STAT462ASS_1

Noah King

2024-08-06

Disclaimer: ChatGPT used to generate lateX (usually). P3q7 not implemented. environment reset on each question.

Part 1: Braking Distance Estimation

Preamble For the following 3 examples, we will consider trained models as a means to make estimations on unseen data from several datasets.

Firstly; we consider the *braking.csv* dataset; the dataset records the approximate braking distance (ft) given the vehicle's speed (MPH); we wish to predict stopping distance given speed, although swapping the predictor and response variables is trivial. Ultimately we will generate some linear model with some degree of error (as a perfect regression here would be non linear). We will not perform steps to minimize some loss; instead we will assess our first model, more later.

1 The braking data is given in terms of MPH/FT; we convert these to metric in place with R functions after loading the *braking.csv* file into memory;

```
#data files just in the same dir
braking_data <- read.csv("braking.csv", header=TRUE, sep=",")
print(braking_data[1, ])
```

```
##   speed dist
## 1     4    2
```

```
mph_to_kmh <- function(mph) {
  kmh <- mph * 1.60934
  return(kmh)
}

ft_to_m <- function(ft) {
  meters <- ft * 0.3048
  return(meters)
}
braking_data$speed <- mph_to_kmh(braking_data$speed)
braking_data$dist <- ft_to_m(braking_data$dist)
print(braking_data[1, ])
```

```
##     speed   dist
## 1 6.43736 0.6096
```

2 In order to conduct our regression model; we split the data into train and test with ratio 4:1. (Note the dataset is only 50 rows in length, however for arbitrary lengths we assign 'approximately' 80% of data to training and the remainder as test by just rounding down)

```
n = nrow(braking_data)
train_ind <- sample(1:nrow(braking_data), size = nrow(braking_data) * 0.8)
df_train <- braking_data[train_ind, ]
df_test <- braking_data[-train_ind, ]

cat("The training set contains", nrow(df_train), "rows, while the test set contains", nrow(df_test), "r
```

```
## The training set contains 40 rows, while the test set contains 10 rows.
```

In order to make estimations on unseen data; we will create a linear regression model; of the format :

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$$

For convenience:

```
x <- df_train$speed
y <- df_train$dist
```

We compute this manually as below; where we apply the standard definitions for intercepts and gradients:

The slope b_1 of the regression line is given by:

$$b_1 = \frac{\text{cov}(x, y)}{\text{var}_x}$$

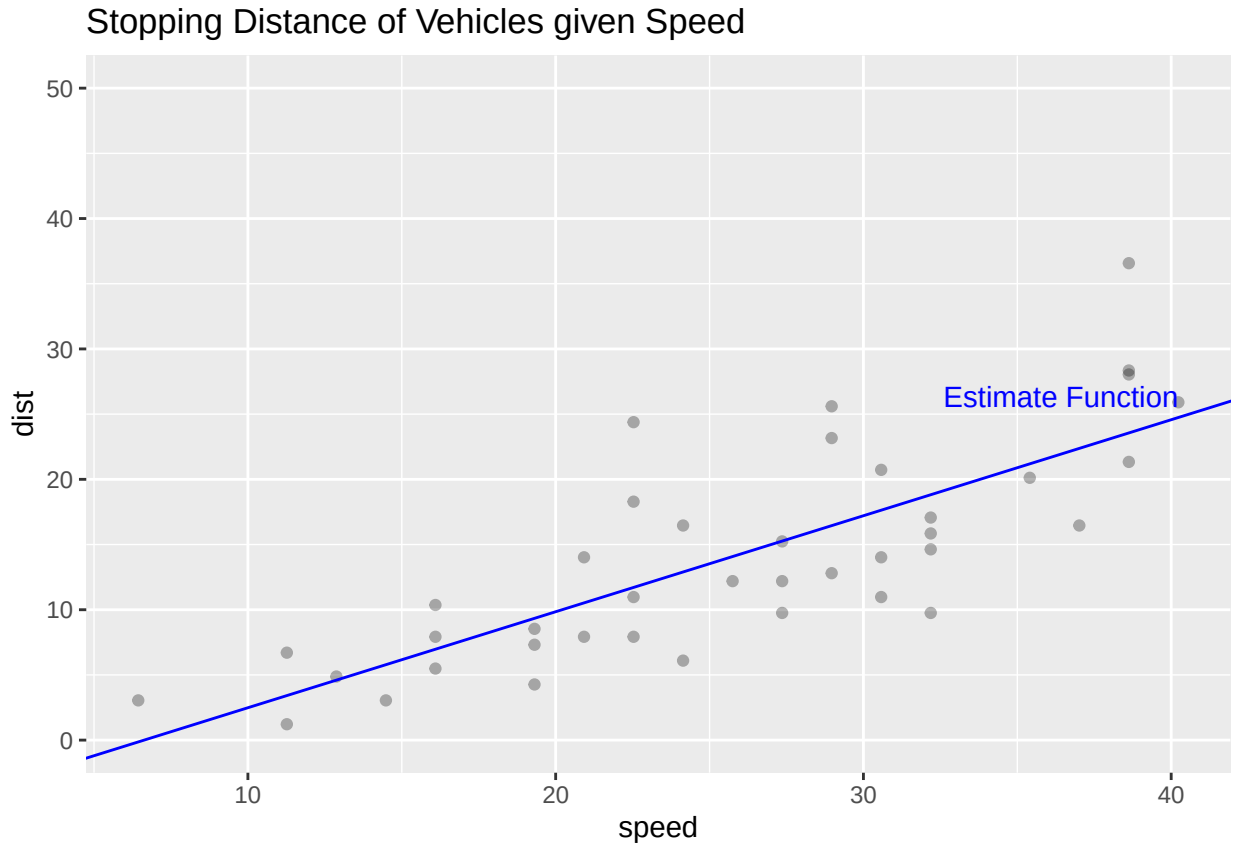
The intercept b_0 is:

$$b_0 = \text{mean}_y - b_1 \cdot \text{mean}_x$$

```
mean_x <- mean(x)
mean_y <- mean(y)
n <- length(x)
x_var <- sum((x - mean_x) ^ 2) / (n - 1)
x_sd <- sqrt(x_var)
xy_cov <- sum((x-mean(x)) *(y-mean(y)))/(n-1)


b_1_h <- xy_cov / x_var
b_0_h <- mean_y - b_1_h * mean_x

lr_func <- function(b0h,b1h,x) {
  a <- b0h + (b1h*x)
  return(a)
}
```

We also define the lr_func to simply return a function whose value is defined by these given x.

```
p <- ggplot(data = df_train, aes(x = speed, y = dist)) +
  geom_point(color = "black", alpha = 0.3) +
  ylim(0, 50) +
  geom_abline(intercept = b_0_h, slope = b_1_h, color = "blue") +
  labs(title = "Stopping Distance of Vehicles given Speed") +
  annotate("text", x = max(df_train$speed), y = b_0_h + b_1_h * max(df_train$speed),
           label = "Estimate Function", hjust = 1, vjust = -0.5, color = "blue")

p
```



Above; we see the training dataset relative to the regression line. In terms of mean squared error (simply the distance between each of these points and the value of the estimate function of that explanatory var in terms of dist) this is the optimal model.

```
## Our initial model yields an intercept of -4.88 and gradient of 0.74. for this selection of randomize
```

2a The LR. model we generate is simply a linear function in *2 space* and thusly we can determine the following:

```
speed_change <- 5
change_dist <- b_1_h * speed_change
change_dist
```

```
## [1] 3.681115
```

```
## We should therefore expect that a change in speed of + 5.00 shall usually increase stopping time 3.68
```

That is; according to our slope we expect `change_dist` per five additional kilometeres (meters)

2b: We can use the R^2 Metric to understand the variance of our model; We manually compute the metric as below (the R code below is reasonably self documenting).

By evaluating R^2 , we can gain insight into how well our model explains the variability of the response variable, (i.e. speed) . A higher R^2 indicates a better fit and a greater degree of variance explained by the model; Essentially; we consider R^2 a measure of the *goodness of fit* of our model relative to other members of the *model space*

```
df_train$predicted_dist <- lr_func(b_0_h,b_1_h,x)
print(head(df_train, 2))
```

```
##       speed    dist predicted_dist
## 28 25.74944 12.1920       14.07918
## 16 20.92142  7.9248       10.52468
```

We create a column of the predicted values. . . ..

```
ESS <- sum((df_train$predicted_dist - mean_y) ^ 2)
TSS <- sum((df_train$dist - mean_y) ^ 2)
R_square = ESS/TSS
cat("The Explained Sum of Squares (ESS) is", ESS, ", the Total Sum of Squares (TSS) is", TSS, ", and the
```

```
## The Explained Sum of Squares (ESS) is 1598.646 , the Total Sum of Squares (TSS) is 2592.887 , and the
```

For this sample (and also the 80% sampled of that sample for our train data), the R^2 value is ‘R_square; indicating a relatively good fit; that is around 61.7% of the variance is explainable by the regression model

The relationship here is only *moderately strong.*

2c: Further to the idea of the relationship as *moderately strong*, we consider whether speed is a signficiatn predictor for dist (i.e. speed vs braking distance) at some confidence level (i.e proportion of times that conf. interval would contain our true theta (true y))

Note, obviously we don’t know population variance; so as such we construct intervals of the following form:

Our intervals are constructed about the mean of our dataset:

In R code this takes the form:

```
n <- nrow(df_train)
alpha <- 0.05
mean_x <- mean(df_train$speed)
sigma_sq <- (1 / (n - 1)) * sum((df_train$speed - mean_x)^2)
se <- sqrt(sigma_sq / n)
dof <- n - 1
t_critical <- qt(1 - alpha / 2, df = dof)
l <- mean_x - (t_critical * se)
r <- mean_x + (t_critical * se)

cat(sprintf("Mean: %.4f\n", mean_x))
```

```
## Mean: 25.6287
```

```
cat(sprintf("95%% Confidence Interval: [%.4f, %.4f]\n", l, r))
```

```
## 95% Confidence Interval: [22.8475, 28.4100]
```

Note; as our confidence interval does not include zero; we can be sure that at the 95% level that indeed speed is a significant predictor for stopping distance.

Therefore; we can infer that at a 95% confidence interval; our data exists within ##s speed a significant predictor for dist at the 95% confidence level?

2d As we mentioned earlier (and also the entire inclusion of confidence intervals alludes to) our sample cannot perfectly reflect the pop. Therefore; our regression model is *good within some degree of accuracy.*

A sample query of our regression model is *If Im going 30kmh, what is the max / min distance which I will actually stop in* and we can compute that as follows:

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 \cdot X$$

$$\text{RSS} = \text{TSS} - \text{ESS}$$

$$\sum (X_i - \bar{X})^2 = \sum_{\text{i=1}}^{n} (X_i - \bar{X})^2$$

$$\text{SE}(\hat{y}) = \sqrt{\left(\frac{1}{n} + \frac{(X - \bar{X})^2}{\sum (X_i - \bar{X})^2} \right) \cdot \frac{\text{RSS}}{n-2}}$$

$$\alpha = 1 - \frac{\text{confidence\_level}}{100}$$

$$\text{df} = n - 2$$

$$t_{\text{critical}} = t\left(1 - \frac{\alpha}{2}, \text{df}\right)$$

$$\text{Lower Bound} = \hat{y} - t_{\text{critical}} \cdot \text{SE}(\hat{y})$$

$$\text{Upper Bound} = \hat{y} + t_{\text{critical}} \cdot \text{SE}(\hat{y})$$

Or in R code:

```
speed <- 30
certainty <- 80 #percent
```

```r
pred_sped <- lr_func(b_0_h, b_1_h, speed)
RSS <- TSS - ESS
sum_squared_diff <- sum((df_train$speed - mean_x)^2)
se_pred <- sqrt((1 / n + (speed - mean_x)^2 / sum_squared_diff) * RSS / (n - 2))
alpha <- 1-(certainty/100)
df <- n - 2
t_critical <- qt(1 - alpha / 2, df)
lower_bound <- pred_sped - t_critical * se_pred
upper_bound <- pred_sped + t_critical * se_pred
```

```
## Predicted braking distance for a car going at 30.0 km/h: 17.21 meters
```

```
## Bounded in [16.02, 18.39] meters
```

Predictions given K-NN KNN is an alternate approach for finding relationships in data and generating a model; We can consider KNN topologically; in <4 space we're simply making a prediction by averaging the neighbours of our input variable. In our example, we consider only the *distance* on the *speed* variable; which is more easily labelled *the x axis*.

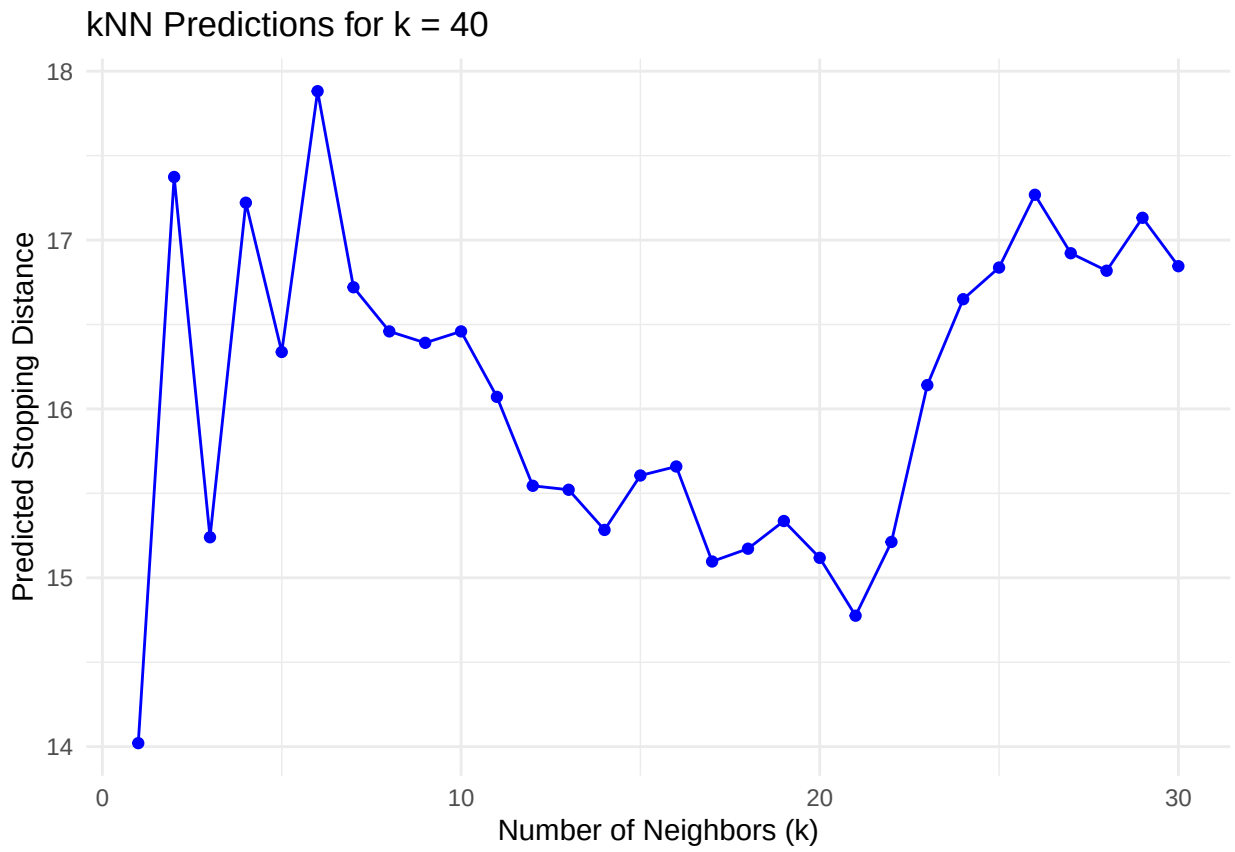
```r
k <- 10


kNN <- function(data, k_ = 5, x0, y0, xstar){
  ystar <- data %>%
    mutate(dist_diff = abs(!!as.name(x0)-xstar)) %>%
    arrange(dist_diff) %>%
    slice(1:k_) %>%
    pull(!!as.name(y0)) %>%
    mean()

  return (ystar)
}

kNN(data = df_train, k_ = k, x0 = "speed", y0 = "dist", xstar = 30)
```

```
## [1] 16.4592
```

We can show that for k=10; a vehicle travelling at 30kmh would take the 16 or so meters above to stop... According to the KNN model:

```r
k_values <- 1:30
results <- data.frame(
  k = k_values,
  predicted_dist = sapply(k_values, function(k_) kNN(data = df_train, k_ = k_, x0 = "speed", y0 = "dist
)

# Plot the results
ggplot(results, aes(x = k, y = predicted_dist)) +
  geom_line(color = "blue") +
  geom_point(color = "blue") +
  labs(title = paste("kNN Predictions for k =", 40),
```

```
       x = "Number of Neighbors (k)",
       y = "Predicted Stopping Distance") +
  theme_minimal()
```

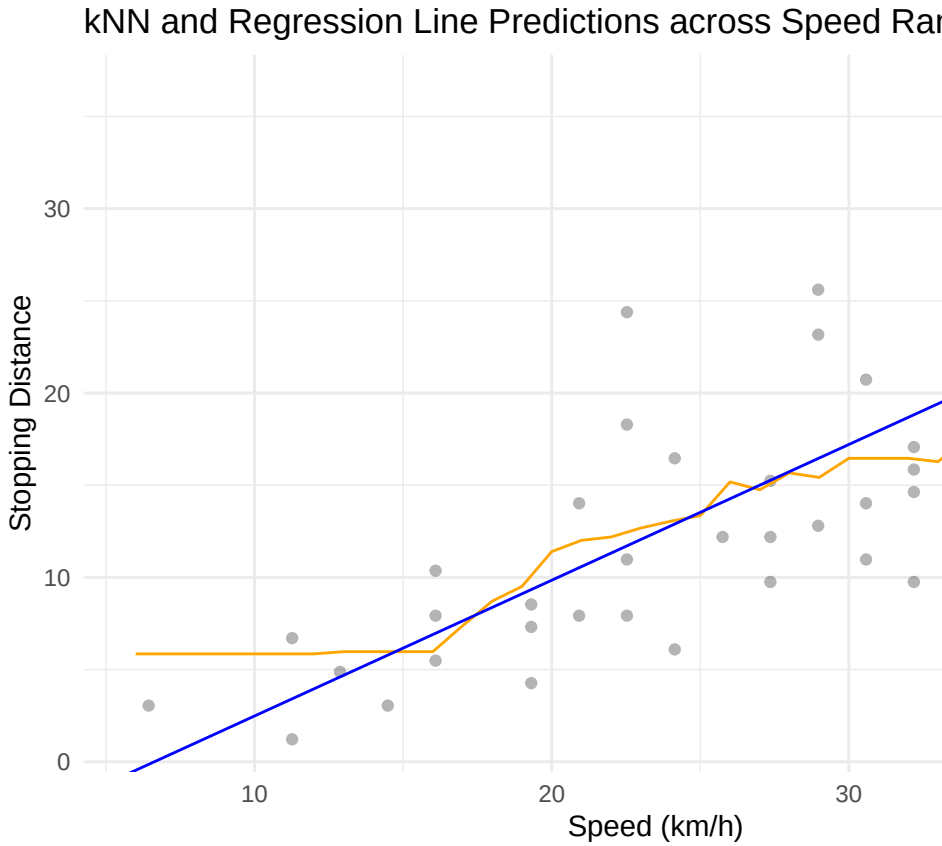


While can use our KNN model to make predictions, but we must note: Our sample is *small* and the accuracy (see precsion) of the prediction we make considers the K (the number of neighbours) and also the skew, or outliers in the data.

Note that we can confirm our KNN, when scoped to the entire dataset, produces the mean for distance. Remembering again that this sample is only 40 deep; KNN doesn't provide a particularly insightful measure of *accuracy*; rather were n to increase; we could make greater assumptions; also we are operating only in R^1 here; Knn in R^2 is an option, we will see this later on.

```
x_range <- seq(from = floor(min(df_train$speed)), to = ceiling(max(df_train$speed)), by = 1)
predictions <- sapply(x_range, function(xstar) kNN(data = df_train, k_ = k, x0 = "speed", y0 = "dist", x

predictions_df <- data.frame(
  speed = x_range,
  predicted_dist = predictions
)
```

```
ggplot(data = df_train, aes(x = speed, y = dist)) +
  geom_point(color = "black", alpha = 0.3) +   # Original data points
  geom_line(data = predictions_df, aes(x = speed, y = predicted_dist), color = "orange", linetype = "so
geom_abline(intercept = b_0_h, slope = b_1_h, color = "blue") +
  labs(title = "kNN and Regression Line Predictions across Speed Range (TRAINING)",
       x = "Speed (km/h)",
       y = "Stopping Distance") +
  theme_minimal()
```



relative performance of KNN vs LR

So what of testing?

We run, again, into the issue of selecting *k*. Our test set is only *10* rows and the train set only *40*.

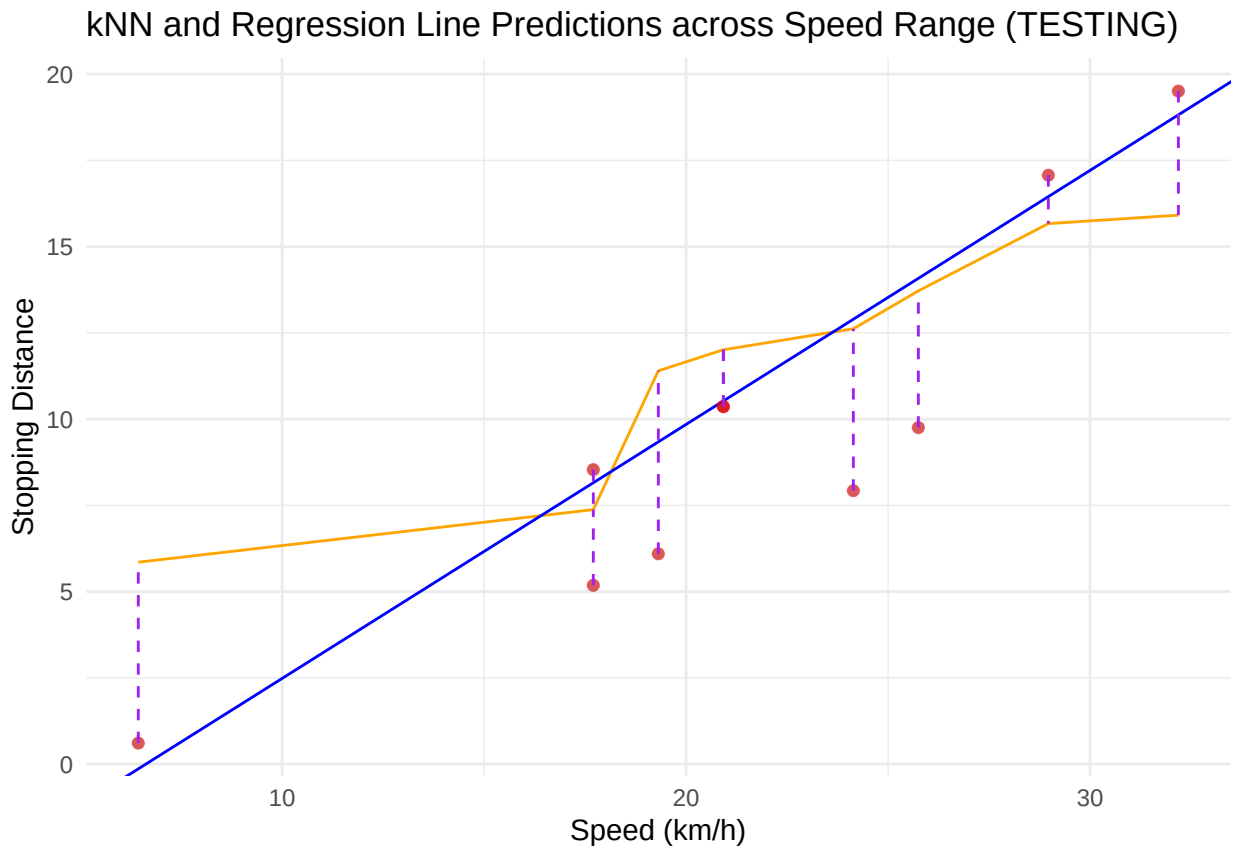
Nonetheless; we consider the speed values in the df_test set against the df_train model...

We have selected 3 here, values greater than three are essentially useless

```
# Define the number of neighbors
k <- 10


# Predict the stopping distance for a vehicle traveling at 30 km/h

predictions_knn <- sapply(df_test$speed, function(xstar) kNN(data = df_train, k_ = k, x0 = "speed", y0 =

# Create a data frame for plotting
predictions_df_knn <- data.frame(
  speed = df_test$speed,
```

```
  predicted_dist = predictions_knn
)


x_range <- seq(from = floor(min(df_test$speed)), to = ceiling(max(df_test$speed)), by = 1)


ggplot(data = df_test, aes(x = speed, y = dist)) +
  geom_point(color = "black", alpha = 0.3) +
  geom_line(data = predictions_df_knn, aes(x = speed, y = predicted_dist), color = "orange", linetype =
  geom_abline(intercept = b_0_h, slope = b_1_h, color = "blue") +
   geom_point(data = df_test, aes(x = speed, y = dist), color = "red", alpha = 0.5, size = 2, shape = 1
   geom_segment(data = df_test, aes(x = speed, xend = speed, y = dist, yend = sapply(speed, function(x)

  # Labels and theme
  labs(title = "kNN and Regression Line Predictions across Speed Range (TESTING)",
       x = "Speed (km/h)",
       y = "Stopping Distance") +

  theme_minimal()
```

```
## Warning in geom_point(data = df_test, aes(x = speed, y = dist), color = "red",
## : Ignoring unknown parameters: `label`
```



Above; we see the *orange* KNN model @ 10 and the regression model (also one set of differences).

It is challenging (because the test set is so small) to determine which is the most accurate model. For the entirety of the dataset: MSE is calculated as such:

We can assess the MSE more specifically:

```
predicted_dist_lr <- lr_func(b_0_h, b_1_h, df_test$speed)
mse_lr <- mean((df_test$dist - predicted_dist_lr) ^ 2)
mse_lr
```

```
## [1] 6.43857
```

```
predictions_knn <- sapply(df_test$speed, function(xstar) kNN(data = df_train, k_ = k, x0 = "speed", y0 =
mse_knn <- mean((df_test$dist - predictions_knn) ^ 2)
mse_knn
```

```
## [1] 11.98226
```

At k = 10, the MSE is much *worse* for our KNN model; specifically.

Due to the small size of our sample, it is a poor choice for generating predictions for this data. Changing K can / will impact this; however the regression model remains best (linear).

```
rm(list = ls())
```

Resetting for atomicity between questions.

As before; we will generate linear regression models such that we can predict unseen data:

The datset *fillipino household income* can be used to make this kind of prediction;

From the *dataset for Filipino HouseHold Income* we can make some inferences about household income for new, unseen data. While the dataset here contains 60 columns; see 59 predictors for predicting some single variable; we will consider only the relationship between the number of children (5-17) in a household and the relative household income.

```
income <- read.csv("income.csv", header=TRUE, sep=",")
print(ncol(income))
```

```
## [1] 60
```

Note that the entire dataset is around 22MB uncompressed; for convenience we will drop all but the two concerned columns; Our predictive model will consider the Total Household income relative to the number of children in said household between 5 and 17 years of age in the Phillipines.

1: Firstly; as the dataset itself is ~22mb; we drop all but the concerned columnss (as a above) and rename them for convienience

```
income <- income[, c("Total.Household.Income", "Members.with.age.5...17.years.old"), drop = FALSE]
colnames(income)[colnames(income) == "Total.Household.Income"] <- "income"
colnames(income)[colnames(income) == "Members.with.age.5...17.years.old"] <- "children"
```

Note that the value for *number of children* is discrete (nobody has half a child) making the comment from part 1 about swapping the pred/eval variables not applicable here. The values are bounded between 0 and 8

```r
max_value <- max(income$children)
max_value
```

```
## [1] 8
```

```r
min_value <- min(income$children)
min_value
```

```
## [1] 0
```

2: As usual for any ML training, we split our data into test / train at 4:1 ratio. As before; it is unlikely that each time we make a random split that the row counts persist, We expect 80% +/- 1 row (non leaking) We are not using a validation set for this exploration.

```r
train_ind <- sample(1:nrow(income), size = nrow(income) * 0.8)
df_train <- income[train_ind, ]
df_test <- income[-train_ind, ]

print(nrow(df_train))
```

```
## [1] 33235
```

```r
print(nrow(df_test))
```

```
## [1] 8309
```

```r
print(nrow(income))
```

```
## [1] 41544
```

3: Now; using in built functions we generate our linear model as in part 1;

income = b0 + b1 * childre

We are looking to fit the above formula again; this is a simple linear function with intercept and gradient b0, b1...

```r
model <- lm(income ~ children, data = df_train)
summary(model)
```

```
##
## Call:
## lm(formula = income ~ children, data = df_train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
##  -251002  -141336   -80387    43866 11553701
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept)   262287       2242 116.986   <2e-16 ***
## children      -10773       1150  -9.371   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 294400 on 33233 degrees of freedom
## Multiple R-squared:  0.002635,   Adjusted R-squared:  0.002605
## F-statistic: 87.81 on 1 and 33233 DF,  p-value: < 2.2e-16
```

```
b0 <- as.numeric(coef(model)[1])
b1 <- as.numeric(coef(model)[2])

b0
```

```
## [1] 262287.1
```

```
b1
```

```
## [1] -10772.87
```

```
b0
```

```
## [1] 262287.1
```

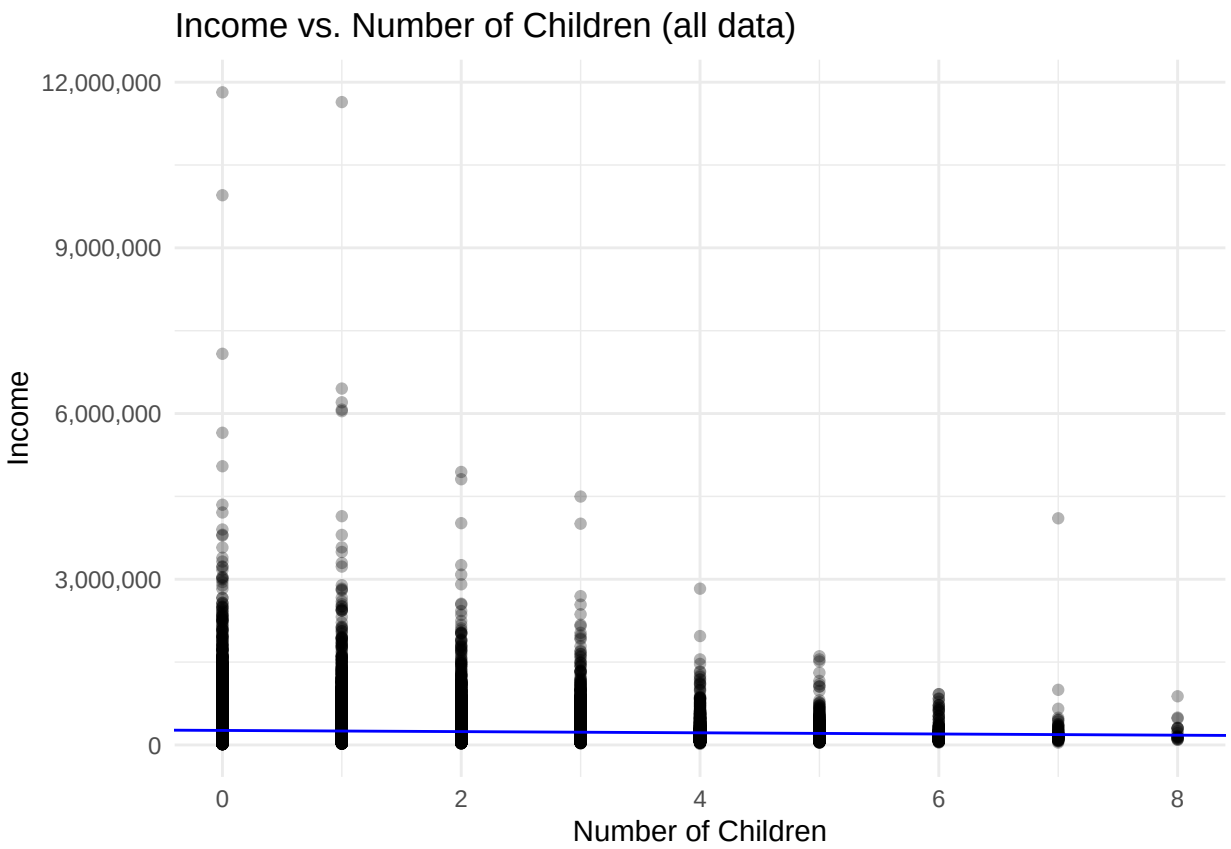
```
b1
```

```
## [1] -10772.87
```

We can simply read off the model; with zero children we should expect ₱262287.09 with a decrease of ₱-10772.87 per child between 5 and 17.

Later, we will briefly look into some socioeconomic reasons that we see a decrease. Not that this dataset is given in *Phillipine Pesos* who currently convert at around *0.028 NZD = 1 PHP*. For the sake of readability, all currencies in this document are given to 2dp, although for the purpose of the model we simply use R's inbuilt floating point precision.

```
# Format Labels to be readable
format_y_labels <- function(x) {
  format(round(x, 0), big.mark = ",", scientific = FALSE)
}


p <- ggplot(data = income, aes(x = children, y = income)) +
  geom_point(color = "black", alpha = 0.3) +
  geom_abline(intercept = b0, slope = b1, color = "blue") +
  labs(title = "Income vs. Number of Children (all data)",
       x = "Number of Children",
       y = "Income") +
  theme_minimal() +
  scale_y_continuous(labels = format_y_labels)  # Apply custom label formatting

print(p)
```

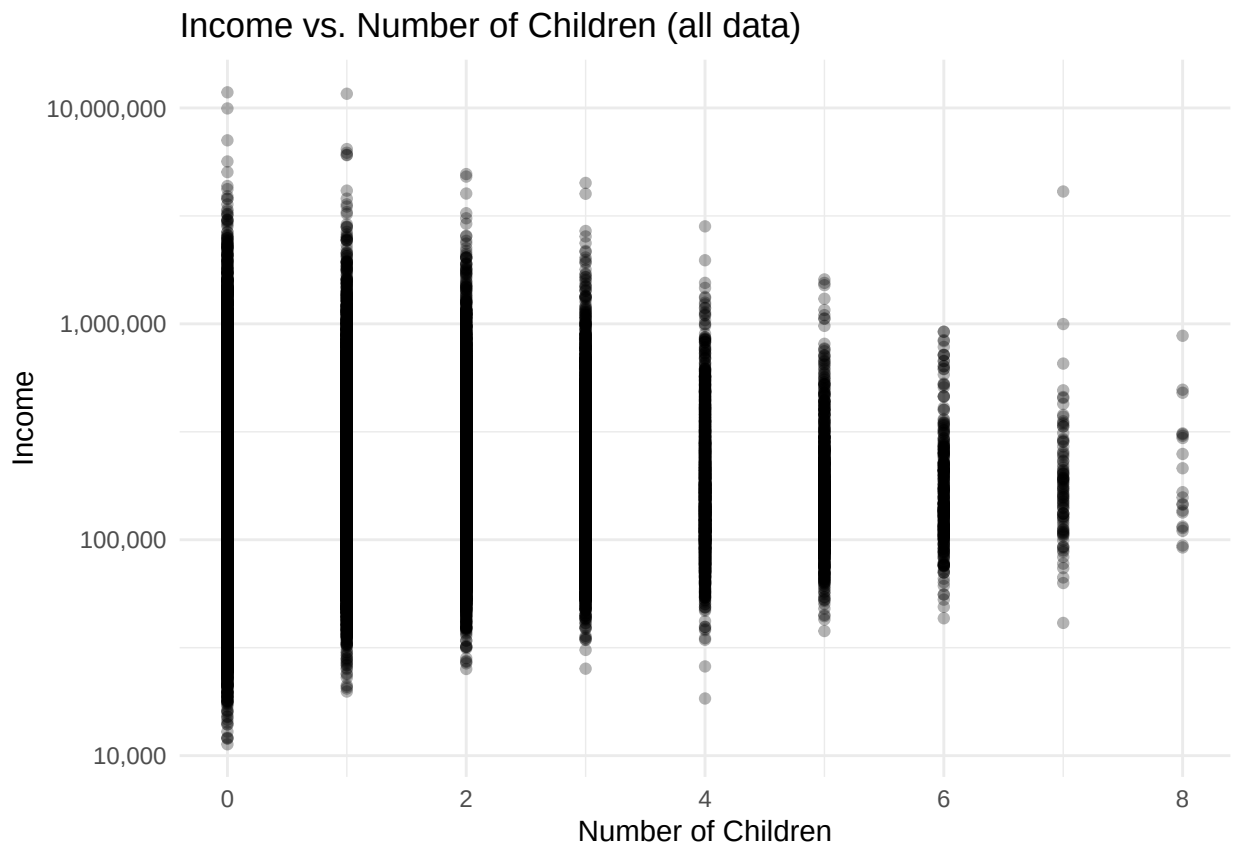


Immediately apparent from the dataset is that our data is strongly aggregated toward the *lower* parts of our scale. The regression line is given in *blue* for our model; it is indeed decreasing.

Whether or not to consider extreme values as outliers is a matter of the domain; here, deciding to remove them is imprudent, as there's 'no good reason' that the data cannot exist and be true.

This is the entire dataset; questioning note that we will not remove any outlying data. While our regression line appears flat; it is infact decreasing as we expect; The distribution of wealth / income is best described by the logarithmic graph as follows:

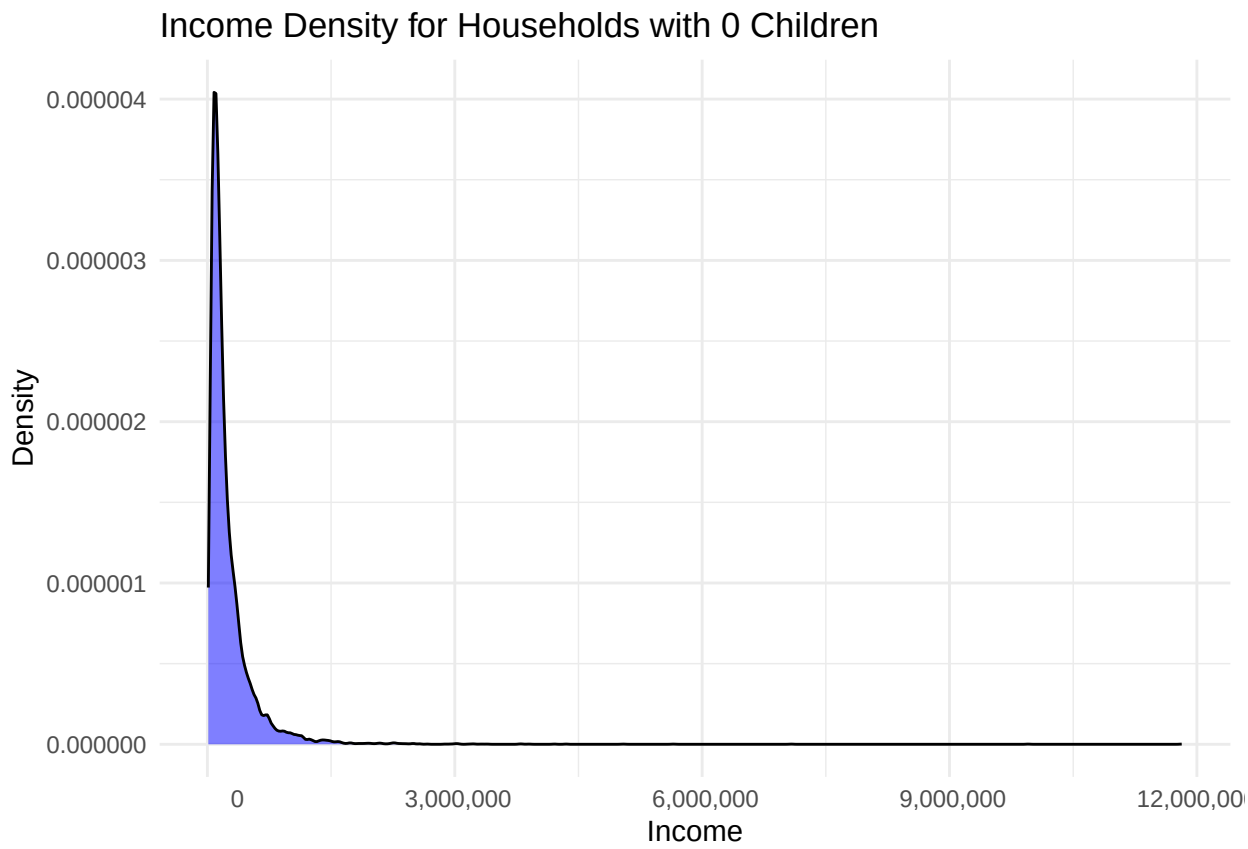
```
p <- ggplot(data = income, aes(x = children, y = income)) +
  geom_point(color = "black", alpha = 0.3) +
  geom_abline(intercept = b0, slope = b1, color = "blue") +
  labs(title = "Income vs. Number of Children (all data)",
       x = "Number of Children",
       y = "Income") +
  theme_minimal() +
  scale_y_log10(labels = format_y_labels)  # Apply custom label formatting

print(p)
```



For each of the 8 discrete number of *children* values; we calculate their mean and upper/lower bounds @90% certainty (90% pred intervals)

For example; We can see as follows that the distribution for *zero children* is very centralised

```
zerokids <- subset(df_train, children ==0 )

label_format <- function(x) {
  format(x, big.mark = ",", scientific = FALSE)
}

ggplot(zerokids, aes(x = income)) +
  geom_density(fill = "blue", alpha = 0.5) +
  labs(title = "Income Density for Households with 0 Children",
       x = "Income",
       y = "Density") +
  scale_x_continuous(labels = label_format) +
  scale_y_continuous(labels = label_format) +
  theme_minimal()
```



3b Again, using in built methods we have the following:

```
new_data <- data.frame(children = 0:8)
predictions <- predict(model, newdata = new_data, interval = "prediction", level = 0.1)
```

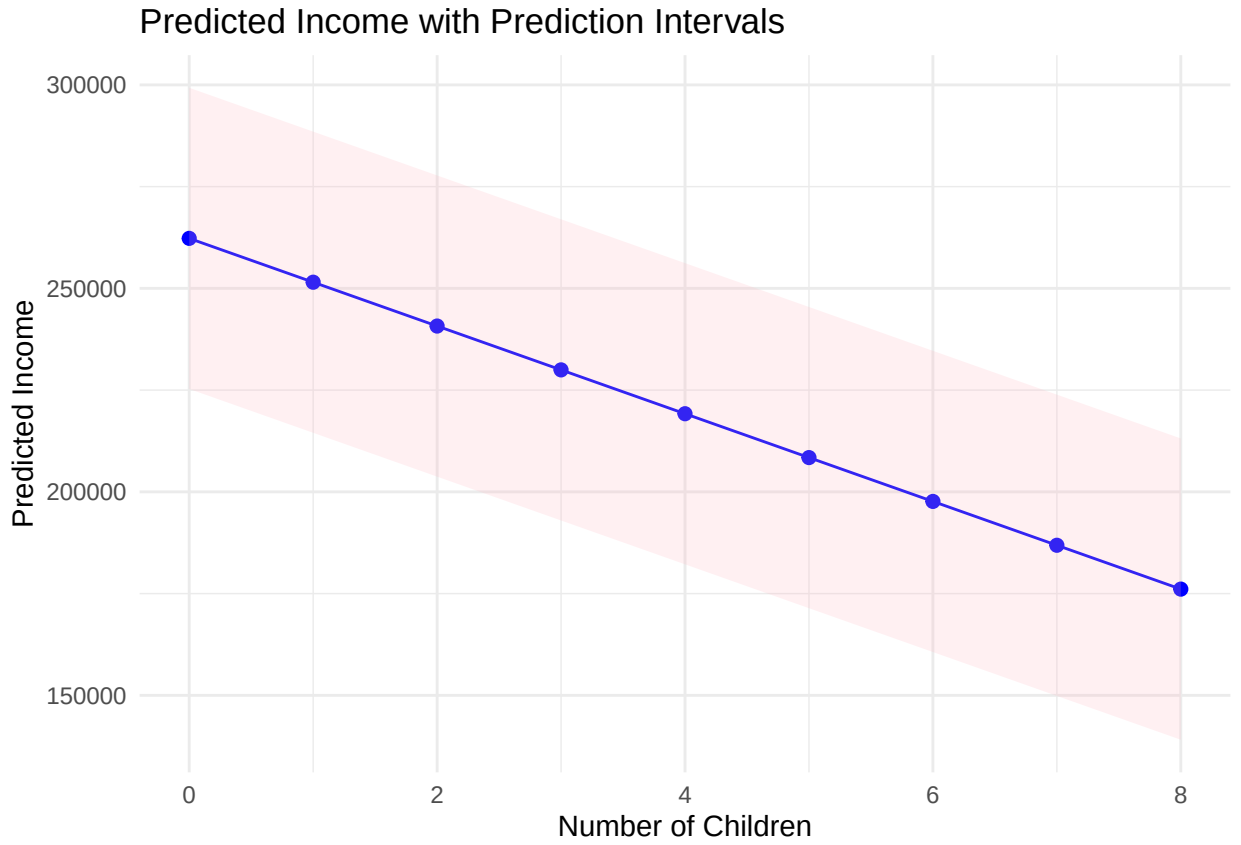
```
predictions_df <- data.frame(
  children = new_data$children,
  fit = predictions[, "fit"],
  lwr = predictions[, "lwr"],
  upr = predictions[, "upr"]
)
kable(predictions_df,
      col.names = c("Number of Children", "Predicted Income", "Lower Bound (10%)", "Upper Bound (90%)"),
      caption = "Predicted Income with 10% Prediction Intervals for Different Numbers of Children",
      digits = 2) # Format the numbers to 2 decimal places
```

Table 1: Predicted Income with 10% Prediction Intervals for Different Numbers of Children

Number of Children	Predicted Income	Lower Bound (10%)	Upper Bound (90%)
0	262287.1	225287.3	299286.8
1	251514.2	214515.0	288513.5
2	240741.4	203742.0	277740.7
3	229968.5	192968.5	266968.5
4	219195.6	182194.4	256196.8

Number of Children	Predicted Income	Lower Bound (10%)	Upper Bound (90%)
5	208422.8	171419.8	245425.7
6	197649.9	160644.6	234655.2
7	186877.0	149868.8	223885.2
8	176104.1	139092.5	213115.9

```
ggplot(predictions_df, aes(x = children, y = fit)) +
  geom_point(color = "blue", size = 2) +
  geom_line(color = "blue") +
  geom_ribbon(aes(ymin = lwr, ymax = upr), alpha = 0.2, fill = "lightpink") +
  labs(title = "Predicted Income with Prediction Intervals",
       x = "Number of Children",
       y = "Predicted Income") +
  theme_minimal()
```



For each of the predictions given #of children (here represented as n+1 from zero) we can see the 90% confidence intervals (the lower abd upr bounds) around the predicted values @ 0 through 8 children. (the two bounds represented in pink)

3c We use our test set to determine the actual performance of the model on unseen data; There are many applicable metrics; but a simple, cursory calculation is simply determining the overall proportion of data which falls within our confidence intervals in the test set...

We do that as follows; we use the inbuilt predict to find mean, bounds, simply filter out members of the test set who are not bounded and we have our percentage

```r
predictions <- predict(model, newdata = df_test, interval = "prediction", level = 0.9)
predicted <- predictions[, "fit"]
lower_bound <- predictions[, "lwr"]
upper_bound <- predictions[, "upr"]
within_interval <- df_test$income >= lower_bound & df_test$income <= upper_bound
count_within_interval <- sum(within_interval)
total_points <- length(within_interval)
percent_within_interval <- (count_within_interval / total_points) * 100

cat("Number of points within the interval:", count_within_interval, "\n")
```

```
## Number of points within the interval: 7947
```

```r
cat("Total number of points:", total_points, "\n")
```

```
## Total number of points: 8309
```

```r
cat("Percentage within the interval:", percent_within_interval, "%\n")
```

```
## Percentage within the interval: 95.64328 %
```

4: Earlier, we noticed that it is difficult to build a model which accounts for the large concentration of data points around the mean. As a means to mitigate this (and lessen the effect of outlying data) we now move forward with predicting log_income = log(income).

Firstly; we transform the data (for the sake of convienience we do this in place)

```r
df_train$log_income <- log(df_train$income)
df_test$log_income <- log(df_test$income)
```

... fitting the model...

```r
log_model <- lm(log_income~children, data=df_train)
summary(log_model)
```

```
##
## Call:
## lm(formula = log_income ~ children, data = df_train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.7529 -0.5260 -0.0815  0.4896  4.2009
##
## Coefficients:
##              Estimate Std. Error  t value Pr(>|t|)
## (Intercept) 12.084089   0.005876 2056.448   <2e-16 ***
## children     0.001545   0.003013    0.513    0.608
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7717 on 33233 degrees of freedom
## Multiple R-squared:  7.91e-06,   Adjusted R-squared:  -2.218e-05
## F-statistic: 0.2629 on 1 and 33233 DF,  p-value: 0.6081
```

```r
b0 <- as.numeric(coef(log_model)[1])
b1 <- as.numeric(coef(log_model)[2])

b0
```

```
## [1] 12.08409
```

```r
b1
```

```
## [1] 0.001544844
```

We see an intercept; but a very flat increaese for the log income.

Our predictions at 90% again

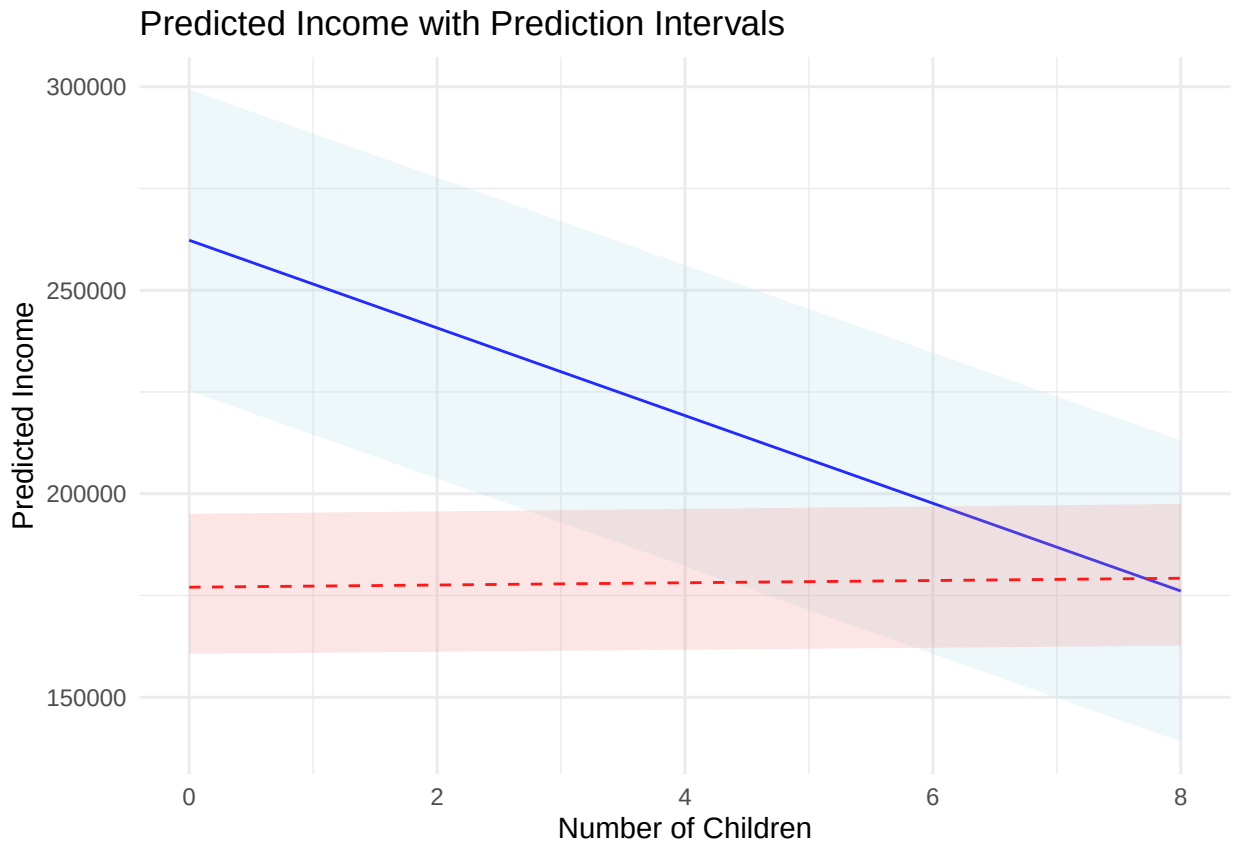
```r
log_predictions <- predict(log_model, newdata = new_data, interval = "prediction", level = 0.9)
```

```r
log_test_predictions <- predict(log_model, newdata = df_test, interval = "prediction", level = 0.9)
log_predicted <- log_test_predictions[, "fit"]
log_lower_bound <- log_test_predictions[, "lwr"]
log_upper_bound <- log_test_predictions[, "upr"]
log_within_interval <- df_test$log_income >= log_lower_bound & df_test$log_income <= log_upper_bound
log_count_within_interval <- sum(log_within_interval)
log_total_points <- length(log_within_interval)
log_percent_within_interval <- (log_count_within_interval / log_total_points) * 100

cat("Percentage within the log-transformed interval:", log_percent_within_interval, "%\n")
```

```
## Percentage within the log-transformed interval: 90.42003 %
```

```r
# Create new data for prediction
new_data <- data.frame(children = 0:8)

# Predict on new data (log-transformed predictions)
non_log_predictions <- predict(model, newdata = new_data, interval = "prediction", level = 0.1)

# Generate predictions for the log-transformed model
# Ensure you have fitted the model for log_income
log_predictions <- predict(log_model, newdata = new_data, interval = "prediction", level = 0.1)

# Transform log predictions back to original scale
log_predictions_original_scale <- exp(log_predictions) - 1

# Combine both prediction results into a single data frame
predictions_combined <- data.frame(
  children = new_data$children,
  fit_non_log = non_log_predictions[, "fit"],
  lwr_non_log = non_log_predictions[, "lwr"],
  upr_non_log = non_log_predictions[, "upr"],
  fit_log = log_predictions_original_scale[, "fit"],
  lwr_log = exp(log_predictions[, "lwr"]) - 1,
```

```
  upr_log = exp(log_predictions[, "upr"]) - 1
)

# Plot both log and non-log predictions with their bounds
ggplot(predictions_combined, aes(x = children)) +
  # Non-log predictions
  geom_line(aes(y = fit_non_log), color = "blue", linetype = "solid") +
  geom_ribbon(aes(ymin = lwr_non_log, ymax = upr_non_log), alpha = 0.2, fill = "lightblue") +

  # Log-transformed predictions
  geom_line(aes(y = fit_log), color = "red", linetype = "dashed") +
  geom_ribbon(aes(ymin = lwr_log, ymax = upr_log), alpha = 0.2, fill = "lightcoral") +

  labs(title = "Predicted Income with Prediction Intervals",
       x = "Number of Children",
       y = "Predicted Income") +
  theme_minimal()
```



4/5: At a high level; we firstly consider the negative gradient in the first model; while the Phillipines govt does maintain a program similar to *working for families* in the form of a cash payment; it is not accessible to all and there is extensive evidence that succeptability to poverty is proportional to family size. [https://scholarspace.manoa.hawaii.edu/server/api/core/bitstreams/cd83470d-e91a-483f-ac29-643b3476e627/content].

With that in mind; our *decreasing* gradient is justified.

Our goal, with fitting the logarithmic model; was to mitigate the effect of outliers; as is typical in income data, there are outliers whose occurance is legitimate; yet non mitigable. We transform the original data to

reduce the effects of those outliers on our residuals. The lost information (to the model, recoverable trivially) is, for our use, not of particular importance.

After applying the log transformation, the means of the dataset and the prediction intervals tend to be lower and more centralized. This centralization reflects the reduced influence of extreme values and provides a model that is more focused on the central portion of the data.

Intuitively; while we care about the bulk of the information the *most*; it is clear that we can no longer *accurately* predict outliers (where accurately means with as much accuracy) as information is lost to the top end of the *logarithmic* func.

- Is this a good fit for our data / predictor / use-case?

Because, as we saw earlier, the data is so heavily concentrated around the mean and so heavily skewed; this context *is* a good fit... *in some cases*; In outlier-critical cases, like government policy creation, we would prioritize a direct understanding of income levels.

In cases like retail pricing analysis or dynamic pricing, where people over some threshold are all a member of the same ceiling, but where we care about small changes on a smaller scale; it holds that using the logarithmic model is prudent.

Predicting Possum Age

```
rm(list = ls())
```

Firstly; We consider the dataframe created from our possum dataset; note that here we handle all rows containing *NA* values by simply removing them.

```
#data files just in the same dir
pdata <- read.csv("possums.csv", header=TRUE, sep=",")
pdata <- pdata[!is.na(pdata$age), ]
pdata <- na.omit(pdata)
print(pdata[1, ])
```

```
##   case site Pop sex age hdlngth skullw totlngth taill footlgth earconch  eye
## 1    1    1 Vic   m   8    94.1   60.4       89    36     74.5     54.5 15.2
##   chest belly
## 1    28    36
```

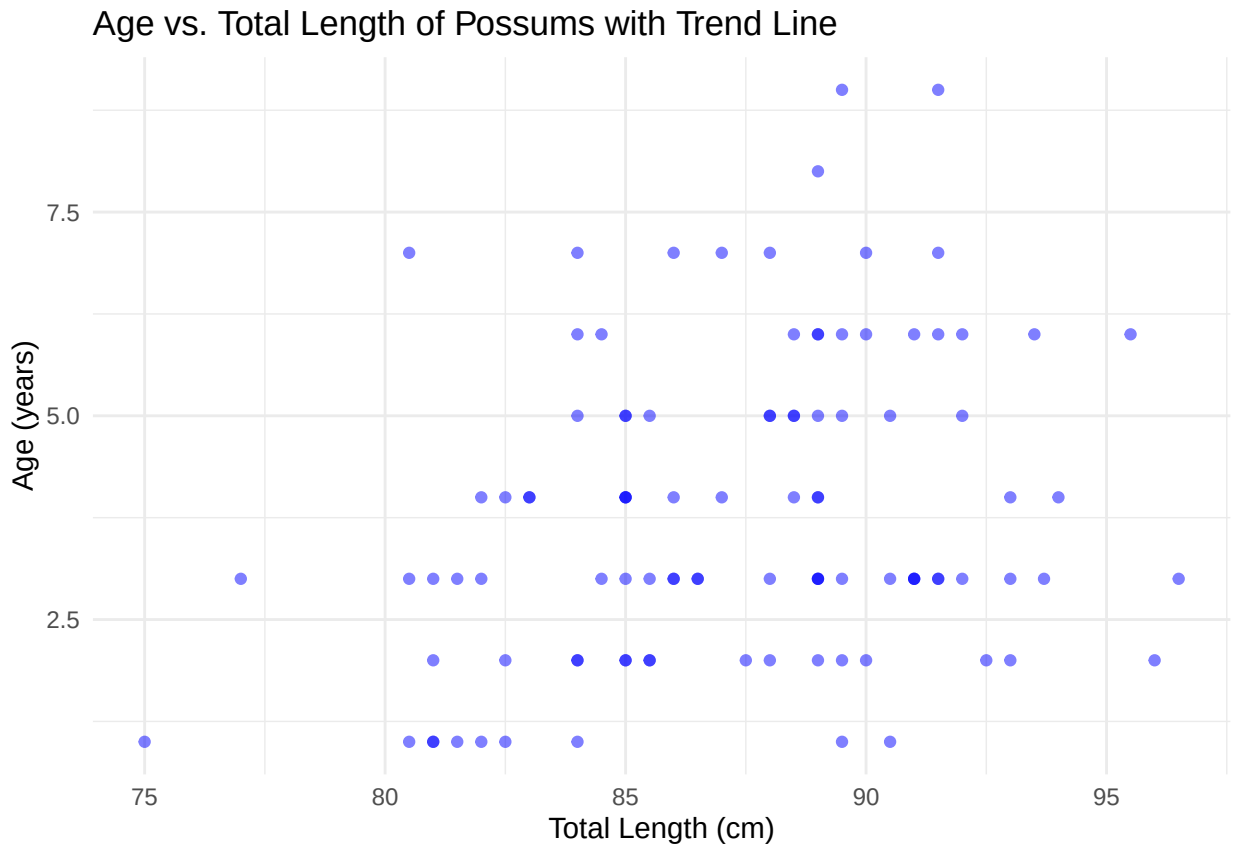
In this section we explore *multiple regression*. It holds that predicting a value given more than one explanatory variable should help to create more accurate predictions than a binary relationship; however; creating a linear separation is not always possible and we will explore creating a separation in higher spaces.

To create the optimal model, we wish to select features that are independent, ceteribus paribus, meaning that when all other variables are held constant, each feature's contribution to the model should be distinct and not redundant with other features....

However, we also wish to select features that best explain the variability in the predictor.

1:

```r
ggplot(pdata, aes(x = totlngth, y = age)) +
  geom_point(color = "blue", alpha = 0.5) +
  labs(title = "Age vs. Total Length of Possums with Trend Line",
       x = "Total Length (cm)",
       y = "Age (years)") +
  theme_minimal()
```



We can see a very slight correlation, a few outliers; there is nothing here that indicates a strong correlation. . .

```r
age_len_model <- lm(age ~ ., data = pdata)
summary(age_len_model)
```

```
##
## Call:
## lm(formula = age ~ ., data = pdata)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.6493 -1.1199 -0.0241  0.9293  4.5272
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -4.28694    8.02286  -0.534   0.5945
## case        -0.05664    0.02714  -2.086   0.0399 *
## site         0.80381    0.35646   2.255   0.0266 *
```

```
## Popother    -2.88255    1.50906  -1.910   0.0594 .
## sexm        -0.13824    0.40535  -0.341   0.7339
## hdlngth      0.13569    0.09570   1.418   0.1598
## skullw       0.06004    0.08782   0.684   0.4960
## totlngth    -0.03272    0.08924  -0.367   0.7148
## taill        0.12151    0.16495   0.737   0.4633
## footlgth    -0.19583    0.09341  -2.096   0.0389 *
## earconch    -0.08327    0.12306  -0.677   0.5004
## eye          0.19349    0.20511   0.943   0.3481
## chest        0.06376    0.14385   0.443   0.6587
## belly        0.15308    0.08752   1.749   0.0838 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.751 on 87 degrees of freedom
## Multiple R-squared:  0.2728, Adjusted R-squared:  0.1641
## F-statistic:  2.51 on 13 and 87 DF,  p-value: 0.005761
```

And indeed we can see that totlength is one of the least significant predictors for age in our datset; the p-value measures the significance of a particular predictor's effect on our model; so the lower the value, the less impactful.

In fact totlnght has a high Pvalue @ 0.7~ and is therefore not a substantial contributor; being the highest it is likely to be excluded from feature selection later....

2a We remove the pop(population) and case(case number) columns; neither are particularly useful to our regression model and as such are disregarded less they introduce noise

```
pdata <- pdata[, !(names(pdata) %in% c("case", "Pop"))]
print(pdata[1, ])
```

```
##   site sex age hdlngth skullw totlngth taill footlgth earconch  eye chest belly
## 1    1   m   8    94.1   60.4       89    36     74.5     54.5 15.2    28    36
```

2b There are 7 unique classifications for *site* as such we encode them into identifiers on six columns;

For regressions using one-hot encoding, some n-1 columns are created (where n is the number of unique values).

This is done in the interest of preserving compute (i.e. the seventh site here would be the default where none of 1-6 apply). Furthermore, for each observation; the values of cols across 1-7 for the onehot would become colinear, summing to one always; we wish to preserve the independance.

We one hot encode the classifications (just using inbuilt methods for generating the new columns and adding them to the original)

```
unique(pdata$site)
```

```
## [1] 1 2 3 4 5 6 7
```

```
pdata$site <- as.factor(pdata$site)
onehot <- model.matrix(~ site - 1, data=pdata)
pdata <- cbind(pdata, onehot)
pdata$site7 <- NULL
```

Drop the site column

```
pdata$site <- NULL
print(pdata[5,])
```

```
##   sex age hdlngth skullw totlngth taill footlgth earconch  eye chest belly
## 5   f   2    91.5   56.3     85.5    36       71     53.2 15.1  28.5    33
##   site1 site2 site3 site4 site5 site6
## 5     1     0     0     0     0     0
```

2c Encode the sex column as a 1,0 for female/. not female

```
pdata$female <- ifelse(pdata$sex == "m", 0, 1)
pdata$sex <- NULL
print(pdata[5,])
```

```
##   age hdlngth skullw totlngth taill footlgth earconch  eye chest belly site1
## 5   2    91.5   56.3     85.5    36       71     53.2 15.1  28.5    33     1
##   site2 site3 site4 site5 site6 female
## 5     0     0     0     0     0      1
```

2d Finally, split the data into training, validation and test sets; Noting that we won't usually have a completely perfect 80,10,10 % split, but we're always within one or two rows. We also pre-emptively remove 'age' to mitigate collisions later on

```
n <- nrow(pdata)
train_ind <- sample(1:n, size = round(n * 0.8))
temp_ind <- setdiff(1:n, train_ind)
test_ind <- sample(temp_ind, size = round(n * 0.1))
validation_ind <- setdiff(temp_ind, test_ind)


df_train <- pdata[train_ind, ]
df_test <- pdata[test_ind, ]
df_val <- pdata[validation_ind, ]

features <- colnames(df_train)
features <- setdiff(colnames(df_train), "age")


cat("Training Set:", nrow(df_train), "rows\n")
```

```
## Training Set: 81 rows
```

```
cat("Validation Set:", nrow(df_val), "rows\n")
```

```
## Validation Set: 10 rows
```

```r
cat("Test Set:", nrow(df_test), "rows\n")
```

```
## Test Set: 10 rows
```

3 We now have a our data with sites delineated by unique columns, sex denoted into a single column and indeed a total of 18 columns, including weight. San's the *age* column. Shortly, we will select the best *greedy* linear regression model. We can generate 2^k permutations of columns (as there is no requirement to include all or any even as this 2^k includes the null model (mean))

Therefore, the entire space of unique models is 131,072 (2^17 18 cols, 1x response, 17x predictor)

4 We will now perform step wise feature selection (forward) to find features which are optimal for our linear regression model.

Broadly; we take the greedy approach; we will begin with the null (mean) model

Firstly; our manual r_square function is defined:

```r
R_Squared <- function(model, data) {
  response_var <- all.vars(formula(model))[1]
  predictions <- predict(model, data)
  residuals <- data[[response_var]] - predictions
  SST <- sum((data[[response_var]] - mean(data[[response_var]]))^2)
  SSE <- sum(residuals^2)

  R_squared <- 1 - (SSE / SST)
  return(R_squared)
}
```

The following executes these broad steps:

We intialise a null model (i.e. the gradient / mean). We then iteratively add features to the *running features*, assess the *R^2* metric as a means of analysis of fit, we train each model on the subset of data and subsequently store the best models, features in the named lists as below.

```r
predictor <- c("1")
best_features <- vector(length = length(features) +1)
selection <- as.formula("age ~ 1") # Explicitly fit a model with just the intercept
fit_models <- lm(formula = selection, data = df_train)
b0 <- coef(fit_models)[1]
mean_age <- mean(df_train$age)
best_features[1] <- predictor
cat("The mean age of the dataset is", mean_age, "and the intercept of the base model is", b0, ", so they
```

```
## The mean age of the dataset is 3.876543 and the intercept of the base model is 3.876543 , so they ma
```

```r
# Initialize the null model with just the intercept
best_features <- vector("list", length = length(features) + 1)
M_models <- vector("list", length = length(features) + 1)
M_features <- vector("list", length = length(features) + 1)
M_r_square <- vector("list", length = length(features) + 1)
```

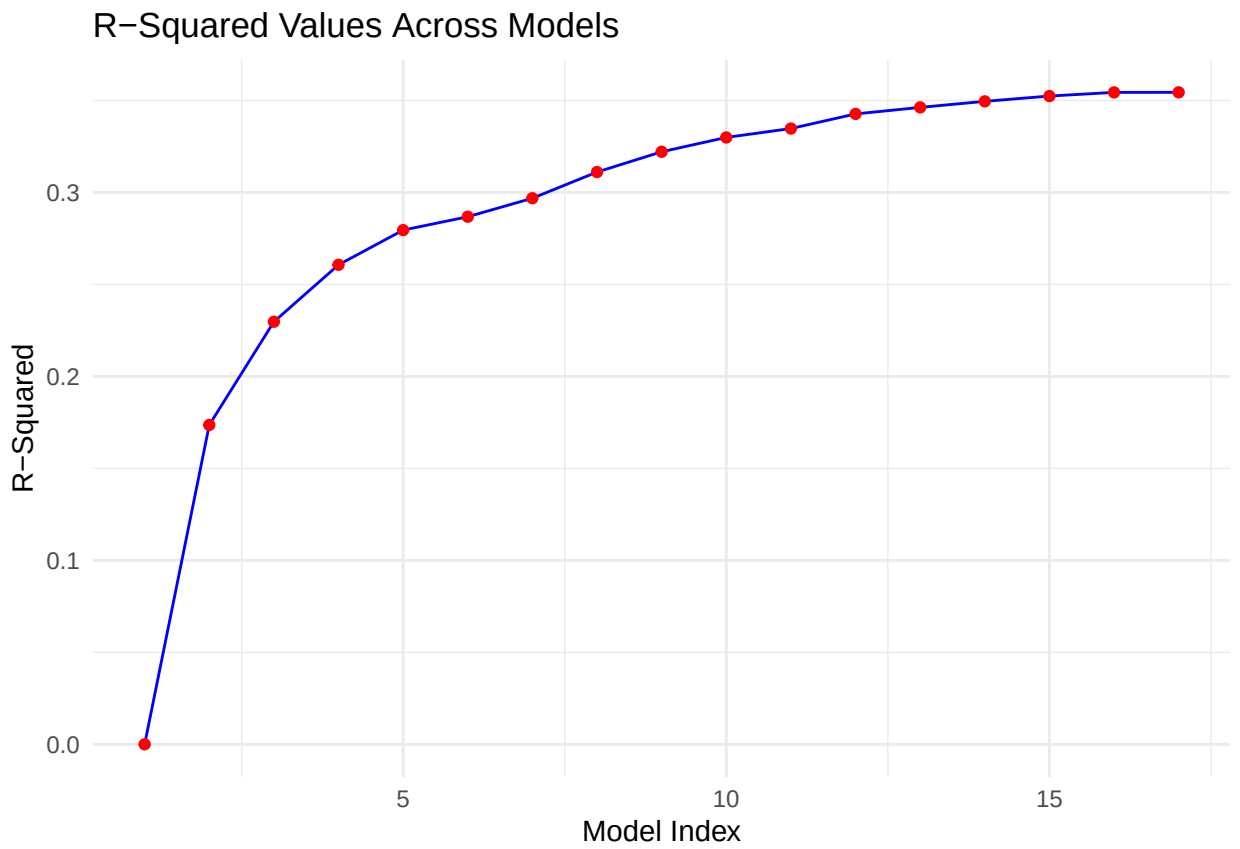
```r
# Start with the null model
null_model <- lm(age ~ 1, data = df_train)
b0 <- coef(null_model)[1]
mean_age <- mean(df_train$age)

best_features[[1]] <- "1"
M_models[[1]] <- null_model
M_features[[1]] <- "Mean Y"
M_r_square[[1]] <- summary(null_model)$r.squared
for (i in 1:length(features)) {
  p <- i
  if (p == 1) {
    predictor <- combn(features, p, simplify = FALSE)
  } else {
    predictor <- Filter(function(x) all(best_features[2:p] %in% x), combn(features, p, simplify = FALSE)
  }

  current_selections <- sapply(predictor, function(x) paste("age", "~", paste0(x, collapse = "+"), sep =
  current_models <- lapply(current_selections, function(x) lm(x, data = df_train))

  R2_of_current_models <- sapply(current_models, function(m) R_Squared(m, df_train))
  index_of_best <- which.max(R2_of_current_models)
  M_models[[p + 1]] <- current_models[[index_of_best]]
  M_r_square[[p + 1]] <- R2_of_current_models[[index_of_best]]

  M_features[[p + 1]] <- current_selections[[index_of_best]]
  best_features[[p + 1]] <- setdiff(predictor[[index_of_best]], best_features[2:p])
}
```

As such we now have *number of features -1* models. Noting that the selection process is greedy wrt R^2, there is no guarantee that our selection is optimal, or rather, that any member of our selection is optimal.

The following are our models as selected based on maximizing R^2....

Note that R^2 continually increases... so it stands to reason that our selected forward model at this stage is the final model.

```r
r2_df <- data.frame(
  Index = seq_along(M_r_square),
  R_Squared = unlist(M_r_square)
)

# Plot using ggplot2
ggplot(r2_df, aes(x = Index, y = R_Squared)) +
  geom_line(color = "blue") +
  geom_point(color = "red") +
  labs(title = "R-Squared Values Across Models",
       x = "Model Index",
       y = "R-Squared") +
  theme_minimal()
```




```
non_null_features <- M_features[!sapply(M_features, is.null)]
model_formulas <- data.frame(Step = seq_along(non_null_features), Formula = unlist(non_null_features),R
knitr::kable(model_formulas, col.names = c("Step", "Model Formula", "Rsquared"), align = "l", caption =
```

Table 2: Stepwise Feature Selection Results

Step	Model Formula	Rsquared
1	Mean Y	0.0000000
2	age ~ chest	0.1736034
3	age ~ chest+site4	0.2297285
4	age ~ eye+chest+site4	0.2606762
5	age ~ totlngth+eye+chest+site4	0.2795438
6	age ~ totlngth+eye+chest+site4+site5	0.2868173
7	age ~ totlngth+earconch+eye+chest+site4+site5	0.2968778
8	age ~ totlngth+footlgth+earconch+eye+chest+site4+site5	0.3111114
9	age ~ totlngth+taill+footlgth+earconch+eye+chest+site4+site5	0.3221274
10	age ~ totlngth+taill+footlgth+earconch+eye+chest+belly+site4+site5	0.3299211
11	age ~ skullw+totlngth+taill+footlgth+earconch+eye+chest+belly+site4+site5	0.3347649
12	age ~ skullw+totlngth+taill+footlgth+earconch+eye+chest+belly+site4+site5+site6	0.3427035
13	age ~ hdlngth+skullw+totlngth+taill+footlgth+earconch+eye+chest+belly+site4+site5+site6	0.3462706
14	age ~ hdlngth+skullw+totlngth+taill+footlgth+earconch+eye+chest+belly+site1+site4+site5+site6	0.3495425

Step	Model Formula	Rsquared
15	age ~ hdl-ngth+skullw+totlngth+taill+footlgth+earconch+eye+chest+belly+site1+site2+site4+site5+site6	0.3524237
16	age ~ hdl-ngth+skullw+totlngth+taill+footlgth+earconch+eye+chest+belly+site1+site2+site3+site4+site5+site6	0.3544083
17	age ~ hdl-ngth+skullw+totlngth+taill+footlgth+earconch+eye+chest+belly+site1+site2+site3+site4+site5+site6+female	0.3544453

The reader may need to cross reference the index in the above table, features against the graph.

5: Our predictors which are related to body measurements and the site variables are the most important for predicting age. The features hdlngth, footlgth, earconch, eye, and body measurements (chest, belly) are consistently associated with higher R-squared values, indicating their significant contribution to explaining the variance in age; making them good all good choices for predictors. From our graph; chest (in some random passes hndlength) is obviously the largest mover of our R2 metric; however it is noted that that is against the mean; so the next most ocntributing feature: *site4* is the most important predictor (assuming we always use a multilinear model)

6:

```
non_null_models <- Filter(Negate(is.null), M_models)

#print(non_null_models)

MSE_val <- sapply(non_null_models, function(model) mean((df_val$age - predict(model, newdata = df_val))



non_null_features <- M_features[!sapply(M_features, is.null)]
mrs <- M_r_square[!sapply(M_r_square, is.null)]
print(R2_of_current_models)
```

```
## [1] 0.3544453
```

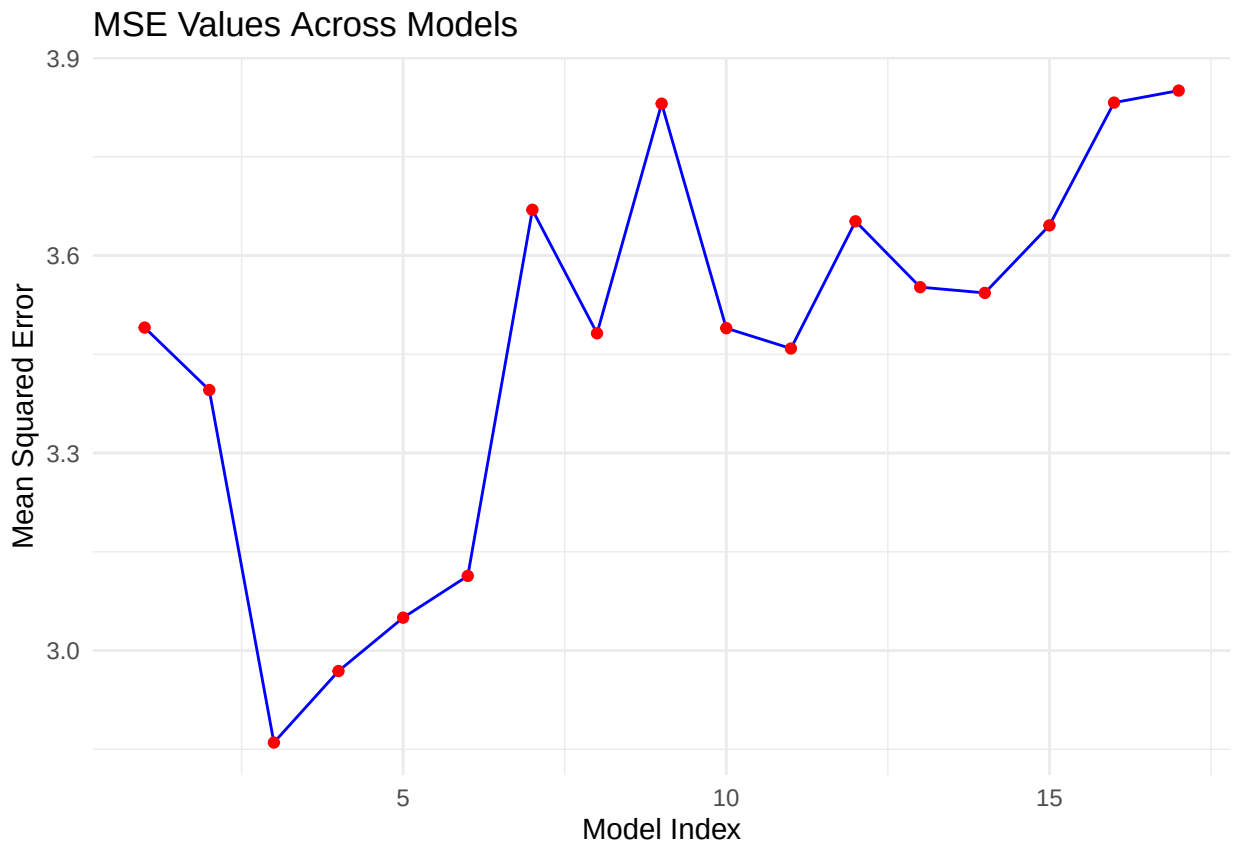
```
model_formulas <- data.frame(Step = seq_along(non_null_features),
                             RSquared = unlist(mrs),
                             MSE = unlist(MSE_val))
knitr::kable(model_formulas, col.names = c("Step",  "R^2 (train)", "MSE (Validation)"), align = "l", ca
```

Table 3: Stepwise Feature Selection Results

Step	R^2 (train)	MSE (Validation)
1	0.0000000	3.490550
2	0.1736034	3.395703
3	0.2297285	2.860291
4	0.2606762	2.968902
5	0.2795438	3.049945
6	0.2868173	3.113354
7	0.2968778	3.669451
8	0.3111114	3.481828

Step	R^2 (train)	MSE (Validation)
9	0.3221274	3.830622
10	0.3299211	3.489618
11	0.3347649	3.458746
12	0.3427035	3.651951
13	0.3462706	3.551932
14	0.3495425	3.543191
15	0.3524237	3.645723
16	0.3544083	3.832216
17	0.3544453	3.850491

```
mse_df <- data.frame(
  Index = seq_along(MSE_val),
  MSE = MSE_val
)

# Plot using ggplot2
ggplot(mse_df, aes(x = Index, y = MSE)) +
  geom_line(color = "blue") +
  geom_point(color = "red") +
  labs(title = "MSE Values Across Models",
       x = "Model Index",
       y = "Mean Squared Error") +
  theme_minimal()
```



Above: The MSE for the test set determines the most accurate model.... In our case it is the (16th) 17th model in the set.

```
# Print MSE values
print(MSE_val)
```

```
##  [1] 3.490550 3.395702 2.860291 2.968902 3.049944 3.113354 3.669451 3.481828
##  [9] 3.830622 3.489618 3.458746 3.651951 3.551932 3.543191 3.645724 3.832216
## [17] 3.850491
```

```
Best_overall_index <- which.min(MSE_val)
Best_overall_model <- M_models[[Best_overall_index]]
Best_overall_model_formula <- M_features[[Best_overall_index]]
coefficients <- coef(Best_overall_model)
intercept <- coefficients[1]
gradient <- coefficients[-1]  # Excluding the intercept

# Print the results with an explanatory sentence
cat("The best overall model based on MSE has the formula:", Best_overall_model_formula, "\n")
```

```
## The best overall model based on MSE has the formula: age ~ chest+site4
```

```
cat("Index of the best model:", Best_overall_index, "\n")
```

```
## Index of the best model: 3
```

```
cat("Intercept:", intercept, "\n")
```

```
## Intercept: -8.64671
```

```
cat("Gradient (coefficients for predictors):\n", paste(names(gradient), gradient, sep="=", collapse="\n
```

```
## Gradient (coefficients for predictors):
##  chest=0.468557418328805
## site4=-1.84240054778786
```

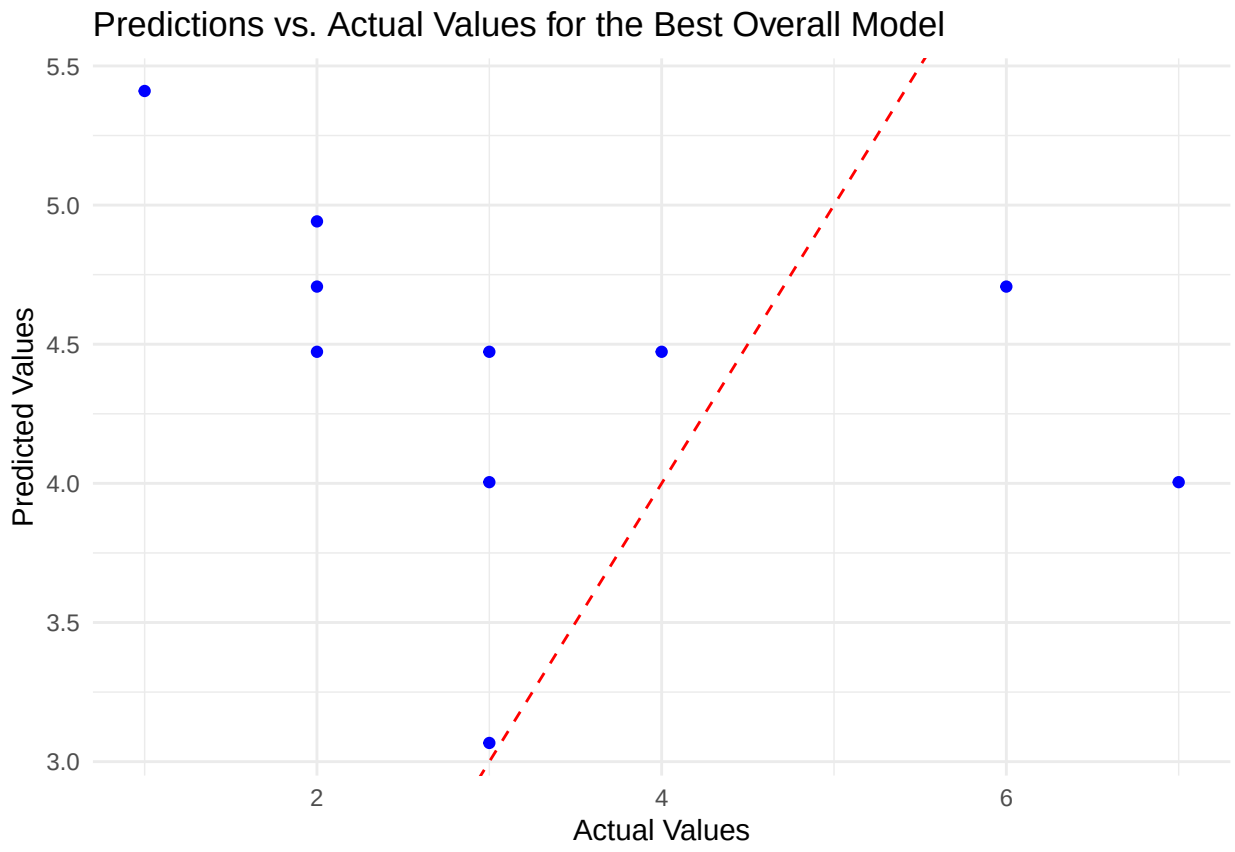
For our validation set; it turns out that the entire dataset is used (augmented from earlier steps). That is, the lowest MSE on our datset is given by the set of all features as predictors.

So; with this in mind; our final measure of performance is of the test set:

```
preds_test <- predict(Best_overall_model, newdata = df_test)
residuals <- df_test$age - preds_test
mse_test <- mean(residuals^2)
cat("The MSE of the best overall model on the test set is:", mse_test, "\n")
```

```
## The MSE of the best overall model on the test set is: 5.559605
```

```r
preds_vs_actual_df <- data.frame(
  Actual = df_test$age,
  Predicted = preds_test
)

# Plot predictions vs actual values
ggplot(preds_vs_actual_df, aes(x = Actual, y = Predicted)) +
  geom_point(color = "blue") +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed", color = "red") +
  labs(title = "Predictions vs. Actual Values for the Best Overall Model",
       x = "Actual Values",
       y = "Predicted Values") +
  theme_minimal()
```



OUR MSE on test is *Huge*.

Essentially, making the selected model useless; there are intuitive choices that would better fit our test set (ie a line rotated say 30d to the clockwise). Again, our test set is very small; the MSE of our train data is not particularly close to this and it stands that using other mixes of test / train would generalize much better (randomization). While other implementations or approaches use MSE across the board, the R2 approach within the train / val set better estimates explainable variance as opposed to objective performance.

/fin