Analysing GP Appointment Capacity and Utilization in NHS England

Technical Report

By Chris Buck

Background

The National Health Service (NHS) in England commissioned this data analytics project to better understand GP appointment capacity, utilization, and the factors that contribute to missed appointments. It also wants to know whether sources of unstructured data such as posts on X (formerly Twitter) yield insights. Using a fishbone structured thinking approach, the project supports this objective by revealing trends in appointment capacity and attendance to diagnose the circumstances that lead patients to miss appointments. It also makes specific recommendations for increasing capacity and utilization of GP appointments as well as using external data sources.

Analytical approach

Importing Data

- Included user defined functions for:
  - Viewing data: print the shape, head, and data types
  - Validating data: check unique values and duplicates; print summary statistics

- Used box plots used to check for outliers

- Kept duplicates in 'ar' DataFrame because it contains data stripped of more granular detail so duplicates are likely rows that had different dates and/or sub-icb locations before those columns were removed

- Kept the outliers in the NHS datasets because appointment counts were compiled for a wide range of contexts (home visits vs. face-to-face appointments), making it difficult to treat the rows as comparable to each other

- Imported 'gp_icb' dataset without a header and created names for relevant columns

- Kept outliers in the registered patients ('pat') DataFrame because these counts vary widely across GP practices and I aggregated this data to the ICB level

- Kept duplicates in GP workforce ('gp') DataFrame because they were less than 1% of the row count.

- Kept outliers in ‘gp’ DataFrame because it is possible for a GP to be 2.0 FTE if they work 75 hours or more per week[1]

Cleaning Data

- Renamed ‘icb_name’ columns for merging
- Formatted ‘gp’ and ‘pat’ column names in lowercase
- Converted relevant columns to category type and set order for unique values
- Converted relevant columns to datetime type
- Created new column in ‘nc’ DataFrame for names of days to determine distribution of appointments across weekdays

Analysing Data

- Two key metrics drove the analysis
  - Capacity: appointments per working day (1.2 million)
  - Utilization: percentage of attended appointments (calculated by dividing attended appointments by total appointments including those with an unknown status)
- Calculated monthly appointments and attended appointments per working day for January 2020-June 2022
- Asked Claude AI Chatbot to draft a function to calculate appointment attendance percentage for a range of variables in the ‘ar’ DataFrame: appointment mode, hcp type, ICB, and time between booking and appointment
- Calculated patient:clinical staff ratio for each ICB in June 2022 using external data
- Filtered, grouped, and aggregated the regional deprivation dataset ‘imd’ in preparation for merging

[1] Bostock, N. (2016) GPs question 'double counting' in full-time equivalent workforce data. Available at: https://www.gponline.com/gps-question-double-counting-full-time-equivalent-workforce-data/article/1419280 (Accessed: 24 February 2025).

- Merged ‘ar’ DataFrame with external data (deprivation, patient:clinical staff ratio) by ICB

- Created a function to count and print search results for the ‘tweets’ DataFrame

Visualisation and insights

Visualisations

- Line plots for time series data of (attended) appointments and registered patients to answer the questions regarding capacity and utilization

- Bar plots for categorical data and attended appointment percentages to answer question regarding factors driving missed appointments

- Scatter plots for relationships between continuous measurements and attended appointment percentages to answer question regarding factors driving missed appointments

- Annotated plots with lines indicating maximum appointment capacity or average attendance percentage across the data

- Used EngFormatter function in matplotlib to ensure that axis labels for large values (millions) displayed properly

Insights

- In the latter part of the date range appointments per working day were at or above maximum capacity but attended appointments per working day were at or below maximum capacity

- Appointment attendance is lower in ICBs with high deprivation levels

- Appointment attendance is lower in ICBs with high patient to clinical staff ratios

- Appointment attendance decreases as time between booking and appointment increases

- Appointments are clustered at the beginning of the week, but research suggests patients are more likely to attend appointments later in the week[2]

- The keyword search of the tweet texts yielded no insights, but other sources of unstructured data such as the rate and review pages of GP practices could indicate places where appointment capacity is not sufficient for patient demand[3]

Patterns, predictions, and recommendations

- The number of registered patients increases month-to-month and will continue to do so as the UK population increases. This in turn will increase demand for GP appointments and highlights the need to increase capacity (number of appointments per working day) by increasing clinical staff levels.

- The impact of additional clinical staff can be enhanced by placing them in ICBs with high deprivation scores and by scheduling the additional appointments later in the week to increase appointment attendance.

- Missed appointments can also be addressed through low-cost interventions such as training GP receptionists to invite patients to write down upcoming appointment details and actively commit to attending.[4]

Technical Recommendations

- Provide data on appointment attendance at a more granular level (daily vs. monthly, GP practice or sub-ICB vs. ICB) to get a better sense of factors contributing to missed appointments

---

[2]London School of Economics and Political Science (2022) New research suggests patients don't like Mondays when it comes to appointments, with important implications for appointment scheduling. Available at: https://www.lse.ac.uk/News/Latest-news-from-LSE/2022/i-September-22/This-study-emerged-to-solve-a-problem-that-costs-millions-and-causes-lots-of-frustration-amongst-NHS-staff (Accessed: 24 February 2025).

[3] Pandey, A et al (2023) Advanced Sentiment Analysis for Managing and Improving Patient Experience: Application for General Practitioner (GP) Classification in Northamptonshire. Available at: https://www.mdpi.com/1660-4601/20/12/6119 (Accessed: 24 February 2025).

[4] Bull, S. et al (2022) Behaviourally informed, patient-led interventions to reduce missed appointments in general practice: a 12-month implementation study. Available at: https://academic.oup.com/fampra/article/40/1/16/6643534 (Accessed: 24 February 2025).

- Provide additional information related to appointment capacity, such as optimal number of appointments per GP per day and/or the optimal number of registered patients per GP to better understand the strains that increasing demand for GP appointments places on the NHS

Appendix

Importing Data

Data viewing and validation Functions

```
# Data viewing function
def view_data(df):
    #shape
    print("DataFrame shape:", df.shape)

    #head
    print("DataFrame head:")
    print(df.head)

    #data types
    print("DataFrame data types:")
    print(df.dtypes)
```

```
# Data validation function
def validate_data(df):
    #check for unique values
    unique_counts = df.nunique()
    print("Unique values per column:")
    print(unique_counts)

    #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
```

Import 'gp_icb' with no header and rename relevant columns

```
# Reload csv with no header
gp_icb = pd.read_csv('epraccur.csv', header=None)

# Create header
gp_icb_header = gp_icb.rename(columns={0: 'Practice.Code', 3: 'icb_code'})
```

Create a duplicate of this ce

Cleaning Data

Rename 'icb_name' columns

```
# Rename columns of icb_name for merging
icb_rename = icb_name.rename(columns={'ICB22CD': 'icb_ons_code', 'ICB22CDH': 'icb_code','ICB22NM': 'icb_name'})
```

Make column names lower case

```
# Make gp columns lower case
gp.columns = gp.columns.str.lower()
```

Converted column to category and set order of unique values

```
# Remove Unknown row
attd_tbbaa_filt = attd_tbbaa.loc[(attd_tbbaa['time_between_book_and_appointment'] != 'Unknown / Data Quality')]

# Change time_between_book_and_appointment to category
attd_tbbaa_filt = attd_tbbaa_filt.astype({'time_between_book_and_appointment':'category'})

# set order of category values
attd_tbbaa_filt['time_between_book_and_appointment'] = attd_tbbaa_filt['time_between_book_and_appointment'] \
    .cat.reorder_categories(['Same Day', '1 Day', '2 to 7 Days', '8 to 14 Days', '15 to 21 Days', \
                             '22 to 28 Days', 'More than 28 Days'], ordered=True)
```

Converted year column to datetime format

```
# Convert year to datetime
imd_icb['Year'] = pd.to_datetime(imd_icb['Year'], format='%Y')
```

Created new column with names of days and set as ordered category

```
# Group by appointment day and sum count of appointments
nc_day = nc.groupby(nc['appointment_date'])[['count_of_appointments']] \
.sum().reset_index()

# Create new column with day names
nc_day['day_of_week'] = nc_day['appointment_date'].dt.day_name()

# Group by day names and sum appointment count
nc_day_of_week_mean = nc_day.groupby(nc_day['day_of_week'])[['count_of_appointments']] \
.mean().reset_index()

# Change time_between_book_and_appointment to category
nc_day_of_week_mean = nc_day_of_week_mean.astype({'day_of_week':'category'})

# set order of category values
nc_day_of_week_mean['day_of_week'] = nc_day_of_week_mean['day_of_week'] \
    .cat.reorder_categories(['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', \
                             'Saturday', 'Sunday'], ordered=True)
```

Analyzing Data

Calculated appointments per working day for each month

```
# Group by month and appointment status
month_appt = ar.groupby([ar['appointment_month']])[['count_of_appointments']].sum().reset_index()

# Change month data type to datetime
month_appt['appointment_month'] = pd.to_datetime(month_appt['appointment_month'])

# Load working days dataset
wd = pd.read_excel('working_days.xlsx')

# Merge datasets
appt_wd = pd.merge(month_appt, wd, on='appointment_month', how='left')

# Calculate appointments per working day
appt_wd['appt_per_day'] = (appt_wd['count_of_appointments'] / appt_wd['working_days'])

# Drop extra columns
appt_wd = appt_wd.drop(columns=['count_of_appointments', 'working_days'])

# Set date as the index
appt_wd.set_index('appointment_month', inplace=True)
```

Asked Claude AI Chatbot to create attended appointments Percentage calculator

After struggling with find an efficient way to calculate percentages for unique values within groups of a DataFrame, I asked the Claude AI chatbot to draft a function to automate this task.

```
def calc_attd_pct_by_cols(df, group_columns):
    """
    Calculate attendance percentages for each unique value in specified columns
    and automatically create DataFrames for each column.
    """
    for column in group_columns:
        # Create DataFrame name based on column
        attd_name = f"attd_{column.lower().replace(' ', '_')}"

        # Calculate percentage
        group_totals = df.groupby(column)['count_of_appointments'].sum()
        attended = df[df['appointment_status'] == 'Attended'].groupby(column)['count_of_appointments'].sum()
        attendance_percentage = (attended / group_totals * 100).round(2)

        # Create result DataFrame
        result_df = pd.DataFrame({
            column: attendance_percentage.index,
            'total_appointments': group_totals.values,
            'attended_appointments': attended.values,
            'attendance_percentage': attendance_percentage.values
        })

        # Create the DataFrame in the global namespace
        globals()[attd_name] = result_df

        # Print information about the created DataFrame
        print(f"\nCreated DataFrame '{attd_name}' for column '{column}'")
        print(f"Shape: {result_df.shape}")
        print("-" * 50)
        print(result_df)
        print("\n")

# Create dataframes for each column
columns_to_analyze = ['icb_ons_code', 'appointment_month', 'hcp_type',
                      'appointment_mode', 'time_between_book_and_appointment']
calc_attd_pct_by_cols(ar, columns_to_analyze)
```

Calculated patient:clinical staff ratio by ICB for June 2022

```
# Filter for staff group != 'Admin/Non-Clinical'
filtered_clin = gp[gp['staff_group'] != 'Admin/Non-Clinical']

# Group by sub-ICB, sum FTE whilst keeping ICB name, ICB code, and region name
grouped_clin = filtered_clin.groupby([filtered_clin['sub_icb_code'], filtered_clin['icb_code'],
                                filtered_clin['icb_name'], filtered_clin['comm_region_name']])[['fte']].sum().reset_index()

# Merge with patient dataframe
clin_pat = pd.merge(grouped_clin,pat_subicb, on='sub_icb_code', how='left')

# Group by ICB, sum FTE and number of patients, keep ICB name and region name Columns
clin_pat_icb = clin_pat.groupby([clin_pat['icb_code'], clin_pat['icb_name'], \
                               clin_pat['comm_region_name']]).agg({'fte':'sum','number_of_patients':'sum'}).reset_index()

# New column with patient to clinical staff ratio
clin_pat_icb['patients_per_clin'] = (clin_pat_icb['number_of_patients'] / clin_pat_icb['fte'])

# Merge gp_pat_icb with icb_rename to add icb ons code
clin_pat_icb_code = pd.merge(clin_pat_icb, icb_rename[['icb_code', 'icb_ons_code']], on='icb_code', how='left')

# Merge gp_pat_icb_code with attendance percentage data for June 2022
clin_pat_attd = pd.merge(clin_pat_icb_code, attd_june22, on='icb_ons_code', how='left')

# View dataframe
```

Example of the filter, group, and aggregate process in preparation for merge

```
# Add icb_code via merge
imd_icb = pd.merge(gp_imd, gp_icb_header[['Practice.Code', 'icb_code']], on='Practice.Code', how='left')

# Convert year to datetime
imd_icb['Year'] = pd.to_datetime(imd_icb['Year'], format='%Y')

# Filter data for date range of provided data
filtered_imd_icb = imd_icb.loc[(imd_icb['Year'] >= '2020') & (imd_icb['Year'] < '2023')]

# Group by ICB and average the IMD score
gb_icb_imd = filtered_imd_icb.groupby(filtered_imd_icb['icb_code'])[['IMD']].mean().reset_index()

# Add icb_ons code via merge with icb_rename
imd_ons = pd.merge(icb_rename, gb_icb_imd, on='icb_code', how='left')
```

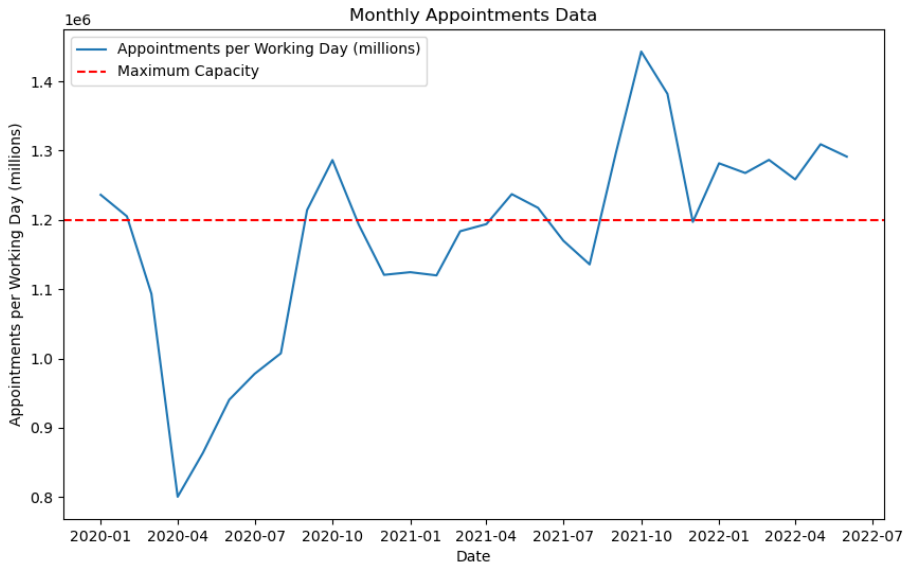
Keyword count and search functions

```
def keyword_count(keyword):
    """ Calculates the number of times a keyword appears in the tweets. """
    return tweets[tweets['tweet_full_text'].str.contains(keyword, case=False, na=False)].shape[0]
```

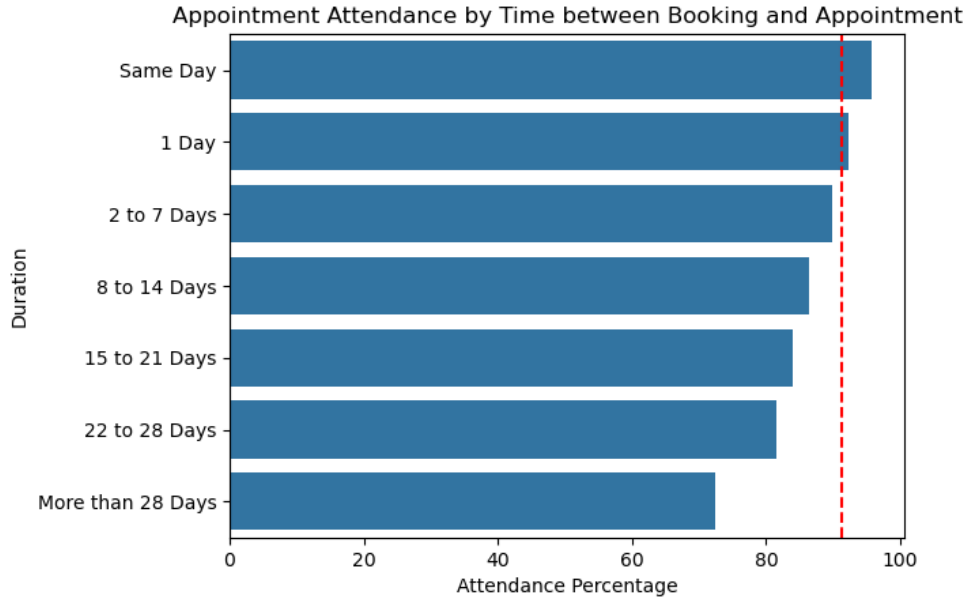
```
def keyword_results(keyword):
    """ Displays the tweets where the keyword appears. """
    pd.set_option('display.max_colwidth', None)
    return tweets[tweets['tweet_full_text'].str.contains(keyword, case=False, na=False)]['tweet_full_text']
```

Visualisations

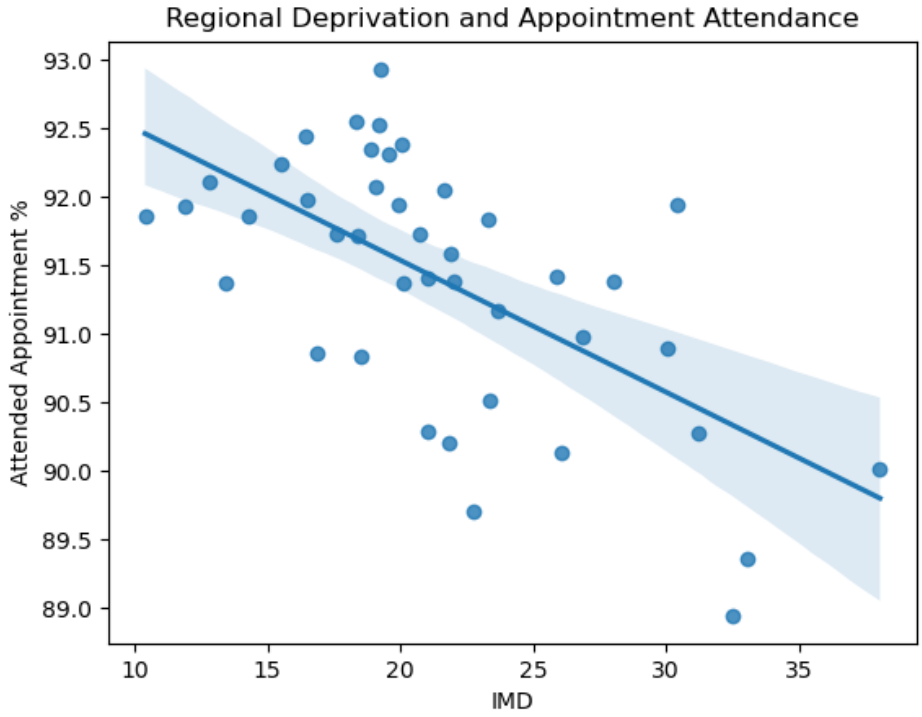
Sample Line Plot



Sample Bar Plot



Sample Scatter Plot

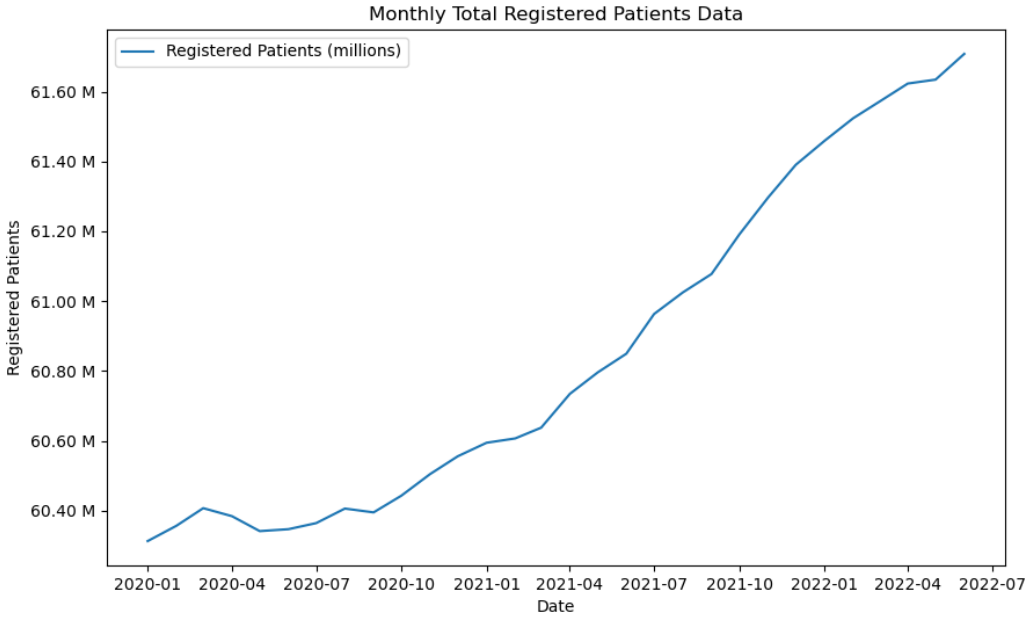


Use EngFormatter function to make large values appear properly on axes

```
plt.gca().yaxis.set_major_formatter(ticker.EngFormatter(unit='', places=2))
```

Patterns

Total Number of Registered Patients by Month



Seasonal Decomposition Trend of Appointments per Working Day