Loading NHSRdatasets

Mairead Bermingham

17 June, 2022

Overview

In this lesson we will load the five datasets from the NHSRdatasets package. Next we will briefly view, explore and tabulate the NHS England accident and emergency attendances and admissions (ae_attendances) data set and save it to your 'RawData' folder. We will then examine the four-hour waiting time target performance for England as a whole, selecting a subset of the variables needed. We will then partitioned the data subset into training and testing data and save them to your working 'Data' folder, ready for downstream exploratory analysis.

Background

The data that you will be managing on the course are from the *NHSRdatasets* package. This package has been created to support skills development in the NHS-R community. It contains several free datasets, including:

- **NHS England accident and emergency attendances and admissions (ae_attendances).** Reported attendances, four-hour breaches and admissions for all A&E departments in England for the years 2016/17 through 2018/19 (Apr-Mar).
- **Hospital length of stay (LOS) data (LOS_model).** Artificially generated hospital data. Fictional patients at ten fictional hospitals, with LOS, age and date status data. Data were generated to learn generalised linear models (GLM) concepts, modelling either death or LOS.
- **Deaths registered weekly in England and Wales, provisional (ons_mortality).** Provisional counts of the number of deaths registered in England and Wales, by age, sex and region, in the latest weeks for which data are available.
- **Stranded patient (Patients flagged as having a greater than 7 day LOS) model (stranded_data).** This model is to be used as a machine learning classification model for supervised learning. The binary outcome is stranded vs not stranded patients.
- **Synthetic national early warning scores data (synthetic_news_data).** Synthetic NEWS data to show the results of the *NHSR_synpop* package. NEWS is a tool developed by the Royal College of Physicians which improves the detection and response to clinical deterioration in adult patients and is a key element of patient safety and improving patient outcomes. NEWS2 has been endorsed by NHS England and NHS Improvement – for use in acute and ambulance settings. These datasets have been synthetically generated. (Chris Mainey, 2021)

Load packages and data

Let's load the packages and data needed for this script.

```
library(NHSRdatasets)
library(tidyverse)
```

Figure 1: Illustration by Allison Horst

The *tidyverse* is a collection of R packages designed for data science introduced by Hadley Wickham and his team that "share an underlying design philosophy, grammar, and data structures" of tidy data.

```
library(here)
```

The *here* package enables easy file referencing in project-oriented workflows. In contrast to using `setwd()` function, which is fragile and dependent on the way you organise your files, here uses the top-level directory of a project to easily build paths to files.

```
library(knitr)
```

knitr is an R package that integrates code into text documents. Files can then be processed into a diverse array of document formats including pdfs, Word documents, etc.

```
library(scales)
```

The *scales* packages provides the internal scaling infrastructure to *ggplot2* and its functions allow programmers to customise the transformations, breaks, guides and palettes used in visualisations. *ggplot2* is a powerful package to draw graphics. It implements the grammar of graphics (and hence its name). *ggplot2* is included in, and loaded with the *tidyverse* package

```
library(lubridate)
```

lubridate provides tools that make it easier to manipulate dates in R. *lubridate* is part of Hadley's tidyverse ecosystem but is not loaded by the *tidyverse* package, which includes only what he thought were the core components. If you are going to be doing a lot of date manipulations, you need to load it separately.

```
library(caret)
```

The *caret* package (short for Classification And REgression Training) is a set of functions that attempt to streamline the process for creating predictive models. The package contains tools for data splitting, pre-processing, feature selection, model tuning using resampling, variable importance estimation amongst others.

NHSRdatasets

Here is the code to load the five NHSRdatasets from the *NHSRdatasets* R package.

```
#Load the ae_attendances data.
data(ae_attendances)
#Load the LOS_model data.
data(LOS_model)
#Load the ons_mortality data.
data(ons_mortality)
#Load the stranded_data.
data(stranded_data)
#Load the synthetic_news_data.
data(synthetic_news_data)
```

NHS England accident and emergency attendances and admissions (ae_attendances) data

In this course, we will be focus on the NHS England accident and emergency attendances and admissions (ae_attendances) data. If you are new to programming, I recommend using this data, as the data has been tidied to be easily usable within the tidyverse of packages. However, if you would like to stretch yourself for the course assignment, I recommend you can choose from one of the other four datasets contained within the *NHSRdatasets* package to manage on this course. You will not gain additional marks, but it will significantly enhance your learning on the course.

Let's have a look at the ae_attendances data

```
data(ae_attendances)
ae<-ae_attendances
class(ae)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

The tbl_df class is a subclass of data.frame. A data.frame is a table or a two-dimensional array-like structure in which each column contains values of one variable and each row contains one set of values from each column. The colloquial term "tibble" refers to a data frame that has the tbl_df class. Objects of class tbl_df have a class attribute of c("tbl_df", "tbl", "data.frame"). Tibble is the central data structure for the set of packages known as the tidyverse. It's best to think of tibbles as an updated and stylish version of the data.frame that exhibit some special behaviour, such as enhanced printing.

```
ae
```

```
## # A tibble: 12,765 x 6
##    period     org_code type  attendances breaches admissions
##    <date>     <fct>    <fct>       <dbl>    <dbl>      <dbl>
##  1 2017-03-01 RF4      1           21289     2879       5060
##  2 2017-03-01 RF4      2             813       22          0
##  3 2017-03-01 RF4      other        2850        6          0
##  4 2017-03-01 R1H      1           30210     5902       6943
##  5 2017-03-01 R1H      2             807       11          0
##  6 2017-03-01 R1H      other       11352      136          0
##  7 2017-03-01 AD913    other        4381        2          0
##  8 2017-03-01 RYX      other       19562      258          0
##  9 2017-03-01 RQM      1           17414     2030       3597
## 10 2017-03-01 RQM      other        7817       86          0
```

```
## # ... with 12,755 more rows
```

An overview of your data is returned when a tibble is printed to the RStudio console. You can see the ae_attendances tibble consists of 12,765 rows of data and six columns with different classes. You have one date variable, period, two character variables (or factors), org_code and type, and three numeric (double precision) variables, attendances, breaches and admissions. A factor is a variable used to categorise and store the data, having a limited number of different values. Factors are an important class for exploratory data analysis, visualisation, and statistical analysis with R.

The dataset contains:

- **period:** the month that this activity relates to, stored as a date (1st of each month).
- **org_code:** the Organisation data service (ODS) code for the organisation. The ODS code is a unique code created by the Organisation data service within NHS Digital, and used to identify organisations across health and social care. ODS codes are required in order to gain access to national systems like NHSmail and the Data Security and Protection Toolkit. If you want to know the organisation associated with a particular ODS code, you can look it up from the following address: https://odsportal.digital.nhs.uk/Organisation/Search. For example, the organisation associated with the ODS code 'AF003' is Parkway health centre.
- **type:** the Department Type for this activity, either
  - **1:** Emergency departments are a consultant led 24 hour service with full resuscitation facilities and designated accommodation for the reception of accident and emergency patients,
  - **2:** Consultant led mono specialty accident and emergency service (e.g. ophthalmology, dental) with designated accommodation for the reception of patients, or
  - **other:** Other type of A&E/minor injury activity with designated accommodation for the reception of accident and emergency patients. The department may be doctor led or nurse led and treats at least minor injuries and illnesses and can be routinely accessed without appointment. A service mainly or entirely appointment based (for example a GP Practice or Out-patient clinic) is excluded even though it may treat a number of patients with minor illness or injury. Excludes NHS walk-in centres.(National Health Service, 2020)
- **attendances:** the number of attendances for this department type at this organisation for this month.
- **breaches:** the number of attendances that breached the four hour target.
- **admissions:** the number of attendances that resulted in an admission to the hospital.(Chris Mainey, 2021)

Let's view at the ae_attendances data

The `glimpse()` function is from *tibble* package and is great to view the columns/variables in a data frame. It also shows data type and some of the data in the data frame in each row. The *tibble* package provides utilities for handling tibbles. The *tibble* package is loaded by the *tidyverse* package, as one of its core components.

```
glimpse(ae)
```

```
## Rows: 12,765
## Columns: 6
## $ period      <date> 2017-03-01, 2017-03-01, 2017-03-01, 2017-03-01, 2017-03-0~
## $ org_code    <fct> RF4, RF4, RF4, R1H, R1H, R1H, AD913, RYX, RQM, RQM, RJ6, R~
## $ type        <fct> 1, 2, other, 1, 2, other, other, other, 1, other, 1, other~
## $ attendances <dbl> 21289, 813, 2850, 30210, 807, 11352, 4381, 19562, 17414, 7~
## $ breaches    <dbl> 2879, 22, 6, 5902, 11, 136, 2, 258, 2030, 86, 1322, 140, 0~
## $ admissions  <dbl> 5060, 0, 0, 6943, 0, 0, 0, 0, 3597, 0, 2202, 0, 0, 0, 3360~
```

Here is the output of the 'glimpse()' function. It starts off with the number of rows and columns and each column in separate rows.

The `head()` function let's you get a look at top n rows of a data frame. By default it shows the first 6 rows in a data frame.

```
head(ae)
```

```
## # A tibble: 6 x 6
##   period     org_code type  attendances breaches admissions
##   <date>     <fct>    <fct>       <dbl>    <dbl>      <dbl>
## 1 2017-03-01 RF4      1           21289     2879       5060
## 2 2017-03-01 RF4      2             813       22          0
## 3 2017-03-01 RF4      other        2850        6          0
## 4 2017-03-01 R1H      1           30210     5902       6943
## 5 2017-03-01 R1H      2             807       11          0
## 6 2017-03-01 R1H      other       11352      136          0
```

We can specify the number of rows we want to see in a data frame with the argument "n".

```
head(ae, n=3)
```

```
## # A tibble: 3 x 6
##   period     org_code type  attendances breaches admissions
##   <date>     <fct>    <fct>       <dbl>    <dbl>      <dbl>
## 1 2017-03-01 RF4      1           21289     2879       5060
## 2 2017-03-01 RF4      2             813       22          0
## 3 2017-03-01 RF4      other        2850        6          0
```

In the example above, we used n=3 to look at the first three rows of the ae_attendances data frame.

The function `tail()` is the counterpart to `head()`. tail() let's you to take a look at the bottom n rows of a data frame. We can adjust the number of rows with the argument "n" as with the `head()` function.

```
tail(ae, n=4)
```

```
## # A tibble: 4 x 6
##   period     org_code type  attendances breaches admissions
##   <date>     <fct>    <fct>       <dbl>    <dbl>      <dbl>
## 1 2018-04-01 RA3      1            3825      476       1016
## 2 2018-04-01 AXG      other        2980       24          0
## 3 2018-04-01 NLX24    other        1538        0          0
## 4 2018-04-01 RA4      1            4388       82       1292
```

In the example above, we used n=4 to look at the last four rows of the ae_attendances data frame.

Missing data

It is common when collecting data for some entries to be missing, which can significantly impact any attempt to gain useful insight from data. Methods have been developed to handle missing data. The simplest of these is to discard any record which contains a missing entry. However, this can lead to such a small sample that it is not useful for obtaining reliable information. In addition to this, there may be reasons why certain demographic groups do not want to supply particular information, reducing the representativeness of the selected sample. We therefore need to check for missing data in the ae_attendances data.

'map()' is a function from the *purr* package for applying a function to each element of a list. The *purr* package enhances R's functional programming toolkit by providing a complete and consistent set of tools for working with functions and vectors. The 'is.na' function produces a matrix, consisting of logical values (i.e. TRUE or FALSE), whereby TRUE indicates a missing value.

Figure 2: Missing data (source: Matt Randall, 2020)

```
# Calculate how many NAs there are in each variable
ae %>%
  map(is.na) %>%
map(sum)
```

```
## $period
## [1] 0
##
## $org_code
## [1] 0
##
## $type
## [1] 0
##
## $attendances
## [1] 0
##
## $breaches
## [1] 0
##
## $admissions
## [1] 0
```

The data is complete. We do not need to worry about handling missing data. If you decide use another NHSRdataset it may not be as 'tidy' as the ae_attendances data.

Let's add an index link column to ae_attendances data

Separating data into training and testing sets is vital for evaluating data collection and analysis tools. Typically, when you divide a data set into a training set and a testing set, most of the data is used for training, and a smaller portion of the data is used for testing. To develop and evaluate your data capture tool, you will need to split the raw data into training and testing sets. We, therefore, need to add an index column to the raw data so we can link the partitioned data sets to the raw data if required in the future. We will use the `rowid_to_column()` function to convert row identities to a column named index.

```
ae <- rowid_to_column(ae, "index")
```

Let's tablulate the raw data for your report

The pipe operator denoted by symbol `%>%` is a special operational function available which allows us to pass the result of one function/argument to the other one in sequence. The the *dplyr* package `mutate_at()` function edits specific columns with a character vector or 'vars()' argument generates a list of columns. The function `comma()` from the *scales* package is useful for presenting numbers using commas to separate the thousands. This is always a good idea as it assists the reader in quickly determining the magnitude of the numbers we are looking at. As a matter of course, I recommend commas in plots (and tables) at all times.

```
ae %>%
  # Set the period column to show in month-year format
  mutate_at(vars(period), format, "%b-%y") %>%
  # Set the numeric columns to have a comma at the 1000's place
  mutate_at(vars(attendances, breaches, admissions), comma) %>%
  # Show the first 10 rows
  head(10) %>%
  # Format as a table
  kable()
```

index	period	org_code	type	attendances	breaches	admissions
1	Mar-17	RF4	1	21,289.0	2,879.0	5,060.0
2	Mar-17	RF4	2	813.0	22.0	0.0
3	Mar-17	RF4	other	2,850.0	6.0	0.0
4	Mar-17	R1H	1	30,210.0	5,902.0	6,943.0
5	Mar-17	R1H	2	807.0	11.0	0.0
6	Mar-17	R1H	other	11,352.0	136.0	0.0
7	Mar-17	AD913	other	4,381.0	2.0	0.0
8	Mar-17	RYX	other	19,562.0	258.0	0.0
9	Mar-17	RQM	1	17,414.0	2,030.0	3,597.0
10	Mar-17	RQM	other	7,817.0	86.0	0.0

Let's save the raw ae_attendances data to your 'RawData' folder

We will use the *here* package to build a path relative to the top-level directory to write the raw ae_attendances data to your 'RawData' folder. The goal of the *here* package is to enable easy file referencing in project-oriented workflows. In contrast to using `setwd()` function, which is fragile and dependent on the way you organise your files, here uses the top-level directory of a project to easily build paths to files.

```
write_csv(ae, here("RawData", "ae_attendances.csv"))
```

Selecting variables for your data capture tool

It really important point to note, is that you are not going to need the full ae_attendances dataset develop your data collection tool. For example if you are interested in exploring four hours performance for England as a whole you will only need the variables: index, period, attendances, breaches. Likewise, you may wish examine performance for the different types of department, we would also need the type variable. For the purpose of this lesson, we will focus on England as a whole.

Examining the four-hour waiting time target performance for England

We will now use the *dplyr* package `select()` function to select the required variables. The *dplyr* package is loaded by the *tidyverse* package, as one of its core components. dplyr provides a grammar of data manipulation, providing a consistent set of verbs that solve the most common data manipulation challenges.

```
ae<-ae %>% select(index, period, attendances, breaches)
```

Let's tabulate the subsetted ae_attendances data

```
ae %>%
  # set the period column to show in Month-Year format
  mutate_at(vars(period), format, "%b-%y") %>%
  # set the numeric columns to have a comma at the 1000's place
  mutate_at(vars(attendances, breaches), comma) %>%
  # show the first 10 rows
  head(10) %>%
  # format as a table
  kable()
```

index	period	attendances	breaches
1	Mar-17	21,289.0	2,879.0
2	Mar-17	813.0	22.0
3	Mar-17	2,850.0	6.0
4	Mar-17	30,210.0	5,902.0
5	Mar-17	807.0	11.0
6	Mar-17	11,352.0	136.0
7	Mar-17	4,381.0	2.0
8	Mar-17	19,562.0	258.0
9	Mar-17	17,414.0	2,030.0
10	Mar-17	7,817.0	86.0

Let's calculate monthly four hour waiting time target performance for England as a whole

We will now use the *dplyr* package `summarise_at()` function to get the `sum` of all attendances and breaches for England as a whole.

```
ENG_performance <- ae %>%
  group_by(period) %>%
  summarise_at(vars(attendances, breaches), sum) %>%
```

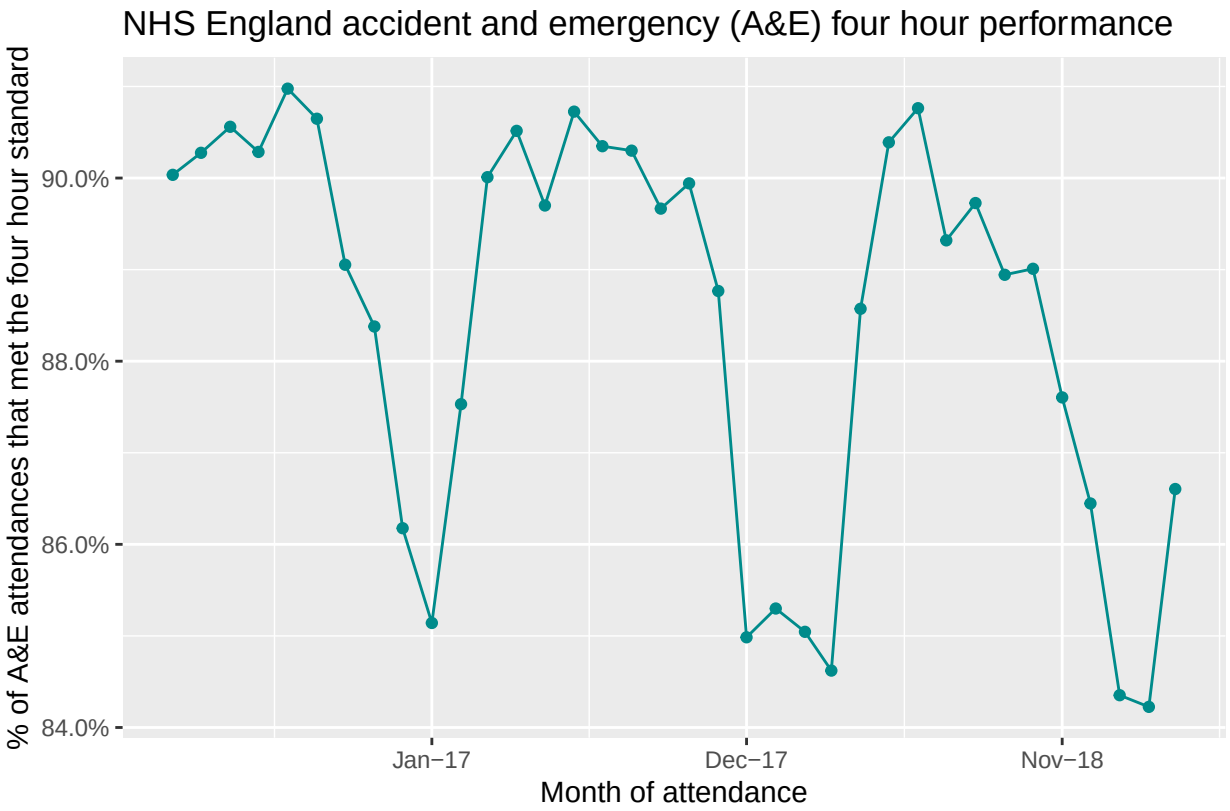
```
  mutate(performance = 1 - breaches / attendances)
glimpse(ENG_performance)
```

```
## Rows: 36
## Columns: 4
## $ period      <date> 2016-04-01, 2016-05-01, 2016-06-01, 2016-07-01, 2016-08-0~
## $ attendances <dbl> 1867781, 2070340, 1958802, 2079034, 1932901, 1952464, 2001~
## $ breaches    <dbl> 186122, 201329, 184912, 201973, 174419, 182597, 219137, 22~
## $ performance <dbl> 0.9003513, 0.9027556, 0.9055994, 0.9028525, 0.9097631, 0.9~
```

Let's visualise monthly four hour waiting time target performance before we save or raw data file.

ggplot2 is a powerful package to draw graphics. It implements the grammar of graphics (and hence its name). The *ggplot2* package is loaded by the *tidyverse* package, as one of its core components. The `ggplot()` function initialises a *ggplot* object. We will use the `geom_line()` to connect and geom_point() to represent the percentage of accident and emergency attendances that met the four hour standard (`y`), ordered by month of attendance(`x`). We will use the `labs()` function to add the `x` and `y` labels, plot title and caption.

```
ggplot(ENG_performance, aes(period, performance)) +
  geom_line(color = "darkcyan") +
  geom_point(color = "darkcyan") +
  scale_y_continuous(labels = percent) +
  scale_x_date(date_labels = "%b-%y", date_breaks = "11 month")+
  labs(x = "Month of attendance",
       y = "% of A&E attendances that met the four hour standard",
       title = "NHS England accident and emergency (A&E) four hour performance",
       caption = "Source: NHSRdatasets")
```



The National Health Service (NHS) is always under considerable pressure over the winter period as demand for services tends to increase significantly with the onset of cold weather and flu.(NHS Providers, 2022) You can clearly see that under the winter pressures where four hour waiting time target performance drops.

Let's select the ae_attendances data subset for further exploratory analysis

Based on this brief probe of the of the ae_attendances data, we have decide we wish to explore this data further for the development and evaluation your data capture tool.

Let's tabulate the subsetted ae_attendances data for your report

```
ae %>%
  # set the period column to show in Month-Year format
  mutate_at(vars(period), format, "%b-%y") %>%
  # set the numeric columns to have a comma at the 1000's place
  mutate_at(vars(attendances, breaches), comma) %>%
  # show the first 10 rows
  head(10) %>%
  # format as a table
  kable()
```

index	period	attendances	breaches
1	Mar-17	21,289.0	2,879.0
2	Mar-17	813.0	22.0
3	Mar-17	2,850.0	6.0
4	Mar-17	30,210.0	5,902.0

index	period	attendances	breaches
5	Mar-17	807.0	11.0
6	Mar-17	11,352.0	136.0
7	Mar-17	4,381.0	2.0
8	Mar-17	19,562.0	258.0
9	Mar-17	17,414.0	2,030.0
10	Mar-17	7,817.0	86.0

Let's save provisional subsetted ae_attendances data to the 'RawData' folder

Of note, when naming folders and files, it is important that you do so in a consistent, logical and predictable way means that information may be located, identified and retrieved by your and your colleagues, as quickly and easily as possible. With this in mind let's name this file "ae_attendances_ENG_4hr_perfom", and write it to the raw data folder.

```
glimpse(ae)
```

```
## Rows: 12,765
## Columns: 4
## $ index       <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,~
## $ period      <date> 2017-03-01, 2017-03-01, 2017-03-01, 2017-03-01, 2017-03-0~
## $ attendances <dbl> 21289, 813, 2850, 30210, 807, 11352, 4381, 19562, 17414, 7~
## $ breaches    <dbl> 2879, 22, 6, 5902, 11, 136, 2, 258, 2030, 86, 1322, 140, 0~
```

```
write_csv(ae, here("RawData", "ae_attendances_ENG_4hr_perfom.csv"))
```

Separating provisional ae_attendances_ENG_4hr_perfom data into training and testing sets

To develop and evaluate your data capture tool, you will need to splint the raw data into test and training data sets. We have added an index column earlier, so we can link the partitioned data sets to the raw ae_attendances and ae_attendances_ENG_4hr_perfom if required in the future.

How many rows are in the ae_attendances_ENG_4hr_perfom dataset?

```
#The ae_attendances_ENG_4hr_perfom dataset is large with
nrow(ae) #rows of data
```

```
## [1] 12765
```

We do not want you to spend hours inputting rows and rows of data into your data capture tool. A test data set of 10-15 records to capture with and evaluate your data capture tool is sufficient for the purpose of the course assignment. Here is how to work out the proportion (`prop`) of the raw data to assign to the training data:

```
prop<-(1-(15/nrow(ae)))
#The proportion of the raw that needs to be assigned to the training data to ensure there is only 10 to
print(prop)
```

```
## [1] 0.9988249
```

We will use the createDataPartition() function from the *caret* package to splint our raw data into test and training data sets. The 'set.seed()' function is a random number generator, which is useful for creating random objects that can be reproduced. This will make sure that every time we run this script, we will partition the raw data into the same test and training data.

```
set.seed(333)
#Partitioning the raw data into the test and training data.
trainIndex <- createDataPartition(ae$index, p = prop,
                                  list = FALSE,
                                  times = 1)
head(trainIndex)
```

```
##      Resample1
## [1,]         1
## [2,]         2
## [3,]         3
## [4,]         4
## [5,]         5
## [6,]         6
```

```
# All records that are in the trainIndex are assigned to the training data.
aeTrain <- ae[ trainIndex,]
nrow(aeTrain)
```

```
## [1] 12753
```

There are 12,753 records in your training data. That is a large dataset!

Let's tabulate ae_attendances_ENG_4hr_perfom training data for your report

```
aeTrain %>%
  # set the period column to show in Month-Year format
  mutate_at(vars(period), format, "%b-%y") %>%
  # set the numeric columns to have a comma at the 1000's place
  mutate_at(vars(attendances, breaches), comma) %>%
  # show the first 10 rows
  head(10) %>%
  # format as a table
  kable()
```

index	period	attendances	breaches
1	Mar-17	21,289.0	2,879.0
2	Mar-17	813.0	22.0
3	Mar-17	2,850.0	6.0
4	Mar-17	30,210.0	5,902.0
5	Mar-17	807.0	11.0
6	Mar-17	11,352.0	136.0
7	Mar-17	4,381.0	2.0
8	Mar-17	19,562.0	258.0
9	Mar-17	17,414.0	2,030.0
10	Mar-17	7,817.0	86.0

Our next task, it to save ae_attendances_ENG_4hr_perfom training data to your working data folder 'Data'

```
write_csv(aeTrain, here("Data", "ae_attendances_ENG_4hr_perfom_train.csv"))
```

Let's extract the ae_attendances_ENG_4hr_perfom test data

#All records that are not in the trainIndex (`-trainIndex`) are assigned to the test data.

```
aeTest  <- ae[-trainIndex,]
nrow(aeTest)
```

```
## [1] 12
```

There are 12 records in your test data. Perfect! You now need to set aside the first record from the ae_attendances_ENG_4hr_perfom test data so that your markers can test and evaluate your data-capture tool.

```
aeTestMarker  <- aeTest[1,]
```

```
aeTestMarker  %>%
  # set the period column to show in Month-Year format
  mutate_at(vars(period), format, "%b-%y") %>%
  # set the numeric columns to have a comma at the 1000's place
  mutate_at(vars(attendances, breaches), comma) %>%
  # show the first 10 rows
  head(10) %>%
  # format as a table
  kable()
```

Let's tabulate ae_attendances_ENG_4hr_perfom marker test data for your report

index	period	attendances	breaches
115	Mar-17	309	1

Our next task, it to save our ae_attendances_ENG_4hr_perfom marker test data to our working data folder 'Data'

```
write_csv(aeTestMarker, here("Data", "ae_attendances_ENG_4hr_perfom_test_marker.csv"))
```

We then need to set aside the remaining records for you to test (or collect with your) your data-capture tool.

```
aeTest  <- aeTest[2:nrow(aeTest),]
```

```
aeTest  %>%
  # set the period column to show in Month-Year format
  mutate_at(vars(period), format, "%b-%y") %>%
  # set the numeric columns to have a comma at the 1000's place
  mutate_at(vars(attendances, breaches), comma) %>%
  # show the first 10 rows
  head(10) %>%
  # format as a table
  kable()
```

Let's tabulate ae_attendances_ENG_4hr_perfom test data for your report

index	period	attendances	breaches
1155	Dec-16	200	0.0
2059	Oct-16	6,452	360.0
3468	May-16	417	0.0
4153	Mar-18	9,376	112.0
4820	Feb-18	245	0.0
7243	Jul-17	5,170	235.0
8057	Apr-17	15,957	1,309.0
8957	Feb-19	7,258	1,374.0
10214	Oct-18	3,197	0.0
10328	Oct-18	2,033	8.0

Our final task, is to save our ae_attendances_ENG_4hr_perfom test data to our working data folder 'Data'

```
write_csv(aeTest, here("Data", "ae_attendances_test.csv"))
```

You are now ready to start exploring your data. Happy coding!