An Analysis of Missed Patient Appointments in Brazil in the Year 2016

by

Jacktone Etemesi

Phone: +254708578181

email: jacktoneetemesi1@gmail.com

Introduction

This project involves an analysis of patient records in Brazil to determine the attendance rates of scheduled medical appointments. The aim of the project is to address the following research questions:

Q1. In general, what sex is associated with the highest no-show rate?

Q2. What age bracket is associated with the highest number of missed appointments?

Q3. What day of the week registered the highest number of missed appointments?

Q4. What are the five leading hospital neighborhoods in terms of alcoholism cases? How do these neighborhoods compare in terms of missed patient appointments?

Q5. Is there a correlation between the number of days a patient in a given region has to wait for an appointment and the number of missed appointments in that particular region?

Q6. Can a model be built to predict if a patient will show up for an appointment?

The dataset used for this analysis consists of information from 110,527 medical appointments in Brazil. Each row contains various characteristics about the patient.

The dataset comprises 14 columns, which are described as follows:

1. *PatientId: The unique ID assigned to each patient.*
2. *AppointmentID: The unique ID assigned to each appointment.*
3. *Gender: The gender of the patient.*
4. *ScheduledDay: The date when the patient scheduled the appointment.*
5. *AppointmentDay: The date when the patient is expected to attend the appointment.*
6. *Age: The age of the patient.*
7. *Neighbourhood: The location where the appointment takes place.*

8. Scholarship: Indicates whether the patient is enrolled in the Brazilian welfare program.
9. Hipertension: Indicates whether the patient has hypertension.
10. Diabetes: Indicates whether the patient has diabetes.
11. Alcoholism: Indicates whether the patient has alcoholism.
12. Handcap: The number of handicaps the patient has.
13. SMS_received: Indicates whether the patient received an SMS reminder for the appointment.
14. No-show: Indicates whether the patient showed up for the appointment or not.

By analyzing these variables, we can gain insights into the factors that contribute to missed patient appointments in Brazil and potentially develop predictive models to enhance appointment attendance rates.

Assumptions:

This analysis is based on the following assumptions:

1. All patients prefer going to hospitals within their neighborhoods. With this assumption, we consider the hospital neighborhood to be the same as that of patients who scheduled for an appointment in that particular hospital neighborhood.

2. We assume that all missed appointments were solely due to causes related to the patient, such as personal reasons or forgetting the appointment. We do not consider appointments that were canceled or terminated by the hospitals.

3. We assume that patients missed their appointments for reasons other than death. This analysis focuses on the factors that contribute to missed appointments, excluding cases where patients were unable to attend due to severe circumstances like death.
</i>

Step 1: Data Wrangling

Importing relevant Libraries for Data cleaning, Exploration and Visualization

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
from plotly import __version__
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
import cufflinks as cf
#for notebook
init_notebook_mode(connected=True)
# For offline use
cf.go_offline()
```

```
import plotly.express as px
%matplotlib inline

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

Checking for the shape of our dataset in terms of number of rows and columns

In [ ]:
```
df.shape
```

Out[ ]:
```
(110527, 14)
```

The DataFrame used in this analysis consists of 110,527 rows and 14 columns.

In [ ]:
```
df.head()
```

Out[ ]:

	PatientId	AppointmentID	Gender	ScheduledDay	AppointmentDay	Age	Neighbourhood	Scholarship	Hipertension	Diabetes	Alcoholism	Handcap	SMS_received	No-show
0	2.987250e+13	5642903	F	2016-04-29T18:38:08Z	2016-04-29T00:00:00Z	62	JARDIM DA PENHA	0	1	0	0	0	0	No
1	5.589978e+14	5642503	M	2016-04-29T16:08:27Z	2016-04-29T00:00:00Z	56	JARDIM DA PENHA	0	0	0	0	0	0	No
2	4.262962e+12	5642549	F	2016-04-29T16:19:04Z	2016-04-29T00:00:00Z	62	MATA DA PRAIA	0	0	0	0	0	0	No
3	8.679512e+11	5642828	F	2016-04-29T17:29:31Z	2016-04-29T00:00:00Z	8	PONTAL DE CAMBURI	0	0	0	0	0	0	No
4	8.841186e+12	5642494	F	2016-04-29T16:07:23Z	2016-04-29T00:00:00Z	56	JARDIM DA PENHA	0	1	1	0	0	0	No

Checking for columns and their data types

In [ ]:
```
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
```

Assessing Statistical Summary of Our Data

In [ ]:
```
df.describe()
```

Out[ ]:

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

The following issues within our columns need to be addressed.

```
i. Column names need to be changed to lowercase
ii. ScheduleDay and AppointmentDay columns need to be converted to datetime dtype
iii. There is a negative value in the Age column
```

Step 2: Data Cleaning

i. Changing column names to lowercase letters

In [ ]:
```
for col in df.columns:
    df.rename(columns=lambda x:x.lower(),inplace=True)
df.head(2)
```

Out[ ]:

	patientid	appointmentid	gender	scheduledday	appointmentday	age	neighbourhood	scholarship	hipertension	diabetes	alcoholism	handcap	sms_received	no-show
0	2.987250e+13	5642903	F	2016-04-29T18:38:08Z	2016-04-29T00:00:00Z	62	JARDIM DA PENHA	0	1	0	0	0	0	No
1	5.589978e+14	5642503	M	2016-04-29T16:08:27Z	2016-04-29T00:00:00Z	56	JARDIM DA PENHA	0	0	0	0	0	0	No

ii. Converting scheduledday and appointmentday columns into datetime dtype

In [ ]:
```
# Creating a function to iterate over a list of given columns in the dataFrame and converting their data types into Datetime
def todtime(df,cols):
    for i in cols:
        df[i]=pd.to_datetime(df[i],utc=True)
    return df

todtime(df,['scheduledday','appointmentday']) #calling the function
df[['scheduledday','appointmentday']].dtypes #checking datatypes
```

Out[ ]:
```
scheduledday      datetime64[ns, UTC]
appointmentday    datetime64[ns, UTC]
dtype: object
```

iii. Removing negative values from the age columns

In [ ]:
```
df[df['age']<0]
```

Out[ ]:

	patientid	appointmentid	gender	scheduledday	appointmentday	age	neighbourhood	scholarship	hipertension	diabetes	alcoholism	handcap	sms_received	n sho
99832	4.659432e+14	5775010	F	2016-06-06 08:58:13+00:00	2016-06-06 00:00:00+00:00	-1	ROMÃO	0	0	0	0	0	0	N

one observation has a negative value in the age column. Since age values should be positive integers, it is unclear why this negative value was recorded. To ensure the integrity of our analysis and prevent potential bias in our model, it is advisable to remove this particular row from the dataset.

In [ ]:
```
df.drop(index=99832,axis=0,inplace=True) # Deleting the row with a negative age value from our dataset.
```

In [ ]:
```
df[df['age']<0]
```

Out[ ]:

patientid	appointmentid	gender	scheduledday	appointmentday	age	neighbourhood	scholarship	hipertension	diabetes	alcoholism	handcap	sms_received	no-show

iv. Checking for Missing Values

In [ ]:
```
df.isnull().sum().sum()
```

Out[ ]:
```
0
```

Our dataset does not have any missing values.

v. Checking for duplicate values

In [ ]:
```
df.duplicated().sum()
```

Out[ ]:
```
0
```

Our dataset does not have any duplicate entries.

vi. Checking for Outliers

Under this section, we will check for outliers within the age column, which is the only continuous numerical variable in our dataset. To begin, we will examine the statistical summary of the age column.

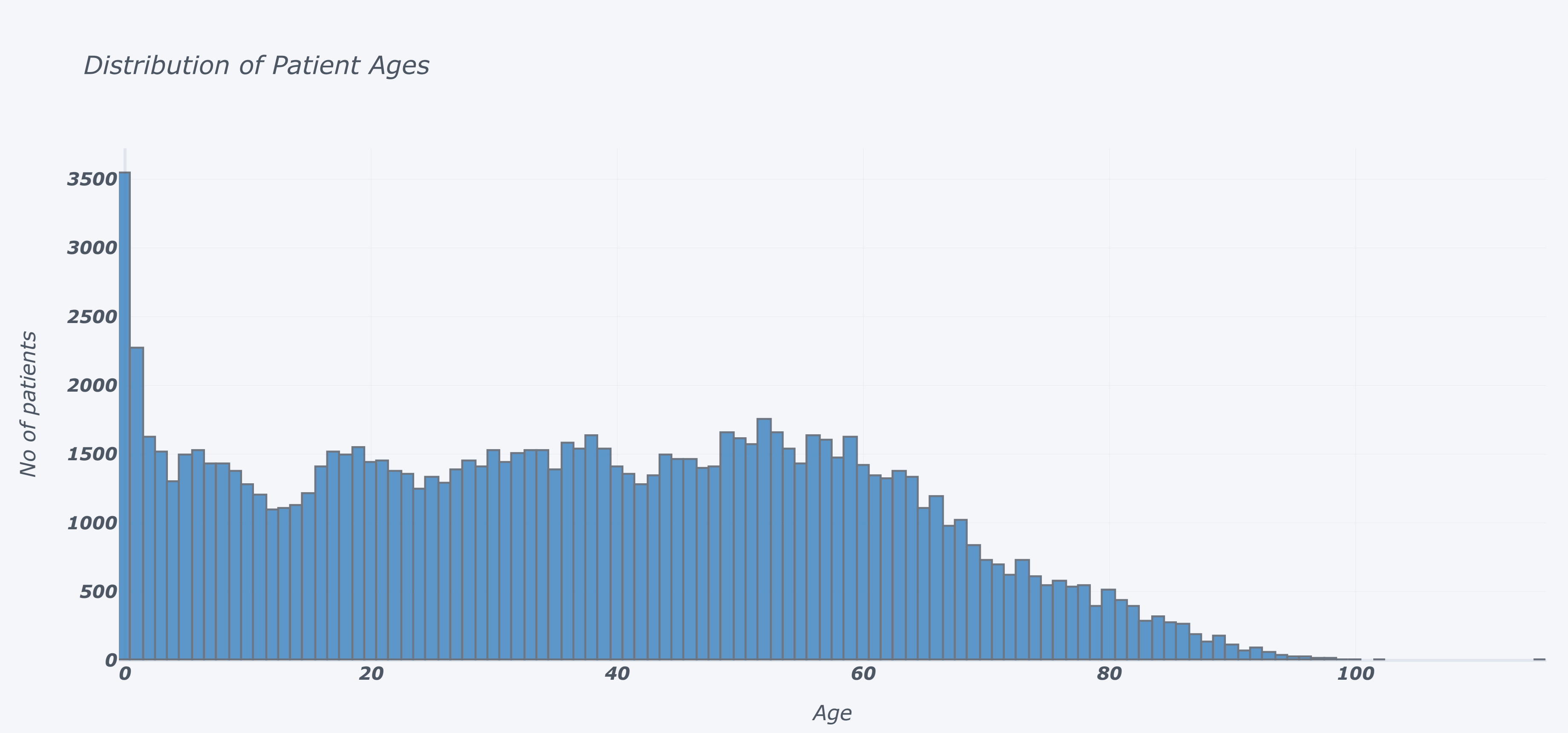
In [ ]:
```
df['age'].describe()
```

Out[ ]:
```
count    110526.000000
mean         37.089219
std          23.110026
min           0.000000
25%          18.000000
50%          37.000000
75%          55.000000
max         115.000000
Name: age, dtype: float64
```

We observe that the youngest patient had less than a year, while the oldest patient was 115 years old. The majority of the patients were approximately 37.10 years old, with a standard deviation of 23.11 years.

We will now proceed to visualize the distribution of patient ages using a histogram and then create a boxplot to investigate the presence of outliers within our dataset. The presence of outliers can potentially skew our model and affect its performance.

In [ ]:
```
df['age'].iplot(kind='hist', title='Distribution of Patient Ages', xTitle='Age',yTitle='No of patients',color='blue')
```



We will perform the Shapiro-Wilk test to determine if our data follows a normal distribution. The Shapiro-Wilk test is a statistical test used to assess the normality of a dataset. It tests the null hypothesis that the data is normally distributed. If the p-value obtained from the test is greater than a chosen significance level (e.g., 0.05), we can conclude that there is no significant evidence to reject the null hypothesis and that our data can be assumed to be normally distributed.

In [ ]:
```
from scipy.stats import shapiro
shapiro(df['age'].sample(5000,random_state=101)) #eliminate random state to test out other samples
```

Out[ ]:
```
ShapiroResult(statistic=0.9712730646133423, pvalue=1.5518840684527575e-30)
```

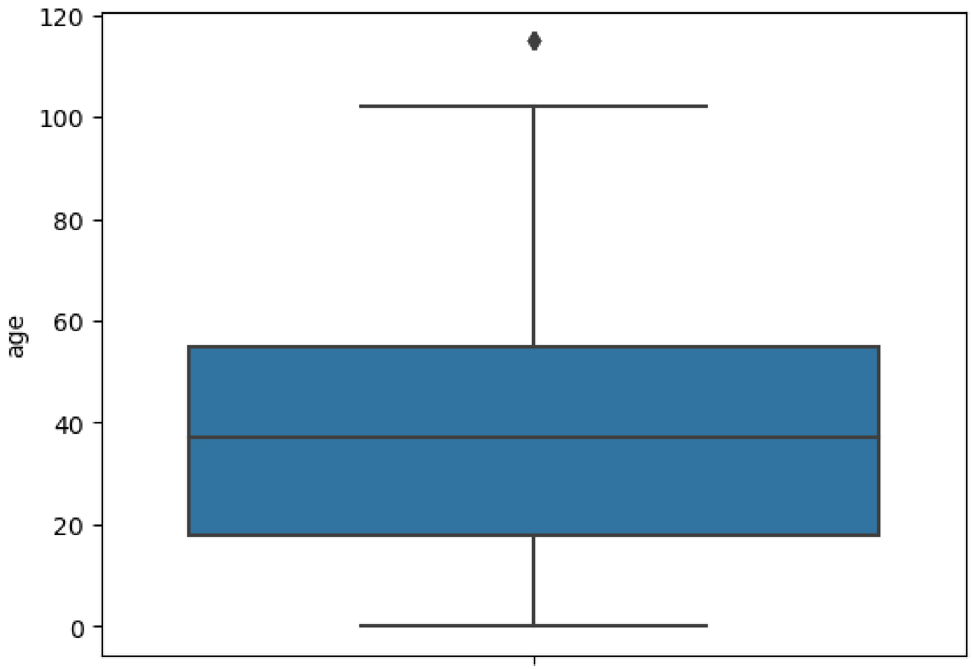
The obtained p-value of 1.55, which is greater than the chosen significance level of 0.05 (alpha), suggests that there is no significant evidence to reject the null hypothesis. Therefore, we can conclude that our data is normally distributed, indicating that the ages of the patients follow a normal distribution.

NB: The Shapiro-Wilk test was conducted on a random sample of 5,000 records. It is worth noting that the accuracy of the p-value tends to be compromised when the sample size exceeds 5,000. </i>

Now that we have established the distribution of our variable, we will proceed to inspect for outliers using a boxplot."

In [ ]:
```
sns.boxplot(y=df['age'])
```

Out[ ]:
```
<Axes: ylabel='age'>
```



From the box plot, we observe that there is an outlier in our dataset. The percentile values for the variable are as follows:

- *The minimum age observed in the dataset is 0.00, indicating the presence of at least one patient with an age of zero or close to zero.*
- *The 25th percentile (25%) value is 18.00, which means that 25% of the patients have an age of 18 or below.*
- *The median (50%) value is 37.00, indicating that 50% of the patients have an age of 37 or below.*
- *The 75th percentile (75%) value is 55.00, suggesting that 75% of the patients have an age of 55 or below.*

Furthermore, the presence of an outlier in the dataset is also noted, which implies that there is at least one patient with an age that significantly deviates from the majority of the other patients (115 years old). </i>

Having identified the outlier age value, let's inspect our dataset to determine the number of patients who are 115 years old.

In [ ]:
```
df.query('age>=115')
```

Out[ ]:

	patientid	appointmentid	gender	scheduledday	appointmentday	age	neighbourhood	scholarship	hipertension	diabetes	alcoholism	handcap	sms_received	n sho
63912	3.196321e+13	5700278	F	2016-05-16 09:17:44+00:00	2016-05-19 00:00:00+00:00	115	ANDORINHAS	0	0	0	0	1	0	Y
63915	3.196321e+13	5700279	F	2016-05-16 09:17:44+00:00	2016-05-19 00:00:00+00:00	115	ANDORINHAS	0	0	0	0	1	0	Y
68127	3.196321e+13	5562812	F	2016-04-08 14:29:17+00:00	2016-05-16 00:00:00+00:00	115	ANDORINHAS	0	0	0	0	1	0	Y
76284	3.196321e+13	5744037	F	2016-05-30 09:44:51+00:00	2016-05-30 00:00:00+00:00	115	ANDORINHAS	0	0	0	0	1	0	N
97666	7.482346e+14	5717451	F	2016-05-19 07:57:56+00:00	2016-06-03 00:00:00+00:00	115	SÃO JOSÉ	0	1	0	0	0	1	N

We observe that there are a total of 5 appointment entries in our dataset where the age values are greater than or equal to 115. To gain further insight into the distribution of this age group, we will inspect our dataset for unique patient IDs corresponding to appointments made by patients whose age is greater than or equal to 115 years, as follows:

In [ ]:
```
df.query('age>=115')['patientid'].unique()
```

Out[ ]:
```
array([3.19632116e+13, 7.48234579e+14])
```

In [ ]:
```
len(df['patientid'].unique())
```

Out[ ]:
```
62298
```

The query above returns two unique patient IDs, indicating that out of the 62,298 patients who scheduled appointments, only two of them had ages of 115 and above. Since the number of such cases is not substantial, it would be reasonable to simply drop these entries from the dataset rather than applying advanced outlier detection techniques.

In [ ]:
```
df.drop(index=df.query('age>=115').index,axis=0,inplace=True,)
```

In [ ]:
```
df.query('age>=115')
```

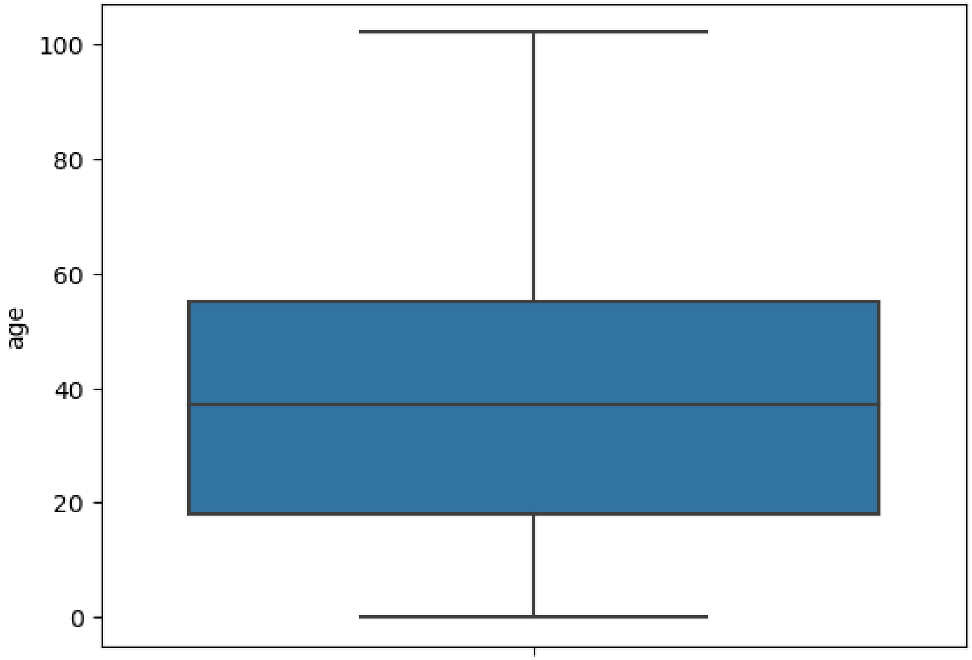
Out[ ]:

patientid	appointmentid	gender	scheduledday	appointmentday	age	neighbourhood	scholarship	hipertension	diabetes	alcoholism	handcap	sms_received	no-show

Now that we have successfully dropped these entries, we will create a box plot to confirm that the outliers have been eliminated from the dataset.

In [ ]:
```
sns.boxplot(y=df['age'])
```

Out[ ]: <Axes: ylabel='age'>



The age column has now been cleaned for outliers, allowing us to proceed with analyzing the data.

Step 3: Answering Analysis Questions

Q1. In general, what gender is associated with the highest no-show up rate?

To address this question, our approach will begin by examining the number of unique genders present in our dataset. Subsequently, we will determine the count of observations corresponding to each gender, allowing us to gain insights into the distribution of gender within our dataset.

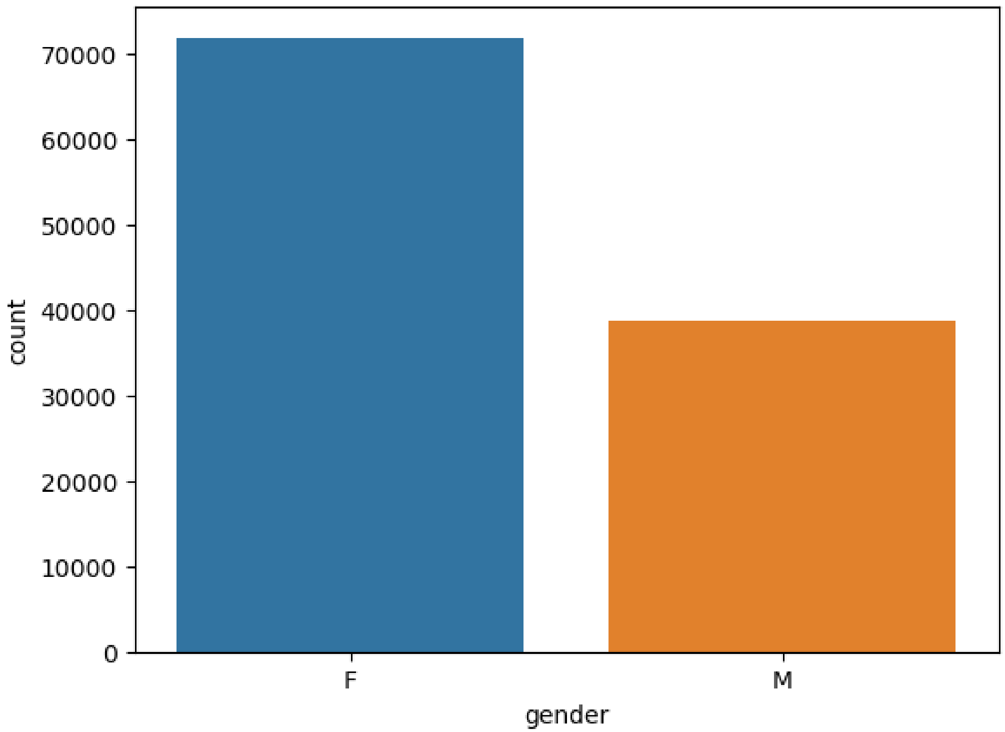
In [ ]:
```
df['gender'].value_counts()
```

Out[ ]:
```
F    71834
M    38687
Name: gender, dtype: int64
```

Within our dataset, we have identified two genders: 'F' represents female patients, while 'M' represents male patients. Our analysis reveals a total of 71,839 male patients and 38,687 female patients, summing up to a dataset containing 110,526 patients in total as shown by the bar graph below

In [ ]:
```
sns.countplot(data=df,x='gender')
```

Out[ ]:
```
<Axes: xlabel='gender', ylabel='count'>
```



Now, let's examine the number of patients who scheduled appointments but did not show up on the designated appointment day.

Having gained an understanding of the patient distribution across genders, we can now proceed to answer our question. This will involve grouping the data by gender and no-show status, resulting in a new dataframe that provides a comparative analysis of appointment attendance based on gender.

In [ ]:
```
attendance=pd.DataFrame(df.groupby(by=['gender','no-show'],as_index=False).count()[['gender','no-show','patientid']])
attendance.columns=['gender','no-show','count']
attendance.head()
```

Out[ ]:

	gender	no-show	count
0	F	No	57243
1	F	Yes	14591
2	M	No	30962
3	M	Yes	7725

From the table above, we observe that out of the 71,839 female patients, 57,245 showed up for their appointments while 14,594 failed to show up. Similarly, among the 38,687 male patients, 30,962 showed up while 7,725 did not. Based on these figures, we can conclude that 20.314% of female patients did not show up for their appointments, while 19.968% of male patients failed to attend their appointments.

When considering the cumulative missed patient appointments, male patients accounted for 34.612% of all missed appointments, whereas female patients constituted 65.388%. These proportions are visually represented in the clustered bar chart below:

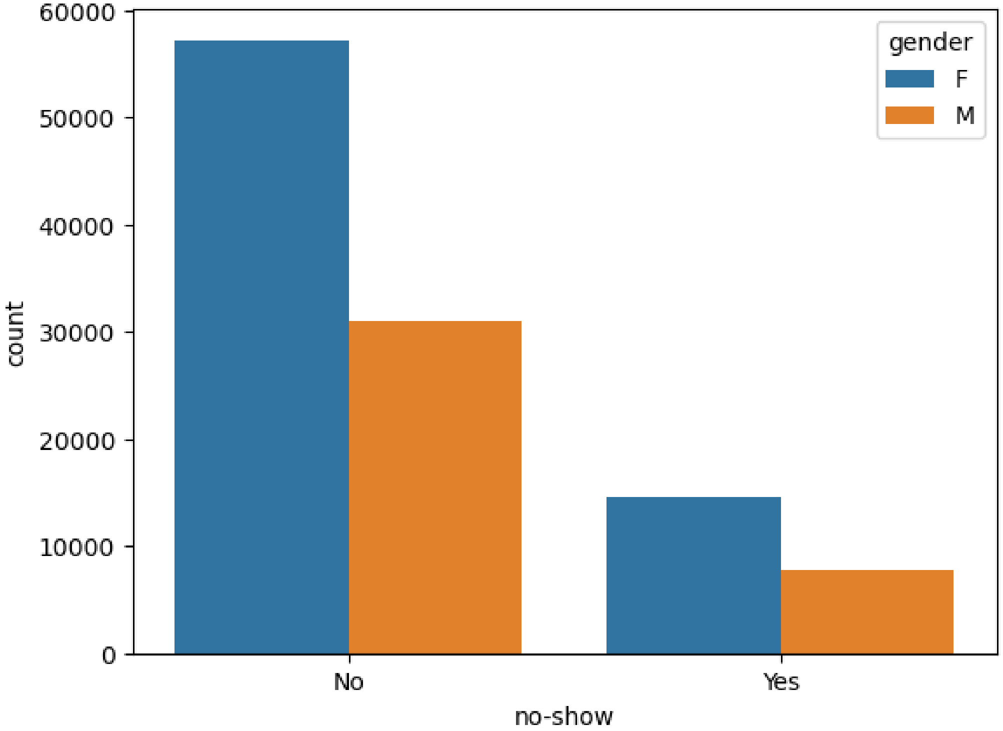
</i>

In [ ]:

```
sns.countplot(data=df,x='no-show',hue='gender')
```

Out[ ]:

```
<Axes: xlabel='no-show', ylabel='count'>
```



Out of the 22,319 patients that missed their appointments;

```
- 14,594 of them were women
- 7,725 of them were men
```

Q2. What age group is associated with the highest number of missed appointments?

This question aims to investigate the relationship between a patient's age and their likelihood of showing up for an appointment. To address this, we will follow these steps:

```
1. Examine the statistical summary of the age column to determine appropriate age groups (bins) for our analysis.
2. Introduce a new column, "age_group," to categorize patients based on their age group.
3. Group the missed appointments based on the age_group and calculate the count of missed appointments for each age group.
4. Visualize the relationship between age groups and missed appointments using a pie chart.
```

Let's begin by examining the age column to gain an initial understanding of the patients' age distribution. </i>

In [ ]:
```python
df['age'].describe()
```

Out[ ]:
```
count    110521.000000
mean         37.085694
std          23.104606
min           0.000000
25%          18.000000
50%          37.000000
75%          55.000000
max         102.000000
Name: age, dtype: float64
```

Based on the previous findings, we have determined that the oldest patient in the dataset is 115 years old, while the youngest patient is less than a year old. To analyze the age distribution effectively, we will need to sort and group the data into four distinct categories with the following labels:

```
- child: Includes patients between 0 and 17 years old.
- Youth: Encompasses patients between 18 and 35 years old.
- Adult: Comprises patients between 36 and 55 years old.
- Elderly: Consists of patients above 55 years old.
```

By organizing the data into these age categories, we can better understand the distribution of patients across different age groups and assess any potential patterns or trends.

</i>

In [ ]:
```python
# Creating labels and bins
labels='child,youth,adult,elderly'.split(',')
bins=[0,18,36,56,115]
# inserting age groups into our dataframe
df['age_group']=pd.cut(x=df['age'],bins=bins, labels=labels,right=False)
```

In [ ]:
```python
df[['age','age_group']].sample(5)
```

Out[ ]:

	age	age_group
48834	60	elderly
10371	8	child
59418	10	child
80395	68	elderly
43015	10	child

Now that we have successfully created an age group column based on the defined age brackets, we can proceed to calculate the number of missed patient appointments within each age group. This analysis will provide valuable insights into the attendance patterns and missed appointment rates across different age groups.

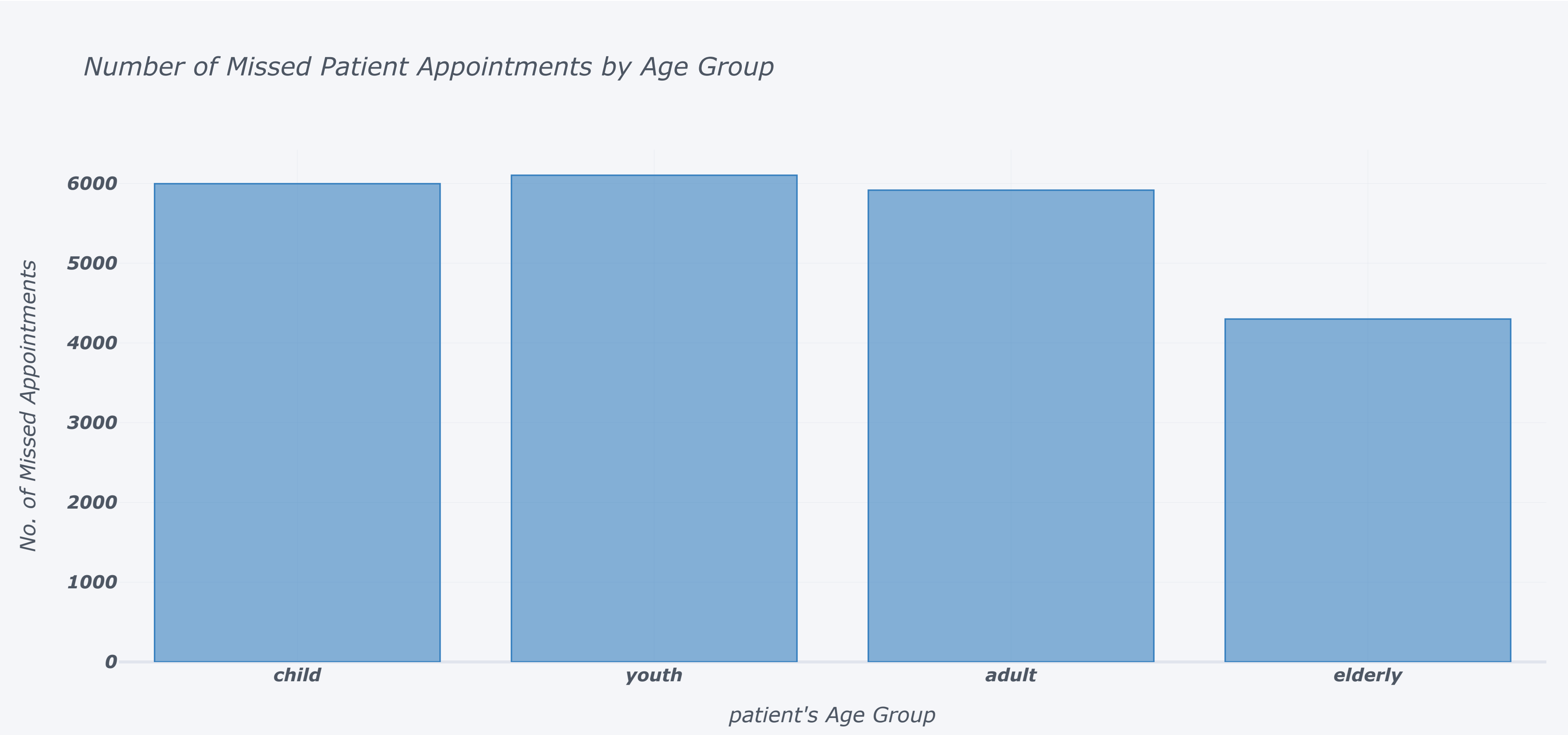
In [ ]:
```
# grouping missed appointments by age_group
age_df=df.query('`no-show`=="Yes"').groupby('age_group',as_index=False).count()[['age_group','no-show']].sort_values(by='no-show',ascending=False,igno
age_df
```

Out[ ]:

	age_group	no-show
0	youth	6103
1	child	5997
2	adult	5916
3	elderly	4300

From the query above, we can observe that the youth age group had the highest number of missed patient appointments, totaling 6,103 appointments. The kid age group follows closely behind with 5,997 missed appointments, while the adult age group had 5,916 missed appointments. In contrast, the elderly age group had the lowest number of missed appointments, with a total of 4,300. This information provides insights into the distribution of missed patient appointments across different age groups and it can be visualized as follows:

In [ ]:
```
df.query('`no-show`=="Yes"').groupby(by='age_group',as_index=False).count()[['age_group','no-show']].iplot(kind='bar',x='age_group',y='no-show',xTitle
```



In terms of percentages, the age_groups rank as follows:

1. ***Youth: Approximately 28.19% of appointments were missed in this age group.***

2. ***Child: Roughly 27.84%of appointments were missed in this age group.***

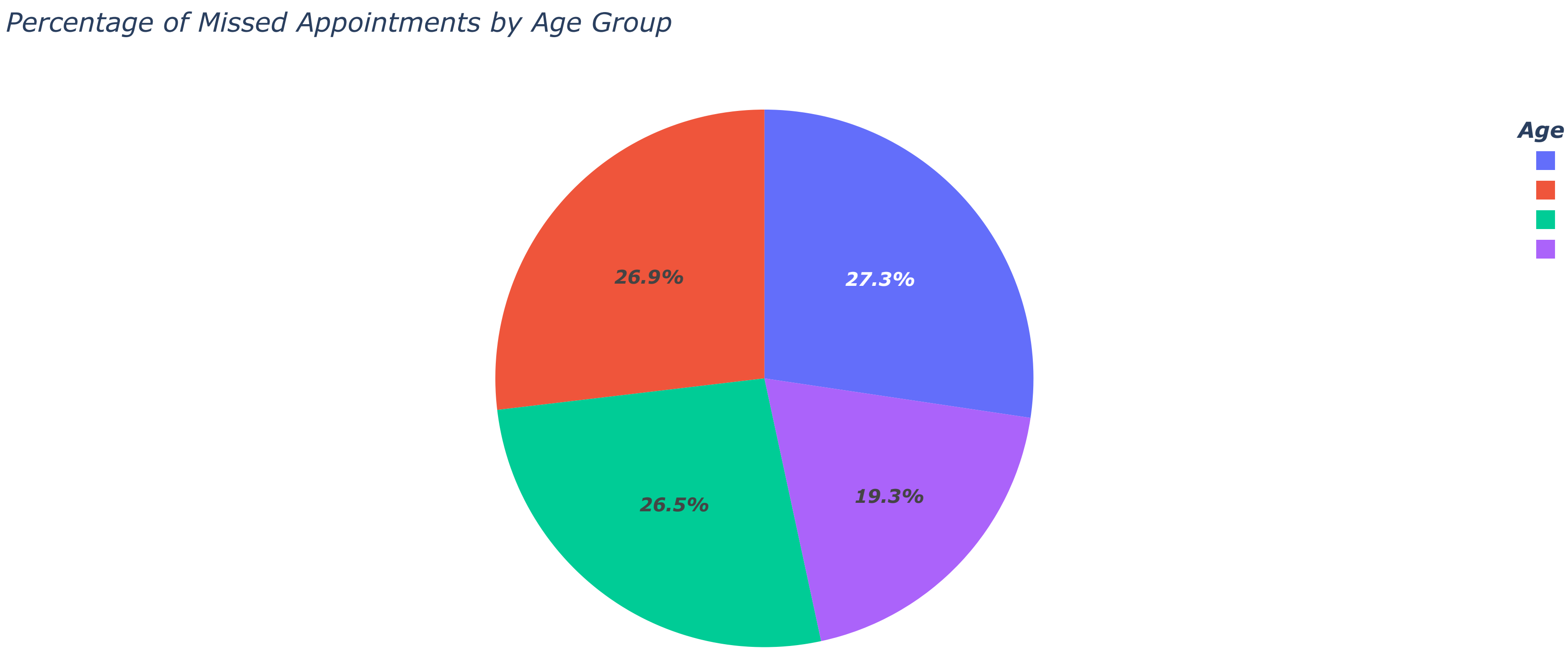
3. ***Adult: Around 27.47%of appointments were missed in this age group.***

4. ***Elderly: Approximately 19.50% of appointments were missed in this age group.***

This can also be represented on a pie chart as follows: </i>

In [ ]:
```
df_missed_appointments = df.query('`no-show` == "Yes"').groupby(by='age_group', as_index=False).count()
fig = px.pie(df_missed_appointments, values='no-show', names='age_group', labels=['Child', 'Youth', 'Adult', 'Elderly'],title=
```

```
"Percentage of Missed Appointments by Age Group")
fig.update_layout(legend_title='Age Group')
fig.show()
```



Based on the previous queries on missed appointments by age group, the age bracket associated with the highest number of missed appointments is the youth age group as it had a total of 6,103 missed appointments, which is the highest among all age groups.

Q3. What day of the week registered the highest number of missed appointments?

To answer this question, we will feature engineer a new column called "week_day" to represent the day of the week when an appointment was scheduled to happen. We will group all the missed appointments by week_day and compare their count to identify which day had the highest number of missed appointments.

To begin, let's create the "week_day" column. </i>

In [ ]:
```
df['week_day']=df['appointmentday'].apply(lambda x:x.strftime('%A'))
df['week_day'].value_counts().sort_values(ascending=False)
```

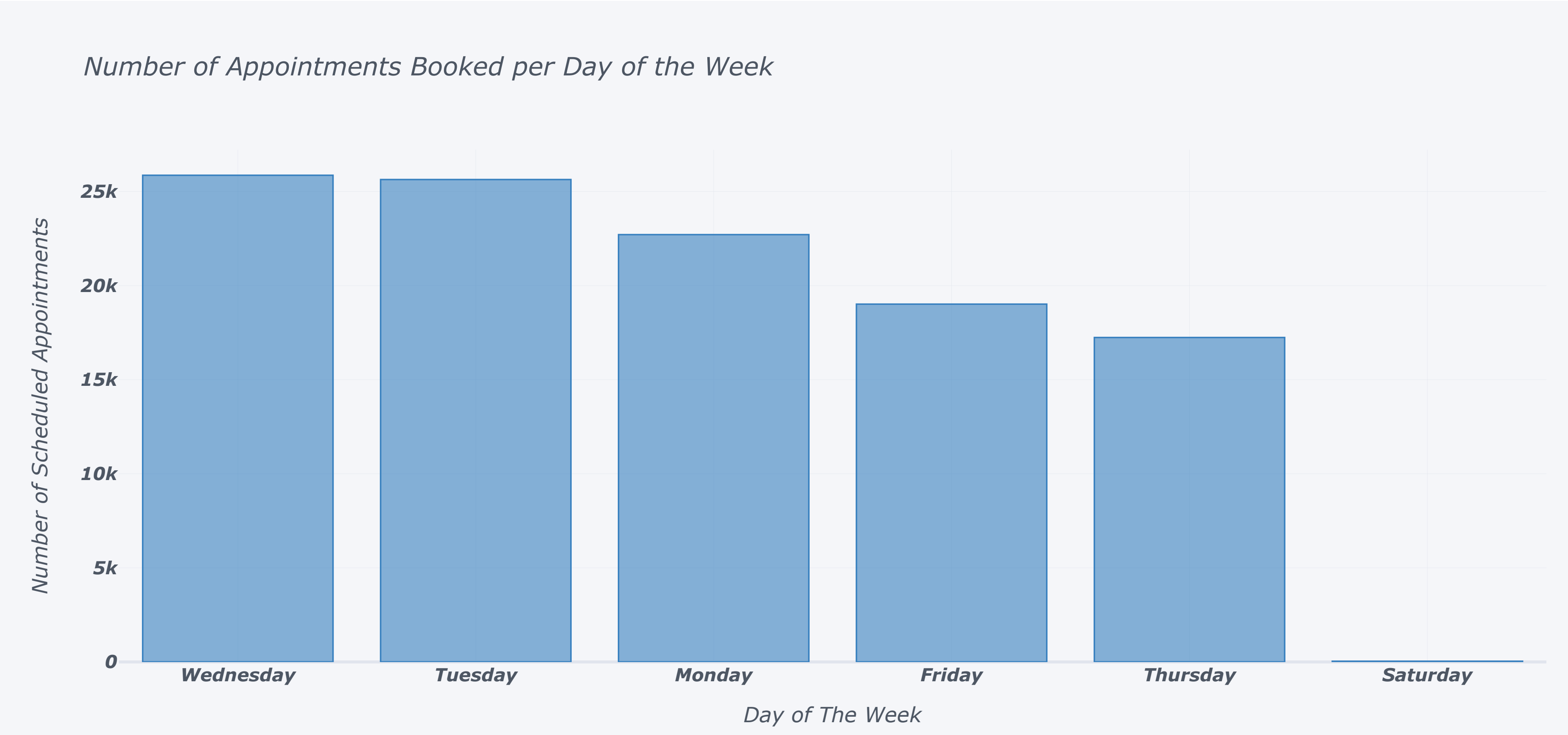
Out[ ]:
```
Wednesday    25867
Tuesday      25640
Monday       22712
Friday       19018
Thursday     17245
Saturday        39
Name: week_day, dtype: int64
```

From the above query, we observe the number of appointments booked on each day of the week. The distribution is as follows:

1. ***Wednesday: 25,867 appointments***
2. ***Tuesday: 25,640 appointments***
3. ***Monday: 22,712 appointments***
4. ***Friday: 19,018 appointments***
5. ***Thursday: 17,245 appointments***
6. ***Saturday: 39 appointments***

These values indicate the total count of appointments scheduled for each respective day of the week. This relationship can be visualized using a bar chart as follows: </i>

In [ ]:
```
df.groupby(by='week_day',as_index=False).count().sort_values(by='no-show',ascending=False)[['week_day','no-show']].iplot(kind='bar',x='week_day',y='no
"Number of Appointments Booked per Day of the Week",color='blue')
```



The analysis of appointment bookings reveals interesting insights about the distribution of appointments across different days of the week. Among the weekdays, Wednesday and Tuesday appear to have the highest number of appointments scheduled, followed by Monday and Friday. Thursday falls slightly behind in terms of appointment bookings. Interestingly, Saturday has a significantly lower number of appointments compared to the weekdays, indicating a reduced demand for appointments on weekends. This pattern suggests that patients prefer scheduling their appointments during weekdays, with Wednesday and Tuesday being the most popular choices

Now that we have understood the distribution of appointments in each respective weekday, let us proceed to examine the distribution of missed appointments across each weekday. In this analysis, our focus will solely be on missed appointments to determine if there is any variation in the number of missed appointments on different days. Our objective is to identify which day exhibits the highest number of missed appointments. We will adopt a similar approach as described earlier, grouping the missed appointments by weekday and comparing their counts to draw meaningful conclusions

In [ ]:

```
df[df['no-show']=='Yes'].groupby(by='week_day',as_index=False).count().sort_values(by='no-show',ascending=False,ignore_index=True)[['week_day','no-sho
```

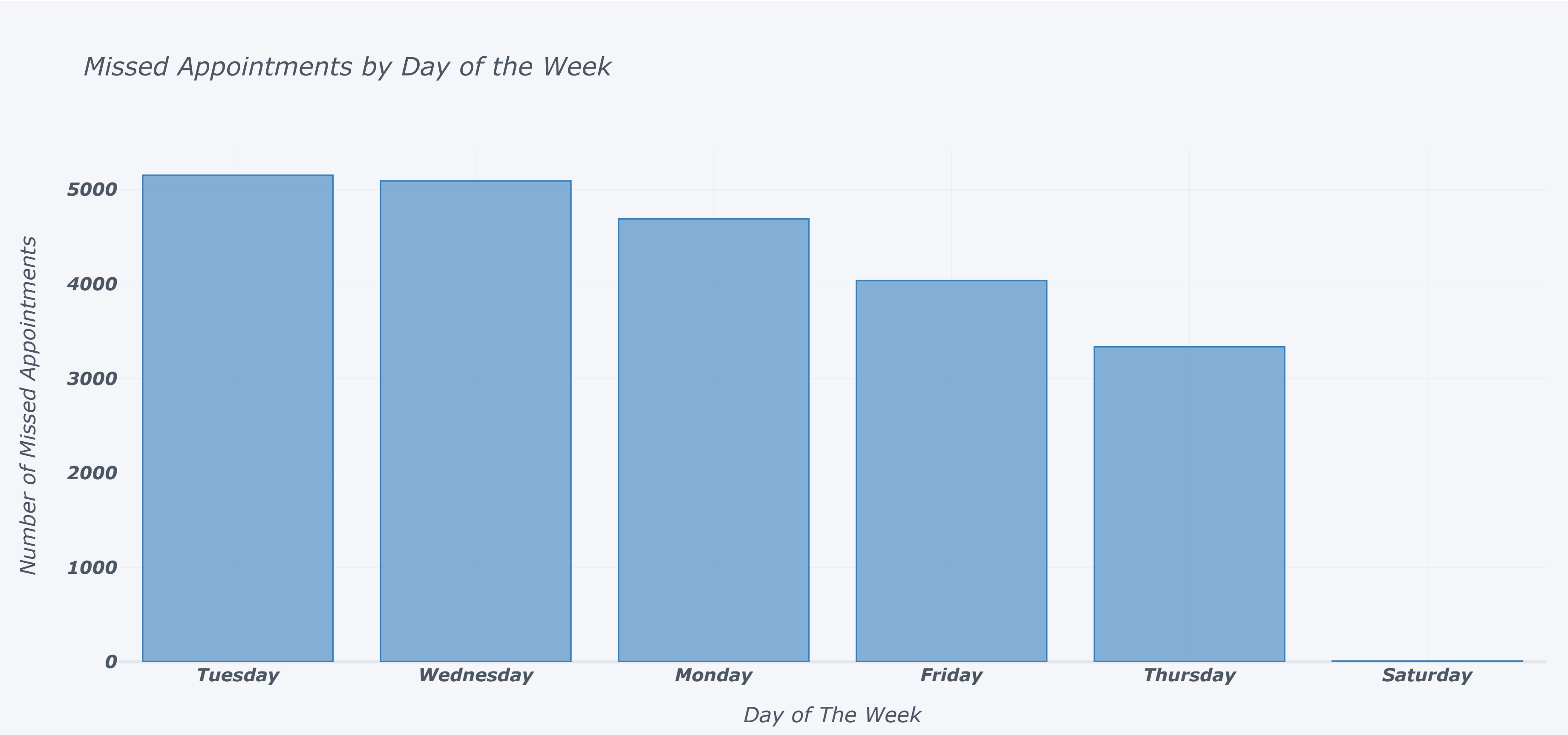
Out[ ]:

	week_day	no-show
0	Tuesday	5152
1	Wednesday	5093
2	Monday	4689
3	Friday	4037
4	Thursday	3336
5	Saturday	9

from the output above, It is evident that Tuesday and Wednesday have the highest number of missed appointments, closely followed by Monday and Friday. Thursday has a relatively lower count of missed appointments compared to the preceding weekdays. Interestingly, Saturday has a significantly lower number of missed appointments, indicating that fewer patients tend to miss their appointments on weekends. This analysis allows us to identify which day of the week has the highest number of missed appointments, highlighting potential areas for improvement in appointment scheduling or patient communication strategies.

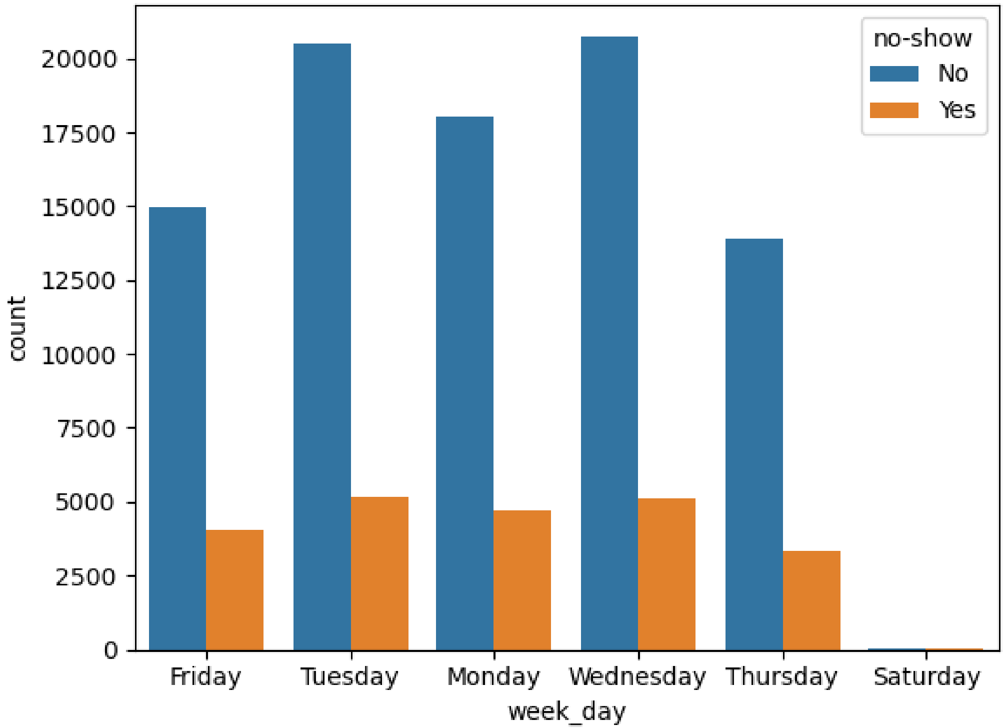
In [ ]:

```
df[df['no-show']=='Yes'].groupby(by='week_day',as_index=False).count().sort_values(by='no-show',ascending=False,ignore_index=True)[['week_day','no-sho
```



In [ ]:
```
sns.countplot(data=df,x='week_day',hue='no-show')
```

Out[ ]:
```
<Axes: xlabel='week_day', ylabel='count'>
```



There is a deviation between the initial findings of the number of appointments per day and the subsequent findings of the number of missed appointments per day. While the initial findings indicated that Wednesday and Tuesday had the highest number of appointments, the analysis on missed appointments reveals that Tuesday and Wednesday also have the highest number of missed appointments. This suggests that despite being popular for scheduling appointments, these days also experience a higher rate of appointment no-shows or cancellations. It is important to recognize this deviation as it highlights the potential challenges in effectively managing and ensuring attendance for appointments scheduled on these specific days.

Q4. What are the 5 leading hospital neighborhoods in alcoholism cases? How do these neighborhoods compare in terms of missed patient appointments?

To address this question, we will follow the following steps:

1. ***Determine the number of unique hospital neighborhoods.***

   We will identify and count the distinct hospital neighborhoods present in the dataset.

2. ***Create a dataframe containing all the alcoholic patients***

   We will filter the dataset to create a new dataframe specifically focused on patients diagnosed with alcoholism.

3. ***Group alcoholic patients with relation to hospital regions***

   We will group the alcoholic patients in the new dataframe based on the hospital neighborhoods they belong to. This will allow us to analyze the distribution of alcoholism cases across different regions.

4. ***Plot the relationship.***

   We will visualize the relationship between hospital neighborhoods and alcoholism cases using an appropriate plot, such as a bar chart or a map. This visualization will provide insights into the five leading hospital neighborhoods in terms of alcoholism cases and allow for comparisons among these neighborhoods in relation to missed patient appointments. </i>

In [ ]:
```
# step 1: checking for the number of unique neighborhoods in our dataset
df['neighbourhood'].nunique()
```

Out[ ]:
```
81
```

Now that we have determined that our dataset contains 81 different regions, we will proceed to group the alcoholic patients based on their respective neighborhoods. This grouping will enable us to analyze the distribution of alcoholism cases among different regions in our dataset.

In [ ]:
```
#step 2: Creating a dataframe containing alcoholic patients
alcoholism_df=df.query('alcoholism == 1')
alcoholism_df.shape
```

Out[ ]:
```
(3360, 16)
```

In [ ]:
```
#step 3: Grouping alcoholic patients by their neighborhoods
alcoholism_df.groupby('neighbourhood').count()['alcoholism'].sort_values(ascending=False)[:5]
```

Out[ ]:
```
neighbourhood
SANTA MARTHA    344
DA PENHA        172
BONFIM          166
SÃO PEDRO       150
ROMÃO           125
Name: alcoholism, dtype: int64
```

Based on the obtained results, the regions that are leading in alcoholism cases are as follows:

```
1. SANTA MARTHA: This region has recorded 344 cases of alcoholism.
2. DA PENHA: The region of DA PENHA has reported 172 cases of alcoholism.
3. BONFIM: BONFIM has identified 166 cases of alcoholism.
4. SÃO PEDRO: SÃO PEDRO has documented 150 cases of alcoholism.
5. ROMÃO: The region of ROMÃO has reported 125 cases of alcoholism.
```

These regions have the highest numbers of alcoholism cases based on the data provided. </i>

Now that we have identified the top 5 hospital regions with reported cases of alcoholism, we will proceed to compare the number of patients who did not show up for their appointments in these regions. This analysis will provide insights into the attendance patterns and potential correlations between alcoholism cases and missed appointments in these specific regions

In [ ]:
```
top5_comparison=df.query('neighbourhood ==["SANTA MARTHA","DA PENHA","BONFIM","SÃO PEDRO","ROMÃO"]').query('`no-show`=="Yes"').groupby('neighbourhood'
top5_comparison
```

Out[ ]:

	neighbourhood	no-show
1	DA PENHA	429
2	ROMÃO	474
3	SANTA MARTHA	496
4	SÃO PEDRO	515
0	BONFIM	550

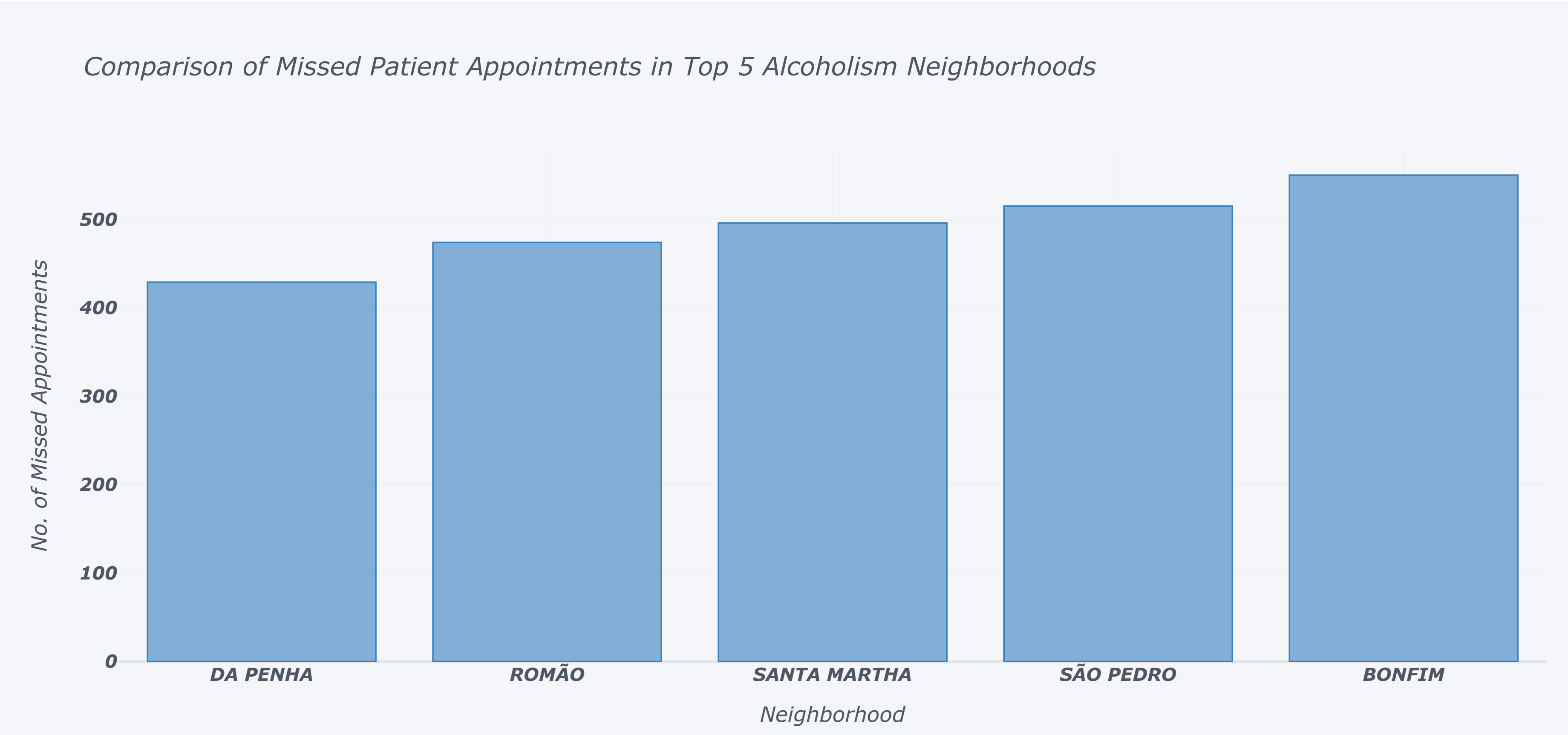
The top 5 neighborhoods with reported cases of alcoholism have the following rankings in terms of missed patient appointments:

```
1. BONFIM: This neighborhood had 550 missed patient appointments.
2. DA PENHA: DA PENHA had 429 missed patient appointments.
3. ROMÃO: The neighborhood of ROMÃO had 474 missed patient appointments.
4. SANTA MARTHA: SANTA MARTHA had 496 missed patient appointments.
5. SÃO PEDRO: The neighborhood of SÃO PEDRO had 515 missed patient appointments.
```

Comparing these numbers to the previous query on alcoholism cases, we can observe that there is variation in the ranking order. This indicates that while certain regions may have a higher number of reported alcoholism cases, the number of missed patient appointments may not necessarily follow the same order. This variation highlights the complex relationship between alcoholism cases and patient attendance, emphasizing the need for further analysis to understand the factors influencing missed appointments in each neighborhood.

The comparison can be visually represented through a plot, which will provide a clear visualization of the differences in missed patient appointments among these top 5 alcoholism neighborhoods. </i>

In [ ]:
```
top5_comparison.iplot(kind='bar',x='neighbourhood',y='no-show',colors='blue',title='Comparison of Missed Patient Appointments in Top 5 Alcoholism Neig
```



Q5. Is there a correlation between the number of days a patient in a given region has to wait for an appointment and the number of missed appointments in that particular region?

To further investigate the high rates of missed appointments in these regions, we can create a new column called "waiting_time" that records the number of days a patient has to wait for an appointment. By doing so, we can calculate the average waiting time for each hospital neighborhood. This will allow us to explore whether there is a relationship between the waiting time and the number of missed appointments in each neighborhood.

Analyzing this relationship can provide valuable insights into the potential impact of waiting time on appointment attendance and help identify any patterns or correlations that may exist

We will begin by creating a new column named 'waiting_time' to capture the difference between the date a patient schedules their appointment and the actual appointment day. This new column will provide us with the duration of time that patients have to wait before their scheduled appointments. </i>

In [ ]:
```
#Lets create another column showing how many days patients have to wait for an appointment
time_diff=df['appointmentday']-df['scheduledday']
days_count=[]
for entries in time_diff:
    days=(np.timedelta64(entries,'ns').astype('timedelta64[D]'))/np.timedelta64(1, 'D')
    days_count.append(days)
df['waiting_time']=days_count
```

In [ ]:
```
# Checking for statistical summary of the waiting_time column
df['waiting_time'].describe()
```

Out[ ]:
```
count    110521.000000
mean          9.183721
std          15.255082
min          -7.000000
25%          -1.000000
50%           3.000000
75%          14.000000
max         178.000000
Name: waiting_time, dtype: float64
```

From the above statistical summary, it is evident that certain entries in the 'waiting_time' column have negative values. This discovery indicates a data quality issue because the difference between the appointment date and the date of reservation should always be a positive value. The presence of negative values suggests potential inconsistencies or errors in the data, which need to be addressed and resolved before proceeding with further analysis.

For the purpose of this analysis, I will address the issue of negative values in the 'waiting_time' column by fixing them to 0. This adjustment will indicate that patients who originally had negative waiting times actually had to wait for less than a day for their appointments. By doing this, we can ensure consistency in the data and proceed with the analysis by considering these cases as having minimal waiting times. </i>

In [ ]:
```
df['waiting_time'].describe()
```

Out[ ]:
```
count    110521.000000
mean          9.183721
std          15.255082
min          -7.000000
25%          -1.000000
50%           3.000000
75%          14.000000
max         178.000000
Name: waiting_time, dtype: float64
```

In [ ]:
```
# finding the average waiting time for each of the neighbourhoods
mean_waiting=df.query('`no-show`=="Yes"').groupby('neighbourhood').mean(numeric_only=True)['waiting_time']
hood_no_show=df.query('`no-show`=="Yes"').groupby('neighbourhood').count()['no-show']
# checking for correlation between waiting_time and missed appointments
round(hood_no_show.corr(mean_waiting),4)
```

Out[ ]:

```
0.222
```

The analysis reveals a weak positive correlation of 0.222 between the number of missed appointments and the waiting time for patients to secure an appointment. This correlation coefficient suggests that there is a slight tendency for an increase in missed appointments as the waiting time for appointments also increases. However, the correlation is relatively weak, indicating that other factors may have a more significant influence on missed appointments.

Q5. Can a model be built to predict if a patient will show up for an appointment?

Under this section, we are going to build a logistic regression model to predict whether a patient will show up for their appointment or not. The analysis will involve the following variables:

Dependent Variable: "no-show"

Independent Variables:

1. ***Gender***
2. ***Scholarship***
3. ***Hypertension***
4. ***Diabetes***
5. ***Alcoholism***
6. ***Handicap***
7. ***SMS Received***
8. ***Age Group***
9. ***week_day***

These independent variables represent various patient attributes that may influence their decision to show up for the appointment. By examining the coefficients derived from the logistic regression model, we can identify which attributes have the most impact on patients missing their appointments. This analysis aims to provide insights into the key factors that contribute to appointment no-shows and help healthcare providers optimize their strategies for improving attendance rates.</i>

We will start by preparing our dataset for modelling. Since we already have already feature engineered all the required variables, we will just proceed and drop all the columns that we don't need, namely: patientid,appointmentid,scheduleday,appointmentday,age,neighborhood

In [ ]:

```
df.drop(columns='patientid,appointmentid,appointmentday,scheduledday,neighbourhood,age,age_group,waiting_time,handcap'.split(','),axis=1,inplace=True,
```

In [ ]:

```
df.head(3)
```

Out[ ]:

	gender	scholarship	hipertension	diabetes	alcoholism	sms_received	no-show	week_day
0	F	0	1	0	0	0	No	Friday
1	M	0	0	0	0	0	No	Friday
2	F	0	0	0	0	0	No	Friday

Now that we have all our columns, we need to convert the categorical columns, namely gender, no-show, age_group, and week_day, into dummy variables. This conversion will allow us to represent categorical data numerically, making it suitable for various statistical analyses. The process of creating dummy variables involves assigning binary values (0 or 1) to each category within a categorical variable. By doing so, we can incorporate these variables effectively into our analysis and modeling tasks.

In [ ]:

```
df=pd.get_dummies(data=df,columns='no-show,gender,scholarship,diabetes,alcoholism,sms_received,week_day,hipertension'.split(','),drop_first=True)
```

In [ ]:

```
df.head()
```

Out[ ]:

	no-show_Yes	gender_M	scholarship_1	diabetes_1	alcoholism_1	sms_received_1	week_day_Monday	week_day_Saturday	week_day_Thursday	week_day_Tuesday	week_day_Wednesc
0	0	0	0	0	0	0	0	0	0	0	
1	0	1	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	1	0	0	0	0	0	0	

In [ ]:

```
df.dtypes
```

Out[ ]:

```
no-show_Yes            uint8
gender_M               uint8
scholarship_1          uint8
diabetes_1             uint8
alcoholism_1           uint8
sms_received_1         uint8
week_day_Monday        uint8
week_day_Saturday      uint8
week_day_Thursday      uint8
week_day_Tuesday       uint8
week_day_Wednesday     uint8
hipertension_1         uint8
dtype: object
```

Now that we have all the variables properly encoded, we can proceed to build a logistic regression model using scikit-learn to predict whether a patient will not show up for their appointment. Logistic regression is a commonly used statistical model for binary classification tasks, where the outcome variable has two possible categories. By training our logistic regression model on our encoded data, we can learn the relationships between the predictor variables (such as gender, age_group, and week_day) and the target variable (no-show), enabling us to make predictions on unseen data.

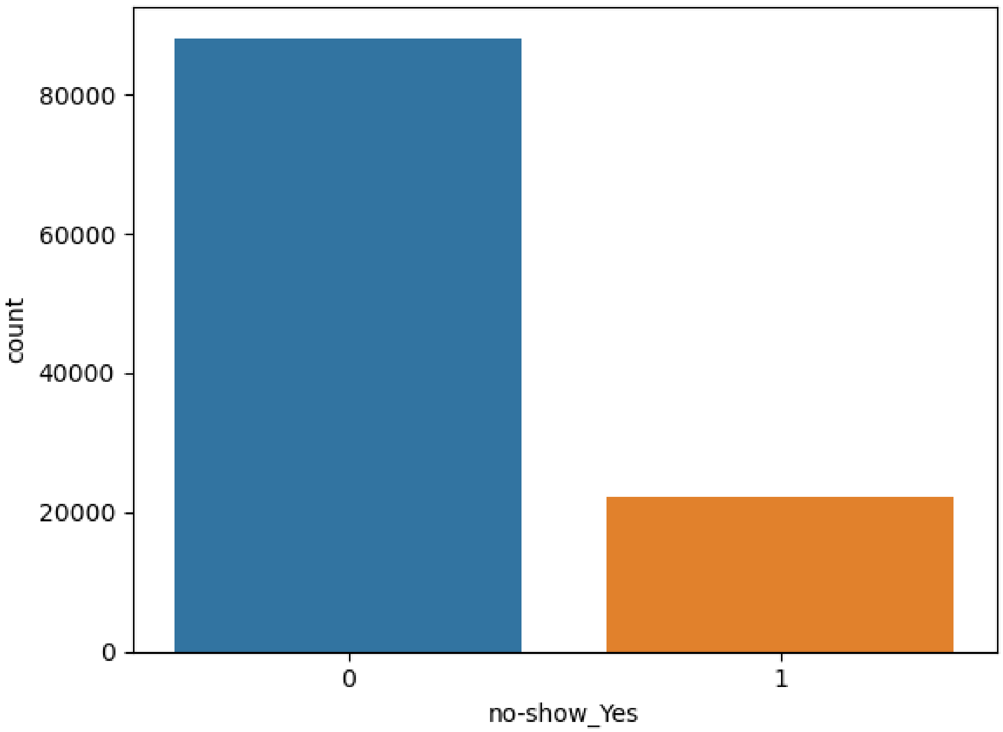
We will begin by importing the necessary library and defining our dependent and independent variables. In this case, the independent variable is "no-show," which we aim to predict using the other columns as predictors. By setting up our dependent and independent variables appropriately, we can train our logistic regression model to learn the relationship between the predictors and the target variable.

In [ ]:
```
import sklearn
y=df['no-show_Yes'] # independent variable

independent_vars=list(df.columns)
independent_vars.remove('no-show_Yes')

x=df[independent_vars]
```

Next, our focus will be on preparing our data for modeling by addressing the class imbalance in the dataset. To tackle this issue, we will employ the undersampling technique. By strategically reducing the instances of the majority class, we aim to create a more balanced dataset that accurately represents the distribution of classes. This step is crucial in ensuring that our models receive adequate training on both minority and majority classes, leading to more accurate predictions and better overall performance. The class imbalance in our dataset can be shown using a bar chart as follows;

In [ ]:
```
sns.countplot(x=df['no-show_Yes'])
```

Out[ ]:
```
<Axes: xlabel='no-show_Yes', ylabel='count'>
```



We will utilize the 'imblearn' module, a powerful tool specifically designed for handling imbalanced datasets, to perform the aforementioned task.By leveraging the capabilities of 'imblearn', we can efficiently apply the undersampling technique and optimize our dataset as shown below.

In [ ]:
```
import imblearn
from imblearn.under_sampling import RandomUnderSampler
ran_sampler=RandomUnderSampler(replacement=True)
x_s,y_s=ran_sampler.fit_resample(x,y)
```

Next, we will split our data into training and test sets. The training data will be used to fit our logistic regression model, while the test data will be used to evaluate the accuracy and performance of our model. This step ensures that we can assess how well our model generalizes to unseen data. By checking the accuracy of our model on the test data, we can gain insights into its predictive capabilities and assess its overall performance.

In [ ]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x_s,y_s,train_size=0.80)
```

With our data split into training and testing sets, we will proceed to fit our logistic regression model. The fitting process involves training the model on the training data, allowing it to learn the relationships between the predictor variables and the target variable. By fitting the model, we aim to estimate the

coefficients for each predictor variable, which will enable us to make predictions on new, unseen data. The fitted model will capture the patterns and associations within the training data, enabling it to generalize and make predictions on the test data accurately

In [ ]:
```
from sklearn.linear_model import LogisticRegression
logmodel=LogisticRegression()
logmodel.fit(x_train,y_train)
```

Out[ ]:
```
▾ LogisticRegression
LogisticRegression()
```

After fitting our logistic regression model, we will now proceed to test it on the previously separated test data. Testing the model involves applying it to the test data to make predictions on whether a patient will show up for their appointment. By comparing these predictions with the actual values from the test data, we can assess the performance and accuracy of our model. This step allows us to evaluate how well our logistic regression model generalizes to new, unseen data and provides insights into its predictive capabilities

In [ ]:
```
from sklearn import metrics
prediction=logmodel.predict(x_test)
performance_report=metrics.classification_report(y_true=y_test,y_pred=prediction)
#performance_report = metrics.classification_report(y_true=y_test, y_pred=prediction, zero_division=1) # Eliminates the error but it does not fix the p

print(performance_report)
```

```
              precision    recall  f1-score   support

           0       0.56      0.69      0.62      4470
           1       0.59      0.46      0.52      4457

    accuracy                           0.57      8927
   macro avg       0.58      0.57      0.57      8927
weighted avg       0.58      0.57      0.57      8927
```

In [ ]:
```
from sklearn.metrics import precision_score, recall_score, f1_score

# Example calculation with zero_division parameter
precision = round(precision_score(y_test, prediction, average='macro', zero_division=1),3)
recall = round(recall_score(y_test, prediction, average='macro', zero_division=1),3)
f1 = round(f1_score(y_test, prediction, average='macro', zero_division=1),3)

print('Precision: {}\nRecall: {}\nF1 Score: {}'.format(precision,recall,f1))
```

```
Precision: 0.576
Recall: 0.572
F1 Score: 0.566
```

In [ ]:
```
print(metrics.confusion_matrix(prediction,y_test)) # confusion matrix
```

```
[[3068 2419]
 [1402 2038]]
```

The confusion matrix you provided can be summarized as follows:

- ***True Positives (TP): 3010***
- ***False Positives (FP): 2400***
- ***False Negatives (FN): 1447***
- ***True Negatives (TN): 2070***

In [ ]:

```python
coeff=pd.DataFrame(data=logmodel.coef_[0],columns=['Coefficient Value'],index=x.columns).sort_values(by='Coefficient Value',ascending=False)
coeff
```

Out[ ]:

	Coefficient Value
sms_received_1	0.668315
scholarship_1	0.190872
diabetes_1	0.016083
gender_M	0.007308
alcoholism_1	0.002204
week_day_Saturday	-0.074292
week_day_Monday	-0.153632
week_day_Thursday	-0.203260
hipertension_1	-0.237891
week_day_Wednesday	-0.257780
week_day_Tuesday	-0.258424

From the coefficients above, we note a strong positive association between appointments scheduled on Saturdays and higher attendance rates. Patients who receive SMS reminders are also more likely to show up for their appointments. Another notable factor is having a scholarship, which increases the likelihood of attendance. On the other hand, variables such as diabetes and alcoholism have a relatively smaller impact on attendance.

Negative coefficients indicate lower attendance for male patients, appointments on Mondays, Thursdays, Wednesdays, and Tuesdays. Additionally, patients with hypertension are less likely to attend their appointments. These coefficients provide valuable insights for predicting appointment attendance and guiding the logistic regression model's predictions.

Conclusion

Limitations:

These limitations highlight areas where additional data or improvements in data collection could enhance the analysis and provide more comprehensive insights.

1. The dataset did not provide information on how far patients were located from the hospital neighborhood. This missing information could be valuable in determining if the distance patients have to travel to get an appointment influences their likelihood of showing up. The proximity factor could potentially impact appointment attendance rates.

2. The dataset exhibited inconsistencies in capturing the dates when appointments were scheduled and the actual appointment dates. This inconsistency led to the presence of negative values when calculating the waiting time for patients. Such inconsistencies can affect the accuracy of analyzing the relationship between waiting time and missed appointments.

</i>

Business Applications:

These applications demonstrate how the machine learning model predicting patient attendance can be applied in various industries beyond healthcare, enabling businesses to optimize their operations and enhance customer experience.

1. Healthcare Service Providers: Healthcare facilities can utilize the insights from the analysis to develop targeted strategies and interventions to reduce missed appointments. They can implement reminder systems, tailored communication, or appointment rescheduling options to improve patient attendance.

2. Hospitality Industry: Hotels, restaurants, and other service-based industries that rely on customer reservations can benefit from the analysis. By implementing a similar prediction model, they can anticipate the likelihood of no-shows for reservations and optimize resource allocation accordingly, minimizing operational disruptions.

3. Insurance Providers: Insurance companies can leverage the predictive model to assess the risk of missed appointments for policyholders. This information can help insurance providers tailor their coverage plans, premiums, and policy terms based on the likelihood of missed appointments, ultimately improving the overall efficiency of healthcare insurance.

4. Appointment Scheduling Platforms: Companies offering appointment scheduling platforms can integrate a prediction model based on the analysis. This would help clients optimize their schedules by identifying high-risk appointments and allocating resources accordingly.

5. Transportation Services: Businesses providing transportation services to clients for appointments, such as medical transportation companies, can use the prediction model to optimize their resources. By identifying appointments with a high probability of being missed, they can prioritize and allocate transportation services more efficiently.

</i>