1. Introduction

The purpose of this document is to describe the technical approach and findings of the analysts that was performed for the National Health Services (NHS).

The NHS incurs significant, potentially avoidable, costs when patients miss appointments. Reducing or eliminating missed appointments would be beneficial financially as well as socially. At this stage of the project the two main questions posed by the NHS are:

- Has there been adequate staff and capacity in the networks?
- What was the actual utilisation of resources?

The analysis utilised Python to explore the available data and extract meaningful insights. We worked in a team of data analyst.

2. Initial findings

The initial analysis started with preparing workstation in Python: importing all necessary libraries and files.

```
# Import the necessary libraries.
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Import the necessary data sets.
ad = pd.read_csv('actual_duration.csv')
ar = pd.read_csv('appointments_regional.csv')
nc = pd.read_excel('national_categories.xlsx')
```

Next, we needed to explore the datasets for datatypes, columns, statistical summary, missing values and etc.

```
# View dataset actual duration.
print(ad.shape)
print(ad.dtypes)
print(ad.columns)

# View dataset regional appointments.
print(ar.shape)
print(ar.dtypes)
print(ar.columns)

# View dataset national categories.
print(nc.shape)
print(nc.dtypes)
print(nc.columns)
```

```
# View whole data.
ad.head()
```

```
# View whole data.
ar.head()
```

```
# View whole data.
nc.head()
```

```
# View descriptive statistics actual duration.
ad.describe()
```

```
# View descriptive statistics regional appointments.
ar.describe()
```

```
# View descriptive statistics national categories.
nc.describe()
```

```
# View dataframes.
ad.info()
ar.info()
nc.info()
```

```
# View missing values.
ad.isnull().sum()
ar.isnull().sum()
nc.isnull().sum()
```

There were no ‘strange’ things in the files, like missing values/column names or data types.

Next, we investigated if merging the files make sense.

```
# Investigate if sum of appointments on certin day is the same in every file. Start with ad.
item_check_ad =  ad.groupby('appointment_date')[['count_of_appointments']].sum() \
.sort_values('count_of_appointments', ascending = False)

# Show top 10 of the table.
item_check_ad.head(10)
```

```
# Investigate if sum of appointments on certin day is the same in every file. Continue with nc.
item_check_nc=  nc.groupby('appointment_date')[['count_of_appointments']].sum() \
.sort_values('count_of_appointments', ascending = False)

# Show top 10 of the table.
item_check_nc.head(10)
```

Concusion:

Merging of the 3 files does not make sense because:

- granularity is not the same of the files (ar file in on date level)
- the number of appointments is not the same on days level (data integrity)

Now, that we ensured that the datasets are correct, the following questions were investigated:

- How many locations are there in the data set?
- What are the five locations with the highest number of records?
- How many service settings, context types, national categories, and appointment statuses are there?
- How many appointments are attended vs not attended overall?

The following scripts were used to answer these questions:

```
# View number of locations.
count_location = ad['sub_icb_location_code'].nunique()
print(f'There are {count_location} number of locations.')
```

```
There are 106 number of locations.
```

```
# View top five locations with the highest number of appointments.
print("The top five locations with the highest number of appointments are:")
print(ad.groupby(["sub_icb_location_name"])["count_of_appointments"].sum().nlargest(5))
```

```
The top five locations with the highest number of appointments are:
sub_icb_location_name
NHS North West London ICB - W2U3Z              6976986
NHS North East London ICB - A3A8R              5341883
NHS Kent and Medway ICB - 91Q                  5209641
NHS Hampshire and Isle Of Wight ICB - D9Y0V    4712737
NHS South East London ICB - 72Q                4360079
Name: count_of_appointments, dtype: int64
```

By running the following scripts to find out the number of service setting etc, we were not sure if some strange category is in there. It is important to view what those categories are and remove the unmapped, inconsistent mapping and unknown categories:

```
# View number of service setting.
count_service_setting =nc['service_setting'].nunique()
print(f'It seems that there are {count_service_setting} service settings.')

# View number of context types.
count_context_types =nc ['context_type'].nunique()
print(f'It seems that there are {count_context_types} context types.')

# View number of national categories.
count_national_categories =nc ['national_category'].nunique()
print(f'It seems that there are {count_national_categories} national categories.')

# View number of appointment status.
count_appointment_status =ar ['appointment_status'].nunique()
print(f'It seems that there are {count_appointment_status} appointment status.')
```

```
It seems that there are 5 service settings.
It seems that there are 3 context types.
It seems that there are 18 national categories.
It seems that there are 3 appointment status.
```

```
# Display list of service setting.
list_service_settings = pd.unique(nc.service_setting)
print(f'The service settings are: {list_service_settings} ' )

# Display lis of context types.
list_context_types = pd.unique(nc.context_type)
print(f'The context types are: {list_context_types} ' )

# Display lis of national categories.
list_appointment_status = pd.unique(ar.appointment_status)
print(f'The appointment statuses are: {list_appointment_status} ' )

# Display lis of appointment status.
list_national_categories = pd.unique(nc.national_category)
print(f'The national categories are: {list_national_categories} ' )
```

Output:

Conclusion: excluding unmapped, inconsistent mapping and unknown categories:

There are 4 service settings:

- Primary Care Network
- General Practice
- Extended Access Provision
- Other

There is 1 context type:

- Care Related Encounter

There are 16 national categories:

- Patient contact during Care Home Round
- Planned Clinics
- Home Visit
- General Consultation Acute
- Structured Medication Review
- Care Home Visit
- Clinical Triage
- Planned Clinical Procedure
- Care Home Needs Assessment & Personalised Care and Support Planning
- General Consultation Routine
- Service provided by organisation external to the practice
- Unplanned Clinical Activity
- Social Prescribing Service
- Non-contractual chargeable work
- Group Consultation and Group Education
- Walk-in

There are 2 appointment status:

- Attended
- DNA

How many appointments are attended vs not attended overall?

```
# View sum of appointments attended vs not attended.
app_status = ar.groupby('appointment_status')[['count_of_appointments']].sum()

# View result.
app_status
```

```
app_status['percentage'] = (app_status['count_of_appointments'] \
/ app_status['count_of_appointments'].sum()*100).round(2)

# View result.
app_status
```

	count_of_appointments	percentage
appointment_status		
Attended	677755876	91.24
DNA	30911233	4.16
Unknown	34137416	4.60

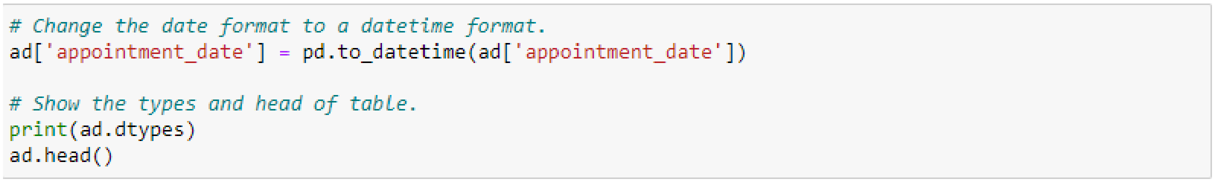
Conclusions:
Currently about 91% of the appointments are attended, rest are not attended (4%) or unknown (5%).

Next we were interested in:

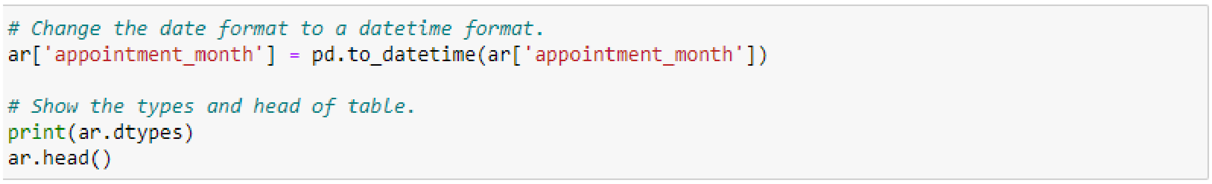
- Between what dates were appointments scheduled?
- Which service setting reported the most appointments in the busiest location and in the busiest time frame?
- Which month had the highest number of appointments?
- What was the total number of records per month?

By answering the questions, we could narrow down the dataset for the further analysis.

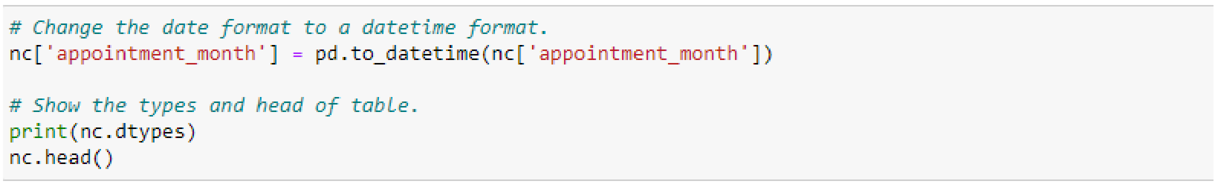
In order to be able to answer the first question, the date format needed to be changed and a new module imported, see below the scrips. After that we used the agg(min) and agg(max) functions to find out the lowest and maximum dates in the datasets.



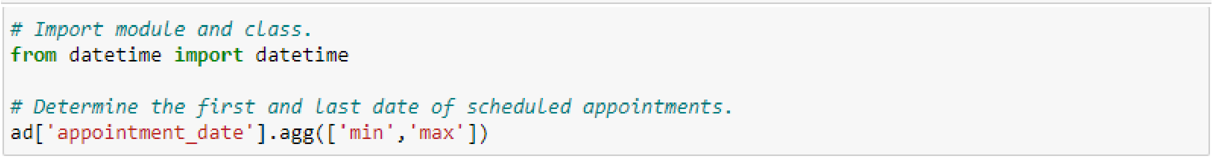
```
# Change the date format to a datetime format.
ad['appointment_date'] = pd.to_datetime(ad['appointment_date'])

# Show the types and head of table.
print(ad.dtypes)
ad.head()
```



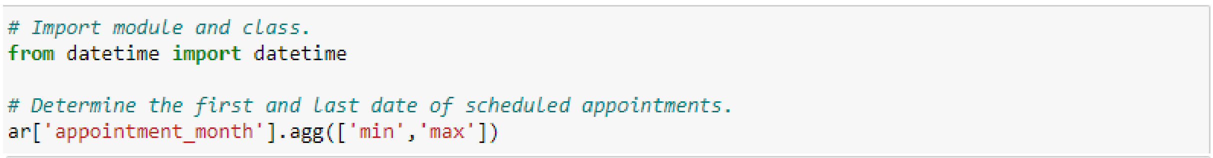
```
# Change the date format to a datetime format.
ar['appointment_month'] = pd.to_datetime(ar['appointment_month'])

# Show the types and head of table.
print(ar.dtypes)
ar.head()
```



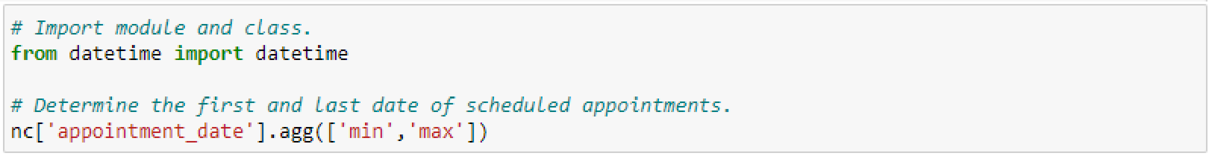
```
# Change the date format to a datetime format.
nc['appointment_month'] = pd.to_datetime(nc['appointment_month'])

# Show the types and head of table.
print(nc.dtypes)
nc.head()
```



```
# Import module and class.
from datetime import datetime

# Determine the first and last date of scheduled appointments.
ad['appointment_date'].agg(['min','max'])
```



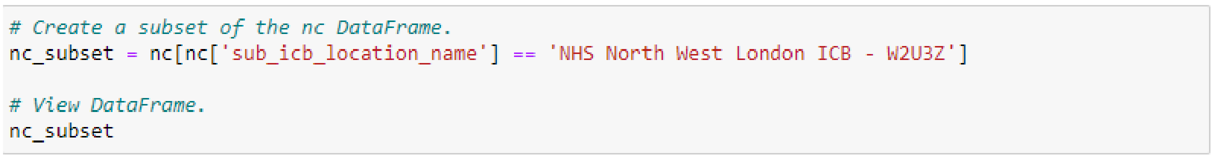
```
# Import module and class.
from datetime import datetime

# Determine the first and last date of scheduled appointments.
ar['appointment_month'].agg(['min','max'])
```



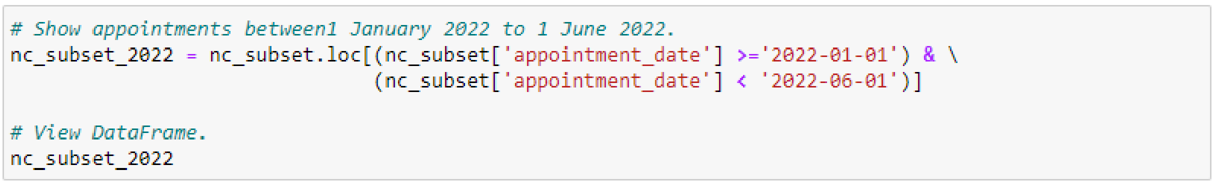
```
# Import module and class.
from datetime import datetime

# Determine the first and last date of scheduled appointments.
nc['appointment_date'].agg(['min','max'])
```

Conclusion: Appointments were scheduled between 1st of January 2020 and 30th of June 2022.

In order to answer the question: which service setting reported the most appointments in the busiest period, we needed to create a subset: (using nc dataset because that contains this info)



```
# Create a subset of the nc DataFrame.
nc_subset = nc[nc['sub_icb_location_name'] == 'NHS North West London ICB - W2U3Z']

# View DataFrame.
nc_subset
```



```
# Show appointments between1 January 2022 to 1 June 2022.
nc_subset_2022 = nc_subset.loc[(nc_subset['appointment_date'] >='2022-01-01') & \
                               (nc_subset['appointment_date'] < '2022-06-01')]

# View DataFrame.
nc_subset_2022
```

```
# Calculate number of appointments per service settings.
nc_subset_appointment = nc_subset_2022.groupby('service_setting')[['count_of_appointments']].sum() \
.sort_values('count_of_appointments', ascending = False)

# View number of appointments per service settings.
nc_subset_appointment
```

Alternative/other script for the above:

```
# Caclulate number of appointments per service settings. Show the same in a different way:
nc_subset_appointment_pivot = pd.pivot_table(nc_subset_2022, values='count_of_appointments', \
                                      index=['service_setting'] ,margins=bool, aggfunc='sum')

# View pivot table.
nc_subset_appointment_pivot
```

Conclusion: General Practice service setting reported the most appointments in North West London from 1 January to 1 June 2022.

In order to answer the question: Which month had the highest number of appointments, we investigated in all 3 files using grouping function with sum and sorted the table to see the highest one on top.

```
# Number of appointments per month ar file.
ar_appointments = ar.groupby('appointment_month')[['count_of_appointments']].sum() \
.sort_values('count_of_appointments', ascending = False)

# View result.
ar_appointments.head()
```

```
# Number of appointments per month ad file.
ad_monthly = ad.groupby([ad['appointment_date'].dt.year, ad['appointment_date'].dt.month]) \
[['count_of_appointments']].sum().sort_values('count_of_appointments', ascending = False)

# View result.
ad_monthly.head()
```

```
# Number of appointments per month nc file.
nc_appointments = nc.groupby('appointment_month')[['count_of_appointments']].sum() \
.sort_values('count_of_appointments', ascending = False)

# View result.
nc_appointments.head()
```

Conclusion: In the month of November 2021 had the highest number of appointments.

Last, we wanted to see which file contains the most records per months (below showing example for 1 file). We used different ways to find the same answer:

First one to show with groupby(), aggr(count) function:

```
# Number of records nc.
nc_monthly_group1 = nc.groupby(['appointment_month'])['count_of_appointments'].agg('count')

# View result.
nc_monthly_group1
```

Second to use pivot.table() function:

```
# Number of records nc using pivot table function.
nc_monthly_pivot = pd.pivot_table(nc, values='count_of_appointments', \
                                       index=['appointment_month'] , aggfunc='count')
# View result.
nc_monthly_pivot
```

Third used categorical count:

```
# Number of records nc - using categorical count.
def cat_cnt(df1):
    print(df1.value_counts())

# Total number of records per month.
df_nc = nc['appointment_month'].astype('category')

# View the output.
cat_cnt(df_nc)
```

The pivot table function shows the nicest way the result.

3. Deep dive analysis for period Aug 2022 – June 2022

Since, the most recent data is for period Aug 2021 and June 2022, and it is the most available, the rest of the analysis in the file below will focus on this time-frame.

This part of the analysis was focusing on visualisation the data in order to see trends.

Before we can visualise in python, a few additional libraries needed to be imported, and set the style/size of the chart correct; and change data type for the x axis correct.

```
# Import the necessary libraries.
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

# Set figure size and style.
sns.set(rc={'figure.figsize':(15, 12)})
sns.set_style('white')

# Font size of the axes titles.
plt.rc('axes', titlesize=18)

# Font size of the x and y labels.
plt.rc('axes', labelsize=14)

# Font size of the Legend.
plt.rc('legend', fontsize=13)

# Default text size.
plt.rc('font', size=13)

# Create an empty plot.
fig, ax = plt.subplots()
```

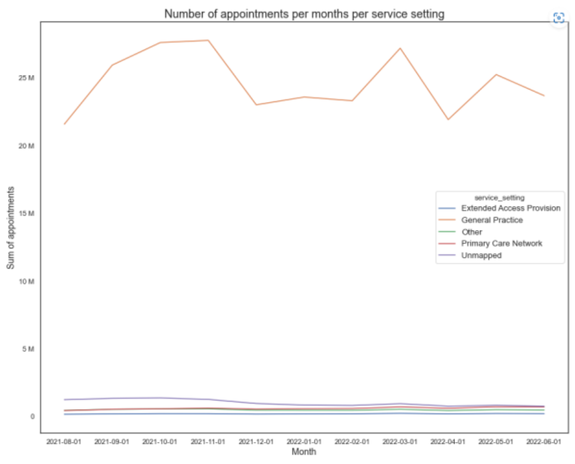
```
# Change the data type of appointment_month to string.
nc['appointment_month']= nc['appointment_month'].astype(str)

# Show the types and head of table.
print(nc.dtypes)
nc.head()
```

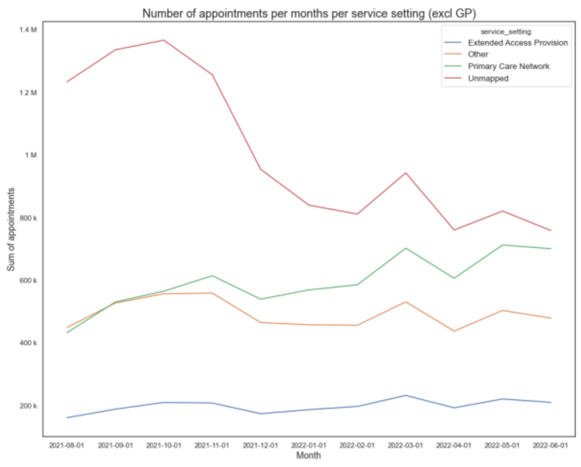
First, since we want to show the monthly trend, we used line charts to show the number of appointments per service setting; per context type; national category and per different seasons. We hoped to find interesting findings/insides.

Creating a line-chart process is the same, below will show one time how to perform it, rest will just show the result and conclusions for the other categories.

```
# Create a time-series.
nc_ss = nc.groupby(['appointment_month','service_setting'])[['count_of_appointments']].sum()\
 .reset_index().copy()

# View the output.
print(nc_ss.head())

# Create a lineplot without confidence level shade.
nc_ss_chart = sns.lineplot(x='appointment_month', y='count_of_appointments', \
                           hue= 'service_setting' , data=nc_ss, ci=None)

# Change formatter to display y values correct.
nc_ss_chart.yaxis.set_major_formatter(ticker.EngFormatter())

# Set labels and title.
nc_ss_chart.set_xlabel("Month")
nc_ss_chart.set_ylabel("Sum of appointments")
nc_ss_chart.set_title("Number of appointments per months per service setting")
```

Number of appointments per service setting





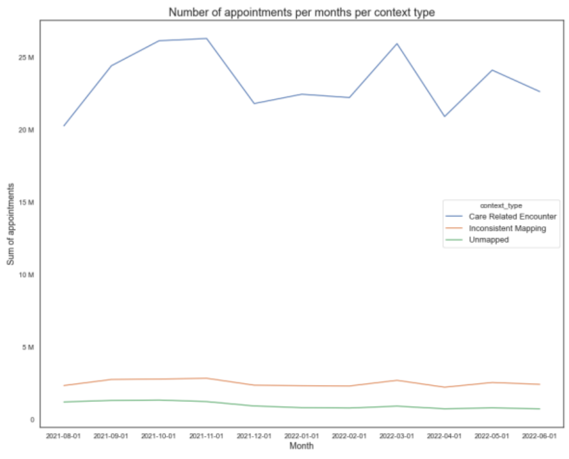
Since in the first chart above shows that the GP is very high vs the rest, the rest of the categories are flat. Therefore , we prepared the second chart to show the same without GP. This is done in python by filtering out GP from the subset:

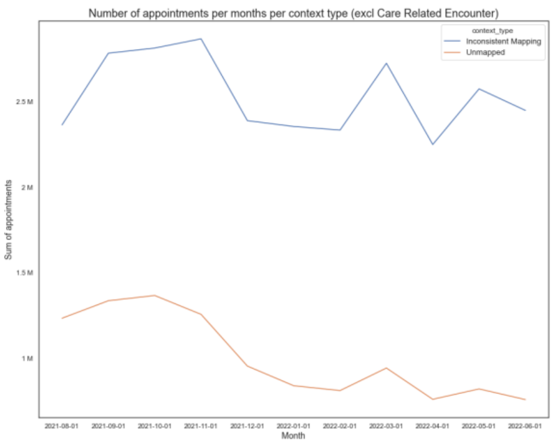
```
# Create a subset without GP.
nc_ss_no_GP = nc_ss[nc_ss['service_setting'] != 'General Practice']

# Create a lineplot without GP.
nc_ss_no_GP_chart = sns.lineplot(x='appointment_month', y='count_of_appointments', \
                            hue= 'service_setting' , data=nc_ss_no_GP, ci=None)

# Change formatter to display y values correct.
nc_ss_no_GP_chart.yaxis.set_major_formatter(ticker.EngFormatter())

# Set labels and title.
nc_ss_no_GP_chart.set_xlabel("Month")
nc_ss_no_GP_chart.set_ylabel("Sum of appointments")
nc_ss_no_GP_chart.set_title("Number of appointments per months per service setting (excl GP)")
```

Conclusion: Appointments related to General Practice are the most, it is more than 20M per month. Unfortunately there were a lot of unmapped categories in 2021 (between 800k and 1.3M), and it is still at 800k level in 2022 per month. The amount of appointment related to Primary Care Network is increasing: since August 2021 to June 2022 is almost doubled.

Number of appointments per context type

Python script is very similar as described above for service setting, only the conclusion is shown below:





Looking at the context type, the most appointments are booked regarding the Care Related Encounter (between 20 and 25M per month). There are monthly around 2.5M inconsistent mapping and about 1M unmapped context type, which is decreasing.

Number of appointments per national category

Line chart for the national category is very busy, since there are many categories. Therefore we decided to show the categories in a table view to see the top 3.

```
# View the appointments per month per national category
nc_nc_pivot = pd.pivot_table(nc, values='count_of_appointments', \
                                       index=['national_category'] , sort=False, aggfunc='sum')

# Sort the appointments.
nc_nc_pivot.sort_values(by=['count_of_appointments'], inplace=True, ascending=False)

# View result.
nc_nc_pivot
```

	count_of_appointments
national_category	
General Consultation Routine	97271522
General Consultation Acute	53691150
Clinical Triage	41546964
Planned Clinics	28019748
Inconsistent Mapping	27890802
Planned Clinical Procedure	25702694
Unmapped	11080810
Unplanned Clinical Activity	3055794
Home Visit	2144452
Structured Medication Review	1858379
Service provided by organisation external to the practice	852133
Patient contact during Care Home Round	810330
Care Home Visit	628279
Social Prescribing Service	475828
Walk-in	412438
Care Home Needs Assessment & Personalised Care and Support Planning	405904
Non-contractual chargeable work	138911
Group Consultation and Group Education	60632

Conclusions: the most appointments are booked regarding the General Consultation Routine (between 8 and 10M per month), following General Consultation Acute (between 4M and 6M pe month), and Clinical Triage (about 4M per month).

Number of appointments per season

Next we wanted to investigate whether the appointments has different pattern in different season. Below the script for the summer season:

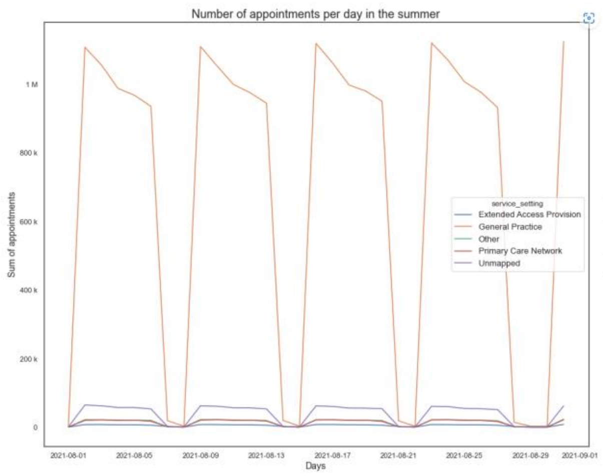
Script starts first to create a new subset, then filter for the summer month. Since the GP service setting is again has a different range, we created a second chart with excluding GP.

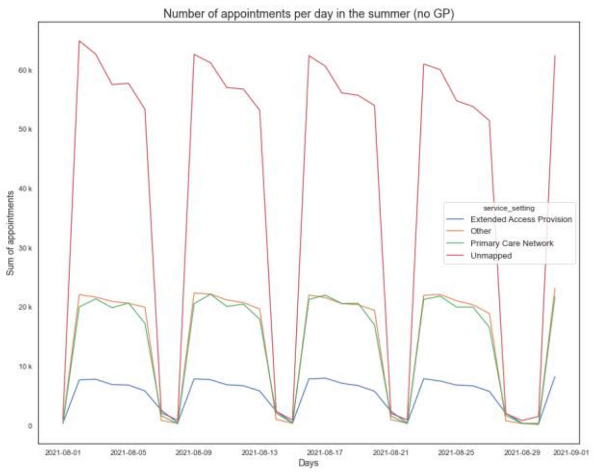
```
# Create a new set.
nc_ss_day = nc.groupby(['appointment_date' , 'appointment_month','service_setting'])\
[['count_of_appointments']].sum().reset_index().copy()

# Make sure that month column is string.
nc_ss_day['appointment_month']= nc_ss_day['appointment_month'].astype(str)

# View data types.
print(nc_ss_day.dtypes)

# View result.
nc_ss_day
```

```
# Filter for summer.
summer = nc_ss_day[nc_ss_day['appointment_month'] == '2021-08-01']

# Create a lineplot without confidence level shade.
summer_chart = sns.lineplot(x='appointment_date', y='count_of_appointments', \
                            hue= 'service_setting' , data=summer , ci=None)

# Change formatter to display y values correct.
summer_chart.yaxis.set_major_formatter(ticker.EngFormatter())

# Set labels and title.
summer_chart.set_xlabel("Days")
summer_chart.set_ylabel("Sum of appointments")
summer_chart.set_title("Number of appointments per day in the summer")
```

```
# Create a subset for summer and without GP.
summer_noGP = summer[summer['service_setting'] != 'General Practice']

# Create a lineplot without confidence level shade.
summer_noGP_chart = sns.lineplot(x='appointment_date', y='count_of_appointments', \
                            hue= 'service_setting' , data=summer_noGP , ci=None)

# Change formatter to display y values correct.
summer_noGP_chart.yaxis.set_major_formatter(ticker.EngFormatter())

# Set labels and title.
summer_noGP_chart.set_xlabel("Days")
summer_noGP_chart.set_ylabel("Sum of appointments")
summer_noGP_chart.set_title("Number of appointments per day in the summer (no GP)")
```

Below the graphs for the summer month as an example:





Conclusions: In all seasons is the pattern is very similar. Low at the weekends, and Monday is the busiest day of the week in all categories.

4. Analysis of Twitter data

The team analyzed the available twitter data. Although, the result of the analysis does not contribute directly to the problem, it does show its potential for NHS. NHS could utilise tweets because:

- It is very useful to look at the re-tweeted and favorite tweet messages in more detail because these are the most popular ones.
- Great way to increase the visibility of customers' tweets / trending topics.
- It is a great way to find out the reactions of people to an event as it happens, whether it's entertainment or breaking news, or medical like COVID.
- maybe even paying for advertising to increase the reach of NHS.

The analysis started with setting up the workstation: import, and analyze the dataset, set figure size and display texts of the twitter data:

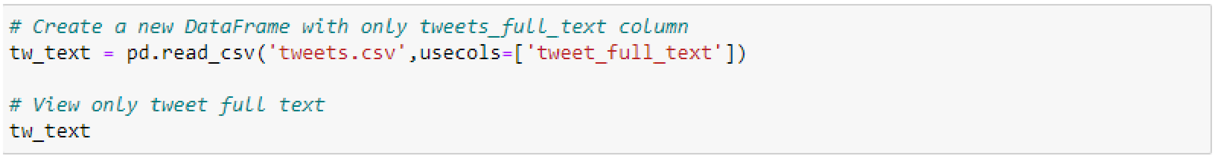
```
# Set figure size.
sns.set(rc={'figure.figsize':(15, 12)})

# Set the plot style as white.
sns.set_style('white')

# Display maximum column width.
pd.options.display.max_colwidth = 200
```

```
# Import the necessary data set.
tw = pd.read_csv('tweets.csv')

# View the DataFrame first five rows.
tw.head(5)
```

```
# View data set.
print(tw.shape)
print(tw.dtypes)
print(tw.columns)
```

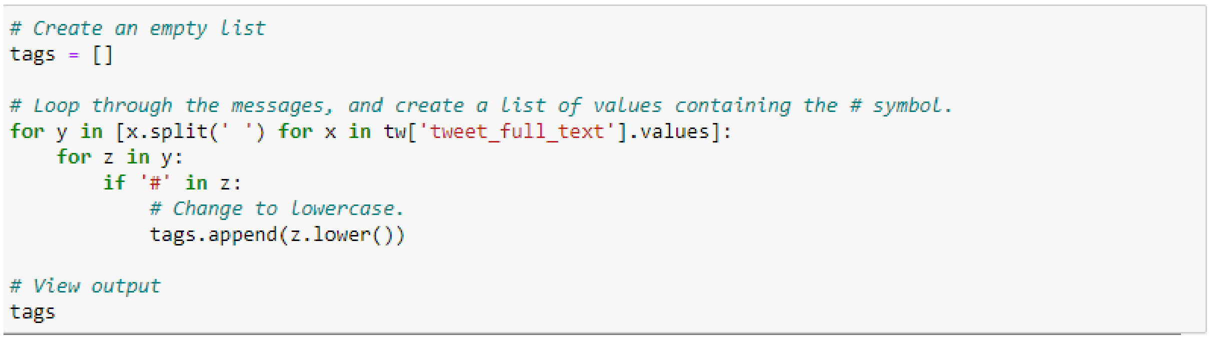
```
# Descriptive statistics.
tw.describe()
```

```
# Explore the data set.
tw['tweet_retweet_count'].value_counts()
```

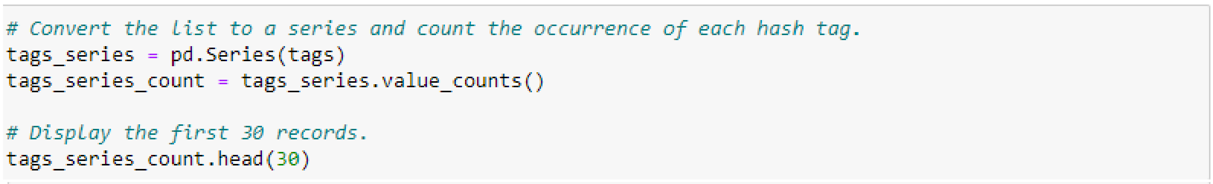


```
# Create a new DataFrame with only tweets_full_text column
tw_text = pd.read_csv('tweets.csv',usecols=['tweet_full_text'])

# View only tweet full text
tw_text
```

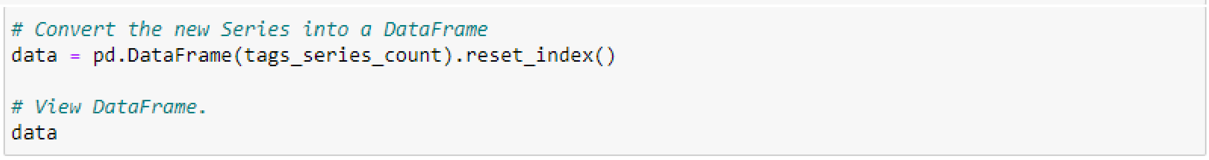
Next we created a list for the hastags:



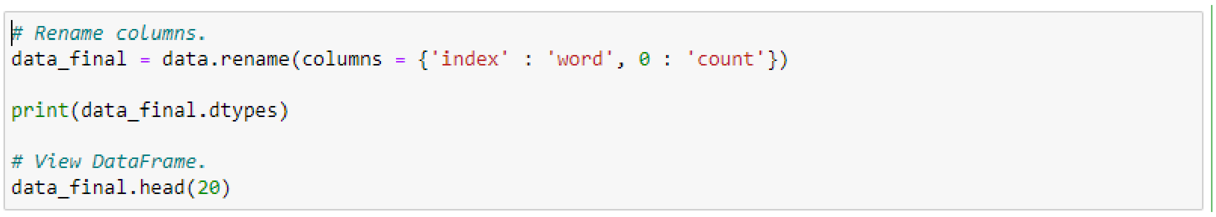
```
# Create an empty list
tags = []

# Loop through the messages, and create a list of values containing the # symbol.
for y in [x.split(' ') for x in tw['tweet_full_text'].values]:
    for z in y:
        if '#' in z:
            # Change to lowercase.
            tags.append(z.lower())

# View output
tags
```



```
# Convert the list to a series and count the occurrence of each hash tag.
tags_series = pd.Series(tags)
tags_series_count = tags_series.value_counts()

# Display the first 30 records.
tags_series_count.head(30)
```

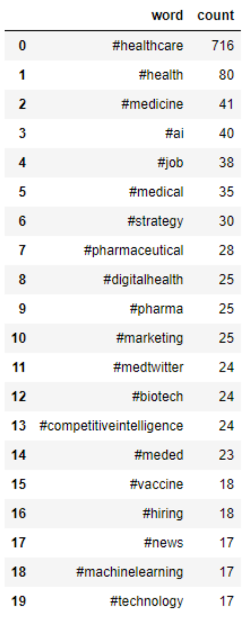


```
# Convert the new Series into a DataFrame
data = pd.DataFrame(tags_series_count).reset_index()

# View DataFrame.
data
```



```
# Rename columns.
data_final = data.rename(columns = {'index' : 'word', 0 : 'count'})

print(data_final.dtypes)

# View DataFrame.
data_final.head(20)
```

As a result , the top 20 trending hashtags are:



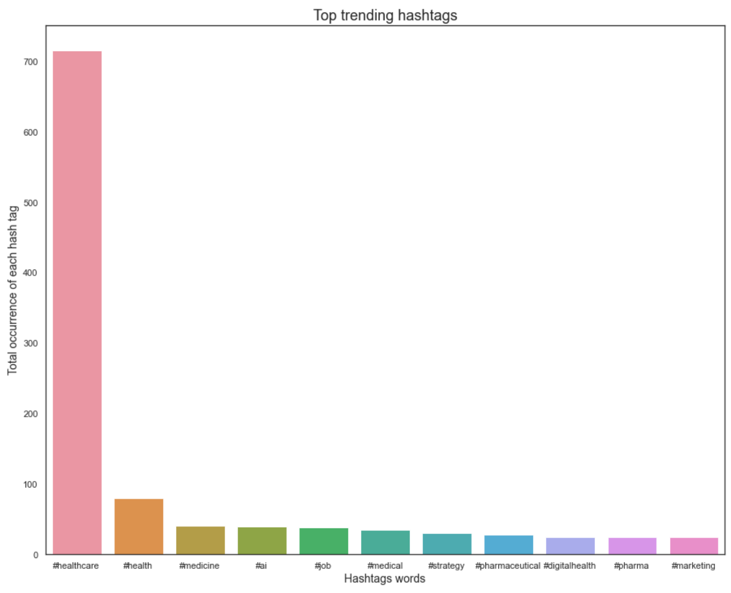
	word	count
0	#healthcare	716
1	#health	80
2	#medicine	41
3	#ai	40
4	#job	38
5	#medical	35
6	#strategy	30
7	#pharmaceutical	28
8	#digitalhealth	25
9	#pharma	25
10	#marketing	25
11	#medtwitter	24
12	#biotech	24
13	#competitiveintelligence	24
14	#meded	23
15	#vaccine	18
16	#hiring	18
17	#news	17
18	#machinelearning	17
19	#technology	17

Next created a bar chart for the visualisation:

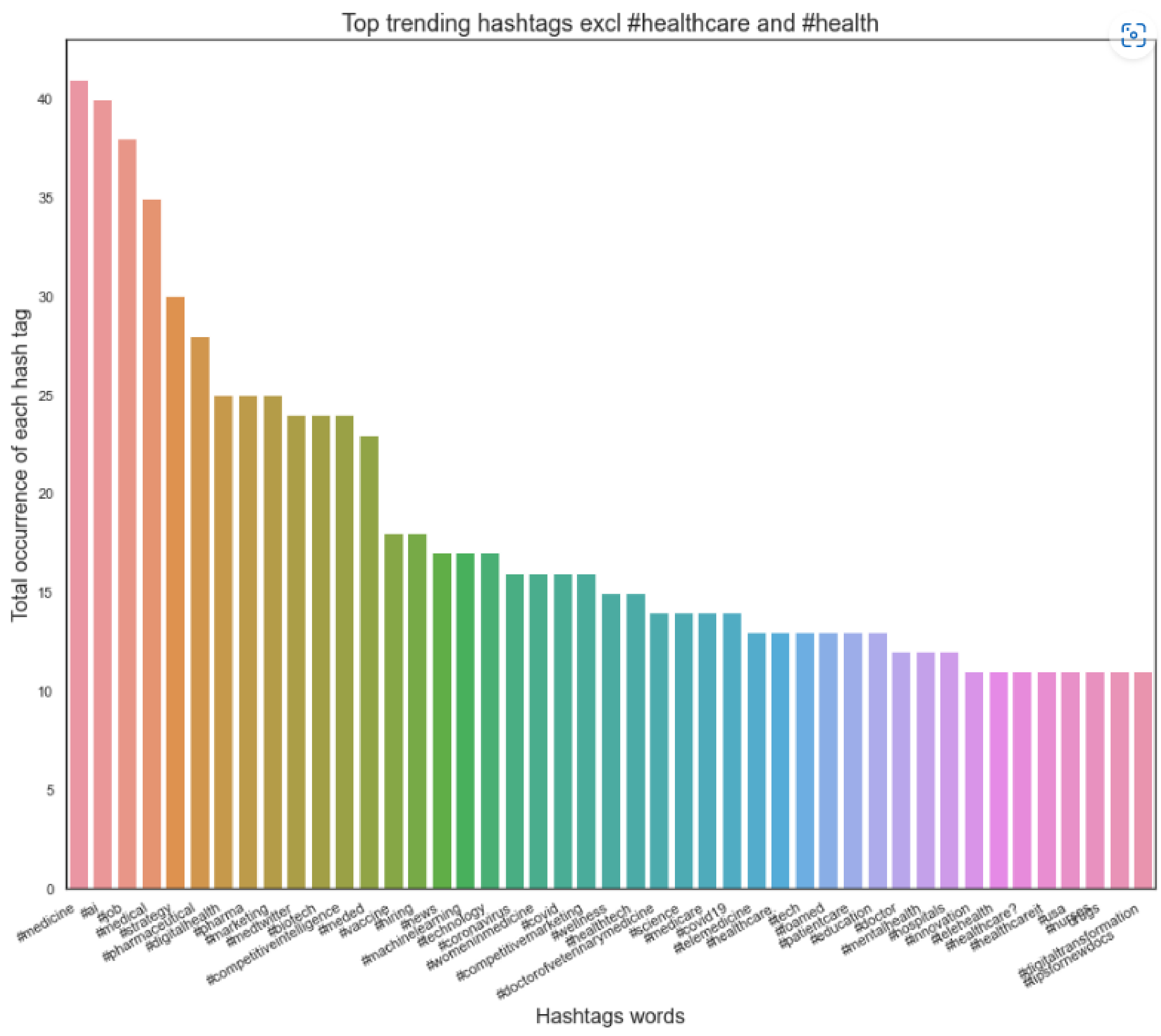
```
# Create subset for count higher than 10.
data_final_final = data_final[data_final['count'] > 10]

# Create a barplot.
data_final_chart = sns.barplot(x='word', y='count', data =data_final_final)

# Font size of the axes titles.
plt.rc('axes', titlesize=18)

# Font size of the x and y labels.
plt.rc('axes', labelsize=16)

# Change formatter to display y values correct.
data_final_chart.yaxis.set_major_formatter(ticker.EngFormatter())

# Set labels and title.
data_final_chart.set_xlabel("Hashtags words")
data_final_chart.set_ylabel("Total occurrence of each hash tag")
data_final_chart.set_title("Top trending hashtags")
```

Chart was not readable, therefore only selected top 10 (count more than 24):

```
# Create subset for count higher than 24.
data_final_final_top10 = data_final[data_final['count'] > 24]

# Create a barplot.
data_final_chart_top10= sns.barplot(x='word', y='count', data =data_final_final_top10)

# Font size of the axes titles.
plt.rc('axes', titlesize=18)

# Font size of the x and y labels.
plt.rc('axes', labelsize=16)

# Change formatter to display y values correct.
data_final_chart_top10.yaxis.set_major_formatter(ticker.EngFormatter())

# Set labels and title.
data_final_chart_top10.set_xlabel("Hashtags words")
data_final_chart_top10.set_ylabel("Total occurrence of each hash tag")
data_final_chart_top10.set_title("Top 10 trending hashtags ")
```



As you can see in the chart above, the most tweets are in the Healthcare and health category, but if we exclude these two “general” categories, the chart below shows all the other most popular hashtags.



5. Investigate the main concerns posed by the NHS

The last part of the analysis investigated the main concerns of NHS: utilization and not attendance.

5.1 Utilisation

First we investigated whether NHS should start looking at increasing staff levels:

1) First created a subset and showed the total visits per month for the above mentioned time period.
2) Next we calculated the monthly utilisation. Since we know that NHS can accommodate a maximum of 1,200,000 appointments per day, we could calculate the monthly actual utilisation as you see in the script below.

```
# Create an aggregated data set to review the different features.
ar_subset_aggr = ar_subset.groupby(['appointment_month', 'appointment_status', 'hcp_type', \
                                     'appointment_mode', 'time_between_book_and_appointment'] \
                                   ) ['count_of_appointments'].agg('sum').reset_index()

# View the DataFrame.
ar_subset_aggr
```

```
# Determine the total number of appointments per month.
ar_app_monhtly= ar_subset.groupby(['appointment_month'])['count_of_appointments'].\
agg('sum').reset_index()

# View the DataFrame.
ar_app_monhtly

# Add a new column to indicate the average utilisation of services.
ar_app_monhtly['utilisation'] = ar_app_monhtly['count_of_appointments'] // 30

# Add a new column to show the maximum appointment.
ar_app_monhtly['max_utilisation'] = 1200000

# Add a new column to show the utilisation %.
ar_app_monhtly['actual_utilisation'] = (ar_app_monhtly['utilisation']/ar_app_monhtly['max_utilisation']
*100).round(2).astype(int)

# View the DataFrame.
ar_app_monhtly
```

Output:

	appointment_month	count_of_appointments	utilisation	max_utilisation	actual_utilisation
0	2021-08-01	23852171	795072	1200000	66
1	2021-09-01	28522501	950750	1200000	79
2	2021-10-01	30303834	1010127	1200000	84
3	2021-11-01	30405070	1013502	1200000	84
4	2021-12-01	25140776	838025	1200000	69
5	2022-01-01	25635474	854515	1200000	71
6	2022-02-01	25355260	845175	1200000	70
7	2022-03-01	29595038	986501	1200000	82
8	2022-04-01	23913060	797102	1200000	66
9	2022-05-01	27495508	916516	1200000	76
10	2022-06-01	25828078	860935	1200000	71

```
# Show lowest actaul utilisation.
actual_utilisation_min = ar_app_monhtly['actual_utilisation'].min()
actual_utilisation_min
```

```
66
```

```
# Show highest actaul utilisation.
actual_utilisation_max = ar_app_monhtly['actual_utilisation'].max()
actual_utilisation_max
```

```
84
```

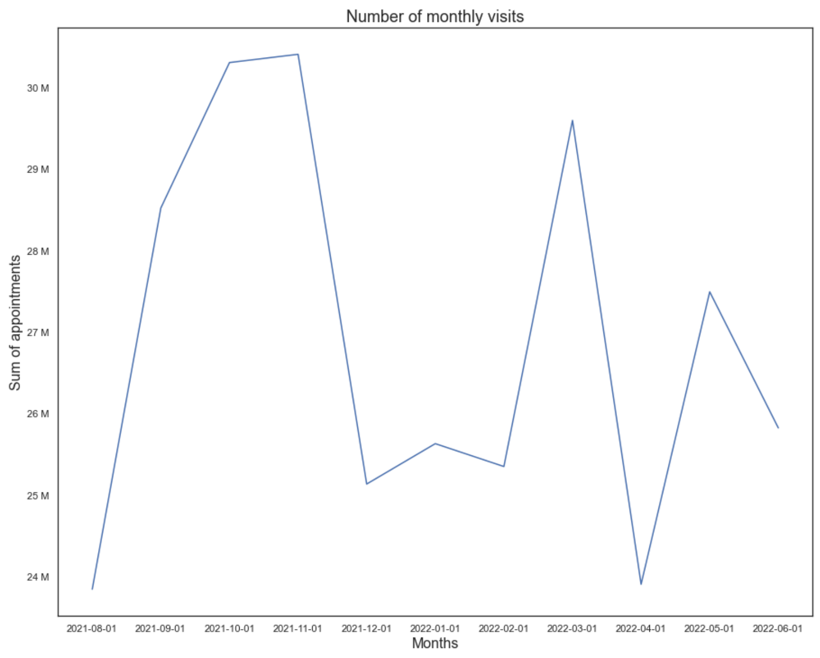
Conclusion:
Actual utilisation is between 66% and 84%.

Creating the visualisations:

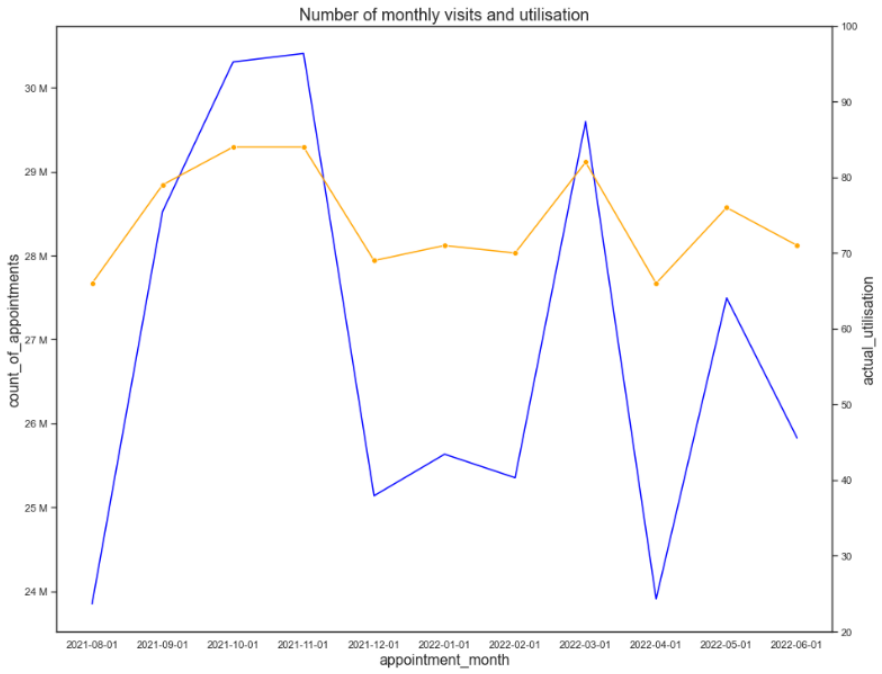
```
# Convert the appointment_month to string data type for ease of visualisation.
ar_app_monhtly['appointment_month']= ar_app_monhtly['appointment_month'].astype(str)

# Set figure size.
sns.set(rc={'figure.figsize':(15, 12)})
sns.set_style('white')

# Font size of the axes titles.
plt.rc('axes', titlesize=18)

# Font size of the x and y labels.
plt.rc('axes', labelsize=16)

# Create a lineplot indicating the number of monthly visits.
ar_app_chart = sns.lineplot(x='appointment_month', y='count_of_appointments', \
                            data=ar_app_monhtly , ci=None)

# Change formatter to display y values correct.
ar_app_chart.yaxis.set_major_formatter(ticker.EngFormatter())

# Set labels and title.
ar_app_chart.set_xlabel("Months")
ar_app_chart.set_ylabel("Sum of appointments")
ar_app_chart.set_title("Number of monthly visits")
```

```
# Create chart to show the utilisation and total visits in one chart.
# Convert the appointment_month to string data type for ease of visualisation.
ar_app_monhtly['appointment_month']= ar_app_monhtly['appointment_month'].astype(str)

# Set figure size.
sns.set(rc={'figure.figsize':(15, 12)})
sns.set_style('white')

# Font size of the axes titles.
plt.rc('axes', titlesize=18)

# Font size of the x and y labels.
plt.rc('axes', labelsize=16)

# Create a lineplot indicating the number of monthly visits and utilisation in one chart.
fig, ax1= plt.subplots()
ax2 = ax1.twinx()
chart1 = sns.lineplot(x='appointment_month', y='count_of_appointments', ax=ax1, color = 'blue', \
                            data=ar_app_monhtly , ci=None)
chart2 = sns.lineplot(x='appointment_month', y='actual_utilisation', ax=ax2, color = 'orange' , \
                            data=ar_app_monhtly , marker = 'o', ci=None)

# Set y axis limit.
chart2.set_ylim(20, 100)

# Change formatter to display y values correct.
chart1.yaxis.set_major_formatter(ticker.EngFormatter())

# Set title.
chart1.set_title("Number of monthly visits and utilisation")

# Show chart.
plt.show()
```

Looking at the total number of visits per month (see below), we can see that the busiest month are October, November and March. The lowest visits are in summer, winter months and in April.



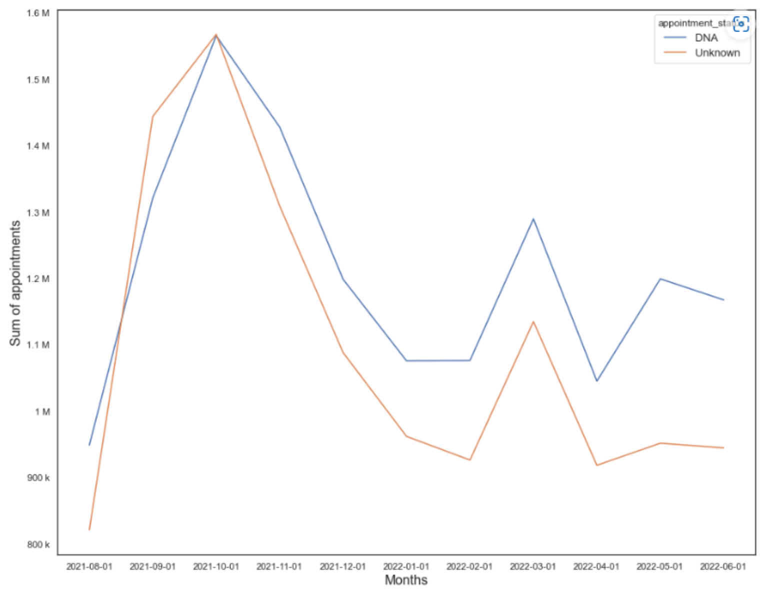
The actual utilisation is between 66% and 84%, they do not have to look at increasing staff levels. Below the monthly visits and the utilization per month:



Analysis also investigated the appointments that did not have attendance or unknown, and to see what the different locations are:

```
# Filter for DNA and unknown
DNA_unknown_f = DNA_unknown[DNA_unknown['appointment_status'] != 'Attended']

# Create a line plot indicating if visits are attended.
DNA_unknown_f_chart = sns.lineplot(x='appointment_month', y='count_of_appointments', \
                           data=DNA_unknown_f, hue='appointment_status', ci=None)

# Change formatter to display y values correct.
DNA_unknown_f_chart.yaxis.set_major_formatter(ticker.EngFormatter())

# Set labels and title.
DNA_unknown_f_chart.set_xlabel("Months")
DNA_unknown_f_chart.set_ylabel("Sum of appointments")
ar_app_chart_as.set_title("Number of monthly visits for DNA and unknown")
```

```
# View top five locations with the highest number of not attended or unknown
# Create a dataframe for the locations
ar_subset_DNA_loc = ar.groupby(["icb_ons_code",'appointment_status'])["count_of_appointments"].\
agg('sum').reset_index().sort_values('count_of_appointments', ascending = False)

# Filter for DNA and unknown
ar_subset_DNA_loc_final = ar_subset_DNA_loc[ar_subset_DNA_loc['appointment_status'] != 'Attended']

# View the DataFrame.
ar_subset_DNA_loc_final.head(10)
```



Top 5 locations with DNA/unknown:
NHS Greater Manchester ICB
NHS North East and North Cumbria ICB
NHS Cheshire and Merseyside ICB
NHS West Yorkshire ICB
NHS North West London ICB

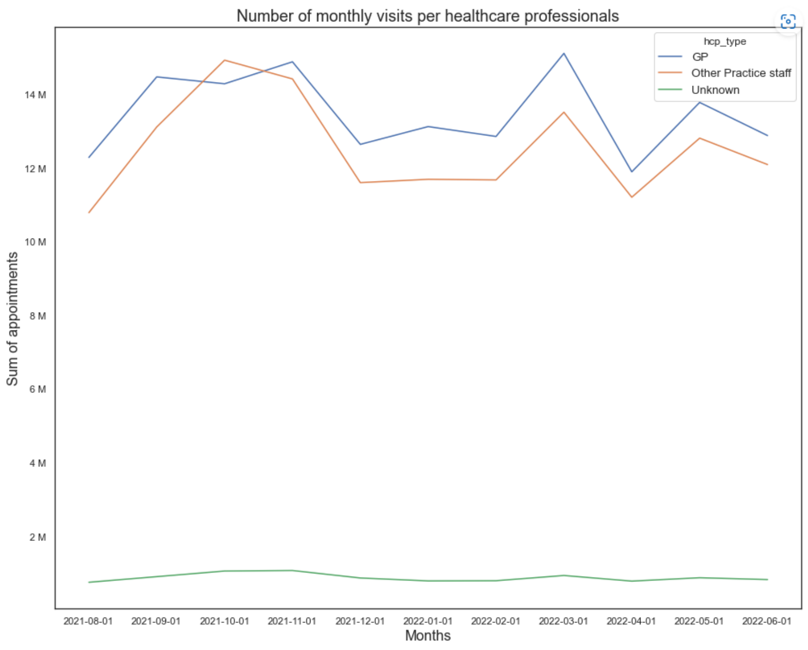
Conclusions: As previously already mentioned: Currently about 91% of the appointments are attended; rest are not attended (4%) or unknown (5%) - between 800k and 1.6M per months.
Most unknown and not attended visits are in October and March. Suggestion is to deep dive why the not attended visits and unknown category is reported, and start with the top 5 locations as shown above.

5.2 Healthcare professional types over time

Since the total number of visits very high level, we started to dive deeper, fist we investigated if healthcare professional types differ over time. Again line chart is used, and starts with creating a subset, and script for the line chart:

```
# Create a subset to see hcp types.
ar_app_monhtly_hcp= ar_subset.groupby(['appointment_month','hcp_type'])['count_of_appointments'].\
agg('sum').reset_index()

# View the DataFrame.
ar_app_monhtly_hcp.head()
```

```
# Convert the appointment_month to string data type for ease of visualisation.
ar_app_monhtly_hcp['appointment_month']= ar_app_monhtly_hcp['appointment_month'].astype(str)

# Create a line plot indicating the healthcare professionals over time.
ar_app_chart_hcp = sns.lineplot(x='appointment_month', y='count_of_appointments', \
                              data=ar_app_monhtly_hcp , hue='hcp_type', ci=None)

# Change formatter to display y values correct.
ar_app_chart_hcp.yaxis.set_major_formatter(ticker.EngFormatter())

# Set labels and title.
ar_app_chart_hcp.set_xlabel("Months")
ar_app_chart_hcp.set_ylabel("Sum of appointments")
ar_app_chart_hcp.set_title("Number of monthly visits per healthcare professionals")
```

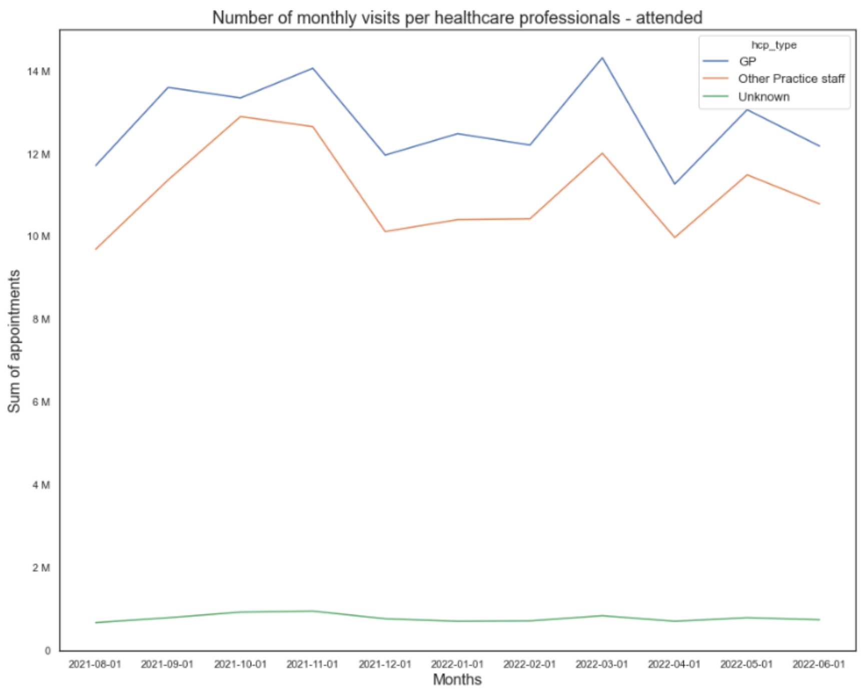


Conclusion:
The trend per professional type is similar: Summer and Winter months are low for both; November is the highest for Other Practice stuff; March is the highest months for GP. But the trend for both is the same: Sept-Oct-Nov are high following low winter months; March is high following lower April and summer months.

But since we know that there is a high amount of not attend and unknow, we checked the trend only for the attended ones:

```
# Create a filter for attended.
ar_app_monhtly_hcp_att = ar_subset.groupby(['appointment_month', 'appointment_status','hcp_type'])\
['count_of_appointments'].agg('sum').reset_index()

ar_app_monhtly_hcp_att = ar_app_monhtly_hcp_att[ar_app_monhtly_hcp_att['appointment_status'] \
                                                 == 'Attended']

ar_app_monhtly_hcp_att.head()
```

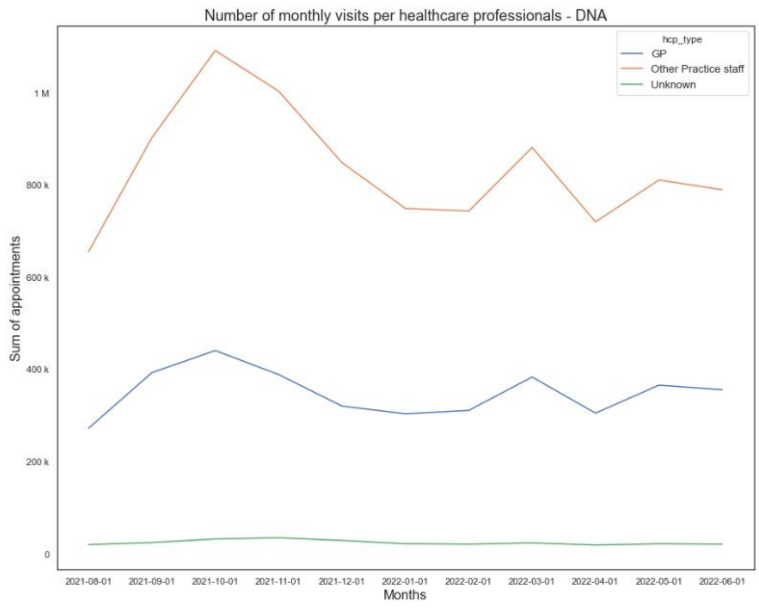
```
# Convert the appointment_month to string data type for ease of visualisation.
ar_app_monhtly_hcp_att['appointment_month']= ar_app_monhtly_hcp_att['appointment_month'].astype(str)

# Create a line plot indicating the healthcare professionals over time for attended group.
ar_app_monhtly_hcp_att_chart = sns.lineplot(x='appointment_month', y='count_of_appointments', \
                             data=ar_app_monhtly_hcp_att , hue='hcp_type', ci=None)

# Change formatter to display y values correct.
ar_app_monhtly_hcp_att_chart .yaxis.set_major_formatter(ticker.EngFormatter())

# Set labels and title.
ar_app_monhtly_hcp_att_chart.set_xlabel("Months")
ar_app_monhtly_hcp_att_chart.set_ylabel("Sum of appointments")
ar_app_monhtly_hcp_att_chart.set_title("Number of monthly visits per healthcare professionals - attende
```



Create same chart with filter for not attended appointments:

```
# Create a filter for not attended.
ar_app_monhtly_hcp_DNA = ar_subset.groupby(['appointment_month', 'appointment_status','hcp_type'])\
['count_of_appointments'].agg('sum').reset_index()

ar_app_monhtly_hcp_DNA = ar_app_monhtly_hcp_DNA[ar_app_monhtly_hcp_DNA['appointment_status'] \
                                                == 'DNA']

ar_app_monhtly_hcp_DNA.head()
```

```
# Convert the appointment_month to string data type for ease of visualisation.
ar_app_monhtly_hcp_DNA['appointment_month']= ar_app_monhtly_hcp_DNA['appointment_month'].astype(str)

# Create a line plot indicating the healthcare professionals over time for attended group.
ar_app_monhtly_hcp_DNA_chart = sns.lineplot(x='appointment_month', y='count_of_appointments', \
                             data=ar_app_monhtly_hcp_DNA , hue='hcp_type', ci=None)

# Change formatter to display y values correct.
ar_app_monhtly_hcp_DNA_chart .yaxis.set_major_formatter(ticker.EngFormatter())

# Set labels and title.
ar_app_monhtly_hcp_DNA_chart.set_xlabel("Months")
ar_app_monhtly_hcp_DNA_chart.set_ylabel("Sum of appointments")
ar_app_monhtly_hcp_DNA_chart.set_title("Number of monthly visits per healthcare professionals - DNA")
```



Conclusion:
After filtering for attended/not attended: The pattern for the 2 different healthcare professional stays the same, but if we zoom into the not attended chart: There are way higher amount of visits that are not attended in the Other Practice staff category. And the most not attended visits are in November. And this drives the high peak in October.

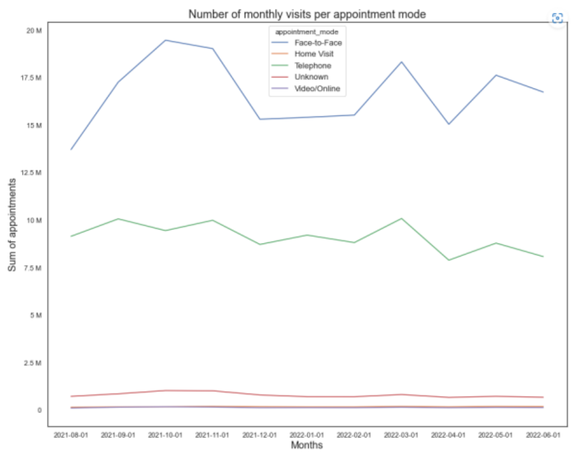
5.3 Trend per Appointment type and the busiest months

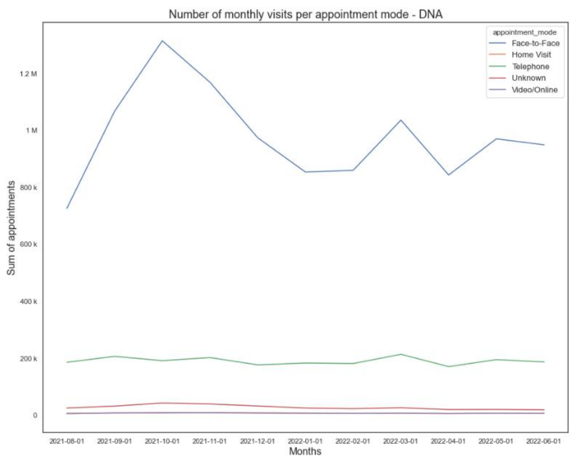
Next we investigated the appointment types. Started to create a subset, plot the result.

```
# Create a subset
ar_app_monhtly_am= ar_subset.groupby(['appointment_month','appointment_mode'])['count_of_appointments']
agg('sum').reset_index()

# View the DataFrame.
ar_app_monhtly_am.head()
```

```
# Convert the appointment_month to string data type for ease of visualisation.
ar_app_monhtly_am['appointment_month']= ar_app_monhtly_am['appointment_month'].astype(str)

# Create a line plot indicating the terms of appointment types and the busiest months.
ar_app_chart_am = sns.lineplot(x='appointment_month', y='count_of_appointments', \
                               data=ar_app_monhtly_am , hue='appointment_mode', ci=None)

# Change formatter to display y values correct.
ar_app_chart_am.yaxis.set_major_formatter(ticker.EngFormatter())

# Set labels and title.
ar_app_chart_am.set_xlabel("Months")
ar_app_chart_am.set_ylabel("Sum of appointments")
ar_app_chart_am.set_title("Number of monthly visits per appointment mode")
```

Since, we know that the not attended is rather high, we wanted to see the trend for this category as well:





As you see in the first chart above the face-to-face visits are the most, and the busiest months for that appointment mode is October and November. The video and home visit are flat month to month. Looking at the telephone visits, the highest is march and very low in April.

Interesting finding is that the face-to-face visit has a high amount of not attend visits in October (more than 1.2M in one month) . Also, the telephone visit has a constant 200k per month non attended visit.

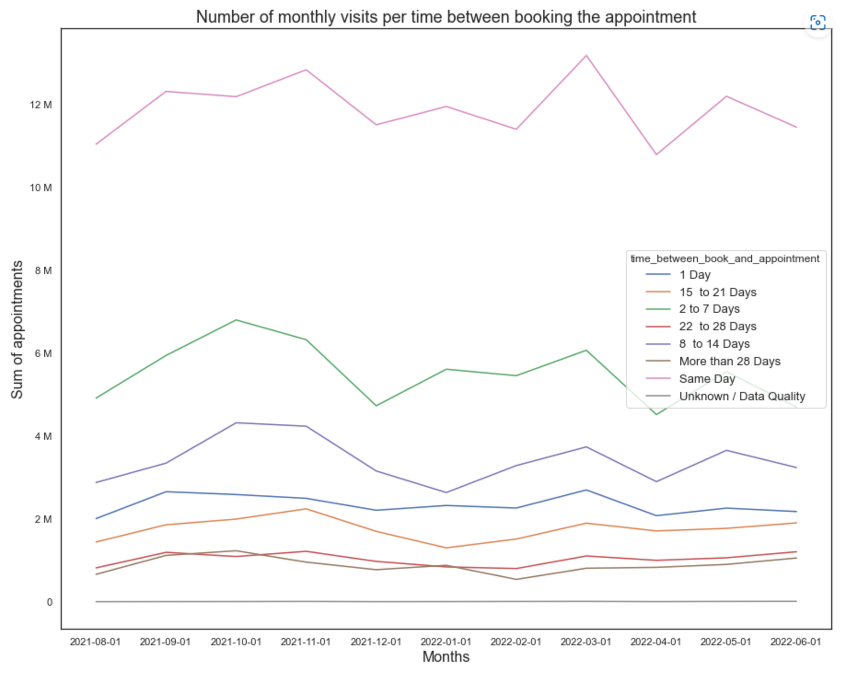
5.4. Trends in time between booking and appointment

Next we investigated the time between booking and appointment. Started to create a subset, plot the result.

```
# Create a subset
ar_app_monhtly_ta= ar_subset.groupby(['appointment_month','time_between_book_and_appointment'])\
['count_of_appointments'].agg('sum').reset_index()

# View the DataFrame.
ar_app_monhtly_ta.head()
```

```
# Convert the appointment_month to string data type for ease of visualisation.
ar_app_monhtly_ta['appointment_month']= ar_app_monhtly_ta['appointment_month'].astype(str)

# Create a line plot indicating the time between booking an appointment.
ar_app_chart_ta = sns.lineplot(x='appointment_month', y='count_of_appointments', \
                                data=ar_app_monhtly_ta , hue='time_between_book_and_appointment', ci=None)

# Change formatter to display y values correct.
ar_app_chart_ta.yaxis.set_major_formatter(ticker.EngFormatter())

# Set labels and title.
ar_app_chart_ta.set_xlabel("Months")
ar_app_chart_ta.set_ylabel("Sum of appointments")
ar_app_chart_ta.set_title("Number of monthly visits per time between booking the appointment")
```

<u>Output:</u>

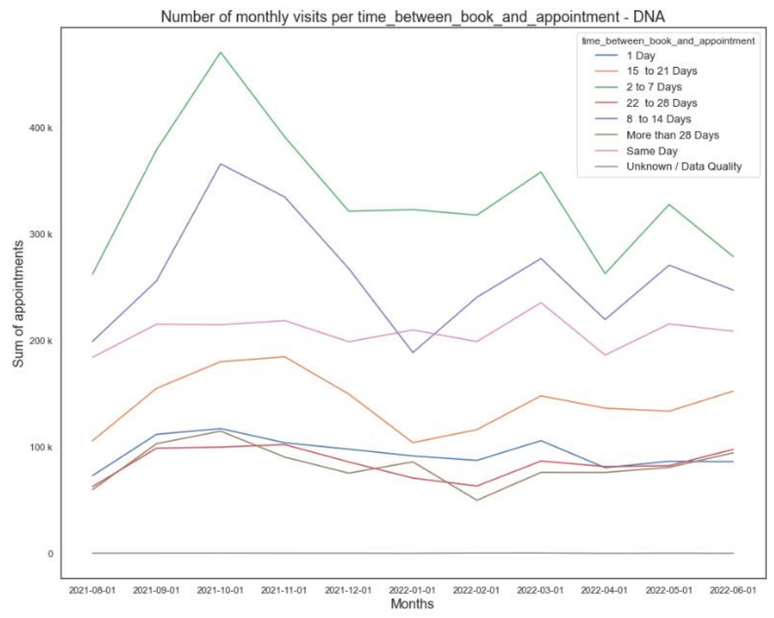


The majority of the appointments are in the same day category, followed by '2 to 7 days' and '8 to 14 days'. The busiest month for the same day is March, while for the 2 to 7 days' and '8 to 14 days' is the October month.

We zoomed into the did not attend ones here as well:

```
# Create a filter for DNA.
ar_app_monhtly_ta_DNA = ar_subset_aggr.groupby(['appointment_month','appointment_status',\
                                              'time_between_book_and_appointment'])\
['count_of_appointments'].agg('sum').reset_index()

ar_app_monhtly_ta_DNA = ar_app_monhtly_ta_DNA[ar_app_monhtly_ta_DNA['appointment_status'] \
                                                     == 'DNA']

ar_app_monhtly_ta_DNA.head()
```

```
# Convert the appointment_month to string data type for ease of visualisation.
ar_app_monhtly_ta_DNA['appointment_month']= ar_app_monhtly_ta_DNA['appointment_month'].astype(str)

# Create a line plot indicating the healthcare professionals over time for attended group.
ar_app_monhtly_ta_DNA_chart = sns.lineplot(x='appointment_month', y='count_of_appointments', \
                             data=ar_app_monhtly_ta_DNA , hue='time_between_book_and_appointment', ci=No

# Change formatter to display y values correct.
ar_app_monhtly_ta_DNA_chart.yaxis.set_major_formatter(ticker.EngFormatter())

# Set labels and title.
ar_app_monhtly_ta_DNA_chart.set_xlabel("Months")
ar_app_monhtly_ta_DNA_chart.set_ylabel("Sum of appointments")
ar_app_monhtly_ta_DNA_chart.set_title("Number of monthly visits per time_between_book_and_appointment -
```



The trend is very different for the not attend ones: Majority is in the 2 to 7 days' and '8 to 14 days', and very high in November.

5.5 Comparing the various service settings

Last we compared the service setting using box plot.

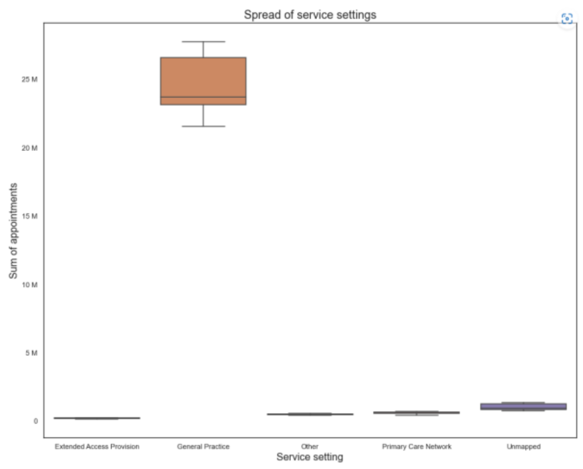
```
# Create a new DataFrame consisting of the month of appointment and the number of appointments.
nc_spread = nc.groupby(['appointment_month','service_setting'])['count_of_appointments'].sum().reset_index()

# View the DataFrame.
nc_spread.head()
```

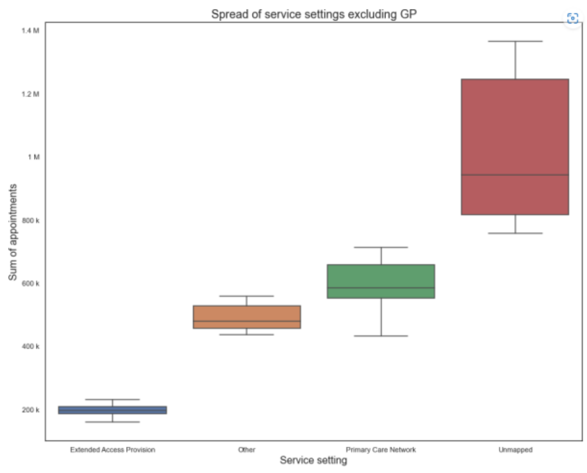
```
# Create a boxplot to investigate spread of service settings.
nc_spread_boxplot = sns.boxplot(data=nc_spread, x='service_setting', y='count_of_appointments')

# Change formatter to display y values correct.
nc_spread_boxplot.yaxis.set_major_formatter(ticker.EngFormatter())

# Set labels and title.
nc_spread_boxplot.set_xlabel("Service setting")
nc_spread_boxplot.set_ylabel("Sum of appointments")
nc_spread_boxplot.set_title("Spread of service settings")
```

Since, the GP has way higher spread, the other categories are not shown good on the chart, therefore we created a separate chart without GP:

```
# Create a subset without GP.
nc_spread_no_GP = nc_spread[nc_spread['service_setting'] != 'General Practice']

# View the DataFrame.
nc_spread_no_GP.head()
```

```
# Create a boxplot to investigate the service settings without GP.
nc_spread_boxplot_no_GP = sns.boxplot(data=nc_spread_no_GP, x='service_setting', \
                                      y='count_of_appointments')

# Change formatter to display y values correct.
nc_spread_boxplot_no_GP.yaxis.set_major_formatter(ticker.EngFormatter())


# Set labels and title.
nc_spread_boxplot_no_GP.set_xlabel("Service setting")
nc_spread_boxplot_no_GP.set_ylabel("Sum of appointments")
nc_spread_boxplot_no_GP.set_title("Spread of service settings excluding GP")
```





The biggest monthly spread, with other words the biggest variation in the amount of visits from month to month is at General Practice. The difference is 3.4m. There is also a lot of unmapped appointments: between 800k and 1.3m. The other categories has relatively stable amount of appointment from month to months and the spread is not as huge.

The calculation of the 3.4m spread for GP was done with the following script:

```
# Calculate upper and lower spread:
# Indicate the column.
cols = ['count_of_appointments']

# Calculate quantiles and IQR.
Q1 = nc_spread_GP[cols].quantile(0.25)
Q3 = nc_spread_GP[cols].quantile(0.75)

print(Q1)
print(Q3)

IQR = Q3 - Q1

print(IQR)
```

```
count_of_appointments    23157376.0
Name: 0.25, dtype: float64
count_of_appointments    26564094.5
Name: 0.75, dtype: float64
count_of_appointments    3406718.5
dtype: float64
```

6. Conclusions and recommendations

- Actual utilisation is between 66% and 84%. There is adequate staff and capacity in the networks. Do not have to look at increasing staff levels currently.
- The highest amount visit is in the General Practice category, but that is also the one that has the biggest variation in the amount of visits from month to month (3.4m).
- Currently about 4% of the visits are not attended. About 5% of the visits are unknown if it is attended or not. More data required to further analyze this. What we know:
    - Most unknown and not attended visits are in October and March.
    - Very high amount are in Other Practice staff; Face-to-face visit and in the '2 to 7 days' time between booking and appointment categories
- Recommended to utilize Twitter in order to better understand customer's behavior.