ECOTS 2018

INTRODUCTION TO R/RSTUDIO FOR NEW USERS

OUTLINE

- Part I: Personal Background
- Part II: For Students with No Background in Programming
- Part III: For Those Educating Students with No Background in Programming
- Part IV: For Those with a Background in Programming
- Appendix: External Resources

PART I

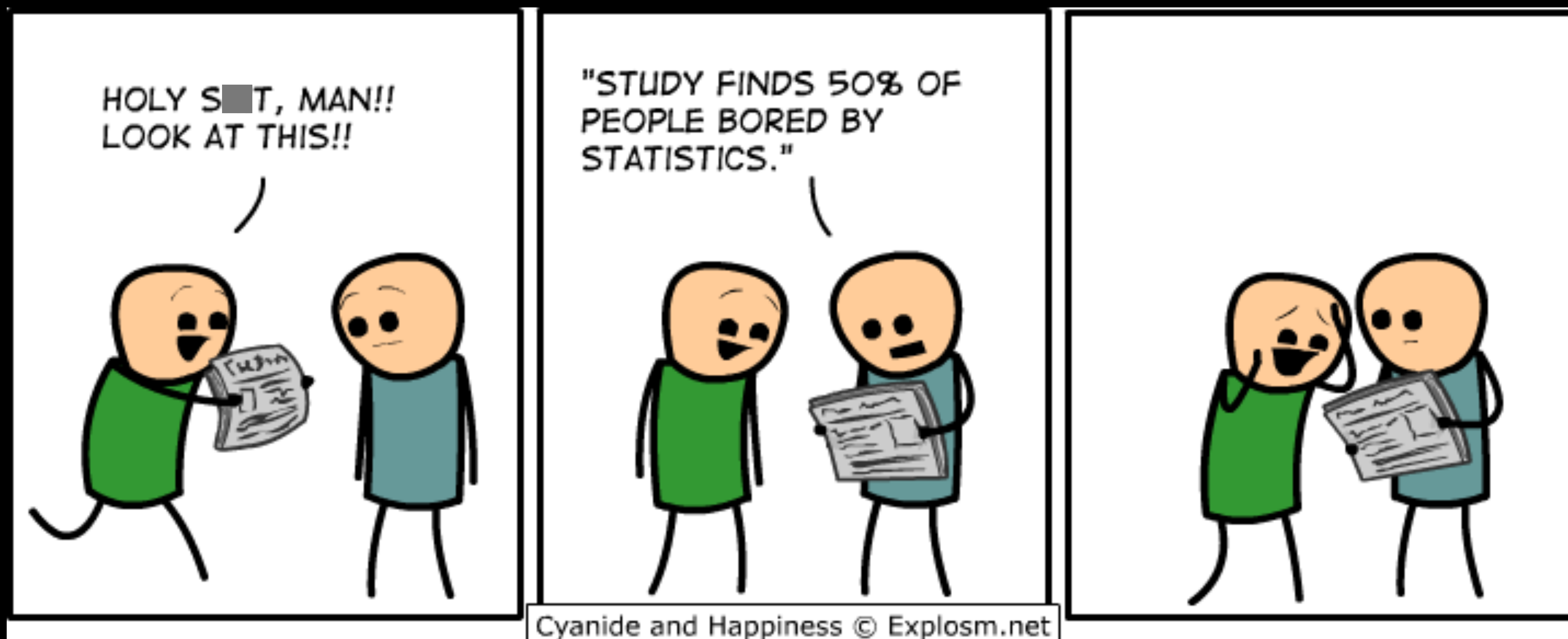
PERSONAL BACKGROUND

CHRISTOPHER PETER MAKRIS

- Studied Logic, Discrete Mathematics, & Statistics
- Graduate of Master's of Statistical Practice Program at Carnegie Mellon University
- Data Scientist
- Director of Data Science
- Programs Administrator; Adjunct Instructor Department of Statistics & Data Science at CMU


HOLY S[redacted]T, MAN!!
LOOK AT THIS!!


HOLY S[redacted]T, MAN!!
LOOK AT THIS!!
"STUDY FINDS 50% OF PEOPLE BORED BY STATISTICS."
Cyanide and Happiness © Explosm.net


HOLY S[redacted]T, MAN!!
LOOK AT THIS!!
"STUDY FINDS 50% OF PEOPLE BORED BY STATISTICS."
Cyanide and Happiness © Explosm.net

Statistics & Data Science

Object Types

Functions

Data Visualization

Statistics & Data Science

Data Transformations

Writing Scripts

EDA & Modeling

Statistics & Data Science

R/RStudio

PART II

FOR STUDENTS WITH NO BACKGROUND IN PROGRAMMING

A BRIEF HISTORY OF R

A BRIEF HISTORY OF R

- R is based on the S language, first developed in the 1960s & 1970s at Bell Laboratories.

A BRIEF HISTORY OF R

- R is based on the S language, first developed in the 1960s & 1970s at Bell Laboratories.

- Under the GNU public license, developers Ross Ihaka & Robert Gentleman released R in the 1990s.

A BRIEF HISTORY OF R

- R is based on the S language, first developed in the 1960s & 1970s at Bell Laboratories.
- Under the GNU public license, developers Ross Ihaka & Robert Gentleman released R in the 1990s.
- The popularity of R has grown because of its flexibility for data analysis, graphical tools, & free availability.

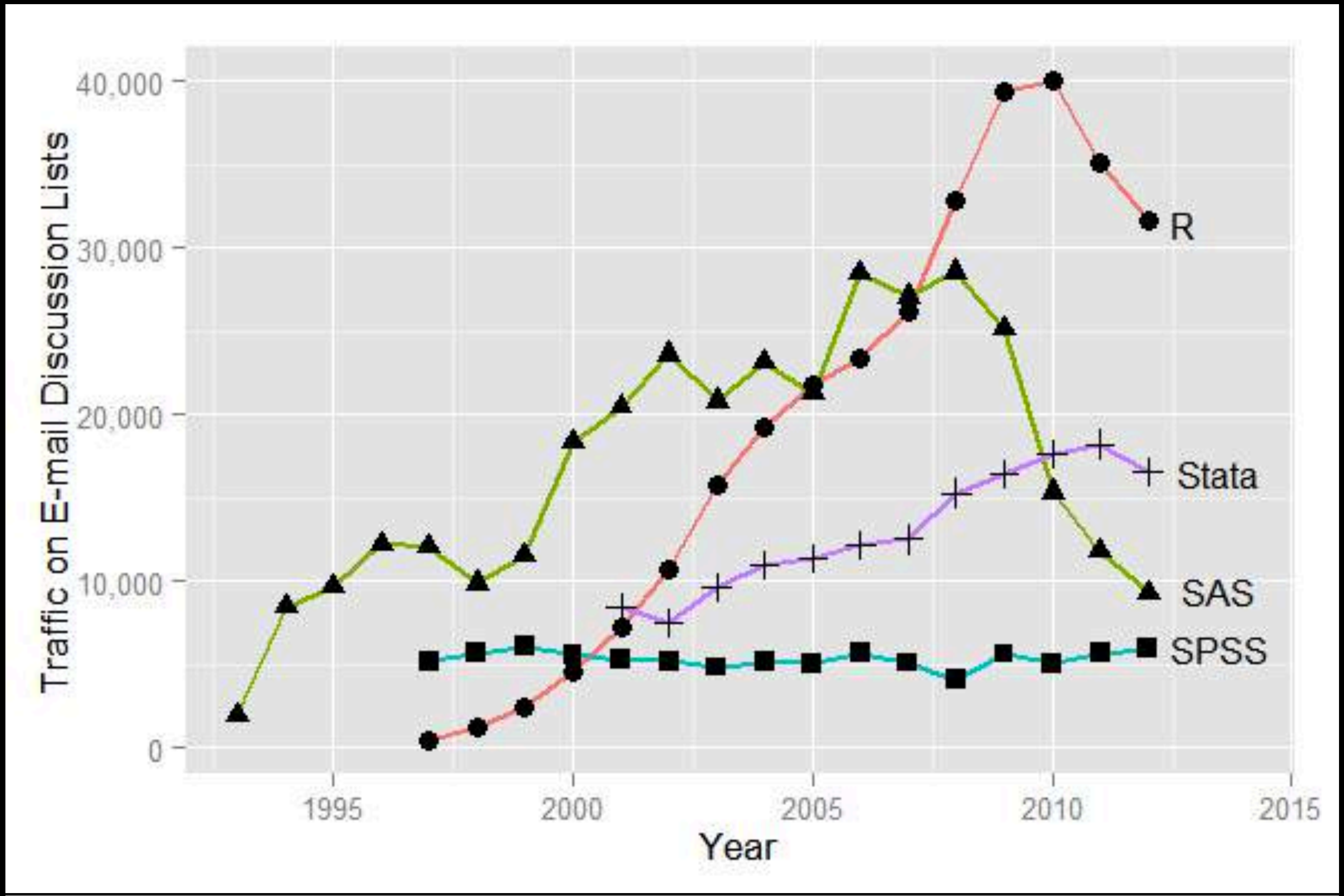
A BRIEF HISTORY OF R

- R is based on the S language, first developed in the 1960s & 1970s at Bell Laboratories.
- Under the GNU public license, developers Ross Ihaka & Robert Gentleman released R in the 1990s.
- The popularity of R has grown because of its flexibility for data analysis, graphical tools, & free availability.
- While the source code archives are maintained by the R Core Team, any researcher can contribute code via packages/libraries.

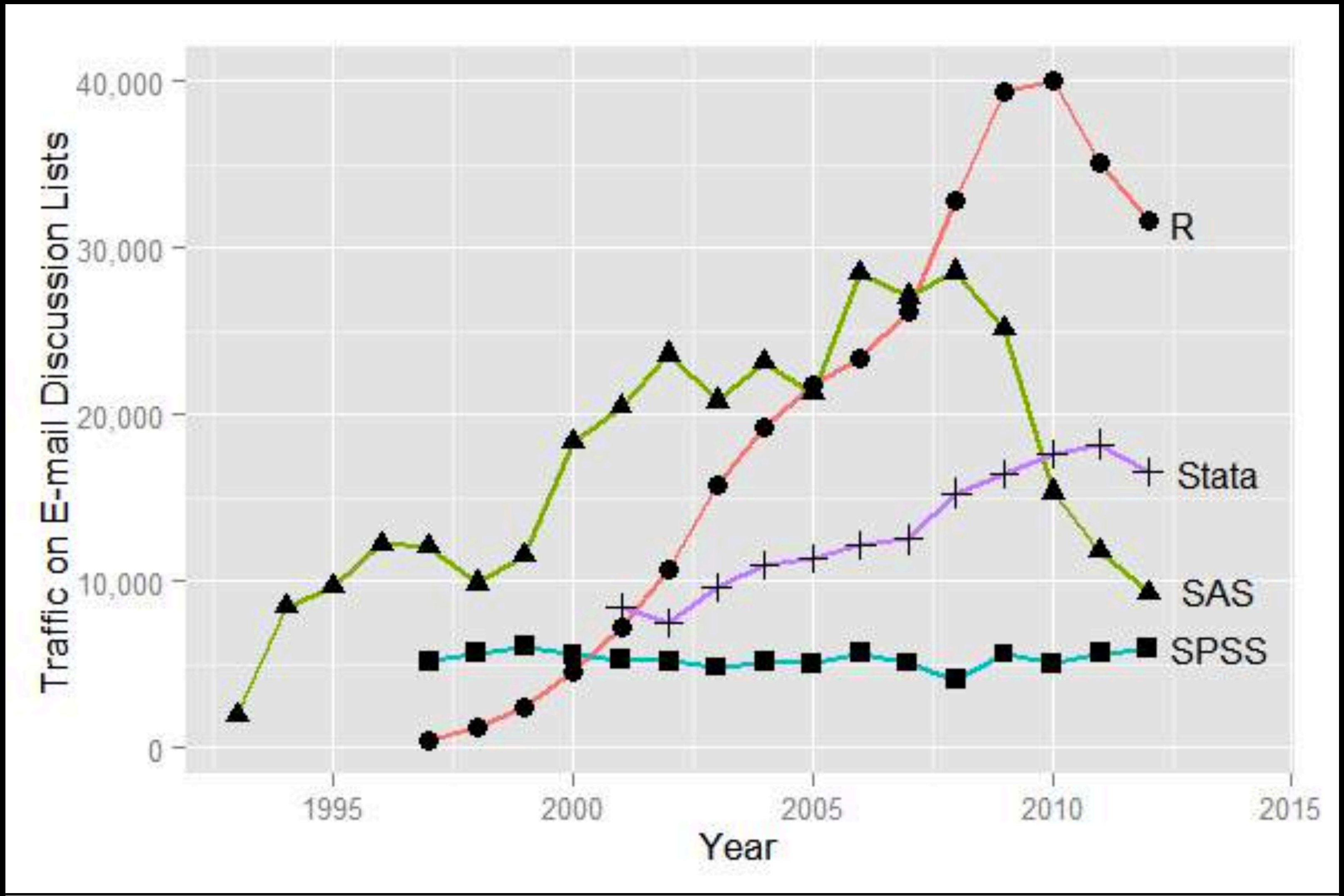
A BRIEF HISTORY OF R

- R is based on the S language, first developed in the 1960s & 1970s at Bell Laboratories.
- Under the GNU public license, developers Ross Ihaka & Robert Gentleman released R in the 1990s.
- The popularity of R has grown because of its flexibility for data analysis, graphical tools, & free availability.
- While the source code archives are maintained by the R Core Team, any researcher can contribute code via packages/libraries.
    - Updates to the core versions of R are relatively frequent, reflecting the growth of the field.

GROWTH IN POPULARITY OF R



GROWTH IN POPULARITY OF R



- Sum of monthly email traffic on each software's main listserv discussion list. (Robert A. Muenchen, StatsBlogs)


OPEN


OPEN

PHASES OF EATING ICE CREAM TOO FAST

PHASES OF EATING ICE CREAM TOO FAST

PHASES OF EATING ICE CREAM TOO FAST

PHASES OF EATING ICE CREAM TOO FAST

PHASES OF EATING ICE CREAM TOO FAST

PHASES OF EATING ICE CREAM TOO FAST

WHERE TO BEGIN FOR LEARNERS?

WHERE TO BEGIN FOR LEARNERS?

- Every decision has benefits & detractions.

WHERE TO BEGIN FOR LEARNERS?

- Every decision has benefits & detractions.
    - Many positives are accompanied by a negative.

WHERE TO BEGIN FOR LEARNERS?

- Every decision has benefits & detractions.
  - Many positives are accompanied by a negative.

- The biggest benefit of the R language is its vast flexibility to perform tasks in a myriad of ways.

WHERE TO BEGIN FOR LEARNERS?

- Every decision has benefits & detractions.
  - Many positives are accompanied by a negative.
- The biggest benefit of the R language is its vast flexibility to perform tasks in a myriad of ways.
  - Can be seen as both a blessing and a curse.

WHERE TO BEGIN FOR LEARNERS?

- Every decision has benefits & detractions.
  - Many positives are accompanied by a negative.

- The biggest benefit of the R language is its vast flexibility to perform tasks in a myriad of ways.
  - Can be seen as both a blessing and a curse.
  - What's the standard in the ever-growing field?

WHERE TO BEGIN FOR LEARNERS?

- Every decision has benefits & detractions.
  - Many positives are accompanied by a negative.
- The biggest benefit of the R language is its vast flexibility to perform tasks in a myriad of ways.
  - Can be seen as both a blessing and a curse.
  - What's the standard in the ever-growing field?
- Keep in mind The 80/20 Rule:

WHERE TO BEGIN FOR LEARNERS?

- Every decision has benefits & detractions.
  - Many positives are accompanied by a negative.
- The biggest benefit of the R language is its vast flexibility to perform tasks in a myriad of ways.
  - Can be seen as both a blessing and a curse.
  - What's the standard in the ever-growing field?
- Keep in mind The 80/20 Rule:
  - For may events, 80% of the effects come from 20% of the causes.

THE PROBLEM: COMPLEX ANSWERS TO SIMPLE QUESTIONS

THE PROBLEM: COMPLEX ANSWERS TO SIMPLE QUESTIONS

- Suppose you want to numerically analyze the variable `x` in a data frame `mydata`. In R, you could choose to run any of the following code blocks:

THE PROBLEM: COMPLEX ANSWERS TO SIMPLE QUESTIONS

- Suppose you want to numerically analyze the variable `x` in a data frame `mydata`. In R, you could choose to run any of the following code blocks:

    1. `summary(mydata$x)`

THE PROBLEM: COMPLEX ANSWERS TO SIMPLE QUESTIONS

- Suppose you want to numerically analyze the variable `x` in a data frame `mydata`. In R, you could choose to run any of the following code blocks:

    1. `summary(mydata$x)`
    2. `with(mydata, summary(x))`

THE PROBLEM: COMPLEX ANSWERS TO SIMPLE QUESTIONS

- Suppose you want to numerically analyze the variable `x` in a data frame `mydata`. In R, you could choose to run any of the following code blocks:

    1. `summary(mydata$x)`
    2. `with(mydata, summary(x))`
    3. `summary(mydata[,1])`

THE PROBLEM: COMPLEX ANSWERS TO SIMPLE QUESTIONS

- Suppose you want to numerically analyze the variable `x` in a data frame `mydata`. In R, you could choose to run any of the following code blocks:

1. `summary(mydata$x)`
2. `with(mydata, summary(x))`
3. `summary(mydata[,1])`
4. `summary(mydata$"x")`
5. `summary(mydata["x"])`
6. `summary(mydata[,"x"])`
7. `summary(mydata[["x"]])`
8. `summary(mydata[1])`
9. `summary(mydata[[1]])`
10. `attach(mydata); summary(x)`
11. `summary(select(mydata, x))`
12. `mydata %>% with(summary(x))`
13. `mydata %>% summary(.$x)`
14. `mydata %$% summary(x)`
15. `summary(subset(mydata, select=x))`

…

THE PROBLEM: COMPLEX ANSWERS TO SIMPLE QUESTIONS

- Suppose you want to numerically analyze the variable `x` in a data frame `mydata`. In R, you could choose to run any of the following code blocks:

1. `summary(mydata$x)`
2. `with(mydata, summary(x))`
3. `summary(mydata[,1])`
4. `summary(mydata$”x”)`
5. `summary(mydata[”x”])`
6. `summary(mydata[,”x”])`
7. `summary(mydata[[”x”]])`
8. `summary(mydata[1])`
9. `summary(mydata[[1]])`
10. `attach(mydata); summary(x)`
11. `summary(select(mydata, x))`
12. `mydata %>% with(summary(x))`
13. `mydata %>% summary(.$x)`
14. `mydata %$% summary(x)`
15. `summary(subset(mydata, select=x))`

…

THE SOLUTION: BEAUTY IN PARSIMONY

THE SOLUTION: BEAUTY IN PARSIMONY

- Some of the most common tasks of basic data analysis require the creation of:

THE SOLUTION: BEAUTY IN PARSIMONY

- Some of the most common tasks of basic data analysis require the creation of:
    1. Variable summaries.

THE SOLUTION: BEAUTY IN PARSIMONY

- Some of the most common tasks of basic data analysis require the creation of:
    1. Variable summaries.
    2. Transformed variables.

THE SOLUTION: BEAUTY IN PARSIMONY

- Some of the most common tasks of basic data analysis require the creation of:
  1. Variable summaries.
  2. Transformed variables.
  3. Reordered observations.

THE SOLUTION: BEAUTY IN PARSIMONY

- Some of the most common tasks of basic data analysis require the creation of:
    1. Variable summaries.
    2. Transformed variables.
    3. Reordered observations.
    4. Subsetted variables.

THE SOLUTION: BEAUTY IN PARSIMONY

- Some of the most common tasks of basic data analysis require the creation of:
    1. Variable summaries.
    2. Transformed variables.
    3. Reordered observations.
    4. Subsetted variables.
    5. Subsetted observations.

THE SOLUTION: BEAUTY IN PARSIMONY

- Some of the most common tasks of basic data analysis require the creation of:
  1. Variable summaries.
  2. Transformed variables.
  3. Reordered observations.
  4. Subsetted variables.
  5. Subsetted observations.
     - Analyze observations under certain conditions.

THE SOLUTION: BEAUTY IN PARSIMONY

- Some of the most common tasks of basic data analysis require the creation of:
    1. Variable summaries.
    2. Transformed variables.
    3. Reordered observations.
    4. Subsetted variables.
    5. Subsetted observations.
        - Analyze observations under certain conditions.

- Employ The 80/20 Rule with `dplyr` — a package for data manipulation that provides consistency and brevity:

THE SOLUTION: BEAUTY IN PARSIMONY

- Some of the most common tasks of basic data analysis require the creation of:
    1. Variable summaries.
    2. Transformed variables.
    3. Reordered observations.
    4. Subsetted variables.
    5. Subsetted observations.
        - Analyze observations under certain conditions.

- Employ The 80/20 Rule with `dplyr` — a package for data manipulation that provides consistency and brevity:
    1. Collapse many values down to a single summary with `summarize()`.

THE SOLUTION: BEAUTY IN PARSIMONY

- Some of the most common tasks of basic data analysis require the creation of:
    1. Variable summaries.
    2. Transformed variables.
    3. Reordered observations.
    4. Subsetted variables.
    5. Subsetted observations.
        - Analyze observations under certain conditions.

- Employ The 80/20 Rule with `dplyr` — a package for data manipulation that provides consistency and brevity:
    1. Collapse many values down to a single summary with `summarize()`.
    2. Create new variables with functions of existing variables with `mutate()`.

THE SOLUTION: BEAUTY IN PARSIMONY

- Some of the most common tasks of basic data analysis require the creation of:
    1. Variable summaries.
    2. Transformed variables.
    3. Reordered observations.
    4. Subsetted variables.
    5. Subsetted observations.
        - Analyze observations under certain conditions.

- Employ The 80/20 Rule with `dplyr` — a package for data manipulation that provides consistency and brevity:
    1. Collapse many values down to a single summary with `summarize()`.
    2. Create new variables with functions of existing variables with `mutate()`.
    3. Change observation order with `arrange()`.

THE SOLUTION: BEAUTY IN PARSIMONY

- Some of the most common tasks of basic data analysis require the creation of:
    1. Variable summaries.
    2. Transformed variables.
    3. Reordered observations.
    4. Subsetted variables.
    5. Subsetted observations.
        - Analyze observations under certain conditions.

- Employ The 80/20 Rule with `dplyr` — a package for data manipulation that provides consistency and brevity:
    1. Collapse many values down to a single summary with `summarize()`.
    2. Create new variables with functions of existing variables with `mutate()`.
    3. Change observation order with `arrange()`.
    4. Pick variables by their names with `select()`.

THE SOLUTION: BEAUTY IN PARSIMONY

- Some of the most common tasks of basic data analysis require the creation of:
    1. Variable summaries.
    2. Transformed variables.
    3. Reordered observations.
    4. Subsetted variables.
    5. Subsetted observations.
        - Analyze observations under certain conditions.

- Employ The 80/20 Rule with `dplyr` — a package for data manipulation that provides consistency and brevity:
    1. Collapse many values down to a single summary with `summarize()`.
    2. Create new variables with functions of existing variables with `mutate()`.
    3. Change observation order with `arrange()`.
    4. Pick variables by their names with `select()`.
    5. Pick observations by their values with `filter()`.

THE SOLUTION: BEAUTY IN PARSIMONY

- Some of the most common tasks of basic data analysis require the creation of:
    1. Variable summaries.
    2. Transformed variables.
    3. Reordered observations.
    4. Subsetted variables.
    5. Subsetted observations.
        - Analyze observations under certain conditions.

- Employ The 80/20 Rule with `dplyr` — a package for data manipulation that provides consistency and brevity:
    1. Collapse many values down to a single summary with `summarize()`.
    2. Create new variables with functions of existing variables with `mutate()`.
    3. Change observation order with `arrange()`.
    4. Pick variables by their names with `select()`.
    5. Pick observations by their values with `filter()`.
        - Analyze observations with subsets with `group_by()`.

THE SOLUTION: BEAUTY IN PARSIMONY

THE SOLUTION: BEAUTY IN PARSIMONY

- Using `dplyr`, we not only get parsimony in few functions to learn, but also consistency in operation. All of the aforementioned functions work similarly:

THE SOLUTION: BEAUTY IN PARSIMONY

- Using `dplyr`, we not only get parsimony in few functions to learn, but also consistency in operation. All of the aforementioned functions work similarly:
    - The first argument is a data frame.

THE SOLUTION: BEAUTY IN PARSIMONY

- Using `dplyr`, we not only get parsimony in few functions to learn, but also consistency in operation. All of the aforementioned functions work similarly:
  - The first argument is a data frame.
  - The subsequent arguments describe what to do with the data frame using variable names.

THE SOLUTION: BEAUTY IN PARSIMONY

- Using `dplyr`, we not only get parsimony in few functions to learn, but also consistency in operation. All of the aforementioned functions work similarly:
    - The first argument is a data frame.
    - The subsequent arguments describe what to do with the data frame using variable names.
    - The result yields a new data frame.

THE SOLUTION: BEAUTY IN PARSIMONY

- Using `dplyr`, we not only get parsimony in few functions to learn, but also consistency in operation. All of the aforementioned functions work similarly:
    - The first argument is a data frame.
    - The subsequent arguments describe what to do with the data frame using variable names.
    - The result yields a new data frame.

- Consistency in properties allows for the chaining of multiple simple steps in order to produce a complex result.

PART III

FOR THOSE EDUCATING STUDENTS WITH NO BACKGROUND IN PROGRAMMING

WHERE TO BEGIN FOR EDUCATORS?

WHERE TO BEGIN FOR EDUCATORS?

- Attempt to get inside the mind of the complete beginner.

WHERE TO BEGIN FOR EDUCATORS?

- Attempt to get inside the mind of the complete beginner.
    - Try to personify objects, methodology, syntax, etc.

WHERE TO BEGIN FOR EDUCATORS?

- Attempt to get inside the mind of the complete beginner.
    - Try to personify objects, methodology, syntax, etc.
    - Ask "why?" five times to get at the root of basic understanding.

WHERE TO BEGIN FOR EDUCATORS?

- Attempt to get inside the mind of the complete beginner.
  - Try to personify objects, methodology, syntax, etc.
  - Ask "why?" five times to get at the root of basic understanding.
  - Ask "what would you do?" to build intuition of complex tasks.

WHERE TO BEGIN FOR EDUCATORS?

- Attempt to get inside the mind of the complete beginner.
    - Try to personify objects, methodology, syntax, etc.
    - Ask "why?" five times to get at the root of basic understanding.
    - Ask "what would you do?" to build intuition of complex tasks.
- Keep in mind the common pitfalls.

WHERE TO BEGIN FOR EDUCATORS?

- Attempt to get inside the mind of the complete beginner.
  - Try to personify objects, methodology, syntax, etc.
  - Ask “why?” five times to get at the root of basic understanding.
  - Ask “what would you do?” to build intuition of complex tasks.

- Keep in mind the common pitfalls.
  - What were things you found troublesome when learning R/RStudio yourself?

WHERE TO BEGIN FOR EDUCATORS?

- Attempt to get inside the mind of the complete beginner.
    - Try to personify objects, methodology, syntax, etc.
    - Ask "why?" five times to get at the root of basic understanding.
    - Ask "what would you do?" to build intuition of complex tasks.
- Keep in mind the common pitfalls.
    - What were things you found troublesome when learning R/ RStudio yourself?
- Remember that hindsight is always 20/20.

WHERE TO BEGIN FOR EDUCATORS?

- Attempt to get inside the mind of the complete beginner.
  - Try to personify objects, methodology, syntax, etc.
  - Ask "why?" five times to get at the root of basic understanding.
  - Ask "what would you do?" to build intuition of complex tasks.
- Keep in mind the common pitfalls.
  - What were things you found troublesome when learning R/RStudio yourself?
- Remember that hindsight is always 20/20.
  - What were the "golden nuggets" you personally wish you learned much earlier?

PERSONIFICATION

PERSONIFICATION

- Goal: Understand the difference between vectors, matrices, data frames, & lists.

PERSONIFICATION

- Goal: Understand the difference between vectors, matrices, data frames, & lists.
    - Also, understand the syntax regarding list subsetting.

PERSONIFICATION

- Goal: Understand the difference between vectors, matrices, data frames, & lists.
    - Also, understand the syntax regarding list subsetting.

- Common Pitfall: Misconceptions in understanding the usage of [ ] and [[ ]] syntax.

PERSONIFICATION

- Goal: Understand the difference between vectors, matrices, data frames, & lists.
  - Also, understand the syntax regarding list subsetting.

- Common Pitfall: Misconceptions in understanding the usage of [ ] and [[ ]] syntax.

- Solution: Personify to relate a non-programming concept to learning R.

PERSONIFICATION

PERSONIFICATION

- Personification: Moving from one house to another.

PERSONIFICATION

- Personification: Moving from one house to another.
    - Lists are the “throw-in-the-kitchen-sink” object; you need to pack everything!

PERSONIFICATION

- Personification: Moving from one house to another.
  - Lists are the "throw-in-the-kitchen-sink" object; you need to pack everything!
  - Various types of objects are placed in labeled boxes.

PERSONIFICATION

- Personification: Moving from one house to another.
  - Lists are the “throw-in-the-kitchen-sink” object; you need to pack everything!
  - Various types of objects are placed in labeled boxes.
  - In order to use your items, you need to both go to the appropriate room and also open the box.

PERSONIFICATION

- Personification: Moving from one house to another.
    - Lists are the "throw-in-the-kitchen-sink" object; you need to pack everything!
    - Various types of objects are placed in labeled boxes.
    - In order to use your items, you need to both go to the appropriate room and also open the box.

PERSONIFICATION

- Personification: Moving from one house to another.
  - Lists are the "throw-in-the-kitchen-sink" object; you need to pack everything!
  - Various types of objects are placed in labeled boxes.
  - In order to use your items, you need to both go to the appropriate room and also open the box.
- Basic Necessary Insights:

PERSONIFICATION

- Personification: Moving from one house to another.
    - Lists are the "throw-in-the-kitchen-sink" object; you need to pack everything!
    - Various types of objects are placed in labeled boxes.
    - In order to use your items, you need to both go to the appropriate room and also open the box.
- Basic Necessary Insights:
    - Lists need not be homogenous.

PERSONIFICATION

- Personification: Moving from one house to another.
  - Lists are the "throw-in-the-kitchen-sink" object; you need to pack everything!
  - Various types of objects are placed in labeled boxes.
  - In order to use your items, you need to both go to the appropriate room and also open the box.
- Basic Necessary Insights:
  - Lists need not be homogenous.
  - Lists can contain objects of varied types, sizes, etc.

PERSONIFICATION

- Personification: Moving from one house to another.
    - Lists are the "throw-in-the-kitchen-sink" object; you need to pack everything!
    - Various types of objects are placed in labeled boxes.
    - In order to use your items, you need to both go to the appropriate room and also open the box.

- Basic Necessary Insights:
    - Lists need not be homogenous.
    - Lists can contain objects of varied types, sizes, etc.
    - Lists are subsetted differently.

ASKING “WHY?” FIVE TIMES

ASKING "WHY?" FIVE TIMES

- Goal: Understand the difference between `tapply()` and `sapply()`.

ASKING "WHY?" FIVE TIMES

- Goal: Understand the difference between `tapply()` and `sapply()`.
- Solution: Get at the root of basic understanding by asking why:

ASKING "WHY?" FIVE TIMES

- Goal: Understand the difference between `tapply()` and `sapply()`.
- Solution: Get at the root of basic understanding by asking why:
    - Why might we use `tapply()` instead of `sapply()`?

ASKING "WHY?" FIVE TIMES

- Goal: Understand the difference between `tapply()` and `sapply()`.
- Solution: Get at the root of basic understanding by asking why:
    - Why might we use `tapply()` instead of `sapply()`?
        1. To tabulate our results by a grouping factor instead of just summarizing in aggregate. Why?

ASKING "WHY?" FIVE TIMES

- Goal: Understand the difference between `tapply()` and `sapply()`.
- Solution: Get at the root of basic understanding by asking why:
    - Why might we use `tapply()` instead of `sapply()`?
        1. To tabulate our results by a grouping factor instead of just summarizing in aggregate. Why?
        2. To understand how relationships among our variables change based upon other variables. Why?

ASKING "WHY?" FIVE TIMES

- Goal: Understand the difference between `tapply()` and `sapply()`.
- Solution: Get at the root of basic understanding by asking why:
    - Why might we use `tapply()` instead of `sapply()`?
        1. To tabulate our results by a grouping factor instead of just summarizing in aggregate. Why?
        2. To understand how relationships among our variables change based upon other variables. Why?
        3. To see if adding an interaction term to our linear model would be appropriate. Why?

ASKING "WHY?" FIVE TIMES

- Goal: Understand the difference between `tapply()` and `sapply()`.
- Solution: Get at the root of basic understanding by asking why:
    - Why might we use `tapply()` instead of `sapply()`?
        1. To tabulate our results by a grouping factor instead of just summarizing in aggregate. Why?
        2. To understand how relationships among our variables change based upon other variables. Why?
        3. To see if adding an interaction term to our linear model would be appropriate. Why?
        4. To account for as much variation in our data as possible. Why?

ASKING "WHY?" FIVE TIMES

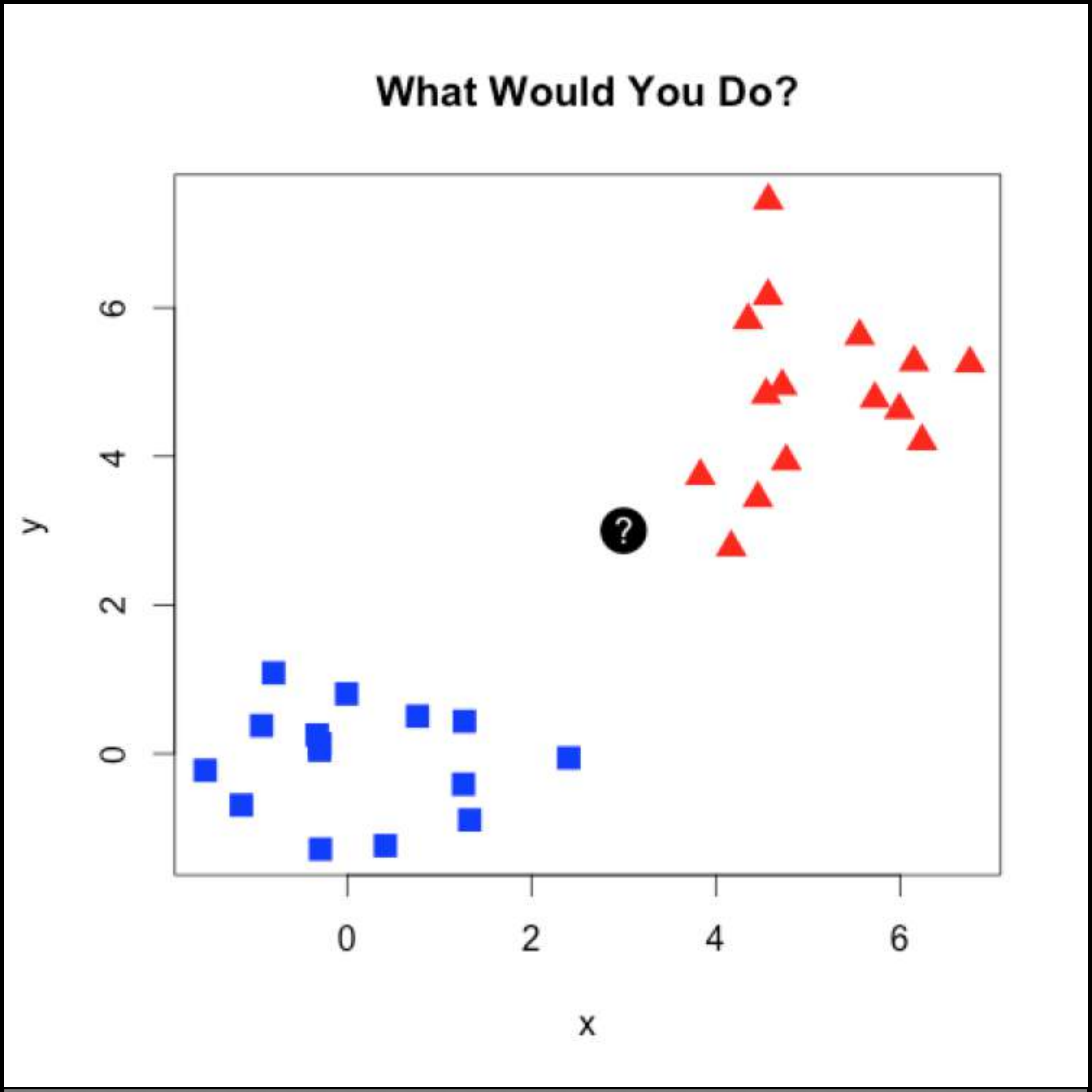
- Goal: Understand the difference between `tapply()` and `sapply()`.
- Solution: Get at the root of basic understanding by asking why:
    - Why might we use `tapply()` instead of `sapply()`?
        1. To tabulate our results by a grouping factor instead of just summarizing in aggregate. Why?
        2. To understand how relationships among our variables change based upon other variables. Why?
        3. To see if adding an interaction term to our linear model would be appropriate. Why?
        4. To account for as much variation in our data as possible. Why?
        5. To glean insight into predicting our outcome.

ASKING "WHY?" FIVE TIMES

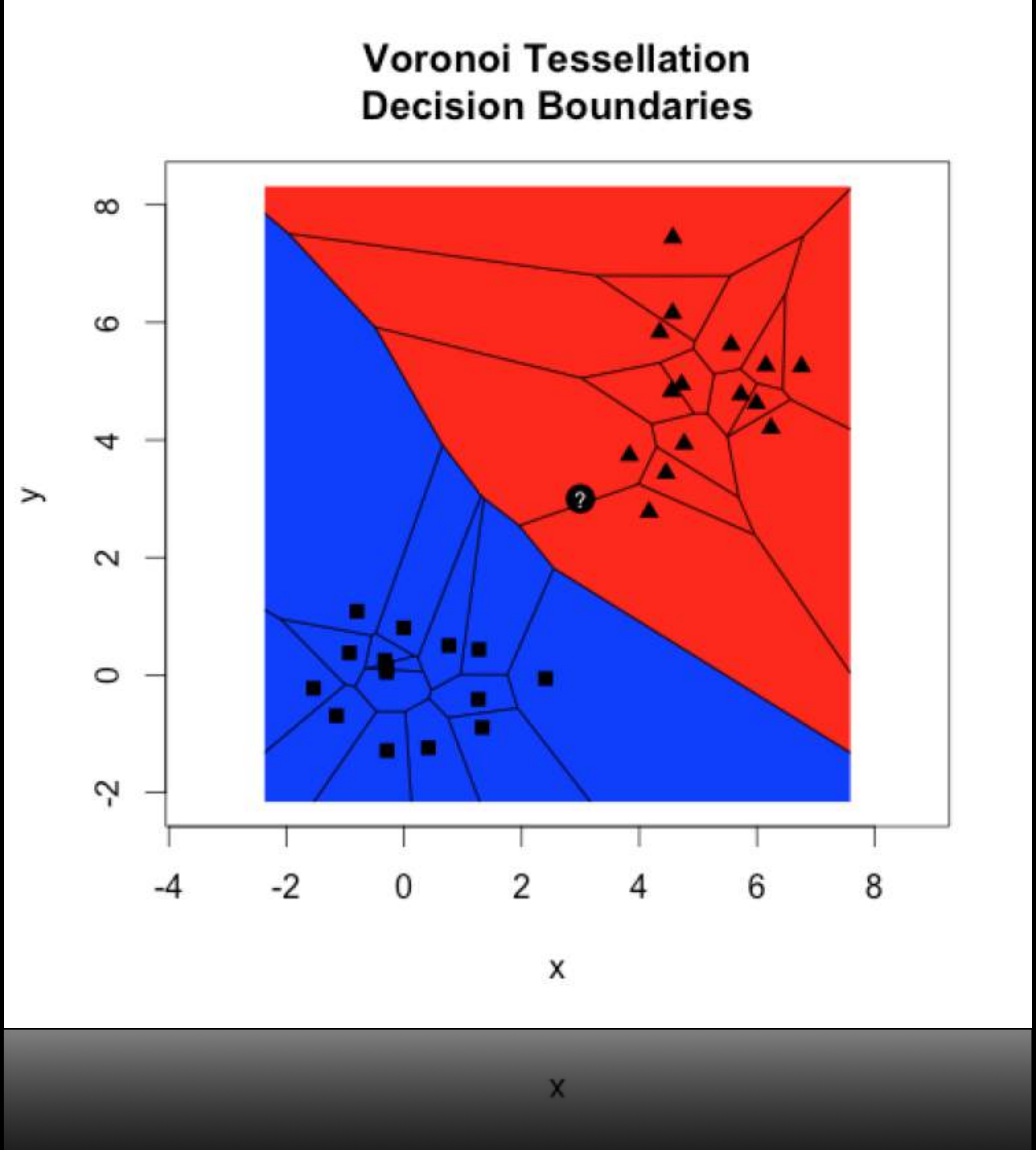
- Goal: Understand the difference between `tapply()` and `sapply()`.
- Solution: Get at the root of basic understanding by asking why:
    - Why might we use `tapply()` instead of `sapply()`?
        1. To tabulate our results by a grouping factor instead of just summarizing in aggregate. Why?
        2. To understand how relationships among our variables change based upon other variables. Why?
        3. To see if adding an interaction term to our linear model would be appropriate. Why?
        4. To account for as much variation in our data as possible. Why?
        5. To glean insight into predicting our outcome.

ASKING “WHAT WOULD YOU DO?”

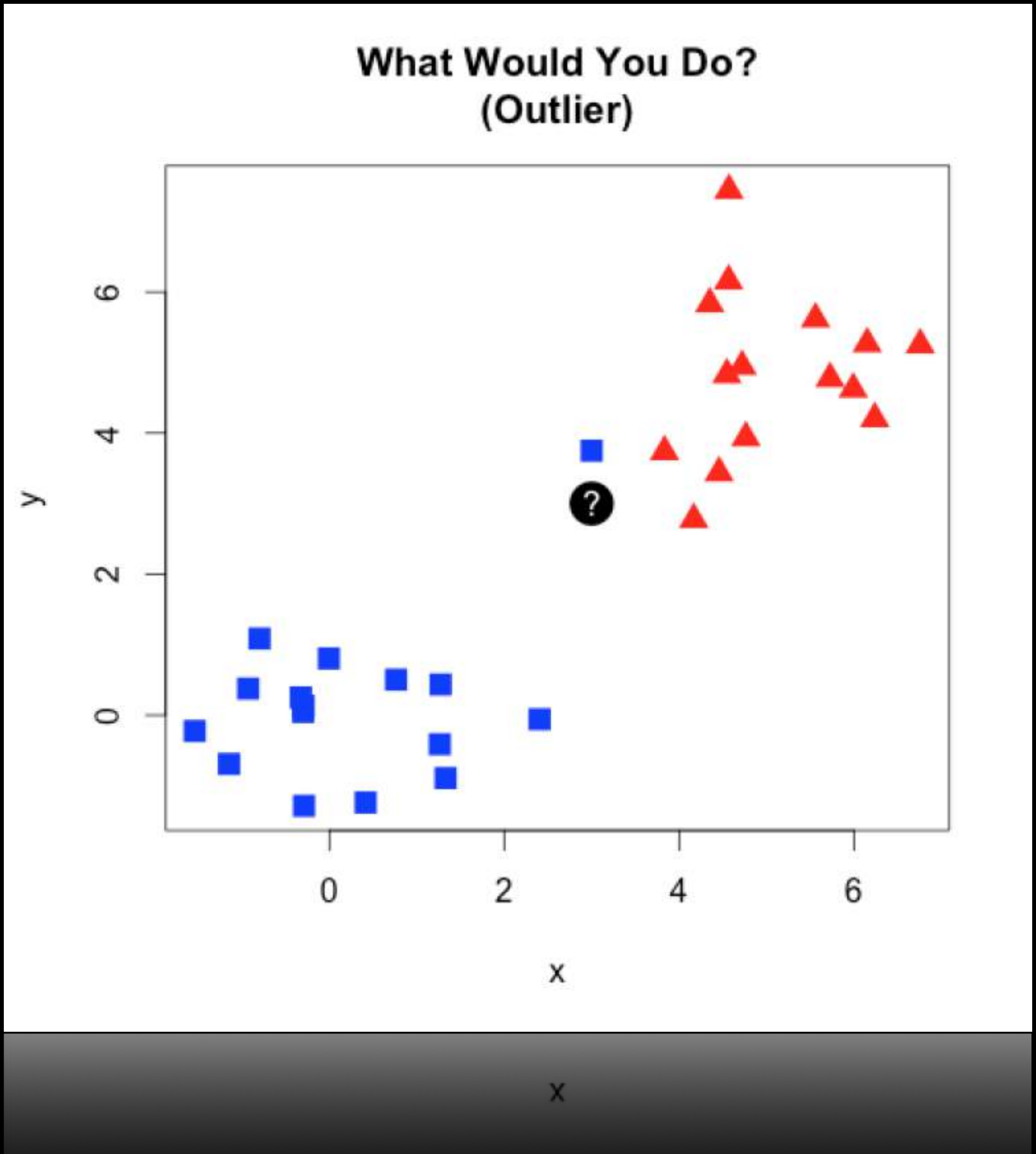
ASKING "WHAT WOULD YOU DO?"



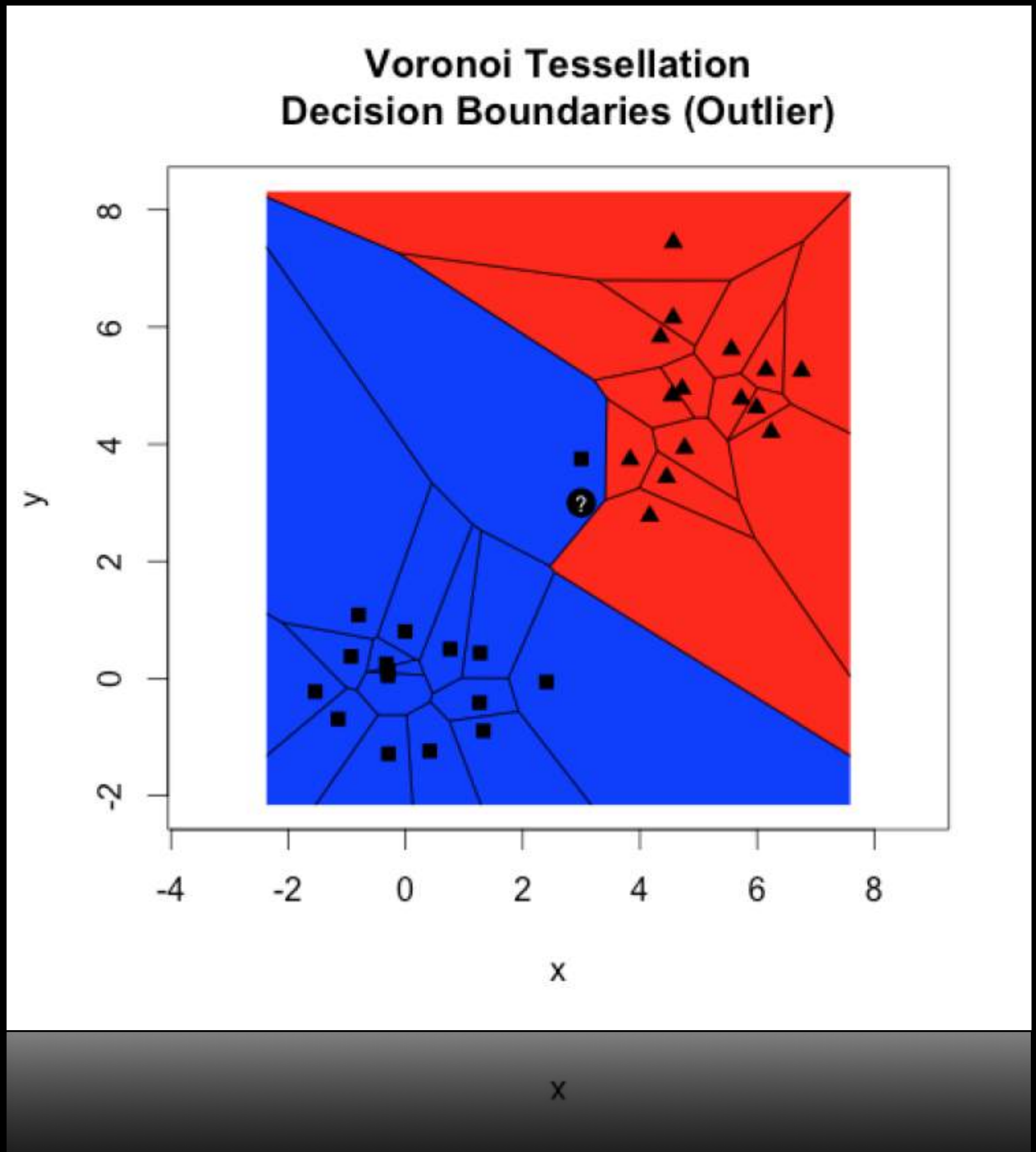
ASKING "WHAT WOULD YOU DO?"



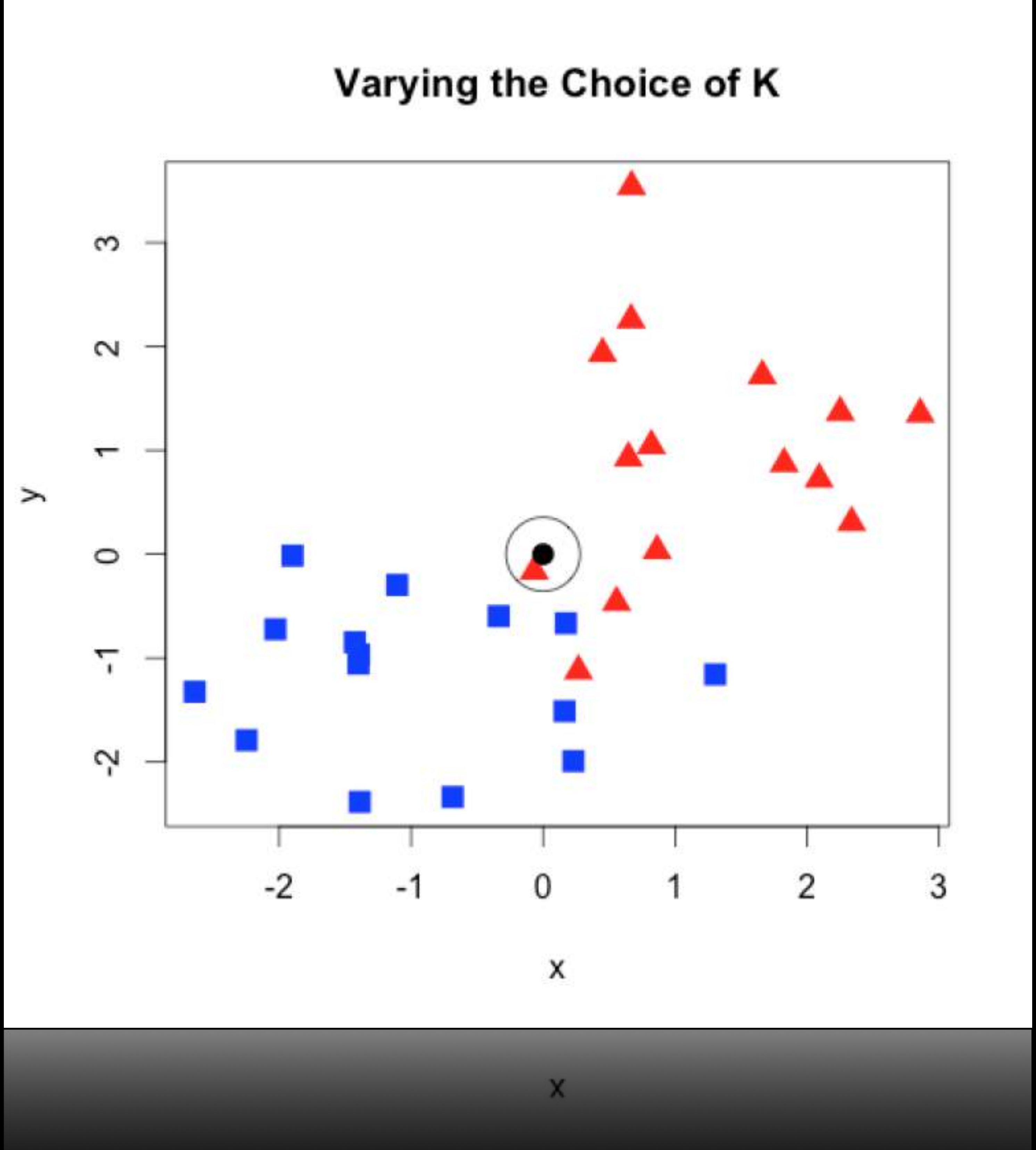
ASKING "WHAT WOULD YOU DO?"



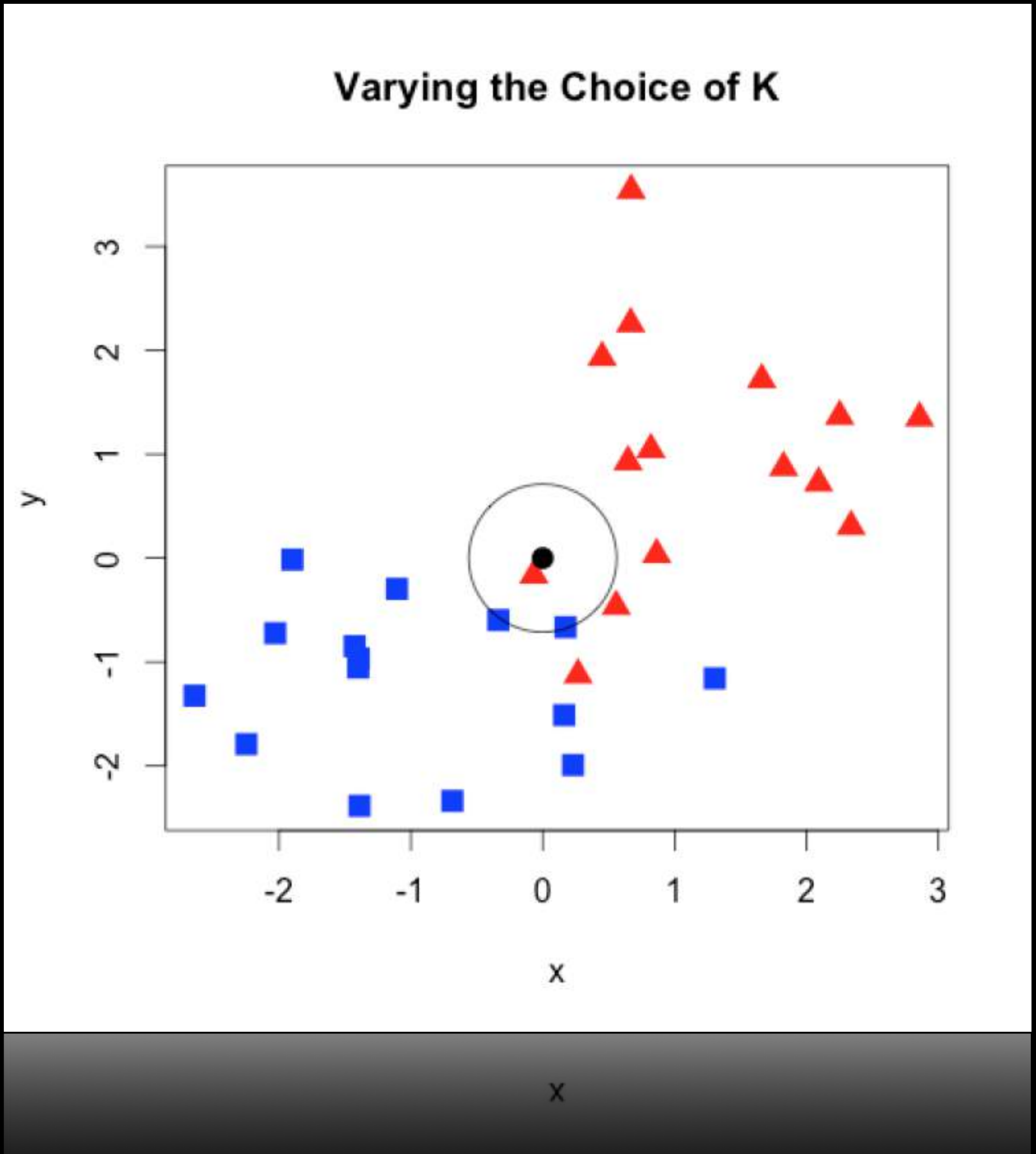
ASKING "WHAT WOULD YOU DO?"



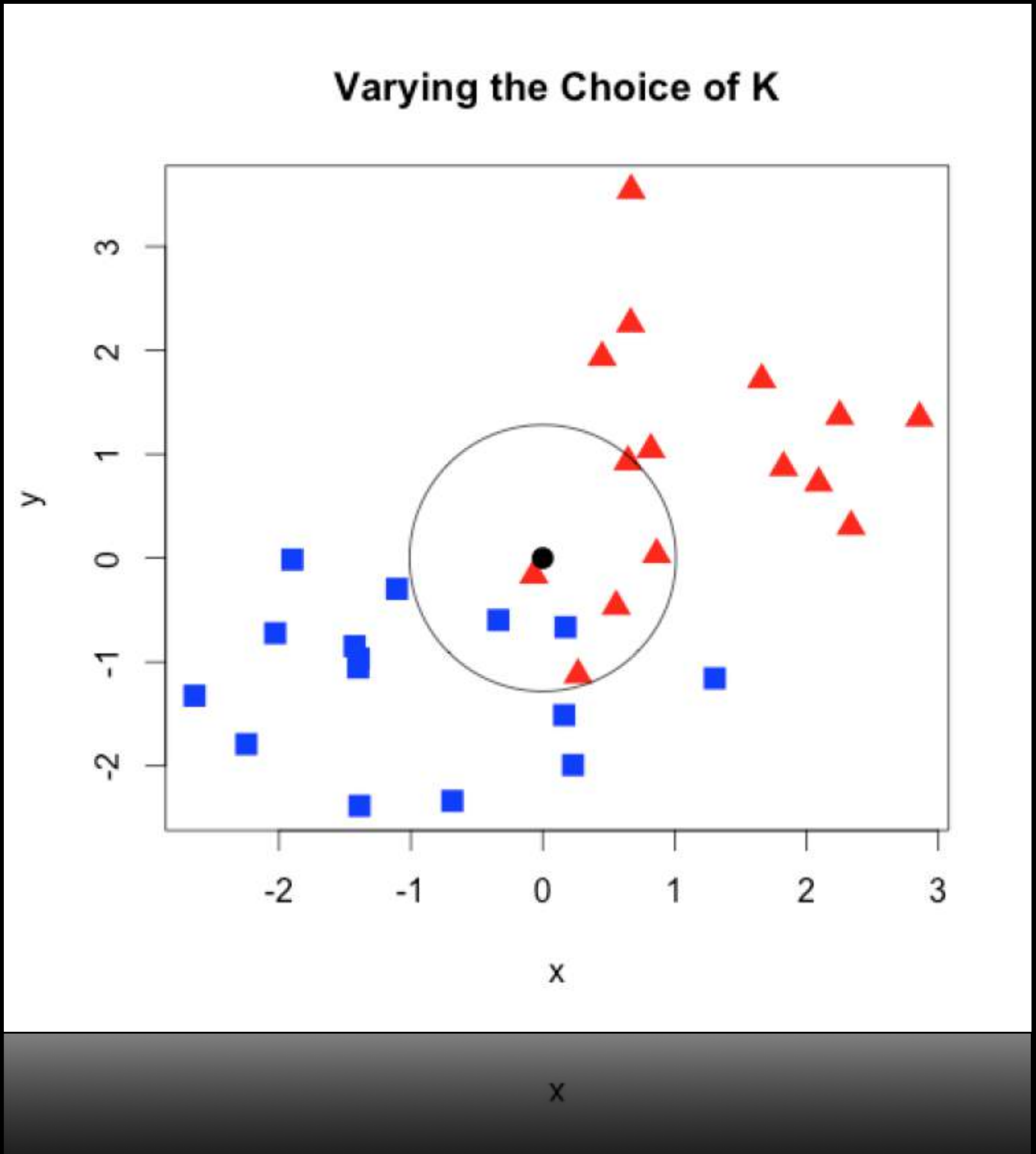
ASKING "WHAT WOULD YOU DO?"



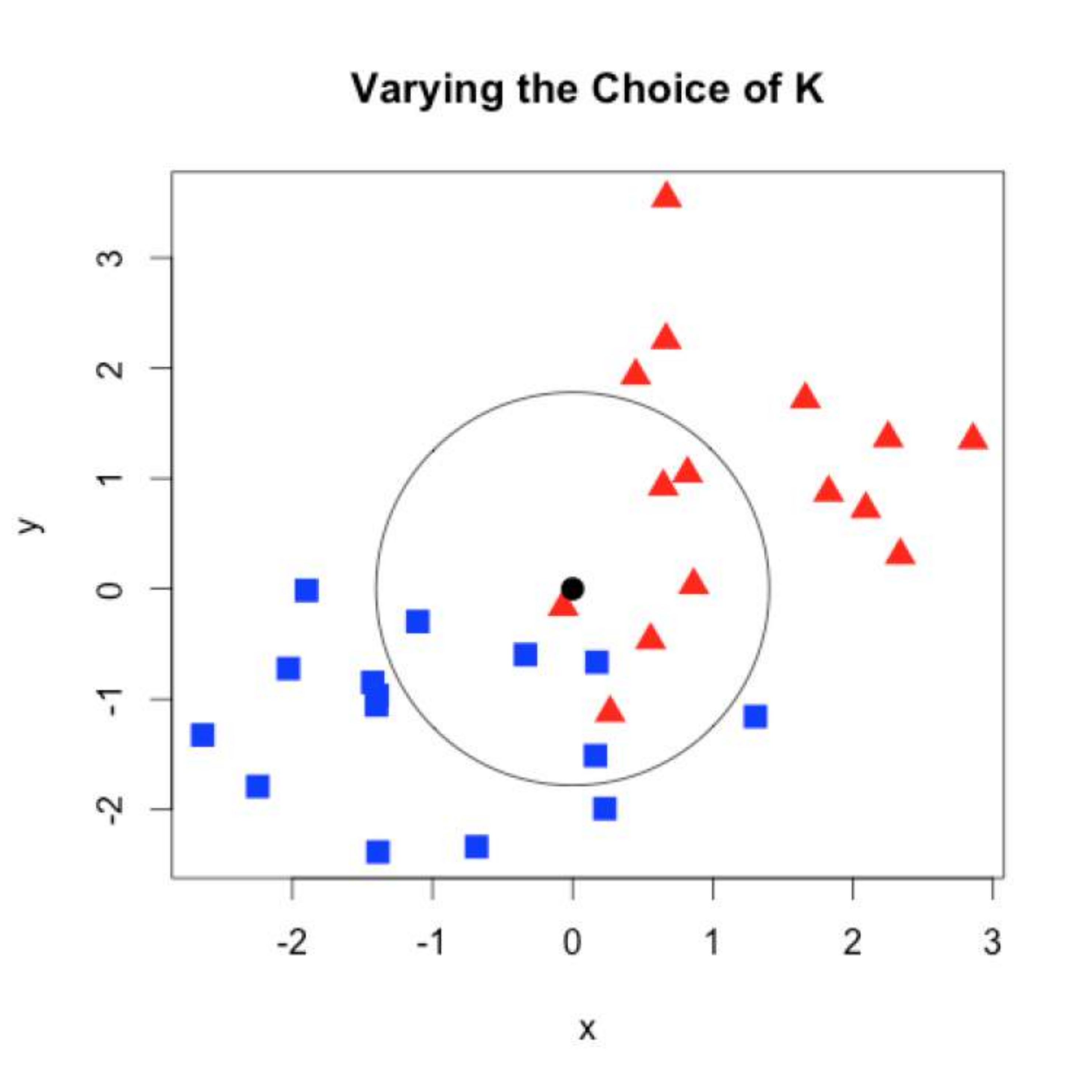
ASKING "WHAT WOULD YOU DO?"

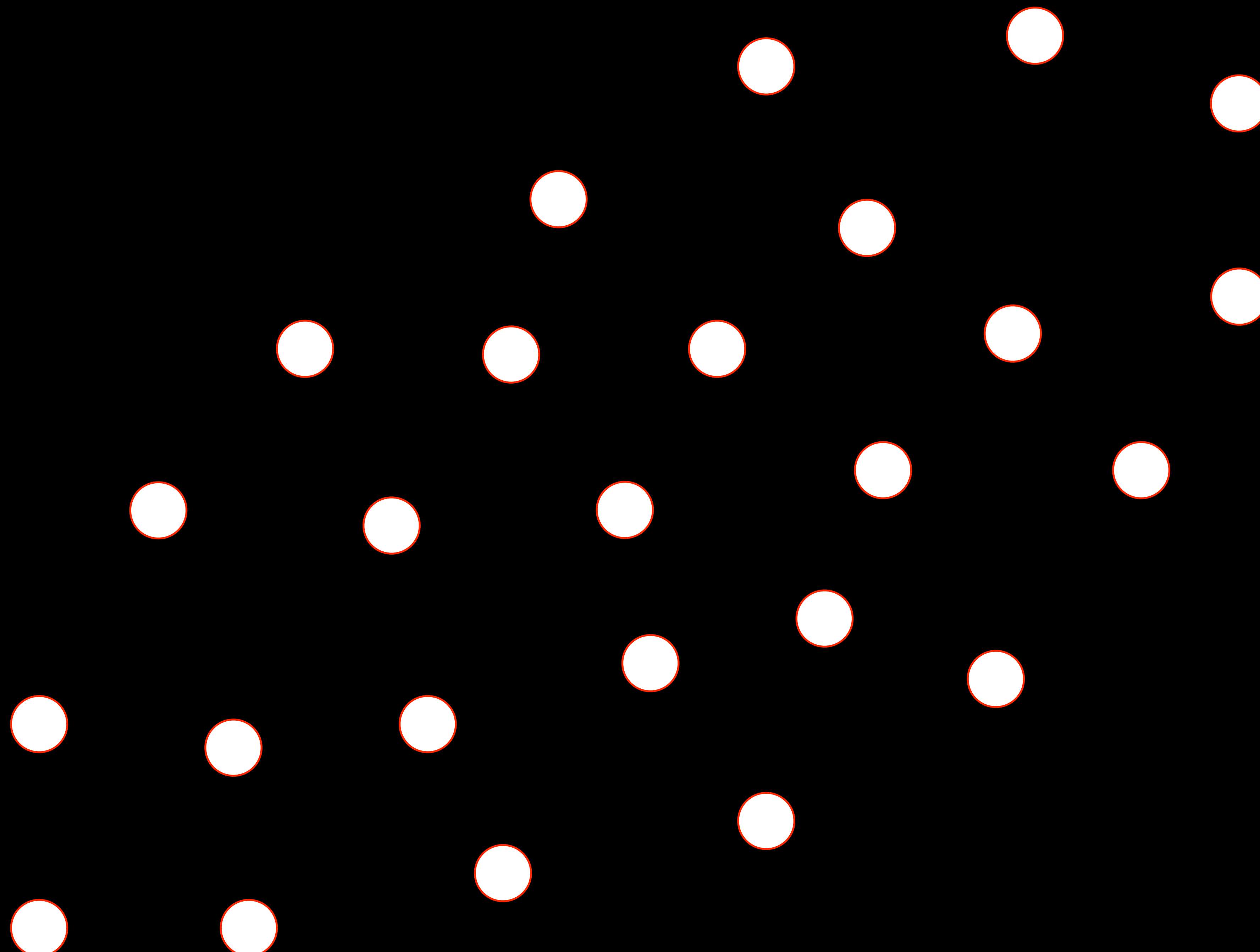


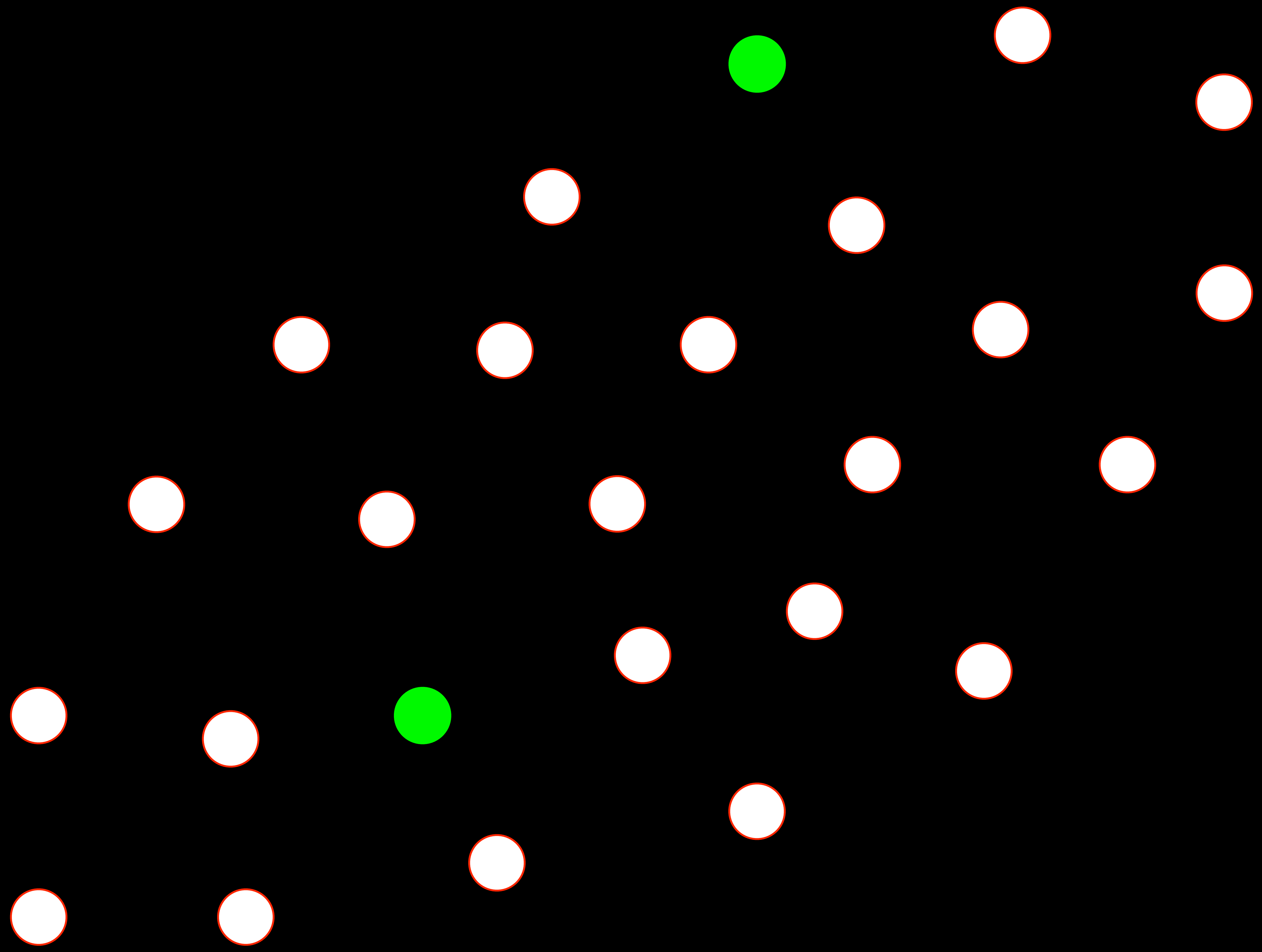
ASKING "WHAT WOULD YOU DO?"

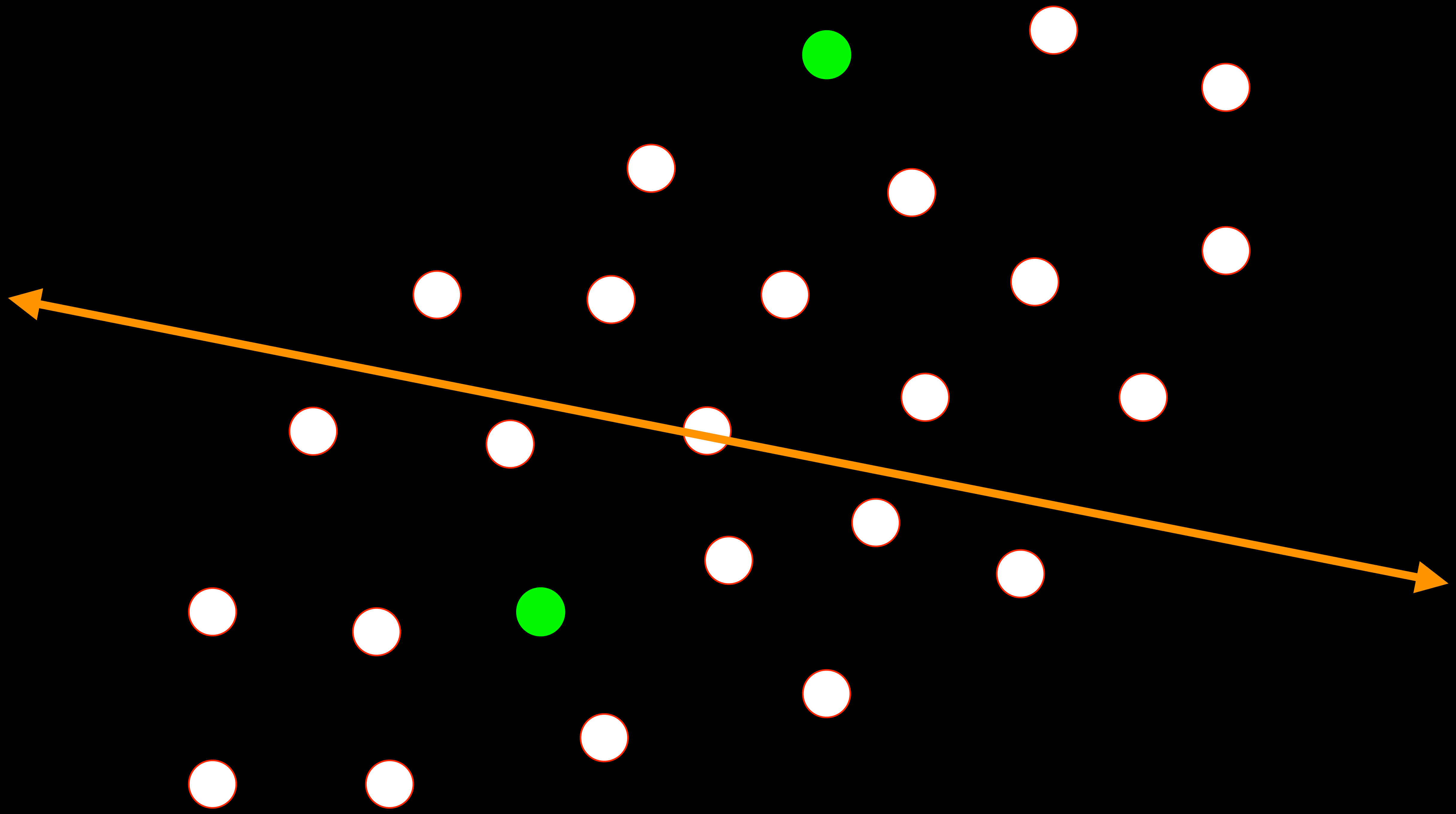


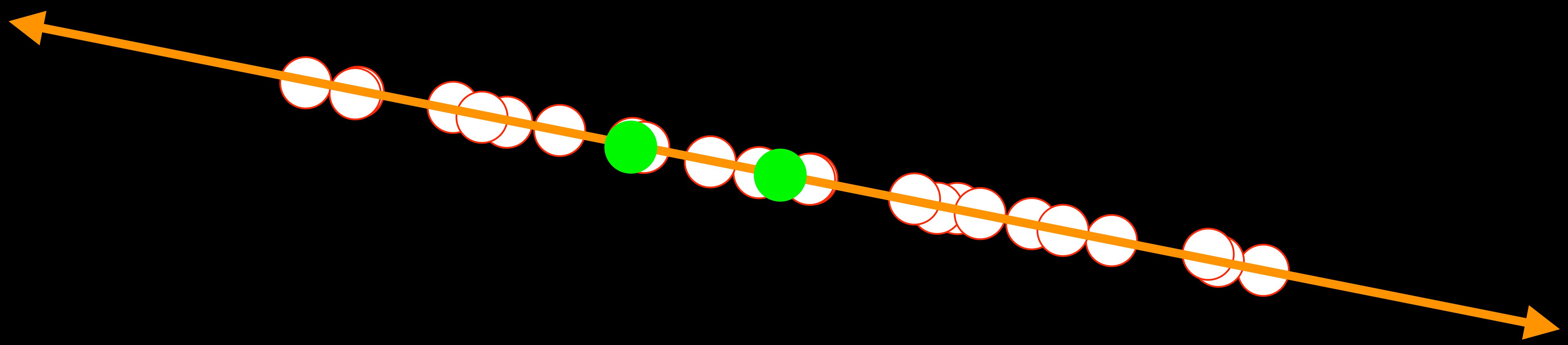
ASKING "WHAT WOULD YOU DO?"

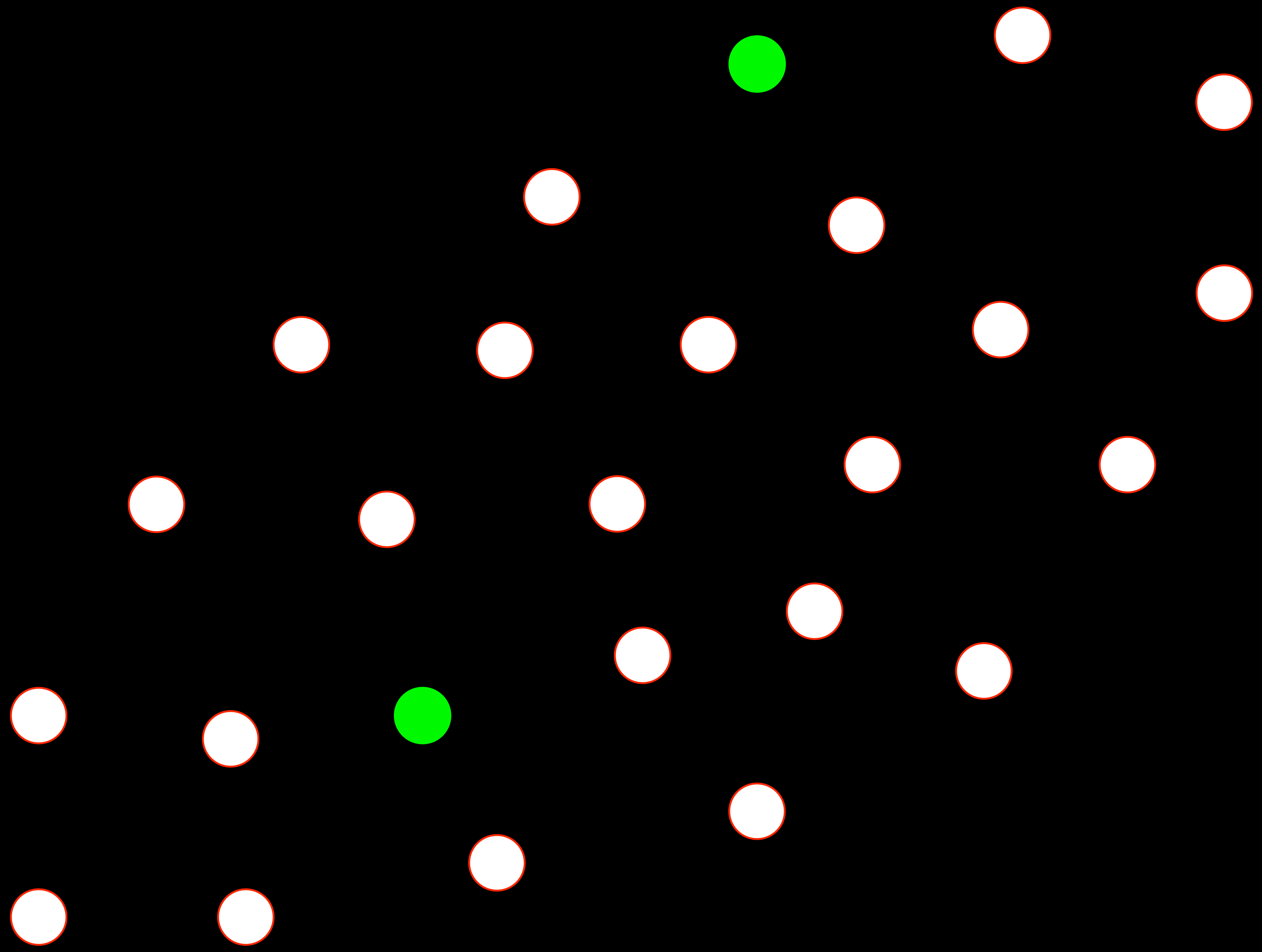


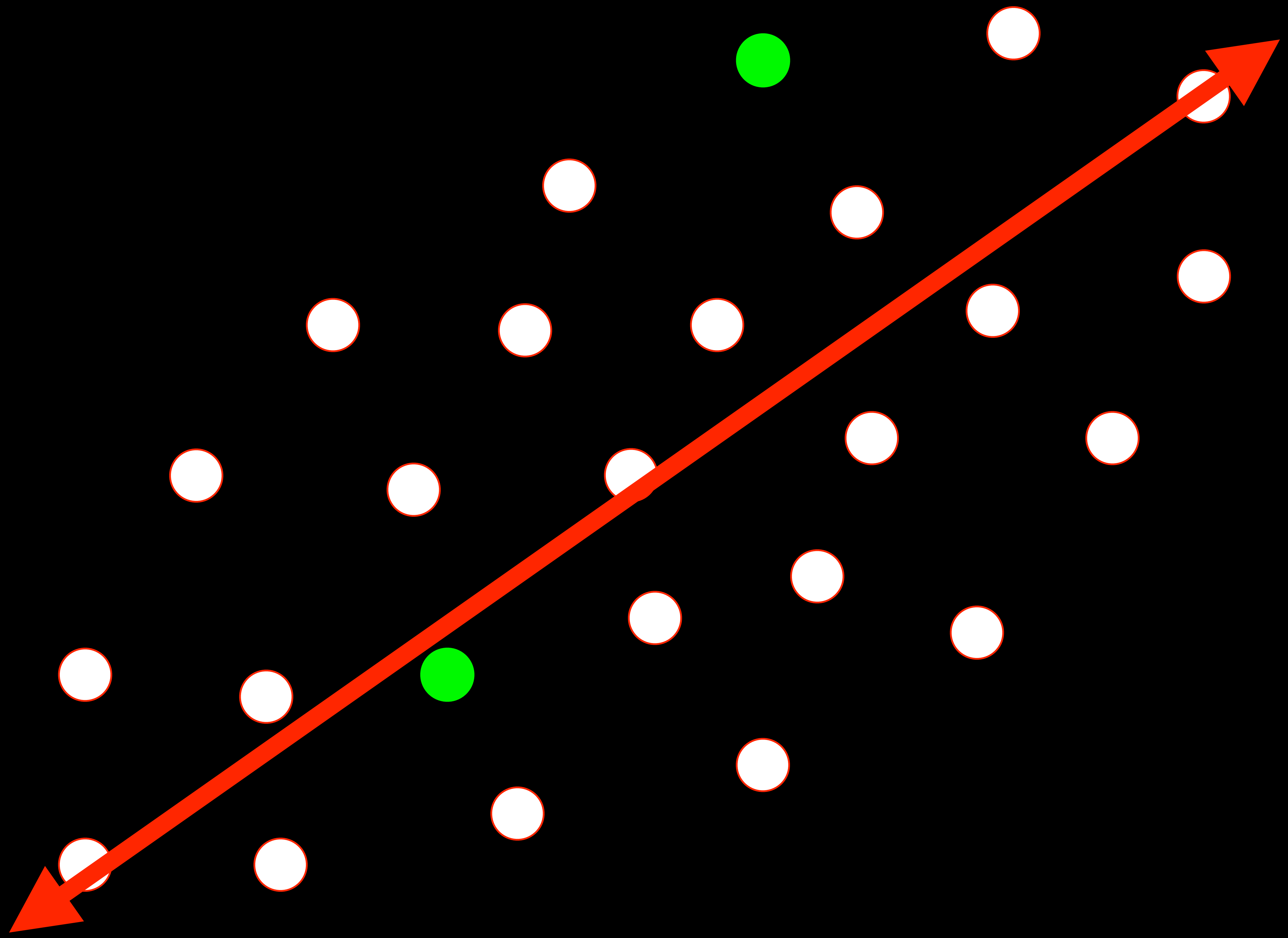


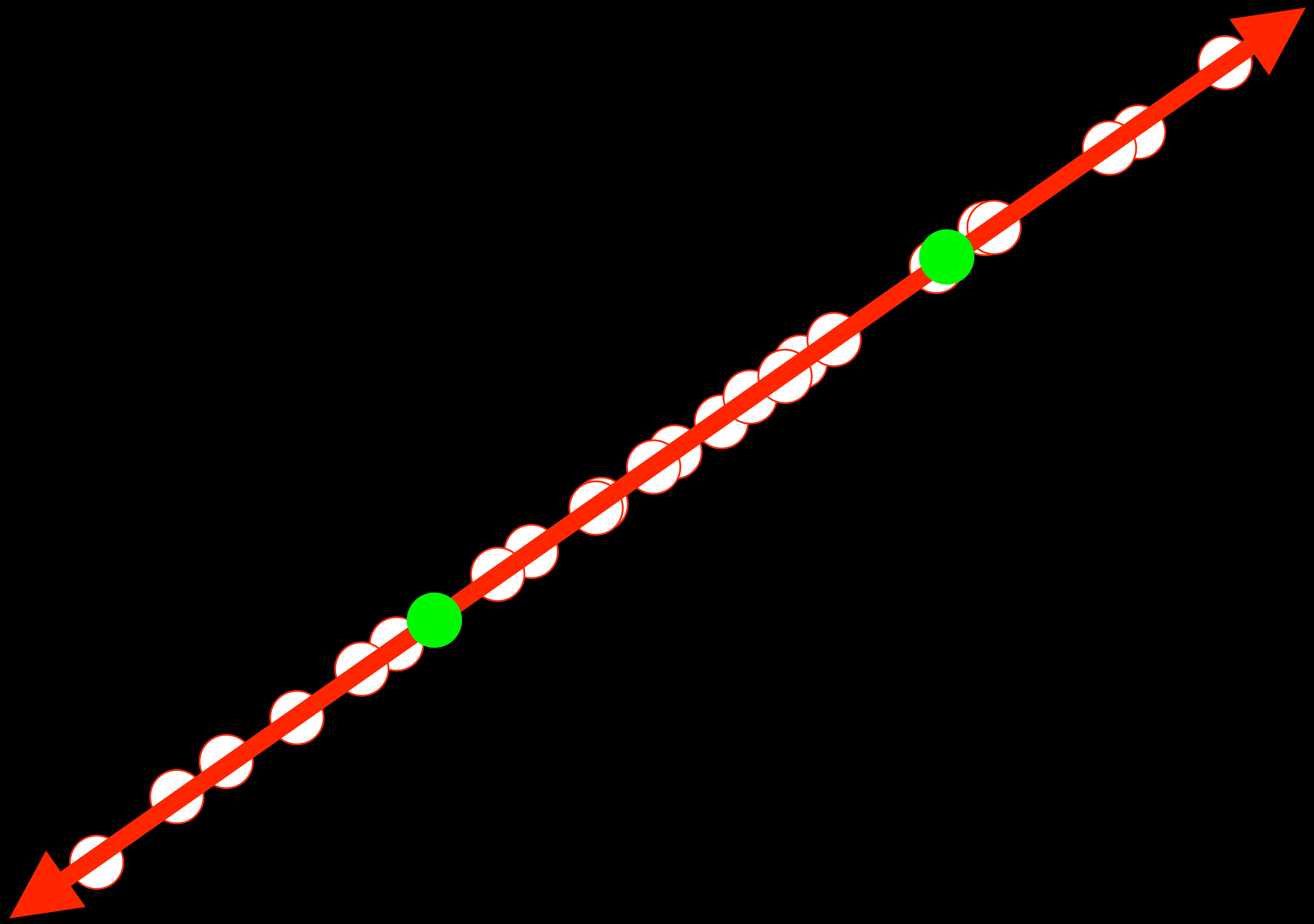


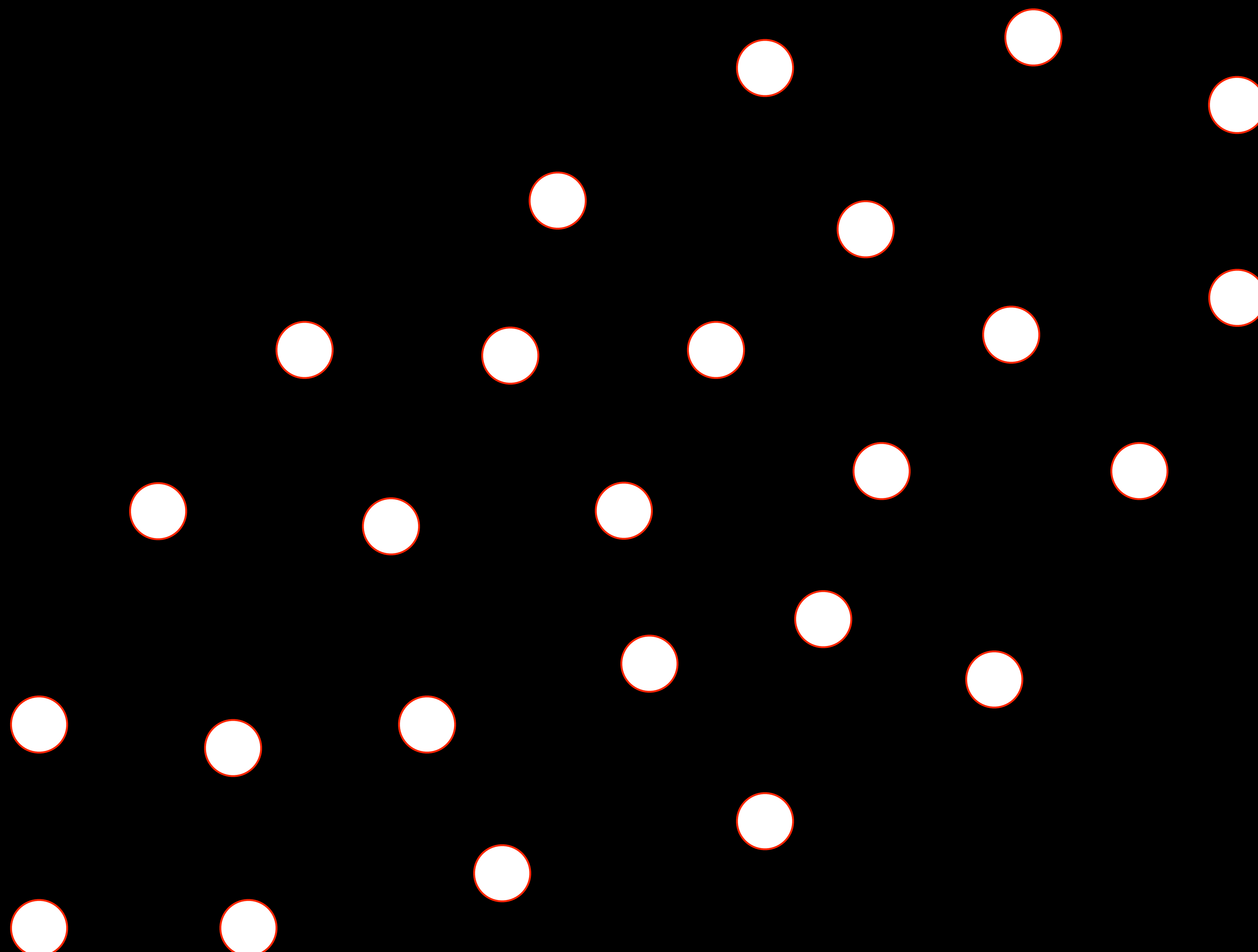


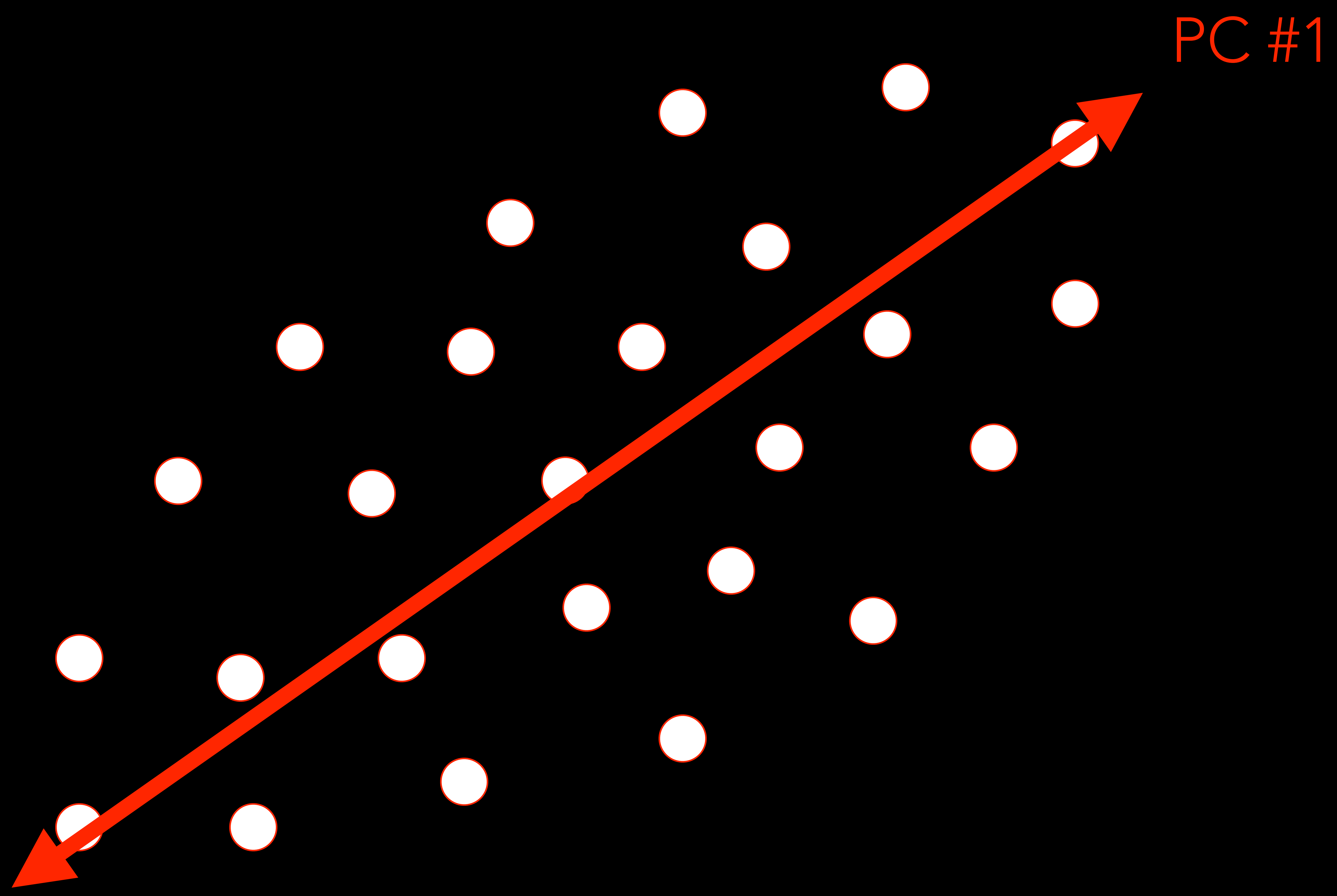


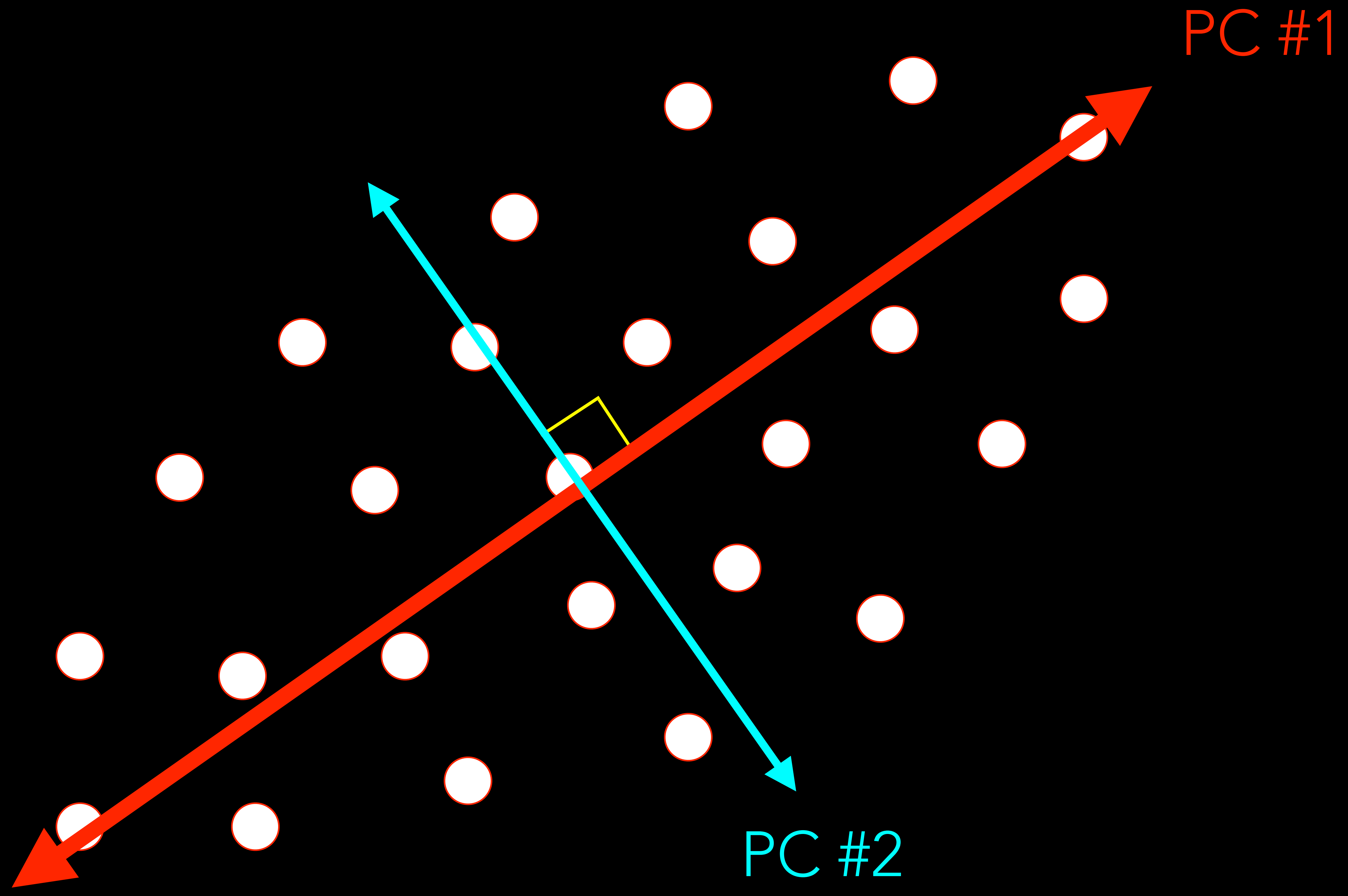


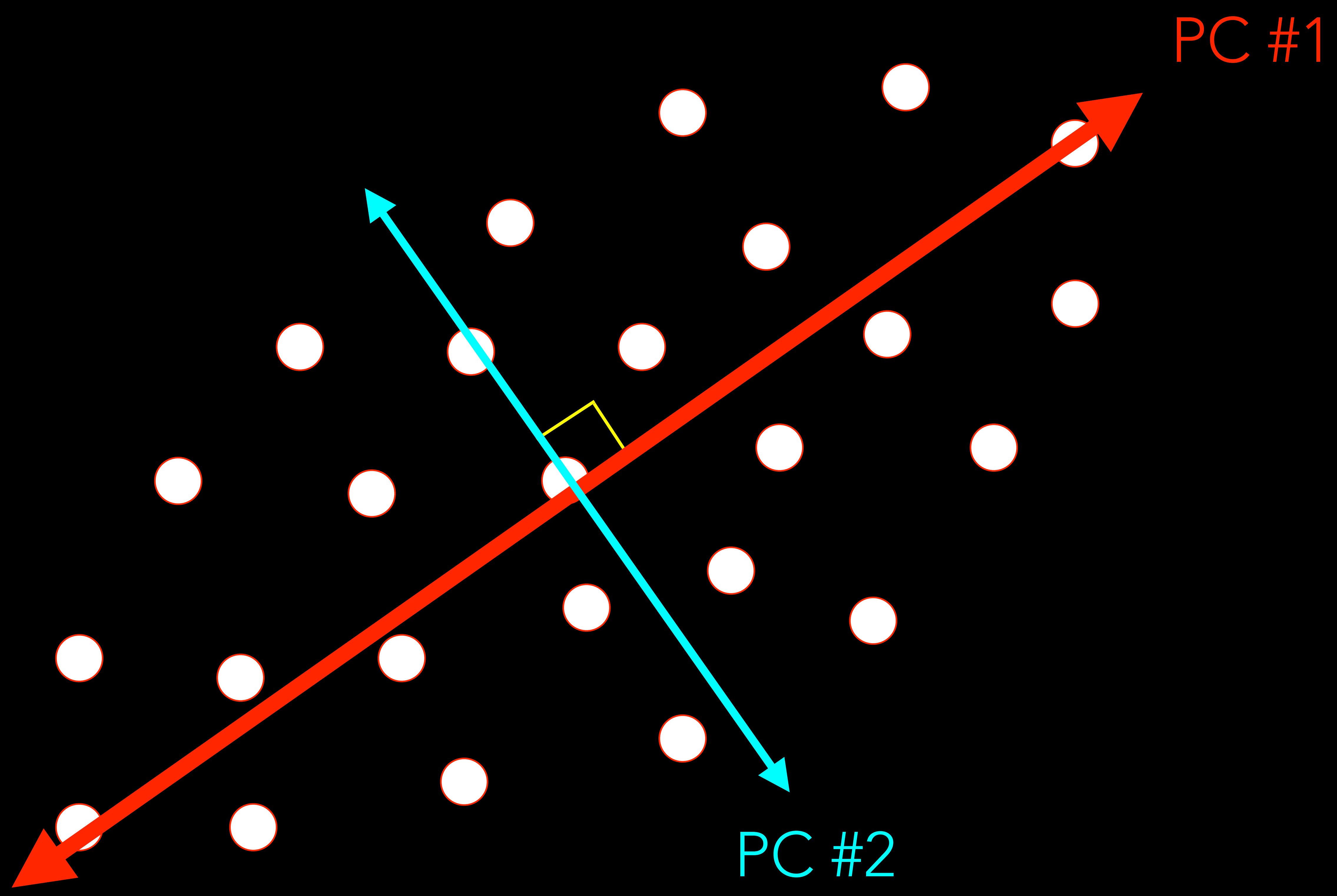


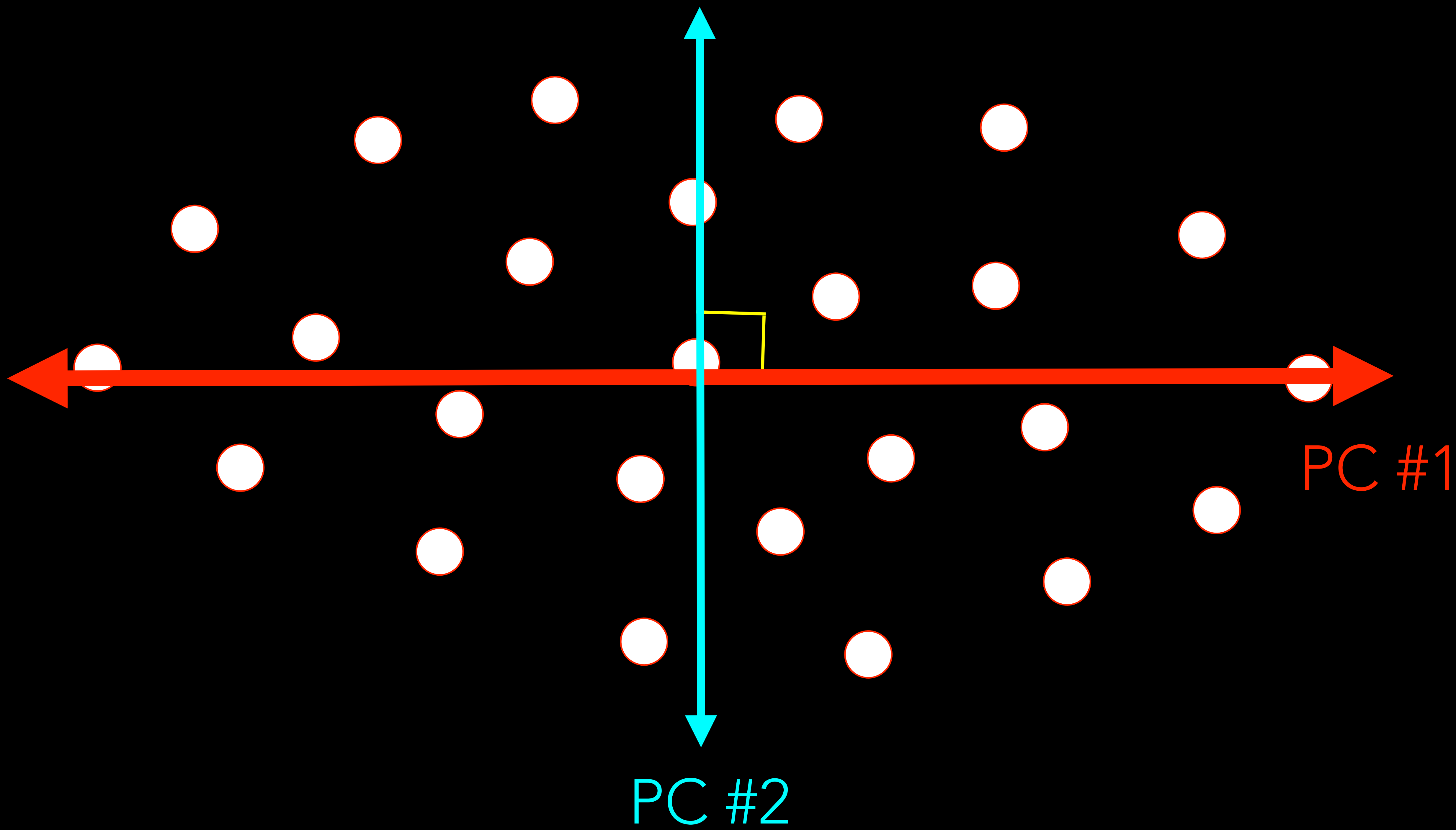





PC #1


PC #1
PC #2


PC #1
PC #2


PC #1
PC #2

WHEN IN DOUBT, TEST IT OUT

WHEN IN DOUBT, TEST IT OUT

- "Golden Nugget": Whenever encountering a new class or object in R, keep in mind some key "back-pocket" functions:

WHEN IN DOUBT, TEST IT OUT

- "Golden Nugget": Whenever encountering a new class or object in R, keep in mind some key "back-pocket" functions:
    - `summary()`

WHEN IN DOUBT, TEST IT OUT

- "Golden Nugget": Whenever encountering a new class or object in R, keep in mind some key "back-pocket" functions:
    - `summary()`
    - `str()`

WHEN IN DOUBT, TEST IT OUT

- "Golden Nugget": Whenever encountering a new class or object in R, keep in mind some key "back-pocket" functions:
    - `summary()`
    - `str()`
    - `names()`

WHEN IN DOUBT, TEST IT OUT

- “Golden Nugget”: Whenever encountering a new class or object in R, keep in mind some key “back-pocket” functions:
    - `summary()`
    - `str()`
    - `names()`
    - `help()`

WHEN IN DOUBT, TEST IT OUT

- "Golden Nugget": Whenever encountering a new class or object in R, keep in mind some key "back-pocket" functions:
    - `summary()`
    - `str()`
    - `names()`
    - `help()`
    - `plot()`

PART IV

FOR THOSE WITH A BACKGROUND IN PROGRAMMING

WHY R MAY BE HARD FOR PYTHON PROGRAMMERS

WHY R MAY BE HARD FOR PYTHON PROGRAMMERS

- Broadly speaking:

WHY R MAY BE HARD FOR PYTHON PROGRAMMERS

- Broadly speaking:
    - Python is a general-purpose programming language with the functionality to perform various statistical computations.

WHY R MAY BE HARD FOR PYTHON PROGRAMMERS

- Broadly speaking:
    - Python is a general-purpose programming language with the functionality to perform various statistical computations.
    - R is a statistical programming language with the functionality to perform like a general-purpose programming language.

WHY R MAY BE HARD FOR PYTHON PROGRAMMERS

- Broadly speaking:
  - Python is a general-purpose programming language with the functionality to perform various statistical computations.
  - R is a statistical programming language with the functionality to perform like a general-purpose programming language.

- Golden Nugget:

WHY R MAY BE HARD FOR PYTHON PROGRAMMERS

- Broadly speaking:
    - Python is a general-purpose programming language with the functionality to perform various statistical computations.
    - R is a statistical programming language with the functionality to perform like a general-purpose programming language.

- Golden Nugget:
    - `map(my_function, my_data)` in Python:

WHY R MAY BE HARD FOR PYTHON PROGRAMMERS

- Broadly speaking:
    - Python is a general-purpose programming language with the functionality to perform various statistical computations.
    - R is a statistical programming language with the functionality to perform like a general-purpose programming language.

- Golden Nugget:
    - `map(my_function, my_data)` in Python:
        - "Map my function to this particular data."

WHY R MAY BE HARD FOR PYTHON PROGRAMMERS

- Broadly speaking:
  - Python is a general-purpose programming language with the functionality to perform various statistical computations.
  - R is a statistical programming language with the functionality to perform like a general-purpose programming language.

- Golden Nugget:
  - `map(my_function, my_data)` in Python:
    - "Map my function to this particular data."
  - `apply(my_data, my_function)` in R:

WHY R MAY BE HARD FOR PYTHON PROGRAMMERS

- Broadly speaking:
  - Python is a general-purpose programming language with the functionality to perform various statistical computations.
  - R is a statistical programming language with the functionality to perform like a general-purpose programming language.

- Golden Nugget:
  - `map(my_function, my_data)` in Python:
    - “Map my function to this particular data.”
  - `apply(my_data, my_function)` in R:
    - “Apply to my data this particular function.”

WHY R MAY BE HARD FOR PYTHON PROGRAMMERS

- Broadly speaking:
    - Python is a general-purpose programming language with the functionality to perform various statistical computations.
    - R is a statistical programming language with the functionality to perform like a general-purpose programming language.

- Golden Nugget:
    - `map(my_function, my_data)` in Python:
        - “Map my function to this particular data.”
    - `apply(my_data, my_function)` in R:
        - “Apply to my data this particular function.”

- Common Pitfalls: Housekeeping on syntax.

WHY R MAY BE HARD FOR PYTHON PROGRAMMERS

- Broadly speaking:
  - Python is a general-purpose programming language with the functionality to perform various statistical computations.
  - R is a statistical programming language with the functionality to perform like a general-purpose programming language.

- Golden Nugget:
  - `map(my_function, my_data)` in Python:
    - "Map my function to this particular data."
  - `apply(my_data, my_function)` in R:
    - "Apply to my data this particular function."

- Common Pitfalls: Housekeeping on syntax.
  - Indexing in Python starts at 0; indexing in R starts at 1.

WHY R MAY BE HARD FOR PYTHON PROGRAMMERS

- Broadly speaking:
  - Python is a general-purpose programming language with the functionality to perform various statistical computations.
  - R is a statistical programming language with the functionality to perform like a general-purpose programming language.

- Golden Nugget:
  - `map(my_function, my_data)` in Python:
    - “Map my function to this particular data.”
  - `apply(my_data, my_function)` in R:
    - “Apply to my data this particular function.”

- Common Pitfalls: Housekeeping on syntax.
  - Indexing in Python starts at 0; indexing in R starts at 1.
  - Python right-truncates when slicing; R does not right-truncate.

WHY R MAY BE HARD FOR PYTHON PROGRAMMERS

- Broadly speaking:
  - Python is a general-purpose programming language with the functionality to perform various statistical computations.
  - R is a statistical programming language with the functionality to perform like a general-purpose programming language.

- Golden Nugget:
  - `map(my_function, my_data)` in Python:
    - "Map my function to this particular data."
  - `apply(my_data, my_function)` in R:
    - "Apply to my data this particular function."

- Common Pitfalls: Housekeeping on syntax.
  - Indexing in Python starts at 0; indexing in R starts at 1.
  - Python right-truncates when slicing; R does not right-truncate.
  - Python uses "." for accessing/applying functions; R uses "." as a character.

WHY R MAY BE HARD FOR SAS PROGRAMMERS

WHY R MAY BE HARD FOR SAS PROGRAMMERS

- Common Benefits of R over SAS:

WHY R MAY BE HARD FOR SAS PROGRAMMERS

- Common Benefits of R over SAS:
    - Reading data to/from various file types is comparatively much simpler; you need not write complex DATA statements that require deep knowledge of the construct of the data.

WHY R MAY BE HARD FOR SAS PROGRAMMERS

- Common Benefits of R over SAS:
    - Reading data to/from various file types is comparatively much simpler; you need not write complex DATA statements that require deep knowledge of the construct of the data.
    - Beauty in parsimony: often receive output tailored specifically to the goal at hand.

WHY R MAY BE HARD FOR SAS PROGRAMMERS

- Common Benefits of R over SAS:
  - Reading data to/from various file types is comparatively much simpler; you need not write complex DATA statements that require deep knowledge of the construct of the data.
  - Beauty in parsimony: often receive output tailored specifically to the goal at hand.
  - Interactivity in the command line.

WHY R MAY BE HARD FOR SAS PROGRAMMERS

- Common Benefits of R over SAS:
  - Reading data to/from various file types is comparatively much simpler; you need not write complex DATA statements that require deep knowledge of the construct of the data.
  - Beauty in parsimony: often receive output tailored specifically to the goal at hand.
  - Interactivity in the command line.
  - Help documentation readily available.

WHY R MAY BE HARD FOR SAS PROGRAMMERS

- Common Benefits of R over SAS:
    - Reading data to/from various file types is comparatively much simpler; you need not write complex DATA statements that require deep knowledge of the construct of the data.
    - Beauty in parsimony: often receive output tailored specifically to the goal at hand.
    - Interactivity in the command line.
    - Help documentation readily available.

- Common Benefits of SAS over R:

WHY R MAY BE HARD FOR SAS PROGRAMMERS

- Common Benefits of R over SAS:
  - Reading data to/from various file types is comparatively much simpler; you need not write complex DATA statements that require deep knowledge of the construct of the data.
  - Beauty in parsimony: often receive output tailored specifically to the goal at hand.
  - Interactivity in the command line.
  - Help documentation readily available.

- Common Benefits of SAS over R:
  - The Output Delivery System makes it easy to produce HTML or .pdf files of output.

WHY R MAY BE HARD FOR SAS PROGRAMMERS

- Common Benefits of R over SAS:
    - Reading data to/from various file types is comparatively much simpler; you need not write complex DATA statements that require deep knowledge of the construct of the data.
    - Beauty in parsimony: often receive output tailored specifically to the goal at hand.
    - Interactivity in the command line.
    - Help documentation readily available.

- Common Benefits of SAS over R:
    - The Output Delivery System makes it easy to produce HTML or .pdf files of output.
        - Consider learning R Markdown as a substitute.

WHY R MAY BE HARD FOR SAS PROGRAMMERS

- Common Benefits of R over SAS:
    - Reading data to/from various file types is comparatively much simpler; you need not write complex DATA statements that require deep knowledge of the construct of the data.
    - Beauty in parsimony: often receive output tailored specifically to the goal at hand.
    - Interactivity in the command line.
    - Help documentation readily available.

- Common Benefits of SAS over R:
    - The Output Delivery System makes it easy to produce HTML or .pdf files of output.
        - Consider learning R Markdown as a substitute.
    - Simple PROC commands produce extended, often superfluous output.

WHY R MAY BE HARD FOR SAS PROGRAMMERS

- Common Benefits of R over SAS:
  - Reading data to/from various file types is comparatively much simpler; you need not write complex DATA statements that require deep knowledge of the construct of the data.
  - Beauty in parsimony: often receive output tailored specifically to the goal at hand.
  - Interactivity in the command line.
  - Help documentation readily available.

- Common Benefits of SAS over R:
  - The Output Delivery System makes it easy to produce HTML or .pdf files of output.
    - Consider learning R Markdown as a substitute.
  - Simple PROC commands produce extended, often superfluous output.
    - Consider learning the basic theory behind the statistical models or inference tests you desire to perform; do not rely upon searching among a torrent of output for a small piece of interest.

WHY R MAY BE HARD FOR SPSS/ MINITAB USERS

WHY R MAY BE HARD FOR SPSS/ MINITAB USERS

- The biggest hurdle imposed by R for SPSS/Minitab users is the lack of a Graphical User Interface.

WHY R MAY BE HARD FOR SPSS/ MINITAB USERS

- The biggest hurdle imposed by R for SPSS/Minitab users is the lack of a Graphical User Interface.
    - Negative: R does not have "drop-down" menus with all the types of statistical models/analyses that could be performed.

WHY R MAY BE HARD FOR SPSS/ MINITAB USERS

- The biggest hurdle imposed by R for SPSS/Minitab users is the lack of a Graphical User Interface.
    - Negative: R does not have “drop-down” menus with all the types of statistical models/analyses that could be performed.
    - Positive: Upon switching to R, you will be required to write code and, thus, become at least a novice programmer.

WHY R MAY BE HARD FOR SPSS/ MINITAB USERS

- The biggest hurdle imposed by R for SPSS/Minitab users is the lack of a Graphical User Interface.
    - Negative: R does not have “drop-down” menus with all the types of statistical models/analyses that could be performed.
    - Positive: Upon switching to R, you will be required to write code and, thus, become at least a novice programmer.
        - May seem like a harsh learning curve, but it’s worth it in the long run.

WHY R MAY BE HARD FOR SPSS/ MINITAB USERS

- The biggest hurdle imposed by R for SPSS/Minitab users is the lack of a Graphical User Interface.
    - Negative: R does not have "drop-down" menus with all the types of statistical models/analyses that could be performed.
    - Positive: Upon switching to R, you will be required to write code and, thus, become at least a novice programmer.
        - May seem like a harsh learning curve, but it's worth it in the long run.

- Common Benefits of SPSS/Minitab over R:

WHY R MAY BE HARD FOR SPSS/ MINITAB USERS

- The biggest hurdle imposed by R for SPSS/Minitab users is the lack of a Graphical User Interface.
    - Negative: R does not have "drop-down" menus with all the types of statistical models/analyses that could be performed.
    - Positive: Upon switching to R, you will be required to write code and, thus, become at least a novice programmer.
        - May seem like a harsh learning curve, but it's worth it in the long run.

- Common Benefits of SPSS/Minitab over R:
    - Data is depicted in a familiar spreadsheet-like visual; in R, it is not seen unless specifically asked for.

WHY R MAY BE HARD FOR SPSS/MINITAB USERS

- The biggest hurdle imposed by R for SPSS/Minitab users is the lack of a Graphical User Interface.
    - Negative: R does not have "drop-down" menus with all the types of statistical models/analyses that could be performed.
    - Positive: Upon switching to R, you will be required to write code and, thus, become at least a novice programmer.
        - May seem like a harsh learning curve, but it's worth it in the long run.

- Common Benefits of SPSS/Minitab over R:
    - Data is depicted in a familiar spreadsheet-like visual; in R, it is not seen unless specifically asked for.
        - Use the `View()` function as a substitute.

APPENDIX

EXTERNAL RESOURCES

EXTERNAL RESOURCES

- *R for Data Science* (Hadley Wickham & Garrett Grolemund)
    - "You'll learn how to get your data into R, get it into the most useful structure, transform it, visualize it and model it."
- *The Book of R* (Tilman M. Davies)
    - "*The Book of R* is a comprehensive, beginner-friendly guide to R…you'll find everything you need to begin using R effectively for statistical analysis."
- swirl (Nick Carchedi, Brian Caffo, Sean Kross, et al.)
    - "swirl teaches you R programming and data science interactively, at your own pace, and right in the R console!"
- RStatistics.net
    - "An educational resource for all things related to R language and its applications in advanced statistical computing and machine learning."
- *An Introduction to R* (Bill Venables & David Smith)
- Quick-R (Robert I. Kabacoff)

"Thank you!"

-CHRISTOPHER PETER MAKRIS