Homework1 - Machine Learning

Kamal

Homework 1

Marlena Kuhn

Some informative viewing

I highly recommend watching this series by 3blue1brown to get a sense of what matrix operations are from a vector space perspective and how to visualize them: Essence of Linear Algebra . So far, we will cover the first 4 videos in the series, the rest are enrichment . . . or review if you've already done statistics. We might cover topics like convolution and polynomial multiplication when we get to deep learning, which are a bit different from the vector representation of matrices. The foundation you build here will be helpful then.

Matrix Multiplication (2.5pts)

You are a linear algebra expert from Youtube university and have been hired by a local space cadet to help them navigate some vector spaces. They have provided you with the following matrices:

```
A <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, byrow = TRUE)
B <- matrix(c(7, 8, 9, 10, 11, 12), nrow = 3, byrow = TRUE)
```

1. Compute the matrix product $C = AB$. What are the dimensions of C? (0.5pts)

```
#Computing matrix product C and printing it
C<- A%*%B
print(C)
```

```
##      [,1] [,2]
## [1,]   58   64
## [2,]  139  154
```

```
#Determining the number of rows and columns in matrix C
nrow(C)
```

```
## [1] 2
```

```
ncol(C)
```

```
## [1] 2
```

```
#The dimensions of \(C\) are 2 by 2.
```

2. Compute the outer product of the first column of A with the first row of B. What is the result? (0.5pts)

```
A
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
```

```
B
```

```
##      [,1] [,2]
## [1,]    7    8
## [2,]    9   10
## [3,]   11   12
```

```
#Subsetting the first column of A, which should have 2 numbers.
A_first_col<- A[, 1]

#Subsetting the first row of B, which should have 2 numbers.
B_first_row<- B [1 ,]

#Calculating the outer product
outer_firsts<- A_first_col%o%B_first_row

#Printing the results
print(outer_firsts)
```

```
##      [,1] [,2]
## [1,]    7    8
## [2,]   28   32
```

3. Verify that the matrix product C can be expressed as a sum of outer products of the columns of A and the rows of B. (0.5pts)

```
#Calculating outer product for second column of A and second row of B
A_sec_col<- A[, 2]
B_sec_row<- B [2 ,]
outer_secs<- A_sec_col%o%B_sec_row

#Calculating the outer product for the third column of A and the third row of B
A_third_col<- A[, 3]
B_third_row<- B [3 ,]
outer_thirds<- A_third_col%o%B_third_row

#Calculating the sum of (the three) outer products of the A columns and B rows
sum_of_outer<- outer_firsts + outer_secs + outer_thirds

#Checking is C and the sum of the outer products are the same
C== sum_of_outer
```

```
##      [,1] [,2]
## [1,] TRUE TRUE
## [2,] TRUE TRUE
```

```
#Yes, C can be expressed as the sum of outer products.
```

4. Explain in your own words the relationship between matrix multiplication and outer products. (0.5pts)
   A- Matrix multiplication is the sum of outer products, and outer products are multiplications done to a subset of matrices.

5. Implement a function that takes two matrices as input and returns their product using the outer product method. (0.5pts)

```
matrix_multiply_outer <- function(A, B) {
  #The number of outer products that need to be done is the number of columns in
    #... A (or number of rows in B, as they should be the same)
  n<- ncol(A)

  #Creating a matrix shell to put the sums into that is number of rows in A by
    #... number of columns in B
  sum_outer<- matrix(0, ncol= ncol(B), nrow= nrow(A))

  #Create a loop to calculate the outer products of certain A columns and B rows
    #... and then add them together
  for (i in 1:n){
    #Subset the specific column from A and row from B
    A_col<- A[, i]
    B_row<- B [i ,]
    #Calculate the outer product
    outer<- A_col%o%B_row
    #Summing the outer products to sum_outer and keeping the new sum_outer
      #... values for the next loop
    new_sum<- sum_outer + outer
    sum_outer<- new_sum
  }
  #The desired result is sum_outer after the loop
  return(sum_outer)
}


#Testing the function using the two matrices defined at the start.
test<-
  matrix_multiply_outer(matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, byrow = TRUE),
                        matrix(c(7, 8, 9, 10, 11, 12), nrow = 3, byrow = TRUE))
#Checking that this result is equal to the previously calculated C.
C== test
```

```
##      [,1] [,2]
## [1,] TRUE TRUE
## [2,] TRUE TRUE
```

```
#As the C from the function matches the C using the operator, my function works
```

Chick-Weight (7.5pts)

You are a farmer named Bob and your chickens are getting too fat. Analyze the ChickWeight dataset in R before your chickens get taken away by the ASPCA. Your task is to explore the relationship between diet and weight gain over time in chicks. Perform the following analyses:

Load the ChickWeight dataset and display its structure. If it's not already included, you can load the attached chick_weight.csv

```
library(ggplot2)
library(ggcorrplot)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
data("ChickWeight")
str(ChickWeight)
```

```
## Classes 'nfnGroupedData', 'nfGroupedData', 'groupedData' and 'data.frame':   578 obs. of  4 variables
##  $ weight: num  42 51 59 64 76 93 106 125 149 171 ...
##  $ Time  : num  0 2 4 6 8 10 12 14 16 18 ...
##  $ Chick : Ord.factor w/ 50 levels "18"<"16"<"15"<..: 15 15 15 15 15 15 15 15 15 15 ...
##  $ Diet  : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 1 1 ...
##  - attr(*, "formula")=Class 'formula'  language weight ~ Time | Chick
##   .. ..- attr(*, ".Environment")=<environment: R_EmptyEnv>
##  - attr(*, "outer")=Class 'formula'  language ~Diet
##   .. ..- attr(*, ".Environment")=<environment: R_EmptyEnv>
##  - attr(*, "labels")=List of 2
##   ..$ x: chr "Time"
##   ..$ y: chr "Body weight"
##  - attr(*, "units")=List of 2
##   ..$ x: chr "(days)"
##   ..$ y: chr "(gm)"
```

```
head(ChickWeight)
```

```
##   weight Time Chick Diet
## 1     42    0     1    1
## 2     51    2     1    1
```

```
## 3     59    4     1    1
## 4     64    6     1    1
## 5     76    8     1    1
## 6     93   10     1    1
```

```
# If you need to load from a CSV file, uncomment the line below
# ChickWeight <- read.csv("chick_weight.csv")
```

1. Create a summary table showing the average weight of chicks for each diet at each time point. (0.5pts)

```
#In my mind, there are two types of summary tables (one where the rows and
  #... columns are diets and times and the all of the entries are the averages
  #... and one where each row has a diet, a date, and an average).



#This is the entries are averages only table.

#Create a list of the different time points
time_list<- unique(ChickWeight$Time)
print (time_list)
```

```
##  [1]  0  2  4  6  8 10 12 14 16 18 20 21
```

```
#Create a list of the different diets
diet_list<- unique(ChickWeight$Diet)
print(diet_list)
```

```
## [1] 1 2 3 4
## Levels: 1 2 3 4
```

```
#Create summary table dataframe with columns named after the time points and
  #... rows names after the diets (and then convert to table)
avg_weight<- data.frame(matrix(0, ncol= length(time_list),
                               nrow= length(diet_list)))
colnames(avg_weight)<- time_list
rownames(avg_weight)<- diet_list

#Loop to populate weight means at each timestep for diet 1
for (i in 1:length(time_list)){
  time<- time_list[i]
  #Subset ChickWeight for entries with Diet 1 only
  diet_1<- ChickWeight[ChickWeight$Diet==1 ,]
  #Subset Diet 1 entries for entries with time equal to place in loop
  diet_1_time_i<- diet_1[diet_1$Time==time ,]
  #Calculate mean and put in avg_weight table
  avg_d1<- mean(diet_1_time_i$weight)
  avg_weight[1, i]<- avg_d1
}

#Loop to populate weight means at each timestep for diet 2
for (i in 1:length(time_list)){
```

```
  time<- time_list[i]
  diet_2<- ChickWeight[ChickWeight$Diet==2 ,]
  diet_2_time_i<- diet_2[diet_2$Time==time ,]
  avg_d2<- mean(diet_2_time_i$weight)
  avg_weight[2, i]<- avg_d2
}

#Loop to populate weight means at each timestep for diet 3
for (i in 1:length(time_list)){
  time<- time_list[i]
  diet_3<- ChickWeight[ChickWeight$Diet==3 ,]
  diet_3_time_i<- diet_3[diet_3$Time==time ,]
  avg_d3<- mean(diet_3_time_i$weight)
  avg_weight[3, i]<- avg_d3
}

#Loop to populate weight means at each timestep for diet 4
for (i in 1:length(time_list)){
  time<- time_list[i]
  diet_4<- ChickWeight[ChickWeight$Diet==4 ,]
  diet_4_time_i<- diet_4[diet_4$Time==time ,]
  avg_d4<- mean(diet_4_time_i$weight)
  avg_weight[4, i]<- avg_d4
}

head(avg_weight)
```

```
##      0     2        4        6         8        10       12       14       16
## 1 41.4 47.25 56.47368 66.78947  79.68421  93.05263 108.5263 123.3889 144.6471
## 2 40.7 49.40 59.80000 75.40000  91.70000 108.50000 131.3000 141.9000 164.7000
## 3 40.8 50.40 62.20000 77.90000  98.40000 117.10000 144.4000 164.5000 197.4000
## 4 41.0 51.80 64.50000 83.90000 105.60000 126.00000 151.4000 161.8000 182.0000
##         18       20       21
## 1 158.9412 170.4118 177.7500
## 2 187.7000 205.6000 214.7000
## 3 233.1000 258.9000 270.3000
## 4 202.9000 233.8889 238.5556
```

```
#This is the row has diet, time, and average table.

#Create a data frame and populate each diet number repeated by the times amount
diet_reps<- c(rep(diet_list[1], times= length(time_list)), rep(diet_list[2],
                            times= length(time_list)),rep(diet_list[3],
                            times= length(time_list)), rep(diet_list[4],
                            times= length(time_list)))

d_t_avg<- data_frame(Diet= diet_reps,
                     Time= NA,
                     Average_Weight= NA)
```

```
## Warning: `data_frame()` was deprecated in tibble 1.1.0.
## i Please use `tibble()` instead.
```

```
## This warning is displayed once every 8 hours.
## Call 'lifecycle::last_lifecycle_warnings()' to see where this warning was
## generated.
```

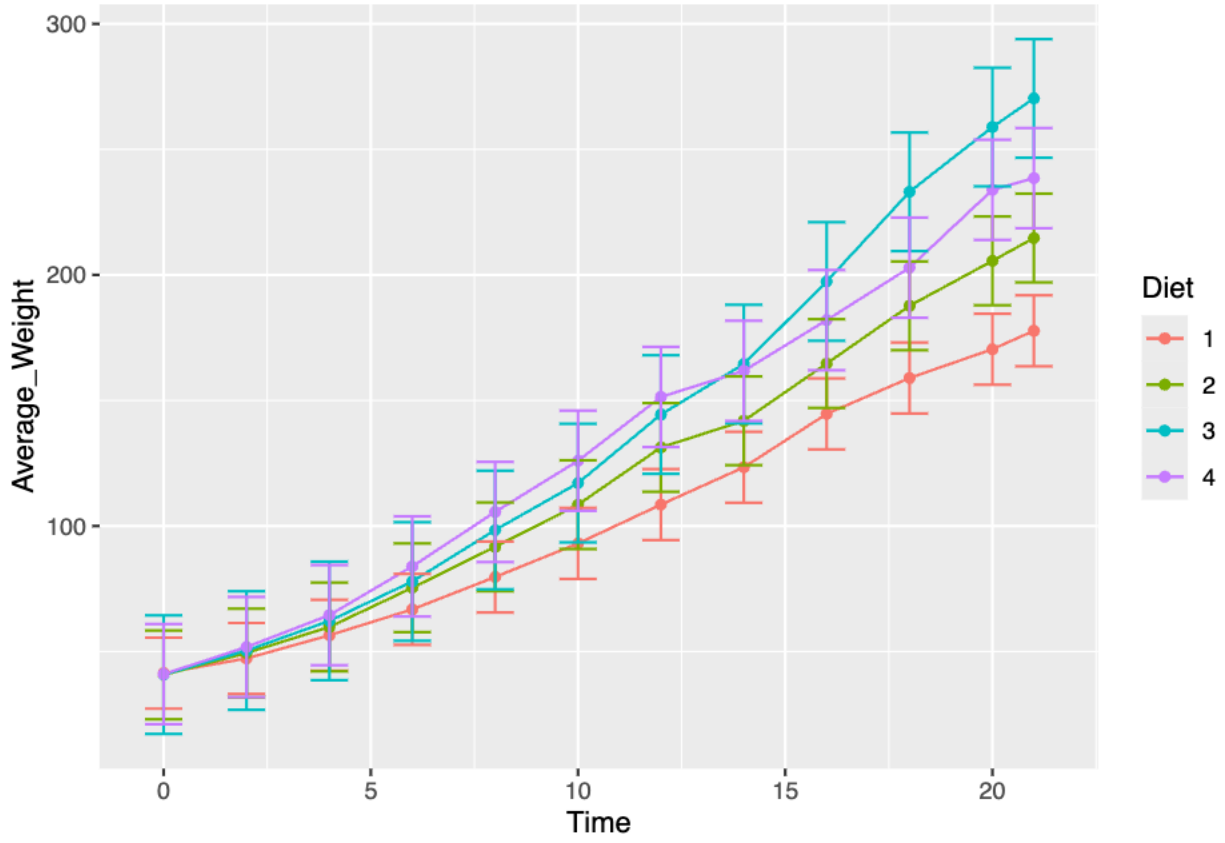
```
#Populate time values with a loop
for (i in 1:length(time_list)){
  d_t_avg$Time[i]<- time_list[i]
  d_t_avg$Time[(i + length(time_list))]<- time_list[i]
  d_t_avg$Time[(i + 2*length(time_list))]<- time_list[i]
  d_t_avg$Time[(i + 3*length(time_list))]<- time_list[i]
}

#Calculate and populate teh weight averages with a loop
for (i in 1:nrow(d_t_avg)){
  #Set the diet and time for each row as variables
  d<- d_t_avg$Diet[i]
  t<- d_t_avg$Time[i]
  #Subset the origional chick weight data set by the specific diet and time
  d_subset<- ChickWeight[ChickWeight$Diet==d ,]
  d_t_subset<- d_subset[d_subset$Time==t ,]
  #Calculate and populate the averages
  avg<- mean(d_t_subset$weight)
  d_t_avg$Average_Weight[i]<- avg
}

head(d_t_avg)
```

```
## # A tibble: 6 x 3
##   Diet   Time Average_Weight
##   <fct> <dbl>          <dbl>
## 1 1         0           41.4
## 2 1         2           47.2
## 3 1         4           56.5
## 4 1         6           66.8
## 5 1         8           79.7
## 6 1        10           93.1
```

2. Visualize the weight gain over time for each diet using a line plot with error bars representing the standard error of the mean using ggplot. (0.5pts)

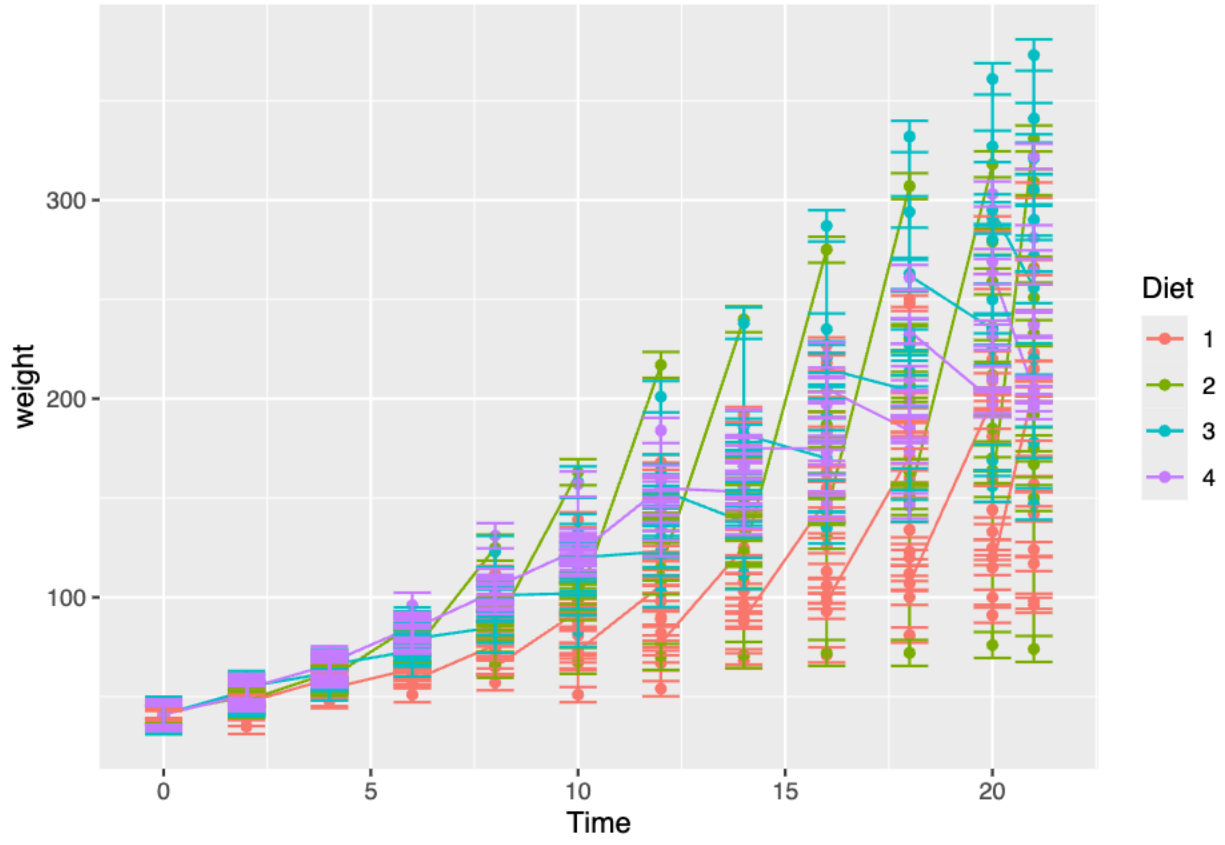
```
#Line plot with error bars for average weights for each diet over time

#Calculate the standard error for each diet group (standard deviation/sqrt n)
d_1<- d_t_avg[d_t_avg$Diet==1 ,]
sd_1<- sd(d_1$Average_Weight)
se_1<- sd_1/sqrt(nrow(d_1))

d_2<- d_t_avg[d_t_avg$Diet==2 ,]
sd_2<- sd(d_2$Average_Weight)
se_2<- sd_2/sqrt(nrow(d_2))

d_3<- d_t_avg[d_t_avg$Diet==3 ,]
sd_3<- sd(d_3$Average_Weight)
```

```r
se_3<- sd_3/sqrt(nrow(d_3))

d_4<- d_t_avg[d_t_avg$Diet==4 ,]
sd_4<- sd(d_4$Average_Weight)
se_4<- sd_4/sqrt(nrow(d_4))

#Plot average weight vs time to examine the change (grouped by diet)
ggplot(data= d_t_avg, aes(x=Time, y=Average_Weight, group_by(Diet),
                          color= Diet))+
  geom_line()+
  geom_point()+
  #Separate error bars representing standard error of each diet group
  geom_errorbar(data= d_t_avg[d_t_avg$Diet==1 ,], aes(ymax= Average_Weight+se_1,
                                                      ymin= Average_Weight-se_1))+
  geom_errorbar(data= d_t_avg[d_t_avg$Diet==2 ,], aes(ymax= Average_Weight+se_2,
                                                      ymin= Average_Weight-se_2))+
  geom_errorbar(data= d_t_avg[d_t_avg$Diet==3 ,], aes(ymax= Average_Weight+se_3,
                                                      ymin= Average_Weight-se_3))+
  geom_errorbar(data= d_t_avg[d_t_avg$Diet==4 ,], aes(ymax= Average_Weight+se_4,
                                                      ymin= Average_Weight-se_4))
```

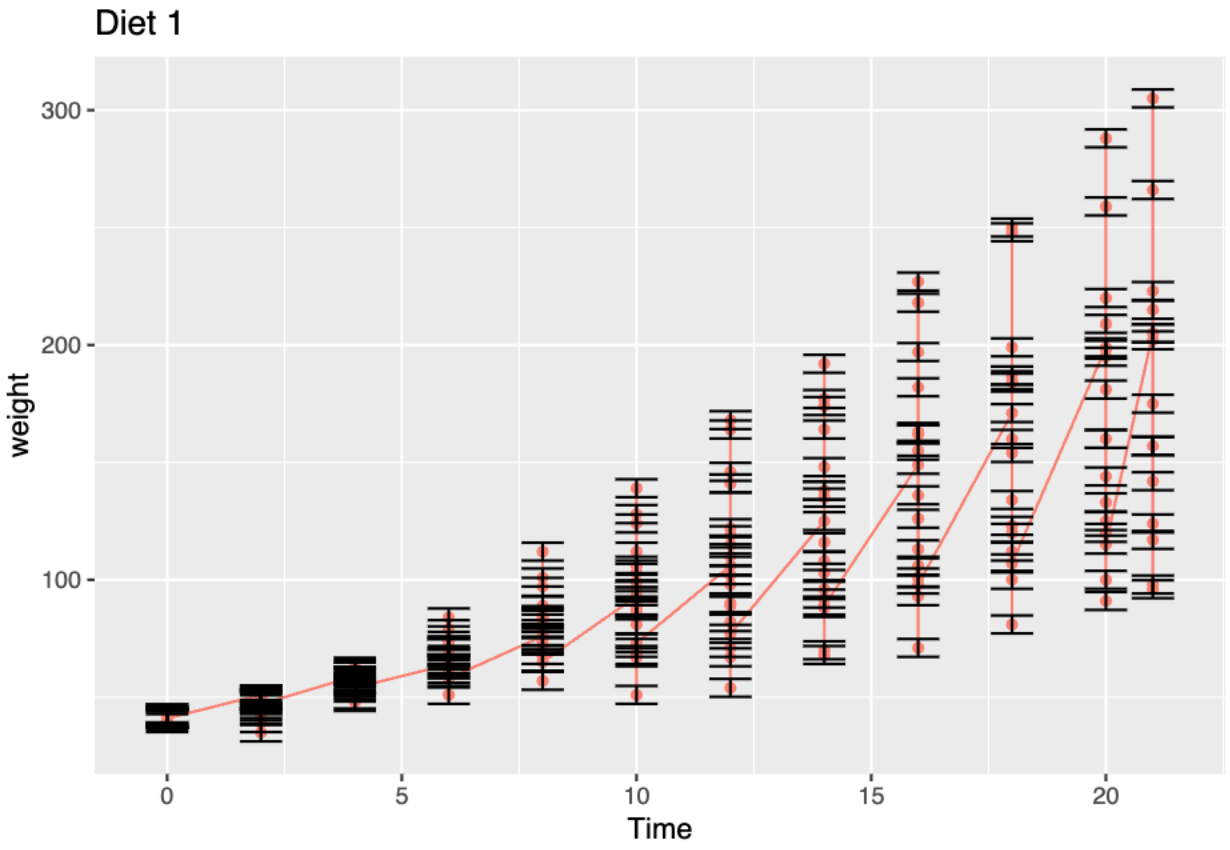



```r
#Line plot with error bars for Chick Weight

#Calculate the standard error for each diet (standard error = sd/ sample size)
fsd_1<- sd(ChickWeight[ChickWeight$Diet==1 ,]$weight)
```

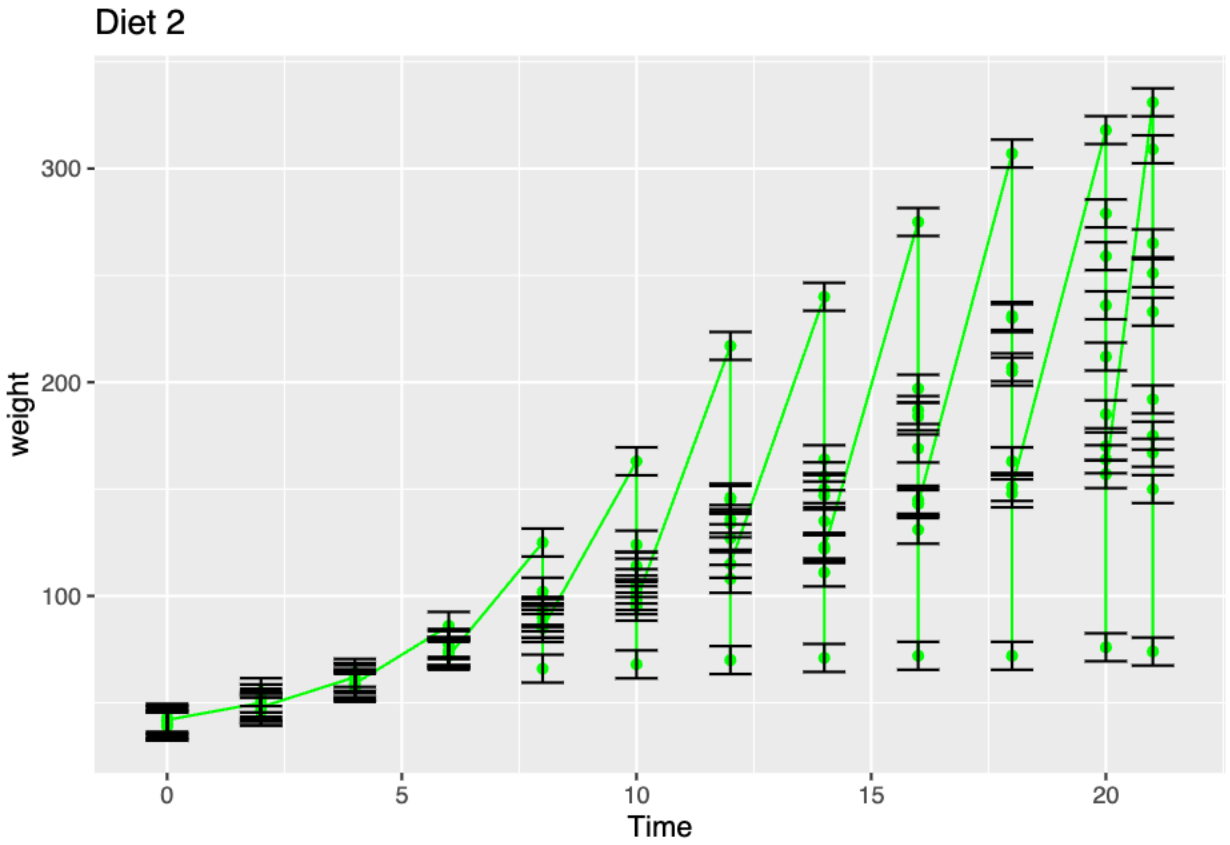
```r
fse_1<- fsd_1/sqrt(nrow(ChickWeight[ChickWeight$Diet==1 ,]))

fsd_2<- sd(ChickWeight[ChickWeight$Diet==2 ,]$weight)
fse_2<- fsd_2/sqrt(nrow(ChickWeight[ChickWeight$Diet==2 ,]))

fsd_3<- sd(ChickWeight[ChickWeight$Diet==3 ,]$weight)
fse_3<- fsd_3/sqrt(nrow(ChickWeight[ChickWeight$Diet==3 ,]))

fsd_4<- sd(ChickWeight[ChickWeight$Diet==4 ,]$weight)
fse_4<- fsd_4/sqrt(nrow(ChickWeight[ChickWeight$Diet==4 ,]))

#Plot weight vs time separated by diet
ggplot(data= ChickWeight, aes(x=Time, y=weight, group_by(Diet), color= Diet))+
  geom_line()+
  geom_point()+
  #Separate error bars representing standard error of each diet group
  geom_errorbar(data= ChickWeight[ChickWeight$Diet==1 ,],
                aes(ymax= weight+fse_1, ymin= weight-fse_1))+
  geom_errorbar(data= ChickWeight[ChickWeight$Diet==2 ,],
                aes(ymax= weight+fse_2, ymin= weight-fse_2))+
  geom_errorbar(data= ChickWeight[ChickWeight$Diet==3 ,],
                aes(ymax= weight+fse_3, ymin= weight-fse_3))+
  geom_errorbar(data= ChickWeight[ChickWeight$Diet==4 ,],
                aes(ymax= weight+fse_4, ymin= weight-fse_4))
```



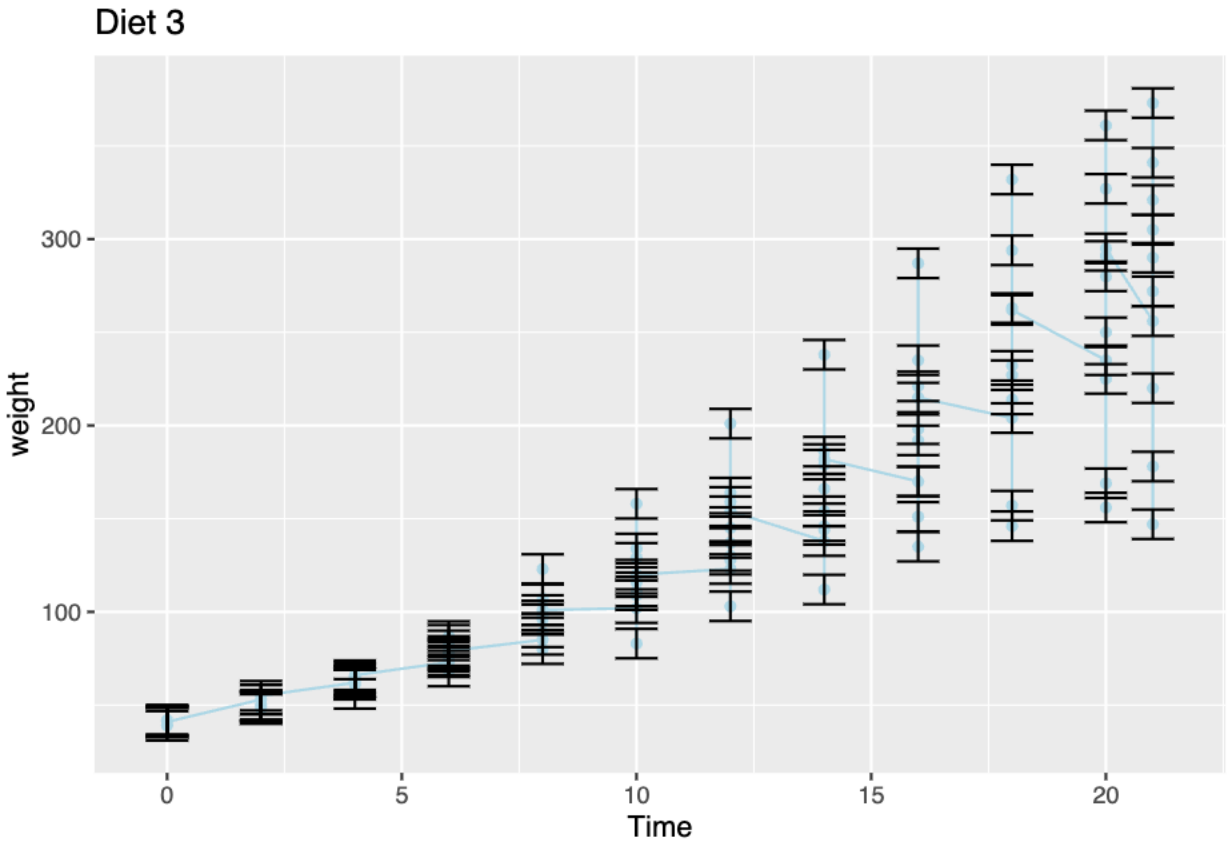
```
#This looks extremely busy, so I am plotting the weight vs times separately for
  #... the diets (with colors corresponding to their original colors in the
  #...combined diet graph)
ggplot(data= ChickWeight[ChickWeight$Diet==1 ,], aes(x=Time, y=weight))+
  geom_line(color= 'salmon')+
  geom_point(color= 'salmon')+
  geom_errorbar(data= ChickWeight[ChickWeight$Diet==1 ,],
                aes(ymax= weight+fse_1, ymin= weight-fse_1))+
  ggtitle("Diet 1")
```



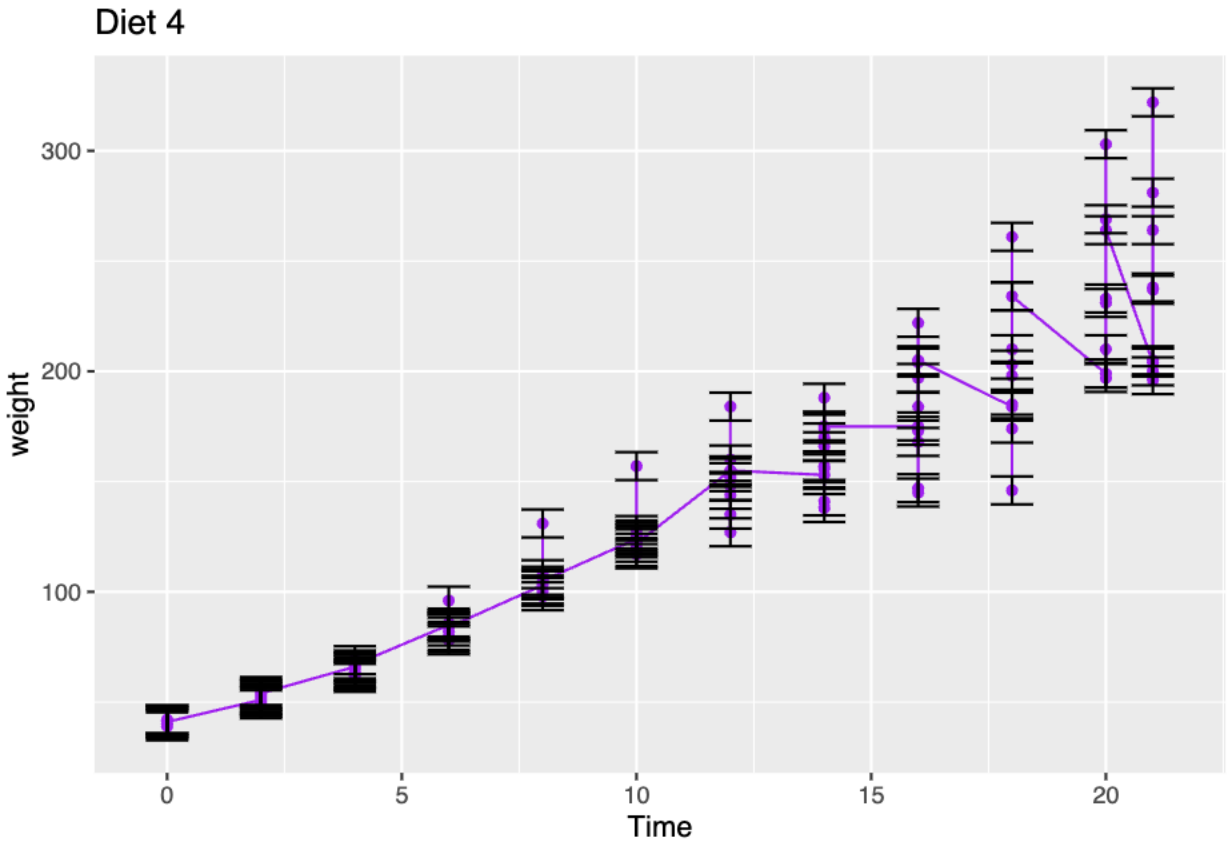

```
ggplot(data= ChickWeight[ChickWeight$Diet==2 ,], aes(x=Time, y=weight))+
  geom_line(color= 'green')+
  geom_point(color= 'green')+
  geom_errorbar(data= ChickWeight[ChickWeight$Diet==2 ,],
                aes(ymax= weight+fse_2, ymin= weight-fse_2))+
  ggtitle("Diet 2")
```




```
ggplot(data= ChickWeight[ChickWeight$Diet==3 ,],
       aes(x=Time, y=weight))+
  geom_line(color= 'lightblue')+
  geom_point(color= 'lightblue')+
  geom_errorbar(data= ChickWeight[ChickWeight$Diet==3 ,],
                aes(ymax= weight+fse_3, ymin= weight-fse_3))+
  ggtitle("Diet 3")
```




```
ggplot(data= ChickWeight[ChickWeight$Diet==4 ,],
       aes(x=Time, y=weight))+
  geom_line(color= 'purple')+
  geom_point(color= 'purple')+
  geom_errorbar(data= ChickWeight[ChickWeight$Diet==4 ,],
                aes(ymax= weight+fse_4, ymin= weight-fse_4))+
  ggtitle("Diet 4")
```



3. Use the ggcorrplot package to create a correlation heatmap of the numeric variables in the dataset. Convert the diet column to dummy variables (columns) and include them in the correlation analysis. (1pts)

```
#Use dummyVars to create dummy variables for diet
require("caret")
```

```
## Loading required package: caret
```

```
## Warning: package 'caret' was built under R version 4.3.3
```

```
## Loading required package: lattice
```

```
## Warning: package 'lattice' was built under R version 4.3.3
```

```
dummy <- dummyVars(" ~ .", data = ChickWeight)
chick_dum <- data.frame(predict(dummy, newdata = ChickWeight))
head(chick_dum)
```

```
##   weight Time    Chick.L     Chick.Q   Chick.C     Chick.4    Chick.5   Chick.6
## 1     42    0 -0.1028992 -0.07445124 0.1666681 -0.06432918 -0.1134107 0.1601967
## 2     51    2 -0.1028992 -0.07445124 0.1666681 -0.06432918 -0.1134107 0.1601967
## 3     59    4 -0.1028992 -0.07445124 0.1666681 -0.06432918 -0.1134107 0.1601967
```

```
## 4     64    6 -0.1028992 -0.07445124 0.1666681 -0.06432918 -0.1134107 0.1601967
## 5     76    8 -0.1028992 -0.07445124 0.1666681 -0.06432918 -0.1134107 0.1601967
## 6     93   10 -0.1028992 -0.07445124 0.1666681 -0.06432918 -0.1134107 0.1601967
##       Chick.7    Chick.8  Chick.9   Chick.10   Chick.11  Chick.12   Chick.13
## 1 -0.02173999 -0.1423219 0.143173 0.02033441 -0.1613266 0.1190363 0.05873516
## 2 -0.02173999 -0.1423219 0.143173 0.02033441 -0.1613266 0.1190363 0.05873516
## 3 -0.02173999 -0.1423219 0.143173 0.02033441 -0.1613266 0.1190363 0.05873516
## 4 -0.02173999 -0.1423219 0.143173 0.02033441 -0.1613266 0.1190363 0.05873516
## 5 -0.02173999 -0.1423219 0.143173 0.02033441 -0.1613266 0.1190363 0.05873516
## 6 -0.02173999 -0.1423219 0.143173 0.02033441 -0.1613266 0.1190363 0.05873516
##     Chick.14   Chick.15   Chick.16   Chick.17   Chick.18  Chick.19  Chick.20
## 1 -0.1711044 0.09147065 0.09123366 -0.1736117 0.06428526 0.1167551 -0.171856
## 2 -0.1711044 0.09147065 0.09123366 -0.1736117 0.06428526 0.1167551 -0.171856
## 3 -0.1711044 0.09147065 0.09123366 -0.1736117 0.06428526 0.1167551 -0.171856
## 4 -0.1711044 0.09147065 0.09123366 -0.1736117 0.06428526 0.1167551 -0.171856
## 5 -0.1711044 0.09147065 0.09123366 -0.1736117 0.06428526 0.1167551 -0.171856
## 6 -0.1711044 0.09147065 0.09123366 -0.1736117 0.06428526 0.1167551 -0.171856
##     Chick.21  Chick.22   Chick.23   Chick.24  Chick.25   Chick.26   Chick.27
## 1 0.04110876 0.1352397 -0.1694872 0.02536754 0.1476441 -0.1659863 0.00231206
## 2 0.04110876 0.1352397 -0.1694872 0.02536754 0.1476441 -0.1659863 0.00231206
## 3 0.04110876 0.1352397 -0.1694872 0.02536754 0.1476441 -0.1659863 0.00231206
## 4 0.04110876 0.1352397 -0.1694872 0.02536754 0.1476441 -0.1659863 0.00231206
## 5 0.04110876 0.1352397 -0.1694872 0.02536754 0.1476441 -0.1659863 0.00231206
## 6 0.04110876 0.1352397 -0.1694872 0.02536754 0.1476441 -0.1659863 0.00231206
##    Chick.28   Chick.29    Chick.30  Chick.31   Chick.32   Chick.33   Chick.34
## 1 0.1584437 -0.1524575 -0.01013847 0.1783995 0.09281028 -0.1174624 -0.2991533
## 2 0.1584437 -0.1524575 -0.01013847 0.1783995 0.09281028 -0.1174624 -0.2991533
## 3 0.1584437 -0.1524575 -0.01013847 0.1783995 0.09281028 -0.1174624 -0.2991533
## 4 0.1584437 -0.1524575 -0.01013847 0.1783995 0.09281028 -0.1174624 -0.2991533
## 5 0.1584437 -0.1524575 -0.01013847 0.1783995 0.09281028 -0.1174624 -0.2991533
## 6 0.1584437 -0.1524575 -0.01013847 0.1783995 0.09281028 -0.1174624 -0.2991533
##     Chick.35  Chick.36  Chick.37   Chick.38   Chick.39   Chick.40   Chick.41
## 1 -0.3028137 0.1823152 0.1099617 -0.1309946 -0.1387938 -0.1109887 -0.1470112
## 2 -0.3028137 0.1823152 0.1099617 -0.1309946 -0.1387938 -0.1109887 -0.1470112
## 3 -0.3028137 0.1823152 0.1099617 -0.1309946 -0.1387938 -0.1109887 -0.1470112
## 4 -0.3028137 0.1823152 0.1099617 -0.1309946 -0.1387938 -0.1109887 -0.1470112
## 5 -0.3028137 0.1823152 0.1099617 -0.1309946 -0.1387938 -0.1109887 -0.1470112
## 6 -0.3028137 0.1823152 0.1099617 -0.1309946 -0.1387938 -0.1109887 -0.1470112
##   Chick.42  Chick.43  Chick.44    Chick.45   Chick.46  Chick.47   Chick.48
## 1 0.209595 0.1926583 0.1605267 -0.02731468 0.07594405 0.1901825 -0.1633319
## 2 0.209595 0.1926583 0.1605267 -0.02731468 0.07594405 0.1901825 -0.1633319
## 3 0.209595 0.1926583 0.1605267 -0.02731468 0.07594405 0.1901825 -0.1633319
## 4 0.209595 0.1926583 0.1605267 -0.02731468 0.07594405 0.1901825 -0.1633319
## 5 0.209595 0.1926583 0.1605267 -0.02731468 0.07594405 0.1901825 -0.1633319
## 6 0.209595 0.1926583 0.1605267 -0.02731468 0.07594405 0.1901825 -0.1633319
##   Chick.49 Diet.1 Diet.2 Diet.3 Diet.4
## 1 0.101449      1      0      0      0
## 2 0.101449      1      0      0      0
## 3 0.101449      1      0      0      0
## 4 0.101449      1      0      0      0
## 5 0.101449      1      0      0      0
## 6 0.101449      1      0      0      0
```

```r
#Subset the dummyVars version to only include weight and time (numeric)
  #... and all of the diets, excluding the chick number variables.
head(chick_dum)
```

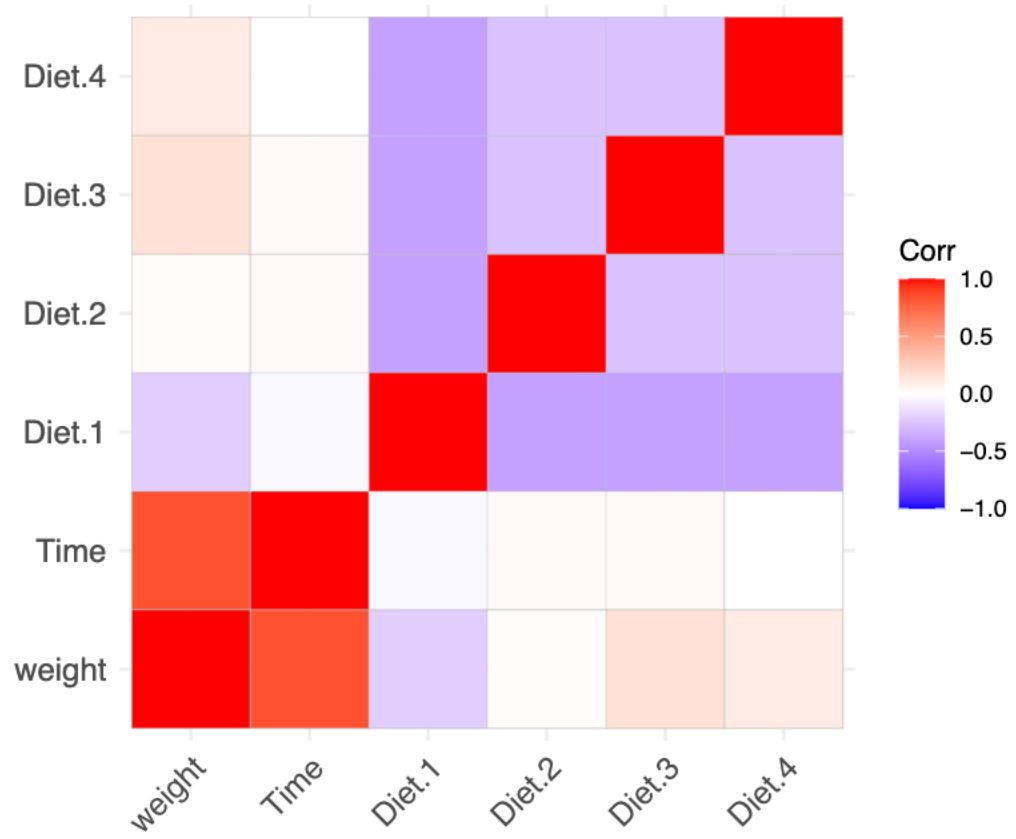
```
##   weight Time    Chick.L     Chick.Q   Chick.C     Chick.4    Chick.5   Chick.6
## 1     42    0 -0.1028992 -0.07445124 0.1666681 -0.06432918 -0.1134107 0.1601967
## 2     51    2 -0.1028992 -0.07445124 0.1666681 -0.06432918 -0.1134107 0.1601967
## 3     59    4 -0.1028992 -0.07445124 0.1666681 -0.06432918 -0.1134107 0.1601967
## 4     64    6 -0.1028992 -0.07445124 0.1666681 -0.06432918 -0.1134107 0.1601967
## 5     76    8 -0.1028992 -0.07445124 0.1666681 -0.06432918 -0.1134107 0.1601967
## 6     93   10 -0.1028992 -0.07445124 0.1666681 -0.06432918 -0.1134107 0.1601967
##       Chick.7    Chick.8  Chick.9   Chick.10   Chick.11  Chick.12   Chick.13
## 1 -0.02173999 -0.1423219 0.143173 0.02033441 -0.1613266 0.1190363 0.05873516
## 2 -0.02173999 -0.1423219 0.143173 0.02033441 -0.1613266 0.1190363 0.05873516
## 3 -0.02173999 -0.1423219 0.143173 0.02033441 -0.1613266 0.1190363 0.05873516
## 4 -0.02173999 -0.1423219 0.143173 0.02033441 -0.1613266 0.1190363 0.05873516
## 5 -0.02173999 -0.1423219 0.143173 0.02033441 -0.1613266 0.1190363 0.05873516
## 6 -0.02173999 -0.1423219 0.143173 0.02033441 -0.1613266 0.1190363 0.05873516
##     Chick.14   Chick.15   Chick.16   Chick.17   Chick.18  Chick.19  Chick.20
## 1 -0.1711044 0.09147065 0.09123366 -0.1736117 0.06428526 0.1167551 -0.171856
## 2 -0.1711044 0.09147065 0.09123366 -0.1736117 0.06428526 0.1167551 -0.171856
## 3 -0.1711044 0.09147065 0.09123366 -0.1736117 0.06428526 0.1167551 -0.171856
## 4 -0.1711044 0.09147065 0.09123366 -0.1736117 0.06428526 0.1167551 -0.171856
## 5 -0.1711044 0.09147065 0.09123366 -0.1736117 0.06428526 0.1167551 -0.171856
## 6 -0.1711044 0.09147065 0.09123366 -0.1736117 0.06428526 0.1167551 -0.171856
##     Chick.21  Chick.22   Chick.23   Chick.24  Chick.25   Chick.26   Chick.27
## 1 0.04110876 0.1352397 -0.1694872 0.02536754 0.1476441 -0.1659863 0.00231206
## 2 0.04110876 0.1352397 -0.1694872 0.02536754 0.1476441 -0.1659863 0.00231206
## 3 0.04110876 0.1352397 -0.1694872 0.02536754 0.1476441 -0.1659863 0.00231206
## 4 0.04110876 0.1352397 -0.1694872 0.02536754 0.1476441 -0.1659863 0.00231206
## 5 0.04110876 0.1352397 -0.1694872 0.02536754 0.1476441 -0.1659863 0.00231206
## 6 0.04110876 0.1352397 -0.1694872 0.02536754 0.1476441 -0.1659863 0.00231206
##    Chick.28   Chick.29    Chick.30  Chick.31   Chick.32   Chick.33   Chick.34
## 1 0.1584437 -0.1524575 -0.01013847 0.1783995 0.09281028 -0.1174624 -0.2991533
## 2 0.1584437 -0.1524575 -0.01013847 0.1783995 0.09281028 -0.1174624 -0.2991533
## 3 0.1584437 -0.1524575 -0.01013847 0.1783995 0.09281028 -0.1174624 -0.2991533
## 4 0.1584437 -0.1524575 -0.01013847 0.1783995 0.09281028 -0.1174624 -0.2991533
## 5 0.1584437 -0.1524575 -0.01013847 0.1783995 0.09281028 -0.1174624 -0.2991533
## 6 0.1584437 -0.1524575 -0.01013847 0.1783995 0.09281028 -0.1174624 -0.2991533
##     Chick.35  Chick.36  Chick.37   Chick.38   Chick.39   Chick.40   Chick.41
## 1 -0.3028137 0.1823152 0.1099617 -0.1309946 -0.1387938 -0.1109887 -0.1470112
## 2 -0.3028137 0.1823152 0.1099617 -0.1309946 -0.1387938 -0.1109887 -0.1470112
## 3 -0.3028137 0.1823152 0.1099617 -0.1309946 -0.1387938 -0.1109887 -0.1470112
## 4 -0.3028137 0.1823152 0.1099617 -0.1309946 -0.1387938 -0.1109887 -0.1470112
## 5 -0.3028137 0.1823152 0.1099617 -0.1309946 -0.1387938 -0.1109887 -0.1470112
## 6 -0.3028137 0.1823152 0.1099617 -0.1309946 -0.1387938 -0.1109887 -0.1470112
##   Chick.42  Chick.43  Chick.44    Chick.45   Chick.46  Chick.47   Chick.48
## 1 0.209595 0.1926583 0.1605267 -0.02731468 0.07594405 0.1901825 -0.1633319
## 2 0.209595 0.1926583 0.1605267 -0.02731468 0.07594405 0.1901825 -0.1633319
## 3 0.209595 0.1926583 0.1605267 -0.02731468 0.07594405 0.1901825 -0.1633319
## 4 0.209595 0.1926583 0.1605267 -0.02731468 0.07594405 0.1901825 -0.1633319
## 5 0.209595 0.1926583 0.1605267 -0.02731468 0.07594405 0.1901825 -0.1633319
## 6 0.209595 0.1926583 0.1605267 -0.02731468 0.07594405 0.1901825 -0.1633319
```

```
##   Chick.49 Diet.1 Diet.2 Diet.3 Diet.4
## 1 0.101449      1      0      0      0
## 2 0.101449      1      0      0      0
## 3 0.101449      1      0      0      0
## 4 0.101449      1      0      0      0
## 5 0.101449      1      0      0      0
## 6 0.101449      1      0      0      0
```

```
chick_dum_chickless<- chick_dum[, !grepl("Chick", colnames(chick_dum))]
head(chick_dum_chickless)
```

```
##   weight Time Diet.1 Diet.2 Diet.3 Diet.4
## 1     42    0      1      0      0      0
## 2     51    2      1      0      0      0
## 3     59    4      1      0      0      0
## 4     64    6      1      0      0      0
## 5     76    8      1      0      0      0
## 6     93   10      1      0      0      0
```

```
#Use ggcorplot and the cor function to create heatmap
require(ggcorrplot)
ggcorrplot(cor(chick_dum_chickless))
```



```r
#Weight appears to be most correlated with time.
```

4. Calculate a slope coefficient for each diet and time combination using a custom function. HINT: use the lm() function inside calculate slope and add 0+ in the front of the independent variables to get slopes for all but no intercepts, otherwise the lm function will drop one of the diet columns to avoid collinearity. (1pts)

```r
calculate_slope <- function(data) {
  #Create empty data frame to hold coefficients.
  #Create empty data frame (from matrix) with column names corresponding to the
    #... overall slope names/types of slopes (Time*Diet emcompasses all of them)
    #... that will hold the calculated slopes
  name_sum<- lm(weight~ 0+Time*Diet, data = ChickWeight)
  name_vec<- names(name_sum$coefficients)
  slope_mat<- matrix(NA, 5, length(name_vec)+1)
  slope_mat[1,1]<- "Variables"
  for (i in 1:length(name_vec)) {
    slope_mat[1,i+1]<- name_vec[i]
    }
  colnames(slope_mat)<-slope_mat[1, ]
  slope_mat<- slope_mat[-1, ]
  slope_df<- as.data.frame(slope_mat)

  #Add the desired variable combinations in the "Variable" collumn of the df
  slope_df$Variables[1]<- colnames(data)[2]
  slope_df$Variables[2]<- colnames(data)[4]
  slope_df$Variables[3]<- paste(colnames(data)[2], colnames(data)[4], sep= '+')
  slope_df$Variables[4]<- paste(colnames(data)[2], colnames(data)[4], sep= '*')

  #Calculate the lm models and slopes using a loop
  for (i in 1:nrow(slope_df)){
    #Combine desired variables with weight to form weight~0+variables
    form<-
      paste(colnames(data)[1], paste(0, slope_df$Variables[i], sep='+'),
            sep= '~')
    model<- lm(as.formula(form), data= data)
    slp<- names(model$coefficients)
    #Populate the slopes in the data frame using another loop
    for (j in 1:length(slp)){
      coef_name<- slp[j]
      slope_df[i, colnames(slope_df) %in% coef_name]<- model$coefficients[j]
    }
  }

  #Return slope data frame
  return(slope_df)
}

calculate_slope(ChickWeight)
```

```
##   Variables             Time            Diet1            Diet2            Diet3
## 1      Time 10.6375966276023             <NA>             <NA>             <NA>
```

```
## 2      Diet             <NA> 102.645454545455 122.616666666667          142.95
## 3 Time+Diet 8.75049174223902 10.9243911018037  27.090465147224 47.4237984805574
## 4 Time*Diet 6.84179719838213 30.9309802751399 28.6335955226138 18.2503252155499
##              Diet4       Time:Diet2       Time:Diet3       Time:Diet4
## 1             <NA>             <NA>             <NA>             <NA>
## 2 135.262711864407             <NA>             <NA>             <NA>
## 3 41.1578472804973             <NA>             <NA>             <NA>
## 4 30.7921195068003 1.76733908962271 4.58107377423925 2.87256836363328
```

5. Write a function that calculates residual sum of squares (RSS), and then compare the minimal value to find the optimal slopes for each parameter and parameter combination. (1pts)

```
#Create a function that calculates rss (from model predictions vs data)
calculate_rss <- function(model, data) {
  pred<- predict(model, data)
  residual<- pred-data$weight
  res_sq<- residual^2
  rss<- sum(res_sq)
  return(rss)
}

#Use the function to get the RSS of all four models (from question 4)
time_rss<- calculate_rss(lm(weight~ 0+Time, data= ChickWeight), ChickWeight)
print(time_rss)
```

```
## [1] 996120.2
```

```
diet_rss<- calculate_rss(lm(weight~ 0+Diet, data= ChickWeight), ChickWeight)
print(diet_rss)
```

```
## [1] 2758693
```

```
time_plus_diet_rss<- calculate_rss(lm(weight~ 0+Time+Diet, data= ChickWeight),
                                   ChickWeight)
print(time_plus_diet_rss)
```

```
## [1] 742336.1
```

```
time_astr_diet_rss<- calculate_rss(lm(weight~ 0+Time*Diet, data= ChickWeight),
                                   ChickWeight)
print(time_astr_diet_rss)
```

```
## [1] 661532
```

```
#Find minimum (from vector of the RSSes)
rss_vec<- c(time_rss, diet_rss, time_plus_diet_rss, time_astr_diet_rss)
rss_min<- min(rss_vec)
rss_vec==rss_min
```

```
## [1] FALSE FALSE FALSE  TRUE
```

```
time_astr_diet_rss==rss_min
```

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

```
#The smallest RSS belongs to Time*Diet, therefore, the model containing
  #... weight~0+Time*Diet is the most accurate/best fits the data.
```

You can loop through different models and calculate RSS for each

to find the optimal slopes for each parameter and parameter combination.

You can work through it manually as well

```
#Calculating RSSes of part 4 modles
time_man_rss<- deviance(lm(weight~ 0+Time, data= ChickWeight))
print(time_man_rss)
```

```
## [1] 996120.2
```

```
diet_man_rss<- deviance(lm(weight~ 0+Diet, data= ChickWeight))
print(diet_man_rss)
```

```
## [1] 2758693
```

```
time_plus_diet_man_rss<- deviance(lm(weight~ 0+Time+Diet, data= ChickWeight))
print(time_plus_diet_rss)
```

```
## [1] 742336.1
```

```
time_astr_diet_man_rss<- deviance(lm(weight~ 0+Time*Diet, data= ChickWeight))
print(time_astr_diet_man_rss)
```

```
## [1] 661532
```

```
#Find the minimum RSS
min_man<-
  min(time_man_rss, diet_man_rss, time_plus_diet_rss, time_astr_diet_man_rss)
time_astr_diet_man_rss==min_man
```

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

```
#The Time*Diet model still has the lowest RSS, and is therefore the best model.
```

6. Use anova to compare the RSS to see if they're significant – compare the F statistic. Use the built in anova function. (0.5pts)

```
#Name time, diet, time+diet, and time*diet modles (to place in anova function)
time<- lm(weight~ 0+Time, data= ChickWeight)
diet<- lm(weight~ 0+Diet, data= ChickWeight)
time_plus_diet<- lm(weight~ 0+Time+Diet, data= ChickWeight)
time_astr_diet<- lm(weight~ 0+Time*Diet, data= ChickWeight)

#Plug models into anova
anova(time, diet, time_plus_diet, time_astr_diet)
```

```
## Analysis of Variance Table
##
## Model 1: weight ~ 0 + Time
## Model 2: weight ~ 0 + Diet
## Model 3: weight ~ 0 + Time + Diet
## Model 4: weight ~ 0 + Time * Diet
##   Res.Df     RSS Df Sum of Sq        F    Pr(>F)
## 1    577  996120
## 2    574 2758693  3  -1762573
## 3    573  742336  1   2016357 1737.367 < 2.2e-16 ***
## 4    570  661532  3     80804   23.208 3.474e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
#There is no F statistic for the simple Time and Diet models.
#The large F statistic and small p-value for the Time+Diet model means that the
  #... decrease in RSS (from the Diet model) is significant
#The large F statistic and small p-value for the Time*Diet model means that the
  #... decrease in RSS (from the Time+Diet model) is significant.
```

7. Fit a linear model to assess the effect of diet and time on weight. Use backwards selection to find the best model just against the p-values of the coefficients. Use the same approach with the 0 + leading the independent variables to ensure all lines are present. (1pts)

```
#Large, overall model using the dummy vars version of Chick Weight to
  #... individually account for all possable variables
mod_1<-
  lm(weight~ 0+ Time+Diet.1+Diet.2+Diet.3+Diet.4+Time:Diet.1+Time:Diet.2+
       Time:Diet.3+Time:Diet.4, data= chick_dum_chickless)
anova(mod_1)
```

```
## Analysis of Variance Table
##
## Response: weight
##             Df   Sum Sq  Mean Sq   F value    Pr(>F)
## Time         1 10495787 10495787 9043.5507 < 2.2e-16 ***
## Diet.1       1    23335    23335   20.1062 8.861e-06 ***
```

```
## Diet.2        1     1591     1591    1.3704  0.242224
## Diet.3        1    97202    97202   83.7524 < 2.2e-16 ***
## Diet.4        1   131657   131657  113.4405 < 2.2e-16 ***
## Time:Diet.1   1    58669    58669   50.5516 3.490e-12 ***
## Time:Diet.2   1    14233    14233   12.2639  0.000498 ***
## Time:Diet.3   1     7902     7902    6.8083  0.009312 **
## Residuals   570   661532     1161
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
#Diet.2 has the largest P-value and the only one larger than .05

#Same model as above, but without Diet.2
mod_2<- lm(weight~ 0+ Time+Diet.1+Diet.3+Diet.4+Time:Diet.1+Time:Diet.2+
             Time:Diet.3+Time:Diet.4, data= chick_dum_chickless)
anova(mod_2)
```

```
## Analysis of Variance Table
##
## Response: weight
##              Df   Sum Sq  Mean Sq   F value    Pr(>F)
## Time          1 10495787 10495787 8699.5987 < 2.2e-16 ***
## Diet.1        1    23335    23335   19.3415 1.304e-05 ***
## Diet.3        1    84245    84245   69.8278 4.940e-16 ***
## Diet.4        1    89132    89132   73.8781 < 2.2e-16 ***
## Time:Diet.1   1   102568   102568   85.0153 < 2.2e-16 ***
## Time:Diet.2   1       46       46    0.0379   0.84579
## Time:Diet.3   1     7902     7902    6.5493   0.01075 *
## Residuals   571   688893     1206
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
#Time:Diet.2 has the largest P-value

#Same model as above, but without Time:Diet.2
mod_3<- lm(weight~ 0+ Time+Diet.1+Diet.3+Diet.4+Time:Diet.1+Time:Diet.3+
             Time:Diet.4, data= chick_dum_chickless)
anova(mod_3)
```

```
## Analysis of Variance Table
##
## Response: weight
##              Df   Sum Sq  Mean Sq   F value    Pr(>F)
## Time          1 10495787 10495787 8699.5987 < 2.2e-16 ***
## Diet.1        1    23335    23335   19.3415 1.304e-05 ***
## Diet.3        1    84245    84245   69.8278 4.940e-16 ***
## Diet.4        1    89132    89132   73.8781 < 2.2e-16 ***
## Time:Diet.1   1   102568   102568   85.0153 < 2.2e-16 ***
## Time:Diet.3   1     5341     5341    4.4267   0.03582 *
## Time:Diet.4   1     2607     2607    2.1605   0.14215
## Residuals   571   688893     1206
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
#Time:Diet.4 has the largest P-value

#Same model as above, but without Time:Diet.4
mod_4<- lm(weight~ 0+ Time+Diet.1+Diet.3+Diet.4+Time:Diet.1+Time:Diet.3,
           data= chick_dum_chickless)
anova(mod_4)
```

```
## Analysis of Variance Table
##
## Response: weight
##              Df   Sum Sq  Mean Sq   F value    Pr(>F)
## Time          1 10495787 10495787 8681.9848 < 2.2e-16 ***
## Diet.1        1    23335    23335   19.3024 1.330e-05 ***
## Diet.3        1    84245    84245   69.6864 5.249e-16 ***
## Diet.4        1    89132    89132   73.7285 < 2.2e-16 ***
## Time:Diet.1   1   102568   102568   84.8432 < 2.2e-16 ***
## Time:Diet.3   1     5341     5341    4.4178     0.036 *
## Residuals   572   691500     1209
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
#Time:Diet.3 has the largest P-value and the only one above .05

#Same model as above, but without Time:Diet.3
mod_5<- lm(weight~ 0+ Time+Diet.1+Diet.3+Diet.4+Time:Diet.1,
           data= chick_dum_chickless)
anova(mod_5)
```

```
## Analysis of Variance Table
##
## Response: weight
##              Df   Sum Sq  Mean Sq  F value    Pr(>F)
## Time          1 10495787 10495787 8630.506 < 2.2e-16 ***
## Diet.1        1    23335    23335   19.188 1.409e-05 ***
## Diet.3        1    84245    84245   69.273 6.307e-16 ***
## Diet.4        1    89132    89132   73.291 < 2.2e-16 ***
## Time:Diet.1   1   102568   102568   84.340 < 2.2e-16 ***
## Residuals   573   696840     1216
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
#All p-values (of coefficients) are significant, so backwards selection stops

backwards_mod<- mod_5
print(backwards_mod)
```

```
##
## Call:
## lm(formula = weight ~ 0 + Time + Diet.1 + Diet.3 + Diet.4 + Time:Diet.1,
##     data = chick_dum_chickless)
##
## Coefficients:
```

```
##        Time       Diet.1       Diet.3       Diet.4  Time:Diet.1
##      10.531       30.931       27.985       22.008       -3.689
```

8. Iteratively enhance with backwards selection. When the F statistic becomes insignificant, stop. Do not use the step function, implement your own F test based backwards selection. (1pts)

```
#Create function
f_test_backward_selection <- function(full_model) {
  #To first enter the "when", m has to be greater than .05, so I am starting
    #... with a place-holder
  m=1
  model<- full_model
  while (m>.05) {
    #Find largest variable with p>.05 (if no p>.05, stopping)
    an<- anova(model)
    m<- max(an$`Pr(>F)`, na.rm = T)
    #Remove that variable
    an[an$`Pr(>F)`==m ,]
    del_var<- row.names(an[an$`Pr(>F)`==m ,])[1]
    fix<- paste(". ~ . -", del_var)
    new<- update(model, as.formula(fix))
    #Continue in this "while" until each variable with a p-stat>.05 is
      #... iteratively removed. This coincides with the F statistic becoming
      #... insignificant.
    model<- new
  }
  #Return finalized model formed through backwards selection
  return(model)
}


#The full model used in part 7 is mod_1, so I am using it to test my function
back_result<- f_test_backward_selection(mod_1)
print(back_result)
```

```
##
## Call:
## lm(formula = weight ~ Time + Diet.1 + Diet.3 + Diet.4 + Time:Diet.1 -
##     1, data = chick_dum_chickless)
##
## Coefficients:
##        Time       Diet.1       Diet.3       Diet.4  Time:Diet.1
##      10.531       30.931       27.985       22.008       -3.689
```

```
names(back_result$coefficients)==names(backwards_mod$coefficients)
```

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

```
back_result$coefficients==backwards_mod$coefficients
```

```
##        Time      Diet.1      Diet.3      Diet.4 Time:Diet.1
##        TRUE        TRUE        TRUE        TRUE        TRUE
```

```
#The coefficient names and slopes of the resulting backwards selection models
  #... from part 7 and 8 are the same. I am taking this to mean that both are
  #... (hopefully) correct.
```

9. Create a quadratic line with just weight vs time (quadratic vs linear) – calculate RSS with results from quadratic to see if it's better. (0.5pts)

```
#Create quadratic with only Time predictor
time_quad<- lm(weight~ Time+ Time^2, data= ChickWeight)

#Calculate RSS with function made in part 5
t_quad_rss<- calculate_rss(time_quad, ChickWeight)
print(t_quad_rss)
```

```
## [1] 872212.2
```

```
#Compare RSS to linear model with just time
t_quad_rss<time_rss
```

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

```
#The quadratic RSS is lower, so it is a better model than the linear time model.

#Compare RSS to Time:Diet model (because it had the lowest linear model RSS)
t_quad_rss<time_astr_diet_rss
```

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

```
#The quadratic RSS is not lower, so it is a worse model than the Time:Diet model.
```

10. Generate a null model of chick-weight to hypothetically use for forwards selection. (0.5pts)

```
#The null model is that the slopes for the variables are 0.
#This can be represented by an lm that is all intercept and no variable.
#This is demonstrated below.
#As a control, I am keeping the 0+ phrasing at the beginning.
null_model<- lm(weight~0+1, data=ChickWeight)
null_model
```

```
##
## Call:
## lm(formula = weight ~ 0 + 1, data = ChickWeight)
##
## Coefficients:
## (Intercept)
##       121.8
```

Bonus: Show that slope of linear regression is pearsons correlation r times the ratio of standard deviations for a simple linear model. (1pts)

You can show it analytically by mashing together the equations for ß_1 and r, or you can show it numerically by simulating some data and fitting a linear model and calculating the correlation coefficient and standard deviations.

The equation for sd(X) in terms of sum of squares is:

$$s_x = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2}$$

where X is the predictor variable, n is the number of observations, and $\bar{X}$ is the mean of X.

The equation for sd(Y) in terms of sum of squares is:

$$s_y = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(Y_i - \bar{Y})^2}$$

where Y is the response variable, n is the number of observations, and $\bar{Y}$ is the mean of Y.

The equation for cov(X,Y) is:

$$cov(X,Y) = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})$$

where X is the predictor variable, Y is the response variable, n is the number of observations, and $\bar{X}$ and $\bar{Y}$ are the means of X and Y, respectively.

The equation to calculate pearson's correlation is:

$$r = \frac{cov(X,Y)}{s_x s_y}$$

where $cov(X,Y)$ is the covariance between the predictor variable X and the response variable Y, s_x is the standard deviation of X, and s_y is the standard deviation of Y.

The numerical equation for ß_1 in a simple linear regression is:

$$\beta_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

The symbolic representation of the same equation is:

$$\beta_1 = \frac{cov(X,Y)}{s_x^2}$$

where $cov(X,Y)$ is the covariance between the predictor variable X and the response variable Y, and s_x^2 is the variance of X.

Can you prove the following by apply in algebra with the equations above:

$$\beta_1 = r\frac{s_y}{s_x}$$

where r is Pearson's correlation coefficient, s_y is the standard deviation of the response variable, and s_x is the standard deviation of the predictor variable.

Write the solution by hand and upload a photo of your proof; or, if using a numerical comparison, submit the code here:

```
require(magick)
```

```
## Loading required package: magick
```

```
## Warning: package 'magick' was built under R version 4.3.3
```

```
## Linking to ImageMagick 6.9.12.93
## Enabled features: cairo, fontconfig, freetype, heic, lcms, pango, raw, rsvg, webp
## Disabled features: fftw, ghostscript, x11
```

```
proof<-
  image_read("/Users/marlenakuhn/Desktop/NYU/masters fall/machine learning/Kuhn-H1-Bonus.HEIC")
plot(proof)
```