Assignment: **Diagnostic Analysis using Python. NHS Case Study**
Jurgita Cepure
LSE_DA201_Data Analytics Using Python_P4_2024

Scope. In light of a growing population this study is aimed at supporting the planning and budgeting for future healthcare infrastructure within England's - not the entire UK's - publicly funded system. Two business questions have been posed:

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

Appointment is the unit of measure in the following analysis and three data sets of varying structure, quality and time frame were provided.

Approach. Given the absence of direct data on appointment availability and staff numbers, the general approach involved analysing key metrics individually and then comparing them across time and locations. The analysis aimed to address the following questions:

- Were there significant differences within appointment status categories?
- Were there seasonal variations in appointments, and how were they distributed across locations?
- Who carried out those appointments?
- What modes of consultations were used?
- What was the wait time for appointments?
- What did appointment duration show?
- At a national level, did service setting, context type and national categories reflect trends the appointment metrics?

Prepping datasets. The function was run for all three data frames:

```
def validate_data(df):
    """
    This function validates the data and checks for data types, unique values, missing values
    duplicates, summary stats etc.
    """
    #check for basic data frame info and data types
    print("\n DataFrame info:")
    dataframe_info = df.info()

    #check for unique values
    unique_counts = df.nunique()
    print("\nUnique values per column:")
    print(unique_counts)

    #check for missing values
    missing_values_count = df.isna().sum()
    print("\nNumber of missing values:")
    print(missing_values_count)

    #checking for duplicates
    duplicate_count = df.duplicated().sum()
    print("\nNumber of duplicate rows:")
    print(duplicate_count)

    #summary stats
    summary_stats = df.describe()
    print("\nSummary statistics:")
    print(summary_stats)

    #return unique_counts, duplicate_count, summary_stats
```

There were no missing values in any of data frames, naming conventions consistent, however count_of_appointments was renamed respectively:

```
# Renames 'count_of_appointments' in each DataFrame.
ad = ad.rename(columns={'count_of_appointments': 'ad_count_of_appointments'})

ar = ar.rename(columns={'count_of_appointments': 'ar_count_of_appointments'})

nc = nc.rename(columns={'count_of_appointments': 'nc_count_of_appointments'})
```

The view to keep duplicates returned only in 'ar' out of three data sets was taken due to data frame not being granular enough as well as not having primary key, hence it was impossible to determine whether any of it were true duplicates.

In 'ad' cleaned up 'appointment_date' column and added appointment_month

```
# Change dtype for column 'appointment_date' in ad DataFrame from object to datetime.
# 1. Check for invalid date values with check_invalid_dates function first
check_invalid_dates(ad,'appointment_date')
```

```
Empty DataFrame
Columns: [sub_icb_location_code, sub_icb_location_ons_code, sub_icb_location_name, icb_
ons_code, region_ons_code, appointment_date, actual_duration, ad_count_of_appointments,
appointment_date_issues]
Index: []
```

```
[239]:
```

```
# 2. Given the above confirmation, change 'appointment_date' dtype using function conve
convert_to_datetime(ad,'appointment_date')
```

```
datetime64[ns]
```

```
[240]:
```

```
# Create a new column 'appointment_month' in ad Dataframe. Format YYYY-MM, dtype - stri
ad['appointment_month'] = ad['appointment_date'].dt.strftime('%Y-%m')
```

And in 'ar' added 'appointment_month_datetype' to be in line with other data frames too.

```
# Create a new column 'appointment_month_datetime' in ar Dataframe and change dtype to
ar['appointment_month_datetime'] = ar['appointment_month']

# Convert dtype of new column to datetime using function
convert_to_datetime(ar, 'appointment_month_datetime')
```

```
datetime64[ns]
```

Location-wise meaningful information was possible to extract from Integrated Care Boards (ICBs) level.

```
# Determine the number of locations and NHS England hierarchy
distinct_region_count = ad['region_ons_code'].nunique()
print("Number of Distinct NHS England Regions:", distinct_region_count)

distinct_icb_count = ad['icb_ons_code'].nunique()
print("Number of Distinct NHS ICB locations:", distinct_icb_count)

distinct_sub_icb_count = ad['sub_icb_location_ons_code'].nunique()
print("Number of Distinct NHS sub-ICB locations:", distinct_sub_icb_count)
```

```
Number of Distinct NHS England Regions: 7
Number of Distinct NHS ICB locations: 42
Number of Distinct NHS sub-ICB locations: 106
```

Majority of metrics were analysed within entire time frame of dataset 'ar' covering period of 30months from 2020-01 to 2022-06 despite it included Covid data. However, time adjusted data frame filtered_ar for more granular analysis was created too:

```
# Filtered data set to only look at data from 2021-08 onwards, also preserves the original 'ar' dataframe
# Assumed to be more reliable post-Covid data that can be compaired against nc dataframe too.
filtered_ar = ar.co›››py()
filtered_ar = ar[ar['appointment_month']>='2021-08']
filtered_ar.head()
```

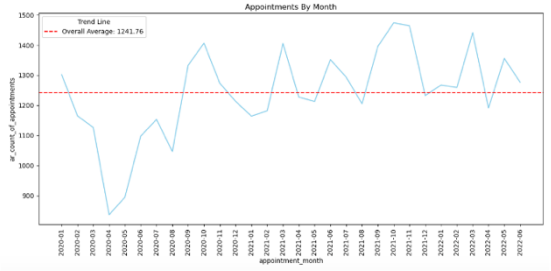
	icb_ons_code	appointment_month	appointment_status	hcp_type	appointment_mode	time_between_book_and_appointment	ar_count_of_appointments
3652	E54000034	2021-08	Attended	GP	Face-to-Face	1 Day	6553
3653	E54000034	2021-08	Attended	GP	Face-to-Face	15 to 21 Days	2390
3654	E54000034	2021-08	Attended	GP	Face-to-Face	2 to 7 Days	10547

The Process.

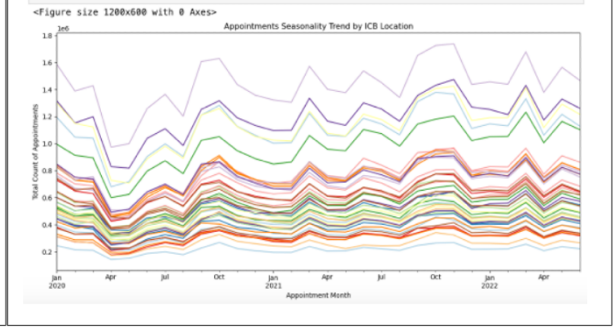
Appointment status, seasonality and spread across locations

Presumably due to size of population within each ICB monthly appointments vary (*Appendix 1*), however seasonal trends appear to be repeated across the entire network.

```
# Set the figure size
plt.figure(figsize=(12, 6))

# Create a line plot for the count of appointments by month
sns.lineplot(data=ar, x='appointment_month', y='ar_count_of_appointments',
             color='skyblue', errorbar=None)

# Calculate the average monthly appointments by calling function.
average_appointments = average_monthly_appointments(ar, 'appointment_month',
                                                    'ar_count_of_appointments')

# Calculate the overall average (mean) of the monthly averages
overall_average = average_appointments.mean()

# Plot the overall average line
plt.axhline(y=overall_average, color='red', linestyle='--',
            label=f'Overall Average: {overall_average:.2f}')

# Set plot title
plt.title("Appointments By Month")

# Rotate x-axis labels
plt.xticks(rotation=90)

# Add a legend to the plot
plt.legend(title='Trend Line')

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




```
# Group by both 'appointment_month_datetime' and 'icb_ons_code'
# and calculate the sum of 'ar_count_of_appointments'
appointments_by_location_and_month = ar.groupby(
    ['appointment_month_datetime','icb_ons_code'])['ar_count_of_appointments'].sum().reset_index()

# Pivot the table to get 'icb_ons_code' as columns and 'appointment_month_datetime' as rows
appointments_pivot = appointments_by_location_and_month.pivot(index='appointment_month_datetime',
                                                              columns='icb_ons_code',
                                                              values='ar_count_of_appointments')
# Plot the chart
plt.figure(figsize=(12, 6))

appointments_pivot.plot(kind='line', figsize=(12, 6), colormap='Paired', legend=False)

# Add labels and title
plt.xlabel('Appointment Month')
plt.ylabel('Total Count of Appointments')
plt.title('Appointments Seasonality Trend by ICB Location')

# Rotate x-axis labels
plt.xticks(rotation=0)

plt.tight_layout()
plt.show()
```



Around 95% of appointments get attended no matter the season and across all locations suggesting that if patients book the appointment, they turn up for it. Given Did not Attend (DNA) category of appointments is less than 5%, further analysis was not adjusted to exclude it and was carried out under the assumption, if appointment was counted, it was attended.

```
# Plot for Average Monthly Appointments by attendance for Each ICB location
# Group by 'icb_ons_code' and 'appointment_status' to get total appointments for
# each status within each icb_ons_code.
appointments_by_status_and_icb = ar.groupby(
    ['icb_ons_code', 'appointment_status'])['ar_count_of_appointments'].sum().unstack(fill_value=0)

# Calculate the average monthly appointments for each icb_ons_code by
# summing the appointments and dividing by the number of months.
average_monthly_appointments_icb = appointments_by_status_and_icb.mean(axis=0)

# Normalize the data by dividing each icb_ons_code's appointment counts by
# the total appointments in that icb_ons_code.
normalized_appointments_icb = appointments_by_status_and_icb.div(
    appointments_by_status_and_icb.sum(axis=1), axis=0) * average_monthly_appointments_icb.sum()

# Plot the stacked bar chart
fig, ax = plt.subplots(figsize=(12, 6))

# Create a stacked bar plot for each icb_ons_code
normalized_appointments_icb.plot(kind='bar', stacked=True, ax=ax, colormap='Paired')

# Add labels and title
ax.set_xlabel('ICB ONS Code')
ax.set_ylabel('Average Monthly Appointments')
ax.set_title('Average Monthly Appointments Attendance for Each ICB location over 30 months')

# Move the legend to the top right
ax.legend(title='Appointment Status', loc='upper right', bbox_to_anchor=(1.2, 1))

# Show the plot with rotated x-axis labels
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



```
# Average Monthly Appointments by Appointment Status.
# Group by 'appointment_month' and 'appointment_status' to get total appointments
# for each status within each month
appointments_by_status_and_month = ar.groupby(
    ['appointment_month', 'appointment_status'])['ar_count_of_appointments'].sum().unstack(fill_value=

# Calculate the average monthly appointments by
#summing the total appointments per month and dividing by the number of months
average_monthly_appointments = appointments_by_status_and_month.mean(axis=0)

# Normalize the data by dividing each month's appointment counts by
# the total appointments in that month
normalized_appointments = appointments_by_status_and_month.div(
    appointments_by_status_and_month.sum(axis=1), axis=0) * average_monthly_appointments.sum()

# Plot the stacked bar chart
fig, ax = plt.subplots(figsize=(12, 6))

# Create a stacked bar plot for each appointment_months
normalized_appointments.plot(kind='bar', stacked=True, ax=ax, colormap='Paired')

# Add labels and title
ax.set_xlabel('Appointment Month')
ax.set_ylabel('Average Monthly Appointments')
ax.set_title('Average Monthly Appointments Attendance over 30 months')

# Move the legend to the top right
ax.legend(title='Appointment Status', loc='upper right', bbox_to_anchor=(1.2, 1))

# Show the plot with rotated x-axis labels
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```


Healthcare Professional Type

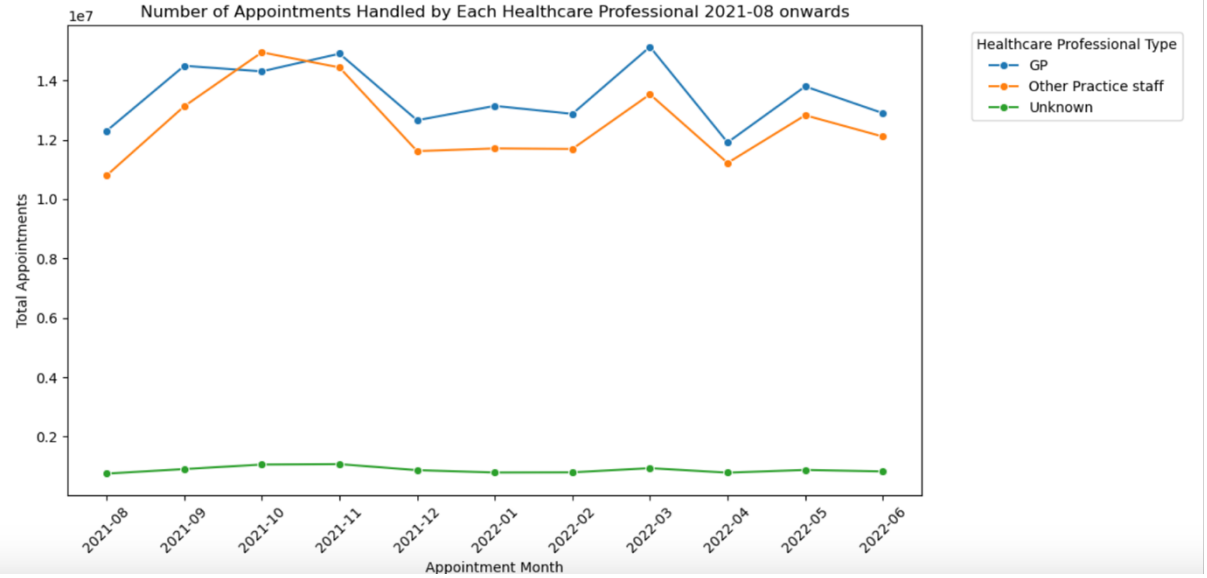
GPs and Other Practice staff handle similar quantity of appointments on average with GPs usually attending to slightly more of it across network.

```
# The average monthly appointments split by 'hcp_type' in 'ar' DataFrame.

# Group by 'hcp_type' and calculate the sum of appointments for each status.
appointments_by_hcp_type = ar.groupby('hcp_type')['ar_count_of_appointments'].sum().reset_index()

# Calculate average monthly appointments per 'hcp_type'
appointments_by_hcp_type_monthly = appointments_by_hcp_type['ar_count_of_appointments'] / ar['appointment_month'].nunique()
appointments_by_hcp_type['average_monthly_appointments'] = appointments_by_hcp_type_monthly

print("\nAverage Monthly Appointments by Health Professional:")
for _, row in appointments_by_hcp_type.iterrows():
    print(f"{row['hcp_type']}: {row['average_monthly_appointments']:,.0f}")

# Calculate the proportion of each 'hcp_type' to the total average monthly appointments
hcp_type_proportions = appointments_by_hcp_type_monthly / overall_average_of_appointments_ar
appointments_by_hcp_type['proportion'] = hcp_type_proportions

print("\nProportions of Average Monthly Appointments by Health Professional (%):")
for _, row in appointments_by_hcp_type.iterrows():
    print(f"{row['hcp_type']}: {row['proportion'] * 100:,.0f}%")
```

```
Average Monthly Appointments by Health Professional:
GP: 12,655,005
Other Practice staff: 11,321,684
Unknown: 783,462

Proportions of Average Monthly Appointments by Health Professional (%):
GP: 48%
Other Practice staff: 43%
Unknown: 3%
```

However, time adjusted plot shows a different trend for Sep-Oct'21peak of appointments where Other Practice staff handles more consultations than GPs. This trend is observed in autumn 2020 too.

```
# Group by month and 'hcp_type' and calculate the sum of 'ar_count_of_appointments'
appt_by_hcp_monthly = filtered_ar.groupby(['appointment_month','hcp_type'])['ar_count_of_appointments'].sum().reset_index()

# Create the lineplot
plt.figure(figsize=(12, 6))
sns.lineplot(x='appointment_month', y='ar_count_of_appointments', hue='hcp_type', data=appt_by_hcp_monthly, marker='o')

# Set the title and labels
plt.title('Number of Appointments Handled by Each Healthcare Professional 2021-08 onwards')
plt.xlabel('Appointment Month')
plt.ylabel('Total Appointments')
plt.xticks(rotation=45)

# Move the legend outside of the plot
plt.legend(title='Healthcare Professional Type', bbox_to_anchor=(1.05, 1), loc='upper left')

# Display the plot
plt.tight_layout()
plt.show()
```



Appointment Mode

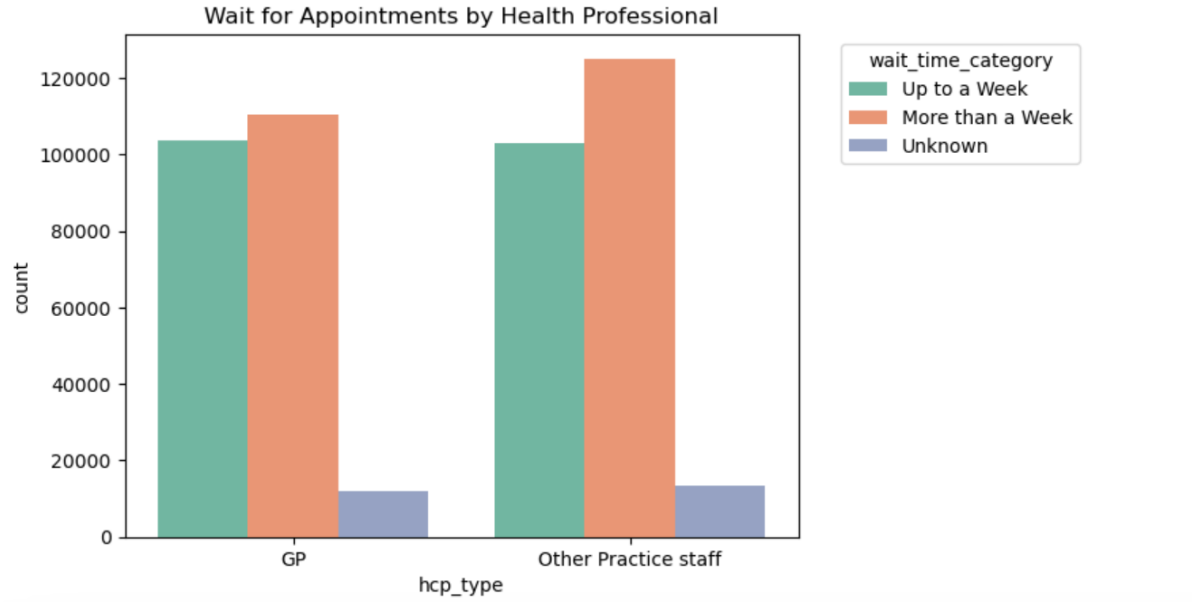
On average vast majority of appointments are Face-to-Face or by phone and are handled in similar quantities either by GP or Other Practice staff. Appendix 2

Wait Time for Appointment

The dataset shows that, on average, 46% of appointments are same-day, a characteristic of the UK booking system where appointments for the day can only be made on the same day. Additionally, 9% are for the next day, 21% within a week, and 12% within 8-14 days. Appendix 3

GPs and Other Practice Staff handle a similar number of appointments for same-day, next-day, and same-week bookings. However, when patients had to wait longer for an appointment, it was more likely to be carried out by Other Practice Staff.

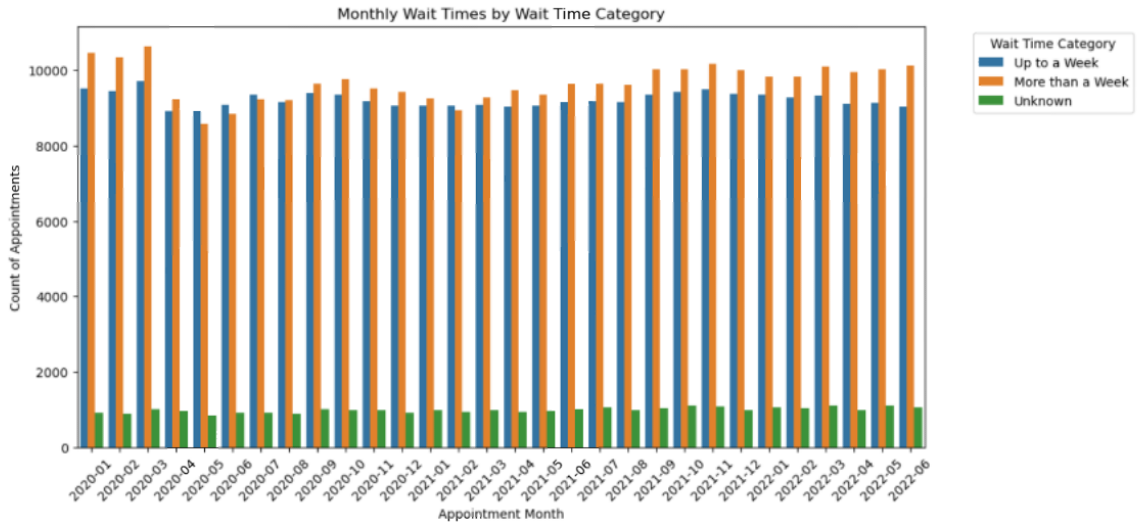
```
# Create countplot to show split between wait time up_to_a_week and more_than_a_week by hcp
#(excl. hcp_type Unknown category = 3% of all).
sns.countplot(x='hcp_type', hue= 'wait_time_category',
              data=ar[ar['hcp_type'].isin(['GP', 'Other Practice staff'])],
              palette='Set2', order=['GP', 'Other Practice staff'])

# Move the legend outside of the plot
plt.legend(title='wait_time_category', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.title('Wait for Appointments by Health Professional')


# Adjust layout to make room for the legend
#plt.tight_layout()

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



Also, Appendix 4.

Except for the early months of the Covid pandemic and January 2021 (a lockdown month), appointments with longer wait times have consistently outpaced those with shorter waits, with their proportion gradually increasing. This may indicate an overall shortage of staff, or a shortage for specialist appointments.

```
# Create plot to check whether some months have more of longer wait time then others.
# Group values for 'wait_time_up_to_a_week' and 'wait_time_more_than_a_week'
wait_time_up_to_a_week = ar[ar['time_between_book_and_appointment'].isin(['Same Day', '1 Day', '2 to 7 Days'])]
wait_time_more_than_a_week = ar[ar['time_between_book_and_appointment'].isin(['8  to 14 Days',
                                                                              '15  to 21 Days', '22  to 28 Days', 'More than 28 Days'])]

# Create a temporary column to distinguish between the two categories
ar['wait_time_category'] = ar['time_between_book_and_appointment'].apply(
    lambda x: 'Up to a Week' if x in ['Same Day', '1 Day', '2 to 7 Days']
    else ('More than a Week' if x in ['8  to 14 Days', '15  to 21 Days', '22  to 28 Days', 'More than 28 Days'] else 'Unknown')
)

# Create the count plot
plt.figure(figsize=(12, 6))
sns.countplot(data=ar, x='appointment_month', hue='wait_time_category')

# Move the legend outside of the plot
plt.legend(title='Wait Time Category', bbox_to_anchor=(1.05, 1), loc='upper left')

# Display the plot
plt.xticks(rotation=45)
plt.title('Monthly Wait Times by Wait Time Category')
plt.xlabel('Appointment Month')
plt.ylabel('Count of Appointments')
plt.show()
```

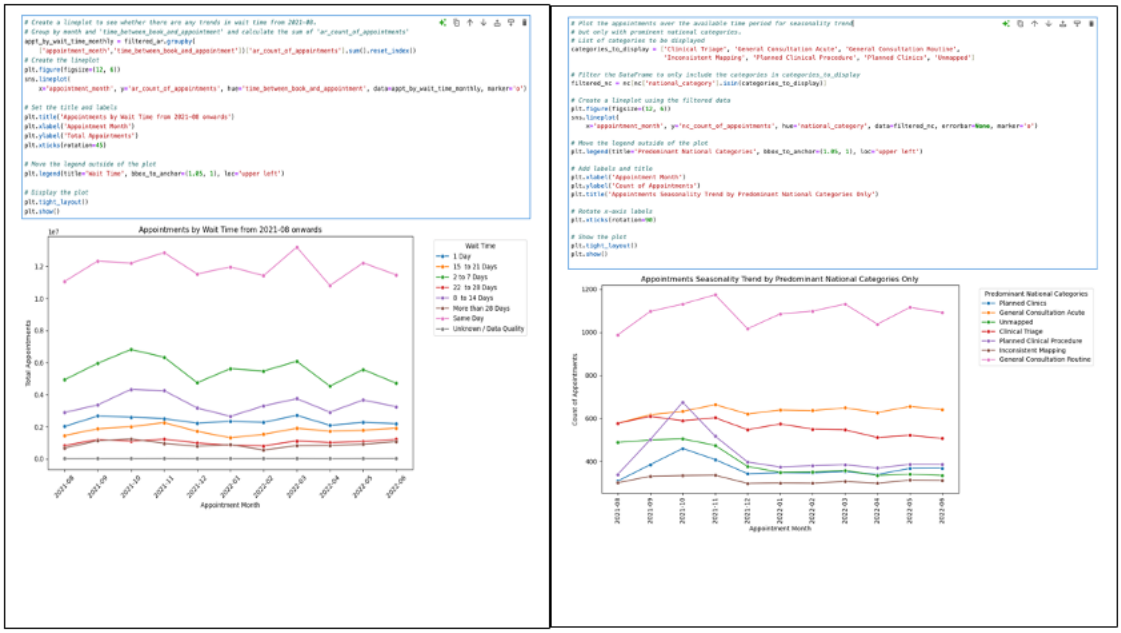


Patients on average wait longer for their appointment across majority of ICBs Appendix 5.

National Categories and Service Setting

Additional insights can be drawn by comparing the adjusted period plot of wait time categories with the corresponding plot for predominant national categories, where during the autumn peak, same-day and next-day appointments decrease, while appointments with wait times of 2-7, 8-14, and 15-21 days increase, likely due to resources being redirected elsewhere i.e. Planned Clinical Procedures and Planned Clinics, which likely explains the overall surge in appointments across all ICBs during this period. Appointments with shorter wait times return to pre-peak levels by December,

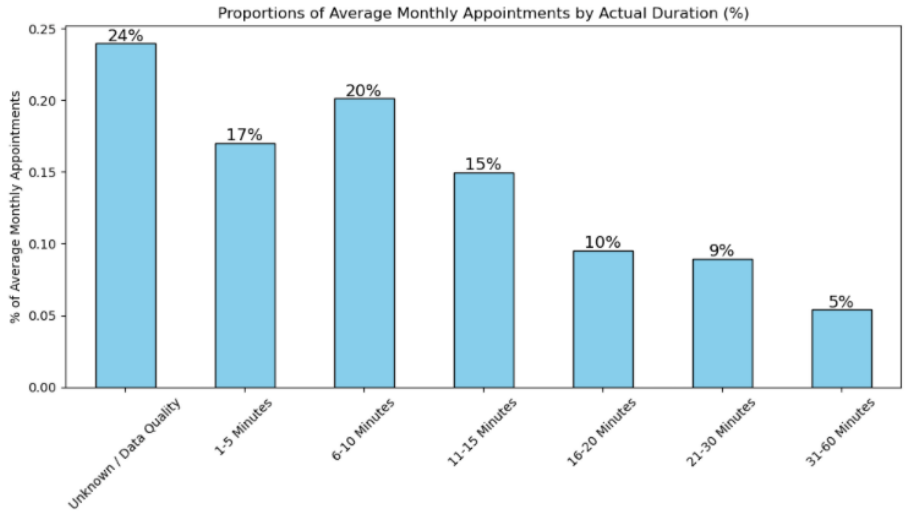
while those with longer wait times recover by January. The quicker recovery of shorter wait time appointments may suggest a shortage of "Other Staff" rather than GPs.
March peak seems to be replicated across all wait time categories in uniform trend.



The predominant category (around 90%) within the Service Setting is General Practice, and seasonal trends typically follow monthly appointment peaks. However, once General Practice is excluded, there is a noticeable spike in the ‘Unmapped’ category. ‘Unknown’ HCP type shows similar pattern during the same peak periods too. This suggests that ‘Unmapped’ category and ‘Unknown’ HCP staff may be helping to handle the increase in Planned Clinics and Procedures, while also supporting the overall level of service across other national appointment categories within the network.

Actual Duration metric can become valuable for assessing staffing needs, as specialists may require more time with patients than others. By mapping these appointments to detailed HCP types, it'd be possible to identify potential staff shortfalls. Additionally, cross-examining this data with appointment modes could reveal whether shorter appointments are predominantly telephone-based, and if not, consider shifting them to currently underutilised online mode Appendix 2. However, in the current setting, this metric is limited due to its novelty and data quality issues.

```python
# The list of categories in the desired order
duration_order = ['Unknown / Data Quality', '1-5 Minutes', '6-10 Minutes', '11-15 Minutes',
                  '16-20 Minutes', '21-30 Minutes', '31-60 Minutes']

# Group by 'actual_duration' and calculate the sum of appointments for each duration
appointments_by_actual_duration = ad.groupby('actual_duration')['ad_count_of_appointments'].sum().reset_index()

# Calculate average monthly appointments per 'actual_duration'
appointments_by_actual_duration['average_monthly_appointments'] = (
    appointments_by_actual_duration['ad_count_of_appointments'] /
    ad['appointment_month'].nunique()
)

# Calculate overall average monthly appointments (sum across all actual durations)
overall_average_of_appointments_ad = appointments_by_actual_duration['average_monthly_appointments'].sum()

# Proportion of each 'actual_duration' to the total average monthly appointments
appointments_by_actual_duration['proportion'] = (
    appointments_by_actual_duration['average_monthly_appointments'] / overall_average_of_appointments_ad
)

# Ensure the DataFrame is sorted by 'actual_duration' in the specified order
appointments_by_actual_duration['actual_duration'] = pd.Categorical(
    appointments_by_actual_duration['actual_duration'],
    categories=duration_order,
    ordered=True
)

# Sort the DataFrame by the 'actual_duration' column according to the specified order
appointments_by_actual_duration = appointments_by_actual_duration.sort_values('actual_duration')

# Plot the bar chart with the ordered 'actual_duration'
fig, ax = plt.subplots(figsize=(10, 6))

appointments_by_actual_duration.plot(kind='bar', x='actual_duration', y='proportion', ax=ax,
                                     color='skyblue', edgecolor='black', legend=False)

# Add the percentage labels to each bar
for i, value in enumerate(appointments_by_actual_duration['proportion']):
    ax.text(i, value - 0.001, f'{value * 100:,.0f}%', ha='center', va='bottom', fontsize=13)

# Adding labels and title
ax.set_ylabel('% of Average Monthly Appointments')
ax.set_title('Proportions of Average Monthly Appointments by Actual Duration (%)')

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

# Adjust layout to avoid overlap
plt.tight_layout()

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



Conclusions.

The guideline of 1,200,000 appointments per day as the maximum capacity for the NHS network shows an average utilization of 70-85% across the entire network, as indicated by the plots in Appendix 6. However, this capacity utilization may vary significantly across different regions or units, influenced by factors such as demand, infrastructure, staff availability, and data quality.

Data quality remains a notable issue, affecting the accuracy of capacity assessments. Additionally, there are three peak appointment periods: autumn, driven by planned clinical procedures, and March and June, observed in general trends.

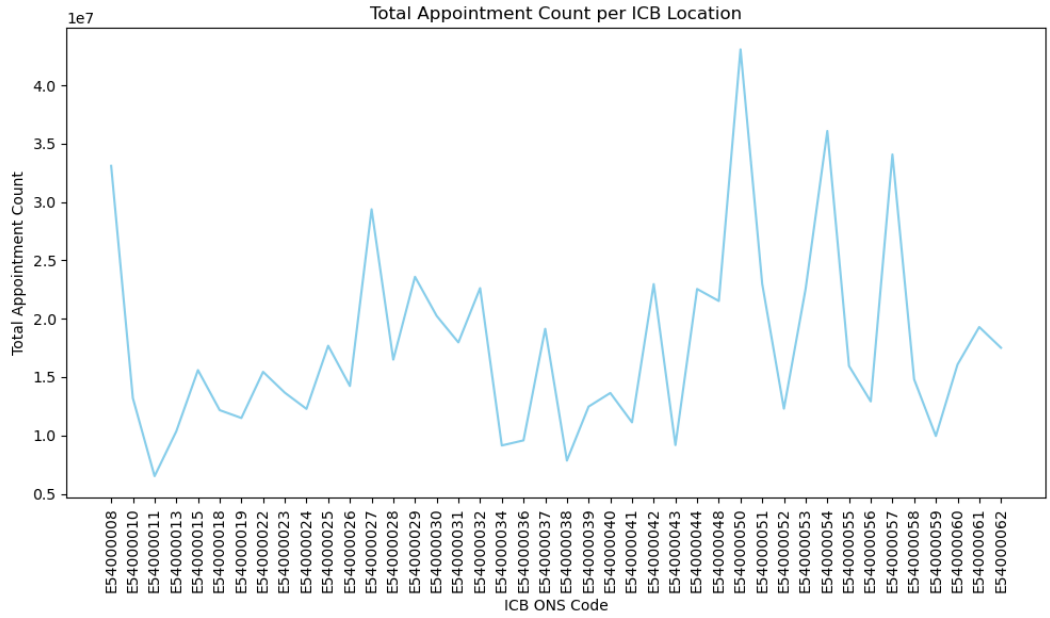
While there is evidence of a shortage of staff, particularly within the broad category of Other Practice Staff, more granular analysis at specific locations is necessary to identify localized staff shortages and address service needs.

Appointment attendance rates remain over 90%, indicating that when patients book appointments, they typically attend. The real challenge lies in the availability of appointments, particularly at specific locations or during certain periods, rather than ensuring patient attendance.

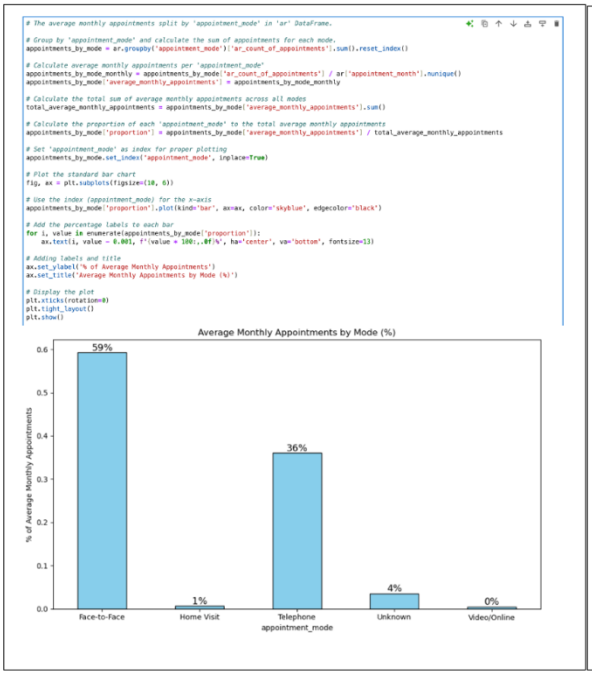
Appendixes

Appendix 1

```
# Monthly appointments in ar Dataframe per ICB
# Group by 'icb_ons_code' and sum the 'ar_count_of_appointments'
appointments_per_icb = ar.groupby('icb_ons_code')['ar_count_of_appointments'].sum().reset_index()

# Create the bar plot
plt.figure(figsize=(10, 6))
sns.lineplot(x='icb_ons_code', y='ar_count_of_appointments', data=appointments_per_icb, color='skyblue

# Set the title and labels
plt.title('Total Appointment Count per ICB Location')
plt.xlabel('ICB ONS Code')
plt.ylabel('Total Appointment Count')

# Rotate x-axis labels
plt.xticks(rotation=90)

# Display the plot
plt.tight_layout()
plt.show()
```



Appendix 2



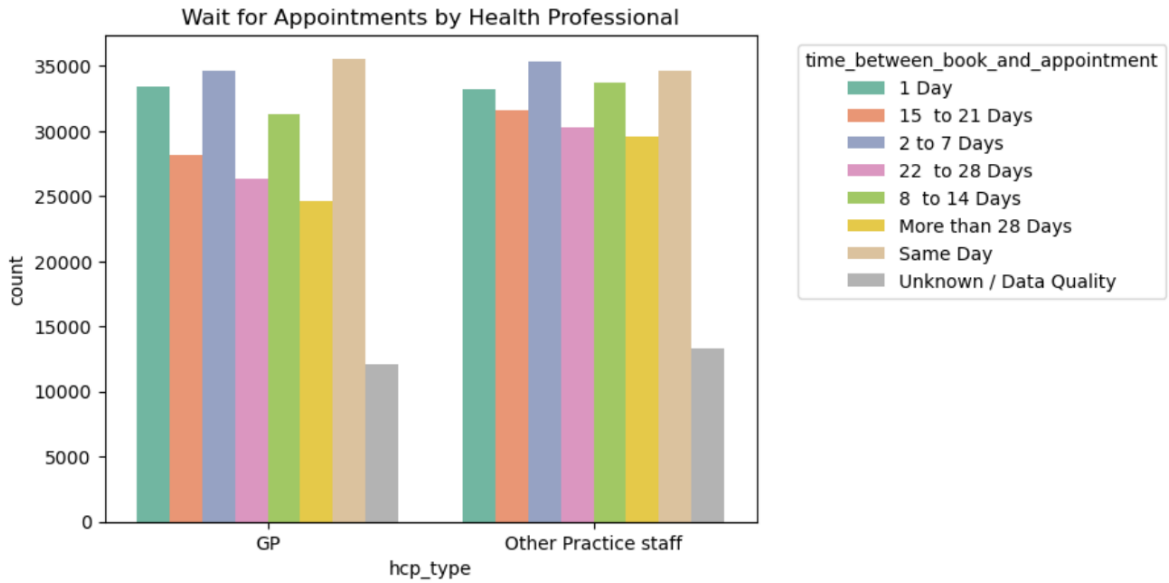
Appendix 3

```python
# The average monthly appointments split by 'time_between_book_and_appointment' in 'ar' DataFrame.

# Group by 'time_between_book_and_appointment' and calculate the sum of appointments for each category.
appointments_by_wait = ar.groupby('time_between_book_and_appointment')['ar_count_of_appointments'].sum().reset_index()

# Calculate average monthly appointments per 'time_between_book_and_appointment'
appointments_by_wait['monthly_avg'] = appointments_by_wait['ar_count_of_appointments'] / ar['appointment_month'].nunique()
print("\nAverage Monthly Appointments by Wait Time:")
for status, value in zip(appointments_by_wait['time_between_book_and_appointment'], appointments_by_wait['monthly_avg']):
    print(f"{status}: {value:,.0f}")

# Calculate the total sum of average monthly appointments across all categories
total_average_monthly_appointments = appointments_by_wait['monthly_avg'].sum()

# Calculate the proportion of each 'time_between_book_and_appointment' to the total average monthly appointments
wait_proportions = appointments_by_wait['monthly_avg'] / total_average_monthly_appointments

# Print the proportions as percentages
print("\nProportions of Average Monthly Appointments by Wait Time (%):")
for wait, value in zip(appointments_by_wait['time_between_book_and_appointment'], wait_proportions):
    print(f"{wait}: {value * 100:,.0f}%")
```

```
Average Monthly Appointments by Wait Time:
1 Day: 2,257,203
15  to 21 Days: 1,423,686
2 to 7 Days: 5,126,484
22  to 28 Days: 851,218
8  to 14 Days: 2,894,884
More than 28 Days: 768,366
Same Day: 11,424,906
Unknown / Data Quality: 13,404

Proportions of Average Monthly Appointments by Wait Time (%):
1 Day: 9%
15  to 21 Days: 6%
2 to 7 Days: 21%
22  to 28 Days: 3%
8  to 14 Days: 12%
More than 28 Days: 3%
Same Day: 46%
Unknown / Data Quality: 0%
```

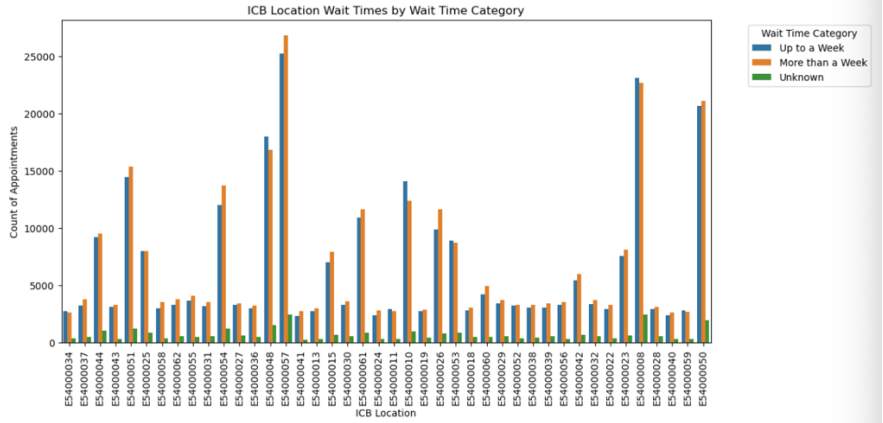
Appendix 4

```
# Create countplot to show how long to wait for the apointment by hcp with only relevant hcp
sns.countplot(x='hcp_type', hue='time_between_book_and_appointment',
              data=ar[ar['hcp_type'].isin(['GP', 'Other Practice staff'])],
              palette='Set2', order=['GP', 'Other Practice staff'])

# Move the legend outside of the plot
plt.legend(title='time_between_book_and_appointment', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.title('Wait for Appointments by Health Professional')


# Adjust layout to make room for the legend
#plt.tight_layout()

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



Appendix 5

```
# Create plot to check what's the split between wait times up_to_a_week and more_than_a_week per ICB locations.
# Create the count plot
plt.figure(figsize=(12, 6))
sns.countplot(data=ar, x='icb_ons_code', hue='wait_time_category')

# Move the legend outside of the plot
plt.legend(title='Wait Time Category', bbox_to_anchor=(1.05, 1), loc='upper left')

# Display the plot
plt.xticks(rotation=90)
plt.title('ICB Location Wait Times by Wait Time Category')
plt.xlabel('ICB Location')
plt.ylabel('Count of Appointments')
plt.show()
```



Appendix 6

```
# Create an aggregated data set to review
#the different features.
ar_agg = filtered_ar[['icb_ons_code',
                      'appointment_month',
                      'appointment_status',
                      'hcp_type',
                      'appointment_mode',
                      'time_between_book_and_appointment',
                      'ar_count_of_appointments']]
# View the DataFrame.
#ar_agg.tail()

# Determine the total number of appointments per month.
ar_df = ar_agg.groupby(
    'appointment_month')['ar_count_of_appointments'].sum().reset_index()
#ar_df.head()

# Add a new column to indicate the average utilisation of services.
# Monthly aggregate / 30 to get to a average daily value.
ar_df_u = ar_df.copy()
ar_df_u['avg_daily_appt'] = ar_df_u['ar_count_of_appointments'] / 30

# Round the 'utilisation' column values
ar_df_u['avg_daily_appt'] = ar_df_u['avg_daily_appt'].round(1)
# View the updated DataFrame
#ar_df_u

# Plot sum of count of monthly visits.
# Change 'appointment_month' to string in  ar_df
ar_df['appointment_month'] = ar_df['appointment_month'].astype(str)

# Given capacity information 1,200,000 appointments per day for 30 days in a month
monthly_capacity = 1200000 * 30

# Calculate capacity utilisation for each month
ar_df_u['capacity_utilisation''] = (
    ar_df_u['ar_count_of_appointments'] / monthly_capacity) * 100

# Plot the total monthly appointments
plt.figure(figsize=(10, 6))
sns.lineplot(
    x='appointment_month',
    y='ar_count_of_appointments',
    data=ar_df_u,
    marker='o',
    color='b'
)
plt.title('Total Monthly Appointments')
plt.xlabel('Appointment Month')
plt.ylabel('Total Appointments')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Plot the monthly capacity utilisation
plt.figure(figsize=(10, 6))
sns.lineplot(
    x='appointment_month',
    y='capacity_utilisation',
    data=ar_df_u,
    marker='o',
    color='g'
)
plt.title('Monthly Capacity Utilisation')
plt.xlabel('Appointment Month')
plt.ylabel('Capacity Utilisation (%)')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```