Transformations

Reminder of Linear Model Assumptions (and Why)

1. Relationship is linear
   - Critical if we're using a line, but...
   - If not, can fit a polynomial or use other methods discussed later in this class
2. Observations are independent
   - Necessary for inference (hypothesis test results and confidence intervals) to be correct
   - Predictions could still be OK: as $n \to \infty$, we still will recover the correct relationship between explanatory and response variables
3. Residuals follow a normal distribution
   - Necessary for hypothesis test results and confidence intervals to be correct
   - Predictions could still be OK: as $n \to \infty$, we still will recover the correct relationship between explanatory and response variables
   - If residual distribution is not normal, estimation methods other than least squares could result in lower variance
4. Residuals have equal variance for all observations (homoskedastic)
   - Necessary for hypothesis test results and confidence intervals to be correct (but they might not be too far off anyways...)
   - Predictions could still be OK: as $n \to \infty$, we still will recover the correct relationship between explanatory and response variables
   - If residual distribution is not normal, estimation methods other than least squares could result in lower variance
5. No outliers/observations with high leverage
   - Could result in incorrect inferences and predictions, especially if n is small.

Summary: Mostly, these problems result in...

- A loss of guarantees of correct Type I Error rates for hypothesis tests
- A loss of guarantees of correct coverage rates for confidence intervals
- Higher-than-necessary variance for parameter estimates and predictions

Our Goal: Fix problems with residuals (non-normal, heteroskedastic/unequal variance), and maybe also outliers.

Method: Transform the variables.

The Ladder of Powers for Transformations

- Imagine a "ladder of powers" of y (or x): We start at y and go up or down the ladder.

$$
\begin{aligned}
&\vdots \\
&y^2 \\
&y \\
&\sqrt{y} \\
&y^{\text{“0”}} \text{ (we use } \log(y) \text{ here)} \\
&-1/\sqrt{y} \text{ (the } - \text{ keeps direction of association between } x \text{ and response)} \\
&-1/y \\
&-1/y^2 \\
&\vdots
\end{aligned}
$$

Which direction?

Idea 1: Tukey's Circle

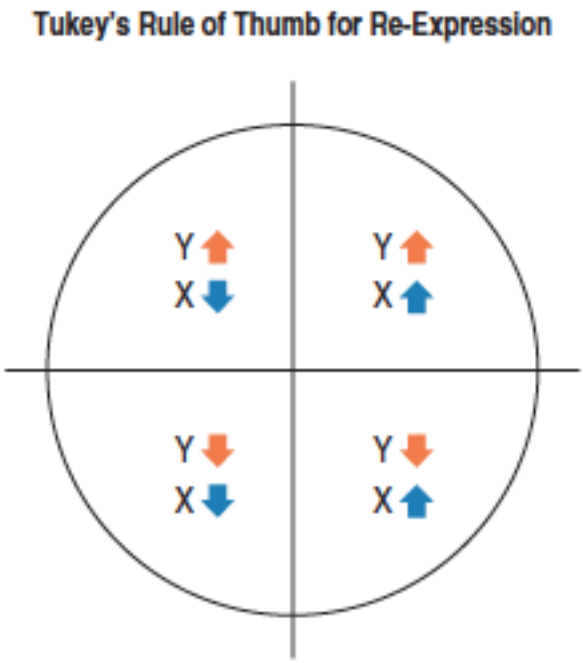


Figure 9.9

Tukey's circle of transformations shows what direction to move on the ladder for each variable depending on what quadrant the scatterplot resembles. For example, if the scatterplot is curved upward like quadrant 3 (bottom left) then either y or *x* should move down the ladder.

Idea 2: Towards normality.

- If a variable is skewed right, move it down the ladder (pull down large values)
- If a variable is skewed left, move it up the ladder (pull up small values)

Example

One last time, let's look at modeling a movie's international gross earnings in inflation-adjusted 2013 dollars (`intgross_2013`). For today, let's just think about using a single quantitative explanatory variable, `budget_2013`.

Here we read the data in and fit a simple linear regression model.

```
library(readr)
library(dplyr)
library(ggplot2) # general plotting functionality
library(GGally) # includes the ggpairs function, pairs plots via ggplot2
library(gridExtra) # for grid.arrange, which arranges the plots next to each other

options(na.action = na.exclude, digits = 7)

movies <- read_csv("http://www.evanlray.com/data/bechdel/bechdel.csv") %>%
  filter(mpaa_rating %in% c("G", "PG", "PG-13", "R"),
    !is.na(intgross_2013),
    !is.na(budget_2013))
```

Function for Model Fitting and Plotting Diagnostics

We're about to fit a bunch of different models and look at residual diagnostic plots for them all. Since we want to do slight variations on the same thing a bunch of times, we should make a function!

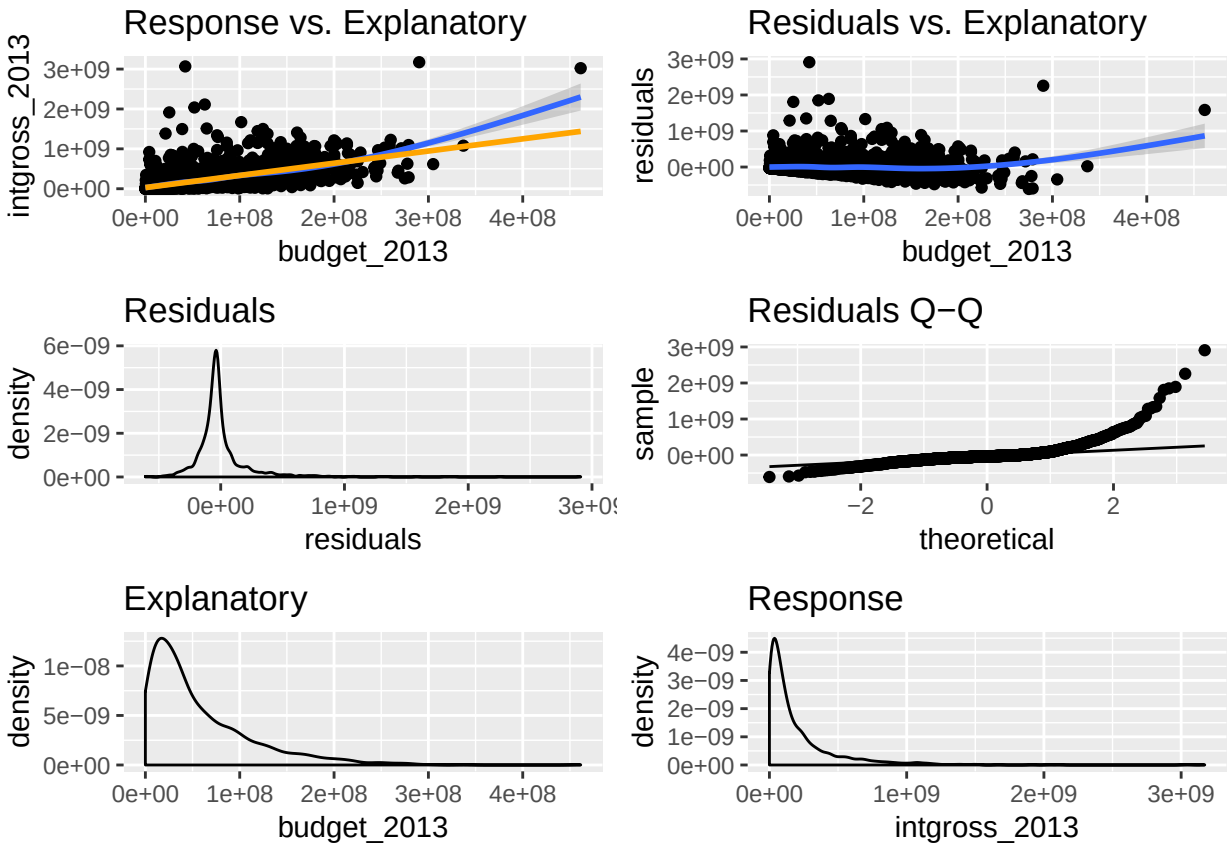
```
#' Fit a linear model with specified response and explanatory variables in the movies data set
#'
#' @param response character: response variable name
#' @param explanatory character: explanatory variable name
fit_model_and_make_plots <- function(response, explanatory) {
  fit_formula <- as.formula(paste0(response, " ~ ", explanatory))
  fit <- lm(fit_formula, data = movies)

  movies <- movies %>%
    mutate(
      residuals = residuals(fit),
      fitted = predict(fit)
    )

  p1 <- ggplot(data = movies, mapping = aes_string(x = explanatory, y = response)) +
    geom_point() +
    geom_smooth() +
    geom_smooth(method = "lm", color = "orange", se = FALSE) +
    ggtitle("Response vs. Explanatory")

  p2 <- ggplot(data = movies, mapping = aes_string(x = explanatory, y = "residuals")) +
    geom_point() +
    geom_smooth() +
    ggtitle("Residuals vs. Explanatory")

  p3 <- ggplot(data = movies, mapping = aes(x = residuals)) +
    geom_density() +
    ggtitle("Residuals")

  p4 <- ggplot(data = movies, mapping = aes(sample = residuals)) +
    stat_qq() +
    stat_qq_line() +
    ggtitle("Residuals Q-Q")
```

```
  p5 <- ggplot(data = movies, mapping = aes_string(x = explanatory)) +
    geom_density() +
    ggtitle("Explanatory")

  p6 <- ggplot(data = movies, mapping = aes_string(x = response)) +
    geom_density() +
    ggtitle("Response")

  grid.arrange(p1, p2, p3, p4, p5, p6, ncol = 2)
}
```

Linear Fit

```
fit_model_and_make_plots(response = "intgross_2013", explanatory = "budget_2013")
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```

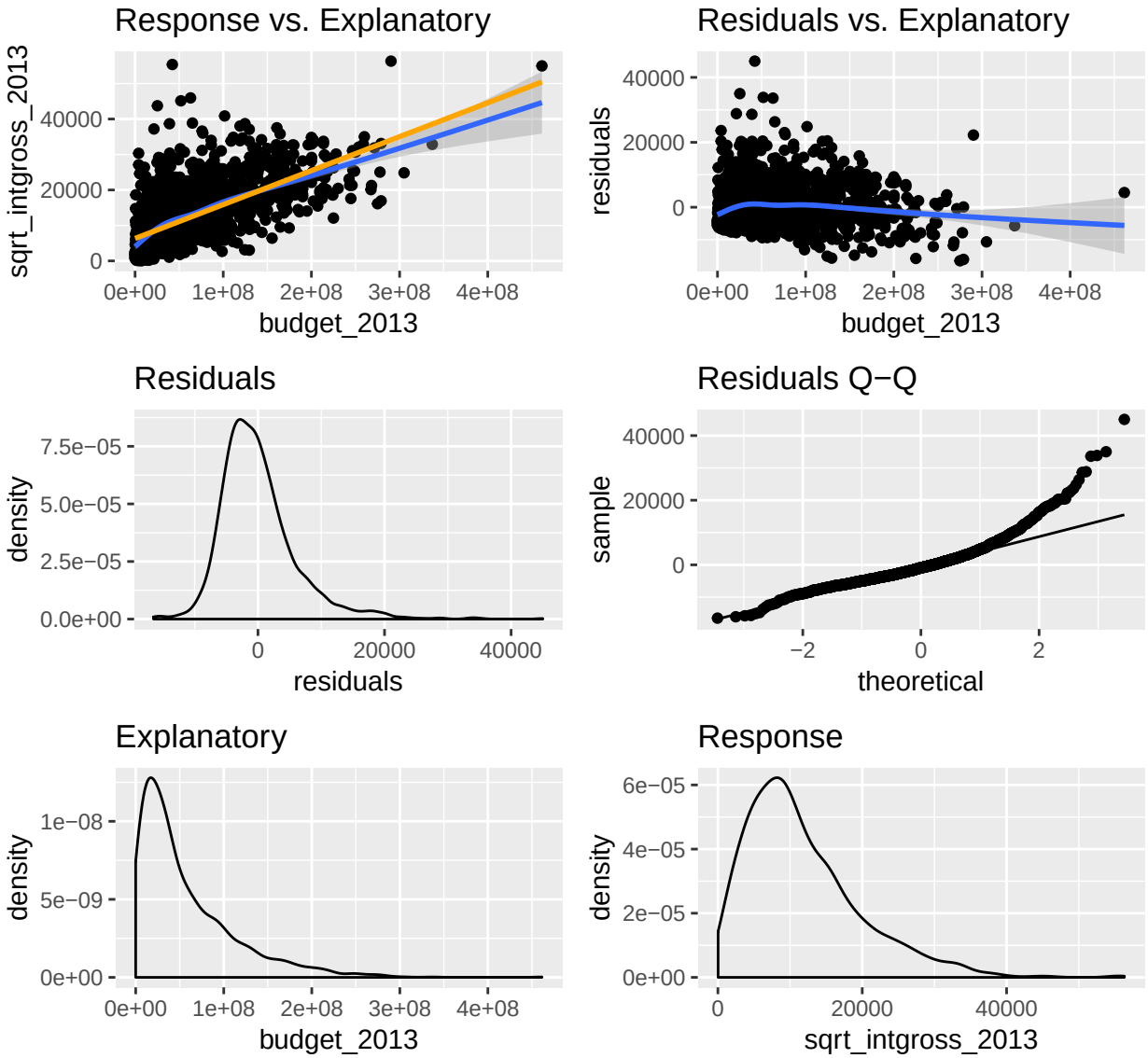


In our example, what are the problems and how are we going to fix them?

Trying $\sqrt{\text{intgross\_2013}}$

```
movies <- movies %>% mutate(
  sqrt_intgross_2013 = sqrt(intgross_2013)
)

fit_model_and_make_plots(response = "sqrt_intgross_2013", explanatory = "budget_2013")
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```

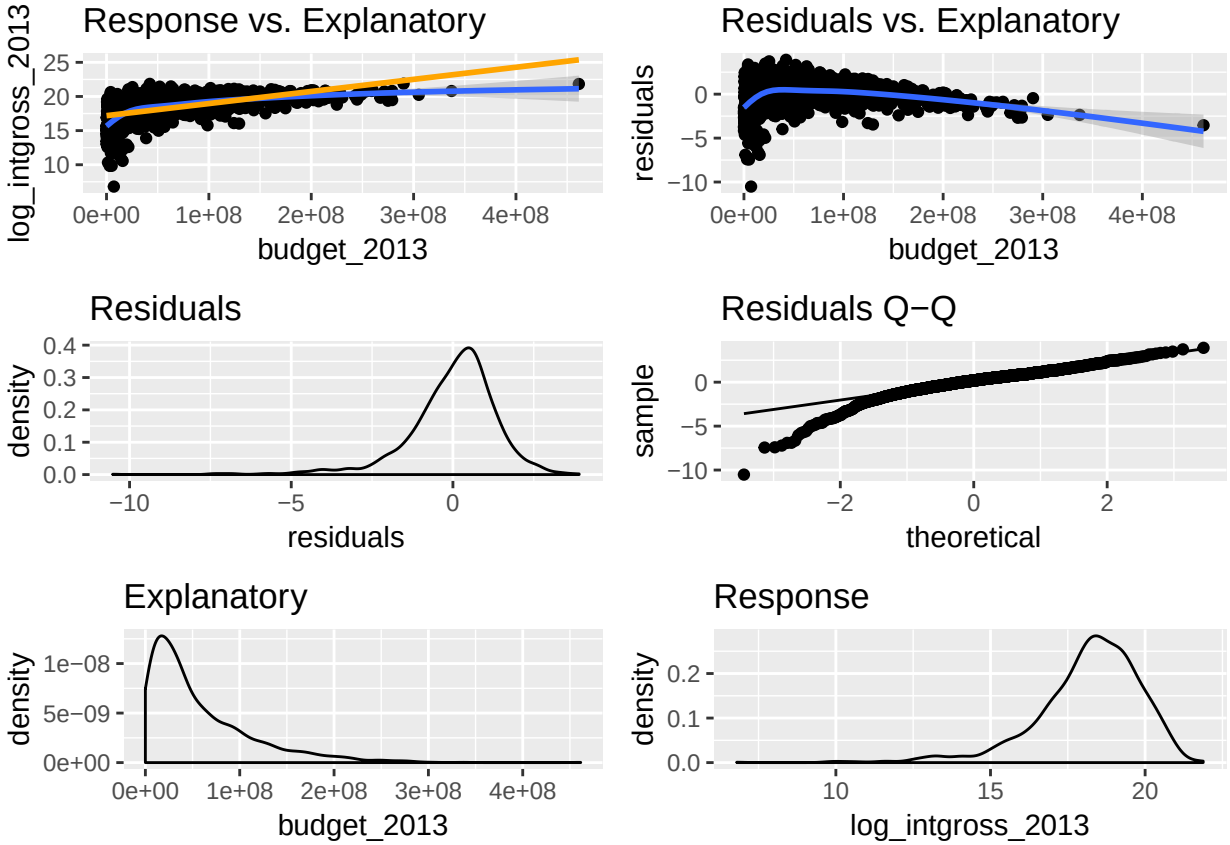


What do we think?

Trying log(intgross_2013)

```
movies <- movies %>% mutate(
  log_intgross_2013 = log(intgross_2013)
)

fit_model_and_make_plots(response = "log_intgross_2013", explanatory = "budget_2013")
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```

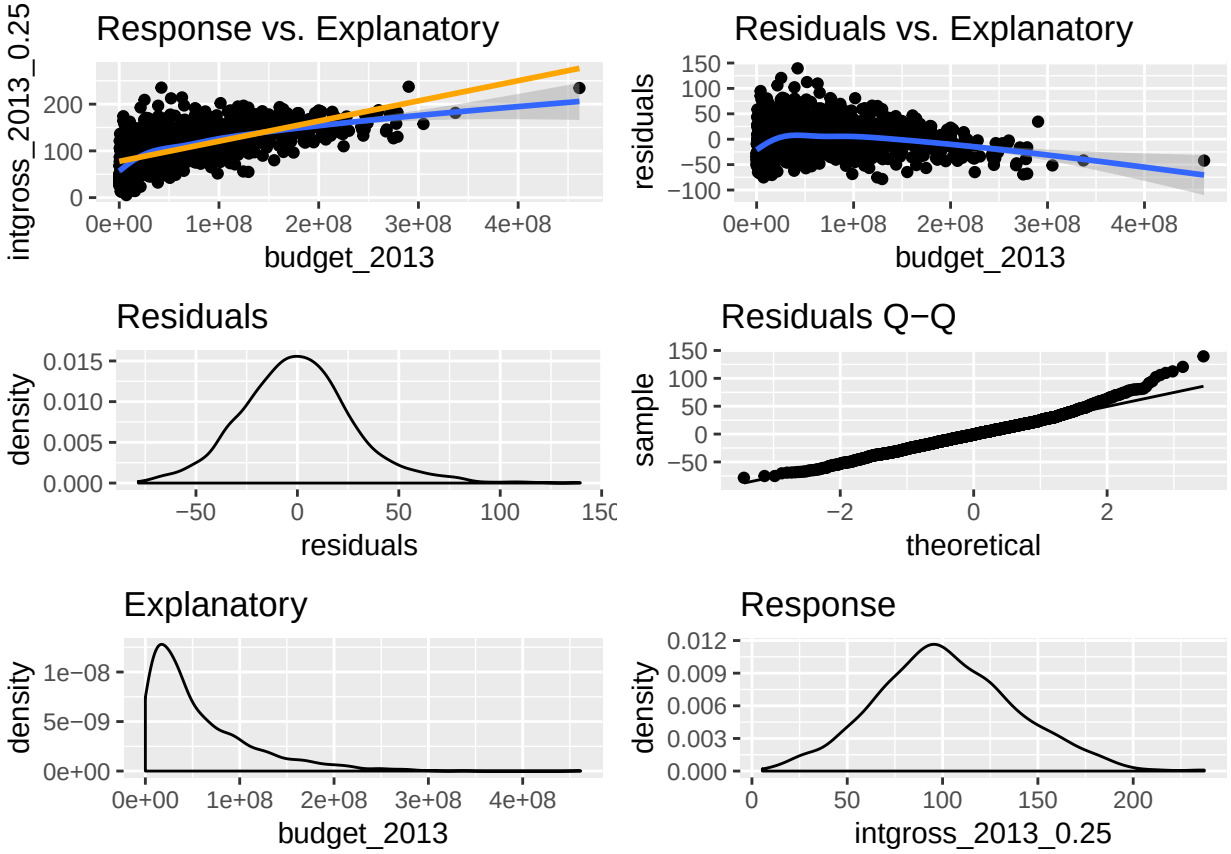


What do we think?

Trying intgross_2013$^{0.25}$

```
movies <- movies %>% mutate(
  intgross_2013_0.25 = intgross_2013^{0.25}
)

fit_model_and_make_plots(response = "intgross_2013_0.25", explanatory = "budget_2013")
```

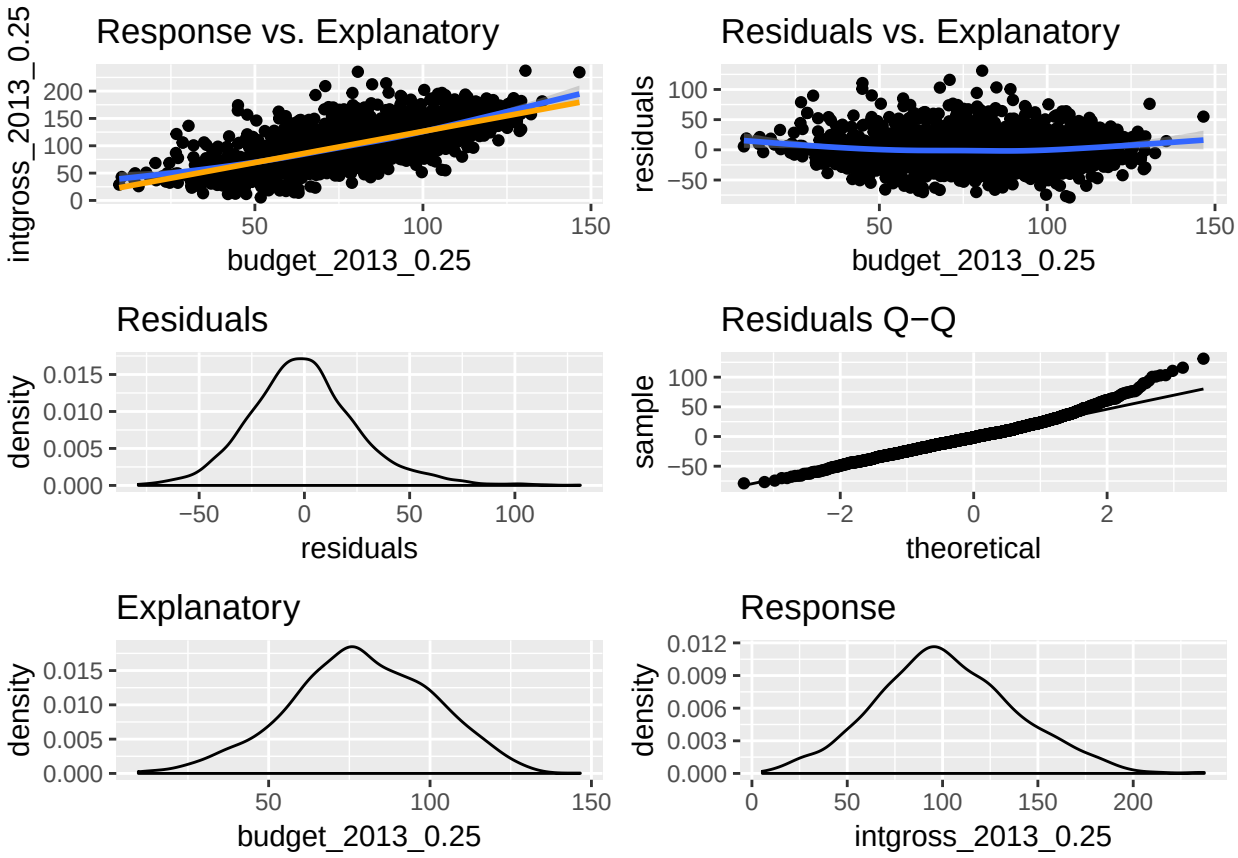
```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



Transformations of both variables...

```
movies <- movies %>% mutate(
  intgross_2013_0.25 = intgross_2013^{0.25},
  budget_2013_0.25 = budget_2013^{0.25}
)

fit_model_and_make_plots(response = "intgross_2013_0.25", explanatory = "budget_2013_0.25")
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



Box-Cox Power Transformations

Box-Cox transformations provide an automated way of choosing transformations to approximate normality.

The transformations come from the following family, indexed by parameter λ:

$$y_i^{(\lambda)} = \begin{cases} \frac{y_i^\lambda - 1}{\lambda} & \text{if } \lambda \neq 0 \\ \log(y_i) & \text{if } \lambda = 0 \end{cases}$$

Box and Cox developed an automatic procedure to choose λ so that the resulting transformed data are as close to normally distributed as possible. It uses maximum likelihood, but we won't get in to the details here.

Instead, we can use functions from the `car` package to perform Box-Cox transformations:

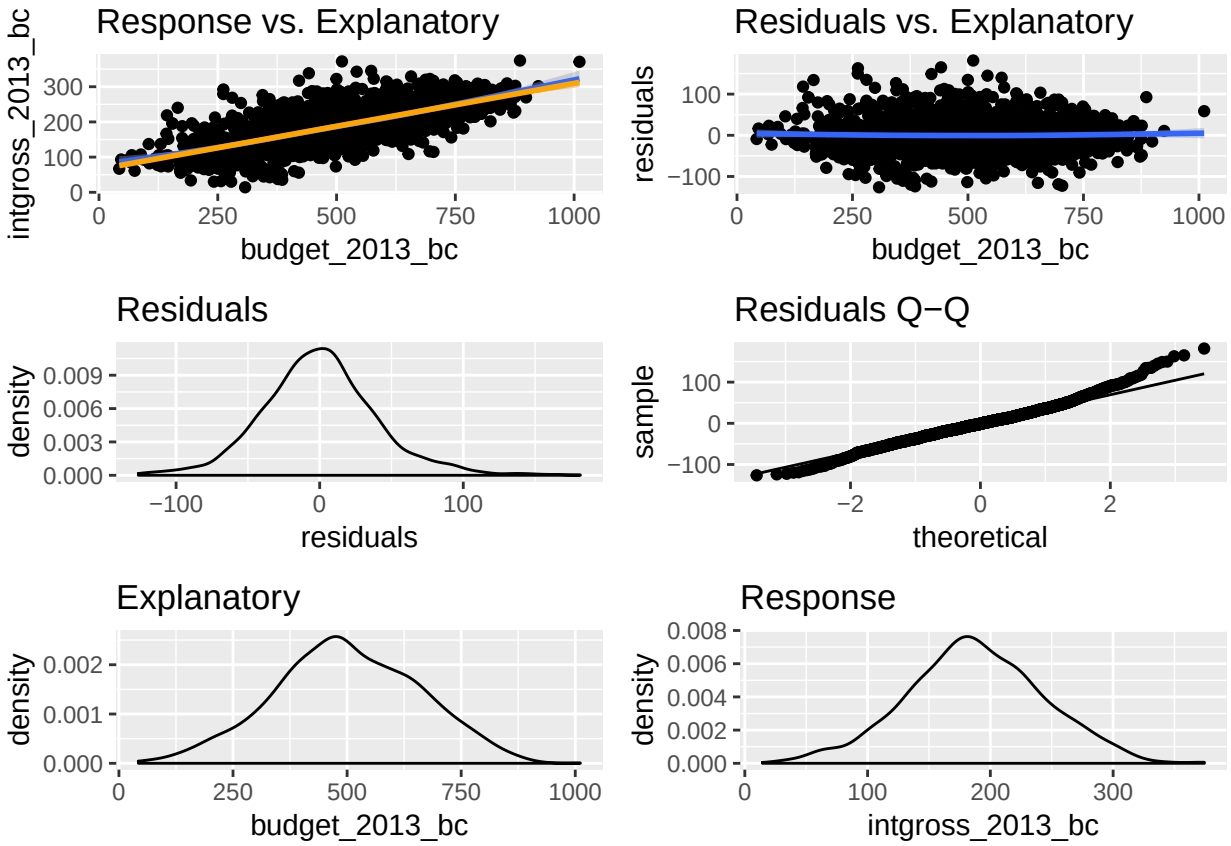
```
library(car)

# Estimate lambda - here, separately for each variable
bc_params_hat_intgross <- powerTransform(movies$intgross_2013, family = "bcPower")
bc_params_hat_intgross$lambda
```

```
## movies$intgross_2013
##            0.1972542
```

```
bc_params_hat_budget <- powerTransform(movies$budget_2013, family = "bcPower")
bc_params_hat_budget$lambda
```

```
## movies$budget_2013
##          0.2838691
```

```
# Add transformed variables to our data set
movies <- movies %>% mutate(
  intgross_2013_bc = bcPower(intgross_2013, lambda = bc_params_hat_intgross$lambda),
  budget_2013_bc = bcPower(budget_2013, lambda = bc_params_hat_budget$lambda)
)

fit_model_and_make_plots(response = "intgross_2013_bc", explanatory = "budget_2013_bc")
```

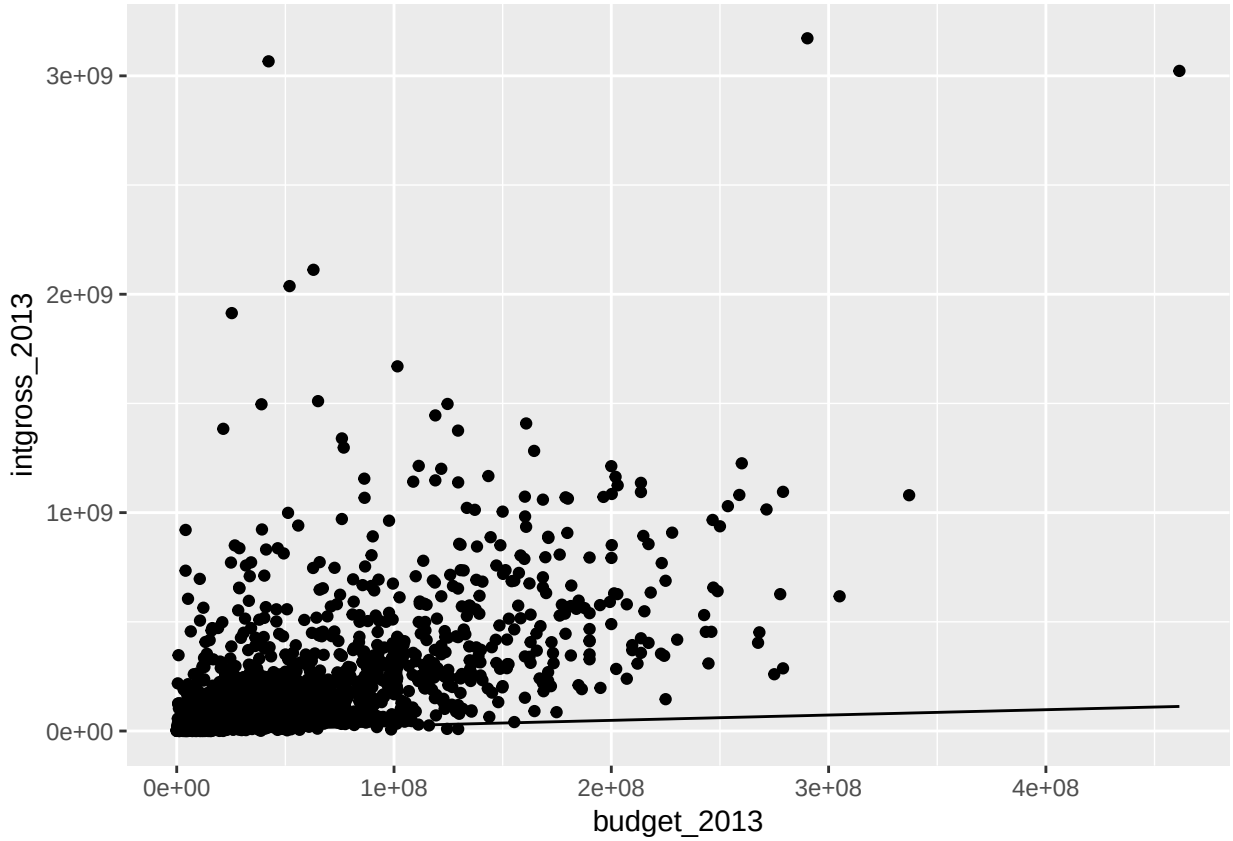


Take Care!

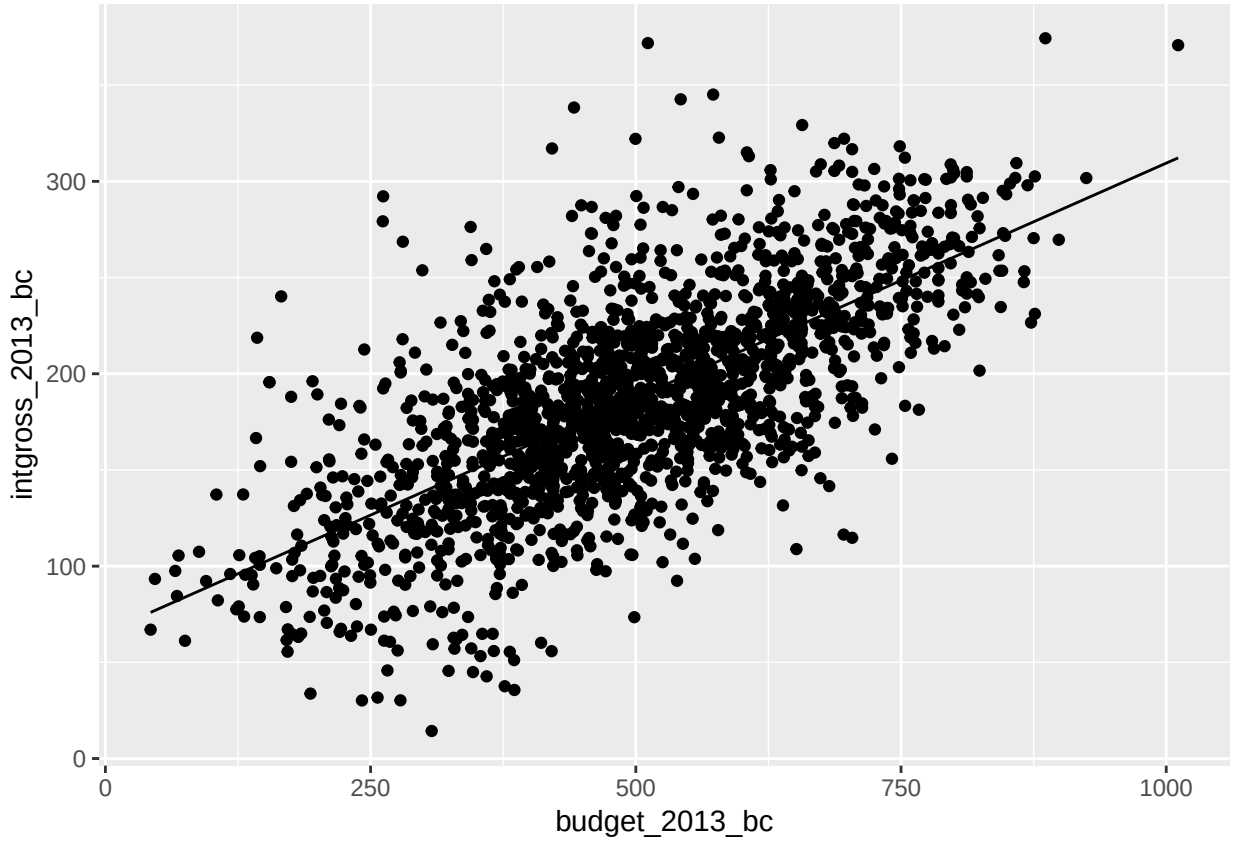
```
fit_bc <- lm(intgross_2013_bc ~ budget_2013_bc, data = movies)
summary(fit_bc)
```

```
##
## Call:
## lm(formula = intgross_2013_bc ~ budget_2013_bc, data = movies)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -126.264  -24.811   -1.128   22.693  181.566
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)    65.570173   3.172075   20.67   <2e-16 ***
## budget_2013_bc  0.243903   0.006046   40.34   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 40 on 1744 degrees of freedom
## Multiple R-squared:  0.4827, Adjusted R-squared:  0.4824
## F-statistic:  1627 on 1 and 1744 DF,  p-value: < 2.2e-16
```

```
predict_transformed_scale <- function(x) {
  predict(fit_bc, data.frame(budget_2013_bc = x))
}

ggplot(data = movies, mapping = aes(y = intgross_2013, x = budget_2013)) +
  geom_point() +
  stat_function(fun = predict_transformed_scale)
```




```
ggplot(data = movies, mapping = aes(y = intgross_2013_bc, x = budget_2013_bc)) +
  geom_point() +
  stat_function(fun = predict_transformed_scale)
```



We have to be careful about the transformations!

```r
#' Invert a Box-Cox transformation.  See car::bcPower for the original
#' transformation.
#'
#' @param y_transformed a univariate numeric vector with data on the transformed scale
#' @param lambda exponent for Box-Cox transformation
#'
#' @return a de-transformed numeric vector
invert_bc_transform <- function(y_transformed, lambda) {
  ## 1) undo box-cox
  if(abs(lambda) <= 1e-10) {
    y <- exp(y_transformed)
  } else {
    y <- (lambda * y_transformed + 1)^(1 / lambda)
  }

  return(y)
}

predict_bc_inverted <- function(x, lambda_x, lambda_y) {
  x_transformed <- bcPower(x, lambda = lambda_x)
  y_hat_transformed <- predict(fit_bc, data.frame(budget_2013_bc = x_transformed))
  y_hat <- invert_bc_transform(y_hat_transformed, lambda = lambda_y)
}

ggplot(data = movies, mapping = aes(y = intgross_2013, x = budget_2013)) +
  geom_point() +
  stat_function(fun = predict_bc_inverted,
    args = list(lambda_x = bc_params_hat_budget$lambda,
      lambda_y = bc_params_hat_intgross$lambda))
```