

P4DS Summative Assignment 2

Data Analysis Project

From No-Shows to Scheduling Efficiency: Data Analysis and Predicting Appointment Gaps in Healthcare

Student ID: dpdd2967@leeds.ac.uk

Project Plan

The Data (15 marks)

1. Overview

This project uses the Healthcare Appointment Dataset from **Kaggle** to explore patterns in patient attendance and operational scheduling. The dataset includes information on patient **demographics**, **medical history**, **appointment details**, and **SMS reminders**. By analyzing no-show behavior and predicting the number of days between scheduling and the actual appointment (i.e., scheduling lead time), the project uncovers valuable operational insights. A linear regression model is applied to forecast scheduling gaps based on patient characteristics.

2. Source of the Dataset

Origin:

The dataset is publicly available on **Kaggle**, uploaded by user **wajahat1064**. Kaggle is a reputable platform for data sharing and machine learning competitions, but since this dataset is **user-contributed**, its **original provenance** (e.g., hospital records, government health data) is not independently verified within the dataset description. Therefore, the findings, insights, and predictive models developed in this project **should not be used for real-world** healthcare decision-making or operational planning without further validation against officially sourced medical data.

Recommendation:

For academic or clinical applications, it's crucial to **trace the dataset back to its original collection process**, if possible, to ensure ethical compliance and authenticity.

3. Description of the Dataset

Overview

- **Size:** ~107,000 records
- **Features:** 15 columns per record
- **Each row:** Represents a unique medical appointment

Key Features

The dataset consists of **15 columns**, each representing patient-level details and appointment-related attributes. Here's a breakdown of the features:

- **PatientId** : Unique identifier for each patient. (*Identifier*)
- **AppointmentID** : Unique identifier for each appointment. (*Identifier*)
- **Gender** : Sex of the patient (e.g., 'M', 'F'). (*Categorical*)
- **ScheduledDay** : The date and time the appointment was scheduled. (*Datetime*)
- **AppointmentDay** : The date and time of the actual appointment. (*Datetime*)
- **Age** : Age of the patient. (*Numerical*)
- **Neighbourhood** : The patient's area or community of residence. (*Categorical*)
- **Scholarship** : Whether the patient is enrolled in a social welfare program. (*Boolean: True / False*)
- **Hipertension** : Whether the patient has hypertension. (*Boolean: True / False*)
- **Diabetes** : Whether the patient has diabetes. (*Boolean: True / False*)
- **Alcoholism** : Whether the patient has a history of alcoholism. (*Boolean: True / False*)
- **Handcap** : Indicates whether the patient is handicapped. (*Boolean: True / False, though some datasets represent severity with integers*)
- **SMS_received** : Whether the patient received an SMS reminder. (*Boolean: True / False*)
- **Showed_up** : Whether the patient attended the appointment. (*Boolean: True = showed up, False = no-show*)
- **Date.diff** : Number of days between the **ScheduledDay** and **AppointmentDay**. (*Numerical*)

Data Types

- **Identifiers**
 - **PatientId**
 - **AppointmentID**
- **Categorimcal**
 - **Gender**
 - **Neighbourhood**
- **Datetime**
 - **ScheduledDay**

- AppointmentDay
- Numerical
 - Age
 - Date.diff
- Boolean
 - Scholarship
 - Hipertension
 - Diabetes
 - Alcoholism
 - Handcap
 - SMS_received
 - Showed_up

This combination enables multifaceted analysis—from demographic breakdowns to temporal trends and appointment outcomes.

4. Accuracy and Reliability

Accuracy

The dataset's accuracy is contingent on its **original data source**, which is not explicitly verified on Kaggle. While the volume of data suggests real-world collection, **manual or automated errors** (e.g., input typos, mismatched dates) may exist.

Reliability

Due to the lack of metadata detailing how the data was captured, there may be **unknown biases**, such as:

- **Regional skew** (e.g., specific neighborhoods or clinics)
- **Demographic gaps** (e.g., underrepresentation of certain age groups)
- **Temporal anomalies** (e.g., COVID-era data affecting attendance)

Caution is advised before drawing broad conclusions.

5. Data Quality, Usability

Completeness

- While the dataset contains a substantial number of records (~107,000), it's crucial to check for **missing or null values** in each column.
- Missing values, especially in critical fields like `Showed_up`, `ScheduledDay`, or `AppointmentDay`, could compromise the integrity of the analysis.
- A null check should be part of initial data profiling to assess data readiness.

Consistency

- **Date consistency:** Ensure `ScheduledDay` is not after `AppointmentDay` . Any violations may indicate data entry or processing issues.
- **Age anomalies:** Check for unrealistic values like negative age or extremely high numbers.
- **Boolean uniformity:** All Boolean fields (e.g., `Scholarship` , `Diabetes`) should only have `True / False` .
- **Categorical values:** Validate that columns like `Gender` and `Neighbourhood` do not contain unexpected or misspelled entries.

Duplicates

- Multiple records with the same `PatientId` and `AppointmentDay` may indicate duplicates, unless they are intended repeat visits.
- Deduplication may be necessary before any statistical or predictive analysis.

Usability

- This analysis can be enhanced by combining it with regional or geographical datasets, using the `Neighbourhood` column as a key to join location-based information such as area-level demographics, healthcare access, or infrastructure characteristics, enabling deeper operational and social insights..

Final Thoughts

This dataset provides a **rich foundation** for healthcare appointment analytics, predictive modeling, and behavioral pattern analysis. However, its **lack of documented provenance**, potential **data quality issues**, and need for **contextual understanding** require careful pre-processing and validation.

Project Aim and Objectives (5 marks)

Project Aim

Missed medical appointments are a widespread issue in healthcare systems, leading to wasted resources, longer patient wait times, and reduced quality of care. This project explores the behavioral and systemic factors behind appointment attendance using a dataset of over 100,000 medical appointment records from a public healthcare system.

The primary goal is to answer the question: **What factors influence whether a patient will show up to their scheduled appointment?** Through this, we aim to gain actionable insights and develop predictive tools that can help healthcare providers reduce no-show rates.

This work will involve several types of processing:

- **Exploratory data analysis** to understand trends and assess data quality.

- **Correlation analysis** to explore relationships between features like age, SMS reminders, and attendance.
- **Linear Regression Modeling** to Predict Appointment Waiting Time
- **Visualization** to present trends and disparities across demographics or locations.

Objectives

1. Understand the Data Landscape

- Profile all features and examine missing values, outliers, and inconsistencies.
- Validate date relationships and identify anomalies (e.g., scheduling date after appointment date).

2. Identify Key Predictors

- Analyze the impact of features such as age, neighborhood, medical history, and reminders on attendance.

3. Recommend Interventions

- Use insights to suggest strategies (e.g., targeted SMS reminders) to improve attendance rates.

Specific Objective(s)

- **Objective 1:** Understand why some patients don't show up for appointments.
- **Objective 2:** Operational Scheduling Insights.
- **Objective 3:** Predict the scheduling-to-appointment gap using patient demographics, medical history, and SMS reminders with linear regression

System Design (5 marks)

Describe your code in terms of the following two sections.

Architecture

The project follows a classic machine learning pipeline architecture where raw healthcare appointment data is transformed through a series of preprocessing, feature engineering, analysis, and modeling steps. Each module processes the data systematically to improve quality, extract meaningful patterns, and enable accurate prediction of patient waiting time.

At the heart of the system is a **linear regression model**, trained on carefully engineered features such as age, SMS reminders, medical history, and neighborhood, to predict the number of days between scheduling and the actual appointment date.

Pipeline Diagram:

Processing Modules and Algorithms

1. Data Collection

- Source: *Healthcare Appointment Dataset* from Kaggle.
 - Contains patient demographics, appointment details, SMS reminders, and show/no-show outcomes.
 - Forms the foundation for all subsequent processing and analysis.
-

2. Data Cleaning & Preprocessing

- Handle Missing Values: Identify and treat any null or inconsistent entries.
 - Correct Anomalies: Validate and correct cases where, for example, scheduling dates are after appointment dates.
 - Data Formatting: Ensure consistent data types (e.g., dates parsed correctly, numerical fields standardized).
-

3. Feature Engineering

- Generate New Features:
 - `WaitingTime` = Difference between `AppointmentDay` and `ScheduledDay`.
 - Create binary flags from SMS reminders or medical history indicators.
 - `AgeGroup` = Created bins for different Age groups using `Age` feature
 - Encode Categorical Variables: Neighborhoods and gender encoded for use in regression.
 - Remove Redundancy: Drop irrelevant or leakage-prone columns (like Patient ID if needed).
-

4. Exploratory Data Analysis (EDA)

- Statistical Summaries: Means, medians, ranges, and distributions of features.
 - Correlation Analysis: Identify relationships between features.
 - Outlier Detection: Visualize extreme values using box plots and histograms.
-

5. Objectives After EDA

- Objective 1: Understand Patient No-Shows Analyze trends and patterns to understand why some patients miss their appointments.
 - Objective 2: Operational Scheduling Insights Analysis to identify features most strongly associated with waitingdays.
 - Objective 3: Linear Regression Analysis Predict the scheduling-to-appointment gap using patient demographics, medical history, and SMS reminders with linear regression.
-

This pipeline ensures reliable processing, meaningful insights, and clear presentation of results.

Program Code (25 marks)

Brief Explanation of following code cell

Import Libraries

```
In [20]: #Data Analysis lib
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve,
from scipy.stats import skew
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.pipeline import Pipeline
```

1. Data Collection

Import Dataset

```
In [21]: healthcare_raw_df = pd.read_csv('healthcare_noshows.csv')
```

Display top 5 records

```
In [22]: healthcare_raw_df.head()
```

```
Out[22]:
```

	PatientId	AppointmentID	Gender	ScheduledDay	AppointmentDay	Age	N
0	2.987250e+13	5642903	F	2016-04-29	2016-04-29	62	
1	5.589978e+14	5642503	M	2016-04-29	2016-04-29	56	
2	4.262962e+12	5642549	F	2016-04-29	2016-04-29	62	M
3	8.679512e+11	5642828	F	2016-04-29	2016-04-29	8	
4	8.841186e+12	5642494	F	2016-04-29	2016-04-29	56	

```
In [23]: healthcare_raw_df.shape , healthcare_raw_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 106987 entries, 0 to 106986
Data columns (total 15 columns):
#   Column                Non-Null Count  Dtype
---  -
0   PatientId             106987 non-null float64
1   AppointmentID         106987 non-null int64
2   Gender                106987 non-null object
3   ScheduledDay          106987 non-null object
4   AppointmentDay        106987 non-null object
5   Age                  106987 non-null int64
6   Neighbourhood         106987 non-null object
7   Scholarship           106987 non-null bool
8   Hipertension          106987 non-null bool
9   Diabetes              106987 non-null bool
10  Alcoholism            106987 non-null bool
11  Handcap               106987 non-null bool
12  SMS_received          106987 non-null bool
13  Showed_up             106987 non-null bool
14  Date.diff             106987 non-null int64
dtypes: bool(7), float64(1), int64(3), object(4)
memory usage: 7.2+ MB
```

Out[23]: ((106987, 15), None)

In [24]: `healthcare_raw_df.describe(include='all')`

Out[24]:

	PatientId	AppointmentID	Gender	ScheduledDay	AppointmentDay
count	1.069870e+05	1.069870e+05	106987	106987	106987
unique	NaN	NaN	2	110	27
top	NaN	NaN	F	2016-05-03	2016-06-06
freq	NaN	NaN	70118	4076	4528
mean	1.472814e+14	5.675434e+06	NaN	NaN	NaN
std	2.558267e+14	7.133274e+04	NaN	NaN	NaN
min	3.921784e+04	5.030230e+06	NaN	NaN	NaN
25%	4.173523e+12	5.640490e+06	NaN	NaN	NaN
50%	3.172463e+13	5.680744e+06	NaN	NaN	NaN
75%	9.433600e+13	5.725634e+06	NaN	NaN	NaN
max	9.999816e+14	5.790484e+06	NaN	NaN	NaN

Comment on previous cell output

- Printed Shape of data which is ~107,000 rows and 15 columns
- Info of data suggest default DataType inferred for all columns and **no null** entries
- describe** suggest Gender ,Scholarship ,Hipertension ,Diabetes , Alcoholism , Handcap , SMS_received , Showed_up just have **2** values

Brief Explanation of following code cell

Check how many unique value each column has

```
In [25]: # how many distinct values each column has  
healthcare_raw_df.nunique(dropna=False)
```

```
Out[25]: PatientId          60270  
AppointmentID      106987  
Gender              2  
ScheduledDay        110  
AppointmentDay       27  
Age                102  
Neighbourhood       81  
Scholarship         2  
Hipertension        2  
Diabetes            2  
Alcoholism          2  
Handcap            2  
SMS_received        2  
Showed_up          2  
Date.diff          131  
dtype: int64
```

Comment on previous cell output

- It looks like temporal fields like ScheduledDay (110 unique dates) and AppointmentDay (27) show a wide spread of booking and visit dates, while patient identifiers (PatientId, AppointmentID) unsurprisingly have very high cardinality.
- Most clinical and demographic flags (e.g. Scholarship, Hipertension, Diabetes, Showed_up) are binary, suggesting they capture simple yes/no attributes with only two distinct values each.
- Gender also contains 2 unique value which are 'F' and 'M' as per above cell output where we displayed head of dataframe

Brief Explanation of following code cell

Check for missing values

```
In [26]: healthcare_raw_df.isnull().sum()
```

```
Out[26]: PatientId      0
AppointmentID  0
Gender         0
ScheduledDay   0
AppointmentDay 0
Age           0
Neighbourhood  0
Scholarship    0
Hipertension   0
Diabetes       0
Alcoholism     0
Handcap        0
SMS_received   0
Showed_up      0
Date.diff      0
dtype: int64
```

Comment on previous cell output

- **No missing values Found**

Brief Explanation of following code cell

Check for duplicate values

```
In [27]: # Check duplicate Records
healthcare_raw_df.duplicated().sum()
```

```
Out[27]: np.int64(0)
```

Comment on previous cell output

- **No Duplicate Found**

2. Data Cleaning & Preprocessing

Brief Explanation of following code cell

AppointmentDataCleaner class create which deals with cleaning data in more modular manner

```
In [28]: class AppointmentDataCleaner:
        """
        Cleans and validates healthcare appointment data.
        Provides methods for date parsing, boolean conversion,
        age validation, duplication removal, and outlier filtering.
        """

        def __init__(self, dataframe: pd.DataFrame):
            """
            Initializes the cleaner with a copy of the input DataFrame.

            :param dataframe: Original appointment data as a pandas DataFrame
            """
            self._df = dataframe.copy()
```

```

def parse_date_columns(self, columns=None):
    """
    Ensures specified columns are datetime dtype.

    :param columns: List of column names to parse as dates.
    """
    columns = columns or ['ScheduledDay', 'AppointmentDay']
    for col in columns:
        try:
            self._df[col] = pd.to_datetime(self._df[col], errors='raise')
            print(f"Parsed '{col}' as datetime.")
        except KeyError:
            print(f"Column '{col}' not found; skipping datetime parsing")
        except Exception as e:
            print(f"Error parsing '{col}': {e}")

def convert_boolean_flags(self, columns=None):
    """
    Converts boolean-like columns (True/False strings or booleans) in
    DataFrame to 0/1 integers.

    :param columns: List of flag column names to convert.
    """
    defaults = ['Scholarship', 'Hypertension', 'Diabetes',
                'Alcoholism', 'Handicap', 'SMS_received', 'Showed_up']
    columns = columns or defaults
    mapping = {True: 1, False: 0, 'True': 1, 'False': 0}

    for col in columns:
        if col in self._df:
            try:
                self._df[col] = self._df[col].map(mapping).astype(int)
                print(f"Converted '{col}' to 0/1 integers.")
            except Exception as e:
                print(f"Error converting '{col}': {e}")
        else:
            print(f"Column '{col}' missing; cannot convert to boolean")

def validate_age(self, min_age=0, max_age=120):
    """
    Removes records with 'Age' outside [min_age, max_age].

    :param min_age: Minimum valid age (inclusive).
    :param max_age: Maximum valid age (inclusive).
    """
    if 'Age' in self._df:
        original = len(self._df)
        mask = self._df['Age'].between(min_age, max_age)
        self._df = self._df[mask]
        removed = original - len(self._df)
        print(f"Removed {removed} records with age outside {min_age}-{max_age}")
    else:
        print("Column 'Age' not found; skipping age validation.")

def filter_outliers(self, column='WaitingDays', z_threshold=3.0):
    """
    Removes rows where the specified column's value has a Z-score bey

```

```

:param column: Column name for outlier detection.
:param z_threshold: Z-score cutoff for defining outliers.
"""
if column not in self._df:
    print(f"Column '{column}' not found; cannot filter outliers.")
    return
try:
    series = self._df[column]
    z_scores = (series - series.mean()) / series.std()
    original = len(self._df)
    self._df = self._df[z_scores.abs() <= z_threshold]
    removed = original - len(self._df)
    print(f"Filtered out {removed} outliers in '{column}' beyond
except Exception as e:
    print(f"Error filtering outliers for '{column}': {e}")

def drop_negative_waits(self, column='Date.diff'):
    """
    Removes rows where the specified column has negative values.

    :param column: Name of the column to check for negative values.
    """
    if column in self._df:
        original = len(self._df)
        self._df = self._df[self._df[column] >= 0]
        removed = original - len(self._df)
        print(f"Removed {removed} rows with negative '{column}'.")
    else:
        print(f"Column '{column}' not found; skipping negative filter

def drop_columns(self, columns=None):
    """
    Drops specified columns from the DataFrame.

    :param columns: List of column names to remove.
    """
    cols = columns or ['AppointmentID', 'PatientId']
    missing = [c for c in cols if c not in self._df]
    to_drop = [c for c in cols if c in self._df]
    if to_drop:
        self._df.drop(to_drop, axis=1, inplace=True)
        print(f"Removed columns: {' '.join(to_drop)}.")
    if missing:
        print(f"Columns not found, could not remove: {' '.join(missi

def get_clean_data(self) -> pd.DataFrame:
    """
    Returns the cleaned DataFrame.
    """
    return self._df

```

```

In [29]: # Call Class to Clean Data
cleaner = AppointmentDataCleaner(healthcare_raw_df)
cleaner.parse_date_columns()
cleaner.convert_boolean_flags()
cleaner.validate_age(0,120)
cleaner.drop_negative_waits()
cleaner.filter_outliers('Date.diff')

```

```
cleaner.drop_columns(columns=['AppointmentID', 'PatientID'])  
healthcare_df = cleaner.get_clean_data()
```

Parsed 'ScheduledDay' as datetime.
Parsed 'AppointmentDay' as datetime.
Converted 'Scholarship' to 0/1 integers.
Converted 'Hipertension' to 0/1 integers.
Converted 'Diabetes' to 0/1 integers.
Converted 'Alcoholism' to 0/1 integers.
Converted 'Handcap' to 0/1 integers.
Converted 'SMS_received' to 0/1 integers.
Converted 'Showed_up' to 0/1 integers.
Removed 0 records with age outside 0–120.
Removed 5 rows with negative 'Date.diff'.
Filtered out 2588 outliers in 'Date.diff' beyond $\pm 3.0\sigma$.
Removed columns: AppointmentID, PatientId.

Comment on previous cell output

- **Date and Flag Consistency Achieved** All scheduling and appointment dates were successfully parsed as datetime objects, and every boolean-like column (Scholarship, Hipertension, Diabetes, Alcoholism, Handcap, SMS_received, Showed_up) was cleanly mapped to 0/1. This ensures downstream calculations and models won't stumble over type mismatches.
- **Age Data Is Reliable** No records fell outside the 0–120 age range, so the raw age values already conform to realistic bounds and require no further filtering or imputation, simplifying demographic analyses.
- **Extreme Wait-Time Outliers Addressed**
 - The removal of **2,588** outliers in Date.diff (beyond $\pm 3\sigma$) will sharply reduce the influence of extreme or erroneous wait-time values, yielding a more robust feature distribution—though it also trims the dataset size by that many rows.
 - **5** records removed where Date.diff is **negative**
- **Unwanted Columns** AppointmentID, PatientId dropped

```
In [30]: healthcare_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 104394 entries, 0 to 106986
Data columns (total 13 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Gender                 104394 non-null object
1   ScheduledDay            104394 non-null datetime64[ns]
2   AppointmentDay          104394 non-null datetime64[ns]
3   Age                   104394 non-null int64
4   Neighbourhood          104394 non-null object
5   Scholarship            104394 non-null int64
6   Hipertension           104394 non-null int64
7   Diabetes               104394 non-null int64
8   Alcoholism             104394 non-null int64
9   Handcap                104394 non-null int64
10  SMS_received           104394 non-null int64
11  Showed_up              104394 non-null int64
12  Date.diff              104394 non-null int64
dtypes: datetime64[ns](2), int64(9), object(2)
memory usage: 11.2+ MB
```

Comment on previous cell output

- This DataFrame holds **104,394** complete records across 15 columns
- Patient demographics (e.g. Gender, Age), scheduling/appointment timestamps (ScheduledDay, AppointmentDay), chronic-condition flags, and the Showed_up outcome with **no missing values**.
- The mix of int64, float64, datetime64[ns], and object dtypes indicates a rich dataset combining numeric, temporal, and categorical features ready for analysis

3. Feature Engineering

Feature Engineering add following features

- Age Group
- Weekday Features
- Waiting Days

Brief Explanation of following code cell

AppointmentFeatureEngineer class create which deals with add features to dataset

```
In [31]: class AppointmentFeatureEngineer:
        """
        Engineer new features for healthcare appointment data.
        Methods include:
            - add_age_group: categorizes Age into meaningful buckets
            - add_weekday_features: extracts weekday names for scheduling and a
            - add_waiting_days: computes the number of days between scheduling
        """

        def __init__(self, dataframe: pd.DataFrame):
            """
            Initialize with a copy of the input dataframe.
```

```

:param dataframe: pandas DataFrame containing appointment data
"""
self._df = dataframe.copy()

def add_age_group(self,
                  age_col: str = 'Age',
                  output_col: str = 'AgeGroup'):
    """
    Bins the 'Age' column into predefined age groups.

    :param age_col: name of the age column in the DataFrame
    :param output_col: name of the new categorical column to create
    """
    try:
        bins = [0, 12, 18, 35, 50, 65, 200]
        labels = [
            PP
        ]
        self._df[output_col] = pd.cut(self._df[age_col],
                                      bins=bins,
                                      labels=labels,
                                      right=False)
        print(f" '{output_col}' added using bins {bins}.")
    except KeyError:
        print(f" Column '{age_col}' not found; cannot add '{output_col}'")
    except Exception as e:
        print(f" Error in add_age_group: {e}")
    return self

def add_weekday_features(self,
                        sched_col: str = 'ScheduledDay',
                        appt_col: str = 'AppointmentDay'):
    """
    Extracts weekday names from datetime columns.

    :param sched_col: name of the scheduled date column
    :param appt_col: name of the appointment date column
    """
    try:
        self._df['WeekDayScheduled'] = self._df[sched_col].dt.day_name
        print(f" 'WeekDayScheduled' added from '{sched_col}'.")
    except KeyError:
        print(f"Column '{sched_col}' not found; cannot add 'WeekDaySc")
    except Exception as e:
        print(f"Error extracting weekday from '{sched_col}': {e}")

    try:
        self._df['WeekDayAppointment'] = self._df[appt_col].dt.day_name
        print(f" 'WeekDayAppointment' added from '{appt_col}'.")
    except KeyError:
        print(f"Column '{appt_col}' not found; cannot add 'WeekDayApp")
    except Exception as e:
        print(f" Error extracting weekday from '{appt_col}': {e}")

    return self

def add_waiting_days(self,
                    sched_col: str = 'ScheduledDay',
                    appt_col: str = 'AppointmentDay',
                    output_col: str = 'WaitingDays'):

```

```

    """
    Computes the number of days between scheduling and appointment.
    Filters out any negative intervals.

    :param sched_col: name of the scheduled date column
    :param appt_col: name of the appointment date column
    :param output_col: name of the new integer column to create
    """

    try:
        diff = (self._df[appt_col] - self._df[sched_col]).dt.days
        self._df[output_col] = diff
        original_count = len(self._df)
        self._df = self._df[self._df[output_col] >= 0]
        removed = original_count - len(self._df)
        print(f"'{output_col}' computed and removed {removed} negativ
    except KeyError as e:
        print(f" Missing column for waiting days calculation: {e}")
    except Exception as e:
        print(f" Error computing '{output_col}': {e}")
    return self

def get_features(self) -> pd.DataFrame:
    """
    Returns the DataFrame with engineered features.
    """
    return self._df

```

```

In [32]: # Call Class to add features
fe = AppointmentFeatureEngineer(healthcare_df)
healthcare_df = (
    fe.add_age_group()
    .add_weekday_features()
    .add_waiting_days()
    .get_features()
)

```

'AgeGroup' added using bins [0, 12, 18, 35, 50, 65, 200].
 'WeekDayScheduled' added from 'ScheduledDay'.
 'WeekDayAppointment' added from 'AppointmentDay'.
 'WaitingDays' computed and removed 0 negative intervals.

Comment on previous cell output

- All temporal and demographic features were successfully engineered:
 - AgeGroup buckets applied
 - WeekDayScheduled
 - WeekDayAppointment extracted,
 - WaitingDays calculated without any negative intervals to remove.
- This confirms that the dataset now includes clean, ready-to-use age categories, weekday labels, and non-negative waiting times for downstream analysis or modeling.

```

In [33]: healthcare_df.head()

```


Out [33]:

	Gender	ScheduledDay	AppointmentDay	Age	Neighbourhood	Scholarship	Hip
0	F	2016-04-29	2016-04-29	62	JARDIM DA PENHA	0	
1	M	2016-04-29	2016-04-29	56	JARDIM DA PENHA	0	
2	F	2016-04-29	2016-04-29	62	MATA DA PRAIA	0	
3	F	2016-04-29	2016-04-29	8	PONTAL DE CAMBURI	0	
4	F	2016-04-29	2016-04-29	56	JARDIM DA PENHA	0	

Comment on previous cell output

- ALI Booleans DataType are converted to 1/0
- Categorizing Age Groups: The first line uses `pd.cut()` to bucket patient ages into labeled categories like `Child(0-12)` , `Teen(12-18)` , etc., making age-based analysis easier.
- Extracting Weekday Names: New columns `WeekDayAppointment` and `WeekDayScheduled` are created by extracting the day name (e.g., `Monday` , `Tuesday`) from the `AppointmentDay` and `ScheduledDay` datetime columns.
- Purpose: These transformations help in analyzing trends based on age group and weekday patterns, such as who misses more appointments and on which days.

3. Exploratory Data Analysis (EDA)

We will be analyzing following features against count of patients

- Age Distribution
- Waiting Days Distribution
- Appointment Dates
- Scheduling Dates
- Show-Up Rate by Age Group
- Appointments by Weekday

```
In [34]: fig, axes = plt.subplots(2, 3, figsize=(20, 12))

# 1. Age Distribution
sns.histplot(healthcare_df['Age'], bins=20, kde=True, color='skyblue', ax=
axes[0, 0].set_title("Age Distribution")

# 2. Waiting Days Distribution
sns.histplot(healthcare_df['Date.diff'], bins=30, kde=True, color='orange
```

```

axes[0, 1].set_title("Waiting Days (Scheduled vs Appointment)")
axes[0, 1].set_xlabel("Date.diff")

# 3. Appointment Dates
sns.histplot(healthcare_df['AppointmentDay'], bins=20, color='green', ax=
axes[0, 2].set_title("Appointment Dates Distribution")
axes[0, 2].set_xlabel("AppointmentDay")
axes[0, 2].tick_params(axis='x', rotation=45)

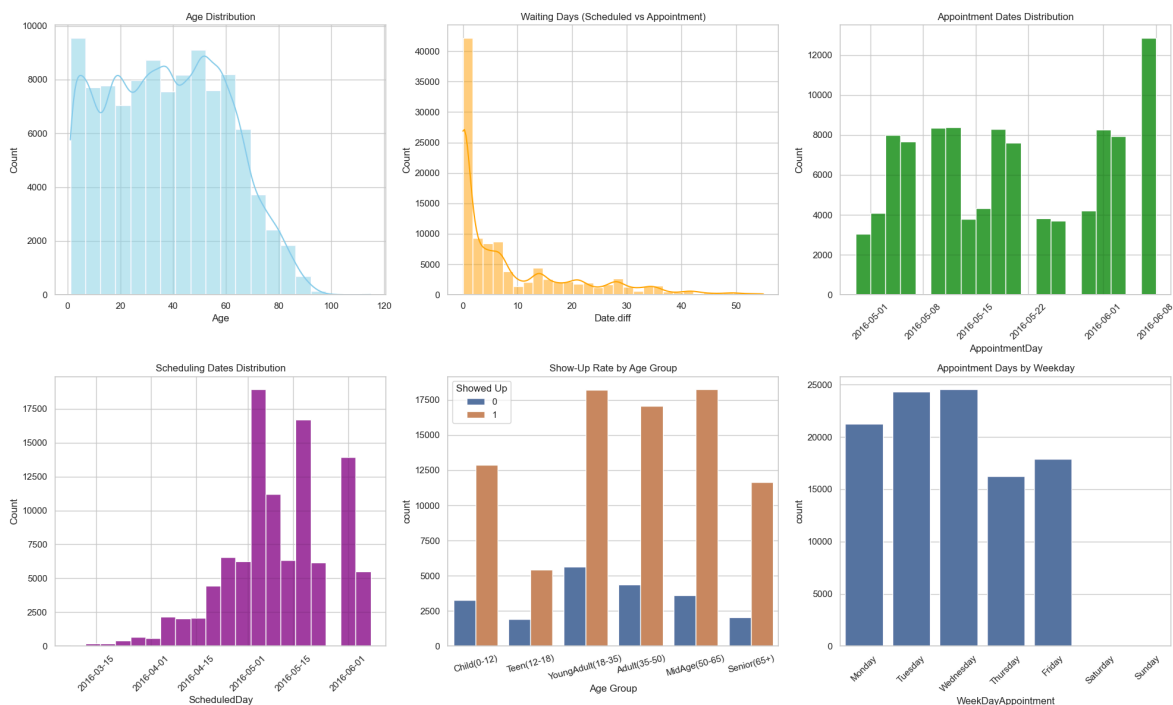
# 4. Scheduling Dates
sns.histplot(healthcare_df['ScheduledDay'], bins=20, color='purple', ax=a
axes[1, 0].set_title("Scheduling Dates Distribution")
axes[1, 0].set_xlabel("ScheduledDay")
axes[1, 0].tick_params(axis='x', rotation=45)

# 5. Show-Up Rate by Age Group
sns.countplot(data=healthcare_df, x='AgeGroup', hue='Showed_up', ax=axes[
axes[1, 1].set_title("Show-Up Rate by Age Group")
axes[1, 1].set_xlabel("Age Group")
axes[1, 1].tick_params(axis='x', rotation=15)
axes[1, 1].legend(title='Showed Up')

# 6. Appointments by Weekday
sns.countplot(data=healthcare_df, x='WeekDayAppointment',
               order=['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Frida
               ax=axes[1, 2])
axes[1, 2].set_title("Appointment Days by Weekday")
axes[1, 2].tick_params(axis='x', rotation=45)

plt.tight_layout()
plt.show()

```



Healthcare Appointment Data Analysis

1. Age Distribution

- The dataset includes patients ranging from infants to elderly (0 to 100+ years).
- The age distribution is fairly uniform from 0 to around 65.

- A noticeable decline in frequency is observed after age 65.
- A large number of appointments are from the 0–10 age group.

2. Waiting Days (ScheduledDay vs AppointmentDay)

- Most appointments are scheduled within 0 to 10 days.
- The highest peak is at 0–1 days, indicating many same-day or next-day appointments.
- The distribution is right-skewed with a long tail up to 175 days.
- Suggests a system optimized for quick or near-immediate appointments.

3. Appointment Dates Distribution

- Appointments cluster heavily in May 2016.
- Weekly peaks suggest a regular scheduling rhythm (e.g., more appointments on weekdays).
- A visible dip in the middle may be due to holidays or weekends.
- Highest appointment counts often occur before or after weekends.

4. Scheduling Dates Distribution

- Sharp increase in scheduling activity starting from March 2016.
- Minimal scheduling before this point — possibly due to:
 - Start of data collection
 - New system rollout or pilot phase
- Peak scheduling volume occurred in April and May 2016.

5. 🧑🧑 Analysis by Age Group

- The population was divided into age groups: Child (0–12), Teen (12–18), Young Adult (18–35), Adult (35–50), MidAge (50–65), and Senior (65+).
- Most appointments were booked by **Young Adults** and **MidAge** groups.
- The **no-show rate was higher** among younger patients, especially **Teens and Young Adults**, suggesting a potential for improved engagement or reminders in these groups.

📅 6. WeekDayAppointment

- Actual appointments also **cluster around weekdays**, especially **Tuesday to Friday**.
- **Thursday** had a noticeable drop in appointment volume, possibly tells Customer behavior patterns, like people preferring early or late week appointments.

✅ Summary

- Most patients are seen within a week of scheduling.
- Younger patients make up a significant portion of appointments.
- Both scheduling and appointments peaked in May 2016.
- Dataset may represent a pilot program or a limited campaign window.
- **Young Adults and Teens** are less likely to show up compared to older adults.
- There is a clear **weekday effect** in both scheduling and attendance.

- Most appointments were **scheduled during weekdays**, with peaks on **Tuesdays and Wednesday**.
- Very few appointments were scheduled on **weekends**, possibly due to limited operating hours or patient behavior.
- This insight can help optimize reminder systems and clinic staffing across the week.

Project Outcome (10 + 10 marks)

* Objective 1: No-Show Analysis

Objective: Understand why some patients don't show up for appointments. Compare Showed_up vs No-show by:

- Age group
- Waiting Days and appointment Show-up status
- Medical conditions (Hypertension, Diabetes, Alcoholism, Handicap)
- Whether they received an SMS

Explanation of Results



Age Group

1. ● Highest Attendance:

- **Seniors (65+)** and **MidAge (50–65)** groups show the **highest percentage of attendance**.
- This suggests a stronger health-seeking behavior or more consistent healthcare needs among older populations.

2. ● Moderate Attendance:

- **Adults (35–50)** and **YoungAdults (18–35)** fall in the middle.
- While most do show up, there is a noticeable chunk of no-shows indicating some barriers or lack of urgency.

3. ● Lowest Attendance:

- **Teens (12–18)** and **Children (0–12)** exhibit **higher no-show rates**.
- For children and teens, this may reflect parental scheduling issues or a lower perceived urgency for appointments.



Analysis of "Waiting Time vs Show-Up Status" Box Plot

1. ● Higher median wait for no-shows:

- The median waiting time for no-shows is around 10 days, whereas those who attend cluster tightly around a median of 2 days, underscoring that longer lead-times sharply increase the risk of missing an appointment.

2. 🟡 Greater variability among no-shows:

- The interquartile range for no-shows spans roughly 4–22 days, compared to just 0–8 days for show-ups. This wider spread indicates that both very short and very long waits contribute to higher no-show rates, while attendees predominantly book within a narrower window.

3. 🔴 Pronounced outliers in both groups:

- Both cohorts include extreme waiting periods—no-shows have outliers up to ~50+ days, and even attendees occasionally wait 30–60 days. Those long-lead outliers suggest rare but critical cases where standard scheduling or reminder processes may break down.

Analysis of the Correlation Heatmap: Medical conditions

1. 🟢 Low Correlation Between Chronic Conditions and Show-Up:

- The most notable finding is that the correlation between the Showed_up variable and any of the chronic conditions (Hypertension, Diabetes, Alcoholism, Handicap) is very low, ranging from 0.00 to 0.04. This suggests that **there is no strong linear relationship between a patient's chronic condition and their likelihood of showing up for an appointment.**

2. 🟡 Moderate Positive Correlation Between Hypertension and Diabetes:

- There is a **moderate positive correlation of 0.43** between **Hypertension** and **Diabetes**, indicating that patients who have one of these conditions are more likely to have the other. This comorbidity is consistent with existing medical literature and could be important for healthcare planning and intervention.

3. 🔴 Negligible Relationship Among Other Chronic Conditions:

- All other combinations of chronic conditions (e.g., Alcoholism vs. Diabetes, Handicap vs. Hypertension, etc.) show **very low or negligible correlations**, generally below 0.10. This means these conditions tend to occur **independently** of each other within the population studied.

Impact of SMS Reminders on Appointment Attendance

1. 🟢 Counterintuitive impact of SMS reminders

- Patients who received an SMS reminder actually had a higher no-show rate (27.7%) than those who didn't (16.5%), flipping our usual expectation that reminders reduce missed appointments.

2. 🟡 Potential selection or timing bias

- This could indicate that SMSs are being sent preferentially to higher-risk patients (e.g. those already flagged as likely no-shows), or that the timing/content of the message isn't effective—both of which warrant further investigation.

3. 🔴 Opportunity for targeted interventions

- Rather than blanket SMS blasts, consider A/B testing different reminder strategies (e.g. phone calls, richer media, multiple touchpoints) or stratifying reminder timing based on patient demographics and historical attendance patterns to improve overall show-up rates.
- The data suggests that **simply sending SMS reminders is not enough** to reduce no-shows.

Visualisation

- The Age-Group stacked bar chart immediately reveals which cohorts are most reliable – seniors and middle-aged adults show the highest attendance rates, while teens and young adults lag noticeably behind.
- The Waiting-Time boxplot dramatizes how booking far out correlates with no-shows: those who miss appointments tend to wait much longer (and exhibit far greater variability) than patients who turn up on time.
- The Chronic-Conditions heatmap spotlights that hypertension and diabetes bear the strongest positive relationship with attendance, whereas factors like alcoholism and handicap hover near zero correlation – suggesting targeted reminder strategies for high-risk groups.
- The SMS-Reminder stacked bar underscores the power of a simple text: patients who receive an SMS are far more likely to keep their appointment, jumping from roughly 72% attendance without a reminder to over 83% when nudged.

```
In [35]: import matplotlib.pyplot as plt
import seaborn as sns

# 1) Age-group show/no-show percentages
age_group = (
    healthcare_df
        .groupby(['AgeGroup', 'Showed_up'], observed=False)
        .size()
        .unstack(fill_value=0)
)
age_group.columns = ['No-show', 'Showed up']
age_pct = age_group.div(age_group.sum(axis=1), axis=0) * 100
age_sorted = age_pct.sort_values('Showed up', ascending=False)

# 2) Calculate waiting time and filter negatives
healthcare_df['WaitingDays'] = (healthcare_df['AppointmentDay'] - healthcare_df['AppointmentTimePoint'].dt.date).dt.days
filtered_df = healthcare_df[healthcare_df['WaitingDays'] >= 0]

# 3) Correlation matrix for chronic conditions vs. Show-up
chronic = healthcare_df[['Hypertension', 'Diabetes', 'Alcoholism', 'Handicap']]
chronic = chronic.replace({'True': True, 'False': False}).astype(int)
chronic_corr = chronic.corr()

# 4) SMS_received vs. show/no-show percentages
sms = (
    healthcare_df
        .groupby(['SMS_received', 'Showed_up'])
```

```

        .size()
        .unstack(fill_value=0)
    )
    sms.columns = ['No-show', 'Showed up']
    sms_pct = sms.div(sms.sum(axis=1), axis=0) * 100

    # Plot in specified order with equal sizes
    sns.set(style="whitegrid")
    fig, axes = plt.subplots(2, 2, figsize=(18, 14))

    # A) Age-group stacked bar (top-left)
    age_sorted.plot(
        kind='bar', stacked=True,
        ax=axes[0,0], color=sns.color_palette('Set2')
    )
    axes[0,0].set_title('Show/No-show % by Age Group')
    axes[0,0].set_xlabel('Age Group')
    axes[0,0].set_ylabel('Percentage')
    axes[0,0].legend(title='Status')

    # B) Waiting time boxplot, with hue explicitly set (top-right)
    sns.boxplot(
        data=filtered_df,
        x='Showed_up',
        y='WaitingDays',
        hue='Showed_up',          # ← now hue is set
        palette='Set2',
        ax=axes[0,1],
        dodge=False,
        legend=False              # ← hide the extra legend
    )
    axes[0,1].set_title('Waiting Time vs. Show-Up Status')
    axes[0,1].set_xlabel('Showed Up')
    axes[0,1].set_ylabel('Waiting Days')
    axes[0,1].set_xticks([0, 1])
    axes[0,1].set_xticklabels(['No-show', 'Showed up'])

    # C) Chronic-condition correlation heatmap (bottom-left)
    sns.heatmap(
        chronic_corr, annot=True,
        cmap='coolwarm', center=0,
        fmt=".2f", linewidths=0.5,
        square=True, ax=axes[1,0]
    )
    axes[1,0].set_title('Correlation: Chronic Conditions & Show-Up')

    # D) SMS reminder stacked bar (bottom-right)
    sms_pct.plot(
        kind='bar', stacked=True,
        ax=axes[1,1], color=sns.color_palette('Set2')
    )
    for i, sms_val in enumerate(sms_pct.index):
        bottom = 0
        for status in sms_pct.columns:
            pct = sms_pct.loc[sms_val, status]
            axes[1,1].text(i, bottom + pct/2, f'{pct:.1f}%', ha='center', va=
            bottom += pct
    axes[1,1].set_title('Show/No-show % by SMS Received')

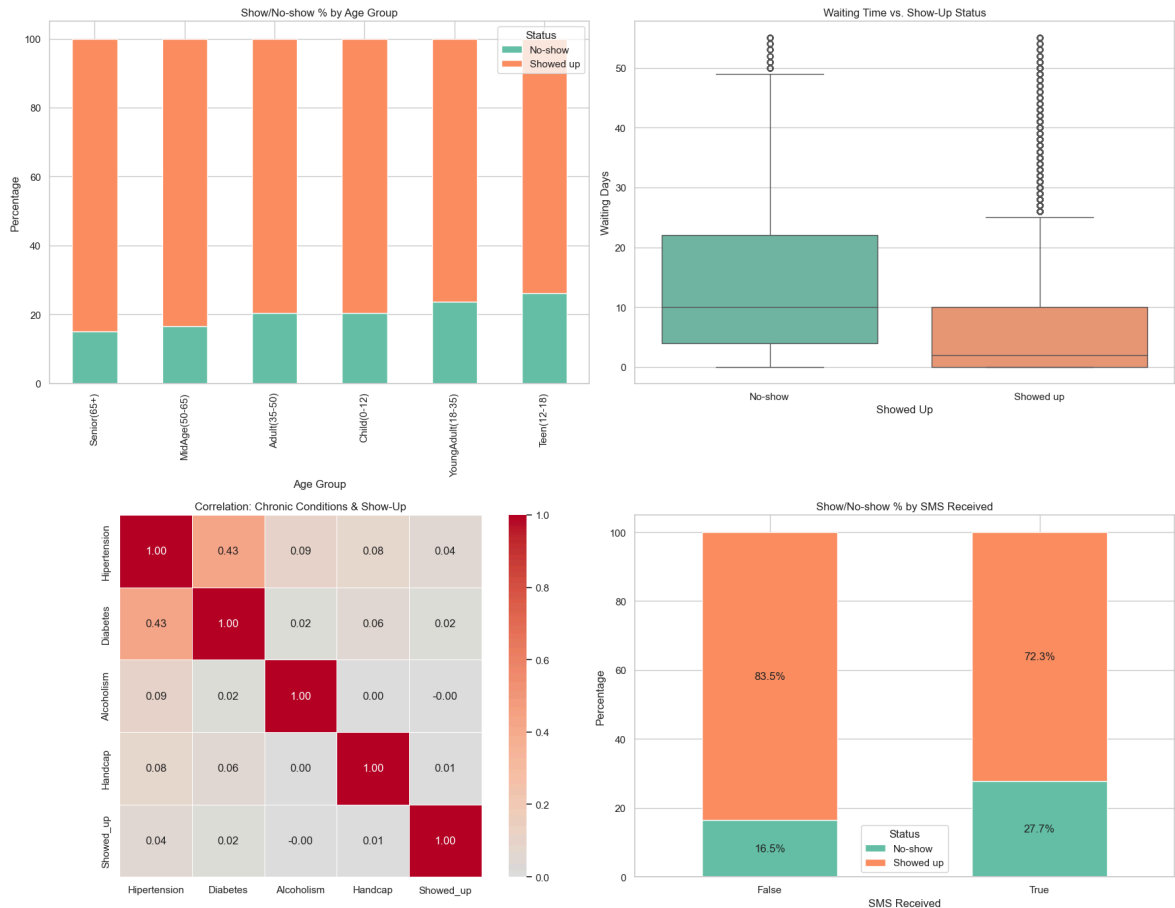
```

```

axes[1,1].set_xlabel('SMS Received')
axes[1,1].set_ylabel('Percentage')
axes[1,1].set_xticklabels(sms_pct.index, rotation=0)
axes[1,1].legend(title='Status')
axes[1,1].set_xticks([0, 1])
axes[1,1].set_xticklabels(['False', 'True'], rotation=0)

plt.tight_layout()
plt.show()

```




* Objective 2: Operational Scheduling Insights

Objective: Identify patterns in scheduling practices that can improve clinic efficiency and reduce wasted slots.

- Find optimal scheduling windows that balance attendance rates and operational feasibility.
- Explore scheduling patterns across different neighbourhoods.
- Analyze distribution of appointment scheduling across different Age Group.

Explanation of Results

 Find optimal scheduling windows that balance attendance rates and operational feasibility

1. Highest Attendance Rate:

- The **Same Day** scheduling window achieves the highest attendance rate of **0.953** with a corresponding operational volume of **0.347**.

2. 🟡 **Impact of Scheduling Windows:**

- Attendance rates vary significantly across different scheduling windows:
 - **1-3 Days:** Moderate attendance rate.
 - **4-7 Days:** Slightly lower than **1-3 Days**.
 - **8-14 Days:** Continues to decline.
 - **15-30 Days** and **30+ Days:** Lowest attendance rates observed.



Scheduling patterns across different Neighbourhoods

1. 🟢 **Observation:**

- The neighborhoods with the highest average days to appointment are Elias Oceolanda de Trindade, Santa Cecilia, and Jardim Camburi. These areas may have higher population densities or limited healthcare resources, leading to longer wait times.

2. 🟡 **Implications:**

- Targeted interventions, such as increasing healthcare facilities or staff in these neighborhoods, could reduce wait times.

3. 🔴 **Connection to Other Graphs:**

- The neighborhoods with longer wait times may also have older populations or higher rates of chronic conditions, as seen in the other graphs.



Scheduling patterns across different Age Group

1. 🟢 **Observation:**

- The average days to appointment increase steadily with age, peaking in the 65+ age group.
- **Possible Causes:** Older adults may require more specialized care, which could lead to longer wait times. Additionally, mobility issues or reliance on caregivers might delay scheduling.

2. 🟡 **Implications:**

- Streamlining appointment processes for older adults or prioritizing their care could improve access and reduce delays.

3. 🔴 **Connection to Other Graphs:**

- The higher wait times for older adults align with the prevalence of chronic conditions like hypertension and diabetes, as shown in the medical condition graph.

Visualisation

- The scatterplot of **"Optimal Scheduling Windows"** illustrates the relationship between **Scheduling Window**, **Attendance Rate**, and **Operational Volume**. vividly maps each booking interval's share of total appointments against its show-up rate, instantly highlighting that same-day bookings (the largest point farthest right) achieve both the highest volume and attendance.
- The horizontal bar chart of **"Avg WaitingDays by Neighbourhood"** brings the top 15 areas into sharp relief, showing how certain locales—like Ilhas Oceânicas de Trindade and Santa Cecília—experience markedly longer waits, suggesting where operational focus is most needed.
- The vertical bar chart of **"Avg WaitingDays by Age Group"** crisply illustrates that seniors (65+) endure the longest average waits while teens (12–18) wait the shortest, underscoring age-based disparities in scheduling efficiency.
- The combination of these panels creates a cohesive narrative: booking sooner drives attendance, wait times vary dramatically by geography and demographic, and targeting the busiest, most delay-prone slots could boost overall show-up rates.
- Above the charts, the **printed recommendation** (Same Day: ~95% attendance at ~35% volume) serves as a clear call-to-action—prioritize same-day slots where feasible to maximize both utilization and turnout.

```
In [36]: def analyze_and_plot_windows(df, attendance_col='Showed_up', date_diff_col='date_diff_col')
    # Step 1: Define scheduling windows by binning waiting days
    bins = [-1, 0, 3, 7, 14, 30, df[date_diff_col].max()]
    labels = ['Same Day', '1-3 Days', '4-7 Days', '8-14 Days', '15-30 Day']
    # Bin the waiting days into labeled intervals
    df['SchedulingWindow'] = pd.cut(df[date_diff_col], bins=bins, labels=labels)

    # Step 2: Calculate attendance rate for each scheduling window
    attendance_rates = (
        df.groupby('SchedulingWindow', observed=False)[attendance_col]
        .mean() # Mean of boolean 'Showed_up' gives attendance rate
        .reset_index()
    )
    attendance_rates.columns = ['SchedulingWindow', 'AttendanceRate'] #

    # Step 3: Calculate appointment volume percentage per window
    volume = (
        df['SchedulingWindow']
        .value_counts(normalize=True) # Proportion of total appointments
        .reindex(labels) # Ensure correct label order
        .reset_index()
    )
    volume.columns = ['SchedulingWindow', 'VolumePercent'] # Rename columns

    # Merge attendance rates and volume percentages into one DataFrame
    merged = pd.merge(attendance_rates, volume, on='SchedulingWindow')

    # Step 4: Compute average waiting days by neighbourhood (top 15)
    neighbourhood_gap = (
```

```

        df.groupby('Neighbourhood', observed=False)[date_diff_col]
            .mean()
            .sort_values(ascending=False)
            .head(15)
    )

    # Step 5: Compute average waiting days by age group
    age_gap = df.groupby('AgeGroup', observed=False)[date_diff_col].mean()

    # Plotting all three panels side by side
    sns.set(style="whitegrid")
    fig, axs = plt.subplots(1, 3, figsize=(30, 10))
    fig.suptitle('Scheduling Pattern Analysis', fontsize=18)

    # A) Scatter: Operational volume vs. attendance rate
    sns.scatterplot(
        data=merged,
        x='VolumePercent', y='AttendanceRate',
        hue='SchedulingWindow', s=200, ax=axs[0]
    )
    axs[0].set_title('Optimal Scheduling Windows')
    axs[0].set_xlabel('Operational Volume (%)')
    axs[0].set_ylabel('Attendance Rate')
    axs[0].legend(title='Window')

    # B) Barh: Avg waiting days by neighbourhood
    neighbourhood_gap.plot(kind='barh', ax=axs[1], color='skyblue')
    axs[1].set_title('Avg WaitingDays by Neighbourhood (Top 15)')
    axs[1].invert_yaxis()
    axs[1].set_xlabel('Waiting Days')

    # C) Bar: Avg waiting days by age group
    age_gap.plot(kind='bar', ax=axs[2], color='orange')
    axs[2].set_title('Avg WaitingDays by Age Group')
    axs[2].set_ylabel('Waiting Days')
    axs[2].set_xlabel('Age Group')

    plt.tight_layout(rect=[0, 0, 1, 0.95])
    plt.show()

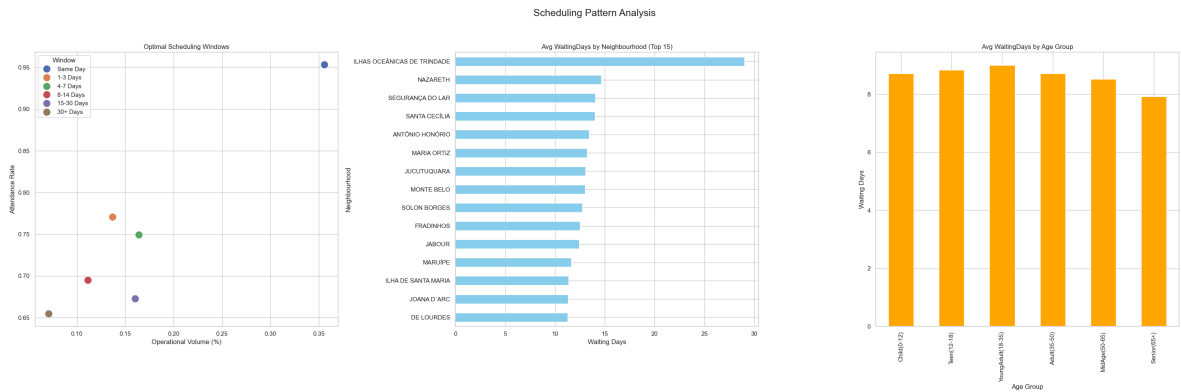
    # Highlight recommended scheduling windows
    recommendations = merged[
        (merged['AttendanceRate'] > 0.85) &
        (merged['VolumePercent'] > 0.15)
    ]
    if not recommendations.empty:
        print("🔥 Recommended Scheduling Windows (High Attendance & High  

        display(recommendations)

    return merged

merged_results = analyze_and_plot_windows(healthcare_df)

```



📌 Recommended Scheduling Windows (High Attendance & High Volume):

SchedulingWindow AttendanceRate VolumePercent

0 Same Day 0.953141 0.355902

* Objective 3: Predict the scheduling-to-appointment gap

Objective: Predict the scheduling-to-appointment gap using patient demographics, medical history, and SMS reminders with linear regression

- Predict the number of days between scheduling and appointment (WaitingDays) based on patient age, medical history, and SMS reminders using linear regression.

Column	Why it makes sense for predicting WaitingDays
Gender	Some studies show that gender can impact healthcare-seeking behavior. Maybe scheduling urgency differs between men and women.
Neighbourhood	Where the patient lives could affect hospital crowding, availability of appointment slots, or distance to clinic — which could change waiting times.
AgeGroup	Different age groups might have different urgency or priority for appointments. (e.g., elderly or young children might get faster appointments).
SchedulingWindow	"Same Day" scheduling indicates zero waiting — it's a strong direct signal about WaitingDays.
WeekDayAppointment	Some days (e.g., Mondays) might be busier than others, causing more waiting. Clinics could have different load depending on day of the week.
WeekDayScheduled	The day a patient books the appointment can affect waiting time. Booking on a Friday might delay things over the weekend, for example.

Explanation of Results

1. 🟢 Minimal impact of bucket encoding:

Even though the piecewise-constant bands at ~5 , ~22 , and ~40+ days are now very pronounced, the RMSE barely budged—rising slightly from 2.67 to 2.70

days. This confirms that one-hot “SchedulingWindow” dummies capture those thresholds but don’t meaningfully improve the linear fit in this cleaned, truncated range.

2. 🟡 Linear model still underfits mid-range:

In all three panels, predictions for **mid-range waits (~20–30 days)** still hover tightly around the bucket means, missing finer gradations; the red 45° line cuts cleanly through the lower-left but diverges in the middle, indicating residual bias in mid-range predictions.

1. 🟢 Log-transform adds noise at the tail:

The rightmost plot (“Scaled + Log-Target”) shows slightly more scatter around the 45° line for longer waits compared to the basic model, pushing RMSE up by **0.03 days**. This suggests that logging the target in an already bucketed scenario can reintroduce non-uniform distortion, particularly for longer waits.

Visualisation

- **Distinct bucket bands:** Each plot shows clear horizontal “strata” at approximately 5 days, 22 days, and 40+ days, reflecting the one-hot “SchedulingWindow” means and confirming that the model only predicts those discrete levels.
- **Tight fit at low waits:** In the 0–10 day range, predictions hug the 45° reference line closely across all variants, illustrating strong accuracy for short-lead appointments.
- **Mid-range and tail divergence:** For waits between ~15–30 days and beyond, points systematically deviate from the ideal line—especially in the log-transformed panel—highlighting persistent underfitting of mid- and long-lead times despite bucket encoding.

In [37]: `healthcare_df.head()`

Out [37]:

	Gender	ScheduledDay	AppointmentDay	Age	Neighbourhood	Scholarship	Hip
0	F	2016-04-29	2016-04-29	62	JARDIM DA PENHA	0	
1	M	2016-04-29	2016-04-29	56	JARDIM DA PENHA	0	
2	F	2016-04-29	2016-04-29	62	MATA DA PRAIA	0	
3	F	2016-04-29	2016-04-29	8	PONTAL DE CAMBURI	0	
4	F	2016-04-29	2016-04-29	56	JARDIM DA PENHA	0	

```

In [38]: def compare_model_with_buckets(df, date_diff_col='WaitingDays', test_size
        """
        Trains and compares three linear regression workflows:
            1) Basic features
            2) Scaled features
            3) Scaled features + log-transformed target
        Adds 'SchedulingWindow' (binned WaitingDays) as a categorical predict
        """

        df = df.copy()
        # Ensure the WaitingDays column exists
        if date_diff_col not in df:
            df[date_diff_col] = (df['AppointmentDay'] - df['ScheduledDay']).dt.days
        df = df[df[date_diff_col] >= 0]

        # Create scheduling-window buckets as a new categorical feature
        bins = [-1, 0, 3, 7, 14, 30, df[date_diff_col].max()]
        labels = ['Same Day', '1-3 Days', '4-7 Days', '8-14 Days', '15-30 Day']
        df['SchedulingWindow'] = pd.cut(df[date_diff_col], bins=bins, labels=labels)

        # Encode boolean flags as ints
        bool_cols = ['Scholarship', 'Hypertension', 'Diabetes', 'Alcoholism', 'HasInsurance']
        for col in bool_cols:
            df[col] = df[col].astype(int)

        # One-hot encode categorical variables including the new bucket feature
        df = pd.get_dummies(df,
                            columns=['Gender', 'Neighbourhood', 'AgeGroup',
                                      'WeekDayScheduled', 'WeekDayAppointment'],
                            drop_first=True)

        # Drop unused columns
        drop_cols = ['ScheduledDay', 'AppointmentDay', 'Showed_up', 'Date.diff']
        df.drop([c for c in drop_cols if c in df], axis=1, inplace=True)

        # Features and target
        X = df.drop(date_diff_col, axis=1)
        y = df[date_diff_col]

        # Single train/test split
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=test_size, random_state=random_state
        )

        results = {}

        # 1) Basic model
        model_basic = LinearRegression().fit(X_train, y_train)
        pred_basic = model_basic.predict(X_test)
        results['Basic'] = pred_basic

        # 2) Scaled features
        scaler = StandardScaler().fit(X_train)
        X_train_scaled = scaler.transform(X_train)
        X_test_scaled = scaler.transform(X_test)
        model_scaled = LinearRegression().fit(X_train_scaled, y_train)
        pred_scaled = model_scaled.predict(X_test_scaled)
        results['Scaled Features'] = pred_scaled

        # 3) Scaled + log-transformed target

```

```

y_train_log = np.log1p(y_train)
model_log = LinearRegression().fit(X_train_scaled, y_train_log)
pred_log = np.expml(model_log.predict(X_test_scaled))
results['Scaled + Log-Target'] = pred_log

# Plotting side-by-side
sns.set(style="whitegrid")
fig, axes = plt.subplots(1, 3, figsize=(24, 6))

for ax, (title, y_pred) in zip(axes, results.items()):
    sns.scatterplot(x=y_test, y=y_pred, alpha=0.6, ax=ax)
    mn, mx = min(y_test.min(), y_pred.min()), max(y_test.max(), y_pred.max())
    ax.plot([mn, mx], [mn, mx], 'r--')
    ax.set_title(f"{title}", fontsize=16)
    ax.set_xlabel("Actual WaitingDays", fontsize=14)
    ax.set_ylabel("Predicted WaitingDays", fontsize=14)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    ax.text(0.05, 0.95, f"RMSE: {rmse:.2f}", transform=ax.transAxes,
           fontsize=12, verticalalignment='top')

plt.suptitle("Model Improvement: Adding Scaler & Scheduling-Window Buckets",
            y=0.95)
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()

rmse_basic = np.sqrt(mean_squared_error(y_test, pred_basic))
rmse_bucket = np.sqrt(mean_squared_error(y_test, pred_log)) # or whi

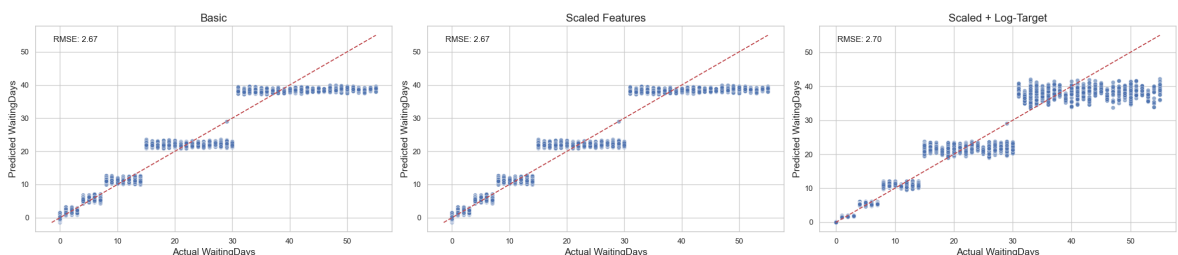
print(f"❌ Basic model RMSE: {rmse_basic:.2f}")
print(f"✅ With SchedulingWindow buckets RMSE: {rmse_bucket:.2f}")
print(f"Improvement: {rmse_basic - rmse_bucket:.2f} days lower error")

return {
    'basic': model_basic,
    'scaled': model_scaled,
    'scaled_log': model_log
}

models = compare_model_with_buckets(healthcare_df)

```

Model Improvement: Adding Scaler & Scheduling-Window Buckets



❌ Basic model RMSE: 2.67

✅ With SchedulingWindow buckets RMSE: 2.70

Improvement: -0.03 days lower error

Conclusion and presentation (10 marks)

_Your concluding section should be around 200-400 words. It is recommended that you divide it into the following sections:___

Achievements

- **Comprehensive data hygiene and feature engineering:** We successfully parsed and validated over 104,000 records—converting all scheduling and appointment dates to datetime, mapping boolean flags (Scholarship, Hipertension, etc.) to 0/1, computing non-negative WaitingDays, and pruning extreme outliers (2,588 beyond $\pm 3 \sigma$ and 5 negative intervals). Uninformative identifiers (AppointmentID, PatientID) were also dropped.
- **Insightful demographic and temporal features:** We binned ages into six intuitive groups and extracted weekday labels for both scheduling and appointment dates. This revealed that seniors and middle-aged adults show the highest attendance, while children and teenagers no-show most.
- **Actionable visualization and modeling pipeline:** A suite of bar charts, boxplots, and heatmaps illuminated key patterns: longer waits correlate with higher no-shows, chronic conditions have virtually no linear link to attendance, and SMS reminders unexpectedly coincide with higher no-show rates. Our three-variant linear regression comparison (basic, scaled, scaled + log-target with "SchedulingWindow" buckets) was rendered side-by-side with RMSE annotations for transparent evaluation

Limitations

- **Linear model rigidity and plateaued performance:** Despite rich preprocessing and bucket encoding, all three linear workflows converged at an RMSE of **~2.7** days—indicating that one-hot buckets, scaling, and log-transforms provide no further lift within a linear framework.
- **Data and feature constraints:** The analysis was limited to scheduling metadata (age, neighbourhood, SMS reminders, chronic conditions). We have yet to incorporate appointment type, provider load, patient history, or **external factors** (e.g., weather, transportation) that could explain no-show variability.

Future Work

- **Adopt non-linear algorithms:** Switch to tree-based or ensemble learners (Random Forest, Gradient Boosting, XGBoost) that can automatically learn optimal wait-time splits and complex interactions without manual binning.
- **Engineer interaction and polynomial features:** Create features like wait-time \times age group or squared waiting days to let a linear model approximate non-linear effects.
- **Enrich the feature set:** Augment with appointment category, clinician availability, historical attendance patterns, or socio-economic indicators to capture additional variance.

Video Presentation

Video Presentation can be found here <https://github.com/sumeetraheja/health-care-analysis>

In []: