Notes re FEV

Authors: Scott S. Emerson, M.D., Ph.D. and Brian D. Williamson
University of Washington Department of Biostatistics

Contents

1 Introduction

This document is meant to give the user an introduction to R, illustrate the use of some of the major `uwIntroStats` functions, and go through an example data analysis. The examples will be shown with R code and output. For more options, see the help files provided with the package. We also assume that the user is using RStudio (found on "http://www.rstudio.com/products/RStudio/") - however, most of the output should be the same. Here we introduce the format for each section and subsection below:

What do we want?

Load the required packages (first install them if you have not already done so).

How do we do that?

The `library()` and `attach()` functions (and `install.packages()` if we need to install a package).

Interpretation

To load the packages that `uwIntroStats` relies on (if you don't have them installed, run `install.packages()` for each):

```
> ## uwIntroStats relies on the Exact, survival,
> ## plyr, and sandwich packages
> library(Exact)
> library(survival)
> library(plyr)
> library(sandwich)
```

Now load the `uwIntroStats` package (if you don't have it installed, go to "emersonstatistics.com/R" and download the appropriate file).

```
> library(uwIntroStats)
```

We have now loaded all of the packages that `uwIntroStats` depends on (the first four calls to `library()`). You must load these packages (and `uwIntroStats`) each time that you start a new R session. If you do not load one of the packages, R will give you errors.

2 The `fev` dataset

2.1 Reading in the data

What do we want?

We want to load the `fev` dataset into R for our analysis.

How do we do that?

We will be working with the `fev` dataset. To prepare that (we will see why we name this `fevDat` later):

```
> fevDat <- read.table("http://www.emersonstatistics.com/Datasets/fev.txt", header=TRUE)
> attach(fevDat)
```

Interpretation

We have now loaded the data into R, and can use it in an analysis. The reference manual for the `fev` dataset can be found on "emersonstatistics.com/Datasets". Make sure that we have read in the data correctly by using the `View(fev)` function. This will bring up a window showing the data. Everything looks good, so we can continue. *Note: Generally it is good practice to clean up the data immediately upon reading it in. However, since we are introducing methods, we will go through data cleanup in the following sections.*

2.2 Standard univariate descriptive statistics

What do we want?

A description of the variables in the dataset.

How do we do that?

The `descrip()` function will do all of this for us.

Interpretation

If we wanted to naïvely get a description of the whole dataset, we could type

```
> descrip(fevDat)
          N     Msng  Mean      Std Dev    Min       25%       Mdn       75%       Max
seqnbr:     654     0   327.5     188.9     1.000     164.2     327.5     490.8     654.0
subjid:     654     0   37170     23691     201.0     15811     36071     53638     90001
   age:     654     0   9.931     2.954     3.000     8.000     10.00     12.00     19.00
   fev:     654     0   2.637     0.8671    0.7910    1.981     2.547     3.119     5.793
height:     654     0   61.14     5.704     46.00     57.00     61.50     65.50     74.00
   sex:     654     0   1.486     0.5002    1.000     1.000     1.000     2.000     2.000
 smoke:     654     0   1.901     0.2994    1.000     2.000     2.000     2.000     2.000
```

Notice that we get output for `seqnbr` and `subjid`. If we read the documentation for the FEV dataset, we would know that these two variables are label variables, and thus none of the descriptives are of interest. For example, the standard deviation for `subjid` is 23691, because we have 654 patients with different subject ids. Thus we can ignore these two labelling variables for descriprive purposes. However they, and all of the other variables, show us that there are no missing data as well. Now if we are interested in seeing that all subjects were unique:

```
> cnt <- length(unique(subjid))
> print(cnt)
[1] 654
> cnt
[1] 654
```

We already knew that there were 654 observations, and now that we know `cnt` is 654, we know that all of the observations are unique (in other words, no subject was counted twice). Notice that here we typed two different things to display `cnt`. This illustrates that when we type `cnt` into R, it actuall calls `print(cnt)`. That is why R is called a *functional* language.
We might also want to check for outliers. Look at the distribution of `age`:

```
> descrip(age)
       N     Msng  Mean      Std Dev    Min       25%       Mdn       75%       Max
age:     654     0   9.931     2.954     3.000     8.000     10.00     12.00     19.00
```

The minimum value is 3. This might be odd, since why would a 3 year old be enrolled in a smoking study? A 3 year old is incredibly unlikely to smoke. How can we tell if this is an outlier? If we look at the mean and the median, we see that they are quite close together - 9.931 and 10, respectively. A large difference between the mean and median would indicate the presence of an outlier, because the mean is heavily influenced by outliers. To illustrate, notice that the formula for the mean

$$\frac{1}{n}\sum_{i=1}^{n} X_i,$$

where the X_i are each person's age in this example, is heavily influenced by a single point. If we made one age 100, then the mean would be pulled to the right. The median, however, is robust to outliers. The middle value will not change if we add in an age of 100.
Now let's look closer at `sex` and `smoke`.

```
> descrip(sex, smoke)
         N     Msng  Mean      Std Dev    Min       25%       Mdn       75%       Max
  sex:     654     0   1.486     0.5002    1.000     1.000     1.000     2.000     2.000
smoke:     654     0   1.901     0.2994    1.000     2.000     2.000     2.000     2.000
```

Each of these variables takes on either a 1 or a 2. Notice that the mean of `sex` is 1.486. The interpretation of this value is that 48.6% of the values in `sex` are 2. Similarly, 90.1% of the values in `smoke` are 2. This is easy to see, because if all of the values were 1 then the mean would be 1, and if one value is 2 it pulls the mean up. However, having the values coded as 1's and 2's is not very useful to us, because we have to remember the documentation each time we look at the data to remember whether `sex = 1` denotes a male or a female. Thus good practice is to create *indicator variables* to reflect the actual data. Usually we say that a 1 is the trait that we want (i.e. if we make a variable called `male`, then a 1 will reflect a male) while a 0 reflects the opposite. Now in the FEV dataset, we know from the documentation that `sex` is coded with 1 = male, so we can recode as follows:

```
> male <- ifelse(sex==1, 1, 0)
```

which is simply saying that if `sex` is a 1 at a given position, set `male` at that position equal to 1. If `sex = 2`, set `male = 0`. We want to create a similar indicator variable out of `smoke`. In this case we want to know if someone smokes or not, so our indicator variable will take on a 1 if they smoke and a 0 if not. In this data, `smoke = 2` for a nonsmoker, so we have to be careful when recoding:

```
> smoker <- ifelse(smoke==2, 0, 1)
```

Our data now has a much more intuitive interpretation. It is always good practice to clean up your data before running any analysis. If we run `descrip()` again we can see this:

```
> descrip(male, smoker)
          N     Msng  Mean      Std Dev    Min       25%       Mdn       75%       Max
  male:     654     0   0.5138    0.5002    0.0000    0.0000    1.000     1.000     1.000
smoker:     654     0  0.09939    0.2994    0.0000    0.0000    0.0000    0.0000    1.000
```

See now that the means have changed. In fact, we see that 51% of the data in `sex` are males, which is a much more informative description than before. However, in practice, the only appropriate descriptives from this output are the count and the number of missing observations.

2.3 Histograms

What do we want?

We want to check for bimodality in any of the variables.

How do we do that?

We use histograms, which plot the frequency of each unique value in the data (or places the data into bins decided by the function). The `hist()` function does all of this.

Interpretation

We can add the arguments `main="maintitle"`, `xlab="x label"`, and `ylab="y label"` to the `hist()` function to set a title, x-axis label, and y-axis label, respectively. By default R will set the x and y-axis labels to the variable(s) entered. Notice that we are allowed to use the name `fev` here because we have named the dataset `fevDat`. If we had named it `fev`, we would be in trouble because R would attempt to create a histogram of the whole dataset.

```
> hist(age, main="Age")
```

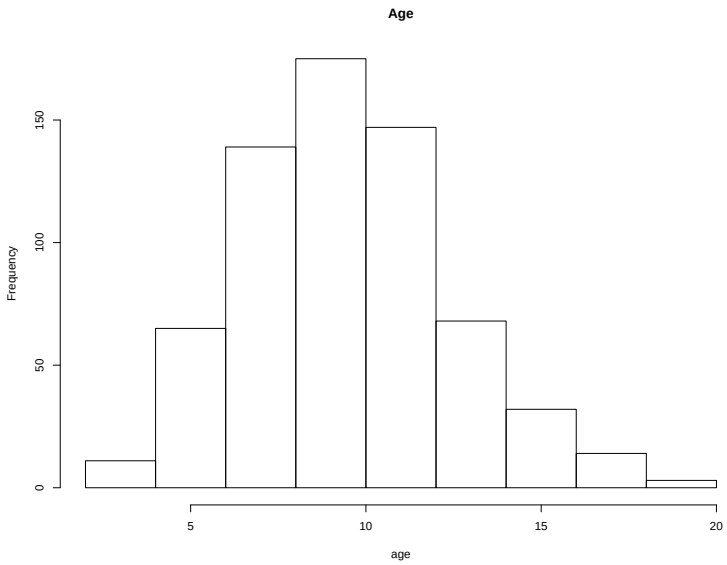

Figure 1: Histogram of Age

```
> hist(height, main="Height")
```

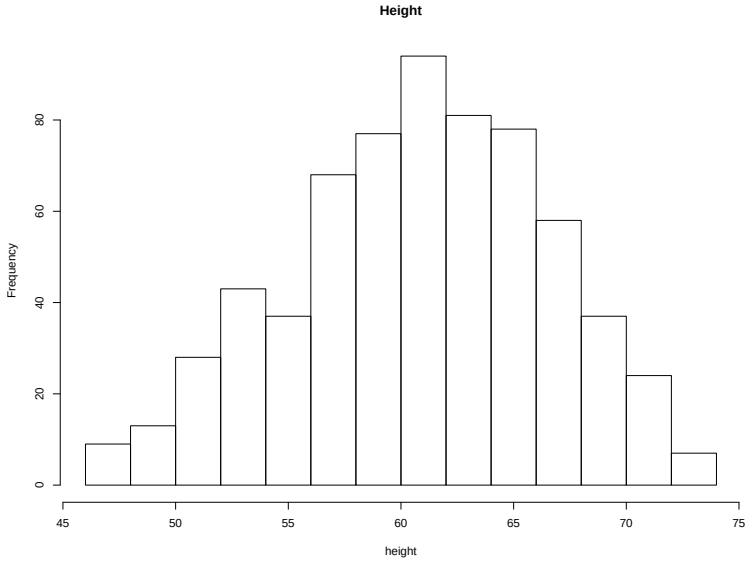

Figure 2: Histogram of Height

```
> hist(fev, main="FEV")
```

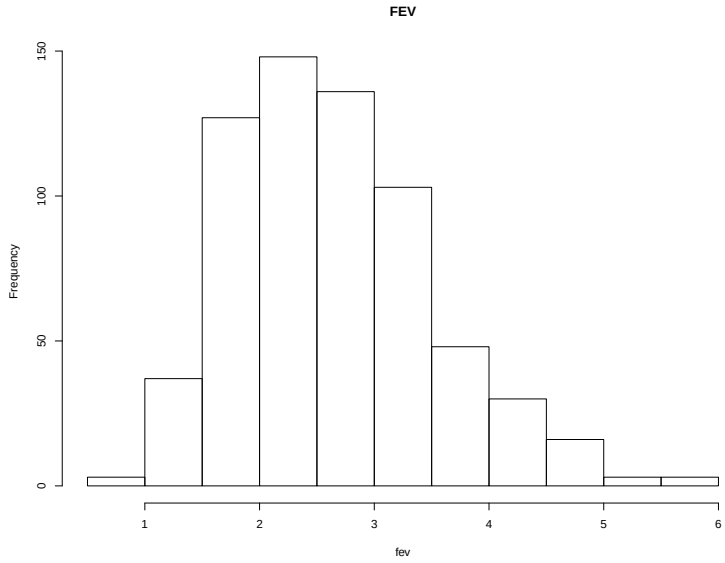

Figure 3: Histogram of FEV

There is no bimodality in the data (each variable has only one peak) and thus we do not need to account for this in our analysis.

2.4 Stratified descriptive statistics

What do we want?

Sometimes we believe that a variable has an effect on another. For example, we believe that sex has an effect on height. To see this effect, we need to stratify the height variable - that is, examine height in each sex individually.

How do we do that?

The `descrip()` function allows us to do this easily. We can stratify on any number of variables by adding them to the `strata` argument.

Interpretation

```
> descrip(age, height, strata=male)
                  N     Msng  Mean      Std Dev    Min       25%       Mdn       75%       Max
   age:  All        654     0   9.931     2.954     3.000     8.000     10.00     12.00     19.00
   age:    Str  0   318     0   9.843     2.933     3.000     8.000     10.00     12.00     19.00
   age:    Str  1   336     0   10.01     2.976     3.000     8.000     10.00     12.00     19.00
height:  All        654     0   61.14     5.704     46.00     57.00     61.50     65.50     74.00
height:    Str  0   318     0   60.21     4.792     46.00     57.50     61.00     63.50     71.00
height:    Str  1   336     0   62.03     6.331     47.00     57.00     62.00     67.50     74.00
```

2.5 Box Plots

What do we want?

We want a graphical representation of the data with the mean, standard deviation, range, and potential outliers displayed.

How do we do that?

The `bplot()` function builds on the base R function `boxplot()`. Now we can add data, means, and standard deviations to the boxplot.

Interpretation

First, the naïve box plot for both age and height by sex - which means use the `male` variable (we only present code for the last plot, with the default values):

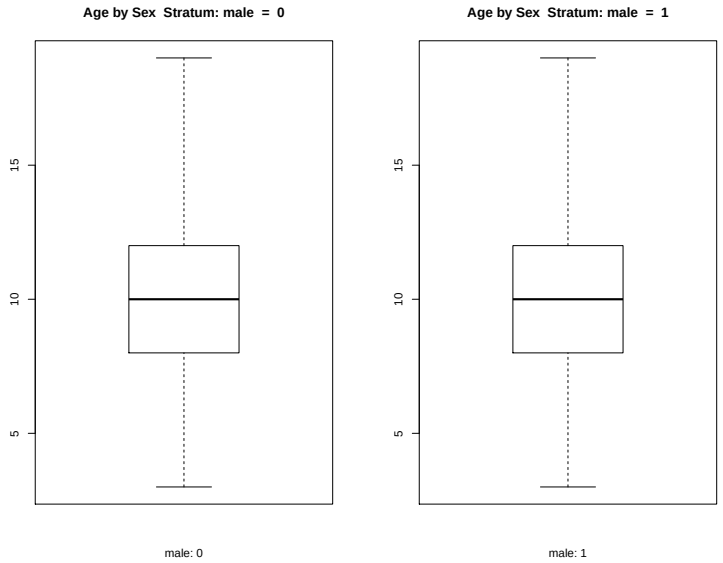

Figure 4: Boxplot of Age by Sex

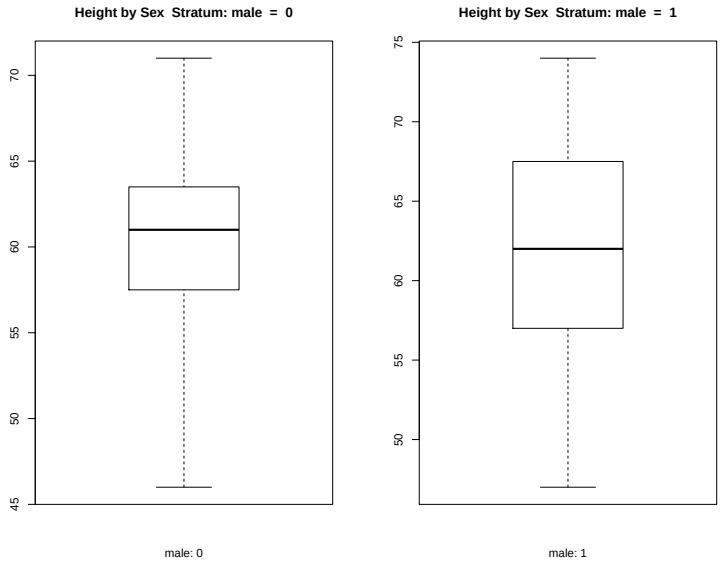

Figure 5: Boxplot of Height by Sex

However, we can add jittered data to make this plot more informative (`bplot()` does this by default), and we can set the x-axis labels:

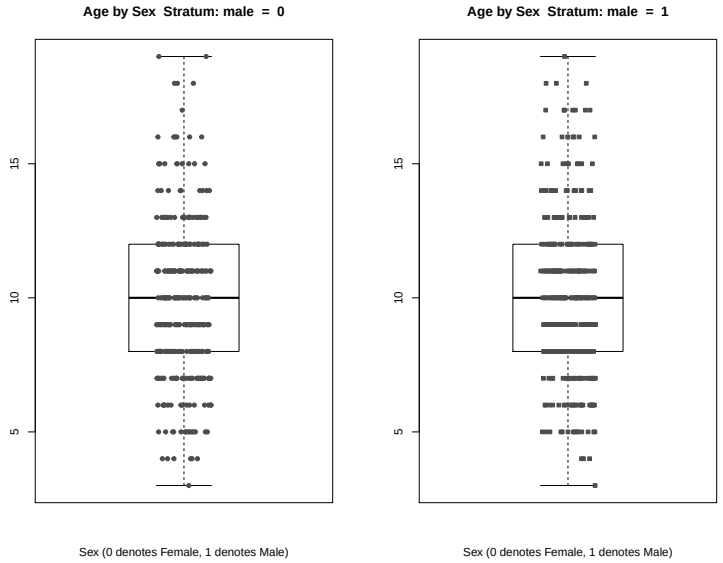

Figure 6: Boxplot of Age by Sex with jittered data

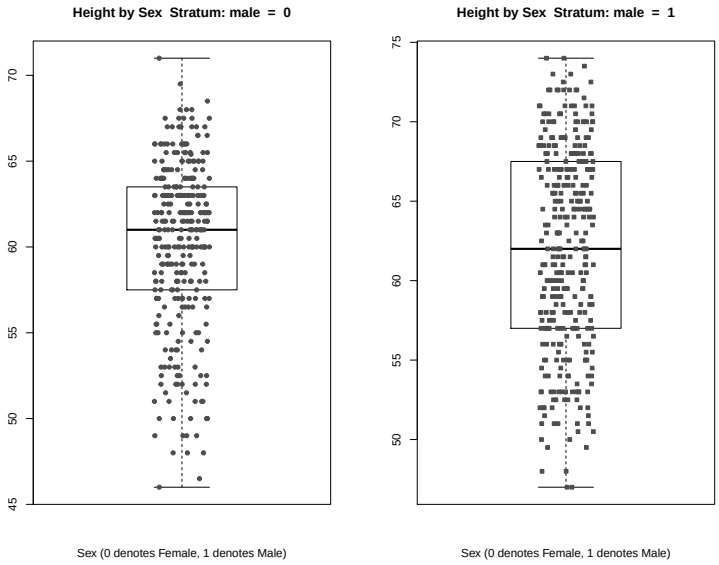

Figure 7: Boxplot of Height by Sex with jittered data

We can also add the mean and standard deviation onto our box plot (`bplot()` does this by default as well) to get the maximum information from the plot:

```
> bplot(age, strata=male, main="Age by Sex", xlab="Sex (0 denotes Female, 1 denotes Male)")
```

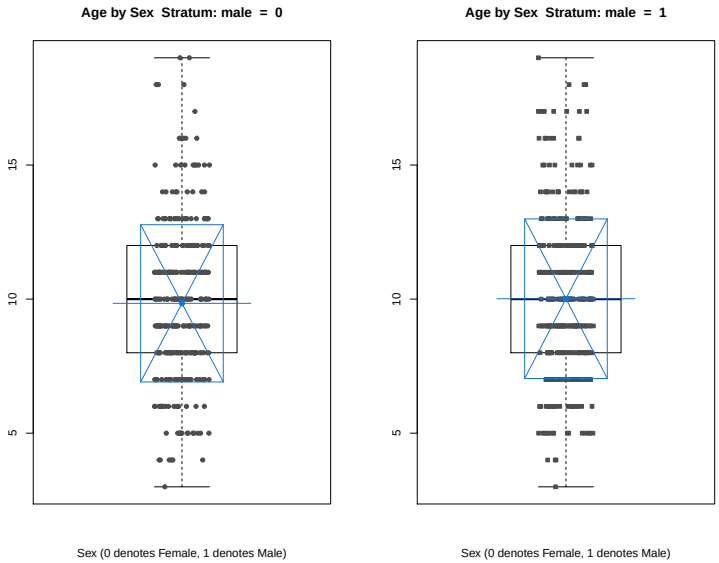

Figure 8: Boxplot of Age by Sex with jittered data and mean and standard deviation overlaid

```
> bplot(height, strata=male, main="Height by Sex", xlab="Sex (0 denotes Female, 1 denotes Male)")
```

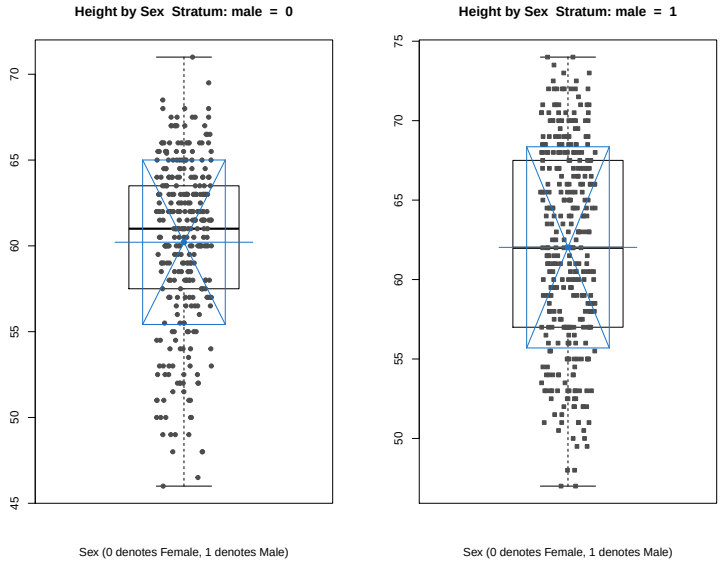

Figure 9: Boxplot of Height by Sex with jittered data and mean and standard deviation overlaid

2.6 Scatterplots

Similar to the `bplot()` function, the `scatter()` function defaults to include some useful information. However, we will go through the derivation of the default values, so the only code shown will be the last (with defaults). The most basic version of the scatterplot does not differentiate between ties in the data (and thus we see much less than the 654 points on the plot):

However, we like to jitter the data so that we can see all of the points:

This allows us to break the ties and see all of the data, but still see the discreteness in the sampling. If we wanted to see a line of best fit, we could create one with the `plotLsfit` argument to our `scatter()` function:

If we look closely at the graph, the data doesn't look quite linear. We can solve this issue by plotting a lowess smooth, which will draw attention to the curves in the data:

```
> scatter(height, age)
```

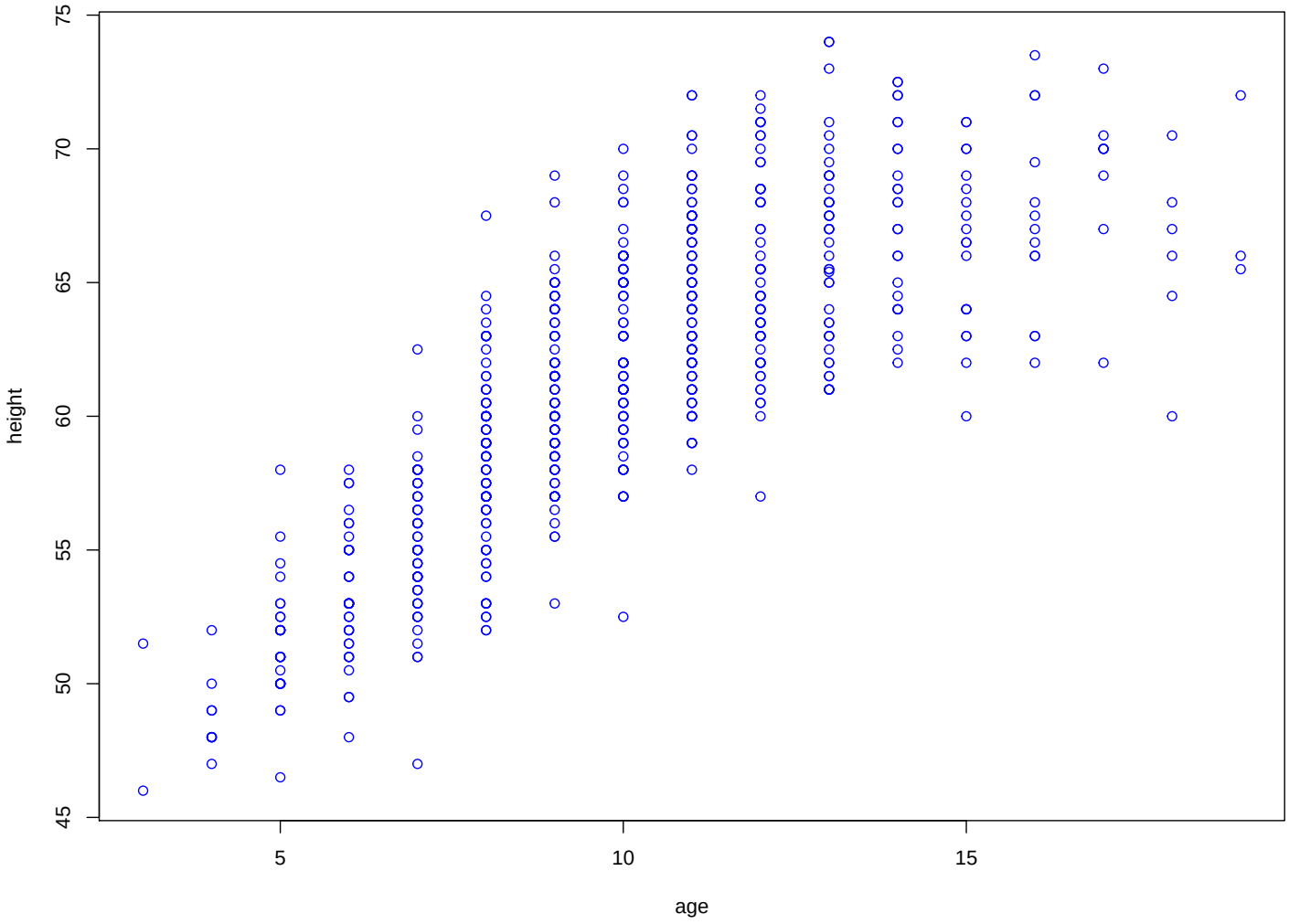

Figure 10: Scatterplot of Age vs Height

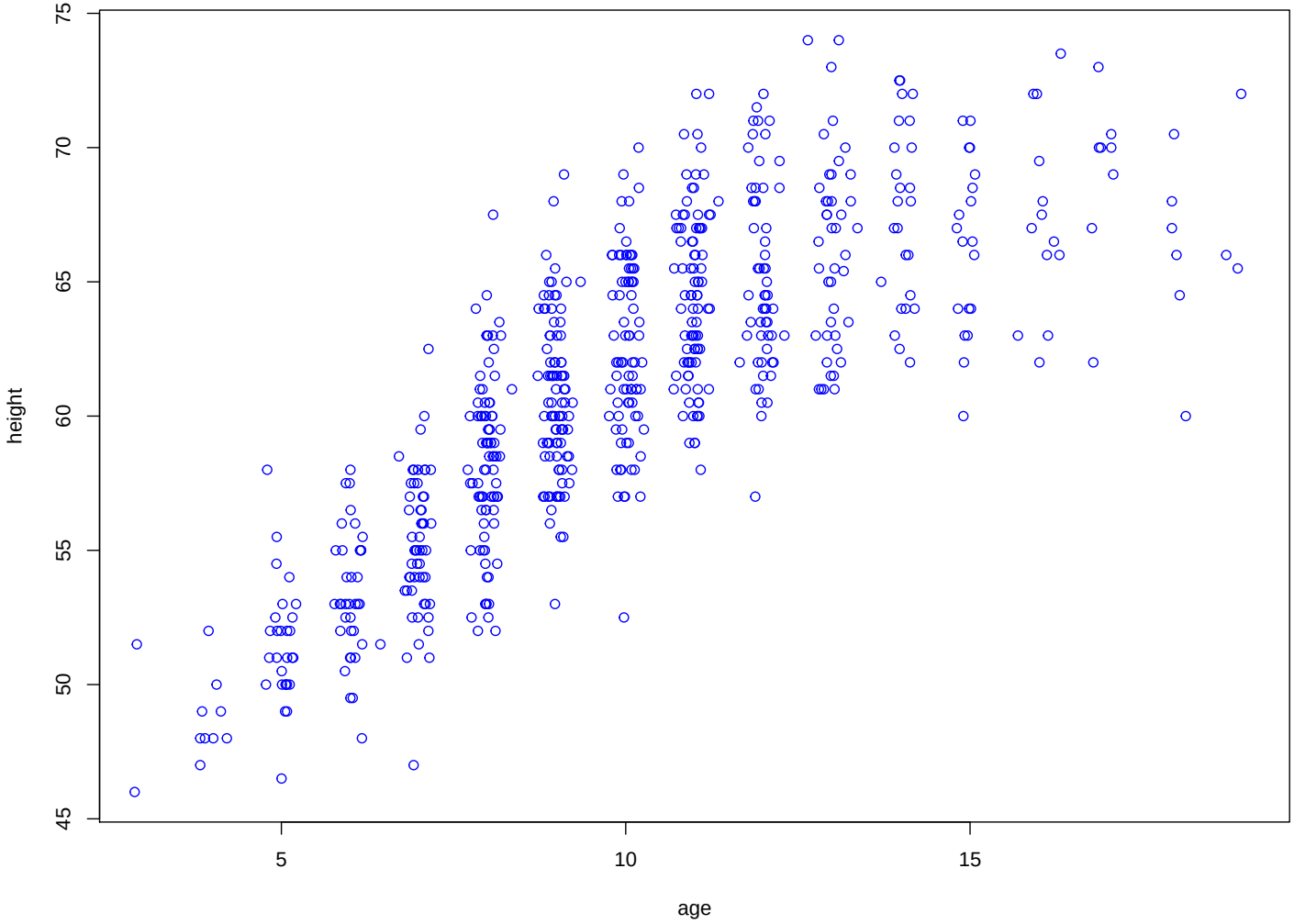

Figure 11: Scatterplot of Age vs Height with jittered data

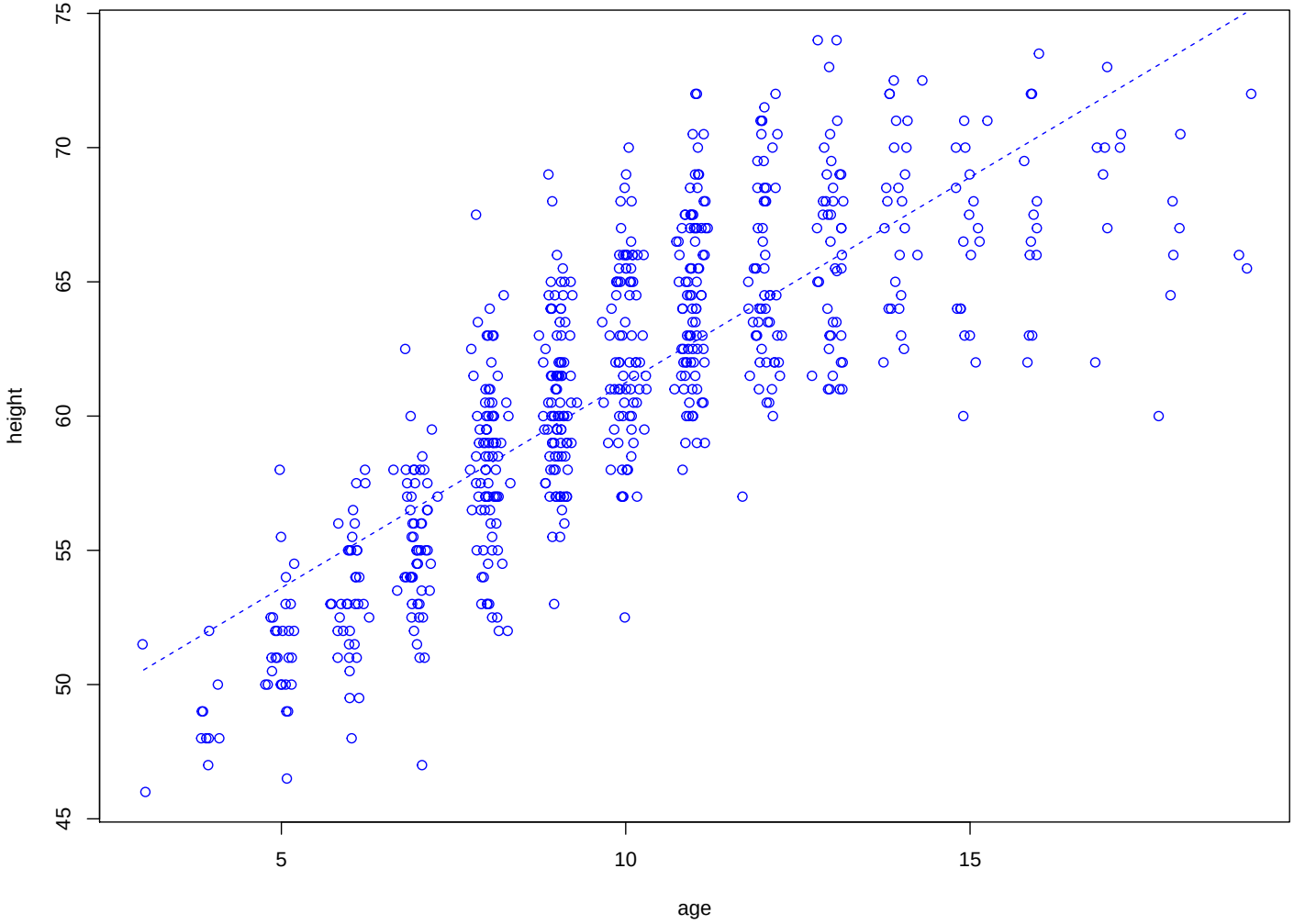

Figure 12: Scatterplot of Age vs Height with jittered data and least squares line

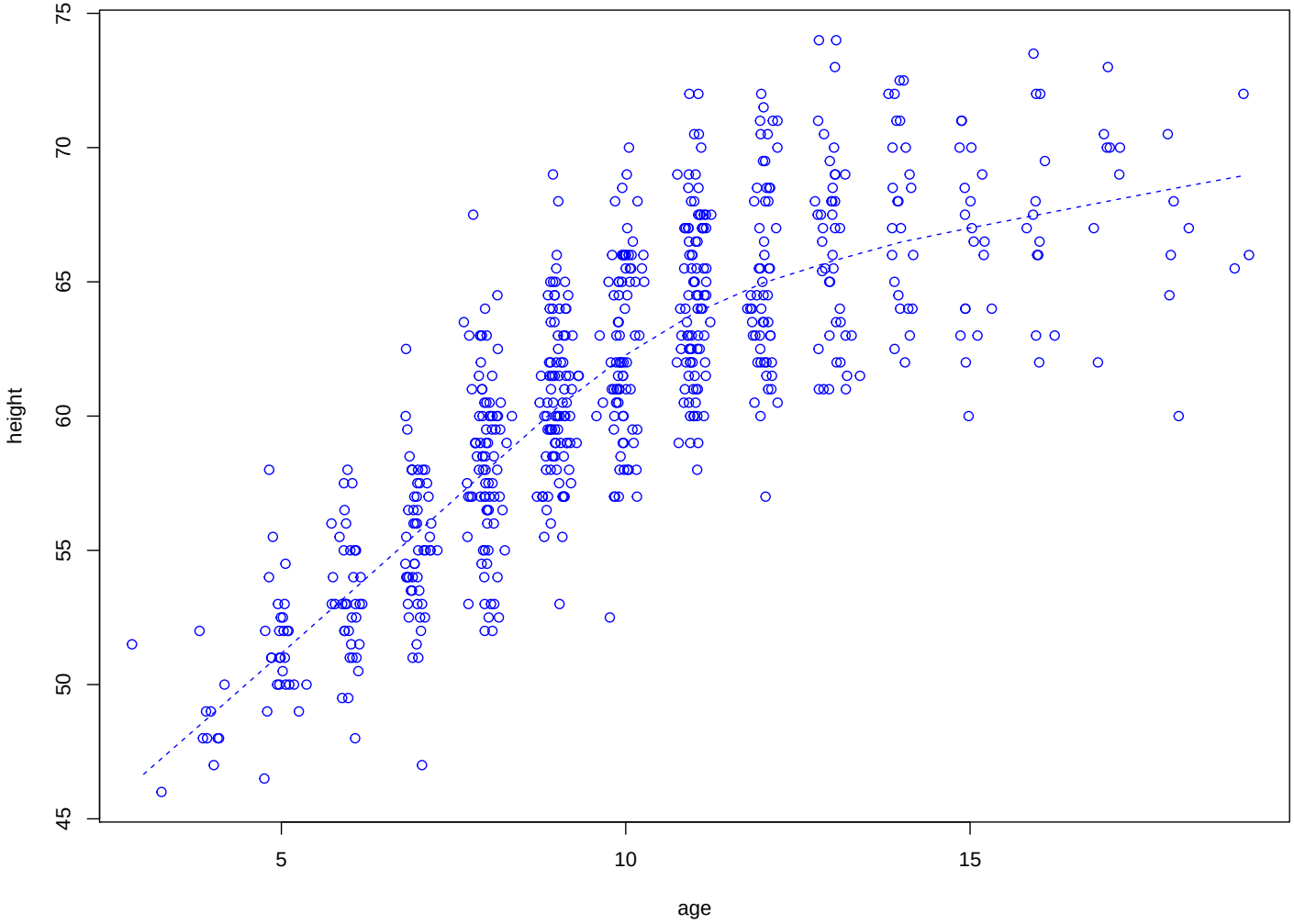

Figure 13: Scatterplot of Age vs Height with jittered data and lowess smooths

2.7 Confounding

There might be potential confounders in the data. For instance, if smoking stunts growth, and only older kids smoke, then at higher ages we will have two populations separated by some vertical distance, while at lower ages we have only one population. It is generally useful to plot a stratified scatterplot with lowess smooths to check for confounders. By default, `scatter()` will use different colors for the strata. We can also label the axes and/or the graph:

```
> scatter(height, age, strata=smoker, main="Height vs Age by Smoking Status")
```

The legend gives us the color and type of point plotted for each stratum. Now if we wanted to see the lowess smooths for males and females, we can add them as a stratification variable:

```
> scatter(height, age, strata=cbind(smoker, male), main="Height vs Age by Smoking Status")
```

Now notice that we can compare groups both across sex (`sex 1` is male and `sex 2` is female) or across smoking.

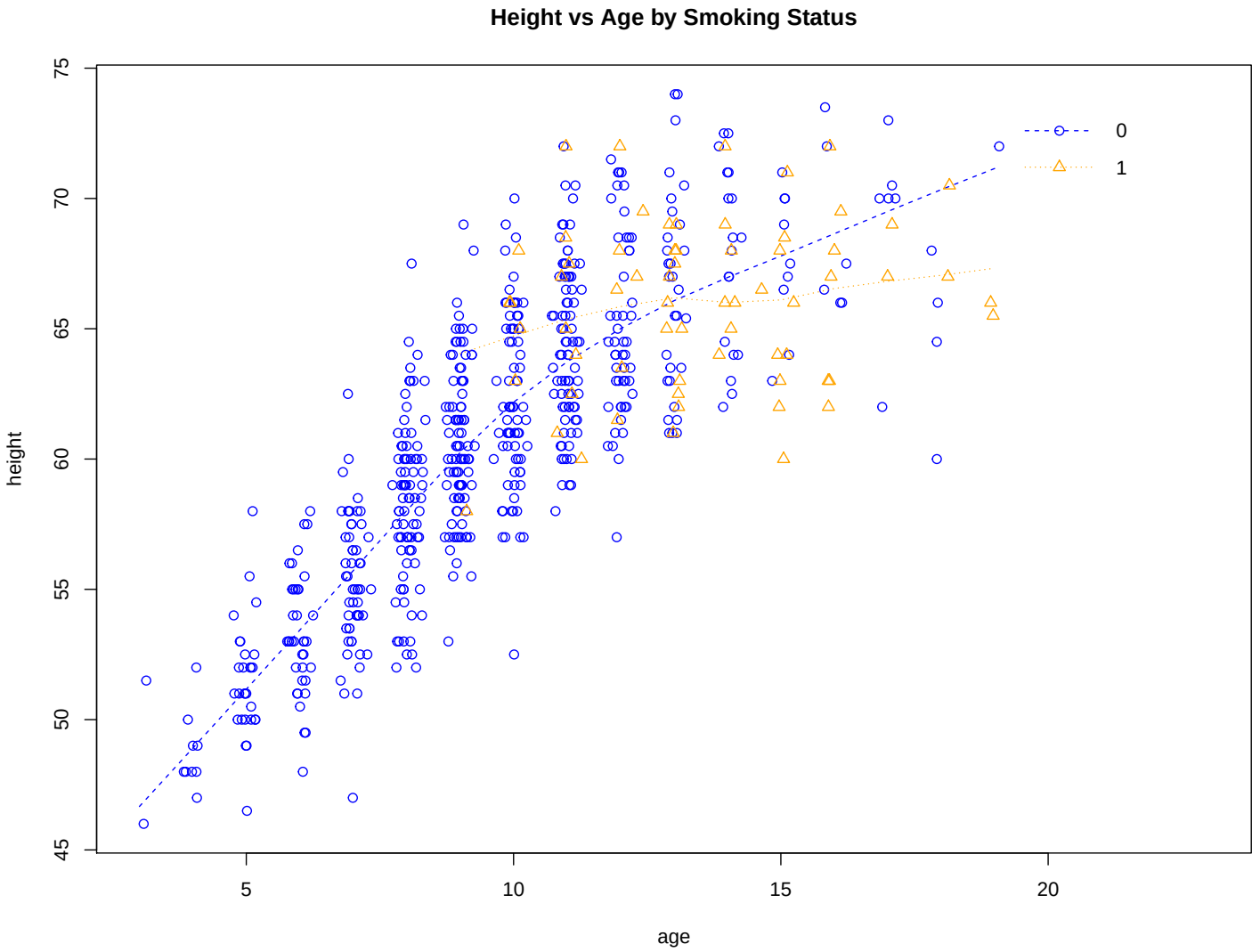

Figure 14: Scatterplot of Age vs Height by Smoking Status

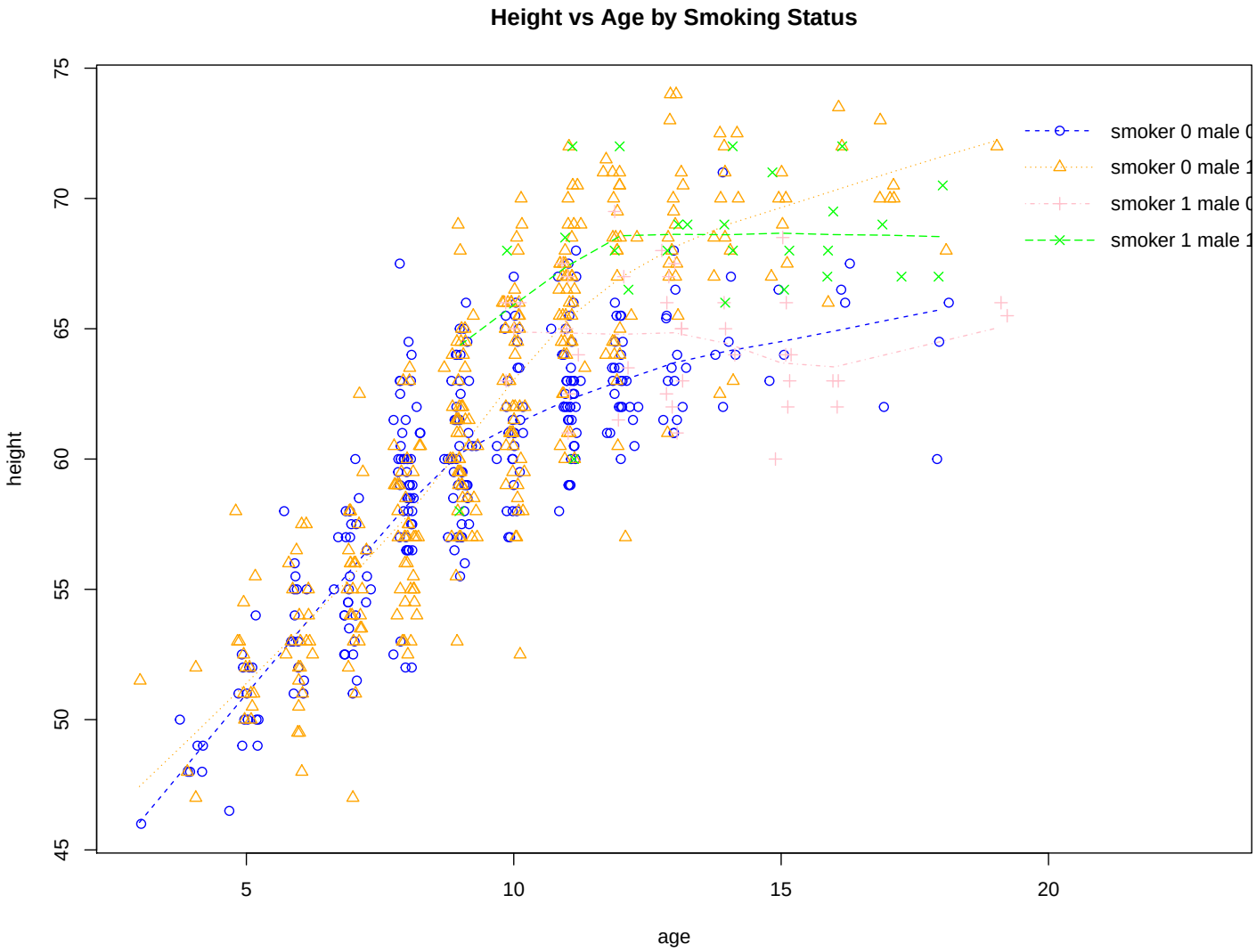

Figure 15: Scatterplot of Age vs Height by Smoking Status and Sex

2.8 Trends and Transforming the Data

Convinced that smoking is not too much of a confounder, we now look at age vs fev. If we do so first with no lowess smooth, it is hard to tell what the trend in the data is:

```
> scatter(fev, age, plotLowess=FALSE, main="FEV vs Age")
```

Any pattern is incredibly hard to find. We might think that the data is linear, but putting a lowess smooth on the plot shows us differently:

```
> scatter(fev, age, main="FEV vs Age")
```

Now we can see a hint of an S-shaped trend in the data. We believe this, because if we believe that FEV is actually best predicted by height, then at young ages height is roughly linear with age but at older ages height is flat. In order to really understand what is going on, we must look at the relationship between FEV and height:

```
> scatter(fev, height, main="FEV vs Height")
```

This is definitely not a linear relationship. In fact, it looks like a cubic relationship. Now if we look at the plot stratified by sex, we don't see too much of a sex effect:

```
> scatter(fev, height, strata=male, main="FEV vs Height by Sex")
```

If we create a new variable called `htcub` (for height cubed) we can see that there is more of a straight line relationship with FEV:

```
> htcub <- height^3
> scatter(fev, htcub, main="FEV by Cubed Height")
```

There is still heteroscedasticity, however. To remedy this, we can try plotting the cube root of FEV vs Height:

```
> cubrtfev <- fev^(1/3)
> scatter(cubrtfev, height, main="Cube Root of FEV by Height")
```


```
> cubrtfev <- fev^(1/3)
> scatter(cubrtfev, height, main="Cube Root of FEV by Height")
```

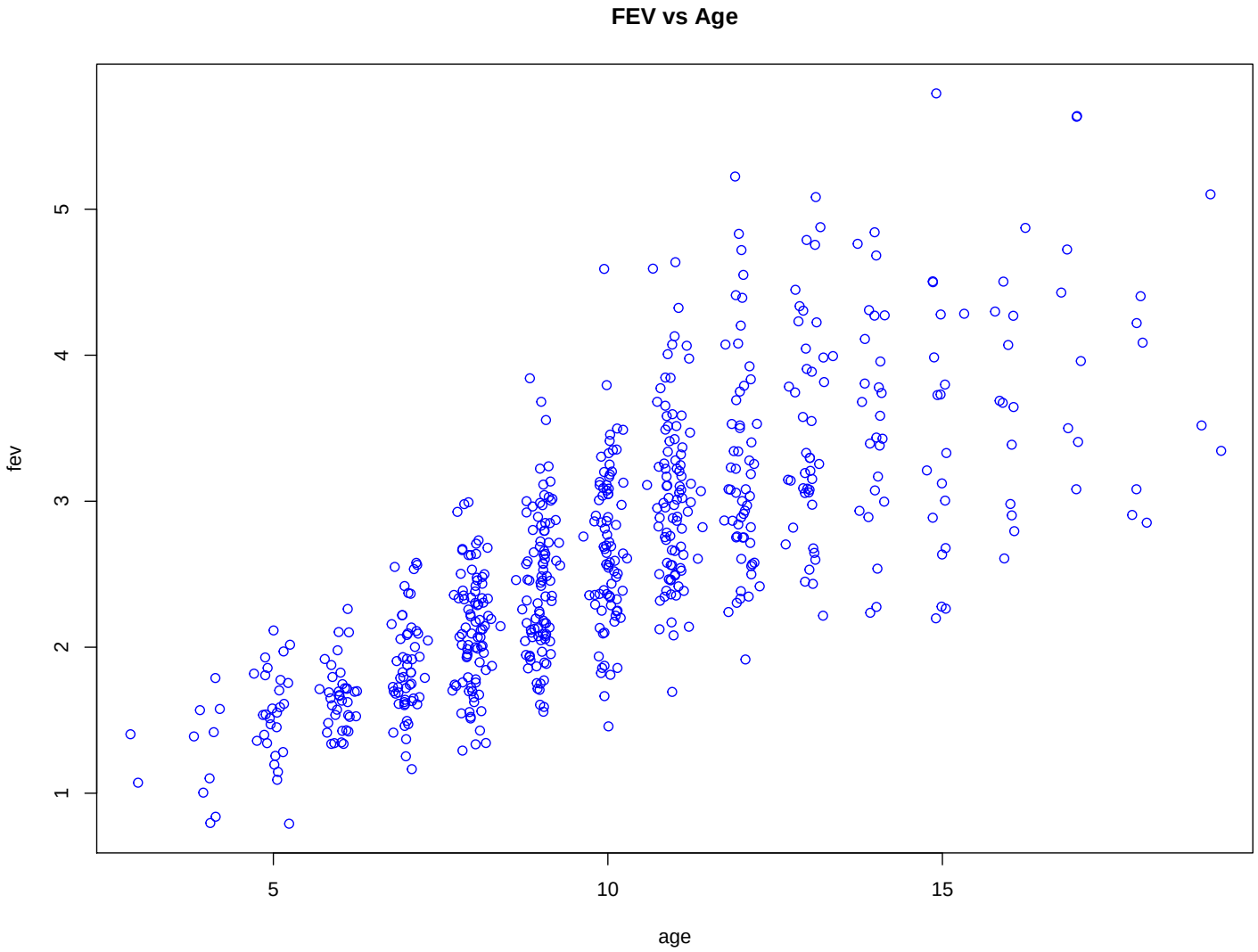

Figure 16: Scatterplot of Age vs Height

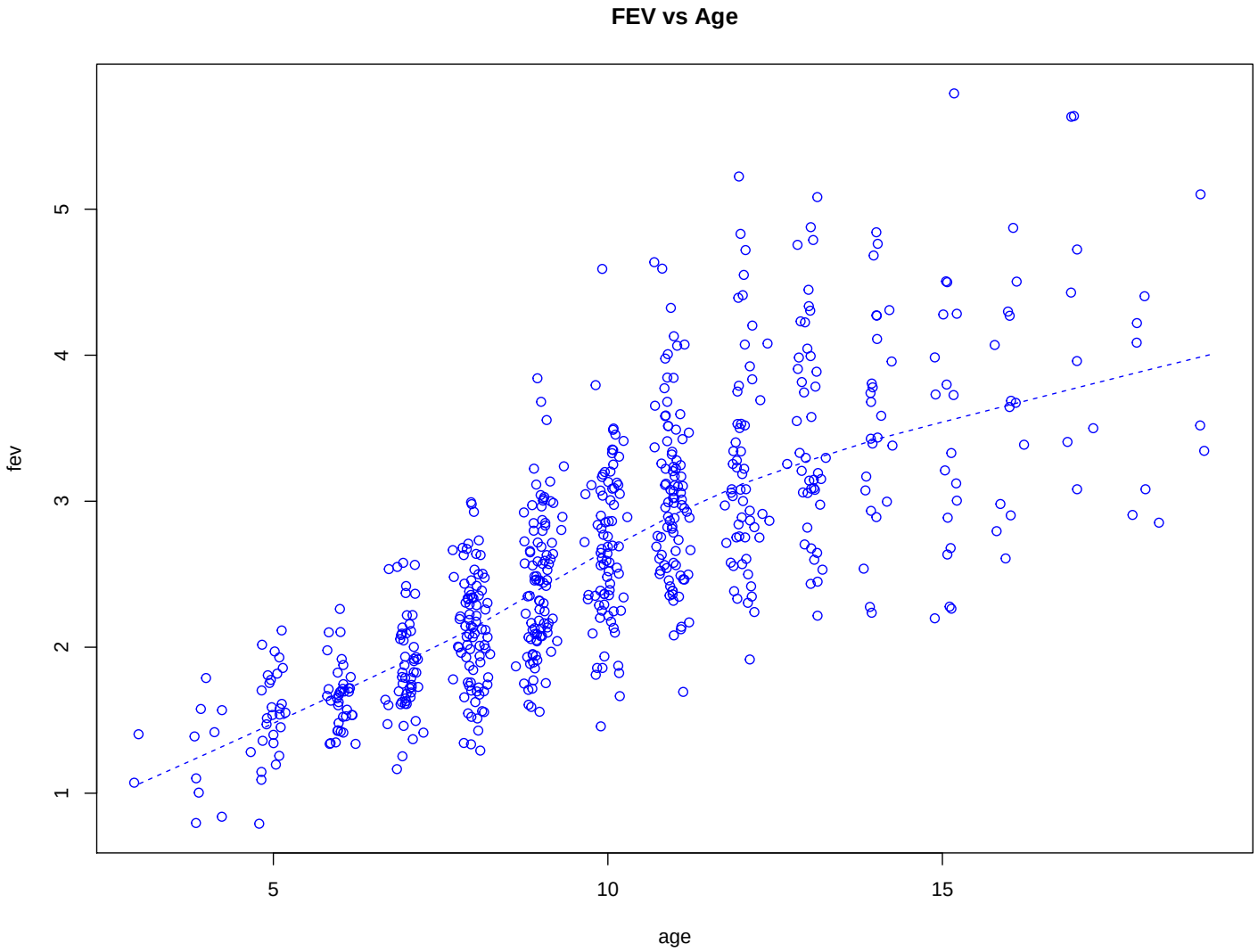

Figure 17: Scatterplot of Age vs Height with lowess smooths

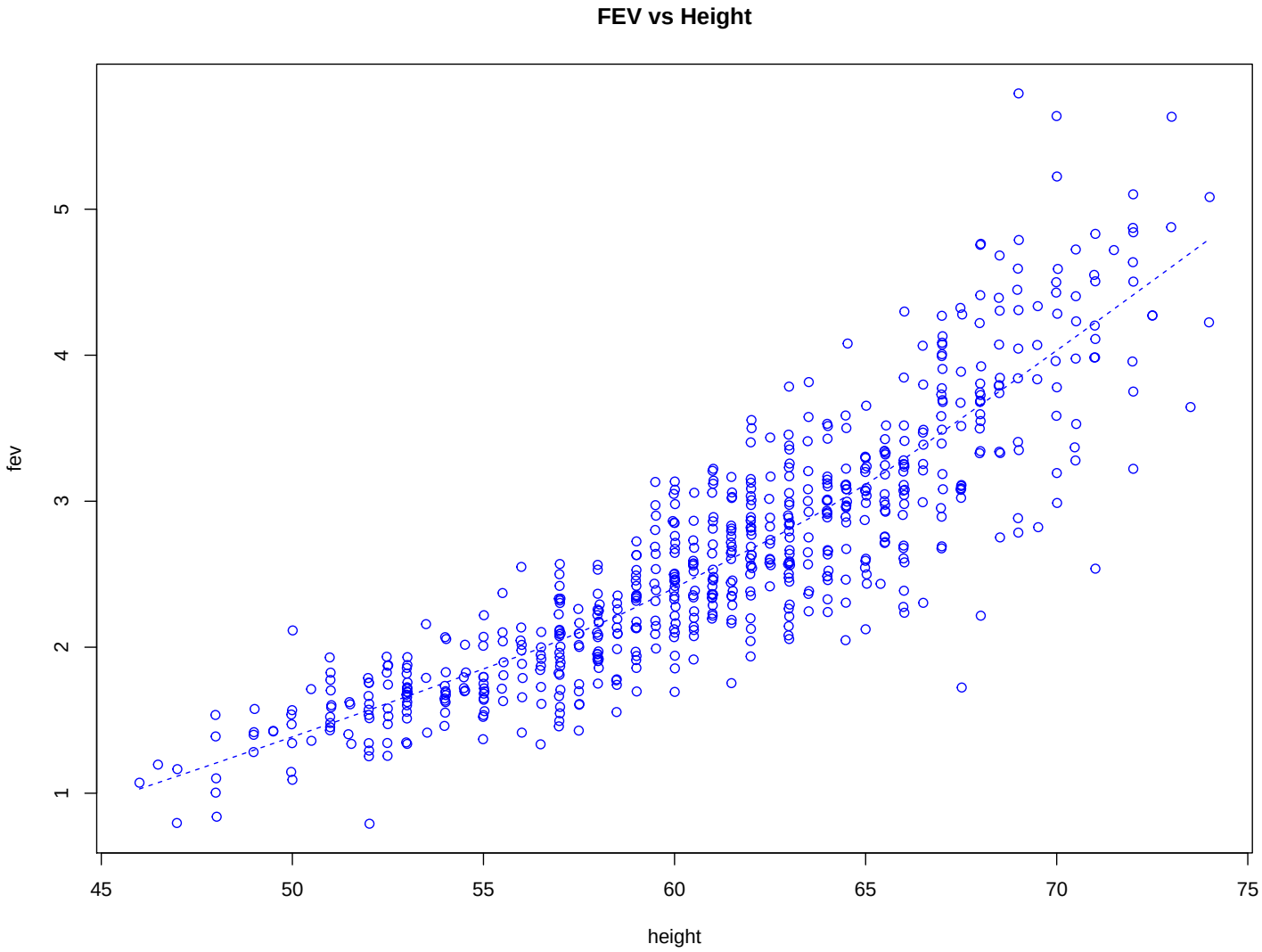

Figure 18: Scatterplot of FEV vs Height

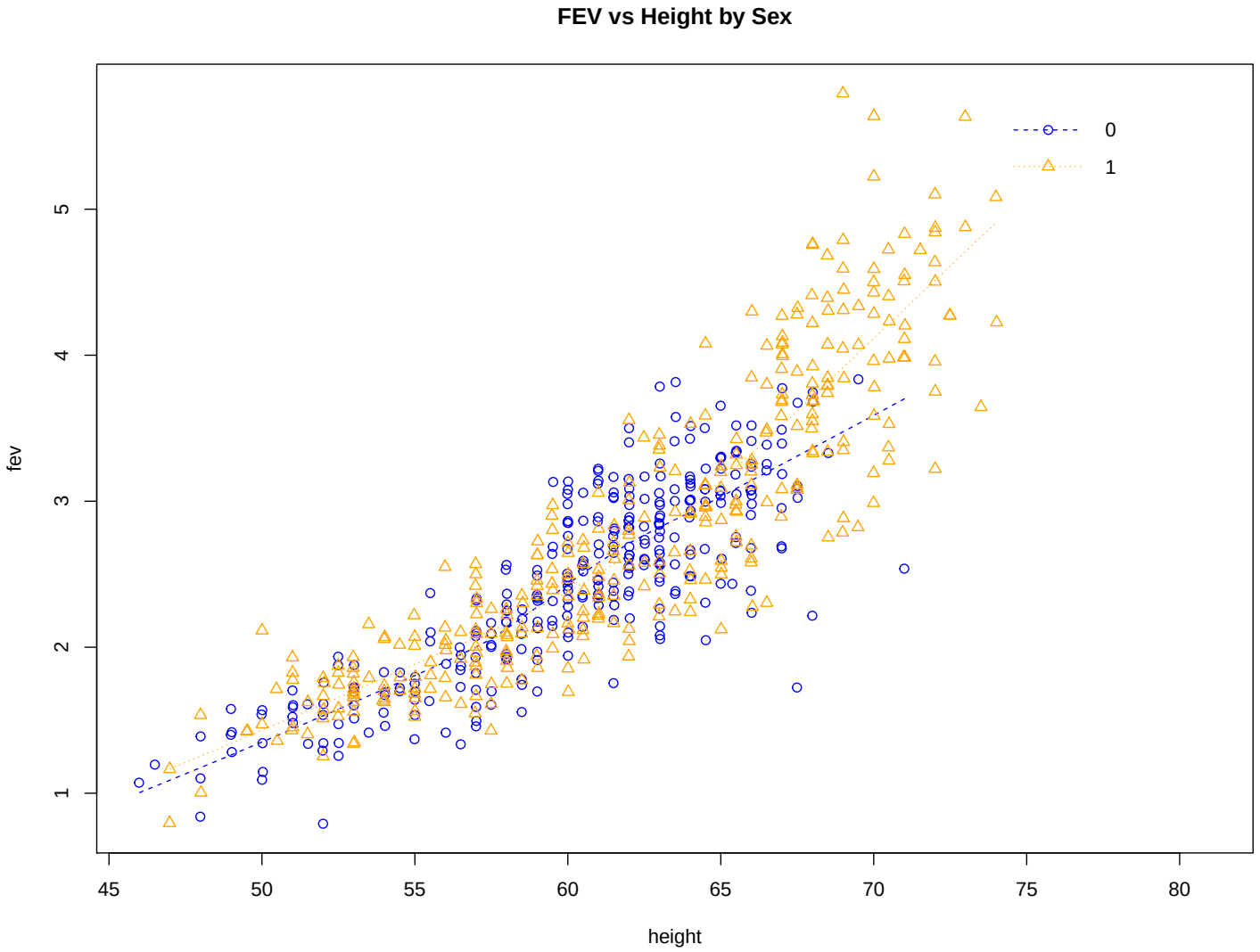

Figure 19: Scatterplot of FEV vs Height by Sex

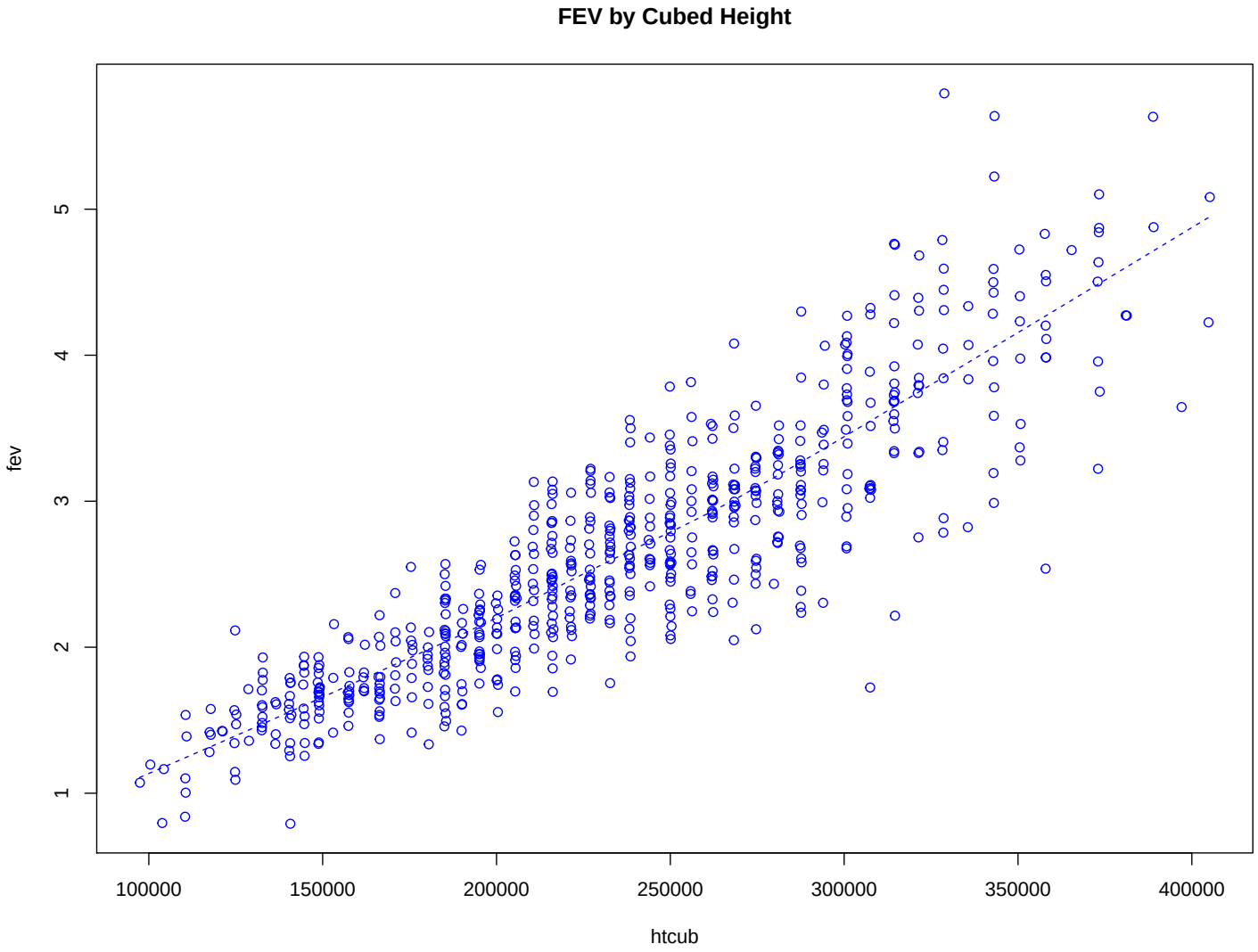

Figure 20: Scatterplot of Age vs Cubed Height

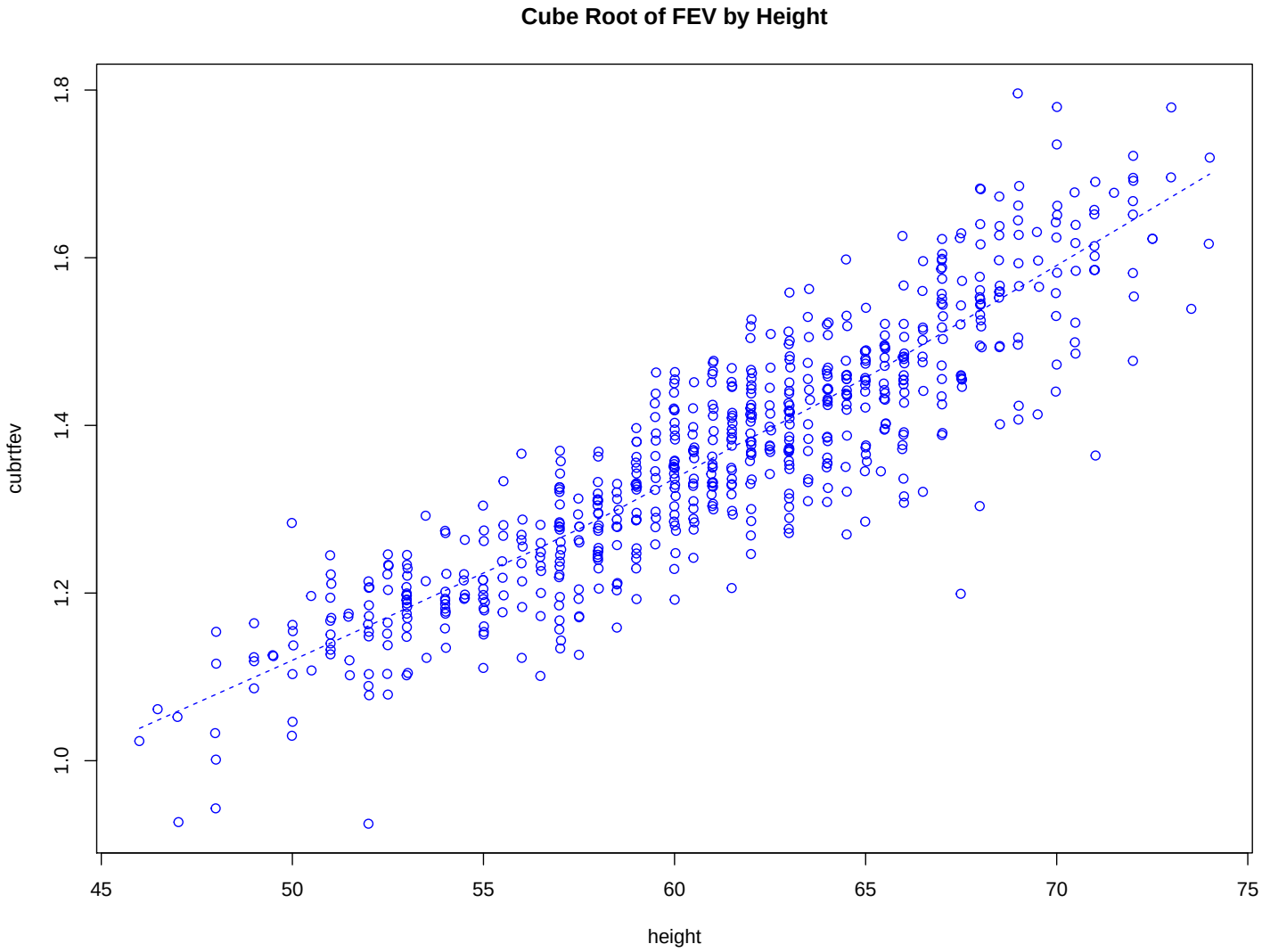

Figure 21: Scatterplot of Cube Root of FEV vs Height

3 The `fev` dataset revisited

Now that we know all of these methods, what would we do if presented with the `fev` dataset? All of our data cleanup would happen immediately when we read in the data, based on the documentation and what we can see in R.

What do we want to do?

We want to read in the `fev` dataset and make it ready for use and analysis.

How do we do that?

We first read in the dataset (renaming it as `fevDat` because we know that one of the variables in the data is named `fev` and we don't want confusion when we attach the data):

```
> fevDat <- read.table("http://www.emersonstatistics.com/Datasets/fev.txt", header=TRUE)
> attach(fevDat)
```

Now we view the data:

```
> View(fevDat)
```

After noticing that `sex` and `smoke` are not coded as indicator variables, we recode them (using the documentation online for reference):

```
> male <- ifelse(sex==1, 1, 0)
> smoker <- ifelse(smoke==2, 0, 1)
```

We also would know which variables are likely confounders or effect modifiers before starting the data analysis. Good practice in statistics dictates the choice of these variables before analysis, so that we are not using our data for exploratory purposes when we want to do inference.

Interpretation

Now we have loaded the dataset, cleaned it up, and know what our confounding variables and effect modifiers are. Now we are ready to create plots and run our analysis.