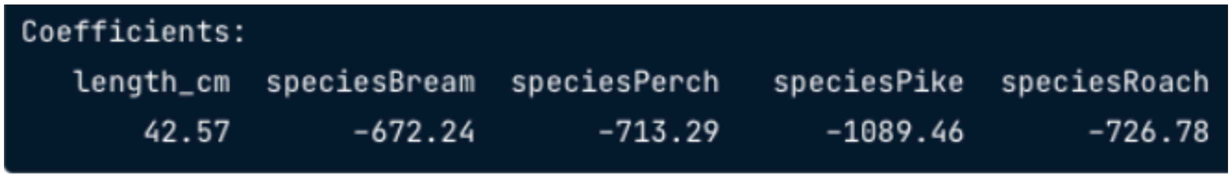
Intermediate Regression in R
by Richie Cotton and datacamp

Multiple regression is a regression model with more than one explanatory variable
more explanatory variables have the potential to give more insight and better predictions

Parallel slopes linear regression is a special case of regression that involves one categorical and one numerical variable
reminder we run a linear regression in R by calling 'lm'
which passes a formula and a data frame
the formula has the response variable on the left followed by a tilde and then on the right has the explanatory variable
*remember with categorical variables to put + 0 after the explanatory variable to make the coefficients easier to read
the + 0 tells R not to include an intercept in the model

Now an example using the fish dataset with multiple explanatory variables including numeric and categorical
lm(mass_g ~ length_cm + species + 0, data = fish)
output>



```
Coefficients:
   length_cm  speciesBream  speciesPerch   speciesPike  speciesRoach
       42.57       -672.24       -713.29      -1089.46       -726.78
```

Visualization for one numeric explanatory variable > scatter plot with geom_point and geom_smooth
Visualization for one categorical explanatory variable > muliple options but simples is the boxplot is a good option >
model coeffs are the means of each category
example in R
#stat_summary method with fun.y argument allows us to visualize summary statistic elements
#'shape' argument allows you to change the shape of the point that represents the summary statistic, 15 makes the shape square

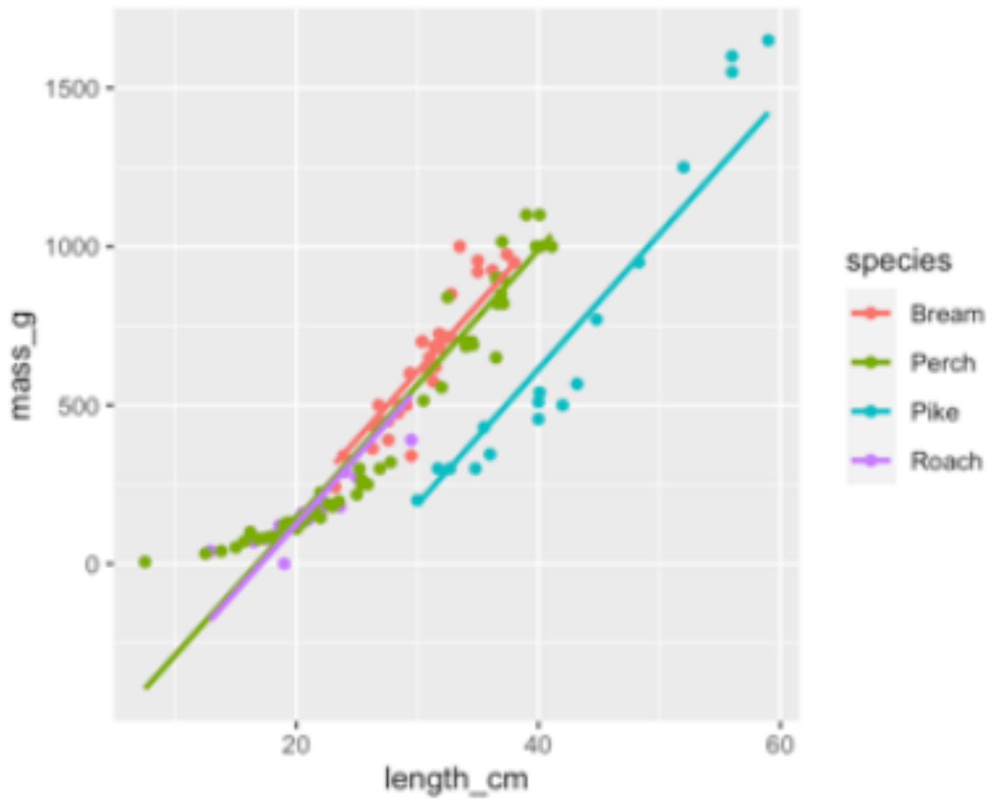
```
ggplot(fish, aes(species, mass_g)) +
        geom_boxplot() +
        stat_summary(fun.y = mean, shape = 15)
```

Visualization with a numeric and a categorical explanatory variable (parallel slopes

regression) > also draw a scatter plot using the moderndive library
example

```
ggplot(fish, aes(length_cm, mass_g, color = species)) +
	geom_point() +
	geom_parallel_slopes(se = FALSE)
```

output>

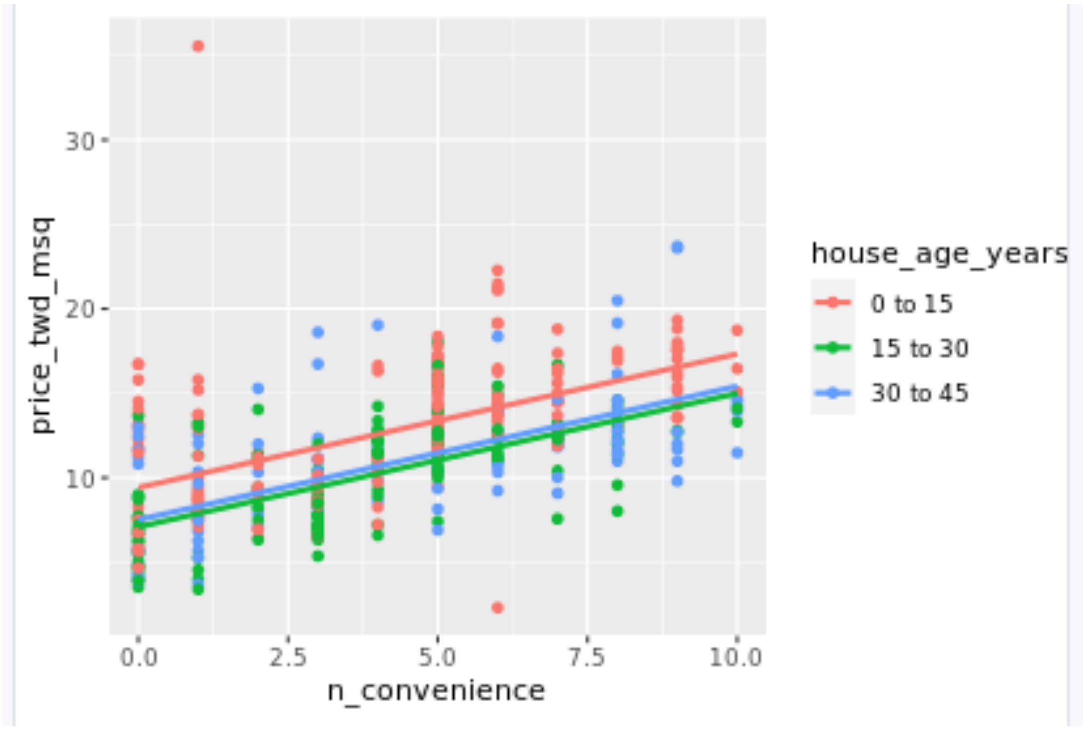


*can see here why it got its name parallel slopes regression

Example

```
# Using taiwan_real_estate, plot price_twd_msq vs. n_convenience colored by house_age_years
ggplot(taiwan_real_estate, aes(n_convenience, price_twd_msq,
color=house_age_years)) +
  # Add a point layer
  geom_point() +
  # Add parallel slopes, no ribbon
  geom_parallel_slopes(se = FALSE)
```

output>

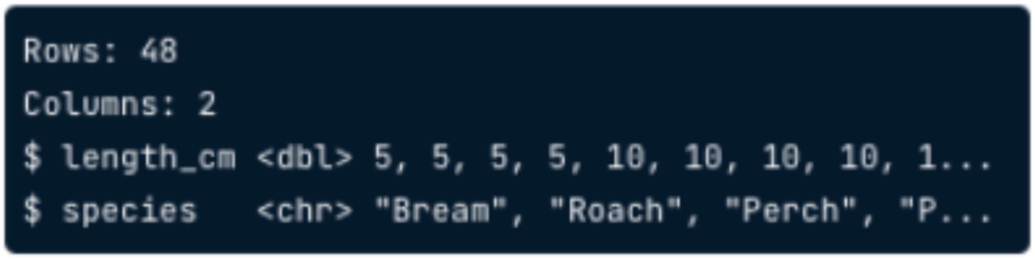


Predicting parallel slopes
prediction workflow in R for single explanatory variable:
#pick any values you want and store them in a data frame or tibble
#here we have chosen a sequence of lengths from 5 to 60cm in steps of 5cm

```
library(dplyr)
explanatory_data <- tibble(
        length_cm = seq(5, 60, 5)
glimpse(explanatory_data)
```

Prediction flow for multiple explanatory variables:
we use the same process but we use expand_grid from the tidyr package
*this returns a data frome of all combinations of its inputs
example

```
library(dplyr)
library(tidyr)
explanatory_data <- expand_grid(
        length_cm = seq(5, 60, 5),
        species = unique(fish$species))
output>
```



```
Rows: 48
Columns: 2
$ length_cm <dbl> 5, 5, 5, 5, 10, 10, 10, 10, 1...
$ species   <chr> "Bream", "Roach", "Perch", "P...
```

next step in the workflow (exact same for single or multiple, just make sure to use the proper model name)

```
prediction_data <- explanatory_data %>%
        mutate(mass_g = predict(
                mdl_mass_vs_both, explanatory_data))
```

Visualizing the predictions

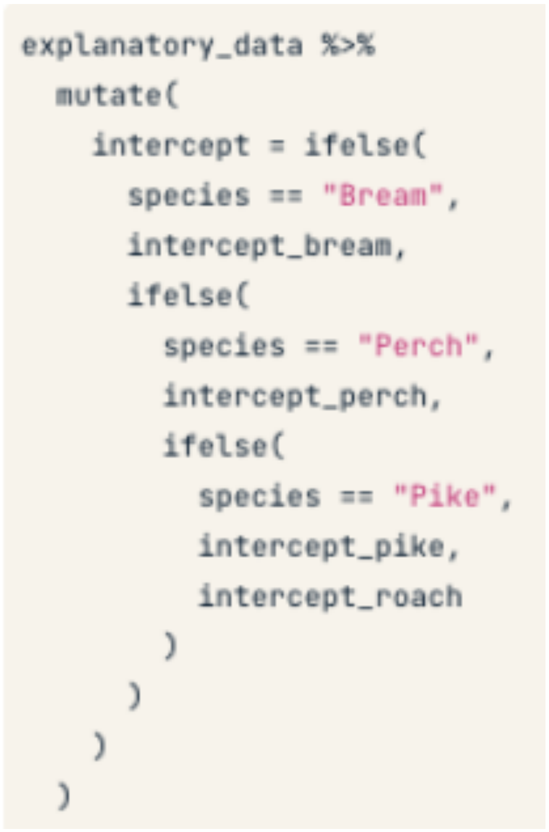
```
ggplot(fish, aes(length_cm, mass_g, color=species)) +
        geom_point() +
        geom_parallel_slopes(se = FALSE) +
        geom_point(
                data=prediction_data,
                size=3, shape=15)
```

Calculating predictions

coefficients method extracts the coefficients

```
coeffs <- coefficient(mdl_price_vs_length)
#extract intercept and slope further to single variables
#remember index starts at 1 not 0
intercept <- coeffs[1]
slope <- coeffs[2]
#response value is the intercept plus the slope times the explanatory variable
explanatory_data %>%
        mutate(
                mass_g = intercept + slope * length_cm)
```

A small side - choosing an intercept with ifelse()



```
explanatory_data %>%
  mutate(
    intercept = ifelse(
      species == "Bream",
      intercept_bream,
      ifelse(
        species == "Perch",
        intercept_perch,
        ifelse(
          species == "Pike",
          intercept_pike,
          intercept_roach
        )
      )
    )
  )
```

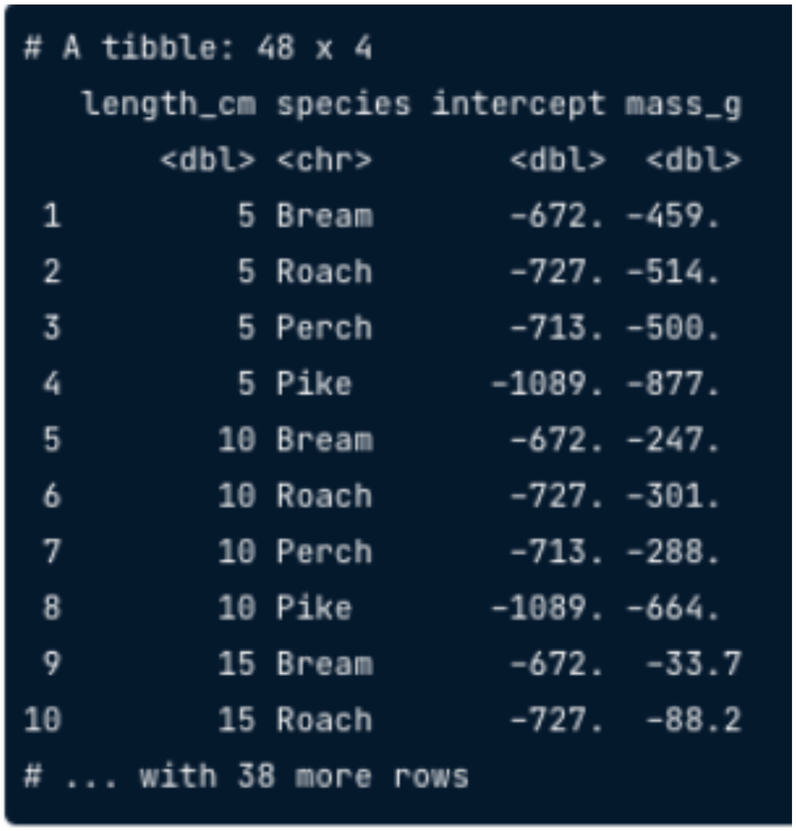
dplyr simplies the process for multiple explanatory variables
we use the dplyr function case_when()
works as a formula
on the left-hand side you have a logical condition and on the right you have the value to give to those rows where the condition is met
example:

```
dataframe %>%
	mutate(case_when(condition_1 ~ value_1, condition_2 ~ value_2, #… and so on))
```

example with the fish dataset:

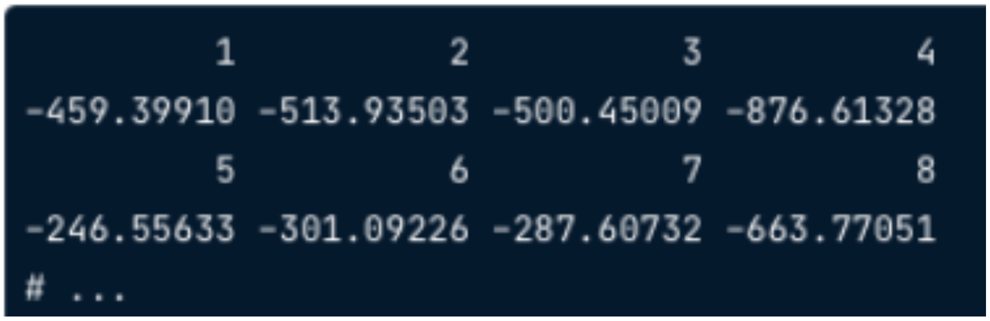
```
explanatory_data %>%
	mutate(intercept = case_when(
		species == ‘Bream’ ~ intercept_bream,
		species == ‘Perch’ ~ intercept_perch,
		species == ‘Roach’ ~ intecept_roach),
#finally calculate the response
	mass_g = intercept + slope * length_cm)
```

output>



```
# A tibble: 48 x 4
   length_cm species intercept mass_g
       <dbl> <chr>       <dbl>  <dbl>
 1         5 Bream       -672. -459.
 2         5 Roach       -727. -514.
 3         5 Perch       -713. -500.
 4         5 Pike       -1089. -877.
 5        10 Bream       -672. -247.
 6        10 Roach       -727. -301.
 7        10 Perch       -713. -288.
 8        10 Pike       -1089. -664.
 9        15 Bream       -672.  -33.7
10        15 Roach       -727.  -88.2
# ... with 38 more rows
```

*the intercept is different for different rows
#can use predict() to confirm values
predict(mdl_mass_vs>both, explanatory_data)
output>

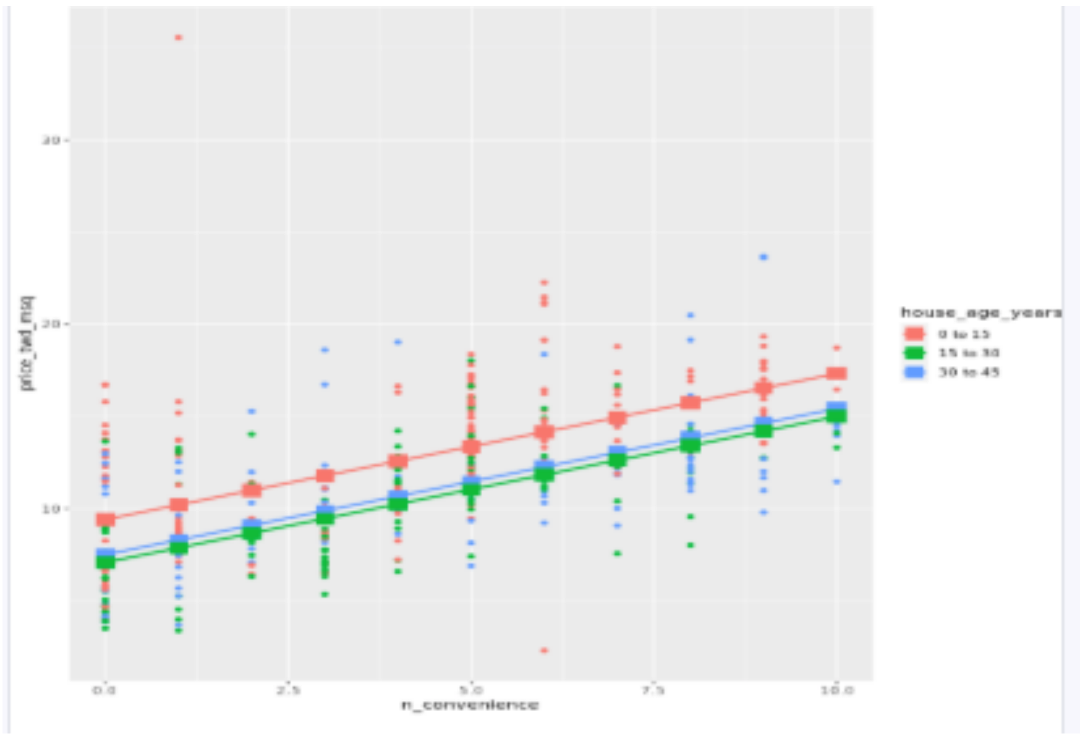


```
         1          2          3          4
-459.39910 -513.93503 -500.45009 -876.61328
         5          6          7          8
-246.55633 -301.09226 -287.60732 -663.77051
# ...
```

Example
# Make a grid of explanatory data
explanatory_data <- expand_grid(
  # Set n_convenience to zero to ten
  n_convenience = seq(0, 10, 1),
  # Set house_age_years to the unique values of that variable
  house_age_years = unique(taiwan_real_estate$house_age_years)
)

```
prediction_data <- explanatory_data %>%
  mutate(
    price_twd_msq = predict(mdl_price_vs_both, explanatory_data)
  )

taiwan_real_estate %>%
  ggplot(aes(n_convenience, price_twd_msq, color = house_age_years)) +
  geom_point() +
  geom_parallel_slopes(se = FALSE) +
  # Add points using prediction_data, with size 5 and shape 15
  geom_point(
    data=prediction_data,
    size=5, shape=15
  )
```

output>

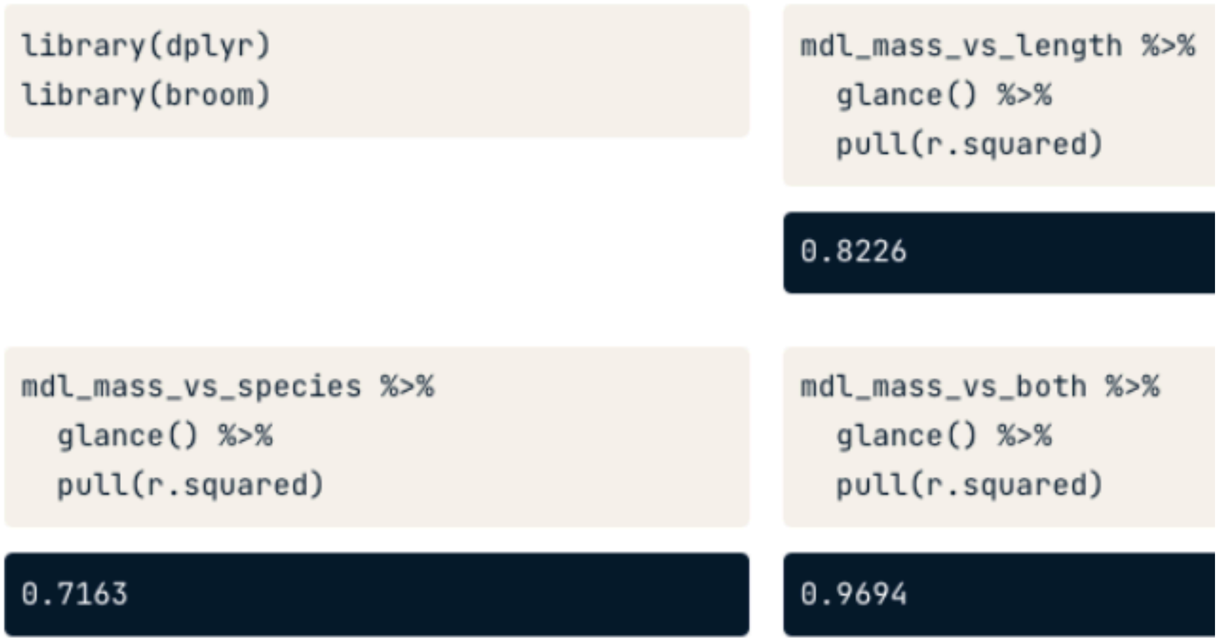


Example

```
coeffs <- coefficients(mdl_price_vs_both)
slope <- coeffs[1]
intercept_0_15 <- coeffs[2]
intercept_15_30 <- coeffs[3]
intercept_30_45 <- coeffs[4]
```

```
prediction_data <- explanatory_data %>%
  mutate(
    # Consider the 3 cases to choose the intercept
    intercept = case_when(
      house_age_years == '0 to 15' ~ intercept_0_15,
      house_age_years == '15 to 30' ~ intercept_15_30,
      house_age_years == '30 to 45' ~ intercept_30_45
    ),

    # Manually calculate the predictions
    price_twd_msq = intercept + slope * n_convenience
  )

# See the results
prediction_data
```

Assessing model performance
common metrics
Coefficient of determination (R-squared) > how well the linear regression line fits the observed values
 - larger is better
Residual standard error (RSE) > the typical size of the residuals
 - smaller is better
example



```
library(dplyr)
library(broom)
```

```
mdl_mass_vs_length %>%
  glance() %>%
  pull(r.squared)
```

```
0.8226
```

```
mdl_mass_vs_species %>%
  glance() %>%
  pull(r.squared)
```

```
0.7163
```

```
mdl_mass_vs_both %>%
  glance() %>%
  pull(r.squared)
```

```
0.9694
```

*we can see here the sometime advantage of multiple explanatory variables > the model with both shows the highes coefficient of determination (R^2) with 1 being the best fit and 0 being no correlation at all

*always remember though there is still an art here because adding too many explanatory variables can lead to overfitting
reminder on overfitting > where a model is optimized for that particular dataset but doesn't properly reflect the general population

Adjusted coefficient of determination > penalizes more explanatory variables

$$\bar{R}^2 = 1 - (1 - R^2)\frac{n_{obs}-1}{n_{obs}-n_{var}-1}$$

takes in coeff of determination, number of observations, and number of explanatory variables including interactions
Penalty is noticeable when R^2 is small, or nvar is large fraction of nobs
example:

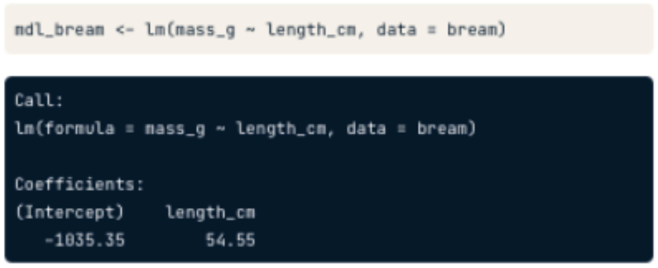
```
mdl_mass_vs_speicies %>%
        glance() %>%
        select(adj.r.squared)
```

Getting RSE

```
mdl_mass_vs_both %>%
        glance() %>%
        pull(sigma)
```

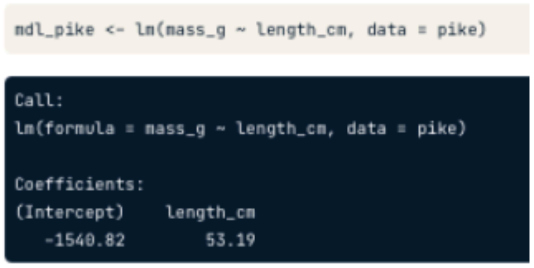
Models for each category
splitting the dataset (multiple ways)
base-R > split() + lapply()
dplyr > nest_by() + mutate()
manually >

```
bream <- fish %>%
        filter(species == "Bream')
perch <- fish %>%
        filter(species == 'Perch')
pike <- fish %>%
        filter(species == 'Pike')
roach <- fis %>%
        filter(species == 'Roach')

#run the 4 models
```



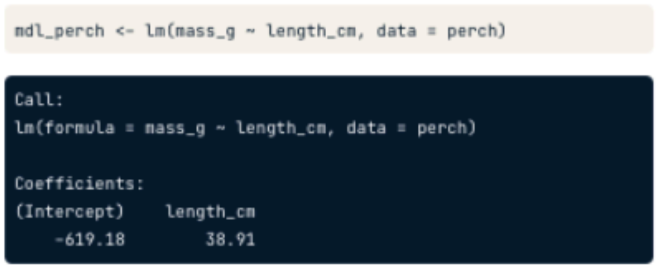
```
mdl_bream <- lm(mass_g ~ length_cm, data = bream)
```

```
Call:
lm(formula = mass_g ~ length_cm, data = bream)

Coefficients:
(Intercept)    length_cm
   -1035.35        54.55
```



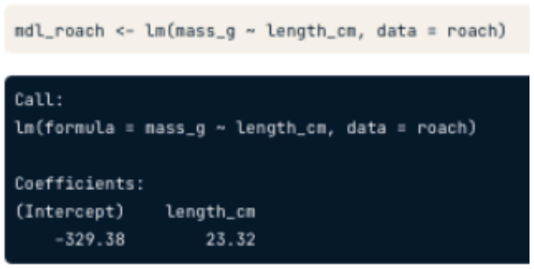
```
mdl_pike <- lm(mass_g ~ length_cm, data = pike)
```

```
Call:
lm(formula = mass_g ~ length_cm, data = pike)

Coefficients:
(Intercept)    length_cm
   -1540.82        53.19
```



```
mdl_perch <- lm(mass_g ~ length_cm, data = perch)
```

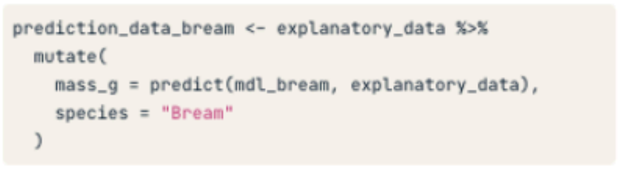
```
Call:
lm(formula = mass_g ~ length_cm, data = perch)

Coefficients:
(Intercept)    length_cm
    -619.18        38.91
```



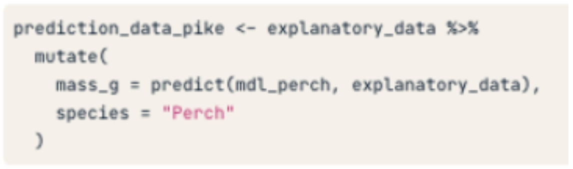
```
mdl_roach <- lm(mass_g ~ length_cm, data = roach)
```

```
Call:
lm(formula = mass_g ~ length_cm, data = roach)

Coefficients:
(Intercept)    length_cm
    -329.38        23.32
```

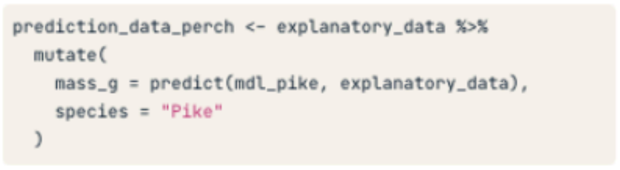
#create a dataframe of explanatory variables
#only need to write this code once because in this example each model has the same explanatory variable
explanatory_data <- tibble(length_cm = seq(5, 60, 5))
#making predictions



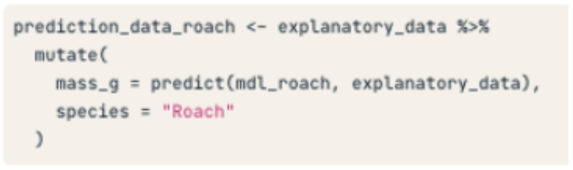
```
prediction_data_bream <- explanatory_data %>%
  mutate(
    mass_g = predict(mdl_bream, explanatory_data),
    species = "Bream"
  )
```



```
prediction_data_pike <- explanatory_data %>%
  mutate(
    mass_g = predict(mdl_perch, explanatory_data),
    species = "Perch"
  )
```

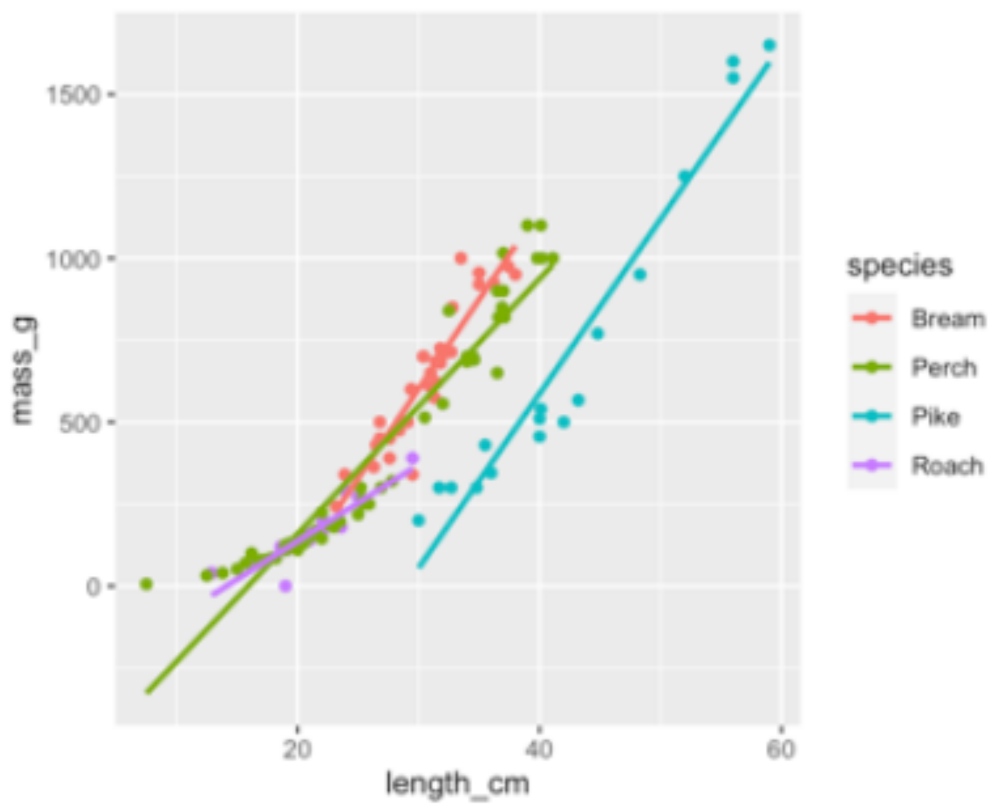


```
prediction_data_perch <- explanatory_data %>%
  mutate(
    mass_g = predict(mdl_pike, explanatory_data),
    species = "Pike"
  )
```

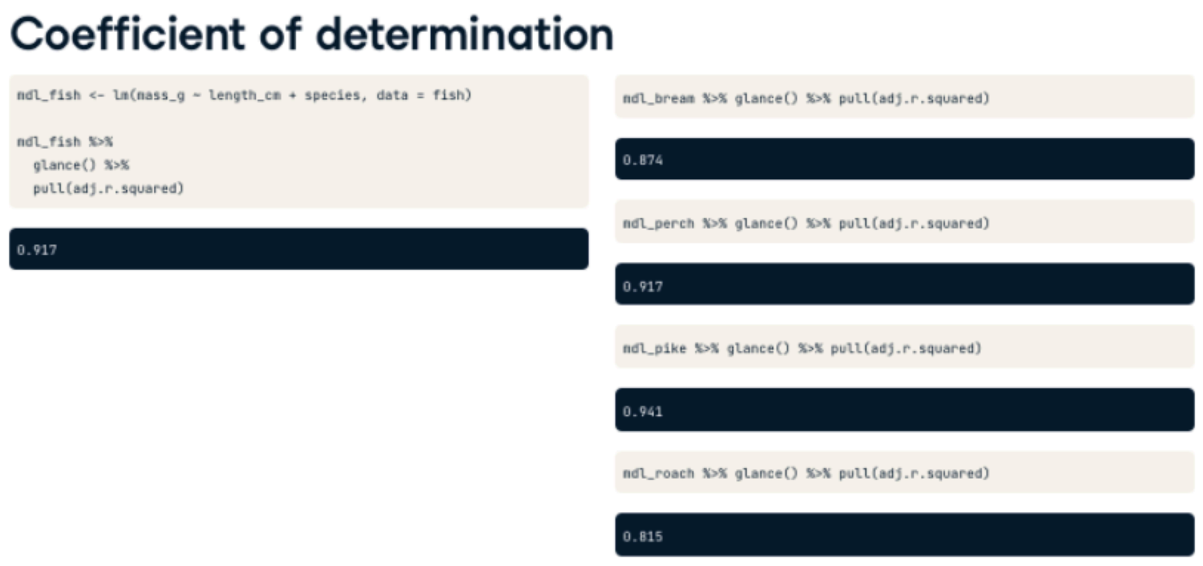


```
prediction_data_roach <- explanatory_data %>%
  mutate(
    mass_g = predict(mdl_roach, explanatory_data),
    species = "Roach"
  )
```

#to make plotting the code easier, include the species in each prediction dataset
#visualizing predictions
#use the color parameter to give each line its own slope
ggplot(fish, aes(length_cm, mass_g, color = species)) +
    geom_point() +
    geom_smooth(method = 'lm', se = FALSE)
output>



Comparing models



Coefficient of determination

```
mdl_fish <- lm(mass_g ~ length_cm + species, data = fish)

mdl_fish %>%
  glance() %>%
  pull(adj.r.squared)
```

```
0.917
```

```
mdl_bream %>% glance() %>% pull(adj.r.squared)
```

```
0.874
```

```
mdl_perch %>% glance() %>% pull(adj.r.squared)
```

```
0.917
```

```
mdl_pike %>% glance() %>% pull(adj.r.squared)
```

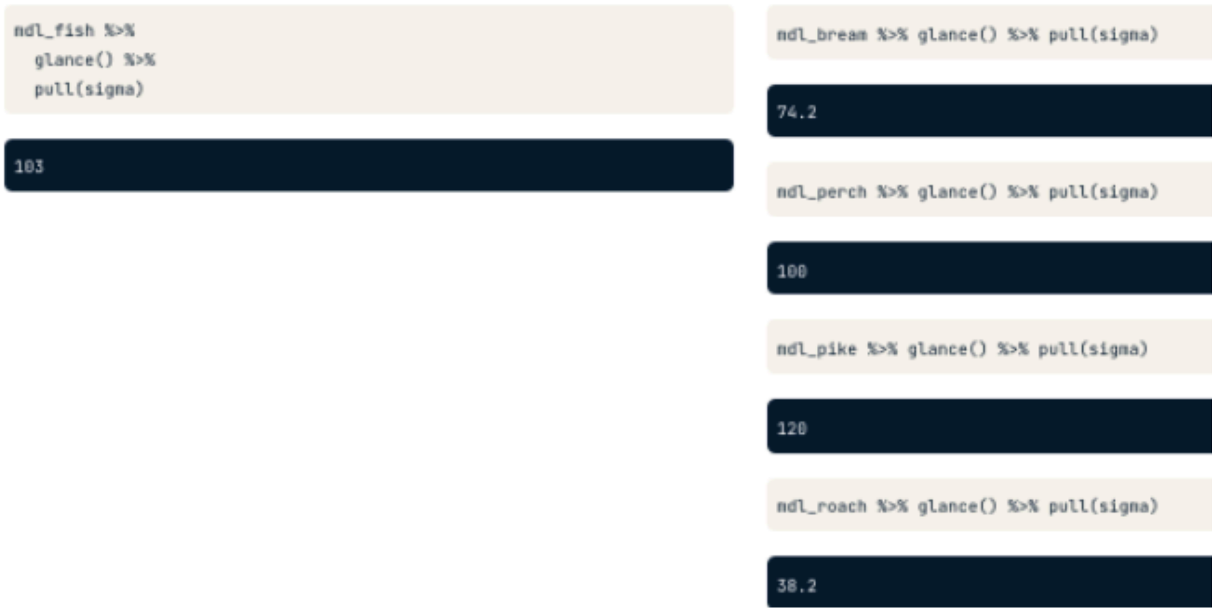
```
0.941
```

```
mdl_roach %>% glance() %>% pull(adj.r.squared)
```

```
0.815
```

*here we can see that perch is equivalent, pike is actually better, but roach and bream are lower
with this info we could say that the model with multiple explanatory variables is better

On our models RSE shows a different picture



```
mdl_fish %>%
  glance() %>%
  pull(sigma)
```

```
103
```

```
mdl_bream %>% glance() %>% pull(sigma)
```

```
74.2
```

```
mdl_perch %>% glance() %>% pull(sigma)
```

```
100
```

```
mdl_pike %>% glance() %>% pull(sigma)
```

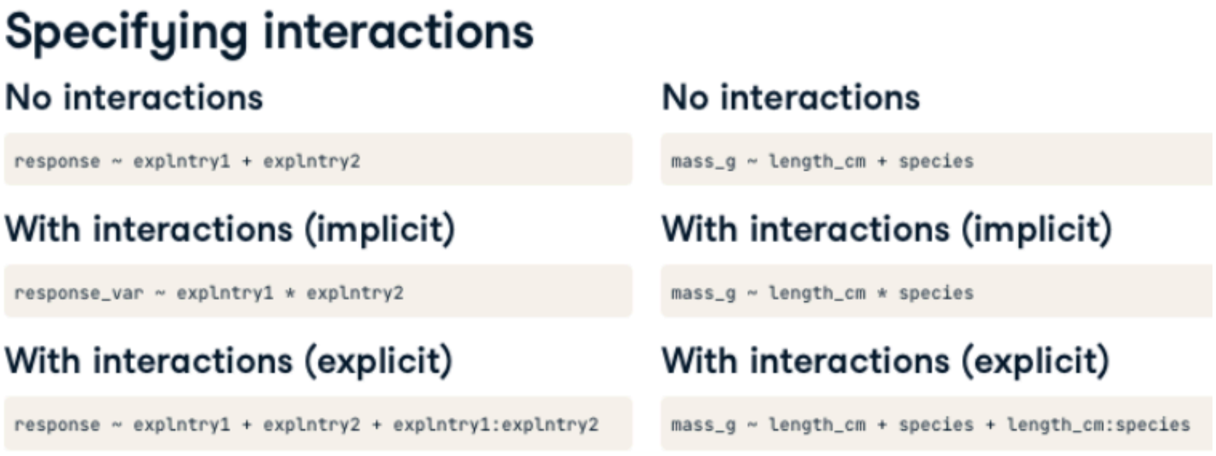
```
120
```

```
mdl_roach %>% glance() %>% pull(sigma)
```

```
38.2
```

*what this tells us is that the whole dataset model benefits from the increased power of more rows of data, whereas individual models benefit from not having to satisfy differing components of data

One model with an interaction
what is an interaction?
the effect of one explanatory variable on the expected response has different values dependent on the values of another explanatory variable, then those two explanatory variables interact
example using the fish dataset > effect of length on the expected mass is different for different species



Specifying interactions

No interactions

```
response ~ explntry1 + explntry2
```

With interactions (implicit)

```
response_var ~ explntry1 * explntry2
```

With interactions (explicit)

```
response ~ explntry1 + explntry2 + explntry1:explntry2
```

No interactions

```
mass_g ~ length_cm + species
```

With interactions (implicit)

```
mass_g ~ length_cm * species
```

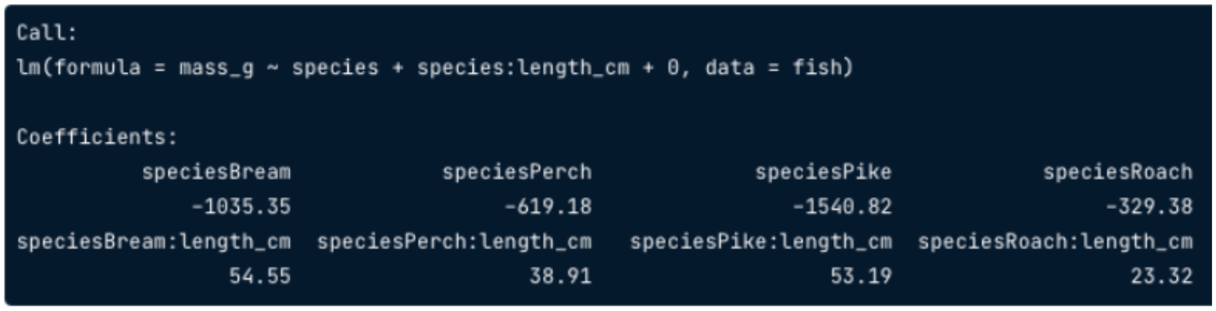
With interactions (explicit)

```
mass_g ~ length_cm + species + length_cm:species
```

*to include an interaction between those variables, you simply swap the plus for a times
above implicit model allows R to figure out the interaction on its own

for more detail the explicit model documents which interactions are included in the model
*the result is exactly the same

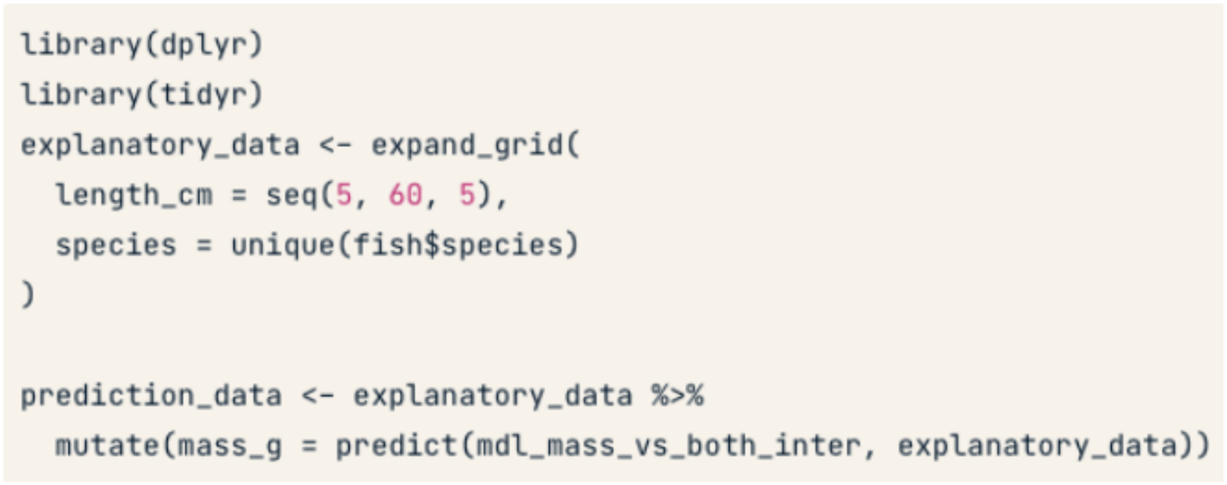
For an easier to read output of our model, we need to run the model with this code:
mdl_inter <- lm(mass_g ~ species + species:length_cm + 0, data = fish)
#right side of the formula shows the categorical explanatory variable then an interaction between two explanatory variables, then 0 to remove the global intercept
output>



```
Call:
lm(formula = mass_g ~ species + species:length_cm + 0, data = fish)

Coefficients:
         speciesBream           speciesPerch            speciesPike           speciesRoach
             -1035.35                -619.18               -1540.82                -329.38
speciesBream:length_cm  speciesPerch:length_cm   speciesPike:length_cm  speciesRoach:length_cm
                54.55                  38.91                  53.19                  23.32
```

now our output gives us an intercept coefficient for each species shown on the top row and a slope coefficient for each species shown on the bottom row
*this output is the same as our above when we did all four models separately
*with interactions you get the convenience of not having to manage four sets of code

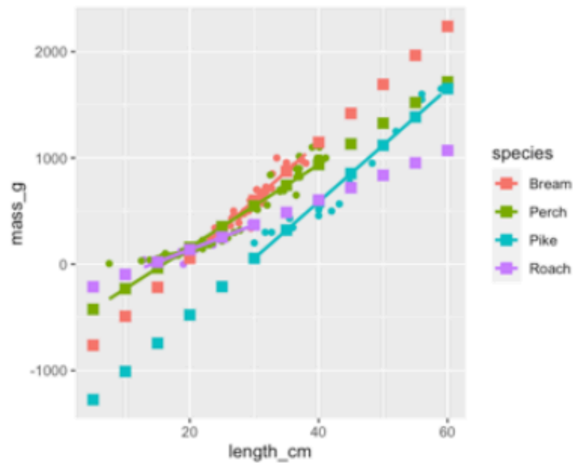
Making predictions with interactions



```
library(dplyr)
library(tidyr)
explanatory_data <- expand_grid(
  length_cm = seq(5, 60, 5),
  species = unique(fish$species)
)

prediction_data <- explanatory_data %>%
  mutate(mass_g = predict(mdl_mass_vs_both_inter, explanatory_data))
```

*same as our code without interactions > R automatically takes care of the interaction
#visualize - same as without interactions

```
ggplot(fish, aes(length_cm, mass_g, color = species)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  geom_point(data = prediction_data, size = 3, shape = 15)
```



#like above get the coefficients and square bracket them out

```
coeffs <- coefficients(mdl_mass_vs_both_inter)
```

```
          speciesBream           speciesPerch            speciesPike           speciesRoach 
           -1035.34757             -619.17511            -1540.82427             -329.37621 
speciesBream:length_cm speciesPerch:length_cm  speciesPike:length_cm speciesRoach:length_cm 
              54.54998               38.91147               53.19487               23.31926 
```

```
intercept_bream <- coeffs[1]
intercept_perch <- coeffs[2]
intercept_pike <- coeffs[3]
intercept_roach <- coeffs[4]
```

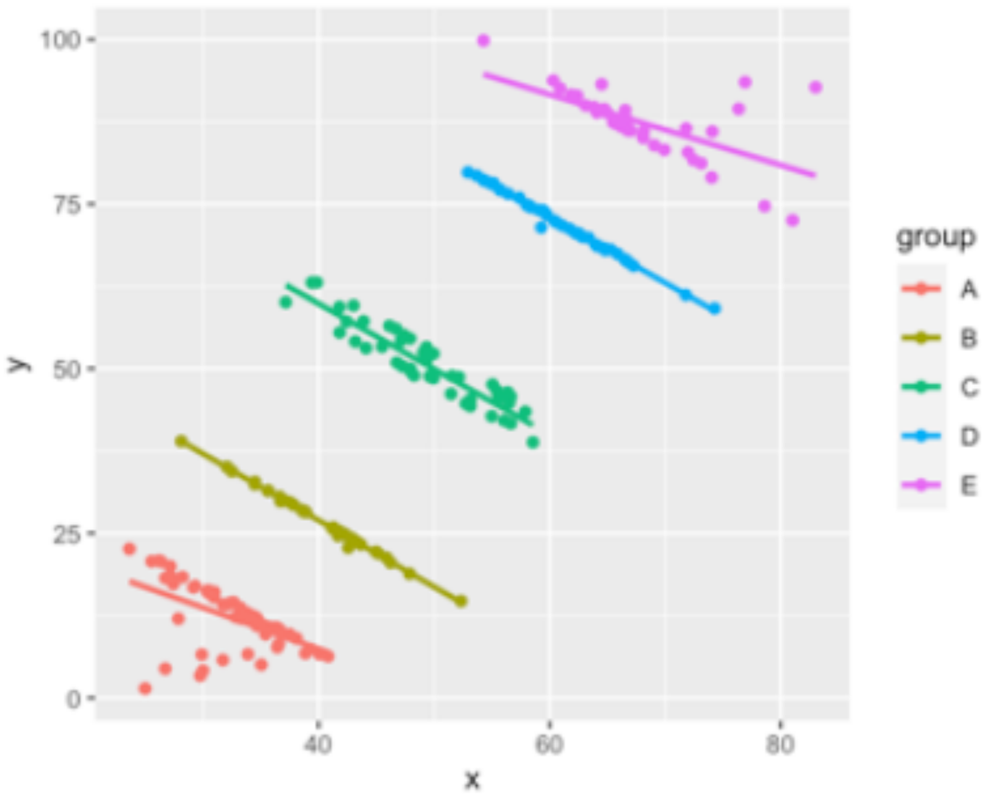
```
slope_bream <- coeffs[5]
slope_perch <- coeffs[6]
slope_pike <- coeffs[7]
slope_roach <- coeffs[8]
```

#calculating the predictions - use the case_when function

```
explanatory_data %>%
  mutate(
    mass_g = case_when(
      species == "Bream" ~ intercept_bream + slope_bream * length_cm,
      species == "Perch" ~ intercept_perch + slope_perch * length_cm,
      species == "Pike" ~ intercept_pike + slope_pike * length_cm,
      species == "Roach" ~ intercept_roach + slope_roach * length_cm
    )
  )
```

Simpson's Pardaox
occurs when the trend of a model on the whole dataset is very different from the trends shown by models on subsets of the dataset
trend = slope coefficient

example > whole dataset model shows positive slope but each explanatory variable shows negative slope
how? > visualize



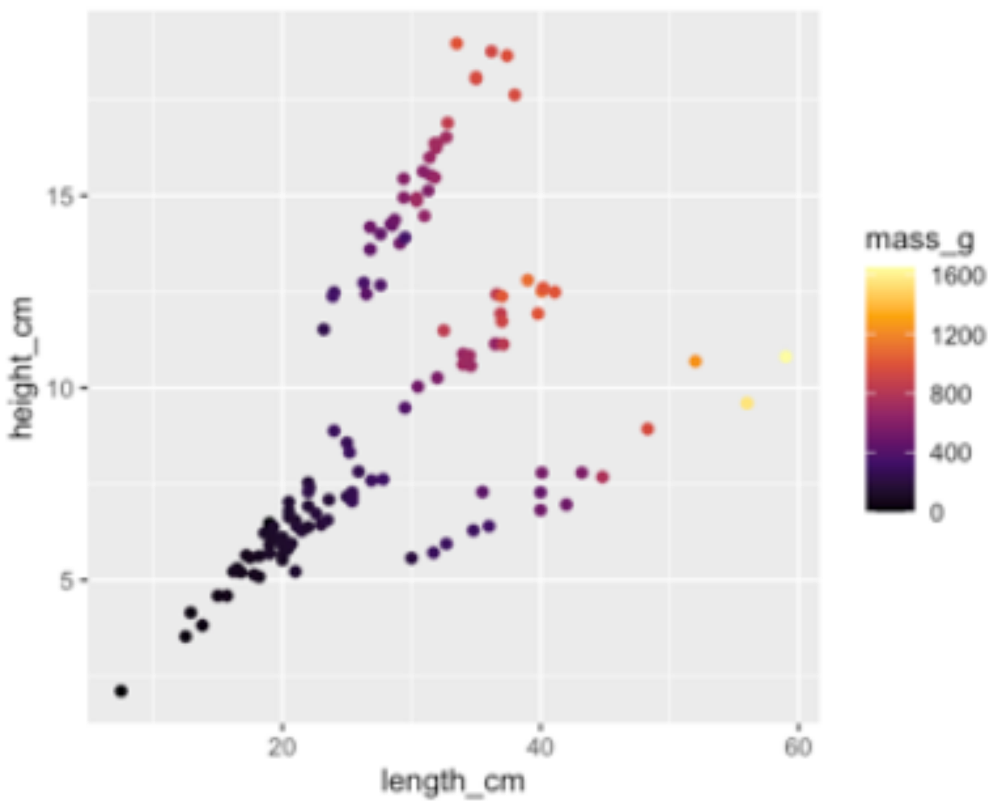
Two numeric explanatory variables
scatter plots are designed to show relationships between two numeric variables
here we have two numeric explanatory variables and one numeric response variable
showing this relationship visually becomes challenging
options are a 3D scatter plot or a 2D scatter plot with response as color

3D scatter plot with R

```
library(plot3D)
scatter3D(fish$lenght_cm, fish$height_cm, fish$mass_g)
#another way that can be more streamline is to use the magrittr package allows us to minimize $ signs
library(magrittr)
fish %$%
	scatter3D(length_cm, height_cm mass_g)
```

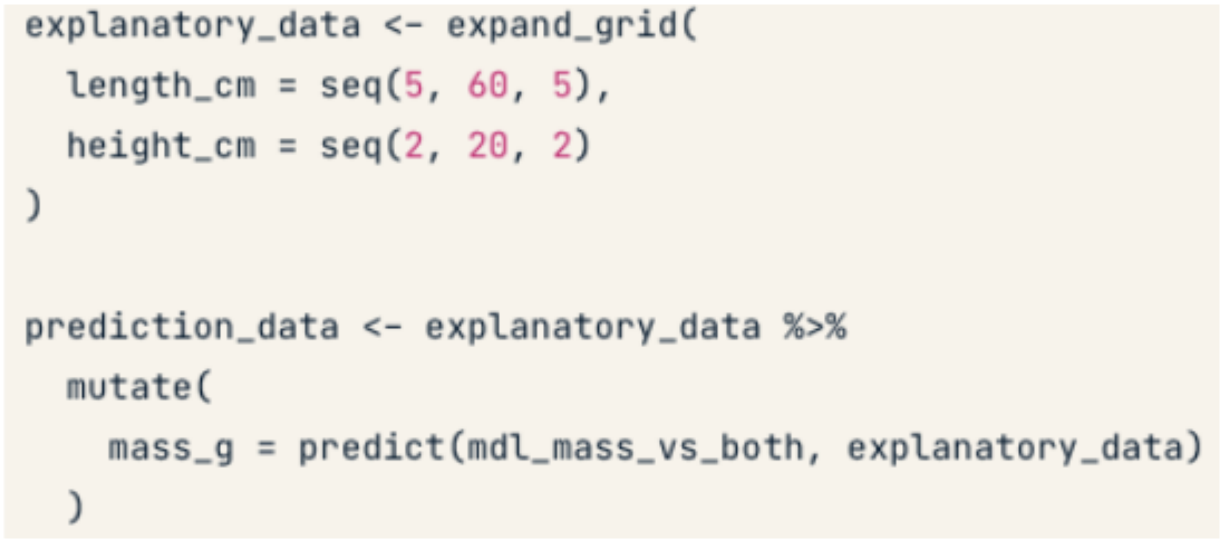
2D scatter plot using color parameter for response variable

```
ggplot(fish, aes(length_cm, height_cm, color=mass_g)) +
	geom_point() +
	#for better color scale
	scale_color_viridis_c(option = 'inferno')
output>
```



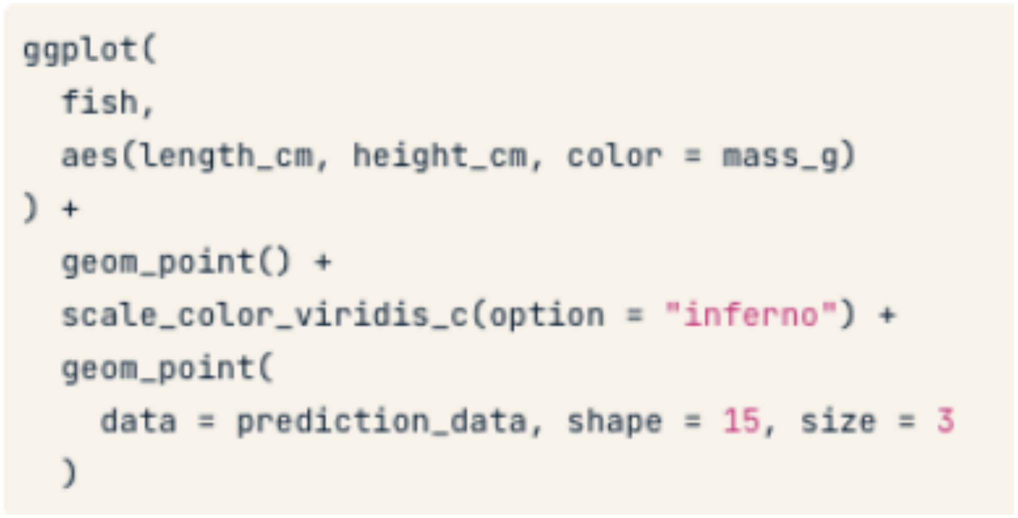
Modeling with two numeric explanatory variables
lm(formula = response variable ~ explanatory + explanatory, data = df)
example with fish dataset:
mdl_mass_vs_both <- lm(mass_g ~ length_cm + height_cm, data = fish)

Prediction flow is the same



```
explanatory_data <- expand_grid(
  length_cm = seq(5, 60, 5),
  height_cm = seq(2, 20, 2)
)

prediction_data <- explanatory_data %>%
  mutate(
    mass_g = predict(mdl_mass_vs_both, explanatory_data)
  )
```

Plotting the predictions



```
ggplot(
  fish,
  aes(length_cm, height_cm, color = mass_g)
) +
  geom_point() +
  scale_color_viridis_c(option = "inferno") +
  geom_point(
    data = prediction_data, shape = 15, size = 3
  )
```

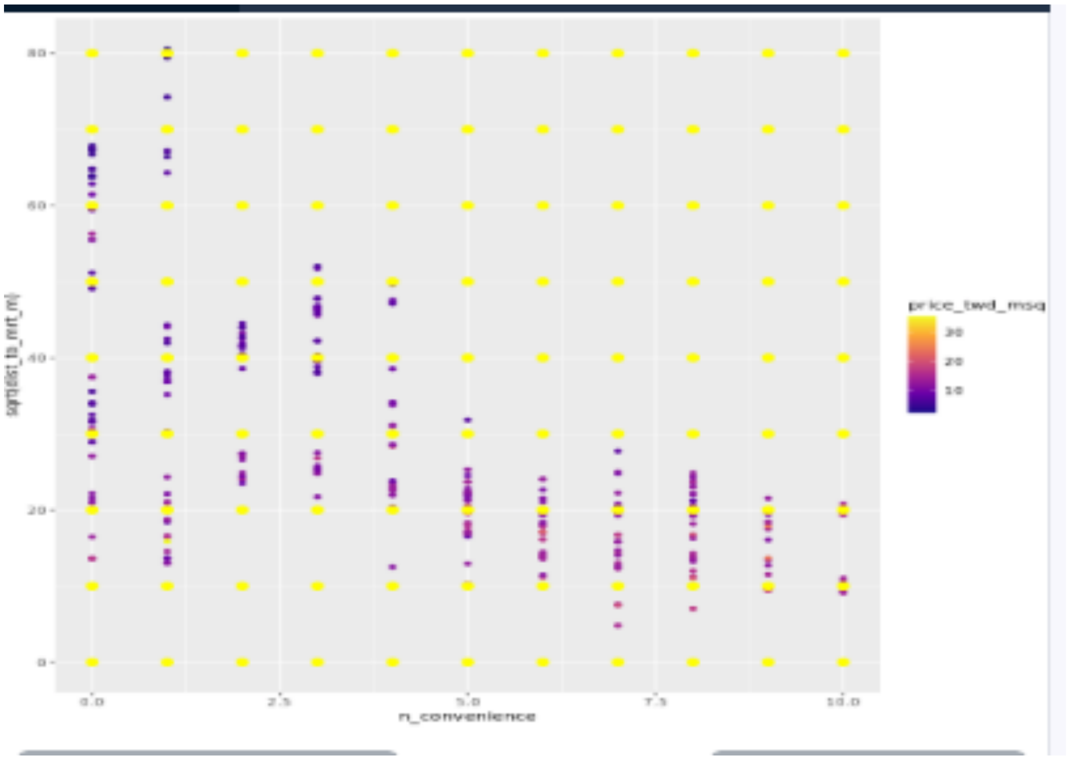
Including an interaction
*simply change to * from + between the explanatory variables

Example
# From previous steps
mdl_price_vs_conv_dist <- lm(price_twd_msq ~ n_convenience + sqrt(dist_to_mrt_m), data = taiwan_real_estate)
explanatory_data <- expand_grid(n_convenience = 0:10, dist_to_mrt_m = seq(0, 80, 10) ^ 2)
prediction_data <- explanatory_data %>%
  mutate(price_twd_msq = predict(mdl_price_vs_conv_dist, explanatory_data))

# Add predictions to plot

```
ggplot(
  taiwan_real_estate,
  aes(n_convenience, sqrt(dist_to_mrt_m), color = price_twd_msq)
) +
  geom_point() +
  scale_color_viridis_c(option = "plasma")+
  # Add prediction points colored yellow, size 3
  geom_point(data = prediction_data, size = 3, color = 'yellow')
```

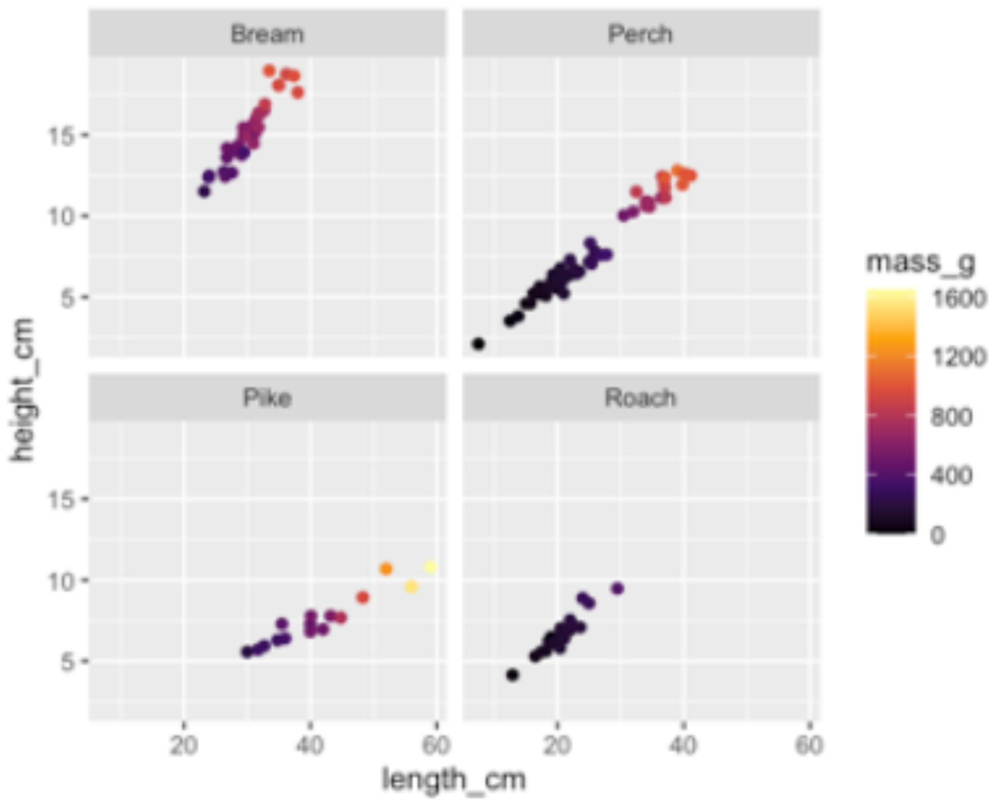
output>



More than 2 explanatory variables
we can use facet_wrap when visualizing

```
ggplot(
  fish,
  aes(length_cm, height_cm, color = mass_g)
) +
  geom_point() +
  scale_color_viridis_c(option = "inferno") +
  facet_wrap(vars(species))
```

output>



*it becomes trickier to include more than 3 numeric variables
*however with faceting you can include as many categorical variables as you like

Different levels of interaction

No interactions

```
lm(mass_g ~ length_cm + height_cm + species + 0, data = fish)
```

2-way interactions between pairs of variables

```
lm(
  mass_g ~ length_cm + height_cm + species + length_cm:height_cm + length_cm:species + height_cm:species + 0,
  data = fish
)
```

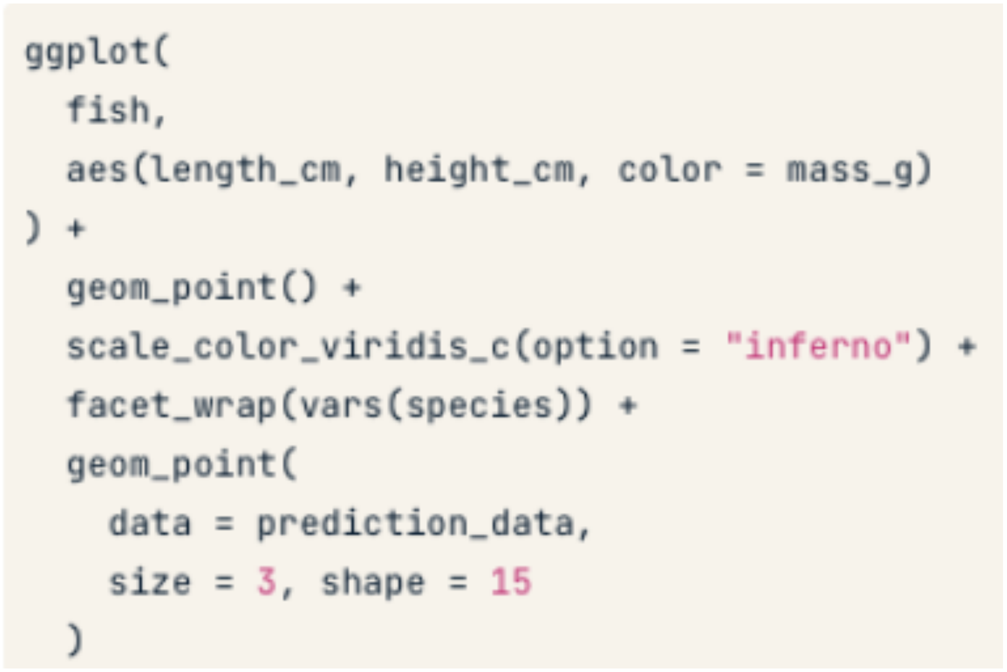
3-way interaction between all three variables

```
lm(
  mass_g ~ length_cm + height_cm + species + length_cm:height_cm + length_cm:species + height_cm:species + length_cm:height_cm:species + 0,
  data = fish
)
```

for easier syntax and a desire for all interactions just switch the + for * in the no interactions formula

for easier syntax and a desire for 2-way interaction:
lm(response ~ (exp + exp + exp) ^ 2 + 0, data = df)
*side - to square an explanatory variable > lm(response ~ I(exp) ^ 2 + exp + exp + 0, data = df)

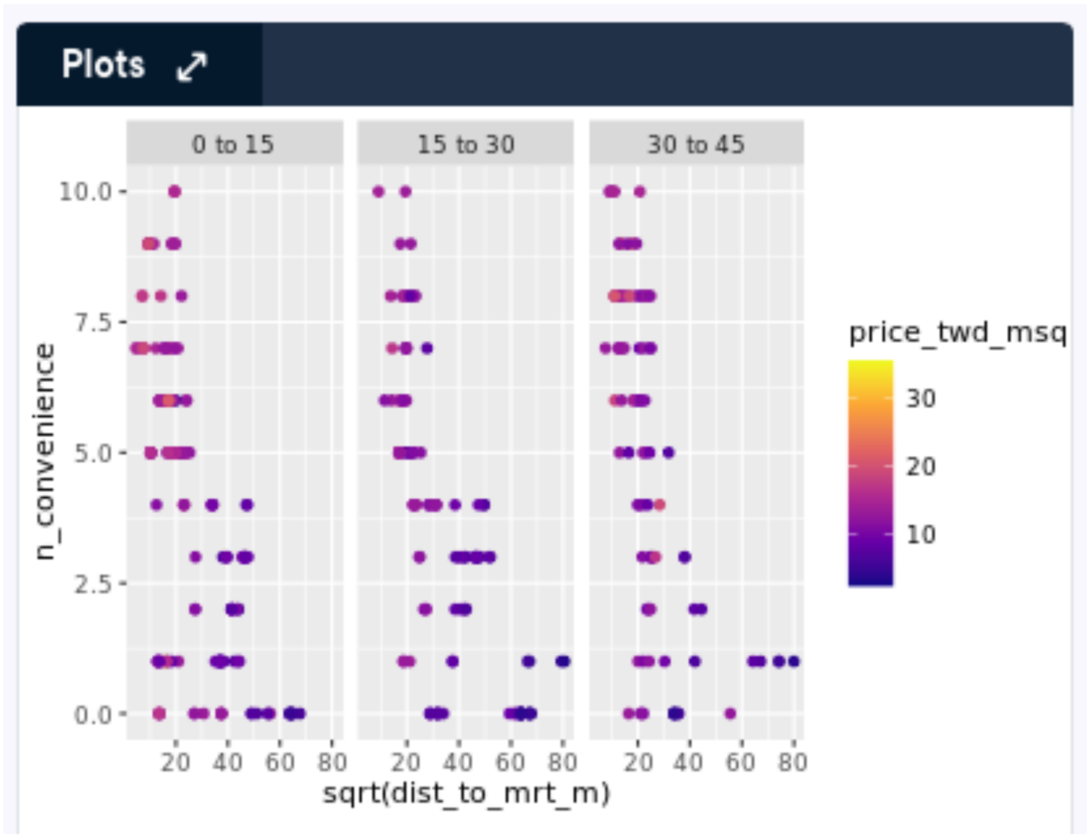
Visualizing/plotting predictions



```
ggplot(
  fish,
  aes(length_cm, height_cm, color = mass_g)
) +
  geom_point() +
  scale_color_viridis_c(option = "inferno") +
  facet_wrap(vars(species)) +
  geom_point(
    data = prediction_data,
    size = 3, shape = 15
  )
```

Example
# Using taiwan_real_estate, no. of conv. stores vs. sqrt of dist. to MRT, colored by plot house price
ggplot(taiwan_real_estate, aes(sqrt(dist_to_mrt_m), n_convenience, color = price_twd_msq)) +
  # Make it a scatter plot
  geom_point() +
  # Use the continuous viridis plasma color scale
  scale_color_viridis_c(option = 'plasma') +
  # Facet, wrapped by house age
  facet_wrap(vars(house_age_years))

output>

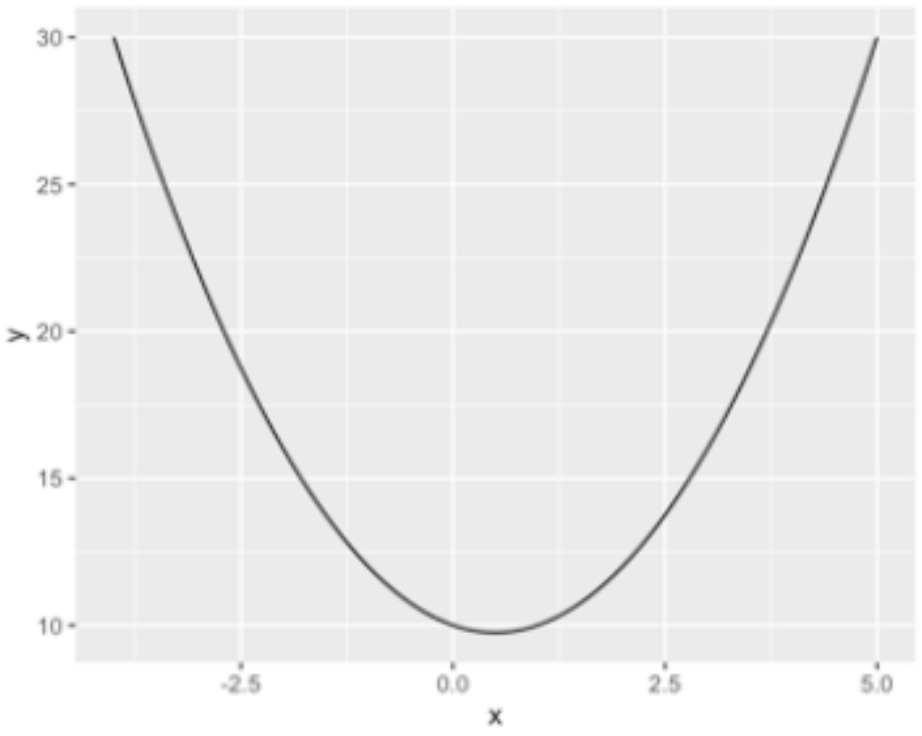


How linear regression works
for best fit, we want a metric that measures the size of all the residuals
we want the residuals to be as small as possible
the first go to is sum of squares > we use squares so that the negative residuals do not cancel out the positive residuals
*the goal is to find the intercept and slope coefficients that will result in the smallest sum of squares

To solve this problem > numerical optimization, meaning finding the minimum point of a function
example:
for the quadratic equation > y = x^2 - x + 10
*here the minimum point of the function occurs when x is a little above 0
with R:

```r
xy_data <- tibble(
  x = seq(-4, 5, 0.1),
  y = x ^ 2 - x + 10
)

ggplot(xy_data, aes(x, y)) +
  geom_line()
```

output>

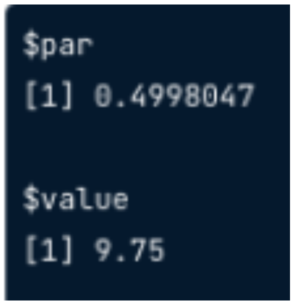


how to find this directly? > calculus can help
y = x^2= x + 10
take the derivative
derivative of y / derivative of x = 2x - 1
set derivative to 0
0 = 2x - 1
x = 0.5
y = 0.5^2 - 0.5 + 10 = 9.75
not all equations can be solved like this
*R can do this for us

optim() function performs numerical optimization
example:
start with function to minimize

```
calc_quadratic <- function(x) {
        x <- coeffs[1]
        x^2 - x + 10}
```

#the function passed to optim is only allowed to hvae one argument > to optimize for multiple variables we pass them as a numeric vector
optim(par = c(x = 3), fn = calc_quadratic)
#first argument is an inital guess > *this number can often be any number
output>



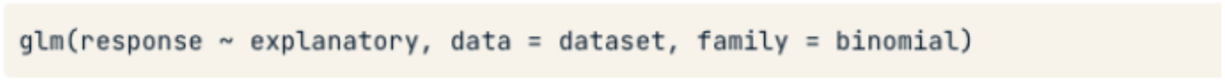
```
$par
[1] 0.4998047

$value
[1] 9.75
```

$par gives the x value
$value gives the y value

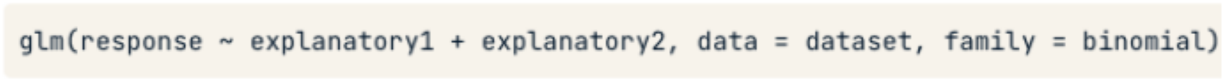
Example

```
# Set the intercept to 10
intercept <- 10

# Set the slope to 1
slope <- 1

# Calculate the predicted y values
y_pred <- slope * x_actual + intercept

# Calculate the differences between actual and predicted
y_diff <- y_actual - y_pred

# Calculate the sum of squares
sum(y_diff^2)

# From previous step
calc_sum_of_squares <- function(coeffs) {
  intercept <- coeffs[1]
  slope <- coeffs[2]
  y_pred <- intercept + slope * x_actual
  y_diff <- y_actual - y_pred
  sum(y_diff ^ 2)
}

# Optimize the metric
optim(
  # Initially guess 0 intercept and 0 slope
  # Need a named vector use c() to do this
  par = c(intercept = 0, slope = 0),
  # Use calc_sum_of_squares as the optimization fn
  fn = calc_sum_of_squares
)

# Compare the coefficients to those calculated by lm()
lm(price_twd_msq ~ n_convenience, data = taiwan_real_estate)
```

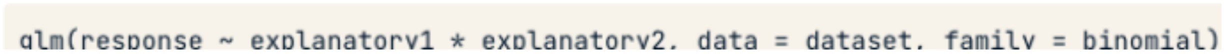
Multiple logistic regression
*prediction is true or false, 0 or 1 (binomial)
to perform a logistic regression
change lm() to generalized linear model glm()
need to include ‘family’ argument



```
glm(response ~ explanatory, data = dataset, family = binomial)
```

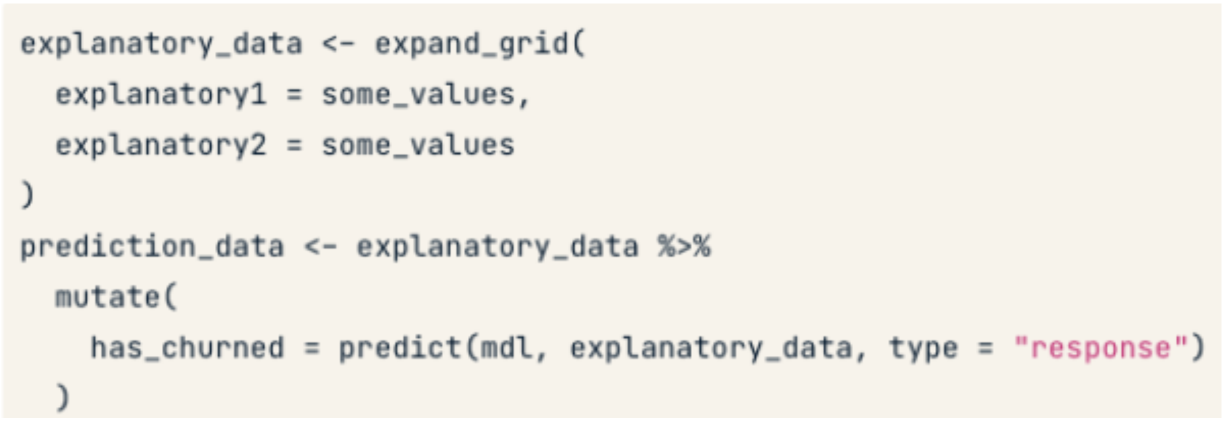


```
glm(response ~ explanatory1 + explanatory2, data = dataset, family = binomial)
```



```
glm(response ~ explanatory1 * explanatory2, data = dataset, family = binomial)
```

Prediction flow same as linear model, now just need to place ‘type’ argument and set to ‘response’



```
explanatory_data <- expand_grid(
  explanatory1 = some_values,
  explanatory2 = some_values
)
prediction_data <- explanatory_data %>%
  mutate(
    has_churned = predict(mdl, explanatory_data, type = "response")
  )
```

**when response variable only has two possible values > there are four outcomes for the model
*creating a confusion matrix

	actual false	actual true
predicted false	correct	false negative
predicted true	false positive	correct

```
actual_response <- dataset$response
predicted_response <- round(fitted(mdl))
```

```
outcomes <- table(predicted_response, actual_response)
```

```
confusion <- conf_mat(outcomes)
```

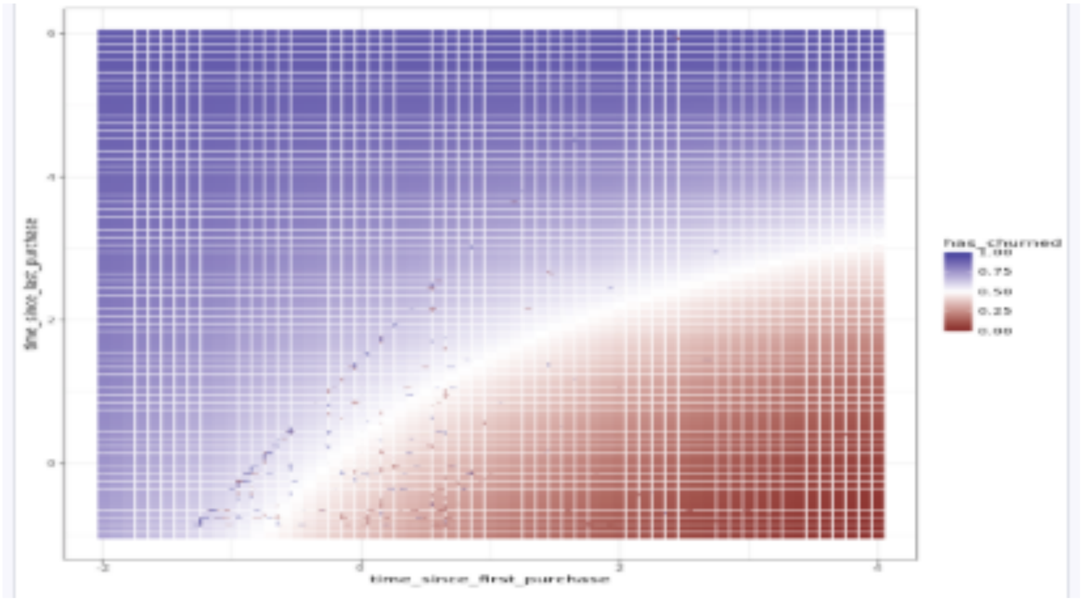
```
autoplot(confusion)
```

```
summary(confusion, event_level = "second")
```

Again visualizing the plot when you have multiple explanatory variables is tricky
use faceting for categorical variables
for 2 numeric explanatory variable, use 'color' for response
give predicted probabilities less thatn 0.5 one color and predicted probabilities above 0.5 another color
with R:
scale_color_gradient2(midpoint = 0.5)

Example
```
# Using churn, plot recency vs. length of relationship colored by churn status
ggplot(churn, aes(time_since_first_purchase, time_since_last_purchase, color = has_churned)) +
  # Make it a scatter plot, with transparency 0.5
  geom_point(alpha = 0.5) +
  # Use a 2-color gradient split at 0.5
  scale_color_gradient2(midpoint = 0.5) +
  # Use the black and white theme
  theme_bw()
```

```
# Fit a logistic regression of churn status vs. length of relationship, recency, and
an interaction
mdl_churn_vs_both_inter <- glm(
        formula = has_churned ~ time_since_last_purchase *
time_since_first_purchase,
        data = churn,
        family = 'binomial')

# See the result
mdl_churn_vs_both_inter

# From previous steps
explanatory_data <- expand_grid(
  time_since_first_purchase = seq(-2, 4, 0.1),
  time_since_last_purchase = seq(-1, 6, 0.1)
)
prediction_data <- explanatory_data %>%
  mutate(
    has_churned = predict(mdl_churn_vs_both_inter, explanatory_data, type =
"response")
  )

# Extend the plot
ggplot(
  churn,
  aes(time_since_first_purchase, time_since_last_purchase, color = has_churned)
) +
  geom_point(alpha = 0.5) +
  scale_color_gradient2(midpoint = 0.5) +
  theme_bw() +
  # Add points from prediction_data with size 3 and shape 15
  geom_point(data = prediction_data, size = 3, shape = 15)

output>
```



Example

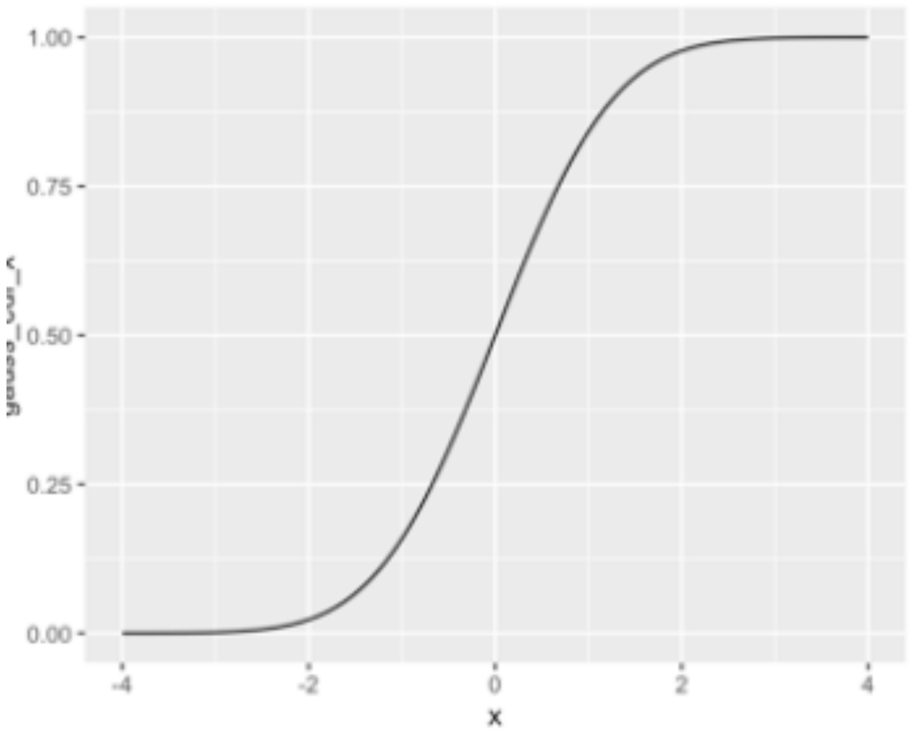
```
# From previous step
actual_response <- churn$has_churned
predicted_response <- round(fitted(mdl_churn_vs_both_inter))
outcomes <- table(predicted_response, actual_response)
confusion <- conf_mat(outcomes)

# "Automatically" plot the confusion matrix
autoplot(confusion)

# Get summary metrics
summary(confusion, event_level = "second")
```

The logistic distribution
for regression we care more about the area under the curve
we care about the cumulative distribution function (CDF)
with R we call pnorm

```
gaussian_distn <- tibble(
  x = seq(-4, 4, 0.05),
  gauss_pdf_x = dnorm(x),
  gauss_cdf_x = pnorm(x)
)
```

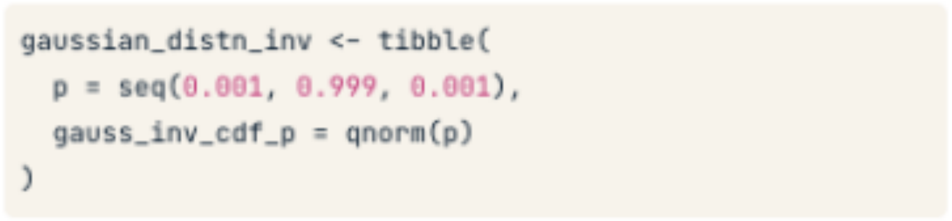
```
ggplot(gaussian_distn, aes(x, gauss_cdf_x)) +
  geom_line()
```

output>

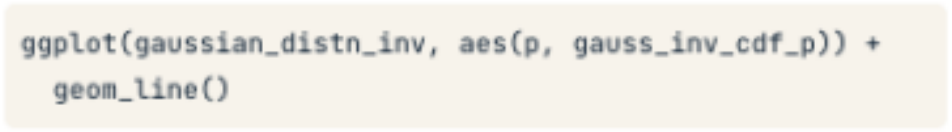


range is infinity
when x has its minimum possible value, y will be 0
when x has its maximum possible value, y will be 1
*for CDF, we essentially are taking the values of x and transforming them to probabilities

Gaussian inverse CDF

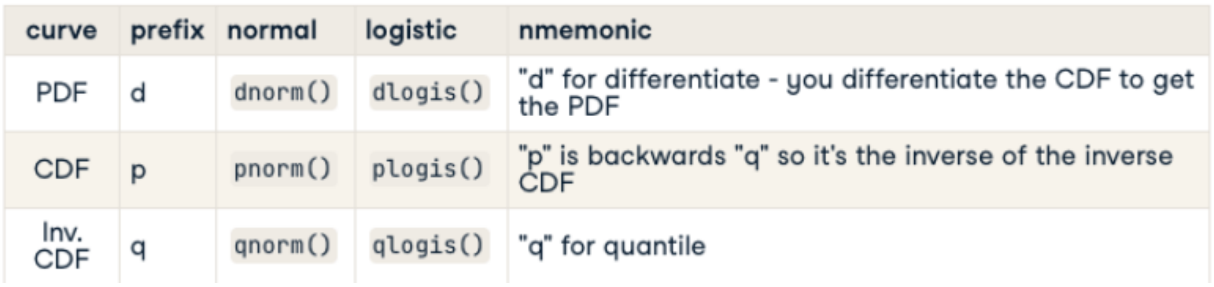


```
gaussian_distn_inv <- tibble(
  p = seq(0.001, 0.999, 0.001),
  gauss_inv_cdf_p = qnorm(p)
)
```



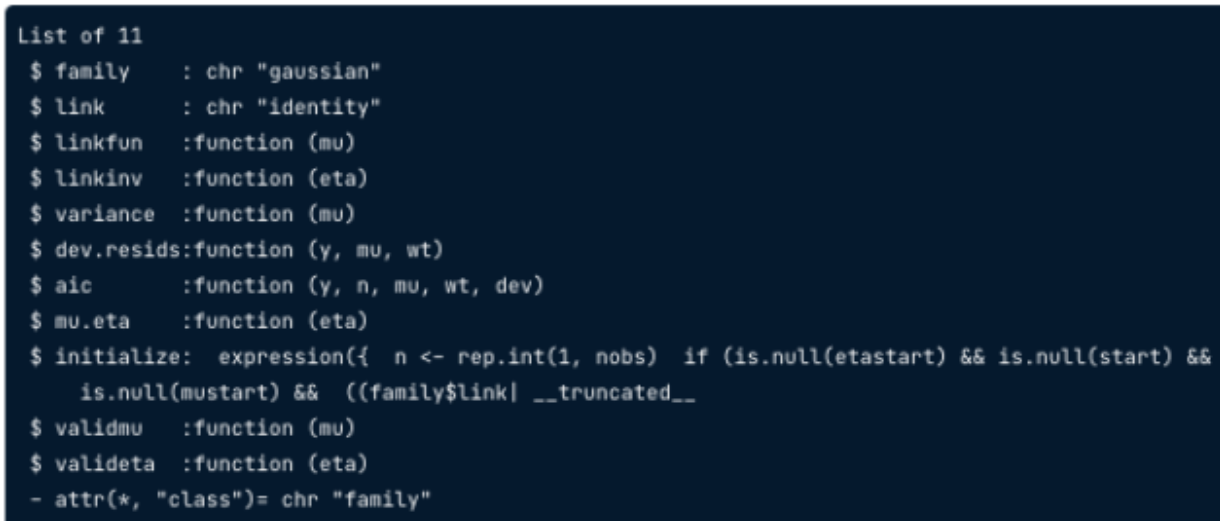
```
ggplot(gaussian_distn_inv, aes(p, gauss_inv_cdf_p)) +
  geom_line()
```

this is how we transform from probabilities to x-values
inverse CDF is calculated with qnorm



curve	prefix	normal	logistic	nmemonic
PDF	d	`dnorm()`	`dlogis()`	"d" for differentiate - you differentiate the CDF to get the PDF
CDF	p	`pnorm()`	`plogis()`	"p" is backwards "q" so it's the inverse of the inverse CDF
Inv. CDF	q	`qnorm()`	`qlogis()`	"q" for quantile

What are glm()'s family arguments?
calling the gaussian function and wrapping the result in the str function shows the structure
the returned object contains several other functions
these functions contain all the details for tuning a generalized regression into a specific type of regression
in R > str(gaussian())
output>

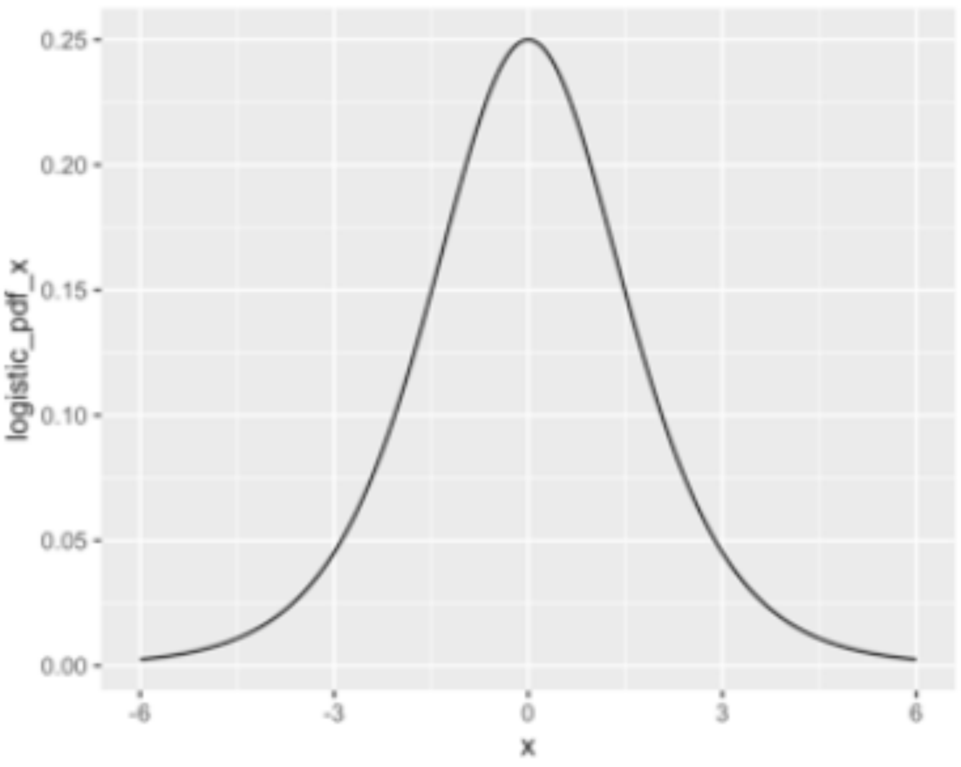


```
List of 11
 $ family    : chr "gaussian"
 $ link      : chr "identity"
 $ linkfun   :function (mu)
 $ linkinv   :function (eta)
 $ variance  :function (mu)
 $ dev.resids:function (y, mu, wt)
 $ aic       :function (y, n, mu, wt, dev)
 $ mu.eta    :function (eta)
 $ initialize:  expression({  n <- rep.int(1, nobs)  if (is.null(etastart) && is.null(start) &&
     is.null(mustart) &&  ((family$link| __truncated__
 $ validmu   :function (mu)
 $ valideta  :function (eta)
 - attr(*, "class")= chr "family"
```

linkfun - Link function is a transformation of the response variable
linkinv - this function undoes that transformation

Logistic PDF
similar to Gaussian PDF but the tails at the extreme left and right of the plot are fatter
in R:

```
logistic_distn <- tibble(
        x = seq(-6, 6, 0.05),
        logistic_pdf_x = dlogis(x))
ggplot(logistic_distn, aes(x, logistic_pdf_x)) +
        geom_line()
```



Logistic distribution CDF is also called the logistic function

$$\text{cdf}(x) = \frac{1}{(1+exp(-x))}$$

Logistic distribution inverse CDF is also called the logit function

$$\text{inverse\_cdf}(p) = log(\frac{p}{(1-p)})$$

the logisitc distribution’s CDF is claculated with the logisitic function
the plot of this has an S-shape, known as a sigmoid curve
an important property of this function is that it takes an input that can be any number from minus infinity to infinity, and returns a value between 0 and 1

Example

```
logistic_distn_cdf <- tibble(
  # Make a seq from -10 to 10 in steps of 0.1
  x = seq(-10, 10, 0.1),
  # Transform x with built-in logistic CDF
  logistic_x = plogis(x),
  # Transform x with manual logistic
  logistic_x_man = 1 / (1 + exp(-x))
)

# Check that each logistic function gives the same results
all.equal(
  logistic_distn_cdf$logistic_x,
  logistic_distn_cdf$logistic_x_man
)
[1] TRUE

# Using logistic_distn_cdf, plot logistic_x vs. x
ggplot(logistic_distn_cdf, aes(x, logistic_x)) +
  # Make it a line plot
  geom_line()
```

Inverse cummulative distribution function
The logistic function (logistic distribution CDF) has anotehr important peoperty:
each x input value is transformed to a unique value
that means that the transformation can be reversed
the logit function is the name for the inverse logistic function, which is also called
the logistic distribution inverse cumulative distribution function
*all three terms mean exactly the same thing
the logit function takes values between 0 and 1, and returns values between minus infinity and infinity

Example
```
# From previous step
logistic_distn_inv_cdf <- tibble(
  p = seq(0.001, 0.999, 0.001),
  logit_p = qlogis(p),
  logit_p_man = log(p / (1 - p))
)

# Using logistic_distn_inv_cdf, plot logit_p vs. p
ggplot(logistic_distn_inv_cdf, aes(p, logit_p)) +
  # Make it a line plot
  geom_line()

# Look at the structure of binomial() function
str(binomial())

# Call the link inverse on x
linkinv_x <- binomial()$linkinv(x)
```

```
# Check linkinv_x and plogis() of x give same results
all.equal(
   linkinv_x,
   plogis(x)
)

# Call the link fun on p
linkfun_p <- binomial()$linkfun(p)

# Check linkfun_p and qlogis() of p give same results
all.equal(
   linkfun_p,
   qlogis(p)
)
```

As ‘location’ increases, the logistic CDF curve moves rightwards. As ‘scale’ increases, the steepness of the slope decreases.

How logistic regression works
same goal as with linear regression > choose a metric that measures how far the predicted responses are from the actual responses
then optimize that metric
*sum of squares does not work here, it optimizes poorly
remember in logistic regression the actual response is always 0 or 1 and the predicted responses are between these two values
the metric we use for logistic regression is the ‘likelihood’ metric
sum of squares goal is to find the minimum value
likelihood goal is to find the maximum value

Solving for likelihood
sum(y_pred * y_actual + (1 - y_pred) * (1 - y_actual))
*we can simplify this depending on y_actual
When y_actual = 1:
y_pred * 1 + (1 - y_pred) * (1-1) = y_pred
example y_pred = 0.8
0.8 * 1 + (1 - 0.8) * 0 = 0.8
0.8 + 0.2 * 0 = 0.8
0.8 + 0 = 0.8
When y_actual = 0:
y_pred * 0 + (1 - y_pred) * (1 - 0) = 1 - y_pred
example y_pred = 0.8
0.8 * 0 + (1 - 0.8) * 1 = 0.2

0 + 0.2 * 1 = 0.2

*as y-pred decreases, the metric increases, and the maximum likelihood occurs when y_pred is 0
*you get a higher likelihood score when the predicted response is close to the actual response

Log-likelihood
when calculating the likelihood, y_pred is often close to 0 or 1, which means you end up adding up lots of very small numbers, which introduces numerical error
it is more efficient to compute the log-likelihood
log(y_pred) * y_actual + log(1 - y_pred) * (1 - y_actual)
optimizing to find the log-likelihood gives the same coefficients as optimizing to find the likelihood
since we want to maximize likelihood, but the optim function defaults to finding minimums, we need to calculate the negative log-likelihood
we add a minus sign when calculating the sum of each observation's likelihood contribution
-sum(log_likelihoods)

Example
```
# Calculate the predicted y values
y_pred <- 1 / (1 + exp(-(intercept + slope * x_actual)))

# Calculate the log-likelihood for each term
log_likelihoods <- log(y_pred) * y_actual + log(1 - y_pred) * (1 - y_actual)

# Calculate minus the sum of the log-likelihoods for each term
-sum(log_likelihoods)
[1] 326.2599
```

Example
```
calc_neg_log_likelihood <- function(coeffs) {
  # Get the intercept coeff
  intercept <- coeffs[1]

  # Get the slope coeff
  slope <- coeffs[2]

  # Calculate the predicted y values
  y_pred <- plogis(intercept + slope * x_actual)

  # Calculate the log-likelihood for each term
  log_likelihoods <- log(y_pred) * y_actual + log(1 - y_pred) * (1 - y_actual)

  # Calculate minus the sum of the log-likelihoods for each term
  -sum(log_likelihoods)
}

# Optimize the metric
optim(
  # Initially guess 0 intercept and 1 slope
  par = c(intercept = 0, slope = 1),
  # Use calc_neg_log_likelihood as the optimization fn
  fn = calc_neg_log_likelihood
)

# Compare the coefficients to those calculated by glm()
glm(has_churned ~ time_since_last_purchase, data = churn, family = binomial)
```