DAEN 500- DL2 – Data Analytics Fundamentals

Fall 2020 Final Examination Exercise Package

10/7 – 10/13/2020

Final Submission Deadline: NLT 11:59PM (EST). Tuesday, 13 Oct 2020

*Failure to submit will result in DAEN COURSE FAIL/NoCredit*

Name: Jim McMahon GMU G#: G00737403

Student Signature (Honor Certification): James S. McMahon

This exam is **OPEN BOOK/OPEN NOTES**. You may consult any of the course texts, and the various reference materials recommended in the syllabus. ***The exam of course IS NOT “Open Web”,*** especially in that you may NOT utilize expert “help” sites such as Stack Overflow, or other programming help or collaboration sites.

HONOR CODE CERTIFICATION

**Your signature above declares that you have followed the conditions of this exam, and that the work is yours alone**. **Specifically:**

This must be your own work, authored and completed by you. As stated earlier, this is an “open source exam” – allowing books, notes or courseware, as well as *general* expert advice gained PRIOR to exam. YOU MAY NOT, HOWEVER, SEED OR USE ANY ADVICE ON HOW TO SOLVE THE QUESTION OR ANY CODE WRITTEN BY ANY OTHER INDIVIDUAL. *Any violation will result in an immediate failure in the exam and for the course, as well as referral to the GMU Honor Committee for determination of any other appropriate disciplinary consequences.*

*NOTE: Your* ***submission*** *of any responses, files, programs, etc. in response to the DAEN500 final exam instructions, will also be your personal certification of your full compliance with the spirit and letter of the* ***GMU Honor Code*** *standards for take home and/or in-class exams.*

Additionally, you are restricted from discussing the substance of the questions on this exam with any other individual, until after you have submitted your final response for grading. The completed exam -- with your answers embedded in this docx document (add extra pages as necessary) should be submitted following instructions contained in the Final Exam Instructions BB site. If you have any trouble submitting and have extra parts of the answers you have trouble appending to this document, you may simply submit additional pages separately (***the exam submission site is set for multiple submissions, just in case***). Make certain all are submitted PRIOR TO THE DEADLINE!

 FINAL EXAM PROBLEMS

COMPLETE ALL & INSERT ANSWERS BELOW QUESTIONS

# Problem 1: Python Programming Problem (15 Points Total)

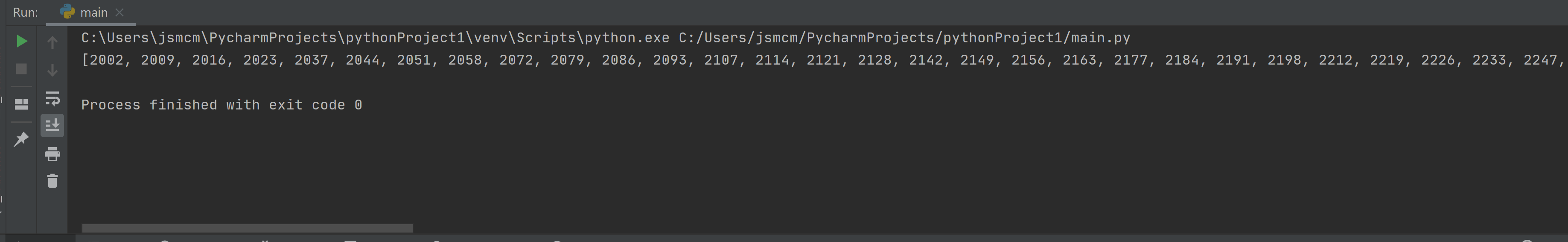
* **Design and implement a Python program that is based on the following requirements: a) program will *find* *all numbers -- within a specified range -- which are divisible by 7 but are not a multiple of 5*; and b) demonstrate the program works by running the program for the range: *numbers between 2000 and 3200*.**
* **INSERT (cut&paste) your Python code in space below and *then insert a screen shot in space below, showing your successful run and output.***

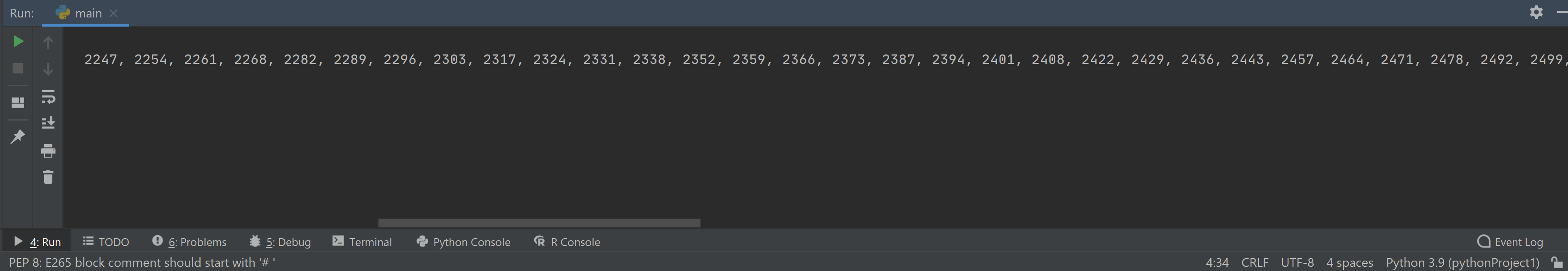
NOTE of alternative for help: To help test your code, you also may use a Python “programming window” found in the. **Zybooks Section 35 Additional Material**.

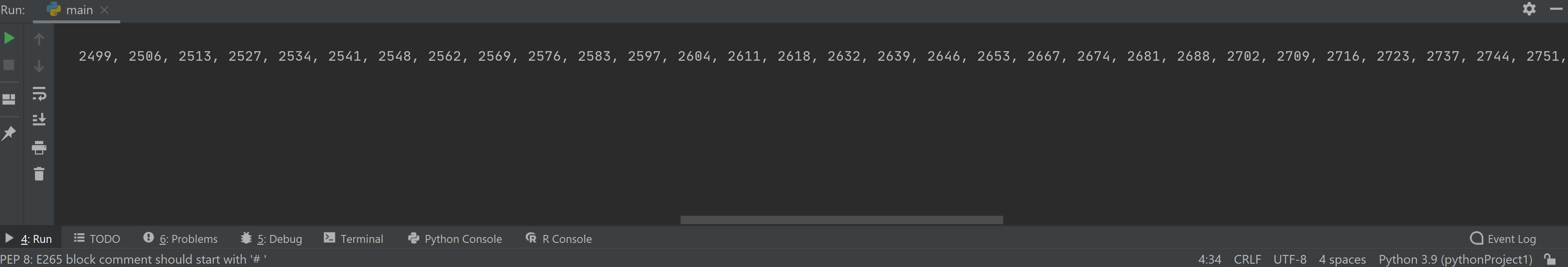
**(Note that the below can be edited, it just captured the dark background when copied)**

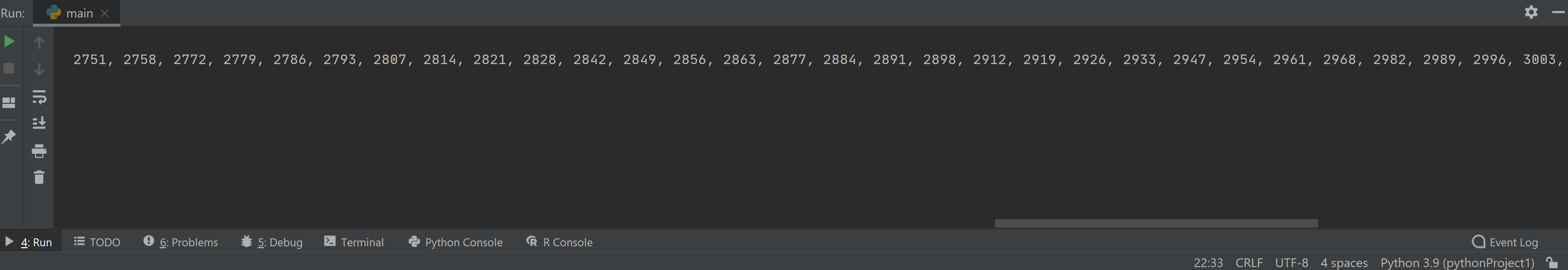
**(Output will be on next page)**

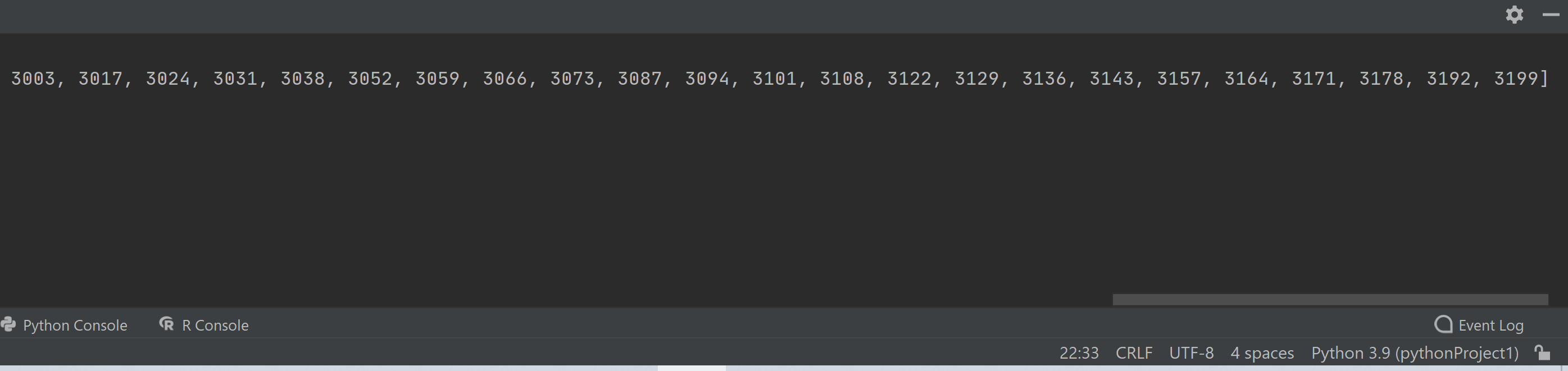
##Jim McMahon  
##DAEN 500 Final  
##Problem 1  
  
def find\_nums(num1, num2):  
 #First ID which number is lower  
 if num1 > num2:  
 clean\_num1 = num2 #clean\_num1 is low end  
 clean\_num2 = num1 #clean\_num2 is high end  
 elif num2 > num1:  
 clean\_num1 = num1 #clean\_num1 is low end  
 clean\_num2 = num2 #clean\_num2 is high end  
 else:  
 clean\_num1 = num1 #for when nums are equal  
 clean\_num2 = num2 #for when nums are equal  
  
 #Now find nums div. by 7 and but not mult. of 5  
 solution\_nums = [] # for storing nums that fit criteria  
 if clean\_num1 == clean\_num2:  
 if clean\_num1 % 5 != 0 and clean\_num1 % 7 == 0:  
 solution\_nums.append(clean\_num1)  
 return solution\_nums  
 elif clean\_num1 != clean\_num2:  
 myRange = range(clean\_num1, clean\_num2 + 1)  
 for item in myRange:  
 if item % 5 !=0 and item % 7 == 0:  
 solution\_nums.append(item)  
 return solution\_nums  
 else:  
 print('ERROR')  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 answer = find\_nums(2000, 3200)  
 print(answer)











**Note that the problem did not say to delivery the *count* of numbers that fit the criteria nor did it ask for the solution to be formatted, so I left the answer like this to fit the problem statement. I showed the overlapping number on each screenshot to show that these are cut and paste from the same line.**

# Problem 2: Python Programming Problem

# (15 Points Total)

* **Design and implement a Python program that is based on the following requirements:**

**a) define a class which has *at least two* methods**

* + **Method 1 – getString: to get a string from console input; and,**
  + **Method 2 - printString: to print the string in upper case.**

**b) *demonstrate code works using three different test input strings***

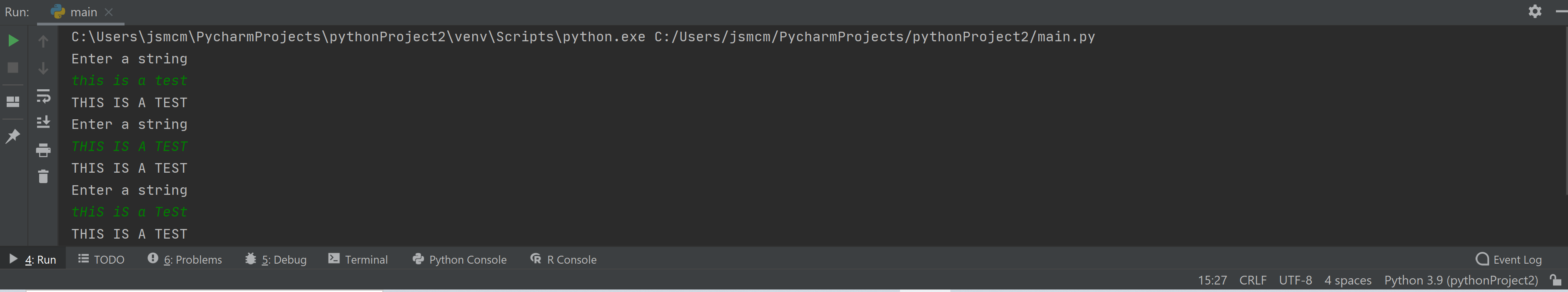
* ***INSERT* *code below* and *INSERT* a screen shot of the program and successfully run output that *includes test input for input strings (test strings must include (a) all upper case, (b) all lower case, and (c) mix of upper and lower case).***

**(Note that the below can be edited, it just captured the dark background when copied)**

**(Output will be on next page)**

##Jim McMahon  
##DAEN 500 Final  
##Problem 2  
  
#this is the class that will manipulate the input string  
class StringManipulator:  
 def \_\_init\_\_(self, currString = ''):  
 self.currString = currString  
  
 #method for getting user string input  
 def getString(self):  
 print('Enter a string')  
 user\_input = input()  
 myString = str(user\_input)  
 self.currString = myString  
  
 #method for making input all caps  
 def printString(self):  
 self.currString = self.currString.upper()  
 print(self.currString)  
  
#methods are repeated below to test all cases mentioned in problem statement  
if \_\_name\_\_ == '\_\_main\_\_':  
 newManipulator = StringManipulator()  
 newManipulator.getString()  
 newManipulator.printString()  
 newManipulator.getString()  
 newManipulator.printString()  
 newManipulator.getString()  
 newManipulator.printString()

**Screenshot of output below, all three test cases performed back to back:**





# Problem 3: R Programming Problem

# (20 Points Total)

* **Perform the following problems using R:**
  + Create a vector of courses (e.g., MATH 101) you have taken previously. Make sure you have at least 8 courses. Name the vector myCourses
  + Get the length of the vector myCourses
  + Get the first two courses from myCourses
  + Get the 3rd and 4th courses from myCourses
  + Sort myCourses using a method
  + Sort myCourse in the reverse direction
* *INSERT* *code below* and *INSERT* a screen shot of the program and successfully run output.

**The below is copied and pasted from the R Console of my IDE. I will also take a screenshot of what it looks like and paste that on the next page so that the formatting looks nicer. Required vector manipulations are completed in the order they were stated in the problem.**

**> myCourses <- c("FNAN 303", "BUS 303", "BUS 200", "HIST 100", "MIS 303", "MKTG 303", "BUS 310", "FNAN 311")**

**> length(myCourses)**

**[1] 8**

**> myCourses[1:2]**

**[1] "FNAN 303" "BUS 303"**

**> myCourses[3:4]**

**[1] "BUS 200" "HIST 100"**

**> sort(myCourses)**

**[1] "BUS 200" "BUS 303" "BUS 310"**

**[4] "FNAN 303" "FNAN 311" "HIST 100"**

**[7] "MIS 303" "MKTG 303"**

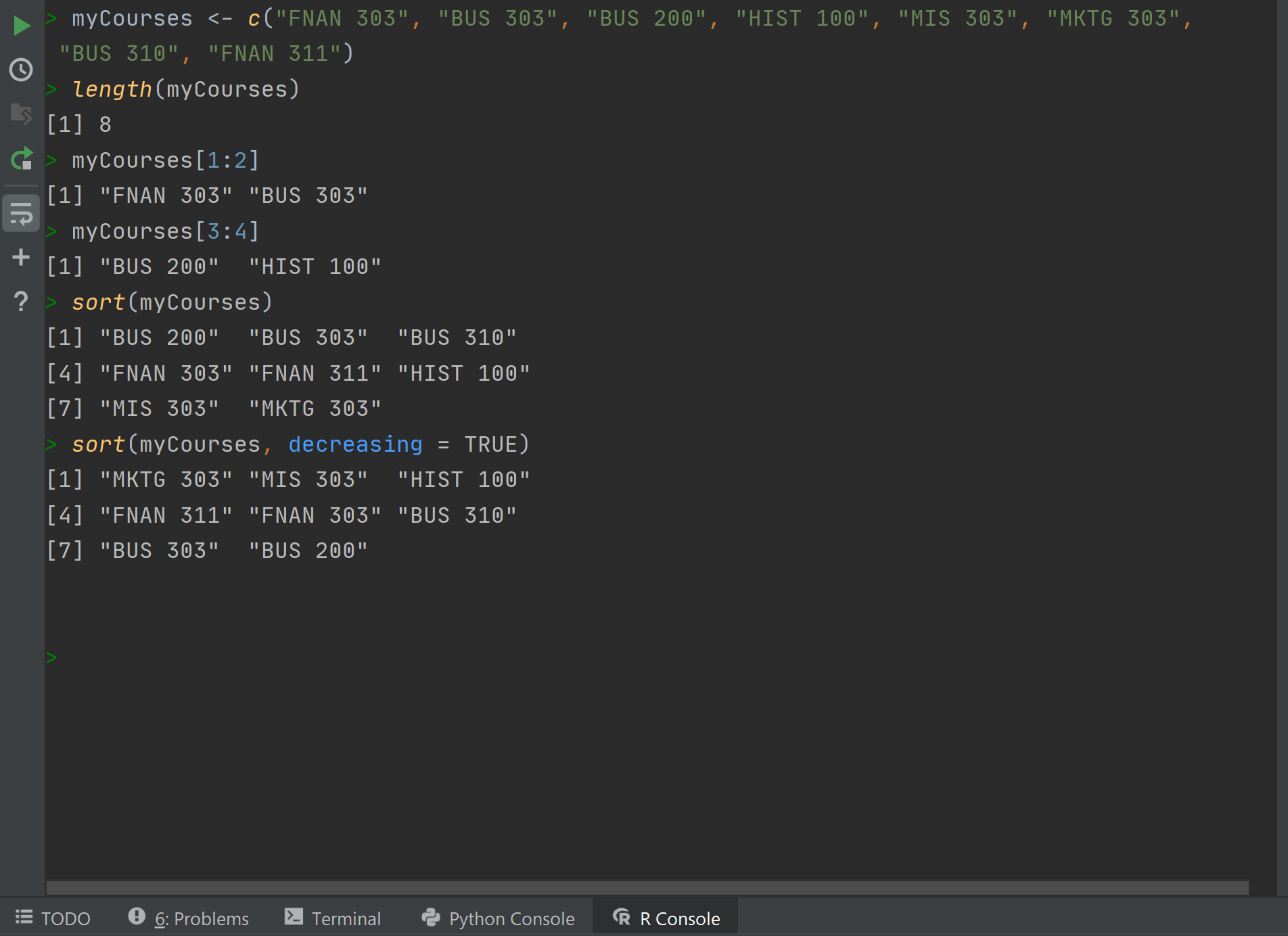
**> sort(myCourses, decreasing = TRUE)**

**[1] "MKTG 303" "MIS 303" "HIST 100"**

**[4] "FNAN 311" "FNAN 303" "BUS 310"**

**[7] "BUS 303" "BUS 200"**

**Same manipulations copied and pasted above as seen in my IDE:**





# Problem 4: Principal Component Analysis

# (25 points)

**Provide a description of the following:**

1. What is a component – Provide a description (5 points)
2. Principal Component Analysis – Provide a description.(5 points)
3. **Provide and explain an specific example of a Principal Component Analysis(15 points)**

**What is a component?**

In the context of Principal Component Analysis, components are relationships within a dataset that represent different amounts of variability. Generally, “principal components” refer to relationships that account for the highest levels of variability as they are more likely to provide useful information from a trends perspective. I suppose the term “component” could refer to any relationship regardless of variability, but “principal components” are what we care about here.

**Description of Principal Component Analysis**

The overall goal of Principal Component Analysis is to simplify large, complex situations into more manageable datasets, thus allowing analysts to make more consistently accurate predictions. By analyzing principal components (which as stated above are relationships within data that account for a large amount of variability relative to other relationships within the same data), we can figure out what factors are most important in the context of a specific analysis. Not surprisingly, there are many situations in the world with hundreds of varying factors at play, but by combining Principal Component Analysis with automated machine learning tools, it is possible to make more accurate predictions based on the factors that Principal Component Analysis deems most relevant.

**Provide a Specific Example of Principal Component Analysis**

The below is a completely hypothetical example intended to illustrate how Principal Component Analysis might work in practice. The factors selected are not based on real studies, they are simply used for the example.

Say we want to predict how many hours of video games an individual plays in a week. Every person has hundreds if not thousands of variables related to them. We might have a study available to us that has lists individual’s ages, eye color, hair color, sex, family size, zip code, income, the possibilities are endless. Principal Component Analysis seeks to narrow down this potentially massive list of variables to what matters most when it comes to predicting hours of video games played per week by identifying what variables have the highest levels of variability. Through this process, we might figure out that “age” and “sex” are the most important factors in making a prediction. Perhaps we would find that young men are most likely to play more than x hours of video games, but this is just for example purposes.

Based on this example, it is easy to see why this type of analysis might be useful to many institutions in real life. Businesses my want to predict who is more likely to buy their product, doctors may want to predict who is most likely to contract a certain type of disease, the government may want to identify who is most likely to be a potential threat to public safety. Principal Component Analysis allows us to leverage seemingly impossibly complex situations into useable information.

# Problem 5: Multiple vs. Logistic

# (30 points)

# Describe: What is difference between Multiple Regression and Logistic Regression? What circumstances might determine which to use? (10 points)

# Demonstrate: Using any data, and any tool set you’ve learned about, show differences (20 points)

# SUGGESTION: may be solved using RapidMiner, or other toolsets, BOTH TO ANALYZE AND TO VISUALIZE REGRESSION DIFFERENCES..

Step 1: Perform a quick search of the [UCIS public data archive](https://archive.ics.uci.edu/), a well-curated site which you already have seen as part of your introductory RapidMiner training.

Step 2: Pick a dataset you find interesting, input dataset into regression tools you’ve chosen.

Step 3: Run the dataset (*may be a significant subset, if the dataset is very large*) first. a Multiple Regression and then a Logistic Regression, .and use visualizations to demonstrate the conceptual answers you provided for 5.(a).

**a**. **What is the difference between Multiple Regression and Logistic Regression? What circumstances might determine which to use?**

The biggest difference is how outcome is measured. Multiple Regression attempts to predict the outcome of a situation based on x number of variables. That outcome is usually measured on a continuous scale (for example, grades on a scale of 0-100). Logistic Regression is similar, but the outcome is measured on a binary scale (for example, instead of 0-100, it might be pass/fail).

I would say circumstances for using each depend on how well the outcome fits each of these descriptions. If I want to predict if a team will win or lose, I think Logistic Regression may be a good fit. If I want to predict how many points a team will score on a given Sunday, Multiple Regression seems to be a better fit. There are ways to make each work in many different situations (especially with Multiple Regression) but it is really up to the analyst to decide what works best on a case by case basis.

**b. Demonstrate: Using any data, and any toolset you have learned about, show differences.**

**Step 1: Perform a quick search of the UCIS public data archive, a well-curated site which you have already seen as part of your introductory RapidMiner training.**

I selected a dataset of early stage diabetes case predictors. Per the author’s request on UCIS, I am included the citation they generated:

Islam, MM Faniqul, et al. 'Likelihood prediction of diabetes at early stage using data mining techniques.' Computer Vision and Machine Intelligence in Medical Image Analysis. Springer, Singapore, 2020. 113-125.

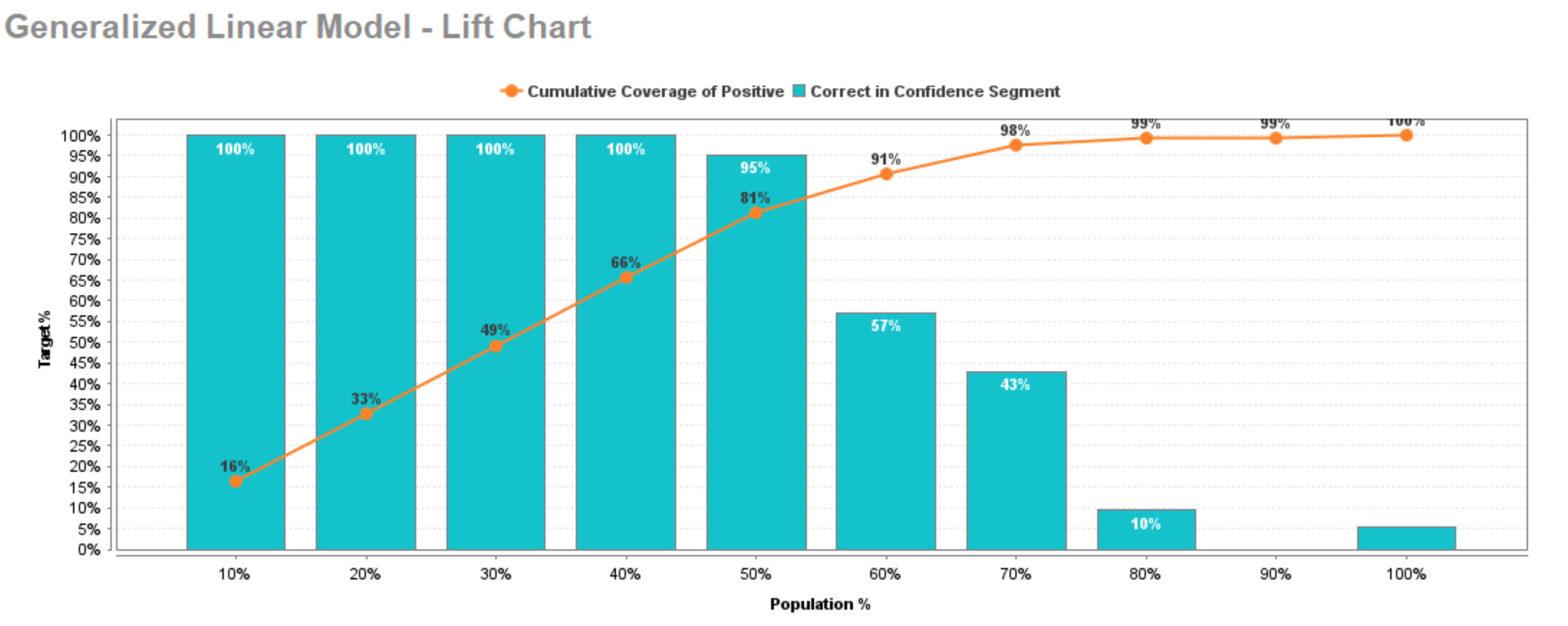
**Step 2: Pick a dataset you find interesting, insert dataset into regression tools you’ve chosen**

I am using RapidMiner to analyze the data.

**Step 3: Run the dataset (may be a significant subset if the dataset is very large). First, a multiple regression. Second, a logistic regression. Use visualizations to demonstrate the conceptual answers you provided for 5(a).**

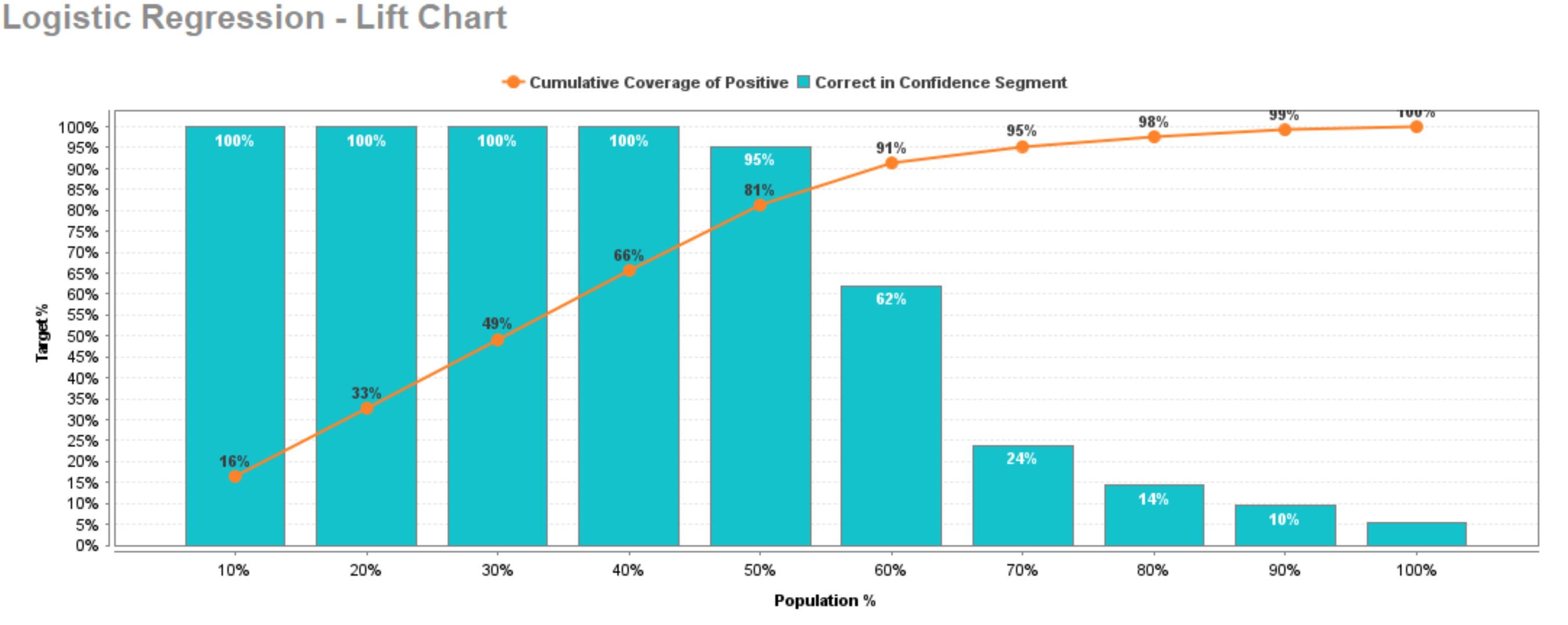
I am selecting what I think were the most useful visualizations I found in RapidMiner. Note that I have several variables available for analysis (age, irritability, gender, various symptoms, etc.) and I am seeking to accurately predict the “class” outcome. “Class” refers to the medical positive or negative. As I would have predicted, the logistic regression model was the best fit for predicting the binary positive or negative outcome. Visualizations shown below.

Multiple Regression visualization tool:



Per RapidMiner, the linear regression model (shown above) had a 9.4% classification error rate. This is higher than the logistic regression which will be shown on the next page.

Logistic Regression visualization tool:



The logistic regression method is reflected in the above chart. As we can see based on the chart, the logistic regression model performs slightly better than the linear regression model. Per RapidMiner, the logistics regression model only had an 8.7% error rate vs. the 9.4% error rate of the linear model. This fits my expectation as I predicted the logistic model to perform better in a situation with binary outcomes (i.e. positive class vs. negative class)