Lab 3

Stat 131A

February 07, 2022

Welcome to the Lab 3! In this lab, you will:

1) Take a closer look on kernel density estimation

2) Implement a permutation test

3) Implement a T-test

We will continue to use the rent price dataset from the Lab 2.

Read in data:

```
craigslist <- read.csv("craigslist.csv", header = TRUE)
```

Kernel density estimation

In Lab 2, we plotted a kernel density curve of the monthly rents as shown below. For this exercise, experiment with the `bw` argument in the density function, which stands for the bandwidth (width of the the moving window) of the kernel density function. For example, start with `bw = 1`, and scale up by a multiple of 10, until `bw` is greater than n, where n is the number of observations. Observe how the resulting curves change.

```
price <- craigslist$price
d <- density(na.omit(price), bw = 100)
hist(price,
     freq = FALSE,
     ylim = c(0, 0.0005),
     breaks = 50)
lines(d, col = "red")
```

Histogram of price
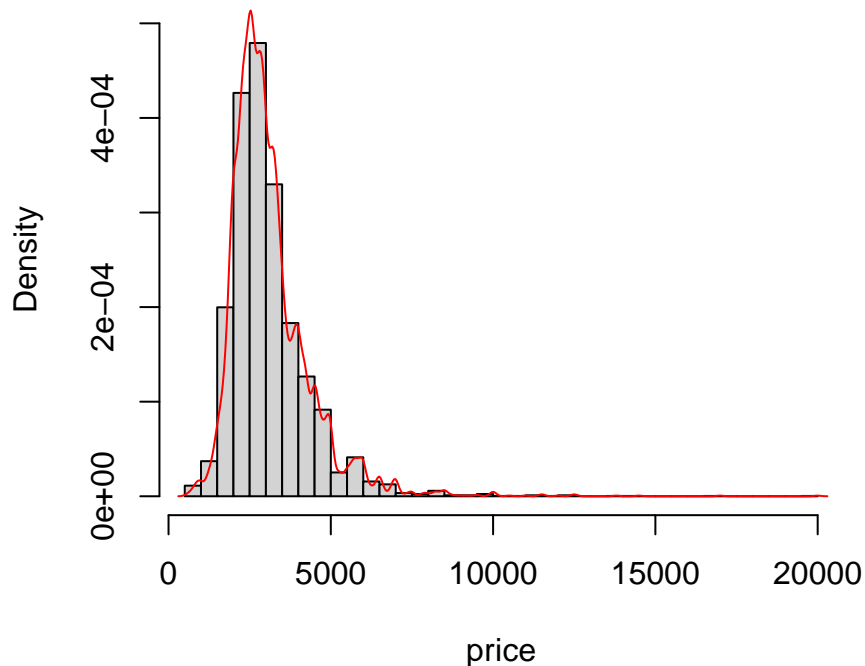
We can recreate the histogram and kernel density curve using `tidy`-style code. The package `tidyverse` includes packages `ggplot2`, `purrr`, `tibble`, `dplyr`, `tidyr`, `stringr`, `readr`, and `forcats`. The code below relies only on `ggplot2` and `dplyr`. Loading `tidyverse` may be more convenient than loading several `tidy` packages individually.

```
# install.packages("tidyverse")
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.4     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.0.2     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
craigslist %>% ggplot(aes(x = price, y = ..density..)) +
  geom_histogram(bins = 50, fill = "lightgrey", color = "black") +
  geom_density(bw = 100, color = "red") +
  labs(x = "Price", y = "Density") +
  ggtitle("Histogram of price") +
  theme_bw()
```

Histogram of price
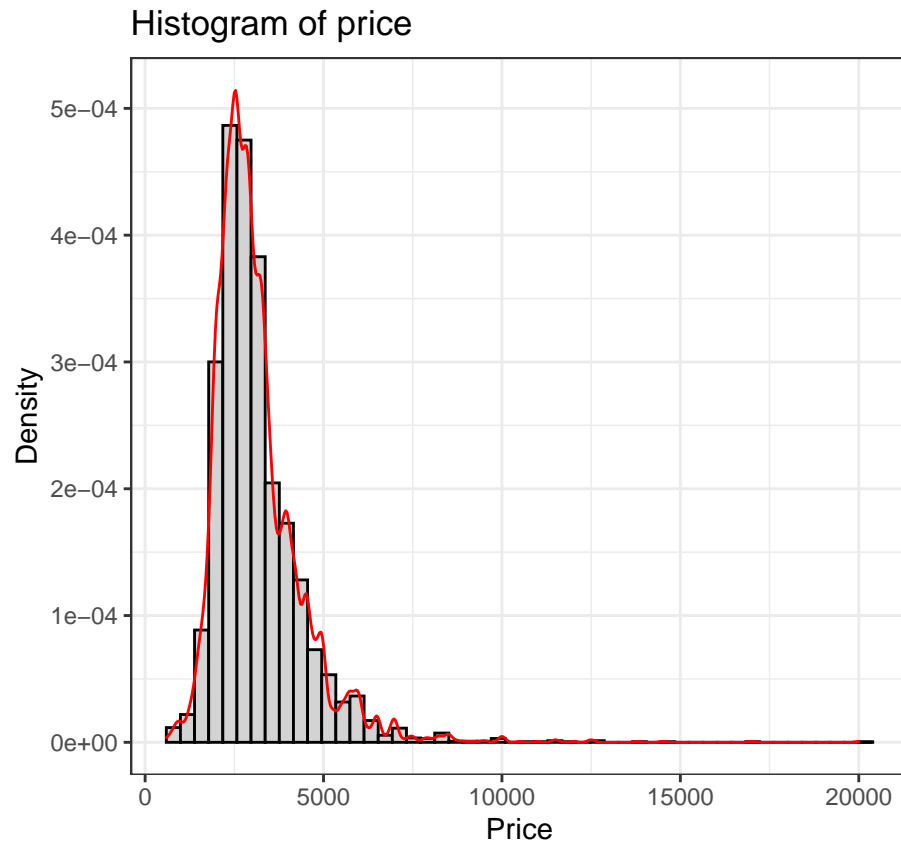
Exercise 1.

(a) What value of bandwidth (approximately) is used as default by the density function on monthly rents? (In other words, what value of the `bw` argument results in the same kernel density estimation as not specifying the `bw` argument?)
Bonus: what is the actual default bandwidth?

Your answer here

(b) How does changing the bandwidth affect the density estimation curve? Briefly explain your intuition on why that is the case.
Hint: consider the weighted kernel function in textbook 2.5.2.2. The bandwidth corresponds to the w in the function.

Your answer here

Replicate

To repeat things in `R`, there is a more convenient and efficient way than using a `for` loop. The function `replicate` is designed for evaluating an expression repeatedly. For example, to get a vector of length 50 where each element is generated independently from the sum of 5 normally distributed samples:

```
set.seed(20172828)
example1 <- replicate(50, sum(rnorm(5)))
```

The equivalence of the above code using for loops will be:

3

```
# Create a vector of length 0 to store the results
example2 <- c()
# Calculate for 50 times
for (i in 1:50) {
  # Combine the calculated sum of normal samples to the end of `results` vector
  example2 <- c(example2, sum(rnorm(5)))
}
```

The first argument of `replicate` function is the number of replications. And the second argument is an expression which will be evaluated repeatedly, which usually involves random sampling and simulation. (Otherwise, you would get a vector of one identical number. That's useless!) Sometimes, the things you want to replicate cannot finish in one line; you may need to use the big curly bracket such as the following. Elements in the `example3` vector would be the value of the last expression in the bracket. (Since we set the same seed for `example1` and `example3`, you may want to compare the value of them to see what happened.)

```
set.seed(20172828)
example3 <- replicate(50, {
  a <- rnorm(5)
  b <- 5
  sum(a) + b     # return the value
})
```

Permutation test

Exercise 2

Here, we will perform a permutation test to compare the average one-bedroom apartment rent price in Berkeley and Palo Alto.

(a) Subset the dataset to only consider one-bedroom apartments in Berkeley and Palo Alto.

```
# Insert code here save the data frame of one bedroom postings
# in Berkeley and Palo Alto as `craigslist.subset`
craigslist.subset <- NULL
```

(b) Calculate the number of postings for Berkeley in the data frame `subset`.

```
# Insert code here save the number of postings for Berkeley as
# `no.berkeley`
no.berkeley <- NULL
```

(c) Calculate the observed statistic, i.e., the absolute difference between the mean of Berkeley and mean of Palo Alto one bedroom rent price.

```
# Insert code here save the observed statistics as
# `stat.obs`
stat.obs <- NULL
```

```

(d) Calculate the permuted statistics (absolute mean difference), repeat 1000 times.
   *Hint:* use `sample` function to sample from a group of observations. You can use either use a `for` loop or the `replicate` function introduced above. However, it is a good practice using `replicate`.

```
# leave seed set as is
set.seed(20172828)
# insert code to save the simulated statistics as
# `stat.bootstrap`
stat.bootstrap <- NULL
```

(e) Calculate the *p*-value of our observed statistic using our approximation of the sampling distribution from part (d). We are testing if there is any difference between the two means. Our null hypothesis is that there is no difference between the two means. Our alternative hypothesis is that there is a difference.

```
# insert code here save the observed statistics as
# `p.value`
p.value <- NULL
```

T-test

With the function `t.test`, implementing T-tests is easy in `R`. We will illustrate using a simulated example:

```r
# Generate 100 samples from N(0, 1)
group1 <- rnorm(100, mean = 0, sd = 1)
# Generate 100 samples from N(0.2, 1)
group2 <- rnorm(100, mean = 0.2, sd = 1)
# perform t test
t.test(group1, group2)
```

```
##
##  Welch Two Sample t-test
##
## data:  group1 and group2
## t = -1.0893, df = 197.4, p-value = 0.2774
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.4252694  0.1226339
## sample estimates:
##   mean of x   mean of y
## 0.008502307 0.159820020
```

`t.test` usually print the results. To obtain the value of the t statistics or p-value, it is generally not wise or convenient to copy and paste from the printed output. Imagine you need to do 1000 T-test simultaneously, it is impossible to copy and paste every time. We need to figure out the output of `t.test`:

```r
ttest.result <- t.test(group1, group2)
```

If you check the `ttest.result` object in your Environment window, you will find that it is a list of 9. And each element in the list stores some information.

```r
names(ttest.result)
```

```
## [1] "statistic"   "parameter"   "p.value"     "conf.int"    "estimate"
## [6] "null.value"  "stderr"      "alternative" "method"      "data.name"
```

To get the p-value, you can use either dollar sign or double square bracket, which are two ways usually used to extract elements from lists.

```r
ttest.result$p.value    # with dollar sign, you can do tab completion
```

```
## [1] 0.2773637
```

```r
ttest.result[['p.value']]
```

```
## [1] 0.2773637
```

Exercise 3

Now, apply T-test to compare the mean rent of one bedroom listings in Palo Alto and Berkeley.

```r
# insert code here and save the t-statistic as
# `rent.1b.tstat`
rent.1b.tstat <- NULL
```

```r
# insert code here and save the p value as
# `rent.1b.pvalue`
rent.1b.pvalue <- NULL
```