Simple Linear Regression: Conditions and Transformations

Sleuth3 Chapter 8

Simple Linear Regression Model and Conditions

- Observations follow a normal distribution with mean that is a linear function of the explanatory variable
- $Y_i \sim \text{Normal}(\beta_0 + \beta_1 X_i, \sigma)$

Conditions: spells "LINE-O"

- **Linear** relationship between explanatory variable and (population mean of) response variable: $\mu = \beta_0 + \beta_1 X$
- **Independent** observations (knowing that one observation is above its mean wouldn't give you any information about whether or not another observation is above its mean)
- **Normal** distribution of responses around the line
- **Equal standard deviation** of response for all values of X
  - Denote this standard deviation by σ
- **no Outliers** (not a formal part of the model, but important to check in practice)

Transformations

- Transformations can sometimes help with the following issues:
  - skewed distributions (but skewness is only a problem if it is very serious)
  - unequal standard deviation for different values of X
  - outliers (but usually only if this is a side effect of serious skewness)
  - non-linear relationship

Reminder of the Ladder of Powers

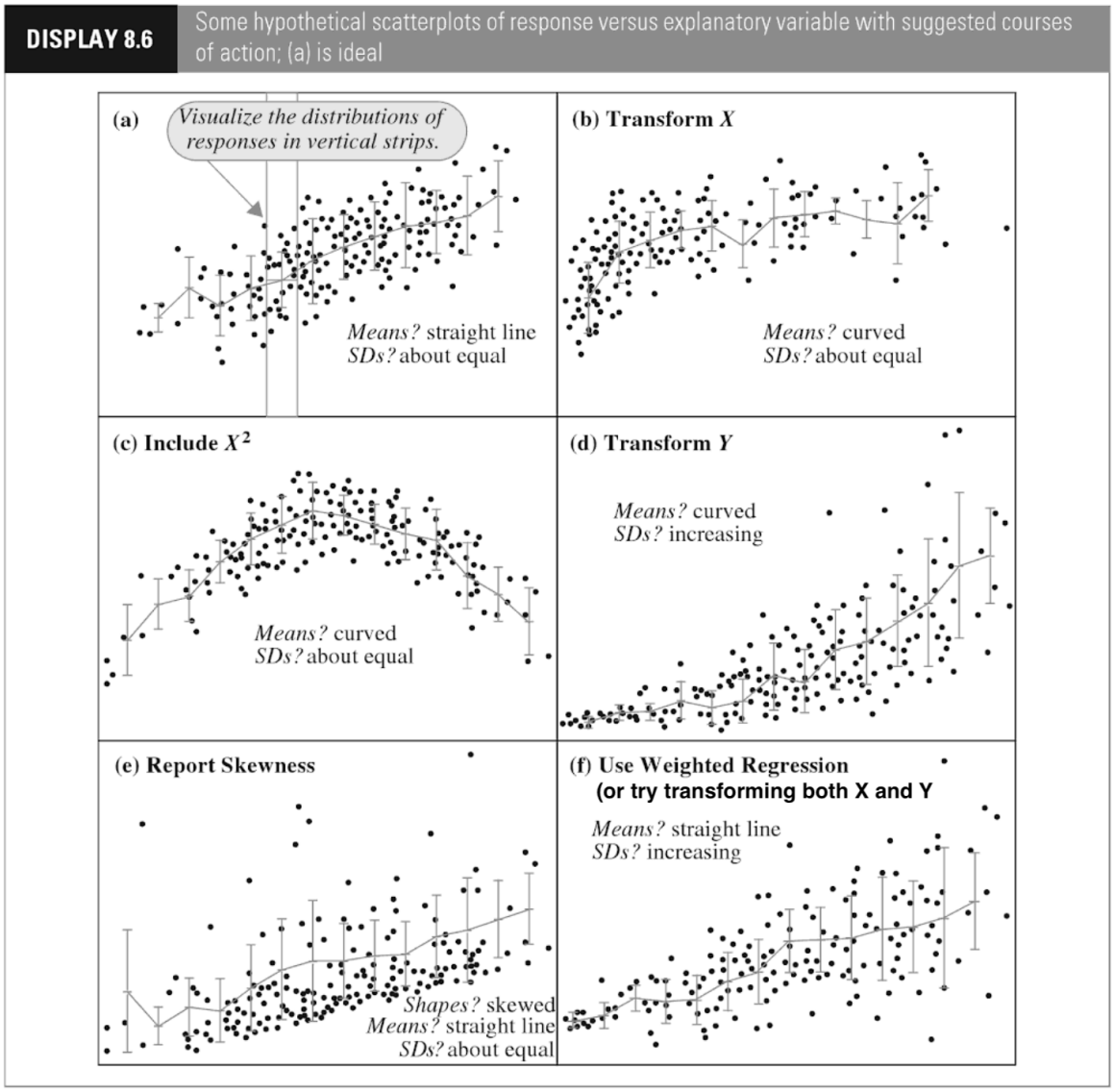
- We start at y and go up or down the ladder.

Transformation	R Code	Comments
$\vdots$		
e^y	`exp(y)`	Exactly where on the ladder the exponential transformation belongs depends on the magnitude of the data, but somewhere around here...
y^2	`y^2`	
y		Start here (no transformation)
$\sqrt{y}$	`sqrt(y)`	
$y^{\text{"0"}}$	`log(y)`	We use $\log(y)$ here
$-1/\sqrt{y}$	`-1/sqrt(y)`	The $-$ keeps the values of y in order
$-1/y$	`-1/y`	
$-1/y^2$	`-1/y^2`	
$\vdots$		

- Which direction?
  - 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)

What to do is based on scatter plots

Figure from The Statistical Sleuth.



Start with the response

Start exploring transformations by looking at the response variable, looking to fix: * Residuals skewed * Non-constant variance (heteroskedasticity)

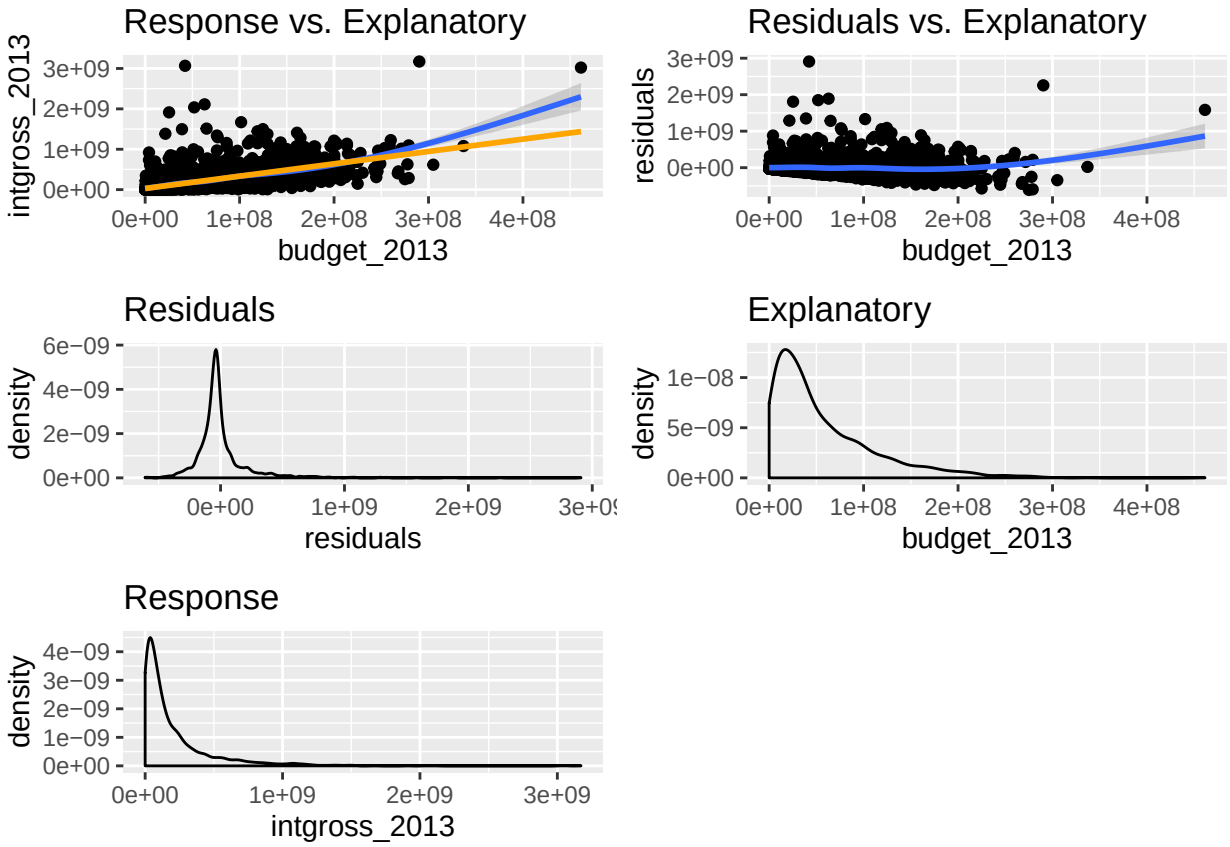
Example

Let's look at modeling a movie's international gross earnings in inflation-adjusted 2013 dollars (`intgross_2013`). Our explanatory variable is the movie's budget, `budget_2013`.

Linear Fit

```
fit <- lm(intgross_2013 ~ budget_2013, data = movies)
movies <- movies %>%
  mutate(
    residuals = residuals(fit),
    fitted = predict(fit)
  )
p1 <- ggplot(data = movies, mapping = aes(x = budget_2013, y = intgross_2013)) +
  geom_point() +
  geom_smooth() +
  geom_smooth(method = "lm", color = "orange", se = FALSE) +
  ggtitle("Response vs. Explanatory")
p2 <- ggplot(data = movies, mapping = aes(x = budget_2013, 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(x = budget_2013)) +
  geom_density() +
  ggtitle("Explanatory")
p5 <- ggplot(data = movies, mapping = aes(x = intgross_2013)) +
  geom_density() +
  ggtitle("Response")
grid.arrange(p1, p2, p3, p4, p5, ncol = 2)
```

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



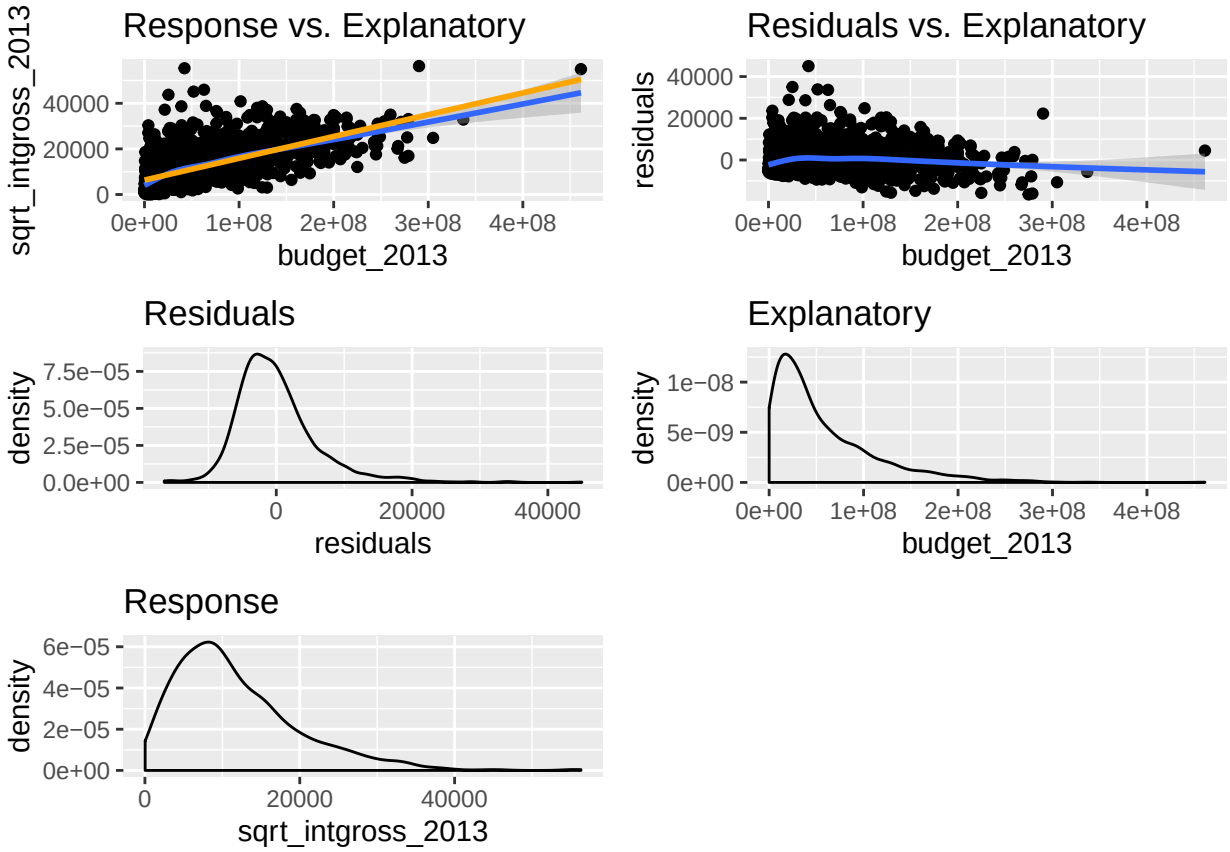
In this example, what are the problems and how are we going to fix them?

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

```
movies <- movies %>% mutate(
  sqrt_intgross_2013 = sqrt(intgross_2013)
)
fit <- lm(sqrt_intgross_2013 ~ budget_2013, data = movies)
```

I've hidden the code for the plots, but here they are:

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



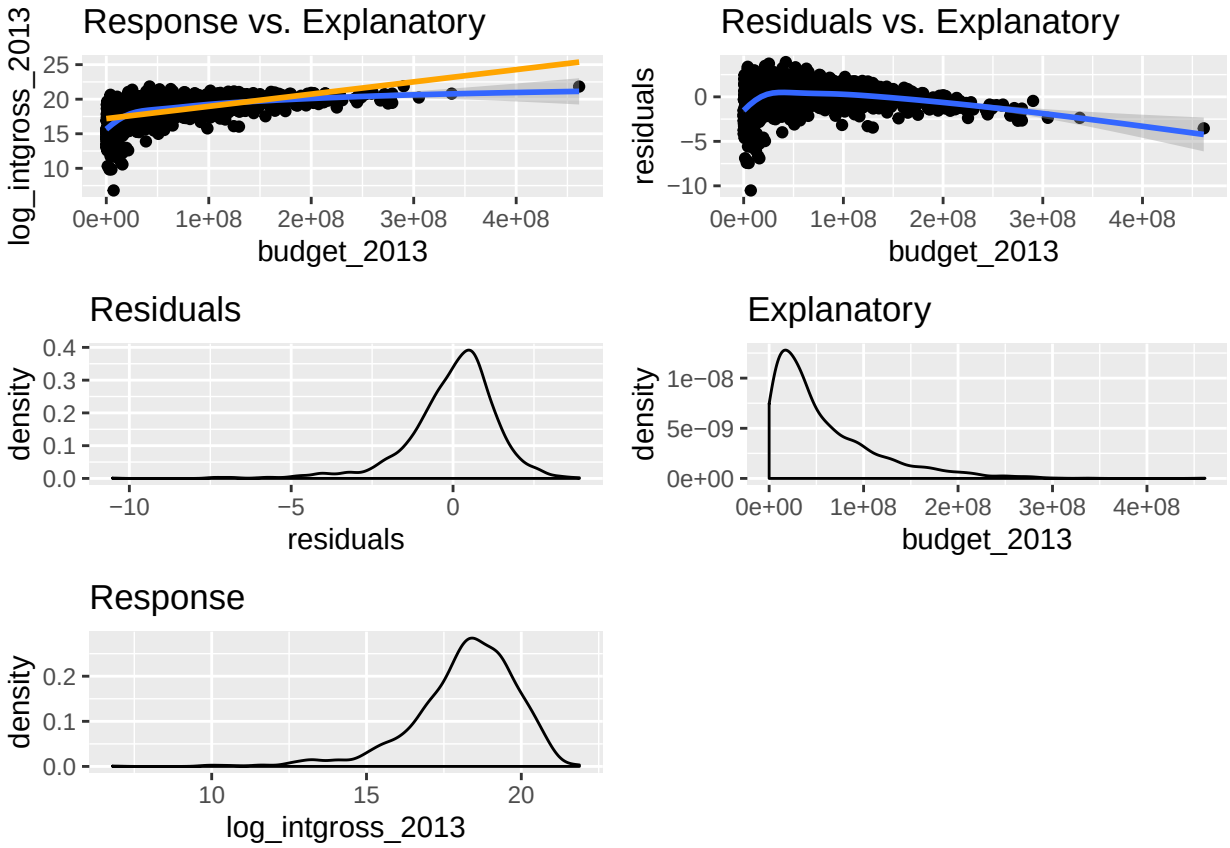
What do we think?

Trying log(**intgross_2013**)

```
movies <- movies %>% mutate(
  log_intgross_2013 = log(intgross_2013)
)
fit <- lm(log_intgross_2013 ~ budget_2013, data = movies)
```

I've hidden the code for the plots, but here they are:

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



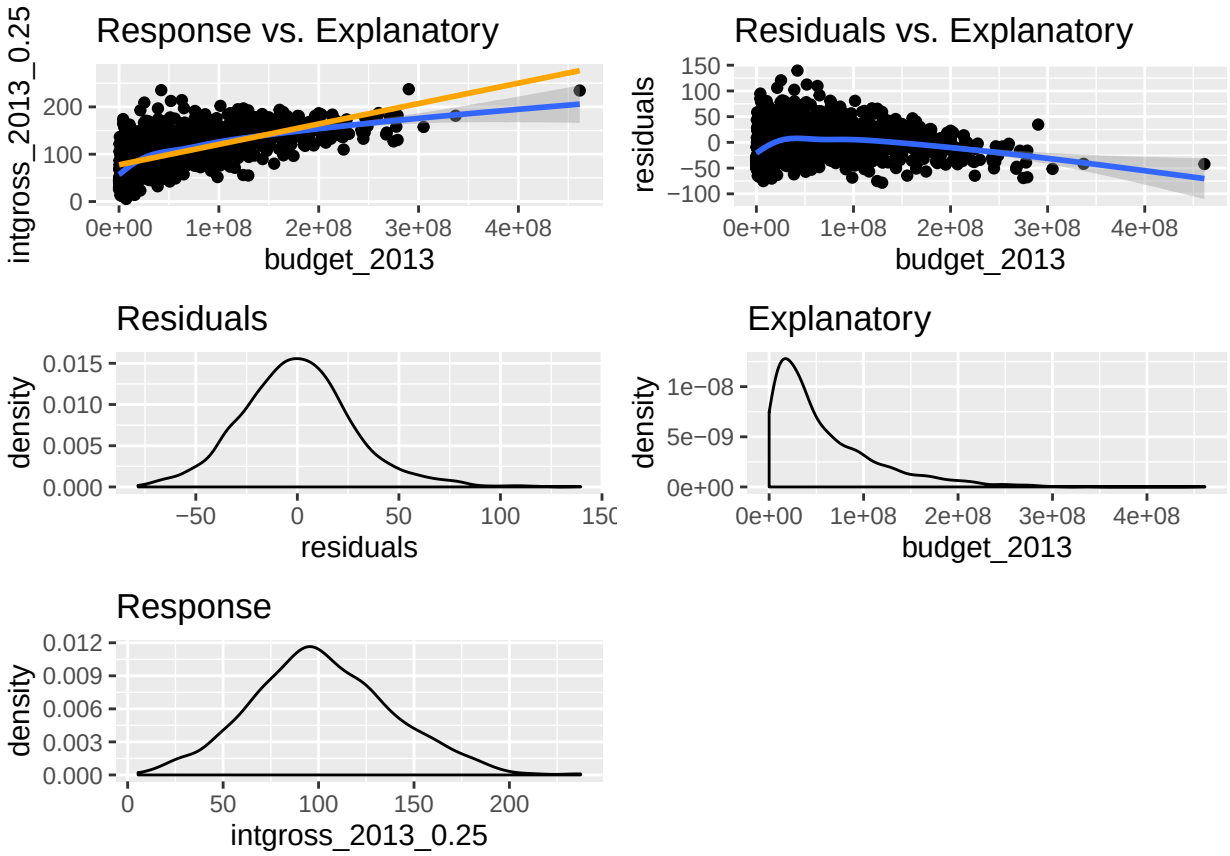
What do we think?

Trying intgross_2013$^{0.25}$

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

I've hidden the code for the plots, but here they are:

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

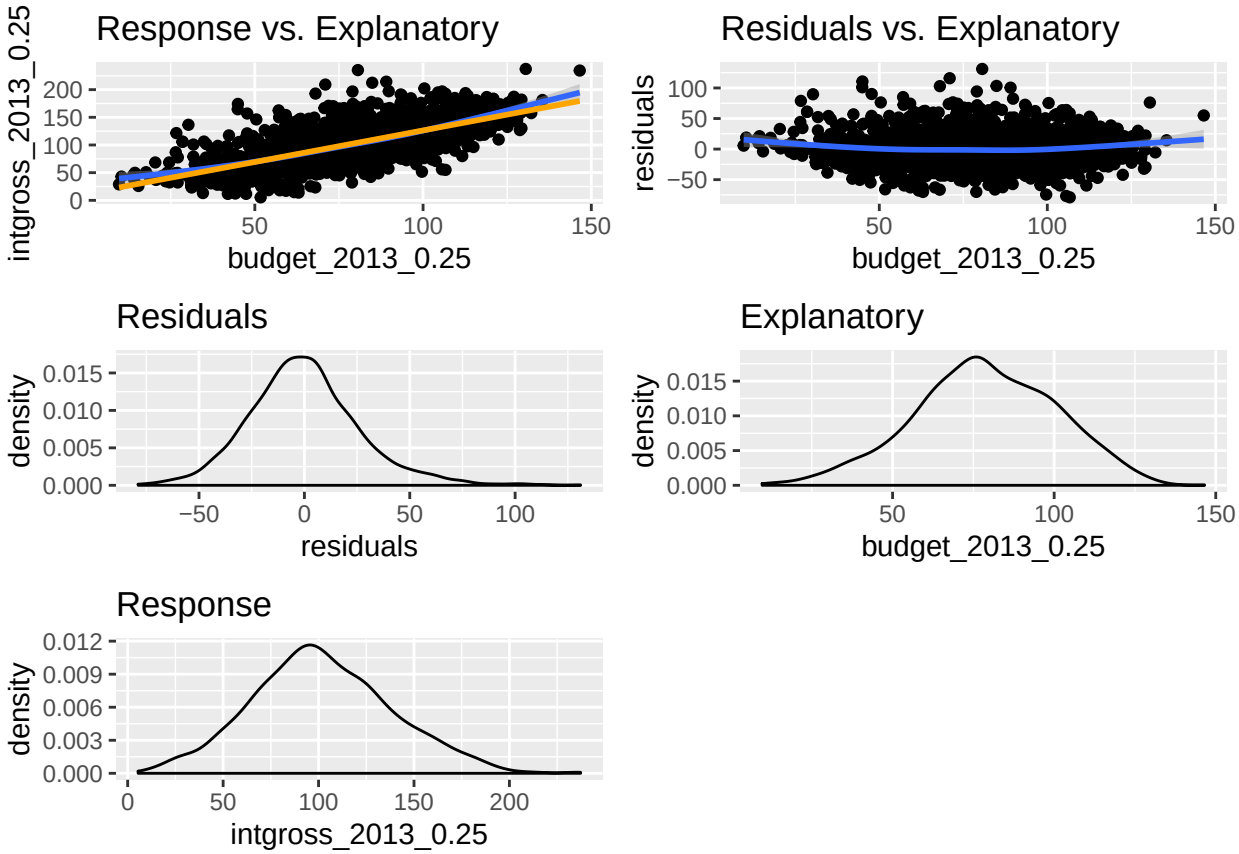


Transformations of both variables...

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

I've hidden the code for the plots, but here they are:

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



Making Predictions in Models with Transformed Variables

Suppose we want to estimate the mean earnings for the population of movies with budgets of 100 million dollars.

We need to do the following steps:

- Transform 100 million dollars to the power of 0.25 to get the input to our model
- Use the model to get the estimated mean of earnings to the power of 0.25
- Raise the estimated mean to the power of 4 to get estimated mean earnings

```
# set up data frame with value of budget at which we want to make a prediction
predict_df <- data.frame(
  budget_2013 = 100000000
)
# transform budget to get the input to our linear model
predict_df <- predict_df %>%
  mutate(
    budget_2013_0.25 = budget_2013^0.25
  )
# predictions based on transformed budget for the test set
# the result is a prediction of (intgross_2013)^0.25
predict_df <- predict_df %>%
  mutate(
    predicted_intgross_2013_0.25 = predict(fit, newdata = predict_df)
  )
# undo the transformation of the response to get predictions of intgross_2013
predict_df <- predict_df %>%
  mutate(
    predicted_intgross_2013 = predicted_intgross_2013_0.25^4
  )
# take a look at the results
predict_df
```

```
##   budget_2013 budget_2013_0.25 predicted_intgross_2013_0.25
## 1       1e+08              100                      126.3231
##   predicted_intgross_2013
## 1               254642931
```

We can apply the same ideas to confidence intervals for the mean gross earnings or prediction intervals for the gross earnings of a particular movie.

```
predict(fit, newdata = predict_df, interval = "confidence")
```

```
##        fit      lwr      upr
## 1 126.3231 124.6004 128.0459
```

```
124.60^4
```

```
## [1] 241030593
```

```
128.05^4
```

```
## [1] 268855132
```

```
predict(fit, newdata = predict_df, interval = "prediction")
```

```
##        fit     lwr      upr
## 1 126.3231 74.5401 178.1062
```

```
74.54^4
```

```
## [1] 30871487
```

```
178.11^4
```

```
## [1] 1006359648
```

Check out how these compare to the data:

```
predict_df <- predict_df %>%
  mutate(
    CI_lower = 124.60^4,
    CI_upper = 128.05^4,
    PI_lower = 74.54^4,
    PI_upper = 178.11^4
  )
predict_df
```

```
##   budget_2013 budget_2013_0.25 predicted_intgross_2013_0.25
## 1       1e+08              100                      126.3231
##   predicted_intgross_2013  CI_lower  CI_upper PI_lower   PI_upper
## 1               254642931 241030593 268855132 30871487 1006359648
```

```
ggplot() +
  geom_point(data = movies, mapping = aes(x = budget_2013, y = intgross_2013)) +
  geom_errorbar(data = predict_df,
    mapping = aes(x = budget_2013, ymin = PI_lower, ymax = PI_upper),
    color = "cornflowerblue",
    size = 1, # how many pixels wide is the error bar; 1 pixel is often good
    width = 10000000) + # in the units of the X variable; in most data sets you'd use a smaller width
  geom_errorbar(data = predict_df,
    mapping = aes(x = budget_2013, ymin = CI_lower, ymax = CI_upper),
    color = "orange",
    size = 1, # how many pixels wide is the error bar; 1 pixel is often good
    width = 10000000) # in the units of the X variable; in most data sets you'd use a smaller width
```