regression

Anandu R

9/21/2020

Loading the galton data

```
library(UsingR)
data("galton")
freqData <- as.data.frame(table(galton$child, galton$parent))
names(freqData) <- c("child", "parent", "freq")
freqData = mutate(freqData, child = as.numeric(as.vector(child)), parent = as.numeric(as.vector(parent))
```

The data contains the heights of parents and child (paired). We try and estimate the child's height(y) using the height of the parents(x).

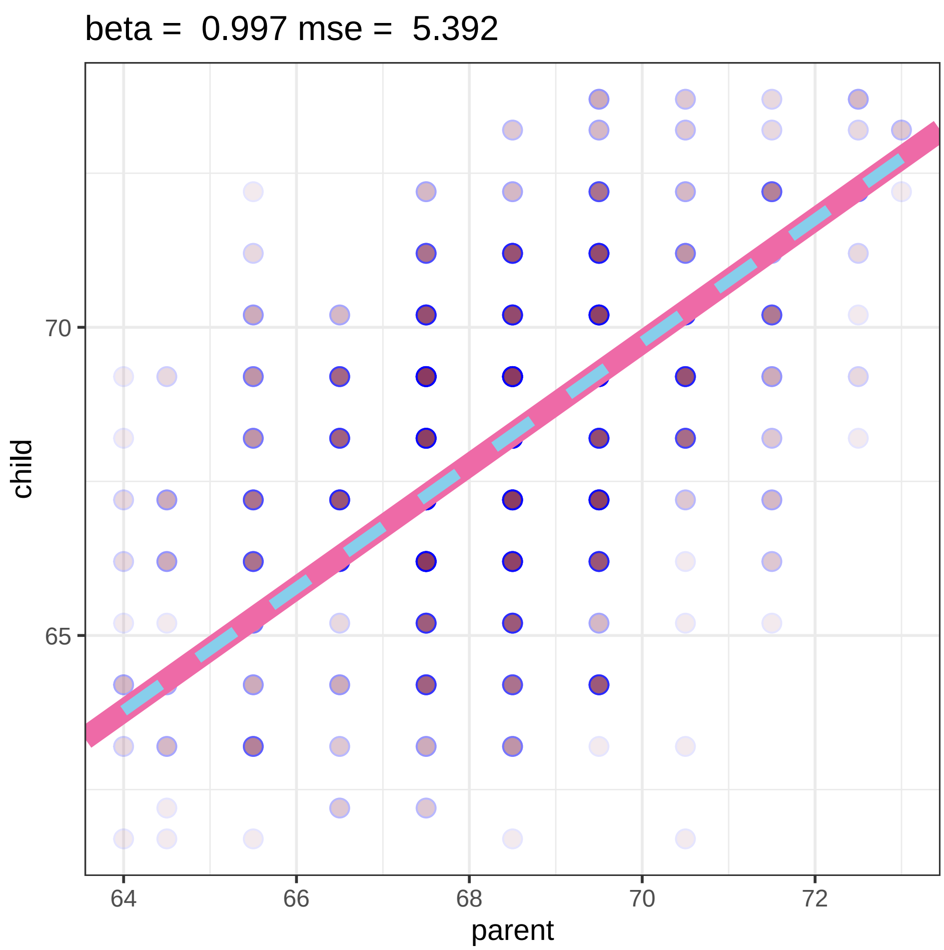
Performing regression analysis without the y-intercept

With y-intercept 0 or in other words without a y-intercept, but this doesn't result in the best line to fit the data,

```
Beta = as.numeric(coef(lm(galton$child ~ galton$parent - 1)))
angle = round(atan(Beta)*180/3.14,2)
mse = mean( (galton$child - Beta * galton$parent)^2 )
```

Using the lm function we find the slope of line to be 0.997, and that the line that best fits the data originating from the slope is angled at 44.92 degrees. Using this value of Beta we can manually find the regression line that fits the data and the mean square error is found to be 5.392

```
ggplot(
  galton,
  aes(
    x = parent,
    y = child
  )
) + geom_point(pch = 21, col = "blue", bg = "hotpink4",
               cex = 3, alpha = 0.1) +
  labs(x = "parent", y = "child") +
  geom_abline(slope = Beta, lwd = 4, col = "hotpink2") +
  geom_point(aes(x = 0, y = 0), cex = 2, pch = 21) +
  geom_smooth(method = "lm", formula = y~x-1, se = F, col = "skyblue", lwd = 2, lty = 2) +
  ggtitle(paste("beta = ", round(Beta,3), "mse = ", round(mse, 3))) +
  theme_bw() +
  coord_cartesian(xlim = range(galton$parent), ylim = range(galton$child))
```



We see that the broken blue lines representing the regression line computed using the lm() function overlaps with the pink line drawn using the Beta coefficient estimated earlier.

Regression with centered data

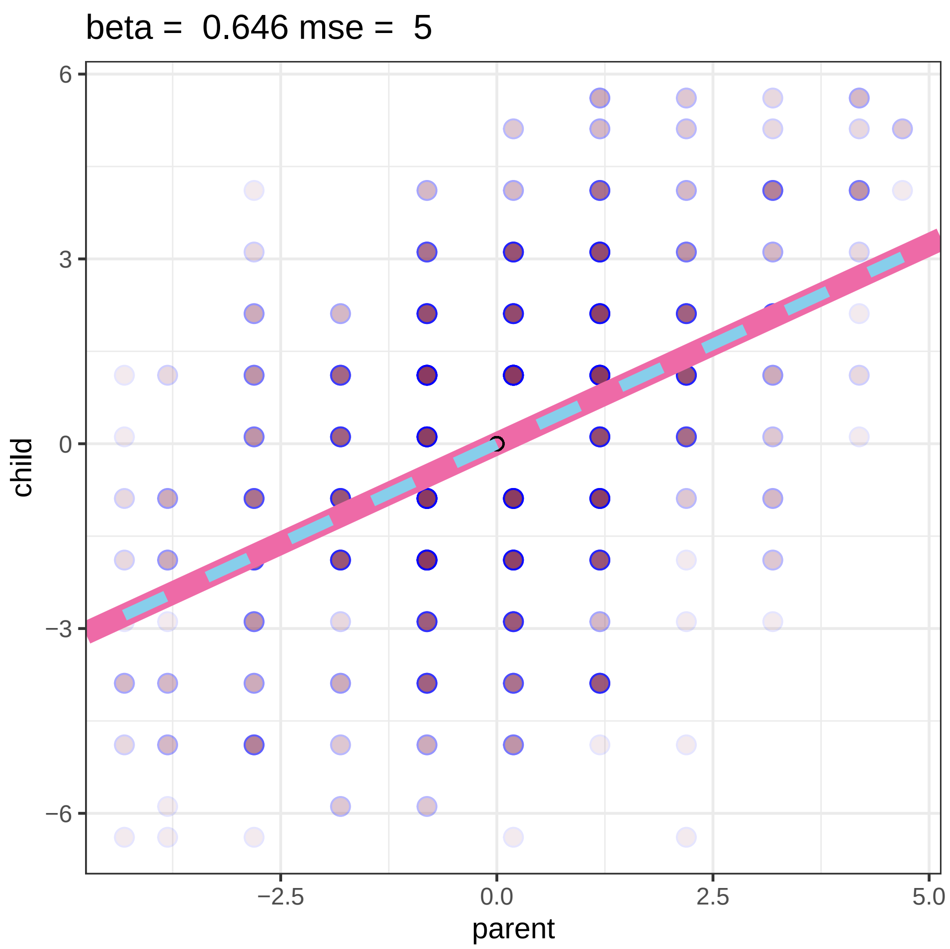
If we were to centre the data around the mean and perform the same analysis, we'd get the same result as we'd if the intercept estimator β_0 was under consideration

```
galtonCentered = mutate(galton, child = child - mean(child), parent = parent - mean(parent))
Beta = as.numeric(coef(lm(galtonCentered$child ~ galtonCentered$parent - 1)))
angle = round(atan(Beta)*180/3.14,2)
mse = mean( (galtonCentered$child - Beta * galtonCentered$parent)^2 )
```

Using the lm function we find the slope of line to be 0.646, and that the line that best fits the data originating from the slope is angled at 32.89 degrees. Using this value of Beta we can manually find the regression line that fits the data and the mean square error is found to be 5

```
ggplot(
  galtonCentered,
  aes(
    x = parent,
    y = child
```

```
  )
) + geom_point(pch = 21, col = "blue", bg = "hotpink4",
               cex = 3, alpha = 0.1) +
  labs(x = "parent", y = "child") +
  geom_abline(slope = Beta, lwd = 4, col = "hotpink2") +
  geom_point(aes(x = 0, y = 0), cex = 2, pch = 21) +
  geom_smooth(method = "lm", formula = y~x-1, se = F, col = "skyblue", lwd = 2, lty = 2) +
  ggtitle(paste("beta = ", round(Beta,3), "mse = ", round(mse, 3))) +
  theme_bw() +
  coord_cartesian(xlim = range(galtonCentered$parent), ylim = range(galtonCentered$child))
```



We see that the broken blue lines representing the regression line computed using the lm() function overlaps with the pink line drawn using the Beta coefficient estimated earlier.

Regression with normalized data

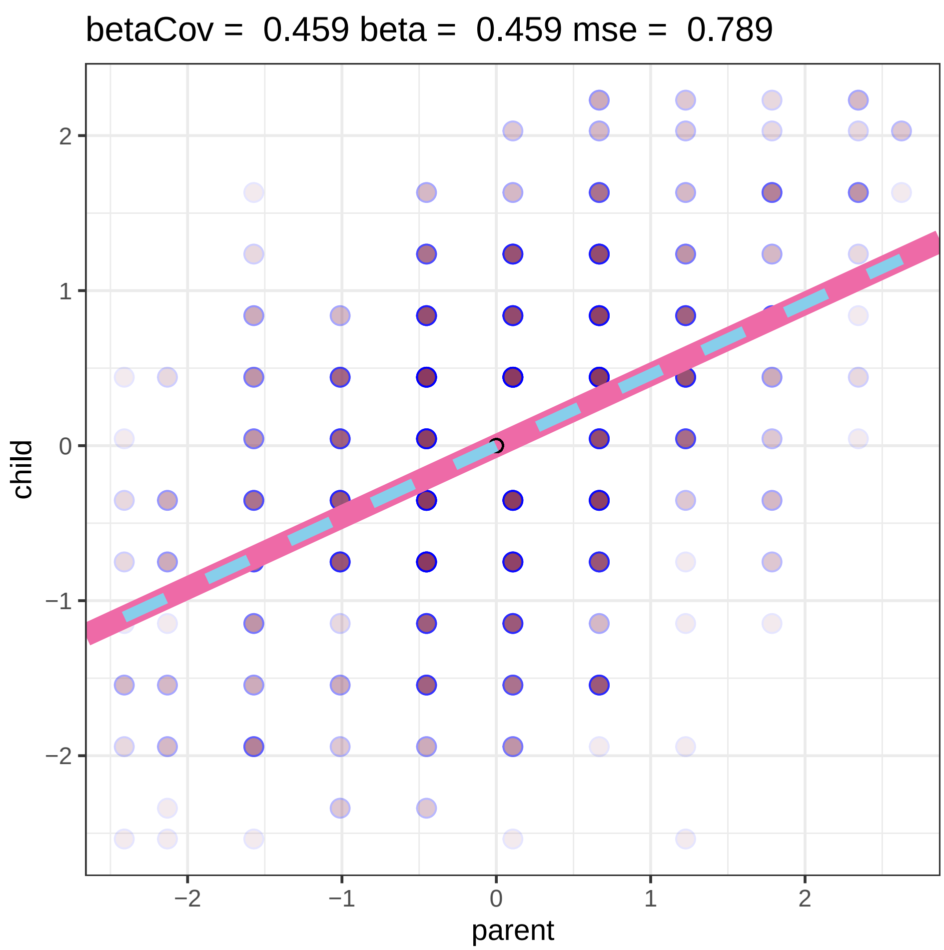
If we were to normalize the data and perform the same analysis, we'd get the same result still, furthmore, the β coefficient can be calculated as the covariance of X and Y.

```
galtonNormalized = mutate(galton, child = (child - mean(child))/sd(child), parent = (parent - mean(paren
Beta = as.numeric(coef(lm(galtonNormalized$child ~ galtonNormalized$parent - 1)))
BetaCov = cor(galtonNormalized$child, galtonNormalized$parent)
```

```r
angle = round(atan(Beta)*180/3.14,2)
mse = mean( (galtonNormalized$child - Beta * galtonNormalized$parent)^2 )
```

Using the lm function we find the slope of line to be 0.459, and that the line that best fits the data originating from the slope is angled at 24.66 degrees. Using this value of Beta we can manually find the regression line that fits the data and the mean square error is found to be 0.789

```r
ggplot(
  galtonNormalized,
  aes(
    x = parent,
    y = child
  )
) + geom_point(pch = 21, col = "blue", bg = "hotpink4",
               cex = 3, alpha = 0.1) +
  labs(x = "parent", y = "child") +
  geom_abline(slope = Beta, lwd = 4, col = "hotpink2") +
  geom_point(aes(x = 0, y = 0), cex = 2, pch = 21) +
  geom_smooth(method = "lm", formula = y~x-1, se = F, col = "skyblue", lwd = 2, lty = 2) +
  ggtitle(paste("betaCov = ", round(BetaCov,3),"beta = ", round(Beta,3), "mse = ", round(mse, 3))) +
  theme_bw() +
  coord_cartesian(xlim = range(galtonNormalized$parent), ylim = range(galtonNormalized$child))
```



We get the exact same result in this case as well.

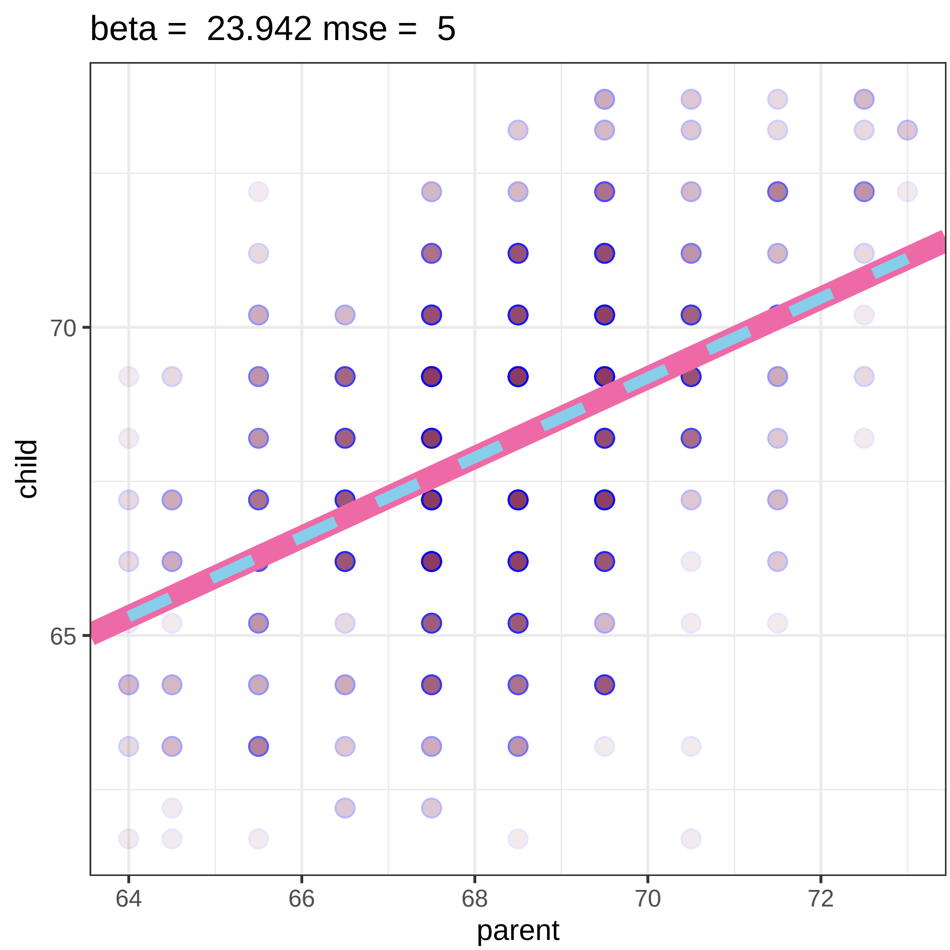
Performing regression analysis with the y-intercept

Complete regression line using the y-intercept

```
Beta = as.numeric(coef(lm(galton$child ~ galton$parent)))
Beta0 = Beta[1]
Beta1 = Beta[2]
angle = round(atan(Beta1)*180/3.14,2)
mse = mean( (galton$child - (Beta0 + Beta1 * galton$parent))^2 )
```

Using the lm function we find the slope of line to be 23.942, 0.646, and that the line that best fits the data originating from the slope is angled at 32.89 degrees. Using this value of Beta we can manually find the regression line that fits the data and the mean square error is found to be 5

```
ggplot(
  galton,
  aes(
    x = parent,
    y = child
  )
) + geom_point(pch = 21, col = "blue", bg = "hotpink4",
               cex = 3, alpha = 0.1) +
  labs(x = "parent", y = "child") +
  geom_abline(intercept = Beta0,slope = Beta1, lwd = 4, col = "hotpink2") +
  geom_point(aes(x = 0, y = 0), cex = 2, pch = 21) +
  geom_smooth(method = "lm", formula = y~x, se = F, col = "skyblue", lwd = 2, lty = 2) +
  ggtitle(paste("beta = ", round(Beta,3), "mse = ", round(mse, 3))) +
  theme_bw() +
  coord_cartesian(xlim = range(galton$parent), ylim = range(galton$child))
```



We get the exact same result as seen in the normalized data regression above.

Residuals

Consider the diamong dataset imported from the usingR package in R. We can plot the scatterplot of x vs y and fit a regression line through them.

Here the residuals of a fitted line is a vector of numbers centered around zero

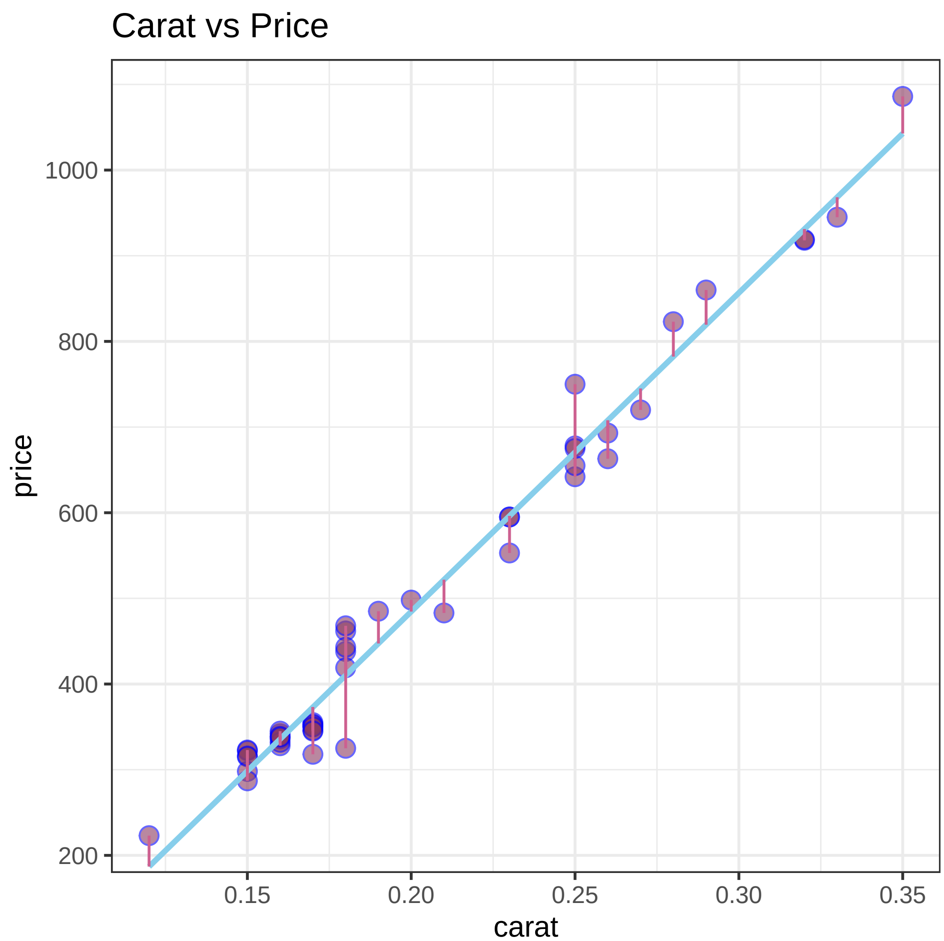
```
y = diamond$price
x = diamond$carat
n = length(y)
fit = lm(y~x)
e = resid(fit) # gives vector of residuals which is part of the lm object referenced here as "fit"
yhat = predict(fit) # since no new data given yhat here will be the points on the fitted regression lin
print(max(abs(e-(y-yhat))))
```

```
## [1] 9.485746e-13
```

```
print(max(abs(e - (y - fit$coefficients[1] - fit$coefficients[2]*x))))
```

```
## [1] 9.485746e-13
```

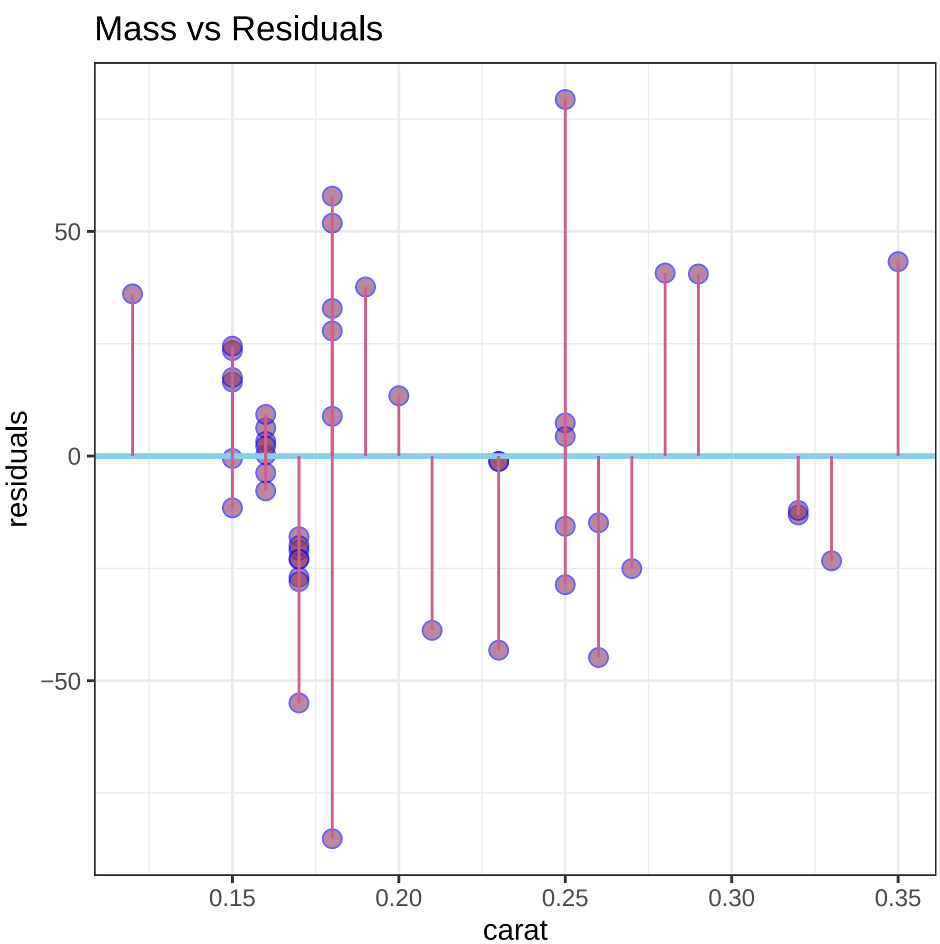
```r
ggplot(
  diamond,
  aes(
    x = carat,
    y = price
  )
) + geom_point(pch = 21, col = "blue", bg = "hotpink4",
               cex = 3, alpha = 0.6) +
  labs(x = "carat", y = "price") +
  geom_smooth(method = "lm", formula = y~x, se = F, col = "skyblue", lwd = 1, lty = 1) +
  ggtitle("Carat vs Price") +
  theme_bw() +
  coord_cartesian(xlim = range(diamond$carat), ylim = range(diamond$price)) +
  geom_segment(aes(x=x,y=y,xend=x,yend=yhat),col = "hotpink3")
```



The above visualization isn't particularly useful for assessing the residual variation, a better alternative would be to plot the residuals on the verticle axis and independent variable on the horizontal axis.

```r
temp_data = data.frame(x = diamond$carat, y = e)
ggplot(
  temp_data,
  aes(
```

```r
    x = x,
    y = y
  )
) + geom_point(pch = 21, col = "blue", bg = "hotpink4",
               cex = 3, alpha = 0.6) +
  labs(x = "carat", y = "residuals") +
  ggtitle("Mass vs Residuals") +
  theme_bw() +
  geom_hline(aes(yintercept=0),col = "skyblue", lwd=1) +
  coord_cartesian(xlim = range(temp_data$x), ylim = range(temp_data$y)) +
  geom_segment(aes(x=x,y=0,xend=x,yend=y),col = "hotpink3")
```



We look for any sort of patterns in the residual plot, ideally a residual plot should nothave any patterns.

Illustration of inference on regression

Illustration to show the inference on the regression line is identical to the output summary statistics generated by the the lm function.

```r
library(UsingR); data(diamond)
y <- diamond$price; x <- diamond$carat; n <- length(y)
beta1 <- cor(y, x) * sd(y) / sd(x)
```

```r
beta0 <- mean(y) - beta1 * mean(x)
e <- y - beta0 - beta1 * x
sigma <- sqrt(sum(e^2) / (n-2))
ssx <- sum((x - mean(x))^2)
seBeta0 <- (1 / n + mean(x) ^ 2 / ssx) ^ .5 * sigma
seBeta1 <- sigma / sqrt(ssx)
tBeta0 <- beta0 / seBeta0; tBeta1 <- beta1 / seBeta1
pBeta0 <- 2 * pt(abs(tBeta0), df = n - 2, lower.tail = FALSE)
pBeta1 <- 2 * pt(abs(tBeta1), df = n - 2, lower.tail = FALSE)
coefTable <- rbind(c(beta0, seBeta0, tBeta0, pBeta0), c(beta1, seBeta1, tBeta1, pBeta1))
colnames(coefTable) <- c("Estimate", "Std. Error", "t value", "P(>|t|)")
rownames(coefTable) <- c("(Intercept)", "x")
print(coefTable)
```

```
##              Estimate Std. Error   t value      P(>|t|)
## (Intercept) -259.6259   17.31886 -14.99094 2.523271e-19
## x           3721.0249   81.78588  45.49715 6.751260e-40
```

```r
fit <- lm(y ~ x);
print(summary(fit)$coefficients)
```

```
##              Estimate Std. Error   t value     Pr(>|t|)
## (Intercept) -259.6259   17.31886 -14.99094 2.523271e-19
## x           3721.0249   81.78588  45.49715 6.751260e-40
```

The confidence interval of the slope and the intercept of the regression line can be calculated as

```r
sumCoef <- summary(fit)$coefficients
sumCoef[1,1] + c(-1, 1) * qt(.975, df = fit$df) * sumCoef[1, 2]
```

```
## [1] -294.4870 -224.7649
```

```r
(sumCoef[2,1] + c(-1, 1) * qt(.975, df = fit$df) * sumCoef[2, 2]) / 10
```

```
## [1] 355.6398 388.5651
```

Which tells us that - "With a 95% confidence interval a 0.1 increase in the carat size we see a resultant increase in price in the range 356 to 389.

Multivariable regression

Example 1

Function to replace a variable in dataset with the residuals of the variable w.r.t the predictor.

```r
# Regress the given variable on the given predictor,
# suppressing the intercept, and return the residual.
regressOneOnOne <- function(predictor, other, dataframe){
  # Point A. Create a formula such as Girth ~ Height -1
```

```r
  formula <- paste0(other, " ~ ", predictor, " - 1")
  # Use the formula in a regression and return the residual.
  resid(lm(formula, dataframe))
}

# Eliminate the specified predictor from the dataframe by
# regressing all other variables on that predictor
# and returning a data frame containing the residuals
# of those regressions.
eliminate <- function(predictor, dataframe){
  # Find the names of all columns except the predictor.
  others <- setdiff(names(dataframe), predictor)
  # Calculate the residuals of each when regressed against the given predictor
  temp <- sapply(others, function(other)regressOneOnOne(predictor, other, dataframe))
  # sapply returns a matrix of residuals; convert to a data frame and return.
  as.data.frame(temp)
}
```

We can use this function to reduce a multivariable regression in n variable to regression in one variable.

Using the trees dataset which can be used to fit a model to compute the volume of a tree given the girth and height of the tree.

```r
head(trees)
```

```
##   Girth Height Volume
## 1   8.3     70   10.3
## 2   8.6     65   10.3
## 3   8.8     63   10.2
## 4  10.5     72   16.4
## 5  10.7     81   18.8
## 6  10.8     83   19.7
```

A residual is the difference between a variable and its predicted value.

```r
fit <- lm(Volume ~ ., trees)
summary(fit)
```

```
## 
## Call:
## lm(formula = Volume ~ ., data = trees)
## 
## Residuals:
##     Min      1Q  Median      3Q     Max 
## -6.4065 -2.6493 -0.2876  2.2003  8.4847 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) -57.9877     8.6382  -6.713 2.75e-07 ***
## Girth         4.7082     0.2643  17.816  < 2e-16 ***
## Height        0.3393     0.1302   2.607   0.0145 *  
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.882 on 28 degrees of freedom
## Multiple R-squared:  0.948,  Adjusted R-squared:  0.9442
## F-statistic:   255 on 2 and 28 DF,  p-value: < 2.2e-16
```

Adding constant column

```
trees$Constant=1
```

Eliminating the "Girth" variable.

```
trees2 <- eliminate("Girth", trees)
head(trees2)
```

```
##     Height     Volume  Constant
## 1 24.38809  -9.793826 0.4057735
## 2 17.73947 -10.520109 0.3842954
## 3 14.64038 -11.104298 0.3699767
## 4 14.29818  -9.019900 0.2482677
## 5 22.19910  -7.104089 0.2339490
## 6 23.64956  -6.446183 0.2267896
```

This is regression in many variables amounts to a series of regressions in one.

```
fit2 <- lm(Volume ~ . - 1, trees2)
lapply(list(fit,fit2),coef)
```

```
## [[1]]
## (Intercept)       Girth      Height
## -57.9876589   4.7081605   0.3392512
##
## [[2]]
##      Height    Constant
##   0.3392512 -57.9876589
```

Example 2

Using the mtcars dataset, fitting a linear model to estimate the mpg given cyl count, weight of vehicle wt, and performance in horsepower hp.

```
data = select(mtcars,wt,cyl,hp,mpg)
rownames(data) <- NULL
data$const = 1
head(data)
```

```
##      wt cyl  hp  mpg const
## 1 2.620   6 110 21.0     1
## 2 2.875   6 110 21.0     1
## 3 2.320   4  93 22.8     1
## 4 3.215   6 110 21.4     1
## 5 3.440   8 175 18.7     1
## 6 3.460   6 105 18.1     1
```

Fitting a linear model on the original dataset with all 4 parameters (including the constant parameter 1 which estimates the mean for the outcome)

```
fit = lm(mpg~.-1,data)
fit
```

```
##
## Call:
## lm(formula = mpg ~ . - 1, data = data)
##
## Coefficients:
##       wt       cyl        hp     const
## -3.16697  -0.94162  -0.01804  38.75179
```

Removing the wt parameter by regressing all other parameters with the wt parameter to predict the values of other parameters which estimates the wt

```
const = as.numeric(resid(lm(const~wt-1,data)))
hp = as.numeric(resid(lm(hp~wt-1,data)))
cyl = as.numeric(resid(lm(cyl~wt-1,data)))
mpg = as.numeric(resid(lm(mpg~wt-1,data)))
data2 = data.frame(const,hp,cyl,mpg)
head(data2)
```

```
##        const         hp         cyl        mpg
## 1 0.25260892  -9.578391  1.06779384  7.1359449
## 2 0.17986666 -21.216746  0.58775088  5.7865807
## 3 0.33818805 -12.886209 -0.36744973 10.5234321
## 4 0.08287698 -36.734552 -0.05230641  4.3874285
## 5 0.01869263  17.996311  1.52412627  0.4968131
## 6 0.01298735 -52.916501 -0.51352416 -0.2090194
```

Fitting a linear model on dataset with wt parameter removed

```
fit2 = lm(mpg~.-1,data2)
```

Comparing the prediction of each models

```
paste("Pred with wt: ",as.character(predict(fit, newdata = data.frame(wt=mean(data$wt),cyl=median(data$
```

```
## [1] "Pred with wt:  4747.98523764184"
```

```
paste("pred without wt: ",as.character(predict(fit2, newdata = data.frame(cyl=median(data$cyl),hp=mean(
```

```
## [1] "pred without wt:  4758.1741818824"
```

Example 3

Now, considering a new subset of original dataset without the constant column,
Regressing hp on cyl, wt, and also regressing on the response mpg

```r
# The intercept constant 1s are by default included
data = select(mtcars,wt,cyl,hp,mpg)
ex = as.numeric(resid(lm(hp~wt+cyl,data)))
ey = as.numeric(resid(lm(mpg~wt+cyl,data)))
```

ex represents the values of hp regressed on wt, cyl and column of 1s.
ey represents the values of mpg regressed on wt, cyl and column of 1s.
Since it is a linear model, the property of linearity holds - we can regress ey on ex(minus the column of 1s - since we already regressed both ex and ey on 1 already) to get same response as we would get if we had regressed mpg on column of 1s, wt, cyl and hp.

```r
fit1 = lm(ey~ex-1)
fit2 = lm(mpg~.,data)
x=round(as.numeric(summary(fit1)$resid),13);y=round(as.numeric(summary(fit2)$resid),13)
identical(x,y)
```

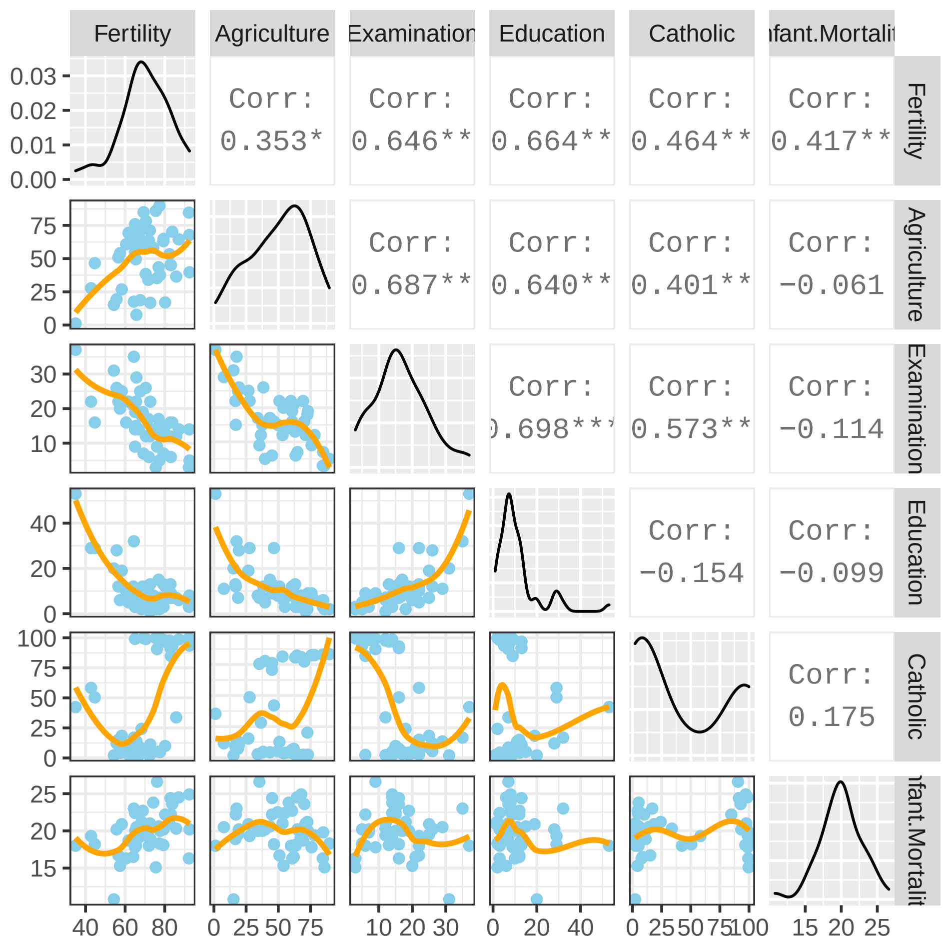
```
## [1] TRUE
```

The values of residuals are practically identical up to 13 decimal numeric precision.

Swiss dataset

Summarizing the variables in the dataset

```r
require(datasets); data(swiss); require(GGally); require(ggplot2)
smooth_plot_fn = function(data, mapping, method="loess",...){
  g = ggplot(
    data = data,
    mapping = mapping
  ) + theme_bw() +
    geom_point(col = "skyblue") +
    geom_smooth(method = method, formula = y~x, ...)
  g
}
ggpairs(swiss, lower = list(continuous=wrap(smooth_plot_fn,se=F,col="orange")), progress = F)
```



Regression on all the variables for illustration purpose

```r
summary(lm(Fertility~.,swiss))$coefficients
```

```
##                    Estimate  Std. Error   t value     Pr(>|t|)
## (Intercept)      66.9151817 10.70603759  6.250229 1.906051e-07
## Agriculture      -0.1721140  0.07030392 -2.448142 1.872715e-02
## Examination      -0.2580082  0.25387820 -1.016268 3.154617e-01
## Education        -0.8709401  0.18302860 -4.758492 2.430605e-05
## Catholic          0.1041153  0.03525785  2.952969 5.190079e-03
## Infant.Mortality  1.0770481  0.38171965  2.821568 7.335715e-03
```

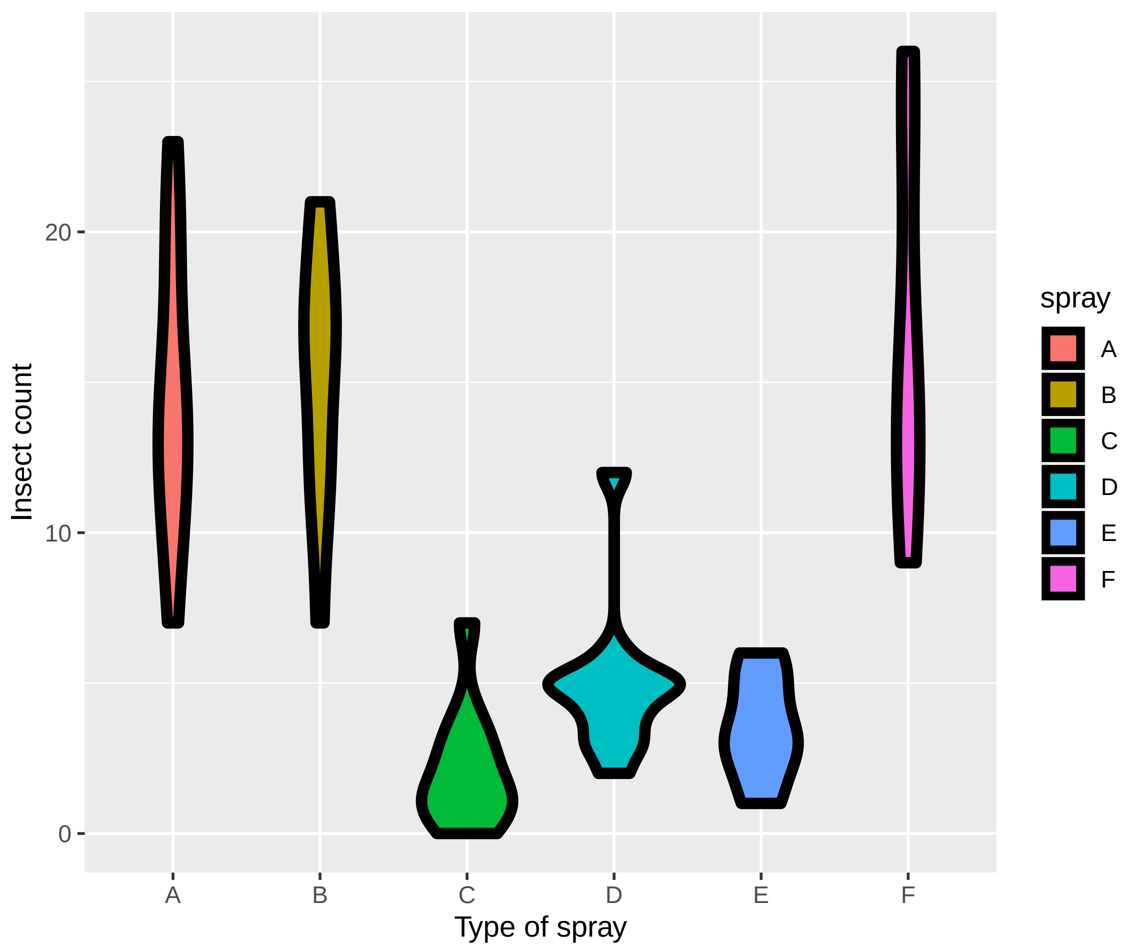
This tells us thatL * There is an expected 0.17 decrease in fertility for unit increase (1% increase in percentage of males) in agriculture, holding the remaining variables constant.
* The Std. Error tells us how precise this observation is.
* t values is the t statistic obtained by dividing the Estimate by the Std. Error terms, used for hypothesis testing.

Regression on factor variables

when dealing with factor variables R by default sets a level as the default value and it is interpreted by the intercept coefficient, whereas all the other levels are interpreted as a comparison of it's level to the default

level.
If we are to subtract the intercept coefficient from a given one we get the relation of this coefficient with the outcome minus the default.

```
require(datasets);data(InsectSprays); require(stats); require(ggplot2)
ggplot(
  data = InsectSprays,
  aes(
    y = count,
    x = spray,
    fill  = spray
  )
) +
  geom_violin(colour = "black", size = 2) +
  xlab("Type of spray") +
  ylab("Insect count")
```



With spray group A as the reference the linear model coefficients are

```
summary(lm(count ~ spray, data = InsectSprays))$coef
```

```
##                  Estimate Std. Error    t value     Pr(>|t|)
```

```
## (Intercept)  14.5000000   1.132156 12.8074279 1.470512e-19
## sprayB        0.8333333   1.601110  0.5204724 6.044761e-01
## sprayC      -12.4166667   1.601110 -7.7550382 7.266893e-11
## sprayD       -9.5833333   1.601110 -5.9854322 9.816910e-08
## sprayE      -11.0000000   1.601110 -6.8702352 2.753922e-09
## sprayF        2.1666667   1.601110  1.3532281 1.805998e-01
```

This can be used to generate relevant inferences such as whether another spray is infact different from the reference spray using the t-statistic.
And so we can test whether spray A is more effective than the others.
To interpret the coeffients exactly as the means of the factors remove the x-intercept, as in

```
summary(lm(count ~ spray-1, data = InsectSprays))$coef
```

```
##         Estimate Std. Error   t value     Pr(>|t|)
## sprayA 14.500000   1.132156 12.807428 1.470512e-19
## sprayB 15.333333   1.132156 13.543487 1.001994e-20
## sprayC  2.083333   1.132156  1.840148 7.024334e-02
## sprayD  4.916667   1.132156  4.342749 4.953047e-05
## sprayE  3.500000   1.132156  3.091448 2.916794e-03
## sprayF 16.666667   1.132156 14.721181 1.573471e-22
```

The t-statistic in this case is telling us whether the mean is further away from zero i.e. whether a spray killed any insect and not with respect to how many a reference spray was able to kill.

To change the reference level we use the relevel function.

```
spray2 = relevel(InsectSprays$spray, "C")
summary(lm(count ~ spray2, data = InsectSprays))$coef
```

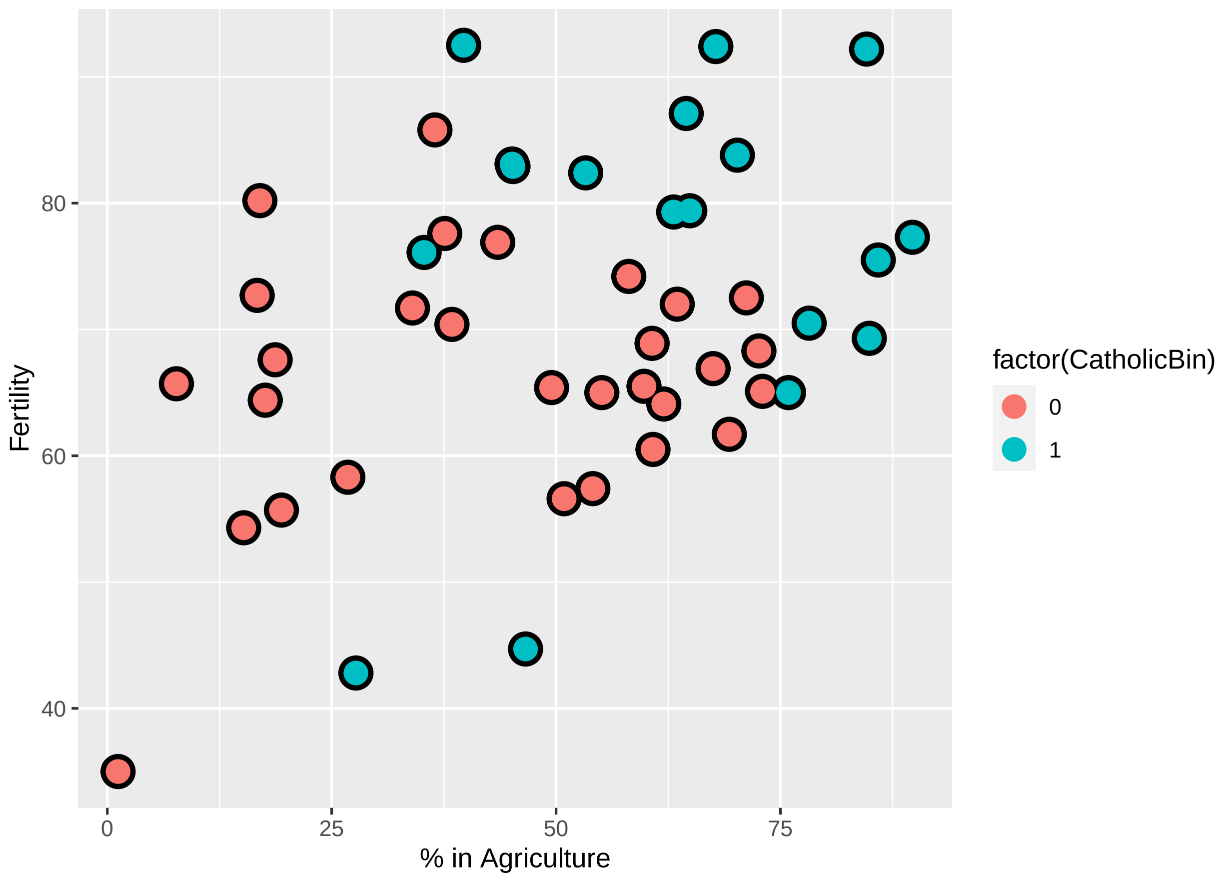
```
##              Estimate Std. Error  t value     Pr(>|t|)
## (Intercept)  2.083333   1.132156 1.840148 7.024334e-02
## spray2A     12.416667   1.601110 7.755038 7.266893e-11
## spray2B     13.250000   1.601110 8.275511 8.509776e-12
## spray2D      2.833333   1.601110 1.769606 8.141205e-02
## spray2E      1.416667   1.601110 0.884803 3.794750e-01
## spray2F     14.583333   1.601110 9.108266 2.794343e-13
```

Illustration of regression with a continous and a factor variable

Continous variable - Agriculture
Factor variable - CatholicBin (Whether >0.5 percent in a locality catholic or not)

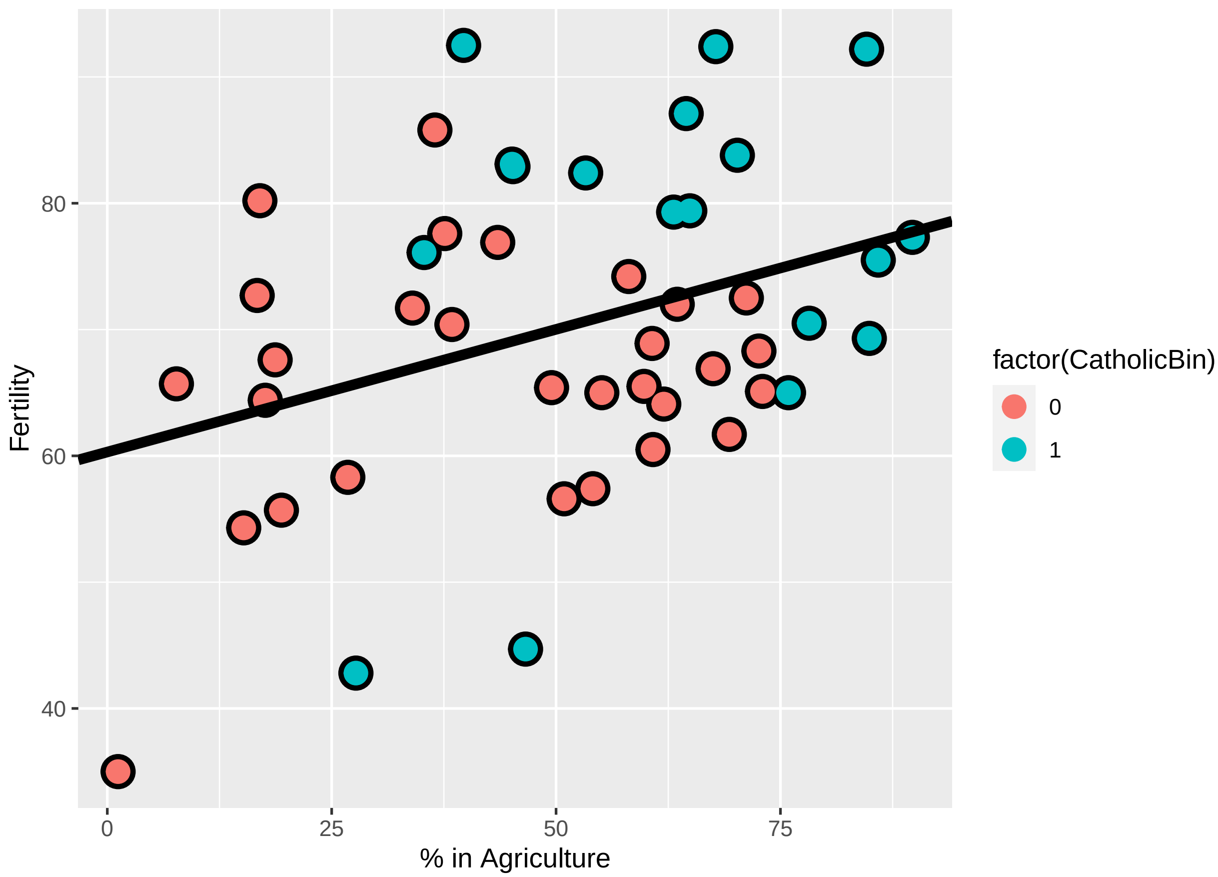
```
swiss = mutate(swiss, CatholicBin = 1 * (Catholic > 50))
g = ggplot(
  swiss,
  aes(
    x = Agriculture,
    y = Fertility,
    colour = factor(CatholicBin)
    )
  ) + geom_point(size = 6, colour = "black") +
```

```
  geom_point(size = 4) +
  xlab("% in Agriculture") +
  ylab("Fertility")
g
```



When disregarding the factor variable - Baseline

```
fit = lm(Fertility ~ Agriculture, data = swiss)
g + geom_abline(intercept = coef(fit)[1], slope = coef(fit)[2], size = 2)
```



Intercept = 60.3043752
Slope = 0.1942017

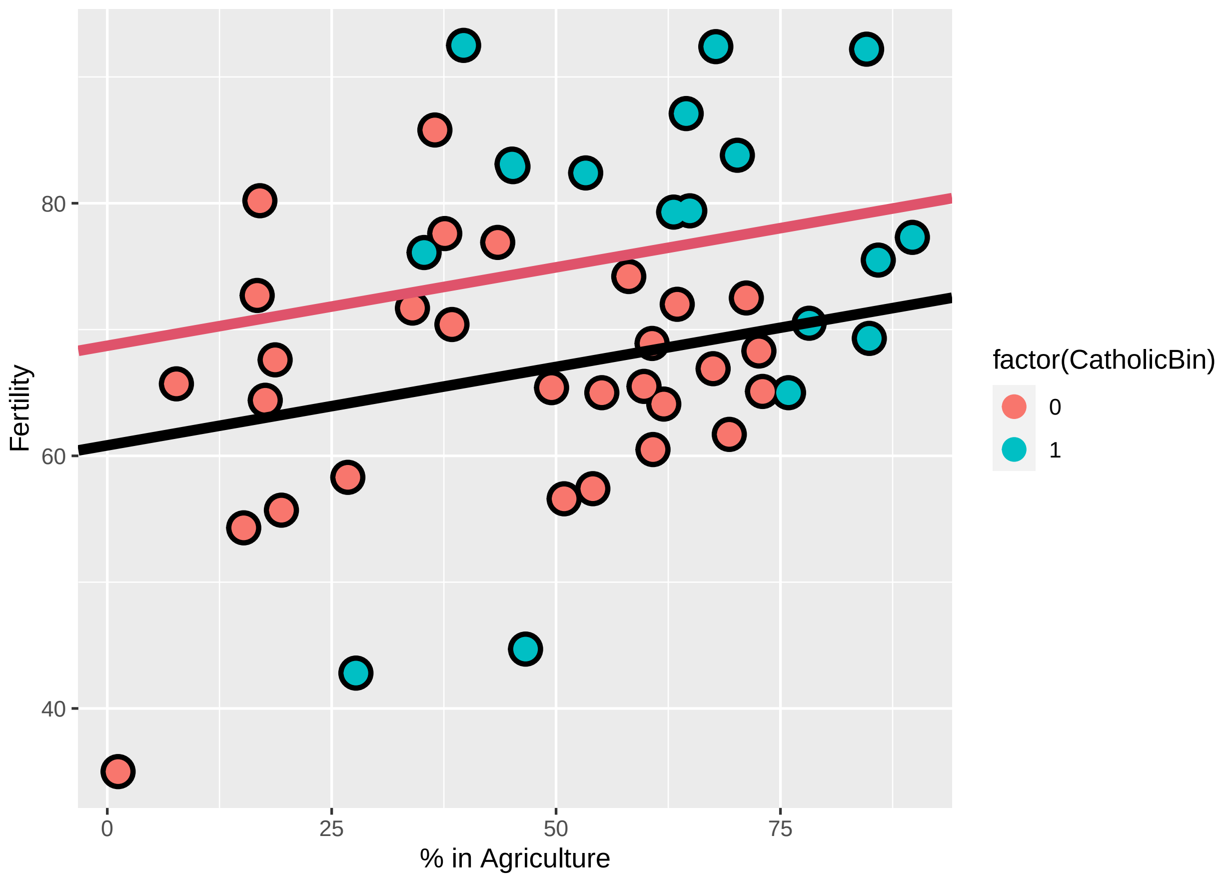
Using factor - Scenario 1 - Only changing the intercept

Adding the factor variable creates just 3 coefficients

```
fit = lm(Fertility ~ Agriculture + factor(CatholicBin), data = swiss)
summary(fit)$coef
```

```
##                        Estimate Std. Error   t value     Pr(>|t|)
## (Intercept)          60.8322366  4.1058630 14.815944 1.032493e-18
## Agriculture           0.1241776  0.0810977  1.531210 1.328763e-01
## factor(CatholicBin)1  7.8843292  3.7483622  2.103406 4.118221e-02
```

```
g + geom_abline(intercept = coef(fit)[1], slope = coef(fit)[2], size = 2) +
  geom_abline(intercept = coef(fit)[1] + coef(fit)[3], slope = coef(fit)[2], size = 2, col = 2)
```



Intercept = 68.7165659
Slope = 0.1241776

Using factor - Scenario 2 - Changing slope and intercept

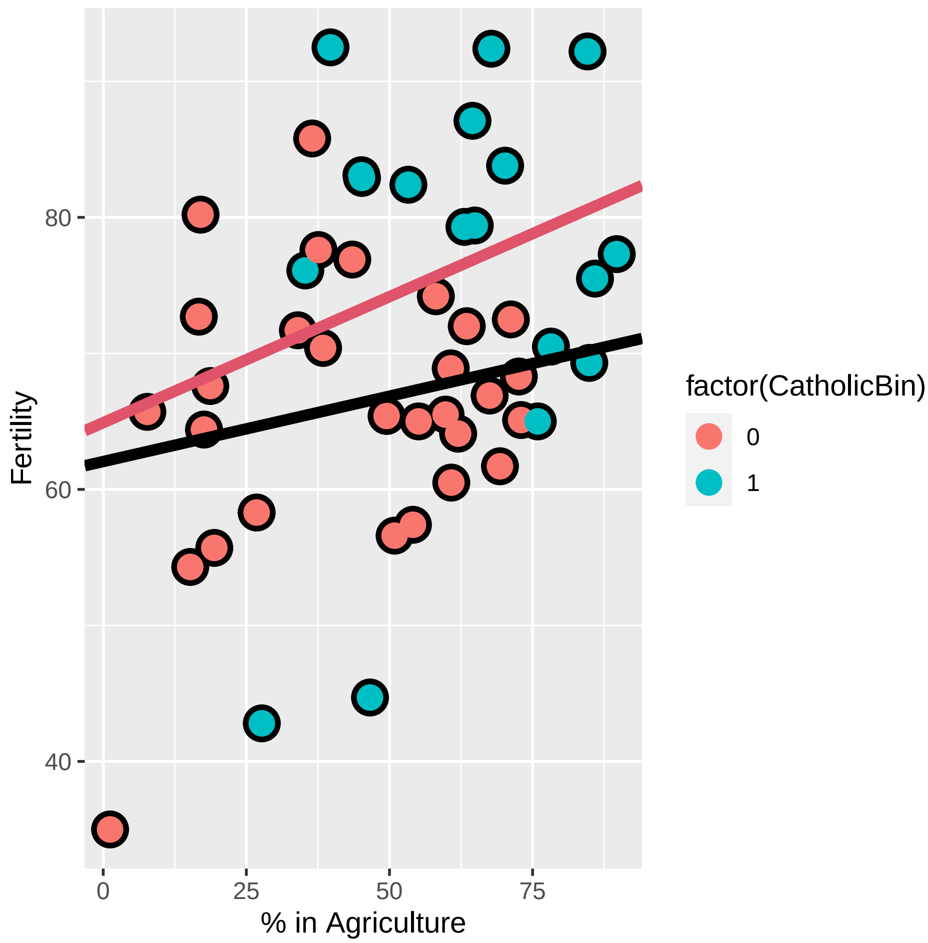
When using 'variable * factor', R creates coefficients of variables in multiples of factor levels.
In this case the factor has two levels, and by default the variable would have 2 coefficients the slope and the intercep, when using the *, there will be 4 coefficients.

```
fit = lm(Fertility ~ Agriculture * factor(CatholicBin), data = swiss)
summary(fit)$coef
```

```
##                                     Estimate  Std. Error    t value
## (Intercept)                      62.04993019  4.78915566 12.9563402
## Agriculture                       0.09611572  0.09881204  0.9727127
## factor(CatholicBin)1              2.85770359 10.62644275  0.2689238
## Agriculture:factor(CatholicBin)1  0.08913512  0.17610660  0.5061430
##                                     Pr(>|t|)
## (Intercept)                      1.919379e-16
## Agriculture                      3.361364e-01
## factor(CatholicBin)1             7.892745e-01
## Agriculture:factor(CatholicBin)1 6.153416e-01
```

Since the coef[1], coef[2] are the reference, whereas the coef[3] is the deviation of intercept from reference when $X_2 = 1$ and coef[4] is deviation of slope coefficient from the reference similarly.

```r
g + geom_abline(intercept = coef(fit)[1], slope = coef(fit)[2], size = 2) +
  geom_abline(intercept = coef(fit)[1] + coef(fit)[3], slope = coef(fit)[2] + coef(fit)[4], size = 2, co
```



Intercept = 64.9076338
Slope = 0.1852508

Regression Multivariable - factors

Fit a model with mpg as the outcome that includes number of cylinders as a factor variable and weight as a possible confounding variable. Compare the effect of 8 versus 4 cylinders on mpg for the adjusted and unadjusted by weight models.

Here, adjusted means including the weight variable as a term in the regression model and unadjusted means the model without weight included.
What can be said about the effect comparing 8 and 4 cylinders after looking at models with and without weight included?.

```r
print(paste('Unadjusted coefficient:',as.character(summary(lm(mpg~cyl,mtcars))$coef[3])))
```

```
## [1] "Unadjusted coefficient: 2.07384360552423"
```

```
print(paste('Adjusted coefficient:',as.character(summary(lm(mpg~cyl+wt,mtcars))$coef[3])))
```

```
## [1] "Adjusted coefficient: -3.19097213898374"
```

Holding weight constant, cylinder appears to have less of an impact on mpg than if weight is disregarded.

Fit a model with mpg as the outcome that considers number of cylinders as a factor variable and weight as confounder. Now fit a second model with mpg as the outcome model that considers the interaction between number of cylinders (as a factor variable) and weight. Give the P-value for the likelihood ratio test comparing the two models and suggest a model using 0.05 as a type I error rate significance benchmark.

```
fitNoInt = lm(mpg~cyl+wt,mtcars)
fitInt = lm(mpg~cyl*wt,mtcars)
lrtest(fitNoInt,fitInt)
```

```
## Likelihood ratio test
##
## Model 1: mpg ~ cyl + wt
## Model 2: mpg ~ cyl * wt
##   #Df  LogLik Df  Chisq Pr(>Chisq)
## 1   4 -74.005
## 2   5 -70.852  1 6.3065    0.01203 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

From the likelihood ratio test, P-value is larger than 0.05. So, according to our criterion, we would fail to reject, which suggests that the interaction terms may not be necessary.

hatvalues of most influencial point

Consider the following data set

```
x <- c(0.586, 0.166, -0.042, -0.614, 11.72)
y <- c(0.549, -0.026, -0.127, -0.751, 1.344)
```

Give the hat diagonal for the most influential point

```
x <- c(0.586, 0.166, -0.042, -0.614, 11.72)
y <- c(0.549, -0.026, -0.127, -0.751, 1.344)
fit = lm(y~x)
hatvalues(fit)
```

```
##         1         2         3         4         5
## 0.2286650 0.2438146 0.2525027 0.2804443 0.9945734
```

Consider the following data set

```
x <- c(0.586, 0.166, -0.042, -0.614, 11.72)
y <- c(0.549, -0.026, -0.127, -0.751, 1.344)
```

Give the slope dfbeta for the point with the highest hat value above.

```r
dfbetas(fit)[5,2]
```

```
## [1] -133.8226
```

Logistic regression

Consider the ravens win/loss rate dataset
Analysing the win rate of Ravens by modelling on the Ravens score in various matches, fitting a binomial distribution to explain the data.

```r
if(!file.exists("./data/ravensData.rda")){
    download.file("https://raw.githubusercontent.com/jtleek/dataanalysis/master/week5/003countOutcomes/d
                  destfile="./data/ravensData.rda",
                  method="curl")
}
load("./data/ravensData.rda")
head(ravensData)
```

```
##   ravenWinNum ravenWin ravenScore opponentScore
## 1           1        W         24             9
## 2           1        W         38            35
## 3           1        W         28            13
## 4           1        W         34            31
## 5           1        W         44            13
## 6           0        L         23            24
```

Linear regression $\text{response}_i = b_0 + b_1\text{predictor}_i + e_i$
Logistic regression $\log\left(\frac{\Pr(response_i|predictor_i, b_0, b_1)}{1-\Pr(response_i|predictor_i, b_0, b_1)}\right) = b_0 + b_1 predictor_i$
If, score==0 then we have
$\log\left(\frac{\Pr(response_i|predictor_i, b_0, b_1)}{1-\Pr(response_i|predictor_i, b_0, b_1)}\right) = b_0$
Thus,
- b_0 - Log odds of Ravens win if they score zero points
- b_1 - Log odds ratio of win probability for each point scored(relative to the zero point)
- $exp(b_1)$ - Odds ratio of win probability for each point scored(relative to the zero point)

and $\frac{e^{b_0}}{1+e^{b_0}}$ is the probability of whether raven's win with score "0"

To find the unit increase in probability of winning,
- Let $b_0 + b_1(predictor_i)$ be the probability of winning for given score, and
- $b_0 + b_1(predictor_i + 1)$" style="display:inline be the probability of winning for given score + 1, which means the probability given unit increase in score.

Subtracting the terms we get, $b_0 + b_1(predictor_i + 1) - b_0 + b_1(predictor_i) = b_1$