Midterm Review

Yi (Chris) Chen

October 30, 2017

Midterm Review

lab

lab one

Recall from your probability class that a random variable X has a normal distribution with mean μ and variance σ^2 (denoted $X \sim N(\mu, \sigma^2)$) if it has a probability density function, or *pdf*, equal to

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

In `R` we can simulate $N(\mu, \sigma^2)$ random variables using the function. For example,

```
rnorm(n = 5, mean = 10, sd = 3)
```

```
## [1] 15.491470  8.386482  5.361511 11.325543 10.412641
```

outputs 5 normally-distributed random variables with mean equal to 10 and standard deviation (this is σ) equal to 3. If the second and third arguments are ommited the default rates are **mean = 0** and **sd = 1**, which is referred to as the "standard normal distribution".

1. Generate 100 random draws rom the standard normal distribution and save them in a vector named **normal100**. Calculate the mean and standard deviation of **normal100**. In words explain why these values aren't exactly equal to 0 and 1.

```
normal100 <- rnorm(n = 100)
mean(normal100)
```

```
## [1] 0.01854359
```

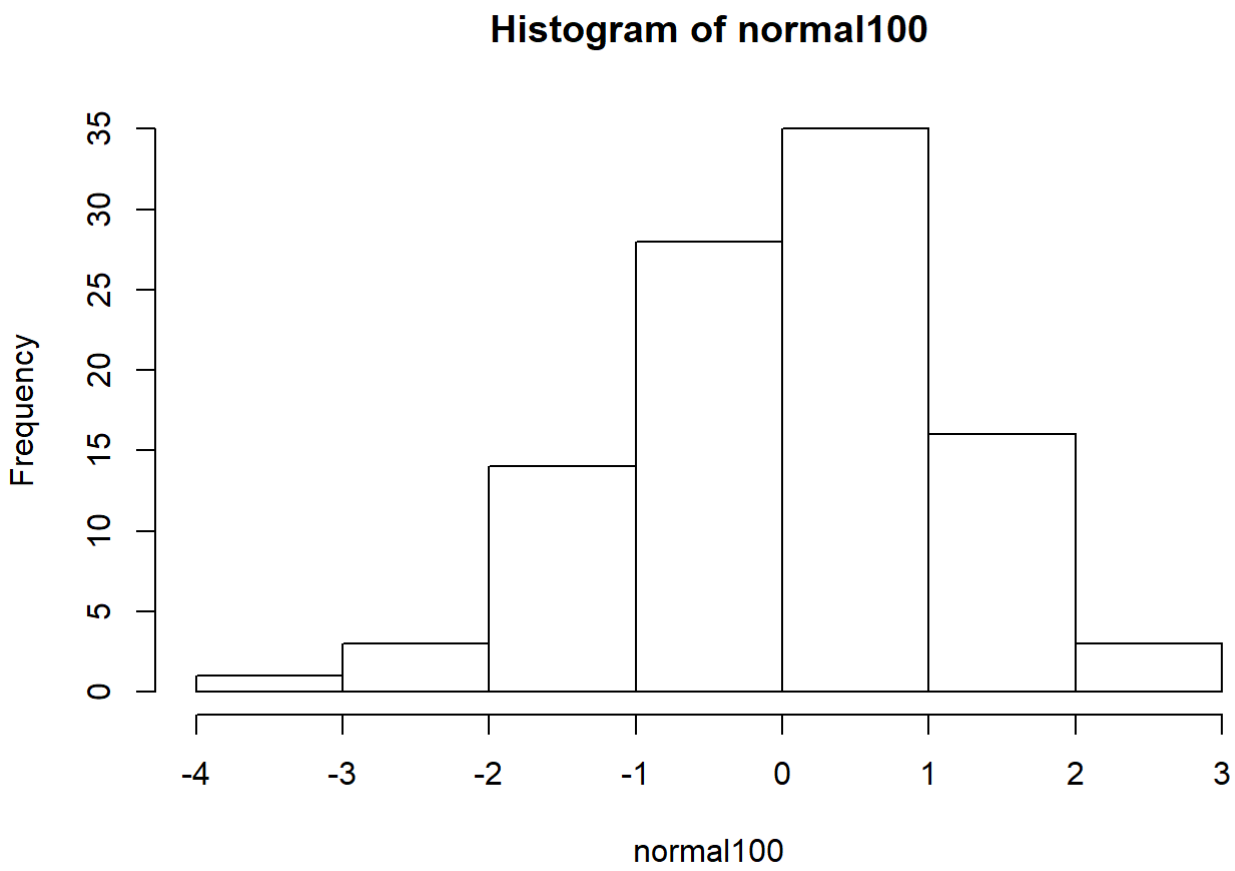
```
sd(normal100)
```

```
## [1] 1.095319
```

The mean and standard deviation aren't exactly equal to 0 and 1 because they are calculated using from random sample.

2. The function **hist()** is a base `R` graphing function that plots a histogram of its input. Use **hist()** with your vector of standard normal random variables from question (1) to produce a histogram of the standard normal distribution. Remember that typing **?hist** in your console will provide help documents for the **hist()** function. If coded properly, these plots will be automatically embedded in your output file.

```
hist(normal100)
```



3. Repeat question (1) except change the number of draws to 10, 1000, 10,000, and 100,000 storing the results in vectors called **normal10**, **normal1000**, **normal10000**, **normal100000**.

```
normal10 <- rnorm(n = 10)
normal1000 <- rnorm(n = 1000)
normal10000 <- rnorm(n = 10000)
normal100000 <- rnorm(n = 100000)
```

4. We want to compare the means of our four random draws. Create a vector called **sample_means** that has as its first element the mean of **normal10**, its second element the mean of **normal100**, its third element the mean of **normal1000**, its fourth element the mean of **normal10000**, and its fifth element the mean of **normal100000**. After you have created the **sample_means** vector, print the contents of the vector and use the **length()** function to find the length of this vector. (it should be five). There are, of course, multiple ways to create this vector. Finally, explain in words the pattern we are seeing with the means in the **sample_means** vector.

```
sample_means <- c(mean(normal10), mean(normal100), mean(normal1000), mean(normal10000), mean(
normal100000))
sample_means
```

```
## [1] -0.1435169593  0.0185435917  0.0050219113  0.0090597580 -0.0001757431
```

```
length(sample_means)
```

```
## [1] 5
```

As the sample size increases, the mean of the sample is becoming closer and closer to 0, which is the mean of the random variables making up the sample. As made explicit in question (10) below, this is because the sample mean is distributed as $N(0, 1/n)$.

lab two

1. Generate 1 million random draws from a normal distribution with $\mu = 3$ and $\sigma^2 = 4$ and save them in a vector named **normal1mil**. Calculate the mean and standard deviation of **normal1mil**.

```
normal1mil <- rnorm(n = 1000000, mean = 3, sd = 2)
mean(normal1mil)
```

```
## [1] 3.000094
```

```
sd(normal1mil)
```

```
## [1] 2.000371
```

2. Find the mean of all the entries in **normal1mil** that are greater than 3. You may want to generate a new vector first which identifies the elements that fit the criteria.

```
mean(normal1mil[normal1mil>3])
```

```
## [1] 4.595736
```

3. Create a matrix **normal1mil_mat** from the vector **normal1mil** that has 10,000 columns (and therefore should have 100 rows).

```
normal1mil_mat <- matrix(normal1mil, ncol = 10000)
```

4. Calculate the mean of the 1234^{th} column.

```
mean(normal1mil_mat[,1234])
```

```
## [1] 3.131722
```

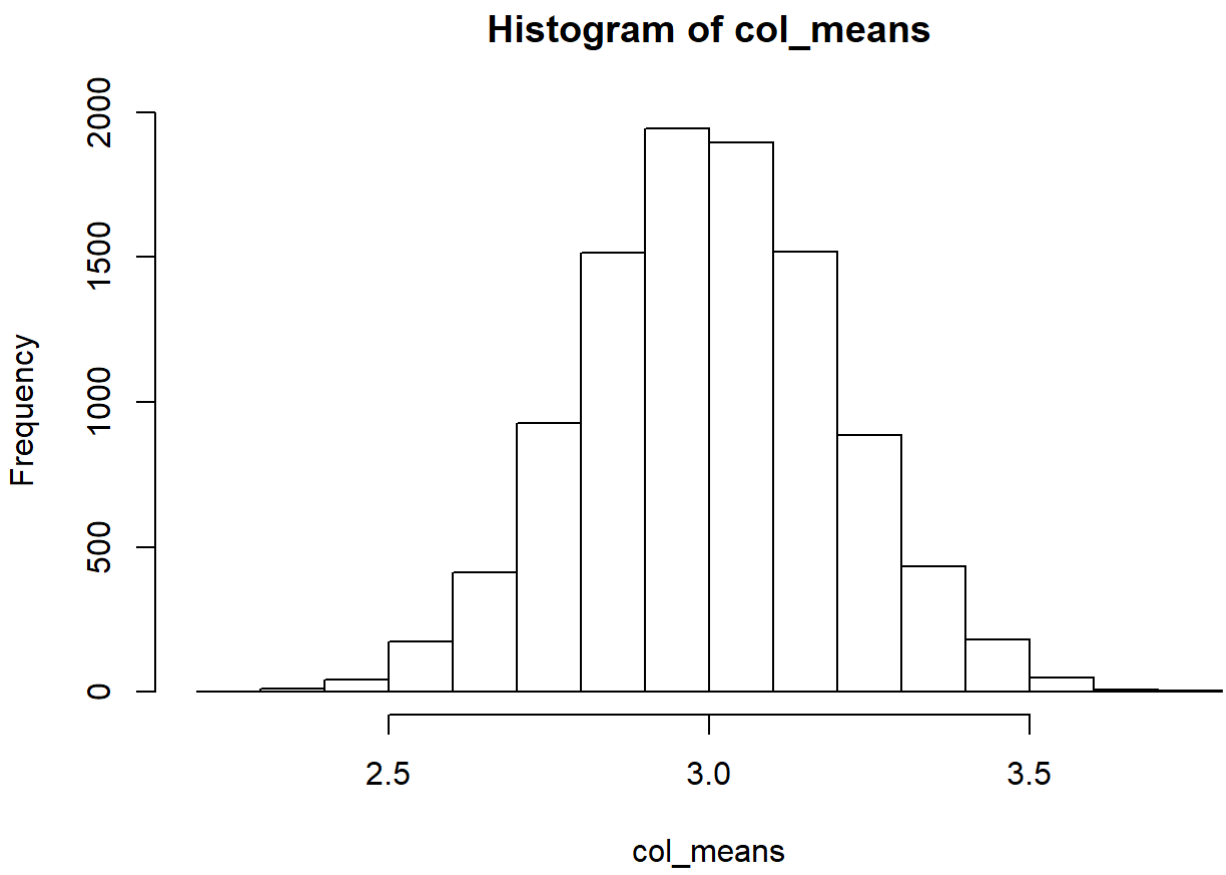
5. Use the **colSums()** functions to calculate the *means* of each column of **normal1mil_mat**. Save the vector of column means as **col_means** as it will be used in the next task. Use the **apply()** function to do the same thing, saving the results in a new vector **col_means2**, then compare the two with **all(col_means == col_means2)**.

```
samp_size <- 100
col_means <- colSums(normal1mil_mat)/samp_size
col_means2 <- apply(normal1mil_mat, 2, sum)/samp_size
identical(col_means, col_means2)
```

```
## [1] TRUE
```

6. Finally, produce a histogram of the column means you calculated in task (5). What is the distribution that this histogram approximates (i.e. what is the distribution of the sample mean in this case)?

```
hist(col_means)
```



7. Only complete the following questions if you have time. We'd like to count the number of values beyond 3 standard deviations from the mean in each column of our matrix and store them in a vector called **num_3devs** (that should have length 10,000). (a) Write code that loops over the columns and calculates this value for each column filling in the vector **num_3devs** as it iterates. (b) Use a smart call to the **apply()** function to do the same thing. (c) Make a table of these values.

```
num_3devs <- rep(0, ncol(normal1mil_mat))
for (i in 1:ncol(normal1mil_mat)) {
  num_3devs[i] <- sum(normal1mil_mat[, i] > 9 | normal1mil_mat[,i] < -3)
}

TFmat        <- normal1mil_mat > 9 | normal1mil_mat < -3
num_3devs_v2 <- apply(TFmat, 2, sum)

all(num_3devs == num_3devs_v2)
```

```
## [1] TRUE
```

```
table(num_3devs)
```

```
## num_3devs
##    0    1    2    3    4
## 7658 2063  257   21    1
```

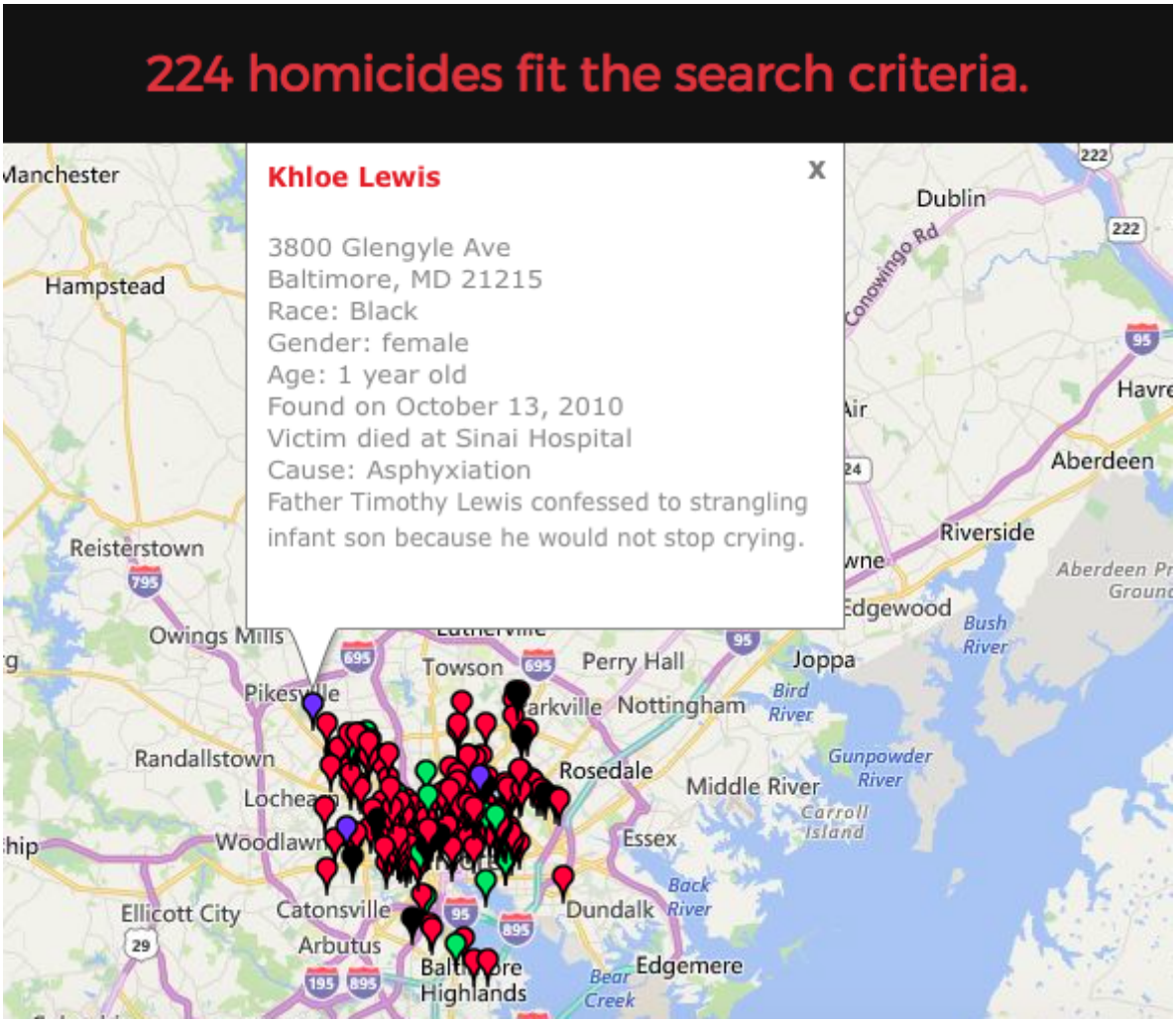
lab three

In this lab, we will use data from homicides in Baltimore City that are collected by the Baltimore Sun newspaper. The data is presented in a map that is publically available at the following website: . I've scraped the data from the website and saved it in a file **BaltimoreHomicides.txt**. Load the data with the following:

```
setwd("C:/Users/cheny/Desktop/study/statistical computing and intro to data science/homework/solution")
data <- readLines("BaltimoreHomicides.txt")
```

The data we have corresponds to all the homicides in 2010. There are 224 lines of HTML in the dataset (verify this using **length(data)**) and 224 homicides in 2010.

1. In the image below, you can see one of the homicides in our dataset. Use a **grep()** call to find which row in the dataset corresponds to the death of Khloe Lewis. (You could search, for example, for her name and the row you should return is 52).



image

```
grep("Khloe Lewis", data)
```

```
## [1] 52
```

2. Suppose we wanted to identify the records for all the victims of shootings (as opposed to other causes). Using the following bit of code and some of your own exploration, how many of the homicides in our dataset are the result of shootings?

```
deaths1 <- grep("shooting", data)
deaths2 <- grep("Shooting", data)
length(deaths1)
```

```
## [1] 172
```

```
length(deaths2)
```

```
## [1] 171
```

```
setdiff(deaths1, deaths2)
```

```
## [1] 105
```

In the above we can see that when we search for “shooting” we find 172 matches but only 171 matches when searching for “Shooting” and the two sets are identical except for row 105. When we look at row 105:

```
data[105]
```

```
## [1] "\"105\" \"\\\"105\\\" \\\"\t\t\t\t\t\tnewpoints[104] = new Array(39.33743900000, -76.
66316500000, icon_homicide_bluntforce, 'p914', '<dl><dt><a href=\\\\\"http://data.baltimoresu
n.com/news/police/homicides/victim/914/\\\\\">Steven Harris</a></dt><dd class=\\\\\"address
\\\\\">4200 Pimlico Road<br />Baltimore, MD 21215</dd><dd>Race: Black<br />Gender: male<br />
Age: 38 years old</dd><dd>Found on July 29, 2010</dd><dd>Victim died at Scene</dd><dd>Cause:
Blunt Force</dd><dd class=\\\\\"popup-note\\\\\"><p>Harris was found dead July 22 and ruled a
shooting victim; an autopsy subsequently showed that he had not been shot,...</dd></dl
>');\\\"\""
```

we see that the cuse of the death was actually blunt force trauma, but in the description of the event, the word “shooting” appears. So there are 171 homicides caused by shooting.

3. The following code creates a vector of the ages of the homicide victims in the dataset:

```
r       <- regexpr(">Age:\\s.*year(s)? old<", data)
age_vec <- regmatches(data, r)
age_vec <- substring(age_vec, 2, nchar(age_vec) - 1)
head(age_vec)
```

```
## [1] "Age: 50 years old" "Age: 25 years old" "Age: 20 years old"
## [4] "Age: 23 years old" "Age: 14 years old" "Age: 43 years old"
```

a. Explain what the regular expression “>Age:\s.*years old<" searches for.

The regular expression looks for the string “>Age:” exactly, followed by anything, then the string “years old<” exactly.

b. Explain in words what the output of the first line of code provides.

The first line of code returns the starting character of the regular expression in each row of the data set and also the length of each match.

c. The second line of code.

The second line of code returns the actual matches (including the brackets “>” and “<”).

d. The third line of code.

The third line of code removes the brackets “>” and “<”, returning only the string “Age: __ years old” for each individual.

4. Use the same strategy as we used in question (3) to create a vector holding each victims’ name. Hint: I had to use the fact that the first letter of the name is capitalized and the specific structure of the HTML code after the name in writing my regular expression.

```
r       <- regexpr(">[A-Z]+.*</a></dt>", data)
name_vec <- regmatches(data, r)
name_vec <- substring(name_vec, 2, nchar(name_vec) - 9)
head(name_vec)
```

```
## [1] "Bernard Clowney"   "David King"        "Raymond Woodland"
## [4] "Keith L. Robinson" "Issac Joyner"      "Mustafa Malik"
```

lab four

Today we’ll be using the **Weekly** dataset from the `ISLR` package. This data is similar to the **Smarket** data from class. The dataset contains 1089 weekly returns from the beginning of 1990 to the end of 2010. Make sure that you have the `ISLR` package installed and loaded by running (without the code commented out) the following:

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

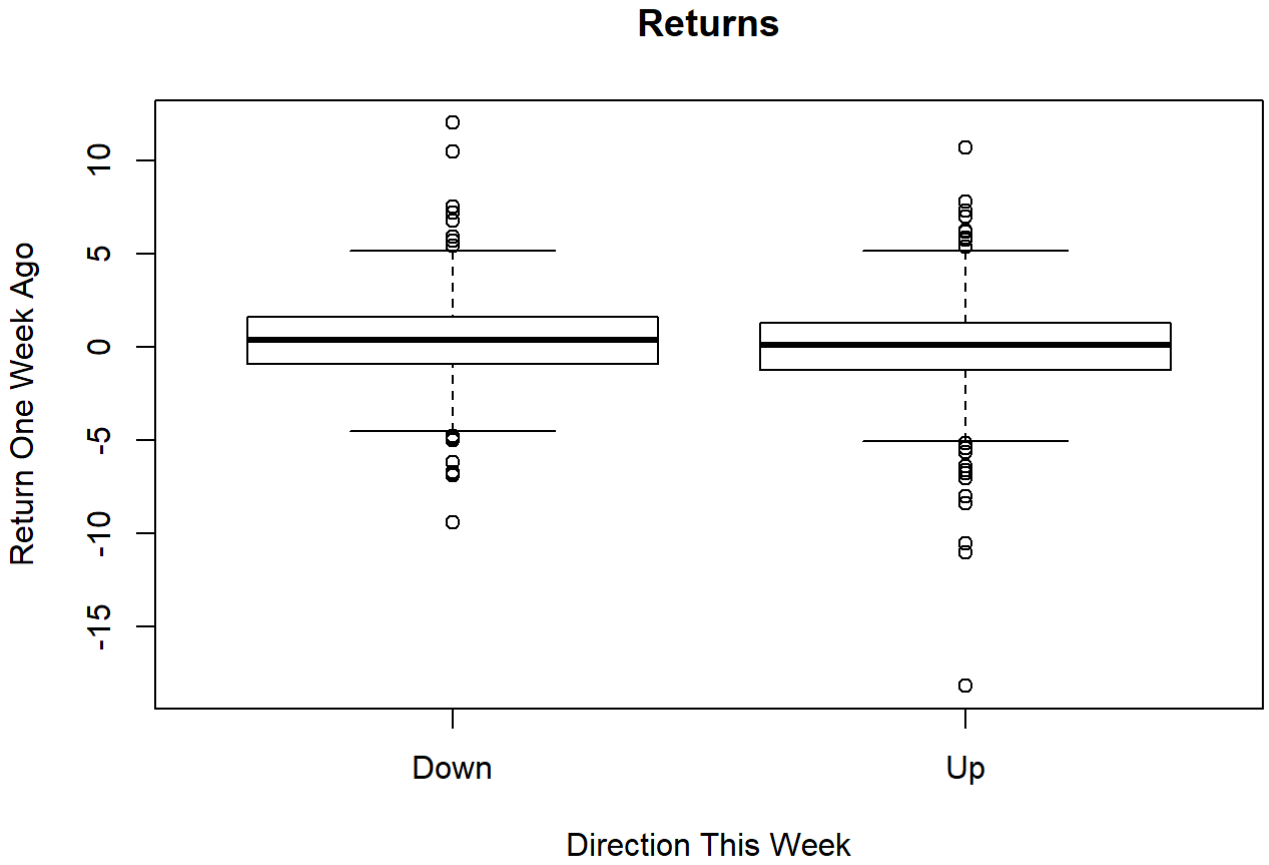
We’d like to see if we can accurately predict the direction of a week’s return based on the returns over the last five weeks. **Today** gives the percentage return for the week considered and **Year** provides the year that the observation was recorded. **Lag1** - **Lag5** give the percentage return for 1 - 5 weeks previous and **Direction** is a factor variable indicating the direction (‘UP’ or ‘DOWN’) of the return for the week considered.

Part 1: Visualizing the relationship between this week’s returns and the previous week’s returns.

1. Explore the relationship between a week’s return and the previous week’s return. You should plot more graphs for yourself, but include in the lab write-up a scatterplot of the returns for the weeks considered (**Today**) vs the return from two weeks previous (**Lag2**), and side-by-side boxplots for the lag one week previous (**Lag1**) divided by the direction of this week’s return (**Direction**).

```
## Scatterplot
plot(Weekly$Lag2, Weekly$Today, main="Returns", xlab="Return Two Weeks Ago",
     ylab=" Return This Week")
```


```
## Boxplot
boxplot(Weekly$Lag1 ~ Weekly$Direction, main="Returns", ylab="Return One Week Ago",
        xlab="Direction This Week")
```



Part 2: Building a classifier

Recall the KNN procedure. We classify a new point with the following steps:

– Calculate the Euclidean distance between the new point and all other points.

– Create the set $\mathcal{N}_{new}$ containing the K closest points (or, nearest neighbors) to the new point.

– Determine the number of 'UPs' and 'DOWNs' in $\mathcal{N}_{new}$ and classify the new point according to the most frequent.

2. We'd like to perform KNN on the **Weekly** data, as we did with the **Smarket** data in class. In class we wrote the following function which takes as input a new point $(Lag1_{new}, Lag2_{new})$ and provides the KNN decision using as defaults $K = 5$, Lag1 data given in **Smarket$Lag1**, Lag2 data given in **Smarket$Lag2**, and Direction data given in **Smarket$Direction**. Update the function to calculate the KNN decision for weekly market direction using the **Weekly** dataset with **Lag1** - **Lag5** as predictors. Your function should have only four input values: (1) a new point which should be a vector of length 5, (2) a value for K (with default of 5), (3) the Lag data which should be a data frame with five columns (and n rows) and have as default the **Weekly** data and (4) the Direction data which should be vector of length n and have as default the **Weekly** data directions. Test your function on a point `c(-.5, .5, -.5, -.5, .5)` . The result should be 'UP'.

```
KNNclass <- function(NewPoint, K = 5, LagData = Weekly[, 2:6], Direction = Weekly$Direction)
{

   n <- nrow(LagData)

   stopifnot(length(NewPoint) == 5, ncol(LagData) == 5, K <= n)

   dists      <- sqrt(rowSums((LagData - rep(NewPoint, each = n))^2))
   neighbors  <- order(dists)[1:K]
   neighb.dir <- Direction[neighbors]
   choice     <- names(which.max(table(neighb.dir)))
   return(choice)
}

# A test point: NewPoint = c(-.5, .5, -.5, -.5, .5)
KNNclass(c(-.5, .5, -.5, -.5, .5))
```

```
## [1] "Up"
```

3. Now train your model using data from 1990 - 2008 and use the data from 2009-2010 as test data. To do this, divide the data into two data frames, `test` and `train`. Then write a loop that iterates over the test points in the test dataset calculating a prediction for each based on the training data with $K = 5$. Save these predictions in a vector. Finally, calculate your test error, which you should store as a variable named `test.error`. The test error calculates the proportion of your predictions which are incorrect (don't match the actual directions). HINT: Since the test data is stored in a dataframe, you may need to use an `as.numeric()` call when you pass it in as your new point. Or maybe not, depending on how you wrote your `KNNclass` function.

```
test  <- Weekly[Weekly$Year >= 2009, ]
train <- Weekly[Weekly$Year < 2009, ]

n.test      <- nrow(test)
predictions <- rep(NA, n.test)

for (i in 1:n.test){
  predictions[i] <- KNNclass(as.numeric(test[i, 2:6]), LagData = train[, 2:6],
                             Direction = train$Direction)
}

test.error <- mean(predictions != test$Direction)
test.error
```

```
## [1] 0.4519231
```

Note that I use the *as.numeric()* call around the test point in the above because when I access a row from a data frame like in *test[i, 2:6]* the output is another data frame. For my code to work, I want the output to be a vector and *as.numeric()* changes the output to be a vector.

4. Do the same thing as in question 3, but instead use $K = 3$. Which has a lower test error?

```
predictions <- rep(NA, n.test)

for (i in 1:n.test){
  predictions[i] <- KNNclass(as.numeric(test[i, 2:6]), K= 3, LagData = train[, 2:6],
                             Direction = train$Direction)
}

test.error <- mean(predictions != test$Direction)
test.error
```

```
## [1] 0.4423077
```

lab five

In this lab we look at dataset containing information on the world's richest people from the World Top Incomes Database (WTID) hosted by the Paris School of Economics [http://wid.world (http://wid.world)]. This is derived from income tax reports, and compiles information about the very highest incomes in various countries over time, trying as hard as possible to produce numbers that are comparable across time and space. For most countries in most time periods, the upper end of the income distribution roughly follows a Pareto distribution, with probability density function

$$f(x) = \frac{(a-1)}{x_{min}}\left(\frac{x}{x_{min}}\right)^{-a}$$

for incomes $X \geq x_{min}$. (Typically, x_{min} is large enough that only the richest 3%-4% of the population falls above it.) As the *Pareto exponent*, a, gets smaller, the distribution of income becomes more unequal, that is, more of the population's total income is concentrated among the very richest people.

The proportion of people whose income is at least x_{min} and whose income is also at or above any level $w \geq x_{min}$ is thus

$$\mathbf{Pr}(X \geq w) = \int_w^\infty f(x)dx = \int_w^\infty \frac{(a-1)}{x_{min}}\left(\frac{x}{x_{min}}\right)^{-a} dx = \left(\frac{w}{x_{min}}\right)^{-a+1}.$$

We will use this to estimate how income inequality changed in the US over the last hundred years or so. (Whether the trends are good or bad or a mix is beyond our scope here.) WTID exports its data sets as `.xlsx` spreadsheets. For this lab session, we have extracted the relevant data and saved it as `wtid-report.csv`.

1. Open the file and make a new dataframe containing only the year, the "P99", "P99.5" and "P99.9" variables; these are the income levels which put someone at the 99th, 99.5th, and 99.9th, percentile of income. Rename the columns of your new dataframe as `Year`, `P99`, `P99.5`, `P99.9`. What was P99 in 1993? P99.5 in 1942? You must identify these using your code rather than looking up the values manually.

```
wtid <- read.csv("wtid-report.csv", as.is = TRUE)
wtid <- wtid[, c("Year", "P99.income.threshold", "P99.5.income.threshold", "P99.9.income.thre
shold")]
names(wtid) <- c("Year", "P99", "P99.5", "P99.9")
wtid$P99[wtid$Year == "1993"]
```

```
## [1] 273534.9
```

```
wtid$P99.5[wtid$Year == "1942"]
```

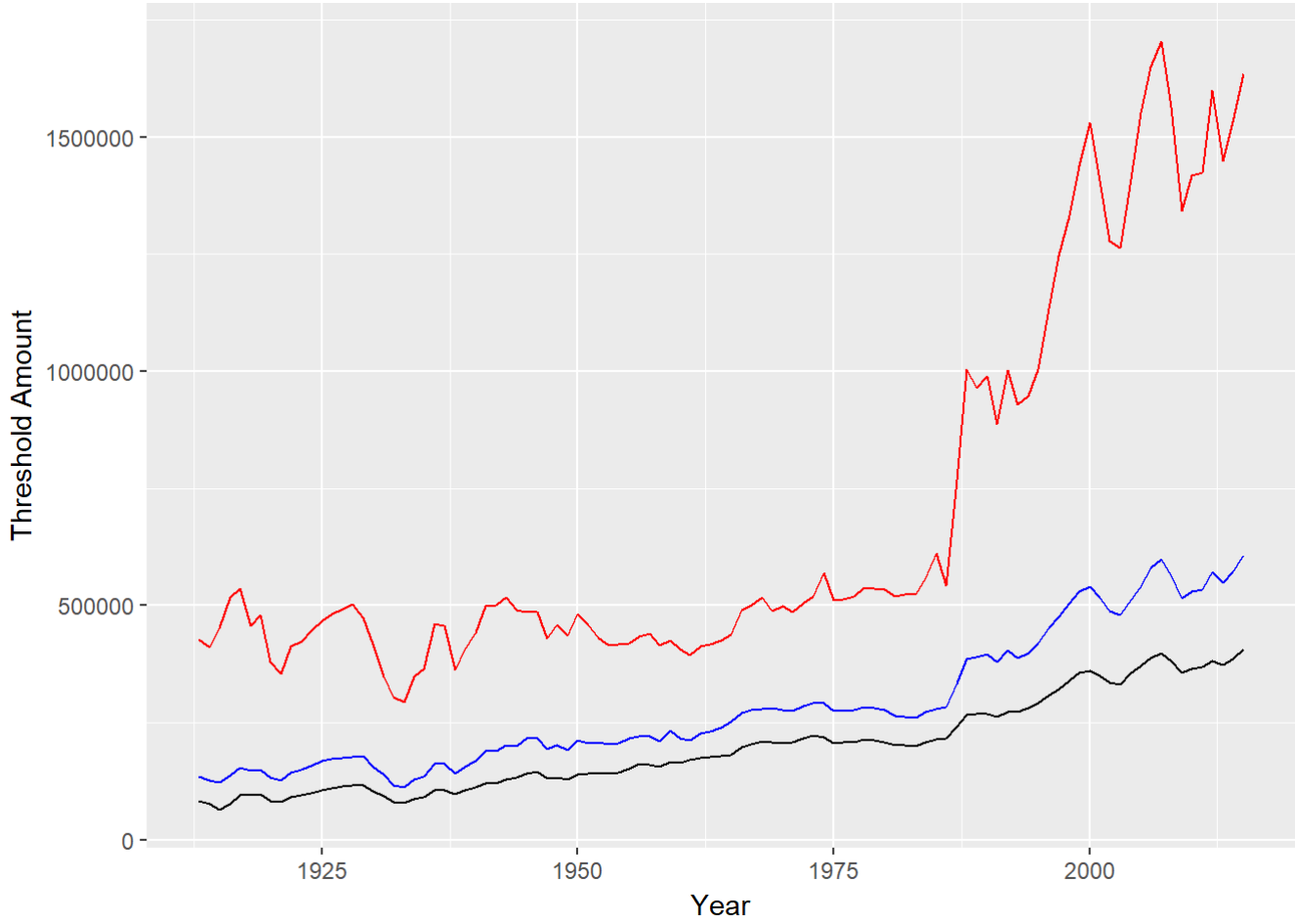
```
## [1] 189140.6
```

2. Plot the three percentile levels against time using `ggplot`. Make sure the axes are labeled appropriately, and in particular that the horizontal axis is labeled with years between 1913 and 2012, not just numbers from 1 to 100. Remember `library(ggplot2)`. In my plot I used multiple layers of `geom_line` and didn't include a lengend (but plotted the years in different colors).

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.4.2
```

```
ggplot(data = wtid) +
  geom_line(mapping = aes(x = Year, y = P99)) +
  geom_line(mapping = aes(x = Year, y = P99.5), color = "blue") +
  geom_line(mapping = aes(x = Year, y = P99.9), color = "red") +
  labs(main = "Thresholds for the Richest People Over Time", x = "Year", y = "Threshold Amoun
t")
```



3. It can be shown from the earlier equations that one can estimate the exponent by the formula

$$a = 1 - \frac{\log 10}{\log\left(\frac{P99}{P99.9}\right)}$$

Write a function, `exponent.est_ratio()` which takes in values for `P99` and `P99.9`, and returns the value of a implied by . Check that if `P99=1e6` and `P99.9=1e7`, your function returns an a of 2.

```
exponent.est_ratio <- function(p99, p999) {
  return(1 - log(10)/(log(p99/p999)))
}
exponent.est_ratio(1e6, 1e7)
```

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

4. Estimate a for each year in the data set, using your `exponent.est_ratio()` function. If the function was written properly, you should not need to use a loop. Plot your estimate of a over time using `ggplot`. Think about whether these results look reasonable. (Remember that smaller exponents mean more income inequality.)

```
ahat <- exponent.est_ratio(wtid$P99, wtid$P99.9)
ggplot(data = wtid) +
  geom_point(mapping = aes(x = Year, y = ahat))
```