In [1]:
```
import pandas as pd
import requests
from io import StringIO
import matplotlib.pyplot as plt
import numpy as np
```

In [2]:
```
from scripts.NHS_Data_Extraction.AandE_data import AandEData
# User input: Start and End Month-Year (Modify these values)
start_date = "April 2018"  # the desired start
end_date = "February 2025"      # the desired end

combined_df = AandEData().download_data(start_date,end_date)
```

```
Accessing: https://www.england.nhs.uk/statistics/statistical-work-areas/ae-waitin
g-times-and-activity/ae-attendances-and-emergency-admissions-2017-18/
Accessing: https://www.england.nhs.uk/statistics/statistical-work-areas/ae-waitin
g-times-and-activity/ae-attendances-and-emergency-admissions-2018-19/
Downloaded & Loaded: March 2019
Downloaded & Loaded: February 2019
Downloaded & Loaded: January 2019
Downloaded & Loaded: December 2018
Downloaded & Loaded: November 2018
Downloaded & Loaded: October 2018
Downloaded & Loaded: September 2018
Downloaded & Loaded: August 2018
Downloaded & Loaded: July 2018
Downloaded & Loaded: June 2018
Downloaded & Loaded: May 2018
Downloaded & Loaded: April 2018
Accessing: https://www.england.nhs.uk/statistics/statistical-work-areas/ae-waitin
g-times-and-activity/ae-attendances-and-emergency-admissions-2019-20/
Downloaded & Loaded: March 2020
Downloaded & Loaded: February 2020
Downloaded & Loaded: January 2020
Downloaded & Loaded: December 2019
Downloaded & Loaded: November 2019
Downloaded & Loaded: October 2019
Downloaded & Loaded: September 2019
Downloaded & Loaded: August 2019
Downloaded & Loaded: July 2019
Downloaded & Loaded: June 2019
Downloaded & Loaded: May 2019
Downloaded & Loaded: April 2019
Accessing: https://www.england.nhs.uk/statistics/statistical-work-areas/ae-waitin
g-times-and-activity/ae-attendances-and-emergency-admissions-2020-21/
Downloaded & Loaded: March 2021
Downloaded & Loaded: February 2021
Downloaded & Loaded: January 2021
Downloaded & Loaded: December 2020
Downloaded & Loaded: November 2020
Downloaded & Loaded: October 2020
Downloaded & Loaded: September 2020
Downloaded & Loaded: August 2020
Downloaded & Loaded: July 2020
Downloaded & Loaded: June 2020
Downloaded & Loaded: May 2020
Downloaded & Loaded: April 2020
Accessing: https://www.england.nhs.uk/statistics/statistical-work-areas/ae-waitin
g-times-and-activity/ae-attendances-and-emergency-admissions-2021-22/
Downloaded & Loaded: March 2022
Downloaded & Loaded: February 2022
Downloaded & Loaded: January 2022
Downloaded & Loaded: December 2021
Downloaded & Loaded: November 2021
Downloaded & Loaded: October 2021
Downloaded & Loaded: September 2021
Downloaded & Loaded: August 2021
Downloaded & Loaded: June 2021
Downloaded & Loaded: May 2021
Downloaded & Loaded: April 2021
Accessing: https://www.england.nhs.uk/statistics/statistical-work-areas/ae-waitin
g-times-and-activity/ae-attendances-and-emergency-admissions-2022-23/
Downloaded & Loaded: March 2023
```

```
Downloaded & Loaded: February 2023
Downloaded & Loaded: January 2023
Downloaded & Loaded: December 2022
Downloaded & Loaded: November 2022
Downloaded & Loaded: October 2022
Downloaded & Loaded: September 2022
Downloaded & Loaded: August 2022
Downloaded & Loaded: July 2022
Downloaded & Loaded: June 2022
Downloaded & Loaded: May 2022
Downloaded & Loaded: April 2022
Accessing: https://www.england.nhs.uk/statistics/statistical-work-areas/ae-waitin
g-times-and-activity/ae-attendances-and-emergency-admissions-2023-24/
Downloaded & Loaded: March 2024
Downloaded & Loaded: February 2024
Downloaded & Loaded: January 2024
Downloaded & Loaded: December 2023
Downloaded & Loaded: November 2023
Downloaded & Loaded: October 2023
Downloaded & Loaded: September 2023
Downloaded & Loaded: August 2023
Downloaded & Loaded: July 2023
Downloaded & Loaded: June 2023
Downloaded & Loaded: May 2023
Downloaded & Loaded: April 2023
Accessing: https://www.england.nhs.uk/statistics/statistical-work-areas/ae-waitin
g-times-and-activity/ae-attendances-and-emergency-admissions-2024-25/
Downloaded & Loaded: February 2025
Downloaded & Loaded: January 2025
Downloaded & Loaded: December 2024
Downloaded & Loaded: November 2024
Downloaded & Loaded: October 2024
Downloaded & Loaded: September 2024
Downloaded & Loaded: August 2024
Downloaded & Loaded: July 2024
Downloaded & Loaded: June 2024
Downloaded & Loaded: May 2024
Downloaded & Loaded: April 2024
Accessing: https://www.england.nhs.uk/statistics/statistical-work-areas/ae-waitin
g-times-and-activity/ae-attendances-and-emergency-admissions-2025-26/
Failed to access https://www.england.nhs.uk/statistics/statistical-work-areas/ae-
waiting-times-and-activity/ae-attendances-and-emergency-admissions-2025-26/
All valid CSV files loaded into memory and combined.
```

In [3]:

```
# Displaying the structure of the merged DataFrame
print(combined_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 17615 entries, 2653 to 15699
Data columns (total 30 columns):
 #   Column                                                    Non-Null Count
Dtype
---  ------                                                    --------------
-----
 0   Period                                                    17603 non-null
object
 1   Org Code                                                  17603 non-null
object
 2   Parent Org                                                17603 non-null
object
 3   Org name                                                  17615 non-null
object
 4   A&E attendances Type 1                                    17615 non-null
int64
 5   A&E attendances Type 2                                    17615 non-null
int64
 6   A&E attendances Other A&E Department                      17615 non-null
int64
 7   Attendances over 4hrs Type 1                              17615 non-null
int64
 8   Attendances over 4hrs Type 2                              17615 non-null
int64
 9   Attendances over 4hrs Other Department                    17615 non-null
int64
 10  Patients who have waited 4-12 hs from DTA to admission    17615 non-null
int64
 11  Patients who have waited 12+ hrs from DTA to admission    17615 non-null
int64
 12  Emergency admissions via A&E - Type 1                     17615 non-null
int64
 13  Emergency admissions via A&E - Type 2                     17615 non-null
int64
 14  Emergency admissions via A&E - Other A&E department       17615 non-null
int64
 15  Other emergency admissions                                17615 non-null
int64
 16  Year                                                      17615 non-null
object
 17  Month                                                     17615 non-null
object
 18  A&E attendances Booked Appointments Type 1                11057 non-null
float64
 19  A&E attendances Booked Appointments Type 2                11057 non-null
float64
 20  A&E attendances Booked Appointments Other Department      11057 non-null
float64
 21  Attendances over 4hrs Booked Appointments Type 1          11057 non-null
float64
 22  Attendances over 4hrs Booked Appointments Type 2          11057 non-null
float64
 23  Attendances over 4hrs Booked Appointments Other Department  11057 non-null
float64
 24  Unnamed: 22                                               0 non-null
float64
 25  Unnamed: 23                                               0 non-null
float64
 26  Unnamed: 24                                               0 non-null
```

```
float64
 27  Unnamed: 25                                                0 non-null
float64
 28  Unnamed: 26                                                0 non-null
float64
 29  a                                                          0 non-null
float64
dtypes: float64(12), int64(12), object(6)
memory usage: 4.2+ MB
None
```

In [4]:
```
combined_df.head()
```

Out[4]:

	Period	Org Code	Parent Org	Org name	A&E attendances Type 1	A&E attendances Type 2	A attendan Other A Departm
2653	MSitAE-April-2018	C82009	NHS ENGLAND MIDLANDS AND EAST (CENTRAL MIDLANDS)	MARKET HARBOROUGH MED.CTR	0	0	
2654	MSitAE-April-2018	NLO11	NHS ENGLAND MIDLANDS AND EAST (CENTRAL MIDLANDS)	MARKET HARBOROUGH URGENT CARE CENTRE	0	0	
2655	MSitAE-April-2018	NLO01	NHS ENGLAND NORTH (CUMBRIA AND NORTH EAST)	NORTHERN DOCTORS URGENT CARE	0	0	4
2656	MSitAE-April-2018	REF	NHS ENGLAND SOUTH WEST (SOUTH WEST SOUTH)	ROYAL CORNWALL HOSPITALS NHS TRUST	6014	0	11
2657	MSitAE-April-2018	RWY	NHS ENGLAND NORTH (YORKSHIRE AND HUMBER)	CALDERDALE AND HUDDERSFIELD NHS FOUNDATION TRUST	11892	0	

5 rows × 30 columns

In [5]:
```
combined_df.tail()
```

Out[5]:

	Period	Org Code	Parent Org	Org name	A&E attendances Type 1	A&E attendances Type 2	atten Oth Depa
15695	MSitAE-FEBRUARY-2025	RYJ	NHS ENGLAND LONDON	IMPERIAL COLLEGE HEALTHCARE NHS TRUST	11067	3367	
15696	MSitAE-FEBRUARY-2025	NQT5F	NHS ENGLAND NORTH WEST	SKELMERSDALE WALK IN CENTRE	0	0	
15697	MSitAE-FEBRUARY-2025	NQT5H	NHS ENGLAND SOUTH WEST	PAULTON MEMORIAL HOSPITAL	0	0	
15698	MSitAE-FEBRUARY-2025	RX1	NHS ENGLAND MIDLANDS	NOTTINGHAM UNIVERSITY HOSPITALS NHS TRUST	14519	1625	
15699	MSitAE-FEBRUARY-2025	RH5	NHS ENGLAND SOUTH WEST	SOMERSET NHS FOUNDATION TRUST	10816	0	

5 rows × 30 columns

In [34]:
```
print("\nSummary Statistics:")
print(combined_df.describe())
```

```
Summary Statistics:
       A&E attendances Type 1  A&E attendances Type 2  \
count            17603.000000            17603.000000   
mean              5965.197410              196.822928   
std               6236.125035              658.177128   
min                  0.000000                0.000000   
25%                  0.000000                0.000000   
50%               5810.000000                0.000000   
75%               9811.500000                0.000000   
max              35503.000000             8623.000000   

       A&E attendances Other A&E Department  Attendances over 4hrs Type 1  \
count                          17603.000000                  17603.000000   
mean                            3064.802420                   1798.076407   
std                             3391.468022                   2433.571806   
min                                0.000000                      0.000000   
25%                              255.000000                      0.000000   
50%                             2035.000000                    695.000000   
75%                             4662.500000                   2998.500000   
max                            20310.000000                  18083.000000   

       Attendances over 4hrs Type 2  Attendances over 4hrs Other Department  \
count                  17603.000000                            17603.000000   
mean                       4.952792                               85.756348   
std                       29.064157                              266.122398   
min                        0.000000                                0.000000   
25%                        0.000000                                0.000000   
50%                        0.000000                                0.000000   
75%                        0.000000                               42.000000   
max                      680.000000                             4808.000000   

       Patients who have waited 4-12 hs from DTA to admission  \
count                                       17603.000000        
mean                                          367.456911        
std                                           547.175197        
min                                             0.000000        
25%                                             0.000000        
50%                                           103.000000        
75%                                           587.000000        
max                                          8944.000000        

       Patients who have waited 12+ hrs from DTA to admission  \
count                                       17603.000000        
mean                                           83.018065        
std                                           222.557451        
min                                             0.000000        
25%                                             0.000000        
50%                                             0.000000        
75%                                            12.000000        
max                                          2453.000000        

       Emergency admissions via A&E - Type 1  \
count                           17603.000000   
mean                             1768.516901   
std                              1886.733267   
min                                 0.000000   
25%                                 0.000000   
50%                              1639.000000   
75%                              2955.000000   
max                             11732.000000   
```

```
       Emergency admissions via A&E - Type 2  \
count                             17603.00000
mean                                  6.77822
std                                  61.44800
min                                   0.00000
25%                                   0.00000
50%                                   0.00000
75%                                   0.00000
max                                1944.00000

       Emergency admissions via A&E - Other A&E department  \
count                                        17603.000000
mean                                            23.011703
std                                            106.876293
min                                              0.000000
25%                                              0.000000
50%                                              0.000000
75%                                              0.000000
max                                           3248.000000

       Other emergency admissions
count                17603.000000
mean                   594.402204
std                    770.244677
min                      0.000000
25%                      0.000000
50%                    323.000000
75%                    927.000000
max                   7741.000000
```

In [7]:

```python
# Checking for missing values after dropping unnecessary columns
missing_values_after_cleanup = combined_df.isnull().sum()
print("Missing values in dataset after initial cleanup:")
print(missing_values_after_cleanup[missing_values_after_cleanup > 0])
```

```
Missing values in dataset after initial cleanup:
Period                                                    12
Org Code                                                  12
Parent Org                                                12
A&E attendances Booked Appointments Type 1              6558
A&E attendances Booked Appointments Type 2              6558
A&E attendances Booked Appointments Other Department    6558
Attendances over 4hrs Booked Appointments Type 1        6558
Attendances over 4hrs Booked Appointments Type 2        6558
Attendances over 4hrs Booked Appointments Other Department    6558
Unnamed: 22                                            17615
Unnamed: 23                                            17615
Unnamed: 24                                            17615
Unnamed: 25                                            17615
Unnamed: 26                                            17615
a                                                      17615
dtype: int64
```

In [8]:

```python
categorical_columns = ['Org Code', 'Parent Org', 'Org name']
for col in categorical_columns:
    combined_df[col] = combined_df[col].astype('category')
```

In [9]:

```python
columns_to_keep = ['Period', 'Org Code', 'Parent Org', 'Org name']  # Columns to
```

```
# Identifying columns to drop (those with null values but NOT in columns_to_keep
columns_to_drop = [col for col in combined_df.columns if col not in columns_to_k

# Dropping only those columns
combined_df.drop(columns=columns_to_drop, inplace=True)

combined_df.dropna(inplace=True) # Dropping rows with missing values
```

In [10]:
```
# Check for duplicate rows
duplicate_count = combined_df.duplicated().sum()
print(f"Number of duplicate rows: {duplicate_count}")

if duplicate_count > 0:
    # Drop duplicate rows
    combined_df.drop_duplicates(inplace=True)
    print("Duplicate rows removed!")
```

```
Number of duplicate rows: 0
```

In [11]:
```
combined_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 17603 entries, 2653 to 15699
Data columns (total 18 columns):
 #   Column                                             Non-Null Count  Dtyp
e
---  ------                                             --------------  ----
-
 0   Period                                             17603 non-null  obje
ct
 1   Org Code                                           17603 non-null  cate
gory
 2   Parent Org                                         17603 non-null  cate
gory
 3   Org name                                           17603 non-null  cate
gory
 4   A&E attendances Type 1                             17603 non-null  int6
4
 5   A&E attendances Type 2                             17603 non-null  int6
4
 6   A&E attendances Other A&E Department               17603 non-null  int6
4
 7   Attendances over 4hrs Type 1                       17603 non-null  int6
4
 8   Attendances over 4hrs Type 2                       17603 non-null  int6
4
 9   Attendances over 4hrs Other Department             17603 non-null  int6
4
 10  Patients who have waited 4-12 hs from DTA to admission  17603 non-null  int6
4
 11  Patients who have waited 12+ hrs from DTA to admission  17603 non-null  int6
4
 12  Emergency admissions via A&E - Type 1              17603 non-null  int6
4
 13  Emergency admissions via A&E - Type 2              17603 non-null  int6
4
 14  Emergency admissions via A&E - Other A&E department  17603 non-null  int6
4
 15  Other emergency admissions                         17603 non-null  int6
4
 16  Year                                               17603 non-null  obje
ct
 17  Month                                              17603 non-null  obje
ct
dtypes: category(3), int64(12), object(3)
memory usage: 2.3+ MB
```

In [12]:

```python
# Convert 'Period' to datetime format (Month-Year format)
combined_df["Period"] = pd.to_datetime(combined_df["Month"] + " " + combined_df[

# Convert to Year-Month format (YYYY-MM) for analysis
combined_df["Period"] = combined_df["Period"].dt.strftime("%Y-%m")

# Verify the conversion
print("Converted 'Period' column data type:")
print(combined_df.dtypes["Period"])

# Display unique periods to confirm formatting
print("\nUnique periods in dataset:")
print(combined_df["Period"].unique())
```

```
Converted 'Period' column data type:
object

Unique periods in dataset:
['2018-04' '2018-05' '2018-06' '2018-07' '2018-08' '2018-09' '2018-10'
 '2018-11' '2018-12' '2019-01' '2019-02' '2019-03' '2019-04' '2019-05'
 '2019-06' '2019-07' '2019-08' '2019-09' '2019-10' '2019-11' '2019-12'
 '2020-01' '2020-02' '2020-03' '2020-04' '2020-05' '2020-06' '2020-07'
 '2020-08' '2020-09' '2020-10' '2020-11' '2020-12' '2021-01' '2021-02'
 '2021-03' '2021-04' '2021-05' '2021-06' '2021-08' '2021-09' '2021-10'
 '2021-11' '2021-12' '2022-01' '2022-02' '2022-03' '2022-04' '2022-05'
 '2022-06' '2022-07' '2022-08' '2022-09' '2022-10' '2022-11' '2022-12'
 '2023-01' '2023-02' '2023-03' '2023-04' '2023-05' '2023-06' '2023-07'
 '2023-08' '2023-09' '2023-10' '2023-11' '2023-12' '2024-01' '2024-02'
 '2024-03' '2024-04' '2024-05' '2024-06' '2024-07' '2024-08' '2024-09'
 '2024-10' '2024-11' '2024-12' '2025-01' '2025-02']
```

In [77]:

```python
from sqlalchemy import create_engine, text
from sqlalchemy.orm import sessionmaker

# Define your PostgreSQL database credentials
POSTGRES_URL = "localhost:5432"
POSTGRES_DB = "mydatabase"
POSTGRES_USER = "myuser"
POSTGRES_PASSWORD = "mypassword"

# Create the database URL for SQLAlchemy
database_url = f"postgresql://{POSTGRES_USER}:{POSTGRES_PASSWORD}@{POSTGRES_URL}

# Create an engine
engine = create_engine(database_url)

# Optional: Use a sessionmaker if you plan to do ORM operations
Session = sessionmaker(bind=engine)
session = Session()
```

In [21]:

```python
combined_df.to_sql(
    name='nhs_ae_attendances',  # Name of the table to write to
    con=engine,  # SQLAlchemy engine created earlier
    index=False,  # Do not write DataFrame index as a column
    if_exists='replace'  # If table exists, drop it, recreate it, and insert dat
)
```

Out[21]:

```
603
```

In [78]:

```python
# Using the engine to execute a raw SQL query to verify the contents
with engine.connect() as connection:
    result = connection.execute(text("SELECT * FROM nhs_ae_attendances LIMIT 5")
    for row in result:
        print(row)
```

```
('2018-04', 'C82009', 'NHS ENGLAND MIDLANDS AND EAST (CENTRAL MIDLANDS)', 'MARKET
HARBOROUGH MED.CTR', 0, 0, 356, 0, 0, 0, 0, 0, 0, 0, 0, 0, '2018', 'April')
('2018-04', 'NLO11', 'NHS ENGLAND MIDLANDS AND EAST (CENTRAL MIDLANDS)', 'MARKET
HARBOROUGH URGENT CARE CENTRE', 0, 0, 637, 0, 0, 14, 0, 0, 0, 0, 0, 0, '2018', 'A
pril')
('2018-04', 'NLO01', 'NHS ENGLAND NORTH (CUMBRIA AND NORTH EAST)', 'NORTHERN DOCT
ORS URGENT CARE', 0, 0, 4532, 0, 0, 154, 0, 0, 0, 0, 0, 0, '2018', 'April')
('2018-04', 'REF', 'NHS ENGLAND SOUTH WEST (SOUTH WEST SOUTH)', 'ROYAL CORNWALL H
OSPITALS NHS TRUST', 6014, 0, 11044, 355, 0, 67, 49, 0, 2685, 0, 33, 1036, '201
8', 'April')
('2018-04', 'RWY', 'NHS ENGLAND NORTH (YORKSHIRE AND HUMBER)', 'CALDERDALE AND HU
DDERSFIELD NHS FOUNDATION TRUST', 11892, 0, 0, 1009, 0, 0, 192, 0, 2939, 0, 0, 13
78, '2018', 'April')
```

In [23]:
```
session.close()
```

Exploratory Data Analysis

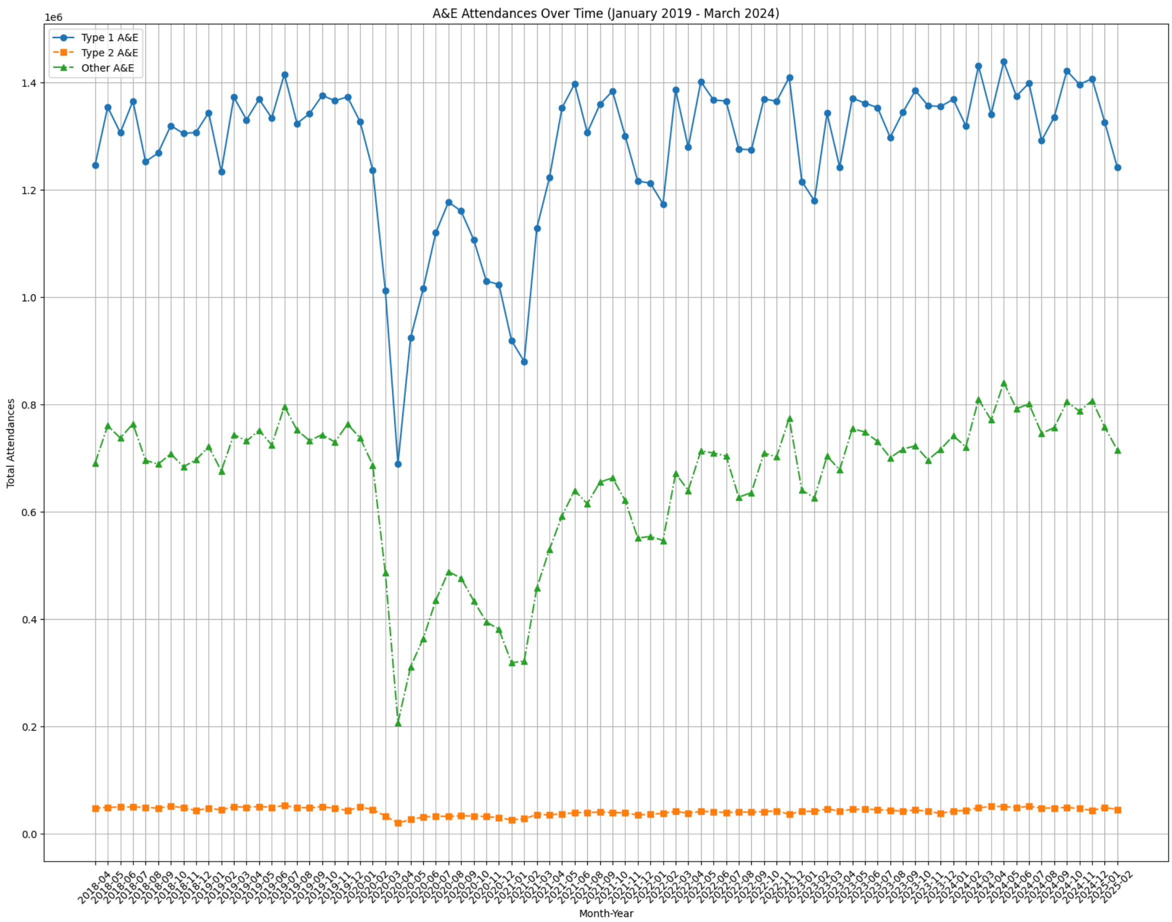
In [37]:
```
from sqlalchemy import create_engine, MetaData, Table

# Create an engine
engine = create_engine('postgresql://myuser:mypassword@localhost:5432/mydatabase

# Reflect the existing database
metadata = MetaData()
metadata.reflect(bind=engine)

# Access the table
table = metadata.tables['nhs_ae_attendances']

# Print column names
print("Field names in the table:")
print([column.name for column in table.columns])
```

```
Field names in the table:
['Period', 'Org Code', 'Parent Org', 'Org name', 'A&E attendances Type 1', 'A&E a
ttendances Type 2', 'A&E attendances Other A&E Department', 'Attendances over 4hr
s Type 1', 'Attendances over 4hrs Type 2', 'Attendances over 4hrs Other Departmen
t', 'Patients who have waited 4-12 hs from DTA to admission', 'Patients who have
waited 12+ hrs from DTA to admission', 'Emergency admissions via A&E - Type 1',
'Emergency admissions via A&E - Type 2', 'Emergency admissions via A&E - Other A&
E department', 'Other emergency admissions', 'Year', 'Month']
```

A&E Attendance Trends Over Time

In [29]:
```
# Aggregating total A&E attendances per period
query = """
SELECT
    "Period",
    SUM("A&E attendances Type 1") AS "A&E attendances Type 1",
    SUM("A&E attendances Type 2") AS "A&E attendances Type 2",
    SUM("A&E attendances Other A&E Department") AS "A&E attendances Other A&E De
FROM
    nhs_ae_attendances
GROUP BY
    "Period"
ORDER BY
    "Period";
```

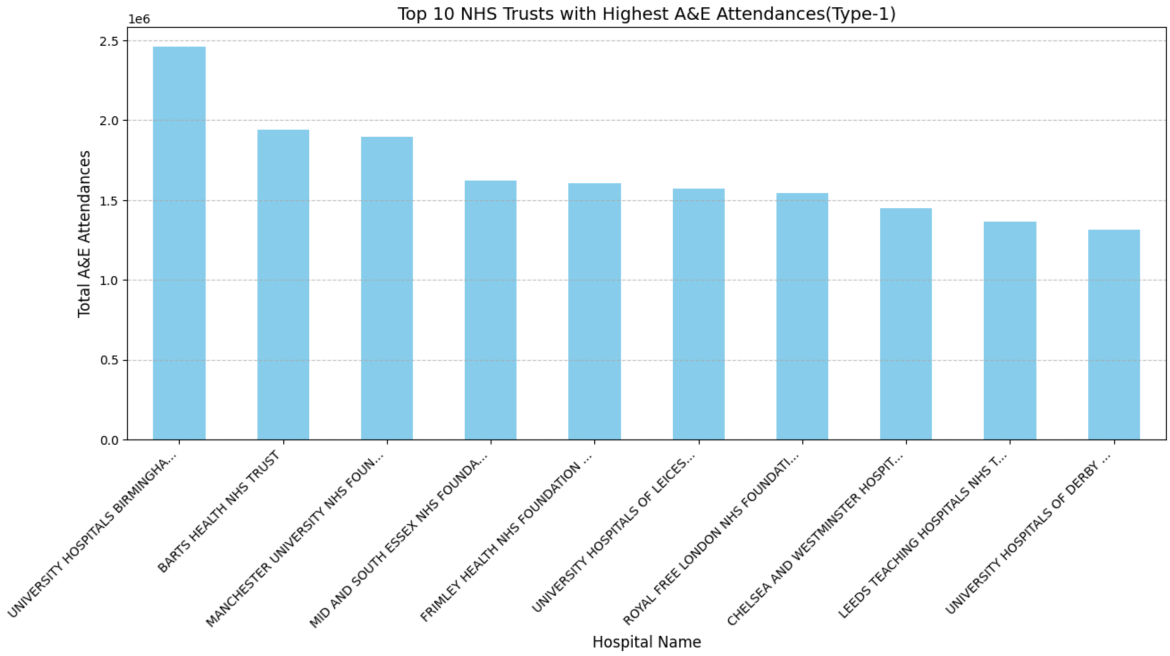
```
"""

# Execute query and load data into DataFrame
monthly_trends = pd.read_sql_query(query, engine)

#monthly_trends = combined_df.groupby("Period")[["A&E attendances Type 1", "A&E

# Plotting the trends
plt.figure(figsize=(20,15))
plt.plot(monthly_trends["Period"], monthly_trends["A&E attendances Type 1"], mar
plt.plot(monthly_trends["Period"], monthly_trends["A&E attendances Type 2"], mar
plt.plot(monthly_trends["Period"], monthly_trends["A&E attendances Other A&E Dep

plt.title("A&E Attendances Over Time (January 2019 - March 2024)")
plt.xlabel("Month-Year")
plt.ylabel("Total Attendances")
plt.legend()
plt.xticks(rotation=45)
plt.grid()
plt.show()
```



Identifying Hospitals with the Highest A&E Attendances

In [30]:
```
# Grouping by hospital and then sum A&E attendances
query = """
SELECT "Org name",
       SUM("A&E attendances Type 1") AS "A&E Attendances Type 1"
FROM nhs_ae_attendances
GROUP BY "Org name"
ORDER BY "A&E Attendances Type 1" DESC LIMIT 10;  -- Order by total to find top
"""
```

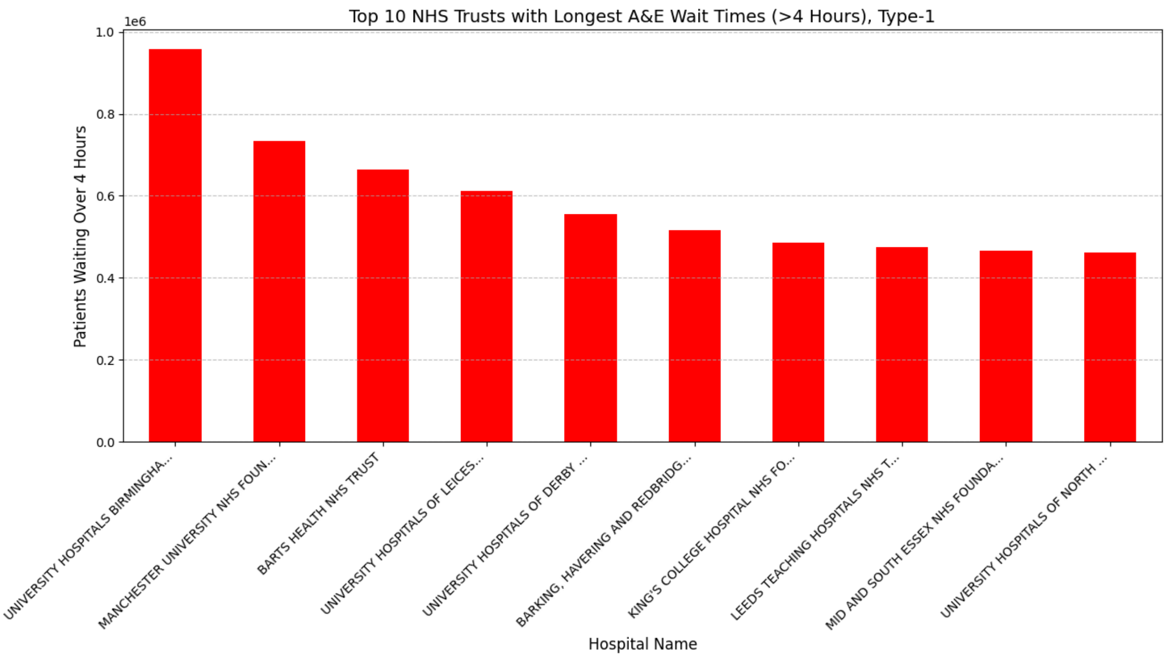
```
top_hospitals = pd.read_sql_query(query, engine)

# Getting the top 10 hospitals with the highest A&E type 1 attendances
'''A&E Attendances Type 1" refers to patients visiting a consultant-led,
    24-hour emergency department with full resuscitation facilities
    for severe, life-threatening conditions.'''

leng = 30
top_hospitals.index = [name[:leng] + "..." if len(name) > leng else name for nam

fig, ax = plt.subplots(figsize=(15, 6))

# Plotting using Matplotlib to maintain figsize control
top_hospitals.plot(kind="bar", color="skyblue", legend=False, ax=ax)

# Adding labels and title
ax.set_title("Top 10 NHS Trusts with Highest A&E Attendances(Type-1)", fontsize=
ax.set_xlabel("Hospital Name", fontsize=12)
ax.set_ylabel("Total A&E Attendances", fontsize=12)

# Formatting x-axis labels
plt.xticks(rotation=45, ha="right")

# Adding grid lines
plt.grid(axis="y", linestyle="--", alpha=0.7)

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



Identifying NHS Trusts with the Worst A&E Waiting Times (>4 Hours)

In [38]:
```
# Grouping by hospital and summing patients who waited over 4 hours in Type 1 A&
query = """
SELECT "Org name",
       SUM("Attendances over 4hrs Type 1") AS "Attendances over 4hrs Type 1"
FROM nhs_ae_attendances
```

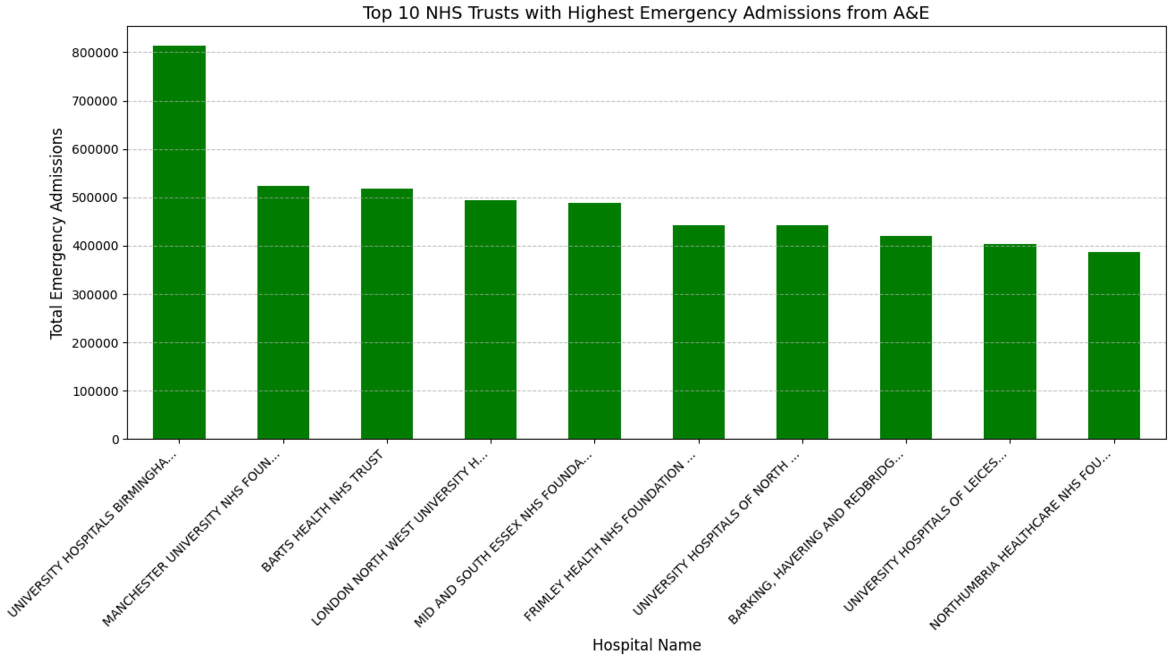
```
GROUP BY "Org name"
ORDER BY "Attendances over 4hrs Type 1" DESC LIMIT 10;  -- Order by total to fin
"""

top_waiting_hospitals = pd.read_sql_query(query, engine)

# Getting the top 10 hospitals with the worst waiting times
''' Attendances over 4hrs Type 1" refers to patients who spent more than 4 hours
    in a consultant-led, 24-hour emergency department before being admitted,
    transferred, or discharged. '''

# Shortening long hospital names for readability
top_waiting_hospitals.index = [name[:30] + "..." if len(name) > 30 else name for

# Creating a Matplotlib figure with correct figsize
fig, ax = plt.subplots(figsize=(15, 6))

# Plotting using Matplotlib to maintain figsize control
top_waiting_hospitals.plot(kind="bar", color="red", legend=False, ax=ax)

# Adding labels and title
ax.set_title("Top 10 NHS Trusts with Longest A&E Wait Times (>4 Hours), Type-1",
ax.set_xlabel("Hospital Name", fontsize=12)
ax.set_ylabel("Patients Waiting Over 4 Hours", fontsize=12)

# Formatting x-axis labels
plt.xticks(rotation=45, ha="right")

# Adding grid lines
plt.grid(axis="y", linestyle="--", alpha=0.7)

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



Comparing Emergency Admissions from A&E

In [40]:
```
import matplotlib.pyplot as plt

# Group by hospital and sum emergency admissions via A&E Type 1
```

```
query = """
SELECT "Org name",
       SUM("Emergency admissions via A&E - Type 1") AS "Emergency admissions via
FROM nhs_ae_attendances
GROUP BY "Org name"
ORDER BY "Emergency admissions via A&E - Type 1" DESC LIMIT 10;  -- Order by tot
"""

# Getting the top 10 hospitals with the highest emergency admissions
''' "Emergency admissions via A&E - Type 1" refers to patients who were admitted
    to the hospital after attending a consultant-led, 24-hour emergency departme

top_admission_hospitals = pd.read_sql_query(query, engine)

# Shortening long hospital names for readability
top_admission_hospitals.index = [name[:30] + "..." if len(name) > 30 else name f

# Creating a Matplotlib figure with correct figsize
fig, ax = plt.subplots(figsize=(15, 6))

# Plotting using Matplotlib to maintain figsize control
top_admission_hospitals.plot(kind="bar", color="green", legend=False, ax=ax)

# Adding labels and title
ax.set_title("Top 10 NHS Trusts with Highest Emergency Admissions from A&E", fon
ax.set_xlabel("Hospital Name", fontsize=12)
ax.set_ylabel("Total Emergency Admissions", fontsize=12)

# Formatting x-axis labels
plt.xticks(rotation=45, ha="right")

# Adding grid lines
plt.grid(axis="y", linestyle="--", alpha=0.7)



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

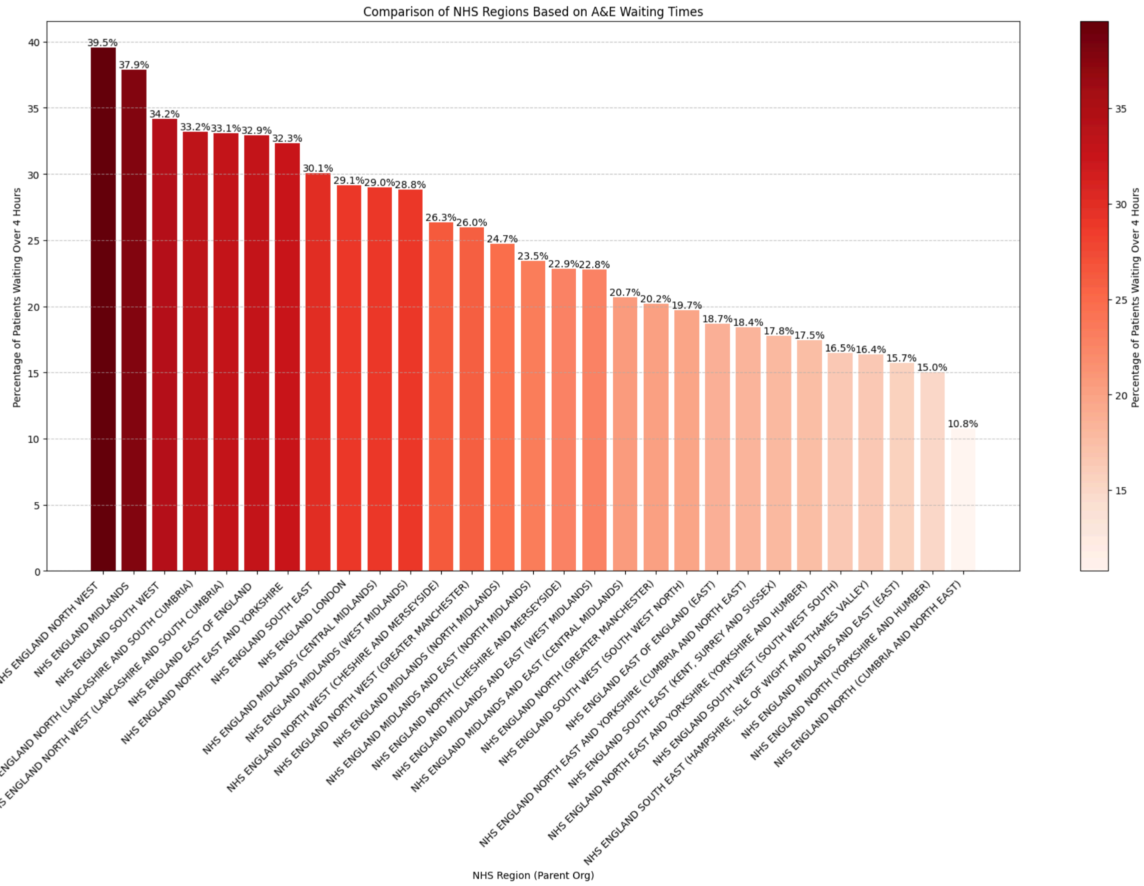


Comparing NHS Regions Based on A&E Performance

In [49]:

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from matplotlib.colors import Normalize
from matplotlib.cm import ScalarMappable

# Grouping data by NHS Parent Org and summing up total A&E attendances and waiti
query = """
SELECT
    "Parent Org",
    SUM("Attendances over 4hrs Type 1") AS "Attendances over 4hrs Type 1", SUM("
FROM
    nhs_ae_attendances
GROUP BY
    "Parent Org";
"""
region_performance = pd.read_sql_query(query, engine)

# Creating a new column for percentage of patients waiting over 4 hours
region_performance["% Waiting Over 4hrs"] = (region_performance["Attendances ove
                                             region_performance["A&E attendances

# Sorting regions based on percentage of patients waiting over 4 hours
region_performance = region_performance.sort_values(by="% Waiting Over 4hrs", as

# Normalizing the data for color mapping
norm = Normalize(vmin=region_performance["% Waiting Over 4hrs"].min(), vmax=regi
cmap = plt.get_cmap("Reds")

# Creating the bar plot
fig, ax = plt.subplots(figsize=(22, 10))

# Applying the color mapping to each bar based on the value of '% Waiting Over 4
bars = ax.bar(region_performance["Parent Org"], region_performance["% Waiting Ov

# Adding color bar for reference
sm = ScalarMappable(cmap=cmap, norm=norm)
sm.set_array([])
fig.colorbar(sm, ax=ax, label='Percentage of Patients Waiting Over 4 Hours')

# Formatting plot
plt.title("Comparison of NHS Regions Based on A&E Waiting Times")
plt.xlabel("NHS Region (Parent Org)")
plt.ylabel("Percentage of Patients Waiting Over 4 Hours")
plt.xticks(rotation=45, ha="right")
plt.grid(axis="y", linestyle="--", alpha=0.7)


# Adding data labels (annotations) to the bars
for bar in bars:
    height = bar.get_height()
    ax.annotate(f'{height:.1f}%',  # Formatting the label to show one decimal po
                xy=(bar.get_x() + bar.get_width() / 2, height),  # Positioning t
                xytext=(0, 10),  # Adjusting the vertical offset of the label ab
                textcoords="offset points",
                ha='center', va='top',  # Centering the label
```

```
                    fontsize=10, color='black')

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

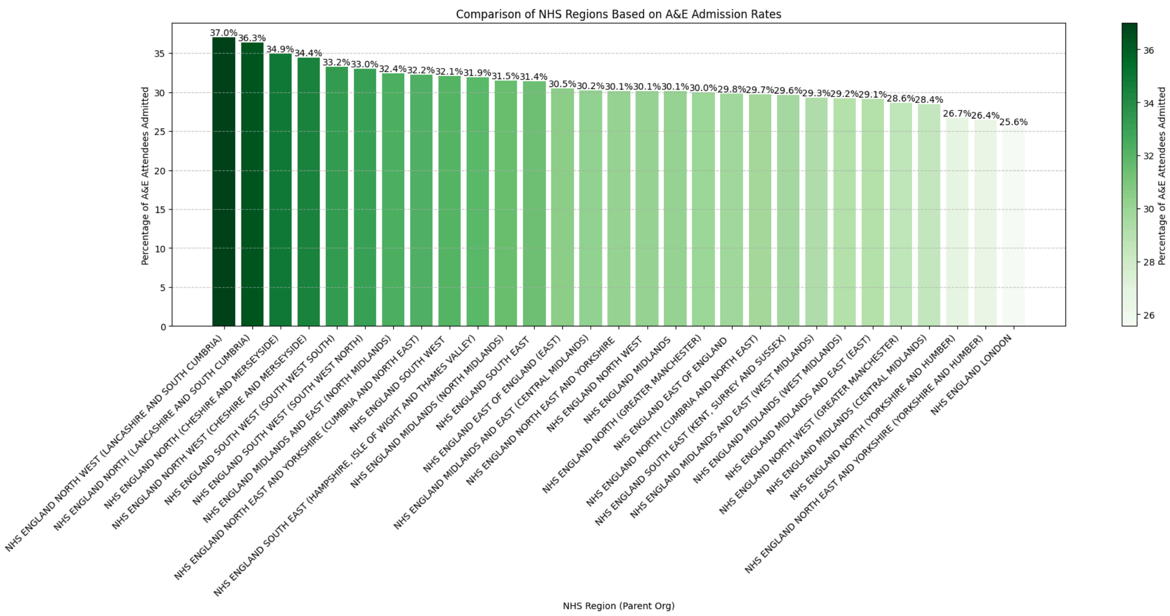


Analyzing Percentage of A&E Attendees Who Were Admitted

In [50]:

```
# Grouping data by NHS Parent Org and summing up total A&E attendances and emerg
query = """
SELECT
    "Parent Org",
    SUM("Emergency admissions via A&E - Type 1") AS "Emergency admissions via A&
FROM
    nhs_ae_attendances
GROUP BY
    "Parent Org";
"""
region_performance = pd.read_sql_query(query, engine)

# Creating a new column for the admission rate (percentage of A&E attendees admi
region_performance["A&E Admission Rate"] = (region_performance["Emergency admiss
                                              region_performance["A&E attendance

# Sorting regions based on the A&E admission rate
region_performance = region_performance.sort_values(by="A&E Admission Rate", asc

# Normalizing the data for color mapping
norm = Normalize(vmin=region_performance["A&E Admission Rate"].min(), vmax=regic
cmap = plt.get_cmap("Greens")

# Creating the bar plot
```

```
fig, ax = plt.subplots(figsize=(22, 6))

# Applying the color mapping to each bar based on the value of 'A&E Admission Ra
bars = ax.bar(region_performance["Parent Org"], region_performance["A&E Admissic

# Adding color bar for reference
sm = ScalarMappable(cmap=cmap, norm=norm)
sm.set_array([])
fig.colorbar(sm, ax=ax, label='Percentage of A&E Attendees Admitted')

# Formatting plot
plt.title("Comparison of NHS Regions Based on A&E Admission Rates")
plt.xlabel("NHS Region (Parent Org)")
plt.ylabel("Percentage of A&E Attendees Admitted")
plt.xticks(rotation=45, ha="right")
plt.grid(axis="y", linestyle="--", alpha=0.7)

# Adding data labels (annotations) to the bars
for bar in bars:
    height = bar.get_height()
    ax.annotate(f'{height:.1f}%',  # Formatting the label to show one decimal po
                xy=(bar.get_x() + bar.get_width() / 2, height),  # Positioning t
                xytext=(0, 10),  # Adjusting the vertical offset of the label ab
                textcoords="offset points",
                ha='center', va='top',  # Centering the label
                fontsize=10, color='black')

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

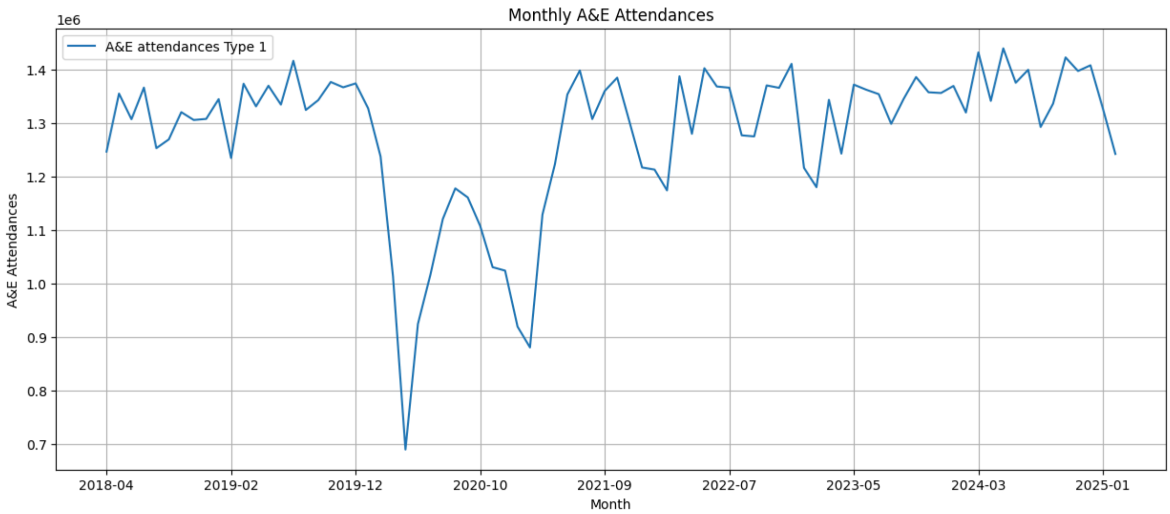


Predictive Analytics

Time-Series Forecasting

In [73]:
```
import pandas as pd

query = """
SELECT
    "Period",
```

```
    SUM("A&E attendances Type 1") AS "A&E attendances Type 1"
FROM
    nhs_ae_attendances
GROUP BY
    "Period"
ORDER BY
    "Period";
"""

# Execute query and load into DataFrame
monthly_data = pd.read_sql_query(query, engine)

# # Ensuring the 'Period' column is in datetime format
# combined_df['Period'] = pd.to_datetime(combined_df['Period'], errors='coerce')

# # Aggregating data by month for time-series forecasting
# monthly_data = combined_df.groupby(combined_df['Period'].dt.to_period('M'))['A

# Visualizing the data to check for trends
monthly_data.set_index('Period', inplace=True)
monthly_data.plot(figsize=(15, 6), title="Monthly A&E Attendances")
plt.xlabel("Month")
plt.ylabel("A&E Attendances")
plt.grid()
plt.show()

monthly_data = monthly_data.reset_index()
```

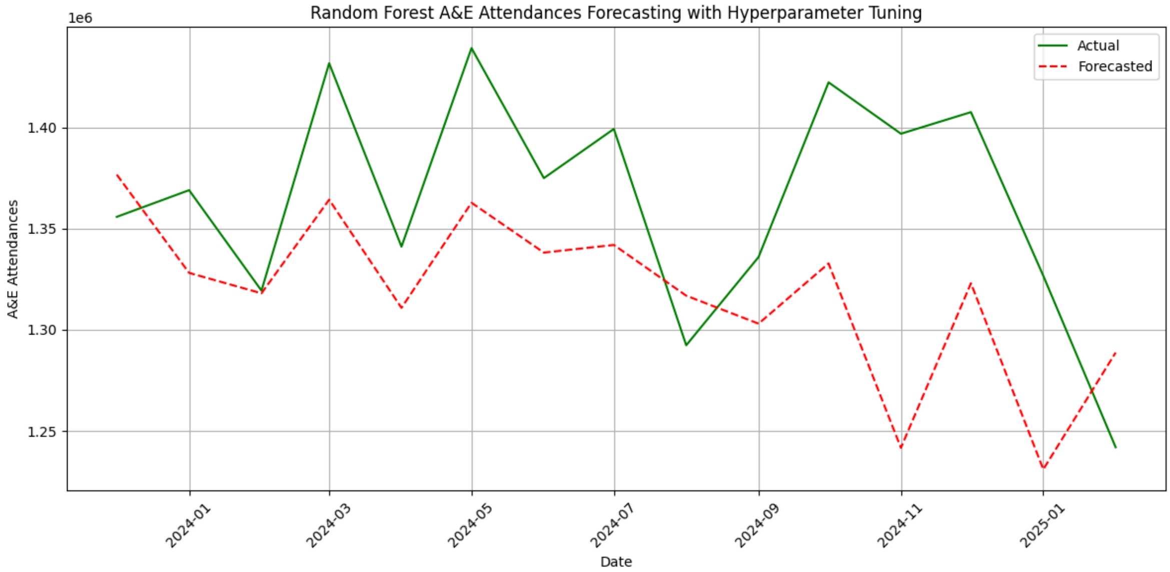


In [75]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error, mean_squared_error
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Preparing data for Random Forest: Create lag features for time series
def create_lag_features(data, lags=12):
    lagged_data = data.copy()
    for i in range(1, lags+1):
        lagged_data[f'lag_{i}'] = lagged_data['y'].shift(i)
    lagged_data.dropna(inplace=True)  # Drop rows with NaN values (due to laggin
    return lagged_data

# Ensuring correct column names
```

```
monthly_data.columns = ['ds', 'y']

# Creating lag features (using previous 12 months as lags)
lagged_data = create_lag_features(monthly_data, lags=12)

# Splitting into train and test data
train_data = lagged_data[:-15]
test_data = lagged_data[-15:]

# Defining features (X) and target (y)
X_train = train_data.drop(columns=['ds', 'y'])
y_train = train_data['y']
X_test = test_data.drop(columns=['ds', 'y'])
y_test = test_data['y']

# Defining Random Forest model
rf = RandomForestRegressor(random_state=42)

# Defining the hyperparameter grid
param_grid = {
    "n_estimators": [50, 100, 200, 500],  # Number of trees in the forest
    "max_depth": [10, 20, 30, None],  # Maximum depth of the tree
    "min_samples_split": [2, 5, 10, 20],  # Minimum samples required to split a
    "min_samples_leaf": [1, 2, 4, 8],  # Minimum samples required at a leaf node
    "criterion": ["squared_error", "absolute_error"]  # Loss function to measure
}

# Performing Grid Search with Cross Validation
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, scoring='n
grid_search.fit(X_train, y_train)

# Getting the best parameters
best_params = grid_search.best_params_
print(f"Best Parameters: {best_params}")

# Training Random Forest with best parameters
best_rf = RandomForestRegressor(**best_params, random_state=42)
best_rf.fit(X_train, y_train)

# Forecasting on the test set
y_pred = best_rf.predict(X_test)

# Plotting actual vs predicted values
plt.figure(figsize=(12, 6))
plt.plot(pd.DatetimeIndex(test_data['ds'].astype(str)), y_test, label='Actual',
plt.plot(pd.DatetimeIndex(test_data['ds'].astype(str)), y_pred, label='Forecaste
plt.title("Random Forest A&E Attendances Forecasting with Hyperparameter Tuning"
plt.xlabel("Date")
plt.ylabel("A&E Attendances")
plt.legend()
plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Evaluating the forecast using MAE, RMSE, and MAPE
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mape = mae / np.mean(y_test) * 100
```

```
print(f"Mean Absolute Error (MAE): {mae}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"Mean Absolute Percentage Error (MAPE): {mape:.2f}%")
```

```
Fitting 5 folds for each of 512 candidates, totalling 2560 fits
Best Parameters: {'criterion': 'squared_error', 'max_depth': 20, 'min_samples_lea
f': 1, 'min_samples_split': 2, 'n_estimators': 200}
```

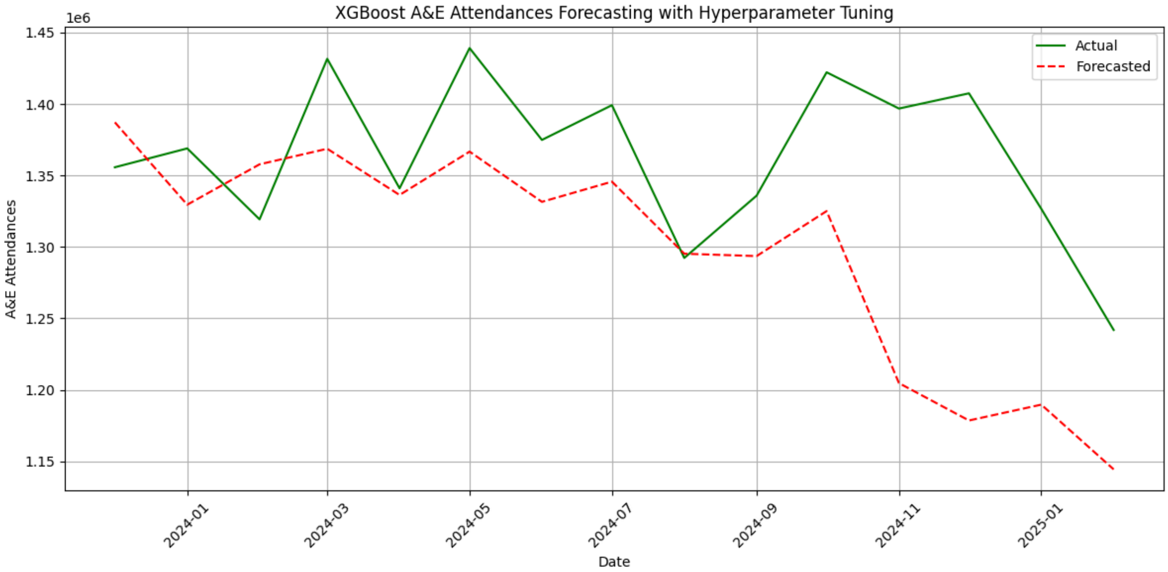



```
Mean Absolute Error (MAE): 57361.44700000002
Root Mean Squared Error (RMSE): 68472.29306007801
Mean Absolute Percentage Error (MAPE): 4.21%
```

In [76]:
```
import xgboost as xgb
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error, mean_squared_error
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Preparing data for XGBoost: Create lag features for time series
def create_lag_features(data, lags=12):
    lagged_data = data.copy()
    for i in range(1, lags+1):
        lagged_data[f'lag_{i}'] = lagged_data['y'].shift(i)
    lagged_data.dropna(inplace=True)  # Drop rows with NaN values (due to laggin
    return lagged_data

# Ensuring correct column names
monthly_data.columns = ['ds', 'y']

# Creating lag features (using previous 12 months as lags)
lagged_data = create_lag_features(monthly_data, lags=12)

# Splitting into train and test data
train_data = lagged_data[:-15]
test_data = lagged_data[-15:]

# Defining features (X) and target (y)
X_train = train_data.drop(columns=['ds', 'y'])
y_train = train_data['y']
X_test = test_data.drop(columns=['ds', 'y'])
y_test = test_data['y']

# Defining XGBoost model
```

```
xgb_model = xgb.XGBRegressor(objective='reg:squarederror', random_state=42)

# Defining hyperparameter grid
param_grid = {
    "n_estimators": [500, 1000],
    "max_depth": [6, 10, 15],
    "learning_rate": [0.001, 0.01, 0.1],
    "subsample": [0.7, 0.9, 1.0],
    "colsample_bytree": [0.7, 0.9, 1.0]
}

# Performing Grid Search with Cross Validation
grid_search = GridSearchCV(
    estimator=xgb_model,
    param_grid=param_grid,
    cv=5,
    scoring='neg_mean_absolute_percentage_error',
    n_jobs=-1,
    verbose=2
)
grid_search.fit(X_train, y_train)

# Getting the best parameters
best_params = grid_search.best_params_
print(f"Best Parameters: {best_params}")

# Training XGBoost with best parameters
best_xgb = xgb.XGBRegressor(**best_params, objective='reg:squarederror', random_
best_xgb.fit(X_train, y_train)

# Forecasting on the test set
y_pred = best_xgb.predict(X_test)

# Plotting actual vs predicted values
plt.figure(figsize=(12, 6))
plt.plot(pd.DatetimeIndex(test_data['ds'].astype(str)), y_test, label='Actual',
plt.plot(pd.DatetimeIndex(test_data['ds'].astype(str)), y_pred, label='Forecaste
plt.title("XGBoost A&E Attendances Forecasting with Hyperparameter Tuning")
plt.xlabel("Date")
plt.ylabel("A&E Attendances")
plt.legend()
plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Evaluating the forecast using MAE, RMSE, and MAPE
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mape = mae / np.mean(y_test) * 100

print(f"Mean Absolute Error (MAE): {mae}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"Mean Absolute Percentage Error (MAPE): {mape:.2f}%")
```

```
Fitting 5 folds for each of 162 candidates, totalling 810 fits
Best Parameters: {'colsample_bytree': 0.9, 'learning_rate': 0.1, 'max_depth': 6,
'n_estimators': 500, 'subsample': 1.0}
```




```
Mean Absolute Error (MAE): 76278.08333333333
Root Mean Squared Error (RMSE): 98938.37513767513
Mean Absolute Percentage Error (MAPE): 5.59%
```