Bios 6301: Assignment 3

Lan Shi

Due Tuesday, 28 September, 1:00 PM

50 points total.

Add your name as `author` to the file's metadata section.

Submit a single knitr file (named `homework3.rmd`) by email to michael.l.williams@vanderbilt.edu. Place your R code in between the appropriate chunks for each question. Check your output by using the `Knit HTML` button in RStudio.

$5^{n=day}$ points taken off for each day late.

Question 1

15 points

Write a simulation to calculate the power for the following study design. The study has two variables, treatment group and outcome. There are two treatment groups (0, 1) and they should be assigned randomly with equal probability. The outcome should be a random normal variable with a mean of 60 and standard deviation of 20. If a patient is in the treatment group, add 5 to the outcome. 5 is the true treatment effect. Create a linear model for the outcome by the treatment group, and extract the p-value (hint: see assigment1). Test if the p-value is less than or equal to the alpha level, which should be set to 0.05.

Repeat this procedure 1000 times. The power is calculated by finding the percentage of times the p-value is less than or equal to the alpha level. Use the `set.seed` command so that the professor can reproduce your results.

1. Find the power when the sample size is 100 patients. (10 points)

```r
set.seed(925)
n = 100
mean(replicate(1e3, {
    treat_grp = rbinom(n, 1, 0.5)  # treat_grp = 1, if in treatment group
    outcome = rnorm(n, mean=60, sd=20)
    outcome[treat_grp==1] = outcome[treat_grp==1] + 5
    dt = data.frame(treat_grp,outcome)
    fit = lm(outcome~treat_grp,dt)
    #pvalue
    coef(summary(fit))[2,4]}) < 0.05)
```

```
## [1] 0.24
```

1. Find the power when the sample size is 1000 patients. (5 points)

```r
set.seed(925)
n = 1000
mean(replicate(1e3, {
    treat_grp = rbinom(n, 1, 0.5)  # treat_grp = 1, if in treatment group
    outcome = rnorm(n, mean=60, sd=20)
```

```
    outcome[treat_grp==1] = outcome[treat_grp==1] + 5
    dt = data.frame(treat_grp,outcome)
    fit = lm(outcome~treat_grp,dt)
    #pvalue
    coef(summary(fit))[2,4]}) < 0.05)
```

```
## [1] 0.981
```

Question 2

14 points

Obtain a copy of the football-values lecture. Save the `2021/proj_wr21.csv` file in your working directory. Read in the data set and remove the first two columns.

```
dt = read.csv('~/Desktop/21 FA/6301_Stats_Computing/Bios6301-main/football-values-main/2021/proj_wr21.c
```

1. Show the correlation matrix of this data set. (4 points)

```
(rho.dt=cor(dt)) # correlation
```

```
##            rec_att   rec_yds   rec_tds  rush_att  rush_yds  rush_tds   fumbles
## rec_att  1.0000000 0.9899611 0.9650160 0.3690670 0.3834924 0.3463555 0.7981497
## rec_yds  0.9899611 1.0000000 0.9746951 0.3452096 0.3611319 0.3244833 0.8011127
## rec_tds  0.9650160 0.9746951 1.0000000 0.3418033 0.3554974 0.3335733 0.7622937
## rush_att 0.3690670 0.3452096 0.3418033 1.0000000 0.9882542 0.8944610 0.3212985
## rush_yds 0.3834924 0.3611319 0.3554974 0.9882542 1.0000000 0.9055524 0.3290909
## rush_tds 0.3463555 0.3244833 0.3335733 0.8944610 0.9055524 1.0000000 0.2843320
## fumbles  0.7981497 0.8011127 0.7622937 0.3212985 0.3290909 0.2843320 1.0000000
## fpts     0.9879394 0.9968696 0.9864975 0.3839939 0.3997444 0.3660350 0.7899300
##               fpts
## rec_att  0.9879394
## rec_yds  0.9968696
## rec_tds  0.9864975
## rush_att 0.3839939
## rush_yds 0.3997444
## rush_tds 0.3660350
## fumbles  0.7899300
## fpts     1.0000000
```

1. Generate a data set with 30 rows that has a similar correlation structure. Repeat the procedure 1,000 times and return the mean correlation matrix. (10 points)

```
# codes are cited from lecture 9
library(MASS)
set.seed(925)

# Assume the joint distribution is normal
means.dt = colMeans(dt)
vcov.dt = var(dt)
(rho.dt = cor(dt))
```

```
##            rec_att   rec_yds   rec_tds  rush_att  rush_yds  rush_tds   fumbles
## rec_att  1.0000000 0.9899611 0.9650160 0.3690670 0.3834924 0.3463555 0.7981497
## rec_yds  0.9899611 1.0000000 0.9746951 0.3452096 0.3611319 0.3244833 0.8011127
## rec_tds  0.9650160 0.9746951 1.0000000 0.3418033 0.3554974 0.3335733 0.7622937
## rush_att 0.3690670 0.3452096 0.3418033 1.0000000 0.9882542 0.8944610 0.3212985
```

```
## rush_yds 0.3834924 0.3611319 0.3554974 0.9882542 1.0000000 0.9055524 0.3290909
## rush_tds 0.3463555 0.3244833 0.3335733 0.8944610 0.9055524 1.0000000 0.2843320
## fumbles  0.7981497 0.8011127 0.7622937 0.3212985 0.3290909 0.2843320 1.0000000
## fpts     0.9879394 0.9968696 0.9864975 0.3839939 0.3997444 0.3660350 0.7899300
##               fpts
## rec_att  0.9879394
## rec_yds  0.9968696
## rec_tds  0.9864975
## rush_att 0.3839939
## rush_yds 0.3997444
## rush_tds 0.3660350
## fumbles  0.7899300
## fpts     1.0000000
```

```
# Generate a data set with similar correlation structure
dt.sim = mvrnorm(30, mu = means.dt, Sigma = vcov.dt)
cor(dt.sim)
```

```
##            rec_att   rec_yds   rec_tds  rush_att  rush_yds  rush_tds   fumbles
## rec_att  1.0000000 0.9941229 0.9876236 0.4945128 0.5292747 0.4365727 0.8536150
## rec_yds  0.9941229 1.0000000 0.9860678 0.5205835 0.5526098 0.4717608 0.8523440
## rec_tds  0.9876236 0.9860678 1.0000000 0.5080379 0.5408655 0.4594828 0.8471275
## rush_att 0.4945128 0.5205835 0.5080379 1.0000000 0.9915423 0.9362555 0.4477226
## rush_yds 0.5292747 0.5526098 0.5408655 0.9915423 1.0000000 0.9348292 0.4464547
## rush_tds 0.4365727 0.4717608 0.4594828 0.9362555 0.9348292 1.0000000 0.3914543
## fumbles  0.8536150 0.8523440 0.8471275 0.4477226 0.4464547 0.3914543 1.0000000
## fpts     0.9929032 0.9981668 0.9914803 0.5527297 0.5848254 0.5042848 0.8476553
##               fpts
## rec_att  0.9929032
## rec_yds  0.9981668
## rec_tds  0.9914803
## rush_att 0.5527297
## rush_yds 0.5848254
## rush_tds 0.5042848
## fumbles  0.8476553
## fpts     1.0000000
```

```
# repeat 1000 times.
rho.sim = 0
loops=1e3

for (i in 1:loops) {
      dt.sim = mvrnorm(30, mu = means.dt, Sigma = vcov.dt)
      rho.sim = rho.sim+cor(dt.sim)/loops
}
# mean correlation matrix
rho.sim
```

```
##            rec_att   rec_yds   rec_tds  rush_att  rush_yds  rush_tds   fumbles
## rec_att  1.0000000 0.9896553 0.9635204 0.3616235 0.3753858 0.3344001 0.7938404
## rec_yds  0.9896553 1.0000000 0.9736767 0.3386527 0.3541400 0.3136504 0.7954490
## rec_tds  0.9635204 0.9736767 1.0000000 0.3362675 0.3494555 0.3233254 0.7546982
## rush_att 0.3616235 0.3386527 0.3362675 1.0000000 0.9876210 0.8901612 0.3152512
## rush_yds 0.3753858 0.3541400 0.3494555 0.9876210 1.0000000 0.9018991 0.3219001
## rush_tds 0.3344001 0.3136504 0.3233254 0.8901612 0.9018991 1.0000000 0.2736436
## fumbles  0.7938404 0.7954490 0.7546982 0.3152512 0.3219001 0.2736436 1.0000000
```

```
## fpts     0.9874097 0.9967265 0.9859653 0.3774267 0.3927538 0.3550744 0.7835940
##               fpts
## rec_att  0.9874097
## rec_yds  0.9967265
## rec_tds  0.9859653
## rush_att 0.3774267
## rush_yds 0.3927538
## rush_tds 0.3550744
## fumbles  0.7835940
## fpts     1.0000000
```

Question 3

21 points

Here's some code:

```r
nDist <- function(n = 100) {
    df <- 10
    prob <- 1/3
    shape <- 1
    size <- 16
    list(
        beta = rbeta(n, shape1 = 5, shape2 = 45),
        binomial = rbinom(n, size, prob),
        chisquared = rchisq(n, df),
        exponential = rexp(n),
        f = rf(n, df1 = 11, df2 = 17),
        gamma = rgamma(n, shape),
        geometric = rgeom(n, prob),
        hypergeometric = rhyper(n, m = 50, n = 100, k = 8),
        lognormal = rlnorm(n),
        negbinomial = rnbinom(n, size, prob),
        normal = rnorm(n),
        poisson = rpois(n, lambda = 25),
        t = rt(n, df),
        uniform = runif(n),
        weibull = rweibull(n, shape)
    )
}
```

1. What does this do? (3 points)

   ```r
   round(sapply(nDist(500), mean), 2)
   ```

   ```
   ##           beta       binomial     chisquared    exponential              f
   ##           0.10           5.43          10.15           1.05           1.09
   ##          gamma      geometric hypergeometric      lognormal    negbinomial
   ##           1.01           1.83           2.62           1.45          31.76
   ##         normal        poisson              t        uniform        weibull
   ##           0.02          25.09           0.00           0.50           0.97
   ```

   ```
   It gives the mean of the 500 random generated samples from each distrition, and rounds off these va
   ```

2. What about this? (3 points)

   ```r
   sort(apply(replicate(20, round(sapply(nDist(10000), mean), 2)), 1, sd))
   ```

```
##           beta        uniform         normal              f    exponential
##    0.000000000    0.002236068    0.006882472    0.007863975    0.008335088
##        weibull          gamma hypergeometric              t      lognormal
##    0.008870412    0.009119095    0.011192102    0.012814466    0.016944181
##       binomial      geometric      chisquared        poisson     negbinomial
##    0.019084301    0.027028250    0.037640963    0.066798991    0.107698409
```

```
First, for each distribution, it randomly sampled 10000 samples and calculated their means (w/ two
```

In the output above, a small value would indicate that `N=10,000` would provide a sufficent sample size as to estimate the mean of the distribution. Let's say that a value *less than 0.02* is "close enough".

3. For each distribution, estimate the sample size required to simulate the distribution's mean. (15 points)

```r
# suff_size: used to store sufficient sample size for each distribution.
suff_size = numeric(15)
suff_sd = numeric(15)
names(suff_size) = names(sapply(nDist(1000), mean))
names(suff_sd) = names(sapply(nDist(1000), mean))
#dist: indicator of distribution
# 1 for beta, 2 for binomial, ..., 15 for weibull
set.seed(925)
for (dist in 1:15){
    n = 0
    std_dev = Inf
    if (dist %in% c(1,14)){ # use different increments for different distribution.
        gap = 5
    }else if(dist %in% c(5,15,11,4,8,6)){
        gap = 200
    }else if(dist %in% c(13,2,7)){
        gap = 1000
    }else{
        gap = 3000
    }
    while (std_dev>=.02){
        n = n+ gap
        sample_sd = apply(replicate(20, round(sapply(nDist(n), mean), 2)), 1, sd)
        std_dev = sample_sd[dist]
    }
    suff_size[dist] = n
    suff_sd[dist] = std_dev
}
suff_size
suff_sd
```

Don't worry about being exact. It should already be clear that N < 10,000 for many of the distributions. You don't have to show your work. Put your answer to the right of the vertical bars (|) below.

distribution	N
beta	5
binomial	7000
chisquared	33000
exponential	3400
f	1000
gamma	1400
geometric	13000

distribution	N
hypergeometric	3800
lognormal	9000
negbinomial	165000
normal	1800
poisson	57000
t	3000
uniform	160
weibull	2400