National Insurance Case

Mitchell J. Lovett

11/1/2020

R Markdown

This case discusses an insurance company, national insurance, that ran a customer satisfaction survey with the goal of understanding their current performance and how their service recovery relates to satisfaction. This is mandate-oriented research with a descriptive analytics focus.

Below the case walks through specific research questions and analytics to address the analytics need.

First we install packages we need, if necessary, and load them into the library. Then we set filenames and directories. Then we load the data two ways using the library `foreign` and the function `read.spss`. The first call to `read.spss` obtains the numeric data, the second obtains the labels data.

```
#install.packages("ggplot2", dependencies=T)
#install.packages("tm",dependencies=T)
#install.packages("wordcloud", dependencies=T)
#install.packages("gtools", dependencies=T)
library(ggplot2)
library(gtools)
library(wordcloud)
```

```
## Loading required package: RColorBrewer
```

```
library(tm)
```

```
## Loading required package: NLP

##
## Attaching package: 'NLP'

## The following object is masked from 'package:ggplot2':
##
##     annotate
```

```
library(reshape2)
dir = "~/Dropbox/Analytics Design/Cases/National Ins"
setwd(dir)

library(foreign)
filnm = "national"; #this is the name of the file
natLabData <- read.spss(paste(filnm,".sav",sep=""),to.data.frame=TRUE,use.value.labels=TRUE,trim_values=
```

```
## Warning in read.spss(paste(filnm, ".sav", sep = ""), to.data.frame = TRUE, :
## Undeclared level(s) 2, 3, 4, 5, 6, 7, 8, 9 added in variable: oq
```

```
natData <- read.spss(paste(filnm,".sav",sep=""),to.data.frame=TRUE,use.value.labels=FALSE); #turning va
```

Including Plots

Familiarize yourself with the data

Before getting to the focal analytics, we first need to familiarize ourselves with the data and response values. A good way to get started with this is to use the **names** and **summary** commands.

```r
names(natData)
```

```
##  [1] "p1"       "p2"       "p3"       "p4"       "p5"       "p6"
##  [7] "p7"       "p8"       "p9"       "p10"      "p11"      "p12"
## [13] "p13"      "p14"      "p15"      "p16"      "p17"      "p18"
## [19] "p19"      "p20"      "p21"      "p22"      "tanimp"   "relimp"
## [25] "resimp"   "asrimp"   "empimp"   "oq"       "rec"      "use"
## [31] "prob"     "resolve"  "sex"      "mstat"    "age"      "inc"
## [37] "ed"       "reliavrg" "empavrg"  "tangavrg" "respavrg" "assuravg"
```

```r
summary(natLabData)
```

```
##                            p1                              p2
##  Agree                      :82   Strongly Agree            :110
##  Strongly Agree             :79   Agree                     : 67
##  Agree Slightly             :42   Agree Slightly            : 44
##  Neither Agree Nor Disagree:38   Neither Agree Nor Disagree: 22
##  Strongly Disagree          :14   Disagree Slightly         : 16
##  (Other)                    :22   (Other)                   : 19
##  NA's                       : 8   NA's                      :  7
##                            p3                              p4
##  Strongly Agree             :79   Strongly Agree            :91
##  Agree                      :68   Agree                     :75
##  Agree Slightly             :48   Agree Slightly            :45
##  Neither Agree Nor Disagree:35   Neither Agree Nor Disagree:33
##  Strongly Disagree          :18   Strongly Disagree         :16
##  (Other)                    :27   (Other)                   :15
##  NA's                       :10   NA's                      :10
##                            p5                              p6
##  Strongly Agree             :85   Strongly Agree            :132
##  Agree                      :72   Agree                     : 68
##  Agree Slightly             :44   Neither Agree Nor Disagree: 28
##  Neither Agree Nor Disagree:35   Agree Slightly            : 28
##  Disagree Slightly          :13   Strongly Disagree         : 10
##  (Other)                    :17   (Other)                   : 13
##  NA's                       :19   NA's                      :  6
##                            p7                              p8
##  Strongly Agree             :91   Strongly Agree            :138
##  Agree Slightly             :66   Agree                     : 62
##  Agree                      :52   Agree Slightly            : 30
##  Neither Agree Nor Disagree:39   Neither Agree Nor Disagree: 24
##  Strongly Disagree          :10   Strongly Disagree         : 10
##  (Other)                    :16   (Other)                   : 12
##  NA's                       :11   NA's                      :  9
##                            p9                             p10
##  Strongly Agree             :88   Strongly Agree            :92
##  Agree                      :52   Agree                     :68
##  Agree Slightly             :46   Agree Slightly            :45
##  Neither Agree Nor Disagree:35   Neither Agree Nor Disagree:25
```

```
##  Disagree Slightly         :25   Disagree Slightly         :17
##  (Other)                   :28   (Other)                   :22
##  NA's                      :11   NA's                      :16
##                         p11                             p12
##  Agree Slightly            :65   Agree                     :62
##  Agree                     :57   Agree Slightly            :59
##  Strongly Agree            :57   Strongly Agree            :51
##  Neither Agree Nor Disagree:43   Neither Agree Nor Disagree:43
##  Disagree Slightly         : 8   Disagree Slightly         : 8
##  (Other)                   : 7   (Other)                   : 9
##  NA's                      :48   NA's                      :53
##                         p13                              p14
##  Strongly Agree            :128   Strongly Agree            :81
##  Agree                     : 81   Agree                     :68
##  Agree Slightly            : 33   Agree Slightly            :59
##  Neither Agree Nor Disagree: 15   Neither Agree Nor Disagree:37
##  Strongly Disagree         :  3   Strongly Disagree         : 5
##  (Other)                   :  3   (Other)                   : 7
##  NA's                      : 22   NA's                      :28
##                         p15                             p16
##  Strongly Agree            :92   Strongly Agree            :113
##  Agree                     :70   Agree                     : 62
##  Agree Slightly            :39   Agree Slightly            : 35
##  Neither Agree Nor Disagree:36   Neither Agree Nor Disagree: 28
##  Disagree Slightly         :11   Strongly Disagree         : 13
##  (Other)                   :16   (Other)                   : 16
##  NA's                      :21   NA's                      : 18
##                         p17                              p18
##  Strongly Agree            :129   Strongly Agree            :105
##  Agree                     : 60   Agree                     : 53
##  Agree Slightly            : 31   Agree Slightly            : 46
##  Neither Agree Nor Disagree: 19   Neither Agree Nor Disagree: 28
##  Strongly Disagree         : 11   Strongly Disagree         : 14
##  (Other)                   : 18   (Other)                   : 19
##  NA's                      : 17   NA's                      : 20
##                         p19                              p20
##  Strongly Agree            :109   Strongly Agree            :107
##  Agree                     : 63   Agree                     : 63
##  Agree Slightly            : 32   Agree Slightly            : 41
##  Neither Agree Nor Disagree: 30   Neither Agree Nor Disagree: 22
##  Strongly Disagree         : 14   Strongly Disagree         : 15
##  (Other)                   : 19   (Other)                   : 16
##  NA's                      : 18   NA's                      : 21
##                         p21                              p22
##  Strongly Agree            :141   Strongly Agree            :107
##  Agree                     : 66   Agree                     : 83
##  Agree Slightly            : 25   Agree Slightly            : 27
##  Neither Agree Nor Disagree: 18   Neither Agree Nor Disagree: 24
##  Strongly Disagree         :  6   Disagree                  :  9
##  (Other)                   :  9   (Other)                   : 13
##  NA's                      : 20   NA's                      : 22
##      tanimp          relimp          resimp          asrimp     
##  Min.   : 0.00   Min.   : 0.00   Min.   :  1.0   Min.   : 0.00  
##  1st Qu.: 5.00   1st Qu.:20.00   1st Qu.: 20.0   1st Qu.:15.00  
```

```
## Median :10.00   Median :25.00   Median : 20.0   Median :20.00
## Mean   :11.09   Mean   :28.81   Mean   : 22.7   Mean   :19.85
## 3rd Qu.:15.00   3rd Qu.:30.00   3rd Qu.: 25.0   3rd Qu.:25.00
## Max.   :60.00   Max.   :75.00   Max.   :100.0   Max.   :96.00
## NA's   :32      NA's   :32      NA's   :32      NA's   :32
##     empimp                   oq       rec                    use
## Min.   : 0.00   Extremely Good:73   Yes :235   Less than 1 year: 36
## 1st Qu.:10.00   9             :62   No  : 32   1 to 2 years    : 16
## Median :20.00   8             :47   NA's: 18   2 to 5 years    : 26
## Mean   :17.55   5             :24              5 or more years :193
## 3rd Qu.:20.00   7             :24              NA's            : 14
## Max.   :60.00   (Other)       :38
## NA's   :32      NA's          :17
##    prob     resolve        sex          mstat        age
## Yes : 80   Yes : 47   Male  :144   Single  : 20   < 25 : 20
## No  :180   No  : 30   Female:132   Married :219   25-44:219
## NA's: 25   NA's:208   NA's  :  9   Widowed : 12   45-64: 12
##                                    Divorced: 24   65+  : 24
##                                    NA's    : 10   NA's : 10
##
##
##             inc                         ed        reliavrg        empavrg
## < $10K        : 12   High School or Less:102   Min.   :1.000   Min.   :1.000
## $10K-$19.9 K  : 20   Some College       : 86   1st Qu.:4.600   1st Qu.:4.800
## $20K - $29.9 K: 59   College Graduate   : 50   Median :5.800   Median :5.800
## $30K-$49.9K   :106   Graduate School    : 35   Mean   :5.404   Mean   :5.524
## $50K-$64.9    : 38   NA's               : 12   3rd Qu.:6.400   3rd Qu.:6.800
## >$65K         : 32                             Max.   :7.000   Max.   :7.000
## NA's          : 18                             NA's   :6       NA's   :4
##    tangavrg        respavrg        assuravg
## Min.   :1.000   Min.   :1.000   Min.   :1.00
## 1st Qu.:5.000   1st Qu.:4.750   1st Qu.:5.25
## Median :5.750   Median :6.000   Median :6.25
## Mean   :5.635   Mean   :5.604   Mean   :5.75
## 3rd Qu.:6.500   3rd Qu.:7.000   3rd Qu.:7.00
## Max.   :7.000   Max.   :7.000   Max.   :7.00
## NA's   :9       NA's   :17      NA's   :16
```

We see variables `p1` to `p22`, which are perceptual measures of the performance of the company on service quality dimensions. They are scaled on These measures use the Likert scale with seven points. These individual scales are combined to make the multi-item scales `reliavrg`, `empavrg`, `tangavrg`, `respavrg`, and `assuravg`. These multi-item scales appear in the last columns of the dataset and are averages of subsets of the measurues `p1` to `p22`. In addition, there are measures `tanimp`, `reliimp`, `resimp`, `asrimp`, `empimp`, which represent the importance out of 100 that individuals allocated to these same six dimensions of service quality. The six dimensions are tangibles (appearance of assets owned by the company), reliability, responsiveness, assurance, and empathy.

`oq` is the overall service quality, `rec` is whether the respondent indicates they would recommend the company, `use` is how long the customer has used the insurance company, `prob` is whether the customer indicated experiencing a problem, `resolve` is whether the customer indicated having their problem resolved by the company.

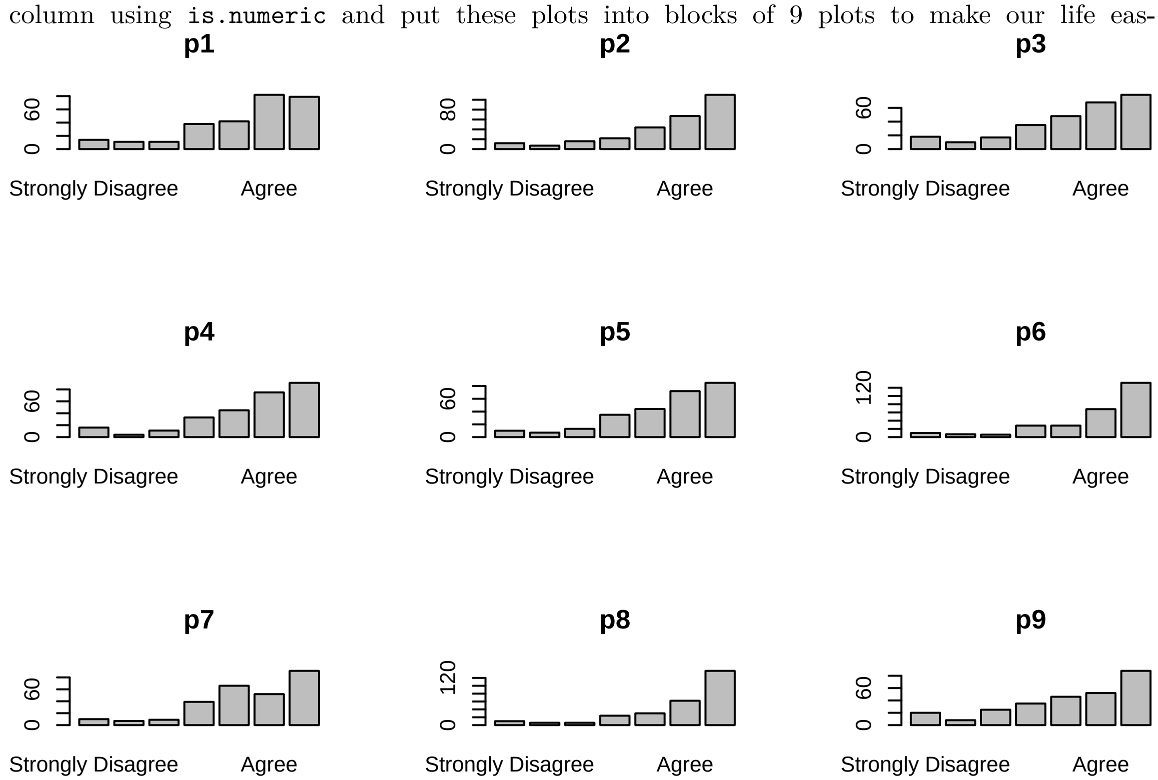
In addition, the company tracks a number of demographic variables including gender (`sex`), maritale status (`mstat`), age (`age`), income (`inc`), and educaton `ed`. These variables are all categorical and follow standard approaches.

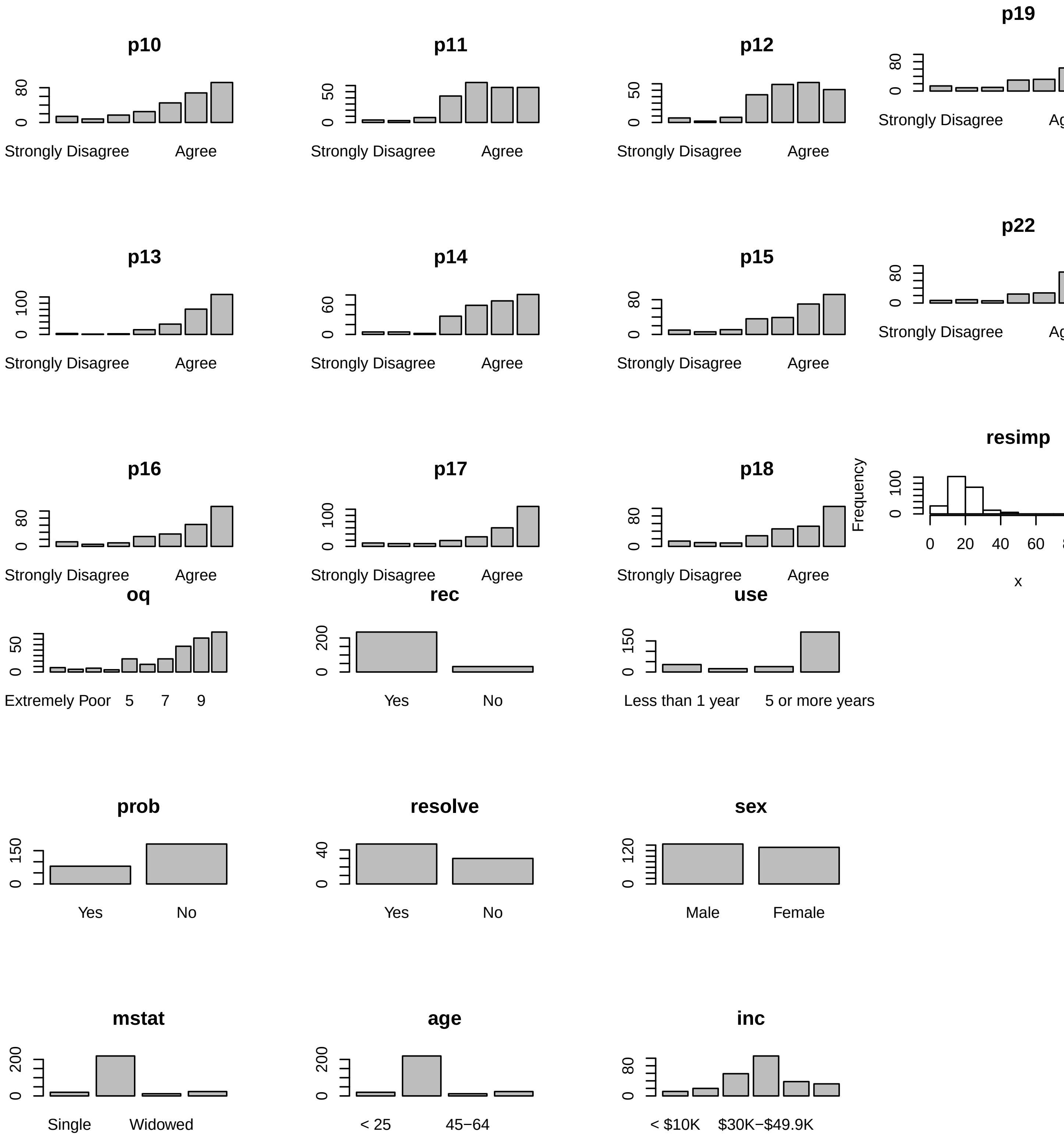
Face validity checks

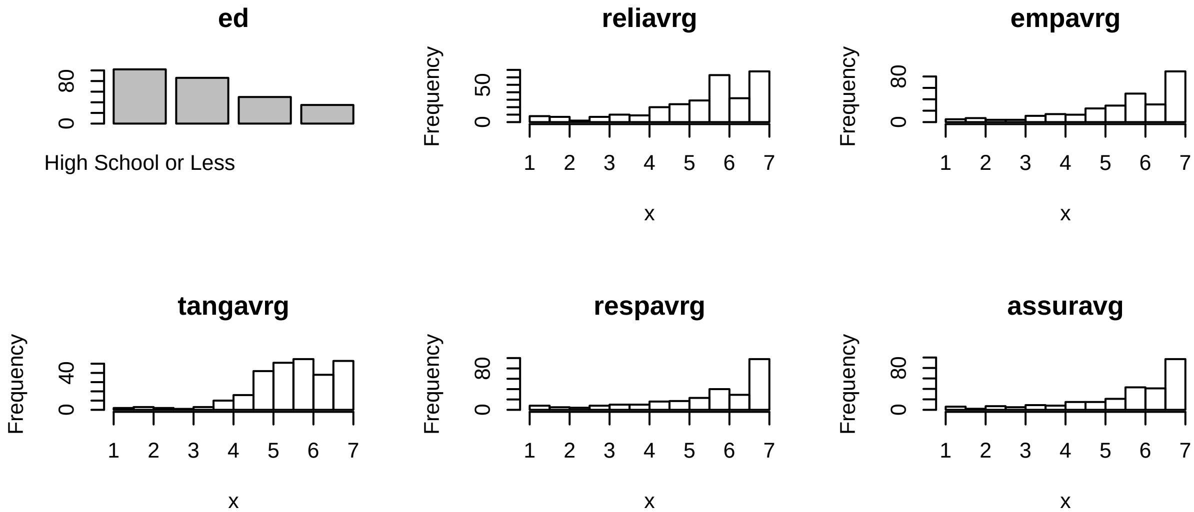
We check the summary output for possible face validity concerns. First one concern is about conformity/validity to specs related to the response values being outside of the expected range or set of response values. We confirm this doesn't appear to occur. Second, we check whether there is excessive missing data. We do this by looking at the number of NAs.

The only case that looks particularly worisome is `resolve` with 208 missing cases. We should always ask ourselves how our data could look like it is being presented to us, how likely it is. To answer this, we have to pose hypotheses about why the data look the way they do. This is thinking about the *Data Generating Process*. In a survey, this means thinking through both the survey questions and flow (e.g., is there branching or conditional logic) as well as thinking through how likely a particular series of responses are. In this case, why do you think `resolve` has so many NAs?

Third, I normally plot the marginal distributions. While summary is helpful, it is sometimes hard to "see" strange things quickly with the text output. We can quickly plot the distribution of the data using either `plot` for factors and `hist` for numeric data. We check the type of data stored in each column using `is.numeric` and put these plots into blocks of 9 plots to make our life easier.




p10
p11
p12
p19
p13
p14
p15
p22
p16
p17
p18
resimp
Frequency
x
oq
rec
use
prob
resolve
sex
mstat
age
inc
Strongly Disagree
Agree
Extremely Poor
Yes
No
Less than 1 year
5 or more years
Male
Female
Single
Widowed
< 25
45−64
< $10K
$30K−$49.9K



We can also run correlations on the data and look for surprisingly high or low correlations. In particular, correlations of 100% are always a bad sign! Here, I present a quick check for 100% correlations.

```
corCheck = cor(natData,use="pairwise.complete.obs") #this ignores NAs by pairs of variables
```

```
## Warning in cor(natData, use = "pairwise.complete.obs"): the standard deviation
## is zero
```

Notice the warning. This is letting us know that we have some cases that have no variation in their pairwise.complete.obs. We can investigate further by checking how many NAs there are and running `summary` on `corCheck` to see where they are. After this investigation we see there are only two cases and they arise between `prob` and `resolve`. This is closely related to our earlier finding that resolve only has observations when there is a problem! This implies there will be no variation in problem when resolve is not NA.

```
sum(is.na(corCheck))/2
```

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

```
summary(corCheck)
```

```
##        p1                 p2                 p3                 p4
##  Min.   :-0.54452   Min.   :-0.60308   Min.   :-0.5425   Min.   :-0.57355
##  1st Qu.: 0.03287   1st Qu.: 0.09199   1st Qu.: 0.0807   1st Qu.: 0.05106
##  Median : 0.57286   Median : 0.63246   Median : 0.5694   Median : 0.63763
##  Mean   : 0.41458   Mean   : 0.44725   Mean   : 0.4076   Mean   : 0.43698
##  3rd Qu.: 0.71317   3rd Qu.: 0.76747   3rd Qu.: 0.6796   3rd Qu.: 0.75324
##  Max.   : 1.00000   Max.   : 1.00000   Max.   : 1.0000   Max.   : 1.00000
##
##        p5                p6                p7                p8
##  Min.   :-0.4627   Min.   :-0.5073   Min.   :-0.4217   Min.   :-0.53438
##  1st Qu.: 0.1130   1st Qu.: 0.1006   1st Qu.: 0.1100   1st Qu.: 0.05866
##  Median : 0.6000   Median : 0.5989   Median : 0.4696   Median : 0.58611
##  Mean   : 0.4114   Mean   : 0.4410   Mean   : 0.3420   Mean   : 0.43473
##  3rd Qu.: 0.6649   3rd Qu.: 0.7299   3rd Qu.: 0.5339   3rd Qu.: 0.71895
##  Max.   : 1.0000   Max.   : 1.0000   Max.   : 1.0000   Max.   : 1.00000
##
##        p9                 p10                p11                p12
##  Min.   :-0.55666   Min.   :-0.5176   Min.   :-0.2068   Min.   :-0.1835
##  1st Qu.: 0.07232   1st Qu.: 0.1253   1st Qu.: 0.0874   1st Qu.: 0.1247
##  Median : 0.62845   Median : 0.6448   Median : 0.3697   Median : 0.3582
```

```
##  Mean   : 0.44021   Mean   : 0.4563   Mean   : 0.3002   Mean   : 0.3014
##  3rd Qu.: 0.74430   3rd Qu.: 0.7576   3rd Qu.: 0.4080   3rd Qu.: 0.4144
##  Max.   : 1.00000   Max.   : 1.0000   Max.   : 1.0000   Max.   : 1.0000
##
##       p13                p14               p15                p16
##  Min.   :-0.27788   Min.   :-0.2812   Min.   :-0.59775   Min.   :-0.5855
##  1st Qu.: 0.04144   1st Qu.: 0.1093   1st Qu.: 0.09579   1st Qu.: 0.0653
##  Median : 0.46109   Median : 0.4795   Median : 0.69137   Median : 0.6538
##  Mean   : 0.33937   Mean   : 0.3438   Mean   : 0.45747   Mean   : 0.4680
##  3rd Qu.: 0.51907   3rd Qu.: 0.5258   3rd Qu.: 0.76548   3rd Qu.: 0.7881
##  Max.   : 1.00000   Max.   : 1.0000   Max.   : 1.00000   Max.   : 1.0000
##
##       p17                p18                p19                p20
##  Min.   :-0.62185   Min.   :-0.58685   Min.   :-0.66398   Min.   :-0.66326
##  1st Qu.: 0.06452   1st Qu.: 0.04568   1st Qu.: 0.06137   1st Qu.: 0.05675
##  Median : 0.63531   Median : 0.64310   Median : 0.68972   Median : 0.66821
##  Mean   : 0.46421   Mean   : 0.46268   Mean   : 0.46077   Mean   : 0.44922
##  3rd Qu.: 0.78823   3rd Qu.: 0.77064   3rd Qu.: 0.78008   3rd Qu.: 0.78142
##  Max.   : 1.00000   Max.   : 1.00000   Max.   : 1.00000   Max.   : 1.00000
##
##       p21                p22              tanimp              relimp
##  Min.   :-0.44876   Min.   :-0.6146   Min.   :-0.390195   Min.   :-0.47606
##  1st Qu.: 0.01586   1st Qu.: 0.0699   1st Qu.:-0.004341   1st Qu.:-0.11956
##  Median : 0.60495   Median : 0.6214   Median : 0.068680   Median :-0.08079
##  Mean   : 0.41993   Mean   : 0.4221   Mean   : 0.058511   Mean   :-0.06526
##  3rd Qu.: 0.70180   3rd Qu.: 0.7275   3rd Qu.: 0.116199   3rd Qu.:-0.03025
##  Max.   : 1.00000   Max.   : 1.0000   Max.   : 1.000000   Max.   : 1.00000
##
##      resimp             asrimp              empimp                oq
##  Min.   :-0.28165   Min.   :-0.35303   Min.   :-0.476056   Min.   :-0.71920
##  1st Qu.:-0.10978   1st Qu.:-0.06321   1st Qu.: 0.008191   1st Qu.: 0.07991
##  Median :-0.00709   Median :-0.02947   Median : 0.083601   Median : 0.64266
##  Mean   :-0.01433   Mean   :-0.01219   Mean   : 0.070745   Mean   : 0.45611
##  3rd Qu.: 0.02771   3rd Qu.: 0.02719   3rd Qu.: 0.119369   3rd Qu.: 0.79825
##  Max.   : 1.00000   Max.   : 1.00000   Max.   : 1.000000   Max.   : 1.00000
##
##       rec                use                prob              resolve
##  Min.   :-0.71920   Min.   :-0.15613   Min.   :-0.44266   Min.   :-0.58355
##  1st Qu.:-0.58650   1st Qu.:-0.04420   1st Qu.: 0.08572   1st Qu.:-0.45243
##  Median :-0.45573   Median :-0.01616   Median : 0.41962   Median :-0.36220
##  Mean   :-0.30530   Mean   : 0.01464   Mean   : 0.29889   Mean   :-0.23498
##  3rd Qu.:-0.06703   3rd Qu.: 0.02761   3rd Qu.: 0.49787   3rd Qu.:-0.03121
##  Max.   : 1.00000   Max.   : 1.00000   Max.   : 1.00000   Max.   : 1.00000
##                                         NA's   :1          NA's   :1
##       sex               mstat               age                inc
##  Min.   :-0.11776   Min.   :-0.12542   Min.   :-0.12542   Min.   :-0.2976
##  1st Qu.:-0.02860   1st Qu.: 0.01237   1st Qu.: 0.01237   1st Qu.:-0.2185
##  Median : 0.01789   Median : 0.05890   Median : 0.05890   Median :-0.1789
##  Mean   : 0.03290   Mean   : 0.08900   Mean   : 0.08900   Mean   :-0.1087
##  3rd Qu.: 0.04983   3rd Qu.: 0.08793   3rd Qu.: 0.08793   3rd Qu.:-0.0930
##  Max.   : 1.00000   Max.   : 1.00000   Max.   : 1.00000   Max.   : 1.0000
##
##        ed              reliavrg            empavrg            tangavrg
##  Min.   :-0.27565   Min.   :-0.61756   Min.   :-0.55437   Min.   :-0.26244
```

```
##  1st Qu.:-0.20897   1st Qu.: 0.07171   1st Qu.: 0.09563   1st Qu.: 0.08235
##  Median :-0.15006   Median : 0.69419   Median : 0.69059   Median : 0.48811
##  Mean   :-0.09791   Mean   : 0.47699   Mean   : 0.48737   Mean   : 0.37660
##  3rd Qu.:-0.07828   3rd Qu.: 0.81473   3rd Qu.: 0.81557   3rd Qu.: 0.54623
##  Max.   : 1.00000   Max.   : 1.00000   Max.   : 1.00000   Max.   : 1.00000
##
##     respavrg           assuravg
##  Min.   :-0.62787   Min.   :-0.66092
##  1st Qu.: 0.07733   1st Qu.: 0.05664
##  Median : 0.69138   Median : 0.71965
##  Mean   : 0.49392   Mean   : 0.48446
##  3rd Qu.: 0.83175   3rd Qu.: 0.84364
##  Max.   : 1.00000   Max.   : 1.00000
##
```

The check on perfect correlation shows us that indeed we have a column that is perfectly correlated–marital status and age. We can go back to our plots from earlier and see, indeed the distributions are exactly the same. This suggests that we have a serious problem. Not both columns can be right! This will likely require us to go back and collaborate with the data provider.

```r
diag(corCheck)=0 #diagonal of correlations return value is always 1 by definition, so eliminating these
sum(corCheck==1,na.rm=TRUE)/2 #correlations are given in full table, so duplicated.
```

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

```r
summary(corCheck==1)
```

```
##      p1              p2              p3              p4
##  Mode :logical   Mode :logical   Mode :logical   Mode :logical
##  FALSE:42        FALSE:42        FALSE:42        FALSE:42
##
##      p5              p6              p7              p8
##  Mode :logical   Mode :logical   Mode :logical   Mode :logical
##  FALSE:42        FALSE:42        FALSE:42        FALSE:42
##
##      p9             p10             p11             p12
##  Mode :logical   Mode :logical   Mode :logical   Mode :logical
##  FALSE:42        FALSE:42        FALSE:42        FALSE:42
##
##     p13             p14             p15             p16
##  Mode :logical   Mode :logical   Mode :logical   Mode :logical
##  FALSE:42        FALSE:42        FALSE:42        FALSE:42
##
##     p17             p18             p19             p20
##  Mode :logical   Mode :logical   Mode :logical   Mode :logical
##  FALSE:42        FALSE:42        FALSE:42        FALSE:42
##
##     p21             p22            tanimp          relimp
##  Mode :logical   Mode :logical   Mode :logical   Mode :logical
##  FALSE:42        FALSE:42        FALSE:42        FALSE:42
##
##    resimp          asrimp          empimp            oq
##  Mode :logical   Mode :logical   Mode :logical   Mode :logical
##  FALSE:42        FALSE:42        FALSE:42        FALSE:42
##
##     rec             use             prob          resolve
```

```
##  Mode :logical   Mode :logical   Mode :logical   Mode :logical  
##  FALSE:42        FALSE:42        FALSE:41        FALSE:41       
##                                  NA's :1         NA's :1        
##     sex            mstat            age             inc         
##  Mode :logical   Mode :logical   Mode :logical   Mode :logical  
##  FALSE:42        FALSE:41        FALSE:41        FALSE:42       
##                  TRUE :1         TRUE :1                        
##      ed           reliavrg        empavrg         tangavrg      
##  Mode :logical   Mode :logical   Mode :logical   Mode :logical  
##  FALSE:42        FALSE:42        FALSE:42        FALSE:42       
##                                                                 
##   respavrg        assuravg      
##  Mode :logical   Mode :logical  
##  FALSE:42        FALSE:42       
## 
```

In summary, we looked at several quick face validity checks:

- Look at `summary` for range of values
- Look at `summary` for number of NA values
- Look at `plot` of factors and `hist` of numeric variables. Does it look strange?
- Look for NAs in the correlation after using `use=pairwise.complete.obs`
- Look for 100% correlations when using `use=pairwise.complete.obs`

How representative is our sample?

This section discusses how to evaluate whether your sample is representative of the population along a specific dimension. When do you need such a test? This test is only relevant if you have the goal of projecting to the population with your sample. If you used a quota sample and met your quotas, then these quotas already provide your assurance that the sample matches the population.[1] So, this test is relevant when either your quotas went awry, you used probability sampling, or you used some other method but still think the sample might be representative.

The goal of this test is to evaluate whether a specific observable variable has the same distribution in the sample as the population. The null hypothesis for this test is that the sample and population proportions are the same. We construct a test statistic based on the expected proportions in the sample if the sample had the true population proportions (P. We subtract this expected proportion from the actual sample proportion (M), multiply by the sample size (n), and square these differences. Summing over these squared differences produces our test statistic, ξ^2_{stat}. Mathematically, where k is the index for categories along our variable we test from 1 to K, the test statistic is constructed as,

$$\xi^2_{stat} = \sum_{k=1}^{K} \left(n\left(P_k - \bar{M}_k\right)\right)^2 \tag{1}$$

We compare this test statistic against quantiles of the *chi-squared* distribution. Thus, we compare against $\xi^2_{1-\alpha}$, where α is the desired p-value. We reject if $\xi^2_{stat} > \xi^2_{1-\alpha}$. In R, there are built-in functions to run this test, so you don't need to do this calculation yourself. However, it is important to recognize that rejecting the test means that we think the sameple *differs* from the population.

The idea is that if we don't reject the null hypothesis, then sample bias is much less likely to arise due to sample selection. The test is not a guarantee against sample bias for three reasons. First, what we want is to

[1]Of course, if you used disproportionate sampling, then you need to reweight the sample for aggregate conclusions.

not reject the null hypothesis. However, simply by being not very confident, we might not reject the null. As a result, we want reasonably large samples before we give much confidence to such a test. For this, a typical thought is to use more than 40 samples for at least most categories, if not all. Second, the variable we select to run the test on is only one of many possible variables that might exhibit the sample selection. If the variable is related to our study variables this increases our faith the test is helping to guard against selection bias, but testing multiple variables is also helpful. Third, even if the variable matches the proportions, there might still be selection within the categories. We always want to ask ourselves whether given our sampling method and potential sample issues, can the respondents within the category represent that category?

In order to conduct this test, we need population proportions. We get these from the same sources as we obtain our quotas. Typical candidates include census products, syndicated large sample surveys, and customer databases. In this case, we take them from the customer database since we are trying to same something about customer satisfaction, so our population is customers.

Below we conduct such tests for gender and length of use. In each block, we set the population values (in the same order as the labels in the sample data). We then print out a table of the values. Finally, we conduct the test. The function `chisq.test` is used to perform a Chi-squared test. The first argument is the observed counts from your sample of all the levels for the categorical variable, and the second, p=. . . , represents the "expected" proportion, i.e., the population proportion.

```
##1) Test whether sample matches with population for gender and length of use
##check gender:
popSEX = c(.54,.46); #true population values, which we can get from customer system data
sampSEX = table(natData$sex)

cbind(popSEX,sampSEX = prop.table(sampSEX)); #creating table as matrix to print it
```

```
##   popSEX   sampSEX
## 1   0.54 0.5217391
## 2   0.46 0.4782609
```

```
chisq.test(sampSEX,p=popSEX)
```

```
##
##  Chi-squared test for given probabilities
##
## data:  sampSEX
## X-squared = 0.37051, df = 1, p-value = 0.5427
```

The R output above shows that the Chi-square statistic is 0.3705, that degree of freedom is 1, and that the p-value is 0.5427. Hence we fail to reject the null hypothesis at any level of significance and the distribution of gender in the sample is not statistically different from that in the population. In other words, the sample matches the population on gender. Next we test length of use of the service.

```
##check length of use
popLU = c(.08,.09,.18,.65); #true population values, which we can get from customer system data
sampLU = table(natData$use)

cbind(popLU,sampLU=prop.table(sampLU)); #creating table as matrix to print it
```

```
##   popLU     sampLU
## 1  0.08 0.13284133
## 2  0.09 0.05904059
## 3  0.18 0.09594096
## 4  0.65 0.71217712
```

```
chisq.test(sampLU,p=popLU)
```

```
##
##  Chi-squared test for given probabilities
##
## data:  sampLU
## X-squared = 24.595, df = 3, p-value = 1.877e-05
```

Chi-square statistic is 24.59, degree of freedom is 3, and p-value is 1.877e-5. Hence we reject the null at any typical level of significance. The sample doesn't match the population on use length. First group and the last group are over-represented in the sample while the other two groups are under-represented.

Since the sample doesn't match the population, we might want to reweight the data. Usually one reweights the sample only if the variable that doesn't match is likely to be correlated (or is correlated) with other variables of interest. To create the weights use the formula $w_k = P_k/M_k$, where w_k is the weight for category k, P_k is the population proportion, and M_k is the sample proportion. Once the weights are created, you need to build a variable that holds these for all the respondents. `wSamp` holds these weights for all cases. We construct this variable through thoughtful indexing into w using the use category levels. Now whenever an analysis is done, one could use the `wSamp` to reweight. Keep in mind that when doing so, this will drop any observations that have missing values for length of use.

```
#weights, if reweighting
w = popLU/prop.table(table(natData$use)); ##formula is w=P/M
#setting weight for each response in sample
wSamp = w[as.numeric(natData$use)]; #this command indexes into w for each response picking the right va
```

For now, we will not use these weights in our analysis.

Analysis for Analytics Goals

We are now going to start addressing the analytics goals that speak to the business needs. We will examine testing and visually presenting means of different variables. We will then discuss testing whether subgroups within a variable are different.

Testing Whether Two Variables Are Different

We start with an analytics goal of describing performance and importance variables. On the performance sides, our analysis design will specifically answer how National Insurance is doing on service quality and what they do best. On the importance side we will ask what customers say is most and least important and how this importance relates to our relative performance.

We will do some set up before running the analysis. When dealing with multiple variables, it is often helpful to group the variables together. THe variables below do this. `perfvars` groups all of the five performance variables together so that if we want to call all five variables we just use `perfvars` rather than calling those five. This is to save effort and make our code more extensible. Similarly we create a vector `impvars`. The versions with "f" at the end are the full names used to make the plots look prettier.

```
##When dealing with sets of variables
##Create a "vector" using c() of the labels for the set
## of variables and another for the full labels

#create indexing for performance and importance variables
perfVars = c("reliavrg","empavrg","tangavrg","respavrg","assuravg"); perfVars=names(natData)[38:42]
impVars = c("relimp","empimp","tanimp","resimp","asrimp")

#create full names for performance and importance variables
genVarsf = c("Reliability","Empathy", "Tangibles","Responsiveness","Assurance")
```

```
perfVarsf = paste(genVarsf,"Perf.")
impVarsf = paste(genVarsf,"Imp.")
```

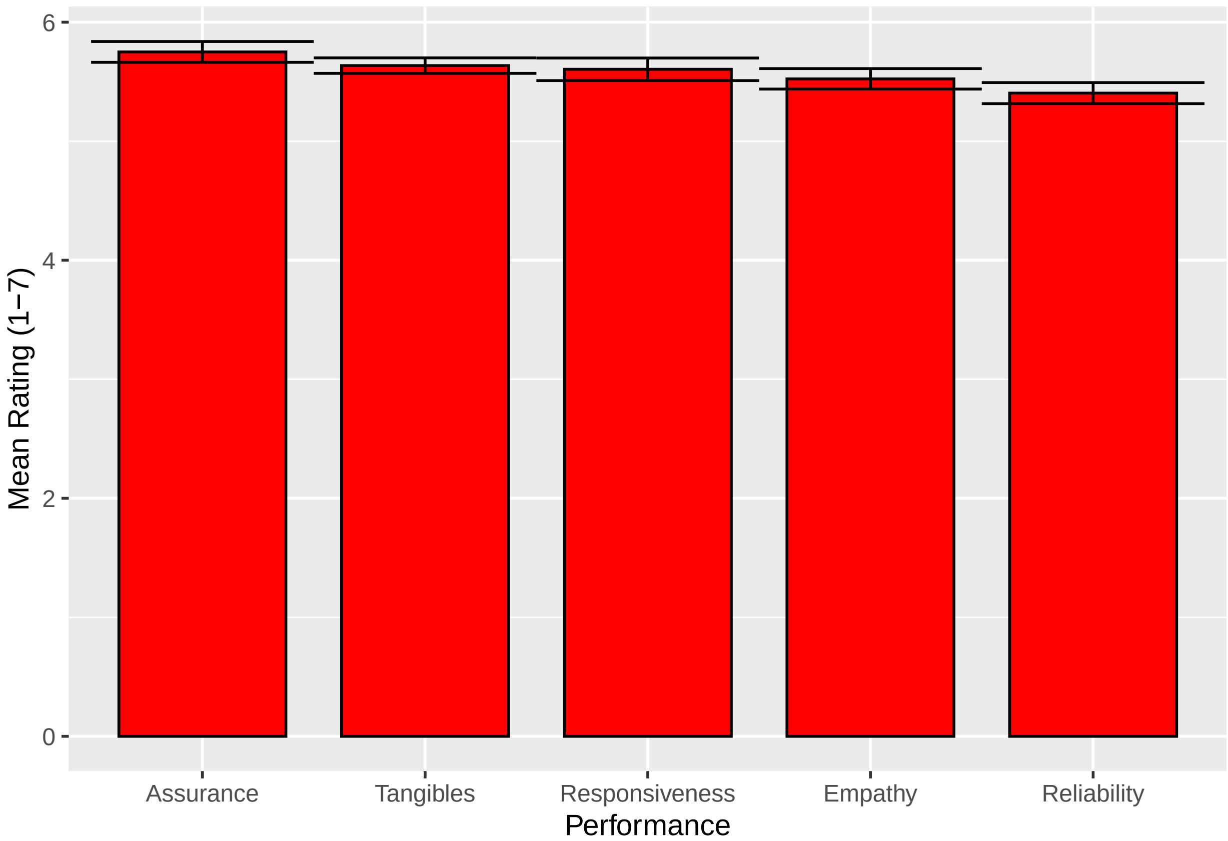
We now use the variables above to efficiently create the information for our plots–means and standard errors

```
#calculate means and standard errors for performance
perfMeans = colMeans(natData[,perfVars],na.rm=TRUE)
perfSE = apply(natData[,perfVars],2,sd,na.rm=TRUE)/
  sqrt(colSums(!is.na(natData[,perfVars])))
#calculate Means and standard errors for importance
impMeans = colMeans(natData[,impVars],na.rm=TRUE)
impSE = apply(natData[,impVars],2,sd,na.rm=TRUE)/
  sqrt(colSums(!is.na(natData[,impVars])))
perfImpDF = data.frame(genVarsf,perfVars,perfVarsf,perfMeans,perfSE,impVars,impVarsf,impMeans,impSE)
```

Now let's make a couple of error bar plots to describe and visualize the data. We will use ggplot2 for this. We will use the ggplot() function with the main geom being geom_bar and we add to it a geom_errorbar. To set up the bars, we include in the aes() object y and x. To set up the error bars, we need to calculate the error bar limits we will use and include them in (same) aes() object as ymax and ymin. We add and subtract one standard error. Using a single standard deviation for error bars can be useful because then you can loosely interpret if the error bars cross-over each other as meaning the variables are not significantly different. Because we have multiple geoms, we create a common positioning object to pass to the creation of all layers

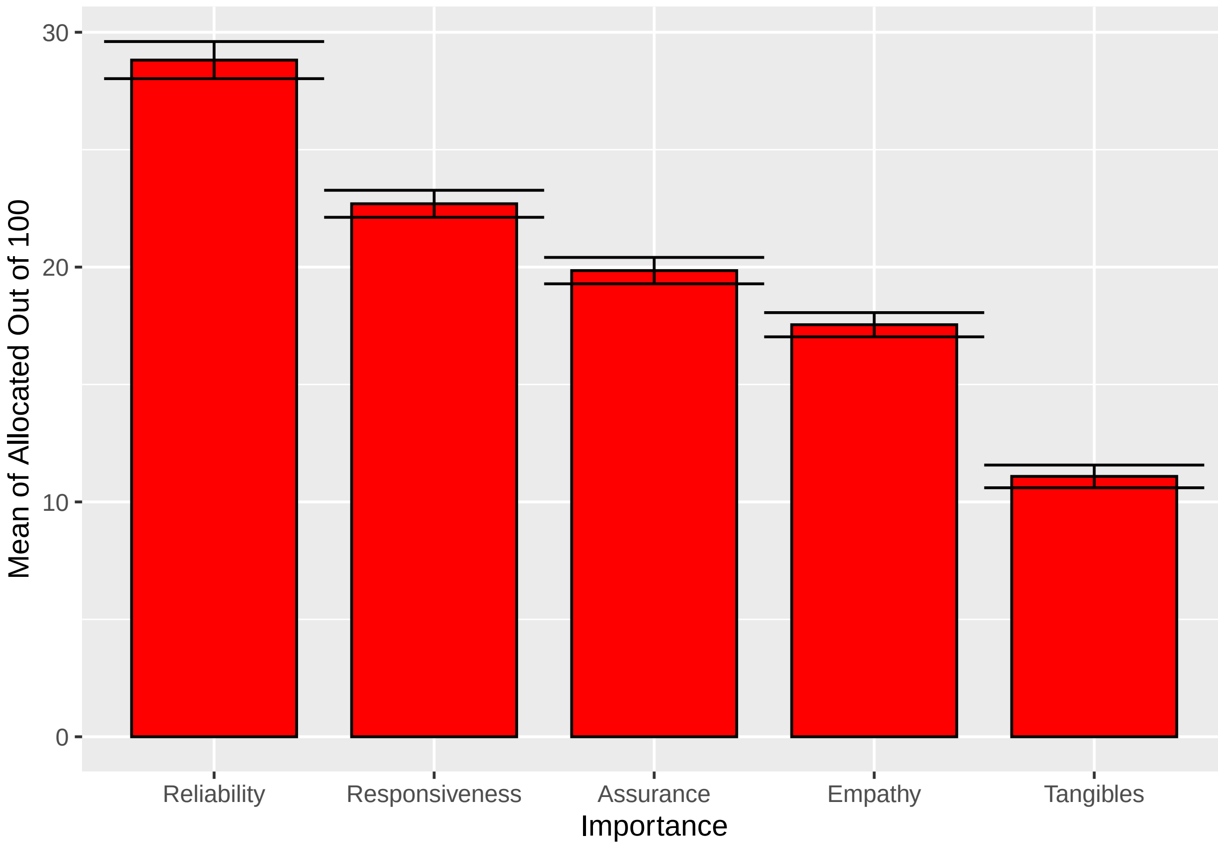
```
#create error bar plots for the performance variables and the importance variables
dodge = position_dodge(width=.75); ##to form constant dimensions positioning for all geom's

#first barplot with standard errors for the performance variables in descending order of means
gp = ggplot(perfImpDF,aes(x=reorder(genVarsf,-perfMeans),y=perfMeans,ymax=perfMeans+perfSE,ymin=perfMean
gp + geom_bar(position=dodge,stat="identity",col=1,fill=2,width=.75) +
  geom_errorbar(position=dodge,width=1) + labs(x="Performance",y="Mean Rating (1-7)")
```



The above graph shows that the differences are not very large between cases, but large enough to show significant differences other than neighboring cases. For instance, reliability performance is significantly lower than assurance performance.

```
##And creating similar graph for importance.
gi = ggplot(perfImpDF,aes(x=reorder(genVarsf,-impMeans),y=impMeans,ymax=impMeans+impSE,ymin=impMeans-im
gi + geom_bar(position=dodge,stat="identity",col=1,fill=2,width=.75) +
  geom_errorbar(position=dodge,width=1) + labs(x="Importance",y="Mean of Allocated Out of 100")
```



The same graph for importances shows that reliability is most important, following by responsiveness, assurance, empathy, and tangibles. Interestingly, this order does not match closely at all the performance ordering. This suggests what we perform well at is not what the consumer thinks is important. An important managerially relevant insight!

If we want to be more precise we can also evaluate the uncertainty using a t-test. A similar concept would be to use the t-tests built into a linear regressions.

```
##can run t-tests on these, but with the s.e., we already have the gist
##to construct all paired combinations
combos = combinations(5,2)
combos = data.frame(combos,v1=perfVarsf[combos[,1]],v2=perfVarsf[combos[,2]],means=numeric(nrow(combos))
for(i in 1:nrow(combos)){
  t_tmp = t.test(natData[,perfVars[combos[i,1]]],natData[,perfVars[combos[i,2]]],paired=TRUE)
  combos[i,c("means","p.values")]=c(t_tmp$statistic,t_tmp$p.value)
}
```

Taking the first example, according to the R output, t statistic is -1.97 and p-value is 0.0493. Hence we reject the null at 5% level of significance. The mean of reliavrg and of empavrg are statistically different. The average evaluations of those two aspects of service quality are different. We could already see this in the figure since the error lines didn't cover the means (the dots).

Multiple tests and p-value adjustments

If we wanted to get technical, these p-values are not accurate because we are testing many hypotheses. When you test many hypotheses, this can lead to the p-values being too low. Essentially, you are violating the principles of statistical testing.

There are multiple methods used to correct this issue. One of the simplest, early methods was Bonferroni's simple method. That approach simply mutiplies the p-value by the total number of hypotheses tested. In our example above, this is 12, the number of rows in `combos`. Other methods provide stronger tests, which are illustrated below.

```
##Construct adjusted p-values with Bonferroni's simple method
##bf.p.values =  p-value*M, where M is number of hypothesis tests
combos = data.frame(combos,bf.p.values = combos[,"p.values"]*nrow(combos))
##Adjust p-values with a more powerful method. For additional information check help on p.adjust
combos = data.frame(combos,adj.p.values=p.adjust(combos[,"p.values"],method="hommel"))
```

We can see a number of the tests that appeared significant at the 5% level previously no longer have such low p-values. The effect of such adjustments allows us to internalize the lower confidence we have in our conclusions. Because these tests are in general conservative, we should interpret the results as being a more accurate reflection of our uncertainty. Also, note there are some conditions on these tests related to the independence of the samples. In this case, we might actually be worried about the possibility since we are using paired t.tests. In practice, you should be aware of these methods and the overconfidence that multiple testing produces without such adjustments. In many circumstances I find that practitioners do not use such correction approaches directly, but being aware of the relative shift (the Bonferroni adjustment rule) and when you need to worry about this (when you do many comparisons).

Testing Whether Variables Differ Between Subgroups

In this example, we test whether our performance variables differ by gender. To do this, we use `t.test` or `lm`. We will use `lm` for this example. Like our past examples, we set up the tests as a `for` loop. This loop is indexing over the set of performance variables, calling one regression for each variable. The regression evaluates whether "Female" customers give significantly higher performance ratings than "Male" customers.

```
sexTests = matrix(nrow=length(perfVars),ncol=4);
rownames(sexTests)=perfVarsf
for(i in 1:length(perfVars)){ #looping over performance variables
  slm = summary(lm(natData[,perfVars[i]]~sex,data=natLabData)) #runs regression
  sexTests[i,]=slm[[4]][2,] #saves the row in the summary table corresponding to the sexFemale variable
}
colnames(sexTests) = colnames(slm[[4]])
```

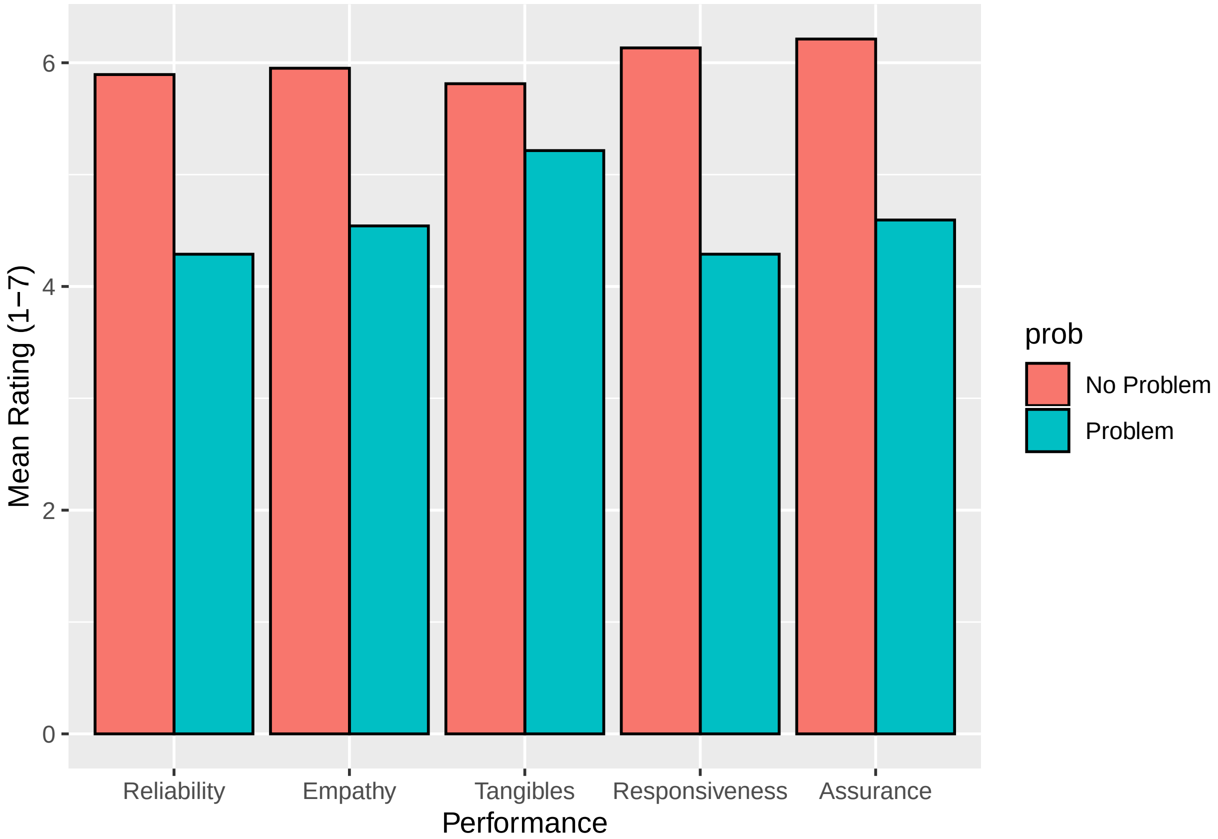
We find that there are no significant differences across the two `sex` subgroups on the performance variables. You should be able to pattern match this code to produce a similar output for the importance variables. We don't need to even consider the adjusted p-values, since these make the p-values larger. Further, we probably don't need to visualize this result and instead just use a concise statement, if any at all.

For the next questions, we ask

```
probTests = matrix(nrow=length(perfVars),ncol=4);
rownames(probTests)=perfVarsf
for(i in 1:length(perfVars)){ #looping over performance variables
  slm = summary(lm(natData[,perfVars[i]]~prob,data=natLabData)) #runs regression
  probTests[i,]=slm[[4]][2,] #saves the row in the summary table corresponding to the probYes variable
}
colnames(probTests) = colnames(slm[[4]])
```

```
perfMeansByProb = apply(natData[,perfVars],2,tapply,natData$prob,mean,na.rm=TRUE)
colnames(perfMeansByProb) = genVarsf
perfMeansByProb = data.frame(prob = c("Problem","No Problem"),
                             perfMeansByProb)
perfMeansByProbLong = melt(perfMeansByProb, id=c("prob"))
gp = ggplot(perfMeansByProbLong,aes(x=variable,y=value,fill=prob))
gp + geom_bar(position=position_dodge(),stat="identity",col=1) +
```

```
    labs(x="Performance",y="Mean Rating (1-7)")
```



Open-ended responses and simple text analysis

Wordclouds are a useful way to present free form text data from surveys. Typically, we might "clean" the text first, taking care of misspellings and potentially obtaining the stems. As this class is not focused on text analysis per se, I provide a simple use of wordclouds with only simple cleaning.

```
txt =  "Many years ago the great British explorer George Mallory, who was to die on Mount Everest, was a
it. He said, \"Because it is there.\"

Well, space is there, and we're going to climb it, and the
moon and the planets are there, and new hopes for knowledge
and peace are there. And, therefore, as we set sail we ask
God's blessing on the most hazardous and dangerous and greatest
adventure on which man has ever embarked."

docs = Corpus(VectorSource(txt))
#Some functions you can use to eliminate unnecessary stuff and make better clouds
docs = tm_map(docs,removeNumbers)
```

```
## Warning in tm_map.SimpleCorpus(docs, removeNumbers): transformation drops
## documents
```

```
docs = tm_map(docs,removePunctuation)
```

```
## Warning in tm_map.SimpleCorpus(docs, removePunctuation): transformation drops
## documents
```

```r
docs = tm_map(docs,stripWhitespace)
```

```
## Warning in tm_map.SimpleCorpus(docs, stripWhitespace): transformation drops
## documents
```

```r
docs = tm_map(docs, content_transformer(tolower))
```

```
## Warning in tm_map.SimpleCorpus(docs, content_transformer(tolower)):
## transformation drops documents
```

```r
docs = tm_map(docs, removeWords, stopwords("english"))
```

```
## Warning in tm_map.SimpleCorpus(docs, removeWords, stopwords("english")):
## transformation drops documents
```

```r
#Here you create the term document matrix and evalually the word frequencies
dtm = TermDocumentMatrix(docs)
matrix = as.matrix(dtm)
words = sort(rowSums(matrix),decreasing=TRUE)
df = data.frame(word = names(words),freq=words)
#Create the wordcloud
set.seed(1234) # for reproducibility
wordcloud(words = df$word, freq = df$freq, min.freq = 1,           max.words=200, random.order=FALSE, r
```

```
## Warning in wordcloud(words = df$word, freq = df$freq, min.freq = 1, max.words =
## 200, : peace could not be fit on page. It will not be plotted.

## Warning in wordcloud(words = df$word, freq = df$freq, min.freq = 1, max.words =
## 200, : space could not be fit on page. It will not be plotted.

## Warning in wordcloud(words = df$word, freq = df$freq, min.freq = 1, max.words =
## 200, : therefore could not be fit on page. It will not be plotted.

## Warning in wordcloud(words = df$word, freq = df$freq, min.freq = 1, max.words =
## 200, : years could not be fit on page. It will not be plotted.
```

Tips on recoding

```
#if you want to recode some columns of to a different value
##create a new variable
osq = natData$oq
##set the values to the new values
osq[osq<5] = 0; #subsetting only those variables that have values less than 5 and then setting those va
osq[osq>=5] = 1 #subsetting to only those variables that have values greater than or equal to 5 and the
```

```
##If you want to drop some observations: recode the values to NA and use the na.rm=TRUE
##what is we don't want to include the most satisfied from the analysis? (variable osq2 is without the 
osq2 = osq; #notice I am creating a new variable - use the new variable for the analysis!
osq2[osq2==10]=NA; ##notice you need to use "==" to mean = when you mean equality. That's because = has
```

Imperative vs. functional programming

We briefly make a slight detour to discuss some programming concepts that can help more broadly, which we will apply to the code above. Entry level programming in R is usually done following the ideas in “imperative” programming. In imperative programming, you execute code to directly change the variables. In contrast in pure functional programming, you only return values using functions that create temporary storage that is returned upon finishing the function. I do NOT advocate pure functional programming, but moving in this direction can help make your code become more reusable and understandable.

We will go through an example here, using the code chunks `combos1` and `combos2` above. The code chunk, `combos1` constructs all possible combinations of comparisons, then sets up a `data.frame` to store the identity and test values, then uses a loop to construct all test values. This code is designed in an imperative programming style, since we are creating new objects and adjusting them in place as we walk through the `for` loop.

We will demonstrate how to construct a function that does this work. We begin with a simple version of the function that literally does the exact work above.

```
allTests1 = function(){ #declare the function
  combos = combinations(5,2)
  combos = data.frame(combos,v1=perfVarsf[combos[,1]],v2=perfVarsf[combos[,2]],
                      means=numeric(nrow(combos)),p.values=numeric(nrow(combos)))
  for(i in 1:nrow(combos)){
    t_tmp = t.test(natData[,perfVars[combos[i,1]]],
                   natData[,perfVars[combos[i,2]]],paired=TRUE)
    combos[i,c("means","p.values")]=c(t_tmp$statistic,t_tmp$p.value)
  }
  ##bf.p.values =  p-value*M, where M is number of hypothesis tests
  combos = data.frame(combos,bf.p.values = combos[,"p.values"]*nrow(combos))
  ##Adjust p-values with a more powerful method.
  combos = data.frame(combos,adj.p.values=p.adjust(combos[,"p.values"],method="hommel"))
  combos #return the combos data.frame
}
combosNew = allTests1() #run function and save result
sum(combos==combosNew)/prod(dim(combos)) #check if imperative code and function produce same results -
```

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

The function `allTests1` returns a data.frame that contains the full combos data. Notice the code is exactly the same except the opening line and the last two lines. In the opening line, we declare the function, `allTests1`. In the closing two lines we write `combos` to indicate to return that value when the function finishes and we close the function with a close brace.

While this function follows some principles of using functions, it fails to help us generalize the function to other settings. The reason is because we haven't passed any arguments. We need to ask ourselves what arguments we need to pass to make this function general. To answer this, we need to understand how we might we want to use this function in the future. This is a software design question, but when we are making an analytics product, this software design question is embedded into our analytics design.

For our purpose now, let us assume that we want our code to be able to do these tests for any setting where we have a `data.frame` and a set of columns from the `data.frame` on which we want to do pairwise comparisons. Thus, we will revise our function to have three arguments: a `data.frame`, a vector of column names, and a vector of names for those columns.

```
##function accepts:
##  a vector of variable names, vars
##  a data frame, data
##  a vector of variable fullnames, varnames
## returns a data.frame containing columns
##  indexes X1, X2 for comparison indexes into vars
##  variable names v1, v2 for each comparison
##  mean differences, means
##  p.values, for unadjusted p.values
##  Bonferroni adjusted p.values, bf.p.values, assuming appropriate
##  Hommel adjusted p.values, adj.p.values, assuming appropriate
allTests2 = function(vars,data,varnames){ #declare the function
  combos = combinations(length(vars),2)
  combos = data.frame(combos,v1=varnames[combos[,1]],v2=varnames[combos[,2]],
                      means=numeric(nrow(combos)),p.values=numeric(nrow(combos)))
  for(i in 1:nrow(combos)){
    t_tmp = t.test(data[,vars[combos[i,1]]],
                   data[,vars[combos[i,2]]],paired=TRUE)
```

```
    combos[i,c("means","p.values")]=c(t_tmp$statistic,t_tmp$p.value)
  }
  ##bf.p.values =  p-value*M, where M is number of hypothesis tests
  combos = data.frame(combos,bf.p.values = combos[,"p.values"]*nrow(combos))
  ##Adjust p-values with a more powerful method.
  combos = data.frame(combos,adj.p.values=p.adjust(combos[,"p.values"],method="hommel"))
  combos #return the combos data.frame
}
combosNew2 = allTests2(perfVars,natData,perfVarsf) #run function and save result
sum(combos==combosNew2)/prod(dim(combos)) #check if imperative code and function produce same
```

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

```
#combosNewFail = allTests2(c(perfVars,"junk"),natData,c(perfVarsf,"junk"))
#sum(combos[1,]==combosNewFail)/prod(dim(combos[1,])) #check if imperative code and function produce sa
```

With this function, we can reuse it any time that we want to do such comparisons where the comparisons are between columns. The major changes to the code `allTests1` include

- The original combinations are constructed by passing the length of vars instead of a fixed number, 5. This allows us to pass different lengths of variables.
- We replaced perfVars with vars, perfVarsf with varnames, and natData with data
- We have added extensive comments at the top about the arguments and the return value.

The extensive comments at the top describe what is expected for the arguments and what is returned. Although the comments aren't necessary, especially when you first start coding this way, by forcing yourself to do this, it can improve the quality of your software design. For production ready functions, you would want to *test* the arguments passed to your function to ensure that they are what you expect. If they are not, you throw warnings or errors since the result you return might not be as expected either.

In this function, three key tests are important to consider. First, we need the length of the vars to match the length of varnames. Second, we need each element in vars to match an index in data. Third, we need vars to have at least a length of 3, otherwise there is no multiple comparisons problem and we can just run the single test. Below we add these checks to our code.

Notice we do all of the tests in a single if clause. If any of the problems arise, we use the `stop` function and pass the text that describes what happened. The `stop` function breaks out of the function where it is called and throws an error. One alternative to having a single `if` statement with all of the tests would be to split the `if` statement into multiple `if` statements and then pass more informative error messages for each problem.

```
##function accepts:
##  a vector of variable names, vars
##  a data frame, data
##  a vector of variable fullnames, varnames
## returns a data.frame containing columns
##  indexes X1, X2 for comparison indexes into vars
##  variable names v1, v2 for each comparison
##  mean differences, means
##  p.values, for unadjusted p.values
##  Bonferroni adjusted p.values, bf.p.values, assuming appropriate
##  Hommel adjusted p.values, adj.p.values, assuming appropriate
allTests3 = function(vars,data,varnames){ #declare the function
  errstmt="";
  if(length(vars)!=length(varnames)) errstmt = paste(errstmt,"vars length not equal to varnames length")
  if(length(vars)!=sum(vars%in%names(data))) errstmt = paste(errstmt,"Not all vars in data")
  if(length(vars)<2) errstmt = paste(errstmt,"Function for vars >2 length")
```

```
  if(length(vars)!=length(varnames) | length(vars)!=sum(vars%in%names(data)) | length(vars)<2){
    stop(paste(" Arguments do not meet expectations:",errstmt)); #stop();
  }
  combos = combinations(length(vars),2)
  combos = data.frame(combos,v1=varnames[combos[,1]],v2=varnames[combos[,2]],
                      means=numeric(nrow(combos)),p.values=numeric(nrow(combos)))
  for(i in 1:nrow(combos)){
    t_tmp = t.test(data[,vars[combos[i,1]]],
                   data[,vars[combos[i,2]]],paired=TRUE)
    combos[i,c("means","p.values")]=c(t_tmp$statistic,t_tmp$p.value)
  }
  ##bf.p.values =  p-value*M, where M is number of hypothesis tests
  combos = data.frame(combos,bf.p.values = combos[,"p.values"]/nrow(combos))
  ##Adjust p-values with a more powerful method.
  combos = data.frame(combos,adj.p.values=p.adjust(combos[,"p.values"],method="hommel"))
  combos #return the combos data.frame
}
combosNew3 = allTests3(perfVars,natData,perfVarsf) #run function and save result
sum(combos==combosNew2)/prod(dim(combos)) #check if imperative code and function produce same
```

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

```
#combosNotFail = allTests3(perfVars[1:2],natData,perfVarsf[1:2]) #run function with values that fail
#combosFail = allTests3(perfVars[1:2],natData,perfVarsf) #run function with values that fail
#combosNewFail = allTests3(c(perfVars,"junk"),natData,c(perfVarsf,"junk"))
```

Using functions like this isn't just about being able to reuse your code. By adding checking, you reduce the chances that when you reuse it you will make a mistake. Further, by using functions, you only have to adjust one part of the code as you extend it. In fact, using functions has many benefits:

1. Ease of reading and understanding code
2. Simple reusability by calling existing functions
3. Can build in error checking to avoid mistakes in reusing code
4. Fixing bugs is simpler, since only one copy of code to adjust
5. Extending code to do more is easier, since only one copy of code to adjust
6. Less namespace polution
7. Less persistent memory use

When do you know you should use a function instead of imperative coding approach? If you find yourself copying and pasting the same code within a single .R or .Rmd file, this is a big hint that you could probably use a function and save yourself trouble. As we move forward in the course, we will use more functions. This is particularly important for understanding how to do automated control goals and for more complex analytics projects.