ACTEX

LEARN TODAY. LEAD TOMORROW.

EXAM PA

Study Manual for SOA Exam PA

by

Ambrose Lo, Ph.D., FSA, CERA

Chapter 2

Data Exploration and Visualization with the ggplot2 Package

*****FROM THE PA EXAM SYLLABUS*****

2. Topic: Data Exploration and Visualization (20-30%)

Learning Objectives

The Candidate will be able to work with various data types, understand principles of data design, and construct a variety of common visualizations for exploring data.

Learning Outcomes

The Candidate will be able to:

d) Apply the key principles of constructing graphs.

e) Apply univariate data exploration techniques.

f) Apply bivariate data exploration techniques.

Chapter overview: An integral part of any predictive analytic exercise is the use of graphical displays to investigate the characteristics of the variables of interest, on their own and in relation to one another. In this regard, one of the key strengths of R as a programming language is that it offers versatile graphing capabilities, both in the base installation and with add-on packages. A wide variety of high-quality graphs can be produced with a minimal amount of code. In Exam PA, you are asked to take advantage of R's graphing capabilities and produce different types of graphical displays. Instead of using R's base graphical platform, you will make graphs using the `ggplot2` package,[i] which may be new to you even if you have used R before. Compared to R's base graphics system, `ggplot2` involves vastly different syntax based on the so-called "grammar of graphics" (in fact, "gg" stands for "grammar of graphics") and lends itself to producing sophisticated graphs that would be cumbersome to create using base R graphics.

Synthesizing the material in the first four chapters of the book *Data Visualization: A Practical Introduction* (which is listed in the exam syllabus), this chapter presents some of the important

[i]The `ggplot2` package is developed by Hadley Wickham, who is also a core developer of RStudio. Earlier the package was called `ggplot`, but substantial changes were made later, so the name of the package was upgraded to `ggplot2`.

graphical functions in the `ggplot2` package most relevant to Exam PA. These functions can be used to construct different types of graphs such as scatterplots, histograms, boxplots, and bar charts, which will all be illustrated in the context of a real insurance dataset. In Section 2.1, we will learn the basic structure of a ggplot, make some simple but informative plots, and learn how to tweak the appearance of a ggplot. Section 2.2 draws upon the data visualization techniques covered in Section 2.1 to perform exploratory data analysis, which is the use of graphs and summary statistics to uncover patterns and relationships in a set of data, and generate hypotheses which can be answered quantitatively in a predictive model at a later stage.

2.1 Making ggplots

2.1.1 Basic Features

Let's begin by installing (make sure to install a package the first time you use it!) and loading the `ggplot2` package.

```
# CHUNK 1
# Uncomment the next line the first time you use ggplot2
#install.packages("ggplot2")
library(ggplot2)
```

With the last command, we can use all the functions in the `ggplot2` package from now onward.

Skeleton. In its simplest form, a ggplot consists of two parts: The core `ggplot()` function (not `ggplot2()`!) and a chain of additional functions pasted together using the plus (`+`) sign defining the exact type of plot to be made.

1. `ggplot()` *function:* The `ggplot()` function initializes the plot, defines the source of data using the `data` argument (almost always a data frame in Exam PA; recall Subsection 1.2.3), and, most importantly, specifies what variables in the data are "mapped" to visual elements in the plot by the `mapping` argument. Mappings in a ggplot are specified using the `aes()` function, with `aes` standing for "aesthetics." They determine the role different variables play in the plot. The variables may, for instance, correspond to visual elements such as the x- or y-variables, color, size, and shape, specified by the `x`, `y`, `color`, `size`, and `shape` aesthetics, respectively.

2. *Geom functions:* Subsequent to the `ggplot()` function, we put in *geometric objects*, or *geoms* for short, which include points, lines, bars, histograms, boxplots, and many other possibilities, by means of one or more *geom functions*. Placed layer by layer, these geoms determine what kind of plot is to be drawn and modify its visual characteristics, taking the data and aesthetic mappings specified in the `ggplot()` function as inputs.

Here is the generic structure of a ggplot: (The uppercase letters are placeholders.)

```
ggplot(data = <DATA>, mapping = aes(<AESTHETIC_1> = <VARIABLE_1>,
                                    <AESTHETIC_2> = <VARIABLE_2>,
                                    ...)) +
```



```
geom_<TYPE>(<...>) +
geom_<TYPE>(<...>) +
<OTHER_FUNCTIONS>  +
...
```

Don't worry if the ideas above seem puzzling. It is commonly acknowledged that the learning curve of `ggplot2` is steep, much more so than R's base graphics system. You will gain a much better understanding of how a ggplot works after going through the example plots in this chapter and in the rest of this study manual.

Case study: Personal injury insurance dataset. To illustrate data visualization and exploration techniques, in this chapter we will look at a personal injury insurance dataset.[ii] This dataset contains the information of 22,036 settled personal injury insurance claims. These claims were reported during the period from July 1989 to the end of 1999, with claims settled with zero payment excluded. The variables in the dataset are described in Table 2.1.

Variable	Description
`amt`	settled claim amount (continuous numeric variable)
`inj`	injury code, with seven levels: 1 (no injury), 2, 3, 4, 5, 6 (fatal), 9 (not recorded)
`legrep`	legal representation (0 = no, 1 = yes)
`op_time`	operational time (a standardized amount of time elapsed between the time when the injury was reported and the time when the claim was settled)

Table 2.1: Data dictionary for the personal injury (`persinj`) insurance claims dataset.

In Section 4.2, we will build a model to predict the size of personal injury insurance claims using other variables in the dataset. For now, we will perform data exploration of the variables in the dataset. The insights we gain here will go a long way towards constructing a good predictive model.

To get started, run CHUNK 2 to load the external CSV file containing the dataset into R as a data frame called `persinj` (meaning "personal injury") using the `read.csv()` function, which takes as an argument the name of the CSV file supplied as a character string. In this preliminary section, we will take out a subset of 50 observations from the `persinj` data, called `persinj50`, and explain the main characteristics of a ggplot using these 50 observations. This will help us appreciate the different types of visual effects that can be produced on a ggplot more easily. In Section 2.2, we will return to the full dataset and learn why we use a certain plot for a certain purpose.

ONE MORE REMINDER!

Please read page xii of the preface of this manual about how to access the Rmd files as well as datasets that go with this manual.

[ii] The original version of the dataset can be found at `http://www.businessandeconomics.mq.edu.au/our_departments/Applied_Finance_and_Actuarial_Studies/research/books/GLMsforInsuranceData/data_sets`, which is part of the companion web page of the textbook *Generalized Linear Models for Insurance Data* (2008), by de Jong and Heller. The dataset has been pre-processed to suit our purpose.


```
# CHUNK 2
persinj <- read.csv("persinj.csv")
# Take out a subset of 50 observations from the full data
persinj50 <- persinj[seq(1, nrow(persinj), length = 50), ]
```

First encounter with ggplots. As our first example, let's make a *scatterplot* for the two numeric variables in the `persinj50` data, `amt` and `op_time`. The plot, produced by the code in CHUNK 3, is given in Figure 2.1.1. The first line of the code makes it clear that we are using the `persinj50` data, where the variables `op_time` and `amt` are mapped to the variables on the x-axis and y-axis through the `x` and `y` aesthetics, respectively. There is no need to name the variables as `persinj50$op_time` or `persinj50$amt` as the data source is already specified in the `data` argument. Given these mappings, we use `geom_point()` to make a scatterplot of `amt` (the y-variable) against `op_time` (the x-variable), which comprises 50 points (hence the name of the function) corresponding to the 50 paired values of the two variables and allows us to see the two variables in comparison with each other. Later, we will fine-tune this plot in different ways to capture different sorts of information.

```
# CHUNK 3
ggplot(data = persinj50, mapping = aes(x = op_time, y = amt)) +
  geom_point()
```

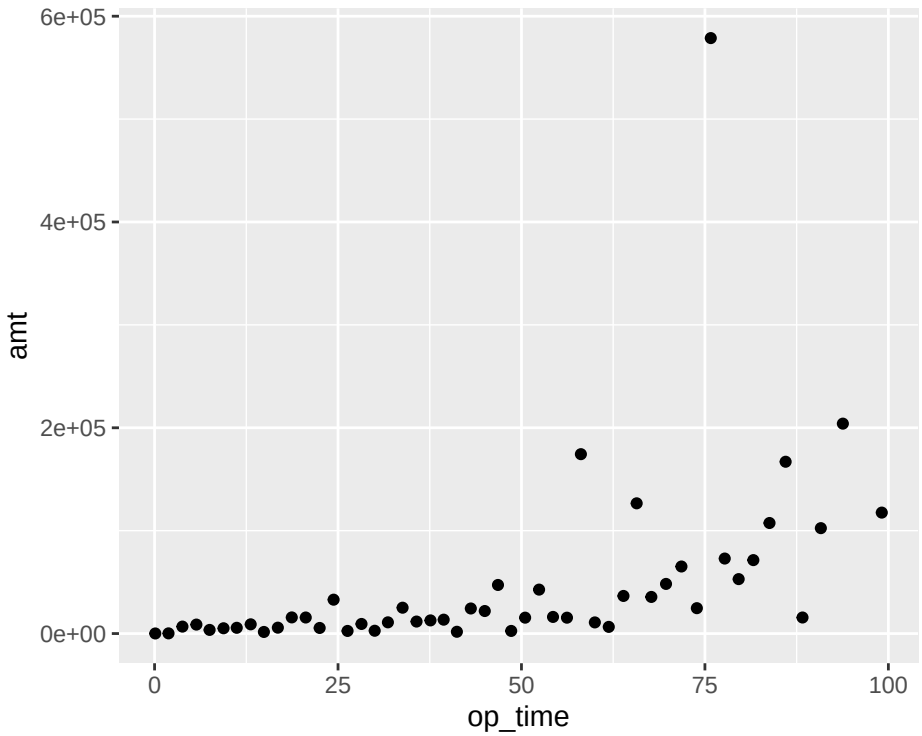

Figure 2.1.1: A basic scatterplot of `amt` against `op_time` in the `persinj50` dataset.

Using aesthetics the right way: The essence of aesthetic mappings. One of the most common ways to modify the appearance of a plot is to color the observations in order to produce a more impressive visual effect. To color the data points say, in blue, you may be tempted to make

use of the `color`[iii] aesthetic and simply insert `color = "blue"` as an additional argument to the `aes()` function, as in CHUNK 4. Doing so will produce unexpected and undesirable results as shown in Figure 2.1.2. To your astonishment, all of the data points are colored in red instead of blue and there is a legend saying "blue." What has gone awry here?

```
# CHUNK 4
# It is OK to suppress the names of the data and mapping arguments
# so long as they are supplied in order
ggplot(persinj50, aes(x = op_time, y = amt, color = "blue")) +
  geom_point()
```

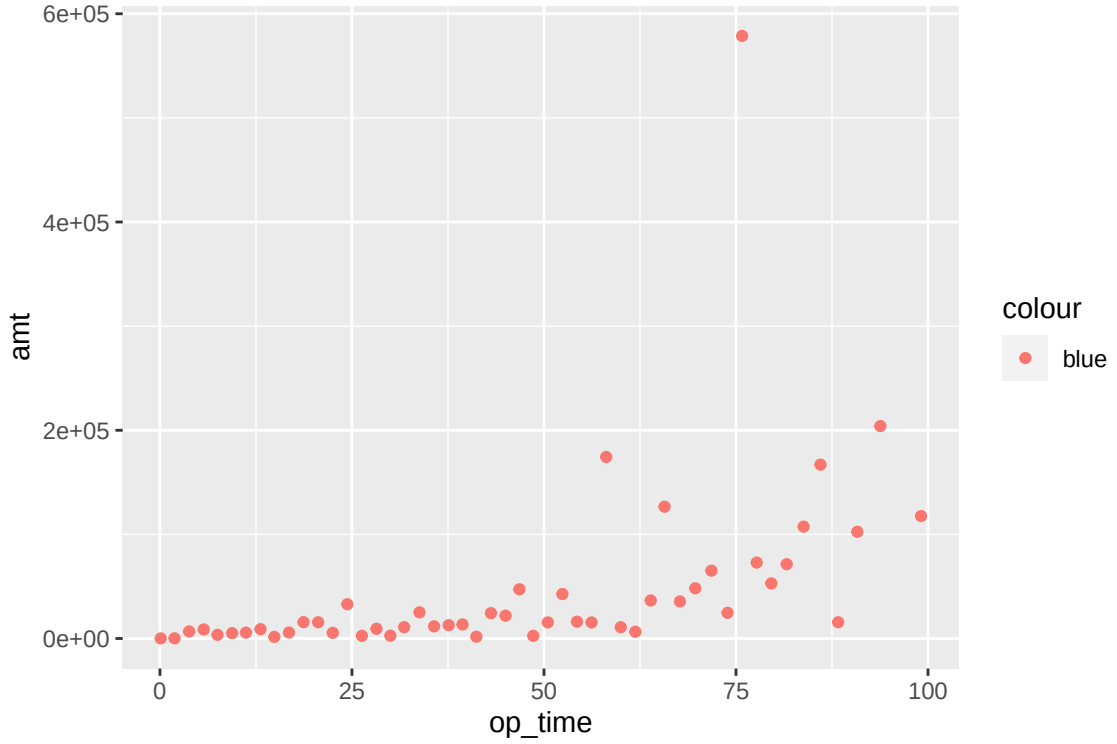

Figure 2.1.2: A version of Figure 2.1.1 with all of the points inadvertently colored in red.

Bear in mind that an aesthetic is a mapping between variables in our data and visual properties of the graph. The use of `color = "blue"` instructs the `aes()` function to map the `color` aesthetic to a variable named `"blue"` in the `persinj50` dataset. There is no such variable in our data, but the `aes()` function will do its best by treating `"blue"` as if it were a variable. The effect is the creation of a new character variable behind the scenes taking one and only one value, `"blue"`. As all observations in the data share the same `"blue"` value, all of them will be mapped to the same color. In `ggplot2`, the default first-category hue is red (not blue!). This explains why every point in the scatterplot becomes red in color.

To do the coloring the right way, we should realize that making all the points blue in color does *not involve any mapping* between variables in our data and the `color` aesthetic. After all, all the observations are colored in blue and are not distinguished on the basis of color. As a result, we should not put `color = "blue"` inside the `aes()` function. It should instead be placed inside the

[iii] Both the American spelling (color) and British spelling (colour) are accepted.

`geom_point()` function to modify the color of the plotted points. Figure 2.1.3 shows the desired scatterplot using the code in CHUNK 5. To our liking, all of the points are colored in blue.

```
# CHUNK 5
ggplot(persinj50, aes(x = op_time, y = amt)) +
  geom_point(color = "blue")
```

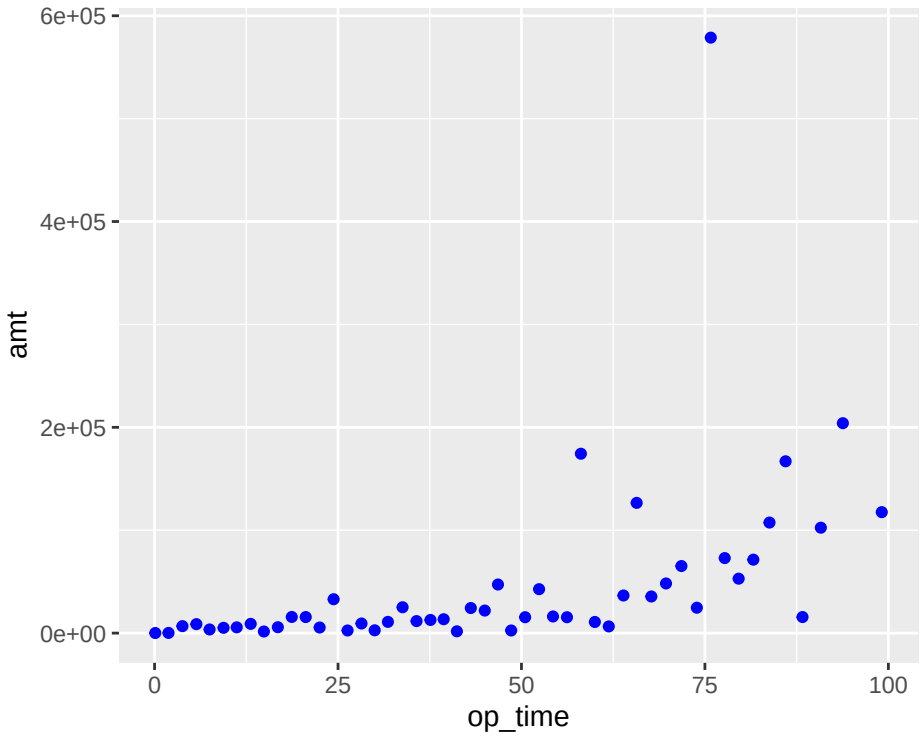

Figure 2.1.3: A version of Figure 2.1.1 with all of the points correctly colored in blue.

Figures 2.1.2 and 2.1.3 show what may go wrong if we fail to grasp the distinction between setting an aesthetic to a constant value (CHUNK 4) and mapping it to a variable (CHUNK 5). They highlight a subtle but extremely important mentality when making ggplots:

The `aes()` function is reserved for mappings between aesthetics and *variables*.

To set a property that affects how a plot looks but does not involve mapping variables to aesthetic elements, we should do it outside the `aes()` function—in the geom functions. To put it another way, the aesthetics determine *what* relationships we want to see in the plot whereas the geoms determine *how* we want to see the relationships.

Now let's see an example of using the `color` aesthetic correctly. In the `persinj50` data, the `legrep` variable is a binary variable equal to 1 for injuries with legal representation and 0 for those without. To color the different injuries according to the presence of legal representation, we map the `color` aesthetic to `legrep` treated as a factor (recall that factors are discussed on page 19). The resulting scatterplot, generated by the code in CHUNK 6, is given in Figure 2.1.4, where injuries without legal representation (`legrep = 0`) are displayed in red whereas those with legal representation (`legrep = 1`) are displayed in green. A legend is produced accordingly. Notice that there is a genuine mapping between the `legrep` variable and `color`, with `legrep = 0` mapped to the red color and `legrep = 1` mapped to the green color. In other words, the observations are


```
# CHUNK 6
ggplot(persinj50, aes(x = op_time, y = amt, color = factor(legrep))) +
  geom_point()
```

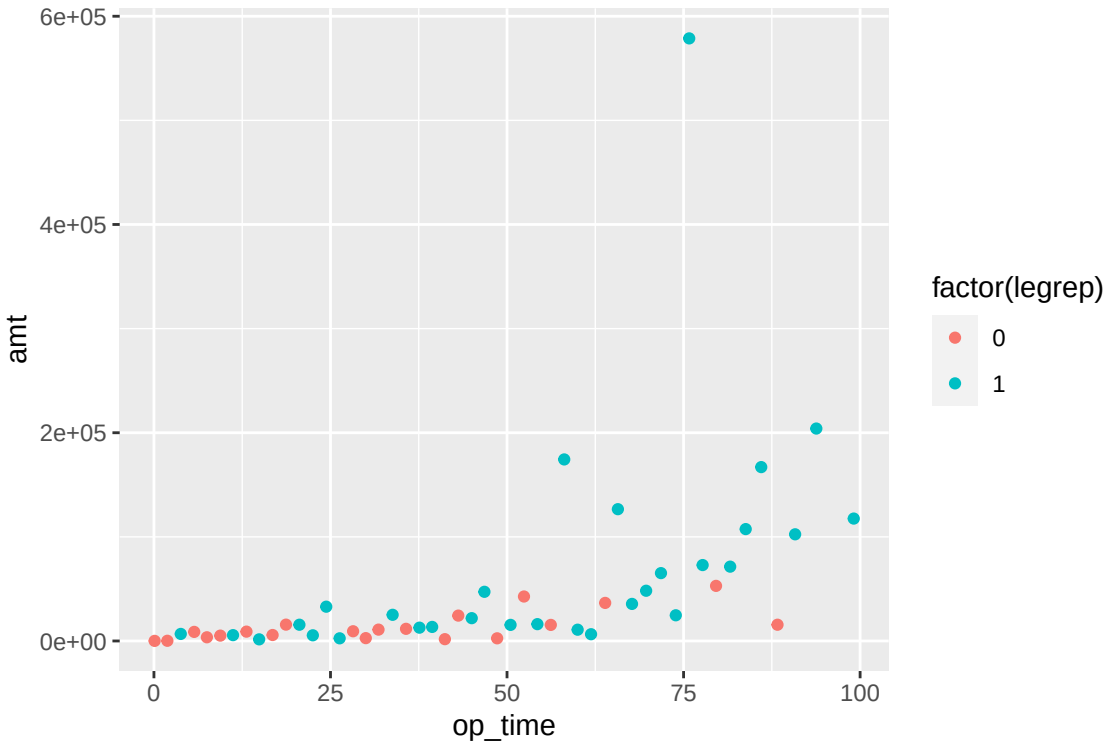

Figure 2.1.4: A version of Figure 2.1.1 with the observations distinguished by `legrep`.

differentiated on the basis of the `legrep` variable by color. (The `color` aesthetic does not say what colors are used to discriminate injuries on the basis of `legrep`, though.)

Exercise 2.1.1. (Why do we need to convert `legrep` to a factor?) To see why the conversion of `legrep` to a factor variable is needed, run the following code in CHUNK 7:

```
# CHUNK 7
ggplot(persinj50, aes(x = op_time, y = amt, color = legrep)) +
  geom_point()
```

What do you notice? Why is the coloring done in the way you observe?

Solution. Running the code (try it in RStudio!) produces a color gradient from 0 to 1. This is because the `legrep` variable in the `persinj50` dataset is treated by design as a continuous numeric variable even though the two levels, 0 and 1, are merely class labels that do not convey any sense of numeric order. R implicitly allows for values between 0 and 1 for the `legrep` variable and therefore uses the color gradient to differentiate the observations by color (though you can observe that there are only two colors in the plot, corresponding to the two extremes in the color gradient). This explains the point made in Subsection 1.2.1 that whether a categorical variable is treated as a factor can affect how the resulting graphical output looks. □

Some important geoms for Exam PA. Besides `geom_point()`, there are a number of geoms that are important for Exam PA. They are listed in Table 2.2 along with their commonly used arguments which affect how the plot looks.

Geom	Type of Object Produced	Frequently Used Arguments
`geom_bar()`	Bar chart	`fill`, `alpha`
`geom_boxplot()`	Boxplot	`fill`, `alpha`
`geom_histogram()`	Histogram	`fill`, `alpha`, `bins`
`geom_point()`	Scatterplot	`color`, `alpha`, `shape`, `size`
`geom_smooth()`	Smoothed curve	`color`, `fill`, `method`, `se`

Table 2.2: Some commonly used geoms in Exam PA.

The names of these geoms are pretty self-explanatory. For example, `geom_smooth()`, as its name suggests, produces a "smoothed" curve and, by default, produces a ribbon around the curve showing the standard error bands. It is typically used in conjunction with `geom_point()`. The smoothed curve can be generated by different statistical methods, such as a linear fit (by setting `method = "lm"`). The default is the use of nonparametric smoothing methods (`method = "gam"` or `method = "loess"`), which are beyond the syllabus of Exam PA. To switch off the standard error bands, you can set `se = FALSE`.

We will illustrate the use of `geom_bar()`, `geom_boxplot()`, and `geom_histogram()` in Section 2.2. For now, let's continue with the scatterplots we have just produced and make them more fancy and informative. In Figure 2.1.5, we plot the 50 observations in the `persinj50` dataset using large points (`size = 2`) and a small amount of transparency (`alpha = 0.5`), classify them according to whether they have legal representation or not, and fit a separate smoothed curve to each kind of injuries via the `geom_smooth()` function. The commands are collected in CHUNK 8.

Note that:

- For each of the two types of injuries, the smoothed curve and the standard error ribbon are indicated by the same color (red for those without legal representation and green for those with legal representation), which is appealing from an aesthetic perspective. The consistent coloring is achieved by mapping both the `color` aesthetic and `fill` aesthetic (which controls filled areas of bars, polygons and, in this case, the interior of standard error bands) to the `legrep` variable treated as a factor variable.

- If you omit `fill = factor(legrep)` (try this in R!), then the two smoothed curves will still be colored according to the presence of legal representation due to the `color` aesthetic, but without the `fill` aesthetic, the `geom_smooth()` function will shade the two standard error ribbons by its default color, which is gray.

- The `alpha` argument controls the transparency of the plotted objects on a scale from 0 (fully transparent) to 1 (opaque); the default value is 1. The transparency of the objects increases by decreasing `alpha`; the lower the value of `alpha`, the more transparent the points. In the limit when `alpha` is exactly zero, the objects become completely invisible. The `alpha` argument is particularly useful when there is a lot of overlapping among the data points. By setting alpha to an intermediate value such as 0.5, we make it easy to see where most of the observations cluster.


```
# CHUNK 8
ggplot(persinj50, aes(x = op_time, y = amt, color = factor(legrep), fill = factor(legrep)))
  geom_point(size = 2, alpha = 0.5) +
  geom_smooth()
```

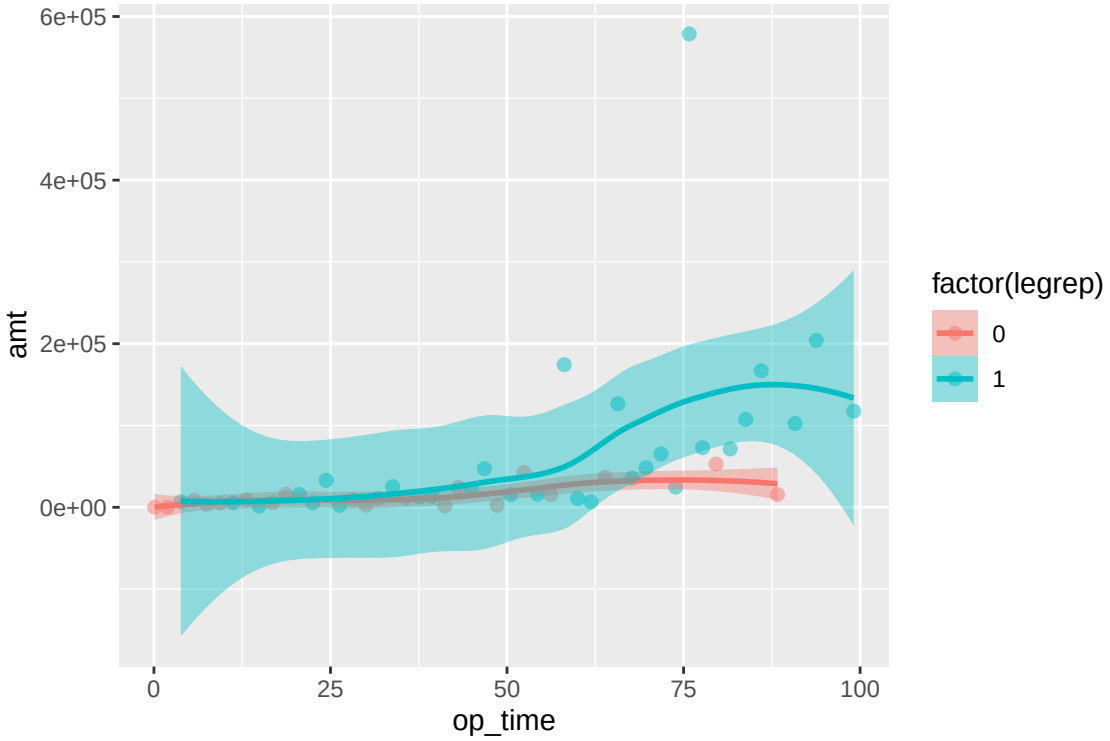

Figure 2.1.5: A version of Figure 2.1.4 with the data points enlarged and the standard error bands added.

Thanks to Figure 2.1.5, we can see that the claim amount of the two types of injuries behaves quite differently with respect to `op_time`, with those for legal representation being much more sensitive to changes in `op_time`. In Section 4.2, we will quantify the difference between the two forms of behavior formally. Figure 2.1.5, produced by `ggplot2`, allows us to discover such a phenomenon in the first place and is a very useful starting point for such an investigation.

Geom-specific aesthetics. What if you want to make just one smoothed curve applied to all 50 observations in the `persinj50` dataset while still having them colored according to the presence of legal representation? We can do so by specifying different aesthetic mappings for different geoms as in CHUNK 9. Notice that the `aes()` function in the `ggplot()` call only has the `x` and `y` aesthetics; the `color` aesthetic is moved to the `geom_point()` function. As a result, the 50 points will have their color distinguished by the `legrep` variable. However, since there is no such mapping in the `geom_smooth()` function, a single smoothed curve (colored in blue by default) fitted to all of the 50 observations surrounded by two standard error bands (colored in gray by default) will be produced as shown in Figure 2.1.6. In general, aesthetic mappings common to most, if not all, geoms can be specified in the initial `ggplot()` call. These mappings will be inherited by all geoms. If needed, you can then put in additional aesthetics that apply only to a particular geom to override the default aesthetics.


```
# CHUNK 9
ggplot(persinj50, aes(x = op_time, y = amt)) +
  geom_point(aes(color = factor(legrep)), size = 2, alpha = 0.5) +
  geom_smooth()
```

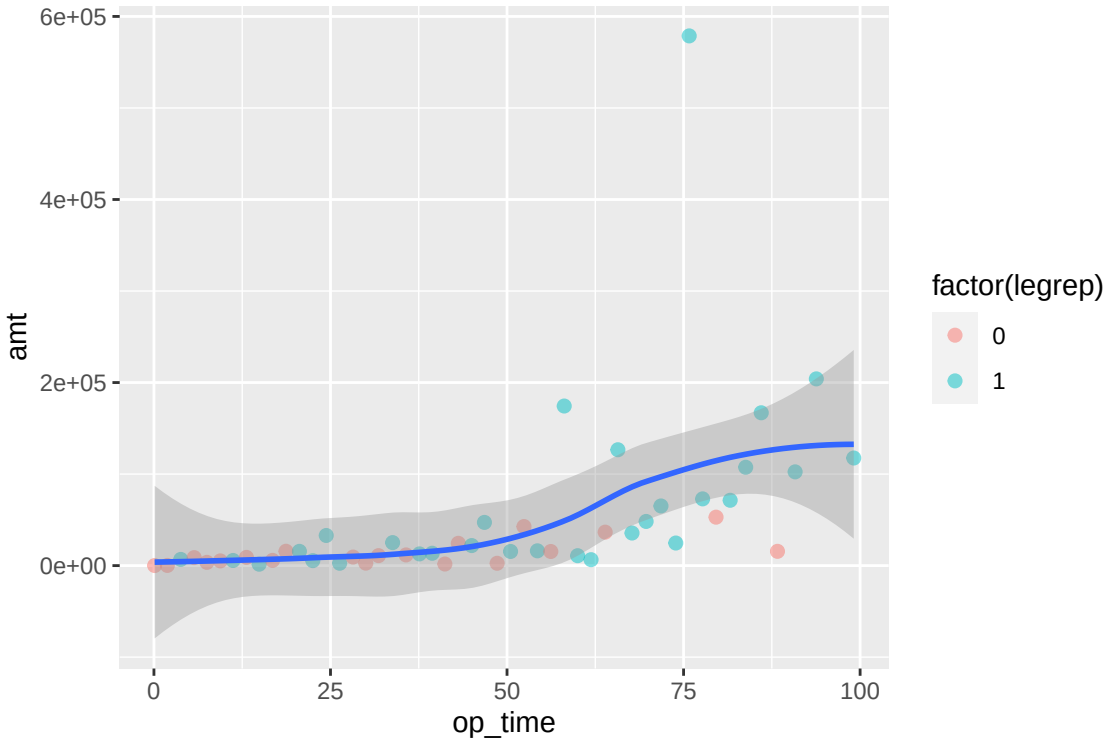

Figure 2.1.6: A variation of Figure 2.1.5 with a single standard error band.

Exercise 2.1.2. (Variations of CHUNK 9) Consider the following variations of CHUNK 9. Think about what kind of plots will be produced. Then run the code in the Rmd file and see what happens.

```
# CHUNK 10
ggplot(persinj50, aes(x = op_time, y = amt, fill = factor(legrep))) +
  geom_point(aes(color = factor(legrep)), size = 2, alpha = 0.5) +
  geom_smooth(se = FALSE)
```

```
# CHUNK 11
ggplot(persinj50, aes(x = op_time, y = amt)) +
  geom_point(aes(color = factor(legrep)), size = 2, alpha = 0.5) +
  geom_smooth(aes(fill = factor(legrep)))
```

Solution. Let's look at the two chunks separately.

ACTEX Learning

Learn Today. Lead Tomorrow.

You have reached the end of the Sample for the SOA Exam PA

Ready to view more?

ACTEX Learning has provided students and professionals with quality educational resources since 1972. Each study manual is carefully constructed to make learning actuarial content easy and painless. ACTEX also offers a variety of add-on material and study programs to help get you to your next destination.

What products accompany the ACTEX PA study manual?
Currently, ACTEX is offering **over 50 Instructional videos** to supplement the core of the manual (Parts I and II, or Chapters 1 to 6) as an add-on purchase. Add value to your learning experience with Professor Lo's step-by-step instruction on the fundamental concepts in predictive analytics, constructing predictive models in R, and a heavy emphasis on key test items for Exam PA.
Videos are not sold separately.

Take the next step in your career **now** by purchasing the full PA Study Manual.