Linear Regression Using R Datasets

Code

Welcome to the off-platform project for linear regression in R! There are two main goals of this project. The first is to investigate the datasets that come built-in to R. The second is to learn how to quickly write and interpret linear regression models.

As you move through this project you'll begin to see that creating and analyzing linear regression model takes very few lines of code. The tricky work of being a data scientist begins to shift from writing your code to explaining your results.

This project is written in an R Notebook. We suggest opening this file using RStudio. If you click the "Preview" button, you should see these instructions rendered to an HTML page. We suggest following along with these instructions from the Preview window while coding along in the provided code blocks.

Let's get started!

Investigating the R Datasets Package

When you first downloaded R to your computer, you also downloaded several datasets. These can be found in the `datasets` package. Let's begin by investigating this package by looking at the documentation. First make sure the `datasets` package is loaded by calling `library(datasets)`. Then type `?datasets` in the code block below. You should see the documentation for the `datasets` package appear in the "Help" tab in RStudio.

Hide

```
# Load datasets and look at its documentation
library(datasets)
?datasets
```

That documentation didn't tell us much — but it did give us a hint on where to go to find a complete list of the datasets. Call the line of code recommended in the documentation of `datasets`. Note that this will open a new tab in RStudio. Make sure to come back to this tab after looking through the datasets.

Hide

```
library(help = "datasets")
```

You should see a list of datasets. The documentation gives us a brief description on the datasets, but it's still a bit tough to understand what the data actually looks like. You can access a dataset in R using the notation `datasets::dataset_name`. For example, take a look at the head of the dataset named `ToothGrowth` using the line `head(datasets::ToothGrowth)`. What columns exist? You can also simply use `ToothGrowth` to access the dataset — you don't need to include the name of the package that it came from.

Finally, you can pull up the documentation for the data using `?ToothGrowth`. What does the `supp` column represent in this case?

Hide

```
# Print the head of the ToothGrowth dataset
head(datasets::ToothGrowth)
```

	len <dbl>	supp <fctr>	dose <dbl>
1	4.2	VC	0.5
2	11.5	VC	0.5
3	7.3	VC	0.5
4	5.8	VC	0.5
5	6.4	VC	0.5
6	10.0	VC	0.5

6 rows

Hide

```
?ToothGrowth
```

Great! We now have the tools to explore all of the datasets that come with R. Let's now think about which ones might be useful for linear regression. With linear regression, we're looking for a dataset where one column might be explained by another (or in the case of multiple linear regression, one column being explained by multiple other columns).

Try to find at least 3 different datasets that might be good candidates for linear regression. Use the code block below to look up the documentation for several datasets. In the next section we'll begin the first step of statistical model building — confirming data assumptions.

Hide

```
# Find at least three datasets that seem like good candidates for linear regression
?ToothGrowth
?cars
?state.x77
```

Confirming Data Assumptions

In this section, we'll walk through a few datasets that both meet and do not meet the assumptions necessary for linear regression. We'll also leave space for you to investigate datasets that you're intrested.

Let's start be testing the first assumption needed for linear regression — the data needs to be roughly linear. Let's do that visually using the `ggplot2` package (make sure this package is installed and loaded).

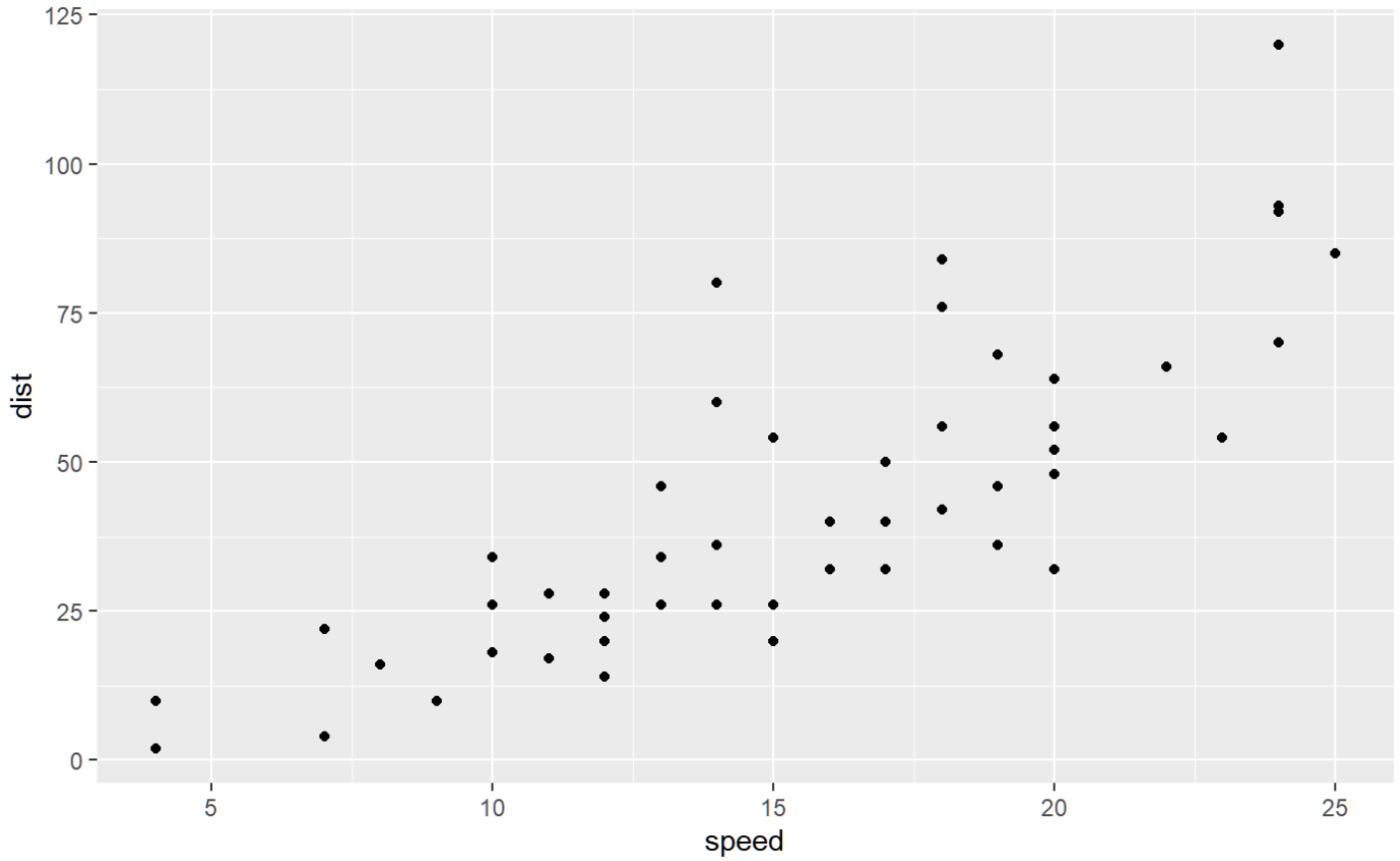
To do so, pass your dataset into a call to `ggplot()` and include an `aes()` with the columns that should go on the x and y axes. Then add a `geom_point()` layer. For example, `ggplot(cars, aes(speed, dist)) + geom_point()` will create a scatter plot showing the speed of cars on the x axis and the distance it took them to stop on the y axis. Does that look like a linear relationship?

We also created a graph using the `state.x77` dataset where `Illiteracy` was on the x axis and `Income` was on the y axis. There's one small hiccup in using this dataset. `state.x77` isn't a data frame. Use `as.data.frame(state.x77)` to transform it into a data frame. Does that one look linear?

Finally, we created a graph using the `pressure` dataset. This dataset descibes the relationship between the temperature and vapor pressure of mercury. Plot `temperature` on the x axis and `pressure` on the y axis. What looks different about this graph?

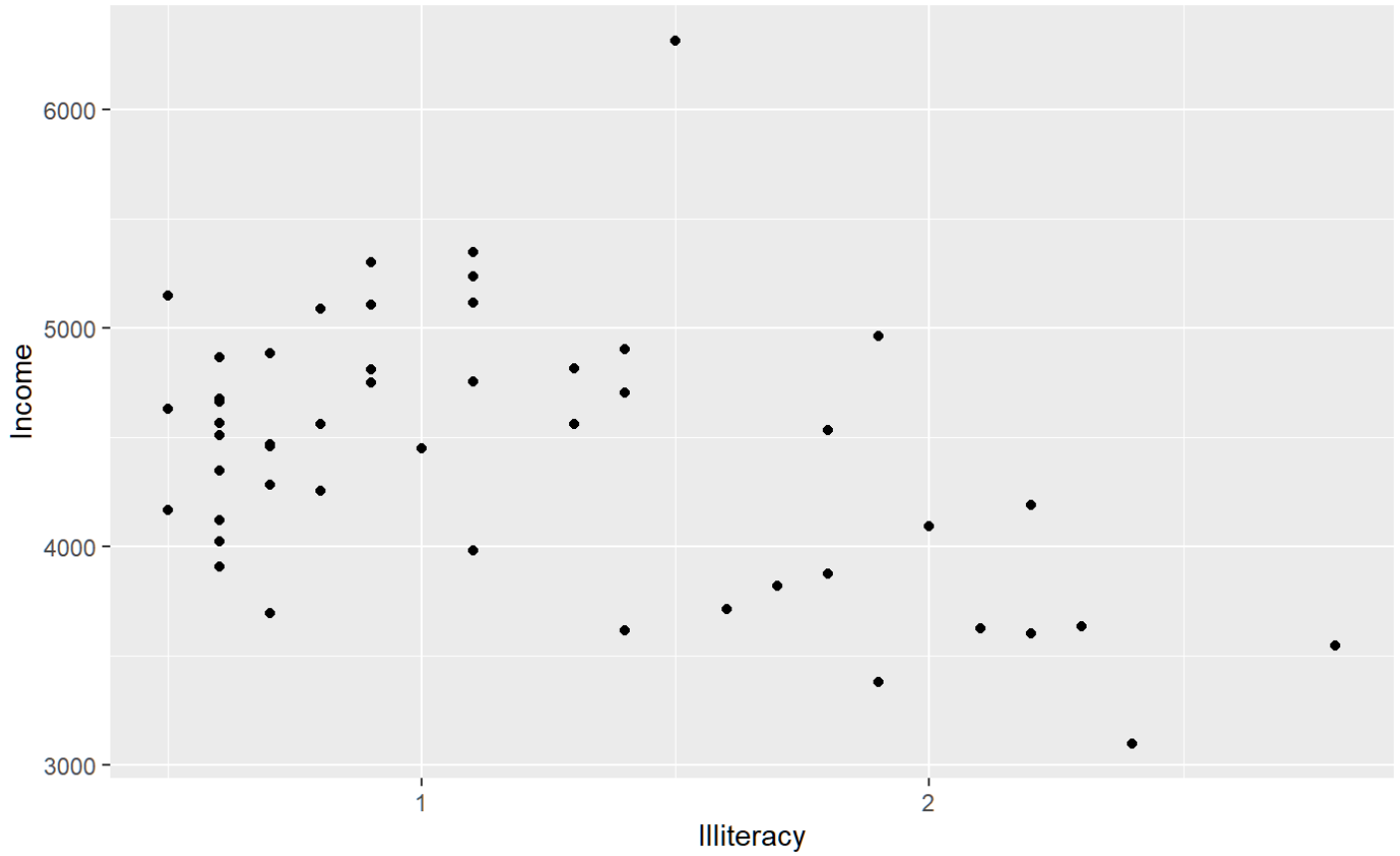
Hide

```
# Load the ggplot2 package
library(ggplot2)

# Plot your datasets
ggplot(cars, aes(speed, dist)) +
  geom_point()
```



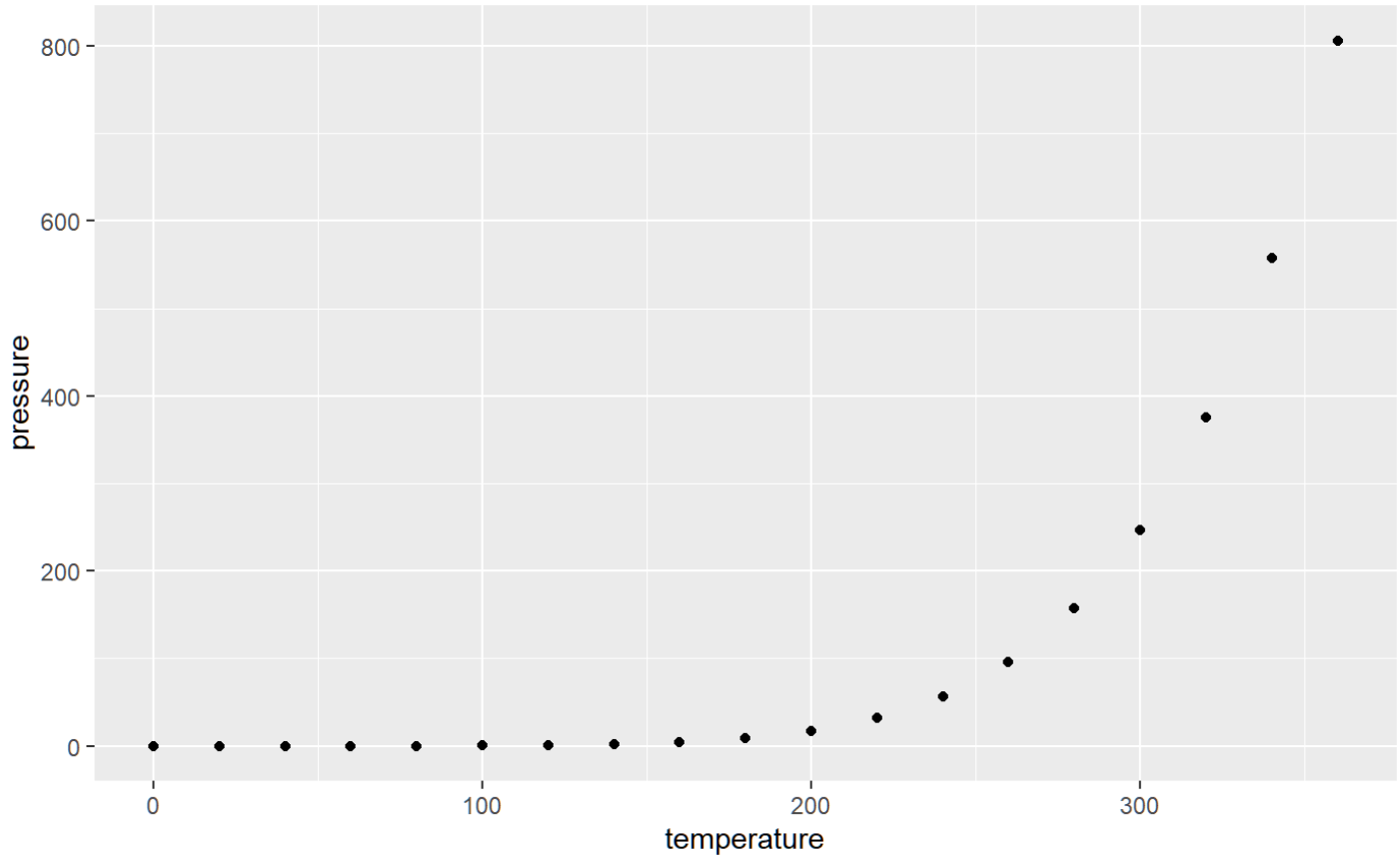
Hide

```
state_df = as.data.frame(state.x77)
ggplot(state_df, aes(Illiteracy, Income)) +
  geom_point()
```



Hide

```
ggplot(pressure, aes(temperature, pressure)) +
  geom_point()
```



That third graph certainly didn't look linear. This looks to be a quadratic or exponential relationship — linear regression wouldn't be appropriate.

Another way of testing for a linear relationship is to compute the correlation coefficient using `cor.test()`. Pass the two columns of your dataset into this function to see the correlation coefficient.

Was there a stronger correlation between the speed and stopping distance in the car dataset or between state illiteracy and state income? Remember, the relationship is stronger if the correlation coefficient is further away from 0.

Hide

```
# Find the correlation coefficient
cor.test(cars$speed, cars$dist)
```

```

    Pearson's product-moment correlation

data:  cars$speed and cars$dist
t = 9.464, df = 48, p-value = 1.49e-12
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.6816422 0.8862036
sample estimates:
      cor
0.8068949
```

Hide

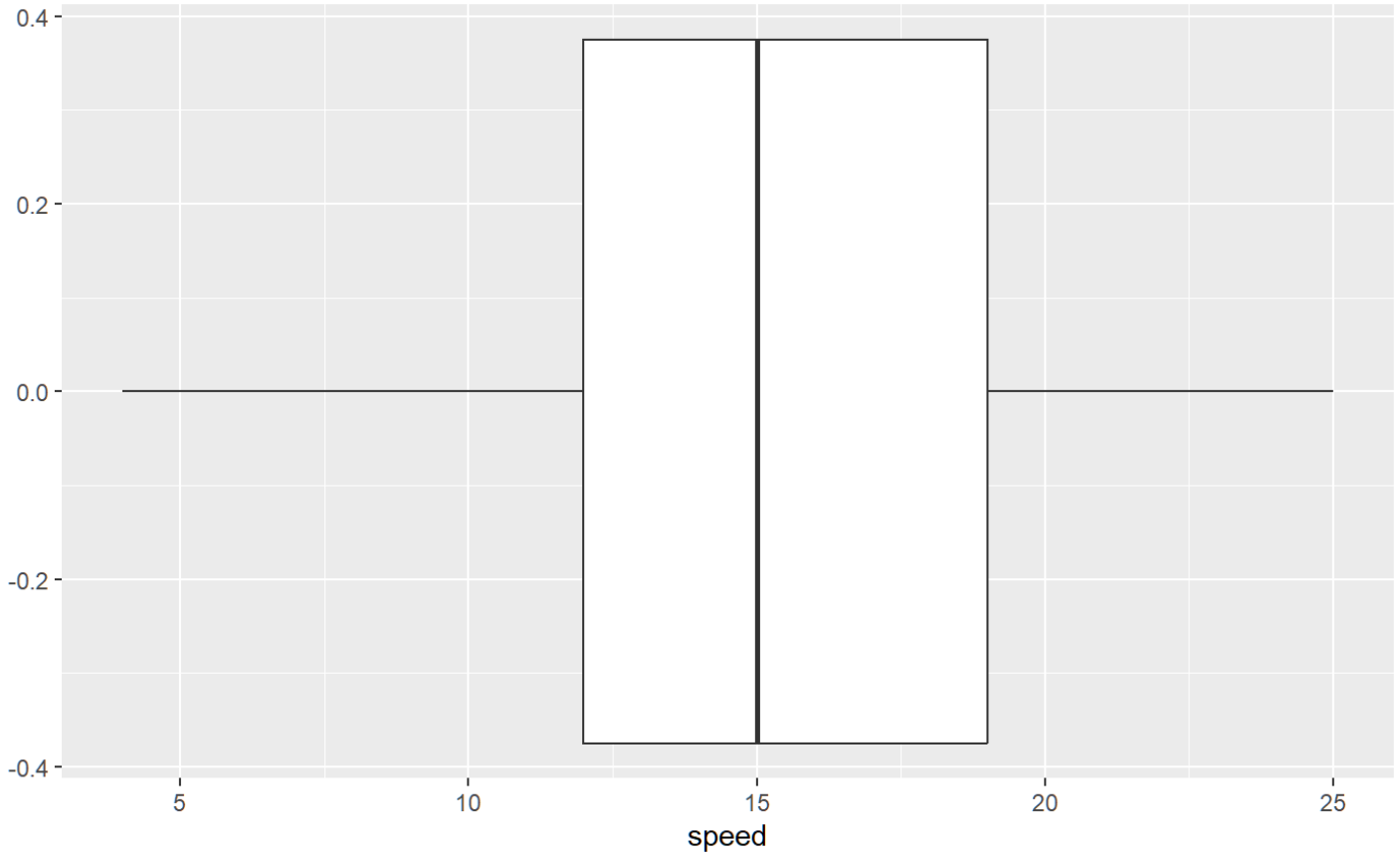
```
cor.test(state_df$Illiteracy, state_df$Income)
```

```
    Pearson's product-moment correlation

data:  state_df$Illiteracy and state_df$Income
t = -3.3668, df = 48, p-value = 0.001505
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.6378257 -0.1807128
sample estimates:
       cor
-0.4370752
```

Finally, let's check for outliers in the dataset to see if anything looks particularly off. For the `cars` dataset, create a boxplot showing the distribuition of the `speed` variable. For the `state.x77` dataset, create a boxplot showing the distribution of the `Illiteracy` dataset. Again, remember that the `state.x77` dataset first has to be put into a data frame using `as.data.frame()`. You can store this data frame version of the data in a variable if you'd like.

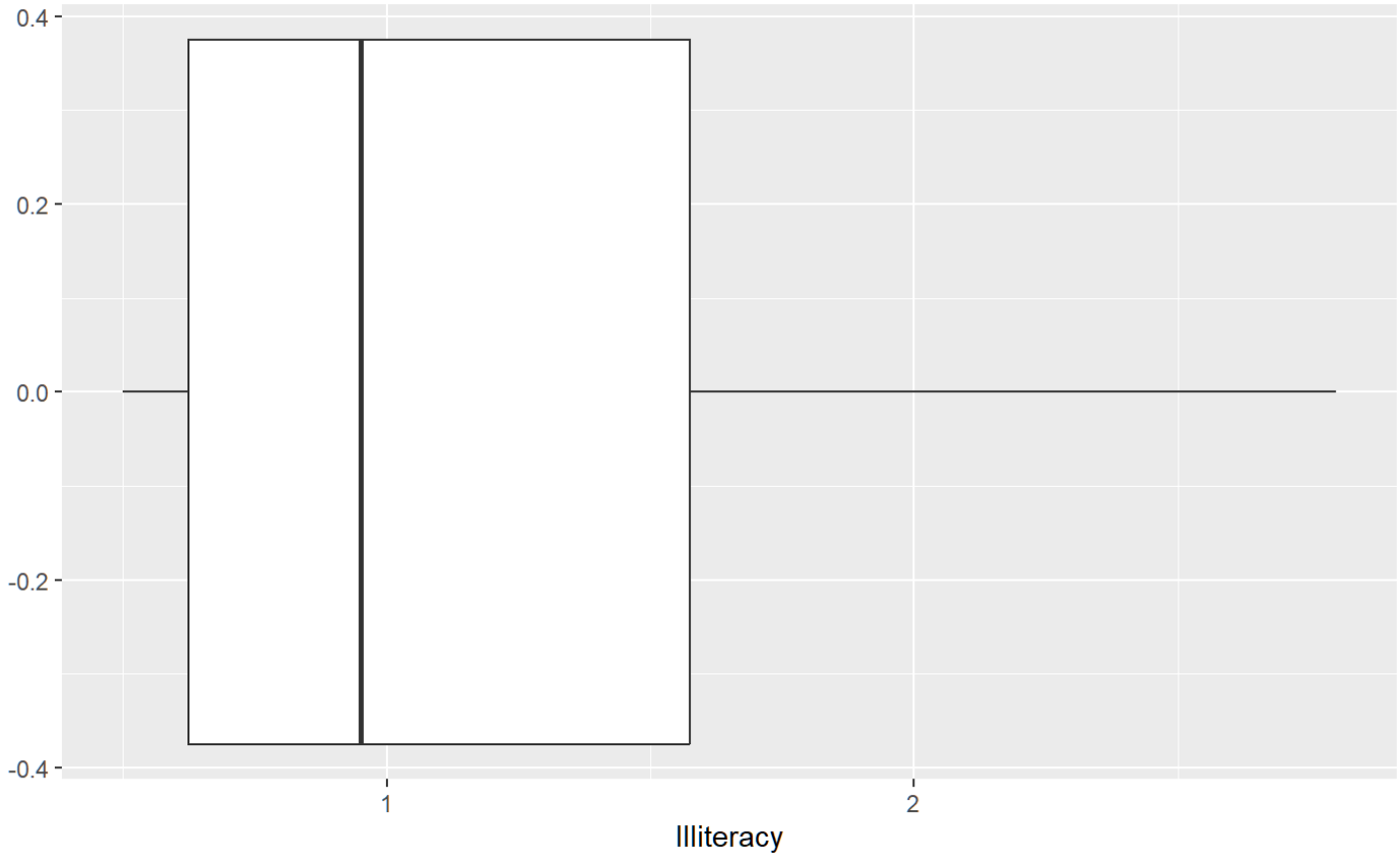
Hide

```
# Create a box plot of your datasets.
ggplot(cars, aes(speed)) +
  geom_boxplot()
```



Hide

```
ggplot(state_df, aes(Illiteracy)) +
  geom_boxplot()
```



While we're looking at the cars and states datasets, we encourage you to experiment with other datasets as well! Use this space to explore assumptions of linear regression for datasets from the `datasets` package that you're interested.

Hide

```
# Explore other datasets here
```

Making and Assessing the Model

Now that we've confirmed the assumptions necessary for linear regression, let's make the models and see how they do!

Before creating our models, let's randomly split our data into training and test sets. 60% of the data should go into the training set and 40% of the data should go into the test set. We included `set.seed(123)` at the top of our code block. This makes it so that if you re-run your code, the data will be split in the same way every time. It's still randomly split, but setting the seed allows us to reproduce the random split every time we run our code.

Hide

```
# Setting the randomizer's seed
set.seed(123)

#Split your data into training and test sets
cars_sample <- sample(c(TRUE, FALSE), nrow(cars), replace = T, prob = c(0.6,0.4))
# subset data points into train and test sets
cars_train <- cars[cars_sample, ]
cars_test <- cars[!cars_sample, ]

state_sample <- sample(c(TRUE, FALSE), nrow(state_df), replace = T, prob = c(0.6,0.4))
# subset data points into train and test sets
state_train <- state_df[state_sample, ]
state_test <- state_df[!state_sample, ]
```

We can now make our linear regression models using the trainings set! Create a linear regression model using the `cars` dataset that predicts `dist` based on `speed`. Similarly, create a model using the `state.x77` dataset that predicts `Income` based on `Illiteracy`.

Hide

```
# Create your linear regression models
car_model = lm(dist ~ speed, data=cars_train)
state_model = lm(Income ~ Illiteracy, data=state_train)
```

Let's now look at the RSE of each model using `sigma()`. What do these number mean? Remember, the units of RSE are specific to each dataset. Write your answer as a comment in your code block.

Hide

```
# Find the RSE of each model
sigma(car_model)
```

```
[1] 15.35049
```

Hide

```
sigma(state_model)
```

```
[1] 627.6051
```

This means that the true distance it takes a car to stop will be `15.35` feet off from what the model predicted, on average. Similarly, the model's prediction for average income was off by `627.61` dollars on average.

But are these numbers good or bad? It's sometimes hard to tell. Let's take a look at the R Squared value for each model. As a reminder, you can find R Squared values using this code `summary(model)$r.squared`.

Try to articulate what these values mean as a comment in your code. We'll discuss our results after the code block.

Hide

```
# Find the R Squared value of each model
summary(car_model)$r.squared
```

```
[1] 0.7110617
```

Hide

```
summary(state_model)$r.squared
```

```
[1] 0.1964859
```

The R Squared value measures the proportion of variance explained. In other words, 71% of the variance found in the car's stopping distance can be explained by the speed of the car. On the other hand, only 20% of the variance in a state's income can be explained by the state's illiteracy level.

Finally, let's look at the rest of `summary()` for each model. This time, focus on the coefficients and the `Pr(>|t|)` value of each model. Again, try writing a comment in your code articulating what these values mean. We'll discuss our results after the code block.

Hide

```
# Print the summary of each model
summary(car_model)
```

```

Call:
lm(formula = dist ~ speed, data = cars_train)

Residuals:
    Min      1Q  Median      3Q     Max
-28.619  -8.412  -2.793   8.185  42.280

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -24.888      8.851  -2.812  0.00889 **
speed          4.275      0.515   8.301 4.94e-09 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 15.35 on 28 degrees of freedom
Multiple R-squared:  0.7111,    Adjusted R-squared:  0.7007
F-statistic: 68.91 on 1 and 28 DF,  p-value: 4.944e-09
```

Hide

```
summary(state_model)
```

```
Call:
lm(formula = Income ~ Illiteracy, data = state_train)

Residuals:
   Min     1Q Median     3Q    Max
-764.4 -337.2 -163.9  288.8 2083.1

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   4964.4      251.0  19.778   <2e-16 ***
Illiteracy    -488.3      186.6  -2.617   0.0142 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 627.6 on 28 degrees of freedom
Multiple R-squared:  0.1965,    Adjusted R-squared:  0.1678
F-statistic: 6.847 on 1 and 28 DF,  p-value: 0.01415
```

Just like with the R Squared value, these values are showing that the `speed` variable in the cars dataset might be a better fit linear regression than the `Illiteracy` variable in the state dataset. The easiest way to see this is to look at the stars by each coefficient. For the car dataset, the `speed` coefficient has three stars, meaning the p-value is less than `0.001`. This means that if there was truly no relationship between `speed` and `dist`, we would expect to see data like this from about 1 out of 1000 random samples.

On the other hand, the states dataset has a slightly higher p-value for `Illiteracy`, and therefore has only one star. A p-value of `0.014` means that if there were no relationship between `Illiteracy` and `Income`, we would find data that looks like this in about 1 out of every 100 random samples. That’s still pretty good, but we’re about ten times as certain that a relationship exists between the variables in the car dataset.

Once again, we’ve added a code block below for you to do this process on another dataset that you feel might be suitable for linear regression.

Graphing the Residuals and Adding A LOESS Smoother

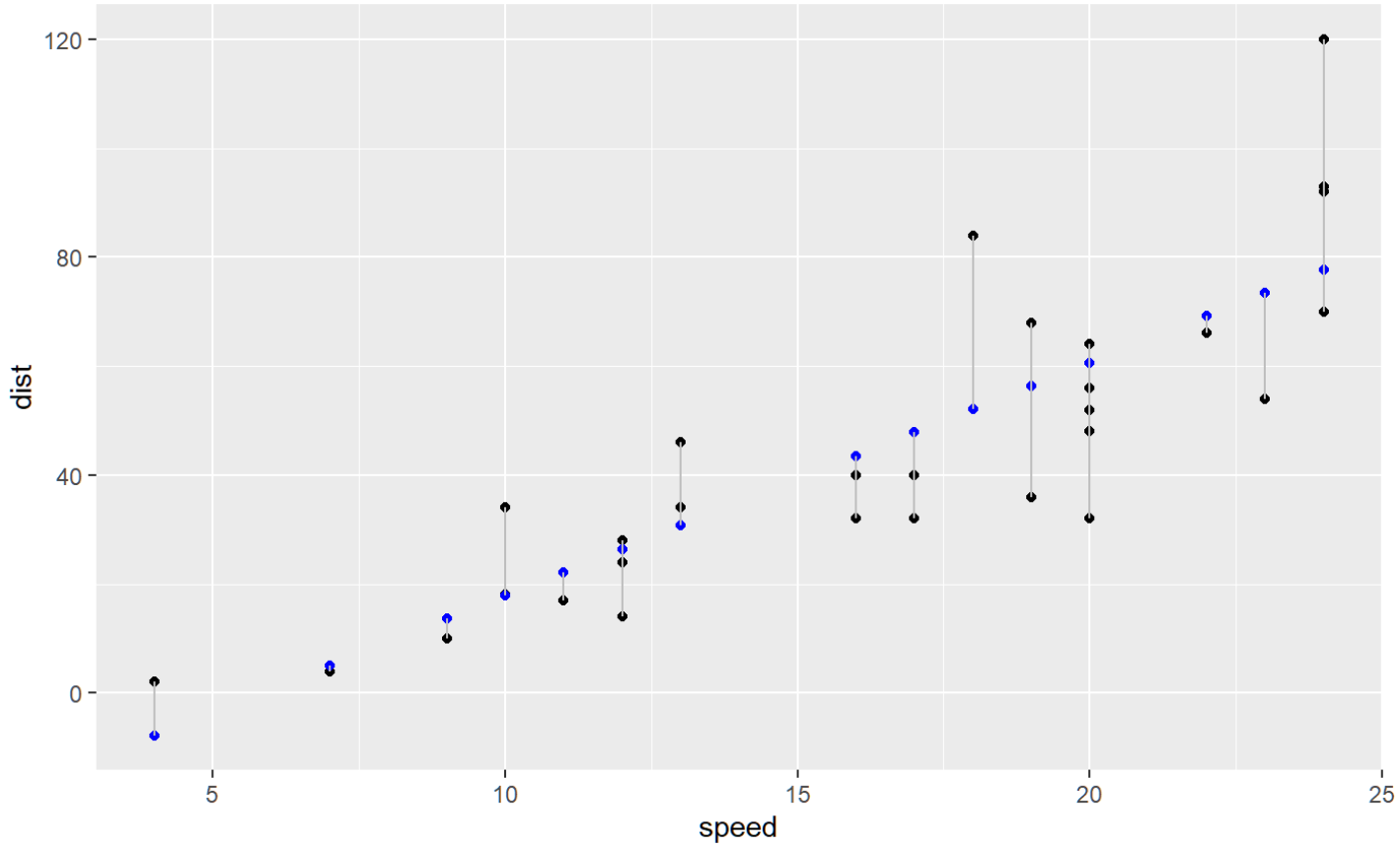
Nice work! Let’s now graph the residuals. Add a column named `estimate` to your training sets. These columns should be created by calling `predict()` using your models as a parameter. Also add a column named `residuals` to the training set by calling `residuals()` using your models as a parameter.

Once those columns have been created, create a graph that shows both the estimated values and the residuals. Your x and y axes should be the same features, however instead of plotting the original data, plot the estimated values and the residuals.

We also included segments to connect each residual to their corresponding estimate.

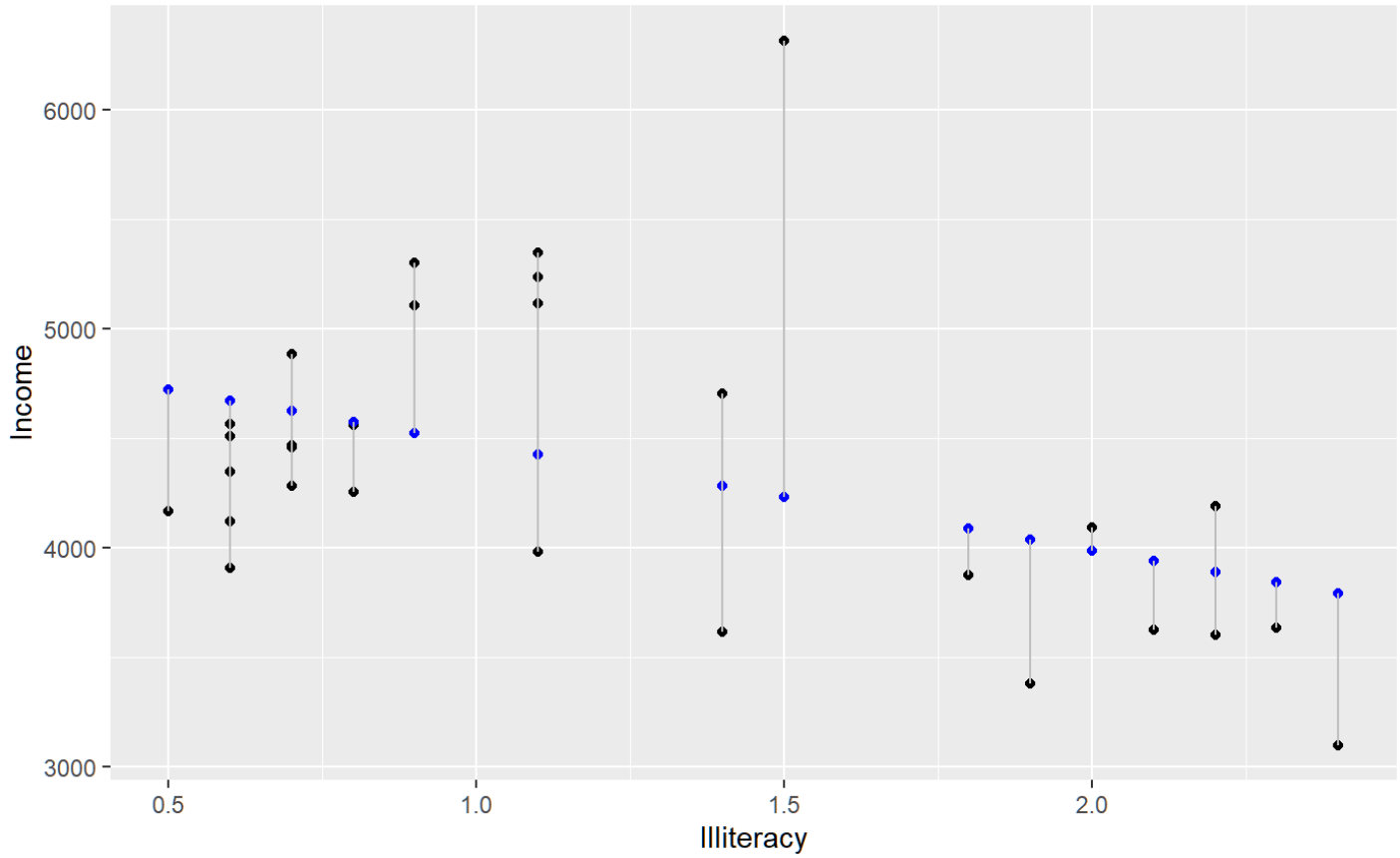
Hide

```
#save predicted and residual values to df
cars_train$estimate = predict(car_model)
cars_train$residuals = residuals(car_model)

#create visualization
ggplot(cars_train, aes(speed, dist)) +
geom_point() + #plot actual values
geom_point(aes(y = estimate), color = "blue") +
geom_segment(aes(xend = speed, yend = estimate), color = "gray")
```



Hide

```
#save predicted and residual values to df
state_train$estimate = predict(state_model)
state_train$residuals = residuals(state_model)

#create visualization
ggplot(state_train, aes(Illiteracy, Income)) +
geom_point() + #plot actual values
geom_point(aes(y = estimate), color = "blue") +
geom_segment(aes(xend = Illiteracy, yend = estimate), color = "gray")
```



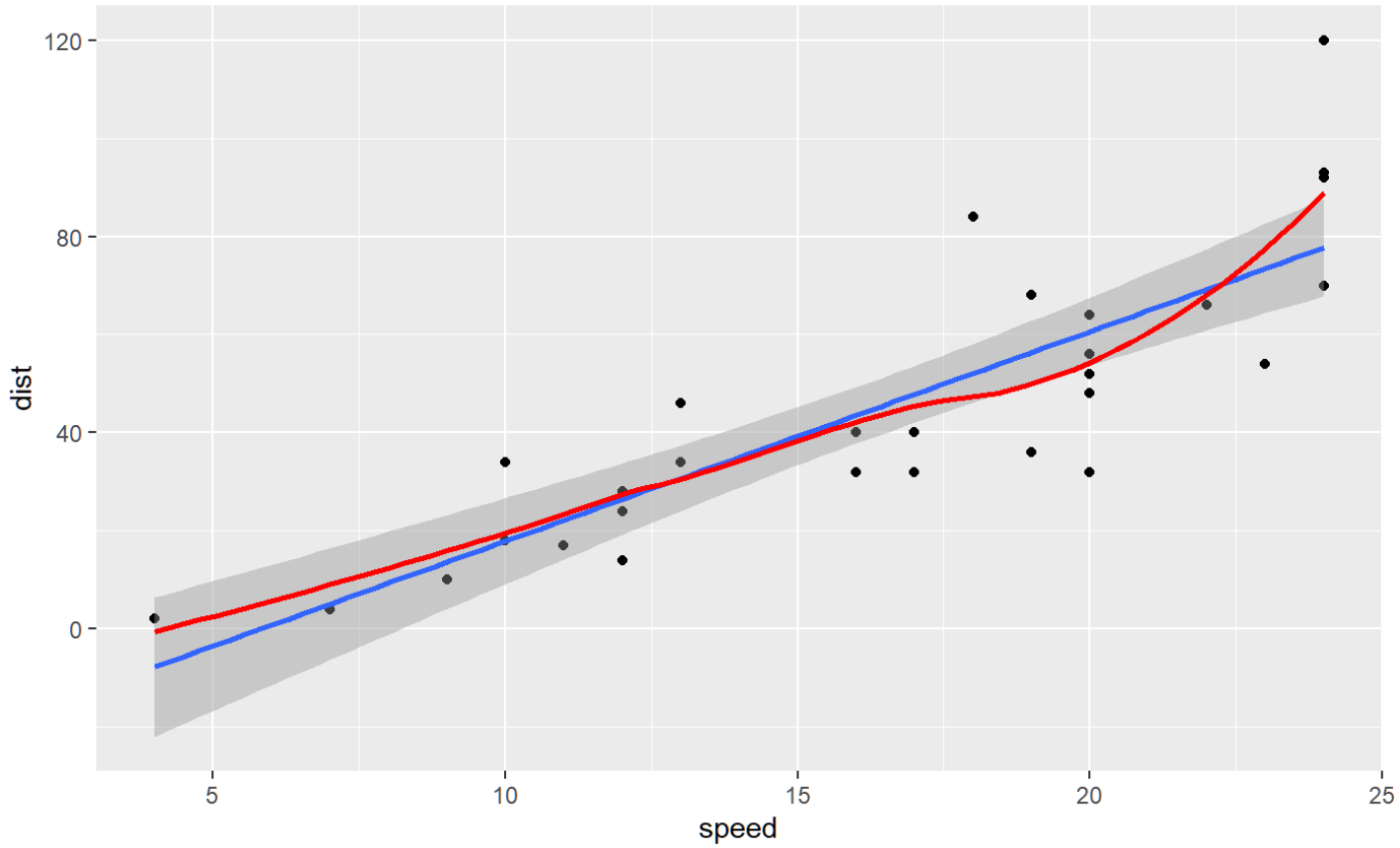
Let's also add a visualization of the model and LOESS smoother to our scatter point. Grab your code for your original scatterplot, but make sure to use your training set rather then the original dataset. Then add `geom_smooth(method = "lm")` to visualize the model and `geom_smooth(se = FALSE, color = "red")` to add a LOESS smoother.

Do these visualizations confirm what you saw when calculating the R Squared value of each model?

Hide

```
# Add a LOESS smoother to your scatterplot

ggplot(cars_train, aes(speed, dist)) +
geom_point() +
geom_smooth(method = "lm") +
geom_smooth(se = FALSE, color = "red")
```

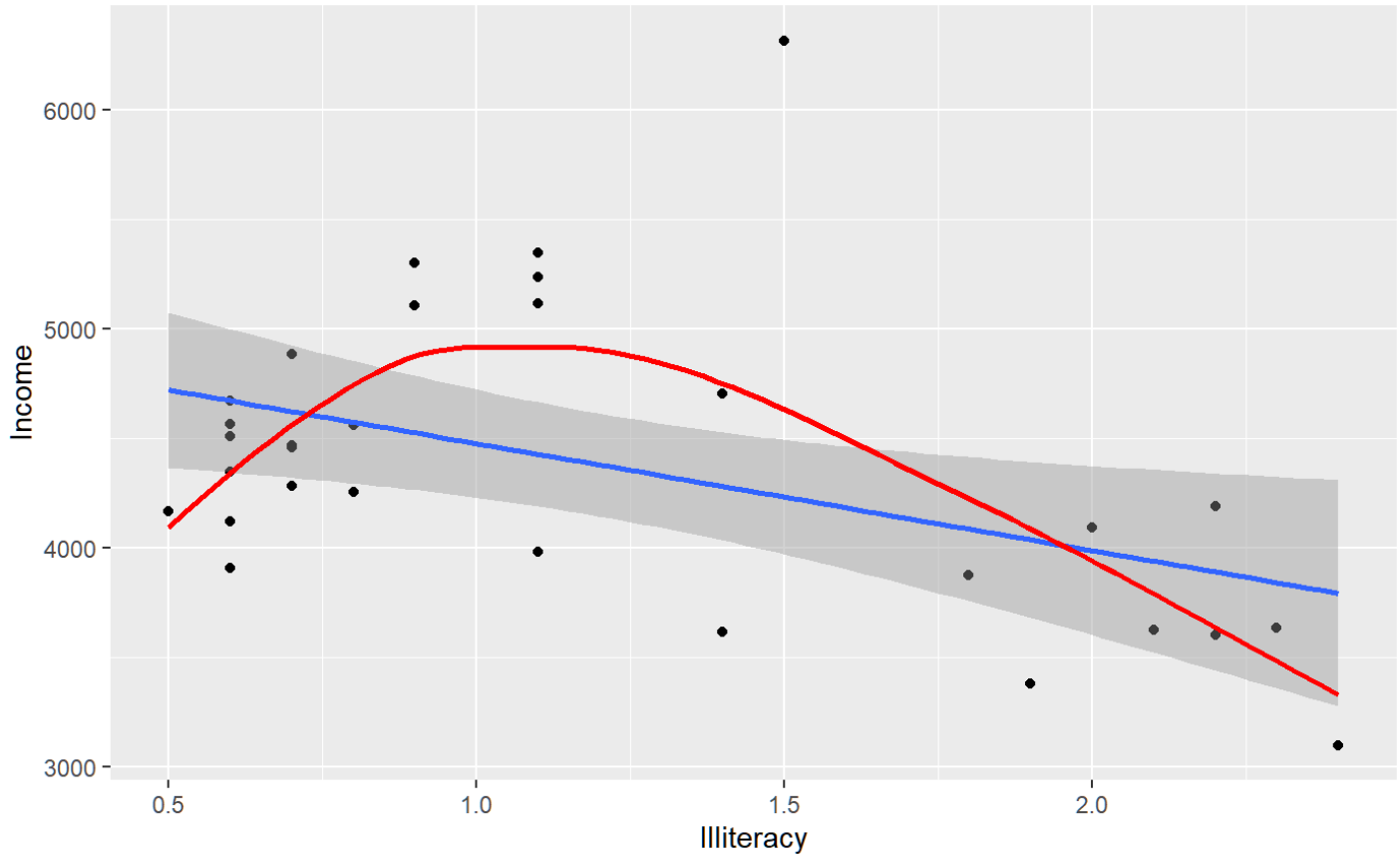
```
`geom_smooth()` using formula = 'y ~ x'
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Hide

```
ggplot(state_train, aes(Illiteracy, Income)) +
geom_point() +
geom_smooth(method = "lm") +
geom_smooth(se = FALSE, color = "red")
```

```
`geom_smooth()` using formula = 'y ~ x'
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Here's a code block for you to try these visualizations on different datasets.

Upgrading To A Multiple Linear Regression Model

Everything that we've done so far seems to indicate that our model for the states dataset is good, but could be better. This is the perfect opportunity to try to upgrade it to a multiple linear regression model. Take a look at the head of that dataset again. Do you think there's another column that could help explain a state's `Income` ? Create a new model using at least two features as predictor variables. Print out the `summary()` of that model. How does it compare to your first model?

Again, try to write a few sentences as a comment describing the summary of your new model. After the code block we'll walk you through our process.

Hide

```
# Create a multiple linear regression model
head(state_train)
```

	Population <dbl>	Inco… <dbl>	Illiteracy <dbl>	Life Exp <dbl>	Mur… <dbl>	HS Grad <dbl>	Frost <dbl>	Area <dbl>	estimate <dbl>
Alabama	3615	3624	2.1	69.05	15.1	41.3	20	50708	3938.879
Alaska	365	6315	1.5	69.31	11.3	66.7	152	566432	4231.875
Arkansas	2110	3378	1.9	70.66	10.1	39.9	65	51945	4036.544
California	21198	5114	1.1	71.71	10.3	62.6	20	156361	4427.205

	Population	Inco…	Illiteracy	Life Exp	Mur…	HS Grad	Frost	Area	estimate
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
Colorado	2541	4884	0.7	72.06	6.8	63.9	166	103766	4622.535
Connecticut	3100	5348	1.1	72.48	3.1	56.0	139	4862	4427.205

Hide

```
state_model_multiple <- lm(Income ~ `HS Grad` + Population, data = state_train)
summary(state_model_multiple)
```

```
Call:
lm(formula = Income ~ `HS Grad` + Population, data = state_train)

Residuals:
    Min      1Q  Median      3Q     Max 
-814.15 -256.00  -96.93  190.71 1289.45 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept) 1.280e+03  6.052e+02   2.116   0.0438 *  
`HS Grad`   5.592e+01  1.128e+01   4.959  3.4e-05 ***
Population  4.197e-02  2.051e-02   2.046   0.0506 .  
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 498.8 on 27 degrees of freedom
Multiple R-squared:  0.5106,    Adjusted R-squared:  0.4743 
F-statistic: 14.08 on 2 and 27 DF,  p-value: 6.47e-05
```

We first created a model using `Illiteracy` and `HS Grad` as predictor variables. Note that because of the space in name `HS Grad`, make sure to put it in backticks in your code. The R Squared value for that model was `0.4352` — a notable improvement from the first model.

However, in looking at the p-value for the coefficients, it appears that `Illiteracy` is no longer statistically significant. Remember, we want a p-value of `0.05` or less!

We then confirmed that intuition by removing `Illiteracy` from the model entirely. The R Squared value barely dropped — `Illiteracy` was barely contributing to the model.

Finally, we decided to see if the population of a state contributed to that state's average income. We added `Population` to the linear regression model and found that coefficient had a smaller p-value of `0.0506`. That is incredibly close to being statistically significant!

Review

To recap, there were two main goals of this project. The first was to explore the datasets that come built-in to R. We specifically used `cars` and `state.x77`, but we hope you explored other datasets on your own. See if you can reproduce this entire linear regression pipeline on your own with a different dataset.

The second goal of this project was to illustrate how R streamlines the linear regression process with almost every dataset. The code hardly changes! Because of this, we can't stress enough the importance of understanding and explaining your model. As you become more familiar with R, the code becomes the easy part! Articulating your findings becomes the real challenge.

To take your linear regression skills to the next level, try finding a dataset online, making a linear regression model, and interpreting your results. There are tons of online resources, be we've found the UCI Machine Learning Repository (http://archive.ics.uci.edu/ml/datasets.php) to be a great starting place.