Data Exploration
Week 5

PH 700A, Spring 2025

Rick Calvo

Table of contents

1 Data Exploration

1.1 Packages Introduced

Exploration of Data -> Repairing Issues -> Understanding Your Data

```
library(explore)

library(describe)

library(tidyverse)

library(gtsummary)
```

1.2 Coding Best Practices

- Good maintenance of data objects
- Add comments with \# to keep track of the code's purpose
- Introduce packages at beginning or immediately before its first use
- Limit "hard coding" variables and objects within main functions/programs
- Segregate code into sections w/ labels – use `.qmd` or `.Rmd` documents
- Take only the data you need to create a solution

Note

R code is "read" from top to bottom. Code that is "lower" is run more recent than those that are higher up. Later lines of code can overwrite or undo other code if multiple criteria are true.

1.3 New Concepts

- Additional `dplyr` Commands
- Quantiles
- Handling Dates and Times
- Outlier Detection

1.4 More on `dplyr`

New Commands:

- `select()`
- `filter()`
- `group_by()`
- `arrange()`
- `summarise()`

`select()` Helper Functions:

- `starts_with()`
- `ends_with()`
- `contains()`

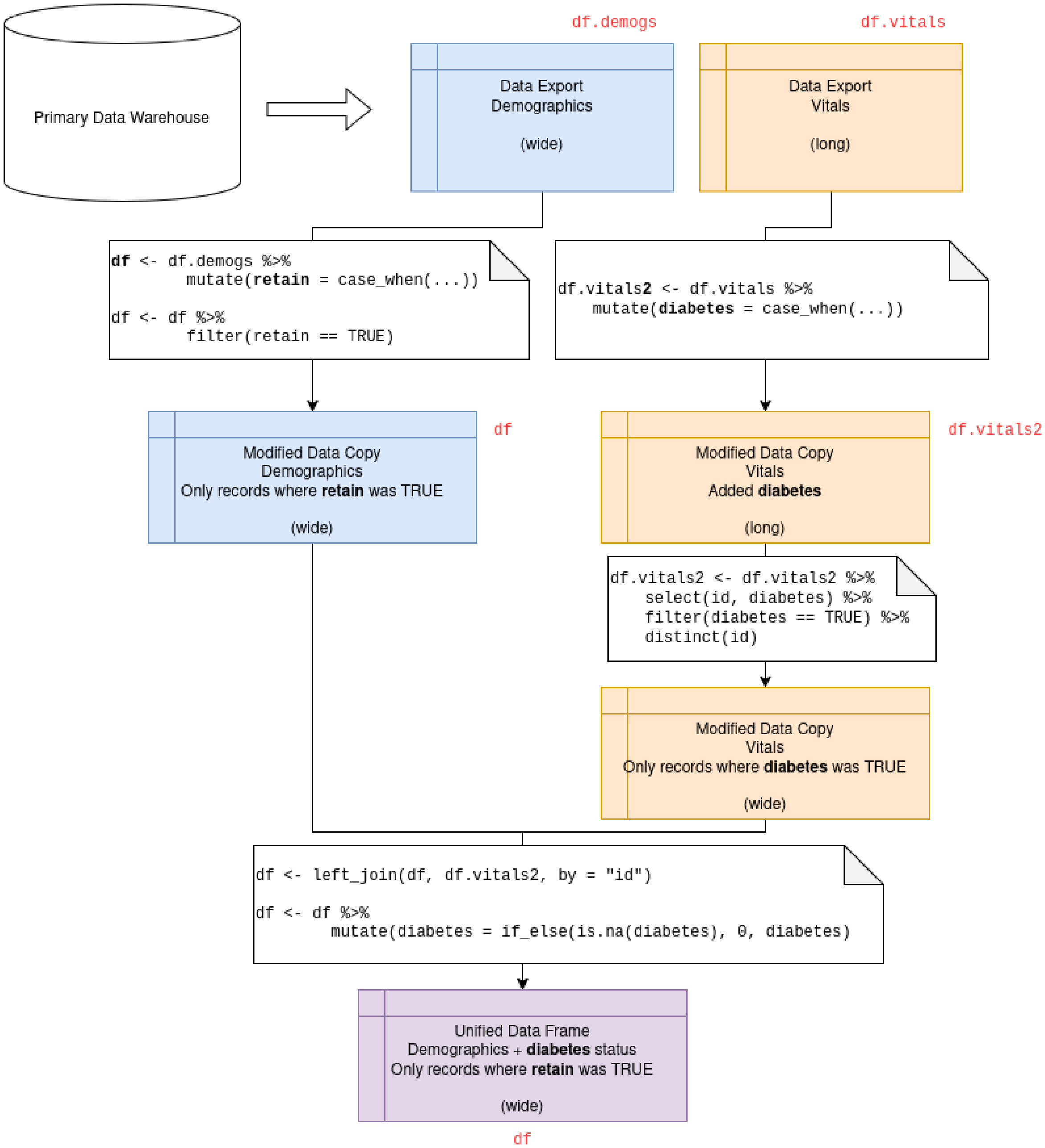
Column-wise Operations:

- `across()`
- `where()`

1.5 Data Maintenance

- *NEVER ALTER RAW DATA!*
- Always create copies to work on when performing any modifications
- This ensures the original data remains preserved
- Data integrity can subsequently be addressed
- Allows for revision of methods
- Retains the value of *you* and *your* code

1.6 Diagram of Data Maintenance



Retain: `df.demog`, `df.vitals`

Remove: `df.vitals2`

Analyze: `df`

1.7 Basic Exploration

Using `table()`, `str()`, `library(explore)`, `library(describe)`, and `library(gtsummary)`, we should be able to get a sense of what variables are present and their respective values.

Identifying univariates and bivariate relationships improve our understanding of our data so that we can issue the right modifications prior to analysis.

1.8 Column Select Recap

```
df2 <- df %>%
  select(a, b, c)
```

- Creates a data frame copy called `df2` from `df`
- `df2` will only contain columns `a`, `b`, and `c`

```
df3 <- df %>%
  select(-d)
```

- Creates a data frame copy called `df3` from `df`
- `df3` will contain all columns from `df` except `d`

1.9 Column Selection Additional Functions

```
df2 <- df %>% select(starts_with("dx_"))
```

- `df2` will only contain the columns that start with "dx_" from `df`

```
df3 <- df %>% select(ends_with("_sum"))
```

- `df3` will only contain the columns that end in "_sum" from `df`

```
df4 <- df %>% select(contains("bloodunits"))
```

- `df4` will only contain columns that contain the full string "bloodunits" in the column name from `df`

1.10 Column Selection Example Usage

```
df.vitals2 <- df.vitals %>%
  select(id, ends_with("admit")) %>%
  mutate(hypotension = if_else(sbp_admit < 90, 1, 0),
         octogenarian = if_else(age_admit >=80, 1, 0),
         neuroderange = if_else(gcstotal_admit < 8, 1, 0,
         diabetes = if_else(fpg_admit >= 126, 1, 0))
        )
```

1.11 Data Frame Restriction w/ `filter()`

The `filter()` command retains only the observations that satisfy the condition.

Essentially the `tidyverse` version of `subset()`

```
df.kids <- df %>%
  filter(age < 18)
```

This will create a new data frame `df.kids` that only contains observations with an `age` less than 18.

```
df.kids.male <- df %>%
  filter(age < 18 & sex == "M")
```

This will create a new data frame `df.kids.male` that only contains observations with an `age` less than 18 and who had a value for `sex` that is "M".

1.12 Grouping Common Data

Observations that have at least one common characteristic can be grouped before additional processing.

This may be important for long data that needs to be collapsed or sorted by a unique identifier.

```
df2 <- df %>%
  group_by(id) %>%
  mutate(meansbp = mean(sbp)) %>%
  ungroup()
```

- This code groups rows based on having the same `id` value and then calculates a mean sbp based on all the values of `sbp` for each unique `id`
- This does not modify the number of observations present in the data and all observations of the same `id` will have the same `meansbp`
- The `ungroup()` command must be *piped* at the end to remove the grouping schema
- Neglecting the `ungroup()` will retain the grouping system by `id` for any subsequent procedure whether intended or not

1.13 Sorting Data

The `arrange()` function allows you to order observations by *ascending* (`asc()`) or *descending* (`desc()`)values.

- `NA` are always at the end regardless of ordering preference

```
df <- df %>%
  arrange(admitdate)
```

This will sort the entire data frame by `admitdate` in ascending order by default.

1.14 Combining `arrange` and `group_by`

If order needs to be performed by a unique grouping, both `arrange()` and `group_by()` can be used together.

```
df <- df %>%
  group_by(id) %>%
  arrange(admitdate)
```

This will first group observations based on common `id` and then put each respective row in ascending order according to `admitdate`

1.15 Summarise

The `summarise()` command collapses data from multiple observations.

Can be combined with `group_by()` to distill data down to a single observation per unique grouping value.

```
df.summarized <- df.long %>%
  group_by(id) %>%
  summarise(across(contains("admit"),
    list(
      obs  = length,
      mean = mean,
      sd   = sd
      )))
```

Creates a new data frame `df.summarized` that contains collapsed statistics for number of observations (`obs`), means (`mean`), and standard deviations (`sd`) for all variables that contain the string, "admit" in their column name.

1.16 Mutating Multiple Columns

Using `across()`

- Applies a function `across` multiple columns in a data frame

```
columnList <- c("var1", "var2", "var3")

df <- df %>%
      mutate(across(columnList, function))
```

This code will perform a `function` of your choice across all variables in `columnList`. Functions could include any of the other commands we discussed (i.e. `coalesce`, `case_when`, `if_else`, `as.factor`, `as.numeric`, `as.ordered`, `as.string`, or others).

1.17 Mutate Across Example Usage

```
columns_to_fix <- c("age", "sbp", "hlos")

df <- df %>%
  mutate(across(columns_to_fix, as.numeric))
```

This will convert the columns in the list to numeric type.

1.18 Additional Column-wise Conditions

Conditions can be added to broadly apply a function to a data frame using `where()`.

```
df <- df %>%
  mutate(across(where(is.character), as.numeric))
```

Mutates ALL columns to *numeric* if they are currently of the *character* type.

1.19 Quantile Generation

Quickly convert a continuous variable to categories.

`df$quantiles <- ntile(df$variable, x)`

- `ntile()` is the command
- `df$variable` is a continuous variable that you want to quantile
- `x` is the number of quantiles
- `df$quantiles` is the new variable with the quantile labels

1.20 Working With Dates and Times

The concepts of `date`, `time`, `datetime`, and *elapsed time* are unique and malleable.

Dates and times have a unique type of variable that differs from standard numerics.

Dates can be displayed in a variety of forms.

Examples of the same date:

`January 10, 2020`

`01/10/2020`

`10jan2020`

`01-10-20`

`10/01/20`

String dates must be converted to "time-numerics" before analysis. The following symbols represent parts of a date string that can be used to tell R what part of a string refers to a part of a date.

Symbol	Definition	Example
%d	Numeric day	10
%a	Abbr. weekday	Mon
%A	Unabbr. weekday	Monday
%m	Numeric month	02
%b	Abbr. month	Feb
%B	Unabbr. month	February
%y	2-digit year	24
%Y	4-digit year	2024

1.21 Converting Date Variables

Computers count in Unix time starting on 00:00:00 UTC on 1 January 1970. Variables containing dates and times must be converted to a number the computers can understand.

1.21.1 Using `as.Date` to convert *from string to date w/ slash delimiter*

```
df.dateNum <- as.Date(df$dateStr, format = %m/%d/%Y)
```

Changes the format of an existing string date to numeric based on month, day, and 4-digit year and delimited by "/".

1.21.2 Using `as.POSIXct` to convert *from string to datetime w/ dash, colon, and space delimiters*

```
df$datetimeNumeric <- as.POSIXct(df$datetimeString, format="%Y-%m-%d %H:%M:%S",
tz="UTC")
```

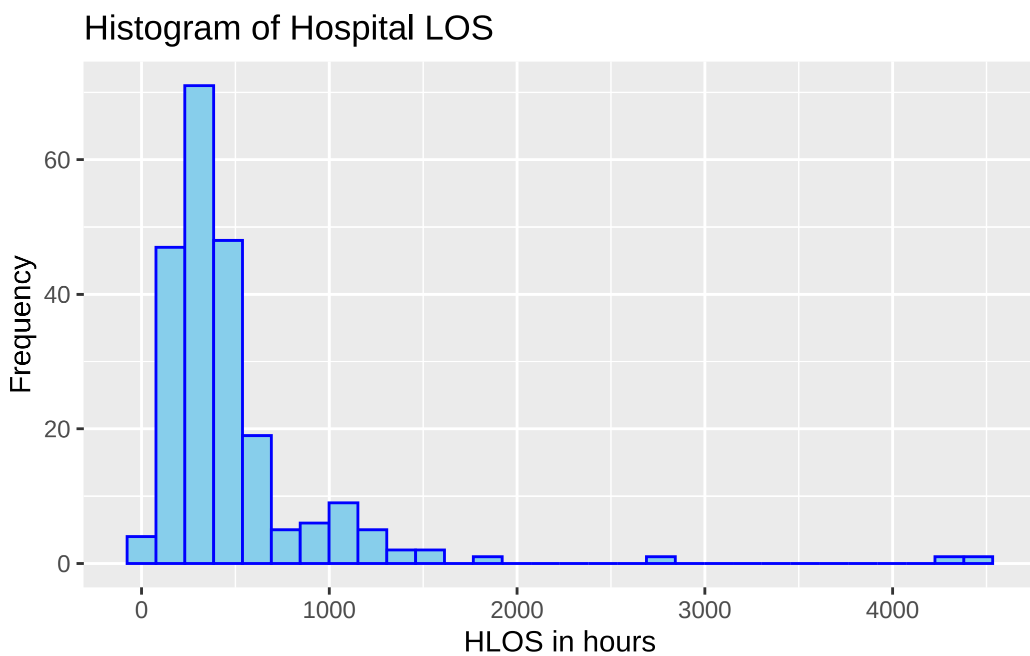
Converts a string in 4-digit year, month, day, hour, minutes, and second order and delimited by dashes for the date, colons for the time, and a space between date and time. Finally sets the timezone to UTC for consistency (unless otherwise known).

> **Note**
>
> POSIX stands for "Portable Operating System Interface" and is a family of standards made to maintain compatibility of a variety of computing measures across all connected devices.

1.22 Outlier Detection

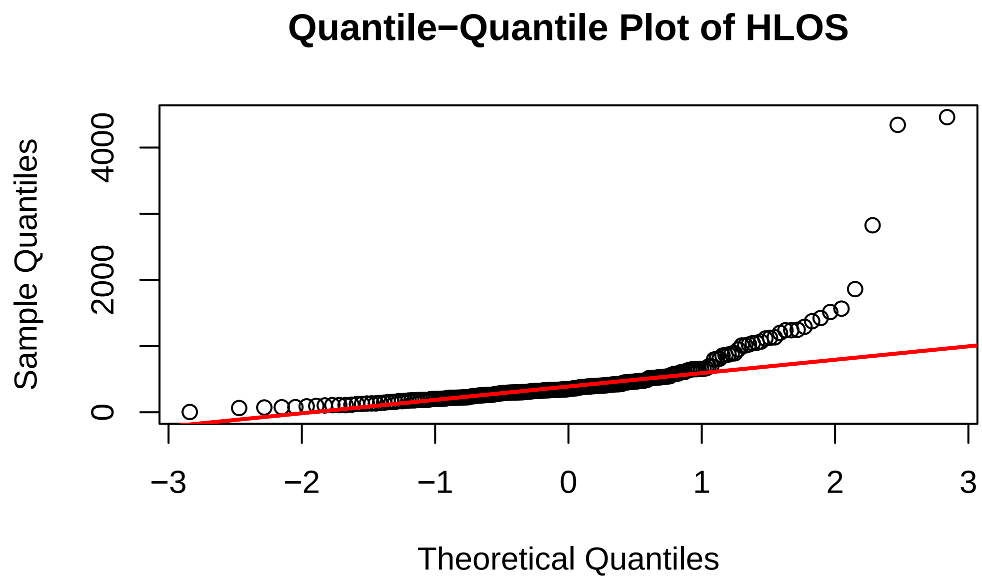
```
library(ggplot2)

ggplot(df.ed,aes(x = hlos)) +
  geom_histogram(bins = 30, color = "blue", fill = "skyblue") +
  labs(title = "Histogram of Hospital LOS", x = "HLOS in hours", y = "Frequency")
```

```
Don't know how to automatically pick scale for object of type <difftime>.
Defaulting to continuous.
```



This is the same image from last week. We want to assess the potential for outliers in HLOS.

```
qqnorm(df.ed$hlos, pch = 1, main = "Quantile-Quantile Plot of HLOS")
  qqline(df.ed$hlos, col = "red", lwd = 2)
```



Also the same as last week.

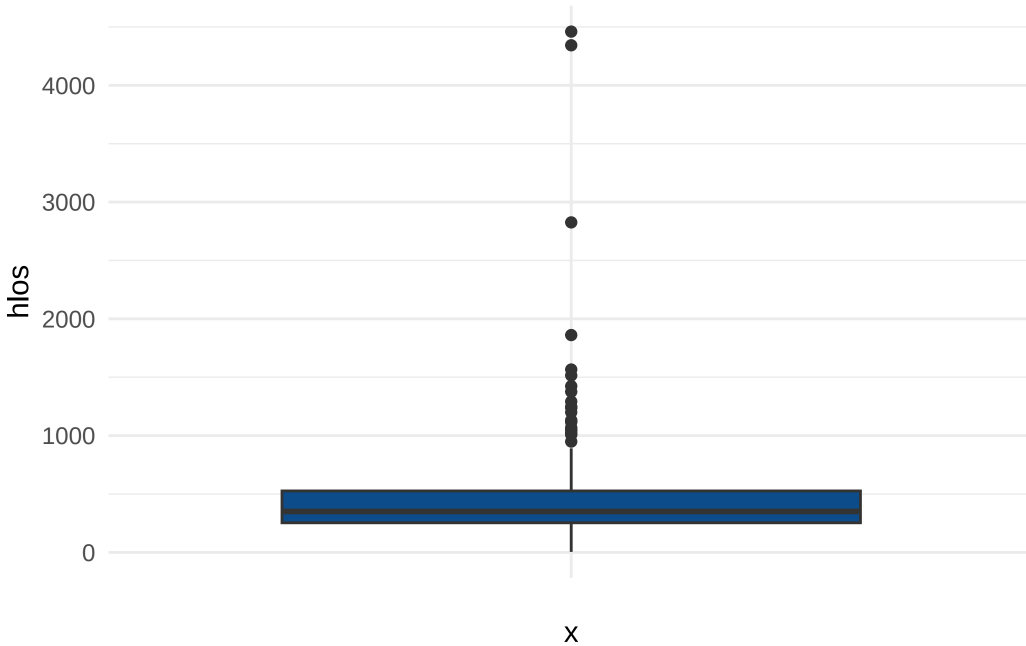
Values > 1000 hours are concerning as they are very far from the main distribution. Same thing is shown with the QQ plot for values above the +1 quantile.

These will be evaluated as outliers to determine if they should be retained.

1.23 Boxplot

```
df.ed %>% ggplot +
  aes(x = "", y = hlos) +
  geom_boxplot(fill = "#0c4c8a") +
  theme_minimal()
```

```
Don't know how to automatically pick scale for object of type <difftime>.
Defaulting to continuous.
```



The boxplot shows that values ~>1000 hours are actually outside the 75th percentile. We can use the `bloxplot.stats` command to tell us the exact values.

```
boxplot.stats(df.ed$hlos)$out
```

```
Time differences in mins
 [1] 1376.000 1566.000 1240.000 1125.000 1291.817 1068.000 4460.000 2826.000
 [9] 1200.000 1515.000 1423.917 1051.000 1009.983 1025.000 1009.000 1861.000
[17] 1247.000 1115.000 1239.000  949.000 1132.000 4343.000 1044.000
```

I don't want to look through all these values so I'll ask R to just return the lowest value in the range for me.

```
min(boxplot.stats(df.ed$hlos)$out)
```

```
Time difference of 949 mins
```

The lowest value outside the 75th percentile is 949 h, so this is the potential threshold for identifying outliers.

1.24 Evaluating Outliers over a Threshold

First we generate a categorical variable to signify the potential outliers.

```
df <- df.ed %>%
      mutate(hlosOutlier = case_when(
        hlos >=949 ~ 1,
        hlos < 949 ~ 0,
       .default = NA
    )
)
df$hlosOutlier <- as.factor(df$hlosOutlier)
```

Add in some other variables to evaluate the potential outliers. If they are outliers on `hlos`, are they also outliers in other variables?

```
df <- left_join(df, df.triage, by = "stay_id")

df.maxdx <- df.dx %>%
  select(seq_num, stay_id) %>%
  group_by(stay_id) %>%
  summarise(seq_num = max(seq_num))

df <- left_join(df, df.maxdx, by = "stay_id")

# cleanup
rm(df.maxdx)
df$subject_id.y <- NULL

df <- df %>%
  rename(subject_id = subject_id.x,
         dxtotal = seq_num)
```

Generate a basic summary comparing variables by potential outlier status

```
df %>%
  select(hlosOutlier, temperature:dbp, dxtotal) %>%
  group_by(hlosOutlier) %>%
  summarise(across(where(is.numeric), list(
    min  = min,
    max  = max,
    mean = mean,
```

```
    sd   = sd,
    p50  = median
  ), na.rm = TRUE
  )) %>%
  as.data.frame()
```

```
  hlosOutlier temperature_min temperature_max temperature_mean temperature_sd
1           0            36.5           100.3         97.73526       4.762178
2           1            96.6           100.2         98.16522       0.868441
  temperature_p50 heartrate_min heartrate_max heartrate_mean heartrate_sd
1            98.1            43           157       91.23429     19.17185
2            98.0            55           128       90.69565     17.00918
  heartrate_p50 resprate_min resprate_max resprate_mean resprate_sd
1            89           14           32      18.16477    3.046517
2            95           15           28      17.91304    2.745460
  resprate_p50 o2sat_min o2sat_max o2sat_mean o2sat_sd o2sat_p50 sbp_min
1           18        78       100   97.61714 2.797113        98      70
2           18        94       100   98.26087 1.737746        98     106
  sbp_max sbp_mean   sbp_sd sbp_p50 dbp_min dbp_max dbp_mean   dbp_sd dbp_p50
1     218 135.5341 28.30303   133.5      42     879 76.95455 62.75338      72
2     181 147.7826 21.44540   149.0      49     110 77.69565 16.93687      77
  dxtotal_min dxtotal_max dxtotal_mean dxtotal_sd dxtotal_p50
1           1           9      2.40404   1.570148           2
2           1           6      3.00000   1.348400           3
```

Note that there is an additional option of `na.rm = TRUE` which removes missing values from the `summarise` process.

The final statement `as.data.frame()` was only added to display the information when rendering the PDF notes.

Potential outliers differ mainly on the minimums and standard deviations which actually may point to typos for other variables.

I've rerun the same data using `gtsummary` so that I can get a sense of missing values and generate a p-value to assess any statistical differences.

```
library(gtsummary)

df %>%
  select(hlosOutlier, temperature:dbp, dxtotal) %>%
  tbl_summary(by = "hlosOutlier") %>%
  add_p()
```

Characteristic	**0** N = 199[1]	**1** N = 23[1]	**p-value**[2]
temperature	98.10 (97.60, 98.50)	98.00 (97.70, 98.80)	>0.9
Unknown	26	0	
heartrate	89 (77, 105)	95 (76, 103)	>0.9
Unknown	24	0	
resprate	18 (16, 18)	18 (16, 20)	0.7
Unknown	23	0	
o2sat	98 (96, 100)	98 (97, 100)	0.4
Unknown	24	0	
sbp	134 (114, 154)	149 (132, 163)	0.020
Unknown	23	0	
dbp	72 (61, 82)	77 (64, 87)	0.2
Unknown	23	0	
dxtotal			0.028
1	77 (39%)	5 (22%)	
2	43 (22%)	1 (4.3%)	
3	37 (19%)	9 (39%)	
4	19 (9.6%)	6 (26%)	
5	12 (6.1%)	1 (4.3%)	
6	6 (3.0%)	1 (4.3%)	
7	3 (1.5%)	0 (0%)	
9	1 (0.5%)	0 (0%)	
Unknown	1	0	

[1]Median (Q1, Q3); n (%)
[2]Wilcoxon rank sum test; Fisher's exact test

I elect to retain these patients because there are no profound differences among the variables between potential outlier status.