Problem Set #02 Solutions Statistics 506

Problem Set #02

Problem 1 Solutions - Dice Game

a.

```
#' Dice game version 1 - using a loop
#' @param n Number of plays to make
#' @return Total won/lost
play1 <- function(n) {
  if (n < 1) {
    # If 0 (or less rolls) no winning or losing
    return(0)
  }

  die <- sample(1:6, n, replace = TRUE)

  total <- 0

  for (i in 1:n) {
    total <- total - 2 # cost to play
    if (die[i] %% 2 == 0) {
      total <- total + die[i]
    }
    # if 1,3,5, no change to total besides loss of 2 above
  }

  return(total)
}

#' Dice game version 2 - with vectorized R functions
```

```r
#' @param n Number of plays to make
#' @return Total won/lost
play2 <- function(n) {
  if (n < 1) {
    # If 0 (or less rolls) no winning or losing
    return(0)
  }

  die <- sample(1:6, n, replace = TRUE)

  odds <- die %% 2 == 1
  # odds is now a logical vector of length n

  winnings <- die
  # We can replace the odd die rolls with 0 to get the total winnings
  winnings[odds] <- 0
  # Be sure to remove the cost to play
  return(sum(winnings) - 2*n)
}

#' Dice game version 3 - using `table`
#' @param n Number of plays to make
#' @return Total won/lost
play3 <- function(n) {
  if (n < 1) {
    # If 0 (or less rolls) no winning or losing
    return(0)
  }

  die <- sample(1:6, n, replace = TRUE)

  # Convert to a factor to include 0 counts
  die <- table(factor(die, levels = 1:6))

  # Add together winnings, then subtract out the total cost (2 per die)
  out <- die[2]*2 + die[4]*4 + die[6]*6 - 2*n
  # `out` will be named (since the table `die` is, so just remove that for a
  # cleaner output.
  names(out) <- NULL

  return(out)
```

```r
}

#' Dice game version 4 - using apply
#' @param n Number of plays to make
#' @return Total won/lost
play4 <- function(n, seed = NULL) {
  if (n < 1) {
    # If 0 (or less rolls) no winning or losing
    return(0)
  }

  die <- sample(1:6, n, replace = TRUE)

  # use vapply for maximum performance.
  return(-2*n + sum(vapply(die, function(x) {
    if (x %% 2 == 0) {
      return(x)
    } else {
      return(0)
    }
  }, 1)))
}
```

b.

```r
c(play1(3), play2(3), play3(3), play4(3))
```

```
[1] -4  0  4 12
```

```r
c(play1(3000), play2(3000), play3(3000), play4(3000))
```

```
[1] -174  220  262  -28
```

c.

To do this, let's add a `seed` argument to each function.

```r
#' Dice game version 1 - using a loop
#' @param n Number of plays to make
#' @param seed If not `null`, a random seed
```

```r
#' @return Total won/lost
play1seed <- function(n, seed = NULL) {
  if (n < 1) {
    # If 0 (or less rolls) no winning or losing
    return(0)
  }

  set.seed(seed)
  die <- sample(1:6, n, replace = TRUE)

  total <- 0

  for (i in 1:n) {
    total <- total - 2 # cost to play
    if (die[i] %% 2 == 0) {
      total <- total + die[i]
    }
    # if 1,3,5, no change to total besides loss of 2 above
  }

  return(total)
}

#' Dice game version 2 - with vectorized R functions
#' @param n Number of plays to make
#' @param seed If not `null`, a random seed
#' @return Total won/lost
play2seed <- function(n, seed = NULL) {
  if (n < 1) {
    # If 0 (or less rolls) no winning or losing
    return(0)
  }

  set.seed(seed)
  die <- sample(1:6, n, replace = TRUE)

  odds <- die %% 2 == 1
  # odds is now a logical vector of length n

  winnings <- die
  # We can replace the odd die rolls with 0 to get the total winnings
```

```
  winnings[odds] <- 0
  # Be sure to remove the cost to play
  return(sum(winnings) - 2*n)
}

#' Dice game version 3 - using `table`
#' @param n Number of plays to make
#' @param seed If not `null`, a random seed
#' @return Total won/lost
play3seed <- function(n, seed = NULL) {
  if (n < 1) {
    # If 0 (or less rolls) no winning or losing
    return(0)
  }

  set.seed(seed)
  die <- sample(1:6, n, replace = TRUE)

  # Convert to a factor to include 0 counts
  die <- table(factor(die, levels = 1:6))

  # Add together winnings, then subtract out the total cost (2 per die)
  out <- die[2]*2 + die[4]*4 + die[6]*6 - 2*n
  # `out` will be named (since the table `die` is, so just remove that for a
  # cleaner output.
  names(out) <- NULL

  return(out)
}

#' Dice game version 4 - using apply
#' @param n Number of plays to make
#' @param seed If not `null`, a random seed
#' @return Total won/lost
play4seed <- function(n, seed = NULL) {
  if (n < 1) {
    # If 0 (or less rolls) no winning or losing
    return(0)
  }

  set.seed(seed)
```

```r
  die <- sample(1:6, n, replace = TRUE)

  # use vapply for maximum performance.
  return(-2*n + sum(vapply(die, function(x) {
    if (x %% 2 == 0) {
      return(x)
    } else {
      return(0)
    }
  }, 1)))
}
```

```r
c(play1seed(3, seed = 1234),
  play2seed(3, seed = 1234),
  play3seed(3, seed = 1234),
  play4seed(3, seed = 1234))
```

```
[1] 6 6 6 6
```

```r
c(play1seed(3000, seed = 543892),
  play2seed(3000, seed = 543892),
  play3seed(3000, seed = 543892),
  play4seed(3000, seed = 543892))
```

```
[1] -122 -122 -122 -122
```

d.

```r
library(microbenchmark)
microbenchmark(loop   = play1seed(100, seed = 123),
               vctrzd = play2seed(100, seed = 123),
               table  = play3seed(100, seed = 123),
               apply  = play4seed(100, seed = 123))
```

```
Warning in microbenchmark(loop = play1seed(100, seed = 123), vctrzd =
play2seed(100, : less accurate nanosecond times to avoid potential integer
overflows
```

```
Unit: microseconds
   expr    min     lq     mean  median      uq    max neval   cld
   loop 15.252 15.580 16.19295 15.8055 16.1335 24.559   100 a
 vctrzd  7.667  8.159  9.03189  8.4050  9.3070 19.885   100  b
  table 30.914 31.980 33.73152 32.7590 33.4150 68.716   100   c
  apply 34.973 35.506 36.58102 36.0390 36.8590 45.797   100    d
```

```
microbenchmark(loop   = play1seed(10000, seed = 123),
               vctrzd = play2seed(10000, seed = 123),
               table  = play3seed(10000, seed = 123),
               apply  = play4seed(10000, seed = 123))
```

```
Unit: microseconds
   expr      min         lq      mean   median        uq      max neval   cld
   loop 1104.376 1128.6890 1267.2251 1156.405 1227.9910 2931.705   100 a
 vctrzd  298.398  319.9025  334.4395  328.615  340.2385  465.719   100  b
  table  496.838  527.1780  555.0715  538.166  554.6890 1056.939   100   c
  apply 2994.927 3057.0215 3300.2368 3109.604 3278.8930 5616.139   100    d
```

Your results may vary of course. As the sample size changes, the performance of the loop vs the table reverses. Apply's is always worst, vectorization is unsurprisingly always best.

Just out of curiousity, does setting the seed affect performance?

```
microbenchmark(noseed   = play2(100),
               nullseed = play2seed(100),
               seed     = play2seed(100, seed = 123))
```

```
Unit: microseconds
     expr   min     lq    mean median     uq    max neval cld
   noseed 5.863 6.1705 6.68751 6.5600 7.0520  8.774   100 a
 nullseed 8.200 8.7535 9.45583 9.1020 9.8810 20.295   100  b
     seed 7.585 7.8720 8.42263 8.0155 8.7945 18.614   100   c
```

```
microbenchmark(noseed   = play2(10000),
               nullseed = play2seed(10000),
               seed     = play2seed(10000, seed = 123))
```

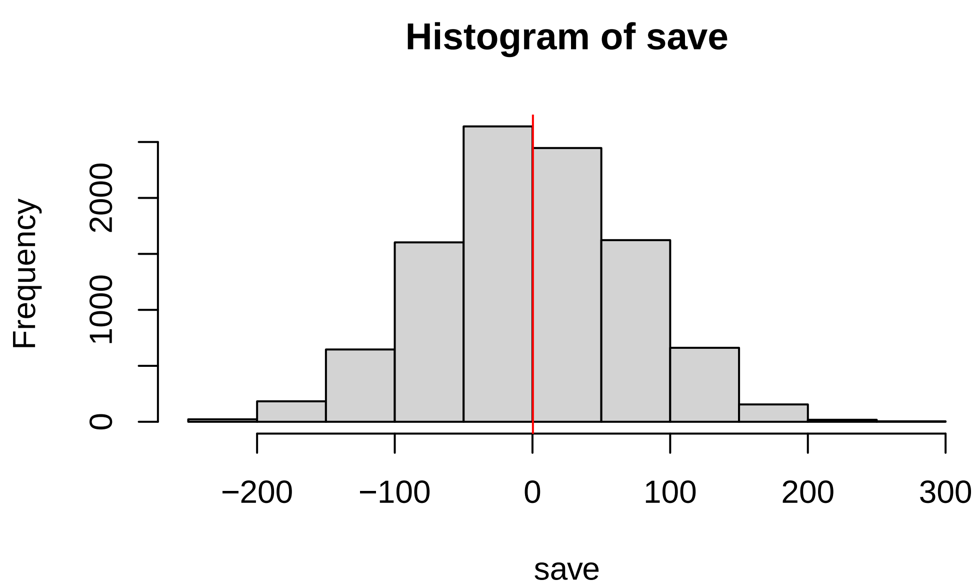
```
Unit: microseconds
     expr     min      lq     mean   median       uq      max neval cld
   noseed 319.062 327.918 339.5768 335.0110 344.1745  412.870   100   a
 nullseed 322.875 330.870 372.5994 336.6920 346.0195 3362.328   100   a
     seed 321.276 329.681 367.1062 335.3595 341.9400 3089.760   100   a
```

With low number of dice, the time it takes the work with the seed does appear to be non-trivial. However, as the number increases, the time for the seed becomes inconsequential.

e.

Let's run a Monte Carlo simulation to see what our winnings or loses average out to.

```r
reps <- 10000
save <- vector(length = reps)
for (i in 1:reps) {
  save[i] <- play2(1000) # use the fastest version
}
hist(save)
abline(v = mean(save), col = "red")
```



The game looks fair! We can of course see this via combinatorics:

$$E(\text{winnings}) = \frac{3}{6} * 0 + \frac{1}{6} * 2 + \frac{1}{6} * 4 + \frac{1}{6} * 6 - 2 = 0$$

Problem 2 Solutions - Linear Regression

```
cars <- read.csv("data/cars.csv")
```

a.

```
names(cars)
```

```
 [1] "Dimensions.Height"
 [2] "Dimensions.Length"
 [3] "Dimensions.Width"
 [4] "Engine.Information.Driveline"
 [5] "Engine.Information.Engine.Type"
 [6] "Engine.Information.Hybrid"
 [7] "Engine.Information.Number.of.Forward.Gears"
 [8] "Engine.Information.Transmission"
 [9] "Fuel.Information.City.mpg"
[10] "Fuel.Information.Fuel.Type"
[11] "Fuel.Information.Highway.mpg"
[12] "Identification.Classification"
[13] "Identification.ID"
[14] "Identification.Make"
[15] "Identification.Model.Year"
[16] "Identification.Year"
[17] "Engine.Information.Engine.Statistics.Horsepower"
[18] "Engine.Information.Engine.Statistics.Torque"
```

```
names(cars) <- c("height", "length", "width", "driveline", "engine_type",
                 "hybrid", "gears", "transmission", "mpg_city", "fuel",
                 "mpg_hwy", "class", "ID", "make", "model_and_year", "year",
                 "horsepower", "torque")
```

b.

```
table(cars$fuel)
```

```
Compressed natural gas            Diesel fuel                      E85
                     2                     27                      456
              Gasoline
                  4591
```

```r
gascars <- cars[cars$fuel == "Gasoline", ]
nrow(gascars)
```

```
[1] 4591
```

c.

```r
mod <- lm(mpg_hwy ~ horsepower + torque + height + length + width +
            as.factor(year), data = gascars)
summary(mod)
```

```

Call:
lm(formula = mpg_hwy ~ horsepower + torque + height + length + 
    width + as.factor(year), data = gascars)

Residuals:
    Min      1Q  Median      3Q     Max 
-10.824  -2.550  -0.452   2.372 202.639 

Coefficients:
                      Estimate Std. Error t value Pr(>|t|)    
(Intercept)         32.2926630  0.7225982  44.690  < 2e-16 ***
horsepower           0.0163556  0.0022772   7.182 7.96e-13 ***
torque              -0.0507425  0.0022030 -23.034  < 2e-16 ***
height               0.0099079  0.0011267   8.794  < 2e-16 ***
length               0.0017290  0.0008836   1.957   0.0504 .  
width               -0.0003343  0.0009045  -0.370   0.7117    
as.factor(year)2010 -0.4539681  0.6768246  -0.671   0.5024    
as.factor(year)2011  0.1711016  0.6757043   0.253   0.8001    
as.factor(year)2012  1.3029279  0.6810076   1.913   0.0558 .  
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 4.602 on 4582 degrees of freedom
Multiple R-squared:  0.4192,	Adjusted R-squared:  0.4182 
F-statistic: 413.3 on 8 and 4582 DF,  p-value: < 2.2e-16
```

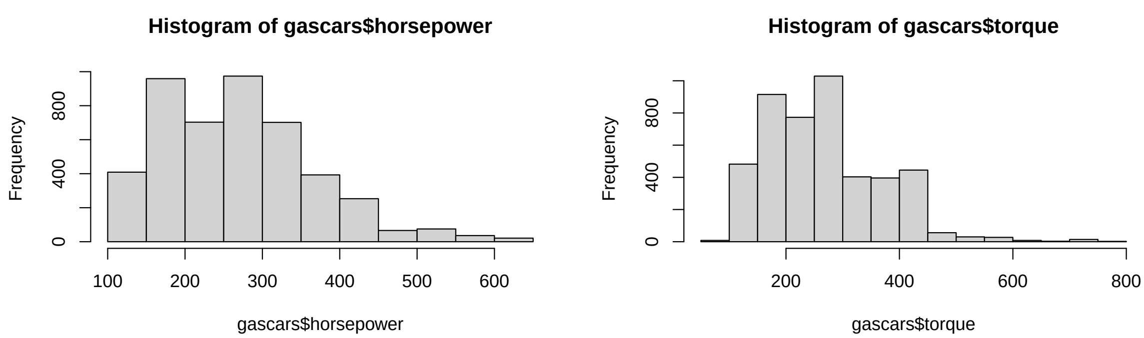
We see a signficant positive relationship - higher horsepower is predicted to yield higher highway mileage, on average.

d.

```
mod <- lm(mpg_hwy ~ horsepower*torque + height + length + width +
            as.factor(year), data = gascars)
```

To choose reasonable values for horsepower and torque, let's look at histograms.

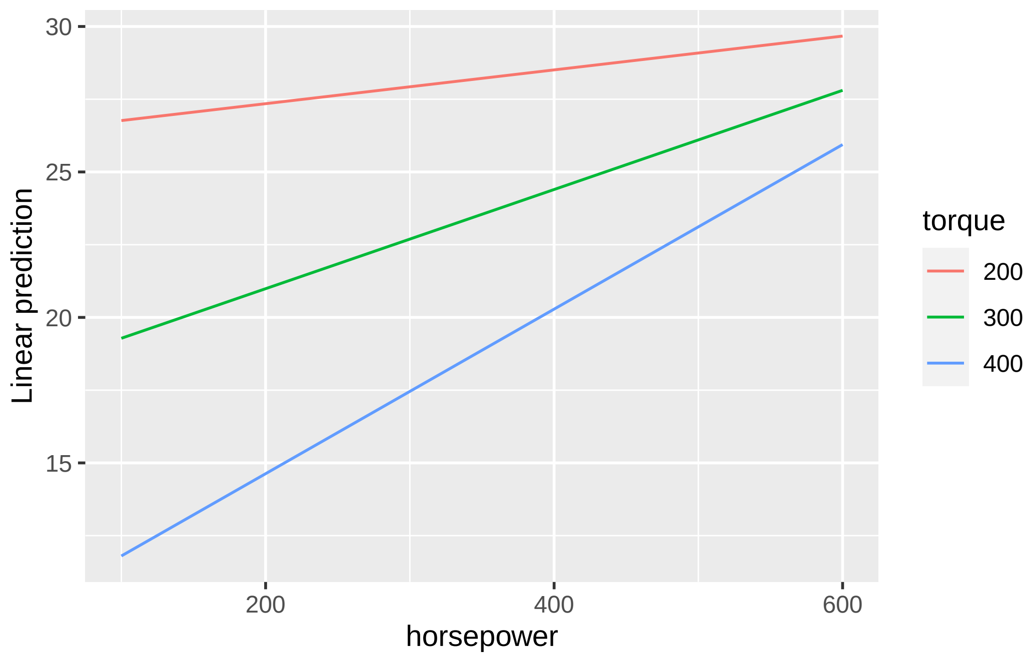
```
hist(gascars$horsepower)
hist(gascars$torque)
```



Horsepower goes from about 100 to 600.

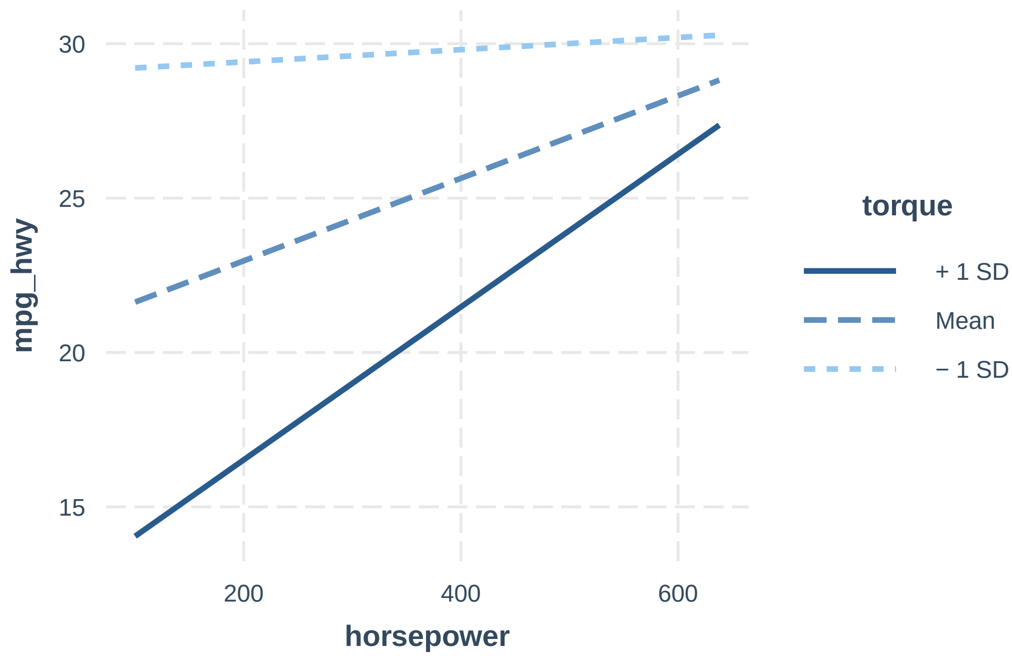
Torque goes from about 150 to 750, but its extremely rare above 400, so we'll restrict to that range.

```
library(emmeans)
emmip(mod, torque ~ horsepower, at = list(horsepower = seq(100, 600, 100),
                                          torque = c(200, 300, 400)))
```




```
library(interactions)
interact_plot(mod, pred = horsepower, modx = torque,
              at = list(year = 2011))
```

```
Using data gascars from global environment. This could cause incorrect
results if gascars has been altered since the model was fit. You can
manually provide the data to the "data =" argument.
```



e.

We can take the same formula, and use it to generate design matrix X.

```
X <- model.matrix(mpg_hwy ~ horsepower*torque + height + length + width +
                   as.factor(year), data = gascars)
y <- gascars$mpg_hwy
betahat <- solve(t(X)%*%X)%*%t(X)%*%y
cbind(mod$coef, betahat)
```

```
                            [,1]          [,2]
(Intercept)         42.1879478687 42.1879478687
horsepower          -0.0166633227 -0.0166633227
torque              -0.0860592704 -0.0860592704
height               0.0065603903  0.0065603903
length               0.0017767232  0.0017767232
width               -0.0011694485 -0.0011694485
as.factor(year)2010 -0.5627857770 -0.5627857770
as.factor(year)2011  0.0725356431  0.0725356431
as.factor(year)2012  1.1970329986  1.1970329986
horsepower:torque    0.0001123567  0.0001123567
```

Problem 3 Solutions - Stata

The complete .Do file can be found here. The results are included in each section below.

I imported the data via the menu, it generated this code:

```
import delimited "/Users/josh/repositories/_teaching/506-f23/data/cars.csv", clear
```

a.

```
. rename dimensionsheight height
. rename dimensionslength length
. rename dimensionswidth width
. rename engineinformationdriveline driveline
. rename engineinformationenginetype engine_type
. rename engineinformationhybrid hybrid
. rename engineinformationnumberofforward gears
. rename engineinformationtransmission transmission
. rename fuelinformationcitympg mpg_city
. rename fuelinformationfueltype fuel
. rename fuelinformationhighwaympg mpg_hwy
. rename identificationclassification class
. rename identificationid ID
. rename identificationmake make
. rename identificationmodelyear model_and_year
. rename identificationyear year
. rename engineinformationenginestatistic horsepower
. rename v18 torque
```

b.

```
. tab fuel

 Fuel Information.Fuel |
                  Type |      Freq.     Percent        Cum.
-----------------------+-----------------------------------
Compressed natural gas |          2        0.04        0.04
           Diesel fuel |         27        0.53        0.57
                   E85 |        456        8.98        9.55
              Gasoline |      4,591       90.45      100.00
-----------------------+-----------------------------------
                 Total |      5,076      100.00
```

```
. keep if fuel == "Gasoline"
(485 observations deleted)

. count
  4,591
```

c.

```
. regress mpg_hwy horsepower torque height length width i.year

      Source |       SS           df       MS      Number of obs   =     4,591
-------------+----------------------------------   F(8, 4582)      =    413.35
       Model |  70043.6695         8  8755.45869   Prob > F        =    0.0000
    Residual |   97055.298     4,582  21.1818634   R-squared       =    0.4192
-------------+----------------------------------   Adj R-squared   =    0.4182
       Total |  167098.968     4,590  36.4050038   Root MSE        =    4.6024

------------------------------------------------------------------------------
     mpg_hwy | Coefficient  Std. err.      t    P>|t|     [95% conf. interval]
-------------+----------------------------------------------------------------
  horsepower |   .0163556   .0022772     7.18   0.000     .0118913      .02082
      torque |  -.0507425    .002203   -23.03   0.000    -.0550614   -.0464236
      height |   .0099079   .0011267     8.79   0.000      .007699    .0121168
      length |    .001729   .0008836     1.96   0.050    -3.36e-06    .0034613
       width |  -.0003343   .0009045    -0.37   0.712    -.0021075    .0014388
             |
        year |
       2010  |  -.4539681   .6768246    -0.67   0.502     -1.78087    .8729342
       2011  |   .1711016   .6757043     0.25   0.800    -1.153604    1.495808
       2012  |   1.302928   .6810076     1.91   0.056    -.0321751    2.638031
             |
       _cons |   32.29266   .7225982    44.69   0.000     30.87602     33.7093
------------------------------------------------------------------------------
```

We get the same results as in R. (Note that this may not be the case for models solved by iterative optimzation, but will be the case for least squares. R and Stata use slightly different algorithms for optimizations - the results should be extremely similar (to the point that differences are almost always ignorable) but you shouldn't expect identical results like we get here.)

d.

The code for the histograms is below but I'm not including the output - it looks identical to

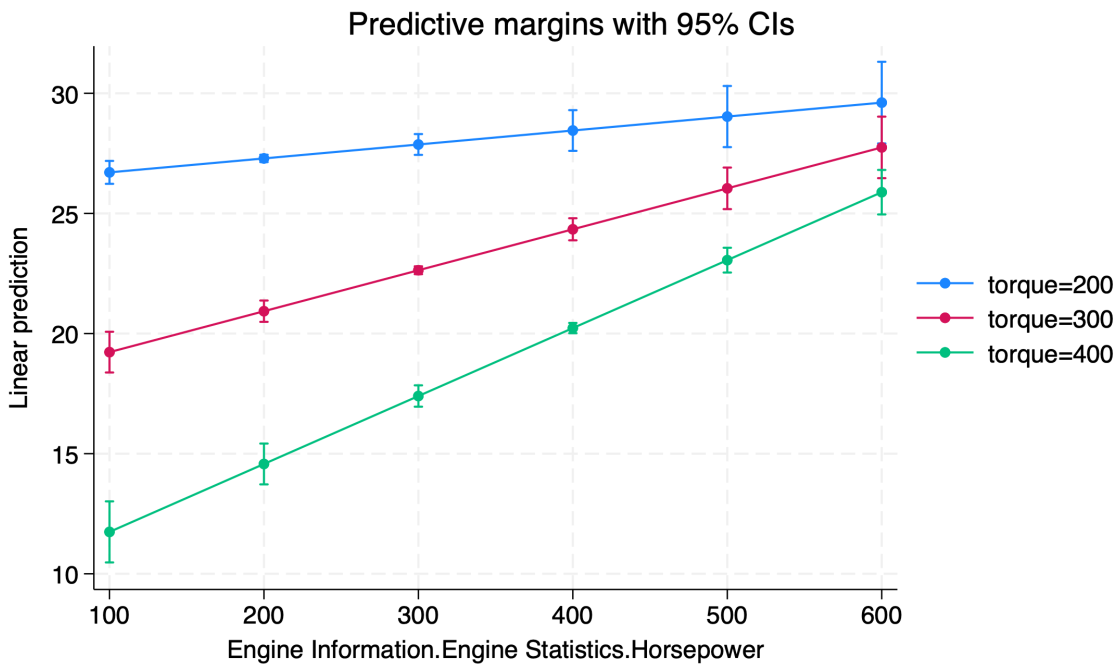
R's of course.

```
. histogram horsepower
. histogram torque
. regress mpg_hwy c.horsepower##c.torque height length width i.year

      Source |       SS           df       MS      Number of obs   =     4,591
-------------+----------------------------------   F(9, 4581)      =    480.07
       Model |  81105.8715         9  9011.76351   Prob > F        =    0.0000
    Residual |   85993.096     4,581  18.7716865   R-squared       =    0.4854
-------------+----------------------------------   Adj R-squared   =    0.4844
       Total |  167098.968     4,590  36.4050038   Root MSE        =    4.3326

------------------------------------------------------------------------------------------
                  mpg_hwy | Coefficient  Std. err.      t    P>|t|     [95% conf. interval]
--------------------------+---------------------------------------------------------------
               horsepower |  -.0166633   .0025388    -6.56   0.000    -.0216406    -.011686
                   torque |  -.0860593   .0025333   -33.97   0.000    -.0910257   -.0810928
                          |
c.horsepower#c.torque |   .0001124   4.63e-06    24.28   0.000     .0001033    .0001214
                          |
                   height |   .0065604   .0010696     6.13   0.000     .0044634    .0086573
                   length |   .0017767   .0008318     2.14   0.033     .0001459    .0034075
                    width |  -.0011694   .0008521    -1.37   0.170      -.00284    .0005011
                          |
                     year |
                    2010  |  -.5627858   .6371716    -0.88   0.377    -1.811949    .6863777
                    2011  |   .0725356   .6361142     0.11   0.909    -1.174555    1.319626
                    2012  |   1.197033   .6411085     1.87   0.062    -.0598488    2.453915
                          |
                    _cons |   42.18795   .7930274    53.20   0.000     40.63323    43.74266
------------------------------------------------------------------------------------------

. quietly margins, at(horsepower = (100(100)600) torque = (200 300 400))
. marginsplot
```



e.

```
. quietly tabulate year, gen(yr) // Generate dummy variables for year
. generate horsepower_torque = horsepower*torque // Generate interaction term
. generate intercept = 1 // Generate an intercept
```

Next, store the X and y matrix as matrix objects.

```
. mkmat intercept horsepower torque horsepower_torque height length width yr2 yr3 yr4, mat
. mkmat mpg_hwy, matrix(y)
```

Drop down to mata for the actual computation.

```
. mata:
-------------------- mata (type end to exit) --------------------
: X = st_matrix("X")
: y = st_matrix("y")
: invsym(X'*X)*X'*y
                  1
     +-----------------+
   1 |   42.18794787   |
   2 |  -.0166633227   |
```

```
   3 |  -.0860592704  |
   4 |    .0001123567  |
   5 |    .0065603903  |
   6 |    .0017767232  |
   7 |  -.0011694485  |
   8 |   -.562785777  |
   9 |    .0725356431  |
  10 |    1.197032999  |
     +----------------+
: end
-----------------------------------------------------------------
```