geologyphd

George Brown and Claudia Chen

2022-11-16

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.4.0      v purrr   0.3.5
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.5.0
## v readr   2.1.3      v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(glmnet)
```

```
## Loading required package: Matrix
##
## Attaching package: 'Matrix'
##
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
##
## Loaded glmnet 4.1-6
```

```
library(rstanarm)
```

```
## Loading required package: Rcpp
## This is rstanarm version 2.21.3
## - See https://mc-stan.org/rstanarm/articles/priors for changes to default priors!
## - Default priors may change, so it's safest to specify priors, even if equivalent to the defaults.
## - For execution on a local, multicore CPU with excess RAM we recommend calling
##   options(mc.cores = parallel::detectCores())
```

```
library(boot)
```

```
##
## Attaching package: 'boot'
##
## The following object is masked from 'package:rstanarm':
##
##     logit
```

```
correlated_dataset <- read_rds("C:/Users/chen1/Downloads/correlated_dataset.RDS")
```

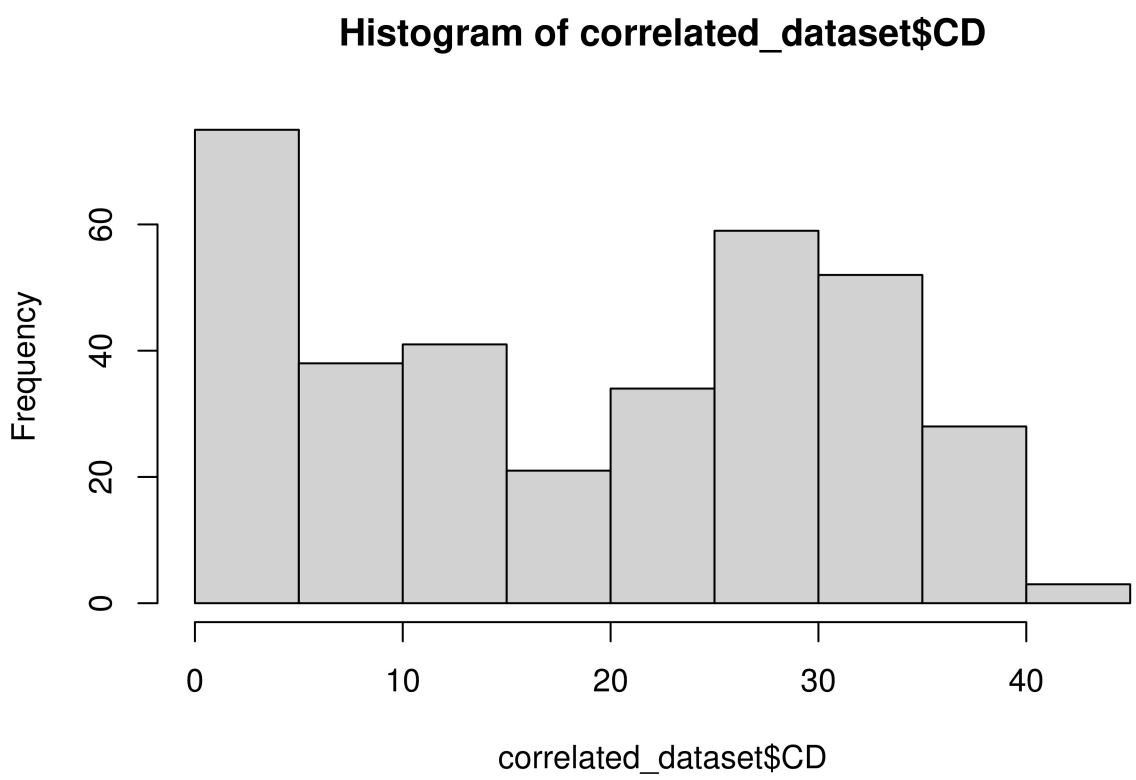
```
head(correlated_dataset)
```

```
## # A tibble: 6 x 5
##   cell_id      organism           label_at2H at2H_percent_dtc `CD%`
##   <chr>        <chr>                   <dbl>            <dbl> <dbl>
## 1 Thy_00_07_01 T. hydrogeniphilus          0          0.00916  2.02
## 2 Thy_00_07_02 T. hydrogeniphilus          0          0.0180   3.16
## 3 Thy_00_07_03 T. hydrogeniphilus          0          0.0257   4.62
## 4 Thy_00_07_04 T. hydrogeniphilus          0          0.00902  4.74
## 5 Thy_00_07_05 T. hydrogeniphilus          0          0        1.43
## 6 Thy_00_07_08 T. hydrogeniphilus          0          0.00952  2.13
```

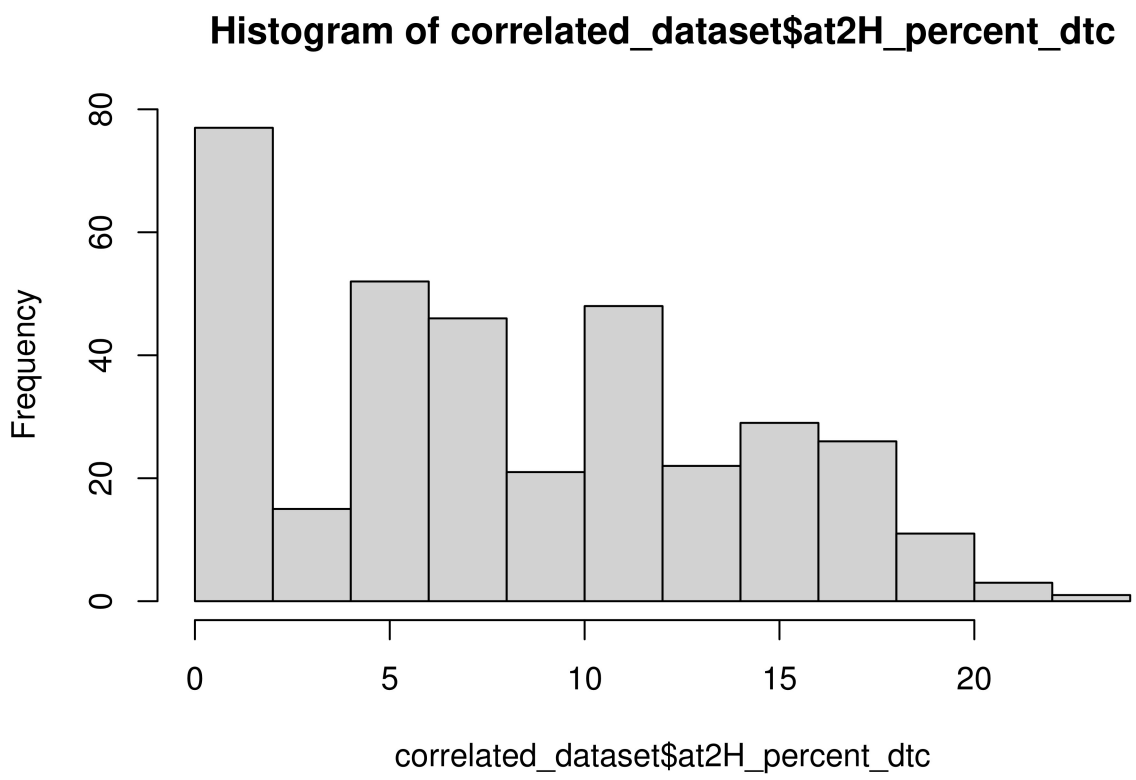
```
names(correlated_dataset)[5] <- "CD"
head(correlated_dataset)
```

```
## # A tibble: 6 x 5
##   cell_id      organism           label_at2H at2H_percent_dtc    CD
##   <chr>        <chr>                   <dbl>            <dbl> <dbl>
## 1 Thy_00_07_01 T. hydrogeniphilus          0          0.00916  2.02
## 2 Thy_00_07_02 T. hydrogeniphilus          0          0.0180   3.16
## 3 Thy_00_07_03 T. hydrogeniphilus          0          0.0257   4.62
## 4 Thy_00_07_04 T. hydrogeniphilus          0          0.00902  4.74
## 5 Thy_00_07_05 T. hydrogeniphilus          0          0        1.43
## 6 Thy_00_07_08 T. hydrogeniphilus          0          0.00952  2.13
```

```
hist(correlated_dataset$CD)
```

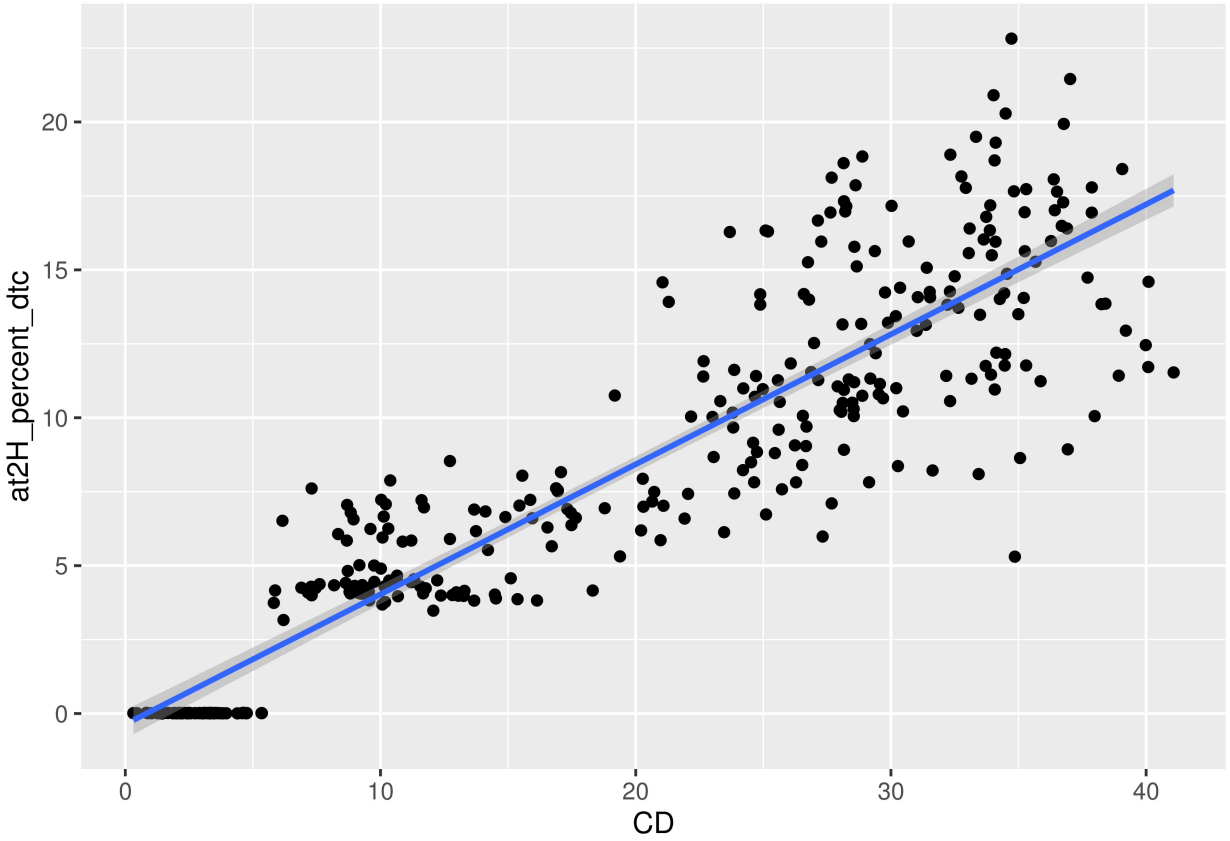



```
hist(correlated_dataset$at2H_percent_dtc)
```




```
#on all the data
ggplot(correlated_dataset, aes(x = CD, y = at2H_percent_dtc)) +
    geom_point() +
    geom_smooth(method = "lm")
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



Can see that variance increases as fractional abundance increases.

```
set.seed(1)
train = sample(c(TRUE,FALSE),nrow(correlated_dataset),rep=TRUE) #randomly choose
test = (!train) #anything not in train

train.geo <- correlated_dataset[train,]
test.geo <- correlated_dataset[test,]

nrow(train.geo)
```

```
## [1] 173
```

```
nrow(test.geo)
```

```
## [1] 178
```

```
#Simple linear model with all data included
#We flip CD% and at2H% because we want to predict against the known quantity
linear.fit <- lm(at2H_percent_dtc ~ CD, data = correlated_dataset)
summary(linear.fit)
```

```
##
## Call:
```

```
## lm(formula = at2H_percent_dtc ~ CD, data = correlated_dataset)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.6515 -1.3046 -0.3259  1.3120  7.9207
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.36173    0.24448   -1.48     0.14
## CD           0.43944    0.01086   40.48   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.497 on 349 degrees of freedom
## Multiple R-squared:  0.8244, Adjusted R-squared:  0.8239
## F-statistic:  1639 on 1 and 349 DF,  p-value: < 2.2e-16
```

```
logarithmic.fit <- lm(at2H_percent_dtc ~ log(CD), data = correlated_dataset)
summary(logarithmic.fit)
```

```
##
## Call:
## lm(formula = at2H_percent_dtc ~ log(CD), data = correlated_dataset)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.5109 -2.0693 -0.7406  1.6056 10.4017
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -4.689      0.431  -10.88   <2e-16 ***
## log(CD)        4.929      0.156   31.60   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.033 on 349 degrees of freedom
## Multiple R-squared:  0.7411, Adjusted R-squared:  0.7403
## F-statistic: 998.8 on 1 and 349 DF,  p-value: < 2.2e-16
```

Test Error for Various Models

```
set.seed(1)

#lm
linear.fit.train = lm(at2H_percent_dtc~CD, data=train.geo)
lm.pred.geo = predict(linear.fit.train, test.geo, type="response")
mean((test.geo$at2H_percent_dtc - lm.pred.geo)^2) #mean squared error
```

```
## [1] 6.364385
```

```
#MSE: residual squared (difference between actual and predicted)

#logarithmic
logarithmic.fit.train = lm(at2H_percent_dtc~log(CD), data=train.geo)
logarithmic.pred.geo = predict(logarithmic.fit.train, test.geo, type="response")
mean((test.geo$at2H_percent_dtc - lm.pred.geo)^2)
```

```
## [1] 6.364385
```

```
#ridge
train.matrix = model.matrix(at2H_percent_dtc~CD, data = train.geo) #for glmnet
test.matrix = model.matrix(at2H_percent_dtc~CD, data = test.geo)

cv.lam = cv.glmnet(train.matrix, train.geo$at2H_percent_dtc, alpha=0)
lam = cv.lam$lambda.min

ridge.fit = glmnet(train.matrix, train.geo$at2H_percent_dtc, alpha = 0)
ridge.pred.geo = predict(ridge.fit, s=lam, newx = test.matrix)
mean((train.geo$at2H_percent_dtc - ridge.pred.geo)^2)
```

```
## Warning in train.geo$at2H_percent_dtc - ridge.pred.geo: longer object length is
## not a multiple of shorter object length
```

```
## [1] 21.03844
```

```
#lasso
cv.lam2 = cv.glmnet(train.matrix, train.geo$at2H_percent_dtc, alpha=1)
lam2 = cv.lam$lambda.min

lasso.fit = glmnet(train.matrix, train.geo$at2H_percent_dtc, alpha = 1)
lasso.pred.geo = predict(lasso.fit, s=lam2, newx = test.matrix)
mean((train.geo$at2H_percent_dtc - lasso.pred.geo)^2)
```

```
## Warning in train.geo$at2H_percent_dtc - lasso.pred.geo: longer object length is
## not a multiple of shorter object length
```

```
## [1] 20.88155
```

Linear model has the best MSE out of all the model options- corroborates idea that we can predict fractional abundance using either method. Actually lower errors across the board using raman to predict nanoSIMS.

Bootstrapping Resampling method with replacement in order to get overall distribution from one sample

```
#Function to find coefficients
coefficients <- function(formula, data, indices) {
  d <- correlated_dataset[indices,] # allows boot to select sample
  fit <- lm(at2H_percent_dtc~CD, data=d)
  coef(fit) #coefficients
  #return(summary(fit)$r.square) #correlation
}
# Performing 1500 replications with boot
```

```
bootModel <- boot(data=correlated_dataset, statistic=coefficients, R=1500)

#coefficients
bootModel$t0
```

```
## (Intercept)          CD
##  -0.3617293   0.4394351
```

```
m=bootModel$t0[2]
b=bootModel$t0[1]

my.function = function(x){
  return(m*x+b)
}

bootPred = sapply(test.geo$CD, my.function)


mean((test.geo$at2H_percent_dtc - bootPred)^2)
```

```
## [1] 6.294873
```

Bootstrapping with a linear model has an MSE error of 6.294893 which is the best model overall.

```
##Calculate various versions of error for the bootstrapping linear model

#MSE: residual squared (difference between actual and predicted)
#more weight to larger errors than smaller ones
mean((test.geo$at2H_percent_dtc - bootPred)^2) #mean squared error
```

```
## [1] 6.294873
```

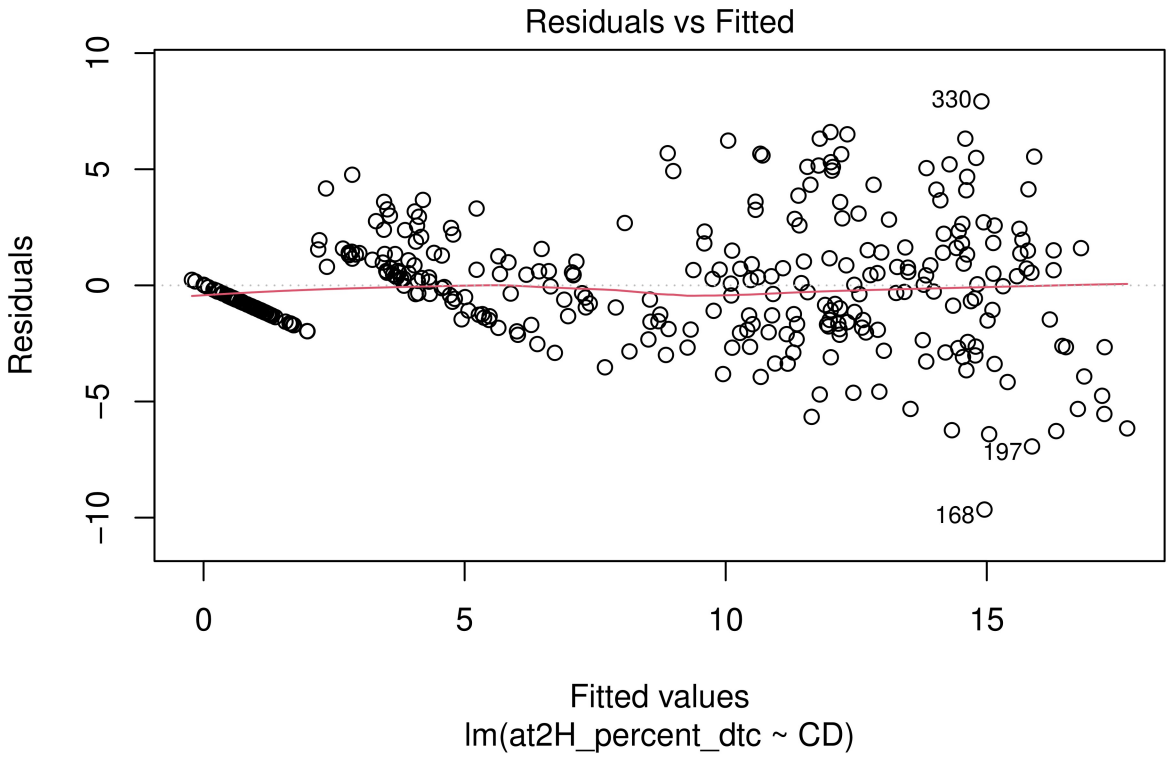
```
#RMSE: square root of MSE (more meaning for units)
(mean((test.geo$at2H_percent_dtc - bootPred)^2))^(1/2)
```

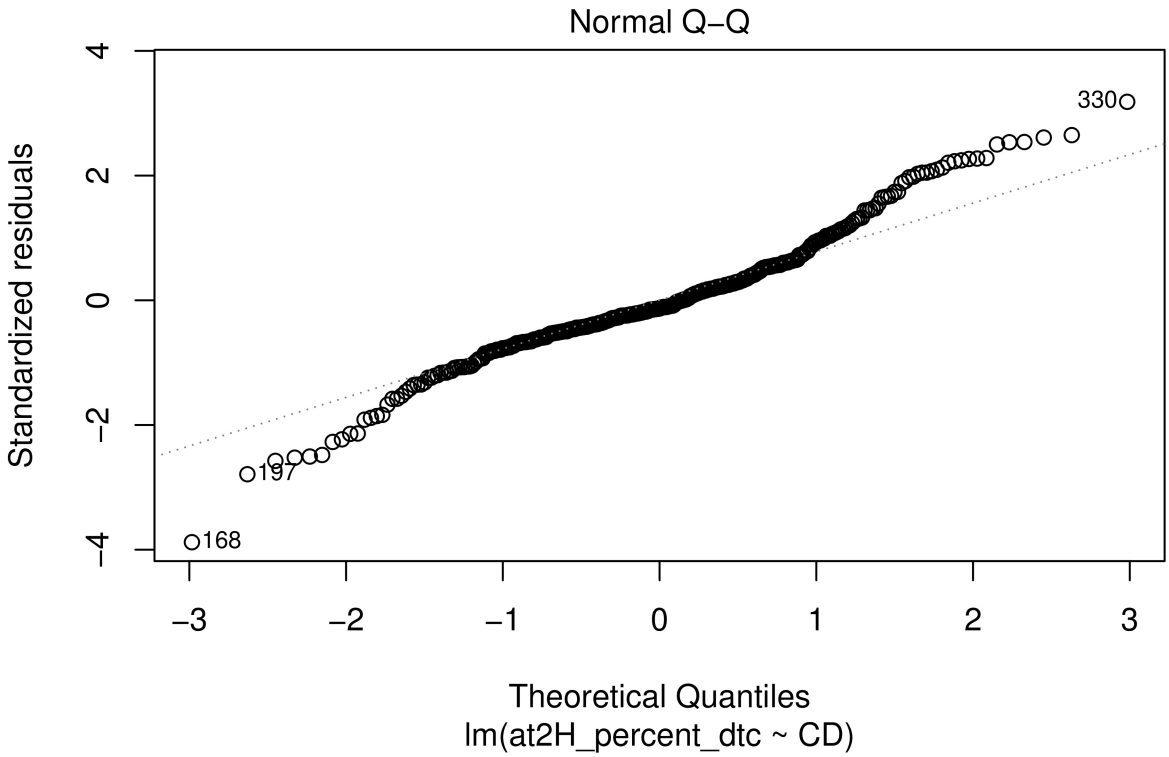
```
## [1] 2.508959
```

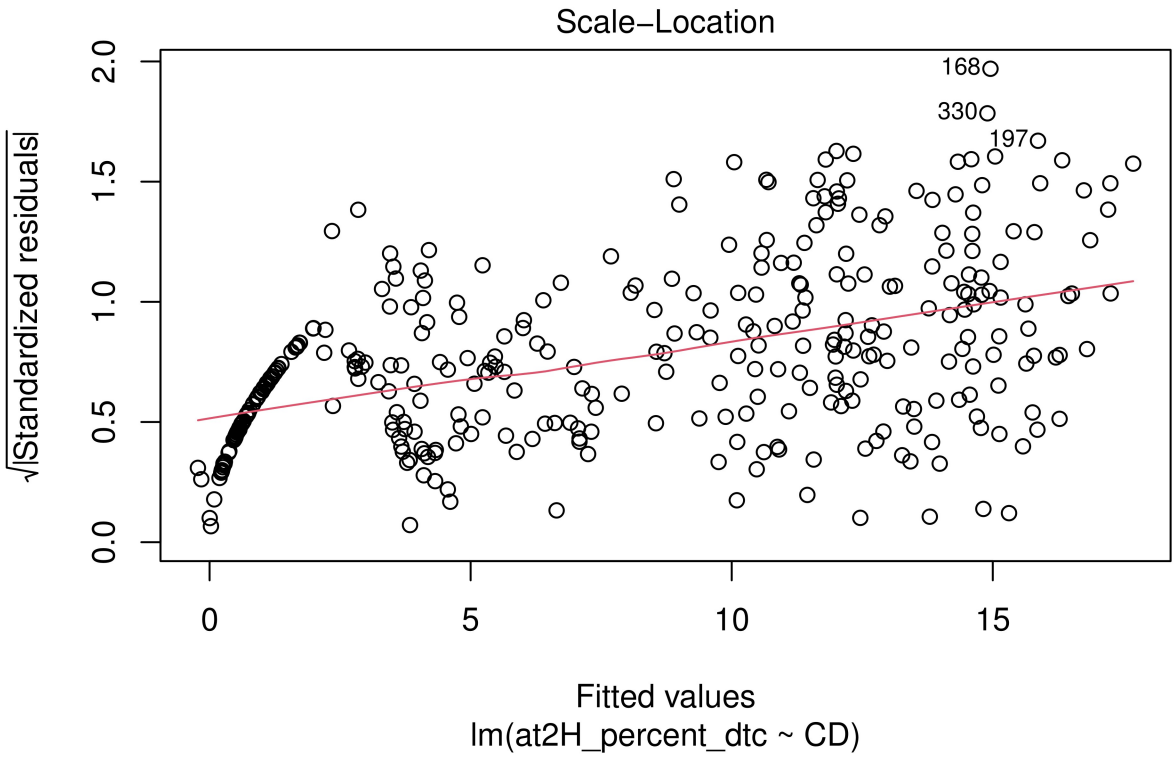
```
#MAE: sum of absolute errors divided by sample size
#same weight for larger and smaller errors
mean(abs(mean(test.geo$at2H_percent_dtc - bootPred)))
```

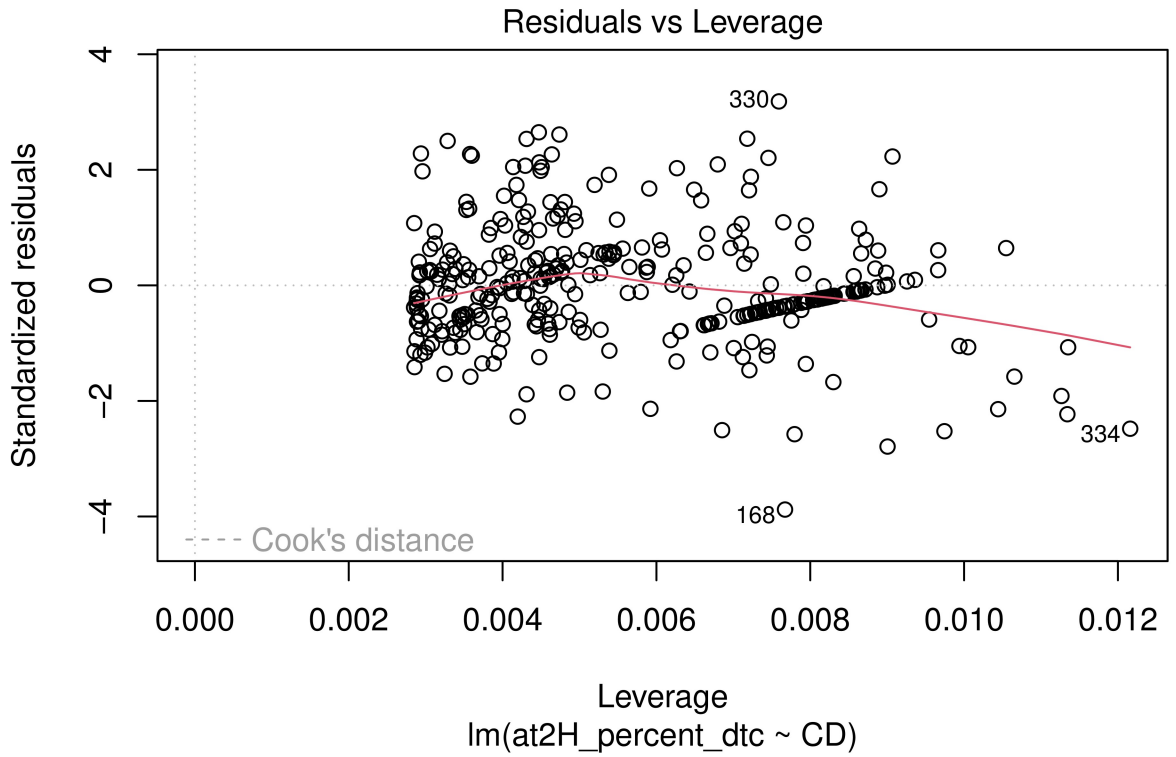
```
## [1] 0.045119
```

```
plot(linear.fit)
```


Residuals vs Fitted
Residuals
10
5
0
−5
−10
330
197
168
0
5
10
15
Fitted values
lm(at2H_percent_dtc ~ CD)


Normal Q−Q
Standardized residuals
Theoretical Quantiles
lm(at2H_percent_dtc ~ CD)
330
197
168
−4
−2
0
2
4
−3
−2
−1
0
1
2
3


Scale−Location
√|Standardized residuals|
0.0
0.5
1.0
1.5
2.0
168
330
197
0
5
10
15
Fitted values
lm(at2H_percent_dtc ~ CD)



Leverage/Cook's distance indicate no influential points in our dataset skewing the data; normality assumption is met indicated Normal QQ plot; residuals vs fitted plot is at zero, per expectations.

Credible Intervals

Sigma indicates standard error in this Bayesian approach.

```
stan.fit <- stan_glm(at2H_percent_dtc ~ CD, data = correlated_dataset)
```

```
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.002 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 20 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
```

```
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.179 seconds (Warm-up)
## Chain 1:                0.208 seconds (Sampling)
## Chain 1:                0.387 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.178 seconds (Warm-up)
## Chain 2:                0.179 seconds (Sampling)
## Chain 2:                0.357 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
```

```
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.107 seconds (Warm-up)
## Chain 3:                0.206 seconds (Sampling)
## Chain 3:                0.313 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.11 seconds (Warm-up)
## Chain 4:                0.123 seconds (Sampling)
## Chain 4:                0.233 seconds (Total)
## Chain 4:
```

```
summary(stan.fit)
```

```
##
## Model Info:
##  function:     stan_glm
##  family:       gaussian [identity]
##  formula:      at2H_percent_dtc ~ CD
##  algorithm:    sampling
##  sample:       4000 (posterior sample size)
##  priors:       see help('prior_summary')
##  observations: 351
##  predictors:   2
##
## Estimates:
##               mean   sd   10%   50%   90%
## (Intercept) -0.4    0.2 -0.7  -0.4  -0.1
## CD           0.4    0.0  0.4   0.4   0.5
## sigma        2.5    0.1  2.4   2.5   2.6
##
## Fit Diagnostics:
##            mean   sd   10%   50%   90%
## mean_PPD 7.9    0.2  7.7   7.9   8.2
```

```
##
## The mean_ppd is the sample average posterior predictive distribution of the outcome variable (for de
##
## MCMC diagnostics
##               mcse Rhat n_eff
## (Intercept)   0.0  1.0  3880
## CD            0.0  1.0  3809
## sigma         0.0  1.0  3241
## mean_PPD      0.0  1.0  3626
## log-posterior 0.0  1.0  1854
##
## For each parameter, mcse is Monte Carlo standard error, n_eff is a crude measure of effective sample
```

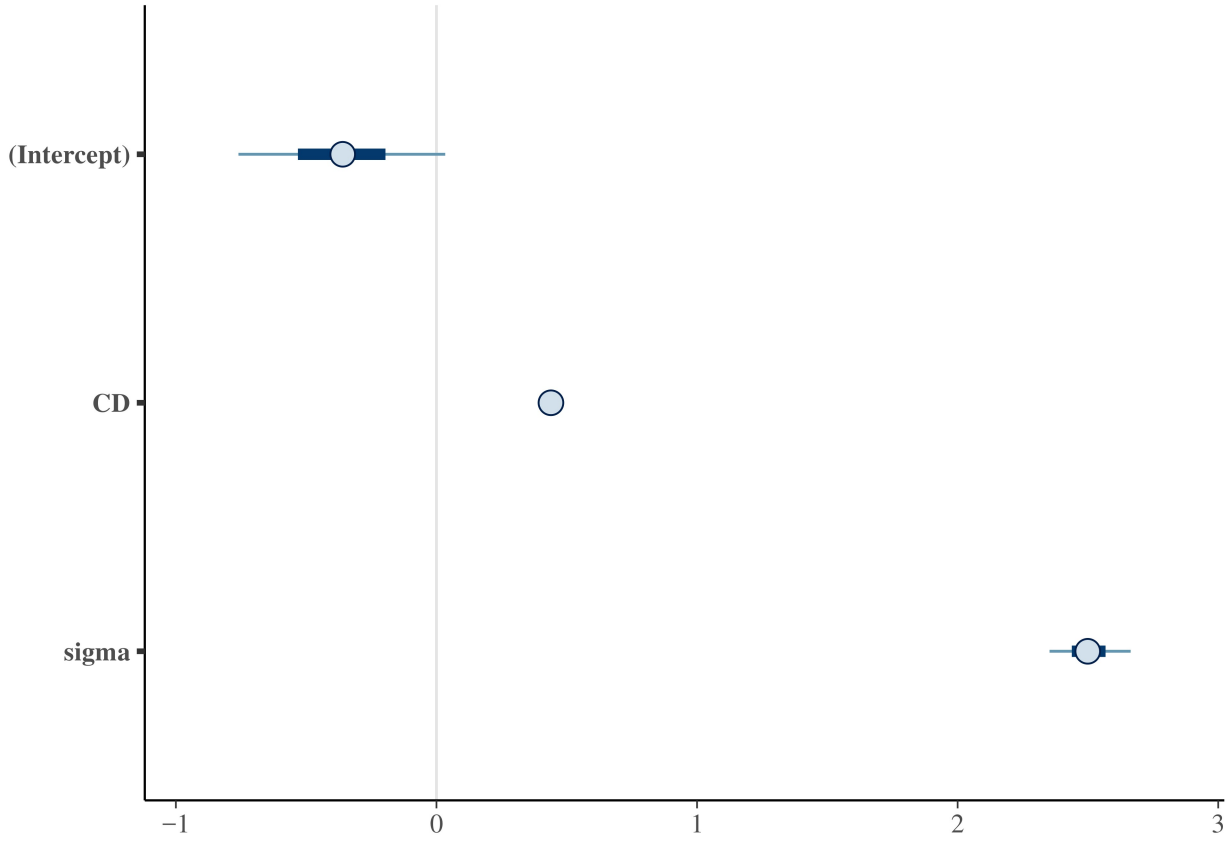
```
posterior_interval(stan.fit, prob=0.95)
```

```
##                   2.5%     97.5%
## (Intercept) -0.8265199 0.1102706
## CD           0.4189777 0.4610868
## sigma        2.3254675 2.7016222
```

```
posterior_interval(stan.fit, prob = 0.66)
```

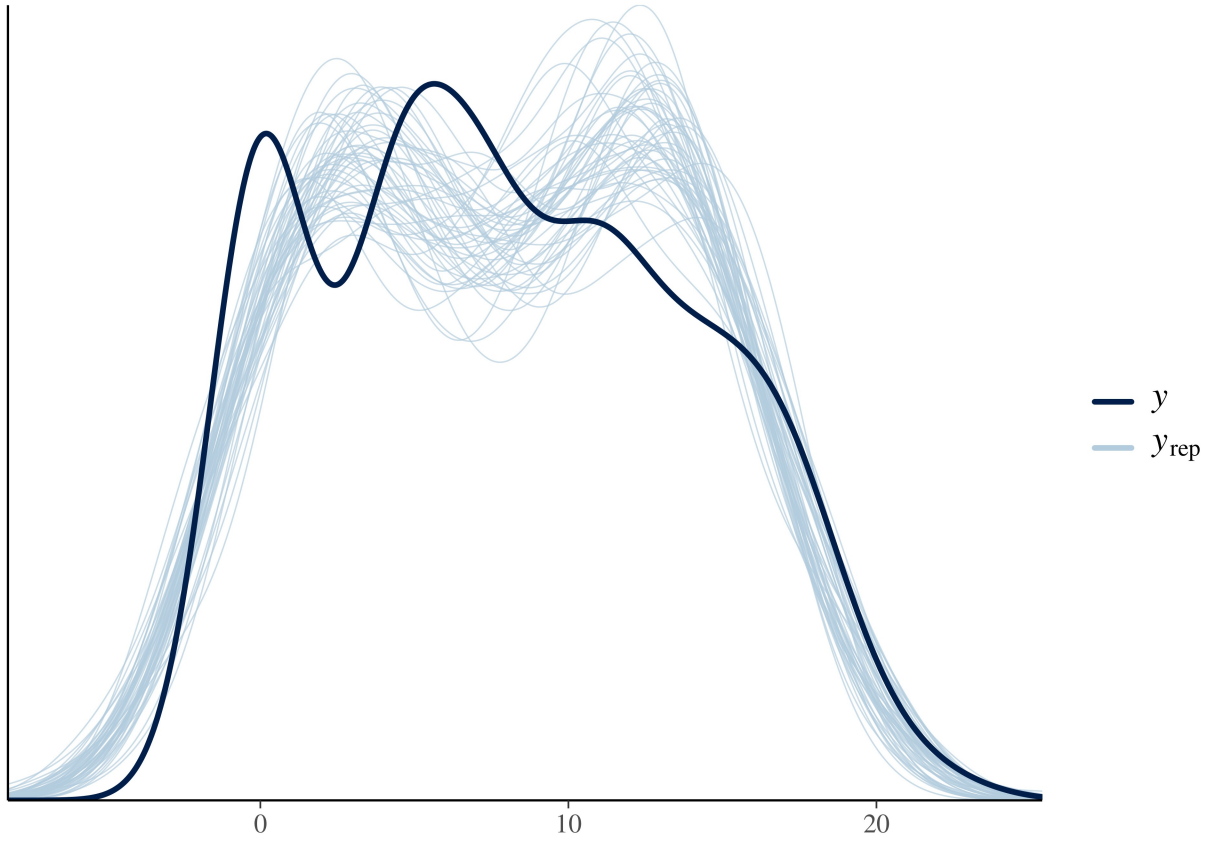
```
##                    17%        83%
## (Intercept) -0.6019096 -0.1260501
## CD           0.4290449  0.4501402
## sigma        2.4128374  2.5952147
```

```
plot(stan.fit)
```



Predictive Power Visualization

```
pp_check(stan.fit)
```




```
#Note: Bayesians view parameters as being random variables in a distribution.
#Actual CD values have peaks at 0 and less than 10. Model tends to predictt
#both peaks as being greater than they actually are.
```

```
help("pp_check")
```

```
## starting httpd help server ... done
```

Conclusion

The data supports the claim that Raman Spectroscopy is an effective way to measure fractional abundance. Low training error for the linear model when predicting nanoSIMS using Raman indicate that, on average, Raman yields similar results. Linear modeling proved to be better than Lasso or Ridge approaches. Bayesian Credible Interval indicates there is a 95% chance the S.E. is between 2.3 and 2.7.