Week 3 Module 6: Stripcharts

CSCI E-5a: Programming in R

Let's clear the global computing environment:

```
rm( list = ls() )
```

Module Overview and Learning Objectives

Hello! And welcome to Module 6: Stripcharts.

In this module, we'll explore a fascinating and under-appreciated visualization method known as a *stripchart*.

- In section 1, we'll define the stripchart, learn how to make a basic stripchart, and then see how to modify the display.
- In section 2, we'll use stripcharts to visualize data on the planets of the solar system.
- In section 3, we'll learn about the ingenious concept of "jitter".

When you've completed this module, you'll be able to:

- Construct a basic stripchart for a numeric vector.
- Modify the display of the stripchart.
- Visualize planetary data.
- Explain the method of jitter, and be able to apply it to a stripchart.

There are three new built-in R functions in this module:

- `stripchart()`
- `log()`
- `log10()`

All right! Let's start by defining the concept of a stripchart.

Section 1: Stripchart Basics

Main Idea: *We can define the concept of a stripchart*

In this section, we'll define the stripchart, learn how to make a basic stripchart, and then see how to modify the display.

The *stripchart* is a method for visualizing a one-dimensional range of numeric values.

The stripchart seems to be a neglected technique.

I had never seen a stripchart before I started working with R.

I really love stripcharts, and I think they are a powerful visualization tool.

Let's start with a simple example.

Suppose we have a numeric vector, and we want to visualize the values in it:

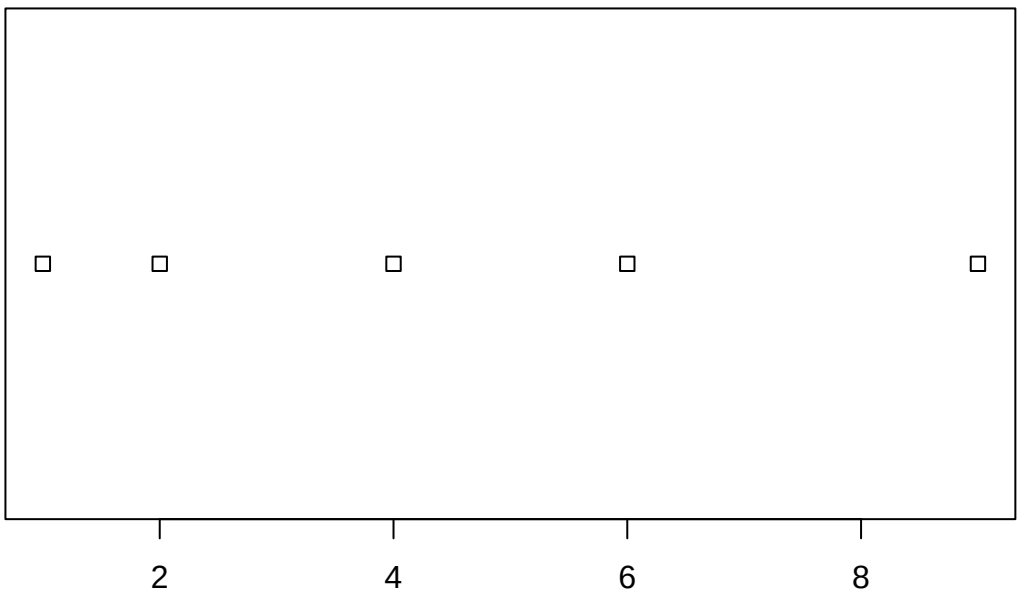
```
very.simple.numeric.vector <-
    c( 1, 2, 4, 6, 9)
```

The stripchart will plot these points so that the x-coordinate of the point is the value in the vector, while the y-coordinate will be the value 1.

So, for the moment, every point will have the same y-coordinate, which will be 1.

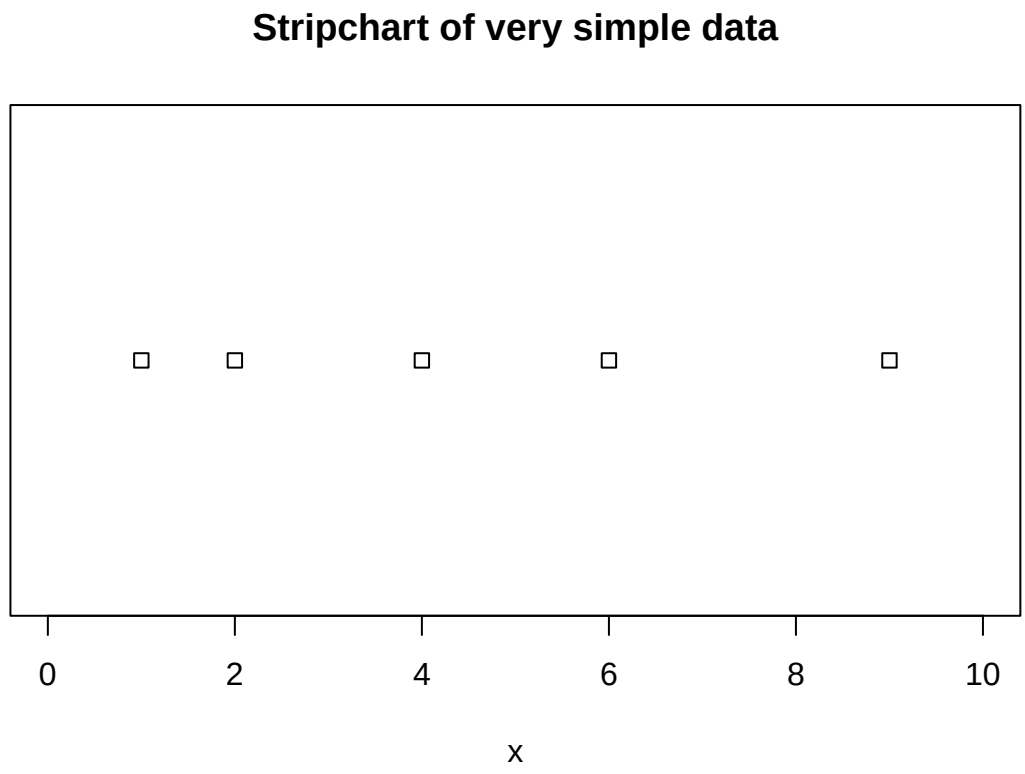
That means that the points will be plotted along a horizontal line.

```
stripchart( very.simple.numeric.vector )
```



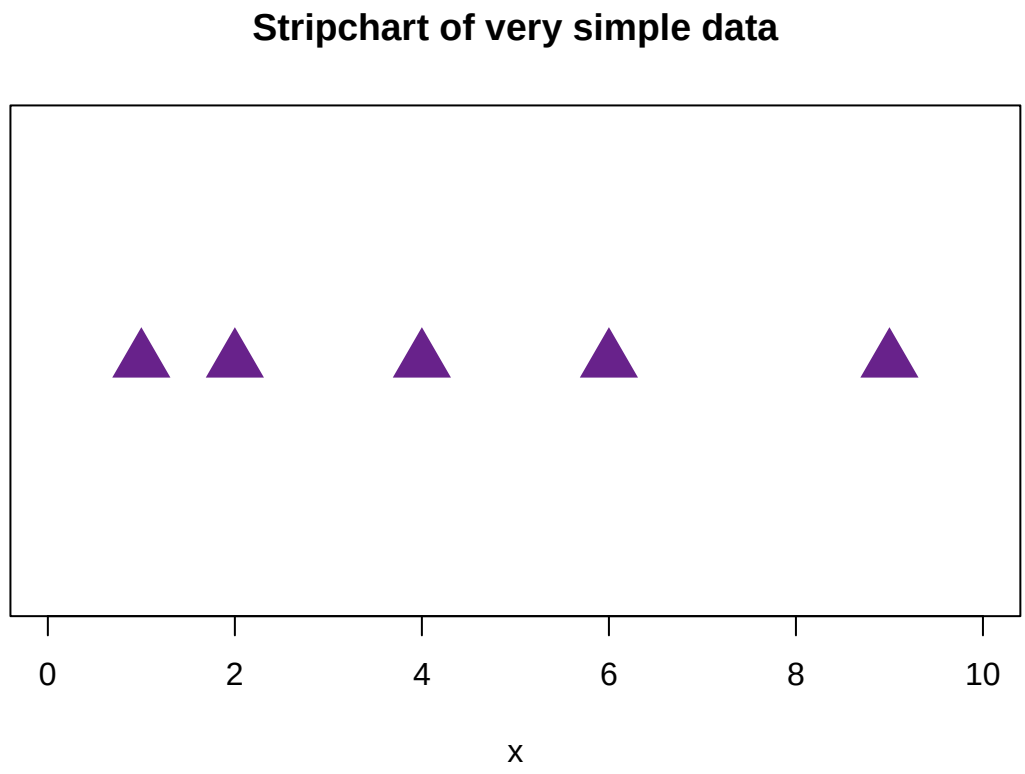
Let's adjust the range of the x-axis, give it a title, and give the graph a main title:

```
stripchart(
    very.simple.numeric.vector,
    xlim = c(0, 10),
    main = "Stripchart of very simple data",
    xlab = "x"
)
```



Now let's do something about the points in the graph.

```
stripchart(
    very.simple.numeric.vector,
    xlim = c(0, 10),
    main = "Stripchart of very simple data",
    xlab = "x",
    pch = 17,
    cex = 3,
    col = "darkorchid4"
)
```



Already this looks a lot better.

So that's the basic concept of a stripchart.

Now let's explore a case study on visualizing planetary data.

Exercise 6.1: Basic stripchart

Make a stripchart for these values:

$$12.7,\ 23.8,\ 17.6,\ 45.2,\ 31.1,\ 26.9,\ 15.2$$

Make it look nice!

Solution

Section 2: Case Study: Visualizing Planetary Data

Main Idea: *We can use stripcharts to visualize planetary data*

In this section, we'll use stripcharts to visualize data on the planets of the solar system.

Here's a table of the distance from the sun, measured in meters, for the planets in the solar system:

Planet	Distance from Sun
Mercury	5.79 * 10^10
Venus	1.08 * 10^11
Earth	1.50 * 10^11
Mars	2.28 * 10^11
Jupiter	7.78 * 10^11
Saturn	1.43 * 10^12
Uranus	2.87 * 10^12
Neptune	4.50 * 10^12

Let's try to visualize the distances by using a stripchart.

First we'll construct a named vector with the distances from the sun:

```
distance.from.sun.vector <-
    c(
        "Mercury" = 5.79e10,
        "Venus" = 1.08e11,
        "Earth" = 1.50e11,
        "Mars" = 2.28e11,
        "Jupiter" = 7.78e11,
        "Saturn" = 1.43e12,
        "Uranus" = 2.87e12,
        "Neptune" = 4.50e12
    )
```

Notice that we're using scientific notation here to represent the numeric values because they are so large.

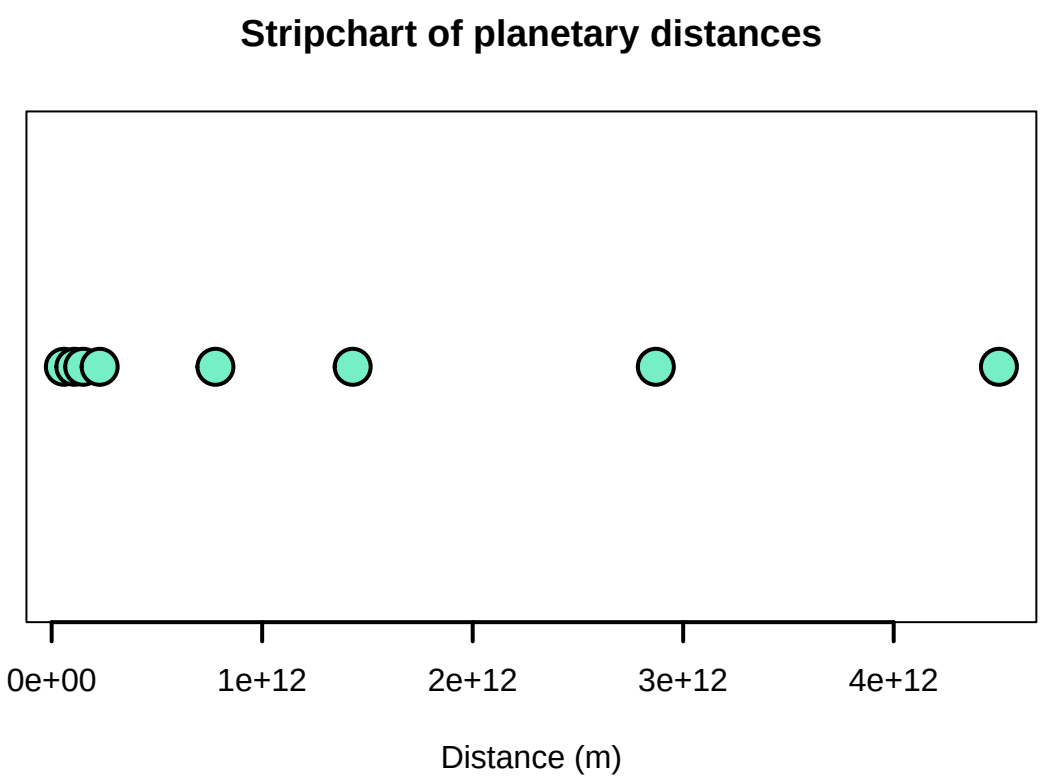
Let's directly display this vector:

```
distance.from.sun.vector
```

```
##  Mercury    Venus    Earth     Mars  Jupiter   Saturn   Uranus  Neptune
## 5.79e+10 1.08e+11 1.50e+11 2.28e+11 7.78e+11 1.43e+12 2.87e+12 4.50e+12
```

Now let's make a stripchart of these distances:

```
stripchart(
    x = distance.from.sun.vector,
    main = "Stripchart of planetary distances",
    xlab = "Distance (m)",
    pch = 21,
    lwd = 2,
    cex = 2.5,
    col = "black",
    bg = "aquamarine2"
)
```



This might look a little odd – why are all those points clustered on the left-hand side.

The answer is that the most distant planet, Neptune, is very far away from the sun.

Let's use `cat()` and `formatC()` functions to print out the distance of Neptune from the sun:

```
cat(
    "Distance of Neptune from the sun in meters:",
    formatC(
        distance.from.sun.vector[ "Neptune" ],
        format = "f",
        big.mark = ",",
        digits = 0
    )
)
```

```
## Distance of Neptune from the sun in meters: 4,500,000,000,000
```

Wow!! That's a lot of zeros.

The distance of Neptune from the sun is about 78 times greater than the distance of Mercury from the sun:

```
neptune.mercury.distance.ratio <-
    distance.from.sun.vector[ "Neptune"] /
    distance.from.sun.vector[ "Mercury" ]

cat(
```

```
    "Neptune / Mercury distance ratio:",
    formatC(
        neptune.mercury.distance.ratio,
        format = "f",
        digits = 1
    )
)
```

```
## Neptune / Mercury distance ratio: 77.7
```

Thus, in order to display the stripchart and include Neptune, the scale of the x-axis has to be so large that the point representing Mercury is all the way over at the left.

The same is true for the points representing Venus, Earth, and Mars.

Looking at the stripchart, you can really see how far away from the sun planets such as Uranus and Neptune are, even relative to the other "outer" planets Jupiter and Saturn.

when you see artist's depictions of the solar system, they have to draw them with unrealistic distances in order to make a visually intelligible display.

The stripchart allows us to appreciate the actual distances.

Sometimes it can be awkward to have to work with such large magnitudes as these distances.

One common way of handling this is to work with the logarithms of the distances.

In R, the `log()` function will compute natural logarithms, which are logarithms using the base e:

```
log( 100 )
```

```
## [1] 4.60517
```

For astronomical data, it's usually more convenient to work with base 10 logarithms, which are calculated by the function `log10()`:

```
log10( 100 )
```

```
## [1] 2
```

Then we can map this function over the values in the `distance.from.sun.vector`:

```
log.distance.from.sun.vector <-
    log10( distance.from.sun.vector )

log.distance.from.sun.vector
```

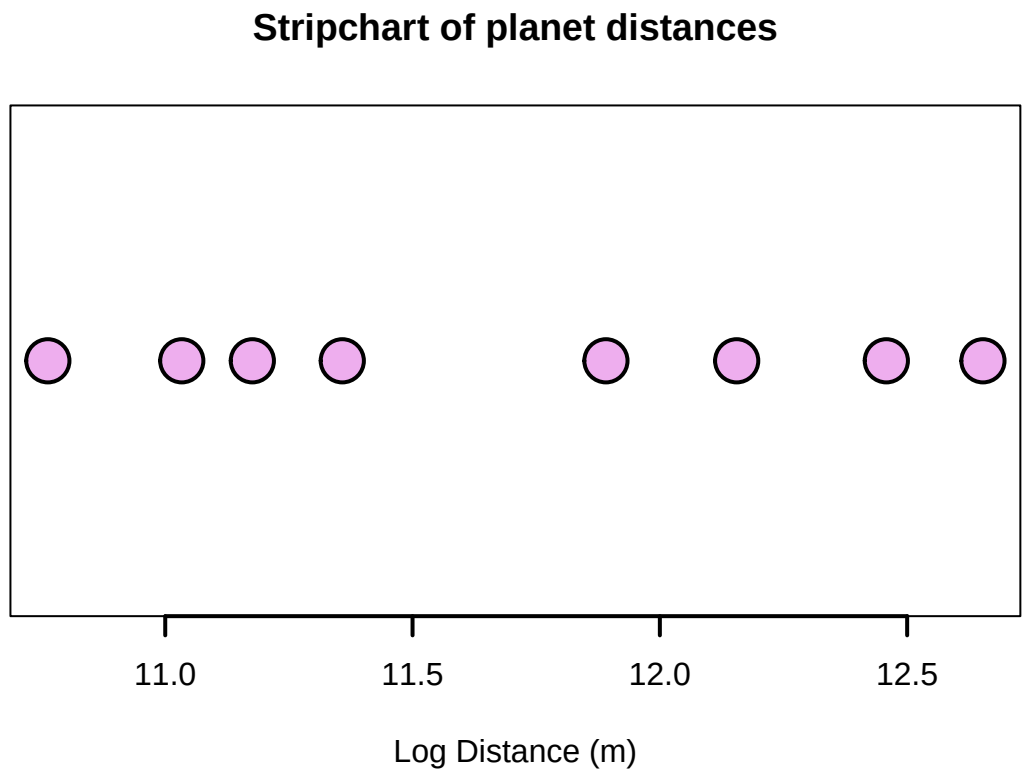
```
##  Mercury    Venus    Earth     Mars  Jupiter   Saturn   Uranus  Neptune
## 10.76268 11.03342 11.17609 11.35793 11.89098 12.15534 12.45788 12.65321
```

Don't worry if you don't remember much about logarithms – most people don't.

Just be aware that it's a numeric function.

Now let's make a stripchart with these log distances:

```
stripchart(
    x = log.distance.from.sun.vector,
    main = "Stripchart of planet distances",
    xlab = "Log Distance (m)",
    pch = 21,
    lwd = 2,
    cex = 3,
    col = "black",
    bg = "plum2"
)
```



Now the data is more spread out, and it's easier to see each individual point.

Of course, the price that we pay is that now we have to interpret the x-axis on a logarithmic scale.

Notice in this case study how many different R concepts we used:

- Named vectors
- Scientific notation
- The `cat()` and `formatC()` functions
- Character string indexing
- Variables
- The numeric functions `log()` and `log10()`

- Mapping a function over a vector
- Stripcharts

You can really see how already we can do some sophisticated work with the tools that we've developed in this course so far.

So that's how to visualize planetary data.

Now let's learn about the concept of jitter.

Exercise 6.2: Planetary data period data stripchart

Now let's repeat this analysis, but with data for planetary periods.

The *period* of a planetary orbit is just the amount of time that it takes the planet to complete one full orbit around the sun.

For instance, the planet Mercury completes an orbit around the sun in 87.97 (Earth) days, so the period is 0.241 years.

On the other hand, the planet Neptune has a period of 164.8 years.

By definition, Earth has a period of exactly 1 year.

Here's a table of the periods of the planetary orbits, measured in years, for the planets in the solar system:

Planet	Period
Venus	0.615
Earth	1.00
Mars	1.88
Jupiter	11.9
Saturn	29.5
Uranus	84.0
Neptune	164.8

First, create a named vector consisting of the planetary data for the periods of the planets.

Then construct a stripchart of this data.

Solution

Exercise 6.3: Planetary data log-period data stripchart

Construct a stripchart of the logarithms of the periods.

Solution

Section 3: Jitter

Main Idea: *We can use jitter to improve stripcharts*

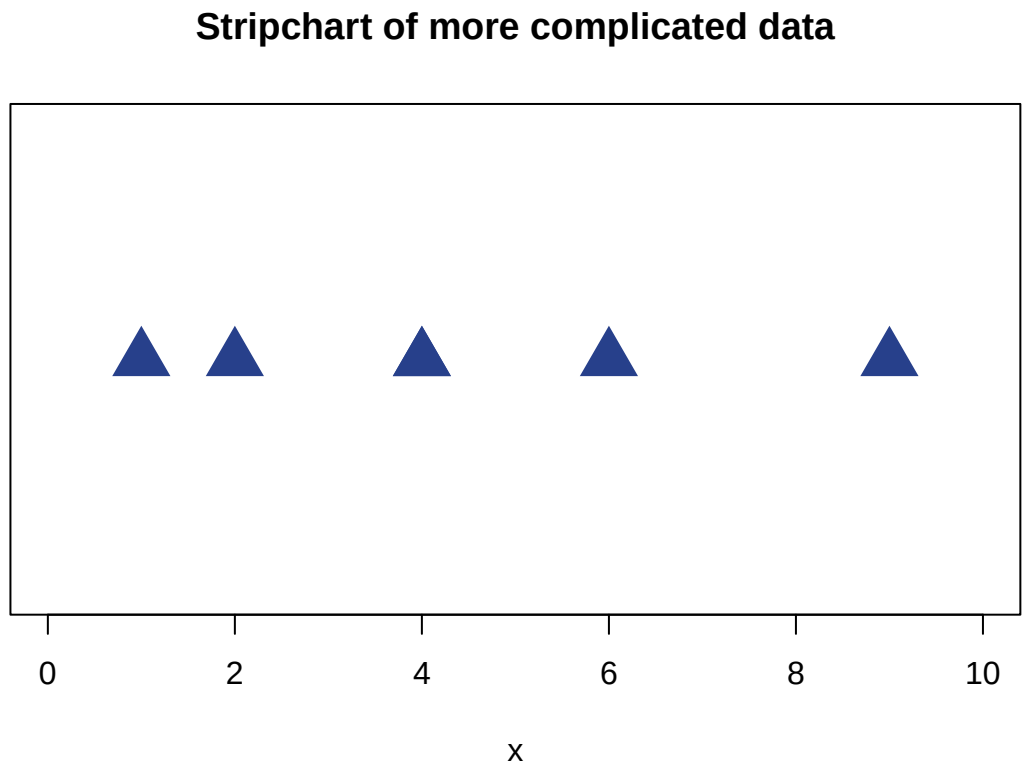
In this section, we'll learn about the ingenious concept of "jitter".

Now let's consider a more complicated case:

```
more.complicated.numeric.vector <-
    c( 1, 2, 4, 4, 6, 9)
```

We'll make the stripchart of this data just like before:

```
stripchart(
    x = more.complicated.numeric.vector,
    xlim = c(0, 10),
    main = "Stripchart of more complicated data",
    xlab = "x",
    pch = 17,
    cex = 3,
    col = "royalblue4"
)
```



Hey! Wait a second – the original data had six values, but the stripchart is only showing 5 points.

What happened to the missing point?

Before we go on, look back at the data and think about how the stripchart works, and try to figure out for yourself what happened to the lost point.

The answer here is that the point is not really "missing".

There were 2 elements of the `more.complicated.numeric.vector` that had the same value: 4.

Thus, the stripchart plotted both points, but they ended up in the same location, so we only see one shape.

How can we show that there are two elements that have the same value of 4?

This is where we can use an amazing feature of the stripchart, called "jitter".

Recall that we are plotting values from a one-dimensional numeric range, using the x-coordinate to display the values of the elements.

We don't use the y-coordinate for anything in this approach, and it contains no information.

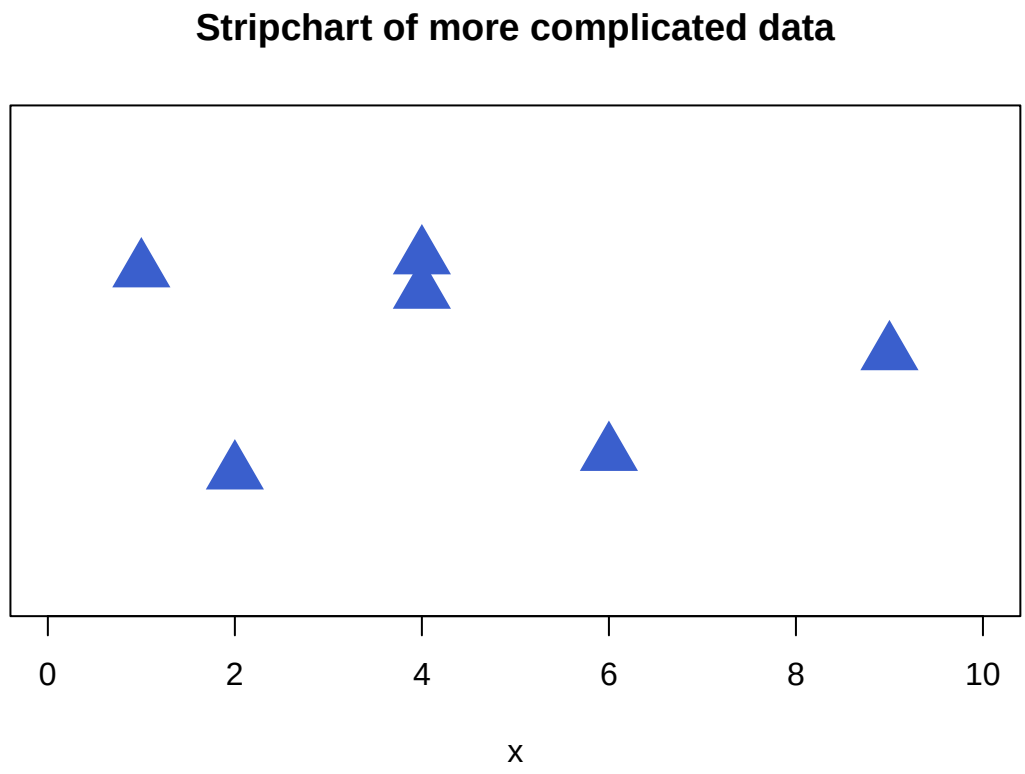
When we use this "jitter" method, R selects a random value for the y-coordinate of each point.

When we use this "jitter" method, we must set three options in the stripchart graph:

- First, we have to specify `method = "jitter"`.
- Second, we have to specify the amount of jitter using the `jitter` parameter.
- Third, for technical reasons we have to specify the y-axis as ranging from 0 to 2 by using the `ylim` option.

Here's the finished graph, with jitter:

```
stripchart(
    x = more.complicated.numeric.vector,
    xlim = c(0, 10),
    ylim = c(0, 2),
    main = "Stripchart of more complicated data",
    xlab = "x",
    method = "jitter",
    jitter = 0.5,
    pch = 17,
    cex = 3,
    col = "royalblue3"
)
```



Run the code chunk a few times, and notice that each time the vertical position of the points changes.

That's fine, because the y-coordinate doesn't contain any information.

The x-coordinate, however, does contain information, and if you run the code chunk a few times, you should notice that although the y-coordinate changes each time, the x-coordinate does not.

This is an amazing idea – that we can improve a data visualization by adding random noise to it.

Of course, the point is that the random noise is along the vertical dimension, and that's OK because the stripchart doesn't use the vertical dimension to represent information.

So that's the concept of jitter.

Now let's review what we've learned in this module.

Exercise 6.4: The `rivers` dataset

Let's see a more real-world application of a stripchart.

The `rivers` dataset is a built-in vector in R that contains the lengths of the 141 major rivers in North America in miles.

Let's check out the documentation on this dataset.

For this exercise, make stripchart of the `rivers` dataset.

You should put in all the fancy stuff: include a main title and an x-axis title, adjust the range of the x values on the horizontal axis, and make the points look nice.

Notice that there are now 141 points, so you'll have make some judgements about the size of the points.

Finally, include jitter:

- Remember that you have to explicitly set the values for the y-axis to range from 0 to 2.
- Include the option `method = "jitter"`.
- Set the jitter amount to give a visually pleasing range for the points.

Solution

Module Review

In this module, we explored a fascinating and under-appreciated visualization method known as a *stripchart*.

- In section 1, we defined the stripchart, learned how to make a basic stripchart, and then saw how to modify the display.
- In section 2, we used stripcharts to visualize data on the planets of the solar system.
- In section 3, we learned about the ingenious concept of "jitter".

Now that you've completed this module, you should be able to:

- Construct a basic stripchart for a numeric vector.
- Modify the display of the stripchart.
- Visualize planetary data.
- Explain the method of jitter, and be able to apply it to a stripchart.

There were three new built-in R functions in this module:

- `stripchart()`
- `log()`
- `log10()`

All right! That's it for Module 6: Stripcharts.

In fact, that's all the content for Week 3: Vectors.

Now you can finish Problem Set 3!

Solutions to the Exercises

Exercise 6.1: Basic stripchart

Make a stripchart for these values:

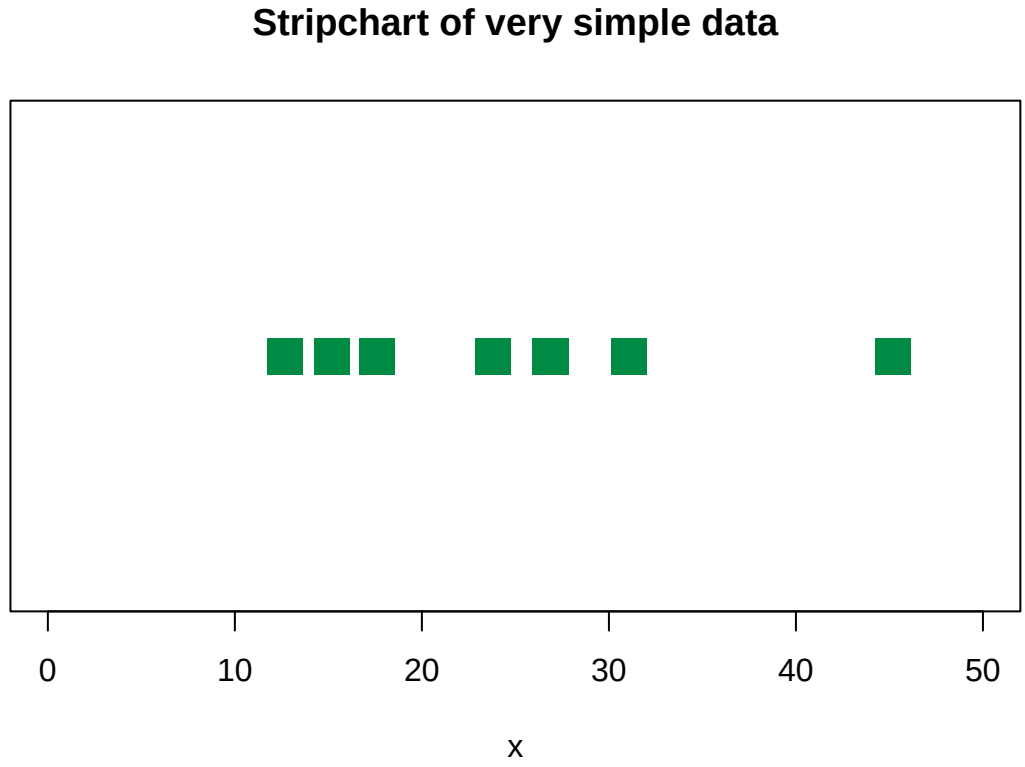
$$12.7, 23.8, 17.6, 45.2, 31.1, 26.9, 15.2$$

Make it look nice!

Solution

```r
exercise.6.1.numeric.vector <-
    c( 12.7, 23.8, 17.6, 45.2, 31.1, 26.9, 15.2 )

stripchart(
    exercise.6.1.numeric.vector,
    xlim = c(0, 50),
    main = "Stripchart of very simple data",
    xlab = "x",
    pch = 15,
    cex = 2.5,
    col = "springgreen4"
)
```



Exercise 6.2: Planetary data period data stripchart

Now let's repeat this analysis, but with data for planetary periods.

The *period* of a planetary orbit is just the amount of time that it takes the planet to complete one full orbit around the sun.

For instance, the planet Mercury completes an orbit around the sun in 87.97 (Earth) days, so the period is 0.241 years.

On the other hand, the planet Neptune has a period of 164.8 years.

By definition, Earth has a period of exactly 1 year.

Here's a table of the periods of the planetary orbits, measured in years, for the planets in the solar system:

Planet	Period
Venus	0.615
Earth	1.00
Mars	1.88
Jupiter	11.9
Saturn	29.5
Uranus	84.0
Neptune	164.8

First, create a named vector consisting of the planetary data for the periods of the planets.

Then construct a stripchart of this data.

Solution

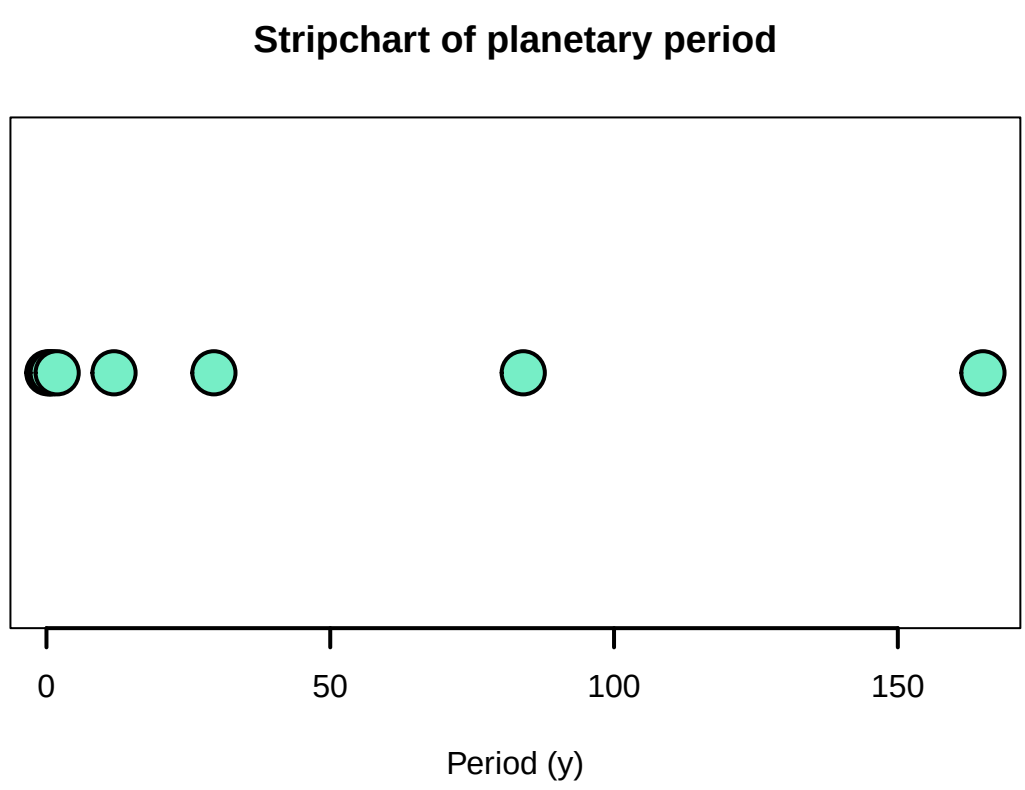
Exercise 6.3: Planetary data log-period data stripchart

Construct a stripchart of the logarithms of the periods.

Solution

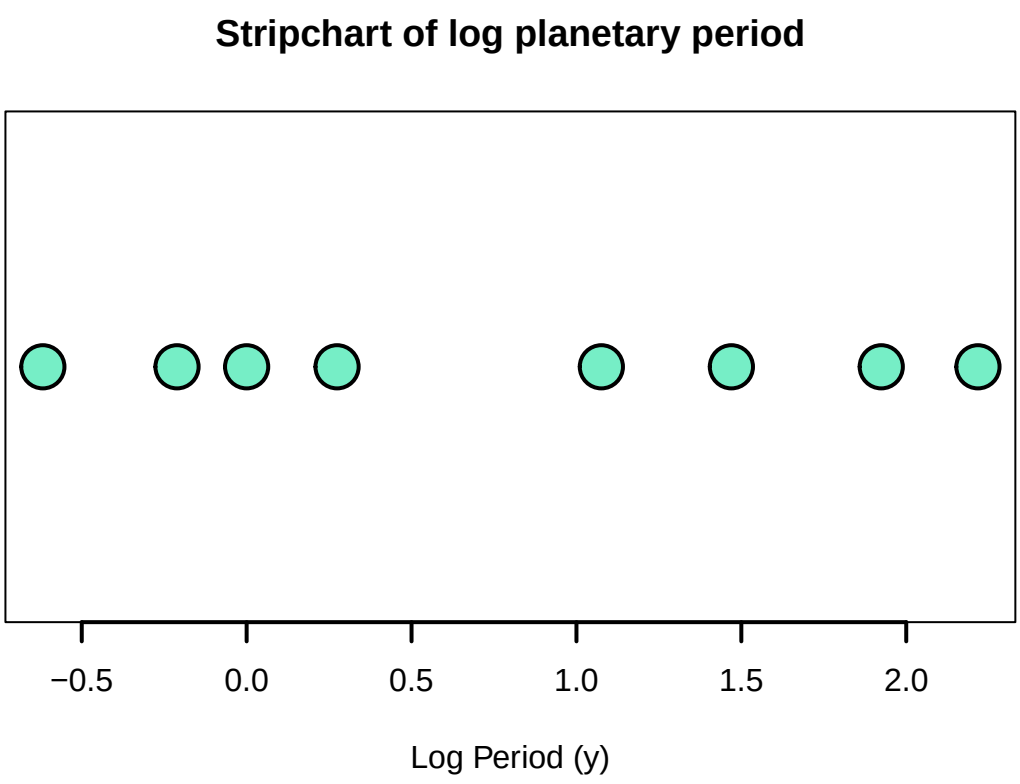
```
period.vector <-
    c(
        "Mercury" = 0.241,
        "Venus" = 0.615,
        "Earth" = 1.00,
        "Mars" = 1.88,
        "Jupiter" = 11.9,
        "Saturn" = 29.5,
        "Uranus" = 84.0,
        "Neptune" = 165
    )
```

```
stripchart(
    x = period.vector,
    main = "Stripchart of planetary period",
    xlab = "Period (y)",
    pch = 21,
    lwd = 2,
    cex = 3,
    col = "black",
    bg = "aquamarine2"
)
```




```
log.period.vector <-
    log10( period.vector )
```

```
stripchart(
    x = log.period.vector,
    main = "Stripchart of log planetary period",
    xlab = "Log Period (y)",
    pch = 21,
    lwd = 2,
    cex = 3,
    col = "black",
    bg = "aquamarine2"
)
```



Exercise 6.4: The `rivers` dataset

Let's see a more real-world application of a stripchart.

The `rivers` dataset is a built-in vector in R that contains the lengths of the 141 major rivers in North America in miles.

Let's check out the documentation on this dataset.

For this exercise, make stripchart of the `rivers` dataset.

You should put in all the fancy stuff: include a main title and an x-axis title, adjust the range of the x values on the horizontal axis, and make the points look nice.

Notice that there are now 141 points, so you'll have make some judgements about the size of the points.

Finally, include jitter:

- Remember that you have to explicitly set the values for the y-axis to range from 0 to 2.
- Include the option `method = "jitter"`.
- Set the jitter amount to give a visually pleasing range for the points.

Solution

```r
stripchart(
    rivers,
    xlim = c(0, 4000),
    ylim = c(0, 2),
    main = "Stripchart of the rivers dataset",
    xlab = "x",
    method = "jitter",
    jitter = 0.7,
    pch = 19,
    cex = 1.2,
    col = "darkseagreen4"
)
```