Simple Linear Regression in R

Hannah Waddel

2/10/2023

Simple Linear Regression in R: Relationship between age and weight in chicks

Our strategy for a simple linear regression in R is as follows:

1. Read in the data.
2. Perform descriptive statistics to explore the data.
3. Fit the linear model.
4. Report interpretations and conclusions.

Creating the dataset

We first must create the dataset in R. We will create a data frame called chick_weight with the chick data.

To create the age data, since the chicks were measured at days 6 to 16, we use the function **seq()** which will create a sequence of numbers between the "from=" argument and the "to=" argument.

```
seq(from=6,to=16)
```

```
##  [1]  6  7  8  9 10 11 12 13 14 15 16
```

To create a data frame, we use the data.frame function and give it each data column as a list. We have two columns, age and weight. When you create a data frame, give the names of the columns as arguments in the function, and assign the columns with the equal sign = .

```
chick_weight <- data.frame("age"=seq(from=6,to=16),
                           "wgt"=c(0.029,0.052,0.079,0.125,0.181,0.261,0.425,0.738,1.13,1.882,2.812))

print(chick_weight)
```

```
##    age   wgt
## 1    6 0.029
## 2    7 0.052
## 3    8 0.079
## 4    9 0.125
## 5   10 0.181
## 6   11 0.261
## 7   12 0.425
## 8   13 0.738
## 9   14 1.130
## 10  15 1.882
## 11  16 2.812
```

Exploring the Data

Now that we have created the data, let's explore the data with some descriptive statistics. We'll use the **mean()**, **sd()**, **median()**, **IQR()**, and **range()** functions. Recall that we access each column/variable of the data frame chick_weight with the **$** operator.

```
mean(chick_weight$age)
```

```
## [1] 11
```

```
sd(chick_weight$age)
```

```
## [1] 3.316625
```

```
median(chick_weight$age)
```

```
## [1] 11
```

```
IQR(chick_weight$age)
```

```
## [1] 5
```

```
range(chick_weight$age)
```

```
## [1]  6 16
```

```
mean(chick_weight$wgt)
```

```
## [1] 0.7012727
```

```
sd(chick_weight$wgt)
```

```
## [1] 0.9037761
```

```
median(chick_weight$wgt)
```

```
## [1] 0.261
```

```
IQR(chick_weight$wgt)
```

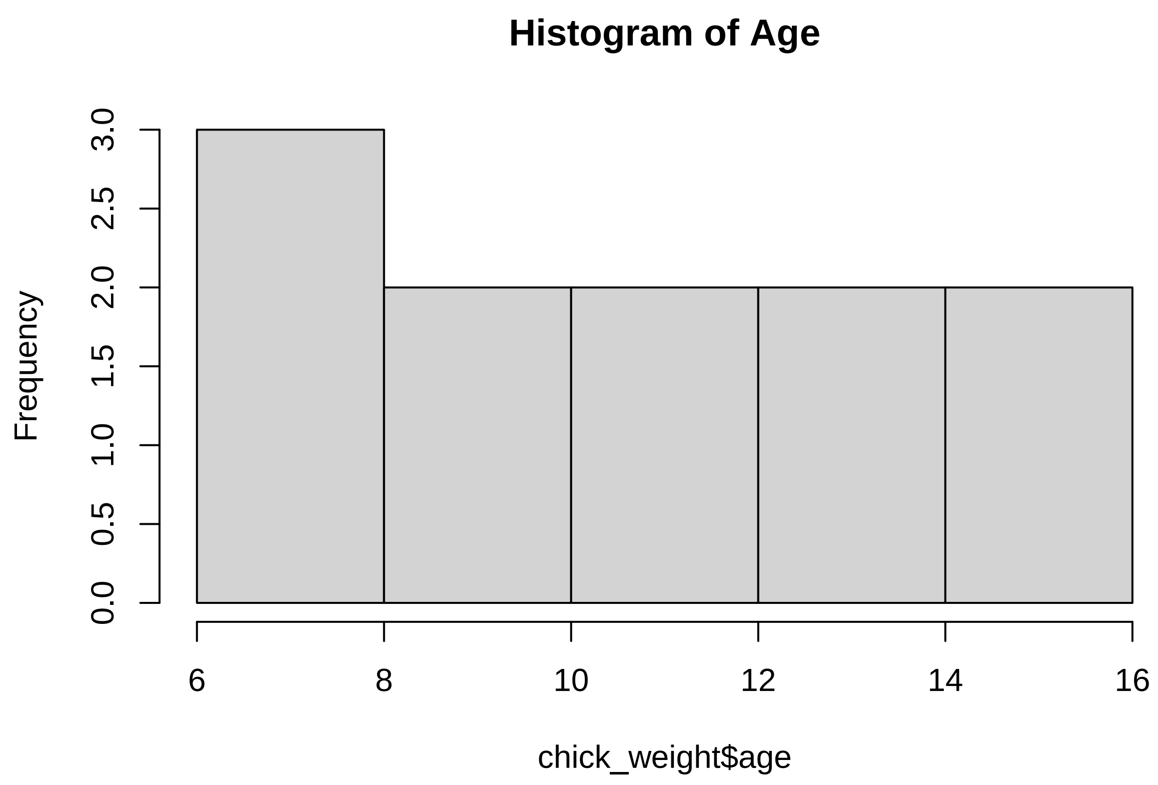
```
## [1] 0.832
```

```
range(chick_weight$wgt)
```

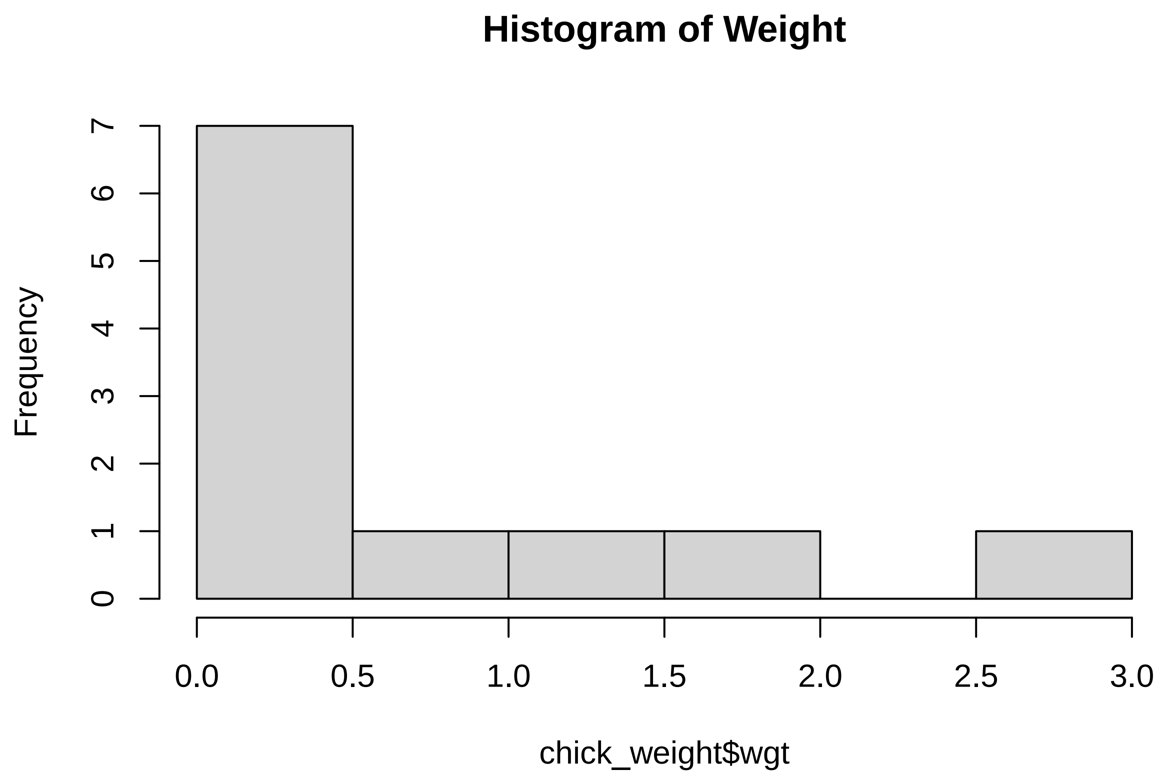
```
## [1] 0.029 2.812
```

We will also plot histograms of the data to examine the distributions of age and weight. We use the **hist()** function, and we now set the titles of the histogram plots with the **main=** argument, which takes a string in quotation marks and sets it as the title of the histogram.

```r
hist(chick_weight$age,main="Histogram of Age")
```

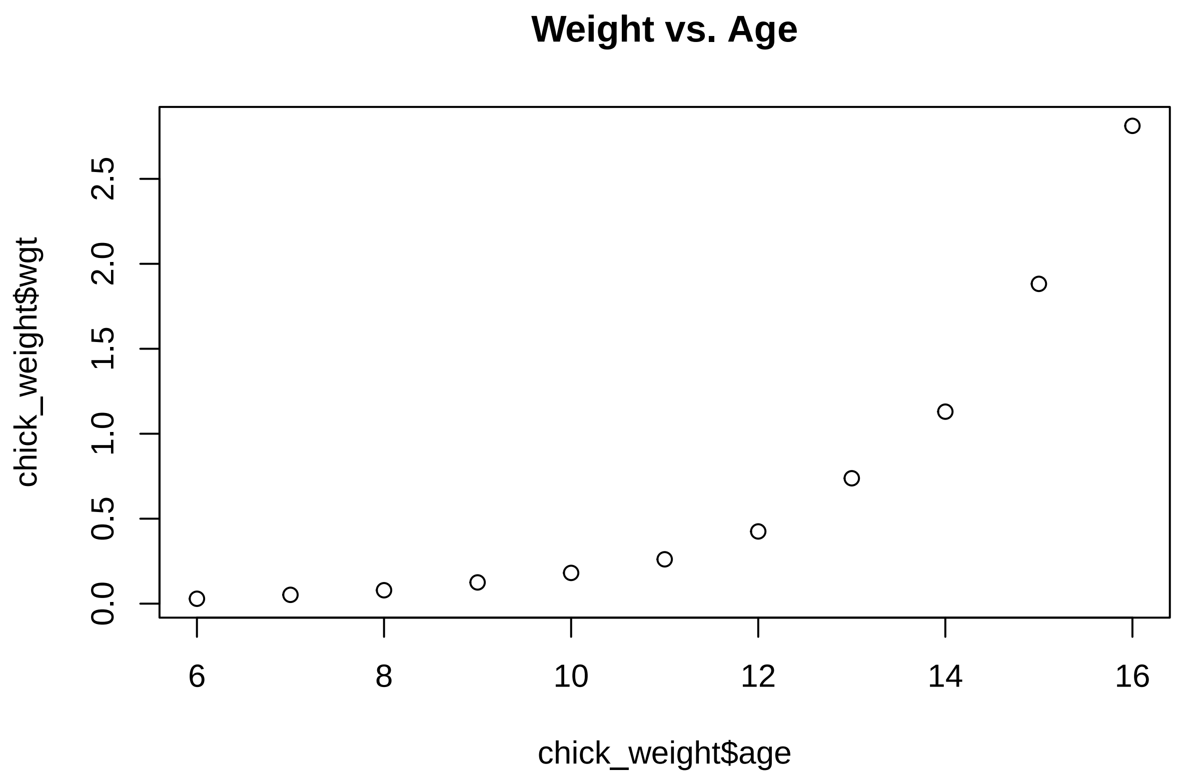


```r
hist(chick_weight$wgt,main="Histogram of Weight")
```



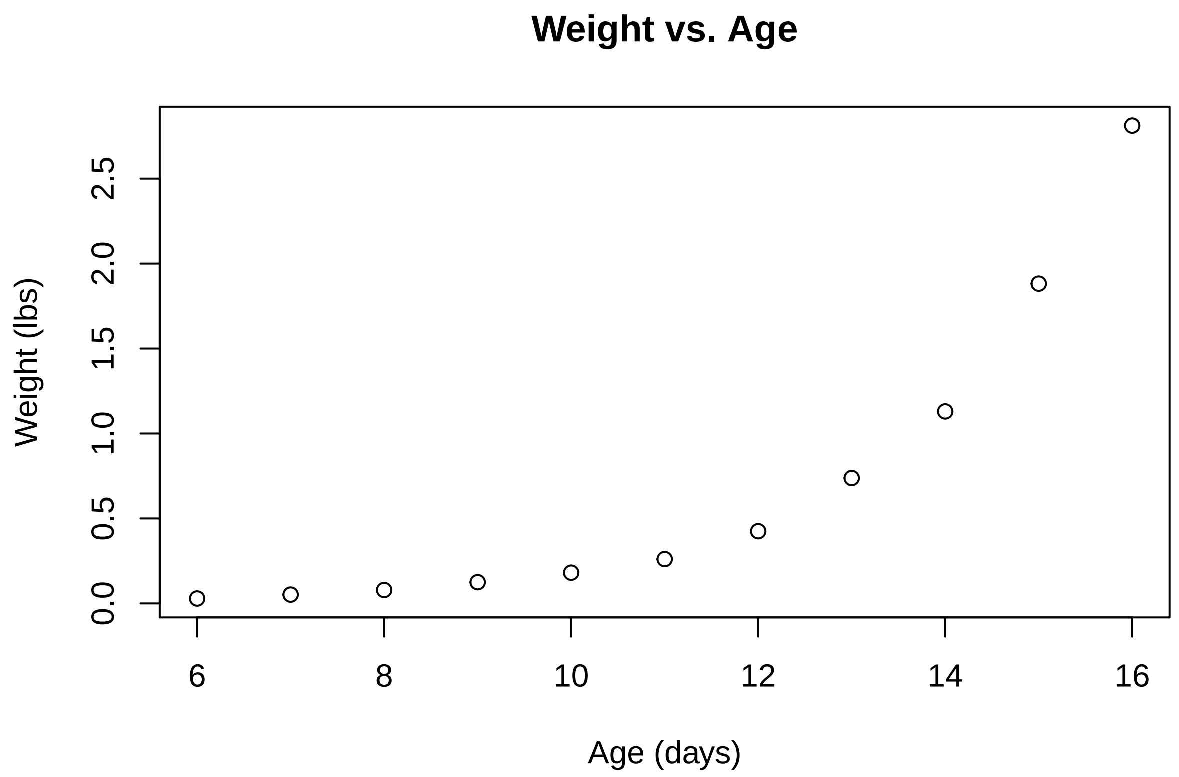
We also plot a simple scatterplot of the weight versus the age of the chick, with the **plot()** function. We can use the "main=" argument to set the plot title again.

```
plot(chick_weight$age,chick_weight$wgt,main="Weight vs. Age")
```



The plot labels on the x and y axis don't look so great We can use the "xlab=" and "ylab=" arguments to give them new labels.

```
plot(chick_weight$age,chick_weight$wgt,main="Weight vs. Age",xlab="Age (days)",ylab="Weight (lbs)")
```



There are many more options for customizing your plots, and we'll discuss more as we go, but this looks good for now.

Now, we want to calculate R and R^2 between age and weight. We can do this with the **cor()** function, which takes the age and weight columns as arguments.

```
cor(chick_weight$age,chick_weight$wgt)
```

```
## [1] 0.8626562
```

```
cor(chick_weight$age,chick_weight$wgt)^2
```

```
## [1] 0.7441757
```

The correlation, R, between age and weight is 0.86. R^2 is 0.74, meaning that 74% of the variation in age is explained by the variation in weight.

In order to get the confidence interval for the correlation coefficient, as well as test its significance, we use the function **cor.test()**.

```
cor.test(chick_weight$age,chick_weight$wgt)
```

```
##
##  Pearson's product-moment correlation
##
```

```
## data:  chick_weight$age and chick_weight$wgt
## t = 5.1167, df = 9, p-value = 0.0006308
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.5446083 0.9637857
## sample estimates:
##       cor
## 0.8626562
```

The confidence interval for the correlation coefficient, R, is (0.54,0.96).

Fit Linear Model

Now we will fit the simple linear regression model between age and weight. To do this, we will use the **lm()** (linear model) function. This is a powerful function, and we will use it many times during the rest of the course.

For a simple linear regression, the **lm()** function takes a formula as its argument, created using the ~ operator. We will put the dependent variable (weight) on the left side of the ~, and the independent variable (age) on the right side. The other argument the **lm()** function takes is "data=", and here we give it chick_weight.

```
chick.lm <- lm(wgt ~ age,data=chick_weight)
```

The **lm()** function creates a linear model object that has many different parts. Luckily, there are some functions we can use which will help us summarize it. One easy function is the **summary()** function. We will use this to get an overall summary of the linear model. If you fit other types of models in R, after this class (non-linear models, time series models, etc), you can typically use **summary()** on those fit models as well.

```
summary(chick.lm)
```

```
##
## Call:
## lm(formula = wgt ~ age, data = chick_weight)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.5113 -0.3593 -0.1061  0.2657  0.9354
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.88453    0.52584  -3.584 0.005895 **
## age          0.23507    0.04594   5.117 0.000631 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4818 on 9 degrees of freedom
## Multiple R-squared:  0.7442, Adjusted R-squared:  0.7158
## F-statistic: 26.18 on 1 and 9 DF,  p-value: 0.0006308
```

The summary includes most of the information we are interested in. It first gives the formula for the linear model, then summarizes the residuals (we want to see a median residual of 0). Next, it gives the summary of the fit coefficients. β_0 is the coefficient for the intercept and β_1 is the coefficient associated with age.

It gives the fitted value (estimate), standard error, T statistic, and P-value for each coefficient.

It also gives the "residual standard error", which is *not* the MSE. It also gives the R^2 value, which we calculated earlier. The "adjusted R-squared" statistic is more useful later, for multiple linear regression.

The F-statistic and p-value test whether β_1 is not equal to 0.

To get the ANOVA table for the linear model, we use the **anova()** function.

```
anova(chick.lm)
```

```
## Analysis of Variance Table
##
## Response: wgt
##           Df Sum Sq Mean Sq F value    Pr(>F)
## age        1 6.0785  6.0785   26.18 0.0006308 ***
## Residuals  9 2.0896  0.2322
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

If we are interested in seeing confidence intervals for the estimated coefficients, we can use the **confint()** function.

```
# Let's see the confidence intervals for estimated coefficients
confint(chick.lm)
```

```
##                  2.5 %     97.5 %
## (Intercept) -3.0740495 -0.6950050
## age          0.1311437  0.3390018
```

The confidence interval for β_1 is (0.13, 0.33).

Predicting Outcomes

Now we want to predict the age at the median age of 11 days. R has a built in **predict()** function, but we have to make sure to give it the correct arguments.

The **predict()** function takes in a fitted linear model from the **lm()** function, and a data frame of the values for which we want to predict the outcome. The data frame is given for the argument "newdata=".

We create a data frame, "median.chick", that has one column named age, and contains one value of age: 11. Then, we give the **predict()** function the "chick.lm" object we created earlier, and the data frame "median.chick".

```
# Predict weight at 11 days
median.chick <- data.frame(age=11)
predict(chick.lm,newdata=median.chick)
```

```
##         1
## 0.7012727
```

We predict that the average weight for chicks at 11 days is 0.701 pounds. The "1" in this data output is an index–it is saying that the first value in the list is 0.701.

If we want to calculate a confidence interval at 11 days, we can specify this in the **predict()** function with an additional argument, "interval=".

```
predict(chick.lm,newdata=median.chick,interval="confidence")
```

```
##         fit       lwr      upr
## 1 0.7012727 0.3726202 1.029925
```

We are 95% confident that the average weight of chicks at 11 days is between 0.37 and 1.03.

The default confidence interval is 95%, but we can specify the confidence level with the "level=" argument.

```
predict(chick.lm,newdata=median.chick,interval="confidence",level=0.9)
```

```
##         fit      lwr       upr
## 1 0.7012727 0.434953 0.9675925
```

We are 90% confident that the average weight of chicks at 11 days is between 0.43 and 0.97.

Plotting Results

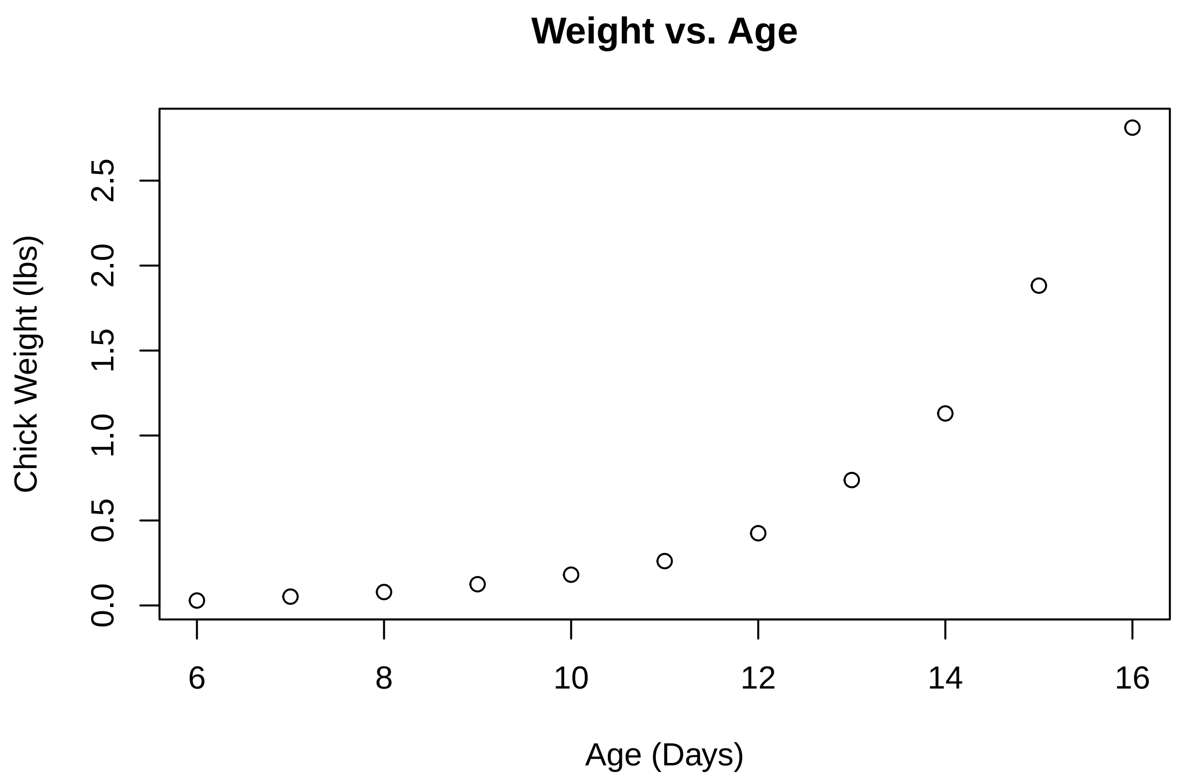
Now, we want to plot our data, the predicted line, the confidence intervals, and prediction intervals.

We first create a sequence of the age values we are interested in, and save it as a data frame. Then, we use it as an argument in the **predict()** function, and save the predicted values at each age. We also save the confidence intervals and prediction intervals.

```
age.vals <- data.frame(age=seq(from=6,to=16))
age.predict <- predict(chick.lm,newdata=age.vals)
weight.ci <- predict(chick.lm,newdata=age.vals,interval="confidence")
weight.pi <- predict(chick.lm,newdata=age.vals,interval="predict")
```

We initialize a plot of our data with the **plot()** function, making sure to set the title and axis labels.

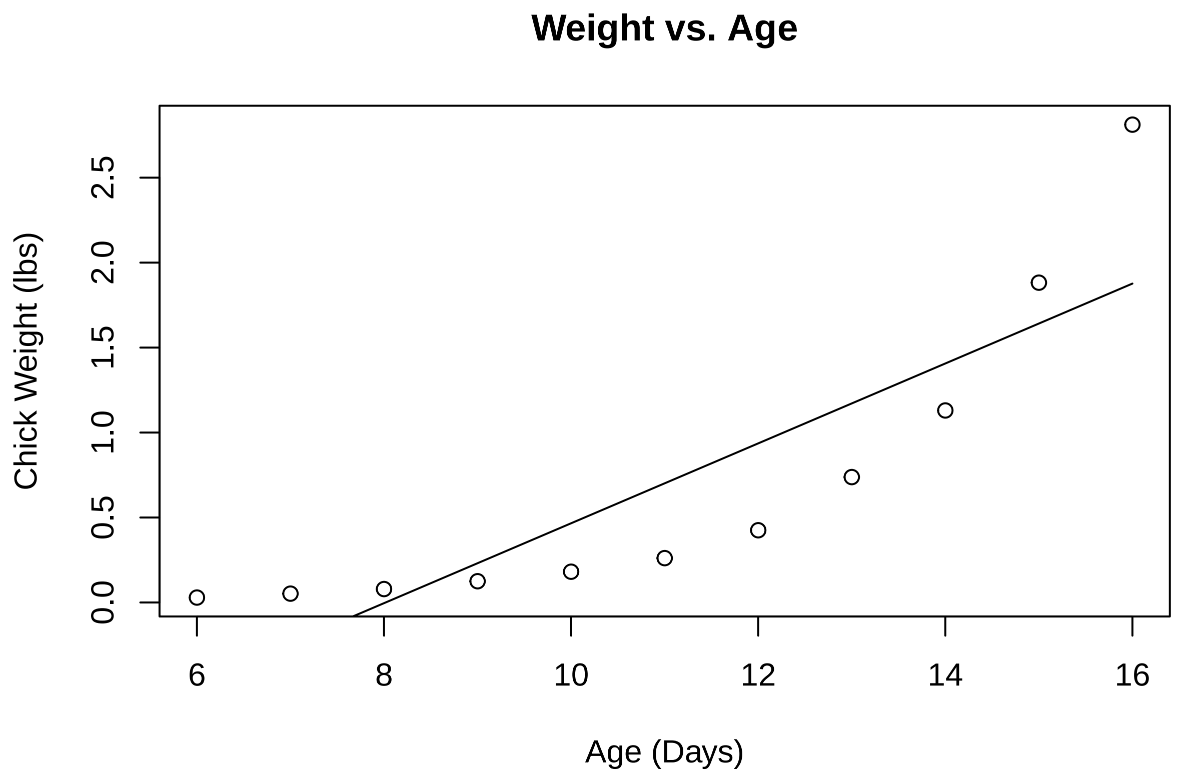
```
plot(chick_weight$age,chick_weight$wgt,main="Weight vs. Age",
     xlab="Age (Days)",ylab="Chick Weight (lbs)")
```



We plot the predicted values using the **lines()** function, which adds a line through the x and y coordinates given to the function. Note that the **lines()** function can't create a plot on its own, it must be added to an existing plot.

```
plot(chick_weight$age,chick_weight$wgt,main="Weight vs. Age",
     xlab="Age (Days)",ylab="Chick Weight (lbs)")

lines(age.vals$age,age.predict)
```



Now, we want to add lines for the confidence intervals onto the plot. Our weight.ci matrix has 3 columns: the fit (predicted) value, the lower limit of the confidence interval, and the upper limit of the confidence interval. Make sure that we get the lower and upper limits of the confidence interval. The **$** operator will not work here, because the object is a matrix, not a data frame. We use the square brackets [,] to select the second and third columns to plot.

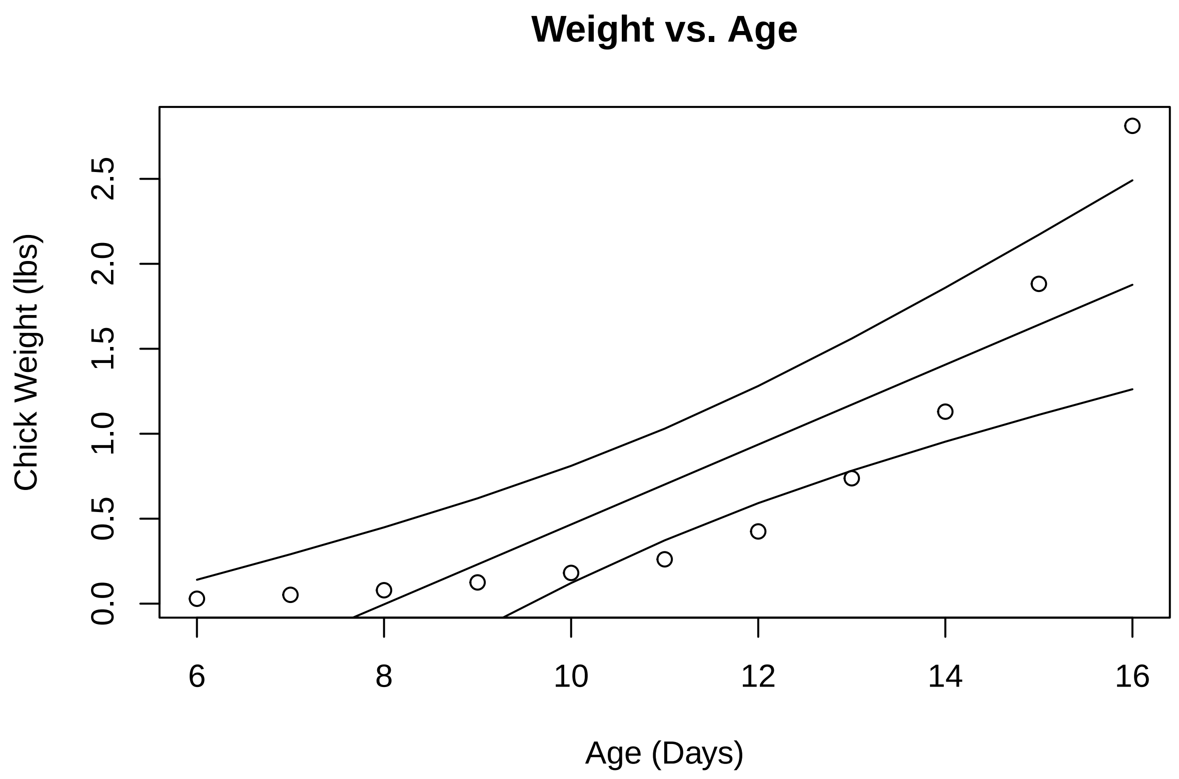
```
weight.ci
```

```
##             fit        lwr       upr
## 1  -0.474090909 -1.0889434 0.1407616
## 2  -0.239018182 -0.7689544 0.2909180
## 3  -0.003945455 -0.4569616 0.4490707
## 4   0.231127273 -0.1577396 0.6199941
## 5   0.466200000  0.1215064 0.8108936
## 6   0.701272727  0.3726202 1.0299252
## 7   0.936345455  0.5916518 1.2810391
## 8   1.171418182  0.7825513 1.5602850
## 9   1.406490909  0.9534747 1.8595071
## 10  1.641563636  1.1116274 2.1714998
## 11  1.876636364  1.2617839 2.4914889
```

```
plot(chick_weight$age,chick_weight$wgt,main="Weight vs. Age",
     xlab="Age (Days)",ylab="Chick Weight (lbs)")

lines(age.vals$age,age.predict)

lines(age.vals$age,weight.ci[,2])
lines(age.vals$age,weight.ci[,3])
```



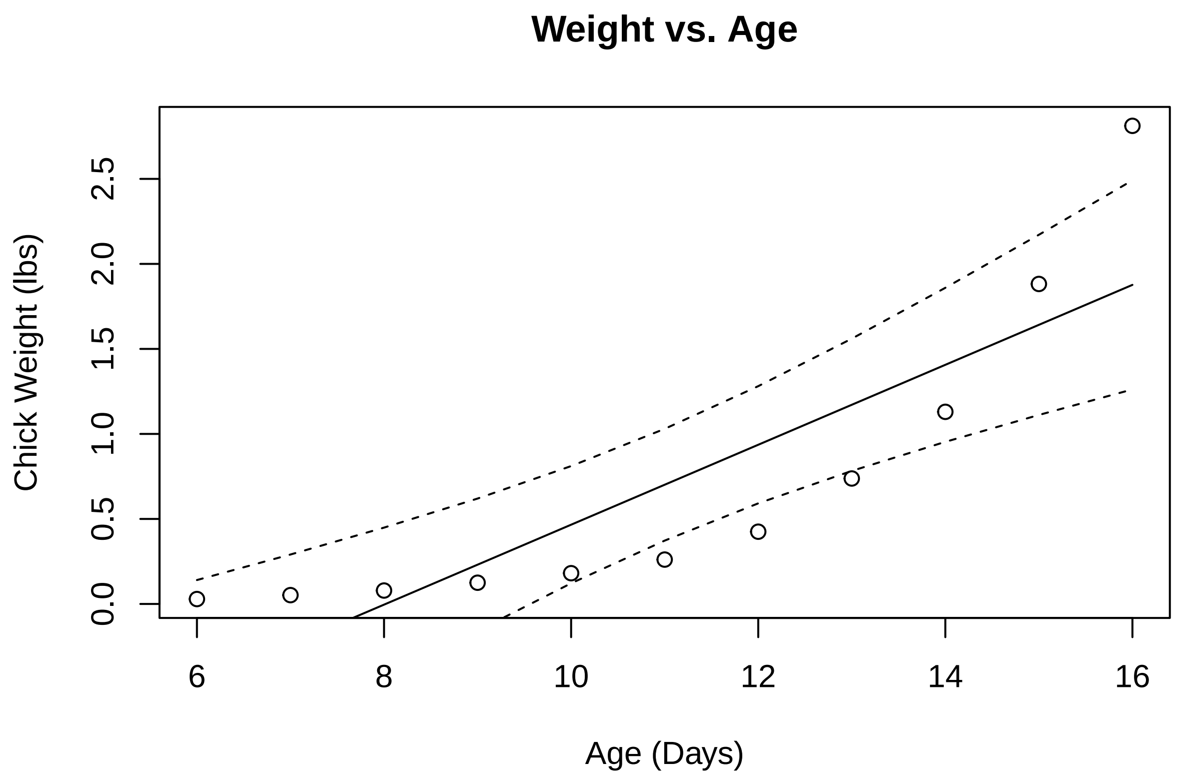
Weight vs. Age
Chick Weight (lbs)
0.0
0.5
1.0
1.5
2.0
2.5
6
8
10
12
14
16
Age (Days)

That is a lot of lines, and is a bit busy. If we give the **lines()** function the “lty=” argument (line type), and set “lty=2”, we can get a dashed line.

```
# Plot prediction interval lines
plot(chick_weight$age,chick_weight$wgt,main="Weight vs. Age",
     xlab="Age (Days)",ylab="Chick Weight (lbs)")

lines(age.vals$age,age.predict)

lines(age.vals$age,weight.ci[,2],lty=2)
lines(age.vals$age,weight.ci[,3],lty=2)
```

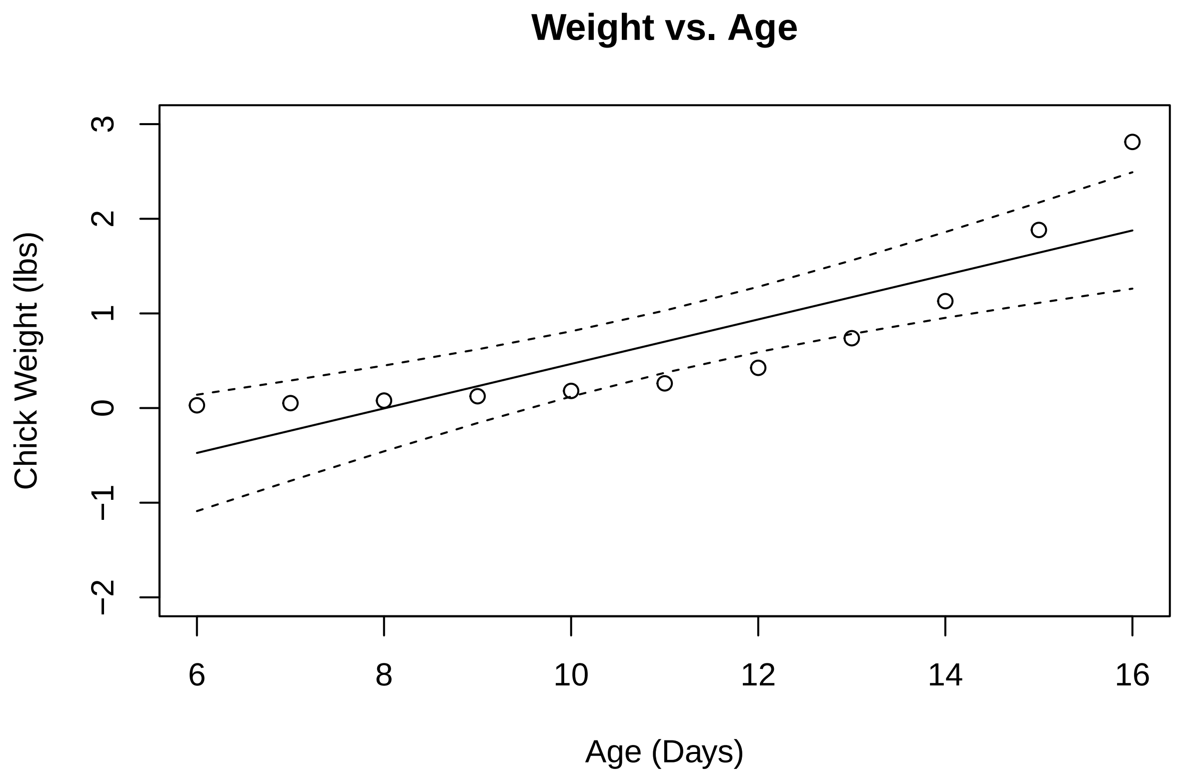


The lines look better now, but they leave the plot area. Let's fix the range of the y axis in our plot. We do this with the "ylim=" argument. This argument takes in a list of two numbers, created with **c()**. The first number is the minimum y value and the second number is the maximum y value. The argument "xlim=" works the same way.

```
plot(wgt~age,data=chick_weight,main="Weight vs. Age",
     xlab="Age (Days)",ylab="Chick Weight (lbs)",
     ylim=c(-2,3))

lines(age.vals$age,age.predict)

lines(age.vals$age,weight.ci[,2],lty=2)
lines(age.vals$age,weight.ci[,3],lty=2)
```



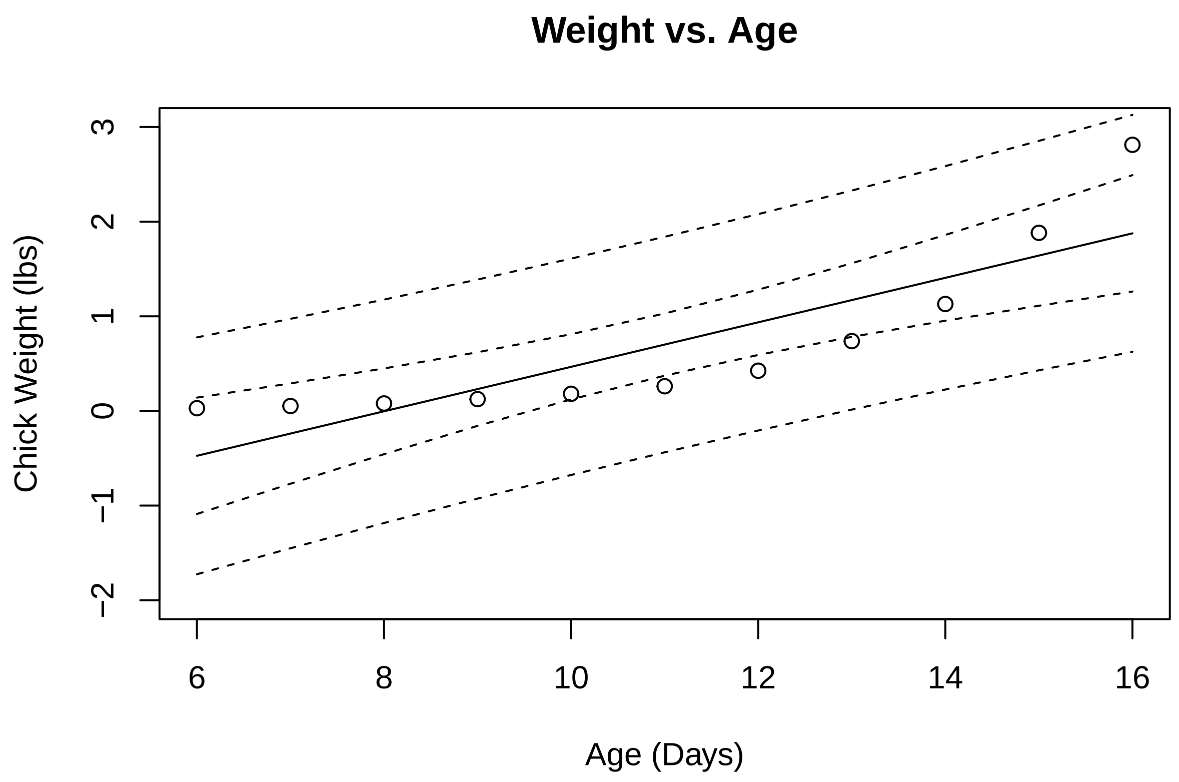
This looks better. Now, we will add dashed lines for the prediction intervals in the same way that we added the confidence intervals.

```
plot(wgt~age,data=chick_weight,main="Weight vs. Age",
     xlab="Age (Days)",ylab="Chick Weight (lbs)",
     ylim=c(-2,3))

lines(age.vals$age,age.predict)

lines(age.vals$age,weight.ci[,2],lty=2)
lines(age.vals$age,weight.ci[,3],lty=2)

lines(age.vals$age,weight.pi[,2],lty=2)
lines(age.vals$age,weight.pi[,3],lty=2)
```



There are a few more things we can do to make this plot look better. To make the data points easier to see, we can use the "pch=" argument (point character) in the **plot()** function to change the symbol for the data points. There are a lot of different pch values that you can select to distinguish your data points. I often like to use "pch=19", which is a filled dot. Here are the "pch=" options you can use in R:

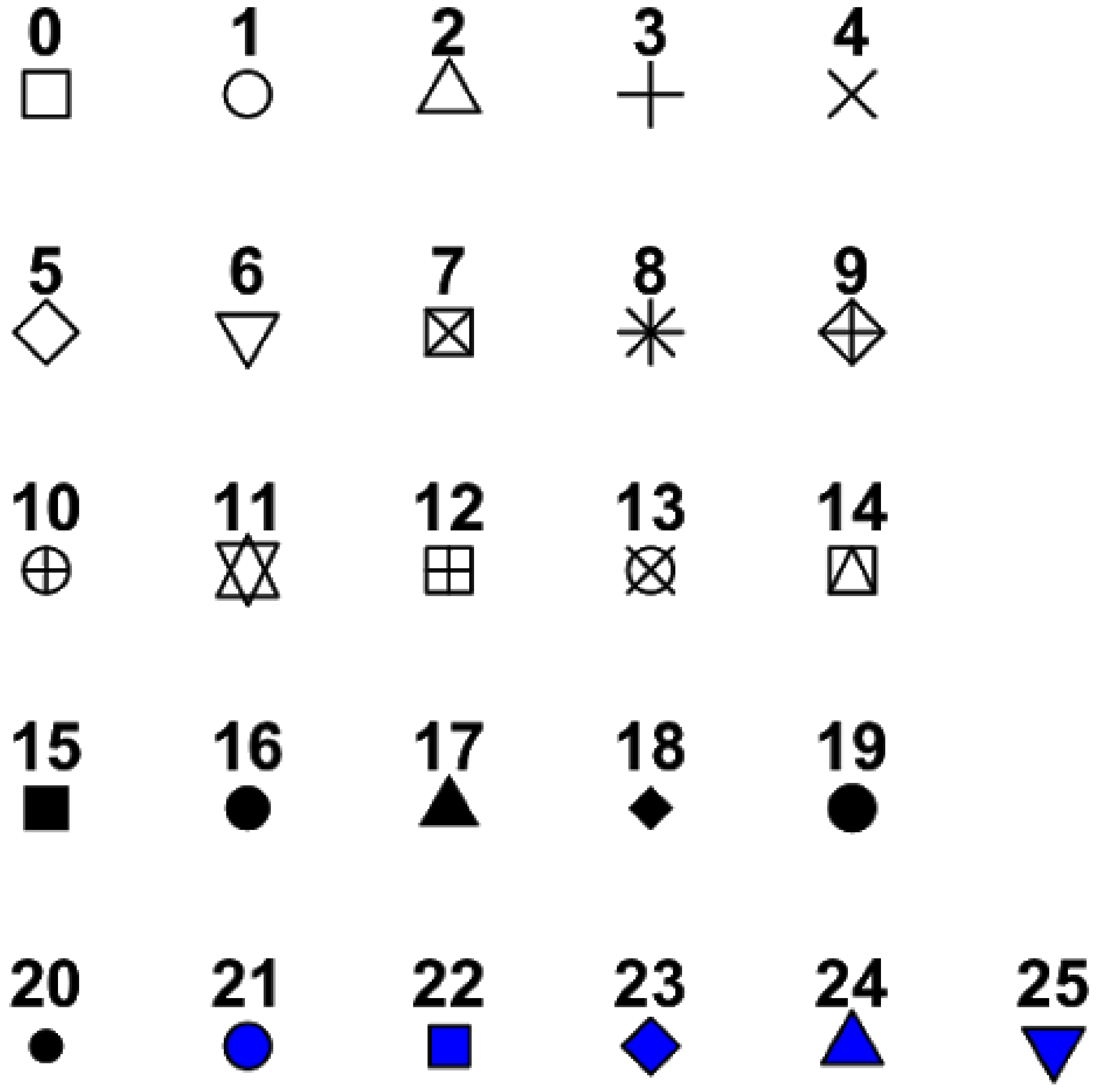


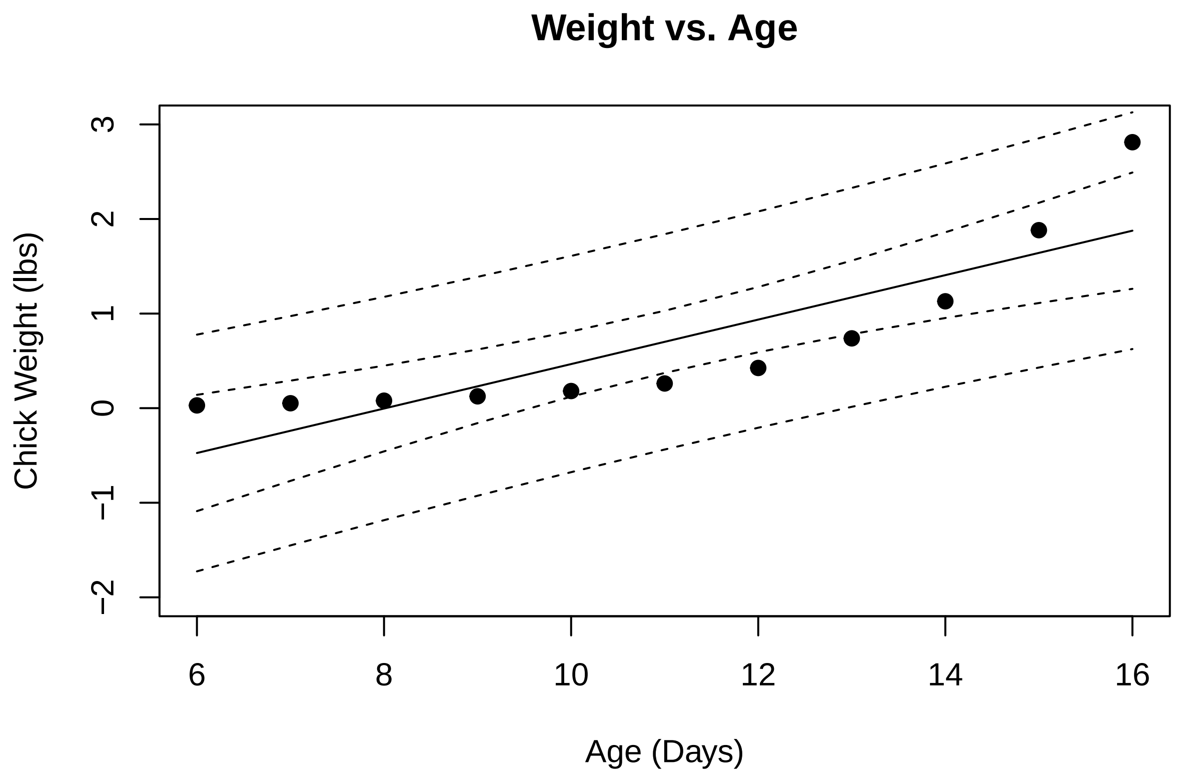
Figure 1: source: sthda.com

```r
plot(wgt~age,data=chick_weight,main="Weight vs. Age",
     xlab="Age (Days)",ylab="Chick Weight (lbs)",
     ylim=c(-2,3),pch=19)

lines(age.vals$age,age.predict)

lines(age.vals$age,weight.ci[,2],lty=2)
lines(age.vals$age,weight.ci[,3],lty=2)

lines(age.vals$age,weight.pi[,2],lty=2)
lines(age.vals$age,weight.pi[,3],lty=2)
```



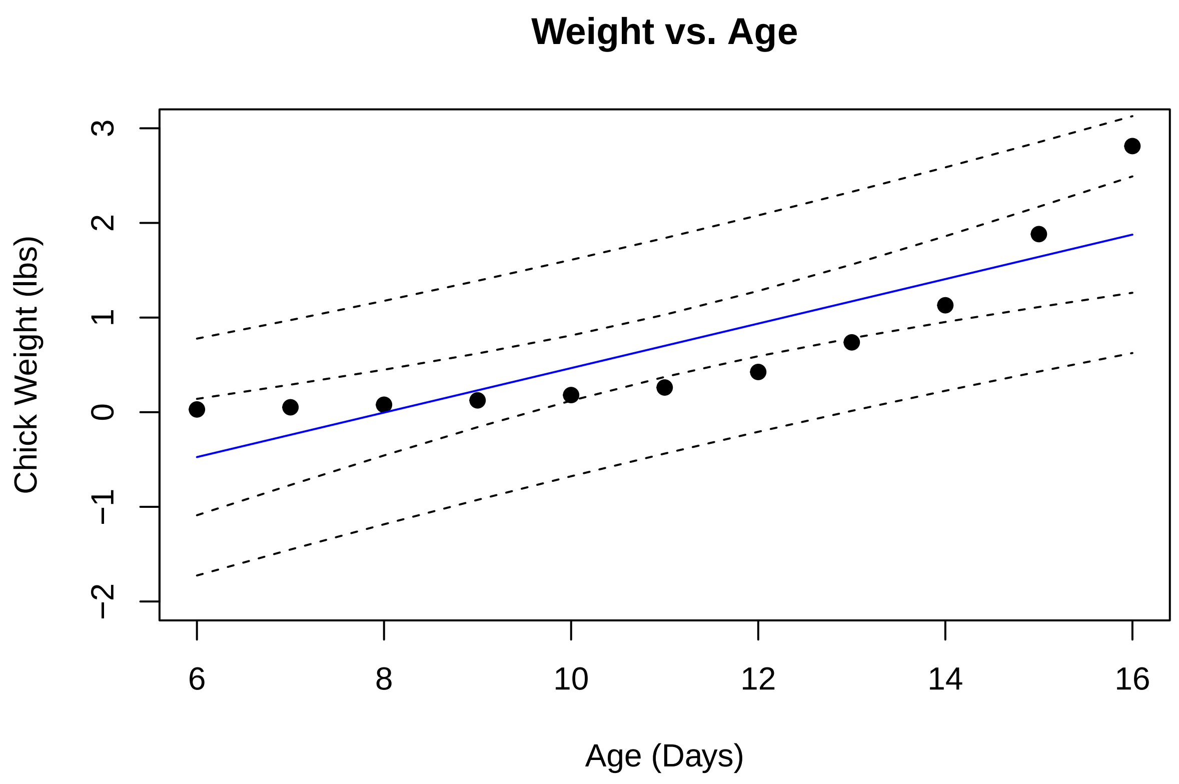
We'll do one more thing to clean up this plot. For the **plot()** function or the **lines()** function, we can use the "col=" argument to set the color. R takes colors in a lot of different formats, but the easiest one is to give it a string with the color we want. There are a lot of different R color names, but for now we can just use "blue". We'll set the fitted line to blue to make it easier to distinguish from the data points.

```r
plot(wgt~age,data=chick_weight,main="Weight vs. Age",
     xlab="Age (Days)",ylab="Chick Weight (lbs)",
     ylim=c(-2,3),pch=19)

lines(age.vals$age,age.predict,col="blue")

lines(age.vals$age,weight.ci[,2],lty=2)
lines(age.vals$age,weight.ci[,3],lty=2)

lines(age.vals$age,weight.pi[,2],lty=2)
lines(age.vals$age,weight.pi[,3],lty=2)
```



This plot looks much better. To save it and put it in a report, you can right-click on the plot in Rstudio and save the image, after adjusting the plot to the size you want it to be. You can also click "export > save as image" in the plot window of RStudio.