Chapter 7: Statistical Testing – Concepts and Strategy

Ram Gopal, Dan Philps, and Tillman Weyde

Summer 2022

Contents

Permutation Testing

The general approach to conduct permutation testing is:

1. Choose a statistic of interest (eg. Mean, Percentile, Correlation, etc)
2. Generate a hypothesis that enables you to describe the (hypothesized) population
3. Draw samples from the population and construct the sampling distribution
4. Locate the observed statistic on this distribution to draw your inference

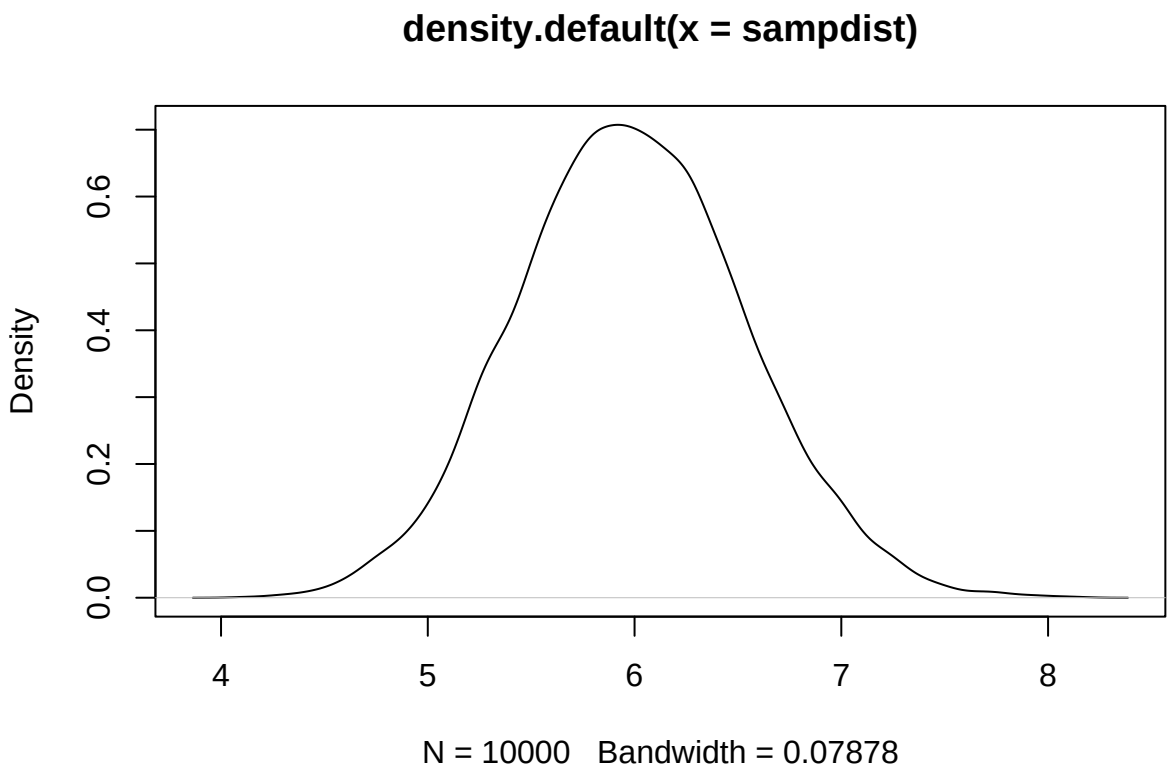
First Example

Suppose a device manufacturer comes up with a new design and claims that the number of complaints against the device per day will be no more than six. You want to test this claim.

Step 1: State the hypothesis.
H_0: the average number of complaints per day is less than or equal to six.

Step 2: Describe the data generation process and the population.
Since the number of complaints per day is a count variable, we will assume that the data generation process is Poisson. Based on this and the hypothesis, we will describe the population as $Pois(6)$.
Step 3: Create a sampling distribution.
Now that the population is defined, we can create the sampling distribution. To create the sampling distribution, we need the **statistic of interest** and the **sample size**. The statistic of interest is simply the mean. Let us say that the sample size we will collect will be 20. The following will create the sampling distribution.

```
set.seed(87654321)
f1 = function(){
  s1 = rpois(n = 20, lambda = 6)
  return(mean(s1))
}
sampdist = replicate(10000, f1())
plot(density(sampdist))
```



Just with a little bit of code, we are able to create the sampling distribution!

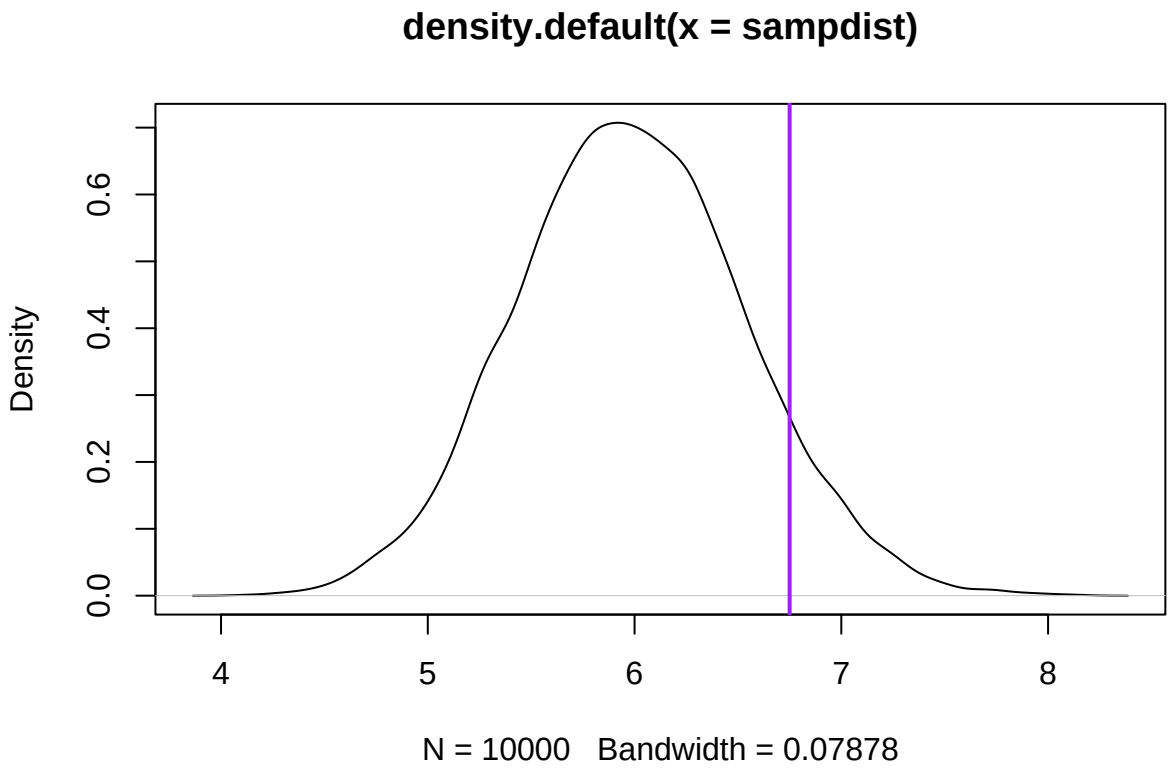
Step 4: Get the actual sample and compute the statistic.
Now we need to get a sample of 20 observations and compute the statistic for this sample. The statistic of interest here is the mean.

Suppose you have gone out and collected this data, and suppose this data is as follows.

Observed number of complaints = $(4, 3, 5, 13, 7, 10, 9, 9, 3, 6, 4, 3, 7, 10, 7, 6, 7, 8, 7, 7)$

Now we can compute the test statistic and plot it on the sampling distribution.

```r
svalues = c(4,3,5,13,7,10,9,9,3,6,4,3,7,10,7,6,7,8,7,7)
tstat = mean(svalues)
plot(density(sampdist))
abline(v = tstat, col = "purple", lwd = 2)
```




```r
print(tstat)
```

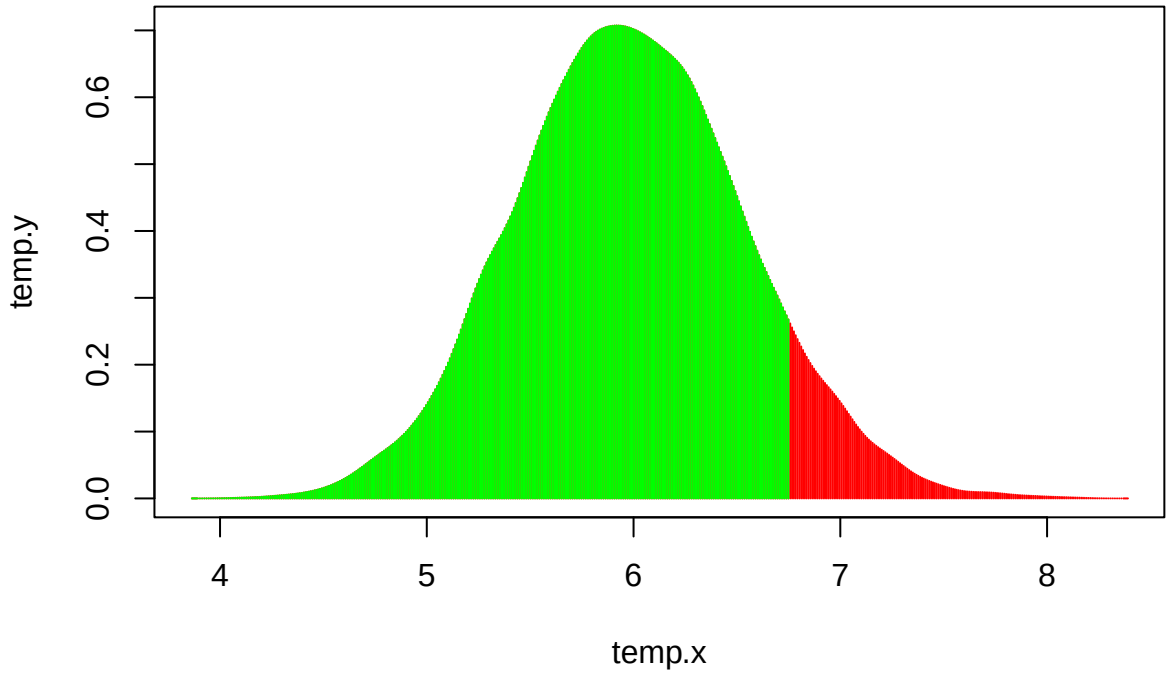
```
## [1] 6.75
```

Now what do we do? It looks like the sample value lies near the right tail, but we are not entirely sure what to make of it. This is where p-values come in very handy.

P-Value

Look at the plot above. All the points to the right of the purple line "go against" the hypothesis. The area to the right of the purple line represent samples that you would get where the mean of the sample is even higher than 6.75 complaints per day that we had with our sample. That area (to the right of the purple line) basically gives us reasons to not support the hypothesis. In other words, points to the left of the vertical line are supportive of the hypothesis compared to our sample, and the points to the right of the line are not supportive of the hypothesis compared to our sample. To the left is the "green" area, and to the right is the "red" area. To make this visually clear, let us color the two parts of the sampling distribution.

```
temp = density(sampdist)
df = data.frame(temp$x, temp$y)
formula1 = df$temp.x<tstat
df1 = df[formula1,]
plot(df, col = "red", type = "h")
points(df1, col = "green", type = "h")
```



p-value simply is the area of the red portion of the sampling distribution. Remember again, the red area represents samples that are even worse than the sample that we have.

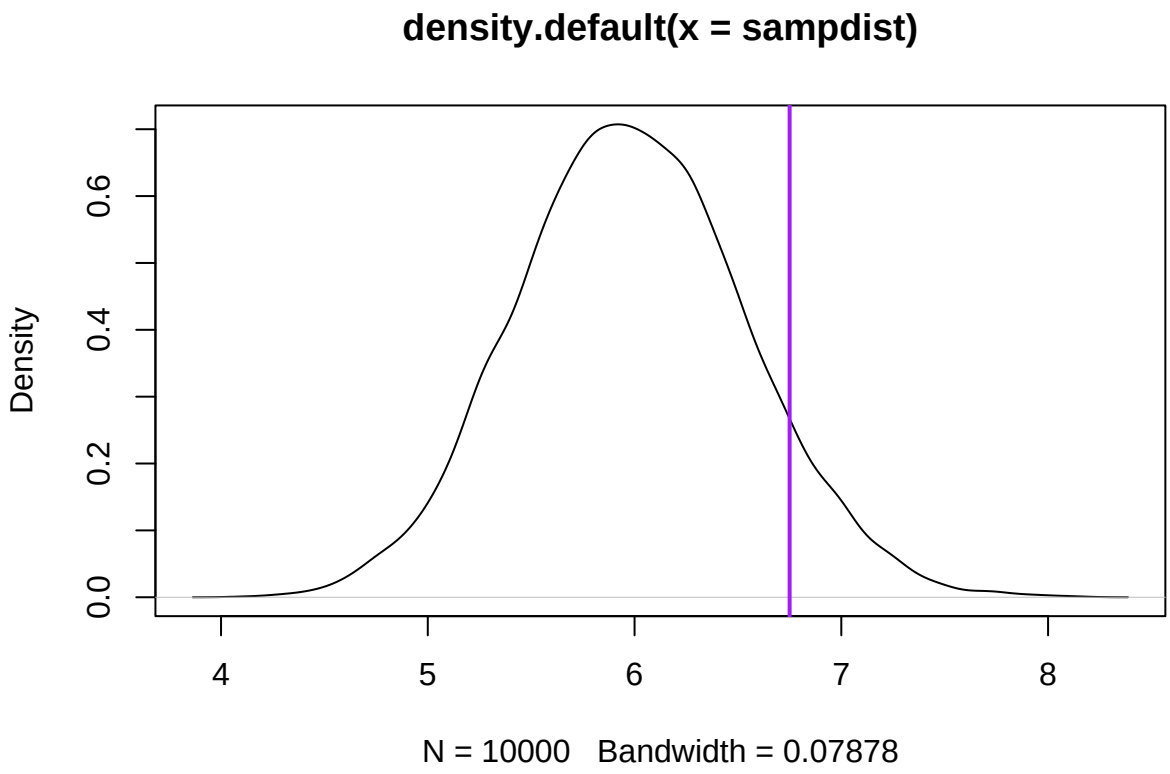
```
pvalue = length(sampdist[sampdist>tstat])/(length(sampdist))
print(pvalue)
```

```
## [1] 0.0783
```

Tale of Tails

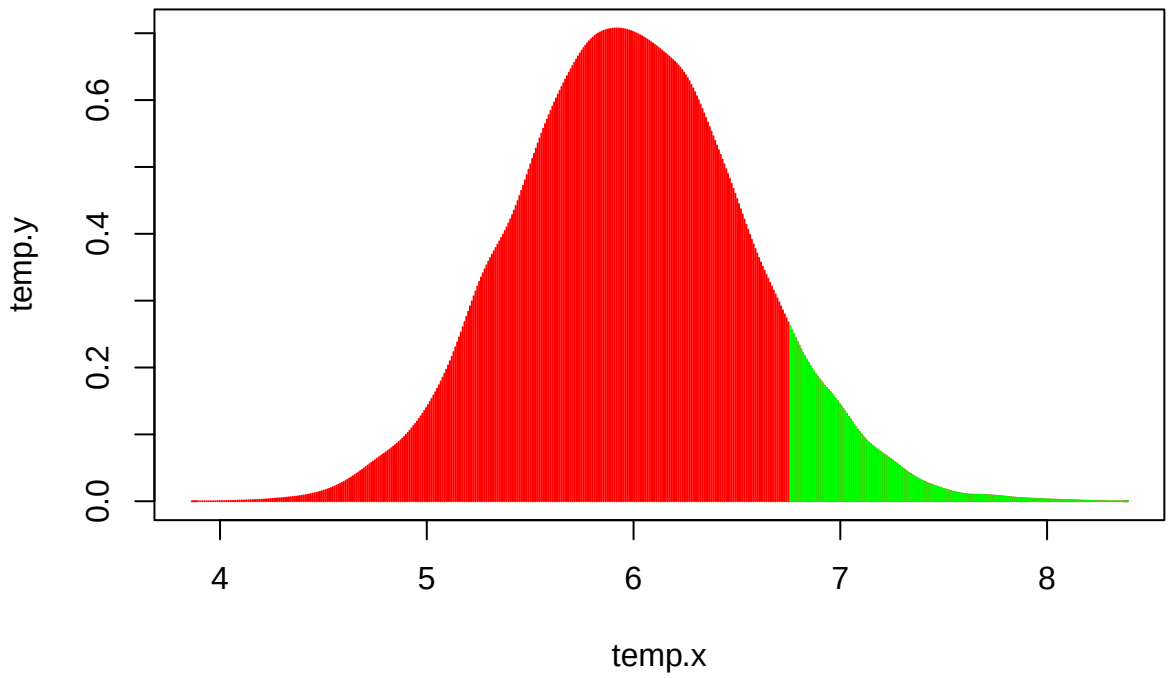
What if our hypothesis was that the mean number of complaints per day is six or more? In this case, we need to rethink the "red" and "green" areas of the density plot.

```
plot(density(sampdist))
abline(v = tstat, col = "purple", lwd = 2)
```



In this case, the region to the left of the purple line goes against the null hypothesis, and thus it should be colored red. This will obviously change the p-value as well. The code is:

```
temp = density(sampdist)
df = data.frame(temp$x, temp$y)
formula1 = df$temp.x>tstat
df1 = df[formula1,]
plot(df, col = "red", type = "h")
points(df1, col = "green", type = "h")
```



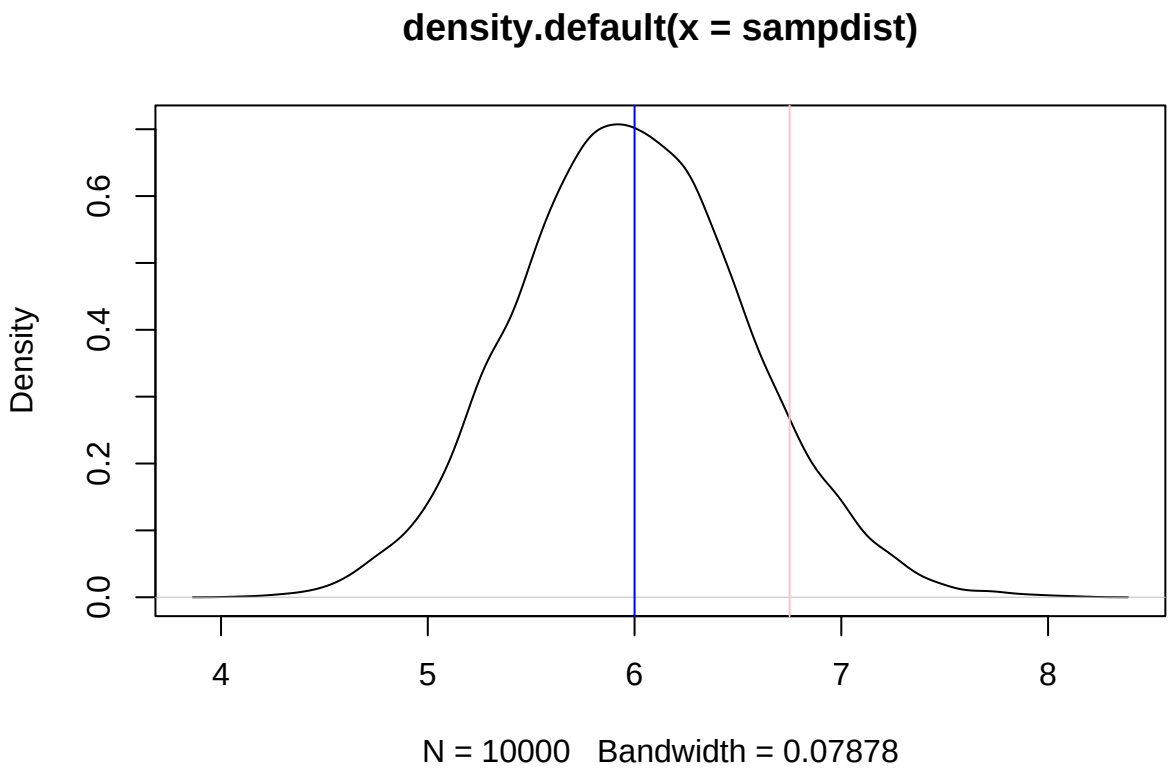

```
pvalue = length(sampdist[sampdist<tstat])/(length(sampdist))
print(pvalue)
```

```
## [1] 0.9079
```

Since the p value is much larger than 0.05, we are comfortable not rejecting the null hypothesis that the average number of complaints per day is larger than six. The above two tests are called **one-tailed tests**. This makes sense because you are looking at the tail end of the one side of the sampling distribution.

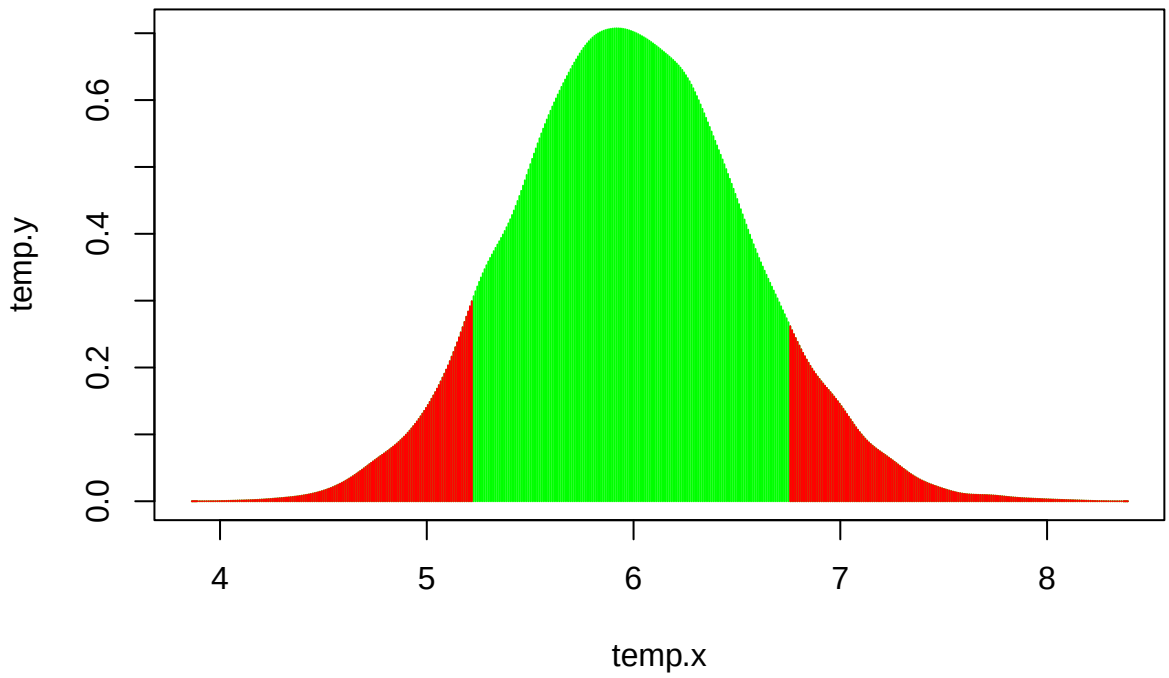
What if our null hypothesis was that the average number of complaints per day is equal to six?

```
plot(density(sampdist))
abline(v = 6, col = "blue")
abline(v = tstat, col = "pink")
```



What is our "red" region in this case? The gap between the null hypothesis and the `tstat` is 6.75 - 6 = .75. Any sample which deviates from the null hypothesis by more than this amount is even worse than our sample statistic. Any sample whose average deviates from the null hypothesis by more than .75, in either direction, should be colored "red". Let us color this and compute the p value.

```
hyp = mean(sampdist)
cutoff1 = hyp - abs(tstat-hyp)
cutoff2 = hyp + abs(tstat-hyp)
temp = density(sampdist)
df = data.frame(temp$x, temp$y)
formula1 = df$temp.x<cutoff1 | df$temp.x>cutoff2
df1 = df[formula1,]
plot(df, col = "green", type = "h")
points(df1, col = "red", type = "h")
```




```
pvalue = length(sampdist[sampdist<cutoff1 | sampdist>cutoff2])/(length(sampdist))
print(pvalue)
```

```
## [1] 0.1583
```

We cannot reject the null hypothesis of this **two-tailed test** either.

We can package p-value calculations into a function for use later.

```
p_rtail = function(sampdist,tstat)
  {
  temp = density(sampdist)
  df = data.frame(temp$x, temp$y)
  formula1 = df$temp.x<tstat
  df1 = df[formula1,]
  plot(df, col = "red", type = "h")
  points(df1, col = "green", type = "h")
  pvalue = length(sampdist[sampdist>tstat])/(length(sampdist))
  return(pvalue)
  }

p_ltail = function(sampdist,tstat)
  {
  temp = density(sampdist)
  df = data.frame(temp$x, temp$y)
```

```
  formula1 = df$temp.x>tstat
  df1 = df[formula1,]
  plot(df, col = "red", type = "h")
  points(df1, col = "green", type = "h")
  pvalue = length(sampdist[sampdist<tstat])/(length(sampdist))
  return(pvalue)
  }

p_2tail = function(sampdist,tstat)
  {
  hyp = mean(sampdist)
  cutoff1 = hyp - abs(tstat-hyp)
  cutoff2 = hyp + abs(tstat-hyp)
  temp = density(sampdist)
  df = data.frame(temp$x, temp$y)
  formula1 = df$temp.x<cutoff1 | df$temp.x>cutoff2
  df1 = df[formula1,]
  plot(df, col = "green", type = "h")
  points(df1, col = "red", type = "h")
  pvalue = length(sampdist[sampdist<cutoff1 |  sampdist>cutoff2])/(length(sampdist))
  return(pvalue)
  }
```

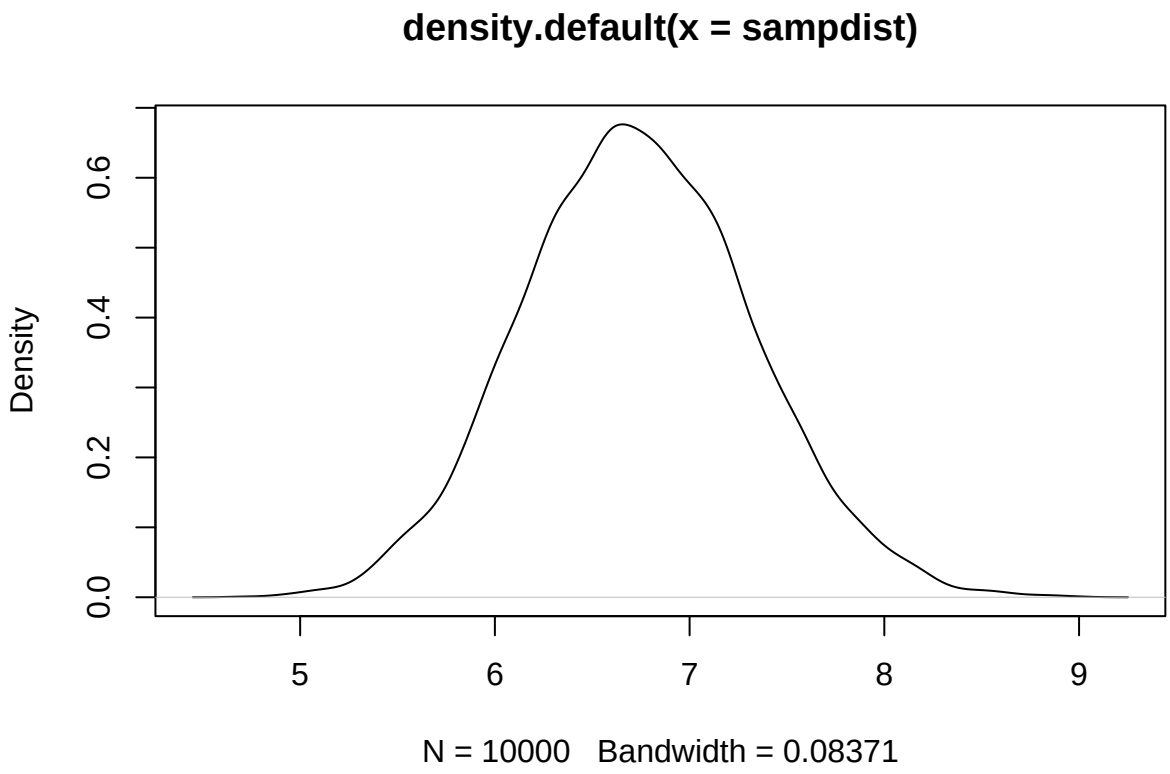
You Be the Judge

As we mentioned before, a common rule of thumb for p values is to use a cutoff of 0.05. However, this is just a rule of thumb and you should not be following it blindly. You have to think about the underlying problem context. What is the implication of rejecting the null hypothesis? Would this cost someone's life? Does it have significant financial or health consequences? If so, you may want to use a much lower cutoff value. For example, if judging someone as guilty can lead to a long term prison sentence, or even death, then it is perhaps wiser to set a much lower cutoff for the p value.

Confidence Intervals

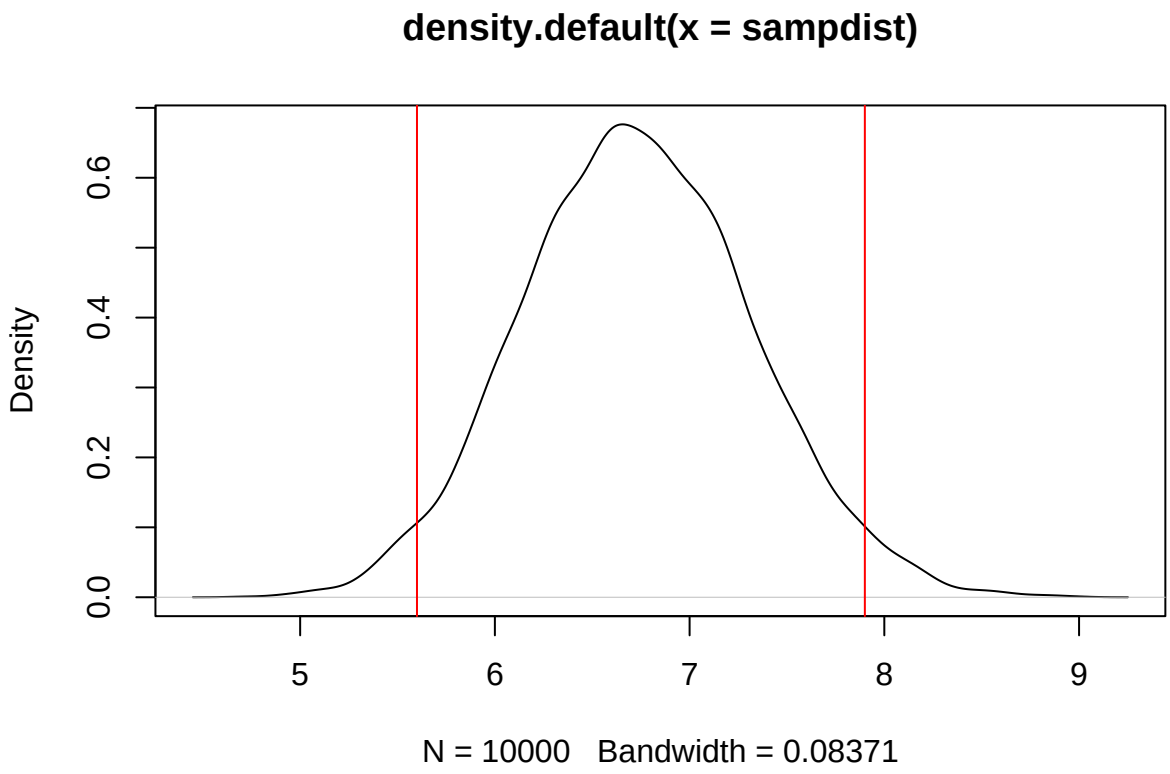
Create a sampling distribution where the population parameters are the same as from the sample.

```
v1 = c(4,3,5,13,7,10,9,9,3,6,4,3,7,10,7,6,7,8,7,7)
set.seed(87654321)
f1 = function(){
  s1 = rpois(n = 20, lambda = mean(v1))
  return(mean(s1))
}
sampdist = replicate(10000, f1())
plot(density(sampdist))
```



Once we have the sampling distribution, we can compute things like the 95% confidence interval.

```
q1 = quantile(sampdist, c(.05/2,1-(.05/2)))
plot(density(sampdist))
abline(v = q1, col = "red")
```



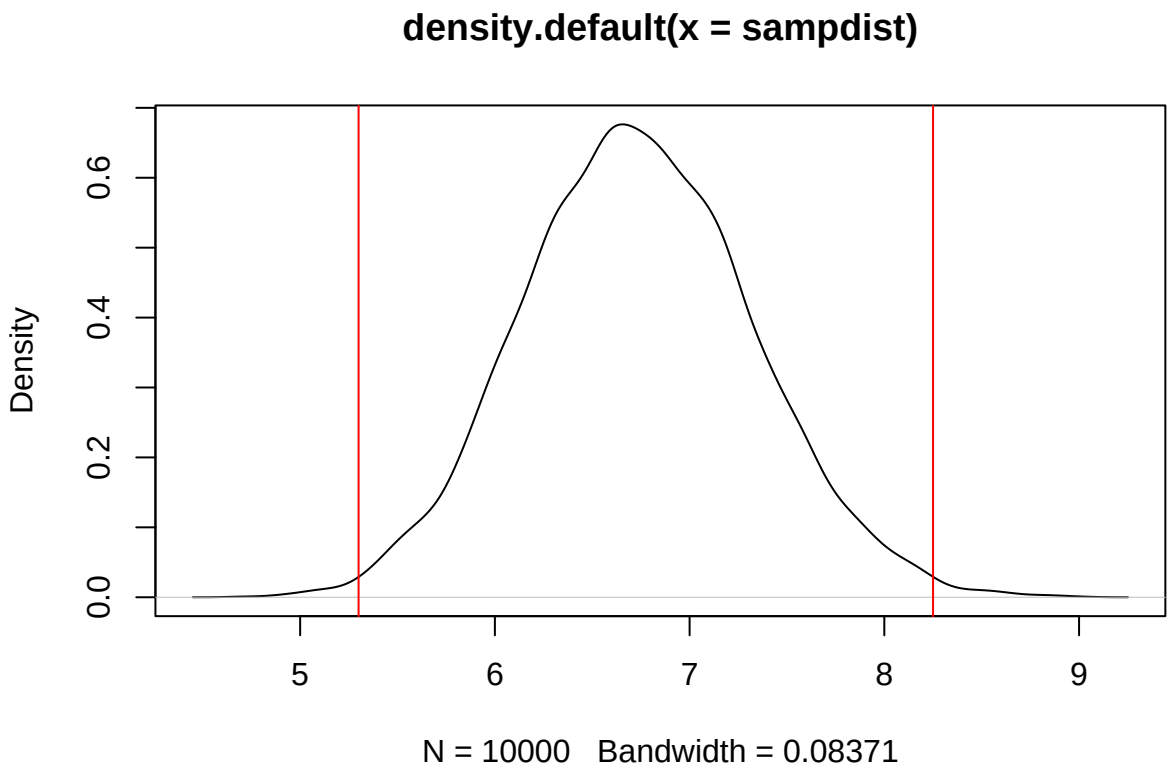

```
output = paste0("[",q1[1],",",q1[2],"]")
print(output)
```

```
## [1] "[5.6,7.9]"
```

In this case, the 95% confidence interval is between 5.6 and 7.9.

```
conf_int = function(sampdist,conlevel=0.95)
{
  left_v = (1 - conlevel)/2
  right_v = 1 - left_v
  q1 = quantile(sampdist,c(left_v,right_v))
  plot(density(sampdist))
  abline(v = q1, col = "red")
  output = paste0("[",q1[1],",",q1[2],"]")
  return(output)
}
```

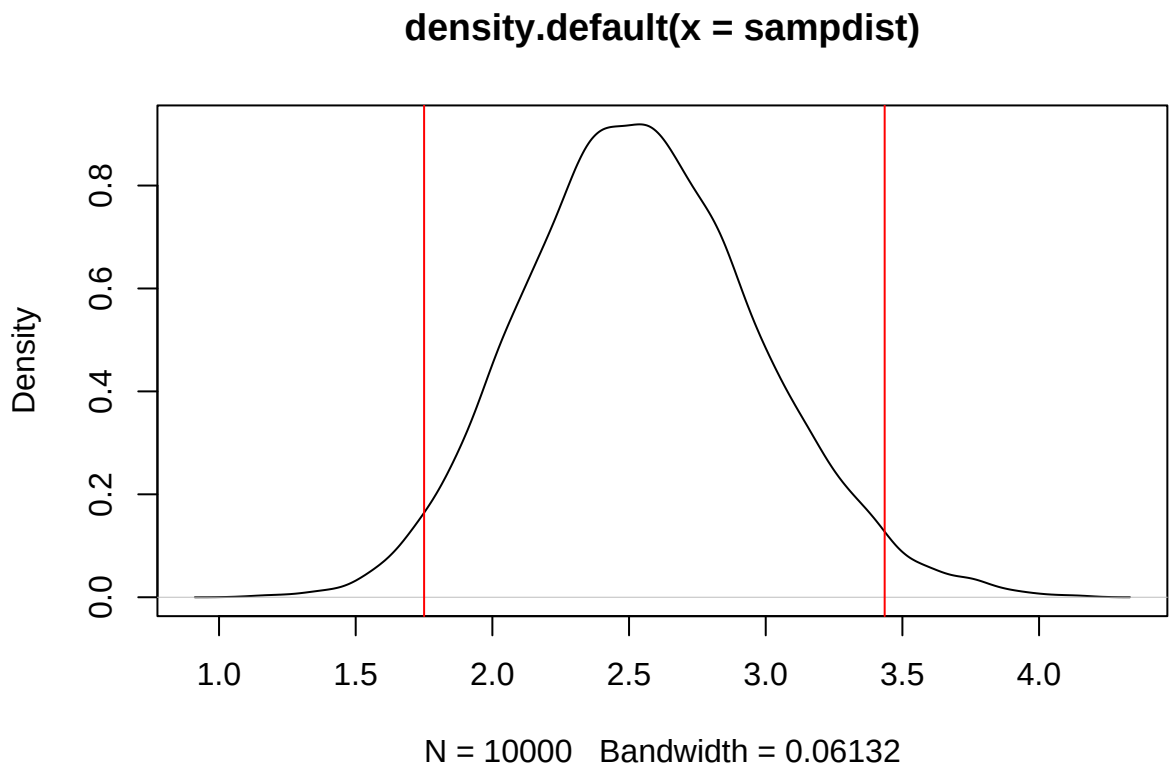
```
conf_int(sampdist,0.99)
```




```
## [1] "[5.3,8.25024999999996]"
```

We can similarly construct confidence intervals for other statistics of interest. Suppose we want to construct a confidence interval for the population standard deviation.

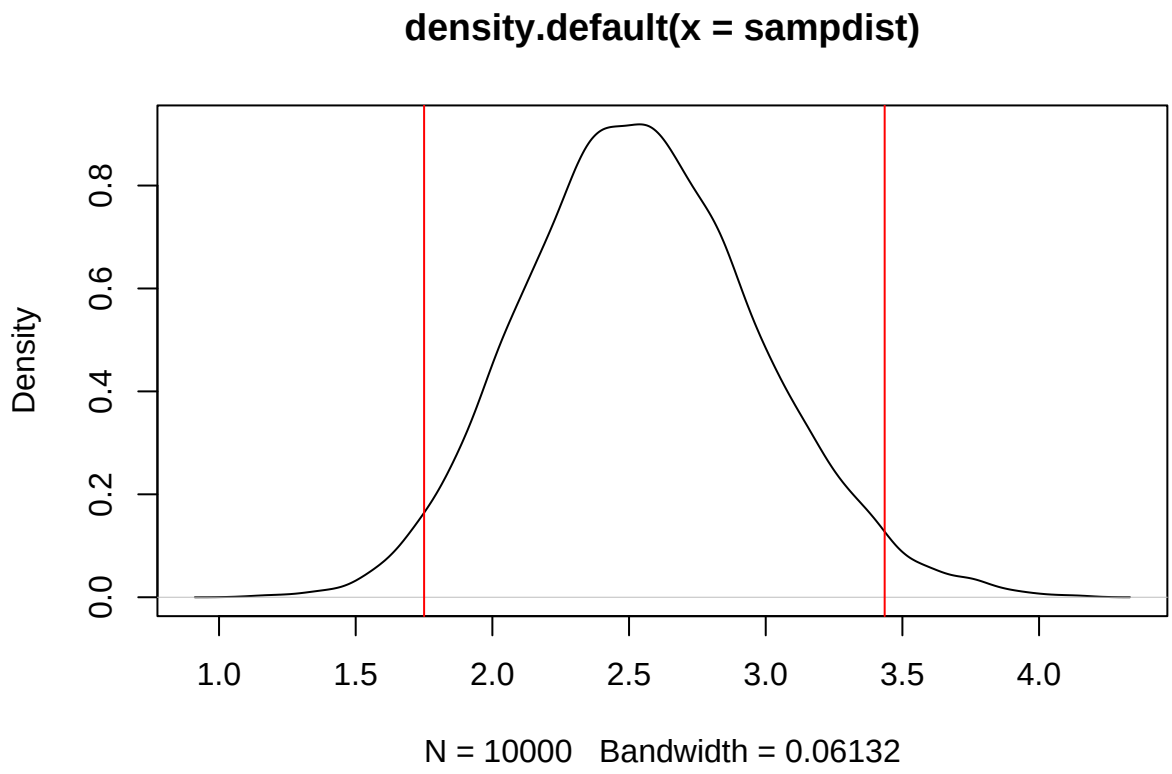
```
v1 = c(4,3,5,13,7,10,9,9,3,6,4,3,7,10,7,6,7,8,7,7)
set.seed(87654321)
f2 = function(){
  s2 = rpois(n = 20, lambda = mean(v1))
  return(sd(s2))
}
sampdist = replicate(10000, f2())
conf_int(sampdist)
```




```
## [1] "[1.75018795983084,3.43517981351968]"
```

Now let us compute the confidence interval.

```
q2 = quantile(sampdist, c(.05/2,1-(.05/2)))
plot(density(sampdist))
abline(v = q2, col = "red")
```

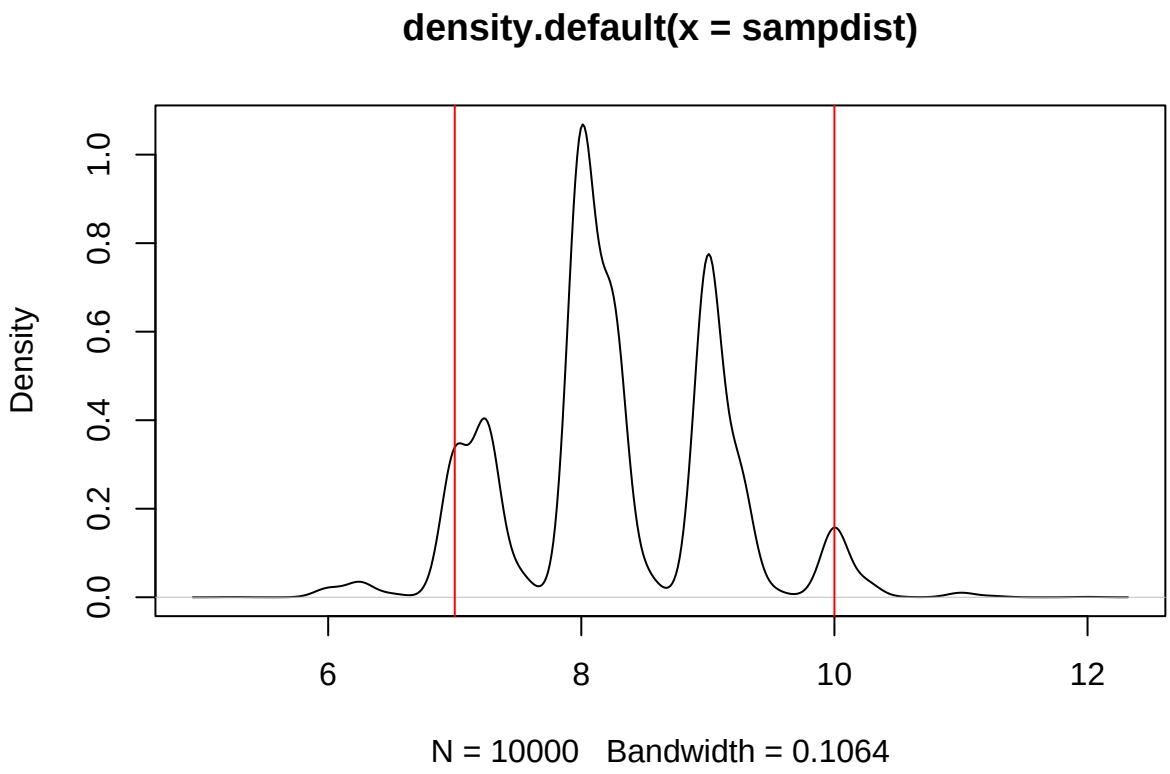



```
paste("95% Confidence interval = ", q2)
```

```
## [1] "95% Confidence interval =  1.75018795983084"
## [2] "95% Confidence interval =  3.43517981351968"
```

Finally, let us compute the confidence interval for another statistic of interest: 75% percentile.

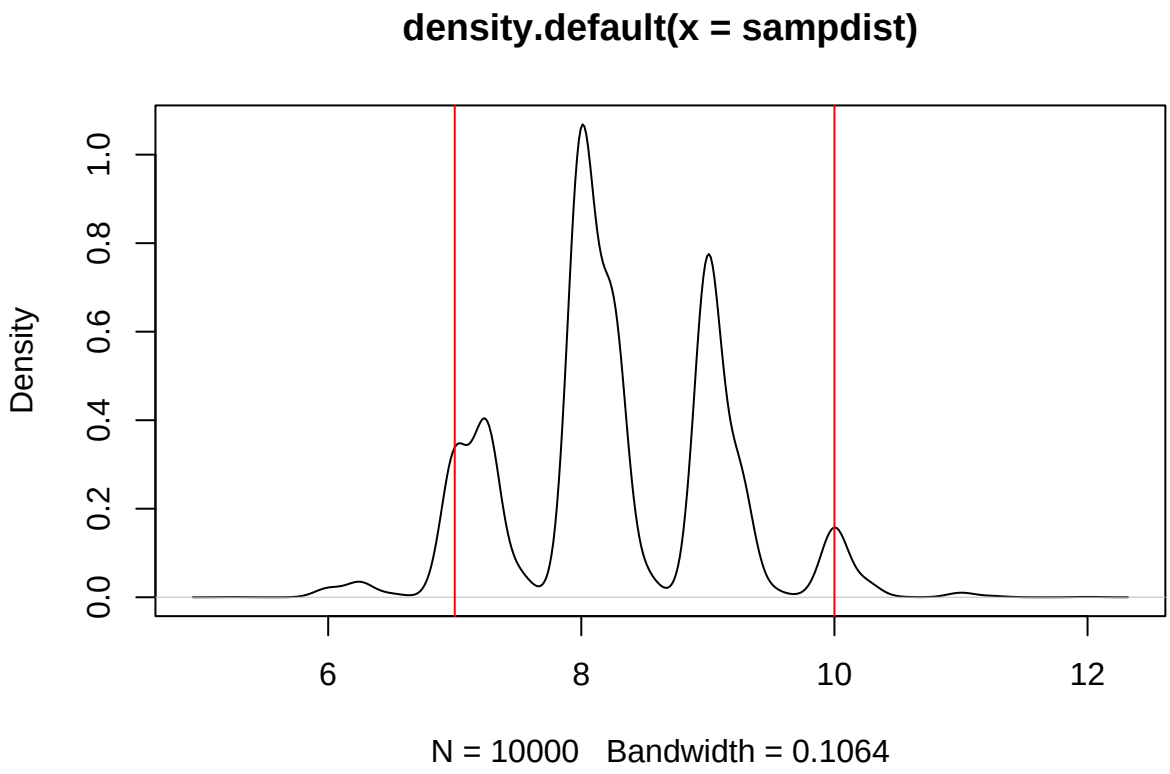
```
v1 = c(4,3,5,13,7,10,9,9,3,6,4,3,7,10,7,6,7,8,7,7)
set.seed(87654321)
f3 = function(){
  s3 = rpois(n = 20, lambda = mean(v1))
  return(quantile(s3, probs = .75))
}
sampdist = replicate(10000, f3())
conf_int(sampdist)
```




```
## [1] "[7,10]"
```

Let us compute the 95% confidence interval.

```
q3 = quantile(sampdist, c(.05/2,1-(.05/2)))
plot(density(sampdist))
abline(v = q3, col = "red")
```




```
paste("95% Confidence interval = ", q3)
```

```
## [1] "95% Confidence interval =  7"  "95% Confidence interval =  10"
```

Use Case: Outlier Detection in Product Data

Testing whether a sample is likely to have come from a given population is a powerful use of probability, as discussed above. Using the same tools, we can also express whether a certain sample is an outlier in our observations: known as Outlier detection. Outlier detection is critical for manufacturing processes, detecting components that fall outside of the required tolerances; it can be important in identifying fraudulent transactions in a FinTech environment, along with many other uses in business.

We can use the tools introduced above to identify outliers in a dataset, and once identified, we as business analysts, or managers, will need to determine whether the presence of outliers is acceptable (or indeed, beneficial).

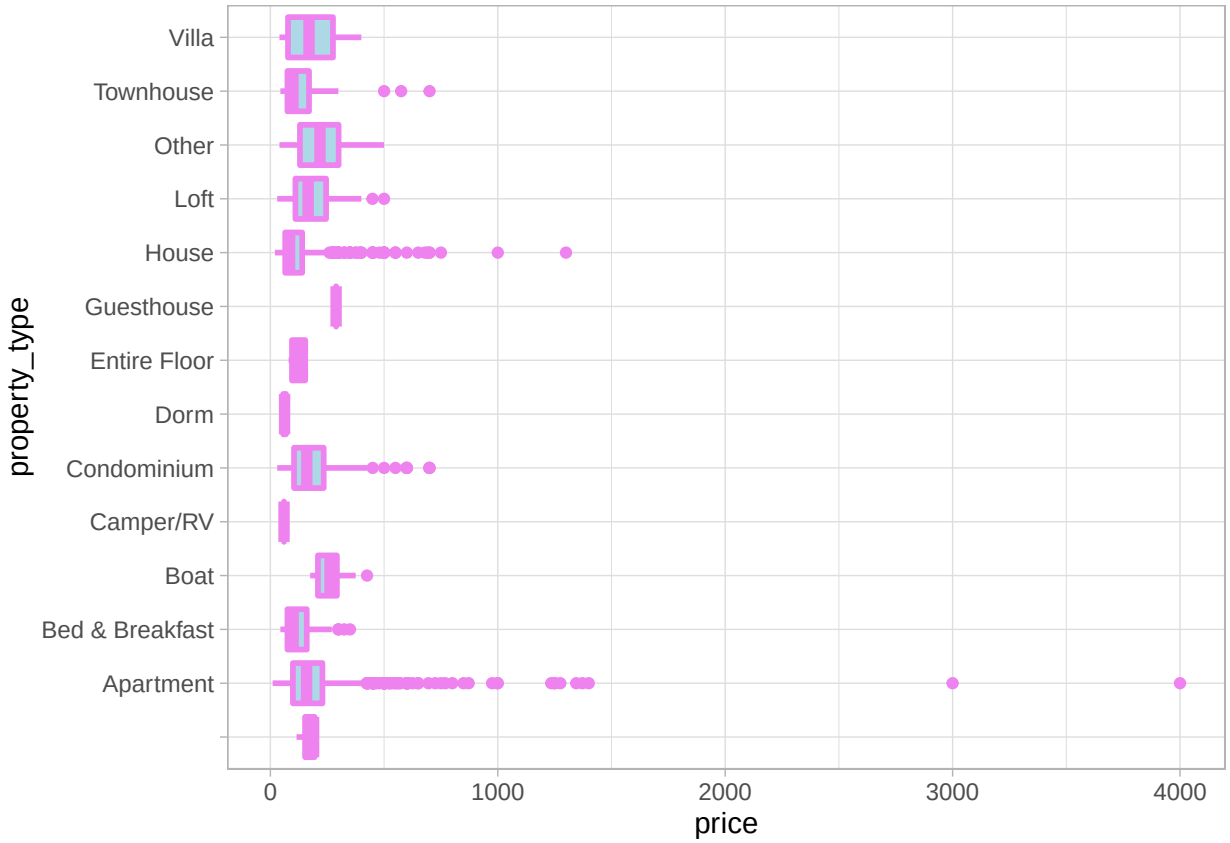
```
df = read.csv("../../data/listings - wrangled.csv")
```

AirBnB Outliers in Boston

Taking the Boston Airbnb Open Data, which has listings, ratings and so on for different properties advertised and rented through AirBnB, we can examine the different properties on offer for outliers, that may or may not benefit AirBnB's offering. Below, you can see the distribution of price for different property types offered

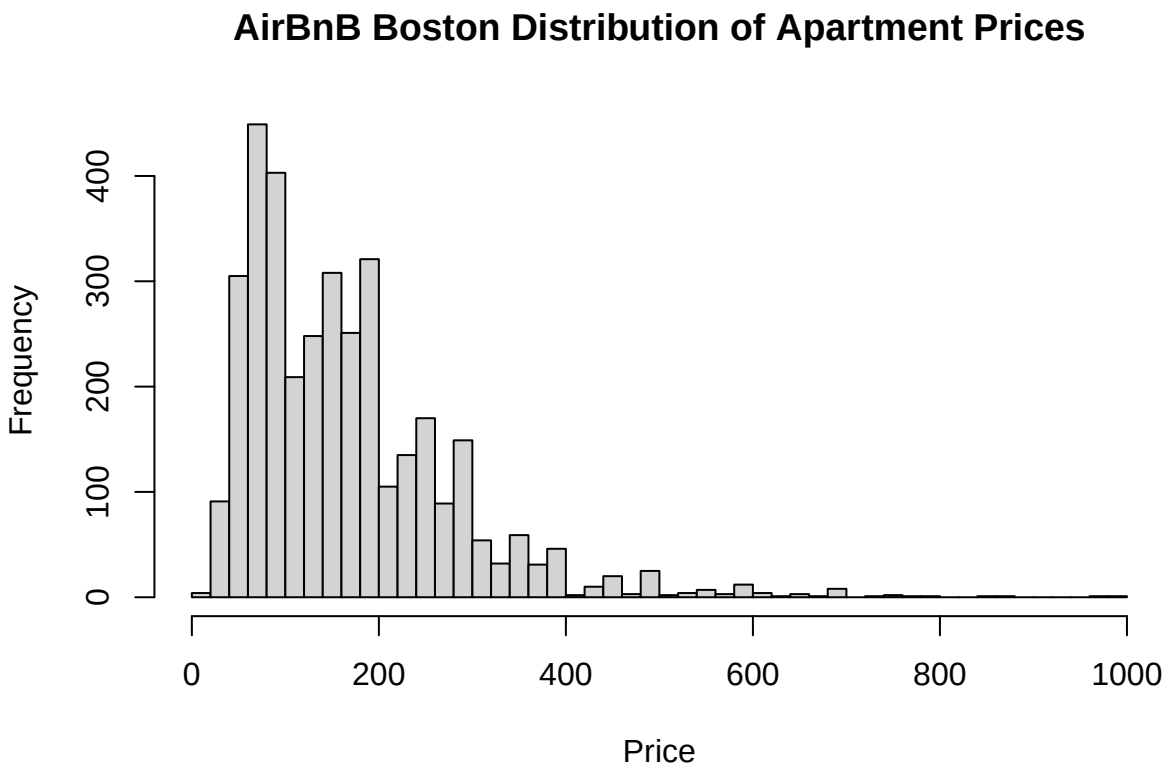
in Boston by AirBnB (note that we have to go through some data wrangling steps to clean the data ready for analysis):

```
library(ggplot2)
ggplot(df,aes(y = property_type ,x=price)) +
  geom_boxplot(col = "violet", fill = "lightblue", size = 1) +
  theme_light()
```



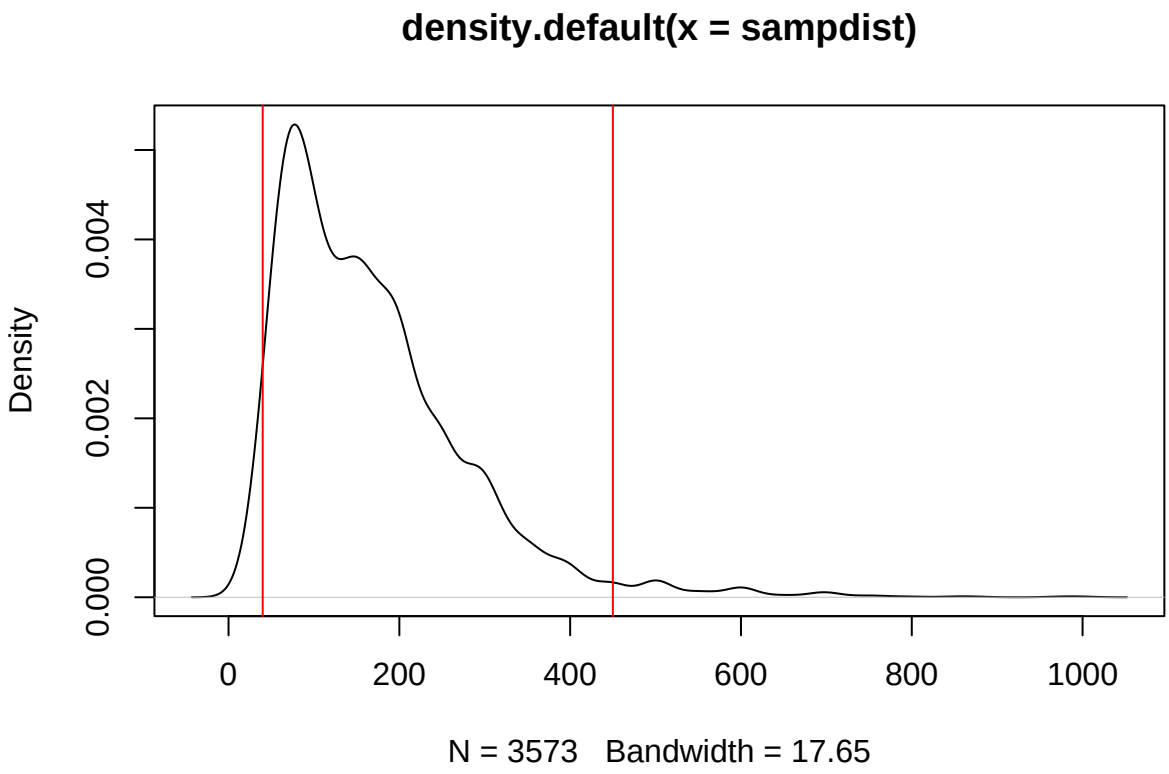
We can see from the chart that there are outliers in most categories, but particularly the Apartments category, where one apartment is offered for around $4000, whereas the median is $159. There are clearly a few very expensive apartments for rent on AirBnB in Boston. If we drill down into apartment prices, plotting a frequency distribution by binning apartment prices into 50 bins using a column chart, this helps visualize the distribution (note that we remove all outliers over $1000 in this case):

```
hist(df$price[df$price<1000],breaks=50,
     main = "AirBnB Boston Distribution of Apartment Prices",
     xlab = "Price")
```



Applying the confidence interval function we have learned above, conf_int, we can replot this data distribution with 95% confidence intervals:

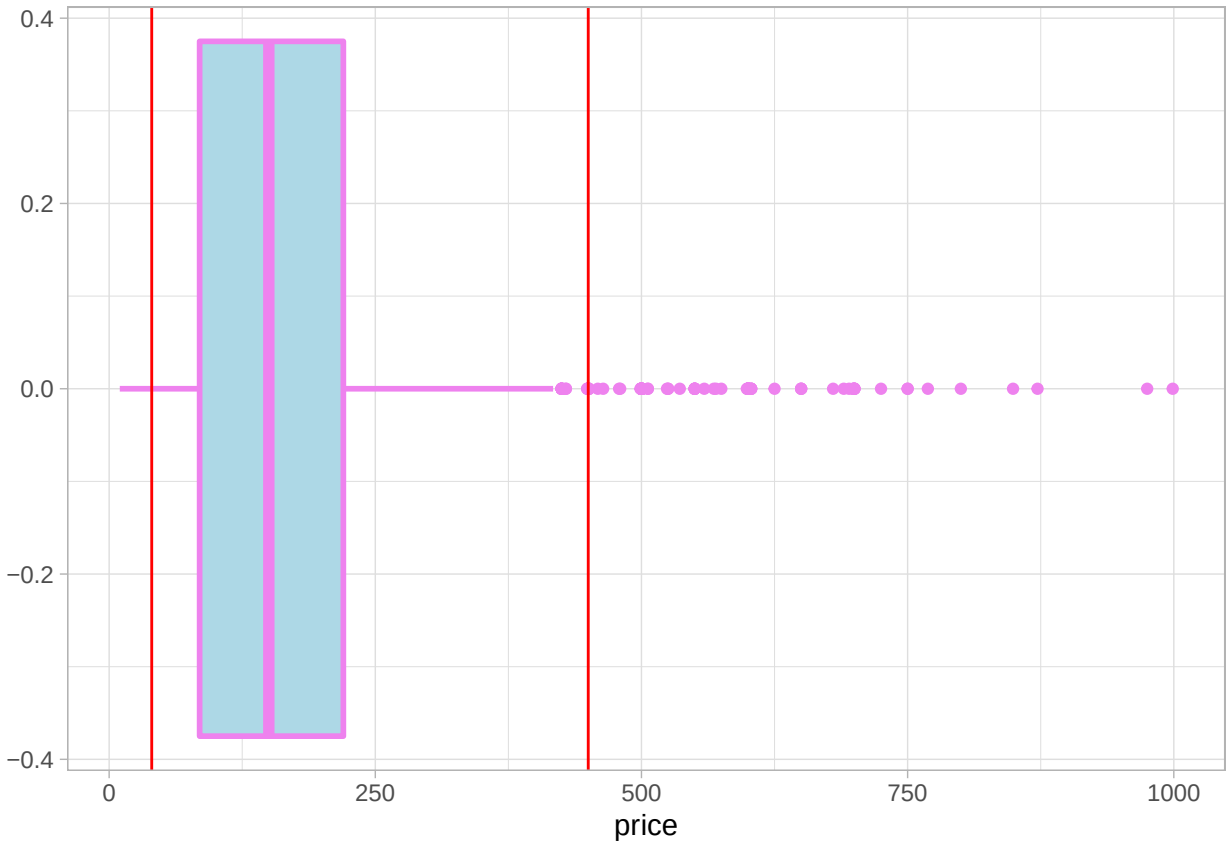
```
conf_int = function(sampdist,conlevel=0.95)
{

  left_v = (1 - conlevel)/2
  right_v = 1 - left_v
  q1 = quantile(sampdist,c(left_v,right_v))
  plot(density(sampdist))
  abline(v = q1, col = "red")
  output = paste0("[",q1[1],",",q1[2],"]")
  return(output)
}
conf_int(df$price[df$price<1000])
```




```
## [1] "[40,450]"
```

If we prefer to look at the distribution as a boxplot, we can see the extremes more clearly:

```
qvalues = quantile(df[df$price<1000,]$price,c(0.05/2,1-0.05/2))
ggplot(df[df$price<1000,],aes(x=price)) +
  geom_boxplot(col = "violet", fill = "lightblue", size = 1) +
  theme_light() +
   geom_vline(xintercept=qvalues,col="red")
```



It is certainly the case that the apartments renting for prices above the right-hand confidence interval or below the left-hand confidence interval, are significantly different offerings from the majority of the apartments on offer. They are outliers, and perhaps we can consider them different beasts – not from the same population as typical AirBnB rentals. It could be that these apartments rarely rent and represent an overhead for the platform. It could also be that they add prestige (and budget options) to the offering. But having identified these very different offerings statistically, we are now able to take whatever qualitative business actions are appropriate.