```
from google.colab import files
files.upload()
```

- **UPHFinal_2.csv**(application/vnd.ms-excel) - 24807725 bytes, last modified: 4/4/2019 - 100% done

```
Saving UPHFinal_2.csv to UPHFinal_2.csv
{'UPHFinal_2.csv': b'"","AppointmentID","PatientID","ClinicNM","AppointmentDTS","Appoi
```

```
import numpy as np
import pandas as pd
from scipy import stats as ss
import statsmodels.api as sm
import sklearn.metrics as ssm
from datetime import datetime
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.stats.proportion import proportions_ztest

import os
df = pd.read_csv("UPHFinal_2.csv")
```

```
print('variables: ' + str(df.columns))
```

```
variables: Index(['Unnamed: 0', 'AppointmentID', 'PatientID', 'ClinicNM',
       'AppointmentDTS', 'AppointmentMonthNBR', 'AppointmentWeekdayNBR',
       'AppointmentHourNBR', 'AgeNBR', 'SexFLG', 'HispanicFLG', 'SingleFLG',
       'LivesInApartmentFLG', 'EmailFLG', 'ApptLagNBR', 'InsuranceDSC',
       'HypertensionFLG', 'AsthmaFLG', 'HeartDiseaseFLG', 'ObeseFLG',
       'DiabetesFLG', 'Noshow24NBR', 'CancellationsNBR', 'Latearrivals24NBR',
       'CheckintoCheckoutNBR', 'AppttoCheckoutNBR', 'CheckintoApptNBR',
       'Arrived24NBR', 'Providers24CNT', 'ThatProvider24NBR',
       'NoshowRate24NBR', 'EdVisitsNBR', 'IpVisitsNBR', 'NoShowFLG',
       'CancelledLateFLG', 'NewPatient', 'Cost', 'Rand', 'Revenue', 'Profit',
       'Age_bin', 'ClinicNM_num', 'SexFLG_num', 'InsuranceDSC_num',
       'Noshow24NBR_Bin', 'CheckintoCheckoutNBR_Bin', 'CheckintoApptNBR_bin',
       'Arrived24NBR_bin', 'Providers24CNT_bin', 'ThatProvider24NBR_bin',
       'NoshowRate24NBR_bin', 'CancellationsNBR_Bin', 'AppttoCheckoutNBR_Bin',
       'Latearrivals24NBR_Bin'],
      dtype='object')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 95221 entries, 0 to 95220
Data columns (total 54 columns):
Unnamed: 0                  95221 non-null int64
AppointmentID               95221 non-null int64
PatientID                   95221 non-null int64
ClinicNM                    95221 non-null object
AppointmentDTS              95221 non-null object
AppointmentMonthNBR         95221 non-null int64
AppointmentWeekdayNBR       95221 non-null int64
AppointmentHourNBR          95221 non-null int64
AgeNBR                      95221 non-null int64
SexFLG                      95221 non-null object
HispanicFLG                 95221 non-null int64
SingleFLG                   95221 non-null int64
LivesInApartmentFLG         95221 non-null int64
EmailFLG                    95221 non-null int64
ApptLagNBR                  95221 non-null int64
InsuranceDSC                95221 non-null object
HypertensionFLG             95221 non-null int64
AsthmaFLG                   95221 non-null int64
HeartDiseaseFLG             95221 non-null int64
ObeseFLG                    95221 non-null int64
DiabetesFLG                 95221 non-null int64
Noshow24NBR                 95221 non-null int64
CancellationsNBR            95221 non-null int64
Latearrivals24NBR           95221 non-null int64
CheckintoCheckoutNBR        95221 non-null int64
AppttoCheckoutNBR           95221 non-null int64
CheckintoApptNBR            95221 non-null int64
Arrived24NBR                95221 non-null int64
Providers24CNT              95221 non-null int64
ThatProvider24NBR           95221 non-null int64
NoshowRate24NBR             95221 non-null int64
EdVisitsNBR                 95221 non-null int64
IpVisitsNBR                 95221 non-null int64
NoShowFLG                   95221 non-null int64
CancelledLateFLG            95221 non-null int64
NewPatient                  95221 non-null int64
Cost                        95221 non-null int64
Rand                        95221 non-null float64
Revenue                     95221 non-null int64
Profit                      95221 non-null int64
Age_bin                     95221 non-null int64
ClinicNM_num                95221 non-null int64
SexFLG_num                  95221 non-null int64
InsuranceDSC_num            95221 non-null int64
Noshow24NBR_Bin             95221 non-null object
CheckintoCheckoutNBR_Bin    95221 non-null object
CheckintoApptNBR_bin        95221 non-null object
Arrived24NBR bin            95221 non-null object
```

```
df.shape
```

```
(95221, 54)
```

1. Univariate Analysis

a) PatientId

To build new features related to the apointment, is necessary to check how many patients there are, and how much appointments per patient there is. But first, let's check the variable type.

```
print('Total appointments: ' + format(df.shape[0], ",d"))
print('Distinct patients: ' + format(df['PatientID'].unique().shape[0], ",d"))
```

```
Total appointments: 95,221
Distinct patients: 33,473
```

```
print('Patients with more than one appointment: ' + format((df['PatientID'].value_counts() > 1
```

```
Patients with more than one appointment: 20,942
```

Nearly 60% of patients have more than one appointment. It's enough to justify the creation of the new variable: number of previous appointments booked and no-show rate based on previous appointments.**bold text**

```
df['PreviousApp'] = df.sort_values(by = ['PatientID','AppointmentDTS']).groupby(['PatientID']
```

```
a = df.groupby(pd.cut(df.PreviousApp, bins = [-1, 0,1,2,3,4,5, 85], include_lowest = True))[[
b = pd.DataFrame(a)
b.set_index(pd.Series(['0', '1', '2', '3', '4', '5', '> 5']))
```

	PreviousApp
0	33473
1	20942
2	13329
3	8734
4	5716
5	3788
> 5	9239

We need to build the rate of previous no-show per patients, for those with more than 1 PreviousApp.

```
df['NoShowFLG'] = (df['NoShowFLG'] == 1 )*1
df['PreviousNoShow'] = (df[df['PreviousApp'] > 0].sort_values(['PatientID', 'AppointmentDTS']
```

```
df['PreviousNoShow'].describe()
```

```
count    61748.000000
mean         0.092094
std          0.222620
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max          1.000000
Name: PreviousNoShow, dtype: float64
```

More than half of people with previous appointment have gone to all the appointments scheduled. Later on we'll study if this variable is important to predict No-Show, even though it has lots of missings.

b) Gender

```
df['SexFLG'].value_counts()
```

```
F    57070
M    38151
Name: SexFLG, dtype: int64
```

```
colors = ['lightcoral', 'lightskyblue']

plt.pie([57070, 38151], explode = (0.1, 0), labels = ['Female', 'Male'], colors = colors, aut

plt.title('Patient Gender', fontsize=15)
plt.gcf().set_size_inches(8, 8)
plt.show()
```

We can see that almost two thirds of the appointments are done by women, a number much higher than men. There are no missing nor atypical values.

c) ScheduledDay

First we must change the variable type to DateTime.

```
df['AppointmentDTS'] = pd.to_datetime(df['AppointmentDTS'])
df['AppointmentDTS2'] = df.apply(lambda x: x.AppointmentDTS.strftime("%x"), axis = 1)
AppointmentDTS = df.groupby(['AppointmentDTS2'])[['AppointmentDTS']].count()
```

```
AppointmentDTS.reset_index(inplace = True)
AppointmentDTS.columns = ['Date', 'Count']
```

```
AppointmentDTS['Date'] = pd.to_datetime(AppointmentDTS['Date'])
```

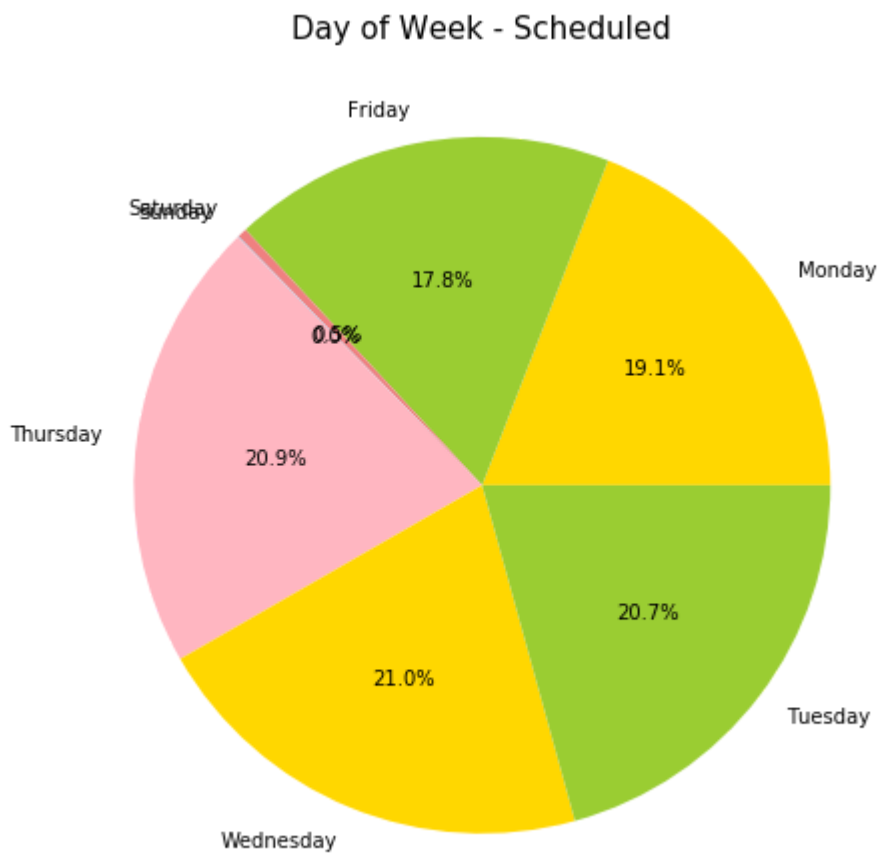
```
print('first appointment date: ' + str(AppointmentDTS.Date.min()))
print('most appointment date: ' + str(AppointmentDTS.Date.max()))
```

```
first appointment date: 2018-01-13 00:00:00
most appointment date: 2018-12-28 00:00:00
```

```
df['WeekdayScheduled'] = df.apply(lambda x: x.AppointmentDTS.isoweekday(), axis = 1)
df['WeekdayScheduled'].value_counts()
```

```
3    19983
4    19936
2    19711
1    18196
5    16929
6      429
7       37
Name: WeekdayScheduled, dtype: int64
```

```
colors = ['gold', 'yellowgreen', 'lightcoral', 'lightskyblue', 'lightpink']

plt.pie([df['WeekdayScheduled'].value_counts()[1], df['WeekdayScheduled'].value_counts()[5], (
         df['WeekdayScheduled'].value_counts()[4], df['WeekdayScheduled'].value_counts()[3], (
        labels = ['Monday','Friday','Saturday','Sunday','Thursday','Wednesday' ,'Tuesday'],
        colors = colors, autopct='%1.1f%%')

plt.title('Day of Week - Scheduled', fontsize=15)
plt.gcf().set_size_inches(8, 8)
plt.show()
```

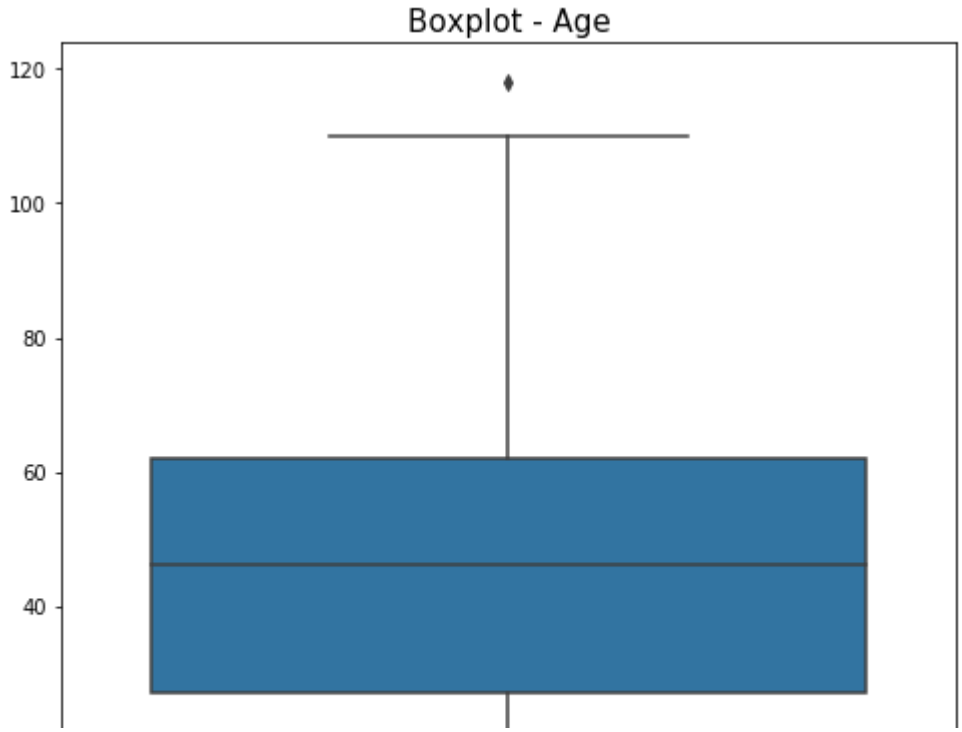


d) Age

```
df['AgeNBR'].describe()
```

```
count    95221.000000
mean        44.532792
std         22.984130
min          0.000000
25%         27.000000
50%         46.000000
75%         62.000000
max        118.000000
Name: AgeNBR, dtype: float64
```

```
ax = sns.boxplot(x=df['AgeNBR'], orient = 'v')

#plt.xlabel(' ')
plt.ylabel(' ')
plt.title('Boxplot - Age', fontsize = 15)
plt.gcf().set_size_inches(8, 8)
plt.show()
```




```
df['ClinicNM'].value_counts()
```

```
B    29990
A    26370
E    15084
C    14480
D     9297
Name: ClinicNM, dtype: int64
```

Additional Features:

e) PreviousDisease

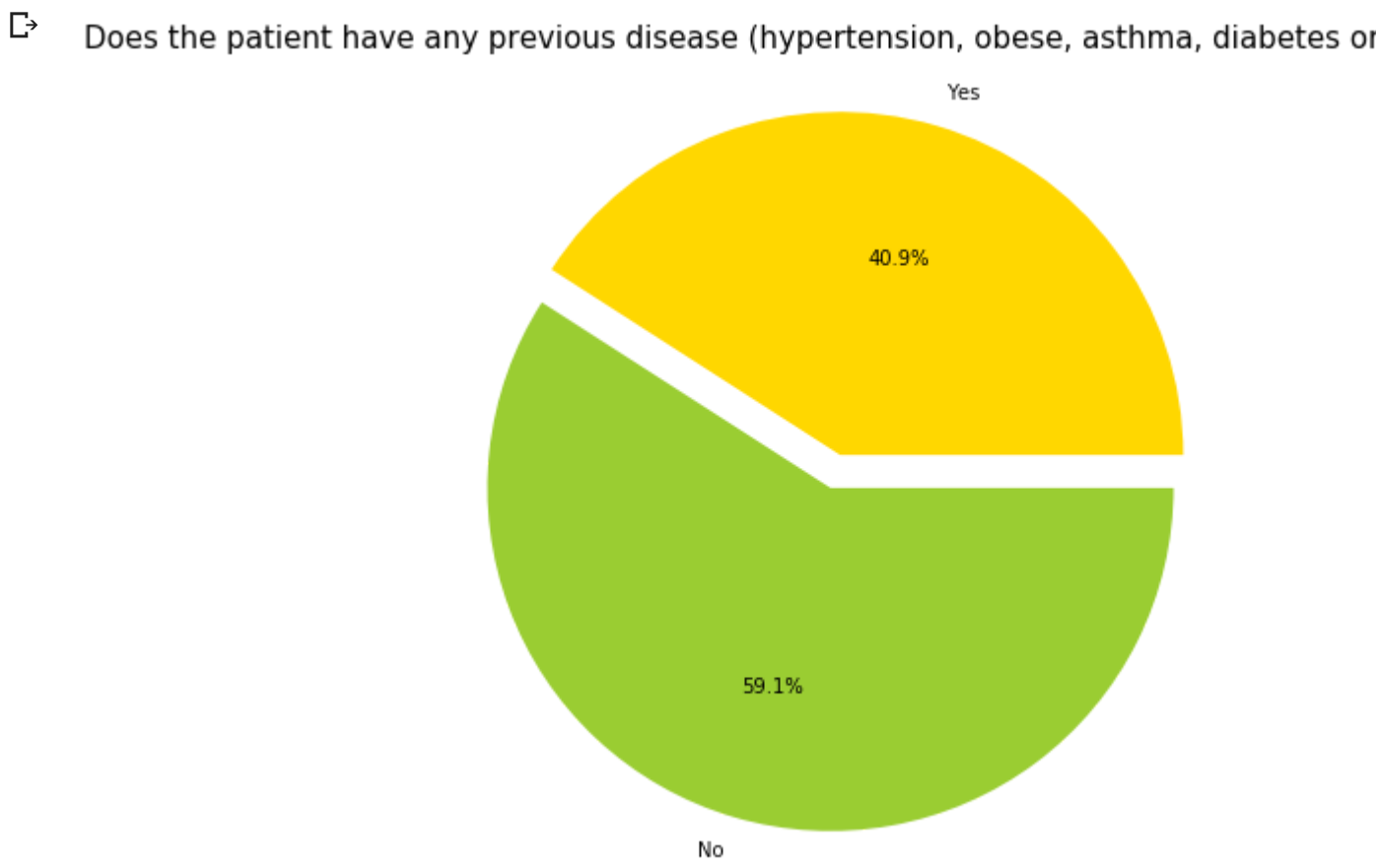
This variable will summarize all patients with some disease diagnosed: hypertension, diabetes or alcoholism.

```
df['PreviousDisease'] = df.apply(lambda x: ((x.HypertensionFLG == 1 )| x.AsthmaFLG == 1 | x.H
```

```
df['PreviousDisease'].value_counts()
```

```
0    56271
1    38950
Name: PreviousDisease, dtype: int64
```

```
plt.pie([df['PreviousDisease'].value_counts()[1] ,df['PreviousDisease'].value_counts()[0] ],
        explode = (0.1, 0), labels = ['Yes', 'No'], colors = colors, autopct='%1.1f%%')

plt.title('Does the patient have any previous disease (hypertension, obese, asthma, diabetes
plt.gcf().set_size_inches(8, 8)
plt.show()
```



The reason to create this variable is to check if patients with diseases (no matter which) have similar behavior regarding medical appointment no-show.

f) DaysBeforeApp

Indicates the number of days between scheduled day and appointment day

```
df['DaysBeforeApp'] = df['ApptLagNBR']
```

```
df['DaysBeforeApp'].value_counts()
```

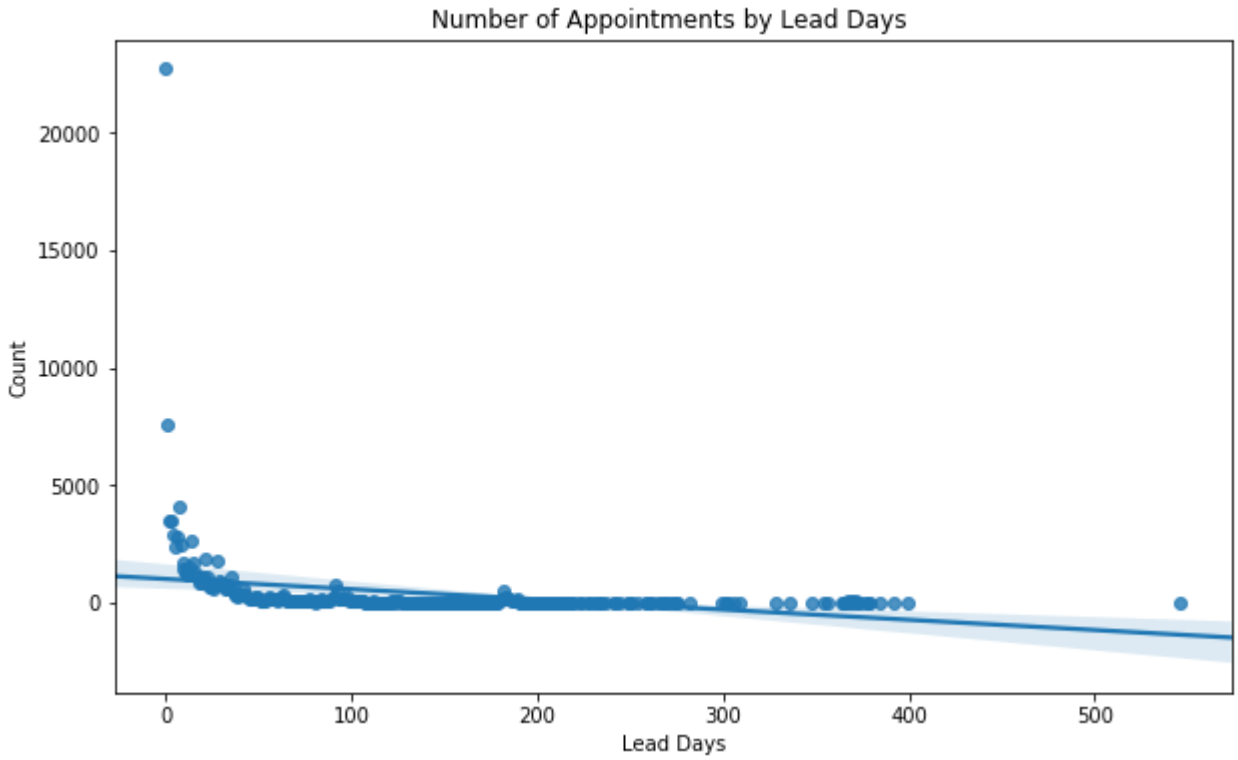
```
0      22713
1       7550
7       4038
2       3491
3       3439
4       2886
6       2794
14      2644
8       2447
5       2409
21      1814
28      1757
9       1716
15      1665
13      1506
10      1389
16      1245
11      1240
12      1152
20      1110
22      1105
35      1097
17      1081
29       899
19       859
18       859
30       824
23       807
91       771
27       761
       ...
234        1
399        1
546        1
205        1
235        1
299        1
384        1
269        1
207        1
259        1
241        1
354        1
274        1
215        1
244        1
309        1
276        1
377        1
379        1
374        1
```

Most of appointments where scheduled less than one day in advance.

```
348        1
```

```
days_before = df.groupby(['DaysBeforeApp'])[['NoShowFLG']].count()
days_before.reset_index(inplace = True)
days_before.columns = ['Days Ahead', 'Count']
```

```
sns.regplot(x = 'Days Ahead', y = 'Count', data = days_before)
plt.title('Number of Appointments by Lead Days ')
plt.xlabel('Lead Days')
#plt.xlim('2016-04-28', '2016-06-09')
plt.gcf().set_size_inches(10, 6)
plt.show()
```



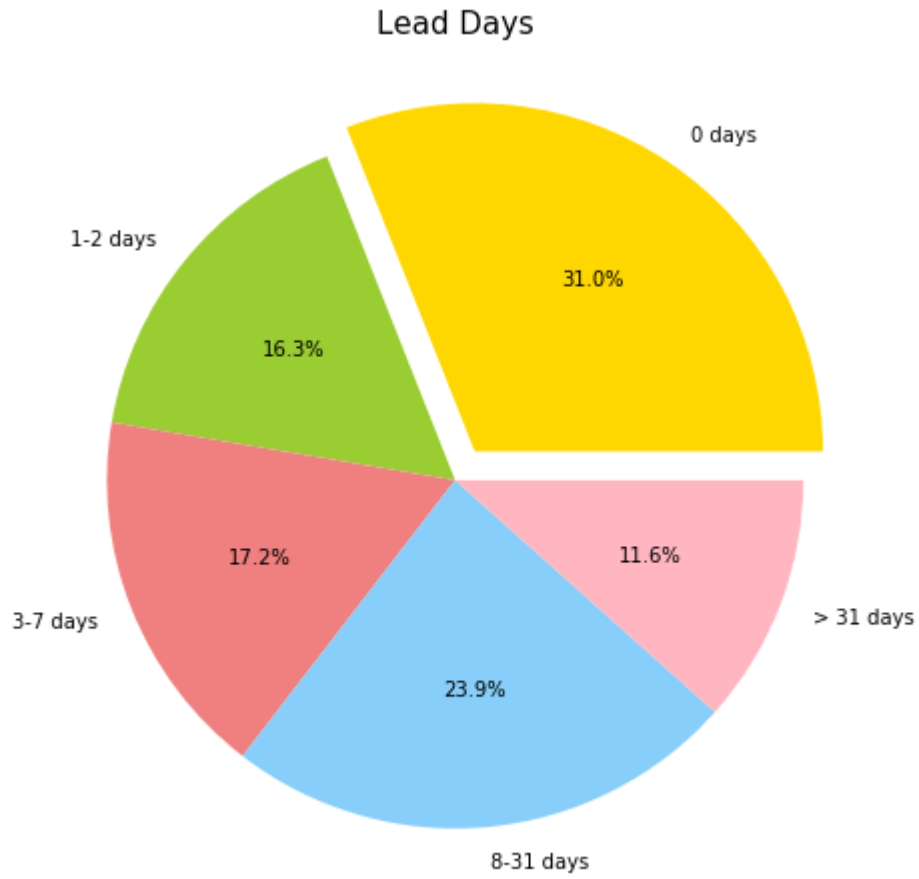
Most appointments are scheduled less than a day in advance. We will categorize this variable to group similar situations (this will implicate that, for the model, one hot encoding will be necessary in order to use the categorized variable).

```
def DaysBeforeCat(days):
    if days == 0:
        return '0 days'
    elif days in range(1,3):
        return '1-2 days'
    elif days in range(3,8):
        return '3-7 days'
    elif days in range(8, 32):
        return '8-31 days'
    else:
        return '> 31 days'

df['DaysBeforeCat'] = df.DaysBeforeApp.apply(DaysBeforeCat)
```

```
df['DaysBeforeCat'].value_counts()
```

```
8-31 days     29516
0 days        22713
> 31 days     16385
3-7 days      15566
1-2 days      11041
```

```
colors = ['gold', 'yellowgreen', 'lightcoral', 'lightskyblue', 'lightpink']

plt.pie([df['DaysBeforeCat'].value_counts()[0], df['DaysBeforeCat'].value_counts()[3],
         df['DaysBeforeCat'].value_counts()[2], df['DaysBeforeCat'].value_counts()[1], df['Da
        labels = ['0 days','1-2 days' ,'3-7 days','8-31 days','> 31 days'],
        explode = (0.1, 0, 0, 0, 0),
        colors = colors, autopct='%1.1f%%')

plt.title('Lead Days', fontsize=15)
plt.gcf().set_size_inches(8, 8)
plt.show()
```



g) No-show

For XGBoost, it's very important to know the proportion of yes/no in the sample, as the parameter scale_pos_weight allows to work with unbalanced data without sampling or discarding observations

```
df['NoShowFLG'].value_counts()[1]
```

```
9106
```

```
ns = df['NoShowFLG'].value_counts()[1]
```

```
show = df['NoShowFLG'].value_counts()[0]
rate = (show + 0.0) / ns
print('For every no-show, there are {:1.2f} shows'.format(rate))
```

```
For every no-show, there are 9.46 shows
```

We have approximately 9 shows for every no-show.

2. Consistency Check

We will check a couple of "common sense" rules regarding the data we have:

Medical conditions (such as hypertension, diabetes and asthma) should have a unique value per patient and gender should have a unique value per patient

```
def unique_condition(df, var, cols):
    if df.groupby(cols).ngroups == df[var].unique().size:
        return 'Sizes match: unique value per ' + var
    else:
        return 'Mismatch: more than one value per ' + var

unique_condition(df, 'PatientID', ['PatientID','HypertensionFLG', 'AsthmaFLG',
                                   'HeartDiseaseFLG','ObeseFLG','DiabetesFLG', 'SexFLG', 'Hi
```

```
'Sizes match: unique value per PatientID'
```

Last, but not least, we'll check that ages do not differ in more than a year per patient:

```
inconsist = []
for num in df['PatientID'].unique():
    ages = df[df['PatientID'] == num]['AgeNBR'].unique()
    if ages.size == 1:
        break
    if ages.size > 2:
        inconsist.append(num)
        print('Patient ' + str(num)+ 'has age inconsistency')
    else:
        if abs(ages[0]-ages[1]) > 1:
            inconsist.append(num)
            print('Patient ' + str(num)+ 'has age inconsistency')

if len(inconsist) == 0:
    print('There is no inconsistency in ages')
```

```
There is no inconsistency in ages
```

```
df.head()
```

	Unnamed: 0	AppointmentID	PatientID	ClinicNM	AppointmentDTS	AppointmentMonthNBR
0	1	21725	1	E	2018-04-10	4
1	2	11206	2	A	2018-02-07	2
2	3	12548	2	A	2018-02-08	2

```
df3 = df

df3.tail()
```

	Unnamed: 0	AppointmentID	PatientID	ClinicNM	AppointmentDTS	AppointmentMonthN
95216	95217	20483	33472	A	2018-03-21	
95217	95218	81514	33472	A	2018-10-19	
95218	95219	81507	33472	A	2018-10-18	
95219	95220	1763	33473	A	2018-06-05	
95220	95221	46027	33473	A	2018-12-18	

5 rows × 61 columns

3. Bivariate Analysis

So far we have managed to collect and analyze the following features: Gender, ScheduledDay, AppointmentDay, Age, Hypertension, Diabetes, Alcoholism, email received, PreviousApp, PreviousNoShow, WeekdayScheduled, WeekdayAppointment, PreviousDisease, DaysBeforeApp, DaysBeforeCat.

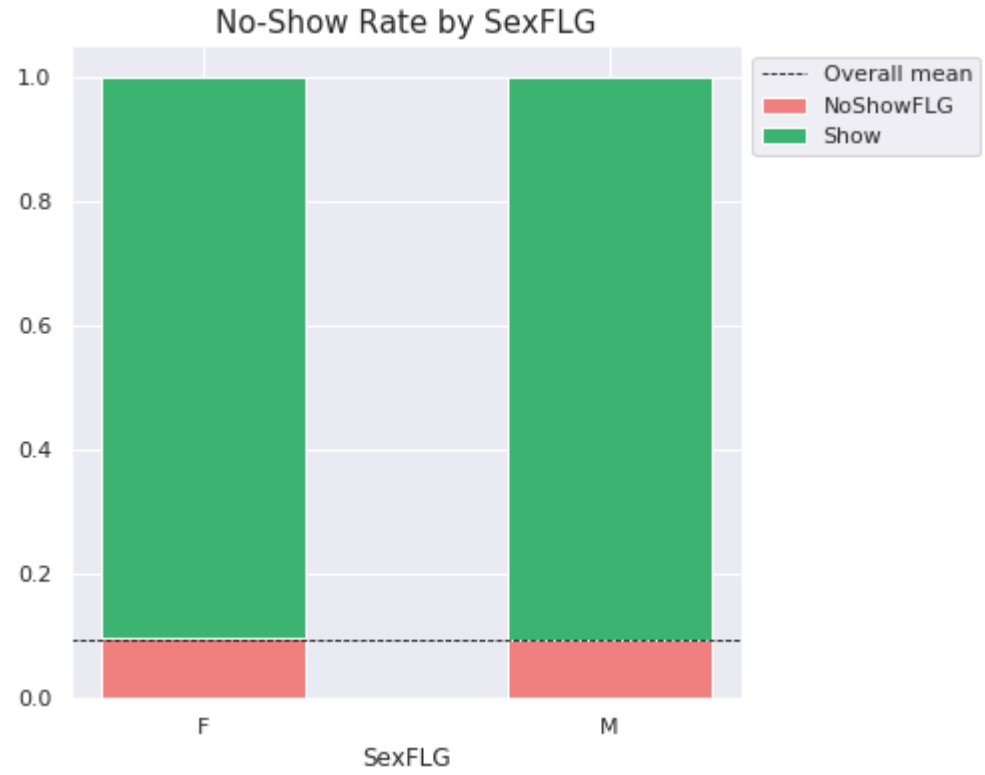
Based on previous analysis, we've discarded ScheduledDay, AppointmentDay (we are going to use the weekday of both) and DaysBeforeApp (as we categorized them). Now we will study each feature and its relation with No-Show:

a) Gender

```
sns.set()

def cat_var(df3, var):

    print(df3.groupby([var])['NoShowFLG'].mean())

    ns_rate = [df3.groupby([var])['NoShowFLG'].mean()[i] for i in df3[var].unique()]
    s_rate = [1-df3.groupby([var])['NoShowFLG'].mean()[i] for i in df3[var].unique()]
    barWidth = 0.5

    plt.bar(df3[var].unique(), ns_rate, color='lightcoral', edgecolor='white', width=barWidth
    plt.bar(df3[var].unique(), s_rate, bottom=ns_rate, color='mediumseagreen', edgecolor='whi
    plt.axhline(y=df3['NoShowFLG'].mean(), color='black', linewidth= 0.8, linestyle='--', lab
    plt.xticks(df3[var].unique())
    plt.xlabel(var)
    plt.legend(loc='upper left', bbox_to_anchor=(1,1), ncol=1)
    plt.title('No-Show Rate by '+ var, fontsize=15)
```

```
    plt.gcf().set_size_inches(6, 6)
    plt.show()

    counts = np.array(df3.groupby([var])['NoShowFLG'].sum())
    nobs = np.array(df3.groupby([var])['NoShowFLG'].count())

    table = df3.groupby(['NoShowFLG', var]).size().unstack(var)
    pvalue = ss.chi2_contingency(table.fillna(0))[1]

    print('Means test p-value: {:1.3f}'.format(pvalue))
    if pvalue < 0.05:
        print('Reject null hypothesis: no-show rate is different for at least one group')
    else:
        print('Cannot reject no-show rates are same for all groups')


cat_var(df3, 'SexFLG')
```

```
SexFLG
F    0.097126
M    0.093392
Name: NoShowFLG, dtype: float64
```



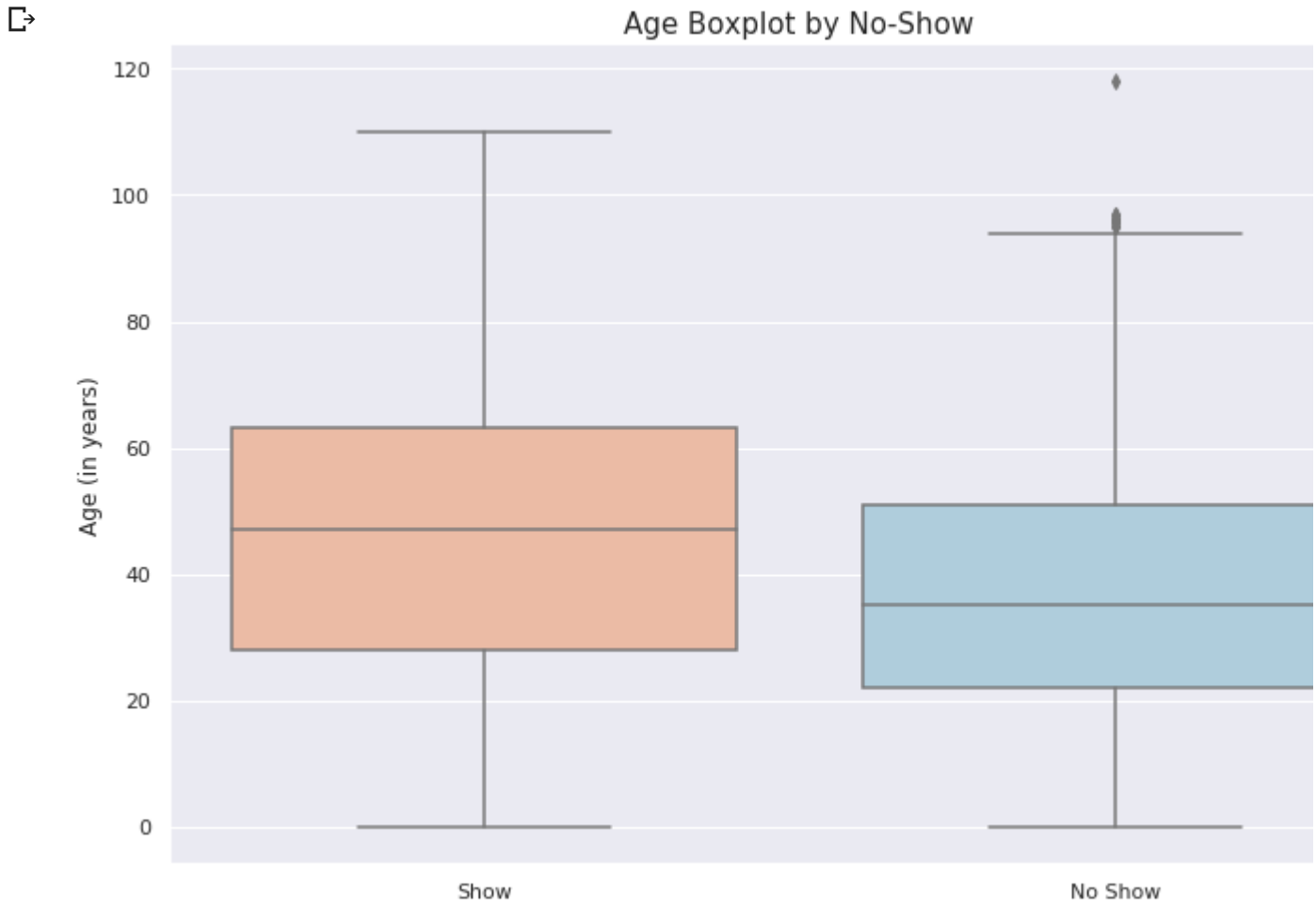

```
Means test p-value: 0.056
Cannot reject no-show rates are same for all groups
```

Based on the analysis, we cannot say that no-show rates are different for men and women, which could indicate that the variable Gender is not a relevant one when predicting no-show. We shall include the variable as well because, interacting with others, Gender could gain relevance and became a good discriminator.

b) Age

```
ax = sns.boxplot(x="NoShowFLG", y="AgeNBR",data= df3, palette = 'RdBu')
ax.set_xticklabels(['Show', 'No Show'])
plt.xlabel(' ')
```

```
plt.ylabel('Age (in years)')
plt.title('Age Boxplot by No-Show', fontsize = 15)
plt.gcf().set_size_inches(12, 8)
plt.show()
```



From the boxplot, we can see that people who don't show up to appointments tend to be younger than those who attend (based on quartiles).

```
print('Correlation with No-Show: %.3f' % ss.pointbiserialr(df3['NoShowFLG'], df3['AgeNBR'])[0
```

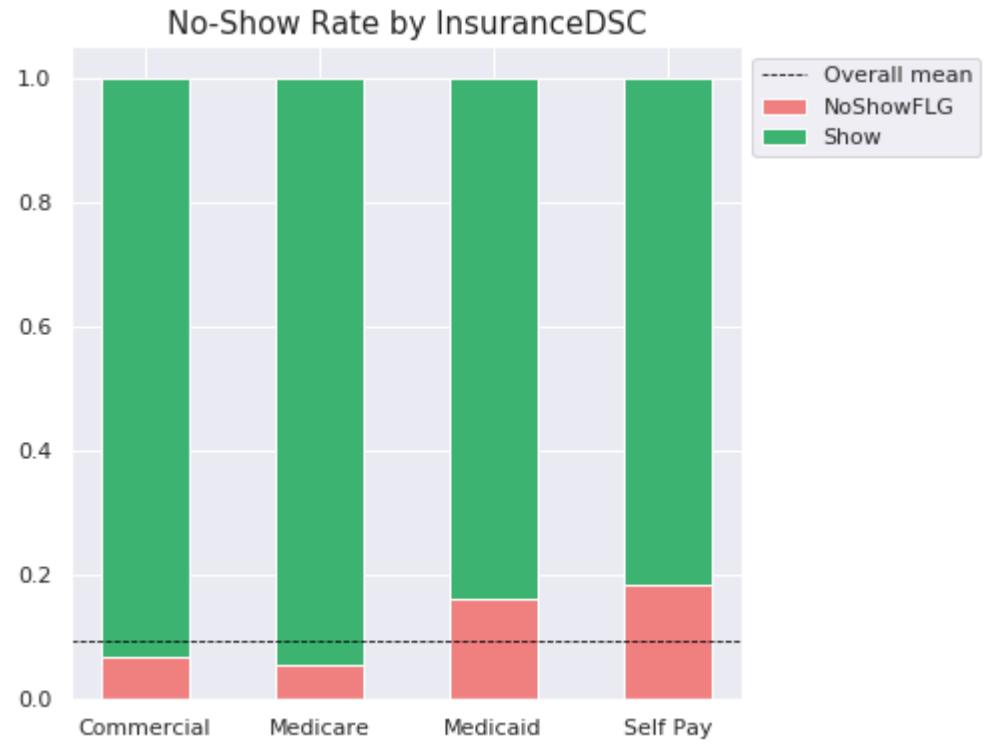
```
Correlation with No-Show: -0.109
```

Correlation with no-show is very close to zero,which could indicate that the variable is not of much interest when predicting no-show.

c) InsuranceDSC

```
cat_var(df3, 'InsuranceDSC')
```

```
InsuranceDSC
Commercial    0.067083
Medicaid      0.161370
Medicare      0.055891
Self Pay      0.185075
Name: NoShowFLG, dtype: float64
```

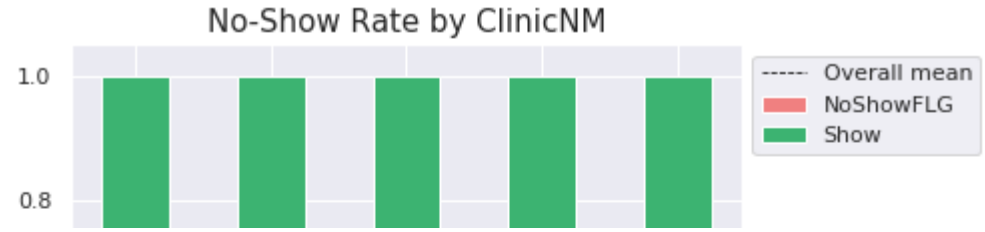


From the above, we can assure that patients who are part of the Medicaid and self pay insurance program have significantly higher no-show rates than peopleusing commercial and medicare. This could mean that the variable is very important for predicting no-show.

d) Clinic

```
cat_var(df3, 'ClinicNM')
```

```
ClinicNM
A    0.045355
B    0.121507
C    0.118508
D    0.066365
E    0.128149
Name: NoShowFLG, dtype: float64
```



As before, we can reject that people with various clinics present same no-show rates. This difference is significant, which indicates that the variable is of interest for predicting no-show. Clinic A and D tend to have lower no-show rates (maybe because they are in some kind of treatment...)

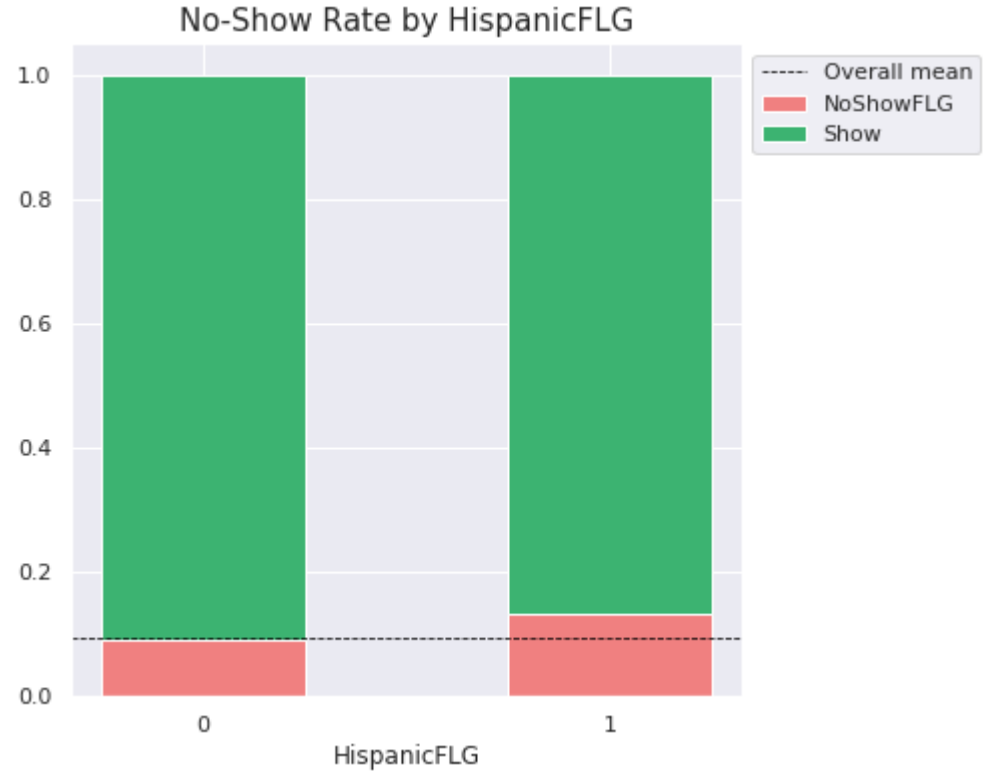
e) Appointment month


```
cat_var(df3, 'AppointmentMonthNBR')
```

```
AppointmentMonthNBR
1     0.089944
2     0.084698
3     0.089242
4     0.093549
5     0.087766
6     0.097049
```

```
cat_var(df3, 'HispanicFLG')
```

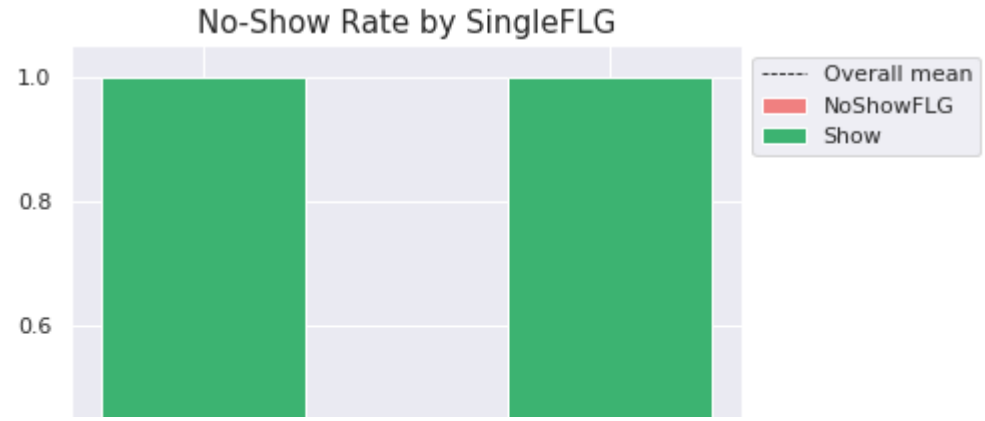
```
HispanicFLG
0    0.090218
1    0.132004
Name: NoShowFLG, dtype: float64
```




```
Means test p-value: 0.000
Reject null hypothesis: no-show rate is different for at least one group
```

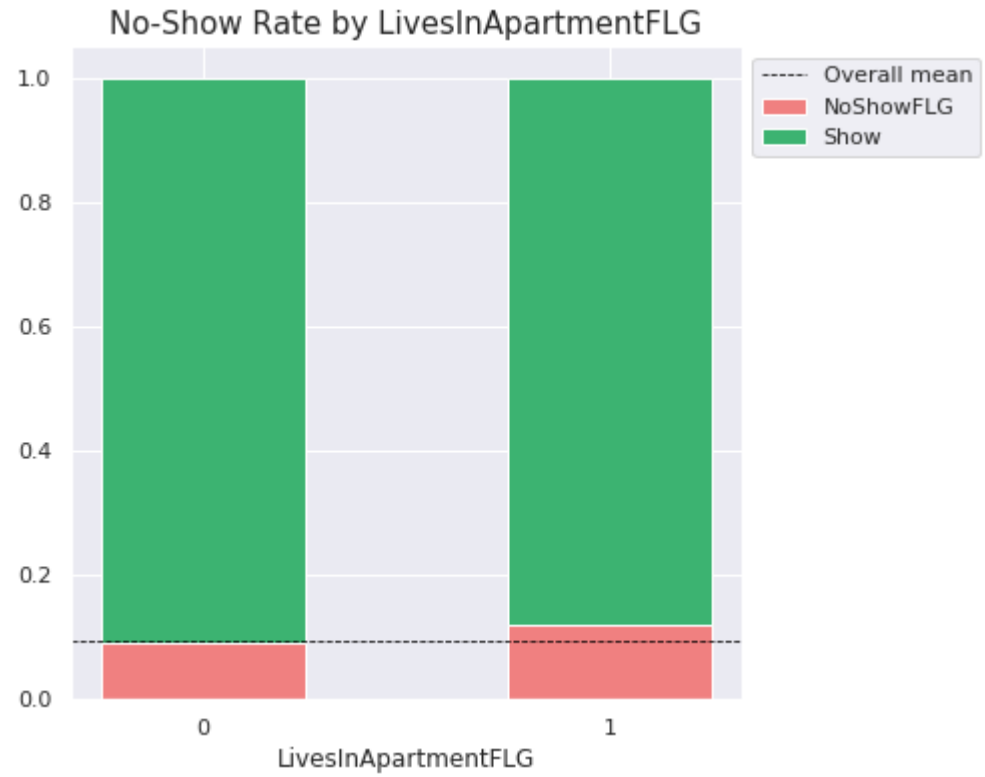
```
cat_var(df3, 'SingleFLG')
```

```
SingleFLG
0    0.069432
1    0.116751
Name: NoShowFLG, dtype: float64
```




```
cat_var(df3, 'LivesInApartmentFLG')
```

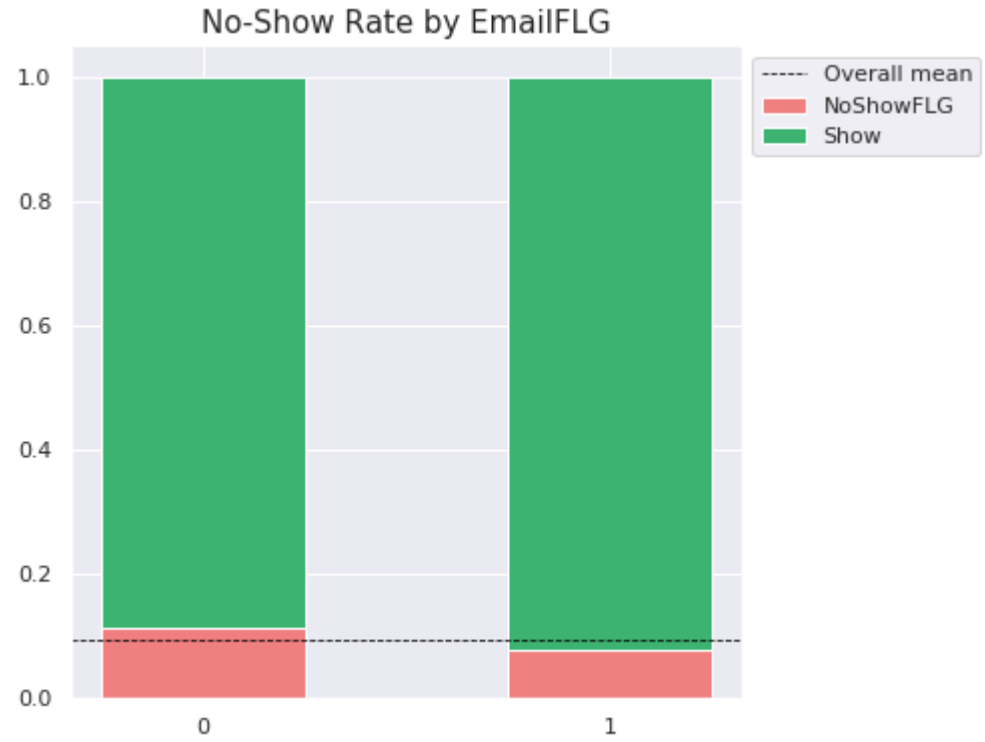
```
LivesInApartmentFLG
0    0.089958
1    0.118849
Name: NoShowFLG, dtype: float64
```




```
Means test p-value: 0.000
Reject null hypothesis: no-show rate is different for at least one group
```

```
cat_var(df3, 'EmailFLG')
```

```
EmailFLG
0    0.112856
1    0.077738
Name: NoShowFLG, dtype: float64
```

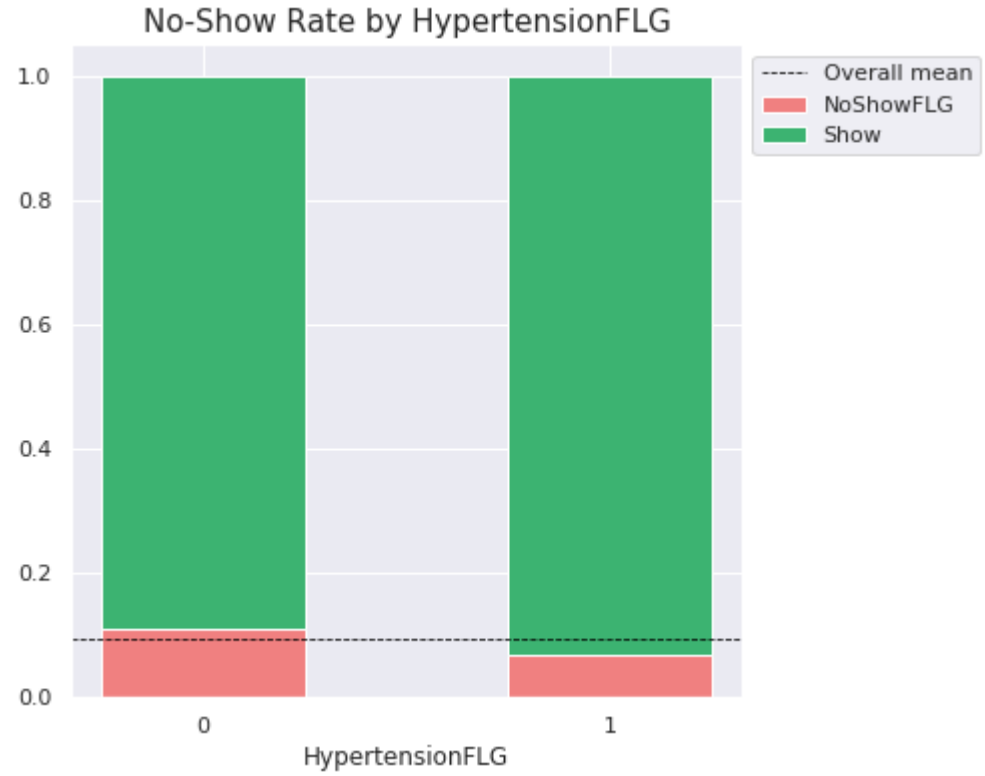



```
aux = df3.groupby(['DaysBeforeApp'])[['EmailFLG']].agg(['count','sum'])
aux.columns = ['count', 'EmailFLG']
aux[:20]['EmailFLG'].plot()
plt.gcf().set_size_inches(10, 6)
plt.xlabel('Anticipation Days')
plt.ylabel('Count (n)')
plt.title('Frequency of Days Before Appointment (anticipation)')
plt.xticks(range(0, 20))
plt.show()
```

Double-click (or enter) to edit


```
cat_var(df3, 'HypertensionFLG')
```

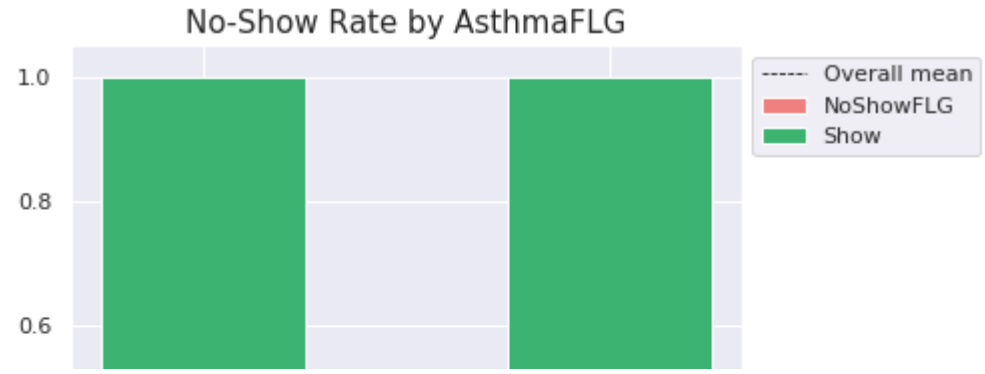
```
HypertensionFLG
0    0.110144
1    0.069595
Name: NoShowFLG, dtype: float64
```




```
Means test p-value: 0.000
Reject null hypothesis: no-show rate is different for at least one group
```

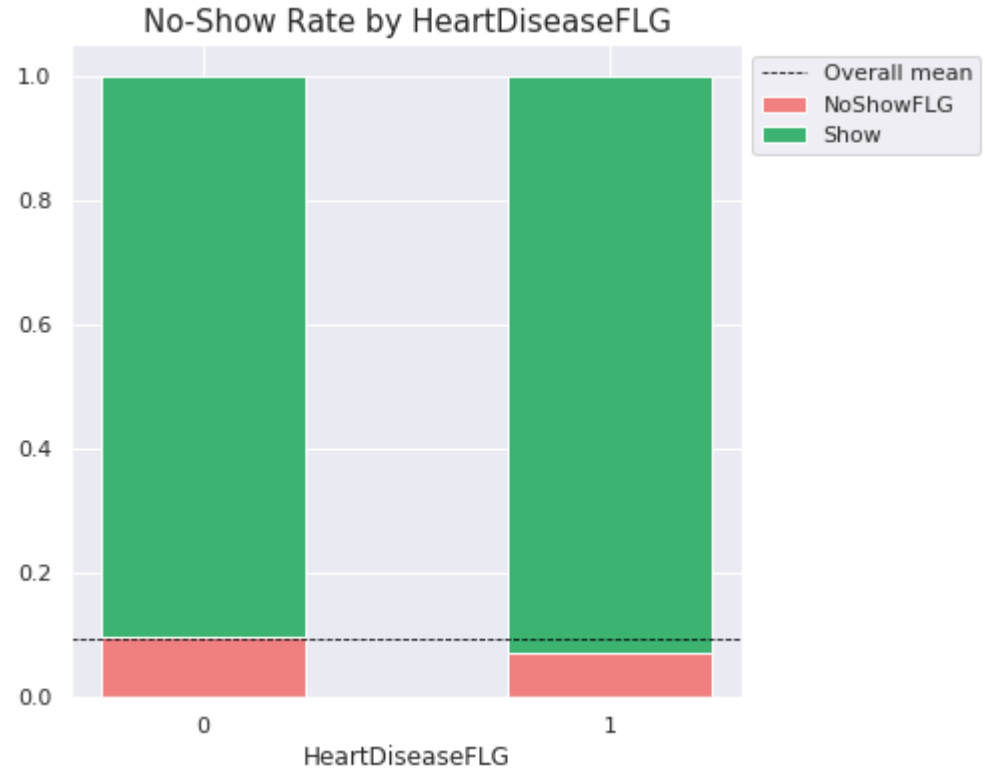
```
cat_var(df3, 'AsthmaFLG')
```

```
AsthmaFLG
0    0.095044
1    0.101271
Name: NoShowFLG, dtype: float64
```




```
cat_var(df3, 'HeartDiseaseFLG')
```

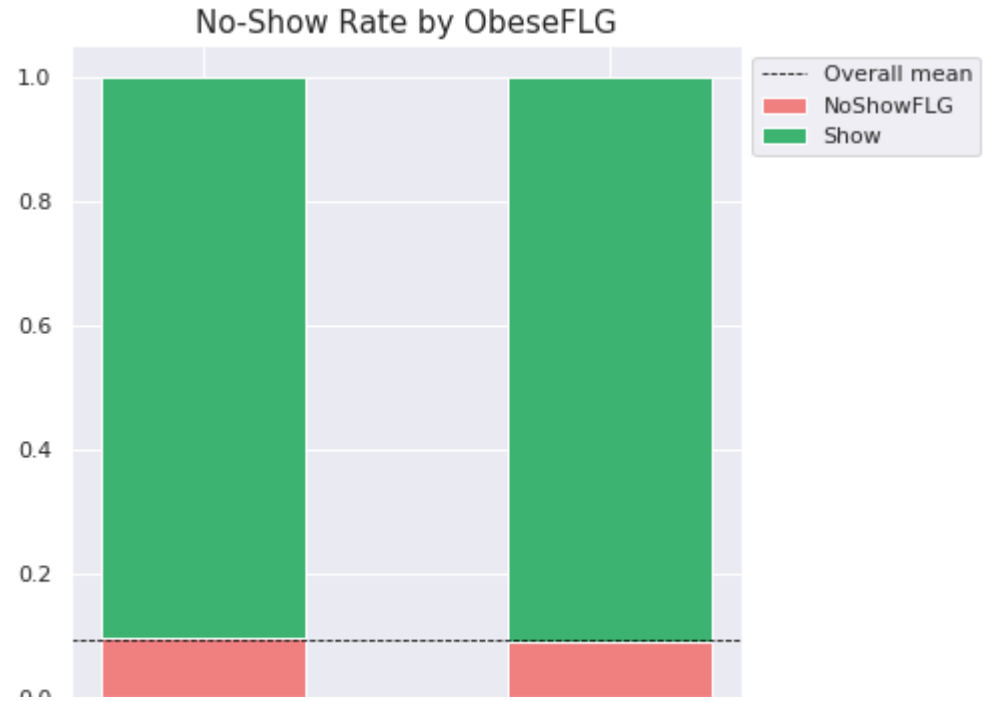
```
HeartDiseaseFLG
0    0.098536
1    0.071903
Name: NoShowFLG, dtype: float64
```




```
Means test p-value: 0.000
Reject null hypothesis: no-show rate is different for at least one group
```

```
cat_var(df3, 'ObeseFLG')
```

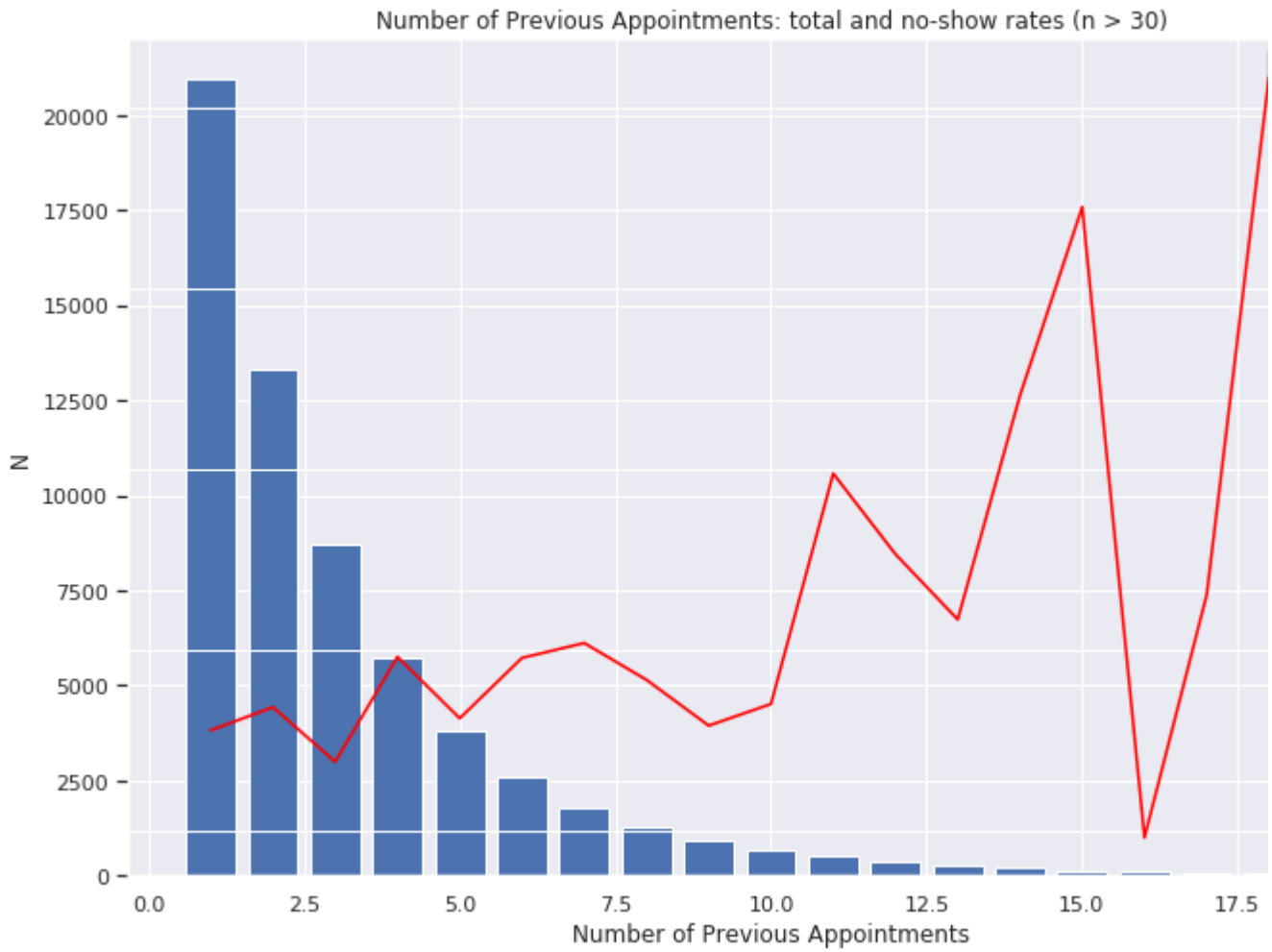
```
ObeseFLG
0    0.097687
1    0.089748
Name: NoShowFLG, dtype: float64
```




```
cat_var(df3, 'DiabetesFLG')
```

```
    DiabetesFLG
```

```
prevapp = df3.groupby(['PreviousApp'])[['NoShowFLG']].agg(['count', 'mean'])
prevapp.columns = ['count', 'NoShow_rate']
prevapp.reset_index(inplace = True)
import warnings
warnings.filterwarnings("ignore")
prevapp = prevapp[(prevapp['count'] > 30) & (prevapp['PreviousApp'] > 0)]
fig = plt.figure()

count = fig.add_subplot(111)
rate = count.twinx()

count.set_ylabel('N')
rate.set_ylabel('No-Show rate')

line1 = count.bar(prevapp['PreviousApp'], prevapp['count'])
line2 = rate.plot(prevapp['PreviousApp'], prevapp['NoShow_rate'], color = 'red', label = 'No-
count.legend([line1, line2], ['Count', 'No-show Rate'])
plt.gcf().set_size_inches(12, 8)
count.set_xlabel('Number of Previous Appointments')
plt.title('Number of Previous Appointments: total and no-show rates (n > 30)')
plt.show()
```



We decided to eliminate from the plot appointments without previous appointments, as they are a much higher number than other categories and makes the plot more confusing. From the above, and considering we are only observing groups with more than 30 observations, we can say that no show rate are higher for patients with 18 AND above previous appointments and then the no show rates descend drastically

(patients with lots of previous appointments are likely to be in a treatment as the data is condensed in 41 days).

```
print('Correlation with No-Show (all appointments): %.3f' % ss.pointbiserialr(df3['NoShowFLG'
print('Correlation with No-Show (1 or more previous app): %.3f' % ss.pointbiserialr(df3[df3['
```

```
Correlation with No-Show (all appointments): 0.002
Correlation with No-Show (1 or more previous app): 0.011
```

The point biserial correlation is 0.002 and if we calculate the correlation without first-time-appointments, the value is closer to 1, indicating a stronger negative correlation.

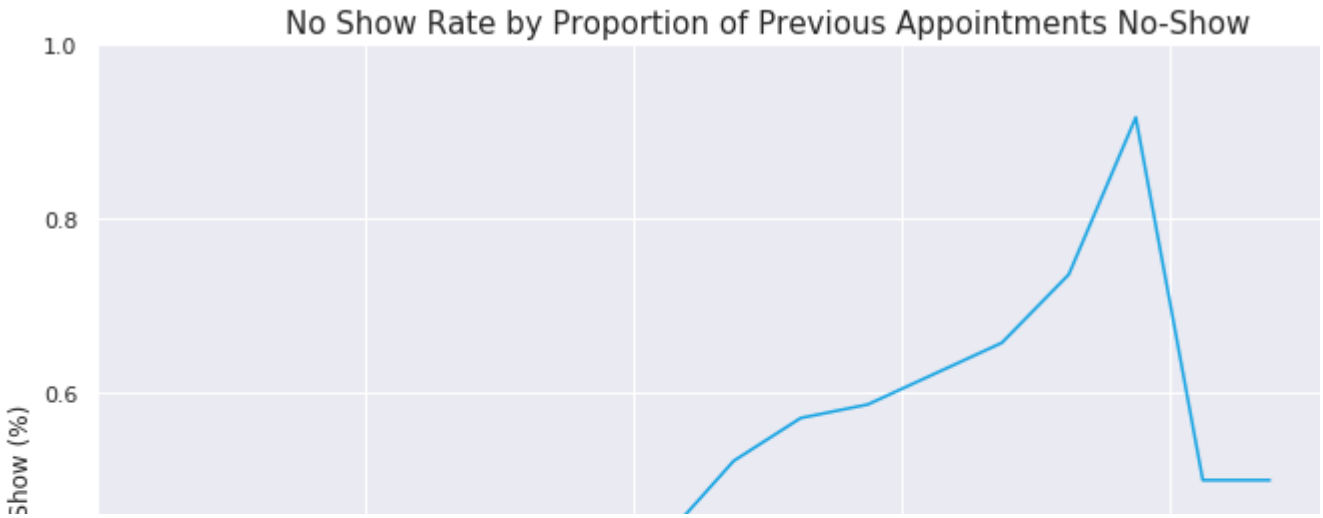
i) PreviousNoShow

We will study no-show rates, grouping appointments by PreviousNoShow deciles.

```
prop_ns = df3.groupby(pd.cut(df3['PreviousNoShow'], np.arange(0, 1.05, 0.05), include_lowest =
prop_ns = prop_ns.reset_index()
prop_ns['middle'] = np.arange(0.025, 1.025, 0.05)
prop_ns.iloc[0,2] = 0
prop_ns.iloc[19,2] = 1
```

```
no_na = df3.dropna(subset = ['PreviousNoShow'])
```

```
prop_ns = prop_ns.drop([12], axis = 0)
plt.plot(prop_ns['middle'], prop_ns['NoShowFLG'], color = '#16a4e3')

plt.xlim(0,1)
plt.ylim(0,1)
plt.xlabel('Previous Appointments No-Show Rate', labelpad=10)
plt.ylabel('No-Show (%)')
plt.grid(True)
plt.gcf().set_size_inches(12, 8)
plt.title('No Show Rate by Proportion of Previous Appointments No-Show', fontsize = 15)

plt.show()

print('Correlation with No-Show: %.3f' % ss.pointbiserialr(no_na['NoShowFLG'], no_na['Previous
```



As expected, previous patient behavior and no-show rate have an almost perfect linear relation. This supports the hypothesis that people tend to mantain certain behaviors in time. Moreover, the correlation between the variables is very close to 1, indicating a strong and positive correlation between previous no show and appointment noshow. For this, pevious behavior is a variable of interest for our model.

j) WeekdayScheduled

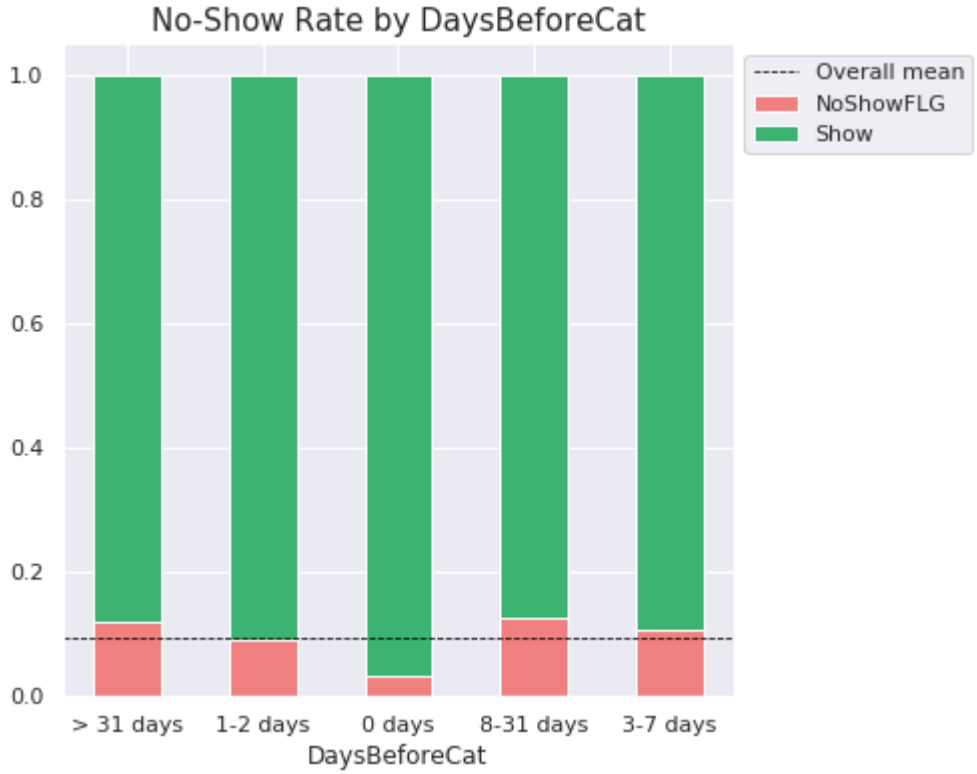
```
cat_var(df3, 'AppointmentWeekdayNBR')
```

```
AppointmentWeekdayNBR
1    0.540541
2    0.101066
3    0.093653
```

k) Days before CAT

```
cat_var(df3, 'DaysBeforeCat')
```

```
DaysBeforeCat
0 days       0.031964
1-2 days     0.089847
3-7 days     0.108506
8-31 days    0.125864
> 31 days    0.121086
Name: NoShowFLG, dtype: float64
```




```
Means test p-value: 0.000
Reject null hypothesis: no-show rate is different for at least one group
```

```
df3.head()
```

	Unnamed: 0	AppointmentID	PatientID	ClinicNM	AppointmentDTS	AppointmentMonthNBR
0	1	21725	1	E	2018-04-10	4

```
df3.head()
```

	Unnamed: 0	AppointmentID	PatientID	ClinicNM	AppointmentDTS	AppointmentMonthNBR
0	1	21725	1	E	2018-04-10	4
1	2	11206	2	A	2018-02-07	2
2	3	12548	2	A	2018-02-08	2
3	4	12727	2	A	2018-03-08	3
4	5	86882	2	A	2018-11-09	11

5 rows × 61 columns

From the plot, it is clear that no-show rates are very different in each group: moreover, rates seem to be increasing as the number of days of anticipation is higher. For appointments scheduled in the same day (0 days of anticipation), no-show rates are dramatically lower and very close to zero (only 4.6%). This is clearly evidence enough to consider the variable as highly interesting and important to predict no-show.

```
df_done=df3
```

```
df_done.tail()
```

	Unnamed: 0	AppointmentID	PatientID	ClinicNM	AppointmentDTS	AppointmentMonthN
95216	95217	20483	33472	A	2018-03-21	
95217	95218	81514	33472	A	2018-10-19	
95218	95219	81507	33472	A	2018-10-18	
95219	95220	1763	33473	A	2018-06-05	
95220	95221	46027	33473	A	2018-12-18	

5 rows × 61 columns

As XGBoost only receives numerical features, we must transform the categorical variables into dummy variables. We could change each variable to a number but this makes no sense as, for example, Mondays (1 in WeekdayAppointment) is not twice as Tuesdays (2 in WeekdayAppointment). Values must make sense. Given this, the transformations we will apply are:

Change Gender to "IsFemale" which takes the value 1 if Gender = F and 0 in other case Change WeekdayScheduled to five variables: ScheduledMonday, ScheduledTuesday, ..., ScheduledFriday. Each variable is 1 if the appointment was scheduled in that particular day and 0 in other case. Same transformation will be apply to WeekdayAppointment (variables will be AppointmentMonday and so on) Change DaysBeforeCat to five variables

```
# WeekdayAppointment to dummies
df_done = df_done.assign(AppointmentSunday = (df['AppointmentWeekdayNBR'] == 1)*1,
                         AppointmentMonday = (df['AppointmentWeekdayNBR'] == 2)*1,
                         AppointmentTuesday = (df['AppointmentWeekdayNBR'] == 3)*1,
                         AppointmentWednesday = (df['AppointmentWeekdayNBR'] == 4)*1,
                          AppointmentSaturday = (df['AppointmentWeekdayNBR'] == 7)*1,
                         AppointmentThursday = (df['AppointmentWeekdayNBR'] == 5)*1,
                         AppointmentFriday = (df['AppointmentWeekdayNBR'] == 6)*1)

# Gender to dummy
df_done['IsFemale'] = (df_done['SexFLG'] == 'F')*1

# DaysBeforeCat to dummies
def ant_days(df):
    df.loc[:, 'Ant0Days'] = (df['DaysBeforeCat'] == '0 days')*1
    df.loc[:, 'Ant12Days'] = (df['DaysBeforeCat'] == '1-2 days')*1
    df.loc[:, 'Ant37Days'] = (df['DaysBeforeCat'] == '3-7 days')*1
    df.loc[:, 'Ant831Days'] = (df['DaysBeforeCat'] == '8-31 days')*1
    df.loc[:, 'Ant32Days'] = (df['DaysBeforeCat'] == '> 31 days')*1

ant_days(df_done)
```

```
df_done.columns
```

```
Index(['Unnamed: 0', 'AppointmentID', 'PatientID', 'ClinicNM',
       'AppointmentDTS', 'AppointmentMonthNBR', 'AppointmentWeekdayNBR',
       'AppointmentHourNBR', 'AgeNBR', 'SexFLG', 'HispanicFLG', 'SingleFLG',
       'LivesInApartmentFLG', 'EmailFLG', 'ApptLagNBR', 'InsuranceDSC',
       'HypertensionFLG', 'AsthmaFLG', 'HeartDiseaseFLG', 'ObeseFLG',
       'DiabetesFLG', 'Noshow24NBR', 'CancellationsNBR', 'Latearrivals24NBR',
       'CheckintoCheckoutNBR', 'AppttoCheckoutNBR', 'CheckintoApptNBR',
       'Arrived24NBR', 'Providers24CNT', 'ThatProvider24NBR',
       'NoshowRate24NBR', 'EdVisitsNBR', 'IpVisitsNBR', 'NoShowFLG',
       'CancelledLateFLG', 'NewPatient', 'Cost', 'Rand', 'Revenue', 'Profit',
       'Age_bin', 'ClinicNM_num', 'SexFLG_num', 'InsuranceDSC_num',
       'Noshow24NBR_Bin', 'CheckintoCheckoutNBR_Bin', 'CheckintoApptNBR_bin',
       'Arrived24NBR_bin', 'Providers24CNT_bin', 'ThatProvider24NBR_bin',
       'NoshowRate24NBR_bin', 'CancellationsNBR_Bin', 'AppttoCheckoutNBR_Bin',
       'Latearrivals24NBR_Bin', 'PreviousApp', 'PreviousNoShow',
       'AppointmentDTS2', 'WeekdayScheduled', 'PreviousDisease',
       'DaysBeforeApp', 'DaysBeforeCat', 'AppointmentSunday',
       'AppointmentMonday', 'AppointmentTuesday', 'AppointmentWednesday',
       'AppointmentSaturday', 'AppointmentThursday', 'AppointmentFriday',
       'IsFemale', 'Ant0Days', 'Ant12Days', 'Ant37Days', 'Ant831Days',
       'Ant32Days'],
      dtype='object')
```

```
# Clinic to dummies
```

```
def ClinicNM(df):
    df.loc[:, 'ClinicNM_A'] = (df['ClinicNM'] == 'A')*1
    df.loc[:, 'ClinicNM_B'] = (df['ClinicNM'] == 'B')*1
    df.loc[:, 'ClinicNM_C'] = (df['ClinicNM'] == 'C')*1
    df.loc[:, 'ClinicNM_D'] = (df['ClinicNM'] == 'D')*1
    df.loc[:, 'ClinicNM_E'] = (df['ClinicNM'] == 'E')*1

ClinicNM(df_done)

# InsuranceDSC to dummies
def InsuranceDSC(df):
    df.loc[:, 'DSCCommercial'] = (df['InsuranceDSC'] == 'Commercial')*1
    df.loc[:, 'DSCMedicare'] = (df['InsuranceDSC'] == 'Medicare')*1
    df.loc[:, 'DSCSelf_Pay'] = (df['InsuranceDSC'] == 'Self Pay')*1
    df.loc[:, 'DSCMedicaid'] = (df['InsuranceDSC'] == 'Medicaid')*1

InsuranceDSC(df_done)




df_done2=df_done
df_done=df_done2



df_done.dtypes
```

```
Unnamed: 0                      int64
AppointmentID                   int64
PatientID                       int64
ClinicNM                       object
AppointmentDTS           datetime64[ns]
AppointmentMonthNBR             int64
AppointmentWeekdayNBR           int64
AppointmentHourNBR              int64
AgeNBR                          int64
SexFLG                         object
HispanicFLG                     int64
SingleFLG                       int64
LivesInApartmentFLG             int64
EmailFLG                        int64
ApptLagNBR                      int64
InsuranceDSC                   object
HypertensionFLG                 int64
AsthmaFLG                       int64
HeartDiseaseFLG                 int64
ObeseFLG                        int64
DiabetesFLG                     int64
Noshow24NBR                     int64
CancellationsNBR                int64
Latearrivals24NBR               int64
CheckintoCheckoutNBR            int64
AppttoCheckoutNBR               int64
CheckintoApptNBR                int64
Arrived24NBR                    int64
Providers24CNT                  int64
ThatProvider24NBR               int64
```

```
# Special thanks to https://www.kaggle.com/somang1418/tuning-hyperparameters-under-10-minutes
def reduce_mem_usage(df, verbose=True):
    numerics = ['int8','int16', 'int32', 'int64', 'float16', 'float32', 'float64']
    start_mem = df.memory_usage().sum() / 1024**2
    for col in df.columns:
        col_type = df[col].dtypes
        if col_type in numerics:
            c_min = df[col].min()
            c_max = df[col].max()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
                    df[col] = df[col].astype(np.int64)
                else:
                  if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float16).max:
                    df[col] = df[col].astype(np.float16)
                  elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float32).max:
                    df[col] = df[col].astype(np.float32)
                  else:
                    df[col] = df[col].astype(np.float64)
    end_mem = df.memory_usage().sum() / 1024**2
    if verbose: print('Mem. usage decreased to {:5.2f} Mb ({:.1f}% reduction)'.format(end_mem
    return df
```

```
ClinicNM_D                      int64
```

```
df_done.columns
```

```
Index(['Unnamed: 0', 'AppointmentID', 'PatientID', 'ClinicNM',
       'AppointmentDTS', 'AppointmentMonthNBR', 'AppointmentWeekdayNBR',
       'AppointmentHourNBR', 'AgeNBR', 'SexFLG', 'HispanicFLG', 'SingleFLG',
       'LivesInApartmentFLG', 'EmailFLG', 'ApptLagNBR', 'InsuranceDSC',
       'HypertensionFLG', 'AsthmaFLG', 'HeartDiseaseFLG', 'ObeseFLG',
       'DiabetesFLG', 'Noshow24NBR', 'CancellationsNBR', 'Latearrivals24NBR',
       'CheckintoCheckoutNBR', 'AppttoCheckoutNBR', 'CheckintoApptNBR',
       'Arrived24NBR', 'Providers24CNT', 'ThatProvider24NBR',
       'NoshowRate24NBR', 'EdVisitsNBR', 'IpVisitsNBR', 'NoShowFLG',
       'CancelledLateFLG', 'NewPatient', 'Cost', 'Rand', 'Revenue', 'Profit',
       'Age_bin', 'ClinicNM_num', 'SexFLG_num', 'InsuranceDSC_num',
       'Noshow24NBR_Bin', 'CheckintoCheckoutNBR_Bin', 'CheckintoApptNBR_bin',
       'Arrived24NBR_bin', 'Providers24CNT_bin', 'ThatProvider24NBR_bin',
       'NoshowRate24NBR_bin', 'CancellationsNBR_Bin', 'AppttoCheckoutNBR_Bin',
       'Latearrivals24NBR_Bin', 'PreviousApp', 'PreviousNoShow',
       'AppointmentDTS2', 'WeekdayScheduled', 'PreviousDisease',
       'DaysBeforeApp', 'DaysBeforeCat', 'AppointmentSunday',
       'AppointmentMonday', 'AppointmentTuesday', 'AppointmentWednesday',
       'AppointmentSaturday', 'AppointmentThursday', 'AppointmentFriday',
       'IsFemale', 'Ant0Days', 'Ant12Days', 'Ant37Days', 'Ant831Days',
       'Ant32Days', 'ClinicNM_A', 'ClinicNM_B', 'ClinicNM_C', 'ClinicNM_D',
       'ClinicNM_E', 'DSCCommercial', 'DSCMedicare', 'DSCSelf_Pay',
       'DSCMedicaid'],
      dtype='object')
```

```
features = ['AppointmentMonthNBR',
       'AppointmentWeekdayNBR', 'AppointmentHourNBR', 'AgeNBR', 'HispanicFLG',
       'SingleFLG', 'LivesInApartmentFLG', 'EmailFLG', 'ApptLagNBR',
       'HypertensionFLG', 'HeartDiseaseFLG', 'ObeseFLG', 'Noshow24NBR',
       'CancellationsNBR', 'Latearrivals24NBR', 'CheckintoCheckoutNBR',
       'AppttoCheckoutNBR', 'CheckintoApptNBR', 'Arrived24NBR',
       'Providers24CNT', 'ThatProvider24NBR', 'NoshowRate24NBR', 'EdVisitsNBR',
       'IpVisitsNBR', 'CancelledLateFLG', 'NewPatient',
       'PreviousApp', 'PreviousNoShow', 'WeekdayScheduled', 'PreviousDisease',
       'DaysBeforeApp', 'AppointmentSunday', 'AppointmentMonday',
       'AppointmentTuesday', 'AppointmentWednesday', 'AppointmentSaturday',
       'AppointmentThursday', 'AppointmentFriday', 'IsFemale', 'Ant0Days',
       'Ant12Days', 'Ant37Days', 'Ant831Days', 'Ant32Days', 'ClinicNM_A',
       'ClinicNM_B', 'ClinicNM_C', 'ClinicNM_D', 'ClinicNM_E', 'DSCCommercial',
       'DSCMedicare', 'DSCSelf_Pay', 'DSCMedicaid']

label = 'NoShowFLG'
```

```
!pip install scikit-optimize
!pip install shap
!pip install bayesian-optimization
import numpy as np
import pandas as pd
from scipy import stats as ss
import statsmodels.api as sm
from sklearn import metrics
from datetime import datetime
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
import xgboost as xgb
from xgboost.sklearn import XGBClassifier
from skopt import gp_minimize
```

```
from bayes_opt import BayesianOptimization

from skopt import gp_minimize

from skopt  import BayesSearchCV
from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

import shap
```

```
Requirement already satisfied: scikit-optimize in /usr/local/lib/python3.6/dist-packag
Requirement already satisfied: scikit-learn>=0.19.1 in /usr/local/lib/python3.6/dist-p
Requirement already satisfied: scipy>=0.14.0 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from s
Requirement already satisfied: shap in /usr/local/lib/python3.6/dist-packages (0.28.5)
```

```
df_done = reduce_mem_usage(df_done)

X_train, X_test, y_train, y_test = train_test_split(df_done[features], df_done[label], test_s
y_train = pd.DataFrame(y_train)
train = X_train.merge(y_train, left_index = True, right_index = True)
y_test = pd.DataFrame(y_test)
test = X_test.merge(y_test, left_index = True, right_index = True)
```

```
Mem. usage decreased to 20.80 Mb (65.5% reduction)
```

```
Requirement already satisfied: PyWavelets>=0.4.0 in /usr/local/lib/python3.6/dist-pack
```

```
def param_opt_xgb(X, y, init_round=10, opt_round=10, n_folds=3, random_seed=6, output_process
    # Prepare data
    dtest = xgb.DMatrix(X, y)

    def xgb_eval(learning_rate, n_estimators, max_depth, min_child_weight, gamma, subsample,
        params = {'objective' : 'binary:logistic', 'nthread' : 4, 'seed' : random_seed, "sile
        params['learning_rate'] = max(min(learning_rate, 1), 0)
        params['n_estimators'] = int(round(n_estimators))
        params['max_depth'] = int(round(max_depth))
        params['min_child_weight'] = int(round(min_child_weight))
        params['gamma'] = gamma
        params['subsample'] = max(min(subsample, 1), 0)
        params['colsample_bytree'] = max(min(colsample_bytree, 1), 0)
        params['scale_pos_weight'] = int(round(scale_pos_weight))
        cv_result = xgb.cv(params, dtest, nfold=n_folds, seed=random_seed, stratified=True, v


        return max(cv_result['train-auc-mean'])

    xgbBO = BayesianOptimization(xgb_eval, {'learning_rate': (0.01, 0.3),
                                            'n_estimators': (100, 200),
                                            'max_depth': (2, 7),
                                            'min_child_weight': (0, 7),
                                            'gamma': (0, 0.3),
                                            'subsample':(0.5,1),
                                            'colsample_bytree': (0.5, 1),
                                            'scale_pos_weight':(2,7)}, random_state=rando
    xgbBO.maximize(init_points=init_round, n_iter=opt_round)

    model_aucpr=[]
    for model in range(len(xgbBO.res)):
        model_aucpr.append(xgbBO.res[model]['target'])

    # return best parameters
    return xgbBO.res[pd.Series(model_aucpr).idxmax()]['target'],xgbBO.res[pd.Series(model_auc
```

```
Installing collected packages: bayesian-optimization
```

```
opt_params = param_opt_xgb(df_done[features], df_done[label])
```

```
[0]	train-auc:0.897703+0.0649485	test-auc:0.895355+0.0680459
[9]	train-auc:0.954216+0.000828263	test-auc:0.95162+0.000899979
| 2        | 0.9542   | 0.6677   | 0.1868   | 0.1371   | 5.679    | 3.626    |
[0]	train-auc:0.897294+0.0664883	test-auc:0.893688+0.0710744
[9]	train-auc:0.956313+0.00101683	test-auc:0.953738+0.00106234
| 3        | 0.9563   | 0.9099   | 0.124    | 0.2641   | 6.119    | 0.3813   |
[0]	train-auc:0.90431+0.0623652	test-auc:0.897929+0.067949
[9]	train-auc:0.954566+0.00170255	test-auc:0.950184+0.00313104
| 4        | 0.9546   | 0.8546   | 0.1623   | 0.0462   | 6.788    | 2.823    |
[0]	train-auc:0.896708+0.0648331	test-auc:0.893436+0.0679591
[9]	train-auc:0.954875+0.000741323	test-auc:0.95232+0.000961041
| 5        | 0.9549   | 0.6278   | 0.2014   | 0.1837   | 5.587    | 6.561    |
[0]	train-auc:0.902482+0.063869	test-auc:0.895867+0.0684311
[9]	train-auc:0.956647+0.000973105	test-auc:0.952209+0.00236523
| 6        | 0.9566   | 0.8733   | 0.2172   | 0.1278   | 6.947    | 3.153    |
[0]	train-auc:0.854718+0.0788496	test-auc:0.853269+0.0795739
[9]	train-auc:0.946186+0.000577766	test-auc:0.945732+0.00072916
| 7        | 0.9462   | 0.6992   | 0.2313   | 0.2318   | 3.419    | 1.327    |
[0]	train-auc:0.887899+0.0678021	test-auc:0.887362+0.0698105
[9]	train-auc:0.953876+0.000399907	test-auc:0.952064+0.00139341
| 8        | 0.9539   | 0.5946   | 0.09933  | 0.2548   | 5.075    | 6.197    |
[0]	train-auc:0.895358+0.0653016	test-auc:0.891945+0.0694658
[9]	train-auc:0.953903+0.000780027	test-auc:0.951448+0.00188921
| 9        | 0.9539   | 0.8092   | 0.1428   | 0.1272   | 5.709    | 5.782    |
[0]	train-auc:0.862114+0.0858447	test-auc:0.860607+0.0876803
[9]	train-auc:0.950476+0.00098263	test-auc:0.94959+0.00106204
| 10       | 0.9505   | 0.5294   | 0.0578   | 0.2782   | 4.026    | 1.038    |
[0]	train-auc:0.902564+0.061492	test-auc:0.897332+0.0670718
[9]	train-auc:0.953273+0.000602502	test-auc:0.948945+0.0019962
| 11       | 0.9536   | 0.6005   | 0.2967   | 0.01644  | 6.926    | 0.3362   |
[0]	train-auc:0.902544+0.0611294	test-auc:0.896829+0.0668943
[9]	train-auc:0.95577+0.0001008	test-auc:0.95127+0.00222168
| 12       | 0.9558   | 0.629    | 0.08302  | 0.111    | 6.846    | 2.197    |
[0]	train-auc:0.901735+0.063051	test-auc:0.896112+0.0684115
[9]	train-auc:0.957439+0.000382205	test-auc:0.951729+0.00153364
| 13       | 0.9574   | 0.5896   | 0.246    | 0.1765   | 6.916    | 0.06258  |
[0]	train-auc:0.888904+0.0809602	test-auc:0.877961+0.0952864
[9]	train-auc:0.950599+0.00263033	test-auc:0.944823+0.00498001
| 14       | 0.9512   | 0.5182   | 0.03419  | 0.03434  | 6.943    | 6.195    |
[0]	train-auc:0.901657+0.0617168	test-auc:0.896339+0.0674249
[9]	train-auc:0.958429+0.000915196	test-auc:0.953785+0.00121568
| 15       | 0.9584   | 0.7383   | 0.1844   | 0.2372   | 6.812    | 0.3501   |
[0]	train-auc:0.82021+0.0876238	test-auc:0.819568+0.0885246
[9]	train-auc:0.935059+0.0029886	test-auc:0.933888+0.00186369
| 16       | 0.9351   | 0.8175   | 0.06322  | 0.2461   | 2.106    | 0.386    |
[0]	train-auc:0.888779+0.0810194	test-auc:0.877552+0.0959301
[9]	train-auc:0.959174+0.00024753	test-auc:0.951859+0.00294662
| 17       | 0.9592   | 0.5269   | 0.1771   | 0.2359   | 6.726    | 6.609    |
[0]	train-auc:0.849428+0.075084	test-auc:0.848863+0.0764433
[9]	train-auc:0.941849+0.000561635	test-auc:0.941096+0.000817375
| 18       | 0.9418   | 0.606    | 0.137    | 0.07407  | 3.047    | 0.08993  |
[0]	train-auc:0.862105+0.0858793	test-auc:0.860618+0.0878164
[9]	train-auc:0.951087+0.000506186	test-auc:0.949979+0.00120609
| 19       | 0.9511   | 0.5471   | 0.2728   | 0.2965   | 3.784    | 6.881    |
[0]	train-auc:0.886031+0.0690319	test-auc:0.88386+0.0715741
[9]	train-auc:0.953497+0.00100932	test-auc:0.952117+0.00043529
| 20       | 0.9535   | 0.7446   | 0.2236   | 0.2823   | 4.616    | 6.639    |
```

```
==========================================================================================
```

```
def modelfit(alg, dtrain, dtest, predictors, target, eval_metric = True):
        #Fit the algorithm on the data
        if eval_metric:
            alg.fit(dtrain[predictors], dtrain[target].values.ravel(), eval_metric = ['auc'])
        else:
            alg.fit(dtrain[predictors], dtrain[target].values.ravel())

        #Predict training set:
        dtrain_predictions = alg.predict(dtrain[predictors])
        dtrain_predprob = alg.predict_proba(dtrain[predictors])[:,1]

        #Predict test set:
        dtest_predictions = alg.predict(dtest[predictors])
        dtest_predprob = alg.predict_proba(dtest[predictors])[:,1]

        #Print model report:
        print( " Model Report")
        print("Accuracy Train: %.4g" % metrics.accuracy_score(dtrain[target].values, dtrain_pr
        print("Recall Train: %.4g" % metrics.recall_score(dtrain[target].values, dtrain_predi
        print("Accuracy Test: %.4g" % metrics.accuracy_score(dtest[target].values, dtest_pred
        print("Recall Test: %.4g" % metrics.recall_score(dtest[target].values, dtest_predicti
        print("AUC Score (Train): %f" % metrics.roc_auc_score(dtrain[target], dtrain_predprob
```

```
xgb1 = XGBClassifier(
        learning_rate =opt_params['learning_rate'],
        n_estimators=opt_params['n_estimators'],
        max_depth=6,
        min_child_weight=opt_params['min_child_weight'],
        gamma=opt_params['gamma'],
        subsample=opt_params['subsample'],
        colsample_bytree=opt_params['colsample_bytree'],
        objective= 'binary:logistic',
        nthread=4,
        scale_pos_weight=opt_params['scale_pos_weight'],
        seed=6)

modelfit(xgb1, train, test, features, target = label)
```

```
 Model Report
Accuracy Train: 0.9178
Recall Train: 0.9441
Accuracy Test: 0.8989
Recall Test: 0.8304
AUC Score (Train): 0.980017
```

```
print('Accuracy naive model: {:1.4f}'.format(1-test[label].mean()))
```

```
Accuracy naive model: 0.9042
```

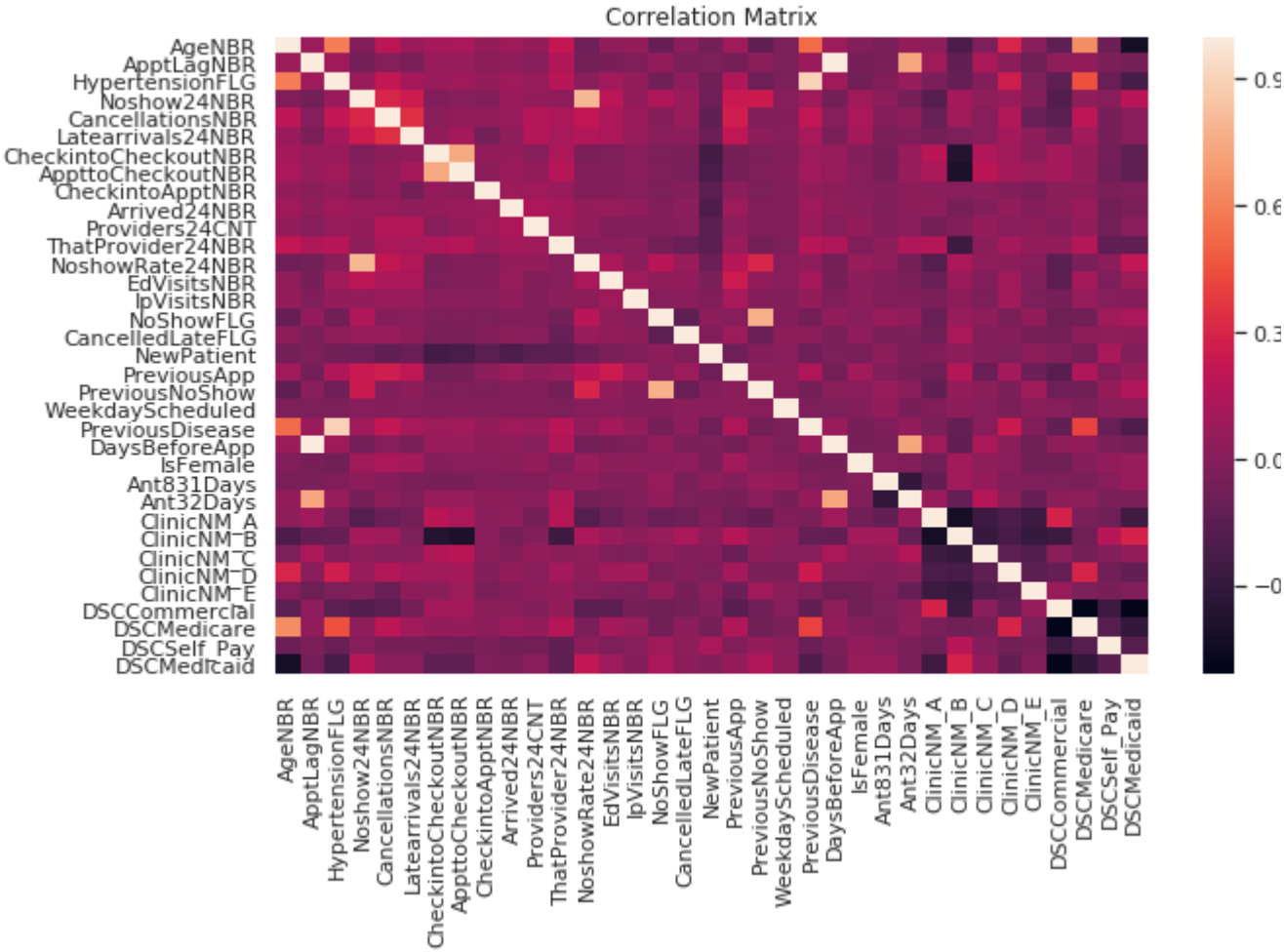
We've gained 2% from the naive model, not so much.

For this reason, we'll try with Logistic Regression (back to my dear glms) and check accuracy for train and test datasets. First, we will study correlations:

```
df_done.columns
```

```
Index(['Unnamed: 0', 'AppointmentID', 'PatientID', 'ClinicNM',
       'AppointmentDTS', 'AppointmentMonthNBR', 'AppointmentWeekdayNBR',
       'AppointmentHourNBR', 'AgeNBR', 'SexFLG', 'HispanicFLG', 'SingleFLG',
       'LivesInApartmentFLG', 'EmailFLG', 'ApptLagNBR', 'InsuranceDSC',
       'HypertensionFLG', 'AsthmaFLG', 'HeartDiseaseFLG', 'ObeseFLG',
       'DiabetesFLG', 'Noshow24NBR', 'CancellationsNBR', 'Latearrivals24NBR',
       'CheckintoCheckoutNBR', 'AppttoCheckoutNBR', 'CheckintoApptNBR',
       'Arrived24NBR', 'Providers24CNT', 'ThatProvider24NBR',
       'NoshowRate24NBR', 'EdVisitsNBR', 'IpVisitsNBR', 'NoShowFLG',
       'CancelledLateFLG', 'NewPatient', 'Cost', 'Rand', 'Revenue', 'Profit',
       'Age_bin', 'ClinicNM_num', 'SexFLG_num', 'InsuranceDSC_num',
       'Noshow24NBR_Bin', 'CheckintoCheckoutNBR_Bin', 'CheckintoApptNBR_bin',
       'Arrived24NBR_bin', 'Providers24CNT_bin', 'ThatProvider24NBR_bin',
       'NoshowRate24NBR_bin', 'CancellationsNBR_Bin', 'AppttoCheckoutNBR_Bin',
       'Latearrivals24NBR_Bin', 'PreviousApp', 'PreviousNoShow',
       'AppointmentDTS2', 'WeekdayScheduled', 'PreviousDisease',
       'DaysBeforeApp', 'DaysBeforeCat', 'AppointmentSunday',
       'AppointmentMonday', 'AppointmentTuesday', 'AppointmentWednesday',
       'AppointmentSaturday', 'AppointmentThursday', 'AppointmentFriday',
       'IsFemale', 'Ant0Days', 'Ant12Days', 'Ant37Days', 'Ant831Days',
       'Ant32Days', 'ClinicNM_A', 'ClinicNM_B', 'ClinicNM_C', 'ClinicNM_D',
       'ClinicNM_E', 'DSCCommercial', 'DSCMedicare', 'DSCSelf_Pay',
       'DSCMedicaid'],
      dtype='object')
```

```
features0 = ['AgeNBR',  'ApptLagNBR',
       'HypertensionFLG', 'Noshow24NBR',
       'CancellationsNBR', 'Latearrivals24NBR', 'CheckintoCheckoutNBR',
       'AppttoCheckoutNBR', 'CheckintoApptNBR', 'Arrived24NBR',
       'Providers24CNT', 'ThatProvider24NBR', 'NoshowRate24NBR', 'EdVisitsNBR',
       'IpVisitsNBR', 'NoShowFLG', 'CancelledLateFLG', 'NewPatient',
       'PreviousApp', 'PreviousNoShow', 'WeekdayScheduled', 'PreviousDisease',
       'DaysBeforeApp',  'IsFemale', 'Ant831Days', 'Ant32Days', 'ClinicNM_A',
       'ClinicNM_B', 'ClinicNM_C', 'ClinicNM_D', 'ClinicNM_E', 'DSCCommercial',
       'DSCMedicare', 'DSCSelf_Pay', 'DSCMedicaid']
corr = df_done[features0].corr()
sns.heatmap(corr,
            xticklabels=corr.columns.values,
            yticklabels=corr.columns.values)
plt.title('Correlation Matrix')
plt.gcf().set_size_inches(10, 6)
plt.show()

corr
```



	AgeNBR	ApptLagNBR	HypertensionFLG	Noshow24NBR	Cancellatic
AgeNBR	1.000000	0.078076	0.587933	-0.013576	0.1
ApptLagNBR	0.078076	1.000000	0.084989	-0.060661	0.0
HypertensionFLG	0.587933	0.084989	1.000000	0.073388	0.2
Noshow24NBR	-0.013576	-0.060661	0.073388	1.000000	0.3
CancellationsNBR	0.198770	0.004148	0.216079	0.302115	1.0
Latearrivals24NBR	0.071979	-0.034814	0.099760	0.249358	0.3
CheckintoCheckoutNBR	0.109558	0.067094	0.074145	-0.019448	0.0
AppttoCheckoutNBR	0.122254	0.060251	0.090263	0.004465	0.0
CheckintoApptNBR	0.048247	0.027778	0.046383	-0.008088	0.0
Arrived24NBR	0.091780	0.030508	0.089657	0.074106	0.0
Providers24CNT	0.052468	0.001493	0.063333	0.053951	0.1
ThatProvider24NBR	0.212905	0.144813	0.175181	0.032787	0.1
NoshowRate24NBR	-0.082063	-0.062936	0.016210	0.797974	0.2
EdVisitsNBR	0.039130	-0.045436	0.097005	0.190977	0.1

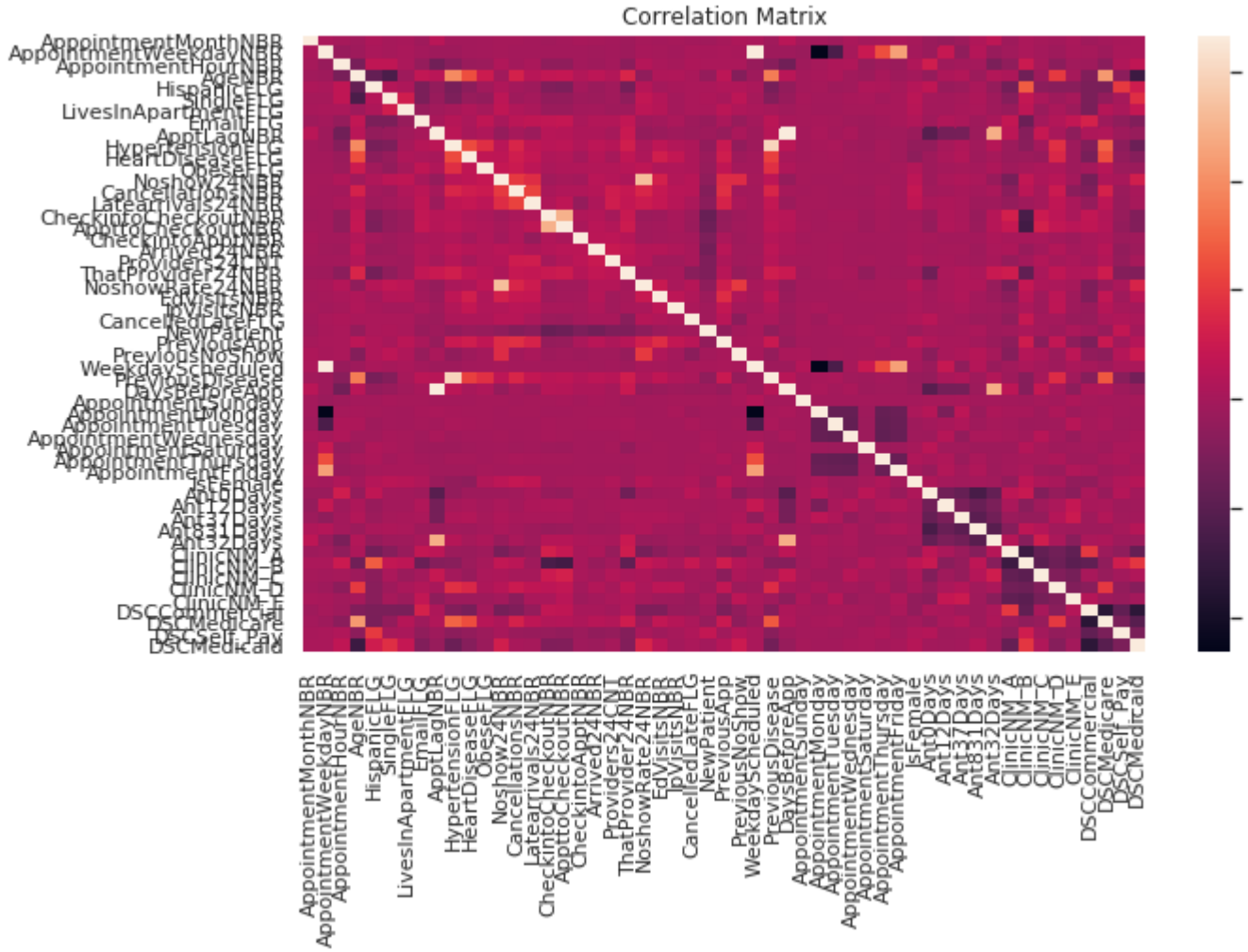
IpVisitsNBR	0.046347	-0.020927	0.048718	0.049949	0.(
NoShowFLG	-0.109362	0.044746	-0.066101	0.151653	-0.(
CancelledLateFLG	0.018296	-0.035112	0.029881	0.066888	0.(
NewPatient	-0.060313	-0.029135	-0.073676	-0.080547	-0.1
PreviousApp	0.087720	-0.055386	0.140846	0.238977	0.2
PreviousNoShow	-0.131518	0.013890	-0.078227	0.241278	-0.(
WeekdayScheduled	-0.030399	-0.030288	-0.027647	0.001601	-0.(
PreviousDisease	0.532415	0.070839	0.897433	0.092491	0.2
DaysBeforeApp	0.078076	1.000000	0.084989	-0.060661	0.(
IsFemale	-0.011850	-0.043860	-0.071768	0.089835	0.1
Ant831Days	-0.039406	-0.039192	-0.017092	0.022399	-0.(
Ant32Days	0.049802	0.729768	0.061767	-0.051966	0.(
ClinicNM_A	0.031724	0.089992	-0.037243	-0.176514	-0.1
ClinicNM_B	-0.205502	-0.124579	-0.108732	0.110722	0.(

From the matrix, we can see that Hipertension is highly correlated with PreviousDisease, Age and Diabetes. Also, Age is highly correlated with PreviousDisease. For this reason, we will eliminate PreviousDisease from the analysis but shall keep the others (correlation is high but not extreme).

Previously, NaN values didn't matter, as XGBoost learned from the data which path was the best. As now observations with missing values can not be used, we will change the variable PreviousNoShow as MissedAppointments : number of previous appointments with no-show.

Also, we will scale the data so all values are between 1 and 0.

```
corr = df_done[features].corr()
sns.heatmap(corr,
            xticklabels=corr.columns.values,
            yticklabels=corr.columns.values)
plt.title('Correlation Matrix')
plt.gcf().set_size_inches(10, 6)
plt.show()

corr
```



	AppointmentMonthNBR	AppointmentWeekdayNBR	AppointmentHourNB
AppointmentMonthNBR	1.000000	0.022109	0.00486
AppointmentWeekdayNBR	0.022109	1.000000	-0.0466
AppointmentHourNBR	0.004862	-0.046610	1.00000
AgeNBR	-0.014247	-0.029996	-0.1748
HispanicFLG	0.026752	0.019057	0.1051
SingleFLG	0.003322	0.005820	0.1054
LivesInApartmentFLG	0.008807	0.014906	0.0210
EmailFLG	-0.021491	-0.008308	-0.0631
ApptLagNBR	0.073841	-0.029727	-0.2044
HypertensionFLG	-0.012622	-0.027530	-0.1223
HeartDiseaseFLG	-0.009809	-0.012260	-0.0438
ObeseFLG	-0.004977	-0.004625	-0.0364
Noshow24NBR	0.001852	0.001254	0.0435
CancellationsNBR	0.006215	-0.002706	-0.0112

Latearrivals24NBR	-0.005357	0.000015	0.0169
CheckintoCheckoutNBR	-0.014716	-0.009192	-0.0832
AppttoCheckoutNBR	-0.006321	-0.007639	-0.0817
CheckintoApptNBR	0.007054	-0.007590	-0.0127
Arrived24NBR	0.002987	-0.014990	-0.0223
Providers24CNT	-0.003836	-0.010107	-0.0034
ThatProvider24NBR	-0.022338	-0.027670	-0.0937
NoshowRate24NBR	0.001431	0.008376	0.0552
EdVisitsNBR	-0.002264	0.005878	0.0144
IpVisitsNBR	0.002491	-0.010008	0.0063
CancelledLateFLG	-0.007968	0.016459	0.0271
NewPatient	-0.015354	0.008494	0.0219
PreviousApp	0.004499	0.016420	0.0215
PreviousNoShow	0.028115	0.007833	0.0215
WeekdayScheduled	0.022066	0.995088	-0.0486
PreviousDisease	-0.014923	-0.026906	-0.1070
DaysBeforeApp	0.073841	-0.029727	-0.2044
AppointmentSunday	-0.000283	-0.042343	-0.0206
AppointmentMonday	-0.012349	-0.694515	0.0020
AppointmentTuesday	-0.020870	-0.362848	0.0327
AppointmentWednesday	0.013992	0.004407	0.0351
AppointmentSaturday	-0.003416	0.145631	-0.0588
AppointmentThursday	0.005973	0.374260	-0.0216
AppointmentFriday	0.014169	0.672410	-0.0399
IsFemale	0.005043	-0.003249	0.0388
Ant0Days	-0.043297	-0.042307	0.1702
Ant12Days	-0.061397	0.126162	-0.0230
Ant37Days	-0.038633	-0.082683	-0.0109
Ant831Days	0.035016	0.055690	0.0000
Ant32Days	0.095913	-0.046476	-0.1620
ClinicNM_A	-0.031596	-0.014004	-0.0841
ClinicNM_B	0.022479	0.024015	0.1489

ClinicNM_B	0.022479	0.024015	0.1489
ClinicNM_C	0.004278	-0.000645	0.0058
ClinicNM_D	-0.010789	-0.030017	-0.0958

Logistic regression

DSCCommercial	-0.019729	0.004232	-0.0255
DSCSelf_Pay	0.000811	0.014716	0.0459

```
df_done.loc[:, 'MissedAppointments'] = df_done.sort_values(['AppointmentWeekdayNBR']).groupby
```

```
features0 = ['AppointmentMonthNBR','MissedAppointments',
       'AppointmentWeekdayNBR', 'AppointmentHourNBR', 'AgeNBR', 'HispanicFLG',
       'SingleFLG', 'LivesInApartmentFLG', 'EmailFLG', 'ApptLagNBR',
       'HypertensionFLG', 'HeartDiseaseFLG', 'ObeseFLG', 'Noshow24NBR',
       'CancellationsNBR', 'Latearrivals24NBR', 'CheckintoCheckoutNBR',
       'AppttoCheckoutNBR', 'CheckintoApptNBR',
       'Providers24CNT',  'EdVisitsNBR',
       'IpVisitsNBR', 'CancelledLateFLG', 'NewPatient',
       'PreviousApp',  'WeekdayScheduled', 'PreviousDisease',
       'DaysBeforeApp', 'AppointmentSunday', 'AppointmentMonday',
       'AppointmentTuesday', 'AppointmentWednesday', 'AppointmentSaturday',
       'AppointmentThursday', 'AppointmentFriday', 'IsFemale', 'Ant0Days',
       'Ant12Days', 'Ant37Days', 'Ant831Days', 'Ant32Days', 'ClinicNM_A',
       'ClinicNM_B', 'ClinicNM_C', 'ClinicNM_D', 'ClinicNM_E', 'DSCCommercial',
       'DSCMedicare', 'DSCSelf_Pay', 'DSCMedicaid']
```

```
scaler = StandardScaler().fit(df_done[features0])
df_rescaled = scaler.transform(df_done[features0])

X_train, X_test, y_train, y_test = train_test_split(df_done[features0], df_done[label], test_
y_train = pd.DataFrame(y_train)
train = X_train.merge(y_train, left_index = True, right_index = True)
y_test = pd.DataFrame(y_test)
test = X_test.merge(y_test, left_index = True, right_index = True)
```

```
logit = LogisticRegression(class_weight = 'balanced', solver = 'liblinear')
modelfit(logit, train, test, features0, label, eval_metric = False)
```

```
Model Report
Accuracy Train: 0.9178
Recall Train: 0.9863
Accuracy Test: 0.9197
Recall Test: 0.9854
AUC Score (Train): 0.967270
```

Accuracy is worst than XGBoost ! Compared to naive model,

We will train a Decision Tree, just to check how does this model compare to the two we already have.

```
tree = DecisionTreeClassifier(max_depth=12, random_state=0)
modelfit(tree, train, test,features0, label, eval_metric=False)
```

```
 Model Report
Accuracy Train: 0.9599
Recall Train: 0.7777
```

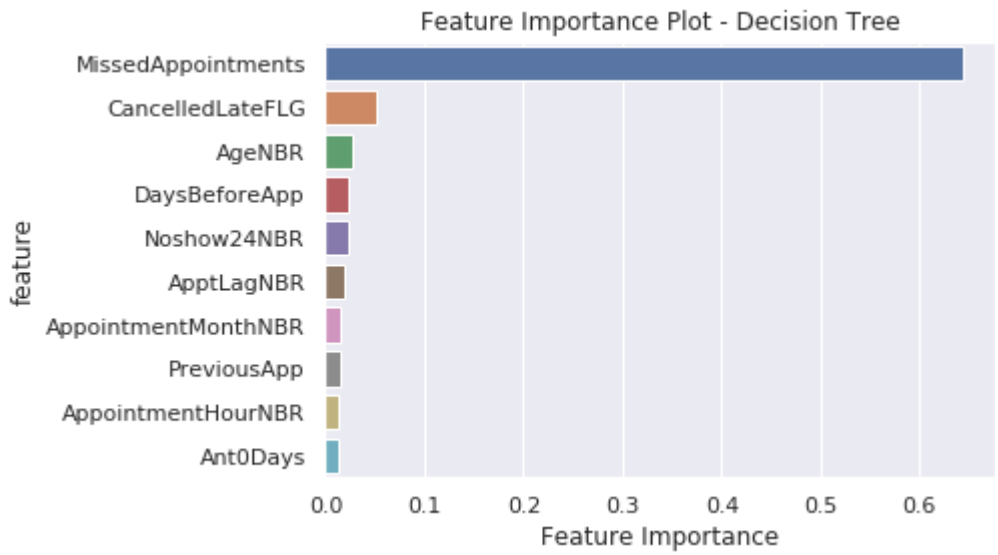
Finally, we will train a Random Forest: as one decision tree performed better, maybe several decision trees perform best.

```
rf = RandomForestClassifier(random_state = 0, class_weight = 'balanced')
modelfit(rf, train, test,features0, label, eval_metric=False)
```

```
 Model Report
Accuracy Train: 0.9968
Recall Train: 0.9694
Accuracy Test: 0.937
Recall Test: 0.5267
AUC Score (Train): 0.999909
```

The best so far !! We will use this model to study variable importance and predictive power.

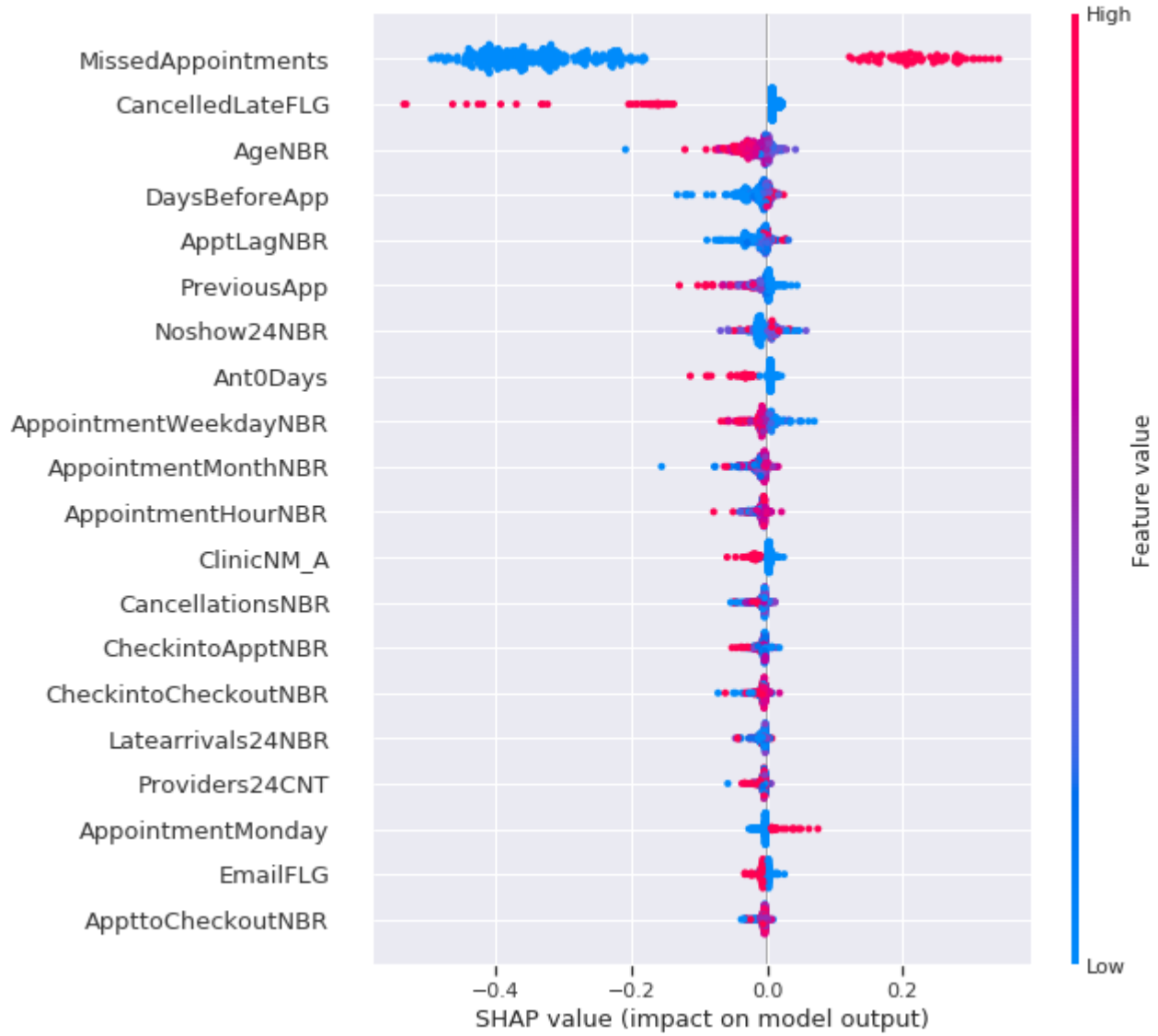
```
feature_importance = pd.DataFrame({'feature' : features0,
                                   'importances' : rf.feature_importances_})
ordered = feature_importance.sort_values(['importances'], ascending = False)
best = ordered[:10]
sns.barplot(x = 'importances', y = 'feature', data = best)
plt.xlabel('Feature Importance')
plt.title('Feature Importance Plot - Decision Tree')
plt.show()
```



We can see that the most important feature is "missed appointments", "cancelled late" and "checkintocheck outnbr". Two of these three variables make reference to the patient's past behavior, which supports the hypothesis of patients repeating behaviors over time. As we have little history this is not enough to say that patients will always behave the same way but for this period of time this is true.

For better understanding, we will use SHAP Values. DanB made a marvelous job explaining advanced uses of SHAP Values (see here) and for better understanding you can check this medium post .

```
sample = X_test.sample(300)
shap.initjs()
explainer = shap.TreeExplainer(rf)
shap_values = explainer.shap_values(sample[features0])
shap.summary_plot(shap_values[1], sample[features0])
```



From SHAP Values, we can see that the most important variable is "MissedAppointments" (as mentioned before). In fact, this variable is able to divide the data, impact negatively on the model output for low values of MissedApointments and impact positively for higher values of missed appointments (which makes sense).

Another variable to watch is "Ant0days". As a reminder, this variable takes the value 1 when the appointment was scheduled with less than a day of anticipation and 0 for any other case (which means that low values of the variable correspond to appointments scheduled with at least one day of anticipation, while high values are for appointments with less than a day of anticipation). In the plot above, we can see there's a clear separation for Ant0days: appointments with value 1 lower the chanses of noshow, while appointments with value 0 receive a positive impact (elevates chances of no-show, also makes sense).

Now we shall study how the model clasificates in general:

```
df_done.head()
```

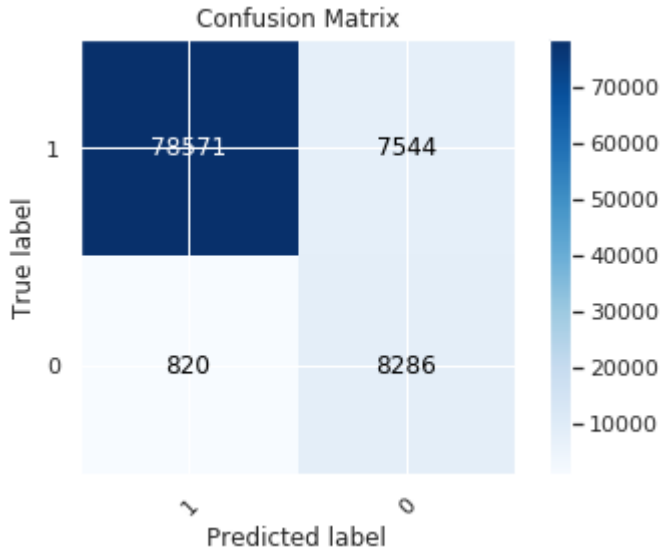
	Unnamed: 0	AppointmentID	PatientID	ClinicNM	AppointmentDTS	AppointmentMonthNBR
0	1	21725	1	E	2018-04-10	4
1	2	11206	2	A	2018-02-07	2
2	3	12548	2	A	2018-02-08	2
3	4	12727	2	A	2018-03-08	3
4	5	86882	2	A	2018-11-09	11

5 rows × 84 columns

```
from google.colab import files

#check for nulls :pd.isnull(df_done).sum() > 0
# e.g. save pandas output as csv
df_done.to_csv('df_done.csv')
files.download('df_done.csv')
```

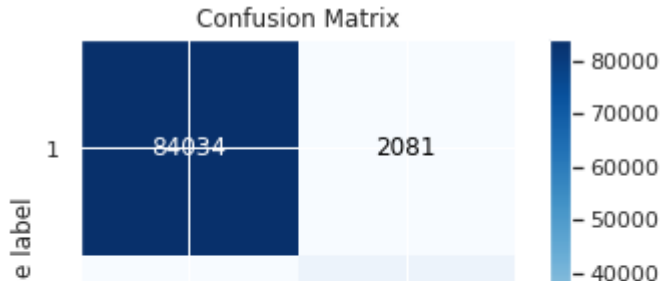
```
#xgboost predictions
df_done['preds'] = xgb1.predict(df_done[features])
cm = metrics.confusion_matrix(df_done['NoShowFLG'], (df_done['preds'] > 0.5)*1)
normalize = False

fig, ax = plt.subplots()
im = ax.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
ax.figure.colorbar(im, ax=ax)
# We want to show all ticks...
ax.set(xticks=np.arange(cm.shape[1]),
       yticks=np.arange(cm.shape[0]),
       xticklabels=['1', '0'], yticklabels=['1', '0'],
       title='Confusion Matrix',
       ylabel='True label',
       xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
         rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
fmt = '.2f' if normalize else 'd'
thresh = cm.max() / 2.
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        ax.text(j, i, format(cm[i, j], fmt),
                ha="center", va="center",
                color="white" if cm[i, j] > thresh else "black")

ax

print('True Positive Rate (recall): {:1.3f}'.format(metrics.recall_score(df_done['NoShowFLG']
print('Accuracy: {:1.3f}'.format(metrics.accuracy_score(df_done['NoShowFLG'], (df_done['preds
```

```
True Positive Rate (recall): 0.910
Accuracy: 0.912
```




```
#tree predictions
df_done['preds'] = tree.predict(df_done[features0])
cm = metrics.confusion_matrix(df_done['NoShowFLG'], (df_done['preds'] > 0.5)*1)
normalize = False

fig, ax = plt.subplots()
im = ax.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
ax.figure.colorbar(im, ax=ax)
# We want to show all ticks...
ax.set(xticks=np.arange(cm.shape[1]),
       yticks=np.arange(cm.shape[0]),
       xticklabels=['1', '0'], yticklabels=['1', '0'],
       title='Confusion Matrix',
       ylabel='True label',
       xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
         rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
fmt = '.2f' if normalize else 'd'
thresh = cm.max() / 2.
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        ax.text(j, i, format(cm[i, j], fmt),
                ha="center", va="center",
                color="white" if cm[i, j] > thresh else "black")

ax
print('True Positive Rate (recall): {:1.3f}'.format(metrics.recall_score(df_done['NoShowFLG']
print('Accuracy: {:1.3f}'.format(metrics.accuracy_score(df_done['NoShowFLG'], (df_done['preds
```

```
True Positive Rate (recall): 0.750
Accuracy: 0.954
```




```
#logistic regression predictions
df_done['preds'] = logit.predict(df_done[features0])
cm = metrics.confusion_matrix(df_done['NoShowFLG'], (df_done['preds'] > 0.5)*1)
normalize = False

fig, ax = plt.subplots()
im = ax.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
ax.figure.colorbar(im, ax=ax)
# We want to show all ticks...
ax.set(xticks=np.arange(cm.shape[1]),
       yticks=np.arange(cm.shape[0]),
       xticklabels=['1', '0'], yticklabels=['1', '0'],
       title='Confusion Matrix',
       ylabel='True label',
       xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
         rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
fmt = '.2f' if normalize else 'd'
thresh = cm.max() / 2.
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        ax.text(j, i, format(cm[i, j], fmt),
                ha="center", va="center",
                color="white" if cm[i, j] > thresh else "black")

ax

print('True Positive Rate (recall): {:1.3f}'.format(metrics.recall_score(df_done['NoShowFLG']
print('Accuracy: {:1.3f}'.format(metrics.accuracy_score(df_done['NoShowFLG'], (df_done['preds
```

```
True Positive Rate (recall): 0.986
Accuracy: 0.918
```

From the above, we can see how xgboost classifies correctly or incorrectly each appointment. The model is able to correctly predict more than 90% of No-Shows, which is very high. At the same time, more than 91% of cases are correctly labeled so the hospital should expect less than 9% of error (low)

```
#randomforest predictions
df_done['preds'] = rf.predict(df_done[features0])
cm = metrics.confusion_matrix(df_done['NoShowFLG'], (df_done['preds'] > 0.5)*1)
normalize = False

fig, ax = plt.subplots()
im = ax.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
ax.figure.colorbar(im, ax=ax)
# We want to show all ticks...
ax.set(xticks=np.arange(cm.shape[1]),
       yticks=np.arange(cm.shape[0]),
       xticklabels=['1', '0'], yticklabels=['1', '0'],
       title='Confusion Matrix',
       ylabel='True label',
       xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
         rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
fmt = '.2f' if normalize else 'd'
thresh = cm.max() / 2.
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        ax.text(j, i, format(cm[i, j], fmt),
                ha="center", va="center",
                color="white" if cm[i, j] > thresh else "black")

ax
print('True Positive Rate (recall): {:1.3f}'.format(metrics.recall_score(df_done['NoShowFLG']
print('Accuracy: {:1.3f}'.format(metrics.accuracy_score(df_done['NoShowFLG'], (df_done['preds
```

```
True Positive Rate (recall): 0.836
Accuracy: 0.979
```