Assignment5

Part 1

Development:

1. Requirements:
   a. Complete the following tutorial: **Introduction_to_R_Setup_and_Tutorial (includes A5 requirements): Save as: learn_to_use_r.R**
   b. Code and run **lis4369_a5.R** (see below). Include link to file in **a5 README.md** file.
   c. Include **at least two** *4-panel* **RStudio** screenshots : 1) **learn_to_use_r.R**, and 2) **lis4369_a5.R** code.
   d. Also, be sure to include **at least four plots**—that is, at least **two** plots for the **tutorial**, and **two** plots for the **assignment file** (below), in your **README.md file**.

2. Be sure to test your program using **RStudio**.

Part 2

README.md file should include the following items:

1. **Assignment requirements, as per A1.**
2. Screenshots of output from code below, ***and* from tutorial**.

Deliverables:

1. Provide **Bitbucket** read-only access to **lis4369** repo, include links to the repos you created in the above tutorials in **README.md**, using Markdown syntax (**README.md** must also include screenshots as per above.)
2. **FSU's Learning Management System**: lis4369 **Bitbucket** repo

R and Python Comparisons

Source: Python for R Users: A Data Science Approach

	R	Python (using pandas package*)
Getting the names of rows and columns of data frame "df"	rownames(df)	df.index
	returns the name of the rows	*returns the name of the rows*
	colnames(df)	df.columns
	returns the name of the columns	*returns the name of the columns*
Seeing the top and bottom "x" rows of the data frame "df"	head(df,x)	df.head(x)
	returns top x rows of data frame	*returns top x rows of data frame*
	tail(df,x)	df.tail(x)
	returns bottom x rows of data frame	*returns bottom x rows of data frame*
Getting dimensions of data frame "df"	dim(df)	df.shape
	returns in this format: rows, columns	*returns in this format: (rows, columns)*
Length of data frame "df"	length(df)	len(df)
	returns no. of columns in data frames	*returns no. of columns in data frames*

Note: combine demo1.R and demo2.R into lis4369_a5.R

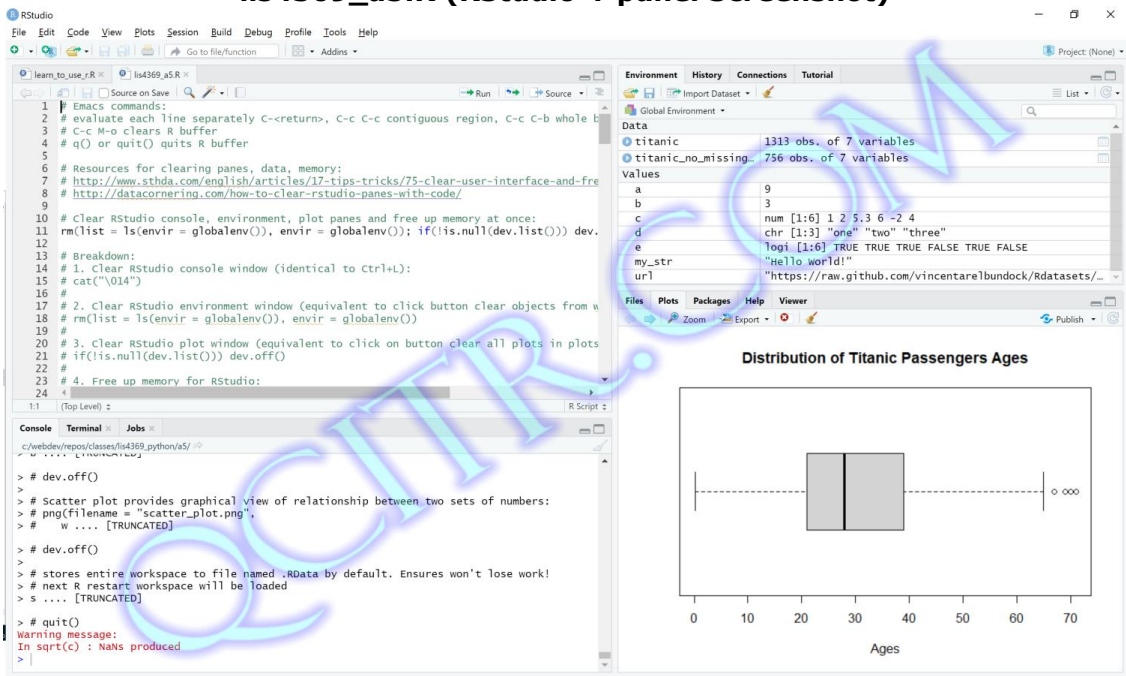
demo1.R

```
# Data Types:
# R has a wide variety of data types including:
# scalars, vectors (numerical, character, logical), matrices, data frames, and lists.

# Assignment:
# Scalars: The most basic way to store a number is through assignment.
# Assignment is specified with "<-" characters.
# Assigns number on right of assignment symbol, and stores it in variable on left.
a <- 9
a # print value of a

a + 5 # print a + 5

b <- sqrt(a) # assign square root of a to b
b # print value of b

# Nonscalar data types:
# Easiest way to store list of numbers, through assignment, using c command.
# Note: c means "combine"
# Vectors (one-dimensional arrays), by default, are specified with the c command.

c <- c(1,2,5.3,6,-2,4) # numeric vector
# or...
# c <- vector(1,2,5.3,6,-2,4) # also, numeric vector
print(c)

typeof(c) # print data type

is.list(c)   #FALSE
is.vector(c)  #TRUE

d <- c("one","two","three") # character vector
d

typeof(d) # print data type

e <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE) #logical vector
e

typeof(e) # print data type

# To refer to row in Python, use index.
# In R, refer to object in ith row and jth column by OBJECTNAME[i,j].
# In R, refer to column name by OBJECTNAME$ColumnName.
# ***Note***: Python, index starts with 0. R starts with 1!
d[1]

# String specified by using quotes--either single or double quotes work:
my_str <- "Hello World!"
my_str

typeof(my_str) # print data type

# Some intrinsic functions on scalar a, and vector c:
sqrt(a)

sqrt(c) # Research why this produces an error!

a^2 # scalar squared

c^2 # vector squared

min(c)

max(c)

mean(c)

sum(c)

# Reading CSV file (comma separated values):
# Command: read.csv() reads file into data frame (similar to Python)
# Requires at least one argument: name of file.
# If three arguments:
  # 1) name of file
  # 2) indicates if first row are labels/headers
  # 3) indicates separator character

url = "https://raw.github.com/vincentarelbundock/Rdatasets/master/csv/Stat2Data/Titanic.csv"
titanic <- read.csv(file=url,head=TRUE,sep=",") # reads file, and assigns to variable
# To get more information, use help command: help(read.csv)

titanic # displays all data from file

summary(titanic) # summary(): generic function used to produce result summaries
# Note: Python uses NaN to denote missing values, while R uses NA.

dir() # list files in current working directory

getwd() # determine current working directory

names(titanic) # print headers (Note: "X" column is used for numbering rows)

# Variable "titanic" contains the 7 columns of data.
# Each column assigned name based on header row (first line in file).
# Access each column using "$":
titanic$Name # prints names

titanic$Age # prints ages

attributes(titanic) # returns object's attribute list
```

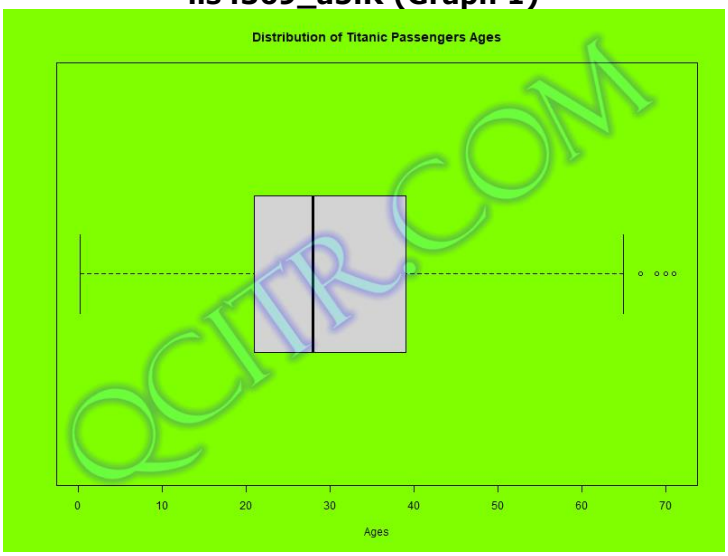
demo2.R

```
titanic$Age # prints ages

attributes(titanic) # returns object's attribute list

ls() # print list of variables defined in session

# Basic Statistics:
# Get mean, median, quantiles, minimum, maximum, variance, and standard deviation of passengers' ages:
mean(titanic$Age) # returns NA, due to missiing values

# Fix: remove missing values
mean(titanic$Age, na.rm=TRUE)
median(titanic$Age, na.rm=TRUE)
quantile(titanic$Age, na.rm=TRUE)
min(titanic$Age, na.rm=TRUE)
max(titanic$Age, na.rm=TRUE)
var(titanic$Age, na.rm=TRUE)
sd(titanic$Age, na.rm=TRUE)

# summary() function prints min, max, mean, median, and quantiles (here, also number of NA's):
summary(titanic$Age, na.rm=TRUE)

# complete.cases() returns logical vector indicating which cases are complete
# list rows of data with missing values
titanic[!complete.cases(titanic),]

# na.omit() returns object with listwise deletion of missing values
# create new dataset without missing data
titanic_no_missing_data <- na.omit(titanic)
titanic_no_missing_data # display new data set w/o missing values!

# Charting/plotting - RESEARCH THE FOLLOWING COMMANDS!)
# https://www.cyclismo.org/tutorial/R/plotting.html
help(stripchart) # help command
stripchart(titanic_no_missing_data$Age) # strip chart

# histogram
hist(titanic_no_missing_data$Age,main="Distribution of Titanic Passengers Ages",xlab="Ages")

# boxplot
boxplot(titanic_no_missing_data$Age)

# Or...plotted horizontally
boxplot(titanic_no_missing_data$Age,
        main='Distribution of Titanic Passengers Ages',
        xlab='Ages',
        horizontal=TRUE)



# Scatter plot provides graphical view of relationship between two sets of numbers:
plot(titanic_no_missing_data$Age,titanic_no_missing_data$Survived,
     main="Relationship Between Ages and Survival",
     xlab="Age",
     ylab="Survived")
```

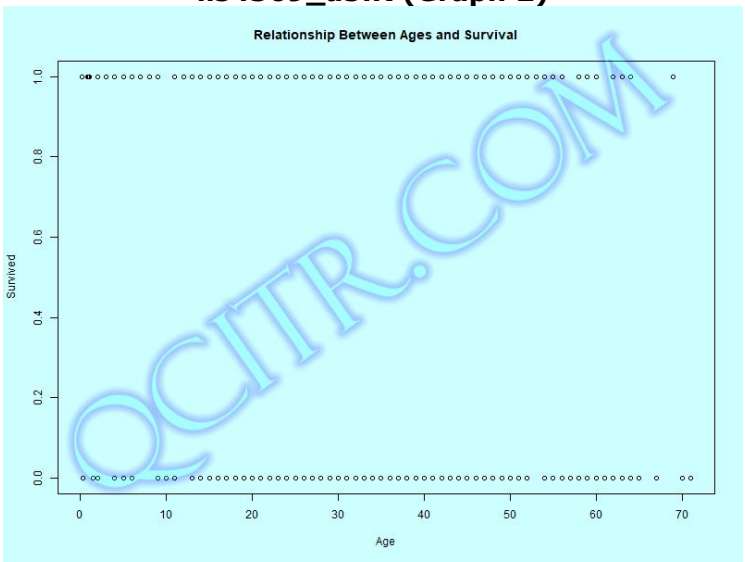
lis4369_a5.R (RStudio 4-panel Screenshot)



lis4369_a5.R (Graph 1)



lis4369_a5.R (Graph 2)



Part 3
Questions (Python: Chs. 11, 12):

1. When you subtract one datetime object from another, you get
   a datetime object
   a timedelta object
   the number of seconds between the two times
   the number of microseconds between the two times

2. You can access the parts of a date/time object by using its
   attributes
   properties
   methods
   functions

3. A dictionary stores a collection of
   ordered items
   unordered items
   mutable items
   immutable items

4. Consider the following code:

```
pets = {
   "dog": {"type": "poodle", "color": "white"},
   "cat": {"type": "Siamese", "color":  "black and white"},
   "bird": {"type": "parrot", "color": "green"}
}
pet = pets["dog"]
pet = pets["bird"]
print(pet["color"], pet["type"])
```

What will display after the code executes?
   poodle white
   white poodle
   green parrot
   parrot green

5. Consider the following code:

```
pets = {
   "dog": {"type": "poodle", "color": "white"},
   "cat": {"type": "Siamese", "color":  "black and white"},
   "bird": {"type": "parrot", "color": "green"}
}
pet = pets["dog"]
pet = pets["bird"]
print(pet["color"], pet["type"])
```

Which of the following could you add to the end of this example to print "black and white Siamese cat" to the console?

```
   pet = "cat"
print(pets[pet]["color"], pets[pet]["type"])
   pet = "cat"
print(pets[pet]["color"], pets[pet]["type"], pet)
   pet = "cat"
print(pets[pet]["type"], pets[pet]["color"], pet)
   print(pets["cat"])
```

6. Each item in a dictionary is
    a key/value pair
    a sequence
    a string
    a list

7. How many items does the following dictionary contain?

flowers = {"red": "rose", "white": "lily", "yellow": "buttercup"}
    1
    3
    6
    7

8. If each row in a list of lists has exactly two columns, you can convert it to a dictionary by using the
    list() constructor
    dict() constructor
    convert() method
    items() method

9. Consider the following code:

```
1.  flowers = {"red": "rose", "white": "lily", "yellow": "buttercup"}
2.  print(flowers)
3.  flowers["blue"] = "carnation"
4.  print(flowers)
5.  print("This is a red flower:", flowers.get("red", "none"))
6.  key = "white"
7.  if key in flowers:
8.      flower = flowers[key]
9.      print("This is a", key, "flower:", flower)
10. key = "green"
11. if key in flowers:
12.     flower = flowers[key]
13.     del flowers[key]
14.     print(flower + " was deleted")
15. else:
16.     print("There is no " + key + " flower")
```

Which of the following represents a key/value pair for the dictionary named flowers defined on line 1?
    lily/white
    red/rose
    blue/carnation
    yellow/flower

10. Consider the following code:

```
1.  flowers = {"red": "rose", "white": "lily", "yellow": "buttercup"}
2.  print(flowers)
3.  flowers["blue"] = "carnation"
4.  print(flowers)
5.  print("This is a red flower:", flowers.get("red", "none"))
6.  key = "white"
7.  if key in flowers:
8.      flower = flowers[key]
9.      print("This is a", key, "flower:", flower)
10. key = "green"
11. if key in flowers:
12.     flower = flowers[key]
13.     del flowers[key]
14.     print(flower + " was deleted")
15. else:
16.     print("There is no " + key + " flower")
```

Which of the following will be displayed by the print statement on line 4?

{'red': 'rose', 'white': 'lily', 'blue': 'carnation'}
{'red': 'rose', 'white': 'lily', 'yellow': 'buttercup'}
{'red': 'rose', 'yellow': 'buttercup', 'white': 'lily'}
{'blue': 'carnation', 'red': 'rose', 'white': 'lily', 'yellow': 'buttercup'}

11. Consider the following code:

```
1.  flowers = {"red": "rose", "white": "lily", "yellow": "buttercup"}
2.  print(flowers)
3.  flowers["blue"] = "carnation"
4.  print(flowers)
5.  print("This is a red flower:", flowers.get("red", "none"))
6.  key = "white"
7.  if key in flowers:
8.      flower = flowers[key]
9.      print("This is a", key, "flower:", flower)
10. key = "green"
11. if key in flowers:
12.     flower = flowers[key]
13.     del flowers[key]
14.     print(flower + " was deleted")
15. else:
16.     print("There is no " + key + " flower")
```

What would the print statement on line 5 display?

This is a red flower:
This is a red flower: none
This is a red flower: red
This is a red flower: rose

12. Consider the following code:

```
1.  flowers = {"red": "rose", "white": "lily", "yellow": "buttercup"}
2.  print(flowers)
3.  flowers["blue"] = "carnation"
4.  print(flowers)
5.  print("This is a red flower:", flowers.get("red", "none"))
6.  key = "white"
7.  if key in flowers:
8.      flower = flowers[key]
9.      print("This is a", key, "flower:", flower)
10. key = "green"
11. if key in flowers:
12.     flower = flowers[key]
13.     del flowers[key]
14.     print(flower + " was deleted")
15. else:
16.     print("There is no " + key + " flower")
```

What would the print statement on line 9 display?

This is a white flower: white
This is a white flower: lily
This is a lily flower: white
This is a lily flower: lily

13. Consider the following code:

```
1.  flowers = {"red": "rose", "white": "lily", "yellow": "buttercup"}
2.  print(flowers)
3.  flowers["blue"] = "carnation"
4.  print(flowers)
5.  print("This is a red flower:", flowers.get("red", "none"))
6.  key = "white"
7.  if key in flowers:
8.      flower = flowers[key]
9.      print("This is a", key, "flower:", flower)
10. key = "green"
11. if key in flowers:
12.     flower = flowers[key]
13.     del flowers[key]
14.     print(flower + " was deleted")
15. else:
16.     print("There is no " + key + " flower")
```

What is the last line that this code prints to the console?

green was deleted
lily was deleted
There is no green flower
There is no flower

14. Consider the following code:

```
1.  flowers = {"white": "lily",
               "red": "rose",
               "blue": "carnation",
               "yellow": "buttercup"}
2.  colors = list(flowers.keys())
3.  colors.sort()
4.  show_colors = "Colors of flowers: "
5.  for color in colors:
6.      show_colors += color + " "
7.  print(show_colors)
8.  pick_color = input("Enter a color: ")
9.  pick_color = pick_color.lower()
10. if pick_color in flowers:
11.     name = flowers[pick_color]
12.     print("Flower name: " + name)
13. else:
14.     print("There is no " + pick_color + " flower.")
```

What will the print statement on line 7 display?

Colors of flowers: blue red white yellow
Colors of flowers: white red blue yellow
Colors of flowers: lily rose carnation buttercup
Colors of flowers: buttercup carnation lily rose

15. Consider the following code:

```
1.  flowers = {"white": "lily",
               "red": "rose",
               "blue": "carnation",
               "yellow": "buttercup"}
2.  colors = list(flowers.keys())
3.  colors.sort()
4.  show_colors = "Colors of flowers: "
5.  for color in colors:
6.      show_colors += color + " "
7.  print(show_colors)
8.  pick_color = input("Enter a color: ")
9.  pick_color = pick_color.lower()
10. if pick_color in flowers:
11.     name = flowers[pick_color]
12.     print("Flower name: " + name)
13. else:
14.     print("There is no " + pick_color + " flower.")
```

If the user enters “Yellow” at the prompt on line 8, what does this code print to the console?

There is no Yellow flower
Flower name: Yellow
Flower name: buttercup
KeyError

16. The items() method returns a

view object containing all of the values in a dictionary
list object containing all of the values in a dictionary
view object containing all of the key/value pairs in a dictionary
list object containing all of the key/value pairs in a dictionary

17. The key in a dictionary can be
    a string
    a number
    a list
    any immutable type

18. The keys() method returns a
    view object containing all of the keys in a dictionary
    list object containing all of the keys in a dictionary
    view object containing all of the key/value pairs in a dictionary
    list object containing all of the key/value pairs in a dictionary

19. To avoid a KeyError when using the pop() method of a dictionary, you can
    use the optional second argument to supply the correct key
    use the optional second argument to supply a default value
    use the exists keyword to check whether the key exists before you call the pop() method
    use the del keyword to check whether the pop() method can delete the key without a KeyError

20. To convert a view object to a list, you can use the
    list() constructor
    dict() constructor
    convert() method
    items() method

21. To delete all items from a dictionary you can
    use the pop() method without any arguments
    use the clear() method
    use the deleteAll() method
    use the del keyword on a dictionary item