EXAM - Stastical Modelling and Inference

Student 138490

Setup Code

Just to keep everything inline!

```
library(dplyr)
library(ggplot2)
library(tidyr)
library(broom)

dat.1.raw <- read.csv("data/exam16_data1.txt", sep=" ")
dat.2.raw <- read.csv("data/exam16_data2.txt", sep=" ")
dat.3.raw <- read.csv("data/exam16_data3.txt", sep=" ")
dat.4.raw <- read.csv("data/exam16_data4.txt", sep=" ")
dat.5.raw <- read.csv("data/exam16_data5.txt", sep=" ")
```

1. 300 Genes

1.1 Why their opinions differ

The first analyst is looking at the p-values for these genes and has selected all genes with a p-value greater than 0.05. This analyst is not considering the fact that given 300 genes, we should not be suprised that 300/20 of them have a p-value that is lower than 1/20!! The 1/20 cutoff is suitable for a single test, but we are effectively testing 300 hypotheses at once, and therefore we should seek some other way to control the error.

There are two major ways to do this, one is to control the Family-Wise Error Rate (FWER) via the Boneforroni Correction, the other more conservative and for this situation (an initial screening), arguably more suitable to control the False Discovery Rate via the Benjamini Hochberg procedure. We will take a look at the data to see which the second is using, but it is clear the second is considering the multiple-testing problem.

1.2 Let's go with number 2

An implementation of the BH FDR procedure is given below, as is the results, which show that no gene is likely to be significant! The boneferroni correction, more conservative still, shows the same. It is likely, therefore, that the second analyst is using one of the two procedures, let's give him credit and say he's controlling the FDR, and therefore, that second one is clearly more credible!

```
bhfdr <- function(vec, alpha = .05) {
    sorted <- sort(vec)
    m <- length(vec)
    accepted = c()
    for (i in 1:m){
        if (vec[i] > (i/m) * alpha) {
            break;
        }
        accepted[i] <- vec[i]
    }
    accepted
```

```
}

bhfdr(dat.1.raw$pvalue)
```

```
## NULL
```

2. Combining Noisy Measurements!

2.1

By simply averaging the two observations, we will get an unbiased estimate of the mean with sampling variance exactly equal to the two squared sd's divided by 4. This is easily verified by simulation:

```
a <- rnorm(1000000, mean = 0, sd = 2)
b <- rnorm(1000000, mean = 0, sd = 3)
c <- (a+b)/2
pdf <- density(c)
cdf <- approxfun(pdf$x, pdf$y, yleft = 0, yright=0)
target.sd <- sqrt((2^2+3^2)/4)
integrate(cdf, -target.sd, target.sd)
```

```
## 0.6824806 with absolute error < 6.7e-05
```

2.2

2.2.a

We can take the mean of our observations as a reasonable estimate to begin with:

```
mean(dat.2.raw[, 2])
```

```
## [1] -0.2523333
```

2.2.b

Underlying assumption is that they are unbiased. Because they are unbiased, we assume that the mean of them will similarly be unbiased!

2.2.c

Indeed, under these assumptions, we can say that this substance looks like it reduces mortality rates. See below for a better proof #### 2.2.d Rather than relying on the point estimate, we can combine these distributions, wihch we assume here to be normal, and get the combined variation, which will give us a proper confidence interval for whether or not this has an effect. This is done below, and we can see that we have $>$ 99% confidence that there is indeed a negative correlation, and therefore a reduction in mortality.

```
sim <- function (dat, n) {
    x <- sapply(1:n, function (i) {
        mean(apply(dat, 1, function (r) {
            rnorm(1, r[2], r[3])
```

```
        }))
    })
    x
}

simmed <- sim(dat.2.raw, 1000)
pdf <- density(simmed)
cdf <- approxfun(pdf$x, pdf$y, yleft = 0, yright=0)
integrate(cdf, -Inf, 0)
```

```
## 0.9947186 with absolute error < 0.00011
```

3. Barcelona Tax Evasion

The data is relatively small, making this a difficult problem. We will walk through a couple of naive and quick initial findings, after which we will attempt to look more closely at the reasons for those findings and discover sources of concern.

We start with a simple linear model, we assume all the given features are linearly correlated with price, and we attemt to see if the dummy variable, for the collaborating firm, is significant in determining price when we control for all other price-affecting features we are given.

This linear model makes a lot of assumptions, we use OLS and our fit is based on assuming the error term is gaussian. In truth, we have no reason to believe that house prices should be gaussian afer controlling for these few features, as prices can vary widely. We will return to this concern. Here is a quick first look at a linear model:

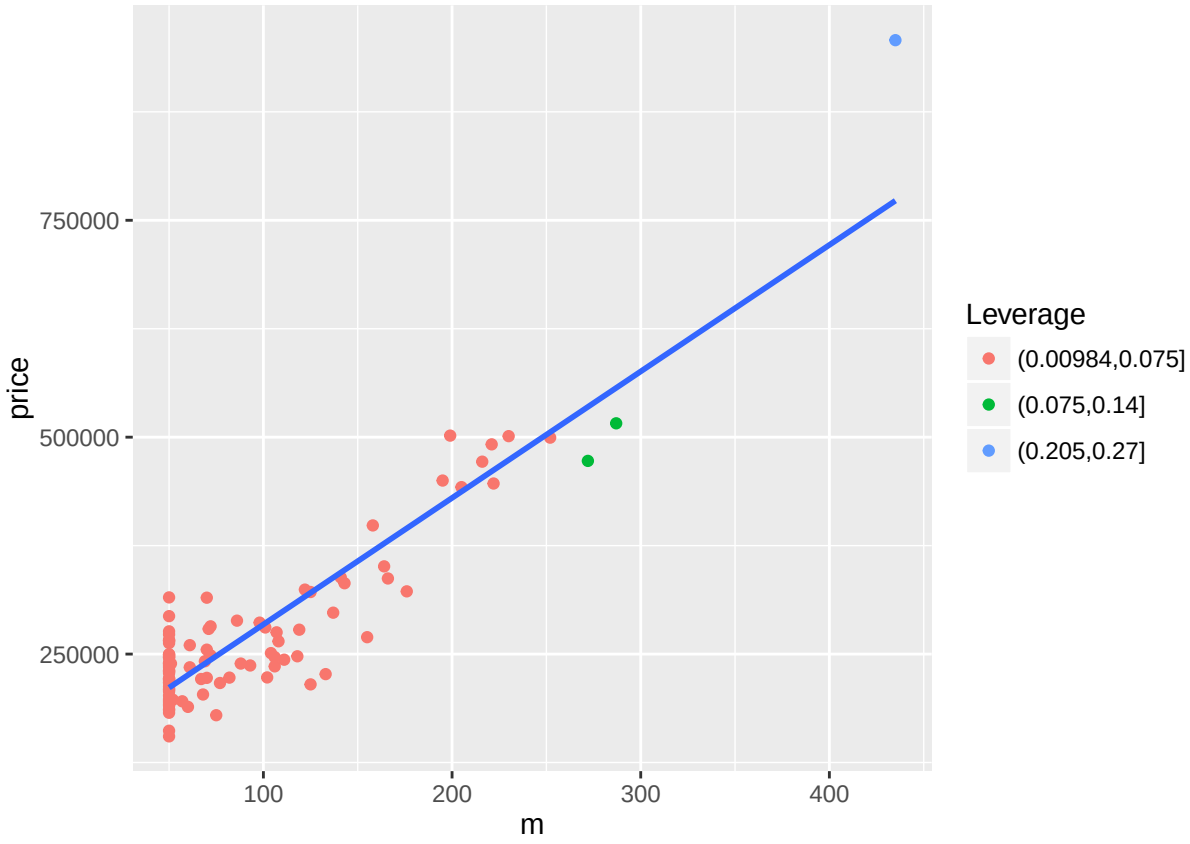
```
colnames(dat.3.raw) <- c("price", "m", "bed", "floor", "collab")
dat.3 <- data.frame(dat.3.raw)

base.model <- lm(price ~ ., data = dat.3)
summary(base.model)
```

```
##
## Call:
## lm(formula = price ~ ., data = dat.3)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -85471 -20972   4990  18675 174527
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 78417.39    9912.77   7.911 4.91e-12 ***
## m            1407.98      61.66  22.835  < 2e-16 ***
## bed         10279.90    4126.82   2.491   0.0145 *
## floor       12773.63    1873.00   6.820 8.71e-10 ***
## collab      39640.27    9086.34   4.363 3.29e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 35240 on 94 degrees of freedom
## Multiple R-squared:  0.8981, Adjusted R-squared:  0.8937
## F-statistic: 207.1 on 4 and 94 DF,  p-value: < 2.2e-16
```

We can see here that we estimate the affect of this dummy variable, the affect of "not cheating", to be highly significant (easily passing even a FWER test). Using this model, we estimate the effect of not cheating to be about 39640, with a 95% confidence interval (obtained through 1.96 times the Standard Error on both sides) to be between about 21831 and 57449. A significant loss in taxes!

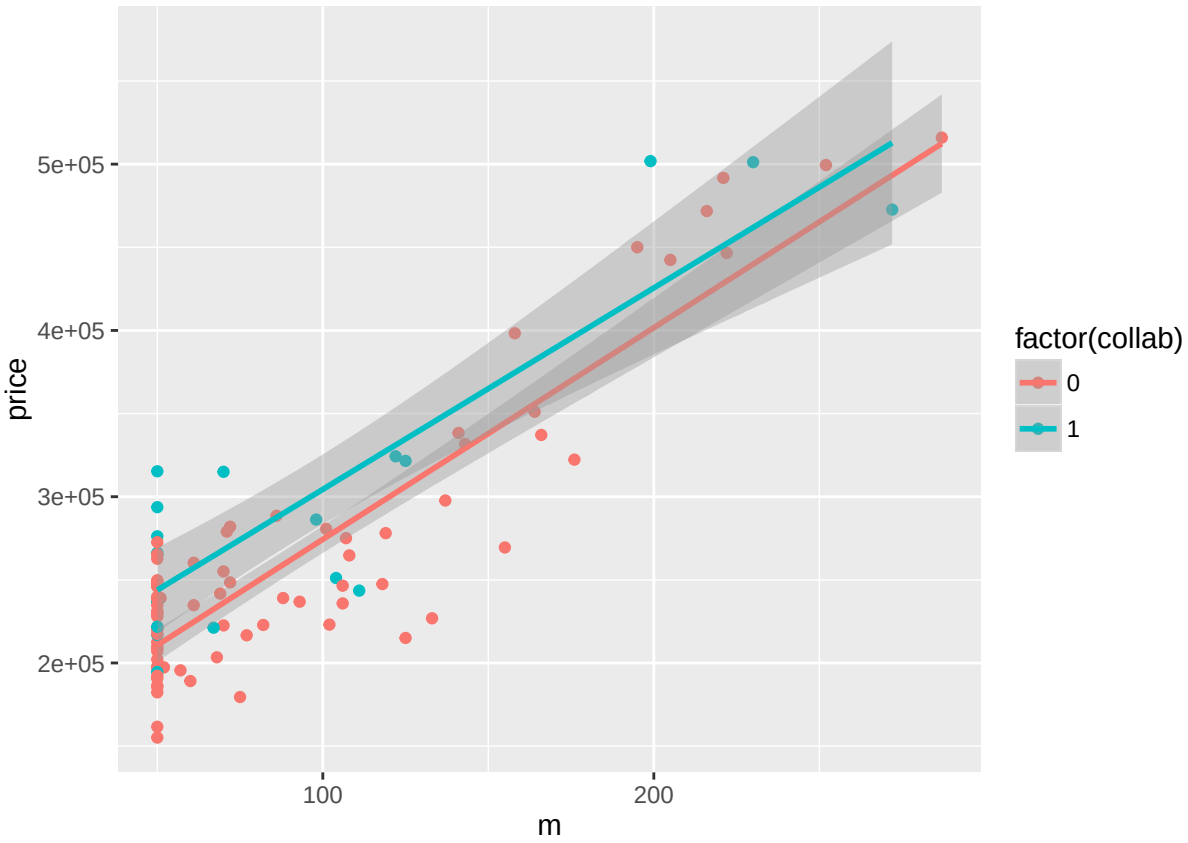
When looking at basic diagnostic plots, we quickly see that there is one outlying point with strong leverage. Plotting the price as a function of square meters, by far the most significant factor, and coloring the leverage, we can easily see the point and the damage it is causing. It is a much larger home than any others, and as such is pulling very strongly on our linear model.

```
dat.3 %>%
    do(fit = lm(.$price ~ .$m)) %>%
    augment(fit) %>%
    rename(price = ..price, m = ..m) %>%
    ungroup() %>%
    mutate(hat.range = cut(.hat, 4)) %>%
    ggplot(aes(y = price, x = m)) +
    geom_point(aes(colour = hat.range)) +
    geom_smooth(method = 'lm', formula = y ~ x, se = FALSE) +
    scale_colour_discrete(name = "Leverage")
```



Removing that outlying data point is a wise move, as we are attempting to make a comparrison between two groups, and the one group has no house that even comes close to comparring. After removal, we can see that a quick linear regression on the square meters shows exactly what we would expect if there was an affect of cheating: the two lines respond to square meterage relatively smoothly, but with a constant difference between the two (the cheating constant!). Plotting with standard errors shaded allows us to see that we have a paucity of data at the upper end of the price/size spectrum, but at the lower end the data is extremely strong.

```
dat.3 %>%
    filter(m < 400) %>%
    ggplot(aes(y = price, x = m, colour = factor(collab))) +
    geom_point() +
    geom_smooth(method = 'lm', formula = y ~ x)
```



It is instructive to attempt our linear model again with the outlying point removed. Here we see that our confidence about the affect of the cheating has gone up, our standard errors around our estimate of the effect has gone down, and we hone in on a better estimate for cheating (maybe).

```
base.model.small <- lm(price ~ ., data = dat.3 %>% filter(m < 400))
summary(base.model.small)
```

```
##
## Call:
## lm(formula = price ~ ., data = dat.3 %>% filter(m < 400))
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -77943 -17817   4905  17598  88170
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 95217.62    8281.34  11.498  < 2e-16 ***
## m            1210.67      56.41  21.463  < 2e-16 ***
## bed         11530.02    3315.57   3.478 0.000771 ***
```

```
## floor       11691.30    1510.12   7.742 1.17e-11 ***
## collab      42753.94    7302.88   5.854 7.14e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 28280 on 93 degrees of freedom
## Multiple R-squared:   0.89,  Adjusted R-squared:  0.8852
## F-statistic:   188 on 4 and 93 DF,  p-value: < 2.2e-16
```

It is worth quickly looking at this data from another angle to validate our findings, at least in our own minds. We see that square meterage has by far the most significant effect, and it's worth looking at the intercept terms as plotted in the previous graph. The idea here is to assume that meterage affects both sets of sales (cheating and not-cheating) equally, then look to see if the intercept is different, and attempt to estimate the difference. Our model is again assuming gaussianity and reporting standard errors, which are simply standard deviations in the gaussian noise, so we will calculate the difference of the two via simulation. This will allow us also to easily extend this beyond gaussian noise and explore other distribution assumptions, if we had the time.

One sees that again we are fairly certain that there is a positive impace of cheating, with this version giving us an estimated mode of 54137, with a larger interval between 17729 and 93829.

```
find.cheaters <- function(n) {

    # wish I had the internet so I could google how to get std error
    # out of this stupid R model object instead hardcoding them...
    sim.cheating <- rnorm(n, mean = 183281, sd = 17706)
    sim.straight <- rnorm(n, mean = 127612, sd = 8248)
    diff <- sim.cheating - sim.straight
    pdf <- density(diff)
    mode <- pdf$x[which.max(pdf$y)]
    cdf <- approxfun(pdf$x, pdf$y, yleft = 0, yright=0)
    interval <- quantile(diff, probs = c(.025, .975))
    list(mode = mode, cdf = cdf, pdf = pdf, interval = interval)
}

cheating <- find.cheaters(10000)
cheating$mode
```

```
## [1] 54153.33
```

```
cheating$interval
```

```
##     2.5%    97.5%
## 17518.16 93716.54
```

We can also see the probability of the true gap being less than or equal to 0:

```
integrate(cheating$cdf, -Inf, 0)
```

```
## 0.00316236 with absolute error < 8.6e-05
```

We can conclude that we have made a lot of assumptions, linearity, gaussianity, cheating having a constant impact. These assumptions are strong, and woudl be worth exploring in other ways, but under these assumptions we find strong reason to believe there is a significant effect of cheating, with a large loss range and reported above!

4. The Smoothie

Basic Strategy

The problem at hand is best subset selection, L0 regression. This is an NP Hard problem, and one simple relaxation is to solve the L1 regression problem, which should point us in the correct direction. We will therefore attempt to use Lasso regression as a means of variable selection, using a stepwise implementation that is designed to give us the most important features. Using lasso and visually inspecting the plots, we will pick a subset of genes, and run a simple logistic regression to try and gain an unregularized coefficient on them, as well as a reasonable p-value, taking into account a False Discovery Rate (FDR) that we want to hold to.

It should be noted, that this is quite rough. From my understanding of genes, they often trigger in groups, and having this knowledge would push us towards an implementation of group lasso or block decomposition that takes advantage of the group structure and correlations among genes. Given more time, we could similarly try and find gene groups ourselves, without prior knowledge, from this data itself, and then proceeed to test all subsets within the groups of genes, solving the best subset problem directly, or at least using group lasso, which would add or remove blocks of genes instead of individuals.

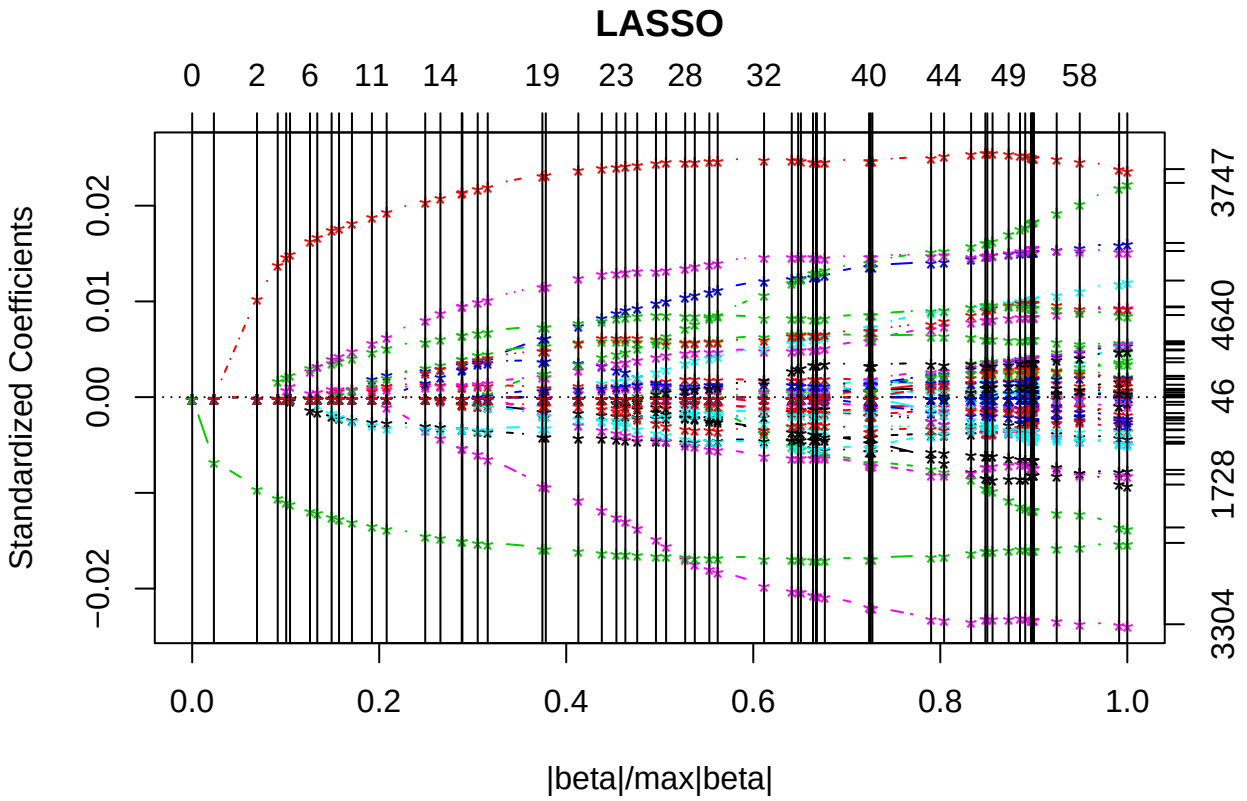
The Sex Question

Not being a biologist, and to simplify this initial analysis, we will ignore sex. Ideally, we would have some expert knowledge as to how much of an affect sex should have (do different genes affect cancer differently across the sexes??). Barring that, we could easily explore this question ourself in the data, given the time.

Modelling

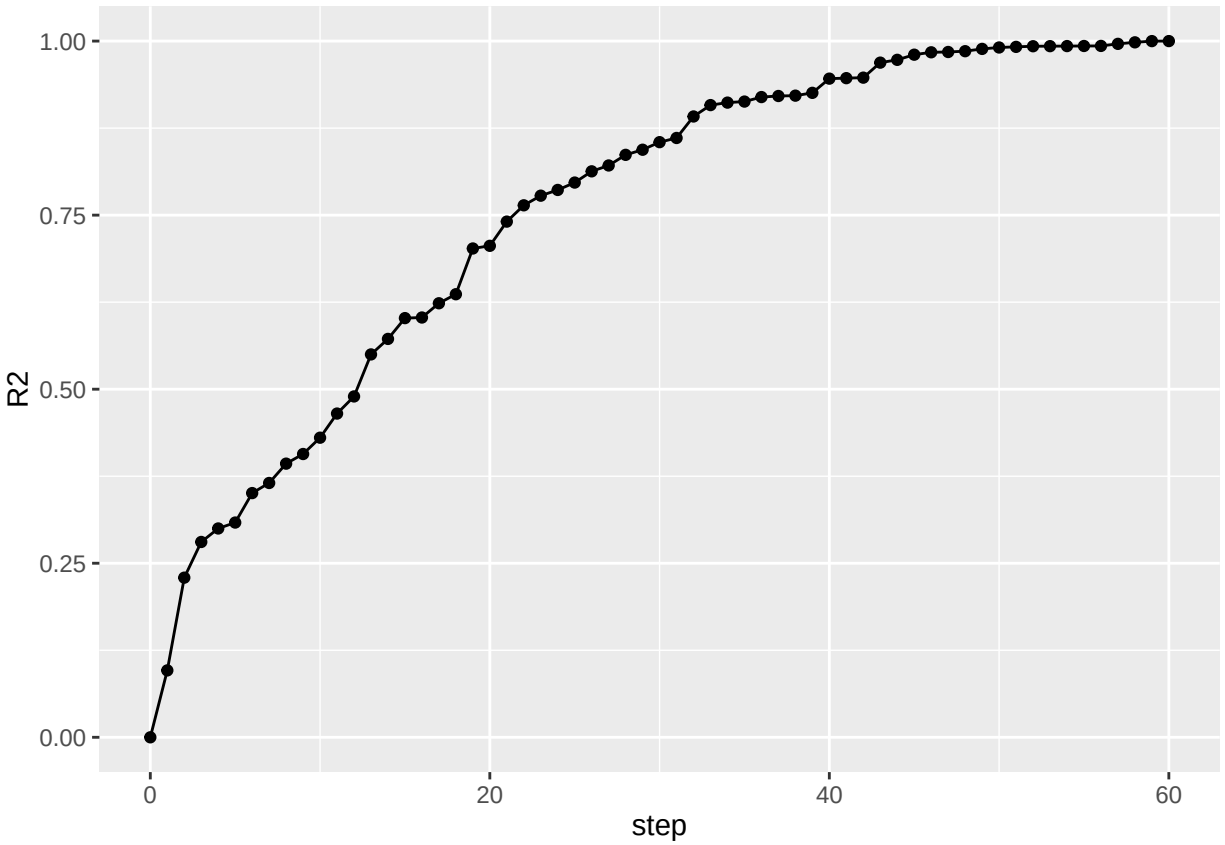
Here we use the lars package and plot the steps of the LASSO regression, which gives us a nice visual as to when each variable becomes effective:

```
library(lars)

y <- dat.4.raw[, 1]
X <- dat.4.raw[, -2]
gene.lasso.model <- lars(as.matrix(X), y, use.Gram=FALSE, normalize=FALSE)
plot(gene.lasso.model)
```



Similarly, we can take a look at a plot of the increasing R squared of each step, which will help us see how much variance is being described and visually inspect for an "elbow", or sweet spot, where we maximize our R squared most cheaply:

```
data.frame(step = 0:60, R2 = gene.lasso.model$R2) %>% ggplot(aes(x = step, y = R2)) + geom_point() + ge
```



We have only 49 observations. This will make it hard to find the linear effect of many genes, and even harder if we would like to transform the features via another kernel that multiplies the number. As mentioned, this is truly a best subset problem, and Lasso has done a nice job of efficiently stepping through the features and giving us the nice visualization above. We can see there is a fairly large elbow around 3 features, after which it's fairly linear until about 20, which is another possible elbow.

At this point we have several options. We simply don't have enough observations to accurately model the exact effect of 20 genes, especially if we want to explore non-linearities in their possible effect on cancer. My recommendation in this situation would most definitely be to flag the top 20 genes given to us by this naive stepwise Lasso exploration, and go from there.

If we did want to move further, one option would be to look at the top 3, which account for an R squared of over 25% on their own, and attempt to run a logistic regression using those 3. We can take a look at the coefficients and compare them to scientific expertise, but we can also attempt to use this extremely sparse model via cross-validation to see if it is actually an effective linear classifier. We begin with a logistic regression model trained on the entire data set, and we see that in the 3 genes we picked,

```
log.regression <- function (y, X, n) {
# we can see that no element was dropped in the first couple dozen steps
# so we only have additions, which makes it easy to filter:
flipped.features <- data.frame(t(X), stringsAsFactors=FALSE)
filtered <- flipped.features %>%
    mutate(
        entry = gene.lasso.model$entry,
        gene = rownames(.)
    ) %>%
    filter(entry <= n) %>%
    filter(entry > 0) %>%
```

```
    mutate(entry = NULL)

# make crazy rerighted dataframe with filtered cols
genes <- filtered$gene
rerighted <- filtered %>%
    mutate(gene = NULL) %>%
    t() %>%
    data.frame(stringsAsFactors=FALSE)

colnames(rerighted) <- genes
rerighted <- rerighted[-50, ]
rownames(rerighted) <- NULL

# regress!
sparse.genes <- rerighted %>% mutate(y = y)
lm.model <- glm(y ~ . - 1, family=binomial(link = "logit"), data = sparse.genes)
list(sparse.dat = sparse.genes, model = lm.model)
}

lm.model <- log.regression(y, dat.4.raw[, -2], 3)$model
summary(lm.model)
```

```
##
## Call:
## glm(formula = y ~ . - 1, family = binomial(link = "logit"), data = sparse.genes)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.2165  -0.0514   0.2763   0.7706   3.4313
##
## Coefficients:
##                    Estimate Std. Error z value Pr(>|z|)
## X.1.02403476872711  0.43393    0.15027   2.888  0.00388 **
## X1.36435131522174   0.11422    0.05934   1.925  0.05424 .
## X4.55610763184199  -0.20214    0.06861  -2.946  0.00322 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 67.928  on 49  degrees of freedom
## Residual deviance: 32.529  on 46  degrees of freedom
## AIC: 38.529
##
## Number of Fisher Scoring iterations: 6
```

We can also try leave-one-out cross validation to see how a linear predictor with the weights determined by logistic regression and the above three genes performs. We can see that it succeeds in classifiying cancer in 45/50 cases, or 90%, compared to a base rate in our data of 72% (36 cases out of 50 observations). Given time, we could use cross-validation to check every subset of the top 20 genes to see which performs best, as well as checking our asssumptions of linearity in the model and using the exponential subsets of the top 20 genes in other classifiers.

```
leave.one.out <- function(dat) {
    rows <- dim(dat)[1]
    success <- c()
    for (i in 1:rows) {
        model <- glm(y ~ . - 1, family=binomial(link = "logit"), data = dat[-i, ])
        if (predict(lm.model, newdata = dat[i, ]) > 0) {
            prediction <- 1
        } else {
            prediction <- 0
        }
        if (prediction == dat[i, "y"]){
            success[i] <- TRUE
        } else {
            success[i] <- FALSE
        }
    }
    success
}


sparse <- log.regression(y, dat.4.raw[, -2], 3)$sparse.dat
sum(leave.one.out(sparse))
```

```
## [1] 45
```

5. The Tough Cookie

It is extremely natural to model the number of seats as a Poisson random variable, as it is a number of counts, so we will start with that as an initial model. Because our counts are truncated, we will use a truncated poisson variable, the truncation point being different for every observation, based on the flight.

Stan gives us an extremely intuitive way to express this situation, as well as a way to simulate this truncated posterior, so we will use that in order to model our data and learn the poisson distributed data as a linear function of the price. With further time it would be worth considering that behavior is NOT linear in price, and including bases functions that transform our price, using local or global kernels, and exploring those models.

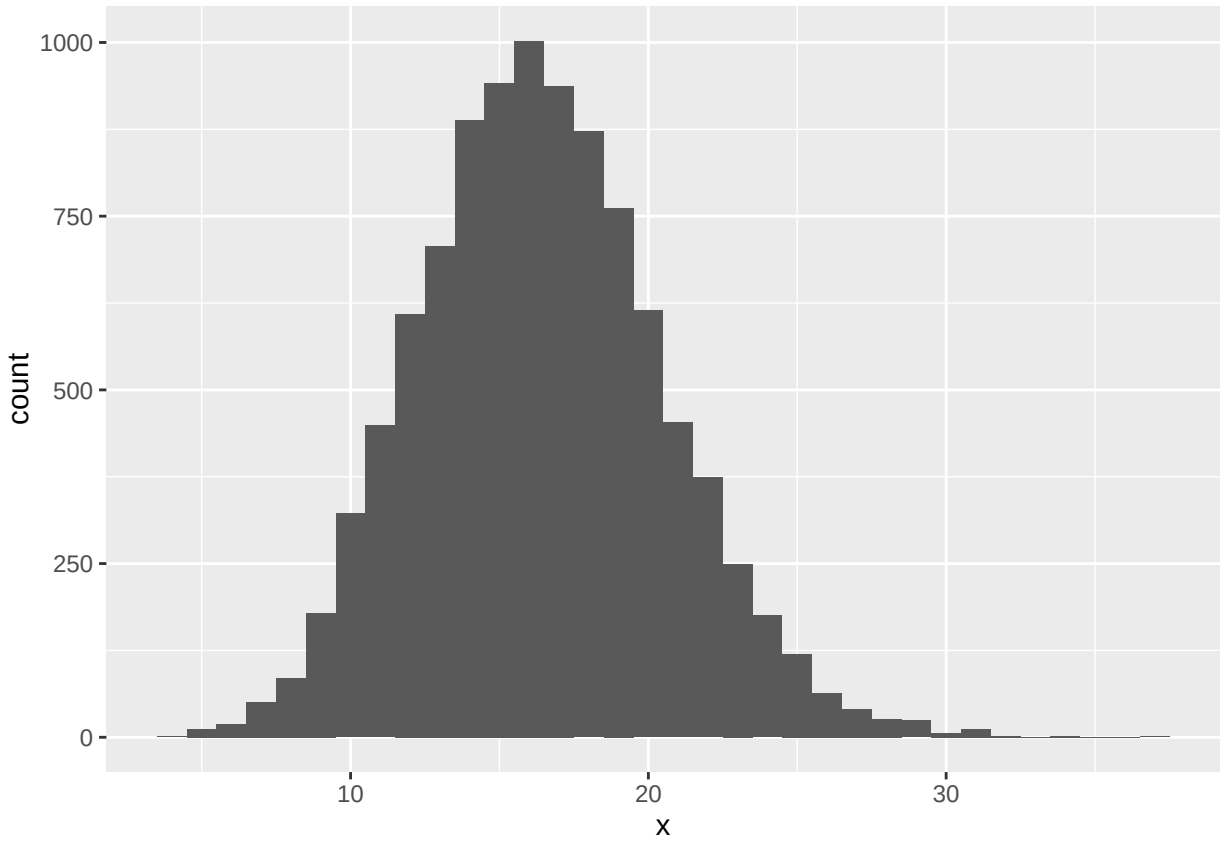
```
library(rstan)
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())

colnames(dat.5.raw) <- c("price", "available", "purchased")
dat.5 <- dat.5.raw

airlines <- list(
    N = dim(dat.5)[1],
    price = dat.5$price,
    available = dat.5$available,
    purchased = dat.5$purchased
)

# model_airlines <- stan("airlines.stan", data = airlines)
```

```
model_airlines
```

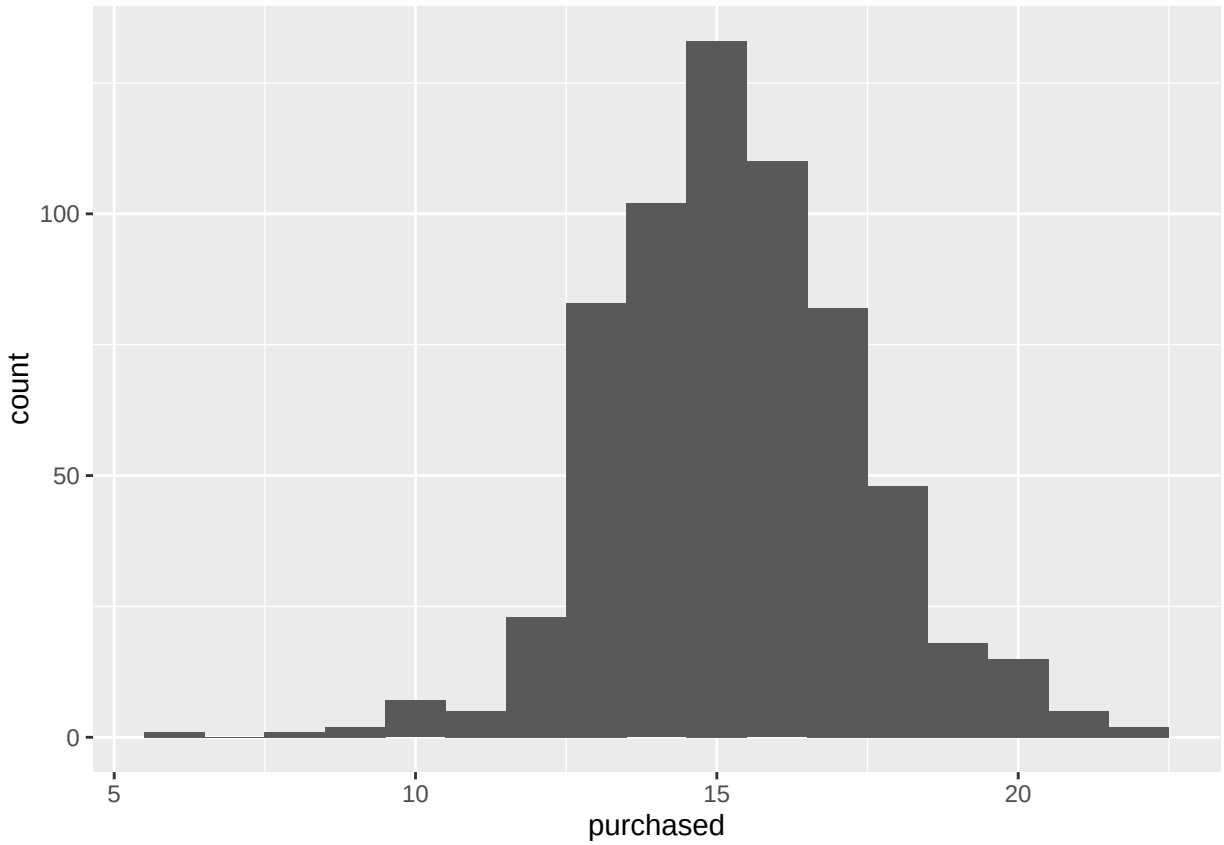
```
## Inference for Stan model: airlines.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##           mean se_mean   sd      2.5%       25%       50%       75%
## x        -8.77    0.01 0.19     -9.14     -8.89     -8.77     -8.64
## b       139.20    0.13 2.66    133.89    137.45    139.21    140.93
## lp__ 227547.74    0.04 1.01 227545.06 227547.39 227548.05 227548.44
##          97.5% n_eff Rhat
## x        -8.39   407 1.01
## b       144.36   408 1.01
## lp__ 227548.70   546 1.01
##
## Samples were drawn using NUTS(diag_e) at Mon Dec 19 15:13:47 2016.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

As we can see, we have a fairly tight distribution that tells us to expect the seats to be modelled as a poisson with lambda equal to 139.2 -8.77*price. Again, we can easily improve this model to consider the price-elasticities as non-linear, given time. As a quick sanity check, we can plot a simulation of our poisson distribution for a price set to 14, and compare that to our empirical data at that point:

```
#simulate and plot
data.frame(x = rpois(10000, 14*-8.77 + 139.20)) %>% ggplot(aes(x)) + geom_histogram(binwidth=1)
```




```
# plot our data at this price point
dat.5 %>% filter(price == 14) %>% ggplot(aes(purchased)) + geom_histogram(binwidth=1)
```



And to finish, here is the Stan code for this model:

```
data {
  int<lower=0> N;
  real<lower=0> price[N];
  int<lower=0> available[N];
  int<upper=max(available)> purchased[N];
}
parameters {
  real x;
  real b;
}
model {
  for (n in 1:N)
    purchased[n] ~ poisson(price[n] * x + b) T[,available[n]];
}
```