HW2

Isaac Kleisle-Murphy

4/14/2022

4.9.5

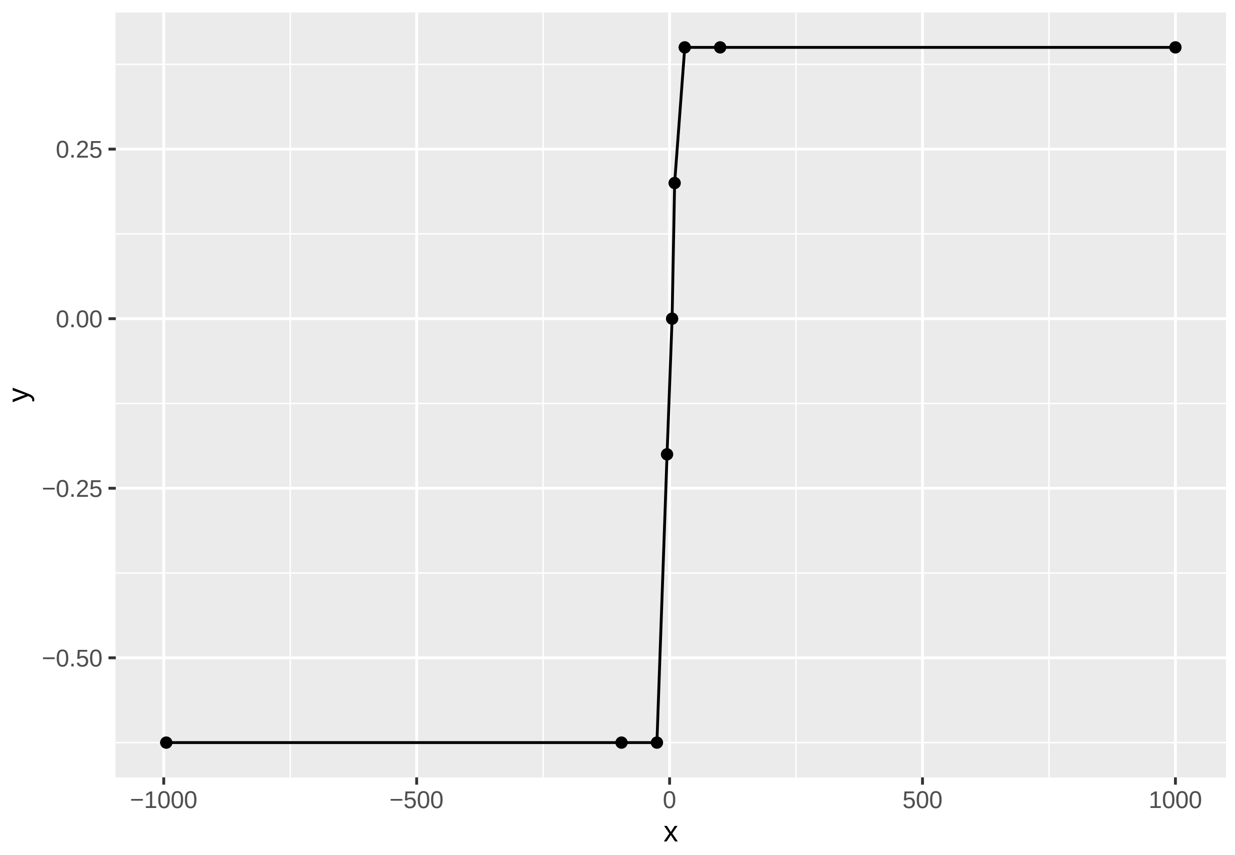
```
### setup data
data = data.frame(
  x=1:15,
  y=c(
    -7, 0, 5, 9, -3, -6, 18,
    8, -9, -20, -11, 4, -1, 7, 5
  )
)

perturb_test_wilcox <- function(data, value, idx=15){
  x = data$x; y = data$y;
  y[idx] = value
  fit = Rfit::rfit(y ~ x)
  coef(fit)[2]
}

# XSEQ = seq(-1000, 1000, length.out=1000)
XSEQ = c(-995, -95, -25, -5, 5, 10, 30, 100, 1000)

sapply(
  # c(-995, -95, -25, -5, 5, 10, 30, 100, 1000),
  XSEQ,
  function(yi){
    perturb_test_wilcox(data, value=yi, idx=15) -
      coef(Rfit::rfit(y ~ x, data=data))[2]
  }
) %>%
  as.numeric() -> result_wilcox

ggplot(
  data.frame(
    x=XSEQ,
    y=result_wilcox
  ),
  aes(x=x, y=y)
) +
  geom_path() +
  geom_point()
```



As we see, values are bounded (pretty close to zero), with a sharp switch from one bound to the other around x=0.

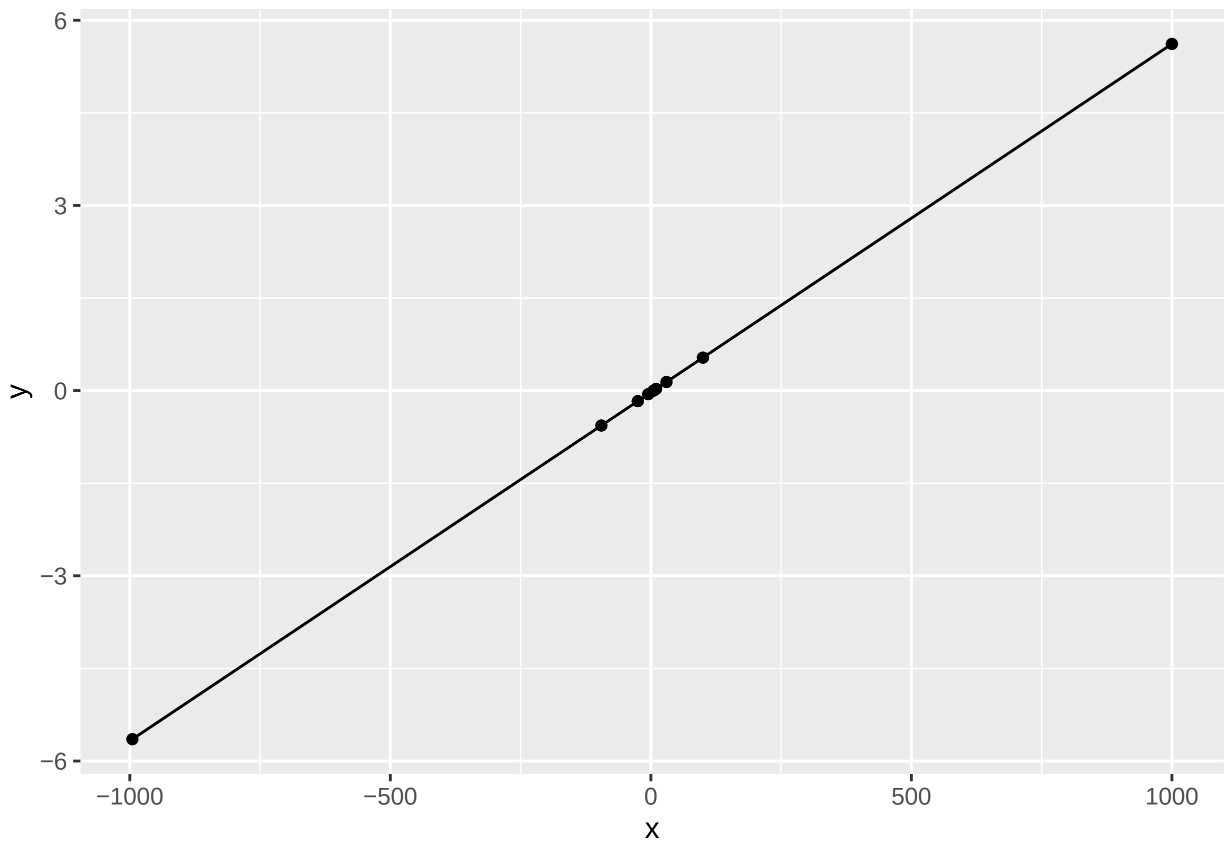
For the L-S fit, the sensitivity curve is given by:

```
ols <- function(x, y){
  y_ = y - mean(y)
  as.numeric(
    solve(t(x) %*% x) %*% t(x) %*% y_
  )
}

perturb_test_ols <- function(data, value, idx=15){
  x = data$x; y = data$y;
  y[idx] = value
  ols(x=x, y=y)
}


sapply(
  # c(-995, -95, -25, -5, 5, 10, 30, 100, 1000),
  XSEQ,
  function(yi){
    perturb_test_ols(data, value=yi, idx=15) -
      ols(data$x, data$y)
  }
) %>%
  as.numeric() -> result_ols

ggplot(
  data.frame(
    x=XSEQ,
    y=result_ols
  ),
  aes(x=x, y=y)
) +
  geom_path() +
  geom_point()
```



Now, sensitivity is linear, and critically, unbounded.

4.9.6

a.)

```
compute_theil_estimator <- function(x, y){
  N = length(x)
  bij = c()
  ### as instructed, 4.96
  for (j in 2:N){
    for (i in 1:(j-1)){ # i < j
      value = (y[j] - y[i]) / (x[j] - x[i])
      bij = append(bij, value)
    }
  }
  median(bij, na.rm=T)
}
```

b.)

```
compute_boot_theil <- function(x, y, S=10000, seed=2022, alpha=.05){
  #' alpha specfies conf.level, e.g. .05 --> .95
  set.seed(seed)
  sapply(1:S, function(s){
    slice_vec = sample(1:length(x), length(x), replace=T)
    xs = x[slice_vec]; ys = y[slice_vec]
    compute_theil_estimator(x=xs, y=ys)
  }) -> boot_result
  quantile(boot_result, c(alpha / 2, 1-alpha / 2))
}
```

Critically, under the bootstrap configuration, it's possible for $x_i = x_j$ due to sampling with replacement; this will throw a divide-by-zero. Here, I remedied this with a `na.rm=TRUE` argument to `median()` in `compute_theil_estimator`; however, I am unsure of best practices here.

c.)

To show this equality, it may be helpful to formalize the setup. Consider responses (group 1) $\{Y_i\}_{i=1,\ldots,n_1}$ corresponding to indicator features (as it is a two-sample problem) $\{X_i = 0\}_{i=1,\ldots,n_1} = 0$, as well as responses (group 2) $\{Y_i\}_{i=n_1+1,\ldots,n_1+n_2}$ corresponding to indicator features $\{X_i = 1\}_{i=n_1+1,\ldots,n_1+n_2}$. This gives designs and responses of

$$X = \begin{bmatrix} x_1 \\ \vdots \\ x_{n_1} \\ x_{n_1+1} \\ \vdots \\ x_{n_1+n_2} \end{bmatrix} = X = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

that are aligned with responses

$$Y = \begin{bmatrix} y_1 \\ \vdots \\ y_{n_1} \\ y_{n_1+1} \\ \vdots \\ y_{n_1+n_2} \end{bmatrix}.$$

If we ignore all i, j pairs such that $x_i - x_j = 0$ (which would throw a divisibility error in the Theil estimator), we are guaranteed that for all "eligible" i, j, we have $x_i - x_j = 1$, by the indicator. Hence, the Theil reduces to

$$\hat{\beta}_T = med\left(\frac{y_i - y_j}{x_i - x_j}\right) = med\left(\frac{y_i - y_j}{1}\right) = med\left(y_i - y_j\right).$$

But again, we only consider i, j where $x_i - x_j > 0$, i.e. only the cross group pairings. As such, the above reduces to

$$\hat{\beta}_T = \underset{i \leq n_1, n_1 < j \leq n_2 + n_1}{median} (y_i - y_j),$$

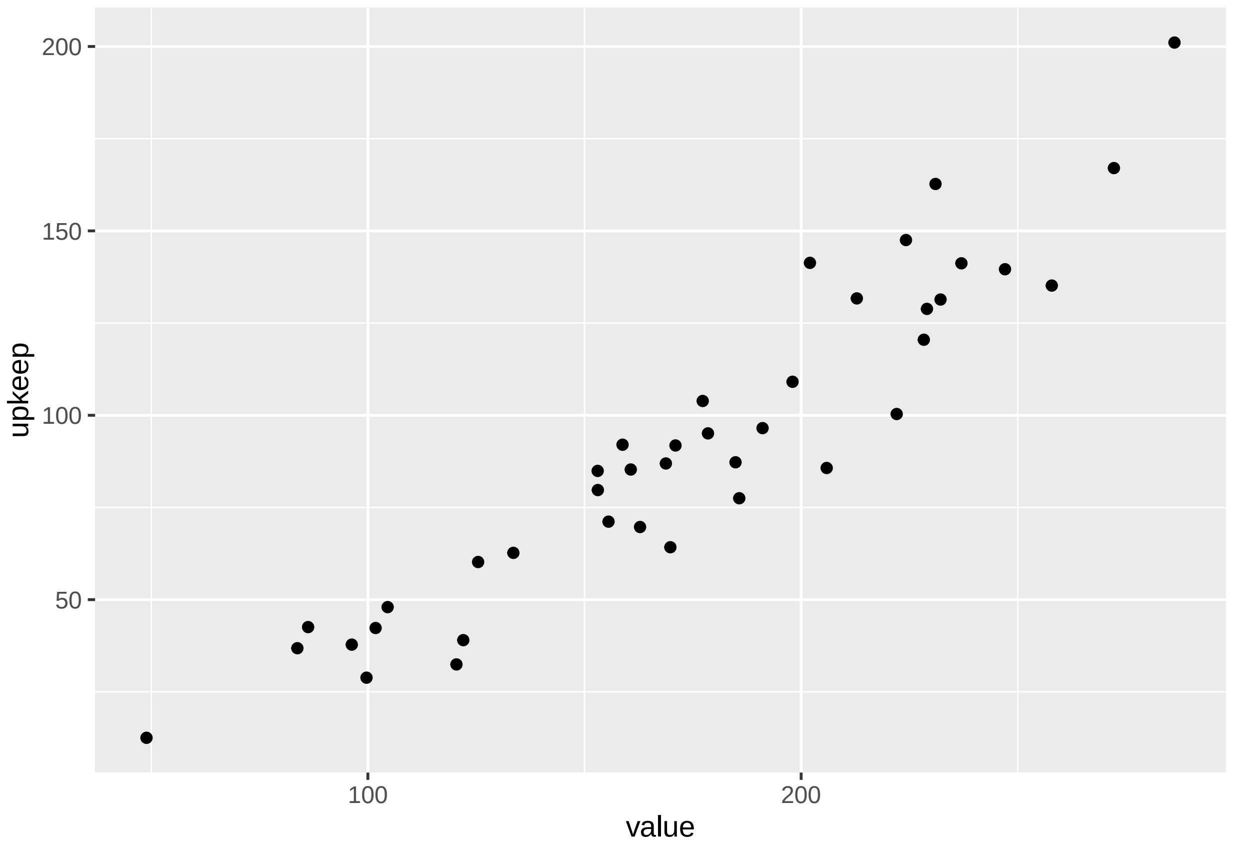
which is precisely the Hodges-Lehmann estimator.

4.9.8

a.)

The scatterplot:

```r
qhic_df = npsm::qhic
ggplot(qhic_df, aes(x=value, y=upkeep)) +
  geom_point()
```



b.)

First, for the linear setup, a Wilcoxon-fit and $\hat{\tau}$

```r
wfit = rfit(upkeep ~ value, qhic_df)
cat("Tau_hat: ", wfit$tauhat, "\n")
```

```
## Tau_hat:  13.76084
```

```r
cat("Robust R^2: ", summary(wfit)$R2)
```

```
## Robust R^2:  0.8145442
```

Then, for the quadratic, we similarly have

```r
wfit2 = rfit(upkeep ~ value + I(value^2), qhic_df)
cat("Tau_hat: ", wfit2$tauhat, "\n")
```

```
## Tau_hat:  15.77001
```

```r
cat("Robust R^2: ", summary(wfit2)$R2)
```

```
## Robust R^2:  0.8008794
```

The simpler model – the linear one – appears to hold up better, achieving a slightly better Robust R-squared. Hence, we'll go with that as our final model.

c.)

The prediction for 155,000 and associated CI is, in `(LWR, MEAN, UPR)` format:

```r
predict_rfit_ci <- function(fit, x0){
  x10 = c(1, x0)
  sterr = sqrt(
    t(x10) %*% vcov(fit) %*% x10
  ) %>% as.numeric()
  eta0 = as.numeric(t(x10) %*% fit$coefficients)
  c(
    # p = 1
    eta0 - qt(.975, df=length(fit$fitted.values) - 2) * sterr,
    eta0,
    eta0 + qt(.975, df=length(fit$fitted.values) - 2) * sterr
  )
}


predict_rfit_ci(wfit, 155)
```

```
## [1] 74.52601 78.91497 83.30393
```

d.)

For 255,000, we similarly have:

```r
predict_rfit_ci(wfit, 250)
```

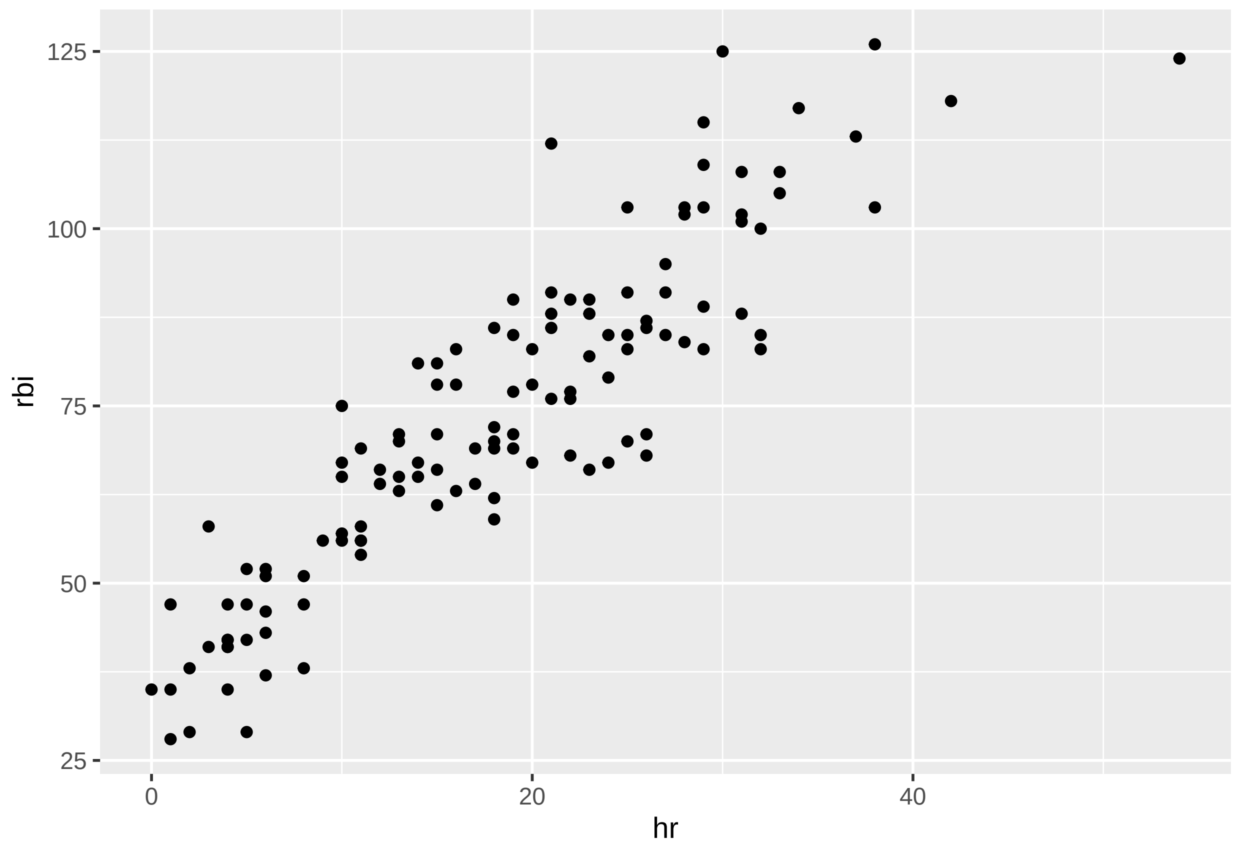
```
## [1] 137.8678 145.1009 152.3339
```

4.9.15

First, the scatterplot:

```
bb10 = npsm::bb2010

ggplot(bb10, aes(x=hr, y=rbi)) +
  geom_point()
```



Pearson:

```
cor(bb10$hr, bb10$rbi, method="pearson")
```

```
## [1] 0.9033529
```

Spearman:

```
cor(bb10$hr, bb10$rbi, method="spearman")
```

```
## [1] 0.9028116
```

Kendall:

```r
cor(bb10$hr, bb10$rbi, method="kendall")
```

```
## [1] 0.7467822
```

As we see, the Pearson and Spearman correlations are nearly identical, indicating that the correlation of the values is about that of the correlation of the rankings of the values – given the near perfect linearity, this makes some amount of sense. Elsewhere, the kendall correlation is much lower, suggesting the association is not as strong under Kendall's framework.

4.9.19

a.)

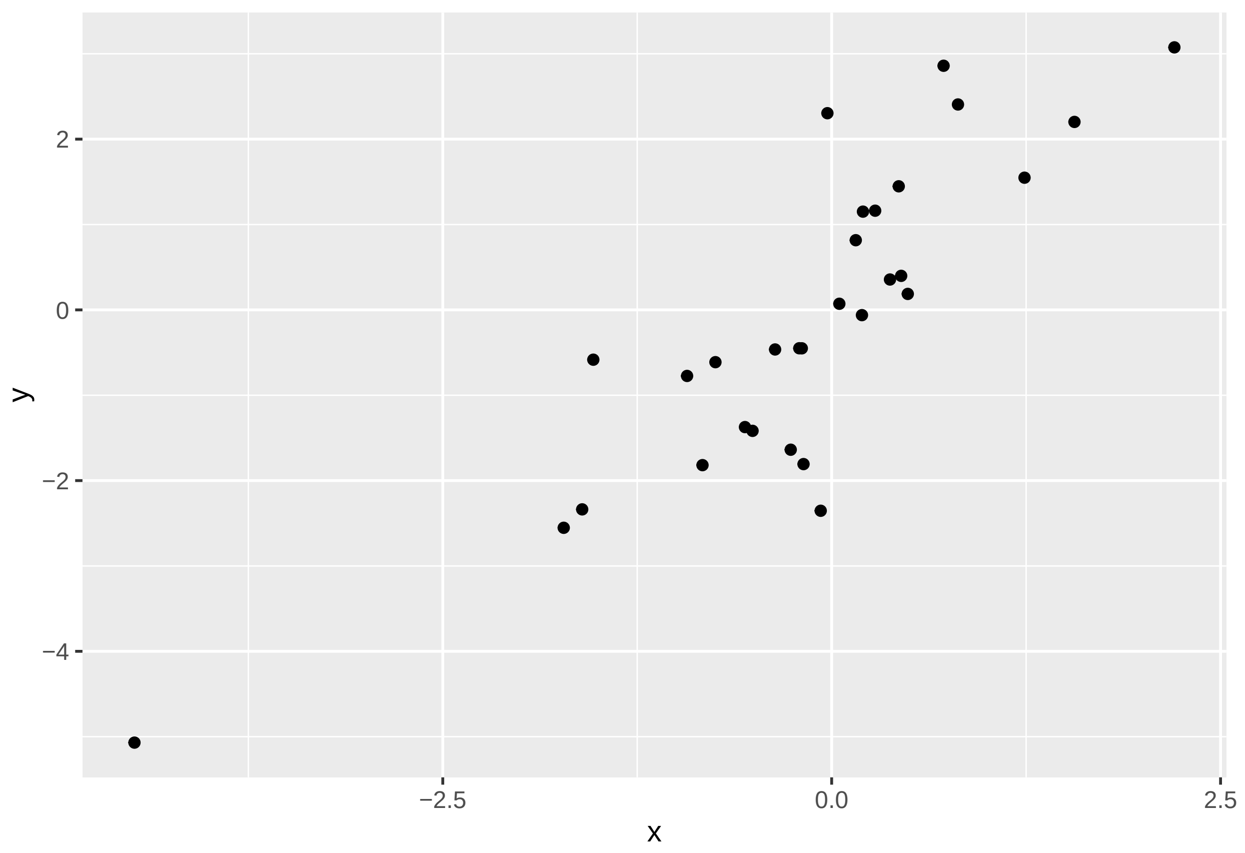
First, we write the generator function. We then run it once, make the scatter, and compute the correlations

```r
suppressMessages(library(jmuOutlier))

make_dataset <- function(n=30){
  X = jmuOutlier::rlaplace(30)
  Y = X + rnorm(30)
  data.frame(x=X, y=Y)
}

set.seed(2022)
testdata = make_dataset()

ggplot(testdata, aes(x=x, y=y)) + geom_point()
```



```r
cat("Pearson: ", cor(testdata$x, testdata$y, method="pearson"), "\n")
```

```
## Pearson:  0.8498657
```

```r
cat("Spearman: ", cor(testdata$x, testdata$y, method="spearman"), "\n")
```

```
## Spearman:  0.8660734
```

```r
cat("Kendall: ", cor(testdata$x, testdata$y, method="kendall"), "\n")
```

```
## Kendall:  0.6781609
```

b.)

Doing this 10,000 times over then yields the following:

```r
lapply(1:10000, function(i, ...){
  set.seed(i)
  testdata = make_dataset(...)
  c(
    cor(testdata$x, testdata$y, method="pearson"),
    cor(testdata$x, testdata$y, method="spearman"),
    cor(testdata$x, testdata$y, method="kendall")
  )
}) %>%
  do.call("rbind", .) %>%
  data.frame(.) %>%
  `colnames<-`(c("Pearson",  "Spearman", "Kendall")) -> sim_data
```

Means are:

```
colMeans(sim_data)
```

```
##   Pearson  Spearman   Kendall
## 0.6924902 0.6230481 0.4626423
```

SDs are:

```
sapply(sim_data, sd)
```

```
##   Pearson  Spearman   Kendall
## 0.1129089 0.1248887 0.1051303
```

and 95 CI's are:

```
sapply(sim_data, function(x) quantile(x, c(.025, .975)))
```

```
##         Pearson  Spearman   Kendall
## 2.5%  0.4283112 0.3454950 0.2413793
## 97.5% 0.8663953 0.8278087 0.6505747
```

Here, we see that the Spearman has the largest SD (`.1249`), while the Kendall has the smallest SD (`.1051`). Further, the Pearson correlation is generally the highest, with an average of `.692`. The Spearman is on average second highest, sitting at `.623`. Lastly, the Kendall correlation is a distant third, with an average of `.462`.

5.8.2

Setup and original test:

```
normal <- c(2.9,3.0,2.5,2.6,3.2)
obstruct <- c(3.8,2.7,4.0,2.4)
asbestosis <- c(2.8,3.4,3.7,2.2,2.0)
x <- c(normal,obstruct,asbestosis)
g <- c(rep(1,5),rep(2,4),rep(3,5))
kruskal.test(x,g)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  x and g
## Kruskal-Wallis chi-squared = 0.77143, df = 2, p-value = 0.68
```

The F_W fit is then given by:

```
Rfit::oneway.rfit(x, g)
```

```
## Call:
## Rfit::oneway.rfit(y = x, g = g)
##
## Overall Test of All Locations Equal
##
## Drop in Dispersion Test
## F-Statistic     p-value
##     0.32036     0.73244
##
##
##  Pairwise comparisons using Rfit
##
## data:  x and g
##
##   1    2
## 2 0.69 -
## 3 0.89 0.59
##
## P value adjustment method: none
```

where now, the p-value runs slightly higher than that of the original Kruskal-Wallis p-value. Lastly, for the MCP analysis via Tukey's method, we have:

```
summary(Rfit::oneway.rfit(x, g), method="tukey")
```

```
##
## Multiple Comparisons
## Method Used  tukey
##
##   I J Estimate  St Err Lower Bound CI Upper Bound CI
## 1 1 2      0.3 0.72587       -1.66048        2.26047
## 2 1 3     -0.1 0.68436       -1.94836        1.74835
## 3 2 3      0.4 0.72587       -1.56047        2.36047
```

As all of the `I, J` intervals contain zero here, we (by the duality of CIs and p-values) again would have a p-value that would fail to reject. This is yet another finding in line with the K-W analysis, which concluded similarly.

5.9.3

Results and p-value are presented below:

```
g1 = c(40, 35, 38, 43, 44, 41)
g2 = c(38, 40, 47, 44, 40, 42)
g3 = c(48, 40, 45, 43, 46, 44)
x = c(g1, g2, g3)
g = c(rep(1, 6), rep(2, 6), rep(3, 6))
kruskal.test(x, g)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  x and g
## Kruskal-Wallis chi-squared = 4.3615, df = 2, p-value = 0.113
```

This would probably not be enough to reject any null.

5.9.13

We proceed by running our usual Wilcoxon regression (much like at the top of the HW), albeit with `latitude` and `longitude` as features.

```
library(HSAUR2)
```

```
## Loading required package: tools
```

```
##
## Attaching package: 'HSAUR2'
```

```
## The following object is masked from 'package:robustbase':
##
##     epilepsy
```

```
mel_df = HSAUR2::USmelanoma

x1 = mel_df$latitude; x2 = mel_df$longitude
y = mel_df$mortality

fit = Rfit::rfit(y ~ x1 + x2)

fit %>% summary()
```

```
## Call:
## rfit.default(formula = y ~ x1 + x2)
##
## Coefficients:
##              Estimate Std. Error t.value   p.value
## (Intercept) 395.51462   32.65174 12.1131 6.521e-16 ***
## x1           -5.84221    0.69961 -8.3507 9.042e-11 ***
## x2           -0.13289    0.21699 -0.6124    0.5433
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Multiple R-squared (Robust): 0.5760564
## Reduction in Dispersion Test: 31.2525 p-value: 0
```

As we see, the drop-in-dispersion test returns a p-value of close to zero, indicating that under a null of no association and after controlling for `lat`, `long`, such a test would likely be rejected, and there does appear to be some meaningful association (after controlling for `lat`, `long`).

7.9.1

Setup:

```
data = data.frame(
  x = c(1:10, 20),
  y = c(5, 7, 6, 14, 14, 25, 29, 33, 31, 41, 75),
  y2 = c(5, 7, 6, 14, 14, 25, 29, 33, 31, 41, 20)
)
```

a.)

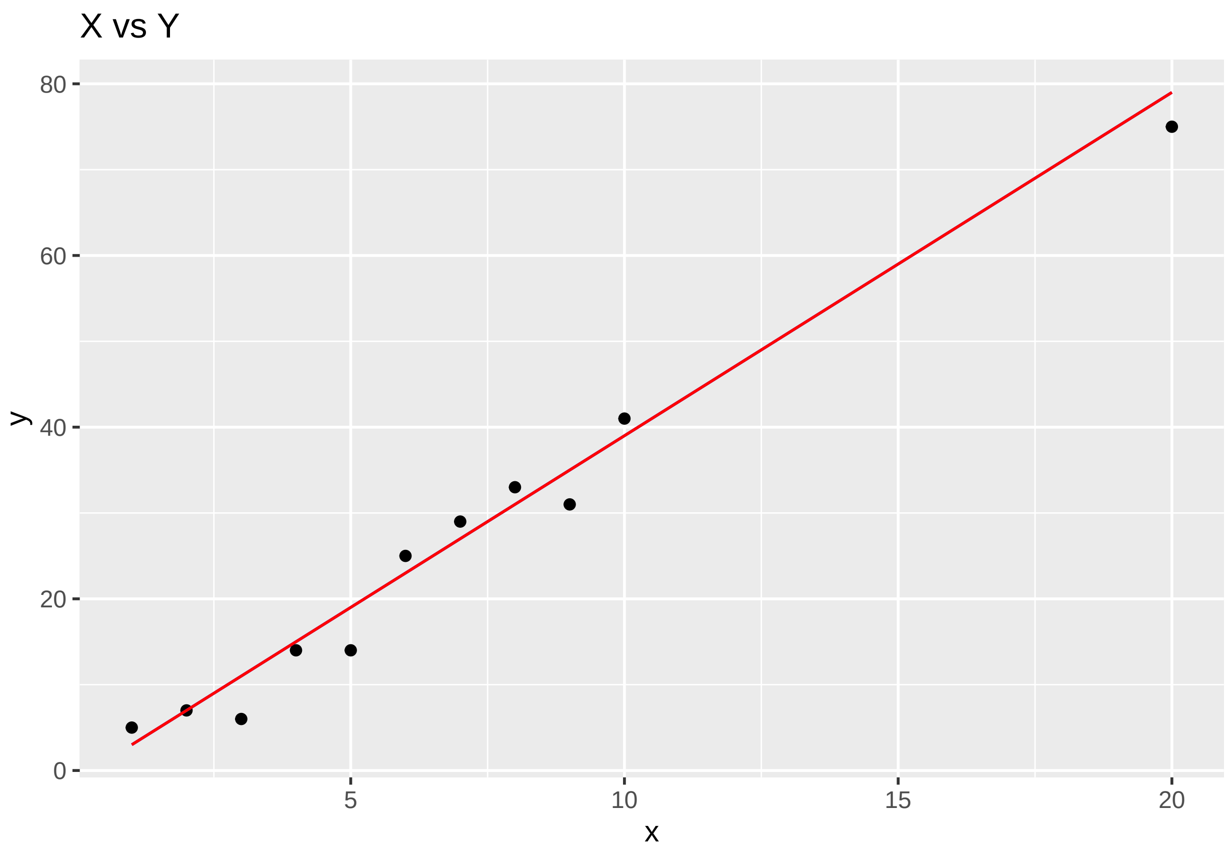
In the non-outlying setting, the two regressions are equivalent. The red represents the `HBR` fit, while the blue represents the `Wilcoxon` fit.

```
# install.packages("quantreg) # need rq()
# install.packages("remotes")
# remotes::install_github("kloke/hbrfit")
library(quantreg)
library(hbrfit)

### wilcox fit
wilcox_fit = Rfit::rfit(y ~ x, data)
yhat_wilc = wilcox_fit$fitted.values
data$yhat_wilcox = yhat_wilc

hbr_fit = hbrfit::hbrfit(y ~  x, data)
data$yhat_hbr = hbr_fit$fitted.values

ggplot(data, aes(x=x, y=y)) +
  geom_point() +
  labs(title="X vs Y") +
  geom_path(
    aes(x=x, y=yhat_wilcox),
    color="blue"
  ) +
  geom_path(
    aes(x=x, y=yhat_hbr),
    color="red"
  )
```



b .)

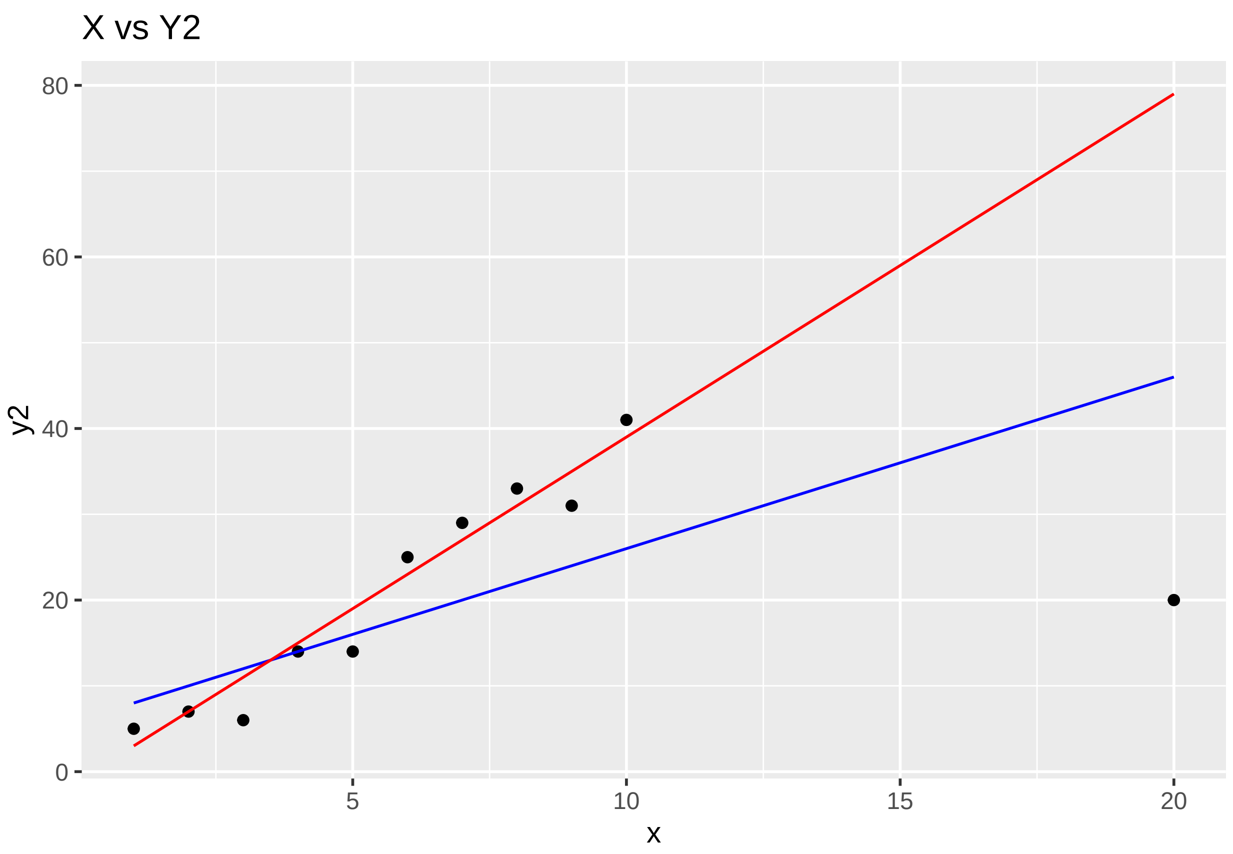
Now, the `HBR` fit handles the included outlier much more gracefully, as it is not pulled down by the high leverage point at `y2=20`. Again, the red represents the `HBR` fit, while the blue represents the `Wilcoxon` fit.

```
# install.packages("quantreg") # need rq()
# install.packages("remotes")
# remotes::install_github("kloke/hbrfit")
library(quantreg)
library(hbrfit)

### wilcox fit
wilcox_fit = Rfit::rfit(y2 ~ x, data)
yhat_wilc = wilcox_fit$fitted.values
data$yhat_wilcox = yhat_wilc

hbr_fit = hbrfit::hbrfit(y2 ~  x, data)
data$yhat_hbr = hbr_fit$fitted.values

ggplot(data, aes(x=x, y=y2)) +
  geom_point() +
  labs(title="X vs Y2") +
  geom_path(
    aes(x=x, y=yhat_wilcox),
    color="blue"
  ) +
  geom_path(
    aes(x=x, y=yhat_hbr),
    color="red"
  )
```



c.)

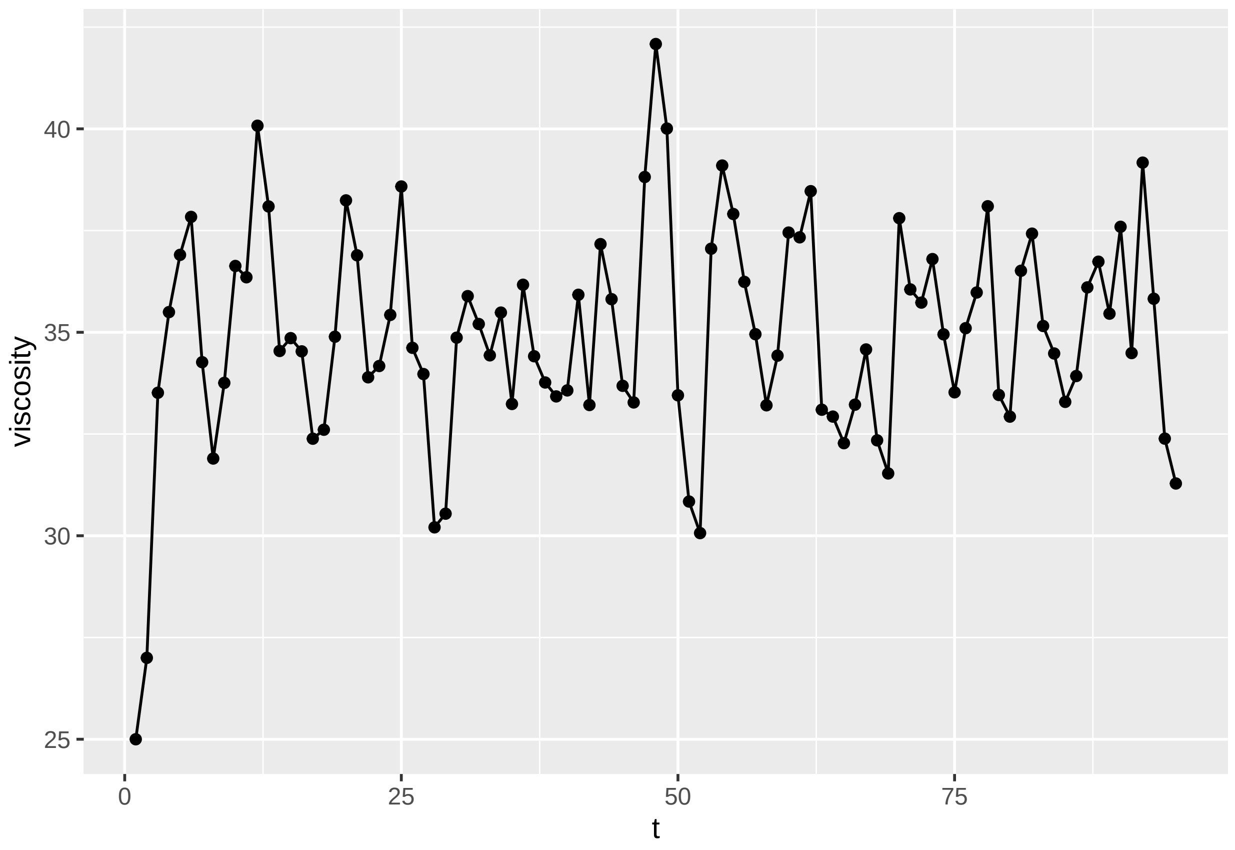
See commentary above.

7.9.19

a.)

```
# install.packages("remotes")
# remotes::install_github("kloke/npsmReg2")
library(npsmReg2)
library(quantreg)
data("viscosity")
visc_df = data.frame(t=1:length(viscosity), viscosity=viscosity)

ggplot(visc_df, aes(x=t, y=viscosity)) +
  geom_point() +
  geom_path()
```



b.)

```
visc = lagmat(viscosity, 4)
x = visc[, 1]
xmat = visc[, 2:ncol(visc)]
hbr = hbrfit(x ~ xmat)
varcov = vcov(hbr, details=T)

theta = hbr$coefficients
arorder(length(x), 4, theta, varcov)$results
```

```
##   [,1]      [,2]         [,3]
##      4 0.3708743 5.441335e-01
##      3 2.6951274 7.323830e-02
##      2 8.2931755 6.598272e-05
```

Here, we choose `order=2`, as per the procedure in 7.8.1. This would seem to be in agreement with Bowerman et al.

c.)

```
visc = lagmat(viscosity, 2)
x = visc[, 1]
xmat = visc[, 2:3]
fit = Rfit::rfit.default(x ~ xmat)
summary(fit)
```

```
## Call:
## Rfit::rfit.default(formula = x ~ xmat)
##
## Coefficients:
##             Estimate Std. Error t.value   p.value
## (Intercept) 27.69619    3.48326  7.9512 5.138e-12 ***
## xmat         0.57944    0.09989  5.8008 9.698e-08 ***
## xmat        -0.37235    0.09236 -4.0315 0.0001158 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Multiple R-squared (Robust): 0.2697196
## Reduction in Dispersion Test: 16.62017 p-value: 0
```

For the AR(2) params, this gives Wald confidence intervals of

```
cat(
  "theta_1: ",
  0.57944 - qnorm(.975) * 0.09989,
  ", ",
  0.57944 + qnorm(.975) * 0.09989 ,
  "\n"
)
```

```
## theta_1:  0.3836592 ,  0.7752208
```

```
cat(
  "theta_2: ",
  -0.37235 - qnorm(.975) * 0.09236,
  ", ",
  -0.37235 + qnorm(.975) * 0.09236
)
```

```
## theta_2:  -0.5533723 ,  -0.1913277
```

If we instead opt for a T-distribution for the CI (whereas Wald uses normal quantiles), we then get (subtract 2 + 1 df, for fit and usual -1):

```
cat(
  "theta_1: ",
  0.57944 - qt(.975, nrow(visc) - 3) * 0.09989,
  ", ",
  0.57944 + qt(.975, nrow(visc) - 3) * 0.09989 ,
  "\n"
)
```

```
## theta_1:  0.3809911 ,  0.7778889
```

```
cat(
  "theta_2: ",
  -0.37235 - qt(.975, nrow(visc) - 3) * 0.09236,
  ", ",
  -0.37235 + qt(.975, nrow(visc) - 3) * 0.09236
)
```

```
## theta_2:  -0.5558393 ,  -0.1888607
```

d.)

The prediction is:

```
theta = coef(fit)
x96 = c(1, visc[nrow(visc), 1:2])

yhat_96 = sum(theta %*% x96)
yhat_96
```

```
## [1] 33.76438
```

e.)

The confidence interval is:

```
sterr = sqrt(t(x96) %*% vcov(fit) %*% x96)

ci_96 = c(yhat_96 - qnorm(.975) * sterr, yhat_96 + qnorm(.975) * sterr)
ci_96
```

```
## [1] 32.87478 34.65398
```

f.) // g.)

Note - in this problem, we are instructed to follow the confidence interval procedure set forth in 4.4.4. As instructed, I do this in the code chunks below, but I am not sure this is the correct procedure. More specifically, I don't think that a simple $\sqrt{x^T \Sigma x}$ is appropriate for the AR(2) model in the n-ahead CIs (i.e. the`t=97, 98` cases in the second half of the problem), as the entries of the "x" vector are being updated with predictions, so uncertainty should compound the further out you go. Just making a note of it here – I'm following textbook instructions, but this may be worth clarification in class.TL; DR, I don't think it's as simple as taking an OLS CI in the n-ahead situations, even though we are instructed to do so.

The same, but for `t=97`. Here, we're just stepping forward, adding old predictions into the lag and updating after each iteration. The mean:

```
x97 = c(1, yhat_96, x96[2])
yhat_97 = sum(theta %*% x97)
yhat_97
```

```
## [1] 35.61194
```

The CI is:

```
sterr = sqrt(t(x97) %*% vcov(fit) %*% x97)

ci_97 = c(yhat_97 - qnorm(.975) * sterr, yhat_97 + qnorm(.975) * sterr)
ci_97
```

```
## [1] 34.77366 36.45021
```

Lastly, for `t=98`, we repeat once more. The mean:

```
x98 = c(1, yhat_97, yhat_96)
yhat_98 = sum(theta %*% x98)
yhat_98
```

```
## [1] 35.75915
```

The CI is:

```
sterr = sqrt(t(x98) %*% vcov(fit) %*% x98)

ci_98 = c(yhat_98 - qnorm(.975) * sterr, yhat_98 + qnorm(.975) * sterr)
ci_98
```

```
## [1] 35.10278 36.41552
```