Bios 6301: Assignment 3

Max Rohde

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

```r
set.seed(27182)

simulate <- function(n){
    # Create the group assignment by using a random permutation
    groups <- sample(rep(c("Treatment", "Control"), n/2))

    # Create a data frame to organize our results
    df <- tibble(subject_id =1:n,
                 group = groups,
                 outcome = rnorm(n, mean=60, sd=20))

    # Add 5 to the outcome for all subjects in the treatment group
    df <- mutate(df,
                 outcome = ifelse(group=="Treatment", outcome+5, outcome))

    # Create the linear model
    model <- lm(outcome ~ group, data = df)

    # Get the p-value from the model
    p <- summary(model)$coefficients[[2,4]]

    # return TRUE if statistically significant, FALSE otherwise
    return(p<=0.05)
}

# Simulate 1000 times for studies of 100 subjects
power_n100 <- map_lgl(1:1000, ~simulate(n=100)) %>% mean()
```

```r
# Simulate 1000 times for studies of 1000 subjects
power_n1000 <- map_lgl(1:1000, ~simulate(n=1000)) %>% mean()
```

```
The estimated power with 100 subjects is 0.238
The estimated power with 1000 subjects is 0.974
```

Question 2

14 points

Obtain a copy of the football-values lecture. Save the `2020/proj_wr20.csv` file in your working directory. Read in the data set and remove the first two columns.

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

```r
# Read in the data
df <- read_csv('https://raw.githubusercontent.com/couthcommander/football-values/master/2020/proj_wr20.c

# Remove first two columna
select(df, -(1:2)) -> df

# Show correlation matrix
cor(df)
```

```
##            rec_att   rec_yds   rec_tds  rush_att  rush_yds  rush_tds   fumbles
## rec_att  1.0000000 0.9915980 0.9762985 0.4243375 0.4208867 0.2955319 0.8599863
## rec_yds  0.9915980 1.0000000 0.9872802 0.4083053 0.4043565 0.2888207 0.8576713
## rec_tds  0.9762985 0.9872802 1.0000000 0.3949094 0.3902740 0.2773522 0.8433106
## rush_att 0.4243375 0.4083053 0.3949094 1.0000000 0.9813709 0.8502802 0.4507697
## rush_yds 0.4208867 0.4043565 0.3902740 0.9813709 1.0000000 0.8631215 0.4441662
## rush_tds 0.2955319 0.2888207 0.2773522 0.8502802 0.8631215 1.0000000 0.2763207
## fumbles  0.8599863 0.8576713 0.8433106 0.4507697 0.4441662 0.2763207 1.0000000
## fpts     0.9898311 0.9981916 0.9919078 0.4439299 0.4408144 0.3261509 0.8565599
##               fpts
## rec_att  0.9898311
## rec_yds  0.9981916
## rec_tds  0.9919078
## rush_att 0.4439299
## rush_yds 0.4408144
## rush_tds 0.3261509
## fumbles  0.8565599
## 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)

```r
# Store the column means and covariance matrix
means <- colMeans(df)
covariance <- var(df)

# This generates one data set, with 30 rows, that has similar correlation structure
# to the original dataset. We do this simulating a multivariate normal random variable
# with the same mean vector and covariance matrix
MASS::mvrnorm(30, mu=means, Sigma=covariance)
```

```
##          rec_att     rec_yds    rec_tds   rush_att  rush_yds     rush_tds
##  [1,] 37.3852511 483.1540947 2.79949411 0.38775991  4.649269 0.068672804
```

```
##  [2,]   7.1805103  151.8290694  1.34050199  1.21875884   6.697488  0.035633623
##  [3,] -10.9329090 -215.5304309 -1.95013736 -2.90206291 -22.100596 -0.116676025
##  [4,]  44.0808065  560.4871552  3.44138589  2.16638867  18.129144  0.030584587
##  [5,] -41.6620466 -509.3530951 -3.16813259 -1.65906216 -15.049493 -0.017036452
##  [6,]  33.9518038  465.1485947  3.46724161  5.00835995  29.334496  0.268234554
##  [7,] -22.3212191 -289.8170074 -1.61920725  0.82536496   1.116605 -0.011520775
##  [8,]  14.0773190  254.6279087  1.91698117 -0.83743397  -7.918677 -0.008679039
##  [9,]  76.1828166 1008.4319956  6.41573097  2.83469984  22.302712  0.180778038
## [10,]  -3.8734388    0.7773838  0.69316315  2.10813066   8.704510  0.151293976
## [11,]  47.1455815  624.2357370  3.83959253  3.52399424  21.451713  0.015001484
## [12,]  55.2322172  766.3196871  4.56688113  0.55616179   4.213652  0.047976621
## [13,] -17.0083748 -197.5274730 -1.37381356  1.52222918  14.572487  0.207736951
## [14,]  24.0664613  372.8099256  2.51078716  0.63594011   8.234054  0.010519562
## [15,]   2.9780645    4.5458340  0.06792477  0.31721760   2.982391 -0.003581532
## [16,]  55.7624594  730.8758636  4.52038187  0.44792281   1.374372  0.041887476
## [17,]  -0.2654242   62.9972732  0.21564744 -2.39618154 -22.417075 -0.092271584
## [18,]  13.8922647   89.9633039  0.47886711  3.97598725  24.981426  0.116495461
## [19,]  26.7110558  422.0751013  2.66325940  1.70303724  11.265567  0.133594461
## [20,]  61.7661704  886.3340374  5.45732863 -0.48847547  -7.354597 -0.141422254
## [21,]  35.6399621  491.4669472  3.01658328  3.05197958  13.526096 -0.053687420
## [22,]  41.4112354  515.0334892  3.22471088  2.15833949  12.799373  0.061377369
## [23,]  14.7973357  195.8724085  1.30710526 -0.06547853   1.017905 -0.011853824
## [24,] -31.5716023 -361.6760655 -1.80811409 -2.10352114 -15.899716 -0.101673599
## [25,]  14.3837974  258.6776484  1.40138783 -3.04578172 -20.104777 -0.174815291
## [26,]  55.3891036  735.5899278  4.71920099  5.62216948  37.210784  0.128573993
## [27,]  16.8169043  204.3673431  1.17099905  2.91775236  23.108334  0.182047782
## [28,]  45.6635614  643.1927315  3.94199357 -3.03079217 -19.672453 -0.108952336
## [29,]  69.7585499  900.4064968  5.67739869  6.53154715  45.504903  0.287702830
## [30,]  30.2890135  337.0130405  2.20908319 -3.85569600 -30.801941 -0.210344215
##           fumbles        fpts
##  [1,]  0.21165261  65.3561832
##  [2,]  0.15116317  23.5907891
##  [3,] -0.02936617 -36.0252998
##  [4,]  0.41230902  77.8744809
##  [5,] -0.40339247 -70.6279996
##  [6,]  0.18152630  71.1903593
##  [7,] -0.31800133 -38.1044252
##  [8,]  0.21139789  35.6675025
##  [9,]  0.61495097 141.6651113
## [10,] -0.12972804   6.1732997
## [11,]  0.59824412  86.2418889
## [12,]  0.48382797 104.1462353
## [13,] -0.05597154 -25.2138341
## [14,]  0.03440669  53.1945553
## [15,]  0.06768509   0.9068102
## [16,]  0.30465676 100.1299288
## [17,] -0.11645884   5.1685344
## [18,]  0.46694675  14.5078958
## [19,]  0.27142100  59.7369602
## [20,]  0.53640734 118.8303987
## [21,]  0.31837778  67.5277492
## [22,]  0.05875391  72.4458245
## [23,] -0.06379426  27.3093059
## [24,] -0.39895065 -48.6674510
```

```
## [25,] -0.01321679  31.2201662
## [26,]  0.50335166 105.4845009
## [27,]  0.01246794  30.8852322
## [28,]  0.38044295  84.7188558
## [29,]  0.44284367 129.5256865
## [30,]  0.11488811  42.7695473
```

```
# Repeat the above simulation 1000 times, and store each matrix in a list
simulated_datasets <- map(1:1000, ~MASS::mvrnorm(30, mu=means, Sigma=covariance))

# Take the correlation of all the simulated matrices
simulated_correlation_matrices <- map(simulated_datasets, ~cor(.x))

# Take the average of all the simulated correlation matrices by summing them up
# and dividing by the total number of them
mean_correlation_matrix <- reduce(simulated_correlation_matrices, `+`) / length(simulated_correlation_ma

# View the mean correlation matrix from the 1000 simulations
mean_correlation_matrix
```

```
##            rec_att   rec_yds   rec_tds  rush_att  rush_yds  rush_tds   fumbles
## rec_att  1.0000000 0.9913964 0.9757739 0.4185241 0.4150325 0.2879686 0.8568449
## rec_yds  0.9913964 1.0000000 0.9869639 0.4033824 0.3994370 0.2817037 0.8550508
## rec_tds  0.9757739 0.9869639 1.0000000 0.3897496 0.3853983 0.2710301 0.8407170
## rush_att 0.4185241 0.4033824 0.3897496 1.0000000 0.9804176 0.8453839 0.4440133
## rush_yds 0.4150325 0.3994370 0.3853983 0.9804176 1.0000000 0.8585033 0.4379914
## rush_tds 0.2879686 0.2817037 0.2710301 0.8453839 0.8585033 1.0000000 0.2692011
## fumbles  0.8568449 0.8550508 0.8407170 0.4440133 0.4379914 0.2692011 1.0000000
## fpts     0.9895770 0.9981394 0.9916929 0.4385167 0.4354573 0.3186730 0.8540064
##               fpts
## rec_att  0.9895770
## rec_yds  0.9981394
## rec_tds  0.9916929
## rush_att 0.4385167
## rush_yds 0.4354573
## rush_tds 0.3186730
## fumbles  0.8540064
## fpts     1.0000000
```

Question 3

21 points

Here's some code:

```
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)

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

```
##           beta       binomial     chisquared    exponential              f
##           0.10           5.21           9.95           0.95           1.17
##          gamma      geometric hypergeometric      lognormal    negbinomial
##           0.96           2.09           2.56           1.66          31.72
##         normal        poisson              t        uniform        weibull
##           0.02          25.20           0.04           0.46           1.06
```

```
The above code takes samples of size 500 from the distribtions defined in nDist() and computes the mean
```

1. What about this? (3 points)

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

```
##           beta        uniform        weibull              f          gamma
##    0.000000000    0.003077935    0.008944272    0.009233805    0.009514532
##         normal    exponential              t hypergeometric      lognormal
##    0.010399899    0.010711528    0.011964861    0.014317821    0.019761739
##       binomial      geometric        poisson      chisquared    negbinomial
##    0.022542358    0.027028250    0.039483508    0.041877013    0.108171404
```

```
The above code takes samples of size 10000 from the distribtions defined in nDist() and computes the mea

In the output above, a small value would indicate that `N=10,000` would provide a sufficent sample size
```

The below code estimates the sufficient sample size to estimate the mean for each distribution.

For each distribution, the procedure described above is computed starting from a sample size of 2, and the standard deviation of the 20 means is then stored. Until the standard deviation of the 20 means is below 0.02, the while loop continues, increasing the sample size each time. The first sample size to produce a standard deviation less than 0.02 is a rough estimate of the sufficient sample size to estimate the mean of the distribution.

```
df <- 10
prob <- 1/3
shape <- 1
size <- 16

l <- list(
        beta = function(n) rbeta(n, shape1 = 5, shape2 = 45),
        binomial = function(n) rbinom(n, size, prob),
        chisquared = function(n) rchisq(n, df),
        exponential = function(n) rexp(n),
```

```r
        f = function(n) rf(n, df1 = 11, df2 = 17),
        gamma = function(n) rgamma(n, shape),
        geometric = function(n) rgeom(n, prob),
        hypergeometric = function(n) rhyper(n, m = 50, n = 100, k = 8),
        lognormal = function(n) rlnorm(n),
        negbinomial = function(n) rnbinom(n, size, prob),
        normal = function(n) rnorm(n),
        poisson = function(n) rpois(n, lambda = 25),
        t = function(n) rt(n, df),
        uniform = function(n) runif(n),
        weibull = function(n) rweibull(n, shape)
    )
```

```r
######################################################
# Simulation to estimate the sufficient sample size
# to estimate the mean of each distribution
#
# This code may take a few minutes to run!
######################################################
set.seed(31415)

for (func in l){ # loop over all the distributions
    n <- 2
    stdev <- 1
    while (stdev > 0.02) # Loop until desired precision
    {
        # Simulate the data and take the standard deviation
        stdev <- map_dbl(1:20, ~func(n) %>% mean()) %>% sd()

        # Adaptively adjust the step size
        if(n <500){n <- n+1}
        else if(n < 1000){n <- n+10}
        else if(n < 10000){n <- n+50}
        else{n <- n+100}
    }
    print(func)
    print(n)
    print(stdev)
}
```

```
## function(n) rbeta(n, shape1 = 5, shape2 = 45)
## <bytecode: 0x7fa3504f95b8>
## [1] 6
## [1] 0.01993172
## function(n) rbinom(n, size, prob)
## <bytecode: 0x7fa367dca0a8>
## [1] 4150
## [1] 0.0153046
## function(n) rchisq(n, df)
## <bytecode: 0x7fa355ad4d50>
## [1] 20200
## [1] 0.01658918
## function(n) rexp(n)
## <bytecode: 0x7fa367876978>
```

```
## [1] 2000
## [1] 0.01780965
## function(n) rf(n, df1 = 11, df2 = 17)
## <bytecode: 0x7fa347e6f9f0>
## [1] 403
## [1] 0.01931758
## function(n) rgamma(n, shape)
## <bytecode: 0x7fa361ded8b0>
## [1] 1700
## [1] 0.01936692
## function(n) rgeom(n, prob)
## <bytecode: 0x7fa367e13b30>
## [1] 8400
## [1] 0.01764024
## function(n) rhyper(n, m = 50, n = 100, k = 8)
## <bytecode: 0x7fa351e932b8>
## [1] 2000
## [1] 0.01976113
## function(n) rlnorm(n)
## <bytecode: 0x7fa3558b5328>
## [1] 8450
## [1] 0.01937498
## function(n) rnbinom(n, size, prob)
## <bytecode: 0x7fa355fc7358>
## [1] 73900
## [1] 0.01789058
## function(n) rnorm(n)
## <bytecode: 0x7fa3605614e8>
## [1] 1300
## [1] 0.01866401
## function(n) rpois(n, lambda = 25)
## <bytecode: 0x7fa35066a038>
## [1] 33600
## [1] 0.01880582
## function(n) rt(n, df)
## <bytecode: 0x7fa36045ffc8>
## [1] 2050
## [1] 0.01923908
## function(n) runif(n)
## <bytecode: 0x7fa353b71cb8>
## [1] 129
## [1] 0.01993969
## function(n) rweibull(n, shape)
## <bytecode: 0x7fa354d17b10>
## [1] 1600
## [1] 0.01971699
```

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

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	5,000

distribution	N
chisquared	20,000
exponential	2,000
f	500
gamma	2,000
geometric	8,000
hypergeometric	2,000
lognormal	10,000
negbinomial	80,000
normal	2,000
poisson	35,000
t	2,000
uniform	100
weibull	1,500