Module 11 Homework: Linear regression

Natalie Nelson, PhD

Source: Bob Nichols/Photographer/United States Department of Agriculture

Background

As a country develops economically, its population becomes more capable of purchasing high value food products, such as meat. Accordingly, a nation's meat consumption is often directly linked to its economic status, though there are additional cultural factors that nuance this relationship (e.g., religious dietary restrictions).

In this homework assignment, you will create a regression model to predict the amount of meat a country produces. You have two options for explanatory variables (also referred to as "independent" variables): gross domestic product (GDP), and total population. Ideally, we would estimate the total meat *consumed* in a country, but the data we have available are in terms of meat *produced*. For this assignment, we will assume that the meat produced by a country serves as a good proxy for the amount of meat consumed in the country.

The data you will work with are curated by the Food and Agriculture Organization (FAO) of the United Nations. The data are split across three files:

- `meat-production-2017.csv`: Different types of meats produced in 2017 per country. Units: tons.
- `gdp-2017.csv`: Gross domestic product in 2017 per country. Units: millions of US Dollars.
- `population-2017.csv`: Total population in 2017 per country. Units: 1000 persons.

Grading

You will submit one R script for this assignment. To grade this assignment, I will be checking your script to make sure that you:

- (2 points) Followed along with the examples presented here
- (3 points) Fit all of the regression models specified
- (3 points) Inspected each of the models through `tidy()`+`glance()` or `summary()`, as well as with the two diagnostic plots outlined in the assignment
- (2 points) Provided an answer to the question: which model is best and why?

Packages

This assignment will use functions from the following packages: `tidyverse`, `modelr`, `broom`, and `corrr`. If you haven't already, install and load these packages.

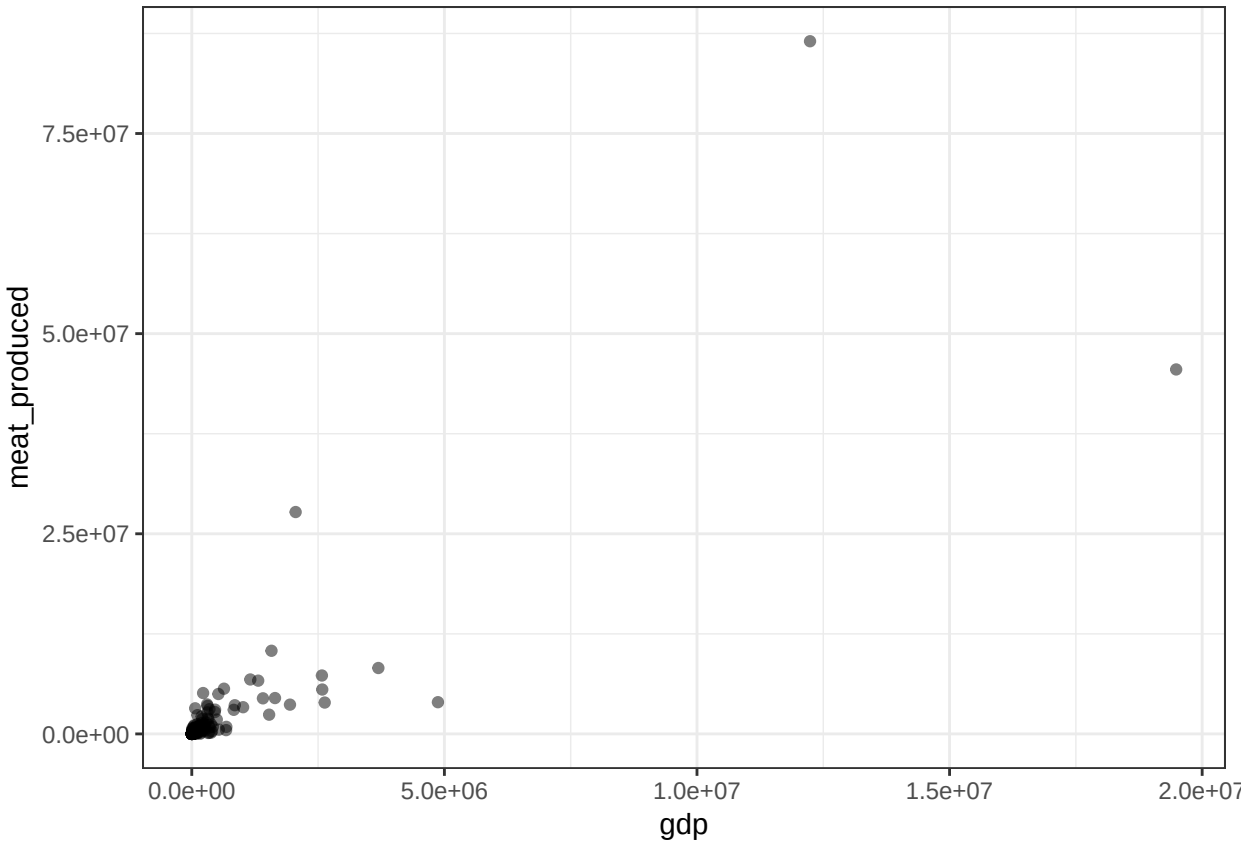
Load and organize data

Create one dataframe that includes the following columns: `country`, `meat_produced`, `total_population`, `gdp`. This will require you to join the data from the three .csv files you've been provided. Prior to joining the data, you will need to `summarize` the meat production data so that you have the **total** meat produced per country. To calculate the total meat production per country, use the `sum()` function and be sure to include `na.rm = T` as an argument. Once you join your data, your merged dataframe should have 198 rows and 4 columns. I've included the top 10 rows below for reference.

```
## # A tibble: 10 x 4
##    country             meat_produced      gdp total_population
##    <chr>                       <dbl>    <dbl>            <dbl>
##  1 Afghanistan                284565   21993.          36296.
##  2 Albania                     89874   13039.           2884.
##  3 Algeria                    728959  167555.          41389.
##  4 Angola                     287023  126506.          29817.
##  5 Antigua and Barbuda           291    1510.             95.4
##  6 Argentina                 5647680  637486.          43937.
##  7 Armenia                    109338   11537.           2945.
##  8 Australia                 4444891 1408676.          24585.
##  9 Austria                    889733  416836.           8820.
## 10 Azerbaijan                 316827   40749.           9845.
```

Visualize data

Now we'll plot our data to visualize the relationships between `meat_produced` and `gdp`, then `meat_produced` and `total_population`. We will use `alpha = 0.5` to see which points are overlapping.



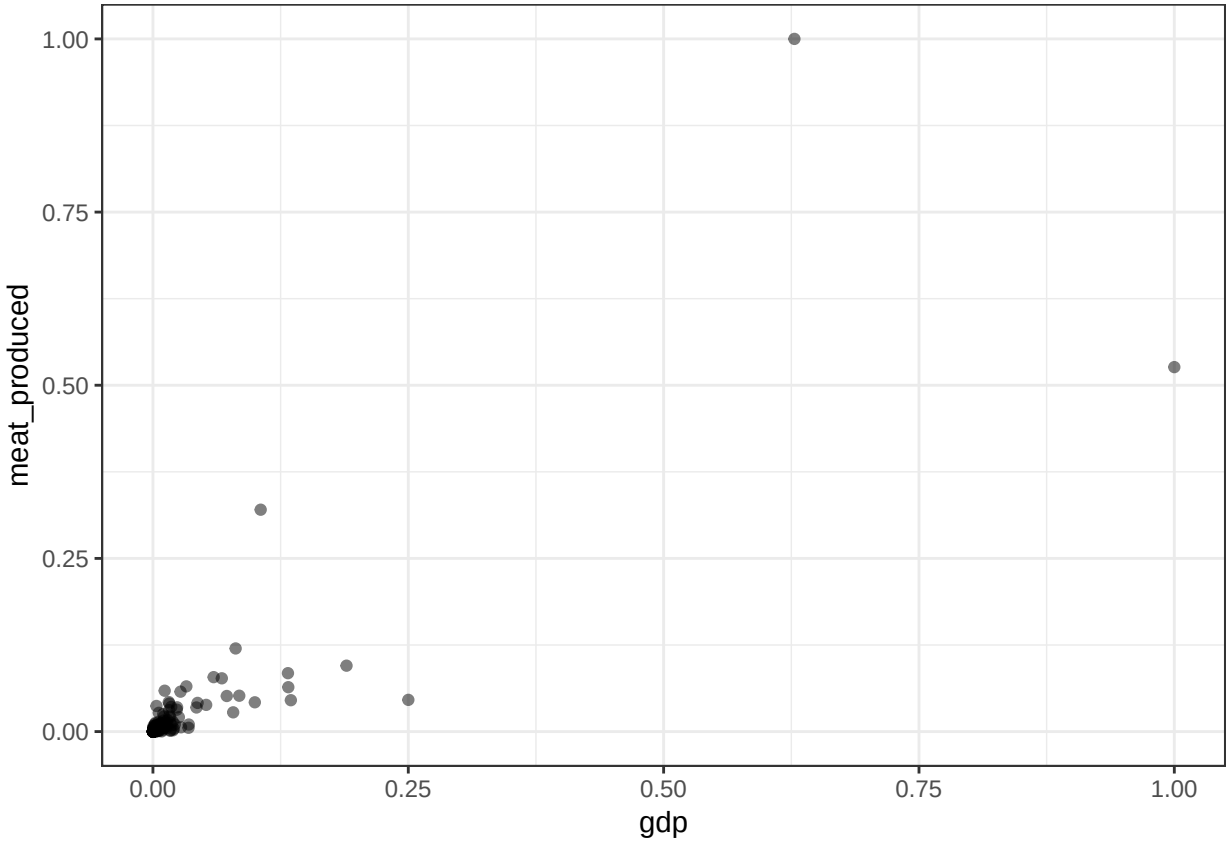
As you can see, the numbers we're working with are really large. These values capture national trends, so it's to be expected that the values are big. To make plotting easier, let's go ahead and normalize our data. We can create a function, `normalize`, that will do this for us. After applying the `normalize()` function, each of our columns will have values that range from 1 to 0, with the value that corresponds to 1 being the maximum, and the value that corresponds to 0 being the minimum. Note that although the values have changed, we have preserved the behavior of the individual variables and the relationships between variables is also preserved.

```
# User-defined function for normalizing
normalize <- function(x) {
  n <- (x - min(x)) / (max(x) - min(x))
  return(n)
}

# Normalize the meat_produced, gdp, and total_population columns
d %>%
  mutate(meat_produced = normalize(meat_produced),
         gdp = normalize(gdp),
         total_population = normalize(total_population)) -> d_norm

# Check that each column in the new dataframe ranges from 0-1
summary(d_norm)
```

```
##    country          meat_produced            gdp
##  Length:198         Min.   :0.0000000   Min.   :0.0000000
```

```
##  Class :character   1st Qu.:0.0003751   1st Qu.:0.0003931
##  Mode  :character   Median :0.0025079   Median :0.0014704
##                     Mean   :0.0192958   Mean   :0.0207062
##                     3rd Qu.:0.0093182   3rd Qu.:0.0102781
##                     Max.   :1.0000000   Max.   :1.0000000
##  total_population
##  Min.   :0.000000
##  1st Qu.:0.001389
##  Median :0.005866
##  Mean   :0.026724
##  3rd Qu.:0.019079
##  Max.   :1.000000
```

```
# Re-plot the meat_produced vs. gdp figure
ggplot(data = d_norm) +
  geom_point(mapping = aes(x = gdp, y = meat_produced), alpha = 0.5) +
  theme_bw()
```

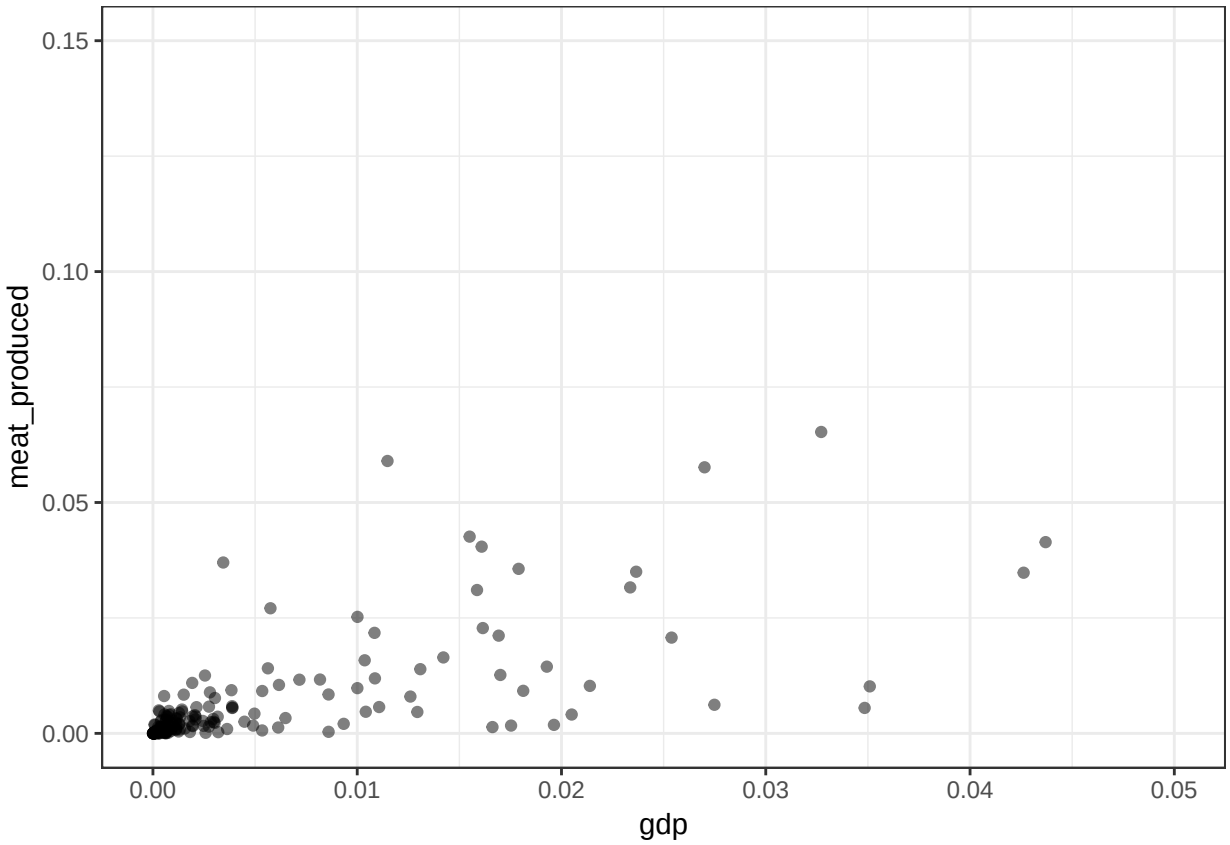


Note that the normalized plot looks exactly the same as the non-normalized plot, except the x- and y-axis limits range from 0 to 1. With the plot created from normalized data, we can easily adjust the axis limits to zoom in on specific parts of our plot:

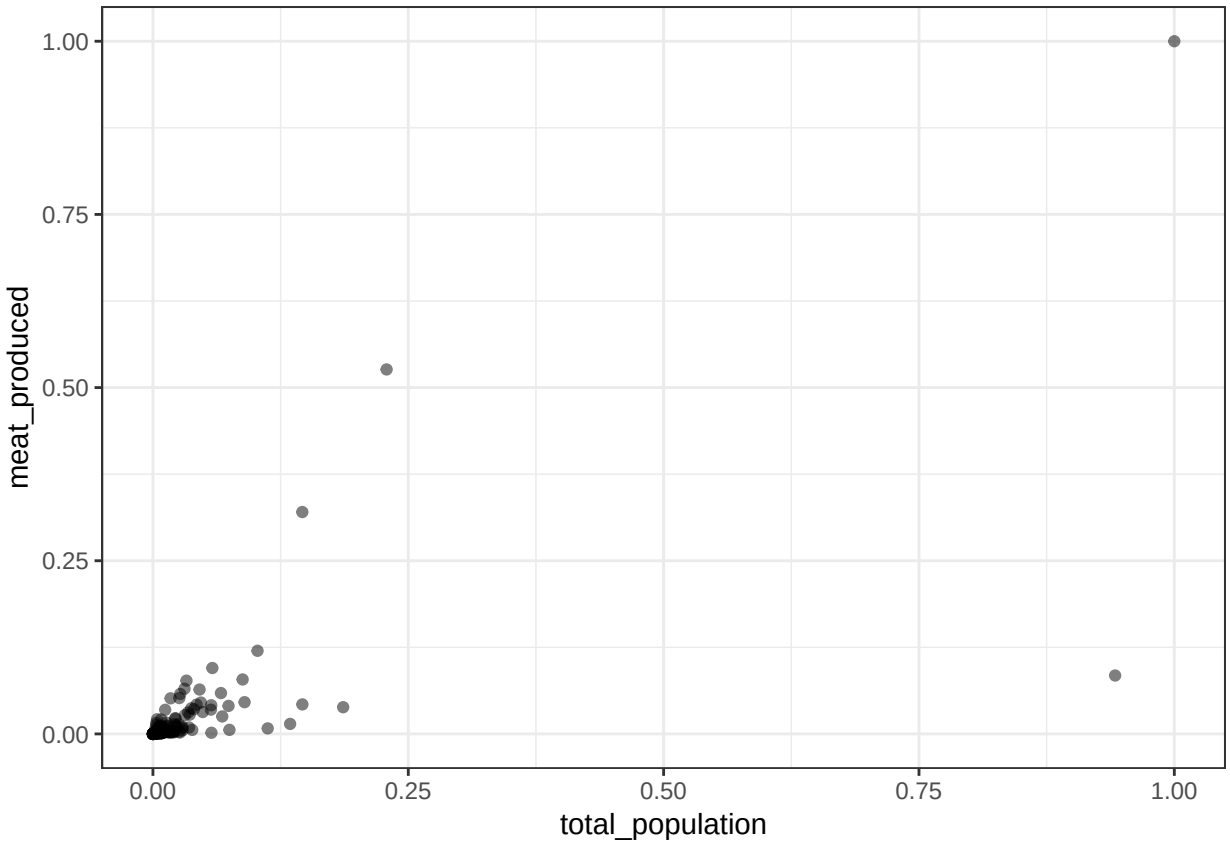
```
# Re-plot the meat_produced vs. gdp figure
ggplot(data = d_norm) +
  geom_point(mapping = aes(x = gdp, y = meat_produced), alpha = 0.5) +
```

```
  lims(x = c(0,0.05),
       y = c(0, 0.15)) +
  theme_bw()
```

```
## Warning: Removed 16 rows containing missing values (geom_point).
```



Go ahead and create a plot for `meat_produced` vs `total_population`.



Calculate and visualize correlations

Next, let's calculate the correlations between `meat_produced`, `gdp`, and `total_population`. We can already see from the plots that `meat_produced` is correlated with `gdp` and `total_population`, but which correlation is stronger? It's hard to say just from the plots. We will use functions from the `corrr` package to calculate and visualize correlations. Prior to calculating correlations, be sure to remove the `country` column. The correlation function can only calculate correlation coefficients from numeric data. Since `country` is a character vector, you can't include this column when calculating correlations.

```
d_norm %>%
  dplyr::select(-country) %>%
  correlate() %>%
  fashion()
```

```
##
## Correlation method: 'pearson'
## Missing treated using: 'pairwise.complete.obs'

##            rowname meat_produced  gdp total_population
## 1    meat_produced                .83              .74
## 2              gdp           .83                   .57
## 3 total_population           .74  .57
```

The output is provided as a matrix. From the correlations, we can see that `meat_produced` is fairly strongly correlated with both `total_population` and `gdp`, but more so with `gdp` (note that what constitutes a

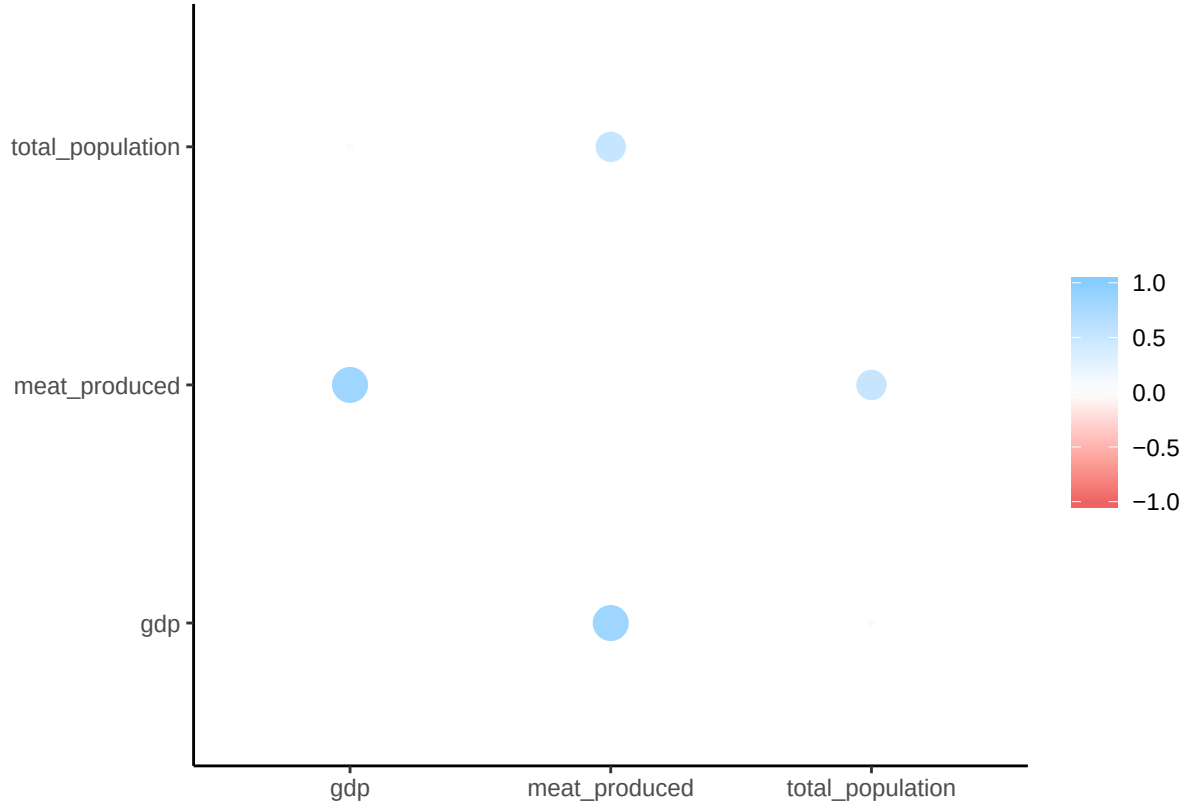
"strong" correlation can vary by discipline). To interpret the output, look at the intersections of the columns and rows to find the corresponding correlation coefficients. For example, we know that the correlation between `meat_produced` and `gdp` is 0.83 because this value is at the intersection of the `meat_produced` column and the `gdp` row (or, the `meat_produced` row and the `gdp` column). Similarly, we can see that `gdp` is correlated with `total_population` with a coefficient of 0.57.

We can also visualize the correlation coefficients through `rplot()`. The plot isn't particularly interesting in this case since we're only working with three variables, but these kinds of plots are nice when you're calculating correlations across many variables.

```
d_norm %>%
  dplyr::select(-country) %>%
  correlate() %>%
  rplot()
```

```
##
## Correlation method: 'pearson'
## Missing treated using: 'pairwise.complete.obs'
```

```
## Don't know how to automatically pick scale for object of type noquote. Defaulting to continuous.
```



Fit regression models

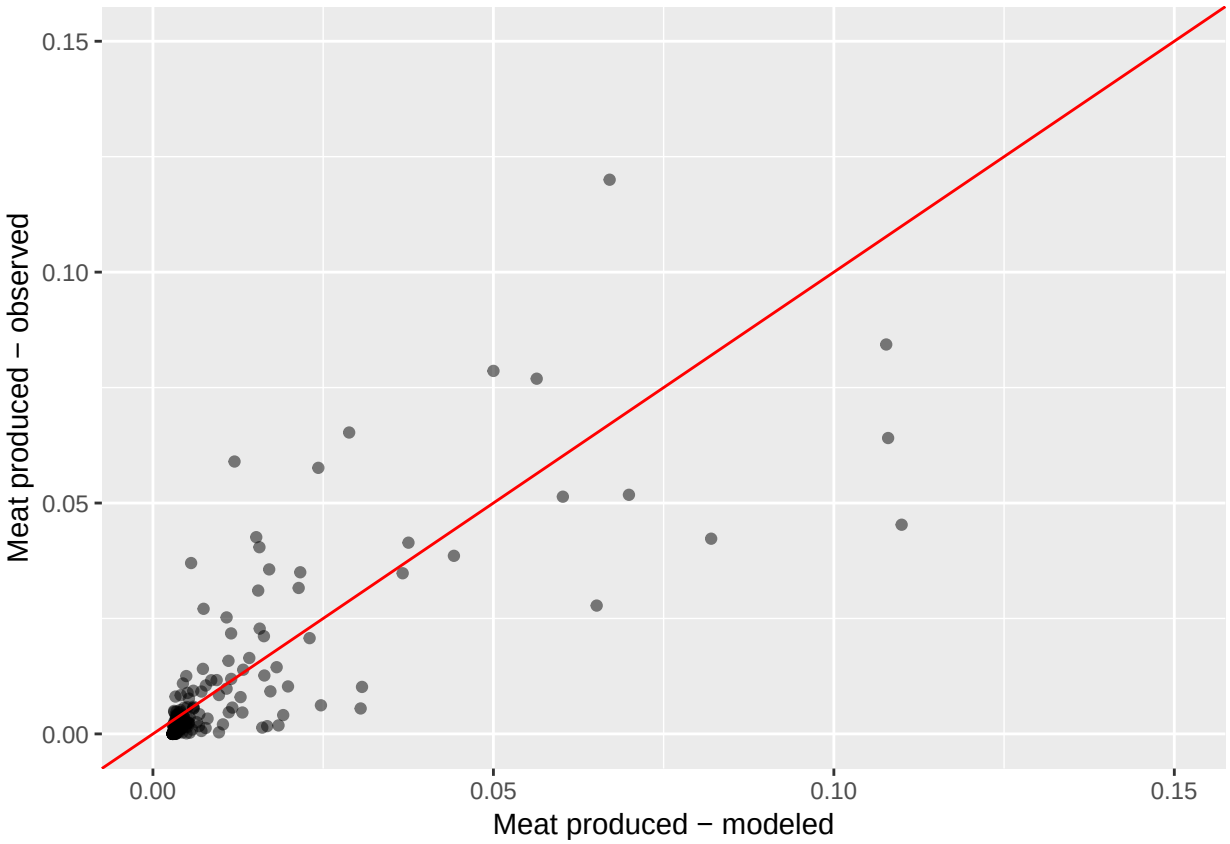
Now, you will fit several regression models to identify the model that best predicts a country's meat production. Specifically, fit the following models from your **normalized data**:

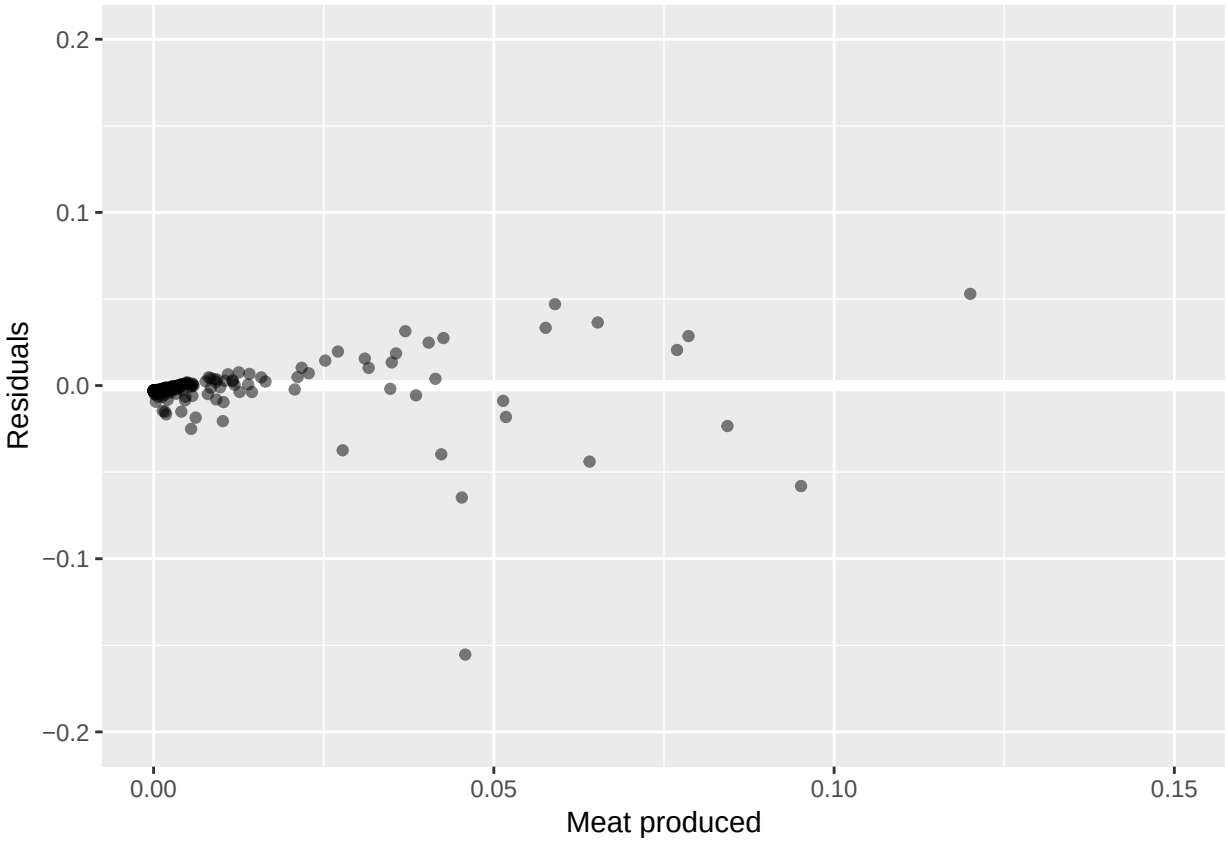
- `meat_produced` as a function of `gdp`
- `meat_produced` as a function of `total_population`
- `meat_produced` as a function of `gdp`, `total_population` and the interaction between `gdp` and `total_population` (refer to R4DS 23.4.2 to see how you should setup the formula for this model)
- `log(meat_produced)` as a function of `log(gdp)`
- `log(meat_produced)` as a function of `log(total_population)`

Inspect the models: To evaluate each model, use the `tidy()` and `glance()` functions from the `broom` package. Note that you can also use `summary()`. Additionally, evaluate your model by inspecting the following two plots using `ggplot`: (1) observed `meat_produced` vs. fitted `meat_produced` with a 1-to-1 reference line (made with `geom_abline(intercept = 0, slope = 1)`), and (2) residuals vs. `meat_produced` with a `ref_line` at `h = 0`. You will be able to make these plots quickly and easily if you use `augment()` from the `broom` package to create a data frame that includes your residuals and modeled values in a single tibble with your data. Alternatively, you can use `add_predictions()` or `add_residuals()` with `mutate()` to add the modeled values and residuals to your original tibble.

Note that the terms "fitted", "modeled", and "predicted" are used interchangeably to refer to values produced by your regression model. For example, let's say you have a model, mass = 0.5*diameter + 0.2, that you can use to estimate the mass of a leaf based on its diameter. You want to test the model, so you take diameter and mass measurements of a leaf. The leaf has a diameter of 3 and a mass of 2. If you were to estimate the mass of the leaf based on the model, your "modeled" leaf mass (= "fitted" leaf mass = "predicted" leaf mass) would be 0.5(3) + 0.2 = 1.7. In this case, the real leaf mass (= "observed" leaf mass) was 2, so our residual is 2 - 1.7 = 0.3. If the model had perfectly estimated the observed leaf mass (that is, if the model predicted hte leaf mass as 2), the residual would have been 0.

So, when a model performs well, observed `meat_produced` will be approximately equal to fitted `meat_produced`, meaning the points will fall close to the 1-to-1 line. Similarly, a model is performing well when the residuals are close to 0. Examples of these diagnostic plots for the first regression model are shown below.


0.15
0.10
0.05
0.00
Meat produced − observed
0.00
0.05
0.10
0.15
Meat produced − modeled



For both of the diagnostic plots, the x and y axis limits were adjusted to zoom in on the region where most of the points are located.

Recreate these plots for each of the models. **After looking at these plots and your model statistics, which model to you think is best for predicting the meat produced by a country? Include your answer as a comment at the bottom of your code. Provide a few sentences explaining your rationale.**

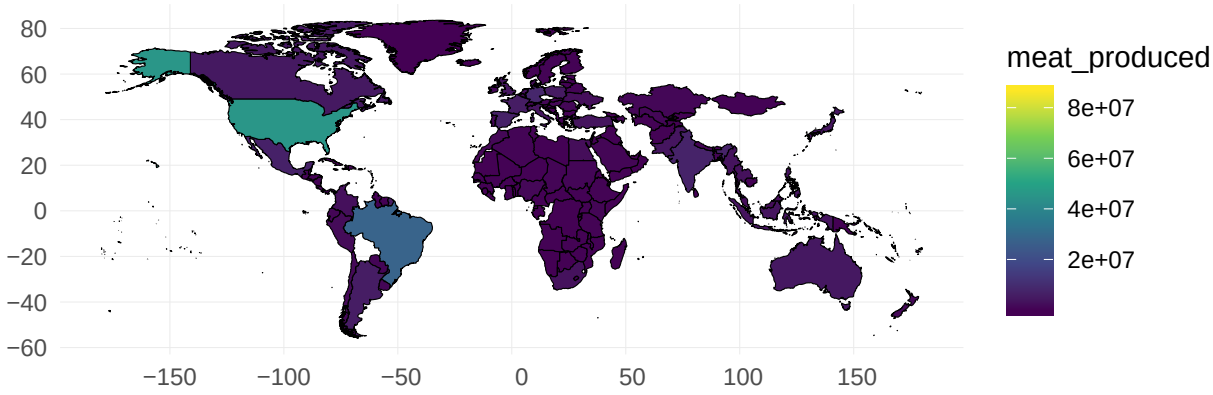
Just for fun

Now that you've worked with the `sf` package, it's easy to quickly create a map that visualizes the data we worked with in this assignment. Although we could search online for a shapefile that includes the boundaries of all the countries in the world, there's a R package that already includes this file for us in an easy-to-access format: `rnaturalearth`. In `rnaturalearth`, there's a function called `ne_countries()` that we can use to call a multipolygon spatial tibble that includes the geometry of the world's national boundaries. We can join this spatial tibble with our data to create a map that visualizes meat production across the world (or total population or gdp).

```
library(rnaturalearth)
library(sf)
```

```
## Linking to GEOS 3.6.1, GDAL 2.1.3, PROJ 4.9.3
```

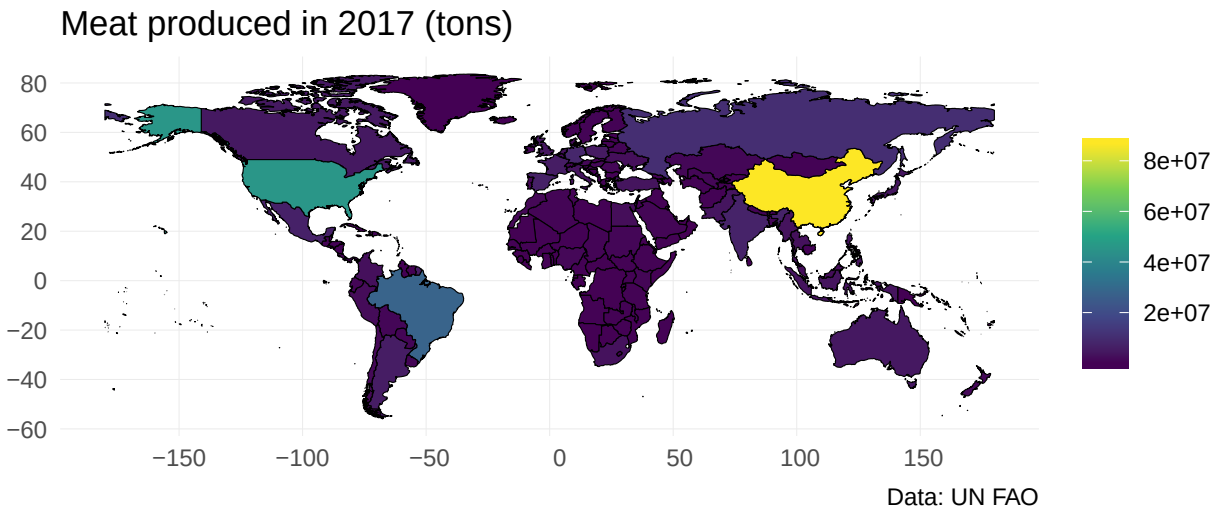
```r
world <- ne_countries(scale = "medium", returnclass = "sf")
# Merge the world spatial tibble with our assignment data, d. We will merge by country, with the key co
d %>% left_join(world, by = c("country" = "admin")) -> d_world
# Check on your own to make sure the data were joined properly

# Create a map where fill varies by meat produced
d_world %>%
  ggplot() +
  geom_sf(aes(fill = meat_produced), color = "black", size = 0.2) +
  scale_fill_viridis_c() +
  theme_minimal()
```



There are a few countries missing, most notably China, Russia, and Bolivia. This is due to the FAO data dividing China into three country units: China, mainland; China, Hong Kong SAR; China, Macao SAR. FAO also refers to Russia as "Russian Federation" and Bolivia as "Bolivia (Plurinational State of)". If we went back into our data and adjusted these names, the join will work as intended.

```r
d$country[d$country == "China, mainland"] <- "China"
d$country[d$country == "China, Hong Kong SAR"] <- "China"
d$country[d$country == "China, Macao SAR"] <- "China"
d$country[d$country == "Russian Federation"] <- "Russia"
d$country[d$country == "Bolivia (Plurinational State of)"] <- "Bolivia"

d %>% left_join(world, by = c("country" = "admin")) -> d_world
```

```
d_world %>%
  ggplot() +
  geom_sf(aes(fill = meat_produced), color = "black", size = 0.2) +
  scale_fill_viridis_c() +
  labs(fill = NULL,
       title = "Meat produced in 2017 (tons)",
       caption = "Data: UN FAO") +
  theme_minimal()
```



‘‘‘