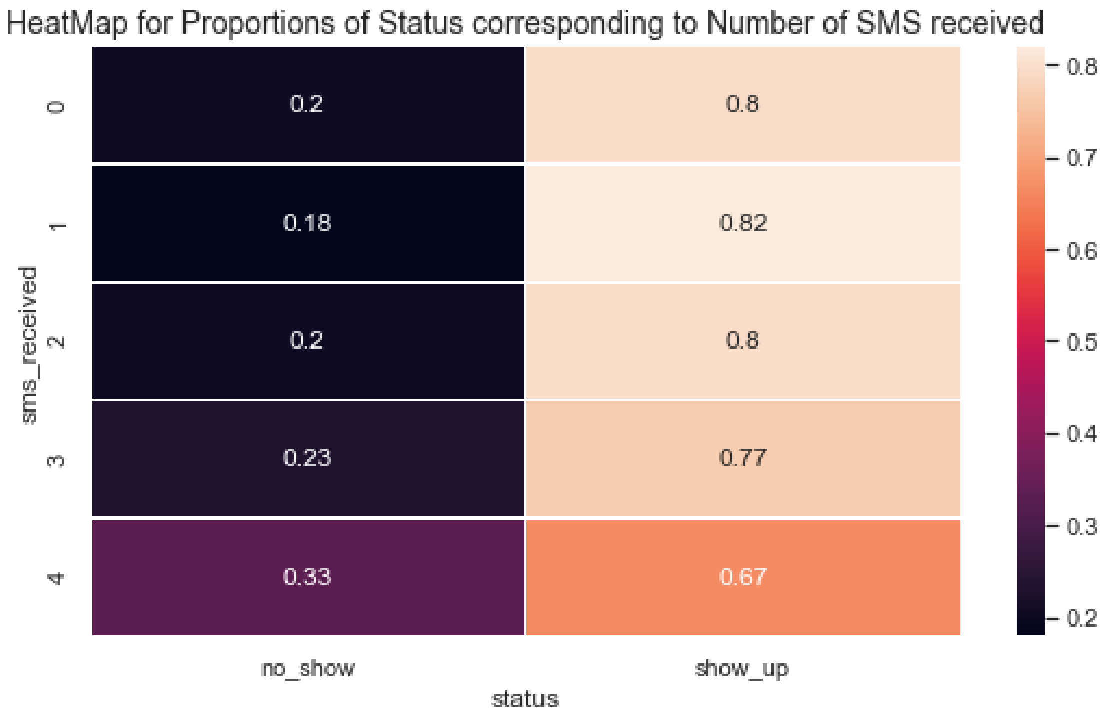
[67]:
```
status         no_show   show_up
sms_received
0             0.202353  0.797647
1             0.179236  0.820764
2             0.202186  0.797814
3             0.230769  0.769231
4             0.333333  0.666667
```

[68]:
```
# Plot the response
sms_propor.plot(kind='bar', color=['pink','green'], figsize=(10,6))
plt.title('Proportions of Status of the appointments corresponding to Number of␣
↪SMS received', fontsize=16)
plt.ylim(0.1,1.0)
plt.xlabel('SMS received')
plt.ylabel('Status Proportion');
```



- Further Visualisation using heat map

[69]:
```
# Heatmap visualisation
plt.figure(figsize=(9,5))
sns.heatmap(sms_propor, annot=True, linewidths=.4)
plt.title('HeatMap for Proportions of Status corresponding to Number of SMS␣
↪received', fontsize=14);
```



conclusion : - We can see that **1 SMS** has increased likelihood of patient showing up by **0.2%** (0.82) compared to No SMS (0.797) in proportion. - However, the increase in number of SMS decreased the likelihood of patient showing up. Because they may have already decided not to show up

Research Question 3 : How does Gender affects the No Show occurence?

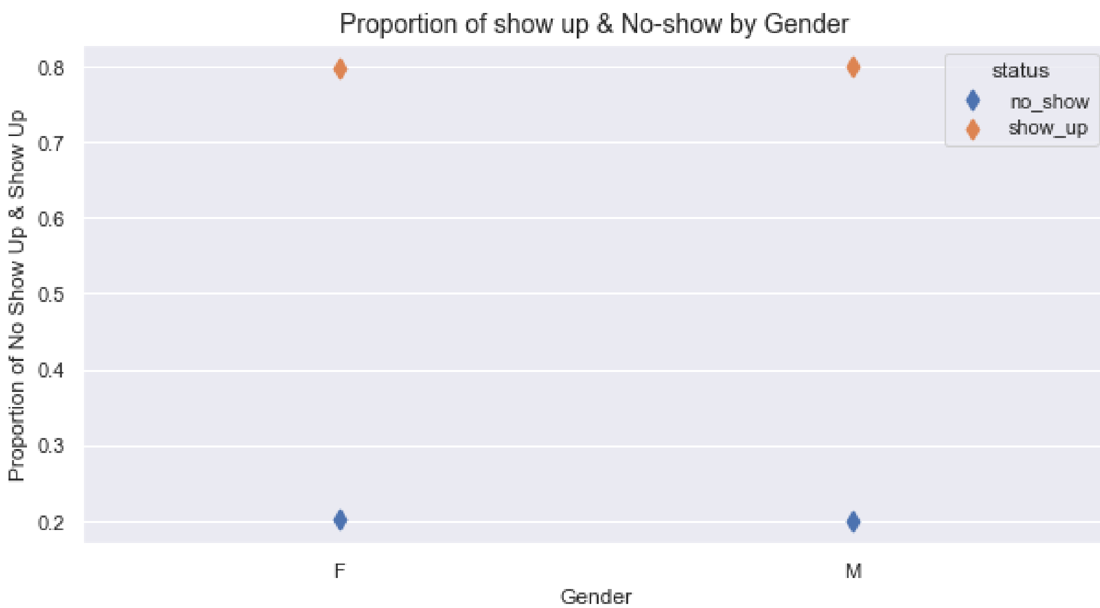
Process : - Group by **gender** & **no_show** - Calculate proportion of no_show

[70]:
```
# Proportion of no-show & show up by Gender
gen = (df.groupby(['gender','no_show']).size() / df.groupby('gender').size()).
↪reset_index()

# Column names changing
gen.columns = ['gender', 'status', 'proportion']

# Converting 0 & 1 to corresponding status value
gen.status = gen.status.apply(lambda x : 'no_show' if x==0 else 'show_up')
gen
```

[70]:

	gender	status	proportion
0	F	no_show	0.203146
1	F	show_up	0.796854
2	M	no_show	0.199679
3	M	show_up	0.800321

[71]:
```
# Plot the responses using pointplot
plt.figure(figsize=(10,5))
sns.pointplot(data=gen, x='gender', y='proportion',hue='status', join=False,␣
↪markers="d", palatte='dark')
plt.title('Proportion of show up & No-show by Gender', fontsize=14)
plt.xlabel('Gender')
plt.ylabel('Proportion of No Show Up & Show Up');
```



conclusion : - We can see that Proportions of 'Female' & 'Male' for both no-show & show-up are almost **equal** - It is clear that 'Gender' **doesn't influence** the No Show occurence

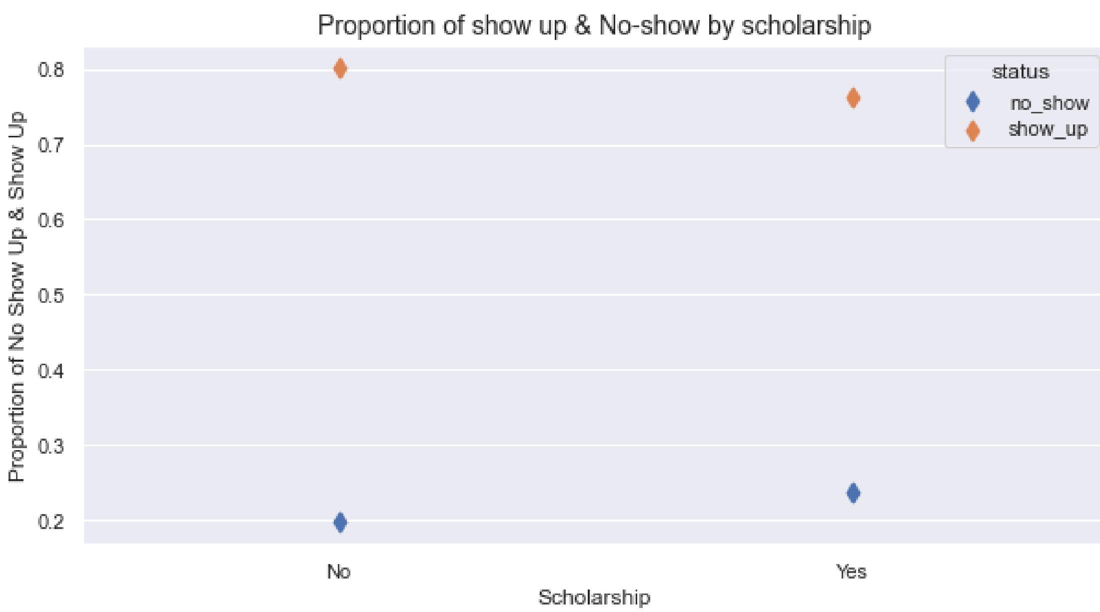
Research Question 4 : How does Scholarship affects the No Show occurence?

Process : - Group by **scholarship** & **no_show** - Calculate proportion of no_show

[72]:
```
# Proportion of no-show & show up by scholarship
scho = (df.groupby(['scholarship','no_show']).size() / df.
↪groupby('scholarship').size()).reset_index()

# Column names changing
scho.columns = ['scholarship', 'status', 'proportion']

# Converting 0 & 1 to corresponding status value
scho.status = scho.status.apply(lambda x : 'no_show' if x==0 else 'show_up')
scho.scholarship = scho.scholarship.apply(lambda x : 'No' if x==0 else 'Yes')

scho
```

[72]:
```
  scholarship   status  proportion
0          No  no_show    0.198072
1          No  show_up    0.801928
2         Yes  no_show    0.237363
3         Yes  show_up    0.762637
```

[73]:
```
# Plot the responses using pointplot
plt.figure(figsize=(10,5))
sns.pointplot(data=scho, x='scholarship', y='proportion',hue='status',␣
 ↪join=False, markers="d", palatte='dark')
plt.title('Proportion of show up & No-show by scholarship', fontsize=14)
plt.xlabel('Scholarship')
plt.ylabel('Proportion of No Show Up & Show Up');
```



conclusion : - We can see that patient's having Scholarship tends to have **higher** No Show-up rate

Research Question 5 : How does Age affects the No Show rate?

Process : - Categorize age into different groups - Calculate the proportion for each age category

[74]:
```
df.age.describe()
```

[74]:
```
count    110527.000000
mean         38.273933
std          22.104720
min           1.000000
```

```
25%           20.000000
50%           37.000000
75%           55.000000
max          115.000000
Name: age, dtype: float64
```

We can classify the age groups as,

```
> - **children :** 1-14 years
> - **youth :** 15-24 years
> - **adult :** 25-64 years
> - **seniors :** above 65 years
```

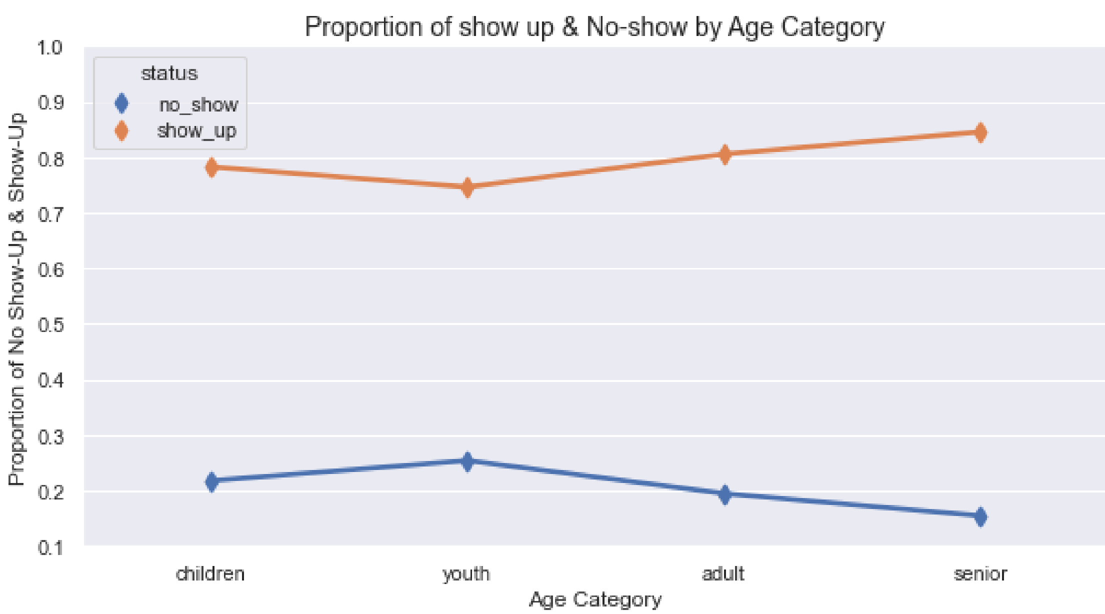
[75]:
```
# categorize age
bins = np.array([ 1, 15, 25, 65, 116])
df['age_category'] = pd.cut(df.age, bins=bins, labels=['children', 'youth',␣
↪'adult', 'senior'], right=False)
```

[76]:
```
# Proportion calculation
age_cat = (df.groupby(['age_category','gender', 'no_show']).size()/df.
↪groupby(['age_category','gender']).size()).reset_index()

# Column names changing
age_cat.columns = ['age_category', 'gender', 'status', 'proportion']

# Converting 0 & 1 to corresponding status value
age_cat.status = age_cat.status.apply(lambda x : 'no_show' if x==0 else␣
↪'show_up')
age_cat
```

[76]:

	age_category	gender	status	proportion
0	children	F	no_show	0.216818
1	children	F	show_up	0.783182
2	children	M	no_show	0.217969
3	children	M	show_up	0.782031
4	youth	F	no_show	0.251204
5	youth	F	show_up	0.748796
6	youth	M	no_show	0.256303
7	youth	M	show_up	0.743697
8	adult	F	no_show	0.199569
9	adult	F	show_up	0.800431
10	adult	M	no_show	0.189751
11	adult	M	show_up	0.810249
12	senior	F	no_show	0.156215
13	senior	F	show_up	0.843785
14	senior	M	no_show	0.152633
15	senior	M	show_up	0.847367

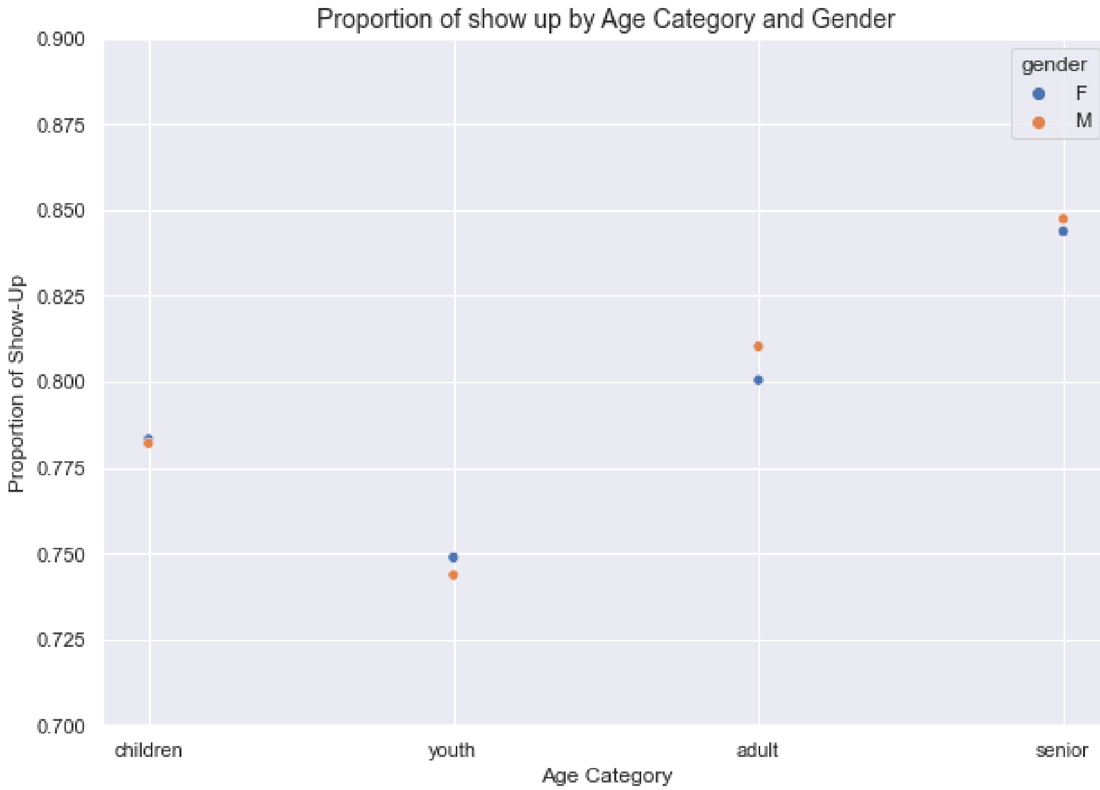
```
[77]: # Plot the responses using pointplot
plt.figure(figsize=(10,5))
sns.pointplot(data=age_cat, x='age_category', y='proportion',hue='status',␣
 ↪markers="d", palatte='dark')
plt.title('Proportion of show up & No-show by Age Category', fontsize=14)
plt.ylim(0.1, 1.0)
plt.ylabel('Proportion of No Show-Up & Show-Up')
plt.xlabel('Age Category');
```



- **Youth** has the highest No Show-Up rate
- **Seniors** have the highest Show-Up rate

Now, let's plot with Gender details for each Age group

```
[78]: # Plot with gender & age group details
plt.figure(figsize=(10,7))
sns.scatterplot(x="age_category", y="proportion",
                hue="gender", data=age_cat)
plt.title('Proportion of show up by Age Category and Gender', fontsize=14)
plt.ylim(0.7, 0.9)
plt.ylabel('Proportion of Show-Up')
plt.xlabel('Age Category');
```



- In adult & seniors, **Males** have **Show-Up** rate slightly higher than Females
- While in youth, **Males** have slightly lower **Show-Up** rate than Females

[79]:
```
# Group by age
age = (df.groupby(['age','gender','no_show']).size()/df.
 ↪groupby(['age','gender']).size()).reset_index()

# Column names changing
age.columns = ['age', 'gender', 'status', 'proportion']

# Converting 0 & 1 to corresponding status value
age.status = age.status.apply(lambda x : 'no_show' if x==0 else 'show_up')
age
```

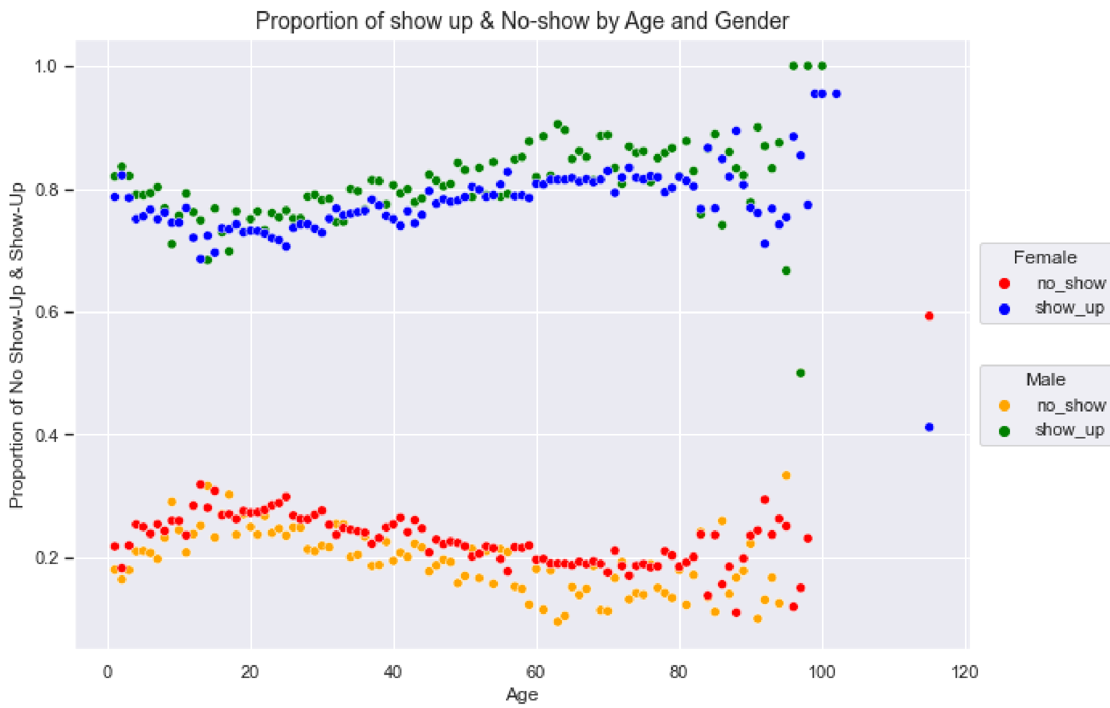
[79]:

	age	gender	status	proportion
0	1	F	no_show	0.185662
1	1	F	show_up	0.814338
2	1	M	no_show	0.179747
3	1	M	show_up	0.820253
4	2	F	no_show	0.146631
..	…	…	…	…
391	100	F	show_up	1.000000

```
392  100      M  show_up     1.000000
393  102      F  show_up     1.000000
394  115      F  no_show     0.600000
395  115      F  show_up     0.400000

[396 rows x 4 columns]
```

[80]:

```
# Plot by Age and Gender
plt.figure(figsize=(10,7))

# Get current axes
ax = plt.gca()
male = sns.scatterplot(x="age", y="proportion",
                hue="status", palette=['orange','green'],
                data=age.query('gender=="M"'), ax=ax)

plt.ylabel('Proportion of No Show-Up & Show-Up')
plt.xlabel('Age')

# Create a twin axis - only use it  for the legend
ax2 = ax.twinx()
female = sns.scatterplot(x="age", y="proportion",
                hue="status", palette=['red','blue'],
                data=age.query('gender=="F"'), ax=ax2)

# Remove the twin y-axis
ax2.get_yaxis().set_visible(False)

# add both legends
ax.legend(loc='center left', bbox_to_anchor=(1, 0.4), title="Male")
ax2.legend(loc='center left', bbox_to_anchor=(1, 0.6), title="Female")

plt.title('Proportion of show up & No-show by Age and Gender', fontsize=14)
plt.ylim(0.0, 1.1);
```



Conclusion : - **Youth** has the highest No Show-Up rate, as they may have higher immunity - **Seniors** have the highest Show-Up rate, - In adult & seniors, **Males** have **Show-Up** rate slightly higher than Females - While in youth, **Males** have slightly lower **Show-Up** rate than Females - Few patients of age between 95 and 115 have the 100% Show_Up

Research Question 6 : Does the gap between scheduled date and appointment date affect patient's no-show rate?

Process : - Let's describe the **awaiting_days** and categorize the gap - Create a new column **awaiting_days_category** to store the gap categorizations - Calculate the proportion of status of the appointment for each gap

[81]:
```
df.awaiting_days.describe()
```

[81]:
```
count    110527.000000
mean          9.532829
std          15.027683
min           0.000000
25%           0.000000
50%           3.000000
75%          14.000000
max         178.000000
Name: awaiting_days, dtype: float64
```

Since the min and the 25% quantile are both zero, we'll then make our lowest category to be no_gap. For the rest, we'll categorize the schedule gap as either short, medium, or long gap. The categorizations will then be as follows:

- no_gap: 0 days gap
- short: 1 to 3 days gap
- medium: 4 to 14 days gap
- long : 15 to 178 days gap

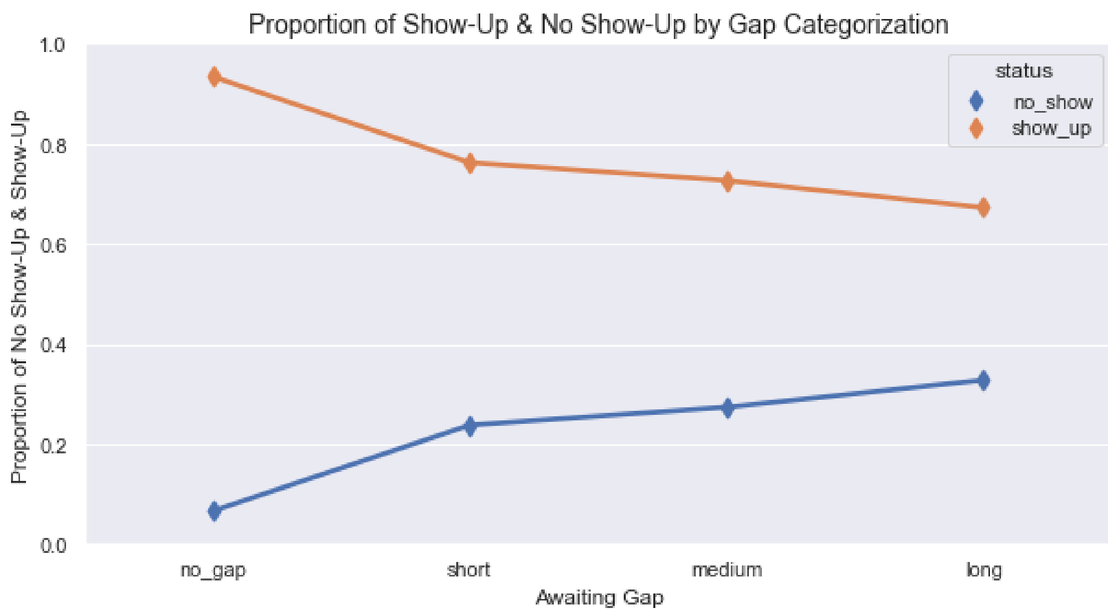
[82]:
```
# Create bins from describe
bins = df.awaiting_days.describe()[['min', '25%', '50%', '75%', 'max']].values
bins[1] = 0.1    # because the 25% quantile is also 0.0 days, need to set to␣
↪small number so that the cut works

# Create column
df['awaiting_days_category'] = pd.cut(df.awaiting_days, bins=bins,␣
↪labels=['no_gap', 'short', 'medium', 'long'], right=False)
```

[83]:
```
# calculate the proportion of status of the appointment for each gap
days = (df.groupby(['awaiting_days_category','no_show']).size()/df.
↪groupby(['awaiting_days_category']).size()).reset_index()

# Column names changing
days.columns = ['awaiting_days_category', 'status', 'proportion']

# Converting 0 & 1 to corresponding status value
days.status = days.status.apply(lambda x : 'no_show' if x==0 else 'show_up')
days
```

[83]:

	awaiting_days_category	status	proportion
0	no_gap	no_show	0.066361
1	no_gap	show_up	0.933639
2	short	no_show	0.237693
3	short	show_up	0.762307
4	medium	no_show	0.273497
5	medium	show_up	0.726503
6	long	no_show	0.327481
7	long	show_up	0.672519

[84]:
```
# Plot the responses using pointplot
plt.figure(figsize=(10,5))
sns.pointplot(data=days, x='awaiting_days_category',␣
↪y='proportion',hue='status', markers="d")
plt.title('Proportion of Show-Up & No Show-Up by Gap Categorization',␣
↪fontsize=14);
plt.ylim(0.0, 1.0)
plt.ylabel('Proportion of No Show-Up & Show-Up')
plt.xlabel('Awaiting Gap');
```



Conclusion : - As we can see, longer the gap between **scheduled_day** and **appointment_day**, no-show rate is **higher** - The patients seems to Show Up immediately after booking the appointment, as **No Gap** has the highest show-up rate - Even, the no show-up rate of short gap increased significantly compared to no_gap. This may be because patients have changed their mind / got cured

Research Question 7 : Does the Day of the week affect patient's no-show probability?

Process : - Let's create a new column **appointment_weekday** to store the day of the week of **appointment_day** - Calculate the proportion of status of the appointment for each weekday

[85]:
```
# create new column
df['appointment_weekday'] = df.appointment_day.dt.dayofweek

# print out few lines
df[['appointment_day', 'appointment_weekday']].head()
```

[85]:

	appointment_day	appointment_weekday
0	2016-04-29	4
1	2016-04-29	4
2	2016-04-29	4
3	2016-04-29	4
4	2016-04-29	4

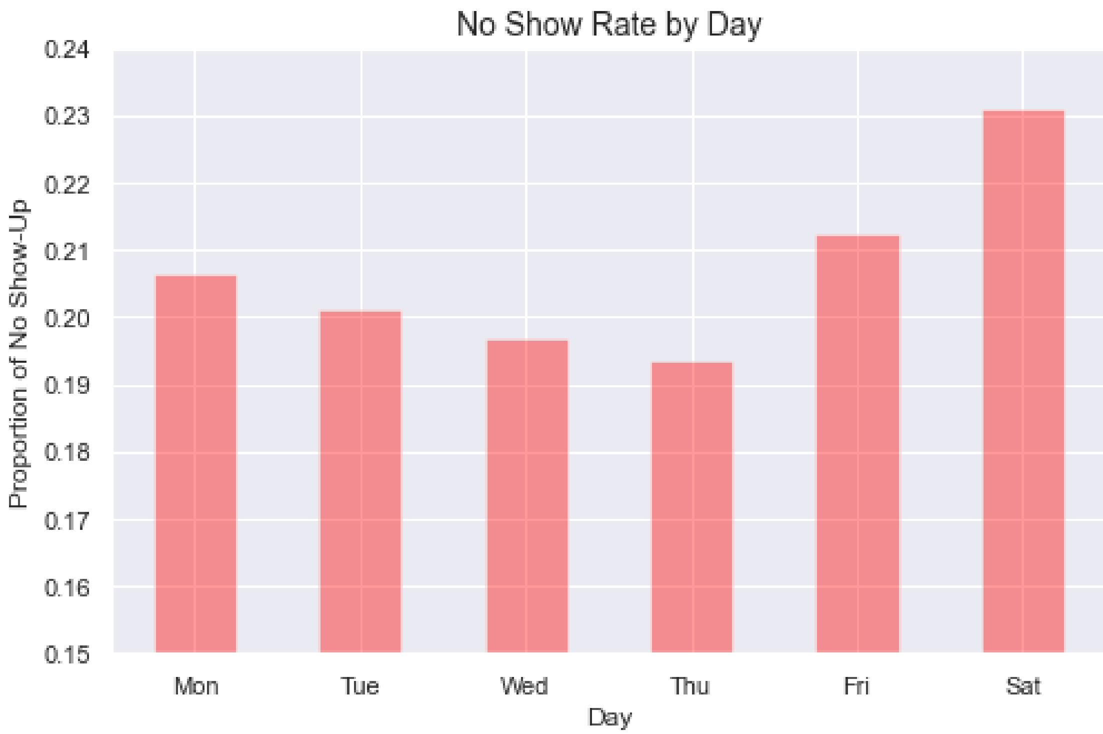
[86]:
```
# calculate the proportion of status of the appointment for each weekday
week_day = (df.groupby(['no_show','appointment_weekday']).size()/df.
↪groupby(['appointment_weekday']).size())
```

```
week_day
```

[86]:
```
no_show  appointment_weekday
0        0                      0.206471
         1                      0.200936
         2                      0.196892
         3                      0.193541
         4                      0.212261
         5                      0.230769
1        0                      0.793529
         1                      0.799064
         2                      0.803108
         3                      0.806459
         4                      0.787739
         5                      0.769231
dtype: float64
```

Value of weekday corresponds as, - 0 - Monday - 1 - Tuesday - 2 - Wednesday - 3 - Thursday - 4 - Friday - 5 - Saturday - 6 - Sunday

[87]:
```
# Plot the response
plt.figure(figsize=(8,5))
week_day[0].plot(kind='bar', color='red', alpha=0.4)
plt.title('No Show Rate by Day', fontsize=14)

# As we have week day value 0-5
labels = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat']
plt.xticks(ticks=range(6), labels=labels, rotation=0)
plt.ylim(0.15, 0.24)
plt.ylabel('Proportion of No Show-Up')
plt.xlabel('Day');
```



Conclusion : - We can see that proportion to No show-up rate decreases from Monday to Thursday and again starts increasing, and reach its peak on **Saturday** - **Mid-week (Thursday)** is having lowest No show-up rate - This may be because of the **higher workload** on initial & end of the week, lower workload in mid-week

Conclusions

Note: Findings are tentative. Not verified by the principles of statistics and machine learning.

- People with **no diseases** books for an appointment the most, followed by **handicap**
- However, **Handicaps** are those who fails to show up for an fixed appointment the most, as handicap is the only one having No Show-Up rate greater than the Show-Up rate.

- **1 SMS** has increased likelihood of patient showing up. However, the **increase** in number of SMS **decreased** the likelihood of patient showing up.

- 'Gender' **doesn't influence** the No Show occurence, as both no-show & show-up rate for 'Female' & 'Male' are almost **equal**

- Patient's having Scholarship tends to have **higher** No Show-up rate

- **Youth** has the highest No Show-Up rate, while **Seniors** have the highest Show-Up rate

- In **adult & seniors**, Males have Show-Up rate **slightly higher** than Females, while in **youth**, Males have slightly **lower** Show-Up rate than Females
- Few patients of age between 95 and 115 have the 100% Show_Up
- **Longer** the gap between **scheduled_day** and **appointment_day**, **higher** the no-show rate. **No Gap** has the highest show-up rate. Even, the no show-up rate of short gap increased significantly compared to no_gap.
- No show-up rate decreases from Monday to Thursday and again starts increasing, and reach its peak on **Saturday**. **Mid-week (Thursday)** is having lowest No show-up rate

So, What factors are important for us to know in order to predict if a patient will show up for their scheduled appointment? > **Tentatively,** > - Awaiting days > - Week Day > - Age Group > - Type of disease > - No.of SMS > - Scholarship

Reference websites used in addition to course material,

https://www.kaggle.com/joniarroba/noshowappointments

https://pandas.pydata.org

https://numpy.org/doc/stable/contents.html

https://stackoverflow.com

https://github.com

https://seaborn.pydata.org/index.html