

Midterm Project

Stat 133, Fall 2016, Prof. Sanchez

Xuanpei Ouyang

Set Up: Packages and Functions

You will need to use the following packages: "stringr", "ggplot2", "dplyr", and "readr". You can also use any other packages. If you don't have any of the packages installed in your computer, install them first (outside this Rmd).

```
# use this code chunk to load all the packages that you will be using  
# (do not include commands to install the packages)
```

```
library(stringr)  
library(ggplot2)  
library(dplyr)
```

```
##  
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':  
##  
##   filter, lag
```

```
## The following objects are masked from 'package:base':  
##  
##   intersect, setdiff, setequal, union
```

```
library(readr)
```

To “import” the functions in your R script file, use the function `source()`

```
# source your functions  
source("~/Desktop/STAT 133/stat133-my-repo/midterm_project/stat133-midterm-functions-template.R")
```

Import Raw Data in R

We are assuming that you already downloaded a copy of the CSV files with the raw data (you don't need to show the commands you used for this step):

- `womens-high-jump-raw.csv`
- `mens-high-jump-raw.csv`

Now, use the function `read_csv()` from the package "readr" to import the data sets in R.

```
# download.file(
#   url = 'https://raw.githubusercontent.com/ucb-stat133/stat133-fall-2016/
#   master/data/womens-high-jump-raw.csv',
#   destfile = '~/Desktop/STAT 133/stat133-my-repo/midterm_project/
#   womens-high-jump-raw.csv')

# read in womens-high-jump-raw.csv
womens_high_jump_raw = read_csv(
  '~/Desktop/STAT 133/stat133-my-repo/midterm_project/womens-high-jump-raw.csv',
  col_names = TRUE)
```

```
## Parsed with column specification:
## cols(
##   Height = col_character(),
##   Athlete = col_character(),
##   Date = col_character(),
##   Place = col_character()
## )
```

```
# check structure with str()
str(womens_high_jump_raw)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':   56 obs. of  4 variables:
## $ Height : chr  "1.46 m (4 ft 9 1/2 in)" "1.485 m (4 ft 10 1/2 in)" "1.485 m (4 ft 10 1/2 in)" "1.5
## $ Athlete: chr  "Nancy Voorhees (USA)" "Elizabeth Stine (USA)" "Sophie Elliott-Lynn (GBR)" "Phyllis (
## $ Date : chr  "20 May 1922" "26 May 1923" "6 August 1923" "11 July 1925" ...
## $ Place : chr  "Simsbury[1]" "Leonia[1]" "Brentwood[1]" "London[1]" ...
## - attr(*, "spec")=List of 2
## ..$ cols :List of 4
## .. ..$ Height : list()
## .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
## .. ..$ Athlete: list()
## .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
## .. ..$ Date : list()
## .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
## .. ..$ Place : list()
## .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
## ..$ default: list()
## .. ..- attr(*, "class")= chr  "collector_guess" "collector"
## ..- attr(*, "class")= chr "col_spec"
```

```
# download.file(
#   url = 'https://raw.githubusercontent.com/ucb-stat133/stat133-fall-2016/
#   master/data/mens-high-jump-raw.csv',
#   destfile = '~/Desktop/STAT 133/stat133-my-repo/midterm_project/
#   mens-high-jump-raw.csv')

# read in mens-high-jump-raw.csv
mens_high_jump_raw = read_csv(
  '~/Desktop/STAT 133/stat133-my-repo/midterm_project/mens-high-jump-raw.csv',
  col_names = TRUE)
```

```
## Parsed with column specification:
## cols(
##   Height = col_character(),
##   Athlete = col_character(),
##   Venue = col_character(),
##   Date = col_character()
## )
```

```
# check structure with str()
str(mens_high_jump_raw)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':   40 obs. of  4 variables:
## $ Height : chr  "2.00 m (6 ft 6 3/4 in)" "2.022 m (6 ft 7 5/8 in)" "2.038 m (6 ft 8 1/4 in)" "2.04 m (6 ft 8 1/2 in)"
## $ Athlete: chr  "George Horine (USA)" "Edward Beeson (USA)" "Harold Osborn (USA)" "Walter Marty (USA)"
## $ Venue : chr  "Palo Alto, California" "Berkeley, California" "Urbana, Illinois" "Fresno, California"
## $ Date : chr  "18 May 1912[1]" "2 May 1914[3]" "27 May 1924[4]" "13 May 1933[1]" ...
## - attr(*, "spec")=List of 2
## ..$ cols :List of 4
## .. ..$ Height : list()
## .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
## .. ..$ Athlete: list()
## .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
## .. ..$ Venue : list()
## .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
## .. ..$ Date : list()
## .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
## ..$ default: list()
## .. ..- attr(*, "class")= chr  "collector_guess" "collector"
## ..- attr(*, "class")= chr "col_spec"
```

Cleaning raw data of women's high jump world records

We'll start by cleaning the women's data set.

Column Height

To clean the contents of column `Height`, use your function to extract height values to generate a new **numeric** vector `height` with the values of meters:

```
# vector 'height' with numeric values of height in meters
height = extract_height(womens_high_jump_raw$Height)
```

Clean the column `height` by keeping the meter units and removing the values in US customary units.

Column Athlete

The column `Athlete` contains the name of the athlete, together with the country (inside parenthesis). Use your function to extract the athlete's name to create a new **character** vector `athlete`:

```
# vector 'athlete' with character values of athlete's name
athlete = extract_athlete(womens_high_jump_raw$Athlete)
```

Clean the column **athlete** by extracting the name of athlete from the **Athlete** column.

Likewise, use your function to extract the country names to create a new **character** vector **country** that contains the initials of the countries:

```
# character vector 'country' with country abbreviations
country = extract_country(womens_high_jump_raw$Athlete)
```

Extract the column **country** by extracting the country name inside parenthesis from the **Athlete** column.

Column Date

The column **Date** contains the dates when the records were established. From this column you have to create three new vectors: **day**, **month**, and **year**.

- Use your function to extract the day numbers to create a new **numeric** vector **day**:

```
# numeric vector 'day'
day = extract_day(womens_high_jump_raw$Date)
```

Extract the column **day** by extracting the first number (day) from the **Date** column.

- Use your function to extract the month names to create a new **character** vector **month**:

```
# character vector 'month'
month = extract_month(womens_high_jump_raw$Date)
```

Extract the column **month** by extracting the second number (month) from the **Date** column.

- Use your function to extract the year number to create a new **numeric** vector **year**:

```
# numeric vector 'year'
year = extract_year(womens_high_jump_raw$Date)
```

Extract the column **year** by extracting the third number (year) from the **Date** column.

Likewise, use your function to reformat date to create a new vector **new_date** by reformatting the values in column **Date**:

```
# 'new_date' vector of class "Date"
new_date = reformat_date(womens_high_jump_raw$Date)
```

Reformat the date to a new vector in year-month-day format.

Column Place

Use your function to remove brackets—and the content inside them— to create a new character vector **city** from column **Place**:

```
# character vector 'city' with name of city
city = remove_brackets(womens_high_jump_raw$Place)
```

Extract the column `city` by removing the brackets from `Place` column.

New vector `gender`

Create a **character** vector `gender` filled with "female" values, having length equal to the number of rows in the women's data set.

```
# vector 'gender'
gender = rep("female", nrow(womens_high_jump_raw))
```

New data frame `womens`

Use the vectors `height`, `athlete`, `gender`, `country`, `city`, `new_date`, `day`, `month`, and `year` to build a new data frame `womens`. The column corresponding to `new_date` should have name "date". Do NOT convert character strings as factors:

```
# data frame 'womens'
womens = data.frame(
  height = height,
  athlete = athlete,
  gender = gender,
  country = country,
  city = city,
  date = new_date,
  day = day,
  month = month,
  year = year,
  stringsAsFactors = FALSE
)

str(womens)
```

```
## 'data.frame':   56 obs. of  9 variables:
## $ height : num  1.46 1.49 1.49 1.52 1.55 ...
## $ athlete: chr   "Nancy Voorhees" "Elizabeth Stine" "Sophie Elliott-Lynn" "Phyllis Green" ...
## $ gender : chr   "female" "female" "female" "female" ...
## $ country: chr   "USA" "USA" "GBR" "GBR" ...
## $ city   : chr   "Simsbury" "Leonia" "Brentwood" "London" ...
## $ date   : Date, format: "1922-05-20" "1923-05-26" ...
## $ day    : num   20 26 6 11 2 6 3 5 18 12 ...
## $ month  : chr   "May" "May" "August" "July" ...
## $ year   : num  1922 1923 1923 1925 1926 ...
```

Now, I obtain a clean dataset for women's high jump world records with 9 columns which are height, athlete, gender, country, city, date, day, month and year. Also I convert all the data column to appropriate data type for manipulating and plotting data in the later section. —

Cleaning raw data of men's high jump world records

In this section you will clean the men's data set.

Column Height

To clean the contents of column `Height`, use your function to extract height values to generate a new **numeric** vector `height` with the values of meters:

```
# vector 'height' with numeric values of height in meters
height = extract_height(mens_high_jump_raw$Height)
```

Clean the column `height` by keeping the meter units and removing the values in US customary units.

Column Athlete

The column `Athlete` contains the name of the athlete, together with the country (inside parenthesis). Use your function to extract the athlete's name to create a new **character** vector `athlete`:

```
# vector 'athlete' with character values of athlete's name
athlete = extract_athlete(mens_high_jump_raw$Athlete)
```

Clean the column `athlete` by extracting the name of athlete from the `Athlete` column.

Likewise, use your function to extract the country names to create a new **character** vector `country` that contains the initials of the countries:

```
# character vector 'country' with country abbreviations
country = extract_country(mens_high_jump_raw$Athlete)
```

Extract the column `country` by extracting the country name inside parenthesis from the `Athlete` column.

Column Venue

The column `Venue` contains the name of the city where the record was established. As you can tell from the values in this column, some names contain more than just the name of the city (e.g. some include US State, some include name of country). Use your function to extract the name of the city in order to obtain a new **character** vector `city`:

```
# character vector 'city' with name of city
city = extract_city(mens_high_jump_raw$Venue)
```

Column Date

The column `Date` contains the dates when the records were established. As you can tell from the values in this column, they also contain extra characters (numbers inside brackets).

Use your function to remove brackets—and the content inside them—to create a new character vector `clean_date`:

```
# vector 'clean_date'
clean_date = remove_brackets(mens_high_jump_raw$Date)
```

Clean the `Date` by removing the brackets followed the data of date

Now take `clean_date` to create three new vectors: `day`, `month`, and `year`.

- Use your function to extract the day numbers to create a new **numeric** vector `day`:

```
# numeric vector 'day'
day = extract_day(clean_date)
```

Extract the column `day` by extracting the first number (day) from the `Date` column.

- Use your function to extract the month names to create a new **character** vector `month`:

```
# character vector 'month'
month = extract_month(clean_date)
```

Extract the column `month` by extracting the second number (month) from the `Date` column.

- Use your function to extract the year number to create a new **numeric** vector `year`:

```
# numeric vector 'year'
year = extract_year(clean_date)
```

Extract the column `year` by extracting the third number (year) from the `Date` column.

Likewise, use your function to reformat date to create a new vector `new_date` by reformatting the values in column `Date`:

```
# 'new_date' vector of class "Date"
new_date = reformat_date(clean_date)
```

Reformat the date to a new vector in year-month-day format.

New vector gender

Create a **character** vector `gender` filled with "male" values, having length equal to the number of rows in the men's data set.

```
# vector 'gender'
gender = rep("male", nrow(mens_high_jump_raw))
```

New data frame mens

Use the vectors `height`, `athlete`, `gender`, `country`, `city`, `new_date`, `day`, `month`, and `year` to build a new data frame `mens`. The column corresponding to `new_date` should have name "date". Do NOT convert character strings as factors:

```
# data frame 'mens'
mens = data.frame(
  height = height,
  athlete = athlete,
  gender = gender,
  country = country,
  city = city,
  date = new_date,
  day = day,
  month = month,
  year = year,
  stringsAsFactors = FALSE
)

str(mens)
```

```
## 'data.frame': 40 obs. of 9 variables:
## $ height : num 2 2.02 2.04 2.04 2.06 ...
## $ athlete: chr "George Horine" "Edward Beeson" "Harold Osborn" "Walter Marty" ...
## $ gender : chr "male" "male" "male" "male" ...
## $ country: chr "USA" "USA" "USA" "USA" ...
## $ city : chr "Palo Alto" "Berkeley" "Urbana" "Fresno" ...
## $ date : Date, format: "1912-05-18" "1914-05-02" ...
## $ day : num 18 2 27 13 28 12 12 12 17 27 ...
## $ month : chr "May" "May" "May" "May" ...
## $ year : num 1912 1914 1924 1933 1934 ...
```

Here, I obtain a clean dataset for men's high jump world records with 9 columns which are height, athlete, gender, country, city, date, day, month and year. Also, I convert all the data column to appropriate data type for manipulating and plotting data in the later section.

Data Manipulation

Create a new data frame `records` by merging (or “stacking”) the data frames `womens` and `mens`. Only the columns `gender` and `country` should be converted as R factors. Make sure that the column `date` is of class `"Date"`.

```
# new data frame 'records'
records = rbind(womens, mens)
records$gender = as.factor(records$gender)
records$country = as.factor(records$country)

str(records)
```

```
## 'data.frame': 96 obs. of 9 variables:
## $ height : num 1.46 1.49 1.49 1.52 1.55 ...
## $ athlete: chr "Nancy Voorhees" "Elizabeth Stine" "Sophie Elliott-Lynn" "Phyllis Green" ...
## $ gender : Factor w/ 2 levels "female","male": 1 1 1 1 1 1 1 1 1 ...
## $ country: Factor w/ 17 levels "AUT","BUL","CAN",...: 17 17 7 7 7 3 10 3 10 10 ...
```



```
## $ city : chr "Simsbury" "Leonia" "Brentwood" "London" ...
## $ date : Date, format: "1922-05-20" "1923-05-26" ...
## $ day : num 20 26 6 11 2 6 3 5 18 12 ...
## $ month : chr "May" "May" "August" "July" ...
## $ year : num 1922 1923 1923 1925 1926 ...
```

I merge the two data frame for women and men to create a new data frame called records, which contains the same columns as men's and women's dataset. Here, I convert the **gender** and **country** to R factors for plotting purpose.

Use functions in "dplyr" to compute the following frequency tables. Assign each table to its own object.

Frequency Table for number of records per country

```
# number of records per country
# object: per_country
per_country = records %>%
  group_by(country) %>%
  summarise(records = n())

# invoke head() on per_country
head(per_country)
```

```
## # A tibble: 6 × 2
##   country records
##   <fctr>   <int>
## 1     AUT         1
## 2     BUL         5
## 3     CAN         2
## 4     CHN         4
## 5     CUB         3
## 6     FRG         4
```

per_country is a frequency table containing the numbers of high jump records per country from year 1912 to 1993 sorted by the alphabetic order of country name.

```
# number of records per country in descending order
# object: per_country_desc
per_country_desc = per_country %>%
  arrange(desc(records))

# invoke head() on per_country_desc
head(per_country_desc)
```

```
## # A tibble: 6 × 2
##   country records
##   <fctr>   <int>
## 1     USA        24
## 2     URS        15
## 3     ROM        14
## 4     GDR         8
## 5     GBR         6
## 6     BUL         5
```

per_country_desc is a frequency table containing the numbers of high jump records per country from year 1912 to 1993 sorted by the descending order of records counts. And we can see that USA has the most high jump records during 1912 and 1993.

```
# number of records per year in descending order
# object: per_year_desc
per_year_desc = records %>%
  group_by(year) %>%
  summarise(records = n()) %>%
  arrange(desc(records))

# invoke head() on per_year_desc
head(per_year_desc)
```

```
## # A tibble: 6 × 2
##   year records
##   <dbl>   <int>
## 1  1961       7
## 2  1960       6
## 3  1958       5
## 4  1977       5
## 5  1983       5
## 6  1956       4
```

per_year_desc is a frequency table containing the numbers of high jump records per year from year 1912 to 1993 sorted by the descending order of records counts. And we can see that the highest number of high jump records were made in 1961.

```
# number of records by gender
# object: by_gender
by_gender = records %>%
  group_by(gender) %>%
  summarise(records = n())

# invoke head() on by_gender
head(by_gender)
```

```
## # A tibble: 2 × 2
##   gender records
##   <fctr>   <int>
## 1 female    56
## 2  male     40
```

by_gender is a frequency table containing the numbers of high jump records separated by gender from year 1912 to 1993 and sorted by the descending order of records counts.

Here is one more challenge that involves some data wrangling with year. The objective is to compute the number of records per decade (regardless of gender):

```
# number of records per decade (1920-1929, 1930-1939, 1940-1949, etc)
# object: per_decade
per_year = records %>%
  group_by(year) %>%
```

```

summarise(records = n()) %>%
  arrange(year)

intervals = seq(from = 1910, to = 2000, by = 10)
decade_records = integer(9)

# Use for loop to select appropriate years and count the sum of number
# of records in those selected years
for (i in 1:length(decade_records)){
  filtered_year = filter(per_year, year >= intervals[i] , year < intervals[i+1])
  filtered_records = filtered_year[,2]
  decade_records[i] = sum(filtered_records)
}

decade_names = c("1910-1919", "1920-1929", "1930-1939", "1940-1949", "1950-1959",
  "1960-1969", "1970-1979", "1980-1989", "1990-1999")

# create a data frame with appropriate names per_decade
per_decade = data.frame(
  decade = decade_names,
  records = decade_records,
  stringsAsFactors = FALSE
)
# invoke head() on per_decade
head(per_decade)

```

```

##      decade records
## 1 1910-1919        2
## 2 1920-1929       10
## 3 1930-1939        9
## 4 1940-1949        4
## 5 1950-1959       16
## 6 1960-1969       16

```

`per_decade` is a frequency table containing the number of records per decade regardless of gender and country and is sorted by earliest to latest year. The year interval used here is 1910-1929, 1920-1929, 1930-1939, 1940-1949, 1950-1959, 1960-1969, 1970-1979, 1980-1989, 1990-1999. And, we can see that the number of high jump world records made during 1980-1989 is the highest.

Data Visualization

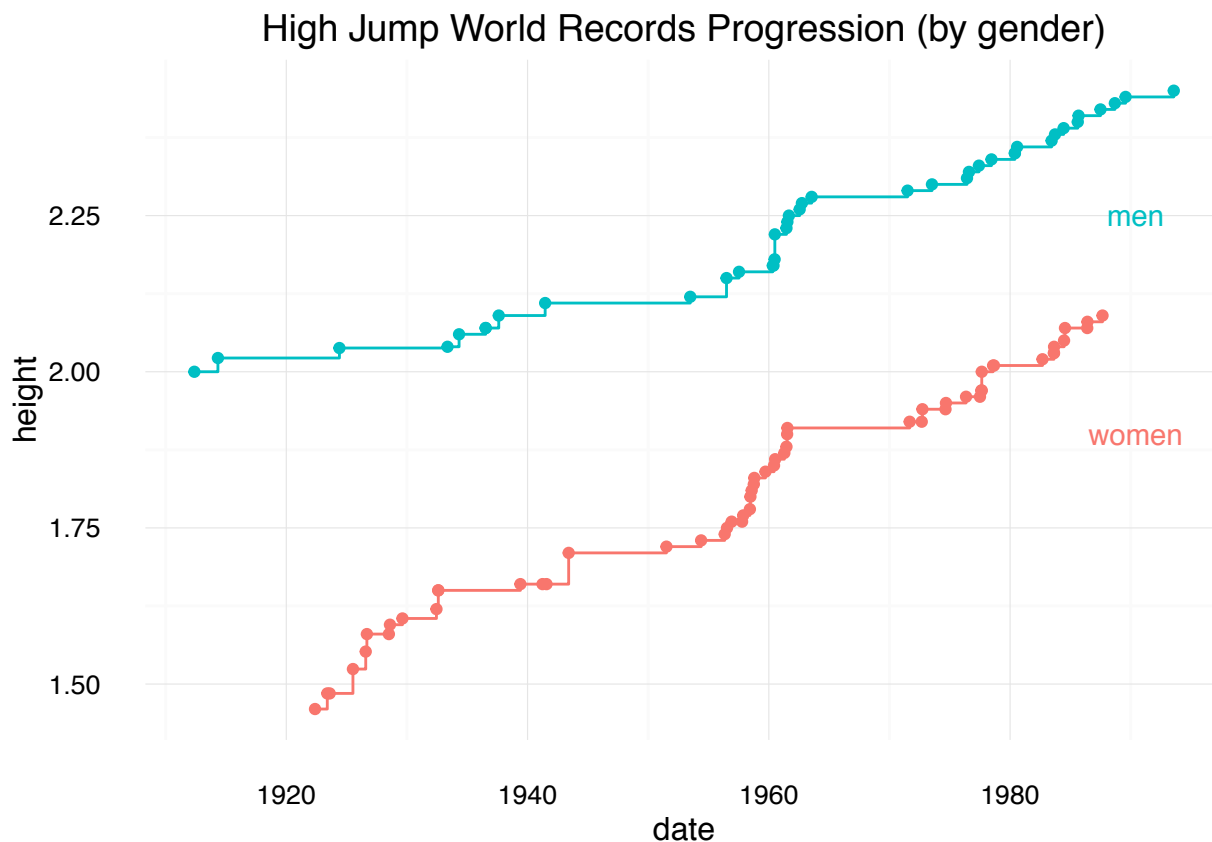
Use the `records` data frame, and functions in "ggplot2", to create charts similar to those displayed in the PDF with the instructions for this project:

- Include a title on all graphics
- Include a description for each graphic

```

# step-line plot, with points, showing the progression of records
# (lines colored by gender)
# (the default ggplot color values are: '#00BFC4', '#F8766D')
ggplot(records, aes(x = date, y = height, group = gender)) +
  geom_point(aes(color = gender), show.legend = FALSE) +
  geom_step(aes(color = gender), show.legend = FALSE) +
  annotate("text", x = as.Date("1990-06-01"), y = 1.90, label = "women",
          color = "#F8766D") +
  annotate("text", x = as.Date("1990-06-01"), y = 2.25, label = "men",
          color = "#00BFC4") +
  ggtitle("High Jump World Records Progression (by gender)") +
  theme_minimal()

```

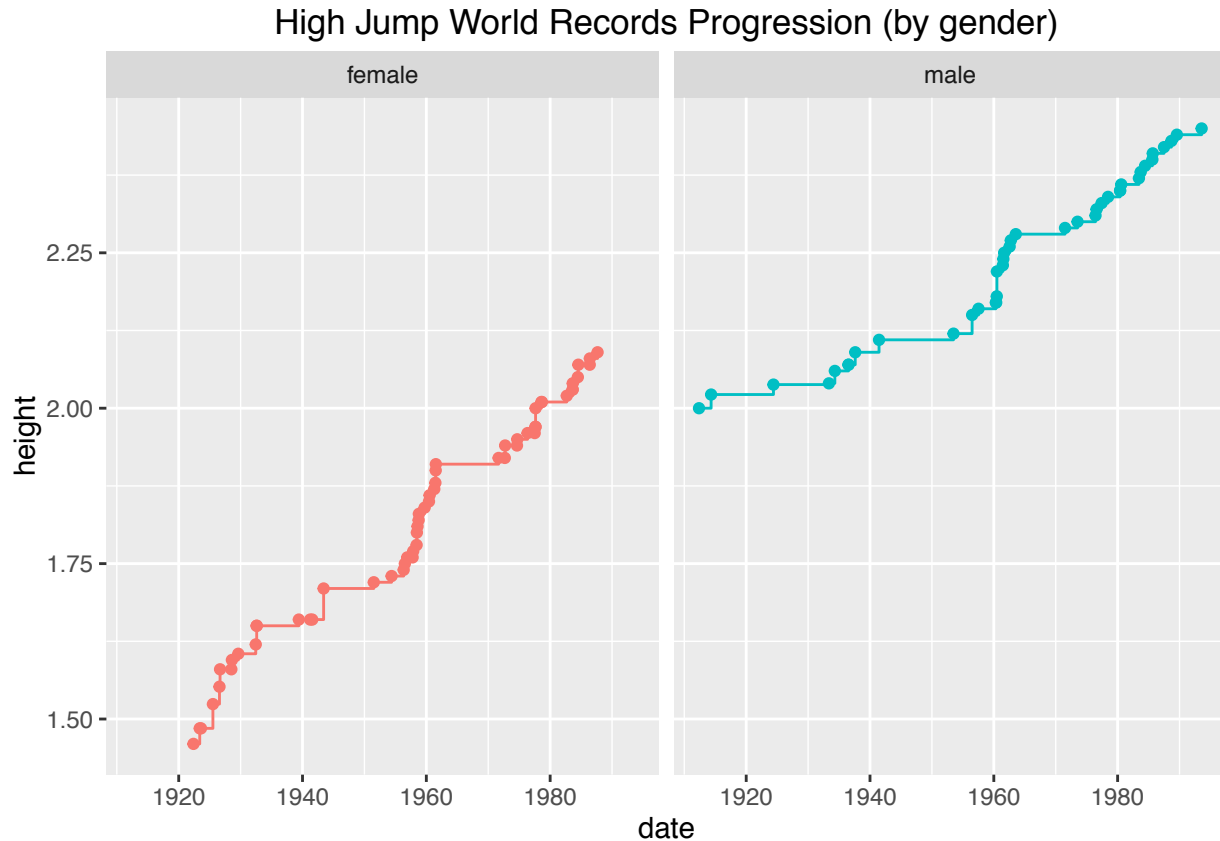


High Jump World Records Progression (by gender)

The plot above is a step-line graph, with points, of the progression of the high jump world records by gender. This graph shows the relationship between world records and the time stamp of the creation of world records. Based on the dataset of high jump world records for men and women, I plots each data point, joints them up with a step line and also color them seperately. From the plot, we can see that the overall pattern between the world records value and date has a postivie association. Also, these two variables has approximately linear association. Because of the special properties of world records that it can only increase, the relationship between world record and the time is mostly likely to be positive. For both men's and women's high jump world records, we can see that world records for both men and women plateaued several times, especially from 1941 to 1952 and from 1961 to 1971, then increase rapidly after the plateau possibly due to some breakthroughs in high jump technique. Also, the progression step-line for men's world records is absolutely

above the progression step-line for women's world records indicates that men may have advantage in high jump because of some physical reasons.

```
# step-line plot, with points, showing the progression of records
# (facetting by gender)
ggplot(records, aes(x = date, y = height)) +
  geom_point(aes(color = gender), show.legend = FALSE) +
  geom_step(aes(color = gender), show.legend = FALSE) +
  ggtitle("High Jump World Records Progression (by gender)") +
  facet_grid(~ gender)
```

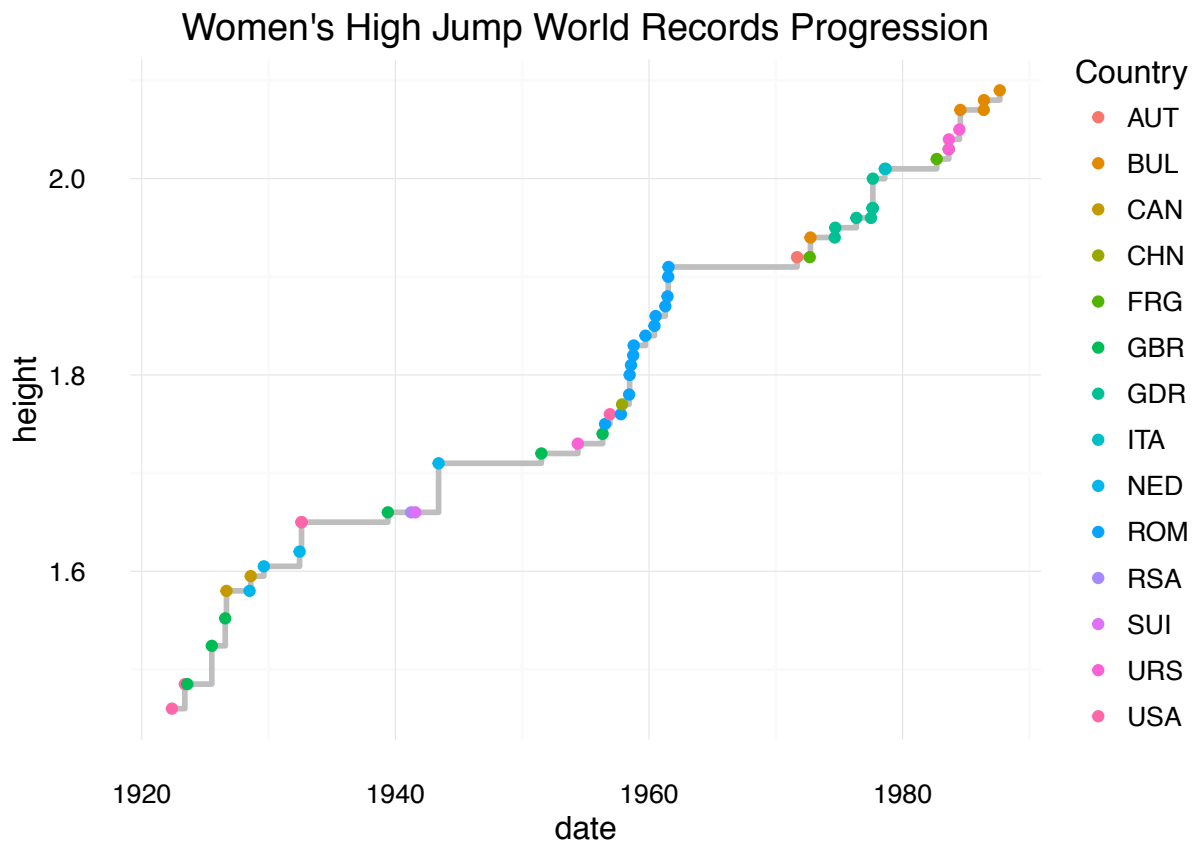


High Jump World Records Progression (by gender)

The plot is a step-line graph, with points, for showing the progression of high jump world records by gender. Here, I sliced the dataset by gender and plot the relationship between world records and date by gender in sub-panels in the same plot. From the plot, we can compare the progression of men's and women's high jump world records over time easily. The overall trends of their progressions are both positive but men's world records is always higher than women's world records. Also, by the end of 1977, women's world record reach men's first world records created around 1912.

```
# step-line plot, with points, showing the progression of women records
# (points colored by country)
womens = filter(records, gender == "female")
Country = womens$country
ggplot(womens, aes(x = date, y = height)) +
  geom_step(color = "gray", size = 1) +
```

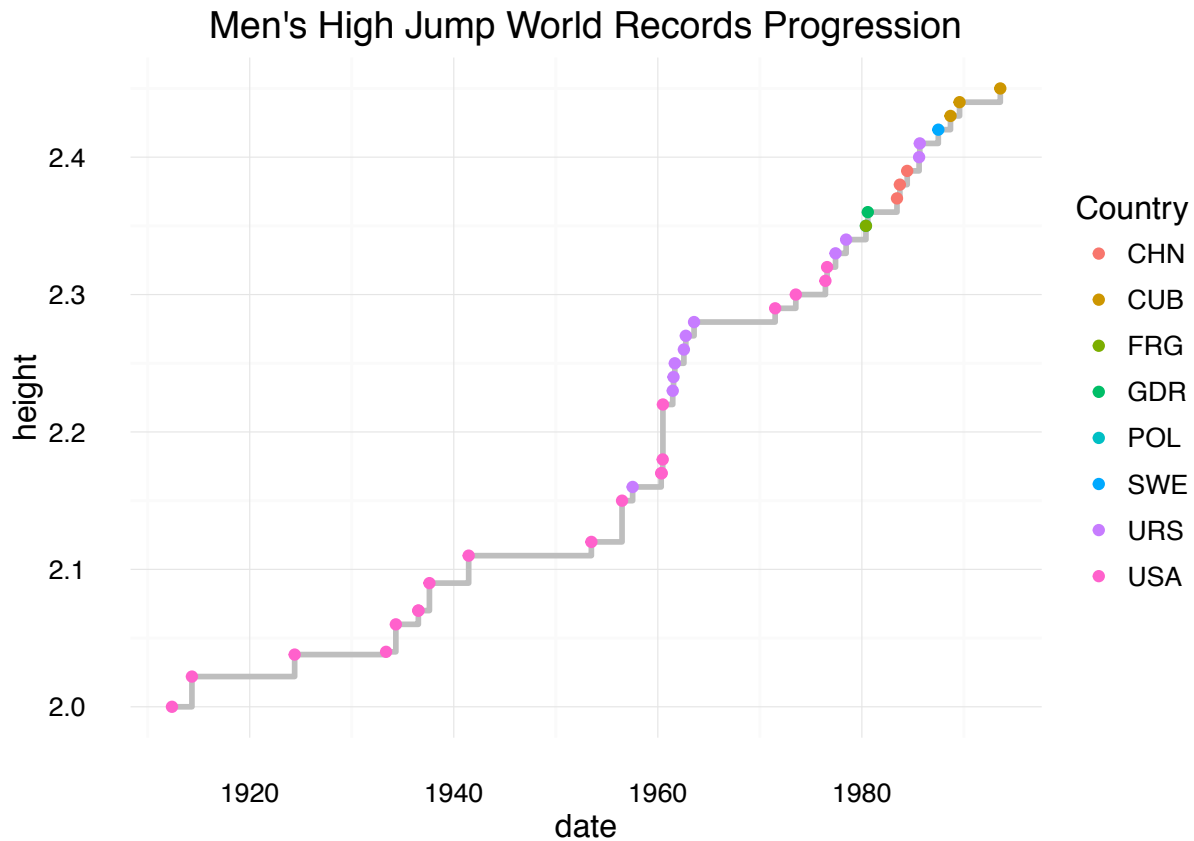
```
geom_point(aes(color = Country)) +
ggtitle("Women's High Jump World Records Progression") +
theme_minimal()
```



Women's High Jump World Records Progression

This plot is a step-line graph, with points, showing the progression of women records. Also, in this plot, I made data points colored by country to visualize the corresponding country for each record. From the plot, we can see that world records are spread among different countries. And in some small time interval, athletes from the same country created world records consecutively, such as Romanian from 1958 to 1962, Great Britain from 1975 to 1980.

```
# step-line plot, with points, showing the progression of men records
# (points colored by country)
mens = filter(records, gender == "male")
Country = mens$country
ggplot(mens, aes(x = date, y = height)) +
  geom_step(color = "gray", size = 1) +
  geom_point(aes(color = Country)) +
  ggtitle("Men's High Jump World Records Progression") +
  theme_minimal()
```



Men's High Jump World Records Progression

This plot is a step-line graph, with points, showing the progression of men records. Also, I made data points colored by country to visualize the corresponding country for each record. We can see that before year 1960, most of the world records were created by USA athletes and during year 1960 to 1967, most of the world records were created by Russia athletes. And overall, before 1980, men's world records only came from USA and Russia and after that, other countries such as China, Cuba, West Germany, East Germany, Poland, Soviet Union and Sweden.

Model Fitting

Now let's do some basic model fitting with a simple regression analysis.

Women's regression model

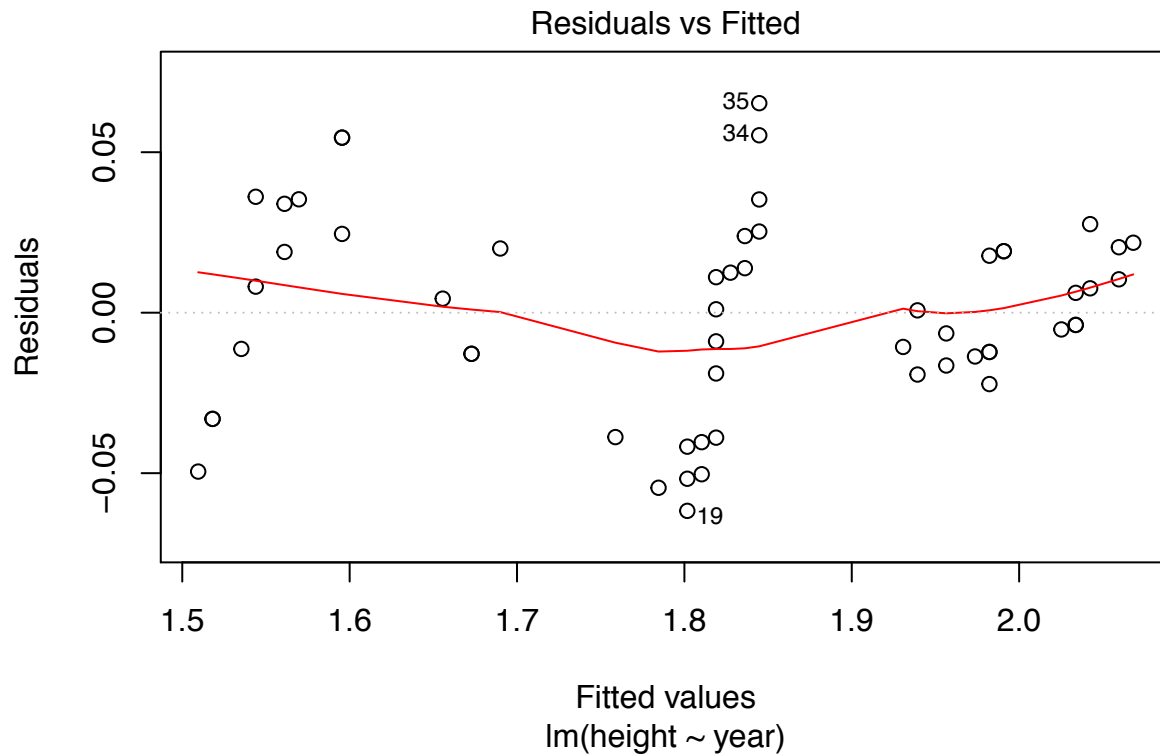
Subset the data `records` for those observations of gender `female`, and use the function `lm()` to compute a linear model by regressing `height` on `year`:

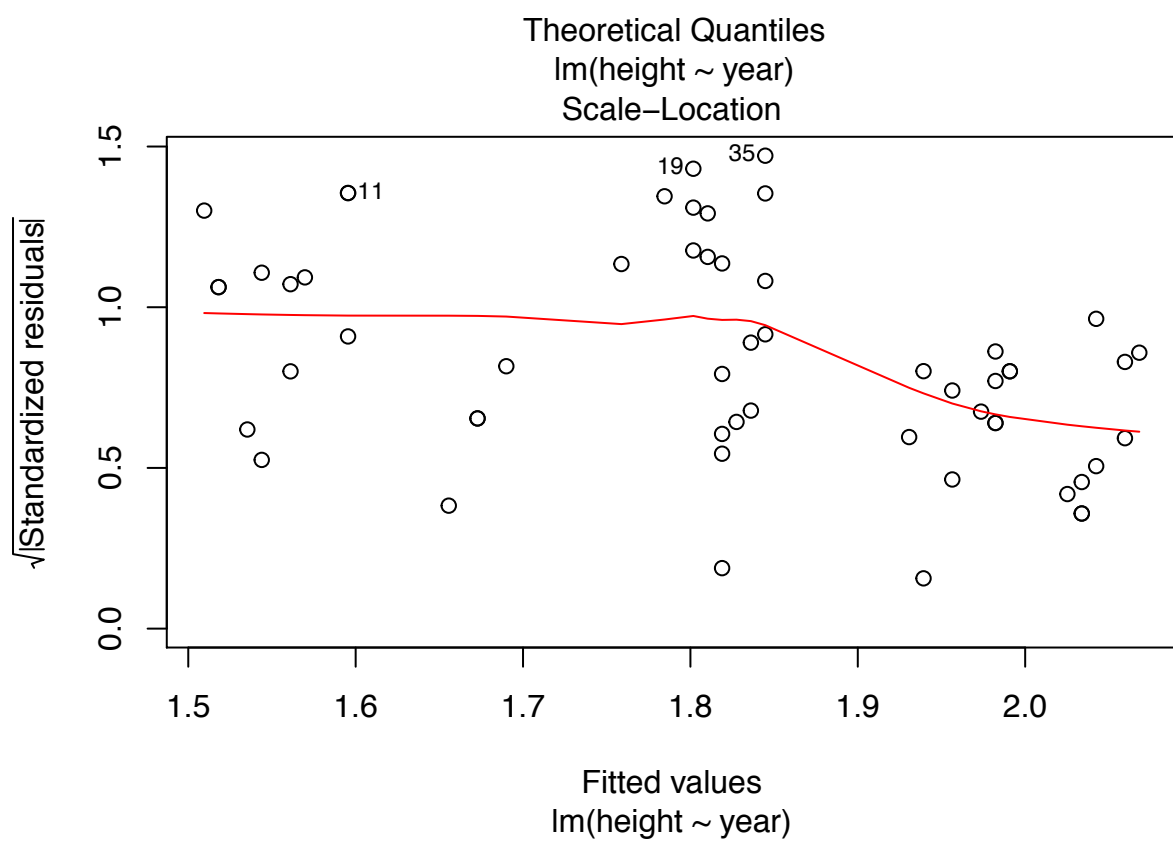
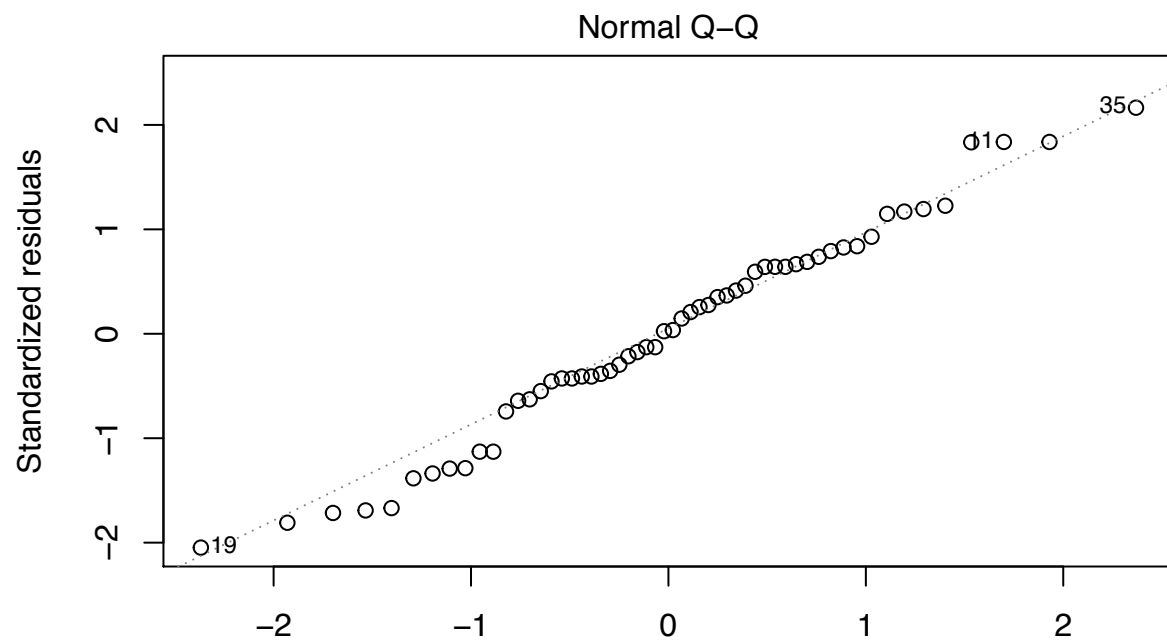
```
# Model for women
# regression of 'height' on 'year'
women = subset(records, gender == "female")
```

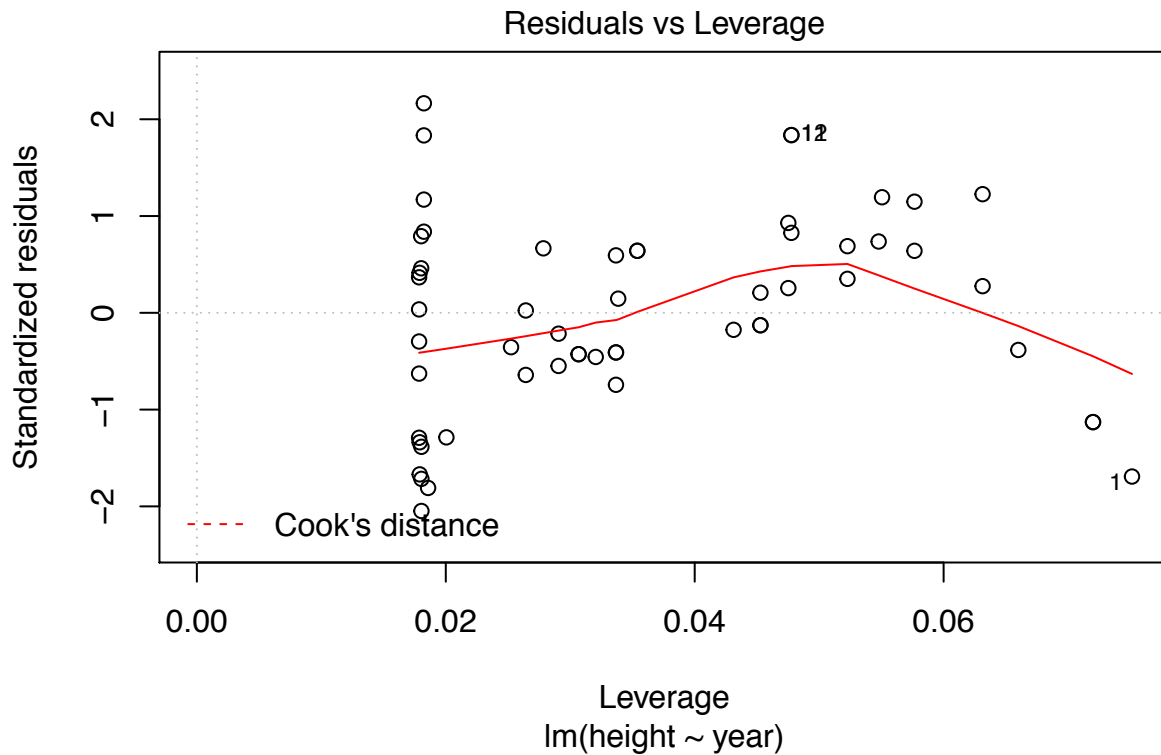
```
height = women$height
year = women$year
height_on_year_female = lm(height ~ year, data = women)
height_on_year_female
```

```
##
## Call:
## lm(formula = height ~ year, data = women)
##
## Coefficients:
## (Intercept)      year
## -15.010584    0.008595
```

```
plot(height_on_year_female)
```





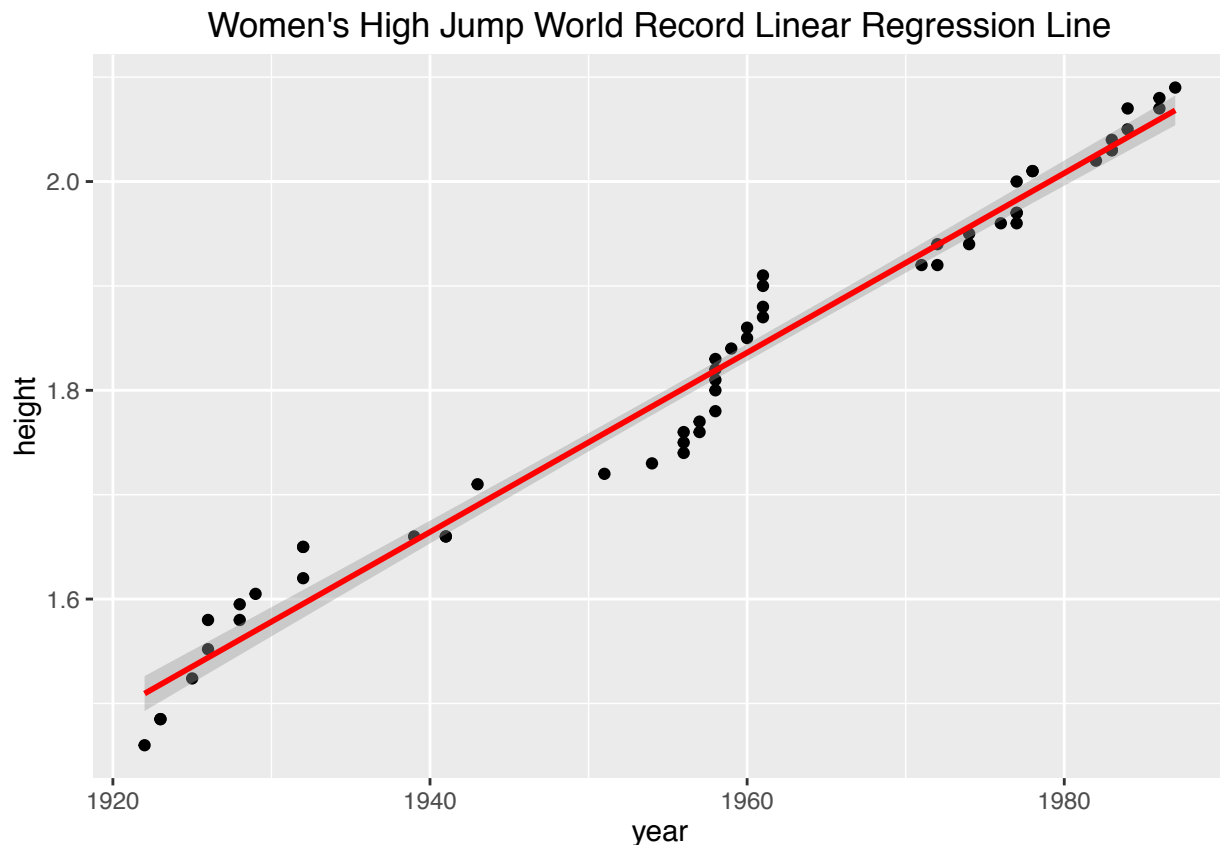


Use the object "lm" object in the previous step to compute a “prediction” of what the women’s high jump world record could have been in years: 2000, 2004, 2008, 2012, and 2016.

```
# predictions for olympic years
predict_year = c(2000, 2004, 2008, 2012, 2016)
years_for_prediction = data.frame(
  year = predict_year)
predict_height_women = predict.lm(height_on_year_female, years_for_prediction)
prediction_women = data.frame(
  year = predict_year,
  predict_height = predict_height_women
)
prediction_women
```

```
##   year predict_height
## 1 2000      2.179933
## 2 2004      2.214314
## 3 2008      2.248695
## 4 2012      2.283076
## 5 2016      2.317457
```

```
ggplot(womens, aes(x = year, y = height)) +
  geom_point() +
  ggtitle("Women's High Jump World Record Linear Regression Line") +
  stat_smooth(method = "lm", col = "red")
```



I compute a linear model for women's high jump world record by regressing `height` on `year`. Then I use the computed linear model to predict the women's high jump world record and the prediction output is displayed in the above table. Also, to visualize how well the regression line fits, I plot the regression line on scatter plot. We can see that the regression line fits well with the original dataset.

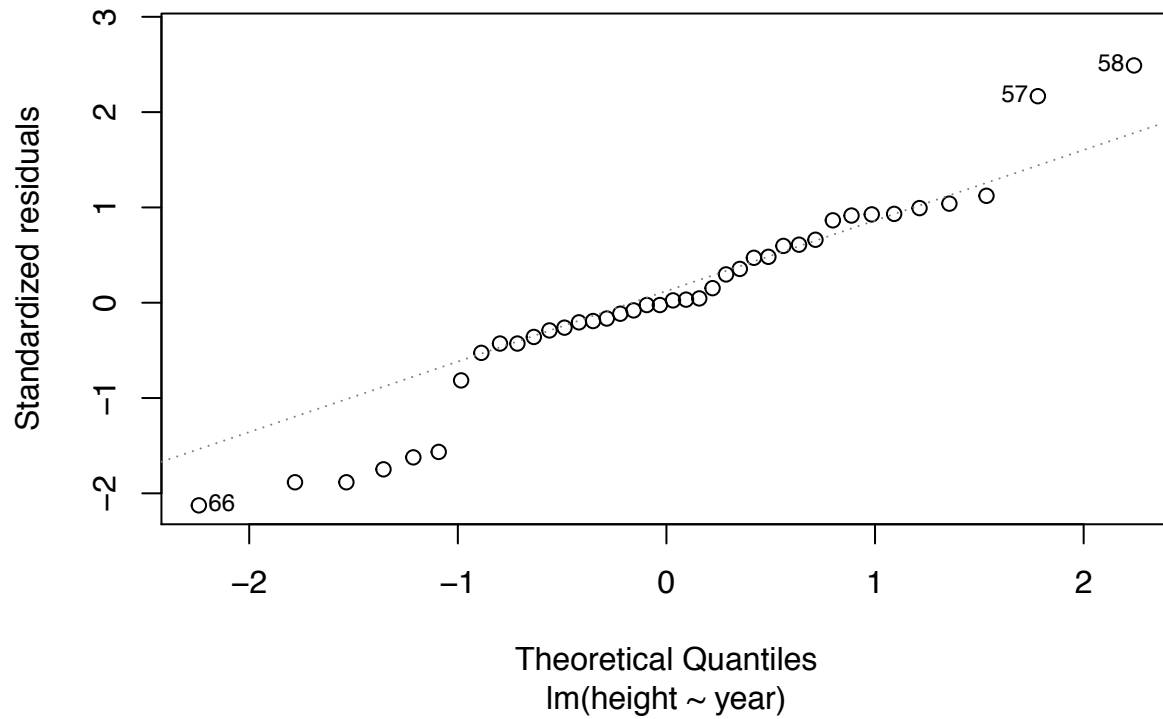
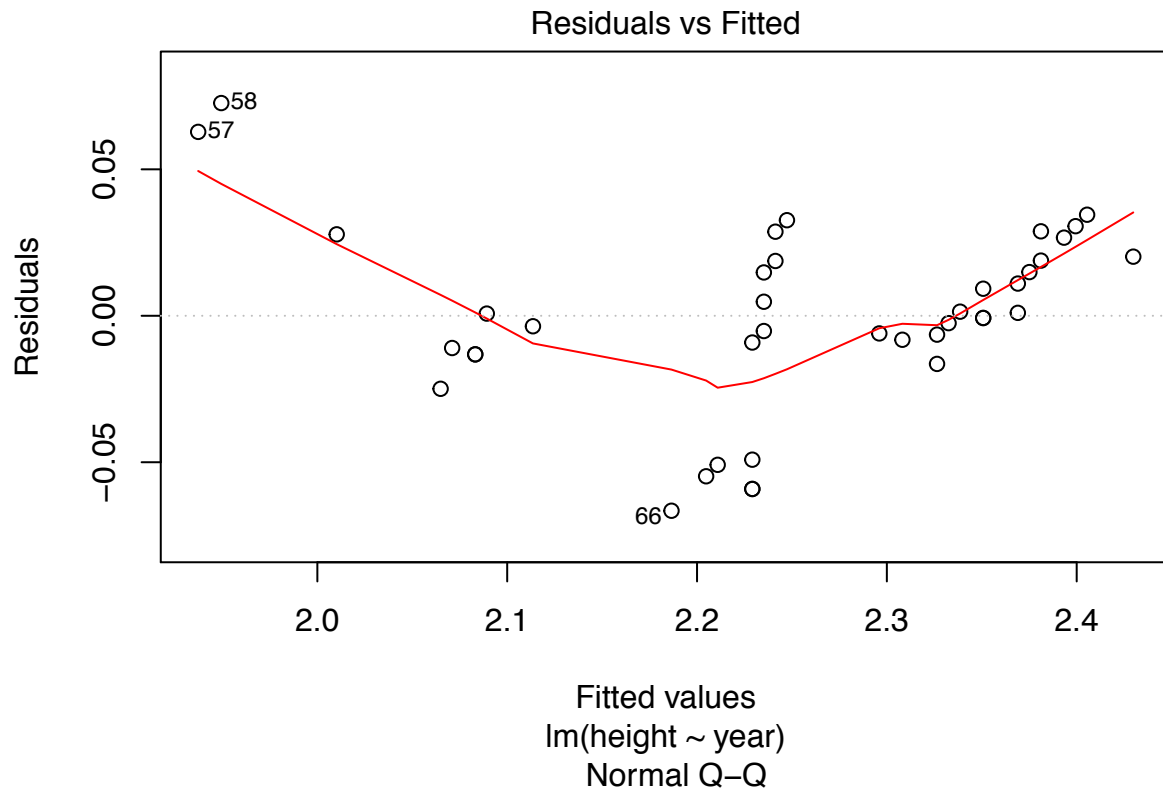
Men's regression model

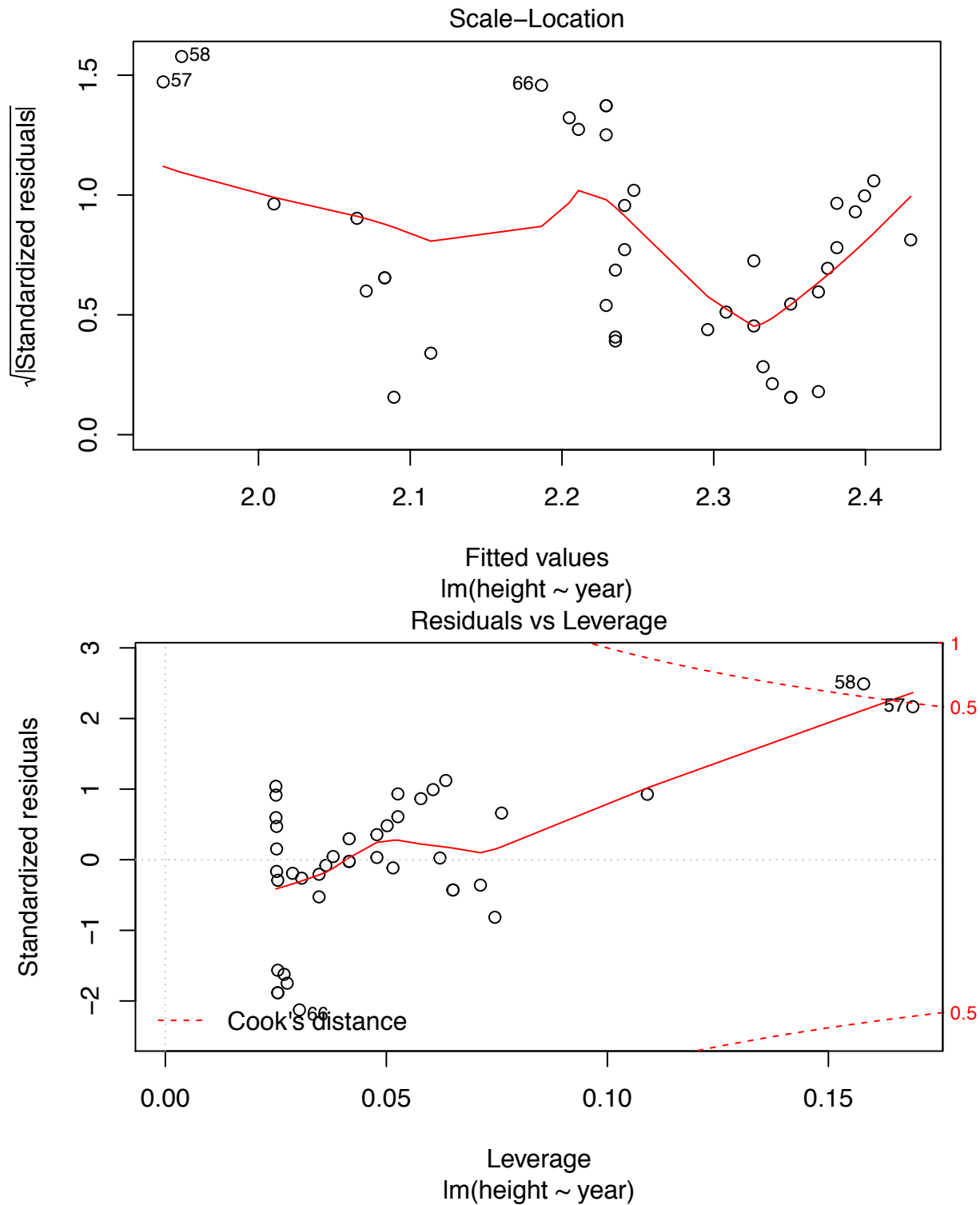
Subset the data `records` for those observations of gender `male`, and use the function `lm()` to compute a linear model by regressing `height` on `year`:

```
# Model for men
# regression of 'height' on 'year'
men = subset(records, gender == "male")
height = men$height
year = men$year
height_on_year_male = lm(height ~ year, data = men)
height_on_year_male
```

```
##
## Call:
## lm(formula = height ~ year, data = men)
##
## Coefficients:
## (Intercept)      year
##   -9.690620    0.006082
```

```
plot(height_on_year_male)
```





Use the object "lm" object in the previous step to compute a "prediction" of what the men's high jump world record could have been in years: 2000, 2004, 2008, 2012, and 2016.

```
# predictions for olympic years
years_for_prediction = data.frame(
  year = predict_year)
```

```

predict_height_men = predict.lm(height_on_year_male, years_for_prediction)
prediction_men = data.frame(
  year = predict_year,
  predict_height = predict_height_men
)
prediction_men

```

```

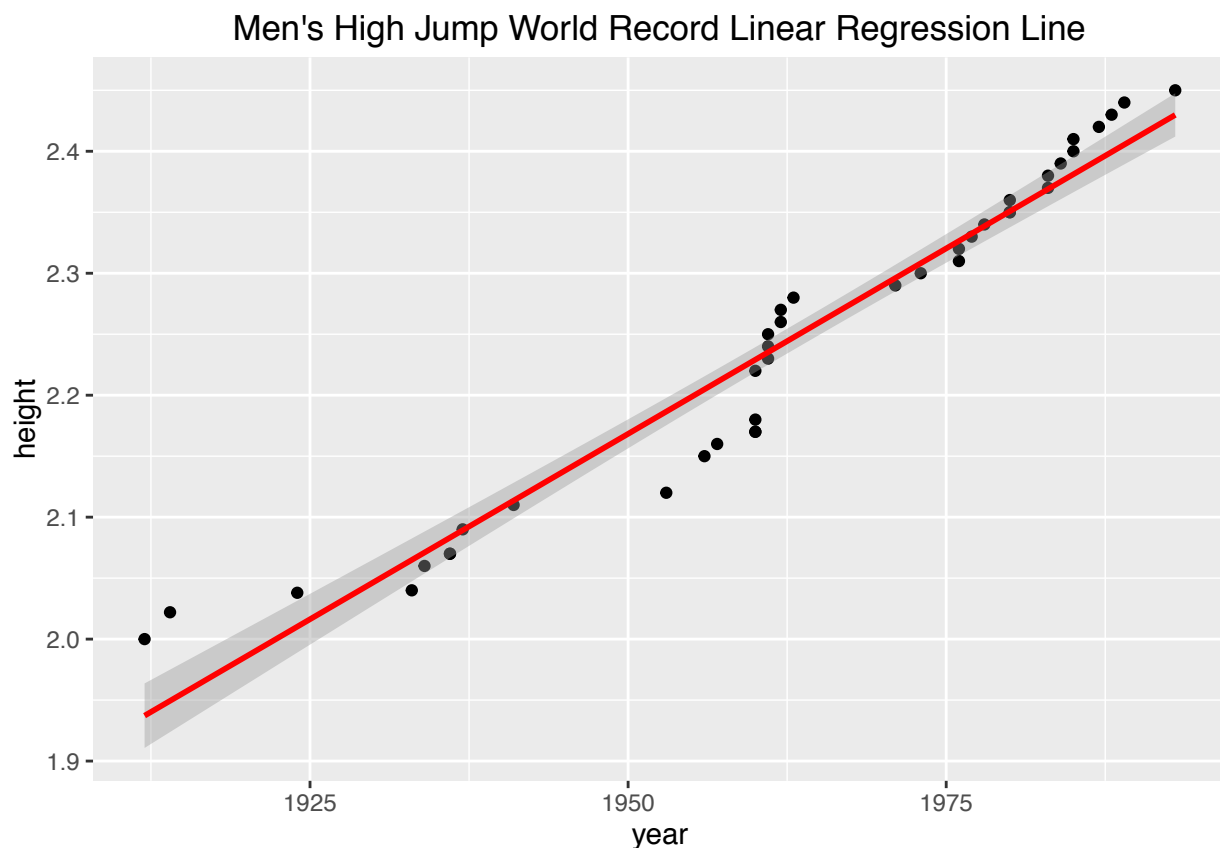
##   year predict_height
## 1 2000      2.472384
## 2 2004      2.496710
## 3 2008      2.521036
## 4 2012      2.545362
## 5 2016      2.569688

```

```

ggplot(mens, aes(x = year, y = height)) +
  geom_point() +
  ggtitle("Men's High Jump World Record Linear Regression Line") +
  stat_smooth(method = "lm", col = "red")

```



I compute a linear model for men's high jump world record by regressing `height` on `year`. Then I use the computed linear model to predict the men's high jump world record and the prediction output is displayed in the above table. Also, to visualize how well the regression line fits, I plot the regression line on scatter plot. We can see that the regression line fits well with the original dataset.

Extra Credit

This part of the project is optional. If you provide a satisfactory solution, you will get extra credit.

Consider the first women's world record which is 1.46 meters. This value corresponds to "4 ft 9 1/2 in". Likewise, a height value of 2.04 meters can be expressed in US customary units as "6 ft 8 3/8 in".

The goal is to write a function that takes a height value (in meters), and which returns a character vector in US customary units. In other words, if you pass the value 1.46 to your function, the output will be the string "4 ft 9 1/2 in".

You should be able to apply your function on the entire vector `height` to get the corresponding strings in US customary units. Your results should match the values in wikipedia's html tables.

Don't write a long function. You can use as many auxiliary short functions as necessary. All the functions must be well documented with a general description for the function's purpose, the arguments, and the output. Include these functions in the same .R script file with the other functions (those used to clean the columns of the raw data sets).

Your code should be accompanied with descriptions of how you approached this problem, and the decisions you made to implement it in that way.

```
# Code for testing
height_in_meters = records$height
converted_height_string = character(length(height_in_meters))

for (i in 1:length(height_in_meters)){
  converted_height_string[i] =
    convert_height_units_string(height_in_meters[i])
}

converted_height_string
```

```
## [1] "4 ft 9 1/2 in" "4 ft 10 1/2 in" "4 ft 10 1/2 in" "5 ft 0 in"
## [5] "5 ft 1 1/8 in" "5 ft 2 1/4 in" "5 ft 2 1/4 in" "5 ft 2 3/4 in"
## [9] "5 ft 3 1/4 in" "5 ft 3 3/4 in" "5 ft 5 in" "5 ft 5 in"
## [13] "5 ft 5 3/8 in" "5 ft 5 3/8 in" "5 ft 5 3/8 in" "5 ft 7 3/8 in"
## [17] "5 ft 7 3/4 in" "5 ft 8 1/8 in" "5 ft 8 1/2 in" "5 ft 8 7/8 in"
## [21] "5 ft 9 1/4 in" "5 ft 9 1/4 in" "5 ft 9 5/8 in" "5 ft 10 1/8 in"
## [25] "5 ft 10 7/8 in" "5 ft 11 1/4 in" "5 ft 11 5/8 in" "6 ft 0 in"
## [29] "6 ft 0 1/2 in" "6 ft 0 7/8 in" "6 ft 1 1/4 in" "6 ft 1 5/8 in"
## [33] "6 ft 2 in" "6 ft 2 3/4 in" "6 ft 3 1/4 in" "6 ft 3 5/8 in"
## [37] "6 ft 3 5/8 in" "6 ft 4 3/8 in" "6 ft 4 3/8 in" "6 ft 4 3/4 in"
## [41] "6 ft 5 1/8 in" "6 ft 5 1/8 in" "6 ft 5 1/2 in" "6 ft 5 1/2 in"
## [45] "6 ft 6 3/4 in" "6 ft 7 1/8 in" "6 ft 7 1/8 in" "6 ft 7 1/2 in"
## [49] "6 ft 7 7/8 in" "6 ft 7 7/8 in" "6 ft 8 3/8 in" "6 ft 8 3/4 in"
## [53] "6 ft 9 1/2 in" "6 ft 9 1/2 in" "6 ft 9 7/8 in" "6 ft 10 1/4 in"
## [57] "6 ft 6 3/4 in" "6 ft 7 5/8 in" "6 ft 8 1/4 in" "6 ft 8 3/8 in"
## [61] "6 ft 9 1/8 in" "6 ft 9 1/2 in" "6 ft 9 1/2 in" "6 ft 10 1/4 in"
## [65] "6 ft 11 1/8 in" "6 ft 11 1/2 in" "7 ft 0 5/8 in" "7 ft 1 in"
## [69] "7 ft 1 3/8 in" "7 ft 1 3/8 in" "7 ft 1 7/8 in" "7 ft 3 3/8 in"
## [73] "7 ft 3 3/4 in" "7 ft 4 1/4 in" "7 ft 4 5/8 in" "7 ft 5 in"
## [77] "7 ft 5 3/8 in" "7 ft 5 3/4 in" "7 ft 6 1/8 in" "7 ft 6 1/2 in"
## [81] "7 ft 7 in" "7 ft 7 3/8 in" "7 ft 7 3/4 in" "7 ft 8 1/8 in"
## [85] "7 ft 8 1/2 in" "7 ft 8 1/2 in" "7 ft 8 7/8 in" "7 ft 9 1/4 in"
## [89] "7 ft 9 3/4 in" "7 ft 10 1/8 in" "7 ft 10 1/2 in" "7 ft 10 7/8 in"
## [93] "7 ft 11 1/4 in" "7 ft 11 5/8 in" "8 ft 0 1/8 in" "8 ft 0 1/2 in"
```