TIME SERIES ANALYSIS OF MEDICAL APPOINMENTS USING PYTHON AND SQL

OVERVIEW

This project involves analyzing real-world medical appointment data through Time Series Analysis. The tasks include dataset cleaning, comprehensive analysis, and extracting insights using Python and MySQL.

PROJECT DESCRIPTION

The "Healthcare Appointments Analytics" project is a comprehensive data analysis initiative that delves into the intricacies of medical appointments and explores factors influencing patient attendance and no-show rates. By leveraging advanced analytical techniques, the project aims to uncover valuable insights that can positively impact patient engagement and resource allocation in healthcare settings.

TASKS

1. How many values are there in the given dataset
2. Count the number of appointments for each day in the given dataset
3. Calculate the average number of appointments(Set to nearest whole number) per day in the given dataset.
4. Find the day with the highest number of appointments in the given dataset.
5. Calculate the monthly average number of appointments in the given dataset.
6. Find the month with the highest number of appointments in the given dataset.
7. Calculate the weekly average number of appointments in the given dataset.
8. Find the week with the highest number of appointments in the given dataset.
9. What is the distribution of appointments based on gender in the dataset?
10. Calculate the number of appointments per weekday in the given dataset.
11. Calculate the average time between scheduling and the appointment day in the given dataset.

In [42]:
```
# import the necessary package
import pandas as pd
import sqlite3
```

In [3]:
```
# load in the hospital dataset
df = pd.read_csv('Hospital_patients_datasets.csv')
```

DATA INSPECTION

In [5]:
```
# check the dataframe
df.head()
```

Out[5]:

	PatientId	AppointmentID	Gender	ScheduledDay	AppointmentDay	Age	Neighbourhood	Scholarship	Hipertension	Diabe
0	2.987250e+13	5642903	F	2016-04-29T18:38:08Z	2016-04-29T00:00:00Z	62	JARDIM DA PENHA	0	1	
1	5.589980e+14	5642503	M	2016-04-29T16:08:27Z	2016-04-29T00:00:00Z	56	JARDIM DA PENHA	0	0	
2	4.262960e+12	5642549	F	2016-04-29T16:19:04Z	2016-04-29T00:00:00Z	62	MATA DA PRAIA	0	0	
3	8.679510e+11	5642828	F	2016-04-29T17:29:31Z	2016-04-29T00:00:00Z	8	PONTAL DE CAMBURI	0	0	
4	8.841190e+12	5642494	F	2016-04-29T16:07:23Z	2016-04-29T00:00:00Z	56	JARDIM DA PENHA	0	1	

In [76]:
```
# check the info about the dataset
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
```

In [10]:
```python
# check the shape of the dataset
shape = list(df.shape)
print(shape)
print('There are ', shape[0], 'rows and ', shape[1], ' in the dataframe')
```

```
[110527, 14]
There are  110527 rows and  14  in the dataframe
```

In [13]:
```python
# check for null values and sum any
null = df.isna().any().sum()
df.isna().sum()
print('There are',null ,'null values int the dataset')
```

```
There are 0 null values int the dataset
```

In [16]:
```python
# check for duplicates and some across all columns
dup = df[:].duplicated().sum()
print(dup)
print('There are', dup, 'Duplicate value in the dataset')
```

```
0
There are 0 Duplicate value in the dataset
```

DATA CLEANING

In [17]:
```python
# convert the datatype of day columns to datetime
df['ScheduledDay'] = pd.to_datetime(df['ScheduledDay']).dt.date.astype('datetime64[ns]')
df['AppointmentDay'] = pd.to_datetime(df['AppointmentDay']).dt.date.astype('datetime64[ns]')
```

In [18]:
```python
# rename some particular columns
df = df.rename( columns={'Hipertension': 'Hypertension',
                         'Handcap': 'Handicap',
                         'SMS_received': 'SMSRecevied',
                         'No-show': 'NoShow'})
```

In [19]:
```python
# drop columns that are not of interest in the dataframe
df.drop(['PatientId'], axis=1, inplace=True)
df.drop(['AppointmentID'], axis=1, inplace=True)
df.drop(['Neighbourhood'], axis=1, inplace=True)
```

In [20]:
```python
# drop age rows with 0 values
df.drop(df[df['Age'] == 0].index, inplace=True)
```

In [21]:
```python
# Generating labels for age intervals (e.g., '1 - 20', '21 - 40', etc.)
labels = ["{0} - {1}".format(i, i + 20) for i in range(1, 118, 20)]

# create the bins
bins = range(1, 130, 20)

# use the cut function to categorize the age into groups
df['Age_group'] = pd.cut(df['Age'], bins=bins, labels=labels, right=False)
```

In [22]:
```python
# drop the Age column
df.drop('Age', axis=1, inplace=True)
```

In [23]:
```python
# convert 'NoShow' values into binary values (1 for 'Yes' and 0 for 'No').
df['NoShow'] = df['NoShow'].map({'Yes':1, 'No':0})
```

In [24]:
```
# check the cleaned dataframe
df.head()
```

Out[24]:

	Gender	ScheduledDay	AppointmentDay	Scholarship	Hypertension	Diabetes	Alcoholism	Handicap	SMSRecevied	NoShow
0	F	2016-04-29	2016-04-29	0	1	0	0	0	0	0
1	M	2016-04-29	2016-04-29	0	0	0	0	0	0	0
2	F	2016-04-29	2016-04-29	0	0	0	0	0	0	0
3	F	2016-04-29	2016-04-29	0	0	0	0	0	0	0
4	F	2016-04-29	2016-04-29	0	1	1	0	0	0	0

DATA EXPLORATION

In [25]:
```
# check the distribution of Age group
df['Age_group'].value_counts()
```

Out[25]:
```
Age_group
41 - 61      30081
21 - 41      28835
1 - 21       28309
61 - 81      16910
81 - 101      2845
101 - 121        7
Name: count, dtype: int64
```

The dataset consists of most obeservations from people of age group 41-61 and least from 101-121

In [40]:
```
# check the distribution of Gender
df['Gender'].value_counts()
```

Out[40]:
```
Gender
F    70119
M    36869
Name: count, dtype: int64
```

The dataset consists of more female than male

In [27]:
```
# check the distribution of NoShow
df['NoShow'].value_counts()
```

Out[27]:
```
NoShow
0    85308
1    21680
Name: count, dtype: int64
```

In [ ]:
```
# export the cleaned dataset and set the index=false
df.to_csv('patients.csv', index=False)
```

QUERRYING USING SQLITE

In [92]:
```
#set the connection
cnn = sqlite3.connect('jupyter_sql_tutorial.db')
```

In [ ]:
```
# load in the already cleaned and saved dataset
df.to_sql('patients', cnn)
```

In [ ]:
```
%load_ext sql
```

In [ ]:
```
%sql sqlite:///jupyter_sql_tutorial.db
```

TASK 1: How many values are there in the given dataset?

In [ ]:
```
%%sql


SELECT
        COUNT(*) as count
FROM
        patients
```

```
 * sqlite:///jupyter_sql_tutorial.db
Done.
```

Out[ ]:

count
106988

106,989 observations sucessfully loaded into the database

TASK 2: Count the number of appointments for each day in the given dataset:

In [ ]:
```
%%sql


SELECT
            AppointmentDAY as appointmentday,
            count(AppointmentDAY) as total_appointment
FROM
            patients
GROUP BY
            1
```

```
 * sqlite:///jupyter_sql_tutorial.db
Done.
```

Out[ ]:

appointmentday	total_appointment
2016-04-29 00:00:00	3104
2016-05-02 00:00:00	4214
2016-05-03 00:00:00	4129
2016-05-04 00:00:00	4048
2016-05-05 00:00:00	4113
2016-05-06 00:00:00	3791
2016-05-09 00:00:00	4352
2016-05-10 00:00:00	4177
2016-05-11 00:00:00	4347
2016-05-12 00:00:00	4233
2016-05-13 00:00:00	3885
2016-05-14 00:00:00	39
2016-05-16 00:00:00	4449
2016-05-17 00:00:00	4227
2016-05-18 00:00:00	4220
2016-05-19 00:00:00	4109
2016-05-20 00:00:00	3707
2016-05-24 00:00:00	3876
2016-05-25 00:00:00	3768
2016-05-30 00:00:00	4360
2016-05-31 00:00:00	4158
2016-06-01 00:00:00	4351
2016-06-02 00:00:00	4204

appointmentday	total_appointment
2016-06-03 00:00:00	3978
2016-06-06 00:00:00	4529
2016-06-07 00:00:00	4264
2016-06-08 00:00:00	4356

The table above shows the number of appointments each day

TASK 3: Calculate the average number of appointments(Set to nearest whole number) per day in the given dataset.

In [ ]:
```
%%sql


SELECT
        ROUND(AVG(T1.count),0) as average_appointment

FROM
        (SELECT
                AppointmentDAY as appointmentday,
                COUNT(AppointmentDAY) as count
        FROM
                patients
        GROUP BY
                1) as T1
```

```
 * sqlite:///jupyter_sql_tutorial.db
Done.
```

Out[ ]:

average_appointment
3963.0

The average number of appointments per day is 3963

TASK 4: Find the day with the highest number of appointments in the given dataset.

In [ ]:
```
%%sql


SELECT
        AppointmentDAY, COUNT(AppointmentDAY) as number_of_appointments
FROM
        patients
GROUP BY
        1
HAVING
        COUNT(AppointmentDAY) = (SELECT
                                        MAX(T1.daycount)
                                FROM
                                        (SELECT
                                                AppointmentDAY,
                                                COUNT(AppointmentDAY) as daycount
                                        FROM
                                                patients
                                        GROUP BY
                                                AppointmentDAY) as T1)
```

```
 * sqlite:///jupyter_sql_tutorial.db
Done.
```

Out[ ]:

AppointmentDay	number_of_appointments
2016-06-06 00:00:00	4529

06-06-2016 is the day with the highest number of appointment (4,526)

TASK 5: Calculate the monthly average number of appointments in the given dataset.

Hint : Use 'DATE_FORMAT()' function. But the function is not supported in Sqlite, we rather use STRFTIME function.

In [ ]:
```
%%sql

SELECT
```

```
            STRFTIME('%Y-%m', AppointmentDay) AS year_and_month,
            AVG(AppointmentDay) as monthly_avg
FROM
            patients
GROUP BY
            1
```

```
 * sqlite:///jupyter_sql_tutorial.db
Done.
```

Out[ ]:

year_and_month	monthly_avg
2016-04	2016.0
2016-05	2016.0
2016-06	2016.0

The table above shows the monthly average number of appointments

TASK 6: Find the month with the highest number of appointments in the given dataset.

In [ ]:

```
%%sql

SELECT
        STRFTIME('%Y-%m', AppointmentDay) as month,
        COUNT(AppointmentDAY) as appointment_no
FROM
        patients
GROUP BY
        1
HAVING
        COUNT(AppointmentDAY) = (SELECT
                                        MAX(T1.daycount)
                                FROM
                                        (SELECT
                                                STRFTIME('%Y-%m', AppointmentDay) AS year_and_month,
                                                COUNT(AppointmentDAY) as daycount
                                        FROM
                                                patients
```

```
                                        GROUP BY
                                                year_and_month) as T1)
```

```
 * sqlite:///jupyter_sql_tutorial.db
Done.
```

Out[ ]:

month	appointment_no
2016-05	78202

15-2016 is the month with the highest number of appointments (78,202)

TASK 7: Calculate the weekly average number of appointments in the given dataset.

In [ ]:

```
%%sql

 SELECT
        STRFTIME('%Y', AppointmentDay) AS year,
        STRFTIME('%m', AppointmentDay) AS week,
        COUNT(AppointmentDAY) as week_average
 FROM
        patients
 GROUP BY
        1,2
```

```
 * sqlite:///jupyter_sql_tutorial.db
Done.
```

Out[ ]:

year	week	week_average
2016	04	3104
2016	05	78202
2016	06	25682

The table above shows the weekly average of appointments in the dataset

TASK 8: Find the week with the highest number of appointments in the given dataset.

In [ ]:
```sql
%%sql

SELECT
        STRFTIME('%Y', AppointmentDay) as year,
        STRFTIME('%w', AppointmentDay) as week,
        COUNT(AppointmentDAY) as appointment_no
FROM
        patients
GROUP BY
        1,2
HAVING
        COUNT(AppointmentDAY) = (SELECT
                                        MAX(T1.daycount)
                                FROM
                                        (SELECT
                                                STRFTIME('%w', AppointmentDay) as week,
                                                COUNT(AppointmentDAY) as daycount
                                        FROM
                                                patients
                                        GROUP BY
                                                1) as T1)
```

```
 * sqlite:///jupyter_sql_tutorial.db
Done.
```

Out[ ]:

year	week	appointment_no
2016	3	25090

The third week in the date range within the dataset has the highest number of appointments (25,090)

TASK 9: What is the distribution of appointments based on gender in the dataset?

In [ ]:
```sql
%%sql

SELECT
        Gender, count(Gender) as gender_count
FROM
        patients
```

```
GROUP BY
        1
```

```
 * sqlite:///jupyter_sql_tutorial.db
Done.
```

Out[ ]:

Gender	gender_count
F	70119
M	36869

As said before, There are more Female than Male in the dataset

TASK 10: Calculate the number of appointments per weekday in the given dataset. Order the appointment counts in descending.

Hint : Use 'DAYNAME()' function. this dunction is not supported in sqlite3, we rathe use loop.

In [ ]:

```
%%sql

SELECT
        case cast(strftime('%w', AppointmentDay) as integer)
           when 0 then 'Sunday'
           when 1 then 'Monday'
           when 2 then 'Tuesday'
           when 3 then 'Wednesday'
           when 4 then 'Thursday'
           when 5 then 'Friday'
           else 'Saturday' end as day,
        count(*) as day_count
FROM
        patients
GROUP BY
        1
ORDER BY
        2 DESC
```

```
 * sqlite:///jupyter_sql_tutorial.db
Done.
```

Out[ ]:

day	day_count
Wednesday	25090
Tuesday	24831
Monday	21904
Friday	18465
Thursday	16659
Saturday	39

The table above shows the weekly number of apppointments. wenesday has the highest while Saturday has the lowest number of appointment

TASK 11: Calculate the average time between scheduling and the appointment day in the given dataset. Set to nearest whole number

In [ ]:
```
%%sql

SELECT
        ROUND(AVG(T1.time_diff),0) as avg_time
FROM
        (SELECT
                julianday(AppointmentDay) -  julianday(ScheduledDay) as time_diff
                FROM
                patients) as T1
```

```
 * sqlite:///jupyter_sql_tutorial.db
Done.
```

Out[ ]:

avg_time
10.0

The average time between schedule and appointment day is 10 days

CONCLUSION

The dataset is wrangled entailing, Data loading, Data inspection and Data cleaning before exploring it. The cleaned dataset was saved as a csv file before importing into the sqlite3 database. Solutions were provided to the 11 imposed Tasks using (Task, Code, Observation) method. That is, each task was solved individually.