Overview

This document will guide you through some data analysis tasks with a focus on performing variable selection. For this exercise, we consider a categorical outcome.

While this is in some sense a stand-alone analysis, I assume that you have worked through the *Data Analysis* exercise and are familiar with the dataset and all the things we discovered during the cleaning process. We'll use the same dataset here but focus on a different outcome. Other than that, the way to work through the exercise is like in the *Data Analysis* exercise, namely by writing/completing the missing code.

Project setup

We need a variety of different packages, which are loaded here. Install as needed. If you use others, load them here.

```
library('tidyr')
library('dplyr')
library('forcats')
library('ggplot2')
library('knitr')
library('mlr') #for model fitting.
library('parallelMap') #for using multiple processors when running models through mlr
library(gridExtra)
library(visdat)
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

Data loading and cleaning

We will again use the Norovirus dataset.

```
#Write code that loads the dataset
#You can of course re-use code you wrote in the other file.
norodata_raw <- read.csv("./norodata.csv")

glimpse(norodata_raw)
```

```
## Observations: 1,022
## Variables: 139
## $ id                    <int> 2, 17, 39, 40, 41, 42, 43, 44, 67, 74, 7...
## $ Author                <fct> Akihara, Becker, Boxman, Boxman, Boxman,...
## $ Pub_Year              <int> 2005, 2000, 2009, 2009, 2009, 2009, 2009...
## $ pubmedid              <int> 15841336, 11071673, 19205471, 19205471, ...
## $ EpiCurve              <fct> Y, Y, N, N, N, N, N, N, N, N, Y, Y, Y, Y...
## $ TDComment             <fct> , , , , , , , , , , , , , , , , , , , No...
## $ AHComment             <fct> , , , , , , , , , , , , , , , , , , , , ...
## $ Trans1                <fct> Unspecified, Foodborne, Foodborne, Foodb...
## $ Trans1_O              <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
```

```
## $ Trans2               <fct>   (not applicable), Person to Person,   ...
## $ Trans2_O             <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ Trans3               <fct>   (not applicable),   (not applicable), ...
## $ Trans3_O             <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ Risk1                <dbl> 0.00000, 108.00000, 130.00000, 4.00000, ...
## $ Risk2                <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ RiskAll              <dbl> 0.00000, 108.00000, 130.00000, 4.00000, ...
## $ Cases1               <int> 15, 43, 27, 4, 15, 6, 40, 10, 116, 45, 1...
## $ Cases2               <int> NA, 22, NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ CasesAll             <int> 15, 65, 27, 4, 15, 6, 40, 10, 116, 45, 1...
## $ Rate1                <dbl> NA, 39.814815, 20.769231, 100.000000, 60...
## $ Rate2                <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ RateAll              <dbl> 0.000000, 39.814815, 20.769231, 100.0000...
## $ Hospitalizations     <int> 0, 0, 0, 0, 0, 0, 0, 0, 5, 10, 3, 0, 0, ...
## $ Deaths               <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ Vehicle_1            <fct> 0, Boxed Lunch, 0, 0, 0, 0, 0, 0, 0, Oys...
## $ Veh1                 <fct> Unspecified, Yes, Unspecified, Unspecifi...
## $ Veh1_D_1             <fct> 0, Turkey Sandwich in boxed lunch, 0, 0,...
## $ Veh2                 <fct> No, Yes, No, No, No, No, No, No, No, No,...
## $ Veh2_D_1             <fct> 0, Football players, 0, 0, 0, 0, 0, 0, 0...
## $ Veh3                 <fct> No, No, No, No, No, No, No, No, No, No, ...
## $ Veh3_D_1             <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ PCRSect              <fct> Capsid, Polymerase, Both, Both, Both, Bo...
## $ OBYear               <fct> 1999, 1998, 2006, 2006, 2006, 2006, 2006...
## $ Hemisphere           <fct> Northern, Northern, Northern, Northern, ...
## $ season               <fct> Fall, Fall, Fall, Fall, Fall, Fall, Fall...
## $ MeanI1               <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ MedianI1             <int> 0, 37, 0, 0, 0, 0, 0, 0, 0, 31, 34, 33, ...
## $ Range_S_I1           <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 6, 0, 0...
## $ Range_L_I1           <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 69, 0, 96, 0,...
## $ MeanD1               <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 24, ...
## $ MedianD1             <dbl> 0, 36, 0, 0, 0, 0, 0, 0, 0, 48, 37, 24, ...
## $ Range_S_D1           <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 10, 0, 5, 4, ...
## $ Range_L_D1           <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 168, 0, 120, ...
## $ MeanA1               <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ MedianA1             <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ Range_Y_A1           <fct> 0.75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ Range_O_A1           <dbl> 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ Action1              <fct> Unspecified, Unspecified, Unspecified, U...
## $ Action2_1            <fct> "0", "0", "0", "0", "0", "0", "0", "0", ...
## $ Secondary            <fct> No, Yes, No, No, No, No, No, No, No, No,...
## $ MeanI2               <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ MedianI2             <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ Range_S_I2           <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ Range_L_I2           <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ MeanD2               <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ MedianD2             <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ Range_S_D2           <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ Range_L_D2           <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ Mea.2                <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ Media.2              <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ Range_Y_A2           <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ Range_O_A2           <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ Comments_1           <fct> "Outbreak took place during a study on g...
```

```
## $ Path1                <fct> No, No, Unspecified, Unspecified, Unspec...
## $ Path2_1              <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ Country              <fct> Japan, USA, Other, Other, Other, Other, ...
## $ Category             <fct> Daycare, Foodservice, Foodservice, Foods...
## $ State                <fct> "0", "NC, FL", "0", "0", "0", "0", "0", ...
## $ Setting_1            <fct> "Daycare Center", "Boxed lunch, football...
## $ StartMonth           <int> 11, 9, 9, 10, 11, 11, 11, 11, 11, 11, 11...
## $ EndMonth             <int> 12, 9, 0, 0, 0, 0, 0, 0, 11, 11, 11, 11,...
## $ GGA                  <int> 2, 1, 2, 0, 2, 0, 0, 0, 2, 0, 0, 0, 0, 0...
## $ CA                   <int> 4, 0, 4, 0, 4, 0, 0, 0, 4, 0, 0, 0, 0, 0...
## $ SA                   <fct> Lordsdale, Thistle Hall 1/91, GII.4 2006...
## $ new_GGA              <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ new_CA               <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ new_SA               <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ SA_resolved_from     <fct> , , , , , , , , , , , , , , , , , , , Zh...
## $ GGB                  <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ CB                   <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ SB                   <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ new_GGB              <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ new_CB               <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ new_SB               <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ SB_resolved_from     <fct> , , , , , , , , , , , , , , , , , , , http...
## $ GGC                  <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ CC                   <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ SC                   <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ new_ggc              <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ new_cc               <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ new_sc               <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ SC_resolved_from     <fct> , , , , , , , , , , , , , , , , , , , , ...
## $ GGD                  <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ CD                   <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ SD                   <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ new_ggd              <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ new_cd               <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ new_sd               <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ SD_resolved_from     <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ StrainOther          <fct> "0", "0", "0", "0", "0", "0", "0", "0", ...
## $ strainother_rc       <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ gge                  <fct> 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0...
## $ ce                   <int> 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0...
## $ se                   <fct> 0, 0, 0, 0, 0, 0, 0, 0, GII.4a 2004, 0, ...
## $ SE_resolved_from     <fct> , , , , , , , , abstraction of StrainOth...
## $ ggf                  <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ cf                   <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ sf                   <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ ggg                  <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ cg                   <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ sg                   <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ ggh                  <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ ch                   <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ sh                   <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ ggi                  <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ ci                   <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ si                   <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
```

```
## $ ggj                   <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ cj                    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ sj                    <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ Country2              <fct> "0", "0", "The Netherlands", "The Nether...
## $ Veh1_D_2              <fct> "0", "Boxed Lunch", "0", "0", "0", "0", ...
## $ Veh2_D_2              <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ Veh3_D_2              <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ Action2_2             <fct> "0", "0", "0", "0", "0", "0", "0", "0", ...
## $ Comments_2            <fct> "Limited data", "0", "Outbreak 19 of 26 ...
## $ Path2_2               <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ Setting_2             <fct> 0, 0, Buffet, Restaurant, Buffet, take o...
## $ category1             <fct> School/Daycare, Foodservice, Foodservice...
## $ strainothergg2c4      <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ gg2c4                 <fct> Yes, , Yes, , Yes, , , , Yes, , , , , , ...
## $ Vomit                 <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
## $ IncInd                <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ SymInd                <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ PooledLat             <dbl> 0, 37, 0, 0, 0, 0, 0, 0, 0, 31, 34, 33, ...
## $ PooledSym             <int> 0, 36, 0, 0, 0, 0, 0, 0, 0, 48, 37, 24, ...
## $ PooledAge             <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ IndividualLatent      <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ IndividualSymptomatic <fct> , , , , , , , , , , , , , , , , , , , , ...
```

Looking at the outcome

For this analysis, our main outcome of interest is if an outbreak was caused by the G2.4 strain of norovirus or not, and how other factors might be correlated with that strain.

```
#write code to take a look at the outcome variable (gg2c4)
table(norodata_raw$gg2c4)
```

```
##
##     Yes
## 716 306
```

```
print(norodata_raw$gg2c4)
```

```
##    [1] Yes     Yes     Yes             Yes
##   [18]     Yes     Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes
##   [35] Yes Yes Yes         Yes                                        Yes
##   [52] Yes             Yes         Yes                     Yes
##   [69] Yes         Yes                         Yes Yes Yes Yes Yes
##   [86]         Yes                     Yes Yes Yes     Yes
##  [103]         Yes Yes                                 Yes Yes         Yes
##  [120] Yes Yes     Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes
##  [137]                         Yes                 Yes Yes
##  [154]                                 Yes Yes         Yes
##  [171]     Yes Yes Yes Yes                                 Yes
##  [188] Yes Yes Yes Yes     Yes Yes             Yes Yes Yes     Yes Yes
##  [205]
##  [222]                                     Yes         Yes
##  [239]                                 Yes
##  [256]                         Yes     Yes
##  [273]     Yes                     Yes             Yes
##  [290] Yes                                                         Yes
```

```
##  [307]                          Yes     Yes Yes
##  [324]                                              Yes Yes
##  [341]     Yes Yes         Yes                          Yes Yes Yes Yes
##  [358]                 Yes Yes                              Yes     Yes
##  [375] Yes     Yes
##  [392]     Yes Yes Yes Yes             Yes Yes             Yes
##  [409]         Yes             Yes             Yes             Yes Yes Yes
##  [426]                     Yes         Yes         Yes Yes     Yes
##  [443]                                     Yes
##  [460] Yes
##  [477]                 Yes                 Yes Yes Yes Yes Yes Yes
##  [494]                                             Yes                 Yes
##  [511]                     Yes     Yes             Yes
##  [528]                                         Yes Yes     Yes Yes Yes Yes
##  [545] Yes Yes Yes Yes                 Yes     Yes Yes Yes             Yes
##  [562] Yes Yes
##  [579]     Yes                                     Yes             Yes
##  [596]
##  [613]
##  [630]                                                         Yes Yes
##  [647]             Yes     Yes         Yes Yes Yes Yes     Yes Yes
##  [664] Yes Yes                     Yes                                 Yes
##  [681]         Yes         Yes         Yes Yes             Yes
##  [698]                 Yes                     Yes     Yes     Yes Yes
##  [715]         Yes                         Yes             Yes
##  [732] Yes Yes Yes         Yes     Yes Yes     Yes Yes Yes Yes Yes Yes Yes
##  [749] Yes Yes Yes     Yes Yes                 Yes Yes Yes     Yes Yes Yes
##  [766]     Yes     Yes Yes Yes     Yes         Yes     Yes
##  [783]                                 Yes         Yes Yes Yes
##  [800]                                     Yes
##  [817]                                             Yes         Yes Yes
##  [834]             Yes     Yes             Yes Yes Yes     Yes     Yes Yes
##  [851]     Yes Yes     Yes Yes Yes Yes Yes Yes Yes Yes Yes Yes
##  [868]                             Yes     Yes Yes Yes     Yes     Yes Yes
##  [885] Yes Yes                                                         Yes
##  [902] Yes Yes Yes                                         Yes Yes Yes Yes Yes
##  [919]     Yes Yes     Yes Yes Yes Yes Yes Yes Yes     Yes     Yes Yes
##  [936] Yes Yes         Yes
##  [953]                                     Yes         Yes
##  [970]                             Yes     Yes Yes Yes Yes Yes Yes Yes Yes
##  [987] Yes Yes                                 Yes Yes Yes Yes Yes Yes
## [1004] Yes Yes Yes Yes Yes                             Yes     Yes     Yes
## [1021] Yes Yes
## Levels:  Yes
```

Overall, it looks ok, a decent amout in each category (i.e. no unbalanced data). However, we see an odd coding, there are only 2 types of entries, either “Yes” or "“, i.e. an empty space. The codebook tells us that this is how things were coded, if it was G2.4 it got a”Yes“, if it wasn’t it did get an empty space instead of a”No“. That’s somewhat strange and while the analysis should work, it is better to re-code the empty slots with”No".

```
#write code to change the empty values in gg2c4 to "No"

norodata_raw$gg2c4[which(norodata_raw$gg2c4=="")] <- "No"
```

```
## Warning in `[<-.factor`(`*tmp*`, which(norodata_raw$gg2c4 == ""), value =
## structure(c(2L, : invalid factor level, NA generated
```

```
glimpse(norodata_raw$gg2c4) #this makes No as NA so we'll try and fix it
```

```
##  Factor w/ 2 levels "","Yes": 2 NA 2 NA 2 NA NA NA 2 NA ...
```

```
norodata_raw$gg2c4 <- as.character(norodata_raw$gg2c4)

norodata_raw$gg2c4[is.na(norodata_raw$gg2c4)] <- "No"

glimpse(norodata_raw$gg2c4)
```

```
##  chr [1:1022] "Yes" "No" "Yes" "No" "Yes" "No" "No" "No" "Yes" "No" ...
```

```
#lovely
```

Selecting predictors

We will pick similar variables as previously, with some adjustments based on what we learned before. Keep the following variables: `Action1`, `CasesAll`, `Country`, `Deaths`, `GG2C4`, `Hemisphere`, `Hospitalizations`, `MeanD1`, `MeanI1`, `MedianD1`, `MedianI1`, `OBYear`, `Path1`, `RiskAll`, `Season`, `Setting`, `Trans1`, `Vomit`.

```
norodata_rawv <- norodata_raw %>% dplyr::select(Action1, CasesAll, Country, Deaths, gg2c4, Hemisphere, H
dim(norodata_rawv)
```

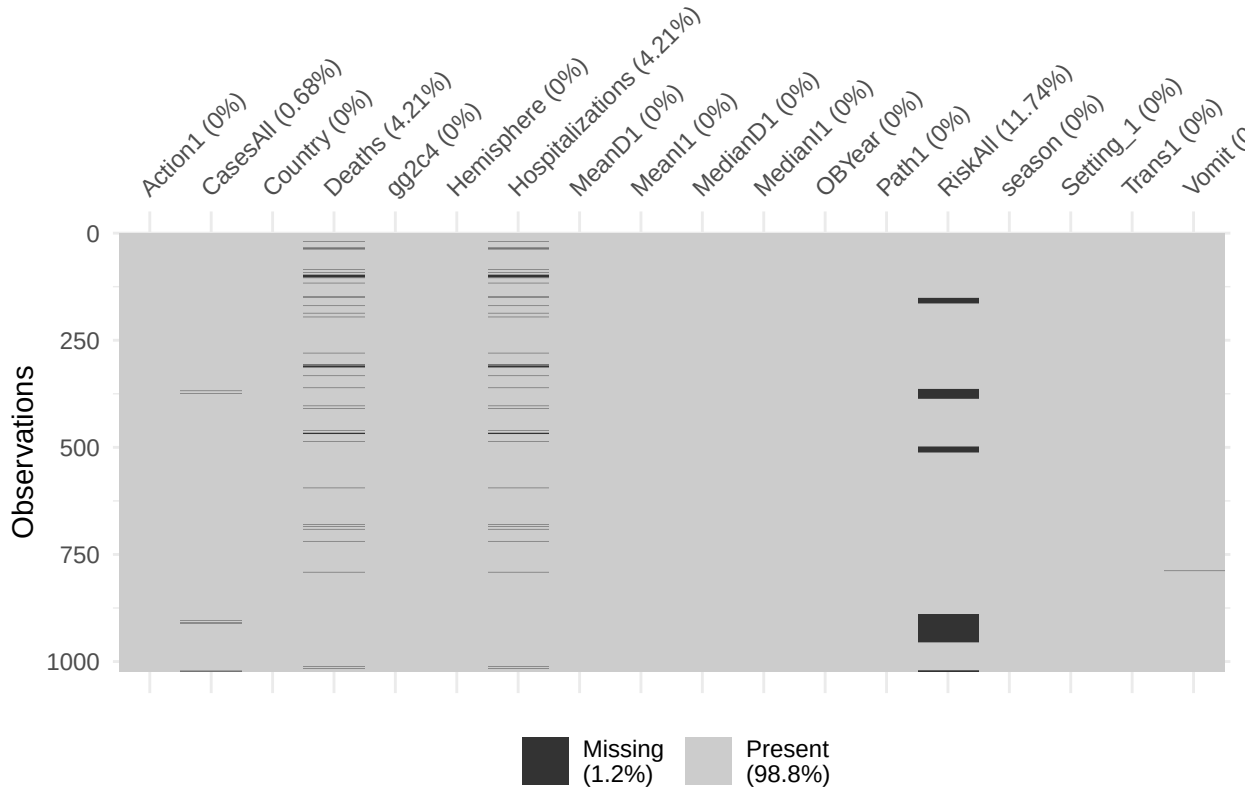
```
## [1] 1022   18
```

Cleaning predictors

We'll likely have to perform similar cleaning steps as before. A difference is that earlier we had to drop about half of the observations because no outcome was available. Here, we have outcome for every outbreak. Thus, there is more data to clean, which - as we will see - introduces a few issues we didn't have before, since we dropped the troublesome observations early in the process.

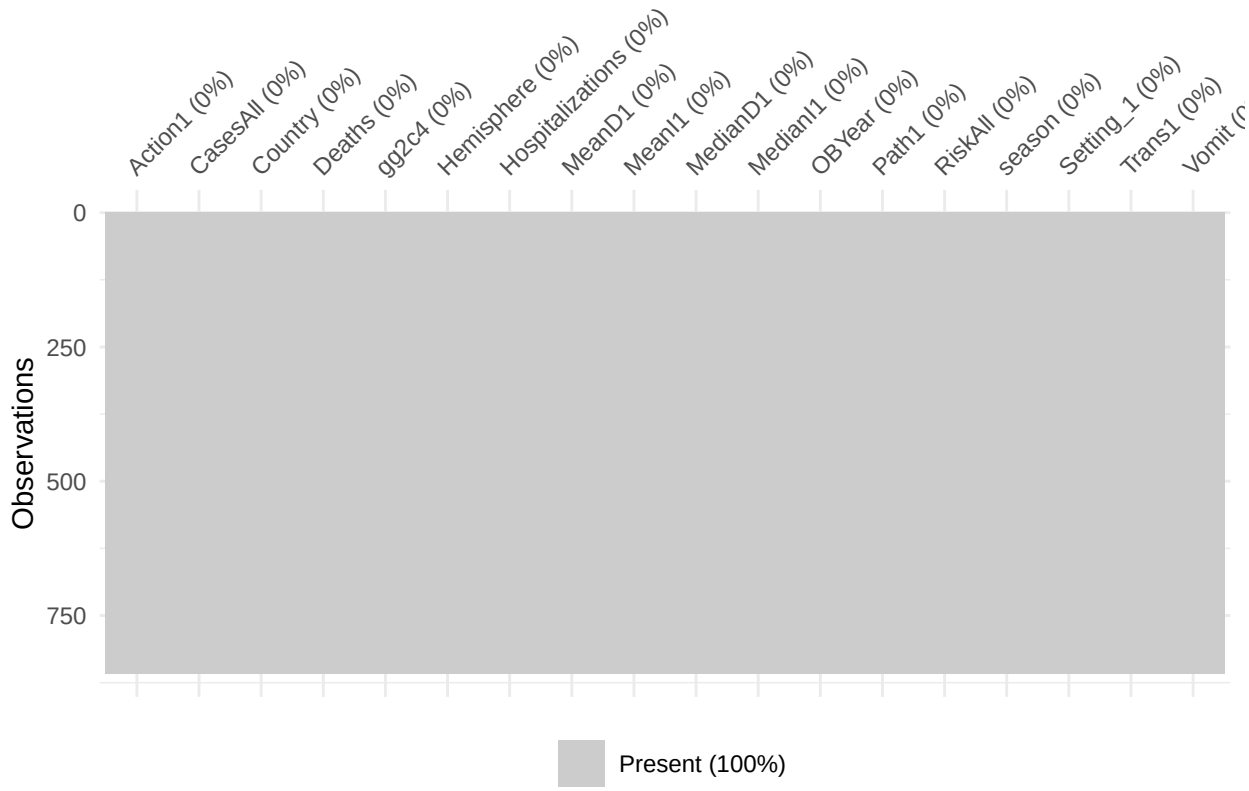
Let's first check for missing values.

```
# write code that looks at missing values
vis_miss(norodata_rawv)
```



Looks like we have again some missing in `Hospitalization` and `Deaths`, and some more in `RiskAll`. Since the missing is not excessive, and to make our life easier, we'll drop them for now. Note however the 'blocks' of missing values for RiskAll. Given that these outbreaks should be entered fairly randomly into the spreadsheet, it is strange to see the NA show up in such blocks. For a real data analysis, it would be worth looking closer and checking why there is a clustering like that.

```
# write code to remove any observations with NA
removena_noro <- na.omit(norodata_rawv)
vis_miss(removena_noro)
```



Let's make sure everything has the right format (numeric/integer/factor). Adjust/recode variables as needed. You will likely find that as you convert `OBYear` to numeric, something doesn't quite work. Take a look. Fix by removing the observation with the troublesome entry, then convert to numeric. Finally, remove the observations that have 0 as OByear - there are more than 1 now.

```
#write code that cleans OBYear, convert it to numeric. Remove observations with OBYear = 0.
unique(removena_noro$OBYear)
```

```
##  [1] 1999 1998 2006 2004 1993 2002 2005 2003 1994 2008 2000 2001 1997 1995
## [15] 1996 2007 2009 1990 0    1983 2010 1992
## 23 Levels: 0 1983 1990 1992 1993 1994 1995 1996 1997 1998 1999 ... 2010
```

```
print(removena_noro$OBYear == "1983")
```

```
##   [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [34] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [45] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [56] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [67] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [78] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [89] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [100] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [111] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [122] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [133] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [144] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
## [155] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [166] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [177] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [188] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [199] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [210] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [221] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [232] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [243] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [254] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [265] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [276] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [287] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [298] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [309] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [320] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [331] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [342] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [353] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [364] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [375] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [386] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [397] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [408]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [419] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [430] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [441] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [452] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [463] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [474] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [485] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [496] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [507] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [518] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [529] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [540] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [551] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [562] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [573] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [584] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [595] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [606] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [617] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [628] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [639] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [650] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [661] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [672] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [683] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [694] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [705] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [716] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [727] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [738] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
## [749] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [760] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [771] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [782] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [793] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [804] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [815] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [826] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [837] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [848] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
remove1983 <- removena_noro[!(removena_noro$OBYear == "1983"), ]

remove1983$OBYear <- as.numeric(levels(remove1983$OBYear))[remove1983$OBYear]
```

```
## Warning: NAs introduced by coercion
```

```
cleanOBY <- remove1983[!(remove1983$OBYear == "0"), ]

unique(cleanOBY$OBYear)
```

```
##  [1] 1999 1998 2006 2004 1993 2002 2005 2003 1994 2008 2000 2001 1997 1995
## [15] 1996 2007 2009 1990 2010 1992
```

```
#also convert any other variables as needed
cleanOBY$gg2c4 <- as.factor(cleanOBY$gg2c4)
glimpse(cleanOBY)
```

```
## Observations: 851
## Variables: 18
## $ Action1          <fct> Unspecified, Unspecified, Unspecified, Unspec...
## $ CasesAll         <int> 15, 65, 27, 4, 15, 6, 40, 10, 116, 45, 184, 1...
## $ Country          <fct> Japan, USA, Other, Other, Other, Other, Other...
## $ Deaths           <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ gg2c4            <fct> Yes, No, Yes, No, Yes, No, No, No, Yes, No, N...
## $ Hemisphere       <fct> Northern, Northern, Northern, Northern, North...
## $ Hospitalizations <int> 0, 0, 0, 0, 0, 0, 0, 0, 5, 10, 3, 0, 0, 0, 0,...
## $ MeanD1           <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 24, 0, 0,...
## $ MeanI1           <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ MedianD1         <dbl> 0, 36, 0, 0, 0, 0, 0, 0, 0, 48, 37, 24, 0, 0,...
## $ MedianI1         <int> 0, 37, 0, 0, 0, 0, 0, 0, 0, 31, 34, 33, 0, 0,...
## $ OBYear           <dbl> 1999, 1998, 2006, 2006, 2006, 2006, 2006, 200...
## $ Path1            <fct> No, No, Unspecified, Unspecified, Unspecified...
## $ RiskAll          <dbl> 0.00000, 108.00000, 130.00000, 4.00000, 25.00...
## $ season           <fct> Fall, Fall, Fall, Fall, Fall, Fall, Fall, Fal...
## $ Setting_1        <fct> "Daycare Center", "Boxed lunch, football game...
## $ Trans1           <fct> Unspecified, Foodborne, Foodborne, Foodborne,...
## $ Vomit            <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
```

Next, we remove the `Unspecified` entry in `Hemisphere` and recode `Action1` and `Path1` as described in the Data Analysis script, i.e. from `Unknown` to `Unspecified`. Also do the same grouping into just `Restaurant` and `Other` with the `Setting_1` variable. Again, remember that there are `restaurant` and `Restaurant` values, so you need to fix that too.

```
# write code that performs the actions described above
# at the end, use the droplevels() command to remove empty factor levels
glimpse(cleanOBY$Hemisphere)
```

```
##  Factor w/ 3 levels "Northern","Southern",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
drop_UH <- cleanOBY[!(cleanOBY$Hemisphere == "Unspecified"), ]

recodeA1P1 <- drop_UH %>% dplyr::mutate(Action1 = recode(Action1, "Unknown" = "Unspecified")) %>% dplyr


RecodeRestaurant <- recodeA1P1 %>% mutate(Setting_1 = recode(Setting_1, "restaurant" = "Restaurant", "ta
"Catering service in Restaurant" = "Restaurant",
"restaurant in Northern Territory" = "Restaurant",
"Shared meal at a restaurant" = "Restaurant",
"Resaurant" = "Restaurant",
"restaurant; catered party" = "Restaurant"))

find_Restaurant <- RecodeRestaurant[grep("Restaurant", RecodeRestaurant$Setting_1), ]



Restnew <- find_Restaurant %>% dplyr::mutate(Setting = "Restaurant")

Other <- dplyr::filter(RecodeRestaurant, !grepl("Restaurant", Setting_1))

Setting_Other <- Other %>% dplyr::mutate(Setting = "Other")

combined_data <- merge(Setting_Other, Restnew, all = TRUE)

combined_data$Setting_1 <- NULL


RestaurantData <- combined_data

RestaurantData$Setting <- as.factor(as.character(RestaurantData$Setting))
glimpse(RestaurantData)
```

```
## Observations: 850
## Variables: 18
## $ Action1          <fct> No, Unspecified, Unspecified, Unspecified, Un...
## $ CasesAll         <int> 6, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, ...
## $ Country          <fct> Unspecified, Japan, Japan, Japan, Japan, Japa...
## $ Deaths           <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ gg2c4            <fct> No, No, No, No, No, No, No, No, No, No, No, N...
## $ Hemisphere       <fct> Northern, Northern, Northern, Northern, North...
## $ Hospitalizations <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ MeanD1           <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ MeanI1           <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ MedianD1         <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ MedianI1         <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ OBYear           <dbl> 2003, 1997, 1999, 2000, 2003, 2003, 1996, 199...
## $ Path1            <fct> Unspecified, Unspecified, Unspecified, Unspec...
## $ RiskAll          <dbl> 0, 2, 0, 0, 1, 1, 0, 0, 2, 2, 2, 0, 0, 3, 2, ...
## $ season           <fct> , Winter, Winter, Winter, Winter, Winter, Sum...
## $ Trans1           <fct> Unspecified, Unknown, Foodborne, Unknown, Foo...
## $ Vomit            <int> 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, ...
## $ Setting          <fct> Other, Other, Other, Other, Other, Other, Oth...
```

```
CompData <- RestaurantData
```

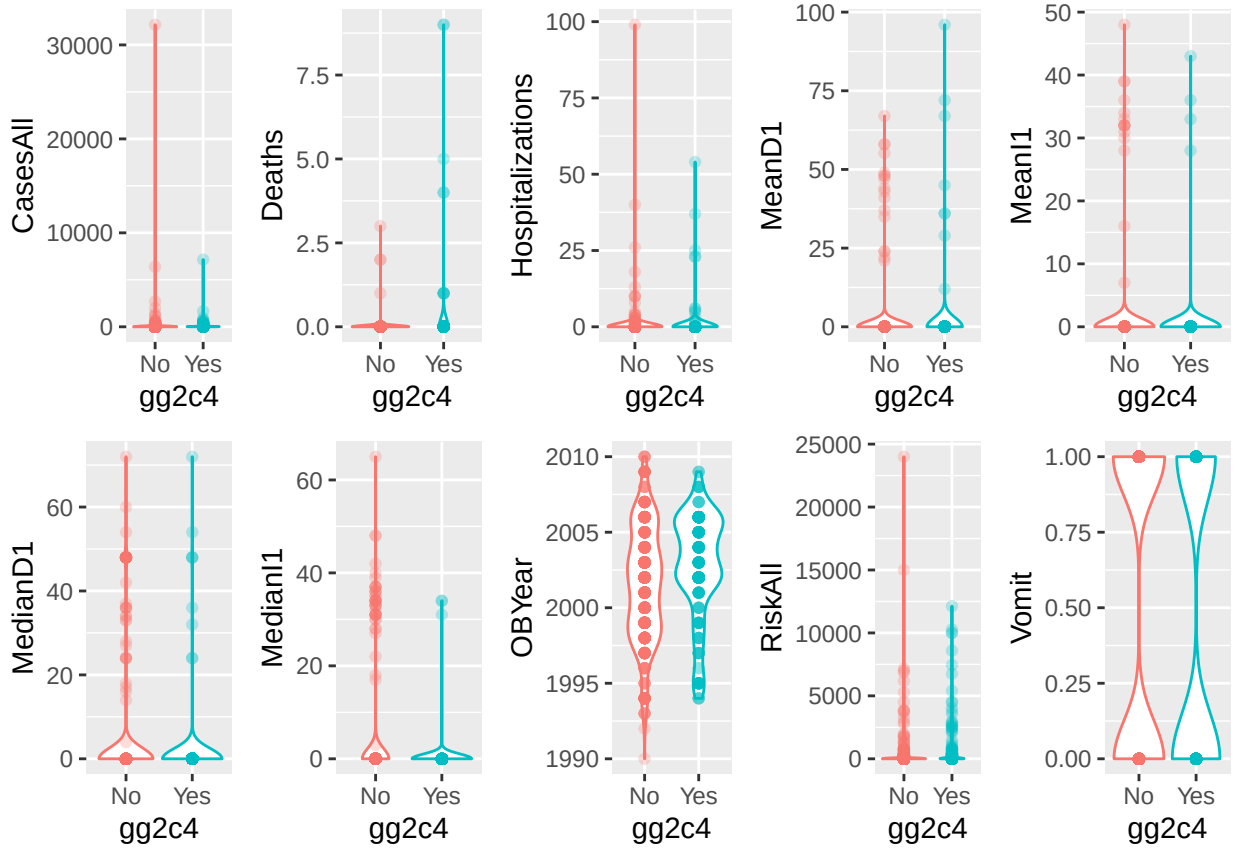
Data visualization

Next, let's create a few plots showing the outcome and the predictors. For the continuous predictors, I suggest scatter/box/violinplots with the outcome on the x-axis.

```
#write code that produces plots showing our outcome of interest on the x-axis and each numeric predicto
#you can use the facet_wrap functionality in ggplot for it, or do it some other way.
gg2c4_vs_casesall <- CompData %>%

  ggplot(aes(x = gg2c4, y = CasesAll, color = gg2c4)) +

  geom_violin() +

  geom_point(alpha = 0.25) +

  theme(legend.position = "none")




gg2c4_vs_deaths <-  CompData %>%

  ggplot(aes(x = gg2c4, y = Deaths, color = gg2c4)) +

  geom_violin() +

  geom_point(alpha = 0.25) +

  theme(legend.position = "none")


gg2c4_vs_hosp <- CompData %>%

  ggplot(aes(x = gg2c4, y = Hospitalizations, color = gg2c4)) +

  geom_violin() +

  geom_point(alpha = 0.25) +

  theme(legend.position = "none")


gg2c4_vs_meand1 <- CompData %>%

  ggplot(aes(x = gg2c4, y = MeanD1, color = gg2c4)) +

  geom_violin() +
```

```
  geom_point(alpha = 0.25) +
  theme(legend.position = "none")



gg2c4_vs_meani1 <- CompData %>%
  ggplot(aes(x = gg2c4, y = MeanI1, color = gg2c4)) +
  geom_violin() +
  geom_point(alpha = 0.25) +
  theme(legend.position = "none")


gg2c4_vs_mediand1 <- CompData %>%
  ggplot(aes(x = gg2c4, y = MedianD1, color = gg2c4)) +
  geom_violin() +
  geom_point(alpha = 0.25) +
  theme(legend.position = "none")


gg2c4_vs_mediani1 <- CompData %>%
  ggplot(aes(x = gg2c4, y = MedianI1, color = gg2c4)) +
  geom_violin() +
  geom_point(alpha = 0.25) +
  theme(legend.position = "none")


gg2c4_vs_obyear <- CompData %>%
  ggplot(aes(x = gg2c4, y = OBYear, color = gg2c4)) +
  geom_violin() +
  geom_point(alpha = 0.25) +
  theme(legend.position = "none")
```

```r
gg2c4_vs_riskall <- CompData %>%
  ggplot(aes(x = gg2c4, y = RiskAll, color = gg2c4)) +
  geom_violin() +
  geom_point(alpha = 0.25) +
  theme(legend.position = "none")

gg2c4_vs_vomit <- CompData %>%
  ggplot(aes(x = gg2c4, y = Vomit, color = gg2c4)) +
  geom_violin() +
  geom_point(alpha = 0.25) +
  theme(legend.position = "none")

grid.arrange(gg2c4_vs_casesall, gg2c4_vs_deaths, gg2c4_vs_hosp, gg2c4_vs_meand1, gg2c4_vs_meani1, gg2c4
```

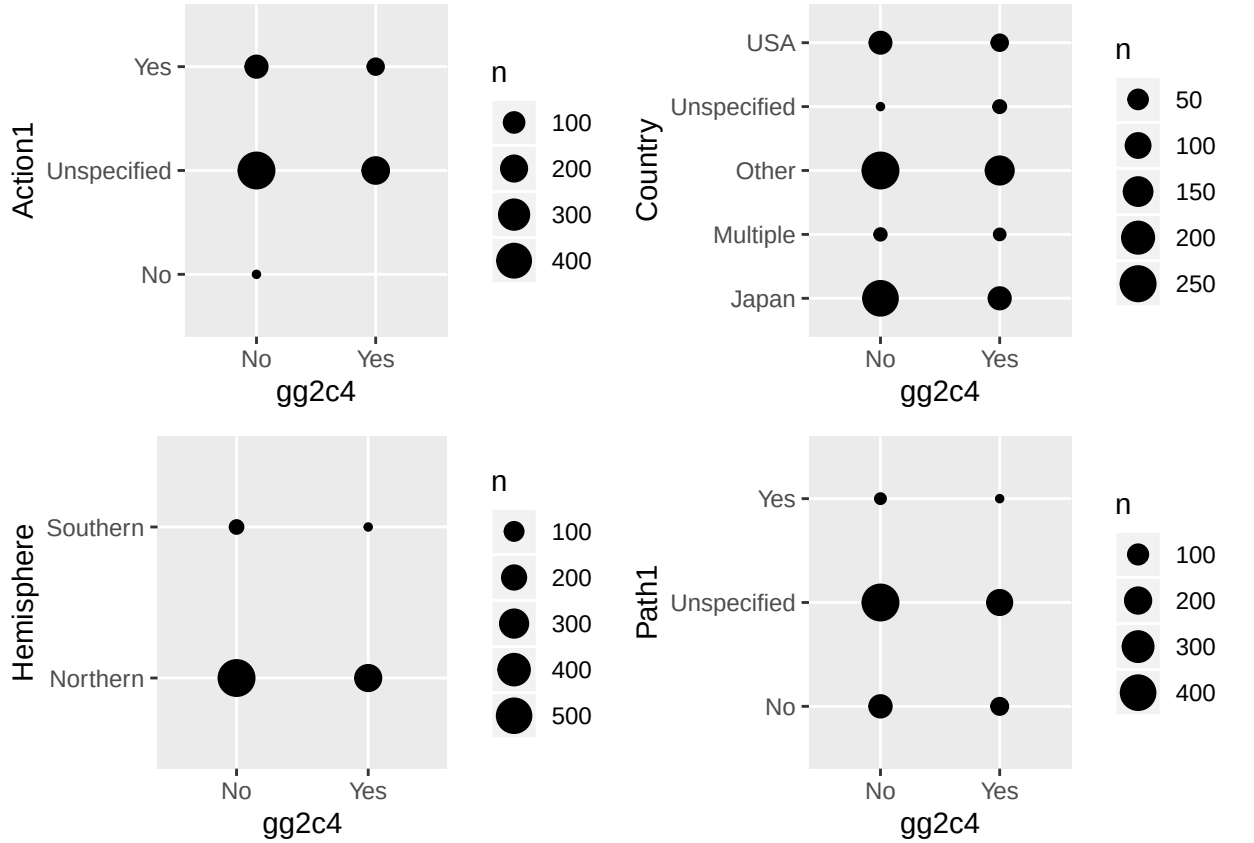


Things look ok, apart from the skew in the predictors we discussed previously.

Next, let's create plots for the categorical variabless. You can use for instance `geom_count` for it, or some other representation. If you prefer lots of tables, that's ok too.

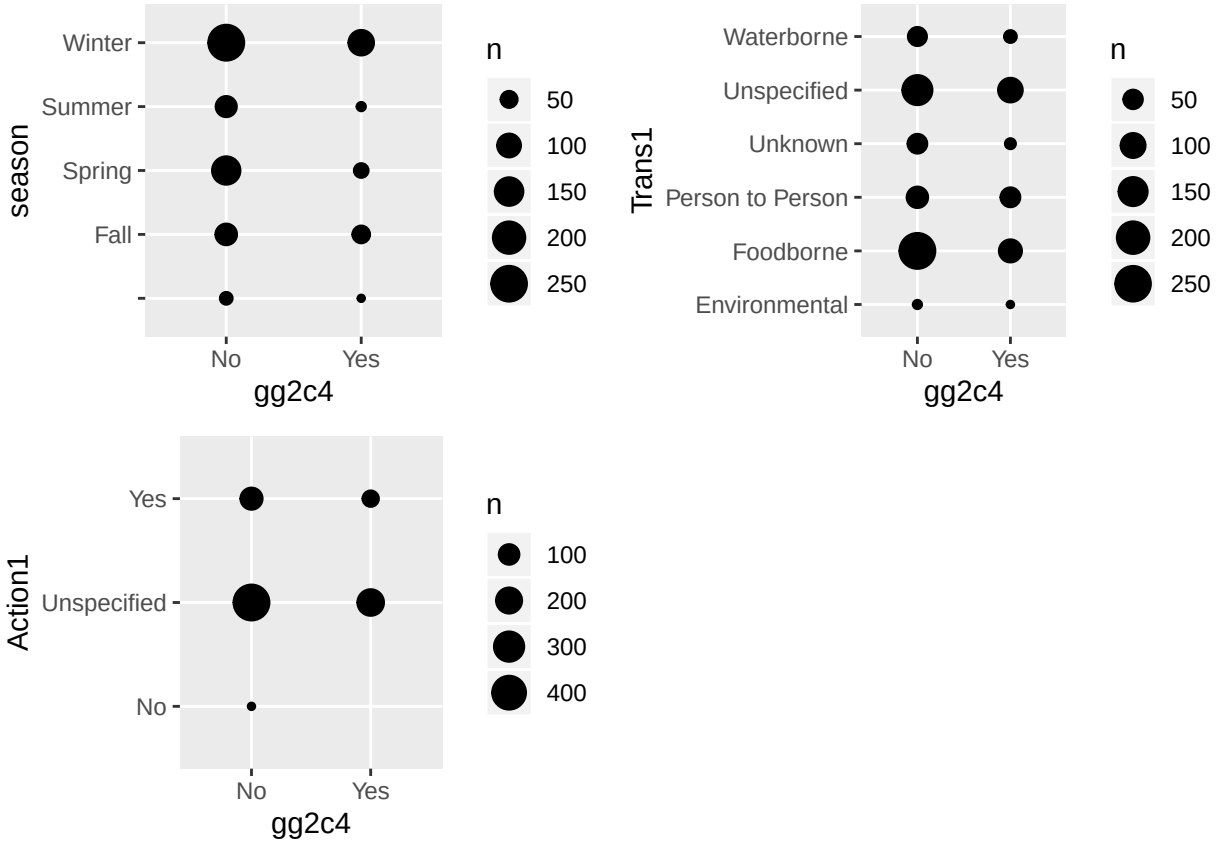
```
#write code that produces plots or tables showing our outcome of interest and each categorical predicto


gg2c4_vs_action1 <- CompData %>%

  ggplot(aes(x = gg2c4, y = Action1)) +

  geom_count()


gg2c4_vs_country <- CompData %>%

  ggplot(aes(x = gg2c4, y = Country)) +

  geom_count()


gg2c4_vs_hemi <- CompData %>%

  ggplot(aes(x = gg2c4, y = Hemisphere)) +
```

```
  geom_count()


gg2c4_vs_path1 <- CompData %>%
  ggplot(aes(x = gg2c4, y = Path1)) +
  geom_count()
grid.arrange(gg2c4_vs_action1, gg2c4_vs_country, gg2c4_vs_hemi, gg2c4_vs_path1, nrow = 2)
```




```
gg2c4_vs_season <- CompData %>%
  ggplot(aes(x = gg2c4, y = season)) +
  geom_count()


gg2c4_vs_tras1 <- CompData %>%
  ggplot(aes(x = gg2c4, y = Trans1)) +
  geom_count()
```

```
gg2c4_vs_Setting <- CompData %>%
  ggplot(aes(x = gg2c4, y = Action1)) +
  geom_count()

grid.arrange(gg2c4_vs_season, gg2c4_vs_tras1, gg2c4_vs_Setting, nrow = 2)
```



You should see from plots or tables that some of the categories are small, e.g. for Action1, the "No" category is very small. Very few entries for a given factor create problems during cross-validation (since we can't have a level show up in the holdout if it wasn't part of the fitting set). So let's look at those factor variables a bit closer and fix as needed.

```
#write code that looks at tables/summaries of factors

summary(CompData)
```

```
##         Action1       CasesAll            Country        Deaths
##  No          :  1   Min.   :    1   Other      :408   Min.   :0.00000
##  Unspecified:677   1st Qu.:    9   Japan      :316   1st Qu.:0.00000
##  Yes         :172   Median :   25   USA        : 98   Median :0.00000
```

```
##                   Mean   :  128   Multiple   : 16   Mean   :0.05176
##                   3rd Qu.:   65   Unspecified: 12   3rd Qu.:0.00000
##                   Max.   :32150   Australia  :  0   Max.   :9.00000
##                                   (Other)    :  0
##  gg2c4           Hemisphere  Hospitalizations      MeanD1
##  No :591   Northern   :798   Min.   : 0.0000   Min.   : 0.000
##  Yes:259   Southern   : 52   1st Qu.: 0.0000   1st Qu.: 0.000
##            Unspecified:  0   Median : 0.0000   Median : 0.000
##                              Mean   : 0.5518   Mean   : 1.311
##                              3rd Qu.: 0.0000   3rd Qu.: 0.000
##                              Max.   :99.0000   Max.   :96.000
##
##      MeanI1           MedianD1         MedianI1          OBYear
##  Min.   : 0.0000   Min.   : 0.000   Min.   : 0.000   Min.   :1990
##  1st Qu.: 0.0000   1st Qu.: 0.000   1st Qu.: 0.000   1st Qu.:1999
##  Median : 0.0000   Median : 0.000   Median : 0.000   Median :2002
##  Mean   : 0.7165   Mean   : 2.312   Mean   : 1.722   Mean   :2002
##  3rd Qu.: 0.0000   3rd Qu.: 0.000   3rd Qu.: 0.000   3rd Qu.:2005
##  Max.   :48.0000   Max.   :72.000   Max.   :65.000   Max.   :2010
##
##          Path1        RiskAll           season                  Trans1
##  No          :196   Min.   :    0.0         : 56   Environmental   : 11
##  Unspecified:614   1st Qu.:    0.0   Fall  :135   Foodborne       :337
##  Yes         : 40   Median :   21.5   Spring:187   Person to Person:119
##                     Mean   :  359.7   Summer:102   Unknown         : 61
##                     3rd Qu.:  112.0   Winter:370   Unspecified     :261
##                     Max.   :24000.0                Waterborne      : 61
##
##      Vomit              Setting
##  Min.   :0.0000   Other     :682
##  1st Qu.:0.0000   Restaurant:168
##  Median :0.0000
##  Mean   :0.4894
##  3rd Qu.:1.0000
##  Max.   :1.0000
##
```

```
table(CompData$Action1)
```

```
##
##          No Unspecified         Yes
##           1         677         172
```

```
table(CompData$Country)
```

```
##
##   Australia     Austria      Brazil       Ca da         Chi      Croatia
##           0           0           0           0           0           0
##     Denmark      France        Iraq      Israel       Italy       Japan
##           0           0           0           0           0         316
##    Multiple Netherlands New Zealand      Norway       Other    Scotland
##          16           0           0           0         408           0
##       Spain          UK Unspecified         USA
##           0           0          12          98
```

```
table(CompData$Trans1)
```

```
##
##    Environmental        Foodborne Person to Person          Unknown
##               11              337              119               61
##      Unspecified       Waterborne
##              261               61
```

You should see from your explorations above that there is only a single entry for *No* in `Action1`, and small entries for *Multiple* and *Unspecified* in `Country` and *Environmental* in `Trans1`. The single *No* should be fixed, the other somewhat small groupings might be ok. It depends on scientific rationale and method of analysis if you should modify/group those or not. We'll do that.

Change things as follows: Remove the observation for which action is *No*, combine Countries into 3 groups Japan/USA/Other, and since I don't even know what biologically the difference is between *Environmental* and *Waterborne* transmission (seems the same route to me based on my norovirus knowledge), we'll move the Waterborne into the environmental. Finally, I noticed there is a blank category for season. That likely means it wasn't stated in the paper. Let's recode that as *Unknown*. Finally, re-order data such that the outcome is the first column and again remove empty factor levels. Then look at your resulting data frame.

```
#write code that does the actions described above
#remove no
remove_no <- CompData %>%

  subset(Action1 != "No")

remove_no <- remove_no %>%

  droplevels(remove_no$Action1)

summary(remove_no$Action1)
```

```
## Unspecified         Yes
##         677         172
```

```
#recode country

summary(remove_no$Country)
```

```
##   Australia     Austria      Brazil       Ca da         Chi      Croatia
##           0           0           0           0           0           0
##     Denmark      France        Iraq      Israel       Italy       Japan
##           0           0           0           0           0         316
##    Multiple Netherlands New Zealand      Norway       Other    Scotland
##          16           0           0           0         408           0
##       Spain          UK Unspecified         USA
##           0           0          11          98
```

```
Multiple<- remove_no %>%

  mutate(Country= fct_recode(Country, "Other" = "Multiple"))



Unspec <- Multiple %>%

  mutate(Country= fct_recode(Country, "Other" = "Unspecified"))
```

```
summary(Unspec$Country)
```

```
##   Australia     Austria      Brazil       Ca da         Chi      Croatia
##           0           0           0           0           0           0
##     Denmark      France        Iraq      Israel       Italy       Japan
##           0           0           0           0           0         316
##       Other Netherlands New Zealand      Norway    Scotland       Spain
##         435           0           0           0           0           0
##          UK         USA
##           0          98
```

```
summary(Unspec$Trans1)
```

```
##    Environmental        Foodborne Person to Person          Unknown
##               11              337              119               61
##      Unspecified       Waterborne
##              260               61
```

```
#environemntal


Environmental <- Unspec %>%

  mutate(Trans1= fct_recode(Trans1, "Environmental" = "Waterborne"))

summary(Environmental$Trans1)
```

```
##    Environmental        Foodborne Person to Person          Unknown
##               72              337              119               61
##      Unspecified
##              260
```

```
#Seasons

summary(Environmental$season)
```

```
##          Fall Spring Summer Winter
##     55    135    187    102    370
```

```
Season <- Environmental %>%

  mutate(season= fct_recode(season, "Unknown" = ""))

summary(Season$season)
```

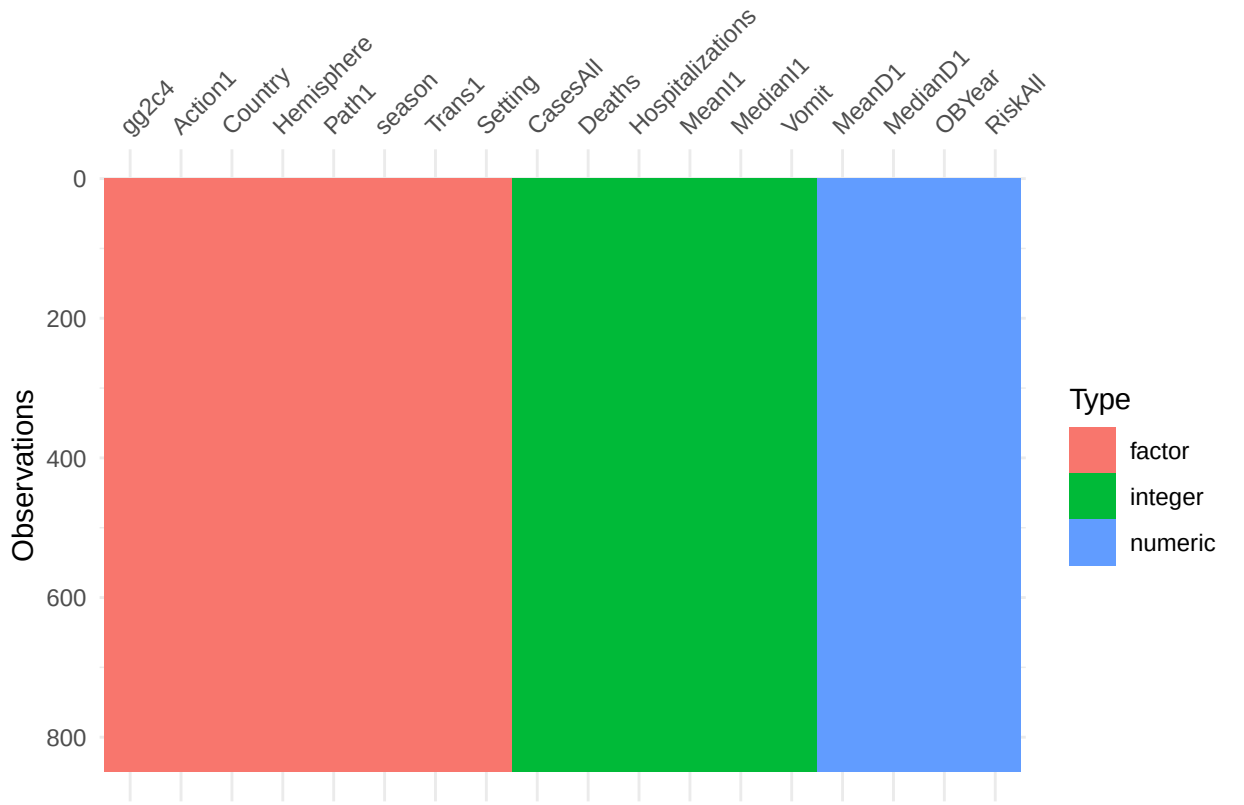
```
## Unknown    Fall  Spring  Summer  Winter
##      55     135     187     102     370
```

```
#reorder outcome
Outcomefirst <- Season %>%

  select(gg2c4, everything())

summary(Outcomefirst)
```

```
##  gg2c4            Action1       CasesAll            Country
```

```
##  No :590   Unspecified:677   Min.   :    1.0   Other    :435
##  Yes:259   Yes        :172   1st Qu.:    9.0   Japan    :316
##                              Median :   25.0   USA      : 98
##                              Mean   :  128.1   Australia:  0
##                              3rd Qu.:   65.0   Austria  :  0
##                              Max.   :32150.0   Brazil   :  0
##                                                (Other)  :  0
##      Deaths           Hemisphere  Hospitalizations      MeanD1
##  Min.   :0.00000   Northern:797   Min.   : 0.0000   Min.   : 0.000
##  1st Qu.:0.00000   Southern: 52   1st Qu.: 0.0000   1st Qu.: 0.000
##  Median :0.00000                  Median : 0.0000   Median : 0.000
##  Mean   :0.05183                  Mean   : 0.5524   Mean   : 1.312
##  3rd Qu.:0.00000                  3rd Qu.: 0.0000   3rd Qu.: 0.000
##  Max.   :9.00000                  Max.   :99.0000   Max.   :96.000
##
##      MeanI1           MedianD1         MedianI1          OBYear
##  Min.   : 0.0000   Min.   : 0.000   Min.   : 0.000   Min.   :1990
##  1st Qu.: 0.0000   1st Qu.: 0.000   1st Qu.: 0.000   1st Qu.:1999
##  Median : 0.0000   Median : 0.000   Median : 0.000   Median :2002
##  Mean   : 0.7173   Mean   : 2.314   Mean   : 1.724   Mean   :2002
##  3rd Qu.: 0.0000   3rd Qu.: 0.000   3rd Qu.: 0.000   3rd Qu.:2005
##  Max.   :48.0000   Max.   :72.000   Max.   :65.000   Max.   :2010
##
##          Path1        RiskAll            season                  Trans1
##  No         :196   Min.   :    0.0   Unknown: 55   Environmental   : 72
##  Unspecified:613   1st Qu.:    0.0   Fall   :135   Foodborne       :337
##  Yes        : 40   Median :   22.0   Spring :187   Person to Person:119
##                    Mean   :  360.1   Summer :102   Unknown         : 61
##                    3rd Qu.:  112.0   Winter :370   Unspecified     :260
##                    Max.   :24000.0
##
##      Vomit              Setting
##  Min.   :0.0000   Other     :681
##  1st Qu.:0.0000   Restaurant:168
##  Median :0.0000
##  Mean   :0.4888
##  3rd Qu.:1.0000
##  Max.   :1.0000
##
```

```
vis_dat(Outcomefirst)
```



At this step, you should have a dataframe containing 850 observations, and 18 variables: 1 outcome, 9 numeric/integer predictors, and 8 factor variables. There should be no missing values. The outcome, `gg2c4`, should be in the 1st slot.

Data splitting

We could do data splitting again as we did in the previous exercise, to have a final test set. But since you saw how it works in the previous exercise, we skip it here. We use the full data to fit to our models. We'll still use cross-validation to get a more honest estimate of model performance. For a real data analysis, the choice to keep some for a final test or not is based on your goals. If your focus is predictive performance, you should consider this split. If your focus is inference or exploratory analysis, you might want to skip this.

Model fitting

So I had planned to use `caret` exclusivley for this course. Last time I tried feature/subset selection with `caret`, I found it buggy and not too well documented. I had hoped this had improved since. Unortunately, I was again not really able to get things to work. So even though I said we likely won't use the `mlr` package, I decided that to be able to nicely practice feature/subset selection, we need to do so. It's not a bad idea to get familiar with that package, at times `caret` can do things `mlr` can't do and the reverse. So knowing how to use both is good. We'll thus use `mlr` to do our model fitting in this exercise.

Parallelization

`mlr` allows you to run things in parallel by using multiple cores (`caret` does too). For instance, if you do 5x cross-validation 5x repeated, you essentially run a very similar piece of code 25 times. Normally, you would

run one at a time. But if you have it on a machine with 25 cores, all those 25 could run at the same time, giving you a speed-up of 25 (or close to, there is usually some overhead related to doing things in parallel). This kind of parallel computing is sometimes called *embarassingly parallel*, because it's so embarassingly simple to split the task into parallel streams.

Since doing the subset selection below starts getting slow, and because it's a good topic to know about, we are going to use parallelization here. `mlr` uses the package `parallelMap` for this. All you need to do is specify the number of cores/processors you want to use and start the parallel system, and `mlr` then automatically does things in parallel if feasible.

```
#set the number of cores you want to use. Note that this is actually the number of 'logical processors'
ncpu=4;
#if you don't want to run things in parallel, or don't have multiple cores (unlikely nowadays),
#just comment out the line below.
parallelStartSocket(ncpu, show.info=FALSE)
```

Setup

Some setup settings that are used in various code chunks below.

```
outcome <- Outcomefirst$gg2c4
outcomename = "gg2c4"
predictors <- Outcomefirst[,-1]
npred=ncol(predictors)
#set sampling method for performance evaluation
#here, we use 5-fold cross-validation, 5-times repeated
sampling_choice = makeResampleDesc("RepCV", reps = 5, folds = 5)
```

A null model

To define a null model, we need to determine what performance measure we want to track. As mentioned in the course materials, there are different performance measures. Accuracy or misclassification error is simple, it just counts the number of times the model got it right/wrong. We'll start with that one, and then try another one later. `mlr` allows for a lot of different performance measures for both categorical and continuous outcomes, see here and here.

For accuracy, the simplest null model always predicts the most frequent category. We can use that as baseline performance.

```
#write code that computes accuracy for a null model
#the null model always predicts "No"
measureACC("No", Outcomefirst$gg2c4)
```

```
## [1] 0.6949352
```

You should find that the null model has an accuracy of around 0.69.

Single predictor models

Now let's consider single predictor models, i.e. we'll fit the outcome to each predictor one at a time to get an idea of the importance of individual predictors. To evaluate our model performance, we will use cross-validation. Since our outcome is categorical, we'll use a logistic model.

```
set.seed(1111) #makes each code block reproducible
#set learner/model. this corresponds to a logistic model.
#mlr calls different models different "learners"
learner_name = "classif.binomial";
```

```
mylearner = makeLearner(learner_name, predict.type = "prob")
# this will contain the results
unifmat=data.frame(variable = rep(0,npred), Accuracy = rep(0,npred))
# loop over each predictor, build simple dataset with just outcome and that predictor, fit it to a glm/
for (nn in 1:npred)
{
    unidata = data.frame(gg2c4 = outcome, Outcomefirst[,nn+1] )
    ## Generate the task, i.e. define outcome and predictors to be fit
    mytask = makeClassifTask(id='unianalysis', data = unidata, target = outcomename, positive = "Yes")
    model = resample(mylearner, task = mytask, resampling = sampling_choice, show.info = FALSE, measures
    unifmat[nn,1] = names(predictors)[nn]
    unifmat[nn,2] = model$aggr
}
kable(unifmat)
```

variable	Accuracy
Action1	0.6949419
CasesAll	0.6932711
Country	0.6949447
Deaths	0.6977905
Hemisphere	0.6949502
Hospitalizations	0.6941998
MeanD1	0.6946940
MeanI1	0.6949419
MedianD1	0.6949516
MedianI1	0.6949488
OBYear	0.6941970
Path1	0.6949279
RiskAll	0.6956408
season	0.6930609
Trans1	0.6949377
Vomit	0.6949572
Setting	0.6949126

So looks like none of the single predictor models have a higher accuracy than the null. Maybe surprising. We'll get back to that below.

Full model

Now let's fit a full logistic model with all predictors.

```
set.seed(1111) #makes each code block reproducible
#do full model with Cross-Validation - to get an idea for the amount of over-fitting a full model does
mytask = makeClassifTask(id='fullanalysis', data = Outcomefirst, target = outcomename, positive = "Yes"
```

```
## Warning in makeTask(type = type, data = data, weights = weights, blocking =
## blocking, : Empty factor levels were dropped for columns: Country
```

```
fullmodel = resample(mylearner, task = mytask, resampling = sampling_choice, show.info = FALSE, measures
ACC_fullmodel = fullmodel$aggr[1]
print(ACC_fullmodel)
```

```
## acc.test.mean
```

```
##     0.7060188
```

So a very small improvement over the simple models.

Now let's do subset selection. The code below does it several ways. It does regular forward and backward selection and floating versions of those. It also uses a genetic algorithm for selection. See the `mlr` website here for details.

Also note that I included a kind of timer in the code, to see how long things take. That's a good idea if you run bigger models. You first run a few iterations on maybe a few cores, then you can compute how long it would take if you doubled the iterations, or doubled the cores, etc. This prevents bad surprises of trying to *quickly* run a piece of code and waiting hours. You should always use short runs to make sure everything works in principle, and only at the end do the real, long "production" runs. Otherwise you might waste hours/days/weeks waiting for results only to realize that you made a basic mistake and you have to do it all over.

```
set.seed(1111)
tstart=proc.time(); #capture current CPU time for timing how long things take
#do 2 forms of forward and backward selection, just to compare
select_methods=c("sbs","sfbs","sfs","sffs")
resmat=data.frame(method = rep(0,4), Accuracy = rep(0,4), Model = rep(0,4))
ct=1;
for (select_method in select_methods) #loop over all stepwise selection methods
{
  ctrl = makeFeatSelControlSequential(method = select_method)
  print(sprintf('doing subset selection with method %s ',select_method))
  sfeat_res = selectFeatures(learner = mylearner,
                             task = mytask,
                             resampling = sampling_choice,
                             control = ctrl,
                             show.info = FALSE,
                             measures = acc)

  resmat[ct,1] = select_methods[ct]
  resmat[ct,2] = sfeat_res$y
  resmat[ct,3] = paste(as.vector(sfeat_res$x), collapse= ', ')
  ct=ct+1;
}
```

```
## [1] "doing subset selection with method sbs "
## [1] "doing subset selection with method sfbs "
## [1] "doing subset selection with method sfs "
## [1] "doing subset selection with method sffs "
```

```
# do feature selection with genetic algorithm
maxit = 100 #number of iterations - should be large for 'production run'
ctrl_GA =makeFeatSelControlGA(maxit = maxit)
print(sprintf('doing subset selection with genetic algorithm'))
```

```
## [1] "doing subset selection with genetic algorithm"
```

```
sfeatga_res = selectFeatures(learner = mylearner,
                                 task = mytask,
                                 resampling = sampling_choice,
                                 control = ctrl_GA,
                                 show.info = FALSE,
                                 measures = acc)
```

```
resmat[5,1] = "GA"
resmat[5,2] = sfeatga_res$y
resmat[5,3] = paste(as.vector(sfeatga_res$x), collapse= ', ')
runtime.minutes_SS=(proc.time()-tstart)[[3]]/60; #total time in minutes the optimization took
print(sprintf('subset selection took %f minutes',runtime.minutes_SS));
```

```
## [1] "subset selection took 1.976833 minutes"
```

```
kable(resmat)
```

method	Accuracy	Model
sbs	0.7203857	Action1, Country, Deaths, MedianI1, OBYear, RiskAll, season, Trans1, Vomit, Setting
sfbs	0.7187539	Action1, Country, Deaths, OBYear, Path1, season, Trans1, Vomit, Setting
sfs	0.6949447	
sffs	0.6949669	
GA	0.7147567	Action1, CasesAll, Country, Deaths, OBYear, Path1, RiskAll, season, Trans1, Vomit, Setting

So we find that some of the sub-models perform somewhat better. Note that the different methods find different submodels and the forward selection method seems to fail (since none of the single-predictor models is better than the null, it stops there). Also note that the genetic algorithm was only allowed to run for a short time (the number if iterations is small). To get potentially better results, that should be increased to a larger number, e.g. 1000 or more. But this will take a longer time, unless you have many cores you can use when you parallelize.

Different performance measures

So far, we found that some of the sub-models provide a minor improvement over the simple null or single-predictor models or the full model. However, the improvement is not much.

We could try tweaking further, e.g. by pre-processing the predictors (which is a good idea to try, but we won't do here) or testing a different model (which we'll do below).

For now, I want to discuss performance measure a bit more. The problem is that logistic regression, as well as many (though not all) algorithms that are used for classification (predicting categorical outcomes) return probabilities for an outcome to belong to a certain category. Those are then turned into Yes/No predictions (in the case with 2 outcomes like we have here), based on some cut-off. By default, a probability value of 0.5 is chosen. If the model predicts that there is a 0.5 or greater probability of the outcome being "Yes", it is scored as such in the prediction, otherwise as no. However, this 0.5 threshold is not always good. It could be that we might get a much better model if we use another threshold. There is a technique that scans over different thresholds and evaluates model performance for each threshold. This is generally called *receiver operating curve*. If this concept is new to you, check out this article on Wikipedia or this tutorial on the mlr website.

Using this approach, a model is evaluated based on the Area under the curve (AUC), with an AUC=0.5 meaning a model is not better than chance, and AUC=1 meaning a perfect model. Let's re-do the analysis above, but replace accuracy with AUC.

Model performance with AUC

```
#write code that computes AUC for a null model
#the null model always predicts "No"
#the auc function in the pROC package is useful
#or you can use any other package/function you want to use to compute AUC/ROC
```

```
nullmod <- roc(Outcomefirst$gg2c4, rep(1,849))
```

```
## Setting levels: control = No, case = Yes
```

```
## Setting direction: controls < cases
```

```
auc(nullmod)
```

```
## Area under the curve: 0.5
```

You should find that the null model has an AUC of 0.5, as expected for a "no information" model.

Now we run the single-predictor models again.

```
#copy and paste the code from above for single predictor fits, but now switch to AUC instead of accurac
set.seed(1111) #makes each code block reproducible
#set learner/model. this corresponds to a logistic model.
#mlr calls different models different "learners"
learner_name = "classif.binomial";
mylearner = makeLearner(learner_name, predict.type = "prob")
# this will contain the results
unifmat=data.frame(variable = rep(0,npred), Accuracy = rep(0,npred))
# loop over each predictor, build simple dataset with just outcome and that predictor, fit it to a glm/
for (nn in 1:npred)
{
    unidata = data.frame(gg2c4 = outcome, Outcomefirst[,nn+1] )
    ## Generate the task, i.e. define outcome and predictors to be fit
    mytask = makeClassifTask(id='unianalysis', data = unidata, target = outcomename, positive = "Yes")
    model = resample(mylearner, task = mytask, resampling = sampling_choice, show.info = FALSE, measures
    unifmat[nn,1] = names(predictors)[nn]
    unifmat[nn,2] = model$aggr
}
kable(unifmat)
```

variable	Accuracy
Action1	0.5086751
CasesAll	0.4464498
Country	0.5807351
Deaths	0.5161977
Hemisphere	0.5076597
Hospitalizations	0.5031918
MeanD1	0.4949457
MeanI1	0.5014621
MedianD1	0.5172409
MedianI1	0.5292838
OBYear	0.5979801
Path1	0.5130772
RiskAll	0.5705659
season	0.5862556
Trans1	0.6045373
Vomit	0.5294234
Setting	0.5625620

Looks like there are a few single-predictor models with better performance than the null model.

Now let's again fit a full model with all predictors.

```
#copy and paste the code from above for the full model fit, but now switch to AUC instead of accuracy.
set.seed(1111) #makes each code block reproducible
#do full model with Cross-Validation - to get an idea for the amount of over-fitting a full model does
mytask = makeClassifTask(id='fullanalysis', data = Outcomefirst, target = outcomename, positive = "Yes")
```

```
## Warning in makeTask(type = type, data = data, weights = weights, blocking =
## blocking, : Empty factor levels were dropped for columns: Country
```

```
fullmodel = resample(mylearner, task = mytask, resampling = sampling_choice, show.info = FALSE, measures
AUC_fullmodel = fullmodel$aggr[1]
print(AUC_fullmodel)
```

```
## auc.test.mean
##      0.672484
```

This is better than any single-predictor models. Let's see if a sub-model can do even better.

```
#copy and paste the code from above for the subset selection, but now switch to AUC instead of accuracy
set.seed(1111)
tstart=proc.time(); #capture current CPU time for timing how long things take
#do 2 forms of forward and backward selection, just to compare
select_methods=c("sbs","sfbs","sfs","sffs")
resmat=data.frame(method = rep(0,4), Accuracy = rep(0,4), Model = rep(0,4))
ct=1;
for (select_method in select_methods) #loop over all stepwise selection methods
{
  ctrl = makeFeatSelControlSequential(method = select_method)
  print(sprintf('doing subset selection with method %s ',select_method))
  sfeat_res = selectFeatures(learner = mylearner,
                             task = mytask,
                             resampling = sampling_choice,
                             control = ctrl,
                             show.info = FALSE,
                             measures = mlr :: auc)

  resmat[ct,1] = select_methods[ct]
  resmat[ct,2] = sfeat_res$y
  resmat[ct,3] = paste(as.vector(sfeat_res$x), collapse= ', ')
  ct=ct+1;
}
```

```
## [1] "doing subset selection with method sbs "
## [1] "doing subset selection with method sfbs "
## [1] "doing subset selection with method sfs "
## [1] "doing subset selection with method sffs "
```

```
# do feature selection with genetic algorithm
maxit = 100 #number of iterations - should be large for 'production run'
ctrl_GA =makeFeatSelControlGA(maxit = maxit)
print(sprintf('doing subset selection with genetic algorithm'))
```

```
## [1] "doing subset selection with genetic algorithm"
```

```
sfeatga_res = selectFeatures(learner = mylearner,
                                  task = mytask,
                                  resampling = sampling_choice,
```

```
                                    control = ctrl_GA,
                                    show.info = FALSE,
                                    measures = mlr :: auc)
resmat[5,1] = "GA"
resmat[5,2] = sfeatga_res$y
resmat[5,3] = paste(as.vector(sfeatga_res$x), collapse= ', ')
runtime.minutes_SS=(proc.time()-tstart)[[3]]/60; #total time in minutes the optimization took
print(sprintf('subset selection took %f minutes',runtime.minutes_SS));
```

```
## [1] "subset selection took 2.232000 minutes"
```

```
kable(resmat)
```

method	Accuracy	Model
sbs	0.6885835	Action1, Country, Deaths, MedianI1, OBYear, Path1, RiskAll, season, Trans1, Setting
sfbs	0.6855044	Action1, Country, Deaths, MedianI1, OBYear, Path1, season, Trans1, Setting
sfs	0.6616717	Country, OBYear, season, Trans1
sffs	0.6798470	Action1, Country, OBYear, season, Trans1, Setting
GA	0.6854682	Action1, Country, Deaths, MedianI1, OBYear, Path1, RiskAll, season, Trans1, Setting

Looks like we get some sub-models that are slightly better than the full model, at least as evaluated using the (cross-validated) AUC measure.

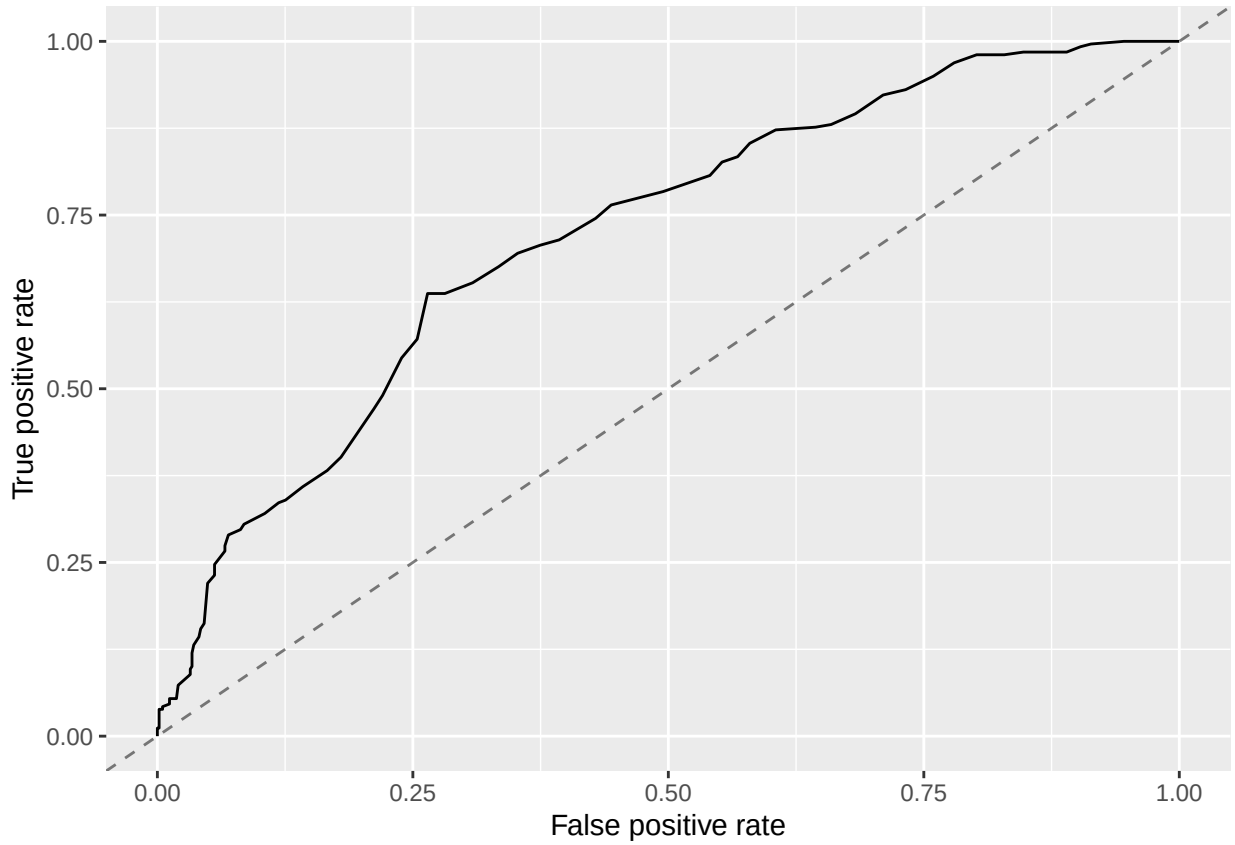
More on ROC/AUC

While this is a common measure and useful to evaluate how models perform at different cut-off levels, one needs to keep in mind that this doesn't really measure the performance of a single model, but instead the combined performance of all models with different cut-offs. In practice, if we want to use a model, we have to pick one cut-off. For that purpose, one can look at the whole ROC curve and one then usually chooses the model in the "top-left" corner of the curve, which gives the best overall performance with a good balance between FP and TP. Let's plot that ROC curve for the model found by the GA.

```
set.seed(1111) #makes each code block reproducible
d2 <- Outcomefirst %>% dplyr::select(gg2c4, sfeatga_res$x )
mytask = makeClassifTask(id='rocanalysis', data = d2, target = outcomename, positive = "Yes")
```

```
## Warning in makeTask(type = type, data = data, weights = weights, blocking =
## blocking, : Empty factor levels were dropped for columns: Country
```

```
log_mod = train(mylearner, mytask)
log_pred = predict(log_mod, task = mytask)
df = generateThreshVsPerfData(list(logistic = log_pred), measures = list(fpr, tpr))
plotROCCurves(df)
```



Based on this plot, a FP of around 0.5 an TP of around 0.75 seem to produce a good model. Digging into the `df` object, you can find the raw data underlying the curve in `df$data` and will find

```
print(df$data[29:31,])
```

```
##          fpr       tpr threshold
## 29 0.4288136 0.7451737 0.2828283
## 30 0.3932203 0.7142857 0.2929293
## 31 0.3745763 0.7065637 0.3030303
```

This means for this model, if we were to use it we might want to set a threshold of around 0.3, i.e. everything above that probability would be predicted as positive/yes.

Trying a different model

Let's see if we can find another model that might perform even better. Let's look at a Linear Discriminant Analysis (LDA) model. See e.g. chapter 4 of ISLR for more on that model. We again consider AUC as our measure. The null-model performance is the same, we need to re-do the remainder.

Let's re-run the single model, followed by the full model and subset selection.

```
#copy and paste the code from above for single predictor fits. Switch the learner from a the logistic m
set.seed(1111) #makes each code block reproducible
#set learner/model. this corresponds to a logistic model.
#mlr calls different models different "learners"
learner_name = "classif.lda";
mylearner = makeLearner(learner_name, predict.type = "prob")
# this will contain the results
```

```
unifmat=data.frame(variable = rep(0,npred), Accuracy = rep(0,npred))
# loop over each predictor, build simple dataset with just outcome and that predictor, fit it to a glm/
for (nn in 1:npred)
{
    unidata = data.frame(gg2c4 = outcome, Outcomefirst[,nn+1] )
    ## Generate the task, i.e. define outcome and predictors to be fit
    mytask = makeClassifTask(id='unianalysis', data = unidata, target = outcomename, positive = "Yes")
    model = resample(mylearner, task = mytask, resampling = sampling_choice, show.info = FALSE, measures
    unifmat[nn,1] = names(predictors)[nn]
    unifmat[nn,2] = model$aggr
}
kable(unifmat)
```

variable	Accuracy
Action1	0.5086751
CasesAll	0.4464498
Country	0.5807351
Deaths	0.5161977
Hemisphere	0.5076597
Hospitalizations	0.5031918
MeanD1	0.4949457
MeanI1	0.5014621
MedianD1	0.5172409
MedianI1	0.5292838
OBYear	0.5979801
Path1	0.5130772
RiskAll	0.5705659
season	0.5862556
Trans1	0.6045373
Vomit	0.5294234
Setting	0.5625620

```
#copy and paste the code from above for the full model fit, but now for LDA. Since you already switched
set.seed(1111) #makes each code block reproducible
#do full model with Cross-Validation - to get an idea for the amount of over-fitting a full model does
mytask = makeClassifTask(id='fullanalysis', data = Outcomefirst, target = outcomename, positive = "Yes"
```

```
## Warning in makeTask(type = type, data = data, weights = weights, blocking =
## blocking, : Empty factor levels were dropped for columns: Country
```

```
fullmodel = resample(mylearner, task = mytask, resampling = sampling_choice, show.info = FALSE, measures
AUC_fullmodel = fullmodel$aggr[1]
print(AUC_fullmodel)
```

```
## auc.test.mean
##     0.6737898
```

```
#copy and paste the code from above for subset selection, now for LDA. Since you already switched the l
set.seed(1111)
tstart=proc.time(); #capture current CPU time for timing how long things take
#do 2 forms of forward and backward selection, just to compare
select_methods=c("sbs","sfbs","sfs","sffs")
resmat=data.frame(method = rep(0,4), Accuracy = rep(0,4), Model = rep(0,4))
```

```
ct=1;
for (select_method in select_methods) #loop over all stepwise selection methods
{
  ctrl = makeFeatSelControlSequential(method = select_method)
  print(sprintf('doing subset selection with method %s ',select_method))
  sfeat_res = selectFeatures(learner = mylearner,
                             task = mytask,
                             resampling = sampling_choice,
                             control = ctrl,
                             show.info = FALSE,
                             measures = mlr :: auc)

  resmat[ct,1] = select_methods[ct]
  resmat[ct,2] = sfeat_res$y
  resmat[ct,3] = paste(as.vector(sfeat_res$x), collapse= ', ')
  ct=ct+1;
}
```

```
## [1] "doing subset selection with method sbs "
## [1] "doing subset selection with method sfbs "
## [1] "doing subset selection with method sfs "
## [1] "doing subset selection with method sffs "
```

```
# do feature selection with genetic algorithm
maxit = 100 #number of iterations - should be large for 'production run'
ctrl_GA =makeFeatSelControlGA(maxit = maxit)
print(sprintf('doing subset selection with genetic algorithm'))
```

```
## [1] "doing subset selection with genetic algorithm"
```

```
sfeatga_res = selectFeatures(learner = mylearner,
                                  task = mytask,
                                  resampling = sampling_choice,
                                  control = ctrl_GA,
                                  show.info = FALSE,
                                  measures = mlr :: auc)
resmat[5,1] = "GA"
resmat[5,2] = sfeatga_res$y
resmat[5,3] = paste(as.vector(sfeatga_res$x), collapse= ', ')
runtime.minutes_SS=(proc.time()-tstart)[[3]]/60; #total time in minutes the optimization took
print(sprintf('subset selection took %f minutes',runtime.minutes_SS));
```

```
## [1] "subset selection took 2.046333 minutes"
```

```
kable(resmat)
```

method	Accuracy	Model
sbs	0.6875153	Action1, Country, Deaths, MedianI1, OBYear, RiskAll, season, Trans1, Setting
sfbs	0.6862458	Action1, Country, Deaths, MedianI1, OBYear, RiskAll, season, Trans1, Setting
sfs	0.6625805	Country, OBYear, season, Trans1
sffs	0.6714397	Country, OBYear, season, Trans1, Setting
GA	0.6846441	Action1, Country, Deaths, MedianI1, OBYear, RiskAll, season, Trans1, Setting

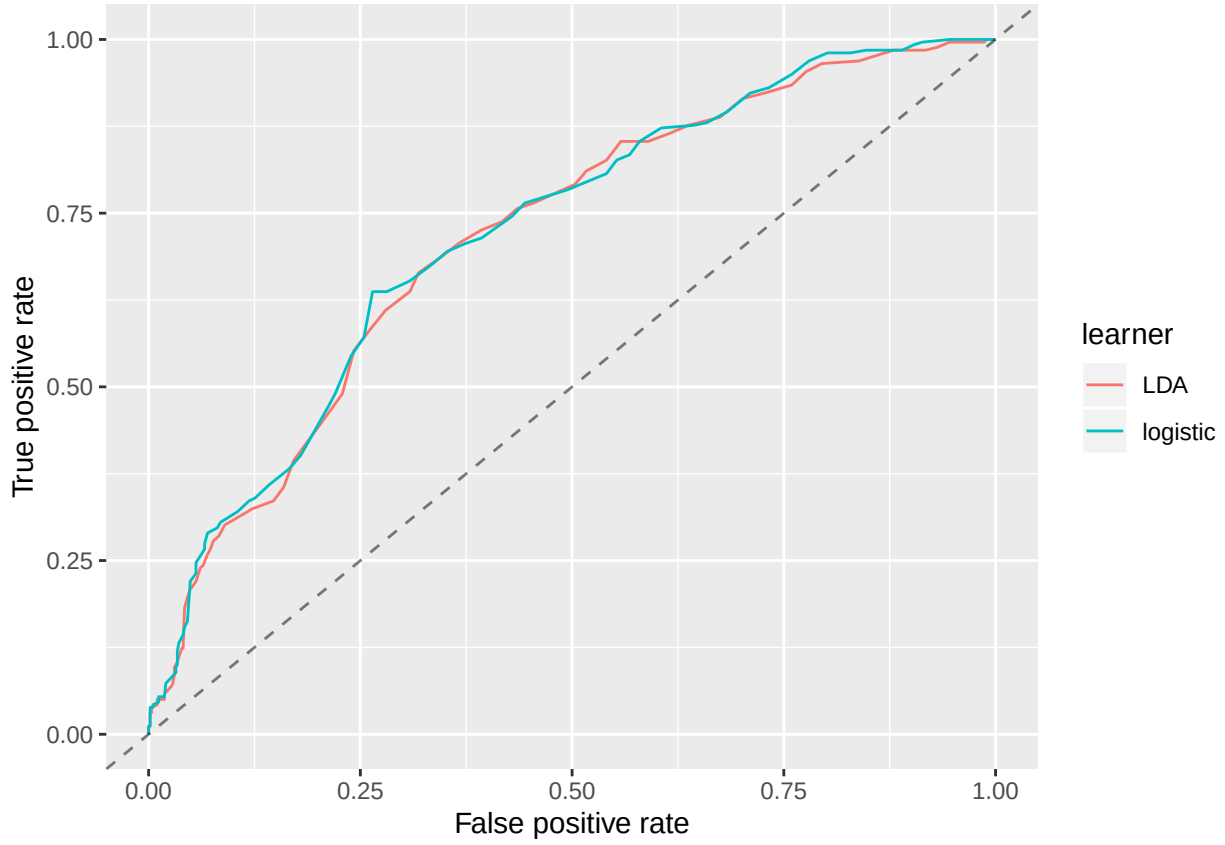
It Looks like we again get some sub-models that are slightly better than the full model, at least as evaluated using the (cross-validated) AUC measure. There seems to be a bit, but not much difference in performance

to the logistic model. We can take the GA model again (because it's conveniently at the end, even if it's not the best) and look at its ROC and compare the curves for the LDA and logistic models.

```
#copy the code from roc-plot above and re-do train/predict byt now save it as lda_mod and lda_pred
#then use generateThreshVsPerfData() with both lda_pred and log_pred (computed above) to create a struc
#best model curves for both logistic and LDA, then plot the curves.
set.seed(1111) #makes each code block reproducible
d2 <- Outcomefirst %>% dplyr::select(gg2c4, sfeatga_res$x )
mytask = makeClassifTask(id='rocanalysis', data = d2, target = outcomename, positive = "Yes")
```

```
## Warning in makeTask(type = type, data = data, weights = weights, blocking =
## blocking, : Empty factor levels were dropped for columns: Country
```

```
lda_mod = train(mylearner, mytask)
lda_pred = predict(lda_mod, task = mytask)
df = generateThreshVsPerfData(list(LDA = lda_pred, logistic = log_pred), measures = list(fpr, tpr))
plotROCCurves(df)
```



You should see from the plot that the LDA model performs very similar to the Logistic model.

```
#close down the parallelization started at the beginning to free up resources.
parallelStop()
```

Wrapping up

Based on the above, we could choose a model simply by best performance. However, we likely also still want to look at model prediction uncertainty and do some residual plots and other diagnostics for our chosen

model before we finalize our choice. As was the case for `caret`, `mlr` also comes with lots of functions that make these - and many other - tasks easier. We'll leave it at this for now, and might revisit some of those topics in further exercises.

Also, none of the models here are that great. We might want to think more about our initial premise, i.e. looking for associations between virus strain type and other variables and what the scientific rationale is for expecting variations. Here, we just ran the model and looked what we might find. That approach, sometimes called *data exploration* or *data mining* or - less charitable *fishing expedition* is ok, but we need to be careful how we interpret and use the results. Going in with a clear hypothesis is usually better.

A few more comments

Things are getting a bit slow now, despite the multiple cores we need minutes to run things. Also, code chunks get larger. At this point, it might be worth considering a structure whereby most of the code lives in one or several **well documented** R scripts, which can be run independently. Those R scripts should save their results, and those results are then loaded here. The advantage is that if one wants to make changes to a small part of the analysis, one can modify and run just that part and update the whole document by re-knitting without having to run all code. For any big/serious analysis, I suggest such a setup that splits R code from RMarkdown files, at least for the computationally intensive parts.