Technical report

NHS resource utilisation: exploratory and diagnostic analysis

Prepared by: Yuliya Pauzunova
28 October 2024

1. Background/context of the business scenario

Client: National Health Services (NHS)

Business problem: Design and implement a financial and operational strategy to ensure NHS infrastructure and resources match population capacity

Analytical problem: Identify utilisation trends in the NHS network to inform the decision-making process around NHS budget allotment and operational efficiency

The project will aim to address the following **analytical objectives**/questions:

- Assess the full capacity and actual utilisation of existing infrastructure and resources
- Identify utilisation trends and patterns, and possible reasons for this
- Recommend potential measures to reduce or eliminate identified inefficiencies

2. Analytical approach

Note: the process is iterative and does not exactly describe the original workflow.

1. Import necessary libraries and datasets

```
# Import the necessary libraries.
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.dates import *
from matplotlib.ticker import FuncFormatter
import seaborn as sns
import requests
import bs4
from bs4 import BeautifulSoup
import re
from datetime import datetime
import holidays
```

```
""" Import the datasets:
 - actual_duration.csv data set as ad.
 - appointments_regional.csv data set as ar.
 - national_categories.xlsx data set as nc.
 - tweets.csv data set as tw."""
ad = pd.read_csv('actual_duration.csv')
ar = pd.read_csv('appointments_regional.csv')
nc = pd.read_excel('national_categories.xlsx')
tw = pd.read_csv('tweets.csv')
```

2. Sense-check (performed pre- and post-cleaning, see Appendix 1): check datatypes, (see 'Data cleaning'), metadata, descriptive statistics, etc.

   Note: There are 21,604 'duplicate' values in the ar data set. Recognising that the count_of_appointments reconciles with the other two datasets (see point 5, Appendix 2), consider all the entries valid. No entries were removed.

3. Logical consistency analysis across related fields and datasets:
    a. Confirmed that 'appointment_date' reconciles with 'appointment_month' in nc
    b. Confirmed that 'icb_ons_code' is a subset of 'region_ons_code' in ad
    c. Reconciliation: Confirmed that the 'count_of_appointments' field reconciles across the three datasets (nd, ar and nc). See Appendix 2 for details

4. Data cleaning & transformation
    a. Change data types using df.astype():
        - Cast ad['appointment_date'] to DateTime
        - Cast 'object' fields in all datasets to string for ease of visualisation.
    b. Columns added to improve comparability between the datasets (supports Reconciliation, Appendix 2):
        - ad['appointment_month']
        - nc['region_ons_code']
        - ar['region_ons_code']

```
# Add the 'appointment_month' field to ad, equal to 'appointment_date' in the 'yyyy-%m' format
ad['appointment_month'] = ad.appointment_date.dt.year.astype(str) + '-' + ad.appointment_date.dt.strftime('%m').astype(str)
```

```
# Add region_ons_code to nc and ar applying ad mapping.

# Create the icb_region_pairs_unique DataFrame
icb_region_pairs_unique = ad[['icb_ons_code', 'region_ons_code']].groupby(by=['icb_ons_code', 'region_ons_code']).sum().sort_values('region_ons_code')

# Create a dictionary with 'icb_ons_code':'region_ons_code' key:value pairs
icb_region_mapping = icb_region_pairs_unique.reset_index(level=-1).to_dict()['region_ons_code']

# create the 'region_ons_code' column in the nc and ar DataFrames and populate it applying icb_region_mapping
nc['region_ons_code'] = nc['icb_ons_code'].map(icb_region_mapping)
ar['region_ons_code'] = ar['icb_ons_code'].map(icb_region_mapping)

# sense check the output
print(nc.region_ons_code.unique())
print(ar.region_ons_code.nunique())
print(ad.region_ons_code.nunique())
```

```
['E40000012' 'E40000010' 'E40000011' 'E40000007' 'E40000005' 'E40000006'
 'E40000003']
7
7
```

    c. Columns added to enrich datasets (supports Utilisation analysis)
        - nc['day_of_week'], ad['day_of_week']
        - nc['day_of_week_type'], ad['day_of_week_type'] ('holiday'/'working' by region); for is_holiday function refer to 'User-defined functions' section

```
# Create the 'day_of_week_type' column in the nc and ad DataFrames and populate it applying is_holiday user-defined function
nc['day_of_week_type'] = nc.appointment_date.apply(is_holiday)
ad['day_of_week_type'] = ad.appointment_date.apply(is_holiday)
```

        - nc['region_ons_name'] (for presentation)
        - categorical to numerical (assume normal distribution within groups):
            - ar['time_between_book_and_appointment_n']
            - ad['actual_duration_n']

```python
# Add time_between_book_and_appointment_n column to ar and populate it with numeric values

# Create a dictionary with 'time_between_book_and_appointment':'time_between_book_and_appointment_n' key:value pairs
# value in the key:value pairs is equal to the middle of the range indicated in the keys
# 'More than 28 Days' is mapped as 28, as no details are available (The numeric value is underestimeted).
# 'Unknown / Data Quality' is equal to 'More than 28 Days'.
time_between_book_and_appointment_to_n_mapping = {'1 Day': 1,
                                                  '15  to 21 Days': 18,
                                                  '2 to 7 Days': 4.5,
                                                  '22  to 28 Days': 25,
                                                  '8  to 14 Days': 11,
                                                  'More than 28 Days': 28,
                                                  'Same Day': 0,
                                                  'Unknown / Data Quality': 28}

# Create the 'time_between_book_and_appointment_n' column in the ar DataFrame and populate it applying time_between_book_and_appointment_to_n_mapping
ar['time_between_book_and_appointment_n'] = ar['time_between_book_and_appointment'].map(time_between_book_and_appointment_to_n_mapping)

# sense check the output
print(ar.time_between_book_and_appointment_n.unique())
ar[['time_between_book_and_appointment', 'time_between_book_and_appointment_n', 'count_of_appointments']]\
.groupby(by=['time_between_book_and_appointment', 'time_between_book_and_appointment_n'], dropna=False).sum()
```

```
[ 1.  18.   4.5 25.  11.  28.   0. ]
```

time_between_book_and_appointment	time_between_book_and_appointment_n	count_of_appointments
1 Day	1.0	67716097
15 to 21 Days	18.0	42710574
2 to 7 Days	4.5	153794531
22 to 28 Days	25.0	25536541
8 to 14 Days	11.0	86846519
More than 28 Days	28.0	23050987
Same Day	0.0	342747171
Unknown / Data Quality	28.0	402105

```python
# Add actual_duration_n column to ad and populate it with numeric values

# Create a dictionary with 'actual_duration':'actual_duration_n' key:value pairs
# value in the key:value pairs is equal to the middle of the range indicated in the keys
# 'Unknown / Data Quality' is populated based on a weighted average 'actual_duration_n' of 14 (rounded).
actual_duration_to_n_mapping = {'1-5 Minutes': 3,
                                '11-15 Minutes': 13,
                                '16-20 Minutes': 18,
                                '21-30 Minutes': 25.5,
                                '31-60 Minutes': 45.5,
                                '6-10 Minutes': 8,
                                'Unknown / Data Quality': 14}

# Create the 'actual_duration_n' column in the ad DataFrame and populate it applying actual_duration_to_n_mapping
ad['actual_duration_n'] = ad['actual_duration'].map(actual_duration_to_n_mapping)

# sense check the output
print(ad.actual_duration_n.unique())
ad[['actual_duration', 'actual_duration_n', 'count_of_appointments']].groupby(by=['actual_duration', 'actual_duration_n'], dropna=False).sum()
```

```
[45.5 25.5  8.  14.  18.  13.   3. ]
```

actual_duration	actual_duration_n	count_of_appointments
1-5 Minutes	3.0	28600865
11-15 Minutes	13.0	25160882
16-20 Minutes	18.0	16004247
21-30 Minutes	25.5	15026365
31-60 Minutes	45.5	9103432
6-10 Minutes	8.0	33800815
Unknown / Data Quality	14.0	40284086

5. User-defined functions:
    a. is_holiday() used to populate nc['day_of_week_type'], ad['day_of_week_type'] columns (support Utilisation analysis)

is_holiday(date) function

```
# create a function to check if a date is a holiday/working day in a particular region. Supports Utilisation analysis

"""The is_holiday(date) function takes one argument:
    date: takes a date in the DateTime format
    The function returns a string value of 'holiday' / 'working'."""

def is_holiday(date):
    if date.weekday() >= 5 or date in holidays.CountryHoliday('GB', prov='ENG'):
        return 'holiday'
    else:
        return 'working'
```

    b. weighted_average() to avoid replication of code and errors

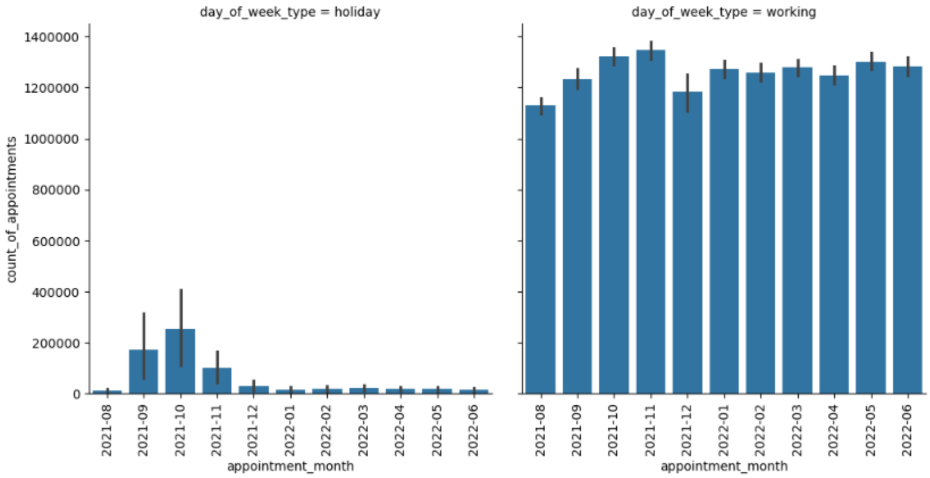
weighted_average(source_df, col_to_weight, dimensions_to_gr_by, wa_col_name, col_to_weight_by = 'count_of_appointments') function

```
""" The weighted_average() function takes a source DataFrame (source_df) and calculates a weighted average
    for a specified column (col_to_weight) grouped by one or more dimensions (dimensions_to_gr_by).
    It uses another column (col_to_weight_by) to weight the values in the col_to_weight column.
    The function returns a new DataFrame with the weighted average column added (wa_col_name).

    Parameters:
    source_df: takes a DataFrame object to transform
    col_to_weight: takes a column name as a string
    dimensions_to_gr_by: takes dimensions to group by as a list of column names
    wa_col_name: takes column name to use as the column name as a str
    col_to_weight_by: default col_to_weight_by='count_of_appointments'

    Output data type: pd.DataFrame"""

def weighted_average(source_df, col_to_weight, dimensions_to_gr_by, wa_col_name, col_to_weight_by = 'count_of_appointments'):
    # Define a variable to save the list of column names for the 'groupby=' parameter in the output
    col_to_gr_by_fin = dimensions_to_gr_by.copy()

    # Form the list of column names for the 'groupby=' parameter in interim calculations and define a variable
    dimensions_to_gr_by.append(col_to_weight)
    col_to_gr_by_int = dimensions_to_gr_by.copy()

    # Form the list of column names to subset in interim calculations and define a variable
    dimensions_to_gr_by.append(col_to_weight_by)
    col_to_subset = dimensions_to_gr_by.copy()

    # Subset 'col_to_subset' and group by 'col_to_gr_by_int' to allow intermediary calculations
    source_df = source_df[col_to_subset].groupby(by=col_to_gr_by_int, dropna=False).sum().reset_index()

    # Add the 'wa_col_name' column and populate it with 'col_to_weight' weighted by 'col_to_weight_by'
    # Group the output by 'col_to_gr_by_fin'
    source_df['tech_sum_time'] = source_df[col_to_weight] * source_df[col_to_weight_by]
    source_df.drop(col_to_weight, axis=1, inplace=True)
    source_df = source_df.groupby(by=col_to_gr_by_fin, dropna=False).sum().reset_index()
    source_df[wa_col_name] = source_df['tech_sum_time'] / source_df[col_to_weight_by]
    source_df.drop('tech_sum_time', axis=1, inplace=True)

    # reset variables
    col_to_subset = []
    col_to_gr_by_int = []
    col_to_gr_by_fin= []

    # Return
    return source_df
```

Analytical approach:

Step 1: Subset and aggregate average appointments per day, to quantify full capacity and actual utilisation, and explore patterns by different dimensions, including:

- average appointment per calendar day monthly: barplot
- average total app per day daily (for a selected month): DataFrame
- split by type of weekday

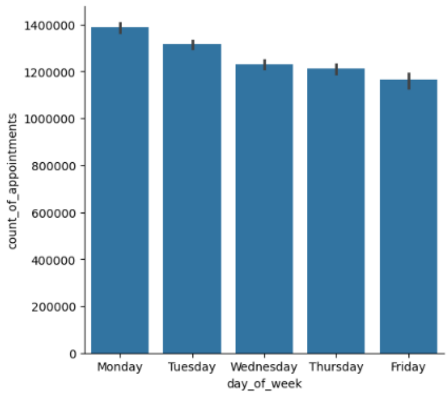


- split by type of weekday and region: catplot(kind='point') and DataFrame

Note: We observe two clusters (working days/holidays) and seasonal trends. Recognising this and the variability in the working/calendar days ratio per month, in utilisation analysis, we focus on working-day daily averages to assess full capacity.

The average daily full capacity of 1,260,000 (rounded to 10,000), which is comparable with the 1,200,000 utilised for planning, is calculated as the daily average number of booked appointments on working days throughout 11 months.

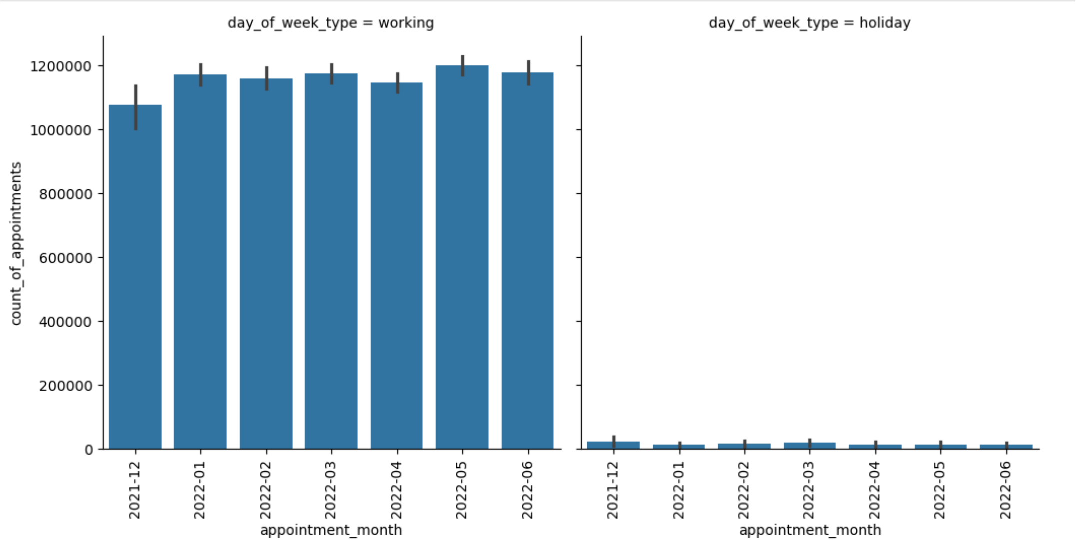
- working days: split by weekday: barplot



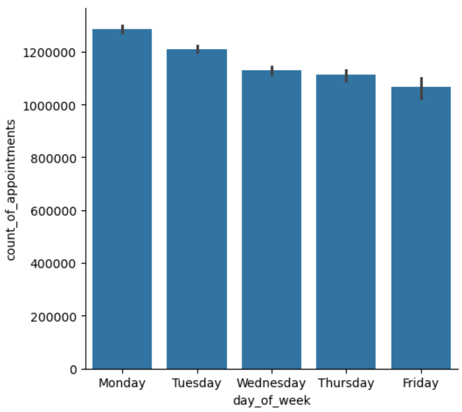
- working days: split by weekday and region: barplot and boxplot

  Note: There is a higher IQR in Dec-21. This is explained by the significantly lower appointment count on 24th and 31st Dec. There are no implications for further analysis as immaterial (Appendix 3). This explains the lower daily average appointments in Dec.

- Average attended appointment per day monthly split by working days and holidays during Dec-12 to June-22: barplot

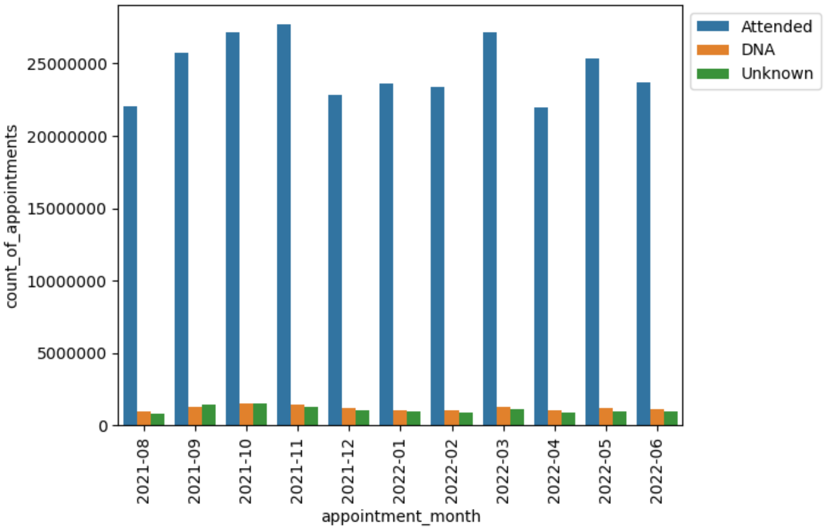


- working days: Attended split by weekday:



Step 2: Subset and aggregate average appointments per day, to identify utilisation patterns, including:

- Total appointments by attendance status monthly



Note: A Significant portion of ‘Unknown’. Higher DNA & Unknown during Sep-Nov.

There are no appointments reported as available. Assume: appointments are fully booked, but not all of them are attended.

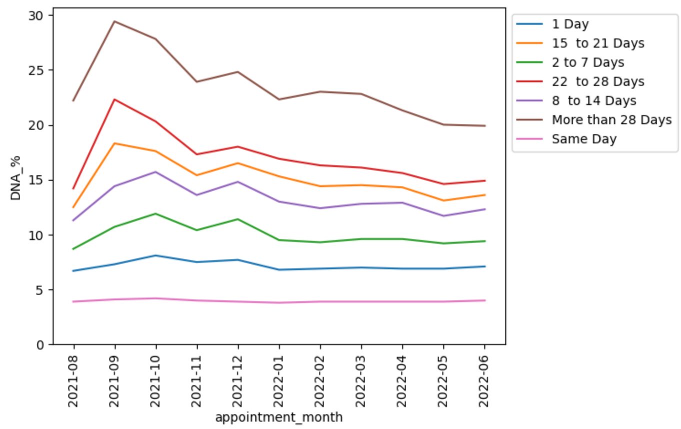
- attendance vs waiting time: DataFrame

appointment_status ween_book_and_appointment	Attended	DNA	Unknown	Total	Attended_row_%	DNA_%_row	Unknown_row_%	Attended_col_%	DNA_col_%	Unknown_col_%
Same Day	328380317	6052604	8314250	342747171	95.8	1.8	2.4	48.5	19.6	24.4
1 Day	62556833	2634536	2524728	67716097	92.4	3.9	3.7	9.2	8.5	7.4
2 to 7 Days	138103022	8697476	6994033	153794531	89.8	5.7	4.5	20.4	28.1	20.5
8 to 14 Days	75092108	6193368	5561043	86846519	86.5	7.1	6.4	11.1	20.0	16.3
15 to 21 Days	35842753	3282752	3585069	42710574	83.9	7.7	8.4	5.3	10.6	10.5
22 to 28 Days	20798309	1999990	2738242	25536541	81.4	7.8	10.7	3.1	6.5	8.0
More than 28 Days	16699531	2036154	4315302	23050987	72.4	8.8	18.7	2.5	6.6	12.6
Unknown / Data Quality	283003	14353	104749	402105	70.4	3.6	26.1	0.0	0.0	0.3
Total	677755876	30911233	34137416	742804525	91.2	4.2	4.6	100.0	100.0	100.0

appointment_status_est time_between_book_and_appointment	Attended_est	DNA_est	Total	Attended_est_row_%	DNA_est_row_%	Attended_est_col_%	DNA_est_col_%
Same Day	328380317	14366854	342747171	95.8	4.2	48.5	22.1
1 Day	62556833	5159264	67716097	92.4	7.6	9.2	7.9
2 to 7 Days	138103022	15691509	153794531	89.8	10.2	20.4	24.1
8 to 14 Days	75092108	11754411	86846519	86.5	13.5	11.1	18.1
15 to 21 Days	35842753	6867821	42710574	83.9	16.1	5.3	10.6
22 to 28 Days	20798309	4738232	25536541	81.4	18.6	3.1	7.3
More than 28 Days	16699531	6351456	23050987	72.4	27.6	2.5	9.8
Unknown / Data Quality	283003	119102	402105	70.4	29.6	0.0	0.2
Total	677755876	65048649	742804525	91.2	8.8	100.0	100.0

Note: Recognising demonstrable similarity in the ‘Unknown’ and ‘DNA’ appointment status patterns, ‘Unknown’ is grouped with ‘DNA’ (‘DNA_est’). Similarly, ‘Unknown / Data Quality’ time between booking and appointment is grouped with ‘More than 28 Days’ (‘time_between_booking_and_appointment_est’).

- Missed app % by month by waiting time sns.lineplot()



Note: DNA_% is higher within each of the categories during the 'peak' period (Sep-Nov) and festive Dec. The longer the waiting period – the higher. Recognising this, grouped to '8+ days' and 'Up to 7 days'.

- Actual daily average attended appointments on working days available for 7 months covered by ar dataset only. To derive an estimate for the another 4 months (out of 11) covered in nc assume that missed appointments rate for working and weekends does not differ materially. Daily average attended appointments are estimated based on avg daily missed appointments rates. Given differences are immaterial, proceed with the daily_avg_att_app_est

Given differences are immaterial, proceed with the daily_avg_att_app_est

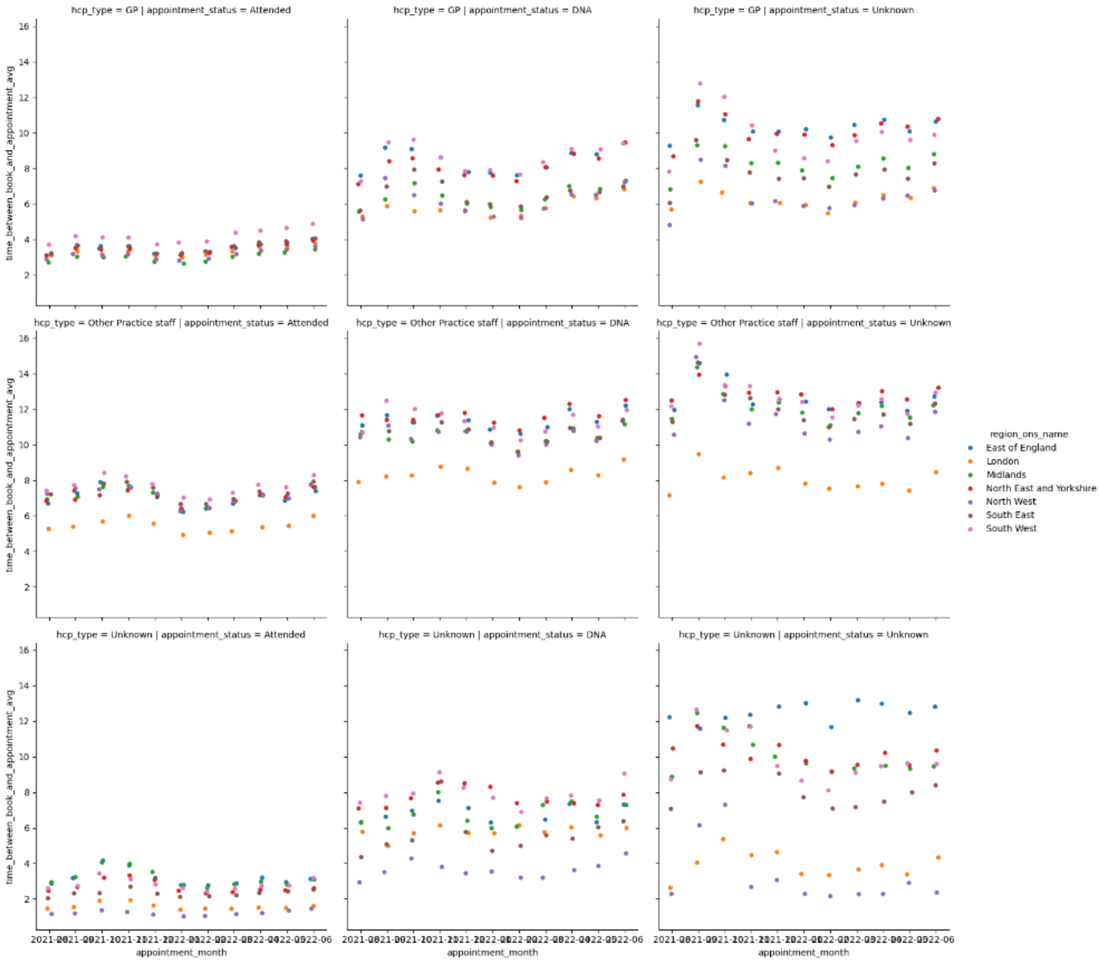
```
# prepare summary DataFrame to estimate daily average attended appointments on working days
summary_df = pd.merge(daily_avg_booked_app_monthly_wd, ar_df_missed_monthly['DNA_est_%'], left_index=True, right_index=True)

summary_df['daily_avg_att_app_est'] = summary_df['daily_avg_booked_app_act'] * (1 - summary_df['DNA_est_%'] / 100)

summary_df = pd.merge(summary_df, daily_avg_booked_app_monthly_wd_act, left_index=True, right_index=True, how='outer')

summary_df['reconciliation_%'] = summary_df['daily_avg_att_app_est'] / summary_df['daily_avg_att_app_act']

summary_df.reset_index(inplace=True)

summary_df
```

	appointment_month	daily_avg_booked_app_act	DNA_est_%	daily_avg_att_app_est	daily_avg_att_app_act	reconciliation_%
0	2021-08	1129529	7.4	1045943.854	NaN	NaN
1	2021-09	1233360	9.7	1113724.080	NaN	NaN
2	2021-10	1321946	10.3	1185785.562	NaN	NaN
3	2021-11	1345518	9.0	1224421.380	NaN	NaN
4	2021-12	1182330	9.1	1074737.970	1077079.0	0.997827
5	2022-01	1272445	8.0	1170649.400	1172308.0	0.998585
6	2022-02	1259643	7.9	1160131.203	1160613.0	0.999585
7	2022-03	1278974	8.2	1174098.132	1174629.0	0.999548
8	2022-04	1247687	8.2	1145376.666	1146101.0	0.999368
9	2022-05	1300806	7.8	1199343.132	1199637.0	0.999755
10	2022-06	1283363	8.2	1178127.234	1178878.0	0.999363

- Average waiting time by 'appointment_status', 'hcp_type' and 'region_ons_name' monthly

```
sns.catplot(data=ar_df_wtime_hcp_reg_monthly, x='appointment_month', y='time_between_book_and_appointment_avg', \
            col='appointment_status', row='hcp_type', hue='region_ons_name', kind='strip')
```

```
<seaborn.axisgrid.FacetGrid at 0x20b4b494bc0>
```



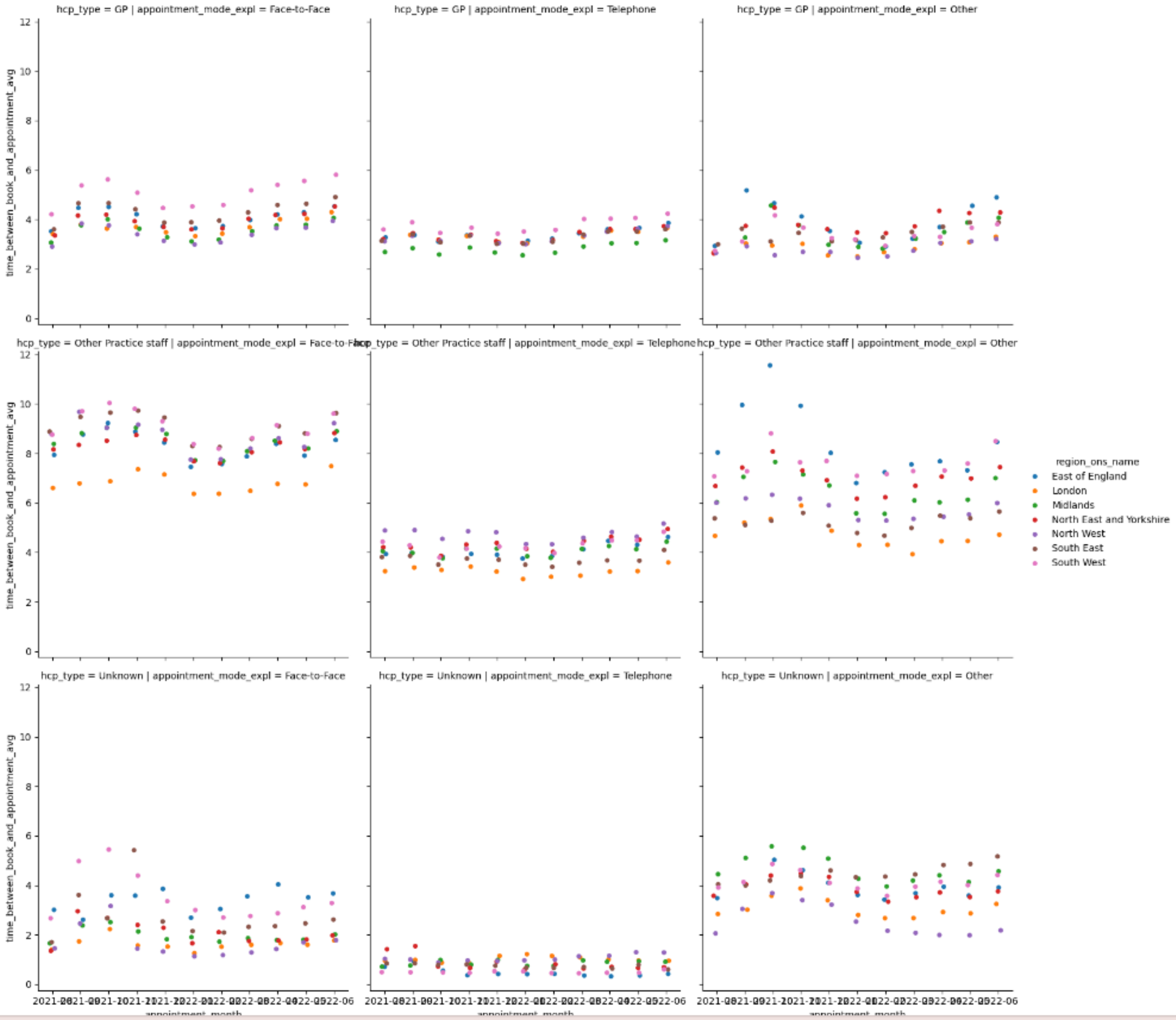
Observation: The average waiting time is lower for appointment status 'Attended'. 'DNA' and 'Unknown' appointment statuses are characterised by longer waiting times. This is fair for all 7 regions. Other Practice staff have higher waiting times compared to the GP.

Other Practice staff in London have noticeably shorter waiting times compared to other regions.

- Average waiting time by 'appointment_status', 'hcp_type' and 'appointment mode' monthly

```
sns.catplot(data=ar_df_att_wtime_hcp_reg_mode_monthly, x='appointment_month', y='time_between_book_and_appointment_avg', \
            col='appointment_mode_expl', col_order=['Face-to-Face', 'Telephone', 'Other'], row='hcp_type', hue='region_ons_name', kind='strip')
```

```
<seaborn.axisgrid.FacetGrid at 0x20b3f81ab10>
```

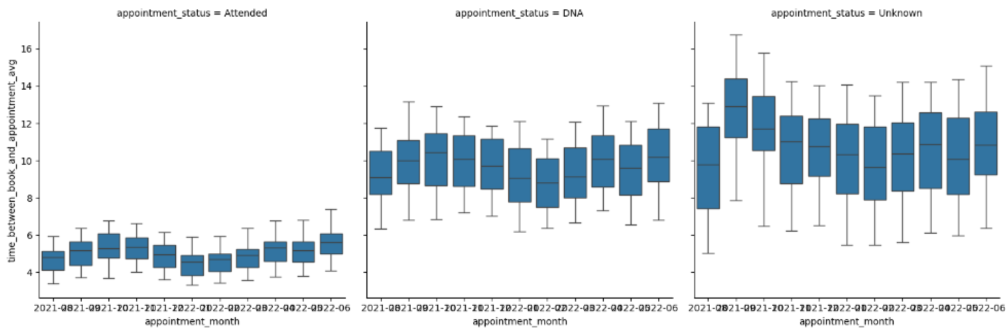


Observation: Higher share of Face-to-Face for Other staff compared to GP. Lower waiting times for GP compared to Other across all categories.

- Per ICB

```
sns.catplot(data=ar_df_icb, x='appointment_month', y='time_between_book_and_appointment_avg', col='appointment_status', kind='box')
```

```
<seaborn.axisgrid.FacetGrid at 0x20b43978c80>
```



- Tweets from Twitter with hashtags related to healthcare in the UK

Alternative approach (tags_alt)

```
[262]: # Create a variable (tags_alt), and assign an empty list to it.
tags_alt = []

# Define the regex pattern to match hashtags
pattern = r"#\w+"
```

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

```
[264]: # sense-check the output
print(type(tags_alt))
print(len(tags_alt))
print(tags_alt)
```

```
<class 'list'>
4378
['#healthcare', '#premisehealth', '#hiring', '#healthcare', '#new', '#blogs', '#digitaltransformation', '#cybersecurity', '#accounting', '#finance',
'#healthcare', '#firstcoastcna', '#cnaexam', '#cnaexampreparation', '#jacksonville', '#cnatraining', '#nurse', '#nursing', '#nurselife', '#nursepracti
tioner', '#nurseproblems', '#nursingschool', '#healthcare', '#new', '#disparities', '#healthcare', '#alert', '#insurance', '#data', '#healthcare', '#h
ealthcare', '#healthcare', '#healthcare', '#hcldr', '#premisehealth', '#hiring', '#premisehealth', '#hiring', '#healthcare', '#qualitypatientcare', '#
jobs', '#job', '#ascp2022', '#ascp100', '#healthcare', '#healthsecretary', '#healthcare', '#ai', '#sdoh', '#healthcare', '#tropicana', '#real', '#juic
e', '#healthcare', '#watch', '#worms', '#fruits', '#healthtips', '#tips', '#healthcare', '#thewoodlands', '#healthcare', '#chicago', '#healthcare', '#
telehealth', '#healthcare', '#virtualcare', '#mediqueststaffing', '#hospital', '#shift', '#newportbeach', '#job', '#mediqueststaffing', '#hospital',
'#shift', '#newportbeach', '#job', '#jobs', '#employment', '#healthcare', '#job', '#hiring', '#antiracist', '#healthcare', '#newhealthcare', '#healthc
are', '#hospitals', '#physicians', '#healthcare', '#newhealthcare', '#healthcare', '#cloud', '#ehr', '#datasecurity', '#options', '#optionstrading',
'#pfizer', '#optionsflow', '#stocks', '#stock', '#stockmarket', '#investing', '#investment', '#healthcare', '#healthcare', '#womenleaders', '#healthca
re', '#albuquerque', '#healthcare', '#healthcare', '#telehealth', '#telemedicine', '#healthcare', '#therapy', '#primarycare', '#home', '#hit', '#covid
19', '#usa', '#selfcare', '#coronavirus', '#photooftheday', '#healthcare', '#health', '#internalmedicine', '#tipsfornewdocs', '#meded', '#medtwitter',
'#medicine', '#medical', '#medica', '#healthcare', '#covid', '#stra', '#strategy', '#competitiveintellige', '#strate', '#strategy', '#competi', '#stra
tegy', '#competitiveintelligence', '#marketing', '#healthcare', '#biotech', '#pharma', '#competitivemarketing', '#pharmaceutical', '#strategy', '#comp
etitiveintelligence', '#marketing', '#pharmaceutical', '#competitivemarketing', '#biotech', '#healthcare', '#pharma', '#internalmedicine', '#tipsforne
```

There are 844 tweets with #healthcare hashtag and 42 with #medicine.

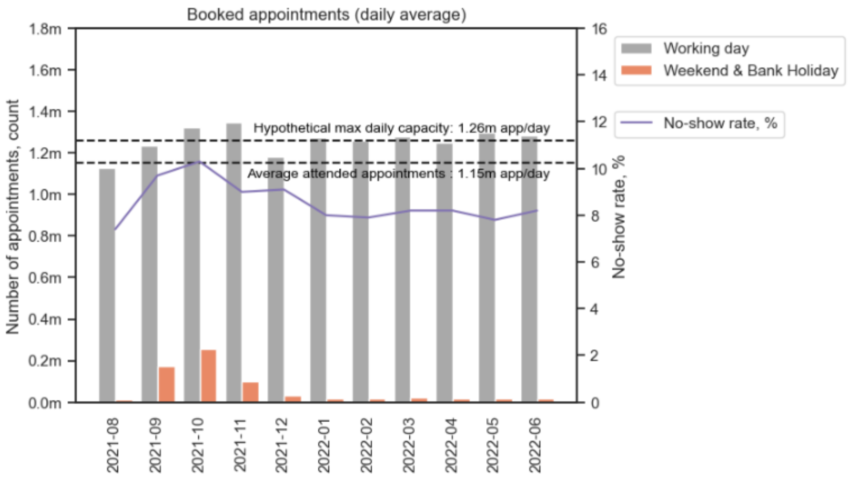
3. Visualization approach

The intended target audience is the NHS management, including CEO and CFO, as well as functional heads. Visualisations are designed to support conclusions. The choice of visualisation types is driven by the key messages they are intended to convey, i.e.:

- Lineplots are used to display trends over time
- Barplots – to compare categories (‘working day’/’holiday’)
- Scatterplots - to visualise clusters (hcp type, appointment mode)

For an example of exploratory visualisations, refer to Appendix 3. Exploratory characterised by higher granularity.

Selected explanatory visualisations:



Please refer to Appendix 4 for details of the code used to prepare the visualisations.

4. Insights patterns and recommendations

Our analysis indicated the following:

- The average daily number of appointments is higher on working days compared to holidays.
- The high season for booked appointments is from September to November, with a higher average number of appointments both on holidays and weekdays.
- The same seasonal pattern is observed for attended appointments, but it is smoother due to the patterns in missed appointments.
- December has a lower average daily appointments count with a higher interquartile range (IQR).
- There is a pattern in the number of average appointments per day through weekdays, with the workload decreasing from Monday to Friday.
- The high peak period is characterized by higher no show rates and waiting times.
- The higher the waiting time, the lower the attendance rates.
- The highest average achieved attendance rate per waiting time group is 95.8% for "Same day".
- Other Practice staff has a higher waiting time and no-show rates compared to GP.
- Waiting time and no-show rates are lower for Telephone appointments compared to Face-to-Face
- Daily average booked appointments in August-21 June-22 are 1.26m.
- The average daily attended appointments per working day over the 11-month period was 1.15m, which implies c.91% utilization in terms of booked appointments on average (compared to 1.26m).

Recommendations given to business users:

- Use appointment reminders to actively manage customer attendance and reduce missed appointments.
- Focus on appointments that have a waiting time of more than 8 days, appointments with other healthcare professionals, face-to-face GP appointments and appointments in regions other than London to optimize no-show rates.
- Take measures to slow down or reverse the trend of decreasing the share of appointments in telephone mode. This can be achieved through increasing client awareness about the Telephone option and, potentially, making the Telephone option default for services where it is suitable.
- Implement ‘smart scheduling’ by scheduling non-urgent appointments for Thursdays and Fridays to smooth workload seasonality on a weekly horizon.
- Implement leave policies that stimulate healthcare professionals to take leave during the low season (December - July) to manage resource availability.

Areas for further analysis:

- Conduct an analysis to identify categories of appointments that can be rescheduled from the high season (September - November) to the low season on an annual horizon to reduce waiting times and working hours on weekends in the high season, resulting in cost savings on compensation for working on holidays.

Appendix 1: Sense-check DataFrames (selected)

Pre-cleaning & transformation

```
# Determine the metadata (e.g. df.info()) of each DataFrame.
print(ad.info())
print(ar.info())
print(nc.info())
print(tw.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 137793 entries, 0 to 137792
Data columns (total 8 columns):
 #   Column                     Non-Null Count   Dtype
---  ------                     --------------   -----
 0   sub_icb_location_code      137793 non-null  object
 1   sub_icb_location_ons_code  137793 non-null  object
 2   sub_icb_location_name      137793 non-null  object
 3   icb_ons_code               137793 non-null  object
 4   region_ons_code            137793 non-null  object
 5   appointment_date           137793 non-null  object
 6   actual_duration            137793 non-null  object
 7   count_of_appointments      137793 non-null  int64
dtypes: int64(1), object(7)
memory usage: 8.4+ MB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 596821 entries, 0 to 596820
Data columns (total 7 columns):
 #   Column                             Non-Null Count   Dtype
---  ------                             --------------   -----
 0   icb_ons_code                       596821 non-null  object
 1   appointment_month                  596821 non-null  object
 2   appointment_status                 596821 non-null  object
 3   hcp_type                           596821 non-null  object
 4   appointment_mode                   596821 non-null  object
 5   time_between_book_and_appointment  596821 non-null  object
 6   count_of_appointments              596821 non-null  int64
dtypes: int64(1), object(6)
memory usage: 31.9+ MB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 817394 entries, 0 to 817393
Data columns (total 8 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   appointment_date       817394 non-null  datetime64[ns]
 1   icb_ons_code           817394 non-null  object
 2   sub_icb_location_name  817394 non-null  object
 3   service_setting        817394 non-null  object
 4   context_type           817394 non-null  object
 5   national_category      817394 non-null  object
 6   count_of_appointments  817394 non-null  int64
 7   appointment_month      817394 non-null  object
dtypes: datetime64[ns](1), int64(1), object(6)
memory usage: 49.9+ MB
None
```

Post-cleaning & transformation

```
# Determine the metadata (e.g. df.info()) of each DataFrame.
print(ad.info())
print(ar.info())
print(nc.info())
print(tw.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 137793 entries, 0 to 137792
Data columns (total 13 columns):
 #   Column                     Non-Null Count   Dtype
---  ------                     --------------   -----
 0   sub_icb_location_code      137793 non-null  string
 1   sub_icb_location_ons_code  137793 non-null  string
 2   sub_icb_location_name      137793 non-null  string
 3   icb_ons_code               137793 non-null  string
 4   region_ons_code            137793 non-null  string
 5   appointment_date           137793 non-null  datetime64[ns]
 6   actual_duration            137793 non-null  string
 7   count_of_appointments      137793 non-null  int64
 8   appointment_month          137793 non-null  string
 9   day_of_week                137793 non-null  string
 10  actual_duration_n          137793 non-null  int64
 11  region_ons_name            137793 non-null  string
 12  day_of_week_type           137793 non-null  string
dtypes: datetime64[ns](1), int64(2), string(10)
memory usage: 13.7 MB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 596821 entries, 0 to 596820
Data columns (total 11 columns):
 #   Column                               Non-Null Count   Dtype
---  ------                               --------------   -----
 0   icb_ons_code                         596821 non-null  string
 1   appointment_month                    596821 non-null  string
 2   appointment_status                   596821 non-null  string
 3   hcp_type                             596821 non-null  string
 4   appointment_mode                     596821 non-null  string
 5   time_between_book_and_appointment    596821 non-null  string
 6   count_of_appointments                596821 non-null  int64
 7   region_ons_code                      596821 non-null  string
 8   time_between_book_and_appointment_n  596821 non-null  int64
 9   region_ons_name                      596821 non-null  string
 10  appointment_mode_expl                596821 non-null  string
dtypes: int64(2), string(9)
memory usage: 50.1 MB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 817394 entries, 0 to 817393
Data columns (total 13 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   appointment_date        817394 non-null  datetime64[ns]
 1   icb_ons_code            817394 non-null  string
 2   sub_icb_location_name   817394 non-null  string
 3   service_setting         817394 non-null  string
 4   context_type            817394 non-null  string
 5   national_category       817394 non-null  string
 6   count_of_appointments   817394 non-null  int64
 7   appointment_month       817394 non-null  string
 8   region_ons_code         817394 non-null  string
 9   day_of_week             817394 non-null  string
 10  region_ons_name         817394 non-null  string
 11  day_of_week_type        817394 non-null  string
 12  national_category_expl  817394 non-null  string
dtypes: datetime64[ns](1), int64(1), string(11)
memory usage: 81.1 MB
None
```

Appendix 2: The 'count_of_appointments' field reconciliation across datasets

Reconciliation output. For more details refer to the ‘The 'count_of_appointments' field reconciliation across the three datasets (nd, ar and nc)’ section in Jupiter Notebook

```
# Calculate the sum of the differences
sum_rec[['Check_1','Check_2']].sum()
```

```
Check_1    0.0
Check_2    0.0
dtype: float64
```

```
# View the full dataframe
sum_rec
```

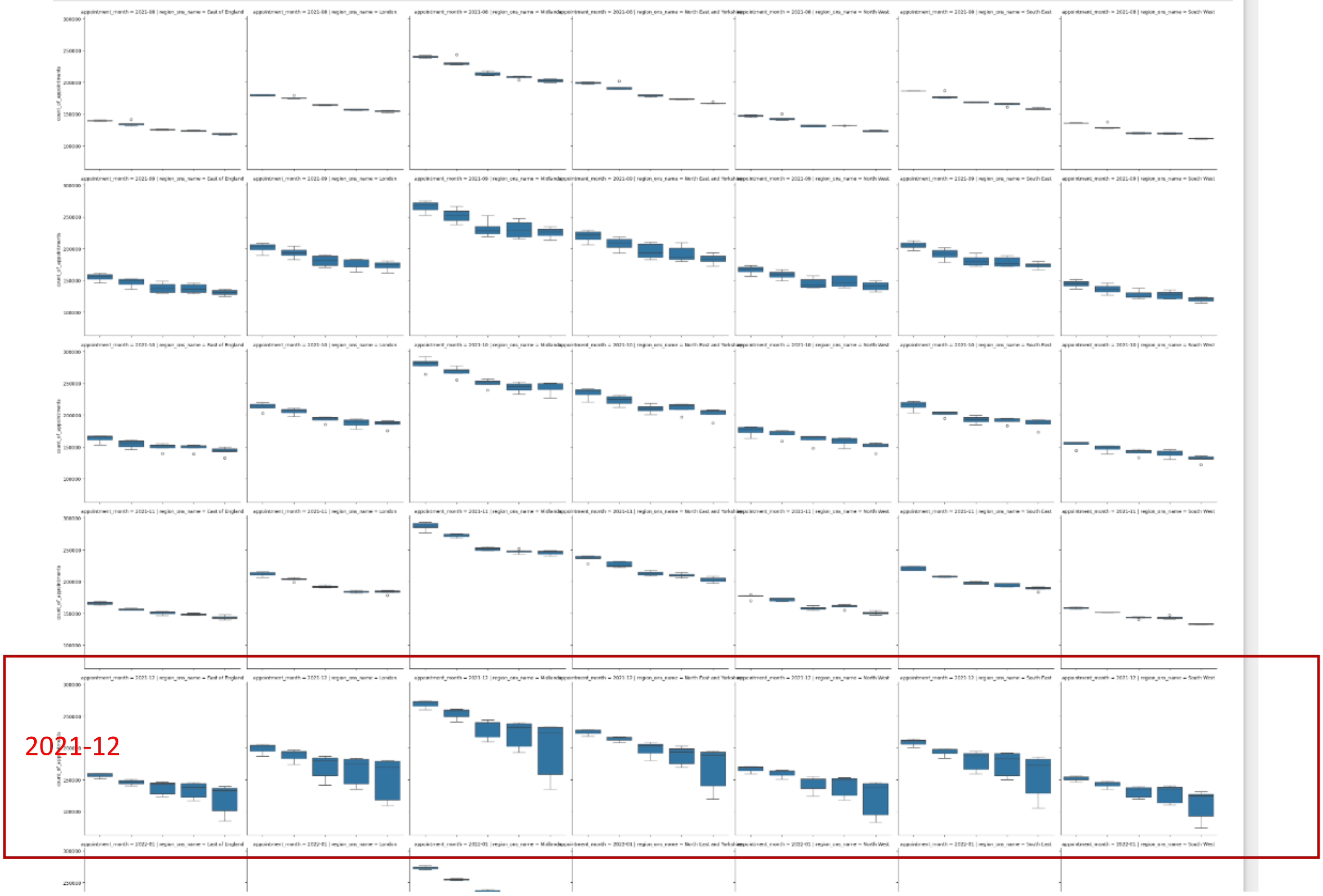
appointment_month	nc_s	ar_att_s	ar_dna_s	ar_unk_s	ar_s	ad_s	Check_1	Check_2
2020-01	NaN	24538291	1298269	1362736	27199296	NaN	NaN	NaN
2020-02	NaN	21640067	1215154	1249400	24104621	NaN	NaN	NaN
2020-03	NaN	20718865	1166314	2168289	24053468	NaN	NaN	NaN
2020-04	NaN	13982824	478766	1546291	16007881	NaN	NaN	NaN
2020-05	NaN	14962850	449057	1005305	16417212	NaN	NaN	NaN
2020-06	NaN	18943022	594382	1153401	20690805	NaN	NaN	NaN
2020-07	NaN	20606888	698327	1186222	22491437	NaN	NaN	NaN
2020-08	NaN	18438932	669462	1042126	20150520	NaN	NaN	NaN
2020-09	NaN	23988492	1145971	1579792	26714255	NaN	NaN	NaN
2020-10	NaN	25529275	1358138	1414519	28301932	NaN	NaN	NaN
2020-11	NaN	23072059	1021408	968135	25061602	NaN	NaN	NaN
2020-12	NaN	21592221	996416	947299	23535936	NaN	NaN	NaN
2021-01	NaN	20645718	899833	946518	22492069	NaN	NaN	NaN
2021-02	NaN	20736205	804659	858705	22399569	NaN	NaN	NaN
2021-03	NaN	25289991	964719	970714	27225424	NaN	NaN	NaN
2021-04	NaN	22056413	893723	929796	23879932	NaN	NaN	NaN
2021-05	NaN	21779605	878219	850571	23508395	NaN	NaN	NaN
2021-06	NaN	24815463	1030367	938352	26784182	NaN	NaN	NaN
2021-07	NaN	23761001	1029665	948553	25739219	NaN	NaN	NaN
2021-08	23852171.0	22081765	949137	821269	23852171	NaN	NaN	0.0
2021-09	28522501.0	25757066	1321348	1444087	28522501	NaN	NaN	0.0
2021-10	30303834.0	27170506	1565624	1567704	30303834	NaN	NaN	0.0
2021-11	30405070.0	27667665	1428087	1309318	30405070	NaN	NaN	0.0
2021-12	25140776.0	22853483	1198866	1088427	25140776	22853483.0	0.0	0.0
2022-01	25635474.0	23597196	1076013	962265	25635474	23597196.0	0.0	0.0
2022-02	25355260.0	23351939	1076658	926663	25355260	23351939.0	0.0	0.0
2022-03	29595038.0	27170002	1289888	1135148	29595038	27170002.0	0.0	0.0
2022-04	23913060.0	21948814	1045455	918791	23913060	21948814.0	0.0	0.0
2022-05	27495508.0	25343941	1199518	952049	27495508	25343941.0	0.0	0.0
2022-06	25828078.0	23715317	1167790	944971	25828078	23715317.0	0.0	0.0

Appendix 3: Average appointment per working day by weekday: split by region and appointment month

```
nc_dwt_reg_daily[(nc_dwt_reg_daily.appointment_month == '2021-12') \
& (nc_dwt_reg_daily.region_ons_name == 'Midlands') \
& (nc_dwt_reg_daily.day_of_week_type == 'working')].sort_values(by='count_of_appointments')
```

	appointment_date	appointment_month	day_of_week	day_of_week_type	region_ons_name	count_of_appointments
1017	2021-12-24	2021-12	Friday	working	Midlands	134893
1066	2021-12-31	2021-12	Friday	working	Midlands	158684
1059	2021-12-30	2021-12	Thursday	working	Midlands	193063
1010	2021-12-23	2021-12	Thursday	working	Midlands	202267
1052	2021-12-29	2021-12	Wednesday	working	Midlands	209771

```
sns.catplot(data=nc_dwt_reg_daily[nc_dwt_reg_daily.day_of_week_type == 'working'], x='day_of_week', y='count_of_appointments', \
            order=['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday'], \
            col='region_ons_name', row='appointment_month', estimator='mean', kind='box')

# change the y-axis format to non-scientific
plt.ticklabel_format(style='plain', axis='y')
```



Appendix 4: Explanatory visualisation (summary)

```
# create a plot with a secondary y-axis
fig, ax1 = plt.subplots()

palette = ['#A9A9A9', 'coral']

# plot booked appointments
g = sns.barplot(data=nc_dwt_daily, x='appointment_month', y='count_of_appointments', \
           hue='day_of_week_type', hue_order=['working', 'holiday'], estimator='mean', errorbar=None, palette=palette)

# Add annotation line: avg daily booked appointments per working day during 11 months
plt.axhline(y=summary_df['daily_avg_booked_app_act'].mean(), color='k', linestyle='--')
plt.text(6.8, summary_df['daily_avg_booked_app_act'].mean() + 55000, \
        f"Hypothetical max daily capacity: {round(summary_df['daily_avg_booked_app_act'].mean()/1000000, 2)}m app/day", \
        fontsize=10, color='black', ha='center', va='center')

# Add annotation line: avg daily attended appointments per working day during 11 months (estimate)
plt.axhline(y=summary_df['daily_avg_att_app_est'].mean(), color='k', linestyle='--')
plt.text(6.7, summary_df['daily_avg_att_app_est'].mean() - 55000, \
        f"Average attended appointments : {round(summary_df['daily_avg_att_app_est'].mean()/1000000, 2)}m app/day", \
        fontsize=10, color='black', ha='center', va='center')

# formatting
plt.title('Booked appointments (daily average)')

# x axis formatting
plt.xlabel('')
plt.xticks(rotation=90)

# ax1 formatting
ax1.set_ylabel('Number of appointments, count')
ax1.set_ylim([0, 1800000])
ax1.set_yticks(g.get_yticks())
ylabels = ['{:,.1f}'.format(x) + 'm' for x in g.get_yticks()/1000000]
g.set_yticklabels(ylabels)

ax2 = ax1.twinx()

# plot missed app ratio
sns.lineplot(data=summary_df, x='appointment_month', y='DNA_est_%', color='m')

# ax2 formatting
ax2.set_ylabel('No-show rate, %')
ax2.set_ylim([0, 16])

# add legend
ax1.legend(['Working day', 'Weekend & Bank Holiday'], bbox_to_anchor=(1.55,1))
ax2.legend(['No-show rate, %'], bbox_to_anchor=(1.43,0.8))

plt.show()
```