STAT547_hw06_JasmineLib

Jasmine’s STAT 547 Homework 06

Sections tackled:

Option 1 Working with Character Data - Completed the Exercises from Strings chapter in R for Data Science

- provide solutions to the exercises, to serve as an example and reference point for future work and assignments.
- create relevant “test vectors” and examples to improve understanding.

Option 2 Writing functions:

- worked through the exercise on a linear regression function customized for gapminder data ()
- as part of this assignment, made a function to perform a quadratic regression on gapminder data.
- also made functions for simple visualization of regression line data on a basic ggplot.

```
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────────────────────────────────────────
─ tidyverse 1.2.1 ──
```

```
## ✔ ggplot2 3.0.0     ✔ purrr   0.2.5
## ✔ tibble  1.4.2     ✔ dplyr   0.7.7
## ✔ tidyr   0.8.1     ✔ stringr 1.3.1
## ✔ readr   1.1.1     ✔ forcats 0.3.0
```

```
## ── Conflicts ───────────────────────────────────────────────────────────────────────── tidy
verse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
```

```
library(stringr)
library(dplyr)
library(stringi)
```

Part 1: Exercises in R for Data Science Strings Chapter:

here I work through all the exercises from the Strings Chapter. in the R for Data Science Textbook (https://r4ds.had.co.nz/strings.html).

14.2 String basics

What is difference between paste() and paste0() and what is the equivalent stringr function?

How do they differ in their handling of NA?
- using paste or paste0 , the NA will get coerced into a character, then pasted.
- using str_c, if there is an NA in either vector, it will return NA.

```
test_vector1 = c("A", "B", NA, "C")
test_vector2 = c("D", "E", "F", NA)

#paste() will concatenate vectors using sep = " "
paste(test_vector1, test_vector2)
```

```
## [1] "A D"  "B E"  "NA F" "C NA"
```

```
#paste0() will concatenate vectors without any separation
paste0(test_vector1, test_vector2)
```

```
## [1] "AD"  "BE"  "NAF" "CNA"
```

```
#str_c is the equivalent stringr function. Here we specify the separation using sep.
str_c(test_vector1, test_vector2, sep = " ")
```

```
## [1] "A D" "B E" NA    NA
```

In your own words, describe the difference between the sep and collapse arguments to str_c().

```
# using collapse will combine ALL entries within a vector, separated by ", " in this case. Using Sep will separat
e the two components being combined.
str_c(letters, LETTERS, collapse = ", " )
```

```
## [1] "aA, bB, cC, dD, eE, fF, gG, hH, iI, jJ, kK, lL, mM, nN, oO, pP, qQ, rR, sS, tT, uU, vV, wW, xX, yY, zZ"
```

```
str_c(letters, LETTERS, sep = ", " )
```

```
##  [1] "a, A" "b, B" "c, C" "d, D" "e, E" "f, F" "g, G" "h, H" "i, I" "j, J"
## [11] "k, K" "l, L" "m, M" "n, N" "o, O" "p, P" "q, Q" "r, R" "s, S" "t, T"
## [21] "u, U" "v, V" "w, W" "x, X" "y, Y" "z, Z"
```

Use str_length() and str_sub() to extract the middle character from a string.

```
#check length of a string:
str_length("teststring")
```

```
## [1] 10
```

```
#for even numbers I chose to return a substring composed of the 5th and 6th (middle characters) of the string
str_sub("teststring", 5,6)
```

```
## [1] "st"
```

```
#what does str_wrap( ) do?
#str wrap helps "wrap" strings into paragraphs, where width is how many characters can fit on one line, and inden
t/exdent determines the indentations within the paragraph. this would be useful for times when you want to displa
y paragraphs.

str_wrap("teststring, string, testing", width = 10, indent = 6, exdent = 5)
```

```
## [1] "      teststring,\n     string,\n     testing"
```

```
#what does str_trim() do?

#removes whitespace at start and end of string.
#opposite of str_trim is str_pad()
str_trim("     teststring, string, testing     ")
```

```
## [1] "teststring, string, testing"
```

Write a function that turns (e.g.) a vector c("a", "b", "c") into the string a, b, and c. Think carefully about what it should do if given a vector of length 0, 1, or 2.

```r
test_vector_3 = c("1", "2")

vector_to_string = function(x) {
  to_return = "vector input invalid"
  if (length(x) == 0) to_return
  else if (length(x) == 1) to_return = str_c(x[1])
  else if (length(x) == 2) to_return = str_c(x[1], ", ", x[2])
  to_return
}


vector_to_string(test_vector_3)
```

```
## [1] "1, 2"
```

14.3 Matching patterns with regular expressions

Explain why each of these strings don't match a : "","\",”\".
- corresponds to a used for an escape symbol also used in strings.
- \ is used for escaping a "." character in regular expressions.
- \ is not used. to escape a symbol, we will need: \\

```r
#How would you match the sequence "'\?
#I would match the sequence using:

j = "\"'\\" #need to add extra \ to the sequence due to needing to "escape" some characters.
str_view(j, "\\\"'\\\\")
```

"'\

```r
#What patterns will the regular expression \..\..\.. match? How would you represent it as a string?

#will match expressions starting with a dot then any character three times.
str_view(c(".d.e.f", "......", ".e.e.ddee"), c("\\..\\..\\.."))
```

.d.e.f

......

.e.e.ddee

14.3.2 Anchors

```r
# How would you match the literal string "$^$"?
x = "$^$"
str_view(x, "\\$\\^\\$")
```

$^$

```r
#Given the corpus of common words in stringr::words, create regular expressions that find all words that:

#Start with “y”.
str_view(words, "^y", match = TRUE)
```

year

yes

yesterday

yet

you

young

```r
#End with "x"
str_view(words, "x$", match = TRUE)
```

box

sex

six

tax

```r
#Are exactly three letters long. (Don't cheat by using str_length()!)
str_view(words, "^...$", match = TRUE)
```

act

add

age

ago

air

all

and

any

arm

art

ask

bad

bag

bar

bed

bet

big

bit

box

boy

bus

but

buy

can

car

cat

cup

cut

dad

day

die

dog

dry

due

eat

egg

end

eye

far

few

fit

fly

for

fun

gas

get

god

guy

hit

hot

how

job

key

kid

lad

law

lay

leg

let

lie

lot

low

man

may

mrs

new

non

not

now

odd

off

old

one

out

own

pay

per

put

red

rid

run

say

see

set

sex

she

sir

sit

six

son

sun

tax

tea

ten

the

tie

too

top

try

two

use

war

way

wee

who

why

win

yes

yet

you

```
#Have seven letters or more.
str_view(words, "^.......", match = TRUE)
```

absolute

account

achieve

address

advertise

afternoon

against

already

alright

although

america

another

apparent

appoint

approach

appropriate

arrange

associate

authority

available

balance

because

believe

benefit

between

brilliant

britain

brother

business

certain

chairman

character

Christmas

colleague

collect

college

comment

committee

community

company

compare

complete

compute

concern

condition

consider

consult

contact

continue

contract

control

converse

correct

council

country

current

decision

definite

department

describe

develop

difference

difficult

discuss

district

document

economy

educate

electric
encourage
english
environment
especial
evening
evidence
example
exercise
expense
experience
explain
express
finance
fortune
forward
function
further
general
germany
goodbye
history
holiday
hospital
however
hundred
husband
identify
imagine
important
improve
include
increase
individual
industry
instead
interest
introduce
involve
kitchen
language
machine
meaning
measure
mention
million
minister
morning

necessary
obvious
occasion
operate
opportunity
organize
original
otherwise
paragraph
particular
pension
percent
perfect
perhaps
photograph
picture
politic
position
positive
possible
practise
prepare
present
pressure
presume
previous
private
probable
problem
proceed
process
produce
product
programme
project
propose
protect
provide
purpose
quality
quarter
question
realise
receive
recognize
recommend
relation
remember

represent
require
research
resource
respect
responsible
saturday
science
scotland
secretary
section
separate
serious
service
similar
situate
society
special
specific
standard
station
straight
strategy
structure
student
subject
succeed
suggest
support
suppose
surprise
telephone
television
terrible
therefore
thirteen
thousand
through
thursday
together
tomorrow
tonight
traffic
transport
trouble
tuesday
understand
university

various

village

wednesday

welcome

whether

without

yesterday

14.3.3 Character classes and alternatives

```
# Create Regular expressions to find all words that:
  #start with a vowel:
str_view(words, "^[aeiou]", match = TRUE)
```

a

able

about

absolute

accept

account

achieve

across

act

active

actual

add

address

admit

advertise

affect

afford

after

afternoon

again

against

age

agent

ago

agree

air

all

allow

almost

along

already

alright

also

although

always

america
amount
and
another
answer
any
apart
apparent
appear
apply
appoint
approach
appropriate
area
argue
arm
around
arrange
art
as
ask
associate
assume
at
attend
authority
available
aware
away
awful
each
early
east
easy
eat
economy
educate
effect
egg
eight
either
elect
electric
eleven
else
employ
encourage
end

engine
english
enjoy
enough
enter
environment
equal
especial
europe
even
evening
ever
every
evidence
exact
example
except
excuse
exercise
exist
expect
expense
experience
explain
express
extra
eye
idea
identify
if
imagine
important
improve
in
include
income
increase
indeed
individual
industry
inform
inside
instead
insure
interest
into
introduce
invest

involve

issue

it

item

obvious

occasion

odd

of

off

offer

office

often

okay

old

on

once

one

only

open

operate

opportunity

oppose

or

order

organize

original

other

otherwise

ought

out

over

own

under

understand

union

unit

unite

university

unless

until

up

upon

use

usual

```
# that contain only consonants
str_view(words, "^[^aeiou]*$", match = TRUE)
```

by

dry

fly

mrs

try

why

```
  # that end with ed but not eed
str_view(words, "[^e]ed$", match = TRUE)
```

bed

hundred

red

```
  #that end with ing or ise
str_view(words, "i(se|ng)$", match = TRUE)
```

advertise

bring

during

evening

exercise

king

meaning

morning

otherwise

practise

raise

realise

ring

rise

sing

surprise

thing

```
#Empirically verify the rule “i before e except after c”.
str_view(words, "cie", match = TRUE)
```

science

society

```
str_view(words, "[^c]ei", match = TRUE)
```

weigh

```
#rule not always true.


#Is “q” always followed by a “u”?
str_view(words, "q[^u]", match = TRUE)
```

```
#in this dataset, yes.


#Write a regular expression that matches a word if it’s probably written in British English, not American Englis
h.
str_view(words, "our", match = TRUE)
```

colour

course

court

encourage

favour

four

hour

labour

resource

```
#Create a regular expression that will match telephone numbers as commonly written in your country.
phone_number_list = c("604-111-2345","514-456-7765", "12344556-232-22", "abcd", "1-604-928-3481", "223-33333" )

str_view(phone_number_list, "^(\\d\\d\\d-\\d\\d\\d-\\d\\d\\d\\d|^\\d-\\d\\d\\d-\\d\\d\\d-\\d\\d\\d\\d)$")
```

604-111-2345

514-456-7765

12344556-232-22

abcd

1-604-928-3481

223-33333

14.3.4 Repetition

```
#Describe the equivalents of ?, +, * in {m,n} form.
#? = 0 or 1 = {0,1}
#+ = 1 or more = {1,}
#* = 0 or more = {0,}


#make a vector to test repeats:
test_repeats = c("abcde", "aabbccde", "cccccdddddd")

str_view(test_repeats, "a{0,1}")
```

abcde

aabbccde

cccccdddddd

```
str_view(test_repeats, "b{1,}")
```

abcde

aabbccde

cccccdddddd

```
str_view(test_repeats, "a{0,}")
```

abcde

aabbccde

ccccccdddddd

```
#Describe in words what these regular expressions match:
# ^.*$ will match to any string.
# "\\{.+\\}" will match to 1 or more characters surrounded by curly braces in a string. ex: {a} or {abcde}
# \d{4}-\d{2}-\d{2} will match to any numbers that fit the following format: 1111-11-11
#"\\\\{4}" will match to any string containing four backslashes.
# for example: \\\\abcd (written as a string: "\\\\\\\\abcd")

str_view("\\\\\\\\abcd", "\\\\{4}")
```

\\\\abcd

```
#Create regular expressions to find all words that:
#start with three consonants:
str_view(words, "^[^aeiou]{3}", match = TRUE)
```

Christ

Christmas

dry

fly

mrs

scheme

school

straight

strategy

street

strike

strong

structure

system

three

through

throw

try

type

why

```
#have three or more vowels in a row:
str_view(words, "[aeiou]{3,}", match = TRUE)
```

beauty

obvious

previous

quiet

serious

various

```
#Have two or more vowel-consonant pairs in a row.
str_view(words, "([aeiou][^aeiou]){2,}", match = TRUE)
```

absolute

agent

along
america
another
apart
apparent
authority
available
aware
away
balance
basis
become
before
begin
behind
benefit
business
character
closes
community
consider
cover
debate
decide
decision
definite
department
depend
design
develop
difference
difficult
direct
divide
document
during
economy
educate
elect
electric
eleven
encourage
environment
europe
even
evening
ever
every

evidence
exact
example
exercise
exist
family
figure
final
finance
finish
friday
future
general
govern
holiday
honest
hospital
however
identify
imagine
individual
interest
introduce
item
jesus
level
likely
limit
local
major
manage
meaning
measure
minister
minus
minute
moment
money
music
nature
necessary
never
notice
okay
open
operate
opportunity
organize

original
over
paper
paragraph
parent
particular
photograph
police
policy
politic
position
positive
power
prepare
present
presume
private
probable
process
produce
product
project
proper
propose
protect
provide
quality
realise
reason
recent
recognize
recommend
record
reduce
refer
regard
relation
remember
report
represent
result
return
saturday
second
secretary
secure
separate
seven

similar

specific

strategy

student

stupid

telephone

television

therefore

thousand

today

together

tomorrow

tonight

total

toward

travel

unit

unite

university

upon

visit

water

woman

```
#Solve the beginner regexp crosswords at https://regexcrossword.com/challenges/beginner.
#was not able to recreate the crossword here
#but I was able to solve 2 of the beginner crosswords that I attempted.
```

14.3.5 Grouping and backreferences

```
#describe in words what these expressions will match:
#(.)\1\1 will match any character that repeats 3 times

#"(.)(.)\\2\\1" will match any characters that fit the following pattern: "lool" or "saas"

#(..)\1 will match any characters that repeat twice such as "haha" or "hehe"

#"(.).\\1.\\1" will match any characters where the first character is repeated three times,
#starting once at the start of the pattern, then two more times but with any character in between the two other t
imes.
#for example "nanin" "tgtkt"

#"(.)(.)(.).*\\3\\2\\1" will match strings with the pattern: "abckcba" or "123i321" or "123ijklmnop321"
a = "123iiddssfe32144444"
str_view(a, "(.)(.)(.).*\\3\\2\\1")
```

123iiddssfe32144444

```
#Construct regular expressions to match words that:
#Start and end with the same character.

str_view(c("bob","snacks","test"),"(.).*\\1")
```

bob

snacks

test

```
#Contain a repeated pair of letters (e.g. “church” contains “ch” repeated twice.)
str_view(c("church","gg123gg"), "(..).*\\1")
```

church

gg123gg

```
#Contain one letter repeated in at least three places (e.g. “eleven” contains three “e”s.)
str_view(c("eleven", "notamatch", "caravans", "carraavvans"), "(.).*\\1.*\\1.*")
```

eleven

notamatch

caravans

carraavvans

14.4.1 Detect matches

```
library(stringi)
library(stringr)
#For each of the following challenges, try solving it by using both a single regular expression,
#and a combination of multiple str_detect() calls.

#words starting or ending in x
words_in_x = c("six", "seven", "xylophone", "exit")
endx = str_detect(words_in_x,"x$")
startx = str_detect(words_in_x,"^x")
words_in_x[endx|startx]
```

```
## [1] "six"       "xylophone"
```

```
#words starting with vowel and ending in consonant
vowel_consonant = c("notamatch","elect", "too", "all", "alice")
start_vowel = str_detect(vowel_consonant, "^[aeiou]")
end_consonant = str_detect(vowel_consonant, "[^aeiou]$")
vowel_consonant[start_vowel & end_consonant]
```

```
## [1] "elect" "all"
```

```
#Are there any words that contain at least one of each different vowel?
#no there are not, as no words are returned from the call below:
contain_a = str_detect(words, "a")
contain_e = str_detect(words, "e")
contain_i = str_detect(words,"i")
contain_o = str_detect(words,"o")
contain_u = str_detect(words,"u")
words[contain_a & contain_e & contain_i & contain_o & contain_u]
```

```
## character(0)
```

```
#What word has the highest number of vowels?
number_vowels = str_count(words, "[aeiou]")
words[which(number_vowels == max(number_vowels))] %>%
  head()
```

```
## [1] "appropriate" "associate"   "available"   "colleague"   "encourage"
## [6] "experience"
```

```
#What word has the highest proportion of vowels? (Hint: what is the denominator?)

#highest proportion:
proportion_vowels = str_count(words, "[aeiou]")/str_length(words)
words[which(proportion_vowels == max(proportion_vowels))] %>%
  head()
```

```
## [1] "a"
```

14.4.3 Extract matches

```
#In the previous example, you might have noticed that the regular expression matched “flickered”, which is not a
 colour. Modify the regex to fix the problem.
#adding [^a-z] will remove any a-z characters before the word red.
#using \\b \\b to surround the word of interest will avoid matching with words that contain it.

colours <- c("\\bred\\b", "orange", "yellow", "green", "blue", "purple")
colour_match <- str_c(colours, collapse = "|")
more <- sentences[str_count(sentences, colour_match) > 1]
str_view_all(more, colour_match)
```

`It is hard to erase blue or red ink.`

`The sky in the west is tinged with orange red.`

```
#extract the first word from each sentence:
sentences %>%
  str_extract("[A-Za-z]+") %>%
  head()
```

```
## [1] "The"   "Glue"  "It"    "These" "Rice"  "The"
```

```
#extract all words ending in ing:

unlist(str_extract_all(sentences, "[a-zA-Z]+ing")) %>%
  head()
```

```
## [1] "stocking" "spring"   "evening"  "morning"  "winding"  "living"
```

```
#plurals
#for this exercise I will look at words ending in "s"

unlist(str_extract_all(sentences, "[a-zA-Z]+s")) %>%
  head()
```

```
## [1] "planks" "eas"    "Thes"   "days"   "is"     "dis"
```

14.4.4 Grouped matches

```
#Find all words that come after a “number” like “one”, “two”, “three” etc.
#Pull out both the number and the word.
tibble(sentence = sentences) %>%
 str_extract_all("(one|two|three|four|five|six|seven|eight|nine|ten) ([^ ]+)"
  )
```

```
## Warning in stri_extract_all_regex(string, pattern, simplify = simplify, :
## argument is not an atomic vector; coercing
```

```
## [[1]]
##  [1] "ten served"      "one over"        "seven books"
##  [4] "two met"         "two factors"     "one and"
##  [7] "three lists"     "seven is"        "two when"
## [10] "one floor.\","   "ten inches.\","  "one with"
## [13] "one war"         "one button"      "six minutes.\","
## [16] "ten years"       "one in"          "ten chased"
## [19] "one like"        "two shares"      "two distinct"
## [22] "one costs"       "five cents"      "ten two"
## [25] "five robins.\"," "four kinds"      "one rang"
## [28] "ten him.\","     "three story"     "ten by"
## [31] "one wall.\","    "three inches"    "ten your"
## [34] "six comes"       "ten than"        "one before"
## [37] "three batches"   "two leaves.\","
```

```
#Find all contractions.

tibble(sentence = sentences) %>%
  str_extract_all("[^ ]+\\'[^ .]")
```

```
## Warning in stri_extract_all_regex(string, pattern, simplify = simplify, :
## argument is not an atomic vector; coercing
```

```
## [[1]]
##  [1] "\"It's"     "man's"      "don't"      "store's"    "workmen's"
##  [6] "\"Let's"    "sun's"      "child's"    "king's"     "\n\"It's"
## [11] "don't"      "queen's"    "don't"      "pirate's"   "neighbor's"
```

14.4.5 Replacing matches

```
#Replace all forward slashes in a string with backslashes.

slashes = c("ab/cd", "/efgh", "/////", "defghi//")
backslashes = str_replace_all(slashes,"\\/", "\\\\")
#returns the raw contents of the string:
writeLines(backslashes)
```

```
## ab\cd
## \efgh
## \\\\\
## defghi\\
```

```
#Implement a simple version of str_to_lower() using replace_all().
str_replace_all(sentences, c("A" = "a", "B" = "b", "C" = "c", "D" = "d", "E" = "e", "F" = "f", "G" = "g", "H" =
"h", "I" = "i", "J" = "j", "K" = "k", "L" = "l", "M" = "m", "N" = "n", "O" = "o", "P"="p", "Q"="q", "R" = "r",
"S" = "s", "T" = "t", "U" = "u", "V" = "v", "W" = "w", "X" ="x", "Y" = "y", "Z" ="z" )) %>%
  head()
```

```
## [1] "the birch canoe slid on the smooth planks."
## [2] "glue the sheet to the dark blue background."
## [3] "it's easy to tell the depth of a well."
## [4] "these days a chicken leg is a rare dish."
## [5] "rice is often served in round bowls."
## [6] "the juice of lemons makes fine punch."
```

```
#Switch the first and last letters in words. Which of those strings are still words?
words [words %in% str_replace(words, "^([a-zA-Z])(.*)([a-z])$", "\\3\\2\\1")] %>%
  head()
```

```
## [1] "a"       "america" "area"    "dad"     "dead"    "deal"
```

14.4.6 Splitting

```
#Split up a string like "apples, pears, and bananas" into individual components.

str_split("apples, pears, and bananas", ", and |, ")
```

```
## [[1]]
## [1] "apples"  "pears"   "bananas"
```

```
#Why is it better to split up by boundary("word") than " "?
#using boundary will handle spaces and other characters not part of a word (linke punctuation or numbers)

#What does splitting with an empty string ("") do? Experiment, and then read the documentation.
str_split("apples, pears, and bananas", "")
```

```
## [[1]]
##  [1] "a" "p" "p" "l" "e" "s" "," " " "p" "e" "a" "r" "s" "," " " "a" "n"
## [18] "d" " " "b" "a" "n" "a" "n" "a" "s"
```

```
#splitting by "" will split the string into individual characters,
#including spaces (whitespace), and punctuation.
```

14.5 Other types of pattern

```
#How would you find all strings containing \ with regex() vs. with fixed()?
#using regex():
test_vector_4 = c("\\\\\\", "abcde", "\\abc")
writeLines(test_vector_4)
```

```
## \\\
## abcde
## \abc
```

```
str_view_all(test_vector_4, regex("\\\\"))
```

\\\

abcde

\abc

```
#using fixed():
str_view_all(test_vector_4, fixed("\\"))
```

\\\

abcde

\abc

```
#What are the five most common words in sentences?
#obtain a vector of all words in the sentences list by using str_extract_all and use unlist to convert the list t
o a vector:
words_in_sentences = unlist(str_extract_all(str_to_lower(sentences), boundary("word")))

topwords = words_in_sentences %>%
  tibble() %>%
  set_names("commonwords") %>%
  group_by(commonwords) %>%
  count()



topwords
```

```
## # A tibble: 1,904 x 2
## # Groups:   commonwords [1,904]
##    commonwords     n
##    <chr>       <int>
##  1 a             202
##  2 about           1
##  3 abrupt          1
##  4 absent          1
##  5 account         1
##  6 acid            1
##  7 across          3
##  8 act             4
##  9 actor           1
## 10 actress         1
## # ... with 1,894 more rows
```

```
#I was not able to get past this point.
#could not determine how to adequately sort the data to obtain top most used words.
#visually, by sorting through this counted data, I was able to determine that some the most common words include:
# "a", "and", "in", "is"
```

14.7 stringi

```
#Find the stringi functions that:
#https://www.rdocumentation.org/packages/stringi/versions/1.2.4
#Count the number of words. = stri_count_boundaries
#Find duplicated strings. = stri_duplicated
#Generate random text. = stri_rand_strings

#How do you control the language that stri_sort() uses for sorting?
#use locale = " " to control the language based on the as a ISO 639 language code https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes
```

Part 2: Write one (or more) functions that do something useful to pieces of the Gapminder data.

```
library(gapminder)
suppressPackageStartupMessages(library(ggplot2))
suppressPackageStartupMessages(library(dplyr))
```

The first part of this exercise is based on the tutorial provided on linear regression functions (http://stat545.com/block012_function-regress-lifeexp-on-year.html)
The first part is heavily based on the tutorial, but is necessary to provide context and understanding for the functions that I make next. I specify below the point where I depart from the tutorial and start with my own work.

The second part of this exercise will branch off from this starting point.

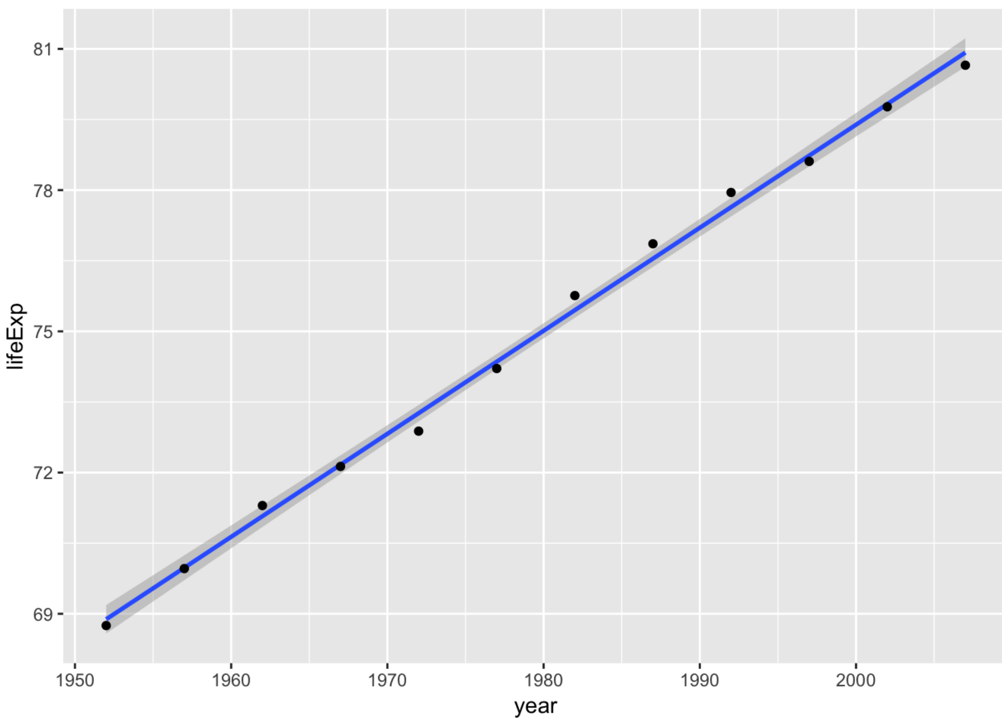
First step, pick one country to work with as a starting point:

```
country_ = "Canada"
country_data_Canada = gapminder %>%
  filter (country =="Canada")
country_data_Canada
```

```
## # A tibble: 12 x 6
##    country continent  year lifeExp      pop gdpPercap
##    <fct>   <fct>     <int>   <dbl>    <int>     <dbl>
##  1 Canada  Americas   1952    68.8 14785584    11367.
##  2 Canada  Americas   1957    70.0 17010154    12490.
##  3 Canada  Americas   1962    71.3 18985849    13462.
##  4 Canada  Americas   1967    72.1 20819767    16077.
##  5 Canada  Americas   1972    72.9 22284500    18971.
##  6 Canada  Americas   1977    74.2 23796400    22091.
##  7 Canada  Americas   1982    75.8 25201900    22899.
##  8 Canada  Americas   1987    76.9 26549700    26627.
##  9 Canada  Americas   1992    78.0 28523502    26343.
## 10 Canada  Americas   1997    78.6 30305843    28955.
## 11 Canada  Americas   2002    79.8 31902268    33329.
## 12 Canada  Americas   2007    80.7 33390141    36319.
```

Next, make a rough plot to get an idea of the data:

```
country_data_Canada %>%
  ggplot() +aes(x = year, y = lifeExp) +
  geom_smooth(method = "lm") +
  geom_point()
```



Visually, it appears that the linear regression line does fit the data, in this case. We should still check the fit of the regression:

```
fit_lm_regression_canada = lm(lifeExp ~ year, country_data_Canada)
#look at the intercepts:
coefficients(fit_lm_regression_canada)
```

```
##  (Intercept)         year
## -358.3488923    0.2188692
```

From this intercept, we know that something is off in our fit, because the intercept should no go all the way down to -358 years. Here, we can reset the parameters for the plot so that we make the intercept match up to 1952, our earliest datapoint:

```
fit_lm_regression_canada = lm(lifeExp ~I(year-1952), country_data_Canada)
coefficients(fit_lm_regression_canada)
```

```
##    (Intercept) I(year - 1952)
##     68.8838462      0.2188692
```

Now we can take this and apply it to a function:

```
linear_regression_fit = function(gap_data, offset = 1952) {
  linear_fit = lm(lifeExp~I(year-offset), gap_data)
  setNames(coef(linear_fit), c("intercept", "slope"))
}

linear_regression_fit(country_data_Canada)
```

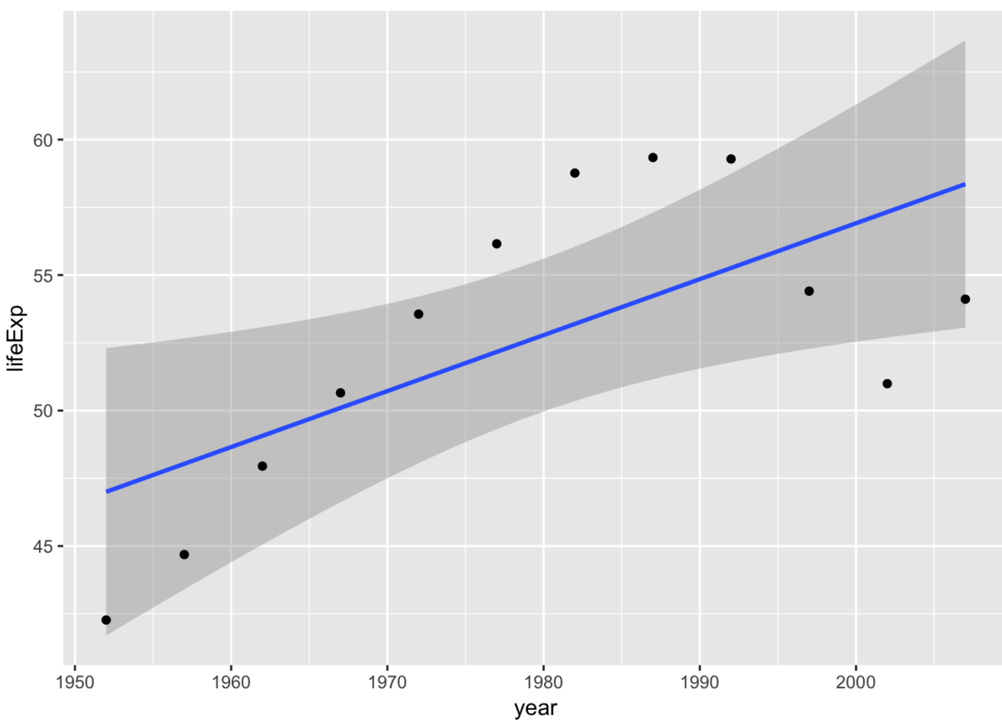
```
##  intercept      slope
## 68.8838462  0.2188692
```

Testing this model on another country:

```
country_data_Kenya = gapminder %>%
  filter(country =="Kenya")

linear_regression_fit(country_data_Kenya)
```

```
##  intercept      slope
## 47.0020385  0.2065077
```

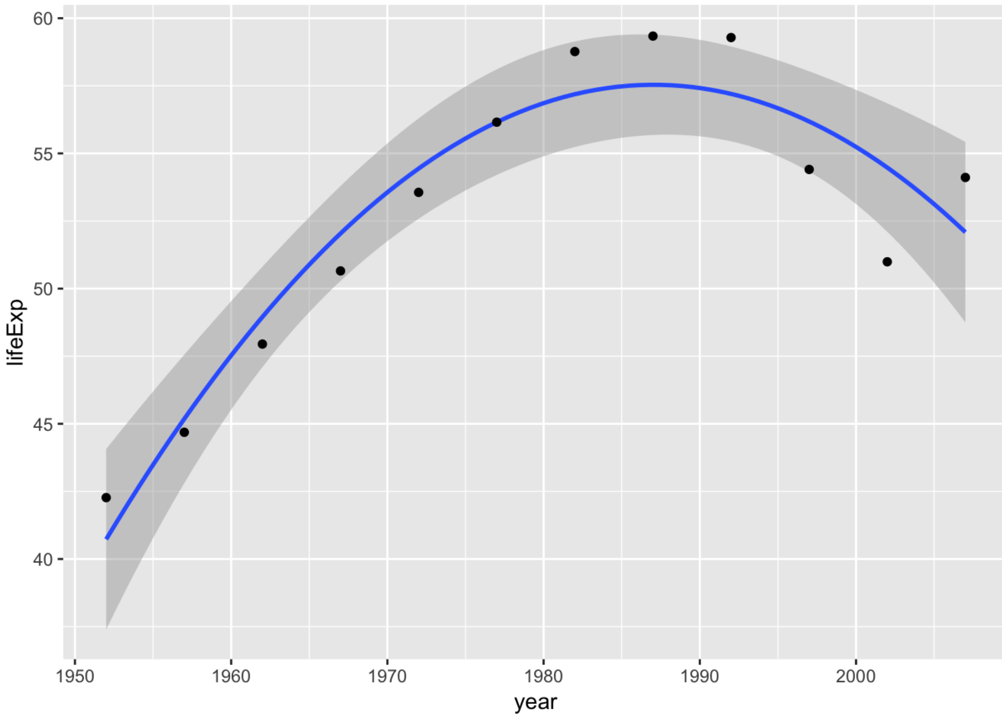
```
country_data_Kenya %>%
  ggplot() + aes(x = year, y = lifeExp) +
  geom_smooth(method = "lm") +
  geom_point()
```



Here is where I start original work

We can see here that while the function worked, it appears to fit a quadratic model, rather than linear. This is how we can generalize the lm method to fit a quadratic model:

```
#first, make a ggplot of what we would be modeling in our function to visualize:
country_data_Kenya %>%
  ggplot() + aes(x = year, y = lifeExp) +
  geom_smooth(method ="lm", formula = y~x+I(x^2)) +
  geom_point()
```




```
#write a function for quadratic equation fitting to the data
quadratic_regression_fit = function(gap_data, offset = 1952) {
  #add a square function here:
  quadratic_fit = lm(lifeExp~I(year-offset)+I((year-offset)^2), gap_data)
  setNames(coefficients(quadratic_fit), c("intercept", "coefficient x", "coefficient x^2"))

}

quadratic_regression_fit(country_data_Kenya)
```

```
##       intercept   coefficient x coefficient x^2
##     40.72898901      0.95927363     -0.01368665
```

This is great, but it would be easier if we didn't have to create a dataset each time we want to call our function. This can be modified by having our function create it's own filtered dataset by filtering by country name.

```
#update linear regression function:
linear_regression_fit = function(countryname = "", offset = 1952) {
  gap_data = gapminder %>%
    filter(country == countryname)
  linear_fit = lm(lifeExp~I(year-offset), gap_data)
  setNames(coef(linear_fit), c("intercept", "slope"))
}

#test to see if we get the same coefficients for Canada as before:
linear_regression_fit("Canada")
```

```
##  intercept      slope
## 68.8838462  0.2188692
```

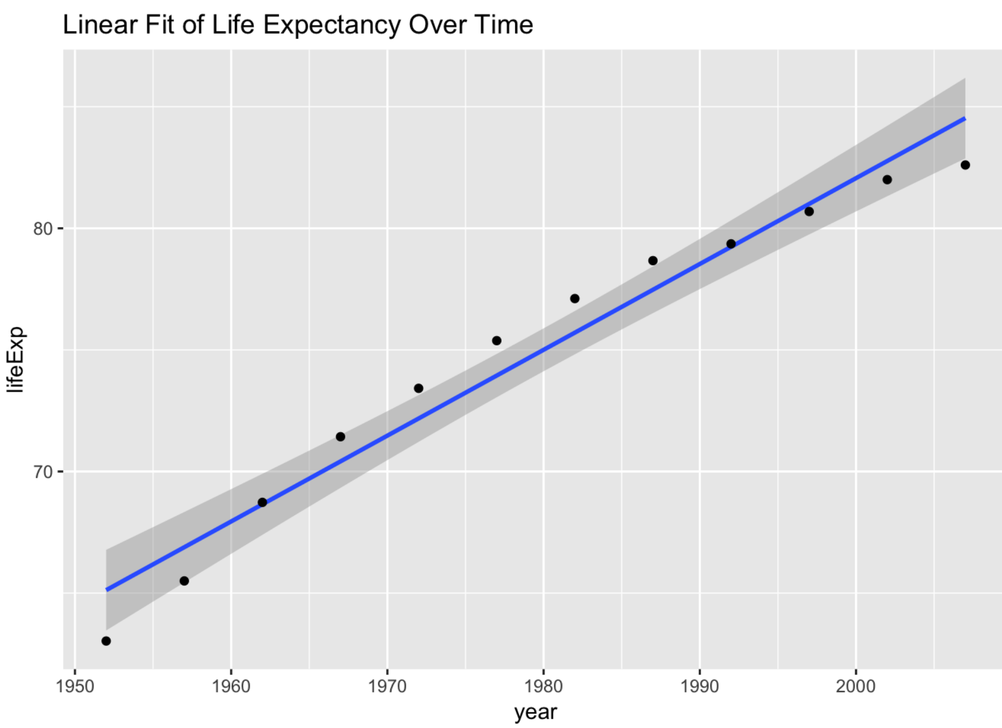
```
#update quadratic regression function:
quadratic_regression_fit = function(countryname = "", offset = 1952) {
  gap_data = gapminder %>%
  filter(country == countryname)
  quadratic_fit = lm(lifeExp~I(year-offset)+I((year-offset)^2), gap_data)
  setNames(coefficients(quadratic_fit), c("intercept", "coefficient x", "coefficient x^2"))
}

#test to see if we get the same coefficients for Kenya as before:
quadratic_regression_fit("Kenya")
```

```
##       intercept   coefficient x coefficient x^2
##     40.72898901      0.95927363     -0.01368665
```

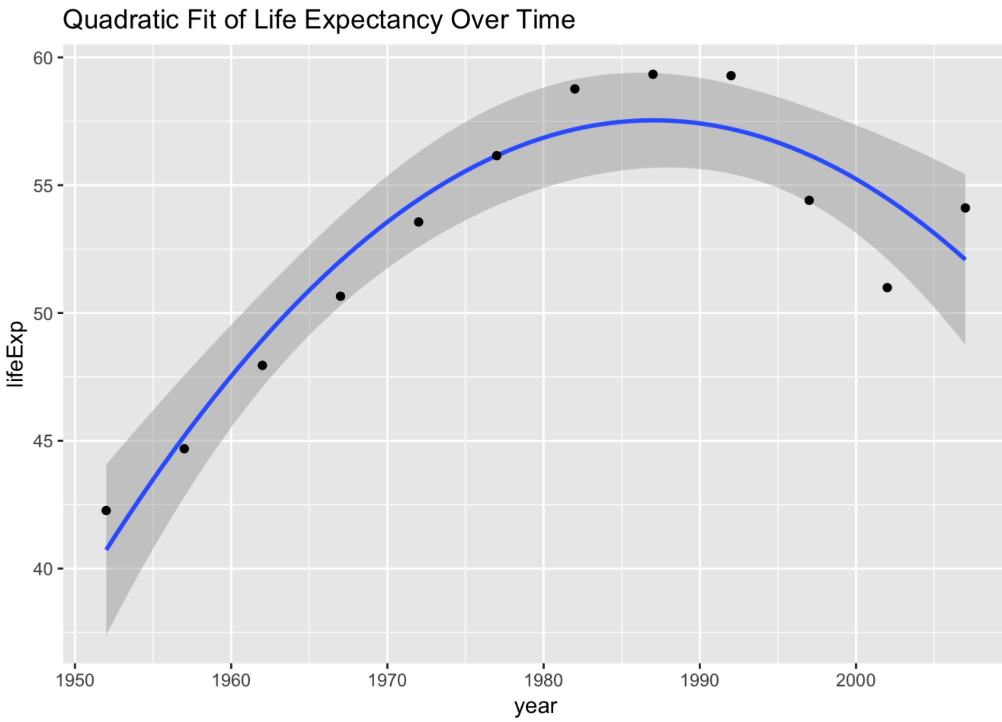
Conclude: Yes, the coefficients obtained for Canada's linear fit and Kenya's quadratic are the same as previously obtained.

For the sake of this assignment, I also found it useful to have a way to directly plot either a linear or quadratic fit into ggplot data:

```
linearfit_ggplot = function (countryname = "") {
  gapminder %>%
  filter (country ==countryname) %>%
  ggplot() + aes(x = year, y = lifeExp) +
  geom_smooth(method ="lm") +
  geom_point() +
  ggtitle("Linear Fit of Life Expectancy Over Time")

}

linearfit_ggplot("Japan")
```




```
quadraticfit_ggplot = function (countryname = "") {
  gapminder %>%
    filter(country == countryname) %>%
    ggplot() + aes(x = year, y = lifeExp) +
    geom_smooth(method = "lm", formula = y~x+I(x^2) )+
    geom_point() +
    ggtitle("Quadratic Fit of Life Expectancy Over Time")

}

quadraticfit_ggplot("Kenya")
```



Conclusion: Here we have simplified the process of making a quick plot to check our functions, by writing another function for linear and quadratic fits in ggplot.

The next step is to test the function: Try to break the function

```
#the function should not work if I enter a country that is not in the gapminder dataset.
#linearfit_ggplot("abc")
#quadratic_regression_fit("abc")
#quadraticfit_ggplot("abc")
#linear_regression_fit("abc")

#the function should not work if I enter a non-string argument:
#linearfit_ggplot(1)
#quadratic_regression_fit(2)
#quadraticfit_ggplot(3)
#linear_regression_fit(4)

quadratic_regression_fit = function(countryname = "", offset = 1952) {

  if (!is.character(countryname)) {
        stop(paste("expecting input for countryname to be a string corresponding to a country in ggplot. You gave
 me ",
                   typeof(countryname)))
    }
  gap_data = gapminder %>%
  filter(country == countryname)
  quadratic_fit = lm(lifeExp~I(year-offset)+I((year-offset)^2), gap_data)
  setNames(coefficients(quadratic_fit), c("intercept", "coefficient x", "coefficient x^2"))
}

#now if we enter a non-character input, we will return an error:
#returns you gave me list:
#quadratic_regression_fit(gapminder)

#returns you gave me double:
#quadratic_regression_fit(2)

#this should work:
quadratic_regression_fit("Kenya")
```

```
##       intercept   coefficient x coefficient x^2
##     40.72898901      0.95927363     -0.01368665
```

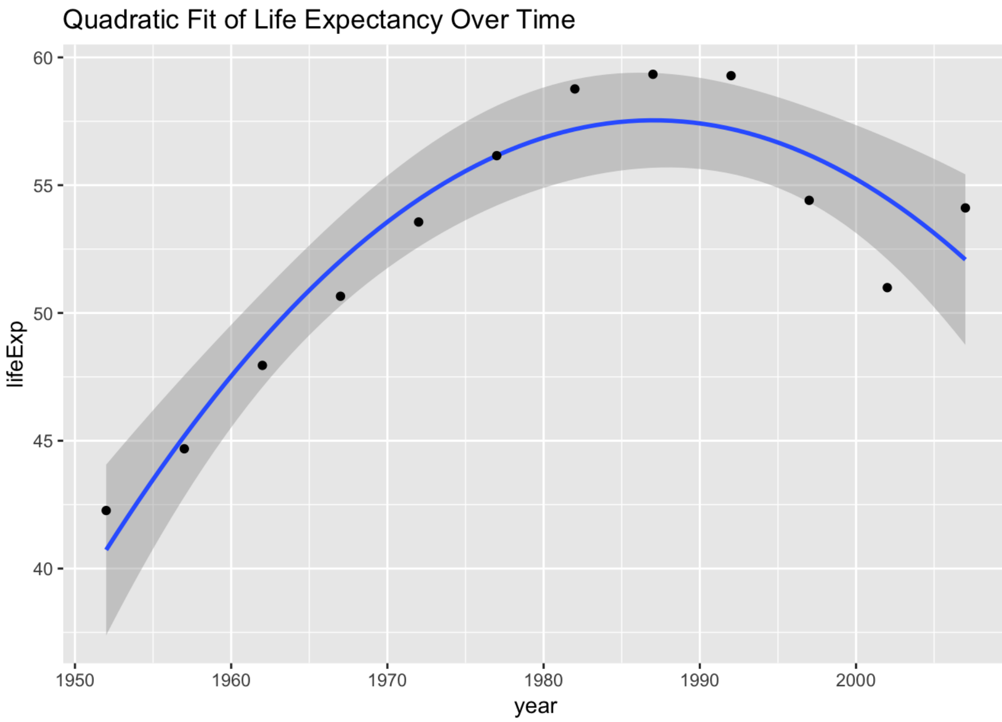
```
#applying the same stop if not for the linear regression function:
linear_regression_fit = function(countryname = "", offset = 1952) {
  if (!is.character(countryname)) {
        stop(paste("expecting input for countryname to be a string corresponding to a country in ggplot. You gave
 me ",
                   typeof(countryname)))
    }
  gap_data = gapminder %>%
    filter(country == countryname)
  linear_fit = lm(lifeExp~I(year-offset), gap_data)
  setNames(coef(linear_fit), c("intercept", "slope"))
}

#returns you gave me double:
#linear_regression_fit(2)
```

Now that we have these two nice functions to make linear and quadratic fits for the data, the next logical step is to be able to estimate life expectancy based on this data.

```
quadratic_regression_fit_estimate = function(countryname = "", year_estimate = 1999, offset = 1952) {
  #we want the year_estimate to be an integer between 1952 and 2007 (useful for estimating years not provided in
 gapminder dataset )
   if (!is.double(year_estimate)) {
        stop(paste("expecting input for year_estimate to be an integer between 1952 and 2007. You gave me ",
                   typeof(year_estimate)))
    }
  if (!is.character(countryname)) {
        stop(paste("expecting input for countryname to be a string corresponding to a country in ggplot. You gave
 me ",
                   typeof(countryname)))
  }
   if (year_estimate > 2007 | year_estimate <1952) {
        stop("expecting input for year_estimate to be an integer between 1952 and 2007.")
    }
  gap_data = gapminder %>%
  filter(country == countryname)
  quadratic_fit = lm(lifeExp~I(year-offset)+I((year-offset)^2), gap_data)
  setNames(coefficients(quadratic_fit), c("intercept", "coefficient x", "coefficient x^2"))
  x2_coeff = coefficients(quadratic_fit)[3]
  x_coeff = coefficients(quadratic_fit)[2]
  intercept_value = coefficients(quadratic_fit)[1]
  offsetyear = year_estimate-offset
 # x2_coeff
  #x_coeff
  #intercept_value
 life_exp_estimate = intercept_value + x2_coeff*offsetyear^2 + x_coeff*offsetyear
 life_exp_estimate
}
#this returns a value of 55.24 years.
quadratic_regression_fit_estimate("Kenya", 2000)
```

```
## (Intercept)
##    55.24007
```

```
#checking with our ggplot quadratic fit, does this estimate seem reasonable?
quadraticfit_ggplot("Kenya")
```



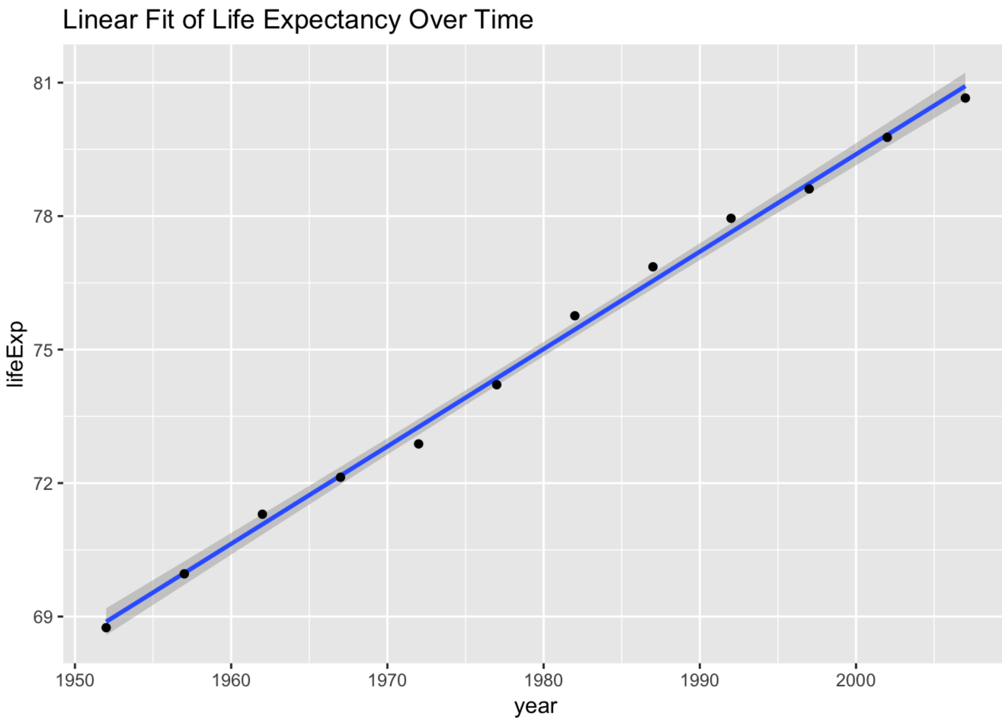
Conclude: Yes, from the output returned and the quadratic fit plot, the estimate of 55 years does seem reasonable.

Now we can repeat the same process to estimate life expectancy for data fitted to a linear model:

```
linear_regression_fit_estimate = function(countryname = "", year_estimate = 1999, offset = 1952) {
  #we want the year_estimate to be an integer between 1952 and 2007 (useful for estimating years not provided in
 gapminder dataset )
   if (!is.double(year_estimate)) {
        stop(paste("expecting input for year_estimate to be an integer between 1952 and 2007. You gave me ",
                   typeof(year_estimate)))
    }
  if (!is.character(countryname)) {
        stop(paste("expecting input for countryname to be a string corresponding to a country in ggplot. You gave
 me ",
                   typeof(countryname)))
  }
   if (year_estimate > 2007 | year_estimate <1952) {
        stop("expecting input for year_estimate to be an integer between 1952 and 2007.")
   }

  gap_data = gapminder %>%
    filter(country == countryname)
  linear_fit = lm(lifeExp~I(year-offset), gap_data)
  setNames(coef(linear_fit), c("intercept", "slope"))
  x_coeff = coefficients(linear_fit)[2]
  intercept_value = coefficients(linear_fit)[1]
  offsetyear = year_estimate-offset
  #x_coeff
  #intercept_value
 life_exp_estimate = intercept_value + x_coeff*offsetyear
 life_exp_estimate
}

linear_regression_fit_estimate("Canada", 2000)
```

```
## (Intercept)
##    79.38957
```

```
linearfit_ggplot("Canada")
```



Conclude:
Sanity Check: the estimated life expectancy for this year matches with what we can see on the plot of the linear regression.