Technical Report

Diagnostic Analysis using Python - NHS

Kuok Tou Leong

1) Background of the Business

The National Health Service (NHS), England's publicly funded healthcare system, confronts resource allocation challenges due to population growth. This report seeks to evaluate the sufficiency of staffing and capacity within NHS networks, as well as the actual utilisation of resources. Through an analysis of internal data encompassing service utilisation and appointment details, alongside external data from sources like Twitter (X), insights will be gathered to inform strategic planning.

2) Analytical approach

The team was provided with four datasets: actual_duration, national_categories, appointments_regional, and tweets. First, the data was imported into Python to perform a sense-check, ensuring accuracy, consistency, and reliability. Descriptive statistics and metadata were then used to understand the data better before conducting more in-depth analysis. Descriptive statistics provided a quick summary of central tendency, dispersion, and overall distribution, while metadata offered insights into the context and structure of the datasets. (A)

Next, data cleaning was performed to investigate duplicates in each dataset, ensuring quality and reliability. In the actual_duration dataset, 21,604 duplicates were found and removed, while no duplicates were found in the remaining datasets. (B)

Furthermore, specific datasets were merged for comprehensive analysis to answer key business questions. An outer join was used to retain all rows from each dataset, ensuring no data was excluded and filling in missing values where there were no matches. This method preserved the complete information from both datasets, enabling a thorough exploration. (C)

Lastly, specific data like outlier or irrelevant data (any unmapped data were moved from the analysis) were filtered out to ensure high quality analysis and clear data visualisations. (D)

3) Visualisation and insights

Initially, the dataset of actual_duration contained 106 locations. Notably, NHS North-West London ICB recorded the highest number of appointments at 12,142,390, followed by NHS North-East London ICB with 9,558,891 appointments. As illustrated in Figure 1, three of the top five locations with the highest number of appointments are situated in London.

Additionally, there are 18 national categories, with General Consultation Routine having the highest number of appointments, followed by General Consultation Acute as shown in Figure 2. Conversely, Group Consultation and Group Education have the lowest number of appointments at 60,632, followed by Non-contractual Chargeable Work as depicted in Figure 3. (E)

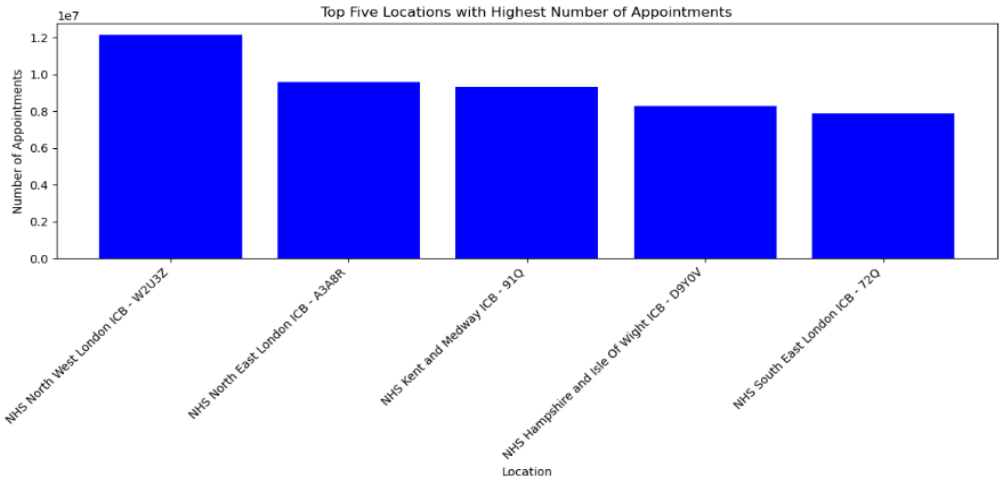

Figure 1: Top Five Location with the Highest Number of Appointments

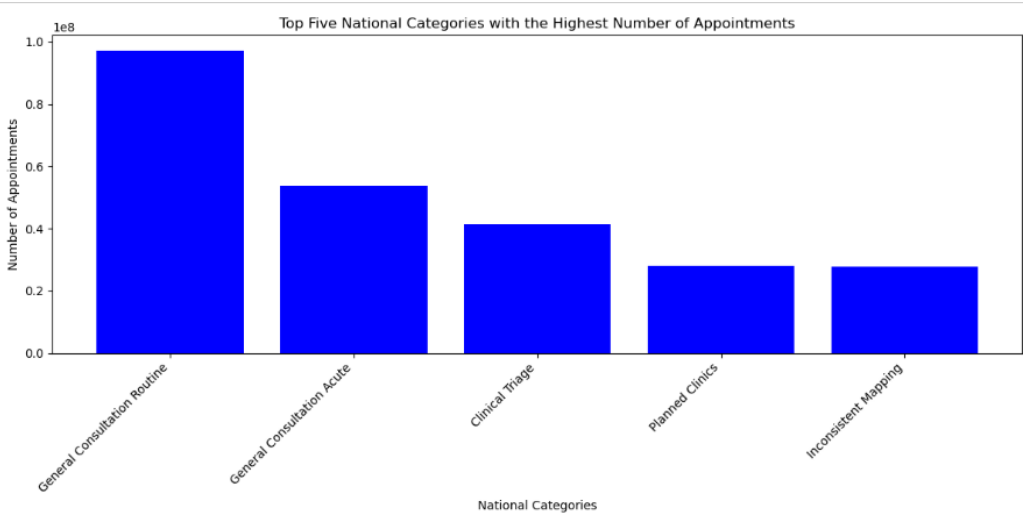

Figure 2: Highest Number of Appointments for National Categories

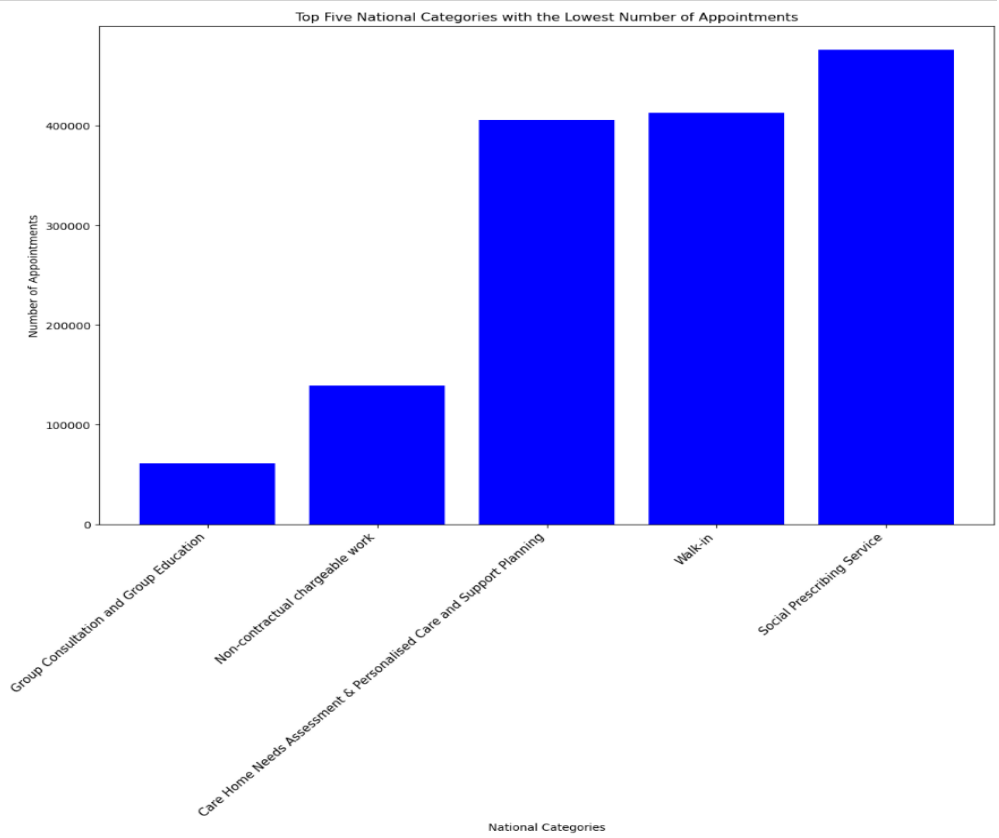

Figure 3: Lowest Number of Appointments for National Categories

The appointments were scheduled between 2021-08-01 and 2022-06-30. Within the North-West London NHS, the highest number of appointments was recorded. Delving deeper, General Practice had the highest number of appointments within North-West London, totalling 4,804,239 during this period. November recorded the highest number of appointments, reaching 30,405,070. Notably, three of the top four months with the highest appointments were in the autumn season (September to November).

Rank	Month		No of Appointments
1	November 2021		30405070
2	October 2021		30303834
3	March 2022		29595038
4	September 2021		28522501

Figure 4: Top 4 Highest Appointments per month.

General Practice accounted for the overwhelming majority of appointments, with a total of 270,811,691. Other service settings, including Primary Care Network, Extended Access Provision, and Others, were considered outliers due to their relatively lower numbers. The line graph indicated an increasing trend in the number of appointments starting in August, peaking in November for General Practice, with a secondary peak in March. A similar trend is observed in Figure 6 for appointments across different service settings and seasons, suggesting a seasonal influence on appointment numbers.

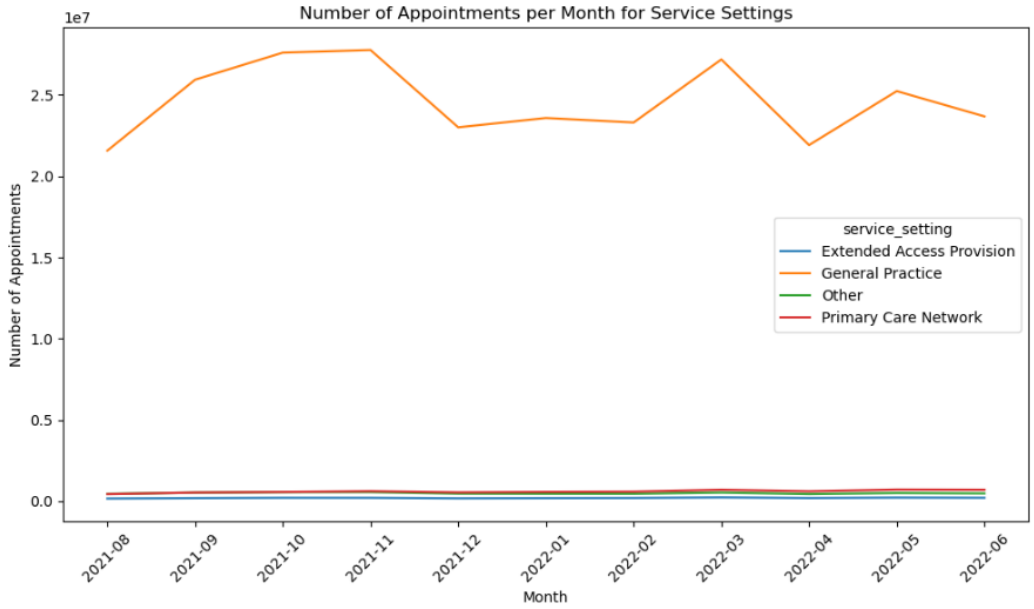

Figure 5: Appointments Number for Service Settings Over Time

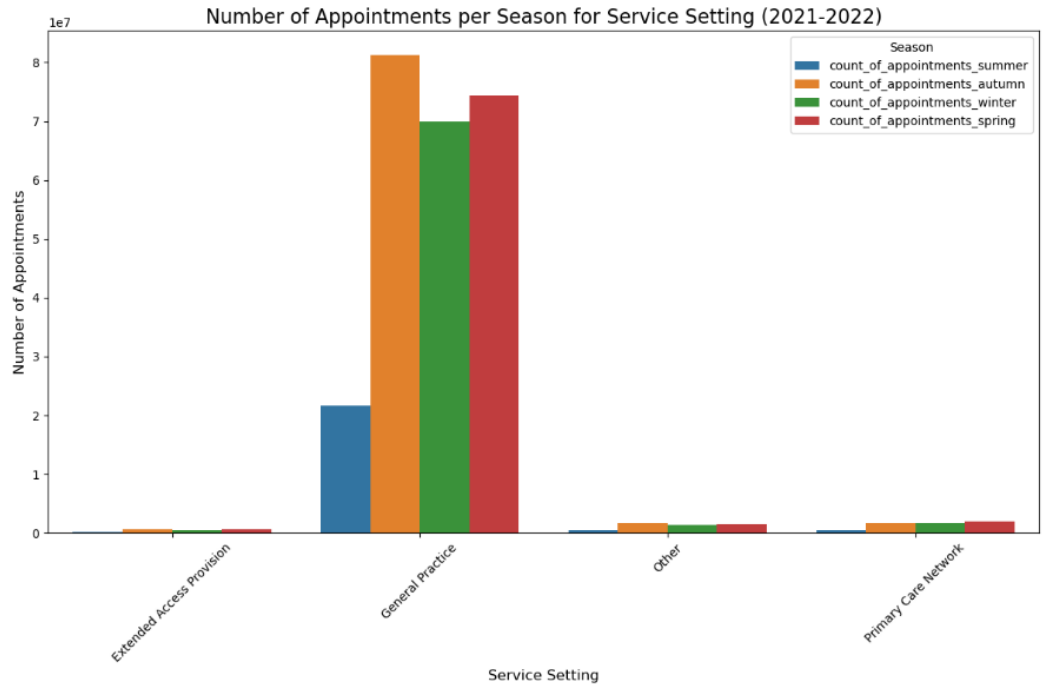

Figure 6: Appointment Numbers for Service Setting across different Season.

For context types illustrated in Figure 7, Care Related Encounter shows a similar trend to General Practice, with appointments rising in August and peaking in November at 26,282,778, followed by a secondary peak in March.

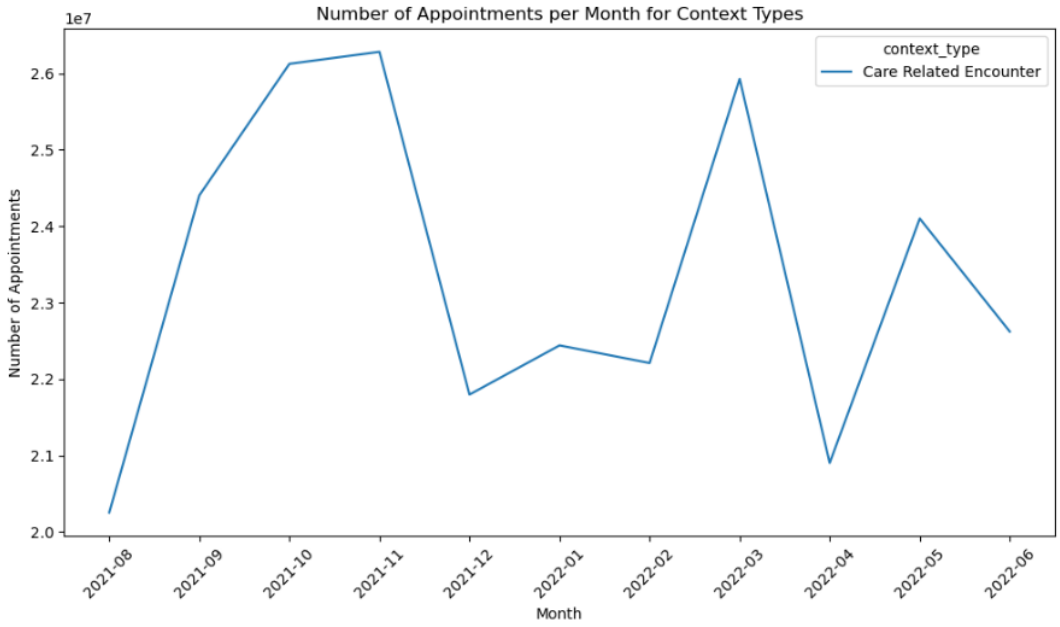

Figure 7: Appointments Number for Context Types Over Time

Among the National Categories, Structured Medication Review recorded the highest number of appointments between August 2021 and June 2022, followed by Care Home Visit and Care Home Needs Assessment & Personalised Care and Support Planning. These categories exhibited similar trends, with March 2022 having the highest number of appointments and August 2021 the lowest. (F)

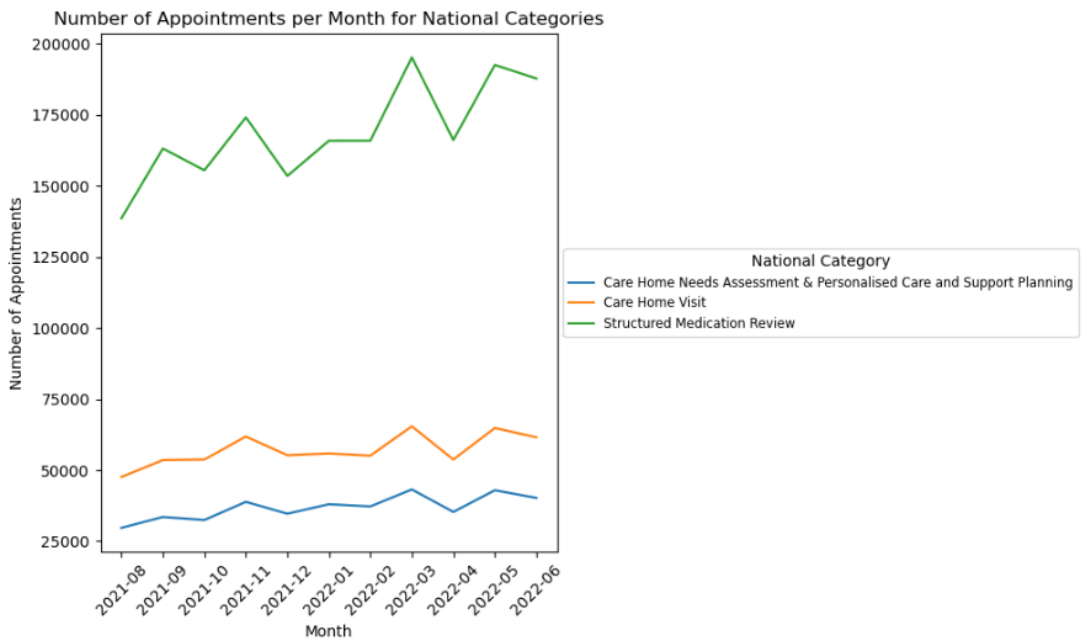

Figure 8: Appointments Number for National Categories Over Time

4) Patterns and Predictions

To address the question of actual resource utilisation, the analysis highlights several key findings: (F)

1. **Correlation Between Appointments and Time from Booking:** Figure 9 reveals a negative correlation between the number of appointments and the time from booking to appointments. However, appointments scheduled within one day deviated from this trend, indicating potential inefficiencies in resource utilisation, particularly in appointment and staff resource management.

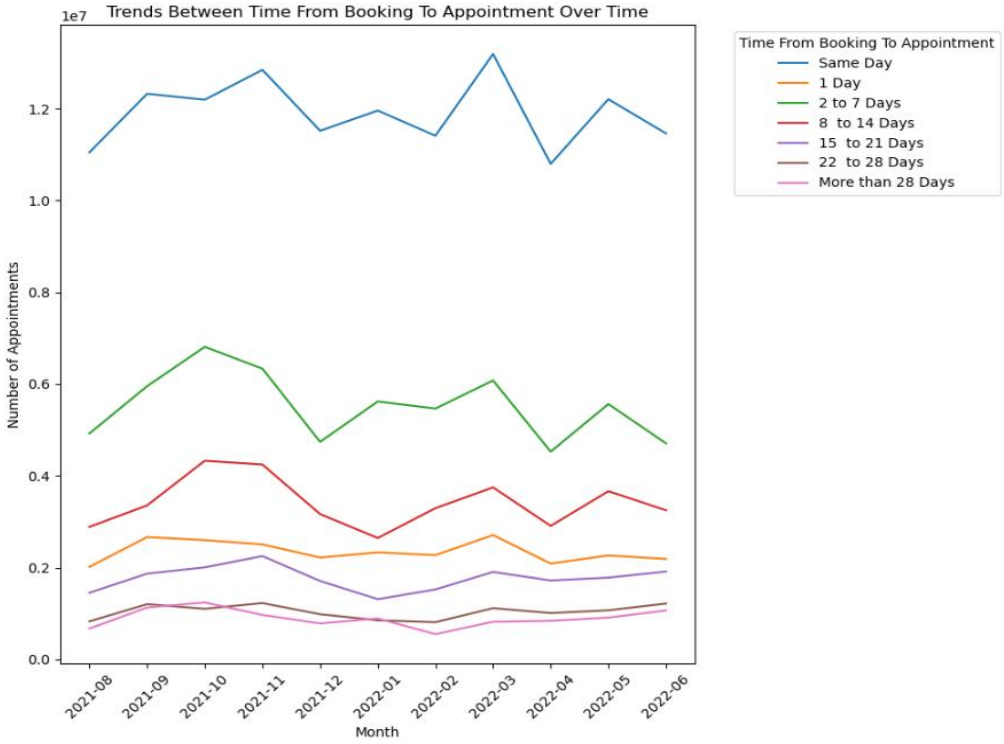

Figure 9: Trends Between Time from Booking to Appointment Over Time.

2. **Appointment Mode Utilisation:** The most used appointment modes observed were Face-to-Face followed by Telephone, with a smaller proportion allocated to Home Visit and Online appointments. As a result, resource allocation should prioritise Face-to-Face and Telephone appointments to accommodate the majority of patient needs efficiently.

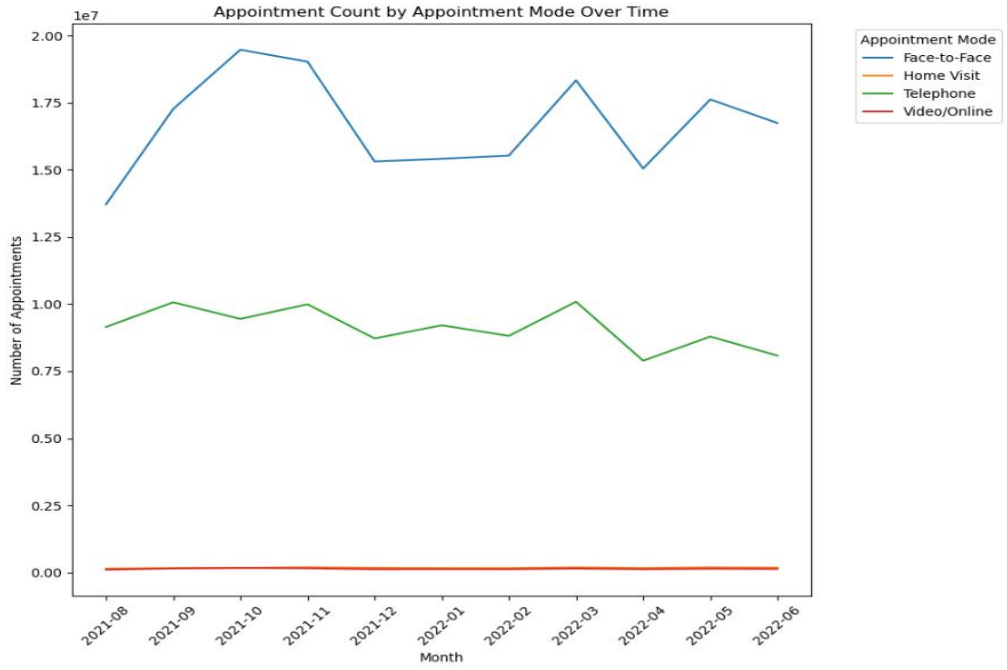

Figure 10: Trends for different Appointment Mode

3. **Missed Appointments (DNAs):** Approximately 4.55%, equivalent to 54,600 patients, did not attend their appointments as shown in Figure 11. Considering the NHS guideline of a maximum capacity of 1,200,000 appointments per day, this represents a significant number. Such non-attendance not only wastes resources like staff time, facility space, and equipment but also denies other patients access to vital care. Urgent implementation of additional strategies is imperative to address this issue effectively.

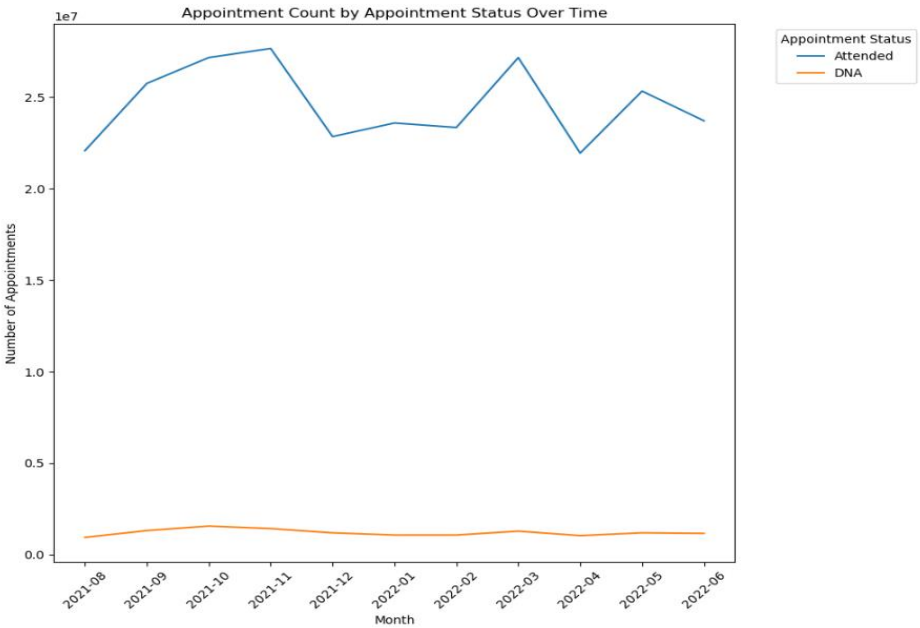

Figure 11: Distribution of Appointment Status

To address the question of adequate staff and capacity in the networks, the analysis highlights several key findings: (G)

1. **Staff Utilisation:** Daily appointment rates fell below the NHS's recommended maximum capacity of 1,200,000 per day shown in figure 12. This suggests that the current staff levels are sufficient to handle the existing workload, alleviating the immediate need for an increase in staffing.

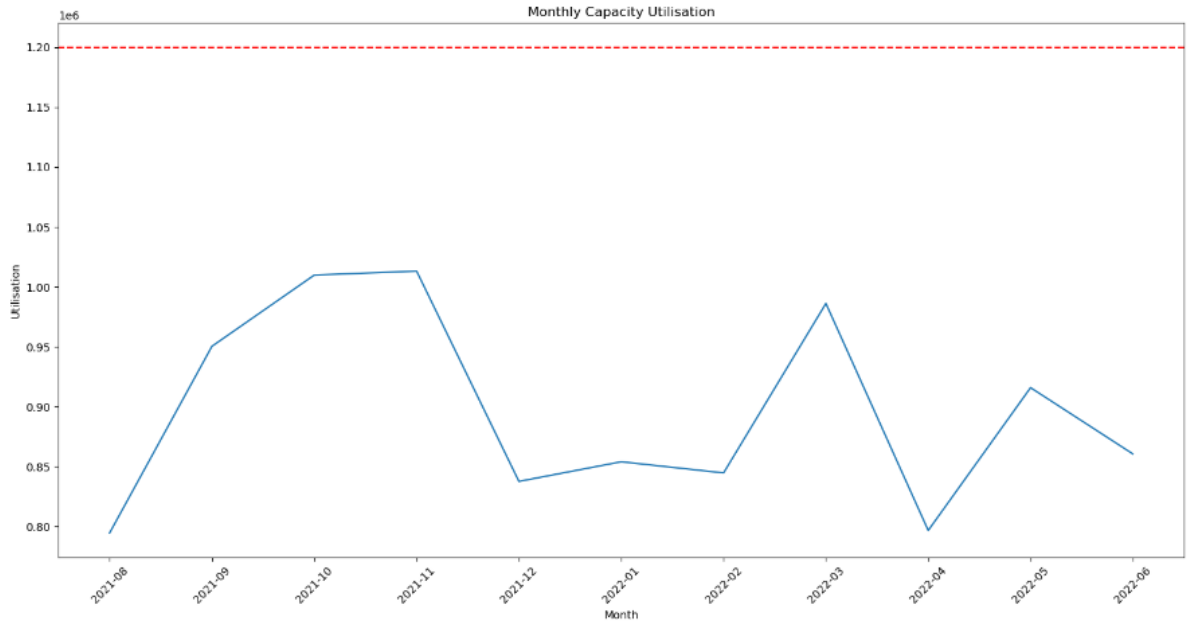

Figure 12: Monthly Capacity Utilisation

2. **Distribution of different health care professional:** Figure 13 highlights a well-balanced distribution of healthcare professionals across the networks. This diversity ensures comprehensive coverage for patients with varying needs and conditions. However, additional examination of specific roles may be warranted in certain cases to optimise care delivery further.

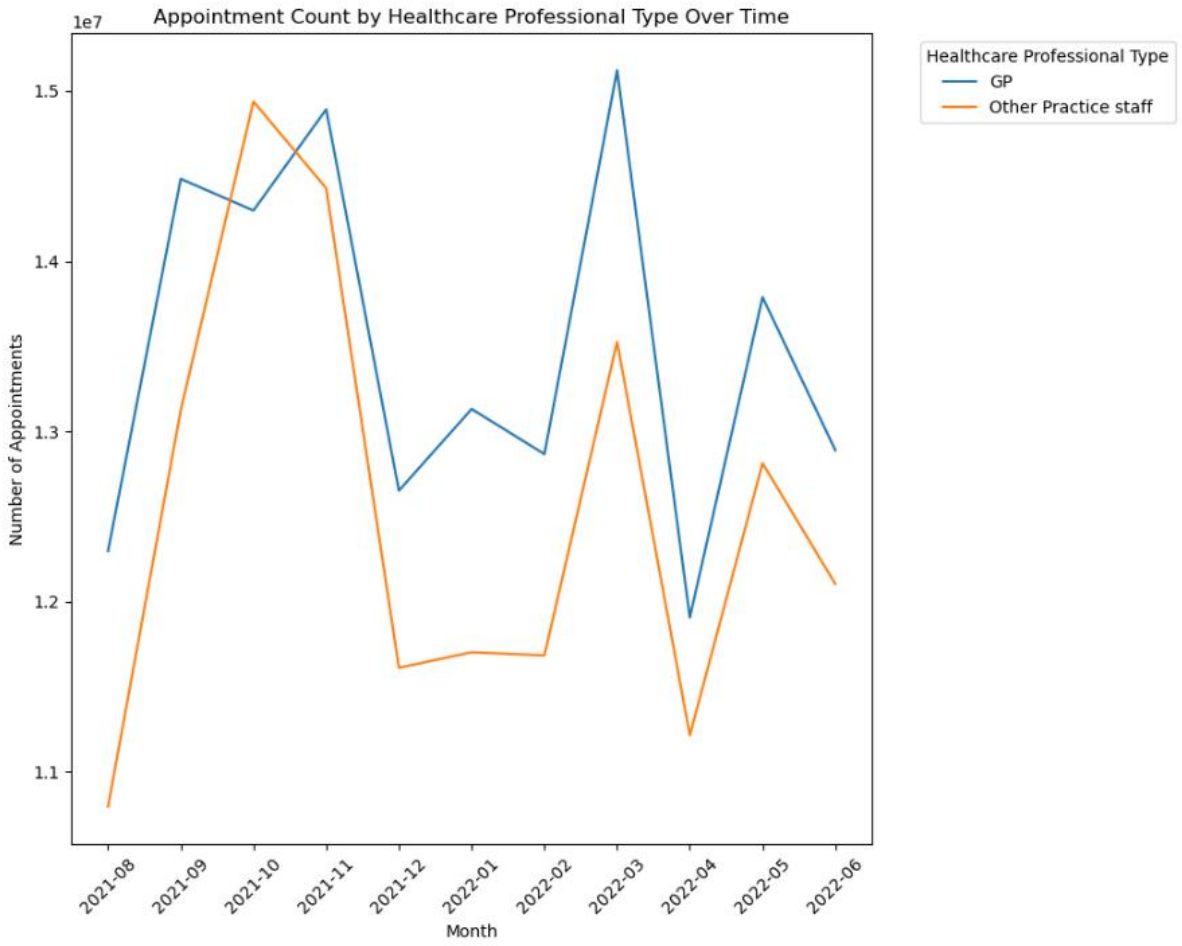

Figure 13: Distribution of Health Care Professional Type

Recommendation:

The lack of a time frame in the Twitter data constrained analytical approaches and limited exploration. Adding a time frame could unlock the data's full potential for more robust analysis and valuable insights.

Appendices

A) Import Data in Python

I first by importing Pandas and Numpy which are the necessary libraries for the analysis. Then I loaded the all 4 data files in the python and sense-checked them using the following codes.

```
import pandas as pd
import numpy as np

ad = pd.read_csv('actual_duration.csv')

# Sense-check the DataFrame
print("Column names:")
print(ad.columns)

print("Number of rows and columns:")
print(ad.shape)

print("Data types:")
print(ad.dtypes)

print("Number of missing values:")
print(ad.isnull().sum())

# Determine descriptive statistics
print("Descriptive statistics:")
print(ad.describe())

# Determine metadata
print("Metadata:")
print(ad.info())
```

```
ar = pd.read_csv('appointments_regional.csv')

# Sense-check the DataFrame
print("Column names:")
print(ar.columns)

print("Number of rows and columns:")
print(ar.shape)

print("Data types:")
print(ar.dtypes)

print("Number of missing values:")
print(ar.isnull().sum())

# Determine descriptive statistics
print("Descriptive statistics:")
print(ar.describe())

# Determine metadata
print("Metadata:")
print(ar.info())
```

```
nc = pd.read_excel('national_categories.xlsx')

# Sense-check the DataFrame
print("Column names:")
print(nc.columns)

print("Number of rows and columns:")
print(nc.shape)

print("Data types:")
print(nc.dtypes)

print("Number of missing values:")
print(nc.isnull().sum())

# Determine descriptive statistics
print("Descriptive statistics:")
print(nc.describe())

# Determine metadata
print("Metadata:")
print(nc.info())
```

```
tweets = pd.read_csv('tweets.csv')

# Sense-check the DataFrame
print("Column names:")
print(tweets.columns)

print("Number of rows and columns:")
print(tweets.shape)

print("Data types:")
print(tweets.dtypes)

print("Number of missing values:")
print(tweets.isnull().sum())

# Determine descriptive statistics
print("Descriptive statistics:")
print(tweets.describe())

# Determine metadata
print("Metadata:")
print(tweets.info())
```

B) Check and remove duplicates for each data file

I checked all the data and found out that only ar file contains duplicates. Therefore, I wrote a code to remove them.

```
# Check for duplicates
duplicate_rows_tweets = tweets.duplicated()
num_duplicates_tweets = duplicate_rows_tweets.sum()
num_duplicates_tweets
```

```
# Check for duplicates
duplicate_rows_ad = ad.duplicated()
num_duplicates_ad = duplicate_rows_ad.sum()
num_duplicates_ad
```

```
# Check for duplicates
duplicate_rows_nc = nc.duplicated()
num_duplicates_nc = duplicate_rows_nc.sum()
num_duplicates_nc
```

```
# Check for duplicates
duplicate_rows_ar = ar.duplicated()
num_duplicates_ar = duplicate_rows_ar.sum()
num_duplicates_ar
```

```
21604
```

```
# Remove duplicates
ar.drop_duplicates(inplace=True)
```

C) Merging data

```
# Filter out 'Unmapped' service settings
nc_filtered = nc[nc['service_setting'] != 'Unmapped']

# 1. Filter the data for appointments from June to August 2021
nc_summer = nc_filtered[(nc_filtered['appointment_month'] >= '2021-06') & (nc_filtered['appointment_month'] <= '2021-08')]
nc_autumn = nc_filtered[(nc_filtered['appointment_month'] >= '2021-09') & (nc_filtered['appointment_month'] <= '2021-11')]
nc_winter = nc_filtered[(nc_filtered['appointment_month'] >= '2021-12') & (nc_filtered['appointment_month'] <= '2022-02')]
nc_spring = nc_filtered[(nc_filtered['appointment_month'] >= '2022-03') & (nc_filtered['appointment_month'] <= '2022-05')]

# 2. Aggregate the appointments per service setting
nc_summer_agg = nc_summer.groupby('service_setting')['count_of_appointments'].sum().reset_index()
nc_autumn_agg = nc_autumn.groupby('service_setting')['count_of_appointments'].sum().reset_index()
nc_winter_agg = nc_winter.groupby('service_setting')['count_of_appointments'].sum().reset_index()
nc_spring_agg = nc_spring.groupby('service_setting')['count_of_appointments'].sum().reset_index()


# Merge the aggregated data into a single DataFrame
nc_seasons = pd.merge(nc_summer_agg, nc_autumn_agg, on='service_setting', how='outer', suffixes=('_summer', '_autumn'))
nc_seasons = pd.merge(nc_seasons, nc_winter_agg, on='service_setting', how='outer')
nc_seasons = pd.merge(nc_seasons, nc_spring_agg, on='service_setting', how='outer', suffixes=('_winter', '_spring'))
```

D) Filtered out unnecessary data

This is one of the example that I used to filter out the value = ‘Unmapped’ in the service_setting data.

```
# Filter out 'Unmapped' service settings
nc_filtered = nc[nc['service_setting'] != 'Unmapped']
```

Another example

```
# Filter out 'Unknown / Data Quality' values
ar_filtered_booking_appointment = ar_agg[
    (ar_agg['time_between_book_and_appointment'] != 'Unknown / Data Quality')
]
```

E) Code for visualisations and insights

```
#What are the five locations with the highest number of appointments?

# Group the data by 'sub_icb_location_name' and sum the appointments
location_appointments = nc.groupby('sub_icb_location_name')['count_of_appointments'].sum()

# Sort the groups in descending order based on the total appointments
location_appointments_sorted = location_appointments.sort_values(ascending=False)

# Select the top five locations
top_five_locations = location_appointments_sorted.head(5)

print("Top five locations with the highest number of appointments:")
print(top_five_locations)
```

```
Top five locations with the highest number of appointments:
sub_icb_location_name
NHS North West London ICB - W2U3Z              12142390
NHS North East London ICB - A3A8R               9588891
NHS Kent and Medway ICB - 91Q                   9286167
NHS Hampshire and Isle Of Wight ICB - D9Y0V     8288102
NHS South East London ICB - 72Q                 7850170
Name: count_of_appointments, dtype: int64
```

```
import matplotlib.pyplot as plt

# Plotting bar chart
plt.figure(figsize=(12, 6))
plt.bar(top_five_locations.index, top_five_locations.values, color='blue')
plt.xlabel('Location')
plt.ylabel('Number of Appointments')
plt.title('Top Five Locations with Highest Number of Appointments')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

```
#Which service setting has the highest number of appointments?

# Group the data by 'sub_icb_location_name' and sum the appointments
service_setting_appointments = nc.groupby('service_setting')['count_of_appointments'].sum()

# Sort the groups in descending order based on the total appointments
service_setting_appointments_sorted = service_setting_appointments.sort_values(ascending=False)

# Select the top five locations
top_five_service_setting = service_setting_appointments_sorted.head(5)

print("Top five service settings with the highest number of appointments:")
print(top_five_service_setting)t
```

```
Top five service settings with the highest number of appointments:
service_setting
General Practice             270811691
Unmapped                      11080810
Primary Care Network           6557386
Other                          5420076
Extended Access Provision      2176807
Name: count_of_appointments, dtype: int64
```

```
# Filtering out 'Unmapped' service setting
filtered_service_setting = service_setting_appointments_sorted.drop('Unmapped', errors='ignore')

# Select the top five service settings (excluding 'Unmapped')
top_five_service_setting = filtered_service_setting.head()

# Plotting bar chart
plt.bar(top_five_service_setting.index, top_five_service_setting.values, color='skyblue')
plt.xlabel('Service Setting')
plt.ylabel('Number of Appointments')
plt.title('Service Settings with Number of Appointments (Excluding "Unmapped")')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

```
#Which national categories has the highest number of appointments?

# Group the data by 'sub_icb_location_name' and sum the appointments
national_categories_appointments = nc.groupby('national_category')['count_of_appointments'].sum()

# Sort the groups in descending order based on the total appointments
national_catergoies_appointments_sorted = national_categories_appointments.sort_values(ascending=False)

# Select the top five national categories
top_five_national_categories = national_catergoies_appointments_sorted.head(5)

print("Top five service settings with the highest number of appointments:")
print(top_five_national_categories)
```

```
Top five service settings with the highest number of appointments:
national_category
General Consultation Routine    97271522
General Consultation Acute      53691150
Clinical Triage                 41546964
Planned Clinics                 28019748
Inconsistent Mapping            27890802
Name: count_of_appointments, dtype: int64
```

```
# Plotting bar chart
plt.figure(figsize=(12, 6))
plt.bar(top_five_national_categories.index, top_five_national_categories.values, color='blue')
plt.xlabel('National Categories')
plt.ylabel('Number of Appointments')
plt.title('Top Five National Categories with the Highest Number of Appointments')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

```
#Which national categories has the lowest number of appointments?

# Group the data by 'sub_icb_location_name' and sum the appointments
national_categories_appointments = nc.groupby('national_category')['count_of_appointments'].sum()

# Sort the groups in descending order based on the total appointments
national_categories_appointments_sorted = national_categories_appointments.sort_values(ascending=True)

# Select the bottom five national categories
bottom_five_national_categories = national_categories_appointments_sorted.head(5)

print("Bottom five service settings with the highest number of appointments:")
print(bottom_five_national_categories)
```

```
Bottom five service settings with the highest number of appointments:
national_category
Group Consultation and Group Education                                60632
Non-contractual chargeable work                                      138911
Care Home Needs Assessment & Personalised Care and Support Planning  405904
Walk-in                                                              412438
Social Prescribing Service                                           475828
Name: count_of_appointments, dtype: int64
```

```
# Plotting bar chart
plt.figure(figsize=(12, 12))
plt.bar(bottom_five_national_categories.index, bottom_five_national_categories.values, color='blue')
plt.xlabel('National Categories')
plt.ylabel('Number of Appointments')
plt.title('Top Five National Categories with the Lowest Number of Appointments')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

F) Code for visualisations (line graph and bar chart)

```
# What dates were appointments scheduled?

min_date_nc = nc['appointment_date'].min()
max_date_nc = nc['appointment_date'].max()

# Display the results
print("The appointments were scheduled between:", min_date_nc, "and", max_date_nc)
```

```
The appointments were scheduled between: 2021-08-01 00:00:00 and 2022-06-30 00:00:00
```

```
# Which service setting reported the most appointments in North West London from 1 January to 1 June 2022?

# Create a subset of the nc DataFrame
nc_subset = nc[(nc['sub_icb_location_name'].str.contains('W2U3Z', case=False)) &
               (nc['appointment_date'] >= '2022-01-01') &
               (nc['appointment_date'] <= '2022-06-01')]

# Group the data by service setting and sum the appointments
service_setting_counts = nc_subset.groupby('service_setting')['count_of_appointments'].sum()

# Find the service setting(s) with the highest number of appointments
most_appointments_service_setting = service_setting_counts.idxmax()
num_appointments_service_setting = service_setting_counts.max()

# Display the results
print("Service Setting with the most appointments in North West London:")
print(most_appointments_service_setting)
print(num_appointments_service_setting)
```

```
Service Setting with the most appointments in North West London:
General Practice
4804239
```

```
# Which month had the highest number of appointments?

# Convert appointment_date column to datetime if not already
nc['appointment_date'] = pd.to_datetime(nc['appointment_date'])

# Group appointments by year and month, then calculate the total number of appointments for each month
appointments_per_month = nc.groupby([nc['appointment_date'].dt.year, nc['appointment_date'].dt.month])['count_of_appointments'].

# Sorting from largest to smallest
sorted_appointments_per_month = appointments_per_month.sort_values(ascending=False)

# Display the results
print("Number of appointments:")
print(sorted_appointments_per_month)
print("Month with the highest appointments")
print(sorted_appointments_per_month.head(1))
```

```
Number of appointments:
appointment_date  appointment_date
2021              11                30405070
                  10                30303834
2022              3                 29595038
2021              9                 28522501
2022              5                 27495508
                  6                 25828078
                  1                 25635474
                  2                 25355260
2021              12                25140776
2022              4                 23913060
2021              8                 23852171
Name: count_of_appointments, dtype: int64
Month with the highest appointments
appointment_date  appointment_date
2021              11                30405070
Name: count_of_appointments, dtype: int64
```

```
import seaborn as sns
import matplotlib.pyplot as plt

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

# Aggregate the appointments per month and service settings
nc_ss = nc.groupby(['appointment_month', 'service_setting'])['count_of_appointments'].sum().reset_index()

# Filtered out unneccessary data
nc_ss_filtered = nc_ss[(nc_ss['service_setting'] != 'Unmapped')]

# View the data
print(nc_ss_filtered)

# Create a lineplot with Seaborn
plt.figure(figsize=(10, 6))
sns.lineplot(x='appointment_month', y='count_of_appointments', hue='service_setting', data=nc_ss_filtered, errorbar=None)
plt.title('Number of Appointments per Month for Service Settings')
plt.xlabel('Month')
plt.ylabel('Number of Appointments')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
# Filter out 'Unmapped' service settings
nc_filtered = nc[nc['service_setting'] != 'Unmapped']

# 1. Filter the data for appointments from June to August 2021
nc_summer = nc_filtered[(nc_filtered['appointment_month'] >= '2021-06') & (nc_filtered['appointment_month'] <= '2021-08')]
nc_autumn = nc_filtered[(nc_filtered['appointment_month'] >= '2021-09') & (nc_filtered['appointment_month'] <= '2021-11')]
nc_winter = nc_filtered[(nc_filtered['appointment_month'] >= '2021-12') & (nc_filtered['appointment_month'] <= '2022-02')]
nc_spring = nc_filtered[(nc_filtered['appointment_month'] >= '2022-03') & (nc_filtered['appointment_month'] <= '2022-05')]

# 2. Aggregate the appointments per service setting
nc_summer_agg = nc_summer.groupby('service_setting')['count_of_appointments'].sum().reset_index()
nc_autumn_agg = nc_autumn.groupby('service_setting')['count_of_appointments'].sum().reset_index()
nc_winter_agg = nc_winter.groupby('service_setting')['count_of_appointments'].sum().reset_index()
nc_spring_agg = nc_spring.groupby('service_setting')['count_of_appointments'].sum().reset_index()


# Merge the aggregated data into a single DataFrame
nc_seasons = pd.merge(nc_summer_agg, nc_autumn_agg, on='service_setting', how='outer', suffixes=('_summer', '_autumn'))
nc_seasons = pd.merge(nc_seasons, nc_winter_agg, on='service_setting', how='outer')
nc_seasons = pd.merge(nc_seasons, nc_spring_agg, on='service_setting', how='outer', suffixes=('_winter', '_spring'))

# Rename the columns appropriately
nc_seasons.columns = ['service_setting', 'count_of_appointments_summer', 'count_of_appointments_autumn', 'count_of_appointments_

# Melt the DataFrame to long format for seaborn
nc_seasons_melted = pd.melt(nc_seasons, id_vars='service_setting', var_name='season', value_name='count_of_appointments')

# Create a bar plot with Seaborn
plt.figure(figsize=(12, 8))
sns.barplot(x='service_setting', y='count_of_appointments', hue='season', data=nc_seasons_melted)

# Add plot title and labels
plt.title('Number of Appointments per Season for Service Setting (2021-2022)', fontsize=16)
plt.xlabel('Service Setting', fontsize=12)
plt.ylabel('Number of Appointments', fontsize=12)
plt.xticks(rotation=45)
plt.tight_layout()

# Add legend
plt.legend(title='Season', loc='upper right')

plt.show()
```

```
# Aggregate the appointments per month and context type
nc_ct = nc.groupby(['appointment_month', 'context_type'])['count_of_appointments'].sum().reset_index()

# Filter the DataFrame to exclude the 'Unmapped' and 'Inconsistent Mapping' category
nc_ct_filtered = nc_ct[(nc_ct['context_type'] != 'Unmapped') & (nc_ct['context_type'] != 'Inconsistent Mapping')]

# View the new DataFrame
print(nc_ct_filtered.sort_values(by='count_of_appointments', ascending=False))

# Create a lineplot with Seaborn
plt.figure(figsize=(10, 6))
sns.lineplot(x='appointment_month', y='count_of_appointments', hue='context_type', data=nc_ct_filtered, errorbar=None)
plt.title('Number of Appointments per Month for Context Types')
plt.xlabel('Month')
plt.ylabel('Number of Appointments')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
# Aggregate the appointments per month and national category
nc_nc = nc.groupby(['appointment_month', 'national_category'])['count_of_appointments'].sum().reset_index()

# Get the top 5 most common national categories by their appointment count
top_national_categories = nc_nc['national_category'].value_counts().head(5).index

# Filter the DataFrame to include only the top 5 categories
nc_nc_top = nc_nc[nc_nc['national_category'].isin(top_national_categories)]

# Further filter the DataFrame to exclude the 'Unmapped' category
nc_nc_top_filtered = nc_nc_top[(nc_nc_top['national_category'] != 'Unmapped') &
                               (nc_nc_top['national_category'] != 'Unplanned Clinical Activity')]

# Create a line plot with Seaborn
plt.figure(figsize=(10, 6))
sns.lineplot(x='appointment_month', y='count_of_appointments', hue='national_category', data=nc_nc_top_filtered, errorbar=None)

# Add plot title and labels
plt.title('Number of Appointments per Month for National Categories')
plt.xlabel('Month')
plt.ylabel('Number of Appointments')

# Rotate the x-axis labels for better readability
plt.xticks(rotation=45)

# Adjust the font size of the legend and place it on the right
plt.legend(title='National Category', fontsize='small', loc='center left', bbox_to_anchor=(1, 0.5))

# Ensure the plot layout is tight to avoid clipping
plt.tight_layout()

# Show the plot
plt.show()
```

G) Code to analysis for actual utilisation of resources.

```
# Question 5
# Are there any trends in time between booking an appointment?

# Group by 'appointment_month' and 'time_between_book_and_appointment' and sum 'count_of_appointments'
ar_agg = ar.groupby(['appointment_month', 'time_between_book_and_appointment'])['count_of_appointments'].sum().reset_index()

ar_agg_filtered = ar_agg[(ar_agg['appointment_month'] >= '2021-08') & (ar_agg['appointment_month'] <= '2022-06')]

# Filter out 'Unknown / Data Quality' values
ar_filtered_booking_appointment = ar_agg_filtered[
    (ar_agg_filtered['time_between_book_and_appointment'] != 'Unknown / Data Quality')
]

# Change datatype of 'appointment_month' to string for ease of plotting
ar_filtered_booking_appointment['appointment_month'] = ar_filtered_booking_appointment['appointment_month'].astype(str)

# Create a line plot to show the trends in time between booking and appointment over time
plt.figure(figsize=(10, 8))

# Define the order of categories for the legend
legend_order = ['Same Day', '1 Day', '2 to 7 Days', '8  to 14 Days', '15  to 21 Days', '22  to 28 Days', 'More than 28 Days']

# Create the line plot
sns.lineplot(x='appointment_month', y='count_of_appointments', hue='time_between_book_and_appointment',
             data=ar_filtered_booking_appointment, errorbar=None, hue_order=legend_order)

# Set the title and labels
plt.title('Trends Between Time From Booking To Appointment Over Time')
plt.xlabel('Month')
plt.ylabel('Number of Appointments')
plt.xticks(rotation=45)

# Show legend
plt.legend(title='Time From Booking To Appointment', bbox_to_anchor=(1.05, 1), loc='upper left')

# Show plot
plt.tight_layout()
plt.show()
```

```
# Question 4
# Are there changes in terms of appointment type and the busiest months?

ar_agg = ar.groupby(['appointment_month', 'appointment_mode'])['count_of_appointments'].sum().reset_index()

ar_agg_filtered = ar_agg[(ar_agg['appointment_month'] >= '2021-08') & (ar_agg['appointment_month'] <= '2022-06')]

# Filter out the 'Unknown' value in 'appointment_mode'
ar_filtered_mode = ar_agg_filtered[ar_agg_filtered['appointment_mode'] != 'Unknown']

# Change datatype of 'appointment_month' to string
ar_filtered_mode['appointment_month'] = ar_filtered_mode['appointment_month'].astype(str)

# Create lineplot for number of monthly visits
plt.figure(figsize=(10, 8))

# Create a line plot to show the count of appointments for each appointment mode over time
sns.lineplot(x='appointment_month', y='count_of_appointments', hue='appointment_mode', data=ar_filtered_mode, errorbar=None)

# Set the title and labels
plt.title('Appointment Count by Appointment Mode Over Time')
plt.xlabel('Month')
plt.ylabel('Number of Appointments')
plt.xticks(rotation=45)

# Show legend
plt.legend(title='Appointment Mode', bbox_to_anchor=(1.05, 1), loc='upper left')

# Show plot
plt.tight_layout()
plt.show()
```

```
# Question 3
# Are there significant changes in whether or not visits are attended?

ar_agg = ar.groupby(['appointment_month', 'appointment_status'])['count_of_appointments'].sum().reset_index()

ar_agg_filtered = ar_agg[(ar_agg['appointment_month'] >= '2021-08') & (ar_agg['appointment_month'] <= '2022-06')]

# Filter out 'Unknown' values
ar_filtered_status = ar_agg_filtered[ar_agg_filtered['appointment_status'] != 'Unknown']

# Change datatype of appointment_month to string
ar_filtered_status['appointment_month'] = ar_filtered_status['appointment_month'].astype(str)

# Create lineplot for number of monthly visits
plt.figure(figsize=(10, 8))

# Create a line plot to show the count of appointments for each healthcare professional type over time
sns.lineplot(x='appointment_month', y='count_of_appointments', hue='appointment_status', data=ar_filtered_status, errorbar=None)

# Set the title and labels
plt.title('Appointment Count by Appointment Status Over Time')
plt.xlabel('Month')
plt.ylabel('Number of Appointments')
plt.xticks(rotation=45)

# Show legend
plt.legend(title='Appointment Status', bbox_to_anchor=(1.05, 1), loc='upper left')

# Show plot
plt.tight_layout()
plt.show()
```

H) Code to analysis adequate staff and capacity in the network.

```
ar_df = ar.groupby('appointment_month')['count_of_appointments'].sum().reset_index()

ar_df_filtered = ar_df[(ar_df['appointment_month'] >= '2021-08') & (ar_df['appointment_month'] <= '2022-06')]

# Calculate the average utilisation of service
ar_df_filtered['utilisation'] = ar_df_filtered['count_of_appointments'] / 30
ar_df_filtered['utilisation'] = ar_df_filtered['utilisation'].apply(lambda x: round(x , 1))

# View the DataFrame
print(ar_df_filtered.head())
```

```
# Change datatype of appointment_month to string
ar_df_filtered['appointment_month'] = ar_df_filtered['appointment_month'].astype(str)
```

```
# Create lineplot for monthly capacity utilisation
plt.figure(figsize=(15, 8))
sns.lineplot(x='appointment_month', y='utilisation', data=ar_df_filtered)

# NHS Guideline for daily appointments
plt.axhline(y=1200000, color='r', linestyle='--', label='Threshold: 1,200,000')
plt.xticks(rotation=45)
plt.title('Monthly Capacity Utilisation')
plt.xlabel('Month')
plt.ylabel('Utilisation')
plt.tight_layout()
plt.show()
```

```
# Question 2
# How do the healthcare professional types differ over time?

ar_agg = ar.groupby(['appointment_month', 'hcp_type'])['count_of_appointments'].sum().reset_index()

ar_agg_filtered = ar_agg[(ar_agg['appointment_month'] >= '2021-08') & (ar_agg['appointment_month'] <= '2022-06')]

# Filter out 'Unknown' values
ar_filtered_hcp = ar_agg_filtered[ar_agg_filtered['hcp_type'] != 'Unknown']

# Change datatype of appointment_month to string
ar_filtered_hcp['appointment_month'] = ar_filtered_hcp['appointment_month'].astype(str)

# Create lineplot for number of monthly visits
plt.figure(figsize=(10, 8))

# Create a line plot to show the count of appointments for each healthcare professional type over time
sns.lineplot(x='appointment_month', y='count_of_appointments', hue='hcp_type', data=ar_filtered_hcp, errorbar=None)

# Set the title and labels
plt.title('Appointment Count by Healthcare Professional Type Over Time')
plt.xlabel('Month')
plt.ylabel('Number of Appointments')
plt.xticks(rotation=45)

# Show legend
plt.legend(title='Healthcare Professional Type', bbox_to_anchor=(1.05, 1), loc='upper left')

# Show plot
plt.tight_layout()
plt.show()
```